AREA-DELAY-POWER EFFICIENT CARRY SELECT ADDER

K SIDDESWARA REDDY 1*, B HARIKA 2*, T ARUN PRASAD 3*
1. M.Tech - Student, Dept of ECE, SRI SARADA INSTITUTE OF SCIENCE & TECHNOLOGY.
2. Asst. Prof, Dept of ECE, SRI SARADA INSTITUTE OF SCIENCE & TECHNOLOGY.
3. Assoc. Prof, Head - Dept of ECE, SRI SARADA INSTITUTE OF SCIENCE & TECHNOLOGY.

ABSTRACT

In this brief, the logic operations involved in conventional carry select adder (CSLA) and binary to excess-1 converter (BEC)-based CSLA are analyzed to study the data dependence and to identify redundant logic operations. We have eliminated all the redundant logic operations present in the conventional CSLA and proposed a new logic formulation for CSLA. In the proposed scheme, the carry select (CS) operation is scheduled before the calculation of final-sum, which is different from the conventional approach. Bit patterns of two anticipating carry words (corresponding to cin = 0 and 1) and fixed cin bits are used for logic optimization of CS and generation units. An efficient CSLA design is obtained using optimized logic units. The proposed CSLA design involves significantly less area and delay than the recently proposed BEC-based CSLA. Due to the small carry-output delay, the proposed CSLA design is a good candidate for square-root (SQRT) CSLA. A theoretical estimate shows that the proposed SQRT-CSLA involves nearly 35% less area–delay–product (ADP) than the BEC-based SQRT-CSLA, which is best among the existing SQRT-CSLA designs, on average, for different bit-widths. The application-specified integrated circuit (ASIC) synthesis result shows that the BEC-based SQRT-CSLA design involves 48% more ADP and consumes 50% more energy than the proposed SQRT-CSLA, on average, for different bit-widths. Index Terms—Adder, arithmetic unit, low-power design.

INTRODUCTION

Low-power, area-efficient, and high-performance VLSI systems are increasingly used in portable and mobile devices, multi standard wireless receivers, and biomedical instrumentation. An adder is the main component of an arithmetic unit. A complex digital signal processing (DSP) system involves several adders. An efficient adder design essentially improves the performance of a complex DSP system. A ripple carry adder (RCA) uses a simple design, but carry propagation delay (CPD) is the main concern in this adder. Carry look-ahead and carry select (CS) methods have been suggested to reduce the CPD of adders. A conventional carry select adder (CSLA) is an RCA–RCA configuration that generates a pair of sum words and output carry bits corresponding the anticipated input-carry (cin = 0 and 1) and selects one out of each pair for final-sum and final-output-carry.

A conventional CSLA has less CPD than an RCA, but the design is not attractive since it uses a dual RCA. Few attempts have been made to avoid dual use of RCA in CSLA design. Kim and Kim used one RCA and one add-one circuit instead of two RCAs, where the add-one circuit is implemented using a multiplexer (MUX). He et al. proposed a square-root (SQRT)-CSLA to implement large bit-width adders with less delay.
In a SQRT CSLA, CSLAs with increasing size are connected in a cascading structure. The main objective of SQRT-CSLA design is to provide a parallel path for carry propagation that helps to reduce the overall adder delay. Ramkumar and Kittur suggested a binary to BEC-based CSLA. The BEC-based CSLA involves less logic resources than the conventional CSLA, but it has marginally higher delay.

A CSLA based on common Boolean logic (CBL) is also proposed in and The CBL-based CSLA of involves significantly less logic resource than the conventional CSLA but it has longer CPD, which is almost equal to that of the RCA. To overcome this problem, a SQRT-CSLA based on CBL was proposed in However, the CBL-based SQRT-CSLA design of involves more logic resource and delay than the BEC-based SQRT-CSLA.

We observe that logic optimization largely depends on availability of redundant operations in the formulation, whereas adder delay mainly depends on data dependence. In the existing designs, logic is optimized without giving any consideration to the data dependence.

