Design and Implementation of FPGA Configuration Logic Block Using Asynchronous Static NCL

Indira P. Dugganapally, Waleed K. Al-Assadi, Tejaswini Tammina and Scott Smith*
Department of Electrical and Computer Engineering,
Missouri University of Science and Technology
301 W. 16th. Street, Rolla, MO 65409-0040

* Department of Electrical Engineering
University of Arkansas, Fayetteville, AR 72701

Abstract -**This paper proposes the design of a FPGA Configurable Logic Block (CLB) using Asynchronous Static NULL Convention Logic (NCL) Library. The proposed design uses three static LUT's for implementing NCL logic functions. Each LUT can be configured to function as any one of the 27 fundamental NCL Static gates. The proposed CLB supports 10 inputs and three different outputs, each with resettable and inverting variations. The CLB has two modes: Configuration mode and Operation mode. The Static NCL FPGA CLB is simulated at the transistor level using the 1.8V, 180nm TSMC CMOS process.**

Keywords: Configurable Logic Block (CLB), Field Programmable Gate Array (FPGA), NULL Convention Logic (NCL), Look Up Table (LUT).

I. Introduction

Synchronous Digital designs have been the primary focus of the Semiconductor Industry for the past few decades. But with an increasing demand for power efficient, higher performance and noise resistant design techniques, the advantages offered by an asynchronous logic paradigm such as Null Convention Logic (NCL) is not to be neglected. To achieve higher performance, chips must dedicate increasingly larger portions of their area for clock drivers to achieve acceptable skew, assuming normal fabrication process variations, causing these chips to dissipate increasingly higher power, especially at the clock edge when switching is most prevalent.

The size of FPGAs is now more than 1 million equivalent gates, making them a viable alternative to custom design for all but the most complex processors. FPGAs are relatively low-cost and are reconfigurable, making them perfect for prototyping, as well as for implementing the final design, especially for low volume production. To compete with this cheap, reconfigurable synchronous implementation, an NCL-specific FPGA is needed, such that NCL circuits can be efficiently implemented without necessitating a prohibitively expensive full-custom design. This will become increasingly important as asynchronous paradigms become more widely used in the industry to increase circuit robustness, decrease power, and alleviate many clock-related issues, as predicted by the International Technology Roadmap for Semiconductors (ITRS). The 2005 ITRS estimates that asynchronous circuits will account for 19% of chip area within the next 5 years, and 30% of chip area within the next 10 years.

II. NCL Overview

NCL is a self timed logic paradigm in which control is inherent in every datum. NCL follows the weak conditions of Seitz's delay insensitive signaling scheme that "all inputs of a combinational circuit must be null before all outputs become null" along with the condition that "all inputs of the circuit must be data before all outputs become data". By these conditions the self timed operation or delay insensitivity is ensured. The first condition is ensured by using inbuilt hysteresis in the basic NCL gates and second condition is obtained by an intelligent circuit design that is both input complete and observable.

NCL circuits are comprised of 27 fundamental gates. These 27 gates comprise the set of all functions consisting of four or fewer variables. Since each rail of NCL is considered a separate variable, a four variable function is not the same as a function of four literals, which in normal case would consist of 8 variables. The primary type of Threshold gate is the *THmn* gate where $1 \leq m \leq n$. *THmn* gates have n inputs. At least m of the n inputs should be asserted before the outputs become asserted and hence m is a threshold. Each of the n inputs is connected to the rounded part of the diagram in figure representation and the output emanates from the pointed end. The gates threshold m, is represented inside the gate as a label. Another type of threshold gate is the weighted threshold gate $\text{THmnWw}_1\text{w}_2\ldots\text{w}_R$. Weighted threshold gates have an integer value $m \geq w_R > 1$ applied to input R. Here $1 \leq R < n$ where n is the number of inputs; m is the threshold; and w_1, w_2, w_3..., w_R each >1, are the integer weights of input1, input2, input R, respectively. For example, consider Th34W2 gate, whose n=4 inputs are labeled A, B, C and D as shown in fig. 1. The weight of A is 2. Since gates threshold is 3, this implies that in order for the output to be asserted, either inputs B, C, D must all be asserted or input A should be asserted along with any other inputs B, C, D. Like mentioned earlier, NCL gates are designed with state holding capability called hysteresis, such that all inputs must be de-asserted before outputs become de-asserted. This ensures a complete transition of inputs back to null before asserting the output with the inputs wave front of the next data set. NCL gates may also include a RESET input to initialize the gate output to 0 or 1. Circuit diagrams designate this by denoting a d or n after the threshold label inside the gate. A d represents that output rail is reset to data or 1 and an n indicates that output is reset to null or 0.

