ELEX 7730: Digital System Design

Game Controller Using an FPGA

<table>
<thead>
<tr>
<th>Prepared by:</th>
<th>Prep Date:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navtej Heir</td>
<td>April 14, 2017</td>
</tr>
<tr>
<td>Andrew Ydenberg</td>
<td>April 14, 2017</td>
</tr>
</tbody>
</table>

N.Heir, A.Ydenberg 1 of 20 April 14, 2017
Contents

List of Figures 3

1 Executive Summary 4

2 Objectives 4

3 Introduction 4

4 Background 5

4.1 User Input .................................................................................................................. 5
4.2 FPGA Logic .................................................................................................................. 5
4.3 Serial Interface ............................................................................................................ 5

5 Design and Implementation 7

5.1 Game Control Hardware ............................................................................................ 7
5.2 Scanning Algorithm and Decoder .............................................................................. 8
5.3 Keypad Decode ........................................................................................................... 8
5.4 UART ........................................................................................................................... 8
5.5 PLL Clock and Pin Assignments .................................................................................. 9

6 Future Work 10

7 Conclusion 11

8 Appendix A: Code 12

8.1 Top Level Module ...................................................................................................... 12
8.2 Scanning Algorithm ................................................................................................. 13
8.3 Keypad Decode Module ............................................................................................ 14
8.4 UART Module ............................................................................................................. 15
8.5 Pin Assignments ......................................................................................................... 16
8.6 PLL Clock ................................................................................................................... 17

9 References 20
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Top Level Block Diagram</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Datasnip Program [1]</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>Keypad Matrix [2]</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>UART Data Framing [3]</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>Adafruit Resistive Touchscreen Overlay [4]</td>
<td>10</td>
</tr>
</tbody>
</table>
1 Executive Summary

Using the DE0-Nano development board by Altera [5], we created a game controller that is capable of interfacing with a computer. A simple 16-button keypad was used to demonstrate the use of the common W,A,S,D game keys. Our interface proved successful in use, and is able to be implemented with a variety of other hardware modules such as touch-screens, joysticks, or other serial interfaces.

2 Objectives

The objectives of this project were to:

- Design a UART interface from the development board to computer
- Design an interface from a keypad module to the development board
- Present the project playing the well-known snake game using a keypad

3 Introduction

This project was created using the DE0-Nano development board by Altera, with the EP4CE22F17C6 Cyclone IV FPGA. Using the development board we developed a game-controller interface. This interface is capable of using different hardware modules and sending there inputs to a computer. The computer is then able to interpret this command as a keyboard input. The interfaces relied on serial communication and a scanning algorithm.
4 Background

This project was developed in multiple stages. As seen in the top level block diagram, there is a different stage for each interface. There is an interface for the user input, FPGA logic, and the output serial connection.

![Top Level Block Diagram](image)

**Figure 1: Top Level Block Diagram**

4.1 User Input

The user input for our project was a 4x4 keypad. Each of the keys on the keypad correspond to a specific row and column. By sequentially scanning each row and column, we are able to determine when a key is pressed. Each key is assigned a value, and combinational logic within the FPGA processes this value.

4.2 FPGA Logic

After the value from the 4x4 keypad is detected, the FPGA uses combinational logic to determine the corresponding ASCII character. The combinational logic uses a lookup table (synthesized multiplexer), and loads the selected ASCII character into a temporary buffer. This temporary buffer is linked to a module of sequential logic, whose job it is to interface to the computer.

4.3 Serial Interface

The serial interface between the development board and the computer uses the well-known UART interface. For our purposes the following UART settings were used:

- Start bits: 1
- Data bits: 8
- Baud rate: 19200
- Parity bits: 0
- Stop bits: 1
- Handshaking: None
The temporary buffer is loaded into an output buffer, which is shifted out over a hardware connection to the computer’s USB port. The output buffer is shifted out at the specified UART settings. On the computers end, we used a simple program called Datasnip, which converts USB input to a simulated key press within the operating system.