In this brief, we made an analysis on logic operations involved in conventional and BEC-based CSLAs to study the data dependence and to identify redundant logic.

Based on this analysis, we have proposed a logic formulation for the CSLA. The main contributions in this brief are logic formulation based on data dependence and optimized carry generator (CG) and CS design.

HARDWARE DESCRIPTION LANGUAGE

Introduction of HDL’s

Hardware description languages (HDLs), mainly to describe logic equations to be realized in programmable logic devices (PLDs).in the 1990s, HDL usage by digital systems designers accelerated as PLDs, CPLDs, and FPGAs became inexpensive and common place. Designers turned to HDLs as a means to design individual modules within a system-on-chip.

The important innovations in HDLs occurred in the mid-1980s, and were the developments of VHDL and VERILOG HDL became popular. There are several steps in an HDL based design process, often called the design flow. These steps are applicable to any HDL based design process and are shown in figure

Figure 2.1: Steps in an HDL Based Design Flow

In any design, specifications are written first. Specifications describe the functionality, interface and overall architecture of the digital circuit to be designed. The next step is the actual writing of HDL code for modules, their interfaces and their internal details.
After the code has written we have to compile the code, this step is known as compilation. Here the HDL compiler analyzes the code for syntax errors and also checks it for compatibility with other modules on which it relies.

The most satisfying step is simulation or verification. The HDL simulator allows to define and apply the inputs to the design and to observe its outputs without ever having to build the physical circuit. There are at least two dimensions to verification.

In timing verification, the circuit operation including estimated delays, the setup, hold and other timing requirements for sequential devices like flip flops are met. In the functional verification the circuit’s logical operation independent of timing considerations; gate delays and other timing parameters are considered to be zero.

After verification step, the synthesis process is done in the back end stage. There are three basic steps, the first synthesis, converting the HDL description into a set of primitive or components that can be assembled in the target technology and it may generate a list of gates and a net list that specifies how they are interconnected.

In the fitter step, a fitter maps the synthesized components on to available device resources. It may mean selecting microcells or laying down individual gates in a pattern and finding ways to connect them within the physical constraints of the FPGA or ASIC die, is called as place and route process.

The final step is post fitting verification of the fitted circuit. It is only at the stage that the actual circuit delays due to wire lengths, electrical loading, and other factors can be calculated with reasonable precision.

VLSI DESIGN FLOW

INTRODUCTION:

Design is the most significant human endeavor: It is the channel through which creativity is realized. Design determines our every activity as well as the results of those activities; thus it includes planning, problem solving, and producing. Typically, the term "design" is applied to the planning and production of artifacts such as jewelry, houses, cars, and cities. Design is also found in problem-solving tasks such as mathematical proofs and games. Finally, design is found in pure planning activities such as making a law or throwing a party. More specific to the matter at hand is the design of manufacturable artifacts. This activity uses all facets of design because, in addition to the specification of a producible object, it requires the planning of that object’s manufacture, and much problem solving along the way. Design of objects usually begins with a rough sketch that is refined by adding precise dimensions. The final plan must not only specify exact sizes, but also include a scheme for ordering the steps of production. A semiconductor process technology is a method by which working circuits can be manufactured from designed specifications. There are many such technologies, each of which creates a different environment or style of design.
In integrated-circuit (IC) design, which is the primary focus of this book, the specification consists of polygons of conducting and semiconducting material that will be layered on top of each other to produce a working chip. When a chip is custom-designed for a specific use, it is called an application-specific integrated circuit (ASIC). Printed-circuit (PC) design also results in precise positions of conducting materials as they will appear on a circuit board; in addition, PC design aggregates the bulk of the electronic activity into standard IC packages, the position and interconnection of which are essential to the final circuit. Printed circuitry may be easier to debug than integrated circuitry is, but it is slower, less compact, more expensive, and unable to take advantage of specialized silicon layout structures that make VLSI systems so attractive [Mead and Conway]. Wire-wrap boards are like printed-circuit boards in that they use packages that must be precisely positioned. However, they allow the wire locations to fall anywhere as long as they connect to the posts of the IC packages. Such boards are typically manufactured as prototypes for less expensive PC boards.

