μSystems Research Group

School of Engineering

Exploiting Robustness in Asynchronous Circuits to Design Fine-Tunable Systems

Benafa Ikechukwu Oyinkuro

Technical Report Series

NCL-EEE-MICRO-TR-2018-210

August 2018

Contact: o.benafa@ncl.ac.uk

Supported by Niger Delta Development Commission (NDDC)

NCL-EEE-MICRO-TR-2018-210

μSystems Research Grou

School of Engineering

Merz Court

Newcastle University

Newcastle upon Tyne, NE1 7RU, UK

http://async.org.uk/

I would like to dedicate this thesis to my loving parents.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Benafa Ikechukwu Oyinkuro

August 2018

Abstract

Robustness property in a circuit defines its tolerance to the effects of process, voltage and temperature variations. The mode signaling and event communication between computing units in a asynchronous circuits makes them inherently robust. The level of robustness depends on the type of delay assumptions used in the design and specification process. In this thesis, two approaches to exploiting robustness in asynchronous circuits to design self-adapting and fine-tunable systems are investigated. In the first investigation, a Digitally Controllable Oscillator (DCO) and a computing unit are integrated such that the operating conditions of the computing unit modulated the operation of the DCO. In this investigation, the computing unit which is a self-timed counter interacts with the DCO in a four-phase handshake protocol. This mode of interaction ensures a DCO and computing unit system that can fine-tune its operation to adapt to the effects of variations. In this investigation, it is shown that such a system will operate correctly in wide range of voltage supply. In the second investigation, a Digital Pulse-Width Modulator (DPWM) with coarse and fine-tune controls is designed using two Kessels counters. The coarse control of the DPWM tuned the pulse ratio and pulse frequency while the fine-tune control exploited the robustness property of asynchronous circuits in an addition-based delay system to add or subtract delay(s) to the pulse width while maintaining a constant pulse frequency. The DPWM realized gave constant duty ratio regardless of the operating voltage. This type of DPWM has practical application in a DC-DC converter circuit to tune the output voltage of the converter in high resolution. The Kessels counter is a loadable self-timed modulo$-n$ counter, which is realized

by decomposition using Horner's method, specified and verified using formal asynchronous design techniques. The decomposition method used introduced parallelism in the system by dividing the counter into a systolic array of cells, with each cell further decomposed into two parts that have distinct defined operations. Specification of the decomposed counter cell parts operation was in three stages. The first stage employed high-level specification using Labelled Petri nets (LPN). In this form, functional correctness of the decomposed counter is modelled and verified. In the second stage, a cell part is specified by combing all possible operations for that cell part in high-level form. With this approach, a combination of inputs from a defined control block activated the correct operation for a cell part. In the final stage, the LPNs were converted to Signal Transition Graphs, from which the logic circuits of the cells were synthesized using the WorkCraft Tool. In this thesis, the Kessels counter was implemented and fabricated in 350 nm CMOS Technology.

Acknowledgements

I would like to express appreciation to Prof. Alex Yakovlev my supervisor who introduced me to asynchronous circuits and systems and also guided and supported me during my PhD research.

I am grateful to my parents Mr Penakeme Benafa and Mrs Preye Benafa for their encouragement, support and advice before and during the period of this research.

I am also grateful to Dr Delong Shang and Dr Danil Sokolov for their assistance and valuable contributions to circuit design and modelling.

I would like to thank Mr Imran Ahmed, for his support and encouragement and not forgetting to mention Lorna Marshall.

I am also grateful to Graham Ewart, Darren Mackie, Paul Killan and Jeffrey Warren for their assistance in the electronics workshop.

This research was supported by the Niger Delta Development Commission (NDDC).

Table of contents

List of figures

List of tables

Acronyms

CL counter cell left part.

CR counter cell right part.

CT_n counter.

C_i ith cell.

c'_i ith control cell.

c_i ith counter cell.

Ack Acknowledgement.

CSC Complete State Coding Conflict.

CSP Communicating Sequential Protocol.

DFF D Flip-Flop.

DI Delay Insensitive.

DLL Delay locked loop.

DPWM Digital Pulse-width Modulator.

HA Half adder.

IoT Internet of Things.

LCO Limit Cycle Oscillation.

LPN Labelled Petri net.

LSB Least significant bit.

MSBs Most significant bits.

PLL Phase locked loop.

PN Petri net.

PVT Process Voltage and Temperature.

QDI Quasi-delay Insensitive.

Req Request.

SI Speed Independent.

STG Signal Transition Graph.

VLSI Very Large Scale Integration.

Chapter 1

Introduction

1.1 Motivation and Objectives

Time constraints which includes setup and hold times are subject to Process Voltage and Temperature (PVT), ageing, coupling and clock jitter etc must be considered when choosing the minimum clock period in synchronous systems [3]. These timing requirements present two operational limits which are of concern in this thesis.

Firstly, a computing system operating in a harsh environment may require an intelligent input clock control system that is sensitive to changes in the operating environment of the computing system to avoid computational error by dynamically tuning the clock frequency [3–10, 2]. In this approach, response time, metastability and sensing precision of the control system are factors which determine the reliability of the system. Addressing these factors are non trivial.

Secondly, due to the discrete nature of digital circuits and the coarse nature of the clock, digital systems are prone to issues of quantisation when applied in measuring or controlling analogue quantities.

For example, quantisation in DC-DC converter that employs a Digital Pulse-width Modulator (DPWM) may cause the converter to go into a state known as Limit Cycle

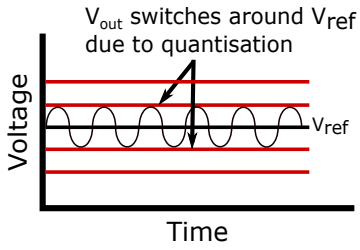


Fig. 1.1 Illustration of LCO due to the effect of quantisation

Oscillation (LCO) [11], illustrated in Figure 1.1. The converter system may spend time and energy trying to exit this state by tuning the input clock frequency and the pulse ratio of the DPWM and this can ultimately affect the quality of the converter output. The minimum clock period required for correct and reliable operation of the DPWM can limit fine-tuning its pulse-width in high frequency.

In this thesis, the two operational limits of synchronous systems mentioned above are addressed by investigating methods by which robustness in asynchronous circuits can be exploited to design adaptable and fine-tunable systems.

Asynchronous systems operate by local synchronisation and communication of events between computing units. The mode of communication and data validation between computing units enable asynchronous systems to perform reliably in the presence of PVT variations and logic gates delay changes, thus asynchronous circuits are said to be robust in nature. Robustness can be exploited in the design of systems that can adapt their operations to changes in circuit operating conditions [12, 13]. In the asynchronous circuits review section in this thesis, it is shown that robustness in asynchronous systems depend on the delay assumptions used in specifying and realising their circuits.

1.2 Research Contributions

Two approaches to methods by which robustness in asynchronous circuits can be exploited to design adaptable and fine-tunable systems were investigated in this thesis.

The first investigation presented a method of integrating a tunable input clock and its triggered computational unit which is self-timed, such that the operation of the system dynamically adapts to annul the effects of variation. In this investigation, a self-timed computing unit and a Digitally Controlled Oscillator (DCO) with coarse and fine-tune controls were integrated as one unit such that the structure of the system ensured the status and operating conditions of the computing unit modulated the DCO operation. This is achieved with the use of a completion detection system between the DCO and the computing unit in a feedback loop ensuring that the oscillator can only send a request (rising transition) when the computing unit is idle.

The computing unit used in the first approach is a self-timed counter whose next task completion time varies, depending on its present state. This behaviour can be seen to model changes in circuit delay due to variation making it suitable for this investigation. Later in this thesis, it is shown through simulation that the operation of the integrated DCO-computation system would operate correctly over a wide range of voltage supply.

The second investigation presented an application in which a more complex and reconfigurable asynchronous system is used to demonstrate how robustness can be exploited to design fine-tunable digital systems employed in analogue control circuitry like the DC-DC converter. A DPWM with coarse and fine-tune controls was designed in this investigation. The main component of the DPWM is a loadable Kessels counter which is a loadable self-timed modulo$-n$ counter. Two such counters formed the coarse control of the DPWM, while a fine-tune control system which is an addition based delay ensured a constant pulse period while fine-tuning the pulse-width. It is proposed that an asynchronous DPWM will accommodate the addition of arbitrary delay(s) to fine-tune the pulse-width in a given period without violating time margin requirements of its circuit. This is investigated in chapter 6 of this thesis.

The loadable kessels counter was realised in a top-down approach in which formal methods are employed in defining, decomposing and specifying the counter behaviour. This includes the following steps:

- High-level rewriting, specification and decomposition of the loadable counter into systolic array of cells using Horner's Method. Each counter cell is further decomposed into left and right cell parts. This method of decomposition was first proposed by Kessels [14].

- Definition of the different conditions and range of possible operations for each counter cell parts. This approach led to a reconfigurable counter operation.

- Specification of the counter cells operations using formal models: High level specification using Labelled Petri net (LPN) [15] and low level specification using Signal Transition Graph (STG) [16]. Low level specification also included Complete State Coding Conflict (CSC) conflict resolution when needed to realise synthesisable specifications. The WorkCraft Tool [1] developed at Newcastle University was used extensively at this stage for LPN and STG editing, verification and simulation of each specification and synthesis of the STGs. This stage involved three steps:

    1. Each cell part operation was specified using LPN. In this stage, the functional correctness of the decomposed counter was modelled and verified by unfolding of actions using directed graphs.

    2. The LPN of all possible operations for each cell part were combined such that a combination of inputs from a defined control block and an adjacent cell part activated the correct operation in a given cell part.

    3. The LPNs were converted to STG, which showed ordered rising and falling transitions of interacting signals in the system. At this stage, CSC errors present

in the specification were resolved by strategic addition of internal signals. The logic equations for each cell part were synthesized from its CSC error free STG.

- Design of a control block which consists of interacting control cells (each cell is divided into two cell parts), each in a one-to-one configuration relationship with the counter cells.
- Specifications of load channel encodings between interacting control cell parts and configuration channel encodings between each interacting control and counter cell parts.
- Synthesis of the logic equations of the specifications from which the counter was implemented and fabricated in 350 nm CMOS Technology. This involved technology mapping of synthesised gates to AMS 350 nm CMOS standard cells library.
- Characterisation of the loadable self-timed modulo$-n$ counter response time and power consumption.

A DPWM with both coarse and fine-tune controls is realised by synchronising the operations of two loadable Kessels counters. A practical example of how robustness in asynchronous circuits can be exploited actively is presented with an addition based tunable delay system used to provide the fine-tuning circuit of the DPWM.

1.3 Publications

The following is a list of publications and paper(s) to be submitted for review as a result of this research work.

- D. Shang, O. Benafa, F. Xia, Y. Xu and A. Yakovlev, "An elastic timer for wide dynamic working range," 2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS), Grenoble, 2015, pp. 1-4.

- O. Benafa, A. Ogweno, D. Shang and A. Yakovlev, "Design of a DCO based on the worst-case delay of a self-timed counter and a digitally controllable delay path," 2016 14th IEEE International New Circuits and Systems Conference (NEWCAS), Vancouver, BC, 2016, pp. 1-4.

- O. Benafa, D. Sokolov and A. Yakovlev, "Loadable Kessels Counter" Presented in ASYNC 2018 conference in Vienna, Austria.

- O. Benafa and A. Yakovlev, "Exploiting robustness in asynchronous systems using loadable self-timed Kessels Counters," Journal to be submitted for review.

1.4 Structure of the Thesis

This thesis is organised as follows:

- Chapter 2 This chapter gives a background study of asynchronous circuits and systems. This includes basic definitions of asynchronous terms, classification of asynchronous circuits and formal methods of specifying asynchronous circuits. This chapter also presents methods and applications in which robustness in asynchronous circuits have been exploited.

- Chapter 3 This chapter presents relevant counter and modulo$-n$ counter decompositions. The counters are grouped under synchronous and asynchronous counters. This chapter is a review of counter decompositions, a precursor to the loadable self-timed modulo$-n$ counter realised in Chapter 5.

- Chapter 4 A robust system formed by integrating a DCO and computing unit is investigated and presented in this chapter.

- Chapter 5 In this chapter, a loadable self-timed modulo$-n$ counter which takes a quantity, n, in space and converts it to time is realised by formal specification and formal

asynchronous design techniques. The counter was also synthesised and fabricated in 350 nm CMOS Technology. Simulation and test results of the counter are included in this chapter.

- Chapter 6 This chapter investigates a practical application of the loadable self-timed counter in the design of a DPWM with coarse and fine-tune controls.

- Chapter 7 This chapter summarises the design methods and results in this thesis. It also discussed the main areas for future works based on the results presented in this thesis.

Chapter 2

Background

This chapter presents basic definitions of asynchronous terms and formal methods of asynchronous specification and verification. This forms the background study for the asynchronous circuits described in the proceeding chapters. The last two sections before concluding this chapter presents a brief review of methods of synthesizing asynchronous circuits and approaches to exploiting asynchronous circuit properties.

2.1 Asynchronous Circuits

Computing operation between units in an asynchronous system is synchronised using handshaking techniques. This method of synchronisation differentiates asynchronous systems from synchronous systems in which operations are synchronised by a global clock.

The use of a global clock simplifies system design and verification process. However, there are some limits imposed on the system when compared to asynchronous systems. These limits are:

- The clock period and clock distribution must be carefully chosen to accommodate the circuits design margins such as, set-up and hold time of the logic gates, ageing which affects the circuits over time, process, voltage and temperature variations [19–21]. It

is required that the clock period is large enough to guarantee hazard free operation of the circuit as the design margins may change over time or due to environmental conditions.

- The clock period must accommodate the worst-case delay of the combinational logic blocks in the system [22].

- Constant switching of the clock results in dynamic power loss [23], although methods of controlling and minimising dynamic power loss have been presented (clock gating [24], dynamic voltage scaling [25] and dynamic frequency and voltage scaling [26]). However, application of any of these techniques may result in a trade-off in system performance in terms of operating speed and area consumed.

- Any arbitrary delay introduced into the system without considering the clock period can result in hazardous or incorrect operation.

In asynchronous circuits, a computing unit initiates a handshake process by sending a request signal to the receiving unit, indicating the validity of the data to be sent over for computation. The receiving unit returns an acknowledgement signal to the sender after consuming the data. The sending unit can initiate a new handshake transaction after it has received an acknowledgement from the receiving unit. This gives asynchronous circuits the following advantages over synchronous circuits [27–31].

- Dynamic power loss is minimised because switching in different sections of the system is event-driven.

- Asynchronous circuits operate at average case speed instead of worst-case speed as in synchronous circuits. This point can be contested especially since bundled data systems typically allows for 100% overestimate of the matching delay to cater for variability.

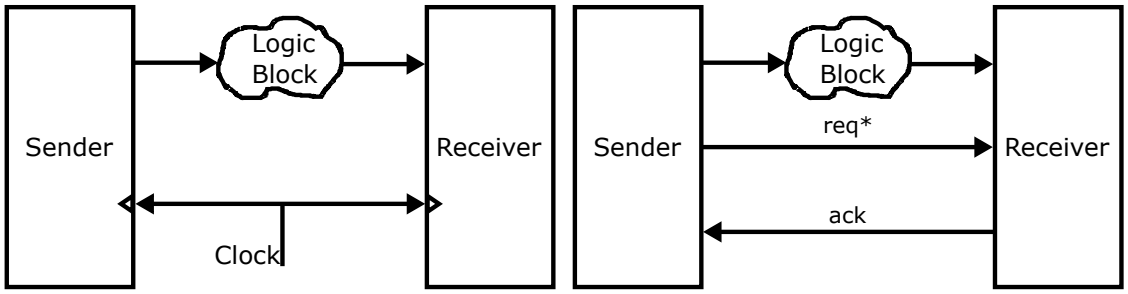
- Asynchronous circuits are robust due to handshake communication between units and thus can tolerate process, voltage and temperature variations to a large extent depending on the delay assumptions used in design the system.
- Modularity of design, large circuits can be realised by composing together smaller modules which have been tested and verified to ensure adherence to well-defined interface protocols.
- Lower electromagnetic interference and emission.

2.2 Classes of asynchronous circuits

Figure 2.1 illustrates the synchronisation methods for synchronous and asynchronous circuits. While the assumption that data will be ready at every clock edge is made in synchronous systems and therefore every unit is expected to compute at the rising edge or falling edge or both edges of the clock, asynchronous systems operate by local synchronisation of events between two computing units that are ready to transfer and accept data. In Figure 2.1b, the request channel is labelled req^* because the request must arrive at the receiver when the data from the combinational circuit is valid.

Asynchronous circuits are classified based on the delay assumptions used to realise their circuits. The delay assumptions to a large extent determine the circuit's tolerance to process, voltage and temperature variations. Asynchronous circuits can be classified as Delay Insensitive (DI), Quasi-delay Insensitive (QDI) and Speed Independent (SI) circuits [32–34].

Delay Insensitive (DI) circuits take into account delays associated with wires and gates and assume that they are finite. Therefore the circuit can work correctly with unbounded gate and wire delays. Asynchronous circuits realised as DI are very robust, i.e. they can tolerate variations to a large degree; however, they are limited to simple gates as it is difficult to realise as complex circuits that are DI.



(a) Synchronous System (b) Asynchronous System

Fig. 2.1 Modes of communication: Synchronous and Asynchronous

Quasi-delay insensitive (QDI) circuits are a modification of DI circuits to allow for complex composition of circuits. Delay matching is achieved by way of isochronic forks. This means that the delay between ends of the fork is assumed to be negligible.

Speed independent (SI) circuits take into account delay associated with gates; however, they assume wire delays are negligible. These circuits require a completion signal to indicate when all computation is complete.

2.3 Signalling and data path communication conventions

Special protocols are used to indicate the validity and consumption of data in asynchronous systems. The communication protocol can either be two-phase also called transition signalling protocol or four-phase also called level signalling protocol [32]. For simple applications, each channel can be encoded as single-rail encoding or dual-rail irrespective of the communication protocol used. Each communication protocol is discussed under these two encodings.

2.3.1 Single-rail encoding

This encoding uses two separate wires, one for request and the other for acknowledgement, bundled along with the data to communicate between computing units. In Figure 2.2, a

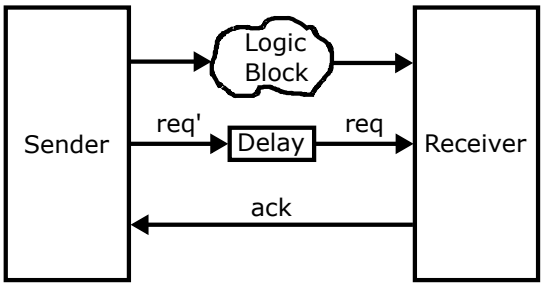


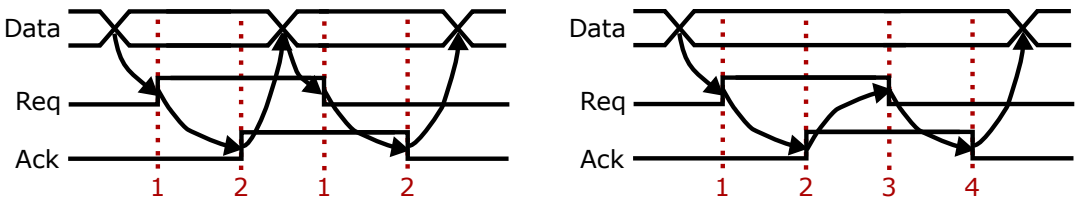
Fig. 2.2 Bundled data

matching delay on the request line between the sender and the receiver is chosen to ensure the validity of the data at the receiving end.

Two protocols of communication are possible on the request and acknowledge channels; these are shown in Figure 2.3. Figure 2.3a illustrates a *two-phase* communication protocol , while Figure 2.3b illustrates a *four-phase* communication protocol.

In Figure 2.3a, a rising or falling transition on the *Request*(Req) line is a request signal indicating the validity of the data, while a rising or falling transition on the *Acknowledgement*(Ack) line is an acknowledgement of the previous request input. In Figure 2.3b, a valid data is signalled by a low to high transition on *Req* which is a request input that must be acknowledged by *Ack* (low to high transition) after which both *Req* and *Ack* must be returned low in the same sequence before another request can be issued.

Bundled data protocol is not very robust because the matching delay may not scale in proportion with the combinational block in the presence of variations [32]. Therefore, the delay line is usually overestimated to improve robustness.



(a) Two-phase protocol (b) Four-phase protocol

Fig. 2.3 Bundled Data Communication Protocols

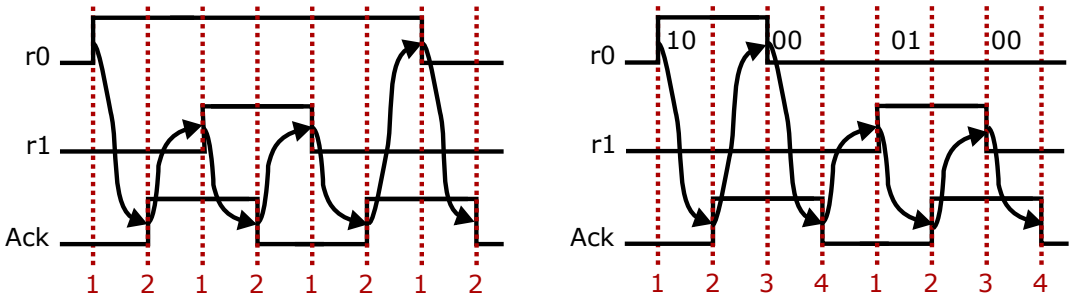
2.3.1.1 Dual-rail encoding

This encoding uses three wires, two to encode request and data and a one wire for acknowledgement to communicate between computing units.

Two protocols of communication are possible on the request and acknowledge channels; these are shown in Figure 2.4a which illustrates a *two-phase* dual-rail communication protocol and Figure 2.4b illustrates a *four-phase* dual-rail communication protocol.

2.3.1.2 Transition signalling protocol

Figure 2.4a shows the dual-rail transition signalling protocol. A transition on either of the request/data wires $r0$ and $r1$ is valid and must be acknowledged by a transition on the acknowledge wire *Ack*. Hence four request states are allowed on the wire pair "$r0$ $r1$": "00", "10", "01" and "11".



(a) Dual-Rail Two-Phase Protocol (b) Dual-Rail Four-Phase Protocol

Fig. 2.4 Dual-Rail Protocol

2.3.1.3 Level signalling protocol

Figure 2.4b shows the dual-rail level signalling protocol. Unlike 2-phase protocol, the data and the request signal can be determined through a low to high transition on only one of the two wire pair $r0$ and $r1$. Once an acknowledge signal (low to high transition on *Ack*) is received, the high signal on the data wire must be returned low. Hence only three states are

valid on the wire pair "$r0$ $r1$": "00", "10" and "01" representing empty/spacer, zero and one respectively.

A designer will have to decide if it is best to use 2-phase or 4-phase signalling in a dual-rail protocol. In making a choice, the increased concurrency coupled with the extra power consumption and delay introduced in 4-phase protocols must be considered. However, 4-phase protocols have the advantage of mapping well into CMOS because they are level sensitive.

The 4-phase communication protocol is used in the self-timed counters used in main contributions of this thesis.

2.4 Formal Specification of Asynchronous Circuits

Asynchronous circuits can be specified in high level or low level forms. A high-level form of the specification includes but is not limited to: Petri Nets, Labelled Petri Nets, regular expression. Low-level specification includes the use of signal transition graphs. The high-level specification allows for easy modelling and verification of asynchronous designs, while actual circuits can be synthesised from the low-level specification. The design of asynchronous circuits can be approached from any level of specification, depending on the complexity of the system. For complex systems, it is best approached in a high to low-level order.

2.4.1 Petri Nets

Petri nets are used to describe and analyse systems with concurrent actions [35]. Petri nets describe the potential behaviour of discrete systems using a directed graph showing possible state transitions which are labelled.

A Petri net (PN) is a tuple $\langle P, T, F, m_o \rangle$ [15] [36] where

$P = \{p_1, ..., p_r\}$ is a finite set of places,

$T = \{t_1, ..., t_r\}$ is a finite set of transitions,

$F \in [P \times T \rightarrow \mathbb{Z}]$ is the incidence function,

$m_o \in [P \rightarrow \mathbb{N}]$ is the initial marking.

A Place P is denoted by a circle; it indicates a condition (preconditions or postconditions) of a Transition. Transitions T are denoted by bars; they represent events. The precondition of a transition is also known as its input place(s), while its postcondition is known as its output place(s).

A transition is enabled when its precondition(s) is/are *marked* (hold a token). An enabled transition will eventually fire, this action consumes token(s) from its input place(s) and adds token(s) to its output place(s). Each place can hold a maximum of one token for $1 - safe$ Petri nets.

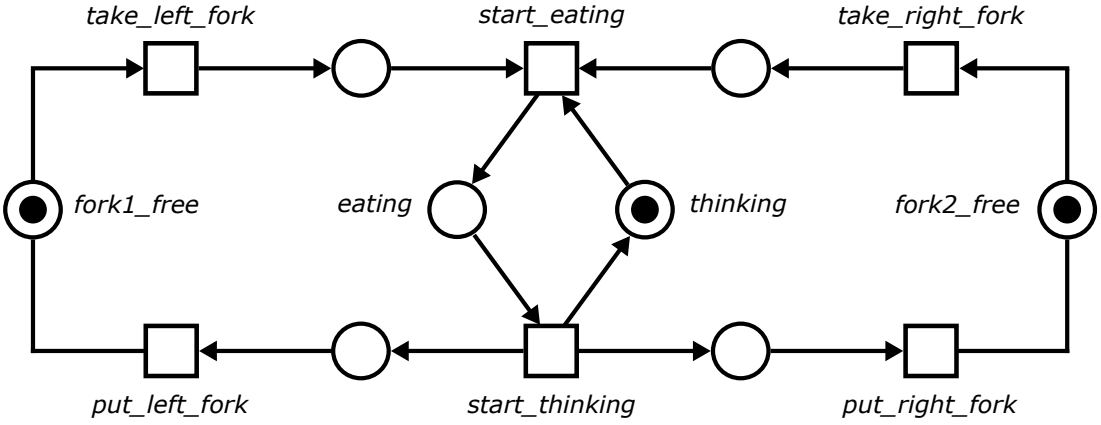
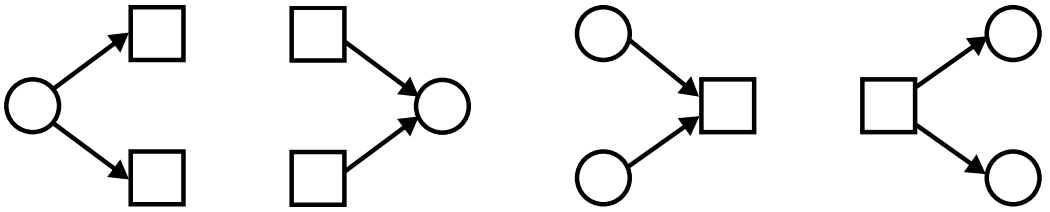


Fig. 2.5 Example: Single dining philosopher Petri net [1]

Figure 2.5 shows a simple PN which consists of eight places, with three places labelled as $fork1\_free$, *thinking* and $fork2\_free$ each holding a token. In this condition, the transitions $take\_left\_fork$ and $take\_right\_fork$ are enabled and each when eventually fired, will consume the token from its input place and output a token in its output place. The firing of both transitions enables transition $start\_eating$.

A Preset ${}^\bullet P$ of a place P is the set of all input transitions of the place P. A Postset $P^\bullet$ of a place P is a set of all output transitions of the place P.

2.4.2 Petri Net Fragments



(a) Choice (b) Merge (c) Join (d) Fork

Fig. 2.6 Petri net Fragments

Figure 2.6 depicts PN fragments. A place with more than one postset transitions such that if any one of the transition fires, it consumes the only token thus disabling other transitions from occurring is called a choice (Figure 2.6a). This definition assumes a $1-safe$ Petri net. A place with more than one preset transition is called a merge (Figure 2.6b). If more than one place has only one transition as a post set, it is called a Join (Figure 2.6c). If more than one place has only one transition as a preset, it is called a Fork (Figure 2.6d).

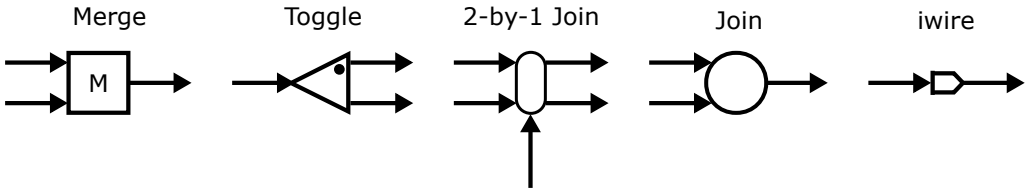


Fig. 2.7 Symbols of Asynchronous Circuits Fragments

Figure 2.7 shows some symbols used to represent standard asynchronous fragments [37]. Two additional fragments are shown, the toggle and iwire. The toggle operates thus, every low to high transition on its input results in a transition in either one of its outputs, with the choice of output alternating, beginning with the output marked by the dot immediately after reset state. The iwire operates thus: it outputs a transition immediately after reset state

after which it must receive an input transition before another output transition can occur. In a 2-by-1 Join, any one of the two inputs on the left enables its adjacent output if a valid transition occurs on the bottom input.

2.4.3 Labelled Petri Nets (LPN)

This is a high-level specification in which names are used to denote events when describing asynchronous systems.

A labelled Petri net is a tuple $N = (PN, S\lambda)$ where PN is a Petri net and $S\lambda : T \to \sum$ is the labelling of transitions.

2.4.4 Signal Transition Graphs (STG)

Signal transition graphs (STGs) are a subclass of interpreted Petri nets which detail the firing of transitions as either a rising or falling of a signal in an ordered sequence. For a transition signal denoted by a, $a+$ means that a transitioned from a logical low to a logical high and $a-$ means vice versa.

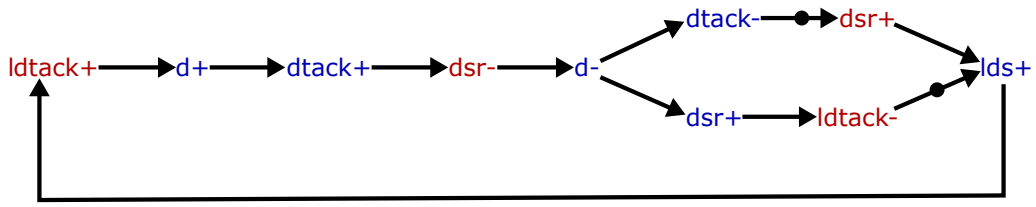


Fig. 2.8 Example: STG of VME Bus Controller [1]

A signal transition graph (STG) is a turple $N = (PN, In, Out, \ell)$ [16] [38]. Where

$Sig \in (In \cup Out)$ Is a set of all signals (input and output),

$\ell : T \to Sig \times \{+, -\}$ is the labelling function,

$Sig \times \{+, -\}$ is the set of signal transitions.

STGs are drawn as Petri nets; however, unmarked pre and post-conditions of a transition can be eliminated, thus allowing an arc to be connected directly from one transition to another when convenient as shown in Figure 2.8.

2.5 Formal Verification of Asynchronous circuits

For an STG to be safe, it needs to satisfy four properties [1]. These are:

- **Consistency:** All signals in any possible trace must transitions from high/low to low/high and then from low/high to high/low before that trace can be repeated.
- **Deadlock-freeness:** No combination of transitions should result in a state where all signals are disabled.
- **Input properness:** An input signal cannot be triggered by internal signals or disabled by internal and output signals. This property ensures that the system responds correctly when a valid input transition occurs.
- **Output persistence:** Internal and output signals must not be disabled by any other signals.

For a circuit to be safe, it needs to satisfy three properties [1]. These are:

- **Conformation:** The circuit must operate as specified by the STG.
- **Deadlock:** In every reachable state, the circuit or the environment (or both) can fire some transition.
- **Hazard:** An excited gate in the circuit cannot be disabled from firing.

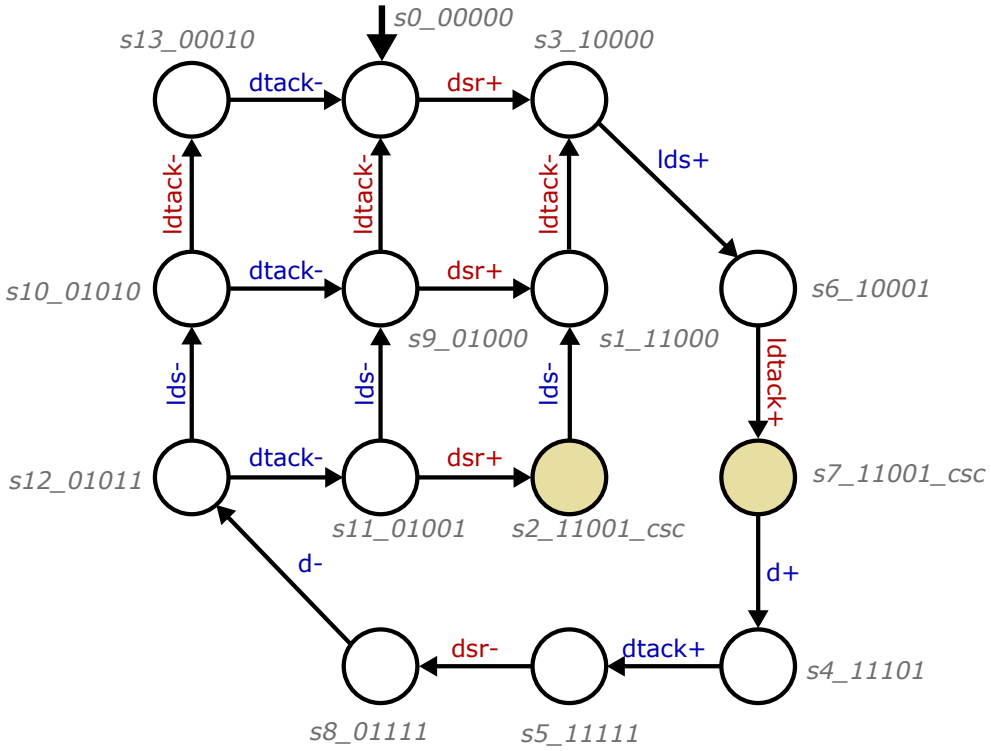


Fig. 2.9 CSC conflicts of STG of Figure 2.8 [1]

2.6 Complete State Coding

Implementability of an STG is determined by its consistency and complete state coding (CSC). Complete state coding guarantees that the state of the system due to each signal transition is unique. Where this is not possible, the STG is said to have CSC conflict(s). Figure 2.9 shows the state coding for the STG of Figure 2.8. The numbers next to the circles denote the state of the system after each transition. The coloured circles have the same coding numbers, and this indicates that those states conflict. This means that when in any of those states, any one of transitions $Ids-$ and $d+$ can occur thus creating a possibility of violating the Conformation property.

One method of resolving this issue is by careful addition of internal signals to distinguish the state coding of the STG at each trace [16, 39].

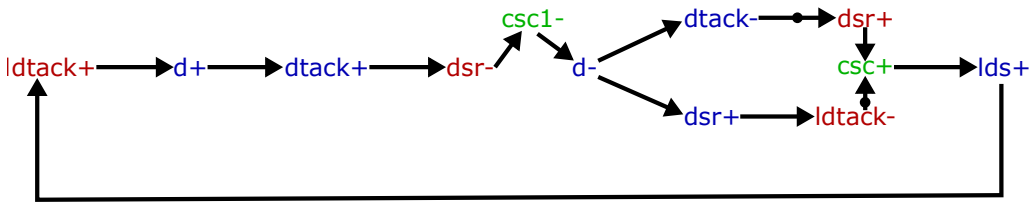


Fig. 2.10 Internal signal used to resolve conflicts [1]

Figure 2.10 shows the modified STG with an internal signal *csc*1 added to eliminate the CSC conflicts. Therefore, a unique state is guaranteed for the system in each transition path. Figure 2.11 shows the new state space graph with an added state for the internal signal.

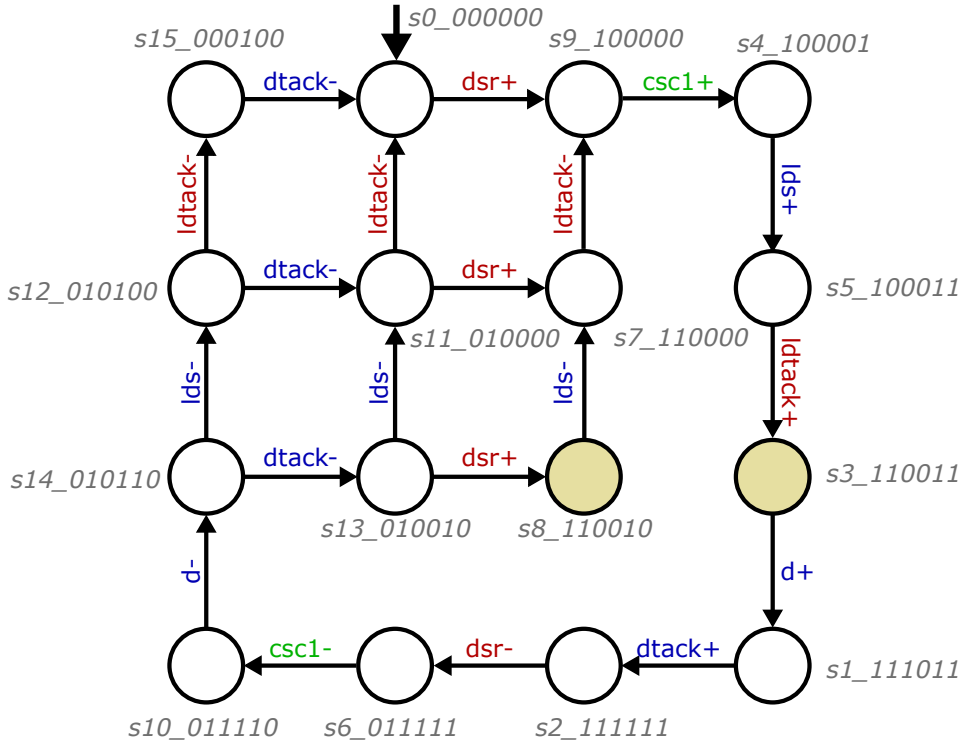


Fig. 2.11 Resolved CSC conflicts of STG of Fig 2.8 [1]

2.7 Synthesis of Asynchronous Circuits from STGs

Usually after behavioural verification and ensuring encoding error free STGs, the next step is to synthesize the logic equations of the described system from the STGs.

In this thesis, the circuits presented were synthesized using the WorkCraft Tool (developed in Newcastle University). WorkCraft uses various back-end tools like the PUnf, MPSat, and PComp for composition and verification of Petri nets and STGs and for synthesis of electronic circuits from STGs. Other back-end tools include Petrify, PGMiner, ScEnco and GraphViz [1].

Other major tools for synthesizing STGs include Teak (developed in Manchester University), which is a tool for creating asynchronous implementations of circuit descriptions written in the Balsa language. Teak transforms concurrent specification of the Balsa language into data driven network [40–42]. Tangram is a tool by Philips, now known as Philips gambit [43–45]. This tool allowed Very Large Scale Integration (VLSI) design to be realised in form of a computer program [46] - Communicating Sequential Protocol (CSP) [47].

2.8 Approaches to Exploiting Asynchronous Properties

Large asynchronous systems can be broken down into a combination of sub-blocks. This approach has the following advantages.

- Reduced design time, efficient testing and verification of each function block. Here, the designer must be aware of the expected input variables, the input combination range and the expected output of each sub-block.
- The behaviour of self-timed systems can be described using mathematical expressions. This allows the system to be decomposed into mathematically correct subsystems which can then be tested, verified and specified using formal asynchronous design methods.
- Modularity of design which allows reuse of sub-blocks in system expansion. This advantage is as a result of the mode of communication in asynchronous circuits using handshake protocols. Thus, the addition or reuse of a verified sub-block in, for instance,

expansion of a design or as part of a different design is not limited by timing issues and technology compatibility [48, 12].

In this section, three methods by which asynchronous circuits properties have been exploited in designing self-timed systems are presented under the following headings: Low Power Circuits, Data-driven Circuits and Circuits Tolerant to timing variations.

2.8.1 Low Power

In [49], power gating techniques were employed to power-down or power-up data-driven sub-blocks depending on the computational needs of the whole system. The power gating controls are determined by the input data encoding. A spacer input (no data) to a sub-block results in no computation, therefore to conserve power a spacer is interpreted as a power down command. This approach exploits the robust intrinsic advantages of QDI circuits (channel encoding).

The need for an asynchronous sleep control circuits for power consumption minimisation was proposed in [50]. In [51], two asynchronous sleep controllers for low power Internet of Things (IoT) are discussed. The controllers are event-driven by incoming data or available energy [52, 53] which are used to wake or put to sleep computing sections in a system.

2.8.2 Robustness to Timing Variations

In [12], the elastic property of asynchronous circuits is exploited in the design of the self-timed micro-pipeline. The pipeline was first built as a modular unit, which is then cascaded to the desired depth. Communication between neighbouring modular units is governed by transition signalling ensuring that data is transferred between computing units that are ready to send and receive data. In this circuit structure, an arbitrary delay in any modular unit does not affect the result of the computation. Therefore in the presence of PVT variation, the circuit internal gates delay will change, but the system will still operate correctly.

2.8.3 Elasticity in Data-Driven Circuits

In [54, 13, 55], event-driven Analogue-to-Digital Converters were presented in which incoming data modulates the operation of the system. Such systems exploited the robust property of asynchronous circuits to tune the granularity of operation from the shape/frequency of the incoming data. The mode of operation of these circuits also gave them a low power consumption profile.

2.9 Design process used in the thesis

In this section, the tools and design process used in this thesis is outlined.

1. Top level specification of the system. This includes formal specification using mathematical expression.

2. From general expression, the system is decomposed into parts that performs specific operation(s).

3. The next step after decomposition is specification of each part operation using Labelled Petri nets (LPN) showing signal names as output and input ports.

4. The LPNs are converted to signal transition graphs (STGs) by refining the signal names into an ordered and interacting sequence of rising and falling transition.

5. The STGs are synthesised using WorkCraft Tool.

   (a) In this stage, the names of the logic gates used in the synthesis were mapped to the logic gates of the standard library to be used in Cadence to minimise gate size and needed connections in the layout implementation.

6. The synthesized logic equation of each part was then imported into Cadence Virtuoso EDA from which their schematics can further be edited.

(a) Each part behaviour is verified by simulation.

(b) The complete system is composed from the verified parts.

(c) The complete system is verified by simulation.

7. After importation to Cadence, the Encounter SoC tool is used to edit the layout of each part and then the complete system.

8. After layout placement, Post Layout simulation is done in Cadence Virtuoso. This step involves design rule checks, layout versus schematic checks and layout extraction before the actual post-layout simulation.

9. The counter presented in chapter 5 was fabricated in 350 nm CMOS technology. After post-layout simulation, the gdsi II files were generated and sent to the foundry for fabrication.

10. Test PCB was designed for the fabricated chip using Altium Designer.

11. Test sequence and patterns for the chip were implemented in an Altera FPGA. The Quartus prime EDA was used for vhdl editing and FPGA configuration.

12. The system was tested and measurement results recorded. This involved the used of Oscillator, low voltage power supply, wave form generators, digital ammeters, pen and paper.

2.10 Conclusion

Basic definitions of fundamental asynchronous circuit terms and parts have been presented in this chapter. This includes classification of asynchronous systems, the methods of event signalling in asynchronous circuits, specification, modelling and synthesis of asynchronous circuits. Asynchronous circuits were shown to offer advantages like low power consumption

and robust operating characteristics. The design of asynchronous circuits is richly supported by proven mathematical methods which aid modelling, verification and synthesis. The advantages of asynchronous circuits can be exploited in the design of low power and data adaptable circuits.

Chapter 3

Counter Decompositions

3.1 Introduction

Digital counters are widely employed in digital and mixed-signal systems like the Phase locked loop (PLL), timers and in systems that require pulse frequency and pulse width control. In these applications, the counter is designed to function as a programmable modulo$-n$ counter.

Generally, digital counters operate in ascending or descending binary order. When applied as modulo$-n$ counting systems they are reset at a predefined value. This describes a counting system that may use an end of count detection circuit to monitor the counter state. Such counters require high-speed count modulo detection circuits. The end of count detection must occur before the next clock transition in synchronous systems. This has led designers to address issues of accuracy, safety of operation and response time of the counter by proposing various decomposition methods.

Some systems require only the end of count output, and therefore, the states through which the counter transit may not be important. This also led to various decompositions of non-binary synchronous and self-timed modulo counters. However, issues of safety of operation and response time of the counter are still major issues to be addressed.

In later chapters in this thesis, two self-timed counters are utilised in the two systems investigated. The counter used in chapter 4 is a self-timed binary counter whose count output is valid only when the acknowledgement signal is produced. This behaviour of the counter was used to model a computing unit with varying task completion time in the investigation presented in chapter 4.

The theory behind the digital pulse-width modulator (DPWM) investigated in chapter 6 is that robustness in asynchronous system can be exploited in the design of a fine-tunable DPWM. Therefore in chapter 5 a loadable self-timed modulo$-n$ counter is realised using formal decomposition, specification and verification methods. In the application of the counter to the DPWM, the counter transition states are not as important as the end of count production. Here I am more interested in the response time and correct/reliable operation of the counter. The self-timed modulo$-n$ counter realised in chapter 5 reconfigures its operation to produce an end of count signal after a time determined by n. Therefore, this chapter on review of counters is a precursor to chapter 5.

Previous approaches to modulo$-n$ counter decompositions were determined by the following three factors; signalling method, sequence of bit transition and end of count detection method.

1. **Signalling protocol**

   Signalling protocol defines the mode of communication of events between sub-unit that make up the counter. This can be synchronous or asynchronous.

   In synchronous modulo$-n$ counters, fast end of count detection time, reduced propagation delay and counter complexity are the main issues addressed in approaches to counter decompositions [56–68]. The aim is to express the counter operation such that propagation delays are eliminated or minimised and the end of count detection reduced to the operation of a fixed circuit or number of gate delays. However quantisation from the coarse nature of the clock can affect the proposed DPWM operation.

In asynchronous modulo$-n$ counters (usually called self-timed counters)[69, 37, 70–77], low power consumption (expressed as bounded), response time on the count outputs and robust operating characteristics are factors that determine the decomposition approach. Asynchronous modulo$-n$ counters have potential application in low power devices such as sleep timers in standby circuitry as proposed in [50]. The first task in asynchronous modulo$-n$ counter decompositions is to express the counter in such a way that counting is generated by internal configuration and signalling of events. The next task is to realise the counter as consisting of basic asynchronous components or a distributed system containing modules assigned a specific counting task which operate in parallel while communicating results with each other. The absence of clock

2. **Sequence of bit transition**

   The sequence of operation defines the mode of operation of the counter as ascending or descending binary order or in any sequence determined by the designer.

   In a binary sequential operation [63, 64, 78, 79, 61], the state of each bit in the counter is important in detecting the end of count transition. In this type of counter, the end of count detection circuit is also decomposed to minimise circuit delay.

   Decomposition of non-binary counters [68, 67, 65, 80] is aimed at a sequence of bit transition, which may be by some logic interactions between the counter bits such that the end of count transition occurs only on a particular bit. Therefore the end of count detection is a task of monitoring that bit.

