A Novel Technique Towards Eliminating the Global Clock in VLSI Circuits

G. Hazari
PART, Poona
ghaza@iitbombay.org

M. P. Desai
Dept. of Elect. Engineering
I. I. T. Bombay
madhav@ee.iitb.ac.in

A. Gupta
NVidia Corp.
AGupta@nvidia.com

S. Chakraborty
Dept. of Comp. Sc. & Engineering
I. I. T. Bombay
supratik@cse.iitb.ac.in

Abstract

As the feature size offered by VLSI technology shrinks, circuit performance as well as circuit complexity increases. This puts considerable pressure on the synchronous design methodology, mainly due to the difficulty of routing a low skew high frequency clock signal across a large die. On the other hand, the synchronous design methodology offers the benefits of a mature design flow and a comprehensive set of design tools. In this paper, we present an approach towards the elimination of the global clock signal in a synchronous design. We present a novel partitioning strategy and the design of a distributed asynchronous controller for this purpose. The transformed circuit can have performance comparable or possibly superior to the original synchronous circuit (provided clock could be distributed in the first place). The technique is demonstrated by a pilot design in a .18 micron TSMC process, and is a good candidate for a clock-less design methodology built around the principle of desychronization.

1. Introduction

For the past fifty years, complex digital systems have been built using synchronous methodologies. This has been accompanied by the development and improvement of appropriate design and verification tools. However, every synchronous design has a hidden cost: distributing a low-skew clock signal to all parts of the chip. The difficulty of this problem increases as the complexity of the system increases, frequency of clock increases, and properties of the distribution network become difficult to control. The design of a complex VLSI circuit in a sub 100nm technology will possibly involve the distribution of a multi-gigahertz clock signal across a circuit with over a hundred million gates. Numerous algorithms have been proposed in [4] [5] [6] for solving this problem. However, they are reasonably complex, implementation is cumbersome, a significant area and power penalty is incurred, and the circuit is prone to malfunctioning if delays along the clock tree are not well matched. To circumvent this difficult design problem, we would like to explore the possibility of working without a global clock, thus venturing into the realm of asynchronous circuits. Other designers have also been investigating such possibilities [2] and have shown certain promising benefits.

A number of approaches are being examined in the asynchronous regime, like asynchronous design from specifications [9] [8]; using asynchronous components in an otherwise synchronous design e.g. RAPPID by Intel [10]; Globally Asynchronous Locally Synchronous [7] methods and modification of synchronous designs e.g. Micropipelines [1], GASP FIFOs [11]. We have adopted the final approach mentioned. This paper demonstrates the benefits possible by following our approach.

- Speed: There is no substantial degradation in performance relative to the equivalent synchronous system (assuming that the equivalent synchronous system can actually be physically implemented). In fact, it is possible that the derived asynchronous system has better performance, and this improvement is achieved through a novel pipelining strategy for handling global signals.
- Power and Area: There will be an overhead due to additional elements we will introduce into the system, but this overhead is predictable and quantifiable. The penalty incurred while designing a clock distribution network is eliminated.
- Ease of Design : We demonstrate that the proposed scheme is almost as easy as the design and verification of a synchronous system, and it also eliminates the clock distribution network design problem.

In this paper, we introduce our approach towards building large digital systems asynchronously, and attempt to demonstrate the properties of this approach that were claimed above. In Section 2 we present the essential idea behind our approach of going from the synchronous to the asynchronous regime. The control circuitry required for the implementation is developed in Section 3. In Section 4 we present a case study, to estimate benefits and overheads of using our approach over the conventional synchronous methodology. In Section 5, we conclude by evaluating the scheme with respect to the problem we had set out to solve, comparing it with other schemes that can be used and exploring its potential towards building large digital systems on chips.

2. Moving from Synchronous to Asynchronous Regimes

This sections demonstrates a top-down approach of building a large digital system starting from a synchronous design.

The first step is to visualize the system as a black box interacting with the environment. It receives inputs and generates outputs through latches, as shown in Figure 1. In the synchronous case, inputs are available on a rising edge of clock, and the system outputs are latched on the next rising edge of clock. The clock period is adjusted so as to accommodate the processing delay between availability of inputs and latching of outputs.

