Hardware Modeling of Binary Coded Decimal Adder in FPGA

MUHAMMAD IBN IBRAHIMY, MD. REZWANUL AHSAN AND IKSANNURAZMI B
BAMBANG SOEROSO
Department of Electrical and Computer Engineering, Faculty of Engineering
International Islamic University Malaysia
Jalan Gombak, Kuala Lumpur 53100
MALAYSIA
Email: ibrahimy@iium.edu.my   http://staff.iium.edu.my/ibrahimy/

Abstract: - There are insignificant relevant research works available which are involved with the Field Programmable Gate Array (FPGA) based hardware implementation of Binary Coded Decimal (BCD) adder. This is because, the FPGA based hardware realization is quiet new and still developing field of research. The article illustrates the design and hardware modeling of a BCD adder. Among the types of adders, Carry Look Ahead (CLA) and Ripple Carry (R-C) adder have been studied, designed and compared in terms of area consumption and time requirement. The simulation results show that the CLA adder performs faster with optimized area consumption. Verilog Hardware Description Language (HDL) is used for designing the model with the help of Altera Quartus II Electronic Design Automation (EDA) tool. EDA synthesis tools make it easy to develop an HDL model and which can be synthesized into target-specific architectures. Whereas, the HDL based modeling provides shorter development phases with continuous testing and verification of the system performance and behavior. After successful functional and timing simulations of the CLA based BCD adder, the design has been downloaded to physical FPGA device. For FPGA implementation, the Altera DE2 board has been used which contains Altera Cyclone II 2C35 FPGA device.

Key-Words: - Binary Coded Decimal Adder, Carry Look Ahead, Ripple Carry, Hardware Description Language, Field Programmable Gate Array

1 Introduction

Addition is used as primitive operation for computing most arithmetic functions, so that it deserves particular attention. The term adder in digital electronics means a circuit to execute addition of numbers. Arithmetic Logic Unit is the main component of central processing unit where the addition, multiplication, comparison and other logical operations are performed. It is typical that digital adders normally use binary numbers to perform addition. However, it is also possible to design an adder from other type of number representation like BCD. Without the modification of adder module, it can perform addition/subtraction of signed numbers by converting the numbers into 1's complement or 2's complement. Nowadays, decimal system of rule is favored, especially when it is working with decimal arithmetic calculation. But, sometimes the decimal arithmetic based conventional software can not cope up with the performance requirement by the applications with widespread range of decimal arithmetic. Before sending the numbers to computer, they need to be converted into binary representation. Contrariwise, the output numbers have to be converted from binary to decimal form. For certain applications, such as business or economical applications required a huge numbers of input/output conversions. The efficiency and performance of the system is then become associated with the rapid conversion of numbers. The BCD system, however, facilitates very fast binary-decimal conversion through encoding each decimal digit separately as a structure of 4 binary bits [1].

With the technological advancement, it is now gaining importance to embed the libraries required for hardware realization in recent commercialized general purpose processors. In classical algorithms, it is proved that the completion time of any program or circuit is dependent on the number of digits/bits available in the operands. Through reviewing a numbers of literatures, it is found that several ideas is proposed for minimizing computational time. Most of the modifications are related to the minimization of carry computation which may reduce proportionality constant [2]. However, the decimal addition has time consumption for carry propagation process within the same range as of binary. It is found that the practical implementation of BCD adders not only save the coding interfaces but also save the time consumption. There are two

techniques being used to design high speed decimal adders. One of the methods produces the direct decimal sums without producing the binary sums. Whereas, another method directly produces the decimal carries through the refinement of carry look ahead. Both of the techniques help to design a unit of parallel processing decimal arithmetic. The designed unit outperforms the binary arithmetic unit in terms of performance and cost. Due to the technological development, the availability of large amount of main memories with computer systems are very common. This is advantageous for multi-programming which results in greater concurrency among I/O, processor and other hardware devices [3]. For future demand environment/application, it would be more attractive to use parallel decimal arithmetic unit to achieve output with a lesser computation.

Field-Programmable Gate Arrays (FPGAs) have emerged as an attractive means of implementing logic circuits, providing instant manufacturing turnaround and negligible prototype [4]. Fig. 1 illustrates the basic architecture of an FPGA. FPGAs are pre-fabricated silicon devices which can be programmed to perform almost any kind of digital circuit or system.
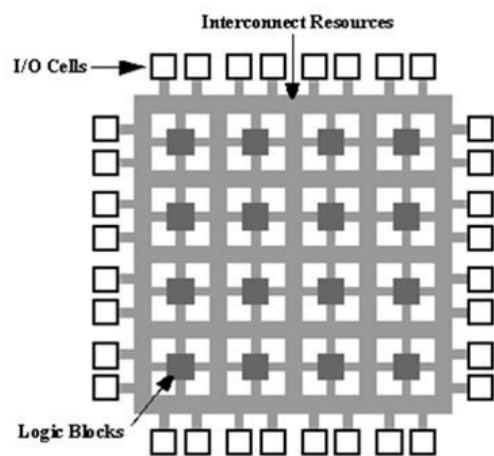

