Deadlock-Freeness of Hexagonal Systolic Arrays

Stefan Gruner & Theuns Steyn
Department of Computer Science
University of Pretoria
South Africa
http://ssfm.cs.up.ac.za/

Abstract: With the re-emergence of parallel computation for technical applications in these days also the classical concept of systolic arrays is becoming important again. However, for the sake of their operational safety, the question of deadlock must be addressed. For this contribution we used the well-known Roscoe-Dathi method to demonstrate the deadlock-freeness of a systolic array with hexagonal connectivity. Our result implies that it is theoretically safe to deploy such arrays on various platforms. Our proof is valid for all cases in which the computational pattern (input-output-behaviour) of the array does not depend on the particular values (contents) of the communicated data.

1 Introduction

Systolic arrays are a useful paradigm of parallel computation, as they can be quite easily implemented in the commodity hardware of field-programmable gate arrays (FPGA); this has been done since the 1990s. Since the early works by Kung [6], many useful types of systolic arrays for various application purposes are known [9]. As always in parallel computation, the possibility of *deadlock* must be taken into account also in the paradigm of systolic arrays; this issue has been addressed already in [7].

In this letter we show the deadlock-freeness of a *hexagonal* systolic array (useful for numeric matrix multiplications) from [9] by application of Roscoe's and Dathi's *CSP*-based *analytic* proof method [10], (in contrast to the approach by model-checking presented in [2] [8] which is based on state space exploration [1]). This kind of proof is 'data-blind' (or 'data-independent' in the terminology of [2] [8]) and is thus valid for *all* similar hexagonal systolic arrays in which the dynamic communication patterns within the array do *not* depend on the *value* (contents) of the data communicated between the processor nodes in the array. We conclude that such hexagonal systolic arrays may thus be safely mapped onto FPGA hardware for a variety of applications: this deadlock-freeness result for data-independent hexagonal systolic arrays is (as far as we know) a new result.

For all details about how our proof was derived, see [11]. For more information about the motivation and research context of our result, see Section 6 (Future Work) below.