3. **End of count detection method**

   Two types of detection methods are mainly used to determine the end of count transition in a modulo counter. The first is a completion detection system in which the sequence of operation ultimately results to end of count transition [65, 73, 74, 37, 76, 77]. The second is a comparison detection system in which the state of the counter is

continuously compared to the count modulo. The end of count signal is issued when there is a match [63, 64, 80].

Some synchronous modulo$-n$ counters realised as consisting a completion detection system, operate as one-hot counters in which a bit represents a single state of the counter [68]. Such counters are limited in their counting range.

The modulo$-n$ counters presented in the proceeding sections are grouped under synchronous and asynchronous signalling protocol from which sequence of bit transition and end of count detection methods will be discussed.

3.2 Synchronous Modulo$-n$ounters

In synchronous counters, circuit time margins, and the critical path delay must be considered when choosing the operating clock frequency. Therefore, efforts towards designing counters have aimed at minimizing the effects of the coarse nature of the clock by expressing the counter structure in more efficient ways, a process which is referred to as decomposition. The decomposition methods reviewed here are aimed at reducing the critical path of the counter system and the end of count detection system to a few gates delay. In this section, such counters are presented under the following headings: Prescaled Binary Counters, Pipelined non-binary counters, Multiplexer based and Unary counters.

3.2.1 Prescaled Binary Counters

In the prescaled counter decompositions presented in [63, 64], an $N-$bit counter is divided into predefined bits of sub-counters. Each sub-counter is triggered by the same clock input, however the outputs of each counter correctly updates at distinct frequencies but in ascending binary sequence due to each counter structure.

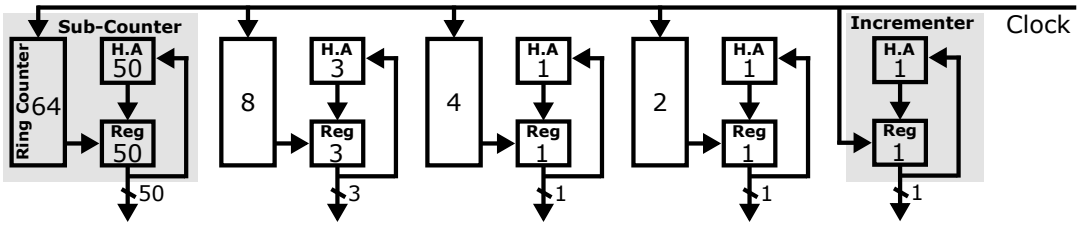


Fig. 3.1 Prescaled Counter

In [63] a binary counter is decomposed into sub-counters, such that the end of count detection delay is minimised to one half adder delay, independent of the counter size.

Figure 3.1 illustrates a 56−bits counter, which is decomposed into sub-counters of different bit(s) sizes. Each sub-counter consists of equal number of bits of half adder(s) (HA) and register(s) (Reg) in a feedback look to form an incrementer which updates on every clock pulse received by the register(s). The number of bits of the HAs and Regs are indicated by the number shown in each. Each sub counter has a dedicated counting range due to its number of bits and operates independent of other sub counters.

The frequency of operation of each sub counter is unique, because the ring counter, acts like a clock divider (based on its number of series flip flop) to the registers in the incrementer system.

The counter structure has an advantage in that is minimises carry propagation delay to the period of one half adder and a register. Therefore, fast detection circuits can be used to detect when each sub-counter has completed its counting operation, with the count modulo generated when all counters have completed counting.

Figure 3.2 illustrates the decomposition method used for the counter, which is applied in top-down partitioning procedure, starting from the total number of bits N, which in this case is 56.

The counter is decomposed by operating on N using the formulas $N - \lceil log_2 N \rceil$ and $\lceil log_2 N \rceil$ to arrive at two sub-counters as indicated by two arrows. The former formula derives the sub-counter which is shown on the left side. This is a resultant sub-counter in

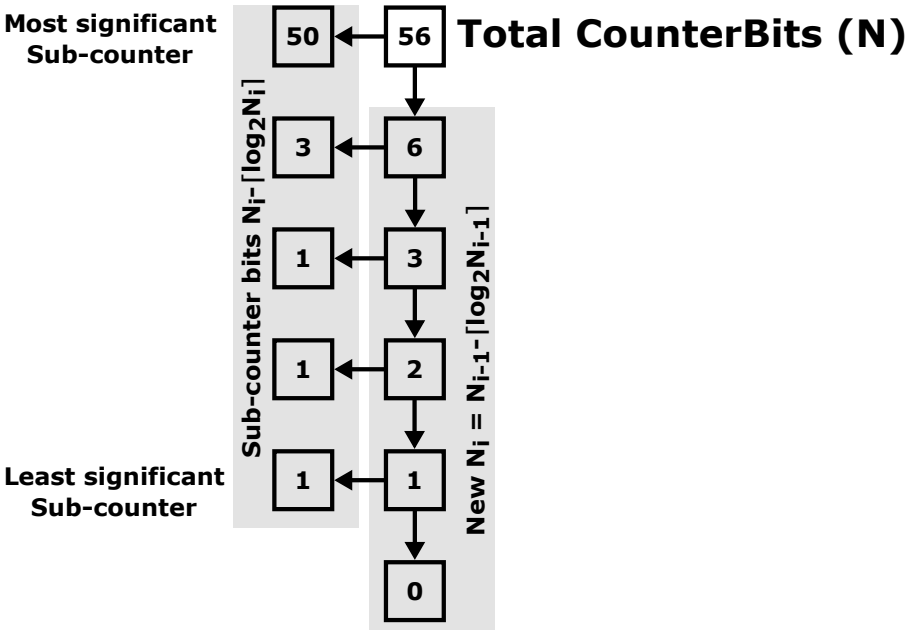


Fig. 3.2 Scaled Sub-Counter Decomposition Tree

the derivation process. The later formula derives a sub-counter which is a new counter bits that can be decomposed further using the two formulas above to get new sub-counters. The iteration ends when the new number of bits is zero. Thus the total number of bits of all sub-counters is equal to N.

The decomposition method presented is used to derive the number of bits of each sub-counter which consists of a ring counter, half-adders and output registers which form an incrementer system.

Each ring counter in Figure 3.1) is a twisted tail counter. For a sub-counter with number of bits derived by $N - \lceil log_2 N \rceil$, the ring counter length is derived by the formula $2^{\lceil log_2 N \rceil}$. This ensures that the output registers of a sub-counter are updated after the defined number of clock pulses such that the complete system operates as an up counter.

Figure 3.3 shows the detection circuit proposed in [63]. It detects when the outputs of a sub-counter and the programmed input match. The correct order of end of count completion detections starts from the most significant sub-counter and occurs in succession till the least significant sub-counter.

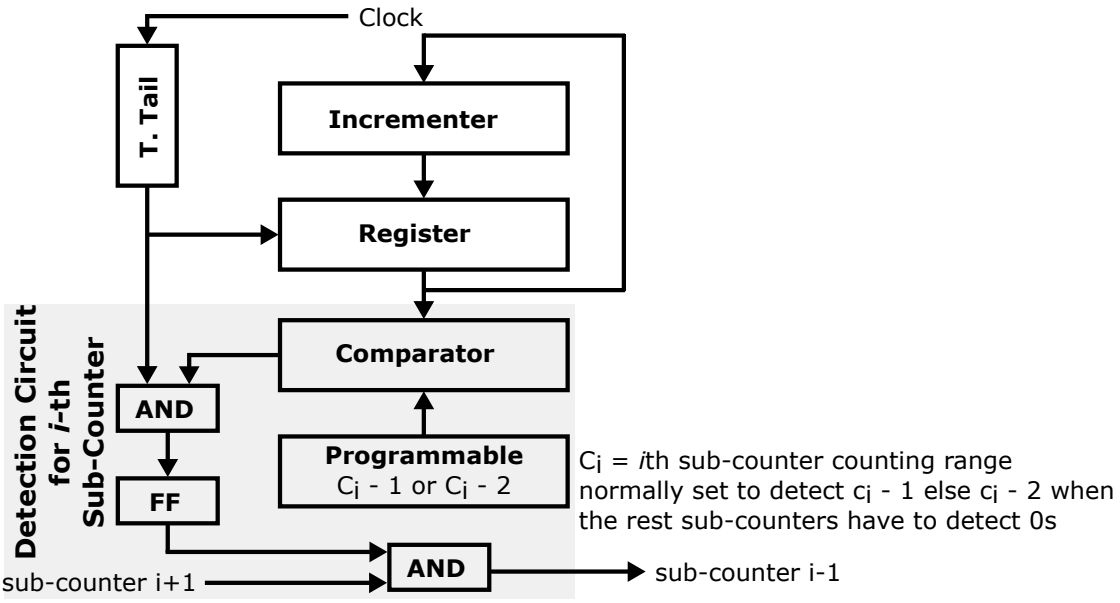


Fig. 3.3 Incrementer circuit which includes programmable unit

The gate delays in the detection circuit are minimised by setting the programmed input of the overall counter to one count lower than the intended count modulo. However, where more than one sub-counters of the lowest significant modules are to detect only zeros, then the system is set to detect two counts lower than the programmed count modulo. Setting the counter to detect one or two counts lower, allows the system to absorb the extra clock period(s) required for the flip-flop and delay in comparator circuits.

This counter will pay heavily for area and consequently increased leakage power and dynamic power consumption (especially the twisted tail counters). In [78, 79], in order to reduce the size of the twisted tail counters, the author modified the decomposition method to include two sets $\{N - \lceil log_2 N \rceil, \lceil log_2 N \rceil\}$ and $\{N - \lfloor log_2 N \rfloor, \lfloor log_2 N \rfloor\}$. The later set is used whenever $N - \lfloor log_2 N \rfloor \leq 2^{\lfloor log_2 N \rfloor}$.

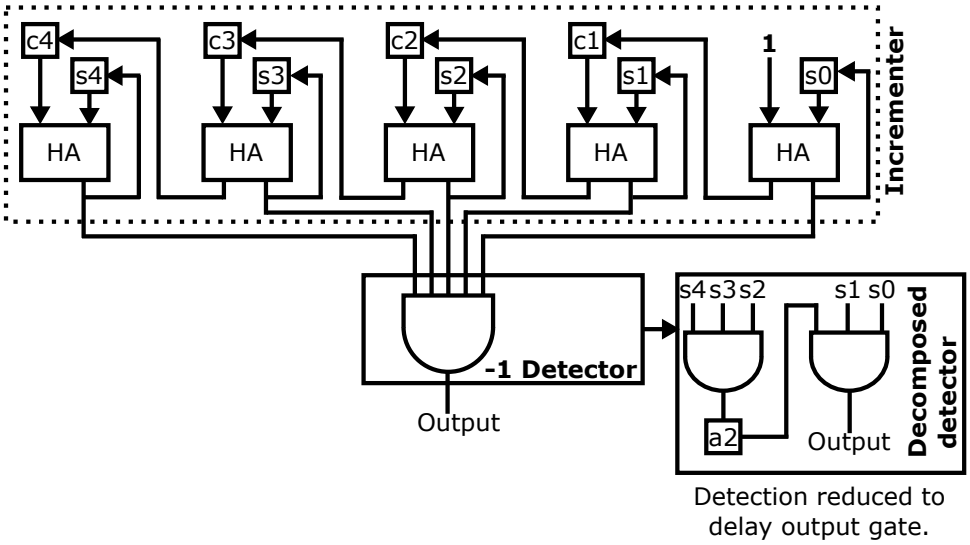


Fig. 3.4 Pipelined Half Adder Based Counter

3.2.2 Partitioned Counters

3.2.2.1 Pipelined Non-Binary Counters

In pipelined non-binary counters, pipelines are employed to reduce carry propagation delay of a ripple-carry counter. The sequence of operation of such counters is not in ascending or descending binary sequence. The end of a count is indicated by a detection circuit sensitive to change in state of a small, fixed section of the counter, irrespective of the count modulo. Such counters also require extra preset circuits to initialise them to zero states at the start of the count. Small area overhead is a major advantage of these type of counters.

The modulo$-k$ counter presented in [65] is an example of a pipelined non-binary counter. It consists of Half adder (HA), registers which hold sum and carry results and a detection circuit to detect when all sum outputs of all half-adders are '1's. For a N-bit counter, the detection circuit is a N-input AND gate. The basic diagram of the counter is shown in Fig. 3.4, with the decomposed detection circuit inset.

Each half adder HA_i has a register at each of its two inputs: sum register s_i which holds the sum output of half adder HA_i and carry register c_i which holds the carry out of adjacent

half adder HA_{i-1}. The carry input to the Least significant bit (LSB) half adder is set to '1'. Each half adder sum register is fed back to its sum input to form an incrementer.

At the start of a count, the sum register is updated to hold the $1's$ complement of the count modulo k. After time t, each half adder sum and carry output will transition to logically reflect its sum and carry inputs. This has to occur before the next transition of the clock which triggers all registers to update to the new input values.

Table 3.1 Table shows scan sequence and state of carry-ins for a modulo−20 counter, starting from $20's$ complement.

Count	**Carry**					**Sum**				
0	0	0	0	0	0	0	1	0	1	1
1	0	0	0	1	0	0	1	0	1	0
2	0	0	1	0	0	0	1	0	0	1
3	0	0	0	1	0	0	1	1	0	0
4	0	0	0	0	0	0	1	1	1	1
5	0	0	0	1	0	0	1	1	1	0
6	0	0	1	0	0	0	1	1	0	1
7	0	1	0	1	0	0	1	0	0	0
8	1	0	0	0	0	0	0	0	1	1
9	0	0	0	1	0	1	0	0	1	0
10	0	0	1	0	0	1	0	0	0	1
11	0	0	0	1	0	1	0	1	0	0
12	0	0	0	0	0	1	0	1	1	1
13	0	0	0	1	0	1	0	1	1	0
14	0	0	1	0	0	1	0	1	0	1
15	0	1	0	1	0	1	0	0	0	0
16	0	0	0	0	0	1	1	0	1	1
17	0	0	0	1	0	1	1	0	1	0
18	0	0	1	0	0	1	1	0	0	1
19	0	0	0	1	0	1	1	1	0	0
20	0	0	0	0	0	1	1	1	1	1

The choice of $1's$ complement as input to the counter ensures that the end of count transition results in the same state irrespective of the count modulo because counting starts from $2^N - 1 - k$ to $2^N - 1$. This is illustrated in Table 3.1 which shows the count sequence for a modulo−20 counter using this approach. Initially all carry registers $c4 - c1$ are "0000",

the sum registers $s4 - s0$ hold the 1's complement of the count modulo 20 ("01011"). The outputs of the half adders are updated to reflect their inputs after a time delay by their circuits. After twenty clock pulses, all half adder outputs are '1's (11111), this causes the AND gate to output a 1.

Although Table 3.1 shows a counter with twenty-one states, count 0 is considered the reset state before any clock pulse. Subsequent states of the counter shows the number of clock pulses received, therefore, the counter detects twenty clock pulses from the initial state.

It can be noticed from Table 3.1 that as the count sequence progresses, the Most significant bits (MSBs) become '1's and remain unchanged for the rest of the count. This transition starts from the MSB, gradually moving towards the LSB. Thus the detection circuit is modified to be sensitive to the two least significant bits. This is illustrated by the decomposed detection circuit on the right side of Figure 3.4. The register $a2$ is updated with the data on its input on the next clock pulse. The delay in the detection circuit is thus reduced to a 3-input AND gate delay.

After the first count sequence, the counter operates as a modulo $2^n - 1$ counter unless its registers are preset to a new value.

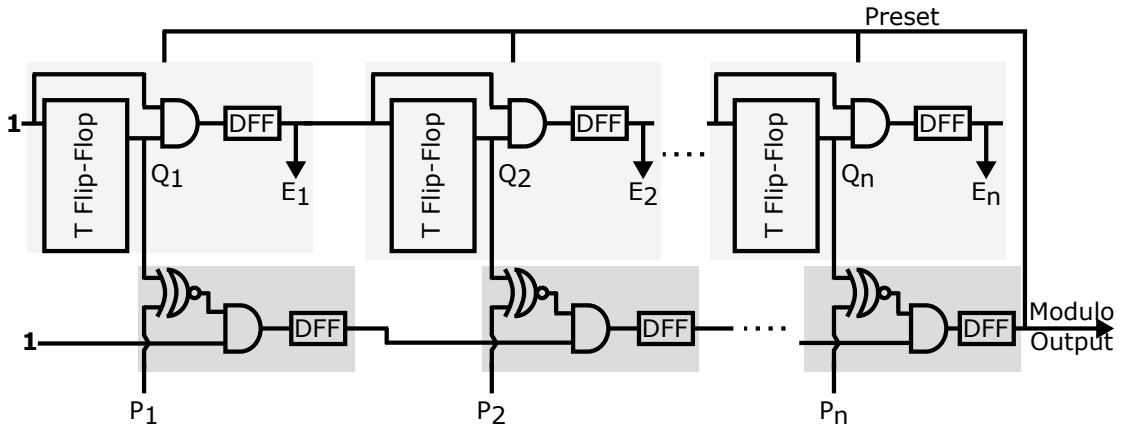


Fig. 3.5 Pipelined Counter with preset logic

Table 3.2 Table shows sequence of scan and state of combinational circuit at each scan.

	4-bit counter								
	3-bit counter								
	2-bit counter								
	1-bit counter								
State	Q_1	E_1	Q_2	E_2	Q_3	E_3	Q_4	E_4	**Comment**
0	0	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	
2	0	1	0	0	0	0	0	0	Zero state for n = 3
3	1	0	1	0	0	0	0	0	Zero state for n = 4
4	0	1	1	0	0	0	0	0	
5	1	0	0	1	0	0	0	0	
6	0	1	0	0	1	0	0	0	
7	1	0	1	0	1	0	0	0	
8	0	1	1	0	1	0	0	0	
9	1	0	0	1	1	0	0	0	
10	0	1	0	0	0	1	0	0	Restart for n = 3
11	1	0	1	0	0	0	1	0	
12	0	1	1	0	0	0	1	0	
13	1	0	0	1	0	0	1	0	
14	0	1	0	0	1	0	1	0	
15	1	0	1	0	1	0	1	0	
16	0	1	1	0	1	0	1	0	
17	1	0	0	1	1	0	1	0	
18	0	1	0	0	0	1	1	0	Restart for n = 3
19	1	0	1	0	0	0	0	1	
20	0	1	1	0	0	0	0	0	Restart for n = 4

Another example of a pipelined non-binary counter is presented in [80]. The counter consists of T flip-flops, AND gates and D Flip-Flop (DFF) as shown in Figure 3.5. Table 3.2 illustrates the sequence of operation of this counter.

Unlike the counter in [65], this counter is capable of repeating the count sequence correctly after the first count sequence because a pre-set logic is activated by the end of count transition.

The major drawback of this counter is that when started from the zero state (all Flip-flop outputs = '0's), this counter has to go through a reset state that is equal to $N-1$ before the

combination of the counter outputs is guaranteed to be unique. Where N is the counter number of bits.

The counter presented in [66] eliminated the zero state problem by replacing the half adders with half subtracters (HS) and instead of counting up, the counter is then made to count down. The detection circuit gives a modulo count output when all counter bits are '0's.

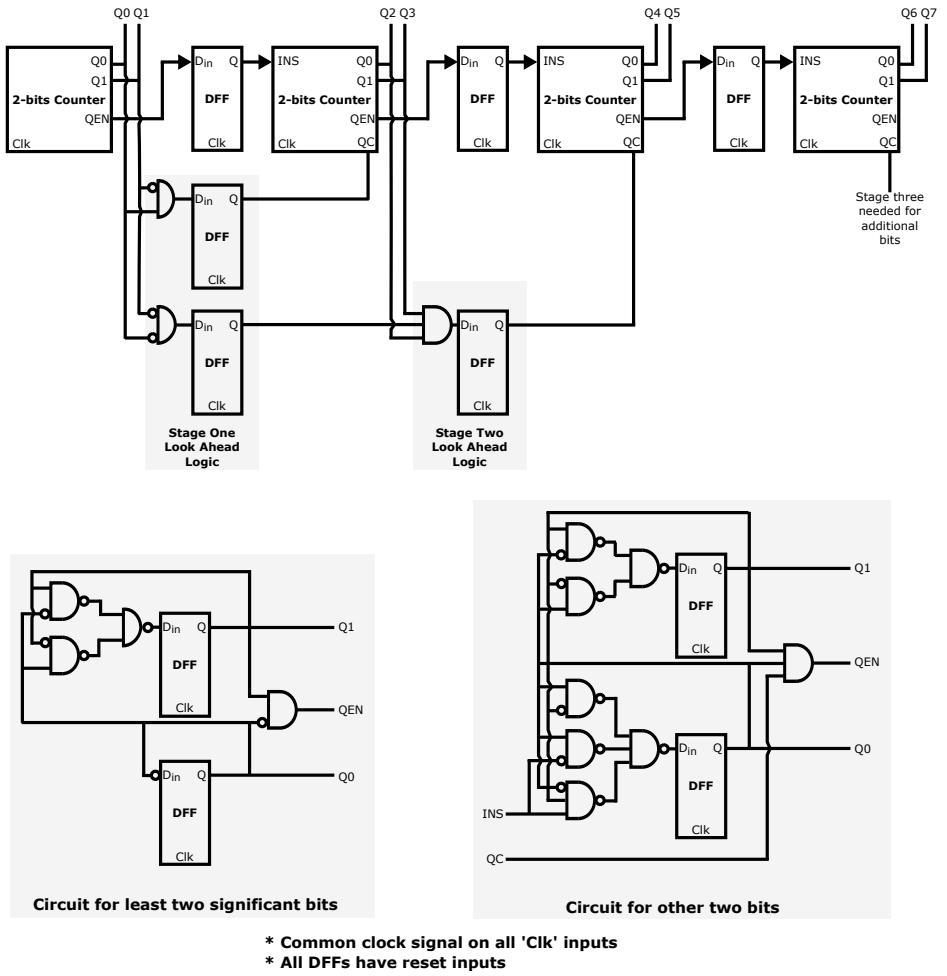


Fig. 3.6 Schematic of the State Look Ahead Logic Counter

3.2.2.2 Pipelined Binary Counters

In more recent counters, look ahead logic is included to speed up the operation of the counter, while maintaining a sequential binary counter [61].

Figure 3.6 is state look ahead counter circuit presented in [61]. The look-ahead logic ensures that the carry output of ith 2-bits counter is enabled two clock cycles early for it to be correctly clocked into the i+1th 2-bits counter D flip-flops. The first clock cycle clocks the carry signal into the DFF before the 2-bits counter; the second clock cycle updates the count output of the concerned 2-bits counter.

The complexity of this counter look ahead logic increases with each addition of two bits counter, resulting in a triangular sequence in which the formula $Total_{LAL} = n(n+1)/2$ gives the total number of lookahead logic. Where n is the number of 2- bits counters added to the first 2-bits counter.

3.3 Asynchronous Modulo$-n$ Counters

One major issue when synchronous counters are employed in modulo counting systems is the need for fast detection circuits. However fast the detection circuit realised is, the clock period must respect the time margin requirements of the circuit to guarantee reliable computation-generally an issue with synchronous systems. This limit is not present in asynchronous systems, there for, decomposition is usually aimed at improving concurrency in counter internal operation and end of count indication, which is usually a completion detection system.

The aim of introducing concurrency is to limit the response time of the counter within some bounds which is the issue usually addressed in asynchronous counter decompositions.

In this section, modulo$-n$ counters are presented under the following headings: Counter decomposed as consisting of even and odd cells, Counter decomposed as consisting of asynchronous circuits fragments, Counter decomposed as consisting of cascaded toggle with completion detection and Counter decomposed as consisting of cascaded toggle with comparison block as end of count detector.

The first step in major modulo$-n$ counter decompositions is to specify the general counter behaviour using a regular expression which is a standard form of expressing the property or operation of a system. In this form, the system specification can be rewritten to express it in the desired decomposition form.

3.3.1 Decomposition to interacting cells

In this decomposition approach, the counter is viewed as consisting of independent and interacting sub-unit called cells [14, 76, 77]. Each cell has specific operation in a count sequence, which may change when the count sequence changes. Modulo detection is built internally as each cell can indicate its task completion in a count sequence.

The general expression from which counters in this category are derived is shown in equation 3.1.

$$(a!^n ; b!)^* \tag{3.1}$$

Equation 3.1 is a self-timed system having two ports a and b which are output ports as denoted by the symbol !. Input ports are denoted by the symbol ?. The symbol $a!^n$ means n successive occurrences of an output event on port a. The specification thus states that after n events on port a an event will occur on port b. The occurrence of an event on b indicates the completion of n count sequence in the counter. The notation $(\ )^*$ indicates arbitrary repetition of the count sequence.

3.3.1.1 Kees van Berkel *et al*.

In [76, 77] the behaviour of the modulo$-n$ counter was approached as specified in equation 3.1. The decomposition method used divided the counter into a Head cell and a modulo-(N div 2) sub-counter. Each sub-counter is further divided using the same method. Thus the

counter is decomposed into combinations of even, odd and one count sub-counters. The decomposition of the counter was achieved by rewriting the count modulo n.

For example, $n \triangleq n\,mod\,2 + 2 \times n\,div\,2$. Therefore equation 3.1 can be rewritten as shown in equation 3.2:

$$(a!^{n\ mod\ 2}\,;\,(a!\,;\,a!)^{n\ div\ 2}\,;\,b!)^* \tag{3.2}$$

The term $a^{n\ mod\ 2}$ can either mean a^0 or a^1 depending if n is even or odd.

From equation 3.2, a counter consisting of two sub-counter systems can be formed. The first sub-counter referred to as the Head counter doubles every communication on its a' input port from the second sub-counter (N div 2). However, it directly passes any communication on b' to port b. The counter system is shown in figure 3.7.

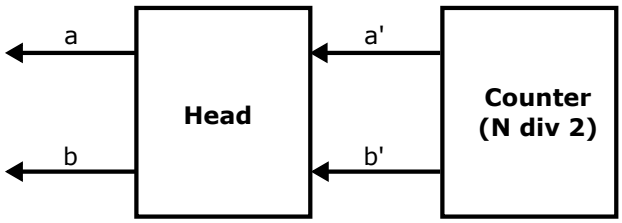


Fig. 3.7 Modulo-N counter showing Head and Count (N div 2) blocks

For odd n, the head counter operation can be expressed as shown in (3.3).

$$(a!\,;\,(a'?\,;\,a!\,;\,a!)^*\,;\,b'?\,;\,b!)^* \tag{3.3}$$

Equation 3.3 specifies the head counter's operation to first output an event on its a channel; then the system goes into a loop in which an input event on a' channel results in double events on a output channel. The system exits the loop by an input on b' which results in an output on the b channel. After this, the process starts again from the beginning.

This operation of (3.3) can be rewritten thus: $(a\ (a'\ a\ +\ b'\ b))^*$. For every repetition of the count sequence, a is first produced. The next output depends on either of inputs a' or (denoted by the '+' sign) b' from the COUNT (N div 2).

Similarly for even N, the head counter expression can be rewritten as

$$((a'?\ a!\ a!)^*\ b'?\ b!)^* \tag{3.4}$$

Rewriting, equation 3.4 becomes $(a'\ a\ a\ +\ b'\ b)^*$. For every repetition of the count sequence, the next output depends on either of inputs a' or (denoted by the '+' sign) b' from the COUNT (N div 2).

Using Tangram notation the expressions for even and odd counts were programmed.

Even counter in Tangram is written as $CE(a,b,a',b') = \#[[a'\ ;\ a\ ;\ a \mid b'\ ;\ b]]$ and odd counter is written as $CO(a,b,a',b') = \#[a\ [a'\ ;\ a \mid b'\ ;\ b]]$.

Thus the program for a modulo$-n$ counter becomes a combination of even, odd and $COUNT_1$ cells. A modulo 18 counter, when decomposed, will result in:

The decomposition for count 18 $(10010_2)$is as follows:

$|[a_1,b_1,a_2,b_2,a_3,b_3,a_4,b_4\ chan$

$\mid CE(a,b,a_1,b_1) \mid\mid CO(a_1,b_1,a_2,b_2) \mid\mid CE(a_2,b_2,a_3,b_3) \mid\mid CE(a_3,b_3,a_4,b_4) \mid\mid COUNT_1$

$]|$

It can be seen that the number of cells needed to realise a modulo-n counter is equal to the number of bits (when n is converted to binary) or by using the formula $\lfloor Log_2 n \rfloor + 1$. It can also be seen that cells corresponding to a '1's operate as odd cells, while cells that correspond to a '0's operate as even cells.

From equation 3.3, it can be seen that some level of parallelism can be obtained from the odd Head cells since the first output on port a occurs without input from its sub-counter. From equation 3.4, the output of the even Head cell would have to wait for input from its

sub-counter. Thus the initial response time of this type of counter is not constant but has an upper bound defined by $O(log_2 n)$, where n is the number of bits.

The decomposition method presented in [76, 77] was first presented by Kessels [14].. Kessels decomposed a cell to consists of two parts. One part performed a multiply by two operation and the other part performed a plus one or plus zero operation. Kessels method was also adopted by Alex [81] to lay the framework of the loadable counter presented in chapter 5. More on Kessels and Alex approaches in chapter 5 introduction.

3.3.2 Decomposition to asynchronous circuit fragments

In this decomposition approach, the counter is viewed as consisting of interacting standard asynchronous fragments.

The general equation from which counters in this category are derived is shown in equation 3.5.

$$pref * \left[(a?\,;\, p!)^{n-1}\ ;\ a?\,;q!\right] \tag{3.5}$$

Equation 3.5 expresses a system that accepts a request on port a denoted by the input symbol ?. The system tracks the number of requests received and responds to each request with an acknowledgement on port p when the total request is less than n and an acknowledgement on q when the total request is equal to n. The notation $pref * [\,]$ indicates arbitrary repetition of the count sequence after an event occur on q.

3.3.2.1 Ebergen *et al*

Ebergen *et al* in [37], showed three methods of decomposing the Modulo$-n$ counter described in equation 3.5 into basic handshake components. The decomposition approaches used arrived at a one hot encoded counter, a divide by two counter and a counter with some level of parallel operation. These are presented below.

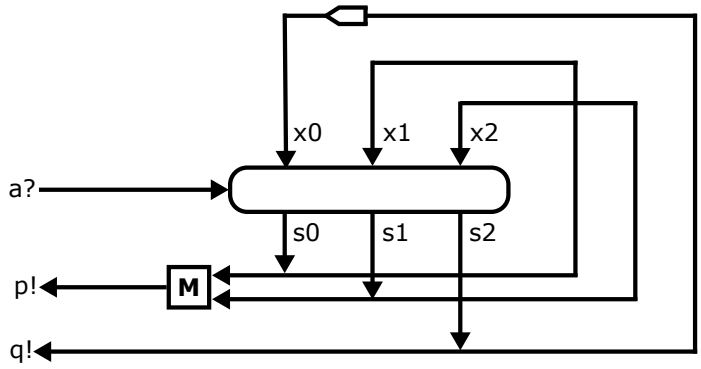


Fig. 3.8 Modulo−3 Counter decomposed into standard asynchronous fragments

One-hot For $n = 3$, the counter is decomposed by first assigning a label to each state of the counter. The decomposition approach is aimed at ensuring that state transition occurs on only one label. Figure 3.8 shows the decomposed counter for $n = 3$. The decomposition process is thus presented by expanding (3.5) to $pref * \{a?\,;\, p!; a?\,;\, p!\, a?\,; q!\}$

= $\{Introduction\ of\ internal\ symbols\ s0,\ s1,\ s2\}$

The aim of introducing these internal signals is to arrive at a one-hot (unique state) representation of each input on a. In this case we have three states ($n = 3$). This gives:

$|[(s0, s1, s2 ::$

$pref * [a?; s0; p!; a?; s1; p!; a?; s2; q!]$

$]|$

The above expression states input transitions on a transit the system from state $S0$ through $S2$. The counter sends an output transition on p when it transits to states $S0$ or $S1$ while an output transition occurs on q when it transits to states $S2$.

The expression above when unfolded yields:

$|[(s0, s1, s2 ::$

$pref(a?; *[s0; p!; a?; s1; p!; a?; s2; q!; a?])$

$]|$

Since each input on a results in a different state, a suitable component for this operation is the $N-by-1$ join. To arrive at a $N-by-1$ JOIN, another set of internal symbols are introduced as inputs to the $N-by-1$ join to be selected by the input trigger (a).

= { Introduction of internal symbols $x0,\ x1,\ x2$ heading for 3-by-1 JOIN }

$||[s0, s1; s2; x0, x1, x2 ::$

$pref((a?\,||\,x0); *[s0; (p!; a?)\,||\,x1); s1; ((p!; a?)\,||\,x2); s2; ((q!; a?)\,||\,x0])$

$]||$

The above expression shows that the counter next state depends on its current state for every input on a.

The following decompositions can be deduced from the interaction shown by the expression above.

{ Decomposition guided by syntax }

$(pref * [(a?\,||\,x0); s0!\,|\,(a?\,||\,x1); s1!\,|\ (a?\,||\,x2); s2!]\,\{3-by-1\,JOIN\}$

$pref * [s0?; x1!]\,\{Wire\}$ (output from state 0 is fed to select input of state 1).

$pref * [s1?; x2!]\,\{Wire\}$ (output from state 1 is fed to select input of state 2).

$pref * [x0?;\ s2!]\,\{IWire\}$ (at counter initial state, state 0 is enabled, after which it can only be enabled when counter gets to state 2).

$pref * [(s0?\,|s1?)\,;\ p!]\{Merge\}$(outputs from states 0 and 1 are merged to p output)

$pref * [s2?; q!]\ \{Wire\}$(State 2 is the end state of counter operation, thus its sent out on the q output).

A general formula can be used to represent this type of decomposition. For n, $n \geq 3$ a modulo$-n$ counter can be decomposed into n-by-1 JOIN, a (n-1) input merge, an iwire and some wires. It can be seen that the merge inputs grows with n.

Divide and conquer Figure 3.9 shows the component of the second approach to modulo$-n$ counter. It contains a Modulo$-m$ counter which counts half of the count modulo n ($m = n/2$) and a subcomponent E which also counts half of the count modulo.

The Modulo$-m$ counter is sensitive to external events from the counter system (Modulo$-m$ counter and E) through its input port $s.a$. It communicates every count received to E on two output ports $s.p$ and $s.q$ (only sends a transition on at the end of the count). It operates as specified in equation 3.5.

The subcomponent E has two output ports p and q. It passes every second output of the $s.q$ output of the Modulo$-m$ counter to its q output. When this occurs, it indicates that the nth count has occurred.

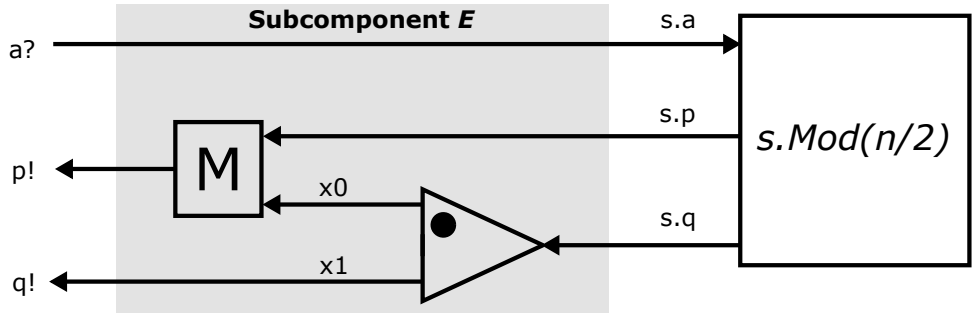


Fig. 3.9 Counter decomposed into two sub-counters E and s.ModC(M)

The decomposition steps shown below detail the origin of the subcomponent E.

Expanding equation 3.5 to include the Modulo$-m$ counter, the new expression is shown below

pref*$[(a?\ ;\ p!)^{m-1}\ ;\ a?\ ;\ p!\ ;\ (a?\ ;\ p!)^{m-1}\ ;\ a?\ ;\ q!]$

{ heading for $s.ModC(m)$, introduction of internal symbols $s.a$, $s.p$ and $s.q$ } (symbols are used to represent interaction between E and the modulo$-m$ counter).

$|[s.a,\ s.p\ s.q,\ ::$

$pref * [(a?\ ;\ s.a\ ;\ s.p\ ;\ p!)^{m-1}\ ;\ a?\ ;\ s.a\ ;\ s.q\ ;\ p!\ ;\ (a?\ ;\ s.a\ ;\ s.p\ ;\ p!)^{m-1}\ ;\ a?\ ;\ s.a\ ;\ s.q\ ;\ q!]$

$|]$

= { introduction of internal symbols $x0$, $x1$ to distinguish occurrences of $s.q$ }

$|[s.a,\ s.p,\ s.q,\ x0,\ x1\ ::$

$pref * [(a?\ ;\ s.a\ ;\ s.p\ ;\ p!)^{m-1}\ ;\ a?\ ;\ s.a\ ;\ s.q\ ;\ x0\ ;\ p!\ ;\ (a?\ ;\ s.a\ ;\ s.p\ ;\ p!)^{m-1}\ ;\ a?\ ;\ s.a\ ;\ s.q\ ;\ x1\ ;\ q!]$

||]

{ decomposition guided by syntax }

(pref*[($s.a?$; $s.p!$)$^{M-1}$; $s.a?$; $s.q!$] {$s.Mod(M)$ } (modulo−M counter responds to all inputs, $M-1$ or less on the p output and the Mth on the q output).

pref*[$a?$; $s.a!$] {Wire}

pref*[($s.p$? | $x0?$); $p!$] {Merge}

pref*[$x1?$; $q!$] {Wire}

pref*[$s.q?$; $x0!$; $s.q?$; $x1!$] {Toggle}

).

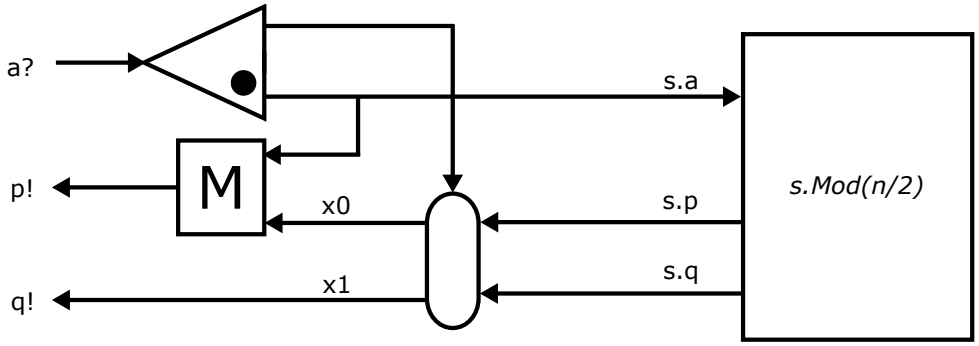


Fig. 3.10 Counter decomposed into parallel operating parts

Introducing parallelism The counter decomposed above is limited by the value of the count modulo n. It cannot operate correctly when n is odd. In the next decomposition, parallelism is introduced. It ensures that every odd number count is not received by the modulo−m counter and at the same time processed to the output through the merge. The modulo−m counter generates a response on one of its two output ports $s.p$ and $s.q$ for every count it receives.

Equation 3.5 can be rewritten as pref*[($a?$; $p!$)$^{n-1}$; $a?$; $q!$]

= { Calculus, $n > 1$, n even, $m = n/2$ }

pref*[($a?$; $p!$; $a?$; $p!$)$^{m-1}$; $a?$; $q!$]

= { heading for $s.ModC(M)$, introduction of $s.a$, $s.p$ and $s.q$ }

|[$s.a$, $s.p$ and $s.q$ ::

pref*[($a?$; ($p!$; $a?$) || ($s.a$; $s.p$) ; $p!$)$^{m-1}$; ($a?$; ($p!$; $a?$) || ($s.a$; $s.q$) ; $q!$)

]|

= { introduction of $a0$, $a1$ and x to distinguish different occurrences of a and p }

pref*[($a?$; $a0$; ($p!$; $a?$; $a1$) || ($s.a$; $s.p$) ; x ; $p!$)$^{m-1}$; ($a?$; $a0$; ($p!$; $a?$; $a1$) || ($s.a$; $s.q$) ; $q!$)

]|

{ decomposition guided by syntax }

(pref*[($s.a?$; $s.p!$)$^{m-1}$; $s.a?$; $s.q!$] {$s.ModC(m)$ }

pref*[$a?$; $s.a!$] {Wire}

pref*[($a0$? | $x?$); $p!$] {Merge}

pref*[$a0?$; $s.a!$] {Wire}

pref*[$a?$; $a0!$; $a?$; $a1!$] {Toggle}

pref*[($a1?$ || $s.p?$; $x!$) | $a1?$ || $s.q?$; $q!$)] {2-by-1 JOIN }

).

In the counter, shown in Figure 3.10 parallel operations occur at every odd input on a, which causes an acknowledgement on p through the merge and a count operation in the sub counter. Due to its asynchronous mode of operation, the result of the sub counter on $s.p$ output will be ready when the next input occurs, to be selected by the 2-by-1 join.

3.3.3 Cascaded toggle with completion detection

The main components of the modulo$-n$ counter presented in [73, 74] are toggle(s) and a completion detection circuit.

The STG of Figure 3.11 illustrates the sequence of operation of the toggle symbol shown on the left. The STG shows the initial state of the toggle as applied in this modulo counter system. At initial state, both outputs of the toggle are high. A high to low transition on the

input E results in a high to low transition on either $U1$ or $U2$, with more priority given to $U1$ if both $U1$ and $U2$ a high. Similarly, a low to high transition on the input E results in a low to high transition on either $U1$ or $U2$, with more priority given to $U1$ if both $U1$ and $U2$ are low.

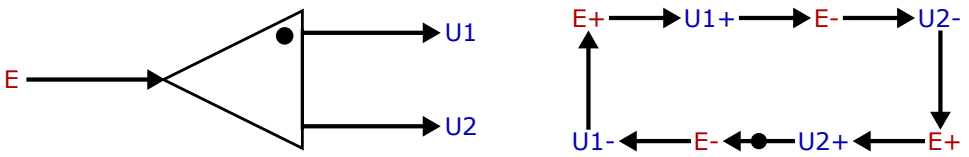


Fig. 3.11 Toggle Symbol and STG illustrating the sequence of operation from initial state

The modulo−n counter is formed by series cascading of the N number of toggles along the $U2$ output. The maximum count output from N series toggle is given by the expression 2^N. Alternatively, the number of cascaded toggles needed to form a modulo−n counter is given by the formula $\lceil Log_2 n \rceil$.

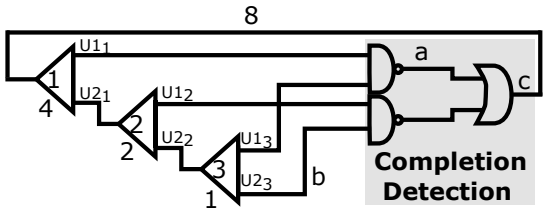
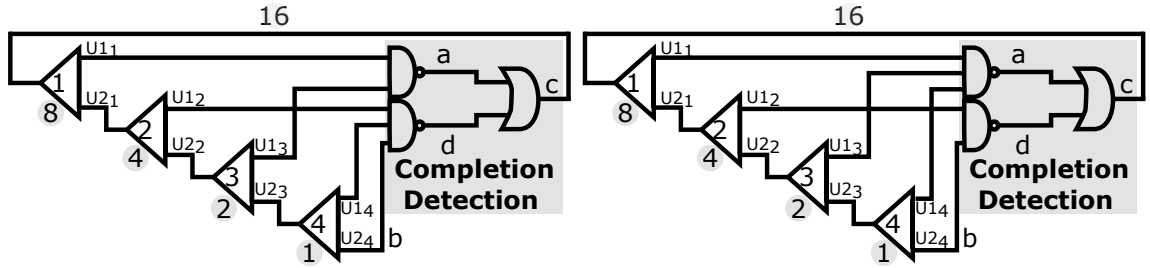


Fig. 3.12 Three-stage Cascaded Toggle showing count output at point a and Completion Detection on point b.

The example shown in Figure 3.12 is a modulo−5 counter with the output gotten from point a. At initial state, $U1_1, U1_2, U2_1, U2_2, U3_1, U3_2 =' 1'$, $a, b =' 0'$ and $c =' 1*'$. The OR gate is excited and C will transition to a '0', causing $U1_1$ to also transition to a '0' after which the completion detection at point a will transition to a '1'. The counter has no stable state as each transition leads to an excited gate. The effect of excitation and firing a gate causes transitions to logically occur on all outputs of the toggles one at a time.

However, the state of the counter is always unique after each count transition. A low to high transition can only occur at point *b* after five low to high transitions at point *a*.

To obtain a different count ratio between point *a* and *b*, a change in the structure of the detection circuit is needed.



(a) Modulo−10ounter (b) Modulo−11ounter

Fig. 3.13 Four stage Cascaded Toggle and Different Completion Detection Circuit to form a 10/11 to 1 output on points *a* and *b*

For example, a four stage cascaded toggle can be used to form a modulo$-1to16$ counter as illustrated in Figure 3.13 with different completion detection circuits for modulo−10 counter (Figure 3.13a) and modulo−11 counter (Figure 3.13b). In a complete count sequence, shaded numbers underneath each toggle is the number of transitions that occur in its output.

Table 3.3 Table shows the logic for completion detection circuit for each count output on point *a* and *d*

Four Stage Toggle			
Count outputs		**Toggles output to CD NAND(s)**	
a	***d***	**NAND for *a***	**NAND for *d***
8	7	¬(8 ∧ b)	¬(4 ∧ 2 ∧ 1)
9	6	¬(8 ∧ 1 ∧ b*)	¬(4 ∧ 2 ∧ b*)
10	5	¬(8 ∧ 2 ∧ b*)	¬(4 ∧ 1 ∧ b*)
11	4	¬(8 ∧ 2 ∧ 1)	¬(4 ∧ b)
12	3	¬(8 ∧ 4 ∧ b*)	¬(2 ∧ 1 ∧ b*)
13	2	¬(8 ∧ 4 ∧ 1)	¬(2 ∧ b)
14	1	¬(8 ∧ 4 ∧ 2)	¬(1 ∧ b)
15	-	¬(8 ∧ 4 ∧ 2 ∧ 1 ∧ b)	-

* b can be input to either one of both NAND gates
CD = Completion Detection

Table 3.3 lists the count combination for points a and d and the input combination to their NAND gates of the completion detection circuits. The numbers shown add up to the count outputs before a single output on point b.

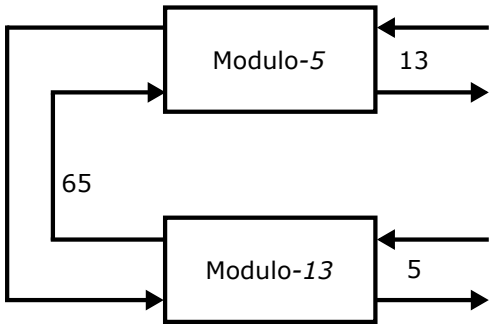


Fig. 3.14 Interaction of a modulo−5 counter and a modulo−13 counter to form a rational modulo $5/13$ counter

A modulo−$5/13$ counter can be formed using two of such toggle counters as shown in Figure 3.14.