Fig. 1: FPGA architecture

FPGAs are reconfigurable devices with first processing time and lower volume cost. FPGAs are future oriented building blocks that permit seamless reconfiguration/customization of the hardware at an attractive price even in low quantities. The physical FPGA devices are commercially available in usable sizes, in terms of I/O ports, memory resources, functionalities etc., with reasonable price tag. This makes them effective factors for cost saving and time-to-market when designing individual configurations of certain standard products. The application specific integration of IP cores in the FPGA device can considerably reduce the time and

avoid expensive redesign. FPGA offers a potential alternative for speeding up the hardware realization which comes with the merits of lower cost, higher density, and shorter design cycle [5]. In FPGA based design, a gate-level netlist is generated by the synthesis tool which can be used to perform timing analysis based on circuit elements. The netlist can also be used for FPGA's mapping, packaging, and place and route software for generating more accurate timing report using real values. Fig. 2 illustrates the HDL based FPGA flow. However, every design has its own trade off. The Application Specific Integrated Circuit (ASIC) design would increase productivity in terms of understanding and debugging the design at the RTL level rather than working with gate-level schematic. In contrast, FPGA design would give better timing estimations and area utilization which generate better quality of results [6].
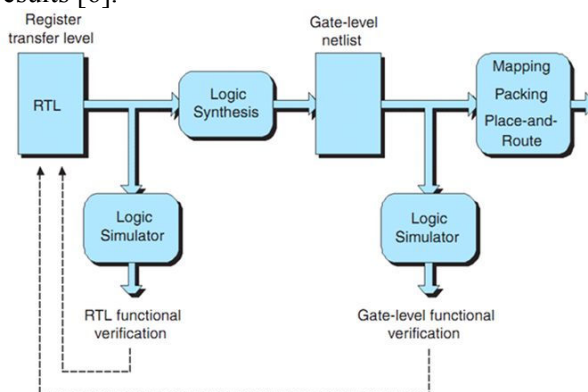

Fig. 2: HDL based FPGA flow

The decimal system arithmetic is preferable than binary number system. Since, it does not only avoid the complexity of coding-decoding interfaces but it also increase the precision and clarity in the results. This article presents a design and hardware modeling of BCD adder implemented into FPGA. This research project aims to develop a decimal adder based on Ripple Carry (R-C) adder and Carry Look Ahead (CLA) adder in FPGA. The development of the design is involved in analyzing some of the major difficulties of complex algorithm. BCD is common in electronic systems where a numeric value is to be shown, especially in system consisting digital logic in its design.

2 Materials and Method

Regardless of R-C or CLA adder, a 4-bit reference adder is used before implementing the final design. For this purpose, a 4-bit CLA is as reference adder for implementing a 8-bit CLA adder. Furthermore, a 4-bit R-C adder is designed to compare its

performance with CLA adder. The target is to get the output in the form of BCD number and displaying it in the 7-segment display.

2.1 4-bit R-C Adder Operation

The R-C, sometimes called a pseudoparallel adder or simply parallel adder. An n-bit R-C is a $(2n + 1)$ input and $(n + 1)$ output combinational logic device that can add two n-bit binary numbers. The block diagram symbol and general operation format for this adder are presented in Fig. 3, together with an illustration of the ripple carry effect in Fig. 4. The general operation format represents the familiar addition algorithm used in conventional arithmetic where carry from an addition operation is always to the next most significant stage.
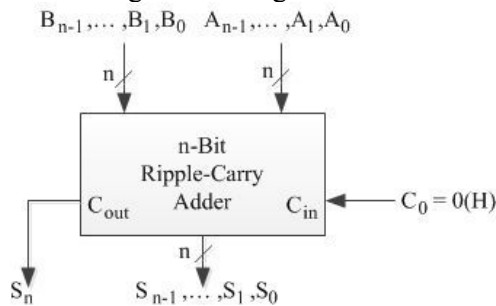

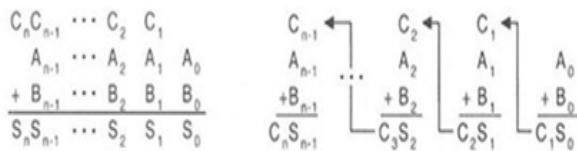Fig. 3:   Block diagram of n-bit ripple-carry adder

Fig. 4:   n-bit ripple-carry adder with operation format and ripple carry effect

The subscripts are consistent with the powers of 2 to the left of the radix point in polynomial notion. Thus, the bits of each word representing a number are written in ascending order of positional weight from right to left. Actually, the position of the radix point in the two numbers is arbitrary, since the adder has no means of sensing these positions. If significant bit positions exist to the right of the radix point for augend A and addend B, meaning that these numbers have a fraction component, then there must be an equal number of such positions for the two numbers, each of n bits total. All that is required a series array of n full adders (FA) designated as FA0, FA1, ..., FAn-1, one for each bit, be connected such that the carry-out of one stage is the carry-in to the next most significant stage. An n-bit ripple-carry adder is more likely to be designed by using n number of m-bit adder modules rather than individual FAs. An example presented in Fig. 5,

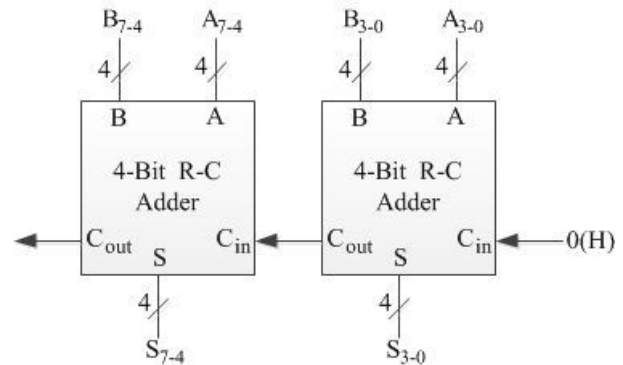features two 4-bit R-C in ripple-carry fashion to produce a 8-bit adder.

