BATTLESHIP!

Darian Primer & Alexander Goldstein
Cal Poly, San Luis Obispo. CPE 133: Digital Design

Introduction

This Basys3 version of *BATTLESHIP!* pits two players, “A” and “B”, against one another in a fight to the death. Both players get two ships (one 3-bit ship, one 2-bit ship), which they may place anywhere along a 10-bit “ocean.” Players take turns attacking each other’s positions; whoever sinks all “bits” of the enemy’s ships first wins!

In this paper, we present the hardware design, and the user guide, for the Basys3 *BATTLESHIP!.* We conclude with a discussion on the current limitations of the design and on possible future improvements.

Hardware Design

In this section we present the structural model and finite state machine designed to implement *BATTLESHIP!*. We implemented *BATTLESHIP!* using Vivado 2018.2 in SystemVerilog for use on the Artix 7 FPGA on the Digilent Basys 3 development board.

Structural Model

Appendix A shows the complete structural model for the *BATTLESHIP!’s* Master board. It makes use of a *Battleship FSM*, three 10-bit *registers*, one 10-bit *mux*, one 10-bit *comparator*, and a modified *Words2* module.

We designed all of the modules except for the *Words2* module and its *clk_divider* (designed by [1]), which we edited to modify the word shown on the seven-segment display.

- The *Battleship FSM* controls the pace of the game; it determines when to move from a setup (“load”) state to the gameplay, alternates between player turns, moves to the win/lose state, and also controls resetting the game. Importantly, the *Battleship FSM* is the only FSM in our design; it is loaded onto the Master board, but also controls the Slave board.
- The *Registers* hold ship positions (one ship = one bit).
    - *Register R1A* holds the ship positions of the Master (“A”) player
    - *Register R2A* holds the attack positions of A
    - *Register R3A* is somewhat useless; it holds ship positions to pass to an Input Checker. Because we removed the *Input Checker*, this register doesn’t currently serve a purpose, but if we develop the *Input Checker* in the future, it is good to have in place.
- The *Mux* is used to determine whether the board’s switches load ships into *R1A* (during the load), or if they indicate attack positions (during gameplay).
- The *Comparator* indicates whether the player is still alive; if the value of A’s ships is zero, then A has zero ships, and has died.

- The *Words2* module displays the following messages, to help both players understand what state the game is in:
    - "Load", "Play", "Halt", "Live", "Dead"

Finite State Machine

Appendix B shows the complete finite state machine for *BATTLESHIP!*. The *Battleship FSM* is made up of 7 distinct states; the FSM remains in each until a trigger condition is met. Each state is further explained below, with a general description of the state's function, followed by the role of the FSM in enabling this functionality:

- LOAD: Players A & B each have 10 positions (bits) in which to place their ships. Each "bit" of a ship is placed by turning the corresponding switch to the "on" position. (All other switches must be off.)
    - Triggers *Words2* to display "Load" on both A's and B's display.
    - When all ships have been placed, both players will press the center button *(BTN1A)* simultaneously; the FSM waits for this condition before advancing to the next state.
    - This state also acts as the "reset" state (eg. clears registers, signals *Words2* to display "Load", etc.), which is triggered when the FSM receives a "clr" signal from both players pressing the "clear" button *(BTN3)*.
- PLAYERA_LOAD: Now that the ship positions are set, Players A & B return all switches to the "off" position. Either player's "Player_LOAD" state is where the player decides which ship position to attack by enabling the corresponding switch.
    - The FSM triggers Player A's *Words2* to display "Play," and "Halt" on Player B's *Words2*.
    - The FSM waits until Player A presses the right-button *(BTN2A)* before advancing to the next state. (This allows the player to turn a switch on & off before committing to a decision.)
- PLAYERA_ATTACK: Neither player takes action in this state; the purpose of this state is to load Player A's attack into *R2A*, so that Player B's *Ships* register (*R1B)* can update those positions, if hit.
- PLAYERB_LOAD: Player B chooses which ship position to attack by enabling the corresponding switch.
    - The FSM trigger's Player A's *Words2* to display "Halt," and "Play" on Player B's *Words2*.
    - The FSM waits until Player B presses the right-button *(BTN2B)* before advancing to the next state.
- PLAYERB_ATTACK: Neither player takes action. Player B's attack is loaded into *R2B*, so that Player A's *Ships* register *(R1A)* can update those positions if hit.
- PLAYERA_WIN: When the *Comparator* indicates Player B is no longer alive, Player A has won.
    - This state can be reached from all states, except LOAD.
    - The FSM triggers *Words2* to display "Live" and "Dead" on Player A's / Player B's boards, respectively.

    - The FSM only exits this state if both players press the "Clear" button *(BTN3)* simultaneously, which returns to LOAD and therefore resets the game.
- PLAYERB_WIN: When the *Comparator* indicates Player A is no longer alive, Player B has won.
    - This state can be reached from all states, except LOAD.
    - The FSM triggers *Words2* to display "Dead" and "Live" on Player A's / Player B's boards, respectively.
    - The FSM only exits this state if both players press the "Clear" button *(BTN3)* simultaneously, which returns to LOAD and therefore resets the game.

User Guide

In this section, we describe how to use *BATTLESHIP!*. Figure 1 shows the user interface components for the design, and Table 1 describes their functionality. A guide (with pictures!) gives an overview for proper gameplay progression.

