

Implementation of Ripple Carry and Carry Skip Adders with Speed and Area Efficient

PUSHPALATHA CHOPPA¹, B.N. SRINIVASA RAO²

PG Scholar (VLSI Design), Department of ECE, Avanthi Institute of Engineering & Tech., Visakhapatnam, AP, India¹

Associate Professor, Department of ECE, Avanthi Institute of Engineering & Technology, Visakhapatnam, AP, India²

Abstract: The binary adder is the critical element in most digital circuit designs including digital signal processors (DSP) and microprocessor datapath units. As such, extensive research continues to be focused on improving the powerdelay performance of the adder. In VLSI implementations, parallel-prefix adders are known to have the best performance. 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. Parallel-prefix adders (also known as carry-tree adders) are known to have the best performance in VLSI designs. This paper investigates three types of carry-tree adders (the Kogge-Stone, sparse Kogge-Stone, and spanning tree adder) and compares them to the simple Ripple Carry Adder (RCA) and Carry Skip Adder (CSA). In this project Xilinx-ISE tool is used for simulation, logical verification, and further synthesizing.

Keywords: Include at least 4 keywords or phrases

I. INTRODUCTION

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.

Binary adders are one of the most essential logic elements within a digital system. In addition, binary adders are also In VLSI implementations, parallel-prefix adders are helpful in units other than Arithmetic Logic Units (ALU), known to have the best performance. Reconfigurable logic such as multipliers, dividers and memory addressing. Therefore, binary addition is essential that any been gaining in popularity in recent years because it offers improvement in binary addition can result in a improved performance in terms of speed and power over performance boost for any computing system and, hence, DSP-based and microprocessor-based solutions for many 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 1.1 demonstrates an example of an 8- bit binary add operation and how the carry chain is affected. This example shows that the worst popularity of mobile and portable electronics, which make case occurs when the carry travels the longest possible extensive use of DSP functions. However, because of the path, from the least significant bit (LSB) to the most significant bit (MSB).

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.

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.

Copyright to IJARCCE

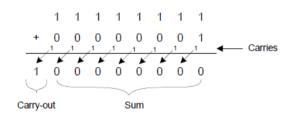


Fig.1. Binary Adder Example.

such as Field Programmable Gate Arrays (FPGAs) has involving mobile practical designs 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 with the growing structure of the configurable logic and routing resources in FPGAs, parallel-prefix adders will have a different In order to improve the performance of carry-propagate 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.



International Journal of Advanced Research in Computer and Communication Engineering Vol. 3, Issue 10, October 2014

The implementations that have been developed in this different functionality. The input lines to the logic block dissertation help to improve the design of parallel- prefix go into the LUT and enable it. The output of the LUT adders and their associated computing architectures. This gives the result of the logic function that it implements and has the potential of impacting many application specific the output of logic block is registered or unregistered purpose computer architectures. and general Consequently, this work can impact the designs of many computing systems, as well as impacting many areas of 1 size SRAM. Number of different possible functions for k engineers and science. In this paper, the practical issues input LUT is 2^2^k. Advantage of such an architecture is involved in designing and implementing tree-based adders that it supports implementation of so many logic functions, on FPGAs are described. Several tree-based adder however the disadvantage is unusually large number of structures are implemented and characterized on a FPGA memory cells required to implement such a logic block in and compared with the Ripple Carry Adder (RCA) and the case number of inputs is large. Carry Skip Adder (CSA). Finally, some conclusions and A simplified version of design flow is given in the flowing suggestions for improving FPGA designs to enable better diagram as shown in fig.3. tree-based adder performance are given.

