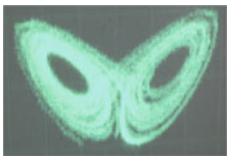
Chapter 2
Designing Hardware for FPGAs



FPGA realization of the Lorenz butterfly

Abstract In this chapter we will cover many of the basic concepts behind FPGA design. We start with an overview of our hardware platform, go through a quick introduction to the Quartus toolset and then review combinational along with sequential logic. We will conclude with the all important concept of timing closure. Although we cover a particular hardware platform, the material in this chapter can be adopted to understand other FPGA hardware platforms. This chapter, along with Chap. 1, lay the groundwork for the rest of the book. Nevertheless, please understand that majority of this chapter is meant primarily as a review. However, the conceptual material on abstracting the FPGA development flow via Simulink should not be skipped.

2.1 The FPGA Development Flow

In order to design for an FPGA, one needs to intimately understand the design process shown in Fig. 2.1 [1]. The first step in the process is design entry. In other words, you specify design functionality (differential equations) using tools such as Hardware Description Languages (HDLs), schematic entry or using a high level block diagram approach like Simulink. Next we compile the design to identify any syntax errors and then simulate the design to verify functionality. If design specifications are not met, we debug the design entry as necessary in order to meet functional specifications. Once the functional specifications have been met, we should run a

© Springer International Publishing Switzerland 2015

B. Muthuswamy and S. Banerjee, *A Route to Chaos Using FPGAs*, Emergence, Complexity and Computation 16, DOI 10.1007/978-3-319-18105-9_2

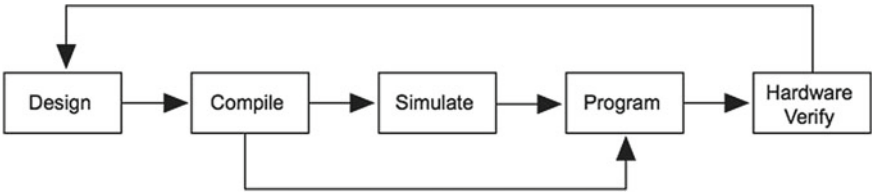


Fig. 2.1 A high-level view of FPGA design flow

timing-intensive simulation, but this topic is beyond the scope of this volume.[1] Once we have confirmed that the design is both functional and satisfies timing, we can download the bitstream onto the FPGA.

The first step in maximizing the capabilities of an FPGA is understanding the underlying architecture, the topic of Sect. 2.2.

2.2 The Architecture of an FPGA

Although the specifics of FPGA architecture vary between each device family (even within the same manufacturer), an FPGA is simply a massively parallel lookup table. Figure 2.2 shows a screen shot from the Quartus chip planner of the FPGA that we will be using in this book, the Cyclone IV.

Note how the device architecture is very repetitive in terms of fundamental structure, i.e.., the FPGA has a two dimensional row and column-based architecture to implement custom logic. Figures 2.3 and 2.4 show just how uniform this structure is.

Let us examine the LE in Fig. 2.4 in some detail [2], since a LE is the basic design unit on an FPGA. Each LE features:

- A four-input look-up table (LUT), which is a function generator that can implement any combinational logic function of four variables
- A programmable register
- A carry chain connection
- A register chain connection
- The ability to drive all types of interconnects: local, row, column, register chain and direct link interconnects
- Support for register packing
- Support for register feedback

An LE can also operate in normal mode or arithmetic mode. Normal mode is suitable for general logic applications and combinational functions. The arithmetic mode is ideal for implementing adders, counters, accumulators and comparators. LEs in

[1] We will however discuss the important concept of timing closure.

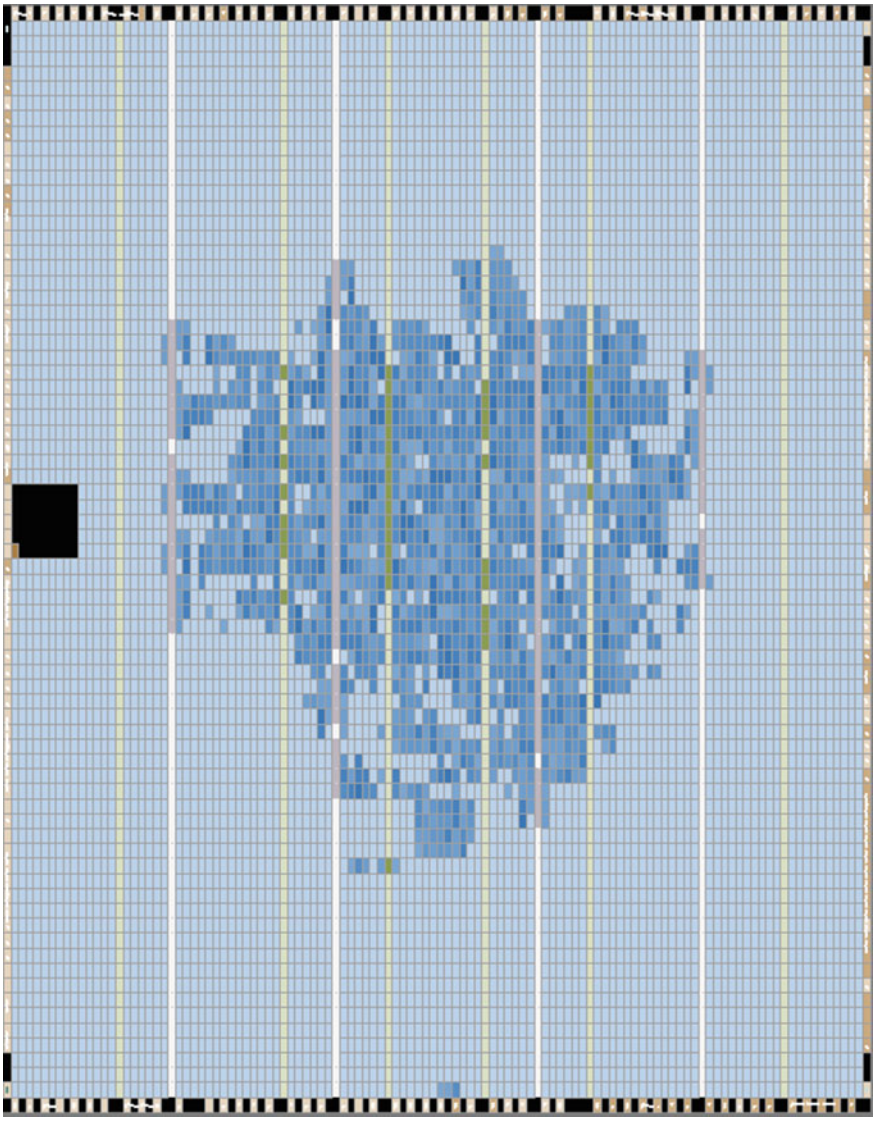


Fig. 2.2 A view of the Cyclone IV from the Chip planner in Quartus. The darker blue indicates filled FPGA regions

arithmetic mode can drive out registered and unregistered versions of the LUT output. Register feedback and register packing are supported when LEs are used in arithmetic mode.

