Examensarbete

Reconfigurable FSM for Ultra-Low Power Wireless Sensor Network Nodes

Master thesis performed in Electronic Devices
by

Rengarajan Ragavan

LiTH-ISY-EX--13/4724--SE
Linköping, October 2013
Reconfigurable FSM for Ultra-Low Power Wireless Sensor Network Nodes

Master thesis in Electronic Devices
at Linköping Institute of Technology
by
Rengarajan Ragavan

LiTH-ISY-EX--13/4724--SE

Supervisor: Olivier Sentieys
Professor, University of Rennes 1 & Head, Cairn research group, IRISA/INRIA, France

Examiner: Atila Alvandpour
Professor, ISY, Linköpings Universitet, Linköping, Sweden

Linköping, October 2013
**Reconfigurable FSM for Ultra-Low Power Wireless Sensor Network Nodes**

**Author(s)**
Rengarajan Ragavan

**Abstract**

Wireless sensor networks (WSN) play an important role in today’s monitoring and control systems like environmental monitoring, military surveillance, industrial sensing and control, smart home systems and tracking systems. As the application of WSN grows by leaps and bounds, there is an increasing demand in placing a larger number of sensors and controllers to meet the requirements. The increased number of sensors necessitates flexibility in the functioning of nodes. Nodes in wireless sensor networks should be capable of being dynamically reconfigured to perform various tasks is the need of the hour.

In order to achieve flexibility in node functionality, it is common to adopt reconfigurable architecture for WSN nodes. FPGA-based architectures are popular reconfigurable architectures by which WSN nodes can be programmed to take up different roles across time. Area and power are the major overheads in FPGA based architectures, where interconnect consumes more power and area than logic cells. The contemporary WSN standard requires longer battery life and micro size nodes for easy placement and maintenance-free operation for years together.

Three solutions have been studied and evaluated to approach this problem: 1) Homogenous embedded FPGA platform, 2) Power gated reconfigurable finite state machines and 3) Pass transistor logic (PTL) based reconfigurable finite state machines. Embedded FPGA is a CMOS 65nm custom developed small homogenous FPGA which holds the functionality of the WSN nodes and it will be dynamically reconfigured from time to time to change the functionality of the node. In Power gated reconfigurable FSM architecture, the functionality of the node is expressed in the form of finite state machines, which will be implemented in a LUT based power gated design. In PTL based reconfigurable finite state machine architecture, the finite state machines are completely realized using PTL based custom designed sets of library components. Low power configuration memory is used to dynamically reconfigure the design with various FSMs at different times.

**Keywords**

Wireless Sensor Networks (WSN), Low Power, Pass Transistor Logic (PTL), FPGA, FSM, Reconfiguration.
ACKNOWLEDGEMENTS

I sincerely express my thanks to Prof. Olivier Sentiyes, the supervisor of this thesis, who gave me the opportunity to work him in prestigious INRIA/IRISA labs. His guidance and confidence in me gave extra boost and cultivated more interest in me to explore new things in new dimensions.

I am very grateful to my examiner and internal guide Prof. Atila Alavandpour for his encouragement and support. He was inspirational to me throughout my studies at Linköping University.

I would like to register my gratitude to Dr. Vivek Dwarknath without whom I am sure this thesis work would not have seen the light of the day. He paved the way for this thesis work and this master thesis work is the extension of his Ph.D. thesis.

I would like to thank Mr. Philippe Quemerais, Research Engineer, IRISA labs for his extensive support in configuring and tutoring various CAD tools, without his support it would not be possible to finish this thesis work on time.

I offer my sincere thanks to all my friends and well-wishers who are supporting me all through myself and looking for my betterment.

Last but not least, I express my everlasting gratitude to my parents, brother and almighty without them I cannot be what I am today.
To my parents
S.P. Ragavan & R. Radha

To my brother
R. Manikandan

And to my beloved teachers
T.M. Thiyagarajan & S. Sankaranarayanan
ABSTRACT

Wireless sensor networks (WSN) play an important role in today’s monitoring and control systems like environmental monitoring, military surveillance, industrial sensing and control, smart home systems and tracking systems. As the application of WSN grows by leaps and bounds, there is an increasing demand in placing a larger number of sensors and controllers to meet the requirements. The increased number of sensors necessitates flexibility in the functioning of nodes. Nodes in wireless sensor networks should be capable of being dynamically reconfigured to perform various tasks is the need of the hour.

In order to achieve flexibility in node functionality, it is common to adopt reconfigurable architecture for WSN nodes. FPGA-based architectures are popular reconfigurable architectures by which WSN nodes can be programmed to take up different roles across time. Area and power are the major overheads in FPGA based architectures, where interconnect consumes more power and area than logic cells. The contemporary WSN standard requires longer battery life and micro size nodes for easy placement and maintenance-free operation for years together.

Three solutions have been studied and evaluated to approach this problem: 1) Homogenous embedded FPGA platform, 2) Power gated reconfigurable finite state machines and 3) Pass transistor logic (PTL) based reconfigurable finite state machines. Embedded FPGA is a CMOS 65nm custom developed small homogenous FPGA which holds the functionality of the WSN nodes and it will be dynamically reconfigured from time to time to change the functionality of the node. In Power gated reconfigurable FSM architecture, the functionality of the node is expressed in the form of finite state machines, which will be implemented in a LUT-based power gated design. In PTL-based reconfigurable finite state machine architecture, the finite state machines are completely realized using PTL-based custom designed sets of library components. Low power configuration memory is used to dynamically reconfigure the design with various FSMs at different times.
# TABLE OF CONTENTS

Table of Contents ................................................................................................................................................. 8
List of Figures ............................................................................................................................................................ 10
List of Tables ............................................................................................................................................................ 10

1 INTRODUCTION ......................................................................................................................................................... 11
1.1 Motivation ......................................................................................................................................................... 11
1.2 Problem statement .......................................................................................................................................... 11
1.3 Structure of the work ...................................................................................................................................... 11
1.4 Organization of the thesis ............................................................................................................................... 12

2 BACKGROUND ............................................................................................................................................................ 13
2.1 Wireless sensor networks ............................................................................................................................. 13
2.2 Finite state machine ...................................................................................................................................... 16
2.3 Low power design ......................................................................................................................................... 17
2.4 Power gating .................................................................................................................................................. 18
2.5 Pass transistor logic ...................................................................................................................................... 19

3 EMBEDDED FPGA ...................................................................................................................................................... 20
3.1 Introduction ....................................................................................................................................................... 20
3.2 Homogenous embedded FPGA .................................................................................................................... 20
3.3 Conclusion ...................................................................................................................................................... 23

4 RECONFIGURABLE FSM .......................................................................................................................................... 24
4.1 Reconfigurable FSM ...................................................................................................................................... 24
4.2 Shannon’s expansion ..................................................................................................................................... 25
4.3 LUT realization of FSM ............................................................................................................................... 25
4.4 Power gating in reconfigurable FSM .......................................................................................................... 27
4.5 Pass transistor based reconfigurable fsm .................................................................................................. 29
  4.5.1 Lean integration pass transistor logic (LEAP) ....................................................................................... 30
  4.5.2 Transmission gates ............................................................................................................................... 31
4.6 Library components ...................................................................................................................................... 31
  4.6.1 LUT selector ........................................................................................................................................... 31
  4.6.2 logic gates................................................................................................................................................ 33
  4.6.3 Decoder .................................................................................................................................................. 34
4.7 Conclusion ...................................................................................................................................................... 37

5 CONFIGURATION LOGIC ......................................................................................................................................... 38
5.1 Introduction ...................................................................................................................................................... 38
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>Bucket Brigade circuit</td>
<td>38</td>
</tr>
<tr>
<td>5.3</td>
<td>Quasi-adiabatic flip flops</td>
<td>39</td>
</tr>
<tr>
<td>5.4</td>
<td>Dual context scan chain</td>
<td>40</td>
</tr>
<tr>
<td>5.5</td>
<td>Dynamic scan chain</td>
<td>41</td>
</tr>
<tr>
<td>5.6</td>
<td>Comparison of various configuration memories</td>
<td>42</td>
</tr>
<tr>
<td>5.7</td>
<td>Conclusion</td>
<td>43</td>
</tr>
<tr>
<td>6</td>
<td>EXPERIMENTS AND RESULTS</td>
<td>44</td>
</tr>
<tr>
<td>6.1</td>
<td>Spice simulation</td>
<td>44</td>
</tr>
<tr>
<td>6.2</td>
<td>Results</td>
<td>45</td>
</tr>
<tr>
<td>7</td>
<td>CONCLUSION</td>
<td>49</td>
</tr>
<tr>
<td>7.1</td>
<td>Future work</td>
<td>50</td>
</tr>
<tr>
<td>8</td>
<td>BIBLIOGRAPHY</td>
<td>51</td>
</tr>
<tr>
<td>Appendix A</td>
<td>Blif File</td>
<td>53</td>
</tr>
<tr>
<td>Appendix B</td>
<td>Kiss2 File</td>
<td>56</td>
</tr>
<tr>
<td>Appendix C</td>
<td>Perl script</td>
<td>58</td>
</tr>
<tr>
<td>Appendix D</td>
<td>Generated spice netlist</td>
<td>68</td>
</tr>
<tr>
<td>Appendix E</td>
<td>Testbench</td>
<td>76</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 2.1 WSN node block diagram.................................................................13
Figure 2.2 Task flow graph........................................................................14
Figure 2.3 Structure of micro-task................................................................15
Figure 2.4 WSN node architecture in micro-task model.................................15
Figure 2.5 Moore FSM.................................................................................16
Figure 2.6 Mealy FSM................................................................................16
Figure 2.7 Power gating...............................................................................18
Figure 3.1 eFPGA .......................................................................................21
Figure 3.2 Array element of eFPGA..............................................................21
Figure 3.3 Configuration logic block (CLB)..................................................22
Figure 3.4 Structure of a configuration bit......................................................22
Figure 3.5 Layout of eFPGA.......................................................................23
Figure 4.1 LUT realization of one state-register bit.........................................26
Figure 4.2 LUT realization of FSM output....................................................27
Figure 4.3 Coarse-grain power gated LUT architecture...............................28
Figure 4.4 Fine-grain power gated LUT architecture.....................................29
Figure 5.1 LEAP PTL................................................................................30
Figure 5.2 Transmission gates..................................................................31
Figure 5.3 LUT decoder............................................................................32
Figure 5.4 2-input AND gate...................................................................33
Figure 5.5 2-input OR gate......................................................................34
Figure 5.6 1:2 Decoder.............................................................................34
Figure 5.7 2:4 Decoder.............................................................................36
Figure 5.8 5:32 Decoder..........................................................................36
Figure 6.1 Bucket brigade scan chain..........................................................38
Figure 6.2 Adiabatic inverter (2N-2P)..........................................................39
Figure 6.3 Adiabatic flip flop (2N-2P)..........................................................40
Figure 6.4 Dual context scan cell.................................................................40
Figure 6.5 Dynamic scan chain.................................................................41
Figure 6.6 Dynamic scan chain working.....................................................42
Figure 7.1 Process flow.............................................................................44
Figure 7.2 abs FSM output waveform.........................................................46
Figure 7.3 abs FSM state changes..............................................................47

LIST OF TABLES

Table 3.1 Energy and power consumption of FSMs in eFPGA..........................22
Table 4.1 Truth table of X-NOR gate............................................................25
Table 4.2 Resource requirement for full reconfigurability................................26
Table 4.3 Energy and power consumption of FSMs in power gated reconfigurable FSM..................................................29
Table 5.1 Power consumption of LUT decoders............................................32
Table 6.1 Energy and power consumption of scan chains.............................42
Table 7.1 FSM configuration.....................................................................45
Table 7.2 Resource consumption by FSMs...................................................45
Table 7.3 Energy consumption by benchmark FSMs in PTL and power gated architectures..............................................48
1 INTRODUCTION

1.1 MOTIVATION

Due to the increased application of wireless sensor networks, WSN node hardware needs to be reconfigurable to perform various functions across time depending on the environment. Also the evolving trend of area reduction in electronic devices adds further constraints on WSN node architecture. Generally, the function of WSN nodes can be changed across time by implementing reconfigurable architectures like FPGAs, which in turn costs more area and power.

