San Jose State University
SJSU ScholarWorks

Master's Theses | Master's Theses and Graduate Research

2011

Low Power Digital Design using Asynchronous Logic

Sathish Vimalraj Antony Jayasekar
San Jose State University

Follow this and additional works at: http://scholarworks.sjsu.edu/etd_theses

Recommended Citation
Antony Jayasekar, Sathish Vimalraj, "Low Power Digital Design using Asynchronous Logic" (2011). *Master's Theses.* Paper 3909.

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

LOW POWER DIGITAL DESIGN USING ASYNCHRONOUS LOGIC

A Thesis

Presented to

The Faculty of the Department of Electrical Engineering

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Sathish Vimalraj Antony Jayasekar

May 2011

© 2011

Sathish Vimalraj Antony Jayasekar

ALL RIGHTS RESERVED

The Designated Thesis Committee Approves the Thesis Titled

LOW POWER DIGITAL DESIGN USING ASYNCHRONOUS LOGIC

by

Sathish Vimalraj Antony Jayasekar

APPROVED FOR THE DEPARTMENT OF ELECTRICAL ENGINEERING

SAN JOSÉ STATE UNIVERSITY

May 2011

Dr. Thuy T. Le	Department of Electrical Engineering
Professor Morris Jones	Department of Electrical Engineering
Dr. Chang Choo	Department of Electrical Engineering

ABSTRACT

LOW POWER DIGITAL DESIGN USING
ASYNCHRONOUS LOGIC

by Sathish Vimalraj Antony Jayasekar

This thesis summarizes research undertaken at San José State University between January 2009 and May 2011, which introduces a new method of achieving low power by reducing the dependency of the clock signal in the design. A clock signal consumes power even when the circuit is idle, but asynchronous circuits by default move into the idle state and involve no transition in the circuit during that state. In addition, in an active system, only the subsystem that is in use dissipates power. This work mainly focused on obtaining low power by implementing asynchronous logic. The work also studied the measure of power consumption using asynchronous logic by designing a simple Display Controller. The Display Controller was designed using Verilog HDL and synthesized using Synopsys Design Compiler. The work also studied the trade-offs in power, area, and design complexity in asynchronous design.

The power consumed by the synchronous and asynchronous display controllers was measured, and the asynchronous design consumed about 17% less power than its synchronous counterpart. The area of the asynchronous design was twice that of the synchronous one. Power can be reduced by reducing the dependency of the clock signal in the design by choosing asynchronous logic.

ACKNOWLEDGEMENTS

I am deeply indebted to Prof. Morris Jones and Dr. Thuy T. Le of the electrical engineering department at San José State University, for lending their valuable time for the success of this research.

This work would not have been possible without the support of my parents, Mr. Antony Jayasekar and Mrs. Susai Annamary, and my friend Mr. Ashwanth Sukumar and others, for their moral and financial support throughout this tough period and for being with me until the completion of this thesis.

Table of Contents

1. Introduction.....1
1.1 Need for Low Power Design.....1
1.2 Power Consumption in Digital CMOS Circuits.....3
1.2.1 Capacitive switching power.....4
1.2.2 Short-circuit power.....5
1.2.3 Leakage power.....6
1.2.4 Static power.....7
1.3 Literature Survey.....7
1.4 An Introduction to Asynchronous Design.....8
1.5 Globally Asynchronous Locally Synchronous (GALS) Design.....10
1.6 Overview of the Thesis.....12
2. Synchronous Display Controller.....14
2.1 Functioning.....14
2.2 Registers.....16
2.2.1 Horizontal Display End Register.....17
2.2.2 Start Horizontal Blanking Register.....17
2.2.3 End Horizontal Blanking Register.....17
2.2.4 Horizontal Total Register.....18
2.2.5 Vertical Display End Register.....18
2.2.6 Start Vertical Blanking Register.....18
2.2.7 End Vertical Blanking Register.....19
2.2.8 Vertical Total Register.....19
2.3 Design.....19
2.3.1 Counter.....21
2.3.2 Horizontal Timing.....22
2.3.3 Vertical Timing.....25
2.3.4 Display Controller.....27
3. Asynchronous Display Controller.....29
3.1 GALS Methodology.....29
3.2 Design of Asynchronous Display Controller.....31
3.2.1 Counter.....31
3.2.2 Horizontal Timing.....34
3.2.3 Vertical Timing.....36
3.2.4 Asynchronous Display Controller.....38
4. Power Analysis of Synchronous and Asynchronous Designs.....39
4.1 Introduction.....39
4.2 Power Analysis in Synchronous Display Controller.....39
4.3 Power Analysis in Asynchronous Display Controller.....41
4.4 Power Reduction Using RTL Clock Gating in Synchronous Circuits.....42
4.5 RTL Clock Gating in Asynchronous Display Controller.....43

4.6 Power Reduction Using Multiple Supply Voltages in Synchronous Circuits..... 44
4.7 Multiple Supply Voltages in Asynchronous Display Controller..... 45
4.8 Power Reduction Using Power Gating Technique in Synchronous Circuits..... 45
4.9 Power Gating in Asynchronous Display Controller..... 46
4.10 Benefits of Power Analysis Tools..... 47
4.11 Synopsys Power Compiler..... 49
4.11.1 RTL Power Optimization.....50
4.11.2 Gate-level Power Optimization.....50
4.12 Tool-based Power Analysis for Synchronous Display Controller..... 51
4.13 Tool-based Power Analysis for Asynchronous Display Controller..... 52
5. Trade-offs in Asynchronous Design.....53
5.1 Area Overhead..... 54
5.1.1 Synchronous Display Controller.....55
5.1.2 Asynchronous Display Controller.....55
5.2 Timing Overhead..... 56
5.3 Power Overhead..... 57
5.4 Design Complexity in Asynchronous Logic..... 58
5.4.1 Hazards.....59
5.4.2 Testing Asynchronous Circuits.....59
6. Conclusion and Future Work.....62
6.1 Conclusion..... 62
6.2 Future Work..... 64
REFERENCES.....66
APPENDIX A.....69
A.1 Synchronous Counter..... 69
A.2 Synchronous Horizontal Timing..... 70
A.3 Synchronous Vertical Timing..... 71
A.4 Synchronous Graphics Controller..... 72
A.5 Asynchronous Counter..... 73
A.6 Asynchronous Horizontal Timing..... 75
A.7 Asynchronous Vertical Timing..... 76
A.8 Asynchronous Graphics Controller..... 77
APPENDIX B.....79
B.1 Script for Synchronous Graphics Controller..... 79
B.2 Script for Asynchronous Graphics Controller..... 80

List of Figures

1. CMOS(Complementary Metal Oxide Semiconductor) inverter..... 4

2. The GALS(Globally Asynchronous Locally Synchronous) architecture …..... 12

3. Model of a display controller..... 15

4. Schematic of synchronous counter..... 22

5. Schematic of synchronous horizontal timing..... 24

6. Schematic of synchronous vertical timing..... 26

7. Schematic of synchronous display controller..... 28

8. GALS design methodology..... 30

9. Schematic of asynchronous counter..... 32

10. Schematic of asynchronous horizontal timing..... 35

11. Schematic of asynchronous vertical timing..... 37

12. Schematic of asynchronous display controller..... 38

13. Design flow with high-level power analysis..... 48

CHAPTER 1

1. Introduction

1.1 Need for Low Power Design

The need for low power design is motivated by several factors, such as the emergence of portable systems, thermal considerations, reliability issues, and, finally, environmental concerns. The evolution of portable or mobile communication devices such as laptops, cellular phones, video games, etc. is the most important factor driving the need for low power design. The demand for portable computers is increasing every year and is projected to increase in the future [1].

As consumers look for powerful yet low-power-consuming devices, there is a clear economic interest in the development of low power circuit design. The main reason behind the development of low power circuits is that many portable devices and their applications require low power dissipation and high throughput. Thus, low power design of digital integrated circuits is currently a rapidly developing field in electrical engineering.

The commercial success of portable or mobile devices depends significantly on their weight, cost, and battery life. In most cases, the cost and weight of batteries become a bottleneck that prevents the reduction of system cost and weight [1].

Moreover, for most portable systems, the IC (Integrated Circuit) components consume a significant portion of the total system power [2].

Portable devices have a strict demand for power consumption since they have limited battery capacity. Though new rechargeable batteries are available on the market, such as nickel-metal hydride (NiMH) batteries that have a higher energy capacity than the conventional nickel-cadmium (NiCd) batteries, such an increase in energy capacity is not expected in the near future [3]. The previously mentioned increase in energy capacity due to new battery technologies such as NiMH would still be insufficient, considering the increasing rate of applications in portable devices.

Low power design also plays a significant role in high-performance integrated circuits such as microprocessors and other high-speed digital circuits, which leads to circuit designs with high clock frequencies. Due to the increase in clock frequency, there is a proportional increase in power dissipation.

The power consumed by the integrated circuit is dissipated in the form of heat. This may lead to problems such as circuit degradation and operating failures. Component failure rates double for every 10ºC increase in operating temperature [1]. The power consumption in microprocessors is projected to grow linearly in proportion to their die size and clock frequency. Various cooling systems have been introduced to reduce the heat from power dissipation and keep the chip temperature at an admissible level. This in turn has increased the packaging cost, which results in large revenue reduction [2].

Moreover, large current levels on metal interconnections lead to electromigration,

which may cause electrical shorts between lines [4]. Along with electromigration, there are many reliability and signal integrity issues in integrated circuits due to excessive power consumption. Furthermore, of the total power consumed by office equipment, about 80% is consumed by computing equipment and mostly when that equipment is not in use [1].

Efficient low power design techniques are required to avoid these problems. Reducing a circuit's average power consumption typically improves the circuit's reliability. This leads to a reduction in cooling requirements, which in turn reduces the packaging and cooling costs. Thus, effective low power design methods are of supreme importance.

1.2 Power Consumption in Digital CMOS Circuits

Power consumption in digital circuits can be summarized as follows:

$$P_{tot} = P_{sw} + P_{sc} + P_{leak} + P_{stat} \tag{1.1}$$

where P_{sw} is the capacitive switching power,

P_{sc} is the short circuit power,

P_{leak} is the leakage power,

and P_{stat} is the static power.

1.2.1 Capacitive switching power

The capacitive switching power consumption is caused by the charging and discharging of parasitic capacitance in the circuit. Consider the inverter circuit below.

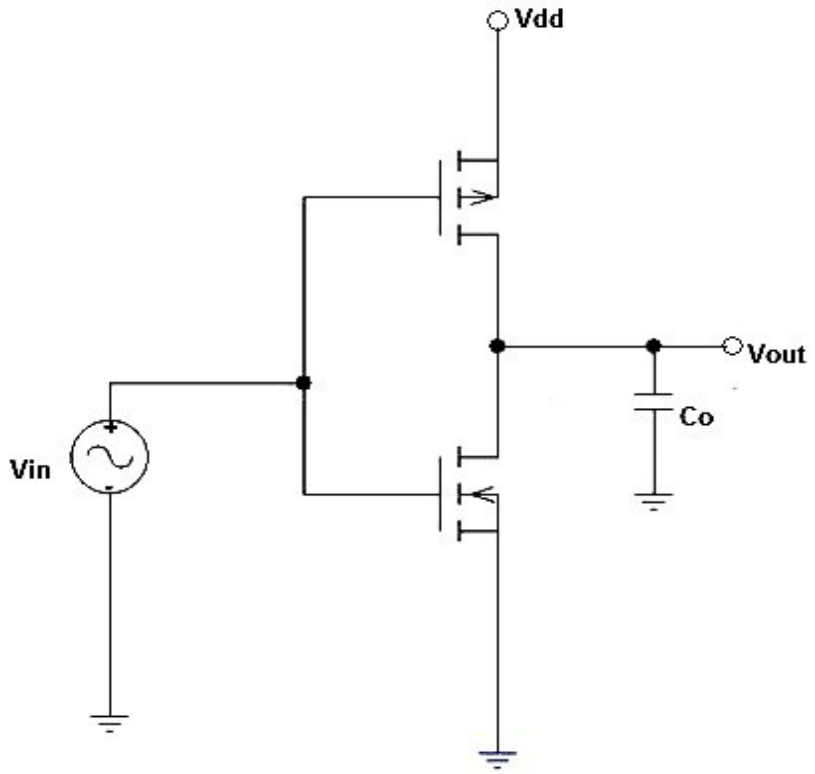

Figure 1. CMOS inverter

The load capacitor C_o represents the total capacitance associated with the NMOS and PMOS transistors, the internal capacitance of the wires, and the input capacitance of the driven gates. Whenever the input of the transistor undergoes falling transition, the PMOS transistor turns on, and the NMOS transistor turns off. This enables C_o to be charged to V_{in}. The energy drawn from the supply during this charging process is C_o

V_{in}^2, half of which is stored in the capacitor while the remaining half is dissipated in the PMOS transistor and the interconnect. Similarly, whenever the input undergoes a rising transition, the NMOS transistor in turned on, and the PMOS transistor is turned off. This discharges the capacitor C_o. During this discharging process, the $\frac{1}{2} C_o V_{in}^2$ energy that was stored in the output capacitor gets dissipated through the NMOS transistor and the interconnect. Based on the above discussion, the capacitive switching power dissipated by the CMOS inverter can be given by [5]:

$$P_{sw} = \frac{1}{2} C_o V_{in}^2 N f \qquad (1.2)$$

where f is the clock frequency

and N is the number of transitions per clock cycle.

Because the capacitive switching power accounts for a dominant part of the total power, most power analysis techniques focus on reducing this component of power consumption.

1.2.2 Short-circuit power

Short-circuit power is caused by direct supply-to-ground connections, created during signal transitions. Consider the CMOS inverter in Figure 1. Whenever there is a transition from 1 to 0 or from 0 to 1, both NMOS and PMOS transistors conduct for a considerable amount of time. This leads to short-circuit currents being drawn from the supply. The short-circuit power dissipation of the CMOS inverter can be given by the following formula [5]:

$$P_{sc} = K \ (V_{in}-2V_T)^3 \ \tau \ N \ f \qquad (1.3)$$

where K is the constant that depends on the transistor size and technology,

V_T is the threshold voltage,

τ is the input rise/fall time,

N is the average number of transitions in the circuit,

and f is the clock frequency.

Short-circuit power can be reduced by sizing the transistors appropriately. It can also be reduced by scaling the supply voltage and by reducing the switching activity at the gate outputs.