In addition to LEs, Cyclone IV provide Phase Locked Loops (PLLs) for general-purpose clocking [2], as well as support for features such as clock multiplication

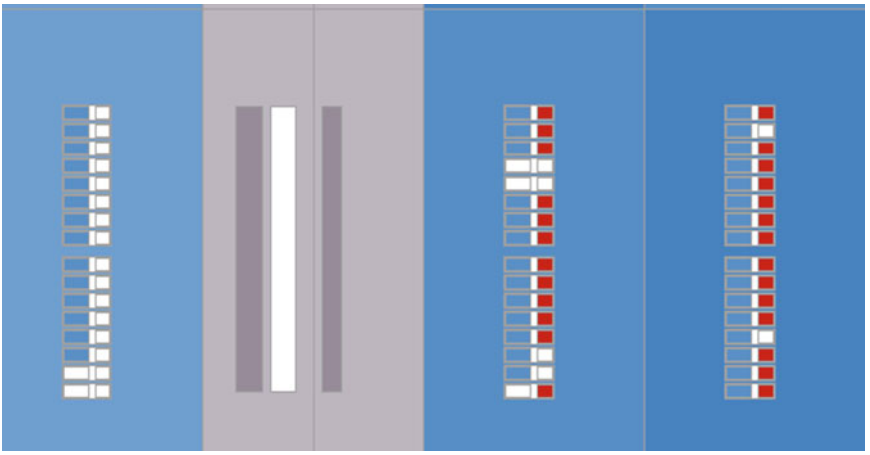


Fig. 2.3 A zoomed in view of the Logic Array Block (LAB) that highlight the 16 Logic Elements (LEs). LABs are the primary design features on the Cyclone FPGA that are grouped into rows and columns across the device [2]

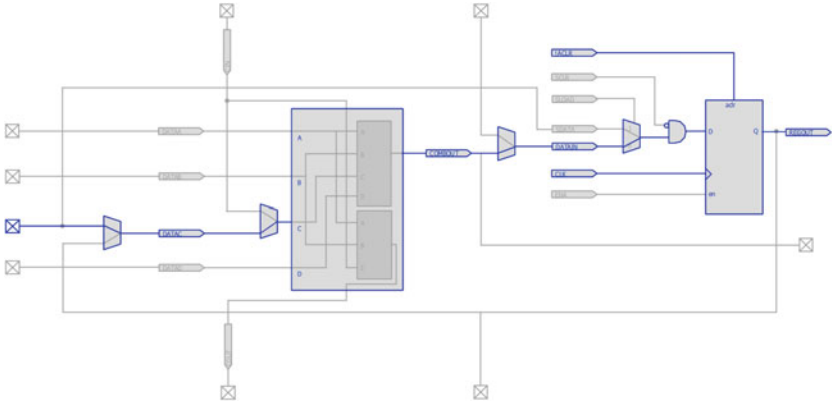


Fig. 2.4 LEs are the smallest units of logic in the Cyclone IV device architecture [2]

(division) and phase shifting. These PLLs are easily configurable via FPGA software tools, this feature will be discussed in the book as necessary. Cyclone IVs also incorporate embedded memory consisting of columns of M9K memory blocks [2]. Each M9K block can implement various types of memory, with our without parity. The M9K blocks are also easy to configure via FPGA software.

We have covered a brief overview of FPGA architecture. Going back to Fig. 2.1, a natural question is: how do we utilize software to realize a design onto an FPGA? The answer is: FPGA manufacturers provide advanced software tools for FPGA hardware design. The toolset that Altera provides is the Quartus suite. However since

we are implementing differential equations, we will primarily utilize MATLAB and Simulink. Nevertheless, we first need to choose the appropriate development board.

2.3 An Overview of the Hardware and Software Development Platform

In order to physically realize our differential equations, we will use the DE2-115 board[2] [3] shown in Fig. 2.5 from Terasic Technologies that utilizes a Cyclone IV (EP4CE115F29C7N) FPGA.

One also requires the Quartus toolset from Altera and the MATLAB (along with Simulink) package from Mathworks Corporation. Please contact the respective companies for the appropriate licenses. Note also that you need miscellaneous hardware such as interface cables and oscilloscopes.

FPGA hardware and software platforms evolve rapidly. A decision had to be made on the particular choice of hardware and software. We first chose the DE2-115 because it offered a large functionality-to-cost ratio for our research project(s) and, consequently, we chose the Quartus toolset. This platform was also available at the time when the volume was first published. However, the concepts covered in this volume should be applicable to any appropriate FPGA development platform and toolset(s).

We will now go over the salient features of our development platform, starting with the DE2-115 board.

2.3.1 An Overview of the Terasic DE2-115 Development Board

Since we have already discussed the FPGA in Sect. 2.2, we will discuss board peripherals.

2.3.1.1 FPGA Clocks

Probably the most important component on the FPGA board is the crystal oscillator for the clock circuitry [4]. The DE2-115 board includes one oscillator that produces 50 MHz clock signal. A clock buffer is used to distribute 50 MHz clock signal with low jitter to the FPGA. The board also includes two Subminiature Version A (SMA) connectors which can be used to connect an external clock source to the board or to drive a clock signal out through the SMA connector. In addition, all these clock

[2] These are not the only possible development platforms that can be used to realize chaotic dynamics. Please utilize the companion website to obtain information on other development platforms and software tools.

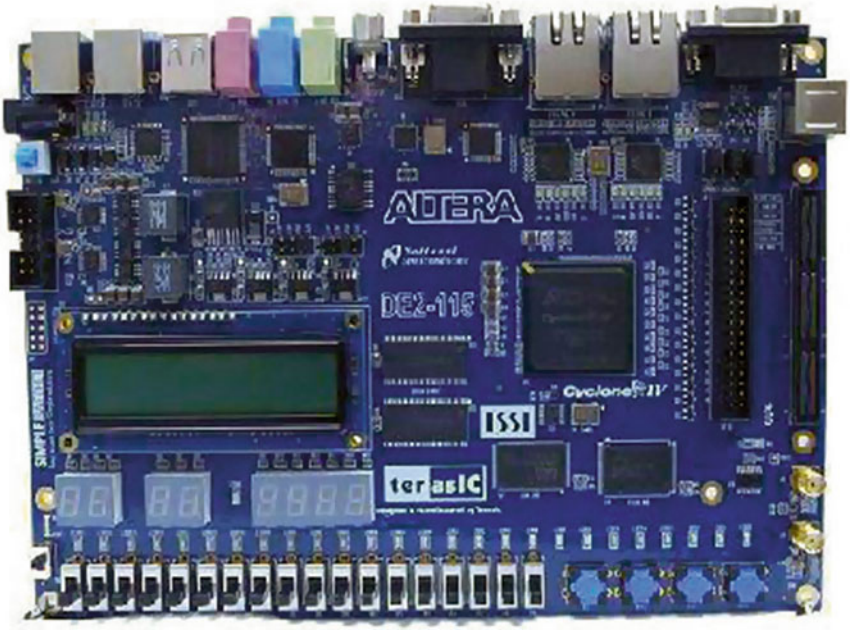


Fig. 2.5 The DE2-115 board [3]

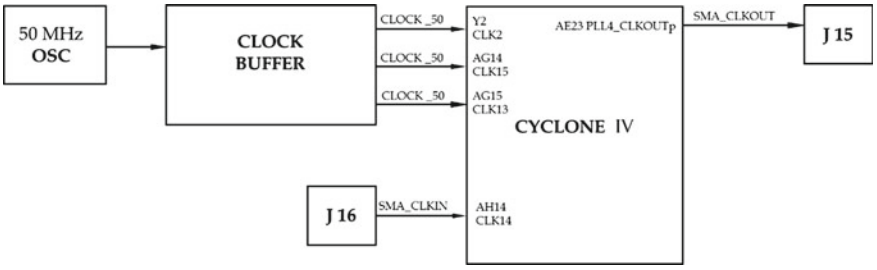


Fig. 2.6 Block Diagram of Clock Distribution on the DE2-115 [4]

inputs are connected to the PLL clock input pins of the FPGA to allow users to use these clocks as a source clock for the PLL circuit [4] (Fig. 2.6).

Since clocks are fundamental to FPGA functionality, pin assignments for clock inputs to FPGA input/output (I/O) pins are listed in Table 2.1.