WSN nodes are generally powered using small batteries and the circuit needs to be functional for the long haul without battery replacement. In energy harvesting WSN nodes, the amount of energy harvested is not sufficient to keep the node alive in the long run. All these factors make us look for a new reconfiguration architecture which will reduce power and area significantly, and which at the same time provides flexibility in comparison to the existing design.

1.2 PROBLEM STATEMENT

The aim of this thesis work is to study in detail the implications of various low power design methods for reconfigurable finite state machines, the optimal granularity for power gating logic clusters based on energy saving vs. design overhead trade-offs, and analyzing the pros and cons of pass transistor logic style for reconfigurable FSM to avoid isolation cells used in power gating techniques.

For pass-transistor logic reconfigurable FSM, a library has to be created with a set of basic components designed using pass transistors which will be used to construct any given FSM in PTL style. Secondly, a piece of software has to be written to read an FSM in KISS2 format and generate a spice netlist with PTL library components.

1.3 STRUCTURE OF THE WORK

Firstly, the eFPGA is designed to achieve flexibility and to study the power consumption of the system. Secondly, reconfigurable FSM architecture has been designed to overcome the limitations of the eFPGA. Also, power gating technique has been introduced in reconfigurable FSM architecture to achieve more energy savings. Next the reconfigurable FSM architecture has been designed using pass
transistor logic. The basic components required to construct a reconfigurable FSMs are designed in pass transistor logic style. Thirdly, various configuration memories have been studied and compared to achieve low power configuration logic. Finally, any given FSMs in HDL description are converted to the state table format called ‘. Kiss2’. Then the software written in Perl is used to read the state table and estimate the number of basic components required to construct PTL based reconfigurable FSM. The Perl script will also generate a spice netlist and testbench for the FSM using PTL library components.

1.4 ORGANIZATION OF THE THESIS

The thesis report has been organized as follows: Chapter 2 gives background details of WSN, and low power design. Chapter 3 discusses custom developed embedded FPGA. Chapter 4 gives the understanding of finite state machines and the realization of reconfigurable FSM also discusses pass-transistor logic and how PTL can be used to implement a reconfigurable FSM. Chapter 5 deals with the investigation of various types of configuration memory used in the reconfigurable FSM. Chapter 6 presents the experiments and the results of this work. Chapter 7 summarizes conclusions and suggests possibilities for future work.
2 BACKGROUND

2.1 WIRELESS SENSOR NETWORKS

Today’s controlling and sensing tasks have become quite easy, thanks to WSN. As an example one can easily monitor various environmental factors like temperature, pressure, humidity and lighting conditions of a chemical plant while sitting few hundred meters away from the plant. Similarly WSNs are used in military for surveillance of the border, intrusion detection in a prohibited area, mechanical stress levels on attached objects and biological and chemical attack detections. WSNs are employed in various kinds of sensing like seismic, magnetic, thermal, noise, visual, speed, and direction.

A WSN node is a basic unit of wireless sensor network. The generic WSN node architecture is shown in figure 2.1. A WSN node consists of a sensing unit, control unit, communication unit and power supply unit. The sensing unit houses a variety of sensors depending on the application, and the communication unit contains transceivers through which the data and control signals are exchanged between node and base station. The control unit acts as the brain of the node, which will control tasks, interface between all these units and perform the memory operations. The power supply unit generally contains a small battery to power the entire node.

![Figure 2.1 WSN node block diagram](image)

A number of commercial and research products are available for a variety of wireless applications. Most of the commercial motes (nodes) use low power microcontrollers like MSP430 series from TI and ATMeg 128 series from Atmel as controllers, and IEEE 802.15.4 Zigbee compatible transceivers. These low power controllers are tailored for low power operation across range of embedded system applications, but are not necessarily well-suited to the event driven
behavior of WSN. Due to the constant evolution in the WSNs and its widespread application, flexibility in the functioning of WSN node is a key property [1].

The complete design flow of ultra low power WSN node controllers in the micro-task model was proposed in [1]. In the micro-task model, a WSN node controller is represented by a task flow graph in terms of tiny, independent control tasks [1, 3, 4] as shown in figure 2.2. The tasks can be an event sensing, MAC, routing, data processing, etc. A hardware realization of these specialized control tasks is called micro-task. A micro-task can be seen as a small datapath micro-architecture which contains finite state machines (FSM) and data path units along with ALU as shown in figure 2.3. The micro-task model of the WSN node controller is shown in figure 2.4. This controller architecture is based on event-centric concurrency model.

![Figure 2.2 Task flow graph](image)

The micro-task model of controller is comprised of micro-tasks, memory, IO interface for peripherals and a monitor which link all these units. The monitor is an FSM, based on the external / internal events; the monitor will enable / disable a particular micro-task. Two different memories are available, a local memory used specifically for a particular micro-task, and a global memory used to store a node-id, node address and important data stored by micro-tasks in case of local memory is shut down. In contrast to an instruction set processor, the control FSM of a micro-task controls a semi-custom datapath. This makes micro-task architecture much more compact and voids the need for instruction decoder and instruction memory. The main focus of this thesis work to study various reconfigurable, low
power architecture to implement control FSM of the micro-task as discussed in Chapter 3 and 4.

Figure 2.3 Structure of micro-task

Figure 2.4 WSN node architecture in micro-task model
2.2 FINITE STATE MACHINE

A finite state machine (FSM) is a mathematical model of sequential logic circuits. It is considered as an abstract machine which represents a sequential system based on its inputs and clock. The machine will be at one state out of a finite number of states at any given time. The state at which the machine is in at any given time is called current state and it can transit from one state to another based on input condition. In day to day life we see lots of systems which are working based on FSM principle e.g. vending machines, traffic lights, etc.

In control applications, finite state machines are classified into 1) Moore machine and 2) Mealy machine. Moore machine is a state based FSM, in which output of the FSM is decided, based on the current state. On the other hand Mealy machine is an input and state based FSM, in which output is a function of both present input and current state. Figures 2.5 and 2.6 shows the pictorial representation of the Moore and Mealy machines, respectively.

![Figure 2.5 Moore FSM](image)

![Figure 2.6 Mealy FSM](image)
2.3 LOW POWER DESIGN

Wireless sensor network nodes are generally powered using a small battery attached to it. In order to reduce WSN node size and to make it more mobile, battery size of WSN nodes is constantly reduced over the years. The need for low power design is highly motivated in WSN, to make it work for the long haul without replacing batteries. Due to the form factor of WSN nodes, it is not possible to place heat sinks to cool down the nodes. Power efficient architectures are a must to reduce the generation of heat and thermal dissipation in such designs. The need of the hour is low power techniques to improve energy efficiency without penalizing performance.

In CMOS circuits the total power consumption of a circuit is expressed as shown in Eq. 2.1. Dynamic power consumption is the sum of switching power ($P_{SW}$) and short circuit power ($P_{SC}$). Static power consumption ($P_{Static}$) is due to leakage current through the transistors. Low power design is very significant in sub-nano meter technologies due to the increased leakage current down the scale.

\begin{equation}
P_{Tot} = P_{SW} + P_{SC} + P_{Static}
\end{equation}

\begin{equation}
P_{Tot} = \alpha f_{clk} V_{dd}^2 C_{load} + V_{dd} I_{sc} + V_{dd} I_{leak}
\end{equation}

Various low power techniques are practiced to reduce each of those power components. Dynamic power can be reduced by shutting off clock signal, which in turn reduces the capacitance that is being charged and discharged at every clock cycle. Techniques like pipelining and interleaving can be used at the architectural level to reduce dynamic power. From Eq. 2.2 we can observe that by reducing supply voltage and clock frequency a large amount of power can be saved. The short circuit power is consumed when input signal is transiting near the logic threshold. During this time both pull-up and pull-down network are in the linear region. By balancing input and output transitions the short circuit power can be reduced. Static power consumption is due to following leakage currents, 1) reverse bias current, 2) sub-threshold leakage, 3) gate-oxide tunneling and 4) gate induced drain leakage. Leakage power remains constant, since it does not depend on input transitions and load capacitance. Techniques like power gating and multi VT transistor are used to reduce leakage power. In power gating, when the circuit or part of the circuit, is not active, supply voltage of the circuit is turned off by connecting logic to virtual ground. In multi VT technique, high threshold voltage
(HVT) transistors are used in non-critical paths which reduce leakage power. Low threshold voltage transistors are used in critical paths to reduce the latency.

Pass transistor logic styles are considered to be one of the best choices to implement high speed, low power designs. Compared to CMOS, PTL needs a few transistors and consumes less static power. In this thesis work, a reconfigurable FSM is implemented using PTL logic style to study performance, power and area consumption.

2.4 POWER GATING

In general in a WSN, as the nodes remain inactive for a long time, the static power dissipation due to leakage contributes more than dynamic power to the total power consumption of the node. Power gating is one of the various low power techniques to reduce leakage power. Power gating is an invasive approach in which an HVT PMOS sleep transistor is inserted between $V_{dd}$ and Virtual-$V_{dd}$ or an HVT NMOS sleep transistor is inserted between Gnd and Virtual-Gnd as shown in figure 2.7. The sleep transistor is sized with large gates such that there is no measurable IR drop across the transistor. The gate terminal of the sleep transistor is controlled by a sleep signal. When the circuit is inactive the sleep signal turns off the sleep transistor, which cuts off $V_{dd}$ to the circuit and eliminates leakage power dissipation.

![Power gating diagram](image)

Figure 2.7 Power gating

When a particular block of the circuit is turned off by the sleep transistor, the outputs of the power gated block remain floating and disturb always-on blocks. In
order to properly clamp the outputs of power gated blocks isolation cells are used. Generally, isolation cell can be AND or OR logic gate which will clamp the output of the power gated block to logic low or high respectively. In special circumstances retention registers are also used to retain the last known state of the outputs. Power gating can be done at various levels of granularity ranging from logic clusters (coarse grain) to every single library cell (fine grain) in the design. In this work, we have studied power gating granularity based on energy saving vs. design overheads.

2.5 PASS TRANSISTOR LOGIC

Pass transistor logic (PTL) is one of the logic families in integrated circuit design. In contrast to the CMOS logic, primary inputs drive gate terminal also source-drain terminals of the transistors in PTL. The main advantage of PTL network is that, any logic function can be realized using either NMOS or PMOS PTL networks, resulting in reduced number of transistors compared to CMOS style. Logic function with $n$ inputs needs only $n$ transistors to realize in PTL. Due to the increased demand for low-power VLSI, PTL network are the proper choice of logic style for implementing tree based structures like multiplexers.

The following is the variety of PTL flavors that are available: Complementary Pass-transistor Logic (CPL), Double Pass-transistor Logic (DPL), Energy Economized Pass-transistor Logic (EEPL), Swing Restored Pass-transistor Logic (SRPL), Push-pull Pass-transistor Logic (PPL), Lean Integration Pass-transistor Logic (LEAP) and Transmission Gates (TG). In this thesis work, both LEAP and TG form of pass transistor network are chosen to study and implement reconfigurable FSM, since both forms of PTL are single-rail logic styles [6, 10].
3 Embedded FPGA

3.1 Introduction

In order to achieve flexibility in node functionality, it is common to adopt reconfigurable architecture for WSN nodes. FPGA based architectures can be realized by using soft core controllers into commercial FPGAs or by interfacing discrete FPGA fabric to the standard controller. In FPGA based architectures area and power are the major overheads, as interconnect consumes more power and area than logic cells. Also the leakage power is dissipated by both used and unused parts of the FPGA. As per experiment conducted by Tuan and Lai [18] it has been found out that 56% of total leakage power is dissipated by unused parts of the FPGA (when 50% of CLBs are used). Contemporary WSN demands a longer battery life and micro size nodes for easy placement and maintenance-free operation for years together. Embedded FPGA is one of the solutions to achieve reconfigurability in WSN nodes. In this thesis work eFPGA is designed along with CAIRN team is used to study how power consumption can be reduced by applying various low power techniques.