The design of these electronic circuits can be achieved at many different refinement levels from the most detailed layout to the most abstract architectures. Given the complexity that is demanded at all levels, computers are increasingly used to aid this design at each step. It is no longer reasonable to use manual design techniques, in which each layer is hand etched or composed by laying tape on film. Thus the term computer-aided design or CAD is a most accurate description of this modern way.

**Overview of the development board (Xilinx Spartan-3E FPGA Starter Kit):**

The Spartan-3 generation of FPGAs includes the Extended Spartan-3A family (Spartan-3A, Spartan-3AN, and Spartan-3A DSP platforms), along with the earlier Spartan-3 and Spartan-3E families. These families of Field Programmable Gate Arrays (FPGAs) are specifically designed to meet the needs of high volume, cost-sensitive electronic applications, such as consumer products. The Spartan-3 generation includes 25 devices offering densities ranging from 50,000 to 5 million system gates.

The Spartan-3 platform was the industry’s first 90 nm FPGA, delivering more functionality and bandwidth per dollar than was previously possible, setting new standards in the programmable logic industry. The Spartan-3E platform builds on the success of the earlier Spartan-3 platform by adding new features that improve system performance and reduce the cost of configuration. The Extended Spartan-3A family builds on the success of the earlier Spartan-3E platform by further enhancing configuration and reducing power to provide the lowest total cost.

The Spartan-3AN platform provides the additional benefits of non-volatility and large amounts of on-board user flash. The Spartan-3A DSP platform extends the density range and adds resources often required in digital signal processing (DSP) applications.
Because of their exceptionally low cost, Spartan-3 generation FPGAs are ideally suited to a wide range of consumer electronics applications, including broadband access, home networking, display/projection, and digital television equipment. The Spartan-3 generation FPGAs provide a superior alternative to mask-programmed ASICs. FPGAs avoid the high initial cost, the lengthy development cycles, and the inherent inflexibility of conventional ASICs. Also, FPGA programmability permits design upgrades in the field with no hardware replacement necessary, an impossibility with ASIC.

**Spartan – 3E FPGA Features and Embedded Processing Functions:**

The Spartan-3E Starter Kit board highlights the unique features of the Spartan-3E FPGA family and provides a convenient development board for embedded processing applications. The board highlights these features:

- Spartan-3E FPGA specific features
- Parallel NOR Flash configuration
- Multi Boot FPGA configuration from Parallel NOR Flash PROM
- SPI serial Flash configuration
- Embedded development
- Micro Blaze™ 32-bit embedded RISC processor
- Pico Blaze™ 8-bit embedded controller
- DDR memory interfaces

**SIMULATION RESULT:**

- Fig 6.1: simulation result of carry select adder
- Fig 6.2 synthesis result of carry select adder

**ADVANTAGES AND APPLICATIONS**
Advantages:

- To reduce the area
- To increase the speed of the adder
- Decrease delay of the CSA
- To reduce the area-delay product

Applications:

- DSP processors
- Multimedia applications
- portable and mobile devices,
- multi standard wireless receivers
- biomedical instrumentation

SOFTWARE REQUIREMENTS

MODELSIM 6.5:

Software Versions:

This documentation was written to support ModelSim SE 6.5 for UNIX and Microsoft Windows 98/Me/NT/2000/XP. If the ModelSim software you are using is a later release, check the README file that accompanied the software. Any supplemental information will be there. Although this document covers both VHDL and Verilog simulation, you will find it a useful reference even if your design work is limited to a single HDL.

