Design and performance analysis of delay insensitive multi-ring structures

Sparsø, Jens; Staunstrup, Jørgen

Published in:
Proceeding of the 26th Hawaii International Conference on System Sciences

Link to article, DOI:
10.1109/HICSS.1993.270630

Publication date:
1993

Document Version
Publisher's PDF, also known as Version of record

Design and Performance Analysis of Delay Insensitive Multi-Ring Structures

Jens Sparsø Jørgen Staunstrup

Department of Computer Science, Building 344,
Technical University of Denmark, DK-2800 Lyngby, Denmark.

E-mail: jsp@id.dth.dk, jst@id.dth.dk

Abstract

This paper describes a set of simple design and performance analysis techniques that have been successfully used to design a number of nontrivial delay insensitive circuits. Examples are building blocks for digital filters and a vector multiplier using a serial-parallel multiply and accumulate algorithm. The vector multiplier circuit has been laid out, submitted for fabrication and successfully tested. Throughout the paper, elements from this design are used to illustrate the design and performance analysis techniques. The design technique is based on a data flow approach using pipelines and rings that are composed into larger multi-ring structures by joining and forking of signals. By limiting to this class of structures, it is possible - even for complex designs - to analyze the performance and establish an understanding of the bottlenecks.

1 Introduction

Design and automatic synthesis of delay insensitive circuits are active areas of research [1, 2, 3, 4, 5, 6]. Because delay insensitive circuits are different from synchronous circuits, and because delay insensitive circuits are difficult to verify by simulation, most research is devoted to the development of formal design methods. Performance analysis and optimization of delay insensitive circuits is a topic that has only been addressed very recently [7, 8, 9, 10], and therefore the material has not yet matured into widespread use.

This paper describes a set of simple design and performance analysis techniques that have been successfully used to design a number of nontrivial delay insensitive circuits. Examples are: building blocks for digital filters [11], and a vector multiplier using a serial-parallel multiply and accumulate algorithm [12]. The design technique is based on a static data flow concept, and the structure of the circuits consists of pipelines and rings that are connected into *multi-ring structures* by forking and joining of signals. The designs are implemented using a small set of building blocks (latches, combinational circuits and switches) that are realized using C-elements and simple gates like inverters and OR-gates. Only static CMOS circuitry is used, and the circuits are delay insensitive except for some isochronous forks at well defined places within the basic building blocks.

The performance analysis technique that we have used is based on signal transition graphs and follow along the lines developed in [8, 9]. The main contribution of the paper is to demonstrate that by limiting to a specific class of pipeline and ring structures implemented from a simple set of circuit elements, it is possible – even for complex designs – to analyze the performance and establish an understanding the bottlenecks.

The paper is organized as follows: Section 2 introduces delay insensitive pipelines, rings and multi-ring structures. Section 3 describes in qualitative terms the basic performance characteristics of these structures. Section 4 describes the implementation of the basic building blocks. Section 5 illustrates the multi-ring concept by describing briefly the vector multiplier design. Sections 6, 7, 8, and 9 deal with performance analysis: Section 6 defines some key performance parameters. Section 7 analyzes simple ring structures composed of identical stages. Section 8 describes a general analysis technique based on signal transition graphs, and finally section 9 illustrates this technique by calculating the critical path that determines the performance of the vector multiplier.

2 Delay insensitive multi-rings

Delay insensitive circuits are asynchronous and the sequencing of their computations is determined by the data flow rather than by clock signals or other global control signals. When inputs to a sub-circuit are ready, the computation can start and as soon as the result is computed, the next computation can be initiated. In this section we describe a class of circuits, called *delay insensitive multi-rings*, using such a data driven approach.

2.1 Data representation

In a delay insensitive circuit there is no clock signal to determine when a computation can start and when it is complete. Instead, it must be possible to detect the arrival of a new input from the data themselves. To be able to do this, a protocol is used where *empty* values are inserted between proper values. The empty value is denoted E and all values representing proper data ready for computation are called *valid* (denoted V). To distinguish between empty and valid values, data is encoded. It is essential that the encoding

0-8186-1060-3425/93 $03.00 © 1993 IEEE

of valid values ensures that no partial result is interpreted as a valid value, otherwise a computation can start prematurely. There are many ways of encoding data for delay insensitive computations [14]; in section 4.1 we describe a particular and very commonly used encoding called *dual-rail*. Alternating empty and valid values are used in a four cycle handshaking protocol.

Consider a simple computation consisting of the composition of three functions F, G and H. When a valid input is given to F, it can start its computation, and as soon as the result from F is ready, G can begin, immediately followed by H. When all three functions have completed their computations, the valid values must be flushed out, before a new computation can be initiated. This is done by giving an empty value on the input of F. This empty value must now ripple through G and H, and when the output of H has changed to empty, the next computation can start. Hence, each sub-circuit corresponding to F, G, or H must be capable of propagating empty values in addition to computing their respective functions. Such a sub-circuit is called a (delay insensitive) *function block*.

2.2 Pipelines

The composition of F, G and H described above can be realized as a pipeline if a latch is added on the output of each function block. Figure 1 shows this three stage pipeline.

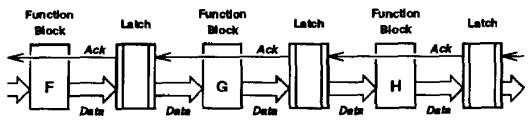

Figure 1: Delay insensitive pipeline.

A delay insensitive latch holds back input data until the successor circuits are ready to receive them. The latch is controlled by acknowledge signals from succeeding latches. A latch may load and hold a valid value when its successor latch in the pipeline holds the empty value (indicated by the incoming acknowledge being false). Similarly a latch may load and hold an empty value when its successor latch in the pipeline holds a valid value (indicated by the incoming acknowledge being true).

