ACTIVE CAPACITOR VOLTAGE STABILISATION IN A MEDIUM-VOLTAGE FLYING-CAPACITOR MULTILEVEL ACTIVE FILTER

CHRISTINE HENRIËTTE HANSMANN

Thesis presented in partial fulfilment of the requirements of the degree Master of Science in Engineering at the University of Stellenbosch

SUPERVISOR: PROF. H. DU T. MOUTON

April 2005

Declarations

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature Date

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

Summary

A switching state substitution must be developed that will make use of both single-phase redundancies and three-phase redundancies in the flying-capacitor topology. Losses should be taken into consideration and the algorithm must be designed for implementation on the existing PEC33 system, with on-board DSP (TMS320VC33) and FPGA (EP1K50QC208). The specific power-electronics application is a medium-voltage active filter.

Existing capacitor voltage stabilisation schemes are investigated and a capacitor-voltage based algorithm is developed that is investigated in parallel with the Donzel and Bornard algorithm. Detailed simulation models are built for the evaluation of both existing and the proposed algorithm. Three-phase control is also evaluated.

Timing analysis of the proposed algorithm shows that a DSP-only implementation of the proposed capacitor-based solution is not feasible. Detail design of the digital controller hereof is implemented in VHDL. Finally, a four-cell controller is fitted into the FPGA. A scalable hardware sorting architecture is utilised.

Opsomming

'n Skakeltoestand toestandsvervangings algoritme word vereis wat van beide die enkelfase en die drie-fase oortollige toestandskombinasies in the swewende-kapasitor-topologie gebruik maak. Verliese moet in ag geneem word en die algoritme moet ontwerp word vir implementasie op die huidige PEC33 stelsel, met DSP (TMS320VC33) en FPGA (EP1K50QC208) ingesluit. Die spesifieke drywings-elektroniese toepassing is 'n medium-spanning aktiewe drywing filter.

Bestaande stabilisasie algoritmes word ondersoek en 'n kapasitor-gebaseerde stabilisasie algoritme word afgelei wat gesamentlik met die Donzel en Bornard algoritme ondersoek word. Gedetailleerde simulasie modelle word opgebou vir evaluasie doeleindes. Ook drie-fase beheerders word ondersoek.

'n Uitvoertyd analise van die algoritme wys uit dat 'n enkele DSP beheerder onvoldoende sal wees in teme van uitvoertyd. Detail ontwerp van die FPGA-gebaseerde ko-prosesseerder word in VHDL gedoen vir die voorgestelde kapasitor-gebaseerde algortime. 'n Vier-sel beheerder word uiteindelik in die FPGA gepas. ’n Uitbreibare hardeware sorteringsalgoritme word gebruik in hierdie oplossing.

Acknowledgements

Thank you, to my family, for your loving support.

Thank you, to my friends and colleagues, for experiences shared.

In addition, I would like to thank the NRF for their financial support.

Finally, I would like to thank Prof. Toit Mouton for his guidance throughout the project, and through Eskom, for financial support.

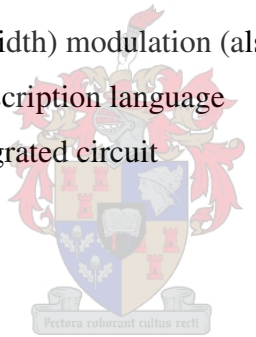
Contents

List of Figures

List of Tables

Glossary

ADC	:	Analogue-to-digital converter
ASIC	:	Application-specific integrated circuit
DSP	:	Digital signal processor
FPGA	:	Field-programmable gate array
IGBT	:	Insulated-gate bipolar transistor
LPM	:	Library of parameterised modules
LSB	:	Least-significant bit
MSB	:	Most-significant bit
PWM	:	Pulsewidth modulation
SVM	:	Space-vector (pulsewidth) modulation (also: SVPWM)
VHDL	:	VHSIC Hardware description language
VHSIC	:	Very-high-speed integrated circuit



Chapter 1
Introduction

1.1 A MEDIUM-VOLTAGE ACTIVE POWER FILTER REALISATION

Traditionally, various strategies are implemented to suppress harmonic related problems and increase line loadability in power systems. Solutions such as static VAR compensators (SVCs), synchronous condensers, static condensers (STATCOMs), series capacitors and shunt L-C filters have been implemented [1, 2, 3].

An active filter is the alternative solution, and overcomes many problems associated with the more conventional remedies. The choice between passive and active compensation strategies, however, is usually based on cost considerations. Various statements regarding cost-efficiency can be found [4, 5].

The traditional boundaries of cost-efficiency may eventually be shifted by the use of multilevel converter topologies (and specifically the flying-capacitor topology) as active filters, since flying-capacitor converters

- are principally high-power converter configurations [6]
- are comparatively high-bandwidth topologies [7]
- have low switching losses (compared to conventional converters, with similar bandwidth) [8]
- can eliminate the need for step-up transformers (together with its associated costs, losses and bulk) in certain applications, such as shunt active filtering [9].

A critical aspect of multilevel converter control, however, is the balancing of capacitor voltages [10]. This study will investigate the concept of a medium voltage active filter realised by means of a flying-capacitor multilevel converter, with special focus on capacitor voltage stabilisation in the topology.

1.2 AN OVERVIEW OF MULTILEVEL CONVERTERS

High-power applications frequently require series or parallel connection of semiconductors. Often, it is found that a single IGBT would not be able to withstand a required voltage or current rating. Series connection overcomes the voltage limitation in higher-voltage applications. The series connection of semiconductors and the series connection of converters evolved into the range of multilevel converters, as they are known today.

The main multilevel configurations are

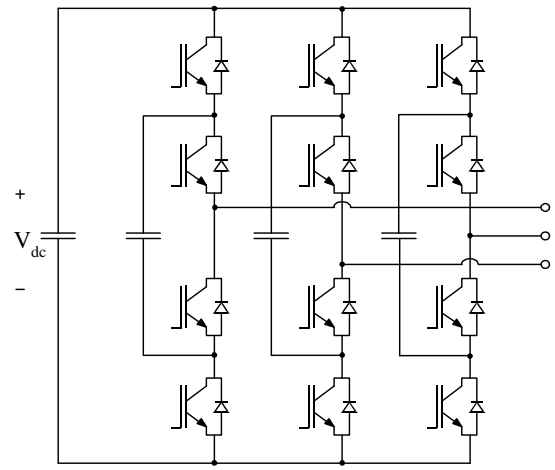
- the *flying-capacitor* topology (also the *multicell*, *imbricated cells* or *capacitor-clamped* topology) [11]
- the *diode-clamped* topology (also the *neutral-point-clamped* topology) [12]
- the *cascaded* topology [13]
- the *series-stacked* topology [14].

Many hybrid or derived topologies can be found, such as [15], [16] and [17].

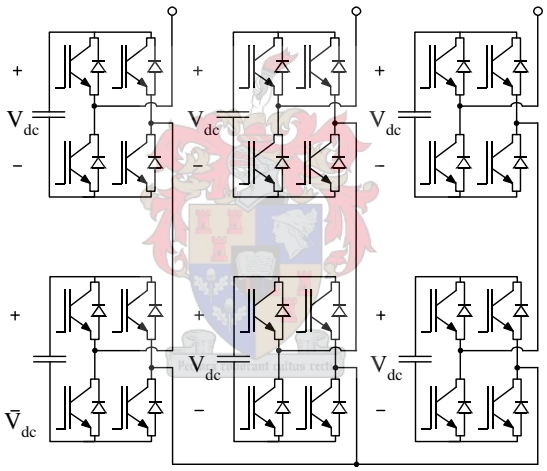
Some of the advantages and drawbacks of the multilevel topologies are evident from the diagrammatic representation in Figure 1.1. For instance, flying-capacitor converters require many capacitors; diode-clamped converters require several diodes. Many voltage sources are required by the cascaded converter in applications where real power transfer is involved. Topologies such as the flying-capacitor, diode-clamped and cascaded configurations are suited to applications where the use of a transformer is optional. Conversely, the series-stacked topology finds its niche in applications where a transformer is functionally required [10].

The increased number of voltage steps in the output voltage is characteristic of this family of converters, hence *multilevel* converters. Generally, an *n*-level inverter has a phase voltage output – relative to the inverter centre point – with *n-1* steps (or levels). The series-stacked converter is the exception to the rule: here the term *n*-level refers to the stacking of *n* converters. In addition, the series-stacked converter has an improved quality output with interleaved switching only [10]. The number of output levels of the flying-capacitor-, cascaded- and diode-clamped converters in Figure 1.1 is 3, 5 and 3, respectively.

(i) flying-capacitor



(ii) cascaded



(iii) diode-clamped

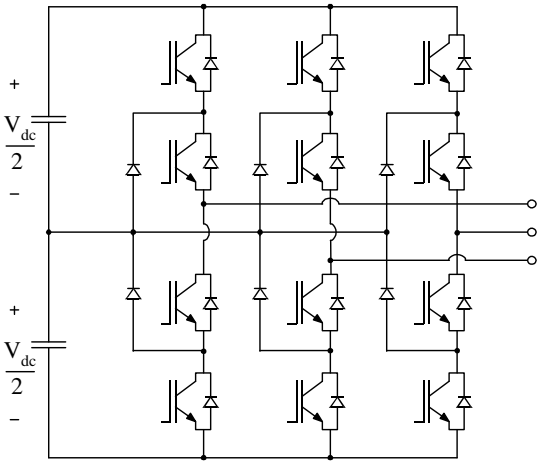


Figure 1.1 *Some multilevel converter topologies [11, 12, 13]*

Multilevel converters allow voltages to be processed that are not within reach of any single switching device [18]. Alternatively, it allows for the use of semiconductors that, combined, have characteristics superior to that of higher-voltage semiconductors [6, 18, 19]. In addition, it improves the output, leading to smaller filters and increased bandwidth [18].

1.3 THE ACTIVE POWER FILTER APPLICATION

Four main active filtering compensation topologies can be considered – the regular series topology, the regular shunt topology, the hybrid series topology and the hybrid shunt topology [4].

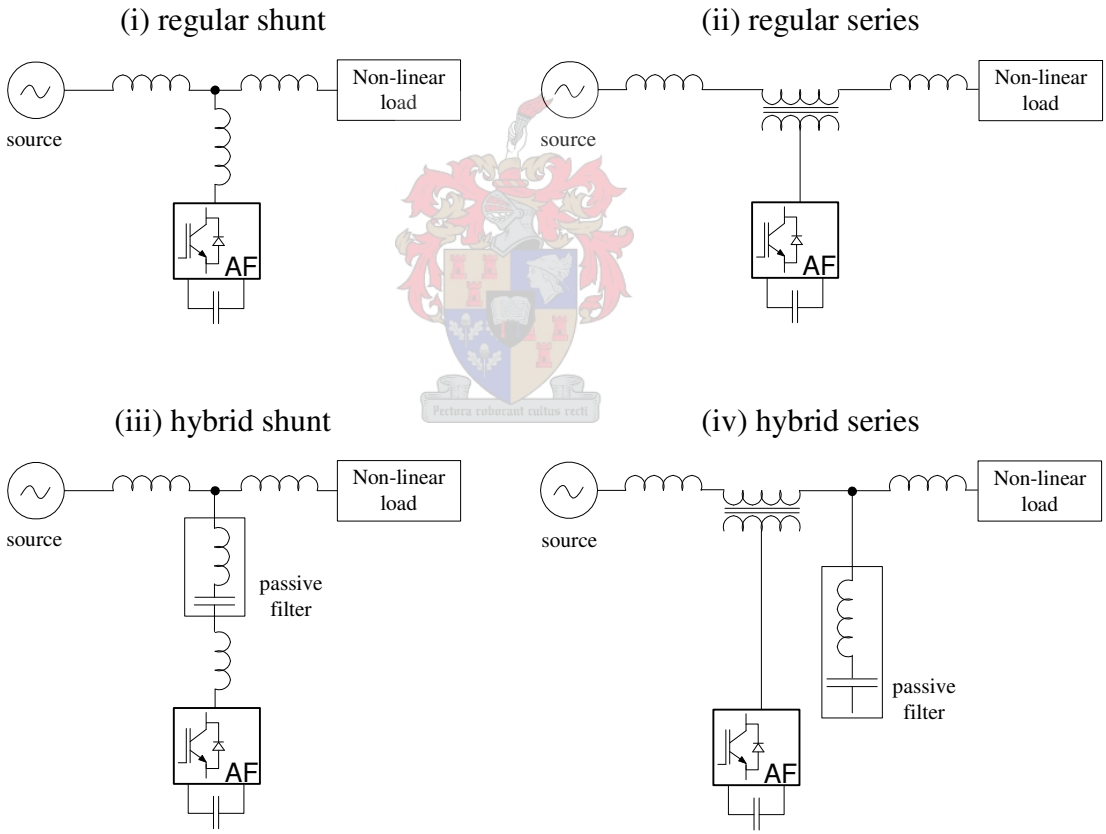


Figure 1.2 *Four main active filtering topologies [2, 4, 20]*

Shunt active filters are suited to compensation of harmonic current-source type of non-linear loads; conversely, series active filters are suited to compensation of harmonic voltage-

source types of loads [21]. In general, the hybrid active filters are proposed in cases where a reduced kVA-rating converter is desirable [2]. The investigation and evaluation of the best possible configuration is not the aim of this thesis, however. A conventional active filter as in Figure 1.2 (a) will be considered in this thesis as an application of the flying-capacitor topology. Various publications on multilevel and flying-capacitor topologies as active filters can be found in existing literature. Multilevel converter realisations of active filters can be found in [9], [17] and [22]. Active filters utilising the flying-capacitor multilevel topology are discussed in [8], [23] and [24]. Furthermore, the idea of a medium voltage active filter is not novel, as active filters for the damping of harmonic propagation in distribution systems are discussed in [25], [26] and [27].

The p-q algorithm, first proposed by Akagi *et al.* in [28] and generalised by Peng *et al.* in [29], will be utilised. The shunt compensator current, $\boldsymbol{i}_C^*$, is calculated as

$$\boldsymbol{i}_C^* = \frac{p_C^* \boldsymbol{v}_s}{\boldsymbol{v}_s \cdot \boldsymbol{v}_s} + \frac{\boldsymbol{q}_C^* \times \boldsymbol{v}_s}{\boldsymbol{v}_s \cdot \boldsymbol{v}_s}, \tag{1.1}$$

where

$$\begin{aligned} p_C^* &= \tilde{p}_L = \boldsymbol{v}_s \cdot \boldsymbol{i}_L \\ \boldsymbol{q}_C^* &= \boldsymbol{q}_L = \boldsymbol{v}_s \times \boldsymbol{i}_L \end{aligned} \tag{1.2}$$

since the load reactive power and harmonics are to be eliminated, and the compensator should supply the ripple active power ($\tilde{p}$) and the instantaneous reactive power (q) required by the load. The emf-voltage vector $\boldsymbol{v}_s$ and the load current vector $\boldsymbol{i}_L$ is given by

$$\boldsymbol{v}_s = \begin{bmatrix} v_{s,a} \\ v_{s,b} \\ v_{s,c} \end{bmatrix}, \quad \boldsymbol{i}_L = \begin{bmatrix} i_{L,a} \\ i_{L,b} \\ i_{L,c} \end{bmatrix}. \tag{1.3}$$

The active filtering current reference will be obtained through the use of equation (1.1) in all three-phase simulation results shown in this thesis.

1.4 THESIS OUTLINE

Existing stabilisation algorithms will be investigated and an algorithm developed to overcome the shortcomings of the existing methods. The algorithm is required to be a switching state substitution method; it should make use of three-phase redundancies and take losses into consideration; it must be designed for implementation on the existing PEC33 system, with on-board DSP (TMS320VC33) and FPGA (EP1K50QC208). A specification central to this thesis is that the controller must take switching losses into account. Practically, this means that the number of switch state transitions should be monitored.

In Chapter 2, the definitions of concepts relating to capacitor balance and imbalance in the topology is set out and an overview of existing capacitor voltage stabilisation schemes is given.

The modelling of these strategies is not straightforward; consequently, the details hereof are given in Chapter 3. The shortcomings of these algorithms are identified and an alternative compensation strategy is proposed.

In Chapter 4, a timing analysis of the proposed algorithm is performed. Detail design of the digital controller follows.

In Chapter 5, results of both the Simplorer simulation models and the simulated VHDL controller architecture are given. Finally the conclusions, a summary of contributions and recommendations for future research are presented in Chapter 6.

Chapter 2
Imbalance and Existing Voltage Regulation Schemes

2.1 INTRODUCTION

The concepts of balance, imbalance and natural balancing in the flying-capacitor topology are central to the thesis. Consequently, these ideas will be defined in the initial stages of this chapter. In addition, an overview of some published compensation strategies will be given and its relevance to the state substitution and the space-vector modulation methods will be demonstrated. The existing state substitution methods are evaluated and a new state substitution control method is proposed in an effort to overcome the limitations thereof.

2.2 IMBALANCE IN THE FLYING-CAPACITOR TOPOLOGY

A balanced capacitor voltage is defined as a capacitor voltage that assumes its assigned value at steady-state [30]. Natural balancing refers to the case where the specific modulation strategy ensures balanced capacitor voltages, without any specific capacitor voltage control.

Capacitor voltage balance is of consequence in all multilevel converters [10]. Various studies on the conditions for natural balancing in multilevel converters have been done. For instance, studies on the natural balancing conditions in series-stacked power quality conditioners [10], in neutral-point-clamped converters with POD PWM methods [31] and in flying-capacitor converters with carrier-based PWM [30] can be found. Natural balancing or self-balancing of capacitor voltages implies a simplified control algorithm [32]. RLC filters tuned at the switching frequency ('balance boosters') to assist balancing with no additional control complexity, have also been investigated [33].

The voltage distribution that must be imposed [11] to ensure capacitor voltage balance in the flying-capacitor multilevel topology, is given by

$$V_{C_k} = k\,\frac{V_{dc}}{n} \qquad k = 1,\ldots,n \qquad (2.1)$$

where V_{C_k} is the voltage of the k^{th} capacitor and n the number of commutation cells, as in Figure 2.1.

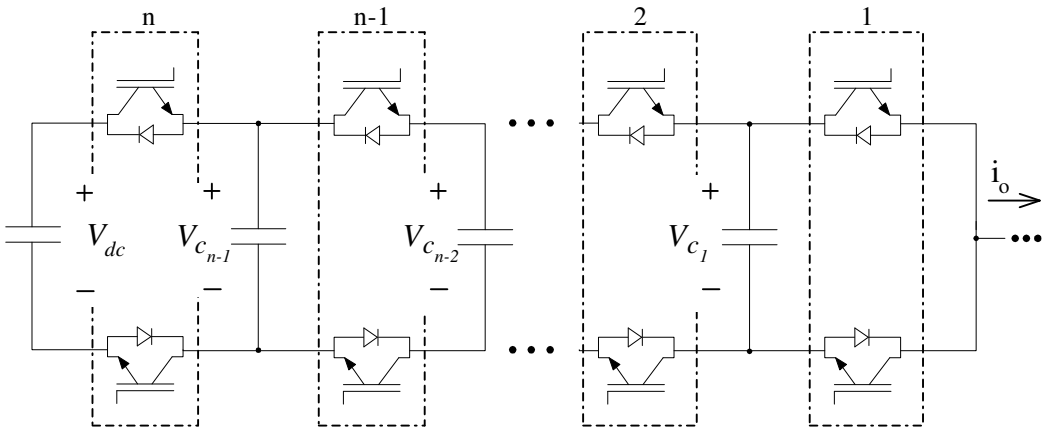


Figure 2.1 *Commutation cell definitions in a flying-capacitor phase-leg*

A cell voltage can be defined [34, 35] as

$$V_{cell_k} = V_{c_k} - V_{c_{k-1}} \qquad k = 1,\ldots,n \qquad (2.2)$$

where

$$V_{c_0} = 0\,, \quad V_{c_n} = E_d\,.$$

No physical meaning regarding the concept of a cell voltage is specified in either [34] or [35]; however, it can be seen that the k^{th} cell voltage is the voltage over a semiconductor in the k^{th} commutation cell, as indicated in Figure 2.1. From equation (2.2), cell voltages under balanced conditions can be derived as

$$\begin{aligned} V_{cell_k} &= k\,\frac{V_{dc}}{n} - (k-1)\,\frac{V_{dc}}{n} \\ &= \frac{V_{dc}}{n} \end{aligned} \qquad (2.3)$$

Thus, by adhering to the balanced voltage distribution of equation (2.1) during both transient and steady-state conditions, equal semiconductor blocking voltages can be ensured. In addition, accurate converter output voltages are guaranteed – see equation (2.8).

A detailed harmonic and steady-state analysis of the flying-capacitor topology was done in [30] to predict imbalance in the flying-capacitor topology. The theory, however, is derived from a carrier-based modulation viewpoint and can therefore only be used with carrier-based modulation strategies or carrier-based equivalent representations.

In [23], it is stated that active capacitor voltage stabilisation is required for active filtering purposes. The specific application is a soft switching, hysteresis current controlled shunt active filter. Consequently, this statement cannot be extended to the space-vector modulation approach of the current study. The motivation for a space-vector based current controller in a high-power application is set out in subsection 2.6.2.

The use of space-vector modulation in the flying-capacitor topology implies a degree of active control – i.e. control not only concerned with output current or voltage regulation – a switching state must be selected from the available three-phase redundant state combinations. In fact, a space-vector implementation as in subsection 2.6.3 most probably cannot have a natural balancing property, as no natural order for the selection of switching states exists. The optimisation flexibility of space-vector modulation is often quoted as an advantage of the specific modulation strategy [36, 37]. However, it may be more realistic to say that space vector modulation, as in subsection 2.6.3, *requires* optimisation – i.e. active control – to make meaningful choices within the available degrees of freedom [35, 38] offered by the flying-capacitor topology. It therefore makes sense to implement a control algorithm for the active stabilisation of capacitor voltages in the flying-capacitor multilevel converter whenever space-vector modulation is used.

2.3 STATE SUBSTITUTION COMPENSATION METHODS

Various redundant switching states can be identified on a per-phase basis in the flying-capacitor topology. Several sets of switching states with equivalent output voltages but dissimilar capacitor charging characteristics exist. This property will be utilised to control the capacitor voltages as is required and will be referred to as *state substitution*. The state substitution principle is described graphically in Figure 2.2. The switching functions s_5, s_4, s_3, s_2 and s_1 represent the on-off state of the top switching elements, as indicated in Figure 2.2.

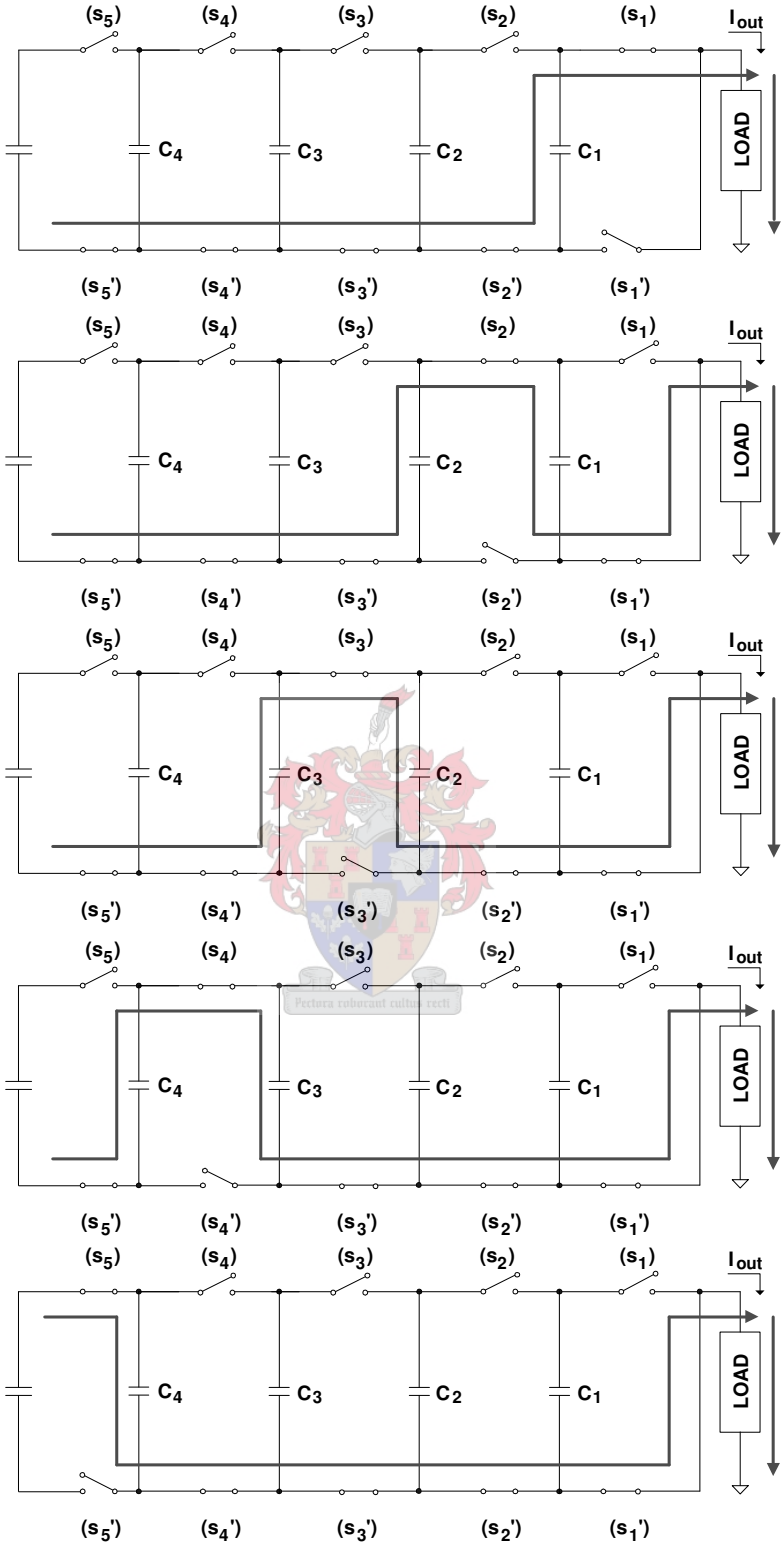


Figure 2.2 *Illustration of the state substitution principle: the configurations shown above form the set of redundant states for an output voltage of level one, in a five-cell converter phase-leg of an active filtering application. The arrows indicate the positive direction of current flow through the converter for a positively defined output load current. The charge/ discharge effects of each switching state can be identified by observing the direction of current flow through the various capacitors.*

For instance, the variable s_5 is set to '1', to indicate that the top leftmost switch is on; conversely, the variable s_5 is set to '0', to indicate that the top leftmost switch is off. The switching functions s_5', s_4', s_3', s_2' and s_1' represent the on-off state of the bottom switching elements, as shown in Figure 2.2. Generally, the s_5', s_4', s_3', s_2' and s_1' switching functions are controlled to be the Boolean inverse of the s_5, s_4, s_3, s_2 and s_1 switching functions, respectively. Some exceptions to this approach occur during blanking times, for example. A standard manner in which pulsewidth modulated switching states are identified, however, is by means of switching functions [21, 34, 39]. For instance, the five configurations shown in Figure 2.2 can be represented, from top to bottom, by the switching states '0 0 0 0 1', '0 0 0 1 0', '0 0 1 0 0', '0 1 0 0 0' and '1 0 0 0 0'. A complete listing of all available switching states and the associated charge/discharge characteristics and output voltage values is specified in Table 2.1 for a five-cell topology. State substitution, as implemented in this thesis, will involve the choice of a specific switching state based on capacitor charge/discharge characteristics, the required output voltage level and control of switching losses – refer to the specification in subsection 1.4.

State substitution allows capacitor voltage stabilisation to be performed independently of output current or voltage regulation. The capacitor voltages can be controlled without modifying the output voltage and therefore the harmonic properties of the output waveforms are preserved [34].

Although never classified as such in existing literature, state substitution methods can be either cell-voltage based or capacitor-voltage based. A cell-voltage based controller can be interpreted as a controller that aims to stabilise cell voltages, or the voltages over the switching elements, at their specified reference value – refer to equation (2.15); a capacitor-voltage based controller can be interpreted as a controller that aims to maintain the capacitor voltages at their respective reference values – refer to equation (2.13).

The state substitution methods that were found in existing literature will be discussed and classified in the following subsections.

switching state					charging characteristic (for positively defined load current)				output voltage	output level
s_5	s_4	s_3	s_2	s_1	c_4	c_3	c_2	c_1		
0	0	0	0	0	*	*	*	*	$-V_{dc}/2$	0
0	0	0	0	1	*	*	*	discharge	$-3V_{dc}/10$	1
0	0	0	1	0	*	*	discharge	charge	$-3V_{dc}/10$	1
0	0	0	1	1	*	*	discharge	*	$-V_{dc}/10$	2
0	0	1	0	0	*	discharge	charge	*	$-3V_{dc}/10$	1
0	0	1	0	1	*	discharge	charge	discharge	$-V_{dc}/10$	2
0	0	1	1	0	*	discharge	*	charge	$-V_{dc}/10$	2
0	0	1	1	1	*	discharge	*	*	$V_{dc}/10$	3
0	1	0	0	0	discharge	charge	*	*	$-3V_{dc}/10$	1
0	1	0	0	1	discharge	charge	*	discharge	$-V_{dc}/10$	2
0	1	0	1	0	discharge	charge	discharge	charge	$-V_{dc}/10$	2
0	1	0	1	1	discharge	charge	discharge	*	$V_{dc}/10$	3
0	1	1	0	0	discharge	*	charge	*	$-V_{dc}/10$	2
0	1	1	0	1	discharge	*	charge	discharge	$V_{dc}/10$	3
0	1	1	1	0	discharge	*	*	charge	$V_{dc}/10$	3
0	1	1	1	1	discharge	*	*	*	$3V_{dc}/10$	4
1	0	0	0	0	charge	*	*	*	$-3V_{dc}/10$	1
1	0	0	0	1	charge	*	*	discharge	$-V_{dc}/10$	2
1	0	0	1	0	charge	*	discharge	charge	$-V_{dc}/10$	2
1	0	0	1	1	charge	*	discharge	*	$V_{dc}/10$	3
1	0	1	0	0	charge	discharge	charge	*	$-V_{dc}/10$	2
1	0	1	0	1	charge	discharge	charge	discharge	$V_{dc}/10$	3
1	0	1	1	0	charge	discharge	*	charge	$V_{dc}/10$	3
1	0	1	1	1	charge	discharge	*	*	$3V_{dc}/10$	4
1	1	0	0	0	*	discharge	*	*	$-V_{dc}/10$	2
1	1	0	0	1	*	charge	*	discharge	$V_{dc}/10$	3
1	1	0	1	0	*	charge	discharge	charge	$V_{dc}/10$	3
1	1	0	1	1	*	charge	discharge	*	$3V_{dc}/10$	4
1	1	1	0	0	*	*	charge	*	$V_{dc}/10$	3
1	1	1	0	1	*	*	charge	discharge	$3V_{dc}/10$	4
1	1	1	1	0	*	*	*	charge	$3V_{dc}/10$	4
1	1	1	1	1	*	*	*	*	$V_{dc}/2$	5

where * indicates neither a charge nor a discharge operation

Table 2.1 *Switching states and output voltages in a five-level flying-capacitor converter*

2.3.1 THE ALGORITHM OF DONZEL AND BORNARD

A cell-voltage based algorithm for imbalance compensation is proposed by Donzel and Bornard in [34]. It is shown that optimal control of capacitor voltages can be achieved by maximising a dot product between a vector of cell voltage errors and a vector of switching functions. The following derivation [34] validates the basis of the proposed algorithm:

The capacitor current in the i^{th} capacitor is given by

$$I_i = \delta_i I_s \tag{2.4}$$

where the differential switching function is calculated as

$$\delta_i = s_{i+1} - s_i . \tag{2.5}$$

The cell voltage value is determined from the difference in capacitor voltage values

$$V_{cell_i} = V_{c_i} - V_{c_{i-1}} \tag{2.6}$$

where

$$V_{c_0} = 0 , \quad V_{c_N} = E_d , \quad s_{N+1} = 0 .$$

The output voltage can be specified in terms of cell voltages

$$V_s = \sum_{i=1}^{N} s_i \cdot V_{cell_i} \tag{2.7}$$

or in terms of capacitor voltages

$$V_s = -\sum_{i=1}^{N} \delta_i \cdot V_{c_i} \tag{2.8}$$

Optimal control of capacitor voltages can be achieved by selecting a state in order to maximise the dot product between the desired displacement direction of the capacitor voltage, and the possible directions of evolution in the capacitor voltage space. The principle is illustrated in Figure 2.3. A dotted line represents the desired displacement direction of the capacitor voltage in state space. The direction of evolution for each switching state is indicated by an arrow.

The dot product to be maximised, is

$$J = \left(\boldsymbol{V}_{c_d} - \boldsymbol{V}_c\right)^T \cdot \boldsymbol{d}_i \tag{2.9}$$

where the direction of capacitor voltage evolution, for small variation in I_s, is represented by

$$\boldsymbol{d}_i = I_s \begin{bmatrix} \delta_{N-1} & \cdots & \delta_2 & \delta_1 \end{bmatrix}^T . \tag{2.10}$$

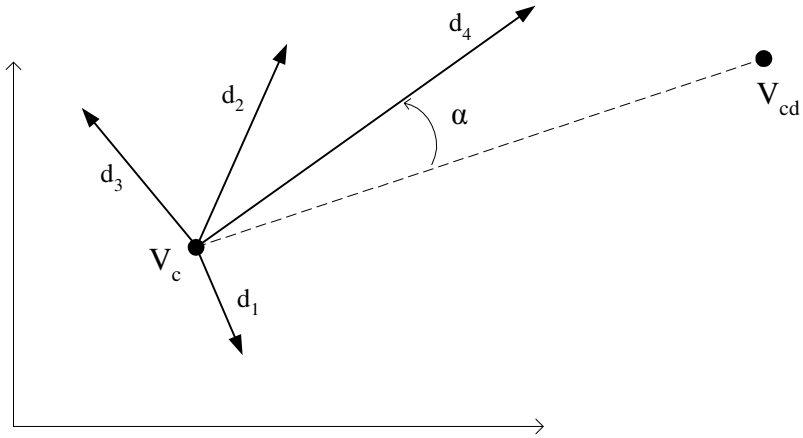


Figure 2.3 *Optimal control in 4-dimensional capacitor space domain for a 5-cell configuration: the state corresponding to the direction of evolution with greatest component alongside (**V**cd -**V**c) is selected [34]*

Given that the error voltages are defined as the difference between actual values and desired values

$$\boldsymbol{eV}_{cell} = \boldsymbol{V}_{cell} - \boldsymbol{V}_{cell_d} , \quad eV_s = V_s - V_{s_d} , \quad \boldsymbol{eV}_c = \boldsymbol{V}_c - \boldsymbol{V}_{c\,d} \tag{2.11}$$

and the differential switching functions and switching function vectors are specified as

$$\boldsymbol{\delta} = \begin{bmatrix} \delta_{N-1} & \cdots & \delta_2 & \delta_1 \end{bmatrix}^T, \quad \boldsymbol{S} = \begin{bmatrix} s_{N-1} & \cdots & s_2 & s_1 \end{bmatrix}^T \tag{2.12}$$

the dot product can be rewritten in terms of capacitor error voltages and differential switching functions,

$$J = -I_s \, \boldsymbol{eV}_c^T \cdot \boldsymbol{\delta} \tag{2.13}$$

or, in terms of the error in output voltage,

$$J = I_s \, eV_s \tag{2.14}$$

or, in terms of cell voltage errors and switching functions

$$J = I_s \, \boldsymbol{eV}_{cell}^T \cdot \boldsymbol{S} \tag{2.15}$$

Equation (2.15) forms the basis of the cell voltage control proposed in [34].

In the basic algorithm of Donzel and Bornard, no limits are placed on the number of commutations. The cell voltage errors are sorted in descending order and the corresponding first n switching functions are set to '1' for a positive output current (where n is the output voltage level). A negative output current implies that the cell voltage errors should be sorted in ascending order. The basic algorithm is similar to the strategy of Martins et al. that is described in the following subsection.

Donzel and Bornard also propose a modified algorithm in [34], whereby the number of concurrent commutations is limited to a specified parameter n_{free}. The absolute values of the cell voltage errors are sorted in descending order and a set A is formed by the n_{free} first elements; all other elements are placed in a set B. A subset A_p is defined and consists of all the positive elements in vector A; all other elements of set A is stored in subset A_n. Similarly, subset B_o contains all the elements corresponding to a switching function previously '1'; subset B_z contains all elements corresponding to a switching function previously '0'.

The subvectors A_p, A_n, B_o, B_z are sorted according to their corresponding $|V_{cell}|$ values. A vector C is formed by concatenation of the subvectors A_p, A_n, B_o, B_z.

For positive output current

$$C = \{ A_p,\ B_o,\ B_z,\ A_n \} \tag{2.16}$$

where the subvectors A_p, A_n, B_o, B_z are sorted in descending order, according to $|V_{cell}|$ values. When the output current is negative

$$C = \{ A_n,\ B_o,\ B_z,\ A_p \} \tag{2.17}$$

where the subvectors A_p, A_n, B_o, B_z are sorted in ascending order, according to $|V_{cell}|$ values. Again the switching functions corresponding to the n first entries in vector C are set to '1', where n is the output voltage level; all other switching functions are set to zero.

The algorithm can be summarised as follows

- sort cells by $|V_{cell}|$ (absolute cell voltage) errors
- subset A: take the n_{free} first elements (the parameter n_{free} is defined as the number of simultaneous commutations)
- subset B: take all other elements
- subset A_p: all the positive elements of A
- subset A_n: all the negative elements of A
- subset B_o: all elements of B corresponding to switching functions previously '1'
- subset B_z: all elements of B corresponding to switching functions previously '0'
- sort A_p, A_n, B_o, B_z vectors by V_{cell} voltage errors
- concatenation : $C = \{A_p, B_o, B_z, A_n\}$ or $C = \{A_n, B_o, B_z, A_p\}$ dependent on the output current direction
- set the first n (output level) elements of C to '1' (others are set to '0')

This algorithm of Donzel and Bornard, a modified version of their basic algorithm, will be referred to as the modified algorithm of Donzel and Bornard in this thesis.

2.3.2 THE MARTINS ET AL. ALGORITHM

A state substitution strategy is proposed by Martins, Roboam, Meynard and Carvalho in [35] for a three-cell flying-capacitor converter in a direct torque control application. Only single-level changes in an output phase-leg voltage are allowed in the application: a hysteresis-type of torque controller specifies whether a phase-leg voltage should increase or decrease by one level. This is in contrast with the space-vector PWM approach that is followed in this thesis.

change in phase-leg output voltage level	output current	select among cells at state...	... the one having
0	x	-	-
-1	negative	"on"	the highest V_{cell}
-1	positive	"on"	the lowest V_{cell}
+1	negative	"off"	the lowest V_{cell}
+1	positive	"off"	the highest V_{cell}

Table 2.2 *The Martins et al. stabilisation algorithm [35]*

In [35] it is stated – for a positive output line current – that the j^{th} cell voltage will never increase after the j^{th} switching function is set to '1'. An algorithm based on this principle is implemented in [35] and is shown in Table 2.2.

This strategy is functionally equivalent to the modified algorithm of Donzel and Bornard [34] in the case of single-level transitions of a phase-leg output voltage. By allowing only single-level changes in a phase-leg output voltage, a stabilisation algorithm is obtained without the intensive sorting and mapping operations of Donzel and Bornard. The Martins *et al.* strategy can not be applied to the SVM approach in this thesis; however, the practical implementation of the controller in [35] is of interest – this will be discussed further in subsection 4.2.

2.3.3 THE METHOD OF ESCALANTE, VANNIER AND ARZANDÉ

Escalante, Vannier and Arzandé describe a state substitution method for capacitor voltage stabilisation for a 4-cell flying-capacitor topology in [39]. Their method is presented in tabular form: one table defines the various switching states; a further two tables define the look-up tables governing state substitution during capacitor imbalance and balance respectively.

The set of available states and corresponding switching functions is defined in Table 2.3. An identifier is assigned to each switching state for referencing purposes in Table 2.4 and Table 2.5. For instance, the identifier ‘00’ is assigned to the switching state ‘0 0 0 0’.

state	switching functions				state	switching functions			
	S_4	S_3	S_2	S_1		S_4	S_3	S_2	S_1
00	0	0	0	0	08	1	0	0	0
01	0	0	0	1	09	1	0	0	1
02	0	0	1	0	0A	1	0	1	0
03	0	0	1	1	0B	1	0	1	1
04	0	1	0	0	0C	1	1	0	0
05	0	1	0	1	0D	1	1	0	1
06	0	1	1	0	0E	1	1	1	0
07	0	1	1	1	0F	1	1	1	1

Table 2.3 *Switching state definitions in a 4-cell flying-capacitor phase-leg [39]*

Follow-up states are specified in Table 2.4, for use during balanced conditions. For instance, by looking at the second row in Table 2.4, it can be seen that the state '00' may be succeeded by the state '00' (for an output voltage of level 0), or by the state '02' (for an output voltage of level 1), or by the state '03' (for an output voltage of level 2).

Table 2.5 identifies substitute states during imbalance, based on capacitor charge or discharge requirements. The first row gives the successor states for conditions where the voltage of capacitor C_1 is too low and the voltages of capacitors C_2 and C_3 are balanced: a switching state of '02' is specified for a level 1 output voltage, a '06' state is given for a level 2 output and a '0E' state is listed for a level 3 output. It can be seen that these successor states are specified without consideration of current switching states.

current switching state	**next switching state**				
	level 0	**level 1**	**level 2**	**level 3**	**level 4**
00	00	01,02, 04,08	x	x	x
01	00	02	03	x	x
02	00	04	06	x	x
03	x	02	06	07	x
04	00	08	0C	x	x
05	x	01	03	07	x
06	x	04	0C	0E	x
07	x	x	06	0E	0F
08	00	01	09	x	x
09	x	01	03	0B	x
0A	x	01	03	07	x
0B	x	x	03	07	0F
0C	x	08	09	0D	x
0D	x	x	09	0B	0F
0E	x	x	0C	0D	0F
0F	x	x	x	0D,0B, 07,0E	0F

Table 2.4 *Control logic for a 4-cell flying-capacitor phase-leg under balanced conditions [39]*

capacitor voltage condition						next switching state		
V_{C3}		V_{C2}		V_{C1}				
above limit	below limit	above limit	below limit	above limit	below limit	**level 1**	**level 2**	**level 3**
0	0	0	0	0	1	02	06	0E
0	0	0	0	1	0	01	09	0D
0	0	0	1	0	0	04	0C	0D
0	0	0	1	0	1	04	0C	0E
0	0	0	1	1	0	01	0C	0D
0	0	1	0	0	0	02	03	0B
0	0	1	0	0	1	02	0A	0E
0	0	1	0	1	0	01	03	0B
0	1	0	0	0	0	08	09	0B
0	1	0	0	0	1	08	0A	0E
0	1	0	0	1	0	08	09	0B
0	1	0	1	0	0	08	0C	0D
0	1	0	1	0	1	08	0C	0E
0	1	0	1	1	0	08	09	0D
0	1	1	0	0	0	08	03	0B
0	1	1	0	0	1	02	0A	0B
0	1	1	0	1	0	08	09	0B
1	0	0	0	0	0	04	06	07
1	0	0	0	0	1	04	06	07
1	0	0	0	1	0	01	05	07
1	0	0	1	0	0	04	0C	07
1	0	0	1	0	1	04	06	07
1	0	0	1	1	0	04	05	0D
1	0	1	0	0	0	02	03	07
1	0	1	0	0	1	02	06	07
1	0	1	0	1	0	01	03	07

Table 2.5 *Control logic for a 4-cell flying-capacitor phase-leg during imbalance [39]*

A tolerance band is utilised in [39] for each capacitor, thereby disregarding slight imbalances.

The method of Escalante, Vannier and Arzandé is a capacitor-voltage based algorithm, as successor states during capacitor imbalance are chosen according to capacitor voltages rather than cell voltages.

2.4 EVALUATION OF EXISTING STATE SUBSTITUTION STRATEGIES

The existing state substitution methods will be evaluated as to their suitability for the active filtering application described in this thesis.

2.4.1 EVALUATION OF THE MARTINS ET AL. ALGORITHM

The Martins *et al.* strategy was proposed in [35] for a pseudo hysteresis controller in an induction motor control application. Only single-level transitions are allowed in this application and can therefore not be applied to the SVM approach of this thesis.

2.4.2 CONSIDERATION OF THE METHOD OF ESCALANTE ET AL.

The Escalante *et al.* truth table implementation [39] can be extended in one of three ways in order to limit the switch commutations: the table for use in balanced conditions can be extended, the table for use during capacitor unbalance can be extended, or both tables can be extended. Such an extension, whichever way it is implemented, must define a transition for each of the 128 switching states to every other 128 successor switching states, based on capacitor charge/discharge requirements, number of switch commutations, output current direction and output voltage level. This means that 128 times 128, or 16384 Boolean equations must be specified, by hand, to set up a capacitor-voltage controller that will limit the number of switch commutations.

2.4.3 DONZEL AND BORNARD ALGORITHM EVALUATION

The algorithm of Donzel and Bornard [34] is cell-voltage based. It can be seen that the five sorting operations, as well as the indexing logic, the variable vector lengths, the selective copying processes and the extensive hard-coding of the algorithm contribute to the implementation difficulty of a controller utilising the Donzel and Bornard strategy. No practical result is given in [34]. An investigation into the practical implementation (PEC33) and algorithm compensating performance will be carried out in this thesis.

2.5 DERIVATION OF A NEW STATE SUBSTITUTION ALGORITHM

A new state substitution algorithm will be proposed in this subsection. It will be seen that the algorithm is an extension of the Escalante *et al.* algorithm. The algorithm is a capacitor-voltage based algorithm (in contrast with the cell-voltage based algorithm of Donzel and Bornard); a controller realisation and also the compensating performance will be investigated. In Table 2.6, a comparison and summary of the existing and proposed state

substitution strategies will be given, in order to clarify the position of the proposed algorithm among the set of existing algorithms.

	Martins *et al.* algorithm	**Escalante *et al.* algorithm**	**Donzel and Bornard algorithm**	**Proposed state substitution algorithm**
Characteristic pertaining to practical implementation	Only single-level transitions are allowed; this is not suitable to the SVM-based control of the active filter described in this thesis.	A 7-cell converter-based solution would require 16384 Boolean control equations to be specified by hand.	The Donzel and Bornard algorithm is a cell-voltage based algorithm. Five sorting operations, as well as the indexing logic, the variable vector lengths, the selective copying processes must be implemented; this, as well as compensating performance will be investigated in the thesis.	This algorithm will be proposed and investigated as a capacitor-voltage based algorithm in the thesis. The practical implementability of the algorithm and the compensating performance will be investigated.
Will the algorithm be investigated in the thesis?	No	No	Yes	Yes

Table 2.6 *Comparison and summary of the existing and proposed state substitution methods*

The new algorithm will be based on a simple weight allocation and summation method. A successor state will be chosen as the state with the highest specified rating (sum of weights). The rating of each possible switching state reflects the level of correspondence between the capacitor charging characteristic of the state and the capacitor charge/discharge response that is required for capacitor voltage balance.

Assuming that each cell capacitor has a similar contribution to the state correspondence, it makes sense to calculate a weight for each cell capacitor – the sum of these weights being the rating for state correspondence.

These weights are allocated according to the cell correspondence definitions listed in Table 2.1. The three basic degrees of cell correspondence are defined in Table 2.1:

correct (*r*), *no effect* (*n*) and *incorrect* (*w*). The weights that are assigned to these correspondence levels will be indicated as (*r*, *n*, *w*) values.

The ranking orders that were obtained with the sets of weights (*r*, *n*, *w*) = (2, 1, 0) and (4, 1, 0) and (5, 4, 0) are displayed in Table 2.2. The ranking orders in the first two columns attempt to maximise the choice of *correct* (*r*) cell correspondences; the ranking order in column 3 attempts to minimise the choice of *incorrect* (*w*) cell correspondences. Both strategies are expected to contribute to good capacitor voltage regulation, as frequent correct choices and infrequent incorrect choices should result in lower capacitor voltage ripple.

cell correspondence level	state characteristic for cell capacitor	cell capacitor charge/discharge requirement
correct (r)	charge discharge no effect/ charge/ discharge	charge discharge neither
no action (n)	no effect	charge/discharge
incorrect (w)	charge discharge	discharge charge

Table 2.1 *Definition of cell correspondence levels*

As can be seen in Table 2.2 and Table 2.3, the state correspondence ranking orders are determined by the choice of cell correspondence weights.

This approach to weight allocation and ranking specification is far from ideal. A few problems can be identified in Table 2.2 – the ranking order in the leftmost column shows the problem of a non-uniquely specified state correspondence ranking order: will a <*r r n n*> really have an equally beneficial effect as a <*r r r w*> state correspondence?

The determination (by hand) of these weights can become quite difficult, as

- not every desired ranking order can be obtained by the limited number of weights already in use
- not every choice of weights gives an uniquely specified state correspondence ranking order

- maximising the choice of *correct* (*r*) cell correspondences, while simultaneously minimising the choice of *incorrect* (*w*) cell correspondences is very difficult or impossible to obtain with the use of three weights.
- similar ranking orders for different number of cells require some very different sets of weights (for instance, a 6-cell ranking order similar to that obtained in Table 2.2 with the weights (*r*, *n*, *w*) = (4, 1, 0), would require values of (*r*, *n*, *w*) = (6,1,0))

(r, n, w) = (2,1,0)					(r, n, w) = (4,1,0)					(r, n, w) = (5,4,0)				
state correspondence				**effective rating**	**state correspondence**				**effective rating**	**state correspondence**				**effective rating**
r	r	r	r	8	r	r	r	r	16	r	r	r	r	20
r	r	r	n	7	r	r	r	n	13	r	r	r	n	19
r	r	n	n	6	r	r	r	w	12	r	r	n	n	18
r	r	r	w	6	r	r	n	n	10	r	n	n	n	17
r	n	n	n	5	r	r	n	w	9	n	n	n	n	16
r	r	n	w	5	r	r	w	w	8	r	r	r	w	15
n	n	n	n	4	r	n	n	n	7	r	r	n	w	14
r	n	n	w	4	r	n	n	w	6	r	n	n	w	13
r	r	w	w	4	r	n	w	w	5	n	n	n	w	12
n	n	n	w	3	n	n	n	n	4	r	r	w	w	10
r	n	w	w	3	r	w	w	w	4	r	n	w	w	9
n	n	w	w	2	n	n	n	w	3	n	n	w	w	8
r	w	w	w	2	n	n	w	w	2	r	w	w	w	5
n	w	w	w	1	n	w	w	w	1	n	w	w	w	4
w	w	w	w	0	w	w	w	w	0	w	w	w	w	0

Table 2.2 *Five-cell state ranking orders, obtained with extended set of weights*

Another concern is that $|V_d|$ (absolute capacitor error voltage) values are not taken into account with these ranking orders. This could be problematic in some cases. For instance, the state correspondence <*r r r w*> is listed as the third best option in the (*r, n*, *w*) = (4, 1, 0) ranking order. When it happens that the cell capacitor with the highest $|V_d|$ value has an *incorrect* (*w*) cell correspondence, <*r r r w*> will be very undesirable, but still will be selected above other, more suitable, state correspondences.

Solutions to this problem are to ignore small $|V_d|$ values when large $|V_d|$ values are encountered, or to make use of an extended set of weights that operate on the capacitor with the largest $|V_d|$ value. It was found that both these strategies tend to regulate one capacitor voltage at the expense of another, however. A ranking order obtained with such a system is listed in Table 2.3.

It would make sense to specify a ranking order that takes $|V_d|$ ordering into account. A simple sorting algorithm is sufficient for the number of unbalance voltage values. Again, the problem is the unique specification of a ranking order.

$(r_3, n_3, w_3, r, n, w) = (26,13,0,4,1,0)$

state correspondence				awarded rating	state correspondence				awarded rating
r_3	r	r	r	38	n_3	r	w	w	17
r_3	r	r	n	35	n_3	n	n	n	16
r_3	r	r	w	34	n_3	n	n	w	15
r_3	r	n	n	32	n_3	n	w	w	14
r_3	r	n	w	31	n_3	w	w	w	13
r_3	r	w	w	30	w_3	r	r	r	12
r_3	n	n	n	29	w_3	r	r	n	9
r_3	n	n	w	28	w_3	r	r	w	8
r_3	n	w	w	27	w_3	r	n	n	6
r_3	w	w	w	26	w_3	r	n	w	5
n_3	r	r	r	25	w_3	r	w	w	4
n_3	r	r	n	22	w_3	n	n	n	3
n_3	r	r	w	21	w_3	n	n	w	2
n_3	r	n	n	19	w_3	n	w	w	1
n_3	r	n	w	18	w_3	w	w	w	0

Table 2.3 *Five-cell state ranking orders, obtained with extended set of weights*

A solution would be to view the ranking order as a number of base 3 – each digit has one of three values (r, n, w). If the (r, n, w) were chosen to represent values of (2, 1, 0), a uniquely specified ranking order would be obtained by converting the state correspondence value to a number of base 10. The ranking order obtained with this idea is shown in Table 2.4. This ranking order is not satisfactory, however, as the third best state correspondence contains an incorrect cell correspondence – here a $<r_3\ r_2\ r_1\ w_0>$ state correspondence is specified to be more beneficial than a $<r_3\ r_2\ n_1\ r_0>$ state correspondence.

The final weight allocation strategy shown in Table 2.5 is a variation on the strategy of Table 2.4. The order is obtained by moving all state correspondence entries containing an incorrect cell correspondence, downwards in the state ranking order, but keeping the original order in all other respects. This order is implemented as a binary system: two totals are calculated, one (based solely on the *correct* and *no action* cell correspondences) forms the LSB of the rating and one (based on *incorrect* cell correspondences) forms the MSB of the rating.

$$(r_x, n_x, w_x) = 3^x (2, 1, 0)$$

state correspondence				awarded rating	state correspondence				awarded rating
r_3	r_2	r_1	r_0	80	n_3	n_2	n_1	n_0	40
r_3	r_2	r_1	n_0	79	n_3	n_2	n_1	w_0	39
r_3	r_2	r_1	w_0	78	n_3	n_2	w_1	r_0	38
r_3	r_2	n_1	r_0	77	n_3	n_2	w_1	n_0	37
r_3	r_2	n_1	n_0	76	n_3	n_2	w_1	w_0	36
r_3	r_2	n_1	w_0	75	n_3	w_2	w_1	w_0	35
r_3	r_2	w_1	r_0	74	n_3	w_2	r_1	n_0	34
r_3	r_2	w_1	n_0	73	n_3	w_2	r_1	w_0	33
r_3	r_2	w_1	w_0	72	n_3	w_2	n_1	r_0	32
r_3	n_2	r_1	r_0	71	n_3	w_2	n_1	n_0	31
r_3	n_2	r_1	n_0	70	n_3	w_2	n_1	w_0	30
r_3	n_2	r_1	w_0	69	n_3	w_2	w_1	r_0	29
r_3	n_2	n_1	r_0	68	n_3	w_2	w_1	n_0	28
r_3	n_2	n_1	n_0	67	n_3	w_2	w_1	w_0	27
r_3	n_2	n_1	w_0	66	w_3	r_2	r_1	r_0	26
r_3	n_2	w_1	r_0	65	w_3	r_2	r_1	n_0	25
r_3	n_2	w_1	n_0	64	w_3	r_2	r_1	w_0	24
r_3	n_2	w_1	w_0	63	w_3	r_2	n_1	r_0	23
r_3	w_2	r_1	r_0	62	w_3	r_2	n_1	n_0	22
r_3	w_2	r_1	n_0	61	w_3	r_2	n_1	w_0	21
r_3	w_2	r_1	w_0	60	w_3	r_2	w_1	r_0	20
r_3	w_2	n_1	r_0	59	w_3	r_2	w_1	n_0	19
r_3	w_2	n_1	n_0	58	w_3	r_2	w_1	w_0	18
r_3	w_2	n_1	w_0	57	w_3	n_2	r_1	r_0	17
r_3	w_2	w_1	r_0	56	w_3	n_2	r_1	n_0	16
r_3	w_2	w_1	n_0	55	w_3	n_2	r_1	w_0	15
r_3	w_2	w_1	w_0	54	w_3	n_2	n_1	r_0	14
n_3	r_2	r_1	r_0	53	w_3	n_2	n_1	n_0	13
n_3	r_2	r_1	n_0	52	w_3	n_2	n_1	w_0	12
n_3	r_2	r_1	w_0	51	w_3	n_2	w_1	r_0	11
n_3	r_2	n_1	r_0	50	w_3	n_2	w_1	n_0	10
n_3	r_2	n_1	n_0	49	w_3	n_2	w_1	w_0	9
n_3	r_2	n_1	w_0	48	w_3	w_2	r_1	r_0	8
n_3	r_2	w_1	r_0	47	w_3	w_2	r_1	n_0	7
n_3	r_2	w_1	n_0	46	w_3	w_2	r_1	w_0	6
n_3	r_2	w_1	w_0	45	w_3	w_2	n_1	r_0	5
n_3	n_2	r_1	r_0	44	w_3	w_2	n_1	n_0	4
n_3	n_2	r_1	n_0	43	w_3	w_2	n_1	w_0	3
n_3	n_2	r_1	w_0	42	w_3	w_2	w_1	r_0	2
n_3	n_2	n_1	r_0	41	w_3	w_2	w_1	n_0	1
					w_3	w_2	w_1	w_0	0

Table 2.4 *Five-cell state ranking orders, obtained with extended sets of weights*

$$(r_x, n_x, w_x) = (2^{x+cells-1} + 2^x, 2^{x+cells-1}, 0)$$

state correspondence				MS half	LS half	MS total	LS total	state rating	state correspondence				MS half	LS half	MS total	LS total	state rating
r_3	r_2	r_1	r_0	1111	1111	240	15	255	n_3	w_2	r_1	r_0	1011	0011	176	3	179
r_3	r_2	r_1	n_0	1111	1110	240	14	254	n_3	w_2	r_1	n_0	1011	0010	176	2	178
r_3	r_2	n_1	r_0	1111	1101	240	13	253	n_3	w_2	n_1	r_0	1011	0001	176	1	177
r_3	r_2	n_1	n_0	1111	1100	240	12	252	n_3	w_2	n_1	n_0	1011	0000	176	0	176
r_3	n_2	r_1	r_0	1111	1011	240	11	251	r_3	w_2	r_1	w_0	1010	1010	160	10	170
r_3	n_2	r_1	n_0	1111	1010	240	10	250	r_3	w_2	n_1	w_0	1010	1000	160	8	168
r_3	n_2	n_1	r_0	1111	1001	240	9	249	n_3	w_2	r_1	w_0	1010	0010	160	2	162
r_3	n_2	n_1	n_0	1111	1000	240	8	248	n_3	w_2	n_1	w_0	1010	0000	160	0	160
n_3	r_2	r_1	r_0	1111	0111	240	7	247	r_3	w_2	w_1	r_0	1001	1001	144	9	153
n_3	r_2	r_1	n_0	1111	0110	240	6	246	n_3	w_2	w_1	n_0	1001	1000	144	8	152
n_3	r_2	n_1	r_0	1111	0101	240	5	245	n_3	w_2	w_1	r_0	1001	0001	144	1	145
n_3	r_2	n_1	n_0	1111	0100	240	4	244	n_3	w_2	w_1	n_0	1001	0000	144	0	144
n_3	n_2	r_1	r_0	1111	0011	240	3	243	r_3	w_2	w_1	w_0	1000	1000	128	8	136
n_3	n_2	r_1	n_0	1111	0010	240	2	242	n_3	w_2	w_1	w_0	1000	0000	128	0	128
n_3	n_2	n_1	r_0	1111	0001	240	1	241	w_3	r_2	r_1	r_0	0111	0111	112	7	119
n_3	n_2	n_1	n_0	1111	0000	240	0	240	w_3	r_2	r_1	n_0	0111	0110	112	6	118
r_3	r_2	r_1	w_0	1110	1110	224	14	238	w_3	r_2	n_1	r_0	0111	0101	112	5	117
r_3	r_2	n_1	w_0	1110	1100	224	12	236	w_3	r_2	n_1	n_0	0111	0100	112	4	116
r_3	n_2	r_1	w_0	1110	1010	224	10	234	w_3	n_2	r_1	r_0	0111	0011	112	3	115
r_3	n_2	n_1	w_0	1110	1000	224	8	232	w_3	n_2	r_1	n_0	0111	0010	112	2	114
n_3	r_2	r_1	w_0	1110	0110	224	6	230	w_3	n_2	n_1	r_0	0111	0001	112	1	113
n_3	r_2	n_1	w_0	1110	0100	224	4	228	w_3	n_2	n_1	n_0	0111	0000	112	0	112
n_3	n_2	r_1	w_0	1110	0010	224	2	226	w_3	r_2	r_1	w_0	0110	0110	96	6	102
n_3	n_2	n_1	w_0	1110	0000	224	0	224	w_3	r_2	n_1	w_0	0110	0100	96	4	100
r_3	r_2	w_1	r_0	1101	1101	208	13	221	w_3	n_2	r_1	w_0	0110	0010	96	2	98
r_3	r_2	w_1	n_0	1101	1100	208	12	220	w_3	n_2	n_1	w_0	0110	0000	96	0	96
r_3	n_2	w_1	r_0	1101	1001	208	9	217	w_3	r_2	w_1	r_0	0101	0101	80	5	85
r_3	n_2	w_1	n_0	1101	1000	208	8	216	w_3	r_2	w_1	n_0	0101	0100	80	4	84
n_3	r_2	w_1	r_0	1101	0101	208	5	213	w_3	n_2	w_1	r_0	0101	0001	80	1	81
n_3	r_3	w_1	n_0	1101	0100	208	4	212	w_3	n_2	w_1	n_0	0101	0000	80	0	80
n_3	n_2	w_1	r_0	1101	0001	208	1	209	w_3	r_2	w_1	w_0	0100	0100	64	4	68
n_3	n_2	w_1	n_0	1101	0000	208	0	208	w_3	n_2	w_1	w_0	0100	0000	64	0	64
r_3	r_2	w_1	w_0	1100	1100	192	12	204	w_3	w_2	r_1	r_0	0011	0011	48	3	51
r_3	n_2	w_1	w_0	1100	1000	192	8	200	w_3	w_2	r_1	n_0	0011	0010	48	2	50
n_3	r_2	w_1	w_0	1100	0100	192	4	196	w_3	w_2	n_1	r_0	0011	0001	48	1	49
n_3	n_2	w_1	w_0	1100	0000	192	0	192	w_3	w_2	n_1	n_0	0011	0000	48	0	48
r_3	w_2	r_1	r_0	1011	1011	176	11	187	w_3	w_2	r_1	w_0	0010	0010	32	2	34
r_3	w_2	r_1	n_0	1011	1010	176	10	186	w_3	w_2	n_1	w_0	0010	0000	32	0	32
r_3	w_2	n_1	r_0	1011	1001	176	9	185	w_3	w_2	w_1	r_0	0001	0001	16	1	17
r_3	w_2	n_1	n_0	1011	1000	176	8	184	w_3	w_2	w_1	n_0	0001	0000	16	0	16
									w_3	w_2	w_1	n_0	0000	0000	0	0	0

where r_3 is a *correct* cell correspondence for the cell capacitor with largest $|V_d|$
where n_3 is a *no action* cell correspondence for the cell capacitor with largest $|V_d|$
where w_3 is an *incorrect* cell correspondence for the cell capacitor with largest $|V_d|$
where r_2 is a *correct* cell correspondence for the cell capacitor with second largest $|V_d|$
where n_2 is a *no action* cell correspondence for the cell capacitor with second largest $|V_d|$
where w_2 is an *incorrect* cell correspondence for the cell capacitor with second largest $|V_d|$
⋮

Table 2.5 *Five-cell state ranking orders, obtained with extended sets of weights*

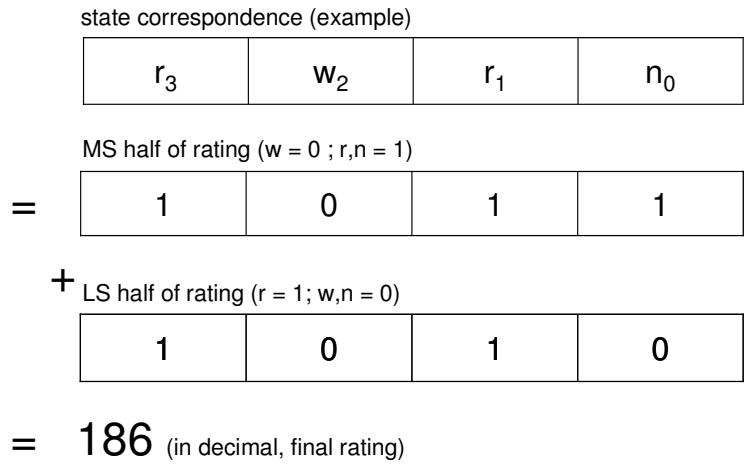


Figure 2.4 *State rating determination process*

In Figure 2.4 the digital implication of the weight allocation strategy in Table 2.5 is illustrated. The same notation used previously, is used again: r_3 indicates a *correct* cell correspondence for a capacitor with the largest $|V_d|$ (absolute error) voltage; w_2 represents an *incorrect* cell correspondence for the second largest error voltage; r_1 represents a *correct* cell correspondence for the third largest error voltage; n_0 indicates a *no action* cell correspondence for the smallest error voltage. The most significant half of the state rating is specified by setting all the bit positions corresponding to *n* or *r*-values to '1' (and all others to zero). The least significant half of the rating is determined by setting all the bit positions corresponding to *r*-values, to '1'. The effect of such a series of operations is to offset all state correspondences with no *w*-correspondences at the top of the state correspondence ranking order, as shown in Table 2.5.

The ratings of all the states that do not have the correct number of switch changes, or that do not represent an output voltage of the desired level, are zeroed out. Finally, the state with the highest rating is selected.

Whenever a number of different states have the same (maximum) rating, all of these states will be marked as the next output state. This poses no problem for the correct working of the simulation, however. Parallel transition events in the simulation state machines are processed sequentially. This means that only one output state (which is dependent on the

state graph order in the simulation model) will be selected. When implementing the control algorithm, care will be taken that only one output state is selected.

(hmin = hmax = 1µs) (Tmax = 300m)		max |Vd|	Mean	RMS AC	comparison (1 = best performance) max |Vd|	RMS AC
(r, n, w) = (2,1,0)	Vc_4	37.324	1439.1	11.103	3	3
	Vc_3	34.733	1079.7	10.128		
	Vc_2	49.488	719.85	10.602		
	Vc_1	38.974	359.81	10.82		
	max value	49.488		11.103		
(r, n, w) = (4,1,0)	Vc_4	37.628	1440.1	11.245	5	6
	Vc_3	53.902	1076.4	12.192		
	Vc2	52.441	720.37	12.273		
	Vc1	38.974	359.21	10.709		
	max value	53.902		12.273		
(r, n, w) = (5,4,0)	Vc_4	38.17	1439.7	11.305	2	4
	Vc_3	34.528	1081.1	10.144		
	Vc_2	38.409	718.18	10.392		
	Vc_1	38.943	358.72	10.688		
	max value	38.943		11.305		
(r_3, n_3, w_3, r, n, w) = (26,13,0,4,1,0)	Vc_4	49.191	1442.1	10.892	6	5
	Vc_3	59.652	1078.8	11.22		
	Vc_2	47.69	719.78	10.444		
	Vc_1	55.854	357.29	11.849		
	max value	59.652		11.849		
(r_x, n_x, w_x) = 3^x (2, 1, 0)	Vc_4	49.676	1443.2	9.9997	4	2
	Vc_3	50.55	1078.7	10.584		
	Vc_2	47.784	720.43	10.866		
	Vc_1	48.221	356.58	10.117		
	max value	50.55		10.866		
(r_x, n_x, w_x) = (2^x+cells + 2^x, 2^x+cells, 0)	Vc_4	38.322	1439.9	9.2294	1	1
	Vc_3	36.089	1080.1	8.5786		
	Vc_2	35.288	719.79	8.7759		
	Vc_1	37.579	360	9.2003		
	max value	38.322		9.2294		

Table 2.6 *Comparative data for various sets of weights*

In Table 2.6, comparative data for different sets of weights – without any switching frequency limitations – is listed. By not limiting the number of switch commutations, a

comparison can be made based solely on the relative merits of the various weight-allocation methods.

2.6 LINE-TO-LINE REDUNDANCY METHODS

The three-phase line-to line redundant state configurations can be utilised in conjunction with a per-phase stabilisation strategy. The following subsections will cover aspects relating to three-phase redundancies in detail.

2.6.1 UTILISATION OF LINE-TO-LINE REDUNDANCIES FOR CAPACITOR VOLTAGE STABILISATION

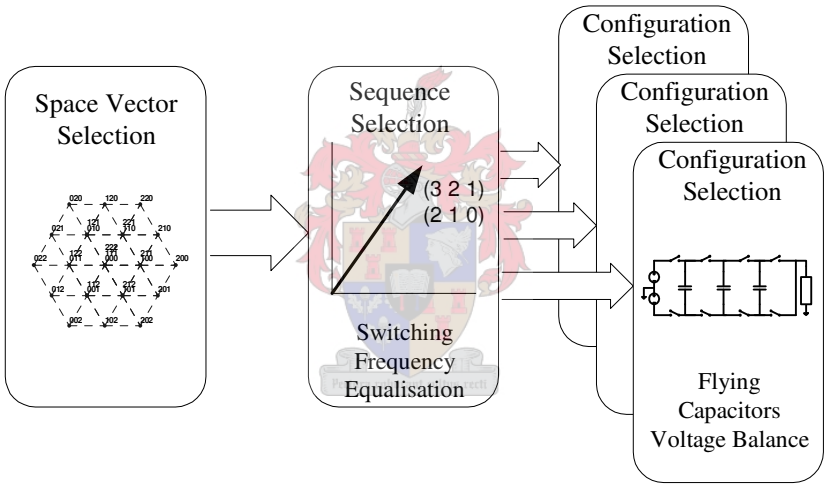


Figure 2.5 *Three-step control of the flying-capacitor multilevel converter [35, 38]*

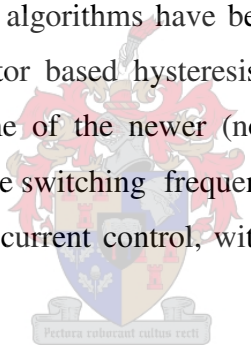
Two degrees of freedom can be identified in the space-vector representation of flying-capacitor converter output voltages. A three-phase output voltage configuration must be chosen from the available line-to-line redundancies to realise a specific output space vector. Thereafter, a switching state must be selected from the available phase redundancies for each phase-leg output voltage.

The three-step approach is implemented in [35] and [38] as an induction motor DTC controller, and is summarised in Figure 2.5. It was found in [38] that a good balance of conduction and switching losses is achieved as a by-product of the capacitor stabilisation strategy.

2.6.2 THREE-PHASE MODULATION TECHNIQUES

Various current control strategies, such as PWM current control, hysteresis control, deadbeat control, sliding-mode control and fuzzy control are generally implemented in active filter applications [2]. Space-vector PWM current control (deadbeat control) is used in this study. Although the basic hysteresis control is simple to implement, it has some inherent drawbacks. Simple hysteresis controllers should be operated at higher frequencies to overcome these problems, thus making it unsuitable for high-power applications [19].

Problems of the basic hysteresis algorithm are variable switching frequency (that complicates calculation of losses in the system, complicates design of switching ripple filters and can cause resonances in the utility grid [40]), high-frequency switching limit-cycles, and instantaneous errors twice the value of the hysteresis band, due to phase interactions. Improved hysteresis algorithms have been proposed at the cost of increased complexity, however. Space-vector based hysteresis controllers, for example, eliminate phase interactions [40, 41]. Some of the newer (non-multilevel) strategies emulate the centred voltage pulses and stable switching frequency (during small current reference variations) of space-vector PWM current control, with increased switching during current transients [42].



In high-power applications, a low switching frequency of the power devices must be realised [43]. Space-vector PWM modulation (especially in multilevel topologies) can be optimised to minimise and achieve low device switching frequencies. Space-vector PWM (SVM) leads to low current ripple and good DC-bus utilisation and therefore is extremely suitable for high-power applications [43]. Previously, the use of space-vector modulation was limited by the computational complexity for increased-level multilevel converters. However, the calculations for SVM of multilevel converters are now significantly simplified by the fast multilevel algorithm of Celanovic *et al.* [37].

2.6.2.1 CONVENTIONAL SPACE VECTOR MODULATION

In space-vector modulation, by tradition, the available output voltage combinations are represented as vectors in a complex (alpha-beta) plane. For a three-phase, three-wire converter, the alpha-beta components can be determined as follows

$$\begin{bmatrix} V_{REF\alpha} \\ V_{REF\beta} \end{bmatrix} = \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix}, \tag{2.18}$$

where the phase voltages V_a, V_b and V_c and the V_{REF} alpha-beta (linear modulation range) components are scaled to a maximum amplitude of 1. Often, the Park transform in equation (2.18) is scaled by an additional factor $\sqrt{2/3}$ to preserve the total power in the transformed system.

A vector in the two dimensional alpha-beta plane can also be represented as a complex number, as in [44, 45]

$$\begin{aligned} \boldsymbol{V} &= V \cdot e^{j\theta} \\ &= V_\alpha + jV_\beta \end{aligned} \tag{2.19}$$

where the alpha-axis component corresponds to the real value and the beta-axis component corresponds to the imaginary component value.

A specified reference output vector is realised by taking a time-average of the three nearest voltage vectors over one switching subcycle. For instance, in sector I, the reference vector can be expressed in terms of $\boldsymbol{V}_1$, $\boldsymbol{V}_2$ and $\boldsymbol{V}_{0/7}$, as in Figure 2.6

$$\begin{aligned} \boldsymbol{V}_{REF} &= \frac{T_1}{T_{sw}}\boldsymbol{V}_1 + \frac{T_2}{T_{sw}}\boldsymbol{V}_2 + \frac{T_{0/7}}{T_{sw}}\boldsymbol{V}_{0/7} \\ &= d_1\boldsymbol{V}_1 + d_2\boldsymbol{V}_2 + d_{0/7}\boldsymbol{V}_{0/7} \end{aligned} \quad . \tag{2.20}$$

The duty cycles d_1, d_2 and $d_{0/7}$ should not be confused with the concept of duty cycles in carrier-based modulation schemes. In this thesis, a duty cycle value indicates a time-duration ratio for a specific vector output, whereas in carrier-based modulation, the concept of duty cycles relates to the average phase-leg output voltage.

Expressions for the calculation of the duty cycles d_1, d_2 and $d_{0/7}$ for two-level SVM are listed in Table 2.6. These expressions can be derived geometrically [46] or by substitution [44]. Generally, a set of inequalities is evaluated to determine the sector or region of the reference vector, such as in [45].

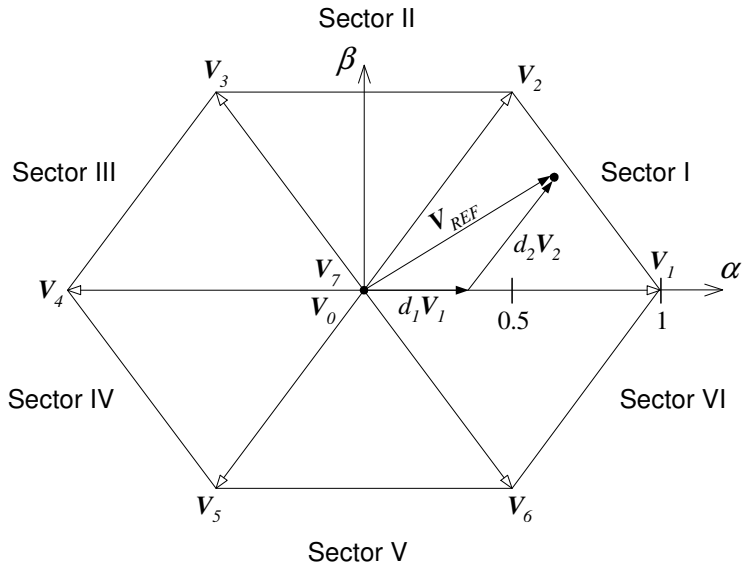


Figure 2.6 *Conventional SVM reference vector decomposition*

Sector I	Sector II	Sector III
$d_1 = \frac{2}{\sqrt{3}} V_{REF} \cos(\theta + \frac{\pi}{6})$	$d_2 = \frac{2}{\sqrt{3}} V_{REF} \cos(\theta + \frac{11\pi}{6})$	$d_3 = \frac{2}{\sqrt{3}} V_{REF} \cos(\theta + \frac{3\pi}{2})$
$d_2 = \frac{2}{\sqrt{3}} V_{REF} \cos(\theta + \frac{3\pi}{2})$	$d_3 = \frac{2}{\sqrt{3}} V_{REF} \cos(\theta + \frac{7\pi}{6})$	$d_4 = \frac{2}{\sqrt{3}} V_{REF} \cos(\theta + \frac{5\pi}{6})$
$d_{0/7} = 1 - d_1 - d_2$	$d_{0/7} = 1 - d_2 - d_3$	$d_{0/7} = 1 - d_3 - d_4$
Sector IV	**Sector V**	**Sector VI**
$d_4 = \frac{2}{\sqrt{3}} V_{REF} \cos(\theta + \frac{7\pi}{6})$	$d_5 = \frac{2}{\sqrt{3}} V_{REF} \cos(\theta + \frac{5\pi}{6})$	$d_6 = \frac{2}{\sqrt{3}} V_{REF} \cos(\theta + \frac{\pi}{2})$
$d_5 = \frac{2}{\sqrt{3}} V_{REF} \cos(\theta + \frac{\pi}{2})$	$d_6 = \frac{2}{\sqrt{3}} V_{REF} \cos(\theta + \frac{\pi}{6})$	$d_1 = \frac{2}{\sqrt{3}} V_{REF} \cos(\theta + \frac{11\pi}{6})$
$d_{0/7} = 1 - d_4 - d_5$	$d_{0/7} = 1 - d_5 - d_6$	$d_{0/7} = 1 - d_6 - d_1$

Table 2.6 *Duty cycle calculations for a reference vector* $\mathbf{V}_{REF} = V_{REF} \cdot e^{j\theta}$ *in conventional SVM [46].*

The duty cycle values can also be calculated directly from alpha-beta values [45], as in Table 2.7. The specified volt-second average can be achieved by switching the three nearest vectors in any order. However, the positioning of the vectors within a switching subcycle has an impact on both the switching frequency and the harmonic content of the synthesised output voltage.