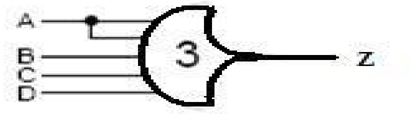

Fig. 1: TH34W2 weighted gate

TABLE I
27 FUNDAMENTAL NCL GATES

NCL Gate	Boolean Function	Transistors (static)	Transistors (semi-static)
TH12	A + B	6	6
TH22	AB	12	8
TH13	A + B + C	8	8
TH23	AB + AC + BC	18	12
TH33	ABC	16	10
TH23w2	A + BC	14	10
TH33w2	AB + AC	14	10
TH14	A + B + C + D	10	10
TH24	AB + AC + AD + BC + BD + CD	26	16
TH34	ABC + ABD + ACD + BCD	24	16
TH44	ABCD	20	12
TH24w2	A + BC + BD + CD	20	14
TH34w2	AB + AC + AD + BCD	22	15
TH44w2	ABC + ABD + ACD	23	15
TH34w3	A + BCD	18	12
TH44w3	AB + AC + AD	16	12
TH24w22	A + B + CD	16	12
TH34w22	AB + AC + AD + BC + BD	22	14
TH44w22	AB + ACD + BCD	22	14
TH54w22	ABC + ABD	18	12
TH34w32	A + BC + BD	17	12
TH54w32	AB + ACD	20	12
TH44w322	AB + AC + AD + BC	20	14
TH54w322	AB + AC + BCD	21	14
THxor0	AB + CD	20	12
THand0	AB + BC + AD	19	13
TH24comp	AC + BC + AD + BD	18	12

A. Static NCL Library

The NCL Static Library consists of the static implementation of 27 fundamental NCL gates given in Table 1. The NCL threshold gates are designed with *hysteresis* state-holding capability, such that after the output is asserted, all inputs must be de-asserted before the output will be de-asserted. Therefore, NCL gates have both *set* and *hold* equations, where the *set* equation determines when the gate will become asserted and the *hold* equation determines when the gate will remain asserted once it has been asserted. The *set* equation determines the gate's functionality as one of the 27 NCL gates, as listed in Table 1, whereas the *hold* equation is the same for all NCL gates, and is simply all inputs ORed together. The general equation for an NCL gate with output *Z* is: Z = set + (Z* • hold), where *Z** is the previous output value and *Z* is the new value. Take the TH23 gate for example. The *set* equation is AB + AC + BC, as given in Table I, and the *hold* equation is A + B + C; therefore the gate is asserted when at least 2 inputs are asserted and it then remains asserted until all inputs are de-asserted. To implement an NCL gate using CMOS technology, an equation for the complement of *Z* (i.e. Z') is also required, which in general form is: Z' = reset + (Z*' • set'), where *reset* is the complement of *hold* (i.e., the complement of each input, ANDed together), such that the gate is de-asserted when all inputs are de-asserted and remains de-asserted while the gate's *set* condition is false. For the TH23 gate, the *reset* equation is A'B'C' and the simplified *set'* equation is A'B' + B'C' + A'C'. Directly implementing these equations for Z and Z', after simplification, yields the static transistor-level implementation of an NCL gate, as shown in Fig. 2 for the TH23 gate. This requires the output, *Z*, to be fed back as an input to the NMOS and PMOS logic to achieve hysteresis behavior. Due to the large transistor count they also dissipate more Power as compared to Semi-Static NCLgates.

