International Journal of Research (IJR)
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Vol. 9 Issue 01
January 2022

Design and Analysis of High Speed Wallace Tree multiplier using Parallel Prefix Adders for VLSI Circuit Designs

Gumpena Vikram Babu[1], Ashok Reddy[2]
[1]P.G. Scholar, [2]Guide, Head of the Department
BRANCH : ECE- VLSI
[1,2] Geethanjali College of Engineering & Technology
Email: [1]gumpenavikram@gmail.com, [2]avrsvr_ece@redissmail.com

Abstract

Variable latency adders have been recently proposed in literature. A variable latency adder employs speculation: the exact arithmetic function is replaced with an approximated one that is faster and gives the correct result most of the time, but not always. The approximated adder is augmented with an error detection network that asserts an error signal when speculation fails. Speculative variable latency adders have attracted strong interest thanks to their capability to reduce average delay compared to traditional architectures. This paper proposes a novel variable latency speculative adder based on Han-Carlson parallel-prefix topology that resulted more effective than variable latency Kogge-Stone topology. The paper describes the stages in which variable latency speculative prefix adders can be subdivided and presents a novel error detection network that reduces error probability compared to previous approaches. Several variable latency speculative adders, for various operand lengths, using both Han-Carlson and Kogge-Stone topology, have been synthesized using the UMC 65 nm library. Obtained results show that proposed variable latency Han-Carlson adder outperforms both previously proposed speculative Kogge-Stone architectures and non-speculative adders, when high-speed is required. It is also shown that non-speculative adders remain the best choice when the speed constraint is relaxed.

Index Terms—Addition, digital arithmetic, parallel-prefix adders, speculative adders, speculative functional units, variable latency adders.

INTRODUCTION TO VLSI

1.1 Very-large-scale integration

Very-large-scale integration (VLSI) is the process of creating integrated circuits by combining thousands of transistors into a single chip. VLSI began in the 1970s when complex semiconductor and communication technologies were being developed. The microprocessor is a VLSI device.

Copyright © authors 2022

p-ISSN: 2348-6848
e-ISSN: 2348-795X
Vol. 9 Issue 01
January 2022

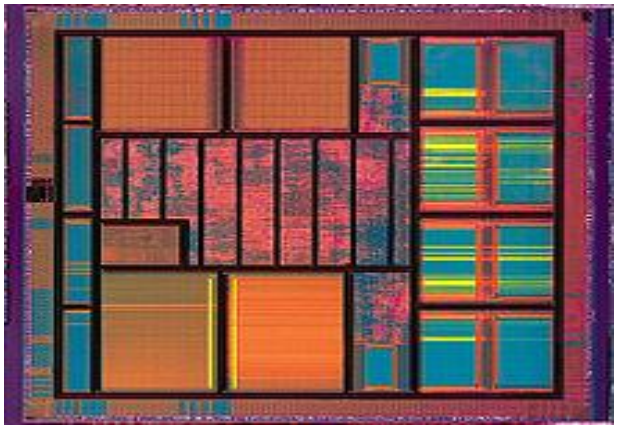


Fig1.1 A VLSI integrated-circuit die

INTRODUCTION TO ADDERS

2.1 Motivation

To humans, decimal numbers are easy to comprehend and implement for performing arithmetic.However, in digital systems, such as a microprocessor, DSP (Digital Signal Processor)or ASIC (Application-Specific Integrated Circuit), binary numbers are more pragmatic for a given computation. This occurs because binary values are optimally efficient at representing many values.

Binary adders are one of the most essential logic elements within a digital system. In addition, binary adders are also helpful in units other than Arithmetic Logic Units (ALU),such as multipliers, dividers and memory addressing. Therefore, binary addition is essential that any improvement in binary addition can result in a performance boost for any computing system and, hence, help improve the performance of the entire system.The major problem for binary addition is the carry chain. As the width of the input operand increases, the length of the carry chain increases. Figure 2.1 demonstrates an example of an 8- bit binary add operation and how the carry chain is affected. This example shows that the worst case occurs when the carry travels the longest possible path, from the least significant bit (LSB) to the most significant bit (MSB). In order to improve the performance of carry-propagate adders, it is possible to accelerate the carry chain, but not eliminate it. Consequently, most digital designers often resort to building faster adders when optimizing a computer architecture, because they tend to set the critical path for most computations.

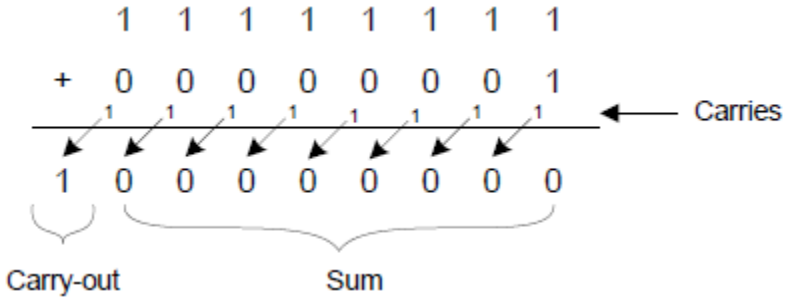


Figure 2.1: Binary Adder Example.

The binary adder is the critical element in most digital circuit designs including digital signal processors (DSP) and microprocessor data path units. As such, extensive research continues to be focused on improving the power delay performance of the adder. In VLSI implementations, parallel-prefix adders are known to have the best performance.

Copyright © authors 2022

p-ISSN: 2348-6848
e-ISSN: 2348-795X
Vol. 9 Issue 01
January 2022

Reconfigurable logic such as Field Programmable Gate Arrays (FPGAs) has been gaining in popularity in recent years because it offers improved performance in terms of speed and power over DSP-based and microprocessor-based solutions for many practical designs involving mobile DSP and telecommunications applications and a significant reduction in development time and cost over Application Specific Integrated Circuit (ASIC) designs.

The power advantage is especially important withthe growing popularity of mobile and portable electronics, which make extensive use of DSP functions. However, because of the structure of the configurable logic androuting resources in FPGAs, parallel-prefix adders will have a different performance than VLSI implementations. In particular, most modern FPGAs employ a fast-carry chain which optimizes the carry path for the simple Ripple Carry Adder (RCA).In this paper, the practical issues involved in designing and implementing tree-based adders on FPGAs are described. Several tree-based adder structures are implemented and characterized on a FPGA and compared with the Ripple Carry Adder (RCA) and the Carry Skip Adder (CSA). Finally, some conclusions and suggestions for improving FPGA designs to enable better tree-based adder performance are given.