Sector I	Sector II	Sector III
$d_1 = V_\alpha - \frac{1}{\sqrt{3}} V_\beta$	$d_2 = V_\alpha + \frac{1}{\sqrt{3}} V_\beta$	$d_3 = \frac{2}{\sqrt{3}} V_\beta$
$d_2 = \frac{2}{\sqrt{3}} V_\beta$	$d_3 = -V_\alpha + \frac{1}{\sqrt{3}} V_\beta$	$d_4 = -V_\alpha - \frac{1}{\sqrt{3}} V_\beta$
$d_{0/7} = 1 - d_1 - d_2$	$d_{0/7} = 1 - d_2 - d_3$	$d_{0/7} = 1 - d_3 - d_4$
Sector IV	**Sector V**	**Sector VI**
$d_4 = -V_\alpha + \frac{1}{\sqrt{3}} V_\beta$	$d_5 = -V_\alpha - \frac{1}{\sqrt{3}} V_\beta$	$d_6 = -\frac{2}{\sqrt{3}} V_\beta$
$d_5 = -\frac{2}{\sqrt{3}} V_\beta$	$d_6 = V_\alpha - \frac{1}{\sqrt{3}} V_\beta$	$d_1 = V_\alpha + \frac{1}{\sqrt{3}} V_\beta$
$d_{0/7} = 1 - d_4 - d_5$	$d_{0/7} = 1 - d_5 - d_6$	$d_{0/7} = 1 - d_6 - d_1$

Table 2.7 *Duty cycle calculations for a reference vector* $\boldsymbol{V}_{REF} = V_\alpha + jV_\beta$ *in conventional SVM [45]. Alpha-beta values scaled as in equation (2.18).*

The minimum switching frequency is achieved when transitions between any two consecutive states involve switching of only one converter leg [47]. When active pulses (i.e. combined active vector pulses) are centred in the half carrier interval, a comparatively good harmonic performance is achieved [36, 47]. (Of the modulation strategies considered in [36], the harmonic performance of centred SVM is surpassed only by discontinuous SVM at modulation indices greater than about 0,9.) In addition, the vector order should be reversed in the second half carrier equivalent cycle for an improved harmonic response [48]. The pulse centring also results in an increased modulation depth [36]. A maximum (linear range) modulation index of 1,15 can be achieved with space vector modulation whereas the maximum linear range modulation index of sinusoidal modulation is 1 [47].

2.6.2.1 MULTILEVEL CONVERTER SPACE VECTOR MODULATION

An N-level converter has N^3 output states and can generate $3N(N-1)+1$ space vectors in the stationary alpha-beta plane [35]. Duty cycles for multilevel converter SVM can be calculated geometrically, as for conventional SVM. An example hereof is given in Figure 2.7 and Table 2.8 for 3-level converter SVM, with the reference vector decomposition as in equation (2.21).

$$\boldsymbol{V}_{REF} = d_a \boldsymbol{V}_a + d_b \boldsymbol{V}_b + d_c \boldsymbol{V}_c . \tag{2.21}$$

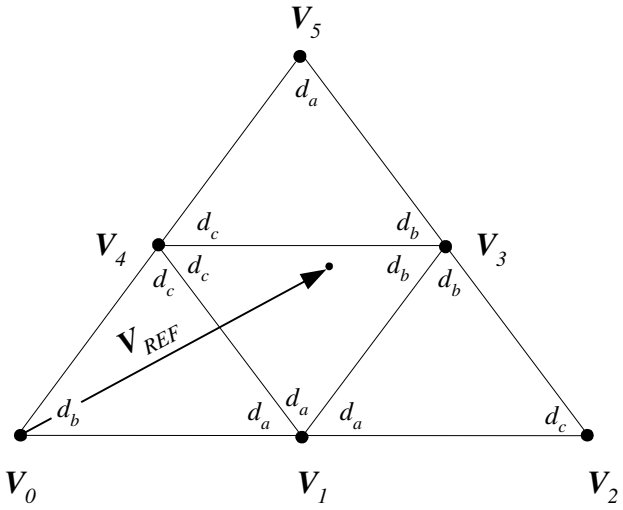


Figure 2.7 *Sector I subdivision for SVM of three-level voltage source converter, for SVM of a three-level converter [44].*

In addition to the duty cycle calculations, inequalities must be set up to delimit each region in the alpha-beta plane.

Region 1	Region 2
$d_a = \frac{4}{\sqrt{3}} V_{REF} \sin(\frac{\pi}{3} - \theta)$	$d_a = 2 - \frac{4}{\sqrt{3}} V_{REF} \sin(\theta + \frac{\pi}{3})$
$d_b = 1 - \frac{4}{\sqrt{3}} V_{REF} \sin(\theta + \frac{\pi}{3})$	$d_b = \frac{4}{\sqrt{3}} V_{REF} \sin(\theta)$
$d_c = \frac{4}{\sqrt{3}} V_{REF} \sin(\theta)$	$d_c = \frac{4}{\sqrt{3}} V_{REF} \sin(\frac{\pi}{3} - \theta) - 1$
Region 3	**Region 4**
$d_a = 1 - \frac{4}{\sqrt{3}} V_{REF} \sin(\theta)$	$d_a = \frac{4}{\sqrt{3}} V_{REF} \sin(\theta) - 1$
$d_b = \frac{4}{\sqrt{3}} V_{REF} \sin(\theta + \frac{\pi}{3}) - 1$	$d_b = \frac{4}{\sqrt{3}} V_{REF} \sin(\frac{\pi}{3} - \theta)$
$d_c = \frac{4}{\sqrt{3}} V_{REF} \sin(\theta - \frac{\pi}{3}) + 1$	$d_c = 2 - \frac{4}{\sqrt{3}} V_{REF} \sin(\theta + \frac{\pi}{3})$

Table 2.8 *Duty cycle calculations for a reference vector* $\mathbf{V}_{REF} = V_\alpha + jV_\beta$ *in a three-level converter [45]. Alpha-beta values scaled as in equation (2.18).*

In general, algorithms such as these are not used for more-than-3-level converters, due to the computational complexity [37]. Strategies such as [49], aim to simplify calculations, by implementing multilevel converter SVM in terms of conventional SVM that is null-shifted.

A slightly different approach is followed by Celanovic [37] – this is detailed in subsection 2.6.3.

Again, with SVM for multilevel converters, the vector order is significant. The optimal harmonic performance is achieved by equalising the start and end redundant vector period in each half-carrier equivalent cycle [48]. In addition, the vector order should be reversed in the second half carrier equivalent cycle (as for conventional SVM) for an improved harmonic response [48].

2.6.3 THE FAST MULTILEVEL SPACE-VECTOR ALGORITHM OF CELANOVIC ET AL.

A new approach to space vector modulation for multilevel converters was proposed by Celanovic in [37]. A redefinition of the basis vectors in the alpha-beta plane is done in order to simplify the selection of the three nearest vectors and calculations of duty cycle values. It is stated in [37] that this algorithm can be seen as an important step towards realising a universal (multilevel) digital controller.

2.6.3.1 COORDINATE TRANSFORMATION

The components of the reference voltage along the g-h axes (defined in Figure 2.8) can be calculated from

$$\begin{bmatrix} V_{REF_g} \\ V_{REF_h} \end{bmatrix} = cells \cdot \begin{bmatrix} 1 & -\frac{1}{\sqrt{3}} \\ 0 & \frac{2}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} V_{REF_\alpha} \\ V_{REF_\beta} \end{bmatrix}, \tag{2.22}$$

with the alpha-beta components scaled to a maximum amplitude of 1, and the resulting g-h components scaled to a maximum amplitude *cells*.

The alpha-beta components are determined from the per-unit phase voltages, as in equation (2.23).

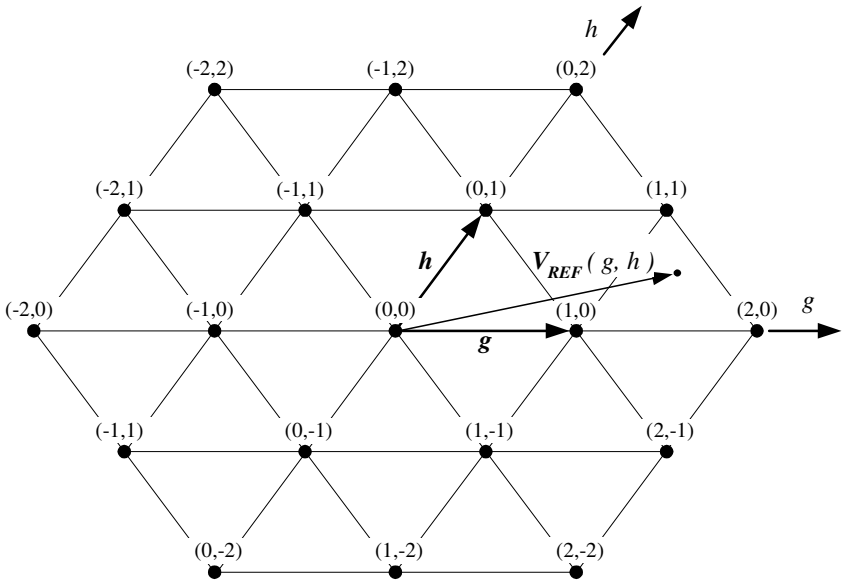


Figure 2.8 *Definition of the new non-orthogonal basis vectors g and h [37]*

$$\begin{bmatrix} V_{REF\alpha} \\ V_{REF\beta} \end{bmatrix} = \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix}. \tag{2.23}$$

2.6.3.2 DETERMINATION OF NEAREST THREE VECTORS

Once the g-h components of the output reference vector are known, the parallelogram containing the reference vector (or point) can be determined. The four corners of the parallelogram are formed by $\boldsymbol{V}_{ul}$, $\boldsymbol{V}_{lu}$, $\boldsymbol{V}_{uu}$ and $\boldsymbol{V}_{ll}$.

$$\begin{aligned} \boldsymbol{V}_{ul} &= \begin{bmatrix} ceil\ V_{REF\,g} \\ floor\ V_{REF\,h} \end{bmatrix} \\ \boldsymbol{V}_{lu} &= \begin{bmatrix} floor\ V_{REF\,g} \\ ceil\ V_{REF\,h} \end{bmatrix} \\ \boldsymbol{V}_{uu} &= \begin{bmatrix} ceil\ V_{REF\,g} \\ ceil\ V_{REF\,h} \end{bmatrix} \\ \boldsymbol{V}_{ll} &= \begin{bmatrix} floor\ V_{REF\,g} \\ floor\ V_{REF\,h} \end{bmatrix} \end{aligned} \tag{2.24}$$

The *floor* function is used to return the greatest integer smaller or equal to the operand; the ceiling or *ceil* function returns the smallest integer greater than or equal to the operand. The reference vector will then fall either into the upper triangle, or into the lower triangle. A general equation for the short diagonal line of the parallelogram can be expressed in terms of g-h coordinates:

$$g + h = V_{ul_g} + V_{ul_h} \,. \tag{2.25}$$

The question of whether the reference vector falls into the upper or lower triangle can be resolved by evaluating the expression

$$V_{REF_g} + V_{REF_h} - V_{ul_g} - V_{ul_h} \,. \tag{2.26}$$

A positive value indicates an upper triangular reference; consequently, $\boldsymbol{V}_{ul}$, $\boldsymbol{V}_{lu}$ and $\boldsymbol{V}_{uu}$ are the three nearest vectors. A negative value shows that the reference is lower triangular and so the nearest three vectors are $\boldsymbol{V}_{ul}$, $\boldsymbol{V}_{lu}$ and $\boldsymbol{V}_{ll}$.

2.6.3.3 DUTY CYCLE AND PHASE VOLTAGE DETERMINATION

The output reference vector can now be expressed as a sum of the three nearest vectors. It can also be expressed in terms of differential vectors:

$$\begin{aligned} \boldsymbol{V}_{REF} &= d_1 \boldsymbol{V}_1 + d_2 \boldsymbol{V}_2 + (1 - d_1 - d_2)\boldsymbol{V}_3 \\ &= d_1 (\boldsymbol{V}_1 - \boldsymbol{V}_3) + d_2 (\boldsymbol{V}_2 - \boldsymbol{V}_3) + \boldsymbol{V}_3 \end{aligned} \tag{2.27}$$

The resulting equation is similar in form to the conventional space-vector decomposition in equation (2.20). This allows for the geometric representation and subsequent determination of the duty cycles d_1, d_2 and d_3, as indicated in Figure 2.9 and Figure 2.10. Such a representation was not used by Celanovic *et al.* in either [37] or [50]. However, the geometric representation in Figure 2.9 and Figure 2.10 assists understanding of the basic vector algebra and clearly identifies the typing error in [50]. It can be seen that d_1 is associated with a horizontal (g-axis) vector in the lower triangular case and with an h-axis vector in the upper triangular case. Similarly, d_2 is related to an h-axis vector in the lower triangular case, whereas d_2 is associated with a g-axis vector in the upper triangular case.

Note that $(\boldsymbol{V}_1 - \boldsymbol{V}_3)$ and $(\boldsymbol{V}_2 - \boldsymbol{V}_3)$ are unit vectors alongside their respective axes because of the specific scaling in equation (2.22). Consequently, for a lower triangular reference as in Figure 2.9, the duty cycles can be computed as

$$\begin{aligned} d_1 &= V_{REF_g} - V_{ll_g} \\ d_2 &= V_{REF_h} - V_{ll_h} \end{aligned} \quad . \tag{2.28}$$

The duty cycles for an upper triangular reference (as in Figure 2.10) are

$$\begin{aligned} d_1 &= V_{uu_h} - V_{REF_h} \\ d_2 &= V_{uu_g} - V_{REF_g} \end{aligned} \quad . \tag{2.29}$$

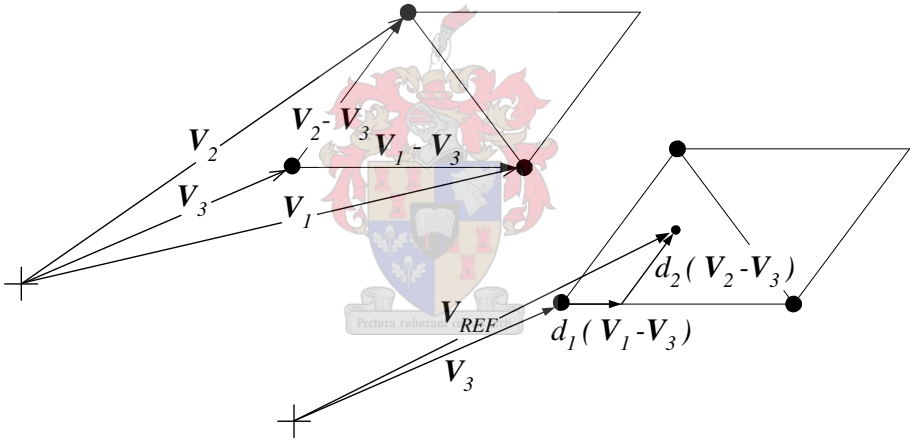


Figure 2.9 *Vector addition for lower triangular reference*

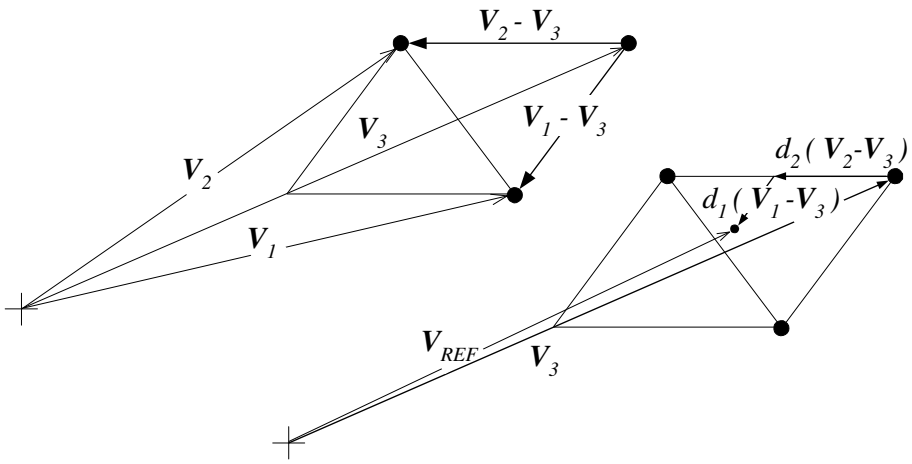


Figure 2.10 *Vector addition for upper triangular reference*

Finally, the converter phase voltages must be determined to select an appropriate output switching state. Through substitution of equation (2.23) into (2.22), the g-h components can be expressed in terms of phase voltages

$$\begin{bmatrix} V_{REF_g} \\ V_{REF_h} \end{bmatrix} = cells \cdot \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix}, \tag{2.30}$$

with the phase voltages having maximum amplitude of 1, and the g-h components scaled to a maximum amplitude of *cells*.

When adding the equation $CM = \frac{1}{3}V_a + \frac{1}{3}V_b + \frac{1}{3}V_c$ to the system, the system of equations can be reversed to obtain

$$\begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} = \frac{1}{cells} \cdot \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & cells \\ -\frac{1}{3} & \frac{1}{3} & cells \\ -\frac{1}{3} & -\frac{2}{3} & cells \end{bmatrix} \begin{bmatrix} V_g \\ V_h \\ CM \end{bmatrix}. \tag{2.31}$$

Valid phase voltages combinations can be obtained by selecting a suitable value of *CM*.

2.7 OTHER COMPENSATION METHODS

Other capacitor stabilisation algorithms that are not directly applicable to the 'two-degrees-of-freedom' approach of subsection 2.6.1 can be found. In [33], a PWM state-feedback controller operating on a linearised, decoupled system model is realised. In [51], a PWM approach with a fuzzy-logic controller based on variable phase shifts between cell control signals is developed. In [52], a modified carrier is proposed as a capacitor stabilisation method. In [23], the difference in duty cycles of adjacent cells (of voltage pulses generated by a hysteresis current controller) is modified in order to control the capacitor voltages.

2.8 SUMMARY

In Chapter 2, relevant concepts such as balance and imbalance have been defined. In addition, an outline of two published strategies that can be classified as state substitution methods, has been given. The space-vector modulation strategy has been set out in detail. In particular, this chapter has illustrated the relevance of the Celanovic *et al.* space-vector algorithm as applied in a multilevel converter as a line-to-line redundancy compensation method.

Chapter 3
Modelling of Voltage Stabilisation Strategies

3.1 INTRODUCTION

This chapter will set out the voltage stabilisation problem and requirements in detail. Here the difficulty is the construction of the Simplorer simulation models. It will be seen, especially in the case of the Donzel and Bornard algorithm, that model construction is particularly problematic. In addition, Simplorer models of the proposed capacitor-voltage stabilisation algorithm and the three-phase vector selection logic will be shown.

3.2 COMPENSATION STRATEGY REQUIREMENTS

In section 2.6.1, the three-step control within the two degrees of freedom in the flying-capacitor topology was set out. The three-phase controller selects the various output voltages: for vector ① in Figure 3.1, for instance, the output voltage levels of the converter phase-legs are 3, 3 and 2 respectively.

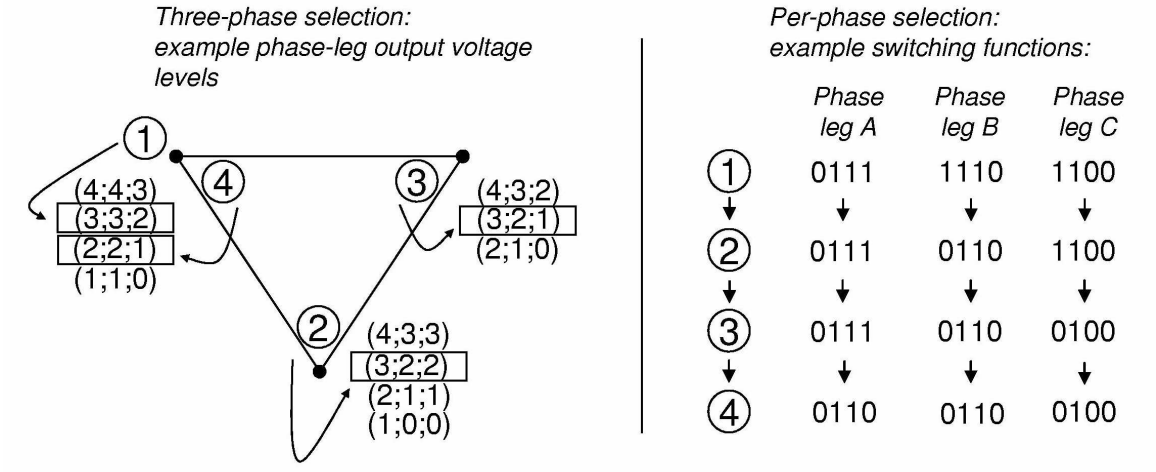


Figure 3.1 *The combined three-phase selections and per-phase selections over a half-carrier equivalent switching period in a 4-cell flying-capacitor converter under space-vector modulation*

There are more than one phase-arm switching state that correspond to an output voltage level of 3. (An explanatory definition of output voltage levels within a five-cell converter topology was shown in Table 2.1.) It is therefore the responsibility of the per-phase controller to select the appropriate switching state – '0111' – in this example. The vector order is specified by the three-phase controller: configuration (3;3;2) is followed by (3;2;2), (3;2;1) and (2;2;1) within the illustrated half-carrier equivalent switching period. This represents the minimum number of level-changes within such a sequence. A sequence of (3;3;2), (1;0;0), (3;2;1) and (2;2;1) does not minimise the changes in output levels. Although the level changes are minimised by the three-phase controller, the minimum switch commutations are still controlled by the per-phase selection process, as there are generally more than one switching state for every voltage output level. This means that the per-phase (state substitution based) controller must take current switching functions into account, when selecting a new switching function to minimise switch commutations but also control the capacitor voltages.

Reference will be made in this thesis to a 'Part A' and a 'Part B' of a state substitution (per-phase) control algorithm. The 'Part A' will refer to all parts of the per-phase control that is concerned with, or dependent on the current switching function (for the per-phase minimisation of switch commutations). The 'Part B' will refer to all parts of the per-phase control that is concerned with, or dependent on the cell capacitors voltage measurements.

3.3 CIRCUIT REQUIREMENT

The multilevel converter that is considered for the medium-voltage active power filter application, is shown in Figure 3.2.

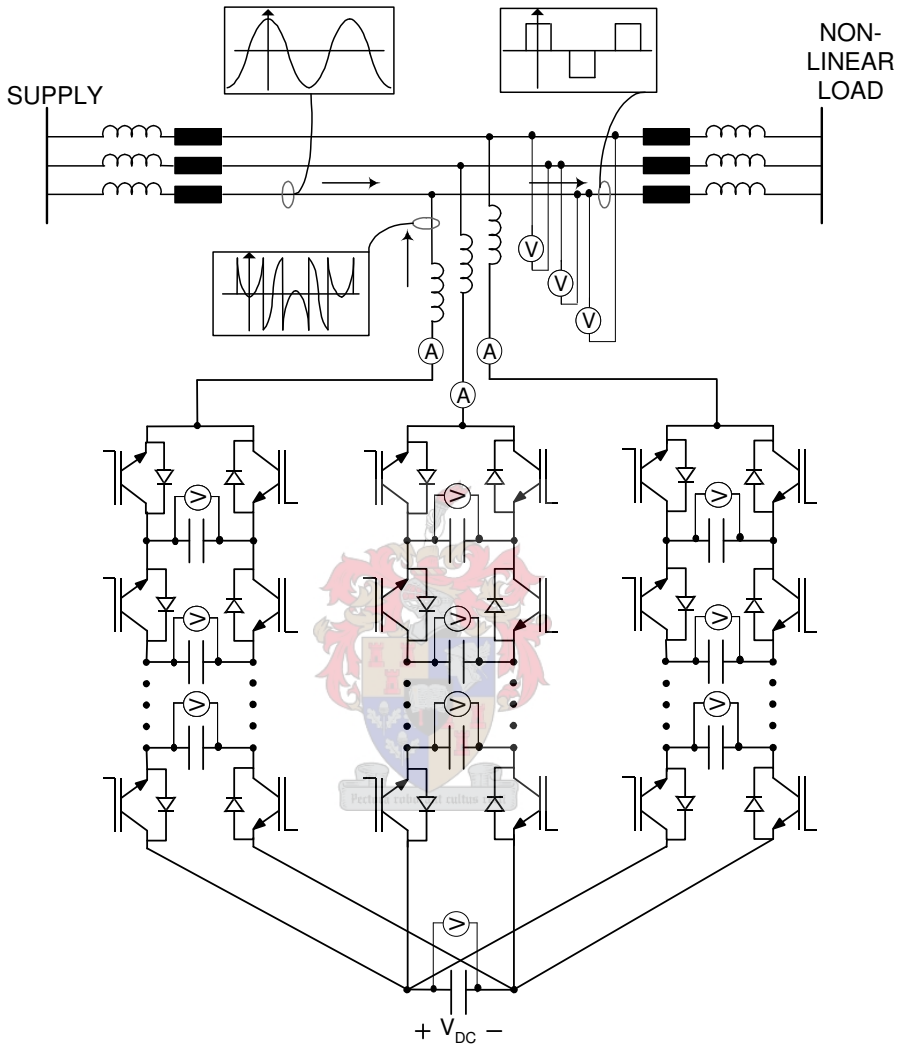


Figure 3.2 *The medium-voltage active power filter configuration. The elimination of the supply current harmonics by the active power filter is illustrated through the idealised current waveforms shown.*

In general, the switching frequency of a high-voltage high-power converter utilising IGBTs should be no higher than 1 kHz, in order to limit the switching power losses in the system [49]. Comparable to the active power filter of this thesis, is the flying-capacitor converter-based power line conditioner of [24] that operates on a 6,9 kV system with a 1 kHz switching frequency. Accordingly, a switching frequency of 1 kHz will be used for the converter in this thesis.

The load voltage amplitude should not exceed 70% of the converter DC bus voltage, in order to have sufficient control over the injected current when compensating for reactive currents [22]. In agreement with this guideline, a DC bus voltage of 800 V is often used for shunt active filters operating on a 400V three-phase system; hence, a DC bus value of 22 kV will be used for an active filter in an 11 kV three-phase system.

Ideally, the required number of cells can be obtained by means of a division of the DC bus value by the IGBT (maximum) blocking voltage. However, there are some practical issues to be considered, such as parasitic circuit inductance and consequent voltage overshoot. The flying-capacitor topology is a non-standard circuit in terms of IGBT modules. Measures that have been taken in one specific instance to minimise circuit inductance in the flying-capacitor topology, is described in [6]. A safety scaling factor of 1,8 will be used to accommodate such overshoot in the subsequent calculations. Currently, the maximum blocking voltage of commercially available IGBTs stands at 6.5 kV [53]. Therefore, the required number of cells for a 11 kV three-phase system with 22 kV DC bus is

$$ceil\ (\ 22\ 000\ /\ (6\ 500\ /\ 1{,}8)\ ) = 7$$

with the *ceil* or *ceiling* function as defined in subsection 2.6.3.2.

Consequently, all calculations and reasoning in this thesis will be done with a 7-cell converter in mind.

IGBTs with blocking voltages in the range 4.5 kV to 6.5 kV typically have turn-off delay times of 6 µs [58, 59]. Switch blanking times of about 7 µs will therefore be required.

3.4 TIMING REQUIREMENTS

The uncertainty regarding the length of time available for state substitution can be reduced by setting up a timing budget for the medium-voltage active power filtering application.

A certain length of time has to be allocated for each of the following tasks within a specific control time-interval:

- Sampling of signals
- A/D conversion time
- Time to send data via optical fibre to the PEC33 board
- Time to calculate reference values
- Time to calculate converter output voltage levels in a SVM algorithm
- Time for state substitution algorithm
- Time for output of SVM-calculated output voltage levels and the activation thereof

Together, these time durations form a timing budget for the medium voltage active filter application and govern the choice of the specific software/hardware implementation of the state substitution algorithm. These time durations will be investigated in detail in subsection 4.3, where the specific hardware/software choices will be set out.

3.5 SIMULATION ENVIRONMENT

It may be said that the Simplorer simulation environment, although extremely powerful, is not entirely suited to the simulation models that are implemented. At the start of this project, the available options were Simplorer 4.2, Simulink/Matlab and PSim. Instabilities of the PSim environment were discovered when using the C/C++ interface. Simulink models typically require a Laplace representation. A Matlab-only simulation would require finding solutions to the system differential equations [60]; consequently Simplorer was used. As the C-interface of Simplorer 4.2 was problematic also, it was decided to make extensive use of the state-graph module in Simplorer. It is interesting to note that some other parties also investigated implementation possibilities of power electronics control circuits in environments not originally intended for the application. A C-interface was realised in PSpice by exploiting a C-based MOSFET (four-node) model description [61]. The model convergence information that must be supplied on demand by such a C-based model, is used to control the time step of an event-driven (asynchronous) controller simulation. Presumably, the C-interface of PSIM is much improved today, as a paper on the use thereof has been published very recently [62].

3.6 A PER-PHASE MODEL

First, a look will be taken at the requirements of a per-phase compensation strategy. Here, models will be built up for both a cell-voltage based controller and a capacitor-voltage based controller.

3.6.1 MODELLING OF THE DONZEL AND BORNARD ALGORITHM

The cell-voltage based control algorithm of Donzel and Bornard [34] set out in subsection 2.3.1 will now be implemented in the Simplorer simulation environment.

3.6.1.1 SORTING OF ABSOLUTE CELL VOLTAGE ERRORS

The absolute cell voltage errors are sorted in descending order for positive output current and in ascending order for negative output current.

```
temp  :=  5*((a5>a4)and(a5>a3)and(a5>a2)and(a5>a1)) +
    4*((a4>a5) and (a4>a3) and (a4>a2) and (a4>a1)) +
    3*((a3>a5) and (a3>a4) and (a3>a2) and (a3>a1)) +
    2*((a2>a5) and (a2>a4) and (a2>a3) and (a2>a1)) +
      ((a1>a5) and (a1>a4) and (a1>a3) and (a1>a2))
```

Table 3.1 *Partial sorting algorithm utilised in the Simplorer realisation of the Donzel and Bornard algorithm*

The cell error voltages are represented by *a* in the fragment above. Once the largest voltage is determined, the corresponding *a* variable is set to '-1'. As all the absolute error voltages are positive, this allows for the repetitive determination of the maximum value, until a sorted list of error voltages is obtained. The indices of the cell voltage errors are stored in sorted order in another vector, *g*. The first element in the *g* vector holds the index of the largest (absolute) cell voltage; the second element in the *g* vector is assigned the index of the second largest absolute cell voltage, et cetera.

3.6.1.2 GENERATION OF A AND B VECTORS

The first n_{free} cell error voltages (of which the indices are stored in the sorted vector g), are copied into the A vector.

Extensive use is made of zero-one constructs[1] in the Simplorer simulation models, to reduce the number of state graph elements that are required. In Figure 3.1 for example, only one value is allocated to the first element in the A vector (variable A_1), dependent on the Boolean values of the expressions ($g1$ = 1), ($g1$ = 2) et cetera. Undesired values are then zeroed-out, as they do not contribute to the sum that is copied to A_1.

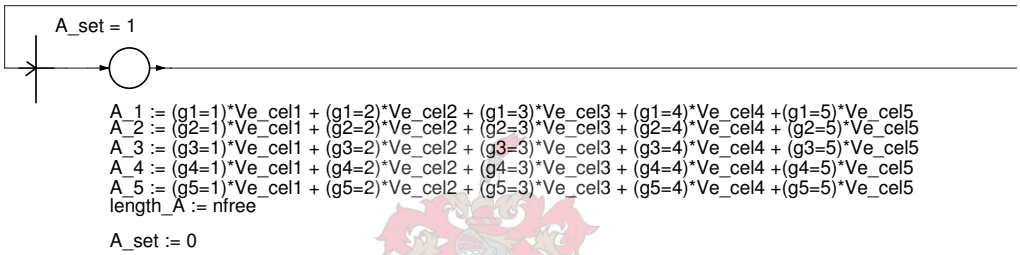


Figure 3.2 *Specification of the A-vector*

The values of the switching functions selected at the previous switching instant of the remaining elements (that are not referenced by vector A) are copied into the B vector. The entries of the B vector (which corresponds to the $(n_{free}+1)^{th}$ entry in the g vector) cannot be hard-coded in one single state, as the value of n_{free} is variable. Consequently, the allocation of the B vector is hard-coded for every possible value of n_{free}. For example, in Figure 3.2, when the n_{free} parameter has a value of '3', switching functions corresponding to the fourth largest and fifth largest cell voltages are copied, as the other three items have been handled by the A vector allocation process.

[1] Zero-one construct: a concept borrowed from MATLAB, to implement fast and efficient IF structures

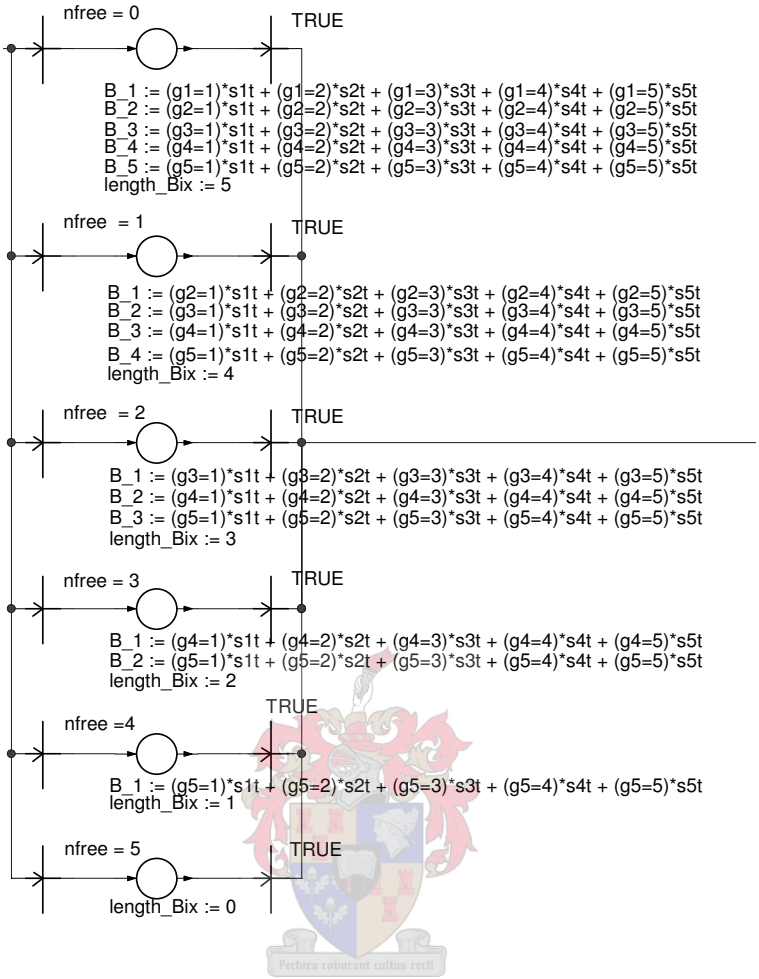


Figure 3.3 *Specification of the B-vector*

The process that is followed to select the positive elements from vector A into vector A_p, is quite similar to the process followed to copy the negative elements in A to A_n. Consequently, only the A_p generation process is shown in Figure 3.4. Here, the variable *p_count* keeps track of the number of elements that has been copied into the A_p vector already. As a result the selected indices can be copied into variable, but sequential positions of the A_p vector. Entries in the A vector are checked sequentially for grouping into the A_p and A_n, vectors, starting at the first entry (which corresponds to the largest absolute cell voltage error). Consequently, the state machine logic can be hard-coded to start the sequential check at the first element in the g vector.

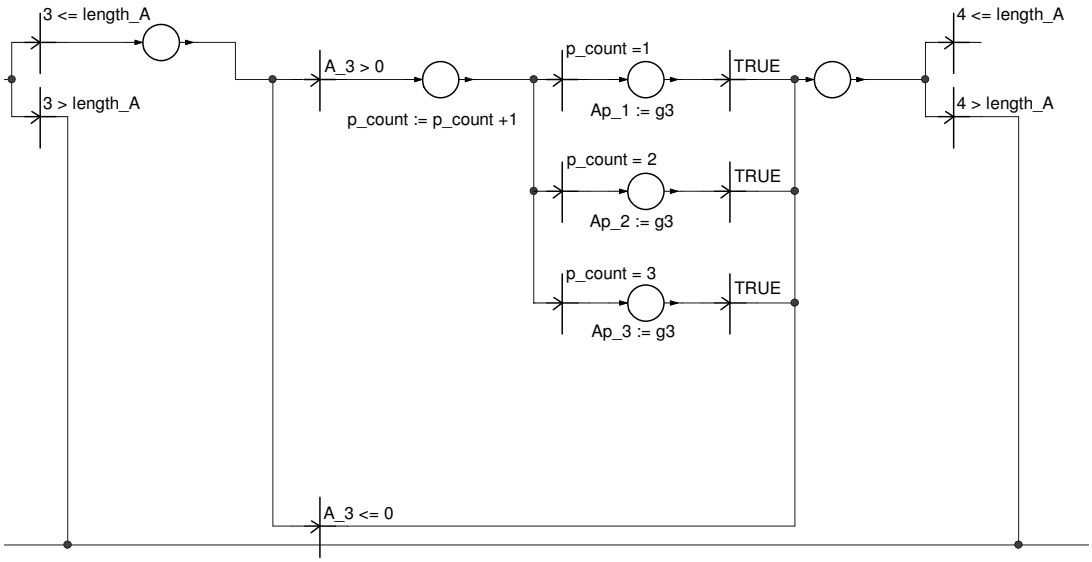


Figure 3.4 *Copying of third element (g3) in the sorted list of cell error voltages. Note that the Ap vector index is variable.*

The assignment of the B_z and the B_o vectors requires a more intricate logic, however. As the first entry in the B vector is the $(n_{free}+1)^{th}$ entry in the sorted list of absolute cell error voltages, special care is needed to ensure that the correct cell index is copied into the B_o and B_z vectors. A two-step process is implemented as both the starting index in the sorted list of cell error voltages (the source index) and the index of the destination vector, is variable.

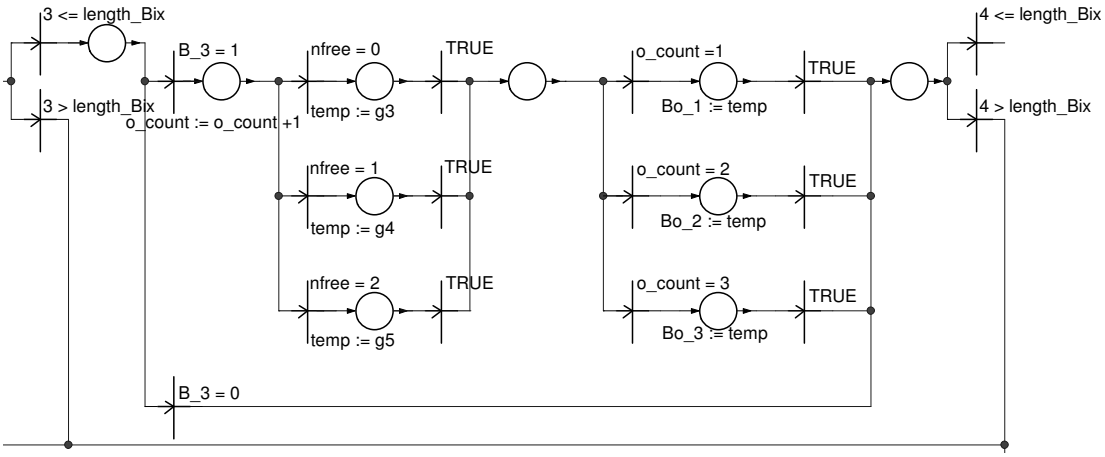


Figure 3.5 *Two-step process in the copying of the cell error voltage investigated third-most in the Bo-vector specification. Note that both the Bo-vector index and the index into the sorted list of cell error voltages (g-vector) are variable.*

Again, only the B_o vector specification is shown in Figure 3.5 as two similar processes are implemented for the generation of the B_o and the B_z vectors.

It should be noted that the A vector stores absolute cell voltage errors and the B vector stores switching function values; the A_p, A_n, B_o and B_z vectors store indices of cell error voltages.

3.6.1.3 SORTING OF CELL VOLTAGE ERRORS

Once the A_p, A_n, B_o and B_z vectors have been generated, they are sorted according to their corresponding cell error voltages

The C vector is formed by concatenation: $C = [A_p\ B_o\ B_z\ A_n]$ for positive output current, and $C = [A_n\ B_o\ B_z\ A_p]$ for negative output current.

Again the copying process is executed in two steps as the specific indices involved are variable. First, a value is copied from the source, with variable index count, to a temporary variable; then the temporary value is copied to the destination, with variable index count. Four loops are implemented, with loop counters keeping track of the index into the A_p, B_o, B_z and A_n vectors. Another counter stores the length of the destination vector (C vector).

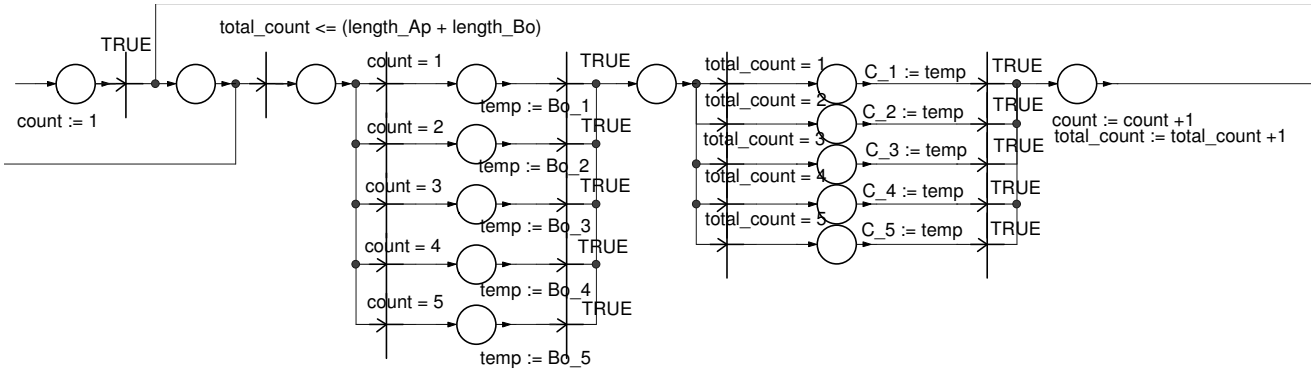


Figure 3.6 *Two-step concatenation process: a temporary variable stores the value indexed by the loop variable (source index) count. This value is copied to the C vector, which is indexed by the destination index, total_count.*

3.6.1.4 SWITCHING FUNCTION DETERMINATION

The switching functions are allocated by generating a temporary *D* vector (refer to Figure 3.7). The first n entries are set to '1' - representing the switching functions as is referenced (in the specified order) in vector *C*. For example, the switching function referenced by the first element in the *C* vector is assigned the value of the first element in the *D* vector. The correct switching function is then specified with the use of a zero-one vector construct.

3.6.1.5 SWITCHING FREQUENCY REGULATION

The switching frequency is regulated by means of the n_{free} parameter, as it specifies the maximum allowable number of switch commutations during a switching instant.

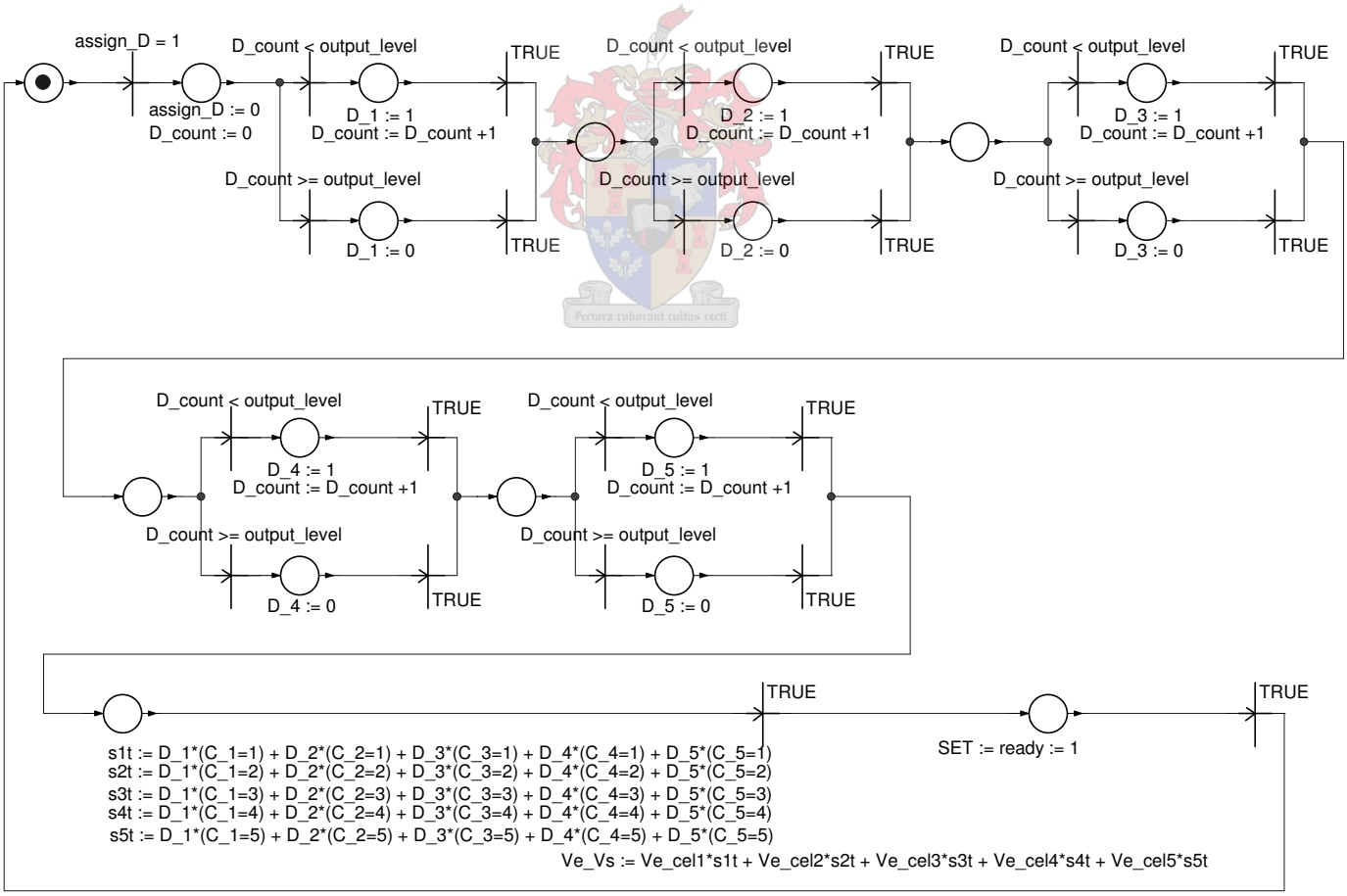
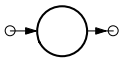


Figure 3.7 *Allocation of the switching function values. The first n entries of the D vector are set to '1'; then the switching functions are set by a zero-one construct.*

3.6.2 MODELLING OF THE DERIVED CAPACITOR-VOLTAGE CONTROLLER

The proposed stabilisation algorithm as set out in subsection 2.5 will now be modelled in Simplorer. Sequential parts of the model for the capacitor-voltage based algorithm are shown in Figure 3.8 to Figure 3.16. Only the first few instructions are shown in the graphic representations, for simplicity.

First the cell correspondence value calculations, according to the definition in Table 2.1, are shown in Figure 3.8. The various constant, or initial value declarations, as required by the Simplorer model, is shown in Figure 3.9.



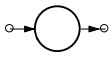
r_A_c1:= ((A_c1_ref = cap_c1_ref) and (A_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_B_c1:= ((B_c1_ref = cap_c1_ref) and (B_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_C_c1:= ((C_c1_ref = cap_c1_ref) and (C_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_D_c1:= ((D_c1_ref = cap_c1_ref) and (D_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_E_c1:= ((E_c1_ref = cap_c1_ref) and (E_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_F_c1:= ((F_c1_ref = cap_c1_ref) and (F_c1_ref <> 2) or (cap_c1_ref = 2))*r_r

Figure 3.8 *Test for 'correct' cell correspondence and partial weight allocation*

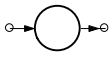
ICA :	ICA :	ICA :	ICA :	ICA :
A5 := 1	A4 := 1	A3 := 1	A2 := 1	A1 := 0
B5 := 1	B4 := 1	B3 := 1	B2 := 0	B1 := 1
C5 := 1	C4 := 1	C3 := 0	C2 := 1	C1 := 1
D5 := 1	D4 := 0	D3 := 1	D2 := 1	D1 := 1
E5 := 0	E4 := 1	E3 := 1	E2 := 1	E1 := 1
F5 := 1	F4 := 1	F3 := 1	F2 := 0	F1 := 0

C1_charge_ref ICA :	C2_charge_ref ICA :	C3_charge_ref ICA :	C4_charge_ref ICA :	VA1 :
A_c1_ref := 1	A_c2_ref := 2	A_c3_ref := 2	A_c4_ref := 2	A_level := A5 + A4 + A3 + A2 + A1
B_c1_ref := 0	B_c2_ref := 1	B_c3_ref := 2	B_c4_ref := 2	B_level := B5 + B4 + B3 + B2 + B1
C_c1_ref := 2	C_c2_ref := 0	C_c3_ref := 1	C_c4_ref := 2	C_level := C5 + C4 + C3 + C2 + C1
D_c1_ref := 2	D_c2_ref := 2	D_c3_ref := 0	D_c4_ref := 1	D_level := D5 + D4 + D3 + D2 + D1
E_c1_ref := 2	E_c2_ref := 2	E_c3_ref := 2	E_c4_ref := 0	E_level := E5 + E4 + E3 + E2 + E1
F_c1_ref := 2	F_c2_ref := 1	F_c3_ref := 2	F_c4_ref := 2	F_level := F5 + F4 + F3 + F2 + F1

Figure 3.9 *Definition of switching functions, charge/discharge characteristics and output levels – per switching state*



```
r_p1 := (1*(Vd1_r=0) + 2*(Vd1_r=1) + 4*(Vd1_r=2) + 8*(Vd1_r=3))
r_p2 := (1*(Vd2_r=0) + 2*(Vd2_r=1) + 4*(Vd2_r=2) + 8*(Vd2_r=3))
r_p3 := (1*(Vd3_r=0) + 2*(Vd3_r=1) + 4*(Vd3_r=2) + 8*(Vd3_r=3))
r_p4 := (1*(Vd4_r=0) + 2*(Vd4_r=1) + 4*(Vd4_r=2) + 8*(Vd4_r=3))
```

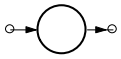


```
w_p1 := (16*(Vd1_r=0) + 32*(Vd1_r=1) + 64*(Vd1_r=2) + 128*(Vd1_r=3))
w_p2 := (16*(Vd2_r=0) + 32*(Vd2_r=1) + 64*(Vd2_r=2) + 128*(Vd2_r=3))
w_p3 := (16*(Vd3_r=0) + 32*(Vd3_r=1) + 64*(Vd3_r=2) + 128*(Vd3_r=3))
w_p4 := (16*(Vd4_r=0) + 32*(Vd4_r=1) + 64*(Vd4_r=2) + 128*(Vd4_r=3))
```

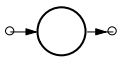
Figure *3.10* *Zero-one construct for cell capacitor multiplier determination*

Figure 3.10 shows the calculation of weights, which is combined in Figure 3.11 to form the ratings of the available switching states, in accordance with Table 3.5.

Note that any fragment that can be written as a zero-one construct can be realised by means of a multiplexer in an FPGA.

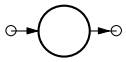


```
r_A_rating := r_A_c1*r_p1 + r_A_c2*r_p2 + r_A_c3*r_p3 + r_A_c4*r_p4
r_B_rating := r_B_c1*r_p1 + r_B_c2*r_p2 + r_B_c3*r_p3 + r_B_c4*r_p4
r_C_rating := r_C_c1*r_p1 + r_C_c2*r_p2 + r_C_c3*r_p3 + r_C_c4*r_p4
r_D_rating := r_D_c1*r_p1 + r_D_c2*r_p2 + r_D_c3*r_p3 + r_D_c4*r_p4
r_E_rating := r_E_c1*r_p1 + r_E_c2*r_p2 + r_E_c3*r_p3 + r_E_c4*r_p4
r_F_rating := r_F_c1*r_p1 + r_F_c2*r_p2 + r_F_c3*r_p3 + r_F_c4*r_p4
```



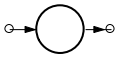
```
w_A_rating := w_A_c1*w_p1 + w_A_c2*w_p2 + w_A_c3*w_p3 + w_A_c4*w_p4
w_B_rating := w_B_c1*w_p1 + w_B_c2*w_p2 + w_B_c3*w_p3 + w_B_c4*w_p4
w_C_rating := w_C_c1*w_p1 + w_C_c2*w_p2 + w_C_c3*w_p3 + w_C_c4*w_p4
w_D_rating := w_D_c1*w_p1 + w_D_c2*w_p2 + w_D_c3*w_p3 + w_D_c4*w_p4
w_E_rating := w_E_c1*w_p1 + w_E_c2*w_p2 + w_E_c3*w_p3 + w_E_c4*w_p4
w_F_rating := w_F_c1*w_p1 + w_F_c2*w_p2 + w_F_c3*w_p3 + w_F_c4*w_p4
```

Figure 3.11 *Allocation of most significant and least significant halves of state ratings*



A_rating_final1 := (r_A_rating + w_A_rating) * (A_level = current_level)
B_rating_final1 := (r_B_rating + w_B_rating) * (B_level = current_level)
C_rating_final1 := (r_C_rating + w_C_rating) * (C_level = current_level)
D_rating_final1 := (r_D_rating +w_D_rating) * (D_level = current_level)
E_rating_final1 := (r_E_rating + w_E_rating) * (E_level = current_level)
F_rating_final1 := (r_F_rating + w_F_rating) * (F_level = current_level)

Figure 3.12 *Zeroing of unsuitable state ratings*



A_rating_final := A_rating_final1 * ((A_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
B_rating_final := B_rating_final1 * ((B_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
C_rating_final := C_rating_final1 * ((C_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
D_rating_final := D_rating_final1 * ((D_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
E_rating_final := E_rating_final1 * ((E_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
F_rating_final := F_rating_final1 * ((F_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))

Figure 3.13 *Zero-one construct to eliminate successor states with non-minimum switching commutations*

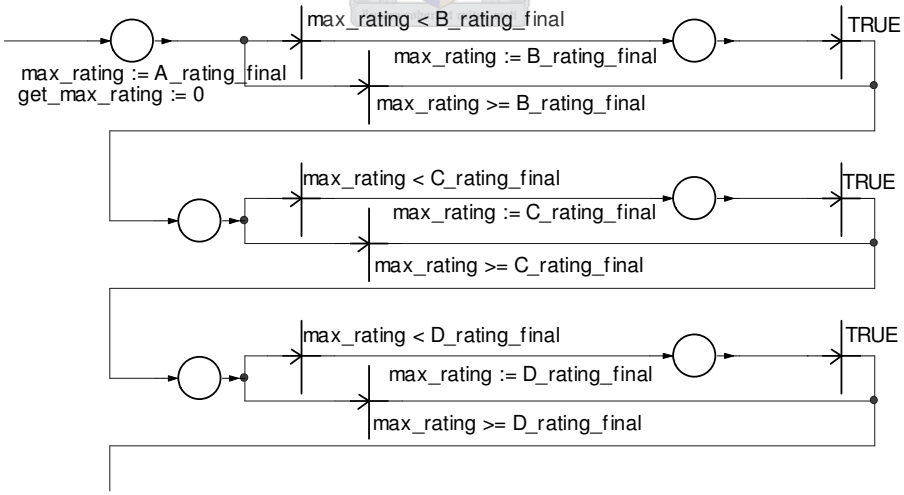
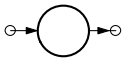


Figure 3.14 *Acquirement of maximum state rating*

Figures 3.12 and 3.13 show the elimination process of unsuitable switching states. In Figure 3.14 the maximum state rating is determined. In Figures 3.15 and 3.16, the output switching functions that correspond to this maximum rating, are selected.



Ostate_A := (max_rating = A_rating_final)
Ostate_B := (max_rating = B_rating_final)
Ostate_C := (max_rating = C_rating_final)
Ostate_D := (max_rating = D_rating_final)
Ostate_E := (max_rating = E_rating_final)
Ostate_F := (max_rating = F_rating_final)

Figure 3.15 *Identification of output state*

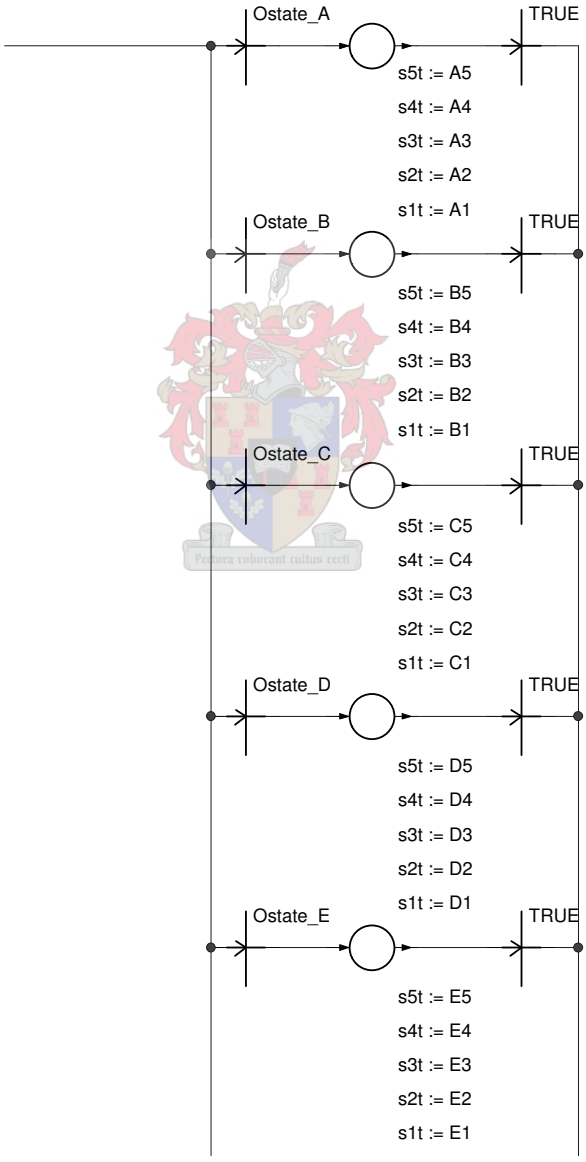


Figure 3.16 *Specification of output switching functions*

3.7 A THREE-PHASE MODEL

Switching frequency minimisation is not achieved by the per-phase control strategy alone. The number of switch commutations are minimised by the per-phase control strategy for a specified reference converter output voltage level; however, the series of three-phase successor states that will minimise the total change in phase-leg output voltages is chosen by the three-phase SVM strategy. This is in accordance with the three-step approach described in subsection 2.6.1.

The algorithm of Celanovic *et al.* is used to determine the three nearest vectors in the space vector plane. The realisation thereof is shown in Figure 3.17, in accordance with equation (2.24).

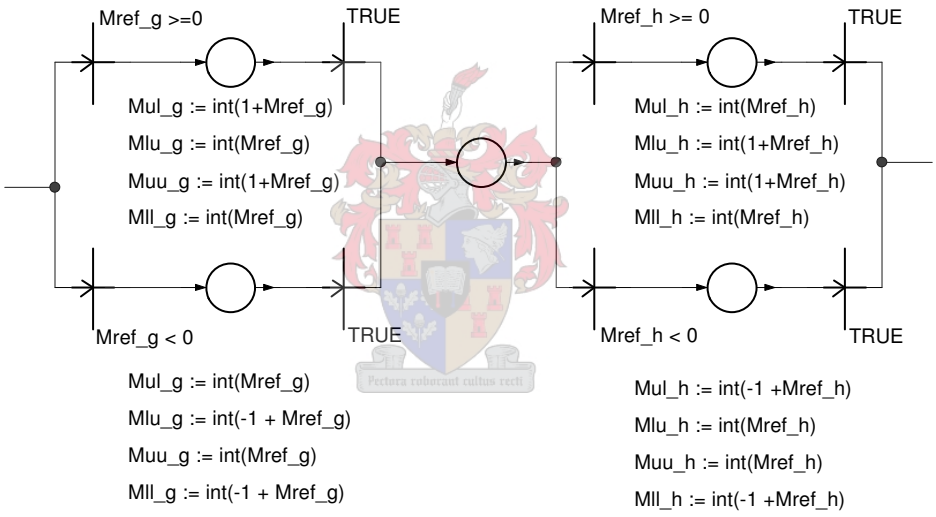


Figure 3.17 *Determination of the three space vectors nearest to the reference output vector*

In Figure 3.18 and Figure 3.19, the identification of the basic three-phase combination and also, the other three-phase redundant state combinations are done.

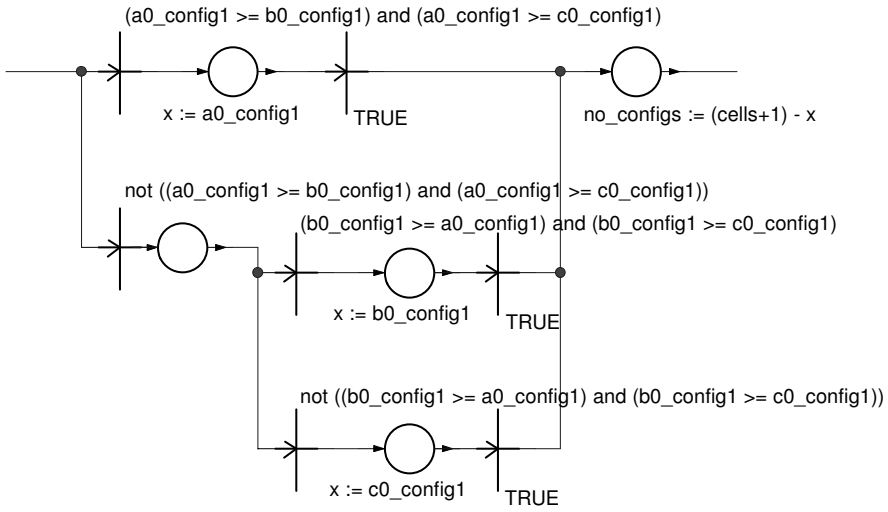


Figure 3.18 *Determination of the first three-phase redundant combination*

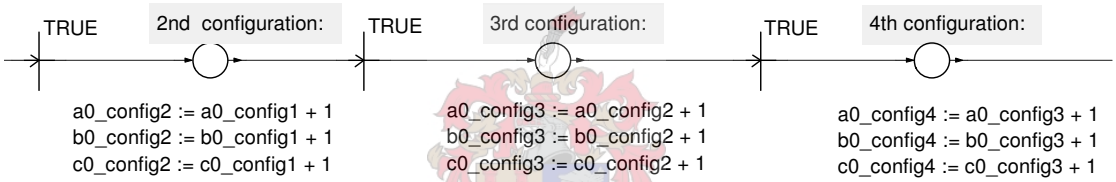


Figure 3.19 *Calculation of remaining three-phase redundant configurations*

In conventional (non-multilevel) converters, SVM is often implemented by changing the space-vector duty cycles into pulsewidth-modulation duty cycles or modulation indices on a per-phase basis. Such a technique is used specifically to generate the phase-leg output voltages for a known space-vector voltage combination. There are several problems with the application of this technique for the flying-capacitor topology. The specification of a modulation index for a phase-leg, will not guarantee the selection of a particular per-phase redundant state. In addition, the space vector order for such a scheme will be fixed – an undesirable property in general. Most importantly, such a scheme would not take capacitor charge/discharge requirements into account; the use of such a scheme would defeat the per-phase balancing purpose.

3.8 SUMMARY

The modelling of the proposed state substitution method and the compensation strategies in Simplorer has been described in detail; digital implementation hereof will follow in Chapter 4.

Chapter 4
State Substitution Controller Design

4.1 INTRODUCTION

Chapter 4 will focus on the digital implementation options of the state substitution controller. It will be shown that the Donzel and Bornard algorithm should allow a DSP-only implementation and that the C-code listed in Appendix A should be sufficient for implementation. However, it will be seen that a mixed DSP-and-FPGA solution would be required for the proposed capacitor-voltage stabilisation algorithm. This chapter gives an overview of the choices that were made and the hardware implementation hereof.

4.2 DSPS AND FPGAS AS ALGORITHMIC SOLUTIONS

The objective of this thesis is to develop a voltage stabilisation algorithm that can be implemented on an existing DSP (TMS320VC33) and FPGA (EP1K50QC208) system. The choice of task distribution between the DSP and FPGA is not straightforward. Therefore, before this issue is addressed, a look will be taken at current trends in DSP and FPGA solutions.

The performance of high-end DSPs has progressed at a phenomenal rate. Currently, industry's fastest DSPs are the C64x generation, with clock rates up to 1 GHz [64]. Perhaps the greatest strength of DSPs is its relative ease of development.

Traditionally, FPGAs were never used for DSP tasks, as they lacked the required gate capacity and did not have suitable tools support [65]. However, both Altera and Xilinx now offer multipliers embedded in their newer FPGAs, with the aim to implement DSP functions on FPGA. Altera offers intellectual property libraries for common DSP functions, tools and development platforms – including a C-code based design flow for programmable logic – and their development environment can now interface with Simulink [65, 66]. Third-party company Berkeley Design Technology, Inc. (BDTI) offers benchmarks for FPGA

performance compared to DSP performance [65]. BDTI benchmarks of Texas Instruments DSPs are available on the Texas Instruments internet site [67]; hence, ironically, the reader is assured of the credibility and authority of BDTI.

There are many advantages to DSPs: the development of DSP code is generally easier than the development of FPGA code; DSPs are more power-efficient in general than FPGAs. Both DSP and FPGA supporters agree that FPGAs should be used as a co-processor when a two-chip (DSP and FPGA) solution is feasible [68].

There are now many parties that have published some thoughts on when to use DSPs, when to use FPGAs, when to use a combination and how the task distribution should be determined. Although the target FPGA is not DSP-oriented, we will use this information to show that the solution proposed in this chapter, although justified in terms of the specific problem requirements, is also justified in terms of current ideas on DSP/FPGA solutions.

Objectives of controller design may be to optimise cost, time-to-market and performance; in our case, it is to optimise performance of an existing system that is applied to a new control application. In general, more than one optimal solution can be identified. The flexibility and performance characteristics of various DSP/FPGA solutions according to Altera Corporation [68] are shown in Figure 4.1.

FPGAs, however, are ideal for computational-intensive algorithms that can be performed in parallel [68]. DSPs have fixed architectures (except, the rather pricey configurable DSPs) and, with their *fetch-compute-store* operation, do not have the computational efficiency inherent to FPGAs [68, 69]. Applications that may have an improved performance through FPGA implementation have several levels of parallelism: multiple channels are likely, similar operations are performed in each channel and, within each operation, individual calculations are performed repeatedly [68].

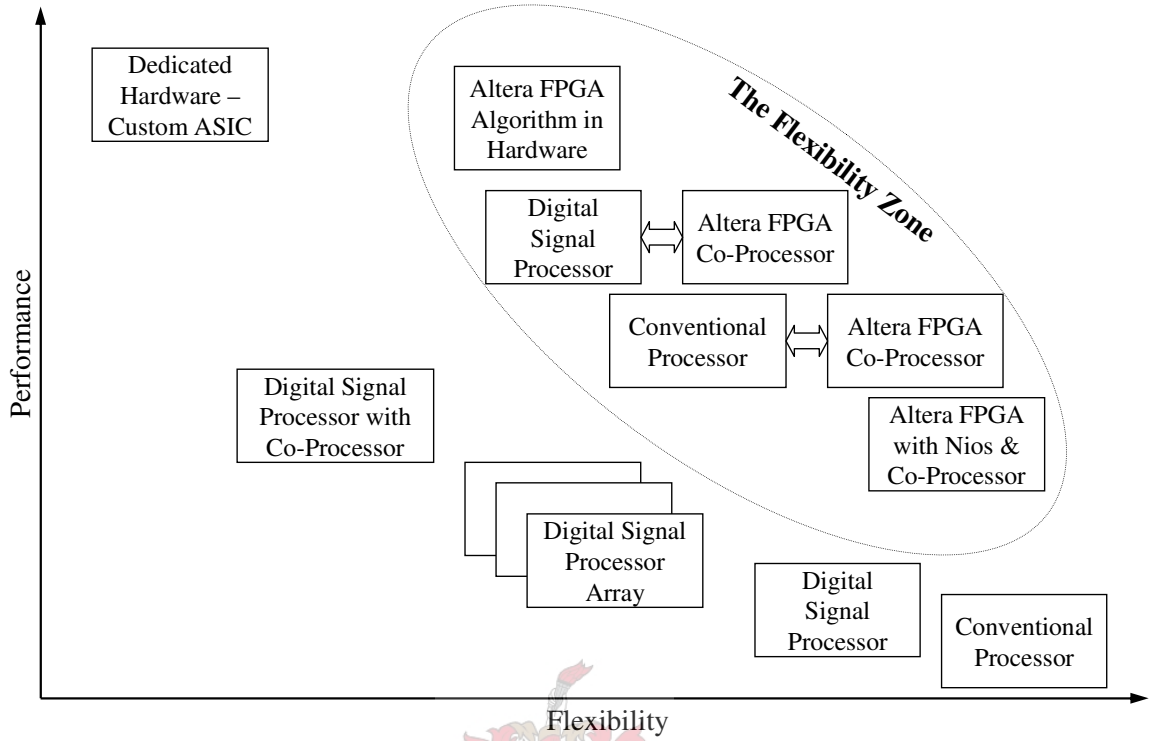


Figure 4.1 *Performance vs. flexibility, according to Altera Corporation [68]*

In power-electronics applications, a slight shift in the use of FPGAs can also be detected. In [70], a deadbeat controller is realised by means of an FPGA. Here, calculations and measurements are performed within 1,15 µs, so measurements are taken and the associated control output generated within the IGBT switching dead-time.

In [7], an FPGA-based estimator is presented as an alternative to the use of expensive sensors to measure capacitor voltages in the flying-capacitor multilevel topology. Here, it is stated that, in most cases, a DSP associated with an FPGA gives the optimal solution; the DSP is generally used for complex calculations and the FPGA for calculations requiring a small latency and computation time. However, the conclusion reached in [7], is that an FPGA-only solution is sufficient for the estimator design, as the number of gates in a single chip allows for the implementation of complex functions. Figure 4.2 shows the old and new task distribution, as is realised for the estimator design.

Somewhat closer to the application of this thesis, is the direct-torque-control imbalance compensator of a flying-capacitor converter of Martins, Roboam, Meynard and Carvalho [35] discussed in subsection 2.3.2. Here the high-level space-vector control was implemented in a DSP; the balancing algorithm was implemented in an FPGA.

Another application of FPGA-based control, is ASIC (Application-specific integrated circuit) design. FPGAs can be used to generate prototypes of hard-mask versions. Again, this reasoning has validity for power-electronics control applications. In [45], a SVM algorithm is realised in an FPGA-form, to be implemented as an ASIC.

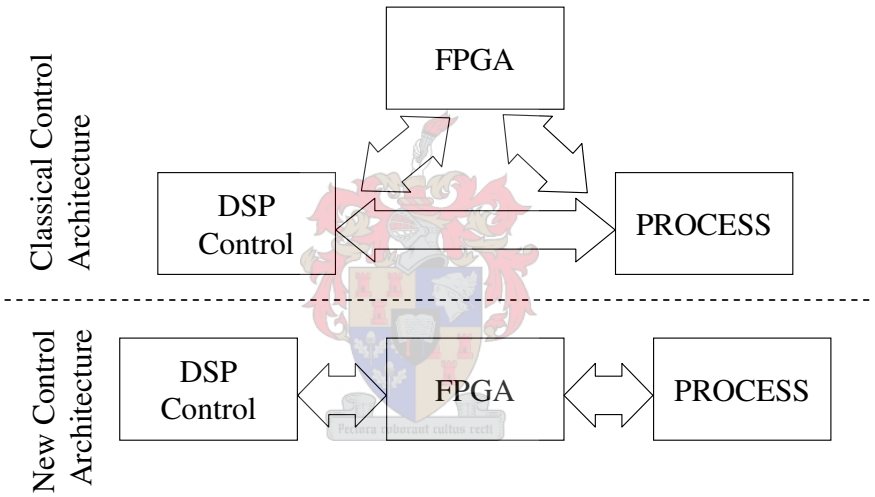


Figure 4.2 *Old (top) and new (bottom) duty distribution of the emulator architecture in [7]*

Perhaps the essence of current thoughts on FPGA usage can be obtained from Altera Corporation's online documentation: in the light of their recent advances in hardware capabilities and development tools, high-end FPGAs can be used stand-alone or as co-processor for demanding DSP applications; when used as co-processor it can accelerate performance-critical functions and increase overall system performance [66].

The target FPGA in this thesis (EP1K50QC208) is not DSP-oriented. However, it will be seen that the workload allocation of the FPGA co-processor is justified in terms of realisable options; in addition, it is justified in terms of the level of performance that will be obtained.

4.3 TIMING ANALYSIS OF THE VOLTAGE STABILISATION ALGORITHM

A certain length of time has to be allocated for each of the following tasks within a specific control time-interval of a state substitution controller.

- Sampling of signals

  The minimum time interval between capacitor voltage samples is fixed by the sum of the total time required for A/D conversion of all measurements and the total time required for optical transmission; a lower sampling rate may be used, however. The measurements must be isolated from the controller, for an active filter operating on an 11 kV system. Generally, the isolation prevents noise coupling and grounding problems [54]. For the flying-capacitor topology, differential voltage sensors are usually used to measure the flying capacitor voltages [33]. These differential measurements must be isolated in order to retain the intended functionality of the converter topology.

- A/D conversion time

  From Figure 3.2 *it can be seen that 25 measurements must be done for a 7-cell converter. When the outputs of the current sensors are measured via A/D, a maximum time duration, from a timing budget viewpoint, can be expected. The timing characteristics of the PEC33 on-board A/D converters will be used for a worst-case timing figure. Faster A/D converters are generally available and may be used for the measurements of the 11 kV system instead. However, a scale model realisation would be likely to utilise the on-board A/D converters. The PEC33 board has four TLV1570 8-channel, 10-bit serial analogue-to-digital converters (ADCs). The minimum time required for one (consecutive) conversion is 16 clock cycles [55], or 213,33 ns. The 25 measurements will mean that one A/D will have to perform a maximum of 7 measurements. Therefore, the conversion time – without overheads – can be estimated as 1,49 µs.*

- Time to send data via optical fibre to the PEC33 board

  Standard optical fibre systems operating at 10 MBd will result in 1,6 µs transmission time, for the transmission of 12-bit data with 4 additional control bits. The total transmission time, for 25 measurements, is 40 µs.

- Time to calculate reference values

  The time can be taken as a variable, $T_{refCalc}$; this allows for inequality (4.1) to be set up.

- Time to calculate converter output voltage levels in a SVM algorithm

  A measurement of the required time for an optimised DSP computation is shown on page 112 to be 9,53 µs.

- Time for state substitution algorithm

  This is the only part of the timing requirement specification that is dependent on the state substitution algorithm itself. Both the Donzel and Bornard algorithm and the proposed capacitor-voltage algorithm can be split into 'Part A' and 'Part B' in general. The specifics regarding the subdivision were set out in subsection 3.2. An estimation of the worst-case timing requirement for the 'part A' of the proposed state substitution algorithm is listed, for optimised DSP code, in Appendix A as 233,71 µs; the corresponding 'Part B' time restriction is listed as 2,88 ms. An estimation of the worst-case timing requirement for the 'part A' of the Donzel and Bornard algorithm is listed, for optimised DSP code, in Appendix A as 4,44 µs; the corresponding 'Part B' time restriction is listed as 8,68 µs. As was explained in section 3.2, the 'Part A' of the relevant state substitution algorithms must be performed 7 times during one switching period. This means that the worst-case, optimised (DSP-based) timing requirement of the proposed algorithm is 4,51 ms within one switching period; the total of the Donzel and Bornard algorithm is 39,76 µs.

  The time restriction of the state substitution algorithm will be listed as the variable $T_{stateSubstitution}$ in inequality (4.1).

- Time for output of SVM-calculated output voltage levels and the activation thereof

  The time required would be the time to write to a set of 12 volatile variables that is memory mapped to the FPGA. Full-speed consecutive writes - with zero wait states - require two clock cycles; any wait state that is used adds one clock cycle to the duration [56]. Current applications of the PEC33 board utilise three wait states

for the external memory interface [57]; the output time is therefore estimated to be at least 12x5 = 60 clock cycles, or 800,00 ns.

The timing estimates is summarised in Table 3.1.

