THEORY, IMPLEMENTATIONS AND APPLICATIONS OF SINGLE-TRACK DESIGNS

by

Pankaj Pankaj

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY

(ELECTRICAL ENGINEERING)

May 2010

Copyright 2010 Pankaj Pankaj

Dedication

To my family

Acknowledgements

It is my good fortune to have Dr. Peter Beerel as my dissertation advisor. He has been an unfailing source of inspiration and encouragement to me. In his conduct, I have found an ideal to aspire for. He has given generously of his time and his ideas. It is difficult for me to imagine a better advisor. He and his lovely family created an extremely friendly and conducive environment suitable for exchange of ideas.

I will also like to thank Dr. Ken Stevens for generously giving me time and introducing me to the new paradigms of asynchronous circuit design. He has been an excellent teacher and has helped me build a strong technical understanding.

I am grateful to my thesis committee members Dr. Sandeep Gupta, Dr. Massoud Pedram, Dr. Jeff Draper and Dr. Ivan Kukavica for taking out time from their busy schedules to entertain my questions and giving me valuable feedback. I also thank the EE-systems staff, especially Annie Yu, Diane Demetras and Tim Boston for guiding me through the various requirements of the EE department at USC.

I would like to thank my friends from the USC Asynchronous CAD / VLSI group, Nam-hoon Kim, Arash Saifhashemi, Mallika Prakash, Prasad Joshi, Amit Bandlish, Roger Su, Gokul Govindu, Rahul Rithe and Ritej Bacchhawat for all these years of endless discussions and their support, comments and invaluable suggestions.

I thank Nand Kishore Jha, Sudip Shekhar, Aashish Prakash, Anurag Jain, Neha Jha, Akshay Kedia, Sunil Narang, Chiranjeeb Chaudhary, Subhankar Ghosh, Anuj Madaria, Arvind Pereira, Nivedita Singh, Praveen Kumar, Manoj Gopalkrishnan, Adarsh Shekhar, Joyita Dutta, and all others. They have been wonderful friends. I am grateful for their support through times of sorrow and joy, despair and aspiration.

I wish to thank my parents, my brother and sister-in-law for their unconditional love and support. They have been the pillar of my strength without which this dissertation would not have been possible. They have worked extremely hard to provide me with a great platform for succeeding in life. I dedicate this thesis to them.

Table of Contents

List of Tables

List of Figures

Abstract

Asynchronous design is increasingly becoming an attractive alternative to synchronous design because of its potential for high performance, low power, and adaption to process variations and reduced EMI noise. However to compete with the existing synchronous designs there is a need to reduce the overwhelming design time of asynchronous designs by developing circuits that can easily be verified along with mature CAD flows to support them. In this thesis we propose to use template based methodology which facilitates standard cell based design that can easily be verified and allows using standard ASIC tools that have already been developed by semiconductor industry.

In this thesis we develop three novel non homogeneous single-track templates, including theory, CAD, and design examples. These static single-track templates follow a two phase static single-track handshake protocol and provide better performance than four phase asynchronous designs and better forward latency than asynchronous bundled data and synchronous designs. Compared with earlier homogeneous versions, non homogeneous single-track templates have the flexibility of having multiple levels of logic which helps in reducing the control area overhead by sharing the control logic among multiple levels of logic. These templates are more complex however and warrant more automation.

To quantify some of these advantages, an asynchronous turbo decoder was developed using a single-track standard cell library in IBM 0.18 µm technology and compared against a synchronous turbo decoder. Comparisons shows that asynchronous turbo decoder can provide 1.3X – 2X improvement in throughput per area over the synchronous version for block sizes of 2K – 768 bits. Moreover, due to its low latency advantages, it can support smaller block sizes at higher throughputs than possible using synchronous design.

To help automate such designs, we propose a CAD flow to synthesize single-track asynchronous design by extending an existing CAD tool called Proteus. Comparisons on ISCAS benchmarks shows that the proposed templates provide on an average 30% improvement in throughput per area over QDI templates and 75% improvement over MLD templates. We also demonstrate library characterization and SDF back annotation flow on a single-track standard cell library. Experimental results on a 64 bit prefix adder design indicates that the back annotation flow yields over two orders of magnitude advantage in simulation speed on analog verification flow with less than 5% error.

Finally we demonstrate the low latency and high performance advantages of single-track protocol over other asynchronous and synchronous communication protocols by comparing their throughput, latency, energy and bandwidth for a NoC interconnect link.

1 Introduction

Synchronous designs have been dominating the market of VLSI and ASIC design for many years. However as the VLSI industry follows Moore's law where the number of transistors that can be placed in a chip is increasing in an exponential manner, the challenges of designing circuits with a global clock has been increasing. As global wiring delays becomes more dominant compared to gate delays [18] distribution of the clock across a large design while minimizing the clock skew is becoming a challenge. This is particularly challenging in the sub-45nm regime where *process variation* is a significant issue. More specifically, as transistor dimensions continue to shrink, it becomes successively harder to precisely control the fabrication process. Due to process fabrication imperfections, different transistors on the same chip exhibit different values of parameters such as threshold voltage or effective channel length. These parameters in turn determine the switching speed and leakage of transistors, which are also subject to substantial fluctuation. Variation in transistor switching speed leads to slowing down of some unit in a pipeline which leads to the complete design operating at a lower frequency than normal. Variation is already forcing designers to put more margins in the designs and Bowman *et al.* suggest that variation may wipe out the performance gains of a full technology generation [16].

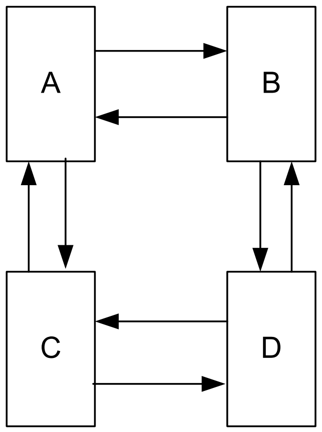


Figure 1 Asynchronous blocks communicating using handshake signals.

In the absence of a global clock that controls register and state updating, asynchronous designs rely on handshaking to transfer data between functional blocks as shown in Figure 1. While academic research in this area can be traced back to the 1950s, it has taken until the late 1990s and 2000s for this technology to mature. Several start-up companies have commercialized asynchronous designs for a wide variety of applications [1][33][60][40]. Asynchronous designs have demonstrated potential benefits in many aspects of system design such as Amulet2e[34], ARM996HS[13], ASPRO16[58], MiniMIPS[51], Asynchronous FPGA[74]. In particular asynchronous NoC[60] where systems can accommodate multiple clocking islands that many of today's complex SoC designs use. Asynchronous communication methods can provide numerous advantages in Network-on-a-chip (NoC)[6][60] as each module can be designed for its best frequency and power. Moreover the ability of the asynchronous designs to adapt themselves to physical properties leads to more robust design. Ken Stevens in [68] compares synchronous and asynchronous communication protocols used to implement interconnect links in NoC by creating parameterized throughput, energy and bandwidth models.

Asynchronous circuits have the following advantages:

1. **Absence of clock skew**. Clock skew is defined as the arrival time difference of the clock signal to different parts of the circuit. In synchronous design the clock period is increased to ensure correct operation in the presence of clock skew affecting the performance of the design. Moreover due to extra margin in the clock period to account for the worst case clock skew the latency of the system suffers. On the other hand due to absence of a global synchronization signal, asynchronous designs do not face such problems.

2. **Average-case performance**: In synchronous systems the clock period is dictated by the slowest stage in worst case conditions to ensure correct operation. However asynchronous designs can have average-case delay due to data-dependent flow and/or functional units that exhibit data-dependent delay. In both cases, the average-case delay may be faster than the synchronous worst-case delay.

3. **Automatic adaptation to physical properties**: To combat process variations synchronous designs have to put up extra margins to avoid any timing violations thus degrading the performance and latency of the design. Many asynchronous circuits can robustly adapt to changing conditions, yielding improved performance [51]. This is far more difficult to do in a synchronous design, as the variations can be local, whereas the impact on the clock is far more global.

4. **Potential for low power**: Due to constant activity of the clock signal synchronous designs consume power even though the design is idle. Even though methods such as clock gating where clock signal is disabled in parts of the designs which are idle power is still consumed in clock distribution networks which is responsible to distribute clock to all parts of the chip. Studies have shown that clock distribution network can dissipate 20-50% of the total power on the chip [55]. Due to its data-driven nature asynchronous circuits provides ideal clock gating and there is no requirement of clock distribution networks in asynchronous designs. However, these power advantages are not universally true among all asynchronous design styles. In particular, some asynchronous circuits designed for high-performance have more average transitions per data bit than comparable synchronous designs, due to dual-rail or other multi-rail data encoding and/or completion detection logic. However these circuits provide distinct advantage in terms of $E\tau^2$ [45] and the power consumption of these circuits can be reduced by operating them at a lower voltage.

5. **Reduced electromagnetic interference**: In a synchronous design, all activity is locked into a very precise frequency. The result is that nearly all the energy is concentrated in very narrow spectral bands around the clock frequency and its harmonics. Therefore, there is substantial electromagnetic noise at these frequencies that can adversely affect neighboring analog circuits. Activity in an asynchronous circuit is uncorrelated, resulting in a more distributed noise spectrum and lower peak noise [48] [34].

Despite all the above mentioned advantages and examples demonstrated by asynchronous research community, semiconductor industry is still skeptical about asynchronous designs. This is because while synchronous design techniques has matured over time and are widely known, asynchronous design techniques are still new and are complex in nature. Chips with high volumes such as microprocessors, memory, and FPGAs may be able to support full-custom techniques with advanced circuit styles, such as asynchronous design. In fact, asynchronous techniques have been used in memory for years and a recent start-up is commercializing high-speed FPGAs enabled by high-speed asynchronous circuits [1]. For asynchronous designs to be adopted widespread in semi custom ASIC designs there is a need to develop asynchronous design techniques which are simpler in nature and can be easily verified along with enhancing existing CAD tool suites and developing new tools to support asynchronous circuits.

1.1 Template based asynchronous designs

Among the numerous asynchronous design styles being developed template-based design styles have demonstrated numerous advantages in performance and design time. These template-based designs follow standard ASIC design flows which helps a designer in creating push-play designs thus reducing the design times [32]. The template based design approach reduces the design complexity by dividing the large system into leaf cells that are the smallest components that communicate with its neighbors through asynchronous handshaking. This decomposition of a large system into smaller leaf cells results in simple designs which can be verified easily. Template based designs have the key advantage being that instead of finding the timing constraints of a large variety of unique controllers which is quite error prone [81], the timing constraints of template based controller are simpler are known before hand and can be captured in standard delay constraint file [44]. The template based design approach also leads to well defined modular and local designs and are well suited to an ASIC flow.

Different template-based designs trade-off robustness to performance and to area [50][61][64][71]. One of the most robust design template is *Quasi Delay Insensitive* (QDI) template proposed by Lines [50]. QDI template based designs uses 1-of-N data encoding and use 4-phase handshaking. On the other side the most aggressive design style in terms of performance is *GasP* proposed by Sutherland [71]. GasP designs use 2-phase single-track handshaking and offers ultra high throughput but are based on bundled data designs where the clock is replaced by local controllers controlling the opening and closing of the flop.

Single Track Full Buffer (STFB) [30] is an asynchronous design template which uses two phase single-track handshake protocol [9] and takes the advantages of both the world. STFB is more robust than GasP, but has more timing constraints compared to QDI designs. The STFB templates use 1-of-N data encoding, faster domino logic and follow 2-phase single-track protocol. One limitation of STFB designs is that after the data transaction the communication wires (channel) are left in tri-state condition, due to which they are susceptible to noise. In order to improve the noise robustness a new design style *Static Single Track Full Buffer (SSTFB)* was also proposed [30] [38].

1.2 Static Single-Track template based designs

SSTFB has similar properties as STFB with the key difference being that it follows static single-track handshake [9] protocol which keeps the communication wires always statically driven. SSTFB has two key advantages over GasP, one that it removes the bundling data constraint which makes the design more robust to process variations and other that it provides better latency than GasP circuits. The key advantage of SSTFB over QDI is that it uses 2-phase handshaking which leads to higher throughput and lower power than QDI based designs.

To demonstrate the advantages of static single-track circuits, a high speed asynchronous turbo decoder is designed as a case study. Turbo decoding [12] is becoming a very popular solution for error correction especially in wireless applications. Another key aspect of this algorithm is that the entire block of data has to be written into the memory before the next iteration on the data can start. As the degree of parallelism is increased the processing time can be linearly reduced, but the pipeline latency remains constant yielding diminishing benefits. Performance degradation is more pronounced in cases with small data block sizes where the pipeline latency is comparable or in extreme situations larger than the actual processing time [24].

The key limitation of SSTFB templates is that they only support homogeneous pipelines which limit a pipeline stage to have single level of logic and a fixed pulse width of 3 on the input and output drivers. The constraint of having only single level of logic makes SSTFB suitable only for high performance applications. For medium to low performance applications due to control overhead associated with each pipeline stage SSTFB designs become area inefficient. The constraint on the fixed pulse width of 3 on drivers constrains the amount of load on the output channel. Moreover for multi output gate there exists a relative timing constraint between the outputs, such that if one output has a very small load compared to other outputs, then the evaluation logic of other outputs may not get sufficient time to generate output tokens inputs leading to failure. These timing constraints can be satisfied by putting a maximum limit on the output wire length as advocated in [30] but for deep sub micron ASIC designs in presence of crosstalk and process variations it is getting increasingly difficult to satisfy.

Even with all the advantages of asynchronous designs discussed above they are still not widely accepted. A key roadblock has been the lack of EDA tools to support the generation and verification of asynchronous designs. In past several HDL based synthesis approaches such as de-synchronization [21], weaving [65], null-convention logic [29], phased logic [49] and syntax directed translation synthesis approach such as Balsa [4], Tangram [46] have been proposed. While Balsa and tangram requires the designer to use a new high level language and require the development of some other tools for simulation and performance verification making HDL based synthesis approaches a more attractive choice. At *USC Asynchronous CAD / VLSI* research group a new HDL based synthesis tool *Proteus* [23] is being developed which converts a RTL synchronous design into an asynchronous design. Some unique features of this tool are that it uses a more accurate model for slack matching [8], and in order to achieve better performance at lower area penalty area optimization techniques like clustering of different pipeline stages and fan out optimization are performed. All of the above mentioned synthesis approaches either target bundled data designs or QDI designs leaving a big void for single-track designs.

1.3 Contributions

In this thesis we offer the following contributions:

- **Latency critical applications – A turbo decoder:** A case study comparing SSTFB and synchronous implementations of a turbo decoder in a 0.18μm technology. Turbo decoding is an interesting application for static single-track circuits because of the iterative nature of the algorithm where in order to achieve a certain decoded data rate the decoder has to run several times faster internally in order to keep up with the data. The case study demonstrates that high performance asynchronous designs using SSTFB circuits provide 1.3X - 4X advantage in throughput per area over their synchronous counterparts in certain processing intensive applications. Moreover, in this particular application due to low latency advantages of SSTFB over synchronous designs, it can support certain information rates which synchronous design cannot support at the same throughput.
- **Theoretical analysis of static single-track handshake circuits:** We propose two constraints on the drivers of processes. The first constraint is single-track handshake constraint which ensures that there

is no fight among communicating processes, and the other constraint is rail-2-rail swing constraint which ensures that the communicating wires (channels) satisfies a user defined noise robustness criteria. These two constraints provide guidelines to design non homogeneous single-track circuits. These constraints can also be used to derive relative timing constraints which can be then later used for STA using standard EDA tools [44].

- **Non homogeneous single-track templates:** Development of three non homogeneous single-track design templates which provides the advantages STFB and SSTFB provides but targets a broader range of pipeline sizes, including medium grained pipelines, leading to typically much lower control and energy overhead, while at the same time providing lower latency and a wider range of throughputs than other existing circuit families.

- **CAD support for single-track circuits**: The contribution of this thesis in CAD support for single-track circuits is as follows:

  - **RTL to single-track asynchronous designs:** In this work we extend Proteus to generate single-track designs. Specifically we have developed prototype to convert single rail asynchronous design to single-track asynchronous design targeting the proposed single-track templates styles. We also develop prototype for test bench generation to verify the generated designs and have modified the slack matching algorithm to include slack matching for two phase asynchronous designs. Using the synthesis flow we verify the advantages of our proposed templates on ISCAS benchmarks along with some other interesting examples. Specifically we compare the area and performance of the three proposed design styles to the industry standard design styles QDI and MLD.

  - **SDF back annotation flow for single-track circuits**: In this work we demonstrate library characterization flow for single-track libraries STFB [30] and SSTFB [38]. Due to two-phase handshaking nature and use of bidirectional wires, timing characterization is challenging for single-track circuits. We identified timing arcs and characterized them in industry standard liberty format. Finally we demonstrate back end SDF back annotation flow on a 260K transistor parallel prefix 64 bit added design [32]. Experimental results demonstrate the proposed back annotation flow yields two orders of magnitude advantage in simulation speed on analog verification flow with less than 5% error [37].

- **NoC interconnect links**: In this work we extend the work done by Ken Stevens in [68] by including two-phase single-track protocol in the comparisons. Additionally we characterize all the communication protocols to derive the value of the parameters used to model throughput, energy and bandwidth and include the effects of clock distribution network on synchronous protocols in our comparison. Comparisons show that two-phase single-track protocol provides the best throughput and latency advantages over all asynchronous protocols. Comparisons also shows that at low bus activity factor for the same bandwidth bundled data and single-track protocols can yield significant advantages over synchronous communication in average energy per transaction.

1.4 Organization

The organization of the remainder of this thesis is as follows. Chapter 2 has been devoted to the various existing asynchronous design styles. In Chapter 3 we derive the theory behind design of non homogeneous single-track circuits. In Chapter 4 design of three new proposed non homogeneous single-track design templates is presented. Chapter 5 presents the integration flow of the single-track templates into Proteus and we compare the throughput per area of the proposed templates against QDI and MLD design templates. Chapter 6 presents the design of the asynchronous turbo decoder. In Chapter 7 we present the parameterized analytical models comparing synchronous and asynchronous communication protocols. Finally we conclude in Chapter 8.

2 Background

2.1 Introduction

In synchronous design, a global clock acts as a synchronizing signal and controls latches and//or flip-flops that surround combinational logic. The clock edge determines when the latches/flops sample data and the clock period is set to guarantee sampled data is valid. In asynchronous designs synchronization is achieved between blocks though handshaking. Asynchronous designs differ among themselves by employing asynchronous channels which differ from each other in type of handshaking protocols, data encoding and logic styles.

This chapter first presents background on asynchronous channels and their data encoding and then presents various asynchronous design styles that are popular among design community.

2.2 Asynchronous channels

Asynchronous designs are often composed of a hierarchical network of blocks, which contain ports interconnected via asynchronous channels. These channels are simply a bundle of wires and a protocol for synchronizing computation and communicating data between blocks. Numerous forms of channels have been developed that trade off robustness to timing variations for improved power and performance and this section reviews some of the most popular forms.

2.2.1 Bundled-data channels

Bundled data channels consists of data along with one wire that acts as a Req signal and one wire that acts as an Ack signal. The data is encoded as single-rail with one wire per bit. In a typical bundled data channel, illustrated in Figure 2, the sender initiates the communication and tells the receiver when new valid data is available using the Req signal. The receiver after using the data sends an acknowledgement to the sender using the Ack signal.

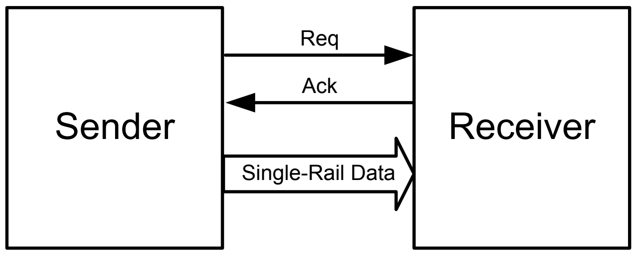


Figure 2. Bundled data channel

Bundled data channel can be implemented in both two-phase [71] and four-phase handshaking protocols [35] as shown in Figure 3. A typical bundled data channel with four phase handshaking is shown in Figure 3(a) where every rising edge (or falling edge) on the Req wire implies as a new data and similarly every rising edge (or falling edge) on the Ack wire acts as the acknowledgement to the data. In four-phase handshaking protocol there is a separate phase where the Req and Ack signals are resetted to there default value as seen in Figure 3(a). Four-phase handshaking protocols are also known as reset to zero phase (RZ) phase. A typical bundled data channel with two-phase handshaking protocol is shown in Figure 3(b) where every edge of Req signal implies presence of a new data and every edge on Ack wire implies the acknowledgement of the data. As we can see from Figure 3(b) there is no separate reset phase. Two-phase handshaking protocols are also known as non-return-to-zero (NRZ) protocols. Since in one complete cycle there are only two transitions in two-phase compared to four transitions in four-phase handshaking, two-phase handshaking protocols often offer better performance and lower energy consumption. However the control circuit required for four-phase protocols may be much simpler then two-phase handshake protocols.

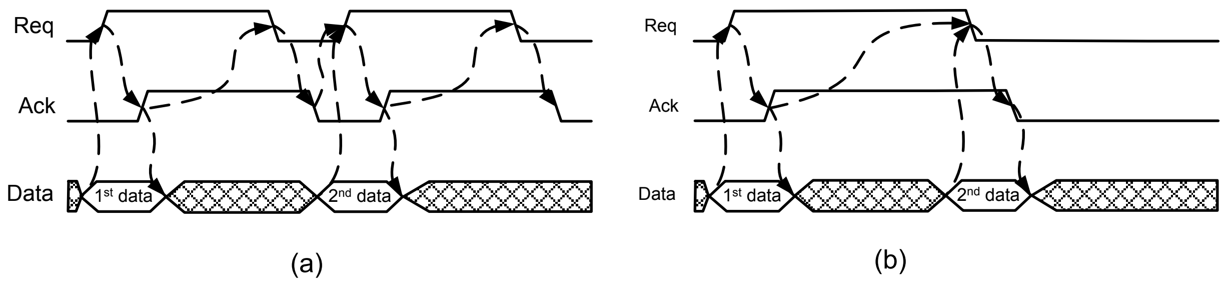


Figure 3. Bundled data channel with (a) four-phase handshaking protocol (b) two-phase handshaking protocol

Bundled data channels are area efficient because the request and acknowledge line overhead is distributed over the width of the entire data bus. It is also power efficient because the activity on the data bus may be low and the power consumption of the handshaking wires is relatively small if the data path is wide (e.g., 64 bits). The key timing assumption (also known as bundling data constraint) in a bundled data channel is that the data should be valid before there is an associated transition on request line. To satisfy this constraint the request signal need to be delayed using a delay line such that this delay is greater then the worst case delay of the data. However in deep-sub micron technology where interconnect delay do not scale in the same fashion as date delays, more margin is sometimes necessary to ensure correct operation. This margin is on the forward latency path of the circuit which is often critical to system performance.

2.2.2 1-of-N channels

1-of-N asynchronous channel consists of data encoded as 1-of-N where N wires are used to represent $\log_2 N$ bits of data and an acknowledge wire. A typical 1-of-N channel is illustrated in Figure 4. Generally 1-of-N channels are implemented using four-phase handshake protocols.

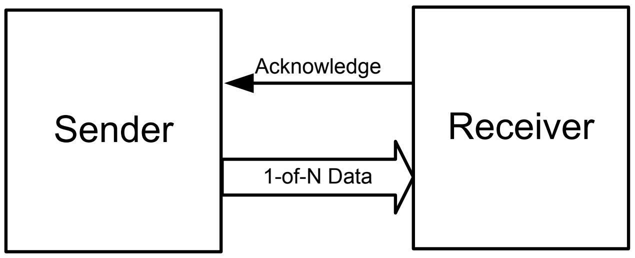


Figure 4. 1-of-N asynchronous channel

In a 1-of-N channel the sender initiates the communication by driving one of the N wires high; the receiver after sensing that one of the wires has been driven high will use the data and will send the acknowledgment by driving the acknowledgement wire high. The sender then resets the data wires and then receiver resets the acknowledgement wire completing the four phases of the handshake. This protocol facilitates delay-insensitive communication between blocks in that the data as the data validity is derived from the N wires used to represent data rather then a separate request wire. Consequently, no timing assumption is needed. This increases robustness to variations, reducing the amount of timing verification required.

The most well known form of this channel is dual-rail, also known as 1-of-2, which uses two data wires per bit of data and is shown in Figure 5. One wire is referred to as the Data_0 or false wire and the other the Data_1 or true wire. The receiver checks the validity of the data by using an OR function on these two wires.

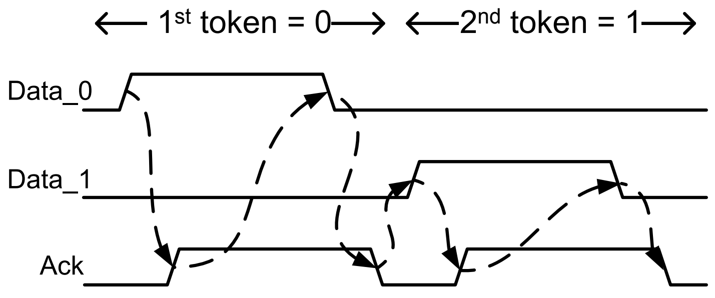


Figure 5. 1-of-2 channel

2.2.3 Single-track 1-of-N channels

In a single-track 1-of-N channels we have data encoded as 1-of-N with no acknowledgement, as illustrated in Figure 6.

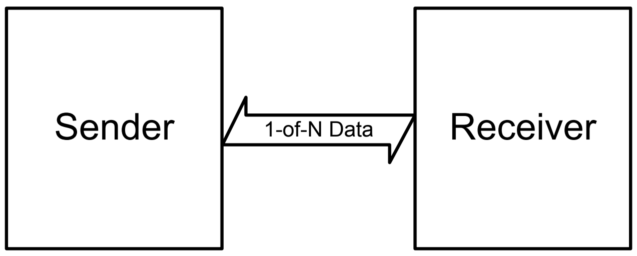


Figure 6. Single-track 1-of-N channel

The single-track 1-of-N channel is implemented with two-phase single-track handshake protocol [9] as shown in Figure 7. The sender drives one of the N wires high thereby sending a token and releasing the channel, after receiving this token, the receiver drives the same wire low and releasing the channel. After driving the wire to its desired state, the sender and receiver(s) must tri-state the wire to ensure that they do not try to drive the wire in opposite directions at the same time.

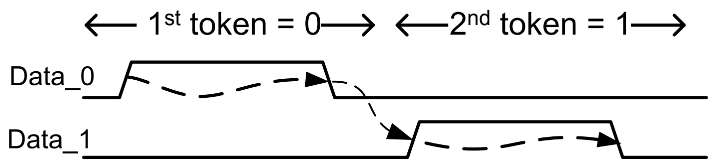


Figure 7. 1-of-2 single-track channel

The two-phase single-track protocol avoids the overhead of the reset phases yielding substantially higher performance than 4-phase protocols. Moreover, compared to bundled-data protocols there is no timing assumption that requires margins on the forward latency, yielding additional performance improvement. Compared to four-phase 1-of-N protocols, there are fewer transitions per bit, resulting in substantially lower power. Compared to bundled-data channels, however, the number of transitions per bit is often larger, yielding higher power consumption per bit transmitted.

Static Single-Track (SST) protocol

The basic concern of single-track handshake protocol is that the channel can be tri-stated for some period of time with only a small staticizer fighting leakage and crosstalk noise. While effective for 250nm, the noise margin for this technology may be too low in deeper submicron processes [30]. In particular, a cross-coupling noise event on a long tri-stated wire can either create a new token or remove a token from the system causing system failure (often in the form of a deadlock). Moreover, in smaller geometries leakage currents may become so high that the staticizers would need to be made stronger to the point that they cannot be easily over-powered. To solve the problem of communications channel in tri-state channel a new variant of single-track handshake protocol was proposed called static single-track protocol (SST) [9][30][38]. The functionality of a SST protocol is the same as single-track handshake protocol with a noticeable difference that after the sender drives the line high, the receiver is responsible for actively keeping the line high until it wants to drive it low. After the receiver drives the channel low, the sender is responsible for actively driving the channel low until it drives it high again. Note that the channel is always actively driven either by sender or receiver hence improving noise immunity.

2.3 Asynchronous design templates

Template-based pipelined design styles have demonstrated very high performance while at the same time being amenable to standard-cell-based ASIC flows and fast design times (e.g., [39] [44]). Different template-based designs trade-off robustness to performance [71][62][63][54]. In this section we present various design templates that are most commonly used for asynchronous designs. These styles are collections of design libraries, protocol definitions and constraints that have been designed and verified to produce functional designs.

2.3.1 GasP

GasP [71] uses a bundled data channel where the request/acknowledge wires are merged into a single-track wire. For GasP, a low level signalizes the presence of a request token in the control channel, while acknowledging it is done by driving the wire back high. Figure 8 shows the GasP circuit where, after reset, L, R, and A are high. When L is driven low by the left environment, the self-resetting NAND will fire, driving A low. This will restore L, activate the data latches, and drive R low, propagating the signal and avoiding re-evaluation until after R is restored high by the right environment. The self-resetting NAND will restore itself by driving A high after 3 transitions. The output of the NAND controls the latches in a parallel single-rail data-path.

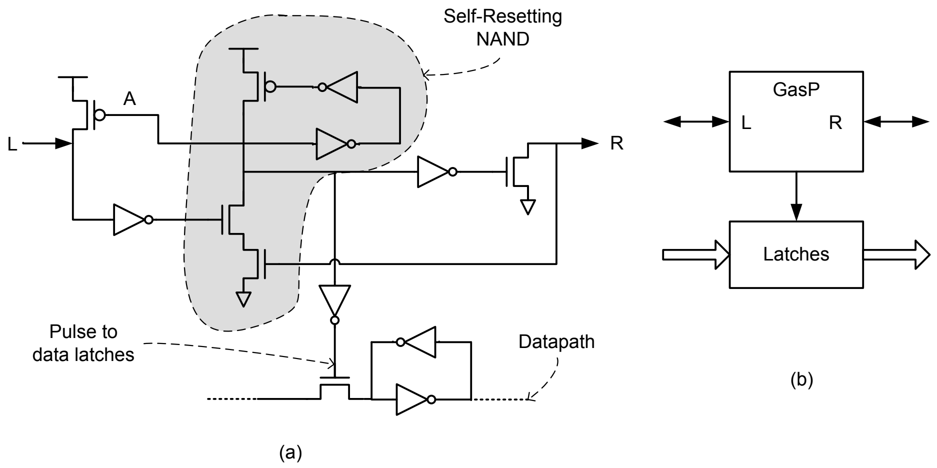


Figure 8. GasP (a) schematic and (b) block diagram

GasP circuits require 4 transitions to forward data and 2 transitions to reset, of the 4 transitions forward latency approximately two transitions are required for the latency through the latches in the data-path and to satisfy the setup and hold times, leaving approximately two transitions for computation in the data-path. Due to its high performance, the timing assumption associated with the data-path being stable before latching (bundle data timing constraint) implies that GasP requires full-custom design as well as new CAD flows that automatically verify this requirement

2.3.2 Pre-charged half buffer (PCHB)

The Pre-Charge Half Buffer (PCHB) is another QDI template was presented in [50] and use 1-of-N asynchronous channels. Each gate has an input completion detection unit and the output also has an output completion detection unit. The acknowledgement signals (Lack and Rack) are active low signals. The block level diagram of the template is shown in Figure 9

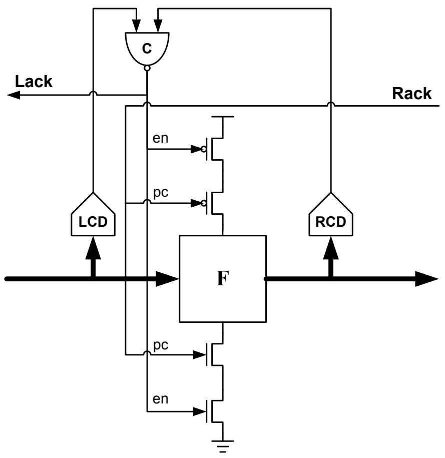


Figure 9. Block level diagram of the PCHB template

The function blocks are designed using dynamic logic which provides higher performance and lower latency then static logic. Unlike WCHB [50] the functional logic is non-weak conditioned. The RCD is used to detect the validity/neutrality of the outputs and LCD is used to detect the validity/neutrality of all the inputs.

The operation of the buffer is as follows. After the buffer has been reset, all data lines are low and acknowledgment lines, Lack and Rack, are high. Consequently en signal is also high and the functional logic is ready to evaluate on the inputs. When input arrives by one of the input rails going high, the functional block evaluate and generates the output. The RCD and LCD check for validity of all the outputs and inputs respectively and the corresponding C-element output will go low, lowering the left-side acknowledgment Lack. After the right environment asserts Rack low (pc = 0) acknowledging that the data has been received the functional block enters the pre-charge phase and resets its outputs to 0. After the left environment resets the inputs and the right environment asserts Rack high the functional block is ready to compute on the next set of inputs. The cycle time of this template depends upon the functional logic but generally varies from 14 to 18 transitions.

2.3.3 Multi-level domino (MLD)

Multi-Level Domino [23] is another design style that uses four phase handshaking protocol and 1-of-N asynchronous channels. Figure 10 shows the block level diagram of the design style.

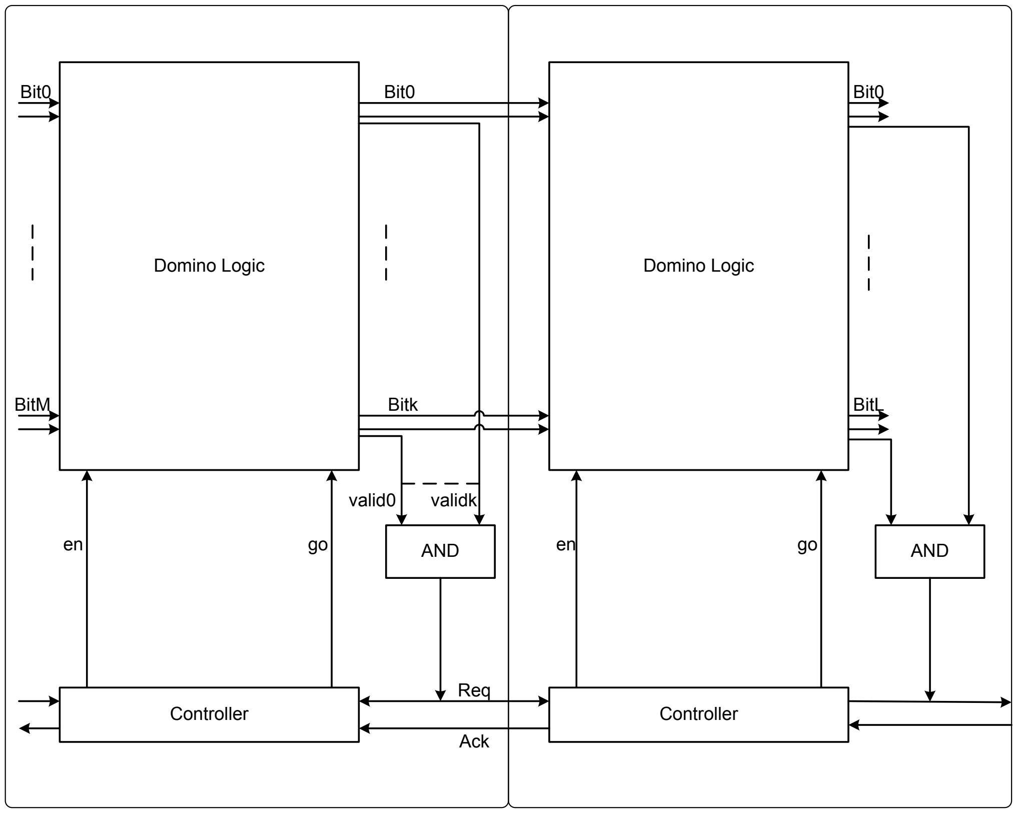


Figure 10. Block level diagram of a MLD pipeline

The data path is constructed out of dynamic logic gates which makes it faster than synchronous designs. A completion detection unit exists for each output and all the validity signals are then combined through an AND gate tree to generate the output valid signal and request signal for the next pipeline stage. The next pipeline stage on receiving the request signal and the output valid signal from AND gate will then send the acknowledgement signal. The overhead of the reset phases are partially hidden through 'eager evaluation' strategy by using two different eval / pre-charge control signal for last level of logic and intermediate levels of logic. This strategy helps in hiding the overhead of reset phases by letting intermediate levels of logic evaluate concurrently with the handshake going between the controllers. The style is targeted more towards medium-grain pipelining and several layers of logic and many data paths in parallel are typically used in a single pipeline stage. This yields a small overhead from the addition of the pipeline stage control units and hence an area efficient design. The cycle time of this design style is dependent upon the number of logic levels, number of outputs and number of fan out channels in the pipeline stages.

