ECE 385 Final Report

Fall 2018
Section AB5
Rahul Surti (rsurti2) + Bala Chandrasekaran (bmchand2)

Introduction

For our final project, we built Pac-Man on the FPGA. This game leveraged the use of many components from previous labs such as the NIOS II processor, the VGA display, and the USB keyboard and EZ-OTG chip. We began with building this project using Lab 8 as a skeleton. The general structure of ball.sv along with ColorMapper is prevalent throughout our design in representing and displaying each of the game elements including the maze, Pac-Man, ghosts, and pellets. A vast majority of this project was built entirely in hardware, with the exception of the NIOS II processor supplying the USB keyboard direction input for Pac-Man, and the random direction input for the ghosts.

Operation

Hardware

As mentioned previously, a vast majority of this project was built entirely in hardware. Detailed descriptions of each module and its functionality is explained below in **Module Descriptions**, so this section will be dedicated to explaining the system as a whole.

This project does not use a FSM, but instead keeps track of various game components amongst their corresponding registers. For example, the lives and score are kept internally in the lives and score modules, and control signals for resetting the game are sent from these modules to various other components of the game. No FSM was needed as this game was relatively simple to implement since most actions are collision based.

In general, the logical flow of this project is as follows:

Keyboard input for Pac-Man and random input for the ghosts are fed from NIOS II into the FPGA. From there, the movement logic modules determine the next actual direction for Pac-Man and the ghosts to take, factoring in their locations, current direction, and queued direction. The queuing system for direction input is explained in detail later. Should any collisions happen (Pac-Man-pellet, Pac-Man-ghost), various modules respond accordingly, ie. the score is updated and pellets are collected, the game resets when Pac-Man dies.

The representative information about different game components(Pac-Man & ghost locations, pellet & maze information) are fed into the Color mapper module modified from lab 8, and the game is then shown onto the VGA display.

The VGA display is 480x640 pixels, and with 16x16 pixel tiles, that becomes a 30x40 indexed display. Both pellet and maze modules populate registers in on-chip memory with contents read from files storing the respective information. Pellet representation is 1-bit/index and maze representation is 5-bits/index into the VGA display. The total amount of memory needed to represent both the pellets and the maze is 1200 + 6000 = 7200 bits. The pellet file and the original 1-bit representation maze file (wall vs. space) were generated manually, however, the 5-bit maze representation containing adjacent wall information (detailed later), was generated via a python script.

Software

As specified above, the only role of the NIOS II processor actually used in this project was supplying USB keyboard input into the FPGA to control Pac-Man and random direction input to control the 4 ghosts. The setup for NIOS and the EZ-OTG chip was exactly the same as lab 8, thus we will not dive into detail for this matter.

The directions are generated randomly for the ghosts using the `rand()` function from the C standard library. To prevent a large amount of undesired random motion for the ghosts, ie. moving backwards, when the backwards direction is selected, the `rand()` function is rerun to generate a new direction. This slightly reduces the probability for ghosts to move backwards and thus keeps them moving forward with a slightly higher probability.