Fig. 5:   8-bit ripple-carry adder implementation with 4-bit ripple-carry adders

2.2 4-bit CLA Adder Operations

Basically, the notion of having R-C is to let each adder compute a carry and forward it to a subsequent adder. One way to improve this method is by having an algorithm to pre-calculate the carries before forwarding the sum Co the next adder. Therefore, such implementation can be done in CLA by expediting the carry propagation and eliminating the inter stage carry delay. To invoke this algorithm [7], carry propagate as well as carry generate are being used. The CLA circuit is shown in Fig. 6 and logic circuit for CLA adder is presented in Fig. 7.
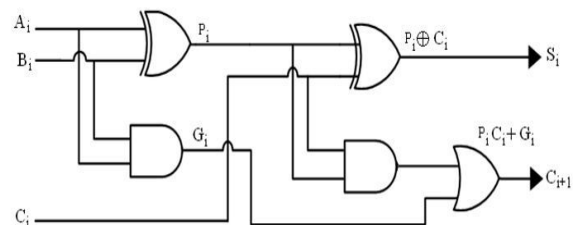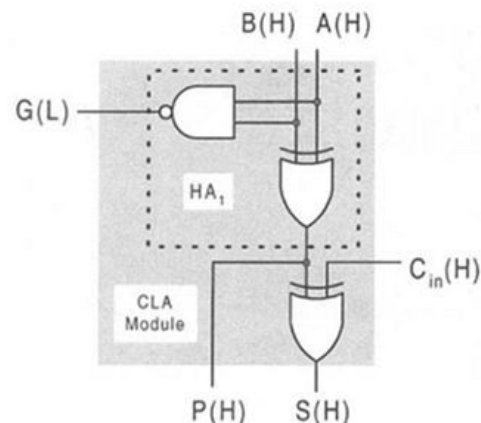

Fig. 6:   2-bit carry look ahead adder

Fig. 7: The logic circuit for carry look ahead adder

Algorithm below shows how to calculate propagate and generate function.

$P_i = A_i \oplus B_i$ and $G_i = A \bullet B$

Sum and Carryout can be calculated by

$S_i = P_i \oplus C_i$ and $C_{i+1} = G_i + P_iC_i$

where, G_i is carry generate, P_i is carry propagate, C_i and C_{i+1} is the Carryout in first and next stage.

2.3 Complete Design Scheme

The complete design flow is given in the block diagram in Fig. 8. The model is implemented for 8-bit binary numbers which are inputted by user. Summation operation is operated in decimal adder module. Both the 8-bit CLA and R-C adders are successfully modeled for decimal adder module. The resulted 8-bit binary number is converted to BCD number by BCD converter module. The decoder for binary to BCD number conversion is shown in Fig. 9. By utilizing BCD number system, the manipulation of numerical data can be greatly simplified by treating each digit as a separate single sub-circuit for display purpose. This matches much more closely the physical reality of display hardware which enables to use a series of separate identical 7-segment display to build a metering circuit. If the numeric values are stored and manipulated as pure binary, interfacing to such a display would require complex circuitry. Therefore, the calculations associated with BCD are relatively simple which leads to a simpler overall system than converting to binary.
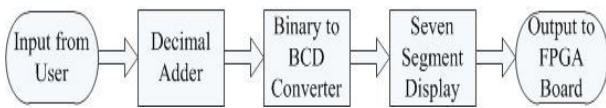

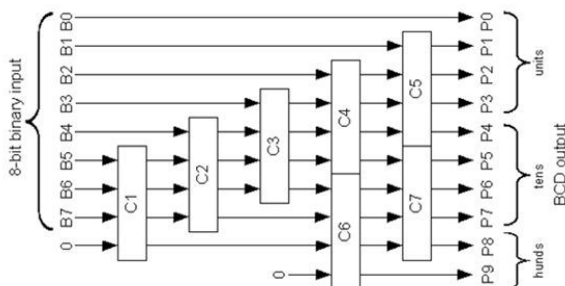Fig. 8: Block diagram of the complete design flow

Fig. 9: Binary to BCD decoder block diagram

The final result is displayed in FPGA board by 7-segment display module. The configuration for the 7-segment LED display is given in Fig. 10. The outputs of BCD to 7-Segment Decoder are assigned to 7 different alphabets which are a, b, c, d, e, f and g. The signal '0' indicates that the LED is 'ON' while signal '1' indicates that the LED is 'OFF'. In addition, for overall digit representation (1-9) is shown in Table 1.
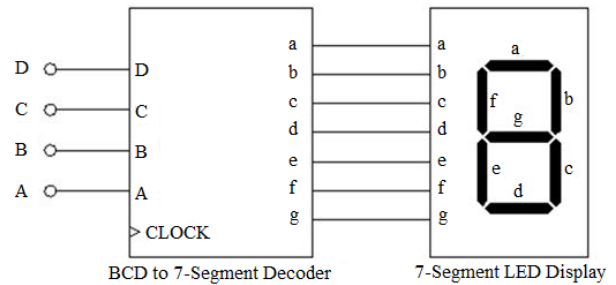

Fig. 10: 7-Segment LED display configuration

Table 1: BCD to 7-Segment Decoder Truth-Table

Binary inputs				Decoder outputs							7-segment display outputs
D	C	B	A	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	1	0	1	1	9