Carry-Propagate Adders

Binary carry-propagate adders have been extensively published, heavily attacking problems related to carry chain problem. Binary adders evolve from linear adders, which have a delay approximately proportional to the width of the adder, e.g. ripple-carry adder (RCA),to logarithmic-delay adder, such as the carry-lookahead adder (CLA). There are some additional performance enhancing schemes, including the carry-increment adder and the Ling adder that can further enhance the carry chain, however, in Very Large Scale Integration (VLSI) digital systems, the most efficient way of offering binary addition involves utilizing parallel-prefix trees, this occurs because they have the regular structures that exhibit logarithmic delay.

Research Contributions

The implementations that have been developed in this dissertation help to improve the design of Carry select adders and their associated computing architectures. This has the potential of impacting many application specific and general purpose computer architectures. Consequently, this work can impact the designs of many computing systems, as well as impacting many areas of engineers and science. In this paper, the practical issues involved in designing and implementing Carry select adders on FPGAs are described. Several carry select adder structures are implemented and characterized on a FPGA and compared with the CSLA with Ripple Carry Adder (RCA) and the CSLA with Binary Excess Converter. Finally, some conclusions and suggestions for improving FPGA designs to enable better carry select adder performance are given.

FPGA Implementation

Introduction to FPGA

FPGA contains a two dimensional arrays of logic blocks and interconnections between logic blocks. Both the logic blocks and interconnects are programmable. Logic blocks are programmed to implement a desired function and the interconnections are programmed using the switch boxes to connect the logic blocks.

To be more clear, if we want to implement a complex design (CPU for instance), then the design is divided into small sub functions and each sub function is implemented using one

Copyright © authors 2022

p-ISSN: 2348-6848
e-ISSN: 2348-795X
Vol. 9 Issue 01
January 2022

logic block. Now, to get our desired design (CPU), all the sub functions implemented in logic blocks must be connected and this is done by programming the internal structure of an FPGA which is depicted in the following figure 7.1.

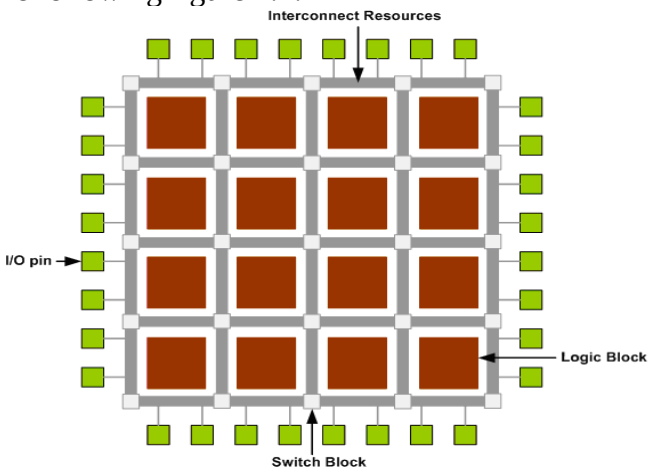


Figure 7.1: FPGA interconnections

FPGAs, alternative to the custom ICs, can be used to implement an entire System On one Chip (SOC). The main advantage of FPGA is ability to reprogram. User can reprogram an FPGA to implement a design and this is done after the FPGA is manufactured. This brings the name "Field Programmable."

Custom ICs are expensive and takes long time to design so they are useful when produced in bulk amounts. But FPGAs are easy to implement within a short time with the help of Computer Aided Designing (CAD) tools (because there is no physical layout process, no mask making, and no IC manufacturing). Some disadvantages of FPGAs are, they are slow compared to custom ICs as they can't handle vary complex designs and also they draw more power. Xilinx logic block consists of one Look Up Table (LUT) and one Flip-Flop. An LUT is used to implement number of different functionality. The input lines to the logic block go into the LUT and enable it. The output of the LUT gives the result of the logic function that it implements and the output of logic block is registered or unregistered output from the LUT. SRAM is used to implement a LUT.A k-input logic function is implemented using 2^k * 1 size SRAM. Number of different possible functions for k input LUT is 2^2^k. Advantage of such an architecture is that it supports implementation of so many logic functions, however the disadvantage is unusually large number of memory cells required to implement such a logic block in case number of inputs is large.

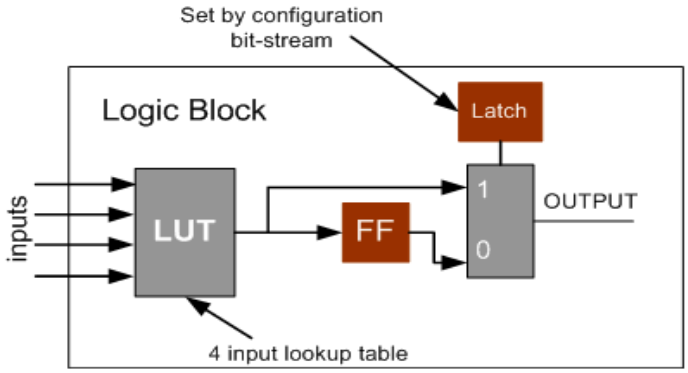


Figure 7.2 shows a 4-input LUT based implementation of logic block

Copyright © authors 2022

p-ISSN: 2348-6848
e-ISSN: 2348-795X
Vol. 9 Issue 01
January 2022

LUT based design provides for better logic block utilization. A k-input LUT based logic block can be implemented in number of different ways with tradeoff between performance and logic density.An n-LUT can be shown as a direct implementation of a function truth-table. Each of the latch hold's the value of the function corresponding to one input combination. For Example: 2-LUT can be used to implement 16 types of functions like AND, OR, A +not B.... Etc.

Interconnects

A wire segment can be described as two end points of an interconnection with no programmable switch between them. A sequence of one or more wire segments in an FPGA can be termed as a track.