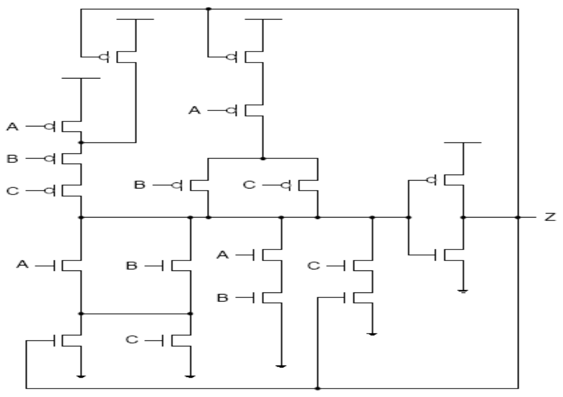

Fig. 2: Static CMOS implementation of TH23 gate
Z=AB+BC+AC

III. Previous Work

There have been a number of asynchronous FPGAs developed over the past 10+ years [2-9]. MONTAGE [2] was developed to support both synchronous and asynchronous circuits. STACC [3] targets bundled data systems in which there are separate data and control paths where the delay in the data path must be matched in the control path. To implement this delay matching, both architectures include some sort of programmable delay element. MONTAGE [2] both use a lookup table (LUT) based design, where the output is fed back as one of the inputs, to implement the C-element's state-holding capability. STACC [3] is based on fine grain FPGA architectures where the global clock is replaced by an array of timing-cells that generate local register control signals. These systems rely heavily on the placement and routing tools to yield a functional FPGA circuit, where all delays are correctly matched. Another type of asynchronous FPGA uses a programmable phased logic cell [4], [5]. In an effort to design a delay-insensitive, reconfigurable logic device, Theseus Logic developed an FPGA based on the Atmel AT40K family [9]. The design involved replacing the D-type Flip Flop within each logic block with a threshold-configurable NCL THm4 gate, and removing the associated clock trees from the original design. Atmel's routing algorithm for this chip was then modified to convert an NCL gate-level schematic to a bit stream to program the FPGA. This method is advantageous in that it reuses a proven architecture, but the design only utilizes a fraction of the NCL threshold gates, thus increasing area and delay for realizing most non-trivial NCL circuits. It also has the disadvantage of being unable to use all of the LUTs in the FPGA, thus resulting in inefficient resource utilization [9]. A more efficient configurable logic element for an NCL FPGA was presented in [1].

IV. CLB Design And Implementation

The proposed CLB design supports 10 logical input variables (A, B, C, D, E. F, G, H, Din_0, Din_1) and supports three different outputs (X, Y and Z). Each output comes with resettable and inverting variations. Fig. 3 shows CLB block diagram. The CLB consists of: 3 x Static LUT, Decoder, Output Reset logic, Output Inversion logic, Programmable Muxes. The CLB has two modes which are Configuration and operating mode. In Configuration mode the 3 LUT' s are programmed to implement different or similar functions and Output reset, Inversion logic along with the programmable muxes are also configured. Once configured the CLB is ready for operation and should operate as the programmer wants it to. The configuration scheme is explained in detail in further sections.

A. Static LUT

The reconfigurable logic portion consists of a 5 bit-address LUT, shown in fig. 4. The static LUT contains 27 NCL static fundamental gates shown in fig. 4, and 28 multiplexers (MUX). The gate inputs, *A*, *B*, *C*, and *D*, are connected to each of the 27 gates and the programmed Dp value decides which output to pass to the LUT output through the MUX logic. Since all gate inputs (i.e. *A*, *B*, *C*, and *D*) are connected to a series of NCL static gates, the LUT function output will be logic 1 only when the selected gate's output is logic 1. The LUT is outputting logic 0 for Address 0. There is no hysteresis logic as the Static gates are designed with internal feedback.

To configure this LUT as a specific NCL gate, the LUT should be programmed with corresponding Dp for any set of inputs corresponding to the gate's set condition, shown in fig. 2. Take for example a TH23 gate, whose equation is AB + AC +BC. The LUT should be programmed with Dp = "00111". The LUT outputs logic 1 for the following four input patterns: *ABC* = 011, 101, 110, and 111, which correspond to setting condition of TH23 gate. The other four combinations *ABC* = 000, 001, 010, and 100, corresponding to logic 0 output for gate TH23 based on its previous state.

For gates with less than four inputs, the unused inputs are not connected. Hence, for the TH23 gate, *D* would be unconnected to TH23 gate. The LUT mentioned in the fig.2 correspond to the static LUT's.