2.4 Timing Analysis and Synthesis

Process of generating a logic circuit from an initial specification is called synthesis that may be given in the form of schematic diagram or code written in the hardware description language which means an abstract form of desired circuit behavior. Typically, it represents the register transfer level (RTL), and is turned into a design implementation in terms of logic gates [4]. On the other hand, timing analysis may be referred to as the measurement of the delay along with the various timing paths and verifies the performance and operation of the design. In order to meet the timing requirements, user could specify time constraints and assignments. For timing analysis, Altera Quartus II has been used while Synopsys [8] is used for synthesis part. In timing analysis, the pin to pin delay time (t_{pd}) can be observed by specifying the clock setup time (t_{su}), clock to output delay time (t_{co}) and clock hold time (t_h). In contrast, synthesis of the designed model has covered the optimization and mapping process. Optimization means the process of finding an equivalent representation of the specified logic

circuit under one or more specified constraints. Mapping on the other hand means a process of fitting logic produced by synthesis and placing it into particular programmable logic device [9]. In order to start the timing analysis, classic timing analyzer wizard is chosen. Actually, those timing requirements are crucial for any design before it could be implemented. Fig. 11 shows the illustration of timing requirement for t_{su}, t_{co}, and t_h. In Fig. 12, t_{su}, t_h, t_{co} and t_{pd} are specified in order to get full timing analysis report. Next, the frequency is set to 50 MHz appropriately to meet the entire timing requirements. Fig. 13 shows the timing summary for the settings.
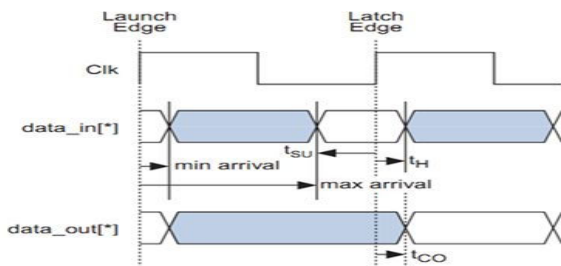

Fig. 11: Timing requirement t_{su}, t_{co} and t_h

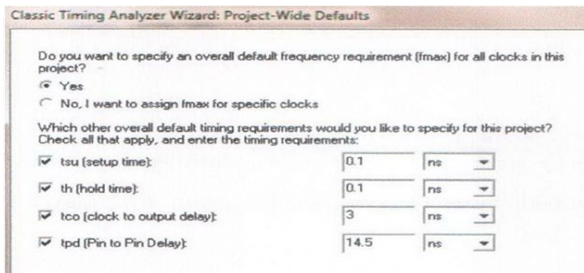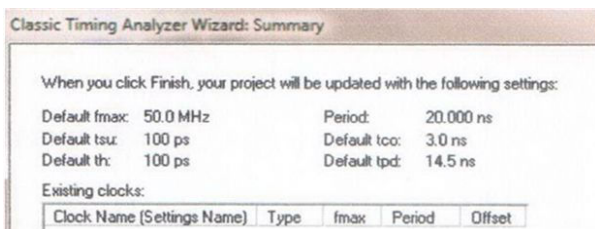Fig. 12: Timing requirement settings

Fig. 13: Timing requirement and frequency summary

3 Simulation and Results

In this work, Altera Quartus II and Synopsys EDA tools are used for timing analysis and synthesis. The simulation output for both 4-bit R-C and 4-bit CLA adders are presented through comparison in terms of timing analysis and area utilization. After verifying the block diagram, the behaviour of both 4-bit R-C and CLA adders are verified by simulation through testbench process.

3.1 4-bit R-C Adder Simulation

The 4-bit R-C adder has been simulated with appropriate inputs. As shown in Fig. 14, the value of Sum and Cout depend on the value of X, Y and Cin. When time t = 0 ns, there is no input so the value for Sum is zero and Cout at low state (Cout = 0). At time t = 0 s and t = 20 ns, the Sum supposed to get the same result which is 15. But at t = 20 ns, since Cin = 1 which results in Sum = 0. Whereas, at the same time the value for Cout become high state(Cout = 1). This means, the Carryout receive its value since the value for summation is equal or more than 15.