Typically an FPGA has logic blocks, interconnects and switch blocks (Input /Output blocks). Switch blocks lie in the periphery of logic blocks and interconnect. Wire segments are connected to logic blocks through switch blocks. Depending on the required design, one logic block is connected to another and so on.

FPGA DESIGN FLOW

In this part of tutorial we are going to have a short intro on FPGA design flow. A simplified version of design flow is given in the flowing diagram.

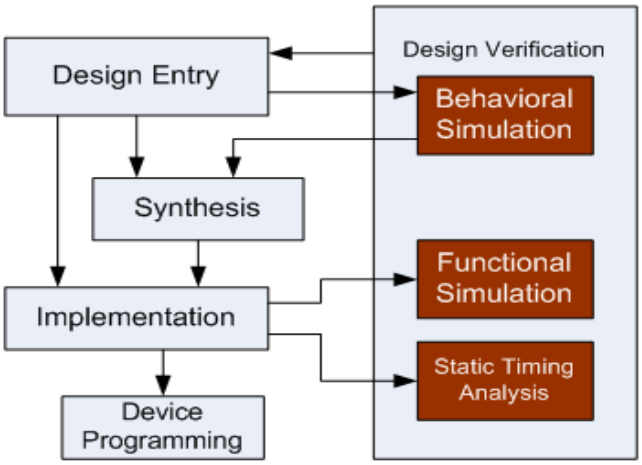


Figure 7.3 FPGA Design Flow

Design Entry

There are different techniques for design entry. Schematic based, Hardware Description Language and combination of both etc. . Selection of a method depends on the design and designer. If the designer wants to deal more with Hardware, then Schematic entry is the better choice. When the design is complex or the designer thinks the design in an algorithmic way then HDL is the better choice. Language based entry is faster but lag in performance and density.

HDLs represent a level of abstraction that can isolate the designers from the details of the hardware implementation. Schematic based entry gives designers much more visibility into the hardware. It is the better choice for those who are hardware oriented. Another method but rarely used is state-machines. It is the better choice for the designers who think the design as a series of states. But the tools for state machine entry are limited. In this documentation we are going to deal with the HDL based design entry.

Copyright © authors 2022

p-ISSN: 2348-6848
e-ISSN: 2348-795X
Vol. 9 Issue 01
January 2022

Synthesis

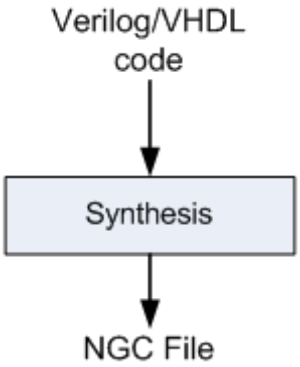


Figure 7.4 FPGA Synthesis

The process that translates VHDL/ Verilog code into a device netlist format i.e. a complete circuit with logical elements (gates flip flop, etc…) for the design. If the design contains more than one sub designs, ex. to implement a processor, we need a CPU as one design element and RAM as another and so on, then the synthesis process generates netlist for each design element Synthesis process will check code syntax and analyze the hierarchy of the design which ensures that the design is optimized for the design architecture, the designer has selected. The resulting netlist(s) is saved to an NGC (Native Generic Circuit) file (for Xilinx® Synthesis Technology (XST)).

7.2.3 Implementation

This process consists of a sequence of three steps

- Translate
- Map
- Place and Route

Translate:

Process combines all the input netlists and constraints to a logic design file. This information is saved as a NGD (Native Generic Database) file. This can be done using NGD Build program. Here, defining constraints is nothing but, assigning the ports in the design to the physical elements (ex. pins, switches, buttons etc) of the targeted device and specifying time requirements of the design. This information is stored in a file named UCF (User Constraints File). Tools used to create or modify the UCF are PACE, Constraint Editor Etc.

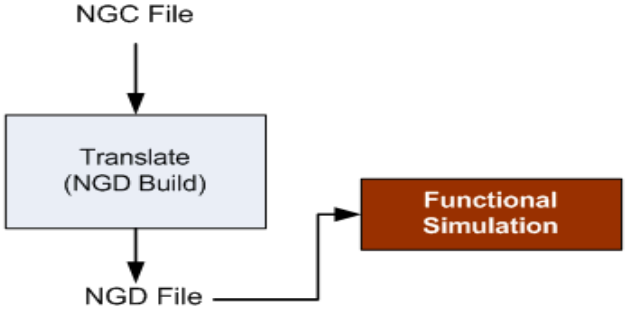


Figure 7.5 FPGA Translate

Map:

Process divides the whole circuit with logical elements into sub blocks such that they can be fit into the FPGA logic blocks. That means map process fits the logic defined by the

Copyright © authors 2022

p-ISSN: 2348-6848
e-ISSN: 2348-795X
Vol. 9 Issue 01
January 2022

NGD file into the targeted FPGA elements (Combinational Logic Blocks (CLB), Input Output Blocks (IOB)) and generates an NCD (Native Circuit Description) file which physically represents the design mapped to the components of FPGA. MAP program is used for this purpose.

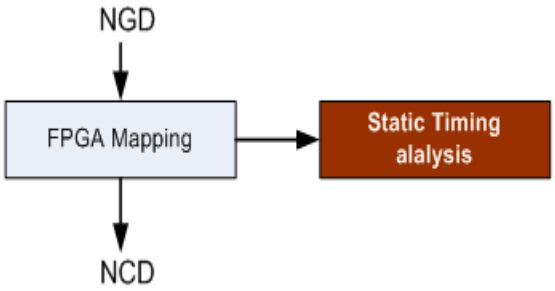


Figure 7.6 FPGA map

Place and Route:

PAR program is used for this process. The place and route process places the sub blocks from the map process into logic blocks according to the constraints and connects the logic blocks. Ex. if a sub block is placed in a logic block which is very near to IO pin, then it may save the time but it may affect some other constraint. So tradeoff between all the constraints is taken account by the place and route process.

The PAR tool takes the mapped NCD file as input and produces a completely routed NCD file as output. The output NCD file consists of the routing information.