Algorithm component	Required length of time
A/D conversion time	1,49 µs
Serial optical fibre transmission time	40 µs
Time for calculation of reference values	$T_{refCalc}$, dependent on the specific algorithm that is implemented
Time for calculation of SVM output voltage levels	9,53 µs
Time for state substitution algorithm	$T_{stateSubstitution}$, dependent on specific implementation strategy: Donzel and Bornard (DSP-only): 39,76 µs Proposed capacitor-voltage stabilisation algorithm (DSP-only): 4,51 ms
Time to output and activate converter output voltage levels	0,8 µs
Total time	1,49 µs + 40 µs + $T_{refCalc}$ + 9,53 µs + $T_{stateSubst}$ + 0,8 µs
Apparent switching period of a 7-cell converter switching at 1 kHz	142,9 µs

Table 4.1 *A timing budget using relevant timing estimates*

The following inequality, based on the estimates set out above, can now be presented

$$1{,}49\ \mu s + 40\ \mu s + T_{refCalc} + 9{,}53\mu s + T_{stateSubstitution} + 0{,}8\mu s \leqslant 142{,}9\mu s$$
$$\text{or}$$
$$T_{refCalc} + T_{stateSubstitution} \leqslant 91{,}08\mu s \qquad (4.1)$$

where $T_{refCalc}$ and $T_{stateSubstitution}$ is as detailed in Table 3.1.

Several remarks are in order:

- A DSP-based implementation of the Donzel and Bornard algorithm should not pose any problems, and the investigation as to practical implementability is therefore completed.

  It can be seen from inequality (3.1) that the $T_{stateSubstitution}$ estimate of 39,76 µs leaves a maximum value of 51,32 µs for the calculation of the $T_{refCalc}$ value.

- A partial FPGA-based implementation of the proposed capacitor voltage stabilisation algorithm is justified.

  The proposed capacitor-voltage stabilisation algorithm requires a $T_{stateSubstitution}$ value of 4,51 ms. It can be seen that this value cannot satisfy the inequality (for a positive $T_{refCalc}$ value). A partial FPGA-based solution is therefore justified, as a DSP-only solution cannot be achieved.

- A new set of capacitor voltage measurements can only be sampled every 73 (apparent) switching periods, or every 10 ms.

  This can be seen from the 'Part B' estimate of the proposed capacitor-voltage stabilisation algorithm of 2,88 ms and the 40 µs total transmission time. The implication thereof, for an apparent switching period of 142,9 µs, is that a new set of capacitor voltage measurements can only be taken every 73 (apparent) switching periods, or every 10 ms. Consequently, to ensure that the cell capacitor voltage values do not change drastically during this time, a capacitance value that would result in a time constant exceeding 10 ms (for a specific rated load current) should be selected. The capacitance value would be dependent on the specific rated load current, which is still uncertain at this stage, however.

- It can be said that the fast FPGA-based sorting (in the 'Part B' algorithm) of the proposed capacitor stabilisation algorithm is, strictly speaking, unnecessary for a capacitance value chosen for a time constant that exceeds 10 ms.

  However, the sorting operation in the proposed capacitor voltage stabilisation algorithm was implemented in hardware in this thesis. This can be implemented along with a parallel measurement acquisition system. Although the specific measuring interface was not implemented in this thesis, an FPGA-based measurement interface can be envisaged that would allow voltage measurements to be performed in parallel, as opposed to the serial acquisition of a DSP-based solution. This means that the 10 ms time restraint would no longer be applicable, as the sampling rate can be increased; smaller capacitors can be used and the voltage regulation performance can be maximised for the specific capacitance values.

- 'Part A' of the proposed capacitor voltage stabilisation algorithm must execute in the blanking time of 7 μs if per-phase verification (i.e. verification on a single-phase scale model) is to be performed, for comparison with the results shown in Figures 5.5-5.7 and Table 5.12.

  As the next switching instants are generally not known beforehand with the carrier-based modulation of a single-phase converter, the 'Part A' algorithm can only start to execute when a switching state transition is detected. Because of this, an additional timing restraint comes into effect: the whole of the 'Part A' algorithm must execute within the switch blanking time.

4.4 VHDL FOR A CAPACITOR-VOLTAGE BASED CONTROLLER

As it was decided that the 'Part A' of the proposed capacitor-voltage stabilisation algorithm and also the sorting of the capacitor error values would be performed within the FPGA, the bit-wise implication of the weight-allocation strategy set out in Table 2.5 and Figure 2.4 can also be considered for FPGA-specific implementation. This eliminates potential interfacing problems that could be encountered when doing only the calculation of the state ratings in the DSP.

The main components in the voltage stabilisation algorithm can be set out as follows:

1. measurement of the capacitor error voltages
2. sorting of the absolute capacitor error voltages
3. generation of switching functions
4. determination of the cell correspondences for each capacitor, as defined in Table 2.1
5. determination of the state rating
6. elimination of switching states representing the incorrect output voltage
7. elimination of switching states representing a non-minimum switch commutations
8. finding the maximum state rating
9. selecting the output switching function

However, with a FPGA realisation, unlike a DSP implementation, the logic internal to these 7 steps can be performed in parallel.

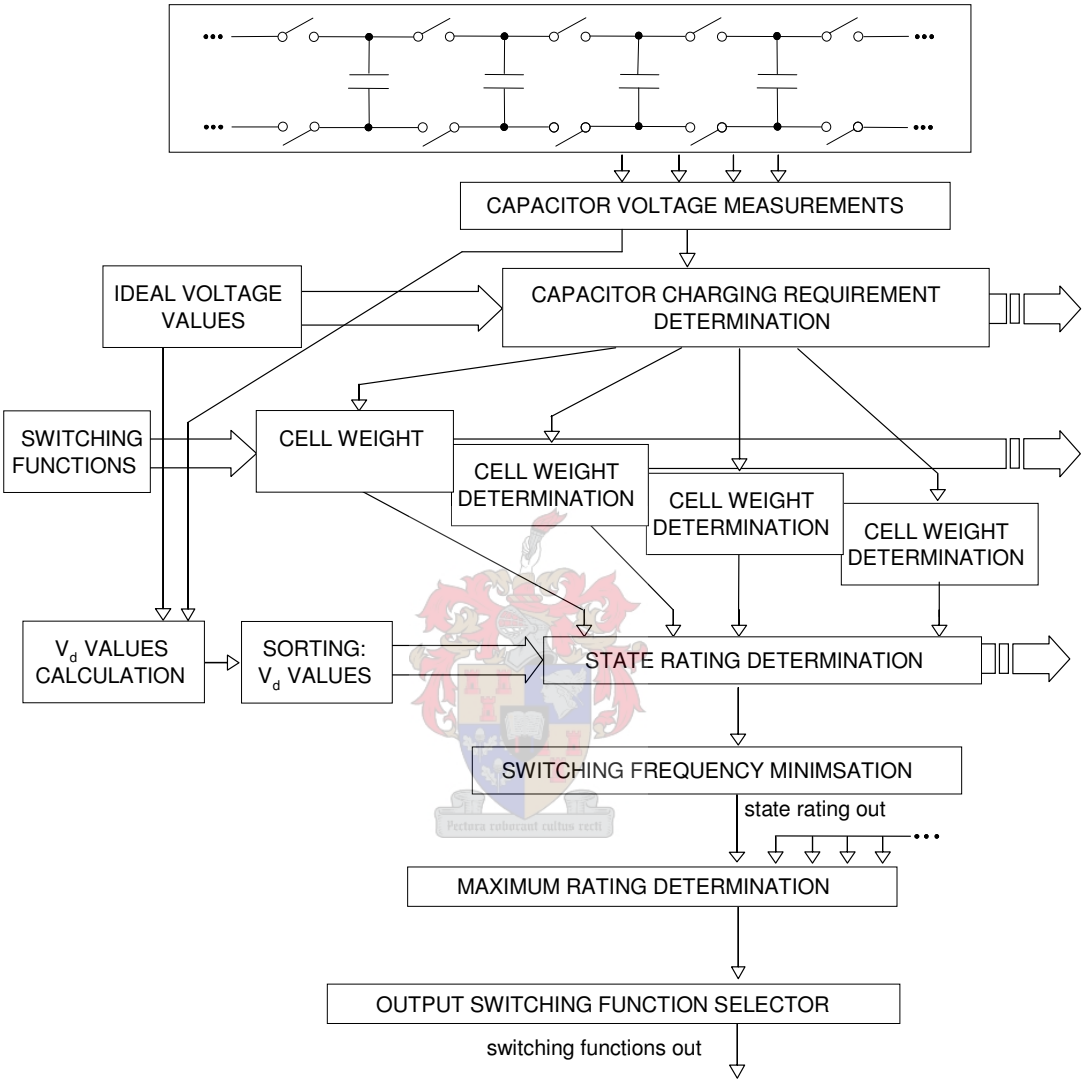


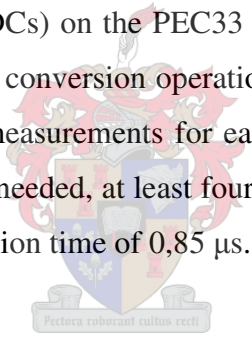
Figure 4.3 *The voltage stabilisation algorithm*

A synchronous design methodology is followed in the VHDL implementation of the voltage stabilisation algorithm. With a synchronous design, registers are used that are only sensitive to input signals on clock transitions. With the proper adherence to setup and hold times, the system loses its susceptibility to hazards. Asynchronous set and reset inputs of flip-flops should not be used during normal system operation, as these inputs can not be guaranteed to change the output at a particular time [72]. For this reason, the *sset* (synchronous-set) input of a *LPM_FF* is used; similarly, a behavioural description of the *Latch_with_sset* is defined.

Two coding styles are adopted in the VHDL realisation. Some blocks are written in a LPM-only (library-of-parameterised-modules-only) approach; others make use of behavioural descriptions. LPM blocks are efficient – Altera usually recommends using the LPM function rather than any other equivalent function. In addition, as the gate-equivalent description of the basis of some major blocks is described on a gate-and-multiplexer level, it makes sense to realise these in terms of LPM functions, rather than in terms of behavioural descriptions.

4.5 CONTROLLER DESIGN OVERVIEW

The architecture described in the previous section, will be utilised for both the sorting of the absolute capacitor error voltages and the determination of the maximum state rating. Measurements will be taken through the available four TLV1570 8-channel, 10-bit serial analogue-to-digital converters (ADCs) on the PEC33 system board. The ADC requires 16 clock cycles for one sampling and conversion operation. As a 7-cell converter implies that 13 voltage measurements – two measurements for each floating capacitor and one current sensor voltage measurement – are needed, at least four conversions have to be performed by one of the ADCs, in a total conversion time of 0,85 μs.



The available switching functions are generated by means of *LPM_constant* declarations.

The $(j+1)^{th}$ and the j^{th} switching functions of a single switching state $S_{(j+1)}$, S_j	**The charge/discharge requirement of the j^{th} capacitor C_j**	**The cell correspondence of the j^{th} capacitor C_j**
0,0 (no effect)	no req. (00)	correct
0,0 (no effect)	discharge (01)	no action
0,0 (no effect)	charge (10)	no action
0,1 (discharge)	no req. (00)	correct
0,1 (discharge)	discharge (01)	correct
0,1 (discharge)	charge (10)	incorrect
1,0 (charge)	no req. (00)	correct
1,0 (charge)	discharge (01)	incorrect
1,0 (charge)	charge (10)	correct
1,1 (no effect)	no req. (00)	correct
1,1 (no effect)	discharge (01)	no action
1,1 (no effect)	charge (10)	no action

Table 4.2 *Permutated definition of the j^{th} cell correspondence*

The cell-correspondence-determination component is specified through a combinational function, based on a slightly permutated version of the truth table specified in Table 2.1.

Here the state characteristics of 00, 01, 10 and 11 represent the *no effect*, *discharge*, *charge* and *no effect* state characteristics respectively. These values can be obtained from a specific state switching function quite easily, as the state characteristic for a specific capacitor is formed by the two specific adjacent switching functions. The cell capacitor requirements of 00, 01 and 10 are defined as *no requirement*, *discharge requirement* and *charge requirement* respectively.

A combinational function, based on the truth table specification in Table 4.2, is implemented to calculate the various cell correspondences. These cell correspondences are calculated as *correct* (*r*) and *incorrect* (*w*) values only, as is shown in Figure 4.4. It can be seen that the *no action* (*n*) weights used in Figure 2.4 have been eliminated for a simplified digital implementation.

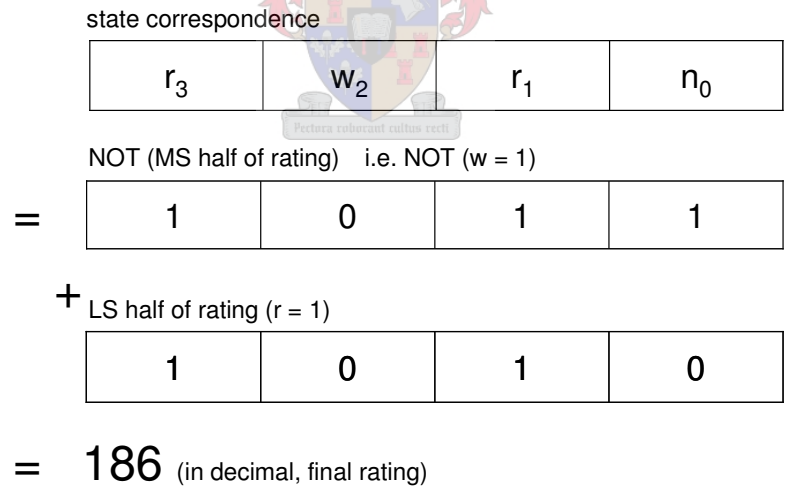


Figure 4.4 *Digital implication of weight-allocation strategy*

A critical component of the design, is the permutation of the cell correspondences according to the sorted order of the absolute capacitor error voltages. A *Custom_mux* (a custom multiplexer, as the selector input is not number-encoded, but 'one-hot') block is utilised, where the selector indicates the index of the input that should be connected to the output.

When the *Custom_mux* block is used in conjunction with the output of the sorting architecture in Figure B.15, the various cell correspondences of a state is permutated to form the state rating.

The rank-ordering architecture is utilised again for the identification of the maximum state rating. However, now only one cycle through the single hardware stage is implemented. The output index of this block is used as a selector for the *Custom_mux* block described in the following subsection.

Finally, a *Custom_mux* component is used in the *sw_fn_select* block to generate the output switching functions based on the indices obtained from the priority encoder input.

Cell_corresp_det	The cell correspondence values (correct, incorrect or no action) – as defined in Table 2.1 – are calculated from both the input switching functions and the input capacitor charge/discharge requirements.
Rating_det	Two permutations of the cell correspondence block output are performed according to the output indices of the sorting block (*ROF_core*)
State_sw_fn_gen	The state switching functions are generated by means of *LPM_constant* definitions.
Priority_encoder	This block ensures that only one output index is generated by the sorting block (*ROF_core*) at a specific instant. This is especially of importance when two or more inputs are equal.
ROF_cell_mod	This block specifies the basic modified rank order filter architecture that is shown in Figure B.15 (Shift register and internal enables not included).
ROF_bit_plane	This is a wrapper for the *ROF_cell_MOD* and *majority_greatest* block.
ROF_core	This is a wrapper block for the *ROF_bit_plane* block.
Majority_greatest	Here, the majority decision functionality is implemented by means of an OR gate.
Sorter_greatest	This block encapsulates the *ROF_core* and shift registers, for determination of the maximum state rating.
Sorter_ranked	This block encapsulates the *ROF_core*, shift registers and internal enabling signals (generated by the *enabler_repeat* block), for a full sorting of the absolute capacitor error voltages.
Enabler_repeat	Generates intricate enabling signals internal to the full sorting architecture in *sorter_ranked*.
Custom_mux	This block is utilised in the permutation of cell correspondences in *rating_det* and also in the output switching function selection in *sw_fn_select*
Latch_with_sset	This block forms part of the bit-mask unit that is utilised in *sorter_ranked*.
Delay_1	This block is used to delay enabling signals internal to the *sorter_ranked* block.
Sample_and_hold	Various copies of the block are used to sample the sorter block output indices at specific time instants and to hold constant these values for use by *custom_mux* blocks.

Table 4.3 *Summary of VHDL block functions*[2]

[2] Refer to Appendix C for VHDL source code

Table 4.3 sets out the functionality of the various FPGA-based blocks that are used in the hardware co-controller realisation. A listing of the VHDL source is given in Appendix C. Figure 4.4 shows the interdependencies of the various VHDL-defined blocks. Simulated results will be shown in Chapter 5.

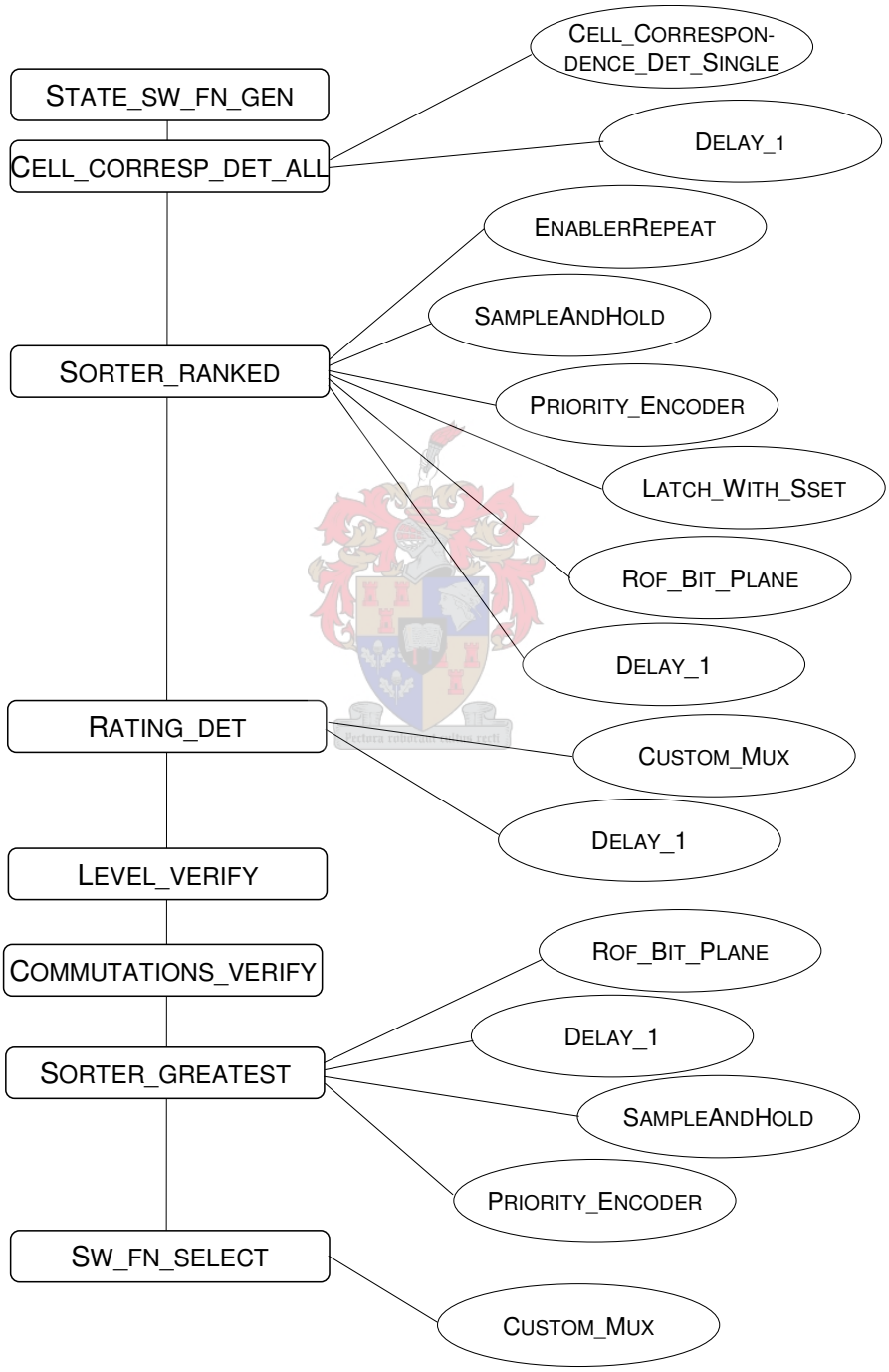


Figure 4.5 *Schematic outline of VHDL implementation*

4.6 SUMMARY

It was shown that a DSP-only solution would be inadequate for the proposed capacitor-voltage stabilisation algorithm. In addition, it was shown that the idea of a VHDL-based implementation is not at all far-fetched when looking at current DSP and FPGA industry trends. A VHDL-based co-controller was realised by means of a scalable design approach. Use was made of the hardware sorting architecture proposed by Hatirnaz and Leblebici.

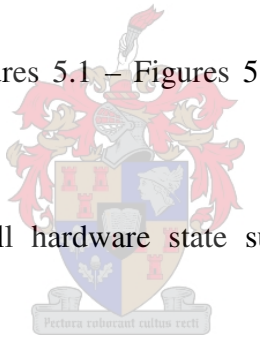
Chapter 5
Results

5.1 INTRODUCTION

In this chapter, results of the VHDL-based co-controller will be given. An example of the agreement between the Simplorer results and the Quartus results will be shown. Also, comparative results of the Donzel and Bornard algorithm performance and the proposed capacitor-voltage based algorithm will be given.

5.2 SIMULATED RESULTS

Various results are shown in Figures 5.1 – Figures 5.11. These will be discussed in the subsequent paragraphs.



The Quartus results of the 4-cell hardware state substitution algorithm are shown in Figure 5.1.

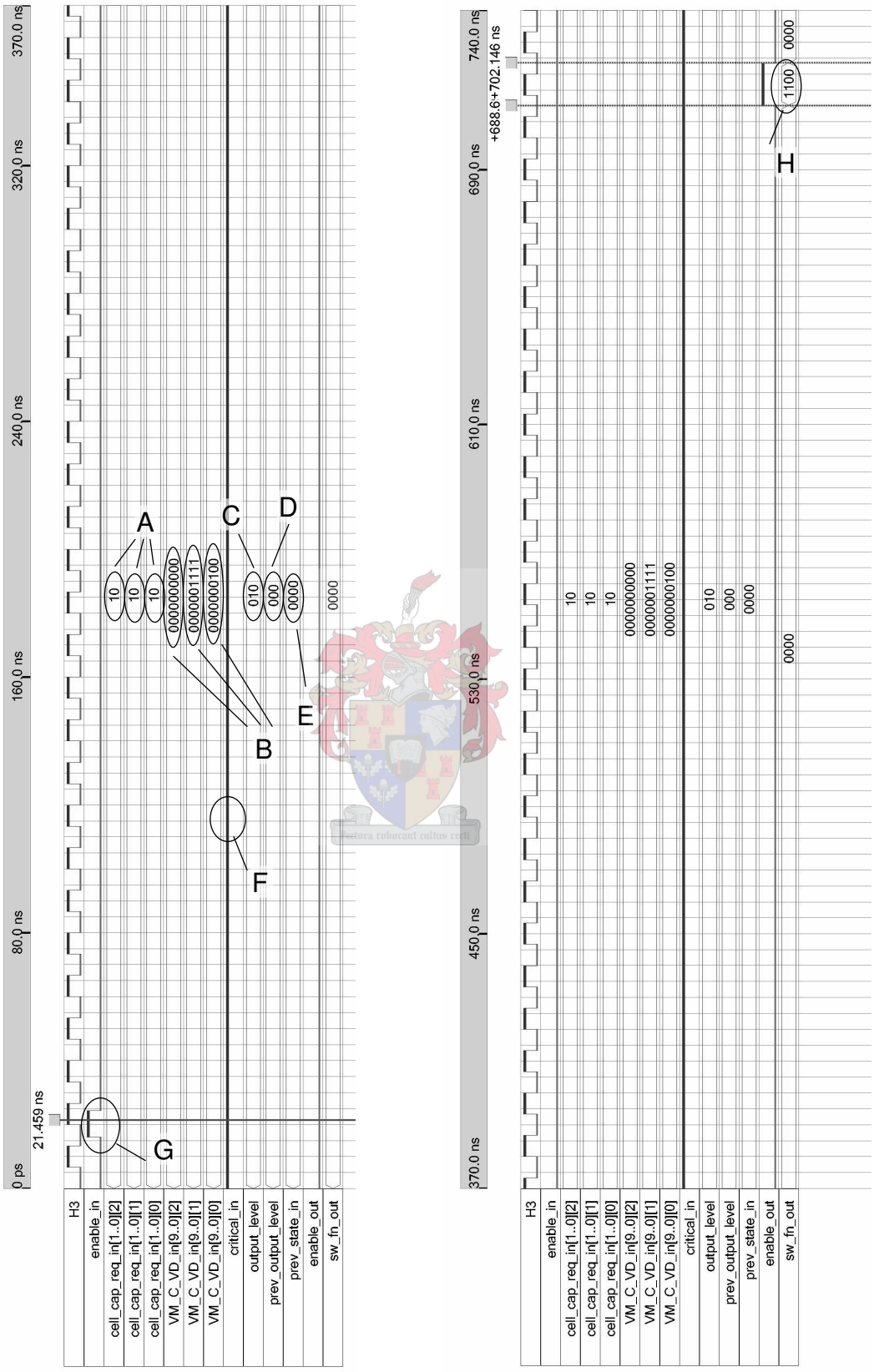


Figure 5.1 *State substitution switching function generator*

In Figure 5.1, the Quartus simulation results are shown, compiled and simulated for the EP1K50QC208 FPGA with a 75 MHz clock applied. Inputs to the controller block are as follows: all capacitor requirement values are set to '10' (this can be seen at the points marked 'A' in Figure 5.1), indicating that all capacitors require charging – refer to Table 4.2. Arbitrary values are chosen for the three absolute capacitor voltage errors (*VM_C_VM_in* inputs as can be seen at the points marked 'B'). It can be seen from the values specified at 'B' that that the capacitor C_2 in Figure 2.1 has the largest absolute error voltage; capacitor C_1 has the second largest absolute error voltage and capacitor C_1 has the third largest error voltage. The required output voltage level is specified to be '2', as is specified at point 'C'; the output voltage level of the previous switching state is specified as '0' at point 'D'. The previous switching state is specified as '0000' at point 'E'. The *critical_in* input is specified as '1' (as can be seen at point 'F'), indicating that the number of switch commutations should be ignored in this case. The output switching state ('1100') and *enable_out* signal can be sampled at about 693 ns (at point 'H') after the *enable_in* signal has been sampled (at point 'G').

A four-cell compensator could be fitted into the FPGA. Care was taken to implement register interfaces and few output pins, for the determination of the maximum number of cells that could be fitted. However, with the results shown in Figure 5.2 an FPGA with more output pins were selected in order to route internal nodes to output pins for explanatory purposes.

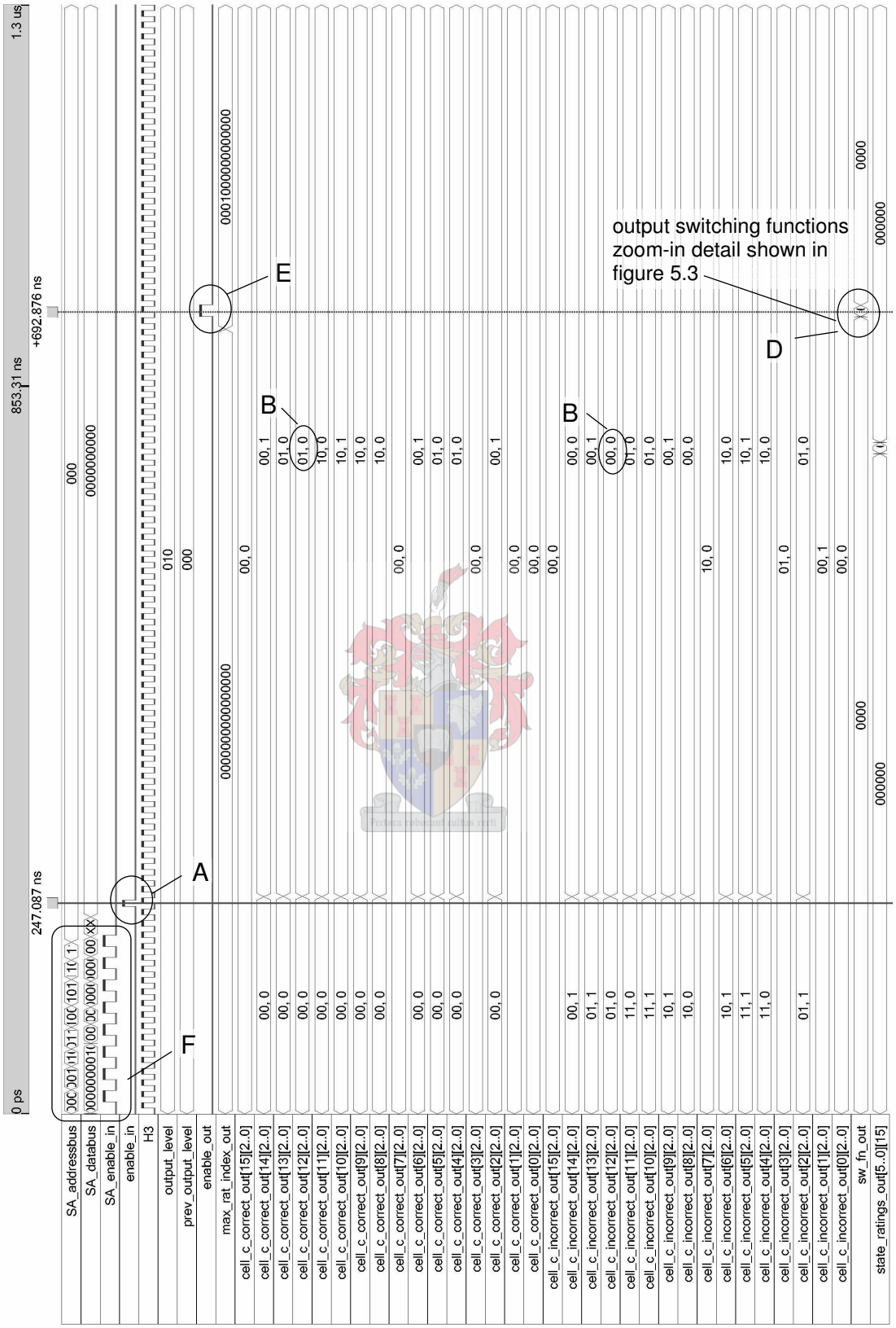


Figure 5.2 *State substitution switching function generator, shown with inner node values exposed*

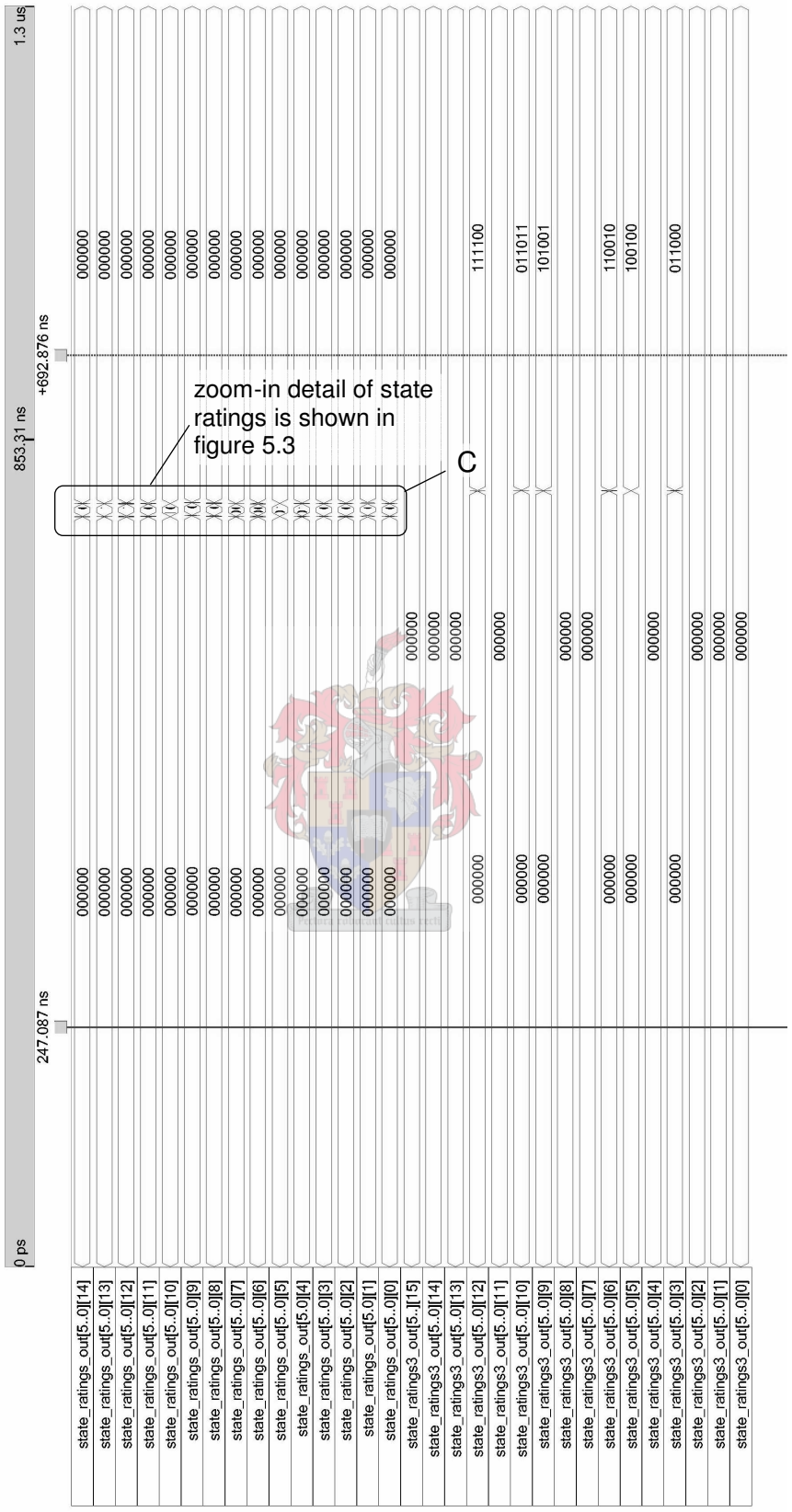


Figure 5.2 *(continued) State substitution switching function generator, shown with inner node values exposed*

The control block is enabled at point ‘A’; The values specified at points ‘A’ to ‘E’ in Figure 5.1, are now specified through a register interface at point ‘A’ in Figure 5.2.

The cell correspondences values that are computed, are shown at points ‘B’ for the 12^{th} switching function (in binary: ‘1100’): the value of the *cell_c_correct_out* vector is ‘010’ and the value of the *cell_c_incorrect_out* is ‘000’). This indicates that the cell correspondence of the capacitor C_1, as defined in Figure 2.2 (look at index 0 of the *cell_c_correct_out* and the *cell_c_incorrect_out* vectors in Figure 5.2) is neither *correct* nor *incorrect* – it is *no action*; the cell correspondence of capacitor C_2, as defined in Figure 2.2, is *correct* (look at index 1); the cell correspondence of capacitor C_3 is neither *correct* nor *incorrect* – it is therefore *no action.* As was explained with Figure 5.1, inputs indicate that all capacitors require charging; capacitor C_2 in Figure 2.1 has the largest absolute error voltage; capacitor C_1 has the second largest absolute error voltage and capacitor C_1 has the third largest error voltage. The cell correspondence values that are obtained, can therefore be verified to be correct by means of Table 4.2.

The state rating that is computed from these specific cell correspondences are discussed along with Figure 5.3, where zoom-in detail of points ‘C’ and ‘D’ of Figure 5.2 are given.

The output switching functions are ready to be sampled at point ‘D’, when the *enable_out* signal goes high at point ‘E’ in Figure 5.2.

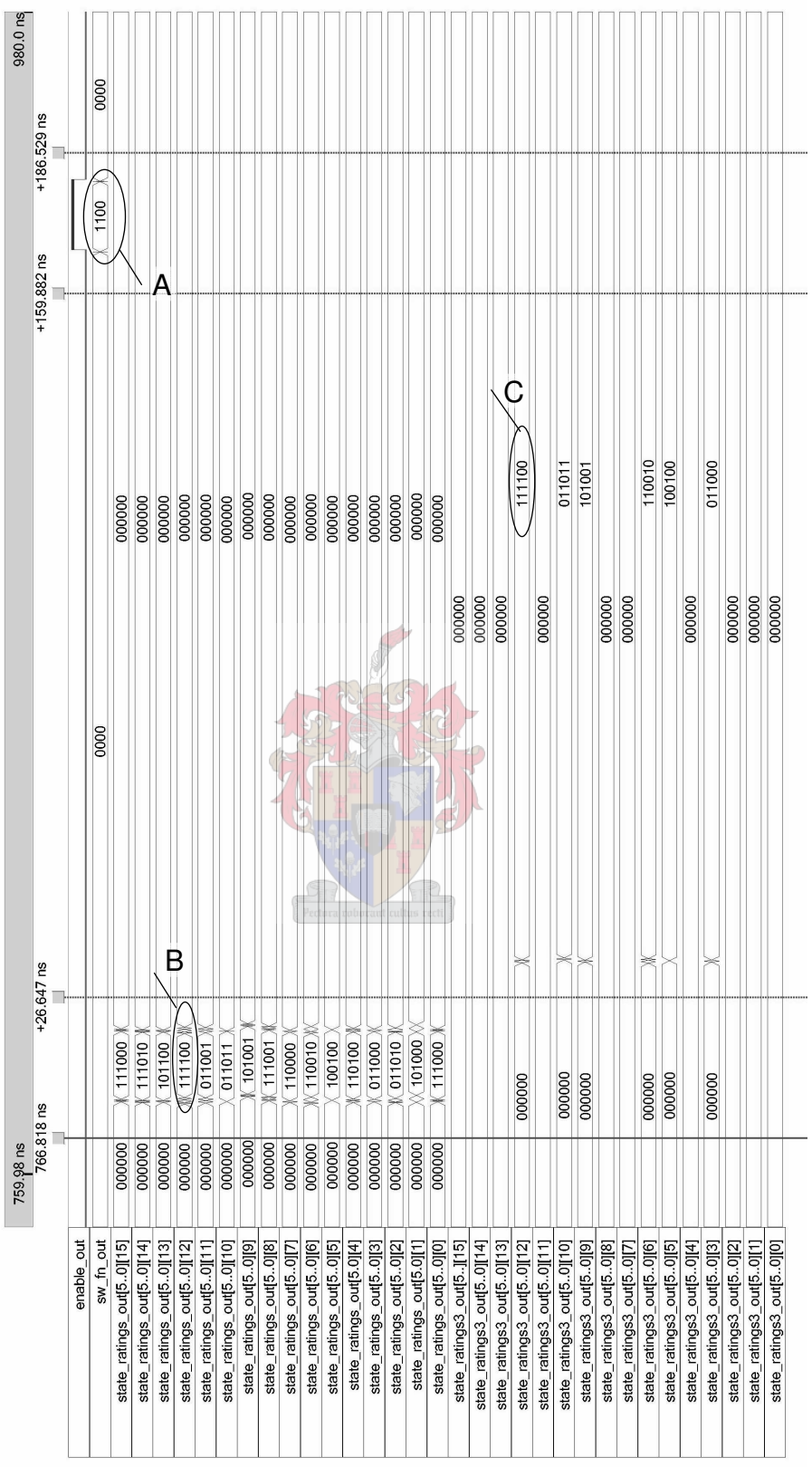


Figure 5.3 *Zoom-in detail of the state substitution switching function generator output in Figure 5.2.*

In Figure 5.3, zoom-in detail of points ‘C’ and ‘D’ in Figure 5.2 are given. It can be seen from the *sw_fn_out* output at point ‘A’, that the rating of the 12th switching function (‘1100’ in binary) has been identified as the largest, suitable state rating. When tracing the origins of the selected state rating, it is seen that the *correct* cell correspondences vector (the *cell_c_correct_out* signal) and the *incorrect* cell correspondences vector (the *cell_c_incorrect_out* signal) have values of '010' and '000' respectively (as was shown at points ‘B’ in Figure 5.2. The permutation and inversion of these values according to the sorted order of the absolute error voltages, as defined in Figure 4.4 and Table 2.5, results in a state rating of '111100' (as is shown in the *state_ratings_out*[5..0][12] output). Indeed, when the maximum of the state ratings corresponding to switching states with the correct output level – 2 in this case – is determined, '111100' is obtained.

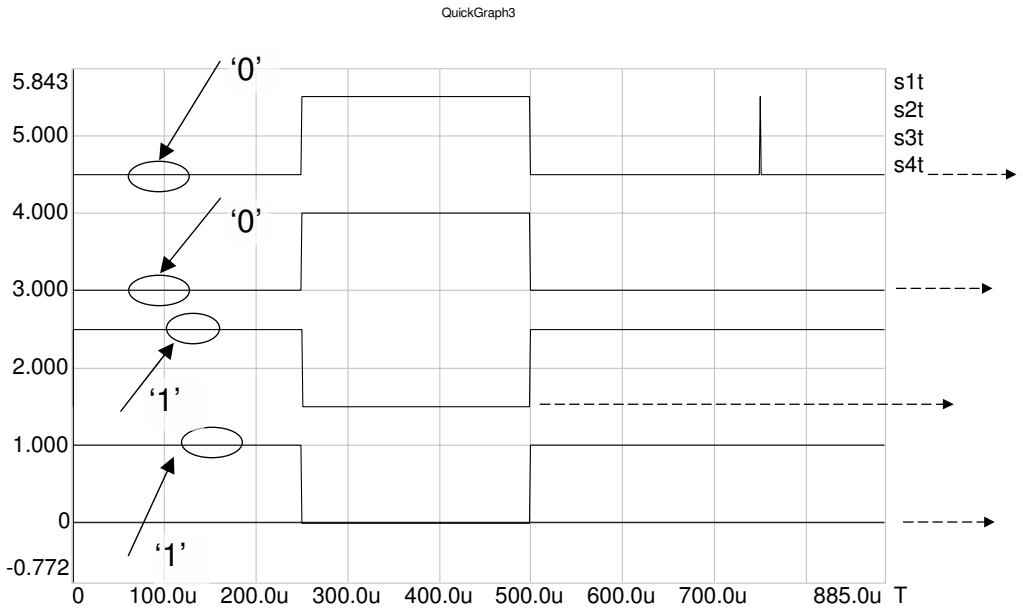


Figure 5.4 *An example of agreement in results of Simplorer and Quartus for the capacitor voltage stabilisation algorithm – initial output switching states are '1100' – see also: Figure 5.3. The zero-levels of the four signals are indicated (for clarity) with the dotted arrow lines.*

In Figure 5.4, the per-phase simulation model realised in Simplorer is set up to verify the specific state selection of the VHDL output in Figure 5.1 to Figure 5.3. The initial conditions of the capacitor voltages were set so that the absolute error voltages are, in descending order, $|V_{C2}|$, $|V_{C1}|$ and $|V_{C3}|$. These values were chosen so that all the capacitors require a charging operation. The output reference level was chosen as two. In addition, the initial condition of the output voltage level of the previous switching state was set to zero. In the result shown, the first switching function output can be identified as '1100'. This result is in accordance with the result of the Quartus simulation shown in Figure 5.1 to Figure 5.3.

Of course, a co-simulation verification process, such as followed in [7] would enable a more precise verification. A tool such as Simplorer 6, with VHDL-AMS functionality, could be used; however, the VHDL-AMS differs from standard VHDL (*for .. generate* constructs are not allowed, for instance).

Quasi-active-filtering results of the Donzel and Bornard [34] algorithm, as well as the capacitor-voltage based algorithm derived in section 3.6.2 will be presented in Figures 5.5 - 5.7. These are per-phase model results.

It must be noted that the capacitor with the highest voltage rating – the leftmost capacitor in Figure 2.1 – has been implemented as an ideal voltage source in all simulations. When, in future work, the capacitor is implemented directly, active power must be drawn to compensate for the losses in the converter, as in [74]. The leftmost capacitor, in active filter applications in general, is not used as one of the floating voltage sources required by the flying-capacitor topology but as an energy store, or in some instances, simply to facilitate the switching of the converter [29]. The capacitor voltage stabilisation algorithm will therefore still be applied in its original form; however, a modified active filtering current reference will be utilised.

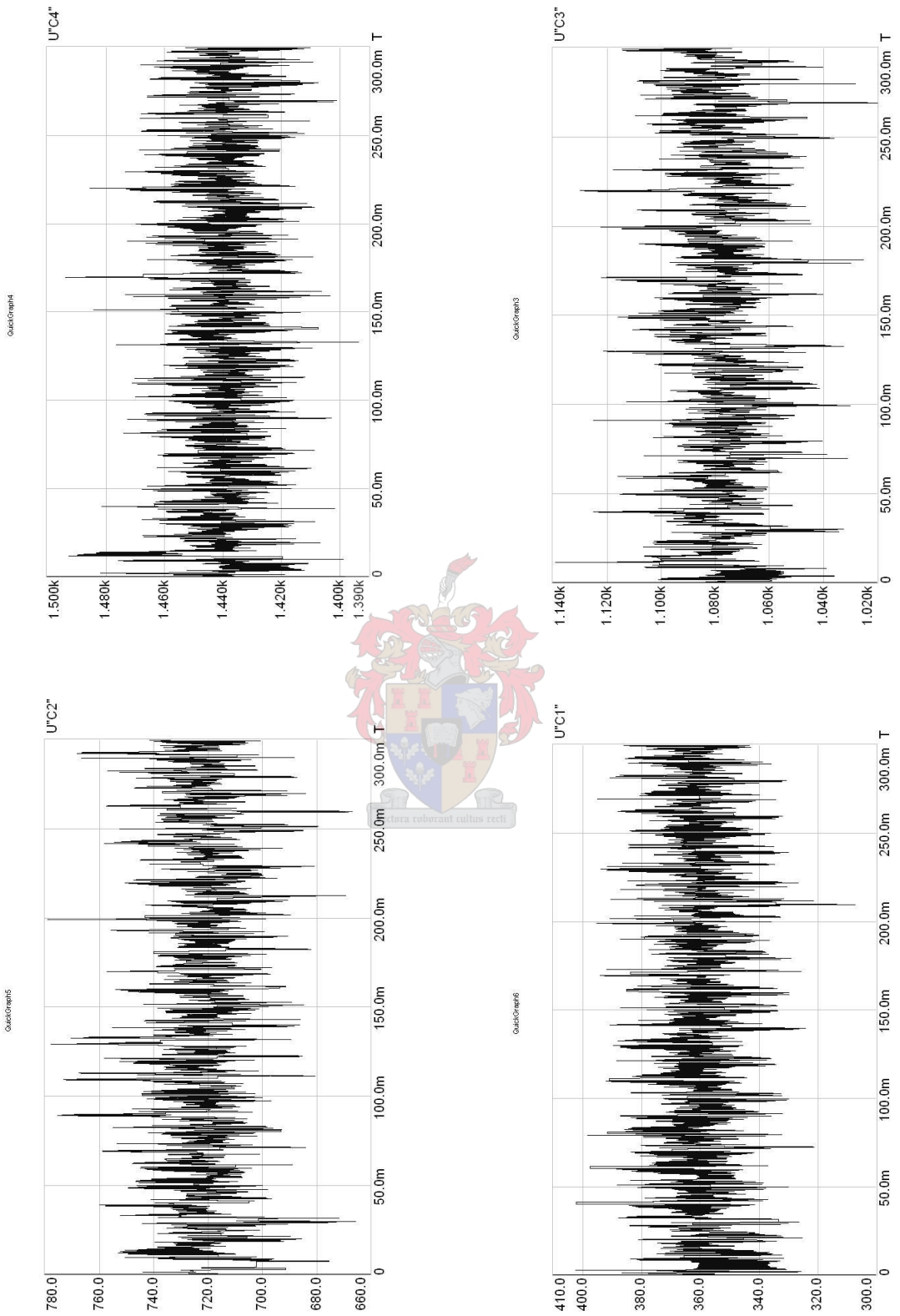


Figure 5.5 *Simulated capacitor voltages of the Donzel and Bornard algorithm in a single-phase quasi-active-filter application*

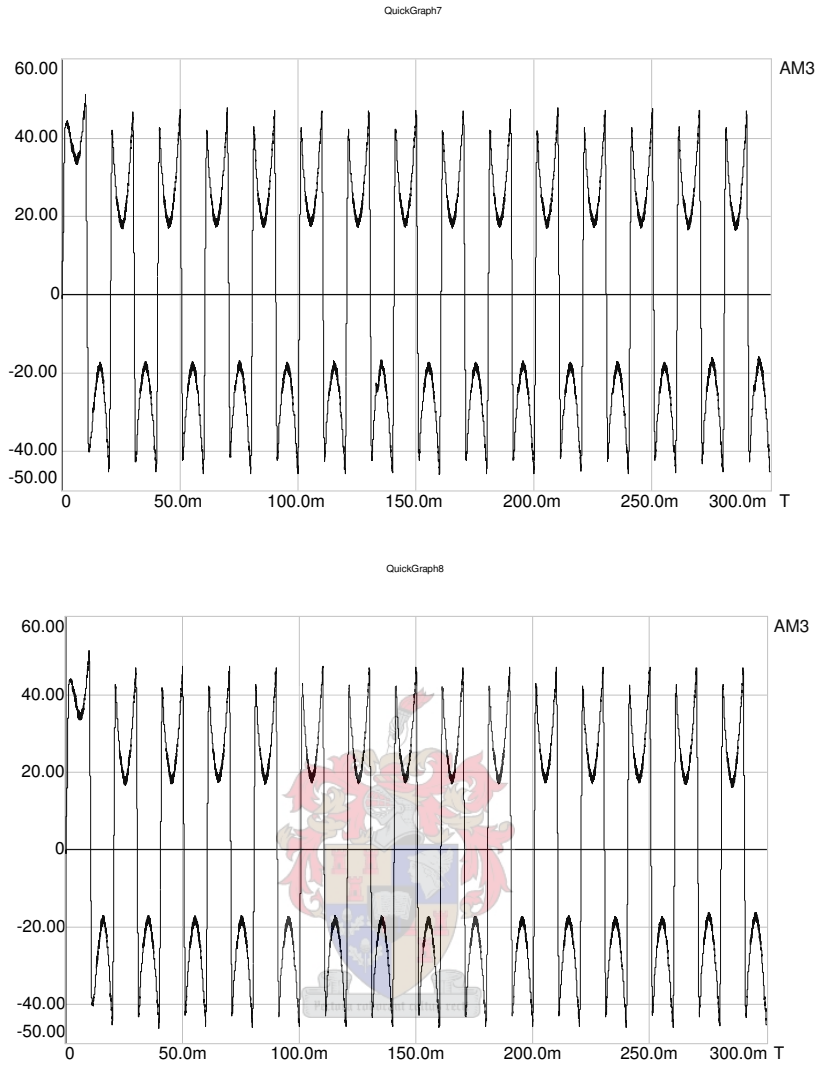


Figure 5.6 *Simulated converter output current for the Donzel and Bornard algorithm (top) and the proposed capacitor-voltage algorithm (bottom)*

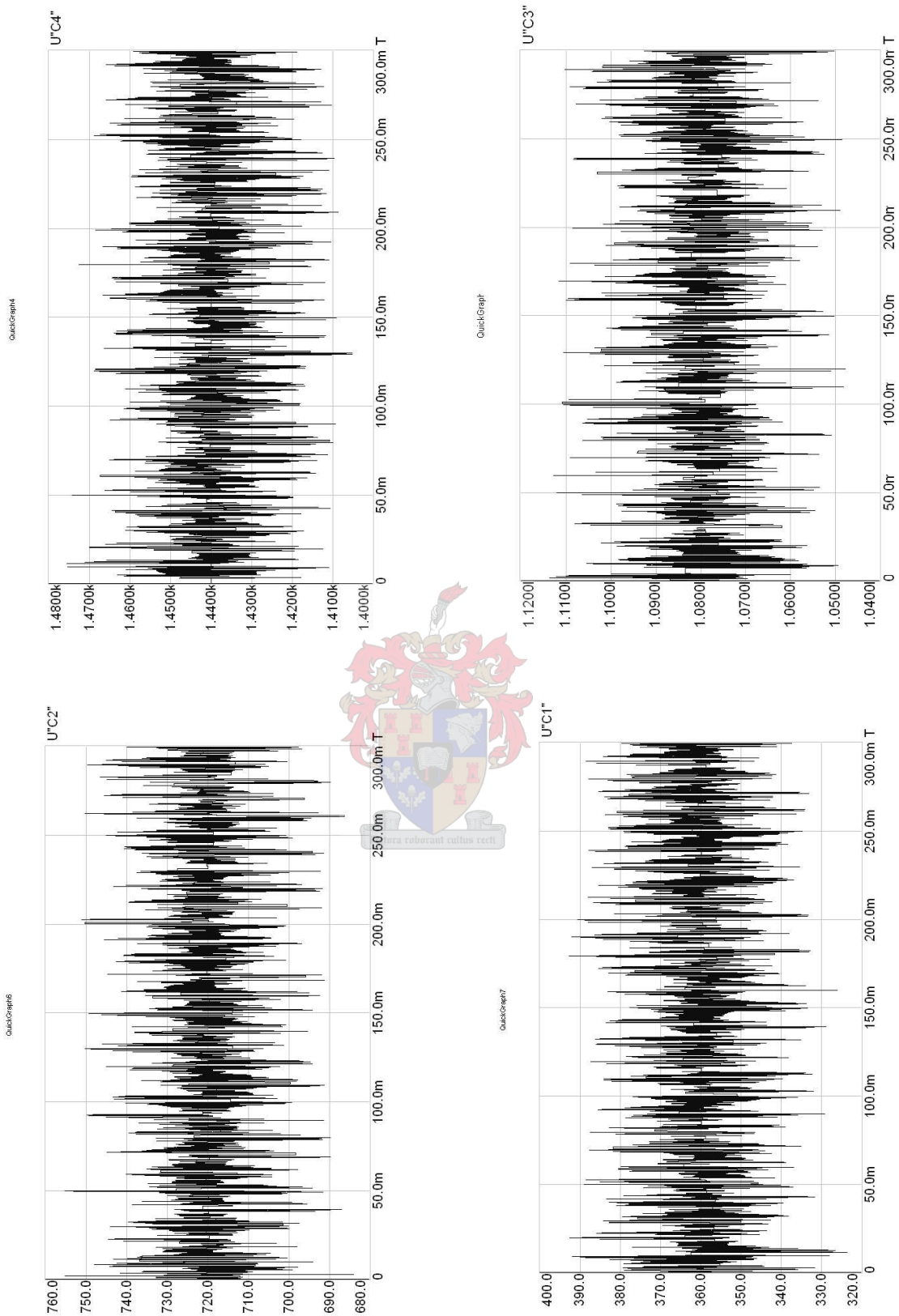


Figure 5.7 *Simulated capacitor voltages of the proposed capacitor-voltage algorithm for a single-phase quasi-active filter application*

A visual representation of the simulated capacitor voltages obtained through the Donzel and Bornard algorithm is shown in Figure 5.5. Similarly, simulated capacitor voltages obtained in the proposed capacitor-voltage algorithm is shown in Figure 5.7. A discrete-time approximation [75] of a Fourier transform is used to obtain the quasi-active filtering current reference. Visual representations of the relevant output current waveforms are shown in Figure 5.6. These two simulations were performed with minimum and maximum time step values of 1μs and for time duration 300 ms. The single-phase diode rectifier current is specified as 50 A, the DC bus voltage is 1800 V, the switching frequency is set to 1 kHz and the injection inductance value is specified as 9 mH. Characteristic values of the simulation were obtained through the Simplorer Day tool; data can be inspected in the following table.

	Donzel and Bornard algorithm		**capacitor-voltage based algorithm**	
	mean value	AC RMS value	mean value	AC RMS value
V_{c4}	1439,5 V	11,968 V	1439,8 V	9,7672 V
V_{c3}	1077,0 V	13,387 V	1080,1 V	8,5689 V
V_{c2}	721,93 V	13,698 V	720,01 V	9,1003 V
V_{c1}	360,00 V	12,011 V	360,38 V	9,2241 V

Table 5.4 *Performance data of the two algorithms under consideration*

From the data in Table 5.4, it can be seen that the proposed capacitor-voltage based algorithm compares quite favourably to the Donzel and Bornard algorithm, as it gives a voltage regulation, in this case, superior to that obtained with the Donzel and Bornard control.

Three-phase output waveforms will be shown in an active filtering application, utilising the generalised theory of Peng *et al.* and the fast algorithm of Celanovic *et al.* Here, all flying capacitors are replaced by floating voltage sources. The presented result therefore serves as an instantiation of the Celanovic *et al.* algorithm.

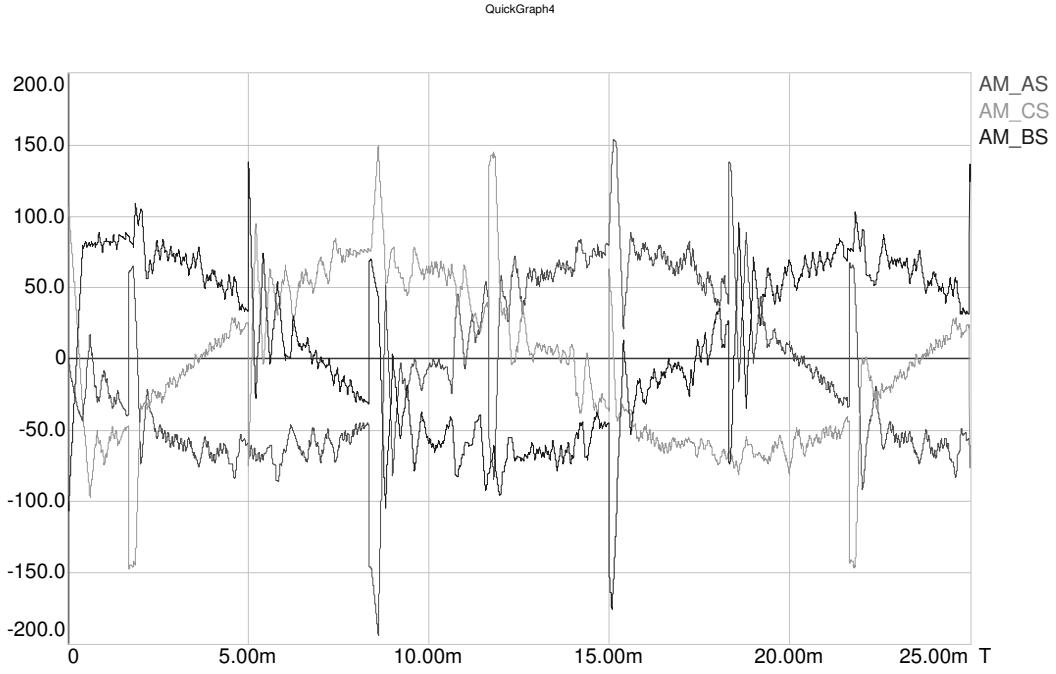


Figure 5.8 *Three-phase active filter simulated results for a 5-cell converter*

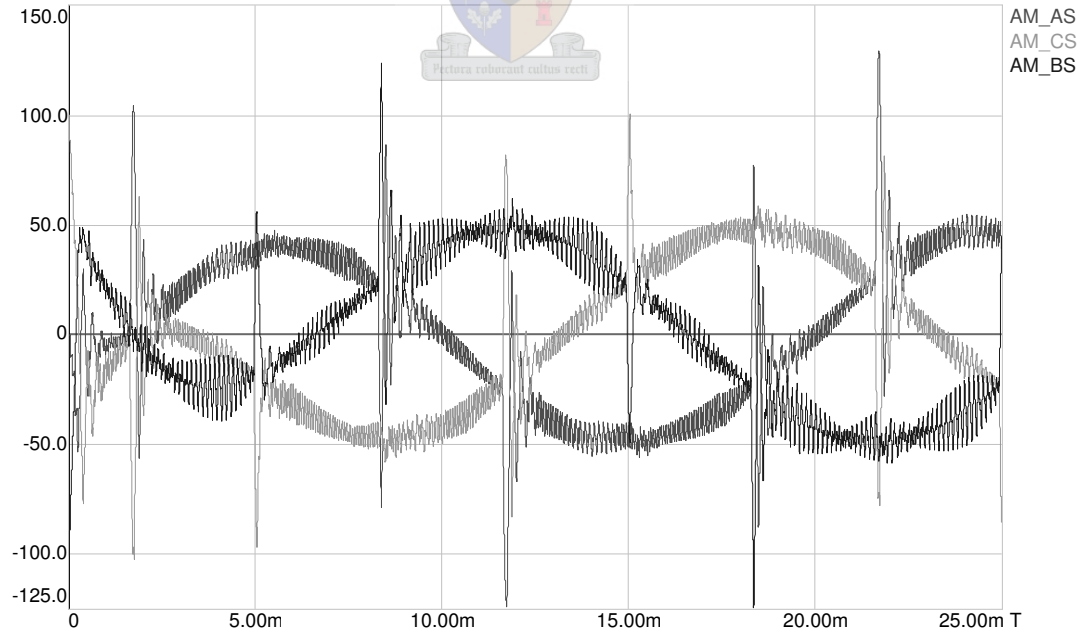
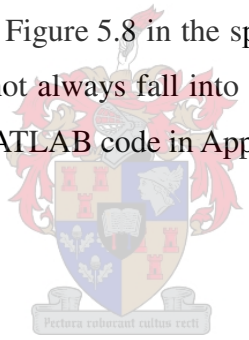


Figure 5.9 *Three-phase active filter simulated results for a 2-cell converter*

In Figure 5.8 a three-phase 5-cell converter active filter result is shown for a three-phase diode rectifier application; the rectifier current is 100 A, the DC bus voltage is 22 kV, the injection inductance is 7,5 mH and the simulated apparent switching frequency is 5 kHz. In Figure 5.9 a three-phase 2-cell converter active filter result is shown for a three-phase diode rectifier application; the rectifier current is 100 A, the DC bus voltage is 22 kV, the injection inductance is 7 mH and the simulated apparent switching frequency is 8 kHz.

The ripple in these results does look somewhat problematic. A fixed vector switching order was implemented; from Figure 5.11 is seems as if successive output space vectors do not necessarily fall into adjacent cells. These two factors may be linked to the undesirable ripple effects; however, this is speculation, and further investigation is needed.

Figure 5.10 and Figure 5.11 shows the output reference values of the three-phase 5-cell converter active filtering results of Figure 5.8 in the space vector plane. Here it can be seen that successive output vectors do not always fall into adjacent cells. The space vector plots were generated by means of the MATLAB code in Appendix D.



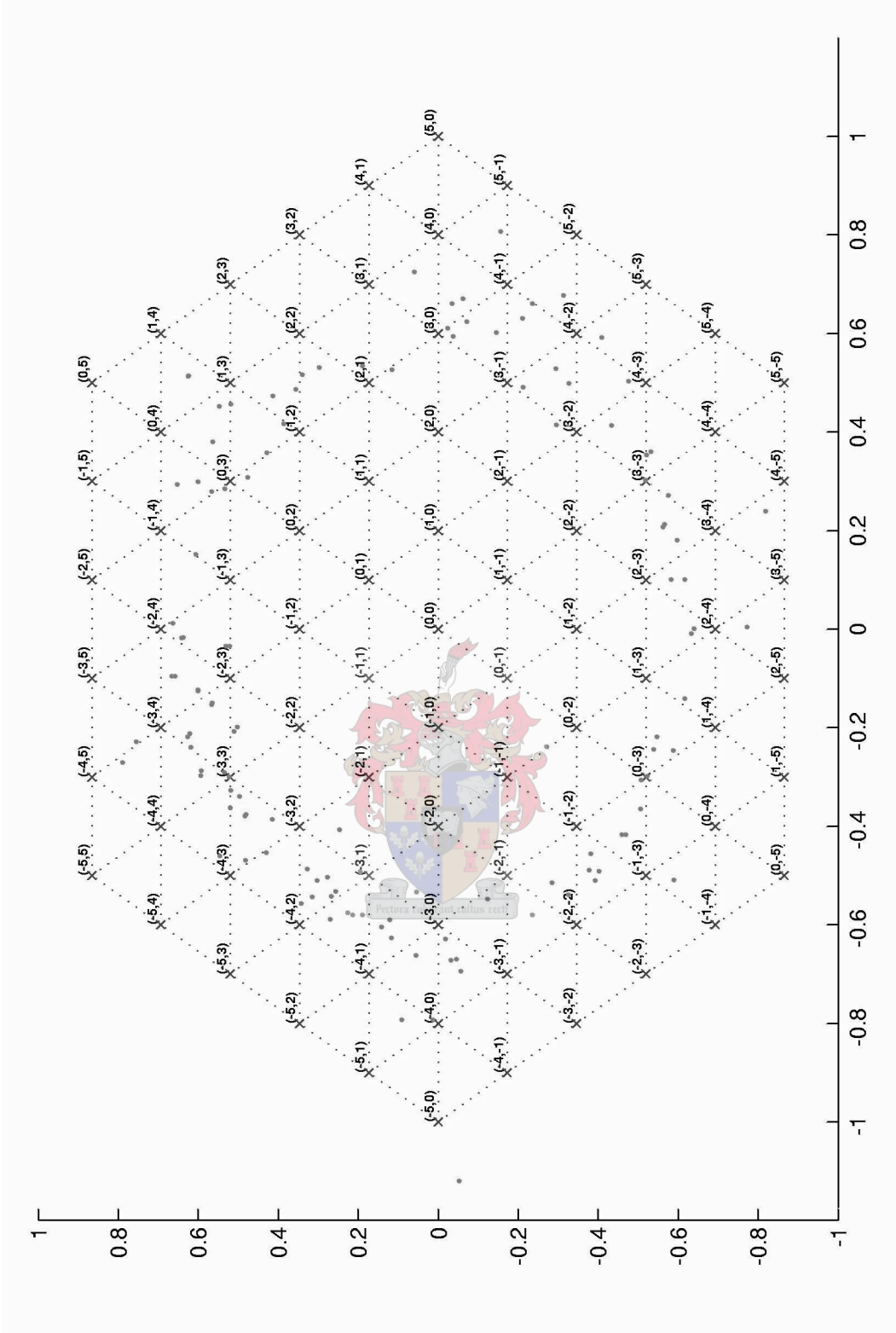


Figure 5.10 *Space-vector representation of output reference vectors*

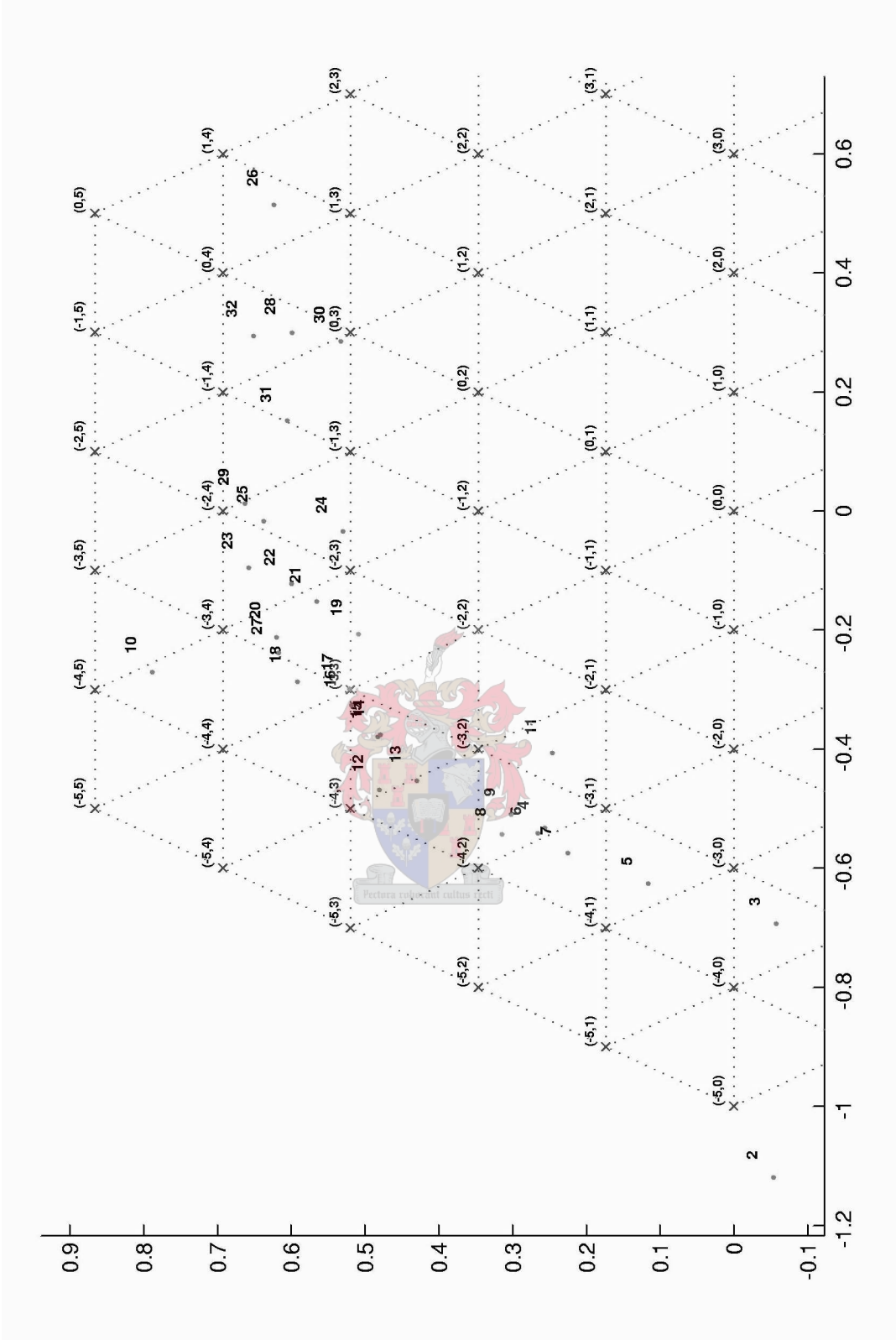


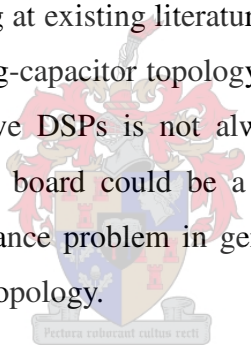
Figure 5.11 *Zoom-in detail of previous data – note that subsequent output vectors do not necessarily fall into adjacent cells*

5.3 DISCUSSION

The fact that only a 4-cell controller could be fitted into the FPGA will now be evaluated. The questions to be asked are whether the FPGA implementation is efficient, whether the FPGA should be used and whether the derived capacitor-voltage algorithm is efficient or not.

The most likely source of inefficiency in the FPGA solution would be the parallel generation of the available switching functions. The sorting algorithm, however, and the determination of maximum values are very efficient. The modifications to the sorting architecture of Hatirnaz and Leblebici (refer to subsection 4.5) performs well.

In this thesis, it was shown that a DSP-only solution would not be sufficient. Consequently, an FPGA was used. When looking at existing literature [7, 35] it can be seen that balancing control and estimation in the flying-capacitor topology was implemented in an FPGA. Use of the newest and more expensive DSPs is not always the best solution. Cost and/or development time of a controller board could be a factor. A small, compact, fast and inexpensive solution to the imbalance problem in general, could do much to increase the popularity of the flying-capacitor topology.



When looking at the FPGA implementation of Martins *et al.* in [35], it can be seen that a maximum or minimum value of the inputs is determined. Donzel and Bornard use five sorting operations in their algorithm [34]. Hence, the sorting operation in the derived capacitor-voltage based algorithm is justified.

Further investigation of capacitor-voltage based control and cell-voltage based control are needed; however it is suspected that a capacitor-based control would result in an improved capacitor-voltage regulation.

5.4 SUMMARY

Detailed results of the VHDL-based co-controller have been given. Quasi-active-filtering results of both the Donzel and Bornard algorithm, as well as the proposed capacitor-voltage based algorithm (on a per-phase basis) have been presented. Three-phase output waveforms have been shown in an active filtering application, utilising the generalised theory of Peng *et al.* and the fast SVM algorithm of Celanovic *et al.*

Chapter 6
Conclusions

6.1 CONCLUSIONS

As it was shown that a DSP-only (TMS320VC33) implementation of the proposed capacitor-voltage stabilisation algorithm would not meet the timing requirement, an FPGA-based (EP1K50QC208) voltage stabilisation co-controller architecture was designed. FPGA-based co-controllers are gaining popularity and offer some significant advantages. A 4-cell capacitor-voltage based algorithm could be fitted into the FPGA. It is difficult to evaluate this result: the aim was to fit three 7-cell compensators into one FPGA; however, no comparable result is available to serve as a benchmark.

The sorting process in the stabilisation algorithm is justified, as Donzel and Bornard use sorting in their cell-voltage based algorithm [34]; Martins *et al.* also make use of maximum and minimum values in their algorithm [35]. A sorting operation makes sense from a control viewpoint, as it is probably the only way in which a state substitution method can generate the greatest control effort for the greatest imbalance. The sorting algorithm, adapted from the scalable ASIC architecture proposed by Hatirnaz and Leblebici [63], is very time-efficient, when compared to a DSP sorting operation.

One instance where the proposed capacitor-voltage stabilisation algorithm out-performs the Donzel and Bornard algorithm is shown in Figures 5.5-5.7 and Table 5.12. It is suspected that capacitor-voltage based control would offer better voltage regulation than would cell-voltage based control. However, further investigation would be needed for verification.

A starting point for the capacitor-voltage based algorithm is the generation of all available switching functions. This ensures that realisable switching functions are obtained as output; however, the architecture area escalates because of this approach. Consequently the use of a cell-voltage based controller may now be re-evaluated.

The Celanovic algorithm [37] will have increased complexity when all three-phase redundancies and variable vector orders are considered. Perhaps the carrier-based SVM realisation of McGrath that was proposed for the neutral-point-clamped and the cascaded topologies in [48] should be investigated instead.

6.2 CONTRIBUTIONS

The sorting architecture of Hatirnaz and Leblebici [63] was modified to give the indices of the sorted numbers as output rather than the sorted numbers themselves. The modified architecture was also extended to accommodate zero inputs as well as inputs that are equal.

A scalable capacitor-based voltage stabilisation algorithm was derived and a 4-cell version hereof was implemented in VHDL and simulated in Quartus.

The fast space-vector algorithm of Celanovic [37] was implemented in Simplorer (up to 5-cell); both the Donzel and Bornard algorithm [34] and the proposed capacitor-voltage algorithm was implemented in Simplorer (5-cell). MATLAB m-files were written to plot space-vector outputs generated by Simplorer in an alpha-beta plane.

6.3 FUTURE RESEARCH

Aside from a practical verification, a few topics that may deserve some additional attention can be identified.

A model of a variable order space-vector modulation controller should be realised to verify the effect of vector order on the harmonic performance of increased-cells converters. Due to the increasing complexity of the SVM algorithm for increasing cells, it may be worthwhile to investigate the carrier-based optimised SVM approach of McGrath [48]. Another interesting alternative for the multilevel converter may be a hysteresis approach with constant switching frequency in steady-state, such as was proposed by Buso *et al*. [42] for conventional converters.

Currently, all available switching states are specified at the outset of both the proposed capacitor-voltage based controller and also the Escalante *et al.* capacitor-voltage based

controller [39]. It was argued that such an approach ensures that all output switching functions are physically realisable; it also contributes to the computational complexity of the algorithm. Another solution could perhaps be found.

References

[1] Glover, J.D. and M. Sarma (1994), *Power System Analysis and Design*, PWS, p. 242.

[2] Singh, B., K. Al-Haddad and A. Chandra (1999), "A review of active filters for power quality improvement," *Industrial Electronics, IEEE Transactions on,* vol. 46, pp. 960-971.

[3] Beukes, H. J. (1998), *Synthesis of a high-performance power converter for electric distribution applications*, PhD Thesis, University of Stellenbosch, February 1998.

[4] Cheng, P.-T., S. Bhattacharya and D. Divan (2000), "Experimental verification of dominant harmonic active filter for high-power applications," *Industry Applications, IEEE Transactions on,* vol. 36, pp. 567-577.

[5] Bhattacharya, S., P.-T. Cheng and D. M. Divan (1997), "Hybrid solutions for improving passive filter performance in high power applications," *Industry Applications, IEEE Transactions on,* vol. 33, pp. 732-747.

[6] Meynard, T. A., H. Foch, *et al*. (2002), "Multicell converters: basic concepts and industry applications," *Industrial Electronics, IEEE Transactions on,* vol. 49, pp. 955-964.

[7] Ruelland, R., G. Gateau, *et al*. (2003), "Design of FPGA-based emulator for series multicell converters using co-simulation tools," *Power Electronics, IEEE Transactions on,* vol. 18, pp. 455-463.

[8] Xu, L. and V. G. Agelidis (2002), "Flying capacitor multilevel PWM converter based UPFC," *Electric Power Applications, IEE Proceedings-,* vol. 149, pp. 304-310.

[9] Peng, F. Z., J. W. McKeever and D. J. Adams (1998), "A power line conditioner using cascade multilevel inverters for distribution systems," *Industry Applications, IEEE Transactions on,* vol. 34, pp. 1293-1298.

[10] Mouton, H. du T., J. H. R. Enslin and H. Akagi (2003), "Natural balancing of series-stacked power quality conditioners," *Power Electronics, IEEE Transactions on,* vol. 18, pp. 198-207.

[11] Meynard, T. A. and H. Foch (1992), "Multi-level conversion: high voltage choppers and voltage-source inverters," *Power Electronics Specialists Conference, 1992. PESC '92 Record., 23rd Annual IEEE.*

[12] Nabae, A., I. Takahashi and H. Akagi (1981), "A new neutral-point-clamped PWM inverter," *Industry Applications, IEEE Transactions on,* vol. IA-17, pp. 518-523.