In general the function blocks pass the acknowledge signals backwards without any modification, and throughout the rest of the paper the bus symbol representing data signals also implies the associated acknowledge signal.

The handshaking described above ensures that the data flowing through a delay insensitive pipeline always consist of alternating valid and empty values, and the term *data token* or just *token* is used to denote a valid-empty data pair. A token thus occupies two latches or pipeline stages.

When data has been propagated from one latch, L_1, to the succeeding latch, L_2, the contents of L_1 is no longer needed, and it can later be overwritten by data propagated from the preceding latch. Note, that there is period where data stemming from the same token occupies two or more neighboring stages. This is called a *bubble*. Bubbles can be viewed as catalysts: They are necessary for propagating data. When data values flow forward in the pipeline, bubbles flow backwards.

As we shall see in the following sections the number of bubbles in a delay insensitive circuit has a dominating influence on its performance.

2.3 Delay insensitive rings

In this section, we describe a number of generalizations of the pipeline leading to a general characterization of the delay insensitive circuit structures used in this work.

In a latched pipeline with at least three latches, it is possible to connect the output of the last stage to the input of the first, forming a *delay insensitive ring*. Such a ring is capable of performing an iterative computation. In [5, 8] it is described how such a ring is used to perform floating point division. The data in the ring represents a partial remainder, and each stage computes one bit of the final result and forms a new partial remainder which is sent to the next stage etc. Consider a delay insensitive ring with three latches. These latches will always contain one of two patterns: either two valid and one empty element or two empty and one valid. A sequence of computations is shown in figure 2.

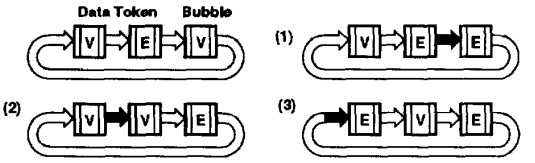

Figure 2: Sequence of computations in a delay insensitive ring.

It is quite simple to build rings with more than three pipeline stages. The choice is mainly governed by performance considerations. In a large ring with many pipeline stages, computations can be overlapped by having more tokens in the ring, and this can improve the throughput.

Independent rings (and pipelines) can be connected in different ways using fork and/or join elements. In a simple join element both inputs are propagated when either they are both empty or they are both valid. There are many possible variations and combinations of join and fork elements. The switch described in section 4.3 is an example such an element: It has two data inputs and two data outputs and a control signal selects how inputs and outputs are connected.

Combining the various building blocks such as function blocks, latches, fork elements, join elements and switches, it is possible to construct a class of delay insensitive circuits consisting of interacting rings and pipelines called *multi-ring* structures.

3 Basic performance characteristics

This section describes in qualitative terms the basic performance characteristics of pipelines, rings and multi-ring structures.

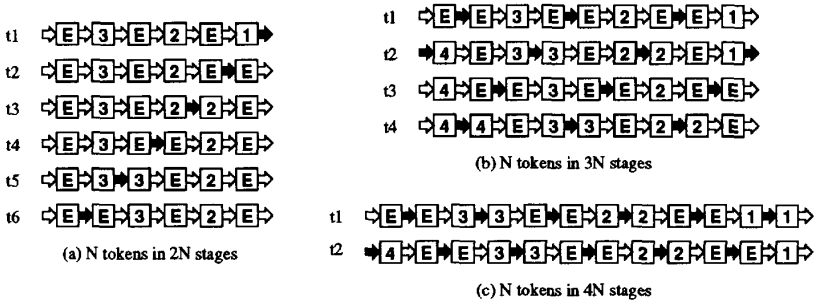

Figure 3: Performance of different delay insensitive shift register structures.

3.1 Introduction

In a synchronous circuit all storage elements are updated in parallel and in this sense a synchronous design is highly concurrent. Another characteristic of a synchronous circuit is that its performance can be determined by a static analysis of the structure of the circuit.

In both respects a delay insensitive circuit is different: The updating of storage elements in a delay insensitive circuit depends on the state (content) of neighbouring storage elements, and if the circuit is not designed carefully these dependencies can greatly reduce the number of operations that can take place in parallel – thus reducing the performance significantly. Furthermore, the performance of a delay insensitive circuit depends not only on the structure of the circuit but also on how the circuit is initialized and on how it is used by the environment.

An understanding of this dynamic behaviour is essential to the design of efficient delay insensitive circuits, and this section introduces some basic principles and develops a qualitative understanding of the topic. Sections 6-9 in the paper describe in detail the analysis techniques.

3.2 Basic concepts

The basic concepts can be illustrated by a simple example: A shift register in which there are N tokens. A shift register is simply a pipeline that is used by its environment in such a way that the number of tokens is invariant. This example is relevant because (1) the vector multiplier described in section 5 contains a number of shift registers, and (2) because it models the behaviour of a ring (in which the number of tokens is also constant).

Figure 3 illustrate the behaviour of different implementations with $2N$, $3N$, and $4N$ stages respectively (for the case $N = 3$). The boxes represent pipeline stages (i.e. latches) and the numbers represent different valid data values. The boldfaced bus-arrows depict state changes (data transfers).

The difference between the three pipeline realizations is the number of bubbles. In figure 3(a) reading a value at the output introduces a bubble that travels backwards causing the data transfers to take place one at a time. Hence, the time it takes to move all elements one stage to the right is at least $2N$. In Figure 3(b) the same computation is illustrated in a pipeline with $3N$ stages. This pipeline contains N bubbles, and therefore N data transfers can occur simultaneously. Hence, the time it takes to move all elements is at least 2. Finally in figure 3(c) the pipeline consist of $4N$ stages. This makes it possible for all valid and empty data values to move simultaneously reducing the time to 1. Increasing the number of bubbles beyond the $2N$ found in figure 3(c) does not increase the performance further.