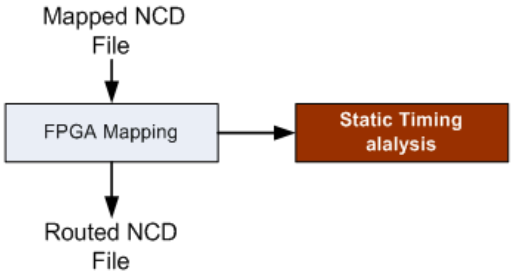


Figure 7.7 FPGA Place and route

7.3 Synthesis Result

To investigate the advantages of using our technique in terms of area overhead against "Fully ECC"and against the partially protection, we implemented andsynthesized for a Xilinx XC3S500E different versions of a32-bit, 32-entry, dual read ports, single write port registerfile. Once the functional verification is done, the RTL model is taken to the synthesis process using the Xilinx ISE tool. In synthesis process, the RTL model will be converted to the gate level netlist mapped to a specific technology library. Here in this Spartan 3E family, many different devices were available in the Xilinx ISE tool. In order to synthesis this design the device named as "XC3S500E" has been chosen and the package as "FG320" with the device speed such as "-4".

RTL Schematic

The RTL (Register Transfer Logic) can be viewed as black box after synthesize of design is made. It shows the inputs and outputs of the system. By double-clicking on the diagram we can see gates, flip-flops and MUX.

SOURCE CODE FOR 16 BIT HAN CARLSON ADDER:

`timescale 1ns / 1ps

//

Copyright © authors 2022

p-ISSN: 2348-6848
e-ISSN: 2348-795X
Vol. 9 Issue 01
January 2022

```
// Company:
// Engineer:
//
// Create Date:    18:47:47 07/29/2016
// Design Name:
// Module Name:    HCA16
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module HCA16(sum,cout,a,b,cin);
input  [15:0] a,b;
output [15:0] sum;
inputcin;
outputcout;
// input stage
wire [15:0] g,p;
assign g = a & b;
assign p = a ^ b;
//LAC tree
wire [14:0] G;
  //wire [15:0] c;

//level1 output
wire [7:0]l;
wire [7:1]m;
//level2 output
wire [6:0]q;
wire [7:1]r;
//level3 output
wire [6:0]s;
wire [6:2]t;
//level4 output
wire [3:0]v;
//level4 output
wire [8:0]w;
  //level1
gn GC0  (.g2(g[0]),  .p2(p[0]),  .g1(cin),  .G(l[0]));

gp BC0  (.g2(g[2]),  .p2(p[2]),  .g1(g[1]),  .p1(p[1]),  .G(l[1]),  .P(m[1]) );
```

Copyright © authors 2022

p-ISSN: 2348-6848
e-ISSN: 2348-795X
Vol. 9 Issue 01
January 2022

```
gp BC1  (.g2(g[4]),  .p2(p[4]),  .g1(g[3]),  .p1(p[3]),  .G(l[2]),  .P(m[2]) );
gp BC2  (.g2(g[6]),  .p2(p[6]),  .g1(g[5]),  .p1(p[5]),  .G(l[3]),  .P(m[3]) );
gp BC3  (.g2(g[8]),  .p2(p[8]),  .g1(g[7]),  .p1(p[7]),  .G(l[4]),  .P(m[4]) );
gp BC4  (.g2(g[10]), .p2(p[10]), .g1(g[9]),  .p1(p[9]),  .G(l[5]), .P(m[5]));
gp BC5  (.g2(g[12]), .p2(p[12]), .g1(g[11]), .p1(p[11]), .G(l[6]), .P(m[6]));
gp BC6  (.g2(g[14]), .p2(p[14]), .g1(g[13]), .p1(p[13]), .G(l[7]), .P(m[7]));
  //level2
gn GC1  (.g2(l[1]),  .p2(m[1]),  .g1(cin),  .G(q[0]));
gp BC7  (.g2(l[2]),  .p2(m[2]),  .g1(l[1]),  .p1(m[1]),  .G(q[1]),  .P(r[1]) );
gp BC8  (.g2(l[3]),  .p2(m[3]),  .g1(l[2]),  .p1(m[2]),  .G(q[2]),  .P(r[2]) );
gp BC9  (.g2(l[4]),  .p2(m[4]),  .g1(l[3]),  .p1(m[3]),  .G(q[3]),  .P(r[3]) );
gp BC10 (.g2(l[5]),  .p2(m[5]),  .g1(l[4]),  .p1(m[4]),  .G(q[4]),  .P(r[4]) );
gp BC11 (.g2(l[6]),  .p2(m[6]),  .g1(l[5]),  .p1(m[5]),  .G(q[5]),  .P(r[5]) );
gp BC12 (.g2(l[7]),  .p2(m[7]),  .g1(l[6]),  .p1(m[6]),  .G(q[6]),  .P(r[6]) );
  //level3
gn GC2  (.g2(q[1]),  .p2(r[1]),  .g1(cin),   .G(s[0]));
gn GC3  (.g2(q[2]),  .p2(r[2]),  .g1(q[0]),   .G(s[1]));
gp BC13 (.g2(q[3]),  .p2(r[3]),  .g1(q[1]),  .p1(r[1]),  .G(s[2]),  .P(t[2]) );
gp BC14 (.g2(q[4]),  .p2(r[4]),  .g1(q[2]),  .p1(r[2]),  .G(s[3]),  .P(t[3]) );
gp BC15 (.g2(q[5]),  .p2(r[5]),  .g1(q[3]),  .p1(r[3]),  .G(s[4]),  .P(t[4]) );
gp BC16 (.g2(q[6]),  .p2(r[6]),  .g1(q[4]),  .p1(r[4]),  .G(s[5]),  .P(t[5]) );
  //level4
gn GC4  (.g2(s[2]),  .p2(t[2]),  .g1(l[0]),  .G(v[0]) );
gn GC5  (.g2(s[3]),  .p2(t[3]),  .g1(q[0]),  .G(v[1]) );
gn GC6  (.g2(s[4]),  .p2(t[4]),  .g1(s[0]),  .G(v[2]) );
gn GC7  (.g2(s[5]),  .p2(t[5]),  .g1(s[1]),  .G(v[3]) );
  //level5
gn GC8  (.g2(g[1]),  .p2(p[1]),  .g1(l[0]),  .G(w[0]));
gn GC9  (.g2(g[3]),  .p2(p[3]),  .g1(q[0]),  .G(w[1]));
gn GC10 (.g2(g[5]),  .p2(p[5]),  .g1(s[0]),  .G(w[2]));
gn GC11 (.g2(g[7]),  .p2(p[7]),  .g1(s[1]),  .G(w[3]));
gn GC12 (.g2(g[9]),  .p2(p[9]),  .g1(v[0]),  .G(w[4]));
gn GC13 (.g2(g[11]),  .p2(p[11]),  .g1(v[1]),  .G(w[5]));
gn GC14 (.g2(g[13]),  .p2(p[13]),  .g1(v[2]),  .G(w[6]));
 // gn GC15 (.g2(g[15]),  .p2(p[15]),  .g1(v[3]),  .G(w[7]));
  /*gp BC17 (.g2(q[6]), .p2(r[6]), .g1(s[5]), .p1(t[5]), .G(s[6]), .P(t[6]));
gn GC16 (.g2(s[6]), .p2(t[6]), .g1(cin), .G(w[8]));*/
  // output stage
assign sum[0] = p[0] ^ cin;
assign sum[15:1] = (p[15:1]) ^ G;
assign G = {v[3],w[6],v[2],w[5],v[1],w[4],v[0],w[3],s[1],w[2],s[0],w[1],q[0],w[0],l[0]};
assigncout = w[6];
endmodule
```