II. FPGA IMPLEMENTATION AND DESIGN FLOW

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 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 1.2

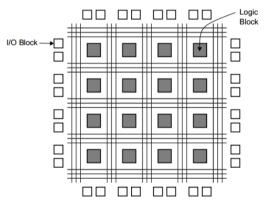


Fig. 2: FPGA interconnections

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

Copyright to IJARCCE

output from the LUT. SRAM is used to implement a LUT.A k-input logic function is implemented using 2^k *

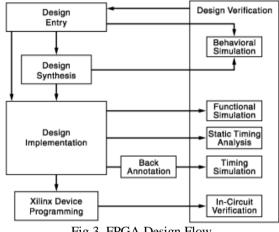


Fig.3. FPGA Design Flow

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.

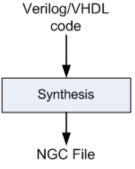


Fig.4. FPGA Synthesis



The process that translates VHDL/ Verilog code into a Carry-Select Adders (CSEA) device netlist format i.e. a complete circuit with logical Simple adders, like ripple-carry adders, are slow since the elements (gates flip flop, etc...) for the design. If the carry has to to travel through every full adder block. There design contains more than one sub designs, ex. to is a way to improve the speed by duplicating the hardware implement a processor, we need a CPU as one design due to the fact that the carry can only be either 0 or 1. The element and RAM as another and so on, then the synthesis method is based on the conditional sum adder and process generates netlist for each design element extended to a carry-select adder. With two RCA, each Synthesis process will check code syntax and analyze the computing the case of the one polarity of the carry-in, the hierarchy of the design which ensures that the design is sum can be obtained with a 2x1 multiplexer with the optimized for the design architecture, the designer has carry-in as the select signal. An example of 16-bit carryselected. The resulting netlist(s) is saved to an NGC (Native Generic Circuit) file (for Xilinx® Synthesis Technology (XST)).

III.EXISTING METHODS

Ripple-Carry Adders (RCA)

The simplest way of doing binary addition is to connect the carry-out from the previous bit to the next bit's carryin. Each bit takes carry-in as one of the inputs and outputs sum and carry-out bit and hence the name ripple-carry adder. This type of adders is built by cascading 1-bit full adders. A 4-bit ripple-carry adder is shown in Figure 2.3. Each trapezoidal symbol represents a single-bit full adder. In Fig.6, each two adjacent 4-bit blocks utilizes a carry At the top of the figure, the carry is rippled through the adder from cin to cout.

It can be observed in Figure 3.1 that the critical path, highlighted with a solid line, is from the least significant bit (LSB) of the input (a0 or b0) to the most significant bit (MSB) of sum (sn-1). The ripple carry adder is constructed by cascading full adders (FA) blocks in series. One full adder is responsible for the addition of two binary digits at any stage of the ripple carry. The carryout of one stage is fed directly to the carry-in of the next stage.

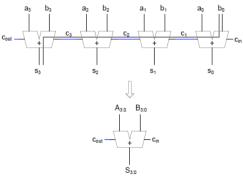


Fig.5.Ripple-Carry Adder[13].

Assuming each simple gate, including AND, OR and XOR gate has a delay of 2/ and NOT gate has a delay of 1/. All the gates have an area of 1 unit. Using this analysis and assuming that each add block is built with a 9-gate full adder, the critical path is calculated as follows.

ai, bi \rightarrow si = 10/\

ai, bi \rightarrow ci+1 = 9/\

ci \rightarrow si = 5/\

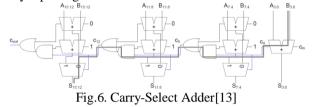
ci \rightarrow ci+1 = 4/\

The critical path, or the worst delay is

trca = $\{9 + (n-2) \times 4 + 5\} \land = \{f4n + 6\} \land$

As each bit takes 9 gates, the area is simply 9n for a n-bit RCA.

select adder is shown in Figure 3.2. In the figure, the adder is grouped into four 4-bit blocks. The 1-bit multiplexors for sum selection can be implemented as Figure 3.2 shows. Assuming the two carry terms are utilized such that the carry input is given as a constant 1 or 0:



relationship

ci+4 = c0i+4 + c1i+4. ci

Temporary sums can be defined as follows.

s0 i+1 = ti+1 . c0 i

s1 i+1 = ti+1 . c1 i

The final sum is selected by carry-in between the

temporary sums already calculated.