![Datassniprogram](image)

**Figure 2: Datasnip Program**

This simulated key-press allows the ASCII input character to interface with the various applications on the computer. For the purposes of example within this program, we had the keyboard simulate the W,A,S,D keys to play the classic Snake game. The game was playable with slight latency, which was mainly due to the stiffness of the buttons on the keypad. Less stiff buttons would have allowed the user to actuate each key faster.
5 Design and Implementation

This section details the user input hardware, control hardware and the UART hardware.

5.1 Game Control Hardware

The input hardware consist of a 4x4 switch matrix keyboard. A switch matrix contains SPST switches at each intersection of row and column lines. A switch matrix decoder scans the rows and columns to determine which switch is closed. The FPGA is connected to the switch as per defined in pins.qsf file as signals named kpr[3] through kpr[0] (rows, top to bottom) and kpc[3] through kpc[0] (columns, left to right):

![Keypad Matrix](image)

Figure 3: Keypad Matrix [2]
5.2 Scanning Algorithm and Decoder

The scanning algorithm (Appendix A) used for the project was identical to the one created in [2]. The FPGA was configured with internal pull-up resistors on the kpr row input pins. The scan algorithm sets one column low at a time and checks the row levels. If a switch connected to that column is closed it will pull the corresponding row line low. For example if the pushbutton on the intersection of the third column and second row were pressed, then the second row output would read low only when the third column was set low:

![Keypad Matrix]

Figure 4: Keypad Matrix

5.3 Keypad Decode

When a row is found to be low, the scan algorithm suspends. At this time, the scanning algorithm outputs the row and column values to the decode module (Appendix A). Inside this module is a case statement which determines the key that is pressed and writes the corresponding ASCII code to a buffer which is then read by the UART module.

The ASCII codes have already been constructed to the correct data framing when they are written to the buffer. Beginning with the start bit "1", the 7 data bits with LSB first, one parity bit, and finishing with the stop bit "1". If any other keys are pressed the module will output all ones to the buffer. An output of all "1’s" represents a standby state.

![UART Data Framing]

Figure 5: UART Data Framing

5.4 UART

The UART module is used to send out each bit of the ASCII code for the four keys that were programed W,A,S,D. It does this by outputting each bit of the ASCII code on every clock edge of the "bclk". This clock is a 9600 Hz clock, which corresponds with the baud rate of the UART.

The UART portion of the MSP430F5529 was used as the hardware UART interface between the output of the UART module from the FPGA to the computer.
5.5 PLL Clock and Pin Assignments

The PLL clock wizard was used to generate both the system clock of 1 MHz and the 19200 Hz "bclk" used to output data from the UART module. A 1 MHz was chosen as it would be able to poll the keypad very rapidly compared to a lower clock. The output clock was set to 19200 Hz as the UART programmed to this spec.

The pins for the Keypad to the DE0 Nano development board were assigned as follows:

<table>
<thead>
<tr>
<th>keypad pin</th>
<th>signal</th>
<th>color</th>
<th>conn. pin</th>
<th>GPIO pin</th>
<th>FPGA pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>k0</td>
<td>kpr[3]</td>
<td>brown</td>
<td>14</td>
<td>09</td>
<td>D5</td>
</tr>
<tr>
<td>k1</td>
<td>kpr[2]</td>
<td>red</td>
<td>16</td>
<td>011</td>
<td>A6</td>
</tr>
<tr>
<td>k2</td>
<td>kpr[1]</td>
<td>orange</td>
<td>18</td>
<td>013</td>
<td>D6</td>
</tr>
<tr>
<td>k3</td>
<td>kpr[0]</td>
<td>yellow</td>
<td>20</td>
<td>015</td>
<td>C6</td>
</tr>
<tr>
<td>k4</td>
<td>kpc[0]</td>
<td>green</td>
<td>22</td>
<td>017</td>
<td>E6</td>
</tr>
<tr>
<td>k5</td>
<td>kpc[1]</td>
<td>blue</td>
<td>24</td>
<td>019</td>
<td>D8</td>
</tr>
<tr>
<td>k6</td>
<td>kpc[2]</td>
<td>violet</td>
<td>26</td>
<td>021</td>
<td>F8</td>
</tr>
<tr>
<td>k7</td>
<td>kpc[3]</td>
<td>gray</td>
<td>28</td>
<td>023</td>
<td>E9</td>
</tr>
</tbody>
</table>