2.3.4 Single-track Full Buffer (STFB)

Figure 11 shows a typical STFB [30] cell's block diagram which uses 1-of-N single-track channel. The function blocks are designed using dynamic logic which provides higher performance and lower latency then static logic. When there is no token in the right channel (R) (the channel is empty), the Right environment Completion Detection block (RCD) asserts the "B" signal, enabling the processing of a next token. In this case, when the next token arrives at the left channel (L) it is processed lowering the state signal "S", which creates an output token to the right channel (R) and causes the State Completion Detection block (SCD) to assert "A", removing the token from the left channel through the Reset block. The presence of the output token on the right channel resets the "B" signal which activates the two PMOS transistors at the top of the N-stack, restoring "S", and deactivates the NMOS transistor at the bottom of the N-stack, as shown in Figure 11, disabling the stage from firing while the output channel is busy.

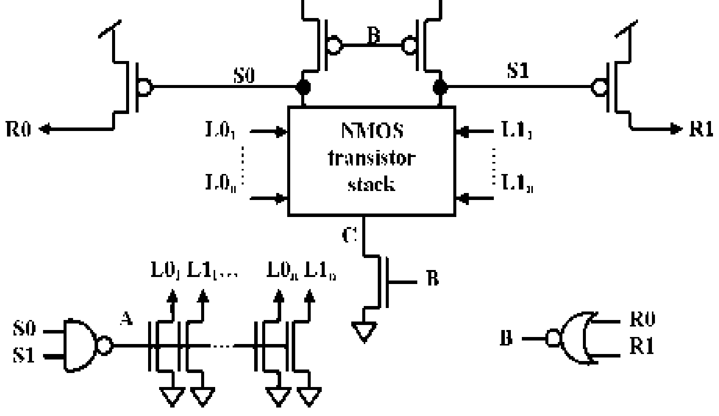


Figure 11. STFB transistor level diagram

The cycle time of the STFB template can be as low as 6 transitions with a forward latency of 2 transitions. This implies that the peak pipeline throughput can be achieved with just three stages per token, which allows high pipeline occupancy and the implementation of high performance small rings. The full-buffer characteristic of STFB stage refers to each stage capacity of holding one token

2.3.5 Static Single-track Full Buffer (SSTFB)

SSTFB is a variant of STFB templates designed for ultra-high-speed fine-grained asynchronous pipelines. It uses 1-of-N single-track channels with static single-track handshake protocol and the functional data path is made of dynamic logic. The functionality of SSTFB template is similar to the STFB template with cycle time of the SSTFB template as low as 6 transitions (~1.2GHz in 180nm technology) [32] with a forward latency of 2 transitions.

Figure 12 (a) shows an output stage of a SSTFB sender on the left hand side driving a single-track wire associated with a channel connected to the SSTFB receiver on the right hand side and Figure 12 (b) shows the symbol of the output and input drivers. Initially the wire is at logic state 0 M1,M4 and M6 are turned off while M2, M3 and M5 are on. The sender initiates the handshake by driving the wire high by turning M1 on. When the wire reaches the switching threshold of INV_LO the inverter turns M4 on which then statically drives the wire high until the receiver drives the channel low. Similarly, after the receiver drives the wire low by turning M6 on, INV_HI turns M3 on this then statically drives the wire low until the sender initiates a new communication by driving the wire high.

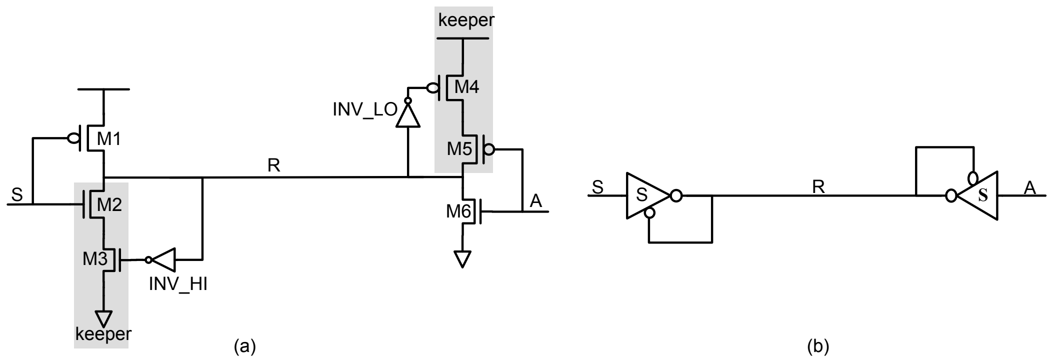


Figure 12. Static single-track protocol interface

Another key advantage of static single-track protocol lies with the sizing of the keepers shown in Figure 12. Typically in domino logic the size of the keeper is constrained to be sufficiently weaker then the pull down logic to reduce the fight between keeper and pull down logic. This in turn degrades the noise immunity as the keeper is not strong enough. In SSTFB there is no such fight between the keeper and the pull down logic. Hence the keeper circuit can be sized to a suitable strength creating a tradeoff between area and robustness to noise.

The use of semi-weak conditioned domino logic makes them inefficient for gates with complex functionalities. As the number of inputs of the gate increases, the size of the NMOS stack increases which causes slower domino logic, increases in area as well as charge sharing problems. SSTFB template also has a rigid timing assumption of pulse-width of 3 transitions on nodes S0, S1 and A in Figure 11. During this pulse width the SUP and SDOWN gates are responsibly for charge / discharge of the input and output wires. This pulse-width is independent of the load offered on the wires which in deep submicron technology is getting difficult to satisfy.

Additionally in case of non-linear pipelines with multiple outputs implementing complex functionalities, there exists a critical race between outputs as we have the same control signal for evaluation of the domino logic. This control signal stops evaluation as soon as one of the dual-rail outputs becomes valid while the other output functionalities may not have completed evaluation. This race is addressed in [30] by constraining the length, and thus, capacitance, of the longest channel driven. However, in deep submicron

ASIC design, the presence of process variations and crosstalk noise makes it increasingly difficult and inefficient to satisfy these constraints.

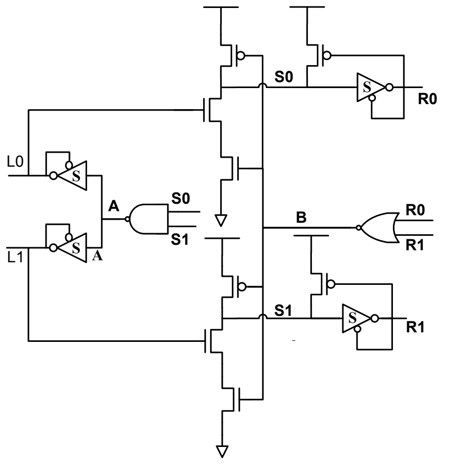


Figure 13. SSTFB transistor-level diagram

2.4 Summary

In this chapter we have given an overview of different asynchronous channels and different type of asynchronous design templates. As one can see there are many design alternatives depending upon the type of asynchronous channel, logic styles, data encoding and pipeline granularity; which could yield substantial benefits depending on the specifications of the design.

3 Theory of single-track designs

A single-track channel acts as a shared resource driven by the two pipeline stages or processes that must drive the channel mutually exclusively. In this chapter we will present principles for the design of such processes to communicate with each other. This work builds off the work done by Nystrom and Martin [53] on developing constraints for design of single-track processes. In particular, their work focuses on homogenous single-track processes which have a single-level of logic with the same forward latency and same pulse-widths on input and output drivers. The homogeneous processes simplify analysis of the designs at the cost of design optimality. We thus extend this work to cover non-homogeneous single-track processes which can have multiple levels of logic and different driver pulse-widths. Multiple levels of logic enable a new trade-off between performance and area reducing the effective control area overhead without sacrificing latency. In addition, the flexibility of having different pulse-widths on input and output drivers increases noise margin robustness.

A key challenge in the design of non-homogeneous processes is the proper management of the single-track channel between non-homogenous pipeline stages. This chapter first proposes extensions of Nystrom and Martin's relative-timing constraints to non-homogeneous processes to ensure a notion of correct handshaking that includes both avoiding a fight on the channel wires and guaranteeing voltages swing satisfy a user defined noise margin. Then, we derive certain properties on pulse widths of non homogenous single-track processes that are more design intuitive compared to relative timing constraints. Moreover, we derive a lower bound on the cycle time of the channel which better guides micro-architectural performance optimizations.

This chapter is organized as follows, in Section 3.1 we provide the background on design rules for homogeneous processes proposed by Nystrom and Martin. In Section 3.2 we derive the design rules for non-homogeneous processes and in Section 3.2.3 we derive the cycle time constraints of a channel.

3.1 Homogeneous processes

Consider a single-track handshake performed on a channel driven by two processes P and Q with these two processes driving the channel mutually exclusively. Once process P is responsible for driving the channel high and the other process Q is responsible for driving the channel low. Assume each process is homogeneous such that they have the same forward latency and have same pulse widths on the drivers. There exists a timing constraint on the duration which each process can drive the channel in order to avoid interference. Nystrom and Martin defined this constraint by defining a maximum single-track hold time constraint.

Maximum Single-track hold time: If a process P begins driving an interface node to a new value v at time t, then P must have stopped driving the node at some time $t + \sigma_v$, where σ_v is a (system-wide) global constraint; P must drive the node to v once it has detected that the node has left v.

Similarly there is a constraint where once a process Q detects that the channel has been driven high; how soon can it drive the channel low so that it does not interfere with process P driving the channel high. Nystrom and Martin define this constraint by defining the minimum single-track set up time in the following definition.

Minimum Single-track setup time: If a process Q detects that an interface node has switched to a new value v at time t, and then Q must not drive that node away from v until the time $t + \xi_v$, where ξ_v is a (system-wide) global constraint.

To ensure the correctness of the handshake protocol process P and Q should not interfere with each other which means that when P is driving the channel Q should not drive it and vice versa. Nystrom and Martin captured this condition in the following definition

Single-track handshake constraint: A set of processes S satisfies the single-track handshake constraint if $\xi_v \geq \sigma_v$ for all v and all processes in S.

These above mentioned properties will guarantee that there will never be any interference between two processes sharing a channel. However the constants are global leading to single-track designs with homogeneous processes in S designed to meet the global constants ξ_v and σ_v.

3.2 Non-homogeneous processes

In this section we develop the theory that governs the single-track handshake protocol in the presence of single-track non-homogeneous processes. As explained above non-homogeneous processes can have different levels of logic leading to different forward latency and driver pulse widths.

Like in Section 3.1 we assume a single track system to be composed of two processes driving a common channel, which can be driven to value v, where v ∈ {0, 1}. Let S be the set of all such processes and let P be the subset of S which drives the channel to a value 1 and Q be the subset of S which drives the channel to a value 0. Note that P ∩ Q = Φ. Let C be the set of all channels P x Q. Let PWA and PWI be the pulse width of a process where it is actively driving a channel and where it is in inactive state respectively such that PWA: $S \rightarrow R^+$ and PWI: $S \rightarrow R^+$.

3.2.1 Single-track channels

We first define the channel forward latency as the time required by a process s to drive a channel c to either to 0 or 1. This is formalized in the following definition.

Definition 1. (Channel forward latency) *For any process s, s ∈ S the time required to drive a channel c, c ∈ C to a value v, v ∈ {0, 1} is the forward latency D_c^v, where $D_c^v \in R^+$.*

We define σ_c^v as the release time of a channel c, c ∈ C, which defines the amount of time by which one process should release the channel after driving it to a value v in order not to interfere with the other process subsequently driving the channel to a different value. This release time is quite similar to the maximum single-track hold time σ_v proposed by Nystrom and Martin with the key difference begin that while σ_v was defined as a global constant, channel release time is a local variable dependent upon the properties of the channel c, enabling the analysis of non-homogeneous processes. This is formalized in the following definition.

Definition 2. (Channel release time) *For any process s ∈ S, if it drives the channel c, c ∈ C to a new value v, v ∈ {0, 1} at time t then it must release the channel by time $t + \sigma_c^v$, where $\sigma_c^v \in R^+$.*

Figure 14 shows a timing diagram of a single-track handshake protocol with forward latency and channel release time.

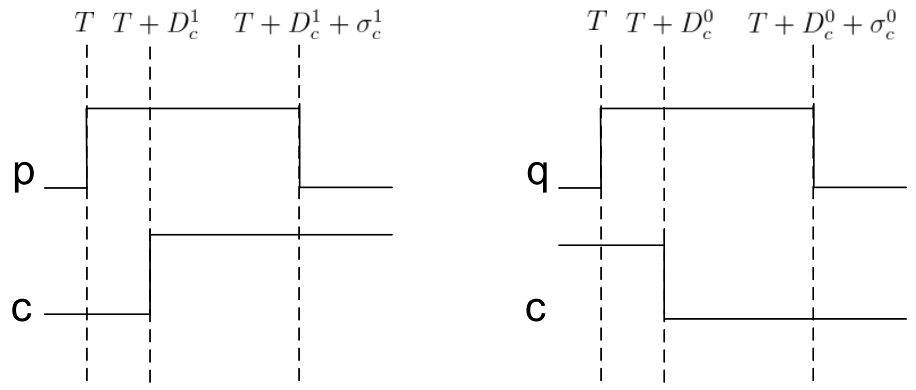


Figure 14. Forward latency and release time of a single-track channel

We define ξ_c^v to be the setup time of a channel c, c ∈ C, which defines the minimum amount of time a process should wait after a channel has been driven by some other process, before it can drive the channel to a new value v. This setup constraint is similar to the minimum single-track setup time proposed by Nystrom and Martin with the key difference being that while ξ_v is a global constant, in our work setup time is a local variable dependent upon the properties of the channel c. This is formalized in the following definition

Definition 3. (Channel setup time) *For any processes p and q, p, q ∈ S, if it detects that the channel c, c ∈ C: (p, q) has been driven to a value v, v ∈ {0, 1} by process q at time t then the earliest it should drive the channel to a new value u, u ≠ v and u ∈ {0, 1} is* $t + \xi_c^v$, *where* $\xi_c^v \in R^+$.

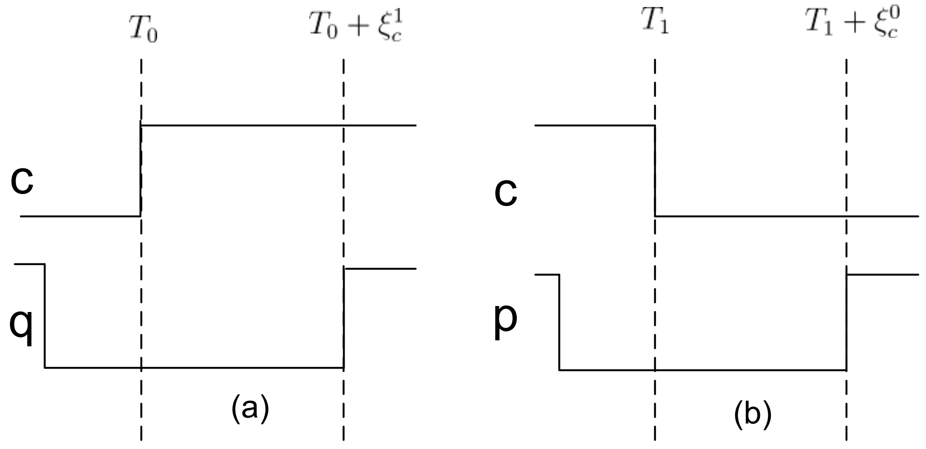


Figure 15. Setup time of a single-track channel

The setup time of a channel is similar to the set up time in synchronous system where the process acts somewhat like a flip-flop. In particular, the setup-time is the minimum time difference between validity of the data and the process/flip-flop acting on the valid data. The difference is that the process acts on the data by internal recognizing it is valid, using this valid value, and driving it to a new value whereas the flip-flop acts on the data by sampling it only after an external clock signal indicates it is valid. Figure 15 shows a timing diagram of a single-track handshake protocol with channel setup time. Here in Figure 15 (a) after channel c is driven high, process q has to wait for setup time ξ_c^1 to drive channel c to low, similarly in Figure 15 (b) process p has to wait for setup time ξ_c^0 before driving channel high.

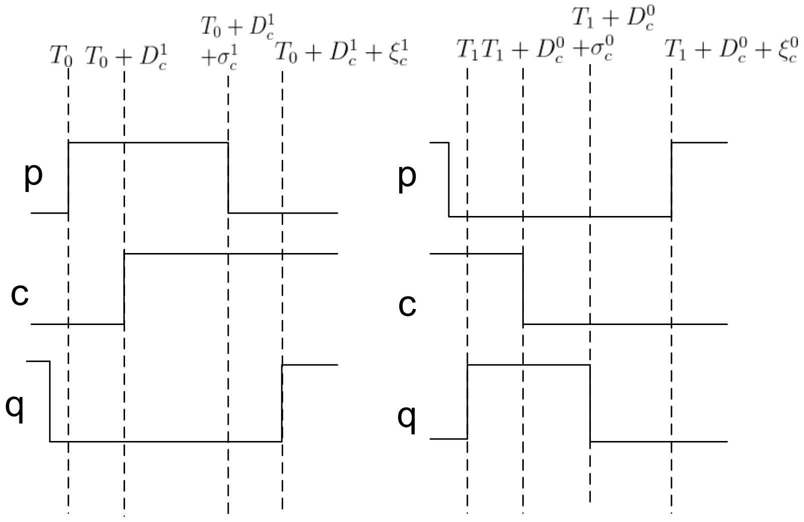


Figure 16. Channel handshake constraint on channel c

In order to ensure the correct transfer of a token on a channel c driven by two processes p and q we need to avoid fight between these processes such that process p should stop driving the channel high before process q starts driving the channel low as shown in Figure 16 , similarly process q should stop driving the channel low before process p starts driving the channel high. This can be formalized in the following definition.

Definition 4. (Channel handshake constraint) *For any c in C: c = (p, q) such that processes p ∈ P and q ∈ Q, c satisfies the channel handshake constraint if* $\xi_c^v \geq \sigma_{c,}^v$ *where v ∈ {0, 1}* .

We also define η_c^v to be the hold time of a channel c, c ∈ C, which defines the minimum amount of time a process should hold the channel after driving it to some value v as shown in Figure 17. The hold time of a channel is similar to the hold time in synchronous system as it defines a minimum time for which the data on the channel should remain stable. However the key difference between them is that while hold time in synchronous system is interpreted as the time after the reference clock signal triggers the sampling of the signal before which the data must remain valid, the hold time of a single-track system is independent of any external sampling signal. Rather, it is simply a function of the load offered on the channel and guarantees full voltage swing on the channel. This is formalized in the following definition

Definition 5. (Channel hold time) *For any process $s \in S$, if it drives the channel c, $c \in C$ to a new value v, $v \in \{0, 1\}$ at time t then it should hold the channel by time $t + \eta_c^v$, where $\eta_c^v \in R^+$.*

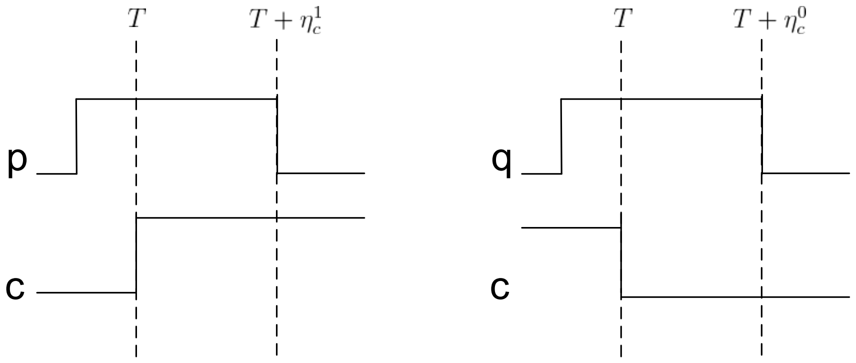


Figure 17. Hold time of the channel

3.2.2 Single-track processes

Now that we define characteristics of the single-track channel, we define characteristics of the processes that drive them. We first define hold time of a process as the time a process holds its state after driving a channel to a value *v*.

Definition 6. (Process hold time) *For any process $s \in S$, after driving the channel c, $c \in C$ to a new value v, $v \in \{0, 1\}$ the time during which it holds its state is called the hold time (σ_p^v) of the process, where $\sigma_p^v \in R^+$.*

The pulse width of a process actively driving a channel can be decomposed into two phases as shown in Figure 18, during the first phase the process is driving the channel to v and the during the second phase while the channel has been driven to v the process holds its state. The first phase is the forward latency of the channel and the second phase is the hold time of the process. This is formalized in the following definition.

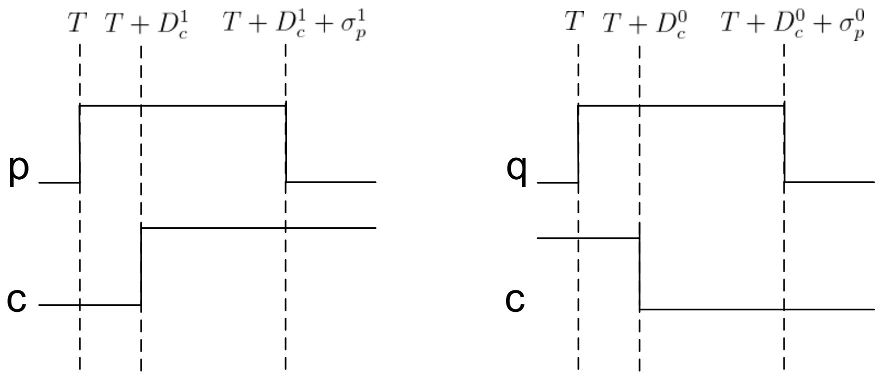


Figure 18. Timing diagram of single--track process with process hold time and pulse width active

Definition 7. (Pulse width active) *For any p, p ∈ P the pulse width active of p is defined as the sum of forward latency and hold time of process p. We have* $PWA_p = \sigma_p^1 + D_c^1$ *and similarly for any q in Q, we have* $PWA_q = \sigma_q^0 + D_c^0$*.*

Next we define setup time of a process as the time a process waits after the channel has been driven, before driving it to a new value.

Definition 8. (Process setup time) *For any process q, q ∈ S, if it detects that the channel c, c ∈ C has been driven to a value v, v ∈ {0, 1} by some other process p, p ∈ S at time t then the time it waits before driving the channel to a new value u, u ≠ v and u ∈ {0, 1} is called as setup time (*ξ_q^v*) of process q,, where* $\xi_q^v \in R^+$.

The pulse width of a process q in an inactive state can be decomposed into two phases as shown in Figure 19, first phase during which either the other process p is driving the channel or is about to drive the channel. This first phase will be equal to or greater than the forward latency of the channel, and the second phase during which after the channel has been driven to a value v by p, q waits for its setup time before it drives the channel. This is formalized in the following definition

Definition 8. (Pulse width inactive) *For any p, p ∈ P and for any q, q ∈ Q driving a channel c, c ∈ C: c = (p, q), we have* $PWI_p \geq D_c^0 + \xi_p^0$ *and* $PWI_q \geq D_c^1 + \xi_q^1$

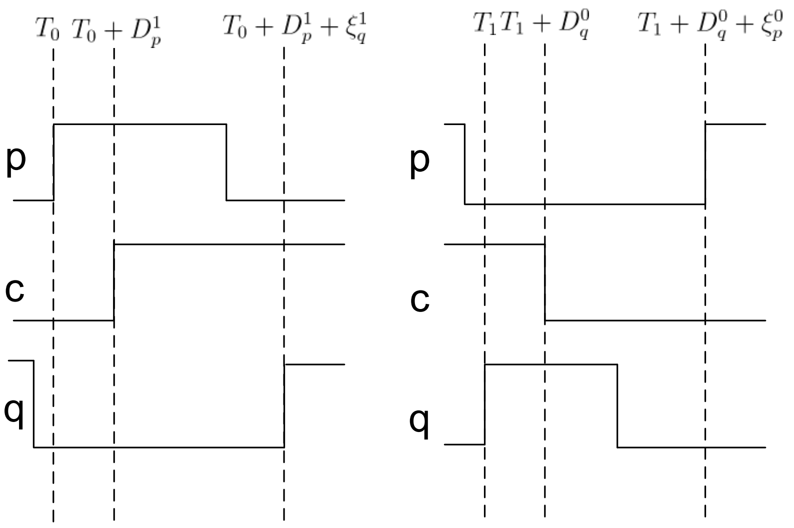


Figure 19. Timing diagram of a single-track process with setup time and pulse width inactive

In a single-track system a channel c is driven by processes p and q and the hold time of these processes should always be less than hold time of the channel. The intuition behind this constraint is to put a bound on the time by which processes should release the channel to ensure mutually exclusivity among processes. This bound is defined as the release time of the channel. This is formalized in the following definition.

Definition 9. (Release time constraint) *For any c in C: c = (p, q) such that p ∈ P and q ∈ Q, p and q satisfies the release time constraint if the release time of process p and q are less than the release time of the channel,* $\sigma_p^1 \leq \sigma_c^1$ *and,* $\sigma_q^0 \leq \sigma_c^0$.

Similarly we define a setup time constraint which states that the setup time of processes p and q should always be greater than setup time of the channel c. The intuition behind this constraint is to put a bound on how early a process can start driving the channel to a different value to ensure mutually exclusivity among processes. This bound is defined as setup time of the channel. This is formalized in the following definition.

Definition 10*. *(Setup time constraint) *For any c in C: c = (p, q) such that p ∈ P and q ∈ Q, p and q satisfied the setup time constraint if setup time of processes p and q are greater than setup time of the channel,* $\xi_p^0 \geq \xi_c^0$ *and,* $\xi_q^1 \geq \xi_c^1$.

Next we define a hold time constraint which states that the hold time of processes p and q should always be greater than hold time of the channel c. This constraint guarantees that a process drives the channel sufficiently long enough for the channel to have a full rail-rail voltage swing on it. This is formalized in the following definition.

*Definition 11. (**Hold time constraint**) For any c in C: c = (p, q) such that p ∈ P and q ∈ Q, c satisfies the hold time constraint if hold time of processes p and q are greater than hold time of the channel,* $\sigma_p^1 \geq \eta_c^1$ *and* $\sigma_q^0 \geq \eta_c^0$.

In a single-track design with channel c driven by two processes p and q, we define a correct handshake as one in which channel c satisfies the channel handshake constraint and processes p and q satisfy their associated release time, setup time constraint and hold time constraints. The intuition is to avoid overlap between processes actively driving the channel while at the same time satisfying a voltage swing on the channel. This is formalized in the following definition.

***Definition 12**. (**Single-track handshake constraint**) In a single-track design single-track handshake constraint is satisfied if, for all c in C: c = (p, q) where p ∈ P and q ∈ Q*

1. *c satisfies the channel handshake constraint*
2. *p and q satisfies the release time constraint*
3. *p and q satisfies the setup time constraint*
4. *p and q satisfies the hold time constraint*

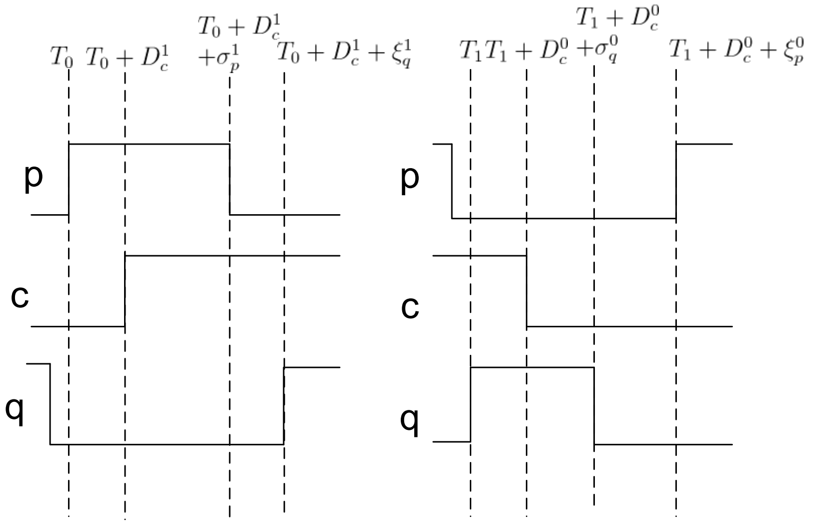


Figure 20. Single-track handshake constraint

It can be observed from Figure 20 in a single-track design as a result of single-track handshake constraint, for any channel c with process p and q the pulse-width inactive of q (PWI_q) should always be greater then pulse-width active of p (PWA_p) and similarly pulse-width inactive of p (PWI_p) should always be greater then pulse-width active of q (PWH_q). These observations can be proven in the following lemma.

Lemma 1. ***If a single-track design satisfies single-track handshake constraint, then for all c in C: (p, q) such that $p \in P$ and $q \in Q$, we have $PWI_q \geq PWA_p$ and $PWI_p \geq PWA_q$.***

Proof: $PWI_q \geq PWA_p$

$PWI_q \geq D_c^1 + \xi_q^1$ ....From Definition 8.

$PWI_q \geq D_c^1 + \xi_c^1$ ....From Definition 12 and Definition 10 .

$PWI_q \geq D_c^1 + \sigma_c^1$ ....From Definition 12 and Definition 3.

$PWI_q \geq D_c^1 + \sigma_p^1$ ....From Definition 12 and Definition 9.

$PWI_q \geq PWA_p$ ....From Definition 7. ■

$\mathbf{PWI_q \geq PWA_p}$

$PWI_p \geq D_c^0 + \xi_p^0$ ....From Definition 8.

$PWI_p \geq D_c^0 + \xi_c^0$ ....From Definition 12 and Definition 10.

$PWI_p \geq D_c^0 + \sigma_c^0$ ....From Definition 12 and Definition 3.

$PWI_p \geq D_c^0 + \sigma_q^0$ ....From Definition 12 and Definition 9.

$\mathbf{PWI_p \geq PWA_p}$ ....From Definition 7. ■

However in a single-track design if process p and q satisfy the pulse width constraints proved above, it does not ensure correct handshake on channel c as it does not guarantee mutually exclusive nature of the processes and hence violating condition one or more of 1, 2 and 3 conditions defined in Definition 12. As a result the lemma proved above is a necessary condition for single-track design.

3.2.3 Rail-2-rail voltage swing

As a result of hold time constraint there exists a lower bound on the pulse width active of a process as shown in Figure 21. This lower bound can be defined as minimum pulse width which defines the amount of time a process should drive the channel to a value v so that the channel can satisfy the user defined voltage swing constraints for value v. This minimum pulse width can be decomposed into two phases; one where process is driving the channel to a value v this time is equal to the forward latency of the channel. The other phase is where process should hold the channel; this phase is equal to the hold time of the channel.

*Definition 13. (**Minimum pulse width**) For any c in C: (p,q) where p $\in P$ and q $\in Q$, the minimum pulse width (PWN_c^v) required for the channel to be driven to a value v can be defined as the sum of forward latency of the channel (D_c^v) and hold time of the channel (η_c^v)*

$$PWN_c^v = D_c^v + \eta_c^v$$

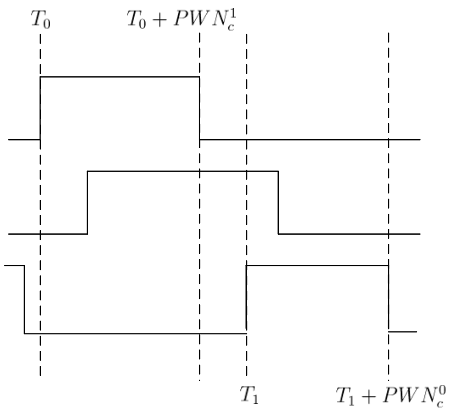


Figure 21. Timing diagram of a single-track system with rail-2-rail swing constraint

*Lemma 2. (**Rail-2-Rail swing constraint**) If a single-track design satisfies the single-track handshake constraint, then for all c in C: (p, q) such that p $\in P$ and q$\in Q$ we have $PWA_p \geq PWN_c^1$ and $PWA_q \geq PWN_c^0$.*

Proof: $PWA_p \geq PWN_c^1$

$PWA_p = D_c^1 + \sigma_p^1$ ………..From Definition 9.

$PWA_p \geq D_c^1 + \eta_c^1$ ………..From Definition 11.

$PWA_p \geq PWN_c^1$ ………..From Definition 13. ■

$PWA_q \geq PWN_c^0$

$PWA_q = D_c^0 + \sigma_q^0$ ………..From Definition 9.

$PWA_q \geq D_c^0 + \eta_c^0$ ………..From Definition 11.

$\mathrm{PWA_q} \geq PWN_c^0$ ....From Definition 13. ■

The rail-2-rail constraint proposed puts a lower bound on the pulse width of the processes which is necessary to maintain a user defined voltages swing on the channels. This lower bound is dependent upon the technology specifications and the wire length of channel This lower bound can have margins to account for worst case crosstalk noise as well as process variations leading to increased robustness to noise and process variations.

The proposed constraints provide design rules for non-homogeneous single-track processes. While the use of homogeneous gates limits the amount of time for computation to a single level of logic making single-track designs expensive in terms of area and energy. The proposed theory removes this limitation by guiding the design of non homogeneous processes with the flexibility of having different pulse widths. This flexibility enables the following advantages.

1. Multiple levels of logic: With the flexibility of having different pulse width the processes can have multiple levels of logic. This helps in reducing the control area overhead by sharing the same control logic among several levels of logic thus making the design more area and energy efficient.
2. Long wires: In presence of homogeneous processes to satisfy rail-2-rail swing constraint one needs to insert pipeline buffers for long channel wires as proposed by Ferretti in [30]. This may increase the imbalance in the pipelines thus degrading the performance. In case of non-homogeneous buffers instead of inserting pipeline buffers the pulse width of the drivers can be increased as per defined in Definition 13.
3. Timing Analysis: The constraints proposed in this chapter can then be verified using static timing engine as shown in [44] and in case of a violation can be satisfied by changing the pulse widths of the drivers using standard P&R ECO flow.

3.3 Performance metric

The performance of non-homogeneous processes is measured in terms of cycle time which is formalized in the definition below

Definition 13. *For any $s \in S$ the cycle time (T_s) of a process s is defined as $T_s = PWA_s + PWI_s$*

If two processes p and q are related through a channel c then the cycle time of the channel c will be the same as cycle time of q and cycle time of process p.

Definition 14. (Local cycle time of a channel): *For any c in C: (p, q) such that $p \in P$ and $q \in Q$ the cycle time of the channel c, T_c equals the cycle time of p and q, i.e.*

$T_c = T_p = T_q$

$T_c = PWA_p + PWI_p = PWA_q + PWI_q$

As a result of single-track handshake constraint and rail-2-rail swing constraint the cycle time of a channel c in C: (p,q) has a lower bound on the cycle time T_c which is formalized in Definition 15 and is proven in Theorem1.