1.2.3 Leakage power

Leakage power (P_{leak}) is due to reverse biased diode current and sub-threshold leakage current. Reverse biased diode current is formed between the diffusion region and the substrate. Sub-threshold leakage current is due to transistors conducting some current even when they are idle. Leakage power is significant for devices that are mostly in an idle state [5].

1.2.4 Static power

Static power (P_{stat}) consumption is due to continuous conduction in the supply-to-ground path. These situations are undesirable and can be avoided by carefully designing the circuit [5].

Since capacitive switching power is the most important component of power consumption, the discussion in this thesis will be restricted to dynamic (switching) power consumption.

1.3 Literature Survey

Circuits become slower when the supply voltage is low and the threshold voltage is high. Power dissipation becomes greater when the supply voltage is high and the threshold voltage is low. Thus, a trade-off is required between circuit speed and power dissipation. Lowering the supply voltage and the threshold voltage enables high speed and low power operation [6,7]. This technique has a few disadvantages. When the supply voltage is low, speed is degraded during a fluctuation in threshold voltage. In addition, when the threshold voltage is low, the standby power dissipation is greater [8]. The high power dissipation is due to the sub-threshold leakage current at the low threshold voltage.

The sub-threshold leakage current results in static power consumption, which

accounts for more than 50% of the power used by modern ICs [9]. This can be reduced by increasing the threshold voltage and decreasing the supply voltage. However, both changes affect circuit speed. To enhance the speed, two supply voltages are used, one to speed up the significant parts of the circuit and another to lower power in non-significant parts of the circuit. To reduce power consumption further without performance loss, different transistors with different threshold voltages in different parts of the circuit are used.

By shutting down a leaky functional block until it is used, leakage current can be reduced significantly. This can be done by using sleep transistors to disable an entire block when it is not in use [9]. For systems that function for only a short period, this solution is very effective. Since this type of system works in some isolated locations monitoring some activities, power consumption is a key factor.

1.4 An Introduction to Asynchronous Design

Low power digital system design can be obtained at various levels, such as the process level, circuit level, architecture level, and algorithm level, by reducing the number of switching events for a given task. This thesis will concentrate on the algorithm level of the design, which can be applied to reduce power dissipation in digital integrated circuits.

Most of the digital integrated circuits designed and fabricated today are synchronous in nature. In synchronous circuits, all components share a common time,

defined by a clock signal distributed throughout the circuit [12]. In high-speed circuits, as the clock frequency increases, power consumption also increases gradually.

An effective method for reducing power consumption is reducing the dependency on the clock signal in the circuit. To achieve this, the digital system should be divided into smaller autonomous blocks. These blocks should not share a common time defined by a clock signal. This leads to the asynchronous design style.

Unlike the design of conventional devices, asynchronous design does not have a centralized clock to coordinate the progress of data. A pipeline controller logic triggers the next stage of the design when the current stage is complete. This ensures that a centralized clock is unnecessary. Components in the device can run at different speeds without waiting for the centralized clock [10]. In addition, the clock signal consumes a considerable amount of chip power and continues to run even when a system is idle. Asynchronous circuits have the advantage of going into an idle state by nature, and there will be no transitions in the circuit during the idle state. Thus, by going in for asynchronous logic, power is employed only for useful work. Another favorable circumstance that influences asynchronous design is that, even in an active system, only the subsystem required for computation will dissipate power [13]. Therefore, the power equivalent to that consumed by a clock signal is saved. Moreover, the supply voltage can be safely reduced, either statically or dynamically, to match the actual throughput to the desired computation rate, thereby saving power [11].

To meet timing requirements, synchronous design builds complex circuitry, to quicken rare, worst case conditions. This in turn consumes more power. Asynchronous

design can allow worst-case operations to proceed slowly and use the resources and power consumption in operations that occur frequently [11].

Asynchronous logic has begun to gain interest due to the observation that synchronous logic has started to reach its limits. As the number of transistors increase, global synchrony becomes difficult, and clock skew becomes a problem. By contrast, asynchronous logic generates local timing signals to handle the global synchrony and clock skew issues that emerge in synchronous logic [11].

Asynchronous design is not a new approach. Circuits have been designed using asynchronous logic for 20 years. Due to the inherent difficulties, asynchronous operation has been discontinued. However, recent development in methodologies have overcome those problems and allowed asynchronous techniques to emerge [11].

1.5 Globally Asynchronous Locally Synchronous (GALS) Design

In synchronous systems, the clock signal is used for a variety of purposes. The clock signal is global in nature. During the clock edge, the flip-flops are updated, and the new state ripples through the circuit to compute the next state. This provides a variety of structured design methods. The structured design of asynchronous circuits requires a timing discipline to replace the global clock. Simple request and acknowledge signaling can be used for this purpose. The subsystem on the transmitting side plays the active role and initiates the transition, whereas the subsystem on the receiving side waits and acknowledges. This is called handshaking.

The two most common handshake components used in data-paths are the handshake latch and the transferrer. The handshake latch functions like a register in the synchronous circuits. The transferrer forms the prime interface with the control part of the handshake circuits. In a handshake latch, the energy required for the write operation amounts for 2 or 4 transitions, and the read operation requires 4 transitions [14]. The transferrer does not require any energy for its operations.

Asynchronous design, while reducing power consumption, also increases chip area. This is due to the overhead of extra circuits for handshaking and increased routing area. Asynchronous design can make the circuit resilient to delay variations. The speed independent and delay-insensitive models can operate in the presence of delay variations, in gates and interconnects. These methods have immense design complexity and require a great deal of engineering time. This can be compromised by using the Globally Asynchronous Locally Synchronous (GALS) design style. As the name suggests, asynchronous handshaking will be used to link various synchronous domains. Figure 2 shows the GALS architecture.

By eliminating the global clock, the major source of power consumption is eliminated. In addition, synchronous blocks operate asynchronously with respect to one other, and the operating frequency of each synchronous clock can be modified according to its needs, thereby reducing the average frequency and overall power consumption.

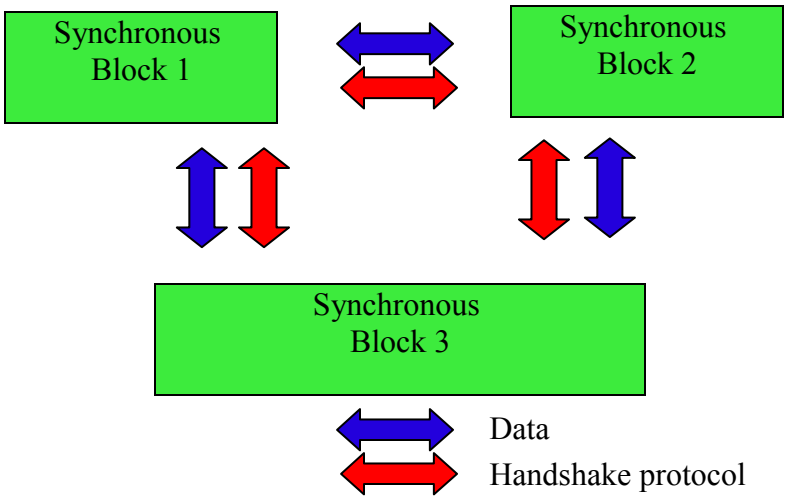

Figure 2. The GALS architecture

1.6 Overview of the Thesis

The focus of the thesis is to analyze the potential of asynchronous design for low power consumption. To begin, a synchronous display controller was designed and analyzed for power consumption. For many synchronous applications, the generation and distribution of the clock signal account, directly or indirectly, for more than half the power dissipation [14]. The main building block for the display controller is a counter that is synchronous in type. Much of this wasted clock power can be saved by opting for asynchronous design.

Later, a display controller that is mostly asynchronous in type was designed using the GALS design style and analyzed for power consumption. The display controller was

designed using Verilog HDL and simulated in VCS. The design was synthesized using Synopsys Design Compiler, and power was measured using Synopsys Power Compiler.

We develop a qualitative understanding of the hardware-level design of the synchronous display controller in Chapter 2. This enables us to better understand the hardware-level design of the asynchronous display controller using the GALS design style, discussed in Chapter 3. A good understanding of the hardware-level design of synchronous and asynchronous display controllers is necessary for analyzing some important performance parameters. In Chapter 4, power, an important performance parameter in digital CMOS circuits, is discussed in detail. The power analysis is based theoretically on capacitive switching power and based on the results from Synopsys Power Compiler. The other performance parameters such as the speed and area of the design are discussed in detail in Chapter 5. These parameters are very important for making first-level design decisions. Finally, we will discuss how this work can be extended and developed to achieve better performance characteristics.

CHAPTER 2

2. Synchronous Display Controller

2.1 Functioning

A display controller is a unit that reads video signals from the RAM attached to the unit and outputs the video signals to the display through a ROM. The display controller is the main component in a video signal generator. A display controller also is responsible for generating timing signals, such as horizontal and vertical sync signals, display end signals, etc.

Figure 3 shows the model of a display controller. Horizontal timings are based on a unit called character clocks. Each character is about 8 or 9 pixels. **Horizontal Display End** marks the end of the display for the horizontal line. It shows the last horizontal character read from memory. **Horizontal Blanking Start** marks the beginning of the blank area. Beyond this point, there is no display. **Horizontal Blanking End** marks the end of the blank area. As soon as the value of the character count is equal to the value of the **Horizontal Total**, the **Horizontal Retrace period** starts. **Horizontal Retrace Start** and **Horizontal Retrace End** mark the beginning and end of the retrace period, respectively.

Vertical timings are similar to horizontal timings, with the exception that these registers operate on scan lines instead of character clocks. The vertical line increments

after each horizontal line. **Vertical Retrace End** works the same way as **Horizontal Retrace End**, though they have different sizes and vertical retrace requires more time than horizontal retrace.

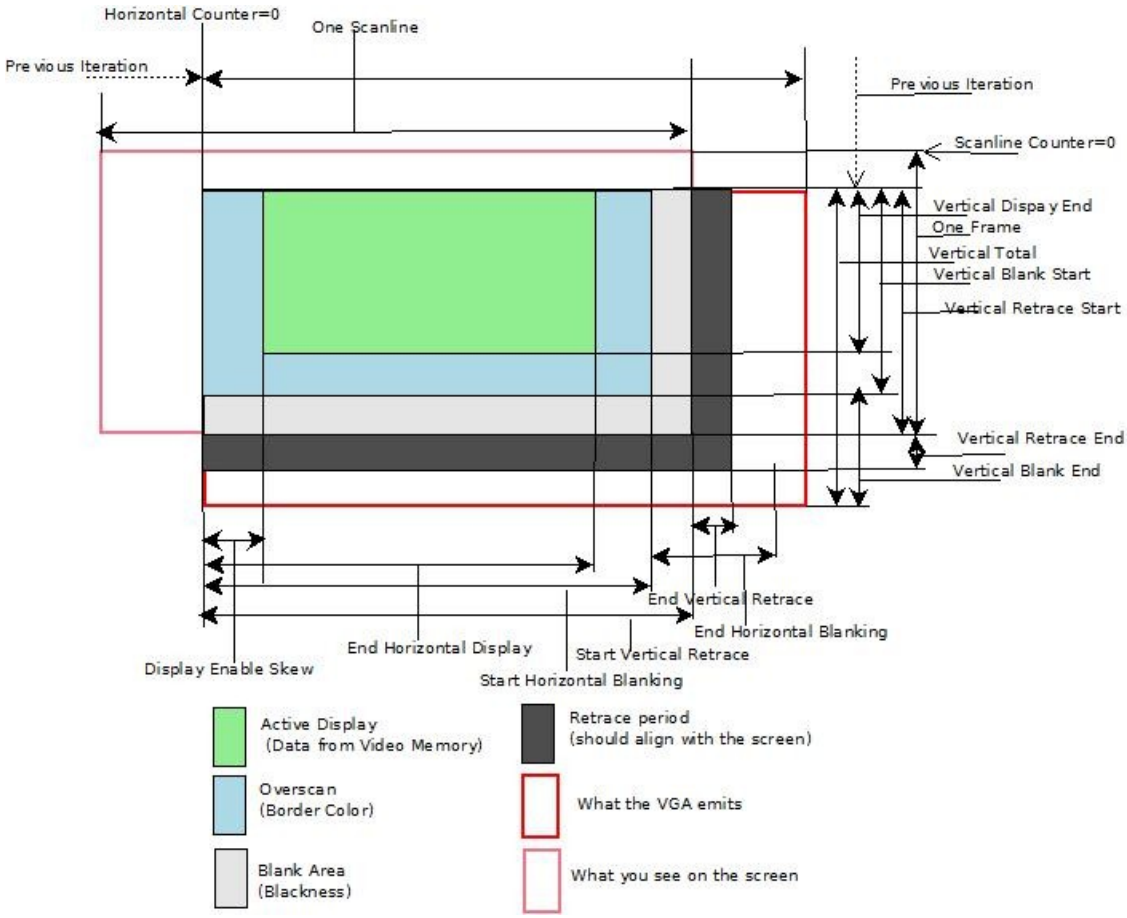

Figure 3. Model of a display controller [15]

This display relies mainly on the memory buffer that contains the full frame of data in the display memory. The display memory is in the form of shift registers. The data is read out of the shift registers synchronously with the scanning electron beam. Thus, when the first data is read from the shift register, the electron beam will be at the

top left corner. As the electron beam scans through the first horizontal line, the relevant data is read from the shift register. Once the horizontal total value is reached for the first line, the electron beam retraces its path and positions itself at the start of the second line. During this retrace period, no information is read from the shift register. As soon as the electron beam is ready for the second line, it starts reading data from the shift register and outputs it. This process continues until the last scan line is read and output. At this point, the vertical retrace occurs, and the electron beam returns to the top left corner of the screen [16].

Even though the display memory appears to be parallel, it has to be converted into a serial data stream in order to interface with the electron beam in the monitor [16].

2.2 Registers

The display controller registers form the largest register group of the EGA and VGA [16]. These registers control the display timing and synchronization functions. Not all registers were used to design the display controller for this thesis. Below are descriptions of some of the registers used in this design.

2.2.1 Horizontal Display End Register

This register stores the value of the number of characters in the display area. This value marks the end of the active display area. After the internal counter reaches the value stored in this register, the blanking period starts.

2.2.2 Start Horizontal Blanking Register

This register stores the value at which the horizontal blanking signal becomes active. It is actually one count more than the horizontal display end register. When the internal counter reaches this value, the horizontal blanking signal is generated.