Module Descriptions

```
module pacman_top_level(
input CLOCK_50,
input [3:0]KEY,                          //bit 0 is set up as
Reset
output logic [6:0]HEX0, HEX1, HEX2, HEX3, HEX4, // VGA Interfa
ce
output logic [7:0]VGA_R,                 //VGA Red
                  VGA_G,                 //VGA Green
                  VGA_B,                 //VGA Blue
output logic      VGA_CLK,               //VGA Clock
                  VGA_SYNC_N,            //VGA Sync signal
                  VGA_BLANK_N,           //VGA Blank signal
                  VGA_VS,                //VGA virtical sync
signal
                  VGA_HS,                //VGA horizontal syn
c signal
                                         //CY7C67200 Interfac
e
inout  wire  [15:0]OTG_DATA,             //CY7C67200 Data bus
16 Bits
output logic [1:0]OTG_ADDR,              //CY7C67200 Address
2 Bits
output logic      OTG_CS_N,              //CY7C67200 Chip Sel
ect
                  OTG_RD_N,              //CY7C67200 Write
                  OTG_WR_N,              //CY7C67200 Read
                  OTG_RST_N,             //CY7C67200 Reset
input             OTG_INT,               //CY7C67200 Interrup
t
                                         // SDRAM Interface f
or Nios II Software
```

```
output logic [12:0]DRAM_ADDR,             //SDRAM Address 13 B
its
inout  wire  [31:0]DRAM_DQ,               //SDRAM Data 32 Bits
output logic [1:0]DRAM_BA,                //SDRAM Bank Address
2 Bits
output logic [3:0]DRAM_DQM,               //SDRAM Data Mast 4
Bits
output logic DRAM_RAS_N,                  //SDRAM Row Address
Strobe
             DRAM_CAS_N,                  //SDRAM Column Addre
ss Strobe
             DRAM_CKE,                    //SDRAM Clock Enable
             DRAM_WE_N,                   //SDRAM Write Enable
             DRAM_CS_N,                   //SDRAM Chip Select
             DRAM_CLK                     //SDRAM Clock
);
```

Description: This module is the top level module for the entire project. It connects all the FPGA hardware pins into the project, as well as interface the NIOS II SOC with the EZ OTG chip.

```
module hpi_io_intf(
input        Clk, Reset,
input        [1:0]from_sw_address,
output       [15:0]from_sw_data_in,
input        [15:0]from_sw_data_out,
input        from_sw_r, from_sw_w, from_sw_cs, from_sw_reset,
// Active low
inout        [15:0]OTG_DATA,
output       [1:0]OTG_ADDR,
output       OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N // Active
low
);
```

Description: This module's functionality did not change from lab8. It ensures the hpi interface does not glitch at all during the communication between the fpga and the hpi interface. It also tristates the NIOS II to ensure that NIOS is only either reading from or writing to the bus.

```
module lab8_soc (
input  wire        clk_clk,                //             clk.
clk
output wire [7:0]  keycode_export,         //         keycode.
export
output wire [15:0] ghost_direction_export_export,// ghost_dire
ction_export.export
input  wire [15:0] ghost_status_export_export,  //    ghost_st
atus_export.export
output wire [1:0]  otg_hpi_address_export, // otg_hpi_address.
export
output wire        otg_hpi_cs_export,      //      otg_hpi_cs.
export
input  wire [15:0] otg_hpi_data_in_port,   //    otg_hpi_data.
in_port
output wire [15:0] otg_hpi_data_out_port,  //                .
out_port
output wire        otg_hpi_r_export,       //       otg_hpi_r.
export
output wire        otg_hpi_reset_export,   //   otg_hpi_reset.
export
output wire        otg_hpi_w_export,       //       otg_hpi_w.
export
input  wire        reset_reset_n,          //           reset.
reset_n
output wire        sdram_clk_clk,          //       sdram_clk.
clk
output wire [12:0] sdram_wire_addr,        //      sdram_wire.
addr
```

```
output wire [1:0]  sdram_wire_ba,          //                .
ba
output wire        sdram_wire_cas_n,       //                .
cas_n
output wire        sdram_wire_cke,         //                .
cke
output wire        sdram_wire_cs_n,        //                .
cs_n
inout  wire [15:0] sdram_wire_dq,          //                .
dq
output wire [1:0]  sdram_wire_dqm,         //                .
dqm
output wire        sdram_wire_ras_n,       //                .
ras_n
output wire        sdram_wire_we_n         //                .
we_n
);
```