Figure 1: *User inputs and outputs*

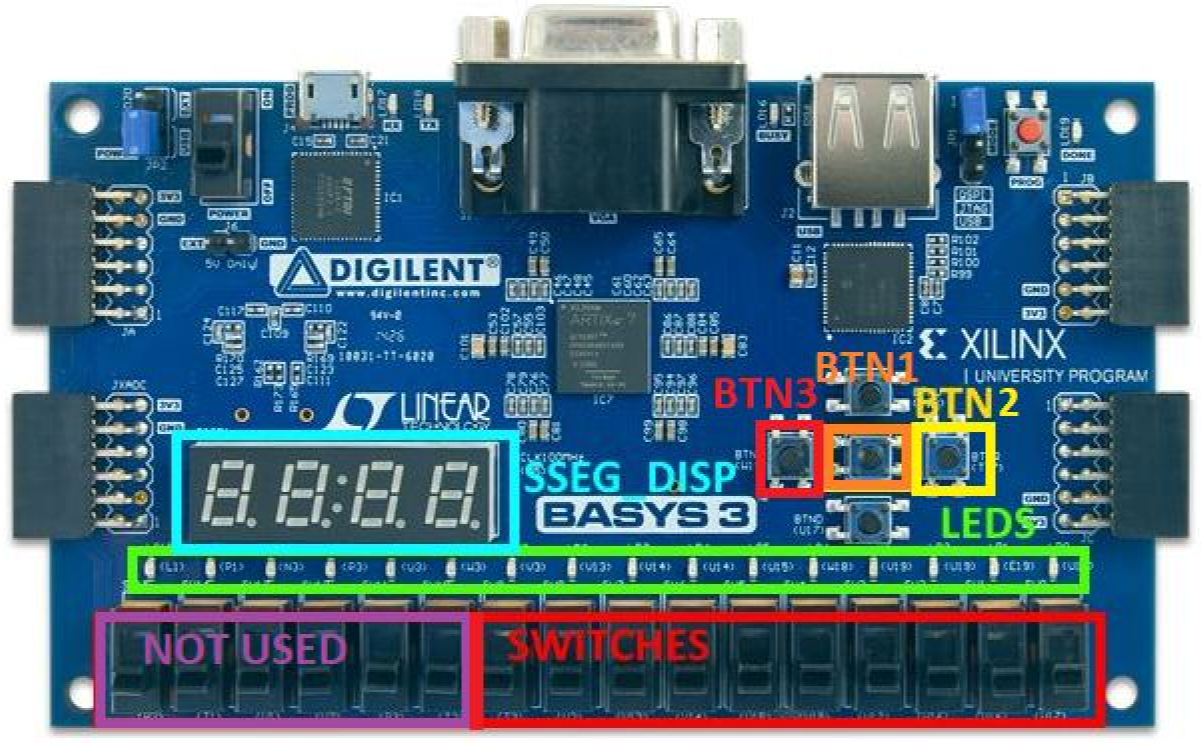


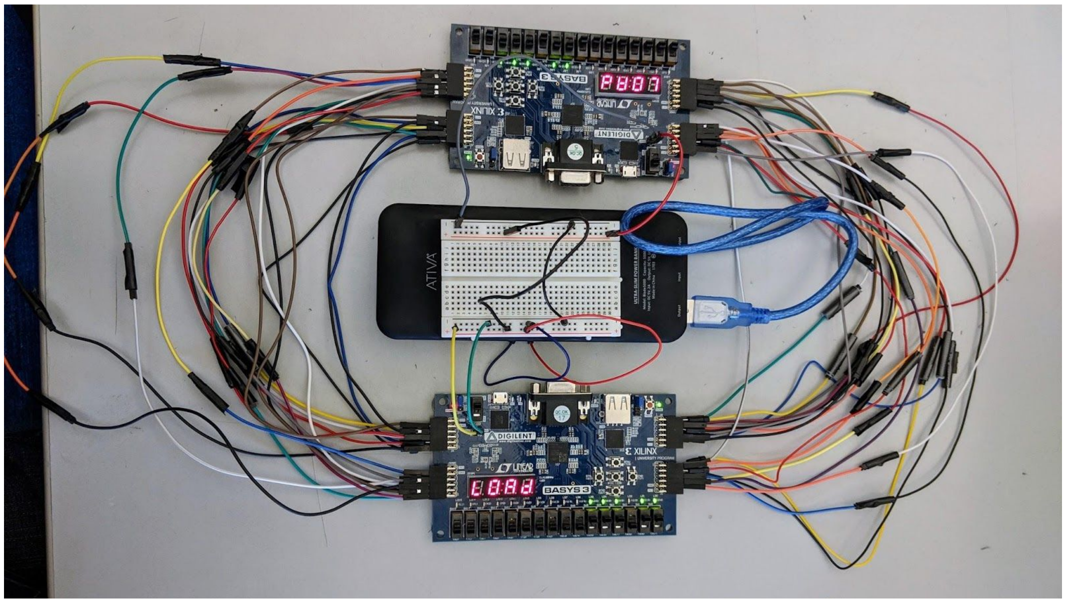
Table 1: *Description of user interface*

Component	**Description**
SSEG_DISP	Displays output of the *Words2* module, to inform Players about the current state of the game.
SWITCHES	Allows players to input ship positions at the start of the game, and select attack positions on their turn.
LEDS	Light up to indicate living ship positions. Upon being hit, the light goes out.
BTN1	Sets player ship positions at the beginning of the game, and begins play with Player A's turn.
BTN2	Confirms player's attack position, and advances to the opposing player's turn.
BTN3	Resets the game. Used at the end of a game to begin a new game. Also useful for players wishing to exit from a losing game.

Players should be positioned such that they cannot see each other's boards. This might include the use of a box or other divider, or at minimum, the use of an honor system.

The game begins in the LOAD state. Players must input their ship positions using the switches 1-10. (The switches are ordered from greatest to least, such that they go: 10, 9, 8, … 3, 2, 1.) Each player gets 5 "bits" to place, in groupings as a "two-bit" ship and a "three-bit" ship. All other switches must be in the "off" position; this is further indicated by the corresponding LED, which will only turn "on" if the switch is active. Figure 2 shows a valid setup for Players A & B.

Figure 2: LOAD state with valid 2-bit and 3-bit ships input



When both players are ready, they press their board's center button (BTN1) simultaneously, which loads the ships into the game's memory. Next, the players return all switches to the "off" position, to begin the game. (The ships will remain, as signified by the persistent LEDs.)

On a player's turn, their display will change to show the word "Play"; similarly, the opposing player will see the word "Halt," indicating that they must wait for the other player's turn to complete. (Figure 3) Each player is allowed to attack one position during their turn; to do this, they must enable the switch corresponding to a position, and press the right-button (BTN2) to "fire." If it is a hit, the opposing player will see one of their LEDs turn off. Players take turns attacking additional positions until either player has lost all of their ships, upon which the game will display "Live" and "Dead" to indicate which player won/lost. (Figure 4)

Players can press BTN3 to reset the game and play again.

Figure 3: Game in progress. "Play" and "halt" shown for Player A & B.

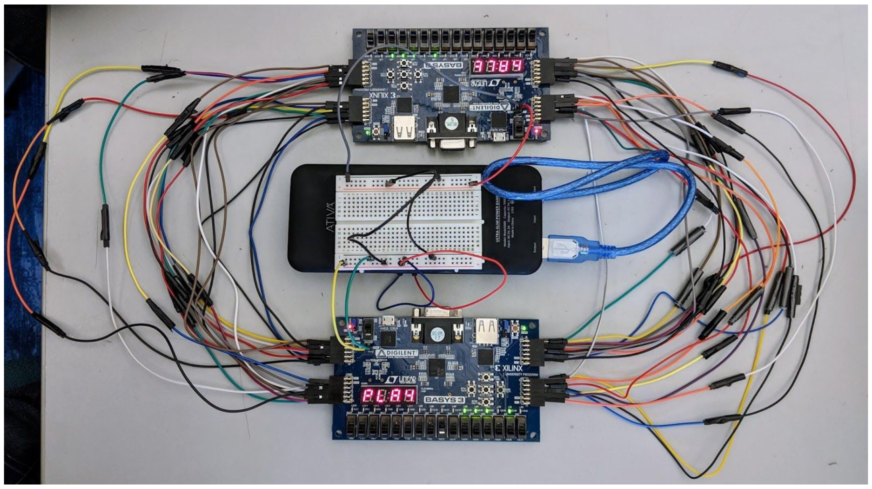
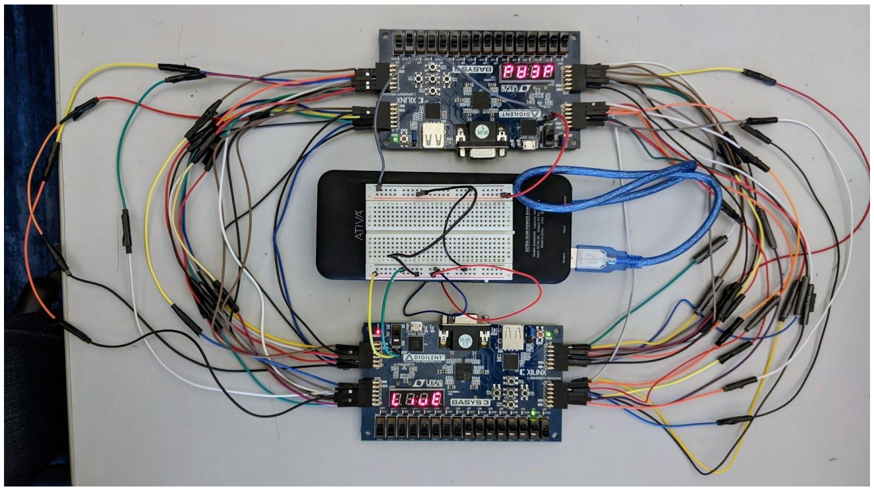


Figure 4: "Live" & "Dead" shown for Player A & B. This indicates the end of the game.



Troubleshooting

1. My boards are acting weird! Eg. Some ships I input disappear immediately after starting a game!
    a. Check that all wiring is correct. Note that it does matter which board is the master/slave, because the wiring is not symmetrical. If this does not resolve the problem, check that all wires are secure.
2. How do you wire this stuff anyway?!?
    a. Refer to Figure 5 & Figure 6 in the section on "Port Mapping" for a detailed PMOD wiring diagram.

Port Mapping

Figure 5: High-level port mapping overview for the Master/Slave PMOD wiring.