2.3.1.2 Switches and Light Emitting Diodes (LEDs)

The DE2-115 board provides four push-button switches as shown in Fig. 2.7 [4]. Each of these switches is debounced using a Schmitt Trigger circuit, as indicated in

Table 2.1 Pin assignments for clock inputs

Signal name	FPGA pin no.	Description	I/O standard
CLOCK_50	PIN_Y2	50 MHz clock input	3.3 V
CLOCK2_50	PIN_AG14	50 MHz clock input	3.3 V
CLOCK3_50	PIN_AG15	50 MHz clock input	Depending on JP6
SMA_CLKOUT	PIN_AE23	External (SMA) clock output	Depending on JP6
SMA_CLKIN	PIN_AH14	External (SMA) clock input	3.3 V

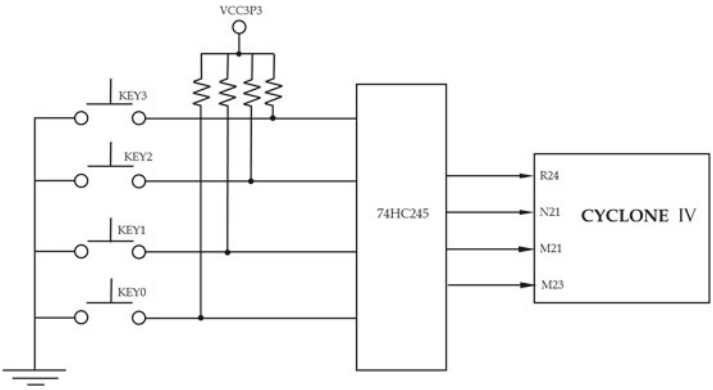


Fig. 2.7 Connections between the push-button and Cyclone IV FPGA [4]

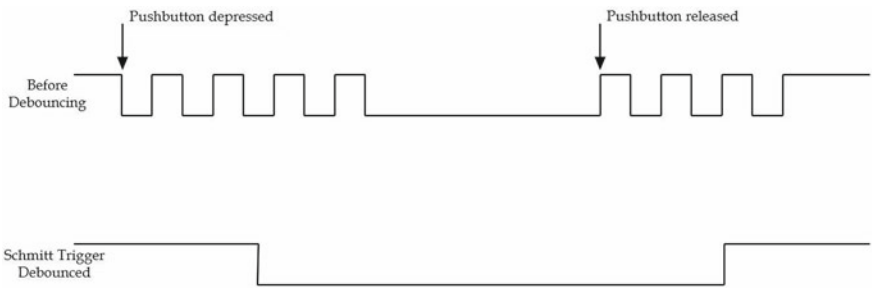


Fig. 2.8 Push button debouncing [4]

Fig. 2.8. The four outputs called KEY0, KEY1, KEY2, and KEY3 of the Schmitt Trigger devices are connected directly to the Cyclone IV E FPGA. Each push-button switch provides a high logic level when it is not pressed, and provides a low logic level when depressed. Since the push-button switches are debounced, they are appropriate for using as clock or reset inputs in a circuit.

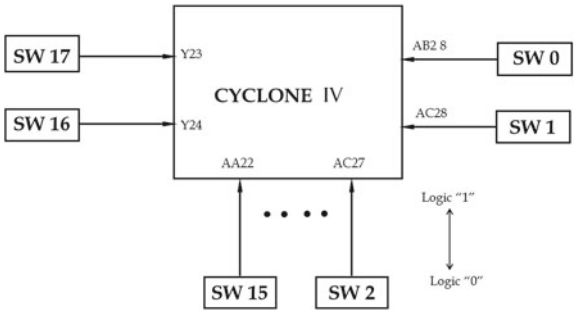


Fig. 2.9 Connections between the slide switches and Cyclone IV FPGA [4]

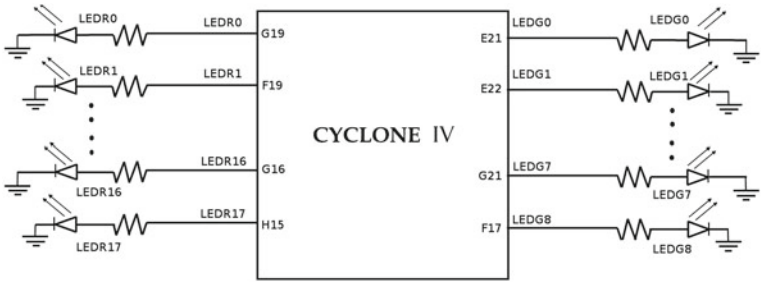


Fig. 2.10 Connections between the LEDs and Cyclone IV FPGA [4]

There are also 18 slide switches on the DE2-115 board (See Fig. 2.9) [4]. These switches are not debounced, and are assumed for use as level-sensitive data inputs to a circuit. Each switch is connected directly to a pin on the Cyclone IV FPGA. When the switch is in the down position (closest to the edge of the board), it provides a low logic level to the FPGA, and when the switch is in the up position it provides a high logic level.

There are 27 user-controllable LEDs on the DE2-115 board [4]. Eighteen red LEDs are situated above the 18 Slide switches, and eight green LEDs are found above the push-button switches (the 9th green LED is in the middle of the 7-segment displays). Each LED is driven directly by a pin on the Cyclone IV FPGA; driving its associated pin to a high logic level turns the LED on, and driving the pin low turns it off. Figure 2.10 shows the connections between LEDs and Cyclone IV FPGA.

2.3.1.3 7-Segment Displays

The DE2-115 board has eight 7-segment displays. These displays are arranged in two pairs and a group of four. As indicated in the schematic in Fig. 2.11, the seven segments (common anode) are connected to pins on the Cyclone IV. The 7-segment displays are actve low.

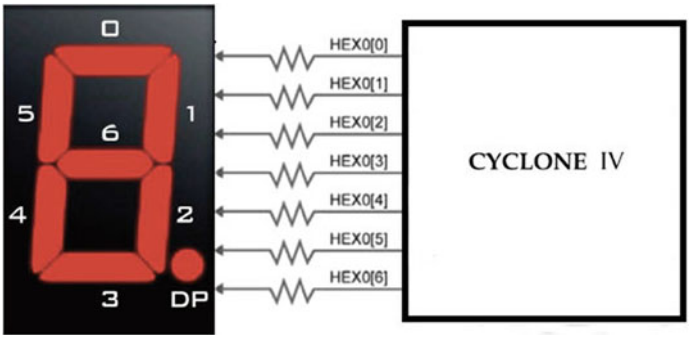


Fig. 2.11 Connections between the 7-segment display HEX0 and the Cyclone IV FPGA [4]

Table 2.2 JP7 settings for different I/O standards

JP7 jumper settings	Supplied voltage to VCCIO5 and VCCIO6 (V)	IO voltage of HSMC connector (JP8) (V)
Short pins 1 and 2	1.5	1.5
Short pins 3 and 4	1.8	1.8
Short pins 5 and 6	2.5	2.5 (Default)
Short pins 7 and 8	3.3	3.3

Table 2.3 JP6 settings for different I/O standards

JP6 jumper settings	Supplied voltage to VCCIO4 (V)	IO voltage of expansion header (JP5) (V)
Short pins 1 and 2	1.5	1.5
Short pins 3 and 4	1.8	1.8
Short pins 5 and 6	2.5	2.5
Short pins 7 and 8	3.3	3.3 (Default)

2.3.1.4 I/O Standards

The I/O voltage levels (standards) for the High Speed Mezzanine Card (HSMC) and the expansion header on the DE2-115 can be set by using JP7 and JP6 respectively [4]. Nevertheless, these jumpers also set the I/O standards for peripherals, for example the clock input in Table 2.1. Hence we list the JP7 and JP6 settings in Tables 2.2 and 2.3.

2.3.2 VHDL Primer and Using the Quartus Toolset

Although we will be primarily utilizing Simulink for a high level functional specification of our chaotic systems, it is important to have an idea of the underlying

hardware to which our design synthesizes to. There are many abstraction levels for understanding synthesized hardware. In this book, we will utilize the HDL approach. Specifically, we will use VHDL—one of the two HDLs that have an IEEE standard associated with them [5]. The other IEEE standard HDL is Verilog. We use VHDL in this book because it has better support for parameterized design [6].

Discussing every nuance of VHDL is beyond the scope of this book. Fortunately, many details are abstracted away because of our functional approach to specifying chaotic systems. However, we will still discuss some of the most important ideas behind VHDL in this section. For further study, two very good references on VHDL are Brown and Vranesic that deals with basic VHDL concepts [7] and Chu's book on VHDL for efficient synthesis [6].

Before we begin, an important note: as the name indicates, HDL describes hardware [6]. We are not writing a software program and hence it is essential to approach HDL from the hardware's perspective. The synthesis software should also be treated as a tool to perform transformation and local optimization. It cannot alter the original architecture or convert a poor design into a good one [6]. A rule of thumb: if we as humans cannot understand the underlying hardware functionality via the HDL, the synthesizer will not translate the design into a correct functional specification.