Description: This module is autogenerated by qsys in platform designer and contains the NIOS II processor. It allows us to interface the NIOS II processor with other necessary hardware components like the ez otg chip to use the USB keyboard. Since lab 8, we've added ghost_direction_export_export and ghost_status_export_export. The ghost direction is a 16 bit value that controls the directions of all 4 ghosts from the NIOS II processor. The ghost status is the current direction in which all the ghosts are moving from the FPGA. Using these two variables can allow sophisticated ghost movement. Had we had more time, we could have implemented AI movement for the ghosts, instead of just random movement using `rand()` from the C standard library.

```
module  VGA_controller (
    input              Clk,         // 50 MHz clock
                       Reset,       // Active-high reset signa
l
    output logic       VGA_HS,      // Horizontal sync pulse.
 Active low
```

```
                        VGA_VS,      // Vertical sync pulse.  A
ctive low
    input               VGA_CLK,     // 25 MHz VGA clock input
    output logic        VGA_BLANK_N, // Blanking interval indic
ator.  Active low.
                        VGA_SYNC_N,  // Composite Sync signal.
 Active low.  We don't use it in this lab,
                                     // but the video DAC on th
e DE2 board requires an input for it.
    output logic [9:0] DrawX,        // horizontal coordinate
                        DrawY        // vertical coordinate
);
```

Description: This module supplies the control signals for the VGA display as well as the current vga drawing x and y coordinates.

```
module  color_mapper (
    input                is_wall, is_pellet,
    input [3:0]          adjacent_walls_vga,
    input [9:0]          DrawX, DrawY,       // Current pixel c
oordinates
                         pacman_x, pacman_y,
    input [3:0] [9:0]    ghost_x, ghost_y,
    output logic [7:0]   VGA_R, VGA_G, VGA_B // VGA RGB output
);
```

Description: This module determines what the color output of any given vga x,y coordinate should be given the Pac-Man and ghosts' coordinates, and the maze and pellet values at that coordinate. It also uses the adjacent walls of the current vga index into the maze to further color/design the walls.

```
module HexDriver (
input  [3:0]  In0,
output logic [6:0]  Out0
```

```
);
```

Description: This is the hex driver used to translate a 4 bit input into the appropriate 7 segment display for that number. This is used for displaying the score and the current number of lives.

```
module lives_reg (
    input  logic Clk, Reset, Restart,
    output logic [1:0] out,
    output Reset_game
);
```

Description: This module maintains the current number of lives remaining. On a Restart signal, the internal lives register is decremented. The default number of lives is 3. On the loss of all 3 lives, the Reset_game signal is triggered, resetting the game to its initial state. Specifically, it triggers the reset of the pellets, filling the maze up again and resets the score. A normal Restart would not trigger the reset of pellets or score, solely the reset Pac-Man and the ghosts to their original positions. It also provides an output of current lives for the hex display.

```
module score_reg (
    input  logic Clk, Reset, Reset_game, increment,
    output logic [15:0] out
);
```

Description: This module maintains the current score and outputs to the hex display. On a Reset (hardware) or Reset_game signal, the score is reset to 0. On an increment signal, it adds to the score. To circumvent the score displaying in hexadecimal, we take the following measure. Whenever the least significant digit of the hex display is `0xA`, we add `0x6` to the score. Likewise we add `0x60` whenever the second least significant digit is `0xA`, and so on. This effectively displays the score in base 10 on the hex display, despite being quite a “hacky” solution. An alternative route would be to represent the score in Binary-Coded Decimal, however that would be quite an expensive operation.

```
module maze(
```

```
    input Clk,
    input [10:0] index, pacman_index,
    input [3:0][10:0] ghost_index,
    output is_wall,
    output [3:0] adjacent_walls, adjacent_walls_vga,
    output [3:0][3:0] adjacent_walls_ghost
);
```

Description: This module represents the maze for Pac-Man to move around in. Initially, it reads the maze in from file and loads the contents into on-chip memory. The maze bitmap encoding scheme is as follows. The least significant bit indicates a wall (1), or a space (0). The next bits, in increasing significant bit, indicate whether there is a wall or a space in the corresponding direction, up, right, down, left, respectively. Thus each index of the 30x40 vga display is represented by 5 bits of data, for a total of 6,000 bits to represent the entire maze. This module takes in the vga, Pac-Man, and ghosts indeces — total of 6 indeces — from the index_calculator modules, and outputs whether the current vga index is a wall or not, and the 4 adjacent wall values for the vga, Pac-Man, and 4 ghosts.

```
module pellets(
    input Clk, Reset, Reset_game,
    input [10:0] index, pacman_index,
    output logic is_pellet, seen_pellet
);
```

