ECE 429, Spring 2013
Final Project: Design and Synthesis of Carry Propagation Adders

04/08/2013

Project duration:	*4 weeks - 04/08 – 05/10*
Oral Presentation:	*04/29 Monday or 04/30 Tuesday (Your Regular Lab Session)*
Final Report Due:	*05/10 (Friday) Midnight Chicago time*

1 Design Objective

This project introduces VLSI design concepts including datapath circuit design, standard cell based design flow, and design validation and verification through construction of fast adder architectures in Verilog, to be synthesized using commercial EDA tools from Synopsys and Cadence Design Systems. The intent is to show, first, how to construct a nontrivial adder hardware; second, how to make design trade-offs, e.g. performance, cost, and design time, through architectural exploration; and third, how the EDA tools transform the design implementation from higher abstraction levels, e.g. Verilog, to lower abstraction levels, e.g. layout, through cell-based design flow, what the differences are among the implementations and how to verify their correctness at the different abstraction levels.

In this project, you will be exploring four different adder architectures with different topologies and different word-lengths (8-bit and 32-bit). Therefore, you will be able examine the impact of the word-length on the delay (performance) and also the area (cost). You can also observe how each adder's performance scales as the word-length increases. A reference 8-bit carry-ripple adder design and its stimulus file is provided. You can compare your results to this reference adder design. Finally, as a bonus problem, you can implement an 16x16 array multiplier. All the designs should be synthesized through the cell-based flow. A pre-defined performance constraint must be met after the synthesis and the correctness of the design at different abstraction levels should be verified.

This project can be done as a team with a single partner. Only a single report needs to be submitted, with both students' signatures on the cover page.

Discussions with other students are encouraged. However, all the writings, results, and screen shots should be by yourself. COPY without proper CITATION, and extensive COPY from other materials including but not limited to project instructions and textbooks, will be treated as PLAGIARISM and called for DISCIPLINARY ACTION. $\boxed{\textbf{NEVER share your reports with others}}$.

2 Architectural Exploration of Adder Architectures

Your first task is to implement the 8-bit carry ripple adder design provided to you (`adder8.v`). However, you will need to modify it for 32-bit word-length. 32-bit ripple adder performance will suffer since the carry has to be propagated through 32 stages. The performance of the ripple adder can be used as a reference point for the remaining adder architectures. Next, you are required to implement three fast carry propagate adder architectures which are discussed in Chapter 11 of the textbook:

1. Carry-Skip Adder
2. Carry Select Adder
3. Prefix Adder - Sklansky

Each of these adders will be implemented as 8-bit, and also 32-bit adders. Therefore, including the carry ripple adder, you will design and synthesize 8 adders in total. Note that you will need to decide the parameter for certain adder architectures, e.g. for implementing 32-bit carry-skip adder, you can choose among the one with 8 4-bit groups or the one with 4 8-bit groups.

All adder implementations need to be done using *structural Verilog.*

2.1 Carry Skip Adder

The objective of Carry-Skip Adder is to reduce the worst case delay by reducing the number of full-adder cells through which the carry has to propagate. Consequently, to reduce the length of the propagation of the carry, a skip network is provided for each m bits so that when a carry is propagated by this group, the skip network makes the carry bypass the group. See Figure 1 and Chapter 11 for more details.

2.2 Carry Select Adder

Carry Select Adder is based on the fact that the main component in the delay of the carry-ripple adders is the propagation of the carry, so that to obtain the sum of bit i, it is necessary to wait until the carry has propagated from bit 0 to bit i. Because of this, the idea of carry select adder is to compute in parallel two conditional sums, one for a 0-carry and one for a 1-carry, and then select among them when the carry is available. Basic principle is to divide the adder intro groups of m-bits and to compute

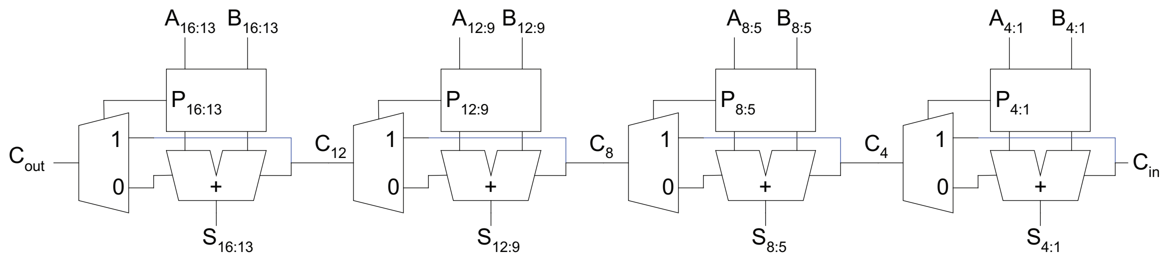

Figure 1: 16-bit Carry Skip Adder

for each group two conditional sums and carry-outs. See Figure 2 and Chapter 11 for more details.