 $si+1 = cj \cdot s0 i+1 + cj \cdot s1 i+1$

Assuming the block size is fixed at r-bit, the n-bit adder is composed of k groups of r-bit blocks, i.e. $n = r \times k$.

The critical path with the first RCA has a delay of (4r +5)/\ from the input to the carry-out, and there are k - 2blocks that follow, each with a delay of 4/ for carry to go through. The final delay comes from the multiplexor. which has a delay of 5/, as indicated in Figure 2.5. The total delay for this CSEA is calculated as

 $tcsea = 4r + 5 + 4(k - 2) + 5/ = {4r + 4k + 2}/ (k - 2) + 5/ = {4r + 4k + 2}/ (k - 2) + 5/ (k$

The area can be estimated with (2n - r) FAs, (n - r)multiplexors and (k - 1) AND/OR logic. As mentioned above, each FA has an area of 9 and a multiplexor takes 5 units of area. The total area can be estimated

9(2n - r) + 2(k - 1) + 4(n - r) = 22n - 13r + 2k - 2

The delay of the critical path in CSEA is reduced at the cost of increased area. For example, in Figure 3.2, k = 4, r = 4 and n = 16. The delay for the CSEA is 34/\ compared to 70 \land for 16-bit RCA. The area for the CSEA is 310 units while the RCA has an area of 144 units.

The delay of the CSEA is about the half of the RCA. But the CSEA has an area more than twice that of the RCA. Each adder can also be modified to have a variable block sizes, which gives better delay and slightly less area.

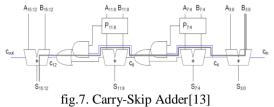
Carry-Skip Adders (CSKA)

There is an alternative way of reducing the delay in the carry-chain of a RCA by checking if a carry will propagate through to the next block. This is called carry-skip adder

ci+1 = Pi:j Gi:j + Pi:j cj



International Journal of Advanced Research in Computer and Communication Engineering Vol. 3, Issue 10, October 2014



The carry-out of each block is determined by selecting the carry-in and Gi:j using Pi:j. When Pi:j = 1, the carry-in cj is allowed to get through the block immediately. Otherwise, the carry-out is determined by Gi:j. The CSKA has less delay in the carry-chain with only a little additional extra logic.

Further improvement can be achieved generally by making the central block sizes larger and the two-end block sizes smaller.

Assuming the n-bit adder is divided evenly to k r-bit blocks, part of the critical path is from the LSB input through the MSB output of the final RCA.

The first delay is from the LSB input to carry-out, which is 4r + 5. Then, there are k - 2 skip logic blocks with a delay of 3/. Each skip logic block includes one 4-input AND gate for getting Pi+3:i and one AND/OR logic.

The final RCA has a delay from input to sum at MSB, which is 4r+6. The total delay is calculated as follows.

tcska = $\{4r + 5 + 3(k - 2) + 4r + 6\} \land$ = $\{8r + 3k + 5\} \land$

The CSKA has n-bit FA and k - 2 skip logic blocks. Each skip logic block has an area of 3 units. Therefore, the total area is estimated as 9n + 3(k - 2) = 9n + 3k - 6.

IV. RESULTS

The results pertaining to the proposed work are given in this section. Output representation regarding koggee stone algorithm for 16 bit, 32 bit and 64 bit are presented in Fig.8 through Fig.10.