Definition 15. (Minimum cycle time): *For a channel c in C, minimum cycle time T_{min} is defined as $T_{min} = PWN_c^1 + PWN_c^0$*

Theorem1. *If a single-track design satisfies the single-track handshake constraint, for any c in C: (p, q) the cycle time of c (T_c) is always greater then T_{min}.*

Proof:

$Tc = PWA_p + PWI_p$ ……………………From Definition 13

$\geq PWA_p + PWA_q$ ……………….....From Lemma 1

$\geq PWA_p + PWN_c^0$ …………………..From Lemma 2

$\geq PWN_c^1 + PWN_c^0$ … ……………………From Lemma 2

Hence $Tc \geq T_{min}$ ■

It has been shown in [8] that better the performance of pipeline / slack matching buffers less number of buffers are required for slack matching. The flexibility of having non-homogeneous buffers helps in reducing the number of slack matching buffers making the design more area and energy efficient. The lower bound of cycle time proved above guides the designer with estimation of the performance of a channel and provides the pulse-widths required on the input and output drivers of sender and receiver to achieve the cycle time.

4 Non-homogeneous single-track templates

4.1 Introduction

In this chapter we present three new design templates implementing 1-of-N single-track channel with two phase static single-track protocol for handshaking. These design templates can support a wide range of pipeline granularity giving the designers flexibility in terms of performance and area overhead. The proposed design styles through use of faster dynamic logic provides better forward latency than synchronous designs because of the use of dynamic logic in the data path and no delay margin required to meet setup time of the synchronous storage elements. These designs styles are based on *Single-track Full Buffer* (STFB) circuit family proposed by Ferretti [30]. The STFB circuit family uses semi-weak conditioned domino logic due to which makes it inefficient for gates with complex functionalities. Due to the semi weak-conditioned domino logic, as the number of inputs of the gate increases, the size of the NMOS stack increases which causes slower domino logic, increases in area and leakage current, and presents more potential charge sharing problems. The STFB circuit family also has a rigid timing assumption on the pulse-width required to charge / discharge a wire which in deep submicron technology are difficult to satisfy. The proposed design styles use non weak-conditioned logic which leads to smaller N stack leading to smaller area and better forward latency. The proposed design styles also support configurable pulse-width on channels which are dependent on the wire load of the channel to satisfy the hold time constraint proposed in Chapter 3. Due to the two phase handshaking the design templates provide better throughput per area than other existing design styles for a broader range of performance targets and lower power consumption than their counterpart QDI designs.

This chapter is organized as follows. In Section 4.2 we present the first template which targets high performance applications, in Section 4.3 we present the multi-level single-track (MLST) template which targets medium and low performance applications and in Section 4.4 we present a variant of the MLST template which uses conventional dynamic logic with single-track controller and finally we offer some conclusions in Section 4.5.

4.2 Single-Level Single-Track (SLST) template

This template is optimized for high performance applications with smaller pipeline stages limiting themselves to single level of logic. It also has the flexibility of having different pulse widths on their input and output drivers. The pulse width can be adjusted per the load offered on the wires using a standard P&R ECO flow.

4.2.1 Block level diagram

Figure 22 illustrates the block level diagram of the proposed template. L0 and L1 are dual rail single-track inputs while R0 and R1 are dual rail single-track outputs, *Ov* is the valid signal generated by the domino logic block indicating the validity / neutrality of the output channel and *Iv* is the signal generated by the Dreset block indicating the validity and neutrality of the input signal. *ICv* (*OCv*) is the input (output) channel valid signal generated by Left (Right) Channel Completion detector (LCCD / RCCD) indicating the validity and neutrality of all the input (output) channels. *Ack* is the common acknowledgement signal for all the input channels and *IOv* detects the combined validity of all the input and output channels and neutrality of all output channels.

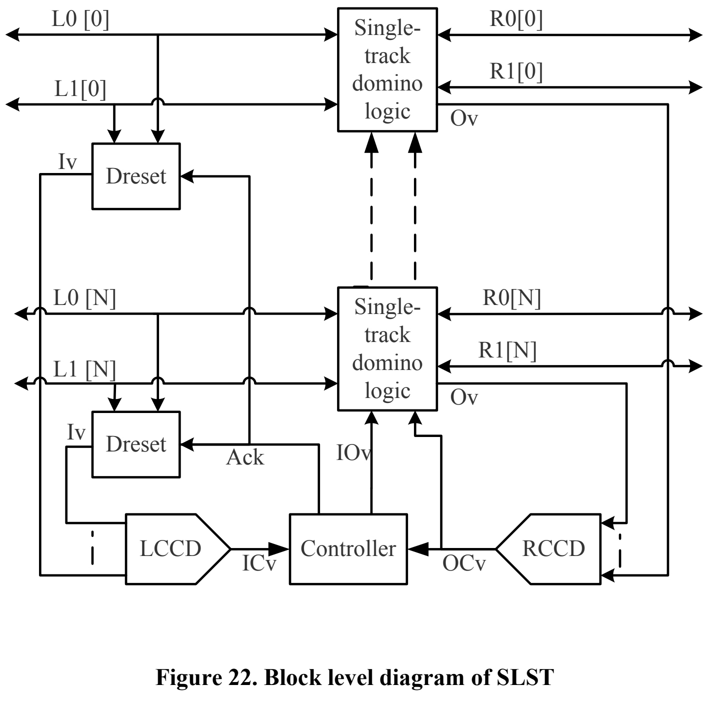


Figure 22. Block level diagram of SLST

Functional logic

The functional data path is based on dynamic logic with evaluation and pre-charge controlled by the “go” signal as shown in Figure 23 (a). The dynamic logic is based on non-weak conditioned logic which improves forward latency and average throughput and provides lower area and less charge sharing compared to semi weak conditioned logic used in STFB.

After coming out of the reset phase, the domino logic is left in evaluate phase and in presence of data on inputs it drives one of the dual-rail state signals (S0 or S1) low generating a valid token on the dual-rail single-track wires (R0 / R1). At the same time, the output valid (Ov) is driven high through the nand gate indicating presence of a valid token on the output channel. At the same time the eval / precharge signal ‘go’ is being driven low leading to the pre-charge of S0 and S1. The pulse-width on the dual-rail stage signals can be increased by changing the length of the delay line to satisfy the hold time constraint defined in Section 3.2. When all output channels of the pipe stage are acknowledged (OCv=0), go signal is driven back high leading dynamic logic back in evaluate phase.

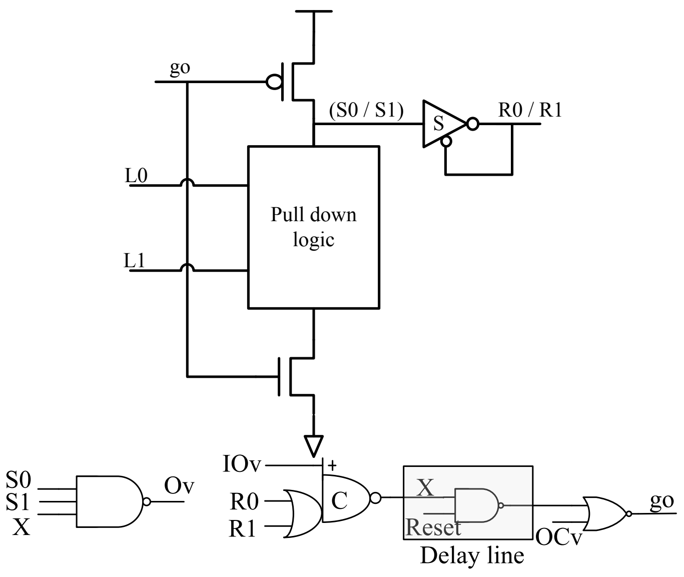


Figure 23. Transistor level diagram of functional block

In the case of non linear pipelines, one output channel can become valid and be acknowledged by its receiver before other outputs become valid or all input channels have arrived. In this situation a token on that output channel will be lost leading to deadlock. This critical race is resolved using the state signal X where it is driven low as soon as one of the two rails R0 / R1 goes high is driven high only when R0 / R1 is driven low by its receiver and controller indicates the validity of all input and output channels are by "input output valid" signal (IOv) being low. The generation of IOv signal will be explained when we discuss the design of the controller.

Controller

Figure 24 shows the transistor level diagram of the controller used. The controller is responsible for generating the acknowledgement signal Ack which drives the input channel low. When all the input channels are valid (ICv = 1) and all output channels are valid (OCv = 1), the "Ack" signal is driven low which then drives input channels low. The Ack signal is driven back high when LCCD detects the neutrality of all input channels by driving ICv low using the generalized C-element. The controller is also responsible for generating the "input output valid" (*IOv*) signal which is driven low when all input and output channels are valid and is driven high when all output channel have been driven low by their receivers.

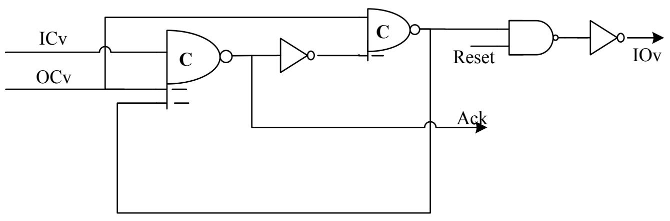


Figure 24. Transistor level diagram of the controller

Dreset

Figure 25 shows the data reset block which is responsible for driving the input rails low. The reset block drives the dual rail input low when Ack signal is driven low by the controller. The reset block uses a delay line which can be configured by changing the delay line to increase the pulse width of A to satisfy the hold time constraint defined in Section 3.2.3. The NOR gate shown in Figure 25 detects the validity and neutrality of the data on dual rails such that the input channel valid signal (Iv) is driven high when one of the rails (L0 / L1) is driven high by the sender and is driven back low when the input channel is driven low.

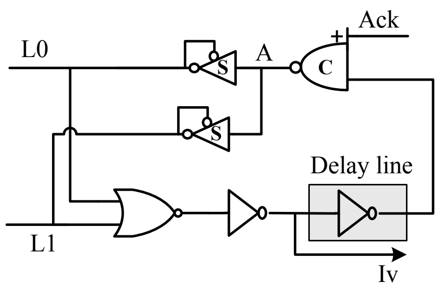


Figure 25. Transistor level diagram of the Data reset block

Right / Left channel completion detectors

The function of right channel completion detector (RCCD) is to ensure the validity and neutrality of all the output channels before acknowledging the input channels. When all the output channels are valid the completion detector drives the left channel valid (OCv) signal high, and when all the output channels are driven low; the completion detector drives the OCv low**.** In a similar fashion the left channel completion detector (LCCD) is responsible for detecting the validity and neutrality of the input channels. When all the input channels are valid the completion detector drives the left channel valid (ICv) signal high, and when all the input channels are driven low; the completion detector drives the ICv low.

The completion detectors are implemented using special asynchronous gates C-elements. Figure 26 (a) shows the 2 input C-element symbol and Figure 26 (b) shows the transistor level diagram of it. These C-elements are then combined in a tree fashion to implement completion detectors for more number of inputs.

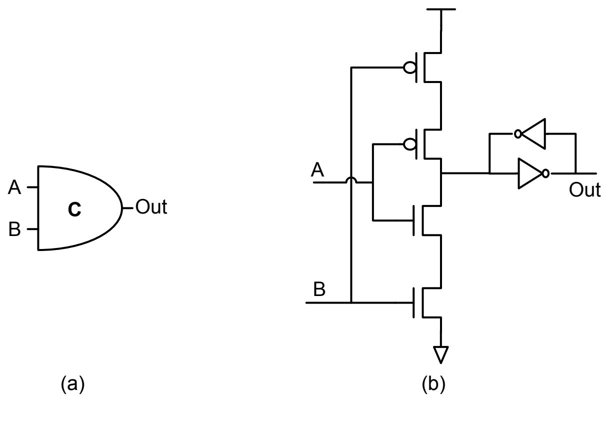


Figure 26. (a) C-element symbol (b) Transistor level description of C-element

In Figure 27 we show an example of SLST pipeline stage with two output functions, one function which implement a buffer and the other function which implement an OR gate.

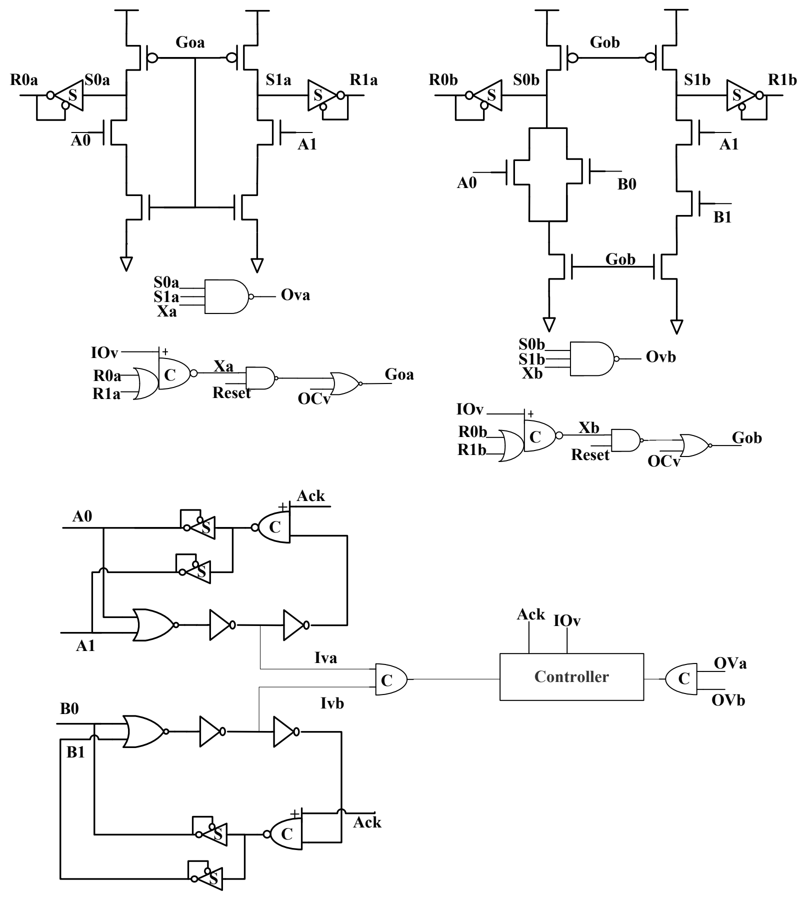


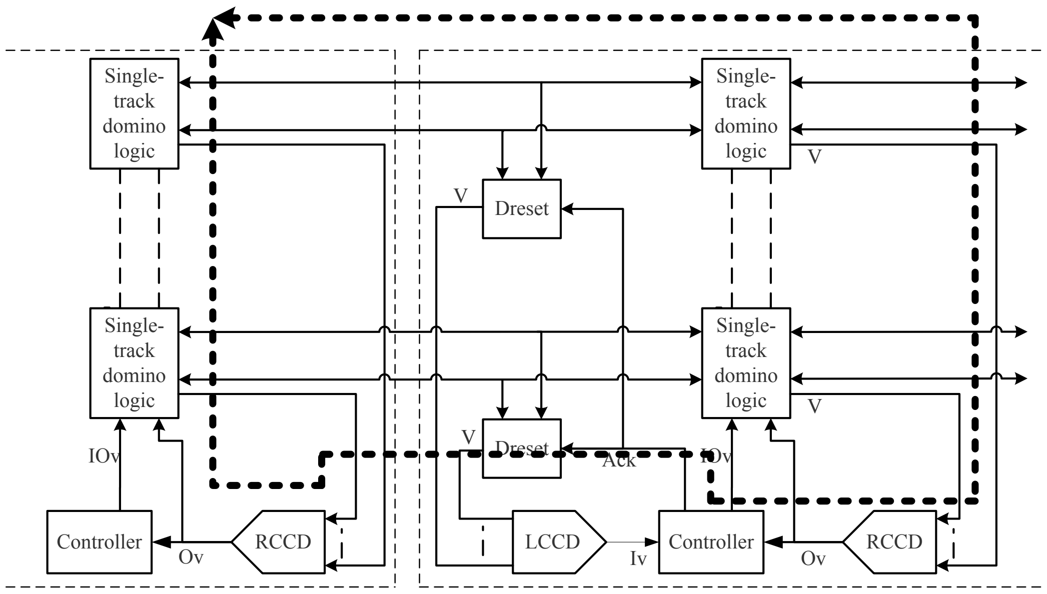
Figure 27. SLST with a buffer and OR function

4.2.2 Performance model

The critical cycle in of channel c between pipelines stages A and B is shown in Figure 35 and can be calculated as

$\tau_c = 2 * T_{logic} + 2 * T_{RCCD} + T_{control1} + T_{control2}$

where T_{logic} is the delay of the logic blocks in the data path, T_{RCCD} is the delay of the right channel completion detector, $T_{control1}$ is the delay of the controller at the receiver pipeline stage and include the delay required to drive the input wires low. $T_{control2}$ is the controller delay at the sender side to drive the control signal responsible for evaluation of the sender high. For a linear pipeline with only 1 channel between sender rand receiver pipelines stages ($T_{RCCD} = 0$) this cycle time is estimated to 10 transitions.



Single-track domino logic
Single-track domino logic
Single-track domino logic
Single-track domino logic
V
Dreset
V
Dreset
V
V
IOv
Ack
IOv
Controller
Ov
RCCD
LCCD
Iv
Controller
Ov
RCCD

Figure 28. SLST template with critical cycle

4.3 Multi-level Single Track template

The second proposed template is designed for medium performance applications where a pipeline stage can have multiple levels of logic thus effectively sharing the control area overhead over several levels of logic. The motivation is to have a design template that can support a wide range of pipeline granularity giving the designers flexibility in terms of performance and control overhead.

4.3.1 Bundled valid wire

It is observed that for medium performance applications we can have larger pipeline stages where a subset of outputs of sender pipeline stage goes to the same receiver. In such scenarios the cost of completion detectors can be shared among the two pipeline stages; for example consider a linear pipeline with stages A and B as shown in Figure 29. Here all the outputs of stage A acts as the input to B, so along with outputs from A another 1-of-1 single-track valid wire can be sent from stage A to B removing the need for completion detection of these individual inputs at B.

In particular, when all the dual-rail outputs from A to B are valid this 1-on-1 wire will be driven high acting as a request signal for that particular outputs. After receiving the valid wire and all the outputs are valid stage B will acknowledge by driving the valid wire and dual-rail inputs low. Stage A will detect the neutrality of this valid wire and assume that all the dual-rail outputs between stage A and B are also driven low. Use of this valid wire eliminates the need of the completion detector in stage B.

This valid wire may typically arrive later then the actual inputs but it doesn't stop the stage B to evaluate this thus causes no degradation in forward latency. Additionally there is no bundled timing data constraint associated with this wire as inputs are acknowledged only when the valid wire arrive and all outputs of the receiver are valid. By reducing the number of completion detectors on the dual rail wires thus effectively reducing the capacitive load the forward latency will only be better compared to the earlier proposed template.

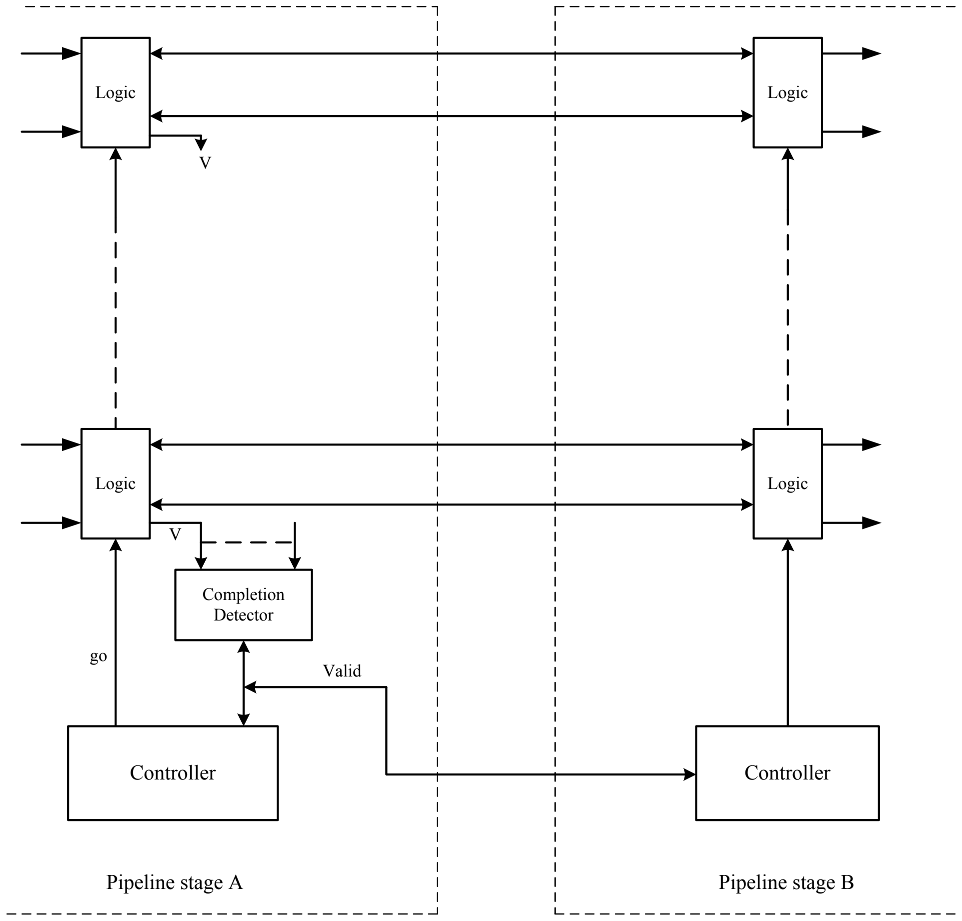


Figure 29. Linear pipeline with 1-of-1 valid signal

4.3.2 Block level diagram

Figure 30 shows the block level diagram of the proposed design template. V_L is 1-of-1 single-track valid wire associated with the dual-rail inputs and V_R is 1-of-1 single-track valid wire associated with dual rail outputs. *V* is the output valid generated by the single-track domino logic detecting the validity of the dual-rail outputs while *OCv* is the output channel valid signal generated by completion detector (PCCD). In case of non-linear pipelines we will use C-element trees to combine 1-of-1 valid signals at the input and OCv signals at the output.

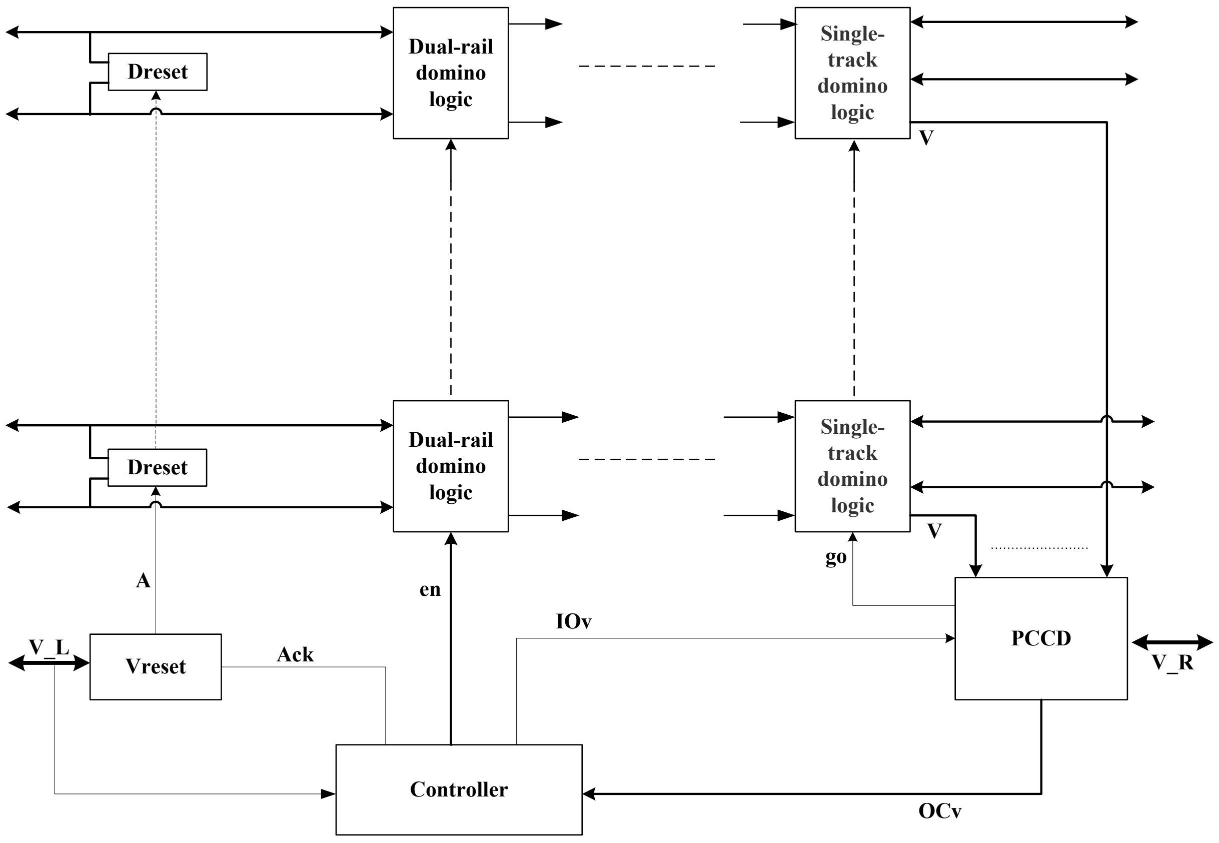


Figure 30 Block level diagram of MLST template

Another key distinction between SSTFB and MLST is in what constitutes a channel. In the SSTFB template every dual rail single-track wire between two pipeline stages acts as a channel, while in this proposed template a bundle of dual rail single-track wires along with the 1-of-1 valid single-track wire between a pair of sender and receiver constitutes a channel. This distinction is important because the proposed template bundles all dual-rail outputs going to the same pipeline stage into one channel which effectively reduces the number of channels between pipeline stages. This also reduces the size of the completion detectors improving not only area efficiency but also performance.

The main components of the design template are explained below.

Data path

Unlike SLST which supports single level of domino logic, this template can support multiple levels of logic where intermediate logic levels can be shared among outputs. The last level of logic is controlled by the "go" signal generated by Pre-charged completion detector (PCCD) while the intermediate levels of logic are controlled by the "en" signal. The use of separate signals to control the evaluation of the last level and the remaining level of logic allows eager evaluation where the intermediate levels of logic do not have to wait for handshake at the output to finish before being ready to evaluate again. This strategy helps in improving the average latency and average throughput of the system.

As shown in Figure 30 the intermediate levels of logic use conventional domino logic to generate dual-rail signals, last level of logic is responsible for generating the dual-rail single-track outputs as well as the generating the validity signal (*V*) indicating the validity of the data on the dual-rail single-track wires.

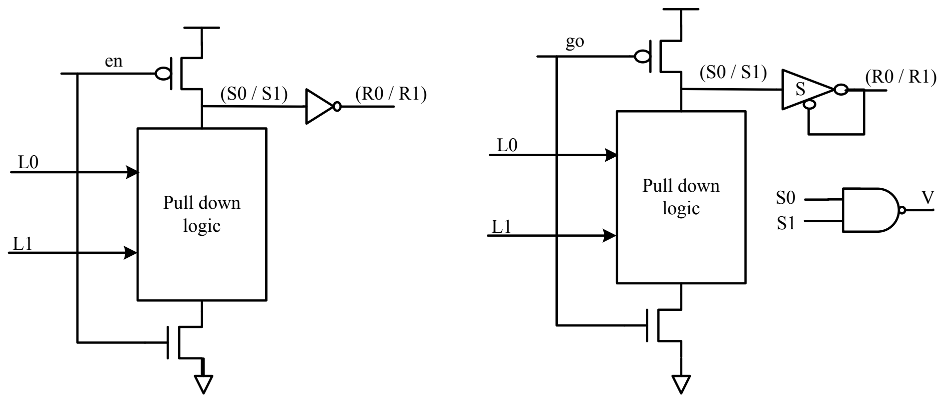


Figure 31. Intermediate and final level of logic

Pre-charge completion detector (PCCD)

Pre-charged completion detector is used to detect the validity of the dual-rail outputs associated with a channel and subsequently generating the 1-of-1 single-track valid wire. It uses a dynamic AND gate to detect the validity of outputs. Figure 32 shows a transistor level diagram of an eight input dynamic AND gate detecting validity of 8 outputs and generating valid wire (V_R). When all the dual-rail single-track outputs are valid, signals (S0 and S1) are driven low driving the single-track wire V_R high. In the mean time output channel valid signal (OCv) is driven high indicating the validity of the channel and OCv is driven low when 1-of-1 valid signal goes low. The PCCD is also responsible for generating the "go" signal which controls the evaluation / pre-charge of the last level of logic as well as evaluation / pre-charge of the PCCD. When V_R signal is driven high go signal is driven low which then pre-charges the last level of the domino logic and the dynamic AND gate and go signal is driven high when all the output channels of the pipeline stage are driven low.

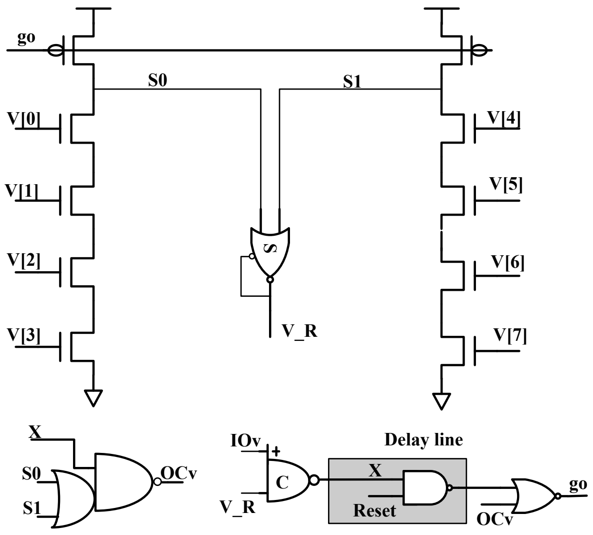


Figure 32. Transistor level diagram of PCCD

Vreset and Dreset

The Vreset and Dreset blocks are responsible for driving the 1-of-1 valid wire and dual-rail wires low. It accepts the Ack signal from the controller to generate the A signal which then drives the 1-of-1 valid channel and dual-rail inputs low. Figure 33 shows a transistor level diagram of a Vreset. The pulse-width on A can be changed by changing the length of the delay line.

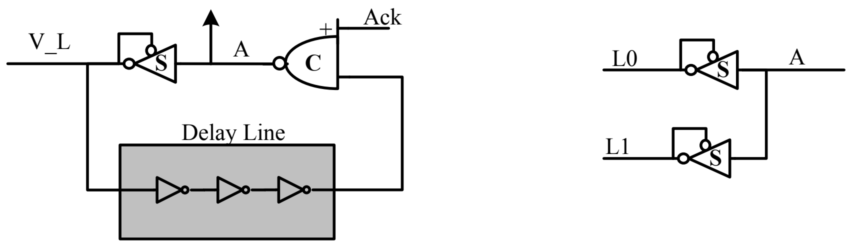


Figure 33. Transistor level diagram of the (a) Valid reset and (b) Data reset block

Controller

Figure 34 shows the transistor level diagram of the controller used. The controller is similar to the controller of SLST proposed in Section 4.2 with additional responsibility of generating en signal which controls the evaluation and **p**re-charge of intermediate levels of logic.

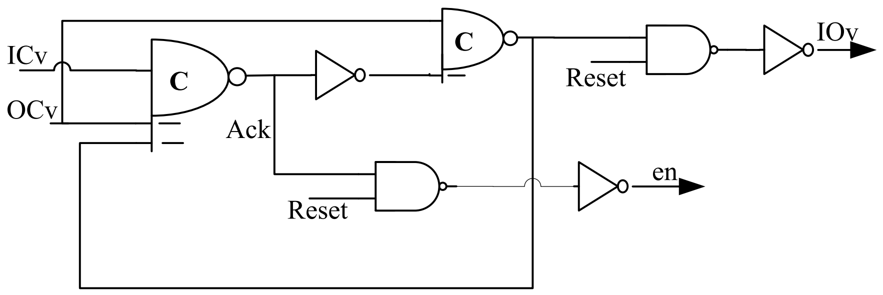


Figure 34. Transistor level diagram of MLST controller

4.3.3 Performance model

The critical cycle in of channel c in a linear pipeline between pipelines stages A and B is shown in Figure 35 and can be calculated as

$\tau_c = T_{logic} + L * T_{logic} + T_{PCCD} + T_{control1} + T_{control2}$

where T_{logic} is the delay of the last level of logic in sender and L is the number of levels of logic in the receiver, T_{PCCD} is the delay of the pre-charged completion detector, $T_{control1}$ is the delay of the controller at the receiver pipeline stage and include the delay required to drive the input wires and 1-of-1 valid wire low. $T_{control2}$ is the controller delay at the sender side to drive the control signal responsible for evaluation of the sender high. For non-linear pipelines the cycle time will increase to incorporate the delay of the C-element trees.

4.3.4 Timing constraints

The correct operations of this proposed family of circuits depends on the relative timing of logic gates to avoid critical races. Careful choice of the transistor sizes can avoid these races quite easily. There are two critical races in our proposed template and are explained below

- **Pre-charge of intermediate logic**: In our proposed template the pre-charge of the intermediate levels of logic is caused by the “en” signal being driven low by the controller. The “en” signal is driven back high when the controller detects that left valid signal is driven low. The timing assumption is that this pulse on en signal is enough for the intermediate logic levels to pre-charge. In case of a linear pipeline this pulse is 5 gate delays and in case of non-linear pipelines this pulse will be longer. It is also worth noting that all the intermediate levels of logic pre-charge in parallel and hence we are comfortable that by proper sizing of transistors this timing constraint can be satisfied.

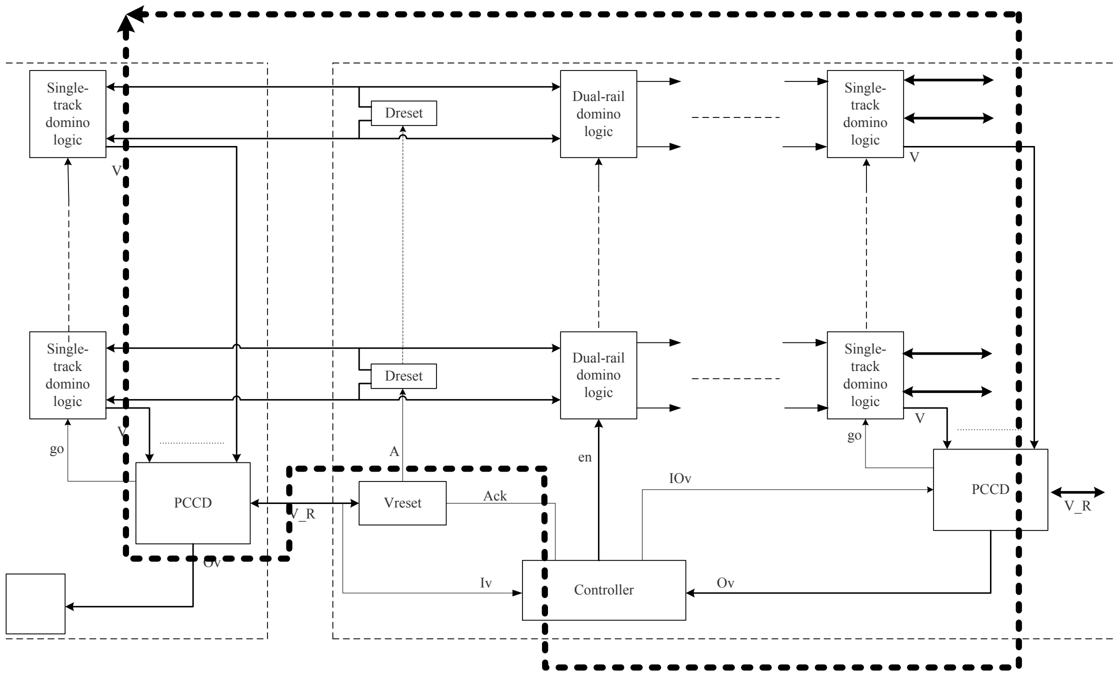


Figure 35. MLST template with critical cycle

- **Reset of dual-rail inputs**: In our proposed template we have assume that if the 1-of-1 valid wire associated with a channel is low then all the dual rail inputs associated with that channel are also driven low. The timing assumption is that the dual rail inputs should be driven low before the receiver detects the neutrality of 1-of-1 valid wire and reasserts en signal and similarly the sender reasserts the go signal for evaluation. In case of linear pipeline this timing race is of type 1 gate delay against 5 gate delay and will only be more relaxed in case of non-linear pipelines. We are comfortable that this timing constraint can be satisfied through proper sizing of the SDOWN gate and / or limiting the wire load on the dual rail input wires.