3.2 Homogenous Embedded FPGA

Embedded FPGA (eFPGA) is a tiny island style, custom developed FPGA. The eFPGA is suitable for small or medium sized logic functions like micro-tasks. The eFPGA can be glued to microcontrollers to attain flexibility. The functionality of the micro-task can be changed using dynamically reconfigurable memory without disturbing the current task. The minimum number of CLBs in eFPGA guarantees flexibility at less area and minimum leakage power dissipation from unused parts of the FPGA.

The eFPGA is designed using a matrix of array elements (AE) as shown in the figure 3.1. Totally 576 array elements are placed in 24x24 matrix. Figure 3.2 shows the schematic of single array element. Each array element is made up of a configuration logic block (CLB), connection boxes CHAN-X and CHAN-Y and a switch-box (SB). In order to prevent signal deterioration, bi-directional buffers are placed in the non overlapping alternate array elements. All the array elements are connected through 5-channel programmable interconnections.
Figure 3.3 shows the structure of a CLB. Every CLB is made up of a 4-input look-up table (LUT), a D-Flipflop and a 2:1 multiplexer. The LUT contains 16 configuration bits to store the configuration of the FPGA. Figure 3.4 shows the schematic of a single configuration bit. Each configuration bit contains a latch to
hold the active configuration and a flipflop to dynamically reconfigure the configuration during run time without disturbing the current configuration.

![Configuration logic block (CLB)](image)

Figure 3.3 Configuration logic block (CLB)

![Structure of a configuration bit](image)

Figure 3.4 Structure of a configuration bit

To study the performance and resource utilization, eFPGA is designed and fabricated using 65nm CMOS technology. Figure 3.5 shows the layout of eFPGA. Some of the finite state machines used in the WSN node controller are mapped, placed and routed into eFPGA using Verilog-to-routing (VTR) tool [19]. The FSMs were chosen from the benchmark descriptions in SenseBench [16] that includes tasks such as arithmetic absolute value, 8-bit and 16-bit cyclic redundancy check and FIR filtering. Table 3.1 shows the energy and power consumed by eFPGA for different benchmark FSMs.

<table>
<thead>
<tr>
<th>FSM</th>
<th>No. of CLBs</th>
<th>$f_{clk} = 100$MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Power (mW)</td>
</tr>
<tr>
<td>abs</td>
<td>50</td>
<td>5.92</td>
</tr>
<tr>
<td>crc8</td>
<td>84</td>
<td>10.01</td>
</tr>
<tr>
<td>receiveData</td>
<td>94</td>
<td>11.22</td>
</tr>
<tr>
<td>crc16</td>
<td>143</td>
<td>18.16</td>
</tr>
<tr>
<td>firBasic</td>
<td>217</td>
<td>30.58</td>
</tr>
</tbody>
</table>

Table 3.1 Energy and power consumption of FSMs in eFPGA
3.3 CONCLUSION

The eFPGA is a general reconfigurable fabric used to attain flexibility in WSN nodes. From the experiments conducted in the eFPGA, it is evident that even though the size of eFPGA is small compared to most commercial FPGAs, the leakage power dissipated from the interconnects and the unused parts of an FPGA contributes a significant portion of the total leakage power dissipation. Further to reduce the leakage power we have designed a reconfigurable architecture for FSM implementation using power gating technique as discussed in chapter 4.

Figure 3.5 Layout of eFPGA
4 RECONFIGURABLE FSM

4.1 RECONFIGURABLE FSM

Reconfigurable FSM is a sequential logic circuit whose behavior can be configured across time. From figure 4.1 and 4.2, we can see that there are two combinational networks, say $F$ and $G$, which will compute the next state and output of an FSM respectively. At time instance $t$, let the $n$ primary inputs to the FSM be denoted by $X(t) = \{ x_0(t), x_1(t), ..., x_{n-1}(t) \}$, the $m$ outputs of FSM be denoted by $Y(t) = \{ y_0(t), y_1(t), ..., y_{m-1}(t) \}$ and $N$ state bits be denoted by $S(t) = \{ s_0(t), s_1(t), ..., s_{N-1}(t) \}$. In binary logic, all the inputs and outputs take values from the set (0,1); corresponding to $2^n, 2^m$ and $2^N$ possible patterns.

The next state of a Moore FSM can be represented as a function of primary inputs and present state as shown in Eq. (4.1). The output of the FSM is represented as shown in Eq. (4.2).

$$s_i(t+1) = f_i(x(t), s(t)) \quad i= 0,1,...,N-1$$ (4.1)
$$y_i(t) = g_i(s(t)) \quad i= 0,1,...,m-1$$ (4.2)

Similarly, Eq. (4.3) and Eq. (4.4) shows the representation of next state and output of Mealy FSM.

$$s_i(t+1) = f_i(x(t), s(t)) \quad i= 0,1,...,N-1$$ (4.3)
$$y_i(t) = g_i(x(t), s(t)) \quad i= 0,1,...,m-1$$ (4.4)

For any given $n$ inputs, there are $2^n$ unique Boolean functions possible, therefore for $n+N$ inputs of $f_i$, $2^{2^n+N}$ functions possible for a single state bit. For N state bits the total number of unique Boolean functions possible for next state combinational network F is $N * 2^{2^n+N}$. Similarly the total number of unique Boolean functions for output combinational network G is $m * 2^n$ or $m * 2^{2^n+N}$ depending on the type of FSM. For a FSM to be reconfigurable, it should be configured to support more than one set of Boolean functions $f_i$ and $g_i$ across time.
4.2 SHANNON’S EXPANSION

Shannon’s expansion is one of the methods to obtain the canonical sum of product (SOP) or product of sum (POS) of a logic function for any given truth table.

\[
\begin{array}{|c|c|c|}
\hline
a_1 & a_0 & y = f(a_1, a_0) \\
\hline
0 & 0 & 1 = f(0,0) \\
0 & 1 & 0 = f(0,1) \\
1 & 0 & 0 = f(1,0) \\
1 & 1 & 1 = f(1,1) \\
\hline
\end{array}
\]

Table 4.1 Truth table of X-NOR gate

From the above shown truth table of the X-NOR gate, the following SOP form can be obtained using Shannon’s expansion.

\[
f(a_1, a_0) = \overline{a_1} \cdot \overline{a_0} \cdot f(0,0) + \overline{a_1} \cdot a_0 \cdot f(0,1) + a_1 \cdot \overline{a_0} \cdot f(1,0) + a_1 \cdot a_0 \cdot f(1,1)
\]

(4.5)

In a similar way, the next state of the FSM in Eq. 4.1 can be rewritten as

\[
s_i(t+1) = f_i(x_0, x_1, \ldots, x_{n-1}, s_0, s_1, \ldots, s_{N-1})
\]

(4.6)

\[
s_i(t+1) = x_0 \cdot x_1 \cdot f_i(0,0, \ldots, x_{n-1}, s_0, s_1, \ldots, s_{N-1}) + \\
x_0 \cdot x_1 \cdot f_i(0,1, \ldots, x_{n-1}, s_0, s_1, \ldots, s_{N-1}) + \\
x_0 \cdot x_1 \cdot f_i(1,0, \ldots, x_{n-1}, s_0, s_1, \ldots, s_{N-1}) + \\
x_0 \cdot x_1 \cdot f_i(1,1, \ldots, x_{n-1}, s_0, s_1, \ldots, s_{N-1})
\]

(4.7)

and more generally as

\[
s_i(t+1) = \sum_{k=0}^{2^{n+N-k}-1} m_k \cdot f_i(n(m_k), \ldots, s_{N-1})
\]

(4.8)

Where \(k\) corresponds to the number of variables on which function \(f_i\) depends after Shannon’s decomposition. The minterms generated by first \(n+N-k\) input variables of the sequence \(x_i, \ldots, s_i(t+1)\) is denoted by \(m_k\).

4.3 LUT REALIZATION OF FSM

A reconfigurable FSM is realized using look-up tables and basic logic gates as shown in figure 4.3 [4]. In this method, the next state function of the FSM is decomposed such that \(f_i\) function depends only on \(k\) input variables, where \(k\) can be
4 or 6. As shown in the figure 4.3, 6 input variables on which \( f_i \) depends, acts like selection input for all the LUTs. The remaining \( n+N-6 \) variables are connected to the decoder input to generate \( 2^{(n+N-6)} \) prime implicants (\( m_0, m_1, \ldots, m_2^{(n+N-6)} \)). Similarly the output bit of an FSM can be realized using LUTs as shown in figure 4.4. Table 4.2 shows the resource requirement for a fully reconfigurable FSM with \( n \) primary inputs, \( N \) state bits and \( m \) outputs. When the sum of primary inputs and state register bits \( n+N < k \), the factor \( 2^{(n+N-k)} \) has to be replaced by 1. The components of a reconfigurable FSM are implemented using 65nm static CMOS library components.

<table>
<thead>
<tr>
<th>Resources</th>
<th>Next state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-LUT</td>
<td>( N \times 2^{(n+N-k)} )</td>
<td>( m \times 2^{(N-k)} )</td>
</tr>
<tr>
<td>Decoder</td>
<td>( (n+N-k) : 2^{(n+N-k)} )</td>
<td>( (n+N-k) : 2^{(n+N-k)} )</td>
</tr>
<tr>
<td>2- inp AND</td>
<td>( N \times 2^{(n+N-k)} )</td>
<td>( m \times 2^{(n+N-k)} )</td>
</tr>
<tr>
<td>( 2^{(n+N-k)} ) – OR</td>
<td>( N )</td>
<td>( N )</td>
</tr>
</tbody>
</table>

Table 4.2 Resource requirement for full reconfigurability
For a fully reconfigurable FSM architecture the complexity of logic increases exponentially as a function of number of state register bits and primary inputs. This increases the total area of the circuit and power consumption due to the leakage current. Power gating is considered to be one of the viable options to reduce dynamic and static power by connecting the supply rails of the circuit to virtual $V_{dd}$ using sleep transistors. In reconfigurable FSMs, power gating options are investigated at various granularities from coarse-grain to fine-grain level. The main challenge in power gating is the sizing of the sleep transistor. In active mode, IR drop across the sleep transistor can be reduced by increasing the gate width. But during the standby mode a wider transistor leaks more. This dilemma leads to careful design of sleep transistor based on the design goals.

In reconfigurable FSMs, properly sized PMOS header switches are used to power gate entire/part of the design. Figure 4.3 shows the coarse-grain power gating of reconfigurable FSM, in which the entire logic cluster for computing the single state bit is power gated by a header switch. Though coarse grain power gating is beneficial because of lesser area overhead in terms of minimum power
switches and isolation cells, it results in bigger inrush current and longer wake-up time.

In fine grain power gating, library cells with inbuilt power gating features have been used to design the circuit. In this case, smaller wake up time and inrush current is being achieved at the cost of more sleep transistors and isolation cells to handle multiple power domains, which increases the complexity of the FSM. Another challenge faced with fine grain power gating is that routing the sleep signal is more complex compared to coarse grain level.

Considering the energy savings and design overheads, an optimum power gating granularity has been achieved as shown in figure 4.4. Instead of using power switches for every library cell, fewer cells are grouped as a power island. This results in significant savings in terms of both area and energy.

![Figure 4.3 Coarse-grain power gated LUT architecture](image)

Figure 4.3 Coarse-grain power gated LUT architecture
Table 4.3 shows the energy per operation of a power gated reconfigurable FSM architecture for various benchmark FSMs, operated at different clock frequencies.