2.3.2.1 VHDL Modules: Entity, Ports and Architecture

Listing 2.1 below is a sample VHDL hardware specification.

Listing 2.1 Combinational logic in VHDL

```
1  -- Lines starting with -- are comments in VHDL.
2  library ieee; -- include ieee library for the use statements
       below
3  use ieee.std_logic_1164.all;
4  use ieee.numeric_std.all;
5
6  entity simpleLogicGates is port (
7      x1,x2,x3 : in std_logic;
8      x : in std_logic_vector(3 downto 0);
9      y : out std_logic_vector(4 downto 0));
10 end simpleLogicGates;
11
12 architecture combinational of simpleLogicGates is
13 begin
14     y(0) <= (x1 and x2) or x3;
15     y(2 downto 1) <= (x1&x(0)) AND (x2&x(1));
16     y(4 downto 3) <= x(3 downto 2);
17 end combinational;
```

The first three statements are IEEE standardized libraries to facilitate synthesis [6]. Line 1 invokes the IEEE library and line 2 allows us to use the predefined datatypes,

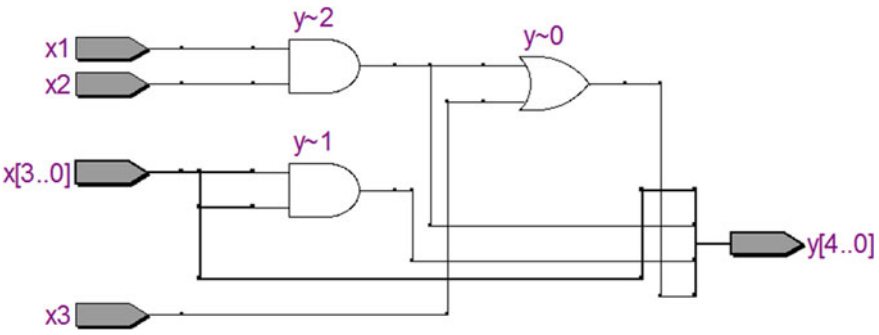


Fig. 2.12 Schematic diagram of our hardware specification, generated using the Quartus RTL viewer. The thicker lines imply multi-bits or a bus

std_logic and std_logic_vector. The numeric_std package enables us to utilize other datatypes such as signed and unsigned.

The entity declaration in line 6 describes the external interface of our circuit [6]. Our design communicates with the outside world via ports. These ports can be input, output or inout (bidirectional). In our example, we have three 1-bit wide input ports (x1, x2, x3) and two multi-bit wide (bus) ports. Note that the std_logic (and std_logic_vector) types support more than '1' and '0'. We will additionally only utilize the high impedance ('Z') support in std_logic, as this is the only other type defined for proper synthesis. A potential issue of utilizing 'Z' is support for tri-state devices in the underlying hardware but the Cyclone IV on the DE2-115 does support these devices. Also note that for interfacing to external physical components, we should not use VHDL types such as integers.[3]

The architecture body specifies the internal operation or organization of the digital logic system [6]. The architecture body consists of concurrent statements that describe combinational logic and process statements that describe sequential logic. We will first give examples of combinational logic design.

2.3.2.2 VHDL Combinational Logic Design and Using Quartus

Going back to our design specification in Sect. 2.3.2.1, we have three concurrent statements that synthesize to the RTL description in Fig. 2.12.

We will now summarize the main steps for using Quartus.

1. The first step in using Quartus is to obtain the software. Although the Quartus web-edition [8] will suffice for this section, we will need the Quartus Subscription Edition [9], ModelSim-Altera [10] and DSP Builder [11] for synthesis, simulation and mathematical functionality specification respectively. Please contact Altera

[3]Of course, we are free to choose any type for internal communication between modules. Such flexibility is the purpose of abstraction.

corporation for details on obtaining academic licenses or purchasing software. In this book, we will utilize the 12.0 versions of the toolset, although any version after 12.0 is acceptable. Note that some of the screens and menu actions may be different in newer versions of Quartus.
2. Next we need to download the system CD for our board, the DE2-115, from the board manufacturer (Terasic) website [3]. Although a system CD is part of the board kit, the latest version is available online. The CD has a plethora of useful documentation and reference designs.
3. Now that we have Quartus installed and the system CD, we can start our design by creating a new folder [4] for the combinational logic project.

Please refer to the online video and reference design on the companion website: http://www.harpgroup.org/muthuswamy/ARouteToChaosUsingFPGAs/ReferenceDesigns/volumeI-ExperimentalObservations/chapter2/combinationalLogicDesign/ for completing the simple combinational logic design.

Let us examine a frequently used VHDL construct for combinational logic design, the selected signal assignment, shown in listing 2.2. The selected signal assignment synthesizes to a multiplexer at the RTL level.

Listing 2.2 VHDL selected signal assignment

```
1  with selectBits select
2      output0 <= input0 when "00",
3                 input1 when "01",
4                 input2 when "10",
5                 input3 when others;
```

The careful reader should have noticed that the type of input (and consequently output) cannot be inferred from the VHDL snippet above. The type could be std_logic, std_logic_vector or integers. In fact, now would be a good time to look at the online video for a reference design that realizes an arithmetic logic unit using a mux at the output for selecting between different operations: http://www.harpgroup.org/muthuswamy/ARouteToChaosUsingFPGAs/ReferenceDesigns/volumeI-ExperimentalObservations/chapter2/alu/.

The video should help the reader complete the simple combinational logic design. An RTL view of the ALU is shown in Fig. 2.13.

2.3.2.3 VHDL Parameterization

In this section, we will utilize parameterization functionality of VHDL. We will let the synthesizer infer and connect multiple instances of the same module. Although we will be using a for loop for synthesis, please remember that we are designing hardware, not programming. The Quartus RTL view of the top level from our design is shown in Fig. 2.14.

[4] It is not a good idea to include spaces in the project path.

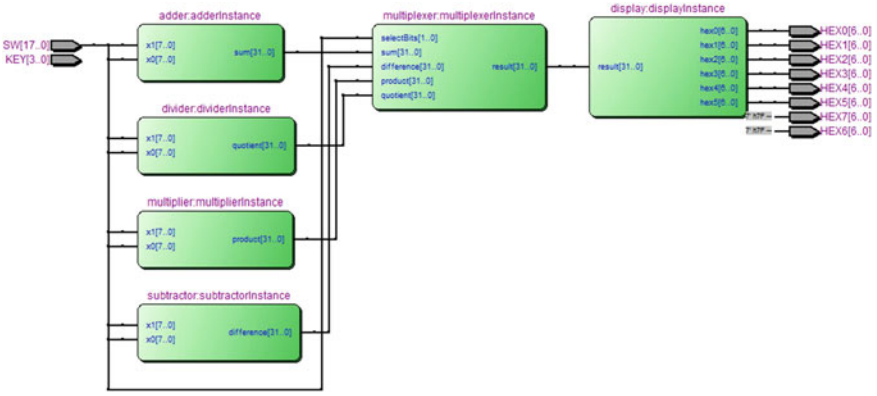


Fig. 2.13 RTL view for the ALU design

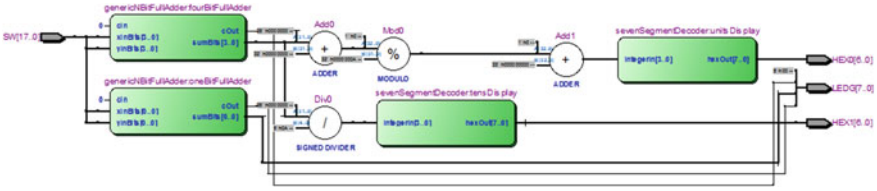


Fig. 2.14 RTL view for the ripple carry adder. Extraction of digits in the sum is done at the top level

The design primarily contains three components:

1. oneBitFullAdder: The oneBitFullAdder simply implements a structural one bit adder. The boolean equations for the sum input and the carry outputs for the ith one bit full adder are shown in Eqs. (2.1) and (2.2) respectively.

$$s_i = x_i \oplus y_i \oplus c_i \tag{2.1}$$

$$c_{i+1} = x_i y_i + c_i (x_i + y_i) \tag{2.2}$$

2. genericNBitFullAdder: In order to realize the genericNBitFullAdder, we will connect n one bit full adders in a ripple carry structure, as shown in listing C.1.
3. sevenSegmentDecoder: The seven segment decoder is a standard decimal to hex decoder module. The VHDL description is shown in listing C.2.