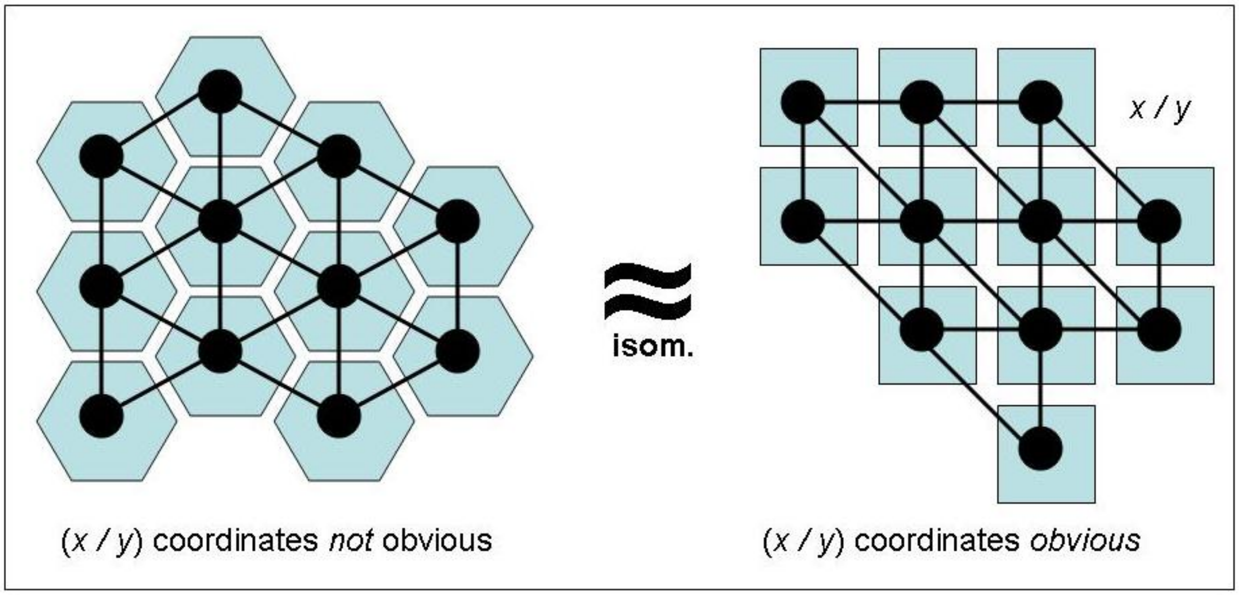

Figure 1: *Sketch of a Hexagonal Systolic Array*

2 Definition: Hexagonal Array Representation in CSP

In a hexagonal systolic array the processing elements are topologically arranged like in 'bee hives' and can communicate in three directions. In this arrangement it is not obvious that such an array can be expressed in terms of Cartesian coordinates, but there is indeed a graph isomorphism ($\approx$), as sketched in Figure 1, which makes it obvious that also hexagonal systolic arrays can be described in terms of Cartesian coordinates. In other words: a hexagonal systolic array is a rectangular systolic array with additional diagonal communication lines in one direction (*not* in both diagonal directions, otherwise it would be an octogonal array).

The lengthy CSP formula in the subsequent paragraph, which is used to formally specify the processes of each individual node in the hexagonal systolic array, is graphically illustrated in Figure 2 and has the following intuitive meaning:

- The process of every network cell is divided into two subsequent phases, P and Q, which are iterated until termination.
- Whereas P represents the *reading* phase (notation: "?"), in which the cell receives input (to process it), Q represents the *writing* phase (notation: "!"), in which the cell delivers its (processed) output.
- Input for the reading phase of a cell comes from its three North-Western neighbour cells, in any (arbitrary) order (in an 'interleaving' mode of concurrency), whenever the cell's *environment* makes input available.

- Output from the writing phase of the cell goes to its three South-Eastern neighour cell, also in any (arbitrary) order (in an 'interleaving' mode of concurrency); this output can then serve as input to those neighbour cells.
- The horizontal, vertical and diagonal *channels*, which carry those signals from cell to cell, are identified in the subsequent CSP formula by various variables $h_{i,j}$, $v_{i,j}$ and $d_{i,j}$.

In CSP [3], and on the basis of a 2-dimentional Cartesian coordinate system in (i,j) for the position of the processors P, our hexagonal systolic array chosen from [9] has thus the following representation:

$$P_{i,j}(a,b,c) = (\mathcal{X} \;\square\; \mathcal{Y} \;\square\; \mathcal{Z}),$$

whereby:

$$\begin{aligned}\mathcal{X} &:= (h_{i,j}?a \rightarrow (\mathcal{B} \;\square\; \mathcal{C})),\\ \mathcal{Y} &:= (v_{i,j}?b \rightarrow (\mathcal{A} \;\square\; \mathcal{C}')),\\ \mathcal{Z} &:= (d_{i,j}?c \rightarrow (\mathcal{A}' \;\square\; \mathcal{B}')),\end{aligned}$$

with:

$$\begin{aligned}\mathcal{B} &:= (v_{i,j}?b \rightarrow d_{i,j}?c \rightarrow Q_{i,j}(a,b,c)),\\ \mathcal{C} &:= (d_{i,j}?c \rightarrow v_{i,j}?b \rightarrow Q_{i,j}(a,b,c)),\\ \mathcal{A} &:= (h_{i,j}?a \rightarrow d_{i,j}?c \rightarrow Q_{i,j}(a,b,c)),\\ \mathcal{C}' &:= (d_{i,j}?c \rightarrow h_{i,j}?a \rightarrow Q_{i,j}(a,b,c)),\\ \mathcal{A}' &:= (h_{i,j}?a \rightarrow v_{i,j}?b \rightarrow Q_{i,j}(a,b,c)),\\ \mathcal{B}' &:= (v_{i,j}?b \rightarrow h_{i,j}?a \rightarrow Q_{i,j}(a,b,c)).\end{aligned}$$

After P-processes have thus *read* the incoming data *from* their neighbours, the Q-processes will *write* the (processed) data *to* other neighbours as follows:

$$Q_{i,j}(x,y,z) = ((\mathcal{X}' \;|||\; \mathcal{Y}' \;|||\; \mathcal{Z}') \;;\; P_{i,j}(a,b,c)),$$

whereby:

$$\begin{aligned}\mathcal{X}' &:= (h_{(i+1),j}!x \rightarrow \text{SKIP}),\\ \mathcal{Y}' &:= (v_{i,(j+1)}!y \rightarrow \text{SKIP}),\\ \mathcal{Z}' &:= (d_{(i+1),(j+1)}!(z+xy) \rightarrow \text{SKIP}).\end{aligned}$$

Note that a, b, c are the (numerical) *data* which are communicated through the channels between the processors; however the *contents* of those data is *irrelevant* for the *form* of communication between the processes.