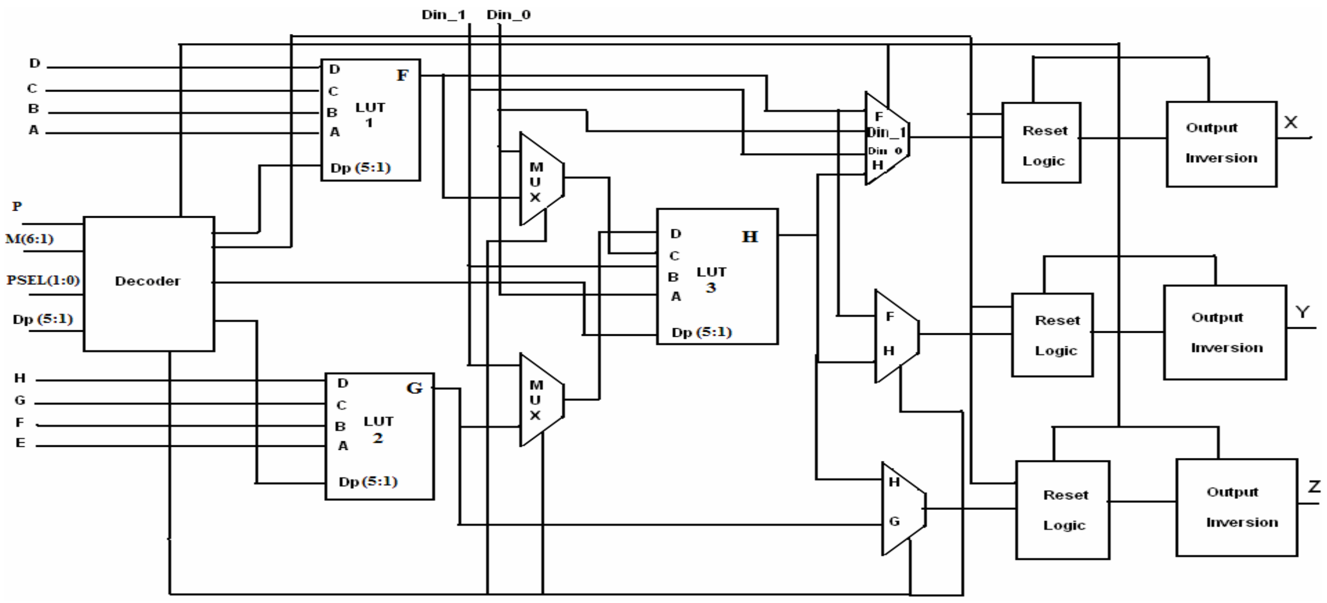

Fig. 3: Static CLB Block Diagram.