[13] Hammond, P. W. (1997), "A new approach to enhance power quality for medium voltage AC drives," *Industry Applications, IEEE Transactions on*, vol. 33, pp. 202-208.

[14] Kuang, J. and B. T. Ooi (1994), "Series connected voltage-source converter modules for force-commutated SVC and DC-transmission," *Power Delivery, IEEE Transactions on,* vol. 9, pp. 977-983.

[15] Corzine, K. A. and X. Kou (2003), "Capacitor voltage balancing in full binary combination schema flying capacitor multilevel inverters," *Power Electronics Letters, IEEE,* vol. 1, pp. 2-5.

[16] Delmas, L., G. Gateau*, et al.* (2002), "Stacked multicell converter (SMC): control and natural balancing," *Power Electronics Specialists Conference, 2002. PESC 02. 2002 IEEE 33rd Annual*.

[17] Tan, P. C., P. C. Loh and D. G. Holmes (2003), "A robust multilevel hybrid compensation system for 25 kV electrified railway applications," *Power Electronics Specialist, 2003. PESC '03. IEEE 34th Annual Conference on*.

[18] Meynard, T. A. and H. Foch (1995), "Multilevel converters and derived topologies for high power conversion," *Industrial Electronics, Control, and Instrumentation, 1995., Proceedings of the 1995 IEEE IECON 21st International Conference on*.

[19] Holtz, J. (1992), "Pulsewidth modulation-a survey," *Industrial Electronics, IEEE Transactions on,* vol. 39, pp. 410-420.

[20] Fujita, H. and H. Akagi (1991), "A practical approach to harmonic compensation in power systems-series connection of passive and active filters," *Industry Applications, IEEE Transactions on,* vol. 27, pp. 1020-1025.

[21] Lai, J.-S. and F. Z. Peng (1996), "Multilevel converters-a new breed of power converters," *Industry Applications, IEEE Transactions on,* vol. 32, pp. 509-517.

[22] Tolbert, L. M., F. Z. Peng and T. G. Habetler (2000), "A multilevel converter-based universal power conditioner," *Industry Applications, IEEE Transactions on,* vol. 36, pp. 596-603.

[23] Song, B.-M., J.-S. Lai, *et al*. (2001), "A soft-switching high-voltage active power filter with flying capacitors for urban Maglev system applications," *Industry Applications Conference, 2001. Thirty-Sixth IAS Annual Meeting. Conference Record of the 2001 IEEE*.

[24] Liang, Y. and C. O. Nwankpa (2000), "A power-line conditioner based on flying-capacitor multilevel voltage-source converter with phase-shift SPWM," *Industry Applications, IEEE Transactions on,* vol. 36, pp. 965-971.

[25] Akagi, H. and H. Fujita (1995), "A new power line conditioner for harmonic compensation in power systems," *Power Delivery, IEEE Transactions on,* vol. 10, pp. 1570-1575.

[26] Akagi, H. (1997), "Control strategy and site selection of a shunt active filter for damping of harmonic propagation in power distribution systems," *Power Delivery, IEEE Transactions on,* vol. 12, pp. 354-363.

[27] Akagi, H. (1996), "New trends in active filters for power conditioning," *Industry Applications, IEEE Transactions on,* vol. 32, pp. 1312-1322.

[28] Akagi, H., Y. Kanazawa and A. Nabae (1984), "Instantaneous reactive power compensators comprising switching devices without energy storage components," *Industry Applications, IEEE Transactions on,* vol. IA-20, pp. 625-630.

[29] Peng, F. Z., G. W. Ott, Jr. and D. J. Adams (1998), "Harmonic and reactive power compensation based on the generalized instantaneous reactive power theory for three-phase four-wire systems," *Power Electronics, IEEE Transactions on,* vol. 13, pp. 1174-1181.

[30] Wilkinson, R. H. (2004), *Natural Balancing of Multicell Converters*, PhD Thesis, University of Stellenbosch, April 2004.

[31] Salagae, I. M. and H. du T Mouton (2003), "Natural balancing of neutral-point-clamped converters under POD pulsewidth modulation," *Power Electronics Specialist, 2003. PESC '03. IEEE 34th Annual Conference on.*

[32] Yuang, X., H. Stemmler and I. Barbi (2001), "Self-balancing of the clamping-capacitor-voltages in the multilevel capacitor-clamping-inverter under sub-harmonic PWM modulation," *Power Electronics, IEEE Transactions on,* vol. 16, pp. 256-263.

[33] Gateau, G., M. Fadel, *et al*. (2002), "Multicell converters: active control and observation of flying-capacitor voltages," *Industrial Electronics, IEEE Transactions on,* vol. 49, pp. 998-1008.

[34] Donzel, A. and G. Bornard (2000), "New control law for capacitor voltage balance in multilevel inverter with switching rate control (CVC)," *Industry Applications Conference, 2000. Conference Record of the 2000 IEEE.*

[35] Martins, C. A., X. Roboam, *et al*. (2002), "Switching frequency imposition and ripple reduction in DTC drives by using a multilevel converter," *Power Electronics, IEEE Transactions on,* vol. 17, pp. 286-297.

[36] Holmes, D. G. (1996), "The significance of zero space vector placement for carrier-based PWM schemes," *Industry Applications, IEEE Transactions on,* vol. 32, pp. 1122-1129.

[37] Celanovic, N. and D. Boroyevich (2001), "A fast space-vector modulation algorithm for multilevel three-phase converters," *Industry Applications, IEEE Transactions on,* vol. 37, pp. 637-641.

[38] BenAbdelghani, A., C. A. Martins, *et al*. (2002), "Use of extra degrees of freedom in multilevel drives," *Industrial Electronics, IEEE Transactions on,* vol. 49, pp. 965-977.

[39] Escalante, M. F., J.-C. Vannier and A. Arzande (2002), "Flying capacitor multilevel inverters and DTC motor drive applications," *Industrial Electronics, IEEE Transactions on,* vol. 49, pp. 809-815.

[40] Buso, S., L. Malesani and P. Mattavelli (1998), "Comparison of current control techniques for active filter applications" *Industrial Electronics, IEEE Transactions on,* vol. 45, pp. 722-729.

[41] Kazmierkowski, M. P. and L. Malesani (1998), "Current control techniques for three-phase voltage-source PWM converters: a survey," *Industrial Electronics, IEEE Transactions on,* vol. 45, pp. 691-703.

[42] Buso, S., S. Fasolo*, et al.* (2000), "A dead-beat adaptive hysteresis current control," *Industry Applications, IEEE Transactions on,* vol. 36, pp. 1174-1180.

[43] Rodriguez, J., J.-S. Lai and F. Z. Peng (2002), "Multilevel inverters: a survey of topologies, controls, and applications," *Industrial Electronics, IEEE Transactions on,* vol. 49, pp. 724-738.

[44] Lee, Y.-H., B.-S. Suh and D.-S. Hyun (1996), "A novel PWM scheme for a three level voltage source inverter with GTO thyristors" *Industry Applications, IEEE Transactions on,* vol. 32, pp. 260-268.

[45] Zhou, Z., T. Li, *et al*. (2004), "Design of a universal space vector PWM controller based on FPGA," *Applied Power Electronics Conference and Exposition, 2004. APEC '04. Nineteenth Annual IEEE.*

[46] Zhou, K. and D. Wang (2002), "Relationship between space-vector modulation and three-phase carrier-based PWM: a comprehensive analysis [three-phase inverters]," *Industrial Electronics, IEEE Transactions on,* vol. 49, pp. 186-196.

[47] Van der Broeck, H. W., H.-C. Skudelny and G. V. Stanke (1988), "Analysis and realization of a pulsewidth modulator based on voltage space vectors," *Industry Applications, IEEE Transactions on,* vol. 24, pp. 142-150.

[48] McGrath, B. P., D. G. Holmes and T. A. Lipo (2001), "Optimised space vector switching sequences for multilevel inverters," *Applied Power Electronics Conference and Exposition, 2001. APEC 2001. Sixteenth Annual IEEE.*

[49] Seo, J. H., C. H. Choi and D. S. Hyun (2001), "A new simplified space-vector PWM method for three-level inverters," *Power Electronics, IEEE Transactions on,* vol. 16, pp. 545-550.

[50] Celanovic, N. and D. Boroyevich (1999), "A fast space vector modulation algorithm for multilevel three-phase converters," *Industry Applications Conference, 1999. Thirty-Fourth IAS Annual Meeting. Conference Record of the 1999 IEEE.*

[51] Gateau, G., P. Maussion and T. Meynard (1997), "Fuzzy phase control of series multicell converters," Fuzzy Systems, 1997., *Proceedings of the Sixth IEEE International Conference on.*

[52] Lee, S.-G., D.-W. Kang, *et al*. (2001), "The carrier-based PWM method for voltage balance of flying capacitor multilevel inverter," *Power Electronics Specialists Conference, 2001. PESC. 2001 IEEE 32nd Annual*.

[53] Suekawa, E. and Kawaguchi (2004), "6.5kV IGBTs," *Mitsubishi Electric ADVANCE*, March 2004, pp. 17-20.

[54] Qiu, D. Y., Chung, H. S-H. and Hui, S. Y. R (2003), "On the Use of Current Sensors for the Control of Power Converters," *Power Electronics, IEEE Transactions on*, vol. 18, pp. 1047-1055.

[55] Texas Instruments (2000), "TLV1570 Serial Analog-to-Digital Converter," Literature Number SLAS169B, Available: http://www.ti.com.

[56] Texas Instruments (2004), "TMS320C3x User's Guide," Literature Number SPRU031F, Available: http://www.ti.com.

[57] Henning, P. (2004), PEC33 C-code for ADC interface, Available: lamut//public\projects\Pec 33\incoming\Quikstart\measure\measure.c

[58] Mitsubishi (2003), "CM900HB-90H 2nd-Version HVIGBT Modules," *Mitsubishi HVIGBT Modules*, Available: http://www.mitsubishichips.com/products/power/datasheet/power_hvigbt.html.

[59] Eupec (2002), "IGBT-Modules FZ 600 R 65 KF1," *Technical Information*, Available: http://www.eupec.com/gb/2_PRODUCTS/2_1_ProductRange/2_1_1_IGBT/gb_2_1_1_001_1_6500.htm.

[60] Mouton, H. du T. (2001), "Simulasie van drywingselektroniese bane", *Course notes: Electronics 414*.

[61] Detjen, D., S. Schroder and R. W. De Doncker (2003), "Embedding DSP control algorithms in PSpice," *Power Electronics, IEEE Transactions on,* vol. 18, pp. 294-300.

[62] Kaminski, B., K. Wejrzanowski and W. Koczara (2004), "An application of PSIM simulation software for rapid prototyping of DSP based power electronics control systems" *Power Electronics Specialists Conference, 2004. PESC. 2004 IEEE 35th Annual*.

[63] Hatirnaz, I. and Y. Leblebici (2000), "Scalable binary sorting architecture based on rank ordering with linear area-time complexity," *ASIC/SOC Conference, 2000. Proceedings. 13th Annual IEEE International*.

[64] Texas Instruments, "DSP Platforms : C6000™ DSPs," [Online], Available: http://dspvillage.ti.com/docs/catalog/dspplatform/overview.jhtml?templateId=5154&path=templatedata/cm/dspovw/data/c6000_ovw

[65] Eyre, J. (2002), "FPGA/DSP blend tackles telecom apps," *Electronic Engineering Times*, Available: http://www.bdti.com/articles/info_eet0207fpga.htm.

[66] Altera Corporation, "Stratix II DSP Blocks," [Online], Available: http://www.altera.com/products/devices/stratix2/features/dsp/st2-dsp_block.html.

[67] Texas Instruments, "Benchmarks : Independent Analysis," [Online], Available: http://dspvillage.ti.com/docs/catalog/generation/details.jhtml?templateId=5154&path=templatedata/cm/dspdetail/data/c6000_benchmarks_iba

[68] Shandle, J. (2003), "DSP meets FPGA: Is massive parallelism enough?" *TechOnLine*, Available: http://www.techonline.com/community/tech_group/31151.

[69] Altera Corporation, "FPGA co-processors for DSP applications," [Online], Available: http://www.altera.com/technology/dsp/devices/fpga/dsp-fpga_coprocessor.html.

[70] Ide, T. and T. Yokoyama (2004), "A study of deadbeat control for three phase PWM inverter using FPGA based hardware controller," *Power Electronics Specialists Conference, 2004. PESC. 2004 IEEE 35th Annual*.

[71] Preiss, B. R. (1999), *Data Structures and Algorithms with Object-Oriented Design Patterns in Java*, John Wiley & Sons, Inc., pp. 35 – 66 & pp. 528 – 532.

[72] Zwolinski, M. (2000), *Digital System Design with VHDL*, Pearson Education Limited, pp. 77 & 110.

[73] Kar, B. K. and D. K. Pradhan (1993), "A new algorithm for order statistic and sorting," *Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on],* vol. 41, pp. 2688-2694.

[74] Henning, P., H. du T. Mouton, A. D. le Roux (2003), "Control of 1.5 MW active power filter and regeneration converter for a Spoornet DC Substation," *South African Universities' Power Engineering Conference (SAUPEC),* 2003.

[75] Le Roux, A. D. (2002), Simulation fragment for the determination of a non-instantaneous discrete-time Fourier-transform approximation in Simplorer.

[76] Texas Instruments (1999), "Code Composer User's Guide," Literature Number SPRU296A, Available: http://www.ti.com.

[77] Texas Instruments (1998), "Optimizing C Compiler User's Guide," Literature Number SPRU034H, Available: http://www.ti.com.

Appendix A – Profiling of C-Code

In Appendix A, a time-estimate of the execution time of various algorithms will be obtained for use in the timing analysis of Chapter 4. A look will be taken at both the Preiss [71] model for general algorithm analysis and the Code Composer Profiling tool; a comparison between the respective results will be made.

A.1 THE PREISS MODEL PERFORMANCE ANALYSIS METHODOLOGY

Two types of running time analyses are described in [71]. One is concerned with specifics such as fetch times, storage times, etc. Another type is asymptotic-upper-bound analysis, where the 'Big Oh' characteristic of an algorithm is determined, in general terms. Many standard sorting algorithms are available and often they are classified according to the asymptotic upper bounds of their running times. For example, Binary Insertion Sort, Straight Insertion Sort and Bubble Sort are all $O(n^2)$ [71]; Quick Sort is $O(n \log n)$ [71] and Radix Sort is $O(n)$ [71]. Whenever the exact problem sizes are not known beforehand, the asymptotic upper bounds of algorithm running times are considered for very large problem sizes. These upper bounds are then used to determine whether one algorithm is better than another one. However, as only a small number of items are sorted in the voltage stabilisation algorithm, the asymptotic upper bounds of running times are of no consequence.

The Preiss model [71] is concerned with fetch times, storage times, etc. and attempts to analyse algorithm performance by allocating a constant time to each fundamental operation in a high-level programming language.

Axiom 1	The time required to fetch an operand from memory is a constant, T_{fetch}, and the time required to store a result in memory is a constant, T_{store}.
Axiom 2	The times required to perform elementary arithmetic operations, such as addition, subtraction, multiplication, division and comparison, are all constants. These times are denoted by T_+, T_-, T_*, $T_/$ and $T_<$, respectively.
Axiom 3	The time required for the address calculation implied by an array subscripting operation, for example, a[i], is a constant, $T_{[\cdot]}$. This time does not include the time to compute the subscript expression, nor does it include the time to access (i.e. fetch or store) the array element.

Table A.1 *The Preiss model for algorithm performance analysis [71]*

The time allocations for each fundamental operation in defined in Table A.1.

A.1.1 PERFORMANCE ANALYSIS RESULTS

Sorting algorithms from [71] and other algorithms relevant to the active filter controller described in this thesis, will be evaluated on a fetch time, storage time, etc basis, in order to find the minimum DSP sorting algorithm running time. The sorting algorithms listed in this thesis are described in detail by Preiss in [71].

RADIX SORT

Numbers of base two are used in the Radix sort example, to ensure an easy realisation through shifting operations and bit masking operations. Note that the radix is chosen by specifying the value of r in the first line. In the code fragment above, a value of $r = 8$ is used, implying a radix of $R = 2^8 = 256$. The number of passes is then determined from the R-value: it is the ceiling value of the number of input bits divided by the number of bits in the radix – r. The number 2^{10} is used when determining the number of passes, as the input from the ADC can only be 10 bits wide.

Statement	Code	Running time
36(a)	i=0	$(T_{fetch}+T_{store})$
36(b)	i<p	$(2T_{fetch} + T_{<})(p)$
36(c)	i++	$(2T_{fetch} + T_{store} + T_{+})(p-1)$
37(a)	j=0	$(T_{fetch} + T_{store})(p)$
37(b)	j <R	$(T_{fetch} + T_{store})(p)(R)$
37(c)	j++	$(2T_{fetch} + T_{store} + T_{+})(p)(R-1)$
38	count[j] = 0	$(2T_{fetch} + T_{[.]} + T_{store})(p)(R)$
40(a)	k=0	$(T_{fetch} + T_{store})(p)$
40(b)	k<num_elements	$(2T_{fetch} + T_{<})(num\_elements)(p)$
40(c)	k++	$(2T_{fetch} + T_{+} + T_{store})(num\_elements-1)$
41	++count[unsorted[sorted[k]]>>(r*i) & (R-1)]	$(9T_{fetch} + T_{+} + T_{-} + T_{<<} + T_{\&} + 3T_{[.]} + T_{store})(num\_elements)(p)$
42	tempArray[k] = sorted[k]	$(3T_{fetch} + 2T_{[.]} + T_{store})(num\_elements)(p)$
44	pos=0	$(T_{fetch} + T_{store})(p)$
45(a)	j=0	$(T_{fetch} + T_{store})(p)$
45(b)	j < R	$(2T_{fetch} + T_{<})(R)(p)$
45(c)	j++	$(2T_{fetch} + T_{store} + T+)(R-1)(p)$
46	tmp = pos	$(T_{fetch} + T_{store})(R)(p)$
47	pos += count[j]	$(3T_{fetch} + T_{[.]} + T_{+} + T_{store})(R)(p)$
48	count[j] = tmp	$(2Tfetch + T_{[.]} + T_{store})(R)(p)$
50(a)	k=0	$(T_{fetch} + T_{store})(p)$
50(b)	k < num_elements	$(2T_{fetch} + T_{<})(num\_elements)(p)$
50(c)	++k	$(2T_{fetch} + T_{store} + T_{+})(num\_elements-1)(p)$
51	j = unsorted[tempArray[k]] >> (r*i) &(R-1)	$(7T_{fetch} + 2T_{[.]} + T_{>>} + T_{\&} + T_{*} + T_{-} + T_{store})(num\_elements)(p)$
52	sorted[count[j]+1]=tempArray[k]	$(T_{fetch} + 3T_{[.]} + T_{+} + T_{store})(num\_elements)(p)$

Table A.2 *Running time analysis of the radix sort algorithm*

Number of elements	p	R	Evaluated expression	Clock cycles	Running time
6	2	256	sum of terms in Table A.2	15 560	207 µs

Table A.3 *Radix sort analysis for n = 6, for T_{+} , T_{-} , T_{*} , $T_{/}$, $T_{<}$, T_{fetch} , $T_{[.]}$ and T_{store} values taken as '1'.*

It can be seen that the running time of the Radix Sort algorithm is projected to be 207 µs by means of the Preiss model and T_{+} , T_{-} , T_{*} , $T_{/}$, $T_{<}$, T_{fetch} , $T_{[.]}$ and T_{store} values taken as '1'.

INSERTION SORT

Statement	Code	Running time (worst case)
16(a)	i=0	$T_{fetch} + T_{store}$
16(b)	i < num_elements	$(num\_elements-1)(2T_{fetch} + T_{<})$
16(c)	i++	$(num\_elements-2)(2T_{fetch} + T_{+} + T_{store})$
17(a)	j=i	$(num\_elements-1)(2T_{fetch} + T_{store})$
17(b)	((j>0) && (unsorted[sorted_indices[j-1]]> unsorted[sorted_indices[j]]))	$(num\_elements-1)(2Tf_{etch} + T_{<})$
17(c)	--j	$(I)(7T_{fetch} + T_{\&\&} + 4T_{[.]} + T_{<}) + (I-1)(2T_{fetch} + T_{-} + T_{store})$
18	temp = sorted_indices[j];	$(I)(3T_{fetch} + T_{store} + T_{[.]})$
19	sorted_indices[j] = sorted_indices[j-1];	$(I)(5T_{fetch} + T_{store} + 2T_{[.]} + T_{-})$
20	sorted_indices[j-1] = temp	$(I)(4T_{fetch} + T_{store} + T_{[.]} + T_{-})$

Where I is the total number of times that the inner loop (lines 18 to 20) runs
and the line numbers refer to the C-code of the *insertionSortArray* function in the *sorting.c* listing.

Table A.4 *Running time analysis of the Insertion Sort algorithm*

The quantity I represents the number of iterations of the inner loop (lines 18-20): the worst-case number is given by 15 for a 6-number sort.

Number of elements	Evaluated expression	I (worst-case estimation)	Clock cycles	Running time
6	$T_{fetch} + T_{store} + (num\_elements-1)(2T_{fetch} + T_{<}) + (num\_elements-2)(2T_{fetch} + T_{+} + T_{store}) + (num\_elements-1)(2T_{fetch} + T_{store}) + (num\_elements-1)(2Tf_{etch} + T_{<}) + (I)(7T_{fetch} + T_{\&\&} + 4T_{[.]} + T_{<}) + (I-1)(2T_{fetch} + T_{-} + T_{store}) + (I)(3T_{fetch} + T_{store} + T_{[.]}) + (I)(5T_{fetch} + T_{store} + 2T_{[.]} + T_{-}) + (I)(4T_{fetch} + T_{store} + T_{[.]} + T_{-})$	1+2+3+ 4+5=15	629	8,39 µs

Table A.5 *Insertion Sort analysis for T+ , T- , T* , T/ , T< , Tfetch , T[.] and Tstore values taken as '1'.*

It can be seen that the running time is projected to be 8,39 µs by means of the Preiss model and T_{+} , T_{-} , T_{*} , $T_{/}$, $T_{<}$, T_{fetch} , $T_{[.]}$ and T_{store} values taken as '1'.

BUBBLE SORT

Statement	Code	Running time (worst case)
61(a)	i = num_elements	$T_{fetch} + T_{store}$
61(b)	i > 1	$(2T_{fetch} + T_{<})(num\_elements-1)$
61(c)	i--	$(2T_{fetch} + T_{+})(num\_elements-2)$
62(a)	j = 0	$(T_{fetch} + T_{store})\ (num\_elements-1)$
62(b)	j < i-1	$(3T_{fetch} + T_{-} + T_{<})(J)$
62(c)	j++	$(2T_{fetch} + T_{+})(J+2-num\_elements)$
63	if (unsorted[sorted_indices[j]] > unsorted[sorted_indices[j+1]])	$(2T_{[.]} + 7T_{fetch} + T_{+} + T_{<})(J)$
64	temp = sorted_indices[j];	$(2T_{fetch} + T_{[.]} + T_{store})(I)$
65	sorted_indices[j] = sorted_indices[j+1];	$(4T_{fetch} + 2T_{[.]} + T_{+} + T_{store})(I)$
66	sorted_indices[j+1] = temp	$(3T_{fetch} + T_{store} + T_{[.]} + T_{+})(I)$

The line numbers refer to the C-code of the insertionSortArray function in the sorting.c listing.

Table A.6 *Running time analysis of the Bubble Sort algorithm*

Here it is assumed that the *if*-instruction will execute every cycle. In addition, the jump operation associated with the if (T_{jmp}) is taken as only one clock cycle, although, it may at times need four clock cycles to execute.

Number of elements	Evaluated expression	I (worst-case estimation)	J (worst-case estimation)	Clock cycles	Running time
6	$T_{fetch} + T_{store} + (2T_{fetch} + T_{<})(num\_elements-1) + (2T_{fetch} + T_{+})(num\_elements-2) + (T_{fetch} + T_{store})\ (num\_elements-1) + (3T_{fetch} + T_{-} + T_{<})(J) + (2T_{fetch} + T_{+})(J+2-num\_elements) + (2T_{[.]} + 7T_{fetch} + T_{+} + T_{<})(J) + (2T_{fetch} + T_{[.]} + T_{store})(I) + (4T_{fetch} + 2T_{[.]} + T_{+} + T_{store})(I) + (3T_{fetch} + T_{store} + T_{[.]} + T_{+})(I)$	1+2+3+4+5 = 15	1+2+3+4+5 =15	579	7,72 µs

Table A.7 *Bubble Sort analysis, for T_{+}, T_{-}, T_{*}, $T_{/}$, $T_{<}$, T_{fetch}, $T_{[.]}$ and T_{store} values taken to be '1'.*

The values of the quantities I and J are computed as 15 both, for the worst-case analysis. The quantity I represents the number of swaps that occur (lines 64-66). A worst-case scenario for an ascending sort is an input set that is sorted in descending order; here a swap will take place during every single iteration of the inner loop (15 times for a 6-number sort). The quantity J represents the number of iterations of the inner loop (especially line 63, which will run every iteration); again the worst-case number is given by 15 for a 6-number sort.

It can be seen that the running time of the Bubble sort algorithm is projected to be 7,72 µs by means of the Preiss model and T_+ , T_- , T_* , $T_/$, $T_<$, T_{fetch} , $T_{[.]}$ and T_{store} values taken as '1'.

A HARD-CODED SORTING ALGORITHM

Statement	Code	Running time
90	if ((a[[0] >= a[1]) &&	$(T_{jmp} + 4T_{fetch} + T_< + T_{\&\&})R$
91	(a[0] >= a[2]) &&	$(4T_{fetch} + T_< + T_{\&\&})R$
92	(a[0] >= a[3]) &&	$(4T_{fetch} + T_< + T_{\&\&})R$
93	(a[0] >= a[4]) &&	$(4T_{fetch} + T_< + T_{\&\&})R$
94	(a[0] >= a[5]))	$(4T_{fetch} + T_<)R$
95	sorted_indices[index_count] = 0	$(3T_{fetch} + T_{[.]} + T_{store})R$
96	a[0] = 0;	$(3T_{fetch} + T_{[.]} + T_{store})R$

where R is the number of times that the specific code fragment is executed
The line numbers refer to the C-code of the *insertionSortArray* function in the *sorting.c* listing.

Table A.8 *Running time analysis of the hard-coded algorithm*

Only the lines 90-96 are analysed, as the group of instructions are repeated another five times in the hard-coded sorting algorithm. If it is assumed that all the numbers differ from each other (a realistic assumption for a worst-case running time estimation) it can also be assumed that the various blocks will execute 1, 2, 3, 4, 5 and 6 times respectively. This will happen when a number is identified as the largest number, and when the program exits the current block of code. Hence, a worst-case value of R is taken to be – for the 6-number hard-coded algorithm – 1+2+3+4+5+6 = 21. Assuming that each T_+, T_-, T_*, $T_/$, $T_<$, T_{fetch}, $T_{[.]}$ and T_{store} represents one clock cycle, an estimation of the worst-case running time can now be computed.

Number of elements	Evaluated expression	R (worst-case estimation)	Clock cycles	Running time
6	$6*[(T_{jmp} + 4T_{fetch} + T_< + T_{\&\&})R + (4T_{fetch} + T_< + T_{\&\&})R + (4T_{fetch} + T_< + T_{\&\&})R + (4T_{fetch} + T_< + T_{\&\&})R + (4T_{fetch} + T_<)R + (3T_{fetch} + T_{[.]} + T_{store})R + (3T_{fetch} + T_{[.]} + T_{store})R]$	1+2+3+4 +5+6=21	840	11,2 µs

Table A.9 *Analysis of hard-coded algorithm, for T_+ , T_- , T_* , $T_/$, $T_<$, T_{fetch} , $T_{[.]}$ and T_{store} values taken to be '1'.*

It can be seen that the running time is projected to be 11,2 µs by means of the Preiss model when T_+ , T_- , T_* , $T_/$, $T_<$, T_{fetch} , $T_{[.]}$ and T_{store} values are taken as '1'.

A.2 CODE COMPOSER PROFILING

The Code Composer environment has built-in functionality for the profiling of DSP code. A detailed method to increase the profiling accuracy is explained in [76]. This method is summarised in Table A.11 and minimises the measurement errors that are introduced through pipeline flushing, missing pipeline conflicts and extra program fetches when the processor is halted for profiling purposes.

1. Set a breakpoint at point C that is at least four instructions past point B in the program flow.
2. Set a breakpoint at point A and run to that breakpoint.
3. Reset the clock, and remove the breakpoint at point A.
4. Run to the breakpoint at point C and record the value of the CLK variable, which represents the cycle count between points A and C.
5. Repeat steps 2 through 4 using point B instead of point A. Make sure your program is in the same state as it was for measuring the cycles between point A and point C.
6. Subtract the cycle count between points B and point C from the cycle count between point A and point C. This eliminates the measurement errors introduced by stopping the processor at point C.

Table A.10 *Profiling method to obtain accurate instruction cycle counts between two points, A and B in the program [76]*

The question whether code can simultaneously be debugged and optimised is answered in [77]. Usually the –g option (to generate symbolic debugging information) disables many optimisations. However, when symbolic debugging is used in conjunction with the optimiser, the –mn option on the cl30 compiler must be used. This re-enables the optimisations disabled by –g [77].

In the results of Tables A.12 - A.15, an additional –a directive was used, for ‘assume aliased variables’, resulting in a –man directive.

The various actions of the Code Composer optimiser are summarised in Table A.11.

Level 0	*Level 1*
• performs control-flow-graph simplification • allocates variables to registers • performs loop rotation • eliminates dead code • simplifies expressions and statements • expands calls to functions declared as inline	Performs all level 0 features, plus: • performs local copy/constant propagation • removes local dead assignments • eliminates local common sub expressions
Level 2	*Level 3*
Performs all level 1 features, plus: • performs loop optimizations • eliminates global common sub expressions • eliminates global redundant assignments • converts array references in loops to incremented pointer form • performs loop unrolling	Performs all level 2 features, plus: • removes all functions that are never called • simplifies functions that have return values that are never used • expands calls to small functions inline • reorders function definitions so the attributes of called functions are known when the caller is optimized • propagates arguments into function bodies when all call sites pass the same value in the same argument position • identified file-level variable characteristics

Table A.11 *Optimisations performed by the Code Composer optimiser [77]*

In the results of Tables A.12 – A.15, the phrase 'fully optimised' refers to code that has been optimised at Level 3.

Algorithm	Point a marker	Point b marker	Point c marker	a to c [clock cycles]	b to c [clock cycles]	a to b = (a to c) – (b to c) [clock cycles]	execution time
Celanovic SVM calculation of output levels	a1	b1	c1	736	12	724	9,65 µs
Proposed algorithm, Part B	a2	b2	c2	264 875	17532	247 343	3,30 ms
Proposed algorithm, Part A	a3	b3	c3	18330	802	17 528	233,71 µs
Donzel and Bornard algorithm, Part B	a8	b8	c8	529	196	333	4,44 µs
Donzel and Bornard algorithm, Part A	a9	b9	c9	786	337	449	5,99 µs
Hard-coded sort	a4	b4	c4	4167	3 280	887	11,83 µs
Bubble sort	a5	b5	c5	3262	2914	348	4,64 µs
Radix sort	a6	b6	c6	2895	332	2563	34,17 µs
Insertion sort	a7	b7	c7	316	6 #	310	4,13 µs

Table A.12 *Profile of C-code, fully inline but with no optimisations for the data set {100, 20, 60, 30, 45, 70}. The compiler command line settings was set to (-gkqqs -v33 -ma -alsx -x2 -frC:\Source_Dir)*

Algorithm	Point a marker	Point b marker	Point c marker	a to c [clock cycles]	b to c [clock cycles]	a to b = (a to c) – (b to c) [clock cycles]	execution time
Celanovic SVM calculation of output levels	a1	b1	c1	727	12	715	9,53 µs
Proposed algorithm, Part B	a2	b2	c2	230 201	17 424	212 777	2,84 ms
Proposed algorithm, Part A	a3	b3	c3	12 121 *	802	11 319	150,92 µs
Donzel and Bornard algorithm, Part B	a8	b8	c8	777	337	440	5,87 µs
Donzel and Bornard algorithm, Part A	a9	b9	c9	529	196	333	4,44 µs
Hard-coded sort	a4	b4	c4	4152	3271	881	11,75 µs
Bubble sort	a5	b5	c5	3253	2905	348	4,64 µs
Radix sort	a6	b6	c6	2887 *	325	2562	34,16 µs
Insertion sort	a7	b7	c7	309	6 #	303	4,04 µs

Table A.13 *Profile of C-code, fully inline and fully optimised for the data set {100, 20, 60, 30, 45, 70}. The compiler command line settings was set to (-gkqqs -v33 -man -alsx -o3 -x2 -frC:\Source_Dir)*

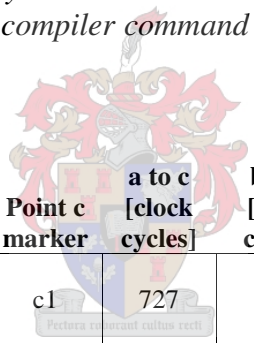
* The point a6 has been moved backwards by one assembler statement in this profiling measurement, to enable breakpoint functionality at point a6. Presumably the breakpoint interferes with the RPTS (repeat single instruction) early on in the Radix Sort function, resulting in system instability.

\# The point c7 has been moved forward by one assembler statement in these profiling measurements, to ensure that point c7 is 4 assembler instructions past the point b[7].

The data set used for the Donzel and Bornard algorithm is {100, 20, 60, 30, 45, 70, 71}

Algorithm	Point a marker	Point b marker	Point c marker	a to c [clock cycles]	b to c [clock cycles]	a to b = (a to c) – (b to c) [clock cycles]	execution time
Celanovic SVM calculation of output levels	a1	b1	c1	736	12	724	9,65 µs
Proposed algorithm, Part B	a2	b2	c2	246 494	17 532	228 962	3,05 ms
Proposed algorithm, Part A	a3	b3	c3	18 335	807	17 528	233,71 µs
Donzel and Bornard algorithm, Part B	a8	b8	c8	1 010	337	673	8,97 µs
Donzel and Bornard algorithm, Part A	a9	b9	c9	529	196	333	4,44 µs
Hard-coded sort	a4	b4	c4	4538	3420	1 118	14,91 µs
Bubble sort	a5	b5	c5	3402	3038	364	4,85 µs
Radix sort	a6	b6	c6	3020	456	2564	34,19 µs
Insertion sort	a7	b7	c7	440	6	434	5,79 µs

Table A.14 *Profile of C-code, fully inline but with no optimisations for the worst-case data set {100, 70, 60, 45, 30, 20}. The compiler command line settings was set to (-gkqqs -v33 -ma -alsx -x2 -frC:\Source_Dir)*



Algorithm	Point a marker	Point b marker	Point c marker	a to c [clock cycles]	b to c [clock cycles]	a to b = (a to c) – (b to c) [clock cycles]	execution time
Celanovic SVM calculation of output levels	a1	b1	c1	727	12	715	9,53 µs
Proposed algorithm, Part B	a2	b2	c2	233 663	17532	216 131	2,88 ms
Proposed algorithm, Part A	a3	b3	c3	18 333	805	17 528	233,71 µs
Donzel and Bornard algorithm, Part B	a8	b8	c8	988	337	651	8,68 µs
Donzel and Bornard algorithm, Part A	a9	b9	c9	529	196	333	4,44 µs
Hard-coded sort	a4	b4	c4	4515	3403	1 112	14,83 µs
Bubble sort	a5	b5	c5	3385	3021	364	4,85 µs
Radix sort	a6	b6	c6	3003	441	2562	34,16 µs
Insertion sort	a7	b7	c7	425	6	419	5,59 µs

The worst-case data set used for the Donzel and Bornard algorithm is {100, 70, 60, 45, 30, 20, 10}

Table A.15 *Profile of C-code, fully inline and fully optimised for the worst-case data set {100, 70, 60, 45, 30, 20}. The compiler command line settings was set to (-gkqqs -v33 -man -alsx -o3 -x2 -frC:\Source_Dir)*

The results of the Preiss performance analysis method can now be compared to the profiling results that were generated with Code Composer. Non-optimised compiled code of the Code Composer environment is used in the comparison, as the Preiss model was applied to C-code that was not compiled.

	Preiss model performance results	Code Composer profiling results
Hard-coded sort	11,2 µs	14,91 µs
Bubble sort	7,72 µs	4,85 µs
Radix sort	207 µs	34,19 µs
Insertion sort	8,39 µs	5,79 µs

Table A.16 *Comparison of the Preiss model results and the worst-case non-optimised Code Composer profiling results*

It can be seen that the performance data generated by the Preiss model differs somewhat from the Code Composer profiling results. The Preiss model does give a conservative estimation of the running time; this can be attributed to the generalised approach of the Preiss model. For example, the operand *fetch* and *store* time allocation (as defined in Table A.1) is done with the assumption that all variables will be copied from internal memory to a DSP register before an addition, comparison, etc. is performed; this may lead to a conservative time estimation in cases where compiled or assembler code using immediate or indirect addressing is generated.

The worst-case running times of Table A.15 (measured with the Code Composer Profiler for optimised DSP code) represent more accurate running times and are used in Chapter 4 for the timing analysis.

A.3 C-CODE SOURCE FOR PROFILING

The C-code of the algorithms shown in Tables A.12 – A.15 will be listed in this subsection. A summary of the various algorithms, the corresponding C-functions and the relevant program listings is given in Table A.16.

Algorithm	Corresponding C-functions	File listing
Celanovic SVM calculation of output levels	*calc_SVPWM*	*spwm.c*
Proposed algorithm, Part B	*bubbleSortArray* *getRatings* *measured_Vc*	*proposed.c*
Proposed algorithm, Part A	*testOutputLevel* *testMinSwitching* *getMaxIndex*	*proposed.c*
Donzel and Bornard algorithm, Part B	*sortArray(sorted_Vcell_indices, …* *getSetA* *getSetB* *getSet_Ap_An* *sortArray(Ap, …* *sortArray(An, …*	*DandB.c*
Donzel and Bornard algorithm, Part A	*getSet_Bo_Bz* *sortArray(Bo, …* *sortArray(Bz, …* *concat_Ap_An_Bo_Bz* *sw_fn_specification*	*DandB.c*
Hard-coded sort	*hardCode_6_SortArray*	*sorting.c*
Bubble sort	*bubbleSortArray*	*sorting.c*
Radix sort	*radixSortArray*	*sorting.c*
Insertion sort	*insertionSortArray*	*sorting.c*

Table A.17 *Summary of algorithms and the relevant C-code source*

DANDB.C

```
1    #include <string.h>
2    #include <math.h>
3    #include <stdio.h>
4    #include <stdlib.h>
5    #include <float.h>
6    #include <limits.h>
7    #include <time.h>
8
9    #define cells 7
10   #define states 128 /*  pow(2,7)  */
11
12   inline void sortArray(int sorted_indices[], int unsorted[], int num_elements) {
13   int i;
14   int j;
15   int temp;
16
17       for (i=0; i < num_elements; i++)
```

```
18          for (j=i; ((j>0) && (unsorted[sorted_indices[j-1]]>
19                              unsorted[sorted_indices[j]])); --j) {
20             temp = sorted_indices[j];
21             sorted_indices[j] = sorted_indices[j-1];
22             sorted_indices[j-1] = temp;
23          }
24    }
25
26    inline void getSetA(int *A, int *length_A, int *sorted_Vcell_indices,  int *measured_Vcell, int nfree, int no_of_cells) {
27     /*  formation of set A: the nfree first elements (the parameter nfree is  defined as the number of simultaneous
      commutations) */
28
29    int i;
30       for (i = 0; i < nfree; i++) {
31            *(A+i) = *(sorted_Vcell_indices+i);
32       }
33       *length_A = nfree;
34    }
35
36    inline void getSetB(int *B, int *length_B, int *sorted_Vcell_indices,
37       int *measured_Vcell, int nfree, int no_of_cells) {
38
39       /*  formation of set B: take all other elements                    */
40
41    int i;
42       for (i = nfree; i < no_of_cells; i++) {
43            *(B+i-nfree) = *(sorted_Vcell_indices+i);
44       }
45       *length_B = no_of_cells-nfree;
46    }


47    inline void getSet_Ap_An(int Ap[], int *length_Ap, int An[], int *length_An, int A[], int measured_Vcell[], int length_A) {
48    /*  formation of subset Ap: all the positive elements of A              */
49
50    int i;
51    int Ap_index;
52    int An_index;
53
54       Ap_index = 0;
55       An_index = 0;
56       for (i = 0; i < length_A; i++) {
57          if (measured_Vcell[A[i]] > 0) {
58             Ap[Ap_index] = A[i];
59             Ap_index++;
60          } else {
61             An[An_index] = A[i];
62             An_index++;
63          }
64       }
65       *length_Ap = Ap_index;
66       *length_An = An_index;
67    }
68
69    inline void getSet_Bo_Bz(int Bo[], int *length_Bo, int Bz[], int *length_Bz, int B[],  int length_B, int
      prev_switching_function) {
70    /*  formation of subset Bo: all elements of B corresponding to switching functions previously '1' */
71
```

```
72      int i;
73      int Bo_index;
74      int Bz_index;
75      int bit_mask;
76
77          Bo_index = 0;
78          Bz_index = 0;
79          for (i = 0; i < length_B; i++) {
80
81              bit_mask = 2 << B[i];                   /* bit_mask = pow(2,B[i]) */
82
83              if (bit_mask & prev_switching_function) { /* if corresp. sw.fn. '1' */
84                  Bo[Bo_index] = B[i];
85                  Bo_index++;
86              } else {                            /* if corresp. sw. fn '0' */
87                  Bz[Bz_index] = B[i];
88                  Bz_index++;
89              }
90          }
91          *length_Bo = Bo_index;
92          *length_Bz = Bz_index;
93      }
94
95      inline void concat_Ap_An_Bo_Bz(int concat_vector[], int Ap[], int An[], int Bo[], int Bz[],
96          int length_Ap, int length_An, int length_Bo, int length_Bz,  int output_i_positive) {
97      int i;
98      int count;
99
100         count = 0;
101         if (output_i_positive) {                /* output current positive */
102             for (i = 0; i < length_Ap; i++) {
103                 concat_vector[i] = Ap[i];
104             }
105             count += length_Ap;
106             for (i = 0; i < length_Bo; i++) {
107                 concat_vector[i+count] = Bo[i];
108             }
109             count += length_Bo;
110             for (i=0; i < length_Bz; i++) {
111                 concat_vector[i+count] = Bz[i];
112             }
113             count += length_Bz;
114             for (i=0; i < length_An; i++) {
115                 concat_vector[i+count] = An[i];
116             }
117         } else {                            /* output current negative */
118             for (i = 0; i < length_An; i++) {
119                 concat_vector[i] = An[i];
120             }
121             count += length_An;
122             for (i = 0; i < length_Bo; i++) {
123                 concat_vector[i+count] = Bo[i];
124             }
125             count += length_Bo;
126             for (i=0; i < length_Bz; i++) {
127                 concat_vector[i+count] = Bz[i];
128             }
129             count += length_Bz;
```

```
130             for (i=0; i < length_Ap; i++) {
131               concat_vector[i+count] = Ap[i];
132             }
133         }
134     }
135
136     inline void sw_fn_specification(int *output_sw_fn, int concat_vector[], int output_level) {
137     int i;
138         *output_sw_fn = 0;
139         for (i = 0; i < output_level; i++) {
140             /* concat_vector[i] gee nou die indeks van die sw. fn. wat ons na '1' wil stel */
141
142             (*output_sw_fn) += (2 << concat_vector[i]);
143         }
144     }
145
146     inline void get_abs_values(int *measured_abs_Vcell, int *measured_Vcell, int no_of_cells) {
147     int i;
148         for (i = 0; i< no_of_cells; i++) {
149           *(measured_abs_Vcell+i) = abs( *(measured_Vcell+i) );
150         }
151     }
152
153     main() {
154           int measured_Vcell[cells] = {100, 20, 60, 30, 45, 70, 71};
155           int measured_abs_Vcell[cells];
156           int sorted_Vcell_indices[cells] = {0, 1, 2, 3, 4, 5, 6};
157           int A[cells];
158           int concat_vector[cells];
159           int Ap[cells];
160           int An[cells];
161           int B[cells];
162           int Bo[cells];
163           int Bz[cells];
164           int length_A;
165           int length_Ap;
166           int length_An;
167           int length_B;
168           int length_Bz;
169           int length_Bo;
170           int n_free=1;
171           int prev_switching_function = 3;
172           int output_level = 4;
173           int output_i_positive = 1;
174           int output_sw_fn;
175
176           /* part C: */
177               get_abs_values(measured_abs_Vcell, measured_Vcell, cells);
178           /* part B: */
179               sortArray(sorted_Vcell_indices, measured_abs_Vcell, cells);                    /*a8*/
180               getSetA(A, &length_A, sorted_Vcell_indices, measured_Vcell, n_free,
181                       cells);
182               getSetB(B, &length_B, sorted_Vcell_indices, measured_Vcell, n_free,
183                       cells);
184                        getSet_Ap_An(Ap, &length_Ap, An, &length_An, A, measured_Vcell,
185                       length_A);
186                sortArray(Ap, measured_Vcell, length_Ap);
187                sortArray(An, measured_Vcell, length_An);
```

```
188
189             /* part A: */
190                 getSet_Bo_Bz(Bo, &length_Bo, Bz, &length_Bz, B, length_B,          /*a9*/ /*b8*/
191                         prev_switching_function);
192                 sortArray(Bo, measured_Vcell, length_Bo);
193                 sortArray(Bz, measured_Vcell, length_Bz);
194                 concat_Ap_An_Bo_Bz(concat_vector, Ap, An, Bo, Bz, length_Ap,
195                         length_An, length_Bo, length_Bz, output_i_positive);
196                 sw_fn_specification(&output_sw_fn, concat_vector, output_level);
197
198                 sortArray(sorted_Vcell_indices, measured_abs_Vcell, cells);        /*c8*/ /*b9*/
199                 while (1) {
200                   output_sw_fn++;
201                   }                                                                /*c9 */
202     }
```

PROPOSED.C

```
1       #include <string.h>
2       #include <math.h>
3       #include <stdio.h>
4       #include <stdlib.h>
5       #include <float.h>
6       #include <limits.h>
7       #include <time.h>
8
9       #define cells 7
10      #define states 128 /*  pow(2,7)  */
11
12      inline void testOutputLevel(int *ratings, int *switching_functions, int output_level,
13          int no_of_cells, int no_of_states) {
14      int i;
15      int j;
16      int sum;
17      int bit_mask;
18      int temp_sw_fn;
19
20          bit_mask = 1;
21          for (j=0; j < no_of_states; j++) {
22              sum = 0;
23              temp_sw_fn = *(switching_functions+j);
24
25              for (i=0; i < no_of_cells; i++) {
26                  sum += (temp_sw_fn & bit_mask);
27
28                  temp_sw_fn = temp_sw_fn >> 1;
29              }
30              if (sum != output_level) {
31                  *(ratings+j) = 0;
32              }
33          }
34      }
35
36
37      inline void testMinSwitching(int *ratings, int *switching_functions, int prev_sw_state,
38          int no_of_states, int no_of_cells, int output_level, int prev_output_level) {
39      int i;
40      int j;
```

```
41      int sw_count;
42      int bit_mask;
43      int temp_sw_fn;
44      int sw_count_allowed;
45
46          sw_count_allowed = abs(output_level - prev_output_level);
47          if (sw_count_allowed == 0) {
48            sw_count_allowed = 1;
49          }
50          for (j=0; j < no_of_states; j++) {
51              sw_count = 0;
52              temp_sw_fn = *(switching_functions+j);
53              bit_mask = 1;
54              for (i=0; i < no_of_cells; i++) {
55                  if (( prev_sw_state & bit_mask ) != (temp_sw_fn & bit_mask)) {
56                    sw_count += 1;
57                  }
58                  bit_mask = bit_mask << 1;
59              }
60
61              if ( sw_count_allowed != sw_count ) {
62                *(ratings+j) = 0;
63              }
64          }
65      }
66
67
68      inline int getMaxIndex(int *ratings, int no_of_states) {
69      int i;
70      int max_index = 0;
71
72          for (i = 0; i < no_of_states; i++) {
73              if ( (*(ratings+i)) > *(ratings+max_index) ) {
74                    max_index = i;
75              }
76          }
77
78          return max_index;
79      }
80
81      inline void bubbleSortArray(int sorted_indices[], int unsorted[], int num_elements) {
82      int i;
83      int j;
84      int temp;
85
86          for (i=num_elements; i>1; i--) {
87              for (j=0; j<i-1; j++) {
88                  if (unsorted[sorted_indices[j]] > unsorted[sorted_indices[j+1]]) {
89                    temp = sorted_indices[j];
90                    sorted_indices[j] = sorted_indices[j+1];
91                    sorted_indices[j+1] = temp;
92                  }
93              }
94          }
95      }
96
97      inline void insertionSortArray(int sorted_indices[], int unsorted[], int num_elements) {
98      int i;
```

```
99      int j;
100     int temp;
101
102         for (i=0; i < num_elements; i++)
103             for (j=i; ((j>0) && (unsorted[sorted_indices[j-1]]> unsorted[sorted_indices[j]])); --j) {
104                 temp = sorted_indices[j];
105                 sorted_indices[j] = sorted_indices[j-1];
106                 sorted_indices[j-1] = temp;
107             }
108     }
109
110     inline void sortArray(int sorted_indices[], int unsorted[], int num_elements) {
111     int i;
112     int j;
113     int temp;
114
115         for (i=0; i < num_elements; i++)
116             for (j=i; ((j>0) && (unsorted[sorted_indices[j-1]]> unsorted[sorted_indices[j]])); --j) {
117                 temp = sorted_indices[j];
118                 sorted_indices[j] = sorted_indices[j-1];
119                 sorted_indices[j-1] = temp;
120             }
121     }
122
123     inline void getRatings(int *ratings, int *sw_functions, int *ideal_values, int *measured_Vc,
124                 int *sorted_Vd_indices, int no_of_states, int no_of_cells) {
125     int i,j;
126     int discharge_req;
127     int charge_req;
128     int no_req;
129     int discharge_effect;
130     int charge_effect;
131     int no_effect;
132     int bit_mask;
133     int effect;
134     int temp_sw_fn;
135     int correct;
136     int no_action;
137     int incorrect;
138
139     bit_mask = 3;
140
141         for (i= 0; i < no_of_states; i++) {
142
143             temp_sw_fn = *(sw_functions+i);
144             *(ratings+i) = 0;
145                 for (j= 0; j < no_of_cells-1; j++) {
146                         discharge_req = (( *(measured_Vc + j) ) > ( *(ideal_values +j) ));      /* cap. voltage too high */
147                         charge_req = ((*(measured_Vc + j)) < (*(ideal_values +j)));             /* cap. voltage too low */
148                         no_req = ((*(measured_Vc + j)) == (*(ideal_values +j)));                /* cap voltage just fine! */
149                         effect = (bit_mask & temp_sw_fn);                                       /* bitwise AND: & */
150                         switch (effect) {
151                                 case 0:
152                                         no_effect = 1;
153                                         charge_effect = 0;
154                                         discharge_effect = 0;
155                                         break;
156                                 case 1:
```

```
157                                     no_effect = 0;
158                                     charge_effect = 0;
159                                     discharge_effect = 1;
160                                     break;
161                             case 2:
162                                     no_effect = 0;
163                                     charge_effect = 1;
164                                     discharge_effect = 0;
165                                     break;
166                             case 3:
167                                     no_effect = 1;
168                                     charge_effect = 0;
169                                     discharge_effect = 0;
170                             }
171                                                                             /* effect now: 00, 01, 10 or 11 */
172                     correct = (charge_effect && charge_req) || (discharge_effect && discharge_req) || (no_req);
173                     no_action = (no_effect && charge_req) || (no_effect && discharge_req);
174
175                     incorrect = (charge_effect && discharge_req) ||
176                                             (discharge_effect && charge_req);
177
178                     *(ratings+i) += (correct)*( pow(2,(*(sorted_Vd_indices+j) +no_of_cells-1)) + pow(2,(no_of_cells-1)) )

179                     + (no_action)*( pow(2,(*(sorted_Vd_indices+j) +no_of_cells-1) ) );
180                     temp_sw_fn = temp_sw_fn >> 1;                                   /* right-shift 1 bit */
181             }
182     }
183 }
184
185 inline void getIdealValues(int ideal_bus_value, int *ideal_values, const int no_of_cells) {
186 int i;
187     for (i= 0; i < no_of_cells-1; i++) {
188             *(ideal_values+i) = (i+1)*((float)ideal_bus_value/(float)no_of_cells);
189     }
190 }
191
192 inline void generateSwitchingFunctions(int *switching_functions, int no_of_states) {
193 int i;
194    for (i=0; i < no_of_states; i++) {
195      *(switching_functions+i) = i;
196     }
197 }
198
199 main() {
200     int switching_functions[states];
201     int ratings[states];
202     int ideal_values[cells-1];
203     int measured_Vd[cells-1] = {100, 70, 60, 45, 30, 20};
204     int measured_Vc[cells-1] = {10, 20, 30, 40, 50, 60};
205     int sorted_Vd_indices[cells-1] = {0, 1, 2, 3, 4, 5};
206     int correct_correspondences[cells-1];
207     int no_action_correspondences[cells-1];
208     int incorrect_correspondences[cells-1];
209     int output_switching_function[cells];
210     int output_level = 3;
211     int prev_output_level = 3;
212     int prev_switching = 5;
213     int max_index;
```

```
214         volatile int *max_index_ptr = (int *)0x600009;
215
216
217         /* part C: */
218         getIdealValues(1800, ideal_values, cells);
219         generateSwitchingFunctions(switching_functions, states);
220          /* part B: */
221         bubbleSortArray(sorted_Vd_indices, measured_Vd, cells-1);                    /* a2 */
222         getRatings(ratings, switching_functions, ideal_values,
223         measured_Vc, sorted_Vd_indices, states, cells);
224
225          /* part A: */
226         testOutputLevel(ratings, switching_functions, output_level, cells, states);          /* a3, b2 */
227         testMinSwitching(ratings, switching_functions, prev_switching, states,  cells, output_level, prev_output_level);
228         max_index = getMaxIndex(ratings, states);                                    /* b3, c2 */
229         *max_index_ptr = max_index;
230         *max_index_ptr += 1;
231         while (1) {
232       }                                                                              /* c3 */
233   }
```

SORTING.C


```
1     #include <string.h>
2     #include <math.h>
3     #include <stdio.h>
4     #include <stdlib.h>
5     #include <float.h>
6     #include <limits.h>
7     #include <time.h>
8
9     #define cells 7
10    #define states 128 /*  pow(2,7)  */
11
12    inline void insertionSortArray(int sorted_indices[], int unsorted[], int num_elements) {
13    int i;
14    int j;
15    int temp;
16       for (i=0; i < num_elements; i++)
17          for (j=i; ((j>0) && (unsorted[sorted_indices[j-1]]> unsorted[sorted_indices[j]])); --j) {
18             temp = sorted_indices[j];
19             sorted_indices[j] = sorted_indices[j-1];
20             sorted_indices[j-1] = temp;
21          }
22    }
23
24    inline void radixSortArray(int sorted_indices[], int unsorted[], int num_elements) {
25    #define r 8       /*int r = 8;       */
26    #define R 256   /*int R = 1<<r;              */
27    #define p 2       /*int p = (10+r-1)/r;*/
28    int i;
29    int k;
30    int pos;
31    int tmp;
32    int j;
33    int count[R];
34    int tempArray[6];                                    /* allocate maximum number of elements:6 */
35
```

```
36          for (i = 0; i < p; i++) {
37              for (j = 0; j <R; j++) {
38                  count[j] = 0;
39              }
40              for (k = 0; k < num_elements; k++) {
41                  ++count[unsorted[sorted_indices[k]] >> (r*i) & (R-1)];
42                  tempArray[k] = sorted_indices[k];
43              }
44              pos = 0;
45              for (j =0; j<R; j++) {
46                  tmp = pos;
47                  pos += count[j];
48                  count[j] = tmp;
49              }
50              for (k=0; k<num_elements; ++k) {
51                  j = unsorted[tempArray[k]] >> (r*i)&(R-1);
52                  sorted_indices[count[j]++] = tempArray[k];
53              }
54          }
55      }
56
57      inline void bubbleSortArray(int sorted_indices[], int unsorted[], int num_elements) {
58      int i;
59      int j;
60      int temp;
61          for (i=num_elements; i>1; i--) {
62              for (j=0; j<i-1; j++) {
63                  if (unsorted[sorted_indices[j]] > unsorted[sorted_indices[j+1]]) {
64                      temp = sorted_indices[j];
65                      sorted_indices[j] = sorted_indices[j+1];
66                      sorted_indices[j+1] = temp;
67                  }
68              }
69          }
70      }
71
72      void hardCode_6_SortArray(int sorted_indices[], int unsorted[]) {
73      int i;
74      int j;
75      int temp;
76      int bitmask[6] = {1, 1, 1, 1, 1, 1};
77      int a[6];
78      int index_count;
79
80      /* all sorting: ascending */
81      a[5] = unsorted[5];
82      a[4] = unsorted[4];
83      a[3] = unsorted[3];
84      a[2] = unsorted[2];
85      a[1] = unsorted[1];
86      a[0] = unsorted[0];
87
88      for (index_count = 5; index_count >= 0; index_count--) {
89
90      if ((a[0] >= a[1]) &&
91              (a[0] >= a[2]) &&
92                (a[0] >= a[3]) &&
93                  (a[0] >= a[4]) &&
```

```
94                  (a[0] >= a[5])) {
95                    sorted_indices[index_count] = 0;
96                    a[0] =0;
97      }   else  if ((a[1] >= a[0]) &&
98              (a[1] >= a[2]) &&
99                (a[1] >= a[3]) &&
100               (a[1] >= a[4]) &&
101               (a[1] >= a[5])) {
102                   sorted_indices[index_count] = 1;
103                   a[1] =0;
104     }   else
105       if ((a[2] >= a[0]) &&
106           (a[2] >= a[1]) &&
107             (a[2] >= a[3]) &&
108               (a[2] >= a[4]) &&
109               (a[2] >= a[5])) {
110                   sorted_indices[index_count] = 2;
111                   a[2] =0;
112     }   else
113       if ((a[3] >= a[0]) &&
114           (a[3] >= a[1]) &&
115             (a[3] >= a[2]) &&
116               (a[3] >= a[4]) &&
117               (a[3] >= a[5])) {
118                   sorted_indices[index_count] = 3;
119                   a[3] =0;
120     }   else
121       if ((a[4] >= a[0]) &&
122           (a[4] >= a[1]) &&
123             (a[4] >= a[2]) &&
124               (a[4] >= a[3]) &&
125               (a[4] >= a[5])) {
126                   sorted_indices[index_count] = 4;
127                   a[4] =0;
128     }   else
129       if ((a[5] >= a[0]) &&
130           (a[5] >= a[1]) &&
131             (a[5] >= a[2]) &&
132               (a[5] >= a[3]) &&
133               (a[5] >= a[4])) {
134                   sorted_indices[index_count] = 5;
135                   a[5] =0;
136                 }
137       }
138   }
139
140   main() {
141   int measured_Vd[cells-1] = {100, 70, 60, 45, 30, 20};
142   int sorted_Vd_indices[cells-1] = {0, 1, 2, 3, 4, 5};
143   int i;
144         for (i=0; i< cells-1; i++)
145                 sorted_Vd_indices[i] = i;
146         hardCode_6_SortArray(sorted_Vd_indices, measured_Vd);                    /*a4*/
147
148         for (i=0; i< cells-1; i++)                                               /*b4*/
149                 sorted_Vd_indices[i] = i;
150
151           bubbleSortArray(sorted_Vd_indices, measured_Vd, cells-1);              /*a5*/
```

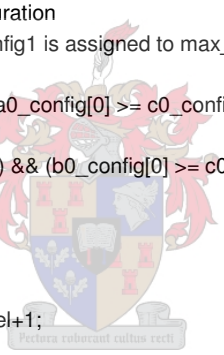
```
152
153         for (i=0; i< cells-1; i++)                                          /*b5*/
154                 sorted_Vd_indices[i] = i;
155
156          radixSortArray(sorted_Vd_indices, measured_Vd, cells-1);           /*a6*/
157
158         for (i=0; i< cells-1; i++)                                          /*b6*/
159                 sorted_Vd_indices[i] = i;
160
161                 insertionSortArray(sorted_Vd_indices, measured_Vd, cells-1);  /*a7*/
162         while (1) {
163                 for (i=0; i< cells-1; i++)                                  /*c4*/ /*c5*/  /*c6*/ /*c5*/ /*b7 */
164                         sorted_Vd_indices[i] += 1;                          /*c7*/
165          }
166     }
```

SPWM.C

```
1       #include <string.h>
2       #include <math.h>
3       #include <stdio.h>
4       #include <stdlib.h>
5       #include <float.h>
6       #include <limits.h>
7       #include <time.h>
8
9       #define cells 7
10      #define states 128 /*  pow(2,7)  */
11
12      int getMinIndex(int *values, int no_of_values) {
13      int i;
14      int min_index = 0;
15         for (i = 0; i < no_of_values; i++) {
16            if ( (*(values+i)) > *(values+min_index) ) {
17                  min_index = i;
18            }
19         }
20         return min_index;
21      }
22
23      inline void calc_SVPWM(int V_alpha_norm, int V_beta_norm, int *a_level, int *b_level, int *c_level) {
24      int i;
25      float Vref_g;
26      float Vref_h;
27
28      int Vul_g;
29      int Vul_h;
30
31      int Vlu_g;
32      int Vlu_h;
33
34      int Vuu_g;
35      int Vuu_h;
36
37      int Vll_g;
38      int Vll_h;
39
40      int upper_tri;
```

```
41
42      int V3_g;
43      int V3_h;
44
45      float levela0_try1;
46      float levelb0_try1;
47      float levelc0_try1;
48
49      float d0;
50      float d1;
51      float d2;
52
53      float offset_0;
54      int config_switchings[7];
55
56      int a0_config[7];
57      int b0_config[7];
58      int c0_config[7];
59
60      int max_configlevel;
61      int no_of_configs;
62
63      int prev_a0_config;
64      int prev_b0_config;
65      int prev_c0_config;
66
67      int index;
68
69          Vref_g = cells*((float)V_alpha_norm - ((float)1)/sqrt(3)*(float)V_beta_norm);
70          Vref_h = cells*2/sqrt(3)*(float)V_beta_norm;
71
72          Vul_g = ceil(Vref_g);
73          Vul_h = floor(Vref_h);
74
75          Vlu_g = floor(Vref_g);
76          Vlu_h = ceil(Vref_h);
77
78          Vuu_g = ceil(Vref_g);
79          Vuu_h = ceil(Vref_h);
80
81          Vll_g = floor(Vref_g);
82          Vll_h = floor(Vref_h);
83
84          /* question: does reference value fall into the upper or lower triangle? */
85          upper_tri = ((Vref_g + Vref_h - Vul_g - Vul_h) > 0);
86
87            if (upper_tri) {
88                    d2 = (float)Vuu_g - Vref_g;
89                    d1 = (float)Vuu_h - Vref_h;
90                    V3_g = Vuu_g;
91                    V3_h = Vuu_h;
92            } else {
93                    d1 = Vref_g - (float)Vll_g;
94                    d2 = Vref_h - (float)Vll_h;
95                    V3_g = Vll_g;
96                    V3_h = Vll_h;
97            }
98
```

```
99          d0 = 1 - d2 - d1;
100
101         /* calculation of basic 3-phase redundant combination */
102         levela0_try1 = 2.0/3.0*(float)V3_g + 1.0/3.0*(float)V3_h;
103         levelb0_try1 = -1.0/3.0*(float)V3_g + 1.0/3.0*(float)V3_h;
104         levelc0_try1 = -1.0/3.0*(float)V3_g - 2.0/3.0*(float)V3_h;
105
106
107         /* the offset_0 value is computed to be the smallest number in the set: levela0_try1, levelb0_try1, levelc0_try1 */
108
109         if ((levela0_try1 <= levelb0_try1) && (levela0_try1 <= levelc0_try1)) {
110                 offset_0 = abs(levela0_try1);
111         } else if ((levelb0_try1 <= levela0_try1) && (levelb0_try1 <= levelc0_try1)) {
112                 offset_0 = abs(levelb0_try1);
113         } else
114                 offset_0 = abs(levelc0_try1);
115
116
117         a0_config[0] = levela0_try1 + offset_0;
118         b0_config[0] = levelb0_try1 + offset_0;
119         c0_config[0] = levelc0_try1 + offset_0;
120
121         /* the highest level in the basic configuration
122                  a0_config1, b0_config1, c0_config1 is assigned to max_configlevel       */
123
124         if ((a0_config[0] >= b0_config[0]) && (a0_config[0] >= c0_config[0])) {
125                 max_configlevel = a0_config[0];
126         } else if ((b0_config[0] >= a0_config[0]) && (b0_config[0] >= c0_config[0])) {
127                 max_configlevel = b0_config[0];
128         } else
129                 max_configlevel = c0_config[0];
130
131         no_of_configs = cells - max_configlevel+1;
132
133         config_switchings[0] = abs(prev_a0_config - a0_config[0]) +
134                                         abs(prev_b0_config - b0_config[0]) +
135                                                 abs(prev_c0_config - c0_config[0]);
136
137         for (i = 1; i < 7; i++) {
138                 a0_config[i] = a0_config[i-1] +1;
139                 b0_config[i] = b0_config[i-1] +1;
140                 c0_config[i] = c0_config[i-1] +1;
141                 config_switchings[i] = abs(prev_a0_config - a0_config[i]) +
142                                         abs(prev_b0_config - b0_config[i]) +
143                                                 abs(prev_c0_config - c0_config[i]);
144         }
145
146         /* now: choose the configuration with the minimum number of switchings */
147         index = getMinIndex(config_switchings, no_of_configs);
148
149         *a_level = a0_config[index];
150         *b_level = b0_config[index];
151         *c_level = c0_config[index];
152 }
153
154 main() {
155 volatile int *a_out_ptr = (int *)0x600009;
156 volatile int *b_out_ptr = (int *)0x60000A;
```



```
157     volatile int *c_out_ptr = (int *)0x60000B;
158     volatile int *V_alpha_norm_ptr = (int *)0x60000C;
159     volatile int *V_beta_norm_ptr = (int *)0x60000D;
160     int V_alpha_norm;
161     int V_beta_norm;
162     int a_out;
163     int b_out;
164     int c_out;
165
166       V_alpha_norm = *V_alpha_norm_ptr;
167       V_beta_norm = *V_beta_norm_ptr;                                   /* a1 */
168
169       calc_SVPWM(V_alpha_norm, V_beta_norm, &a_out, &b_out, &c_out);
170
171       *a_out_ptr = a_out;                                               /* b1 */
172       *b_out_ptr = b_out;
173       *c_out_ptr = c_out;
174       *a_out_ptr++;                                                     /* c1 */
175       *b_out_ptr++;
176       *c_out_ptr++;
177       while (1){
178       }
179     }
```

Appendix B – A Hardware Sorting Architecture

In section 4.5, reference is made to an FPGA-based sorting architecture that is used in the realisation of the capacitor-voltage stabilising controller. Appendix B will cover the derivation aspects thereof.

B.1 AN OVERVIEW OF THE RANK-ORDERING SORTING ALGORITHM

A hardware sorting architecture proposed by Hatirnaz and Leblebici in [63] is used as a basis for an FPGA-based sort of the absolute capacitor error voltages and the determination of the maximum state rating. The sorting algorithm is derived from a rank ordering algorithm originally proposed by Kar and Pradham in [73]. A short overview will be given here.

A rank order filter is a non-linear discrete-time translation invariant filter, used mainly in speech and image processing [73]. A rank order filter calculates the i^{th} order statistic of a set of n elements or put simply, the i^{th} smallest element in the set. The rank-ordering algorithm proposed in [73] is based on an equivalence of a rank-order selection for n-bit numbers and n rank-order selections for 1-bit numbers.

In the rank-ordering algorithm, first the MSBs of all the input numbers are sorted to find the i^{th} ranked bit; then, in each input number, the bits to the right of the MSB are set to the MSB value whenever the MSB differs from the i^{th} ranked bit. This process is repeated for every bit in every input number, starting at the MSB and ending at the LSB.

It can be seen that the algorithm (for k inputs) successively changes every number smaller than the i^{th} ranked number to zero, and changes every number greater than the i^{th} ranked number to 2^k-1.

set of input numbers	1011	1101	0001	0111	0101	**3rd ranked bit**	**3rd ranked output**
1st column (MSB)	**1**011	**1**101	**0**001	**0**111	**0**101	**0**	0xxx
2nd column	1**1**11	1**1**11	0**0**01	0**1**11	0**1**01	**1**	01xx
3rd column	11**1**1	11**1**1	00**0**0	01**1**1	01**0**1	**1**	011x
4th column (LSB)	111**1**	111**1**	000**0**	011**1**	000**0**	**1**	**0111**

Table B.1 *Rank-ordering algorithm, for 5 numbers of 4 bits each, to find the third smallest number [73]*

The principle is illustrated in Table B.1. All digits in boldface in a specific row represent 1-bit numbers that are examined to find the i^{th} ranked 1-bit number. The underlined digits represent digits that have been modified. The i^{th} ranked number (or third smallest number in this example) is built up from the series of i^{th} ranked 1-bit numbers. It can be verified that 0111 is indeed the third smallest element in the set defined in Table B.1.

B.2 THE HATIRNAZ AND LEBLEBICI ARCHITECTURE

The rank-ordering algorithm set out above is utilised in the sorting architecture of Hatirnaz and Leblebici [63]. It can be seen that four stages are needed in Table B.1 to complete the ranking operation for inputs of width four. In general, k stages are needed for inputs of width k, when finding the i^{th} smallest number. These multiple stages are 'recycled' in the architecture of Hatirnaz and Leblebici: by successively re-applying different ranks (values of i), a sorted list of m inputs of width k can be obtained with k hardware stages in $(m + k - 1)$ clock cycles. The multiple hardware stages of the Hatirnaz and Leblebici architecture are evident from the diagrammatic overview in Figure B.12. The logic in each rank-order-filter (ROF) cell determines whether the specific ROF input bit should be modified based on the result of 1-bit ranking operations in the previous hardware stage.

The *majority decision* block in Figure B.12 calculates the i^{th} smallest number of m single-bit inputs in one specific hardware stage. In the general case, the output is determined from the Boolean expression $(sum > m - i)$; an explanatory truth table (for the $m = 4$ case) is set out in Table B.2.

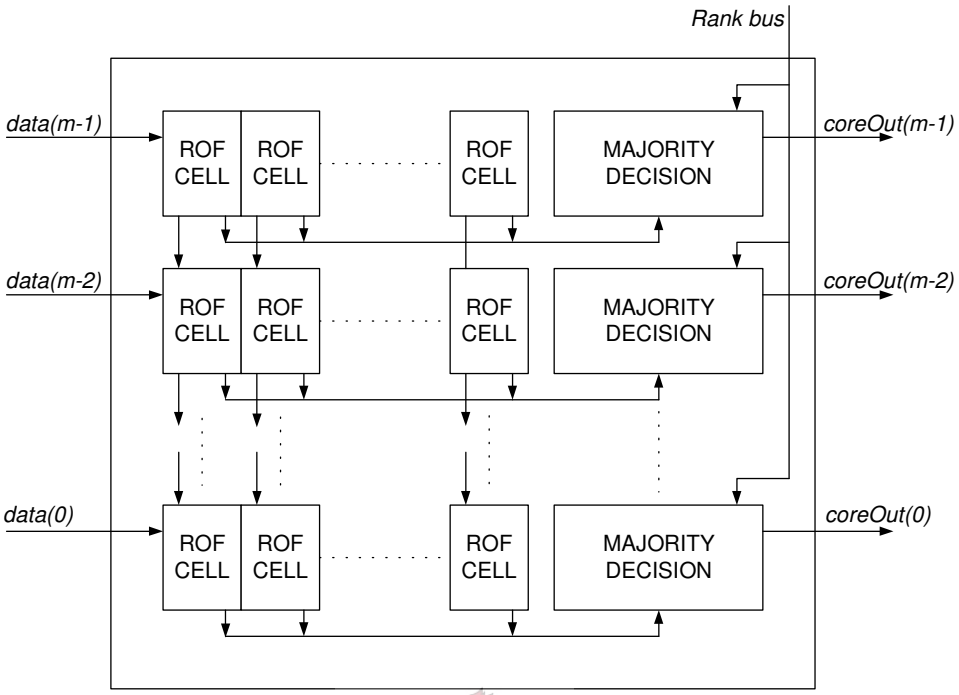


Figure B.12 *Rank order filter core [63]*

					output = (sum > 4 – i)			
inputs				sum	i = 1	i = 2	i = 3	i = 4
0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	1
0	0	1	0	1	0	0	0	1
0	0	1	1	2	0	0	1	1
0	1	0	0	1	0	0	0	1
0	1	0	1	2	0	0	1	1
0	1	1	0	2	0	0	1	1
0	1	1	1	3	0	0	1	1
1	0	0	0	1	0	0	0	1
1	0	0	1	2	0	0	1	1
1	0	1	0	2	0	0	1	1
1	0	1	1	3	0	0	1	1
1	1	0	0	2	0	0	1	1
1	1	0	1	3	0	1	1	1
1	1	1	0	3	0	1	1	1
1	1	1	1	4	1	1	1	1

Table B.2 *Truth table definition of the majority decision block for m = 4 and all corresponding values of i*

In Figure B.13, a diagrammatic representation of the sorting operation is shown for five input numbers of four bits each. Written in binary format, the five numbers are *A1 A2 A3 A4*, *B1 B2 B3 B4*, *C1 C2 C3 C4*, *D1 D2 D3 D4* and *E1 E2 E3 E4*, where *A1..E1*

are the most-significant bits (MSBs) of the inputs. The values *R1*, *R2*, *R3* and *R4* are the ranks that are applied. For instance, in Figure B.13 (a), a rank of *R1* is applied to the bits *E1..A1*, meaning that the $R1^{th}$ smallest element (*C1* in this example) is obtained. In Figure B.13 (b)..(d), new ranks (*R2..R4*) are applied to the input MSBs, and the old rank values are shifted downwards. When the rank values (*R1*, *R2*, *R3*, *R4*) are chosen as (4, 3, 2, 1) respectively –or (*R1*, *R2*, … , *Rm-1, Rm*) are chosen as (*m*, *m*-1, .. ,2, 1) in general, for *m* inputs – the output is sorted in descending order. The staggered output in Figure B.13 indicates the descending order of the inputs: *C*, *A*, *D* and then *B*.

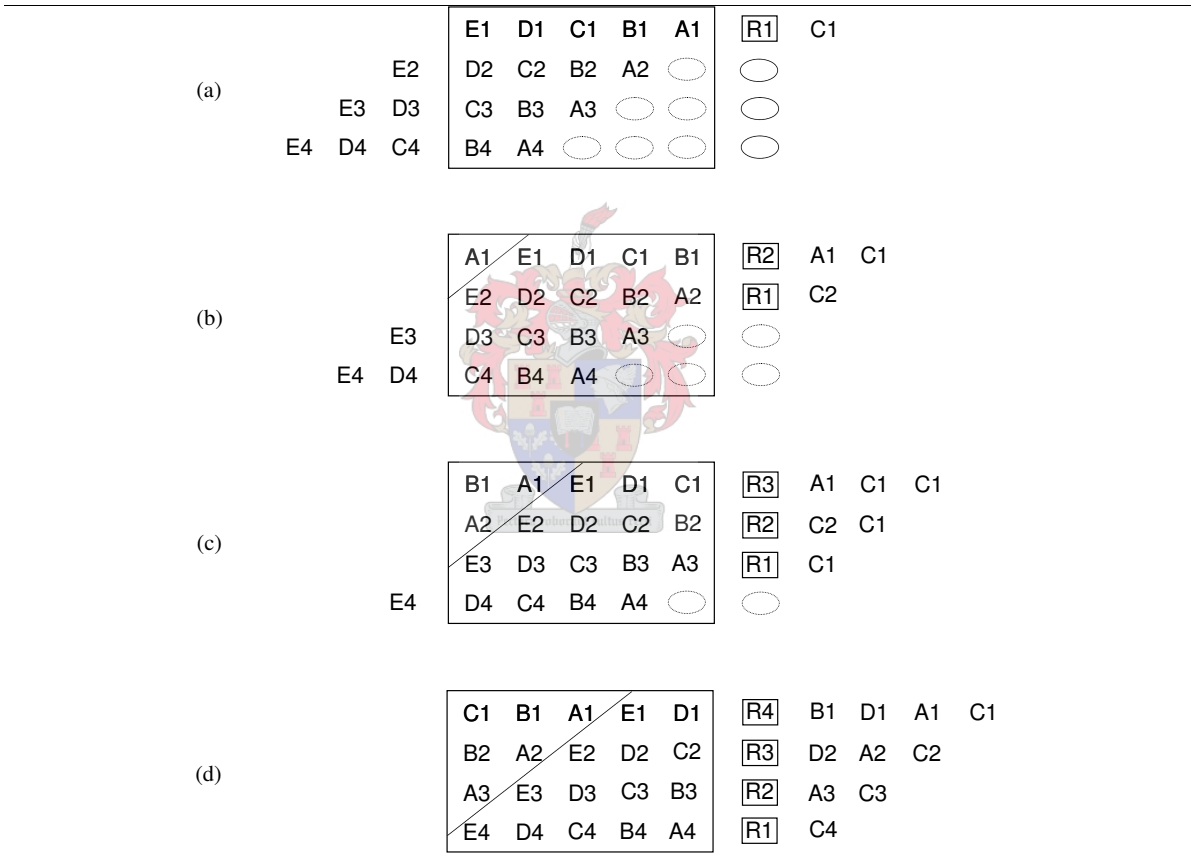


Figure B.*13* *Sorting operation on five 4-bit vectors, with the output vectors in descending order: C, A, D, B as in [63]*

In Figure B.14 the gate-level hardware description of the rank-order-filter cell, defined in Figure B.12, is given [63]. The output *Next_Select* indicates whether the bits of the following hardware levels must be modified. The output *Next_Data* supplies the modified value to the next stage. The *Next_Select* outputs are connected to the *Prev_Select* inputs of

the following hardware stage. Similarly, the *Next_Data* outputs are connected to the *Prev_Data* inputs of the following hardware stage.