Results involving implementation of RCA 16 bit, RCA 32 bit and RCA 64 bit and Carry Skip for 16 bit, carry skip 32 bit and carry skip 64 are presented in the subsequent figures.

	rch Simulation Windo	w Help			
APTIPEXX C . S. P.		6800 A¥	M	11 V 00	
arces for, Behavioral Site, Number of LUTs randoty € toges Not and stole. a Calsole.	* (x) b(15.0) (x) cin (x) (x) sum(15.0)		5 7	2450 ns 2850 ns 3250 1 1 1 1 1 6 8 8 14	ns 3650 ns 1 1 1
Source of Separations Octoraries	3				
Piocesses Therachy-kowfbw Process "Generate Expected Simu Compiling Verilog file "D:/Xilis	2 Design Sammary	completed suc			
•	mscle 🙀 Find in Files				,

Fig.8. Koggee stone 16 bit

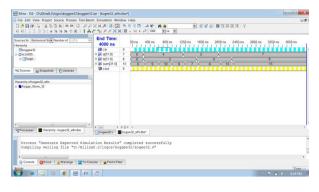


Fig.9. Kogeee stone 32 bit

File Edit View Project Source Process Test Ber	ch Simulation W	indow Help										G
19 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1				<i>8</i> 6 8 6		• • •	7 😥 🐼	70 30 30 30	9			
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○	tach	₽₽XX	≣ + (=)	7 1000	• na •							
outes for: Behavioral Sie Number of LUTS e feranchy @loggeed4 @xc3550. @ @kpse. Charge.	End Time: 4000 ns H plt a(63:0) H plt b(63:0) plt t(63:0) plt cin H plt sum(63:0) plt cout	10 (0) 9 (6) 0	450 ns 3	850 ns 3	1250 ns	1.1.1	X	ns 245 7 16	8	0 ns 3250 X 15 X	X	50 ns 10 9
	1											
rocesses	1											
Add Existing Source	1											
Create New Source												
View Generated TestBench As HDL Add Test Bench To Preject												
Mod Test batch To Project												
Generate Expected Simulation Results												
Simulate Behavioral Model												
	<pre> •</pre>											
C Processes	koggee64 v	T Design!	mmary 🕿	sgee64_wf.ts	w Sin	ulation	kogea64	with				
Stopped at time : 4.000 us : Fil	e "D:/Xilinx	8.2/1000	(konnee 64	/konee64	of the	Line S	50					
Stopped at line=50 file name=D:/												
<												
Console OEnors AWarnings Tol Co	nsole 🛛 🙀 Find in F	Tes Si	Console - kog	+64_wf								
												Time
-tv												

Fig.10.Kogeee stone 64 bit

E File Edit View Project Source Process 1	a16bitise - [rca16_w(tbw*]	
		,
	2 PPXXP 8 2 5 8 8 7 4 9 8 8 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	
十百四当 月戸田田田田田田田	μ ² 1000 € ns €	
	End Time:	
Sources for Behavioral Si Number of LUTs	4000 ns 50 ns 450 ns 850 ns 1250 ns 1650 ns 2050 ns 2450 ns 2850 ns	3250 ns 3650 ns
Herarchy		
The (sca32bit v)		<u>X 2</u>
Arca (rca 64 bit fbw)	* (x)	<u>8 X 1</u>
rca16_wf(rca16_wftbw)	All cin 1	
III (rca32 (rca32 tbw)	* * \$4(\$(15.0) 4 0 4 8 7 (\$\)10	15 (8) 4
in California (rcalifbitt.tbw)	Matery 0	
III ReaG4wf (reaG4wf tbw)	*	
Scurces and Snapshots Dubraries		
Construction of Construction		
Herarchy of rca16_wt		
x 🐚:ca_16		
<[F 4 m F 4 m F 4	
	12 Design Summary and Ica 16 will but	
	Toold county	
≪Processes ■ Hierarchy-rca15_wf.tbw		
RCProcesses Henarchy-rca15_wf.tbw		
*CProcesses Herarchy-scale_wf.bw Process "Editing Thw file"	completed successfully	
CProcesses Herarchy-scale_wf.bw Process "Editing Thw file"	completed successfully /xllinx8.2/logos/ccal6bit/ccal6bit.v	
CProcesses Herarchy-scale_wf.bw Process "Editing Thw file"	completed successfully /Xilinx8.J/logos/real6bit/real6bit.v	
CProcesses Herarchy-scale_wf.bw Process "Editing Thw file"	completed successfully /Xilinx8.2/logos/rcal6bit/rcal6bit.v	
<pre>Processes Hearthy.noi6_withw Process "Editing Thw file" Compiling verilog file "D:/ </pre>	completed successfully XIIInxS.2/logos/rcal6bi//rcal6bit.v*	
<pre>Processes Hearthy.noi6_withw Process "Editing Thw file" Compiling verilog file "D:/ </pre>	/Xilinx8.2/logos/rcal@bit/rcal@bit.**	Time