2.2.3 End Horizontal Blanking Register

This register marks the width of the blanking period. It stores the value at which the horizontal blanking period becomes inactive. During the horizontal blanking period, the address for the next scan line is stored in the memory.

2.2.4 Horizontal Total Register

This register stores the value of the number of characters in the horizontal scan line plus the horizontal retrace period. This value marks the end of the horizontal scan line. The internal counter resets after this value is reached. This marks the beginning of the next scan line. Once the internal counter reaches this value, the horizontal retrace signal is generated. As soon as all the internal counters are reset, the horizontal retrace period ends. This procedure repeats for every horizontal scan line.

2.2.5 Vertical Display End Register

This register stores the value of the number of scan lines in the active display area. Once the internal counter reaches this value, vertical blanking starts. This register also determines the last scan line at the bottom of the screen. This value also marks the beginning of the vertical blanking period.

2.2.6 Start Vertical Blanking Register

This register stores the value at which the vertical blanking signal starts. Vertical blanking prevents the beam from writing in the display area during the retrace. When

the internal counter reaches this value, the blanking signal is generated.

2.2.7 End Vertical Blanking Register

This register stores the value at which the vertical blanking signal becomes inactive. This marks the width of the blanking period or the time required to keep the display area blank before the beam moves to the top of the screen.

2.2.8 Vertical Total Register

This register determines the number of scan lines on the monitor plus the vertical retrace period. This value also marks the end of the frame. When the internal counter reaches this value, the vertical retrace period begins. This retrace period allows the electron beam to move back to its initial position at the top left corner. During this period, all the internal counters are reset to their initial value. Once the internal counters are reset, the retrace signal becomes inactive.

2.3 Design

The display screen is divided into numerous rows and columns, with the

intersection point named “dot.” To form a single character, a series of dots have to be highlighted on the screen on successive scan lines. The dot information is serially inputted in the display through the DAC. Each dot’s information consists of 8-bit data, and many dots are required to represent an alphanumeric character. Finally, every line will have more than one character. Therefore, a great deal of memory is required to display information on the screen.

To display alphanumeric data on the screen, the dot pattern for the first scan line must be given sequentially. Once the first scan line is completed, the dot pattern for the second scan line must be fed sequentially to the display. This has to be repeated until all the scan lines in that particular character row have been completed.

The dot pattern of a character for a particular scan line is read from the ROM with the help of row-select inputs. As the dot pattern for each row is read from the ROM, the dot pattern is loaded into the shift register to be sent by the sequencer serially to the DAC. Once the dot pattern for the last scan line is fed into the shift register, the row-select is reset to its initial value for the new character sequence.

Moreover, the sequencer has to stop sending the dot pattern beyond the display area marked by the horizontal display end resister. Beyond this point, the DAC will move into the blanking state, no matter what the sequencer sends.

Thus, a synchronous display controller is designed using counters to send the dot pattern sequentially into the display, and registers to mark the limits, such as the display end and the blanking period. The counters and registers discussed in the previous section are the basic building blocks for the display controller.

2.3.1 Counter

Figure 4 shows the schematic of a synchronous counter. The global synchrony for the counter is brought about by the clock. The counter also has an incrementer block and a comparator block. The counter begins with the initial value stored in the flip-flop. The incrementer increments the counter by a factor of 1. The comparator compares the incremented value with the limit stored in the display controller registers. Four such counters are required to design the horizontal timing for the display controller.

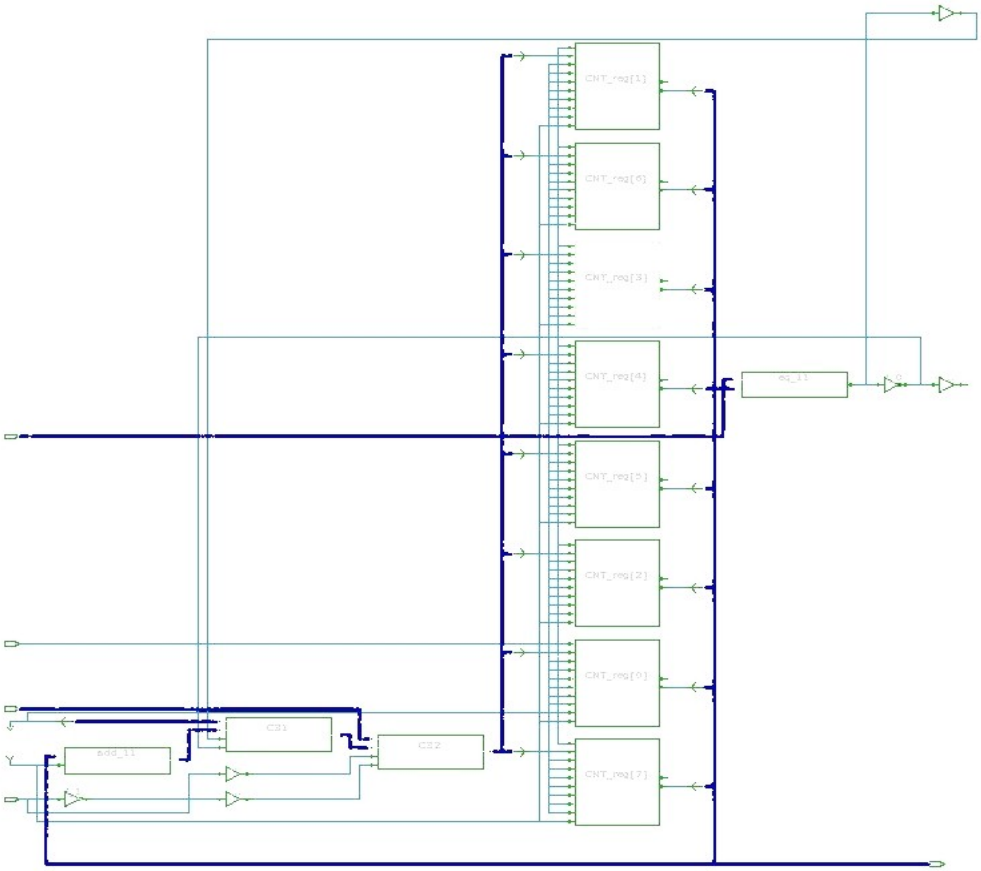

Figure 4. Schematic of synchronous counter

2.3.2 Horizontal Timing

Figure 5 shows the schematic of the horizontal timing of a display controller. As shown in the schematic, four counters, one each for the display end, blanking start, blanking end, and horizontal total, are connected in parallel. Their limit values are given

by their respective registers.

The counter begins with an initial value of zero. As and when each dot information or pixel is displayed, the counter is incremented. This process is repeated until the pixel value is stored in the display end register. Once the blanking start value is reached, a blanking signal is generated, indicating to the DAC to stop transmitting valid data. The blanking signal becomes inactive after the blanking end pixel value. The retrace signal is generated after the horizontal total value is reached, indicating to the electron beam to move to the next scan line.

The counter increments at every clock edge, for every pixel value generated as the electron beam moves to its next valid position. As the limit is reached, the counter resets to its initial value. Beyond the display area, control signals, such as blanking signals and retrace signals, control the functioning of the display controller.

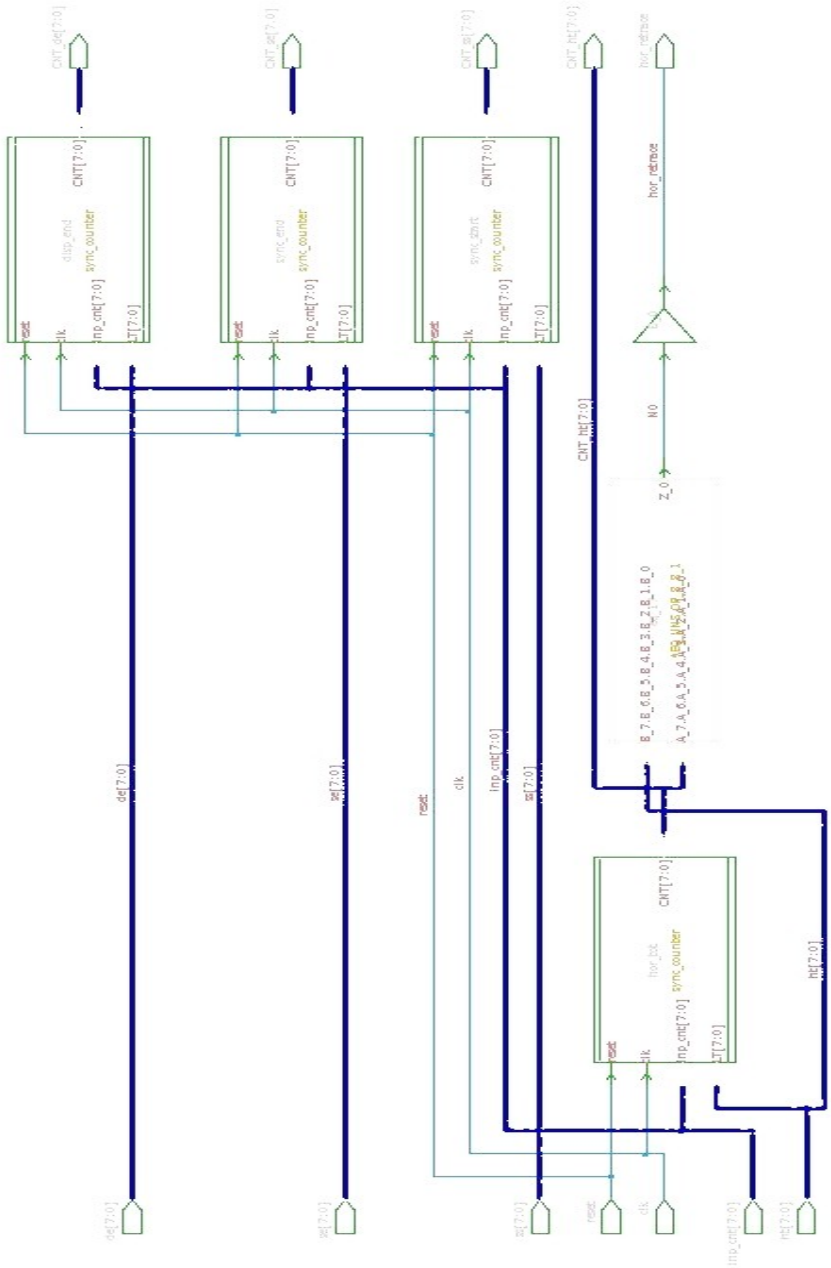

Figure 5. Schematic of synchronous horizontal timing

2.3.3 Vertical Timing

Figure 6 shows the schematic of the vertical timing of the display controller. The vertical timing functions the same way as the horizontal timing with the exception that the counters in horizontal timing increment every clock cycle, whereas those in vertical timing increment every scan line; i.e., the counters in vertical timing increment after completion of each horizontal line. Since the horizontal retrace signal is generated at the end of the horizontal line, the signal is used to denote the completion of a horizontal line. Thus, the counters in vertical timing are incremented at the positive edge of the horizontal retrace signal.

In addition, the register values in horizontal timing and vertical timing may vary depending on the display dimensions. The vertical retrace signal marks the end of the current frame. Once the vertical retrace signal goes high, all the counters are reset, and the electron beam moves to the top left corner of the display for the next frame.

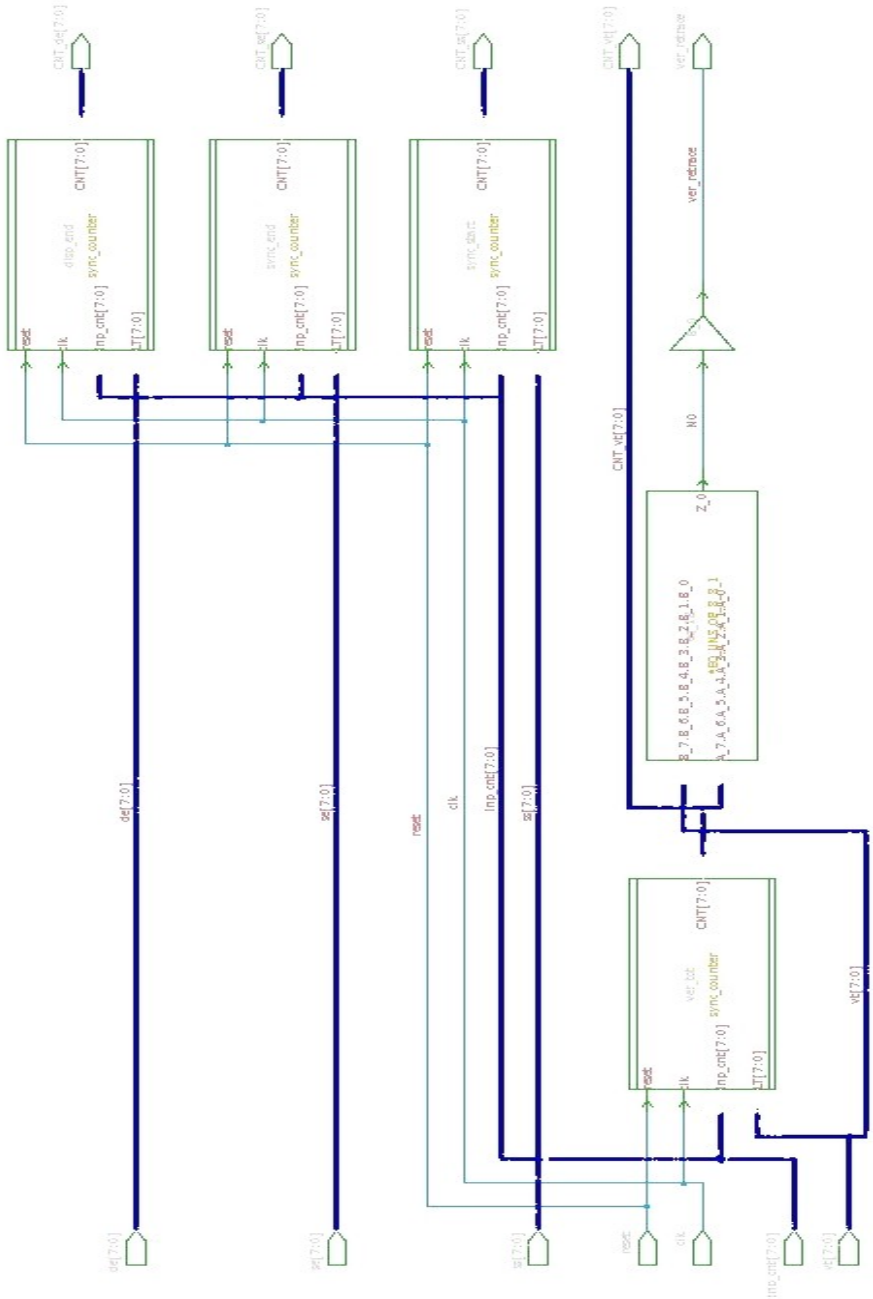

Figure 6. Schematic of synchronous vertical timing

2.3.4 Display Controller

Figure 7 shows the schematic of a synchronous display controller. The global synchrony is maintained by the clock signal. The clock signal synchronizes all the counters in the horizontal timing, and the horizontal retrace signal synchronizes all the counters in the vertical timing. The register values define the timing for activating and deactivating the control signals.