4.4 Multi-level domino with single-track (MLD-ST) controller

This design template is a variant of the MLST design template proposed in Section 4.3 with the distinction being the use of conventional domino logic used for last level of logic instead of single-track domino logic used. The motivation is to remove the pulse-width constraints required to charge / discharge the dual-rail wires by using static inverters to drive the dual rail wires instead of S gates. Additionally it also improves area efficiency as the data path is composed of conventional domino logic which uses smaller inverter instead of bigger SUP and SDOWN gates.

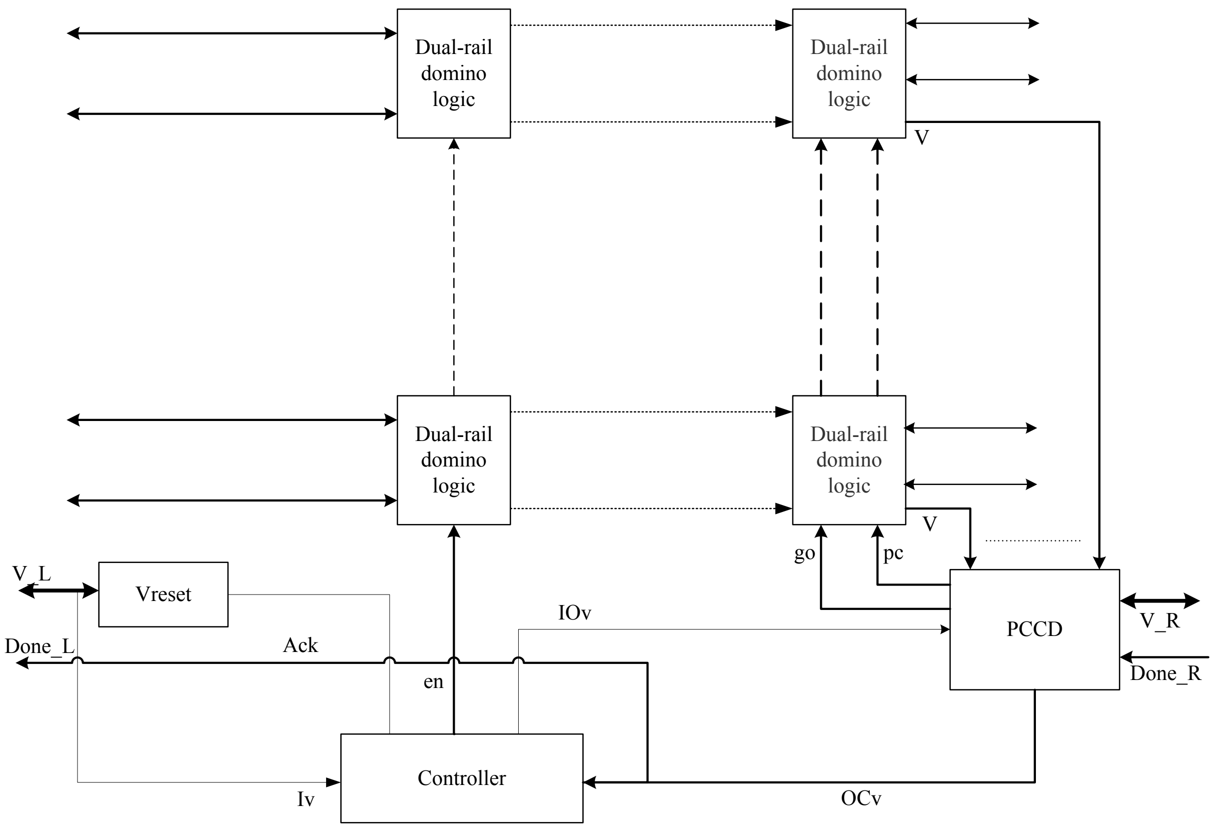


Figure 36. Block diagram of MLD-ST template

The key idea is to use an early acknowledgement signal from the receiver to pre-charge the dual rail domino path of the sender while the sender and receiver complete the single-track handshake protocol on the 1-of-1 single-track valid wire. Figure 36 shows the block diagram of the template with "Done" input signal coming from the receiver side. This "Done" input which is essentially the Output Channel valid (OCv) signal of the receiver which indicates that the outputs of receiver are valid and the inputs are no longer needed, is used to pre-charge then the dual rail data path of the sender as shown. Figure 32 shows the modified PCCD block which accepts the "Done" input and generates the pre-charge signal "pc" which then pre-charges the dual-rail domino path as shown in Figure 38. After detecting a valid token on all the output channels of the receiver the right channel completion detector of the receiver will drive OCv signal high. The NAND gate then drives "pre-charge" (pc) signal low which then pre-charges the dual rail domino path as shown in Figure 38. As the receiver completes the single-track handshake by driving the 1-of-1 valid wire low and the sender is ready to evaluate again pc signal is driven back high.

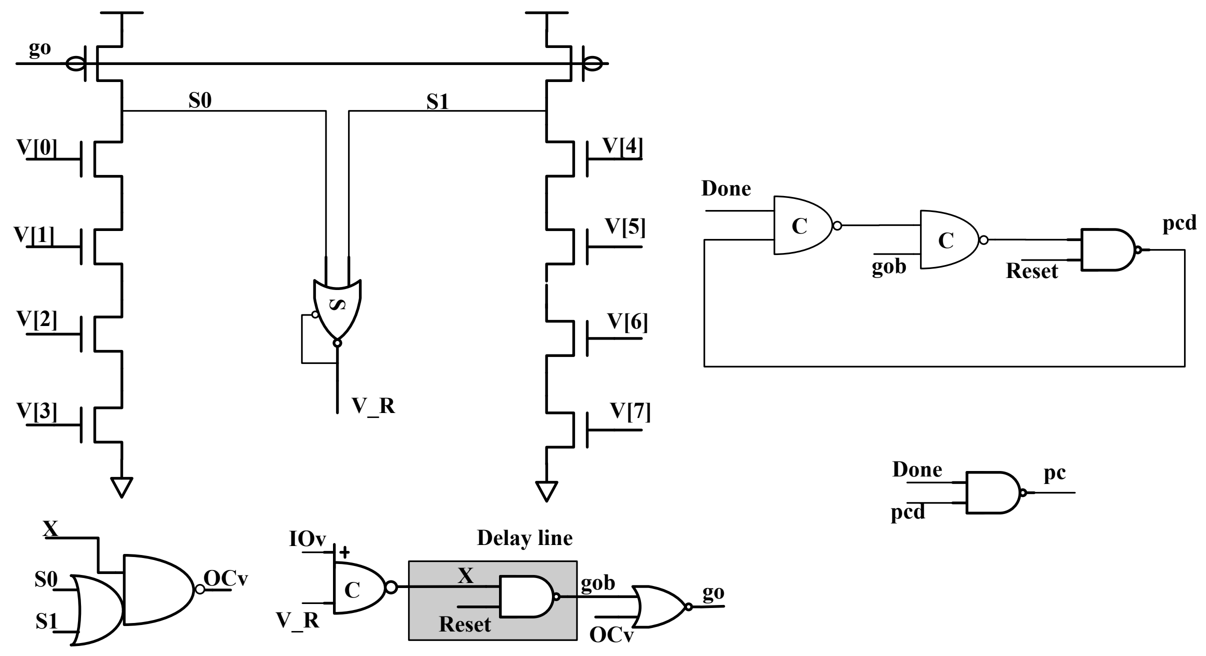


Figure 37. Transistor level diagram of PCCD with Done input

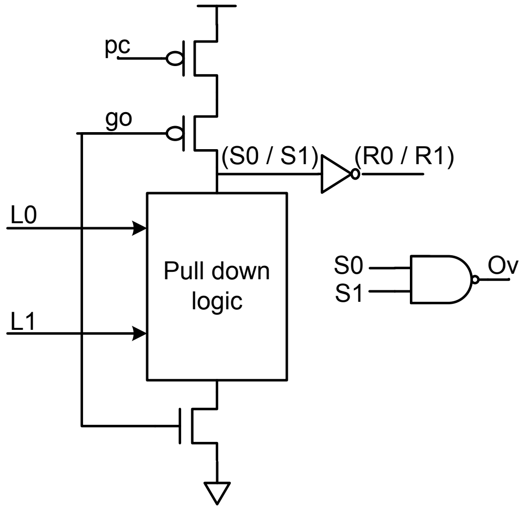


Figure 38. Domino logic used as last level of logic

4.4.1 Timing constraints

The correct operations of this proposed family of circuits depends on the relative timing of logic gates to avoid critical races. Careful choice of the transistor sizes can avoid these races quite easily. There are two critical races in our proposed template and are explained below

1. **Evaluation of receiver:** This timing race is related to the pre-charge of the dual-rail wires before the receiver gets ready to evaluate. The timing assumption is that the last level of domino logic in the sender should pre-charge before the intermediate levels of logic of the receiver are ready to evaluate again. In case of linear pipeline this timing race is of type 3 gate delays against 6 gate delays and will only be more relaxed in case of non-linear pipelines. This timing race can be satisfied through proper sizing of pre-charge transistors.

2. **Evaluation of sender:** This timing race is related to the pre-charge of the dual-rail and evaluation of the sender. The timing assumption is that last level of domino logic in the sender should pre-charge before it is ready to evaluate again. In case of linear pipeline this timing race is of type 3 gate delays against 6 gate delays and will only be more relaxed in case of non-linear pipelines. This timing race can be satisfied through proper sizing of pre-charge transistors.

4.5 Conclusions

In this chapter we have presented three new design styles implementing single-track two phase handshake protocol. We have also discussed design issues with these design styles as well as the critical timing races and their analytical performance model. These design styles provide distinctive advantage in terms of throughput per area and forward latency over existing design styles.

5 Proteus – RTL to asynchronous synthesis tool

5.1 Introduction

Even with all the advantages provided by asynchronous design, they are still not widely accepted by the semiconductor industry. A key roadblock has been the lack of CAD tools which can automate the generation of asynchronous designs. Some methods have been proposed over the years [65][29][21], but they leverage off existing synchronous techniques, resulting in circuits that are bound by the characteristics of their synchronous counterparts. To remove this limitation an automated RTL to asynchronous synthesis tool "Proteus" [23] is being developed at USC Asynchronous CAD / VLSI group. This tool is based on de-synchronization approach for generating asynchronous designs, from an arbitrary HDL representation of a circuit. The tool is design-style agnostic and is thus applicable to many asynchronous design styles. Our main contribution in the development of this tool has been to develop a prototype to convert the generated single rail asynchronous netlist to single-track dual rail netlist, modify the slack matching method for two phase circuits and prototype for generation of testbench generation for verification of final single-track designs.

This chapter is organized as follows, in Section 5.2 we provide background on the other existing synthesis approaches, in Section 5.3 we provide a short explanation of the approach used by Proteus tool to synthesize asynchronous netlist, in Section 5.4 we discuss the flow we used to convert the single rail asynchronous design to our library specific single track dual rail netlist and slack matching steps. In Section 5.5 we discuss about the verification steps to validate the correctness of generated asynchronous design, and in Section 5.6 we discuss the comparison of our library to the other existing styles, specifically we compare the throughput per area of single-track designs to PCHB and MLD templates. Finally we conclude in Section 5.7.

5.2 Background

De-synchronization was introduced in [21] where the key idea is to remove the clock tree completely and generate the clocking signal for each flip flop locally. The clocking signal is locally generated using asynchronous controllers following a handshake protocol. The task of these controllers is to enable the latches and control the flow of data so that the flow of data in the asynchronous netlist is equivalent to the flow of data in the synchronous netlist. The handshake protocol selected uses a delay line to match the worst case delay through the combinational block and also accounts for the margin required to meet setup time constraints of the flip flop as shown in Figure 39.

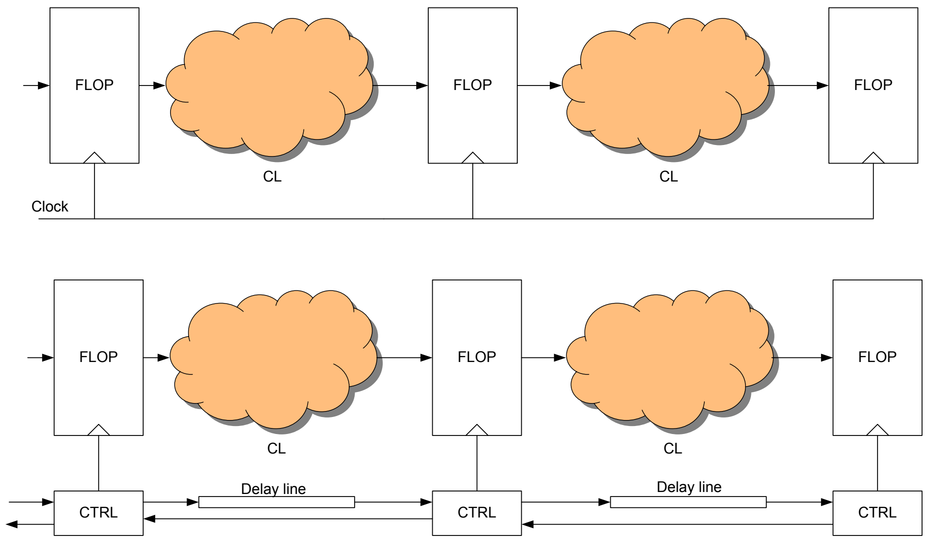


Figure 39. Desynchronization technique

In this asynchronous conversion method the combinational blocks are the same as in the synchronous circuit, making both the conversion and verification of the circuit straight forward. Also, there is no need for a new logic cell library or any logic synthesis. The original synchronous cell library can be used, only a few new cells (such as controllers and latches) are added to the library. As such, this method can be fully automated and use a synchronous flow with minor changes. The limitation of this method is that it only supports bundled data protocol; therefore, we still have to assume worst case delay for each stage. Also, the matched delays should be carefully chosen so that the timing assumptions remain valid in all corners of the implementing process. Finally, slack matching has not been incorporated in this method.

Weaving was introduced in [65] where synchronous netlist is converted into fine-grain micropipeline implementations using a micropipeline library. However the asynchronous conversion is guided by the synchronous architecture generated by the commercial synthesis tools which does not take into account of the synchronization delay in asynchronous design. Moreover weaver flow converts synchronous design into fine-grained asynchronous design which makes them area inefficient for slower performance requirements.

5.3 Proteus

Proteus is an EDA tool developed by Dimou in [23] which converts a synchronous netlist to an asynchronous design. The key idea behind the tool is same as the de-synchronization method discussed earlier. A key difference between the two methods is that while de-synchronization method only supports bundled data protocol, Proteus is more general and can generate asynchronous designs specific to a user defined template. As of now only two templates PCHB [50] and MLD [23] are supported.

Another key difference between the two methods is that in order to achieve better performance at lower area penalty, area optimization technique like clustering of different pipeline stages are performed. For each pipeline stage a single asynchronous controller is instantiated. The clustering is done subject to a given performance constraint such that two pipeline states will be clustered without violating the performance constraint. The performance constraint is generally defined by user in terms of cycle time. After clustering, the tool tries to optimize the performance of the design through slack matching.

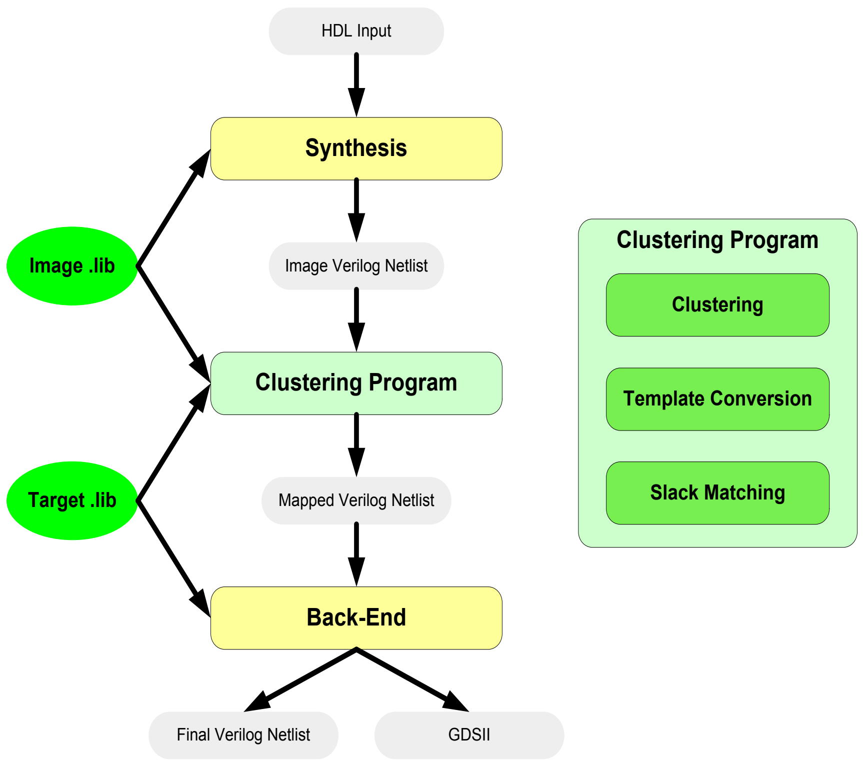


Figure 40. Overview of Proteus ASIC Flow

Figure 40 shows the overview of the design flow in Proteus. The original synchronous netlist is first synthesized using a commercially available high level synthesis tool. The generated synchronous gate level netlist is then converted into an asynchronous netlist using an image library. The image library is use to map the functionality of the asynchronous gates to their equivalent synchronous gate.

The next step is clustering. Initially we start with each gate as a cluster or a pipeline stage. Then, clustering is done by merging two pipeline stages into one and executing one such move at a time. The clustering algorithm is a greedy algorithm where the tool looks at all possible moves that are available at each iteration and executes the one that provides the most area improvement without violating the performance constraint.

5.4 Single-track template conversion and slack matching

After clustering the next step is to convert the single rail asynchronous netlist to the asynchronous netlist using a user defined template. In our case we want to convert the single rail asynchronous design to dual rail single-track asynchronous design. This step involves creation of single-track channels, conversion of single rail signals into dual rail signals and instantiation of control logic blocks associated with each pipeline stage We also need to map the abstract logic gates used by the tool to the logic gates that are present in the single-track library that we developed.

For each pipeline stage we need to instantiate input channel reset blocks (DataAck and ValidAck) and controller (STCONT) to complete handshaking between two pipeline stages. In case of non linear pipeline stages c-elements are instantiated as left /right channel completion detection trees (LCCD / RCCD) to detect the validity and neutrality of all the input and output channels of the clusters. . In case of MLST and MLD-ST templates we also need to instantiate a 1-of-1 valid wire for each channel and for each output channel we need to instantiate a pre-charged completion detector (PCCD).

Due to point-to-point connection of the signals in the single-track handshake protocol, signals are constrained to have a fanout of only one. To handle outputs with multiple fanouts, one solution proposed by Ferretti is to insert special FORK cells [30] between the source and destination pipeline stages. However the addition of these FORK cells can lengthen the critical path increasing the latency of the design.

Addition of FORK cells can also increase the imbalance among the pipeline stages requiring more slack matching buffers thus increasing the area and energy inefficiency of the design.

We propose to solve this problem differently where instead of adding FORK cells, we duplicate the last level of logic in the source pipeline stage thus creating a new output channel and a new dual rail output signal for each fan out. In this fashion we do not increase the length of the path and we also don't increase the imbalance of the pipeline.

The next step involves slack matching which can be thought of as the process of properly aligning the timing of the handshakes between the pipeline stages in the design, so that a circuit can maximize its performance. The tool uses Full-Buffer-Channel-Net [8] to model the slack matching problem and calculate the minimum number of slack matching buffers required to satisfy a given performance constraint. The model used also take advantage of the fact that non homogeneous buffers can provide significant advantages in number of slack matching buffers required. The tool uses a linear programming solver to calculate the number of slack matching buffers required per channel. As a part of this thesis we modified the model adopted in the tool for two phase circuits which is a natural implementation of the formulation based in [8].

5.5 Verification

To check the correctness of the asynchronous design created by the tool, a testbench is generated by the tool from which we compare the results generated by asynchronous design and synchronous design. Figure 41 shows our verification methodology. The tool creates information that is required for verification of the resulting design. A header file is written that contains some important parameters for the design, such as the number of inputs, outputs and vectors that are going to be used. Another file that contains random input vectors is also generated. The original synchronous RTL netlist is simulated with the generated input vectors and the results are stored.

To simulate the asynchronous design the generated input vectors are converted by CSP Bit generator into input asynchronous tokens. Then these input tokens are converted into single-track signals by CSP_to_ST shims. The single-track outputs generated from the simulation of asynchronous designs are then converted into output tokens by ST_to_CSP shim after which the data value is read by CSP Bit Bucket.

The results of this run are compared against the results of the synchronous netlist and if they match the testbench will indicate the validity of the asynchronous design. From the simulation of asynchronous designs we also average the number of tokens received over time to calculate the average throughput of the design.

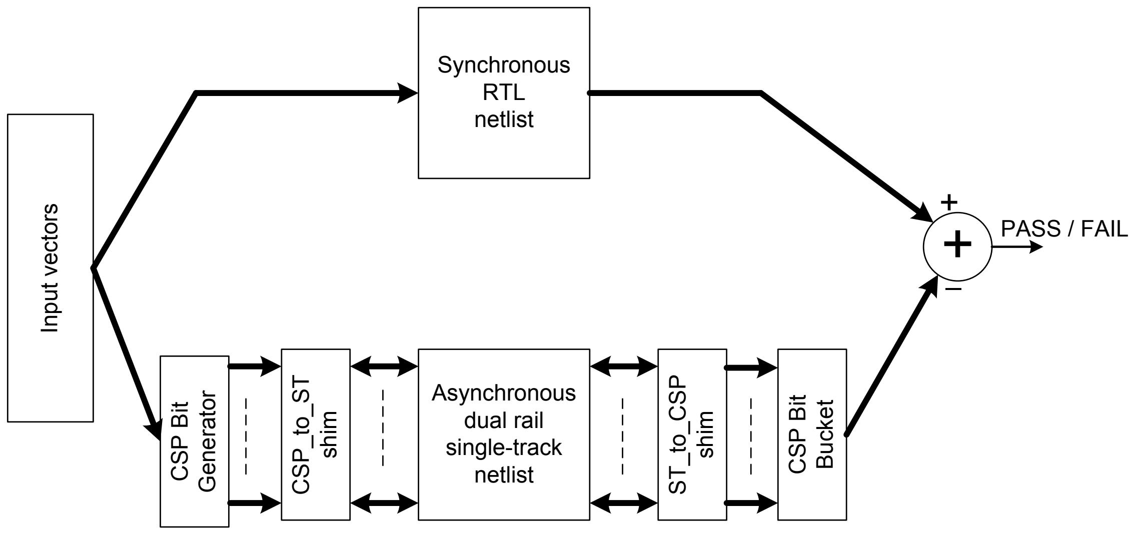


Figure 41. Verification environment for single-track asynchronous design

5.6 Comparisons

In this section we present the comparison results of our proposed single-track templates against other existing design styles like PCHB and MLD. This section is organized as follows, in Section 5.6.1 we present the performance and area model we chose and in Section 5.6.2 we present our results.

5.6.1 Performance and area model

The performance of the design is estimated from the average throughput measured through the Verilog simulations. We assume that each transition has an equal unit delay. The area of the design is estimated by calculating the number of transistors in the design. For simplicity we neglect the interconnect area.

5.6.2 Experimental Results

We have illustrated the advantages of the proposed design styles on various examples from the ISCAS benchmark set that includes examples that are purely combinational (hence include no cycles) or mixed sequential and combinational (include state elements and cycles). We also added a few examples that are generally common in commercial circuit designs. We compare the throughput per area of the proposed template with industry standard PCHB and MLD templates.

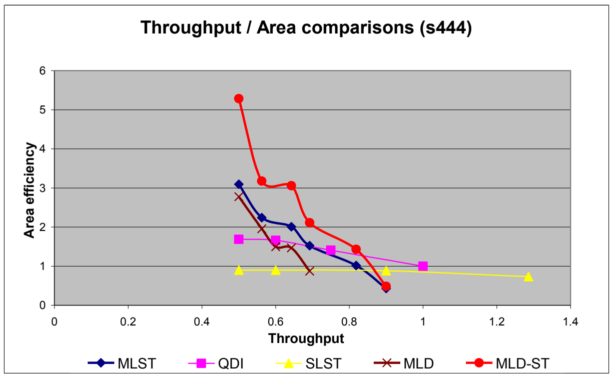


Figure 42. Throughput / area tradeoff curves for s444 example

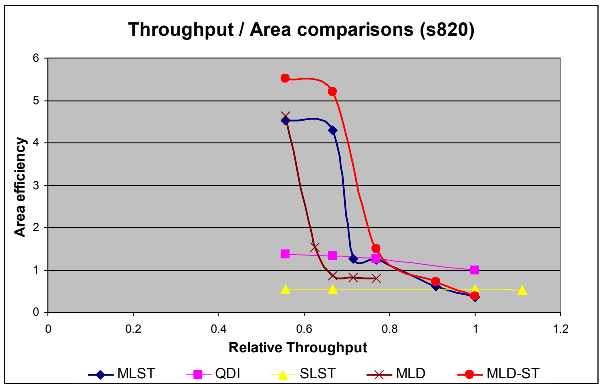


Figure 43. Throughput / area tradeoff curves for s444 example

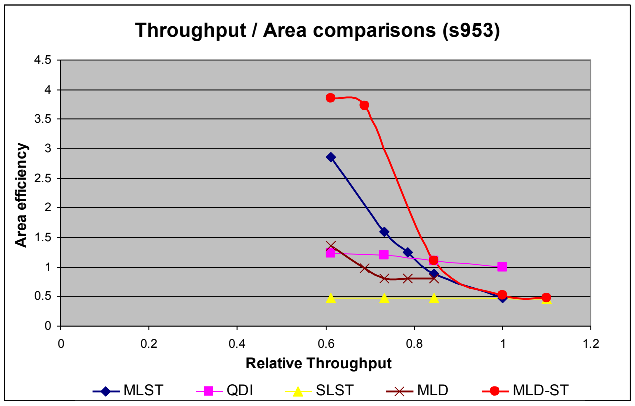


Figure 44. Throughput / area tradeoff curves for s953 example

In Figure 42 to Figure 44 we compare the throughput area trade off curves for SLST, MLST and MLD-ST against PCHB and MLD templates for three ISCAS examples. We varied the target performance constraint and compared area. It can be seen that for high performance targets, SLST is more area efficient than PCHB. For medium performance targets, SLST and PCHB blocks can reduce area only be reducing the number of slack matching buffers. However they still suffer a large control area overhead. Multi-level templates like MLST, MLD-ST and MLD, on the other hand, can improve area efficiency by taking advantage of clustering and the sharing of control logic across multiple levels of logic, thus effectively reducing the control area overhead. With the high performance nature of single-track templates, for the same target performance MLST and MLD-ST can support bigger pipeline stages providing better throughput / area compared to four-phase MLD templates. In Figure 42 to Figure 44 we just show throughput area trade-off curves for three examples but similar behavior was experienced in all ISCAS examples.

In Table 2, we compare the throughput per area of MLST, MLD-ST template against PCHB and MLD template. We set the target cycle time constraint for PCHB template to 18 transitions as these templates are designed to be area efficient for 18 transitions. For MLST, MLD-ST and MLD templates we have selected a representative target performance constraint to be a somewhat slower 26 transitions (~800 MHz in 65nm) It can be seen that at 26 transitions MLST provides 32% better throughput per area over PCHB templates and 75% better throughput per area over MLD template. MLD-ST template provides 2X better throughput per area over PCHB and 2.76X over MLD templates.

Note the throughput / area metric is a function of the target performance constraint we set. As we decrease the target performance requirement, the multi-level templates improve area efficient because of its flexibility of having bigger pipeline stages as was seen in Figure 42 through Figure 44. Also note that the comparison between MLST and PCHB templates is a function of the clustering algorithm and number of primary inputs in the design. Due to the point-to-point limitation of single-track wires, additional buffers (FORKs) are inserted between primary inputs and computational logic. These additional buffers are added after the clustering step and are hence not optimized. This effectively makes our comparison numbers somewhat conservative as a better clustering algorithm designed for single-track template could try to cluster these *FORK* buffers which would improve their relative efficiency.

Design	**Cycle time (T)**	**Area (A)**	**Throughput comparison against PCHB**	**Area comparison against PCHB**
			(T_{PCHB}/T_{SLST})	**(A_{SLST}/A_{PCHB})**
s298	14	7050	1.285714286	1.290972349
COUNTER	14	3764	1.285714286	0.799830004
s27	14	889	1.285714286	0.953862661
s344	16	8119	1.125	1.24162716
s349	16	8119	1.125	1.265627436
s386	14	8000	1.285714286	1.628664495
s400	14	10471	1.285714286	1.487146712
s420	14	9305	1.285714286	1.21729461
s444	14	9779	1.285714286	1.365019542
s510	17	12000	1.176470588	1.21703854
s526	14	11159	1.285714286	1.31174327
s641	16	11000	1.375	1.411161001
s713	16	14094	1.25	1.621118012
s820	18	20000	1.111111111	1.67196121
s832	16	18000	1.25	1.497628754
s838	14	21854	1.285714286	1.336390876
s953	18	24000	1.222222222	1.608471282
S1196	18	53674	1	1.532535762
S1238	18	48000	1.111111111	1.574389924
S1423	40	32541	0.75	1.607836356
C3540	14	65000	1.285714286	1.502612234
HIT_DETECT	14	100000	1.285714286	1.161156977
MAC32	18	90000	1.222222222	1.618850616
C7552	20	32000	0.9	0.585480094
			1.198374767	1.354517495

Table 1. Throughput and Area comparisons of SLST against PCHB

5.7 Conclusions

In this chapter we propose a synthesis flow for the proposed non-homogeneous single-track design templates using academic tool *Proteus* and compare them with existing design styles like PCHB and MLD. The synthesis flow takes the advantage of commercial synthesis tools and then performs re-pipelining to generate area efficient designs. Comparisons on several ISCAS benchmarks show that the proposed templates provide a wide range of throughput area tradeoffs. They all have very good latency characteristics, using domino logic with no explicit latches or bundling constraints. For high performance applications, the SLST template provides an average of 20% better throughput than PCHB pipelines at the cost of 36% more average area. At lower frequencies MLST template provides 1.32X better average throughput / area over PCHB and 1.75X over MLD pipelines. On the other hand MLD-ST provides 2X advantage better throughput / area over PCHB and 2.75X over MLD pipeline styles.

Design	**Area – MLST (A)**	**Area MLD-ST (A)**	**MLST vs PCHB (T / A)**	**MLST vs MLD (T / A)**	**MLD-ST vs PCHB (T/A)**	**MLD-ST vs MLD (T / A)**
s298	2491	1533	2.077	3.375	3.621	5.884231126
COUNTER	3072	1982	1.451	2.249	1.534	2.377821446
s27	886	731	0.728	0.883	0.554	0.671682627
s344	4080	2835	1.311	1.887	1.924	2.768574635
s349	4080	2835	1.286	1.851	1.924	2.768574635
s386	1713	1239	1.985	2.745	4.402	6.086732477
s400	3744	2680	1.302	1.819	2.392	3.340987371
s420	3974	2782	1.332	1.902	1.821	2.601725377
s444	4704	3394	1.054	1.461	1.747	2.421921037
s510	2173	1707	3.49	4.443	6.866	8.740931008
s526	4085	2683	1.442	2.195	2.529	3.850769804
s641	9512	7619	0.693	0.866	1.045	1.30443123
s713	9293	7482	0.72	0.894	0.989	1.228178397
s820	9681	8011	0.95	1.149	1.88	2.271338448
s832	7480	6278	1.236	1.473	2.277	2.712659593
s838	8831	6391	1.19	1.645	1.935	2.674296444
s953	11940	8030	0.982	1.46	1.894	2.816296033
s1196	16472	14047	1.32	1.548	1.775	2.081048107
s1238	11296	13318	1.928	1.635	2.502	2.122535559
s1423	13819	8512	0.845	1.372	0.915	1.485748626
c3540	36488	30524	0.762	0.911	0.946	1.130744894
HIT_DETECT	19299	13495	2.51	3.59	2.118	3.028297518
MAC32	33727	33727	1.133	1.133	2.337	2.337424393
c7552	52637	52637	0.584	0.584	0.956	0.95553508
c5315	39310	34126	0.737	0.849	1.264	1.456285949
			1.322	1.757	2.086	2.764750872

Table 2. Throughput / Area comparisons of MLST and MLD-ST

6 Library characterization flow

6.1 Introduction

Semi-custom standard-cell based design methodologies offers good performance with typically 12-month design times. They are supported by a large array of evolving CAD tools that support simulation, synthesis, verification, and test. A large library of standard-cell components that have been carefully designed, verified, and characterized supports the synthesis task. This library is generally limited to static CMOS gates because they are robust to different environmental loads and have high noise margins, thus requiring little block-level analog verification. But the time to market advantage of standard-cell based design is being attacked by the increasing difficult task of estimating wire delay and increasing process variability making worst case design overly conservative. These limitations have created an opportunity for alternative circuit styles, such as asynchronous design (e.g., [13][51][40][32]).

In particular, Fulcrum Microsystems has demonstrated the commercial viability of high speed asynchronous design. Their chips have asynchronous cores with approximately 2X the performance of standard-cell-based synchronous design using overly conservative design style called quasi-delay-insensitive (QDI) [33][50]that costs in area and limits performance. Moreover, Fulcrum Microsystems relies on a semi-automated full- custom flow that includes the development of a liquid library and time-consuming analog simulation for performance verification. In conjunction with a larger effort at Columbia University, USC has been for past five years exploring a semi-custom approach to designing such high-speed asynchronous designs as well as alternative circuit styles that tradeoff robustness with performance. Ferretti et.al have recently developed single-track full-buffers (STFB), which have 3X the performance of quasi-delay-insensitive circuits and are 50% smaller [31]. Ferretti demonstrated the advantages of our STFB library and the standard-cell flow on a 260K-transistor test chip that includes a 64 bit adder and test circuitry in TSMC 0.25u technology [32]. The chips worked flawlessly at a performance of over 1.4 GHz over a wide range of voltages and temperatures, giving more than 3X faster than most standard-cell based ASIC designs in this process. At that time, however, our design process was immature and required extensive and time consuming block-level analog performance and timing verification.

The next step in the evolution of this standard-cell-based design flow is to determine if we can properly model these cells in a standard HDL, effectively characterize them in a standard library format, and enable accurate back-annotation using a standard back-annotation flow, thereby enabling digital simulation-based performance and timing verification. These questions are particularly interesting for STFB designs that use a non-standard single-track protocol that involves complicated tri-state logic and several atypical timing constraints that must be modeled. This paper answers this question in the affirmative. In particular, it describes how STFB cells can be modeled in Verilog, what timing arcs must be characterized to enable accurate performance and timing verification, and how these timing arcs can be represented in the commercially standard Liberty format. Moreover, it successfully shows how this characterization enables performance verification and simulation based validation of the timing constraints using back-annotated Verilog on several designs including test chip. Experimental results show that the modeling error is less than 5%.

The remainder of this paper is organized as follows. Section 2 reviews asynchronous channels and STFB templates and semi-custom design flow. Section 3 describes modeling STFB cells in Verilog, timing arc definition in the Liberty format, and back-annotation using SDF. Section 4 presents the experimental results and Section 5 draws some conclusions.