3.2 4-bit CLA Adder Simulation

The 4-bit CLA adder is also simulated with appropriate inputs. As shown in Fig. 15, the value of Sum and Cout depend on the values of X, Y and Cin. When time t = 0 ns, there is no input so the value for Sum is zero and Cout at low state (Cout = 0). At time t = 80 ns and t = 90 ns, the Sum is supposed to get the result of 17, but the hexadecimal value could not exceed 15. Therefore, it yields an output of 1, because the most significant bit binary addition will have a Carryout. Thus, the Carryout is generated and is transferred to the next stage. This is the advantage of CLA that, it could guess the future carry out before propagating the value.
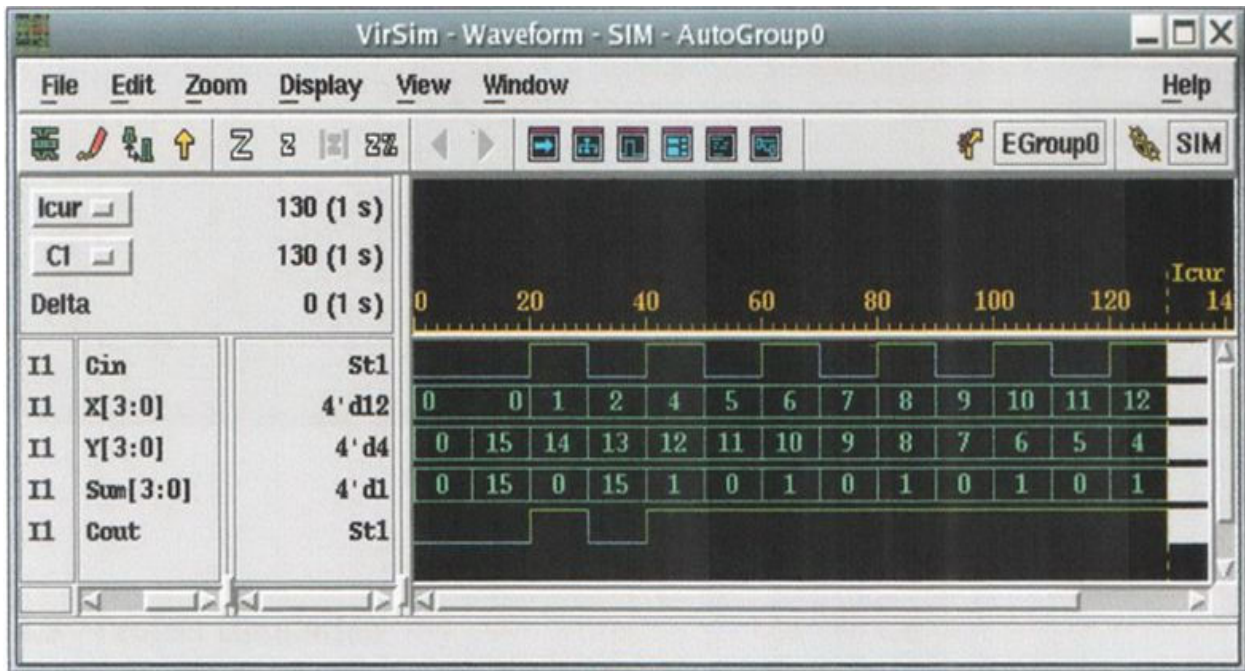
Muhammad Ibn Ibrahimy, Md. Rezwanul Ahsan,
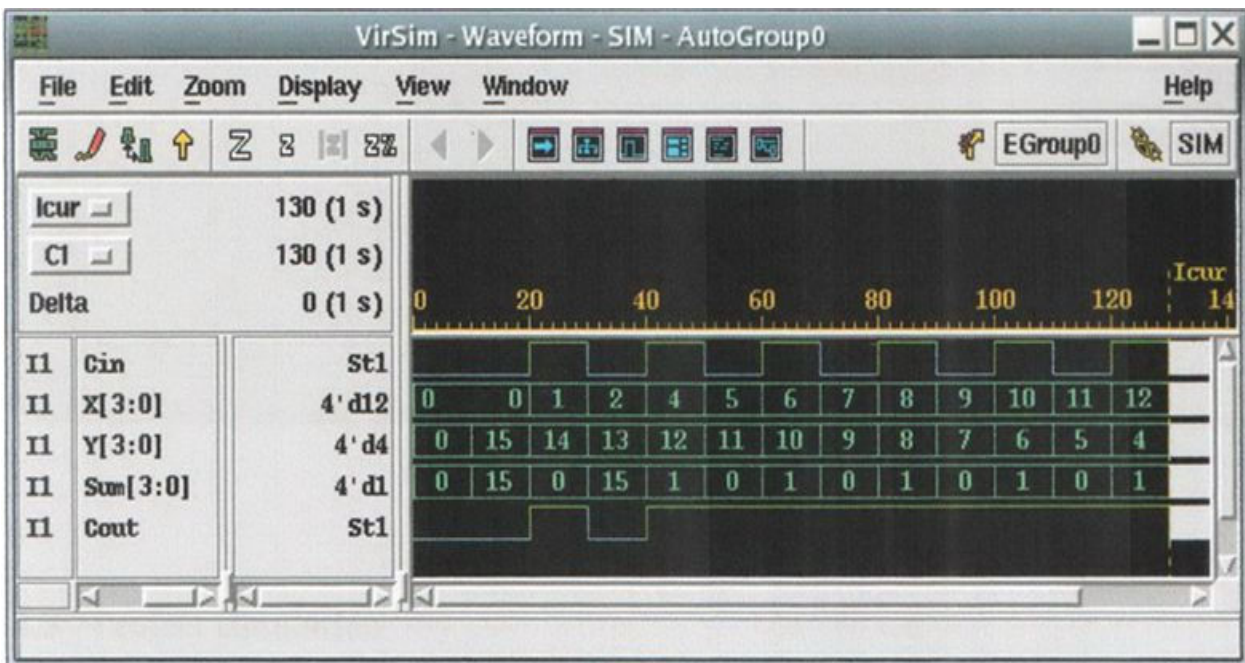Iksannurazmi B. Bambang Soeroso