The corresponding schematics of the Han Carlson adder after synthesis is shown below.

Copyright © authors 2022

International Journal of Research (IJR)
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Vol. 9 Issue 01
January 2022

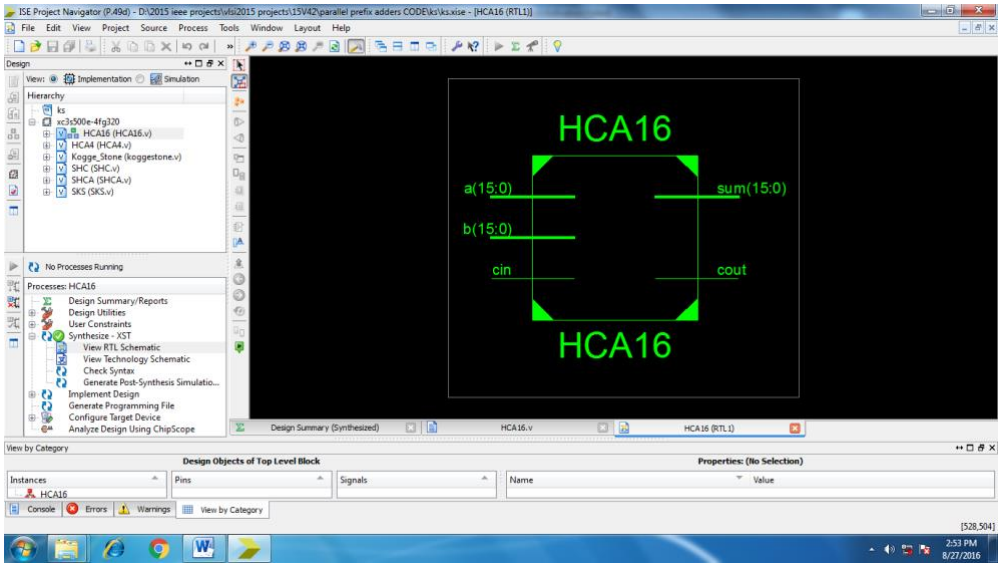


Figure 7.13: RTL schematic of Top-level 16-Bit Han Carlson Adder

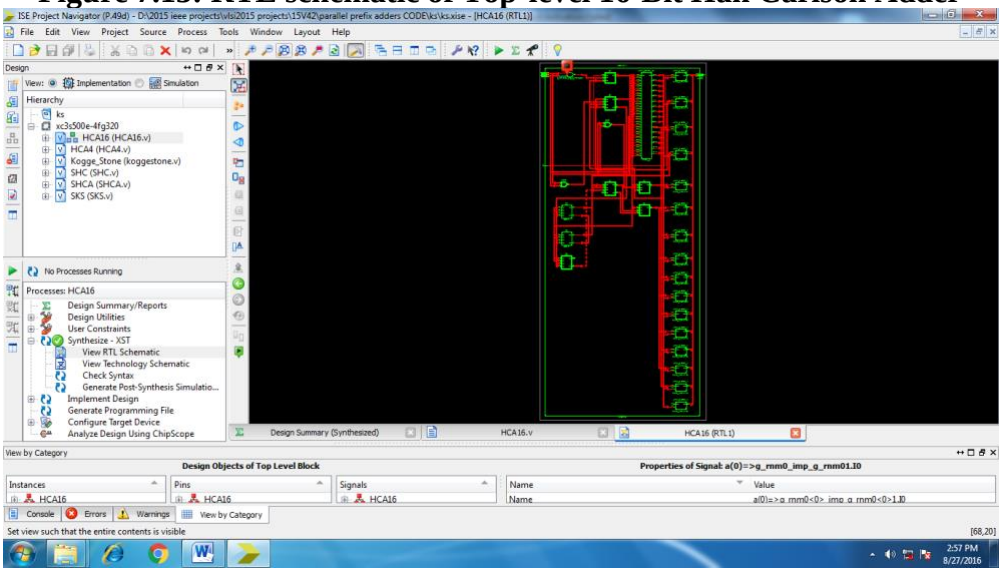
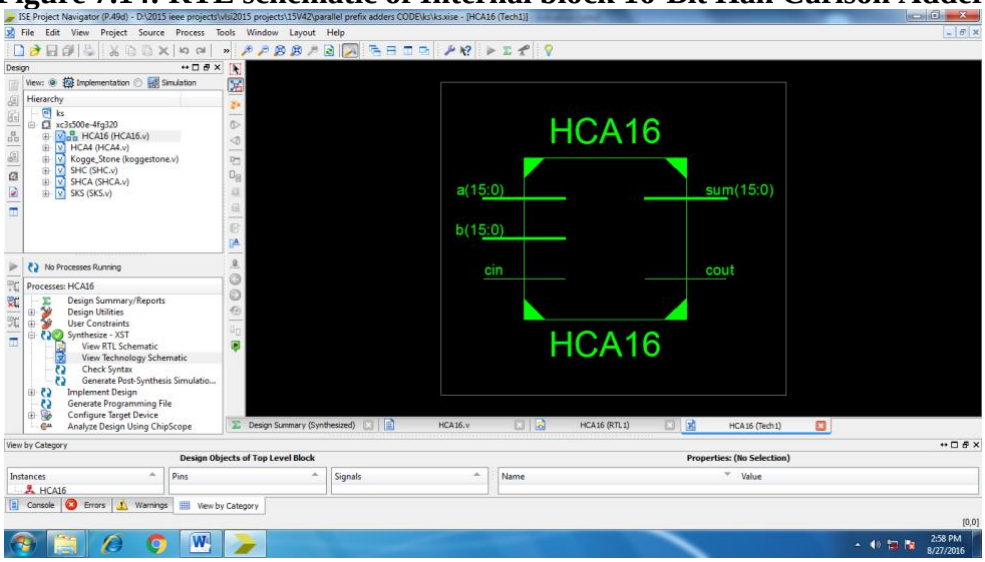