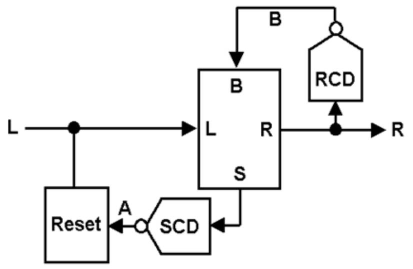
6.2 Background

This section first describes asynchronous channels, the basic structure and operation of single-track full-buffers (STFB). It then introduces the Standard Delay Format (SDF) and Liberty Format (.lib).

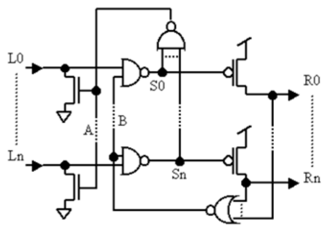
6.2.1 Single Track Full Buffer (STFB) Template

Figure 45 shows a typical STFB cell's block-diagram and more detailed transistor level implementation of a STFB buffer with a single 1-of-N input and output channel. In the STFB buffer, the NOR gate with inputs R0 and R1 is the Right Completion Detector (RCD) and the NAND gate with inputs S0 and S1 is the State Completion Detector (SCD). The operation of a STFB cell when a token arrives at the input can be partitioned into two cases depending on the state of the output channels:

Output channel is free: The cell is waiting for a token from left environment. The RCD sets the "B" signal high indicating that the output channel is free and enabling the processing of the new input token. The token arrives at the left channel and then the state signal "S" is lowered thus creating an output token for the right environment. The SCD then asserts the "A" signal on detection of the change in state "S". The "A" signal in turn removes the token from left channel through reset block. The presence of output token on the right channel resets the "B" signal which will restore the state signal "S" and disable the stage from firing while the output channel is busy even if a new token arrives at the left channel.



(a) Block diagram



(b) Schematic

Figure 45. Block diagram and transistor level schematic of a STFB buffer

Output channel is busy: In this case, when the token arrives at the left environment, the RCD sets the "B" signal low indicating that the output channel is busy and thus disables the processing of the new input token. The new token waits for the output channel to be free at the left channel. When the right environment consumes the token, the RCD sets the 'B" signal high indicating that the output channel is free and thus enables the processing of the new input token. The state signal "S" is lowered, thus passing the token to the output channel. The SCD asserts the "A" signal on detection of change in state "S". The "A" signal in turn removes the token from the left channel through the reset block. The presence of the output token will reset the "B' signal which in turn restores the state signal "S".

The cycle time of this circuit family amount is only 6 gate delays, which enable ultra-high speed performance.

6.2.2 Asynchronous ASIC design flow

The asynchronous ASIC design flow shown in Figure 46 was proposed by Ozdag et al. [54] and adopted for STFB-based designs Ferretti et al. [32]. After creating a standard cell library, they successfully adopted a largely conventional standard-cell ASIC back-end design flow using conventional place and route tools for physical design [32][54]. Performance and timing verification is performed through transistor level extraction of the layout followed by analog simulation using Synopsys's Nanosim©. This is in contrast to conventional flows that use library characterization, back-annotation, and static analysis for performance and timing verification.

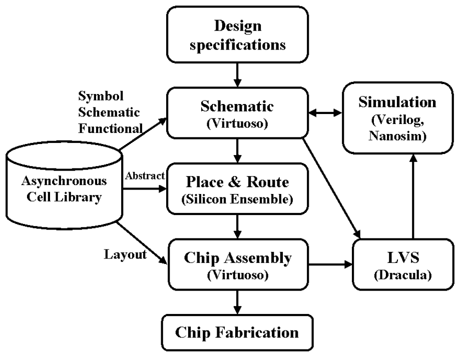


Figure 46. Asynchronous ASIC design flow

6.2.3 Standard Delay Format (SDF)

In conventional synchronous design flows, an SDF file [66] is often used to store the timing data generated by EDA tools for use at many stages in the design and verification process. The SDF file is tool-independent and can include delays, timing checks, timing constraints, timing environments, as well as scaling and technology parameters.

6.3 Liberty Format

The Liberty Format [73] is a standard format used to characterize timing and power consumption properties of a standard cell library. Liberty format supports many timing models including the linear delay model, table-lookup model, scalable polynomial delay model, and piecewise linear delay model. In our work, we adopted the three term linear model for characterizing delay, which provides a reasonable tradeoff between accuracy and simplicity. Of course, more accurate models can also be used and are expected to provide more accurate estimate of the delays at the expense of a somewhat more time-consuming library characterization process. In the linear model the delay of the cell is modeled as the sum of the intrinsic delay, delay due to the load capacitance and delay due to the input slope.

6.4 Proposed Design Flow

The key area this paper addresses is timing and performance verification in asynchronous ASIC design flows. The transistor level extraction and analog simulation required by the ASIC design flow proposed by Ozdag et. al. and Ferretti et. al. [32] is very computationally expensive for large designs and can often represent a significant part of the design cycle. Figure 47 shows the proposed enhanced back-end design flow that includes library characterization and SDF back-annotation to begin to address this problem.

In particular, an SDF file is generated after P&R using a characterized cell library and is used to simulate the design in Verilog. Compared to transistor level extraction, SDF file generation is far less computationally intensive because all transistor level details have been abstracted in the characterized library. Moreover, compared to analog simulation, Verilog simulation is orders of magnitude faster with modest cost in accuracy. One key application of this back-annotation is simulation-based performance verification.

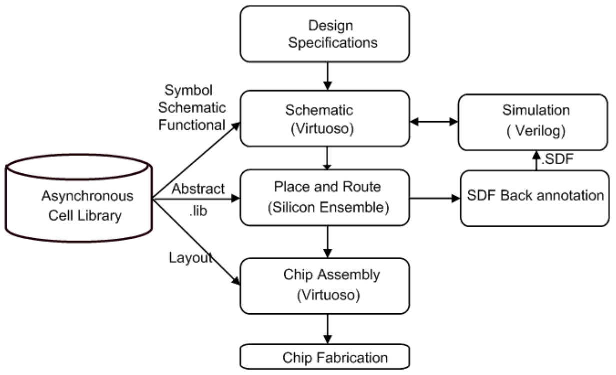


Figure 47. SDF back annotation design flow

Another application of this flow is timing verification, particularly for pipeline templates that involve intra-cell and inter cell timing constraints. There are two basic related tasks: the first is to identify the timing constraints for a given template and the second is to verify that in a post-layout circuit the timing constraints are satisfied with a comfortable margin. Traditionally the identification step is done by hand and the analysis step is done with post-layout spice level simulations or formal verification techniques [20]. Because some templates may be quite complex an important area of research is automatic timing constraint identification [67]. Some tools for the identification and simplification of relative timing assumptions for complex asynchronous circuits have been developed [20][67][47][80]. Once these timing constraints are generated we can verify them using SDF back-annotation based simulation shown in the proposed design flow which is much faster than transistor level simulation.

6.4.1 Identifying the Timing Arcs

For characterizing the timing information of cells in STFB library we need to identify a set of timing arcs that can completely model the performance and timing assumptions of STFB cells. As explained earlier the behavior of a STFB cell can be partitioned into two cases.

The right channels are free and a token arrives at the left environment. Since all the channels are bidirectional (tri-state) wires, all the transitions are either from high impedance to logic state or logic state to high impedance. In this case, for the right channel to go high the timing arcs corresponding to it will be from L to R and the transition is L going from high impedance (Z) to 1. Similarly after R goes high the left channel resets itself through signal A, so the timing arc corresponding to it will be L to L with the transition at L going from high impedance to 0. After L goes low, the right channel tri-states itself and the left channel using signal B and A respectively. So in this case the timing arcs are R to R (1 => Z) and R to L (0 => Z) respectively as shown in Figure 48.

The output channels are busy and a token arrives at the input channels. In this case the token at the input channels will have to wait for the output channels to be free. So in this case the triggering event is output channels getting free and this can be characterized by two new timing arcs and two timing arcs that are in common with case 1. For processing of the input token, the new timing arc is R => R (Z => 1), for resetting of the input channels the new timing arc is R => L (Z => 0) and, for tri-stating of the input and output channels the common timing arcs are R => R (1 => Z) and R => L (0 => Z).

So the timing arcs for both the cases are as shown in Figure 48:

L => R (Z => 1) R => R (Z => 1)

L => L (Z => 0) R => L (Z => 0)

R => R (1 => Z) R => L (0 => Z)

These same arcs exist for all N data lines associated with the 1-of-N channel. Thus, for a dual rail STFB Buffer, there are 6 unique timing arcs per rail (shown in Figure 48) and 12 timing arcs in total.

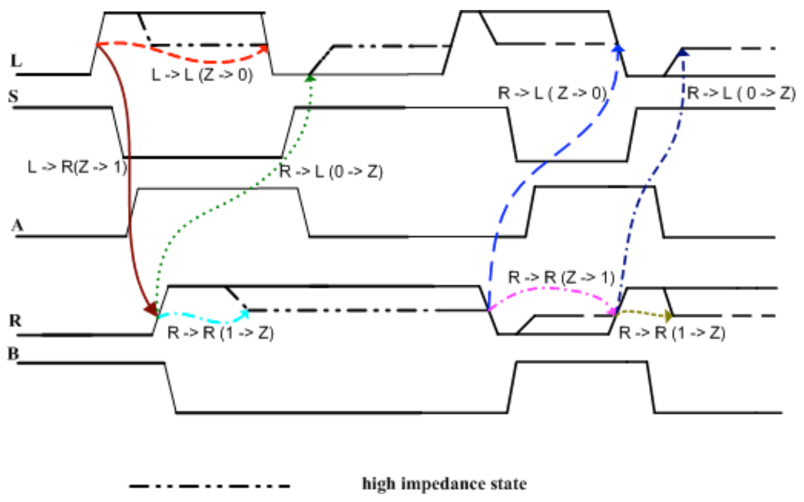


Figure 48. Timing arcs on a single rail of STFB buffer

6.4.2 Characterization Methodology

Because the input and output ports of the cell under consideration are bidirectional and must tri-state, in our simulation environment the left and right environment are controlled using similar STFB cells. The load for the cell under consideration is varied by changing the length of the interconnect wire thus varying the interconnect capacitance. The delay values are then fitted into a plane of form as follows.

$$Delay = A * C_L + B * Slew_{in} + D_{int} \qquad (1)$$

$C_{L,}$ $Slew_{in}$ and D_{int} indicate the load capacitance, the input slew, and the intrinsic delay of the cell, respectively. For the transition going to high impedance, we measure the delay from 50% of L to the delay of signal "A" going to the threshold voltage when transition L goes from 0 to Z. Note that determining if and how commercially available library characterization tools can be used to automate this process is still an open question.

6.4.3 Functional Description of Cells

For SDF back annotation to be possible, we have to specify the pin to pin delays in the functional view of the cells. In Verilog-HDL we can describe the pin to pin delays in the specify block [76]. For example the functional description of a STFB Buffer will look like as shown in Figure 49, where (L0 => R0) means that a timing path exist from L0 to R0 and the six delay values in parenthesis correspond to 0 -> 1, 1 ->0, 0 -> Z, Z -> 1, 1 -> Z, Z -> 0 transitions respectively. Because each *specify* line handles all transitions between unique input and output pins, there are fewer lines than timing arcs. In particular, for a STFB Buffer we need 8 total lines ignoring global reset. It is also important to note that this timing specification is independent of the description of the behavior of the cell, which simplifies the Verilog specifications.

There are many ways to model the behavior of STFB cells in Verilog. We developed a consistent approach in which the Verilog models all have three *always* blocks for each cell. The first always block is activated whenever a new input comes at the input channels and the output channels are free, the second always block is activated when the output channels consumes the tokens present at the input channels and then drive both the input and output channels to high impedance state. The third always block is used for the global reset signal. Since the input and output channels are bidirectional we have to declare them as trireg variables. Unfortunately, trireg variable cannot be assigned inside an always block. To solve this problem we have to declare a temporary register variable and assigned it in the always block, and then using standard Verilog primitives like *bufif* which model a tri-state buffer to transfer the value to the trireg variables [76]. To tri-state a *trireg* variable, we just assign the enable input of the *bufif* gates to 0. Using these standard primitives solves the bidirectional problem and it also does not affect the functionality or timing of the cells in any case.

6.4.4 Timing Arcs in Liberty Format

We use the standard Liberty format to represent the characterized asynchronous cell library. We are using *generic_cmos* model which means the linear delay model to represent the delay values. Since all the transitions in STFB templates are from tri-state to logic or from logic to tri-state, we have to use *three_state_enable* and *three_state_disable* constructs, where *three_state_disable* is a construct used to model the delay leading to the transitions (0 -> Z and 1 -> Z) and *three_state_enable* is used to model the delay leading to the transitions (Z -> 1 and Z -> 0). A timing model for timing arc L->R (Z->1) is shown in Figure 48.

6.5 Experimental Results

The proposed back-annotation flow was evaluated on asynchronous linear pipelines with variable number of stages and a 260 K transistor 64-bit parallel prefix asynchronous adder with the input and output circuitry to feed the adder and sample the results [32]. The SDF back annotation flow is also used to verify if these designs meet the timing constraints of STFB [31].

```
module STFBBuffer(L0, L1, R0, R1, NReset);
inout L0, L1, R0, R1;
input NReset;
trireg L0, L1, R0, R1;
reg a,b,c,d,enable1, enable2 ;

bufif  b1(L0,a, enable1);
bufif  b2(L1,b, enable1);
bufif  b3(R0,c, enable2);
bufif  b4(R1,d, enable2);

specify
  (L0 => R0) = (0, 0, 0, 2, 0, 0);
  (L0 => L0) = (0, 0, 0, 0, 0, 3);
  (R0 => L0) = (0, 0, 3, 0, 0, 4);
  (R0 => R0) = (0, 0, 0, 3, 3, 0);
  (L1 => R1) = (0, 0, 0, 2, 0, 0);
  (L1 => L1) = (0, 0, 0, 0, 0, 3);
  (R1 => L1) = (0, 0, 3, 0, 0, 4);
  (R1 => R1) = (0, 0, 0, 3, 3, 0);
 endspecify

always @(L0 or L1)
wait( (L0 | L1) && (R0 == 0 && R1 == 0) && NReset == 1))
begin c <= L0; d <= L1; a <= 0; b <= 0; enable1 <= 1; enable2 <= 1; end

always @(R0 or R1)
wait (R0 == 1 | R1 == 1)  begin  enable1 <= 0; enable2 <= 0; end

always @(NReset)
begin
if (NReset == 0) begin a <= 0; b <= 0; enable1 <= 1; end
else enable1 <= 0;
end

endmodule
```

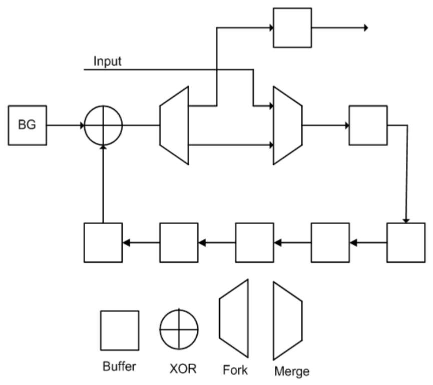
```
library(STFB) { // Start Library
technology (cmos) ;
delay_model : generic_cmos ;
..
cell(STFBBuffer) { // Start Cell
  pin(L0) {
   ..
  }
  ..
  pin(R0) {
  direction : inout;
  capacitance : 0.034;
  timing(){
    related_pin : "L0";
    timing_type : three_state_enable;
    intrinsic_rise : 0.080;     // Factor C in equation 1
    rise résistance : 0.70;    // Factor A in equation 1
    slope_rise : 0.11;          // Factor B in equation 1
     }
   }
  ..
 } // End Cell
..
} // End Library
```

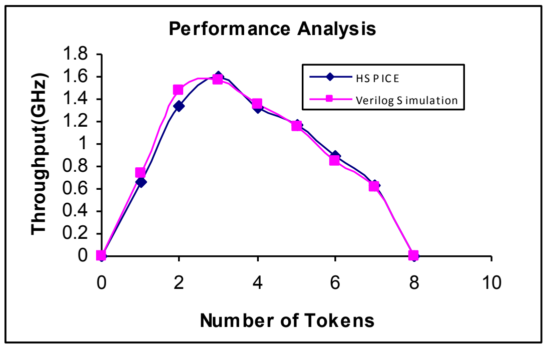
(a) Functional description (b) Liberty model

Figure 49. Functional and Liberty model for STFB buffer

6.5.1 Performance of Pipelines

We analyzed the throughput of the 9-stage pipeline by varying number of tokens in the pipeline as shown in Figure 50(a). Our pipeline structure consists of 6 buffers, a fork, a merge, an exclusive OR and a controlled bit-generator (BG), and the experimental result using HSPICE and Verilog simulation with back-annotation is shown in Figure 50 (b). In addition, the maximum throughput was measured on pipelines of different lengths with the results in Table 1 which shows that the back annotation flow yielded a maximum error of 4.4%.





(a) 9-stage ring (b) Performance analysis

Figure 50. Performance analysis of a 9 stage ring

Design	HSPICE Simulation	Verilog simulation with back annotation	Error (%)
9 stage ring	1.6 GHz	1.58GHz	1.8
15 stage ring	1.7 GHz	1.78 GHz	4.4
30 stage ring	1.6 GHz	1.58 GHz	0.6

Table 3. Throughput comparison of pipelines of different length

6.5.2 Prefix Adder

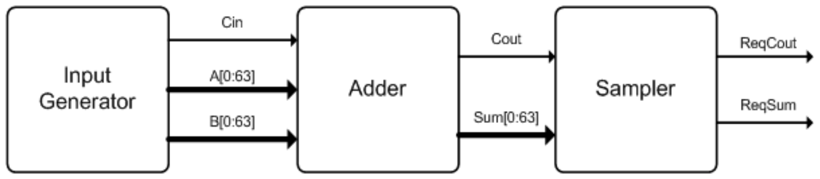


Figure 51. Block diagram of Prefix Adder and its test circuitry

Figure 51 shows the test circuitry used to demonstrate the ASIC standard design flow on a 64 bit asynchronous prefix adder which uses cells from STFB standard cell library. The input bit generator is made up of input rings which generate input fast enough to continuously feed the adder block so that the adder can work at its peak throughput. The sampler block has the responsibility of sampling the addition results by a variable sampling ratio and will set the ReqCout and ReqSum high. In our simulations we have set the sampling ratio to be 1:1000 that means out of every 1000 additions the sampler block will set the ReqSum and the ReqCout variable high for just one addition. In this case the throughput will be the throughput of ReqCout field multiplied by the sampling ratio, i.e. 1000. The adder design has around 5000 STFB standard cells and the layout of this design occupies an area of 4.1 mm^2.

The throughput of the adder after simulation using transistor level simulation with Nanosim© is 1.01 GHz[1]. After performing the same experiment using Verilog Simulation with SDF Back annotation we get the throughput to be 1.06 GHz. The error that we get is 4% which is within acceptable limits. Simulation speed increases the speed of simulation over two orders of magnitude.

6.5.3 Simulation Based Verification of Timing Constraints.

Using Verilog simulations with SDF back annotation we can verify the timing assumptions. STFB cells are based on the timing assumption that one stage will tristate the channels associated with it before the next stage can drive the wire. Let us consider two cases for a test circuit shown in Figure 52.

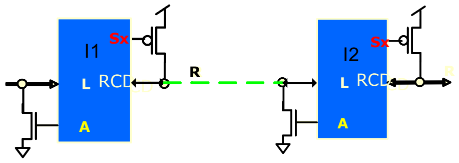
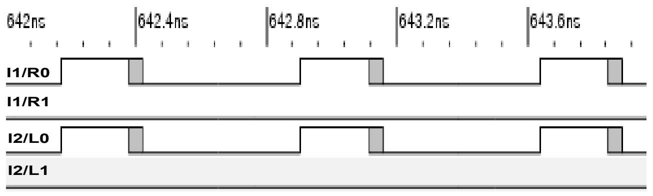


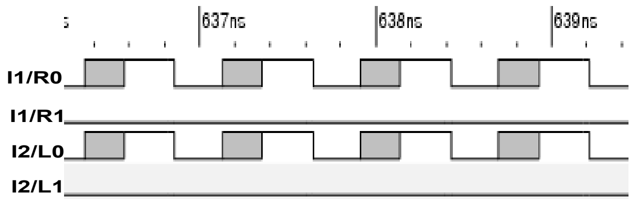
Figure 52. Test circuit to validate the timing constraints using SDF Back annotation

Case 1) I_1 drives R high After R goes high, Sx of the first buffer (I1) should go high, driving the wire to high impedance, before A of the second buffer (I2) goes high which would drive the wire low. If this constraint is not satisfied it will cause a fight between pulling the wire to a logic state '1' and logic state '0' leading to short-circuit current. This fight is detectable during Verilog simulations and is shown by shaded or colored regions in commercially standard waveform viewers, as shown in Figure 53 (a).

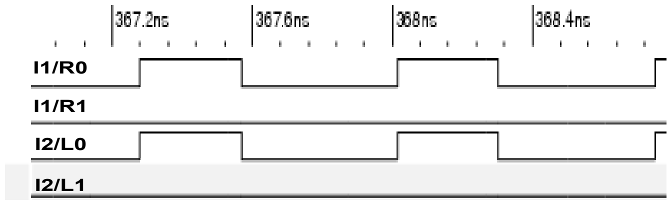


(a)

[1] The difference in the throughput of the design reported here and that reported in [3] is because we used more conservative spice model card to simulate the design.



(b)



(c)

Figure 53. Timing Assumption violated for (a) Case 1 (b) Case 2 (c) No timing violation

Case 2) I2 drives L low: After L goes low, A of the second buffer (I2) should go low, driving the wire to high impedance, before the Sx of the first buffer (I1) goes low which would drive the wire high. If this constraint is not satisfied there will be a fight between logic state'0' and logic state '1', again leading to short circuit current. As in Case 1, this is visible as a shaded region in Verilog waveform viewers, as illustrated in Figure 53 (b). Figure 53 (c) shows that all the timing assumptions are satisfied.

6.6 Conclusion

This paper presents a STFB cell characterization and back-end timing flow using SDF back-annotation. The experimental result indicates that the flow yields over two orders of magnitude increase in simulation speed with less than 5% error. Our future work involves testing and improving this flow on smaller nanometer processes in which the introduced error may be larger for a variety of reasons, including increased crosstalk noise. In addition to back-annotation of timing data for analysis, SDF supports the forward annotation of timing constraints to design synthesis tools. Thus, we also hope to expand the use of this SDF flow to logic synthesis, floor planning, and timing driven placement and routing. In particular, trial or custom placements can be verified against maximum legal interconnect delays specified in a pre-layout SDF file.

In an ASIC environment, it is also very desirable to perform timing verification with some type of static analysis and in particular without relying on simulation, whether it be analog or logic level. Thus, a key next hurdle that we are attacking is to determine how commercial static timing analysis tools can be used to verify timing constraints in the presence of the numerous timing loops in the proposed model.

7 Asynchronous Turbo decoder

7.1 Introduction

Turbo decoding is becoming a very popular solution for error correction especially in wireless applications. Turbo coding started with the introduction of Parallel Concatenated Convolution Codes (PCCC) that were proven to achieve performance that is very close to the theoretical coding bound defined by Shannon's Capacity [19]. Since then several variations have been introduced, such as Serially Concatenated Convolutional Codes (SCCC) and Low Density Parity Check Codes (LDPCC). All these codes achieve Turbo-Like performance, but vary slightly in terms of performance and computational complexity and the selection for a particular design is made based on the operational conditions of the final system.

This chapter presents the design of a turbo decoder using a homogeneous 6-transition high-performance static single-track standard cell library [38]. In particular, we implement a SCCC which is known for its very low error floor capabilities, but the design proposed could be easily modified to decode any of the other types of codes listed above. For comparison purposes, we also designed a synchronous turbo decoder using ARTISAN standard cell libraries. The comparisons show that our single-track asynchronous design has significantly higher throughput per area over its synchronous counterpart.

This chapter is organized as follows; Section 7.2 gives a background on turbo decoding process and also presents the high speed implementation challenges. Section 7.3 presents the design of synchronous turbo decoder. Section 7.4 presents the design of asynchronous turbo decoder. Section 7.5 presents the verification strategy for our design and also presents the comparison results of the two designs. Finally we summarize and conclude in Section 7.6.

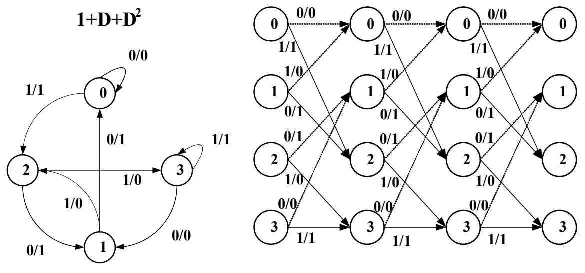


Figure 54. an example of a 4-state FSM encoder and the corresponding trellis used for decoding

7.2 Background

The basic Turbo-Like encoder structure involves an interleaver and a set of simple error correction/detection codes (most commonly convolutional codes). The structure on the decoder side looks very similar to that of the encoder, with two key differences and is illustrated in Figure 55 . First every code in the encoder is replaced by a Soft-In-Soft-Out (SISO) module. Second the data flow on the decoder is bi-directional and iterative, and an interleaver/de-interleaver module in the decoder replaces the interleaver on the encoder side. In our implementation one SISO is used that can perform both operations to reduce design complexity.

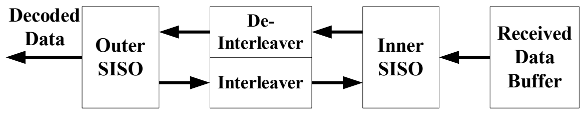


Figure 55. The decoder structure where each SISO is used to decode one CC.

During each SISO operation the data block is processed along a trellis that represents the state of the encoder during the transmission process, as illustrated in Figure 54. The state number is indicated inside each state, while each branch is characterized by the values of the inputs and outputs of the state machine (shown as x/y on each branch in Figure 54). The length of the trellis matches the number of bits in the data block. Each branch in the trellis has a branch metric associated with it (not shown in Figure 56), updated each iteration, which corresponds to a notion of the relative probability assuming that branch took place in the encoder. The SISO module is responsible for updating the probability that at time k the value b was encoded by finding the shortest path through the entire trellis that has value b at time k.

To do this, for each trellis transition the decoder computes the forward state metrics which represent the shortest path from the beginning of the trellis up to that point and the backward state metrics which represent the shortest path information from the end of the trellis up to that transition. This is shown in Figure 56, where the shortest path is shown by the bold lines in the trellis and can be derived by the values of the state metrics. The decoder then can find the shortest path where b_k=1 and the shortest path on which bk=0. Using this information it can produce the new probability for this bit being a 1 relative to being a 0 based on the information available across all the branches of the trellis.

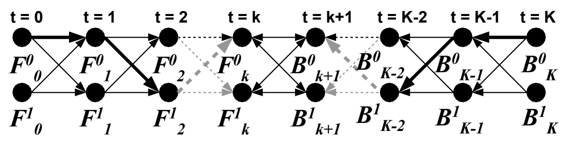


Figure 56. An example of a 2-state trellis and the associated metrics during the decoding process.

In particular, we have chosen to use the Min-Sum algorithm for the SISO operation. The Soft Input to the SISO is defined for each bit at time k as the negative log-likelihood ratio of the probability of a 1 being transmitted over that of a 0:

$$SI_{b_k} = \left(-\log\left(P\{b_k = 1\}\right)\right) - \left(-\log\left(P\{b_k = 0\}\right)\right). \quad \textbf{(1)}$$

The branch metrics between state i and state j (if such exists) is the joint probability for the particular trellis branch given all SI, or:

$$BM_k^{i,j} = \sum_{b_k=1} SI_{b_k} \quad . \quad \textbf{(2)}$$

The forward and backward state metrics for time instance k are then defined recursively as follows:

$$F_k^j = \min_{i,j}\left(F_{k-1}^i + BM_{k-1}^{i,j}\right) \quad \textbf{(3)}$$

$$B_k^i = \min_{i,j}\left(B_{k+1}^j + BM_k^{i,j}\right) \quad \textbf{(4)}$$

where, F_k^i is the value of the forward state metric for state i at time k. The index j only takes the values for which the transition from state j to state i is valid. The state metric calculations are also referred to as the Add-Compare-Select operation or ACS and constitute the majority of the processing taking place in the SISO. An example 4-bit ACS is shown in Figure 57.

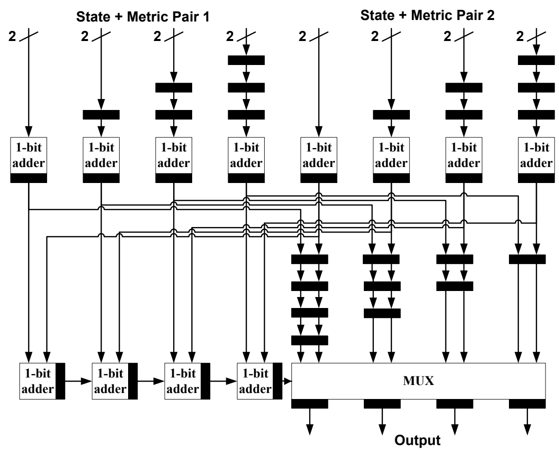


Figure 57. A bit-pipelined 4-bit ACS operator. The black rectangles indicate pipeline boundaries

The SISO outputs are called the Soft Outputs (SO) for all input and output bits of the encoder FSM. To prevent the values that are sent between SISO modules in the decoder from growing indefinitely, instead of the actual value the SISO outputs the differential (called the extrinsic SO) between the actual value calculated (also called intrinsic SO) and the original SI inputted. The final quantities are also saturated to a fixed bitwidth, to reduce the complexity of the SISO modules. So the Soft Output for bit b at time k is defined as:

$$SO_{\mathrm{int}r\,b_k} = \min_{\substack{i,j\\ b_k=1}}\left(F_k^i + BM_k^{i,j} + B_{k+1}^j\right) - \min_{\substack{i,j\\ b_k=0}}\left(F_k^i + BM_k^{i,j} + B_{k+1}^j\right) \qquad (5)$$

$$SO_{extr\,b_k} = SO_{\mathrm{int}r\,b_k} - SI_{b_k}. \qquad (6)$$

The decoding process from a top-level standpoint starts by the received signal being translated into metrics that represent the probabilities for each of the received bits. Then a SISO module uses the received sequence as inputs and produces soft outputs. The soft outputs are then interleaved (or de-interleaved depending on the code) and passed onto the SISO modeling the next convolutional encoder in the transmit sequence as Soft Inputs. During the decoding process the SISOs that correspond to all the codes exchange Soft Input data both in the encoder sequence and in the reverse direction. Each SISO is fired several times and the process iterates until the metrics stop improving, or the maximum number of iterations is reached. For our comparisons we use 6 iterations which achieve most of the coding gain without being too computationally intensive.

High-speed implementation challenges

The immediate effect of this iterative approach is that in order to achieve a certain decoded data rate the SISO has to run several times faster internally in order to keep up with the data. For example a system using a code with two convolutional codes and decoding using 5 iterations would have to run roughly 10 times faster internally than the target throughput. The calculation is iterative and cannot be speed up easily. Several tiling approaches have been developed that break the block into sub-blocks that can be processed in parallel, but normally the logic itself cannot be sped up further than the state metric calculation process (F_k^i and B_k^i) due to the data dependency in that calculation. The adopted Tree-SISO architecture addresses this problem and will be described in detail in a later section. Even if the state update loop is broken (as in Massera's and Tree-SISO architectures [15][5][75]) there are other practical problems that hinder the throughput. A popular approach to increase throughput is to use many units in parallel. Although this generally works, it has implementation problems that make the design of very high-speed Turbo decoders extremely challenging.

The first problem is related to memory access. As mentioned above the data after being processed has to be interleaved between SISO modules. In order to get good coding performance, the interleaver must use a permutation that is ideally random. Therefore, with a high degree of parallelism many bits per cycle have to first be stored into a RAM structure and then retrieved in random order. The usual approach is to have multiple banks of RAM that each receives data corresponding to one processed bit. As the data comes out of those banks in random order, it then has to be multiplexed and distributed back to the SISOs. Constraints are placed on the interleaver to ensure it is clash-free. That not only adds significant complexity to the decoder, due to the crossbar switch that has to be built into the interleaver, but also places constraints on the interleaver design that could yield very sub-optimal interleaver performance, due to lack of randomness. From a hardware standpoint it also requires the instantiation of many more RAM cores that are extremely small and shallow, that consequently require a lot more area and power than fewer larger and narrower RAM instances.

The second problem is also a side effect of the interleaver presence. The entire block of data has to be written into the interleaver before the data can read to start the next SISO process. This is due to the interleaver's random permutation, which implies that the first bits of data that have to be fetched are likely to be among the last bits of data previously stored. Consequently, as the degree of parallelism is increased the processing time can be linearly reduced, but the pipeline latency remains constant yielding diminishing benefits.

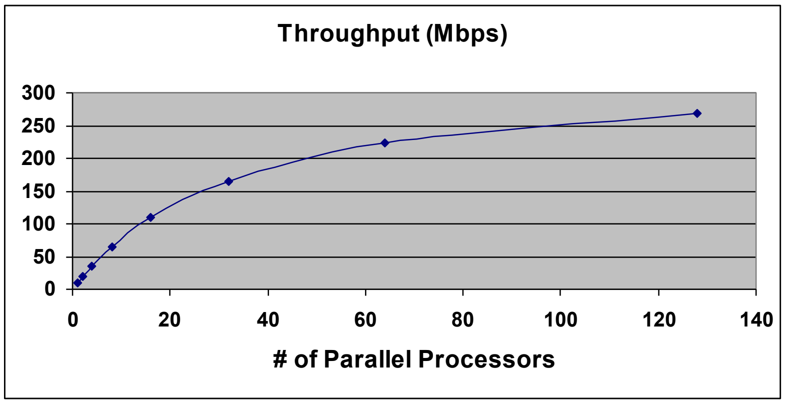


Figure 58. Throughput vs. # of processors

Performance degradation is more pronounced in cases with small data block sizes where the pipeline latency is comparable or in extreme situations larger than the actual processing time. This does not only occur once, but occurs every SISO operation. Figure 58 illustrates this point by showing the throughput that a decoder can achieve as a function of processors to perform one SISO operation. The graph assumes that each processor runs at 100 MHz for 5 iterations for a code that has two convolutional codes and a data block size of 2 Kbits.

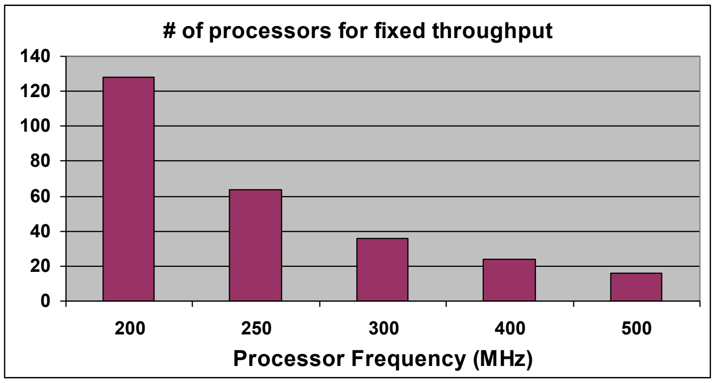


Figure 59. # of processors vs. processor frequency

Figure 59 illustrates this point from a different perspective by showing the number of processors that can be used to achieve a throughput of 540 Mbps with varying processor frequency (assuming 5 iterations and 2 Kbit data block size). It is easy to see that increasing the processor frequency can achieve much larger reduction in the number of processors than the expected linear function.

7.3 Synchronous high speed Turbo baseline

In order to evaluate the performance of our design and demonstrate the capabilities that our asynchronous technology has to offer, we designed a synchronous core as well to be able to compare area, performance and power between the two designs. We chose to design the same unit using both technologies, using our SSTFB library for the one and the Artisan library for the other.