3.3.4 Cascaded toggle with comparison circuit

Figure 3.15 is a modulo−n counter presented in [75]. This counter has two main blocks, a compare block and a toggle block. The operation of the counter is listed below:

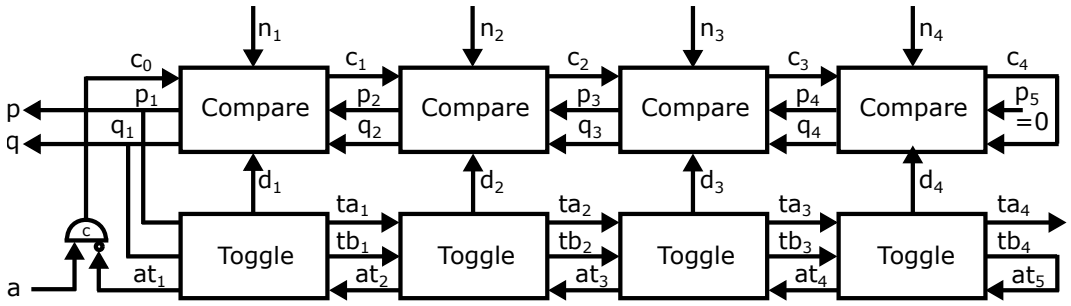


Fig. 3.15 Block diagram of the compare and toggle modulo−n counter

1. At initial state, C_0, at_1, p and q are all '0'.

2. The intended count modulo in binary is input on n_1 to n_4 (LSB to MSB)

3. A count request is received from the environment on input a, this causes C_0 to transition to a '1'.

4. The input C_0 is a compare command to the first compare cell.

5. The first compare cell has the following options:

   (a) If n_1 and d_1 are the same, it issues a compare command to the next compare cell, which does similar in operation comparing n_2 and d_2. All compare cell are similar.

   (b) If n_i and d_i are not the same, it returns an acknowledgement on p_i which will eventually appear on p_1.

6. An acknowledgement on p_1 causes the first toggle cell to transition to a '1' if d_1 was a '0' else all toggles are flipped in succession until a toggle is flipped from '0' to '1'.

7. After flipping, each toggle affected will return an acknowledgement on the at_i line, starting from the most significantly flipped toggle.

8. The request to the compare block on C_0 can only be lowered when at_1 = '1' and request a = '0'. After which the ack on p_1 will be lowered.

9. If all bits compared are the same q_1 will transition to a '1'. This is an acknowledge signal to the environment and it also causes a reset of all toggles to '0's.

10. The reset to all all toggles is also acknowledged on the at_i line.

3.4 Conclusion

In this chapter, decomposition approaches to synchronous and asynchronous modulo$-n$ counters were presented.

The major aim of synchronous counter decompositions is to arrive at a fast count transition and detection of the end of count. The counters decomposed either consumed small area and were non-binary in operation thus requiring complex detection systems [65, 67, 80] or consumed large area when the counter operated in ascending or binary sequence [78, 79, 63, 64].

The main issues with synchronous counters are that the clock period is limited by the time margins requirements of logic blocks and critical path in the counter, making them susceptible to quantisation from the clock. While simple decomposition was presented in [65, 67] extra circuits would need to be added to make the counter programmable.

Since asynchronous circuits operate without a global clock, but by explicit signalling of events decomposition of asynchronous counters become an issue of efficiently expressing the counter as consisting of asynchronous fragments or even and odd counter cells. The counters decomposed as asynchronous fragments [37, 73–75] were simple and had fixed structures and thus were not loadable counters. The counter decomposed as consisting of even and odd cells provided bounded response time on its outputs, but were not detailed enough to show how each cell can be specified to perform either an odd or even operation. This is essential for the loadable counter proposed for the implementation of the asynchronous digital pulse-width modulator in chapter 1. Therefore, the task is to efficiently decompose and specify the counter such that the operation of each cell can be reconfigured by the loaded count modulo.

Chapter 4

Adaptive Digitally Controlled Oscillator based on Self-Timed Counter

4.1 Introduction

In a typical synchronous system, the clock generation circuit operates separately from the triggered computing units. For example, a counter which tracks the occurrence of an event may be a different circuit from the source of the event.

In this set-up, the clock period must be large enough to ensure that the time margin requirements of the triggered circuits are not violated when subjected to process, voltage and temperature (PVT) variations. In some cases, this may require dynamic tuning of the clock frequency, which is non-trivial because changes in circuit gates delay does not scale in direct proportion to PVT variation.

The main proposal behind this chapter is that a robust system can be realised by subjecting the clock generator operation to the operating conditions of the triggered computing units. Systems level approaches have been proposed like the Dynamic voltage scaling and dynamic frequency scaling approaches [3–9].

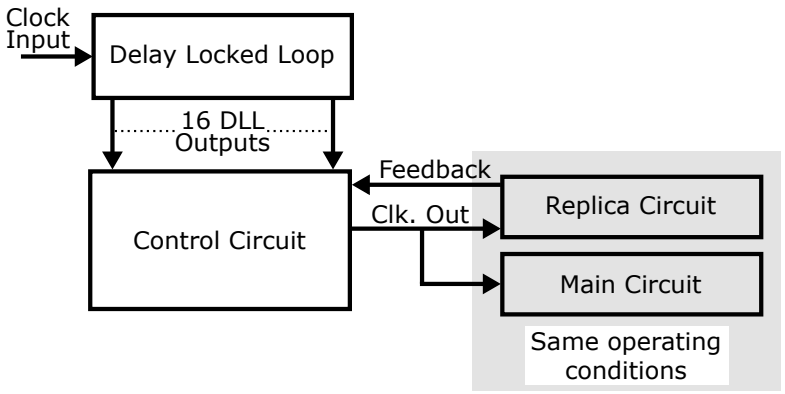


Fig. 4.1 Block diagram of the adaptive system proposed in [2]

In [2], the clock frequency is tuned by a circuit that tracks a change in operating voltage supply. An operating threshold is set, so that when the voltage supply falls below the threshold, the clock frequency is slowed down. The clock frequency is restored once the voltage supply rises above the threshold voltage.

This adaptive function is achieved with the use of an externally triggered multiple phase output (sixteen phase outputs) Delay locked loop (DLL) which feeds a control circuit, a replica circuit which is a tunable delay line whose input and output is in a feedback loop with the the control circuit as shown in Figure 4.1.

The replica circuit is made subject to the same operating conditions of the computing circuit to be clocked. The controller monitors the sixteen outputs of the DLL and the feedback from the replica circuit using an edge detector and edge monitor to give an output frequency command to the clock generator based on any differences.

This approach is limited in that the response time of the controlled clock output is a few cycles after a trigger is received to change the clock frequency. Also, a watchdog circuit is needed to resolve metastability which cannot be avoided in this type of circuit and it is almost impossible to absolutely resolve metastability. The watchdog circuit bounds the decision time of the controller. It ensures that after two clock pulses without a decision from the controller, the input clock signal is recovered in the circuit. The effects of PVT variations

on the replica circuit and the computing unit may not always be the same for all operating conditions since their circuits are different [83].

In [10], the clock frequency is tuned with the use of a time borrowing circuit that detects when there is a mismatch between the clock period and the critical path in a logic circuit. The time borrowed is used to stretch the clock period from a controlled Phase Locked Loop (PLL).

In the above mentioned methods of adapting the clock operation to the conditions of the triggered computing unit, a sensing and control mechanism is needed. This approach requires that the issues of system response time, metastability and sensing precision are addressed.

In the approach investigated in this chapter, the Digitally Controlled Oscillator (DCO) operation is modulated by a computing unit (self-timed counter), such that the system will operate correctly in a wide range of voltage variation and arbitrary delay in the system. This chapter is based on the results presented in [17, 18]. I used the AMS 350 nm CMOS standard cells in the circuit design.

The DCO consists of a tunable delay line. The delay line consists of 4 3-input NAND gates in series connected to one input of a 2-input multiplexer with an inverted output. The output of the mux is fed back into the series NAND gates to form an oscillator. The oscillator triggers a 4-bit programmable modulo counter in the range 1 to 15. The programmable count modulo is used to tune the delay length in a multiplicative way, thus providing coarse frequency control for the DCO. I chose the method of delay path control used in this chapter to minimise area consumption.

Other methods of delay length control are delay path reconfiguration [84], current starving[85] or by output capacitance control[86]. Current starving method of control offers high resolution. However, this approach requires weighted transistor sizing to implement a linear controllable delay path [85]. Delay path reconfiguration can be combined with output capacitance control to design a DCO with both coarse and fine-tune control presented in

[84, 87–89]. Coarse control is achieved by selectively tapping from different sections of the long delay line, while fine tuning is achieved by controlling the output capacitances along the delay path. In each of these methods, the architecture of the delay path must be such that digital codes can be used to alter the controlled parameter.

In this thesis, the computing unit is a self-timed counter which is based on the $T-$Latch presented and decomposed into logic gates in [73, 74]. With a slight modification, a $N-$bit counter can be formed by cascading N $T-$latches.

In [90, 91], the $T-$latch based counter is used to code and characterise harvested energy. The information from the characterised energy is used to modulate computation based on available energy.

In this thesis, the operation of the DCO operation is modulated by the operating time and condition of the computing unit, in this case, the $T-$latch based counter. The reconfigurable delay path described earlier is complemented by the computation time of the computing unit. This structure ensures that a change in operating conditions (due to PVT) of the computing unit equally adjusts the DCO operation.

4.2 Overview of the Adaptive DCO

Figure 4.2 is the block diagram of the adaptive DCO system. It consists of a digitally controllable delay and a self-timed counter interacting in a closed loop system to form an oscillator.

Let the initial state of the system be such that the output of the delay and *ack* are ’0’s. This condition results in a ’1’ at the output of the DCO (clock out). Through the feedback loop, this ‘1’ appears at the input of the counter *Req* as a count request *req*+, and the input of the delay line which will output a ‘1’ after time D_{Total}. The request input will cause the counter to update its state by counting up one step after which *ack* is raised high. The combination of two ‘1’ inputs to the C-element results in the output of the DCO going to a ‘0’

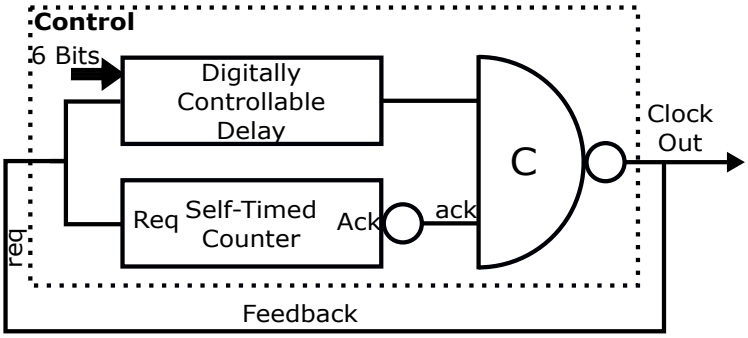


Fig. 4.2 Block Diagram of Proposed adaptive DCO System

($req-$). Consequently, the '0' is propagated through the delay line and *ack* is also returned low. This mode of operation causes the system to oscillate as this process will repeat.

Assume that the minimum delay of the controllable delay is less than the worst-case delay of the counter which is the time T_{ack} it takes for *Ack* output to change in response to a change in input *Req*. The counter operates such that T_{ack} varies, depending on the counter state for a $N-$bit counter.

Equation 4.1 defines the relationship between the DCO frequency and the delay path when $D_{Total} > T_{ack}$.

$$frequency_{DCO} = 1/2 * D_{Total} \tag{4.1}$$

Equation 4.2 defines the relationship between the DCO frequency and the delay path when $D_{Total} < T_{ack}$.

$$frequency_{DCO} = 1/2 * T_{ack} \tag{4.2}$$

From equations 4.1 and 4.2, it can be seen that the clock period is set by the slowest of either the delay line or the computing unit (in this case the counter).

The digitally controllable delay has two control modes, $4-$bit coarse control, and $2-$bit fine-tune control. The coarse control can be any number of bits greater than zero. However, for simplicity of design and simulation, it is limited to $4-$bits.

4.3 Circuit Implementation

4.3.1 Computing Unit (Self-Timed Counter)

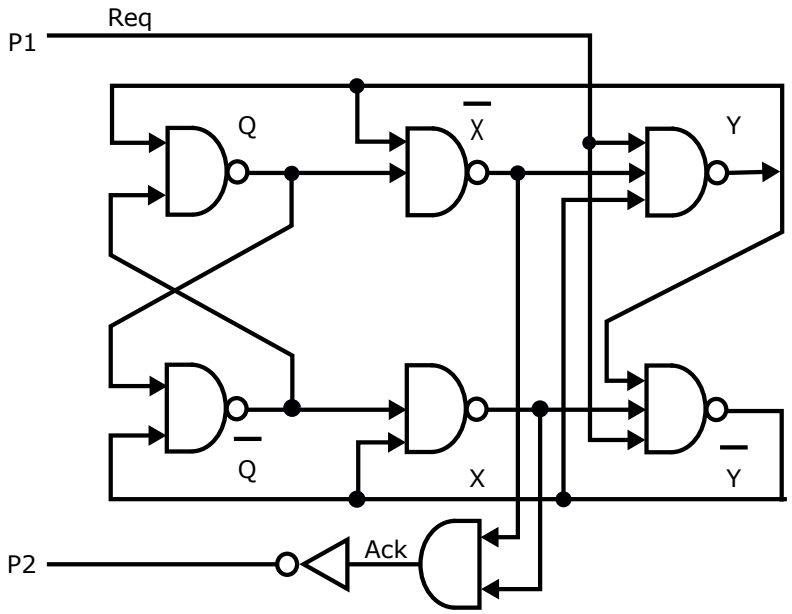


Fig. 4.3 Schematic of a 1-bit Self-timed counter

Fig. 4.3 shows a 1-bit self-timed up counter that will self-trigger when points $P1$ and $P2$ are closed. The STG of figure 4.4 show the sequence of operation of the closed circuit.

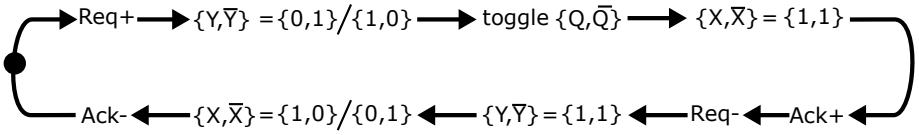


Fig. 4.4 STG of 1-bit T-Latch counter

Let the initial state of the T-latch be $Q,\overline{Q} = 0,1$; $X,\overline{X} = 0,1$; $Y,\overline{Y} = 1$; $Ack = 0$ and $Req* = 0$. The inverter is excited at this state and would fire after time T_1 to logically reflect its input. This results in $Req+$ which is a toggle request to the latch. The transition $Req+$ consumes the token and causes the transition $Y,\overline{Y} = 0,1$. This toggles $Q,\overline{Q} = 1,0$, which is the count output of the counter. The output $\overline{Q} = 0$ results in the transition $X = 1$, which results in the transition $Ack+$, an acknowledgement of the toggle request. The request is withdrawn

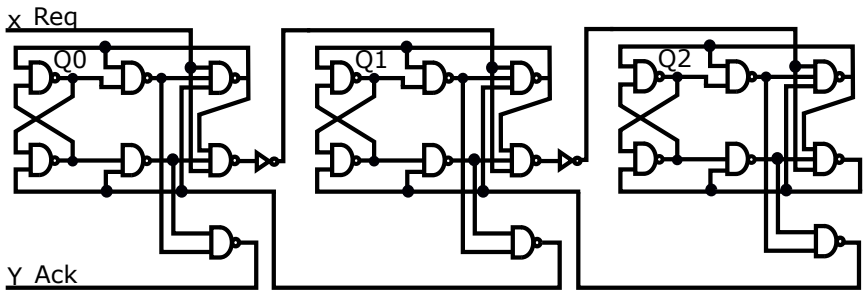


Fig. 4.5 Schematic of a 3-bits Self-timed counter

Req− after *Ack*+ transition. Withdrawing the request causes the circuit to withdraw the acknowledgement transition by first raising Y ($Y+$), which in turn cause the transition $\overline{X}-$ and consequently *Ack*−. At this point, the circuit is ready for another count request which is sent when the inverter transitions to a '1' which consequently, result in toggling Q.

In Figure 4.5, three T-latches were cascaded to form a 3-bit self-timed counter. The STG of Figure 4.6 shows the sequence of operation of a 3−bits self-timed counter with outputs Q_0, Q_1 and Q_2 in LSB to MSB order when points X and Y are closed. The outputs toggle in ascending linear binary sequence.

The count request *Req*+ is sent on the LSB. A request causes the LSB to toggle its Q output and return an acknowledgement on *Ack* only if the LSB is a '0'. If the LSB is a '1', or any bit is a '1', a request received is propagated to the next bit until a latch with Q output = '0' is flipped to a '1'. After which those bits of a lower order that were skipped will each be flipped to a '0', starting from the more significant bit.

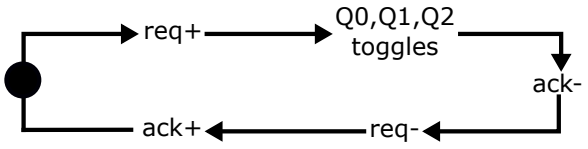


Fig. 4.6 General State Transition Graph of a 3-bits Self-timed counter

The worst case delay of the self-timed counter occurs when the most significant bit (MSB) of a N-bit counter changes state. In the 3-bits counter, this occurs when transitioning from count 3 (011) to count 4 (100). In state 3, when the request is raised high, the MSB is toggled

first before other subsequent lower significant bits are toggled to the correct state. This worst-case delay is expressed in terms of gate transitions T_i. Where T_1, T_2 and T_3 represent the inverter, 2-input NAND and 3-input NAND gate transition delay respectively. The term T_i denotes a bounded delay in that when a gate is excited, it settles in a state that logically reflects its input after a time given by the bounds $\delta T_i \leq T_i \leq \Delta T_i$. The term δT_i denotes a lower limit which is the earliest time the gate will fire, while ΔT_i denotes an upper limit. This aids in characterising the worst case delay of the system based on the number of transitions per logic gate, which is shown in equation 4.3.

$$T_{worstcase} = 6NT_2 + 2NT_3 + 2(N-1)T_1 \tag{4.3}$$

where N is the total number of counter bits. In this thesis, $N = 5$.

This delay expression differs from the equations presented in [91], where the aim was to show the relationship between the output count and number of gate transitions in the counter as opposed to 4.3 which express the worst-case delay in a $N-$bit counter by the highest number of gate transitions before *Ack* outputs a response to an input in *Req*.

In the circuits of Figures 4.3 and 4.5, irrespective of the delay in each gate, the circuit will still operate without error because it is Speed independent/quasi-delay insensitive circuit.

This mode of operation of the counter as stated above highlights a varying response time on the *Ack* signal to a toggle request.

4.3.1.1 Measurements and Post-Layout Simulation of a 5-bits Counter

A 5-bits self-timed counter is implemented and simulated in AMS 350 nm CMOS library for illustrative purpose in this subsection. However, a 9-bits self-timed counter is used in the elastic DCO system.

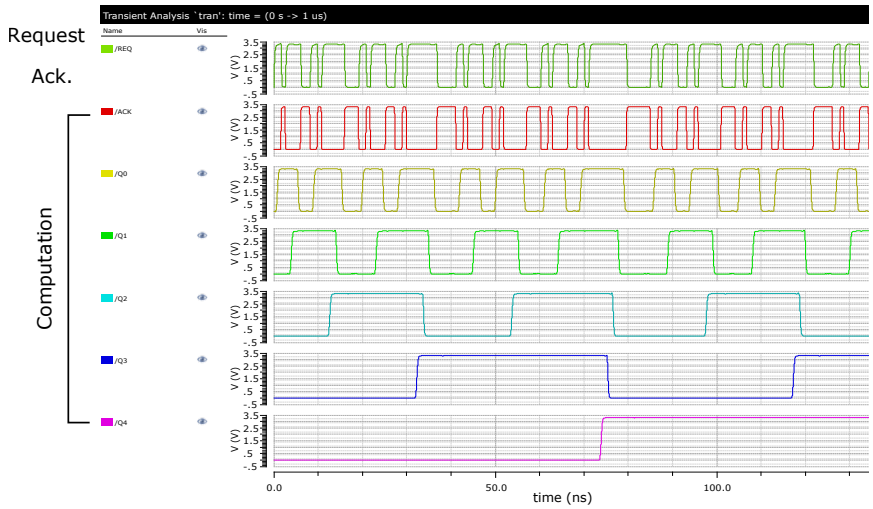


Fig. 4.7 Simulated Operation of the Self-Time Counter

Figure 4.7 is a Post layout simulation plot of a 5-bits self-timed counter. Figures 4.8 and 4.9 are plots showing that the task completion times of the counter varies, this is simulated at 1.8 V and 3.3 V respectively. The task completion time depends on the counter state.

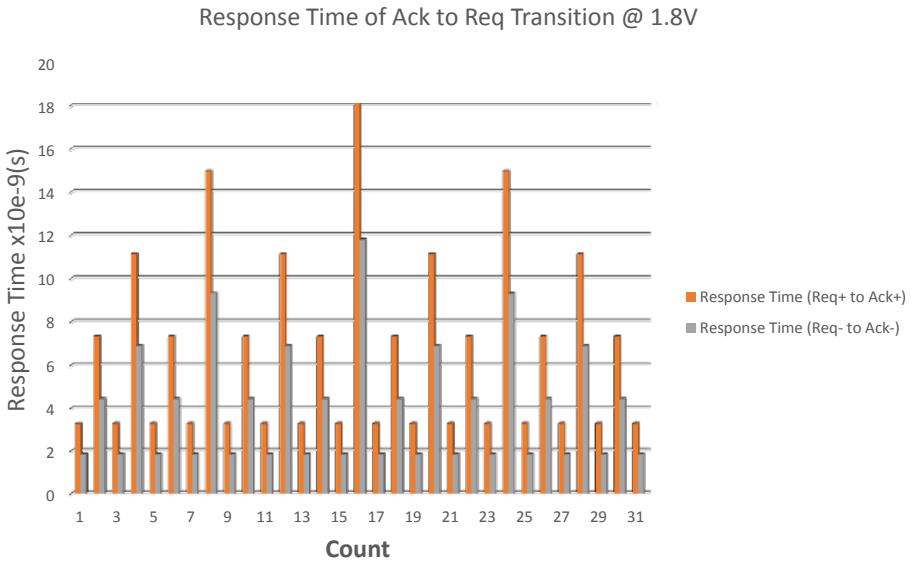


Fig. 4.8 Response Time Plots at V_dd = 1.8V

Figures 4.8 and 4.9 highlight that the response time for odd numbers in a given voltage domain is always constant. Typical values for the response time on *ack* to a count request at 1.8V, 2.8V and 3.3V are 3.31e-9(s), 2.13e-9(s) and 1.49e-9(s) respectively. In each of the

plots, the response time on *ack* after withdrawing the request is shown, with typical values at 1.89e-9(s), 1.2e-9(s) and 0.841e-9(s) respectively.

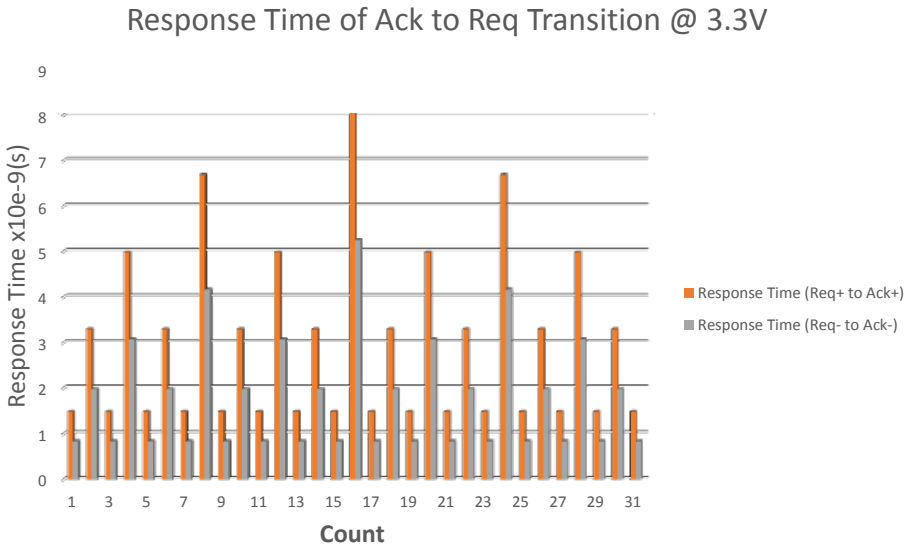


Fig. 4.9 Response Time Plots at V_dd = 3.3V

4.3.2 Delay system

For even numbers, the response time on *ack* is not constant but varies, with the longest occurring at count sixteen. Simulated values for the response time for count sixteen at 1.8V, 2.8V and 3.3V are 18.82e-9(s), 12.0e-9(s) and 8.4e-9(s) respectively for count request and 11.83e-9(s), 7.51e-9(s) and 5.26e-9(s) respectively for withdrawal of count request.

This varying response time characteristics of the counter makes it suitable for the application in this investigation, as the counter response time can be viewed as modelling a computing unit with different task completion times.

Figure 4.10 is the block diagram of the proposed digitally controllable delay system. It consists of a unit delay, event tracker, asynchronous controller, output flip-flop and a fine tune unit.

The delay system is activated when the asynchronous controller detects a transition event in the DCO output. When a transition is detected, the controller raises the request signal

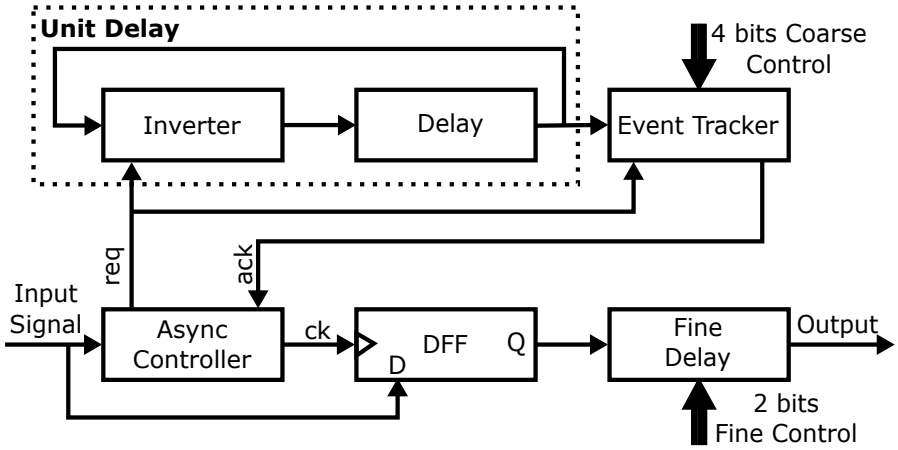


Fig. 4.10 Block Diagram of the Proposed Digital Controllable Delay

high, enabling the unit delay block and the event tracker. Due to the feedback loop, the unit delay will oscillate when enabled.

The ack signal from the event tracker is raised high when the number of oscillations equals the binary value of the coarse tuning code. This causes the controller to simultaneously lower the request signal and raise *ck* output, clocking the DCO output into the D-Flip flop (DFF). A 4-input multiplexer (mux) with three of its inputs connected to different sizes of smaller delay serves as the output fine-tune delay system. The mux select input serves as the fine tuning control inputs.

The value of coarse control bits affects the unit delay in a multiplicative way via the effect of the loop.

The delay system is expressed as shown in equation 4.4.

$$D_{Total} = D_{Unit} \times C_{coarse} + D_{fine} \times C_{fine} \tag{4.4}$$

C_{coarse} and C_{fine} are the control code and can be defined as $\{C_{coarse/fine} | C_{coarse/fine} \in \mathbb{N}, C_{coarse/fine} < 2^N\}$

Where $\mathbb{N}$ is a set of natural numbers and N is the number of bits for either coarse or fine-tune controls. In this system, $D_{fine} < D_{Unit}$

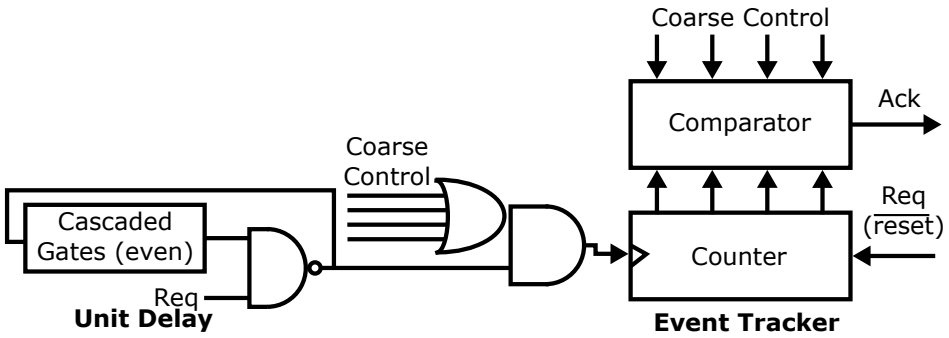


Fig. 4.11 Unit Delay and Event Tracker Circuits that forms the Coarse Delay Path

Equation 4.4 does not take into account gate delays for the event tracker, asynchronous controller, DFF, mux and the C-element with its input and output inverters. At a given supply voltage, these values can be assumed to be constant since the control codes have no effect on them. I summed up these constants to get $D_{constant}$. Therefore, equation 4.4 is modified to:

$$D_{Total} = D_{Unit} \times C_{coarse} + D_{fine} \times C_{fine} + D_{constant} \tag{4.5}$$

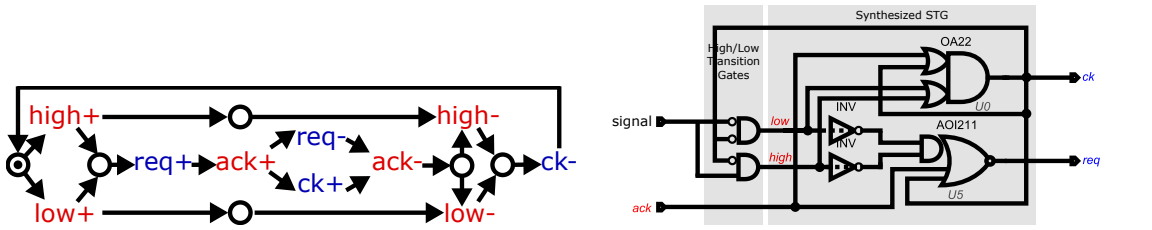
4.3.2.1 Coarse Delay Path

Figure 4.11 shows the circuit implementation of the Coarse Delay Path. It consists of a unit delay and an event tracker. The unit delay is an oscillator, and the event tracker counts the number of oscillation from the unit delay. The event tracker produces an end of count signal when the number of oscillations equals the input coarse control word.

I designed the unit delay using 4 3-input NAND gates and a 2-input multiplexer.

A ’0’ input from the *req* output of the Async controller disables the unit delay.

The event tracker is made of binary counters and a programmable comparator that raises the ack signal high when the counter value matches the input control code. The input clock from the unit delay to the binary counter is disabled when input code on the coarse control is "0000".



(a) STG of the Asynchronous Controller (b) Circuit implementation of Asynchronous Controller

Fig. 4.12 STG and Circuit of the Asynchronous Controller

4.3.2.2 Asynchronous Controller

The Asynchronous Controller modulates the operation of the coarse delay path by detecting when a transition occurs in the DCO output (either low to high or high to low). A change in state of the DCO output results in the controller enabling the ring oscillator of the coarse delay by raising *req* which is a request output from the controller. The enabled ring oscillator clocks the programmed modulo counter. The controller will disable the ring oscillator at the end of the coarse delay path operation- when the programmed modulo count is reached which results in a low to high input transition on *ack* of the controller. The delay path will be disabled until the DCO output changes.

The STG of Figure 4.12a specifies the asynchronous controller (Async Controller) operation. Only one of the choice transitions $low+$ or $high+$ can occur at a time since they share a single preset condition. This ensures that the delay path will only be enabled after a complete transition and when the output of the DCO transitions state. This prevents glitches from occurring in the output of the DCO during the controller operation. When the DCO output changes state, *req* is raised and can only be lowered when *ack* is raised from the coarse delay path. The signal *ck* clocks DFF by transitioning high and is lowered after the event tracker is reset and DFF has been updated to the DCO output. The signals *high* and *low* model the DCO output transition high and low respectively.

Figure 4.12b shows the synthesized circuit of the modelled controller. The circuit was synthesized using the WorkCraft tool [1]. Extra logic gates were added to ensure that once

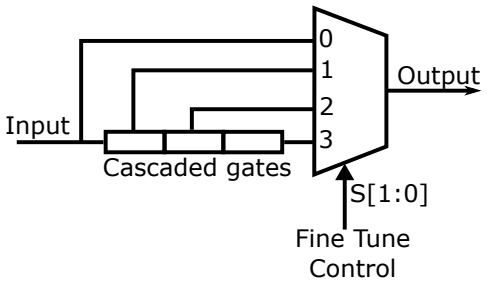


Fig. 4.13 Fine Tuning Circuit

the output DFF has been clocked by *ck*, high and low transitions are forced to a low. This ensures that the controller is ready to detect another signal change from high to low or vice versa by returning the token to initial place.

4.3.2.3 Fine Delay

Figure 4.13 shows the fine-tune delay block diagram. This unit consists of a 4-input Mux, with three of the inputs fed from each stage of a series delay line divided into three equal stages of even number AND gates (I used two gates for each stage). The output of each delay stage is connected to an input in the multiplexer. A 2-bits control input to the mux select is used to select from any one of the mux inputs to tune the delay. Since the zero input of the mux is connected directly to the incoming signal from DFF, the signal will be time-shifted by the gate delay of only the mux. Otherwise, the delay C_{fine} from input to output is a function of the mux gates delay and the delay path determined by the fine-tune code.

4.4 Simulation Measurement

Figure 4.14 shows the simulation result of the DCO system at 3.3V. The signal labelled "Delay" is the output of the delay path, while the signal labelled "Computing Unit" is the acknowledgement output of the computing unit.

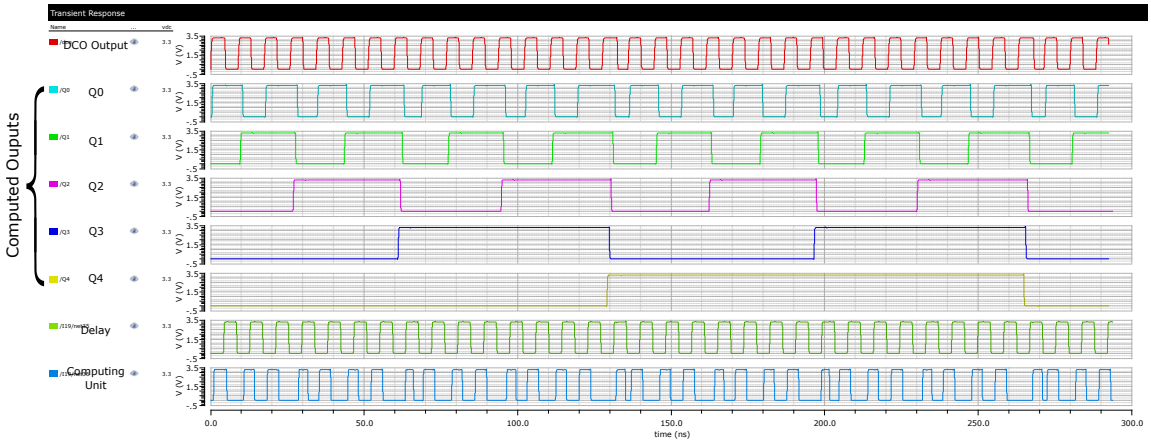


Fig. 4.14 Simulation Result of the DCO and integrated computing unit system at 3.3V

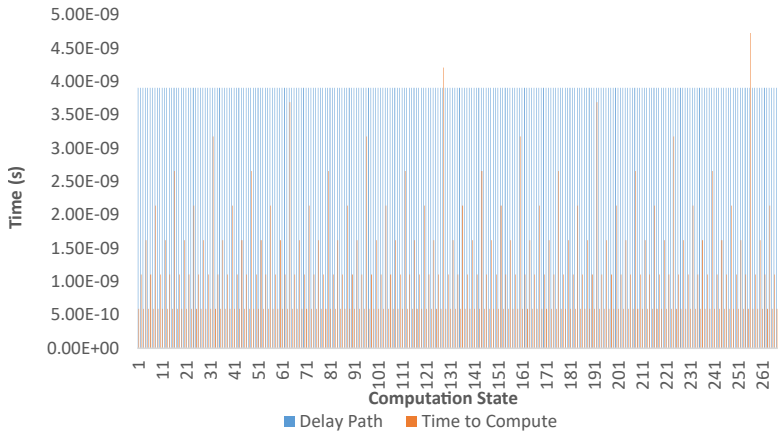


Fig. 4.15 Response Time of Delay Path and the Computing Unit @ 3.3V

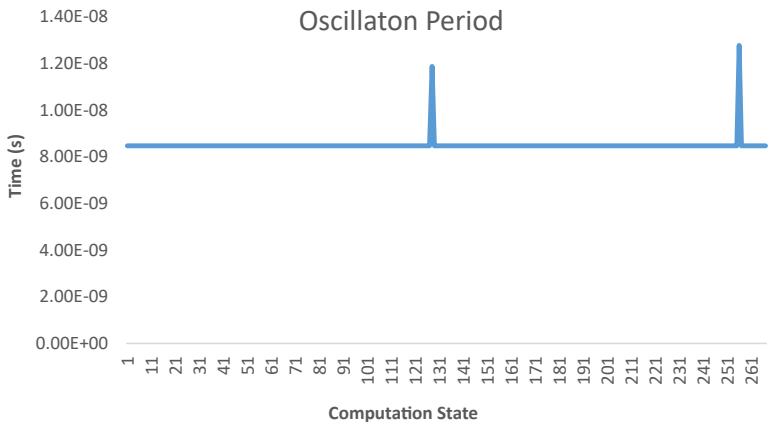


Fig. 4.16 Response of DCO output period to change in computing time @ 3.3V

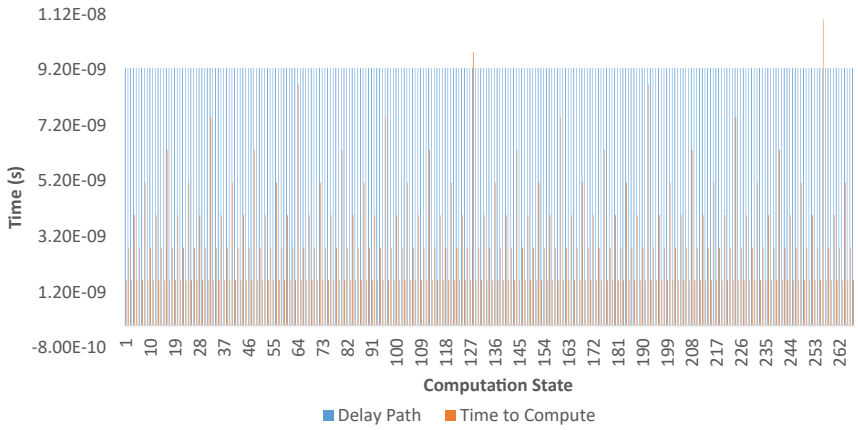


Fig. 4.17 Response Time of Delay Path and the Computing Unit @ 1.8 V

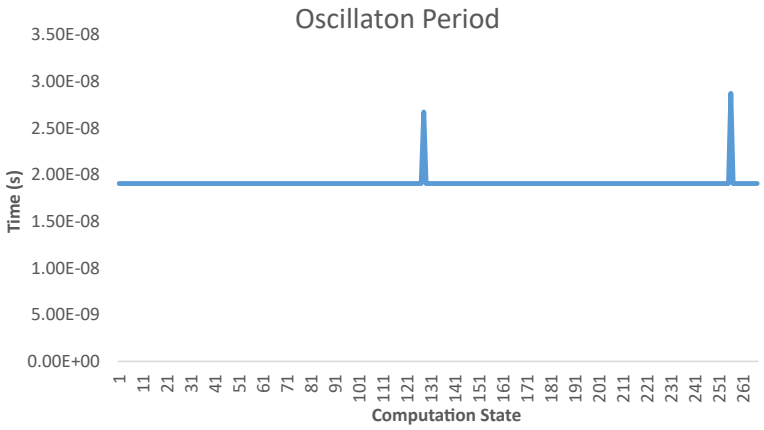


Fig. 4.18 Response of DCO output period to change in computing time @ 1.8 V

In most instances at a given voltage, the computing unit responded first to a transition on the request input before the delay path. However, due to the synchronising operation of the C element, the next request transition will occur after both delay path and computing unit respond to the previous request transition.

The delay path response time to an input is constant (3.9ns, 9.27ns @ 3.3V and 1.8V respectively) irrespective of the computation state of the computing unit.

Figures 4.15 and 4.17 show simulation results of the computing unit and the delay path. At certain times, the computing unit took more time to complete a task. The DCO output is shown to automatically adjust when this occurs in the simulation outputs of 4.16 and 4.18 at 3.3 V and 1.8 V respectively.

By tuning the coarse control, the DCO output frequency ranged from 8MHz to 121.5MHz at 3.3V and 3MHz to 52.48MHz at 1.8V.

Average power consumed by the system is 1.99μW at 3.3V and 236.2μW at 1.8V.

4.5 Conclusion

A method of integrating the operations of an oscillator and a self-timed computing unit to exploit the robustness property of asynchronous circuits was presented in this chapter. The oscillator and the computing unit interact in a 4-phase handshake relationship. In extreme conditions, the oscillator operation is modulated by the computing time of the self-timed computing unit.

Chapter 5

Loadable Kessels Counter

5.1 Introduction

This chapter presents the decomposition, specification and implementation of a loadable self-timed modulo$-n$ counter that is applied in the design of a fine-tunable digital pulse-width modulator in Chapter 6.

The counter is decomposed into a linear array of independent and interacting cells, each with defined operations. This decomposition approach gives the counter bounded response time on its outputs. The method by which the modulo$-n$ counter is decomposed into an array of cells was first presented by Kessels [14], hence the title of this chapter. However, in this chapter the decomposition method is further investigated to create cells with a range of operations that can be dynamically activated depending on the value of the count modulo n. This approach eliminated the need for a comparison circuit to detect the end of the count.

Formal asynchronous design techniques is employed in the specification of the decomposed counter cells. The specified counter was synthesised using the WorkCraft Tool and implemented using standard cells in AMS 350 nm CMOS technology.

This chapter is based on the idea presented in [81] and the work presented in [92].

The realisation of the loadable modulo$-n$ is presented in the following steps:

- General specification of a loadable self-timed modulo$-n$ counter.
- Decomposition of the modulo$-n$ counter into array of cells (each cell further decomposed into left and right cell parts) by rewriting n using Horner's method.
- High level specification of each decomposed cell part operations using Labelled Petri Nets (LPN).
- Modelling and verification of the decomposed counter.
- Definition of the different conditions and range of operations for left and right parts of a cell.
- Composition of each counter cell part to contain all its possible operations and its specification to allow activation of any of its operations (one at a time) from a corresponding control cell part using LPN and Signal Transition Graphs (STG).
- Definition of the different input conditions of each control cell part and range of activation commands to its corresponding counter cell part.
- Specification of each control cell part using LPN and STG.
- The response-time characterisation of the complete counter outputs.

A top-down approach was used in the counter cells specification which include: high level specification using Labeled Petri Nets (LPN) [15] and low level specification using Signal Transition Graphs (STG) [16] with all complete state coding conflicts (CSC) [93, 39] resolved leading to a synthesizable specification.

The loadable counter was implemented in 350 nm CMOS technology. This involved technology mapping of synthesised gates to AMS 350 standard library.

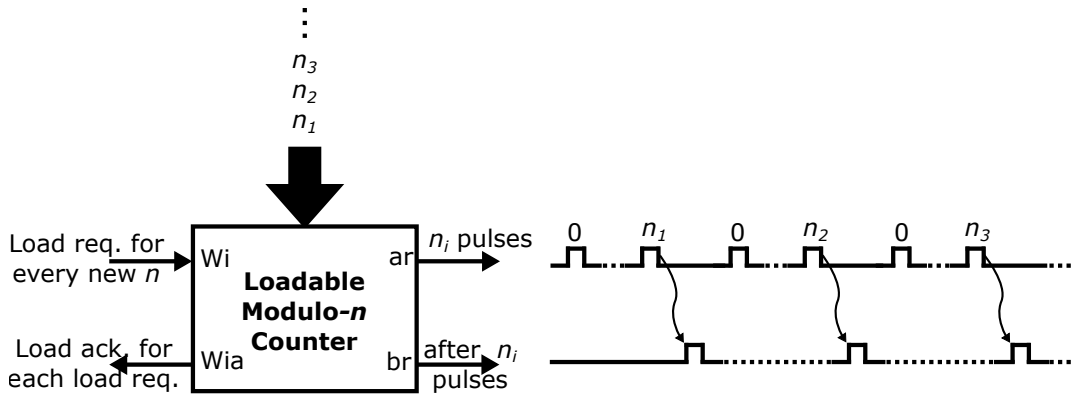


Fig. 5.1 Block Diagram of the Loadable Counter

5.2 Loadable Modulo−n Counter Overview

Figure 5.1 illustrates the behaviour of the proposed loadable self-timed modulo−n counter (CT_n). It describes a counter that can reconfigure its operation from an input quantity n_i in space to produce a time quantity on its output ports ar and br. The term n_i denotes the ith modulo loaded into the counter. Typically, the range of n the counter can accept is defined by the number of input bits the counter can accept (n is input in binary).

A load request to the counter from the environment on input port Wi loads the counter with the present count modulo n_i at its input, after which an acknowledgement to the load request is sent back to the environment on output port Wia. Loading the counter results in n_i pulses on the ar output after which an end of count pulse is produced on the br output port. The counter operation is specified using a Labelled Petri Net as shown in Figure 5.2.

The relationship between the count modulo n and the operation on ports ar and br of CT_n is described by the regular expression 5.1, where $n \geq 1$. The terms ?, ! and $*$ denote an input port, output port and arbitrary repetition of the operation respectively.

$$CT_{n?} = (ar!^n \; br!)^* \tag{5.1}$$

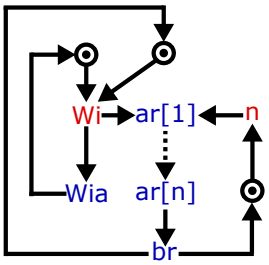


Fig. 5.2 High Level Specification of the Loadable Modulo$-n$ Counter

Two channels of operations are present in CT_n. These channels are *ar* and *br* channels which do the n count operation and end of count operation respectively. Expression 5.1 does not sufficiently provide details as to the order of internal events on the *ar* and *br* channels.

In order to specify and realise a loadable counter, n is initially considered to be static. From this, Kessels method of decomposition is applied [14].

In the next two sections, details of *ar* channel operation are defined by way of decomposition of the count modulo n.

5.3 Decomposition of $-n$ by recursive rewriting

The count modulo n is rewritten using basic mathematical expression such that $n = n$ MOD 2 + 2(n DIV 2). Consequently the count operation on *ar* channel becomes $ar^n = ar^{n\ MOD\ 2}\ 2(n\ DIV\ 2) = ar^{n\ MOD\ 2}\ ar^{2(n\ DIV\ 2)}$.

Let the expression n MOD 2 = d_0 and n DIV 2 = n_0, therefore, $n = d_0 + 2 * n_0$. This is the first level of rewriting n. In the first level, n is divided into two interacting counters d_0 and n_0. The d_0 term is either a 0 or 1 counter, however the n_0 term is a sub-counter that can be further decomposed using the same method to get d_1 and n_1 terms.