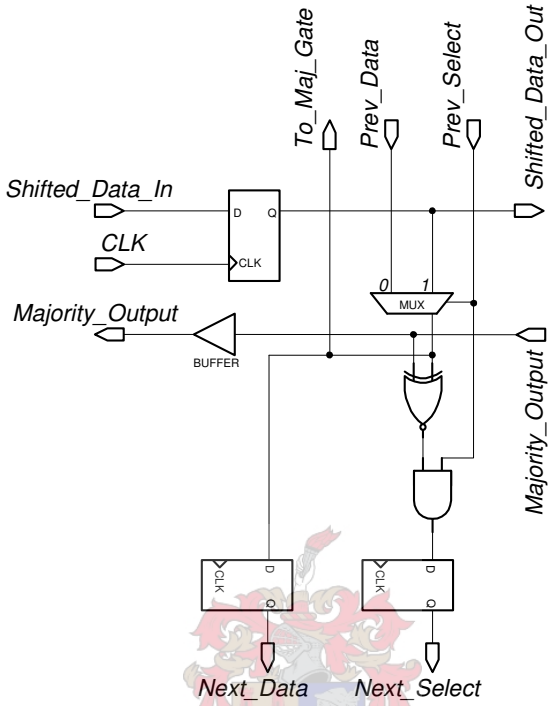


Figure B.14 *Rank order filter cell architecture [63]*

B.3 A DERIVED FPGA-BASED SORTING ARCHITECTURE

The implementation of the sorting architecture can be approached in one of two ways: either the variable-rank approach of Hatirnaz and Leblebici [63] can be implemented directly (with minor adjustments) or, the largest number can be identified repeatedly and already-selected numbers set to zero.

When implementing the first option problems are experienced with the timing requirements of the adders and comparisons required for the variable rank *majority decision* block. An OR gate as a *majority decision* block, as can be used when identifying the largest number in a group of inputs, was identified as a much simpler solution. The second option is implemented, where the largest number is identified repeatedly. FPGA area utilisation efficiency is likely to improve, as a single hardware stage is used; running time is still within the specified limits.

The sorted values themselves, as given by the architecture of Hatirnaz and Leblebici, are of no use in this specific application. Instead, the indices of sorted elements in the input number array are required. The necessary modification is accomplished by means of the *Next_Select* outputs of the final stage in the rank order filter core. The bit-column that is not modified, represents the index of the sorted number at a specific instant. This corresponds to a *Next_Select* value of zero. The *Next_Select* values are checked for zero values at fixed instants to determine the sorted indices.

A priority encoder is used to ensure that only one index is generated – this is essential whenever two or more numbers are equal. Special care was taken to ensure that two or more equal numbers map to different indices. This was achieved by inserting bit-mask functionality between the sorter block and the priority encoder block. In effect, this allows a zero *Next_Select* output to be let through to the priority encoder block only once. As a side-effect this scheme also allows zero-valued inputs to be processed.

The horizontal scrolling action utilised in the architecture of Hatirnaz and Leblebici is not necessary for the intended application; all values are stationary when the enable signal is active. Therefore it will not be implemented.

The modified rank-order-filter cell architecture is shown in Figure B.15. A shift register is used to realise the parallel-in serial-out data. The *Prev_Data* and *Prev_Select* inputs are now looped internally. The initial condition value of the *Prev_Select* line is realised through the *sset* (synchronous-set) input of the D-type flip-flop. No initial value is set for the *Prev_Data* line, as it will have assumed a correct value whenever a zero *Prev_Select* line is processed.

The multiplexer and *Latch_with_sset* combination provides the bit-mask functionality. Similarly to the *Prev_Select* line initialisation, the multiplexer select input is initialised through the *enable_in* and *sset* of the *Latch_with_sset*. The *LPM_Shiftreg* loops through the bits of the input *data_in*. Two flip-flops are inserted in the *LPM_Shiftreg* loop to add two dummy clock cycles to the data-looping cycle. One of the extra clock cycles is needed by the *Priority encoder* block for sampling and output generation; another is needed for the

(synchronous) clearing operation of the input shift register. Thereafter, the cleared and data and original data can again be looped and supplied to the sorter block.

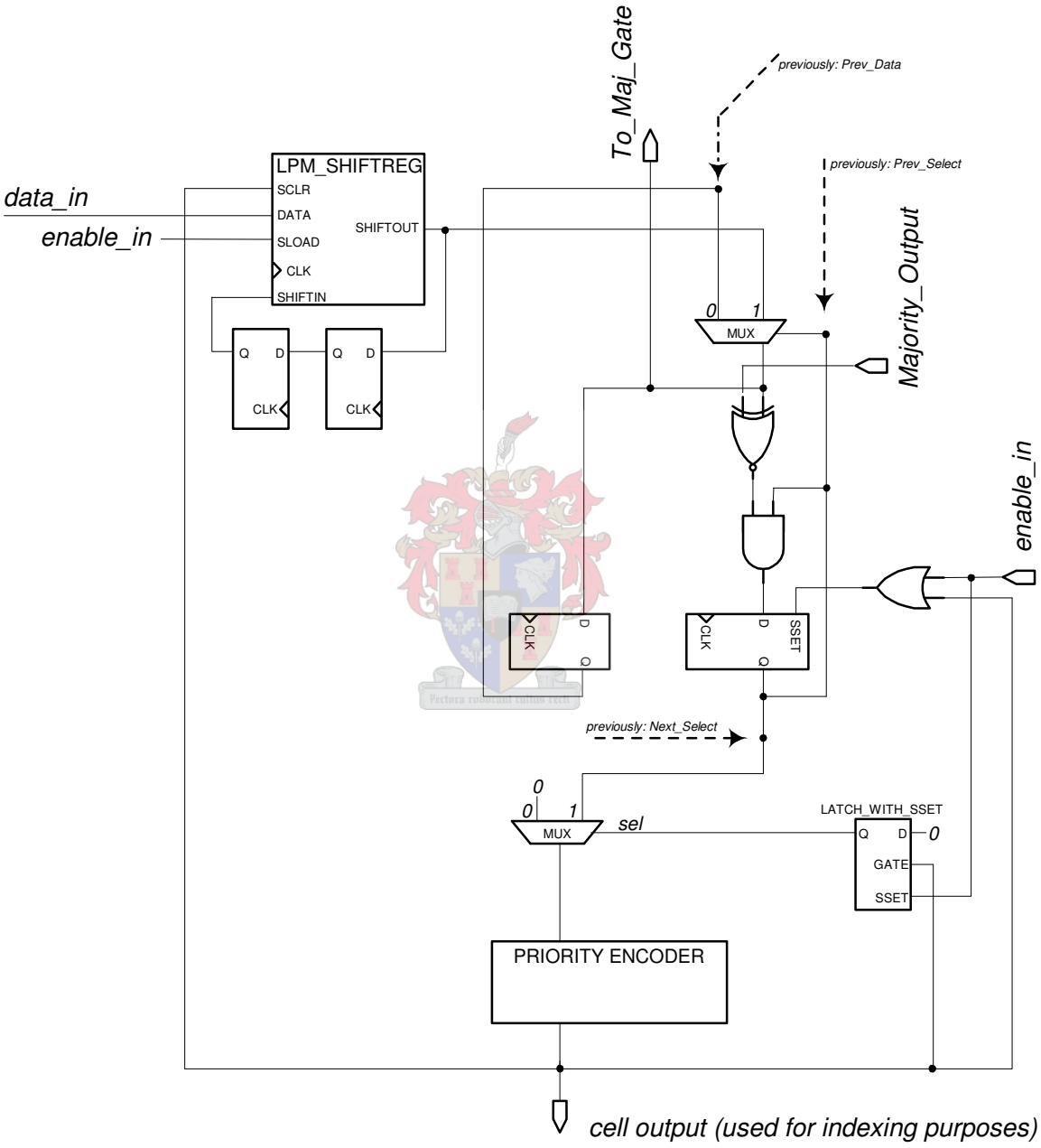


Figure B.15 *Modified rank-order-filter cell architecture*

B.4 RESULTS OF THE FPGA-BASED SORTER

In Figure B.16 the performance of the sorting algorithm that is realised through the *Sorter_ranked* block, is demonstrated. Four arbitrary inputs ('1100011', '1110001','0000000' and '1110001') are processed. The sorted order of indices is available 38 clock cycles after the *enable_in* signal was sampled. Interpretation of the *sorted_out*[3..0][3] value of '1000', for instance, indicates that the value at index 3 of the input vector (*data_in*) is the largest of these. The second largest input is identified through the *sorted_out*[3..0][2] value of '0010' to at index 1 of the input vector: '1110001'. Note that the specific example illustrates the capability of the algorithm to handle zero-valued inputs and also equal inputs.

Figures B.17 – B.19 shows the performance of the full-sorting algorithm for 4, 5, and 6 number inputs respectively.

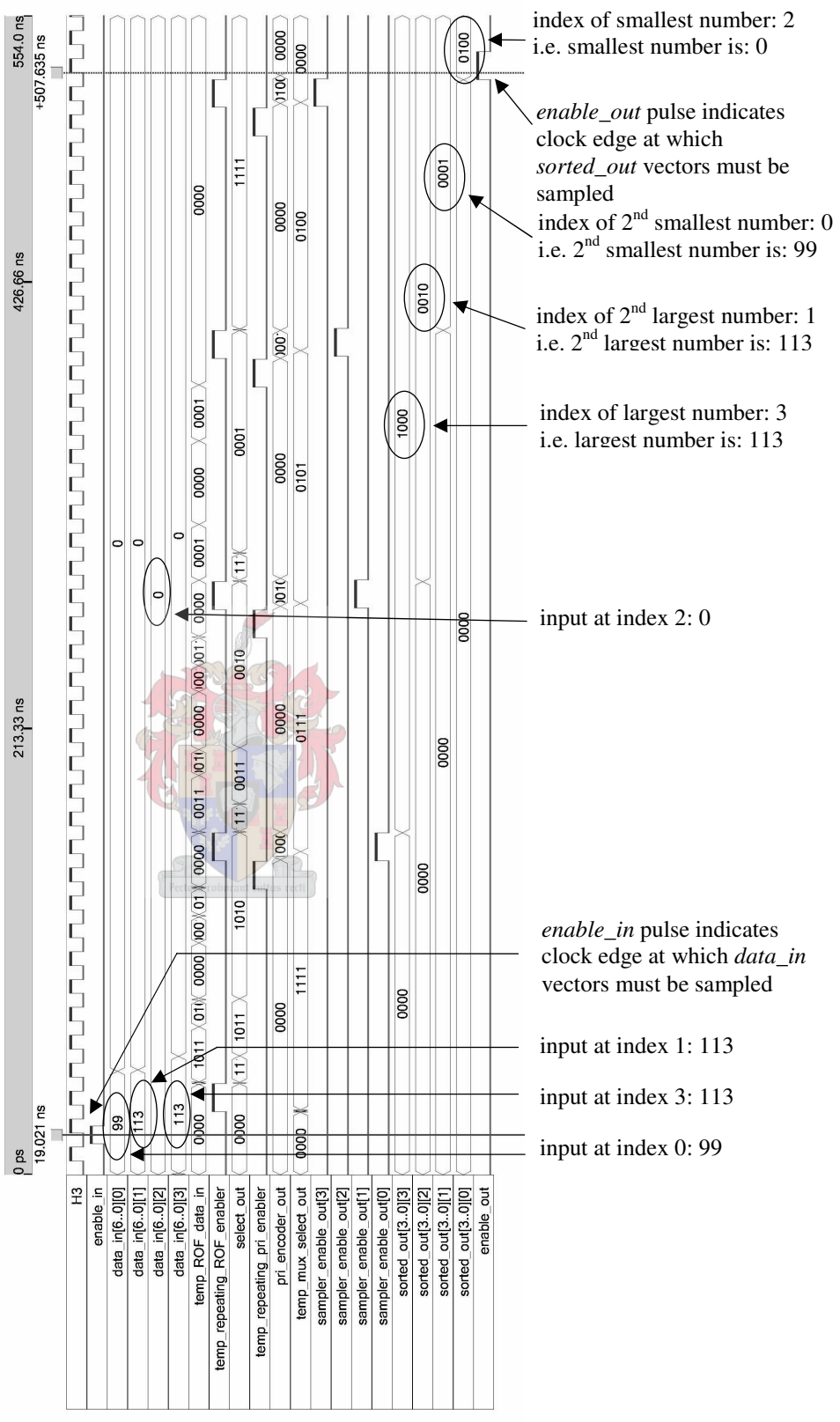


Figure B.16 *Sorter_ranked block output for four arbitrary inputs of 7 bits each. Note that the indices of the sorted outputs may be sampled 38 clock cycles (or 0,51 µs) after the inputs have been sampled*

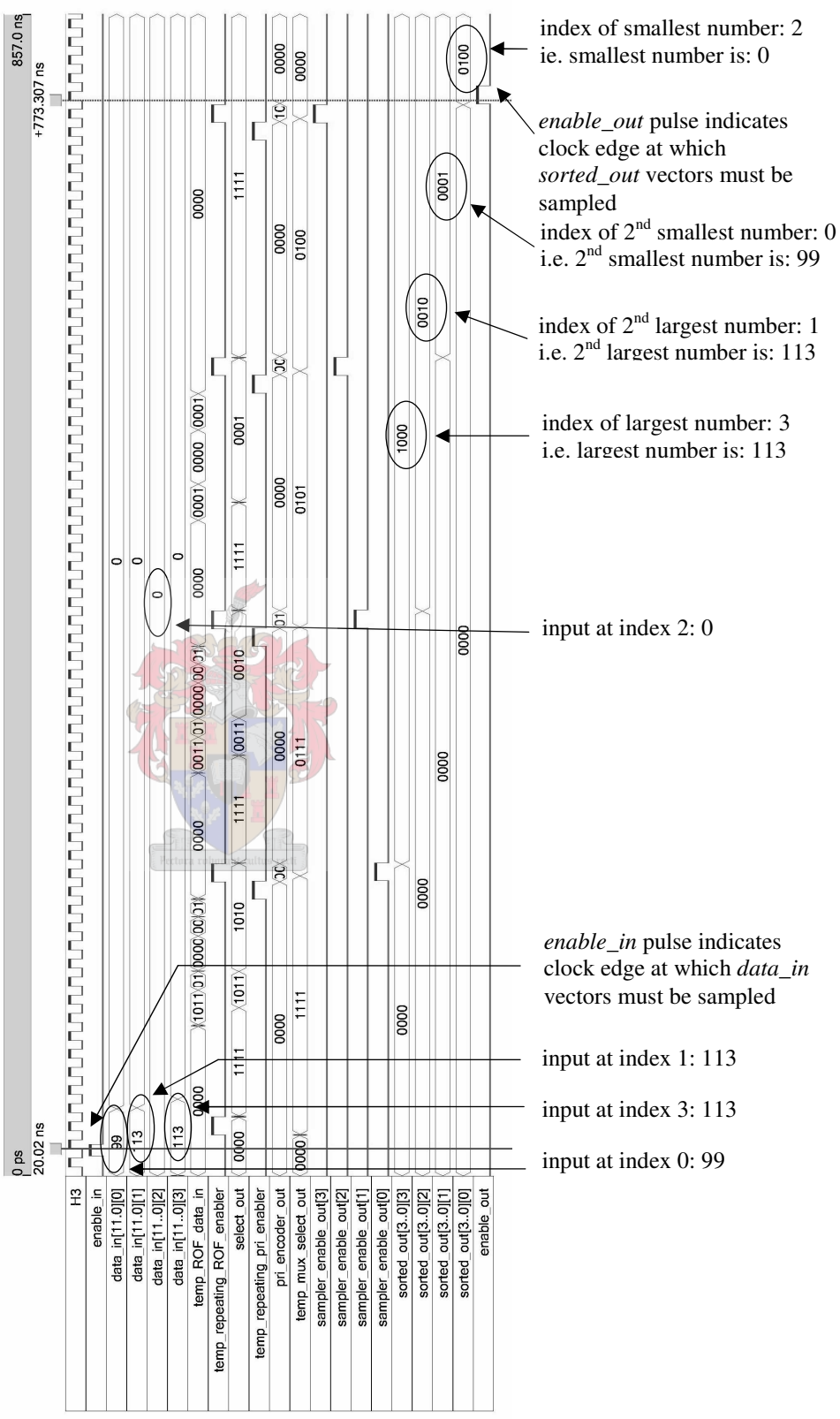


Figure B.17 *Sorter_ranked block output for four arbitrary inputs of 12 bits each. Here the indices of the sorted outputs are available for sampling 58 clock cycles (or 0,77 µs) after the inputs have been sampled.*

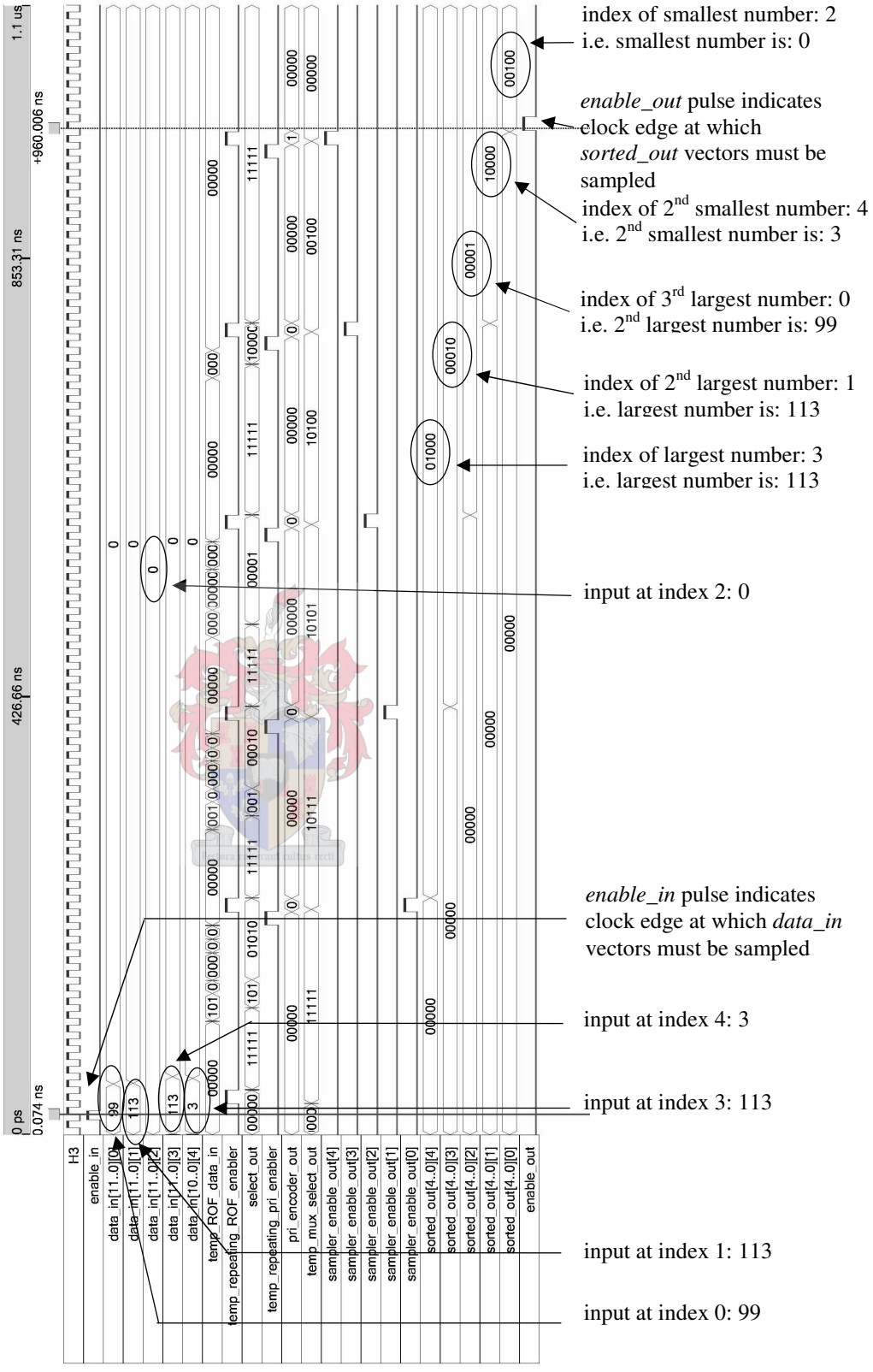


Figure B.18 *Sorter_ranked block output for five arbitrary inputs of 12 bits each. The indices of the sorted outputs is available 72 clock cycles (or 0,96 µs) after the inputs have been sampled.*

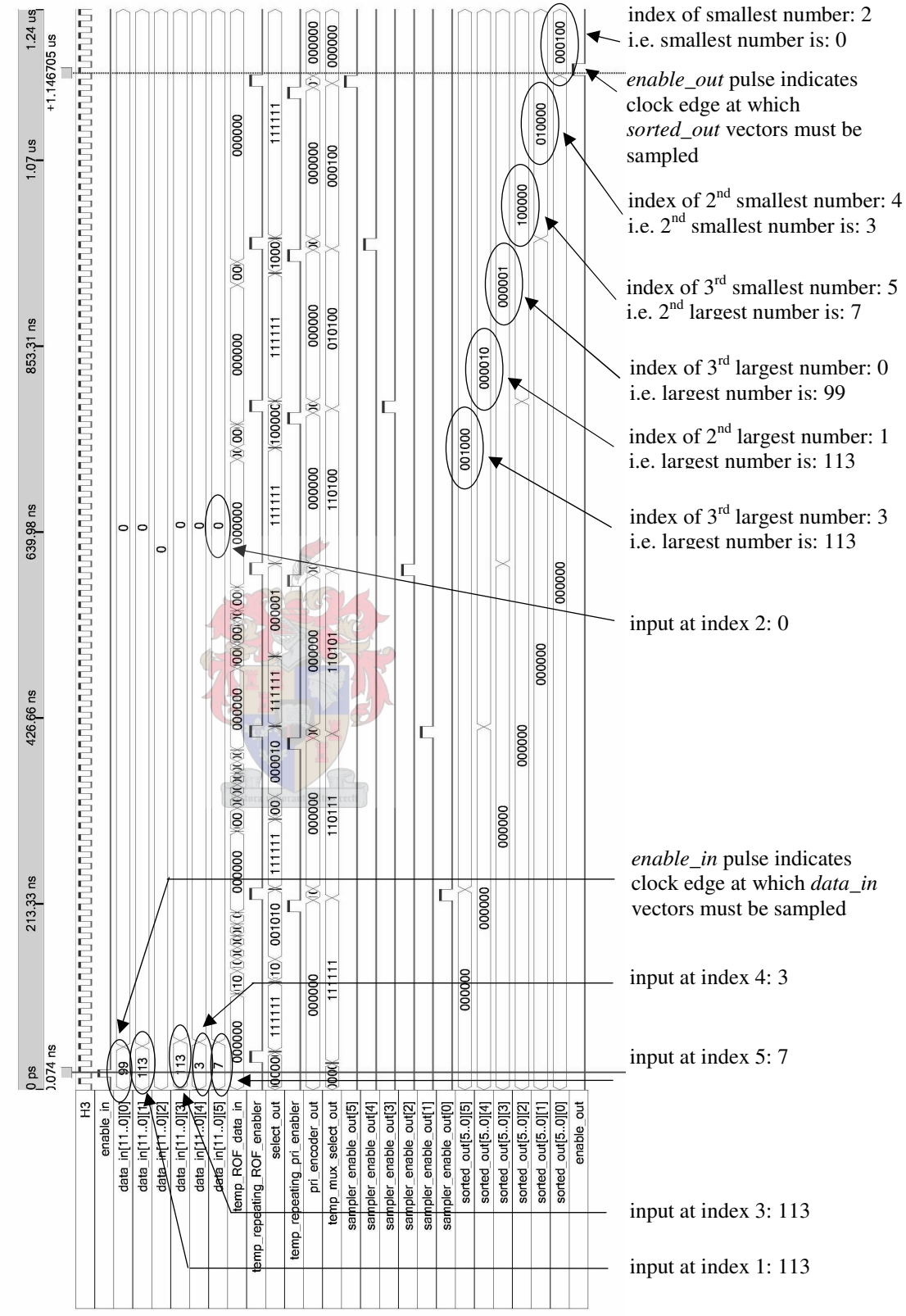


Figure B.19 *Sorter_ranked block output for six arbitrary inputs of 12 bits each. Note that the indices of the sorted outputs may be sampled 86 clock cycles (or 1,15 µs) after the inputs have been sampled.*

In Figure B.20 the determination of a maximum value by means of the *Sorter_greatest* block is illustrated. The output value (*sorted_out*) of '100' indicates that the maximum value is at index 2 of the input vector.

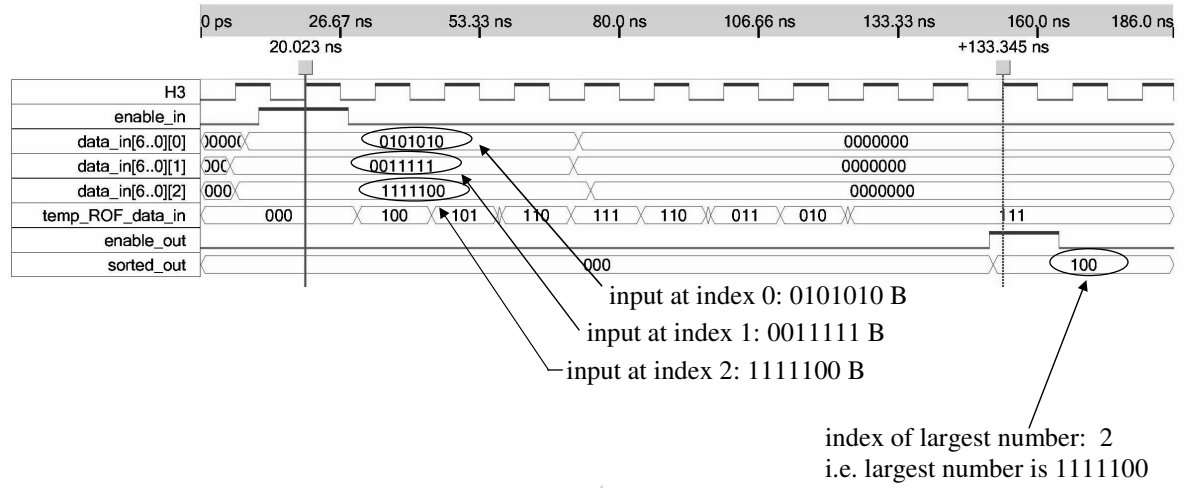


Figure B.20 *Sorter_greatest block output for three arbitrary inputs*

Table B.3 summarises the running times of the hardware sorting algorithms for 12-bit inputs (from an A/D converter, for instance). The 6-input sorting time (1,15 µs) result may be compared to the worst-case (6-input) optimised software-sorting timing-profiles of Table A.15.

n	Clock cycles	Running time
4	58	0,77 µs
5	72	0,96 µs
6	86	1,15 µs

Table B.3 *Running times of the hardware sorting algorithm for12-bit inputs*

It must be emphasised that the specific realisation of the FPGA-based sorter determines the largest number in a set of inputs sequentially, instead of the full sorting procedure of Hatirnaz and Leblebici in [63], for reasons explained in subsection B.3.

Appendix C – VHDL source code

DESIGN.VHD

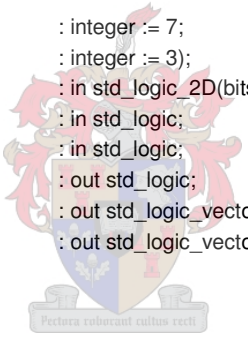
```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_arith.all;
4   use ieee.std_logic_unsigned.all;
5   library lpm;
6   use lpm.lpm_components.all;
7
8   entity Design is
9       generic (no_cells                   : integer := 4;
10                  ADC_width               : integer := 10;
11                  SA_databus_width        : integer := 10;
12                  SA_addressbus_width     : integer := 3);
13      port    (SA_addressbus              : in std_logic_vector(SA_addressbus_width-1 downto 0); -- stabilization
    algorithm address bus
14                  SA_databus              : in std_logic_vector(SA_databus_width-1 downto 0);
15                  SA_enable_in            : in std_logic;
16                  enable_in               : in std_logic;
17                  H3                      : in std_logic;
18                  output_level            : in integer range no_cells downto 0;
19                  prev_output_level       : in integer range no_cells downto 0;
20                  enable_out              : out std_logic;
21                  sw_fn_out               : out std_logic_vector(no_cells-1 downto 0);
22
23                  max_rat_index_out       : out std_logic_vector(2**no_cells-1 downto 0);
24                  state_ratings_out       : out std_logic_2D((2*(no_cells-1)-1) downto 0, 2**no_cells-1 downto
    0);
25                  cell_c_correct_out      : out std_logic_2D(2**no_cells-1 downto 0, no_cells-2 downto 0);
26                  cell_c_incorrect_out    : out std_logic_2D(2**no_cells-1 downto 0, no_cells-2 downto 0);
27                  state_ratings3_out      : out std_logic_2D((2*(no_cells-1)-1) downto 0, 2**no_cells-1 downto 0)

28              );
29  end entity Design;
30
31
32  architecture E of Design is
33      component Registers is
34          generic (no_cells               : integer := 5;
35                      ADC_width           : integer := 10;
36                      SA_databus_width    : integer := 10;
37                      SA_addressbus_width : integer := 3);
38          port    (SA_addressbus          : in std_logic_vector(SA_addressbus_width-1 downto 0); -- stabilization
    algorithm (local) address bus
39                      SA_databus          : in std_logic_vector(SA_databus_width-1 downto 0);
40                      enable_in           : in std_logic;
41
42                      cell_cap_req_in     : out std_logic_2D(1 downto 0, no_cells-2 downto 0);
43                      VM_C_VD_in          : out std_logic_2D(ADC_width-1 downto 0, no_cells-2 downto 0);

44                      critical_in         : out std_logic;
```

```
45                          prev_state_in       : out std_logic_vector(no_cells-1 downto 0));
46      end component Registers;
47
48      component state_sw_fn_gen is
49          generic (no_cells                   : integer := 3);
50          port (sw_fns_out                    : out std_logic_2D(2**no_cells-1 downto 0, no_cells-1 downto 0));
51      end component state_sw_fn_gen;
52
53      component cell_corresp_det_all is
54          generic (no_cells                   : integer := 4;
55                          ADC_width           : integer := 10);
56          port (cell_cap_req_in               : in std_logic_2D(1 downto 0, no_cells-2 downto 0);
57                  state_sw_fns_in             : in std_logic_2D(2**no_cells-1 downto 0, no_cells-1 downto 0);
58                  enable_in                   : in std_logic;
59                  VM_C_VD_in                  : in std_logic_2D(ADC_width-1 downto 0, no_cells-2 downto 0);

60                  H3                          : in std_logic;
61                  enable_out                  : out std_logic;
62                  cell_c_correct_out          : out std_logic_2D(2**no_cells-1 downto 0, no_cells-2 downto 0);
63                  cell_c_incorrect_out        : out std_logic_2D(2**no_cells-1 downto 0, no_cells-2 downto 0));
64      end component cell_corresp_det_all;
65
66      component Sorter_greatest is
67          generic (bits_per_num               : integer := 7;
68                          no_nums             : integer := 3);
69        port (data_in                         : in std_logic_2D(bits_per_num-1 downto 0, no_nums-1 downto 0);
70                  H3                          : in std_logic;
71                  enable_in                   : in std_logic;
72                  enable_out                  : out std_logic;
73                  temp_ROF_data_in            : out std_logic_vector(no_nums-1 downto 0);
74                  sorted_out                  : out std_logic_vector(no_nums-1 downto 0));
75      end component Sorter_greatest;
76
77      component Sorter_ranked is
78          generic (bits_per_num               : integer := 7;
79                          no_nums             : integer := 4);
80        port (data_in                         : in std_logic_2D(bits_per_num-1 downto 0, no_nums-1 downto 0);
81                  H3                          : in std_logic;
82                  enable_in                   : in std_logic;
83                  enable_out                  : out std_logic;
84                  sorted_out                  : out std_logic_2D(no_nums-1 downto 0, no_nums-1 downto 0);
85                                                                  --for dubugging purposes only:
86                  temp_repeating_pri_enabler: out std_logic;
87                  temp_repeating_ROF_enabler: out std_logic;
88                  temp_mux_select_out         : out std_logic_vector(no_nums-1 downto 0);
89                  temp_ROF_data_in            : out std_logic_vector(no_nums-1 downto 0);
90                  select_out                  : out std_logic_vector(no_nums-1 downto 0);
91                  pri_encoder_out             : out std_logic_vector(no_nums-1 downto 0);
92                  sampler_enable_out          : out std_logic_vector(no_nums-1 downto 0) );
93      end component Sorter_ranked;
94
95      component Delay_1 is
96          generic (delay_cycles               : integer := 3);
97        port (signal_in                       : in std_logic;
98                  H3                          : in std_logic;
99                  signal_out                  : out std_logic);
100     end component Delay_1;
101
```



```
102         component Rating_Det is
103                 generic (no_cells                   : integer := 4);
104                 port (enable_in                     : in std_logic;
105                         H3                          : in std_logic;
106                         enable_out                  : out std_logic;
107                         cell_c_correct_in           : in std_logic_2D(2**no_cells-1 downto 0, no_cells-2 downto 0);
108                         cell_c_incorrect_in         : in std_logic_2D(2**no_cells-1 downto 0, no_cells-2 downto 0);
109                         cap_ev_ranked_in            : in std_logic_2D(no_cells-1-1 downto 0, no_cells-1-1 downto 0);
110                         out_state_rating            : out std_logic_2D((2*(no_cells-1)-1) downto 0, 2**no_cells-1 downto
    0));
111         end component Rating_Det;
112
113         component Commutations_Verify is
114           generic (no_cells                         : integer := 3);
115           port (sw_fns_in                           : in std_logic_2D(2**no_cells-1 downto 0, no_cells-1 downto 0);
116                         prev_output_level           : in integer range no_cells downto 0;
117                         prev_state_in               : in std_logic_vector(no_cells-1 downto 0);
118                 output_level                        : in integer range no_cells downto 0;
119                         enable_in                   : in std_logic;
120                         critical_in                 : in std_logic;
121                         H3                          : in std_logic;
122                         ratings_in                  : in std_logic_2D((2*(no_cells-1)-1) downto 0, 2**no_cells-1 downto 0);
123                         enable_out                  : out std_logic;
124                         ratings_out                 : out std_logic_2D((2*(no_cells-1)-1) downto 0, 2**no_cells-1 downto
    0));
125         end component Commutations_Verify;
126
127         component Level_Verify is
128           generic (no_cells                         : integer := 3);
129           port (sw_fns_in                           : in std_logic_2D(2**no_cells-1 downto 0, no_cells-1 downto 0);
130                 output_level                        : in integer range no_cells downto 0;
131                         enable_in                   : in std_logic;
132                         H3                          : in std_logic;
133                         ratings_in                  : in std_logic_2D((2*(no_cells-1)-1) downto 0, 2**no_cells-1 downto 0);
134                         enable_out                  : out std_logic;
135                         ratings_out                 : out std_logic_2D((2*(no_cells-1)-1) downto 0, 2**no_cells-1 downto
    0));
136         end component Level_Verify;
137
138         component sw_fn_select is
139                 generic (no_cells                   : integer := 3);
140             port (sw_fns_in                         : in std_logic_2D(2**no_cells-1 downto 0, no_cells-1 downto 0);
141                         enable_in                   : in std_logic;
142                         index_in                    : in std_logic_vector(2**no_cells-1 downto 0);
143                         H3                          : in std_logic;
144                         enable_out                  : out std_logic;
145                         sw_fn_out                   : out std_logic_vector(no_cells-1 downto 0));
146         end component sw_fn_select;
147
148     signal cell_c_correct                           : std_logic_2D(2**no_cells-1 downto 0, no_cells-2 downto 0);
149     signal cell_c_incorrect                         : std_logic_2D(2**no_cells-1 downto 0, no_cells-2 downto 0);
150     signal cap_ev_ranked                            : std_logic_2D(no_cells-1-1 downto 0, no_cells-1-1 downto 0);
151     signal state_ratings                            : std_logic_2D((2*(no_cells-1)-1) downto 0, 2**no_cells-1 downto 0);
152     signal state_ratings2                           : std_logic_2D((2*(no_cells-1)-1) downto 0, 2**no_cells-1 downto 0);
153     signal state_ratings3                           : std_logic_2D((2*(no_cells-1)-1) downto 0, 2**no_cells-1 downto 0);
154     signal sw_fns                                   : std_logic_2D(2**no_cells-1 downto 0, no_cells-1 downto 0);
155     signal cap_ev_sorter_enable                     : std_logic;
156     signal rating_det_enable                        : std_logic;
```

```
157   signal max_rat_det_enable                : std_logic;
158   signal sw_fn_select_enable               : std_logic;
159   signal level_verify_enable               : std_logic;
160   signal comm_ver_enable                   : std_logic;
161   signal max_rat_index                     : std_logic_vector(2**no_cells-1 downto 0);
162
163   signal cell_cap_req_in                   : std_logic_2D(1 downto 0, no_cells-2 downto 0);
164   signal VM_C_VD_in                        : std_logic_2D(ADC_width-1 downto 0, no_cells-2 downto 0);
165   signal critical_in                       : std_logic;
166   signal prev_state_in                     : std_logic_vector(no_cells-1 downto 0);
167
168   begin
169
170       max_rat_index_out <= max_rat_index;
171       state_ratings_out <= state_ratings;
172       cell_c_correct_out <= cell_c_correct;
173       cell_c_incorrect_out <= cell_c_incorrect;
174       state_ratings3_out <= state_ratings3;
175
176   ---------------------- registers, to decrease IO pin use -----------------------------
177       r1a: Registers
178                   generic map             (no_cells => no_cells,
179                                            ADC_width => ADC_width,
180                                            SA_databus_width => SA_databus_width,
181                                            SA_addressbus_width => SA_addressbus_width)
182                   port map                (SA_addressbus => SA_addressbus,
183                                            SA_databus => SA_databus,
184                                            enable_in => SA_enable_in,
185
186                                            cell_cap_req_in => cell_cap_req_in,
187                                            VM_C_VD_in => VM_C_VD_in,
188                                            critical_in => critical_in,
189                                            prev_state_in => prev_state_in);
190
191   ----------------------sw fn generation-----------------------------------------------
192       s1 : state_sw_fn_gen
193                   generic map             (no_cells => no_cells)
194                   port map                (sw_fns_out => sw_fns);
195
196   ----------------------cell correspondence determination -----------------------------
197       ce1 : cell_corresp_det_all
198                           generic map     (no_cells => no_cells,
199                                            ADC_width => ADC_width)
200                           port map        (cell_cap_req_in => cell_cap_req_in,
201                                            state_sw_fns_in => sw_fns,
202                                            enable_in => enable_in,
203                                            VM_C_VD_in => VM_C_VD_in,
204                                            H3 => H3,
205                                            enable_out => cap_ev_sorter_enable,
206                                            cell_c_correct_out => cell_c_correct,
207                                            cell_c_incorrect_out => cell_c_incorrect);
208
209   ----------------------sorting of capacitor error voltages -------------------------
210
211       s2 : Sorter_ranked
212                           generic map     (bits_per_num => ADC_width,
213                                            no_nums => no_cells -1)
214                           port map        (data_in => VM_C_VD_in,
```

```
215                                     H3 => H3,
216                                     enable_in => cap_ev_sorter_enable,
217                                     enable_out => rating_det_enable,
218                                     sorted_out => cap_ev_ranked);
219
220     ----------rating determination ----------
221         r1:  Rating_Det
222                         generic map     (no_cells => no_cells)
223                         port map        (enable_in => rating_det_enable,
224                                         H3 => H3,
225                                         enable_out => level_verify_enable,
226                                         cell_c_correct_in => cell_c_correct,
227                                         cell_c_incorrect_in => cell_c_incorrect,
228                                         cap_ev_ranked_in => cap_ev_ranked,
229                                         out_state_rating => state_ratings);
230
231     ----------zeroing-out of states with incorrect output level----------
232         l1 : Level_Verify
233                         generic map     (no_cells => no_cells)
234                         port map        (sw_fns_in => sw_fns,
235                                         output_level => output_level,
236                                         enable_in => level_verify_enable,
237                                         H3 => H3,
238                                         ratings_in => state_ratings,
239                                         enable_out => comm_ver_enable,
240                                         ratings_out => state_ratings2);
241
242     ----------zeroing-out of states with non-minimum switch commutations---
243         c1 : Commutations_Verify
244                         generic map     (no_cells => no_cells)
245                         port map        (sw_fns_in => sw_fns,
246                                         prev_output_level => prev_output_level,
247                                         prev_state_in => prev_state_in,
248                                         output_level => output_level,
249                                         enable_in => comm_ver_enable,
250                                         critical_in => critical_in,
251                                         H3 => H3,
252                                         ratings_in => state_ratings2,
253                                         enable_out => max_rat_det_enable,
254                                         ratings_out => state_ratings3);
255
256     ----------finding of maximum state rating ----------
257         so1 : Sorter_greatest
258                         generic map     (bits_per_num => 2*(no_cells-1),
259                                         no_nums => 2**no_cells)
260                         port map        (data_in => state_ratings3,
261                                         H3 => H3,
262                                         enable_in => max_rat_det_enable,
263                                         enable_out => sw_fn_select_enable,
264                                         sorted_out => max_rat_index );
265
266     ----------selection of output switching functions ----------
267         sw1 : sw_fn_select
268                     generic map         (no_cells => no_cells)
269                     port map            (sw_fns_in => sw_fns,
270                                         enable_in => sw_fn_select_enable,
271                                         index_in => max_rat_index,
272                                         H3 => H3,
```

```
273                                     enable_out => enable_out,
274                                     sw_fn_out => sw_fn_out);
275     end architecture E;
```

CELL_CORRESP_DET_ALL.VHD

```
1       library ieee;
2       use ieee.std_logic_1164.all;
3       use ieee.std_logic_arith.all;
4       use ieee.std_logic_unsigned.all;
5       library lpm;
6       use lpm.lpm_components.all;
7
8       entity cell_corresp_det_all is
9           generic (no_cells                   : integer := 4;
10                      ADC_width               : integer := 10);
11          port (cell_cap_req_in               : in std_logic_2D(1 downto 0, no_cells-2 downto 0);
12                  state_sw_fns_in             : in std_logic_2D(2**no_cells-1 downto 0, no_cells-1 downto 0);
13                  enable_in                   : in std_logic;
14                  VM_C_VD_in                  : in std_logic_2D(ADC_width-1 downto 0, no_cells-2 downto 0);

15                  H3                          : in std_logic;
16                  enable_out                  : out std_logic;
17                  cell_c_correct_out          : out std_logic_2D(2**no_cells-1 downto 0, no_cells-2 downto 0);
18                  cell_c_incorrect_out        : out std_logic_2D(2**no_cells-1 downto 0, no_cells-2 downto 0));
19      end entity cell_corresp_det_all;
20
21
22      architecture A of cell_corresp_det_all is
23
24      component cell_corresp_det_single is
25          port (cell_cap_req                  : in std_logic_vector(1 downto 0);
26                  state_sw_fn_in              : in std_logic_vector(1 downto 0);
27                  H3                          : in std_logic;
28                  enable_in                   : in std_logic;
29                  cell_c_correct_out          : out std_logic;
30              cell_c_incorrect_out            : out std_logic);
31      end component cell_corresp_det_single;
32
33      component Delay_1 is
34          generic (delay_cycles               : integer := 3);
35        port (signal_in                       : in std_logic;
36                  H3                          : in std_logic;
37                  signal_out                  : out std_logic);
38      end component Delay_1;
39
40      begin
41          gen0:
42          for j in 2**no_cells-1 downto 0 generate            -- number of states
43              gen2 :
44                  for i in (no_cells-2) downto 0 generate
45                          c1 : cell_corresp_det_single
46                                  port map        (cell_cap_req(1) => cell_cap_req_in(1,i),
47                                                   cell_cap_req(0) => cell_cap_req_in(0,i),
48                                                   state_sw_fn_in(1) => state_sw_fns_in(j,i+1),       --mapping NB
49                                                   state_sw_fn_in(0) => state_sw_fns_in(j,i),         --mapping NB
50                                                   H3 => H3,
51                                                   enable_in => enable_in,
```

```
52                                          cell_c_correct_out => cell_c_correct_out(j,i),
53                                          cell_c_incorrect_out => cell_c_incorrect_out(j,i) );
54              end generate;
55          end generate;
56
57          e2 : Delay_1
58                              generic map     (delay_cycles => 1)
59                              port map        (signal_in => enable_in,
60                                               H3 => H3,
61                                               signal_out => enable_out);
62
63      end architecture A;
```

CELL_CORRESP_DET_SINGLE.VHD

```
1       library ieee;
2       use ieee.std_logic_1164.all;
3
4       entity Cell_Corresp_Det_Single is
5                   port   (cell_cap_req                    : in std_logic_vector(1 downto 0);
6                           state_sw_fn_in                  : in std_logic_vector(1 downto 0);
7                           H3                              : in std_logic;
8                           enable_in                       : in std_logic;
9                           cell_c_correct_out              : out std_logic;
10                          cell_c_incorrect_out            : out std_logic);
11      end entity Cell_Corresp_Det_Single;
12
13      architecture D of Cell_Corresp_Det_Single is
14          type state_type is (state_0, state_1);
15          signal state: state_type;
16      begin
17          process (H3) is
18          begin
19                  if rising_edge(H3) then
20                          if1:
21                          if enable_in = '1' then
22                                  if2: if ((state_sw_fn_in = "00") and (cell_cap_req = "00"))
23                                    or ((state_sw_fn_in = "00") and (cell_cap_req = "11"))
24                                    or ((state_sw_fn_in = "01") and (cell_cap_req = "00"))
25                                    or ((state_sw_fn_in = "01") and (cell_cap_req = "01"))
26                                    or ((state_sw_fn_in = "01") and (cell_cap_req = "11"))
27                                    or ((state_sw_fn_in = "10") and (cell_cap_req = "00"))
28                                    or ((state_sw_fn_in = "10") and (cell_cap_req = "10"))
29                                    or ((state_sw_fn_in = "10") and (cell_cap_req = "11"))
30                                    or ((state_sw_fn_in = "11") and (cell_cap_req = "00"))
31                                    or ((state_sw_fn_in = "11") and (cell_cap_req = "11")) then
32                                          cell_c_correct_out <= '1';                  -- cell correspondence is "correct"
33                                      cell_c_incorrect_out <= '0';
34
35                                  elsif ((state_sw_fn_in = "01") and (cell_cap_req = "10"))
36                                    or ((state_sw_fn_in = "10") and (cell_cap_req = "01")) then
37                                      cell_c_correct_out <= '0';                  -- cell correspondence is "incorrect"
38                                      cell_c_incorrect_out <= '1';
39
40                                  elsif ((state_sw_fn_in = "00") and (cell_cap_req = "01"))
41                                    or ((state_sw_fn_in = "00") and (cell_cap_req = "10"))
42                                    or ((state_sw_fn_in = "11") and (cell_cap_req = "01"))
```

```
43                              or ((state_sw_fn_in = "11") and (cell_cap_req = "10")) then
44                                  cell_c_correct_out <= '0';                  -- cell correspondence is "no_action"
45                                  cell_c_incorrect_out <= '0';
46                              end if if2;
47                          end if if1;
48                  end if;
49              end process;
50      end architecture D;
```

CUSTOM_MUX.VHD

```
1       library ieee;
2       use ieee.std_logic_1164.all;
3       use ieee.std_logic_arith.all;
4       use ieee.std_logic_unsigned.all;
5       library lpm;
6       use lpm.lpm_components.all;
7
8       -- Custom multiplexer, for use in selection of ordered output generated by a ROF core...
9       -- 2 cases: multiplexer for final sw.fn. selection
10      --                  and multiplexer for cell correspondence placement within the state rating...
11
12      -- the multiplexer is custom in the sense that the selection input is (priority) encoded
13      -- conventionally, a number is used as a selector...
14
15      entity Custom_MUX is
16              generic     (bits_per_channel       : integer := 10;
17                          no_channels_in          : integer := 6);
18              port        (channels_in            : in std_logic_2D(no_channels_in-1 downto 0, bits_per_channel-1
        downto 0);
19                          H3                      : in std_logic;
20                          enable_in               : in std_logic;
21                          enable_out              : out std_logic;
22                          sel_in                  : in std_logic_vector(no_channels_in-1 downto 0);
23                          channel_out             : out std_logic_vector(bits_per_channel-1 downto 0));
24      end entity Custom_MUX;
25
26      architecture M of Custom_MUX is
27
28      component lpm_or
29              generic     (LPM_WIDTH              : POSITIVE;
30                          LPM_SIZE                : POSITIVE;
31                          LPM_TYPE                : STRING := "LPM_OR";
32                          LPM_HINT                : STRING := "UNUSED");
33              port        (data                   : in STD_LOGIC_2D(LPM_SIZE-1 downto 0, LPM_WIDTH-1
        downto 0);
34              result                              : out STD_LOGIC_VECTOR(LPM_WIDTH-1 downto 0));
35      end component;
36
37      signal temp                                 : std_logic_2D(no_channels_in-1 downto 0, bits_per_channel-1
        downto 0);
38      signal temp_channel_out                     : std_logic_vector(bits_per_channel-1 downto 0);
39
40      begin
41        gen0:
42        for k in (no_channels_in-1) downto 0 generate
43          gen1:
```

```
44          for p in (bits_per_channel-1) downto 0 generate
45            temp(k,p) <= channels_in(k,p) and sel_in(k);
46          end generate;
47        end generate;
48
49        OR1: lpm_or generic map (lpm_width => bits_per_channel,      --bitwise OR
50                                              lpm_size => no_channels_in)
51                            port map (data => temp,
52                                              result => temp_channel_out);
53        process (H3)
54        begin
55          if rising_edge(H3) then
56            if enable_in = '1' then
57                  channel_out <= temp_channel_out;
58                  enable_out <= '1';
59            else
60               channel_out <= conv_std_logic_vector(0,bits_per_channel);
61                  enable_out <= '0';
62            end if;
63          end if;
64        end process;
65
66      end architecture M;
```

COMMUTATIONS_VERIFY.VHD

```
1       library ieee;
2       use ieee.std_logic_1164.all;
3       use ieee.std_logic_arith.all;
4       use ieee.std_logic_unsigned.all;
5       library lpm;
6       use lpm.lpm_components.all;
7
8
9       entity Commutations_Verify is
10                generic     (no_cells                 : integer := 3);
11                port        (sw_fns_in                : in std_logic_2D(2**no_cells-1 downto 0, no_cells-1 downto 0);
12                             prev_output_level        : in integer range no_cells downto 0;
13                             prev_state_in            : in std_logic_vector(no_cells-1 downto 0);
14                             output_level             : in integer range no_cells downto 0;
15                             enable_in                : in std_logic;
16                             critical_in              : in std_logic;
17                             H3                       : in std_logic;
18                             ratings_in               : in std_logic_2D((2*(no_cells-1)-1) downto 0, 2**no_cells-1
        downto 0);
19                             enable_out               : out std_logic;
20                             ratings_out              : out std_logic_2D((2*(no_cells-1)-1) downto 0, 2**no_cells-1
        downto 0));
21      end entity Commutations_Verify;
22
23
24      architecture U of Commutations_Verify is
25      begin
26        -- we are now going to zero-out all ratings with non_minimum number of commutations...
27        process (H3)
28          variable c_sum                          : integer range no_cells downto 0;
29          variable commutations_allowed           : integer range no_cells downto 0;
```

```
30      begin
31
32        if4:
33          if rising_edge(H3) then
34                  if5:
35                  if (enable_in = '1') then
36                          enable_out <= '1';
37                          commutations_allowed := abs(prev_output_level - output_level);
38
39                          loop_all_ratings:
40                          for i in (2**no_cells-1) downto 0 loop
41
42                                  c_sum := 0;
43                                  loop_every_bit:
44                                  for k in (no_cells-1) downto 0 loop
45                                          if1:
46                                          if (sw_fns_in(i,k) /= prev_state_in(k)) then
47                                            c_sum := c_sum + 1;
48
49                                          end if if1;
50                                  end loop loop_every_bit;
51
52
53                              loop4:
54                                  for j in (2*(no_cells-1)-1) downto 0 loop
55                                          if (c_sum /= commutations_allowed) and (critical_in = '0') then
56                                                  ratings_out(j,i) <= '0';
57                                          else
58                                                  ratings_out(j,i) <= ratings_in(j,i);
59                                          end if;
60                                  end loop loop4;
61
62                          end loop loop_all_ratings;
63                  else
64                          enable_out <= '0';
65                  end if if5;
66              end if if4;
67        end process;
68      end architecture U;
```

DELAY_1.VHD

```
1       library ieee;
2       use ieee.std_logic_1164.all;
3       use ieee.std_logic_arith.all;
4       use ieee.std_logic_unsigned.all;
5       library lpm;
6       use lpm.lpm_components.all;
7
8       entity Delay_1 is
9                   generic     (delay_cycles               : integer := 3);
10                  port        (signal_in                  : in std_logic;
11                               H3                         : in std_logic;
12                               signal_out                 : out std_logic);
13      end entity Delay_1;
14
15      architecture G of Delay_1 is
```

```
16      component DFF
17          port   (d    : in STD_LOGIC;
18                  clk  : in STD_LOGIC;
19                  clrn : in STD_LOGIC;
20                  prn  : in STD_LOGIC;
21                  q    : out STD_LOGIC );
22      end component;
23
24      signal temp : std_logic_vector(delay_cycles downto 0);
25
26      begin
27          temp(delay_cycles) <= signal_in;
28
29          gen1:
30          for i in (delay_cycles-1) downto 0 generate
31                  d1: DFF
32                          port map(d => temp(i+1),
33                                          clk => H3,
34                                          clrn => '1',
35                                          prn => '1',
36                                          q => temp(i) );
37          end generate;
38
39          signal_out <= temp(0);
40      end architecture G;
```


ENABLER_REPEAT.VHD

```
1       library ieee;
2       use ieee.std_logic_1164.all;
3       use ieee.std_logic_arith.all;
4       use ieee.std_logic_unsigned.all;
5       library lpm;
6       use lpm.lpm_components.all;
7
8       entity EnablerRepeat is
9                   generic     (delay_cycles               : integer := 3;
10                              repeats                     : integer := 4);
11                  port        (enable_in                  : in std_logic;
12                              H3                          : in std_logic;
13                              enable_out                  : out std_logic);
14      end entity EnablerRepeat;
15
16      architecture G of EnablerRepeat is
17      begin
18        PROCESS (H3)
19                  variable    delay_cnt                   : INTEGER RANGE 0 TO delay_cycles-1;
20                  variable    repeat_cnt                  : INTEGER range 0 to repeats;
21                  variable    enabler_active              : std_logic := '0';
22                  variable    temp_signal_out             : std_logic;
23        BEGIN
24          IF rising_edge(H3) THEN
25                   IF enable_in = '1' THEN
26                  delay_cnt := delay_cycles-1;
27                          repeat_cnt := repeats;
28                   end if;
29
```

```
30                      if (delay_cnt = 0) then
31                              if (repeat_cnt = 0) then
32                                              temp_signal_out := '0';
33                                      else
34                                              repeat_cnt := repeat_cnt -1;
35                                              delay_cnt := delay_cycles-1;
36                                              temp_signal_out := '1';
37                                      end if;
38
39                      else
40                              temp_signal_out := '0';
41                        delay_cnt := delay_cnt -1;
42                      end if;
43          END IF;
44          enable_out <= temp_signal_out;
45        END PROCESS;
46      end architecture G;
```

LATCH_WITH_SSET.VHD

```
1       library ieee;
2       use ieee.std_logic_1164.all;
3       use ieee.std_logic_arith.all;
4       use ieee.std_logic_unsigned.all;
5       library lpm;
6       use lpm.lpm_components.all;
7
8       entity Latch_with_sset is
9               port            (enable_in              : in std_logic;
10                              sset_in                 : in std_logic;
11                              H3                      : in std_logic;
12                              q                       : out std_logic);
13      end entity Latch_with_sset;
14
15      architecture G of Latch_with_sset is
16              signal          temp_q                  : std_logic;
17      begin
18          PROCESS (H3, enable_in)
19          BEGIN
20                  IF rising_edge(H3) THEN
21                          IF (sset_in = '1' )THEN
22                                  temp_q <= '1';
23                          END IF;
24                  END IF;
25            IF (enable_in = '1') THEN
26               temp_q <= '0';
27            END IF;
28          END PROCESS;
29          q <= temp_q;
30      end architecture G;
```

LEVEL_VERIFY.VHD

```
1       library ieee;
2       use ieee.std_logic_1164.all;
```

```
3       use ieee.std_logic_arith.all;
4       use ieee.std_logic_unsigned.all;
5       library lpm;
6       use lpm.lpm_components.all;
7
8
9       entity Level_Verify is
10          generic     (no_cells                   : integer := 3);
11          port        (sw_fns_in                  : in std_logic_2D(2**no_cells-1 downto 0, no_cells-1 downto 0);
12                       output_level               : in integer range no_cells downto 0;
13                       enable_in                  : in std_logic;
14                       H3                         : in std_logic;
15                       ratings_in                 : in std_logic_2D((2*(no_cells-1)-1) downto 0, 2**no_cells-1
        downto 0);
16                       enable_out                 : out std_logic;
17                       ratings_out                : out std_logic_2D((2*(no_cells-1)-1) downto 0, 2**no_cells-1
        downto 0));
18      end entity Level_Verify;
19
20
21      architecture U of Level_Verify is
22      begin
23       -- we are now going to zero-out all ratings with an incorrect output level...
24       process (H3)
25          variable sum : integer range no_cells downto 0;
26       begin
27
28         if4:
29          if rising_edge(H3) then
30                  if5:
31                  if (enable_in = '1') then
32                          enable_out <= '1';
33                          loop_all_ratings:
34                          for i in (2**no_cells-1) downto 0 loop
35
36                                  sum := 0;
37                                  loop_every_bit:
38                                  for k in (no_cells-1) downto 0 loop
39                                          if1:
40                                          if (sw_fns_in(i,k) = '1') then
41                                            sum := sum + 1;
42
43                                          end if if1;
44                                  end loop loop_every_bit;
45
46                              loop4:
47                                  for j in (2*(no_cells-1)-1) downto 0 loop
48                                          if (sum /= output_level) then
49                                                  ratings_out(j,i) <= '0';
50                                          else
51                                                  ratings_out(j,i) <= ratings_in(j,i);
52                                          end if;
53                                  end loop loop4;
54
55                          end loop loop_all_ratings;
56                  else
57                          enable_out <= '0';
58                  end if if5;
```

```
59          end if if4;
60
61
62        end process;
63      end architecture U;
```

MAJORITY_GREATEST.VHD

```
1       library ieee;
2       use ieee.std_logic_1164.all;
3       use ieee.std_logic_arith.all;
4       use ieee.std_logic_unsigned.all;
5       library lpm;
6       use lpm.lpm_components.all;
7
8
9       entity majority_greatest is
10          generic     (no_inputs                  : integer := 8);
11          port        (bits_in                    : in std_logic_vector(no_inputs-1 downto 0);
12                       H3                         : in std_logic;
13                       majority_out               : out std_logic);
14      end entity majority_greatest;
15
16
17      architecture E of majority_greatest is
18
19      component lpm_or
20          generic     (LPM_WIDTH                  : POSITIVE;
21                       LPM_SIZE                   : POSITIVE;
22                       LPM_TYPE                   : STRING := "LPM_OR";
23                       LPM_HINT                   : STRING := "UNUSED");
24          port        (data                       : in STD_LOGIC_2D(LPM_SIZE-1 downto 0, LPM_WIDTH-1
        downto 0);
25                       result                     : out STD_LOGIC_VECTOR(LPM_WIDTH-1 downto 0));
26      end component;
27
28      component DFF
29          port        (d                          : in STD_LOGIC;
30                       clk                        : in STD_LOGIC;
31                       clrn                       : in STD_LOGIC;
32                       prn                        : in STD_LOGIC;
33                       q                          : out STD_LOGIC );
34      end component;
35
36      signal
37          temp                                    : std_logic_2D(no_inputs-1 downto 0, 0 downto 0);
38      begin
39          gen1:
40          for i in no_inputs-1 downto 0 generate
41                  temp(i,0) <= bits_in(i);
42          end generate;
43
44          maks_OR: lpm_or
45                  generic map     (lpm_width => 1,
46                                   lpm_size => no_inputs)
47                  port map        (data => temp,
48                                   result(0) => majority_out);
```

```
49      end architecture E;
```

PRIORITY_ENCODER.VHD

```
1       library ieee;
2       use ieee.std_logic_1164.all;
3       use ieee.std_logic_arith.all;
4       use ieee.std_logic_unsigned.all;
5       library lpm;
6       use lpm.lpm_components.all;
7
8
9       entity Priority_encoder is
10          generic     (no_bits_in                 : integer := 6);
11          port        (encoder_in                 : in std_logic_vector(no_bits_in-1 downto 0);
12                       H3                         : in std_logic;
13                       enable_in                  : in std_logic;
14                       encoder_out                : out std_logic_vector(no_bits_in-1 downto 0));
15      end entity Priority_encoder;
16
17      architecture P of Priority_encoder is
18      begin
19       process (H3)
20          variable    temp1                       : std_logic;
21          variable    temp2                       : std_logic;
22
23       begin
24          if rising_edge(H3) then
25           if enable_in = '1' then
26                  loop_outer:
27                  for index in (no_bits_in-1) downto 0 loop
28                              temp1 := encoder_in(index);
29                              temp2 := '1';
30
31                              if (no_bits_in-1) >= (index+1) then
32                                      loop_inner:
33                                              for count in (no_bits_in-1) downto (index+1) loop
34                                                temp2 := temp2 and not(encoder_in(count));
35                                              end loop loop_inner;
36                              end if;
37
38                              encoder_out(index) <= temp1 and temp2;
39                  end loop loop_outer;
40
41           else
42                   encoder_out <= conv_std_logic_vector(0, no_bits_in);
43           end if;
44          end if;
45       end process;
46
47      end architecture P;
```

RATING_DET.VHD

```
1       library ieee;
```

```
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_arith.all;
4   use ieee.std_logic_unsigned.all;
5   library lpm;
6   use lpm.lpm_components.all;
7
8   entity Rating_Det is
9       generic     (no_cells                   : integer := 4);
10      port        (enable_in                  : in std_logic;
11                   H3                         : in std_logic;
12                   enable_out                 : out std_logic;
13                   cell_c_correct_in          : in std_logic_2D(2**no_cells-1 downto 0, no_cells-2 downto 0);
14                   cell_c_incorrect_in        : in std_logic_2D(2**no_cells-1 downto 0, no_cells-2 downto 0);
15                   cap_ev_ranked_in           : in std_logic_2D(no_cells-1-1 downto 0, no_cells-1-1 downto 0);
16                   out_state_rating           : out std_logic_2D((2*(no_cells-1)-1) downto 0, 2**no_cells-1
    downto 0));
17  end entity Rating_Det;
18
19  architecture E of Rating_Det is
20      component Custom_MUX is
21          generic     (bits_per_channel       : integer := 10;
22                       no_channels_in         : integer := 6);
23          port        (channels_in            : in std_logic_2D(no_channels_in-1 downto 0, bits_per_channel-1
    downto 0);
24                       H3                     : in std_logic;
25                       enable_in              : in std_logic;
26                       enable_out             : out std_logic;
27                       sel_in                 : in std_logic_vector(no_channels_in-1 downto 0);
28                       channel_out            : out std_logic_vector(bits_per_channel-1 downto 0));
29      end component Custom_MUX;
30
31      component Delay_1 is
32          generic     (delay_cycles           : integer := 3);
33          port        (signal_in              : in std_logic;
34                       H3                     : in std_logic;
35                       signal_out             : out std_logic);
36      end component Delay_1;
37
38  type cell_corresp is array (2**no_cells -1 downto 0) of std_logic_2D(no_cells-2 downto 0, 0 downto 0);
39  type sel_type is array (no_cells-2 downto 0) of std_logic_vector(no_cells-2 downto 0);
40
41
42  signal sel_sorted_out                       : sel_type;
43  signal temp_cell_c_correct                  : cell_corresp;
44  signal temp_cell_c_not_incorrect            : cell_corresp;
45  signal lower_half_rating_out                : std_logic_2D(2**no_cells -1 downto 0, no_cells-2 downto 0);
46  signal upper_half_rating_out                : std_logic_2D(2**no_cells -1 downto 0, no_cells-2 downto 0);
47
48  begin
49      gen0:
50      for j in 2**no_cells-1 downto 0 generate            -- number of states
51          genA2:
52          for m in (no_cells-2) downto 0 generate
53              temp_cell_c_correct(j)(m,0) <= cell_c_correct_in(j,m);
54              temp_cell_c_not_incorrect(j)(m,0) <= not cell_c_incorrect_in(j,m);
55          end generate;
56
57          gen23:
```

```
58              for m in (no_cells-2) downto 0 generate
59              correct1 : Custom_MUX
60                      generic map     (bits_per_channel => 1,
61                                       no_channels_in => no_cells-1)
62                      port map        (channels_in => temp_cell_c_correct(j),
63                                       H3 => H3,
64                                       enable_in => enable_in,
65                                       sel_in => sel_sorted_out(m),
66                                       channel_out(0) => lower_half_rating_out(j,m));
67
68              incorrect1 : Custom_MUX
69                      generic map     (bits_per_channel => 1,
70                                       no_channels_in => no_cells-1)
71                      port map        (channels_in => temp_cell_c_not_incorrect(j),
72                                       H3 => H3,
73                                       enable_in => enable_in,
74                                       sel_in => sel_sorted_out(m),
75                                       channel_out(0) => upper_half_rating_out(j,m));
76              end generate;
77
78              gen5:
79              for p in (no_cells-2) downto 0 generate
80                      out_state_rating(p,j) <= lower_half_rating_out(j,p);
81                      out_state_rating(p+no_cells-1,j) <= upper_half_rating_out(j,p);
82              end generate;
83      end generate;
84
85      genB1:
86      for k in no_cells-2 downto 0 generate
87              genB2:
88              for j in no_cells-2 downto 0 generate
89                      sel_sorted_out(k)(j) <= cap_ev_ranked_in(j,k);
90              end generate;
91      end generate;
92
93      e2 : Delay_1
94              generic map     (delay_cycles => 1)
95              port map        (signal_in => enable_in,
96                               H3 => H3,
97                               signal_out => enable_out);
98  end architecture E;
```

ROF_BIT_PLANE.VHD

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_arith.all;
4   use ieee.std_logic_unsigned.all;
5   library lpm;
6   use lpm.lpm_components.all;
7
8   entity ROF_Bit_plane is
9       generic     (no_bits                    : integer := 7);
10      port        (data_in                    : in std_logic_vector(no_bits-1 downto 0);
11                   H3                         : in std_logic;
12                   sset_in                    : in std_logic;
13                   select_out                 : out std_logic_vector(no_bits-1 downto 0));
```

```
14      end entity ROF_Bit_plane;
15
16      architecture G of ROF_Bit_plane is
17      component ROF_cell_MOD is
18          port    (serial_data_in                 : in std_logic;
19                   H3                             : in std_logic;
20                   majority_current_level_in      : in std_logic;
21                   sset_in                        : in std_logic;
22                   majority_current_level_out     : out std_logic;
23                   select_out                     : out std_logic);
24      end component ROF_cell_MOD;
25
26      component majority_greatest is
27          generic     (no_inputs                  : integer := 8);
28          port        (bits_in                    : in std_logic_vector(no_inputs-1 downto 0);
29                       H3                         : in std_logic;
30                       majority_out               : out std_logic);
31      end component majority_greatest;
32
33      signal to_maj                               : std_logic_vector(no_bits-1 downto 0);
34      signal from_maj                             : std_logic;
35      signal temp_data                            : std_logic_vector(no_bits-1 downto 0);
36      signal temp_select                          : std_logic_vector(no_bits-1 downto 0);
37
38      begin
39       gen1:
40       for i in no_bits-1 downto 0 generate
41          ROF_cell_comp:
42          ROF_cell_MOD port map
43                   (serial_data_in => data_in(i),
44                    H3 => H3,
45                    select_out => temp_select(i),
46
47                    majority_current_level_in => from_maj,
48                    sset_in => sset_in,
49                    majority_current_level_out => to_maj(i)  );
50       end generate;
51
52       Maj_Decision:
53       majority_greatest
54       generic map
55                   (no_inputs => no_bits)
56       port map
57                   (bits_in => to_maj,
58                    H3 => H3,
59                    majority_out => from_maj);
60
61       select_out <= temp_select;
62      end architecture G;
```

ROF_CELL_MOD.VHD

```
1       -- select line: active low...
2       library ieee;
3       use ieee.std_logic_1164.all;
4       use ieee.std_logic_arith.all;
```

```
5   use ieee.std_logic_unsigned.all;
6   library lpm;
7   use lpm.lpm_components.all;
8
9   entity ROF_cell_MOD is
10      port    (serial_data_in                 : in std_logic;
11               H3                             : in std_logic;
12               majority_current_level_in      : in std_logic;
13               sset_in                        : in std_logic;
14               majority_current_level_out     : out std_logic;
15               select_out                     : out std_logic);
16  end entity ROF_cell_MOD;
17
18  architecture E of ROF_cell_MOD is
19  component DFF
20      port   (d                               : in STD_LOGIC;
21              clk                             : in STD_LOGIC;
22              clrn                            : in STD_LOGIC;
23              prn                             : in STD_LOGIC;
24              q                               : out STD_LOGIC );
25  end component;
26
27  COMPONENT lpm_ff
28      GENERIC  (LPM_WIDTH                     : POSITIVE;
29                LPM_AVALUE                    : STRING := "UNUSED";
30                LPM_SVALUE                    : STRING := "UNUSED";
31                LPM_PVALUE                    : STRING := "UNUSED";
32                LPM_FFTYPE                    : STRING := "DFF";
33                LPM_TYPE                      : STRING := "LPM_FF";
34                LPM_HINT                      : STRING := "UNUSED");
35      PORT      (data                         : IN STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
36                 clock                        : IN STD_LOGIC;
37                 enable                       : IN STD_LOGIC := '1';
38                 sload, sclr, sset, aload, aclr, aset : IN STD_LOGIC := '0';
39                 q                            : OUT STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0));
40  END COMPONENT;
41
42  component mux
43      generic    (WIDTH                       : POSITIVE;
44                  WIDTHS                      : POSITIVE);
45      port       (data                        : in STD_LOGIC_VECTOR(WIDTH-1 downto 0);
46                  sel                         : in STD_LOGIC_VECTOR(WIDTHS-1 downto 0);
47                  result                      : out STD_LOGIC);
48  end component;
49
50  signal bit_prop_or_shifted                  : std_logic;
51  signal propagate_instruction                : std_logic;
52  signal XOR_out                              : std_logic;
53  signal NOT_out                              : std_logic;
54  signal temp_data                            : std_logic;
55  signal temp_select                          : std_logic;
56
57  begin
58     Prop_or_shifted_data_out:
59       dff port map
60                  (d    => bit_prop_or_shifted,
61                   clk  => H3,
62                   clrn => '1',
```

```
63                      prn  => '1',
64                      q    => temp_data);
65
66      Propagate_instruction_out:
67         lpm_ff generic map
68                      (lpm_width => 1)
69                      port map
70                      (data    => conv_std_logic_vector(propagate_instruction,1),
71                       clock  => H3,
72                       sset  => sset_in,
73                       q(0)    => temp_select); -- omdat vektorlengte = 1 is...
74
75      select_out <= temp_select;
76
77      Mux_for_propagation:
78         mux generic map
79                      (width => 2, widths => 1)
80            port map
81            -- when sel = '0', the propagated data (or prev_level_data_in) should pass to 'result'...
82                      (data(0) => temp_data,
83                       data(1) => serial_data_in,
84                       sel(0) => temp_select,
85                       result => bit_prop_or_shifted);
86
87      XOR_out <= bit_prop_or_shifted XOR majority_current_level_in;
88      NOT_out <= NOT XOR_out;
89
90      propagate_instruction <= NOT_out AND temp_select;
91
92      majority_current_level_out <= bit_prop_or_shifted;
93   end architecture E;
```

SAMPLE_AND_HOLD.VHD

```
1    library ieee;
2    use ieee.std_logic_1164.all;
3    use ieee.std_logic_arith.all;
4    use ieee.std_logic_unsigned.all;
5    library lpm;
6    use lpm.lpm_components.all;
7
8    entity SampleAndHold is
9        generic     (width                  : integer := 5);
10       port        (signal_in              : in std_logic_vector(width-1 downto 0);
11                    enable_in              : in std_logic;
12                    H3                     : in std_logic;
13                    signal_out             : out std_logic_vector(width-1 downto 0) );
14   end entity SampleAndHold;
15
16   architecture G of SampleAndHold is
17   begin
18     PROCESS (H3)
19     BEGIN
20       IF rising_edge(H3) THEN
21             IF enable_in = '1' THEN
22                   signal_out <= signal_in;
23             END IF;
```

```
24          END IF;
25        END PROCESS;
26      end architecture G;
```

SORTER_GREATEST.VHD

```
1       -- select line: active low...
2       library ieee;
3       use ieee.std_logic_1164.all;
4       use ieee.std_logic_arith.all;
5       use ieee.std_logic_unsigned.all;
6       library lpm;
7       use lpm.lpm_components.all;
8
9       entity Sorter_greatest is
10          generic     (bits_per_num           : integer := 7;
11                       no_nums                : integer := 3);
12          port        (data_in                : in std_logic_2D(bits_per_num-1 downto 0, no_nums-1 downto
        0);
13                       H3                     : in std_logic;
14                       enable_in              : in std_logic;
15                       enable_out             : out std_logic;
16                       temp_ROF_data_in       : out std_logic_vector(no_nums-1 downto 0);
17                       sorted_out             : out std_logic_vector(no_nums-1 downto 0));
18      end entity Sorter_greatest;
19
20      architecture A of Sorter_greatest is
21          component ROF_Bit_plane is
22          generic     (no_bits                : integer := 7);
23          port        (data_in                : in std_logic_vector(no_bits-1 downto 0);
24                       H3                     : in std_logic;
25                       sset_in                : in std_logic;
26                       select_out             : out std_logic_vector(no_bits-1 downto 0));
27          end component ROF_Bit_plane;
28
29          component Delay_1 is
30                generic     (delay_cycles     : integer := 3);
31                port        (signal_in        : in std_logic;
32                             H3               : in std_logic;
33                             signal_out       : out std_logic);
34          end component Delay_1;
35
36          component SampleAndHold is
37                generic     (width            : integer := 5);
38                port        (signal_in        : in std_logic_vector(width-1 downto 0);
39                             enable_in        : in std_logic;
40                             H3               : in std_logic;
41                             signal_out       : out std_logic_vector(width-1 downto 0) );
42          end component SampleAndHold;
43
44          component Priority_encoder is
45                generic     (no_bits_in       : integer := 6);
46                port        (encoder_in       : in std_logic_vector(no_bits_in-1 downto 0);
47                             H3               : in std_logic;
48                             enable_in        : in std_logic;
49                             encoder_out      : out std_logic_vector(no_bits_in-1 downto 0));
50          end component Priority_encoder;
```

```
51
52      subtype data_subtype is std_logic_vector(bits_per_num-1 downto 0);
53      type data_type is array ((no_nums-1) downto 0) of data_subtype;
54
55      signal shifter_data_in                          : data_type;
56      signal sorted_out_enable                        : std_logic;
57      signal temp_select_out                          : std_logic_vector(no_nums-1 downto 0);
58      signal temp_pri_encoder_out                     : std_logic_vector(no_nums-1 downto 0);
59      signal ROF_data_in                              : std_logic_vector(no_nums-1 downto 0);
60      signal temp_shifter_bit_out                     : std_logic_vector(no_nums-1 downto 0);
61      signal SAH_enable                               : std_logic;
62
63      begin
64       temp_ROF_data_in <= ROF_data_in;
65
66       enabler_Delay_12 : Delay_1
67                                 generic map       (delay_cycles => bits_per_num + 3)
68                                 port map          (signal_in => enable_in,
69                                                    H3 => H3,
70                                                    signal_out => enable_out);              -- enable_out
71
72          ----------------------input data for shiftreg----------------------------------
73       genA1:
74       for k in no_nums-1 downto 0 generate
75          genA2:
76        for j in bits_per_num-1 downto 0 generate
77                  shifter_data_in(k)(j) <= data_in(j,k);
78          end generate;
79       end generate;
80
81          ----------------------shiftreg-------------------------------------------------
82       gen1:
83       for i in no_nums-1 downto 0 generate
84          s1 : lpm_shiftreg
85                                 generic map       (lpm_width => bits_per_num)
86                                 port map          (data => shifter_data_in(i),
87                                                    clock => H3,
88                                                    load => enable_in,
89                                                    shiftout => temp_shifter_bit_out(i));
90
91          ROF_data_in(i) <= temp_shifter_bit_out(i);
92       end generate;
93
94
95       b1 : ROF_Bit_plane
96                                 generic map       (no_bits => no_nums)
97                                 port map          (data_in => ROF_data_in,
98                                                    H3 => H3,
99                                                    sset_in => enable_in,
100                                                   select_out => temp_select_out);
101
102         ----------------------priority encoder--
103
104      p1 : priority_encoder
105                                generic map       (no_bits_in => no_nums)
106                                port map          (encoder_in => temp_select_out,
107                                                   H3 => H3,
108                                                   enable_in => sorted_out_enable,
```