<table>
<thead>
<tr>
<th>FSM</th>
<th>No. of primary inputs</th>
<th>No. of states</th>
<th>Energy / operation (pJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abs</td>
<td>5</td>
<td>20</td>
<td>17.76</td>
</tr>
<tr>
<td>crc8</td>
<td>5</td>
<td>39</td>
<td>25.46</td>
</tr>
<tr>
<td>receiveData</td>
<td>5</td>
<td>51</td>
<td>32.23</td>
</tr>
<tr>
<td>crc16</td>
<td>6</td>
<td>70</td>
<td>29.38</td>
</tr>
<tr>
<td>firBasic</td>
<td>5</td>
<td>112</td>
<td>31.31</td>
</tr>
</tbody>
</table>

Table 4.3 Energy and power consumption of FSMs in power gated reconfigurable FSM

4.5 PASS TRANSISTOR BASED RECONFIGURABLE FSM

Pass transistor logic (PTL) network is the proper choice of logic style for implementing tree based structures like multiplexers. Pass transistor logic offers significant reduction in power dissipation and area by removing power switches and isolation cells used in power gating. In this thesis work, both LEAP and TG form of pass transistor network are chosen to study and implement reconfigurable FSM, since both forms of PTL are single-rail logic styles [6, 10].
4.5.1 LEAN INTEGRATION PASS TRANSISTOR LOGIC (LEAP)

LEAN integration Pass transistor logic is a single rail pass transistor logic. Any logic function can be represented by the NMOS pass transistor network as shown in the figure 4.5. The advantage of using only NMOS transistors is, only \( n \) transistors are required for \( n \) input logic function.

On the other hand, an NMOS transistor is effective at pulling down a node to Gnd, but poor at pulling node to \( V_{dd} \). Thus, when input node is high the output node charges only up to \( V_{dd} - V_{th} \). This results in strong ‘0’ and weak ‘1’ at output node and this becomes worse due to body effect.

To overcome the above mentioned problem, an additional swing restoration circuit is realized by a feedback pull-up PMOS transistor as shown in the figure 4.5. When the output node \( O \) is at 0V, \( \overline{O} \) is at \( V_{dd} \) and it turns off the feedback PMOS connected to the output node. When the output node is charged up to \( V_{dd} - V_{th} \) this potential is enough to switch the output of the inverter low, turning on the feedback PMOS and pulling the node \( O \) all the way to \( V_{dd} \). Since the level restorer is active only when the input is high, there is no static current path that can exist through level restorer and pass transistor.

Though the level restorer avoids weak ‘1’ at the output it costs extra area and increases the complexity of the circuit. When the input node makes a transition from high to low, the NMOS network tries to pull down the node \( O \) while the feedback PMOS pulls up the node \( O \) to \( V_{dd} \). In order to make the circuit function correctly, the NMOS network needs to be stronger than pull up PMOS to switch the output node. In order to make the NMOS network stronger, the circuit needs to be sized such that the voltage at node \( O \) drops below the logic threshold of the inverter, which is a function of the resistance of the NMOS network and that of the
feedback PMOS. Though proper sizing of transistor makes the circuit function correctly, but it makes the circuit ratioed.

4.5.2 TRANSMISSION GATES

Transmission gates or pass gates is another single rail pass transistor logic. In contrast to LEAP PTL, transmission gates use both NMOS and PMOS transistors as shown in figure 4.6. Due to the presence of both NMOS and PMOS transistors, this logic style consumes more transistors, $2n$ transistors compared to $n$ transistors in LEAP for any $n$ input logic.

The level restorer circuit is not required in transmission gates since PMOS and NMOS is capable of passing strong 1 and strong 0, respectively. In order to decouple gate inputs and outputs and to provide acceptable output driving capabilities, output inverters are used.

![Transmission Gates Diagram](image)

Figure 4.6 Transmission gates

4.6 LIBRARY COMPONENTS

In PTL based reconfigurable finite state machine architecture, the finite state machines are completely realized using the following PTL based custom designed set of library components.

4.6.1 LUT SELECTOR

The LUT selector is a multiplexer structure used to select one out of multiple configuration inputs connected to LUTs [6, 7, 8]. Figure 4.7 shows the structure of 3-input LUT. Select lines $s0$ to $s2$ and their complementary signals are
Figure 4.7 LUT decoder

connected to gate terminals of the transmission gates. Similarly, the $c_0$ to $c_7$ configuration bits are connected to source/ drain terminals of the transmission gates. Based on the select lines, one of those configuration bits will be passed on to the output terminal. In the reconfigurable FSM 6 input variables on which next state or output function depends after Shannon decomposition are connected to the select lines of all the LUTs. Based on the minterms one of the LUT outputs will be selected as the output or the state bit.

A LUT selector has been implemented in both LEAP and TG style to study power consumption and area. Table 4.4 displays the results of a few LUT selector configurations in both LEAP and TG.

<table>
<thead>
<tr>
<th>k-LUT</th>
<th>Dynamic Power (nW)</th>
<th>Leakage Power (nW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-LUT LEAP</td>
<td>3.19</td>
<td>0.0227</td>
</tr>
<tr>
<td>3-LUT TG</td>
<td>45</td>
<td>0.3380</td>
</tr>
<tr>
<td>4-LUT LEAP</td>
<td>4.4</td>
<td>0.0306</td>
</tr>
<tr>
<td>4-LUT TG</td>
<td>307</td>
<td>0.4540</td>
</tr>
<tr>
<td>6-LUT TG</td>
<td>1680</td>
<td>2.4200</td>
</tr>
</tbody>
</table>

Table 4.4 Power consumption of LUT decoders

The LEAP style 6-LUT selector is not implemented due to limitation in sizing of the level restorer to flip the output node correctly. From the above results it is
clear that LEAP style PTL is consuming less power than transmission gates. Due to the restriction on the minimum size of PMOS transistor for a particular design kit, the NMOS network of bigger k-LUT needs to be sized much bigger to have proper flip in the output node. This results in additional area compared to transmission gate PTL. Due to this most of the PTL library components of this work were designed using transmission gate PTL.

4.6.2 LOGIC GATES

As shown in figure 4.3, 2-input AND gates are used to select one of the LUT output based on minterms (output from decoder). In PTL library, 2-input AND gate is designed using transmission gates as shown in figure 4.8, where $a$ and $b$ are inputs to the gate and $y$ is the output. When $b$ is 1, $T_1$ is open, $T_0$ starts conducting and the potential of $a$ is passed on to output $y$. When $b$ is 0, $T_0$ is open, $T_1$ starts conducting, and connects output $y$ to 0.

![Figure 4.8 2-input AND gate](image)

Similarly, 2-input OR gate is constructed using transmission gates as shown in figure 4.9. OR gate can be designed by just changing the polarity of transmission gates in AND gate. When $b$ is 1, $T_0$ is open, $T_1$ starts conducting, and connects output $y$ to 1. When $b$ is 0 $T_1$ is open, $T_0$ starts conducting, and the potential at input $a$ is passed on to output $y$. 
4.6.3 DECODER

After Shannon’s decomposition, 6 inputs on which the next state or output function depends will act like the select lines of the LUT decoder. The remaining $n+N-6$ inputs are connected to decoder as shown in figure 4.3 to generate $2^{(n+N-6)}$ minterms which will be fed as one of the inputs to the 2-input AND gate.

In reconfigurable FSM, the size of the decoder depends on the number of primary inputs, the number of state-register bits and the number of select lines of the LUT decoder. Since the decoder of any size can be constructed using 1:2 and 2:4 decoder, the PTL library contain only basic 1:2 and 2:4 decoder designed using pass transistors. A Perl script is written to generate any decoder of size $n:2^n$ using 1:2 and 2:4 decoders. Figure 4.10 shows the pass transistor representation of 1:2 decoder.
In which \( x0 \) is the input and \( en_n \) is the active low enable signal. When \( en_n \) is high, the potential at the output nodes \( y0 \) and \( y1 \) will be grounded using transistors M4 and M5. Also a transmission gate is placed in between \( V_{dd} \) and logic will be turned off when \( en_n \) is high to prevent short path between the supply rails through M0 or M1 when \( x0 \) is low or high. When \( en_n \) is low and \( x0 \) is also low, M0 starts conducting and output \( y0 \) will be high at the same time as M3 is closed which will ground the output node \( y1 \). Similarly when \( x0 \) is high, transistors M2 and M1 starts conducting making output \( y1 \) high and \( y0 \) low.

As shown in figure 4.11, 2:4 decoder is constructed using two 1:2 decoders, where input \( x0 \) and its complementary signal are connected as enable signal of 1:2 decoders. When \( x1 \) is low, one of the outputs \( y0 \) and \( y1 \) will be asserted, based on the input \( a0 \) value. Similarly \( x1 \) is high, one of the output \( y2 \) and \( y3 \) will be asserted based on the input \( a0 \) value. In order to have enable option for 2:4 decoder, 4 NMOS transistors are connected to output nodes with gate terminal tied to the enabling signal \( en_n \). When \( en_n \) is high all the output nodes will be connected to ground. A transmission gate connected between \( V_{dd} \) and logic prevents the short between the supply rails when \( en_n \) is high.

Figure 4.12 shows an example of how 5:32 decoder can be constructed using the basic decoders available in the PTL library. The 5:32 decoder is constructed using eight 2:4 decoders in the last stage and seven 1:2 decoders in three stages.

The most significant bit of input (\( a4 \)) is connected to the first stage decoder along with active low enable signal (\( en_n \)). The complemented output of each stage will be given as enable signal for next stage decoders. The two least significant input bits (\( a0, a1 \)) are connected to the inputs of the last stage 2:4 decoders which will result in 32 output bits.

Since the basic decoders are designed using pass transistors, the output may have weak ‘0’ or weak ‘1’. The inverters used to complement the output at every stage will boost the signal strength, whereby signal deterioration is avoided.
Figure 4.11 2:4 Decoder

Figure 4.12 5:32 Decoder
4.7 CONCLUSION

In reconfigurable FSM, the complexity of routing is significantly reduced as the switches are localized to the input selector decoder. This results in better utilization of silicon area. The unused LUT logic is always power gated in contrast to the FPGA in which optimal location of sleep transistors cannot be decided due to configuration based mapping of resources. Any given FSM can be easily mapped to this architecture which leads to simple mapping tools. Power gating offers significant reduction in power dissipation at the extra cost of power switches and isolation cells. Pass transistors have been used to implement reconfigurable FSMs to study area and power consumption compared to static CMOS design.
5 Configuration Logic

5.1 Introduction

In the LUT realization of FSM, the contents of the LUT are stored in the configuration memory [12]. In this thesis work, scan chain style configuration memory is designed using 1) Bucket Brigade circuit, 2) Quasi-adiabatic flip flops, 3) Dual-context scan cell and 4) Dynamic scan cell. The main focus of this study is to compare the power consumption and area required for the above mentioned configuration memory types.

5.2 Bucket Brigade Circuit

Bucket Brigade was developed during late 60’s [13]. This circuit was mainly used to implement an analog delay in a variety of audio applications. This circuit works on the basic principle of charge transfer from one bucket (storage unit) to another. Figure 5.1 shows the scan chain structure of the bucket brigade circuit. The capacitor connecting gate and the drain terminal of a transistor is called bucket, basic storage cell. Transistors with DC bias are used to provide isolation and to improve charge transfer efficiency. The gate terminals of alternate buckets are connected to complementary clocks $\phi_1$ and $\phi_2$.

Let us assume that capacitor $c_1$ contains the signal sample. At the same time, bias voltage appears across $c_2$ and $c_3$. In order to transfer signal from $c_1$ to $c_2$, the gate potential of M4 is raised by asserting $\phi_1$ which turns off M2 and M6. Since the bias potential across $c_2$ is greater than the sample voltage across $c_1$, M3 starts conducting. Charge now flows from $c_2$ to $c_1$ until the voltage across $c_1$ is equal to the bias voltage, which results in the sample voltage getting stored in $c_2$. The same process is continued to move bits along the scan chain.

![Bucket brigade scan chain](image)

Figure 5.1 Bucket brigade scan chain
5.3 QUASI-ADIABATIC FLIP FLOPS

Adiabatic circuits are energy recovery circuits which use reversible logic to conserve energy. The dynamic power can be reduced by decreasing supply voltage, physical capacitance and switching activity. But there is a limitation in voltage scaling, since the energy dissipation due to leakage becomes a dominant factor at lesser supply voltage. Adiabatic circuits are the best candidates to address the above stated issue. In this work, we have tried to design a scan chain using quasi-adiabatic circuit flip flops, which can be used as a configuration memory for the reconfigurable FSM [14, 15].