The following levels of decompositions are thus defined:

- $d_0 = n$ MOD 2, $n_0 = n$ DIV 2
- $d_1 = n_0$ MOD 2, $n_1 = n_0$ DIV 2

Each stage of rewriting can be expressed as:

- $d_i = n_{i-1}$ MOD 2, $n_i = n_{i-1}$ DIV 2

$i = \{1, ..., \lfloor log_2(n) \rfloor\}$. With each decomposition, the term n_i tends towards 0 and becomes 0 when $i = \lfloor log_2(n) \rfloor$.

5.4 Decomposition of $-n$ using Horner's Method (Kessels approach)

If the aim of rewriting n is to decompose n partially, the approach in section 5.3 can be applied until the level of decomposition is achieved. However, if the above procedure is followed to the last decomposition, it amounts to an expression equal to Horner's method, shown in (5.2). Horner's method gives a complete decomposition of n and was applied by Kessels in his decomposition.

$$n = ((..((0 \times 2 + d_{N-1})2 + d_{N-2})2 + ..)2 + d_1)2 + d_0) \tag{5.2}$$

5.4.1 Identification of Cells and Cell Parts in decomposed counter

In (5.2), each closing bracket contains an expression in the form $(\times 2 + d_i)$ which is referred to as a counter cell, denoted by c_i.

Each cell contains two parts, a left part *CL* which does the $(\times 2)$ operation and a right part *CR* which does the $(+d_i)$ operation.

The d_i term is either a '0' or a '1'. Therefore, Horner's method can be said to expresses n as an unsigned binary number in the range d_{N-1} to d_0. Where d_{N-1} and d_0 are the Most Significant Bit (MSB) and Least Significant Bit (LSB) respectively.

The number of bits N representing n is equal to the number of decomposed counter cells, given by the expression $\lfloor log_2 n \rfloor + 1$.

Table 5.1 Table shows decomposition of N in the range of 0 to 15 into Left and Right operation.

N		c_3		c_2		c_1		c_0	
	Binary	CL_3	CR_3	CL_2	CR_2	CL_1	CR_1	CL_0	CR_0
0	0000	0	0	0	0	0	0	0	0
1	0001	0	0	0	0	0	0	0	+1
2	0010	0	0	0	0	0	+1	×2	+0
3	0011	0	0	0	0	0	+1	×2	+1
4	0100	0	0	0	+1	×2	+0	×2	+0
5	0101	0	0	0	+1	×2	+0	×2	+1
6	0110	0	0	0	+1	×2	+1	×2	+0
7	0111	0	0	0	+1	×2	+1	×2	+1
8	1000	0	+1	×2	+0	×2	+0	×2	+0
9	1001	0	+1	×2	+0	×2	+0	×2	+1
10	1010	0	+1	×2	+0	×2	+1	×2	+0
11	1011	0	+1	×2	+0	×2	+1	×2	+1
12	1100	0	+1	×2	+1	×2	+0	×2	+0
13	1101	0	+1	×2	+1	×2	+0	×2	+1
14	1110	0	+1	×2	+1	×2	+1	×2	+0
15	1111	0	+1	×2	+1	×2	+1	×2	+1

The notations ith counter cell (c_i) and its parts counter cell left part (CL) and counter cell right part (CR) denote the ith counter cell, its left and right parts respectively. The notation d_i denotes the ith bit representing n.

5.5 Operations of Counter Cell Parts

A loadable counter implemented in silicon has a fixed number of counter cells. Let the total cells in an implemented counter be denoted by N_T, to differentiate the number of bits representing the count modulo n in the relationship $n \in [0..2^{N_T} - 1]$, which represents the range of values of n that can be loaded to the counter.

Tables 5.1 and 5.2 illustrate a counter with $N_T = 4$ number of cells, with the range of numbers for n decomposed into cells with left and right parts. The set of counter cells directly mapped to a valid bit of n are the active cells involved in the computation of n. The left and

Table 5.2 Table shows operation of dormant cell parts as spacers (-) in both left and right parts.

N		c_3		c_2		c_1		c_0	
	Binary	CL_3	CR_3	CL_2	CR_2	CL_1	CR_1	CL_0	CR_0
0	0000	-	-	-	-	-	-	-	-
1	0001	-	-	-	-	-	-	0	+1
2	0010	-	-	-	-	0	+1	$\times 2$	P
3	0011	-	-	-	-	0	+1	$\times 2$	+1
4	0100	-	-	0	+1	$\times 2$	P	$\times 2$	P
5	0101	-	-	0	+1	$\times 2$	P	$\times 2$	+1
6	0110	-	-	0	+1	$\times 2$	+1	$\times 2$	P
7	0111	-	-	0	+1	$\times 2$	+1	$\times 2$	+1
8	1000	0	+1	$\times 2$	P	$\times 2$	P	$\times 2$	P
9	1001	0	+1	$\times 2$	P	$\times 2$	P	$\times 2$	+1
10	1010	0	+1	$\times 2$	P	$\times 2$	+1	$\times 2$	P
11	1011	0	+1	$\times 2$	P	$\times 2$	+1	$\times 2$	+1
12	1100	0	+1	$\times 2$	+1	$\times 2$	P	$\times 2$	P
13	1101	0	+1	$\times 2$	+1	$\times 2$	P	$\times 2$	+1
14	1110	0	+1	$\times 2$	+1	$\times 2$	+1	$\times 2$	P
15	1111	0	+1	$\times 2$	+1	$\times 2$	+1	$\times 2$	+1

right parts of an inactive cell operate spacers "-" in Table 5.2 since they do not add to the count sequence.

The operation of *CL* for the most significant active cell is always a zero operation (0×2). For all other active cells, the operation of *CL* is always ($\times 2$). To realise a loadable counter, the operations of *CL* in each cell must include a spacer (-) operation, a $\times 2$ and zero operations. The choice of operation depends on the binary sequence of n. The only exception to this is CL_{N_T-1} which is the left part of the Most Significant Cell (MSC) c_{N_T-1} in the implemented counter. From Table 5.2, this cell part operates as either a spacer (-) when its corresponding bit is a '0' or a Zero operation (0×2) when its corresponding bit is a '1'. The range of possible operations of *CL* is referred to as **EVEN Operations**.

The operation of *CR* for an active cell c_i adds the corresponding bit value d_i once to the count sequence. When $d_i = 0$, nothing is added to the count sequence. In this case, CR_i operates as a channel for passing counts received from CL_i to CL_{i-1}. This operation is

referred to as a Pass (P) operation as shown in Table 5.2. To realise a loadable counter, the operations of CR in each cell must include a spacer (-), a +1 and a P. The only exception to this is CR_{N_T-1} of the Most Significant Cell (MSC) in the counter which can operate as either a spacer (-) when its corresponding bit is a "0" or a +1 when its corresponding bit is a '1'. The range of possible operations of CR is referred to as **ODD Operations**.

5.5.1 Concurrency in Decomposed CT_n

The decomposition of n in (5.2) shows independent interactions between CR_i and CL_{i-1}. This interaction occurs concurrently across all cells. The result of each interaction is communicated to an adjacent cell part CR_{i-1}.

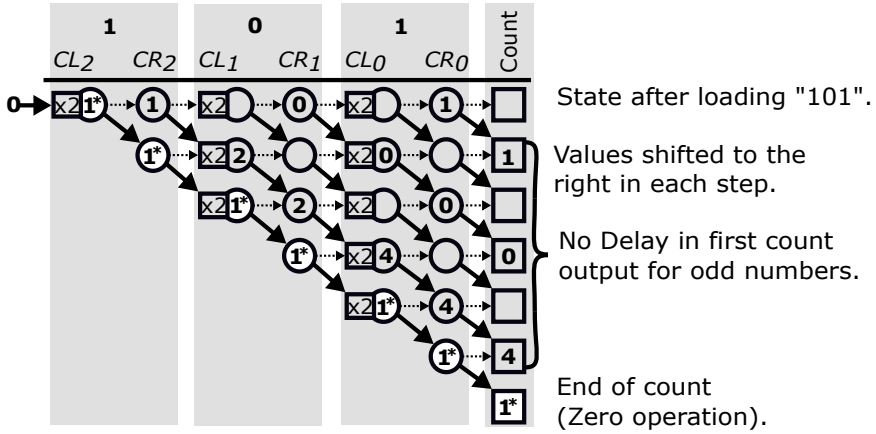


Fig. 5.3 Diagram illustrates parallel operation in counter cells for count 5 with no delay before first count output

Consider the decomposition example for n = 5, decomposed as $(((0\times 2+1)2+0)2+1)$. Concurrent operation between cells and cell parts in each counter decomposition is illustrated in Figures 5.3.

In Figures 5.3, dashed arrows between cell parts indicate the direction of data flow. The new state of each counter cell part after receiving input is shown in the row below, with solid arrows used to reiterate the origin of data.

At the initial state, cell c_i right part CR_i independently loads and computes the value of its corresponding bit. The result of the computation is communicated to CL_{i-1}. This action occurs in parallel across the counter. The part CL_i doubles every data input received and communicates the result to CR_i. This operation also occurs in parallel. The part CR_i holds and passes data received to part CL_{i-1}. As numbers are shifted to the right, counts begin to appear on the count output column, which is the *ar* output of the counter.

The counting operations so far described have only considered operations on the *ar* channel of the counter. Since an event on *br* can only occur after *n* event(s) on *ar*, production of the *br* signal starts from the left part of c_{N-1} to account for its zero operation. This zero operation is represented by 1^* in Figures 5.3. It occurs only on the *br* channel, and it is not doubled when communicated within or without cell parts. It can only be passed to a cell part that has completed its computation for *n* on its *ar* channel. Hence, it appears on the *br* output of the counter after *n* counts on its *ar* output.

The numbers shown in the count column are the counts output resulting from the interaction of the counter cells. The blank spaces between output counts do not model the delay in the system.

5.6 High Level Specification of Counter Cell Parts Operations

In this section, the operations of each counter cell part are specified. The specification for each cell part includes the two channels *ar* and *br* previously defined. The decomposed computation for *n* occurs on the *ar* channel, while the zero operation occurs on the *br* channel.

Tables 5.3 and 5.4 lists the input and output signal names of each counter cell part. These names are used in the specification of the cell part operation. The signal names in the bracket

Table 5.3 Input Signal Names of Counter Cell Parts in each Channel

Input Signal Names				
Cell Part	CL_{N_T-1}	CR_{N_T-1}	CL_i	CR_i
Config. Input	n10 (n1a)	n21 (n2a)	n10, n11 (n1a)	n20, n21 (n2a)
ar Channel		cr (ca)	cr' (ca')	cr (ca)
br Channel		dr (da)	dr' (da')	dr (da)
Note: The names in bracket are the acknowledgement signal names to these inputs.				
These are used in the STG specification for the concerned cell part.				

Table 5.4 Output Signal Names of Counter Cell Parts in each Channel

Output Signal Names				
Cell Part	CL_{N_T-1}	CR_{N_T-1}	CL_i	CR_i
ar Channel		ar (aa)	ar' (aa')	ar (aa)
br Channel	br' (ba')	br (ba)	br' (ba')	br (ba)
Note: The names in bracket are the acknowledgement signal names to these outputs.				
These are used in the STG specification for the concerned cell part.				

are the acknowledge signal names used in the STG specification later. For an input signal, its acknowledge signal is an output signal and vice-versa.

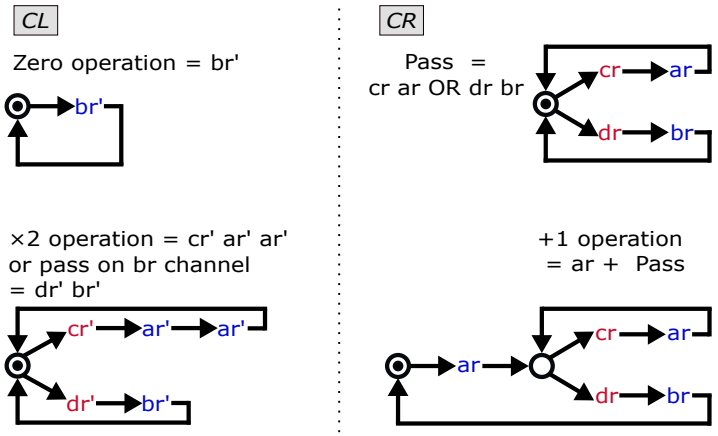


Fig. 5.4 LPN specifying even operations

On the left side of Figure 5.4, two LPNs for *CL* are shown. The first is a zero operation specified as an enabled br' output signal which can fire independently. The second is a $\times 2$ operation, with a choice for an input event on cr' and dr'. An input event on cr' results in

two output events on ar', this is the times two operation. An input event on dr' results in an output event on br', which is effectively passing a zero operation.

On the right side of Figure 5.4, two LPNs are shown for CR. The first LPN is an enabled choice of pass operations. An input event on cr or dr will result in an output event on ar or br respectively. The second LPN is a $+1$ operation in which an event on ar output is enabled and can fire independently. This action enables a choice of pass operation on cr and dr. An input event on cr results in an output event on ar and re-enables the choice of pass operation, while an input event on dr results in an output event on br after which the $+1$ operation is re-enabled.

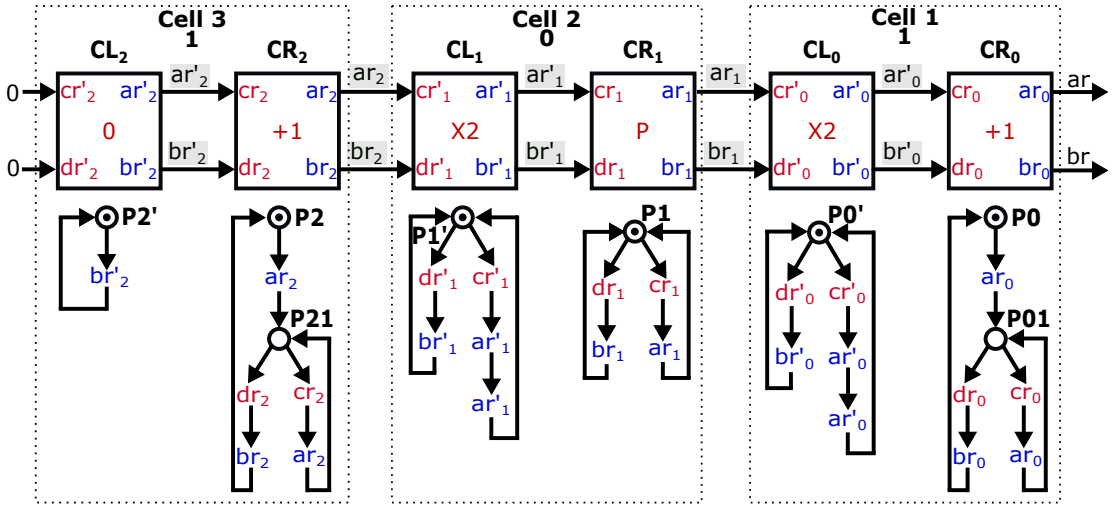


Fig. 5.5 Counter Configuration for count 5

5.6.1 Modelling of Decomposed Count Modulo 5 by Unfolding

Figure 5.5 shows the interaction of counter cell parts as a result of decomposition of count modulo 5. The interaction between signal names of the LPNs of cell parts is indicated by arrows to show origin and destination of an event.

Figure 5.6 shows the unfolding of events, based on the interaction between cells parts in Figure 5.5. In the nomenclature used for the unfolding, an interaction between signals of two

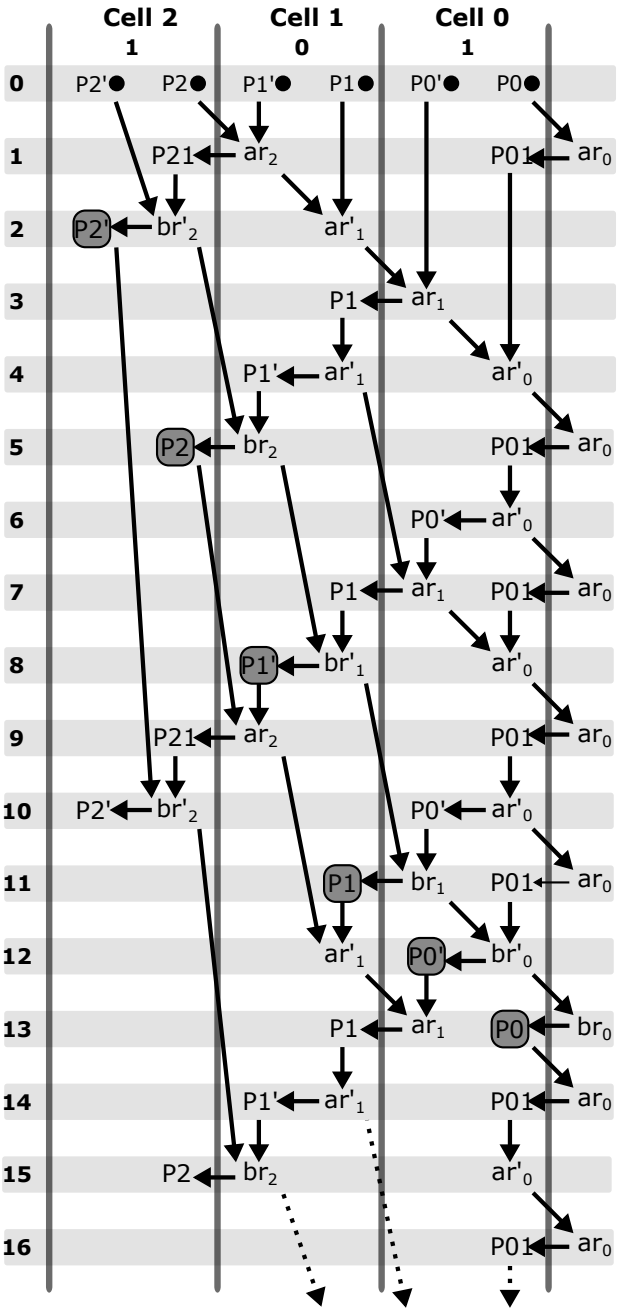


Fig. 5.6 Unfolding of signal transition for count 5

cell parts is given the output signal name. For example in Figure 5.5, the interaction between CR_2 and CL_1 on the *ar* channel is shown by the output to input connection ($ar_2 \rightarrow cr'_1$), is named "ar_2". The event ar_2 in Figure 5.6 means an output event on ar_2 of CR_2 consumed tokens from $P2$ and $P1'$ and then placed a token in $P21$ while enabling ar'_1 output.

An event between two interacting signals is shown to occur when the pre-conditions of both signals are satisfied. For example in Figure 5.5, the interaction ($br_2 \rightarrow dr'_1$) named br_2 in Figure 5.6 could not be shown to occur until a valid input transition on dr_2 enabled br_2.

In the unfolding of Figure 5.6, it is assumed that all enabled transitions will fire in minimal step bundle. This assumption does not affect the method used to verify the functional correctness of the decomposition and specification approach.

In Figure 5.6, transition events are grouped and shown in steps indicated by the numbers on the left-hand side. The unfolding is explained in the following steps:

Step 0: Each counter cell part is in the initial state as shown in its LPNs. Transitions br'_2, ar_2 and ar_0 are enabled.

Step 1: Transitions ar_2 and ar_0 fire. An event on ar_2 consumes a token from $P2$ and $P1'$ and places a token in $P21$ enabling input dr_2 and output transition ar'_1 respectively. An event on ar_0 placed a token in $P01$, this enabled a choice of inputs dr_0 or cr_0. This ar_0 event is also an output from the counter.

Step 2: Transition br'_2 is shown to have fired in this step because transition dr_2 pre-condition was satisfied in step 1, this enabled output transition br_2. Transition ar'_1 also fired in this step, and consequently enabled transition ar_1.

Step 3: Transition ar_1 is shown to have fired in this step, this action consumed a token from $P0'$ through input cr'_0 and consequently enabled transition ar'_0 and placed a token in $P1$.

Step 4: Transition ar'_1 and ar'_0 fired in this step. Transition ar'_1 is the second ar'_1 transition as a result of ar_2 transition in step 1. This places a token on $P1'$ which is the initial state of CL_1. Transition event on ar'_0 also enabled transition ar_0 through input transition cr_0 which was enabled in step 1.

The rest of the unfolding followed this pattern of token flow as a result of interaction between enabled output and input signals of two interacting cell parts. After five ar_0 transitions, a br_0 transition occurred in line 13.

After the first ar_0 transition event in step 1, the next ar_0 transition event occurred after cr_0 received an input event from ar'_0. From the graph of Figure 5.6, ar'_0 transition did not occur until step 4. This is because CR_1 operates as a Pass, requiring an input event on cr_1 to produce an output event on ar_1. A chain of causes and effects beginning from ar_2 transition in step 1 unfolded until step 3 before ar'_0 transition occurred in step 4, thus the next transition on ar_0 is shown in step 5.

The br'_2 transition in step 2 is a zero operation which is gradually passed through cell parts from the MSC to the LSC on the *br* channel. Its transition in two interacting cell parts is shown to occur when the pre-conditions of both input and output transitions are satisfied. This occurred in steps 2, 5, 8, 11, 12 and on the output of the counter (br_0) in step 13.

In steps 2, 5, 8, 11, 12 and 13 each highlighted Place indicates that cell part is in the initial state (token in initial place and the cell part on its right has completed its computation for *n*). This return to initial state action began from the MSB cell left part, and gradually moved towards the LSB cell when a cell part performs a computation on its *br* channel.

After completion of the first count sequence (output on b_0), a token is placed in $P0$ as shown in line 13. This action enabled transition ar_0 because the right cell operation of the LSB is a (+1) thus, the second count sequence is shown in the output on line 14. For an even count modulo, the first ar_0 transition would not appear in step 1, because the LSC right part performs a pass operation which requires an input transition on cr_0. After the first count sequence, a step is skipped before the start of the next count sequence for the same reason.

The delay noticed after the first transition on ar_0 in the first count sequence is eliminated in subsequent count sequences because as cell parts return to the initial state, transitions for the next count sequence are enabled and can even fire before the end of the active count sequence. This is shown in lines 9, 10, 12 and 13 where transitions events on ar_2, br'_2, ar'_1 and ar_1 occurred respectively.

5.7 Loadable Modulo$-n$ Counter decomposed into control and counter blocks

In section 5.5, the possible operations of the counter cell parts were defined as follows:

- Odd operation (Left Part)
  - CL_{N_T-1}: Spacer (-), Zero (0).
  - CL_i: Spacer (-), Zero (0) and $\times 2$.
- Even operation (Right Part)
  - CR_{N_T-1}: Spacer (-), $+1$.
  - CR_i: Spacer (-), Pass (P) and $+1$.

Each of these operations were specified in high level using LPN in section 5.6.

In order to realize a loadable counter, each cell part must be able to perform all its possible operations, which will be determined by the sequence of bits for the count modulo n. Therefore, each counter cell part requires a corresponding control cell part to configure its operations.

Figure 5.7 shows the block diagram of the loadable modulo$-n$ counter with cell ith cell (C_i) which contains a counter cell c_i and ith control cell (c'_i).

The counter cell c_i performs the even and odd operations earlier defined. The control cell c'_i is the counter that determines the active counter cells from the binary sequence of n and the correct even or odd operation for each active cell parts.

The combination of cells dedicated to performing the counting function is called the counter block and those which perform configuration function are called the control block.

In Figure 5.7, the control and counter blocks each contain equal number of cells in a bijection interaction. The wrapper shown in the control block provides an interface between

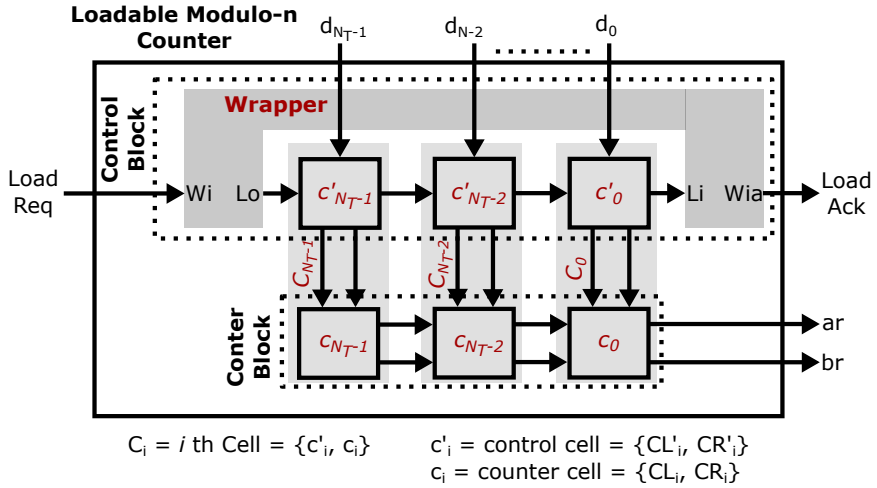


Fig. 5.7 Block diagram of the loadable modulo$-N$ counter.

the environment and the control cells on input *Wi* through which it receives a load request and an output *Wia* through which it sends a load acknowledgement.

When a load request is received, the control block encodes from the input binary number the appropriate activation and configuration codes for each cell in the counter block.

The loadable modulo$-n$ counter CT_n is now defined as:

$CT_n = \{C_i \mid i \in \mathbb{N}, 0 \leq i \leq N_T - 1\}$ is a finite set of cells, N_T is the total number of cells in an implemented counter.

$C_i = (c'_i, c_i)$, is a pair of control and counter cells respectively.

The control block c' defined as $c' = \{c'_i \mid i \in \mathbb{N}, 0 \leq i \leq N_T - 1\}$ is a finite set of interacting control cells, which determines from the binary value of n the correct configuration for each counter cells.

$c'_i = (CL'_i, CR'_i)$ is a pair of control cell parts.

The counter block c defined as $c = \{c_i \mid i \in \mathbb{N}, 0 \leq i \leq N_T - 1\}$ is a finite set of counter cells.

The active counter cells in a count sequence c_i defined as $c_i = \{c_i \mid i \in \mathbb{N}, 0 \leq i \leq n - 1\}$ form a finite set of counter cells directly mapped to a valid bit of n, where $N \leq N_T$.

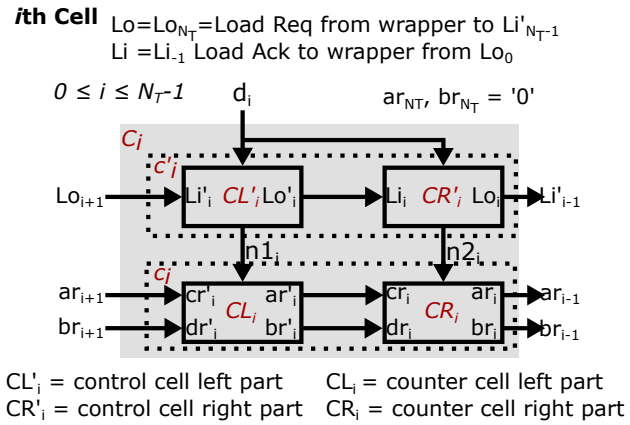


Fig. 5.8 Decomposed control and counter cells left and right parts

$c_i = (CL, CR)$ is a counter cell with a pair of left and right parts, which performs even and odd operations respectively.

For cell C_i, the control cell c'_i parts CL'_i and CR'_i directly configures only the counter cell C_i parts CL_i and CR_i respectively.

Figure 5.8 illustrates a high-level interaction between left and right part CL' and CR' of a control cell c'_i and left and right part CL and CR of a counter cell c_i using signal names in each cell part and an arrow to indicate origin and destination of an action between each cell part. This form is suitable for high-level specification of a cell part operations using LPN, in which signal names are used to represent events.

Specifying each cell part operation using LPN, allows easy modelling and verification of the counter operation by unfolding of actions in which a directed graph is used to show interactions between signal names [94].

In asynchronous systems, a valid communication involves a handshake between computing units. The operation of each cell part is specified using STGs in which explicit signal transitions (rising and falling) form a 4-phase handshake interaction protocol [32].

To specify CL', CR', CL and CR using STG, the signal names are first refined to request and acknowledge signals pairs, as shown in Figure 5.9. The request signals are further refined in single or dual rail to encode the different request data in an interaction channel correctly.

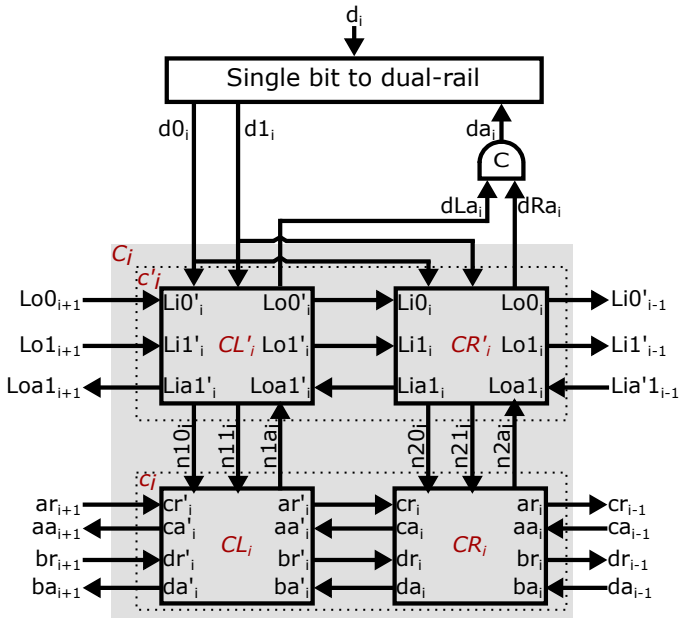


Fig. 5.9 Block diagram of cell parts showing refined signal names

5.8 Specification of Combined Operations in Each Counter Cell Part

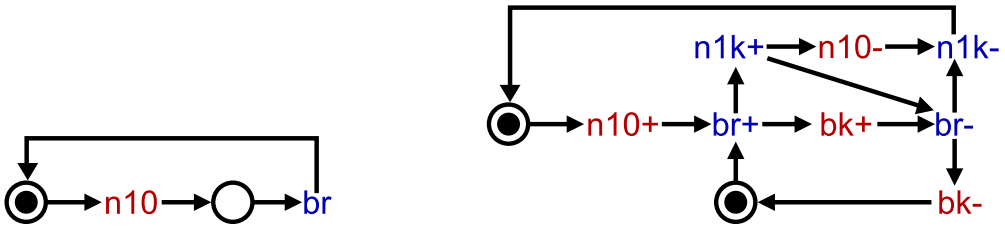
In this section, all possible operations for a counter cell part are combined and specified. The aim is to allow a choice of operation to be activated by a combination of a count event on either *ar* or *br* channel inputs from an adjacent counter cell part on the left and a configuration request input from an adjacent control cell part. The left parts are specified first before the the right cells. The left and right parts of the most significant cell (MSC) performs fewer operation than their corresponding parts in other cells, hence the left and right parts of the MSC are specified separately.

For signal/port name references, refer to Figure 5.8 for high level specifications and Figure 5.9 for low level specifications.

5.8.1 Counter Cell Part CL_{N_T-1} Specification

CL_{N_T-1} is the left part of the most significant counter cell in an implemented counter. As has been defined in section 5.5, this cell part can only perform a spacer and zero operations. The specifications for CL_{N_T-1} is shown in Figure 5.10.

5.8.1.1 High Level Specification



(a) LPN of CL_{N_T-1}, showing a zero operation

(b) STG of CL_{N_T-1}, showing a zero operation

Fig. 5.10 High and low level specifications for CL_{N_T-1}

Figure 5.10a shows the LPN specification for CL_{N_T-1} operations. It states that an input transition on $n10$ results in an output transition on br'. The firing of br' is a zero operation which will be propagated to the least significant cell output through the br channel in each cell part. Figure 5.10b shows the STG specification for CL_{N_T-1}.

5.8.1.2 Circuit Synthesis and Implementation

Figure 5.11 shows the synthesised circuit for CL_{N_T-1}.

5.8.2 Counter Cell Part $CL_i(0 \leq i < N_T - 1)$

CL_i is the left part of a counter cell other than the most significant counter cell in an implemented counter. This cell part can perform a spacer, a $\times 2$ and zero operations.

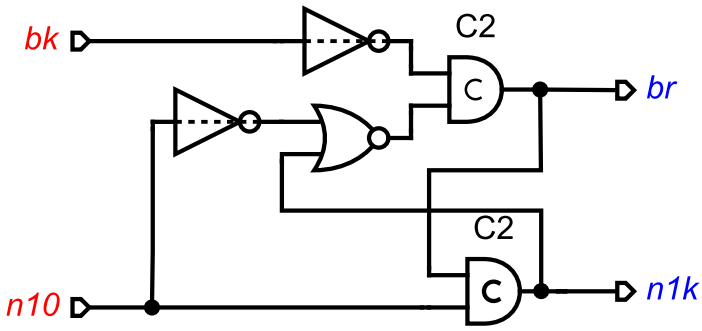


Fig. 5.11 Synthesised circuit of CL_{N_T-1} STG

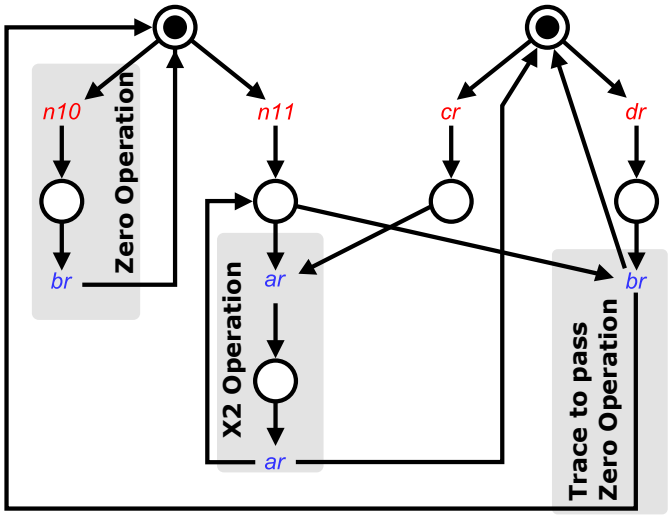


Fig. 5.12 LPN of CL_i, with all possible operations

5.8.2.1 High Level Specification

Figure 5.12 shows the LPN specification for CL_i. The zero operation is activated by a configuration request input on $n10$ from CL_i', while the $\times 2$ operation is activated by a configuration request input on $n11$ and a count input event on cr'. This cell part also contains a channel for passing events on the br channel. This channel is activated by a configuration input on $n11$ and a zero event on dr'.

5.8.2.2 Low Level Specification

In Figure 5.13, a choice transition $cr'+$ disables $dr'+$ and activates the ar channel of CL. In this state, n1 = $n10 = 0, n11 = 1$ configuration request input activates the $\times 2$ operation, which

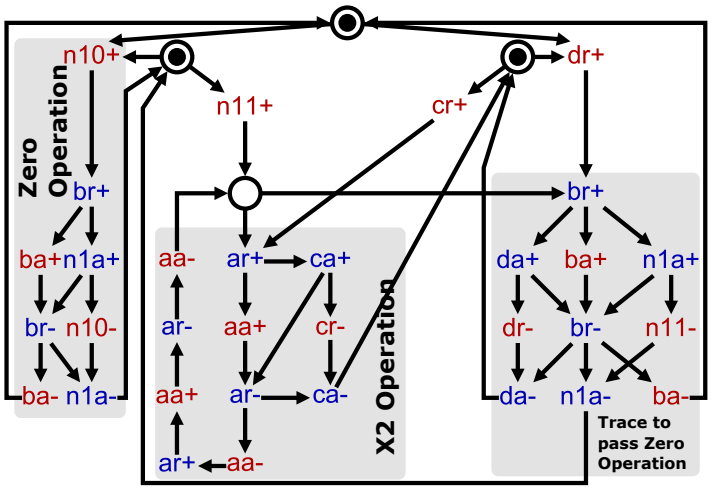


Fig. 5.13 STG of CL_i, with all possible operations

will result in two $ar'+$ transitions. Configuration request input n1 = $n10 = 1, n11 = 0$ enables the zero operation and br' will fire. A combination of $dr'+$ or $cr'+$ with configuration input n1 = $n10 = 0, n11 = 1$ opens a pass operation on the br channel.

In the STG of Figure 5.14, internal signals were added to resolve CSC issues in Figure 5.13.

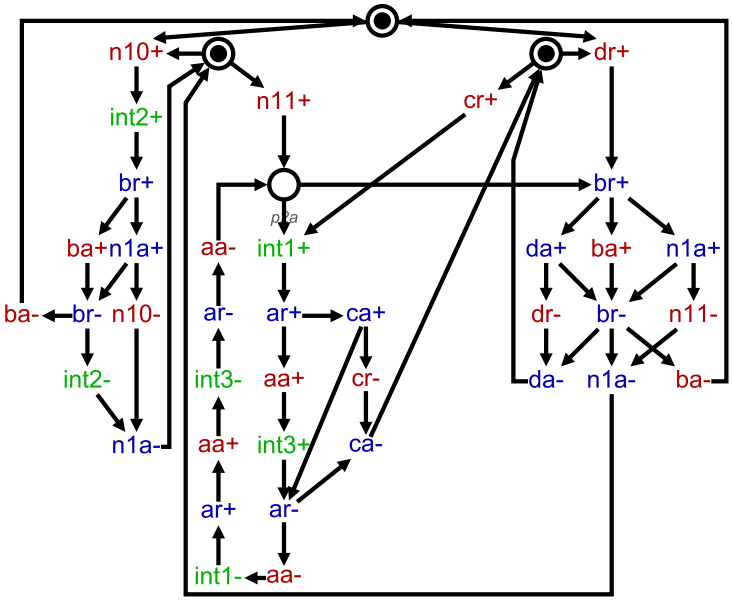


Fig. 5.14 STG of CL_i, with csc resolved

5.8.3 Counter Cell Part CR_{N_T-1}

CR_{N_T-1} is the right part of the most significant counter cell in an implemented counter. This cell part can perform a spacer and $+1$ operations.

5.8.3.1 High Level Specification

Figure 5.15 shows the LPN specification for CR_{N_T-1}. The $+1$ operation is activated by a configuration input on $n21$ from CR'_{N_T-1} after which an input event dr opens the br channel.

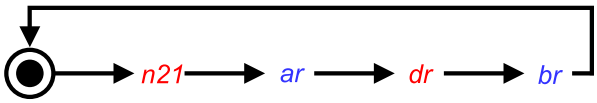


Fig. 5.15 LPN of CR_{N_T-1}, showing a +1 operation

5.8.3.2 Low Level Specification

In Figure 5.16, n2 = $n20 = 0, n21 = 1$ activates the $+1$ operation, which will result one $ar'+$ event after which a $dr+$ results in a pass operation on the br channel.

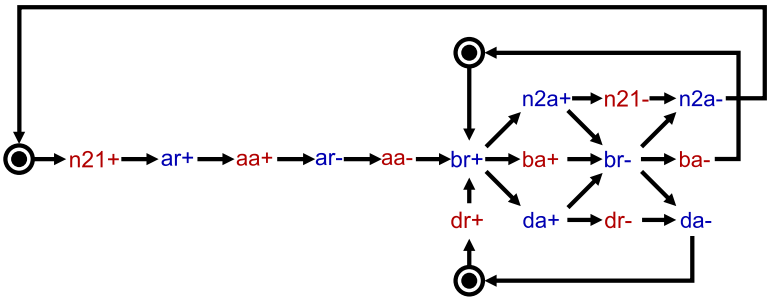


Fig. 5.16 STG of CR_{N_T-1}, showing a +1 operation

In STG of Figure 5.17, internal signals were added to resolve CSC issues in Figure 5.16.

5.8.4 Counter Cell Part $CR_i(0 \leq i < N_T - 1)$

CR_i is the right part of a counter cell other than the most significant counter cell in an implemented counter. This cell part can perform a spacer, $+1$ and pass operations.

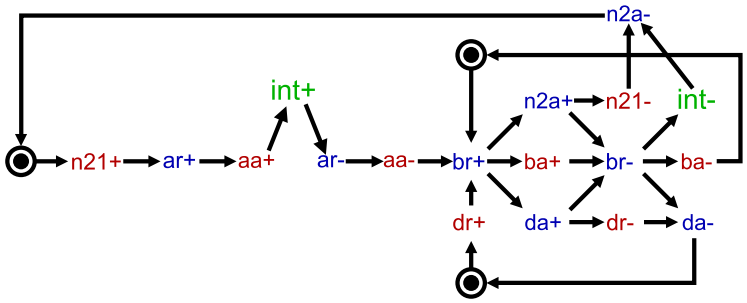


Fig. 5.17 STG of CR_{N_T-1}, with csc resolved

5.8.4.1 High Level Specification

Figure 5.18 shows the LPN specification for CR_i. The Pass operation is activated by a configuration request input on $n20$ or $n21$ from CR'_i and an count input event on cr which opens the ar channel or a zero input event on dr which opens the br channel. A configuration request input on $n21$ adds a ar transition to the pass operation.

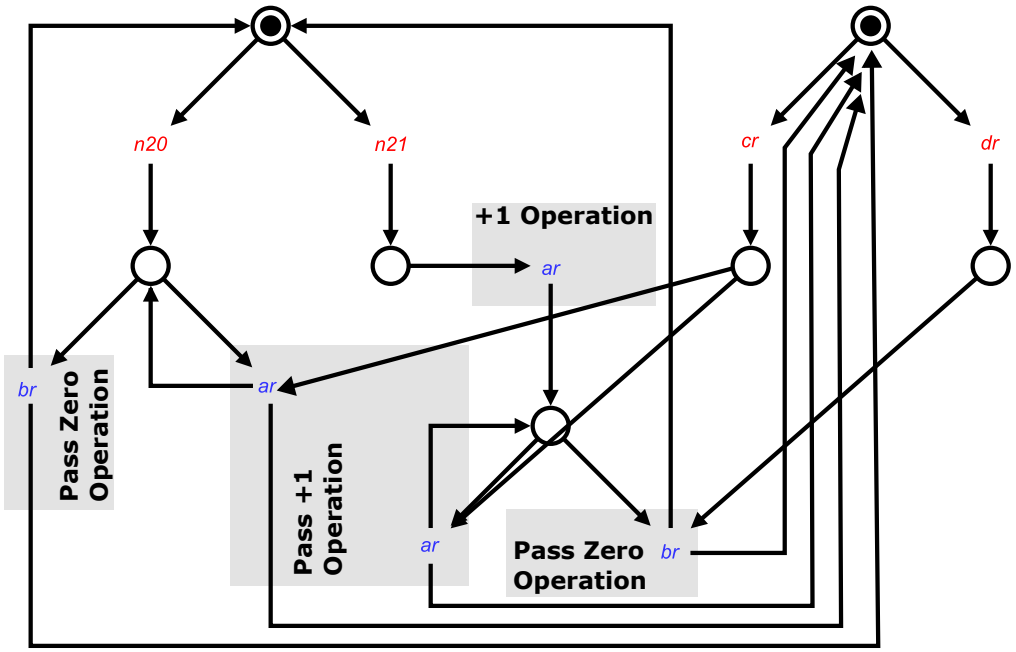


Fig. 5.18 LPN of CR_i, showing a +1 and Pass operations

5.8.4.2 Low Level Specification

In Figure 5.19, a choice transition $cr+$ disables $dr+$ and activates the ar channel of CR. In this state, n2 = $(n20 = 0, n21 = 1)$ configuration input activates the $+1$ operation, which will result one $ar'+$ event after which a $cr+$ or $dr+$ results in a pass operation in ar or br channel respectively. Similarly, $dr+$ transition and configuration input n2 = $n20 = 1, n21 = 0$ opens the br channel for a pass operation.

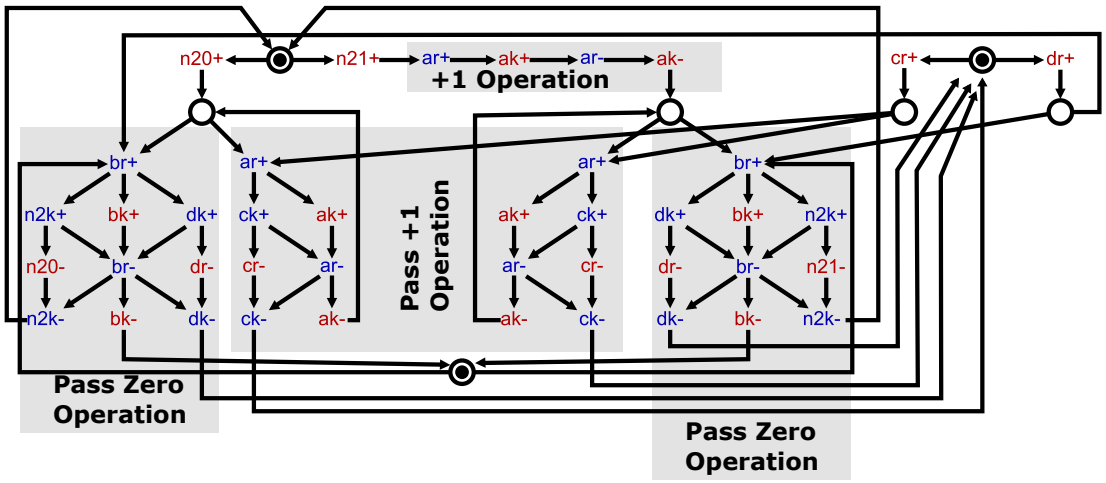


Fig. 5.19 STG of CR_i, showing a +1 and Pass operations

In STG of Figure 5.20, internal signals were added to resolve CSC issues in Figure 5.19.

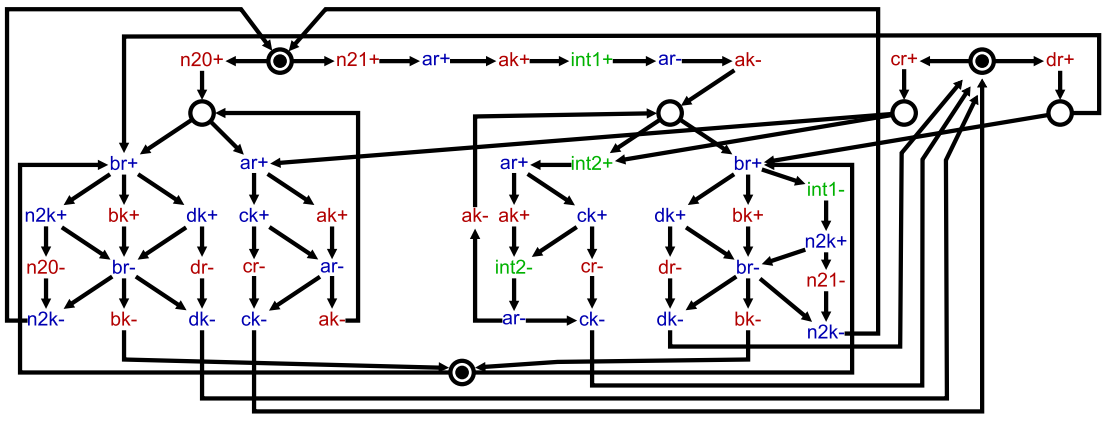


Fig. 5.20 STG of CR_i, with csc resolved

5.9 Specification of Control Block Parts

In this section, the wrapper, control cells left and right parts are specified using a high-level language (LPNs) and low-level level language (STGs). STGs containing CSC conflicts were resolved by the careful addition of internal signals [95]. The logic circuit for each cell part is synthesised from an STG without CSC conflicts which also satisfied the requirements of consistency, deadlock freeness, input properness and output persistence.