The top level realization of the generic ripply carry adder is shown in listing C.3. In order to completely understand the design, please look at the online video for the genericNBitFullAdder: http://www.harpgroup.org/muthuswamy/ARouteToChaosUsingFPGAs/ReferenceDesigns/volumeI-ExperimentalObservations/chapter2/rippleCarryAdder/.

2.3.2.4 VHDL Sequential Logic Design

So far we have seen designs whose output only dependent on the current input, not on the past history of inputs. Digital circuits where the input depends on both the present and past history of inputs (or the entire sequence of input values) are called sequential logic circuits [6]. We need sequential logic circuits to implement memory via registers (flip-flops). We store the system's state or state variables in memory and hence sequential logic circuits can also be specified using finite state machines (or state machines). Figure 2.15 shows a block diagram of a Moore (Mealy) state machine. We will examine a 24-h clock design in order to understand the concepts behind state machines.

The synthesized Quartus project and a 20-minute video on the design can be obtained from: http://www.harpgroup.org/muthuswamy/ARouteToChaosUsingFPGAs/ReferenceDesigns/volumeI-ExperimentalObservations/chapter2/twentyForHourClock/.

Considering Fig. 2.15, one can infer that we should be able to specify each of the blocks via VHDL. However before we discuss the VHDL realization, we need to understand the concept of a globally synchronous design [6].

A large digital system should consist of subsystems that operate at different rates or different clock frequencies. In our 24-h clock, the seconds counter should be updated at 1 Hz, the minutes counter at $\frac{1}{60}$ Hz and the hours counter at $\frac{1}{3600}$ Hz. There are primarily two approaches for clocking, shown in Figs. 2.16 and 2.17.

There are two main problems with the approach in Fig. 2.16. First, the system is no longer synchronous because if the subsystems interact, then the timing analysis becomes very involved. Second problem is the placement and routing of multiple clock signals. Since a clock signal needs a special driver and distribution network, having derived clock signals makes this process more difficult.

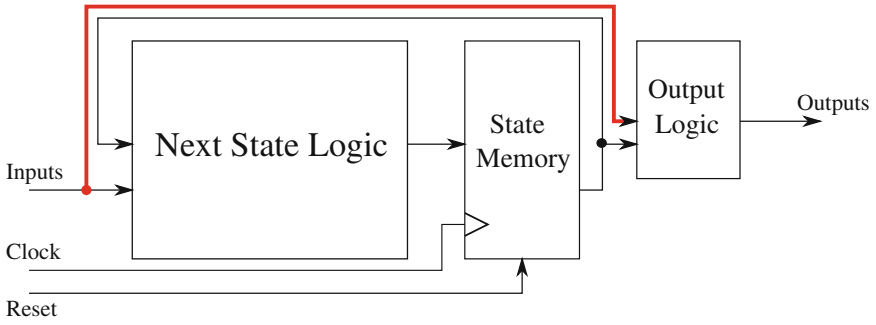


Fig. 2.15 Generic block diagram for a finite state machine. State machines are composed of next state logic (a combinational function of inputs and synchronous current state). The output function combinational logic function can be a function of current state only (Moore machine) or current state and input (Mealy). There is a single clock to ensure the design is fully synchronous. All finite state machines must have a well defined global reset

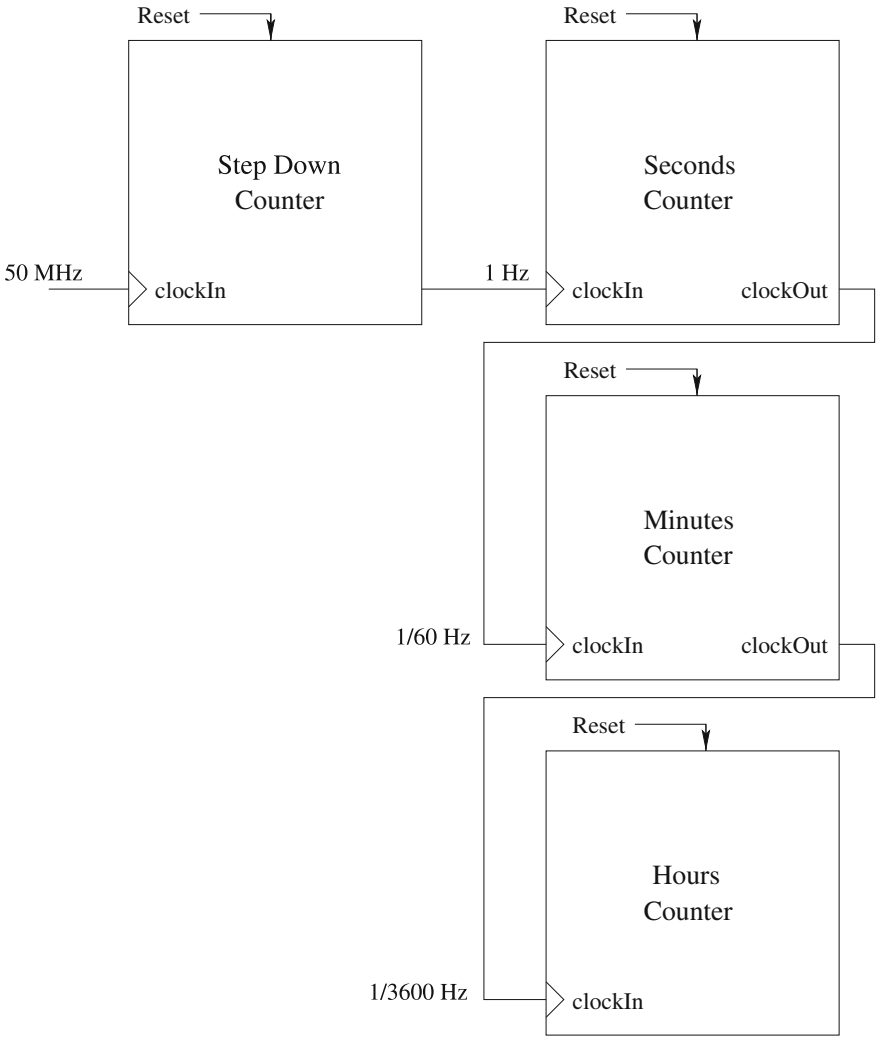


Fig. 2.16 System with multiple clock frequencies. The global clock is stepped down to a 1 Hz clock and each module subsequently outputs clocks with different frequencies

In contrast, the low rate single-clock enable pulse design shown in Fig. 2.17 is the preferred approach since the different subsystems are driven with the same clock frequency.

The specification of the seconds counter, along with a single pulse generator is in listing C.4 (obtained from the online Quartus project). The single pulse generator is a state machine that helps us implement the scheme shown in Fig. 2.17. The state machine generates a pulse that is exactly one clock cycle long, everytime the seconds counter overflows.