To summarise the meaning of the formula of above: It describes the process executed in each (every) cell of the hexagonal systolic array. This process has two sub-processes: One for getting input from its three north-western neighbours (in any order), and one for writing output for its three south-eastern neighbours (in any order). In Figure 2 these events are abbreviated by the letters A, B, C (for reading) and D, E, F (for writing). Thereafter the process repeats itself.

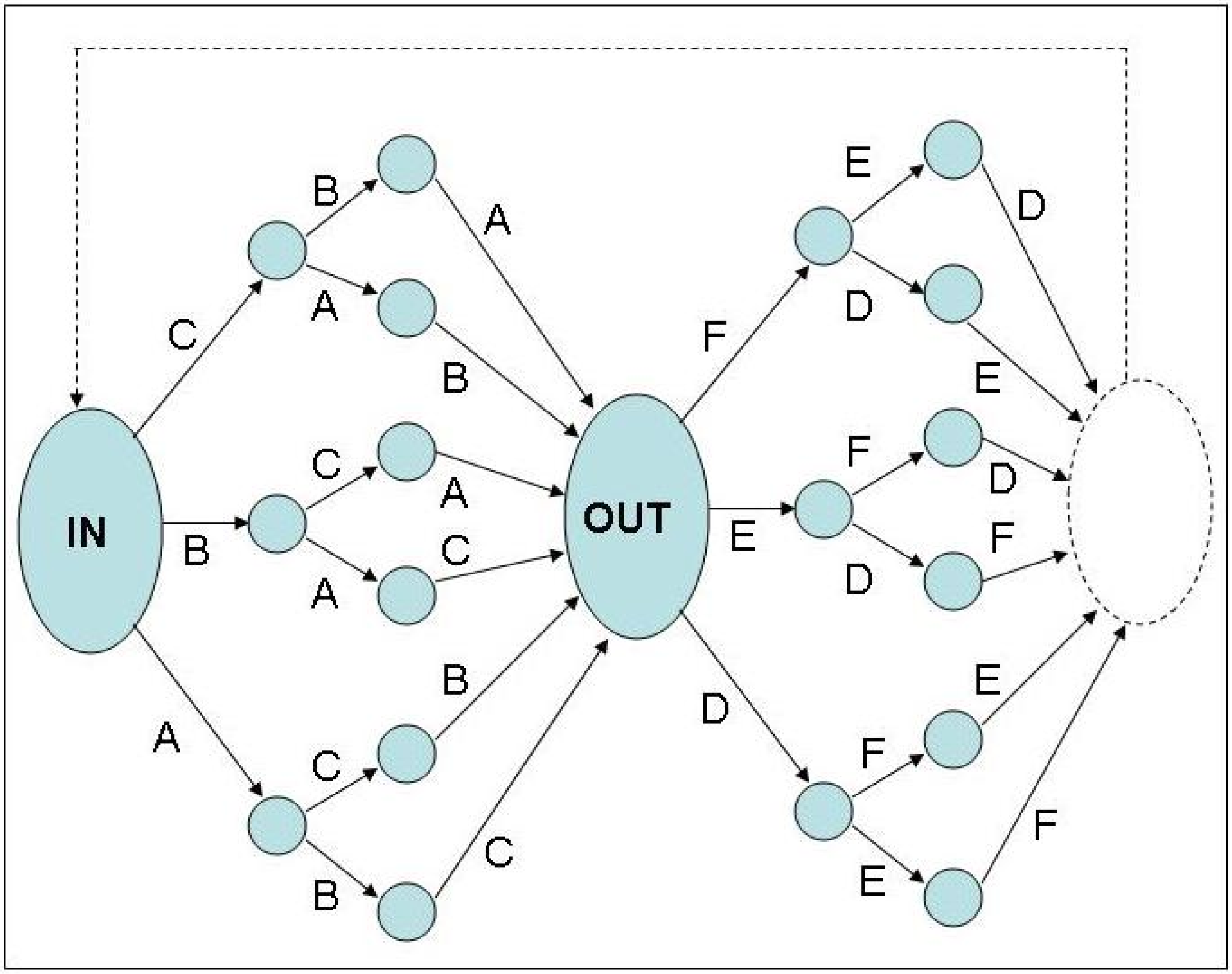

Figure 2: *Process executed by every Array Cell*

3 Method Applicability: Prerequisites

The Roscoe-Dathi-method can only be applied if a network, such as our hexagonal systolic array as specified in the previous section, fulfills the following conditions [10]:

- Each individual process within the network is deadlock-free by itself;
- The communication structure of the entire network is *triple-disjoint*.