As the number of bubbles in a design depends on the number of latches per token, the above analysis illustrates that performance optimization of a given circuit is primarily a task of structural modification – circuit level optimization like transistor sizing is of secondary importance.

Finally, it must be emphasized that the above is only a simplified analysis that illustrates some fundamental qualitative properties. It is, however, sufficient to understand the following illustrative but more complex example.

3.3 Example – A shift register

In the vector multiplier described in section 5 we use a shift register with parallel load, a Parallel-In-Serial-Out register called *PISO*. Figure 4(a) shows an initial design with one switch (SW) and two latches per bit. The switch (see figure 7(a)) controls when to parallel load new data and when to perform a right shift. The two latches contain a data token. This design has the same problem as the pipeline in figure 3(a) – there are too few bubbles.

To obtain a reasonable performance 3 latches are needed per stage. This creates more bubbles which enables more data transfers to take place concurrently when data is being shifted. Instead of having 3 latches in the data path the 3rd latch is added in the control path, figure 4(b). In this way, broadcasting of the switch control signal to all switches in the PISO can be avoided, and as will be clear from the following sections, broadcasting of information can have severe impact on the performance.

The PISO design in figure 4(b) exhibits an interesting and illustrative dynamic behaviour: Initially, the data latches in the PISO are densely packed with data tokens, and the latches in the control signal path all contain empty values. For each control token that is input to the PISO it outputs a data token at the serial port or it performs a parallel load. A parallel

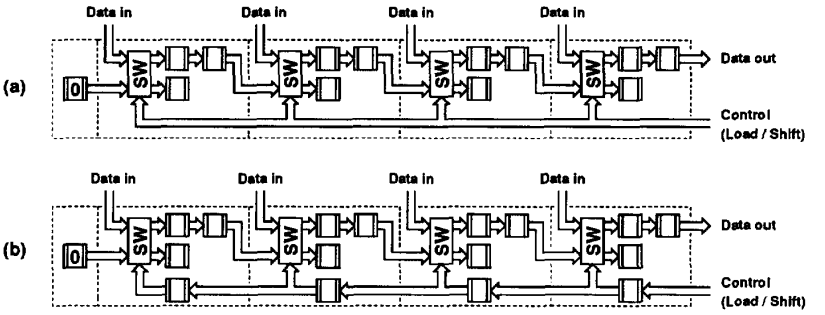

Figure 4: Shift register sections from the PISO: (a) initial and poor design, and (b) final design that avoids broadcasting of the switch control signal.

load replaces the valid data items with new values. As data is being read from the serial port the data tokens will spread over more latches, and control tokens will be travelling in the opposite direction in the control latch chain. When a control token reaches the leftmost end of the PISO a "0-data token" is input from the environment, and as part of this operation the control token disappears. Finally, when the environment stops reading data from the PISO the control latch chain will eventually flush, and the chain of data latches will again fill with valid data items in every second register stage. Notice finally, that the number of data- and control tokens in the design is invariant.

4 Realization of building blocks

In this section, we describe the realization of the most important building blocks (function blocks, latches and switches) that are needed for constructing the delay insensitive multi-ring structures introduced in section 2. More details on the complete set of circuit elements are given in [12].

4.1 Function blocks

A function block computes a specific (combinational) function when its input are valid, and it also propagates empty values. To synthesize such a delay insensitive circuit for a function block we use a technique called *delay insensitive min-term synthesis (DIMS)*. This technique resembles the traditional sum of products approach, but there are also a few important differences:

- the min-terms are formed using C-elements (instead of AND-gates),
- reduction of the boolean equations by combining min-terms into simpler terms is (in general) not allowed.

Together, these requirements assure that the combinational circuits do not produce any valid output signals until all input signals are valid, and that none of the output signals change back to the empty value until all inputs are empty. A similar technique has been used by others [13].

Data is represented in a dual-rail code where two wires, `x.t`, `x.f`, are used to represent a single bit `x`. The value empty (E) is represented with the signals on both wires low, true (T) is represented by `x.t` high and `x.f` low, and false (F) by `x.f` high and `x.t` low.

The DIMS technique can be illustrated with the circuit for a delay insensitive dual-rail AND-gate, see figure 5. The DIMS technique has been automated in a tool which can synthesize delay insensitive circuits from high-level descriptions [15], and we are currently working on a more refined synthesis tool.

a	b	out.t	out.f
E	E	0	0
E	T/F	NO CHANGE	
T/F	E		
F	F	0	1
F	T	0	1
T	F	0	1
T	T	1	0

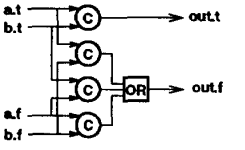

Figure 5: Dual-rail AND-gate.

Multi output function blocks

The DIMS technique does not in general allow reduction of boolean equations. If, however, multiple logic functions depend on the same input, they can share the C-elements and thus achieve a reasonably efficient circuit implementation. As an example, we mention a full adder where both the sum and the carry depend on the two input operands and the incoming carry. A delay insensitive full adder can thus be built using 8 C-elements and 4 OR-gates.

4.2 Latches

A latch for a single dual-rail encoded bit is built from two C-elements, an OR-gate and an inverter, see figure 6.