```
109                                     encoder_out => temp_pri_encoder_out);
110
111
112     s1 : Delay_1
113                         generic map     (delay_cycles => bits_per_num + 1)
114                         port map        (signal_in => enable_in,
115                                          H3 => H3,
116                                          signal_out => sorted_out_enable);
117
118     de1 : Delay_1
119                         generic map     (delay_cycles => 1)
120                         port map        (signal_in => sorted_out_enable,
121                                          H3 => H3,
122                                          signal_out => SAH_enable);
123
124     sample_and_hold1: SampleAndHold
125                         generic map     (width => no_nums)
126                         port map        (signal_in => temp_pri_encoder_out,
127                                          enable_in => SAH_enable,
128                                          H3 => H3,
129                                          signal_out => sorted_out);
130
131 end architecture A;
```

SORTER_RANKED.VHD


```
1   -- select line: active low...
2   library ieee;
3   use ieee.std_logic_1164.all;
4   use ieee.std_logic_arith.all;
5   use ieee.std_logic_unsigned.all;
6   library lpm;
7   use lpm.lpm_components.all;
8
9   entity Sorter_ranked is
10      generic     (bits_per_num                   : integer := 7;
11                   no_nums                        : integer := 4);
12      port        (data_in                        : in std_logic_2D(bits_per_num-1 downto 0, no_nums-1 downto
    0);
13                   H3                             : in std_logic;
14                   enable_in                      : in std_logic;
15                   enable_out                     : out std_logic;
16                   sorted_out                     : out std_logic_2D(no_nums-1 downto 0, no_nums-1 downto 0);
17                                                          --for dubugging purposes only:
18                   temp_repeating_pri_enabler     : out std_logic;
19                   temp_repeating_ROF_enabler     : out std_logic;
20                   temp_mux_select_out            : out std_logic_vector(no_nums-1 downto 0);
21                   temp_ROF_data_in               : out std_logic_vector(no_nums-1 downto 0);
22                   select_out                     : out std_logic_vector(no_nums-1 downto 0);
23                   pri_encoder_out                : out std_logic_vector(no_nums-1 downto 0);
24                   sampler_enable_out             : out std_logic_vector(no_nums-1 downto 0) );
25  end entity Sorter_ranked;
26
27  architecture A of Sorter_ranked is
28      component EnablerRepeat is
29          generic     (delay_cycles               : integer := 3;
30                       repeats                    : integer := 4);
```

```
31          port        (enable_in              : in std_logic;
32                      H3                      : in std_logic;
33                      enable_out              : out std_logic);
34      end component EnablerRepeat;
35
36      component lpm_latch
37          generic     (LPM_WIDTH              : POSITIVE;
38                      LPM_AVALUE              : STRING := "UNUSED";
39                      LPM_PVALUE              : STRING := "UNUSED";
40                      LPM_TYPE                : STRING := "LPM_LATCH";
41                      LPM_HINT                : STRING := "UNUSED");
42          port        (data                   : in STD_LOGIC_VECTOR(LPM_WIDTH-1 downto 0);
43                      gate: in STD_LOGIC;
44                      aclr, aset: in STD_LOGIC := '0';
45                      q: out STD_LOGIC_VECTOR(LPM_WIDTH-1 downto 0));
46      end component;
47
48      component mux
49          generic     (WIDTH                  : POSITIVE;
50                      WIDTHS                  : POSITIVE);
51          port        (data                   : in STD_LOGIC_VECTOR(WIDTH-1 downto 0);
52                      sel                     : in STD_LOGIC_VECTOR(WIDTHS-1 downto 0);
53                      result                  : out STD_LOGIC);
54      end component;
55
56      component DFF
57          port        (d                      : in STD_LOGIC;
58                      clk                     : in STD_LOGIC;
59                      clrn                    : in STD_LOGIC;
60                      prn                     : in STD_LOGIC;
61                      q                       : out STD_LOGIC );
62      end component;
63
64      component SampleAndHold is
65          generic     (width                  : integer := 5);
66          port        (signal_in              : in std_logic_vector(width-1 downto 0);
67                      enable_in               : in std_logic;
68                      H3                      : in std_logic;
69                      signal_out              : out std_logic_vector(width-1 downto 0) );
70      end component SampleAndHold;
71
72      component Priority_encoder is
73          generic     (no_bits_in             : integer := 6);
74          port        (encoder_in             : in std_logic_vector(no_bits_in-1 downto 0);
75                      H3                      : in std_logic;
76                      enable_in               : in std_logic;
77                      encoder_out             : out std_logic_vector(no_bits_in-1 downto 0));
78      end component Priority_encoder;
79
80      component Latch_with_sset is
81          port        (enable_in              : in std_logic;
82                      sset_in                 : in std_logic;
83                      H3                      : in std_logic;
84                      q                       : out std_logic);
85      end component Latch_with_sset;
86
87      component ROF_Bit_plane is
88          generic      (no_bits               : integer := 7);
```

```
89              port        (data_in                : in std_logic_vector(no_bits-1 downto 0);
90                          H3                      : in std_logic;
91                          sset_in                 : in std_logic;
92                          select_out              : out std_logic_vector(no_bits-1 downto 0));
93          end component ROF_Bit_plane;
94
95          component Delay_1 is
96              generic     (delay_cycles           : integer := 3);
97              port    (signal_in                  : in std_logic;
98                          H3                      : in std_logic;
99                          signal_out              : out std_logic);
100         end component Delay_1;
101
102     subtype data_subtype is std_logic_vector(bits_per_num-1 downto 0);
103     type data_type is array ((no_nums-1) downto 0) of data_subtype;
104
105     type output_type is array (no_nums-1 downto 0) of std_logic_vector(no_nums-1 downto 0);
106
107     signal shifter_data_in                      : data_type;
108     signal ROF_data_in                          : std_logic_vector(no_nums-1 downto 0);
109     signal repeating_priority_enabler           : std_logic;
110     signal repeating_ROF_enabler                : std_logic;
111     signal temp_ROF_enable_in                   : std_logic;
112     signal delayed_reset_for_ROF                : std_logic;
113     signal ROF_reset                            : std_logic;
114     signal temp_select_out                      : std_logic_vector(no_nums-1 downto 0);
115     signal temp_pri_encoder_out                 : std_logic_vector(no_nums-1 downto 0);
116     signal temp_shifter_bit_in                  : std_logic_vector(no_nums-1 downto 0);
117     signal temp_shifter_bit_out                 : std_logic_vector(no_nums-1 downto 0);
118     signal temp_mux_select                      : std_logic_vector(no_nums-1 downto 0);
119     signal temp_pri_in                          : std_logic_vector(no_nums-1 downto 0);
120     signal sorted_out_enable                    : std_logic_vector(no_nums-1 downto 0);
121     signal temp_output                          : output_type;
122     signal enable_in_delayed1                   : std_logic;
123
124     begin
125      temp_ROF_data_in <= ROF_data_in;
126      select_out <= temp_select_out;
127      temp_repeating_pri_enabler <= repeating_priority_enabler;
128      temp_repeating_ROF_enabler <= repeating_ROF_enabler;
129         ----------------------enables-------------------------------------------------------
130         ----------------------enable line out, specifies priority encoder sampling instant--
131
132      enabler_1 : EnablerRepeat
133                              generic map     (delay_cycles => bits_per_num+2,
134                                               repeats => no_nums)
135                              port map        (enable_in => enable_in,
136                                               H3 => H3,
137                                               enable_out => repeating_priority_enabler);
138
139      enabler_Delay_1 : Delay_1
140                              generic map     (delay_cycles => 1)
141                              port map        (signal_in => repeating_priority_enabler or enable_in,
142                                               H3 => H3,
143                                               signal_out => repeating_ROF_enabler);
144
145      enabler_Delay_12 : Delay_1
146                              generic map      (delay_cycles => no_nums*(bits_per_num + 2)+2)
```

```
147                         port map        (signal_in => enable_in,
148                                         H3 => H3,
149                                         signal_out => enable_out);          -- enable_out
150
151     enabler_Delay_3 : Delay_1
152                         generic map     (delay_cycles => 1)
153                         port map        (signal_in => enable_in,
154                                         H3 => H3,
155                                         signal_out => enable_in_delayed1);  -- for initial shifteg load op.
156
157             ----------------------modified ROF bit plane, for a full sort----------------------
158     b1 : ROF_Bit_plane
159                         generic map     (no_bits => no_nums)
160                         port map        (data_in => ROF_data_in,
161                                         H3 => H3,
162                                         sset_in => repeating_ROF_enabler,
163                                         select_out => temp_select_out);
164
165       ----------------------priority encoder--
166     p1 : priority_encoder
167                         generic map     (no_bits_in => no_nums)
168                         port map        (encoder_in => temp_pri_in,
169                                         H3 => H3,
170                                         enable_in => repeating_priority_enabler,
171                                         encoder_out => temp_pri_encoder_out);
172
173     pri_encoder_out <= temp_pri_encoder_out;
174
175       ----------------------input data for shiftreg---------------------------------
176     genA1:
177     for k in no_nums-1 downto 0 generate
178       genA2:
179      for j in bits_per_num-1 downto 0 generate
180             shifter_data_in(k)(j) <= data_in(j,k);
181       end generate;
182     end generate;
183
184       ----------------------shiftreg-----------------------------------------------
185     gen1:
186     for i in no_nums-1 downto 0 generate
187       s1 : lpm_shiftreg
188                         generic map     (lpm_width => bits_per_num)
189                         port map        (data => shifter_data_in(i),
190                                         clock => H3,
191                                         load => enable_in_delayed1,
192                                         shiftin => temp_shifter_bit_in(i),
193                                         sclr => temp_pri_encoder_out(i),
194                                         shiftout => temp_shifter_bit_out(i));
195
196       ROF_data_in(i) <= temp_shifter_bit_out(i);
197
198       shiftreg_delay2 : Delay_1
199                         generic map     (delay_cycles =>2)
200                         port map        (signal_in => temp_shifter_bit_out(i),
201                                         H3 => H3,
202                                         signal_out => temp_shifter_bit_in(i));
203     end generate;
204
```

```
205         ----------------------multiplexers and latches, to differentiate between real and reset zero values--
206       gen:
207       for i in no_nums-1 downto 0 generate
208         Mux_for_zeros: mux
209                         generic map         (width => 2, widths => 1)
210                         port map            (data(0) => '0',
211                                              data(1) => temp_select_out(i),
212                                              sel(0) => temp_mux_select(i),
213                                              result => temp_pri_in(i));
214         Latch_for_zeros: Latch_with_sset
215                                 port map    (enable_in => temp_pri_encoder_out(i),
216                                              sset_in => enable_in,
217                                              H3 => H3,
218                                              q => temp_mux_select(i));
219       end generate;
220
221     temp_mux_select_out <= temp_mux_select;
222     --============================================================================
223
224       gen_a1:
225       for i in no_nums-1 downto 0 generate
226         step_delay_output : Delay_1
227                         generic map         (delay_cycles => (i)*(bits_per_num+2)+(2+bits_per_num)+1)
228                         port map            (signal_in => enable_in,
229                                              H3 => H3,
230                                              signal_out => sorted_out_enable(i));
231
232         sample_and_hold1: SampleAndHold
233                         generic map         (width => no_nums)
234                         port map            (signal_in => temp_pri_encoder_out,
235                                              enable_in => sorted_out_enable(i),
236                                              H3 => H3,
237                                              signal_out => temp_output(no_nums-1 - i));
238       end generate;
239
240     sampler_enable_out <= sorted_out_enable;
241
242       genB1:
243       for k in no_nums-1 downto 0 generate
244         genB2:
245         for j in no_nums-1 downto 0 generate
246                 sorted_out(k,j) <= temp_output(j)(k);
247         end generate;
248       end generate;
249
250     end architecture A;
```

STATE_SW_FN_GEN.VHD

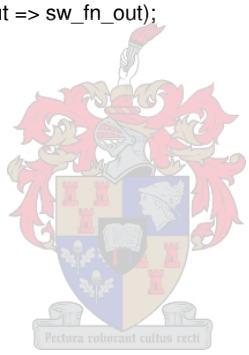
```
251     library ieee;
252     use ieee.std_logic_1164.all;
253     use ieee.std_logic_arith.all;
254     use ieee.std_logic_unsigned.all;
255     library lpm;
256     use lpm.lpm_components.all;
257
258     entity state_sw_fn_gen is
```

```
259             generic     (no_cells                   : integer := 3);
260             port        (sw_fns_out                 : out std_logic_2D(2**no_cells-1 downto 0, no_cells-1 downto 0));
261     end entity state_sw_fn_gen;
262
263     architecture C of state_sw_fn_gen is
264         component lpm_constant is
265             generic     (LPM_WIDTH                  : POSITIVE;
266                          LPM_CVALUE                 : NATURAL;
267                          LPM_STRENGTH               : STRING := "UNUSED";
268                          LPM_TYPE                   : STRING := "LPM_CONSTANT";
269                          LPM_HINT                   : STRING := "UNUSED");
270             port        (result                     : out STD_LOGIC_VECTOR(LPM_WIDTH-1 downto 0));
271         end component;
272
273     subtype sw_subtype is std_logic_vector( no_cells-1 downto 0);
274     type sw_type is array ((2**no_cells-1) downto 0) of sw_subtype;
275
276     signal
277       temp : sw_type;
278
279     begin
280       gen1: for i in 2**no_cells-1 downto 0 generate
281         u1 : lpm_constant
282             generic map
283                 (lpm_width => no_cells,
284                  lpm_cvalue => conv_integer(i))
285             port map
286                 (result => temp(i));
287
288         gen2:
289         for j in no_cells-1 downto 0 generate
290             sw_fns_out(i,j) <= temp(i)(j);
291         end generate;
292       end generate;
293
294     end architecture C;
```

SW_FN_SELECT.VHD

```
1       library ieee;
2       use ieee.std_logic_1164.all;
3       use ieee.std_logic_arith.all;
4       use ieee.std_logic_unsigned.all;
5       library lpm;
6       use lpm.lpm_components.all;
7
8       entity sw_fn_select is
9           generic     (no_cells                   : integer := 3);
10          port        (sw_fns_in                  : in std_logic_2D(2**no_cells-1 downto 0, no_cells-1 downto 0);
11                       enable_in                  : in std_logic;
12                       index_in                   : in std_logic_vector(2**no_cells-1 downto 0);
13                       H3                         : in std_logic;
14                       enable_out                 : out std_logic;
15                       sw_fn_out                  : out std_logic_vector(no_cells-1 downto 0));
16      end entity sw_fn_select;
17
18      architecture G of sw_fn_select is
```

```
19      component Custom_MUX is
20      generic     (bits_per_channel           : integer := 10;
21                   no_channels_in             : integer := 6);
22      port        (channels_in                : in std_logic_2D(no_channels_in-1 downto 0, bits_per_channel-1
    downto 0);
23                   H3                         : in std_logic;
24                   enable_in                  : in std_logic;
25                   enable_out                 : out std_logic;
26                   sel_in                     : in std_logic_vector(no_channels_in-1 downto 0);
27                   channel_out                : out std_logic_vector(bits_per_channel-1 downto 0));
28      end component Custom_MUX;
29
30
31  begin
32      c1 : Custom_MUX
33          generic map     (bits_per_channel => no_cells,
34                           no_channels_in => 2**no_cells)
35          port map        (channels_in => sw_fns_in,
36                           H3 =>      H3,
37                           enable_in => enable_in,
38                           enable_out => enable_out,
39                           sel_in => index_in,
40                           channel_out => sw_fn_out);
41  end architecture G;
```



Appendix D – MATLAB source code

The following two MATLAB files were used to generate the results in Figure 5.10 and Figure 5.11.

SPACE_VECTOR_PLOT.M

```
clear all;

const_T         = 1;
const_first     = 2;
const_Park_beta  = 3;
const_Park_alpha = 4;
format long;
global A;
close all
figure(1)
hold on
texth = 0.02;
A = dlmread('c:\Tesis-sim\final\af_3f_5s_nf_vs2dF.d',';');
cells = 5;

celanovic_with_t(cells)

B = [];
V_alpha_out  = [A(:,const_Park_alpha)];
V_beta_out   = [A(:,const_Park_beta)];
first = A(:,const_first);

figure(3)
hold on;
celanovic_with_t(cells)
prev_beta = V_beta_out(1);
prev_alpha = V_alpha_out(1);
counter = 1;

D(1,1) = A(1,const_T);
D(1,2) = V_alpha_out(1);
D(1,3) = V_beta_out(1);

%for loop = 2:(length(V_beta_out)/4)
for loop = 2:(length(V_beta_out))
  if (prev_beta ~= V_beta_out(loop)) | (prev_alpha ~= V_alpha_out(loop))
      counter = counter +1;
     a = [num2str(counter)];
     D(counter,1) = A(loop,const_T);
     D(counter,2) = V_alpha_out(loop);
     D(counter,3) = V_beta_out(loop);

%      plot(V_alpha_out(loop), V_beta_out(loop),'.r')
%      handle = text(V_alpha_out(loop)+0.03, V_beta_out(loop)+0.03, a);
```

```
45      %       set(handle, 'fontsize',7)
46      %       set(handle,'fontweight','bold')
47
48              t = num2str(A(loop,const_T));
49
50
51              plot(V_alpha_out(loop), V_beta_out(loop),'.r')
52              prev_beta = V_beta_out(loop);
53              prev_alpha = V_alpha_out(loop);
54          end;
55      end;
```

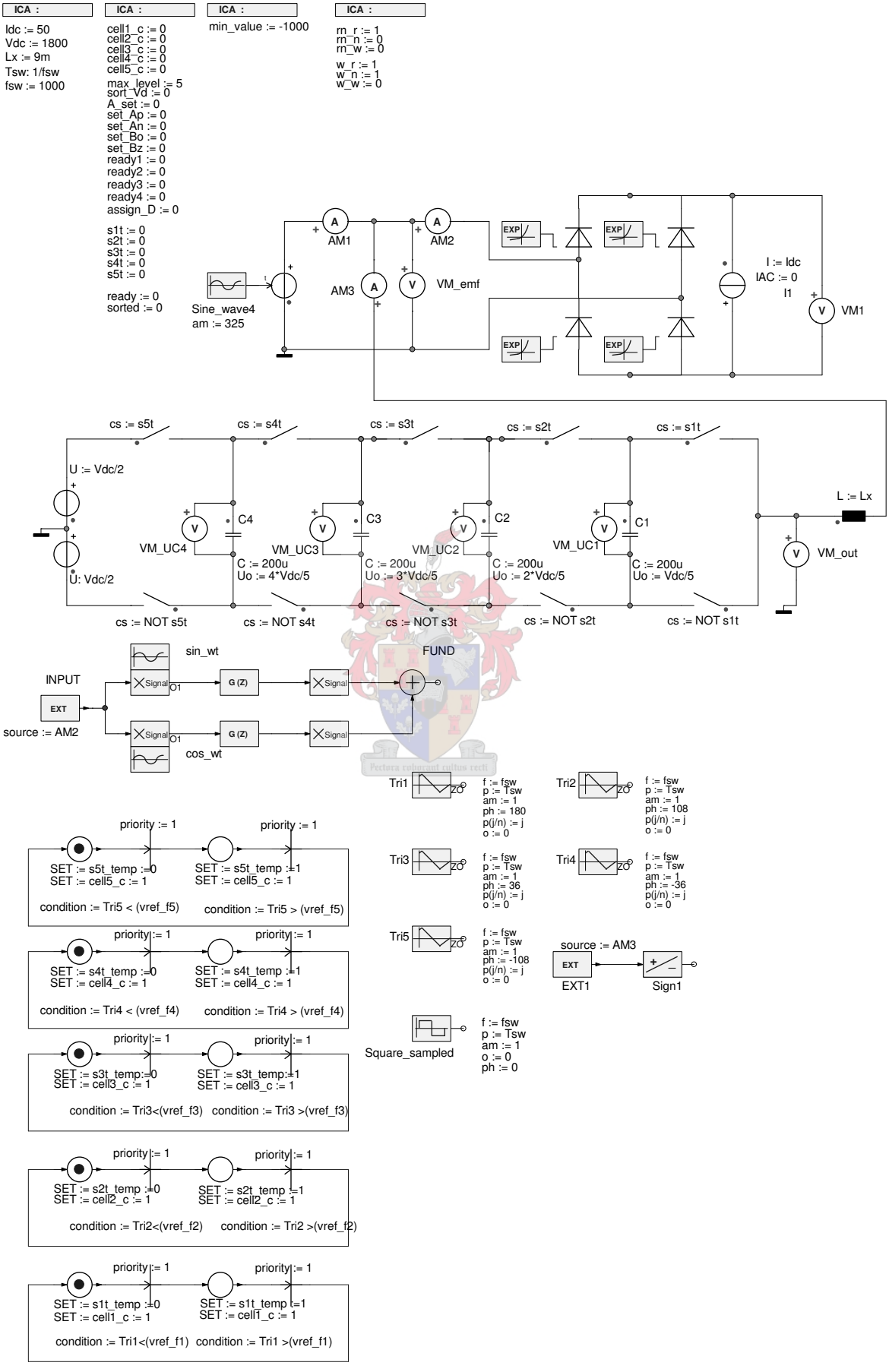
CELANOVIC_WITH_T.M

```
1       function [] = Celanovic_with_t(cells)
2
3       global AA_gh AA_gh_inv
4                   % AA_gh     : transformation matrix: from alpha-beta values to g-h values
5                   % AA_gh_inv : transformation matrix: from g-h values to alpha-beta values
6
7       AA_gh = cells*[1 -1/sqrt(3); 0 2/sqrt(3)];
8       AA_gh_inv = inv(AA_gh);
9
10      Aalphabeta = [1 -0.5 -0.5; 0 sqrt(3)/2 -sqrt(3)/2];
11      hold on
12      tolerance = 0.001;
13      texth = 0.15;
14
15      for a = 0:cells
16          for b = 0:cells
17              for c = 0:cells
18                  V = [a/cells; b/cells; c/cells];                    % line-to-neutral
19                  Valphabeta = Aalphabeta*V;
20
21                  alpha = Valphabeta(1,1); if abs(alpha) <0.0001;  alpha = 0; end;
22                  beta = Valphabeta(2,1);  if abs(beta) <0.0001 ; beta = 0; end;
23                  T_alphabeta = ['(' num2str(alpha,2) ',' num2str(beta,2) ')' ];
24
25                  V_gh = AA_gh * Valphabeta;
26                  g = V_gh(1,1); if abs(g) <0.0001;  g = 0; end;
27                  h = V_gh(2,1); if abs(h) <0.0001;  h = 0; end;
28                  text_gh = ['(' num2str(g)  ','  num2str(h) ')'];
29
30                  T = [  num2str(a) num2str(b) num2str(c)  T_alphabeta];
31                  handle = text(alpha+0.001, beta+texth*(0.15), [text_gh]);
32                  set(handle, 'fontsize',6)
33                  set(handle,'fontweight','bold')
34                  plot(alpha, beta, 'x')
35
36                  g1=g+1; h1=h; temp = AA_gh_inv*[g1;h1];                %horizontal line 2
37                  if ((abs(g1) -cells) <= tolerance) & ((abs(h1) -cells) <= tolerance) & ((abs(g1+h1) -cells) <= tolerance)
38                  l = line([alpha temp(1,1)], [beta temp(2,1)]);
39                  set(l, 'linestyle', ':');
40                  end
41
42
43                  g1=g; h1=h+1; temp = AA_gh_inv*[g1;h1];                %line parallel to h-axis
```