In our case, the first condition is obviously fulfilled. As far as the second condition is concerned, it has been shown in [11] that

$$(\alpha A \ \cap \ \alpha A' \ \cap \ \alpha A'') = \emptyset$$

for all possible combinations of

$$A, A', A'' \in \{P_{i,j}, P_{(i+1),j}, P_{i,(j+1)}, P_{(i+1),(j+1)}\},$$

such that the Roscoe-Dathi method is indeed applicable to the case of our hexagonal systolic array. $\surd$

4 Process Communication Analysis

In the next phase of the proof, one has to analyse the *pairwise* communication behaviour of *any* two adjacent processes P and P': forward and backward, in all possible directions, horizontally, vertically as well as diagonally. Thus, six communicative constellations must be analysed: $(P \xrightarrow{h} P')$, $(P \xleftarrow{h} P')$, $(P \xrightarrow{v} P')$, $(P \xleftarrow{v} P')$, $(P \xrightarrow{d} P')$, $(P \xleftarrow{d} P')$. For each of these constellations, the sets of *ungranted requests* [10] must be found, which can be done by looking at the *state transition diagrams* $S(P)$ and $S(P')$ of the two processes under consideration, and by analysing the possible state pairs (s, s') with $s \in S(P)$ and $s' \in S(P')$. Analysis of the 'ungranted requests' is so crucial because they are the 'seeds' of deadlock in unfortunate circumstances (such as 'circular wait' situations).

If done 'manually', this is a lenghty and tedious procedure which starkly highlights the practical need for proof automation; see Section 6 (Future Work) below. For our hexagonal systolic array this lenghty proof procedure was fully carried out 'manually' and described in all details in [11]. In this short letter we can only present the main intermediate results to the final result and have to omit many minute steps inbetween.

4.1 Communication Case: $P_{i,j} \longrightarrow P_{(i+1),j}$

In the situations of 'ungranted requests' [10] after traces t in this case, and for its relevant communication event D, the following characteristic inequalities have found to be valid [11] about the according numbers of event observations.

- *Requests:* $|t \uparrow \alpha P_{i,j}| \geq 6 \cdot |t \uparrow \{D\}| + 3$,
- *Refusals:* $|t \uparrow \alpha P_{(i+1),j}| \leq 6 \cdot |t \uparrow \{D\}| - 1$,

whereby $|(t \uparrow \{D\})|$ indicates the process cycle number and $|(t \uparrow \alpha P_{i,j})|$ the number of steps which the two processes can make together until the relevant situation occurs [10]. Merging these two inequalities arithmetically leads eventually to the following characteristic description of this case as **Lemma 1**:

$$|t \uparrow \alpha P_{i,j}| > |t \uparrow \alpha P_{(i+1),j}| + 3. \ \surd$$

4.2 Communication Case: $P_{(i+1),j} \longrightarrow P_{i,j}$

For this case, following the same procedure [10] as above, the following inequalities have been found [11].

- *Requests:* $|t \uparrow \alpha P_{(i+1),j}| \geq 6 \cdot |t \uparrow \{D\}|$,
- *Refusals:* $|t \uparrow \alpha P_{i,j}| \leq 6 \cdot |t \uparrow \{D\}| + 2$,

Merging these two inequalities arithmetically we eventually yield **Lemma 2**:

$$|t \uparrow \alpha P_{(i+1),j}| > |t \uparrow \alpha P_{i,j}| - 3. \ \surd$$

4.3 Communication Case: $P_{i,j} \longrightarrow P_{i,(j+1)}$

Now we are dealing with a different communication event, E, which this constellation of processes has in common. For this case, following the same procedure [10] as above, the following inequalities have been found [11].