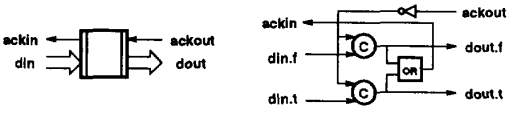

Figure 6: Latch, symbol and implementation.

The OR-gate generates the acknowledge signal, indicating whether the latch holds a valid or empty value. The corresponding acknowledge from the succeeding register, `ackout`, determines whether the register should hold its current value or load a new. No-

tice the similarity between the latch implementation and the function block implementation.

Two degenerated forms of the latch are also needed, one for consuming values (and generating acknowledgements) and one for producing constant values (and consuming acknowledgements). These are quite simple variations of the fundamental latch shown in figure 6. They are, for example, used to "terminate" unused inputs and outputs at the boundary of multi-ring structures.

4.3 Switches

Switches are used as data flow control elements. In the general case two data signals are either crossed or just passed through, determined by a control signal. For the vector multiplier design discussed in this paper we need an asymmetric switch where either both data signals are passed through or only one of them is crossed over and the other waits, see figure 7(a).

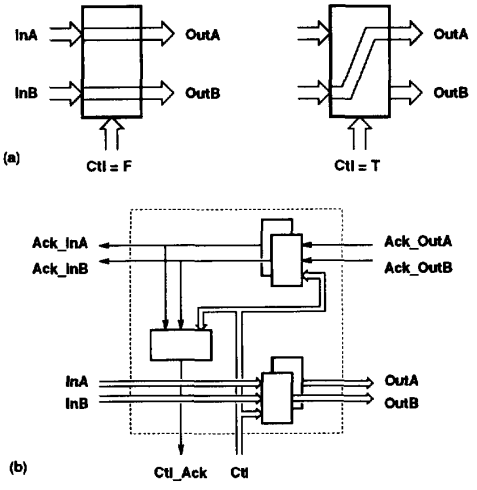

Figure 7: Asymmetric switch.

The asymmetric switch makes it possible to implement shift register structures and to combine rings with different data rates. It should be noted that the control input also follows the four cycle protocol alternating between empty and valid values.

The switch consist of five single output combinational DIMS circuits, see figure 7(b): two for generating data on the output ports, two for generating the acknowledge signals on the input ports and one for generating the acknowledge signal on the control port.

4.4 Fork and join elements

The fork and the join elements complete the set of building blocks. Forks are used when the same signal is input to more circuits and joins are used when signals from more sources are input to a circuit. A join is simply concatenation of data busses – it does not involve any active circuitry. Similarly, a fork is mainly just wires. It only require a C-element to combine the acknowledge signals from the sinks into a single acknowledge signal.

4.5 Initialization

The initialization of a delay insensitive circuit plays a major role. Because the circuit is data driven, it is important to insert tokens and bubbles in such a way that it will start to operate (i.e. to avoid deadlocks). In a synchronous circuit it is very often the case that only a subset of the registers are reset. The remaining registers will then assume well defined values during the first clock cycles of normal operation. This is not a feasible scheme in delay insensitive circuits because, all latches depend not only on their input but also on the output of the succeeding state holding element (via the acknowledge signal). This bidirectional flow of information (data forward and acknowledge backward) makes it necessary to explicitly initialize all C-elements including those used in combinational logic.

5 The vector multiplier

Based on the structural and circuit design techniques presented in the preceding sections we have designed a number of experimental chips. One of these is a vector multiplier which is described briefly below – both to illustrate the design technique, and to provide an example for the following sections on performance analysis. More details on the design are given in [12].

5.1 Algorithm

Input to the vector multiplier are two streams of vector elements (in bit-parallel form) and output is the inner product of the two vectors (also in bit-parallel form). An iterative serial-parallel multiplication algorithm implementing the "paper and pencil approach" is used, figure 8. In each iteration step the circuit performs a multiply, add and shift operation, corresponding to the processing of one row of bit products.

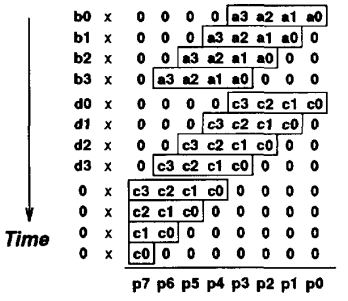

Figure 8: Example of an inner product calculation, $P = A \times B + C \times D$ where P = p7p6p5p4p3p2p1p0 is an 8 bit unsigned integer, and where A, B, C and D are 4 bit unsigned integers.

This algorithm requires: (1) A multiply-accumulate unit in which the result is gradually formed. The width of the accumulator corresponds to the width of the result. (2) A shift register that converts one of the operands into serial representation, and (3) a shift register for shifting the other operand (extended with zeros at both ends) one place to the left in each iteration. The two operands are called the "serial operand" and the "parallel operand" respectively.

To avoid ripple carry propagation in each iteration step, and because it fits nicely with the serial-parallel algorithm, the temporary result is represented in carry save form. Conversion into binary representation is postponed until after the last two vector elements have been multiplied, and the conversion is then done by extending the last "serial operand" with leading zeroes and by taking the circuit through a number of additional iteration steps as indicated in figure 8.

5.2 Design

The core of the design is a combined and bit-sliced implementation of the multiply-accumulate unit and the shift register for the parallel operand. This block is called the Multiply-Accumulate-Shift (MAS) block, and figure 9 shows the design of a bit-slice of this block. In addition, the design consist of a parallel-in-serial-out shift register for the serial operand (called PISO), and a small and simple control unit that in each iteration step issues the control signals (`Ctl1` and `Ctl2`) for the switches in the MAS-block. The implementation of the PISO is explained in section 3.3. A description of the control unit is beyond the scope of this paper, and the interested reader is referred to [12].