The steps in this section are:

- Definition and decomposition of interaction channels between control cell parts (including the wrapper and the digits representing the count modulo) into single-rail or dual-rail channels. Figure 5.8 and Figure 5.9 already illustrate this decomposition showing single-rail or dual-rail request signals and an acknowledgement signal in a channel.

- Specification of each cell part operation using LPN and STG, leading to synthesis and implementation of the circuits.

5.9.1 Channel Encodings

In this section, the channels of interactions between the wrapper and control cell parts, between control cell parts and between control cell parts and counter cell parts are refined into single or dual rail request and acknowledge pairs to allow the encoding of all possible communications in a channel. Please refer to Figure 5.7, 5.8 and 5.9.

Two channels of communication are defined for a control cell part. These channels are the Load Channel and the Configuration Channel.

The "Load Channel" processes a load request from the environment between the wrapper and control cells and between control cells across the control block.

The "Configuration Channel" exists between each control cell part and its corresponding counter cell part to allow for encoding of a count operation request from the former to the later.

Table 5.5 Relationship between refined signal names of Figure 5.9.

	Inputs		
Cell Part	CL'_{N_T-1}	CL'_i	CR'_i
Load Channel	Li'	Li0', Li1'	Li0, Li1

Table 5.6 Relationship between refined signal names of Figure 5.9.

	Outputs			
Cell Part	CL'_{N_T-1}	CL'_i	CR'_{N_T-1}	CR'_i/CR'_0
Load Channel	Lo0', Lo1'	Lo0', Lo1'	Lo0, Lo1	Lo0, Lo1/Lo
Configuration Channel	n10	n10, n11	n21	n20, n21

Tables 5.5 and 5.6 lists the refined input and output request signal names of each control cell part. Channels with only one and two signal name(s) indicate refinement into single and double rails respectively. In the proceeding sub-headings, the conditions and encoding of these channels are presented.

5.9.1.1 Single Bit to Dual-Rail Conversion of d_i

Each bit of the count modulo d_i is refined to dual-rail $d0_i$ and $d1_i$ to encode a valid bit request as "00" for '-', "10" for '0' and "01" for '1'.

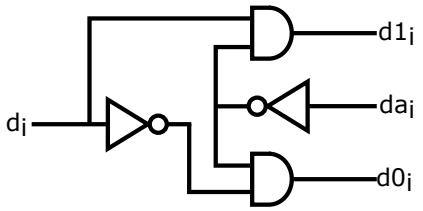


Fig. 5.21 Single Bit to Dual-Rail Conversion Circuit

The schematic of Figure 5.21 converts a single bit to its dual rail equivalent. The inverter which is fed by the da_i input receives an acknowledgement from control cell c'_i after which

both outputs $d0_i$ and $d1_i$ will be forced to '0's. Thus, the circuit output is seen by c'_i is in a 4-phase request and acknowledge handshake sequence.

5.9.1.2 Wrapper

The load requests between the wrapper and the control cell parts are encoded in single-rail because $\nexists d_j > d_{N_T-1}$ on the CL'_{N_T-1} end and $\nexists c'_j < c'_0$ on the CR'_0 end respectively, therefore only a single-rail load request is sufficient. The channel *Lo* is refined to Request: *Lo* and acknowledgement *Loa* in the interface between wrapper and CL'_{N_T-1}, while the channel *Li* is refined to Request: *Li* and acknowledgement *Lia* in the interface between the wrapper and CR'_0. Please refer to Figure 5.7.

5.9.1.3 Control Cell Parts

A control cell part communicates a load request and load status along the load channel. A control cell communicates the configuration operation to its adjacent counter cell part on the configuration channel.

For CL'_i or CR'_i to load their corresponding bit d_i and correctly configure its corresponding counter cell part CL_i or CR_i respectively, it needs to be informed if there exist d_j = '1' where $j \geq i$. Thus load channels between control cell parts are encoded in dual-rail.

The encoded load request and conditions between control cell parts are:

- "00": if load request $= 0$
- "10": if $\forall j \geq i : d_j = 0 \wedge$ load request $= 1$.
- "01": if $\exists j \geq i : d_j = 1 \wedge$ load request $= 1$.

Where $i = \{0 \leq i \leq N_T - 1\}$

CL'_i and CR'_i interact with their corresponding counter cell parts CL_i and CR_i through configuration channels n1 and n2 respectively. Each channel is refined to dual-rail request and an acknowledge pair to accommodate all possible configuration requests.

The encoded configuration request and conditions between control and counter cell parts for cell C_i are:

- $i = N_T - 1$ (most significant cell (MSC) in an implemented counter)

  Left Part Control

  $$n1 = \begin{cases} - & if\ d_i = 0 :\ \text{(n10 = '0')} \\ 0 & if\ d_i = 1 :\ \text{(n10 = '1')} \end{cases}$$

  Right Part Control

  $$n2 = \begin{cases} - & if\ d_i = 0 :\ \text{(n21 = '0')} \\ +1 & if\ d_i = 1 :\ \text{(n21 = '1')} \end{cases}$$

  - Refinement: Single-rail for both cell parts n1 = n10, n2 = n21

- For $0 \leq i < N_T - 1$ (all control cells other than the MSC)

  Left Part Control

  $$n1 = \begin{cases} - & if\ \forall j \geq i : d_j = 0 :\ \text{(n10 = '0', n11 = '0')} \\ 0 & if\ \forall j > i : d_j = 0 \wedge d_i = 1 :\ \text{(n10 = '1', n11 = '0')} \\ \times 2 & if\ \exists j > i : d_j = 1 :\ \text{(n10 = '0', n11 = '1')} \end{cases}$$

  Right Part Control

  $$n2 = \begin{cases} - & if\ \forall j \geq i : d_j = 0 :\ \text{(n20 = '0', n21 = '0')} \\ P & if\ \exists j > i : d_j = 1 \wedge d_i = 0 :\ \text{(n20 = '1', n21 = '0')} \\ +1 & if\ d_i = 1 :\ \text{(n20 = '0', n21 = '1')} \end{cases}$$

  - Refinement: Dual-rail for both cell parts n1 = n10, n11, n2 = n20, n21.

5.9.2 Wrapper Specification

The wrapper interfaces with the environment from which the count request (load request) is produced and the control cells in the block. The wrapper ensures that the environment can only receive an acknowledgement to a count request after all the control cells have received a load request. High-level and low-level specification of the wrapper is now presented.

5.9.2.1 High Level Specification

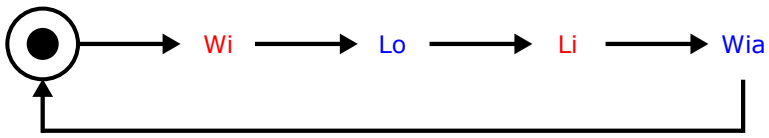


Fig. 5.22 LPN of the Wrapper.

Figure 5.22 is a high-level specification of the wrapper using LPN. Transition *Wi* is an input port through which a request is sent from the environment to the wrapper. The wrapper interfaces with CL'_{N_T-1} on its output port *Lo*, through which it sends the received load request to the control cells.

The wrapper also interacts with CR'_0 on its input port *Li*. The control cell part CR'_0 sends a load request to the wrapper after all control cells have received the load request. This input to the wrapper results in an output on *Wia* port to the environment. This output appears to the environment as an acknowledgement of the load request sent to the counter.

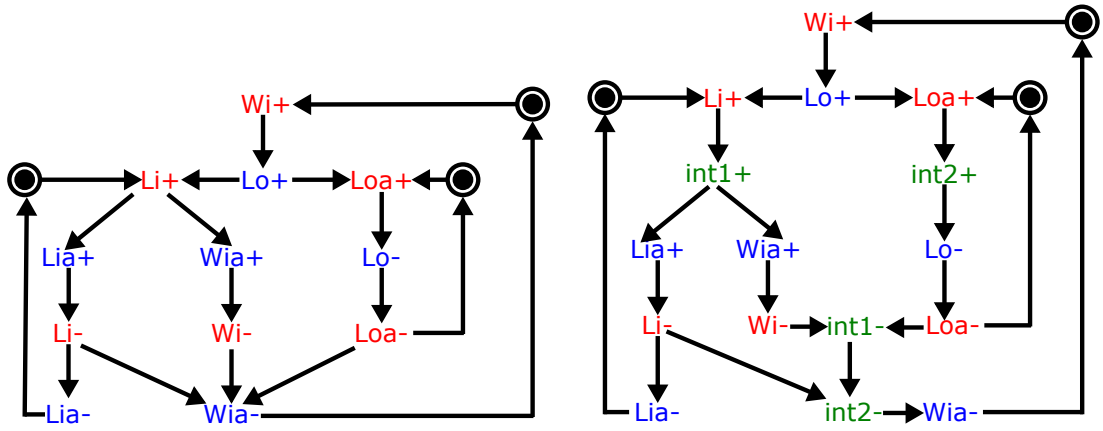
The specification of the wrapper ensures that until a previous Load command has been acknowledged, a new load command is ineffective.

5.9.2.2 Low Level Specification

The channel between the wrapper and CL'_{N_T-1} is refined to single rail request and acknowledgement *Lo* and *Loa* respectively. Similarly, the channel between CR'_0 and the wrapper is refined to single rail request and acknowledgement *Li* and *Lia* respectively. Single rails

encoding between the wrapper and control cell parts CL'_{N_T-1} and CR'_0 is sufficient to encode a load request and a load acknowledgement from and to the environment.

Figure 5.23a is a low-level specification of the wrapper using STG. It specifies the wrapper will send a request $Lo+$ to CL'_{N_T-1} after a load request $Wi+$ is received from the environment.



(a) STG specification of the wrapper

(b) STG with internal signals added to resolve CSC conflicts

Fig. 5.23 Specification of the wrapper

This action enables two parallel routes which are listed below.

1. The wrapper receives an acknowledgement $Loa+$ from CL'_{N_T-1} to load request output $Lo+$ and the handshake between the wrapper and CL'_{N_T-1} is independently completed.

2. The wrapper receives an input Load Complete request transition $Li+$ from CR'_0 after loading is complete. This also opens two parallel routes.

   (a) An acknowledgement $Wia+$ is sent to the environment and the load request is subsequently withdrawn ($Wi-$).

   (b) The handshake sequence between the wrapper and CR'_0 is completed after the wrapper sends an acknowledgement to CR'_0 on the output port $Lia+$.

After sub-route 2a, the signal Wia can only be lowered when routes 1, 2 and 2b have com-

pleted their transition sequence. It is only after this the environment can send another load request for a new count modulo n.

The STG of Figure 5.23a does not present a unique state coding for every possible transition. This can result in complete state coding issues (CSC) which must be resolved in order to synthesize the correct and safe circuit.

In Figure 5.23b, internal signals have been added to resolve CSC conflicts. The wrapper circuit is synthesised from this specification and implemented using standard cells from AMS 350 nm CMOS technology.

In Table 5.7, measurements from the simulation of implemented circuits are shown for the response time of the wrapper when a load request is received from the environment and when a load complete is received from CR'_0.

Table 5.7 Response Time of the wrapper

Response Time (*ns*)						
	1.8V	**2.0V**	**2.4V**	**2.8V**	**3.0V**	**3.3V**
Lo	1.05	0.87	0.67	0.55	0.51	0.47
Wia	2.24	1.86	1.42	1.18	1.09	1.0

5.9.3 Control Cell CL'_{N_T-1}pecification

CL'_{N_T-1} encodes a load request to CR'_{N_T-1} and a configuration request to CL_{N_T-1} when activated by a combination of load request from the wrapper and the bit d_{N_T-1} value. It can only configure CL_{N_T-1} to operate as a spacer or zero. These encoded requests are activated through two traces in the LPN of Figure 5.24.

5.9.3.1 High Level Specification

Trace 1 of LPN in Figure 5.24 is activated when a load request is received from the wrapper and the bit d_{N_T-1} is '0', (dual-rail equivalent $d0$ and $d1$ = '1' and '0'). This trace encodes

a load request to CR'_{N_T-1} by a transition on $Lo0'$ and configures CL_{N_T-1} to operates as a spacer.

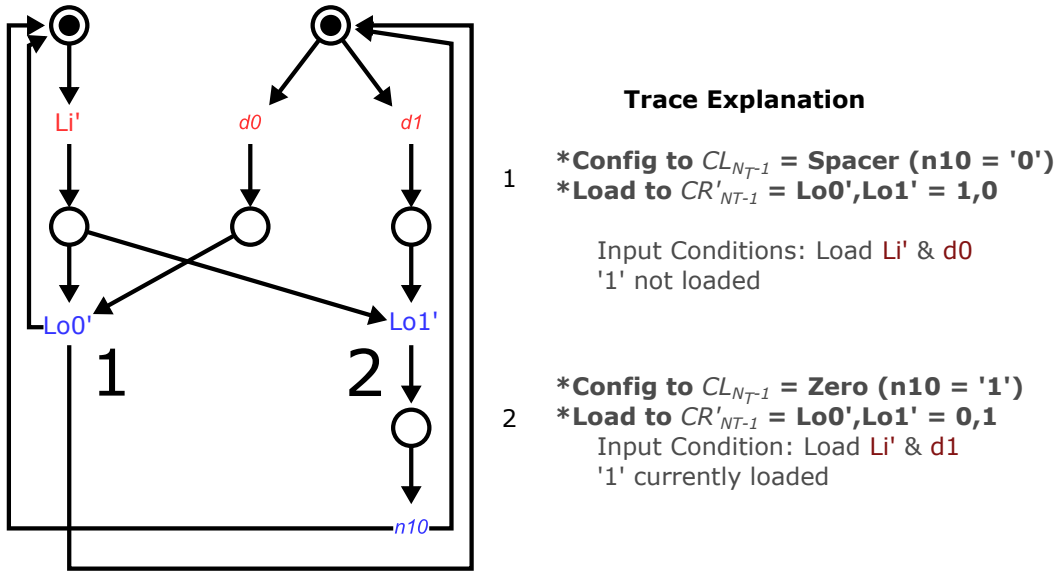


Fig. 5.24 LPN of MSC left control cell part (CL'_{N_T-1}).

Trace 2 is activated when a load request is received from the wrapper and the bit d_{N_T-1} is '1', (dual-rail equivalent $d0$ and $d1$ = '0' and '1'). This trace encodes the load request sent to CR'_{N_T-1} on transition $Lo1'$ which is an indication that the condition d_i = '1' has occurred. This trace also encodes a configuration request to CL_{N_T-1} to perform a "Zero" operation.

5.9.3.2 Low Level Specification

CL'_{N_T-1} is specified in the STG of Figure 5.25. The STG of Figure 5.26 shows internal signals added to resolve CSC.

5.9.4 Control Cell $CL'_i(0 \leq i < N_T - 1)$

controlcell2

CL'_i encodes a load request to CR'_i and a configuration request to CL_i when activated by a combination of an encoded load request from CR'_{i+1} and the bit d_i value. It can configure

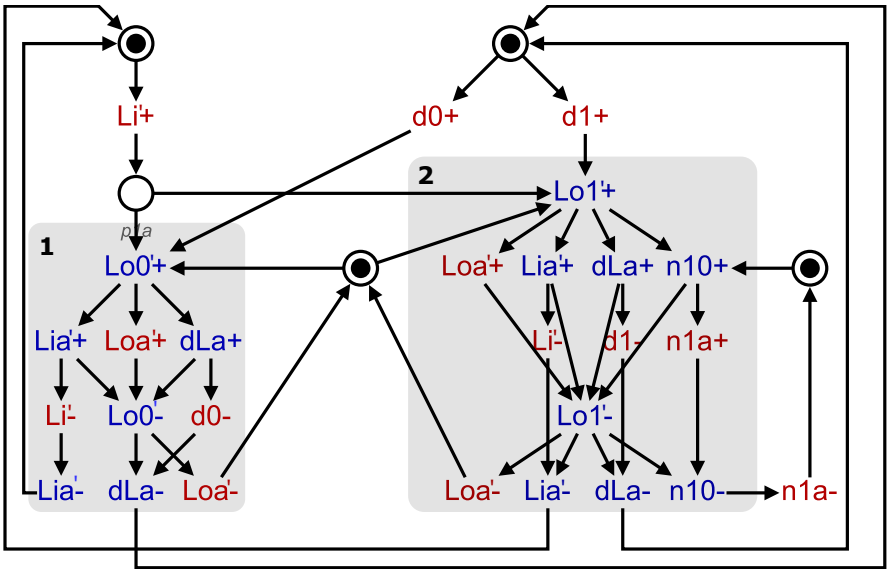


Fig. 5.25 STG specification for CL'_{N_T-1}

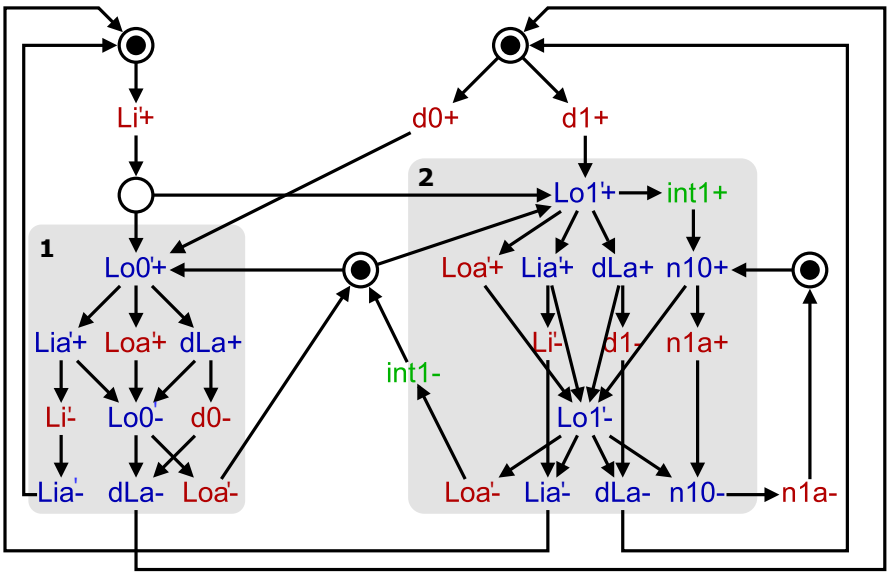


Fig. 5.26 Resolved CSC conflicts for STG Fig 5.25

CL_i to operate as a spacer, a zero or a $\times 2$. These encoded requests are activated through four traces in the LPN of Figure 5.27.

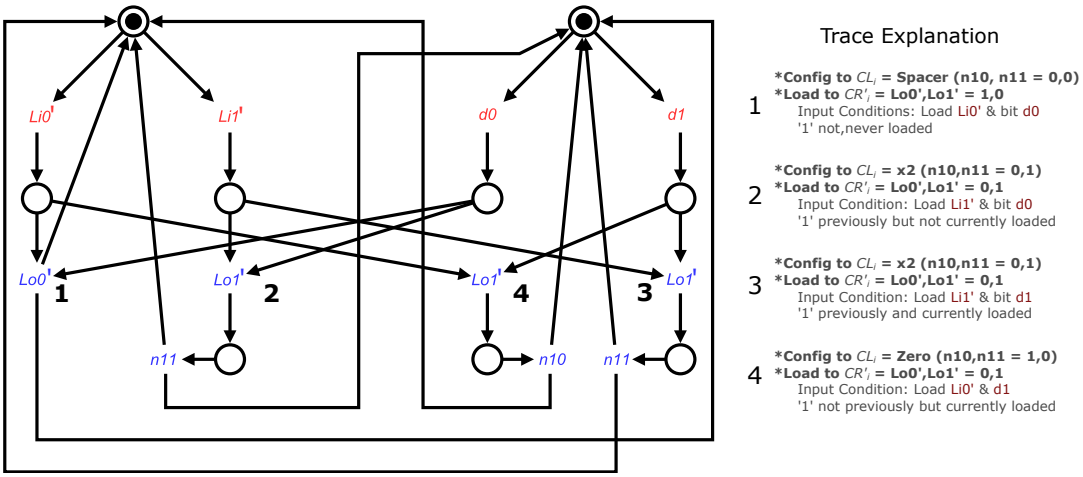


Fig. 5.27 LPN of left control cell part (CL'_i, $0 \leq i < N_T - 1$)

5.9.4.1 High Level Specification

Trace 1 of LPN in Figure 5.27 is activated when the encoded load request ($Li0'$) received from CR'_{i+1} indicates that $\forall j > i : d_j =' 0' \wedge d_i =' 0'$, (dual-rail equivalent $d0$ and $d1$ = '1' and '0''). This trace only informs CR'_{i-1} of the load request by a transition on $Lo0'$ and encodes a configuration request to CL_i to operates as a spacer.

Trace 2 is activated when the encoded load request ($Li1'$) received from CR'_{i+1} indicates that $\exists j \geq i : d_j = 1 \wedge d_i =' 0'$, (dual-rail equivalent $d0$ and $d1$ = '1' and '0'). This trace encodes the load request sent to CR'_{i-1} on transition $Lo1'$ which is an indication of the condition $\exists j \geq i : d_j = 1$. This trace encodes a configuration request to CL_i to operate as a $\times 2$ on $n11$.

Trace 3 is activated when the encoded load request ($Li1'$) received from CR'_{i+1} indicates that $\exists j \geq i : d_j = 1 \wedge d_i =' 1'$, (dual-rail equivalent $d0$ and $d1$ = '0' and '1'). This trace encodes the load request sent to CR'_{i-1} on transition $Lo1'$ which is an indication of the condition $\exists j \geq i : d_j = 1$. This trace configures CL_i to operate as a $\times 2$ on $n11$.

Trace 4 is activated when the encoded load request ($Li1'$) received from CR'_{i+1} indicates that $\forall j > i : d_j =' 0' \wedge d_i =' 1'$ is '1', (dual-rail equivalent $d0$ and $d1$ = '0' and '1'). This trace encodes the load request sent to CR'_{i-1} on transition $Lo1'$ which is an indication of the

condition $\exists j \geq i : d_j = 1$. This trace encodes a configuration request to CL_i to operate as a Zero on $n10$.

5.9.4.2 Low Level Specification

CL'_i is specified in the STG of Figure 5.28. The STG of Figure 5.29 shows internal signals added to resolve CSC.

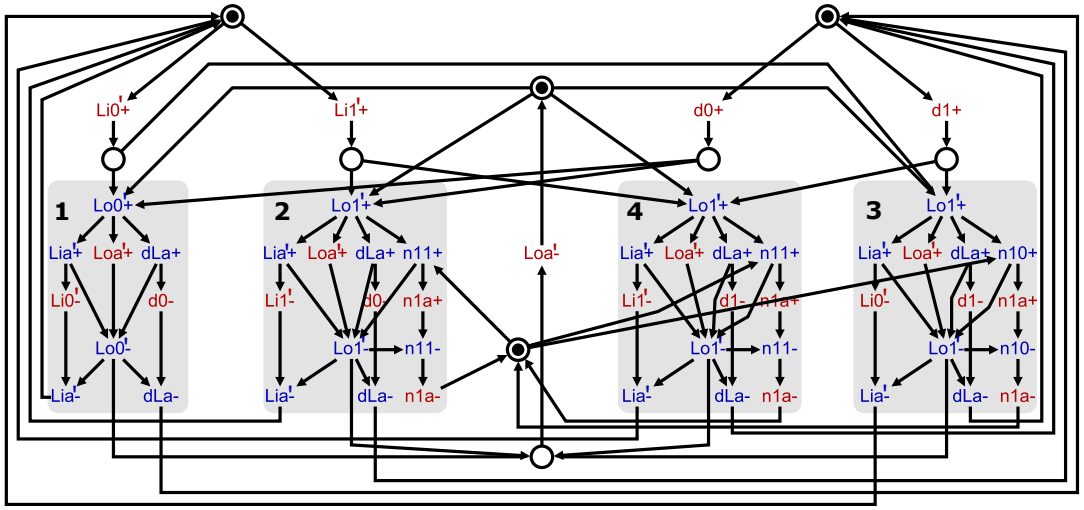


Fig. 5.28 STG of control cell part (CL'_i, $0 \leq i < N_T - 1$)

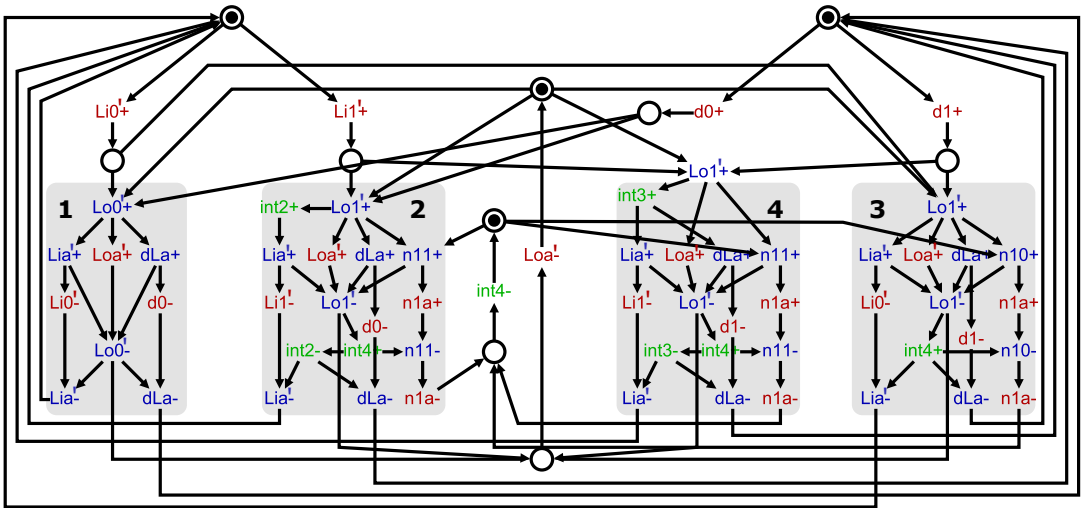


Fig. 5.29 Resolved CSC conflicts for STG of Fig 5.28

5.9.5 Control Cell CR'_{N_T-1}

controlcell3

CR'_{N_T-1} encodes a load request to CL'_{N_T-2} and a configuration request to CR_{N_T-1} when activated by a combination of encoded load request CL'_{N_T-1} and d_{N_T-1}. It can only configure CR_{N_T-1} to operate as a spacer or a +1. These encoded requests are activated through two traces in the LPN of Figure 5.30.

5.9.5.1 High Level Specification

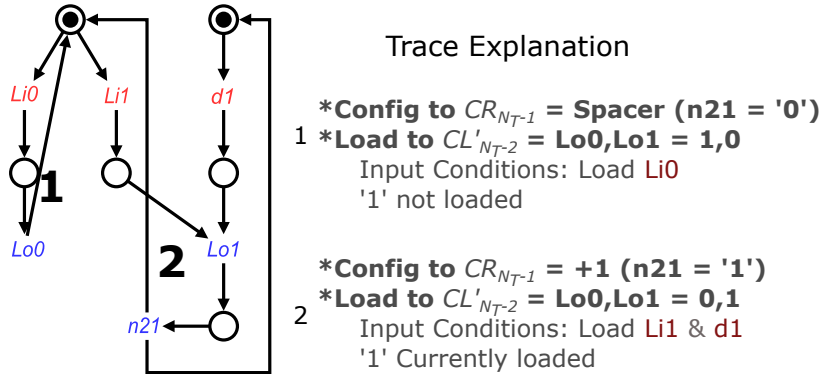


Fig. 5.30 LPN of MSC right control cell part (CR'_{N_T-1}).

Trace 1 of LPN in Figure 5.30 is activated when the encoded load request ($Li0$) received from CL'_{N_T-1} indicates that $d_{N_T-1} =$ '0'. This trace only informs CL'_{N_T-2} of the load request by a transition on $Lo0$ and configures CR_{N_T-1} to operates as a spacer.

Trace 2 is activated when the encoded load request ($Li1$) received from CL'_{N_T-1} indicates that $d_{N_T-1} =$ '1' and $d_{N_T-1} =$ '1'. This trace encodes the load request sent to CL'_{N_T-2} by a transition on $Lo1$ which is an indication of the condition $\exists j \geq i : d_j = 1$. This trace encodes a configuration request to CR_{N_T-1} to operate as a +1 on $n21$.

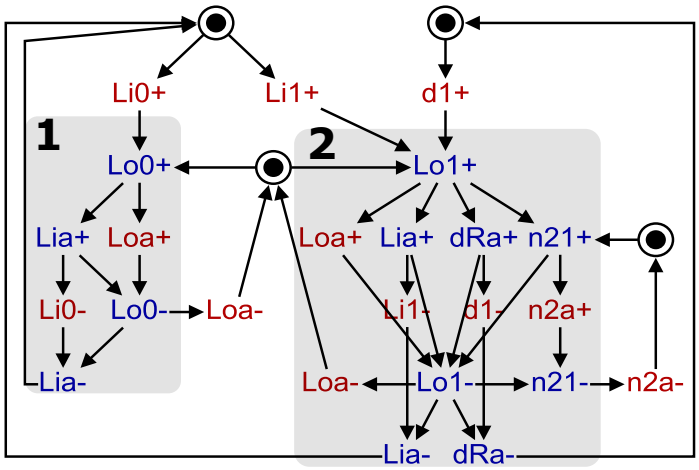


Fig. 5.31 STG specification for CR'_{N_T-1}

5.9.5.2 Low Level Specification

The control cell CL'_i is specified in the STG of Figure 5.31. The STG of Figure 5.32 shows internal signals added to resolve CSC.

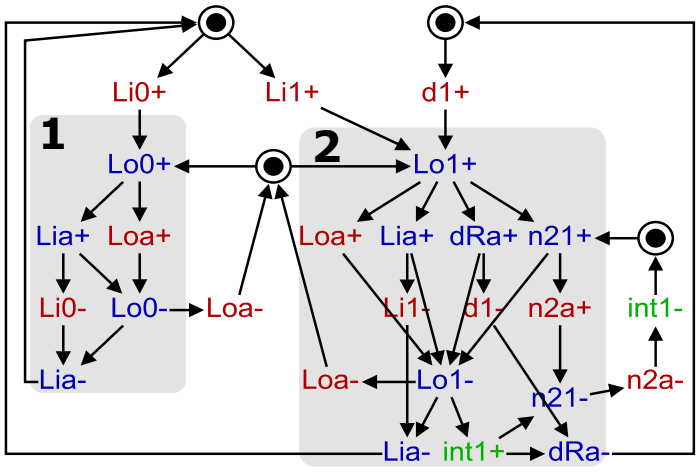


Fig. 5.32 Resolved CSC conflicts for STG of Fig 5.31

5.9.6 Control Cell $CR'_i(0 \leq i < N_T - 1)$

controlcell4

CR'_i encodes a load request to CL'_{i-1} and a configuration request to CR_i when activated by a combination of encoded load request CL'_{i+1} and d_i. It can configure CR_i to operate as a

spacer, a Pass or a +1. These encoded requests are activated through three traces in the LPN of Figure 5.33.

5.9.6.1 High Level Specification

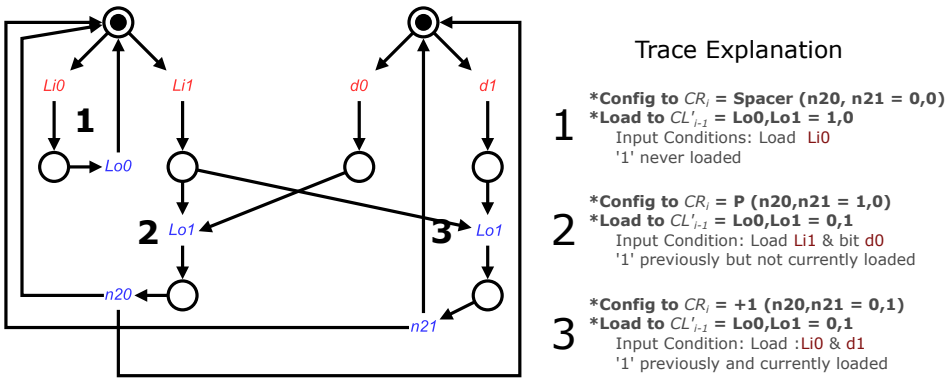


Fig. 5.33 LPN of Right Control Cell Part (CR'_i, $0 < i < n_s - 1$).

Trace 1 of LPN in Figure 5.33 is activated when the encoded load request ($Li0$) received from CL'_{i+1} indicates that $\nexists j > i : d_j = 1$. This trace only informs CL'_{i-1} of the load request by a transition on $Lo0$ and configures CR_i to operate as a spacer.

Trace 2 is activated when the encoded load request ($Li1$) received from CL'_{i+1} indicates that $\exists j > i : d_j = 1$ and $d_i =' 0'$. This trace encodes the load request sent to CL'_{i-1} by a transition on $Lo1$ which is an indication of the condition $\exists j \geq i : d_j = 1$. This trace also configures CR_i to operate as a Pass by a transition on $n20$.

Trace 3 is activated when the encoded load request ($Li1$) received from CL'_{i+1} indicates that $\exists j > i : d_j = 1$ and $d_i =' 1'$. This trace encodes the load request sent to CL'_{i-1} by a transition on $Lo1$ which is an indication of the condition $\exists j \geq i : d_j = 1$. This trace also configures CR_i to operate as a +1 by a transition on $n21$.

5.9.6.2 Low Level Specification

CR'_i is specified in the STG of Figure 5.34. The STG of Figure 5.35 shows internal signals added to resolve CSC.

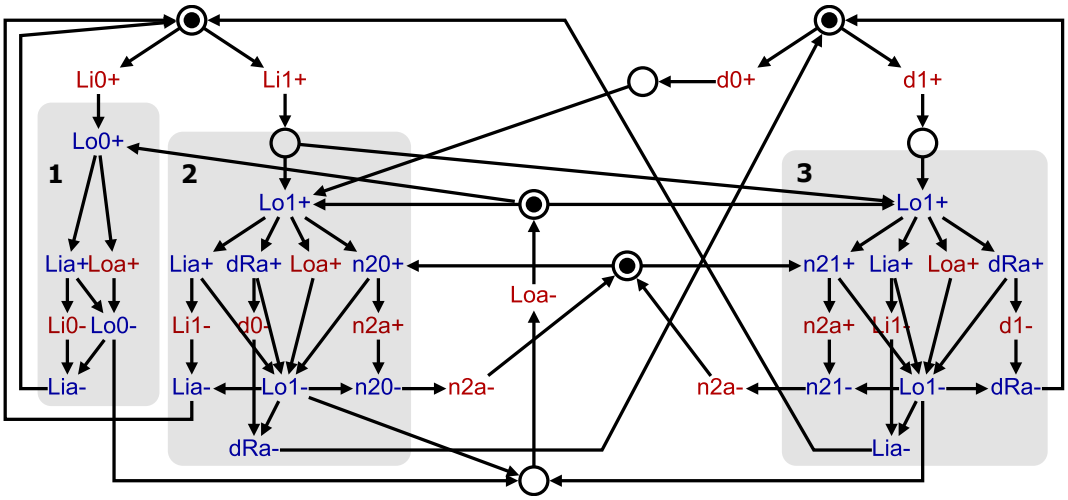


Fig. 5.34 STG specification for CR'_i, $0 < i < N_T - 1$

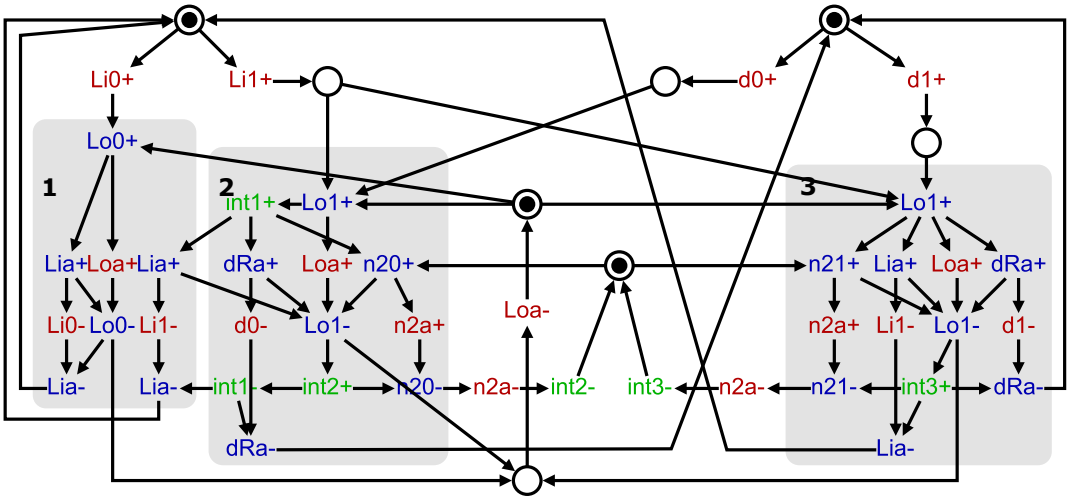


Fig. 5.35 Resolved CSC conflicts for STG of Fig 5.34

5.9.7 Control Cell CR'_0

controlcell5

CR'_0 sends a load request to the wrapper and a configuration request to CR_0 when activated by a combination of encoded load request from CL'_0 and d_0. It can configure CR_0 to operate as a spacer, a Pass or a +1. These encoded requests are activated through three traces in the LPN of Figure 5.36.

5.9.7.1 High Level Specification

Trace 1 of LPN in Figure 5.36 is activated when the encoded load request ($Li0$) received from CL'_1 indicates that $\nexists j > i : d_j = 1$. This trace only informs the wrapper of the load request by a transition on Lo and configures CR_0 to operate as a spacer.

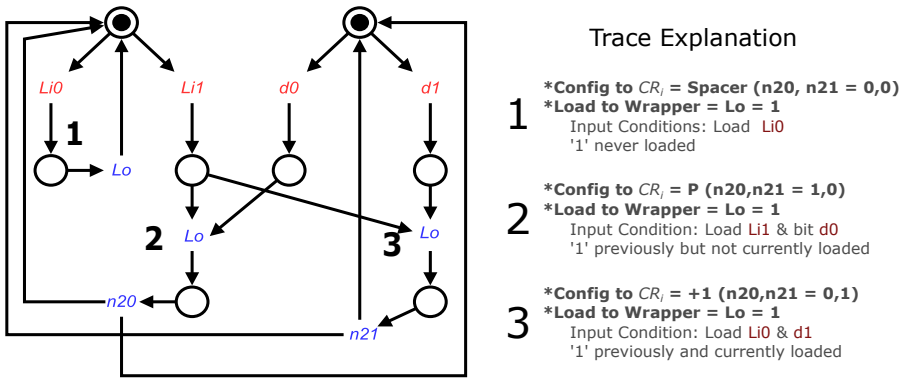


Fig. 5.36 LPN of LSB Right Control Cell Part (CR'_0).

Trace 2 is activated when the encoded load request ($Li1$) received from CL'_1 indicates that $\exists j > 0 : d_j = 1$ and $d_0 =' 0'$. This trace passes the load request to the wrapper by a transition on Lo. This trace also configures CR_0 to operate as a Pass by a transition on $n20$.

Trace 3 is activated when the encoded load request ($Li1$) received from CL'_1 indicates that $\exists j > 0 : d_j = 1$ and $d_0 =' 1'$. This trace passes the load request to the wrapper by a transition on Lo. This trace also configures CR_0 to operate as a +1 by a transition on $n21$.

5.9.7.2 Low Level Specification

CR'_0 is specified in the STG of Figure 5.37. The STG of Figure 5.38 shows internal signals added to resolve CSC.

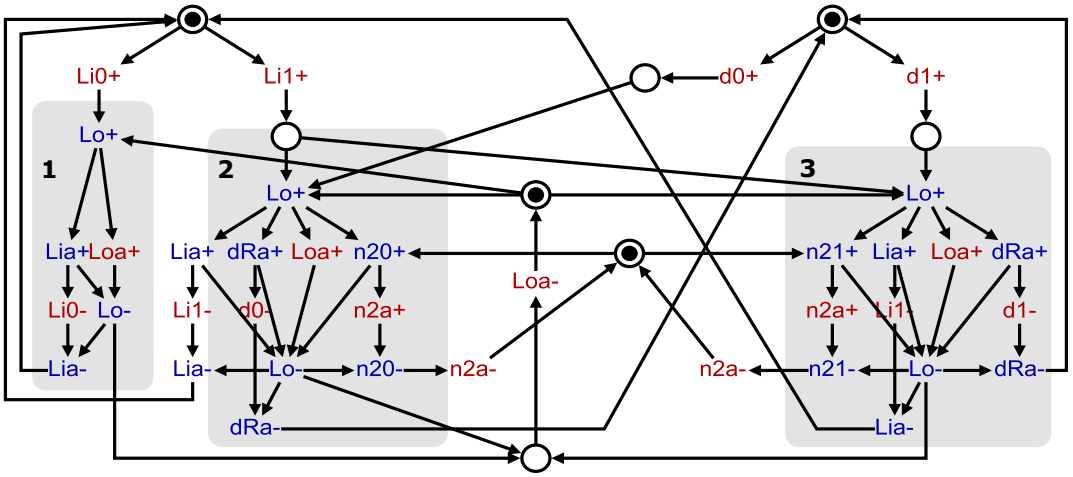


Fig. 5.37 STG specification for CR'_0)

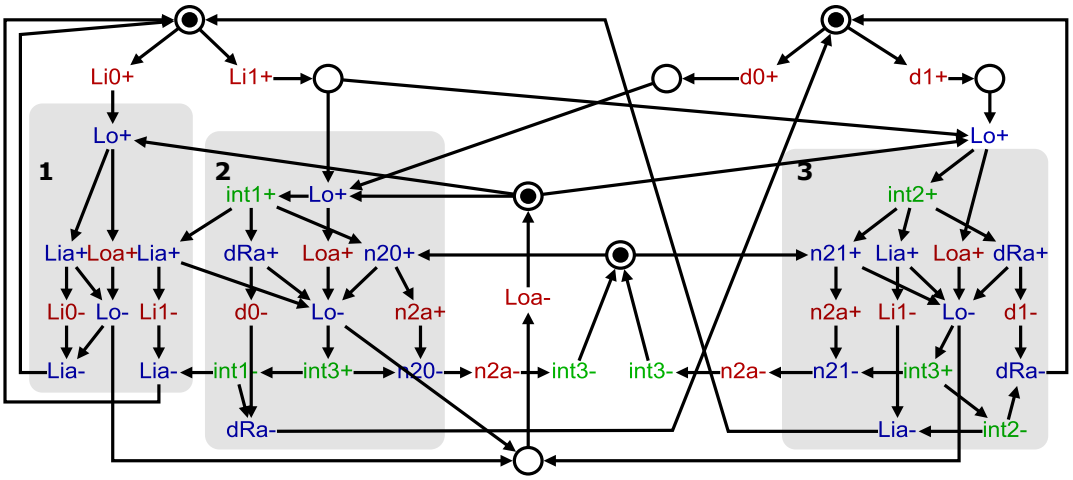


Fig. 5.38 Resolved CSC conflicts for STG of Fig 5.37

5.10 Response Time

Let the term T_x denote the response time of an entity due to the combination of a set of inputs. T_x is defined by lower and upper bounds $\delta T_x \leq T_x \leq \Delta T_x$. In the characterisation presented in this section, the upper bounds of each cell part is more important as this will determine the worst case scenario in the system.

The response time of the wrapper after receiving a load request from the environment is denoted by T_{Wo} and its response time after receiving a load request from CR'_0 be denoted by T_{Wia}.

The response time of CL' and CR' on their load channels is denoted by T'_{Lo} and T_{Lo} respectively. Therefore, the response time for control cell c'_i on the load channel is the sum $(T'_{Lo} + T_{Lo})_i$. Also, the response time of the left and right parts of a control cell configuration channel be denoted by T_{n1} and T_{n2} respectively.

I state that: $\forall CL'_i$, $T'_{Lo} < T_{n1}$ and also, $\forall CR'_i$, $T_{Lo} < T_{n2}$.

The above statements are valid because the STG specifications for CL' and CR' ensured that a transition must occur on the load channel of a control cell part before a configuration request/transition to an adjacent counter cell part occurs.

Let T'_{ar}, T'_{br} and T_{ar}, T_{br} denote the response times on the *ar* and *br* channels of left and right counter cell parts respectively.

5.10.1 Response Time on Load Channel

The time it will take for the counter to send a Load Acknowledgement to the environment after receiving a load request from the environment is given as:

$$T_{LoadAck} = T_{Wo} + T_{Wia} + \sum_{i=0}^{N_T-1} (T'_{Lo} + T_{Lo})_i \tag{5.3}$$

This assumes a valid count modulo n is present at the time the load request is issued.

Tables 5.9 and 5.8 show the response times of each control cell part for all possible input combinations simulated at 3V. The tables also show that the response time on both load and configuration channels of a cell part varied for different input combinations to that cell part.

Table 5.8 Measured Response Time on Load Channel for each Control Cell Part

Cell Part	**Input Comb.**		**Output**	**Response Time**	
	Load	*Bit*	*Load*	*Load Channel (ns)*	*Denoted by*
CL'_{N_T-1}	1	10	10	1.4	T'_{Lo}
CL'_{N_T-1}	1	01	01	1.66	T'_{Lo}
CL'_i	10	10	10	1.18	T'_{Lo}
CL'_i	10	01	01	2.86	T'_{Lo}
CL'_i	01	10	01	2.35	T'_{Lo}
CL'_i	01	01	01	2.77	T'_{Lo}
CR'_{N_T-1}	10	10	10	1.25	T_{Lo}
CR'_{N_T-1}	01	01	01	1.53	T_{Lo}
CR'_i	10	10	10	0.91e-3	T_{Lo}
CR'_i	01	10	01	2.12	T_{Lo}
CR'_i	01	01	01	2.17	T_{Lo}
CR'_0	10	10	1	1.51	T_{Lo}
CR'_0	01	10	1	2.01	T_{Lo}
CR'_0	01	01	1	2.06	T_{Lo}

Table 5.9 Measured Response Time on Configuration Channel for each Control Cell Part

Cell Part	Input Comb.		Output	Response Time	
	Load	*Bit*	*Config.*	*Config. Channel (ns)*	*Denoted by*
CL'_{N_T-1}	1	10	00		
CL'_{N_T-1}	1	01	10	3.45	T_{n1}
CL'_i	10	10	00		T_{n1}
CL'_i	10	01	10	4.09	T_{n1}
CL'_i	01	10	01	4.87	T_{n1}
CL'_i	01	01	01	5.41	T_{n1}
CR'_{N_T-1}	10	10	00		
CR'_{N_T-1}	01	01	01	2.53	T_{n2}
CR'_i	10	10	00		
CR'_i	01	10	10	4.79	T_{n2}
CR'_i	01	01	01	3.48	T_{n2}
CR'_0	10	10	00		T_{n2}
CR'_0	01	10	10	4.76	T_{n2}
CR'_0	01	01	01	4.6	T_{n2}

5.10.2 Response Time on *ar* Channel

The response time T_{ar0} of the counter on ar_0 after load command is sent to the wrapper is expressed thus:

For odd N:

$$T_{ar0} = T_{Wo} + T_{n2} + T'_{Lo} + T_{ar} + \sum_{i=1}^{N_T-1} (T'_{Lo} + T_{Lo})_i \tag{5.4}$$

From equation 5.4, the response time T_{ar_0} is independent of the count modulo n (odd n), but largely dependent on the total number of counter cells.

To derive an expression for the response time when the count modulo n is even, two conditions are considered and they are listed below.

- Load request across the control block is fast enough such that a valid configuration request is present for each counter cell part before the first LSB with a '1' counter cell part performs a +1 operation.

- At the time CR'_0 produces a configuration request to CR_0, the count request from CL_0 is present.