Figure 7.14: RTL schematic of Internal block 16-Bit Han Carlson Adder



Copyright © authors 2022

Figure 7.15: Technology schematic of Top-level 16-Bit Han Carlson Adder

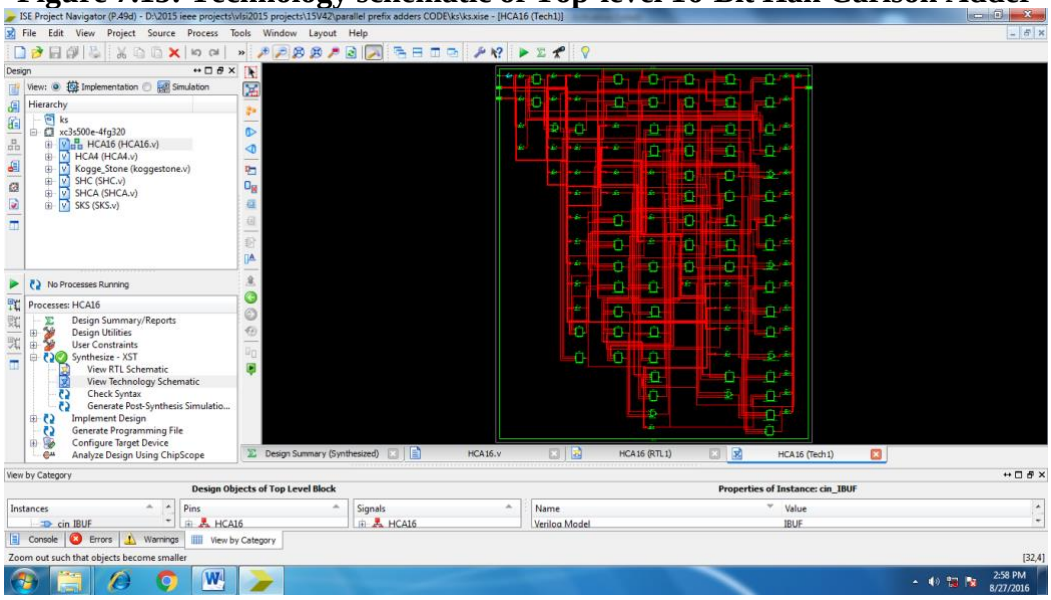


Figure 7.16: Technology schematic of Internal block 16-Bit Han Carlson Adder

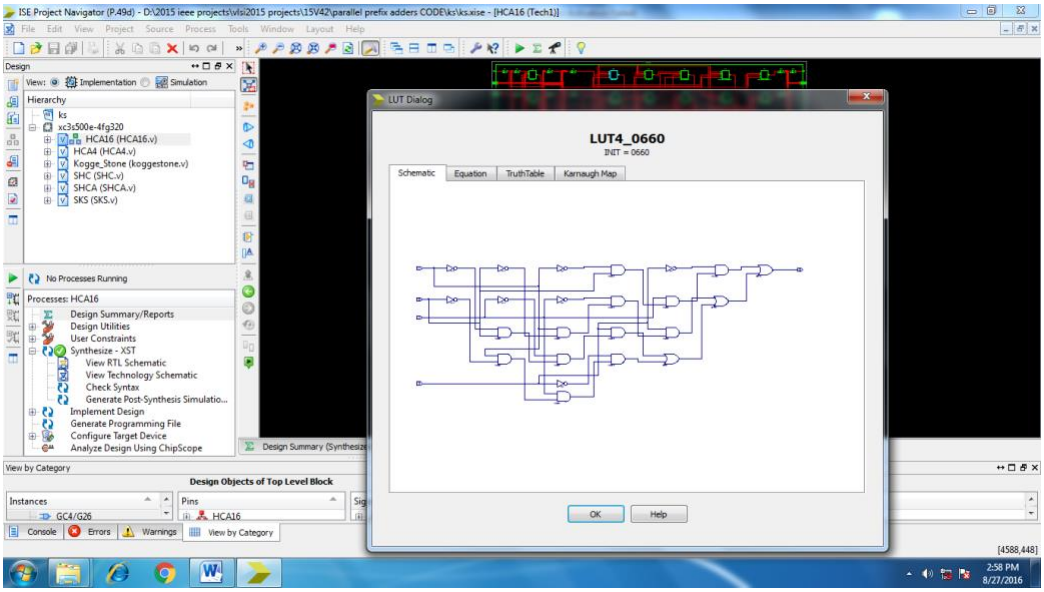


Figure 7.17: Internal block 16-Bit Han Carlson Adder

7.4 Synthesis Report

This device utilization includes the following.