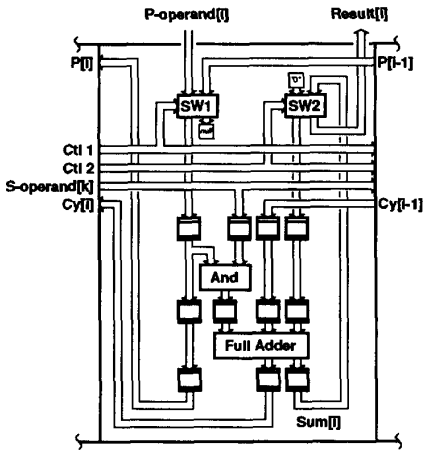

Figure 9: A bit-slice of the MAS unit

The MAS bit-slice in figure 9 is a nice illustration of the multi-ring concept. Starting from the top there are two switches controlling the data flow. Below, a number of signals are broadcast to all MAS bit-slices. These signals are the serial-operand bit and the control signals for the switches. In the bottom there are three rows of latches, an AND-gate and a full-adder, that implement the actual multiply-accumulate-shift function.

¿From a data-flow point of view the bit-slice consists of a small ring and two pipeline sections:

- A 3-stage ring for the sum bit of the accumulator.
- A 3-stage pipeline (or shift register) for the carry bit of the accumulator (the path from `Cy[i-1]` to `Cy[i]`)
- A 3-stage pipeline (or shift register) for the parallel operand (the path from `P[i-1]` to `P[i]`).

The sum ring and the two pipeline sections are connected and synchronized with each other and with the corresponding pipeline sections and rings in the other bit-slices via the the switches and the function blocks.

5.3 Physical implementation

A test chip has been fabricated on EUROCHIP's October 1991 run at ES2 (European Silicon Structures Inc.) in a 1.5 micron CMOS technology. This chip multiplies vectors with 4-bit elements, and it has a 10-bit accumulator for the result. The test of the fabricated chips showed that they are fully functional.

The physical implementation is a standard cell layout. The AutoCells tool (part of the GDT design system from Mentor Graphics Inc.) has been used for the physical implementation. As C-elements are used extensively in the design, we have developed a C-element standard cell generator for the GDT design system. The chip contains 12.450 transistors, and the area of the core of the chip is 7.3 mm^2. The area including pad-cells is 18 mm^2.

The time for a multiply, accumulate and shift iteration step has been determined by a post layout simulation to be 30 ns, which is in accordance with the result of the performance analysis described in section 9.2.

6 Performance parameters

The performance of a pipeline, can be characterized by the parameters: *throughput* and *latency*. A third performance parameter, which does not have an equivalent for synchronous circuits, is the *dynamic wavelength*. Below is a brief definition of these three parameters. A more elaborate definition is given in [8, 9].

Latency: The latency is the delay from input of a data item until the corresponding output data item is produced. When data flows forward, acknowledge signals propagate in the reverse direction, and therefore two parameters are defined: The *forward latency*, L_f, is the delay from a new data on the input of a stage to the production of the corresponding output. It is assumed that the latency is independent of the value of the data. The *reverse latency*, L_r, is the delay from receiving an acknowledge (from the succeeding stage) until the corresponding acknowledge is produced (to the preceding stage).

Period: The *period*, P, is the minimal delay between input of successive tokens. As a token consists of both a valid and an empty data value the period includes a complete four-phase handshake cycle: (1) forward propagation of a valid data value, (2) reverse propagation of acknowledgement, (3) forward propagation of the empty data value, and (4) reverse propagation of acknowledge. Therefore:

$$P = 2 \times L_f + 2 \times L_r$$

Throughput: the *throughput*, T, is the number of tokens that can flow through a pipeline stage per time unit: $T = 1/P$

Dynamic wavelength: The *dynamic wavelength*, W_d, of a pipeline is the number of pipeline stages that a forward propagating data item will pass through during P:

$$W_d = \frac{P}{L_f}$$

The parameters defined above are local performance parameters characterizing the circuit implementation of the individual pipeline stages. When a number of pipeline stages are connected to form a ring the following parameter is relevant:

Cycle time: The *cycle time* of a ring, T_{Cycle}, is the time it takes a token to make one round trip through all pipeline stages in the ring. To achieve maximum performance (minimum cycle time) of a ring, the number of pipeline stages (per token) must match the dynamic wavelength, in which case $T_{Cycle} = P$. If the number of pipeline stages is less, the cycle time will be limited by lack of bubbles, and if there are more pipeline stages the cycle time will be limited by the forward latency through the pipeline stages. In [8, 9] these two modes of operation are called *bubble limited* and *data limited* respectively. The cycle time of a ring with N stages and one token can be computed from one of the following two equations:

When $N \geq W_d$ the cycle time is limited by the forward latency through the N stages:

$$T_{Cycle}(DataLimited) = N \times L_f$$

When $N \leq W_d$ the cycle time is limited by the reverse latency. With N pipeline stages and one token, the ring contains $N-2$ bubbles, and as a cycle involves $2N$ data transfers (N valid and N empty) the cycle time becomes:

$$T_{Cycle}(BubbleLimited) = \frac{2N}{N-2} L_r$$

For the sake of completeness we mention that a third possible mode of operation called "control limited" exists for some circuit configurations [8, 9]. This is, however, not relevant for delay insensitive multi-ring structures implemented using the building blocks described in section 4.