![Figure 5.2 Adiabatic inverter (2N-2P)](image)

Figure 5.2 shows the design of adiabatic inverter circuit using 2N-2P style. Clocked power supply $pc_1$ is used to power up the circuit. The 2N-2P adiabatic circuit operates in four phases, which are 1) Evaluate phase: in this phase clocked supply rises from 0V allowing the circuit to evaluate, 2) Hold phase: in this phase clocked supply will be at a constant high logic, 3) Recovery phase: in this phase charge will be recovered through the conducting PMOS and 4) Wait phase: in this phase supply clock is at 0V.

The adiabatic D flip flop can be constructed by cascading two adiabatic inverters as shown figure 5.3. From the timing diagram, we can observe there is a clock delay between $in$ and $out2$. An adiabatic configuration memory can be constructed by cascading series of adiabatic flip flops.
5.4 DUAL CONTEXT SCAN CHAIN

Dual context scan cell is capable of holding two different configurations at a time [5]. Figure 5.4 shows the design of dual context scan cell. The configuration bits will be shifted through flip flop at every rising edge of the clock signal. Meanwhile, the latch can hold the previous configuration bit. In order to change the configuration of the reconfigurable FSM, enable signal of the latch will be asserted for single clock cycle to latch the flip flop output. The extra register eliminates reconfigurable latency, but it contributes to additional leakage power.

Figure 5.3 Adiabatic flip flop (2N-2P)

Figure 5.4 Dual context scan cell
5.5 DYNAMIC SCAN CHAIN

In order to reduce leakage power due to extra registers in the dual context scan chain, new dynamic scan chain circuit is proposed as shown in figure 5.5. The transmission gates and inverters are used to construct a scan chain, in which gate terminals of alternate transmission gates are connected to complementary clocks. The configuration bits given at scan_in input will be passed to the next scan element in a clock period. Once all the scan bits are shifted across the chain, enable signal of the latch will be asserted for half a clock period to latch the configuration bits. The outputs of the latches are connected to the LUT decoders. This method also gives the flexibility to load new configuration bits while the rest of the circuit running on different configuration.

Figure 5.5 Dynamic scan chain

Figure 5.6 shows an example of how configuration data will be shifted across the chain. Let us consider a 4-bit scan chain and four input data $a$, $b$, $c$ and $d$ are shifted through the scan chain. At the end of 4 clock cycles, the corresponding configuration data are available at respective nodes.
In order to compare the power consumption and area, the above mentioned configuration memories are tested under a common test setup. From the test results it is observed that, bucket brigade circuit and quasi-adiabatic scan chain are not able to shift data over the long chain without errors. The following table lists the power consumption and area of the dual-context scan chain and dynamic scan chain.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Dual-Context Scan chain</th>
<th>Dynamic Scan chain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan length</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Number of transistors</td>
<td>384</td>
<td>256</td>
</tr>
<tr>
<td>Power consumption</td>
<td>37.3 µW</td>
<td>10.51 µW</td>
</tr>
<tr>
<td>Energy / operation</td>
<td>3 pJ</td>
<td>0.58 pJ</td>
</tr>
</tbody>
</table>

Table 5.1 Energy and power consumption of scan chains

**Figure 5.6 Dynamic scan chain working**

5.6 COMPARISON OF VARIOUS CONFIGURATION MEMORIES

The following table lists the power consumption and area of the dual-context scan chain and dynamic scan chain.
From the table 5.1 is it evident that dynamic scan chain consumes less power than the dual-context scan chain. Since the dynamic scan chain is designed without flip-flop to hold the second configuration, it's necessary to make sure the configuration is passed through the scan chain at the right time to avoid bit error while latching the configuration. Pass transistor based reconfigurable FSM have been designed using dynamic scan chain along with other PTL library components. Chapter 6 discusses the experiments and results of static CMOS reconfigurable FSM and PTL based reconfigurable FSM with dynamic scan chain.
6 Experiments and Results

6.1 Spice Simulation

The finite state machines used in the WSN node controller are represented in VHDL or Verilog in order to study the performance, functionality and energy/power consumption. The FSMs were chosen from the benchmark descriptions in SenseBench [16] that includes tasks such as arithmetic absolute value, 8-bit and 16-bit cyclic redundancy check and FIR filtering. Figure 6.1 shows the process flow followed in PTL reconfigurable FSM.

![Figure 6.1 Process flow](image)

Altera’s Quartus II or Berkeley’s ABC synthesis tool is used to convert the HDL description of a FSM into Berkeley Logic Interchange Format (.blif) form which is the textual form of logic-level hierarchical circuit. Appendix A shows the .blif file of one of the benchmark (abs) FSM. The Berkeley SIS tool is used to extract state table format (.kiss2) from .blif as shown in appendix B.

State table describes the number of inputs, states, outputs and state transition of the FSM. A Perl script has been written to read the state table file and calculate
the number of PTL library components required to realize that particular FSM. The PTL library components described in chapter 5 are given to the Perl script in spice (.spi) format. The Perl script generates the spice netlist of the FSM by instantiating the library components based on the state table parameters. The next state and output values in the state table are extracted to create the bitstream of the FSM. The testbench shown in appendix E, is used to simulate the FSM netlist in the Eldo spice simulator.

### 6.2 RESULTS

All the above mentioned benchmark FSMs are realized using transmission gates based PTL reconfiguration FSM architecture with dynamic scan chain and simulated in Eldo spice. Table 6.1 shows the parameters of the benchmark FSMs and table 6.2 shows the resource consumption by benchmark FSMs.

<table>
<thead>
<tr>
<th>FSM</th>
<th>No. of States</th>
<th>No. of Inputs</th>
<th>No. of Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>20</td>
<td>5</td>
<td>33</td>
</tr>
<tr>
<td>crc8</td>
<td>39</td>
<td>5</td>
<td>37</td>
</tr>
<tr>
<td>receiveData</td>
<td>51</td>
<td>5</td>
<td>37</td>
</tr>
<tr>
<td>crc16</td>
<td>70</td>
<td>6</td>
<td>39</td>
</tr>
<tr>
<td>firBasic</td>
<td>112</td>
<td>5</td>
<td>40</td>
</tr>
</tbody>
</table>

Table 6.1 FSM configuration

<table>
<thead>
<tr>
<th>FSM</th>
<th>No. of 6-LUTs</th>
<th>No. of 2-AND gates</th>
<th>No. of 2-OR gates</th>
<th>No. of 2:4 decoders</th>
<th>No. of 1:2 decoders</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>16</td>
<td>16</td>
<td>15</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>crc8</td>
<td>32</td>
<td>32</td>
<td>31</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>receiveData</td>
<td>32</td>
<td>32</td>
<td>31</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>crc16</td>
<td>128</td>
<td>128</td>
<td>127</td>
<td>32</td>
<td>31</td>
</tr>
<tr>
<td>firBasic</td>
<td>64</td>
<td>64</td>
<td>63</td>
<td>16</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 6.2 Resource consumption by FSMs

Figure 6.2 shows the simulation waveforms of inputs, state bits and the clock signal of the ABS FSM. Figure 6.3 is the zoomed picture showing state changes based on the input signals.
Figure 6.2 abs FSM output waveform
Figure 6.3 abs FSM state changes
<table>
<thead>
<tr>
<th>FSM</th>
<th>Energy / operation at $f_{clk} = 100$ MHz (pJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PTL reconfigurable FSM</td>
</tr>
<tr>
<td>abs</td>
<td>2.936</td>
</tr>
<tr>
<td>crc8</td>
<td>6.985</td>
</tr>
<tr>
<td>receiveData</td>
<td>6.987</td>
</tr>
<tr>
<td>crc16</td>
<td>32.59</td>
</tr>
<tr>
<td>firBasic</td>
<td>15.38</td>
</tr>
</tbody>
</table>

Table 6.3 Energy consumption by benchmark FSMs in PTL and power gated architectures

Table 6.3 shows the energy consumption per operation by benchmark FSMs in PTL based reconfigurable FSM and power gated reconfigurable FSM discussed in the chapter 4. From the table 6.3 we can observe that, PTL based reconfigurable FSM consumes less energy compared to power gated reconfigurable FSM for the first three benchmark FSMs. For the bigger FSMs like crc16 and firBasic the PTL reconfigurable FSM consumes more energy than the power gated reconfigurable FSM. When the complexity of the system increases the leakage through sneak paths in PTL reconfigurable also increases. Therefore for bigger systems power gating method offers better performance in terms of power /energy.
7 Conclusion

In this thesis work, various reconfigurable architectures are studied and developed to achieve low power and flexibility in WSN node controllers. Initially the tiny island style embedded FPGA is designed to hold the functionality of the WSN node. Using eFPGA the functionality of WSN node can be dynamically reconfigured across time. In the second method, the functionality of microtask is represented as a finite state machine. Using Shannon’s decomposition, the next state and output of finite state machines are realized using LUT based reconfigurable architecture. The reconfigurable architecture has been power gated in various levels of granularity based on energy and design overhead.

In the third method, pass transistors are used to implement reconfigurable FSM architecture discussed in the second method. PTL has been considered as a choice, since tree based logic can be better implemented using pass transistor logic with lesser number of transistors and lesser power consumption. In this method, a library of basic components like LUT decoder, AND gate, OR gate and decoders have been designed using pass transistor logic. Any given FSM is converted to the state table format called .kiss2 using the Berkeley SIS tool. A software code written in Perl reads the state table file and generates the spice netlist of the FSM using PTL library components.

In order to test and compare the various reconfigurable architectures, the benchmarks FSM was chosen from the benchmark descriptions in SenseBench. It is observed from the test results that LUT based reconfigurable FSM consumes less power and area compared to eFPGA. Secondly, pass transistor based implementation of reconfigurable FSM can be a better choice when the size of the FSM is smaller. For bigger FSMs pass transistor based realization is not efficient enough when compared with power gated reconfigurable FSM design. This is because when the complexity of the system increases the leakage through sneak paths in PTL reconfigurable also increases. Therefore for bigger systems power gating method offers better performance in terms of power /energy.
7.1 FUTURE WORK

Layout and cell characterization of PTL library components can be done to create the layout of reconfigurable FSM which can be further used for placement and routing.

In order to reduce leakage power consumption further, reconfigurable architecture can be implemented on an intrinsically low leakage technology like FD-SOI (Fully depleted silicon on insulator) which also gives flexibility to shrink the size of transistors.
8 Bibliography


Berkeley Logic Interchange Fromat (BLIF) is a textual description of logic level hierarchical circuit. Altera’s Quartus II tool is used to convert .vhd or .v file to .blif.