Figure 6: Pin Assignments for Keypad [3]

The "kpc" pins were all linked to pull-up resistors, and PIN J14 was used as the "TX" (output) pin, and PIN J15 was set as the reset pin and connected to The pins.qsf file can be found in section 5 of the Appendix.
6 Future Work

Additional functionality that can be added to the project includes interfacing different kinds of hardware modules. These kinds of modules may include but are not limited to:

- Touch Screens
- Joysticks
- Motion Sensors
- Light Sensors
- Camera Sensors

![Adafruit Resistive Touchscreen Overlay](image)

Implementing these hardware modules does prove to be more complicated, hence we showed the interface of a keypad for the purposes of this project. Many of these modules need some sort of setup procedure which is done by sending the device serial commands. However after the device is setup, it follows the same procedure of sending the data over a serial port and then having the computer interpret that command as you wish.

In our project this interpretation happened to be an ASCII character, therefore we were able to use one of the many widely available programs called Datasnip [1]. However as the complexity of these modules increases, so does the way the data must be interpreted. For a device such as a camera or motion sensor, software may be more difficult to find, or you may need to write it yourself in the language of your choice.
7 Conclusion

In conclusion, we succeeded in meeting our objectives of creating hardware to interface with a keypad, and hardware to interface with a computer over UART. Thus we were able to use the keypad and control a Snake game on a computer.

In addition to the system working we learned a great deal about the nuances of System Verilog and how to think in terms of hardware rather than regular sequential programing.
8 Appendix A: Code

8.1 Top Level Module

// Top-Level Module ELEX 7660 Term Project
// Created by Ed.Casas 2017-1-11
// Modified by Navtej Heir and Andrew Ydendberg 2017-03-25
module Project ( output logic [3:0] kpc, // column select, active-low
    (* altera_attribute = "-name WEAK_PULL_UP_RESISTOR ON" *)
    input logic [3:0] kpr, // rows, active-low w/ pull-ups
    output logic [3:0] ct, // " digit enables
    output logic TX,
    input logic reset_n, CLOCK_50 ) ;

logic clk ; // 1 MHz clock for keypad scanning
logic bclk ; // 9600 Hz output clock
logic kphit ; // a key is pressed
logic [9:0] num ; // value of pressed key

assign ct = { {3{1'b0}}, kphit } ;
pll pll0 ( .inclk0( CLOCK_50), .c0(clk), .c1(bclk) ) ;

// instantiate of scanning algorithm, decode, and UART module

    colseq colseq_0(.kpr, .clk, .reset_n, .kpc);
kdecode kpdecode_0(.kpc, .kpr, .kphit, .num);
    UART UART_0(.DataIn(num), .clk(bclk), .TX(TX));
endmodule
8.2 Scanning Algorithm

```verilog
module colseq( input logic [3:0] kpr,
               input logic clk,
               input logic reset_n,
               output logic [3:0] kpc);

logic [3:0] column_hold; // Temp hold for column bits
logic [3:0] row_hold; // Temp hold for row bits
logic hold; // 1 - hold, 0 - scan (mode)

// Initialize kpc bit sequence
initial begin
    kpc = 4'b0111;
end

// Controller
always@(posedge clk) begin // Controller holds shift sequence if row is activated
    if(reset_n)begin
        if(kpr < 15) hold = 1;
        else hold = 0;
    end
end

// Datapath Registers
// Datapath scans keypad columns and outputs to kpc
always@(posedge clk) begin // In hold mode kpc and kpr are output, else scans
    if(hold)begin
        column_hold = kpc;
        row_hold = kpr;
    end
    else begin //kpc = column_hold;
        if(kpc == 4'b1110) kpc = 4'b0111; // Set to col 3 if col 0 selected
        else if (!reset_n) kpc = 4'b0111;
        else kpc = (kpc >> 1) + 8; // Scanning col by bit shifting
    end
endmodule
```
8.3 Keypad Decode Module