7 Analysis of simple rings

When the overall structure of a design is being settled, an important design task is to determine the optimal number of pipeline stages in the rings in the design. In order to establish a basis for first order design decisions, this section analyzes some simple rings composed of identical pipeline stages each consisting of a one-level DIMS function block followed by a latch. To get a lower bound on the cycle time, the analysis also includes rings without function blocks, i.e. rings consisting of latches only. Although rings may contain many tokens we restrict to rings with a single token, because this accounts for most practical designs and because it simplifies the analysis.

The cycle time is expressed in terms of the delays in the basic components: C-elements, NOR-gates and inverters (used as buffers). Their delays are denoted t_C, t_N and t_I respectively. From simulations of standard cells implemented in the 1.5 micron CMOS technology that is available to us via EUROCHIP, we have found the following actual delay values: $t_C = 1.1$ ns, $t_N = 0.6$ ns and $t_I = 0.7$ ns. These values include the effect of representative output loads.

It should be noted that although most circuits use OR-gates it is normally possible to optimize the circuits by combining the OR-gates with inverters elsewhere in the circuit (c.f. the latch in figure 6). This is the reason why the analysis use NOR-gate delays.

The forward latency of a pipeline stage is the delay through a simple DIMS function block (figure 5) and a C-element in the succeeding latch (figure 6), and $L_f = 2t_C + t_N$. If the function block is deleted, the forward latency reduces to $L_f = t_C$. The reverse latency is the delay from acknowledge input to acknowledge output of a latch (figure 6), and it is the same in both cases, $L_r = t_C + t_N$. From these figures, it is possible to compute the period and the dynamic wavelength for the two circuit configurations. For the pipeline without function blocks $W_d = 5.1$ stages and for the pipeline with simple function blocks $W_d = 3.2$ stages. We have therefore analyzed the cycle time for rings with 3, 4 and 5 pipeline stages. Table 1 shows the analytical results as well as the actual values.

	Pipeline stages without function blocks		Pipeline stages with simple function blocks	
		ns.		ns.
L_r	$t_C + t_N$	1.7	$t_C + t_N$	1.7
L_f	t_C	1.1	$2t_C + t_N$	2.8
P	$4t_C + 2t_N$	5.6	$6t_C + 4t_N$	9.0
W_d	5.1 stages		3.2 stages	
$T_{Cycle}(3)$	$6\ L_r$	10.2	$6\ L_r$	10.2
$T_{Cycle}(4)$	$4\ L_r$	6.8	$4\ L_f$	11.2
$T_{Cycle}(5)$	$3.3\ L_r$	5.7	$5\ L_f$	14.0

Table 1: Performance of different simple ring configurations.

In order to perform computations a ring must contain at least one function block, and from table 1 it can be seen that the optimal number of pipeline stages is less than 5. Going from 3 to 4 stages result in a marginal performance improvement at a significant area increase. Therefore, rings with 3 pipeline stages is in general the optimal choice with our circuit configuration.

8 Signal transition graph analysis

When all pipeline stages are identical (as it was the case above), it is quite simple to determine the cycle time directly. However, in general a more systematic procedure is needed. Signal transition graphs have been proposed as a formal model in which it is possible

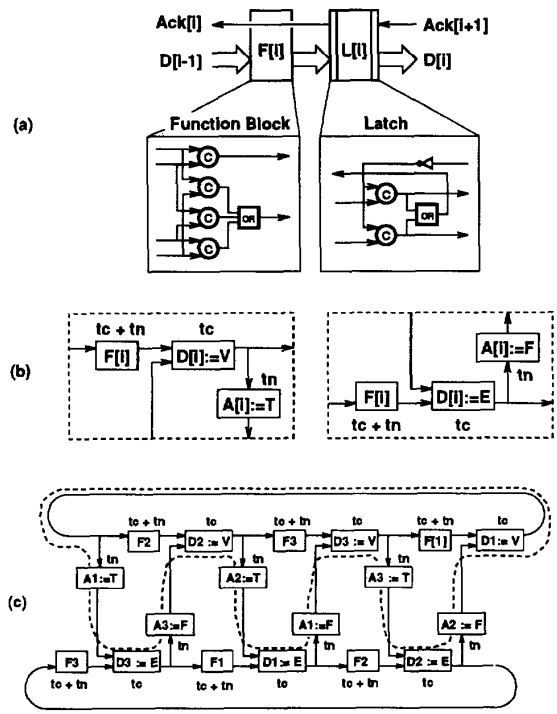

Figure 10: (a) Pipeline stage circuit configuration, (b) the corresponding two sets of nodes for the signal transition graph, and (c) the signal transition graph for the simple 3-stage ring.

to determine the cycle time in a very systematic (or even mechanical way) [8, 9].

Figure 10(c) shows the signal transition graph for the simple 3-stage ring. The nodes in the graph represent signal transitions and the edges represent dependencies between the signal transitions. The graph is closely related to the circuit diagram (explained below): The nodes in the graph correspond to circuit elements and the edges correspond to wires. As signals in delay insensitive circuits alternate between "valid" and "empty" (in the case of data-signals) or between "true" and "false" (in the case of acknowledge signals and other boolean signals) the graph has two nodes per circuit element. Figure 10(a) shows a pipeline stage and figure 10(b) shows the corresponding two sets of nodes - the fragments from which the full dependency graph in figure 10(c) is composed. The labels outside the node boxes denote circuit delays associated with the signal transition. We have developed a particular style for the graphs that we find very illustrative and easy to understand: The nodes corresponding to the forward flow of valid and empty data values is organized as two horizontal rows, and nodes representing the reverse-flowing acknowledge signals appear as segments connecting the rows.