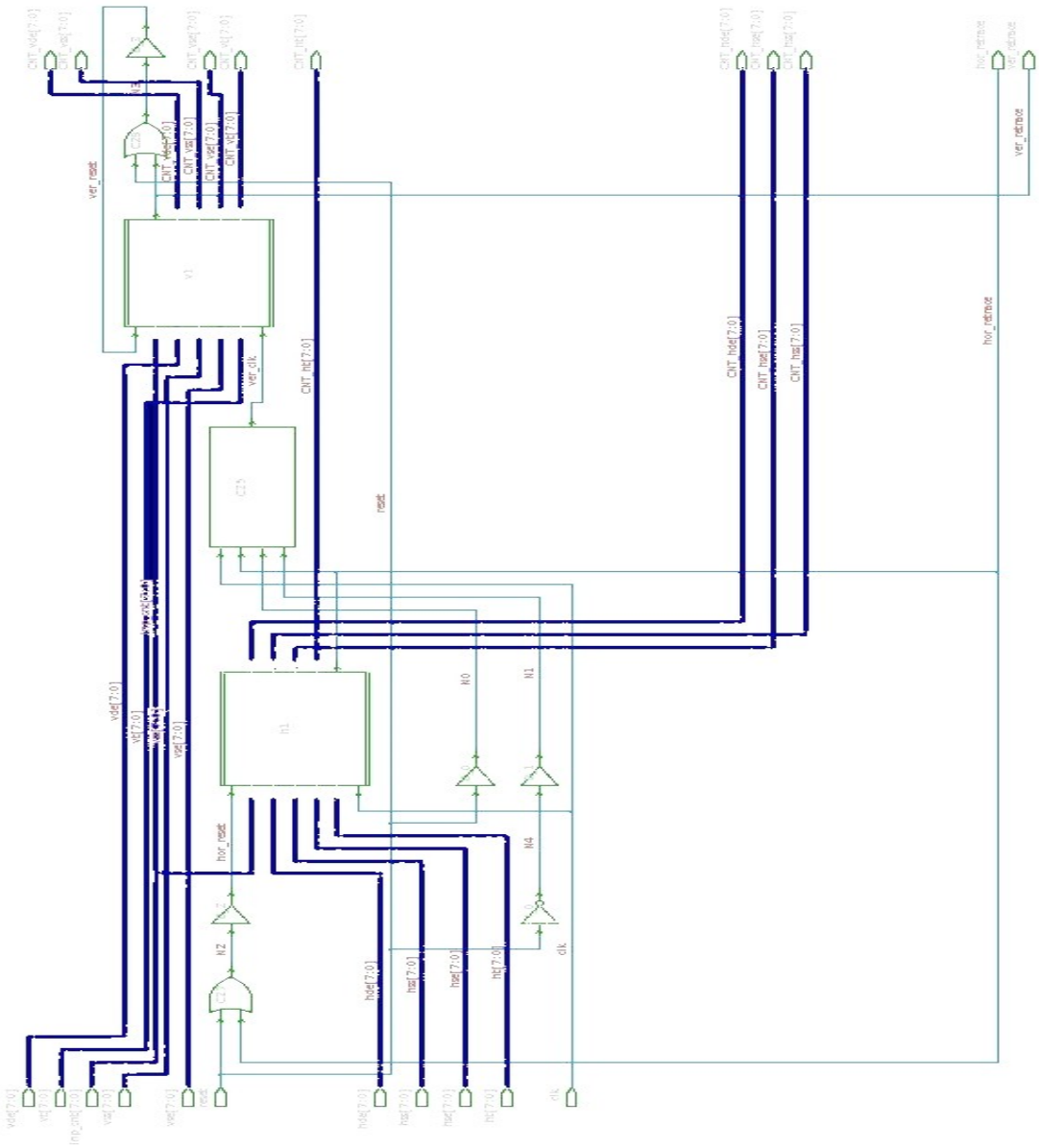

Figure 7. Schematic of synchronous display controller

CHAPTER 3

3. Asynchronous Display Controller

Most digital circuits currently designed are synchronous in nature. All synchronous designs assume a common timing signal that is distributed throughout the circuit. This assumption ignores problems such as hazards and the dynamic state of the circuit. A system designed without this assumption is expected to produce better results. Asynchronous design eliminates this assumption of common and discrete time and has several benefits, such as low power consumption and avoidance of global timing issues, etc. [17].

Asynchronous design is always needed, even though synchronous designs are now very prevalent and commonly used. Asynchronous logic can be used to interface one synchronous system with another. Several methodologies were developed to simplify the asynchronous design logic. One such method is the Globally Asynchronous Locally Synchronous (GALS) design.

3.1 GALS Methodology

The GALS design methodology is a slightly modified version of the synchronous design style. The GALS design extends the synchronous design method in two aspects: the partitioning of the synchronous system into smaller synchronous blocks and the

establishing of asynchronous communication between the synchronous blocks.

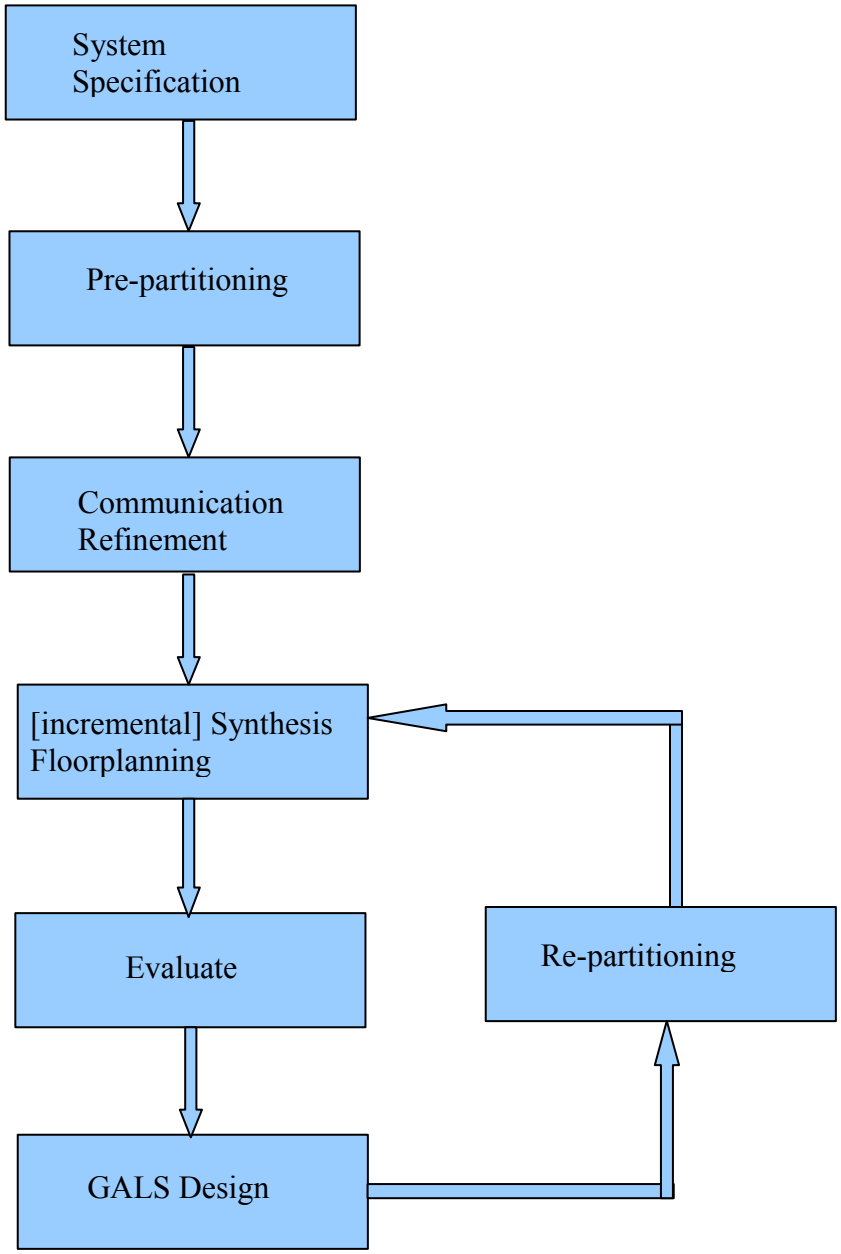

Figure 8. GALS design methodology [18]

Figure 8 gives a clear picture of the GALS design methodology. Starting with the initial system specification, the system could be partitioned into synchronous blocks of optimal size and number. At this stage, called the pre-partition stage, the system is partitioned at the initial planning stage. Next, in the communication refinement stage, the interfacing between the synchronous blocks is decided. This can be chosen from four possible communication modes: send and forget, strobe, handshake, or FIFO. The synchronous blocks are characterized by parameters such as the clock period and the number of transitions of I/O signals that can be done by static analysis [18]. Finally, the synchronous blocks are synthesized and repartitioned (if necessary) for expected results.

3.2 Design of Asynchronous Display Controller

The asynchronous display controller was designed using the GALS methodology by partitioning the synchronous display controller discussed in the previous chapter into various synchronous blocks. Like the synchronous display controller, the asynchronous version is designed using counters to keep track of the dot pattern sent sequentially into the display and registers to mark the limits such as the display end and the blanking period.

3.2.1 Counter

Figure 9 shows the schematic of a counter designed using GALS design

methodology. The synchronous counter from the previous chapter is partitioned into three synchronous blocks:

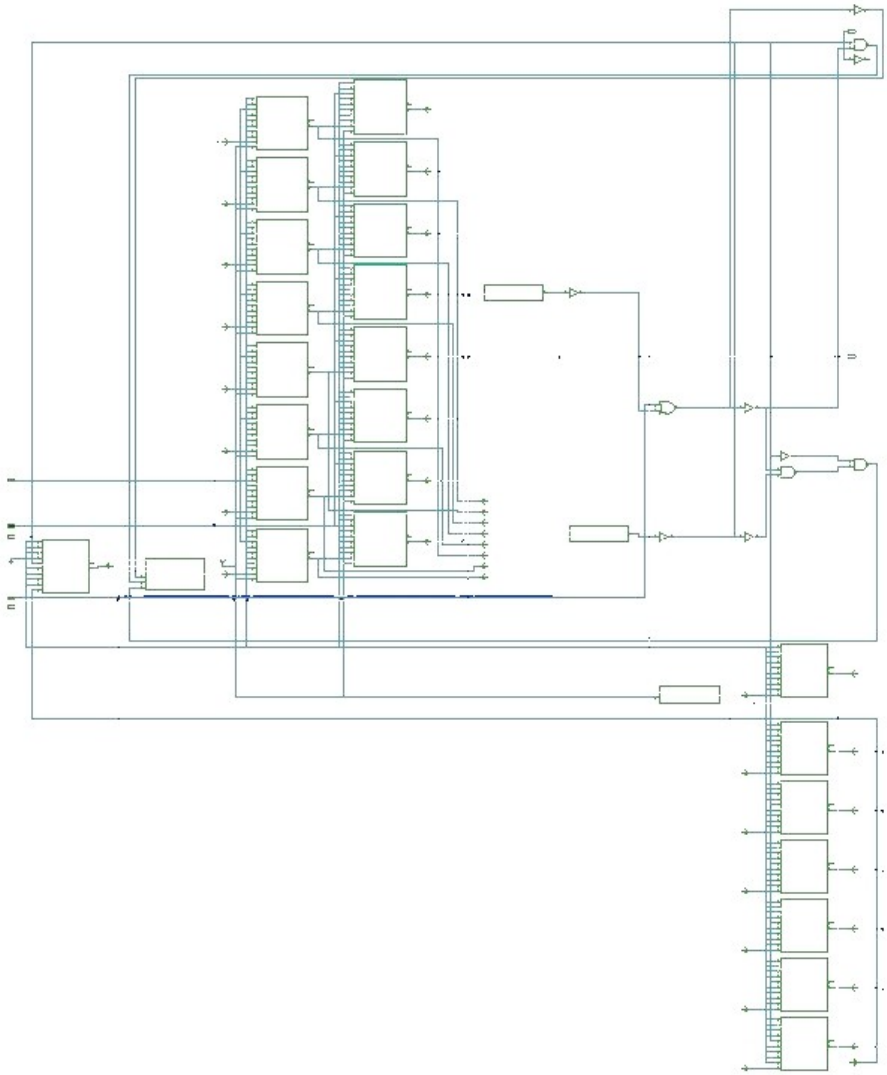

Figure 9. Schematic of the asynchronous counter

1) The Flip-Flop
2) The Incrementer
3) The Comparator

The comparator is designed to work at the clock edge, whereas the flip-flop is designed to function based on the generation of a completion signal from the previous module. The incrementer is designed to work asynchronously.

The three blocks are interfaced by the handshake mode of communication. Various intermediate completion signals are generated to make the individual blocks work sequentially without any racing between them.

3.2.2 Horizontal Timing

Figure 10 shows the schematic of asynchronous horizontal timing. Like synchronous horizontal timing, the asynchronous one has four counters, one each for the display end, blanking start, blanking end, and horizontal total, connected in series, whose limits are given by their respective registers. The counter begins with an initial value of zero. As and when each dot information or pixel is displayed, the counter is incremented.

The first counter, i.e., the counter for the display end, counts on every edge of the dot clock, whereas the other counters count at the completion of the signal from the previous counter. That is, the completion out signal from the display end counter triggers the counter for the blanking start. This is the handshake protocol, in which a module works after the completion of a previous module.