To generate .blif file add the following statement into .qsf file

```
Set_global_assignment -name INI_VARS "no_add_opts = on;
opt_dont_use_mac = on; dump_blif_before_optimize = on"
```

Then execute the project in quartus

```
Quartus_map <project_name>
```

Quartus will generate .blif as shown below.

```
.model abs_fsm
.inputs clk abs_enable lt_GT0 lt_GT1 lt_GT2
.outputs rdSrcAdr rdDstAdr rdDstW rfInSel0 rfInSel1 aluOp0 aluOp1 aluOp2 cIn
op1Sel0 op1Sel1 op2Sel0 op2Sel1 ioPortW ioPortAdr[0] ioPortAdr[1]
abs_event0 abs_event1 gVMASe10 gVMASe11 gVMDSel gVMDirAdr[0] gVMDirAdr[1]
gVMDirAdr[2] gVMDirAdr[3] gVMW

.latch n77 g1 re clk 0
.latch n82 g2 re clk 0
.latch n87 g3 re clk 0
.latch n92 g5 re clk 0
.latch n97 g6 re clk 0
.latch n102 g7 re clk 0
.latch n107 g11 re clk 0
.latch n112 g12 re clk 0
.latch n117 g13 re clk 0
.latch n122 g14 re clk 0
.latch n127 g15 re clk 0
.latch n132 g16 re clk 0
.latch n137 g21 re clk 0
.latch n142 g24 re clk 0
.latch n147 g34 re clk 0
.latch n151 g36 re clk 0
.latch n156 g37 re clk 0
.latch n161 g38 re clk 0
.latch n166 g42 re clk 0
.latch n171 g45 re clk 0
.names g1 g3 g7 n101
000 1
.names abs_enable g16 n77
11 1
.names abs_enable g6 n82
11 1
```
.names abs_enable g2 n87
11 1
.names abs_enable g13 n92
11 1
.names abs_enable g21 n97
11 1
.names abs_enable g42 n102
11 1
.names abs_enable g24 n107
11 1
.names abs_enable g11 n112
11 1
.names abs_enable g1 n117
11 1
.names abs_enable g3 n113 n122
111 1
.names lt_GT0 lt_GT1 lt_GT2 n113
011 1
.names abs_enable g3 n113 n127
110 1
.names abs_enable g45 n132
11 1
.names abs_enable g5 n137
11 1
.names abs_enable g34 n142
11 1
.names abs_enable g38 n147
11 1
.names abs_enable g12 n156
11 1
.names abs_enable g7 n161
11 1
.names abs_enable g14 g15 n166
11- 1
1-1 1
.names abs_enable g36 g37 n171
10- 1
1-1 1
.names g42 rdSrcAdr
1 1
.names g1 g2 g3 rdDstAdr
000 0
.names g5 g6 n101 rdDstW
001 0
.names g3 g6 rfInSel0
00 0
.names g1 g3 g7 rfInSel1
000 0
.names aluOp0
0
.names g3 aluOp1
1 1
.names aluOp2
0
.names cIn 0
.names op1Sel0 0
.names op1Sel1 0
.names op2Sel0 0
.names op2Sel1 0
.names ioPortW 0
.names ioPortAdr[0] 0
.names ioPortAdr[1] 0
.names ioPortAdr[2] 0
.names ioPortAdr[3] 0
.names ioPortAdr[4] 0
.names ioPortAdr[5] 0
.names ioDSel0 0
.names ioDSell 0
.names immRAdr 0
.names g11 abs_event0 1 1
.names g12 abs_event1 1 1
.names gVMASel0 0
.names g13 gVMASell 1 1
.names gVMDSel 0
.names gVMDirAdr[0] 0
.names gVMDirAdr[1] 0
.names gVMDirAdr[2] 0
.names gVMDirAdr[3] 0
.names gVMW 0
.names abs_enable n151 1 1
.end
**APPENDIX – B  KISS2 FILE**

KISS2 is a state table representation of finite state machines. Berkeley SIS tool is used to extract .kiss2 from .blif. the following are the commands to extract below shown .kiss2 file [11].

```
sis>read_blif absfull.blif
sis>stg_extract -a
sis>write_kiss absfull.kiss2
```

SIS will generate .kiss2 as shown below.

```plaintext
.i 5      # No. of Inputs
.o 33     # NO. of Outputs
.p 43     # No. of products
.s 20     # No. of States
.r 00000

#Input Current_state Next_state Output

-1--- 00000 00001 000000000000000000000000
-0--- 00000 00000 000000000000000000000000
-1--- 00001 00010 000000000000000000000000
-0--- 00001 00000 000000000000000000000000
-0--- 00010 00000 000000000000000000000000
-1--- 00010 00011 000000000000000000000000
-0--- 00011 00000 011010000000000000000000
-1--- 00011 00100 011010000000000000000000
-1--- 00010 00101 000000000000000000000000
-1--- 00100 00110 001000000000000000000000
-1--- 00100 00000 010000000000000000000000
-0--- 00100 01001 010000000000000000000000
-1--- 00101 00110 010000000000000000000000
-0--- 00110 00000 000000000000000000000000
-1--- 00110 01110 001010000000000000000000
-0--- 00111 00000 011000000000000000000000
-1--- 00111 01000 011000000000000000000000
-0--- 01000 00000 010000000000000000000000
-1--- 01000 01001 010000000000000000000000
-0--- 01000 00000 010000000000000000000000
-1--- 01000 01001 010000000000000000000000
-0--- 01000 00000 010000000000000000000000
-1--- 01000 01001 100000000000000000000000
-0--- 01100 00000 100000000000000000000000
-1--- 01100 01110 001000000000000000000000
-0--- 01100 00000 010100000000000000000000
-1--- 01110 01111 000000000000000000000000
-1--- 01110 01111 000000000000000000000000
```
-0--- 01110 00000 000000000000000000
-0--- 01111 00000 000000000000000000
-1--- 01111 10000 000000000000000000
-1--- 10000 10001 000000000000000000
-0--- 10000 00000 000000000000000000
-0--- 10001 00000 000000001000000000
-1--- 10001 10010 0000000100000000
-1--- 10010 10011 0000000010000000
-0--- 10010 00000 000000001000000000
-0--- 10011 00000 000000001000000000
-1--- 10011 00001 000000001000000000
-0--- 01011 00000 000000001000000000
-1--- 01011 01100 000000001000000000
#!/usr/bin/perl -w

############################################################################
# CODE TO EXTRACT NO. OF PRIMARY INPUTS, OUTPUTS AND STATE VECTOR FROM KISS2
# TO CREATE SPICE NETLIST FOR ANY GIVEN FSM IN KISS2 FORMAT
# CREATED BY : RENGA RAJAN RAGAVAN
# DATE       : 29-05-2013
# VERSION    : 1.0
############################################################################

use strict;

our $file = $argv[$#argv];
open(ourfile,"$file");    
open(outfile,'>FSM.SPI');

our $cnt = 1;  # TO COUNT FIRST 5 LINES TO EXTRACT INFO OF FSM
our $cnto = 1; # CHARACTER COUNTER TO READ PROPER INFO FROM KISS2
our $char;    # VARIABLE TO READ CHARACTER FROM KISS2
our $tmp = 0;  # TEMP COUNTER TO HOLD THE POSITION OF PRIMARY INPUTS
our $tmp1 = 0; # TEMP COUNTER TO HOLD THE POSITION OF PRIMARY OUTPUTS
our $tmp2 = 0; # TEMP COUNTER TO HOLD THE POSITION OF STATE VECTOR
our $il = 0;   # NO. OF CHARACTER OF PRIMARY INPUT
our $ol = 0;   # NO. OF CHARACTER OF PRIMARY OUTPUT
our $sl = 0;   # NO. OF CHARACTER OF STATE VECTOR
our $pi = 0;   # PRIMARY INPUTS
our $po = 0;   # PRIMARY OUTPUTS
our @chars;    # ARRAY TO HOLD CHARACTERS READ FROM KISS2
our @inp;     # ARRAY OF INPUT CHARACTERS
our @out;     # ARRAY OF OUTPUT CHARACTERS
our $i;       # ITERATION VARIABLE
our $j;       # ITERATION VARIABLE
our $k;       # ITERATION VARIABLE
our $l;       # ITERATION VARIABLE
our $fslname = "TESTFSM";  # OUTPUT SUBCIRCUIT NAME
our $lut6;    # NO. OF 6-LUT REQUIRED
our $and2;    # NO. OF 2-AND REQUIRED
our $od;      # ONE DETECTOR FOR OR GATE
our $od1;     # ONE DETECTOR FOR OR GATE
our $s;       # TEMPORARY VARIABLE
our $s1;      # TEMPORARY VARIABLE
our $t = 0;   # TEMPORARY VARIABLE

while (<ourfile>) {  
  chomp;
  if ($cnt <= 5) {    
    $char = 

OUR $T1 = 0;  # TEMPORARY VARIABLE
OUR $T2 = 0;  # TEMPORARY VARIABLE
OUR $T3 = " ";  # TEMPORARY VARIABLE
OUR $T4;      # TEMPORARY VARIABLE
OUR $P;       # TEMPORARY VARIABLE

###################################################
# READ KISS2 FILE AND EXTRACT NO. PRIMARY INPUTS, OUTPUTS AND STATE VECTOR SIZE
###################################################

WHILE (<OURFILE>)
{
    @CHARS = SPLIT // ;
    IF ($CNT <= 5)
    {
        FOR $CHAR(@CHARS)
        {
            IF ($CHAR EQ "I")
            {
                $TMP = $CNTO + 2;
            }
            IF ($CNTO == $TMP)
            {
                IF ($CHAR NE "B")
                {
                    $TMP ++;
                    $INP[$IL] = $CHAR;
                    $IL ++;
                }
                ELSE
                {
                    $TMP = 0;
                }
            }
            IF ($CHAR EQ "O")
            {
                $TMP1 = $CNTO + 2;
            }
            IF ($CNTO == $TMP1)
            {
                IF ($CHAR NE "B")
                {
                    $TMP1 ++;
                    $OUT[$OL] = $CHAR;
                    $OL ++;
                }
                ELSE
                {
                    $TMP1 = 0;
                }
            }
        }
    }
IF ($CHAR EQ "R")
{
    $TMP2 = $CNTO + 2;
}
IF ($CNTO == $TMP2)
{
    IF ($CHAR NE "B")
    {
        $TMP2 ++;
        $SL ++;
    }
    ELSE
    {
        $TMP2 = 0;
    }
}
$CNTO ++;
}$CNT ++;
}
CLOSE (OURFILE);

PRINT "IL $IL OL $OL \n";
# CONVERING STRING TO DECIMAL
FOR($I = 0;$I < $IL;$I++)
{
    IF ($IL > 1)
    {
        $PI = $PI + ($INP[$I] * (10**($IL-1-$I)));
    }
    ELSE
    {
        $PI = $INP[0];
    }
}
FOR($I = 0;$I < $OL;$I++)
{
    IF ($OL > 1)
    {
        $PO = $PO + ($OUT[$I] * (10**($OL-1-$I)));
    }
    ELSE
    {
        $PO = $OUT[0];
    }
}
PRINT " \n\n";
PRINT "PRIMARY INPUT(S) = $PI \n";
PRINT "PRIMARY OUTPUT(S) = $PO \n";
PRINT "STATE VECTORS = $SL \n";
### RESOURCE CALCULATION

$LUT6 = (2^{(SPI+SL-6)});
$AND2 = (2^{(SPI+SL-6)});
$OR2 = (2^{(SPI+SL-6)})-1;
$DE24 = (2^{(SPI + SL -8)});
$DE12 = (2^{(SPI + SL -8)-1});

PRINT " \N \N";
PRINT "***** RESOURCE REQUIREMENT ***** \N";
PRINT "NO. OF 6-LUTS = $LUT6 \N" ;
PRINT "NO. OF 2-AND = $AND2 \N";
PRINT "NO. OF 2-OR = $OR2 \N";
PRINT "NO. OF 2:4 DECODERS = $DE24 \N" ;
PRINT "NO. OF 1:2 DECODERS = $DE12 \N \N \N";

$TMP = $LUT6;
$J = 0;
$OD = 0;

WHILE ($TMP > 1)
{
    $QU[$J] = INT $TMP / 2;
    $RE[$J] = $TMP % 2;
    IF ($OD == 0 & $RE[$J] == 1)
    { $OD = 1; }
    ELSIF ($OD == 1 & $RE[$J] == 1)
    { $QU[$J] = $QU[$J]+1;
        $RE[$J] = 2;
        $OD = 2;
    }
    $TMP = $QU[$J];
    $J++;

}

PRINT "QUO @QU \NREM @RE \NJ $J \N";

### SUBCIRCUIT CREATION

# SUBCIRCUIT CREATION
PRINT OUTFILE ".SUBCKT $FSMNAME " ;
FOR($I = 0; $I < $PI; $I++)
{
    PRINT OUTFILE "I$I ";  # INPUT PORT NAMES
}
#FOR($I = 0; $I < $PO; $I++)
#
# PRINT OUTFILE "O$I " ;  # OUTPUT PORT NAMES
#
FOR($I = 0; $I < $SL; $I++)
{
    PRINT OUTFILE "S$I ";  # STATE NAMES
}
PRINT OUTFILE "VDD GND INH_VDD INH_VDDS INH_GND INH_GNDS CLK CLKB EN EN_AL
\n";

$TMP = 0;
$P = " ";
$S1 = 0;