The cycle time of the ring is the time from some signal transition until the same signal transition occurs the next time. The cycle time can therefore be determined by finding the longest simple cycle in the graph, i.e. the cycle with the largest accumulated circuit delay. The dashed path in figure 10(c) corresponds to the (bubble limited) longest simple cycle. As in table 1 the cycle time is:

$$T_{Cycle(3)} = 6t_C + 6t_N \quad (1)$$

Notice that the single bubble is involved in 6 data transfers and the bubble therefore makes two reverse round trips in the ring for each forward cycle of data.

Figure 10(c) also illustrates that if function blocks with delays greater than $t_C + 2t_N$ are used, the cycle time will be data limited corresponding to a path through the upper row of "valid assignment nodes" or through the bottom row of "empty assignment nodes."

A dependency graph analysis of a 4-stage ring is very similar. The only difference is that there are two bubbles in the ring. In the signal transition graph this corresponds to the existence of two "bubble cycles" that do not interfere with each other.

9 Cycle time of the vector multiplier

The core of the vector multiplier chip is the multiply-accumulate-shift block and as a designer one would expect the critical path in the design to be found here. As we shall see in the following this is unfortunately not the case in the present design.

9.1 Cycle time of the MAS block

The MAS block in the design (figure 9) consist of 3 pipeline stages with function blocks that are somewhat more complex than the simple DIMS-circuits from the analysis in the previous section. Because of the implicit synchronization of data signals in the function blocks and switches, the MAS-block can be analyzed using an equivalent three stage ring with the following function blocks between the latches:

1. *The full adder* (section 4.1) is a multi-output function block. Conceptually it consists of two combinational circuits to which the input signals are forked. This forking requires a C-element in the acknowledge path to combine the acknowledge signals from the two output ports.
2. *The switch* (figure 7) is even more complex than the adder. It consist of: (a) two combinational circuits in the forward data path (from control input to data output), (b) a combinational circuit in the reverse acknowledge path, and (c) another combinational circuit that computes the acknowledge signal on the control port.
3. *The AND-gate* (figure 5) that computes the bit-product is a simple DIMS circuit.

In order to take the acknowledge circuitry in the adder and the switch into account, the signal transition graph in figure 10 must be extend with extra nodes in the outgoing edges of the appropriate acknowledge assignment nodes. The delay associated with these extra nodes is t_C for the new full-adder acknowledge nodes and $t_C + t_N$ for the new switch acknowledge nodes. As each circuit correspond to two

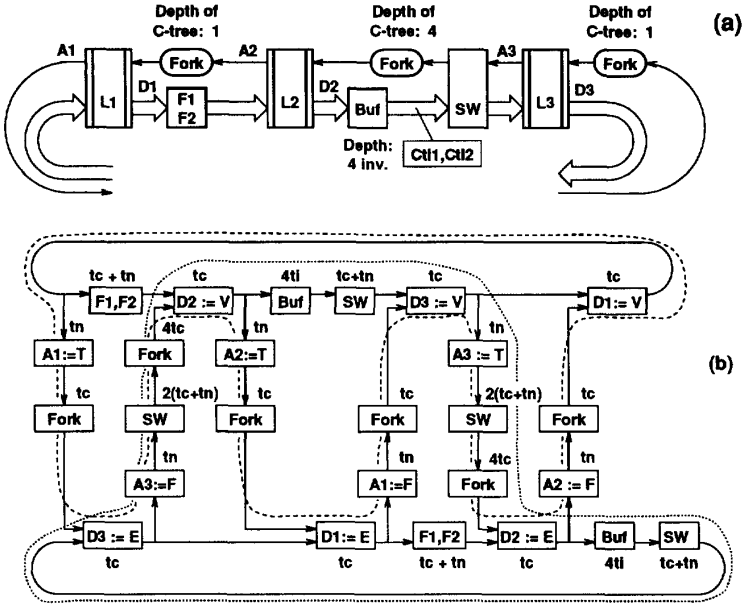

Figure 11: Critical path in the design. Equivalent circuit (a) and corresponding signal transition graph (b).

extra nodes and as these nodes go into the longest cycle in the graph, the period in equation 1 of the ring increases with $2(t_C + t_N) + 2t_C$:

$$T_{Cycle} = 10t_C + 8t_N \qquad (2)$$

This is the minimum period for a step of the serial-parallel multiplication algorithm and therefore the speed at which one would wish the entire circuit to operate. As explained in the next section the critical path is unfortunately somewhat larger.

9.2 The critical path

The critical path in the design is found in a small control block that generates the two signals, `Ctl1` and `Ctl2`, that control the switches in the MAS-block (figure 9) in the PISO (figure 4) and in the control unit itself. The control signals make one four-phase handshake cycle per iteration step.

The control unit is based on the same circuit elements that are used in the PISO and in the MAS-block (switches, latches and function blocks) and its structure is also based on connected pipelines and rings. By virtue of the switch control signal `Ctl1` the control block contains some 3-stage rings. An equivalent circuit diagram from which the critical path can be computed is shown in figure 11(a). The corresponding signal transition graph is shown in figure 11(b).

In comparison with the results of the previous sections, extra delay is associated with forking (or broadcasting) of the control signal `Ctl1` to the many switches. This broadcasting requires buffering of the control signal and the depth of the C-element tree that combines all the acknowledge signals becomes significant. This circuitry is represented by two "fork" nodes in the graph representing C-element trees with a depth of 4. From figure 11(b) we can compute the cycle time of the vector multiplier. There are two cycles in the graph with the same worst case cycle time:

Configuration	Cycle time	
(1) Simple 3-stage ring	$6t_C + 6t_N$	10.2 ns
(2) MAS bit-slice (figure 9)	$10t_C + 8t_N$	15.8 ns
(3) Critical path	$22t_C + 10t_N$	
(4)	$18t_C + 8t_N + 8t_I$	30.2 ns.