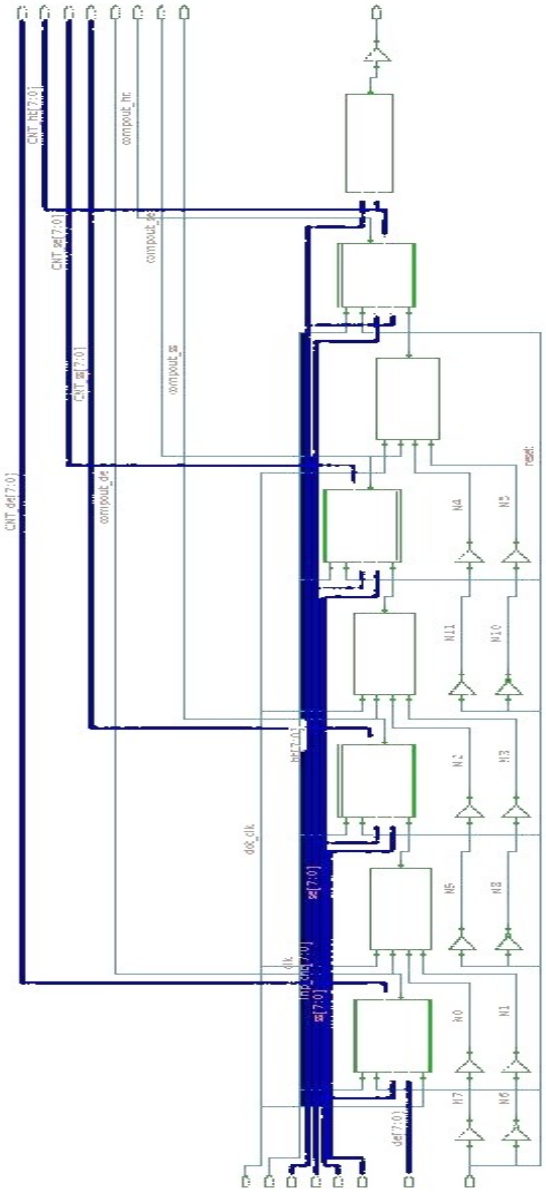

Figure 10. Schematic of asynchronous horizontal timing

3.2.3 Vertical Timing

The asynchronous vertical timing shown in Figure 11 works similarly to the asynchronous horizontal timing, with the exception that the first counter in horizontal timing is triggered by every dot clock whereas the first counter in vertical timing is triggered at the end of every scan line or horizontal line. Since the horizontal retrace signal is generated at the end of the horizontal line, the display end counter in vertical timing is triggered whenever a horizontal retrace signal is generated. The other counters are triggered at the completion of the previous counters.

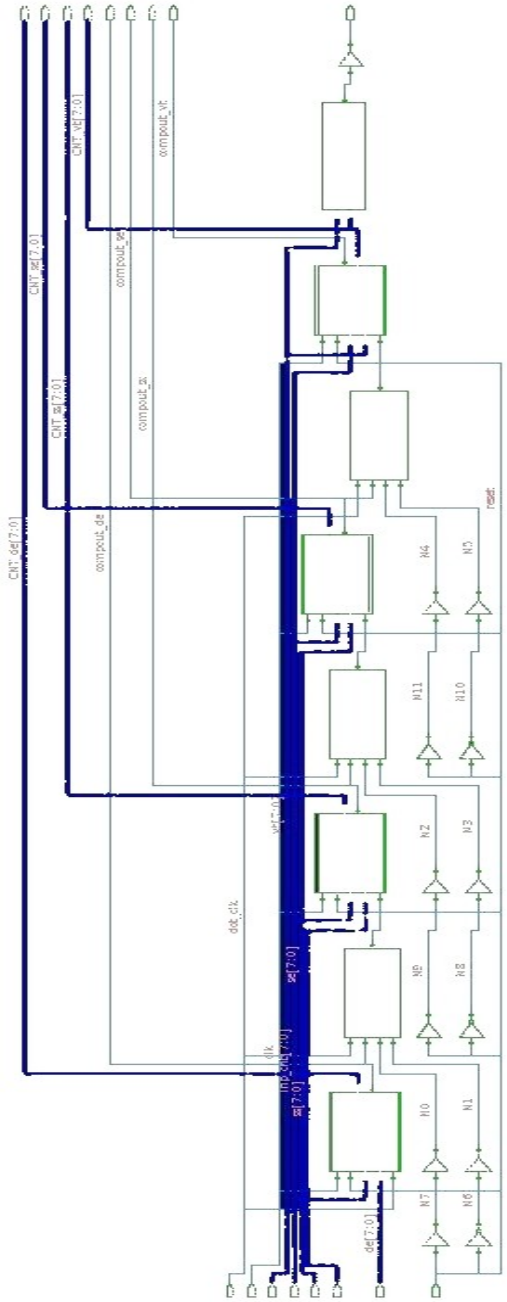

Figure 11. Schematic of asynchronous vertical timing

3.2.4 Asynchronous Display Controller

The asynchronous display controller shown in Figure 12 functions similarly to the synchronous display controller. The dot clock does not maintain global synchrony in this case. The dot clock is used only to initially trigger the counters, and later, the completion out signals from the previous module are used to make the display controller function in a sequential handshake fashion.

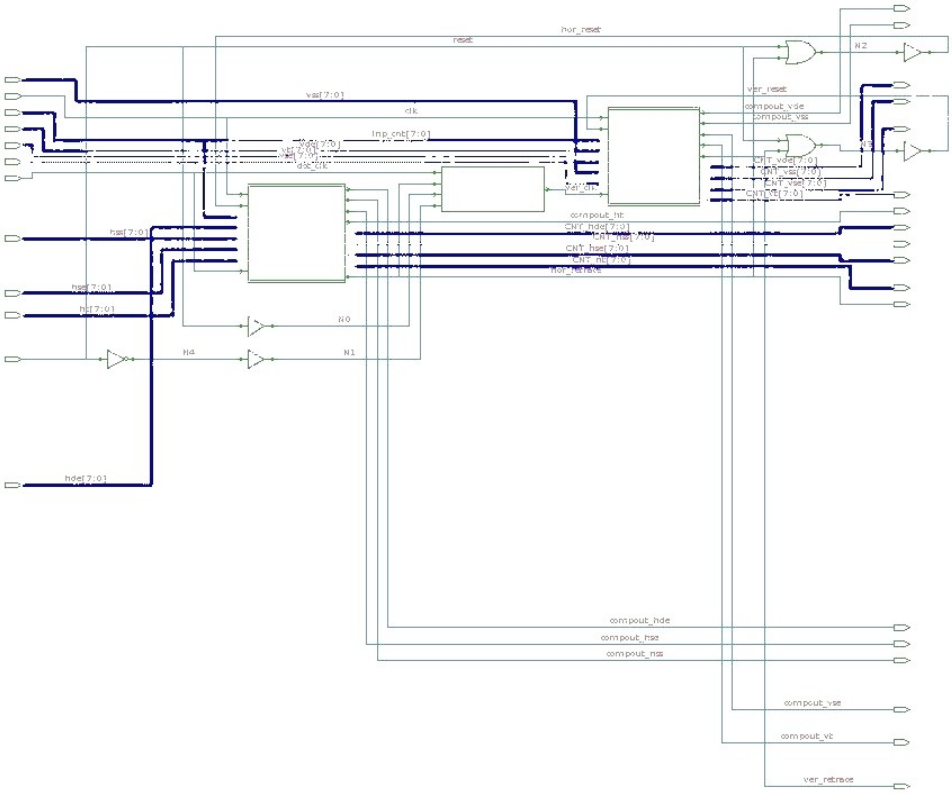

Figure 12. Schematic of asynchronous display controller

CHAPTER 4

4. Power Analysis of Synchronous and Asynchronous Designs

4.1 Introduction

As discussed in Chapter 1, the need for low power design is motivated by several factors, such as the emergence of portable systems, thermal considerations, reliability issues, and finally environmental concerns. Thus, an in-depth power analysis is required to design a low power system and avoid these problems.

4.2 Power Analysis in Synchronous Display Controller

The principal source of power consumption in digital circuits is dynamic switching power consumption, P_{sw}. From equation (1.2), the frequency and voltage values are fixed for a given design. The capacitance is calculated as the sum of all the parasitic capacitances from all the interconnects in the circuits and the load capacitance of the circuit. The only variable in equation (1.2) that needs to be found is the activity factor, N, which is the average number of switching transitions per clock cycle.

The number of signal switching transitions can be observed from the VCD file, which was obtained by simulating the RTL design for a specified number of clock cycles. The synchronous display controller was simulated for about 10,000 clock cycles with the

clock running at a frequency of 500 MHz; it was observed that the design had about 3, 81,019 switching events. The circuit was operated at a global operating voltage of 2.5 V, and for ease of calculation, the total parasitic capacitance of the circuit was assumed to be 1ff.

Now the dynamic switching power in the synchronous display controller can be calculated as below:

$$P_{sw} = \frac{1}{2} C_o V_{in}^2 N f$$

$$= \frac{1}{2} * 1 * 10^{-15} * (2.5)^2 * 38 * 500 * 10^6$$

$$= 59.375\ \mu W$$

This dynamic power, thus calculated, is mostly due to signal transitions in the circuit. This power increases as the clock frequency increases, as there will be more transitions in the circuit. In addition, as the clock frequency increases, the energy/voltage required to run the clock continuously increases. This further increases the dynamic power consumption.

Moreover, in synchronous circuits, the clock will be running continuously, even when the circuit is idle. Thus, the energy required to run the clock and other components of the circuit is wasted, considering that there will be no change in the system output. In addition, due to the continuous generation of clock signal, in the flip-flops there will be internal transitions even if the output does not change.

In this case of synchronous display controller design, the clock runs at 500 MHz; i.e., the clock signal toggles every 2 ns. Since all the flip-flops in the design are positive edge triggered, there will be internal node activity in all the flip-flops every 4 ns even if it

does not result in an output change. Taking into account that the synchronous display controller design has 64 flip-flops, there is adequate power wastage during the idle state.

4.3 Power Analysis in Asynchronous Display Controller

The dynamic power consumption in the asynchronous display controller was also calculated using equation (1.2). As in the synchronous one, the global operating voltage was 2.5 V, and the total parasitic capacitance was assumed to be 1 ff for ease of calculation.

The number of signal switching transitions was observed from the VCD file. The asynchronous display controller was simulated for about 10,000 clock cycles with the clock running at a frequency of 100 MHz; the design had about 3, 52,472 switching events.

Now the dynamic switching power in the synchronous display controller can be calculated as below:

$$P_{sw} = \frac{1}{2} C_o V_{in}^2 N f$$

$$= \frac{1}{2} * 1 * 10^{-15} * (2.5)^2 * 35 * 100 * 10^6$$

$$= 10.938\ \mu W$$

The dynamic power thus obtained was observed to be much less than the one obtained for the synchronous display controller. This power reduction is mainly due to the reduced dependency on the clock signal in the asynchronous display controller. In addition, in the asynchronous display controller, the clock ran at 100 MHz, and the

energy required to drive the clock was much less than the synchronous one.

Moreover, the asynchronous display controller has about 128 flip-flops, 64 of which are triggered at the clock edge while the remaining 64 are triggered at the completion of a previous module. The 64 flip-flops triggered by the completion of a signal from a previous module has no internal node activity during the idle state because the completion signals are generated only when there is a change in the output. In addition, since the clock signal toggles every 9 ns, the flip-flops triggered by the clock signal do not have any internal node activity during idle state, because the necessary output change from the previous module (i.e., the incrementer) occurs before the next positive edge of the clock for the flip-flops to change state.

Moreover, the rest of the display controller works asynchronously without requiring a clock signal to coordinate the data. This reduced dependency on the clock signal in the design has resulted in the reduction of dynamic power consumption.

4.4 Power Reduction Using RTL Clock Gating in Synchronous Circuits

One of the frequently used power reduction techniques for synchronous circuits is clock gating. Clock gating includes additional logic to the circuit to reduce the clock tree. Clock gating disables a portion of the synchronous circuit so that the switching power of the flip-flop goes to zero and only leakage currents are incurred [19].

Clock gating works only by using the enable conditions in the flip-flops to gate the clock signal. If there is no enable signal in the design, implementing clock gating

will not be possible. When the enable signal goes high, the flip-flops are clocked, whereas when the enable signal goes low, the flip-flops maintain their previous state [20].

There are two types of clock gating styles: the latch-based or latch-free clock gating styles. In the latch-free clock gating style, a requirement needs to be imposed on the circuit that all enable signals should be held constant from the rising edge until the falling edge of the clock to avoid truncating the generated clock pulse prematurely or generating multiple clock pulses unnecessarily. Whereas in the case of the latch-based clock gating style, a level-sensitive latch is added in the design to hold the enable signal [20].

Though gated clocks are good for saving power, managing the timing during synthesis and STA is very difficult. In addition, it is very hard to debug the circuit with gated clocks.

4.5 RTL Clock Gating in Asynchronous Display Controller

Clock gating in asynchronous circuits can be termed “perfect clock gating” since asynchronous circuits exhibit data-dependent behavior rather than being dependent on clock signals. Since asynchronous design does not use clock signals, it can be said that clock gating techniques are approximations of asynchronous designs [19].

The asynchronous display controller thus designed does not have a global clock to control all the modules. The clock signal is just used to trigger the initial comparator, and the subsequent modules are triggered by the completion out signal from the previous

module. Thus, there is no activity in the flip-flops and registers unless needed. This saves a considerable amount of power. It can be said that asynchronous circuits are gated by nature.

4.6 Power Reduction Using Multiple Supply Voltages in Synchronous Circuits

The dynamic power consumption of the circuit is quadratically proportional to the supply voltage, according to equation (1.2). Thus, great savings in power consumption can be created by reducing the supply voltage. However, a decrease in the supply voltage will also degrade the circuit speed. Nevertheless, the power consumption of a circuit can be reduced without impacting its performance [21].

In general, some pipeline stages in the circuit will be operating at different clock frequencies than others. There is power wastage in stages that operate at a slower clock frequency due to the global clock and high supply voltage. The difference in latencies in different pipeline stages can be used to reduce power wastage by using multiple supply voltages [21].

Due to the global clock signal and synchronous nature of the circuit, all the stages consume power even when they are idle. Moreover, when the supply voltage is high, the energy consumed by the circuit is also high, but most of the power is wasted by the stages that are idle. Thus, by applying lower voltages to stages that operate at lower frequencies and higher voltages to stages that operate at higher frequencies, lots of power can be saved without compromising speed.

In practice, matching voltage values with the latency of each module is very expensive. It is possible only to have about two or three supply voltages for the entire circuit [21]. In addition, while considering different power voltages, the global clock lines will be powered to the higher voltage, which will limit the amount of power savings.