IF ($SL >= 6)
{
    $S = $SL - 6;
    FOR($I = $S; $I < $SL; $I++)
    {
        $P = "$S"."$I." "$P;
    }
    $S1 = $SL - 6 - 1;
}
ELSE
{
    $S = 6 - $SL;
    $S = $PI - $S;
    FOR($I = $S; $I < $PI; $I++)
    {
        $P = "$I"."$I." "$P;
    }
    FOR($I = 0; $I < $SL; $I++)
    {
        $P = "$S"."$I." "$P;
    }
    $S1 = -1;
}

#print "$S $S $P $P \n";

# BASIC BLOCK CONNECTIONS

FOR ($I = 0; $I < $LUT6; $I++)
{
IF ($I == 0)
{
    PRINT OUTFILE "XI$TM W_DEC_O_$I W_AND_O_$I $P CLK CLKB DIN EN GND W_OUT_$I VDD INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK \n";
    $TMP++;}
ELSIF ($I == $LUT6 -1)
{
    $T = $I -1;
    PRINT OUTFILE "XI$TM W_DEC_O_$I W_AND_O_$I $P CLK CLKB W_OUT_$T EN GND LOUT VDD INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK \n";
    $TMP++;}
ELSE
{
    $T = $I -1;
    PRINT OUTFILE "XI$TM W_DEC_O_$I W_AND_O_$I $P CLK CLKB W_OUT_$T EN GND W_OUT_$I VDD INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK \n";
    $TMP++;}
}

$OD1 = 0;
$L = 0;

# PASS OR CONNECTIONS
FOR ($K = 0; $K < $J; $K++)
{
    $TMP1 = $QU[$K];
    IF ($RE[$K] == 2)
    {
        $TMP1 = $TMP1 - 1;
    }
    #PRINT "$K $K TMP1 $TMP1\n";
    IF ($K == 0)
    {
        FOR ($I = 0; $I < $TMP1*2; $I++)
        {
            $T = $I + 1;
            $T1 = $I / 2;
            #PRINT "$K $K I $I T $T T1 $T1 TMP1 $TMP1\n";
            PRINT OUTFILE "XI$TMP W_AND_O_$I W_AND_O_$T GND VDD W_OR_$K-$T1 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR \n";
            $TMP++;$I++;
        }
    }
}
ELSIF ($K > 0)
{
    FOR ($I = 0; $I < $TMP1*2; $I++)
    {
        $T2 = $K - 1;
        $T1 = $I / 2;
        $T = $I + 1;
        #PRINT "K $K I $I T $T T1 $T1 T2 $T2 TMP1 $TMP1\n";
        PRINT OUTFILE "XISTMP W_OR_$T2-$I W_OR_$T2-$T GND VDD W_OR_$K-$T1 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR \n"
        $TMP++;
        $I++;
        IF ($TMP1 == 1)
        {
            PRINT OUTFILE "XISTMP CLK W_OR_$K-$T1 EN S$L INH_GND INH_GNDS INH_VDD INH_VDDS HS65_LH_DFPHQX4 \n"
            $TMP++;
        }
    }
}

IF ($RE[$K] == 1 && $OD1 == 0 && $TMP1 != 1)
{
    $S = $TMP1*2;
    $OD1 = 1;
    #PRINT "$S $S \n"
}

IF ($K == 0 && $RE[$K] == 1)
{
    $T3 = "W_AND_O"
}
ELSIF ($K > 0 && $RE[$K] == 1)
{
    $T2 = $K - 1;
    $T3 = "W_OR_S".$T2 
}

IF ($RE[$K] == 2 && $OD1 == 1)
{
    $T2 = $K - 1;
    $T1 = $I / 2;
    #PRINT "K $K I $I T $T T1 $T1 T2 $T2 T3 $T3 S $S OD1 $OD1\n"
    PRINT OUTFILE "XISTMP W_OR_$T2-$I W_OR_$T2-$T GND VDD W_OR_$K-$T1 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR \n"
    $T3 = " ";
    $OD1 = 2;
    $TMP++;
}

IF ($TMP1 == 1 && $RE[$K] == 1)
{
    $T2 = $K - 1;
$T = $T1 + 1;
#PRINT "$K $S $I $T $T1 $T2 $T2 $OD1 $OD1\n";
PRINT OUTFILE "XI$TMP W_OR_$T2-$S I W_OR_$S-$T1 GND VDD W_OR_$S-$T1 OD1"
INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR \n";
$TMP++;
IF ($TMP1 == 1)
{
    PRINT OUTFILE "XI$TMP CL W_OR_$S-$T1 EN S$L INH_GND
INH_GNDS INH_VDD INH_VDDS HS65_LH_DFPHQX4 \n";
    $TMP++;
}

IF ($TMP1 == 1 && $OD1 == 1)
{
    $T = $T1 + 1;
    #PRINT "$K $S $I $T $T1 $T2 $T2 $T3 $T3 $S $S $S $OD1 $OD1\n";
    PRINT OUTFILE "XI$TMP $T3-$S W_OR_$S-$T1 GND VDD W_OR_$S-$T1 OD1"
INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR \n";
    $TMP++;
    IF ($TMP1 == 1)
    {
        PRINT OUTFILE "XI$TMP CL W_OR_$S-$T1 EN S$L INH_GND
INH_GNDS INH_VDD INH_VDDS HS65_LH_DFPHQX4 \n";
        $TMP++;
    }
}

# DECODER CONNECTIONS

$TMP1 = $DE24;
$J = 0;
@QU = 0;
WHILE ($TMP1 > 1)
{
    $QU[$J] = INT $TMP1 / 2;
    $TMP1 = $QU[$J];
    $J++;
}

#PRINT "$TMP1 $TMP1 $J $S $QU \n";

FOR ($K = $J-1; $K >= 0; $K--)
{
    $TMP1 = $QU[$K];
    $T4 = 0;
    FOR ($I = 0; $I < $TMP1*2; $I++)
    {
        $T = $K + 2;
        $T1 = $K + 1;
        $T2 = $I + 1;
#PRINT "K $K I $I T $T T1 $T1 T2 $T2 TMP1 $TMP1\n";
IF ($S1 >= 0)
{
    IF ($K == $J - 1)
    {
        PRINT OUTFILE "XI$TMP S$S1 EN_AL GND VDD W_Y_$K-$I W_Y_$K-$T2 DECODER_1_2 \n";  # EN_AL FOR FIRST 1:2 DECODER
    }
    ELSE
    {
        PRINT OUTFILE "XI$TMP S$S1 W_EN_$T1-$T4 GND VDD W_Y_$K-$I W_Y_$K-$T2 DECODER_1_2 \n";
    }
}
ELSE
{
    IF ($K == $J - 1)
    {
        PRINT OUTFILE "XI$TMP I$T EN_AL GND VDD W_Y_$K-$I W_Y_$K-$T2 DECODER_1_2 \n";  # EN_AL FOR FIRST 1:2 DECODER
    }
    ELSE
    {
        PRINT OUTFILE "XI$TMP I$T W_EN_$T1-$T4 GND VDD W_Y_$K-$I W_Y_$K-$T2 DECODER_1_2 \n";
    }
}
$TMP++;
PRINT OUTFILE "XI$TMP W_Y_$K-$I W_EN_$K-$I INH_GND INH_GNDS INH_VDD INH_VDDS HS65_LL_IVX2 \n";
$TMP++;
PRINT OUTFILE "XI$TMP W_Y_$K-$ST2 W_EN_$K-$ST2 INH_GND INH_GNDS INH_VDD INH_VDDS HS65_LL_IVX2 \n";
$TMP++;
$I++;
$T4++;
}
$S1--;
}

$S = 0;
$K++;
$T4 = 0;
#PRINT "/N DEC24 $DE24 /N";
FOR ($I = 0;$I < $DE24;$I++)
{
    $T = $T4 + 1;
    $T1 = $T4 + 2;
    $T2 = $T4 + 3;
    IF ($PI >= 2)
{ 
    PRINT OUTFILE "XI$TMP I0 I1 W_EN_$K-$S GND VDD W_DEC_O_$T4 W_DEC_O_$T W_DEC_O_$T1 W_DEC_O_$T2 INH_GND INH_GNDS INH_VDD INH_VDDS DECODER_2_4 \n";
    }
    ELSE
    {
    PRINT OUTFILE "XI$TMP I0 S0 W_EN_$K-$S GND VDD W_DEC_O_$T4 W_DEC_O_$T W_DEC_O_$T1 W_DEC_O_$T2 INH_GND INH_GNDS INH_VDD INH_VDDS DECODER_2_4 \n";
    }
    $TMP++; 
    $S++; 
    $T4 = $T4 + 4;
}

CLOSE OUTFILE;
** GENERATED FOR: HSPICE

.include "Basic_block.spi"
.include "Models.spi"

.temp 25.0
.option
+ ingold=2
+ parhier=local
+ pstran
.ramp dc 0.1

** LIBRARY NAME: RECOFSM
** CELL NAME: ABS
** VIEW NAME: SCHEMATIC

.subckt absfull i0 i1 i2 i3 i4 s0 s1 s2 s3 s4 din l0 lout vdd gnd inh_vdd inh_vdds inh_gnd inh_gnds clk clkb en en_al