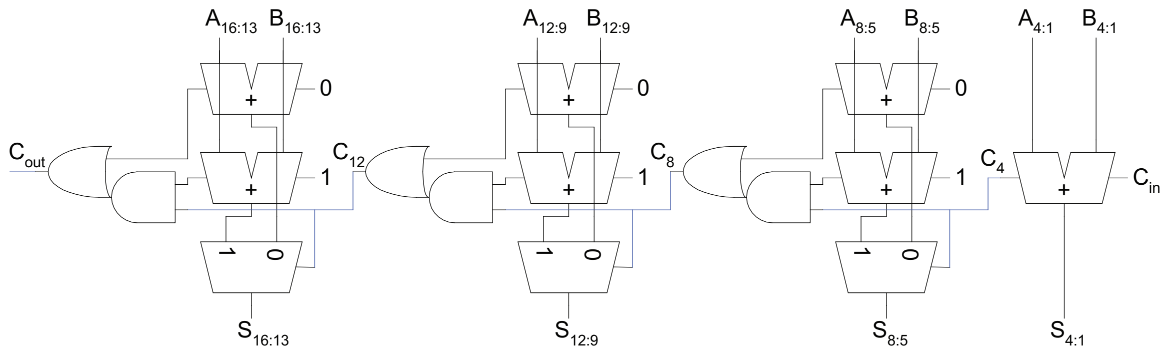

Figure 2: 16-bit Carry-Select Adder

2.3 Prefix Adder: Sklansky

For prefix tree adders such as Sklansky, equations are factored into generate (G) and propagate (P) functions. For groups spanning bits i:j, we can write:
$G_{i:j} = G_{i:k} + P_{i:k} \bullet G_{k-1:j}$
$P_{i:j} = P_{i:k} \bullet P_{k-1:j}$
Sum can be obtained by:
$S_i = P_i \oplus G_{i-1:0}$
where base cases are given as:
$G_{i:i} = G_i = A_i \bullet B_i$
$P_{i:i} = P_i = A_i \oplus B_i$

Figure 3 shows the black cell, gray cell and buffer elements which are used to implement the group G and P functions. There are many variations on tree adders with respect to allocation of black and gray cells. One example is Sklansky as shown in Figure 4.