For the first condition, the response time of the counter is shown in equation 5.5.

$$\begin{aligned} T_{ar_0} = & T_{Wo} + T_{n2} + T'_{Lo} + T_{ar} + \sum_{i=\lambda}^{N_T-1} (T'_{Lo} + T_{Lo})_i \\ & + T'_{ar} + \sum_{i=0}^{\lambda-1} (T'_{ar} + T_{ar})_i \end{aligned} \tag{5.5}$$

Where λ is the number of zero bits encountered before a bit with a '1', starting from the LSB.

For the second condition, the response time is the same as equation 5.4.

5.10.3 Response Time on *br* Channel

Assuming that by the time a cell part completes its computation for n on its *ar* channel, the zero operation on its *br* channel input is ready, then the response time on the *br* channel output of the counter after outputting n pulses (n is the count modulo) on its *ar* output can be said to be constant.

5.11 Power Consumption

In this section, power consumption for each cell part is expressed as a factor of cell part switching activity.

Let β_{ar} and γ_{ar} denote the power consumed by cell parts *CL* and *CR* respectively for a single switching operation on the *ar* channel. Also, let β_{br} and γ_{br} denote the power consumption of cell parts *CL* and *CR* respectively for a single switching operation on the *br* channel.

Power consumed by each cell part activity on both *ar* and *br* channels is shown in equations 5.6, 5.7, 5.8, 5.9.

On the *ar* channel,

$$P_{D_{CL_{ar}}} = \beta_{ar} \sum_{i=0}^{n-1} x_i \tag{5.6}$$

$$P_{D_{CR_{ar}}} = \gamma_{ar} \sum_{i=0}^{n-1} x_i + d_i \tag{5.7}$$

Where $x_i = 2(x_{i-1} + d_{n-1-i})$, $x_0 = 0$. The d term refers to the indexed bit value.

The frequency of transitions on the *ar* channel of each cell part, starting from the LSC, is at least double that of the next higher significant cell.

On the *br* channel,

$$P_{D_{CL_{br}}} = n \times \beta_{br} \tag{5.8}$$

$$P_{D_{CR_{br}}} = n \times \gamma_{br} \tag{5.9}$$

The dynamic power consumed by the counter is therefore the sum $P_{D_{CR_{ar}}} + P_{D_{CR_{ar}}} + P_{D_{CL_{br}}} + P_{D_{CR_{br}}}$.

5.12 Measurement Results

5.12.1 Post Layout Simulation

Table 5.10 shows the response time of the counter obtained from the post-layout simulation for counts 4, 8, 15, 16 and 31 at 3.3V. The difference noticed for even and odd numbers under the $ar \rightarrow br$ column can be explained by the different traces of activation for zero channel

operations (transition on *br*) in Figure 5.20 which may require a different combination of logic gates. The trace activated by $n21+$ and $dr+$ occurs only for odd numbers, while the trace activated by $n20+$ and $dr+$ occur for even numbers.

Table 5.10 Response Time

Count	**Response Time (ns)**	
	$Load \rightarrow ar$	$ar \rightarrow br$
4	28.80	5.17
8	31.06	5.17
15	32.30	5.07
16	30.97	5.17
31	32.71	5.07

5.12.2 Testing of Implemented Counter

A 5-bit counter was implemented in 350 nm CMOS technology using the AMS standard library. In synthesising the logic gates for the wrapper and the control and counter cell parts, technology mapping option was used in WorkCraft to ensure the logic gates were mapped as closely as possible to the available AMS 350 nm standard cells. Reset logic was also added manually to the synthesised logic circuits.

The layout of a 5-bit implemented counter is shown in Figure 5.39, while the die photograph of the fabricated counter is shown in Figure 5.40.

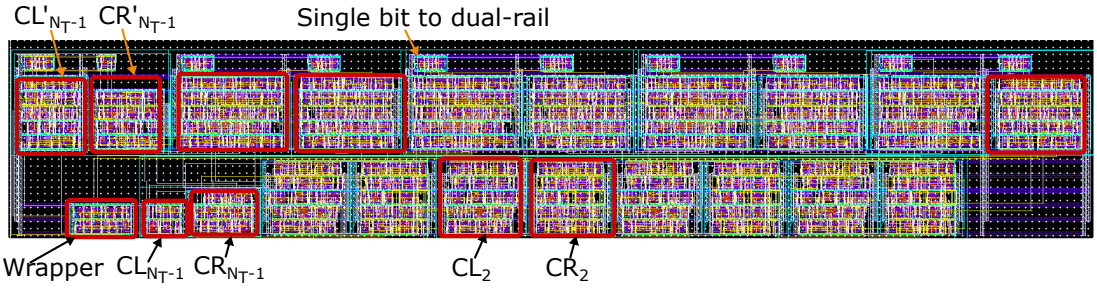


Fig. 5.39 Layout of the implemented 5-bits counter

Table 5.11 shows the measured areas of the distinct parts that make up each cell used in the implemented counter.

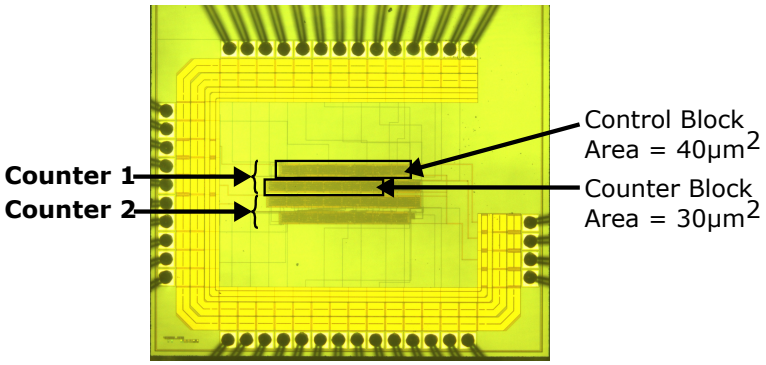


Fig. 5.40 Die photo of a 5-bits self-timed modulo$-n$ counter

Table 5.11 Measured Area Consumed by each Cell Part

Control Cell	**Area** (μm^2)
Wrapper	0.6
CL'_{n_s-1}	3.0
CR'_{n_s-1}	2.5
CL'_i	4.6
CR'_i	4.0
CR'_0	4.0
Counter Cell	**Area** (μm^2)
CL_{n_s-1}	0.72
CR_{n_s-1}	1.8
CL_i	3.5
CR_i	3.2

I configured an Altera FPGA to operate as the environment that interacts with the counter. The environment produced the count modulo n in binary as input to the counter, after which it issues a load request. The counter consequently produced a load complete output on its load channel, counting events on the *ar* channel and end of count event on the *br* channel. Interactions between the counter and the environment was done in a four-phase handshake. That is interactions which originated from the environment must be acknowledged by the counter and vise versa.

Acknowledgements to events on the *ar* and *br* channels of the counter were timed. The time was set at $1\mu s$ (ack timings for *ar* and *br* events in one test instance) and $5\mu s$ (ack timings for *ar* and *br* events in another test).

The counter was tested for seamless counting operation and seamless counting transitions.

For seamless counting operation, I provided a single count modulo after which the FPGA was enabled to produce a load request to the counter. The FPGA also monitored the load acknowledgement output of the counter and after a four phase handshake between Load request and a load acknowledgement, the FPGA automatically issued another load request Figure 5.41.

For seamless count transition, I provided a single count modulo after which the FPGA was enabled to produce a load request to the counter. After some time, I changed the count modulo *n* and re enabled the FPGA to input the new *n* as new count command to the counter Figure 5.42.

5.12.2.1 Continuous Counting and Seamless Loading Operations

Figure 5.41 shows the counter operating continuously for count modulo 16. Figure 5.42 shows seamless count transition from count modulo 3 to count modulo 15. A load command for new count modulo 15 arrived while a modulo 3 count operation was active. At the end of its modulo 3 active operation, the counter immediately reconfigured its operation for the newly loaded modulo 15 count sequence.

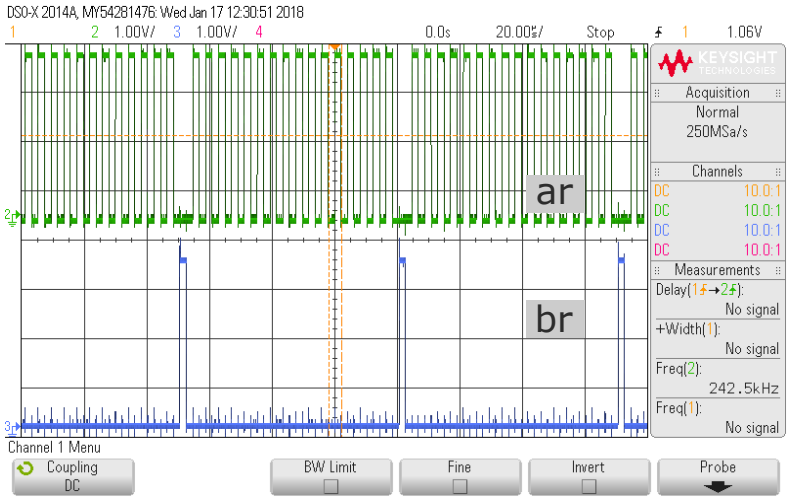


Fig. 5.41 Oscillogram of implemented counter output showing continuous count operation

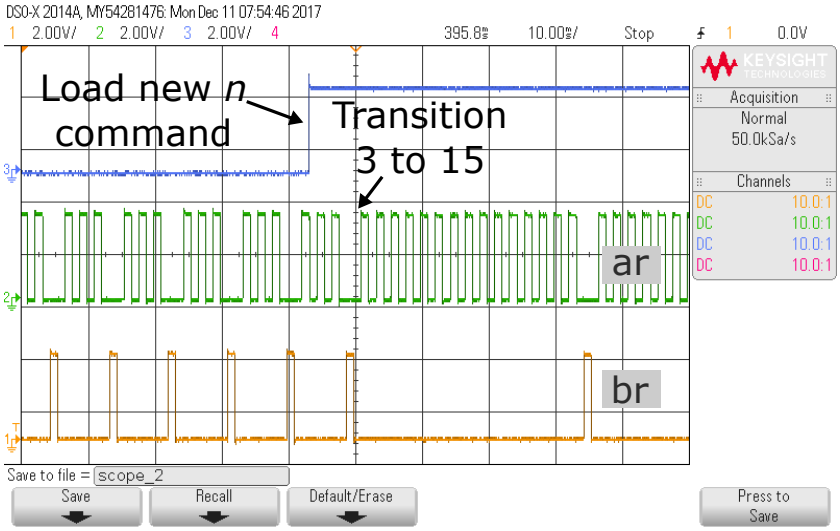


Fig. 5.42 Oscillogram of implemented counter output showing seamless count transition

5.12.2.2 Average Power Consumption

Figs. 5.43 and 5.44 show plots of the average power consumption of the counter for all thirty-one count sequences at supply voltage 2.1V, 2.4V, 2.7V, 3V, 3.3V for 1μs and 5μs acknowledgement timings respectively. The power consumption measured was for one counter excluding pads.

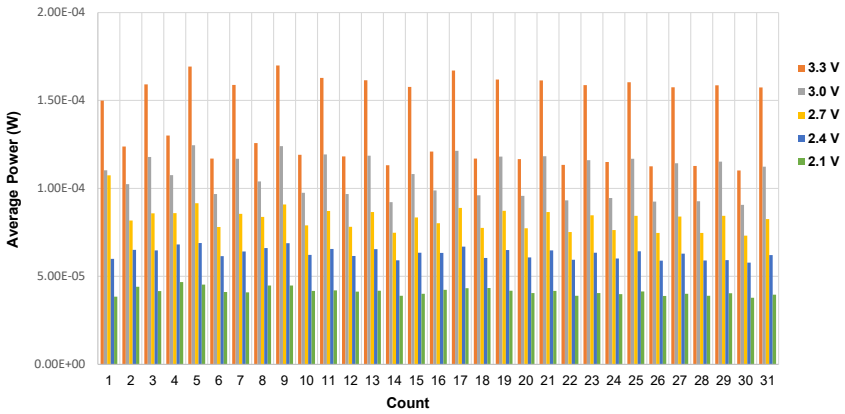


Fig. 5.43 Average Power at delay of 1μs to acknowledgement to *ar* and *br* events

For a given acknowledgement timing and count sequence, the counter operated at a fixed frequency for different voltages. For the two acknowledgement timings, the longer the delay,

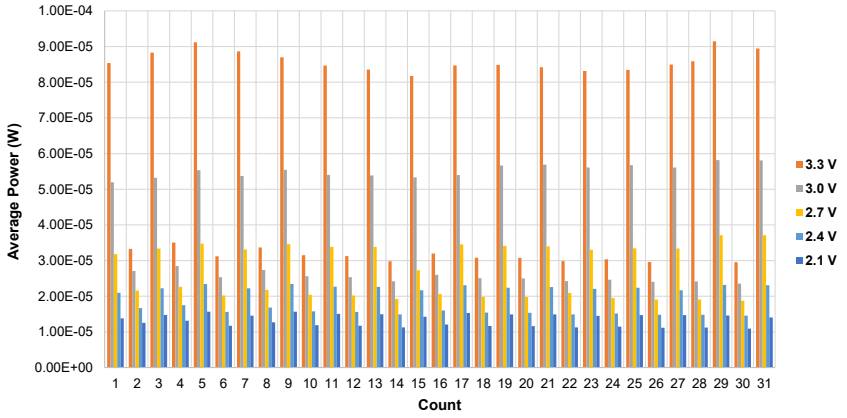


Fig. 5.44 Average Power at delay of 5μs to acknowledgement to *ar* and *br* events

the lower the average power consumed. At 3.3V the average power for counts 18, 31 are 117μW, 157μW and 30μW, 89μW @ 1μs and 5μs acknowledgement timing respectively.

5.13 Conclusion

In this chapter, decomposition, specification, verification and synthesis of a self-timed modulo$-n$ counter was presented. The loadable modulo$-n$ counter was decomposed from its general specification to a linear array of cells by rewriting the count n using Horner's method. A cell was decomposed into left and right cell parts, which does the even and odd operation respectively. The cell parts were specified in a high-level language using Labeled Petri Nets (LPN) from which unfolding of actions was used for verification of the decomposed counter operation and in a low-level language using Signal Transition Graphs (STG) derived from the LPNs. The counter was implemented in 350 nm CMOS Technology. The counter operates correctly over a wide range of voltages and can perform seamless counting for different count modulo. This type of counter can be employed in systems that require stable and predictable computations requiring end of count details like the digital pulse width modulator (DPWM).

Chapter 6

Digital Pulse-width Modulator

In this chapter, a digital pulse-width modulator (DPWM) with coarse and fine tune controls is investigated. The DPWM is realised by synchronising the operations of two of the loadable Kessels counter presented in chapter 5. Coarse control in the DPWM is achieved by tuning the count modulo of the counters. One counter sets the pulse width, while the other sets the pulse period. Fine tuning is achieved by the use of an addition based delay system that ensures that for a given coarse controls, a time quantity k added or subtracted to the ON time of the pulse is also subtracted or added to the OFF time of the pulse. This feature of the fine tune controls will ensure a constant pulse period when the pulse width is fine tuned. The mode of operation and interaction of the sub-systems that make up the presented DPWM illustrates a method of exploiting robustness in asynchronous circuits.

6.1 Introduction

Figure 6.1 is block diagram of the proposed DC-DC converter in which the DPWM realised in this chapter can be applied. It consists of the Digital Pulse-width Modulator (DPWM), Analogue-to-Digital Converter (ADC), digital controller and the output block which consists of the power transistor(s) and a passive filter.

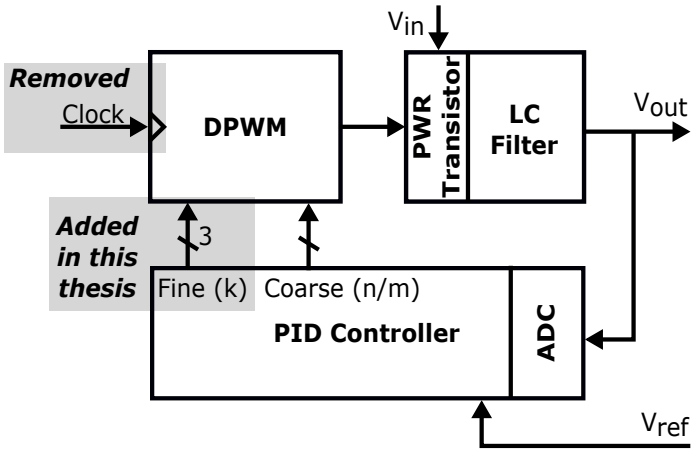


Fig. 6.1 Block Diagram of DC-DC Converter

The ADC converts the output voltage, V_{out}, to a digital value $V_{out}[n]$, a form suitable for the digital controller. The digital controller applies the appropriate correction control to the DPWM operation to correct any difference between the reference voltage $V_{ref}[n]$ and $V_{out}[n]$. The DPWM output switch ON and OFF the power transistors, which delivers an output current from the supply voltage V_{in} through the passive filter to the load.

Digital circuits offer complex computation and control techniques. However, in the DC-DC converter, quantisation can be introduced in the system from the coarse nature of the clock, the ADC and the DPWM resolutions [96–99] and this can result to a state called limit cycle oscillation (LCO) [11].

To control LCO, the ADC resolution is usually decreased so that it can effectively approximate small changes in output voltage level to a digital equivalent within the resolution of the pulse generator.

On the other hand, the DPWM resolution is increased to avoid or exit LCO [100–102]. This is because quantisation error from the DPWM can result in a mismatch between the smallest possible change and the ideal change required of the DPWM output to give the desired voltage output of the DC-DC converter. In this chapter I am interested in investigating a method by which DPWM.

Equation 6.1 expresses the magnitude of quantisation from the DPWM ΔV_{o_DPWM} as a function of the DPWM resolution ΔD [103].

$$\Delta V_{o\_DPWM} = V_{in} \times \Delta D \tag{6.1}$$

The term ΔV_{o_DPWM} is the change in the output voltage of the converter as a result of a change ΔD of the DPWM pulse width. From equation 6.1, the smaller ΔD is, the finer the resolution of the converter output.

Two major methods by which ΔD has been controlled are dithering [96] and by delay-line based DPWM[101]. In dithering method, the input clock frequency to the DPWM is varied such that the duty-cycle is randomized. This will result in an averaged output of the converter. Dithering method requires careful design to avoid idle tones. Delay-line based DPWM is a method by which incremental delays are used to tune DPWM resolution. However, in a synchronous system, the existing approach to delay-line based DPWM fine tuning may be non trivial as illustrated in the scenario described below.

At a given pulse-period, let the pulse-width required to exit LCO and output the desired converter voltage be given by equation 6.2.

$$\Delta V_{o\_ideal} = V_{in} * (\Delta D \pm k) \tag{6.2}$$

Where $\Delta V_{o\_ideal} = \Delta V_{o\_DPWM} \pm V_{o\_k}$, $V_{o\_k} = V_{in} * k$, .

For a DPWM with duty ratio T_N/T, the output frequency f is the reciprocal of the period T, where T_N is the ON time and T is the sum of T_N and OFF time (T_F).

If k is a tunable delay used to fine tune the DPWM output, implementing a synchronous system that satisfies (6.2) may be non-trivial as it may include among others, tuning the clock frequency and care must still be taken as the clock period can still introduce quantisation, and other design time margins must still be considered.

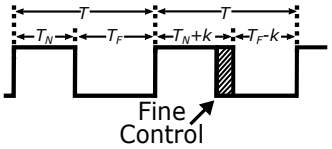


Fig. 6.2 Illustration of fine tune control on a DPWM output.

Since digital circuits offer complex computation and control mechanisms, the aim in this chapter is to eliminate the need for a clock, maintain a digital DPWM while meeting (6.2).

Introducing the incremental quantity k, the DPWM output can be expressed as $(T_N \pm k)/T$. Where $T = (T_N \pm k) + (T_F \mp k) = T_N + T_F$. Here the pulse period does not change, but the ON time of the pulse it tuned by k.

In this chapter, the self-timed loadable counter investigated in Chapter 5 is used in the design of a fine-tunable DPWM that satisfies equation 6.2. In the DPWM presented, an addition-based delay system is used to introduce and manipulate the quantity k in the DPWM system, from which fine-tuning of the DPWM output is achieved while maintaining the output frequency f.

6.2 Proposed Fine-Tunable DPWM

Figure 6.3 shows the block diagram of the proposed DPWM. It consists of two loadable asynchronous modulo$-n$ counters, a synchronisation block to synchronise the operation of the two counters, an addition based tunable delay system for fine tuning the pulse width and an output SR latch.

Counter 1 determines the ON time of the pulse, while counter 2 determines the pulse period. For correct operation of the DPWM, the count modulo of counter 1 must be less than that of counter 2.

The synchronisation block provides a common acknowledgement *aa* to count operations on the *ar* channels of the two loadable counters. However, since the count modulo of both

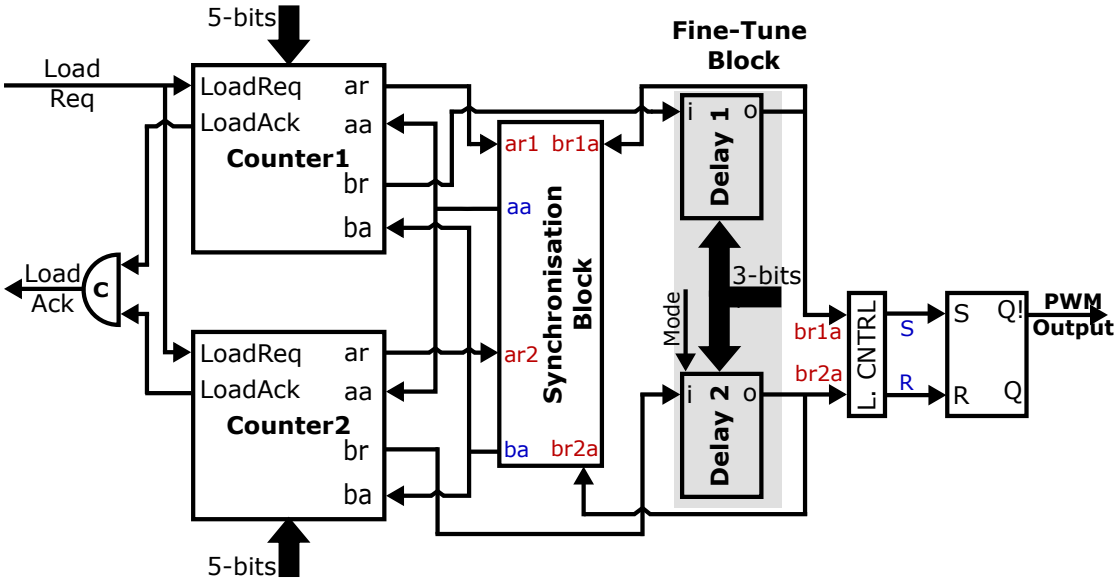


Fig. 6.3 Block diagram of proposed PWM

counters can be different, the synchronisation block specification guarantees the continuous count operation of the counter with a higher count modulo.

The synchronisation block also ensures that an end of count operation on the *br* channels of both counters have a common acknowledge signal *ba*. Therefore after the end of count transition on the *br* output of the counter with smaller count modulo, the other counter must also produce an end of count transition before an acknowledgement signal is sent from the synchronisation block to both counters.

The addition delay consists of two tunable delays: tunable delay 1 and tunable delay 2. Both delays are tuned by a common 3-bit binary input. The *br* outputs of counter 1 and counter 2 feed the inputs of tunable delay 1 and tunable delay 2 respectively. Each delay is made up of a series of gates and an 8-input multiplexer. In tunable delay 1, the series gates are divided into seven stages with outputs from each stage feeding inputs 1 to 7 of the mux, while a non-delayed signal is fed into input 0. Tunable delay 2 is identical to delay 1; however, the inputs to the MUX are transposed.

Therefore, the delay system operates such that whatever time delay k is added or subtracted to the *br* output of counter 1 to set the latch a time delay k is subtracted or added

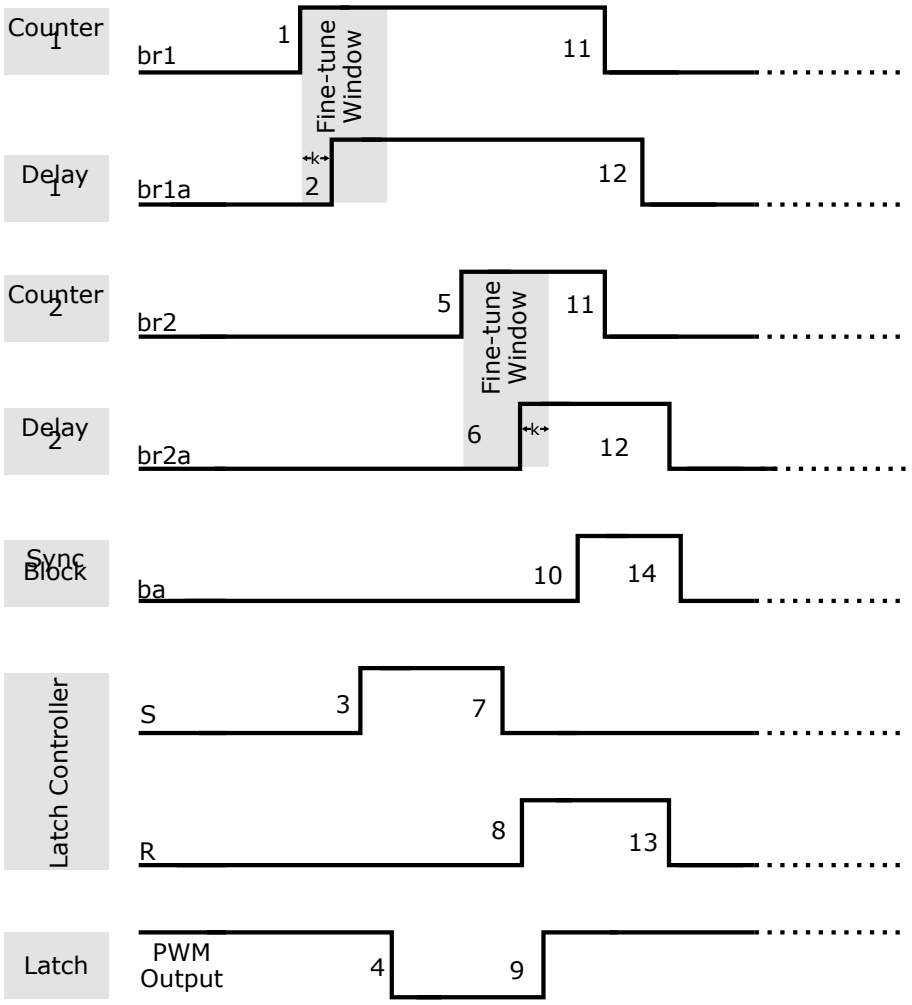


Fig. 6.4 Timing diagram of the proposed DPWM

to the *br* output of counter 2 to reset the latch. This keeps the pulse period constant. The concept of addition based control of a quantity has been presented in [104] where jitter in a ring oscillator was controlled by addition-based current source. However, in my application, the controlled quantity is the total fine tune delay which is maintained by an addition based delay system.

Figure 6.4 shows the sequence of transitions (numbered) of the the subsystems in the proposed DPWM. The labels by the side of each wave shows the source name of the signal. A typical operation of the wave form is explained thus, Counter 1 end of count transition

on its *br* channel sets the output latch (DPWM output) through transitions 1 (counter 1 br channel), 2 (tunable delay output), 3 (Latch controller set output) and 4 (Latch not Q output).

In this design, it is assumed that the time between $br1a$ and $br2a$ transitions is lager enough to ensure a stable change in the latch controller and the latch.

6.2.1 Synchronisation Block

The STG of Figure 6.5 describes the synchronisation block operation. Four traces are labelled based on the possible input combinations from counter 1 and counter 2. The choice pair inputs $ar1$ and $br1a$ are connected to *ar* and the tuned delay of *br* outputs from counter 1 respectively. The choice pair $ar2$ and $br2a$ are connected to *ar* and the tuned delay of *br* outputs from counter 2 respectively. The input *R* is connected to the reset output of the latch controller.

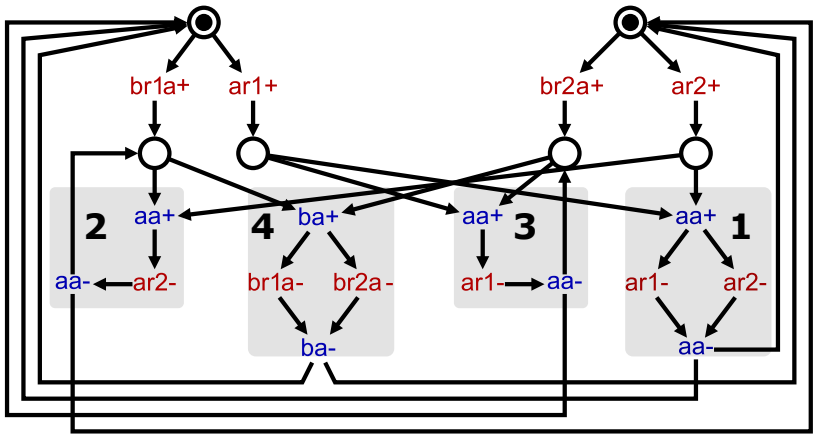


Fig. 6.5 STG of the Synchronisation Block

Trace 1 synchronises the count operation on the *ar* channels of both counters. It is activated when positive transitions occur in the *ar* channels of both counters 1 and 2. This provides an acknowledgement transition $aa+$ to both counters, which can only return low ($aa-$) after both $ar1-$ and $ar2-$ transitions occur, thus deactivating the trace.

Trace 2 ensures that counter 2 operation continues when counter 1 operation ends, in a situation where the count modulo of counter 1 is less than that of counter 2. This trace

is activated by $br1a$ which is the delayed transition of $br1$ and $ar2$. This trace continues to produce an acknowledgement $aa+$ to transitions on the ar channel of counter 2. The synchroniser can only exit to trace 4 when counter 2 completes its count operation.

Trace 3 ensures that counter 1 operation continues when counter 2 operation ends, in a situation where the count modulo of counter 2 is less than that of counter 1. This trace is activated by $br2a$ which is the delayed transition of $br2$ and $ar1$. This trace continues to produce an acknowledgement $aa+$ to transitions on the ar channel of counter 1. The synchroniser can only exit to trace 4 when counter 1 completes its count operation. If the constraint is that count modulo of counter 1 will always be less than that of counter 2, then trace 3 will never be activated. It was included to ensure the system never goes into a deadlock if this constraint is not observed.

Trace 4, synchronises the count operation on the br channels of both counters. It is activated when positive transitions occur in the br channels of both counters 1 and 2. This provides an acknowledgement transition $ba+$ to both counters, which can only return low ($ba-$) after both $br1a-$ and $br2a-$ transitions occur, thus deactivating the trace. In this trace, $ba+$ occurs only after the output latch controller has set and reset the output latch, that is the S and R outputs of the controller have gone through the transition sequence $S+$, $S-$, $R+$ and $R-$.

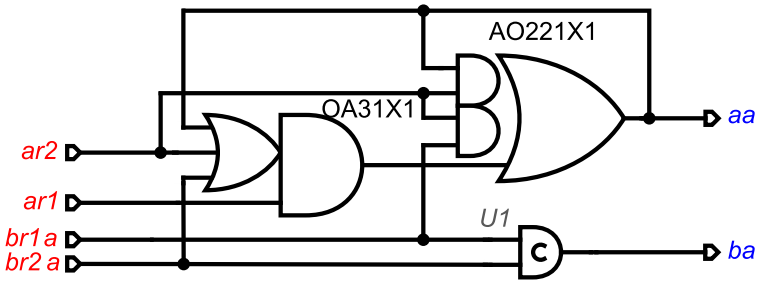


Fig. 6.6 Synthesised circuit of the Synchronisation Block

Figure 6.6 is the synthesised circuit of the synchronisation block.

6.2.2 Fine Tunable Delay Block

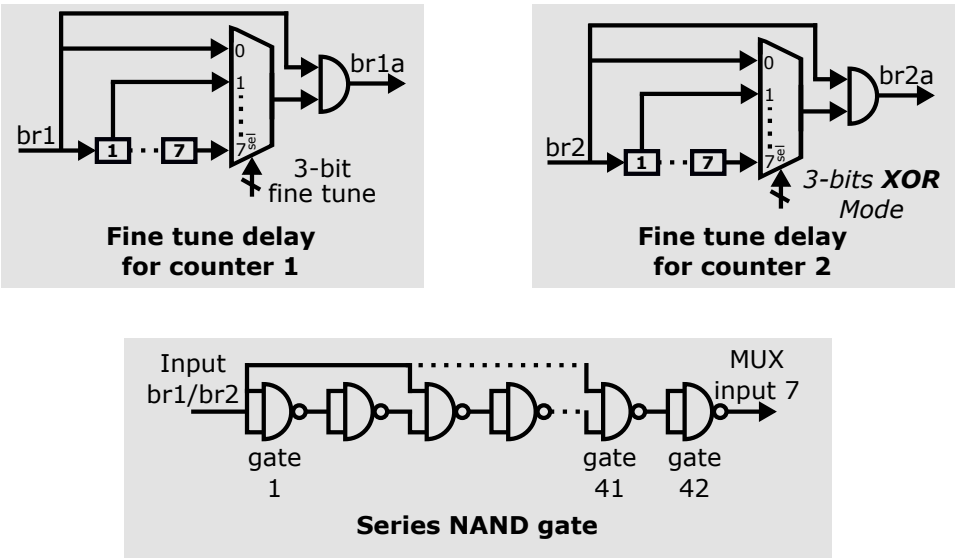


Fig. 6.7 Circuit of Fine tunable delay block

Figure 6.7 shows the tunable delay system used to time-shift the end of count indication from both counters to the output latch control block. It consists of two fine tunable delays. Each input to the fine tunable delays are fed by *br*1 and *br*2 from counter 1 (shown on the left) and counter 2 (shown on the right) respectively. Each tunable delay has an eight input multiplexer (MUX) and a 42-series-NAND-gates delay line divided into seven stages of six gates each. Inputs one to seven of the MUX are fed from outputs of each stage of the delay line, while input zero of the MUX is fed from the input to the delay line.

The input to the delay line (br1/br2) is connected to the output AND gate so that a transition to zero by-passes the series NAND delay and the MUX with the assumption that before the next transition to a '1' on the input of the delay (br1/br2) occurs, the previous transition to zero would have propagated through the output of the multiplexer. To shorten this propagation, one input to all odd numbered NAND gates in the series delay are connected to the input of the delay line (shown in the bottom block of Figure 6.7). Therefore the assumption of the propagation before the next transition to a '1' is reduced to a single NAND gate delay plus the MUX delay.

To realise an addition based delay system in the fine-tune block, one of the tunable delay system (that of Counter 2) can be made to function as a complement of the other delay system (that of Counter 1) with the use of three XOR gates and a common mode select to one input of each XOR gate. When the mode select is '0', the output of each XOR gate is the same as its fine tune input. In this case, both delay systems perform the same time delay function. When the mode select is '1', the output of each XOR gate is the negation of its fine tune input. In this case, both delay systems complement each other. This is explained in the example below.

Let the input select code to both multiplexers in Tunable delay 1 and 2 be "101". Let the multiplexer and output AND gate delay be denoted by T_{muxand} in both delay systems.

When mode = '0' the multiplexer for tunable delay 2 selects from input 5. In this case, the total delay to send a set and reset signal to the output latch control is $2\times$ ($T_{muxand} + 5$ delay segments).

When mode = '1' the multiplexer for tunable delay 2 selects from input 2. In this case, the total delay to send a set and reset signal to the output latch control is $2 \times T_{muxand} + 7$ delay segments.

Still in mode = '1', let the input select code to both multiplexers be changed to "100". In this case, delay 1 selects input 4 while delay 2 selects input 3 to their respective MUXs. The the total delay to send a set and reset signal to the output latch control still remains $2 \times T_{muxand} + 7$ delay segments.

In mode 1, the total delay is always constant. This keeps the pulse period constant while fine-tuning the pulse width.

6.2.3 Output Block (Latch Control and Latch)

The output block consists of the latch control block and an SR latch (pair of cross-coupled NOR gates). The STG specification of the latch control block is shown in Figure 6.8. This

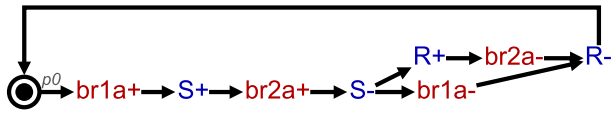


Fig. 6.8 STG of the Latch Control

block ensures that end of count indication from Counter 1 and Counter 2 set and reset the latch. The specification and circuit prevent the output condition S = '1' and R = '1' from occurring at the same time. This condition forces the outputs of the latch to '0's.

In the STG, transition $br1a+$ results in transition $S+$, which sets the latch. The specification states that transition $br2a+$ is ineffective until the controller withdraws the set command ($S-$) after which the latch is reset by output transition $R+$.

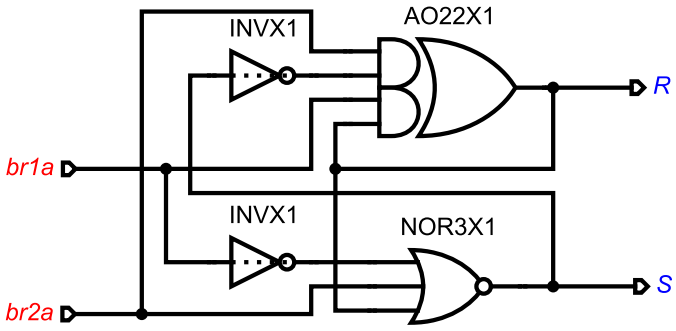


Fig. 6.9 Synthesised Circuit of the Latch Control

Figure 6.9 is the synthesised circuit of the latch controller. It operates as specified in Figure 6.8.

The ideal operation of this circuit in the DPWM system is that it responds to changes in its $br1a$ or $br2a$ inputs fast enough to set the latch before the next input.

6.3 Measurement Results

The realised DPWM was edited in Cadence viruoso environment and implemented using AMS 350 nm CMOS technology library. Post-layout simulation was carried out for the designing in the Cadence environment.

Table 6.1 shows the DPWM output when tuned with only coarse controls (that is when mode input to Delay 2 is '0' and the fine-tune input controls to both Delay 1 and Delay 2 are all zeros). The table compares the ratio of the DPWM output to the ideal ratio.

The coarse column of the table shows the input count modulo to counters 1 and 2 in the relationship counter 1/counter 2. ON, OFF column shows the measured ON, OFF times in a pulse period. The period column is the sum of the ON and OFF times. The ratio column gives the ratio of ON time of a pulse to the pulse period. Ideal Ratio column shows the mathematically expected ratio of ON time of a pulse to the pulse period for the given coarse control combination.

Table 6.1 DPWM Output Ratio with Fine-Tune Mode Off

Coarse	**ON, OFF Time (*n*s)**	**Period (*n*s)**	**Ratio**	**Ideal Ratio**	**% deviation**
3/11	20.82, 50.26	71.08	0.29	0.27	7
10/17	57.84, 45.19	103.03	0.56	0.59	5
4/8	23.1, 26.97	50.07	0.46	0.5	8
21/31	116.3, 61.04	177.34	0.66	0.68	3
6/31	36.58, 142.7	179.28	0.20	0.19	5
Avg. Pow. 2mW to 3mW					

From table 6.1, there is a maximum of 8% deviation from the expected output shown in the % deviation column. The table does not cover all possible combinations of counter 1 and counter 2. The deviation is a result of the response time difference between both counters on their *ar* channels and the response time of the synchronisation block for each *br* input.

Tables 6.2, 6.3, 6.4 and 6.5 show the DPWM output ratio with fine tune mode selected for the coarse control ratio 3/11, 10/17, 4/8 and 21/31 each at 1.8V, 2.3V, 2.8V and 3.3V respectively. The fine column in each table shows the input fine tune codes to the addition based delay system in the DPWM.

Table 6.2 Coarse Control Ratio 3/11

(a) @ 3.3V

Fine	**Up, Down Time (*n*s)**	**Period (*n*s)**	**Ratio**
0	22.42, 45.95	68.37	0.33
1	21.66, 46.83	68.49	0.32
2	20.79, 47.70	68.49	0.3
3	20.00, 48.49	68.49	0.29
4	19.07, 49.43	68.50	0.28
5	18.28, 50.22	68.50	0.27
6	17.41, 51.09	68.50	0.25
7	16.54, 51.84	68.38	0.24
Avg. Pow. 0.48mW to 0.5mW			

(b) @ 2.8V

Fine	**Up, Down Time (*n*s)**	**Period (*n*s)**	**Ratio**
0	26.46, 54.35	80.81	0.33
1	25.60, 55.33	80.93	0.32
2	24.57, 56.36	80.93	0.3
3	23.63, 57.31	80.94	0.29
4	22.53, 58.42	80.95	0.28
5	21.58, 59.36	80.94	0.27
6	20.55, 60.39	80.94	0.25
7	19.53, 61.29	80.92	0.24
Avg. Pow. 1mW to 1.9mW			

(c) @ 2.3V

Fine	**Up, Down Time (*n*s)**	**Period (*n*s)**	**Ratio**
0	33.76, 69.43	103.19	0.33
1	32.64, 70.67	103.31	0.32
2	31.34, 71.99	103.33	0.3
3	30.13, 73.21	103.34	0.29
4	28.76,74.65	103.41	0.28
5	27.52, 75.82	103.34	0.27
6	26.20, 77.13	103.33	0.25
7	24.91, 78.29	103.20	0.24
Avg. Pow. 0.82mW to 1.03mW			

(d) @ 1.8V

Fine	**Up, Down Time (*n*s)**	**Period (*n*s)**	**Ratio**
0	50.05, 102.50	152.55	0.33
1	48.16, 104.40	152.56	0.32
2	46.29, 106.30	152.59	0.3
3	44.51, 108.10	152.61	0.29
4	42.53, 110.10	152.63	0.28
5	40.75, 111.80	152.55	0.27
6	38.85, 113.70	152.55	0.25
7	36.98, 115.50	152.48	0.24
Avg. Pow. 0.48mW to 0.5mW			

Table 6.3 Coarse Control 10/17

(a) @ 3.3V

Fine	Up, Down Time (*n*s)	Period (*n*s)	Ratio
0	59.39, 40.90	100.29	0.59
1	58.62, 41.77	100.39	0.58
2	57.75, 42.64	100.39	0.58
3	57.02, 43.46	100.48	0.57
4	56.08, 44.40	100.48	0.56
5	55.23, 45.16	100.39	0.55
6	54.36, 46.04	100.40	0.54
7	53.49, 46.81	100.30	0.53
Avg. Pow. 2.04mW to 2.68mW			

(b) @ 2.8V

Fine	Up, Down Time (*n*s)	Period (*n*s)	Ratio
0	70.19, 48.29	118.48	0.59
1	69.27, 49.31	118.58	0.58
2	68.24, 50.35	118.59	0.58
3	67.37, 51.33	118.70	0.57
4	66.26, 52.43	118.69	0.56
5	65.25, 53.34	118.59	0.55
6	64.21, 54.39	118.60	0.54
7	63.19, 55.30	118.49	0.53
Avg. Pow. 1.2mW to 1.6mW			

(c) @ 2.3V

Fine	Up, Down Time (*n*s)	Period (*n*s)	Ratio
0	89.63, 61.62	151.25	0.59
1	88.46, 62.91	151.37	0.58
2	87.14, 64.23	151.37	0.58
3	85.93, 65.45	151.38	0.57
4	84.54, 66.84	151.38	0.56
5	83.32, 68.06	151.38	0.55
6	82.00, 69.39	151.39	0.54
7	80.70, 70.56	151.26	0.53
Avg. Pow. 0.7mW to 0.9mW			

(d) @ 1.8V

Fine	Up, Down Time (*n*s)	Period (*n*s)	Ratio
0	132.60, 90.90	223.50	0.59
1	130.90, 92.75	223.65	0.58
2	129.00, 94.65	223.65	0.58
3	127.20, 96.44	223.64	0.57
4	125.20, 98.42	223.62	0.56
5	123.40, 100.20	223.60	0.55
6	121.50, 102.10	223.60	0.54
7	119.60, 103.80	223.40	0.53
Avg. Pow. 0.32mW to 0.4mW			

Table 6.4 Coarse Control 4/8

(a) @ 3.3V

Fine	**Up, Down Time (*n*s)**	**Period (*n*s)**	**Ratio**
0	24.62, 22.71	47.33	0.52
1	23.95, 23.58	47.53	0.50
2	23.09, 24.45	47.54	0.49
3	22.30, 25.24	47.54	0.47
4	21.36, 26.18	47.54	0.45
5	20.58, 26.97	47.55	0.43
6	19.70, 27.84	47.54	0.41
7	18.83, 28.95	47.78	0.39
Avg. Pow. 2.4mW to 3.8mW			

(b) @ 2.8V

Fine	**Up, Down Time (*n*s)**	**Period (*n*s)**	**Ratio**
0	29.10, 26.85	55.95	0.52
1	28.31, 27.88	56.19	0.50
2	27.29, 28.91	56.20	0.49
3	26.35, 29.86	56.21	0.47
4	25.25, 30.96	56.21	0.45
5	24.30, 31.90	56.20	0.43
6	23.27, 32.93	56.20	0.41
7	22.24, 34.28	56.52	0.39
Avg. Pow. 1.3mW to 2mW			

(c) @ 2.3V

Fine	**Up, Down Time (*n*s)**	**Period (*n*s)**	**Ratio**
0	37.13, 34.35	71.48	0.52
1	36.12, 35.65	71.77	0.50
2	34.83, 36.96	71.79	0.49
3	33.62, 38.18	71.80	0.47
4	32.23, 39.57	71.80	0.45
5	31.01, 40.79	71.80	0.43
6	29.70, 42.10	71.80	0.41
7	28.40, 43.85	72.25	0.39
Avg. Pow. 0.8mW to 1.2mW			

(d) @ 1.8V

Fine	**Up, Down Time (*n*s)**	**Period (*n*s)**	**Ratio**
0	54.79, 50.74	105.53	0.52
1	53.35, 52.63	105.98	0.50
2	51.49, 54.54	106.03	0.49
3	49.72, 56.31	106.03	0.47
4	47.74, 58.29	106.03	0.45
5	45.96, 60.07	106.03	0.43
6	44.06, 61.96	106.02	0.42
7	42.16, 64.57	106.73	0.40
Avg. Pow. 0.3mW to 0.5mW			

Table 6.5 Coarse Control 21/31

(a) @ 3.3V

Fine	**Up, Down Time (*n*s)**	**Period (*n*s)**	**Ratio**
0	118.00, 56.77	174.77	0.68
1	117.10, 57.60	174.70	0.67
2	116.30, 58.52	174.82	0.67
3	115.60, 59.31	174.91	0.66
4	114.60, 60.25	174.85	0.66
5	113.80, 61.04	174.84	0.65
6	112.80, 61.86	174.66	0.65
7	112.10, 62.65	174.75	0.64
Avg. Pow. 2mW to 2.7mW			

(b) @ 2.8V

Fine	**Up, Down Time (*n*s)**	**Period (*n*s)**	**Ratio**
0	139.20, 67.03	206.23	0.67
1	138.50, 68.10	206.60	0.67
2	137.30, 69.09	206.39	0.67
3	136.50, 70.09	206.59	0.66
4	135.40, 71.19	206.59	0.66
5	134.50, 72.14	206.64	0.65
6	133.40, 73.17	206.57	0.65
7	132.40, 74.07	206.47	0.64
Avg. Pow. 1.25mW to 1.5mW			

(c) @ 2.3V

Fine	**Up, Down Time (*n*s)**	**Period (*n*s)**	**Ratio**
0	177.90, 85.82	263.72	0.67
1	176.80, 86.97	263.77	0.67
2	175.40, 88.24	263.64	0.67
3	174.30, 89.51	263.81	0.66
4	172.90, 90.91	263.81	0.66
5	171.70, 92.13	263.83	0.65
6	170.40, 93.44	263.84	0.65
7	169.10, 94.60	263.70	0.64
Avg. Pow. 0.6mW to 0.8mW			

(d) @ 1.8V

Fine	**Up, Down Time (*n*s)**	**Period (*n*s)**	**Ratio**
0	262.70, 127.30	390.00	0.67
1	261.50, 128.50	390.00	0.67
2	259.70, 130.40	390.10	0.67
3	257.90, 132.20	390.10	0.66
4	255.90, 134.20	390.10	0.66
5	254.10, 136.00	390.10	0.65
6	252.10, 137.80	389.90	0.65
7	250.30, 139.60	389.90	0.64
Avg. Pow. 0.28mW to 0.3mW			

The general characteristics of the DPWM in each table are discussed below.