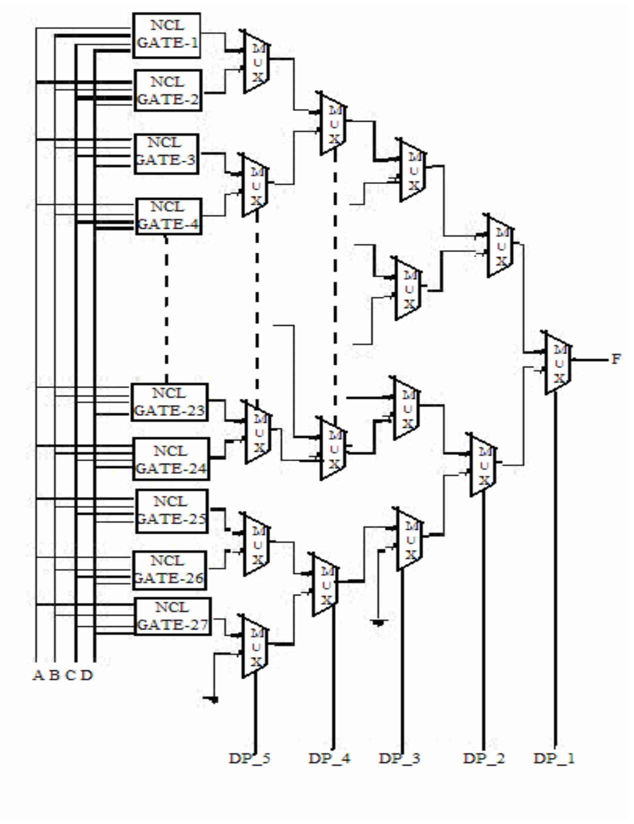

Fig. 4: 4-input Static LUT

B. Decoder

The decoder operates in the Programming mode of the CLB i.e. when P is asserted. Since we have 3 LUT's each of which has a 5 bit programming input Dp, to reduce the number of CLB programming inputs we have designed a decoder. The decoder depending on the two bit input select line muxes the input Dp values to the corresponding LUT.

TABLE II
VALUES OF SELECT INPUTS CORRESPONDING TO LUT

Select Line	**Selected Output**
"00"	LUT 1
"01"	LUT 2
"10"	LUT 3
"11"	Programmable muxes, Reset Value and Output inversion per output setting.

The decoder can be described as two different parts. The following fig. 5 (a) shows decoder hardware for a single input Dp bit for the three LUTs and the fourth th44 gate's output is used for output inversion of one of the three outputs. As seen, when P is set, the first three Th44 gates get selected depending on Psel1 and Psel0 and the programming bit is passed to the LUT. If Psel[1:0] is driven with value "00",Dp[5:1] is updated depending upon the function to be implemented in LUT1. The LUT2 and LUT3 are configured similarly with Psel[1:0] values being "01" and "10" respectively for their configuring. When Psel[1:0] value is "11" , the fourth Th44 gate output decides whether the output inversion takes place or not. Since there are five programming inputs, five sets of four Th44 gates should be present. The fourth output of three sets give the output inversion of three outputs and the fourth and fifth outputs tells us whether the circuit is reset or not (rst) and the reset value (Rv). Deassertion of signal P activates the fourth Th44 gates and also the MUX logic shown in fig. 5(b). The four select lines for four 2:1 muxes and two select lines for one 4:1 mux are given as inputs to the decoder can program the muxes only when P is 0. In this way the Programming of the LUTs and the multiplexers can be done by using the decoder.

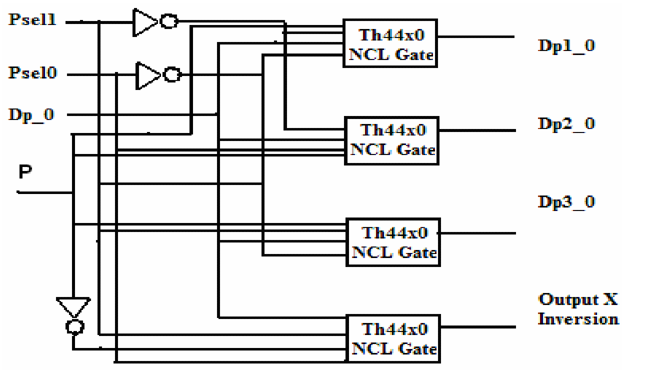

Fig. 5(a)

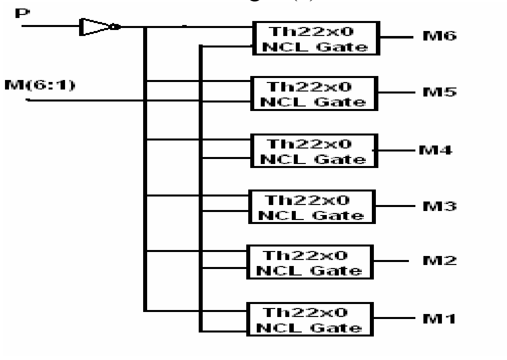

Fig. (b)
Fig. 5(a) & (b): Decoder logic for Single i/p Dp bit.

C. Output Reset and Inversion Logic, Programmable muxes

The proposed CLB has reset logic per output port. The Reset circuit is show below in fig. 6. The reset logic consists of a programmable latch and transmission gate MUX. During the programming phase when *P* is asserted (*nP* is deasserted), the latch stores the value, *Rv*, that the gate will be reset to when *rst* is asserted. *rst* is the MUX select input, such that when it is logic 0, the output of the PUPD function passes through the MUX to be inverted and output on *Z*; and when *rst* is logic 1, the inverse of *Rv* is passed through the MUX. In this way the CLB provides the user the option of resetting the output ports.