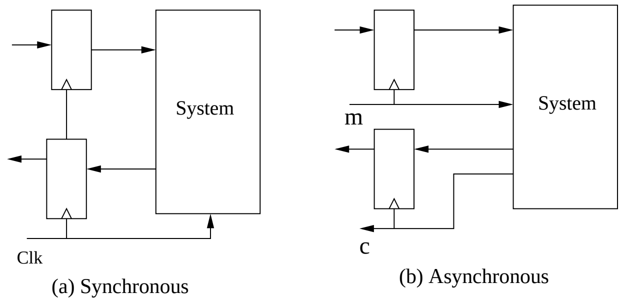

Fig. 1. Digital Systems as seen from the Environment

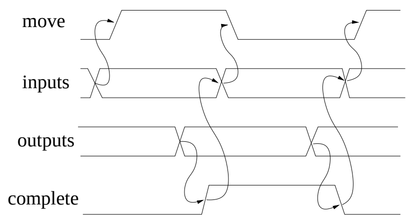

Fig. 2. Bundled Data Convention

In the asynchronous regime, when inputs are available, the environment asserts a move signal 'm'. When the system outputs are available, the system generates a complete signal 'c', enabling the environment to latch outputs and supply the next set of inputs. This protocol is the traditional bundled data convention [1] as elaborated in Figure 2, for 2-phase handshaking.

In the synchronous case, the clock signal is directly applied to all flip-flops in the design so that the system moves to the next state on the clock edge. The derived asynchronous system includes the synchronous system, except that flip-flops are driven not by a global clock, but by a control circuit. The derived system is functionally equivalent to the original system, i.e. state of the derived system after a move-complete sequence is identical to the state of the original system after the corresponding clock edge, given the same set of inputs.

The next step in our top-down approach is to model the processing module as a Finite State Machine [FSM] as shown in Figure 3. Here, we observe that the problem of low skew clock distribution has been converted to that of *designing a low-overhead, high speed control circuit to trigger the flip flops.*

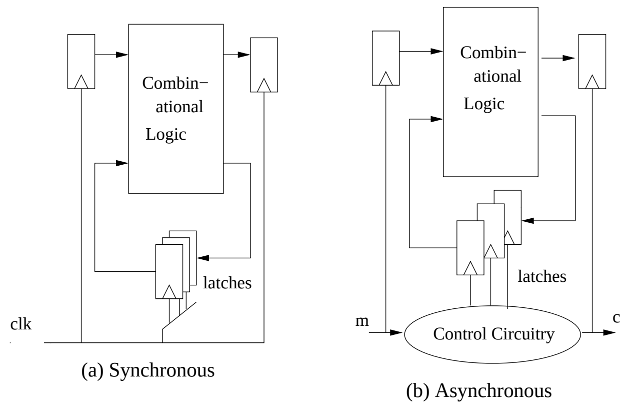

Fig. 3. The Processing Module Modeled as an FSM

Building the controller as a monolithic block would be tough and tedious to design, would have to distribute signals all over the chip and contribute substantially large control delays. Hence, we would like to have a distributed control mechanism. For this, the Processing Module must be partitioned. Further, in order to optimize speed, or rather throughput, a pipelining effect must be introduced.

We shall introduce a controller module at each inter-partition boundary. We would like to have an elastic pipeline, such that once a partition and all its neighbours have completed processing for the k_{th} cycle, it may receive inputs and start processing for its $k+1_{th}$ cycle. It need not wait for non-neighbouring partitions.

Now, the constraints on the partitioning form prerequisites for the design of the controller. We have assumed that the system may be partitioned as shown in Figure 4. Each partition interacts only with its neighbours. Hence, for the environment partition 'A' is its only neighbour. As a first cut, we have assumed a linear pipeline, where each partition has at most two neighbours. Here, all controllers are identical, making the system perfectly modular. Our methodology may also be used to design the controllers for complex pipelines where branches may fork out or join together at a partition boundary. For those cases, the controllers will be similar, but not identical.

Signals crossing partitions are classified as follows:

• Type I signals go from partition i to partition $i+1$, they cross at least one flip-flop in partition i.
• Type II signals go from partition $i+1$ to partition i, they cross at least one flip-flop in partition $i+1$.
• Type III signals go from partition i to partition $i+1$, they are purely combinational, i.e., they do not cross a flip-flop. These are also referred to as global signals and will be critical in determining system throughput.
• Type IV signals go from partition $i+1$ to partition i, they are purely combinational. These are not considered for the first cut design, however strategies for handling them will be discussed.