// kpdecode.sv - ELEX 7660 Term Project
// Navtej Heir and Andrew Ydendberg 2017-03-25

module kpdecode(input logic [3:0] kpc, // Keypad Column
                 input logic [3:0] kpr, // Keypad Row
                 output logic kphit, // Keypad Hit Bit
                 output logic [9:0] num); // Value to be output to display

always_comb begin

    case (kpr) // Determine Keypad Row Hit
        7: begin
            unique case (kpc) // Check Columns...
                7: num = '1;
                11: num = 10'b1011101110; // binary equivalent for ASCII w
                13: num = '1;
                14: num = '1;
            endcase
        endcase
        11: begin
            unique case (kpc)
                7: num = 10'b1011000010; // binary equivalent for ASCII a
                11: num = 10'b1011100110; // binary equivalent for ASCII s
                13: num = 10'b1011001000; // binary equivalent for ASCII d
                14: num = '1;
            endcase
        endcase
        13: begin
            unique case (kpc)
                7: num = '1;
                11: num = '1;
                13: num = '1;
                14: num = '1;
            endcase
        endcase
        14: begin
            unique case (kpc)
                7: num = '1;
                11: num = '1;
                13: num = '1;
                14: num = '1;
            endcase
        endcase
    default: num = '1; // default output "1", standby mode
endcase

kphit = (kpr < 15) ? 1 : 0; // output "1" if key is detected

endmodule

newpage
8.4 UART Module

```vhdl
module UART (input logic [9:0] DataIn,
             input logic clk,
             output logic TX);

  logic [9:0] Data_Next;
  logic [3:0] Count = 4'h00;

  always_ff@posedge clk begin
    if(Count != 10) begin
      TX <= Data_Next[Count];
      Count <= Count + 1;
    end
    else begin
      Count <= 0;
      Data_Next <= DataIn;
    end
  end
endmodule
```
8.5 Pin Assignments

```verbatim
set_location_assignment PIN_J14 -to TX    // Output Pin
set_location_assignment PIN_R8 -to CLOCK_50  // Clock Pin
# set_location_assignment PIN_J15 -to KEY[0]  // Reset Pin
set_location_assignment PIN_J15 -to reset_n

set_location_assignment PIN_A12 -to ct[0]    // Output Enable
set_location_assignment PIN_C11 -to ct[1]
set_location_assignment PIN_E11 -to ct[2]
set_location_assignment PIN_C9  -to ct[3]

set_location_assignment PIN_D5  -to kpr[3]   // Key Pad Pins
set_location_assignment PIN_A6  -to kpr[2]
set_location_assignment PIN_D6  -to kpr[1]
set_location_assignment PIN_C6  -to kpr[0]
set_location_assignment PIN_E6  -to kpc[0]
set_location_assignment PIN_D8  -to kpc[1]
set_location_assignment PIN_E9  -to kpc[3]
set_location_assignment PIN_F8  -to kpc[2]
```
### 8.6 PLL Clock