Fig.11.Rca 16 bit adder

File Edit View Project Source Process Test Ben												100
RA SINGX SEA				-C 10 A			01.02.02	(C) III (C)	here o	0.0		
(1) (1) (1) (1) (1) (1) (1) (1) (1) (1)										00		
Courtes for Behavioral Ste Number of LUTs + Haracty Transbit C no.1501 C 0.015600-410320	End Time: 4000 ns = pt a(31:0) = pt a(31:0)	2	2050 ns	2250 ns	2450 ns	2650 ns	2850 ns	3050 ns	3250 ns	3450 ns	3650 ns	3850 ns
i:	3.0 cin * (34 s(31.0) 3.0 cry	0 8 0			16		(15)	Y		X 8 X	10	X 9 X8
at Sources (Snapshots DLbranes												
Herarchy of rca32 × @rcs32bit	1											
c												
CProcesses Heranhy-rcs32.fw		4.11.9										
Frocess "Generate Expected Simul. Compiling verilog file "D:/Xilin		" cor	pleted	success		1 1 mass	Y	2 DW				
Consula OErrors & Warrings Tri Con	scle 🔥 Find in Fil	61										,
and a second sec												Time:

Fig.12.Rca 32 bit adder



International Journal of Advanced Research in Computer and Communication Engineering Vol. 3, Issue 10, October 2014

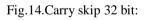
File Edit View Project Source Process	Test Bench Simulation Window Help
	- アクズズダ目気 ちゅうご AF AA
arces for Behavioral Site Number of UUT earchy archafoda +fg220 archafoda +fg220	Image: second time: Solution Solution </th
(Sources gy Snapshots Clubraries	
Processes	Designment (Serger (Serger (Serger Serger)
Process "Generate Expecte Compiling verilog file "D	Simulation Results" completed successfully /xilinx8.2/logos/rca16bit/rca32.v"
·	
	Tcl Consola (a, Find in Files
A	Time.

Fig.12.Rca 64 bit adder

File Edit View Project Source Process Test Ber	ch Simulation Window	Help				1
200日日(2000日) 1日1日(100日) 1日1日(100日) 1000日(100日) 1000日(100日) 1000日(100日)	00×20 🗷 🔊 🤊	8 0 0 4 9	Ø 🕷	- AF AF AF AB TO SO A	CTC 9 00	
Sasteet for Synthesisting Vinnes of LUTs Seastly Resources Preserve Completing DataStODe 45020 With Stifforesist (End Time: 4000 ns ∞ pl a(15:0) 6 ∞ pl b(15:0) 2 pl c1 0 ∞ pl s(15:0) 8 pl c1 0 ∞ pl s(15:0) 8 pl c0 0	50 ns 450 ns 0 4 0 5 0 9	850 ns 1250 ns		2450 ns 2850 ns 32 6 2 8	
t Sources B Stepsters C Libraries						
III III III III IIIIIIIIIIIIIIIIIIIIII	 (m) → < in ∑ Design Summary 					
Started : "Creating Thw file". Compiling verilog file "D:/Xilin		akipp/cskip léb	it.v"			
Console OEmors AWarrings Tol Co	nsole 🙀 Find in Files					Time