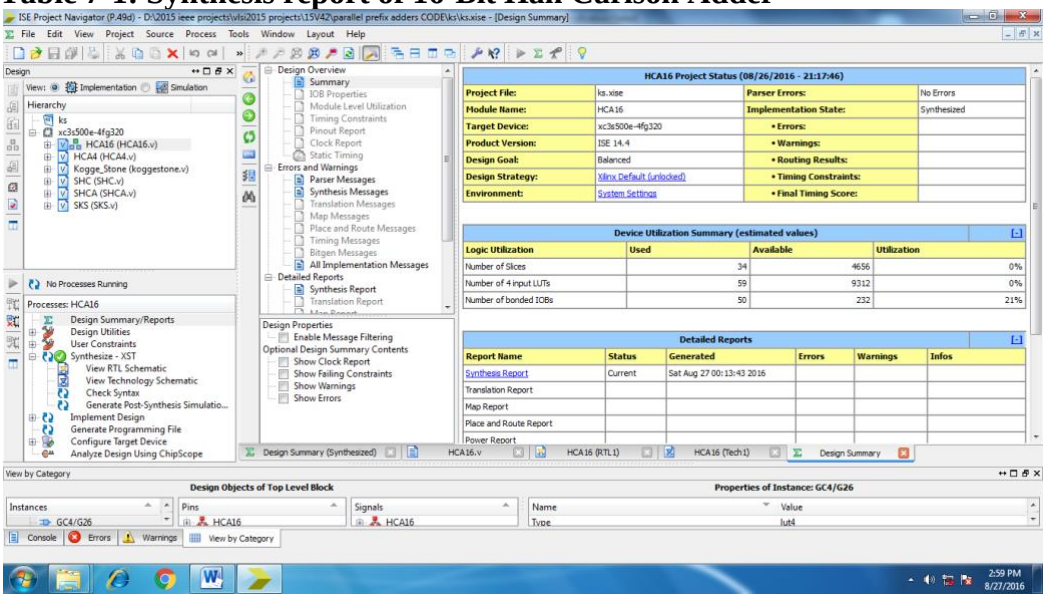
- Logic Utilization
- Logic Distribution
- Total Gate count for the Design

The device utilization summery is shown above in which its gives the details of number of devices used from the available devices and also represented in %. Hence as the result of the synthesis process, the device utilization in the used device and package is shown below.

Copyright © authors 2022

International Journal of Research (IJR)
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Vol. 9 Issue 01
January 2022

Table 7-1: Synthesis report of 16-Bit Han Carlson Adder



SIMULATION RESULTS

The corresponding simulation results of the adders are shown below.