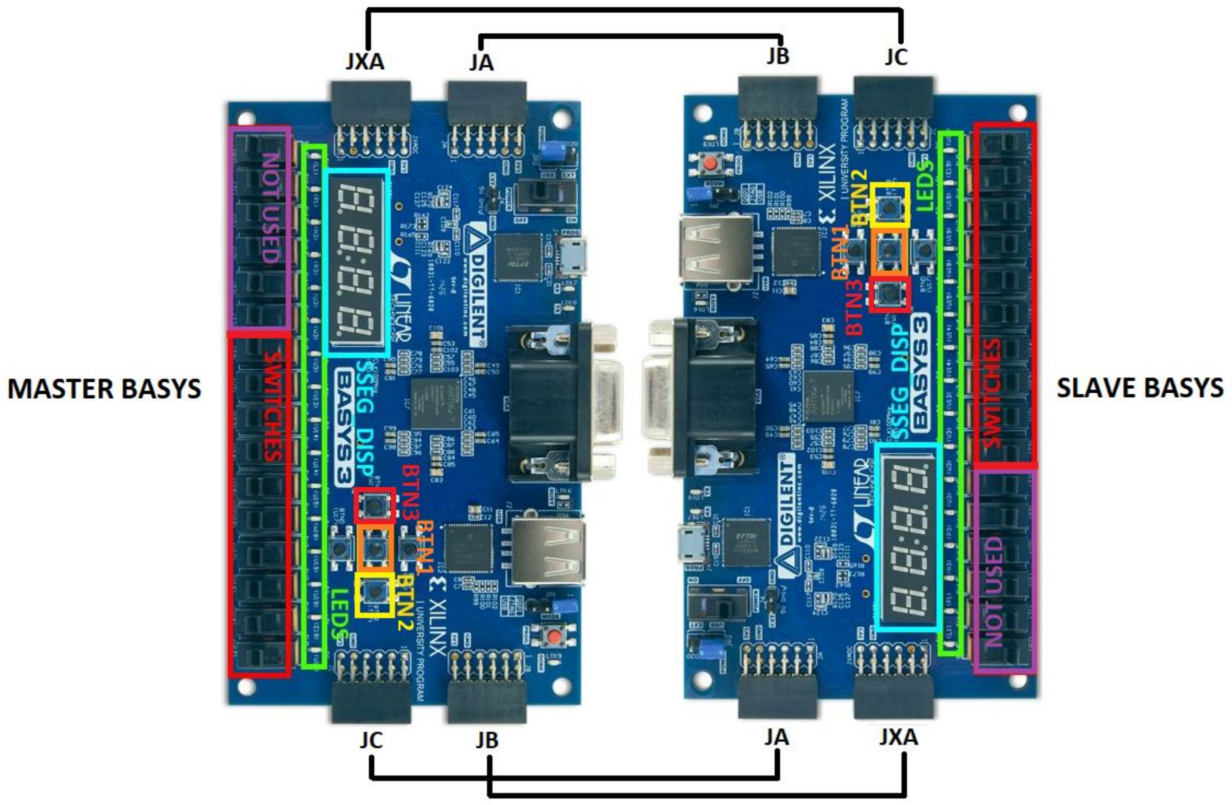


Figure 6: Low-level port mappings. (Each **MASTER PMOD** is listed in bold; non-bolded values (in each cell) indicate where that port maps to on the Slave PMOD.)

MASTER PMOD PORTS													
JA							**JXA**						
-	GND	JB4	JB3	JB2	JB1		-	GND	JC4	JC3	JC2	JC1	
-	GND	JB10	JB9	JB8	JB7		-	GND	JC10	JC9	JC8	JC7	

JC							JB					
-	GND	JA4	JA3	JA2	JA1		-	GND	JXA4	JXA3	JXA2	JXA1
-	GND	JA10	JA9	JA8	JA7		-	GND	JXA10	JXA9	JXA8	JXA7

Discussion

Our original design was for a 16-bit version of the game. Due to the limitation of available ports (The 10-bit version has zero ports to spare), this meant sending the 16-bit data between boards as a serial bitstream. We tried to create a UART, and eventually, a USART, but neither was reliable when integrated into the game.

The original design also featured an "input checker," which prevented a player from enabling more than one "attack" bit per turn; we bypassed this module in order to get the overall game working, and did not re-integrate it into the game. However, we tested the module separately, and it works independently, so we suspect it could be reintroduced -- especially as we did not remove the register that it relies upon (though this would require modifying the FSM, and might still be a complex operation).

After scaling the design down to 10 bits and removing the input checker, we got the game to function properly. Our original design from the brainstorm on day 1 (in class) worked as we expected, and we saw few major changes to our design, with the exception of these removals.

Limitations / Improvements

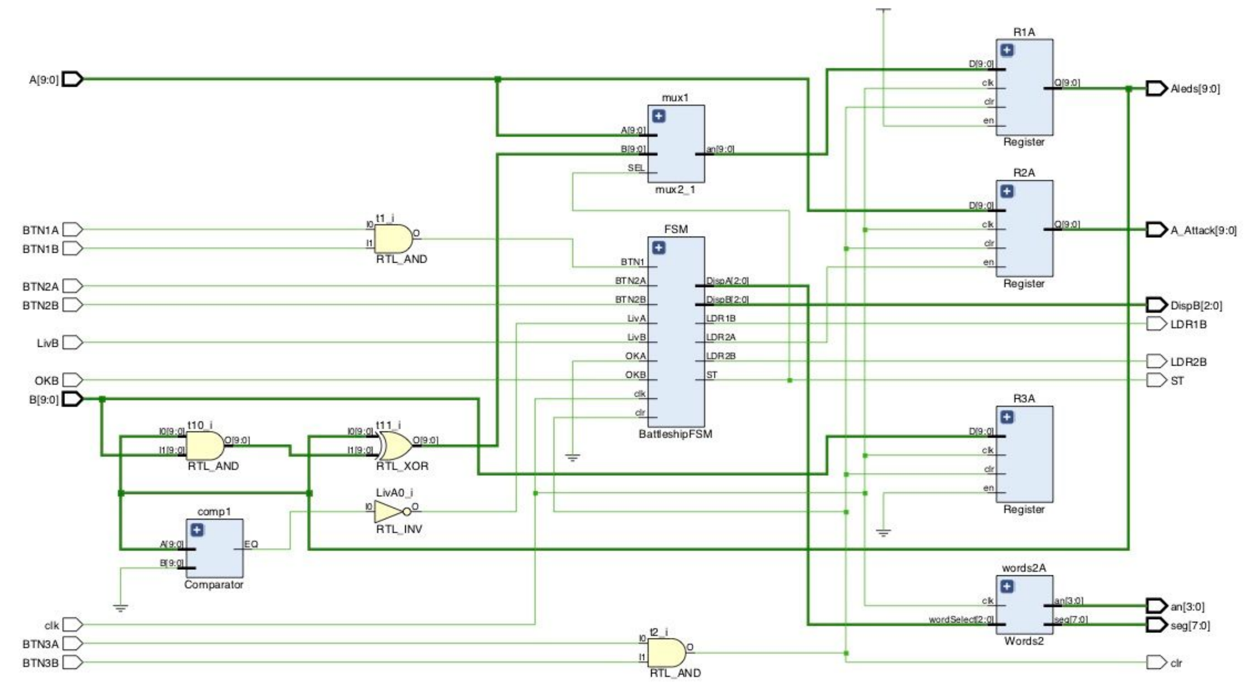
- Cannot go beyond 10-bit game. Requires many wires.
    - Working UART can solve this problem.
- Cannot prevent multiple attacks per turn, nor prevent a player from loading more than 5 bits into memory as their ships.
    - Integrating the input checker module into the design would fix this problem.
    - Alternative fix: Play with people who don't cheat.
    - As an aside, cheating is still somewhat detectable for the other player, because they might see multiple LEDs turn off during a single turn. Additionally, in our testing, this introduced an interesting mechanism, whereby if one player cheated and the other player realized this, they might respond by turning all switches "on," thereby winning against the cheater. Of course, there is no mechanism to prevent the cheater from activating *all* of their switches in the first turn; however, similarly, there is no mechanism to prevent the non-cheater from politely getting up, packing up the game, and finding a more agreeable partner to play against.

References

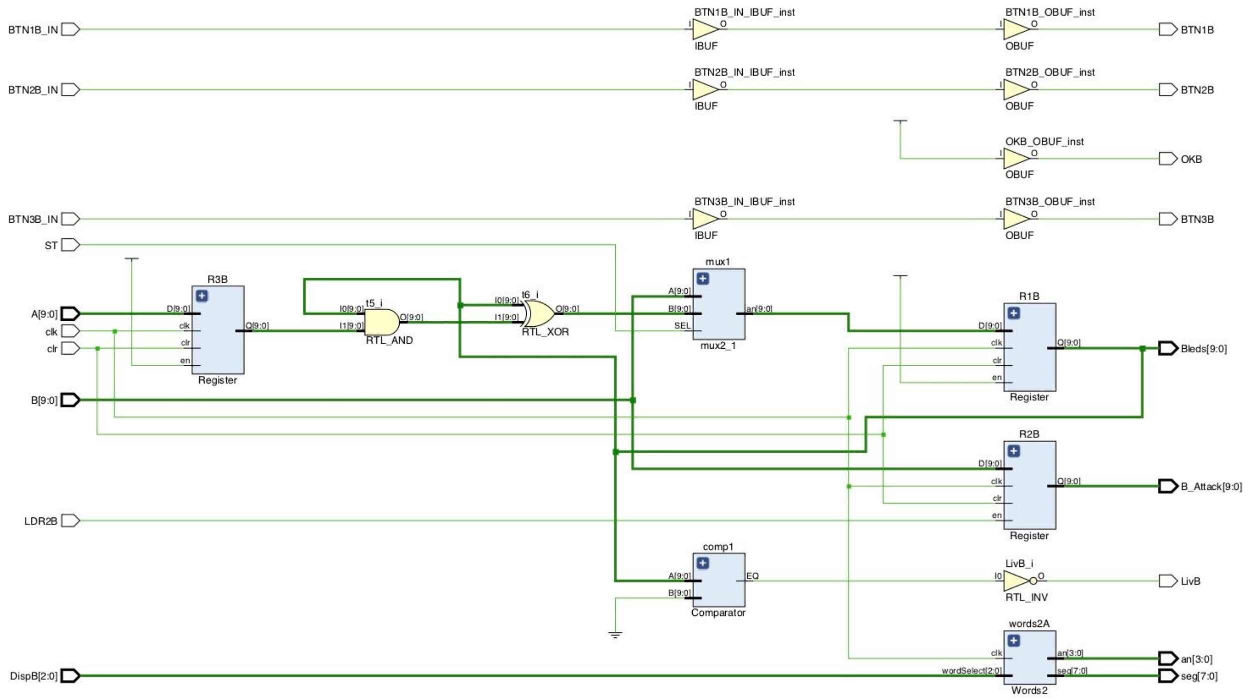
[1] Professor Bridget Benson. Cal Poly State University. Electrical and Computer Engineering. 2018.