- *Requests:* $|t \uparrow \alpha P_{i,j}| \geq 6 \cdot |t \uparrow \{E\}| + 3$,
- *Refusals:* $|t \uparrow \alpha P_{i,(j+1)}| \leq 6 \cdot |t \uparrow \{E\}| - 1$,

Merging these two inequalities arithmetically we eventually yield **Lemma 3**:

$$|t \uparrow \alpha P_{i,j}| > |t \uparrow \alpha P_{i,(j+1)}| + 3. \ \surd$$

4.4 Communication Case: $P_{i,(j+1)} \longrightarrow P_{i,j}$

For this case, following the same procedure [10] as above, the following inequalities have been found [11].

- *Requests:* $|t \uparrow \alpha P_{i,(j+1)}| \geq 6 \cdot |t \uparrow \{E\}|$,
- *Refusals:* $|t \uparrow \alpha P_{i,j}| \leq 6 \cdot |t \uparrow \{E\}| + 2$,

Merging these two inequalities arithmetically we eventually yield **Lemma 4**:

$$|t \uparrow \alpha P_{i,(j+1)}| > |t \uparrow \alpha P_{i,j}| - 3. \ \surd$$

4.5 Communication Case: $P_{i,j} \longrightarrow P_{(i+1),(j+1)}$

For the communications in the diagonal directions of the array, F is now the channel event relevant for the analysis. For this case, following the same procedure [10] as above, the following inequalities have been found [11].

- *Requests:* $|t \uparrow \alpha P_{i,j}| \geq 6 \cdot |t \uparrow \{F\}| + 3$,
- *Refusals:* $|t \uparrow \alpha P_{(i+1),(j+1)}| \leq 6 \cdot |t \uparrow \{F\}| - 1$,

Merging these two inequalities arithmetically we eventually yield **Lemma 5**:

$$|t \uparrow \alpha P_{i,j}| > |t \uparrow \alpha P_{(i+1),(j+1)}| + 3. \ \surd$$

4.6 Communication Case: $P_{(i+1),(j+1)} \longrightarrow P_{i,j}$

For this case, following the same procedure [10] as above, the following inequalities have been found [11].

- *Requests:* $|t \uparrow \alpha P_{(i+1),(j+1)}| \geq 6 \cdot |t \uparrow \{F\}|$,
- *Refusals:* $|t \uparrow \alpha P_{i,j}| \leq 6 \cdot |t \uparrow \{F\}| + 2$,

Merging these two inequalities arithmetically we eventually yield **Lemma 6**:

$$|t \uparrow \alpha P_{(i+1),(j+1)}| > |t \uparrow \alpha P_{i,j}| - 3. \ \surd$$

5 Network Function

The analysis of each of those 6 communication scenarios resulted in the 6 characteristic inequalities described by Lemmata 1-6 of above. They describe only *partial* aspects of our array. To analyse the array as a whole, these 6 inequalities must be merged and transformed mathematically in a clever way, until a function definition emerges which Roscoe and Dathi have called the networks *variant* [10]. On the basis of this variant it can then be decided whether the network is deadlock-free.

The arithmetic operations merging and transformation of those Lemmata 1-6 have been described in all details in [11] and must be omitted in this short letter due to lack of space. Eventually we arrive at a final inequality with parameters i and j, namely:

$$(2 \cdot |t \uparrow \alpha P_{(i+1),(j+1)}| + 3 \cdot (i + 1 + j + 1)) > (2 \cdot |t \uparrow \alpha P_{i,j}| + 3 \cdot (i + j)),$$

from which the following *characteristic network function* (variant) can eventually be derived as **Lemma 7**:

$$f_{i,j}(t) = 2 \cdot |t| + 3 \cdot (i + j). \ \surd$$

The *purpose* of this function, as fully explained by Roscoe and Dathi [10], is to describe at which grid-*positions* (i, j) there can be processors in the network that are *waiting* for input (from other processors) before they are able to proceed with their own computational tasks.

5.1 Monotony of the Network Function

It is basic knowledge that *deadlock* can only occur in a computational system if four conditions are fulfilled, one of which is *circular wait*. If circular wait is not possible, deadlock cannot occur. This information can be retrieved from the network function (variant) of Lemma 7: IF f is a *monotonic* function, then there cannot possibly exist a cycle $\{(i, j) | (i + j) \in \{r, (r + 1)\}\}$ [10] of grid positions at which the processors would forever wait for each other.

5.2 Conclusion of the Proof

We conclude our argument with **Lemma 8:**

> *The network function $f_{i,j}$, given in Lemma 7 for our hexagonal systolic array, is a monotonic function.* $\surd$

This has been demonstrated in further detail in the project report [11] on which this letter is based.