x10 w_dec_o_0 w_and_o_0 s0 i4 i3 i2 i1 i0 clk clkb din en gnd l0 vdd inh_gnd
inh_gnds inh_vdd inh_vdds basic_block
x11 w_dec_o_1 w_and_o_1 s0 i4 i3 i2 i1 i0 clk clkb l0 en gnd l1 vdd inh_gnd
inh_gnds inh_vdd inh_vdds basic_block
x12 w_dec_o_2 w_and_o_2 s0 i4 i3 i2 i1 i0 clk clkb l1 en gnd l2 vdd inh_gnd
inh_gnds inh_vdd inh_vdds basic_block
x13 w_dec_o_3 w_and_o_3 s0 i4 i3 i2 i1 i0 clk clkb l2 en gnd l3 vdd inh_gnd
inh_gnds inh_vdd inh_vdds basic_block
x14 w_dec_o_4 w_and_o_4 s0 i4 i3 i2 i1 i0 clk clkb l3 en gnd l4 vdd inh_gnd
inh_gnds inh_vdd inh_vdds basic_block
x15 w_dec_o_5 w_and_o_5 s0 i4 i3 i2 i1 i0 clk clkb l4 en gnd l5 vdd inh_gnd
inh_gnds inh_vdd inh_vdds basic_block
x16 w_dec_o_6 w_and_o_6 s0 i4 i3 i2 i1 i0 clk clkb l5 en gnd l6 vdd inh_gnd
inh_gnds inh_vdd inh_vdds basic_block
x17 w_dec_o_7 w_and_o_7 s0 i4 i3 i2 i1 i0 clk clkb l6 en gnd l7 vdd inh_gnd
inh_gnds inh_vdd inh_vdds basic_block
x18 w_dec_o_8 w_and_o_8 s0 i4 i3 i2 i1 i0 clk clkb l7 en gnd l8 vdd inh_gnd
inh_gnds inh_vdd inh_vdds basic_block
x19 w_dec_o_9 w_and_o_9 s0 i4 i3 i2 i1 i0 clk clkb l8 en gnd l9 vdd inh_gnd
inh_gnds inh_vdd inh_vdds basic_block
x110 w_dec_o_10 w_and_o_10 s0 i4 i3 i2 i1 i0 clk clkb l9 en gnd l10 vdd
inh_gnd inh_gnds inh_vdd inh_vdds basic_block
x111 w_dec_o_11 w_and_o_11 s0 i4 i3 i2 i1 i0 clk clkb l10 en gnd l11 vdd
inh_gnd inh_gnds inh_vdd inh_vdds basic_block
x112 w_dec_o_12 w_and_o_12 s0 i4 i3 i2 i1 i0 clk clkb l11 en gnd l12 vdd
inh_gnd inh_gnds inh_vdd inh_vdds basic_block
x113 w_dec_o_13 w_and_o_13 s0 i4 i3 i2 i1 i0 clk clkb l12 en gnd l13 vdd
inh_gnd inh_gnds inh_vdd inh_vdds basic_block
x114 w_dec_o_14 w_and_o_14 s0 i4 i3 i2 i1 i0 clk clkb l13 en gnd l14 vdd
inh_gnd inh_gnds inh_vdd inh_vdds basic_block
XI41 W_DEC_O_9 W_AND_O_41 S0 I4 I3 I2 I1 I0 CLK CLKB L40 EN GND L41 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI42 W_DEC_O_10 W_AND_O_42 S0 I4 I3 I2 I1 I0 CLK CLKB L41 EN GND L42 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI43 W_DEC_O_11 W_AND_O_43 S0 I4 I3 I2 I1 I0 CLK CLKB L42 EN GND L43 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI44 W_DEC_O_12 W_AND_O_44 S0 I4 I3 I2 I1 I0 CLK CLKB L43 EN GND L44 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI45 W_DEC_O_13 W_AND_O_45 S0 I4 I3 I2 I1 I0 CLK CLKB L44 EN GND L45 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI46 W_DEC_O_14 W_AND_O_46 S0 I4 I3 I2 I1 I0 CLK CLKB L45 EN GND L46 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI47 W_DEC_O_15 W_AND_O_47 S0 I4 I3 I2 I1 I0 CLK CLKB L46 EN GND L47 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI48 W_DEC_O_0 W_AND_O_48 S0 I4 I3 I2 I1 I0 CLK CLKB L47 EN GND L48 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI49 W_DEC_O_1 W_AND_O_49 S0 I4 I3 I2 I1 I0 CLK CLKB L48 EN GND L49 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI50 W_DEC_O_2 W_AND_O_50 S0 I4 I3 I2 I1 I0 CLK CLKB L49 EN GND L50 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI51 W_DEC_O_3 W_AND_O_51 S0 I4 I3 I2 I1 I0 CLK CLKB L50 EN GND L51 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI52 W_DEC_O_4 W_AND_O_52 S0 I4 I3 I2 I1 I0 CLK CLKB L51 EN GND L52 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI53 W_DEC_O_5 W_AND_O_53 S0 I4 I3 I2 I1 I0 CLK CLKB L52 EN GND L53 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI54 W_DEC_O_6 W_AND_O_54 S0 I4 I3 I2 I1 I0 CLK CLKB L53 EN GND L54 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI55 W_DEC_O_7 W_AND_O_55 S0 I4 I3 I2 I1 I0 CLK CLKB L54 EN GND L55 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI56 W_DEC_O_8 W_AND_O_56 S0 I4 I3 I2 I1 I0 CLK CLKB L55 EN GND L56 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI57 W_DEC_O_9 W_AND_O_57 S0 I4 I3 I2 I1 I0 CLK CLKB L56 EN GND L57 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI58 W_DEC_O_10 W_AND_O_58 S0 I4 I3 I2 I1 I0 CLK CLKB L57 EN GND L58 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI59 W_DEC_O_11 W_AND_O_59 S0 I4 I3 I2 I1 I0 CLK CLKB L58 EN GND L59 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI60 W_DEC_O_12 W_AND_O_60 S0 I4 I3 I2 I1 I0 CLK CLKB L59 EN GND L60 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI61 W_DEC_O_13 W_AND_O_61 S0 I4 I3 I2 I1 I0 CLK CLKB L60 EN GND L61 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI62 W_DEC_O_14 W_AND_O_62 S0 I4 I3 I2 I1 I0 CLK CLKB L61 EN GND L62 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI63 W_DEC_O_15 W_AND_O_63 S0 I4 I3 I2 I1 I0 CLK CLKB L62 EN GND L63 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI64 W_DEC_O_0 W_AND_O_64 S0 I4 I3 I2 I1 I0 CLK CLKB L63 EN GND L64 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI65 W_DEC_O_1 W_AND_O_65 S0 I4 I3 I2 I1 I0 CLK CLKB L64 EN GND L65 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI66 W_DEC_O_2 W_AND_O_66 S0 I4 I3 I2 I1 I0 CLK CLKB L65 EN GND L66 VDD
INH_GND INH_GNDS INH_VDD INH_VDDS BASIC_BLOCK
XI92 W_OR_1_0 W_OR_1_1 GND VDD W_OR_2_0 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI93 W_OR_1_2 W_OR_1_3 GND VDD W_OR_2_1 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI94 W_OR_2_0 W_OR_2_1 GND VDD W_OR_3_0 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI95 CLK W_OR_3_0 EN_AL S0 INH_GND INH_GNDS INH_VDD INH_VDDS HS65_LH_DFPHQX4

XI96 W_AND_O_16 W_AND_O_17 GND VDD W_OR_0_8 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI97 W_AND_O_18 W_AND_O_19 GND VDD W_OR_0_9 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI98 W_AND_O_20 W_AND_O_21 GND VDD W_OR_0_10 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI99 W_AND_O_22 W_AND_O_23 GND VDD W_OR_0_11 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
X100 W_AND_O_24 W_AND_O_25 GND VDD W_OR_0_12 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
X101 W_AND_O_26 W_AND_O_27 GND VDD W_OR_0_13 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
X102 W_AND_O_28 W_AND_O_29 GND VDD W_OR_0_14 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
X103 W_AND_O_30 W_AND_O_31 GND VDD W_OR_0_15 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
X104 W_OR_0_8 W_OR_0_9 GND VDD W_OR_1_4 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
X105 W_OR_0_10 W_OR_0_11 GND VDD W_OR_1_5 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
X106 W_OR_0_12 W_OR_0_13 GND VDD W_OR_1_6 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
X107 W_OR_0_14 W_OR_0_15 GND VDD W_OR_1_7 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
X108 W_OR_1_4 W_OR_1_5 GND VDD W_OR_2_2 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
X109 W_OR_1_6 W_OR_1_7 GND VDD W_OR_2_3 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
X110 W_OR_2_2 W_OR_2_3 GND VDD W_OR_3_1 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
X111 CLK W_OR_3_1 EN_AL S1 INH_GND INH_GNDS INH_VDD INH_VDDS HS65_LH_DFPHQX4

X112 W_AND_O_32 W_AND_O_33 GND VDD W_OR_0_16 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
X113 W_AND_O_34 W_AND_O_35 GND VDD W_OR_0_17 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
X114 W_AND_O_36 W_AND_O_37 GND VDD W_OR_0_18 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI115 W_AND_0_38 W_AND_0_39 GND VDD W_OR_0_19 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI116 W_AND_0_40 W_AND_0_41 GND VDD W_OR_0_20 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI117 W_AND_0_42 W_AND_0_43 GND VDD W_OR_0_21 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI118 W_AND_0_44 W_AND_0_45 GND VDD W_OR_0_22 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI119 W_AND_0_46 W_AND_0_47 GND VDD W_OR_0_23 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI120 W_OR_0_16 W_OR_0_17 GND VDD W_OR_1_8 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI121 W_OR_0_18 W_OR_0_19 GND VDD W_OR_1_9 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI122 W_OR_0_20 W_OR_0_21 GND VDD W_OR_1_10 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI123 W_OR_0_22 W_OR_0_23 GND VDD W_OR_1_11 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI124 W_OR_1_8 W_OR_1_9 GND VDD W_OR_2_4 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI125 W_OR_1_10 W_OR_1_11 GND VDD W_OR_2_5 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI126 W_OR_2_4 W_OR_2_5 GND VDD W_OR_3_2 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI127 CLK W_OR_3_2 EN_AL S2 INH_GND INH_GNDS INH_VDD INH_VDDS HS65_LH_DFPHQX4
XI128 W_AND_0_48 W_AND_0_49 GND VDD W_OR_0_24 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI129 W_AND_0_50 W_AND_0_51 GND VDD W_OR_0_25 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI130 W_AND_0_52 W_AND_0_53 GND VDD W_OR_0_26 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI131 W_AND_0_54 W_AND_0_55 GND VDD W_OR_0_27 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI132 W_AND_0_56 W_AND_0_57 GND VDD W_OR_0_28 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI133 W_AND_0_58 W_AND_0_59 GND VDD W_OR_0_29 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI134 W_AND_0_60 W_AND_0_61 GND VDD W_OR_0_30 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI135 W_AND_0_62 W_AND_0_63 GND VDD W_OR_0_31 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI136 W_OR_0_24 W_OR_0_25 GND VDD W_OR_1_12 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI137 W_OR_0_26 W_OR_0_27 GND VDD W_OR_1_13 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
XI138 W_OR_0_28 W_OR_0_29 GND VDD W_OR_1_14 INH_GND INH_GNDS INH_VDD INH_VDDS PASS_OR
** TESTBENCH

XI221 I0 I1 I2 I3 I4 S0 S1 S2 S3 S4 DIN L0 LOUT VDD 0 VDD! VDDS! 0 GNDS! CLK
CLKB EN EN_AL ABSFULL

** POWER SUPPLY

V1 GNDS! 0 DC 0
V2 VDDS! 0 DC 1
V3 VDD 0 DC 1
V4 VDD! 0 DC 1

** CLOCK AND ENABLE SIGNALS

V5 CLK 0 PULSE 1 0 10E-12 10E-12 5E-9 10E-9
V6 CLKB 0 PULSE 1 0 10E-12 10E-12 5E-9 10E-9
V7 EN 0 PULSE 0 1 0 10E-12 10E-12 5.1195E-5 52E-6
V8 EN_AL 0 PULSE 1 0 10E-12 10E-12 5.1195E-5 52E-6

** RANDOM INPUT SIGNALS

V9 I4 0 PULSE 1 0 5.1195E-5 10E-12 10E-12 150E-9 280E-9
V10 I3 0 PULSE 1 0 5.1195E-5 10E-12 10E-12 100E-9 200E-9
V11 I2 0 PULSE 1 0 5.1195E-5 10E-12 10E-12 200E-9 400E-9
V12 I1 0 PULSE 1 0 5.1195E-5 10E-12 10E-12 400E-9 800E-9
V13 I0 0 PULSE 1 0 5.1195E-5 10E-12 10E-12 800E-9 1.6E-6

** CONFIGURATION BITSTREAM

V14 DIN 0 PWL ( + 0N 0 + 9.99N 0 + 10N 0 + 19.99N 0 + 20N 0 + 29.99N 0 + 30N 0 + 39.99N 0 + 40N 0 + 49.99N 0 + 50N 0 + 59.99N 0 + 60N 0 + 69.99N 0 + 70N 0 + 79.99N 0 + 80N 0 + 89.99N 0 + 90N 0
+ 99.99N 0
+ 100N 0
+ 109.99N 0
+ 110N 0
+ 119.99N 0
+ 120N 0
+ 129.99N 0
+ 130N 0
+ 139.99N 0
+ 140N 0
+ 149.99N 0
+ 150N 0

........................
+ 16160N 1
+ 16169.99N 1
+ 16170N 1
+ 16179.99N 1
+ 16180N 1
+ 16189.99N 1
+ 16190N 1
+ 16199.99N 1
+ 16200N 1
+ 16209.99N 1
+ 16210N 1
+ 16219.99N 1
+ 16220N 1
+ 16229.99N 1
+ 16230N 1
+ 16239.99N 1
+ 16240N 0

........................
+ 51090N 1
+ 51099.99N 1
+ 51100N 1
+ 51109.99N 1
+ 51110N 1
+ 51119.99N 1
+ 51120N 0
+ 51129.99N 0
+ 51130N 0
+ 51139.99N 0
+ 51140N 0
+ 51149.99N 0
+ 51150N 0
+ 51159.99N 0
+ 51160N 0
+ 51169.99N 0
+ 51170N 0
+ 51179.99N 0
+ 51180N 0
+ 51189.99N 0
+ 51190N 0
)
** ANALYSIS AND OUTPUT

.TRAN 0.1NS 52US
.PLOT TRAN V(CLK) V(EN) V(EN_AL) V(VDD) I(V2) I(V3) I(V4) V(DIN) V(I4) V(I3) V(I2) V(I1) V(I0) V(S0) V(S1) V(S2) V(S3) V(S4)
.PLOT TRAN V(L0) V(LOUT)