Table 2: Performance of different circuit configurations.

$$T_{Cycle} = 22t_C + 10t_N \qquad (3)$$
$$T_{Cycle} = 18t_C + 8t_N + 8t_I \qquad (4)$$

9.3 Summary

The results of the previous sections are summarized in table 2. The only difference between (2) and (3) in table 2 is that in (3) broadcasting (i.e. forking) of the switch control signals has been taken into account. Broadcasting of signals is very expensive, primarily because the many acknowledge signals have to be combined into a single acknowledge signal using a tree of C-elements (section 4.4), but in some cases the buffers that are needed to drive the large capacitive load of the many inputs can also enter the critical path.

In general, broadcasting should therefore be avoided. A possible redesign that could reduce the

cycle time would be to distribute the control signal to the switches in the RING in the same way as in the PISO (section 3.3). This would cause some minor skewing of the operations taking place in the bit-slice rings and a corresponding overhead at start up and termination.

In such a redesign the switch control signal would have to bubble through a latch in each of the 10 bit-slices from which the ring is composed, and with a delay of $t_C = 1.1$ ns in each this would take 11.0 ns. With this investment per multiplication the cycle time of the iterations could be reduced from 30.2 ns to 15.8 ns. A design incorporating these modifications is currently in fabrication.

Finally we mention that the analytically determined cycle time of 30.6 ns. of the existing design conforms nicely with the results obtained from a post layout simulation of the entire chip (30 ns c.f. section 5).

10 Conclusion

We have described a data flow based approach to the design of delay insensitive circuits. The underlying structural concept is simple: pipelines and rings. These can be combined into more complex structures – called multi-rings – by joining and forking of signals. Such multi-rings can be implemented from a small set of building blocks, consisting of latches and a variety of combinational circuits: function blocks, switches etc.

One of the nice features of delay insensitive multi-rings is that they make it relatively simple to analyze and optimize the performance of a design. This has been illustrated with examples taken from a fully developed design of a delay insensitive VLSI chip for computing the inner product of two vectors.

Acknowledgement

This has work has been supported by The Danish Technical Research Council. The CAD tools from Mentor Graphics Inc. have been provided via EUROCHIP.

References

[1] Alain Martin et al., "The Design of an Asynchronous Microprocessor," *Decennial Caltech Conference on VLSI*, Ed. C.L. Seitz, MIT Press 1989, pp. 351-357.

[2] Ivan E. Sutherland, "Micropipelines," *Communication of the ACM*, Vol. 32, no. 6, 1989, pp. 720-738.

[3] C.H. van Berkel, C. Niessen, M. Rem, and R. Sajs, "VLSI Programming and Silicon Compilation: A Novel Approach from Philips Research," *Proceedings of IEEE International Conference on Computer Design 1988*, IEEE Computer Science Press, Washington DC, 1988, pp. 150-166.

[4] Jo C. Ebergen, "A formal approach to designing delay-insensitive circuits," *Distributed Computing*, Vol. 5 no. 3. 1991, pp. 107-119.

[5] T. Williams and M. Horowitz, "A Zero-Overhead Self-Timed 160 ns. 54 bit CMOS Divider," *IEEE Journal of Solid State Circuits*, vol. 26, no. 11, Nov. 1991, pp. 1651-1661.

[6] Jørgen Staunstrup and Mark R. Greenstreet, "Designing delay insensitive circuits using 'Synchronized Transitions' ," In Luc J.M. Claesen editor, *Formal VLSI Specification and Synthesis. VLSI Design Methods, volume I*, North-Holland/Elsevier, 1990, pp. 209-226.

[7] Mark R. Greenstreet and Kenneth Steiglitz, "Bubbles Can Make Self-Timed Pipelines Fast," *Journal of VLSI Signal Processing*, 2, 1990. pp. 139-148.

[8] Ted E. Williams, "Self-Timed Rings and their Application to Division", Ph.D. thesis, Department of Electrical Engineering and Computer Science, Stanford University, May 1991.

[9] Ted E. Williams, "Analyzing and Improving Latency and Throughput in Self-Timed Rings and Pipelines", *Proc. of 1992 ACM/SIGDA Workshop on Timing Issues in the Specification and Synthesis of Digital Systems,* Princeton University, March 18-20, 1992,

[10] Steven M. Burns, "Performance Analysis and Optimization of Asynchronous circuits," Ph.D. thesis, Caltech-CS-TR-91-01, Computer Science Department, California Institute of Technology, California USA, 1991.

[11] Erik Plesner, "Self-Timed circuits for digital signal processing" M.Sc. thesis (ID-E 561), Dept. of Computer Science, Tech. Univ. of Denmark, Lyngby, 1992.

[12] Jens Sparsø, Jørgen Staunstrup and Michael Dantzer-Sørensen, "Design of delay insensitive circuits using multi-ring structures," *Proc. of EURO-DAC '92, European Design Automation Conference,* Hamburg, Germany, Sept. 7-10, 1992, IEEE Computer Society Press, pp. 15-20.

[13] N. P. Singh, "A Design Methodology for Self-Timed Systems," M.Sc. thesis, MIT/LCS/TR-258, Laboratory for Computer Science, Massachusetts Institute of Technology, February 1981.

[14] T. Verhoeff, "Delay Insensitive codes – an overview," *Distributed Computing*, Vol. 3, no. 1, 1988. pp. 1-8.

[15] Henrik Hulgaard and Per H. Christensen, "Automated Synthesis of Delay Insensitive Circuits," M.Sc. thesis (ID-E 511), Dept. of Computer Science, Tech. Univ. of Denmark, Lyngby, 1990.