```
44              if ((abs(g1) -cells ) <= tolerance) & ((abs(h1) -cells) <= tolerance) & ((abs(g1+h1) -cells) <= tolerance)
45              l = line([alpha temp(1,1)], [beta temp(2,1)]);
46              set(l, 'linestyle', ':');
47              end
48
49              g1=g+1; h1=h-1; temp = AA_gh_inv*[g1;h1];                    %other lines
50              if ((abs(g1) -cells) <= tolerance) & ((abs(h1) -cells) <= tolerance) & ((abs(g1+h1) -cells) <= tolerance)
51              l = line([alpha temp(1,1)], [beta temp(2,1)]);
52              set(l, 'linestyle', ':');
53              end
54            end
55          end
56        end
```

Appendix E – Simplorer simulation models

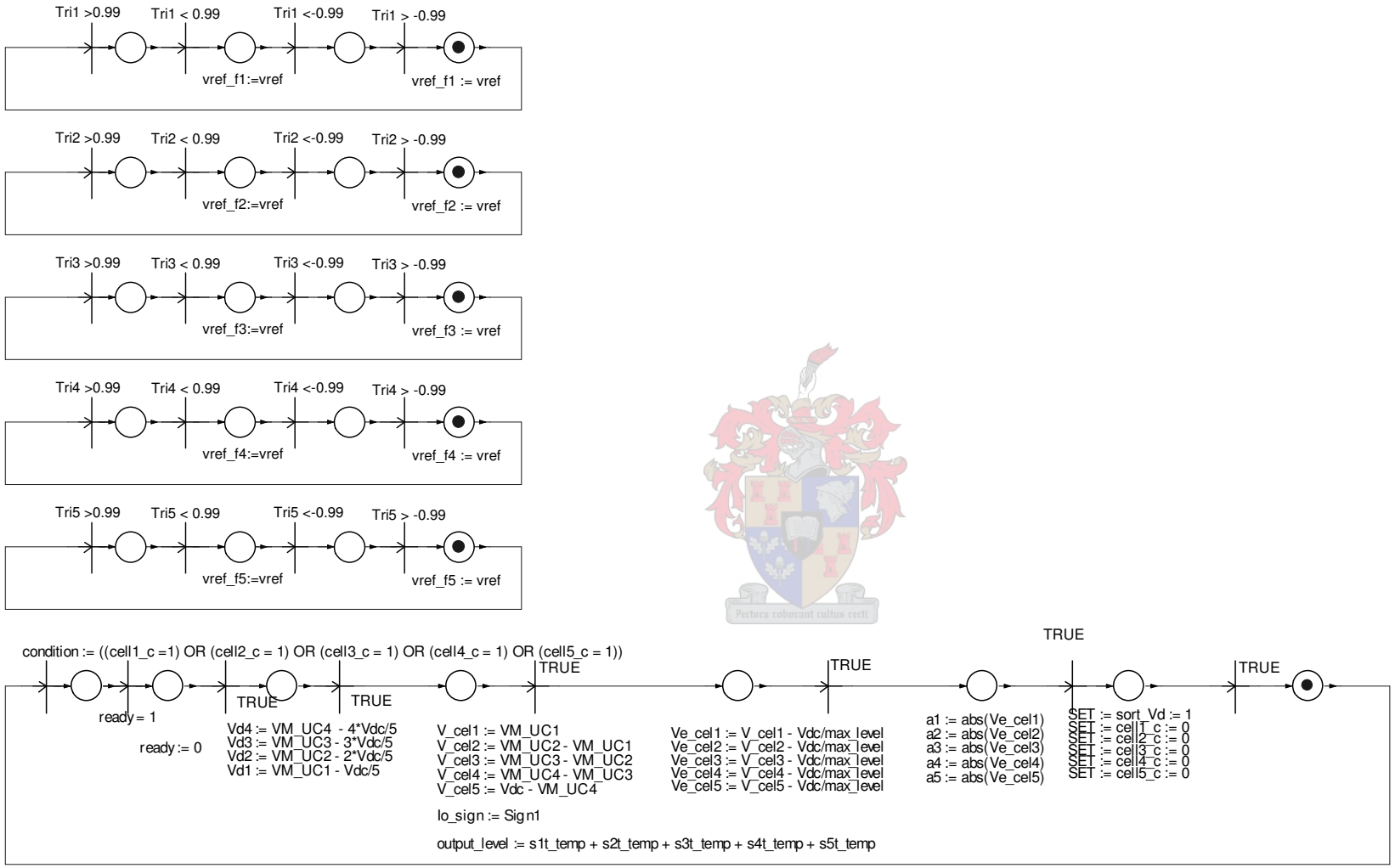
PER-PHASE MODEL – DONZEL AND BORNARD ALGORITHM

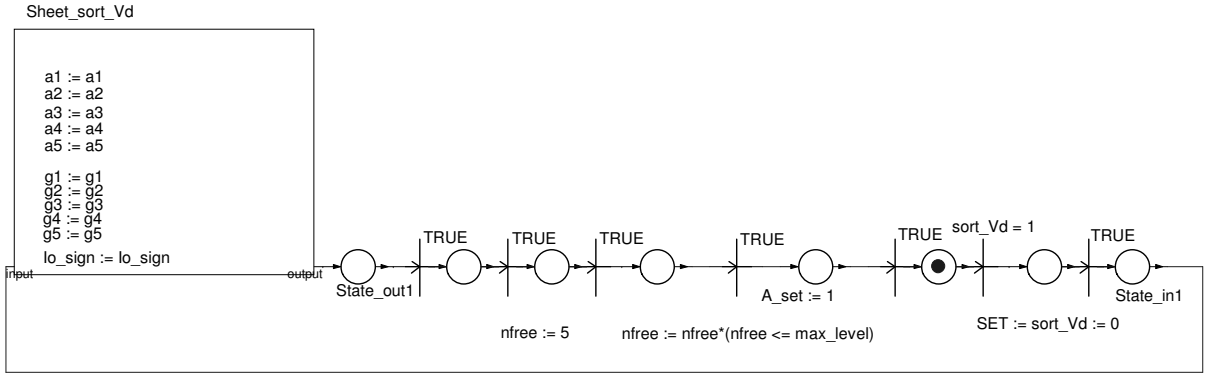

ICA :
Idc := 50
Vdc := 1800
Lx := 9m
Tsw: 1/fsw
fsw := 1000
ICA :
cell1_c := 0
cell2_c := 0
cell3_c := 0
cell4_c := 0
cell5_c := 0
max_level := 5
sort_Vd := 0
A_set := 0
set_Ap := 0
set_An := 0
set_Bo := 0
set_Bz := 0
ready1 := 0
ready2 := 0
ready3 := 0
ready4 := 0
assign_D := 0
s1t := 0
s2t := 0
s3t := 0
s4t := 0
s5t := 0
ready := 0
sorted := 0
ICA :
min_value := -1000
ICA :
rn_r := 1
rn_n := 0
rn_w := 0
w_r := 1
w_n := 1
w_w := 0
AM1
AM2
AM3
VM_emf
Sine_wave4
am := 325
EXP
I := Idc
IAC := 0
I1
VM1
cs := s5t
cs := s4t
cs := s3t
cs := s2t
cs := s1t
U := Vdc/2
U: Vdc/2
C4
C3
C2
C1
VM_UC4
VM_UC3
VM_UC2
VM_UC1
C := 200u
Uo := 4*Vdc/5
Uo := 3*Vdc/5
Uo := 2*Vdc/5
Uo := Vdc/5
L := Lx
VM_out
cs := NOT s5t
cs := NOT s4t
cs := NOT s3t
cs := NOT s2t
cs := NOT s1t
INPUT
EXT
source := AM2
sin_wt
cos_wt
Signal
G (Z)
FUND
Tri1
Tri2
Tri3
Tri4
Tri5
f := fsw
p := Tsw
am := 1
ph := 180
ph := 108
ph := 36
ph := -108
p(j/n) := j
o := 0
Square_sampled
source := AM3
EXT1
Sign1
priority := 1
SET := s5t_temp := 0
SET := cell5_c := 1
SET := s5t_temp := 1
SET := cell5_c := 1
condition := Tri5 < (vref_f5)
condition := Tri5 > (vref_f5)
SET := s4t_temp := 0
SET := cell4_c := 1
SET := s4t_temp := 1
SET := cell4_c := 1
condition := Tri4 < (vref_f4)
condition := Tri4 > (vref_f4)
SET := s3t_temp := 0
SET := cell3_c := 1
SET := s3t_temp := 1
SET := cell3_c := 1
condition := Tri3<(vref_f3)
condition := Tri3 >(vref_f3)
SET := s2t_temp := 0
SET := cell2_c := 1
SET := s2t_temp := 1
SET := cell2_c := 1
condition := Tri2<(vref_f2)
condition := Tri2 >(vref_f2)
SET := s1t_temp := 0
SET := cell1_c := 1
SET := s1t_temp := 1
SET := cell1_c := 1
condition := Tri1<(vref_f1)
condition := Tri1 >(vref_f1)

ICA :	ICA :	ICA :	ICA :	ICA :
A5 := 1	A4 := 1	A3 := 1	A2 := 1	A1 := 0
B5 := 1	B4 := 1	B3 := 1	B2 := 0	B1 := 1
C5 := 1	C4 := 1	C3 := 0	C2 := 1	C1 := 1
D5 := 1	D4 := 0	D3 := 1	D2 := 1	D1 := 1
E5 := 0	E4 := 1	E3 := 1	E2 := 1	E1 := 1
F5 := 1	F4 := 1	F3 := 1	F2 := 0	F1 := 0
G5 := 1	G4 := 1	G3 := 0	G2 := 1	G1 := 0
H5 := 1	H4 := 1	H3 := 0	H2 := 0	H1 := 1
I5 := 1	I4 := 0	I3 := 1	I2 := 1	I1 := 0
J5 := 1	J4 := 0	J3 := 1	J2 := 0	J1 := 1
K5 := 1	K4 := 0	K3 := 0	K2 := 1	K1 := 1
L5 := 0	L4 := 1	L3 := 1	L2 := 1	L1 := 0
M5 := 0	M4 := 1	M3 := 1	M2 := 0	M1 := 1
Q5 := 0	Q4 := 1	Q3 := 0	Q2 := 1	Q1 := 1
R5 := 0	R4 := 0	R3 := 1	R2 := 1	R1 := 1
S5 := 1	S4 := 1	S3 := 0	S2 := 0	S1 := 0
T5 := 1	T4 := 0	T3 := 1	T2 := 0	T1 := 0
U5 := 1	U4 := 0	U3 := 0	U2 := 1	U1 := 0
V5 := 1	V4 := 0	V3 := 0	V2 := 0	V1 := 1
W5 := 0	W4 := 1	W3 := 1	W2 := 0	W1 := 0
X5 := 0	X4 := 1	X3 := 0	X2 := 1	X1 := 0
Y5 := 0	Y4 := 1	Y3 := 0	Y2 := 0	Y1 := 1
Z5 := 0	Z4 := 0	Z3 := 1	Z2 := 1	Z1 := 0
AA5 := 0	AA4 := 0	AA3 := 1	AA2 := 0	AA1 := 1
AB5 := 0	AB4 := 0	AB3 := 0	AB2 := 1	AB1 := 1
AC5 := 1	AC4 := 0	AC3 := 0	AC2 := 0	AC1 := 0
AD5 := 0	AD4 := 1	AD3 := 0	AD2 := 0	AD1 := 0
AE5 := 0	AE4 := 0	AE3 := 1	AE2 := 0	AE1 := 0
AF5 := 0	AF4 := 0	AF3 := 0	AF2 := 1	AF1 := 0
AG5 := 0	AG4 := 0	AG3 := 0	AG2 := 0	AG1 := 1
P5 := 1	P4 := 1	P3 := 1	P2 := 1	P1 := 1
N5 := 0	N4 := 0	N3 := 0	N2 := 0	N1 := 0

C1_charge_ref ICA :	C2_charge_ref ICA :	C3_charge_ref ICA :	C4_charge_ref ICA :
A_c1_ref := 1	A_c2_ref := 2	A_c3_ref := 2	A_c4_ref := 2
B_c1_ref := 0	B_c2_ref := 1	B_c3_ref := 2	B_c4_ref := 2
C_c1_ref := 2	C_c2_ref := 0	C_c3_ref := 1	C_c4_ref := 2
D_c1_ref := 2	D_c2_ref := 2	D_c3_ref := 0	D_c4_ref := 1
E_c1_ref := 2	E_c2_ref := 2	E_c3_ref := 2	E_c4_ref := 0
F_c1_ref := 2	F_c2_ref := 1	F_c3_ref := 2	F_c4_ref := 2
G_c1_ref := 1	G_c2_ref := 0	G_c3_ref := 1	G_c4_ref := 2
H_c1_ref := 0	H_c2_ref := 2	H_c3_ref := 1	H_c4_ref := 2
I_c1_ref := 1	I_c2_ref := 2	I_c3_ref := 0	I_c4_ref := 1
J_c1_ref := 0	J_c2_ref := 1	J_c3_ref := 0	J_c4_ref := 1
K_c1_ref := 2	K_c2_ref := 0	K_c3_ref := 2	K_c4_ref := 1
L_c1_ref := 1	L_c2_ref := 2	L_c3_ref := 2	L_c4_ref := 0
M_c1_ref := 0	M_c2_ref := 1	M_c3_ref := 2	M_c4_ref := 0
Q_c1_ref := 2	Q_c2_ref := 0	Q_c3_ref := 1	Q_c4_ref := 0
R_c1_ref := 2	R_c2_ref := 2	R_c3_ref := 0	R_c4_ref := 2
S_c1_ref := 2	S_c2_ref := 2	S_c3_ref := 1	S_c4_ref := 2
T_c1_ref := 2	T_c2_ref := 1	T_c3_ref := 0	T_c4_ref := 1
U_c1_ref := 1	U_c2_ref := 0	U_c3_ref := 2	U_c4_ref := 1
V_c1_ref := 0	V_c2_ref := 2	V_c3_ref := 2	V_c4_ref := 1
W_c1_ref := 2	W_c2_ref := 1	W_c3_ref := 2	W_c4_ref := 0
X_c1_ref := 1	X_c2_ref := 0	X_c3_ref := 1	X_c4_ref := 0
Y_c1_ref := 0	Y_c2_ref := 2	Y_c3_ref := 1	Y_c4_ref := 0
Z_c1_ref := 1	Z_c2_ref := 2	Z_c3_ref := 0	Z_c4_ref := 2
AA_c1_ref := 0	AA_c2_ref := 1	AA_c3_ref := 0	AA_c4_ref := 2
AB_c1_ref := 2	AB_c2_ref := 0	AB_c3_ref := 2	AB_c4_ref := 2
AC_c1_ref := 2	AC_c2_ref := 2	AC_c3_ref := 2	AC_c4_ref := 1
AD_c1_ref := 2	AD_c2_ref := 2	AD_c3_ref := 1	AD_c4_ref := 0
AE_c1_ref := 2	AE_c2_ref := 1	AE_c3_ref := 0	AE_c4_ref := 2
AF_c1_ref := 1	AF_c2_ref := 0	AF_c3_ref := 2	AF_c4_ref := 2
AG_c1_ref := 0	AG_c2_ref := 2	AG_c3_ref := 2	AG_c4_ref := 2
P_c1_ref := 2	P_c2_ref := 2	P_c3_ref := 2	P_c4_ref := 2
N_c1_ref := 2	N_c2_ref := 2	N_c3_ref := 2	N_c4_ref := 2

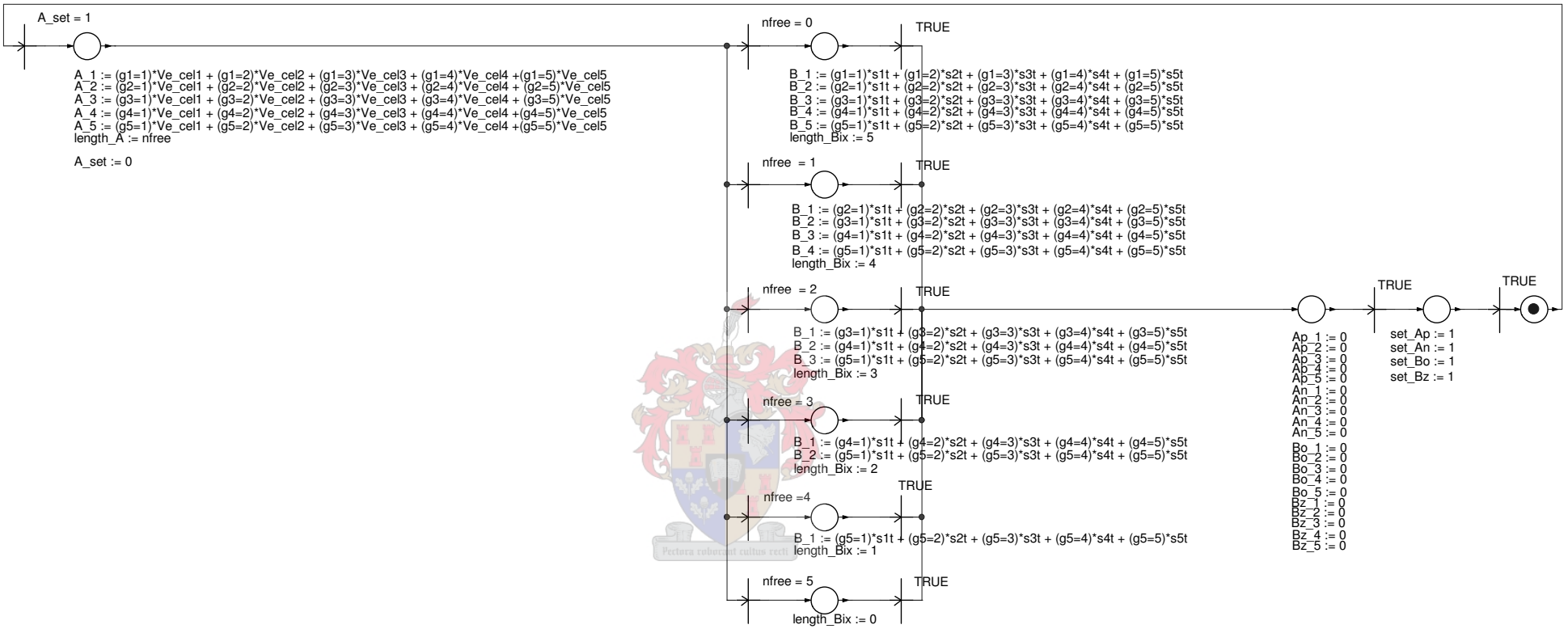
VA1 :
A_level := A5 + A4 + A3 + A2 + A1
B_level := B5 + B4 + B3 + B2 + B1
C_level := C5 + C4 + C3 + C2 + C1
D_level := D5 + D4 + D3 + D2 + D1
E_level := E5 + E4 + E3 + E2 + E1
F_level := F5 + F4 + F3 + F2 + F1
G_level := G5 + G4 + G3 + G2 + G1
H_level := H5 + H4 + H3 + H2 + H1
I_level := I5 + I4 + I3 + I2 + I1
J_level := J5 + J4 + J3 + J2 + J1
K_level := K5 + K4 + K3 + K2 + K1
L_level := L5 + L4 + L3 + L2 + L1
M_level := M5 + M4 + M3 + M2 + M1
Q_level := Q5 + Q4 + Q3 + Q2 + Q1
R_level := R5 + R4 + R3 + R2 + R1
S_level := S5 + S4 + S3 + S2 + S1
T_level := T5 + T4 + T3 + T2 + T1
U_level := U5 + U4 + U3 + U2 + U1
V_level := V5 + V4 + V3 + V2 + V1
W_level := W5 + W4 + W3 + W2 + W1
X_level := X5 + X4 + X3 + X2 + X1
Y_level := Y5 + Y4 + Y3 + Y2 + Y1
Z_level := Z5 + Z4 + Z3 + Z2 + Z1
AA_level := AA5 + AA4 + AA3 + AA2 + AA1
AB_level := AB5 + AB4 + AB3 + AB2 + AB1
AC_level := AC5 + AC4 + AC3 + AC2 + AC1
AD_level := AD5 + AD4 + AD3 + AD2 + AD1
AE_level := AE5 + AE4 + AE3 + AE2 + AE1
AF_level := AF5 + AF4 + AF3 + AF2 + AF1
AG_level := AG5 + AG4 + AG3 + AG2 + AG1
P_level := P5 + P4 + P3 + P2 + P1
N_level := N5 + N4 + N3+ N2 + N1

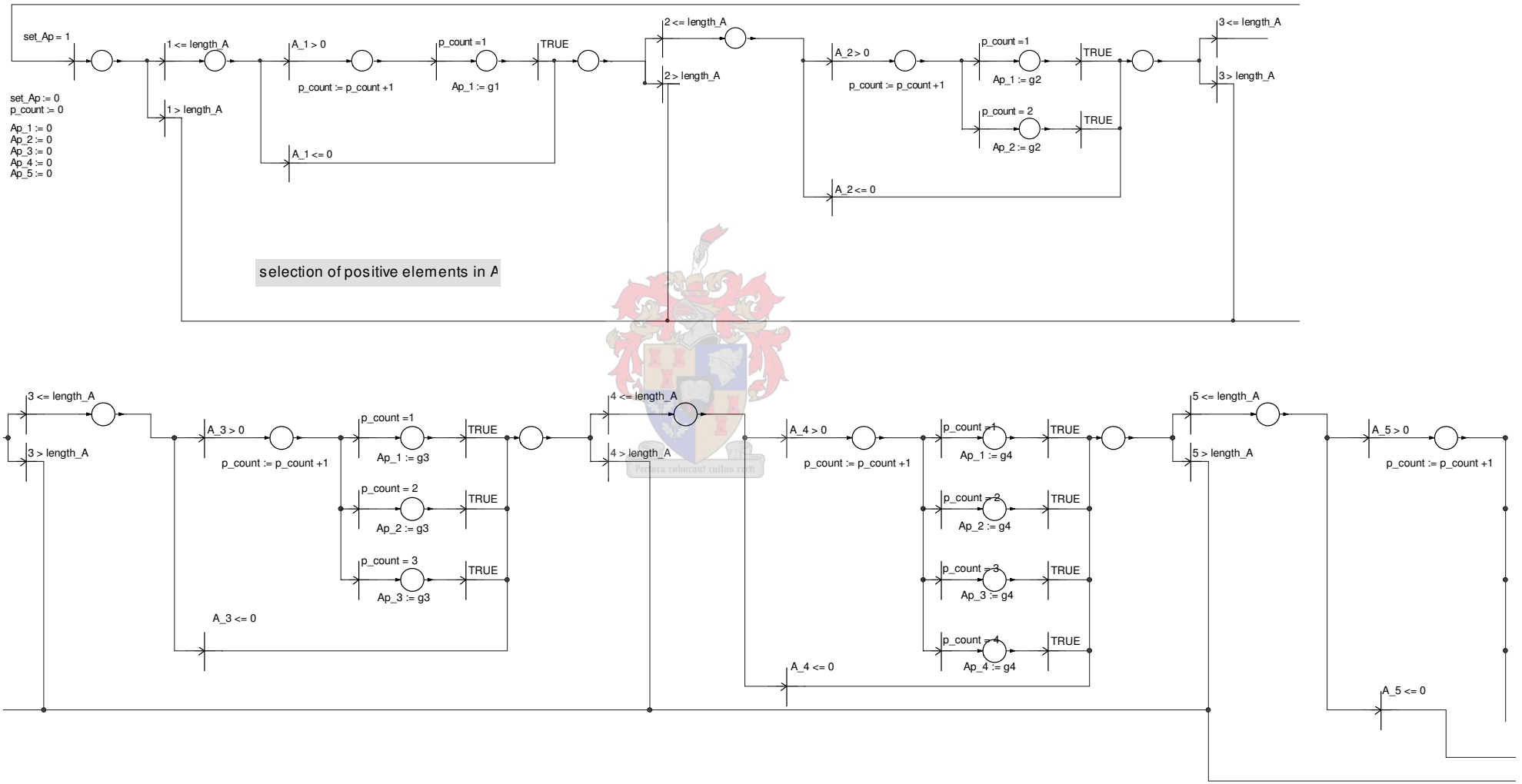

Tri1 >0.99
Tri1 < 0.99
Tri1 <-0.99
Tri1 > -0.99
vref_f1:=vref
vref_f1 := vref
Tri2 >0.99
Tri2 < 0.99
Tri2 <-0.99
Tri2 > -0.99
vref_f2:=vref
vref_f2 := vref
Tri3 >0.99
Tri3 < 0.99
Tri3 <-0.99
Tri3 > -0.99
vref_f3:=vref
vref_f3 := vref
Tri4 >0.99
Tri4 < 0.99
Tri4 <-0.99
Tri4 > -0.99
vref_f4:=vref
vref_f4 := vref
Tri5 >0.99
Tri5 < 0.99
Tri5 <-0.99
Tri5 > -0.99
vref_f5:=vref
vref_f5 := vref
condition := ((cell1_c =1) OR (cell2_c = 1) OR (cell3_c = 1) OR (cell4_c = 1) OR (cell5_c = 1))
ready = 1
ready := 0
TRUE
TRUE
Vd4 := VM_UC4 - 4*Vdc/5
Vd3 := VM_UC3 - 3*Vdc/5
Vd2 := VM_UC2 - 2*Vdc/5
Vd1 := VM_UC1 - Vdc/5
TRUE
V_cel1 := VM_UC1
V_cel2 := VM_UC2 - VM_UC1
V_cel3 := VM_UC3 - VM_UC2
V_cel4 := VM_UC4 - VM_UC3
V_cel5 := Vdc - VM_UC4
lo_sign := Sign1
output_level := s1t_temp + s2t_temp + s3t_temp + s4t_temp + s5t_temp
Ve_cel1 := V_cel1 - Vdc/max_level
Ve_cel2 := V_cel2 - Vdc/max_level
Ve_cel3 := V_cel3 - Vdc/max_level
Ve_cel4 := V_cel4 - Vdc/max_level
Ve_cel5 := V_cel5 - Vdc/max_level
TRUE
a1 := abs(Ve_cel1)
a2 := abs(Ve_cel2)
a3 := abs(Ve_cel3)
a4 := abs(Ve_cel4)
a5 := abs(Ve_cel5)
TRUE
SET := sort_Vd := 1
SET := cell1_c := 0
SET := cell2_c := 0
SET := cell3_c := 0
SET := cell4_c := 0
SET := cell5_c := 0
TRUE

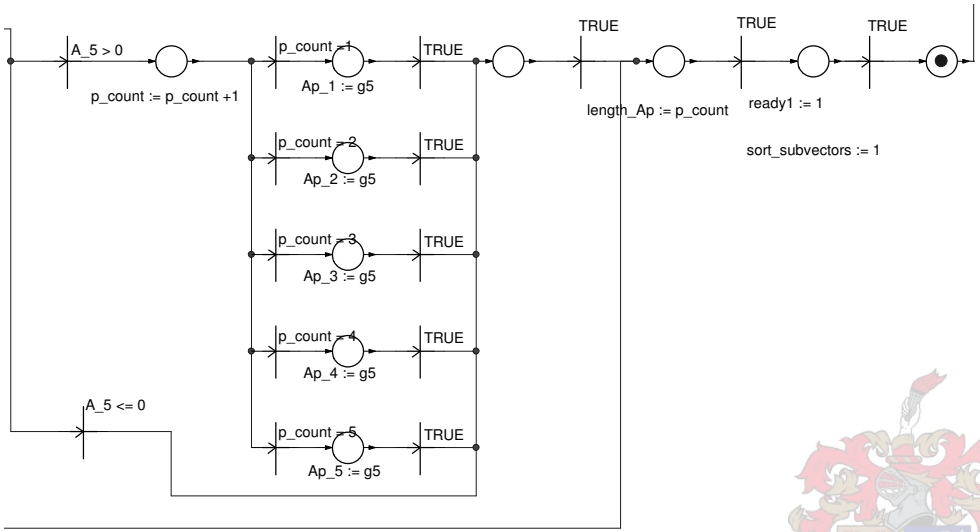

Sheet_sort_Vd
a1 := a1
a2 := a2
a3 := a3
a4 := a4
a5 := a5
g1 := g1
g2 := g2
g3 := g3
g4 := g4
g5 := g5
lo_sign := lo_sign
Input
output
State_out1
TRUE
TRUE
TRUE
TRUE
TRUE
sort_Vd = 1
TRUE
A_set := 1
State_in1
nfree := 5
nfree := nfree*(nfree <= max_level)
SET := sort_Vd := 0

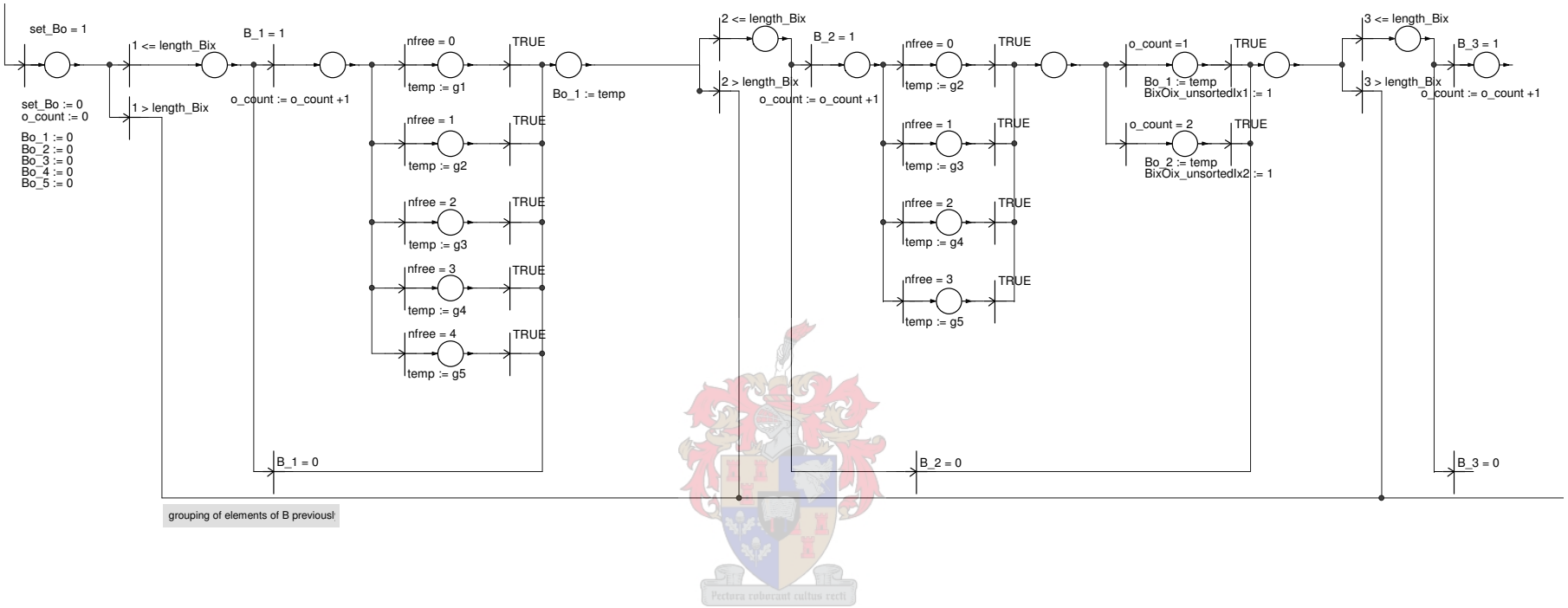

output current negative: sort in ascending order
lo_sign = -1
lo_sign=1
output current positive: sort in descending order
input
P
IN
State_in1
TRUE
SET := temp := 5*((a5>a4)and(a5>a3)and(a5>a2)and(a5>a1)) +4*((a4>a5)and(a4>a3)and(a4>a2)and(a4>a1)) + 3*((a3>a5)and(a3>a4)and(a3>a2)and(a3>a1)) + 2*((a2>a5)and(a2>a4)and(a2>a3)and(a2>a1)) + ((a1>a5)and(a1>a4)and(a1>a3)and(a1>a2))
a1 := a1*(temp<>1) - (temp=1)
a2 := a2*(temp<>2) - (temp=2)
a3 := a3*(temp<>3) - (temp=3)
a4 := a4*(temp<>4) - (temp=4)
a5 := a5*(temp<>5) - (temp=5)
g1 :=temp
calculate greatest
temp := 5*((a5>a4)and(a5>a3)and(a5>a2)and(a5>a1)) +4*((a4>a5)and(a4>a3)and(a4>a2)and(a4>a1)) + 3*((a3>a5)and(a3>a4)and(a3>a2)and(a3>a1)) + 2*((a2>a5)and(a2>a4)and(a2>a3)and(a2>a1)) + ((a1>a5)and(a1>a4)and(a1>a3)and(a1>a2))
g2 := temp
calculate 2nd greatest
g3 := temp
calculate 3rd greatest
g4 := temp
calculate 4th greatest
g5 := temp
calculate smallest
output
State_out1

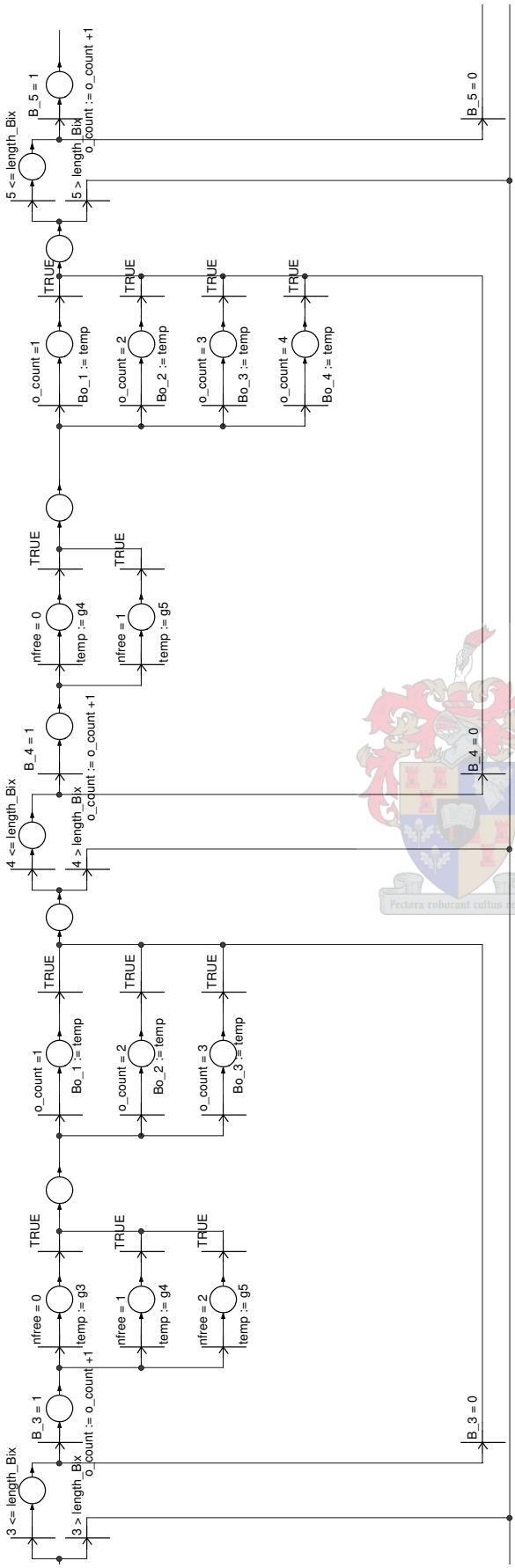

TRUE
SET := temp := 5*((a5>a4)and(a5>a3)and(a5>a2)and(a5>a1)) +4*((a4>a5)and(a4>a3)and(a4>a2)and(a4>a1)) + 3*((a3>a5)and(a3>a4)and(a3>a2)and(a3>a1)) + 2*((a2>a5)and(a2>a4)and(a2>a3)and(a2>a1)) + ((a1>a5)and(a1>a4)and(a1>a3)and(a1>a2))
TRUE
a1 := a1*(temp<>1) - (temp=1)
a2 := a2*(temp<>2) - (temp=2)
a3 := a3*(temp<>3) - (temp=3)
a4 := a4*(temp<>4) - (temp=4)
a5 := a5*(temp<>5) - (temp=5)
g5 :=temp
calculate greatest, but store in (smallest index) g5
TRUE
temp := 5*((a5>a4)and(a5>a3)and(a5>a2)and(a5>a1)) +4*((a4>a5)and(a4>a3)and(a4>a2)and(a4>a1)) + 3*((a3>a5)and(a3>a4)and(a3>a2)and(a3>a1)) + 2*((a2>a5)and(a2>a4)and(a2>a3)and(a2>a1)) + ((a1>a5)and(a1>a4)and(a1>a3)and(a1>a2))
TRUE
a1 := a1*(temp<>1) - (temp=1)
a2 := a2*(temp<>2) - (temp=2)
a3 := a3*(temp<>3) - (temp=3)
a4 := a4*(temp<>4) - (temp=4)
a5 := a5*(temp<>5) - (temp=5)
g4 := temp
calculate 2nd largest value, bur store in (2nd smallest index) g4
TRUE
temp := 5*((a5>a4)and(a5>a3)and(a5>a2)and(a5>a1)) +4*((a4>a5)and(a4>a3)and(a4>a2)and(a4>a1)) + 3*((a3>a5)and(a3>a4)and(a3>a2)and(a3>a1)) + 2*((a2>a5)and(a2>a4)and(a2>a3)and(a2>a1)) + ((a1>a5)and(a1>a4)and(a1>a3)and(a1>a2))
TRUE
a1 := a1*(temp<>1) - (temp=1)
a2 := a2*(temp<>2) - (temp=2)
a3 := a3*(temp<>3) - (temp=3)
a4 := a4*(temp<>4) - (temp=4)
a5 := a5*(temp<>5) - (temp=5)
g3 := temp
store 3rd largest value, but store in g3
TRUE
temp := 5*((a5>a4)and(a5>a3)and(a5>a2)and(a5>a1)) +4*((a4>a5)and(a4>a3)and(a4>a2)and(a4>a1)) + 3*((a3>a5)and(a3>a4)and(a3>a2)and(a3>a1)) + 2*((a2>a5)and(a2>a4)and(a2>a3)and(a2>a1)) + ((a1>a5)and(a1>a4)and(a1>a3)and(a1>a2))
TRUE
a1 := a1*(temp<>1) - (temp=1)
a2 := a2*(temp<>2) - (temp=2)
a3 := a3*(temp<>3) - (temp=3)
a4 := a4*(temp<>4) - (temp=4)
a5 := a5*(temp<>5) - (temp=5)
g2 := temp
calculate 4th largest value, but store in g2
TRUE
temp := 5*((a5>a4)and(a5>a3)and(a5>a2)and(a5>a1)) +4*((a4>a5)and(a4>a3)and(a4>a2)and(a4>a1)) + 3*((a3>a5)and(a3>a4)and(a3>a2)and(a3>a1)) + 2*((a2>a5)and(a2>a4)and(a2>a3)and(a2>a1)) + ((a1>a5)and(a1>a4)and(a1>a3)and(a1>a2))
TRUE
TRUE
a1 := a1*(temp<>1) - (temp=1)
a2 := a2*(temp<>2) - (temp=2)
a3 := a3*(temp<>3) - (temp=3)
a4 := a4*(temp<>4) - (temp=4)
a5 := a5*(temp<>5) - (temp=5)
g1 := temp
calculate smallest value, but store in g1

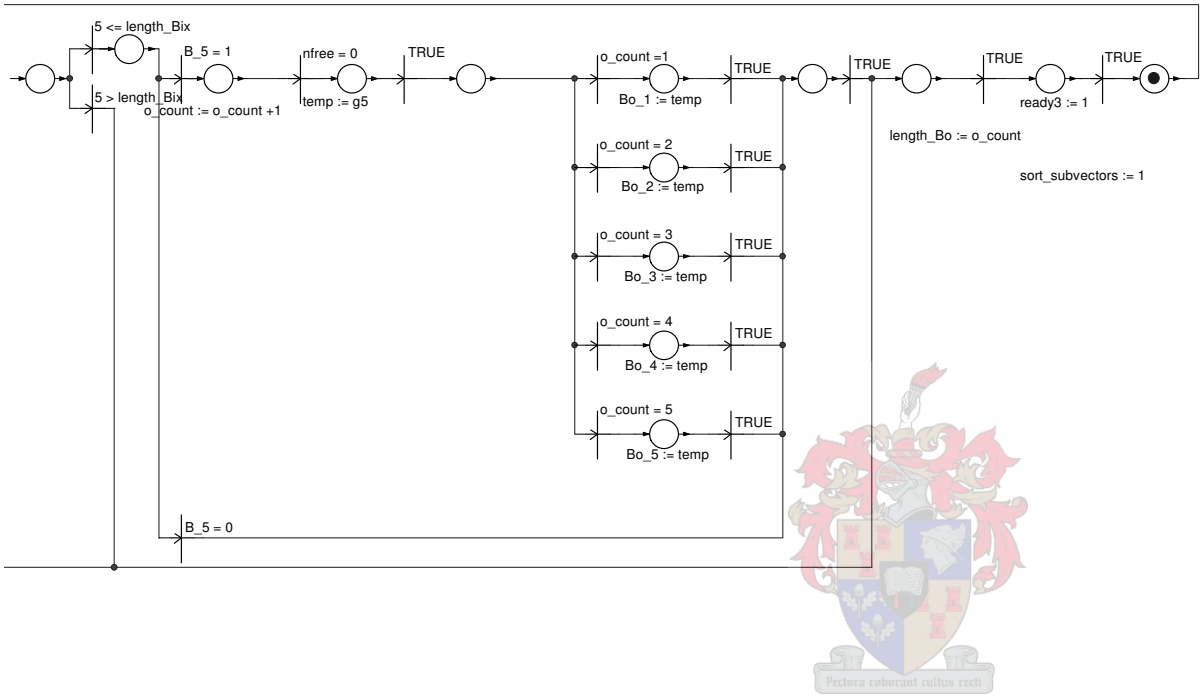


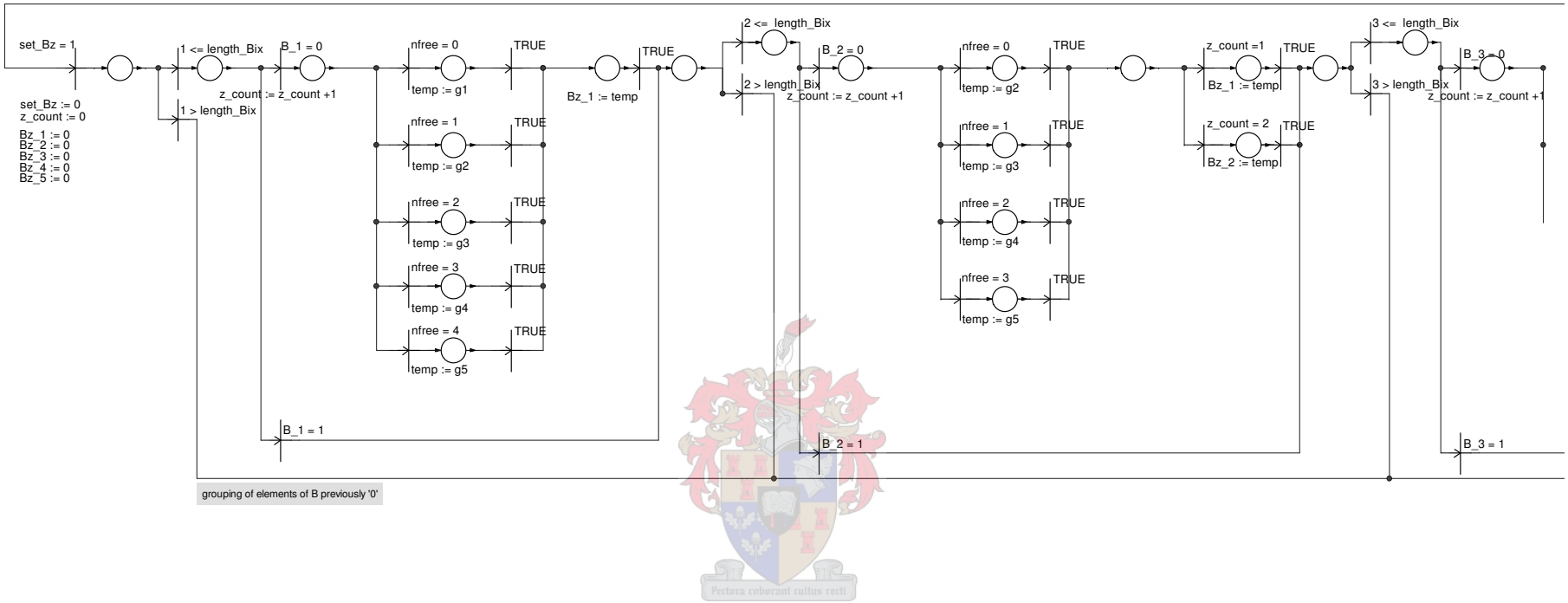

set_Ap = 1
1 <= length_A
A_1 > 0
p_count =1
TRUE
2 <= length_A
A_2 > 0
p_count =1
TRUE
3 <= length_A
p_count := p_count +1
Ap_1 := g1
2 > length_A
p_count := p_count +1
Ap_1 := g2
3 > length_A
set_Ap := 0
p_count := 0
1 > length_A
p_count = 2
TRUE
Ap_1 := 0
Ap_2 := 0
Ap_3 := 0
Ap_4 := 0
Ap_5 := 0
Ap_2 := g2
A_1 <= 0
A_2 <= 0
selection of positive elements in A
3 <= length_A
A_3 > 0
p_count =1
TRUE
4 <= length_A
A_4 > 0
p_count =1
TRUE
5 <= length_A
A_5 > 0
3 > length_A
p_count := p_count +1
Ap_1 := g3
4 > length_A
p_count := p_count +1
Ap_1 := g4
5 > length_A
p_count := p_count +1
p_count = 2
TRUE
p_count = 2
TRUE
Ap_2 := g3
Ap_2 := g4
p_count = 3
TRUE
p_count = 3
TRUE
Ap_3 := g3
Ap_3 := g4
A_3 <= 0
p_count = 4
TRUE
Ap_4 := g4
A_4 <= 0
A_5 <= 0

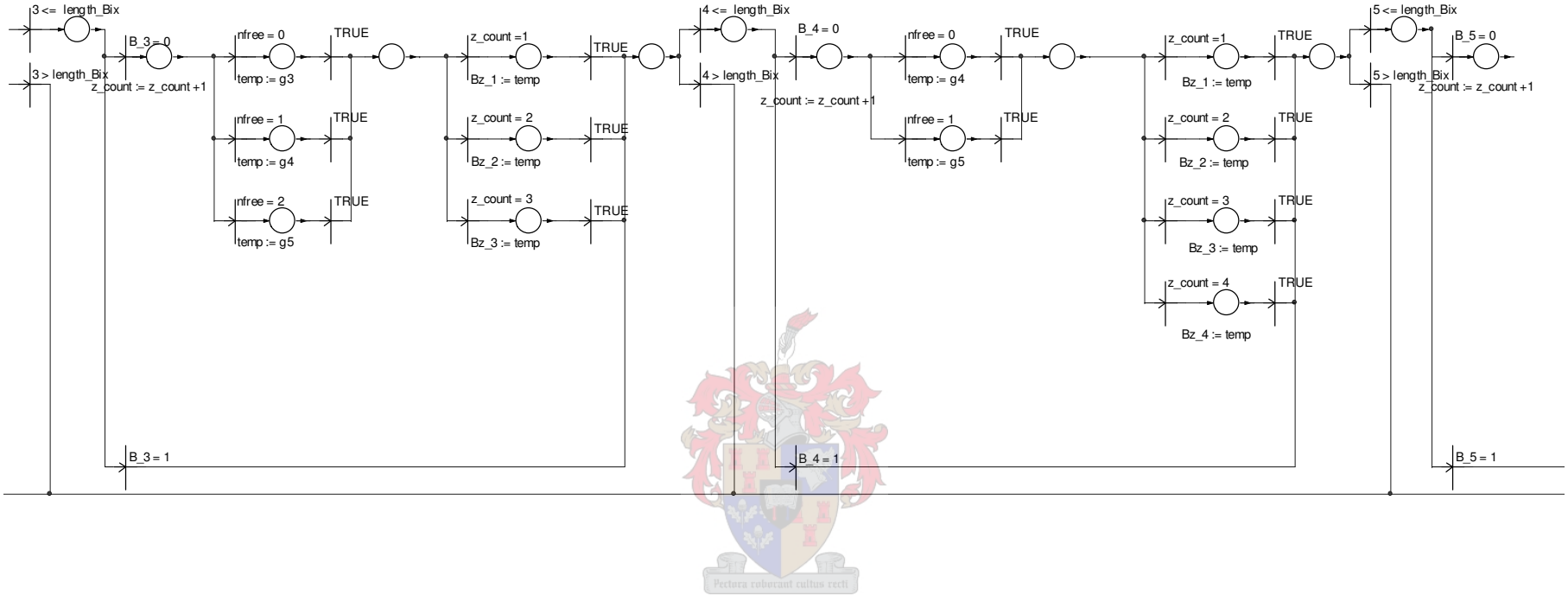

A_5 > 0
p_count := p_count +1
p_count = 1
Ap_1 := g5
TRUE
p_count = 2
Ap_2 := g5
TRUE
p_count = 3
Ap_3 := g5
TRUE
p_count = 4
Ap_4 := g5
TRUE
A_5 <= 0
p_count = 5
Ap_5 := g5
TRUE
TRUE
length_Ap := p_count
TRUE
ready1 := 1
sort_subvectors := 1
TRUE

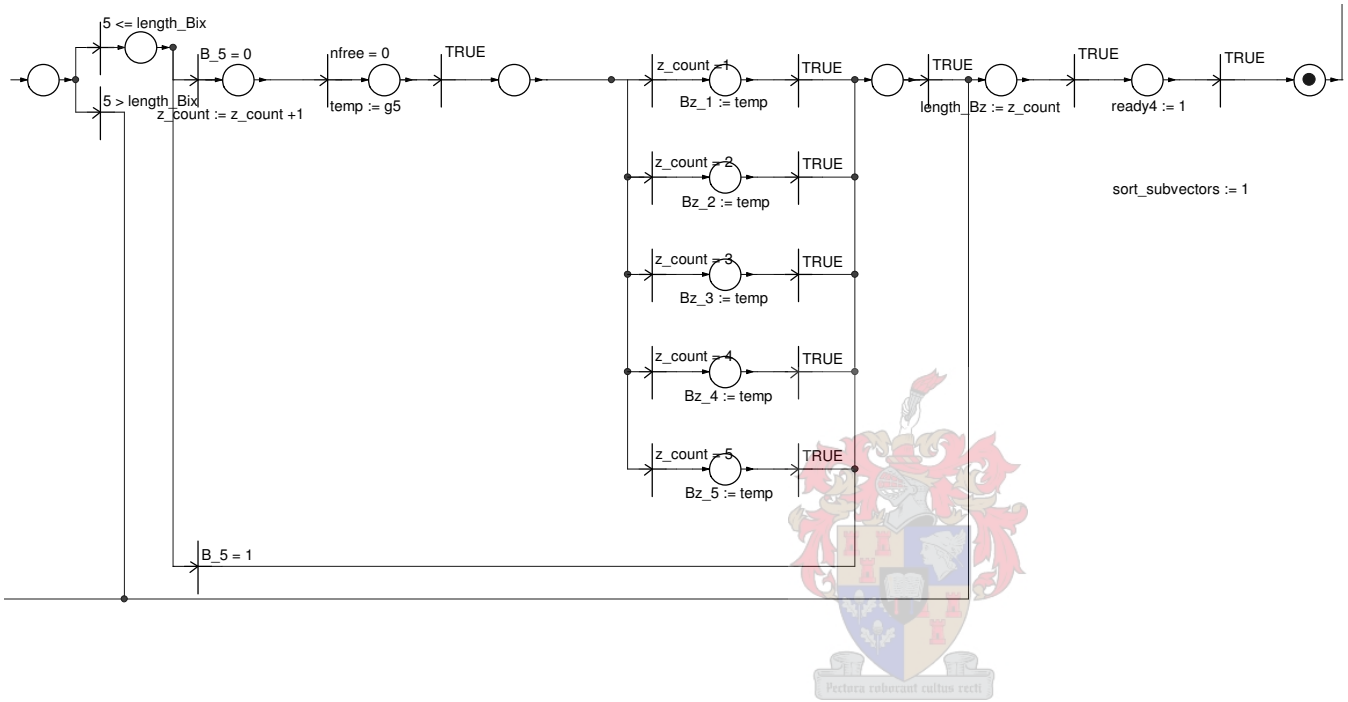


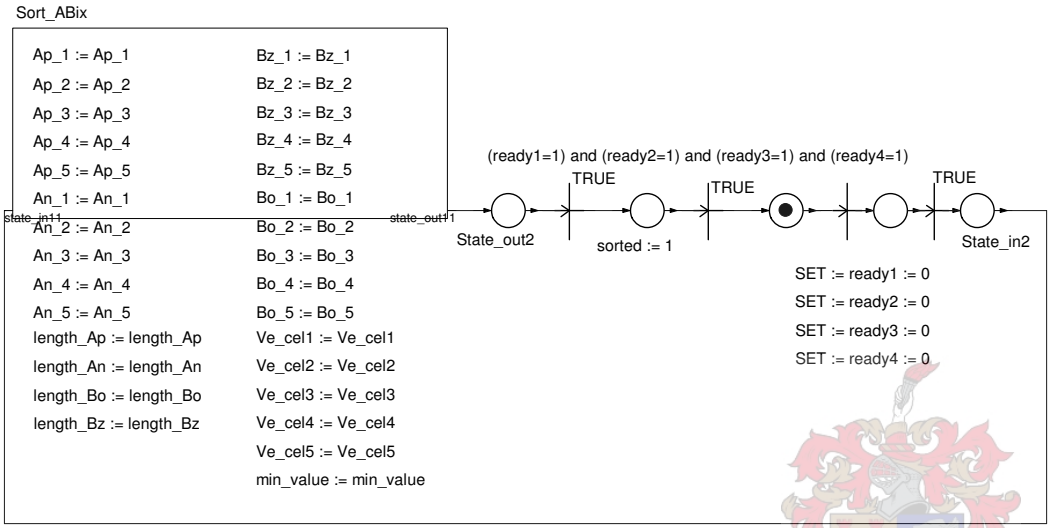

3 <= length_Bix
3 > length_Bix
B_3 = 1
o_count := o_count +1
B_3 = 0
nfree = 0
temp := g3
nfree = 1
temp := g4
nfree = 2
temp := g5
TRUE
o_count =1
Bo_1 := temp
o_count = 2
Bo_2 := temp
o_count = 3
Bo_3 := temp
4 <= length_Bix
4 > length_Bix
B_4 = 1
o_count := o_count +1
B_4 = 0
nfree = 0
temp := g4
nfree = 1
temp := g5
TRUE
o_count =1
Bo_1 := temp
o_count = 2
Bo_2 := temp
o_count = 3
Bo_3 := temp
o_count = 4
Bo_4 := temp
5 <= length_Bix
5 > length_Bix
B_5 = 1
o_count := o_count +1
B_5 = 0

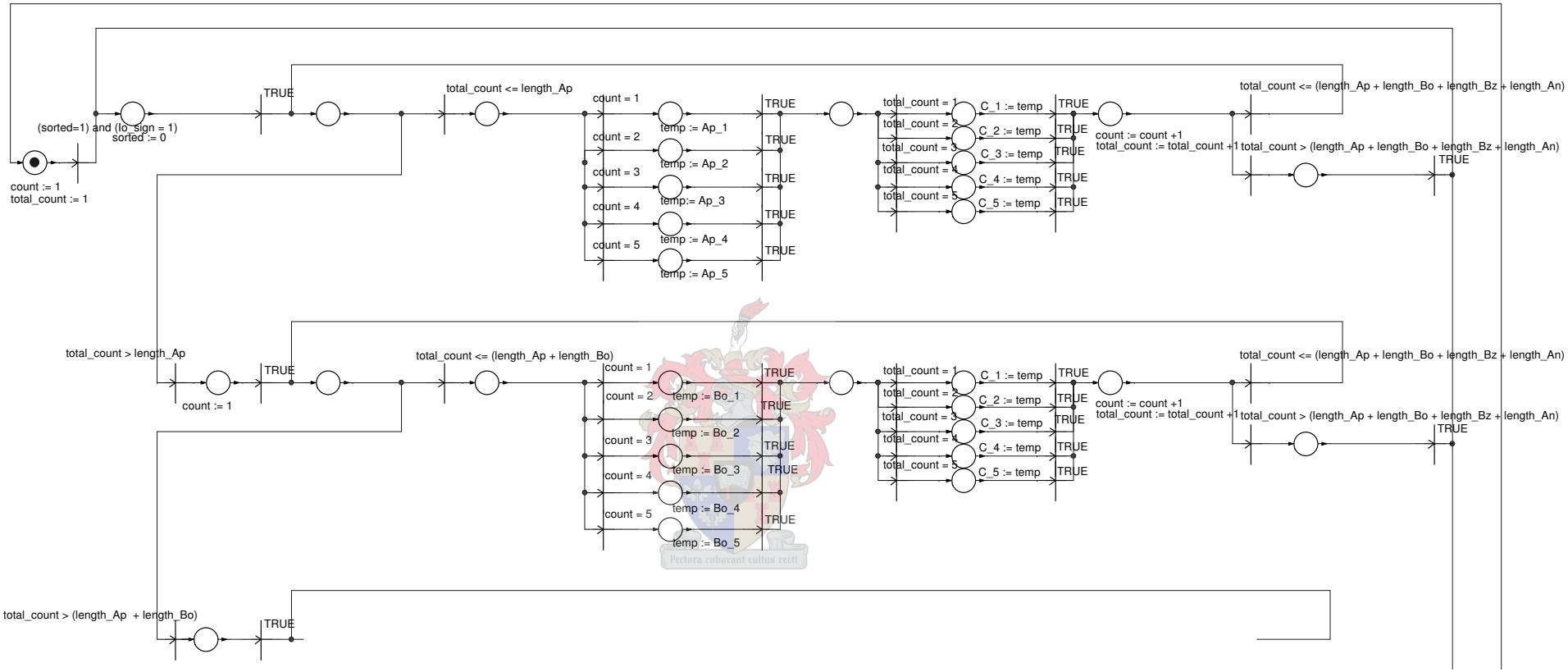

5 <= length_Bix
5 > length_Bix
B_5 = 1
o_count := o_count +1
nfree = 0
temp := g5
TRUE
o_count =1
Bo_1 := temp
TRUE
o_count = 2
Bo_2 := temp
TRUE
o_count = 3
Bo_3 := temp
TRUE
o_count = 4
Bo_4 := temp
TRUE
o_count = 5
Bo_5 := temp
TRUE
TRUE
length_Bo := o_count
TRUE
ready3 := 1
TRUE
sort_subvectors := 1
B_5 = 0

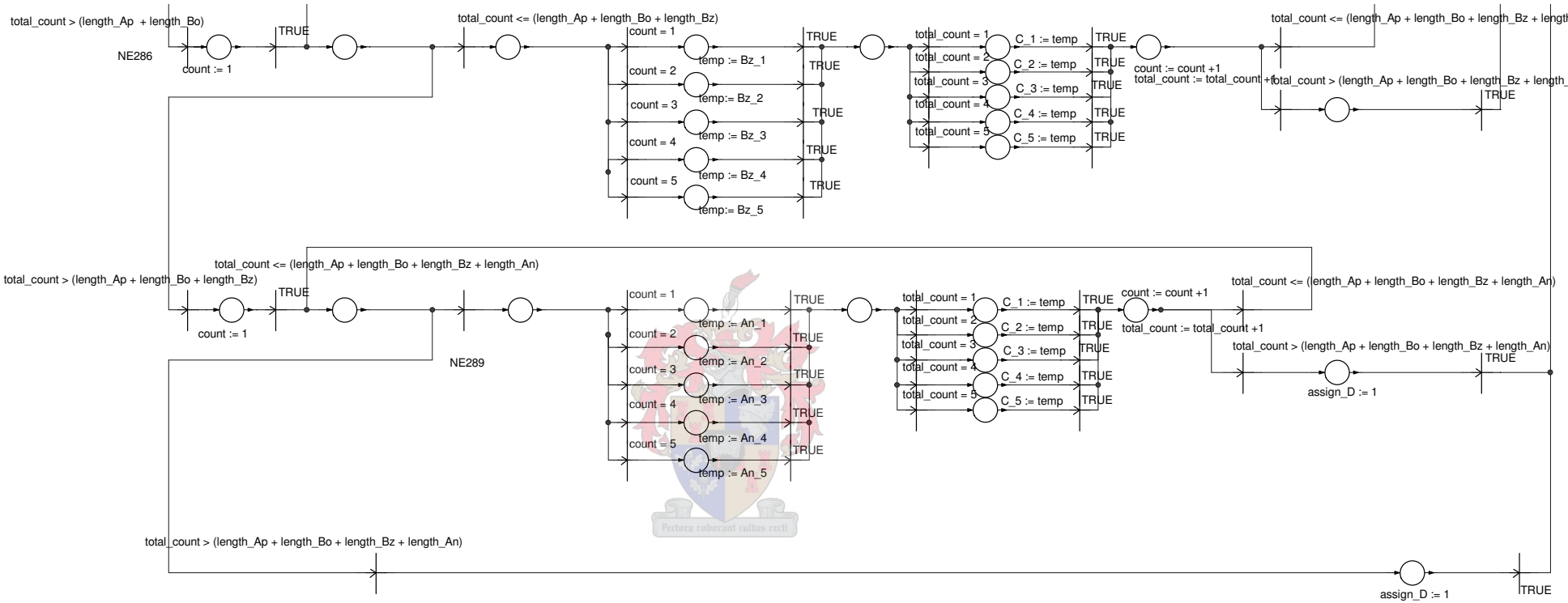


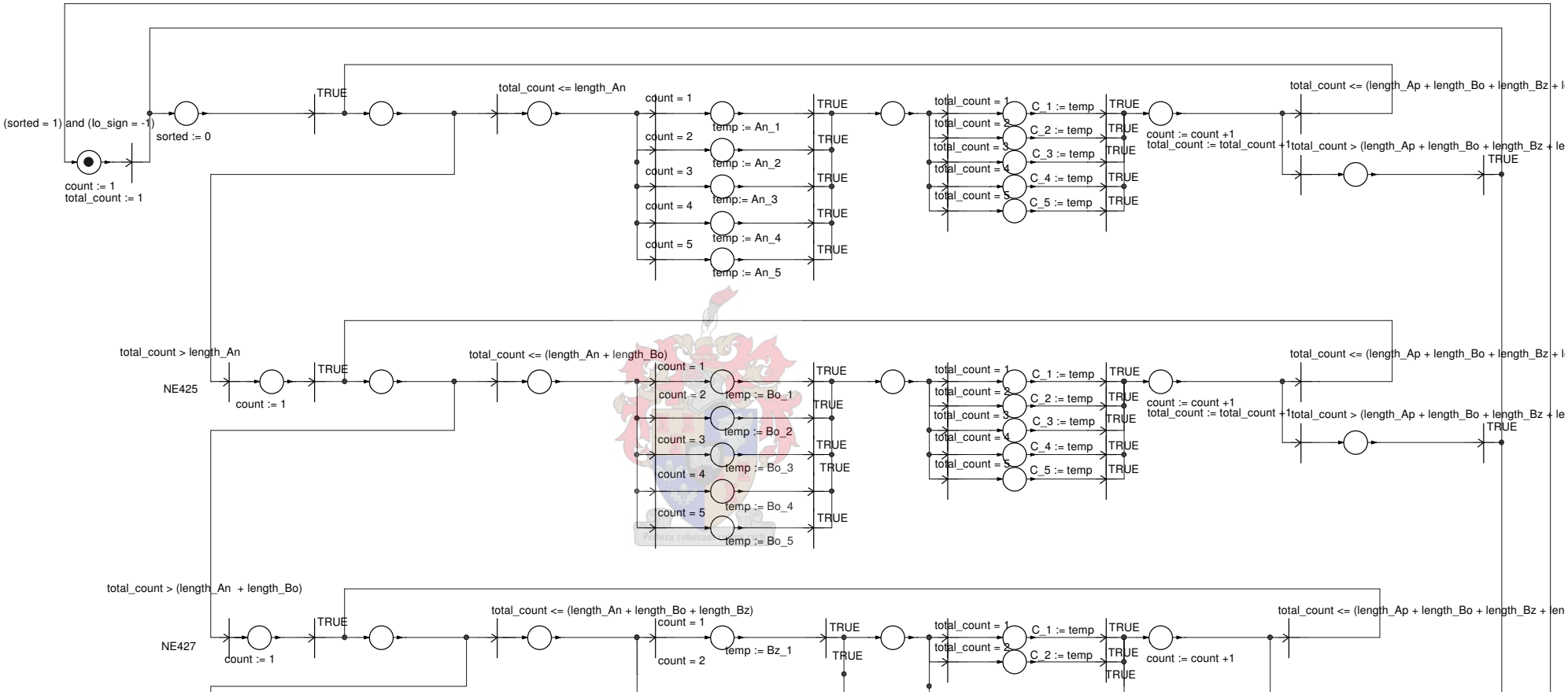


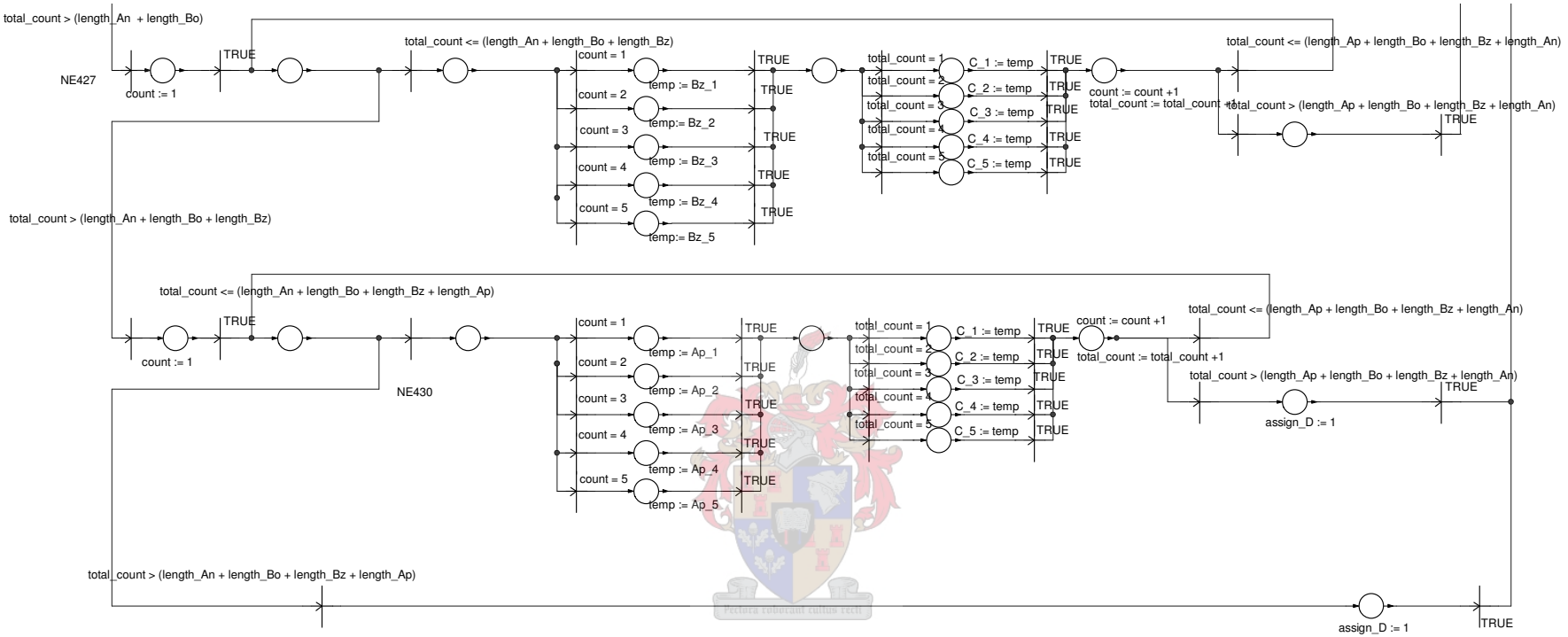

5 <= length_Bix
5 > length_Bix
B_5 = 0
z_count := z_count +1
nfree = 0
temp := g5
TRUE
z_count = 1
Bz_1 := temp
TRUE
z_count = 2
Bz_2 := temp
TRUE
z_count = 3
Bz_3 := temp
TRUE
z_count = 4
Bz_4 := temp
TRUE
z_count = 5
Bz_5 := temp
TRUE
B_5 = 1
TRUE
length_Bz := z_count
TRUE
ready4 := 1
TRUE
sort_subvectors := 1

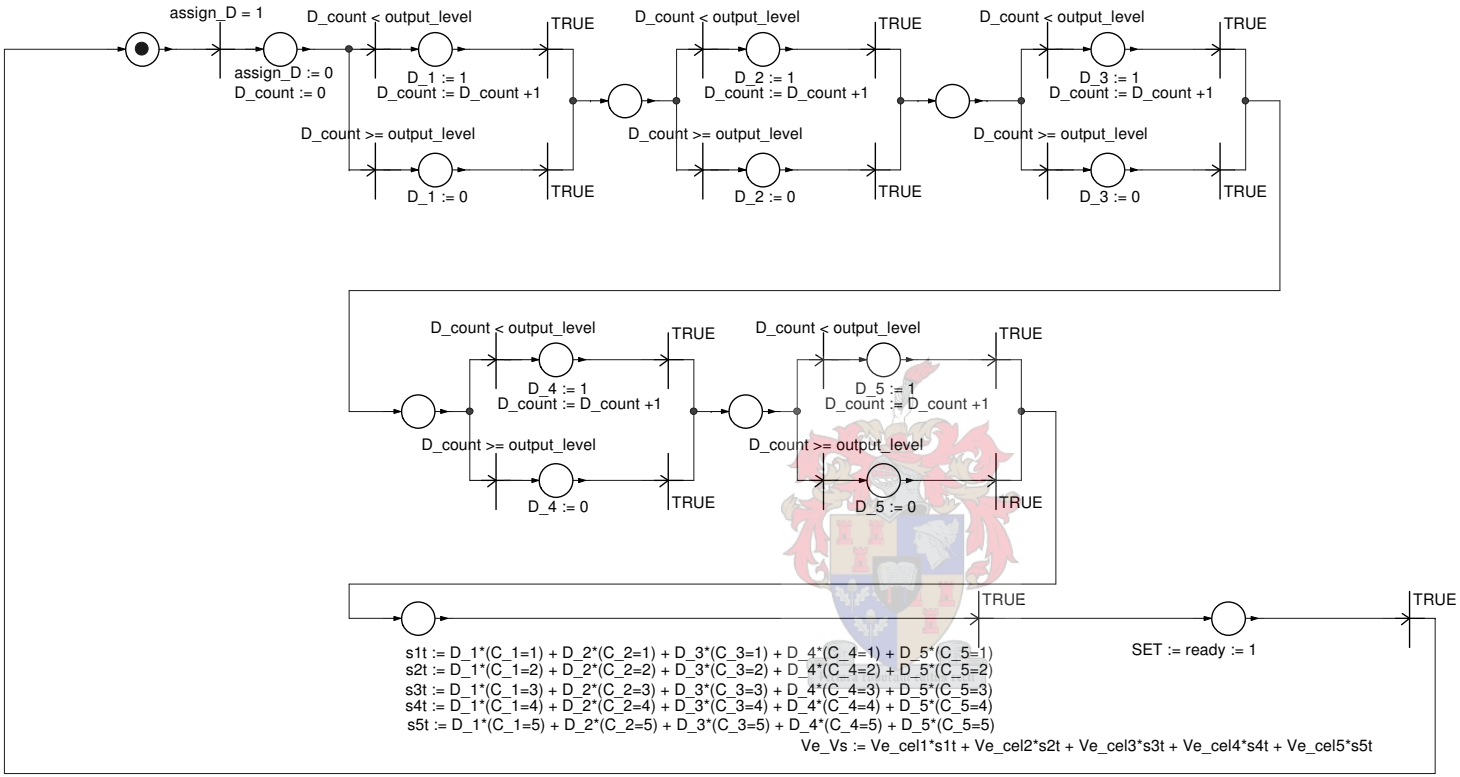

Sort_ABix
Ap_1 := Ap_1
Ap_2 := Ap_2
Ap_3 := Ap_3
Ap_4 := Ap_4
Ap_5 := Ap_5
An_1 := An_1
An_2 := An_2
An_3 := An_3
An_4 := An_4
An_5 := An_5
length_Ap := length_Ap
length_An := length_An
length_Bo := length_Bo
length_Bz := length_Bz
Bz_1 := Bz_1
Bz_2 := Bz_2
Bz_3 := Bz_3
Bz_4 := Bz_4
Bz_5 := Bz_5
Bo_1 := Bo_1
Bo_2 := Bo_2
Bo_3 := Bo_3
Bo_4 := Bo_4
Bo_5 := Bo_5
Ve_cel1 := Ve_cel1
Ve_cel2 := Ve_cel2
Ve_cel3 := Ve_cel3
Ve_cel4 := Ve_cel4
Ve_cel5 := Ve_cel5
min_value := min_value
state_in11
state_out11
State_out2
(ready1=1) and (ready2=1) and (ready3=1) and (ready4=1)
TRUE
sorted := 1
TRUE
TRUE
State_in2
SET := ready1 := 0
SET := ready2 := 0
SET := ready3 := 0
SET := ready4 := 0


count := 1
total_count := 1
(sorted=1) and (lo_sign = 1)
sorted := 0
TRUE
total_count <= length_Ap
count = 1
count = 2
count = 3
count = 4
count = 5
temp := Ap_1
temp := Ap_2
temp:= Ap_3
temp := Ap_4
temp := Ap_5
total_count = 1
total_count = 2
total_count = 3
total_count = 4
total_count = 5
C_1 := temp
C_2 := temp
C_3 := temp
C_4 := temp
C_5 := temp
count := count +1
total_count := total_count +1
total_count <= (length_Ap + length_Bo + length_Bz + length_An)
total_count > (length_Ap + length_Bo + length_Bz + length_An)
total_count > length_Ap
count := 1
total_count <= (length_Ap + length_Bo)
temp := Bo_1
temp := Bo_2
temp := Bo_3
temp := Bo_4
temp := Bo_5
total_count > (length_Ap + length_Bo)

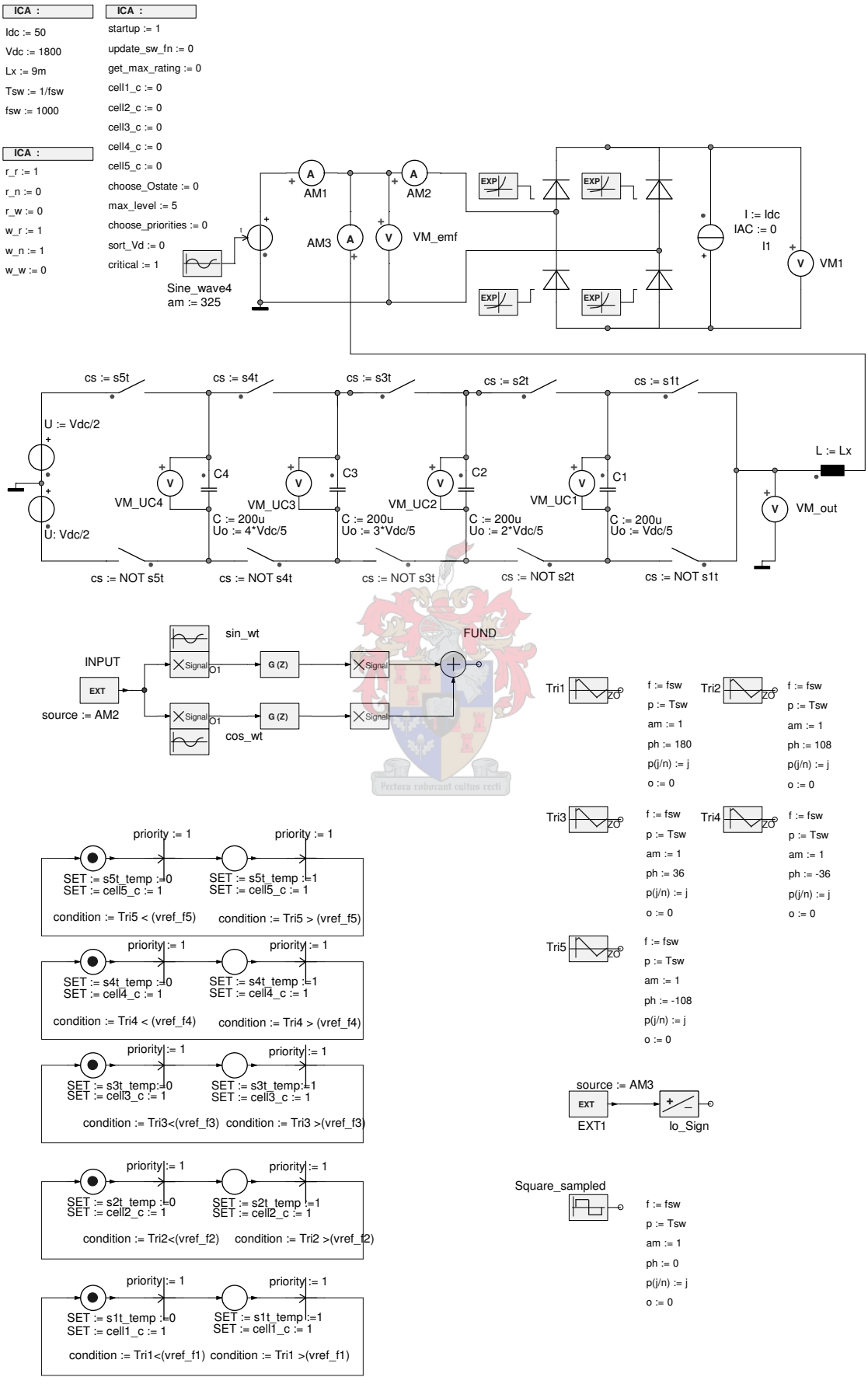

total_count > (length_Ap + length_Bo)
NE286
count := 1
TRUE
total_count <= (length_Ap + length_Bo + length_Bz)
count = 1
temp := Bz_1
count = 2
temp:= Bz_2
count = 3
temp := Bz_3
count = 4
temp := Bz_4
count = 5
temp:= Bz_5
total_count = 1
C_1 := temp
total_count = 2
C_2 := temp
total_count = 3
C_3 := temp
total_count = 4
C_4 := temp
total_count = 5
C_5 := temp
count := count +1
total_count := total_count +1
total_count <= (length_Ap + length_Bo + length_Bz + length_An)
total_count > (length_Ap + length_Bo + length_Bz + length_An)
total_count > (length_Ap + length_Bo + length_Bz)
NE289
temp := An_1
temp := An_2
temp := An_3
temp := An_4
temp := An_5
assign_D := 1


(sorted = 1) and (lo_sign = -1)
count := 1
total_count := 1
sorted := 0
TRUE
total_count <= length_An
count = 1
count = 2
count = 3
count = 4
count = 5
temp := An_1
temp := An_2
temp:= An_3
temp := An_4
temp := An_5
total_count = 1
total_count = 2
total_count = 3
total_count = 4
total_count = 5
C_1 := temp
C_2 := temp
C_3 := temp
C_4 := temp
C_5 := temp
count := count +1
total_count := total_count +1
total_count <= (length_Ap + length_Bo + length_Bz + l
total_count > (length_Ap + length_Bo + length_Bz + le
total_count > length_An
NE425
count := 1
total_count <= (length_An + length_Bo)
temp := Bo_1
temp := Bo_2
temp := Bo_3
temp := Bo_4
temp := Bo_5
total_count > (length_An + length_Bo)
NE427
total_count <= (length_An + length_Bo + length_Bz)
temp := Bz_1
total_count <= (length_Ap + length_Bo + length_Bz + len


total_count > (length_An + length_Bo)
NE427
TRUE
count := 1
total_count <= (length_An + length_Bo + length_Bz)
count = 1
temp := Bz_1
count = 2
temp:= Bz_2
count = 3
temp := Bz_3
count = 4
temp := Bz_4
count = 5
temp:= Bz_5
total_count = 1
C_1 := temp
total_count = 2
C_2 := temp
total_count = 3
C_3 := temp
total_count = 4
C_4 := temp
total_count = 5
C_5 := temp
count := count +1
total_count := total_count +1
total_count <= (length_Ap + length_Bo + length_Bz + length_An)
total_count > (length_Ap + length_Bo + length_Bz + length_An)
total_count > (length_An + length_Bo + length_Bz)
total_count <= (length_An + length_Bo + length_Bz + length_Ap)
NE430
temp := Ap_1
temp := Ap_2
temp := Ap_3
temp := Ap_4
temp := Ap_5
assign_D := 1
total_count > (length_An + length_Bo + length_Bz + length_Ap)


assign_D = 1
assign_D := 0
D_count := 0
D_count < output_level
D_1 := 1
D_count := D_count +1
TRUE
D_count >= output_level
D_1 := 0
D_2 := 1
D_2 := 0
D_3 := 1
D_3 := 0
D_4 := 1
D_4 := 0
D_5 := 1
D_5 := 0
SET := ready := 1
s1t := D_1*(C_1=1) + D_2*(C_2=1) + D_3*(C_3=1) + D_4*(C_4=1) + D_5*(C_5=1)
s2t := D_1*(C_1=2) + D_2*(C_2=2) + D_3*(C_3=2) + D_4*(C_4=2) + D_5*(C_5=2)
s3t := D_1*(C_1=3) + D_2*(C_2=3) + D_3*(C_3=3) + D_4*(C_4=3) + D_5*(C_5=3)
s4t := D_1*(C_1=4) + D_2*(C_2=4) + D_3*(C_3=4) + D_4*(C_4=4) + D_5*(C_5=4)
s5t := D_1*(C_1=5) + D_2*(C_2=5) + D_3*(C_3=5) + D_4*(C_4=5) + D_5*(C_5=5)
Ve_Vs := Ve_cel1*s1t + Ve_cel2*s2t + Ve_cel3*s3t + Ve_cel4*s4t + Ve_cel5*s5t

PER-PHASE MODEL – CAPACITOR-VOLTAGE BASED ALGORITHM


ICA :
Idc := 50
Vdc := 1800
Lx := 9m
Tsw := 1/fsw
fsw := 1000
ICA :
r_r := 1
r_n := 0
r_w := 0
w_r := 1
w_n := 1
w_w := 0
ICA :
startup := 1
update_sw_fn := 0
get_max_rating := 0
cell1_c := 0
cell2_c := 0
cell3_c := 0
cell4_c := 0
cell5_c := 0
choose_Ostate := 0
max_level := 5
choose_priorities := 0
sort_Vd := 0
critical := 1
Sine_wave4
am := 325
AM1
AM2
AM3
VM_emf
EXP
I := Idc
IAC := 0
I1
VM1
cs := s5t
cs := s4t
cs := s3t
cs := s2t
cs := s1t
U := Vdc/2
U: Vdc/2
C4
C3
C2
C1
VM_UC4
VM_UC3
VM_UC2
VM_UC1
C := 200u
Uo := 4*Vdc/5
Uo := 3*Vdc/5
Uo := 2*Vdc/5
Uo := Vdc/5
L := Lx
VM_out
cs := NOT s5t
cs := NOT s4t
cs := NOT s3t
cs := NOT s2t
cs := NOT s1t
INPUT
EXT
source := AM2
Signal
G (Z)
sin_wt
cos_wt
FUND
Tri1
Tri2
Tri3
Tri4
Tri5
f := fsw
p := Tsw
am := 1
ph := 180
ph := 108
ph := 36
ph := -108
p(j/n) := j
o := 0
priority := 1
SET := s5t_temp :=0
SET := s5t_temp :=1
SET := cell5_c := 1
condition := Tri5 < (vref_f5)
condition := Tri5 > (vref_f5)
SET := s4t_temp :=0
SET := s4t_temp :=1
SET := cell4_c := 1
condition := Tri4 < (vref_f4)
condition := Tri4 > (vref_f4)
SET := s3t_temp:=0
SET := s3t_temp:=1
SET := cell3_c := 1
condition := Tri3<(vref_f3)
condition := Tri3 >(vref_f3)
SET := s2t_temp :=0
SET := s2t_temp :=1
SET := cell2_c := 1
condition := Tri2<(vref_f2)
condition := Tri2 >(vref_f2)
SET := s1t_temp :=0
SET := s1t_temp :=1
SET := cell1_c := 1
condition := Tri1<(vref_f1)
condition := Tri1 >(vref_f1)
source := AM3
EXT1
Io_Sign
Square_sampled

ICA :	ICA :	ICA :	ICA :	ICA :
A5 := 1	A4 := 1	A3 := 1	A2 := 1	A1 := 0
B5 := 1	B4 := 1	B3 := 1	B2 := 0	B1 := 1
C5 := 1	C4 := 1	C3 := 0	C2 := 1	C1 := 1
D5 := 1	D4 := 0	D3 := 1	D2 := 1	D1 := 1
E5 := 0	E4 := 1	E3 := 1	E2 := 1	E1 := 1
F5 := 1	F4 := 1	F3 := 1	F2 := 0	F1 := 0
G5 := 1	G4 := 1	G3 := 0	G2 := 1	G1 := 0
H5 := 1	H4 := 1	H3 := 0	H2 := 0	H1 := 1
I5 := 1	I4 := 0	I3 := 1	I2 := 1	I1 := 0
J5 := 1	J4 := 0	J3 := 1	J2 := 0	J1 := 1
K5 := 1	K4 := 0	K3 := 0	K2 := 1	K1 := 1
L5 := 0	L4 := 1	L3 := 1	L2 := 1	L1 := 0
M5 := 0	M4 := 1	M3 := 1	M2 := 0	M1 := 1
Q5 := 0	Q4 := 1	Q3 := 0	Q2 := 1	Q1 := 1
R5 := 0	R4 := 0	R3 := 1	R2 := 1	R1 := 1
S5 := 1	S4 := 1	S3 := 0	S2 := 0	S1 := 0
T5 := 1	T4 := 0	T3 := 1	T2 := 0	T1 := 0
U5 := 1	U4 := 0	U3 := 0	U2 := 1	U1 := 0
V5 := 1	V4 := 0	V3 := 0	V2 := 0	V1 := 1
W5 := 0	W4 := 1	W3 := 1	W2 := 0	W1 := 0
X5 := 0	X4 := 1	X3 := 0	X2 := 1	X1 := 0
Y5 := 0	Y4 := 1	Y3 := 0	Y2 := 0	Y1 := 1
Z5 := 0	Z4 := 0	Z3 := 1	Z2 := 1	Z1 := 0
AA5 := 0	AA4 := 0	AA3 := 1	AA2 := 0	AA1 := 1
AB5 := 0	AB4 := 0	AB3 := 0	AB2 := 1	AB1 := 1
AC5 := 1	AC4 := 0	AC3 := 0	AC2 := 0	AC1 := 0
AD5 := 0	AD4 := 1	AD3 := 0	AD2 := 0	AD1 := 0
AE5 := 0	AE4 := 0	AE3 := 1	AE2 := 0	AE1 := 0
AF5 := 0	AF4 := 0	AF3 := 0	AF2 := 1	AF1 := 0
AG5 := 0	AG4 := 0	AG3 := 0	AG2 := 0	AG1 := 1
P5 := 1	P4 := 1	P3 := 1	P2 := 1	P1 := 1
N5 := 0	N4 := 0	N3 := 0	N2 := 0	N1 := 0

C1_charge_ref

ICA :
A_c1_ref := 1
B_c1_ref := 0
C_c1_ref := 2
D_c1_ref := 2
E_c1_ref := 2
F_c1_ref := 2
G_c1_ref := 1
H_c1_ref := 0
I_c1_ref := 1
J_c1_ref := 0
K_c1_ref := 2
L_c1_ref := 1
M_c1_ref := 0
Q_c1_ref := 2
R_c1_ref := 2
S_c1_ref := 2
T_c1_ref := 2
U_c1_ref := 1
V_c1_ref := 0
W_c1_ref := 2
X_c1_ref := 1
Y_c1_ref := 0
Z_c1_ref := 1
AA_c1_ref := 0
AB_c1_ref := 2
AC_c1_ref := 2
AD_c1_ref := 2
AE_c1_ref := 2
AF_c1_ref := 1
AG_c1_ref := 0
P_c1_ref := 2
N_c1_ref := 2

C2_charge_ref

ICA :
A_c2_ref := 2
B_c2_ref := 1
C_c2_ref := 0
D_c2_ref := 2
E_c2_ref := 2
F_c2_ref := 1
G_c2_ref := 0
H_c2_ref := 2
I_c2_ref := 2
J_c2_ref := 1
K_c2_ref := 0
L_c2_ref := 2
M_c2_ref := 1
Q_c2_ref := 0
R_c2_ref := 2
S_c2_ref := 2
T_c2_ref := 1
U_c2_ref := 0
V_c2_ref := 2
W_c2_ref := 1
X_c2_ref := 0
Y_c2_ref := 2
Z_c2_ref := 2
AA_c2_ref := 1
AB_c2_ref := 0
AC_c2_ref := 2
AD_c2_ref := 2
AE_c2_ref := 1
AF_c2_ref := 0
AG_c2_ref := 2
P_c2_ref := 2
N_c2_ref := 2

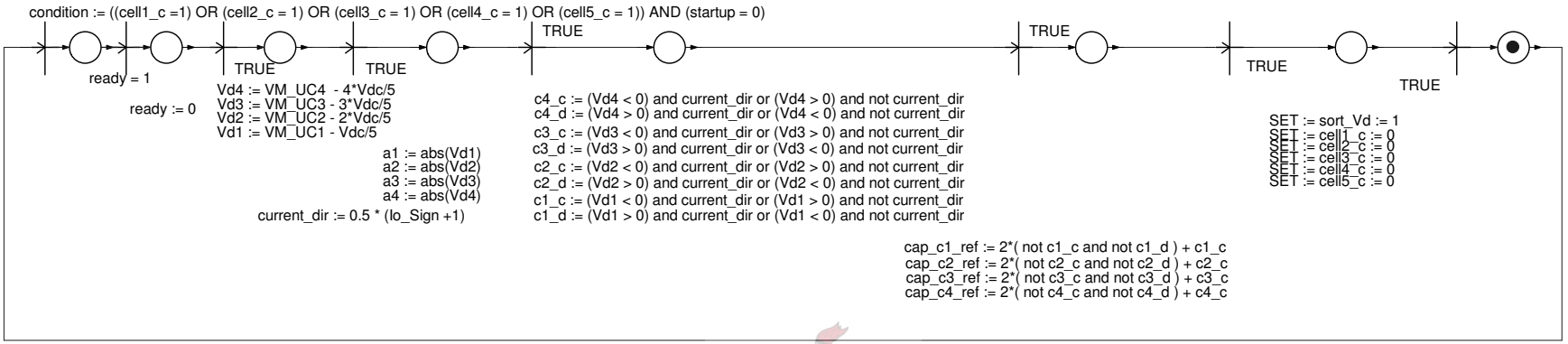
C3_charge_ref

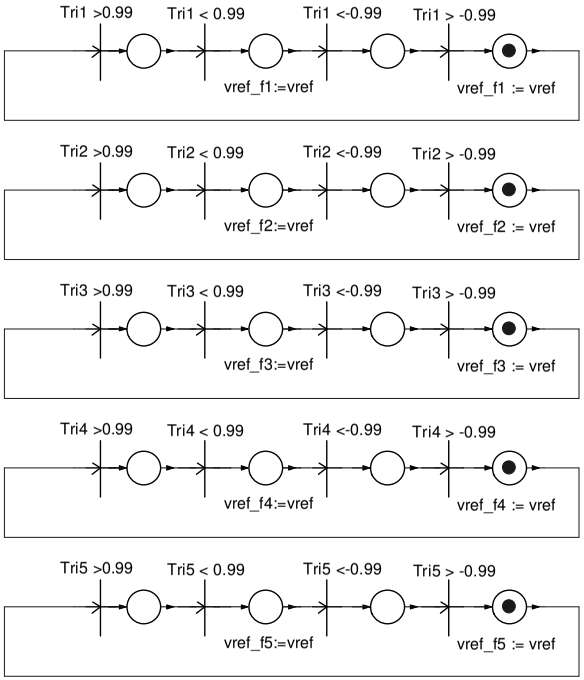
ICA :
A_c3_ref := 2
B_c3_ref := 2
C_c3_ref := 1
D_c3_ref := 0
E_c3_ref := 2
F_c3_ref := 2
G_c3_ref := 1
H_c3_ref := 1
I_c3_ref := 0
J_c3_ref := 0
K_c3_ref := 2
L_c3_ref := 2
M_c3_ref := 2
Q_c3_ref := 1
R_c3_ref := 0
S_c3_ref := 1
T_c3_ref := 0
U_c3_ref := 2
V_c3_ref := 2
W_c3_ref := 2
X_c3_ref := 1
Y_c3_ref := 1
Z_c3_ref := 0
AA_c3_ref := 0
AB_c3_ref := 2
AC_c3_ref := 2
AD_c3_ref := 1
AE_c3_ref := 0
AF_c3_ref := 2
AG_c3_ref := 2
P_c3_ref := 2
N_c3_ref := 2

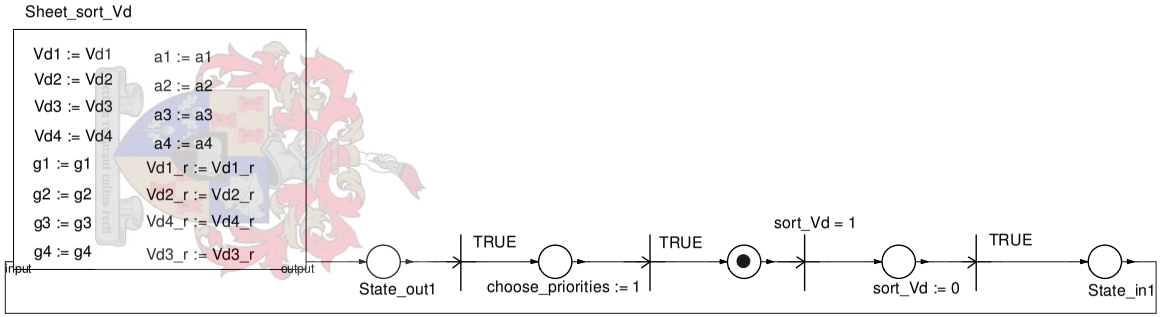
C4_charge_ref

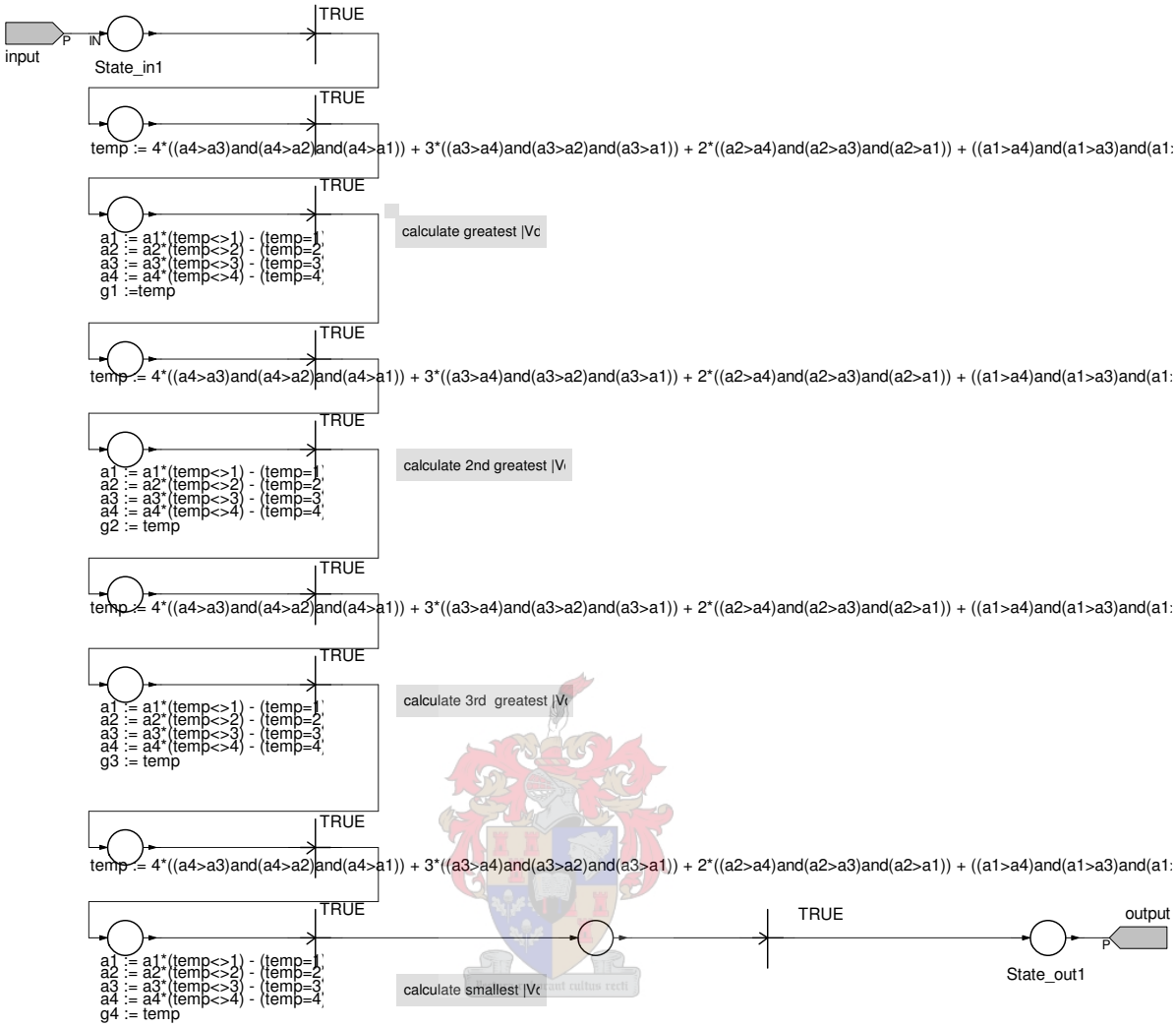
ICA :
A_c4_ref := 2
B_c4_ref := 2
C_c4_ref := 2
D_c4_ref := 1
E_c4_ref := 0
F_c4_ref := 2
G_c4_ref := 2
H_c4_ref := 2
I_c4_ref := 1
J_c4_ref := 1
K_c4_ref := 1
L_c4_ref := 0
M_c4_ref := 0
Q_c4_ref := 0
R_c4_ref := 2
S_c4_ref := 2
T_c4_ref := 1
U_c4_ref := 1
V_c4_ref := 1
W_c4_ref := 0
X_c4_ref := 0
Y_c4_ref := 0
Z_c4_ref := 2
AA_c4_ref := 2
AB_c4_ref := 2
AC_c4_ref := 1
AD_c4_ref := 0
AE_c4_ref := 2
AF_c4_ref := 2
AG_c4_ref := 2
P_c4_ref := 2
N_c4_ref := 2

VA1 :
A_level := A5 + A4 + A3 + A2 + A1
B_level := B5 + B4 + B3 + B2 + B1
C_level := C5 + C4 + C3 + C2 + C1
D_level := D5 + D4 + D3 + D2 + D1
E_level := E5 + E4 + E3 + E2 + E1
F_level := F5 + F4 + F3 + F2 + F1
G_level := G5 + G4 + G3 + G2 + G1
H_level := H5 + H4 + H3 + H2 + H1
I_level := I5 + I4 + I3 + I2 + I1
J_level := J5 + J4 + J3 + J2 + J1
K_level := K5 + K4 + K3 + K2 + K1
L_level := L5 + L4 + L3 + L2 + L1
M_level := M5 + M4 + M3 + M2 + M1
Q_level := Q5 + Q4 + Q3 + Q2 + Q1
R_level := R5 + R4 + R3 + R2 + R1
S_level := S5 + S4 + S3 + S2 + S1
T_level := T5 + T4 + T3 + T2 + T1
U_level := U5 + U4 + U3 + U2 + U1
V_level := V5 + V4 + V3 + V2 + V1
W_level := W5 + W4 + W3 + W2 + W1
X_level := X5 + X4 + X3 + X2 + X1
Y_level := Y5 + Y4 + Y3 + Y2 + Y1
Z_level := Z5 + Z4 + Z3 + Z2 + Z1
AA_level := AA5 + AA4 + AA3 + AA2 + AA1
AB_level := AB5 + AB4 + AB3 + AB2 + AB1
AC_level := AC5 + AC4 + AC3 + AC2 + AC1
AD_level := AD5 + AD4 + AD3 + AD2 + AD1
AE_level := AE5 + AE4 + AE3 + AE2 + AE1
AF_level := AF5 + AF4 + AF3 + AF2 + AF1
AG_level := AG5 + AG4 + AG3 + AG2 + AG1
P_level := P5 + P4 + P3 + P2 + P1
N_level := N5 + N4 + N3+ N2 + N1




Tri1 >0.99
Tri1 < 0.99
Tri1 <-0.99
Tri1 > -0.99
vref_f1:=vref
vref_f1 := vref
Tri2 >0.99
Tri2 < 0.99
Tri2 <-0.99
Tri2 > -0.99
vref_f2:=vref
vref_f2 := vref
Tri3 >0.99
Tri3 < 0.99
Tri3 <-0.99
Tri3 > -0.99
vref_f3:=vref
vref_f3 := vref
Tri4 >0.99
Tri4 < 0.99
Tri4 <-0.99
Tri4 > -0.99
vref_f4:=vref
vref_f4 := vref
Tri5 >0.99
Tri5 < 0.99
Tri5 <-0.99
Tri5 > -0.99
vref_f5:=vref
vref_f5 := vref


Sheet_sort_Vd
Vd1 := Vd1
Vd2 := Vd2
Vd3 := Vd3
Vd4 := Vd4
g1 := g1
g2 := g2
g3 := g3
g4 := g4
a1 := a1
a2 := a2
a3 := a3
a4 := a4
Vd1_r := Vd1_r
Vd2_r := Vd2_r
Vd4_r := Vd4_r
Vd3_r := Vd3_r
input
output
State_out1
TRUE
choose_priorities := 1
TRUE
sort_Vd = 1
sort_Vd := 0
TRUE
State_in1

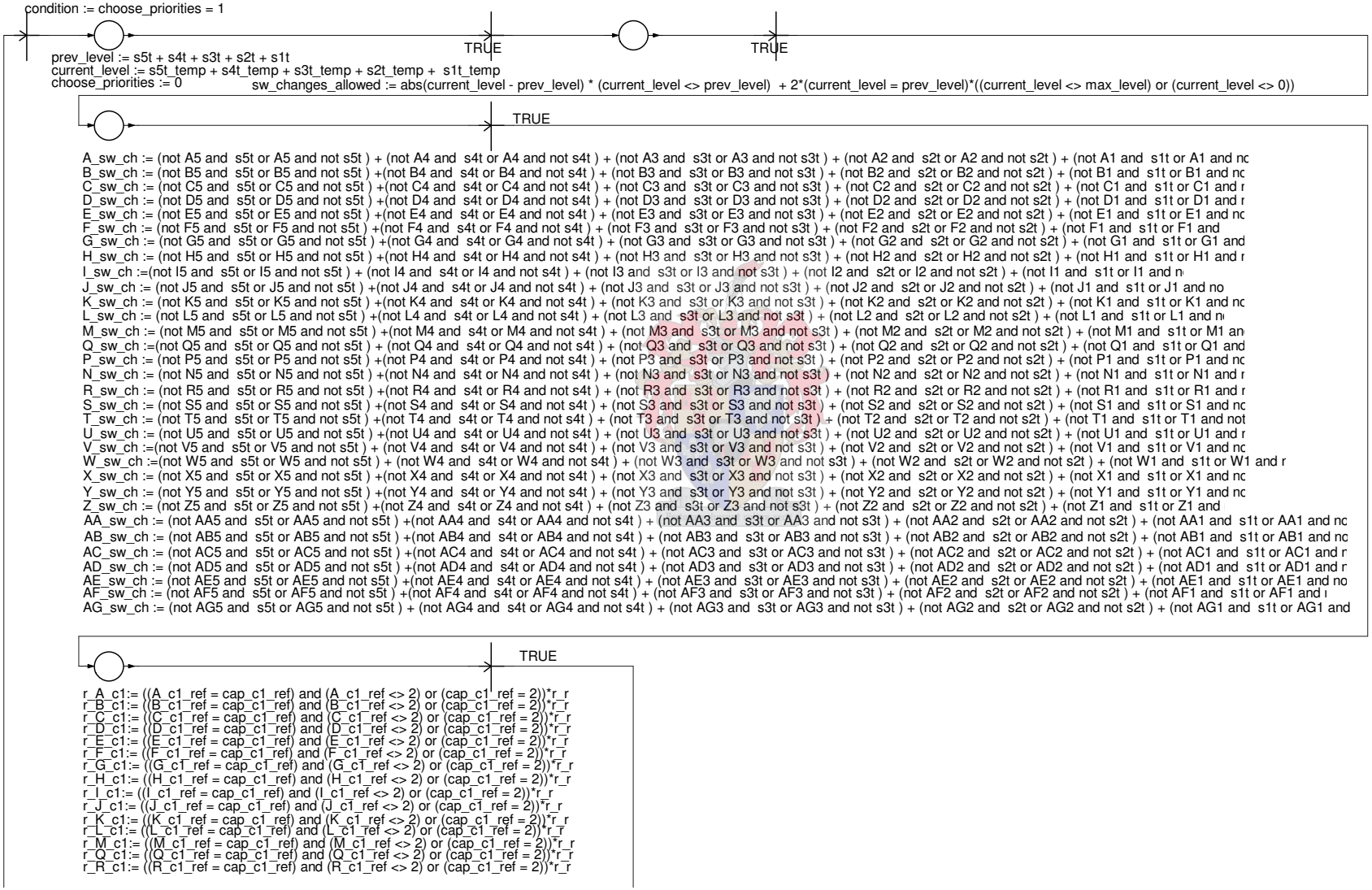


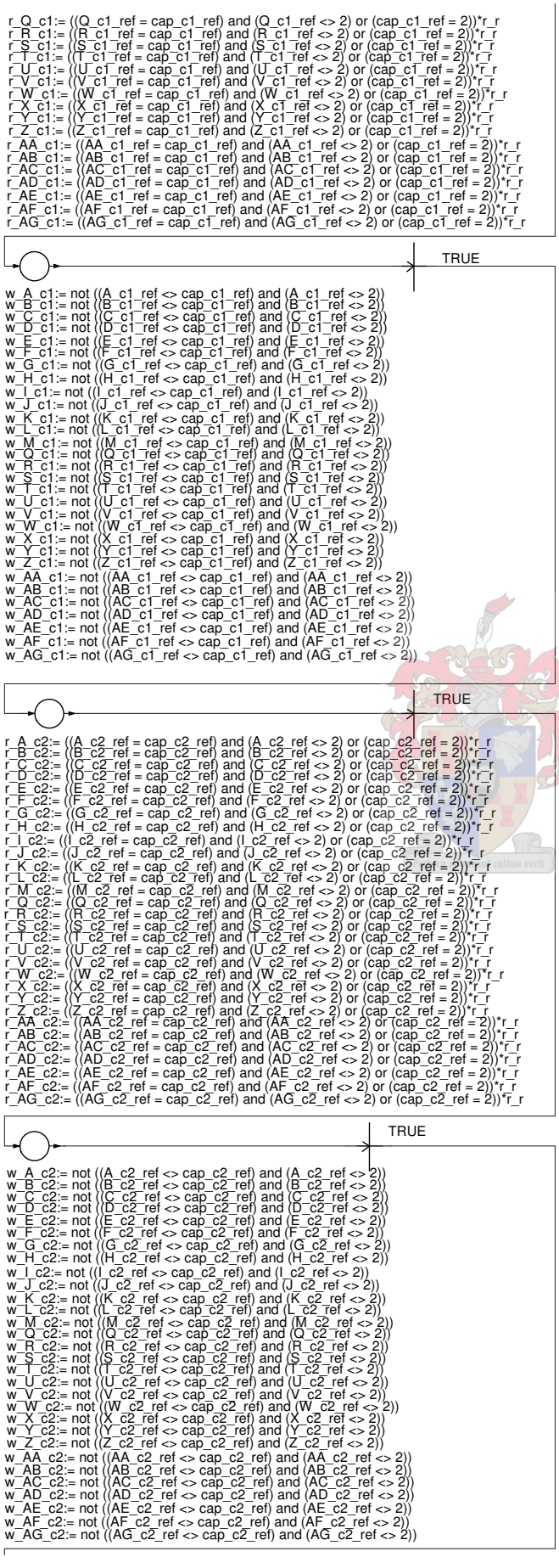
Vd1_r := 3*(g1 = 1) + 2*(g2 = 1) + (g3 = 1

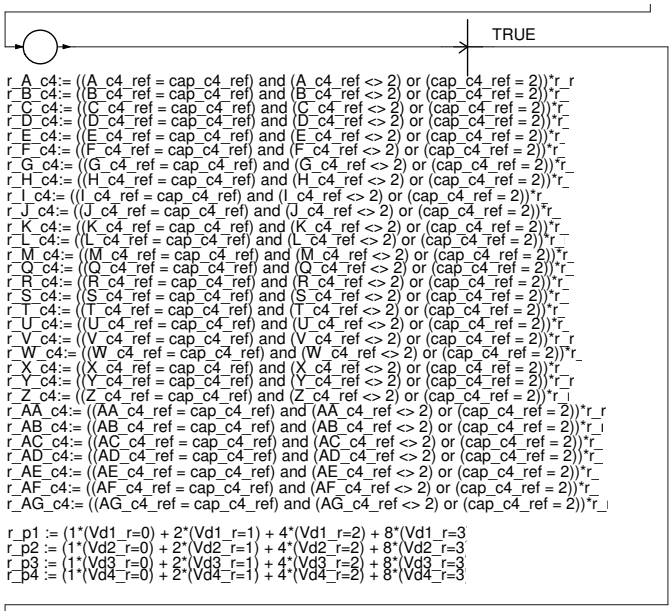
Vd2_r := 3*(g1 = 2) + 2*(g2 = 2) + (g3 = 2

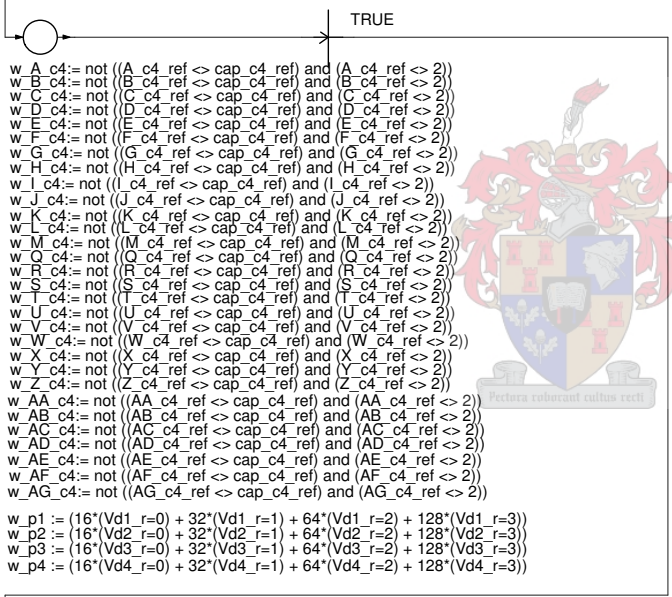
Vd3_r := 3*(g1 = 3) + 2*(g2 = 3) + (g3 = 3

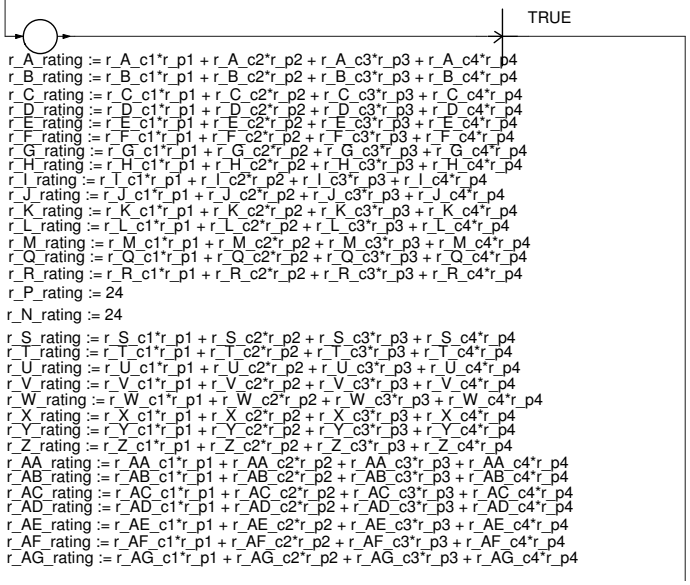
Vd4_r := 3*(g1 = 4) + 2*(g2 = 4) + (g3 = 4)

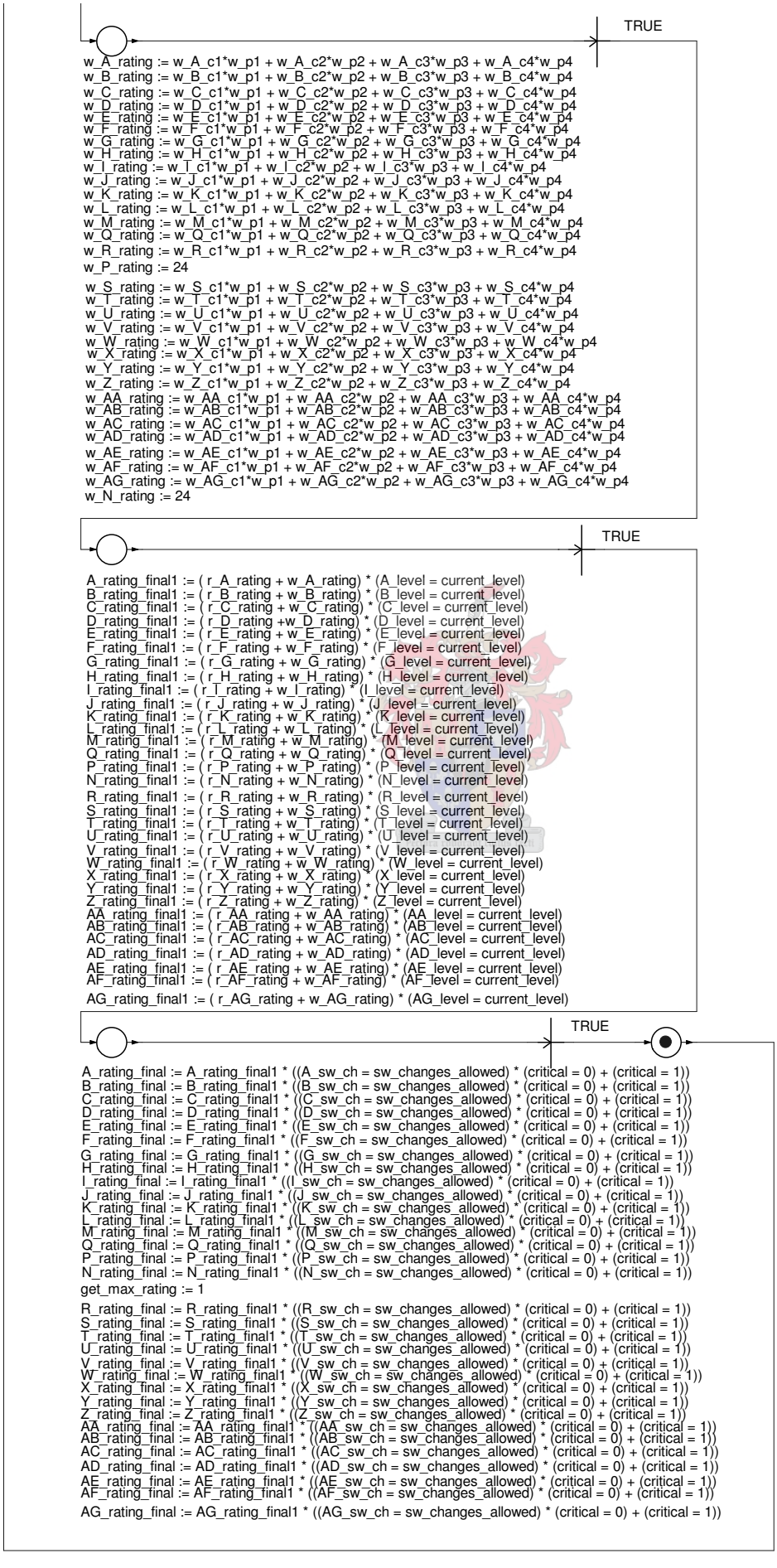

condition := choose_priorities = 1
TRUE
TRUE
prev_level := s5t + s4t + s3t + s2t + s1t
current_level := s5t_temp + s4t_temp + s3t_temp + s2t_temp + s1t_temp
choose_priorities := 0
sw_changes_allowed := abs(current_level - prev_level) * (current_level <> prev_level) + 2*(current_level = prev_level)*((current_level <> max_level) or (current_level <> 0))
TRUE
A_sw_ch := (not A5 and s5t or A5 and not s5t) + (not A4 and s4t or A4 and not s4t) + (not A3 and s3t or A3 and not s3t) + (not A2 and s2t or A2 and not s2t) + (not A1 and s1t or A1 and nc
B_sw_ch := (not B5 and s5t or B5 and not s5t) +(not B4 and s4t or B4 and not s4t) + (not B3 and s3t or B3 and not s3t) + (not B2 and s2t or B2 and not s2t) + (not B1 and s1t or B1 and nc
C_sw_ch := (not C5 and s5t or C5 and not s5t) +(not C4 and s4t or C4 and not s4t) + (not C3 and s3t or C3 and not s3t) + (not C2 and s2t or C2 and not s2t) + (not C1 and s1t or C1 and r
D_sw_ch := (not D5 and s5t or D5 and not s5t) +(not D4 and s4t or D4 and not s4t) + (not D3 and s3t or D3 and not s3t) + (not D2 and s2t or D2 and not s2t) + (not D1 and s1t or D1 and r
E_sw_ch := (not E5 and s5t or E5 and not s5t) +(not E4 and s4t or E4 and not s4t) + (not E3 and s3t or E3 and not s3t) + (not E2 and s2t or E2 and not s2t) + (not E1 and s1t or E1 and nc
F_sw_ch := (not F5 and s5t or F5 and not s5t) +(not F4 and s4t or F4 and not s4t) + (not F3 and s3t or F3 and not s3t) + (not F2 and s2t or F2 and not s2t) + (not F1 and s1t or F1 and
G_sw_ch := (not G5 and s5t or G5 and not s5t) +(not G4 and s4t or G4 and not s4t) + (not G3 and s3t or G3 and not s3t) + (not G2 and s2t or G2 and not s2t) + (not G1 and s1t or G1 and
H_sw_ch := (not H5 and s5t or H5 and not s5t) +(not H4 and s4t or H4 and not s4t) + (not H3 and s3t or H3 and not s3t) + (not H2 and s2t or H2 and not s2t) + (not H1 and s1t or H1 and r
I_sw_ch :=(not I5 and s5t or I5 and not s5t) + (not I4 and s4t or I4 and not s4t) + (not I3 and s3t or I3 and not s3t) + (not I2 and s2t or I2 and not s2t) + (not I1 and s1t or I1 and n
J_sw_ch := (not J5 and s5t or J5 and not s5t) +(not J4 and s4t or J4 and not s4t) + (not J3 and s3t or J3 and not s3t) + (not J2 and s2t or J2 and not s2t) + (not J1 and s1t or J1 and no
K_sw_ch := (not K5 and s5t or K5 and not s5t) +(not K4 and s4t or K4 and not s4t) + (not K3 and s3t or K3 and not s3t) + (not K2 and s2t or K2 and not s2t) + (not K1 and s1t or K1 and nc
L_sw_ch := (not L5 and s5t or L5 and not s5t) +(not L4 and s4t or L4 and not s4t) + (not L3 and s3t or L3 and not s3t) + (not L2 and s2t or L2 and not s2t) + (not L1 and s1t or L1 and n
M_sw_ch := (not M5 and s5t or M5 and not s5t) +(not M4 and s4t or M4 and not s4t) + (not M3 and s3t or M3 and not s3t) + (not M2 and s2t or M2 and not s2t) + (not M1 and s1t or M1 an
Q_sw_ch :=(not Q5 and s5t or Q5 and not s5t) + (not Q4 and s4t or Q4 and not s4t) + (not Q3 and s3t or Q3 and not s3t) + (not Q2 and s2t or Q2 and not s2t) + (not Q1 and s1t or Q1 and
P_sw_ch := (not P5 and s5t or P5 and not s5t) +(not P4 and s4t or P4 and not s4t) + (not P3 and s3t or P3 and not s3t) + (not P2 and s2t or P2 and not s2t) + (not P1 and s1t or P1 and nc
N_sw_ch := (not N5 and s5t or N5 and not s5t) +(not N4 and s4t or N4 and not s4t) + (not N3 and s3t or N3 and not s3t) + (not N2 and s2t or N2 and not s2t) + (not N1 and s1t or N1 and r
R_sw_ch := (not R5 and s5t or R5 and not s5t) +(not R4 and s4t or R4 and not s4t) + (not R3 and s3t or R3 and not s3t) + (not R2 and s2t or R2 and not s2t) + (not R1 and s1t or R1 and r
S_sw_ch := (not S5 and s5t or S5 and not s5t) +(not S4 and s4t or S4 and not s4t) + (not S3 and s3t or S3 and not s3t) + (not S2 and s2t or S2 and not s2t) + (not S1 and s1t or S1 and nc
T_sw_ch := (not T5 and s5t or T5 and not s5t) +(not T4 and s4t or T4 and not s4t) + (not T3 and s3t or T3 and not s3t) + (not T2 and s2t or T2 and not s2t) + (not T1 and s1t or T1 and not
U_sw_ch := (not U5 and s5t or U5 and not s5t) +(not U4 and s4t or U4 and not s4t) + (not U3 and s3t or U3 and not s3t) + (not U2 and s2t or U2 and not s2t) + (not U1 and s1t or U1 and r
V_sw_ch :=(not V5 and s5t or V5 and not s5t) + (not V4 and s4t or V4 and not s4t) + (not V3 and s3t or V3 and not s3t) + (not V2 and s2t or V2 and not s2t) + (not V1 and s1t or V1 and nc
W_sw_ch :=(not W5 and s5t or W5 and not s5t) + (not W4 and s4t or W4 and not s4t) + (not W3 and s3t or W3 and not s3t) + (not W2 and s2t or W2 and not s2t) + (not W1 and s1t or W1 and r
X_sw_ch := (not X5 and s5t or X5 and not s5t) +(not X4 and s4t or X4 and not s4t) + (not X3 and s3t or X3 and not s3t) + (not X2 and s2t or X2 and not s2t) + (not X1 and s1t or X1 and nc
Y_sw_ch := (not Y5 and s5t or Y5 and not s5t) +(not Y4 and s4t or Y4 and not s4t) + (not Y3 and s3t or Y3 and not s3t) + (not Y2 and s2t or Y2 and not s2t) + (not Y1 and s1t or Y1 and nc
Z_sw_ch := (not Z5 and s5t or Z5 and not s5t) +(not Z4 and s4t or Z4 and not s4t) + (not Z3 and s3t or Z3 and not s3t) + (not Z2 and s2t or Z2 and not s2t) + (not Z1 and s1t or Z1 and
AA_sw_ch := (not AA5 and s5t or AA5 and not s5t) +(not AA4 and s4t or AA4 and not s4t) + (not AA3 and s3t or AA3 and not s3t) + (not AA2 and s2t or AA2 and not s2t) + (not AA1 and s1t or AA1 and nc
AB_sw_ch := (not AB5 and s5t or AB5 and not s5t) +(not AB4 and s4t or AB4 and not s4t) + (not AB3 and s3t or AB3 and not s3t) + (not AB2 and s2t or AB2 and not s2t) + (not AB1 and s1t or AB1 and nc
AC_sw_ch := (not AC5 and s5t or AC5 and not s5t) +(not AC4 and s4t or AC4 and not s4t) + (not AC3 and s3t or AC3 and not s3t) + (not AC2 and s2t or AC2 and not s2t) + (not AC1 and s1t or AC1 and r
AD_sw_ch := (not AD5 and s5t or AD5 and not s5t) +(not AD4 and s4t or AD4 and not s4t) + (not AD3 and s3t or AD3 and not s3t) + (not AD2 and s2t or AD2 and not s2t) + (not AD1 and s1t or AD1 and r
AE_sw_ch := (not AE5 and s5t or AE5 and not s5t) +(not AE4 and s4t or AE4 and not s4t) + (not AE3 and s3t or AE3 and not s3t) + (not AE2 and s2t or AE2 and not s2t) + (not AE1 and s1t or AE1 and no
AF_sw_ch := (not AF5 and s5t or AF5 and not s5t) +(not AF4 and s4t or AF4 and not s4t) + (not AF3 and s3t or AF3 and not s3t) + (not AF2 and s2t or AF2 and not s2t) + (not AF1 and s1t or AF1 and r
AG_sw_ch := (not AG5 and s5t or AG5 and not s5t) + (not AG4 and s4t or AG4 and not s4t) + (not AG3 and s3t or AG3 and not s3t) + (not AG2 and s2t or AG2 and not s2t) + (not AG1 and s1t or AG1 and
TRUE
r_A_c1:= ((A_c1_ref = cap_c1_ref) and (A_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_B_c1:= ((B_c1_ref = cap_c1_ref) and (B_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_C_c1:= ((C_c1_ref = cap_c1_ref) and (C_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_D_c1:= ((D_c1_ref = cap_c1_ref) and (D_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_E_c1:= ((E_c1_ref = cap_c1_ref) and (E_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_F_c1:= ((F_c1_ref = cap_c1_ref) and (F_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_G_c1:= ((G_c1_ref = cap_c1_ref) and (G_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_H_c1:= ((H_c1_ref = cap_c1_ref) and (H_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_I_c1:= ((I_c1_ref = cap_c1_ref) and (I_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_J_c1:= ((J_c1_ref = cap_c1_ref) and (J_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_K_c1:= ((K_c1_ref = cap_c1_ref) and (K_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_L_c1:= ((L_c1_ref = cap_c1_ref) and (L_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_M_c1:= ((M_c1_ref = cap_c1_ref) and (M_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_Q_c1:= ((Q_c1_ref = cap_c1_ref) and (Q_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_R_c1:= ((R_c1_ref = cap_c1_ref) and (R_c1_ref <> 2) or (cap_c1_ref = 2))*r_r

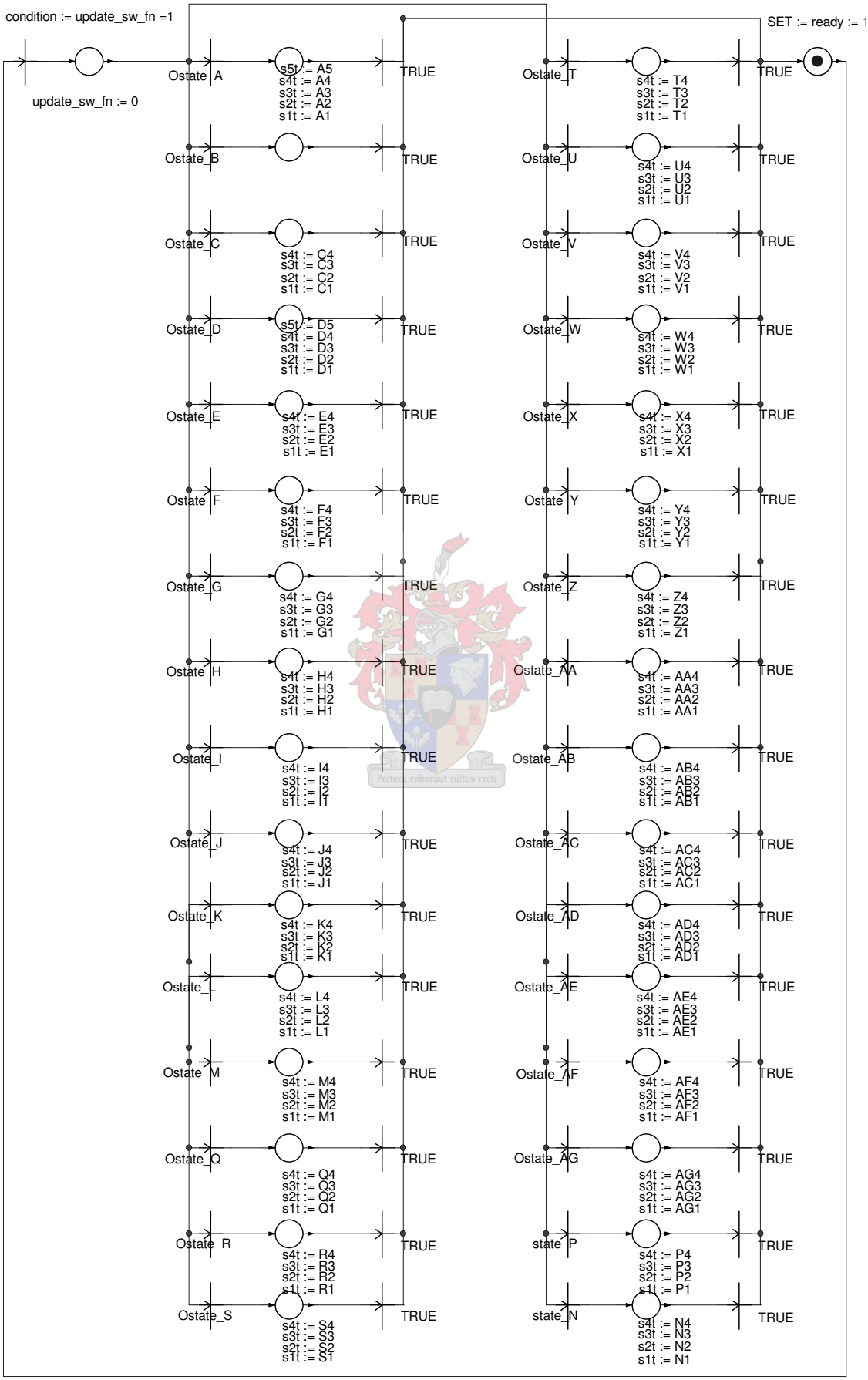