Fig. 14: 4-bit R-C adder simulation

Fig. 15: 4-bit CLA adder simulation

3.3 Simulation of the Design

Fig. 16 shows the RTL architecture of a 8-bit decimal adder which is formed by a CLA adder with decoder. The input is 8-bit binary number and the addition of both inputs yields 8-bit of binary number. The generated binary number is then converted into BCD. The conversion process has been done by implementing a decoder and the output is then sent to 7-segment display in Altera DE-2 board. The RTL diagram as shown in Fig. 17 gives a depth view of 4-bit CLA block and Fig. 18 shows the RTL logic block of the decoder. Furthermore, Fig. 19 shows the testbench used for the simulation where it can be seen that the output is separated by 4-bits as denoted by X and Y respectively. X represents BCD in tens while Y represents BCD in ones. Both combinations yield to two digit decimal number. Mathematically, 14+75 yields to 89 and the given output has been accurately represented in BCD number.
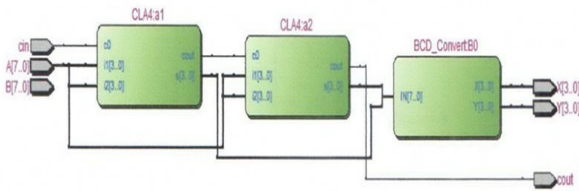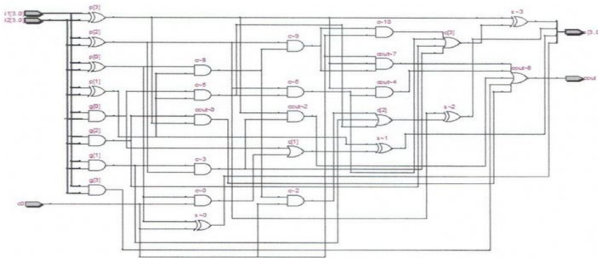
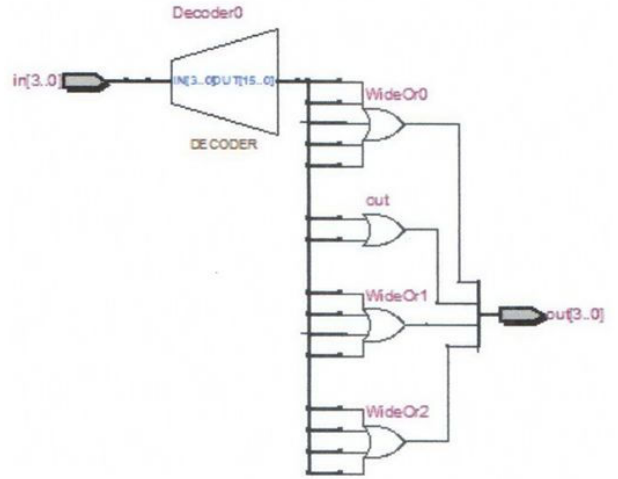Fig. 16: RTL view of 8-bit CLA with decoder

Fig. 17: RTL view of 4-bit CLA

Fig. 18: RTL view of the decoder

Fig. 19: Testbench of 8-bit CLA with decoder

3.4 Timing Analysis

It is important to realize the significance of timing analysis before a design can be proceed to the next stage. In timing analysis, one could eventually set a time constraints for a particular design to enhance its performance. In ASIC design, one could maximize the area utilization since the user determines the number of logic gates involved. In contrast, if a design is implemented using FPGA, one's ability to specifically restrict area maximization seems unreliable. The area utilization is fixed by the restriction of logic gates in FPGA itself which is a plus point for the designer. Fig. 20 and Fig. 21 presents the timing analysis report for 4-bit RCA adder and CLA adder respectively.

Fig. 20: Timing analysis and synthesis of 4-bit RCA adder with Synopsys Timing Report

```
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : CLA
Version: Y-2006.06-SP4
Date   : Thu Apr 21 11:57:45 2011
***************************************

    Startpoint: i1[0] (input port clocked by clk)
    Endpoint: cout (output port clocked by clk)
    Path Group: clk
    Path Type: max

    clock clk (rise edge)              9.79    9.79
    clock network delay (ideal)        0.00    9.79
    output external delay             -3.00    6.79
    data required time                         6.79
    ------------------------------------------------
    data required time                         6.79
    data arrival time                         -6.79
    ------------------------------------------------
    slack (MET)                                0.00
```

Fig. 21: Timing analysis and synthesis of 4-bit CLA adder with Synopsys Timing Report

The summary of the report is presented in Table 2, which clearly shows that the CLA adder is faster than R-C adder with maximized cell area utilization.

Table 2: Summary of synthesis analysis for R-C and CLA adder

Adder Type	Num. of bit	Data arrival time (ns)	Cell area used	Total cell
R-C	4-bit	7.2	860.530029	14
	16-bit	9.24	3765.85	24
CLA	4-bit	6.79	1071.290039	39
	16-bit	8.77	4134.629	34

3.5 Physical Hardware Implementation

The complete design of a 8-bit CLA adder with decoder is downloaded into the Altera Cyclone II 2C35 FPGA device with Altera DE-2 board. Before a Verilog code is programmed into the FPGA of Altera DE-2 board, some steps need to be taken which include the assignment of the pins. Fig. 22 shows the pin assignment environment and Fig. 23 shows the pin assignment needed for selected inputs and outputs. If there is any unassigned pins occurred, the inputs will be in the tri-state. Several binary inputs and their corresponding outputs are tested and verified. Fig. 24 shows one of the example where the output is given in 7-segment display. For this case, the input A is assigned to digit 55 (00110111) and whereas B is assigned to 44 (00101100). The output is given 99 and purely represented in 7-segment display.
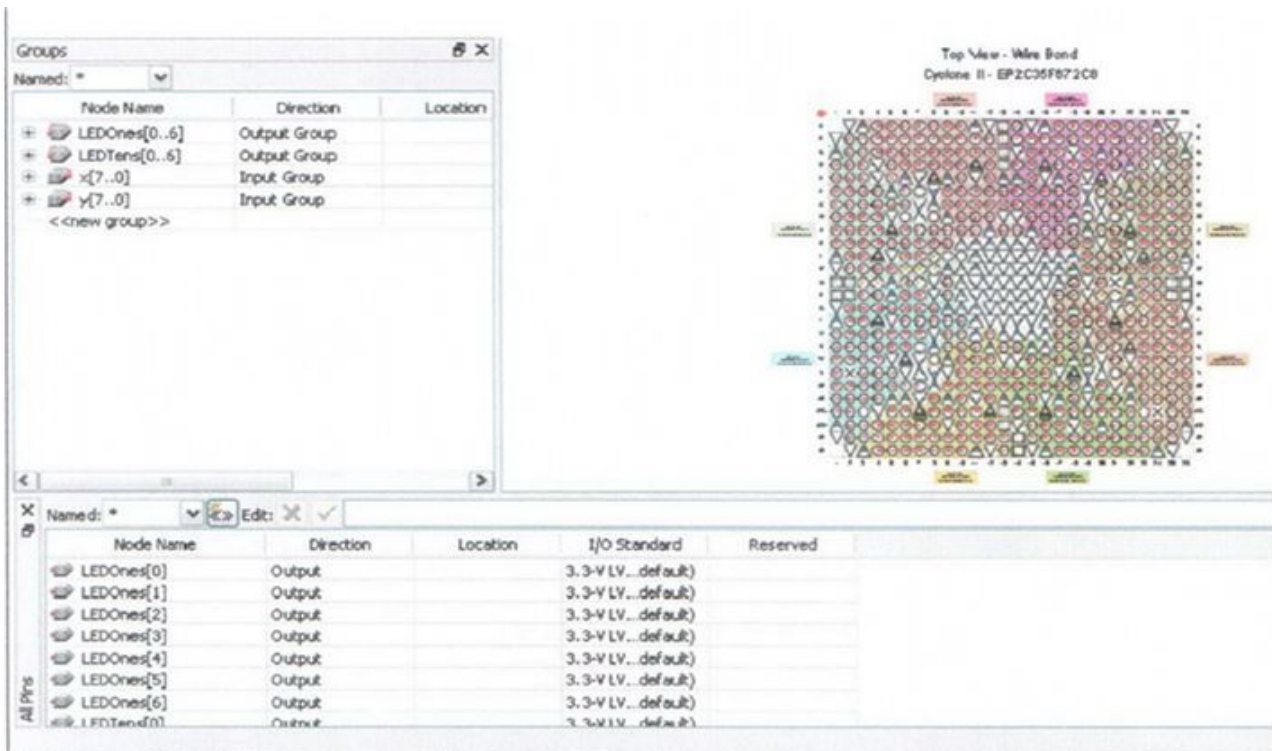