7.3.1 Tree SISO

Designing a very fast Turbo decoder structure has many challenges as mentioned above. Our goal based on our analysis was to design the fastest SISO unit possible so that we can keep the degree of parallelism required to a minimum. For this reason we chose to use a Tree SISO structure [5][75] which removes the recursive nature in the data path and thus enables fine grain pipelining.

In order to illustrate this implementation, we must define one additional operation that merges adjacent transitions of the trellis into larger trellis sections that correspond to more than one time index. In this manner, several state metrics can be computed simultaneously when the appropriate state metric becomes available. The new operation, called the fusion operation, is defined as:

$$BM_{k,l}^{i,j} = \min_{p}(BM_{k,m}^{i,p} + BM_{m,l}^{p,j}), \forall m \in (k,l) \quad (7)$$

The min operation is defined over all valid combinations of branch metric pairs of starting and ending states. The new branch metrics correspond to the shortest possible path derived from the merged trellis sections between any pair of starting and ending states. The structure used to implement the SISO is borrowed from prefix adder structures, but with the addition of a suffix path that is used to perform the operation backwards for the Backward State Metric calculation.

7.3.2 The code

We chose a typical SCCC turbo code structure with two 2-state convolutional codes to reduce the size of the decoder circuit. The data is first encoded using a rate ½ non-recursive convolutional code with polynomials [1+D, 1+D] and the results are interleaved. Next a rate 1 recursive code with a polynomial [1/(1+D)] is used to re-encode the interleaved data before transmission. Overall this yields a rate ½ code. Puncturing could be used to achieve higher code rates and increase flexibility, with minor modifications to the design, but this was not done at this stage for simplicity.

For a block size of K bits the first code has a trellis length of K and the second one of 2K. Therefore the throughput equation is as follows:

$$T = \frac{f * K}{I\left(2p + \frac{3K}{8M}\right)} \quad \textbf{(8)}$$

where K is the block size in bits, f is the clock frequency in Hz (or in the case of the asynchronous design the equivalent throughput), I is the number of iterations and 8M is the number of bits that can be processed in parallel. Finally p is the pipeline latency in terms of cycles. Each SISO operation has to finish and store data back into memory, therefore the pipeline overhead is present for every half-iteration. The execution schedule for every iteration is shown in Figure 60. It should also be noted that only the last iteration produces decoded data, which is why the throughput is inversely proportional to the number of iterations.

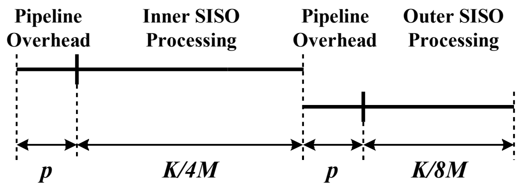


Figure 60. Execution schedule of every decoder iteration

In the case of our asynchronous design M is 1 since we are going to use a single 8-bit wide SISO processor to achieve the desired throughput. In the case of the synchronous design, M will have to be higher since the synchronous design is much slower than our asynchronous one and multiple processors of size 8 would be required to achieve the same throughput. We chose 8 since it is the minimum size Tree-

SISO for a 2-state code, and it should be noted that due to the structure of the Tree-SISO a 16-bit wide processor is more complex than two 8-bit wide ones.

7.3.3 P&R results

After the schematic design was finished it was exported to a Verilog netlist and imported into SOC Encounter for P&R. The libraries for the IBM 0.18 μm technology were used for characterization, and timing constraints were written to define the target frequency. The core was placed as a standalone module without IO pads. The design was placed using timing-driven placement and was then routed using timing-driven routing. After routing was done the clock tree was synthesized and the design was taken through further processing to fix hold time violations and then the final timing analysis was performed. The clock frequency that was achieved was 475 MHz for the entire 8-bit wide core. The area of the core was 2.46 mm^2. We assume that for higher degrees of parallelism multiple copies of this core could be routed separately and that no performance degradation would be induced due to the added circuitry. We also assumed that the clock circuit would be mostly unaffected and that it would just be multiplied in size just like the rest of the circuitry.

As a point of comparison, the previously published fastest turbo design achieves approximately 1 Gbps at 6 iterations in a 0.18 μm process, using 32 mm^2 area and a single-buffered input memory [15]. The code in [15] is a PCCC which requires the processing of 2K trellis steps/iteration, so that structure would decode approximately 667 Mbps for an SCCC code like the one we chose, which requires processing of 3K trellis steps/iteration. Our synchronous design has similar throughput (653 Mbps) for M=6 and using 14.76mm^2 of core area and 2.17mm^2 of memory area, which indicates that our synchronous design is comparable with state-of-the-art decoders found in the literature. Since our synchronous and asynchronous cores would use the same memory area, our comparison only considers the core area.

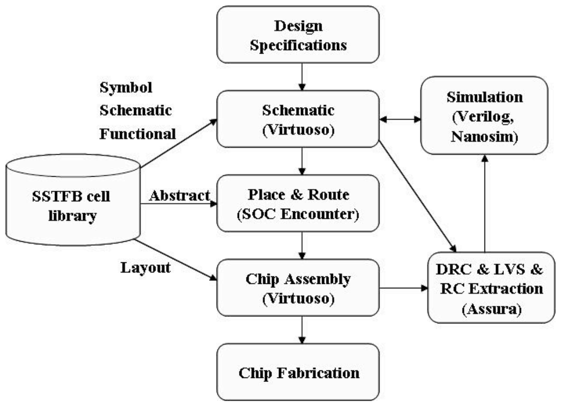


Figure 61. Asynchronous ASIC design flow

7.4 Asynchronous Turbo

This section covers the asynchronous Turbo design, including a brief description of the overall design flow and the P&R results.

7.4.1 Asynchronous ASIC Design Flow

For each SSTFB cell needed we created four library views: functional views contains the behavioral description of the cell in Verilog HDL, schematic views contains the transistor level implementation of the cell, layout view containing detailed GDSII data, an abstract view to support placement and routing in LEF format, and finally its symbol. Using this library, a largely conventional standard-cell ASIC back-end design flow using conventional place and route tools can be used to create the layout, as illustrated in Figure 61.

7.4.2 P&R and design results

The design was place and routed using Cadence SOC Encounter in a similar fashion as the synchronous counterpart. Congestion based placement was performed and the routing was performed on the design using Nanoroute. The final core has 70% utilization and the final area consumed by the logic is 6.92mm2. The core is fully routed using 6 metal layers showing that the design is routable.

7.5 Verification and Comparisons

This section covers the design verification of the asynchronous turbo and its simulation results.

7.5.1 Design Verification

The schematic of the design was converted into a Verilog netlist and the simulation was performed using our Verilog models for the SSTFB cells in NC-Sim. Due to the size of the design and the instantiation of multiple identical components, the verification was performed in a bottom-up fashion. We started with simple cells such as adders and moved up the hierarchy to ACS units, state update nodes, branch metric calculation units, the completion logic, and finally the top-level module. For each module a set of vectors were generated that would test all corner cases of its behavior and the results were verified and cross referenced to the synchronous counterpart.

7.5.2 Post-Layout ECO and simulation results

To estimate the performance of the chip we simulated the 55K-transistor module that implements Equations 5 and 6. This module contains an 8-bit ripple carry chain of full adders which includes the critical cycle of the design. This module represents around 1/30th of the complete design but is the most computationally intensive module in the SISO.

To improve the performance of the design we added SSTFB Buffers on long wires using the ECO flow in SOC Encounter. The addition of the buffers increased the modules throughput by 26% and increased the utilization factor from 70% to 76% but otherwise did not impact area. The final layout was extracted using Assura RC in coupled mode and the circuit was simulated using Nanosim, yielding a throughput of 1.15 GHz.

7.5.3 Comparisons

The frequency of the post P&R synchronous core is 475 MHz. From the post layout simulation explained in Section 7.5.2 we expect the asynchronous core frequency to be approximately 1.15GHz. Thus, we expect the asynchronous core to run 2.4 times faster then its synchronous counterpart.

7.5.3.1 Area comparison

The logic areas of the synchronous and asynchronous cores are 2.46 mm^2 and 6.92 mm^2, respectively. Both asynchronous and synchronous cores implement the exact same function with the same degree of parallelism. However since the asynchronous core is 2.4 times faster and has smaller pipeline latency than the synchronous core, we must instantiate the synchronous core many times in order to match the throughput, as described in Section 7.3.3. Substituting the numbers in Equation (8), we compute that for equivalent throughput with 6 iterations and pipeline latencies of 60 cycles for the synchronous design and 32 equivalent cycles for the asynchronous one, the number of required synchronous cores varies from 11 for a throughput of 418 Mbps for block size of 768 bits to 3 for a throughput of 490 Mbps for block size of 4 Kbits.

7.5.3.2 Throughput/area comparison

Throughput/area is another important metric for the comparison, since it indicates the performance in relation to the area used to achieve it. We chose to use throughput/area instead of just area for comparable throughputs as a metric for our comparisons. This is because the throughput that is achievable by each design is not exactly equal, so the ratio comparison provides a normalized metric that is fairer. From Table 4 we can see for example that for a block size of 1 Kbits which is a very common block size used in wireless applications we obtain a throughput per area advantage of 2.13. The advantages are even bigger for smaller block sizes, and for block sizes of 512 or smaller, the synchronous design cannot match the throughput of the asynchronous counterpart, regardless of the degree of parallelism M. As the block size increases, latency becomes less of a critical factor and the two designs become more comparable.

Block Size (bits)	Async T (Mbps)	Sync T (Mbps)	M	Sync area (mm2)	T/area ratio
512	383	-	-	-	-
768	418	415	11	27.06	3.91
1024	438	440	6	14.76	2.13
2048	471	519	4	9.84	1.28
4096	490	513	3	7.38	1.03

Table 4. Throughput per area comparison

7.5.3.3 Energy Comparisons

From the post-layout spice simulation the power consumed of the selected module is 0.53W. If we extrapolate the number we expect the power of the complete Tree SISO to be approximately 15.5W. The power for a single synchronous core (M=1) is 1.72W. From Table 5 we can see that for smaller block sizes we are more energy efficient than the synchronous design, but for larger block sizes the synchronous design more efficient. It should be stated that the power calculation for both designs was performed for the worst case scenario, namely with the units processing data at the maximum rate. Even though the calculation is based on peak power, we believe that the numbers might be conservative, but the ratio should be representative of the relative power consumption of the two designs. We have also not included leakage power comparisons in the calculations, but given that the asynchronous design requires less area for the same throughput and leakage power is proportional to the total area, we expect to have an advantage in respect to that aspect as well.

Block Size (bits)	Energy per block (sync)	Energy per block (async)	Ratio
768	3.5E-05	2.84E-05	0.81
1024	2.40E-05	3.63E-05	1.5
2048	2.714E-05	6.73E-05	2.5
4096	4.11E-05	12.9E-05	3.1

Table 5. Energy per block comparison

7.6 Summary and conclusions

Our results demonstrate that SSTFB asynchronous turbo decoder is beneficial for small to medium block sizes. Preliminary comparisons show that the asynchronous turbo decoder can offer more than 2X advantage in throughput per area for block sizes of 1K bits or less and smaller energy per block for block sizes of 768 bits or less. Thus the asynchronous design is particularly useful in low latency wireless applications in which block size must be small. More generally, this design experiment demonstrates the potential benefits of high-performance low-latency asynchronous libraries and standard-cell design flows for processing intensive applications.

The current SSTFB library has only one size per cell. While this is sufficient to achieve high performance, multiple sizes for each cell can significantly reduce the overall capacitance and power consumption. In addition, 64% of the cell instances in the Tree SISO design are dual-rail buffers for slack matching. If these are replaced by 1-of-4 or 1-of-8 buffers (that have less switching activity per bit), significant reductions in power consumption is likely.

8 Asynchronous vs Synchronous Communication

8.1 Introduction

A number of effects are impinging on the way we do design. A fundamental change is in the requirement of adding pipeline stages in the interconnect. This increased complexity is due to a number of factors including physical scaling as well as the desire to increase performance. This communication pipelining results in somecircuit and architecture advantages and overheads. Various communication methodologies have differing local and architectural performance implications and CAD requirements. This work models pipelines using various protocols, including clocked and asynchronous, and makes a first-order comparison of energy and performance of these models. These models show that some asynchronous communication methods can be comparable or superior to clocked communication, even under worst case conditions.

Global synchronous design requires the expenditure of a large design effort to create a low skew clock. While this can result in many efficiencies, including no synchronization overhead and uniform performance targets, there are also significant drawbacks. A single global frequency may not be optimal for CPU architectures because each module has different optimal power/performance points. Further, a global frequency requires re-pipelining every module as frequencies change. Furthermore the global clock distribution network also leads to more overhead to account for clock skew and higher energy consumption. If efficient asynchronous communication can be designed, three significant benefits arise. First, each module can be designed for its best frequency and power. Second, the ability to interconnect components with different frequencies will be vastly enhanced resulting in higher design reuse, faster time to market, and easier ability to customize designs. Thirdly the ability of the asynchronous designs to adapt themselves to physical properties leads to more robust design.

The goal of this work is to study communication methodologies to determine if and when asynchronous communication can be competitive at the physical level with synchronous styles. A similar study is done in [26] where bit-rate, power, are and latency are compared between register links, wave pipelined parallel links, and asynchronous serial links [27]. This work, instead, focuses on asynchronous parallel links, comparing them with synchronous methodologies.

8.2 Overview

Eight representative protocols are modeled in this paper. This includes two synchronous protocols, five asynchronous protocols and one source synchronous protocol. The synchronous protocols include the flopped design (clk_flop) and latched design with time borrowing (clk_latch). The asynchronous protocols include dual-rail (2-rail) and one-of-four (1-of-4) DI protocol, NRZ two-phase and RZ four-phase bundled data protocols, and two phase single-track protocol. Finally source synchronous protocol (src_sync) is modeled, which sends the request as a pulse along with the bundled data. The source synchronous protocol has no acknowledgment.

Other protocols such as the PC2/2 protocol [62], the Mousetrap protocol [63] and those using n-of-m codes [3][3]can be similarly modeled but are not included in the paper for clarity. Most of these protocols will be similar to, or bounded by, the models presented in this paper in terms of throughput, energy, and bandwidth.

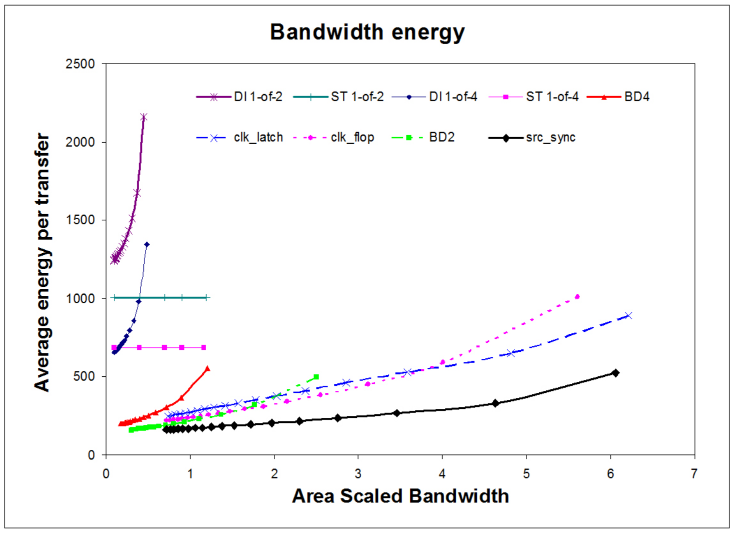


Figure 62. Complete energy / bandwidth graph

Figure 62 and Figure 63 show the final results of the models applied to a long 10,000 μm bus with a critical repeater distance of 555 μm. The distances and parameters are typical of what might be found on a microprocessor fabricated in a 65 nm process. A very long bus was chosen to allow a wide range of pipelining and frequencies to be graphed. The y-axis shows the average energy per transaction, measured in relation to the energy of driving a minimum sized inverter. The x-axis shows the effective bandwidth in terms of the number of concurrent transactions that can be sent down this path. The bandwidth is scaled for area by dividing the overall throughput by the number of wires in the control and data paths.

Each tick mark on every protocol line in the graphs indicates a specific pipeline granularity and thus a particular frequency. The parameter values in this paper use an optimized 10,000 µm wire with 18 repeaters. The far left point of each protocol is the condition where all of the repeaters are inverters. Pipelining is increased moving right along each protocol by replacing one inverter with flops or latches. As more of the inverters of the communication path are replaced with pipeline latches, the frequency and bandwidth increases across the x-axis. The rightmost tick is where all of the inverters have been replaced with flopped repeaters or pipeline control, achieving the maximum bandwidth for the protocol.

Achieving the target bandwidth can come at different frequencies (or pipeline granularity) based on the protocol being used. In general, these results show that the asynchronous protocols require higher pipelining to achieve the same bandwidth as the synchronous protocols. For example, to achieve the area scaled bandwidth of 1, asynchronous 4-phase bundled data communication requires more pipelining than synchronous flop based communication, hence consuming nearly 2 times more energy. This can be seen by determining the pipelining for every protocol that achieves a particular bandwidth target in the graphs.

Increased bandwidth comes at the cost of increased energy per transaction for a fixed bus width. Fine-grain pipelining, or the right-most point on each protocol, is the most energy-hungry operational mode for any protocol. Furthermore for synchronous and source synchronous designs there exists a limit on the clock frequency equal to the minimum pulse width that can be successfully propagated along a critical distance length of a wire. For optimal power and performance, the bandwidth should be matched by appropriate pipeline granularity or frequency.

The asynchronous protocols show a significantly reduced bandwidth range when compared with the synchronous and source synchronous protocols. This is due to the delay overheads in propagating request and acknowledge signals across long wires.

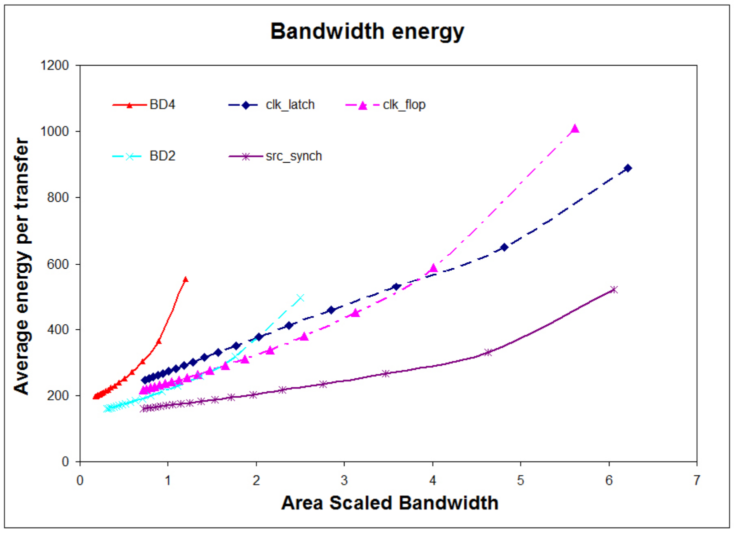


Figure 63. Efficient protocols

Figure 63 zooms in on the energy efficient protocols. Synchronous protocols exhibit the best energy and bandwidth values for coarse grained pipelining having a low clock frequency. The source-synchronous protocol is the best for highly pipelined designs. The bulk of the energy for all protocols is dominated by the wires. However, as the data is increasingly pipelined, the average energy per transaction increases. The slopes of the four-phase asynchronous protocols are steeper, indicating a larger energy penalty for increased bandwidth.

The graphs show the worst case conditions. A vast majority of the asynchronous protocols will operate at a significantly improved frequency on a chip because they adapt to the current fabrication and environment conditions. Nominal values would reduce the slopes on the asynchronous protocols which makes them more competitive to synchronous design.

The DI and single-track protocols exhibit significantly higher average energy per transaction and the lowest bandwidth ranges. The bandwidth limitations are largely due to the inefficient use of wires. Each bit requires two wires in these protocols, effectively halving the area scaled bandwidth reported here. The high energy consumption of these protocols can mostly be attributed to the high activity factors that result from the data encodings. However, because these protocols are delay insensitive the CAD requirements are greatly reduced, allowing quicker time to market and higher robustness to operational parameters. Hence they may still be good choices depending on the overall design requirements.

8.3 Comparison Methodology

The following methodology was employed to compare various modes of communication.

1. Create a set of parameterized first order equations modeling the following for each protocol:
    - (a) delay and delay variations
    - (b) energy and energy variations
    - (c) cycle time and latch and flop overheads
    - (d) bandwidth and wire area
2. Design 65nm circuits for all protocols and simulate them to:
    - (a) provide accurate delay, energy, and area values for the models
    - (b) compare the first order model results against SPICE simulations of the complete design

Name	Equation	FO4 Value
V_c	Cross coupling variation of wire	0.4
V_p	Process variation	0.16
V_{pw}	Pulse width variation	0.2
V_{vt}	Voltage & temperature variation	0.12
V_{cad}	Margin for CAD (PV Accuracy)	0.05
D_i	Ideal stage delay (as FO4)	27.8 – 1
L_w	Latency of a 10,000μ interconnect with repeaters	27.8
T_{cq+}	Delay from clock to output	1.8
T_{cq-}	$T_{cq+}(1 - V_p)$	1.65
T_{sul}	Latch setup time	0.48
T_{suf}	Flop setup time	0.96
T_{skj}	Clock skew and jitter	1
T_{sk}	Clock skew	1
T_{skp}	Time borrow phase skew	0.5
T_h	Hold time	0.25
T_{dqf}	Flop data-to-output delay	2.8
T_{dql}	Latch data-to-output delay	1.4
T_{pw}	Minimum pulse width that can be propagated through a critical distance	2.2
T_{pw+}	$T_{pw}(1 + \frac{V_{pw}}{2})$	2.42
T_{dr}	Domino reset & propagation	0
T_{cad}	$D_i V_{cad}$	1.5–0.05
T_l	Part of D_i attributed to logic	0
T_{lc+}	$T_l(1 + \frac{V_p}{2} + V_{vt})$	0
T_{l+}	$T_l(1 + \frac{V_p}{2})$	0
T_c	$D_i - T_l$	27.8 – 1
T_{cc+}	$T_c(1 + \frac{V_c}{2} + \frac{2V_{vt}}{1+D_c})$	35.9 – 1.29
T_{c+}	$T_c(1 + \frac{V_c}{2})$	33.25 – 1.2
T_{c-}	$T_c(1 - \frac{V_c}{2})$	22.25 – 0.8
N_P	Number of pipeline stages	18 – 1
N_{rep}	Number of repeaters per pipeline stage excluding memory element	18 – 0
T_{csh+}	$T_c(1 + \frac{V_c}{4})$	30.59 – 1.1
T_{fsm+}	Controller delay per phase	2.55 – 6.64
T_{fsm-}	$T_{fsm+}(1 - \frac{V_p}{2})$	2.34 – 6.10
T_{lat+}	Forward latency of async controller	1.18 – 3.95
E_{ctl}	Controller energy per phase per bit	0.072 – 0.90
E_{rep}	Energy of a single repeater	1
E_{datf}	Data energy through flop	2.02
E_{datl}	Data energy through latch	1.78
E_{clkf}	Clock energy for flop	0.22
E_{clkl}	Clock energy for latch	0.2
E_{clkdef}	Clock energy for DETFF	0.42

Table 6. Parameter variables and derivatives

8.3.1 Parameters

The timing of all fabricated circuit elements will be faster or slower than the scalar value of an "ideal" element due to process variations, capacitive coupling, and other effects. Variation from ideal devices and wires are considered an overhead in this work, whether it manifests itself as clock skew or device and wire delay variation. Table 6 and Table 7 show the parameters and their associated values that are used in this paper. All values come from simulation of the designs in 65nm technology using predictive spice models [1]. The parameters in the top section of Table 6 model variations from ideal delays that occur on wires and devices. First-order effects are modeled, including coupling, process variation, voltage and temperature variations (V_c, V_p, and V_{vt} respectively).

Values in the bottom section of Table 6 are scalar delays. These delays are all relative to the fanout of 4 (FO4) of a typical inverter, for example worst case clock skew is assigned a scalar delay of 1 FO4. The ideal stage delay, D_i, is varied from 27.8 FO4 to 1 FO4 to allow evaluating pipelines ranging from coarse to very fine grain. This is the amount of ideal logic delay between latches or flops in a logic based design, and it determines the pipelining or frequency of a communication link. Latency L_w represents the ideal latency to transmit data across an unpipelined 10,000 μm wire.

Name	Equation	Value
A_w	Activity factor of bus wires	0.18
A_b	Activity factor of bus	0.05
O_{wd}	Control wire delay optimization	0.85
O_{wa}	Control wire area optimization	1.85
X_c	Critical wire distance	555μm
W_b	Width of bus	32
D_c	Delay across repeated wire	1.6
$E_{clktree}$	Clock tree energy per transition	16.84
P_{dc}	% of clock tree energy for communication	10%

Table 7. Parameter variables and derivatives

The target delay of any pipeline stage can be distributed as either functional logic delays T_l or communication delays T_c, which includes both repeater and wire delay as shown in Figure 64. For the designs simulated in this paper D_i=T_c. However, for generality our models include function logic delays which significantly change the throughput results for some protocols. Variations are applied differently to logic and communication.

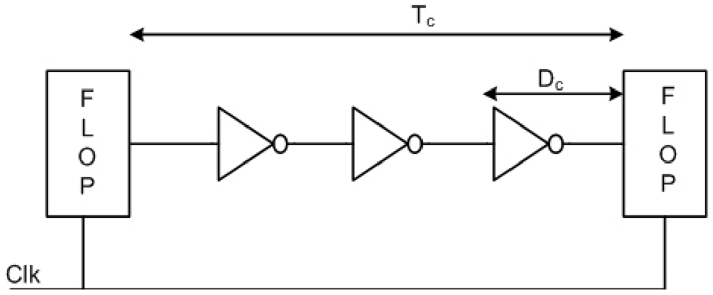


Figure 64. Total communication delay in a pipeline stage delay T_c, versus repeated wire segment delay D_c

More accurate first-order models are achieved by including variation values. A single value V_c is used here to model crosstalk coupling variation. Delays with a maximum variant can append a '+' symbol, and min-delay values append a '-'. For example, T_{c+} is the max communication delay and T_{cq-} is the minimum clock to data output delay of a flop or latch. T_{lc+} and T_{cc+} are the maximum logic delay and communication delays for synchronous systems. Synchronous protocols take first droop effects into account since the fixed frequency could otherwise cause the circuits to fail under the slower operation induced by the voltage droop. The asynchronous protocols will instead adapt itself to the operating conditions. Communication delay for the shielded links T_{csh+} of the DI and single-track protocols is smaller than for the non-shielded bundled data paths.

Variables N_{rep} and N_P in Table 6 represent the number of repeaters per pipeline stage excluding the memory element and the number of pipeline stages respectively. The N_{rep} parameter can be calculated as $\frac{T_c}{D_c}$, while for all modes of communication except synchronous latch based N_P can be calculated as $\frac{\frac{L_w}{D_c}}{N_{rep}+1}$, where $\frac{L_w}{D_c}$ represents the total number of repeaters required for a 10,000mm interconnect. In case of latch based communication there are two latches acting as repeaters per pipeline stage, hence N_P in that case is equal to $\frac{\frac{L_w}{D_c}}{N_{rep}+2}$.

The parameters in Table 7 include the interconnect design information of our process. To identify the minimum delay we performed a 2D sweep on the repeater size and number of repeaters required for a 10,000 μm long interconnect [28]. The optimum design required the number of repeaters to be 18 and the size of the repeater to be 96X a minimum size inverter. The delay across each repeater stage D_c is approximately 1.6 times a nominal FO4 delay with the energy required to drive a critical repeater distance as E_{rep}. Here we assume that repeaters are placed optimally at the distance of critical length. The energy required for data transfer in various protocols will be calculated relative to E_{rep}.

The $E_{clktree}$ parameter in the Table 7 represent the energy required by the clock tree to distribute clock to a 10,000μm interconnect. For sake of simplicity we model the clock distribution network as a single wire running in parallel with data having repeaters at the critical distance as shown Figure 65. Parameter P_{dc} defines the percentage of the energy of the repeated clock wire $E_{clktree}$ that is used for data communication. In particular, P_{dc} models the fact that the clock may be used not only for distributing clock to our data communication setup but also to other logic blocks in the vicinity. With the values set in Table 7, for a 32-bit data path the clock distribution contributes 10% of the energy to the synchronous design in case of extreme pipelining. More complicated models of clock trees are also possible and may be mapped to this simple model.

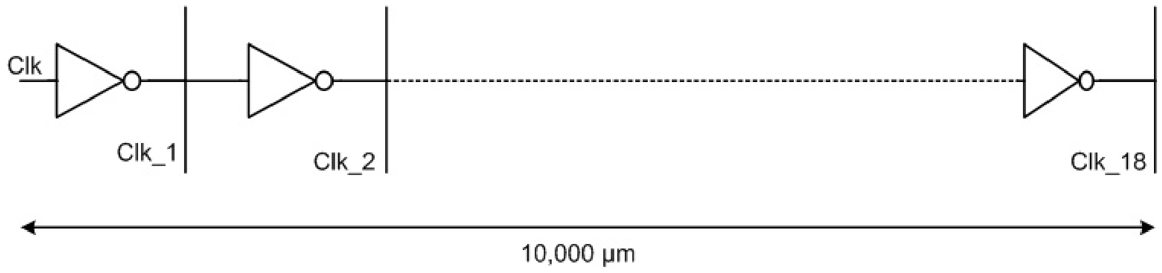


Figure 65. Clock tree to distribute clock over 10,000 μm interconnect

The T_{fsm} parameter in the Table 6 is based on the worst case cycle time of the asynchronous controller. The total cycle time is averaged between the four (two) phases of the four (two) phase asynchronous controller. Different protocols exhibit different cycle time overhead, for example, 2-phase NRZ protocols generally require more complicated control with larger delays compared to 4-phase protocols. The T_{lat} parameter represent the forward propagation latency of the handshake control signals in the asynchronous protocols. Intuitively this represents the delay associated with propagation of the incoming request signal to outgoing request signal. The E_{ctl} parameter represent the average energy per phase per bit for the control logic for the asynchronous protocols. Similar to control delays, different asynchronous protocols exhibit different energy requirements. For a fair comparison T_{fsm}, T_{lat} and E_{ctl} has been characterized through simulation for the different protocols and are listed in Table 8. The sizing and spacing of the control wires in the asynchronous protocols may be optimized differently than the data wires. The parameters O_{wd} allow a reduced control wire delay for a larger area O_{wa}.

The parameter A_b is the activity factor of the the bus, or the percentage of clock cycles during which data is transmitted across the bus. Each flop is considered to be gated during idle cycles in the clocked protocols. A_w is the activity factor of data on the bus, or the probability that any bit will switch values for an average bus transaction.

Name	$\mathbf{T}_{fsm}$	$\mathbf{T}_{lat}$	$\mathbf{E}_{ctl}$
4-phase bundled data	4.28	3.45	0.075
2-phase bundled data	4.77	3.45	0.072
DI 1-of-2	6.64	1.18	0.90
DI 1-of-4	6.04	1.2	0.60
STFB 1-of-2	2.46	1.04	0.87
STFB 1-of-4	2.55	1.16	0.60

Table 8. Asynchronous control parameters from 65nm spice simulations of the designs

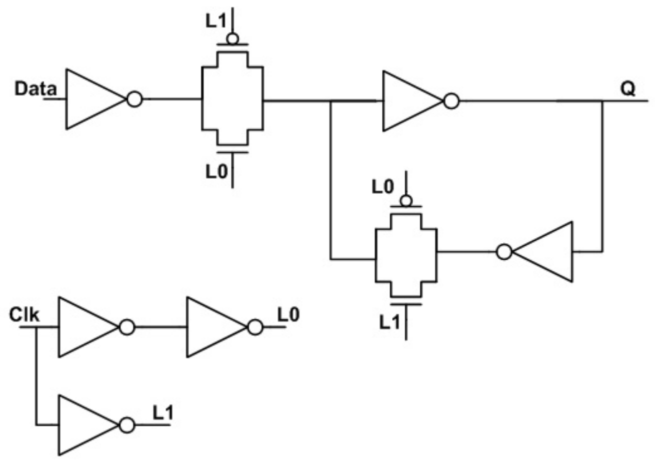
Name		equation	description
T_{ps}	$\leq$	$(T_{fsm+} + T_{cq+} + T_{l+} + T_{cc+} + T_{su}) - (T_{fsm+} + T_{c-} + T_{fsm-})$	Handshake to data margin
T_{cdel}	$\leq$	$(1 + \frac{V_p}{2})\max(0, T_{ps}) - (1 - \frac{V_p}{2})\max(0, T_{ps}) + T_{cad}$	Robust delay element size
T_{pdel}	$\leq$	$(1 + \frac{V_{pw}}{2})\max(0, T_{ps}) - (1 - \frac{V_{pw}}{2})\max(0, T_{ps}) + T_{cad}$	Pulse delay element size
C_{ff+}	$\geq$	$\max(2T_{pw}, (T_{lc+} + T_{cc+} + T_{su} + T_{skj} + T_{cq+} + T_{cad}))$	Clk_flop throughput
C_{l+}	$\geq$	$\max(2T_{pw}, T_{lc+} + T_{cc+} + 2T_{dql})$	Clk_latch throughput
C_{di+}	$\leq$	$4T_{fsm+} + 2T_{csh+} + 2O_{wd}T_{c+} + T_{l+} + T_{dr}$	2-rail and 1-of-4 DI
C_{bd2+}	$\leq$	$2T_{fsm+} + T_{cc+} + T_{cdel} + O_{wd}T_{c+}$	2-phase bundled data cycle time
C_{bd4+}	$\leq$	$4T_{fsm+} + 2T_{cc+} + T_{cdel} + 2O_{wd}T_{c+}$	4-phase bundled data cycle time
C_{ss+}	$\leq$	$\max(2T_{pw}, (T_{cq+} + T_{l+} + T_{cc+} + T_{su}), (T_{c+} + T_{pdel} + T_{fsm+}))$	src-sync cycle time
C_{st+}	$\leq$	$2T_{fsm+} + 2T_{csh+}$	STFB 1-of-2 and 1-of-4

Table 9. Cycle time equations. Cycle time measured in FO4 delays

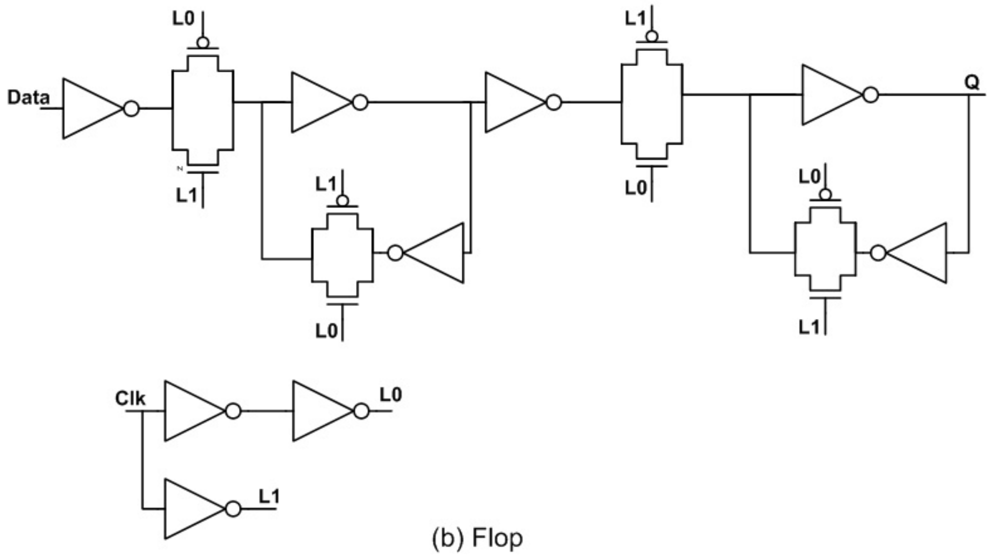
Name		equation	description
L_{ff+}	$\geq$	$\max(2T_{pw}, (T_{lc+} + T_{cc+} + T_{su} + T_{skj} + T_{cq+} + T_{cad}))$	Clk_flop latency
L_{l+}	$\geq$	$\max(2T_{pw}, T_{lc+} + T_{cc+} + 2T_{dql})$	Clk_latch latency
L_{di+}	$\leq$	$T_{lat+} + T_{csh+} + T_{l+}$	2-rail and 1-of-4 DI latency
L_{bd2+}	$\leq$	$T_{lat+} + T_{cc+} + T_{cdel}$	2-phase bundled data latency
L_{bd4+}	$\leq$	$T_{lat+} + T_{cc+} + T_{cdel}$	4-phase bundled data latency
L_{ss+}	$\leq$	$\max(2T_{pw}, (T_{cq+} + T_{l+} + T_{cc+} + T_{su}), T_{c+} + T_{pdel} + T_{fsm+})$	src-sync latency
L_{st+}	$\leq$	$T_{lat+} + T_{csh+}$	STFB 1-of-2 and 1-of-4 latency

Table 10. Latency per stage. delay measured in FO4 delays

8.4 Models

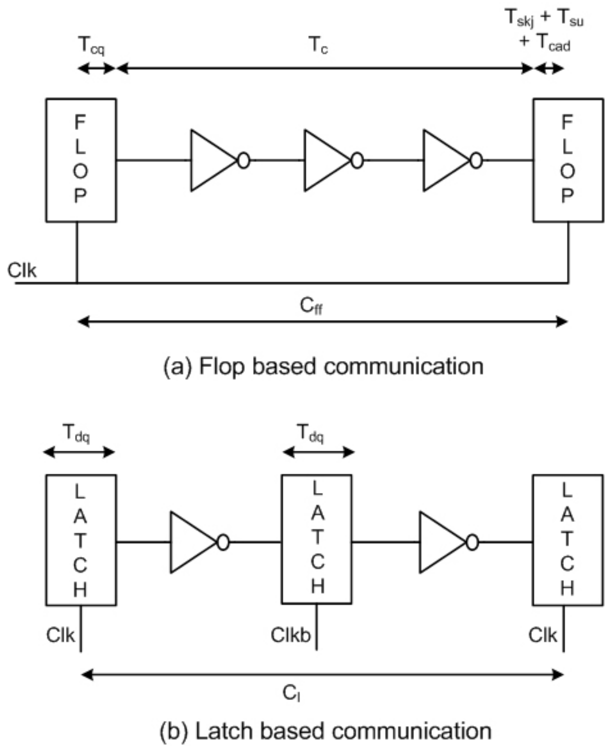


(a) Latch



(b) Flop

Figure 66: Latch and flop transistor level design



$T_{skj} + T_{su} + T_{cad}$
T_{cq}
T_c
FLOP
FLOP
Clk
C_{ff}
(a) Flop based communication
T_{dq}
T_{dq}
LATCH
LATCH
LATCH
Clk
Clkb
Clk
C_l
(b) Latch based communication

Figure 67. Latch and flop pipeline stage configurations for distance of 4 critical repeater distances.