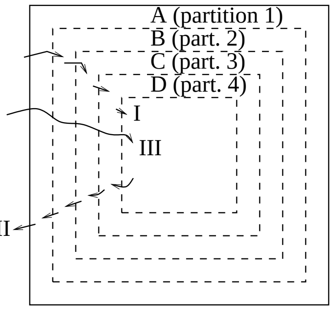

Fig. 4. Partitioning the Processing Module

These signals are illustrated in Figure 4 and Figure 5 respectively.

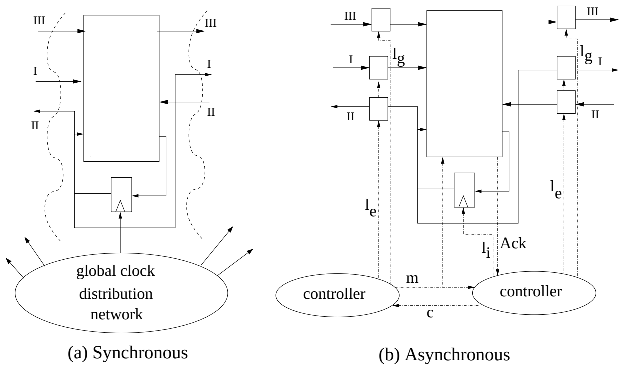

Fig. 5. Each Partition visualized as an FSM

Each partition may further be visualized as an FSM, as shown in Figure 5.

In the synchronous system, all partitions operate simultaneously and the system works correctly if the clock distribution meets set-up and hold time requirements of the flip-flops. This usually implies that clock skew is kept small. In the derived asynchronous system, partitions need not operate simultaneously. Further due to elasticity in the control path, the circuit functions correctly even if the delay of the partitions vary from cycle to cycle. However additional latches are required at the boundaries of the partitions and on the combinational path going in-wards to prevent races. These additional latches are also controlled by the control circuitry and will cause a power and chip area overhead.

In this section, we have taken a top-down look at digital systems with the aim of shifting from synchronous to asynchronous regimes. A superficial estimation of differences in performances was also made, wherever possible. We propose to implement the digital system as an asynchronous circuit using a distributed pipelined control mechanism. In the next section, we describe the design of this control circuitry in more detail.

3. The Controller

In this section, we develop and implement the control circuitry. In the previous section, we introduced a particular partitioning of the system. The constraints on the partitions are critical for designing the controller.

The sequence of operations for a single partition in the synchronous system takes place within successive rising edges on the "clock" signal and is as follows:

1. receive global signals
2. do processing
3. latch outputs

In the asynchronous case, the operation cycle enclosed within the move and complete events and is as follows:

1. receive global inputs ($l_g(n-1)$)
2. do processing (generate Ack)
3. latch internal flip-flops (l_i)
4. latch signals crossing partition boundaries (l_e)

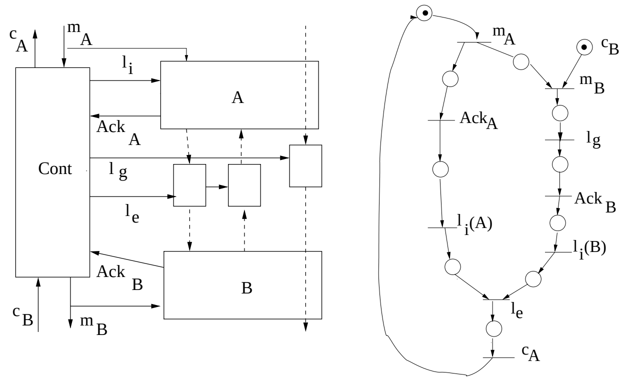

Fig. 6. Two Neighbouring Stages and their Controller

Figure 6 shows two neighbouring partitions 'A', 'B' and their controller. The controller's role in synchronizing the concurrent processing of the partitions is described using a Petri-net [3] [2].

The full system consists of a large number of cascaded controllers, one for each partition boundary. The complete Petri-net is shown in Figure 7.

Some significant observations may be made from Figure 7 regarding the working of the system :

- Correctness of the system can be proved formally [14].
- A slip is permitted across processing in various partitions. A maximum slip of half an operating cycle will be observed between neighbouring partitions. More specifically, there can be a maximum slip of 1 cycle between the p^{th} and $(p+2)^{th}$ partition. This can be seen by doing a token simulation in the Petri-net. This ensures that inputs to a particular partition during processing, from both neighbours are correct.
- The delay between move (mA) and complete (cA) with the environment, (and hence throughput) is independent of the number of partitions in the pipeline.