This code for the PLL Clock is auto-generated with Altera's Quartus PLL Clock [7] Wizard.

```verilog
// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module pll (areset, inclk0, c0, c1, locked);

input areset;
input inclk0;
output c0;
output c1;
output locked;

`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
wire [0:0] sub_wire2 = 1'h0;
wire [4:0] sub_wire3;
wire sub_wire6;
wire sub_wire0 = inclk0;
wire [1:0] sub_wire1 = {sub_wire2, sub_wire0};
wire [1:1] sub_wire5 = sub_wire3[1:1];
wire [0:0] sub_wire4 = sub_wire3[0:0];
wire c0 = sub_wire4;
wire c1 = sub_wire5;
wire locked = sub_wire6;

altpll altpll_component (.
  .areset (areset),
  .inclk (sub_wire1),
  .clk (sub_wire3),
  .locked (sub_wire6),
  .activeclock (),
  .clkbad (),
  .clkena ({6{1'b1}}),
  .clkloss (),
  .clkswitch (1'b0),
  .configupdate (1'b0),
  .enable0 (),
  .enable1 (),
  .extclk (),
  .extclkena ({4{1'b1}}),
  .fbin (1'b1),
);`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif
tri0 areset;

N.Heir, A.Ydenberg 17 of 20 April 14, 2017
.fbmimicbidir (),
.fbout (),
.fref (),
.icdrclk (),
.pfdena (1'b1),
.phasecounterselect ({4{1'b1}}),
.phasedone (),
.phasestep (1'b1),
.phaseupdown (1'b1),
.pllena (1'b1),
.scanclr (1'b0),
.scanclk (1'b0),
.scanclkena (1'b1),
.scandata (1'b0),
.scandataout (),
.scandone (),
.scanread (1'b0),
.scanwrite (1'b0),
.sclkout0 (),
.sclkout1 (),
.vcooverrange (),
.vcounterrange ());

defparam
altpll_component.bandwidth_type = "AUTO",
altpll_component.clk0_divide_by = 50,
altpll_component.clk0_duty_cycle = 50,
altpll_component.clk0_multiply_by = 1,
altpll_component.clk0_phase_shift = "0",
altpll_component.clk1_divide_by = 2604,
altpll_component.clk1_duty_cycle = 50,
altpll_component.clk1_multiply_by = 1,
altpll_component.clk1_phase_shift = "0",
altpll_component.compensate_clock = "CLK0",
altpll_component.inclk0_input_frequency = 20000,
altpll_component.intended_device_family = "Cyclone IV E",
altpll_component.lpm_hint = "CBX_MODULE_PREFIX=gggg",
altpll_component.lpm_type = "altpll",
altpll_component.operation_mode = "NORMAL",
altpll_component_pll_type = "AUTO",
altpll_component.port_activeclock = "PORT_UNUSED",
altpll_component.port_areset = "PORT_USED",
altpll_component.port_clkbad0 = "PORT_UNUSED",
altpll_component.port_clkbad1 = "PORT_UNUSED",
altpll_component.port_clkloss = "PORT_UNUSED",
altpll_component.port_clkswitch = "PORT_UNUSED",
altpll_component.port_configupdate = "PORT_UNUSED",
altpll_component.port_fbin = "PORT_UNUSED",
altpll_component.port_inclk0 = "PORT_UNUSED",
altpll_component.port_inclk1 = "PORT_UNUSED",
altpll_component.port_locked = "PORT_USED",
altpll_component.port_pfdena = "PORT_UNUSED",
altpll_component.port_phasecounterselect = "PORT_UNUSED",
altpll_component.port_phasedone = "PORT_UNUSED",
altpll_component.port_phasestep = "PORT_UNUSED",
altpll_component.port_phaseupdown = "PORT_UNUSED",
altpll_component.port_pllena = "PORT_UNUSED",
altpll_component.port_scanaclr = "PORT_UNUSED",
altpll_component.port_scanclock = "PORT_UNUSED",
altpll_component.port_scanclockena = "PORT_UNUSED",
altpll_component.port_scandata = "PORT_UNUSED",
altpll_component.port_scandataout = "PORT_UNUSED",
altpll_component.port_scandone = "PORT_UNUSED",
altpll_component.port_scanread = "PORT_UNUSED",
altpll_component.port_scanwrite = "PORT_UNUSED",
altpll_component.port_clk0 = "PORT_USED",
altpll_component.port_clk1 = "PORT_USED",
altpll_component.port_clk2 = "PORT_UNUSED",
altpll_component.port_clk3 = "PORT_UNUSED",
altpll_component.port_clk4 = "PORT_UNUSED",
altpll_component.port_clk5 = "PORT_UNUSED",
altpll_component.port_clkena0 = "PORT_UNUSED",
altpll_component.port_clkena1 = "PORT_UNUSED",
altpll_component.port_clkena2 = "PORT_UNUSED",
altpll_component.port_clkena3 = "PORT_UNUSED",
altpll_component.port_clkena4 = "PORT_UNUSED",
altpll_component.port_clkena5 = "PORT_UNUSED",
altpll_component.port_extclk0 = "PORT_UNUSED",
altpll_component.port_extclk1 = "PORT_UNUSED",
altpll_component.port_extclk2 = "PORT_UNUSED",
altpll_component.port_extclk3 = "PORT_UNUSED",
altpll_component.self_reset_on_loss_lock = "OFF",
altpll_component.width_clock = 5;

dendmodule
9 References