8.4.1 Synchronous communication - Latch and Flop

Figure 66 shows the transistor level diagram of the latch and flop used in this work. The sizes of devices in this paper are calculated using logical effort [72]. In case of a flop based communication the data has to go through one flop per clock cycle while in case of latch based communication data has to go through two latches one being driven by clock and the other one being driven by inverted clock as shown in Figure 67.

The minimum cycle time of a flop represented by equation C_{ff+} presented in Table 9 will be bounded by one of the two conditions. First, the clock cycle cannot be shorter than twice the width of the minimum sized pulse $2T_{pw}$ that can be safely engineered and propagated. Otherwise, the delay through a stage will be the sum of the maximum logic and communication delay from the source flop to the destination flop $T_{lc+}+T_{cc+}$. Synchronous systems have additional overheads of setup time, clock skew and jitter, and other CAD related inaccuracies $T_{su}+T_{skj}+T_{cad}$. Upon arrival of the clock edge, the data must also propagate through the flop T_{cq+}. Setup, skew, and flop delays create the overhead for clocking.

Equation C_{l+} in Table 9 is the minimum cycle time for a synchronous latch protocol. Similar to the flop protocol the cycle time is bounded by one of the following two conditions, twice the width of the minimum sized pulse $2T_{pw}$ and the delay through a stage which is equal to the communication delay from the source latch to the destination latch $T_{lc+}+T_{cc+}+2T_{dql}$. Use of latches instead of flops helps in reducing clock skew and jitter overheads because of the time borrowing allowed between stages. However the data must propagate through two latches $2T_{dql}$ to account for the alternate phase control of the latches.

The equations that calculate the energy consumption for latches E_{le} and flops E_{fe} in Figure 66 are shown in Table 11. The energy consumed by the flop / latch to transfer data the critical distance of a communication link is represented as the parameter E_{dataf} / E_{datal}. The average energy consumed per transition by the flop / latch due to switching of the clock is represented as parameter E_{clkf} / E_{clkl}. The average energy per transaction for clock gating of a flop E_{gf} / latch E_{gl} is calculated as a single transition on the clock driver, which is one-fourth the clock load of the flop plus energy into the gating NAND for the average number of idle bus cycles per transaction. The average energy per transaction consumed by the clock distribution network is given as $\frac{1}{A_b}2P_{dc}E_{clktree}$ where $\frac{1}{A_b}$ accounts for the amount of energy consumed by the clock distribution network when the data bus is idle.

The energy consumed per transaction of a single flop / latch pipeline stage can be attributed to two factors: the energy consumed by the memory element (flop / latch), and energy of the repeaters in that pipeline stage. The energy consumed by a flop is $W_b(2(E_{clkf}+E_{gf})+A_wE_{dataf})$ and the energy consumed by the repeaters $W_bA_wE_{rep}N_{rep}$ to drive the data. In case of latch based communication due to two latches per pipeline stage the energy consumed by the latches is $W_b(4(E_{clkl}+E_{gl})+A_w(2E_{datal}))$.

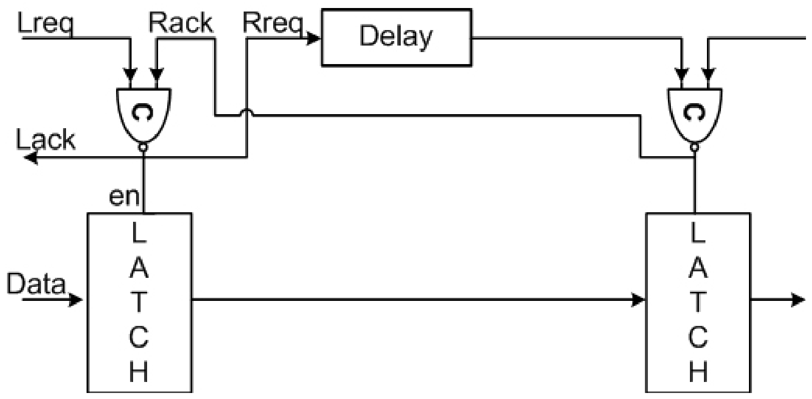


Figure 68 Simulated implementation for bundled data 4-phase protocol

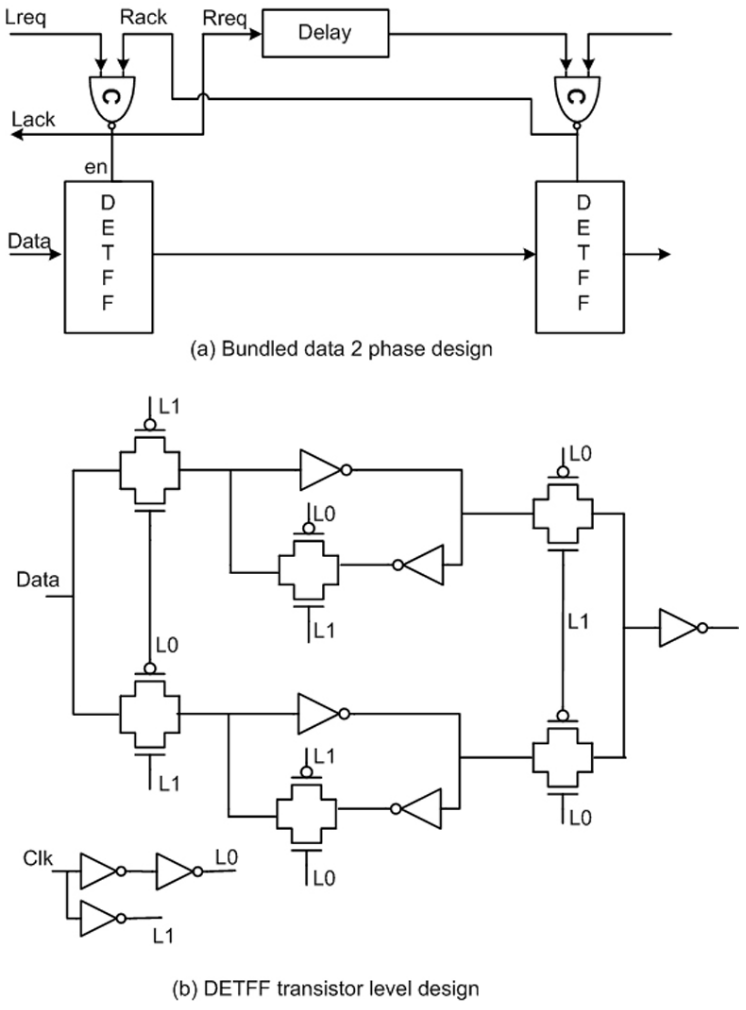


Figure 69. Simulated implementation for bundled data 2-phase protocol

8.5 Asynchronous communication

In this section we present the cycle time and energy models for some representative protocols used for asynchronous communication.

Name	equation	description
E_{gf}	$\frac{1}{A_b}\frac{E_{clkf}}{12}+\frac{E_{clkf}}{4}$	Flop clock gate energy
E_{gl}	$\frac{1}{A_b}\frac{E_{clkl}}{12}+\frac{E_{clkl}}{4}$	Latch clock gate energy
E_{fe}	$\frac{1}{A_b}2P_{dc}E_{clktree}+W_b(2(E_{gf}+E_{clkf})+A_w(E_{datf}+E_{rep}N_{rep}))$	Clk_flop energy
E_{le}	$\frac{1}{A_b}2P_{dc}E_{clktree}+W_b(4(E_{gl}+E_{clkl})+A_w(2E_{datl}+E_{rep}N_{rep}))$	Clk_latch energy
E_{4e}	$4(E_{ctl}+E_{rep}N_{rep})$ $+W_b(2E_{clkl}+A_w(E_{datl}+E_{rep}N_{rep}))$	4-phase bundled data energy
E_{2e}	$2(E_{ctl}+E_{rep}N_{rep})$ $+W_b(E_{clkdef}+A_w(E_{datl}+E_{rep}N_{rep}))$	2-phase bundled data energy
E_{ss}	$2(E_{ctl}+E_{rep}N_{rep})$ $+W_b(2E_{clkf}+A_w(E_{datl}+E_{rep}N_{rep}))$	src_sync energy
E_{di2}	$W_b(4E_{ctl}+2E_{rep}N_{rep})+2E_{rep}N_{rep}$	DI 1-of-2 energy
E_{di4}	$W_b(4E_{ctl}+E_{rep}N_{rep})+2E_{rep}N_{rep}$	DI 1-of-4 energy
E_{st2}	$2W_bE_{ctl}$	STFB 1-of-2 energy
E_{st4}	W_bE_{ctl}	STFB 1-of-4 energy

Table 11. Models for average energy per transaction per pipe stage

8.5.1 Bundled data protocol

The datapath in a bundled data asynchronous communication is similar to the datapath in synchronous communication. Controllers generate the local clock signals for the latches instead of the global clock, as shown in Figure 68. The bundled data protocol can be implemented in both four-phase and two-phase handshaking protocols. The key timing assumption (also known as bundling data constraint) in a bundled data protocol is that the data should be valid before there is an associated transition on the request line. To satisfy this constraint the request signal needs to be delayed using a delay line such that this delay is greater than the worst case delay of the data plus the setup time of the latch.

Equation C_{bd2+} and C_{bd4+} in Table 9 gives the minimum cycle time for a two phase and four phase bundled data protocol respectively. The same controller is used for both the 4-phase and 2-phase designs shown in Figure 68 and Figure 69. The key difference in these designs is the use of double edge triggered flip flops [56] rather than simple latches for 2-phase design. Parameter T_{fsm+} represents the controller delay overhead per phase and is equal to 4.25FO4 for a 4-phase protocol and 4.78FO4 for 2 phase protocol. The parameter T_{ps} in Table 9 is the worst case path separation or margin in terms of FO4 delays between the control and data paths. T_{cdel} is the number of gate delays that must be added to the control path to guarantee that we satisfy the bundling data constraint. The worst-case values must be taken assuming that as pipe stages are composed, it is possible to have worst-case skew between data and control in all stages. This creates a safe margin for the control and data race at the expense of throughput.

Equations E_{4e} and E_{2e} in Table 11 calculates the energy per pipeline stages for 4-phase and 2-phase bundled data protocols respectively. The energy required by the asynchronous controller per phase is characterized as E_{ctl}. This parameter represents the energy consumed in transmitting the control signals (req and ack) a critical distance along the communication link. The parameter E_{clkdef} in Table 6 represent the average energy consumed per transition by a double edge triggered flip flop shown in Figure 69.

The energy consumed per transaction per pipeline stage is the sum of the controller energy and the energy consumed for data transfer through latches. The controller energy for the 4-phase protocol is equal to $4*(E_{ctl}+E_{rep}N_{rep})$ and $2*(E_{ctl}+E_{rep}N_{rep})$ for the 2-phase protocol. The energy consumed by the latch in a 4-phase protocol is calculated as $W_b(2E_{clkl}+A_w(E_{datal}+E_{rep}N_{rep}))$ and for a 2-phase protocol energy is calculated as $W_b(E_{clkdef}+A_w(E_{datal}+E_{rep}N_{rep}))$.

8.5.2 Delay Insensitive protocol

The data path in a DI protocol is comprised of data encoded as 1-of-N rails where N wires are used to represent $\log_2 N$ bits of data and an additional wire which acts as the acknowledgment signal. The most well known of this class of protocols are dual rail (1-of-2) which uses two data wires per data bit, and 1-of-4 which uses four data wires to represent two bits of information.

Delay insensitive protocols are typically implemented using four-phase handshake protocols. Hence there are four iterations through the data acknowledge detection and propagation logic $4T_{fsm+}$. Figure 70 shows a 1 bit

WCHB [50] controller implementing a 1-of-2 DI protocol handshake. The acknowledgment wire can be made wider to reduce delay on its transitions $2O_{wd}T_{c+}$.

The equations that calculate the energy for communication using DI protocols are shown in Table 11. The asynchronous controller is comprised of domino logic which is responsible for forward data propagation along with a completion detection logic. The completion detection logic is responsible for generating the acknowledgment signal such that the ack signal becomes valid only when all the input bits have arrived. The overhead of the generation of the completion signal becomes a limiting factor for the cycle time of the design, particularly for wide data buses of 32 bits and above. The average value of the controller delay per phase T_{fsm} for this particular example of DI protocol is 6.65FO4. However this performance can be improved by using 2D [50] pipelining methodology which motivates a trade-off between performance and area. The main idea behind 2D pipelining is to reduce the cycle time overhead by using multiple completion detection logic modules, each module working on a subset of the data path at the cost of more aggregate area. For the sake of simplicity in this paper we will limit our discussion to only 1D pipelines; however faster values will be obtained for large buses with 2D pipelines.

Equation C_{di2+} and C_{di4+} in Table 9 gives the minimum cycle time for 1-of-2 and 1-of-4 DI protocol respectively. The data transitions have less variation because the encoding result in the wires being half shielded $2T_{csh+}$.

The energy required by the asynchronous controller per phase is characterized as E_{ctl}. Unlike bundled data protocols, in DI protocols the req control signal is encoded within the data itself and the controller shown in Figure 70 transmits 1 bit of data and ack control signal. The parameter E_{ctl} represents the energy of the controller per phase for transmitting 1 bit of data and ack control signal. The energy consumed per transaction per pipeline stage for a DI protocol using 1-of-2 encoding is the sum of the controller energy $4W_bE_{ctl}$ and the energy of the repeaters in that pipeline stage to drive the data $2W_bE_{rep}N_{rep}$ and the energy required to drive the acknowledgment wire $2E_{rep}N_{rep}$. In a 1-of-4 encoding each wire transition represents 2 bits of data, so the energy required by the repeaters to drive the data reduces by half to $W_bE_{rep}N_{rep}$.

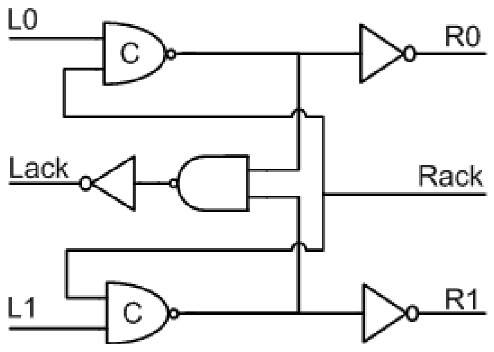


Figure 70. Simulated DI 1-of-2 protocol implementation

8.5.3 Single-track protocol

Single-track protocol is a 2-phase asynchronous protocol which uses 1-of-N data encoding similar to DI protocol with the key difference being that there is no separate acknowledgment wire [9]. The sender drives one of the N wires high thereby sending the data and, after receiving this data, the receiver sends an acknowledgment by driving the same 1-of-N wires low. After driving the wire to its desired state, the sender and receiver(s) must tri-state the wire to ensure that they do not try to drive the wire in opposite directions at the same time. One popular single-track asynchronous controller is Single-Track Full Buffer (STFB)[38]. Figure 71 shows a typical 1 bit 1-of-2 STFB buffer.

The two-phase single-track protocol reduces the overheads of the reset handshake. This yields substantially higher performance than 4-phase DI protocols and fewer transitions per bit which substantially lowers power. Compared to bundled-data protocols there are no timing assumptions that require margins on the forward latency, yielding additional performance improvement. However, the number of transitions per bit is often larger, producing higher average power. One limitation of STFB is that every repeater acts as a pipeline stage, hence STFB designs are restricted to fine-grained pipelining. Although STFB can be used in 1D pipelined designs similar to the DI protocol, this paper reports simulation results that are bit level 2D pipelined; i.e. each completion detection logic detects the validity of only a single bit.

Equation C_{st+} in Table 9 gives the cycle time for 1-of-2 and 1-of-4 STFB respectively. The average value of the controller delay per phase T_{fsm} for 1-of-2 STFB is 4.80FO4 and 5.10FO4 for 1-of-4 STFB. Equations E_{st2} and E_{st4} in Table 11 represent the energy required by an STFB controller to drive data through a critical distance communication link. E_{ctl} represents the energy per phase consumed by the controller. In case of 1-of-2 STFB designs the energy per pipeline stage is equal to the sum of controller energy and the energy of the repeaters in that pipeline stage $2W_bE_{ctl}$ and in case of 1-of-4 STFB designs this energy is equal to W_bE_{ctl}.

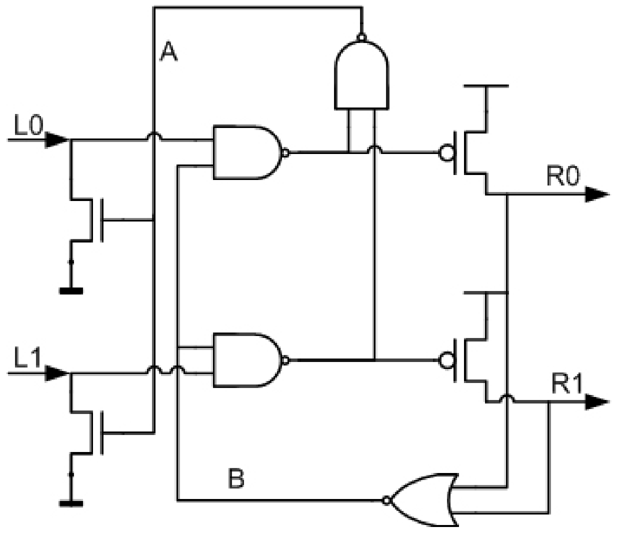


Figure 71. Simulated single-track 1-of-2 protocol design

8.5.4 Source synchronous communication

Another mode of data communication is wave-pipelining [17] where several "waves", i.e data signals, can concurrently propagate through a pipeline with proper timing separation between two consecutive waves. This timing separation is created by a "constructive clock skew" in conventional wave-pipelining, by using a separate timing reference in source synchronous protocols and a "fast" signal in an asynchronous "surfing" protocols [79]. The timing reference signal is routed in parallel with the data. A novel characteristic of asynchronous surfing is that when logic blocks receive the fast reference pulse their computation delay drops, thus creating a surfing effect wherein events are bound in close proximity with the pulse on fast reverence signal.

The key to the performance of any wave-pipelining method is the time separation of waves which defines their cycle time. In this paper, we have explicitly modeled the cycle time of source-synchronous communication with equation C_{ss+} in Table 9. In particular, the cycle time is lower bounded by three terms: the minimum pulse width that can propagate through a critical distance; the communication delay of the data through a repeated wire; and the propagation delay of the clock signal plus the delay through controller which generates the enable signal for the latch. Equation E_{ss} in Table 11 gives the energy model for source synchronous protocol. In the source synchronous models used here, multiple values are not propagated between flop stages.

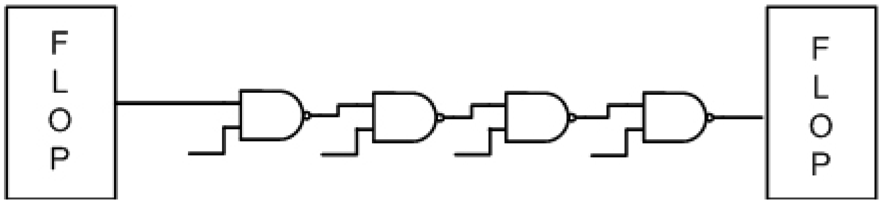


Figure 72. Logic pipeline with delay of 4

8.6 Comparisons

In this section we compare the various modes of communication in terms of maximum throughput, energy consumption and bandwidth they can provide. These results are based on the first order models developed in this paper using parameter values directly derived from SPICE simulations the designs in the 65nm process.

8.6.1 Throughput

Figure 72 shows a pipeline with an ideal logic delay of 4. Ideal pipelines would have no variation, and the flops would have zero latency and no overhead. This would allow a new data item to be propagated through these pipe stages every four gate delays. We compare the actual throughput and overheads of the protocols as calculated by the equations in Table 9 against the ideal pipeline delays. The equations calculate extra delays that are added due to device variation, latch delays, etc.

The throughput models applied to our parameter values are plotted in Figure 73 through Figure 76. The x-axis plots delays based on the pipeline granularity in terms of the number of gate delays between flops or latches. The leftmost side contains pipelines with a logic pipeline depth of 27.8 ideal gate delays per stage which is necessary to propagate a signal down a 10,000 µm wire. The rightmost point contains a single ideal gate delay per stage. The y-axis plots FO4 delays, or the overhead calculated by comparing the modeled worst-case delay to the ideal delay.

These plots allocate all of the delay to communication. As expected, the asynchronous protocols with acknowledgment are the least efficient for communication, with the 4-phase protocols being the worst. The most efficient protocols are the clocked and source-synchronous protocols. The two-phase asynchronous protocol shows significantly better performance due to the reduction in control transitions propagated between sender and receiver. The source synchronous protocol is the best asynchronous protocol at low frequencies because its request is propagated as a pulse and no acknowledgment is explicitly included. This protocol is marginally slower compared to synchronous protocols largely due to the conservative margin in the delay elements. In a real design, one would expect the source-synchronous circuit to out-perform the synchronous design due to its adaptive nature.

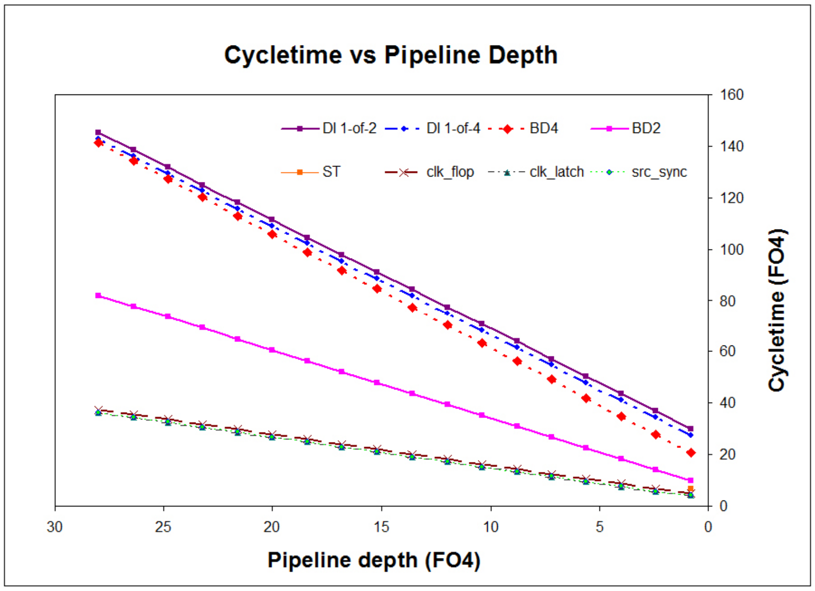


Figure 73: Worst-case protocol throughput

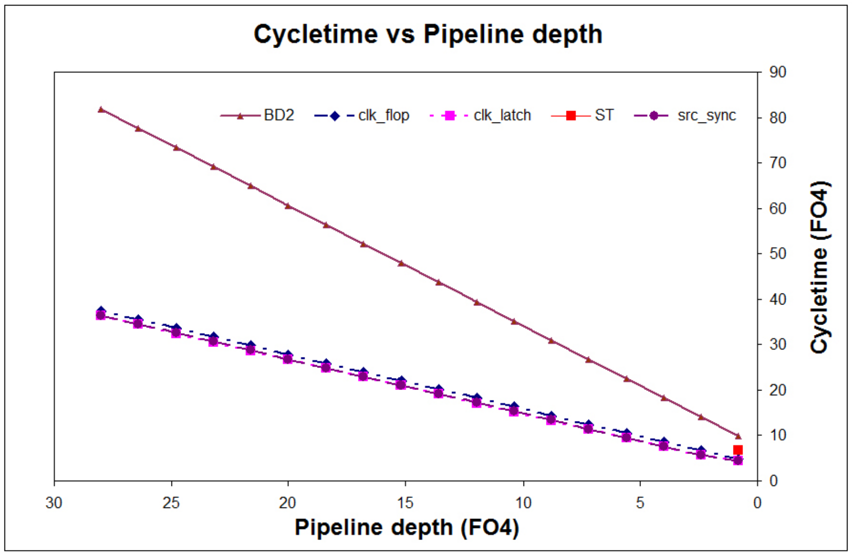


Figure 74. Throughput of efficient protocols

As mentioned earlier one of the limitations of single-track protocol is that every repeater needs to be a single-track pipeline stage, hence limiting single-track to fine-grained pipelines. Due to this limitation, we observe a single point in Figure 73 through Figure 76. In fact at high frequencies 2-phase single-track protocol is the most efficient asynchronous protocol. This is because it is two-phase and avoids the delay margin needed in bundled-data protocols.

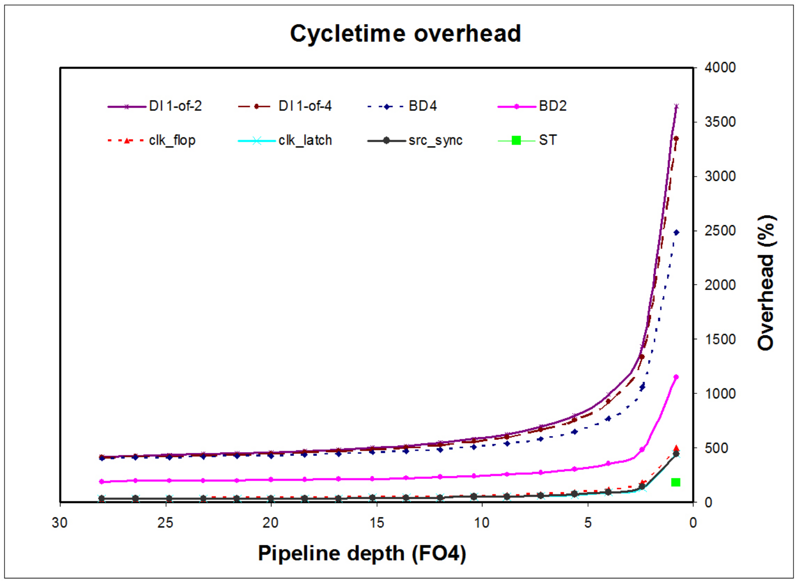


Figure 75. Throughput overhead against ideal

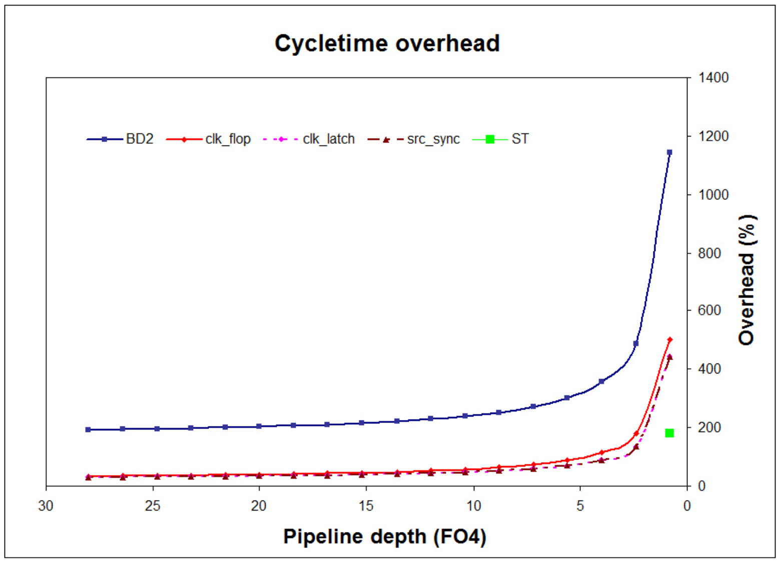


Figure 76. Overhead of efficient protocols

The cost of decreasing the amount of logic in each pipe stage to increase frequency can be inferred from these graphs. Figure 77 shows the historical trend in logic gates per pipe stage for Intel's recent microprocessors. As the amount of logic per stage is reduced, there is a considerable increase in power and performance sapping overhead. As shown in Figure 75 and Figure 76, shortening the pipe depth has come at little cost in the past. As pipe depths continue to be shortened, the overheads start to dramatically increase. In a synchronous latch design there is a 22% loss in efficiency as the pipe depth is decreased from 15 to 10 ideal gate delays. This balloons to a 50% loss if one moves from 10 to five ideal gate pipelines. Hence, future frequency increases for synchronous designs will result in diminishing performance gains. This also comes at an increased energy consumption cost as is show in the next section. The same analysis applies to pipelining asynchronous systems, particularly if the pipeline has significant bubbles during full throughput operation.

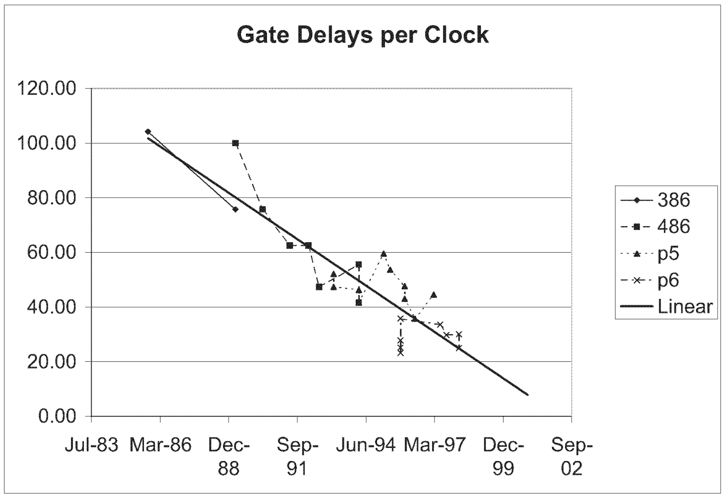


Figure 77. Gate delay scaling of products [41]

8.6.2 Latency

Figure 78 and Figure 79 compare the latency and overhead per pipeline based on the equations given in Table 10. The x-axis plots delays based on the pipeline granularity in terms of the number of gate delays between flops or latches. The overhead is calculated by comparing the latency of the design to the wire latency given by parameter L_W in Table 6. These plots allocate all of the data path delay to communication.

In case of ideal pipelines there are no overheads associated with the memory elements hence the latency is the same as wire delay. However in case of synchronous designs, especially synchronous flop communication, each stage has to suffer an overhead to account for clock to output delay, set-up delay, and clock skew margin. As we increase the pipeline granularity the overhead increases as much as 200% as shown in Figure 79. Margins for set-up time and clock skew are not required for synchronous latch communication due to time borrowing between stages. However this protocol suffers the overhead of two data-to-output delays $2T_{dql}$ through the latches.

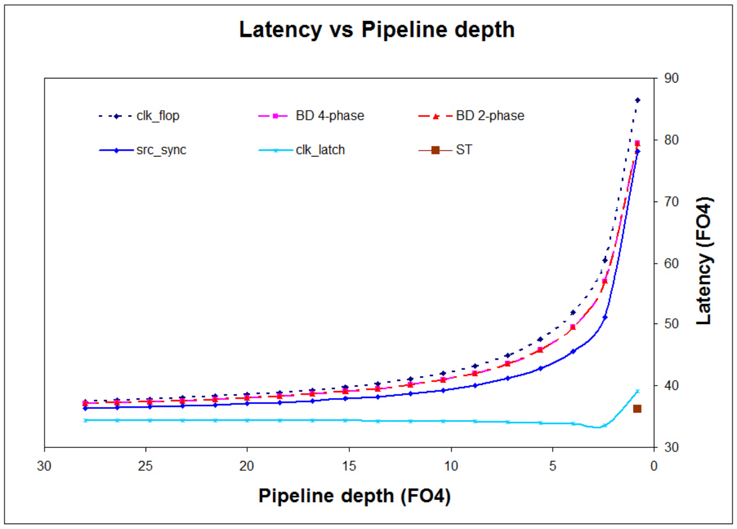


Figure 78. Latency vs pipeline depth

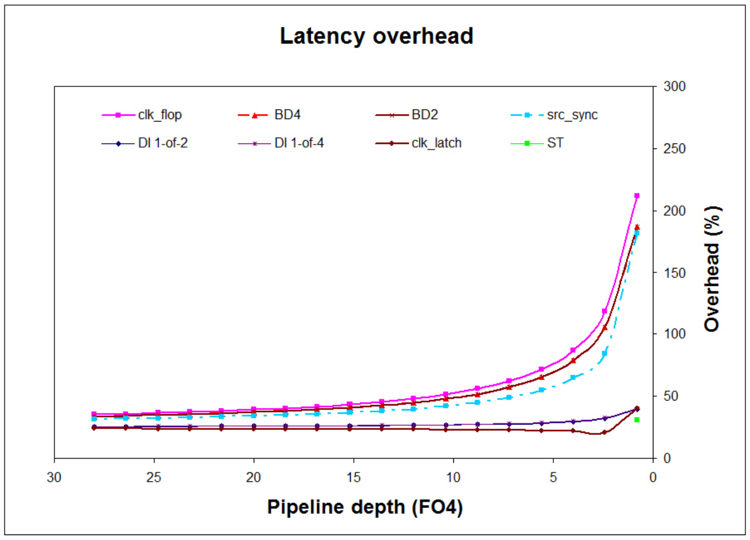


Figure 79. Latency overhead vs pipeline depth

The overhead associated with bundled data asynchronous designs comes from two sources. The first source is the delay associated with the controller used to generate local clock signals. The second is the extra delay margin inserted in the request line as shown in Figure 68 to satisfy bundling data constraint. The DI protocols and single-track protocol provides the lowest latency of all the protocols. This can be attributed to two properties of these circuits. First, since the data coding is delay-insensitive in nature no extra margins are required for setup to the latches. Second, the use of fast domino logic used in the data path leads to faster propagation of data compared to latch / flop designs.