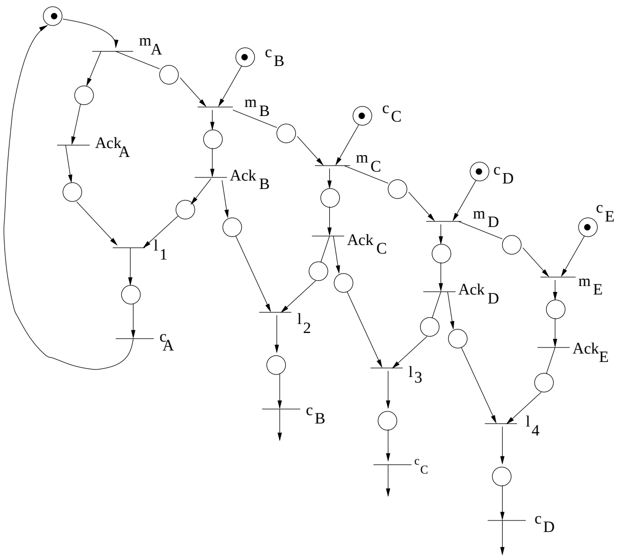

Fig. 7. Petri-net for the Cascaded Controllers

The working of the controllers and these observations may be understood more clearly by generating a set of valid traces for the Petri-net. One particular trace has been generated assuming a unit delay for all transitions and shown below (AA represents AckA and so on).

t	0	1	2	3	4	5	6	7	8	9	10
	mA	mB	mC	mD	mE	mA	mB	mC	mD	mE	mA
	AA	AB	AC	AD	AE	AA	AB	AC	AD	AE	AA
				l1	l2	l3	l4	cD	l1	l2	l3
					cA	cB	cC			cA	cB

Considering the case where Type IV signals exist in the partitioned system, a controller can be built using this methodology, however no slip would be permitted across those partitions. Hence merging partitions involving Type IV signals would give similar performance provided the clock distribution network across them does not become an issue.

We have used a 2-phase protocol [1] to implement the controller. We developed a double-edge triggered flip-flop which is compatible with this protocol. It is similar to a D-flip-flop, excepting that it samples the input on both rising and falling transitions of the clock. These flip-flops have been used for all extra latches introduced.

The partitions may themselves be operating on a 4-phase protocol. The controller will generate a local "clock" for these partitions as well. The falling edge will immediately follow the 'm' signal, and the rising edge will immediately follow the 'Ack' signal for the respective partition. Since we are working in the asynchronous regime, each partition must have a completion detection mechanism; e.g. Differential Cascode Voltage Switch Logic(DCVSL) [12] , Current Sensing Completion Detection(CSCD) [13], or a worst case constant delay. Presently, we have used a worst-case constant delay, which has been embedded in the controller itself.

The controller may now be implemented using three basic elements [1] :

1. delays : inverter chains or inverters with larger lengths for the transistors.
2. C-elements : used as an AND gate for events in a 2-phase protocol.
3. XOR gate : used as a 2-phase to 4-phase converter.

The gate level implementation of the controller is illustrated in Figure 8.

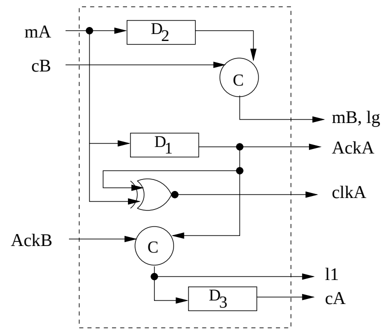

Fig. 8. Gate-level Implementation of the Controller

In this Section, we have developed the control circuitry for the asynchronous system proposed in the earlier section. Controllers may be built for systems having different partitioning constraints using the same methodology. In the next section, we present a case study performed to estimate the benefits and overheads of using our approach towards building digital systems.

4. Estimating the Benefits and Overheads: Case Study of a 16 Stage 8 bit Stack

In the previous sections, we have developed a scheme to build large digital systems. At various stages, a qualitative performance evaluation was done. To verify the claims, we simulated a sample circuit. We selected a "stack", as it is representative of the partitioning constraints introduced in Figure 4. We implemented a 16 stage stack, representing 16 partitions; with an 8-bit wide data path. It had 3 signals, push, pop and hold representing the global signals. The stack is, of course only a representative system; the aim being to get a handle on quantification of benefits and overheads. Here, the combinational logic in each partition is minuscule, when compared to the flip-flops and buffers. Thus, we shall present the benefits and overheads at a micro level. They will have to then be extrapolated for the complete circuit.

The stack was implemented in a .18 micron technology, using TSMC's 6 metal layers process parameters using Magic.