Description: This module functions similar to the maze module in that it reads the original pellet bitmap from file into on-chip memory. The pellets bitmap only needs 1 bit/maze index to fully maintain all pellets. This module also takes in the Pac-Man index in order to “collect” the pellet as Pac-Man moves around the maze. On a collision of Pac-Man and a pellet, a seen_pellet signal is sent to the score register while also being maintained in the pellets module. While the score is then decremented inside the score register, the internal seen_pellet signal indicates that the pellet must now be erased from the internal pellets register. This effectively simulates the pellets being collected and score being updated in a 2-cycle fashion. The Reset_game signal fills the internal pellets register with the contents from the original pellets file, simulating a restart of the game when Pac-Man loses all his lives.

```
module Pac-Man(
    input           Clk, Reset, frame_clk, Restart,
    input [2:0]     direction, id,
    output[9:0]     pacman_x, pacman_y
);
```

Description: This module is the representation of Pac-Man in the game, just a (x,y) coordinate. This module is also reused for the ghosts, with the only separating factor being a 3-bit id. Pac-Man is `100`, while the 4 ghosts are `000` to `011`. The differing id's are used to maintain different respawn locations on a reset. The direction input is fed in from the game_logic module for Pac-Man, and the ghost_movement_logic for ghosts, as they do differ slightly in representation. This module also supplies the (x,y) location output to multiple modules throughout the project.

```
module game_logic (
    input           Clk, Reset, Restart,
    input [9:0]     pacman_x, pacman_y,
    input [3:0]     adjacent_walls,
    input [7:0]     keyboard_direction,
    output [2:0]    direction
);
```

Description: This module is quite complex. While the Pac-Man module takes in a direction to move and always moves exactly in whatever direction is specified, this module is what determines which direction to move. The direction is a 3 bit value, broken down as follows. The most significant bit indicates whether Pac-Man should move at all (1), or stop (0). If stopped, the last 2 bits are ignored. If moving, the remaining two bits indicate the direction: up, right, down, left, mapped from `000` to `011` respectively, just like the adjacent_walls representation. This module also takes in the keyboard direction to determine which direction the player wants to move. However, this input isn't always what the direction output is set to. There is a queuing system in place that factors in the adjacent_walls to Pac-Man. If there is a wall in the direction the player has fed in through the keyboard input, then the current direction is maintained, however the input is stored in the module.

Whenever Pac-Man is “exactly” on an index in the maze, ie. (x,y) both are multiple of 16, an attempt to move in the queued direction is made. If there is still a wall, the current direction is maintained, and the input remains queued. If there is a space, the new direction is taken, and the input is dequeued. This module was difficult to build and effectively simulates the motion that the Pac-Man takes in the real Pac-Man game.

```
module ghost_movement_logic (
    input           Clk, Reset,
    input [9:0]     pacman_x, pacman_y,
    input [3:0]     adjacent_walls,
    input [2:0]     direction_input,
    output [2:0]    direction
);
```

Description: This module is exactly the same as the game_logic module for Pac-Man except for instead of having a keycode input, it takes in a direction input from the NIOS II processor, which has been decoded by the ghost_direction_reader module.

```
module ghost_direction_reader(
    input [15:0] ghost_direction_nios,
    output [3:0][2:0] ghost_direction_fpga
);
```

Description: This module reads the 16 bit ghost direction input from NIOS, where each section of 4 bits is a direction input in the same representation as the direction format specified above, and outputs an packed array of direction inputs for the 4 ghosts. In retrospect, only 2 bits are needed to maintain the directions of all 4 ghosts, not 4 as we used in the project.

```
module ghost_direction_writer(
    input [3:0][2:0] ghost_direction_fpga,
    output [15:0] ghost_direction_nios
);
```

Description: This module is the reverse of the previous module, it converts the packed array of directions of the ghosts from the FPGA to the 16 bit format for the

NIOS II processor.