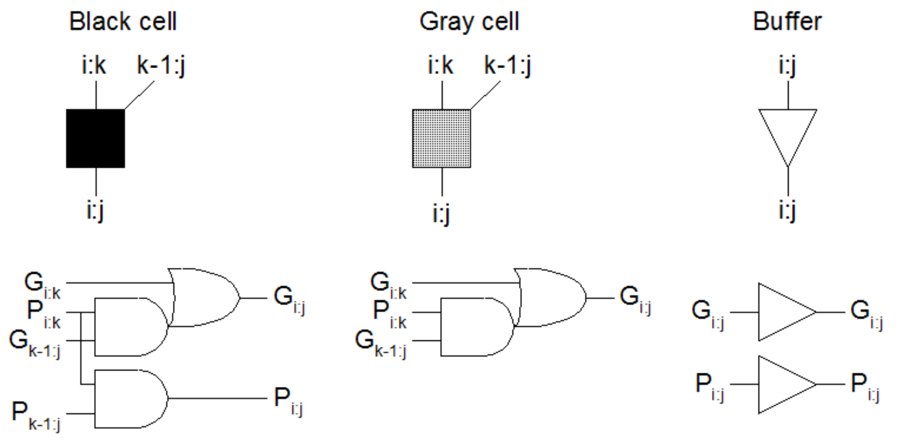

Figure 3: PG Diagram Notation for Prefix Tree Adders

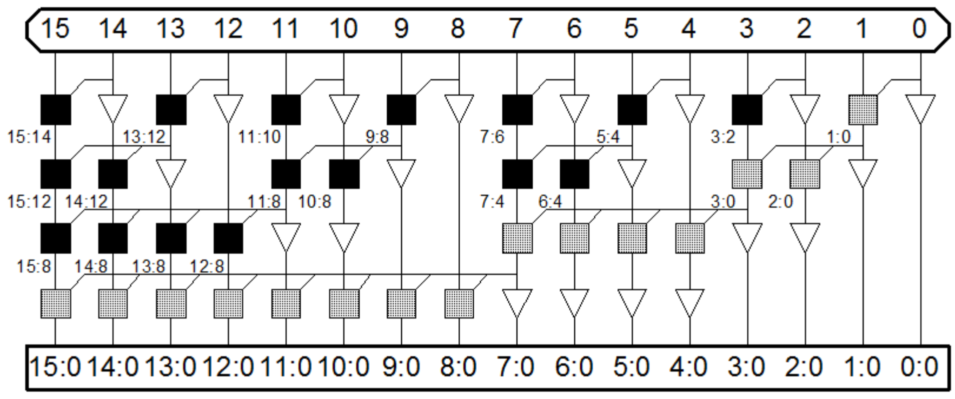

Figure 4: 16-bit Parallel Prefix Tree Adder - Sklansky

3 Automatic Synthesis of Adder Architectures

We follow the standard cell based design flow to synthesize the CPU. Please refer to the online tutorial (Tutorial IV) for detailed information. Pay attention to the circuit characteristics before and after P&R. Is the timing specification met? Determine the critical path delay of the synthesized design.

3.1 Functional Validation

We must ensure that there is no bug in the adder design before synthesis. We validate the functionality of the adders by running testing programs. You will create a stimulus file similar to the provided test file for carry ripple adder (`adder8test.v`)and you should validate that the output is as expected for each implementation. Make sure to include reasonable number of input patterns for the testbench.

3.2 Timing Specification

For logic synthesis with Synopsys Design Compiler, please set the desired clock frequency to be 250Mhz. Synopsys Design Compiler will automatically pass the specification to Cadence Encounter for place & route.

3.3 Equivalence Checking

Please follow the instructions in Tutorial IV to verify the correctness of the synthesized results by equivalence checking, namely `adder8.v` and `final.v` should be equivalent.

4 Bonus Design Problem

As an option, you can also implement a multiplier design. Specifically an 16x16 array multiplier. Figure 5 shows 4x4 multiplication hardware which can be used as the starting point. Again, implementation will be done using structural Verilog code only. You need to provide testbench and verify the functionality of the multiplier. The bonus part will earn you up to 20 more points added to overall project grade.

5 Project Evaluation

The final project will contribute to 14% of the course grade, plus the bonus which contributes an additional 2%. You will be evaluated from two aspects – oral evaluation and final report. evaluation.

Oral Evaluation	Final Report	Bonus Problem
70	70	20

5.1 Oral Evaluation

Oral evaluation will be performed in the last week of the lectures. Each group should prepare a 5 to 10 minutes interview with the instructor/TA to explain the motivation, implementation, and evaluation of the adder designs. The following topics will be covered.

- Circuit level
  - Implementation of components and combinational circuits
  - Critical path and performance
- Synthesis flow
  - Stages in the synthesis flow and tools used