The meaning of Lemma 8 is that ungranted requests in this hexagonal systolic array can only occur in open *chains*, but never in closed cycles. Consequently (if we did not err in our mathematical calculations) it is now theoretically sure that the hexagonal systolic array, which we have chosen from [9] and defined in CSP as shown in Section 2 of above, is deadlock-free.

Note, however, that we did *not* prove the (numeric) *data* correctness of the matrix multiplication algorithm which this type systolic array is supposed to perform [9].

5.3 Technical Implication of the Proof

The type of hexagonal systolic array discussed in this letter, which has many areas of practical application [9], is deadlock free and can thus be safely implemented on FPGA or similar parallel processing hardware platforms.

6 Research Context and Future Work

The proof result about the deadlock-freeness of haxagonal systolic arrays reported in this letter was achieved 'with pen and paper', in a lengthy and tedious exercise. The space of a letter to this journal is not sufficient to describe the proof [11] in all its details. Such kind of proofs are not only theoretically error-prone but also practically infeasible (with too many man-hours needed per proof). Proof automation is thus needed. Whereas *model checking* [1] of examples such as the one presented in this letter does not seem to be a major problem any more [2] [8], *theorem proving* of such examples is still a problem.

For this reason, Isobe and Roggenbach are continuously developing a theorem prover, *CSP-Prover* [4], which shall be specialized for cases such as the one presented in this letter. With *CSP-Prover*, the deadlock-freeness of a *rectangular* systolic array could already be proven [5], but cases of higher complexity are still a problem for this tool. In this context the manually proven correctness result of the hexagonal systolic array, as presented in this letter, shall be used as a basis for the further development or enrichment of the proof heuristics and tactics encoded in the *CSP-Prover* tool.

Efficient heuristics and tactics for this tool, however, can only be implemented after having studied 'inductively' the 'typical characteristics' of *many* manually conducted correctness proof examples, one of which has been the topic of this letter. An ongoing project in our research group shall therefore yield the 'manual' deadlock-freeness proofs for a wider range of systolic array types, mostly selected from [9], as a basis for the further improvement of the proof tactics and heuristics in the ongoing evolution of the *CSP-Prover* tool.

Acknowledgments

The project results summarised in this contribution to IPL are 'embedded' into a long-term research cooperation in which also M. *Roggenbach* (GB) and Y. *Isobe* (JP), are involved: We thank both of them for interesting and fruitful discussions on this topic.

References

[1] E.M. Clarke, O. Grumberg, D.A. Peled, Model Ckecking, MIT Press 1999.

[2] S.J. Creese, A.W. Roscoe, Data Independent Induction over Structured Networks, PDPTA'2000: Proc. Internat. Conf. on Parallel and Distr. Processing Techn. and Applic., Las Vegas, 2000.

[3] C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall 1985.

[4] Y. Isobe, M. Roggenbach, CSP-Prover: A Proof Tool for the Verification of Scalable Concurrent Systems, Journal of Computer Software 25 (4) (2008) 85-92.

[5] Y. Isobe, M. Roggenbach, S. Gruner, Extending CSP Prover by Deadlock Analysis — Towards the Verification of Systolic Arrays, FOSE'05: Proc. 12^{th} Japanese Workshop on Foundations of Softw. Eng., Japan, 2005.

[6] H.T. Kung, Why Systolic Architectures, Computer 15 (1) (1982) 37-46.

[7] H.T. Kung, Deadlock Avoidance for Systolic Communication, ISCA'88: Proc. 15^{th} Annual Internat. Symp. on Comp. Architecture, Honolulu, 1988, pp. 252-260.

[8] R. Lazić, D. Nowak, On a Semantic Definition of Data Independence, LNCS 2701 (2003) 226-240.

[9] N. Petkov, Systolic Parallel Computing, North-Holland 1992.

[10] A.W. Roscoe, N. Dathi, The Pursuit of Deadlock Freedom, Inf. Comput. 75 (3) (1987) 289-327.

[11] T.J. Steyn, Deadlock Analysis of Hexagonal Systolic Arrays using the Roscoe-Dathi Method, Project Report, Department of Computer Science, University of Pretoria, November 2009, http://ssfm.cs.up.ac.za/TR-ThS-2010.pdf