Fig. 22: Pin assignment overview

|		Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard
1		LEDOnes[6]	Output	PIN_V13	8	B8_N0	3.3-V LVTTL (default)
2		LEDOnes[5]	Output	PIN_V14	8	B8_N0	3.3-V LVTTL (default)
3		LEDOnes[4]	Output	PIN_AE11	8	B8_N0	3.3-V LVTTL (default)
4		LEDOnes[3]	Output	PIN_AD11	8	B8_N0	3.3-V LVTTL (default)
5		LEDOnes[2]	Output	PIN_AC12	8	B8_N0	3.3-V LVTTL (default)
6		LEDOnes[1]	Output	PIN_AB12	8	B8_N0	3.3-V LVTTL (default)
7		LEDOnes[0]	Output	PIN_AF10	8	B8_N0	3.3-V LVTTL (default)
8		LEDTens[6]	Output	PIN_AB24	6	B6_N1	3.3-V LVTTL (default)
9		LEDTens[5]	Output	PIN_AA23	6	B6_N1	3.3-V LVTTL (default)
10		LEDTens[4]	Output	PIN_AA24	6	B6_N1	3.3-V LVTTL (default)
11		LEDTens[3]	Output	PIN_Y22	6	B6_N1	3.3-V LVTTL (default)
12		LEDTens[2]	Output	PIN_W21	6	B6_N1	3.3-V LVTTL (default)
13		LEDTens[1]	Output	PIN_V21	6	B6_N1	3.3-V LVTTL (default)
14		LEDTens[0]	Output	PIN_V20	6	B6_N1	3.3-V LVTTL (default)
15		on	Input	PIN_V2	1	B1_N0	3.3-V LVTTL (default)
16		cout	Output	PIN_AE22	7	B7_N0	3.3-V LVTTL (default)
17		x[7]	Input	PIN_C13	3	B3_N0	3.3-V LVTTL (default)
18		x[6]	Input	PIN_AC13	8	B8_N0	3.3-V LVTTL (default)
19		x[5]	Input	PIN_AD13	8	B8_N0	3.3-V LVTTL (default)
20		x[4]	Input	PIN_AF14	7	B7_N1	3.3-V LVTTL (default)
21		x[3]	Input	PIN_AE14	7	B7_N1	3.3-V LVTTL (default)
22		x[2]	Input	PIN_P25	6	B6_N0	3.3-V LVTTL (default)
23		x[1]	Input	PIN_N26	5	B5_N1	3.3-V LVTTL (default)
24		x[0]	Input	PIN_N25	5	B5_N1	3.3-V LVTTL (default)
25		y[7]	Input	PIN_U4	1	B1_N0	3.3-V LVTTL (default)
26		y[6]	Input	PIN_U3	1	B1_N0	3.3-V LVTTL (default)
27		y[5]	Input	PIN_T7	1	B1_N0	3.3-V LVTTL (default)
28		y[4]	Input	PIN_P2	1	B1_N0	3.3-V LVTTL (default)
29		y[3]	Input	PIN_P1	1	B1_N0	3.3-V LVTTL (default)
30		y[2]	Input	PIN_N1	2	B2_N1	3.3-V LVTTL (default)
31		y[1]	Input	PIN_A13	4	B4_N1	3.3-V LVTTL (default)
32		y[0]	Input	PIN_B13	4	B4_N1	3.3-V LVTTL (default)