Constant Pulse Ratio This property of the DPWM is true when it operates with or without fine tune mode selected. In tables 6.2, 6.3, 6.4 and 6.5, at a given coarse and fine tune code, the pulse ratio is constant when the operating voltage is changed. For example in table 6.2 at fine-tune control equal to 3, the pulse ratio is 0.57 across all voltage levels for tables 6.2 (a), 6.2 (b), 6.2 (c) and 6.2 (d).

Constant Pulse Period In tables 6.2, 6.3, 6.4 and 6.5, at a given pulse ratio and operating voltage, the pulse period is 'almost' constant irrespective of the fine tune block control, this is because the fine tune block is an addition-based system.

Fine-Tune Range In tables 6.2, 6.3, 6.4 and 6.5, at a given coarse control input and voltage supply, the range of pulse ratios, considering the deviation ('almost') as a result of fine tune inputs includes the ideal ratio expected from the coarse tuning shown in in Table 6.1.

Average power Consumption Average power consumption of the DPWM is in the range 0.4mW to 3mW at 3.3V and 0.3mW to 0.5mW at 1.8V as shown in Tables 6.2, 6.3, 6.4 and 6.5.

The 'almost' constant period for a given coarse control and operating voltage can be expressed as error in the system given by $T \pm e_T = (T_N \pm k \pm e_N) + (T_F \mp k \pm e_F) = T_N + T_F \pm e_T$. Where $e_T = \pm e_N \pm e_F$ is the error introduced in the system from both fine tune delay paths which may vary for different fine tune input. Therefore, the pulse ratio can be expressed as $(T_N \pm k \pm e_N)/T \pm e_T$.

6.4 Conclusion

In this chapter, a DPWM with coarse and fine-tune controls was investigated. Coarse control was achieved by the use of two modulo$-n$ counters to determine the pulse ratio and pulse period. Fine-tune control was achieved by the use of an addition based delay system that ensured the pulse period remained constant while fine-tuning the pulse width. The circuit realised was entirely asynchronous.

The investigated DPWM in this chapter demonstrated a method of exploiting robustness property in asynchronous circuits. The DPWM can operate correctly in a wide range of voltage supply because ordering of events which is entirely asynchronous ensures that circuit gates delay (depends on the delay assumptions used) does not affect the correct computation of the loadable modulo counters used. Inheriting from the robustness property, the DPWM was also shown to operate predictably when the operating voltage is changed. It can provide a constant pulse ratio over at different voltage levels. This property makes the DPWM suitable in applications where the power supply may change. In a situation where there is a change in power supply, the pulse period may change for a given coarse and fine tune control inputs, how ever the pulse ratio still remains constant. This property has useful applications in energy modulated computing systems.

Chapter 7

Conclusion

Asynchronous circuit design are aided by a number of available of CAD tools and languages to ease specification, verification and synthesis of asynchronous circuits.

However, design and specification of asynchronous circuits is not as straight forward as synchronous circuits because asynchronous circuit designers must adhere to strict timing and ordering sequence of gate switching.

In this thesis, the asynchronous circuits were specified using signal transition graphs, from which their logic circuits were synthesized using the WorkCraft tool. The issues a designer may encounter through this process include: the need to resolve complete state coding (CSC) errors and the need to map of the synthesized equations to the standard cells of the target library for simulation and CMOS implementation.

While WorkCraft provides an option to automatically resolve CSC issues by adding internal signals, the tool tends to add more signals than a human would if done carefully. Manually, a simple STG may take hours to resolve, however, WorkCraft provides an interface to visualise the cores of the encoding conflicts and then signals can be inserted in areas where most cores overlap. More on this process can be found on the WorkCraft website.

Some synthesised equations may not be mappable to standard cells in a target library. At this point, the designer will have to manually optimize the logic equation to use more complex gates.

I have chosen asynchronous design methods because asynchronous circuits offer some advantages when compared to synchronous circuits. One of the major advantage is asynchronous circuits tolerance to variations known as robustness.

In this work, it is shown that robustness property in asynchronous systems and circuits can be actively exploited to design systems with coarse and fine-tunable controls with a guarantee that time margin requirements of the logic gates do not result in system failure when operated in extreme conditions.

This thesis has presented two approaches to exploiting robustness property in asynchronous circuits these are the design of an adaptive system whose Digitally controlled oscillator (DCO) operation is modulated by the triggered computing unit and a DPWM with coarse and fine-tune controls based on a loadable self-timed modulo$-n$ counter.

7.1 Conclusion

Chapter 4 demonstrated the first approach to exploiting robustness, using a self-timed computing unit and a DCO which were integrated as a unit such that the operation of the computing unit modulated the DCO operation. This approach ensured that due to the robust nature of the computing unit, while its task completion time may vary in the presence of PVT variation, hazardous conditions are avoided because the clock signal is sensitive to the operating conditions of the computing unit. The circuits presented were analysed, modelled and characterised using mathematical equations. The computing unit investigated is a nine bits self-timed sequential counter whose time to complete the next computation task varied and depended on its present state. This computing unit was chosen because its correct operation is not affected by PVT variations. However, its time to complete a

computation can be affected by PVT variations. The time to complete a computation is used to modulate the DCO operation. The DCO produces a computation request and must receive an acknowledgement in a four-phase handshake sequence before it can produce another computation request. The asynchronous circuits and controllers were specified using STGs from which the logic circuits were synthesised and mapped to AMS 350 standard cell library using the WorkCraft Tool.

The system presented in Chapter 4 shows that simple computation islands can be optimised in this way by designing them as self-timed circuits that modulates its interaction with the clock. The robustness of the system presented was evaluated by varying the operating voltage. The system was shown to operate correctly over a wide range of voltage supply. This is a an advantage as the operating voltage of a circuit affects the logic gates delay, and set-up and hold times.

Chapter 6 demonstrated the design of Digital Pulse-width Modulator (DPWM) with coarse and fine-tune controls. The DPWM contained two loadable self-timed modulo$-n$ counters whose operations were synchronised to provide a coarse control to tune the pulse width and pulse period. The DPWM also contained an addition based delay system to fine-tune the pulse width while keeping the pulse period constant. The DPWM operation was verified by post layout simulation in Cadence.

The loadable self-timed modulo$-n$ counter was presented in Chapter 5. It was decomposed into cells (further decomposed into cell parts) using Horner's method, specified and implemented using formal asynchronous design techniques in two levels; high-level specification using Labelled Petri nets (LPN) and low-level specification using Signal Transition Graphs (STGs). Starting with a general description, properties of the counter was formally expressed in such a way as to explicitly express internal interactions of independent units. This made it easy to identify areas that can be manipulated to achieve a loadable counter. LPN specifies the system using signal names of each cell to represent actions. In this form,

the system operation can easily be modelled and verified. Also in this form, channels of interaction between each cell and cell parts are encoded in either single or dual-rail depending on the type of data to be conveyed in a request channel. In low-level form, the LPN is translated to STGs from which complete state coding issues are resolved.

The logic gates of each cell and sub-cells are synthesised and mapped to standard gates in AMS 350nm library from their STGs using the WorkCraft Tools. The synthesised logic equation in Verilog was imported into Cadence environment after which the complete counter was integrated from the imported cell parts. The integrated counter was simulated in Cadence environment using Virtuoso to verify that it operates correctly. The layout of the counter was designed, and the counter was fabricated in 350 nm technology. Response time and measurement results of the implemented counter were also presented. The counter offered a bounded response time on its interaction with the environment on the load and count channels such that it is possible to perform continuous and seamless count transitions. The response time of the counter was not dependent on the loaded count modulo n.

The addition based delay was used to fine tune the DPWM output at a given pulse ratio. A quantity of delay added to the ON time is subtracted from the OFF time. This circuit structure was achieved with the use of two multiplexer based delays whose input delay length were transposed.

Two issues arose in the designed DPWM, the first is that the DPWM pulse ratio was not exactly the same as the mathematically ideal ratio of the input coarse controls when fine tune is turned off or when fine tune controls is zero (same quantity of delay is added to the ON and OFF times of the pulse). This ordinarily would present a problem for the proposed system of application (DC-DC converter) of this type of DPWM. However, the fine tune controls included in the design can provide a range of pulse ratios above and below the ideal ratio. The second issue is that the pulse period slightly varied with different fine tune control

inputs (theoretically, it is supposed to be constant because of the addition based delay) at a given coarse control inputs.

7.2 Future Work

The first part of this research presented a simple computing unit to illustrate the concept of exploiting robustness to design adaptable clock-computing unit interactions. A more complex synchronous circuit can be specified and designed using formal asynchronous techniques, and the operation of the system investigated to determine the reliability of this approach.

The second part of this research presented the decomposition and application of a loadable self-timed modulo$-n$ counter in the design of a DPWM. The operation of the counter did not give a uniform response time on the *ar* channel output, however it was bounded. Further work can be done in the specification of the cell parts to improve parallel operations or by strategic addition of handshake decoupling circuits to provide a uniform response time on the output of the counter. This will ultimately give the DPWM a well defined and calculated pulse output based on the input controls.

The Control block of the counter consumed much more area than the counter block, leakage power s an issue here because the control block is usually idle as it has to wait for a count operation to complete before it can issue a new configuration control to the counter cells. More efficient specification and control block decomposition can be used to reduce the area consumed. Also, the inclusion of a suitable configuration holding circuit of smaller area which would latch the configuration data while the control block is power gated is an option to be explored in minimising leakage power losses. Configuration output to the counter block occurs serially, starting from the most significant cell through to the least significant cell. This complexity adds to the delay in the counter response time. A more efficient decomposition and specification method of the control block can be used to avoid this delay.

An important direction for research in the DPWM is to replace the addition delay fine-tune circuit with an analogue addition delay fine-tune circuit for more fine control to the output pulse.

The relationship between the loadable counter operation and its operating voltage can be investigated as this presents a potential application in a constant pulse ratio DPWM over a wide range of voltage supply at a given count modulo.

Appendix A

Verilog Code For synthesised Counter Circuits

This appendix presents the synthesised Verilog code for each counter and control cell parts, mapped to AMS 350nm standard library.

This appendix is in two Sections; the control block under which the wrapper, control cells left and right parts verilog codes are shown and the counter block under which the counter cell left and right parts verilog codes are shown.

A.1 Control Cell Parts

A.1.1 Synthesised Verilog Specification for the Single Bit to Dual-Rail Converter

```
module DualRail ( d0, d1, d, da, rst );


output  d0, d1;
input  d, da, rst;
```

```
INVX1  I2 ( .Q(net11), .A(da));
INVX1  I1 ( .Q(net12), .A(d));
INVX1  I12 ( .Q(reset), .A(rst));
AND3X1  I0 ( .C(net11), .Q(d0), .A(reset), .B(net12));
AND3X1  I11 ( .C(reset), .A(net11), .B(d), .Q(d1));

endmodule
```

A.1.2 Synthesised Verilog Specification for the Wrapper

```
module wrapper(Lo, Lia, Wia, Li, Wi, Loa);
    output Lo, Lia, Wia;
    input Li, Wi, Loa;

    // [Lo] = (Lo | Wi) & ~int2
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE1(.Q(w0), .A(int2));
    OA21X1 U2(.Q(Lo), .A1(Wi), .A2(Lo), .B1(w0));

    // [Lia] = (Wia | int1) & Li
    OA21X1 U3(.Q(Lia), .A1(Wia), .A2(int1), .B1(Li));

    // [Wia] = Wia & int2 | int1
    AO21X1 U4(.Q(Wia), .A1(Wia), .A2(int2), .B1(int1));

    // [int1] = (Li | int1) & (Lo | Loa | Wi)
```

```
    OA32X1 U5(.Q(int1), .A1(Wi), .A2(Loa), .A3(Lo), .B1(Li), .B2(int1));

    // [int2] = (Li | Wi | int1) & int2 | Loa
    OA32X1 U6(.Q(int2), .A1(Li), .A2(Wi), .A3(int1), .B1(Loa), .B2(int2));

    // initial values of the signals
    // !Li !Wi !Loa !Lo !Lia !Wia !int1 !int2 w0
endmodule
```

A.1.3 Synthesised Verilog Specification for Cell Part CL'_{N_T-1}

```
module msc_control_left_cell(Lia,Lo0,Lo1,dLa,n10,d0,d1,Li1,Loa,n1a);
    output Lia, Lo0, Lo1, dLa, n10;
    input d0, d1, Li1, Loa, n1a;

    // [Lia] = Li1 & Lia | Lo0 | Lo1
    AO211X1 U0(.Q(Lia), .A1(Li1), .A2(Lia), .B1(Lo0), .C1(Lo1));

    // [Lo0] = (~Loa | ~deco5) & Lo0 | ~deco1
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE2(.Q(w1), .A(Lo0));
    AOI32X1 U3(.Q(Lo0), .A1(Loa), .A2(deco1), .A3(deco5), .B1(w1), .B2(deco1));

    // [Lo1] = (~Loa | ~deco3 | ~deco5) & Lo1 | ~deco
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE5(.Q(w4), .A(Loa));
```

```
// #PRAGMA: zero delay
INVX1 IN_BUBBLE7(.Q(w6), .A(Lo1));
AOI32X1 U9(.Q(w8), .A1(deco5), .A2(deco), .A3(deco3), .B1(w6), .B2(deco));
C2 U10(.Q(Lo1), .A(w4), .B(w8));

// [dLa] = (d0 | d1) & dLa | Lo0 | Lo1
AOI21X1 U12(.Q(w11), .A1(dLa), .A2(d0), .B1(Lo0));
AOI21X1 U14(.Q(w13), .A1(dLa), .A2(d1), .B1(Lo1));
NAND2X1 U15(.Q(dLa), .A(w11), .B(w13));

// [n10] = Lo1 & int1 | n10 & ~n1a
// #PRAGMA: zero delay
INVX1 IN_BUBBLE17(.Q(w16), .A(n1a));
AO22X1 U18(.Q(n10), .A1(w16), .A2(n10), .B1(Lo1), .B2(int1));

// [deco] = (deco | n10) & deco2
OA21X1 U19(.Q(deco), .A1(n10), .A2(deco), .B1(deco2));

// [int1] = (Lo1 | Loa) & int1 | Lo1 & ~n10 & ~n1a
// #PRAGMA: zero delay
INVX1 IN_BUBBLE21(.Q(w20), .A(Lo1));
NOR3X1 U23(.Q(w22), .A(w20), .B(n10), .C(n1a));
AO221X1 U24(.Q(int1), .A1(int1), .A2(Loa), .B1(int1), .B2(Lo1), .C1(w22));

// [deco1] = monotonic_covers(SET: ~deco4, RESET: ~Loa & deco4 & ~deco5 & ~int1)
// #PRAGMA: zero delay
```

```
INVX1 IN_BUBBLE26(.Q(w25), .A(deco4));
INVX1 OUT_BUBBLE29(.Q(w27), .A(w28));
NOR4X1 U30(.Q(w28), .A(w25), .B(deco5), .C(int1), .D(Loa));
NC2 U31(.Q(deco1), .A(deco4), .B(w27));

// [deco2] = monotonic_covers(SET: int1, RESET: ~Loa & ~deco3 & ~deco5 & ~int1)
INVX1 OUT_BUBBLE34(.Q(w32), .A(w33));
NOR4X1 U35(.Q(w33), .A(deco5), .B(int1), .C(deco3), .D(Loa));
C2 U36(.Q(deco2), .A(int1), .B(w32));

// [deco3] = ~d1 | dLa
// #PRAGMA: zero delay
INVX1 IN_BUBBLE38(.Q(w37), .A(dLa));
NAND2X1 U39(.Q(deco3), .A(d1), .B(w37));

// [deco4] = d0 & ~dLa
// #PRAGMA: zero delay
INVX1 IN_BUBBLE41(.Q(w40), .A(d0));
NOR2X1 U42(.Q(deco4), .A(w40), .B(dLa));

// [deco5] = ~Li1 | Lia
// #PRAGMA: zero delay
INVX1 IN_BUBBLE44(.Q(w43), .A(Lia));
NAND2X1 U45(.Q(deco5), .A(Li1), .B(w43));

// initial values of the signals
```

```
    // !d0 !d1 !Li1 !Loa !n1a !Lia !Lo0 !Lo1 !dLa !n10 deco !int1 deco1 deco2 deco3 !
endmodule
```

A.1.4 Synthesised Verilog Specification for Cell Part CL_i'

```
module MPSAT_mapped_implementation(Lia,Lo0,Lo1,dLa,n10,n11,d0,d1,Li0,Li1,Loa,n1a);
    output Lia, Lo0, Lo1, dLa, n10, n11;
    input d0, d1, Li0, Li1, Loa, n1a;

    // [Lia] = (Li0 | Li1) & Lia | ~deco3
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE1(.Q(w0), .A(deco3));
    AO221X1 U2(.Q(Lia), .A1(Li1), .A2(Lia), .B1(Li0), .B2(Lia), .C1(w0));

    // [Lo0] = (~deco10 | ~deco11) & Lo0 | ~deco5
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE4(.Q(w3), .A(Lo0));
    AOI32X1 U5(.Q(Lo0), .A1(deco5), .A2(deco10), .A3(deco11), .B1(w3), .B2(deco5));

    // [Lo1] = monotonic_covers(SET: (Li0 | d0) & ~deco | deco14, RESET: (n10 | n11)
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE7(.Q(w6), .A(deco));
    OA32X1 U9(.Q(w8), .A1(deco14), .A2(Li0), .A3(d0), .B1(w6), .B2(deco14));
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE11(.Q(w10), .A(int4));
    OAI211X1 U13(.Q(w12), .A1(n11), .A2(n10), .B1(w10), .C1(deco));
    C2 U14(.Q(Lo1), .A(w8), .B(w12));
```

```
// [dLa] = (d0 | d1) & dLa | ~deco2
// #PRAGMA: zero delay
INVX1 IN_BUBBLE16(.Q(w15), .A(deco2));
AO221X1 U17(.Q(dLa), .A1(d1), .A2(dLa), .B1(d0), .B2(dLa), .C1(w15));

// [n10] = (~int4 | ~n1a) & n10 | deco6
// #PRAGMA: zero delay
INVX1 IN_BUBBLE19(.Q(w18), .A(int4));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE21(.Q(w20), .A(n1a));
AO221X1 U22(.Q(n10), .A1(n10), .A2(w18), .B1(w20), .B2(n10), .C1(deco6));

// [n11] = monotonic_covers(SET: (Li1 | ~deco13) & Lo1 & ~int4, RESET: int4 & n
NAND3X1 U24(.Q(w23), .A(n11), .B(int4), .C(n1a));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE26(.Q(w25), .A(Li1));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE28(.Q(w27), .A(Lo1));
AOI211X1 U30(.Q(w29), .A1(w25), .A2(deco13), .B1(w27), .C1(int4));
C2 U31(.Q(n11), .A(w23), .B(w29));

// [deco] = (Loa & deco11 | deco) & ~deco1 | deco & deco4
// #PRAGMA: zero delay
INVX1 IN_BUBBLE33(.Q(w32), .A(deco1));
OA221X1 U35(.Q(w34), .A1(deco11), .A2(deco), .B1(deco), .B2(Loa), .C1(w32));
```

```
AO21X1 U36(.Q(deco), .A1(deco), .A2(deco4), .B1(w34));

// [int2] = monotonic_covers(SET: Li1 & Lo1 & ~d1 & ~int3, RESET: (int4 | ~n11) &
// #PRAGMA: zero delay
INVX1 IN_BUBBLE38(.Q(w37), .A(Li1));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE40(.Q(w39), .A(Lo1));
NOR4X1 U42(.Q(w41), .A(w37), .B(w39), .C(int3), .D(d1));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE44(.Q(w43), .A(n11));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE46(.Q(w45), .A(Lo1));
OAI211X1 U48(.Q(w47), .A1(w43), .A2(int4), .B1(w45), .C1(int2));
C2 U49(.Q(int2), .A(w41), .B(w47));

// [int3] = (~int4 & n11 | Lo1) & (deco14 | int3)
// #PRAGMA: zero delay
INVX1 IN_BUBBLE51(.Q(w50), .A(n11));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE53(.Q(w52), .A(int3));
OAI21X1 U55(.Q(w54), .A1(int3), .A2(deco14), .B1(Lo1));
OAI31X1 U56(.Q(int3), .A1(w50), .A2(w52), .A3(int4), .B1(w54));

// [int4] = (~Lo1 | int4) & (n10 | n11 | n1a)
// #PRAGMA: zero delay
INVX1 IN_BUBBLE58(.Q(w57), .A(Lo1));
```

```
OA32X1 U59(.Q(int4), .A1(n1a), .A2(n10), .A3(n11), .B1(w57), .B2(int4));

// [deco1] = ~deco12 | ~deco4
NAND2X1 U60(.Q(deco1), .A(deco4), .B(deco12));

// [deco2] = ~Lo0 & ~deco7
NOR2X1 U61(.Q(deco2), .A(Lo0), .B(deco7));

// [deco3] = ((int4 | ~n10) & ~Lo0 & ~Lo1 | Li1) & deco13
// #PRAGMA: zero delay
INVX1 IN_BUBBLE63(.Q(w62), .A(int4));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE65(.Q(w64), .A(Li1));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE67(.Q(w66), .A(deco13));
AO221X1 U69(.Q(w68), .A1(w62), .A2(n10), .B1(w64), .B2(Lo1), .C1(w66));
NOR2X1 U70(.Q(deco3), .A(Lo0), .B(w68));

// [deco4] = (Loa | deco11) & deco4 | ~deco9
// #PRAGMA: zero delay
INVX1 IN_BUBBLE72(.Q(w71), .A(deco9));
AO221X1 U73(.Q(deco4), .A1(deco4), .A2(deco11), .B1(Loa), .B2(deco4), .C1(w71))

// [deco5] = monotonic_covers(SET: Loa, RESET: Li0 & ~Loa & ~deco10 & ~deco11)
// #PRAGMA: zero delay
INVX1 IN_BUBBLE75(.Q(w74), .A(Li0));
```

```
INVX1 OUT_BUBBLE78(.Q(w76), .A(w77));
NOR4X1 U79(.Q(w77), .A(w74), .B(deco11), .C(Loa), .D(deco10));
C2 U80(.Q(deco5), .A(Loa), .B(w76));

// [deco6] = ~Li1 & Lo1 & deco13 & ~int4
// #PRAGMA: zero delay
INVX1 IN_BUBBLE82(.Q(w81), .A(Lo1));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE84(.Q(w83), .A(deco13));
NOR4X1 U85(.Q(deco6), .A(Li1), .B(w81), .C(int4), .D(w83));

// [deco7] = ~deco14 & deco8
// #PRAGMA: zero delay
INVX1 IN_BUBBLE87(.Q(w86), .A(deco8));
NOR2X1 U88(.Q(deco7), .A(w86), .B(deco14));

// [deco8] = ~int4 & n10 | Lo1 | ~deco13
// #PRAGMA: zero delay
INVX1 IN_BUBBLE90(.Q(w89), .A(n10));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE92(.Q(w91), .A(Lo1));
OAI211X1 U93(.Q(deco8), .A1(w89), .A2(int4), .B1(w91), .C1(deco13));

// [deco9] = (d1 | deco1) & ~dLa
// #PRAGMA: zero delay
INVX1 IN_BUBBLE95(.Q(w94), .A(dLa));
```

```
OA21X1 U96(.Q(deco9), .A1(d1), .A2(deco1), .B1(w94));

// [deco10] = ~d0 | dLa
// #PRAGMA: zero delay
INVX1 IN_BUBBLE98(.Q(w97), .A(dLa));
NAND2X1 U99(.Q(deco10), .A(d0), .B(w97));

// [deco11] = ~Li0 & ~Li1 | Lia
// #PRAGMA: zero delay
INVX1 IN_BUBBLE101(.Q(w100), .A(Lia));
OAI21X1 U102(.Q(deco11), .A1(Li0), .A2(Li1), .B1(w100));

// [deco12] = monotonic_covers(SET: deco10, RESET: Li1 & ~Loa & ~deco10 & ~deco
// #PRAGMA: zero delay
INVX1 IN_BUBBLE104(.Q(w103), .A(Li1));
INVX1 OUT_BUBBLE107(.Q(w105), .A(w106));
NOR4X1 U108(.Q(w106), .A(w103), .B(deco11), .C(Loa), .D(deco10));
C2 U109(.Q(deco12), .A(deco10), .B(w105));

// [deco13] = ~int2 & ~int3
NOR2X1 U110(.Q(deco13), .A(int2), .B(int3));

// [deco14] = Li1 & d1 & ~deco & ~int3
// #PRAGMA: zero delay
INVX1 IN_BUBBLE112(.Q(w111), .A(d1));
// #PRAGMA: zero delay
```

```
    INVX1 IN_BUBBLE114(.Q(w113), .A(Li1));
    NOR4X1 U115(.Q(deco14), .A(w111), .B(w113), .C(int3), .D(deco));

    // initial values of the signals
    // !d0 !d1 !Li0 !Li1 !Loa !n1a !Lia !Lo0 !Lo1 !dLa !n10 !n11 deco !int2 !int3 !in
endmodule
```

A.1.5 Synthesised Verilog Specification for Cell Part CR'_{N_T-1}

The verilog here is not mapped to AMS 350nm standard cells.

```
module msb_control_right_Cell(Lia,Lo0,Lo1,dRa,n21,d1,Li0,Li1,Loa,n2a);
    output Lia, Lo0, Lo1, dRa, n21;
    input d1, Li0, Li1, Loa, n2a;

    assign Lia = (Li0 | Li1) & Lia | Lo0 | Lo1; // cost=5

    assign Lo0 = (Li0 & ~Lia | Lo0) & ~Loa | ~Lia & Lo0; // cost=6

    assign Lo1 = (~Lia | ~Loa | ~dRa | int1 | ~n21) & Lo1 | Li1 & ~Lia & ~Loa & d1 &

    assign dRa = d1 & dRa | ~int1 & n21 | Lo1; // cost=5

    assign n21 = (~int1 | ~n2a) & n21 | Lo1 & ~int1; // cost=5

    assign int1 = (~Lo1 | int1) & (n21 | n2a); // cost=4
```

```
    // initial values of the signals
    // !d1 !Li0 !Li1 !Loa !n2a !Lia !Lo0 !Lo1 !dRa !n21 !int1
endmodule
```

A.1.6 Synthesised Verilog Specification for Cell Part CR'_i

The verilog here is not mapped to AMS 350nm standard cells.

```
module control_right_Cell(Lia,Lo0,Lo1,dRa,n20,n21,d0,d1,Li0,Li1,Loa,n2a);
    output Lia, Lo0, Lo1, dRa, n20, n21;
    input d0, d1, Li0, Li1, Loa, n2a;

    assign Lia = (Li0 | Li1) & Lia | Lo1 & ~d0 | ~int3 & n21 | Lo0 | int1; // cost=
    // assign Lia = (Li0 | Li1) & Lia | (d1 | dRa) & Lo1 | ~int3 & n21 | Lo0 | int1

    assign Lo0 = (Li0 & ~Lia | Lo0) & ~Loa | ~Lia & Lo0; // cost=6

    assign Lo1 = (~n20 & ~n21 | ~Lia | ~Loa | ~dRa | int2 | int3) & Lo1 | (d0 | d1)

    assign dRa = (d0 | d1) & dRa | Lo1 & ~d0 | ~int3 & n21 | int1; // cost=8

    assign n20 = (~int2 | ~n2a) & n20 | Lo1 & int1 & ~int2 & ~int3; // cost=7

    assign n21 = (Lo1 & ~d0 & ~int1 & ~int2 | n21) & ~int3 | n21 & ~n2a; // cost=8
    // assign n21 = ((dRa & ~int1 | d1) & Lo1 & ~int2 | n21) & ~int3 | n21 & ~n2a;

    assign int1 = (d0 | int1) & Lo1 | ~int2 & n20; // cost=5
```

```
    // assign int1 = (~d1 & ~dRa | int1) & Lo1 | ~int2 & n20; // cost=6

    assign int2 = (n20 | n2a) & int2 | ~Lo1 & n20; // cost=5

    assign int3 = (n21 | n2a) & int3 | ~Lo1 & n21; // cost=5

    // initial values of the signals
    // !d0 !d1 !Li0 !Li1 !Loa !n2a !Lia !Lo0 !Lo1 !dRa !n20 !n21 !int1 !int2 !int3
endmodule
```

A.1.7 Synthesised Verilog Specification for Cell Part CR'_0

```
module lsb_control_right_Cell(Lo,Lia,dRa,n20,n21,d0,d1,Li0,Li1,Loa,n2a);
    output Lo, Lia, dRa, n20, n21;
    input d0, d1, Li0, Li1, Loa, n2a;

    // [Lo] = Lo & deco7 | ~deco
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE1(.Q(w0), .A(deco));
    C2 U2(.Q(Lo), .A(w0), .B(deco7));

    // [Lia] = (Li0 | Li1) & Lia | ~Li1 & Lo | int1 | int2
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE4(.Q(w3), .A(Li1));
    AO211X1 U6(.Q(w5), .A1(Lia), .A2(Li1), .B1(int2), .C1(int1));
    AO221X1 U7(.Q(Lia), .A1(w3), .A2(Lo), .B1(Li0), .B2(Lia), .C1(w5));
```

```
// [dRa] = (d0 | d1) & dRa | int1 | int2
AOI21X1 U9(.Q(w8), .A1(dRa), .A2(d0), .B1(int1));
AOI21X1 U11(.Q(w10), .A1(dRa), .A2(d1), .B1(int2));
NAND2X1 U12(.Q(dRa), .A(w8), .B(w10));

// [n20] = (~int3 | ~n2a) & n20 | deco1 & int1 & ~int3
// #PRAGMA: zero delay
INVX1 IN_BUBBLE14(.Q(w13), .A(n20));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE16(.Q(w15), .A(int3));
OAI211X1 U18(.Q(w17), .A1(deco1), .A2(n20), .B1(w15), .C1(int1));
OAI21X1 U19(.Q(n20), .A1(w13), .A2(n2a), .B1(w17));

// [n21] = (~int3 | ~n2a) & n21 | deco1 & int2 & ~int3
// #PRAGMA: zero delay
INVX1 IN_BUBBLE21(.Q(w20), .A(n21));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE23(.Q(w22), .A(int3));
OAI211X1 U25(.Q(w24), .A1(deco1), .A2(n21), .B1(w22), .C1(int2));
OAI21X1 U26(.Q(n21), .A1(w20), .A2(n2a), .B1(w24));

// [deco] = (Lia | deco) & deco5
C2 U27(.Q(deco), .A(Lia), .B(deco5));

// [int1] = (Li1 & d0 | int1) & Lo | ~int3 & n20
```

```
// #PRAGMA: zero delay
INVX1 IN_BUBBLE29(.Q(w28), .A(n20));
AOI32X1 U31(.Q(w30), .A1(Li1), .A2(Lo), .A3(d0), .B1(int1), .B2(Lo));
OAI21X1 U32(.Q(int1), .A1(w28), .A2(int3), .B1(w30));

// [int2] = (Li1 & d1 | int2) & Lo | ~int3 & n21
// #PRAGMA: zero delay
INVX1 IN_BUBBLE34(.Q(w33), .A(n21));
AOI32X1 U36(.Q(w35), .A1(Li1), .A2(Lo), .A3(d1), .B1(int2), .B2(Lo));
OAI21X1 U37(.Q(int2), .A1(w33), .A2(int3), .B1(w35));

// [int3] = (~Lo | int3) & (~deco6 | n2a)
// #PRAGMA: zero delay
INVX1 IN_BUBBLE39(.Q(w38), .A(int3));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE41(.Q(w40), .A(n2a));
AOI22X1 U42(.Q(int3), .A1(Lo), .A2(w38), .B1(w40), .B2(deco6));

// [deco1] = (int1 | int2) & ~deco | ~deco2
// #PRAGMA: zero delay
INVX1 IN_BUBBLE44(.Q(w43), .A(int1));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE46(.Q(w45), .A(int2));
AOI32X1 U47(.Q(deco1), .A1(w43), .A2(w45), .A3(deco2), .B1(deco), .B2(deco2));

// [deco2] = ~Li0 & ~deco1 & deco4 | ~deco3 | ~deco7
```

```
// #PRAGMA: zero delay
INVX1 IN_BUBBLE49(.Q(w48), .A(deco4));
OAI311X1 U50(.Q(deco2), .A1(Li0), .A2(deco1), .A3(w48), .B1(deco3), .C1(deco7))

// [deco3] = (Li0 | Li1) & ~Lia
// #PRAGMA: zero delay
INVX1 IN_BUBBLE52(.Q(w51), .A(Lia));
OA21X1 U53(.Q(deco3), .A1(Li0), .A2(Li1), .B1(w51));

// [deco4] = ~d0 & ~d1 | dRa
// #PRAGMA: zero delay
INVX1 IN_BUBBLE55(.Q(w54), .A(dRa));
OAI21X1 U56(.Q(deco4), .A1(d0), .A2(d1), .B1(w54));

// [deco5] = deco4 & ~deco6 & ~int3 | ~deco1
// #PRAGMA: zero delay
INVX1 IN_BUBBLE58(.Q(w57), .A(deco4));
OAI31X1 U59(.Q(deco5), .A1(int3), .A2(w57), .A3(deco6), .B1(deco1));

// [deco6] = ~n20 & ~n21
NOR2X1 U60(.Q(deco6), .A(n20), .B(n21));

// [deco7] = ~Loa | deco1
// #PRAGMA: zero delay
INVX1 IN_BUBBLE62(.Q(w61), .A(deco1));
NAND2X1 U63(.Q(deco7), .A(Loa), .B(w61));
```

```
    // initial values of the signals
    // !d0 !d1 !Li0 !Li1 !Loa !n2a !Lo !Lia !dRa !n20 !n21 deco !int1 !int2 !int3 !de
endmodule
```

A.2 Control Cells

A.2.1 Synthesised Verilog Specification for Control Cell c'_{N_T-1}

```
module MSB (Lo0,Lo1,Lia,n10,n21,Loa,Li,d,n1a,n2a,rst);
output  Lo0, Lo1, Lia, n10, n21;
input  Loa, Li, d, n1a, n2a, rst;

DualRail I2 ( .da(net3), .d1(net2), .d0(net1), .d(d), .rst(rst));
DualRail2 I4 ( .da(net4), .d1(net016), .d(d), .rst(rst));
msb_right_control I7 ( .n2a(n2a), .n21(n21), .da(net4), .d1(net016),
     .Lo0(Lo0), .Lo1(Lo1), .Loa(Loa), .L0(net020), .L1(net019), .Lia(net018),
     .rst(rst));
msb_left_control I3 ( .n10(n10), .d0(net1), .Li(Li), .n1a(n1a),
     .da(net3), .d1(net2), .Lo0(net020), .Lo1(net019), .Loa(net018),
     .Lia(Lia), .rst(rst));

endmodule
```

A.2.2 Synthesised Verilog Specification for Control Cell c'_i

```
module Middle (Lo0,Lo1,Lia,n10,n11,n20,n21,Loa,Li0,Li1,d,n1a,n2a,rst);
```

```
output  Lo0, Lo1, Lia, n10, n11, n20, n21;
input  Loa, Li0, Li1, d1, n1a, n2a, rst;

DualRail I2 ( .da(net19), .d1(net020), .d0(net021), .d(d), .rst(rst));
DualRail I3 ( .da(net18), .d1(net16), .d0(net15), .d(d), .rst(rst));
left_control I8 ( .n1a(n1a), .n11(n11), .n10(n10), .da(net19),
     .d1(net020), .d0(net021), .Lo0(net20), .Lo1(net21), .Loa(net22),
     .Li0(Li0), .Li1(Li1), .Lia(Lia), .rst(rst));
right_control I9 ( .n2a(n2a), .n21(n21), .n20(n20), .da(net18),
     .d1(net16), .d0(net15), .Lo0(Lo0), .Lo1(Lo1), .Loa(Loa), .Li0(net20),
     .Li1(net21), .Lia(net22), .rst(rst));

endmodule
```

A.2.3 Synthesised Verilog Specification for Control Cell c'_0

```
module LSB (Lo,Lia,n10,n11,n20,n21,Loa,Li0,Li1,d,n1a,n2a,rst);
output  Lo, Lia, n10, n11, n20, n21;
input  Loa, Li0, Li1, d, n1a, n2a, rst;

left_control I8 ( .n1a(n1a), .n11(n11), .n10(n10), .da(net17),
     .d1(net14), .d0(net13), .K0(net18), .K1(net19), .Loa(net20),
     .Li0(Li0), .Li1(Li1), .Lia(Lia), .rst(rst));
DualRail I2 ( .da(net17), .d1(net14), .d0(net13), .d(d), .rst(rst));
DualRail I3 ( .da(net16), .d1(net016), .d0(net015), .d(d), .rst(rst));
lsb_right_control I10 ( .n2a(n2a), .n21(n21), .n20(n20), .da(net16),
     .d1(net016), .d0(net015), .Lo(Lo), .Loa(Loa), .Li0(net18), .Li1(net19),
```

```
     .Lia(net20), .rst(rst));

endmodule
```

A.3 Control Block

A.3.1 Synthesised Verilog Specification for Five Bit Control Block

```
module control_5bits ( .cdsTerm28(cdsNet27), .cdsTerm29(cdsNet26),
     .cdsTerm30(cdsNet25), .cdsTerm31(cdsNet24), .cdsTerm32(cdsNet21),
     .cdsTerm33(cdsNet20), .cdsTerm34(cdsNet19), .cdsTerm35(cdsNet18),
     .cdsTerm36(cdsNet15), .cdsTerm37(cdsNet14), .cdsTerm38(cdsNet13),
     .cdsTerm39(cdsNet12), .cdsTerm40(cdsNet9), .cdsTerm41(cdsNet8),
     .cdsTerm42(cdsNet7), .cdsTerm43(cdsNet6), .cdsTerm44(cdsNet3),
     .cdsTerm45(cdsNet2), Lo, Lia, .cdsTerm46(cdsNet23),
     .cdsTerm47(cdsNet22), .cdsTerm48(cdsNet17), .cdsTerm49(cdsNet16),
     .cdsTerm50(cdsNet11), .cdsTerm51(cdsNet10), .cdsTerm52(cdsNet5),
     .cdsTerm53(cdsNet4), .cdsTerm54(cdsNet1), .cdsTerm55(cdsNet0), Loa,
     Li, d0, d1, d2, d3, d4, rst );

output  cdsNet27, cdsNet26, cdsNet25, cdsNet24, cdsNet21, cdsNet20,
     cdsNet19, cdsNet18, cdsNet15, cdsNet14, cdsNet13, cdsNet12,
     cdsNet9, cdsNet8, cdsNet7, cdsNet6, cdsNet3, cdsNet2, Lo, Lia;

input  cdsNet23, cdsNet22, cdsNet17, cdsNet16, cdsNet11, cdsNet10,
     cdsNet5, cdsNet4, cdsNet1, cdsNet0, Loa, Li, d0, d1, d2, d3, d4,
     rst;
```

```
Middle I4 ( .n2a(cdsNet16), .n21(cdsNet18), .n20(cdsNet19),
     .n1a(cdsNet17), .n11(cdsNet20), .n10(cdsNet21), .d(d1),
     .Lo0(net102), .Lo1(net101), .Loa(net100), .Li0(net110), .Li1(net111),
     .Lia(net109), .rst(rst));
Middle I3 ( .n2a(cdsNet10), .n21(cdsNet12), .n20(cdsNet13),
     .n1a(cdsNet11), .n11(cdsNet14), .n10(cdsNet15), .d(d2),
     .Lo0(net110), .Lo1(net111), .Loa(net109), .Li0(net113), .Li1(net114),
     .Lia(net112), .rst(rst));
Middle I0 ( .n2a(cdsNet4), .n21(cdsNet6), .n20(cdsNet7),
     .n1a(cdsNet5), .n11(cdsNet8), .n10(cdsNet9), .d(d3), .Lo0(net113),
     .Lo1(net114), .Loa(net112), .Li0(net108), .Li1(net106), .Lia(net107),
     .rst(rst));
MSB I1 ( .n2a(cdsNet0), .n21(cdsNet2), .n1a(cdsNet1), .n10(cdsNet3),
     .d(d4), .Lo0(net108), .Lo1(net106), .Loa(net107), .Li(Li), .Lia(Lia),
     .rst(rst));
LSB I2 ( .n2a(cdsNet22), .n21(cdsNet24), .n20(cdsNet25),
     .n1a(cdsNet23), .n11(cdsNet26), .n10(cdsNet27), .d(d0), .Lo(Lo),
     .Loa(Loa), .Li0(net102), .Li1(net101), .Lia(net100), .rst(rst));

endmodule
```

A.4 Counter Cell Parts

A.4.1 Synthesised Verilog Specification for Cell Part CL_{N_T-1}

```
module msb_counter_left_cell(br, n1k, bk, n10);
```

```
    output br, n1k;
    input bk, n10;

    // mapped implementation; total area=68.00

    // [br] = (n10 & ~n1k | br) & ~bk | br & ~n1k
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE1(.Q(w0), .A(bk));
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE3(.Q(w2), .A(n10));
    NOR2X1 U5(.Q(w4), .A(w2), .B(n1k));
    C2 U6(.Q(br), .A(w0), .B(w4));

    // [n1k] = n10 & n1k | br
    C2 U7(.Q(n1k), .A(n10), .B(br));

    // initial values of the signals
    // !bk !n10 !br !n1k w0 w2 !w4
endmodule
```

A.4.2 Synthesised Verilog Specification for Cell Part CL_i

```
module left_cell(ar, br, ca, da, n1a, aa, ba, cr, dr, n10, n11);
    output ar, br, ca, da, n1a;
    input aa, ba, cr, dr, n10, n11;

    // mapped implementation; total area=516.00
```

```
// [ar] = (ar & ~ca | ~int1) & int3 | int1 & ~int3
// #PRAGMA: zero delay
INVX1 IN_BUBBLE1(.Q(w0), .A(int3));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE3(.Q(w2), .A(ca));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE5(.Q(w4), .A(int3));
AOI32X1 U7(.Q(w6), .A1(w2), .A2(int1), .A3(ar), .B1(w4), .B2(int1));
OAI21X1 U8(.Q(ar), .A1(w0), .A2(int1), .B1(w6));

// [br] = br & ~da & ~int2 | ~deco
// #PRAGMA: zero delay
INVX1 IN_BUBBLE10(.Q(w9), .A(br));
OAI31X1 U11(.Q(br), .A1(w9), .A2(da), .A3(int2), .B1(deco));

// [ca] = ar & int1 | ca & cr
AO22X1 U12(.Q(ca), .A1(ar), .A2(int1), .B1(cr), .B2(ca));

// [da] = br & ~int2 | da & dr
// #PRAGMA: zero delay
INVX1 IN_BUBBLE14(.Q(w13), .A(int2));
AO22X1 U15(.Q(da), .A1(br), .A2(w13), .B1(dr), .B2(da));

// [n1a] = (int2 | n10 | n11) & n1a | br
AO211X1 U17(.Q(w16), .A1(n11), .A2(n11), .B1(int2), .C1(n10));
```

```
C2 U18(.Q(n1a), .A(br), .B(w16));

// [deco] = (ba | deco) & deco1
C2 U19(.Q(deco), .A(ba), .B(deco1));

// [int1] = aa & int1 | deco2
AO21X1 U20(.Q(int1), .A1(aa), .A2(int1), .B1(deco2));

// [int2] = br & int2 | n10 & ~n1a
// #PRAGMA: zero delay
INVX1 IN_BUBBLE22(.Q(w21), .A(n1a));
AO22X1 U23(.Q(int2), .A1(n10), .A2(w21), .B1(br), .B2(int2));

// [int3] = aa & int1 | ~aa & int3
// #PRAGMA: zero delay
INVX1 IN_BUBBLE25(.Q(w24), .A(aa));
AO22X1 U26(.Q(int3), .A1(w24), .A2(int3), .B1(aa), .B2(int1));

// [deco1] = (~n10 & ~n1a | ~deco3) & ~deco5
AOI221X1 U27(.Q(deco1), .A1(n1a), .A2(deco3), .B1(n10),
             .B2(deco3), .C1(deco5));

// [deco2] = (~ca & cr | int1) & ~deco4
// #PRAGMA: zero delay
INVX1 IN_BUBBLE29(.Q(w28), .A(cr));
// #PRAGMA: zero delay
```

```
INVX1 IN_BUBBLE31(.Q(w30), .A(int1));
OAI32X1 U32(.Q(deco2), .A1(w28), .A2(ca), .A3(deco4), .B1(w30),
            .B2(deco4));

// [deco3] = (~da & n11 | ~deco | int2) & ~n1a
// #PRAGMA: zero delay
INVX1 IN_BUBBLE34(.Q(w33), .A(int2));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE36(.Q(w35), .A(n11));
AO211X1 U38(.Q(w37), .A1(deco), .A2(da), .B1(w35), .C1(n1a));
OAI21X1 U39(.Q(deco3), .A1(w33), .A2(n1a), .B1(w37));

// [deco4] = aa | int3 | ~n11 | n1a
// #PRAGMA: zero delay
INVX1 IN_BUBBLE41(.Q(w40), .A(n11));
INVX1 OUT_BUBBLE43(.Q(deco4), .A(w42));
NOR4X1 U44(.Q(w42), .A(aa), .B(w40), .C(n1a), .D(int3));

// [deco5] = monotonic_covers(SET: deco3 & ~deco4 & dr & ~int1,
   RESET: deco4)
// #PRAGMA: zero delay
INVX1 IN_BUBBLE46(.Q(w45), .A(deco4));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE48(.Q(w47), .A(deco4));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE50(.Q(w49), .A(int1));
```

```
    NAND4X1 U52(.Q(w51), .A(w47), .B(w49), .C(deco3), .D(dr));
    NC2 U53(.Q(deco5), .A(w45), .B(w51));

    // initial values of the signals
    // !aa !ba !cr !dr !n10 !n11 !ar !br !ca !da !n1a deco !int1 !int2
       !int3 deco1 !deco2 !deco3 deco4 !deco5 w2 !w16 w30 w33 w0 w6 w4
       w9 w13 w21 w24 w28 w35 w37 w40 !w42 !w45 !w47 w49 w51
endmodule
```