Fig.13.Carry skip 16 bit

ch Simulation Wir ××× ≠ ≠ ≥ ≥ •ms • End Time: 4000 ns ≈≥ k a(31.0) ≈ 20 (31.0) ≈ 20 (31.0)	50 ns	450 ms		288	1 18 % 10 1	8: 92 V	00	2
4000 ns		450 ms	11000 00100					
20 c1 20 c1 ≈ 30 s(31.0) 30 c0	9 0 6 0 15 0 0	× 5 5 × 10	3 9				9 6 15	
1								
		unit mat	ransko 72 ftar					
escle Rind in Fi	les							
	tin co time to time	triani → ciù + ciu	Ver en	Mol 0 1 m 1 m 1 m 1 m 1 m 1 m 1 m 1 m 1 m 1 m	Mod 0 r m r m r m r m r m r m r m r m r m r m	Moo 0 1 m 1 m 1 m 1 m 1 m 1 m 1 m 1 m 1 m 1 m	Mach 0 r → r + r + r + r + r + r + r + r + r +	Mod 0 f int i if joint g Comp Revery company for any state all.J/Lopen/cattrystips/cattry 32811.v*



File Edit View Project Source Process Test Bend	ch Simulation Window Help	
1401 / 2 200x 00 2 22 1401 / 22x 0 . 4 7 100		
surces for: Synthesis/Ingle Number of LUTs • terachy Resources Preserve Comprisipo g parts6006-49200 @ DataS600-49200	4000 ns b) ns 60 ns 60 ns 120 ns 120 ns 200 ns </th <th>3250 ns 3650 ns</th>	3250 ns 3650 ns
C Sources By Stapshitts Dibraries		
Herarchy of Carry G4. © Collip G40 z		
Processes Herardy candidate	й (m) — х кфа с	
Chocesses	😰 Design Summary 🖉 carryskip 16 ibw" 🖉 carryskip 32 ibw" 🖉 carry64 ibw"	
Started : "Creating Tbw file". compiling verilog file "D:/Xilina Console @Erors _AWaningsCTelCer	x8.2/logos/carryskipp/cskip 64bit.v*	

Fig.14.Carry skip 64 bit

V. CONCLUSION

The results presented in the preceding section significantly produced the efficiency of the proposed methodology. The performance evaluation of the simulated models would be good starting point of further research.

REFERENCES

- P. Ndai, S. Lu, D. Somesekhar, and K. Roy, "Fine-Grained Redundancy in Adders," Int. Symp. on QualityElectronic Design, pp. 317-321, March 2007.
- [2] T. Lynch and E. E. Swartzlander, "A Spanning Tree Carry Lookahead Adder," IEEE Trans. on Computers, vol. 41, no. 8, pp. 931-939, Aug. 1992.
- [3] D. Gizopoulos, M. Psarakis, A. Paschalis, and Y. Zorian, "Easily Testable Cellular Carry Lookahead Adders," Journal of Electronic Testing: Theory and Applications 19, 285-298, 2003.
- [4] S. Xing and W. W. H. Yu, "FPGA Adders: Performance Evaluation and Optimal Design," IEEE Design & Test of Computers, vol. 15, no. 1, pp. 24-29, Jan. 1998.
- [5] M. Bečvář and P. Štukjunger, "Fixed-Point Arithmetic in FPGA," Acta Polytechnica, vol. 45, no. 2, pp. 67-72, 2005.
- [6] N. H. E. Weste and D. Harris, CMOS VLSI Design, 4th edition, Pearson–Addison-Wesley, 2011.
- [7] R. P. Brent and H. T. Kung, "A regular layout for parallel adders," IEEE Trans. Comput., vol. C-31, pp. 260-264, 1982.
- [8] D. Harris, "A Taxonomy of Parallel Prefix Networks," in Proc. 37th Asilomar Conf. Signals Systems and Computers, pp. 2213–7, 2003.
- [9] P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," IEEE Trans. on Computers, Vol. C-22, No 8, August 1973