Appendix A: Structural Model

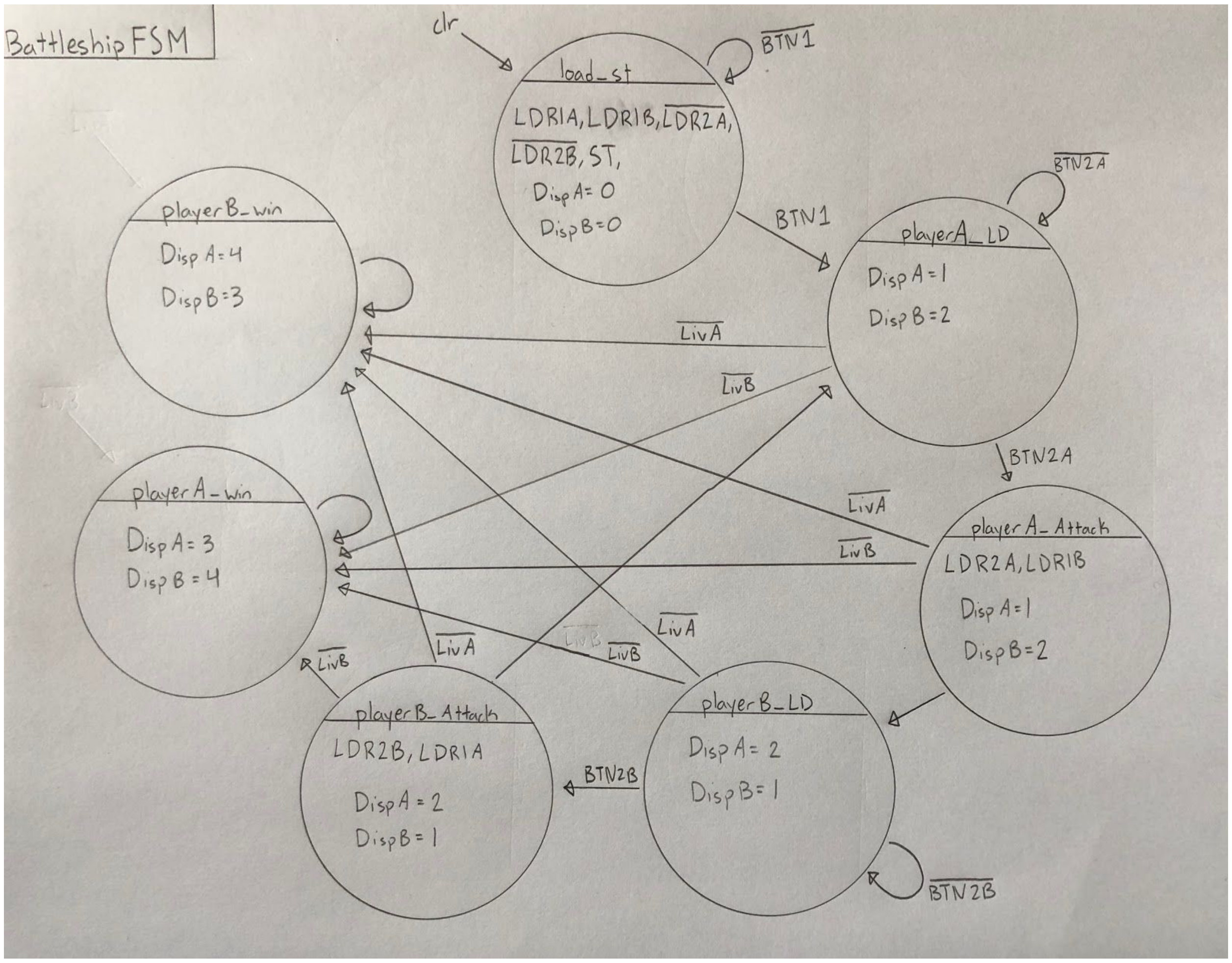
Master Board Schematic



Slave Board Schematic



Appendix B – Finite State Machine



Appendix C – System Verilog Code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO CPE 133
// Engineer: Darian Primer
//
// Create Date: 11/20/2018 01:22:30 AM
// Design Name: 10 bit Top file for Master basys board(player A).
//////////////////////////////////////////////////////////////////////////////////

module Master_Top10Bit(
    input clk,
    input [9:0] A,
    input [9:0] B, //slave-->master
    input BTN1A,
    input BTN1B,
    input BTN2A,
    input BTN2B,
    input LivB,
    input BTN3A,
    input BTN3B,
    input OKB,
    output LDR1B,
    output LDR2B,
    output [2:0] DispB,
    output clr,
    output [9:0] A_Attack, //master-->slave, signal from t5 on LDR2A (look at fsm)
    output [7:0] seg,
    output [3:0] an,
    output ST,
    output [9:0] Aleds //for testing
    );

    //declare all of the interconnects
    logic t1, t2, t3, t4, t5, t12, t13;
    logic [2:0] t6;
    logic [9:0] t7, t8, t9, t10, t11;

    //controls the BTN1 input of the mux. dictates when ships are to be loaded in load_st
    assign t1 = BTN1A & BTN1B;
    //controls the clearing af the FSM and all of the registers.
    assign t2 = BTN3A & BTN3B;
    //locates all of the positions where player A has been hit.
```

```
        assign t10 = t8 & B;
        //kills of all of the values where player A has been hit to push back to register 1A.
        assign t11 = t8 ^ t10;


        BattleshipFSM FSM(.clk(clk), .BTN1(t1), .BTN2A(BTN2A), .BTN2B(BTN2B), .LivA(~t13),
    .LivB(LivB), .clr(t2), .OKA(t12), .OKB(OKB), .LDR1B(LDR1B), .LDR2B(LDR2B), .DispB(DispB),
    .ST(t3), .LDR1A(t4), .LDR2A(t5), .DispA(t6));

        //This register saves the ship positions of player A
        Register #10 R1A(.clk(clk), .D(t7), .en(1), .clr(t2), .Q(t8));
        assign Aleds = t8;

        //This register saves the attack positions of player A
        logic [9:0] t_A_Attack;
        Register #10 R2A(.clk(clk), .D(A), .en(t5), .clr(t2), .Q(t_A_Attack));
        assign A_Attack = t_A_Attack;

        //This register saves the previous attack positions of player B to feed into the input checker.
        Register #10 R3A(.clk(clk), .D(B), .en(t12), .clr(t2), .Q(t9));

        //This mux selects if you are loading in your original ship positions or if you are loading in
    the positions that are still alive
        mux2_1 #10 mux1(.A(A), .B(t11), .SEL(t3), .an(t7));

        //The input checkers assures that player B only chose one new position to attack
        //InputChecker ICA(.A(t9), .B(B), .OK(t12));

        //Decides whether or not Player A is still alive
        Comparator #10 comp1(.A(t8), .B(10'b0000000000), .EQ(t13), .LT(), .GT());

        //Ouptuts game status to sseg for player A
        Words2 words2A(.clk(clk), .wordSelect(t6), .seg(seg), .an(an));

        //assign CLR output
        assign clr = t2;

        //assign ST output
        assign ST = t3;