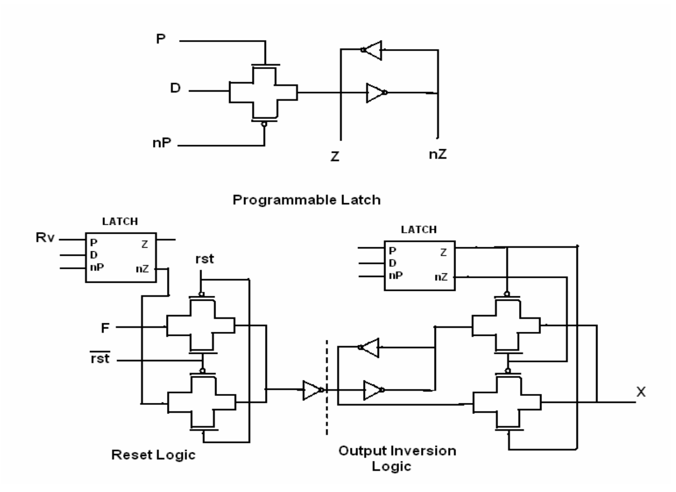

Fig. 6: Output Inversion and Reset Logic

The proposed CLB has inversion and hysteresis logic per output port. The Reset circuit is show above in fig. 6. The output inversion logic also consists of a programmable latch and transmission gate MUX. The programmable latch stores *Inv* during the programming phase, which determines if the gate is inverting or not. There is no need for hysteresis logic as the hysteresis logic is already implemented in each of the 27 fundamental Static gates used to make the LUT. The output and its inverted value are both fed as data inputs to the MUX, so that either the inverted or non-inverted value can be output, depending on the stored value of *Inv*, which is used as the MUX select input. The CLB design includes 5 programmable muxes so that the user has possible combinations at the outputs. During the programming phase i.e signal P asserted and nP deasserted when the Decoder select lines are “11’’, the select lines for these programmable muxes are driven. The user can have a combination of Din_0, LUT1 output, LUT2 output, LUT3 output at the output ports. For example consider a case where LUT1 is configured as AND gate, LUT2 as OR gate, LUT3 as XOR gate and the user can have a combination of LUT1, LUT3 and LUT2 output at X, Y and Z by programming the muxes accordingly.

D. CLB Implementation

A schematic for the Static CLB Design was created using the Mentor Graphics tool “Design Architect”. Each module of the CLB was designed separately and simulated using the “Accusim” tool. The symbols for all the modules were generated. Finally the entire CLB schematic was created by joining the individual symbols. The Physical layout for the proposed CLB design was manually constructed and routed using the Mentor Graphics IC Station tool using a 1.8V, 180nm TSMC CMOS process. As this tool is optimized for standard libraries, we had to manually add NCL cells and route them. Due to the complexity of the design five metals have been used. The VDD and GND port have been made on metal 1. The layout is as shown in fig. 7 below.