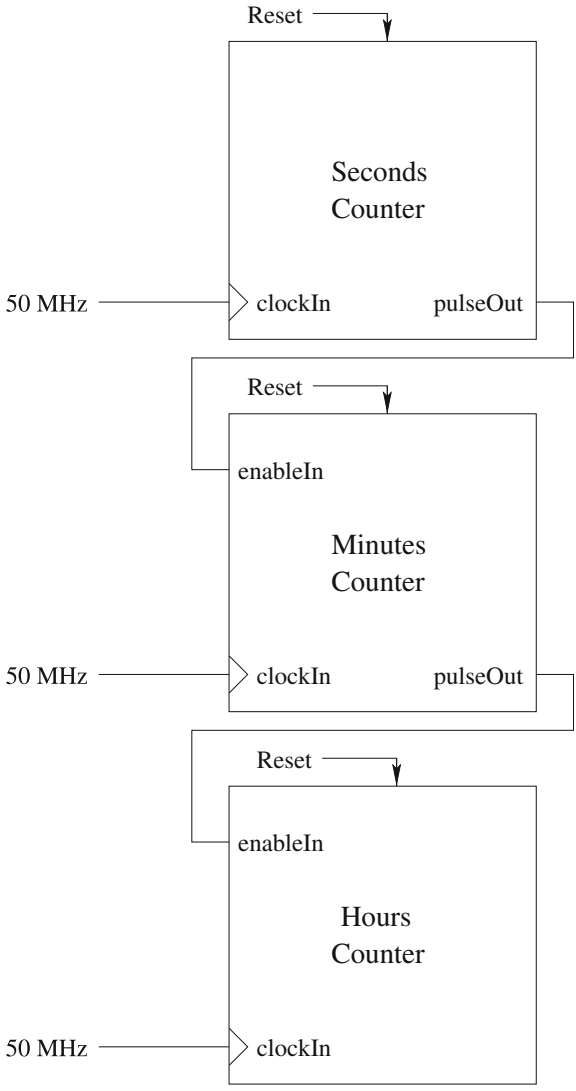


Fig. 2.17 System with a single synchronous clock. Each module uses a single pulse generator that has an enable pulse that is exactly 20 ns wide. This pulse acts as a synchronous trigger input for the subsequent module

One way to visualize an FSM is using State Machine Diagrams. The state transition diagram for the single pulse generator is shown in Fig. 2.18.

Once you download the online design to the DE2-115 board, you will notice that the base design clock is counting much faster than the usual 1 Hz frequency for a seconds counter. Exercise 2.1 asks you to modify the design so we have the 1 Hz frequency for the seconds counter.

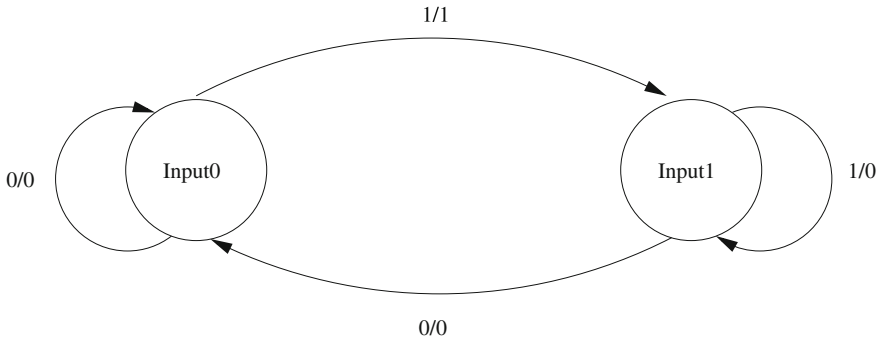


Fig. 2.18 A state transition diagram for the Mealy FSM realization of the single pulse generator. The circles represent states and the arcs represent transitions between states. Input(s)/Output(s) are specified on the arcs. It is assumed that transitions take place only on clock edges and hence synchronous behaviour is implied. Hence, the output is logic 1 only when we transition from the Input0 to Input1 state, in other words, the output is high only for one clock pulse. A Moore FSM is specified in a similar manner, except the outputs are given in the states. How would you specify the single pulse generator as a Moore FSM? *Hint* you need at least one more state

You should also understand that the design has a well defined reset state. Since the DE2-115 keys are debounced in hardware, we can utilize them as reset inputs without a debounce circuit. Exercise 2.4 considers debouncer design.

Going back to the issue of multiple clock frequencies in a design, a natural question is: can we always use a single clock frequency for every design? The answer is: no. This situation is very common when interfacing to external devices. For example, if our design were to interface with external memory (like SDRAM) then we will most likely have our FPGA design running at one frequency while the external SDRAM's clock is at a different frequency. Nevertheless we should clearly separate the different clock domains in our design. If we do, then we can utilize powerful timing closure tools provided by FPGA manufacturers to ensure that our design meets timing requirements. We will discuss timing closure in Sect. 2.4. In fact the next section shows an example design where we do interface to external hardware on the DE1 board.

2.3.3 Audio Codec Interfacing

We need ADC and DAC converters to interface external signals to/from the FPGA respectively. On the DE2 board, there is a Wolfson WM8731 [12] audio coder/decoder (codec) that has on-board ADC and DAC. This section shows how we can interface to these peripherals.

Figure 2.19 shows a top-level block diagram of our design. A discussion of each block follows [13].

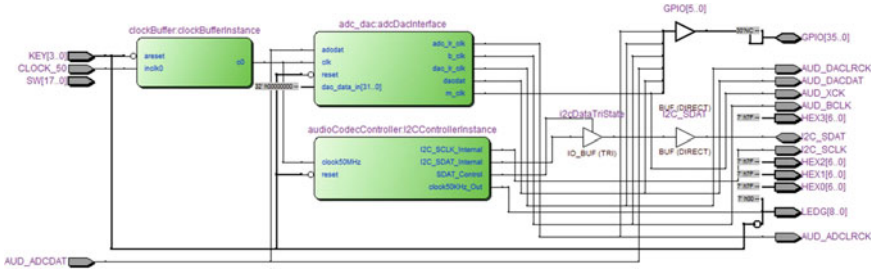


Fig. 2.19 A screenshot from the Quartus RTL view of our i^{2}c interface

The clock buffer uses two PLL modules: one to buffer the 50 MHz global clock and the other to provide a 18.42105 MHz clock (from the 27 MHz clock) for sending data to/from the codec.

The i^{2}c interface initializes the audio codec by sending a series of 10 data packets. In accordance with the i^{2}c protocol, each data packet is 24 bits long, consisting of: the codec address, the control register and the control register settings. The 10 packets were retrieved from ROM, and sent over the two-wire interface via an FSM.

The i^{2}c clock signal was a 50 KHz clock signal, generated from the 50 MHz PLL buffered FPGA clock using a counter. Functionally, the i^{2}c clock lagged the FSM clock by a half-clock cycle. This allowed the FSM to update each bit on the rising edge of the clock but the bit was still correctly interpreted by the codec.

Hence this design utilizes different clock frequencies between modules and this cannot be avoided due to the fact that we are interfacing to an external device. Also, the i^{2}c protocol limits the maximum frequency of the clock to be 3.4 MHz, depending on the mode [14]. Therefore, there is no possibility of clocking the i^{2}c initialization module at the global clock frequency of 50 MHz.

The ADC_DAC controller module's primary function is to place data into and read data from the A/D and D/A registers respectively. First, this controller generates two clock signals:

1. A bit clock with a frequency of 3.07 MHz to clock I/O bits from the codec.
2. A frame clock with a frequency of 192 KHz to signify either the left or right channel.

The synthesized Quartus project for our design can be found online: http://www.harpgroup.org/muthuswamy/ARouteToChaosUsingFPGAs/ReferenceDesigns/volumeI-ExperimentalObservations/chapter2/DE2i2cInterface/.

The State Machine Viewer in Quartus can be used to examine the state transition diagram of our FSM. We can access the state machine viewer using Analysis and Synthesis→Netlist Viewers→State Machine Viewer.

An important point that we need to remember is the voltage range of the ADA are ±2 V. Hence any digital design's I/O must confirm to this range of voltage. Also, the I/O range assumes there is no loading of the codec input and output.