        //assign A_ships output
        //assign A_Ships = t8;
```

```
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO CPE 133
// Engineer: Darian Primer
//
// Create Date: 11/20/2018 01:22:30 AM
// Design Name: 10 Bit Top file for Slave basys board(player B).
//////////////////////////////////////////////////////////////////////////////////
 module Slave_Top(
    input clk,
    input [9:0] A,
    input [9:0] B,
    //input LDR1B,
    //LDR1B is alwasy high because of delays to loading register one.
    //Register still holds correct value due to logic of the ckt.
    input LDR2B,
    input clr,
    input ST,
    input [2:0] DispB,
    input BTN1B_IN,
    input BTN2B_IN,
    input BTN3B_IN,
    output LivB,
    output OKB,
    output [9:0] B_Attack,
    output [7:0] seg,
    output [3:0] an,
    output BTN1B,
    output BTN2B,
    output BTN3B,
    output [9:0] Bleds // for testing
    );

    logic [9:0] t1, t2, t3, t4, t5, t6;
    logic t7, t8;

    //Decides if Register 0ne B is loading Ship positions or alive positions
    mux2_1 #10 mux1(.A(B), .B(t6), .SEL(ST), .an(t1));

    //This register saves the ship positions of player B
    Register #10 R1B(.clk(clk), .D(t1), .en(1), .clr(clr), .Q(t2));
    assign Bleds = t2;
    //This register saves the attack positions of player B
    Register #10 R2B(.clk(clk), .D(B), .en(LDR2B), .clr(clr), .Q(t4));

    //This register saves the previous attack positions of player A to feed into the input checker.
    logic [9:0] t_A = A;
    assign t7 = 1; //This wasn't being set when we disabled the input checker. R3B should always
be enabled.
    Register #10 R3B(.clk(clk), .D(t_A), .en(t7), .clr(clr), .Q(t3));

    //The input checkers assures that player A only chose one new position to attack
    //InputChecker ICA(.A(t3), .B(A), .OK(t7));

    //locates all of the positions where player B has been hit.
    assign t5 = t2 & t3;
    //kills of all of the values where player B has been hit to push back to register 1B.
    assign t6 = t2 ^ t5;

    //Decides whether or not Player A is still alive
    Comparator #10 comp1(.A(t2), .B(10'b0000000000), .EQ(t8), .LT(), .GT());

    //Ouptuts game status to sseg for player A
    Words2 words2A(.clk(clk), .wordSelect(DispB), .seg(seg), .an(an));

    //assign LivB
    assign LivB = !t8;

    //assign OKB
    assign OKB = t7;

    //assign B_Attack
    assign B_Attack = t4;

    //assign BTN1B
    assign BTN1B = BTN1B_IN;

    //assign BTN2B
    assign BTN2B = BTN2B_IN;

    //assign BTN3B
    assign BTN3B = BTN3B_IN;

    endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO CPE 133
// Engineer: Darian Primer
//
// Create Date: 11/26/2018 10:08:22 PM
// Description: This FSM decides the state for a basys board battleship game.
//              It controls all of the registers and the pace of the game.
//
//////////////////////////////////////////////////////////////////////////////////


module BattleshipFSM(
    input clk,
    input BTN1,
    input BTN2A,
    input BTN2B,
    input LivA,
    input LivB,
    input clr,
    input OKA,
    input OKB,
    output logic LDR1B,
    output logic LDR2B,
    output logic [2:0] DispB,
    output logic ST,
    output logic LDR1A,
    output logic LDR2A,
    output logic [2:0] DispA
    );

    logic [3:0] NS;
    logic [3:0] PS = load_st; //Present State

    parameter [3:0] load_st=0, playerA_LD=1, playerA_Attack=2, Redo_playerA=3,
                           playerB_LD=4, playerB_Attack=5, Redo_playerB=6,
                playerA_win=7, playerB_win=8;

     always_ff @ (posedge clk, posedge clr)
                    begin
                        PS <= NS;
                        if (clr==1)
                          PS <=load_st;
```

```
//                              else if (clr & LivA==0)
//                                PS <= playerB_win;
//                              else if (clr & LivB==0)
//                                PS <= playerA_win;
                          end


    always_comb
    begin

    //initialize outputs to zero
    LDR1A=0; LDR1B=0; LDR2A=0; LDR2B=0; DispA=0; DispB=0; ST=0;

    //list what happens in each of the states
    //if CLR goes high, moves back to load state
    case(PS)

        load_st:
          begin
          //Reset values if coming from other state.
          LDR2A=0; LDR2B=0;
          //this is where the players input their ship positions.
          //moves out of this state when both players press BTN1
          LDR1A=1;
          LDR1B=1;
          ST=1;
          DispA=0;
          DispB=0;
          if (BTN1==1)
             NS=playerA_LD;
          else
             NS=load_st;
          end

        playerA_LD:
          begin
          //this is where player A inputs where they want to attack
          //moves out of this state when player A presses BTN2
          DispA=1;
          DispB=2;
          //if (BTN2A==1)
             //NS=playerA_Attack;
                                   if (LivA==0)
```

```
                                        NS=playerB_win;
                        else if (LivB==0)
                                             NS=playerA_win;
                        else if (BTN2A==1)
        NS=playerA_Attack;
    else
      NS=playerA_LD;
    end

playerA_Attack:
    begin
    //this is where attack positions are loaded into player B's registers
    //and the validity of the entry is checked.
    //Game either moves on to player B, or Player A must redo their turn.
    //Moves out of this state depending on output of the input checker.
    LDR2A=1;
    LDR1B=1;
                        DispA=1;
                        DispB=2;
    //if (OKA==1)
      //NS=playerB_LD;
                        if (LivA==0)
                                        NS=playerB_win;
                        else if (LivB==0)
                                             NS=playerA_win;
                        else
                          NS=playerB_LD;
    //else
      //NS=Redo_playerA;
    end

Redo_playerA:
    begin
    //this forces player A to redo their turn because they did not change
    //only one attack position.
    //quickly displays error on player A's board, then waits for new entry.
    DispA=5;
    DispB=2;
    NS=playerA_LD;
    end

playerB_LD:
    begin
```

```
//this is where player B inputs where they want to attack
//moves out of this state when player B presses BTN2
DispA=2;
DispB=1;
//if (BTN2B==1)
    //NS=playerB_Attack;
                           if (LivA==0)
                                       NS=playerB_win;
                           else if (LivB==0)
                                            NS=playerA_win;
                           else if (BTN2B==1)
            NS=playerB_Attack;
else
    NS=playerB_LD;
end

playerB_Attack:
    begin
    //this is where attack positions are loaded into player A's registers
    //and the validity of the entry is checked.
    //Game either moves on to player A, or Player B must redo their turn.
    //Moves out of this state depending on output of the input checker.
    LDR2B=1;
    LDR1A=1;
                                 DispA=2;
                                 DispB=1;
    //if (OKB==1)
        //NS=playerA_LD;
                                 if (LivA==0)
                                       NS=playerB_win;
                                 else if (LivB==0)
                                       NS=playerA_win;
    else
      //  NS=Redo_playerB;
                                       NS=playerA_LD;
    end

 Redo_playerB:
    begin
    //this forces player B to redo their turn because they did not change
    //only one attack position.
    //quickly displays error on player B's board, then waits for new entry.
    DispA=2;
```

```
                DispB=5;
                NS=playerB_LD;
                end

                           playerA_win:
                               begin
                               //this displays that player A won and player B lost.
                               DispA=3;
                               DispB=4;
                               NS=playerA_win;
                               end

                           playerB_win:
                               begin
                               //this displays that player A won and player B lost.
                               DispA=4;
                               DispB=3;
                               NS=playerB_win;
                               end

          endcase
          end
        endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO CPE 133
// Engineer: Darian Primer
//
// Create Date: 10/25/2018 09:19:27 AM
// Description: Parameterized Register
//////////////////////////////////////////////////////////////////////////////////


module Register #(parameter WIDTH = 4)
    (input clk,
    input en,
    input clr,
    input [WIDTH-1:0] D,
    output logic [WIDTH-1:0] Q
    );

    always_ff @ (posedge clk, negedge clr)
    begin
        if(clr)
            Q<=0;
        else if(en)
            Q<=D;
    end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO CPE 133
// Engineer: Darian Primer
//
// Create Date: 10/11/2018 09:38:37 AM
// Description: 2 input mux
//////////////////////////////////////////////////////////////////////////////////


module mux2_1 #(parameter WIDTH = 4)
    (input [WIDTH-1:0] A, B,
    input SEL,
    output [WIDTH - 1:0] an
    );


    assign an = (SEL ? A : B);
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO CPE 133
// Engineer: Darian Primer
//
// Create Date: 11/20/2018 12:43:37 AM
// Design Name: Comparator Module
// Module Name: Comparator