ModelSim’s graphic interface:

While your operating system interface provides the window-management frame, ModelSim controls all internal-window features including menus, buttons, and scroll bars. The resulting simulator interface remains consistent within these operating systems:

- SPARCstation with Open Windows, OSF/Motif, or CDE
- IBM RISC System/6000 with OSF/Motif
- Hewlett-Packard HP 9000 Series 700 with HP VUE, OSF/Motif, or CDE
- Linux (Red Hat v. 6, 7 or later) with KDE or GNOME
- Microsoft Windows 98/Me/NT/2000/XP

Because ModelSim’s graphic interface is based on Tcl/Tk, you also have the tools to build your own simulation environment. Easily accessible preference variables and configuration commands, simulator preference variables, and graphic interface commands give you control over the use and placement of windows, menus, menu options and buttons.

FPGA Design Flow Overview:

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 interconnects 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 inter connects.

Internal structure of an FPGA is depicted in the following figure.

![Internal Architecture of an FPGA](image)

**Fig.8.1 Internal Architecture of an FPGA**

**Design Analysis:**

**Step 1:**

Start the Project Navigator and Create the Project: Open ISE software; Go to Start Menu → Programs → Xilinx ISE 5 → Project Navigator. (or you can also look for the ISE icon on your desktop)

1. Create the Hamming code project; In the Project Navigator, select File → New Project and setup options as in figure 3.

2. Click OK.

Note: It is essential to choose the correct CPLD part number for the project to work. There are ways to change this part number after creating the project, but it is just easier to choose the correct one at a first time.

3. Acquire files for the project; Download the files sequence.vhd and sequence_tb.vhd provided at the lab website and save them in the directory C:/summer04/student_name/sequence. Make sure you are saving them without a "txt" extension but a "VHDL" extension.

4. Add VHDL code to the project; Go to Project → Add source, select the sequence.vhd (VHDL module) and sequence_tb.vhd (test bench associated to sequence.vhd).

![Creation of a project](image)

**Fig.8.8 Creation of a project. The format for Project Location must be “C:\summer04\student_name\”**

**Step 2:**

**Synthesize and Simulate the System:**

1. Synthesize the project; Highlight the file sequence.vhd in the Sources in Project window and double click on Synthesize in the Processes for current sources window. Make sure there are no warnings or errors.

2. Run behavioral simulation; Highlight the file sequence_tb.vhd in the Sources in Project window and double click on Simulate Behavioral VHDL Model in the Processes for current sources window.

3. Verify the operation of the sequence detector by looking at the input and output signals in the Wave window from the simulator. Figure; Simulation output.

Note: This current simulation says that if the required sequence is accompanied
with a low reset and rising clock edges on
each instance of the bit, the result will be
a high output on Z. It says nothing about
what will happen given any other set of
inputs.

Note: Z is a bus; to expand the bus into
its individual signals click on the “+”
symbol by the side of the signal’s name.
Also; sometimes it is useful to change the
radix of the signal. Right click on the bus
Z and change the radix of the signal to
decimal or hexadecimal.

Fig. 8.9 Simulation Output

Step 3:

**Explore a Different Input Case through Simulation:**

1. Open the test bench; Double click on
the file sequence_tb.vhd in the Sources in
Project window to open the file.

2. Modify input sequence; Modify lines
below the comment – Following lines
specify the input sequence of the
simulation – so that the input sequence is
now 001110010.

Step 4:

**Implementing the Design on the XCR board:**

1. Finding FPGA pins available; Look at
the board’s datasheet (it can be download
from the lab website) and find table 3.

2. Creating a User Constraints file;
Download the file sequence.ucf from the
lab website and complete it using the
information from table 3. Make sure you
download the file with extension “.UCF”
and NOT “.txt”. You can edit the file using
any text editor.

3. Adding User Constraints file to the
project; Go to Project → Add source,
select sequence.ucf file.

Step 5:

1. Create programming file; Highlight
sequence.vhd file in the Sources in Project
window and double click in Generate
Programming File in the Processes for
Current Source window.

2. Reading implementation results;
Expand process Implement Design on
Processes for Current Source. Expand Fit
process under Implement Design process
and double click on Fitter Report. Answer
question 4 based on the information of
this report.