r_Q_c1:= ((Q_c1_ref = cap_c1_ref) and (Q_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_R_c1:= ((R_c1_ref = cap_c1_ref) and (R_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_S_c1:= ((S_c1_ref = cap_c1_ref) and (S_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_T_c1:= ((T_c1_ref = cap_c1_ref) and (T_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_U_c1:= ((U_c1_ref = cap_c1_ref) and (U_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_V_c1:= ((V_c1_ref = cap_c1_ref) and (V_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_W_c1:= ((W_c1_ref = cap_c1_ref) and (W_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_X_c1:= ((X_c1_ref = cap_c1_ref) and (X_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_Y_c1:= ((Y_c1_ref = cap_c1_ref) and (Y_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_Z_c1:= ((Z_c1_ref = cap_c1_ref) and (Z_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_AA_c1:= ((AA_c1_ref = cap_c1_ref) and (AA_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_AB_c1:= ((AB_c1_ref = cap_c1_ref) and (AB_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_AC_c1:= ((AC_c1_ref = cap_c1_ref) and (AC_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_AD_c1:= ((AD_c1_ref = cap_c1_ref) and (AD_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_AE_c1:= ((AE_c1_ref = cap_c1_ref) and (AE_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_AF_c1:= ((AF_c1_ref = cap_c1_ref) and (AF_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
r_AG_c1:= ((AG_c1_ref = cap_c1_ref) and (AG_c1_ref <> 2) or (cap_c1_ref = 2))*r_r
TRUE
w_A_c1:= not ((A_c1_ref <> cap_c1_ref) and (A_c1_ref <> 2))
w_B_c1:= not ((B_c1_ref <> cap_c1_ref) and (B_c1_ref <> 2))
w_C_c1:= not ((C_c1_ref <> cap_c1_ref) and (C_c1_ref <> 2))
w_D_c1:= not ((D_c1_ref <> cap_c1_ref) and (D_c1_ref <> 2))
w_E_c1:= not ((E_c1_ref <> cap_c1_ref) and (E_c1_ref <> 2))
w_F_c1:= not ((F_c1_ref <> cap_c1_ref) and (F_c1_ref <> 2))
w_G_c1:= not ((G_c1_ref <> cap_c1_ref) and (G_c1_ref <> 2))
w_H_c1:= not ((H_c1_ref <> cap_c1_ref) and (H_c1_ref <> 2))
w_I_c1:= not ((I_c1_ref <> cap_c1_ref) and (I_c1_ref <> 2))
w_J_c1:= not ((J_c1_ref <> cap_c1_ref) and (J_c1_ref <> 2))
w_K_c1:= not ((K_c1_ref <> cap_c1_ref) and (K_c1_ref <> 2))
w_L_c1:= not ((L_c1_ref <> cap_c1_ref) and (L_c1_ref <> 2))
w_M_c1:= not ((M_c1_ref <> cap_c1_ref) and (M_c1_ref <> 2))
w_Q_c1:= not ((Q_c1_ref <> cap_c1_ref) and (Q_c1_ref <> 2))
w_R_c1:= not ((R_c1_ref <> cap_c1_ref) and (R_c1_ref <> 2))
w_S_c1:= not ((S_c1_ref <> cap_c1_ref) and (S_c1_ref <> 2))
w_T_c1:= not ((T_c1_ref <> cap_c1_ref) and (T_c1_ref <> 2))
w_U_c1:= not ((U_c1_ref <> cap_c1_ref) and (U_c1_ref <> 2))
w_V_c1:= not ((V_c1_ref <> cap_c1_ref) and (V_c1_ref <> 2))
w_W_c1:= not ((W_c1_ref <> cap_c1_ref) and (W_c1_ref <> 2))
w_X_c1:= not ((X_c1_ref <> cap_c1_ref) and (X_c1_ref <> 2))
w_Y_c1:= not ((Y_c1_ref <> cap_c1_ref) and (Y_c1_ref <> 2))
w_Z_c1:= not ((Z_c1_ref <> cap_c1_ref) and (Z_c1_ref <> 2))
w_AA_c1:= not ((AA_c1_ref <> cap_c1_ref) and (AA_c1_ref <> 2))
w_AB_c1:= not ((AB_c1_ref <> cap_c1_ref) and (AB_c1_ref <> 2))
w_AC_c1:= not ((AC_c1_ref <> cap_c1_ref) and (AC_c1_ref <> 2))
w_AD_c1:= not ((AD_c1_ref <> cap_c1_ref) and (AD_c1_ref <> 2))
w_AE_c1:= not ((AE_c1_ref <> cap_c1_ref) and (AE_c1_ref <> 2))
w_AF_c1:= not ((AF_c1_ref <> cap_c1_ref) and (AF_c1_ref <> 2))
w_AG_c1:= not ((AG_c1_ref <> cap_c1_ref) and (AG_c1_ref <> 2))
TRUE
r_A_c2:= ((A_c2_ref = cap_c2_ref) and (A_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_B_c2:= ((B_c2_ref = cap_c2_ref) and (B_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_C_c2:= ((C_c2_ref = cap_c2_ref) and (C_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_D_c2:= ((D_c2_ref = cap_c2_ref) and (D_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_E_c2:= ((E_c2_ref = cap_c2_ref) and (E_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_F_c2:= ((F_c2_ref = cap_c2_ref) and (F_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_G_c2:= ((G_c2_ref = cap_c2_ref) and (G_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_H_c2:= ((H_c2_ref = cap_c2_ref) and (H_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_I_c2:= ((I_c2_ref = cap_c2_ref) and (I_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_J_c2:= ((J_c2_ref = cap_c2_ref) and (J_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_K_c2:= ((K_c2_ref = cap_c2_ref) and (K_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_L_c2:= ((L_c2_ref = cap_c2_ref) and (L_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_M_c2:= ((M_c2_ref = cap_c2_ref) and (M_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_Q_c2:= ((Q_c2_ref = cap_c2_ref) and (Q_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_R_c2:= ((R_c2_ref = cap_c2_ref) and (R_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_S_c2:= ((S_c2_ref = cap_c2_ref) and (S_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_T_c2:= ((T_c2_ref = cap_c2_ref) and (T_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_U_c2:= ((U_c2_ref = cap_c2_ref) and (U_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_V_c2:= ((V_c2_ref = cap_c2_ref) and (V_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_W_c2:= ((W_c2_ref = cap_c2_ref) and (W_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_X_c2:= ((X_c2_ref = cap_c2_ref) and (X_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_Y_c2:= ((Y_c2_ref = cap_c2_ref) and (Y_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_Z_c2:= ((Z_c2_ref = cap_c2_ref) and (Z_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_AA_c2:= ((AA_c2_ref = cap_c2_ref) and (AA_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_AB_c2:= ((AB_c2_ref = cap_c2_ref) and (AB_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_AC_c2:= ((AC_c2_ref = cap_c2_ref) and (AC_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_AD_c2:= ((AD_c2_ref = cap_c2_ref) and (AD_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_AE_c2:= ((AE_c2_ref = cap_c2_ref) and (AE_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_AF_c2:= ((AF_c2_ref = cap_c2_ref) and (AF_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
r_AG_c2:= ((AG_c2_ref = cap_c2_ref) and (AG_c2_ref <> 2) or (cap_c2_ref = 2))*r_r
TRUE
w_A_c2:= not ((A_c2_ref <> cap_c2_ref) and (A_c2_ref <> 2))
w_B_c2:= not ((B_c2_ref <> cap_c2_ref) and (B_c2_ref <> 2))
w_C_c2:= not ((C_c2_ref <> cap_c2_ref) and (C_c2_ref <> 2))
w_D_c2:= not ((D_c2_ref <> cap_c2_ref) and (D_c2_ref <> 2))
w_E_c2:= not ((E_c2_ref <> cap_c2_ref) and (E_c2_ref <> 2))
w_F_c2:= not ((F_c2_ref <> cap_c2_ref) and (F_c2_ref <> 2))
w_G_c2:= not ((G_c2_ref <> cap_c2_ref) and (G_c2_ref <> 2))
w_H_c2:= not ((H_c2_ref <> cap_c2_ref) and (H_c2_ref <> 2))
w_I_c2:= not ((I_c2_ref <> cap_c2_ref) and (I_c2_ref <> 2))
w_J_c2:= not ((J_c2_ref <> cap_c2_ref) and (J_c2_ref <> 2))
w_K_c2:= not ((K_c2_ref <> cap_c2_ref) and (K_c2_ref <> 2))
w_L_c2:= not ((L_c2_ref <> cap_c2_ref) and (L_c2_ref <> 2))
w_M_c2:= not ((M_c2_ref <> cap_c2_ref) and (M_c2_ref <> 2))
w_Q_c2:= not ((Q_c2_ref <> cap_c2_ref) and (Q_c2_ref <> 2))
w_R_c2:= not ((R_c2_ref <> cap_c2_ref) and (R_c2_ref <> 2))
w_S_c2:= not ((S_c2_ref <> cap_c2_ref) and (S_c2_ref <> 2))
w_T_c2:= not ((T_c2_ref <> cap_c2_ref) and (T_c2_ref <> 2))
w_U_c2:= not ((U_c2_ref <> cap_c2_ref) and (U_c2_ref <> 2))
w_V_c2:= not ((V_c2_ref <> cap_c2_ref) and (V_c2_ref <> 2))
w_W_c2:= not ((W_c2_ref <> cap_c2_ref) and (W_c2_ref <> 2))
w_X_c2:= not ((X_c2_ref <> cap_c2_ref) and (X_c2_ref <> 2))
w_Y_c2:= not ((Y_c2_ref <> cap_c2_ref) and (Y_c2_ref <> 2))
w_Z_c2:= not ((Z_c2_ref <> cap_c2_ref) and (Z_c2_ref <> 2))
w_AA_c2:= not ((AA_c2_ref <> cap_c2_ref) and (AA_c2_ref <> 2))
w_AB_c2:= not ((AB_c2_ref <> cap_c2_ref) and (AB_c2_ref <> 2))
w_AC_c2:= not ((AC_c2_ref <> cap_c2_ref) and (AC_c2_ref <> 2))
w_AD_c2:= not ((AD_c2_ref <> cap_c2_ref) and (AD_c2_ref <> 2))
w_AE_c2:= not ((AE_c2_ref <> cap_c2_ref) and (AE_c2_ref <> 2))
w_AF_c2:= not ((AF_c2_ref <> cap_c2_ref) and (AF_c2_ref <> 2))
w_AG_c2:= not ((AG_c2_ref <> cap_c2_ref) and (AG_c2_ref <> 2))

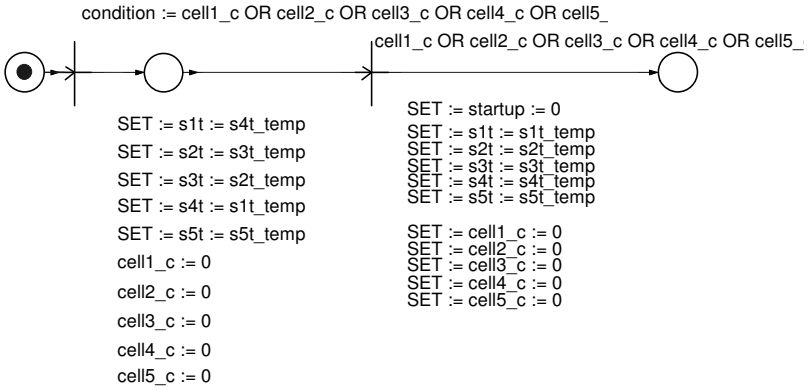

TRUE
r_A_c4:= ((A_c4_ref = cap_c4_ref) and (A_c4_ref <> 2) or (cap_c4_ref = 2))*r_r
r_B_c4:= ((B_c4_ref = cap_c4_ref) and (B_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_C_c4:= ((C_c4_ref = cap_c4_ref) and (C_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_D_c4:= ((D_c4_ref = cap_c4_ref) and (D_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_E_c4:= ((E_c4_ref = cap_c4_ref) and (E_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_F_c4:= ((F_c4_ref = cap_c4_ref) and (F_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_G_c4:= ((G_c4_ref = cap_c4_ref) and (G_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_H_c4:= ((H_c4_ref = cap_c4_ref) and (H_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_I_c4:= ((I_c4_ref = cap_c4_ref) and (I_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_J_c4:= ((J_c4_ref = cap_c4_ref) and (J_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_K_c4:= ((K_c4_ref = cap_c4_ref) and (K_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_L_c4:= ((L_c4_ref = cap_c4_ref) and (L_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_M_c4:= ((M_c4_ref = cap_c4_ref) and (M_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_Q_c4:= ((Q_c4_ref = cap_c4_ref) and (Q_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_R_c4:= ((R_c4_ref = cap_c4_ref) and (R_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_S_c4:= ((S_c4_ref = cap_c4_ref) and (S_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_T_c4:= ((T_c4_ref = cap_c4_ref) and (T_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_U_c4:= ((U_c4_ref = cap_c4_ref) and (U_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_V_c4:= ((V_c4_ref = cap_c4_ref) and (V_c4_ref <> 2) or (cap_c4_ref = 2))*r_r
r_W_c4:= ((W_c4_ref = cap_c4_ref) and (W_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_X_c4:= ((X_c4_ref = cap_c4_ref) and (X_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_Y_c4:= ((Y_c4_ref = cap_c4_ref) and (Y_c4_ref <> 2) or (cap_c4_ref = 2))*r_r
r_Z_c4:= ((Z_c4_ref = cap_c4_ref) and (Z_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_AA_c4:= ((AA_c4_ref = cap_c4_ref) and (AA_c4_ref <> 2) or (cap_c4_ref = 2))*r_r
r_AB_c4:= ((AB_c4_ref = cap_c4_ref) and (AB_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_AC_c4:= ((AC_c4_ref = cap_c4_ref) and (AC_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_AD_c4:= ((AD_c4_ref = cap_c4_ref) and (AD_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_AE_c4:= ((AE_c4_ref = cap_c4_ref) and (AE_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_AF_c4:= ((AF_c4_ref = cap_c4_ref) and (AF_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_AG_c4:= ((AG_c4_ref = cap_c4_ref) and (AG_c4_ref <> 2) or (cap_c4_ref = 2))*r_
r_p1 := (1*(Vd1_r=0) + 2*(Vd1_r=1) + 4*(Vd1_r=2) + 8*(Vd1_r=3
r_p2 := (1*(Vd2_r=0) + 2*(Vd2_r=1) + 4*(Vd2_r=2) + 8*(Vd2_r=3
r_p3 := (1*(Vd3_r=0) + 2*(Vd3_r=1) + 4*(Vd3_r=2) + 8*(Vd3_r=3
r_p4 := (1*(Vd4_r=0) + 2*(Vd4_r=1) + 4*(Vd4_r=2) + 8*(Vd4_r=3


TRUE
w_A_c4:= not ((A_c4_ref <> cap_c4_ref) and (A_c4_ref <> 2))
w_B_c4:= not ((B_c4_ref <> cap_c4_ref) and (B_c4_ref <> 2))
w_C_c4:= not ((C_c4_ref <> cap_c4_ref) and (C_c4_ref <> 2))
w_D_c4:= not ((D_c4_ref <> cap_c4_ref) and (D_c4_ref <> 2))
w_E_c4:= not ((E_c4_ref <> cap_c4_ref) and (E_c4_ref <> 2))
w_F_c4:= not ((F_c4_ref <> cap_c4_ref) and (F_c4_ref <> 2))
w_G_c4:= not ((G_c4_ref <> cap_c4_ref) and (G_c4_ref <> 2))
w_H_c4:= not ((H_c4_ref <> cap_c4_ref) and (H_c4_ref <> 2))
w_I_c4:= not ((I_c4_ref <> cap_c4_ref) and (I_c4_ref <> 2))
w_J_c4:= not ((J_c4_ref <> cap_c4_ref) and (J_c4_ref <> 2))
w_K_c4:= not ((K_c4_ref <> cap_c4_ref) and (K_c4_ref <> 2))
w_L_c4:= not ((L_c4_ref <> cap_c4_ref) and (L_c4_ref <> 2))
w_M_c4:= not ((M_c4_ref <> cap_c4_ref) and (M_c4_ref <> 2))
w_Q_c4:= not ((Q_c4_ref <> cap_c4_ref) and (Q_c4_ref <> 2))
w_R_c4:= not ((R_c4_ref <> cap_c4_ref) and (R_c4_ref <> 2))
w_S_c4:= not ((S_c4_ref <> cap_c4_ref) and (S_c4_ref <> 2))
w_T_c4:= not ((T_c4_ref <> cap_c4_ref) and (T_c4_ref <> 2))
w_U_c4:= not ((U_c4_ref <> cap_c4_ref) and (U_c4_ref <> 2))
w_V_c4:= not ((V_c4_ref <> cap_c4_ref) and (V_c4_ref <> 2))
w_W_c4:= not ((W_c4_ref <> cap_c4_ref) and (W_c4_ref <> 2))
w_X_c4:= not ((X_c4_ref <> cap_c4_ref) and (X_c4_ref <> 2))
w_Y_c4:= not ((Y_c4_ref <> cap_c4_ref) and (Y_c4_ref <> 2))
w_Z_c4:= not ((Z_c4_ref <> cap_c4_ref) and (Z_c4_ref <> 2))
w_AA_c4:= not ((AA_c4_ref <> cap_c4_ref) and (AA_c4_ref <> 2))
w_AB_c4:= not ((AB_c4_ref <> cap_c4_ref) and (AB_c4_ref <> 2))
w_AC_c4:= not ((AC_c4_ref <> cap_c4_ref) and (AC_c4_ref <> 2))
w_AD_c4:= not ((AD_c4_ref <> cap_c4_ref) and (AD_c4_ref <> 2))
w_AE_c4:= not ((AE_c4_ref <> cap_c4_ref) and (AE_c4_ref <> 2))
w_AF_c4:= not ((AF_c4_ref <> cap_c4_ref) and (AF_c4_ref <> 2))
w_AG_c4:= not ((AG_c4_ref <> cap_c4_ref) and (AG_c4_ref <> 2))
w_p1 := (16*(Vd1_r=0) + 32*(Vd1_r=1) + 64*(Vd1_r=2) + 128*(Vd1_r=3))
w_p2 := (16*(Vd2_r=0) + 32*(Vd2_r=1) + 64*(Vd2_r=2) + 128*(Vd2_r=3))
w_p3 := (16*(Vd3_r=0) + 32*(Vd3_r=1) + 64*(Vd3_r=2) + 128*(Vd3_r=3))
w_p4 := (16*(Vd4_r=0) + 32*(Vd4_r=1) + 64*(Vd4_r=2) + 128*(Vd4_r=3))

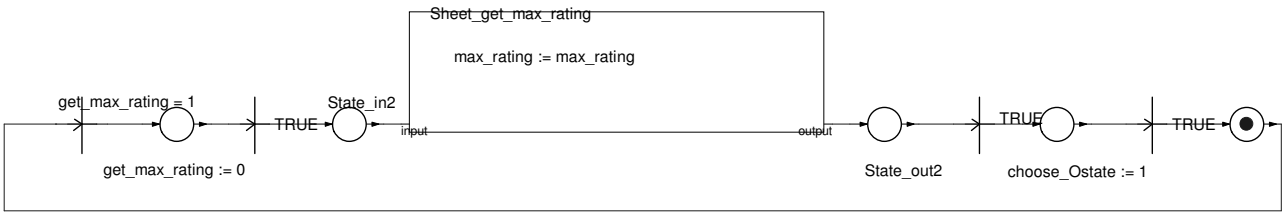

TRUE
r_A_rating := r_A_c1*r_p1 + r_A_c2*r_p2 + r_A_c3*r_p3 + r_A_c4*r_p4
r_B_rating := r_B_c1*r_p1 + r_B_c2*r_p2 + r_B_c3*r_p3 + r_B_c4*r_p4
r_C_rating := r_C_c1*r_p1 + r_C_c2*r_p2 + r_C_c3*r_p3 + r_C_c4*r_p4
r_D_rating := r_D_c1*r_p1 + r_D_c2*r_p2 + r_D_c3*r_p3 + r_D_c4*r_p4
r_E_rating := r_E_c1*r_p1 + r_E_c2*r_p2 + r_E_c3*r_p3 + r_E_c4*r_p4
r_F_rating := r_F_c1*r_p1 + r_F_c2*r_p2 + r_F_c3*r_p3 + r_F_c4*r_p4
r_G_rating := r_G_c1*r_p1 + r_G_c2*r_p2 + r_G_c3*r_p3 + r_G_c4*r_p4
r_H_rating := r_H_c1*r_p1 + r_H_c2*r_p2 + r_H_c3*r_p3 + r_H_c4*r_p4
r_I_rating := r_I_c1*r_p1 + r_I_c2*r_p2 + r_I_c3*r_p3 + r_I_c4*r_p4
r_J_rating := r_J_c1*r_p1 + r_J_c2*r_p2 + r_J_c3*r_p3 + r_J_c4*r_p4
r_K_rating := r_K_c1*r_p1 + r_K_c2*r_p2 + r_K_c3*r_p3 + r_K_c4*r_p4
r_L_rating := r_L_c1*r_p1 + r_L_c2*r_p2 + r_L_c3*r_p3 + r_L_c4*r_p4
r_M_rating := r_M_c1*r_p1 + r_M_c2*r_p2 + r_M_c3*r_p3 + r_M_c4*r_p4
r_Q_rating := r_Q_c1*r_p1 + r_Q_c2*r_p2 + r_Q_c3*r_p3 + r_Q_c4*r_p4
r_R_rating := r_R_c1*r_p1 + r_R_c2*r_p2 + r_R_c3*r_p3 + r_R_c4*r_p4
r_P_rating := 24
r_N_rating := 24
r_S_rating := r_S_c1*r_p1 + r_S_c2*r_p2 + r_S_c3*r_p3 + r_S_c4*r_p4
r_T_rating := r_T_c1*r_p1 + r_T_c2*r_p2 + r_T_c3*r_p3 + r_T_c4*r_p4
r_U_rating := r_U_c1*r_p1 + r_U_c2*r_p2 + r_U_c3*r_p3 + r_U_c4*r_p4
r_V_rating := r_V_c1*r_p1 + r_V_c2*r_p2 + r_V_c3*r_p3 + r_V_c4*r_p4
r_W_rating := r_W_c1*r_p1 + r_W_c2*r_p2 + r_W_c3*r_p3 + r_W_c4*r_p4
r_X_rating := r_X_c1*r_p1 + r_X_c2*r_p2 + r_X_c3*r_p3 + r_X_c4*r_p4
r_Y_rating := r_Y_c1*r_p1 + r_Y_c2*r_p2 + r_Y_c3*r_p3 + r_Y_c4*r_p4
r_Z_rating := r_Z_c1*r_p1 + r_Z_c2*r_p2 + r_Z_c3*r_p3 + r_Z_c4*r_p4
r_AA_rating := r_AA_c1*r_p1 + r_AA_c2*r_p2 + r_AA_c3*r_p3 + r_AA_c4*r_p4
r_AB_rating := r_AB_c1*r_p1 + r_AB_c2*r_p2 + r_AB_c3*r_p3 + r_AB_c4*r_p4
r_AC_rating := r_AC_c1*r_p1 + r_AC_c2*r_p2 + r_AC_c3*r_p3 + r_AC_c4*r_p4
r_AD_rating := r_AD_c1*r_p1 + r_AD_c2*r_p2 + r_AD_c3*r_p3 + r_AD_c4*r_p4
r_AE_rating := r_AE_c1*r_p1 + r_AE_c2*r_p2 + r_AE_c3*r_p3 + r_AE_c4*r_p4
r_AF_rating := r_AF_c1*r_p1 + r_AF_c2*r_p2 + r_AF_c3*r_p3 + r_AF_c4*r_p4
r_AG_rating := r_AG_c1*r_p1 + r_AG_c2*r_p2 + r_AG_c3*r_p3 + r_AG_c4*r_p4

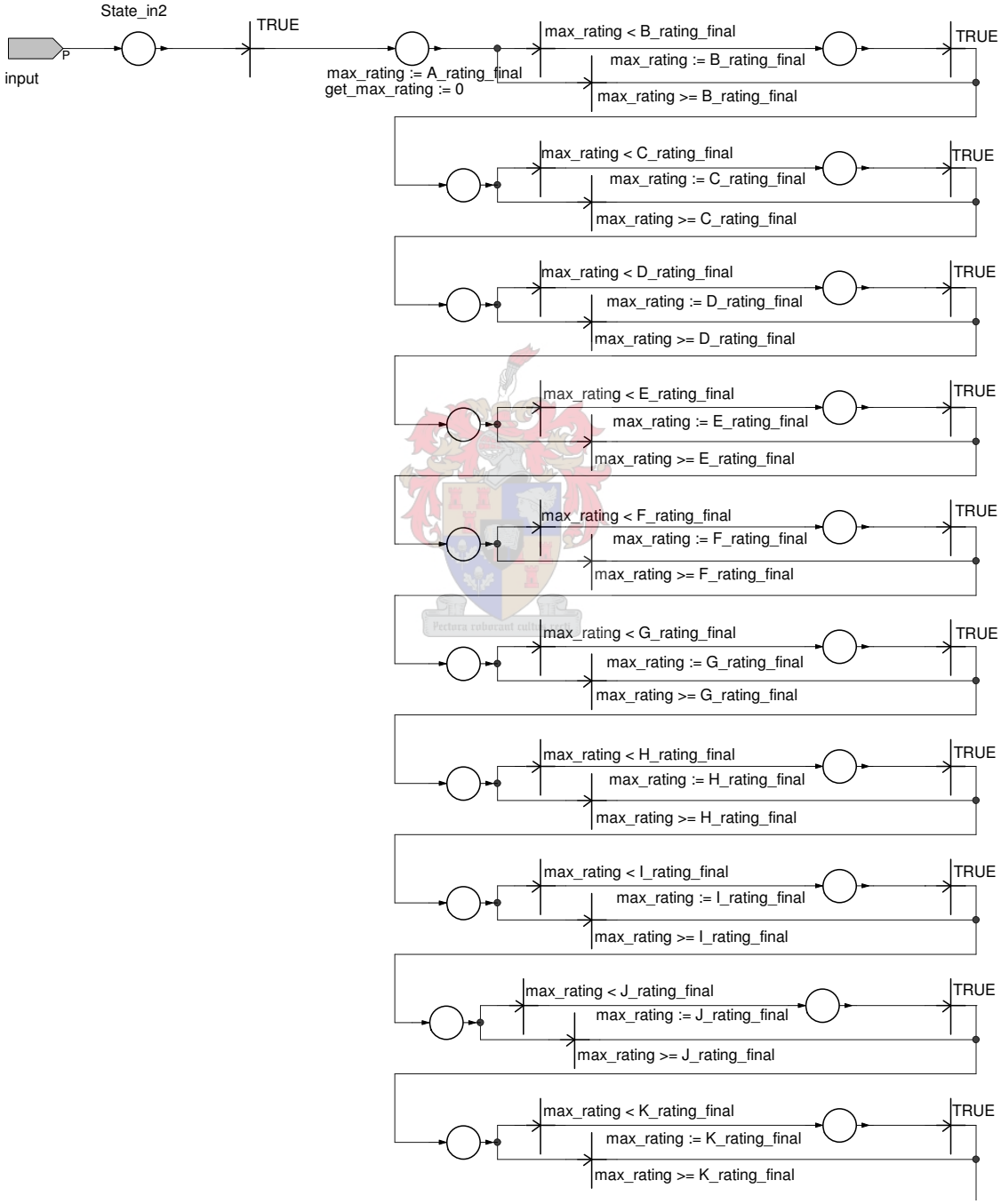

TRUE
w_A_rating := w_A_c1*w_p1 + w_A_c2*w_p2 + w_A_c3*w_p3 + w_A_c4*w_p4
w_B_rating := w_B_c1*w_p1 + w_B_c2*w_p2 + w_B_c3*w_p3 + w_B_c4*w_p4
w_C_rating := w_C_c1*w_p1 + w_C_c2*w_p2 + w_C_c3*w_p3 + w_C_c4*w_p4
w_D_rating := w_D_c1*w_p1 + w_D_c2*w_p2 + w_D_c3*w_p3 + w_D_c4*w_p4
w_E_rating := w_E_c1*w_p1 + w_E_c2*w_p2 + w_E_c3*w_p3 + w_E_c4*w_p4
w_F_rating := w_F_c1*w_p1 + w_F_c2*w_p2 + w_F_c3*w_p3 + w_F_c4*w_p4
w_G_rating := w_G_c1*w_p1 + w_G_c2*w_p2 + w_G_c3*w_p3 + w_G_c4*w_p4
w_H_rating := w_H_c1*w_p1 + w_H_c2*w_p2 + w_H_c3*w_p3 + w_H_c4*w_p4
w_I_rating := w_I_c1*w_p1 + w_I_c2*w_p2 + w_I_c3*w_p3 + w_I_c4*w_p4
w_J_rating := w_J_c1*w_p1 + w_J_c2*w_p2 + w_J_c3*w_p3 + w_J_c4*w_p4
w_K_rating := w_K_c1*w_p1 + w_K_c2*w_p2 + w_K_c3*w_p3 + w_K_c4*w_p4
w_L_rating := w_L_c1*w_p1 + w_L_c2*w_p2 + w_L_c3*w_p3 + w_L_c4*w_p4
w_M_rating := w_M_c1*w_p1 + w_M_c2*w_p2 + w_M_c3*w_p3 + w_M_c4*w_p4
w_Q_rating := w_Q_c1*w_p1 + w_Q_c2*w_p2 + w_Q_c3*w_p3 + w_Q_c4*w_p4
w_R_rating := w_R_c1*w_p1 + w_R_c2*w_p2 + w_R_c3*w_p3 + w_R_c4*w_p4
w_P_rating := 24
w_S_rating := w_S_c1*w_p1 + w_S_c2*w_p2 + w_S_c3*w_p3 + w_S_c4*w_p4
w_T_rating := w_T_c1*w_p1 + w_T_c2*w_p2 + w_T_c3*w_p3 + w_T_c4*w_p4
w_U_rating := w_U_c1*w_p1 + w_U_c2*w_p2 + w_U_c3*w_p3 + w_U_c4*w_p4
w_V_rating := w_V_c1*w_p1 + w_V_c2*w_p2 + w_V_c3*w_p3 + w_V_c4*w_p4
w_W_rating := w_W_c1*w_p1 + w_W_c2*w_p2 + w_W_c3*w_p3 + w_W_c4*w_p4
w_X_rating := w_X_c1*w_p1 + w_X_c2*w_p2 + w_X_c3*w_p3 + w_X_c4*w_p4
w_Y_rating := w_Y_c1*w_p1 + w_Y_c2*w_p2 + w_Y_c3*w_p3 + w_Y_c4*w_p4
w_Z_rating := w_Z_c1*w_p1 + w_Z_c2*w_p2 + w_Z_c3*w_p3 + w_Z_c4*w_p4
w_AA_rating := w_AA_c1*w_p1 + w_AA_c2*w_p2 + w_AA_c3*w_p3 + w_AA_c4*w_p4
w_AB_rating := w_AB_c1*w_p1 + w_AB_c2*w_p2 + w_AB_c3*w_p3 + w_AB_c4*w_p4
w_AC_rating := w_AC_c1*w_p1 + w_AC_c2*w_p2 + w_AC_c3*w_p3 + w_AC_c4*w_p4
w_AD_rating := w_AD_c1*w_p1 + w_AD_c2*w_p2 + w_AD_c3*w_p3 + w_AD_c4*w_p4
w_AE_rating := w_AE_c1*w_p1 + w_AE_c2*w_p2 + w_AE_c3*w_p3 + w_AE_c4*w_p4
w_AF_rating := w_AF_c1*w_p1 + w_AF_c2*w_p2 + w_AF_c3*w_p3 + w_AF_c4*w_p4
w_AG_rating := w_AG_c1*w_p1 + w_AG_c2*w_p2 + w_AG_c3*w_p3 + w_AG_c4*w_p4
w_N_rating := 24
TRUE
A_rating_final1 := (r_A_rating + w_A_rating) * (A_level = current_level)
B_rating_final1 := (r_B_rating + w_B_rating) * (B_level = current_level)
C_rating_final1 := (r_C_rating + w_C_rating) * (C_level = current_level)
D_rating_final1 := (r_D_rating +w_D_rating) * (D_level = current_level)
E_rating_final1 := (r_E_rating + w_E_rating) * (E_level = current_level)
F_rating_final1 := (r_F_rating + w_F_rating) * (F_level = current_level)
G_rating_final1 := (r_G_rating + w_G_rating) * (G_level = current_level)
H_rating_final1 := (r_H_rating + w_H_rating) * (H_level = current_level)
I_rating_final1 := (r_I_rating + w_I_rating) * (I_level = current_level)
J_rating_final1 := (r_J_rating + w_J_rating) * (J_level = current_level)
K_rating_final1 := (r_K_rating + w_K_rating) * (K_level = current_level)
L_rating_final1 := (r_L_rating + w_L_rating) * (L_level = current_level)
M_rating_final1 := (r_M_rating + w_M_rating) * (M_level = current_level)
Q_rating_final1 := (r_Q_rating + w_Q_rating) * (Q_level = current_level)
P_rating_final1 := (r_P_rating + w_P_rating) * (P_level = current_level)
N_rating_final1 := (r_N_rating + w_N_rating) * (N_level = current_level)
R_rating_final1 := (r_R_rating + w_R_rating) * (R_level = current_level)
S_rating_final1 := (r_S_rating + w_S_rating) * (S_level = current_level)
T_rating_final1 := (r_T_rating + w_T_rating) * (T_level = current_level)
U_rating_final1 := (r_U_rating + w_U_rating) * (U_level = current_level)
V_rating_final1 := (r_V_rating + w_V_rating) * (V_level = current_level)
W_rating_final1 := (r_W_rating + w_W_rating) * (W_level = current_level)
X_rating_final1 := (r_X_rating + w_X_rating) * (X_level = current_level)
Y_rating_final1 := (r_Y_rating + w_Y_rating) * (Y_level = current_level)
Z_rating_final1 := (r_Z_rating + w_Z_rating) * (Z_level = current_level)
AA_rating_final1 := (r_AA_rating + w_AA_rating) * (AA_level = current_level)
AB_rating_final1 := (r_AB_rating + w_AB_rating) * (AB_level = current_level)
AC_rating_final1 := (r_AC_rating + w_AC_rating) * (AC_level = current_level)
AD_rating_final1 := (r_AD_rating + w_AD_rating) * (AD_level = current_level)
AE_rating_final1 := (r_AE_rating + w_AE_rating) * (AE_level = current_level)
AF_rating_final1 := (r_AF_rating + w_AF_rating) * (AF_level = current_level)
AG_rating_final1 := (r_AG_rating + w_AG_rating) * (AG_level = current_level)
TRUE
A_rating_final := A_rating_final1 * ((A_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
B_rating_final := B_rating_final1 * ((B_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
C_rating_final := C_rating_final1 * ((C_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
D_rating_final := D_rating_final1 * ((D_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
E_rating_final := E_rating_final1 * ((E_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
F_rating_final := F_rating_final1 * ((F_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
G_rating_final := G_rating_final1 * ((G_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
H_rating_final := H_rating_final1 * ((H_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
I_rating_final := I_rating_final1 * ((I_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
J_rating_final := J_rating_final1 * ((J_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
K_rating_final := K_rating_final1 * ((K_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
L_rating_final := L_rating_final1 * ((L_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
M_rating_final := M_rating_final1 * ((M_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
Q_rating_final := Q_rating_final1 * ((Q_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
P_rating_final := P_rating_final1 * ((P_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
N_rating_final := N_rating_final1 * ((N_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
get_max_rating := 1
R_rating_final := R_rating_final1 * ((R_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
S_rating_final := S_rating_final1 * ((S_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
T_rating_final := T_rating_final1 * ((T_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
U_rating_final := U_rating_final1 * ((U_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
V_rating_final := V_rating_final1 * ((V_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
W_rating_final := W_rating_final1 * ((W_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
X_rating_final := X_rating_final1 * ((X_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
Y_rating_final := Y_rating_final1 * ((Y_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
Z_rating_final := Z_rating_final1 * ((Z_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
AA_rating_final := AA_rating_final1 * ((AA_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
AB_rating_final := AB_rating_final1 * ((AB_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
AC_rating_final := AC_rating_final1 * ((AC_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
AD_rating_final := AD_rating_final1 * ((AD_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
AE_rating_final := AE_rating_final1 * ((AE_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
AF_rating_final := AF_rating_final1 * ((AF_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))
AG_rating_final := AG_rating_final1 * ((AG_sw_ch = sw_changes_allowed) * (critical = 0) + (critical = 1))

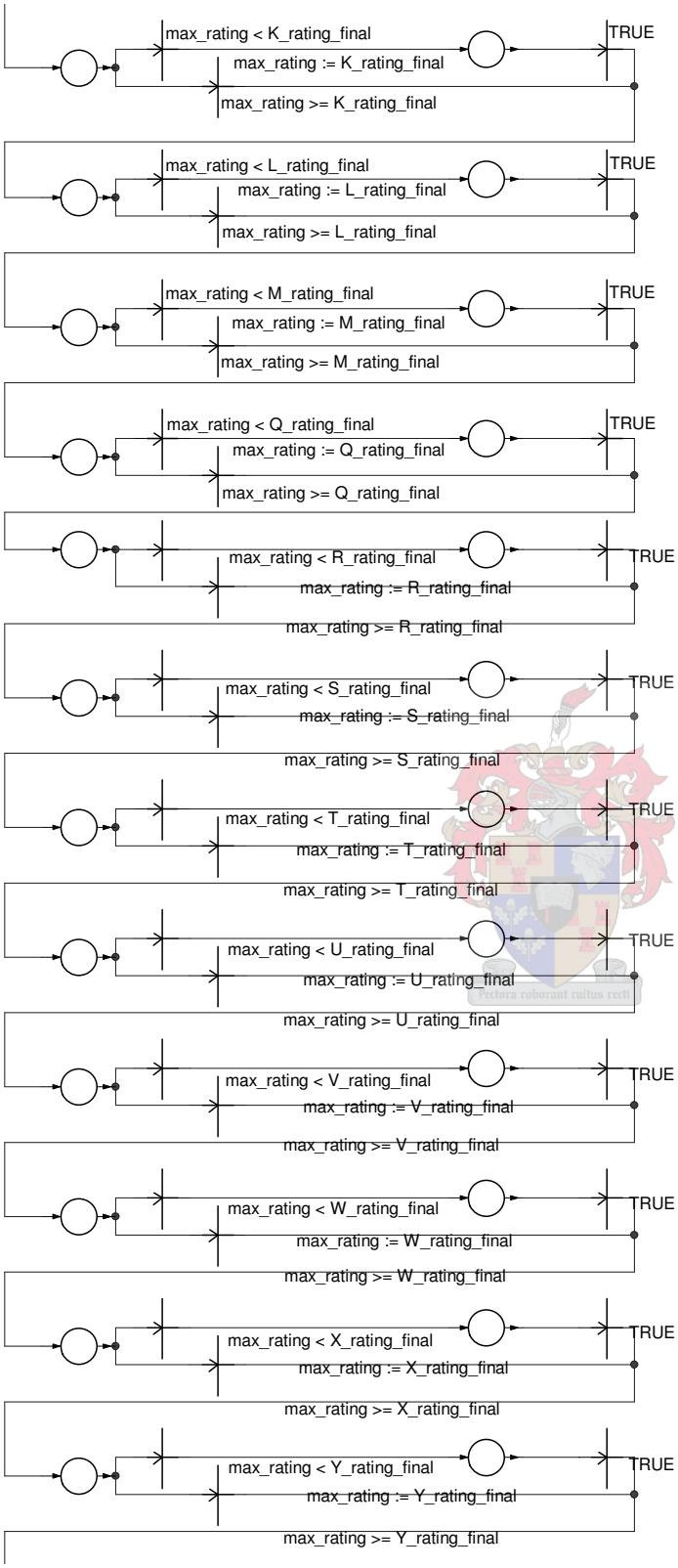

condition := update_sw_fn =1
update_sw_fn := 0
SET := ready := 1
Ostate_A
s5t := A5
s4t := A4
s3t := A3
s2t := A2
s1t := A1
TRUE
Ostate_B
Ostate_C
s4t := C4
s3t := C3
s2t := C2
s1t := C1
Ostate_D
s5t := D5
s4t := D4
s3t := D3
s2t := D2
s1t := D1
Ostate_E
s4t := E4
s3t := E3
s2t := E2
s1t := E1
Ostate_F
s4t := F4
s3t := F3
s2t := F2
s1t := F1
Ostate_G
s4t := G4
s3t := G3
s2t := G2
s1t := G1
Ostate_H
s4t := H4
s3t := H3
s2t := H2
s1t := H1
Ostate_I
s4t := I4
s3t := I3
s2t := I2
s1t := I1
Ostate_J
s4t := J4
s3t := J3
s2t := J2
s1t := J1
Ostate_K
s4t := K4
s3t := K3
s2t := K2
s1t := K1
Ostate_L
s4t := L4
s3t := L3
s2t := L2
s1t := L1
Ostate_M
s4t := M4
s3t := M3
s2t := M2
s1t := M1
Ostate_Q
s4t := Q4
s3t := Q3
s2t := Q2
s1t := Q1
Ostate_R
s4t := R4
s3t := R3
s2t := R2
s1t := R1
Ostate_S
s4t := S4
s3t := S3
s2t := S2
s1t := S1
Ostate_T
s4t := T4
s3t := T3
s2t := T2
s1t := T1
Ostate_U
s4t := U4
s3t := U3
s2t := U2
s1t := U1
Ostate_V
s4t := V4
s3t := V3
s2t := V2
s1t := V1
Ostate_W
s4t := W4
s3t := W3
s2t := W2
s1t := W1
Ostate_X
s4t := X4
s3t := X3
s2t := X2
s1t := X1
Ostate_Y
s4t := Y4
s3t := Y3
s2t := Y2
s1t := Y1
Ostate_Z
s4t := Z4
s3t := Z3
s2t := Z2
s1t := Z1
Ostate_AA
s4t := AA4
s3t := AA3
s2t := AA2
s1t := AA1
Ostate_AB
s4t := AB4
s3t := AB3
s2t := AB2
s1t := AB1
Ostate_AC
s4t := AC4
s3t := AC3
s2t := AC2
s1t := AC1
Ostate_AD
s4t := AD4
s3t := AD3
s2t := AD2
s1t := AD1
Ostate_AE
s4t := AE4
s3t := AE3
s2t := AE2
s1t := AE1
Ostate_AF
s4t := AF4
s3t := AF3
s2t := AF2
s1t := AF1
Ostate_AG
s4t := AG4
s3t := AG3
s2t := AG2
s1t := AG1
state_P
s4t := P4
s3t := P3
s2t := P2
s1t := P1
state_N
s4t := N4
s3t := N3
s2t := N2
s1t := N1

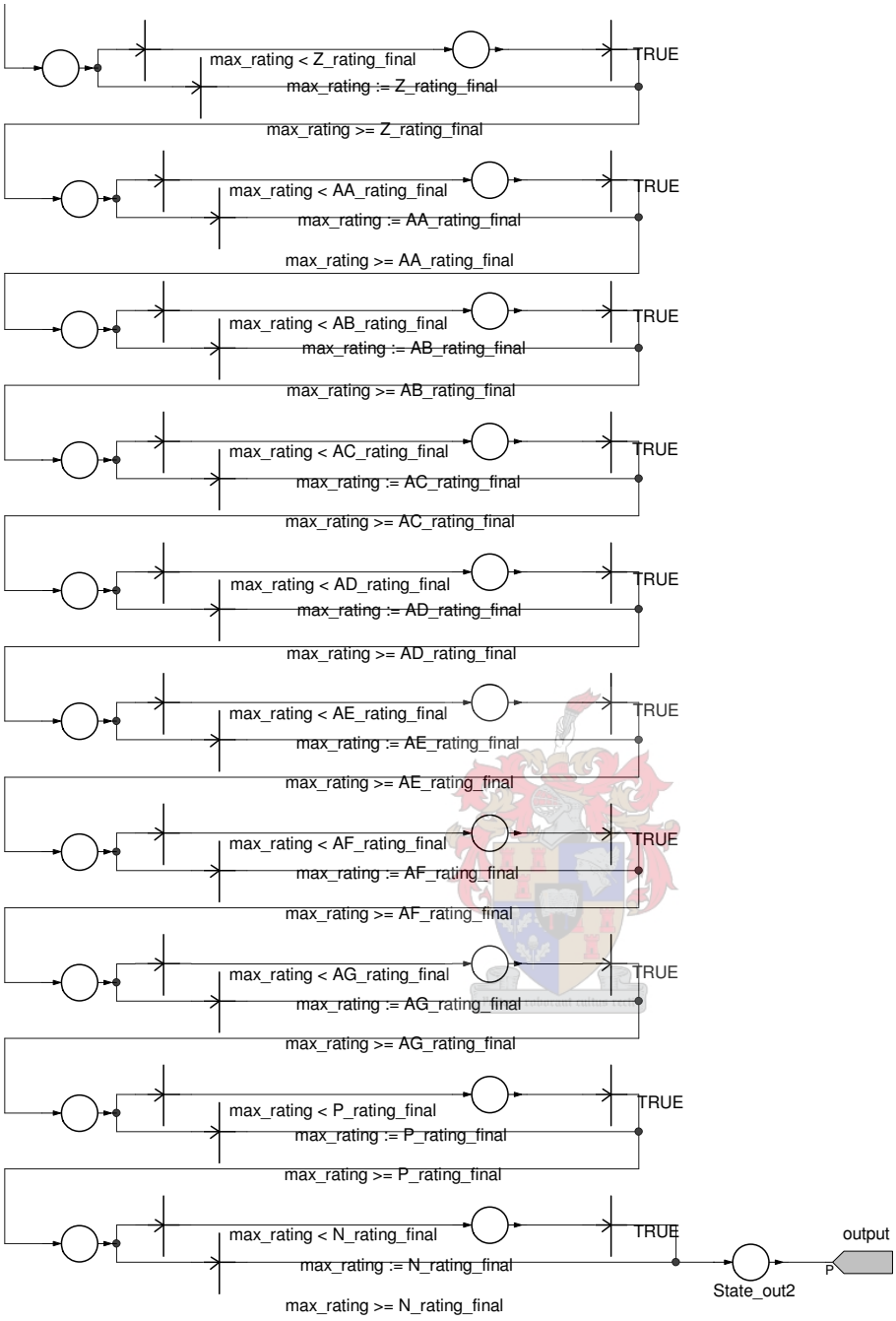

condition := cell1_c OR cell2_c OR cell3_c OR cell4_c OR cell5_
cell1_c OR cell2_c OR cell3_c OR cell4_c OR cell5_
SET := s1t := s4t_temp
SET := s2t := s3t_temp
SET := s3t := s2t_temp
SET := s4t := s1t_temp
SET := s5t := s5t_temp
cell1_c := 0
cell2_c := 0
cell3_c := 0
cell4_c := 0
cell5_c := 0
SET := startup := 0
SET := s1t := s1t_temp
SET := s2t := s2t_temp
SET := s3t := s3t_temp
SET := s4t := s4t_temp
SET := s5t := s5t_temp
SET := cell1_c := 0
SET := cell2_c := 0
SET := cell3_c := 0
SET := cell4_c := 0
SET := cell5_c := 0

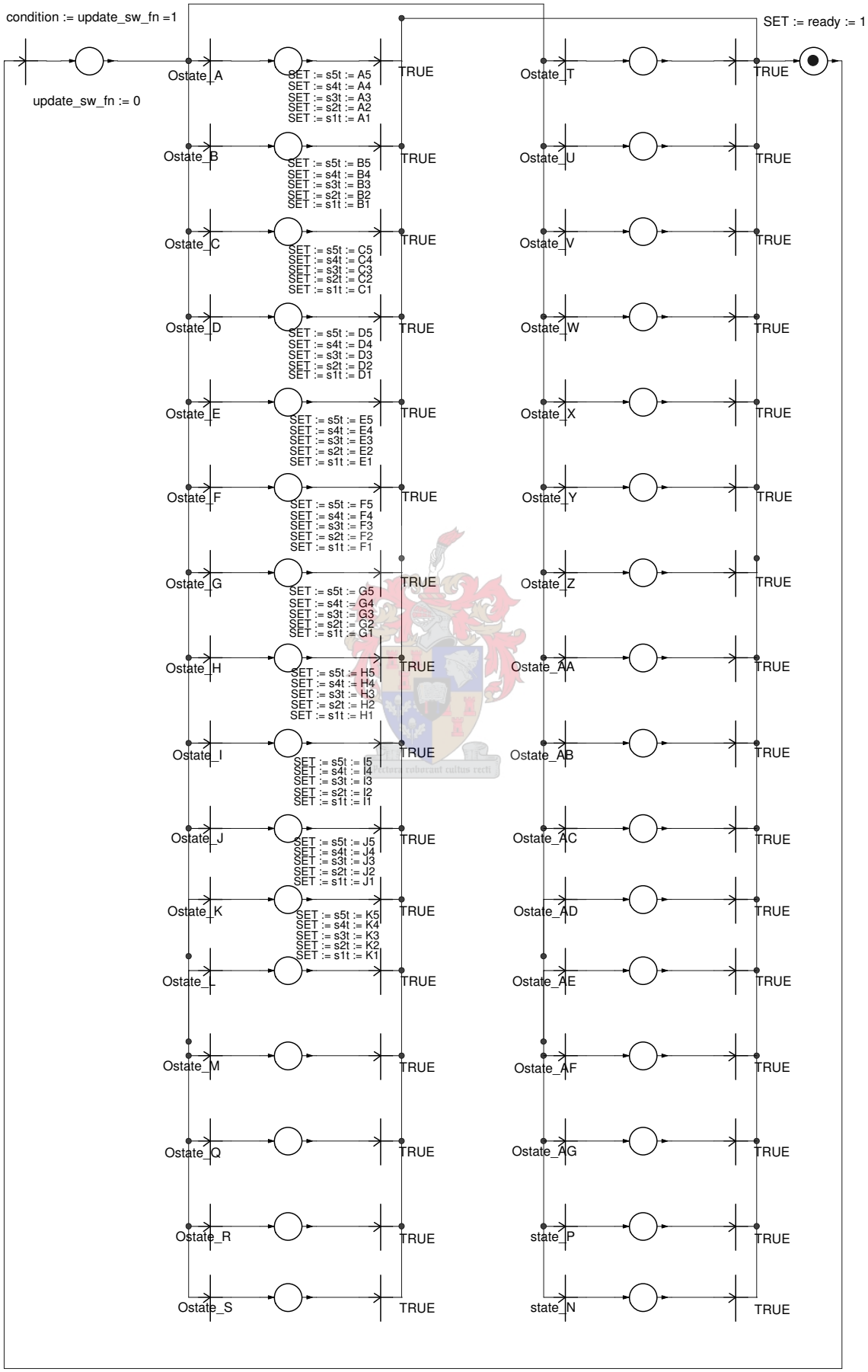

choose_Ostate = 1
TRUE
update_sw_fn := 1
choose_Ostate := 0
Ostate_A := (max_rating = A_rating_final)
Ostate_B := (max_rating = B_rating_final)
Ostate_C := (max_rating = C_rating_final)
Ostate_D := (max_rating = D_rating_final)
Ostate_E := (max_rating = E_rating_final)
Ostate_F := (max_rating = F_rating_final)
Ostate_G := (max_rating = G_rating_final)
Ostate_H := (max_rating = H_rating_final)
Ostate_I := (max_rating = I_rating_final)
Ostate_J := (max_rating = J_rating_final)
Ostate_K := (max_rating = K_rating_final)
Ostate_L := (max_rating = L_rating_final)
Ostate_M := (max_rating = M_rating_final)
Ostate_Q := (max_rating = Q_rating_final)
Ostate_R := (max_rating = R_rating_final)
Ostate_S := (max_rating = S_rating_final)
Ostate_T := (max_rating = T_rating_final)
Ostate_U := (max_rating = U_rating_final)
Ostate_V := (max_rating = V_rating_final)
Ostate_W := (max_rating = W_rating_final)
Ostate_X := (max_rating = X_rating_final)
Ostate_Y := (max_rating = Y_rating_final)
Ostate_Z := (max_rating = Z_rating_final)
Ostate_AA := (max_rating = AA_rating_final)
Ostate_AB := (max_rating = AB_rating_final)
Ostate_AC := (max_rating = AC_rating_final)
Ostate_AD := (max_rating = AD_rating_final)
Ostate_AE := (max_rating = AE_rating_final)
Ostate_AF := (max_rating = AF_rating_final)
Ostate_AG := (max_rating = AG_rating_final)
state_P := (max_rating = P_rating_final)
state_N := (max_rating = N_rating_final)


Sheet_get_max_rating
max_rating := max_rating
get_max_rating = 1
TRUE
State_in2
input
output
TRUE
TRUE
get_max_rating := 0
State_out2
choose_Ostate := 1

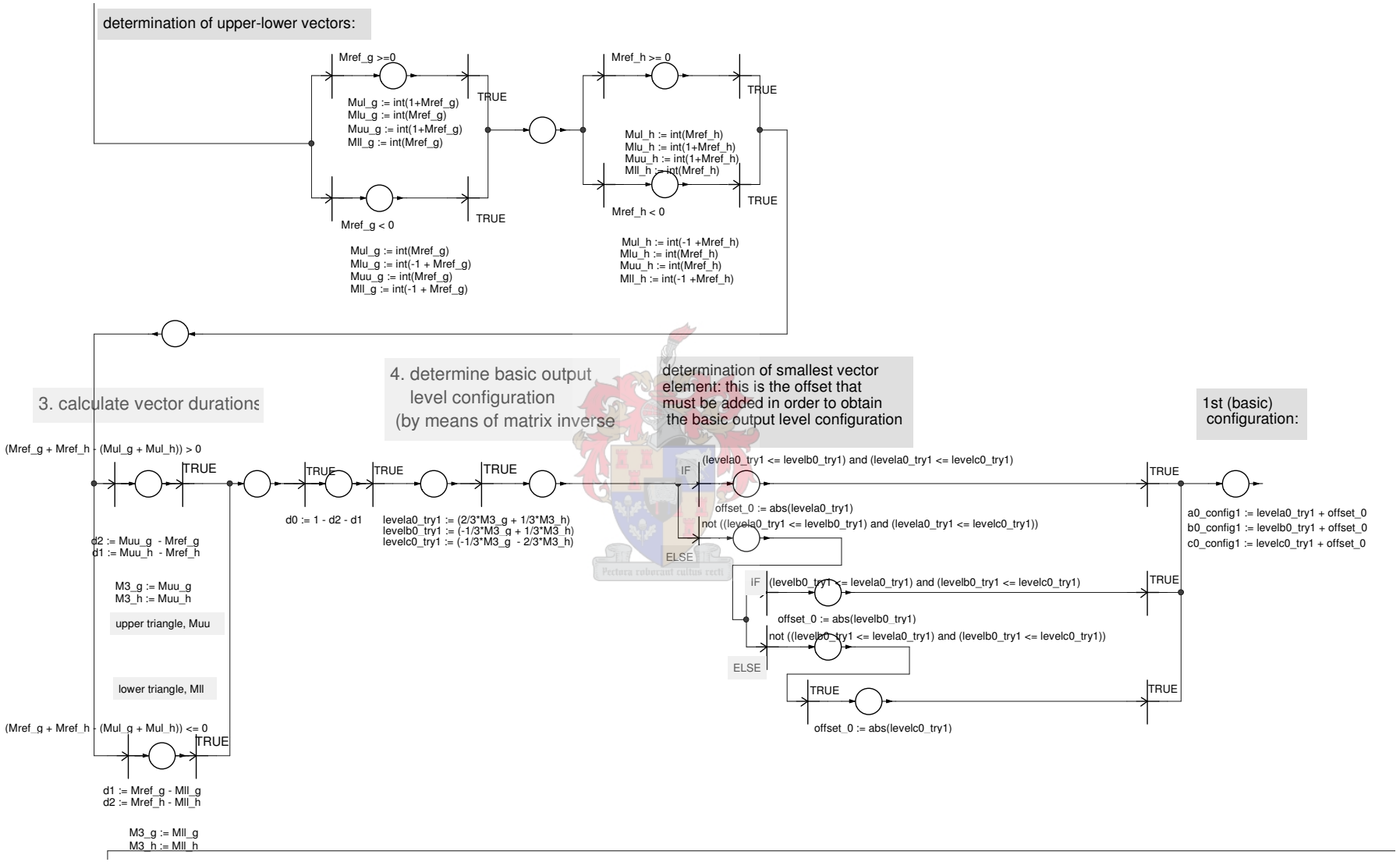

State_in2
TRUE
P
input
max_rating := A_rating_final
get_max_rating := 0
max_rating < B_rating_final
max_rating := B_rating_final
max_rating >= B_rating_final
max_rating < C_rating_final
max_rating := C_rating_final
max_rating >= C_rating_final
max_rating < D_rating_final
max_rating := D_rating_final
max_rating >= D_rating_final
max_rating < E_rating_final
max_rating := E_rating_final
max_rating >= E_rating_final
max_rating < F_rating_final
max_rating := F_rating_final
max_rating >= F_rating_final
max_rating < G_rating_final
max_rating := G_rating_final
max_rating >= G_rating_final
max_rating < H_rating_final
max_rating := H_rating_final
max_rating >= H_rating_final
max_rating < I_rating_final
max_rating := I_rating_final
max_rating >= I_rating_final
max_rating < J_rating_final
max_rating := J_rating_final
max_rating >= J_rating_final
max_rating < K_rating_final
max_rating := K_rating_final
max_rating >= K_rating_final
TRUE

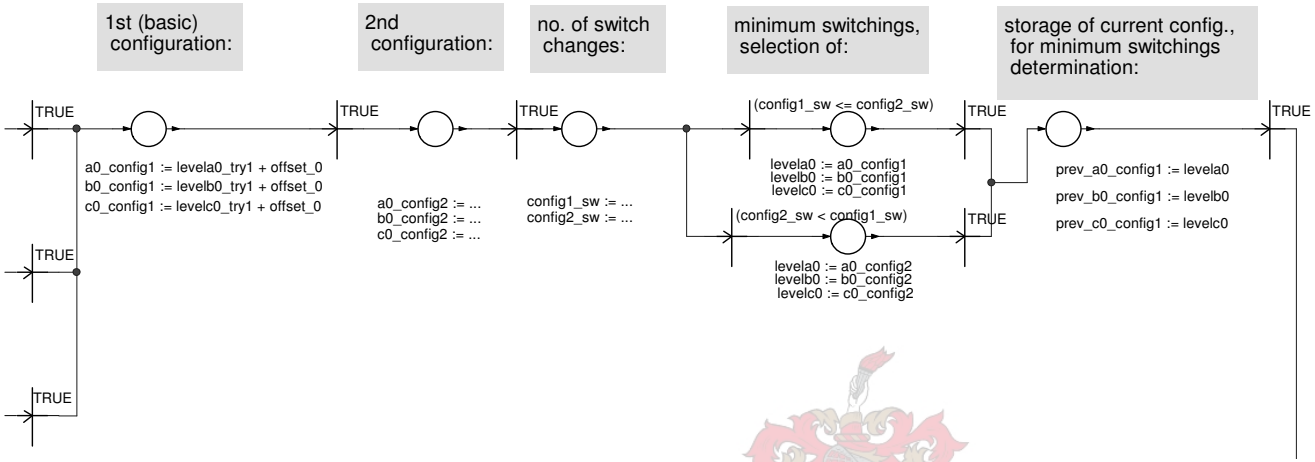

max_rating < K_rating_final
TRUE
max_rating := K_rating_final
max_rating >= K_rating_final
max_rating < L_rating_final
TRUE
max_rating := L_rating_final
max_rating >= L_rating_final
max_rating < M_rating_final
TRUE
max_rating := M_rating_final
max_rating >= M_rating_final
max_rating < Q_rating_final
TRUE
max_rating := Q_rating_final
max_rating >= Q_rating_final
max_rating < R_rating_final
TRUE
max_rating := R_rating_final
max_rating >= R_rating_final
max_rating < S_rating_final
TRUE
max_rating := S_rating_final
max_rating >= S_rating_final
max_rating < T_rating_final
TRUE
max_rating := T_rating_final
max_rating >= T_rating_final
max_rating < U_rating_final
TRUE
max_rating := U_rating_final
max_rating >= U_rating_final
max_rating < V_rating_final
TRUE
max_rating := V_rating_final
max_rating >= V_rating_final
max_rating < W_rating_final
TRUE
max_rating := W_rating_final
max_rating >= W_rating_final
max_rating < X_rating_final
TRUE
max_rating := X_rating_final
max_rating >= X_rating_final
max_rating < Y_rating_final
TRUE
max_rating := Y_rating_final
max_rating >= Y_rating_final

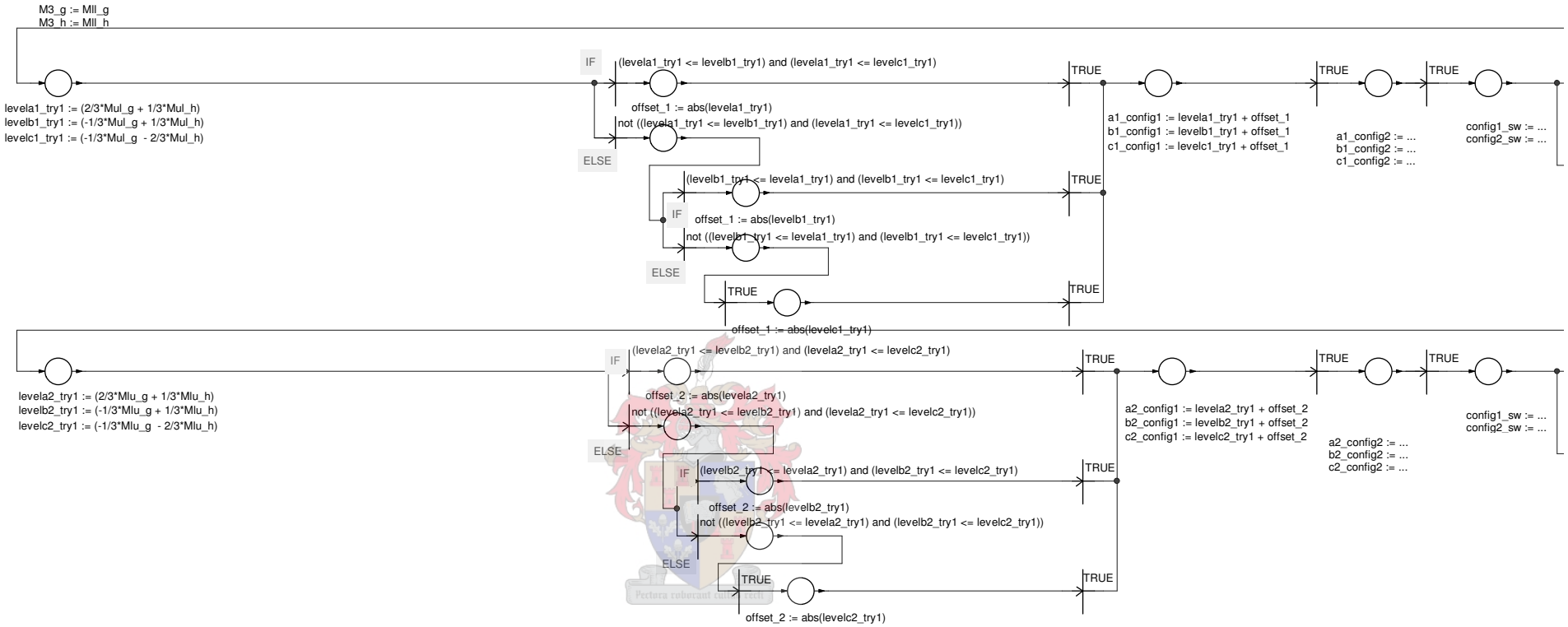

max_rating < Z_rating_final
TRUE
max_rating := Z_rating_final
max_rating >= Z_rating_final
max_rating < AA_rating_final
TRUE
max_rating := AA_rating_final
max_rating >= AA_rating_final
max_rating < AB_rating_final
TRUE
max_rating := AB_rating_final
max_rating >= AB_rating_final
max_rating < AC_rating_final
TRUE
max_rating := AC_rating_final
max_rating >= AC_rating_final
max_rating < AD_rating_final
TRUE
max_rating := AD_rating_final
max_rating >= AD_rating_final
max_rating < AE_rating_final
TRUE
max_rating := AE_rating_final
max_rating >= AE_rating_final
max_rating < AF_rating_final
TRUE
max_rating := AF_rating_final
max_rating >= AF_rating_final
max_rating < AG_rating_final
TRUE
max_rating := AG_rating_final
max_rating >= AG_rating_final
max_rating < P_rating_final
TRUE
max_rating := P_rating_final
max_rating >= P_rating_final
max_rating < N_rating_final
TRUE
max_rating := N_rating_final
max_rating >= N_rating_final
output
P
State_out2

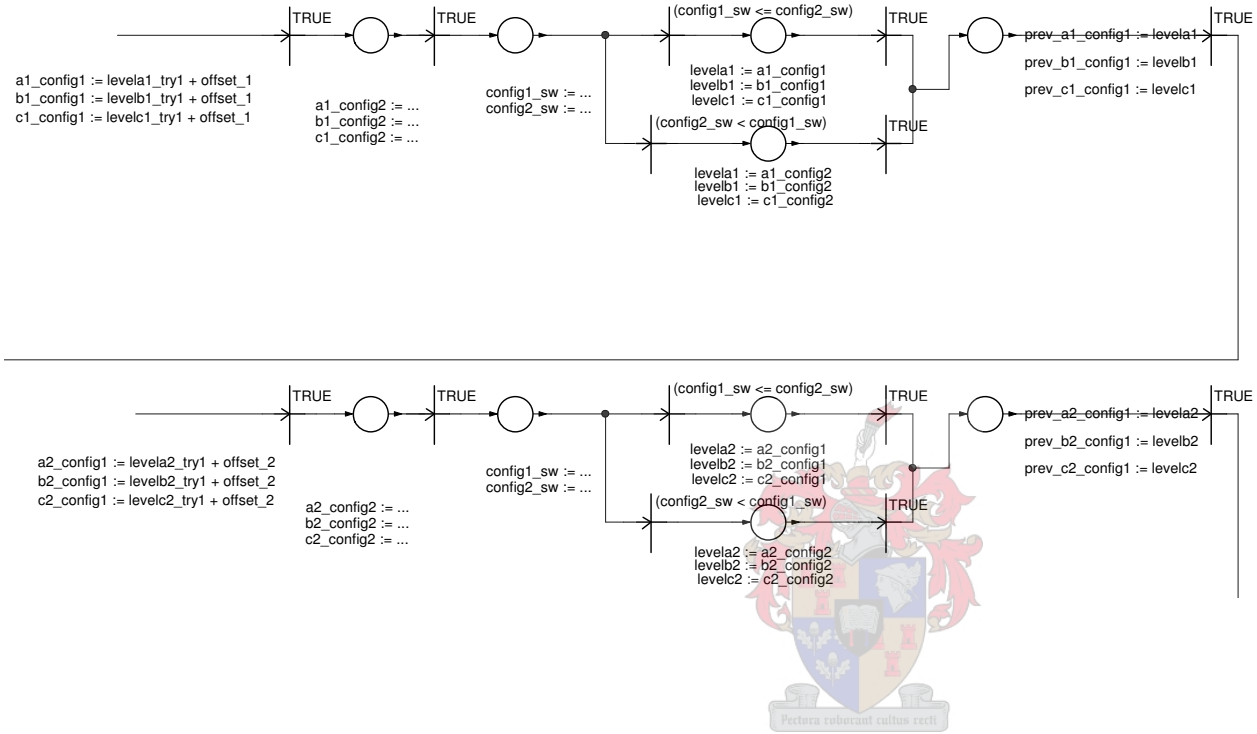

condition := update_sw_fn =1
update_sw_fn := 0
SET := ready := 1
Ostate_A
SET := s5t := A5
SET := s4t := A4
SET := s3t := A3
SET := s2t := A2
SET := s1t := A1
TRUE
Ostate_B
SET := s5t := B5
SET := s4t := B4
SET := s3t := B3
SET := s2t := B2
SET := s1t := B1
Ostate_C
SET := s5t := C5
SET := s4t := C4
SET := s3t := C3
SET := s2t := C2
SET := s1t := C1
Ostate_D
SET := s5t := D5
SET := s4t := D4
SET := s3t := D3
SET := s2t := D2
SET := s1t := D1
Ostate_E
SET := s5t := E5
SET := s4t := E4
SET := s3t := E3
SET := s2t := E2
SET := s1t := E1
Ostate_F
SET := s5t := F5
SET := s4t := F4
SET := s3t := F3
SET := s2t := F2
SET := s1t := F1
Ostate_G
SET := s5t := G5
SET := s4t := G4
SET := s3t := G3
SET := s2t := G2
SET := s1t := G1
Ostate_H
SET := s5t := H5
SET := s4t := H4
SET := s3t := H3
SET := s2t := H2
SET := s1t := H1
Ostate_I
SET := s5t := I5
SET := s4t := I4
SET := s3t := I3
SET := s2t := I2
SET := s1t := I1
Ostate_J
SET := s5t := J5
SET := s4t := J4
SET := s3t := J3
SET := s2t := J2
SET := s1t := J1
Ostate_K
SET := s5t := K5
SET := s4t := K4
SET := s3t := K3
SET := s2t := K2
SET := s1t := K1
Ostate_L
Ostate_M
Ostate_Q
Ostate_R
Ostate_S
Ostate_T
Ostate_U
Ostate_V
Ostate_W
Ostate_X
Ostate_Y
Ostate_Z
Ostate_AA
Ostate_AB
Ostate_AC
Ostate_AD
Ostate_AE
Ostate_AF
Ostate_AG
state_P
state_N

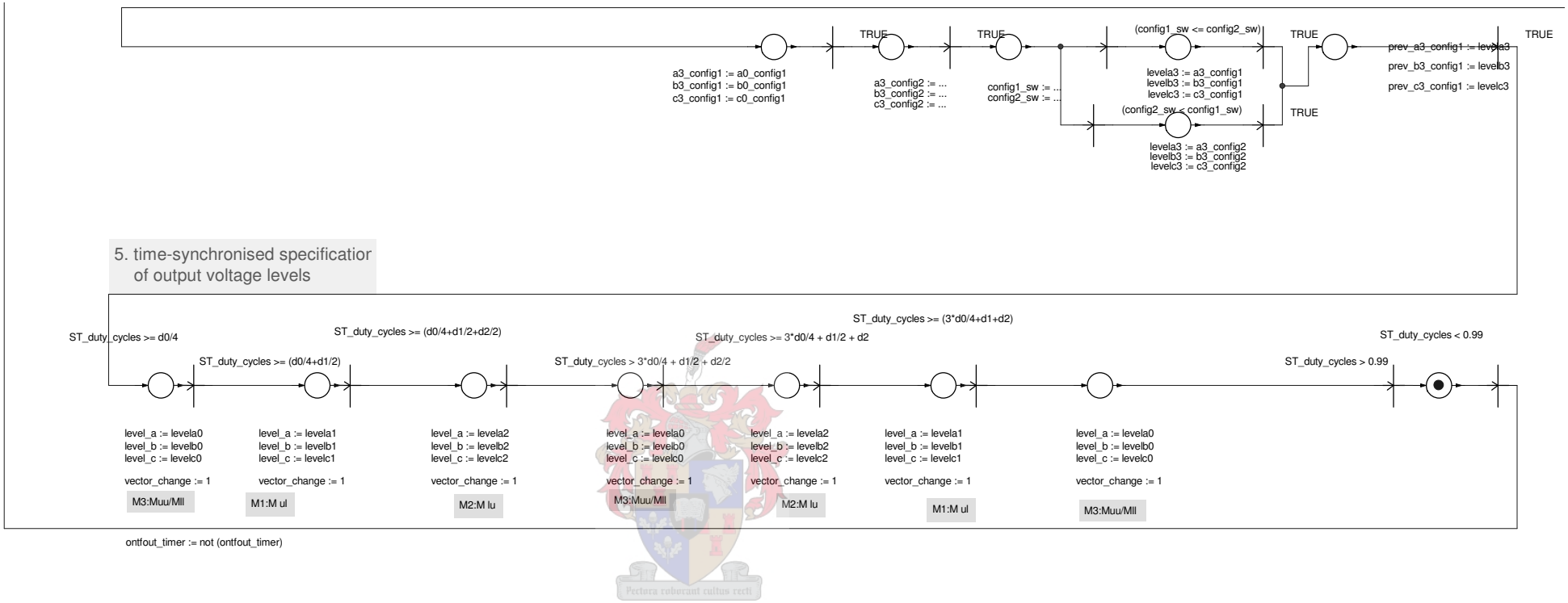
THREE-PHASE MODEL



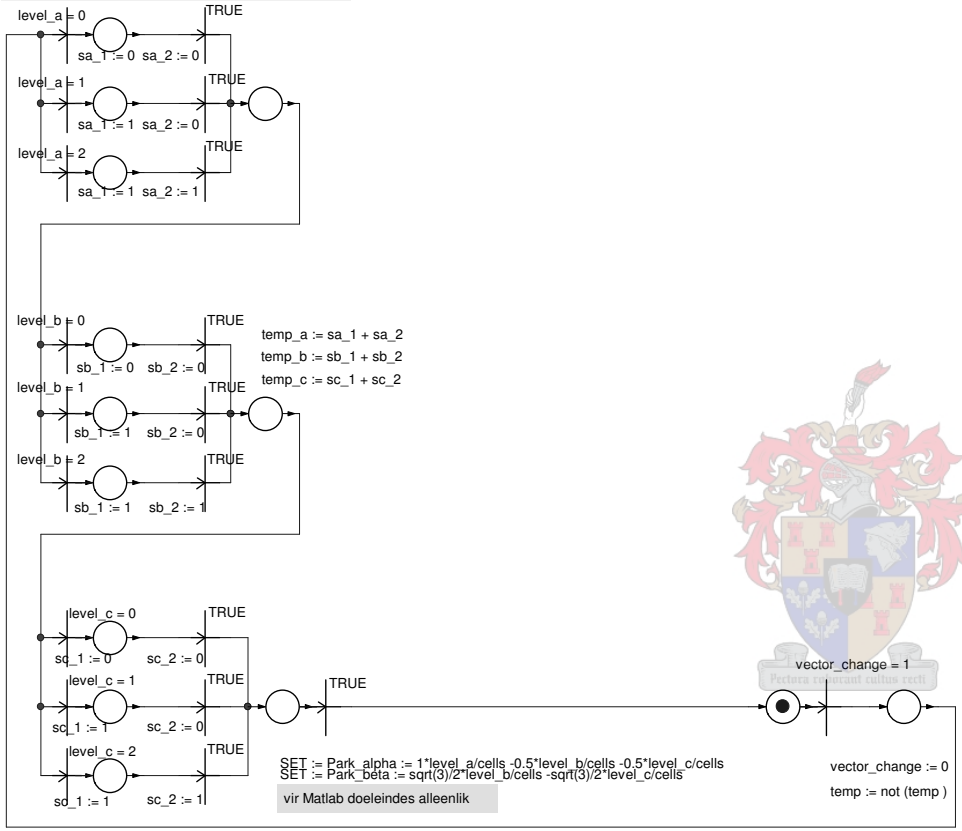

1st (basic) configuration:
2nd configuration:
no. of switch changes:
minimum switchings, selection of:
storage of current config., for minimum switchings determination:
TRUE
a0_config1 := levela0_try1 + offset_0
b0_config1 := levelb0_try1 + offset_0
c0_config1 := levelc0_try1 + offset_0
a0_config2 := ...
b0_config2 := ...
c0_config2 := ...
config1_sw := ...
config2_sw := ...
(config1_sw <= config2_sw)
levela0 := a0_config1
levelb0 := b0_config1
levelc0 := c0_config1
(config2_sw < config1_sw)
levela0 := a0_config2
levelb0 := b0_config2
levelc0 := c0_config2
prev_a0_config1 := levela0
prev_b0_config1 := levelb0
prev_c0_config1 := levelc0


M3_g := Mll_g
M3_h := Mll_h
levela1_try1 := (2/3*Mul_g + 1/3*Mul_h)
levelb1_try1 := (-1/3*Mul_g + 1/3*Mul_h)
levelc1_try1 := (-1/3*Mul_g - 2/3*Mul_h)
IF
(levela1_try1 <= levelb1_try1) and (levela1_try1 <= levelc1_try1)
TRUE
offset_1 := abs(levela1_try1)
not ((levela1_try1 <= levelb1_try1) and (levela1_try1 <= levelc1_try1))
ELSE
(levelb1_try1 <= levela1_try1) and (levelb1_try1 <= levelc1_try1)
IF
offset_1 := abs(levelb1_try1)
not ((levelb1_try1 <= levela1_try1) and (levelb1_try1 <= levelc1_try1))
ELSE
TRUE
offset_1 := abs(levelc1_try1)
a1_config1 := levela1_try1 + offset_1
b1_config1 := levelb1_try1 + offset_1
c1_config1 := levelc1_try1 + offset_1
a1_config2 := ...
b1_config2 := ...
c1_config2 := ...
config1_sw := ...
config2_sw := ...
levela2_try1 := (2/3*Mlu_g + 1/3*Mlu_h)
levelb2_try1 := (-1/3*Mlu_g + 1/3*Mlu_h)
levelc2_try1 := (-1/3*Mlu_g - 2/3*Mlu_h)
(levela2_try1 <= levelb2_try1) and (levela2_try1 <= levelc2_try1)
offset_2 := abs(levela2_try1)
not ((levela2_try1 <= levelb2_try1) and (levela2_try1 <= levelc2_try1))
(levelb2_try1 <= levela2_try1) and (levelb2_try1 <= levelc2_try1)
offset_2 := abs(levelb2_try1)
not ((levelb2_try1 <= levela2_try1) and (levelb2_try1 <= levelc2_try1))
offset_2 := abs(levelc2_try1)
a2_config1 := levela2_try1 + offset_2
b2_config1 := levelb2_try1 + offset_2
c2_config1 := levelc2_try1 + offset_2
a2_config2 := ...
b2_config2 := ...
c2_config2 := ...


TRUE
TRUE
(config1_sw <= config2_sw)
TRUE
prev_a1_config1 := levela1
prev_b1_config1 := levelb1
prev_c1_config1 := levelc1
TRUE
a1_config1 := levela1_try1 + offset_1
b1_config1 := levelb1_try1 + offset_1
c1_config1 := levelc1_try1 + offset_1
a1_config2 := ...
b1_config2 := ...
c1_config2 := ...
config1_sw := ...
config2_sw := ...
levela1 := a1_config1
levelb1 := b1_config1
levelc1 := c1_config1
(config2_sw < config1_sw)
TRUE
levela1 := a1_config2
levelb1 := b1_config2
levelc1 := c1_config2
TRUE
TRUE
(config1_sw <= config2_sw)
TRUE
prev_a2_config1 := levela2
prev_b2_config1 := levelb2
prev_c2_config1 := levelc2
TRUE
a2_config1 := levela2_try1 + offset_2
b2_config1 := levelb2_try1 + offset_2
c2_config1 := levelc2_try1 + offset_2
a2_config2 := ...
b2_config2 := ...
c2_config2 := ...
config1_sw := ...
config2_sw := ...
levela2 := a2_config1
levelb2 := b2_config1
levelc2 := c2_config1
(config2_sw < config1_sw)
TRUE
levela2 := a2_config2
levelb2 := b2_config2
levelc2 := c2_config2



6. switching function specification



Pectora roborant cultus recti