//////////////////////////////////////////////////////////////////////////////////


module Comparator #(parameter WIDTH=16)
    (input [WIDTH-1:0] A,
    input [WIDTH-1:0] B,
    output logic EQ,
    output logic LT,
    output logic GT
    );

    always_comb
    begin
    if(A==B)
        begin
         EQ=1;
         LT=0;
        GT=0;
        end
    else if (A>B)
            begin
            EQ=0;
            LT=0;
            GT=1;
            end
            else if (A<B)
                begin
                EQ=0;
                LT=1;
                GT=0;
                end
                else
                    begin
                    EQ=0;
                    LT=0;
                    GT=0;
                    end
    end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Professor Benson edited by Alex Goldstein
// Create Date: 10/31/2018 09:58:33 AM
// Description: Displays up to 8 words on the seven segment display
// wordSelect
//  0: load
//  1: play
//  2: wait
//  3: live
//  4: die
//  5: error
//////////////////////////////////////////////////////////////////////////////////


module Words2(
    input clk,
    input [2:0] wordSelect,
    output logic [7:0] seg,
    output logic [3:0] an);

    logic sclk;
    logic [1:0] count = 0;

    clk_divder clkdiv2 (.clockin(clk), .clockout(sclk));

    parameter [2:0] wload=0, wplay=1, whalt=2, wlive=3, wdie=4, werror=5;

    always_ff @(posedge sclk)
    begin
        count = count + 1;
    end

    //rotate through the four displays
    always_comb
    begin
        case (count)
            0: an = 4'b1110;
            1: an = 4'b1101;
            2: an = 4'b1011;
            3: an = 4'b0111;
            default: an = 0;
        endcase
```

```
end

//choose the word to display
always_comb
begin
   case (wordSelect)
   wload:
      case (count)
      0: seg <= 8'b10000101; //d
      1: seg <= 8'b00010001; //a
      2: seg <= 8'b00000011; //o
      3: seg <= 8'b11100011; //l
      default: seg <= 0;
      endcase
   wplay:
      case (count)
      0: seg <= 8'b10011001; //y
      1: seg <= 8'b00010001; //a
      2: seg <= 8'b11100011; //l
      3: seg <= 8'b00110001; //p
      default: seg <= 0;
      endcase
   whalt:
      case(count)
      0: seg <= 8'b11100001; //t
      1: seg <= 8'b11100011; //l
      2: seg <= 8'b00010001; //a
      3: seg <= 8'b11010001; //h
      default: seg <= 0;
      endcase
   wlive:
      case(count)
      0: seg <= 8'b01100001; //e
      1: seg <= 8'b11000111; //v
      2: seg <= 8'b11011111; //i
      3: seg <= 8'b11100011; //l
      default: seg <= 0;
      endcase
   wdie:
      case(count)
      0: seg <= 8'b10000101; //d
      1: seg <= 8'b00010001; //a
      2: seg <= 8'b01100001; //e
```