All observations stated and conclusions drawn are from SPICE simulations after extracting the netlist with capacitances from the layouts. In Figure 9, we show the layout of 8 stages. Here, only a 2-bit data path was laid out and stubs were placed to reduce simulation times.

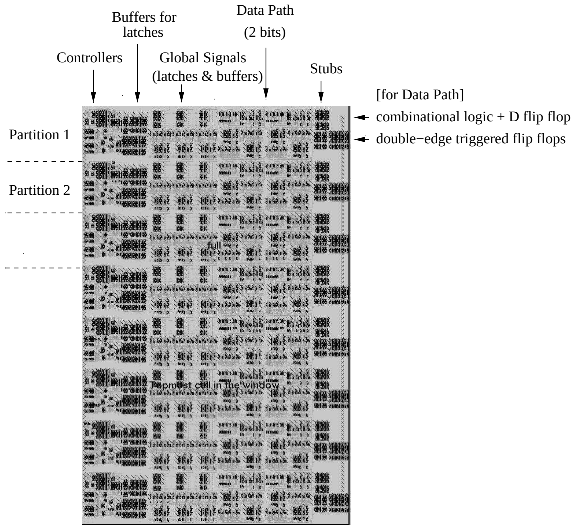

Fig. 9. The Stack Layout(2 bits + stubs matched to 6 bit load)

Any system may be demarcated into logic, flip flops and control (or clock) distribution network. These areas may then be analyzed separately.

4.1. Ease of Design

The overheads are design of the controller logic and double-edge triggered flip flops(DETFF), however these are small modules and can be replicated wherever required. Even matching all controller delays to the slowest partition will give similar throughputs. The elimination of the need to design a global clock distribution network is a huge benefit.

From a verification perspective, the data path is identical to the synchronous system. The control circuitry will have to be verified separately, however it would be much smaller than the data path segment.

4.2. Speed

The throughput achieved in the synchronous and asynchronous circuits were $450 * 10^6$ cycles/second (cycle time = 2.2nS) and $400 * 10^6$ cycles/second (cycle time = 2.5nS) respectively.

The path delays for the synchronous circuit were :

combinational logic (t_1)	: 300pS
buffers within partitions (t_2)	: 300pS
D flip flop (t_3)	: 200pS
global signal distribution (t_4)	: 90pS/partition

For a system with 'N' partitions, the cycle time would be $t_1 + t_2 + t_3 + N(t_4)$. For the particular case of the stack, where there is no logic along the global signals and a tree is used; the last term would be $(log_2 N)t_4$.

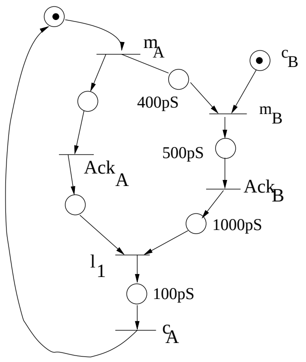

Fig. 10. Petri-net with typical Delays

Figure 10 shows the Petri-net for the asynchronous controller, with typical delays between the respective transitions. It is of course, an elastic pipeline; so delays would vary from cycle to cycle. From the timing analysis it can be seen that there is a potential for improving the throughput by matching controller delays more closely in the layout. There is also a huge potential for hiding the delay along the global signals behind the processing slip across partitions. We observed a marginal change of 5% in cycle time when simulating

8 partitions and 16 partitions. We expect the average throughput to be independent of the number of stages, which is not possible in the synchronous case.

4.3. Power Consumption and Silicon Area Required

There will be an overhead due to control logic and extra flip flops introduced. However, the power and area penalty incurred while matching path lengths in a clock distribution network will be eliminated. The power consumed and silicon area required for each standard cell is given as a table below:

	Power(pJ/cycle)	Area(λ^2)
DETFF	0.1	16800
buffers/DETFF	0.15	945
controller	1.1	39950
D flip flop (DFF)	0.15	10450
buffers/DFF	0.23	945

While estimating the overheads, one would have to account for one controller at each partition boundary and one double-edge triggered flip flop for each signal crossing a partition boundary, including the global signals.

In this section, we have presented our experiments on a sample system implemented synchronously and using our asynchronous approach. We have analyzed their respective performances and presented the benefits and overheads observed. The results conform to the qualitative analysis done in previous sections. We have followed this up, by giving some indications as to how digital system designers may use these figures to evaluate performance, benefits and overheads for the systems they may be attempting to build.