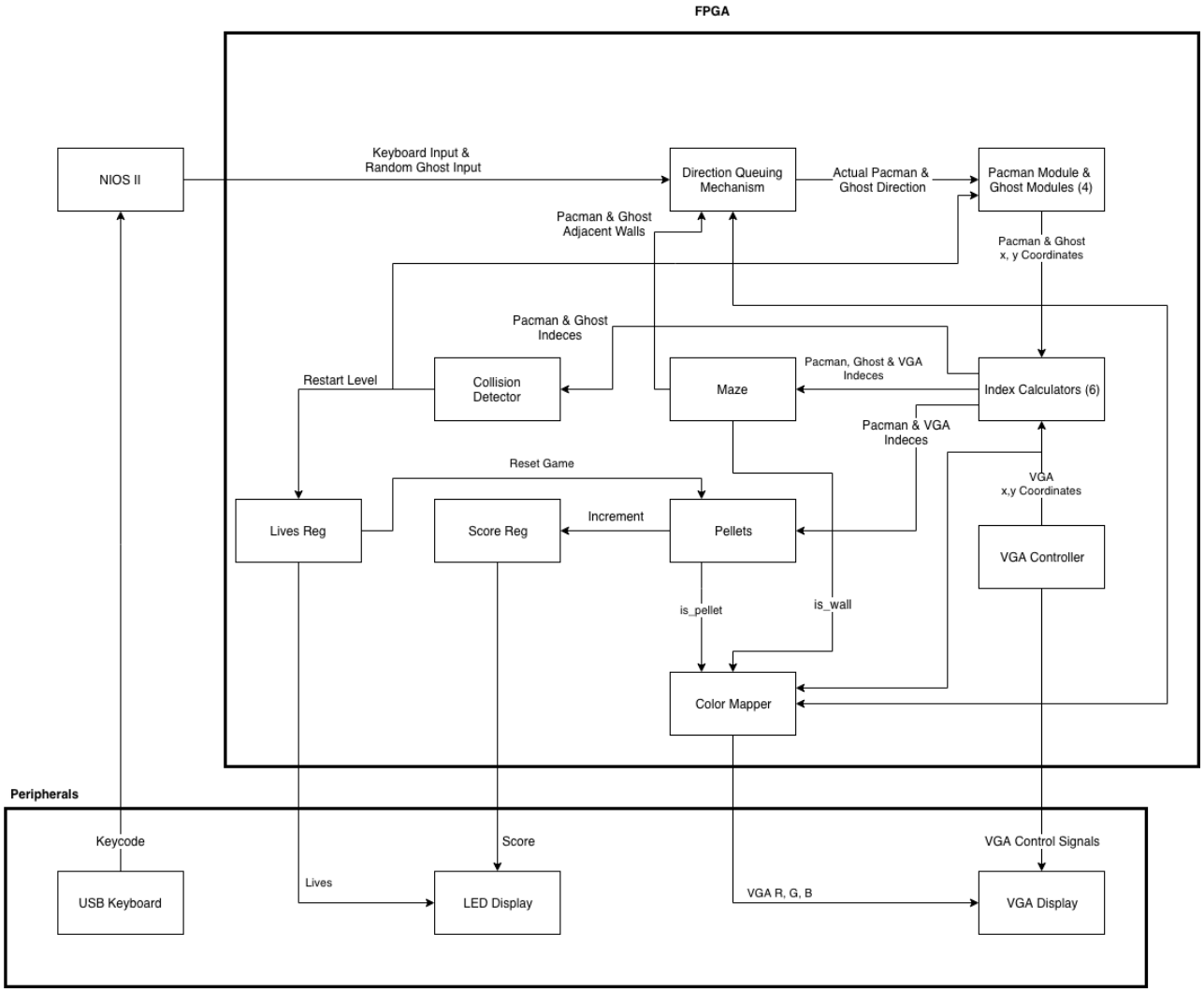
```
module index_calculator(
    input logic [9:0] x, y,
    output logic [10:0] index
);
```