4.7 Multiple Supply Voltages in Asynchronous Display Controller

Since the asynchronous display controller considered in this thesis does not have a global common clock signal, the controller goes into the idle state by nature. As mentioned earlier, each module is triggered by the completion out signal from the previous module. All the stages that operate at different frequencies will consume power only when they are active. Thus, no power is wasted during the idle state.

This nature of the asynchronous display controller saves a considerable amount of chip power and avoids the usage of multiple supply voltage thereby saving the cost of production.

4.8 Power Reduction Using Power Gating Technique in Synchronous Circuits

As the transistor feature size keeps scaling down to nanometer technology, power consumption has been a major problem. And since the supply voltage and threshold voltage are reduced along with the transistor size, there is a rapid increase in leakage

power dissipation [22].

Power gating is a technique that uses multiple sleep mode transistors, with high threshold voltage, gating the supply voltage to the idle blocks in the circuit. The sizing of sleep transistors is a very important design parameter. The power gating technique is used to reduce the stand-by or leakage power [23].

Transistors with different threshold voltages are used in the circuit. The low threshold transistors have high leakage but offer high speed whereas the high threshold transistors reduce the leakage power and reduce the circuit speed. By using both types of transistors in different stages of the circuit, leakage power can be reduced and performance preserved [22].

However, there are some drawbacks. The generation of sleep signals is very critical and often needs complex circuitry. This in turn increases the circuit area and power consumption. When the circuit switches from sleep mode to active mode, the circuit ground takes a long time to discharge through the sleep transistors. This latency affects overall performance and limits power leakage. In addition, synchronous circuits lose data when power transistors are turned off.

4.9 Power Gating in Asynchronous Display Controller

Leakage power is mainly due to the sub-threshold leakage current in the circuit. As discussed earlier, the asynchronous display controller considered in this thesis goes into the idle state by nature and does not consume power during this state. Since each

module works from the completion out signal from the previous module, there will be no activity in those modules until the previous modules change state. When the output of the previous module does not change state, the completion out signal will be less active, the subsequent modules will remain idle, and there will be no internal node activity in the circuit.

4.10 Benefits of Power Analysis Tools

Considering the increasing use of high-level design methods and the importance of low power design, power analysis and optimization are very important in every stage of the design. At higher levels, power optimization opportunities are significantly larger [24]. The power optimization techniques at the algorithmic and architectural levels yield large power savings, whereas the power savings obtained by logic and layout optimizations are much smaller. The power reduction requirements cannot be met for most of the designs by performing logic-level or transistor-level optimizations alone. There is a need to integrate power optimization techniques into the higher levels of the design flow.

The main purposes of power analysis tools are to:

- Validate if power requirements are met at different stages of the design and, if not, to identify the modules that consume more power.
- Evaluate the effect of other optimizations, such as area and timing, on power.

With the help of high-level power analysis tools for the above-mentioned purpose,

the design cycle can be greatly reduced.

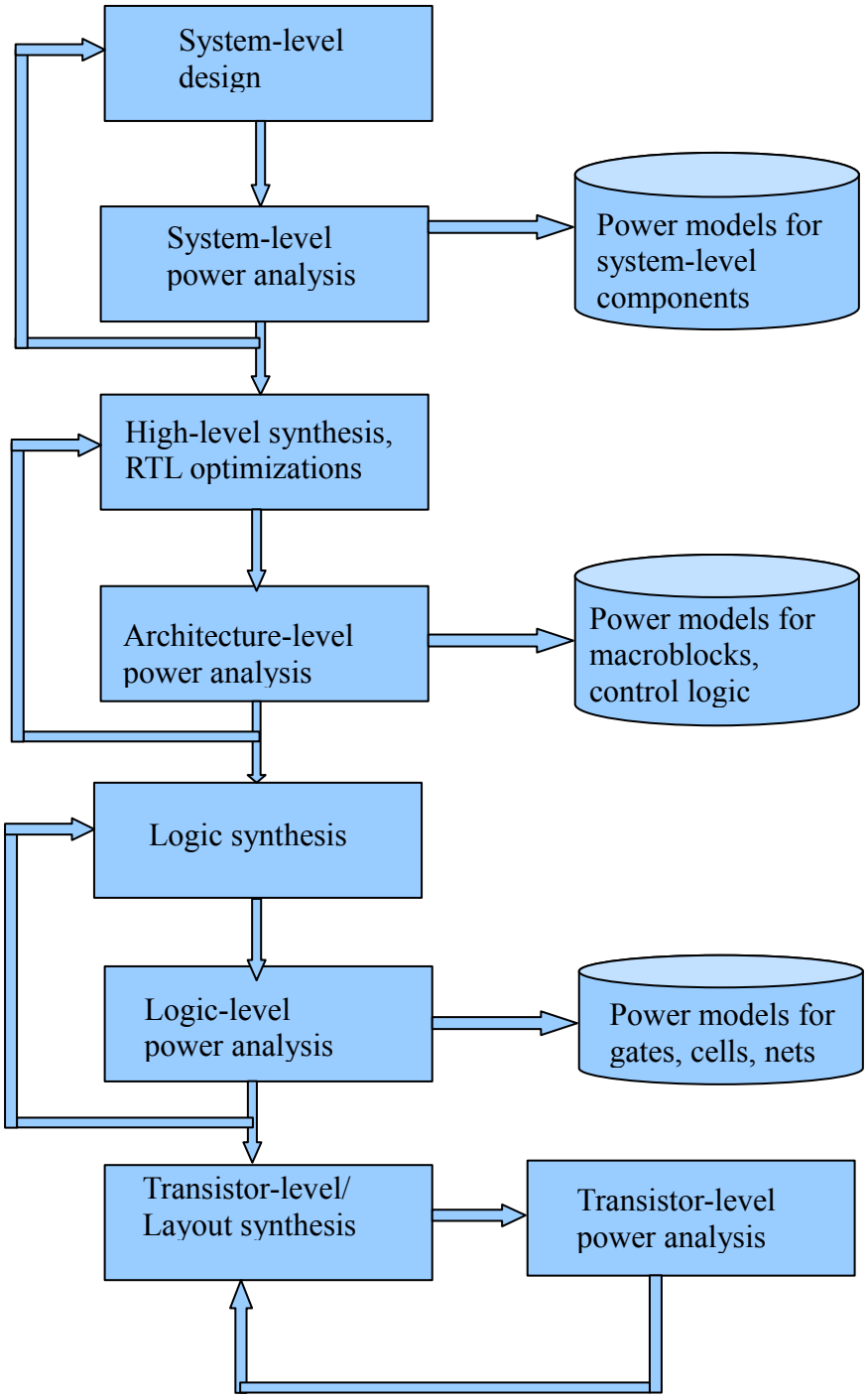

Figure 13. Design flow with high-level power analysis [24]

As Figure 13 shows, power analysis is performed at each level of the design flow. The power requirements at each level of the design are analyzed, and their trade-offs are supported by the corresponding tools at the same level, leading to fewer and faster design iterations. In the absence of high-level power analysis tools, the designer has to design and validate the lower level netlist and then run the transistor-level power analysis. Given the fact that lower-level power analysis tools require large run times and more time is required to obtain and validate the transistor-level netlist, this method is considered very inefficient [24].

The high-level power analysis tools are not as accurate as the lower-level analysis tools but are still helpful in managing the high-level design trade-offs. These tools also make the use of low-level power analysis tools easier and make sure that the power budgets are met confidently.

4.11 Synopsys Power Compiler

The key to successful power analysis tools is the automatic power reduction technique. This helps designers meet power budgets without affecting productivity or time to market. The Synopsys Power Compiler is one tool that automatically minimizes power consumption at the Register Transfer Level (RTL) and gate level. At the design elaboration phase, at RTL, the Power Compiler performs automatic clock gating to reduce power consumption. At the gate level, with the given design constraints, the Power Compiler performs optimizations for timing, area, and power simultaneously [25].

4.11.1 RTL Power Optimization

Synopsys Power Compiler implements automatic clock gating at the RTL without having to change the RTL source. This helps to maintain technology independent RTL source. Clock gating is a common power reduction technique that is used manually to reduce power in many power-critical designs. Power Compiler gates the clocks of individual synchronous load-enable register banks instead of circulating the output to the input when the load-enable condition is invalid. This is automated during the design elaboration phase, with no effort required from the design engineers. Up to 70% power savings can be obtained at the block level using this technique [25].

4.11.2 Gate-level Power Optimization

Power Compiler offers push-button power reduction at the gate level. On average, up to 20% of power can be reduced using Power Compiler at the gate level. Power Compiler delivers the lowest power-consuming design, considering the trade-offs between timing, area, and power, without violating either the timing constraints or the area constraints. The push-button power optimization reduces the dynamic power as well as the leakage power, which is the majority of the power consumed when the device is idle [25].

4.12 Tool-based Power Analysis for Synchronous Display Controller

Synopsys Power Compiler estimates the switching power, internal power, and leakage power. The tool uses the synthesis libraries that are integrated with power information. The libraries also have multiple energy look-up tables that calculate power based on the input and output pin states.

The design was mapped to the cells in the Toshiba tc240c library, and the operating conditions were set. For more information on setting operating conditions, refer to the script files in Appendix B. The synchronous display controller design was synthesized using the above setup, and the total dynamic power consumed was observed to be 1.5108 mW.

Dynamic power, the power consumed when the circuit is active, is further categorized into *internal power* and *switching power*. The internal power is caused by the charging and discharging of internal cell capacitances. Internal power also includes the short-circuit power. The synchronous display controller was built using 6 cells and about 797 NAND gates. Its cell internal power was observed to be 1.1227 mW. The switching power is due to the charging and discharging of load capacitances. The synchronous display controller consisted of 144 nets and 140 ports, and its net switching power was observed to be 388.1162 μW.

4.13 Tool-based Power Analysis for Asynchronous Display Controller

The asynchronous display controller was mapped to the cells in the Toshiba tc240c library, and the initial operating conditions were set. The design was synthesized using the above setup, and the total dynamic power consumed was observed to be 1.2562 mW.

The asynchronous display controller was built using 6 cells and about 1703 NAND gates. Its cell internal power was observed to be 929.2620 μW. It also consisted of 153 nets and 149 ports, and the net switching power was observed to be 329.9382 μW.

Though the asynchronous display controller is built using more NAND gates and nets (interconnects) compared to the synchronous display controller, the dynamic power consumed is much less. This is due to the fact that the asynchronous display controller has fewer switching events than its synchronous counterpart. The switching power and the short circuit power are reduced in asynchronous design.

Thus, the dynamic switching power consumed can be reduced by about 17% by designing the circuit in asynchronous logic, which reduces the dependency on clock signals in the circuit.

CHAPTER 5

5. Trade-offs in Asynchronous Design

Asynchronous circuits have a number of performance-based advantages over their synchronous counterparts. One of the most favorable effects of using asynchronous logic is that it consumes less power due to the absence of a clock signal. Since a clock signal consumes about 50% of the overall power consumed by the chip [26], asynchronous circuits can save an equivalent amount of chip power by reducing the dependency on a clock signal in the circuit. In addition, asynchronous circuits consume power only when the system is functioning, and in an active system, only the subsystem that is functioning consumes power.

Moreover, due to the presence of a global clock, synchronous circuits have problems such as unbalanced stages and clock skew [27]. In the case of unbalanced stages, the worst case delay is calculated by the speed of the stage, and the performance of the system is calculated based on that speed. In contrast, asynchronous circuits do not have a global clock. The worst case delay is calculated based on the delay in data. Hence, the performance of asynchronous circuits is based on the average delay and not the worst case delay.

Since less power is consumed in asynchronous circuits, the reliability of the system also increases. Unlike synchronous systems, asynchronous systems do not have to wait for the clock signal to trigger successive stages events. Due to the pipeline

controller logic and implementation of a handshake protocol, successive stage events are triggered as and when the current stage is completed.

Even though asynchronous circuits have many such performance-based advantages over synchronous circuits, compared to synchronous systems the former are rarely implemented today. This is due to some minor defects, such as area overhead, consecutive delays, and design complexity, which outweigh the advantages of moving completely into asynchronous design.

5.1 Area Overhead

The area of a design is one of the primary interests after synthesizing the design. The reporting commands in Synopsys Design Compiler provide additional information necessary for analyzing the design structure. The total circuit area is a combination of three factors: *combinational*, *non-combinational*, and *net interconnect* area. The total area formed by the logic cells of the design is shown by the *combinational area*. The total area formed by the registers in the design is shown by the *non-combinational area*. The total area formed by the nets or wires connecting the cells in the design is given by the *net interconnect area*. The unit for this area is defined by the libraries to which the cells are mapped.

5.1.1 Synchronous Display Controller

The design was mapped to the cells in the Toshiba tc240c library, and the operating constraints were set. The synchronous display controller design was synthesized using the above setup, and the total area of the design was observed to be 1903.47 μm^2.

The synchronous display controller was built using 6 cells and 797 NAND gates, and the total cell area was observed to be 797.5 μm^2. It also consisted of 144 nets and 140 ports, and the net interconnect area was observed to be 1105.97 μm^2.

5.1.2 Asynchronous Display Controller

The design was also mapped to the cells in the Toshiba tc240c library, and the operating constraints were set. The asynchronous display controller design was synthesized using the above setup, and the total area of the design was observed to be 3821.63 μm^2.

The asynchronous display controller was built using 6 cells and about 1703 NAND gates, and the total cell area was observed to be 1703.5 μm^2. It also consisted of 153 nets and 149 ports, and the net interconnect area was observed to be 2118.13 μm^2.

Thus, the increase in area for the asynchronous display controller is quite evident.

There is a clear increase in the number of logic cells, i.e., the NAND gates, and the number of interconnects and ports. This is mainly due to the handshake circuits, implemented to trigger the flip-flop and the incrementer in the asynchronous counters. There are eight such counters present in the display controller. In addition, handshake circuits were implemented to increment the vertical line after completion of a horizontal line. These handshake circuits occupy a considerable amount of area in the asynchronous display controller design.

Thus, the area occupied by the asynchronous display controller is about twice the area occupied by the synchronous display controller.

5.2 Timing Overhead

Timing or speed of the circuit is another important parameter of interest after the design is synthesized. The reporting commands in Synopsys Design Compiler provide additional information for performing timing analysis in the design. Since synchronous circuits have a global clock, their timing is calculated based on the worst case delay. Synopsys Design Compiler also performs timing analysis based on a global clock.