Note: A graphical view of the chip and its
pin assignment can be obtained by:
highlight sequence.vhd in Project
window; expand Implement Design under Processes in Project
window; expand Fit under Implement Design process and double click on View
Fitted Design (Chip Viewer).

3. Launching impact;
Highlight sequence.vhd in Sources in
Project window; Expand Generate
Programming File tasks on the Processes
for Current Source window and double click on Configure Device (IMPACT). Choose Configure Devices → Next → Boundary-Scan Mode → Next → automatically connect to cable and identify Boundary-Scan chain → Finish → Ok. An Assign New Configuration File window must be the result of this operation.

**Xilinx Design Process:**

**Step 1: design**

Two design entry methods: HDL (Verilog or VHDL) or schematic drawings

**Step 2:** Synthesize to create Net list

Translate V, VHD, SCH files into an industry standard format EDIF file. Will also take into consideration synthesis constraints (Xilinx Constraint File (XCF))

**Step 3: Implement design (net list)**

Translate Map, Place & Route. Will take in consideration implementation constraints (User Constraints File (UCF)). Process generates a configuration file (.JED for CPLDs and .BIT for FPGAs.

**Step 4: Configure FPGA**

![Programming Succeeded](image)

Fig.8.10 IMPACT window and options for FPGA configuration

Note: IMPACT stands for Intelligent Multi-purpose Programming and Configuration Tool. Note that you can only have one instance of IMPACT opened. If you have more than one you will an error indicating problems in the communication with the chip.

1. Programming the device; Browse for the file sequence.jed in your project directory using the resulting window from previous step. Click Ok. Click on the Xilinx chip to highlight it and go to Operations → Program. Figure 6 shows the result of this operation and the options that must be selected.

Click Ok.

Note: By checking the option Read Protect on Program Options window (figure 8.8) we disable the CPLD design from being read back. This is particular useful for intellectual property protection. At the bottom of Program Option window there is a check for XPLA UES. UES stands for user electronic signature. A message (i.e. name and date) can be recorded on the chip for future references.

**Step 5: Project Testing**

![Software Screen](image)
1. Reset the board; Set the switch designed for the reset signal to high (push it away from you).

2. Test reset state; With the reset switch set up to high, change input X (by manipulating switch assigned to signal X) and generate several clock cycles (by pushing the button assigned to the clock signal).

3. Looking for a sequence; Set the reset signal to low. Give a value to the input signal X by manipulating the corresponding switch and generate a clock cycle (by pushing the corresponding button) to make the system read the input X.

4. Demonstrating your system to your T.A.; Introduce a wrong sequence to the system and show the corresponding state changes according to your state diagram. Introduce a right sequence and show the corresponding state changes according to your state diagram. Ask your T.A. to sign your final report.

5. Testing FPGA memory capacity; Switch the power off and on to the power strip

CONCLUSION

We have analyzed the logic operations involved in the conventional and BEC-based CSLAs to study the data dependence and to identify redundant logic operations. We have eliminated all the redundant logic operations of the conventional CSLA and proposed a new logic formulation for the CSLA. In the proposed scheme, the CS operation is scheduled before the calculation of final-sum, which is different from the conventional approach. Carry words corresponding to input-carry ‘0’ and ‘1’ generated by the CSLA based on the proposed scheme follow a specific bit pattern, which is used for logic optimization of the CS unit. Fixed input bits of the CG unit are also used for logic optimization. Based on this, an optimized design for CS and CG units are obtained. Using these optimized logic units, an efficient design is obtained for the CSLA. The proposed CSLA design involves significantly less area and delay than the recently proposed BEC based CSLA. Due to the small carry output delay, the proposed CSLA design is a good candidate for the SQRT adder. The ASIC synthesis result shows that the existing BEC-based SQRT-CSLA design involves 48% more ADP and consumes 50% more energy than the proposed SQRTCSLA, on average, for different bit-widths.

REFERENCES