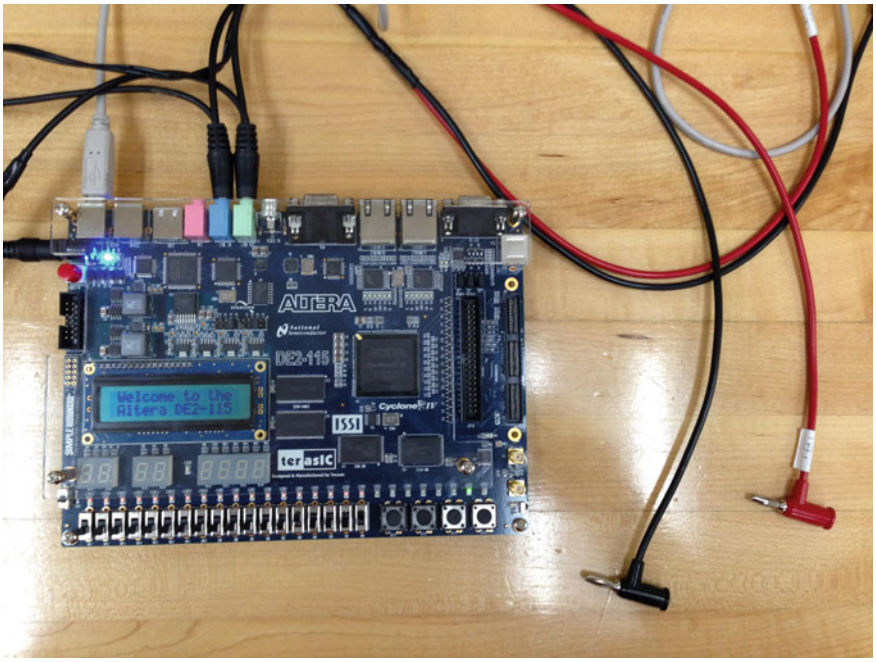


Fig. 2.20 The experimental setup used in this book. We are not using the *left-channel* in this experiment

Figure 2.20 shows the setup that we used throughout the book. We have a stereo-to-banana cable that is used for interfacing to scope probes. Figure 2.21 shows the results of the classic loop-back test: the input from the ADC is directly to DAC output (the online i^2c reference design utilizes loopback).

Nevertheless, only the phase delay is apparent in Fig. 2.21. Effects of sampling are not readily apparent with a sine wave. Figure 2.22 shows the result of the loop-back rest with a square wave.

Although we could implement the nonlinear functions and the numerical integration method in VHDL,[5] it is much easier to utilize Simulink for an abstract specification of the mathematics. Section C.5 highlights the conceptual steps in using DSP Builder, the Simulink library developed by Altera. You should go through that section before continuing on with the rest of the this chapter.

[5]That is, one could use the MegaWizard in Quartus and avoid DSP Builder. However, the DSP builder approach is more visual and this is the approach that we will use in this book. If you don't have access to DSP Builder, then you can utilize the approach using the MegaWizard. You can discuss questions related to this approach in the online forums available on the book's companion website. Note however that we will use the MegaWizard for implementing some of the functionality, such as bifurcations in Chap. 4.

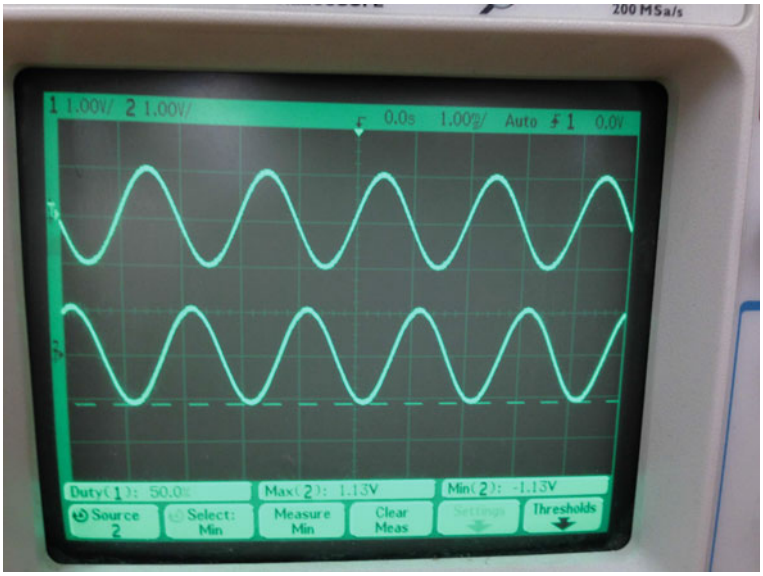


Fig. 2.21 Sine wave loopback, input frequency is 500 Hz. Notice the large phase delay at the output

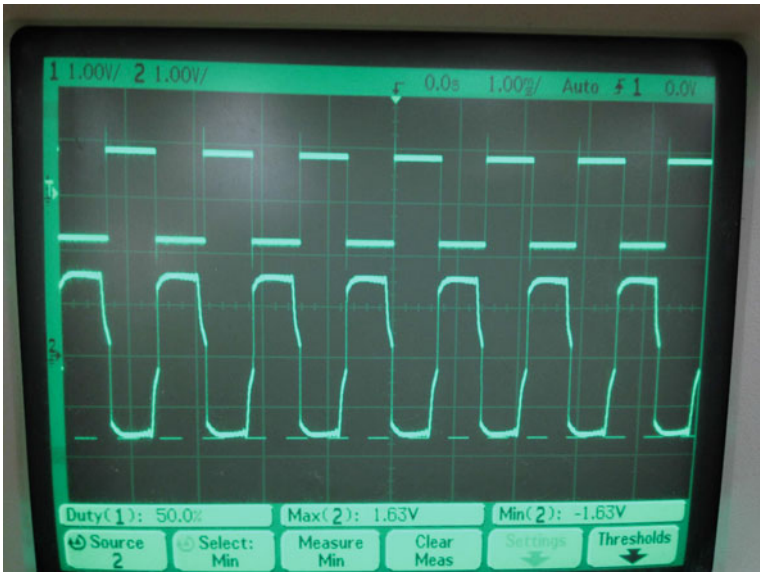


Fig. 2.22 Square wave loopback at 700 Hz. Compare with Fig. 2.21

2.4 Timing Closure

In this section we will discuss the concept of timing closure [5, 15] and look at an example of timing closure for the 24 h clock design from Sect. 2.3.2.4. Although reference designs on the companion website are closed with respect to timing, we will leave the advanced timing closure principles to volume II.

At the start of FPGA technology in the 1980 s, signal propagation delay in logic gates was the main contributor to circuit delay, while wire delay was negligible [15]. Hence cell placement and wire routing did not noticeably affect the final FPGA design. However, starting the in late 990 s, the advent of high density FPGAs and the consequent increase in the size of the final FPGA design implied that there was a need for automated timing closure tools.

Simply put, timing closure is the process by which we ensure the final placed and routed design on the FPGA satisfies timing requirements: setup and hold times for all flip-flops in our design are not violated. Timing closure tools such as TimeQuest (included in Quartus) from Altera adjust propagation delays in the final netlist such that the primary goal of setup and hold time constraints are satisfied. Recall from our basic logic courses that setup (long-path) constraints specify the amount of time a data input signal should be steady before the clock edge for each storage element. Hold time constraints specify the amount of time a data input signal should be stable after the clock edge.

Setup time constraints ensure that no signal transition occurs too late. Initial phases of timing closure focus on these types of constraints, as formulated[6] in Eq. (2.3).

$$t_{\text{clockPeriod}} > t_{\text{combinationalDelay}} + t_{\text{setupTime}} + t_{\text{skew}} \tag{2.3}$$

In Eq. (2.3):

1. $t_{\text{combinationalDelay}}$ is the worst-case combinational logic delay
2. $t_{\text{setupTime}}$ is the setup time of the receiving flip-flop
3. t_{skew} is the clock skew—the maximum time difference between flip-flop clock edges

Setup constraints are usually performed as part of static timing analysis, which defines timing slack as the difference between required arrival time and actual arrival time, as shown in Eq. (2.4). Positive slack means timing requirements have been met.

$$\text{Timing Slack} = \text{Required Arrival Time} - \text{Actual Arrival Time} \tag{2.4}$$

Hold time constraints ensure that signal transitions do not occur too early [15]. Hold violations can occur when a signal path is too short, allowinga receiving flip-flop to capture the signal at the current cycle instead of the next cycle. Thus the hold time constraint is formulated as in Eq. (2.5).

[6] Some authors define $\geq$ instead of $>$ in Eq. (2.3). We have considered the worst-case scenario and thus use $>$.

$$t_{\text{combinationalDelay}} > t_{\text{holdTime}} + t_{\text{skew}} \tag{2.5}$$

Note that clock skew usually affects hold time constraints than setup time constraint. Thus hold time constraints are typically enforced after placing and routing the clock network [15].

In order to experimentally understand these concepts, examine the SDC specification for timing constraints in Sect. C.5. TimeQuest uses the SDC file to close timing. For further instructions, please look at the video on the companion website: http://www.harpgroup.org/muthuswamy/ARouteToChaosUsingFPGAs/ReferenceDesigns/volumeI-ExperimentalObservations/chapter2/twentyFourHourClock/.