Since asynchronous circuits do not have a global clock, the reports on timing, obtained from Synopsys Design Compiler, may not be reliable for asynchronous logic. Hence, the timing analysis for this thesis is done based on the frame rate of the synchronous and asynchronous display controllers.

The synchronous display controller was run for about 10,000 clock cycles, and it

was found that the time taken to complete one frame was about 1500 clock cycles. Similarly, the asynchronous display controller was also run for 10,000 clock cycles, and the time taken to complete one frame was observed to be about 4500 clock cycles.

For an asynchronous design, the use of handshaking makes the timing in one component dependent on the timing of its neighbor, which in turn depends on the timing of its neighbors and so on. In the asynchronous counter, the timing of the flip-flop is dependent on the comparator, and the timing of the incrementer is dependent on the timing of the flip-flop, and the comparator is clocked. In addition, in the asynchronous horizontal and vertical timing circuits, the flip-flop of the next asynchronous counter is triggered by the completion out signal from the previous counter.

Due to this timing dependency, a delay in data from one block gets propagated to the consecutive blocks. In the asynchronous counter, since the timing of the comparator depends on the clock signal and since the clock frequency is very low, i.e., 100 MHz, the delay in the comparator is propagated throughout the circuit, which causes the delay in the asynchronous display controller.

5.3 Power Overhead

Although asynchronous logic saves a considerable amount of chip power by removing the global clock factor from the design, there is a significant amount of overhead in power consumption. As discussed before, the global clock is being replaced by a pipeline controller logic to hand over the control to the next stage once the current

stage is completed. This is done with the help of handshake circuits.

Due to these handshake circuits, numerous control signals are being generated to transfer the control from one stage to the next. These control signals contribute to the capacitive switching power when they are toggling states. If the number of stages increases, the number of control signals will also increase, which further increases the dynamic power of the circuit.

The synchronous display controller design was extracted and observed to have about 5512 signals in total. The asynchronous display controller design was extracted and observed to have about 9826 signals in total. This increase in the number of signals in the design is due to the implementation of handshake circuits, and it contributes to a significant portion of the total dynamic power of the circuit. If it were not for this overhead in power, about 50% of power can be saved by going in for asynchronous logic [26].

5.4 Design Complexity in Asynchronous Logic

The problems discussed in the previous sections, such as area and timing overhead, are normal when the circuit is designed for low power consumption. By removing the global clock from the design, the power overhead due to excessive control signals is also expected. These are some of the minor and expected trade-offs of designing the circuit in asynchronous logic for low power. The major obstacle that hinders designers from completely moving into asynchronous logic is its design

complexity.

5.4.1 Hazards

A hazard is an unwanted glitch in the signal related to the dynamic operation of the circuit. This depends on the changes in the input signal and the gate and wire delays in the circuit [28]. The primary difference among synchronous and asynchronous designs is the difference in handling hazards and glitches in the circuit.

Synchronous systems compute only at the tick of the clock. Hence, glitches in the circuit during the clock cycle are normally not a problem at all. As long as the signal is valid before the next clock edge, the system will function properly. But in asynchronous systems, since there is no global clock, the signal needs to be valid throughout. A small glitch in between can be treated as a real change in the system and can even cause the system to fail. Therefore, asynchronous circuits have to be designed very carefully so that hazards are eliminated throughout the process, which increases the design time and complexity.

5.4.2 Testing Asynchronous Circuits

Proper tools for synthesizing and testing pure asynchronous circuits are not

available. Hence, synchronous tools have to be used with tricky modifications to make them think the design is synchronous. A clock signal has to be included in the asynchronous circuit and should not be made global to synchronize all the components in the circuit. In this way, the synchronous tools can be deceived into thinking the design is synchronous. In the case of the asynchronous counter, the clock signal was used to trigger the comparator block, and the other blocks were made to work asynchronously based on a completion signal from the previous block.

In addition, in synchronous circuits, since all the components are synchronized at the clock edge, there will be sequential execution of blocks and no racing in the circuit. As a result, the timing of the input signal is not important in synchronous circuits. However, in asynchronous circuits, due to the absence of a global clock, each block functions asynchronously based on the input the block receives at that moment. Thus, the timing of the input signal is very important, because it might lead to problems such as racing and an unknown state of the circuit. Therefore, the input signals have to be carefully timed to avoid such problems, which further increases the design time and complexity.

Debugging asynchronous circuits is another challenging task. Due to the absence of a global clock and the dependency of the control signals generated from its neighboring module, the entire logic has to be debugged to find the solution to a problem. This increases the design complexity further.

All of the above problems were encountered while designing and testing the asynchronous display controller, which took approximately 3 months to completely

design and test. By contrast, it took only about 1 month to design and test the synchronous display controller. Another notable difference is that the synchronous display controller has about 566 lines of source code, whereas the asynchronous display controller has about 986 lines of source code.

CHAPTER 6

6. Conclusion and Future Work

6.1 Conclusion

This thesis has provided a novel approach for achieving low power design by using asynchronous logic. Low power design has achieved considerable importance due to the emergence of portable systems and their applications. Other factors, such as thermal considerations, reliability, and environmental concerns, force designers to look for power reduction techniques even in the initial design phase.

In synchronous circuits, the clock signal and clock distribution network consume about 50% of the overall chip power [26]. The major source of power consumption in digital systems is the dynamic capacitive switching power. In high-speed circuits, as the clock frequency increases, the capacitive switching power also increases proportionally.

Power reduction techniques are available at different levels, spanning the architecture level, register transfer level, gate level, and final layout realization level. Power consumption can be handled in any of these levels, ranging from the low level to the algorithmic level.

In this thesis, we opted for the algorithmic level for handling power consumption issues by choosing the asynchronous design style. In asynchronous logic, the clock signal is not entirely removed from the design. Due to the immense design complexity

of purely asynchronous designs, we opted for the GALS (Globally Asynchronous Locally Synchronous) design style.

Unlike clocked circuits, asynchronous systems aim at reducing noise, power consumption, and EMI. They exhibit an implicit idle behavior since they switch only when useful work is being done. Asynchronous circuits also save power caused by glitches, due to monotonic transitions between DATA and NULL. These circuits are also insensitive to delay and tolerant of power supply variations.

A simple display controller was designed in both synchronous and asynchronous logic to study the purpose. Unlike synchronous circuits, asynchronous circuits do not have a global clock to coordinate the progress of events. Hence, a pipeline controller logic is required to trigger the next stage once the current stage is completed. A handshake protocol was used to control the sequence of events. This ensures that the global clock is unnecessary. In addition, the use of the GALS design methodology maintains all the benefits of synchronous systems while eliminating the global clock factor.

The display controllers were designed using Verilog HDL and synthesized using Synopsys Design Compiler. The power analysis was performed using Synopsys Power Compiler, and the power consumed by both designs was measured. It has been shown that the asynchronous display controller consumed 17% less power than its synchronous equivalent.

The lower power in asynchronous design does not come at a lower cost. There is some overhead in the area, timing, and design complexity of the asynchronous design.

The major obstacle during the design of the asynchronous display controller was the timing of the generation of control signals to avoid racing among other asynchronous blocks. Another major design obstacle was the handling of glitches in the design. A small glitch in between asynchronous circuits can be treated as a real change in the system and can even cause the system to fail. In addition, due to lack of availability of proper tools, testing and debugging of asynchronous circuits are also major design obstacles.

The major performance obstacles in asynchronous systems are the area and speed. The area of the asynchronous display controller is twice that of the synchronous one, and the frame rate of the asynchronous display controller is three times slower than that of the synchronous display controller.

These are some of the reasons why most designers opt for synchronous over asynchronous design style.

6.2 Future Work

Since asynchronous design is still in the research phase, many related issues can be explored in the future. As shown in Chapter 4, the asynchronous design can be used to reduce 17% of the power consumption. The display controller designed for this thesis is partially asynchronous in nature. By going for a purely asynchronous design, the power consumption can be further reduced.

In addition, Chapter 5 discusses several hurdles that prevent designers from using

the asynchronous design style as commonly as the synchronous one. All of these issues can be addressed in the future in order to bring the asynchronous design style into common practice. The performance of the asynchronous display controller can be further improved in terms of area and speed. Effective synthesis tools and cell libraries can be created to synthesize fully asynchronous designs. Another very important extension of this work would be to develop an Electronic Design Automation (EDA) tool, which can translate from the existing synchronous design to an asynchronous design, finally saving 17% or the equivalent amount of power.

REFERENCES

[1] A. Ranganathan et al., *High-level Power Analysis and Optimization.* Norwell, MA: Kluwer Academic Publishers, 1998, pp. 2-5.

[2] J. Rabaey and M. Pedram, *Low Power Design Methodologies.* Norwell, MA: Kluwer Academic Publishers, 1996, pp. 21-24.

[3] Kang, *Digital VLSI.* New York, NY: McGraw-Hill, 2002, pp. 91-95.

[4] H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI.* Boston, MA: Addison-Wesley, 1990, pp. 62-66.

[5] A. Ranganathan et al., *High-level Power Analysis and Optimization.* Norwell, MA: Kluwer Academic Publishers, 1998, pp. 18-21.

[6] Anantha P. Chandrakasan et al., "Low-Power CMOS Digital Design," in *IEEE Journal of Solid-State Circuits*, 1992, pp. 473-484.

[7] Dake Liu and Christer Svensson, "Trading Speed for Low Power by Choice of Supply and Threshold Voltages," in *IEEE Journal of Solid-State Circuits*, 1993, pp. 10-17.

[8] Anantha Chandrakasan and Robert Brodersen, *Low-Power CMOS Design.* Norwell, MA: Kluwer Academic Publishers, 1998, pp. 61-63.

[9] Wikipedia. *Low-power electronics* [Online]. Available: http://en.wikipedia.org/wiki/Low-power_electronics

[10] Wikipedia. *Asynchronous circuit* [Online]. Available: http://en.wikipedia.org/wiki/Asynchronous_logic

[11] N. C. Paver and D. A. Edwards, "Is Asynchronous Logic Good for Low Power?," in Low Power Analogue and Digital VLSI: ASICS, Techniques and Applications, 1995 © IEEE. doi: 10.1049/ic:19950791

[12] Jens Sparsø and Steve Furber, *Principles of Asynchronous Circuit Design: A Systems Perspective.* Norwell, MA: Kluwer Academic Publishers, 2001, pp. 3-5.

[13] G. Birtwistle and A. Davis (Eds), *Asynchronous Digital Circuit Design.* Norwell, MA: Kluwer Academic Publishers, 1995, pp. 4-8.

[14] G. Birtwistle and A. Davis (Eds), *Asynchronous Digital Circuit Design.* Norwell, MA: Kluwer Academic Publishers, 1995, pp. 151-180.

[15] OSDev.org. *VGA Hardware* [Online]. Available: http://wiki.osdev.org/VGA_Hardware#The_CRT_Controller

[16] Richard F Ferraro, *Programmers Guide to the EGA and VGA Cards.* Boston, MA: Addison-Wesley, 1988, pp. 56-57.

[17] S. Hauck, "Asynchronous Design Methodologies: An Overview," in *Proceedings of the IEEE*, 1995, pp. 69-93.

[18] A. Hemani, "Lowering Power Consumption in Clock by Using Globally Asynchronous Locally Synchronous Design Style," in Proceedings of the 36th annual ACM/IEEE Design Automation Conference, 1999, © IEEE. doi: 10.1109/DAC.1999.782202

[19] Wikipedia. *Clock gating* [Online]. Available: http://en.wikipedia.org/wiki/Clock_gating

[20] Frank Emnett and Mark Biegel, "Power Reduction Through RTL Clock Gating," in *Automotive Integrated Electronics Corporation*, 2000, pp. 5-9.

[21] Diana Marculescu, "Power Efficient Processors Using Multiple Supply Voltages," in *Workshop on Compilers and Operating Systems for Low Power, in conjunction with PACT,* 2000, pp. 2-6.

[22] K. Agarwal, et al., "Power Gating with Multiple Sleep Modes," in 7th International Symposium on Quality Electronic Design, 2006, © IEEE. doi: 10.1109/ISQED.2006.102

[23] Wikipedia. *Power gating* [Onilne]. Available: http://en.wikipedia.org/wiki/Power_gating

[24] A. Ranganathan et al., *High-level Power Analysis and Optimization.* Norwell, MA: Kluwer Academic Publishers, 1998, pp. 12-15.

[25] Synopsys. *Power Compiler* [Online]. Available: http://www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/Power_Compiler.aspx

[26] Shekhar Borkar. (2003, June 6). *Does asynchronous logic design really have a future?* [Online]. Available: http://www.design-reuse.com/articles/5679/does-asynchronous-logic-design-really-have-a-future.html

[27] Charlie Brej, "High Performance Asynchronous Circuit Design Method and Application," in *School of Computer Science, The University of Manchester*, 2005, pp. 2-5.

[28] Jens Sparsø and Steve Furber, *Principles of Asynchronous Circuit Design: A Systems Perspective.* Norwell, MA: Kluwer Academic Publishers, 2001, pp. 80-84.