8.6.3 Energy

Figure 80 shows the energy dissipated to transmit data across a 10,000 μm bus with varying amounts of pipelining. Each equation in Table 11 calculates the energy for a single pipeline stage. This number is multiplied by the number of pipeline stages N_P which scales the results for the same 10,000mm distance. The x-axis shows the number of repeaters between the flops or control elements in the pipeline, the rightmost value when every inverter is replaced with a flop.

The asynchronous DI protocol based on 1-of-N codes has a much poorer power profile due to their activity factors. The rest of the protocols have fairly similar energy profiles. The curves are very flat except for ultra-fine grain pipelining, where the control overhead becomes a significant power drain on the circuits. Figure 81 shows the most efficient protocols at high frequency where bus activity factor is set at 5% and the activity factor of the data wires is set to be 18%. This shows that at high frequencies when some idle cycles are present the bundled data asynchronous protocols have a distinct advantage over synchronous protocols. This can be explained because of clock switching and energy required in the gating logic even during idle phases, while asynchronous protocols provides ideal clock gating at no extra cost. Observe that while the plots shows asynchronous 2-phase bundled data protocols to be more energy efficient then 4-phase bundled data protocols, this may not always be true.

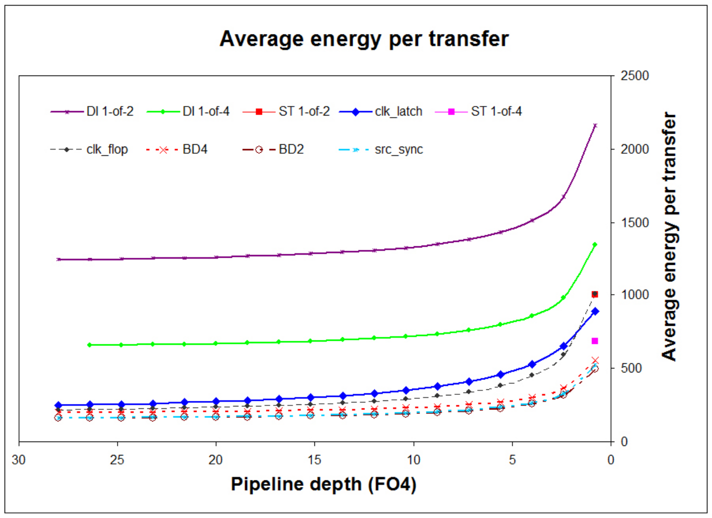


Figure 80. Average energy per transaction

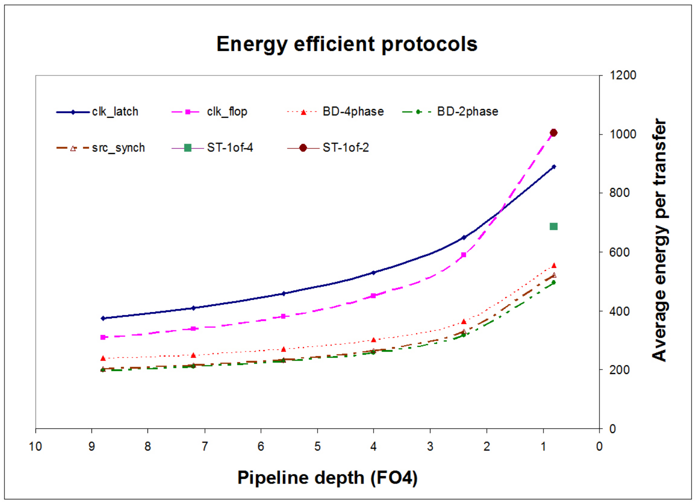


Figure 81. Highly pipelined transfer energy

8.6.4 Bandwidth

Name	equation	description
B_v	$1 + 1/W_b$	synchronous valid bit
B_{da}	$2 + O_{wa}/W_b$	DI data encoding & ack
B_{st}	2	Single-Track data encoding
B_{ra}	$1 + 2O_{wa}/W_b$	req and ack signals
B_r	$1 + O_{wa}/W_b$	req signal

Table 12. Average wires per data bit

Name	equation	description
B_{cf}	$L_w/(B_v C_{ff+})$	Clk_flop bandwidth
B_{cl}	$L_w/(B_v C_{l+})$	Clk_latch bandwidth
B_{di}	$L_w/(B_{da} A_{di+})$	1-of-2-rail and 1-of-4 DI b/w
B_2	$L_w/(B_{ra} A_{2+})$	2-phase bandwidth
B_4	$L_w/(B_{ra} A_{4+})$	4-phase bandwidth
B_{ss}	$L_w/(B_r SS_+)$	Src_sync bandwidth
B_{st2}	$L_w/(B_{st} A_{st2+})$	STFB 1-of-2 bandwidth
B_{st4}	$L_w/(B_{st} A_{st4+})$	STFB 1-of-4 bandwidth

Table 13. Bandwidth equations for the eight protocols

In order to effectively scale a chip, additional metal layers are added to support the increased bandwidth of the design. The effective utilization of wires in a process therefore determines the cost and bandwidth of the design. This work defines bandwidth to be proportional to the wire area. Table 12 shows how throughput is scaled based on the wire area to calculate the bandwidth of a design. The delay insensitive and single-track protocols modeled in this paper require two wires per data bit. Therefore, for the same wire area the bandwidth of these asynchronous designs would have roughly half the bandwidth. The penalty for control signal and valid bit overhead are also calculated for the studied designs as shown.

The final bandwidth models graphed in this paper are shown in Table 13. The value is the bandwidth-scaled number of data words that can be pipelined in the 10,000mm wire. This allows us to compare bandwidth and area for various levels of pipelining across all the designs. The bandwidth values from this table is used to calculate the x-axis point for each protocol and the energy numbers are used for the y-axis values of Figure 62, Figure 63 and Figure 85.

8.7 Analytical models vs simulation

The accuracy of the analytical models were evaluated by comparing the results from the first-order models against SPICE simulations of layout of the complete 65nm designs. Cycle time, latency, and energy required to transfer data on a 10,000mm long interconnect by varying the pipeline granularity were compared against SPICE simulations for three target designs. Figure 82 through Figure 84 compare the cycle time, energy per transfer, and latency for asynchronous 4-phase bundled data, asynchronous 2-phase bundled data and synchronous flop mode of communication. The x-axis plots delay based on the pipeline granularity in terms of the number of gate delays in a single pipeline stage.

Figure 82 compares the cycle time while Figure 83 compares the latency. The proposed models account for worst case process variations, such as delay margin to be added in case of bundled data protocols and clock skew, which are ignored in the spice simulations. Hence for fair comparisons we have set the parameters T_{cdel}, T_{skj} and T_{cad} to 0. Figure 84 compares the energy required per transaction. No clock gating is assumed in the spice simulations for the case of synchronous flop communication. Thus for fair comparisons the parameter E_{gf} in our models is set to zero.

From the above comparisons we can see that the proposed models are fairly accurate as the errors in case of cycle time comparison is less then 15%, less then 20% in case of latency comparisons and less then 10% for energy comparisons. These errors can be attributed to the models assuming a worst case environment which is difficult to include in our spice simulations. For example even though our spice simulation set up capture the loading effects of coupling capacitance but it does not model the worst case crosstalk noise. The shapes of the curves are similar, validating the proposed models.

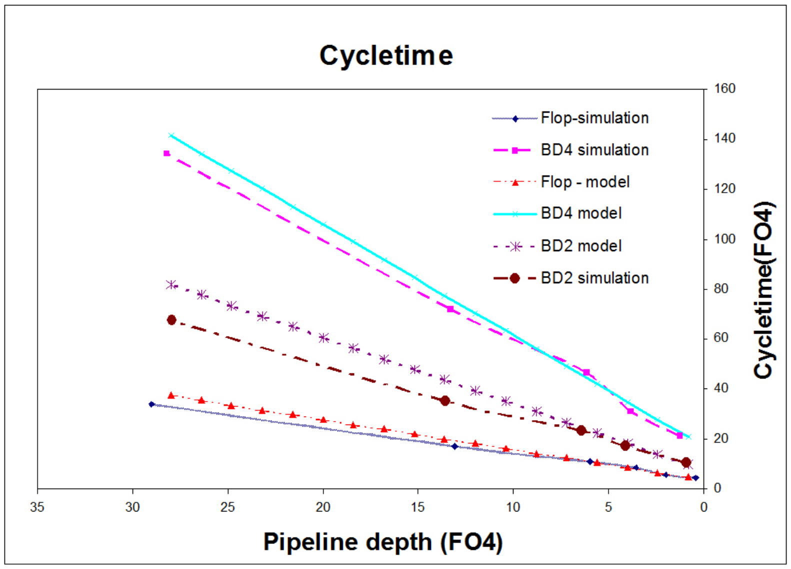


Figure 82. Cycle time comparison of analytical models and SPICE simulation

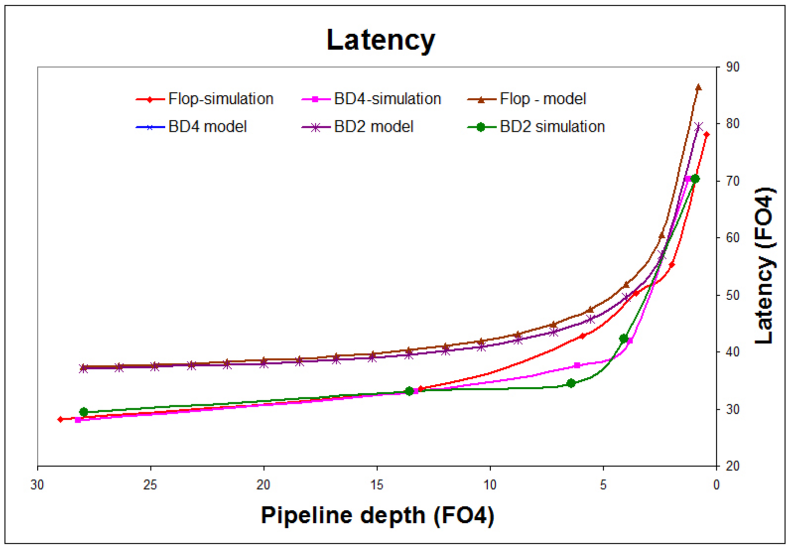


Figure 83. Latency comparison between models and SPICE simulations

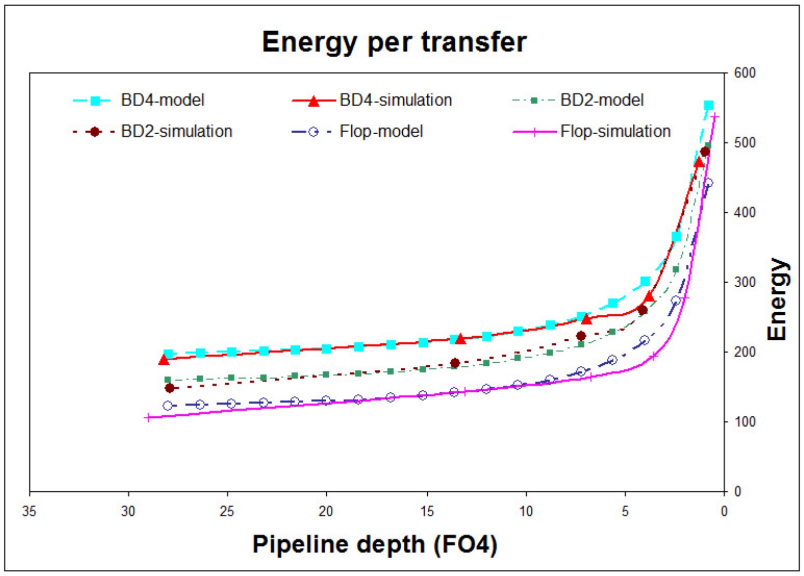


Figure 84. Energy per transfer comparison between models and simulation

8.8 Observation and caveats

Accurate apples-to-apples comparisons are always challenging. These equations can model the first-order effects of most implementation styles. The magnitude of many of these effects, such as first-droop, can be arguable. Hence the models are highly parameterizable to match the argued effect by applying the parameter values to the equations. This also allows one to study the results of an effect that is trending up or down as processes are scaled.

Simulating all of the various architectures is outside the scope of this work, and thus the accuracy bound reported here may not be a worst case. The proposed models are therefore validated by performing SPICE simulations on a subset of the protocols. Further, there are some effects which are difficult to simulate in a simple simulation environment like worst case crosstalk noise, clock tree and clock gating and hence are not been considered during SPICE simulations.

There are other effects that are also difficult to compare. The specific design of the flop and latch, for instance, will have impact on power and performance. This work selected a single simple flop and latch style and applied it to all protocols. This made the work relatively accurate between models, but the absolute value will be different based on the design used. However, the largest difference in most flop designs is the energy required in the clocking. Since this can be determined separately from data energy and delays the variation across flop styles should be rather simple to estimate.

This work can also be applied directly to pipeline protocols for logic blocks, and some of the protocols include this modeling. However, the logic family, design and protocol style, and implementation aspects of the logic blocks have a much greater variability than a repeated wire, so the accuracy of such a model is much more difficult to compare and arguably less accurate. Min-delay issues become very important, but they can largely be ignored in communication since max and min signal propagation delays will be similar.

Fixed bandwidth can be achieved through combinations of throughput and parallelism. For instance doubling the throughput and halving the wires will achieve the same bandwidth, at a smaller area. This can be accomplished using these models. The activity factors for the serialized links may increase significantly.

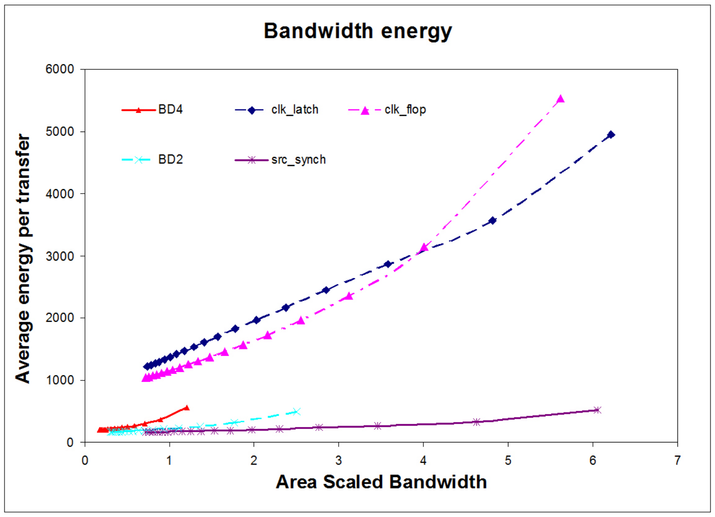


Figure 85. Bandwidth/energy for lower bus utilization

The parameter values used in this paper represent a single design point. For example, increasing the activity factor of the bus by a factor of 10 makes no significant change to Figure 62 and Figure 63. However, decreasing the activity factor by the same amount gives a significant advantage to the asynchronous protocols as shown in Figure 85. Here the 4-phase, 2-phase and source synchronous protocols show significant energy advantages at all bandwidths over synchronous designs. Thus, the parameters need to be configured based on your target application.

Optimization based on these models has not been done. For instance, there is a tradeoff between faster control signal propagation and its deleterious effect on bandwidth.

8.9 Conclusion

Parameterized first-order analytical models are presented for many communication models. SPICE simulation of three of the protocols show the analytical models to be accurate within 15% for cycletime, 20% for latency, and 10% for energy across the full range of pipeline depths.

The parameters can be modified to quickly compare various communication protocols operating at different design targets, such as low frequency and power designs to performance constrained targets. Comparisons and conclusions between various protocols have been performed based on varying the parameters of the models.

The efficiency of asynchronous communication is very dependent on the protocol. Communication is one of the worst-case scenarios for asynchronous design due to the latency of the handshake signals. These latencies require most asynchronous protocols to be pipelined deeper than the clocked protocols to achieve similar bandwidths. Yet the efficient asynchronous protocols are shown to have similar results to the synchronous protocols when measuring average transaction energy for a target bandwidth using parameters targeted for a microprocessor bus.

Many design parameters favor either a synchronous or asynchronous style. Wider buses and high bit-level activity factors favor asynchronous communication. Higher bus utilization factors favor synchronous designs. Changing the operating environment reflected in these parameters can result in asynchronous communication being far superior to synchronous protocols.

The results in this paper demonstrate that energy per transaction as well as latency and cycle time overheads remains relatively flat across most pipelined designs until pipelining becomes aggressive. At that point there begins to be a considerable penalty for increasing the pipelining. Thus there is a broad range of pipeline frequencies that can be implemented with relatively small energy and performance penalty.

Asynchronous designs can exploit this flat region of the graphs since pipelining frequency can be dynamically chosen. This is not the case with clocked protocols where distances and pipelining are fixed relative to the clock frequency. This implies that scalability and the ability to optimize for a particular power/performance point is enhanced in asynchronous designs. Asynchronous designs also provide substantial latency advantages over synchronous designs which are very important in high performance network-on-chip designs [36]. Furthermore, asynchronous designs can be repeated and pipelined at the optimal critical distance for the target topology without requiring margin for future process scaling.

This study only compares the physical data transmission efficiencies. Communication effects on the overall processor performance and power are an important extension [43]. The benefits of implementing asynchronous communication in an otherwise globally synchronous processor must override the cost of synchronization with the destination frequency. Future designs – such as those with multiple on-die cores or designs with power islands – will decompose the chip into different clock domains for power, thermal, and performance reasons. For such architectures, asynchronous communication exhibits significantly lower latency and has been shown, through this study, to have similar or better physical transport efficiencies when compared to synchronous methodologies.

9 Summary and Conclusions

Despite all the advantages of some asynchronous design styles they are still not widely accepted among semiconductor industry. One of the key remaining roadblocks is the overwhelming design time required for design and verification. In order to compete with semi custom synchronous ASIC designs we need asynchronous design styles that can be easily verified along with an automated ASIC CAD flow that supports them. This thesis develops a new template design based asynchronous design style coupled with a fully-automated synthesis and place-and-route CAD flow to achieve this goal. In particular, we propose several novel non homogenous single-track templates.

Non homogeneous single-track templates follow a two phase static single-track handshake protocol. Along with using faster domino logic in their data path, they do not require any margins to satisfy setup time as in bundled data asynchronous designs and thus have relatively very low latency. Consequently, these design templates provide significant overall performance advantages for latency critical systems (e.g. Turbo decoding). To quantify the benefits of static single-track circuits we implemented an asynchronous turbo decoder using SSTFB standard cell library in IBM 0.18µm technology. Comparisons shows that the asynchronous turbo decoder can provide up to 1.3X-4X advantage over synchronous turbo decoder in throughput per area for block sizes of 2048-768 bits. Moreover, due to low latency advantages, it can support higher throughput of designs with small block sizes than possible with its synchronous counterpart.

Non homogeneous single-track circuits have the flexibility of having multiple levels of logic in a pipeline stage and can have different pulse width on drivers. The flexibility of having multiple level of logic in a pipeline stage improves area and energy efficiency by sharing the overhead of handshake control logic among multiple levels of logic. The flexibility of having different pulse width on drivers makes the design more robust to crosstalk noise and process variations. To ensure robust designs using non homogeneous pipeline stages we propose two design constraints. The first constraint is the *single-track handshake constraint* (STHC) which constrains the pulse width on the drivers of the sender and receiver pipeline stages such that only one of them is actively driving the communicating wires (channels) at a given time. The other constraint is the *rail-2-rail swing constraint* which constrains the minimum pulse width on the drivers while actively driving the channels.

In this thesis three non homogeneous single-track design templates have been developed. These design templates provides a trade-off between area and performance such that while SLST design template aim for high performance applications and supports single level of logic per pipeline stage, MLST and MLDST aim for medium to low performance applications and can support multiple levels of logic in a pipeline stage. We discuss several implementation issues including performance model of these templates and have developed behavioral models of logic cells and controllers as a part of standard cell library development.

To increase the acceptability of our single-track design style we also addressed some of the key EDA tool issues. We demonstrate the feasibility of library characterization and back-end SDF back annotation for single-track asynchronous designs and present a RTL to asynchronous synthesis flow targeting the proposed single-track templates.

- The library characterization and SDF back annotation flow reduces the design time by using faster Verilog simulation flow than conventional time consuming analog verification flow. Experimental results on a 260K transistor 64 bit prefix adder design indicates that the back annotation flow yields over two orders of magnitude advantage in simulation speed on analog verification flow with less than 5% error.
- We propose an automated synthesis flow for static single-track asynchronous designs via an expansion of the capabilities of the CAD tool Proteus developed at USC. This tool takes in synchronous RTL netlist as input and generates the corresponding asynchronous design for a given target performance. As a part of this thesis the tool was modified to incorporate the ability to generate static single-track designs, perform slack matching for two phase circuits and automate the generation of test bench for verification. With the integration of these design templates in Proteus one can take the advantage of re-pipelining to generate area efficient designs as well as automate the process of slack matching. Comparisons on several ISCAS benchmarks show that the proposed templates provide a wide range of throughput area tradeoffs. For high performance applications SLST templates provide an average of 20% better throughput than QDI pipelines at the cost of 36% more average area. At lower frequencies MLST template provides 1.32X and 1.75X and MLDST pipeline templates provide 1.5X and 2.34X better average throughput / area over QDI and MLD pipelines respectively.

With the increasing number of components in a system-on-chip (SoC) there is a need for an efficient data communication system. Network-on-a-Chip (NoC) is a new approach to design the communication subsystem of SoC. However a single global frequency may not be optimal because each module has a different optimal power/performance point which makes asynchronous designs an attractive choice for implementing network-on-chip. In this thesis we extend the work done in [68] by characterizing various communication protocols to get the protocol specific value of the parameters used in the models, and including the impact of clock distribution networks for synchronous communication protocols. We also included two phase asynchronous single-track handshake protocol in our comparisons. Comparisons show that for wider data path and low bus activity factors asynchronous bundled data protocol provide significant energy advantages than synchronous protocols. Comparisons also show that single-track protocol provides 33% better throughput and 12% lower latency compared to other asynchronous protocols.

There are still challenges that need to be addressed before we can expect widespread adoption of single-track circuits. Specifically, we need to extend the support of standard ASIC commercial CAD tools like PrimeTime, SoC Encounter ,and Celtic for timing and signal-integrity driven place and route and sign off. In addition, given the growing importance of power consumption in VLSI industry, additional research needs to be done to explore ways to reduce power consumption of the proposed single-track template by reducing the activity factor. Also work needs to be done to quantify the EMI advantages of the proposed templates over state of the art synchronous methods. The next step in asynchronous NoC is to compare asynchronous and synchronous communication protocols with flow control mechanism. Specifically there is a need to study how various communication protocols will compare against each other where there is congestion in the network.

REFERENCES

[1] Achronix Semiconductor Corporation, www.achronix.com.

[2] ASU Predictive spice models. http://www.eas.asu.edu/~ptm/.

[3] Bainbridge W. J. and Furber S. B. Delay insensitive system-on-chip interconnect using 1-of-4 dataencoding. In Proc. 8th International symposium on Asynchronous Circuits and Systems, pages 118–126, 2001.

[4] Bainbridge W. J. Asynchronous System-on-Chip Interconnect. PhD thesis, Department of Computer Science, University of Manchester, March 2000.

[5] Bardsley A. and Edwards D. A. The Balsa Asynchronous Circuit Synthesis System. in Proc. Forum on Design Languages*,* Sept. 2000.

[6] Beerel P. A. and Chugg K.M. A Low Latency SISO with Application to Broadband Turbo Decoding, IEEE Journal on Selected Areas in Communications, Vol : 19 Issue: 5, May. 2001.

[7] Beerel P. A. and Roncken M. E. Low power and energy-efficient asynchronous design. *Journal of Low Power Electronics, 2007.*

[8] Beerel P. A., Chugg K., Dimou G., and Thiennviboon P. Reduced-latency soft-in/soft-out module. USPTO Application #20040237025, Nov. 2004.

[9] Beerel P. A., Lines A., Davies M., Kim N. H. Slack matching asynchronous designs. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'06), 2006

[10] Berkel K. van and Bink A. Single-track handshake signaling with application to micropipelines and handshake circuits. Proceedings of ASYNC'96.

[11] Berkel K. van, Burgess R., Kessels J., Peeters A., Roncken M., Schalij F., and Wiel R. van de. A Single-Rail Re-implementation of a DCC Error Detector Using a Generic Standard-Cell Library," in *Proc. Asynchronous Design Methodologies*, May 1995, pp. 72–79.

[12] Berrou C., Glavieux A. and Thitimajshima P. Near shannon limit error – correcting coding and decoding – Turbo Codes.

[13] Bink A. and York R. ARM996HS: the first licensable, clockless 32 bit processor core, IEEE Micro Mar-Apr 2007.

[14] Bjerregaard T. and Mahadevan S. A survey of research and practices of Network-on-Chip." In: *ACM Computing Surveys (CSUR),* Volume 38, Issue 1, pages 1-51, 2006.

[15] Bougard B., Giulietti A., Van der Perre L., Catthoor F. A Class of Power Efficient VLSI Architectures for High Speed Turbo-Decoding, IEEE Global Telecommunications Conference, Vol. 1, pp. 549 - 553,2002

[16] Bowman K., Duvall S. and Meindl J.. Impact of die-to-die and within die parameters fluctuations on the maximum clock frequency distribution for gigascale integration. *IEEE Journal of Solid-State circuits, 37(2):183-190, 2002.*

[17] Burleson W. P., Ciesielski M., Klass F., and Liu W. Wave-Pipelining: A Tutorial and Research Survey. In IEEE Transactions on Very Large Scale Integraton Systems, pages 464–474, 1998.

[18] Cao Y., Sato T., Sylvester D., Orshansky M., and Hu C. New paradigm of predictive MOSFET and interconnect modeling for early circuit design. In Proc. Custom Integrated Circuits Conference, pages 201–204, 2000.

[19] Chugg K. M., Anastasopoulos A., and Chen X. Iterative Detection: Adaptivity, Complexity Reduction, and Applications. Kluwer Academic Press, 2000.

[20] Cortadella J., Kishinevsky M., Burns S. M., and Stevens K. S. Synthesis of asynchronous control circuits with automatically generated relative timing assumptions. ICCAD.99. Nov. 1999

[21] Cortadella J., Kondratyev A., Lavagno L., and Sotiriou C. De-synchronization: Synthesis of Asynchronous Circuits from Synchronous Specification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. Oct. 2006.

[22] Dally W. J. and Towles B. Route packets, not wires: on-chip interconnection network, DAC, 2001, pp. 684– 689.

[23] Dimou G. Clustering and fanout optimizations of asynchronous circuits. *Ph.D. Thesis, University of Southern California, 2004.*

[24] Dobkin R. and Peleg M. Parallel Interleaver design and VLSI Architecture for Low Latency Map Turbo Decoders, in IEEE Transactions on VLSI Systems, April 2005.

[25] Dobkin R. and Peleg M. Parallel Interleaver design and VLSI Architecture for Low Latency Map Turbo Decoders, in IEEE Transactions on VLSI Systems, April 2005.

[26] Dobkin R., Morgenshtein A., Kolodony A., and Ginosar R. Parallel vs Serial On-Chip Communication. In Proc. International workshop on System level interconnect prediction, pages 43–50, 2008.

[27] Dobkin R., Perelman Y., Lian T., Kolodony A., and Ginosar R. High Rate Wave-pipelined ASynchronous On-Chip Bit-Serial Data Link. In Proc. 13th International symposium on Asynchronous Circuits and Systems, pages 3–14, 2007.

[28] El-Moursy M. A. and Friedman E. G. Optimal wiring of RLC interconnect with repeaters. In Proc. 13th ACM Great Lakes symposium on VLSI, pages 27–32, 2003.

[29] Fant K. M. and Brandt S. A. NULL Conventional Logic: A Complete and Consistent Logic for Asynchronous Digital Circuit Synthesis, in International Conference on Application-specific Systems, Architectures, and Processors. 1996. p. 261--273.

[30] Ferretti M. Single-track Asynchronous Pipeline Template. *Ph.D. Thesis, University of Southern California, 2004.*

[31] Ferretti M. and Beerel P. A. Single-Track Asynchronous Pipeline Templates using 1-OF-N Encoding . In Proc. Design and Test Automation in Europe (DATE), March 2002.

[32] Ferretti M., Ozdag R. O. and Beerel P. A. High performance asynchronous ASIC back-end design flow using single-track full buffer standard cells. Proceedings of ASYNC 2004.

[33] Fulcrum Microsystems Inc., www.fulcrummicro.com.

[34] Furber S. B. AMULET2e: An asynchronous embedded controller. Proceedings of IEEE volume 87, February 1999.

[35] Furber S. B. and Day P, "Four-Phase Micropipeline Latch Control Circuits," *IEEE Transactions on VLSI Systems*, vol. 4, no. 2, pp. 247–253, June 1996.

[36] Gebhardt D. and Stevens K. S.. Elastic Flow in an Application Specific Network-on-Chip. volume 200, pages 3–15, February 2008. Elsevier.

[37] Golani P. and Beerel P. A. Back-annotaion in High-Speed Asynchronous Design. JOLPE 2005

[38] Golani P. and Beerel P. A. High performance noise robust asynchronous circuits. ISVLSI 2006

[39] Golani P., Dimou G. D., Prakash M. and Beerel P. A.. Design of high speed asynchronous turbo decoder. ASYNC 2007

[40] Handshake Solutions, www.handshakesolutions.com.

[41] Ho R., Mai K. W., and Horowitz M. A. The future of wires. Proceedings of the IEEE, 89(4):490–504, April 2001.

[42] International Technology Roadmap for Semiconductors – 2005 Edition.

[43] Iyer A. and Marculescu D. Power-Performance Evaluation of Globally Asynchronous, Locally Synchronous Processors. In Proc. Intl. Symposium on Computer Architecture (ISCA), pages 158–168, Anchorage, AK, May 2002.

[44] Joshi P. Static timing analysis of GasP M.S.Thesis, University of Southern California, 2008.

[45] Kelly IV C., Biermann D., Fang D., Teifel J., and Manohar R. Energy-Efficient Pipelines. Proceedings of the 8th International Symposium on Asynchronous Circuits and Systems, Manchester, UK, March 2002

[46] Kessels J. and Peeters A. The Tangram Framework: Asynchronous Circuits for Low Power. in Proc. of Asia and South Pacific Design Automation Conference, Feb. 2001, pp. 255–260.

[47] Kim H., Beerel P. A. and Stevens K. S. Relative timing based verification of timed circuits and systems, ASYNC'2002. April 2002.

[48] Krol T., Kristic M., Fan X. and Grass E. Modeling and reducing EMI in GALS and synchronous systems. PATMOS 2009.

[49] Linder D. H. and Harden J. C. Phased Logic: Supporting the synchronous design paradigm with delay insensitive circuitry. IEEE transactions on computer. 1996, 45(9):p 1031 – 1044.

[50] Lines A. M. Pipelined asynchronous circuits. *Master's thesis, California Institute of Technology, 1995.*

[51] Martin A. J., Nyström M., and Wong C. G. Three Generations of Asynchronous Microprocessors." IEEE Design & Test of Computers, special issue on Clockless VLSI Design, November/December 2003.

[52] Masera G., Piccinini G., Ruo Roch, M. and Zamboni M. VLSI Architectures for Turbo Codes IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 7, No. 3, September 1999

[53] Nystrom M. and Martin A. J. Asynchronous Pulse logic, Kluwer Academic Publishers, Inc 2002

[54] Ozdag R. and Beerel P. A. "A Channel Based Asynchronous Low Power High Performance Standard-Cell Based Sequential Decoder Implemented with QDI Templates," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Apr. 2004, pp. 187–197.

[55] Pangjun J. and Sapatnekar S. S.. Low power clock distribution network using multiple voltages and reduced swings. *IEEE transaction on very large scale integration systems, 10(3):309-318, 2002.*

[56] Pedram M., Wu Q., and Wu X. A new design of double edge triggered flip-flops. In Design Automation Conference, pages 417–421. ACM/IEEE, Feb 1998.

[57] Piestrak S. J. Membership test logic for delay-insensitive codes. In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 194–204, 1998.

[58] Renaudin M., Vivet P., and Robin F. "ASPRO-216: A standard-cell QDI 16-bit RISC asynchronous microprocessor," in Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, 1998, pp. 22–31.

[59] Seitz C. L.. System timing. In Introduction to VLSI Systems, chapter 7. Addison Wesley, 1980.

[60] Silistix, www.silistix.com.

[61] Singh M. and Nowick S. M. High Performance Asynchronous Pipelines for Fine Grained Dynamic Datapaths. . Proceedings of ASYNC 2000.

[62] Singh M. and Nowick S. M. High-throughput asynchronous pipelines for fine-grain dynamic datapaths. In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 198–209. IEEE Computer Society Press, April 2000.

[63] Singh M. and Nowick S. M. Mousetrap: high speed transition-signaling asynchronous pipelines. IEEE Transactions on Very Large Scale Integration (VLSI) systems, pages 684–698, June 2007.

[64] Singh M. and Nowick S.M. MOUSETRAP: Ultra-High speed Transition Signaling Asynchronous Pipelines. ACM/IEEE International workshop on timing issues in the specification and synthesis of digital systems. Dec 2000.

[65] Smirnov A., Taubin A. and Karpovsky M. Automated pipelining in ASIC synthesis methodology: Gate transfer level. In IWLS 2004 thirteenth international workshop on logic and synthesis 2004.

[66] Standard Delay Format Version 3.0 http://www.eda.org/sdf/sdf_3.0.pdf

[67] Stevens K. S. and Ginosar R. Relative timing ASYNC'99. April 1999

[68] Stevens K. S. Energy and performance models for clocked and asynchronous communication*. Ninth International symposium on asynchronous circuits and systems, 2003.*

[69] Stevens K. S., Rotem S. , Ginosar R. , Beerel P. A., Myers C. , Yun K. , Kol R. , Dike C. , and Roncken M. An Asynchronous Instruction Length Decoder. IEEE Journal of Solid State Circuits, 36(2):217–228, February 2001.

[70] Sutherland I. E. . Micropipelines. In Communications of the ACM, pages 720–738, 1989.

[71] Sutherland I. E. and Fairbanks S. GasP: a Minimal FIFO Control. *Seventh* International symposium on asynchronous circuits and systems, 46-52, 2001.

[72] Sutherland I. E., Sproull B., and Harris D. Logical Effort: Designing Fast CMOS Circuits. Morgan Kaufmann Publishers, Inc., San Francisco, 1999

[73] Synopsys Liberty Format http://www.synopsys.com

[74] Teifel J. and Manohar R.. An asynchronous data flow FPGA architecture," IEEE Transactions on Computer, vol. 53, issue 11, Nov. 2004, pp. 1376 – 1392.

[75] Thiennviboon P. and Chugg K. M. A Low-Latency SISO via Message Passing on a Binary Tree, Allerton Conf., Urbana, IL, Oct. 2000.

[76] Verilog-HDL by Sameer Palnitkar, Pearson Education June 2003

[77] Viglione F., Masera G., Piccinini G., Ruo Roch M., and Zamboni M. A 50 Mbit/s Iterative Turbo Decoder, Proceedings Design, Automation and Test in Europe Conference and Exhibition 2000, pp. 176 –180, 2000

[78] Viterbi A. J. An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes, IEEE Journal on Selected Areas in Communications, Vol.16, No. 2, February 1998.

[79] Winters B.D. and Greenstreet M.R. A negative-overhead self-timed pipeline. In Proc. 8th International symposium on Asynchronous Circuits and Systems, pages 32–41, 2002.

[80] Yoneda T., Kitai T., and Myers C. J. Automatic Derivation of Timing Constraints by Failure Analysis. CAV.2002, July 2002.

[81] Yun K. Y., Beerel P. A., Vakilotojar V., Dooply A. E. and Arceo J.. The design and verification of a high performance low control overhead asynchronous differential equation solver. IEEE transactions on VLSI systems, Dec 1998.