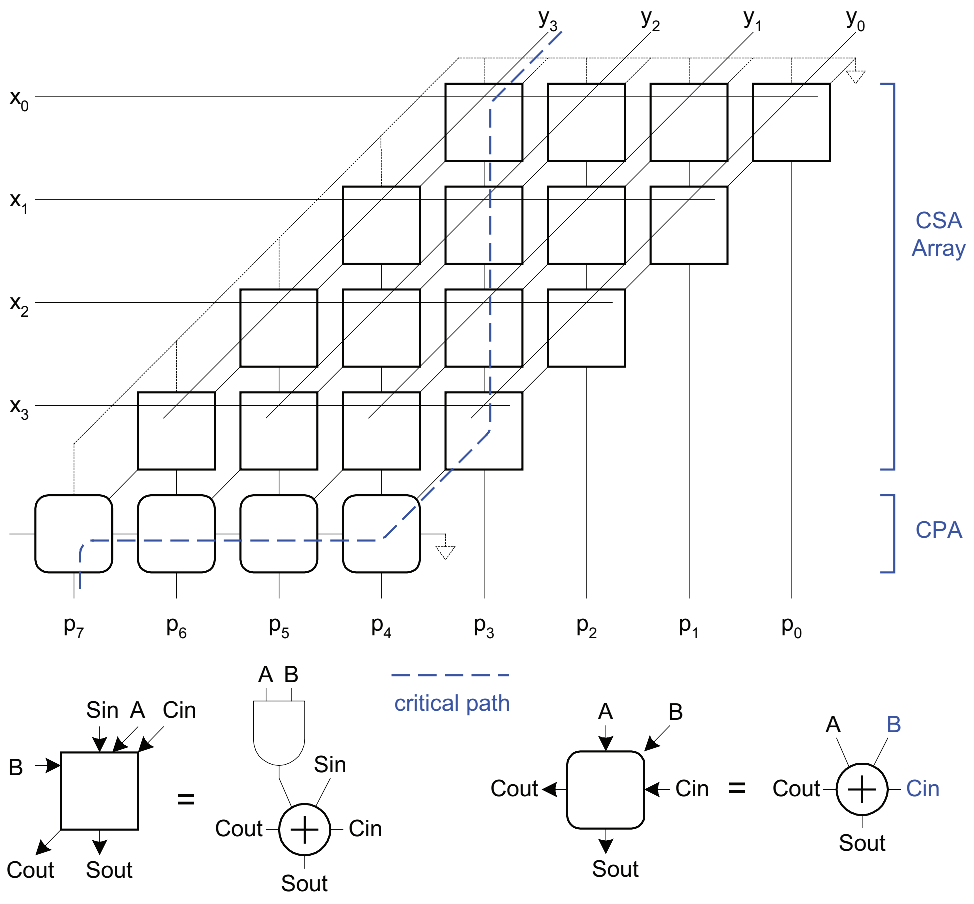

Figure 5: Array Multiplier Architecture

  – Input/output of each stage

- Functional validation and verification

  – Methods and tools to validate and verify the functionality of the designs at each design stage

For main campus students, the evaluation is during your designated lab time in the last week of lectures.

For online and remote students, a PowerPoint presentation should be prepared to cover the topics and is due at the same time of the final report.

5.2 Final Report

You are required to submit a stand-alone report to summarize the whole final project. This report should include the necessary results and analysis as required in the previous sections, and be readable without reference to other materials, including but not limited to the project instructions, the textbook, and the progress report.

The report should be limited to 20 pages at most and be legible when printing on letter-size papers. Please use common sense to format your report, e.g. report with small fonts/figures will not be graded. Your codes/screen shots/results can be attached as the appendix which will not count toward the 20-pages limit. However, you should discuss them within the 20-pages limit.

All the writings, results, codes, and screen shots should be by yourself. COPY without proper CITATION, and extensive COPY from other materials including but not limited to project instructions and textbooks, will be treated as PLAGIARISM and called for DISCIPLINARY ACTION. You should clearly separate your contribution from existing works, e.g. to separate your implementation from the implementation that is already given. $\boxed{\textbf{NEVER share your reports with others}}$.

The following sections are recommended as an effective way to organize your writing in your reports.

- **Abstraction**
  In less than 100 words, briefly discuss your contributions in the project.

- **Introduction**
  Summarize the motivation of the project. Highlight the engineering and design challenges in this project and the methods to overcome these challenges.

- **Background**
  Give concise descriptions of the adder architectures. Note that you should cite various references properly, e.g. the project instructions and the textbook.

- **Architectural Exploration of Adders**
  Discuss the trade-offs among the adders and then motivate your choice of adder architecture and parameters. Explain your implementation in plain English, possibly with pieces of Verilog code.

- **Multiplier Design**
  Discuss your solution to the bonus design problem if you decide to solve it.

- **Functional Validation and verification**
  Discuss the approaches taken to validate and verify your designs, e.g. RTL simulations with the test bench and the equivalence checking between the two adder designs. Justify your claims of correctness with simulation results or equivalence checking reports.

- **Synthesis Results**
  Discuss the synthesis flow including RTL, post-synthesis, place & route, etc. Explain the differences of the designs at various design stages. **Compare your designs in terms of performance and cost.** Create charts/tables to tabulate the performance results and scaling in terms delay and area when the adder is expanded from 8 to 32 bits. Make sure you reach a conclusion with respect to best adder architecture.

- **Conclusion and Future Work**
  Summarize your contributions and discuss possible future works.
- **Appendix**
  Verilog code/screen shots/results listing.
- **References**