A.4.3 Synthesised Verilog Specification for Cell Part CR_{N_T-1}

```
module  msc_right_cell(ar, br, da, n2a, aa, ba, dr, n21);
    output ar, br, da, n2a;
    input aa, ba, dr, n21;

    // mapped implementation; total area=208.00

    // [ar] = ~int & n21 & ~n2a
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE1(.Q(w0), .A(n21));
    NOR3X1 U2(.Q(ar), .A(w0), .B(n2a), .C(int));

    // [br] = (~ba | ~n2a) & br | ~deco
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE4(.Q(w3), .A(br));
    AOI32X1 U5(.Q(br), .A1(ba), .A2(n2a), .A3(deco), .B1(w3), .B2(deco));
```

```
// [da] = da & dr | br
C2 U6(.Q(da), .A(dr), .B(br));

// [n2a] = (int | n21) & n2a | br
OA22X1 U7(.Q(n2a), .A1(n21), .A2(int), .B1(br), .B2(n2a));

// [int] = int & ~n2a | aa | br
// #PRAGMA: zero delay
INVX1 IN_BUBBLE9(.Q(w8), .A(n2a));
AO211X1 U10(.Q(int), .A1(w8), .A2(int), .B1(aa), .C1(br));

// [deco] = deco & n2a | aa | ~deco1 | ~int
// #PRAGMA: zero delay
INVX1 IN_BUBBLE12(.Q(w11), .A(aa));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE14(.Q(w13), .A(deco1));
AOI21X1 U16(.Q(w15), .A1(deco), .A2(n2a), .B1(w13));
NAND3X1 U17(.Q(deco), .A(w11), .B(int), .C(w15));

// [deco1] = (~ba & dr | ~deco) & ~da
// #PRAGMA: zero delay
INVX1 IN_BUBBLE19(.Q(w18), .A(dr));
AOI211X1 U20(.Q(deco1), .A1(ba), .A2(deco), .B1(w18), .C1(da));

// initial values of the signals
// !aa !ba !dr !n21 !ar !br !da !n2a !int deco !deco1 w3 w0 w8 w11
```

```
     w13 !w15 w18
endmodule
```

A.4.4 Synthesised Verilog Specification for Cell Part CR_i

```
module right_cell(ar, br, ck, dk, n2k, ak, bk, cr, dr, n20, n21);
    output ar, br, ck, dk, n2k;
    input ak, bk, cr, dr, n20, n21;

    // [ar] = ar & deco5 | ~deco
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE1(.Q(w0), .A(deco));
    AO21X1 U2(.Q(ar), .A1(ar), .A2(deco5), .B1(w0));

    // [br] = (br | ~deco1) & (~bk | deco5 | ~n2k)
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE4(.Q(w3), .A(deco5));
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE6(.Q(w5), .A(br));
    AOI32X1 U7(.Q(br), .A1(bk), .A2(n2k), .A3(w3), .B1(w5), .B2(deco1));

    // [ck] = (ck | int2 | n20) & ar | ck & cr
    OA31X1 U9(.Q(w8), .A1(int2), .A2(ck), .A3(n20), .B1(ar));
    C2 U10(.Q(ck), .A(cr), .B(w8));

    // [dk] = dk & dr | br
```

```
C2 U11(.Q(dk), .A(dr), .B(br));

// [n2k] = (n20 | n21) & n2k | br & ~int1
AOI221X1 U13(.Q(w12), .A1(n2k), .A2(n21), .B1(n2k), .B2(n20), .C1(br));
NOR2X1 U14(.Q(n2k), .A(int1), .B(w12));

// [deco] = ~deco2 & ~deco3 & ~int2
NOR3X1 U15(.Q(deco), .A(int2), .B(deco2), .C(deco3));

// [int1] = ak & n21 | ~br & int1
// #PRAGMA: zero delay
INVX1 IN_BUBBLE17(.Q(w16), .A(br));
AO22X1 U18(.Q(int1), .A1(w16), .A2(int1), .B1(ak), .B2(n21));

// [int2] = monotonic_covers(SET: ~ck & cr & deco5 & int1,
RESET: ck & ~deco5 & int2)
// #PRAGMA: zero delay
INVX1 IN_BUBBLE20(.Q(w19), .A(deco5));
NAND3X1 U22(.Q(w21), .A(w19), .B(int2), .C(ck));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE24(.Q(w23), .A(ck));
INVX1 OUT_BUBBLE27(.Q(w25), .A(w26));
NAND4X1 U28(.Q(w26), .A(w23), .B(deco5), .C(int1), .D(cr));
C2 U29(.Q(int2), .A(w21), .B(w25));

// [deco1] = bk & ~br | ~deco4 | ~deco5
```

```
// #PRAGMA: zero delay
INVX1 IN_BUBBLE31(.Q(w30), .A(bk));
OAI211X1 U32(.Q(deco1), .A1(w30), .A2(br), .B1(deco4), .C1(deco5));

// [deco2] = ~br & ~int1 & n21 & ~n2k
// #PRAGMA: zero delay
INVX1 IN_BUBBLE34(.Q(w33), .A(n21));
NOR4X1 U35(.Q(deco2), .A(w33), .B(br), .C(n2k), .D(int1));

// [deco3] = monotonic_covers(SET: ~ck & cr & deco5 & n20, RESET: ck)
// #PRAGMA: zero delay
INVX1 IN_BUBBLE37(.Q(w36), .A(ck));
// #PRAGMA: zero delay
INVX1 IN_BUBBLE39(.Q(w38), .A(ck));
NAND4X1 U41(.Q(w40), .A(w38), .B(deco5), .C(n20), .D(cr));
NC2 U42(.Q(deco3), .A(w36), .B(w40));

// [deco4] = ~dk & dr
// #PRAGMA: zero delay
INVX1 IN_BUBBLE44(.Q(w43), .A(dr));
NOR2X1 U45(.Q(deco4), .A(w43), .B(dk));

// [deco5] = (n20 & ~n2k | int1) & ~ak | ~deco1
// #PRAGMA: zero delay
INVX1 IN_BUBBLE47(.Q(w46), .A(n2k));
AOI21X1 U49(.Q(w48), .A1(w46), .A2(n20), .B1(int1));
```

```
    OAI21X1 U50(.Q(deco5), .A1(ak), .A2(w48), .B1(deco1));

    // initial values of the signals
    // !ak !bk !cr !dr !n20 !n21 !ar !br !ck !dk !n2k deco !int1
    !int2 deco1 !deco2 !deco3 !deco4 !deco5 w48 !w8 !w25 w19
    w30 w33 w3 !w0 w5 w12 w16 w21 w23 w26 w36 w38 w40 w43 w46
endmodule
```

A.5 Control Cells

A.5.1 Synthesised Verilog Specification for Control Cell c_{N_T-1}

```
module MSB ( ar, br, n1a, n2a, aa, ba, n10, n21, rst );

output  ar, br, n1a, n2a;

input  aa, ba, n10, n21, rst;

msb_left_counter I17 ( .nk(n1a), .n10(n10), .br(net23), .ba(net24),
     .rst(rst));
msb_right_counter I18 ( .nk(n2a), .n21(n21), .ar(ar), .aa(aa),
     .br(br), .ba(ba), .dr(net23), .dk(net24), .rst(rst));

endmodule
```

A.5.2 Synthesised Verilog Specification for Control Cell c_i

```
module Middle ( ar, br, ck, dk, n1a, n2a, aa, ba, cr, dr, n10, n11,
     n20, n21, rst );

output  ar, br, ck, dk, n1a, n2a;

input  aa, ba, cr, dr, n10, n11, n20, n21, rst;

right_counter I25 ( .nk(n2a), .n21(n21), .n20(n20), .ar(ar), .aa(aa),
     .br(br), .ba(ba), .cr(net35), .ck(net38), .dr(net37), .dk(net36),
     .rst(rst));
left_counter I21 ( .rst(rst), .nk(n1a), .n11(n11), .n10(n10),
     .ar(net35), .aa(net38), .br(net37), .ba(net36), .cr(cr), .ck(ck),
     .dr(dr), .dk(dk));

endmodule
```

A.6 Control Block

A.6.1 Synthesised Verilog Specification for Five Bit Counter Block

```
module counter_5bits ( .cdsTerm28(cdsNet23), .cdsTerm29(cdsNet22),
     .cdsTerm30(cdsNet17), .cdsTerm31(cdsNet16), .cdsTerm32(cdsNet11),
     .cdsTerm33(cdsNet10), .cdsTerm34(cdsNet5), .cdsTerm35(cdsNet4),
     .cdsTerm36(cdsNet1), .cdsTerm37(cdsNet0), ar, br,
     .cdsTerm38(cdsNet27), .cdsTerm39(cdsNet26), .cdsTerm40(cdsNet25),
     .cdsTerm41(cdsNet24), .cdsTerm42(cdsNet21), .cdsTerm43(cdsNet20),
```

```
     .cdsTerm44(cdsNet19), .cdsTerm45(cdsNet18), .cdsTerm46(cdsNet15),
     .cdsTerm47(cdsNet14), .cdsTerm48(cdsNet13), .cdsTerm49(cdsNet12),
     .cdsTerm50(cdsNet9), .cdsTerm51(cdsNet8), .cdsTerm52(cdsNet7),
     .cdsTerm53(cdsNet6), .cdsTerm54(cdsNet3), .cdsTerm55(cdsNet2), aa,
     ba, rst );

output  cdsNet23, cdsNet22, cdsNet17, cdsNet16, cdsNet11, cdsNet10,
     cdsNet5, cdsNet4, cdsNet1, cdsNet0, ar, br;

input  cdsNet27, cdsNet26, cdsNet25, cdsNet24, cdsNet21, cdsNet20,
     cdsNet19, cdsNet18, cdsNet15, cdsNet14, cdsNet13, cdsNet12,
     cdsNet9, cdsNet8, cdsNet7, cdsNet6, cdsNet3, cdsNet2, aa, ba, rst;


specify
    specparam CDS_LIBNAME  = "new_counter";
    specparam CDS_CELLNAME = "counter_5bits";
    specparam CDS_VIEWNAME = "schematic";
endspecify

MSB I0 ( .n2a(cdsNet0), .n21(cdsNet2), .n1a(cdsNet1), .n10(cdsNet3),
     .ar(net79), .aa(net78), .br(net77), .ba(net76), .rst(rst));
Middle I4 ( .n2a(cdsNet22), .n21(cdsNet24), .n20(cdsNet25),
     .n1a(cdsNet23), .n11(cdsNet26), .n10(cdsNet27), .ar(ar), .aa(aa),
     .br(br), .ba(ba), .cr(net101), .ck(net100), .dr(net99),
     .dk(net98), .rst(rst));
```

```
Middle I3 ( .n2a(cdsNet16), .n21(cdsNet18), .n20(cdsNet19),
     .n1a(cdsNet17), .n11(cdsNet20), .n10(cdsNet21), .ar(net101),
     .aa(net100), .br(net99), .ba(net98), .cr(net94), .ck(net93),
     .dr(net92), .dk(net91), .rst(rst));
Middle I2 ( .n2a(cdsNet10), .n21(cdsNet12), .n20(cdsNet13),
     .n1a(cdsNet11), .n11(cdsNet14), .n10(cdsNet15), .ar(net94),
     .aa(net93), .br(net92), .ba(net91), .cr(net87), .ck(net86),
     .dr(net85), .dk(net84), .rst(rst));
Middle I1 ( .n2a(cdsNet4), .n21(cdsNet6), .n20(cdsNet7),
     .n1a(cdsNet5), .n11(cdsNet8), .n10(cdsNet9), .ar(net87),
     .aa(net86), .br(net85), .ba(net84), .cr(net79), .ck(net78),
     .dr(net77), .dk(net76), .rst(rst));

endmodule
```

A.7 Complete Counter

A.7.1 Verilog Specification for Five Bit Loadable Modulo$-n$ Counter

```
module Complete_Counter ( Wia, ar, br, Wi, aa, ba, d0, d1, d2,
     d3, d4, reset );

output  Wia, ar, br;

input  Wi, aa, ba, d0, d1, d2, d3, d4, reset;

control_5bits I0 ( .cdsTerm47(net28), .cdsTerm31(net29),
```

```
     .cdsTerm30(net30), .cdsTerm46(net31), .cdsTerm29(net32),
     .cdsTerm28(net33), .cdsTerm49(net34), .cdsTerm35(net35),
     .cdsTerm34(net36), .cdsTerm48(net37), .cdsTerm33(net38),
     .cdsTerm32(net39), .cdsTerm51(net40), .cdsTerm39(net41),
     .cdsTerm38(net42), .cdsTerm50(net43), .cdsTerm37(net44),
     .cdsTerm36(net45), .cdsTerm53(net46), .cdsTerm43(net47),
     .cdsTerm42(net48), .cdsTerm52(net49), .cdsTerm41(net50),
     .cdsTerm40(net51), .cdsTerm55(net52), .cdsTerm45(net53),
     .cdsTerm54(net54), .cdsTerm44(net55), .d4(d4), .d3(d3), .d2(d2),
     .d1(d1), .d0(d0), .Lo(net20), .Loa(net21), .Li(net16), .Lia(net15),
     .rst(reset));
counter_5bits I1 ( .cdsTerm29(net28), .cdsTerm41(net29),
     .cdsTerm40(net30), .cdsTerm28(net31), .cdsTerm39(net32),
     .cdsTerm38(net33), .cdsTerm31(net34), .cdsTerm45(net35),
     .cdsTerm44(net36), .cdsTerm30(net37), .cdsTerm43(net38),
     .cdsTerm42(net39), .cdsTerm33(net40), .cdsTerm49(net41),
     .cdsTerm48(net42), .cdsTerm32(net43), .cdsTerm47(net44),
     .cdsTerm46(net45), .cdsTerm35(net46), .cdsTerm53(net47),
     .cdsTerm52(net48), .cdsTerm34(net49), .cdsTerm51(net50),
     .cdsTerm50(net51), .cdsTerm37(net52), .cdsTerm55(net53),
     .cdsTerm36(net54), .cdsTerm54(net55), .ar(ar), .aa(aa), .br(br),
     .ba(ba), .rst(reset));
wrapper I2 ( .Loa(net21), .Lo(net20), .Lia(net15), .Li(net16),
     .Wia(Wia), .Wi(Wi), .rst(reset));


endmodule
```

A.8 Counter Test

A.8.1 VHDL Code to Test Implemented Counter

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity test is
  port (clk,reset,ar_in,br_in,load,asel,bsel,loadack: in std_logic;
        num: in std_logic_vector(4 downto 0);
        ak,bk,asel1,asel2, bsel1,bsel2,load2,load3,loadack2,loadack3,
        reset2, reset3,l0, l1, l2, l3, l4, l100, l6, l7: out std_logic;
        num1, num2: out std_logic_vector(4 downto 0));
end entity;

architecture behavoural of test is
    signal reset1, load1: std_logic;
    signal cnt: integer range 0 to 100;
    type state_cntrl is (s0, s1, s2, s3, s4, s100, s6, s7);
    signal state1: state_cntrl ;

begin
    process (clk)
        begin
            if (clk='1' and clk'event) then
                if reset1 = '1' then
```

```
        state1 <=s0; -- reset state
    else
        case state1 is
            when s0 =>
                if load1 = '1' then
                    state1 <=s1;
                end if;
            when s1 =>
                if ar_in = '1' then
                    state1 <=s2;
                elsif br_in = '1' then
                    state1 <= s100;
                end if;
            when s2 =>
                if cnt = 100 then
                    state1 <= s3;
                end if;
            when s3 =>
                if ar_in = '0' then
                    state1 <= s4;
                end if;
            when s4 =>
                if cnt = 100 then
                    state1 <= s1;
                end if;
            when s100 =>
```

```
                        if cnt = 100 then
                            state1 <= s6;
                        end if;
                    when s6 =>
                        if br_in = '0' then
                            state1 <= s7;
                        end if;
                    when s7 =>
                        if cnt = 100 then
                            state1 <= s1;
                        end if;
                end case;
            end if;
        end if;
end process;

process (clk)
begin
if (clk='1' and clk'event) then
if reset1 = '1' or state1 = s1 or state1 = s3 or state1 = s6 then
   cnt <= 0;
else
   cnt <= cnt+1;
end if;
end if;
end process;
```

```
    reset1 <= not(reset);
    reset2 <= not(reset);
    reset3 <= not(reset);
    load1 <= not(load);
    load2 <= not(load);
    load3 <= not(load);

    loadack2 <= not(loadack);
    loadack3 <= not(loadack);

    asel1 <= not (asel);
    asel2 <= not (asel);
    bsel1 <= not (bsel);
    bsel2 <= not (bsel);

    num1 <= not (num);
    num2 <= not (num);

--  ak<=ar_in;
--  bk<=br_in;
            ak <= '1' when state1 = s3 else '1' when state1 = s4 else '0';
            bk <= '1' when state1 = s6 else '1' when state1 = s7 else '0';

            l0  <= '1' when state1 = s0 else '0';
            l1  <= '1' when state1 = s1 else '0';
```

```
            l2  <= '1' when state1 = s2 else '0';
            l3  <= '1' when state1 = s3 else '0';
            l4  <= '1' when state1 = s4 else '0';
            l100  <= '1' when state1 = s100 else '0';
            l6  <= '1' when state1 = s6 else '0';
            l7  <= '1' when state1 = s7 else '0';
end behavoural;
```

Appendix B

Verilog Code For DPWM Circuits

B.1 C element

```
module celement ( c, a, b );


output  c;
input  a, b;


NAND2X1  I5 ( .A(c), .B(a), .Q(net12));
NAND2X1  I4 ( .A(a), .B(b), .Q(net14));
NAND2X1  I6 ( .A(b), .B(c), .Q(net13));
NAND3X1  I7 ( .C(net13), .Q(c), .A(net12), .B(net14));


endmodule
```

B.2 Output Block

B.2.1 Latch Control Block

```
module latch_control(R, S, br1a, br2a);
    output R, S;
    input br1a, br2a;

    // mapped implementation; total area=56.00

    // [R] = R & br1a | ~S & br2a
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE1(.Q(w0), .A(S));
    AO22X1 U2(.Q(R), .A1(br2a), .A2(w0), .B1(br1a), .B2(R));

    // [S] = ~R & br1a & ~br2a
    // #PRAGMA: zero delay
    INVX1 IN_BUBBLE4(.Q(w3), .A(br1a));
    NOR3X1 U5(.Q(S), .A(w3), .B(br2a), .C(R));

    // initial values of the signals
    // !br1a !br2a !R !S w3 w0
endmodule
```

B.2.2 Latch Output Block

```
module new_LatchSync ( R, S, br1, br2 );
```

```
output  R, S;

input  br1, br2;

latch_control I0 ( .S(net7), .R(net6), .br1(br1), .br2(br2));
NOR2X1  I20 ( .A(net7), .B(S), .Q(R));
NOR2X1  I21 ( .A(R), .B(net6), .Q(S));

endmodule
```

B.3 Synchronisation Block

```
module sync_block(aa, ba, ar1, ar2, br1, br2);
    output aa, ba;
    input ar1, ar2, br1, br2;

    // mapped implementation; total area=72.00

    // [aa] = (aa | br1) & ar2 | (aa | ar2 | br2) & ar1
    OA31X1 U1(.Q(w0), .A1(aa), .A2(ar2), .A3(br2), .B1(ar1));
    AO221X1 U2(.Q(aa), .A1(aa), .A2(ar2), .B1(ar2), .B2(br1), .C1(w0));

    // [ba] = (ba | br2) & br1 | ba & br2
    C2 U3(.Q(ba), .A(br1), .B(br2));

    // initial values of the signals
```

```
    // !ar1 !ar2 !br1 !br2 !aa !ba !w0
endmodule
```

B.4 Complete DPWM

B.4.1 DPWM

```
module DPWM_BEST ( LoadAck, Q, a0, a1, a2, a3, a4, b0, b1, b2, b3, b4,
     cn0, cn1, cn2, fine_mode, LoadReq, rst );

output  LoadAck, Q;

input  a0, a1, a2, a3, a4, b0, b1, b2, b3, b4, cn0, cn1, cn2,
     fine_mode, LoadReq, rst;

Complete_Counter I19 ( .LoadAck(net51), .LoadReq(LoadReq), .reset(rst),
     .d0(b0), .d1(b1), .d2(b2), .d3(b3), .d4(b4), .ar(net43),
     .aa(net055), .br(net58), .ba(net050));
Complete_Counter I2 ( .LoadAck(net50), .LoadReq(LoadReq), .reset(rst),
     .d0(a0), .d1(a1), .d2(a2), .d3(a3), .d4(a4), .ar(net56),
     .aa(net055), .br(net57), .ba(net050));
celement I25 ( .c(LoadAck), .a(net50), .b(net51));
new_LatchSync I32 ( .S(Q), .R(net041), .br1(net65), .br2(net47));
ar_br_sync I33 ( .br1(net65), .br2(net47), .ar1(net56), .ar2(net43),
     .aa(net055), .ba(net050));
delay_block3 I24 ( .MD(fine_mode), .cntrl0(cn0), .cntrl1(cn1),
     .cntrl2(cn2), .in(net58), .op(net47));
```

```
delay_block2 I23 ( .cntrl0(cn0), .cntrl1(cn1), .cntrl2(cn2),
     .in(net57), .op(net65));


endmodule
```

References

[1] Workcraft. URL http://www.workcraft.org/.

[2] J. Kwak and B. Nikolić. A self-adjustable clock generator with wide dynamic range in 28 nm fdsoi. *IEEE Journal of Solid-State Circuits*, 51(10):2368–2379, Oct 2016. ISSN 0018-9200. doi: 10.1109/JSSC.2016.2582860.

[3] J. Tschanz, N. S. Kim, S. Dighe, J. Howard, G. Ruhl, S. Vangal, S. Narendra, Y. Hoskote, H. Wilson, C. Lam, M. Shuman, C. Tokunaga, D. Somasekhar, S. Tang, D. Finan, T. Karnik, N. Borkar, N. Kurd, and V. De. Adaptive frequency and biasing techniques for tolerance to dynamic temperature-voltage variations and aging. In *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, pages 292–604, Feb 2007. doi: 10.1109/ISSCC.2007.373409.

[4] K. A. Bowman, C. Tokunaga, J. W. Tschanz, A. Raychowdhury, M. M. Khellah, B. M. Geuskens, S. L. L. Lu, P. A. Aseron, T. Karnik, and V. K. De. All-digital circuit-level dynamic variation monitor for silicon debug and adaptive clock control. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 58(9):2017–2025, Sept 2011. ISSN 1549-8328. doi: 10.1109/TCSI.2011.2163893.

[5] M. Shirasgaonkar, R. Vu, D. Dressler, N. Nguyen, K. Kaviani, and Y. Wang. An adaptive body-biased clock generation system in 28nm cmos. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, pages 580–583, Jan 2014. doi: 10.1109/VLSID.2014.107.

[6] T. Lin, K. S. Chong, J. S. Chang, and B. H. Gwee. An ultra-low power asynchronous-logic in-situ self-adaptive v_{rmDD} system for wireless sensor networks. *IEEE Journal of Solid-State Circuits*, 48(2):573–586, Feb 2013. ISSN 0018-9200. doi: 10.1109/JSSC.2012.2223971.

[7] K. A. Bowman, J. W. Tschanz, S. L. L. Lu, P. A. Aseron, M. M. Khellah, A. Raychowdhury, B. M. Geuskens, C. Tokunaga, C. B. Wilkerson, T. Karnik, and V. K. De. A 45 nm resilient microprocessor core for dynamic variation tolerance. *IEEE Journal of Solid-State Circuits*, 46(1):194–208, Jan 2011. ISSN 0018-9200. doi: 10.1109/JSSC.2010.2089657.

[8] K. A. Bowman, J. W. Tschanz, N. S. Kim, J. C. Lee, C. B. Wilkerson, S. L. L. Lu, T. Karnik, and V. K. De. Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance. *IEEE Journal of Solid-State Circuits*, 44(1):49–63, Jan 2009. ISSN 0018-9200. doi: 10.1109/JSSC.2008.2007148.

[9] J. Tschanz, N. S. Kim, S. Dighe, J. Howard, G. Ruhl, S. Vangal, S. Narendra, Y. Hoskote, H. Wilson, C. Lam, M. Shuman, C. Tokunaga, D. Somasekhar, S. Tang, D. Finan, T. Karnik, N. Borkar, N. Kurd, and V. De. Adaptive frequency and biasing techniques for tolerance to dynamic temperature-voltage variations and aging. In *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, pages 292–604, Feb 2007. doi: 10.1109/ISSCC.2007.373409.

[10] K. Chae and S. Mukhopadhyay. A dynamic timing error prevention technique in pipelines with time borrowing and clock stretching. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(1):74–83, Jan 2014. ISSN 1549-8328. doi: 10.1109/TCSI.2013.2268272.

[11] M. Bradley, E. Alarcon, and O. Feely. Analysis of limit cycles in a pi digitally controlled buck converter. In *2012 IEEE International Symposium on Circuits and Systems*, pages 628–631, May 2012. doi: 10.1109/ISCAS.2012.6272110.

[12] I. E. Sutherland. Micropipelines. *Commun. ACM*, 32(6):720–738, June 1989. ISSN 0001-0782. doi: 10.1145/63526.63532. URL http://doi.acm.org/10.1145/63526.63532.

[13] C. Vezyrtzis, Y. Tsividis, and S. M. Nowick. Designing pipelined delay lines with dynamically-adaptive granularity for low-energy applications. In *2012 IEEE 30th International Conference on Computer Design (ICCD)*, pages 329–336, Sept 2012. doi: 10.1109/ICCD.2012.6378660.

[14] JLW Kessels. Designing counters with bounded response time. *CS Scholten Dedicata: van oude machines en nieuwe rekenwijzen, Academic Service, Schoonhoven*, pages 127–140, 1990.

[15] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Synthesizing petri nets from state-based models. In *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, pages 164–171, Nov 1995. doi: 10.1109/ICCAD.1995.480008.

[16] D. Wist, M. Schaefer, W. Vogler, and R. Wollowski. Signal transition graph decomposition: internal communication for speed independent circuit implementation. *IET Computers Digital Techniques*, 5(6):440–451, November 2011. ISSN 1751-8601. doi: 10.1049/iet-cdt.2010.0162.

[17] D. Shang, O. Benafa, F. Xia, Y. Xu, and A. Yakovlev. An elastic timer for wide dynamic working range. In *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, pages 1–4, June 2015. doi: 10.1109/NEWCAS.2015.7182004.

[18] O. Benafa, A. Ogweno, D. Shang, and A. Yakovlev. Design of a dco based on worst-case delay of a self-timed counter and a digitally controllable delay path. In *2016 14th IEEE International New Circuits and Systems Conference (NEWCAS)*, pages 1–4, June 2016. doi: 10.1109/NEWCAS.2016.7604821.

[19] D. Harris and S. Naffziger. Statistical clock skew modeling with data delay variations. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(6):888–898, Dec 2001. ISSN 1063-8210. doi: 10.1109/92.974902.

[20] Z. Lak and N. Nicolici. On using on-chip clock tuning elements to address delay degradation due to circuit aging. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(12):1845–1856, Dec 2012. ISSN 0278-0070. doi: 10.1109/TCAD.2012.2209883.

[21] H. Xu, V. F. Pavlidis, X. Tang, W. Burleson, and G. De Micheli. Timing uncertainty in 3-d clock trees due to process variations and power supply noise. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(12):2226–2239, Dec 2013. ISSN 1063-8210. doi: 10.1109/TVLSI.2012.2230035.

[22] W. Rim, W. Choi, and J. Park. Adaptive clock generation technique for variation-aware subthreshold logics. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 59 (9):587–591, Sept 2012. ISSN 1549-7747. doi: 10.1109/TCSII.2012.2206933.

[23] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. Reducing power in high-performance microprocessors. In *Proceedings 1998 Design and Automation Conference. 35th DAC. (Cat. No.98CH36175)*, pages 732–737, June 1998. doi: 10.1145/277044.277227.

[24] R. Bhutada and Y. Manoli. Complex clock gating with integrated clock gating logic cell. In *2007 International Conference on Design Technology of Integrated Systems in Nanoscale Era*, pages 164–169, Sept 2007. doi: 10.1109/DTIS.2007.4449512.

[25] S. H. Yang, C. C. Chiu, C. W. Chang, C. M. Chen, C. H. Meng, and K. H. Chen. 87% Overall High Efficiency and 11μA Ultra-Low Standby Current Derived by Overall Power Management in Laptops With Flexible Voltage Scaling and Dynamic Voltage Scaling Techniques. *IEEE Transactions on Power Electronics*, 31(4):3118–3127, April 2016. ISSN 0885-8993. doi: 10.1109/TPEL.2015.2450746.

[26] V. Hanumaiah and S. Vrudhula. Energy-efficient operation of multicore processors by dvfs, task migration, and active cooling. *IEEE Transactions on Computers*, 63(2): 349–360, Feb 2014. ISSN 0018-9340. doi: 10.1109/TC.2012.213.

[27] L. S. Nielsen and J. Sparso. Designing asynchronous circuits for low power: an ifir filter bank for a digital hearing aid. *Proceedings of the IEEE*, 87(2):268–281, Feb 1999. ISSN 0018-9219. doi: 10.1109/5.740020.

[28] S. M. Nowick and M. Singh. Asynchronous design—part 1: Overview and recent advances. *IEEE Design Test*, 32(3):5–18, June 2015. ISSN 2168-2356. doi: 10.1109/MDAT.2015.2413759.

[29] C. H. Van Berkel, M. B. Josephs, and S. M. Nowick. Applications of asynchronous circuits. *Proceedings of the IEEE*, 87(2):223–233, Feb 1999. ISSN 0018-9219. doi: 10.1109/5.740016.

[30] H. S. Low, D. Shang, F. Xia, and A. Yakovlev. Variation tolerant afpga architecture. In *2011 17th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 77–86, April 2011. doi: 10.1109/ASYNC.2011.17.

[31] J. F. Chappel and S. G. Zaky. Emi effects and timing design for increased reliability in digital systems. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 44(2):130–142, Feb 1997. ISSN 1057-7122. doi: 10.1109/81.554331.

[32] Jens Spars and Steve Furber. *Principles of Asynchronous Circuit Design*. Springer, 2002.

[33] A. D. Bailey, Jia Di, S. C. Smith, and H. A. Mantooth. Ultra-low power delay-insensitive circuit design. In *2008 51st Midwest Symposium on Circuits and Systems*, pages 503–506, Aug 2008. doi: 10.1109/MWSCAS.2008.4616846.

[34] M. Renaudin. Asynchronous circuits and systems: A promising design alternative. *Microelectron. Eng.*, 54(1-2):133–149, December 2000. ISSN 0167-9317. doi: 10.1016/S0167-9317(00)80065-9. URL http://dx.doi.org/10.1016/S0167-9317(00)80065-9.

[35] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr 1989. ISSN 0018-9219. doi: 10.1109/5.24143.

[36] D. Sokolov, A. Mokhov, A. Yakovlev, and D. Lloyd. Towards asynchronous power management. In *2014 IEEE Faible Tension Faible Consommation*, pages 1–4, May 2014. doi: 10.1109/FTFC.2014.6828608.

[37] Jo C Ebergen and Ad MG Peeters. Design and analysis of delay-insensitive modulo-n counters. *Formal Methods in System Design*, 3(3):211–232, 1993.

[38] D. Sokolov and A. Yakovlev. Clockless circuits and system synthesis. *IEE Proceedings - Computers and Digital Techniques*, 152(3):298–316, May 2005. ISSN 1350-2387. doi: 10.1049/ip-cdt:20045094.

[39] V. Khomenko. Efficient automatic resolution of encoding conflicts using stg unfoldings. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(7):855–868, July 2009. ISSN 1063-8210. doi: 10.1109/TVLSI.2008.2012156.

[40] Teak. URL http://apt.cs.manchester.ac.uk/projects/teak/.

[41] M. J. Mamaghani, J. Garside, and D. Edwards. De-elastisation: From asynchronous dataflows to synchronous circuits. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 273–276, March 2015. doi: 10.7873/DATE.2015.0759.

[42] A. Bardsley, L. Tarazona, and D. Edwards. Teak: A token-flow implementation for the balsa language. In *2009 Ninth International Conference on Application of Concurrency to System Design*, pages 23–31, July 2009. doi: 10.1109/ACSD.2009.15.

[43] Joep Kessels and Ad Peeters. The tangram framework.

[44] Kees van Berkel, Joep Kessels, Marly Roncken, Ronald Saeijs, and Frits Schalij. The vlsi-programming language tangram and its translation into handshake circuits. In *Proceedings of the conference on European design automation*, pages 384–389. IEEE Computer Society Press, 1991.

[45] CH Van Berkel, M Rem, and RWJJ Saeijs. Vlsi programming. In *Computer Design: VLSI in Computers and Processors, 1988. ICCD'88., Proceedings of the 1988 IEEE International Conference on*, pages 152–156. IEEE, 1988.

[46] K. van Berkel, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij. The vlsi-programming language tangram and its translation into handshake circuits. In *Proceedings of the European Conference on Design Automation.*, pages 384–389, Feb 1991.

[47] Charles Antony Richard Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978. doi: 10.1145/359576.359585. URL http://doi.acm.org/10.1145/359576.359585.

[48] Maitham Shams, Jo C. Ebergen, and Mohammed I. Elmasry. Asynchronous circuits.

[49] Otto Aureliano Rolloff, Rodrigo Possamai Bastos, and Laurent Fesquet. Exploiting reliable features of asynchronous circuits for designing low-voltage components in fd-soi technology. *Microelectronics Reliability*, 55(9):1302 – 1306, 2015. ISSN 0026-2714. doi: https://doi.org/10.1016/j.microrel.2015.07.028. URL http://www.sciencedirect.com/science/article/pii/S0026271415301141. Proceedings of the 26th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis.

[50] J. Kessels and P. Marston. Designing asynchronous standby circuits for a low-power pager. *Proceedings of the IEEE*, 87(2):257–267, Feb 1999. ISSN 0018-9219. doi: 10.1109/5.740019.

[51] E. Beigne, P. Vivet, Y. Thonnart, J. F. Christmann, and F. Clermidy. Asynchronous circuit designs for the internet of everything: A methodology for ultralow-power circuits with gals architecture. *IEEE Solid-State Circuits Magazine*, 8(4):39–47, Fall 2016. ISSN 1943-0582. doi: 10.1109/MSSC.2016.2573864.

[52] J. F. Christmann, E. Beigné, C. Condemine, J. Willemin, and C. Piguet. Energy harvesting and power management for autonomous sensor nodes. In *DAC Design Automation Conference 2012*, pages 1049–1054, June 2012. doi: 10.1145/2228360.2228550.

[53] A. Ogweno, A. Yakovlev, and P. Degenaar. Power gating in asynchronous micropiplines for low power data driven computing. In *2015 11th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*, pages 342–345, June 2015. doi: 10.1109/PRIME.2015.7251405.

[54] A. Ogweno, P. Degenaar, V. Khomenko, and A. Yakovlev. A fixed window level crossing adc with activity dependent power dissipation. In *2016 14th IEEE International New Circuits and Systems Conference (NEWCAS)*, pages 1–4, June 2016. doi: 10.1109/NEWCAS.2016.7604815.

[55] C. Weltin-Wu and Y. Tsividis. An event-driven clockless level-crossing adc with signal-dependent adaptive resolution. *IEEE Journal of Solid-State Circuits*, 48(9): 2180–2190, Sept 2013. ISSN 0018-9200. doi: 10.1109/JSSC.2013.2262738.

[56] Chi-Hsiang Yeh, B. Parhami, and Y. Wang. Designs of counters with near minimal counting/sampling period and hardware complexity. In *Conference Record of the Thirty-Fourth Asilomar Conference on Signals, Systems and Computers (Cat. No.00CH37154)*, volume 2, pages 894–898 vol.2, Oct 2000. doi: 10.1109/ACSSC.2000.910642.

[57] C. Christakis, G. Theodoridis, and A. Kakarountas. High speed binary counter based on 1d cellular automata. In *2016 5th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pages 1–4, May 2016. doi: 10.1109/MOCAST.2016.7495170.

[58] Chi-Hsiang Yeh, B. Parhami, and Y. Wang. Designs of counters with near minimal counting/sampling period and hardware complexity. In *Conference Record of the Thirty-Fourth Asilomar Conference on Signals, Systems and Computers (Cat. No.00CH37154)*, volume 2, pages 894–898 vol.2, Oct 2000. doi: 10.1109/ACSSC.2000.910642.

[59] T. Zhang and Q. Hu. A high-speed and low-power up/down counter in 0.18μm cmos technology. In *2012 International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–3, Oct 2012. doi: 10.1109/WCSP.2012.6542792.

[60] J. E. Vuillemin. Constant time arbitrary length synchronous binary counters. In *[1991] Proceedings 10th IEEE Symposium on Computer Arithmetic*, pages 180–183, Jun 1991. doi: 10.1109/ARITH.1991.145556.

[61] S. Abdel-Hafeez and A. Gordon-Ross. A digital cmos parallel counter architecture based on state look-ahead logic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(6):1023–1033, June 2011. ISSN 1063-8210. doi: 10.1109/TVLSI.2010.2044818.

[62] S. Abdel-Hafeez, S. M. Harb, and W. R. Eisenstadt. High speed digital cmos divide-by-n fequency divider. In *2008 IEEE International Symposium on Circuits and Systems*, pages 592–595, May 2008. doi: 10.1109/ISCAS.2008.4541487.

[63] M. Ercegovac and T. Lang. Binary counter with counting period of one half adder independent of counter size. *IEEE Transactions on Circuits and Systems*, 36(6): 924–926, Jun 1989. ISSN 0098-4094. doi: 10.1109/31.90421.

[64] J. E. Vuillemin. Constant time arbitrary length synchronous binary counters. In *[1991] Proceedings 10th IEEE Symposium on Computer Arithmetic*, pages 180–183, Jun 1991. doi: 10.1109/ARITH.1991.145556.

[65] D. R. Lutz and D. N. Jayasimha. Programmable modulo-k counters. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 43(11):939–941, Nov 1996. ISSN 1057-7122. doi: 10.1109/81.542285.

[66] M. R. Stan. Systolic counters with unique zero state. In *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*, volume 2, pages II–909–12 Vol.2, May 2004. doi: 10.1109/ISCAS.2004.1329420.

[67] J. B. Hughes, E. S. Eilley, and G. J. Glynn. Modulo-n counter technique for the u.h.f. band. *Electronics Letters*, 15(20):640–641, September 1979. ISSN 0013-5194. doi: 10.1049/el:19790455.

[68] B. Parhami. Systolic up/down counters with zero and sign detection. In *1987 IEEE 8th Symposium on Computer Arithmetic (ARITH)*, pages 174–178, May 1987. doi: 10.1109/ARITH.1987.6158710.

[69] Jo C. Ebergen, John Segers, and Igor Benko. *Parallel Program and Asynchronous Circuit Design*, pages 50–103. Springer London, London, 1995. ISBN 978-1-4471-3575-3.

[70] Jo C Ebergen and Ad MG Peeters. Design and analysis of delay-insensitive modulo-n counters. *Formal Methods in System Design*, 3(3):211–232, 1993.

[71] John Segers and Jo C. Ebergen. Design and analysis of up-down counters. *Science of Computer Programming*, 27(2):185 – 204, 1996. ISSN 0167-6423.

[72] Joep L. W. Kessels. Calculational derivation of a counter with bounded response time and bounded power dissipation. *Distributed Computing*, 8(3):143–149, Mar 1995. doi: 10.1007/BF02242716.

[73] M.A. Kishinevsky, A.Yu. Kondratyev, and A.R. Taubin. Formal design of control circuits based on behavioral "circuit assembler" (change diagrams). In *Proc. ESPRIT ACiD-WG Workshop on Asynchronous Controllers and Interfacing*, September 1992.

[74] A.Yu. Kondratyev. A proposal for the specified modulo-n-counter. In *Proc. ESPRIT ACiD-WG Workshop on Asynchronous Controllers and Interfacing*, September 1992.

[75] O. Aumann and C. Heer. Formal design of control circuits based on behavioral "circuit assembler" (change diagrams). In *Proc. ESPRIT ACiD-WG Workshop on Asynchronous Controllers and Interfacing*, September 1992.

[76] Kees Van Berkel. *Handshake circuits: an asynchronous architecture for VLSI programming*, volume 5. Cambridge University Press, 1993.

[77] Kees van Berkel and Martin Rem. Vlsi programming of asynchronous circuits for low power. In *Asynchronous Digital Circuit Design*, pages 151–210. Springer, 1995.

[78] Re F. Tenca, Milos Ercegovac, and Mircea Stan. Synchronous up/down binary counter for lut fpgas with counting frequency independent of counter size. 05 2000.

[79] M. R. Stan. Synchronous up/down counter with clock period independent of counter size. In *Proceedings 13th IEEE Sympsoium on Computer Arithmetic*, pages 274–281, Jul 1997. doi: 10.1109/ARITH.1997.614905.

[80] K. Z. Pekmestzi and N. Thanasouras. Systolic frequency dividers/counters. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 41 (11):775–776, Nov 1994. ISSN 1057-7130. doi: 10.1109/82.331551.

[81] A. Yakovlev. Solving acid-wg design problems with petri net based methods, groningen. In *Proc. ESPRIT ACiD-WG Workshop on Asynchronous Circuit Design*, September 1996.

[82] Roger Sayle. On the synthesis of modulo$-n$ counters. In *Proc. ESPRIT ACiD-WG Workshop on Asynchronous Controllers and Interfacing*, September 1992.

[83] S. Das, D. Roberts, Seokwoo Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. A self-tuning dvs processor using delay-error detection and correction. *IEEE Journal of Solid-State Circuits*, 41(4):792–804, April 2006. ISSN 0018-9200. doi: 10.1109/JSSC.2006.870912.

[84] Chien-Ying Yu, Ching-Che Chung, Chia-Jung Yu, and Chen-Yi Lee. A low-power dco using interlaced hysteresis delay cells. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 59(10):673–677, Oct 2012. ISSN 1549-7747. doi: 10.1109/TCSII.2012.2213357.

[85] M. Maymandi-Nejad and M. Sachdev. A digitally programmable delay element: design and analysis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(5):871–878, Oct 2003. ISSN 1063-8210. doi: 10.1109/TVLSI.2003.810787.

[86] F. Baronti, D. Lunardini, R. Roncella, and R. Saletti. A self-calibrating delay-locked delay line with shunt-capacitor circuit scheme. *IEEE Journal of Solid-State Circuits*, 39(2):384–387, Feb 2004. ISSN 0018-9200. doi: 10.1109/JSSC.2003.821773.

[87] Shu-Yu Hsu, Jui-Yuan Yu, and Chen-Yi Lee. A sub-10-μw digitally controlled oscillator based on hysteresis delay cell topologies for wban applications. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 57(12):951–955, Dec 2010. ISSN 1549-7747.

[88] Duo Sheng, Ching-Che Chung, and Chen-Yi Lee. An ultra-low-power and portable digitally controlled oscillator for soc applications. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 54(11):954–958, Nov 2007. ISSN 1549-7747. doi: 10.1109/TCSII.2007.903782.

[89] Seok Min Jung and Janet Meiling Roveda. Design of a feedback digitally controlled oscillator for linearity enhancement. In *Electron Devices and Solid-State Circuits (EDSSC), 2015 IEEE International Conference on*, pages 277–280, June 2015. doi: 10.1109/EDSSC.2015.7285104.

[90] R. Ramezani, A. Yakovlev, F. Xia, J. P. Murphy, and D. Shang. Voltage sensing using an asynchronous charge-to-digital converter for energy-autonomous environments. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 3(1):35–44, March 2013. ISSN 2156-3357. doi: 10.1109/JETCAS.2013.2242776.

[91] Reza Ramezani. *Design, Analysis and Implementation of Voltage Sensor for Power-Constrained Systems*. PhD thesis, Newcastle University, 2014.

[92] O. Benafa, D. Sokolov, and A. Yakovlev. Loadable self-timed counter: Decomposition, specification and implementation. *Submitted to International Symposium on Asynchronous Circuits and Systems (ASYNC)*, May 2018.

[93] V. Khomenko, M. Koutny, and A. Yakovlev. Detecting state coding conflicts in stg unfoldings using sat. In *Application of Concurrency to System Design, 2003. Proceedings. Third International Conference on*, pages 51–60, June 2003. doi: 10.1109/CSD.2003.1207699.

[94] I. Miyazawa, Y. Itoh, and T. Sekiguchi. A result on the relationship between petri net and directed graph-real time fault diagnosis based on petri net model. In *Industrial Electronics Society, 1998. IECON '98. Proceedings of the 24th Annual Conference of the IEEE*, volume 1, pages 126–131 vol.1, Aug 1998. doi: 10.1109/IECON.1998.723957.

[95] V. Khomenko, A. Madalinski, and A. Yakovlev. Resolution of encoding conflicts by signal insertion and concurrency reduction based on stg unfoldings. In *Sixth International Conference on Application of Concurrency to System Design (ACSD'06)*, pages 57–68, June 2006. doi: 10.1109/ACSD.2006.21.

[96] A. V. Peterchev and S. R. Sanders. Quantization resolution and limit cycling in digitally controlled pwm converters. *IEEE Transactions on Power Electronics*, 18(1): 301–308, Jan 2003. ISSN 0885-8993. doi: 10.1109/TPEL.2002.807092.

[97] H. Peng, A. Prodic, E. Alarcon, and D. Maksimovic. Modeling of quantization effects in digitally controlled dc ndash;dc converters. *IEEE Transactions on Power Electronics*, 22(1):208–215, Jan 2007. ISSN 0885-8993. doi: 10.1109/TPEL.2006.886602.

[98] Y. F. Liu, E. Meyer, and X. Liu. Recent developments in digital control strategies for dc/dc switching power converters. *IEEE Transactions on Power Electronics*, 24(11): 2567–2577, Nov 2009. ISSN 0885-8993. doi: 10.1109/TPEL.2009.2030809.

[99] Y. Sun, V. Adrian, and J. S. Chang. Design of a variable-delay window adc for switched-mode dc-dc converters. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1642–1645, May 2015. doi: 10.1109/ISCAS.2015.7168965.

[100] S. Höppner, S. Haenzsche, S. Scholze, and R. Schüffny. An all-digital pwm generator with 62.5ps resolution in 28nm cmos technology. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1738–1741, May 2015. doi: 10.1109/ISCAS.2015.7168989.

[101] W. W. Wang, Z. H. Shen, X. Tan, N. Yan, and H. Min. Improved delay-line based digital pwm for dc-dc converters. *Electronics Letters*, 47(9):562–564, April 2011. ISSN 0013-5194. doi: 10.1049/el.2011.0095.

[102] L. Corradini, A. Bjeletić, R. Zane, and D. Maksimović. Fully digital hysteretic modulator for dc-dc switching converters. *IEEE Transactions on Power Electronics*, 26(10):2969–2979, Oct 2011. ISSN 0885-8993. doi: 10.1109/TPEL.2010.2055244.

[103] Y. W. Kim, J. H. Kim, K. Y. Choi, B. S. Suh, and R. Y. Kim. A novel soft-switched auxiliary resonant circuit of a pfc zvt-pwm boost converter for an integrated multichip power module fabrication. *IEEE Transactions on Industry Applications*, 49(6):2802–2809, Nov 2013. ISSN 0093-9994. doi: 10.1109/TIA.2013.2265074.

[104] X. Zhang and A. B. Apsel. A low variation ghz ring oscillator with addition-based current source. In *2009 Proceedings of the European Solid State Device Research Conference*, pages 233–236, Sept 2009. doi: 10.1109/ESSDERC.2009.5331514.