Fig. 23: Inputs and outputs pin assignment
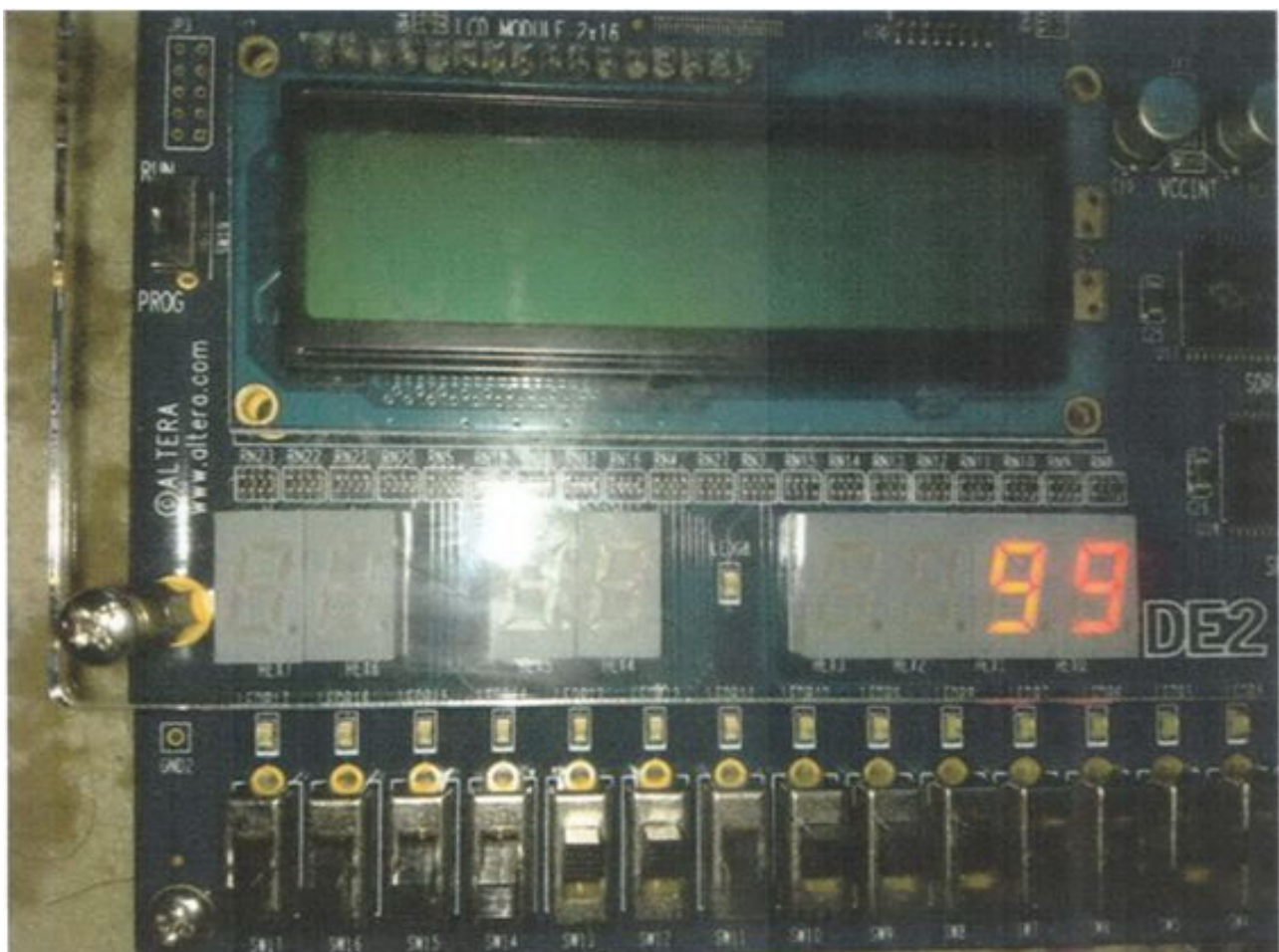
Fig. 24: Output on 7-segment display of Altera DE-2 board

4 Conclusion

The basic algorithm for BCD adder has been implemented in Verilog HDL and verified the behavior of the adder through simulation. The simulation result gives the desired output for both the R-C and CLA adders. In synthesis part, it has been found that CLA adder is faster than R-C adder but it requires much area and cell that lead to consume more power. The physical FPGA model is developed with the help of Altera DE-2 board using the EDA tool Quartus II. FPGA based model has much simpler designing cycle due to the EDA software handles much of routing, placement and timing. Future task may involve with designing the BCD adder with higher numbers of bit using this 8-bit or 4-bit adder as reference adder. The future work may also associate with the realization of layout design where the integrated circuit can be designed corresponding to the pattern of metal, oxide or semiconductor layers.

References:

[1] B. Shirazi, D. Y. . Yun, and C. N. Zhang, "RBCD: redundant binary coded decimal adder," *Computers and Digital Techniques, IEE Proceedings E,* vol. 136, no. 2, pp. 156–160, Mar. 1989.

[2] J.-P. Deschamps, G. J. A. Bioul, and G. D. Sutter, *Synthesis of Arithmetic Circuits: FPGA, ASIC and Embedded Systems*, 1st ed. Wiley-Interscience, 2006.

[3] M. S. Schmookler and A. Weinberger, "High Speed Decimal Addition," *IEEE Transactions on Computers*, vol. C-20, no. 8, pp. 862– 866, Aug. 1971.

[4] S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with Verilog Design*, 2nd ed. McGraw-Hill Science/Engineering/Math, 2007.

[5] I. Kuon, R. Tessier, and J. Rose, *FPGA Architecture*. Now Publishers Inc, 2008.

[6] C. Maxfield, *FPGAs: Instant Access*, First. Newnes, 2008.

[7] R. B. Reese and M. A. Thornton, *Introduction to logic synthesis using Verilog HDL*. Morgan & Claypool Publishers, 2006.

[8] Synopsys, Inc., "RTL Synthesis & Test," *RTL Synthesis & Test*. [Online]. Available: http://www.synopsys.com/Tools/Implementatio n/RTLSynthesis/Pages/default.aspx.

[9] "Quartus II TimeQuest Timing Analyzer Cookbook." Altera Corporation, 2011.