5. Conclusions

Present day chips are extremely fast, have small feature sizes and large functionalities. In such chips, distributing global signals, including the clock efficiently is extremely difficult. We had set out to eliminate this process altogether. We have proposed a generalized, robust, standardizable and scalable methodology to achieve this goal. We took a top-down approach towards modifying an existing synchronous circuit to get a partitioned asynchronous circuit with distributed, pipelined controllers. We then developed the required circuitry and verified it on a sample circuit.

We have demonstrated the methodology for a particular linear partitioning, however it is general enough to handle arbitrary topologies of the partitioned network.

In Section 4, we presented a case study to estimate the benefits and overheads of our scheme. The power consumption and silicon area overheads if any, will not be significant as long as partitions are not small. The throughputs for the synchronous circuit and the asynchronous circuit were similar. The scheme for handling global signals can potentially provide great benefits over a synchronous approach. Designing and verifying the system is almost as easy as a synchronous system, and eliminates the problem of designing a clock distribution network entirely. A modular procedure has been demonstrated, making design of large systems feasible.

The sample circuit chosen was of course a representative circuit, hence benefits and overheads were specified at a micro level. These will have to be extrapolated for the entire system. The formulae for extrapolation will depend on circuit functionality and architecture, and a method to do so was indicated. It is left to circuit designers to evaluate this approach in comparison with the traditional synchronous approach as to which is superior for a chip-level implementation.

References

[1] Ivan Sutherland "Micropipelines", *Communications of the ACM*, vol.32, no.6, June 1989, pp. 720–738

[2] S.Hauk "Asynchronous Design Methodologies:An Overview", *Proc. of the IEEE*, vol.83, no.1, Jan. 1995, pp. 69–93

[3] T. Murata "Petri nets:Properties, analysis and applications" *Proc. of the IEEE*, vol.77, no.4, Apr. 1989, pp. 541–580

[4] J. Cong and C. K. Koh "Minimum-Cost Bounded-Skew Clock Routing", *Proc. IEEE Intl. Symp. on Circuits and Systems*, vol.1, Apr. 1995, pp. 215–218

[5] Q. Zhu, W. W. M. Dai and J. G. Xi "Optimal Sizing of High-Speed Clock Networks Based on Distributed RC and Lossy Transmission Line Models", *Proc. Intl. Conf. on Computer-Aided Design*, 1993, pp. 628–633

[6] J. H. Huang, A. B. Kahng and C. W. A. Tsao "On the Bounded-skew Clock and Steiner Routing Problems", *Proc. ACM/IEEE Design Automation Conf.*, 1995, pp. 508–513

[7] T. Meincke, A. Hemani, S. Kumar, P. Ellervee, J. Berg, D. Liqvist, H. Tenhunen and A. Postula "Evaluating benefits of Globally Asynchronous Locally Synchronous VLSI architecture", *16th Norchip*, Nov. 1998, pp. 50–57

[8] K. Y. Yun and D. L. Dill, "Automatic Synthesis of Extended Burst-Mode Circuits: Part I & II", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol.18, no.2, Feb. 1999, pp. 101–132

[9] T. H. Y. Meng, R. W. Brodersen and D. G. Messerschmitt "Automatic Synthesis of Asynchronous Circuits from High-Level Specifications", *IEEE Trans. on Computer-Aided-Design*, vol.8, no.11, Nov. 1989, pp. 1185–1205

[10] S. Rotem, K. Stevens, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kol, C. Dike M. Roncken and B. Agapiev "RAPPID: An Asynchronous Instruction Length Decoder", *Proc. Intl. Symp. on Advanced Research in Asynchronous Circuits and Systems*, 1999, pp. 60–70

[11] I. Sutherland and S. Fairbanks "GasP: A minimal FIFO control', *Proc. Intl. Symp. on Advanced Research in Asynchronous Circuits and Systems*, 2001, pp. 46–53

[12] C.H.Erdelyi, W.R.Griffin and R.D.Kilmoyer "Cascode Voltage Switch Logic Design", VLSI Design, Oct. 1984, pp. 78–86

[13] M. E. Dean, D. L.Dill and Horowitz "Self-Timed Logic Using Current-Sensing Completion Detection (CSCD)" *Journal of Signal Processing*, vol. 7, 1994. pp. 17–31

[14] G. Hazari "Asynchronous Circuit Design: A Novel Technique Towards Eliminating the Global Clock in VLSI Circuits" *Dual Degree Dissertation* I. I. T. Bombay, July 2002.