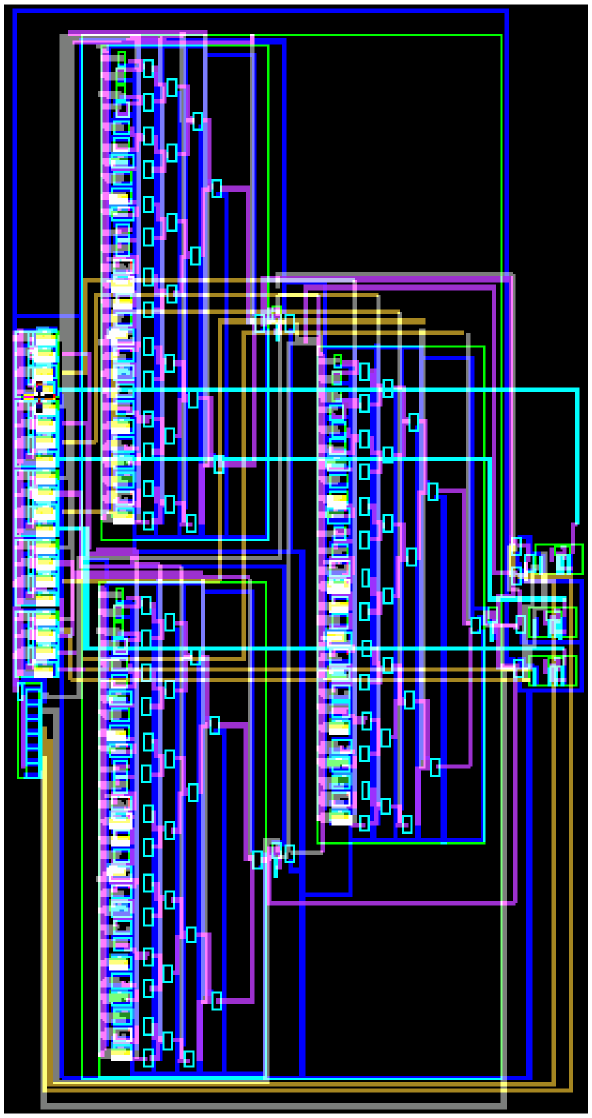

Fig. 7: Layout of the CLB

V. Simulation And Results

The CLB net-list was simulated using Mentor Graphics tools ELDO and Ezwave. Estimated Area is 8197.5x4203 λ^2 which is 737.775 x 378.27 (nm^2), since the value of λ for 180nm technology is 0.09.The Total Power Dissipation of a single LUT has been calculated to be 2.7747E-09 Watts. For the total CLB, the power dissipation was calculated to be 9.1719E-06 Watts. The propagation delay was found to be 1.79 ns. Fig. 8 shows ELDO simulation of this CLB in which LUT1 is programmed as Th44 gate, LUT2 is programmed as Th54w22 gate and LUT 3 is programmed as Th12 gate. The multiplexers have been given select lines in the following order: M1=1, M2=1, M3=0, M4=0, M5=1, M6=1 and the rst is kept ‘0’. The input Din_1 is kept 0 for 210ns and then it is made high. The input Din_0 is kept 1 for 200ns and then it is made low. This configuration makes the output X to be the output of LUT1, output Y to be output of LUT 3 and Z to be output of LUT 2. During the simulation, we make our programming input P go high from 10 to 60 ns and then low till 110ns and high from 110 to 160 ns and then low till 210ns and high till 260ns and then is made low for the whole of simulation time. In the three parts where P is high, we select PSel lines such that three LUTs get configured one after the other. We give the programming inputs Dp_1 to Dp_5 such that the LUT1 gets configured for Th44 gate, that is, Dp(5:1) is "10101" , LUT2 is configured for Th54w22 gate, that is, Dp(5:1) is "10110" and LUT3 is configured for Th12 gate, that is, Dp(5:1) is "00000". Now depending on the inputs A through H and Din(1:0), we get the outputs X, Y and Z. The simulation results are as shown in the fig. 8.