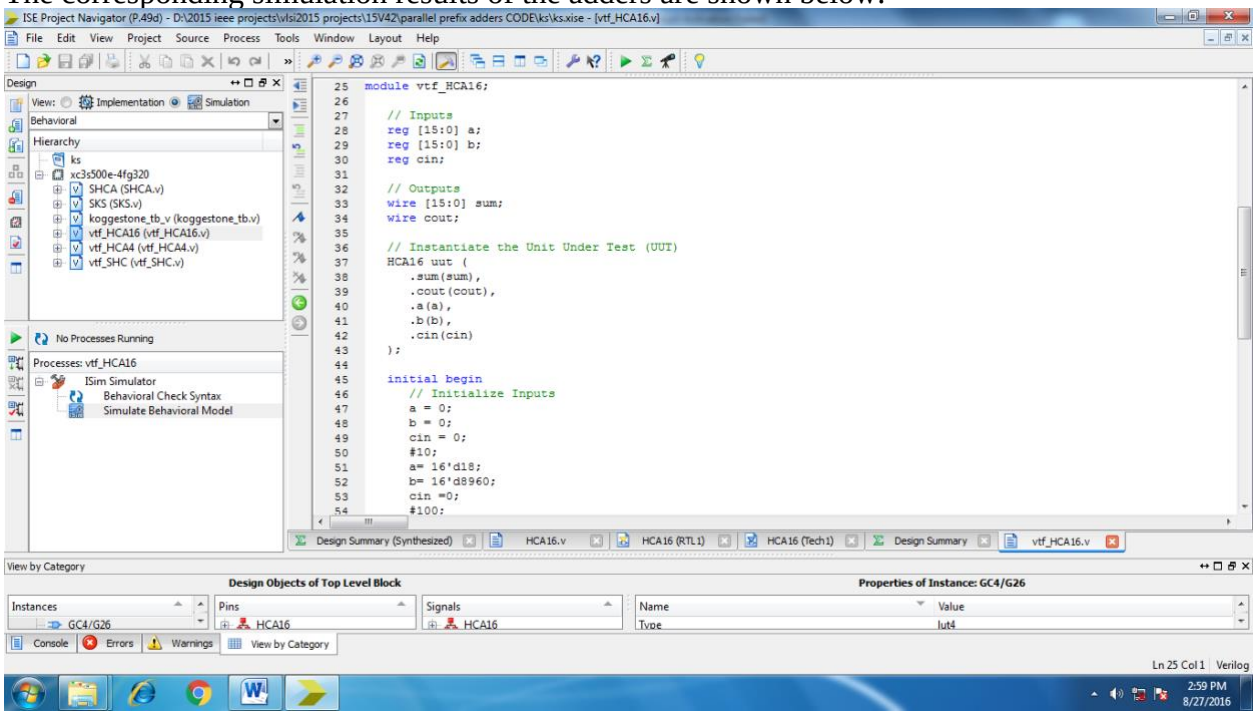


Figure 8-1: Test Bench for 16-Bit Han Carlson Adder

Copyright © authors 2022

p-ISSN: 2348-6848
e-ISSN: 2348-795X
Vol. 9 Issue 01
January 2022

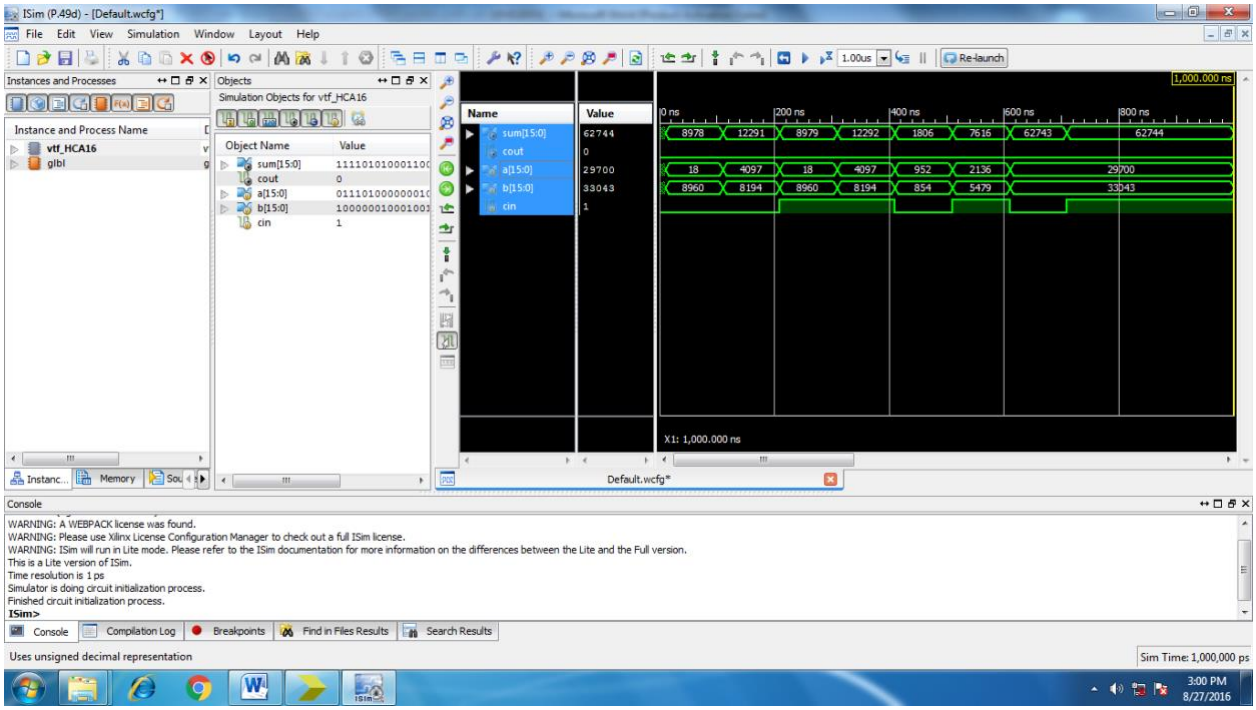


Figure 8-2: Simulated output for 16-Bit Han Carlson Adder

CONCLUSION

In this paper a novel variable latency Han-Carlson parallelprefix speculative adder for high-speed application is proposed. A new, more accurate, error detection network is introduced, which allows reducing the error probability compared to the previous approaches. An extensive set of implementation results for 65 nm CMOS technology shows that proposed Han-Carlson variable latency adders outperforms previously developed variable latency Kogge-Stone architectures. Compared with traditional, non-speculative, adders, our analysis demonstrates that variable latency Han-Carlson adders show sensible improvements when the highest speed is required; otherwise the burden imposed by error detection and error correction stages overwhelms any advantage. Additional work is required to extend the speculative approach to other parallel-prefix architectures, such as Brent-Kung, Ladner-Fisher, and Knowles.

REFERENCES

[1] I. Koren, Computer Arithmetic Algorithms. Natick, MA, USA: A K Peters, 2002.
[2] R. Zimmermann, “Binary adder architectures for cell-based VLSI and their synthesis,” Ph.D. thesis, Swiss Federal Institute of Technology, (ETH) Zurich, Zurich, Switzerland, 1998, Hartung-GorreVerlag.
[3] R. P. Brent and H. T. Kung, “A regular layout for parallel adders,” IEEE Trans. Comput., vol. C-31, no. 3, pp. 260–264, Mar. 1982.
[4] P. M. Kogge and H. S. Stone, “A parallel algorithm for the efficient solution of a general class of recurrence equations,” IEEE Trans. Comput., vol. C-22, no. 8, pp. 786–793, Aug. 1973.
[5] J. Sklansky, “Conditional-sum addition logic,” IRE Trans. Electron. Comput., vol. EC-9, pp. 226–231, Jun. 1960.

Copyright © authors 2022

p-ISSN: 2348-6848
e-ISSN: 2348-795X
Vol. 9 Issue 01
January 2022

[6] T. Han and D. A. Carlson, "Fast area-efficient VLSI adders," in Proc. IEEE 8th Symp.Comput.Arith. (ARITH), May 18–21, 1987, pp. 49–56.
[7] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," J. ACM, vol. 27, no. 4, pp. 831–838, Oct. 1980.
[8] S. Knowles, "A Family of Adders," in Proc. 14th IEEE Symp. Comput.Arith., Vail, CO, USA, Jun. 2001, pp. 277–281.
[9] S.-L. Lu, "Speeding up processing with approximation circuits," Computer, vol. 37, no. 3, pp. 67–73, Mar. 2004.
[10] T. Liu and S.-L.Lu, "Performance improvement with circuit-level speculation," in Proc. 33rd Annu.IEEE/ACM Int. Symp.Microarchit. (MICRO-33), 2000, pp. 348–355.
[11] N. Zhu, W.-L.Goh, and K.-S. Yeo, "An enhanced low-power highspeed Adder For Error-Tolerant application," in Proc. 2009 12th Int. Symp. Integr.Circuits (ISIC '09), Dec. 14–16, 2009, pp. 69–72.
[12] S. M. Nowick, "Design of a low-latency asynchronous adder using speculative completion," IEE Proc. Comput.Digit. Tech., vol. 143, no. 5, pp. 301–307, Sep. 1996.
[13] A. K. Verma, P. Brisk, and P. Ienne, "Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design," in Proc. Design, Autom., Test Eur. (DATE '08), Mar. 2008, pp. 1250–1255.
[14] A. Cilardo, "A new speculative addition architecture suitable for two's complement operations," in Proc. Design, Autom., Test Eur. Conf. Exhib. (DATE '09), Apr. 2009, pp. 664–669.
[15] K. Du, P. Varman, and K. Mohanram, "High performance reliable variable latency carry select addition," in Proc. Design, Autom., Test Eur. Conf. Exhib. (DATE '12), Mar. 2012, pp. 1257–1262.
[16] S. K. Mathew, R. K. Krishnamurthy, M. A. Anders, R. Rios, K. R. Mistry, and K. Soumyanath, "Sub-500-ps 64-b ALUs in 0.18- m SOI/ bulk CMOS: Design and scaling trends," IEEE J. Solid-State Circuits, vol. 36, no. 11, pp. 1636–1646, Nov. 2001.
[17] B. Parhami, Computer Arithmetic: Algorithms and Hardware Design. New York: Oxford Univ. Press, 2000.
[18] A. Tyagi, "A reduced-area scheme for carry-select adders," IEEE Trans. Comput., vol. 42, no. 10, pp. 1163–1170, Oct. 1993.

Copyright © authors 2022