APPENDIX A

A.1 Synchronous Counter

```
module sync_counter(inp_cnt, LT, reset, clk, CNT);
  input clk;
  input [7:0] inp_cnt;
  input [7:0] LT;
  input reset;
  output [7:0]CNT;//output of flip flop before incrementing

  reg [7:0] CNT;//Intermediate register used for increment operation
  wire [7:0] CNT_D;//Intermediate wire from comparator

  assign CNT_D = (CNT==LT)?0:CNT+1;//Incrementer and Comparison

  always @ (posedge clk)
  begin
    if (reset)
      begin
        CNT <= inp_cnt;
      end
    else
      CNT <= CNT_D;//FF operation
  end
endmodule
```

A.2 Synchronous Horizontal Timing

```
module horizontal_timing(clk, inp_cnt, reset, de, ss, se, ht, CNT_de, CNT_ss, CNT_se,
CNT_ht, hor_retrace);
  input clk;
  input [7:0] inp_cnt;
  input reset;
  input [7:0] de;
  input [7:0] ss;
  input [7:0] se;
  input [7:0] ht;
  output [7:0] CNT_de, CNT_ss, CNT_se, CNT_ht;
  output hor_retrace;

  wire [7:0] CNT_de, CNT_ss, CNT_se, CNT_ht, init_ss, init_se, init_ht;
  wire reset_ss, reset_se, reset_ht, hor_retrace;

  assign hor_retrace = (CNT_ht==ht)?1:0;//Start Horizontal Retrace period

  sync_counter disp_end(.inp_cnt(inp_cnt), .LT(de), .reset(reset), .clk(clk),
.CNT(CNT_de));

  sync_counter sync_start(.inp_cnt(inp_cnt), .LT(ss), .reset(reset), .clk(clk),
.CNT(CNT_ss));

  sync_counter sync_end(.inp_cnt(inp_cnt), .LT(se), .reset(reset), .clk(clk),
.CNT(CNT_se));

  sync_counter hor_tot(.inp_cnt(inp_cnt), .LT(ht), .reset(reset), .clk(clk), .CNT(CNT_ht));

endmodule
```

A.3 Synchronous Vertical Timing

```
module vertical_timing(clk, inp_cnt, reset, de, ss, se, vt, CNT_de, CNT_ss, CNT_se,
CNT_vt, ver_retrace);
  input clk;
  input [7:0] inp_cnt;
  input reset;
  input [7:0] de;
  input [7:0] ss;
  input [7:0] se;
  input [7:0] vt;
  output [7:0] CNT_de, CNT_ss, CNT_se, CNT_vt;
  output ver_retrace;

  wire [7:0] CNT_de, CNT_ss, CNT_se, CNT_vt, init_ss, init_se, init_vt;
  wire reset_ss, reset_se, reset_vt, ver_retrace;

  assign ver_retrace = (CNT_vt==vt)?1:0;//Start Vertical Retrace period

  sync_counter disp_end(.inp_cnt(inp_cnt), .LT(de), .reset(reset), .clk(clk),
.CNT(CNT_de));

  sync_counter sync_start(.inp_cnt(inp_cnt), .LT(ss), .reset(reset), .clk(clk),
.CNT(CNT_ss));

  sync_counter sync_end(.inp_cnt(inp_cnt), .LT(se), .reset(reset), .clk(clk),
.CNT(CNT_se));

  sync_counter hor_tot(.inp_cnt(inp_cnt), .LT(vt), .reset(reset), .clk(clk), .CNT(CNT_vt));

endmodule
```

A.4 Synchronous Graphics Controller

```
module sync_graphics_controller (clk, inp_cnt, reset, hde, hss, hse, ht, CNT_hde,
CNT_hss, CNT_hse, CNT_ht, hor_retrace, vde, vss, vse, vt, CNT_vde, CNT_vss,
CNT_vse, CNT_vt, ver_retrace);
  input clk;
  input [7:0] inp_cnt;
  input reset;
  input [7:0] hde, hss, hse, ht;
  input [7:0] vde, vss, vse, vt;
  output hor_retrace, ver_retrace;
  output [7:0] CNT_hde, CNT_hss, CNT_hse, CNT_ht;
  output [7:0] CNT_vde, CNT_vss, CNT_vse, CNT_vt;

  wire [7:0] CNT_hde, CNT_hss, CNT_hse, CNT_ht, CNT_vde, CNT_vss, CNT_vse,
CNT_vt;
  wire hor_retrace, ver_retrace, hor_reset, ver_reset, ver_clk;

  assign hor_reset = (reset || hor_retrace) ? 1:0;
  assign ver_reset = (reset || ver_retrace) ? 1:0;
  assign ver_clk = (reset) ? clk:hor_retrace;

  horizontal_timing h1(.clk(clk), .inp_cnt(inp_cnt), .reset(hor_reset), .de(hde), .ss(hss),
.se(hse), .ht(ht), .CNT_de(CNT_hde), .CNT_ss(CNT_hss), .CNT_se(CNT_hse),
.CNT_ht(CNT_ht), .hor_retrace(hor_retrace));

  vertical_timing v1(.clk(ver_clk), .inp_cnt(inp_cnt), .reset(ver_reset), .de(vde),
.ss(vss), .se(vse), .vt(vt), .CNT_de(CNT_vde), .CNT_ss(CNT_vss),
.CNT_se(CNT_vse), .CNT_vt(CNT_vt), .ver_retrace(ver_retrace));

endmodule
```

A.5 Asynchronous Counter

```
module counter(inp_cnt, comp_in, clk, LT, reset, CNT, CNT_D, INC, CNT0,
comp_out,compare_lt);
  input [7:0] inp_cnt;//Input count from previous counter
  input comp_in; //Completion in signal from previous module
  input clk;//Input clock signal
  input [7:0] LT;//Counter limit value for display, sync etc
  input reset;//to initialize the counter
  output [7:0] CNT, CNT_D, INC;//Output counter value
  output comp_out, CNT0, compare_lt;//Completion out signal from current module

  reg [7:0] CNT, CNT_D, INC;
  wire compare_lt,INC0,counter_trigger,CNT0;

  assign comp_out = (CNT==CNT_D)?1:0;
  assign compare_lt = (CNT==LT)?1:0;//Comparison signal for the comparator
  assign CNT0 = ((CNT==inp_cnt))?1:0;//If INC is incremented

  always @ (posedge comp_in)
  begin
    //FF operation
    if(reset)
      CNT = inp_cnt;
    else
      CNT = CNT_D;
  end

  always @(*)
  begin
```

```
    if (reset)
      begin
        INC = inp_cnt;//Initializing the Counter/Incrementer
      end
    else if(comp_out)// !reset &&
      begin
        INC = CNT+1; //Increment Operation
      end
    else if(!comp_out)
      begin
        INC = INC;
      end
  end

  always @ (posedge clk)
  begin
    //Comparator Operation
    if (reset || compare_lt)
      CNT_D = inp_cnt;
    else if (comp_out)
      CNT_D = INC;
    else if (!comp_out)
      CNT_D = CNT_D;
  end
endmodule
```

A.6 Asynchronous Horizontal Timing

```
module horizontal_timing(inp_cnt, dot_clk, clk, reset, de, ss, se, ht, CNT_de, CNT_ss,
CNT_se, CNT_ht, compout_de, compout_ss, compout_se, compout_ht, hor_retrace);
  input [7:0] inp_cnt;
  input dot_clk;//to control the horizontal timing
  input clk;//for the counter
  input reset;//to initialize the counter
  input [7:0] de, ss, se, ht;
  output [7:0] CNT_de, CNT_ss, CNT_se, CNT_ht;
  output hor_retrace;//retrace signal
  output compout_de, compout_ss, compout_se, compout_ht;

  wire [7:0] CNT_de, CNT_ss, CNT_se, CNT_ht, init_ss, init_se, init_ht;
  wire compin_ss, compin_se, compin_ht;

  assign compin_ss = (reset) ? dot_clk:compout_de;
  assign compin_se = (reset) ? dot_clk:compout_ss;
  assign compin_ht = (reset) ? dot_clk:compout_se;

  //Start Horizontal Retrace period
  assign hor_retrace = (CNT_ht==ht)?1:0;

  counter disp_end(.inp_cnt(inp_cnt), .comp_in(dot_clk), .clk(clk), .LT(de),
.reset(reset), .CNT(CNT_de), .comp_out(compout_de));

  counter sync_start(.inp_cnt(inp_cnt), .comp_in(compin_ss), .clk(clk), .LT(ss),
.reset(reset), .CNT(CNT_ss), .comp_out(compout_ss));

  counter sync_end(.inp_cnt(inp_cnt), .comp_in(compin_se), .clk(clk), .LT(se),
.reset(reset), .CNT(CNT_se), .comp_out(compout_se));
```

```
  counter hor_tot(.inp_cnt(inp_cnt), .comp_in(compin_ht), .clk(clk), .LT(ht),
.reset(reset), .CNT(CNT_ht), .comp_out(compout_ht));

endmodule
```

A.7 Asynchronous Vertical Timing

```
module vertical_timing(inp_cnt, dot_clk, clk, reset, de, ss, se, vt, CNT_de, CNT_ss,
CNT_se, CNT_vt, compout_de, compout_ss, compout_se, compout_vt, ver_retrace);
  input [7:0] inp_cnt;
  input dot_clk;//to control the horizontal timing
  input clk;//for the counter
  input reset;//to initialize the counter
  input [7:0] de, ss, se, vt;
  output [7:0] CNT_de, CNT_ss, CNT_se, CNT_vt;
  output ver_retrace;//retrace signal
  output compout_de, compout_ss, compout_se, compout_vt;

  wire [7:0] CNT_de, CNT_ss, CNT_se, CNT_vt, init_ss, init_se, init_vt;
  wire compin_ss, compin_se, compin_vt;

  assign compin_ss = (reset) ? dot_clk:compout_de;
  assign compin_se = (reset) ? dot_clk:compout_ss;
  assign compin_vt = (reset) ? dot_clk:compout_se;

  //Start Vertical Retrace period
  assign ver_retrace = (CNT_vt==vt)?1:0;

  counter disp_end(.inp_cnt(inp_cnt), .comp_in(dot_clk), .clk(clk), .LT(de),
.reset(reset), .CNT(CNT_de), .comp_out(compout_de));
```

```
  counter sync_start(.inp_cnt(inp_cnt), .comp_in(compin_ss), .clk(clk), .LT(ss),
.reset(reset), .CNT(CNT_ss), .comp_out(compout_ss));


  counter sync_end(.inp_cnt(inp_cnt), .comp_in(compin_se), .clk(clk), .LT(se),
.reset(reset), .CNT(CNT_se), .comp_out(compout_se));


  counter hor_tot(.inp_cnt(inp_cnt), .comp_in(compin_vt), .clk(clk), .LT(vt),
.reset(reset), .CNT(CNT_vt), .comp_out(compout_vt));


endmodule
```

A.8 Asynchronous Graphics Controller

```
module asyn_graphics_controller(inp_cnt, dot_clk, clk, reset, hde, hss, hse, ht, CNT_hde,
CNT_hss, CNT_hse, CNT_ht, compout_hde, compout_hss, compout_hse, compout_ht,
hor_retrace, vde, vss, vse, vt, CNT_vde, CNT_vss, CNT_vse, CNT_vt, compout_vde,
compout_vss, compout_vse, compout_vt, ver_retrace);


  input [7:0] inp_cnt;
  input dot_clk;//to control the horizontal timing
  input clk;//for the counter
  input reset;//to initialize the counter
  input [7:0] hde, hss, hse, ht;
  input [7:0] vde, vss, vse, vt;
  output [7:0] CNT_hde, CNT_hss, CNT_hse, CNT_ht;
  output [7:0] CNT_vde, CNT_vss, CNT_vse, CNT_vt;
  output hor_retrace;//retrace signal
  output ver_retrace;//retrace signal
  output compout_hde, compout_hss, compout_hse, compout_ht;
  output compout_vde, compout_vss, compout_vse, compout_vt;
```

```
  //reg hor_reset, ver_reset;
  wire ver_clk, hor_reset, ver_reset;

  assign hor_reset = (reset) ? reset:hor_retrace;
  assign ver_reset = (reset) ? reset:ver_retrace;
  assign ver_clk = (reset) ? dot_clk:hor_retrace;

  horizontal_timing h1(.inp_cnt(inp_cnt), .dot_clk(dot_clk), .clk(clk), .reset(hor_reset),
.de(hde), .ss(hss), .se(hse), .ht(ht), .CNT_de(CNT_hde), .CNT_ss(CNT_hss),
.CNT_se(CNT_hse), .CNT_ht(CNT_ht), .compout_de(compout_hde),
.compout_ss(compout_hss), .compout_se(compout_hse), .compout_ht(compout_ht),
.hor_retrace(hor_retrace));

  vertical_timing v1(.inp_cnt(inp_cnt), .dot_clk(ver_clk), .clk(clk), .reset(ver_reset),
.de(vde), .ss(vss), .se(vse), .vt(vt), .CNT_de(CNT_vde), .CNT_ss(CNT_vss),
.CNT_se(CNT_vse), .CNT_vt(CNT_vt), .compout_de(compout_vde),
.compout_ss(compout_vss), .compout_se(compout_vse), .compout_vt(compout_vt),
.ver_retrace(ver_retrace));

endmodule
```

APPENDIX B

B.1 Script for Synchronous Graphics Controller

```
set link_library {/apps/toshiba/sjsu/synopsys/tc240c/tc240c.db_NOMIN25 /apps/synopsys/C-2009.06-SP5/libraries/syn/dw02.sldb /apps/synopsys/C-2009.06-SP5/libraries/syn/dw01.sldb }

set target_library {/apps/toshiba/sjsu/synopsys/tc240c/tc240c.db_NOMIN25}

read_verilog sync_counter.v

read_verilog horizontal_timing.v

read_verilog vertical_timing.v

read_verilog sync_graphics_controller.v

check_design

create_clock clk -name clk -period 20

set_propagated_clock clk

set_clock_uncertainty 0.25 clk

set_propagated_clock clk

set_output_delay -clock clk 1 {CNT_hde CNT_hss CNT_hse CNT_ht hor_retrace CNT_vde CNT_vss CNT_vse CNT_vt ver_retrace}

set_input_delay -clock clk 3 {inp_cnt hde hss hse ht vde vss vse vt}

set_wire_load_model -name T8G10TW8_W -library tc240c

compile -map_effort medium

update_timing
```

```
report -cell

report_timing -delay min -max_paths 10

report_timing -max_paths 10

write -f verilog -hierarchy -o sync_graphics_controller_map.v

quit
```

B.2 Script for Asynchronous Graphics Controller

```
set link_library {/apps/toshiba/sjsu/synopsys/tc240c/tc240c.db_NOMIN25 /apps/synopsys/C-2009.06-SP5/libraries/syn/dw02.sldb /apps/synopsys/C-2009.06-SP5/libraries/syn/dw01.sldb }

set target_library {/apps/toshiba/sjsu/synopsys/tc240c/tc240c.db_NOMIN25}

read_verilog counter.v

read_verilog horizontal_timing.v

read_verilog vertical_timing.v

read_verilog asyn_graphics_controller.v

check_design

create_clock clk -name clk -period 10

set_propagated_clock clk

set_clock_uncertainty 0.25 clk

set_propagated_clock clk

set_output_delay -clock clk 2 {CNT_hde CNT_hss CNT_hse CNT_ht compout_hde compout_hss compout_hse compout_ht hor_retrace CNT_vde CNT_vss CNT_vse
```

```
CNT_vt compout_vde compout_vss compout_vse compout_vt ver_retrace}

set_input_delay -clock clk 2.78 {inp_cnt reset hde hss hse ht vde vss vse vt}

set_wire_load_model -name T8G10TW8_W -library tc240c

compile -map_effort medium

update_timing

report -cell

report_timing -delay min -max_paths 10

report_timing -max_paths 10

write -f verilog -hierarchy -o asyn_graphics_controller_map.v

quit
```