```
            3: seg <= 8'b10000101; //d
            default: seg <= 0;
            endcase
        werror:
            case (count)
            0: seg <= 8'b11110101; //r
            1: seg <= 8'b11110101; //r
            2: seg <= 8'b01100001; //e
            3: seg <= 8'b11111111; //
            default: seg <= 0;
            endcase
        default:
            seg <= 8'b11111101;
      endcase
    end
endmodule
```

Battleship_Master10Bit.xdc

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal
names in the project

## Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
	set_property IOSTANDARD LVCMOS33 [get_ports clk]
	create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

## Switches
set_property PACKAGE_PIN V17 [get_ports {A[0]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {A[0]}]
set_property PACKAGE_PIN V16 [get_ports {A[1]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {A[1]}]
set_property PACKAGE_PIN W16 [get_ports {A[2]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {A[2]}]
set_property PACKAGE_PIN W17 [get_ports {A[3]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {A[3]}]
set_property PACKAGE_PIN W15 [get_ports {A[4]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {A[4]}]
set_property PACKAGE_PIN V15 [get_ports {A[5]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {A[5]}]
set_property PACKAGE_PIN W14 [get_ports {A[6]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {A[6]}]
set_property PACKAGE_PIN W13 [get_ports {A[7]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {A[7]}]
set_property PACKAGE_PIN V2 [get_ports {A[8]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {A[8]}]
set_property PACKAGE_PIN T3 [get_ports {A[9]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {A[9]}]
#set_property PACKAGE_PIN T2 [get_ports {A[10]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {A[10]}]
#set_property PACKAGE_PIN R3 [get_ports {A[11]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {A[11]}]
#set_property PACKAGE_PIN W2 [get_ports {A[12]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {A[12]}]
#set_property PACKAGE_PIN U1 [get_ports {A[13]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {A[13]}]
#set_property PACKAGE_PIN T1 [get_ports {A[14]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {A[14]}]
```

```
#set_property PACKAGE_PIN R2 [get_ports {A[15]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {A[15]}]


#LEDs
set_property PACKAGE_PIN U16 [get_ports {Aleds[0]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Aleds[0]}]
set_property PACKAGE_PIN E19 [get_ports {Aleds[1]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Aleds[1]}]
set_property PACKAGE_PIN U19 [get_ports {Aleds[2]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Aleds[2]}]
set_property PACKAGE_PIN V19 [get_ports {Aleds[3]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Aleds[3]}]
set_property PACKAGE_PIN W18 [get_ports {Aleds[4]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Aleds[4]}]
set_property PACKAGE_PIN U15 [get_ports {Aleds[5]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Aleds[5]}]
set_property PACKAGE_PIN U14 [get_ports {Aleds[6]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Aleds[6]}]
set_property PACKAGE_PIN V14 [get_ports {Aleds[7]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Aleds[7]}]
set_property PACKAGE_PIN V13 [get_ports {Aleds[8]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Aleds[8]}]
set_property PACKAGE_PIN V3 [get_ports {Aleds[9]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Aleds[9]}]
#set_property PACKAGE_PIN W3 [get_ports {Breceived[10]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {Breceived[10]}]
#set_property PACKAGE_PIN U3 [get_ports {Breceived[11]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {Breceived[11]}]
#set_property PACKAGE_PIN P3 [get_ports {Breceived[12]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {Breceived[12]}]
#set_property PACKAGE_PIN N3 [get_ports {Breceived[13]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {Breceived[13]}]
#set_property PACKAGE_PIN P1 [get_ports {Breceived[14]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {Breceived[14]}]
#set_property PACKAGE_PIN L1 [get_ports {Breceived[15]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {Breceived[15]}]


##7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[7]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {seg[7]}]
set_property PACKAGE_PIN W6 [get_ports {seg[6]}]
```

```
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]
set_property PACKAGE_PIN U8 [get_ports {seg[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN V8 [get_ports {seg[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN U5 [get_ports {seg[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN V5 [get_ports {seg[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN U7 [get_ports {seg[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]

set_property PACKAGE_PIN V7 [get_ports {seg[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]

set_property PACKAGE_PIN U2 [get_ports {an[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]


#Buttons
set_property PACKAGE_PIN U18 [get_ports BTN1A]
        set_property IOSTANDARD LVCMOS33 [get_ports BTN1A]
#set_property PACKAGE_PIN T18 [get_ports BTN2A]
        #set_property IOSTANDARD LVCMOS33 [get_ports BTN2A]
set_property PACKAGE_PIN W19 [get_ports BTN3A]
        set_property IOSTANDARD LVCMOS33 [get_ports BTN3A]
set_property PACKAGE_PIN T17 [get_ports BTN2A]
        set_property IOSTANDARD LVCMOS33 [get_ports BTN2A]
#set_property PACKAGE_PIN U17 [get_ports btnD]
        #set_property IOSTANDARD LVCMOS33 [get_ports btnD]



#Pmod Header JA
#Sch name = JA1
set_property PACKAGE_PIN J1 [get_ports {B[0]}]
```

```
          set_property IOSTANDARD LVCMOS33 [get_ports {B[0]}]
##Sch name = JA2
set_property PACKAGE_PIN L2 [get_ports {B[1]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {B[1]}]
#Sch name = JA3
set_property PACKAGE_PIN J2 [get_ports {B[2]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {B[2]}]
#Sch name = JA4
set_property PACKAGE_PIN G2 [get_ports {B[3]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {B[3]}]
#Sch name = JA7
set_property PACKAGE_PIN H1 [get_ports {B[4]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {B[4]}]
###Sch name = JA8
set_property PACKAGE_PIN K2 [get_ports {B[5]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {B[5]}]
##Sch name = JA9
set_property PACKAGE_PIN H2 [get_ports {B[6]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {B[6]}]
##Sch name = JA10
set_property PACKAGE_PIN G3 [get_ports {B[7]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {B[7]}]



#Pmod Header JB
#Sch name = JB1
set_property PACKAGE_PIN A14 [get_ports {A_Attack[6]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {A_Attack[6]}]
#Sch name = JB2
set_property PACKAGE_PIN A16 [get_ports {A_Attack[7]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {A_Attack[7]}]
#Sch name = JB3
set_property PACKAGE_PIN B15 [get_ports {A_Attack[8]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {A_Attack[8]}]
#Sch name = JB4
set_property PACKAGE_PIN B16 [get_ports {A_Attack[9]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {A_Attack[9]}]
#Sch name = JB7
set_property PACKAGE_PIN A15 [get_ports {BTN1B}]
          set_property IOSTANDARD LVCMOS33 [get_ports {BTN1B}]
##Sch name = JB8
set_property PACKAGE_PIN A17 [get_ports {BTN2B}]
```

```
        set_property IOSTANDARD LVCMOS33 [get_ports {BTN2B}]
##Sch name = JB9
set_property PACKAGE_PIN C15 [get_ports {BTN3B}]
        set_property IOSTANDARD LVCMOS33 [get_ports {BTN3B}]
##Sch name = JB10
set_property PACKAGE_PIN C16 [get_ports {LivB}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LivB}]



#Pmod Header JC
#Sch name = JC1
set_property PACKAGE_PIN K17 [get_ports {OKB}]
        set_property IOSTANDARD LVCMOS33 [get_ports {OKB}]
#Sch name = JC2
set_property PACKAGE_PIN M18 [get_ports {DispB[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {DispB[0]}]
#Sch name = JC3
set_property PACKAGE_PIN N17 [get_ports {DispB[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {DispB[1]}]
#Sch name = JC4
set_property PACKAGE_PIN P18 [get_ports {DispB[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {DispB[2]}]
#Sch name = JC7
set_property PACKAGE_PIN L17 [get_ports {LDR1B}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LDR1B}]
#Sch name = JC8
set_property PACKAGE_PIN M19 [get_ports {LDR2B}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LDR2B}]
#Sch name = JC9
set_property PACKAGE_PIN P17 [get_ports {clr}]
        set_property IOSTANDARD LVCMOS33 [get_ports {clr}]
#Sch name = JC10
set_property PACKAGE_PIN R18 [get_ports {ST}]
        set_property IOSTANDARD LVCMOS33 [get_ports {ST}]


#Pmod Header JXADC
#Sch name = XA1_P
set_property PACKAGE_PIN J3 [get_ports {B[8]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {B[8]}]
#Sch name = XA2_P
set_property PACKAGE_PIN L3 [get_ports {B[9]}]
```

```
	set_property IOSTANDARD LVCMOS33 [get_ports {B[9]}]
#Sch name = XA3_P
set_property PACKAGE_PIN M2 [get_ports {A_Attack[0]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {A_Attack[0]}]
#Sch name = XA4_P
set_property PACKAGE_PIN N2 [get_ports {A_Attack[1]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {A_Attack[1]}]
#Sch name = XA1_N
set_property PACKAGE_PIN K3 [get_ports {A_Attack[2]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {A_Attack[2]}]
#Sch name = XA2_N
set_property PACKAGE_PIN M3 [get_ports {A_Attack[3]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {A_Attack[3]}]
#Sch name = XA3_N
set_property PACKAGE_PIN M1 [get_ports {A_Attack[4]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {A_Attack[4]}]
#Sch name = XA4_N
set_property PACKAGE_PIN N1 [get_ports {A_Attack[5]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {A_Attack[5]}]



##VGA Connector
#set_property PACKAGE_PIN G19 [get_ports {vgaRed[0]}]
	#set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[0]}]
#set_property PACKAGE_PIN H19 [get_ports {vgaRed[1]}]
	#set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[1]}]
#set_property PACKAGE_PIN J19 [get_ports {vgaRed[2]}]
	#set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[2]}]
#set_property PACKAGE_PIN N19 [get_ports {vgaRed[3]}]
	#set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[3]}]
#set_property PACKAGE_PIN N18 [get_ports {vgaBlue[0]}]
	#set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[0]}]
#set_property PACKAGE_PIN L18 [get_ports {vgaBlue[1]}]
	#set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[1]}]
#set_property PACKAGE_PIN K18 [get_ports {vgaBlue[2]}]
	#set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[2]}]
#set_property PACKAGE_PIN J18 [get_ports {vgaBlue[3]}]
	#set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[3]}]
#set_property PACKAGE_PIN J17 [get_ports {vgaGreen[0]}]
	#set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[0]}]
#set_property PACKAGE_PIN H17 [get_ports {vgaGreen[1]}]
	#set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[1]}]
```

```
#set_property PACKAGE_PIN G17 [get_ports {vgaGreen[2]}]
          #set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[2]}]
#set_property PACKAGE_PIN D17 [get_ports {vgaGreen[3]}]
          #set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[3]}]
#set_property PACKAGE_PIN P19 [get_ports Hsync]
          #set_property IOSTANDARD LVCMOS33 [get_ports Hsync]
#set_property PACKAGE_PIN R19 [get_ports Vsync]
          #set_property IOSTANDARD LVCMOS33 [get_ports Vsync]


##USB-RS232 Interface
#set_property PACKAGE_PIN B18 [get_ports RsRx]
          #set_property IOSTANDARD LVCMOS33 [get_ports RsRx]
#set_property PACKAGE_PIN A18 [get_ports RsTx]
          #set_property IOSTANDARD LVCMOS33 [get_ports RsTx]


##USB HID (PS/2)
#set_property PACKAGE_PIN C17 [get_ports PS2Clk]
          #set_property IOSTANDARD LVCMOS33 [get_ports PS2Clk]
          #set_property PULLUP true [get_ports PS2Clk]
#set_property PACKAGE_PIN B17 [get_ports PS2Data]
          #set_property IOSTANDARD LVCMOS33 [get_ports PS2Data]
          #set_property PULLUP true [get_ports PS2Data]


##Quad SPI Flash
##Note that CCLK_0 cannot be placed in 7 series devices. You can access it using the
##STARTUPE2 primitive.
#set_property PACKAGE_PIN D18 [get_ports {QspiDB[0]}]
          #set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[0]}]
#set_property PACKAGE_PIN D19 [get_ports {QspiDB[1]}]
          #set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[1]}]
#set_property PACKAGE_PIN G18 [get_ports {QspiDB[2]}]
          #set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[2]}]
#set_property PACKAGE_PIN F18 [get_ports {QspiDB[3]}]
          #set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[3]}]
#set_property PACKAGE_PIN K19 [get_ports QspiCSn]
          #set_property IOSTANDARD LVCMOS33 [get_ports QspiCSn]
```

Battleship_Slave10Bit.xdc

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal
names in the project

## Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
	set_property IOSTANDARD LVCMOS33 [get_ports clk]
	create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
#set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]

## Switches
set_property PACKAGE_PIN V17 [get_ports {B[0]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {B[0]}]
set_property PACKAGE_PIN V16 [get_ports {B[1]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {B[1]}]
set_property PACKAGE_PIN W16 [get_ports {B[2]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {B[2]}]
set_property PACKAGE_PIN W17 [get_ports {B[3]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {B[3]}]
set_property PACKAGE_PIN W15 [get_ports {B[4]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {B[4]}]
set_property PACKAGE_PIN V15 [get_ports {B[5]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {B[5]}]
set_property PACKAGE_PIN W14 [get_ports {B[6]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {B[6]}]
set_property PACKAGE_PIN W13 [get_ports {B[7]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {B[7]}]
set_property PACKAGE_PIN V2 [get_ports {B[8]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {B[8]}]
set_property PACKAGE_PIN T3 [get_ports {B[9]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {B[9]}]
#set_property PACKAGE_PIN T2 [get_ports {B[10]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {B[10]}]
#set_property PACKAGE_PIN R3 [get_ports {B[11]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {B[11]}]
#set_property PACKAGE_PIN W2 [get_ports {B[12]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {B[12]}]
#set_property PACKAGE_PIN U1 [get_ports {B[13]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {B[13]}]
#set_property PACKAGE_PIN T1 [get_ports {B[14]}]
```

```
#	set_property IOSTANDARD LVCMOS33 [get_ports {B[14]}]
#set_property PACKAGE_PIN R2 [get_ports {B[15]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {B[15]}]