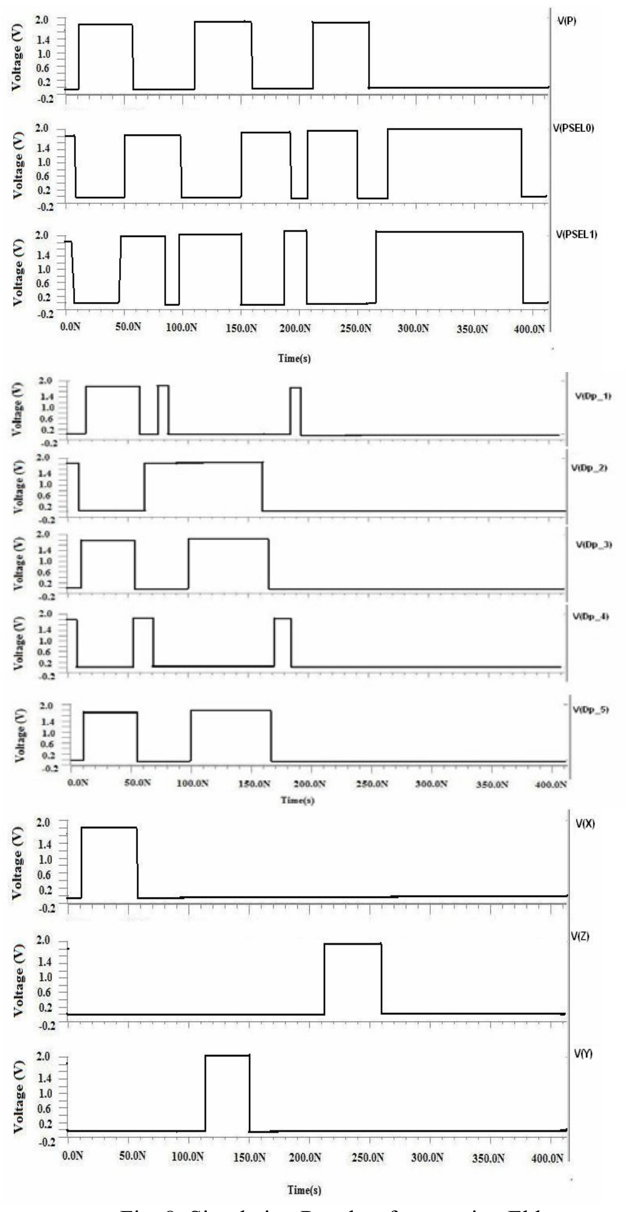

Fig. 8: Simulation Results after running Eldo
Scale: 1 unit = 10 ns

VI. Conclusion And Future Work

As the semester project in MST's graduate-level VLSI course, we designed and implemented a Static FPGA CLB using Asynchronous Static NCL library at all levels of abstraction, from Design schematic to layout, using semi-static NCL gates. The CLB was configured for three different outputs and was successfully simulated and verified to be functionally correct. Furthermore, all of the major system components were implemented, simulated, and verified at the transistor-level and physical-level. However, due to tool problems and time constraints, the system implementation at the physical level was not optimized for area and power. Future work includes optimizing the design for area and power. Also the results can be verified for further complex configurations. Additional topics that need further investigation, but are beyond the scope of this paper, include the overall FPGA architecture, switching matrix, and the FPGA interconnect strategy. Possible choices for overall architecture include island-style or hierarchical. Alternative numbers of LUTs and connection of LUTs within a CLB need to be studied. The overall FPGA interconnect grouping needs to be researched.

Acknowledgment

This work was supported partly by the National Science Foundation under grant DUE 071776.

References

[1] S. C. Smith, "Design of Logic Element for implementing an Asynchronous FPGA." IEEE Transactions on VLSI Systems, Vol. 15/6, June 2007.

[2] S. Hauck, S. Burns, G. Borriello, and C. Ebeling, "An FPGA for Implementing Asynchronous Circuits," IEEE Design & Test of Computers, Vol. 11, No. 3, pp 60-69, 1994.

[3] R. E. Payne, "Self-Timed FPGA Systems," 5th International Workshop on Field Programmable Logic and Applications," pp. 21-35, 1995.

[4] C. Traver, R. B. Reese, and M. A. Thornton, "Cell Designs for Self-Timed FPGAs," 14th Annual IEEE International ASIC/SOC Conference, pp. 175-179, 2001.

[5] M. Aydin and C. Traver, "Implementation of a Programmable Phased Logic Cell," 45th Midwest Symposium on Circuits and Systems, Vol. 2, pp. 21-24, 2002.

[6] J. Teifel,R, Manohar, "An Asynchronous Dataflow FPGA Architecture," IEEE Transactions on Computers, Vol. 53, No. 11, pp.1376-1392, 2004

[7] C. G. Wong, A. J. Martin, and P. Thomas, "An Architecture for Asynchronous FPGAs," IEEE International Conference on Field Programmable Technology, pp. 170-177, 2003.

[8] R. Manohar, "Asynchronous Reconfigurable Logic," Custom Integrated Circuits Conference, 2006.

[9] K. Meekins, D. Ferguson, M. Basta, "Delay Insensitive NCL Reconfigurable Logic," IEEE Aerospace Conference, Vol. 4, pp. 1961-1966, 2002.

[10] D. H. Linder and J. H. Harden, "Phased logic: supporting the synchronous design paradigm with delay-insensitive circuitry," IEEE Transactions on Computers, Vol. 45/9, pp. 1031-1044, 1996.