2.5 Conclusions

Below is a summary of the main concepts in this chapter:

1. The FPGA is an ideal platform for implementing discrete specifications of nonlinear differential equations because of the massively parallel architecture and variable (user-specified) data and address bus widths. Nevertheless, properly utilizing an FPGA requires the user to have a sound knowledge of basic digital logic principles.
2. In the case of sequential logic, one must aim for a globally synchronous design.
3. For implementing abstract mathematical concepts, we will use DSP Builder Advanced Blockset from Altera.
4. Timing closure is the process of satisfying timing constraints by informing the timing closure tool as to how the design should operate (with respect to timing parameters). The industry standard SDC file is used for specifying timing parameters to TimeQuest, the timing closure tool included with Quartus.

This chapter involved a lot of ideas and hopefully most of them were review of digital logic design concepts. In Chap. 3, we will combine digital logic design concepts and the ideas of DSP builder from this chapter to realize some classic chaotic systems on FPGAs via Simulink.

Problems

2.1 Modify the 24-h clock design from Sect. 2.3.2.4 to accurately count seconds, minutes and hours.

2.2 Instantiate the D flip-flops for synchronous reset from the 24-h clock design in VHDL as opposed to a component-based specification.

2.3 Design a finite state machine whose output is a 1 iff there are two consecutive 1 s in the input stream.

1. Design a Moore FSM for this problem.
2. Design a Mealy FSM for this problem.

2.4 In this problem, we will consider debouncer design [16]. The goal is to design a circuit that would compensate for the mechanical bounces in switch contacts. This circuit is necessary because consider our 50 MHz system clock with 20 ns period. Say a mechanical bounce lasts for 1 ms.

1. How many system clock cycles is one mechanical bounce?
2. Let us say we decide to have a timing-based solution: we declare an input change after signal has been stable for at least 5 ms. Design a system that incorporates a finite state machine and timer to accomplish this task.

Test your realization by using the mechanical switches on the DE2 board.

2.5 Design a finite state state machine that returns the remainder when an arbitrary size integer is divided by 5. One way to test your design on the DE2 board is: use two keys as 0 and 1 inputs. You can use an other key to send a "display remainder" signal to your design. Obviously, the fourth key is global reset.

2.6 The concept of recursion is central to computer programming. Consider listing 2.3 that recursively defines the factorial function:

Listing 2.3 Recursive specification of factorial function in MATLAB

```
1  function y = myfact(number)
2  %myfact Recursive realization of factorial function:
3  % n! = n*(n-1)*(n-2)...1
4      if number == 0
5          y=1;
6      else
7          y=number*myfact(number-1);
8      end
9  end
```

A natural question to ask would be: are there recursive structures in hardware? The answer is yes and a classic example is specifying an m-bit 2^n-to-1 mux (m-bits is the input/output bus width with n-select bits) using 2-1 muxes. Using Fig. 2.23 as an example, design and realize on the DE2 board a recursive multiplexer specification.

The elegant solution in Fig. 2.23 was proposed by Jake Nagel in the EE2900 (Combinational Logic Design) course at the Milwaukee School of Engineering in the Winter 2012–2013 quarter.

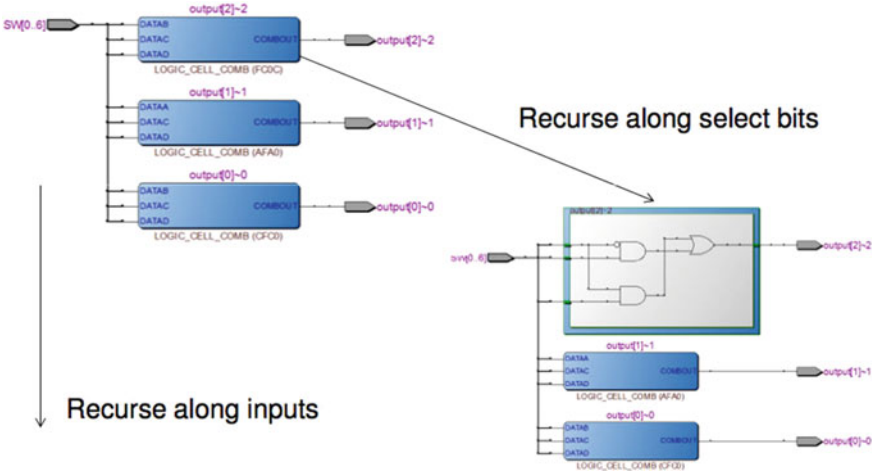


Fig. 2.23 Recursively specifying a 3-bit 2-to-1 mux, technology map viewer from Quartus

Lab 2: Introduction to Altera FPGA Tools

Objective: DE2 LCD interface.

Theory: We first need to thoroughly understand the LCD communication protocol. The DE2-115 user's manual [4] should be our starting point. The display controller is the HD44780 and a data sheet is available on the system CD that accompanies the DE2 board. Nevertheless, you can simply search online and get the latest version of the data sheet.

Lab Exercise:

After going through the LCD documentation, design an FSM to display the words "Hello" on the first line and "World" on the second line of the LCD display. This lab should be a very good review of digital logic concepts, so please take your time to complete the design before looking at the online solution video.

References

1. Altera Corporation (2008) My First FPGA Design Tutorial. In: Altera Corporation Online Tutorials, http://www.altera.com/literature/tt/tt_my_first_fpga.pdf. Accessed 22 Mar 2013
2. Altera Corporation (2013) Cyclone IV Handbook. In: Altera Corporation Online Datasheets, http://www.altera.com/literature/hb/cyclone-iv/cyclone4-handbook.pdf. Accessed 19 Apr 2013
3. Terasic (2013) Altera DE2-115 Development and Education Board. In: Terasic Corporation Online Cyclone Main Boards, http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=502. Accessed 21 Apr 2013

4. Terasic (2013) Altera DE2-115 Board User's Manual on the System CD. In: Terasic Corporation Online Cyclone Main Boards, http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=502. Accessed 21 Apr 2013
5. Simpson P (2010) FPGA design—best practices for team-based design. Springer, New York
6. Chu PP (2006) RTL hardware design using VHDL—coding for efficiency, portability and scalability. Wiley-Interscience, New Jersey
7. Brown S, Vranesic Z (2008) Fundamentals of digital logic design with VHDL, 3rd edn. McGraw-Hill, New York
8. Altera Corporation (2013) Quartus Web Edition, http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html. Accessed 7 May 2013
9. Altera Corporation (2013) Quartus Subscription Edition, http://www.altera.com/products/software/quartus-ii/subscription-edition/qts-se-index.html. Accessed 7 May 2013
10. Altera Corporation (2013) ModelSim-Altera Edition, http://www.altera.com/products/software/quartus-ii/modelsim/qts-modelsim-index.html. Accessed 7 May 2013
11. Altera Corporation (2013) DSP Builder, http://www.altera.com/products/software/products/dsp/dsp-builder.html. Accessed 7 May 2013
12. Wolfson WM8731 datasheet, http://www.wolfsonmicro.com/products/audio_hubs/WM8731/. Accessed 4 Oct 2013
13. Stapleton C (2011) Neuron Project EE2902 Spring 2011 Final Project Report
14. NXP Semiconductors (2012) UM10204 i^2C-bus specification and user manual. http://www.nxp.com/documents/user_manual/UM10204.pdf. Accessed 30 Sep 2013
15. Kahng A et al (2011) VLSI physical design: from graph partitioning to timing closure. Springer, New York
16. Brigham-Young University (2013) ECEn 224 Debouncing a Switch - A Design Example, Available via DIALOG. http://ece224web.groups.et.byu.net/lectures/20%20DEBOUNCE.pdf. Accessed 12 Oct 2013

Springer

http://www.springer.com/978-3-319-18104-2

A Route to Chaos Using FPGAs
Volume I: Experimental Observations
Muthuswamy, B.; Banerjee, S.
2015, XXIII, 219 p. 131 illus., Hardcover
ISBN: 978-3-319-18104-2