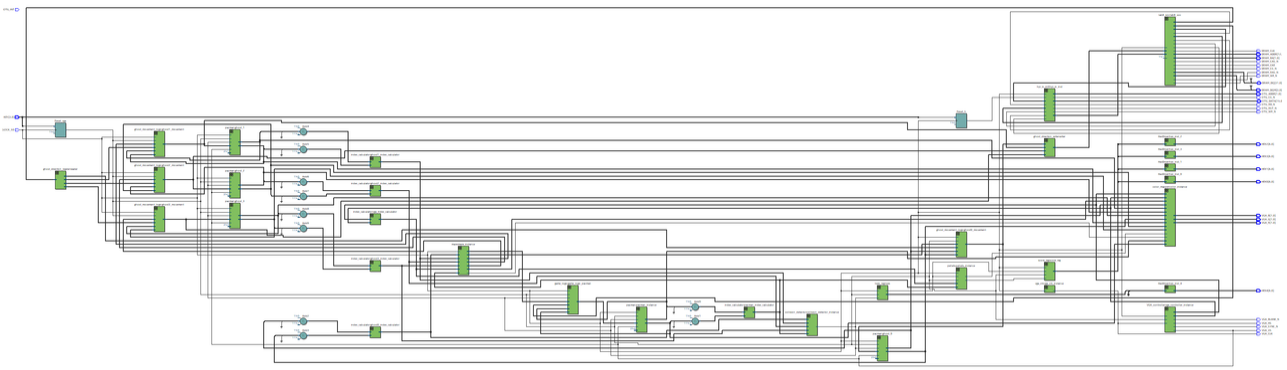
Description: This module is simply a converter from 2-d (x,y) coordinates into an index into the 30x40 maze. In order to represent indices up to 1200, 11 bits are needed for the index output.

```
module collision_detector(
    input [10:0]      pacman_index,
    input [3:0][10:0] ghost_index,
    output Restart
);
```

Description: This module determines whether Pac-Man has collided with a ghost using the corresponding indeces from the IndexCalculators. On a collision, the Restart signal is output to restart the level and the lives counter is decremented. On the loss of 3 lives, a full game start is ensued.

Block Diagram





Analysis

LUT	7,809
DSP	0

Memory(BRAM)	62,592
Flip-flop	3474
Frequency	131.79 MHz
Static Power	105.49 mW
Dynamic Power	0.88 mW
Total Power	183.42 mW

Conclusion

Our baseline implementation for reference is outlined below.

- Working VGA display showing Pac-Man game basics (maze, Pac-Man sprite, Ghost sprites)
- Lives and score showed on hex display
- Pac-Man is able to move around via keyboard input and collect pellets
  - Collecting pellets earns points
- Ghosts move around randomly
- Power Pellets change the behavior and appearance of Ghosts
  - Pac-Man will be able to kill Ghosts and send them back to the Ghost House
- Correct sprites display for all player and adversary motion
- On Pac-Man death, Pac-Man respawns at original position and can attempt to complete the game. Up to 3 lives. Full game reset on loss of last life.

We were able to stick closely to our baseline implementation, fulfilling 5/7 proposed items. With the exception of actual Pac-Man and ghost sprites and Power Pellets, we believe we successfully completed our baseline implementation and built a working version of Pac-Man on the FPGA. If we had more time to work on this project, we could have easily implemented the remainder of our baseline, as well as some of the advanced features.

If we were to change anything in this project, it would be to clean up some of the naming conventions and reduce some of the useless or extraneous variables/bits throughout the project.

Regarding the class as a whole, we wish there were fewer TTL labs and more labs in SystemVerilog. Specifically, a sprite lab would have been very helpful. Or, more time were given for the final project, with 2 checkpoints instead of just 1, since deadlines can be excellent motivators to produce results.

Overall, this project was quite interesting and fun to build and taught a lot about programming the FPGA.