#LEDs
set_property PACKAGE_PIN U16 [get_ports {Bleds[0]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Bleds[0]}]
set_property PACKAGE_PIN E19 [get_ports {Bleds[1]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Bleds[1]}]
set_property PACKAGE_PIN U19 [get_ports {Bleds[2]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Bleds[2]}]
set_property PACKAGE_PIN V19 [get_ports {Bleds[3]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Bleds[3]}]
set_property PACKAGE_PIN W18 [get_ports {Bleds[4]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Bleds[4]}]
set_property PACKAGE_PIN U15 [get_ports {Bleds[5]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Bleds[5]}]
set_property PACKAGE_PIN U14 [get_ports {Bleds[6]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Bleds[6]}]
set_property PACKAGE_PIN V14 [get_ports {Bleds[7]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Bleds[7]}]
set_property PACKAGE_PIN V13 [get_ports {Bleds[8]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Bleds[8]}]
set_property PACKAGE_PIN V3 [get_ports {Bleds[9]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {Bleds[9]}]
#set_property PACKAGE_PIN W3 [get_ports {Bleds[10]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {Bleds[10]}]
#set_property PACKAGE_PIN U3 [get_ports {Bleds[11]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {Bleds[11]}]
#set_property PACKAGE_PIN P3 [get_ports {Bleds[12]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {Bleds[12]}]
#set_property PACKAGE_PIN N3 [get_ports {Bleds[13]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {Bleds[13]}]
#set_property PACKAGE_PIN P1 [get_ports {Bleds[14]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {Bleds[14]}]
#set_property PACKAGE_PIN L1 [get_ports {Bleds[15]}]
#	set_property IOSTANDARD LVCMOS33 [get_ports {Bleds[15]}]


##7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[7]}]
	set_property IOSTANDARD LVCMOS33 [get_ports {seg[7]}]
set_property PACKAGE_PIN W6 [get_ports {seg[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]
set_property PACKAGE_PIN U8 [get_ports {seg[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN V8 [get_ports {seg[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN U5 [get_ports {seg[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN V5 [get_ports {seg[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN U7 [get_ports {seg[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]

set_property PACKAGE_PIN V7 [get_ports {seg[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]

set_property PACKAGE_PIN U2 [get_ports {an[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]


#Buttons
set_property PACKAGE_PIN U18 [get_ports BTN1B_IN]
        set_property IOSTANDARD LVCMOS33 [get_ports BTN1B_IN]
#set_property PACKAGE_PIN T18 [get_ports BTN2B_IN]
#       set_property IOSTANDARD LVCMOS33 [get_ports BTN2B_IN]
set_property PACKAGE_PIN W19 [get_ports BTN3B_IN]
        set_property IOSTANDARD LVCMOS33 [get_ports BTN3B_IN]
set_property PACKAGE_PIN T17 [get_ports BTN2B_IN]
        set_property IOSTANDARD LVCMOS33 [get_ports BTN2B_IN]
#set_property PACKAGE_PIN U17 [get_ports btnD]
        #set_property IOSTANDARD LVCMOS33 [get_ports btnD]


#Pmod Header JA
#Sch name = JA1
set_property PACKAGE_PIN J1 [get_ports {OKB}]
```

```
            set_property IOSTANDARD LVCMOS33 [get_ports {OKB}]
##Sch name = JA2
set_property PACKAGE_PIN L2 [get_ports {DispB[0]}]
            set_property IOSTANDARD LVCMOS33 [get_ports {DispB[0]}]
#Sch name = JA3
set_property PACKAGE_PIN J2 [get_ports {DispB[1]}]
            set_property IOSTANDARD LVCMOS33 [get_ports {DispB[1]}]
#Sch name = JA4
set_property PACKAGE_PIN G2 [get_ports {DispB[2]}]
            set_property IOSTANDARD LVCMOS33 [get_ports {DispB[2]}]
#Sch name = JA7
#set_property PACKAGE_PIN H1 [get_ports {LDR1B}]
#           set_property IOSTANDARD LVCMOS33 [get_ports {LDR1B}]
###Sch name = JA8
set_property PACKAGE_PIN K2 [get_ports {LDR2B}]
            set_property IOSTANDARD LVCMOS33 [get_ports {LDR2B}]
##Sch name = JA9
set_property PACKAGE_PIN H2 [get_ports {clr}]
            set_property IOSTANDARD LVCMOS33 [get_ports {clr}]
##Sch name = JA10
set_property PACKAGE_PIN G3 [get_ports {ST}]
            set_property IOSTANDARD LVCMOS33 [get_ports {ST}]



#Pmod Header JB
#Sch name = JB1
set_property PACKAGE_PIN A14 [get_ports {B_Attack[0]}]
            set_property IOSTANDARD LVCMOS33 [get_ports {B_Attack[0]}]
#Sch name = JB2
set_property PACKAGE_PIN A16 [get_ports {B_Attack[1]}]
            set_property IOSTANDARD LVCMOS33 [get_ports {B_Attack[1]}]
#Sch name = JB3
set_property PACKAGE_PIN B15 [get_ports {B_Attack[2]}]
            set_property IOSTANDARD LVCMOS33 [get_ports {B_Attack[2]}]
#Sch name = JB4
set_property PACKAGE_PIN B16 [get_ports {B_Attack[3]}]
            set_property IOSTANDARD LVCMOS33 [get_ports {B_Attack[3]}]
#Sch name = JB7
set_property PACKAGE_PIN A15 [get_ports {B_Attack[4]}]
            set_property IOSTANDARD LVCMOS33 [get_ports {B_Attack[4]}]
##Sch name = JB8
set_property PACKAGE_PIN A17 [get_ports {B_Attack[5]}]
```

```
          set_property IOSTANDARD LVCMOS33 [get_ports {B_Attack[5]}]
##Sch name = JB9
set_property PACKAGE_PIN C15 [get_ports {B_Attack[6]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {B_Attack[6]}]
##Sch name = JB10
set_property PACKAGE_PIN C16 [get_ports {B_Attack[7]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {B_Attack[7]}]



#Pmod Header JC
#Sch name = JC1
set_property PACKAGE_PIN K17 [get_ports {B_Attack[8]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {B_Attack[8]}]
#Sch name = JC2
set_property PACKAGE_PIN M18 [get_ports {B_Attack[9]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {B_Attack[9]}]
#Sch name = JC3
set_property PACKAGE_PIN N17 [get_ports {A[0]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {A[0]}]
#Sch name = JC4
set_property PACKAGE_PIN P18 [get_ports {A[1]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {A[1]}]
#Sch name = JC7
set_property PACKAGE_PIN L17 [get_ports {A[2]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {A[2]}]
#Sch name = JC8
set_property PACKAGE_PIN M19 [get_ports {A[3]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {A[3]}]
#Sch name = JC9
set_property PACKAGE_PIN P17 [get_ports {A[4]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {A[4]}]
#Sch name = JC10
set_property PACKAGE_PIN R18 [get_ports {A[5]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {A[5]}]


#Pmod Header JXADC
#Sch name = XA1_P
set_property PACKAGE_PIN J3 [get_ports {A[6]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {A[6]}]
#Sch name = XA2_P
set_property PACKAGE_PIN L3 [get_ports {A[7]}]
```

```
        set_property IOSTANDARD LVCMOS33 [get_ports {A[7]}]
#Sch name = XA3_P
set_property PACKAGE_PIN M2 [get_ports {A[8]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {A[8]}]
#Sch name = XA4_P
set_property PACKAGE_PIN N2 [get_ports {A[9]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {A[9]}]
#Sch name = XA1_N
set_property PACKAGE_PIN K3 [get_ports {BTN1B}]
        set_property IOSTANDARD LVCMOS33 [get_ports {BTN1B}]
#Sch name = XA2_N
set_property PACKAGE_PIN M3 [get_ports {BTN2B}]
        set_property IOSTANDARD LVCMOS33 [get_ports {BTN2B}]
#Sch name = XA3_N
set_property PACKAGE_PIN M1 [get_ports {BTN3B}]
        set_property IOSTANDARD LVCMOS33 [get_ports {BTN3B}]
#Sch name = XA4_N
set_property PACKAGE_PIN N1 [get_ports {LivB}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LivB}]



##VGA Connector
#set_property PACKAGE_PIN G19 [get_ports {vgaRed[0]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[0]}]
#set_property PACKAGE_PIN H19 [get_ports {vgaRed[1]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[1]}]
#set_property PACKAGE_PIN J19 [get_ports {vgaRed[2]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[2]}]
#set_property PACKAGE_PIN N19 [get_ports {vgaRed[3]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[3]}]
#set_property PACKAGE_PIN N18 [get_ports {vgaBlue[0]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[0]}]
#set_property PACKAGE_PIN L18 [get_ports {vgaBlue[1]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[1]}]
#set_property PACKAGE_PIN K18 [get_ports {vgaBlue[2]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[2]}]
#set_property PACKAGE_PIN J18 [get_ports {vgaBlue[3]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[3]}]
#set_property PACKAGE_PIN J17 [get_ports {vgaGreen[0]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[0]}]
#set_property PACKAGE_PIN H17 [get_ports {vgaGreen[1]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[1]}]
```

```
#set_property PACKAGE_PIN G17 [get_ports {vgaGreen[2]}]
	#set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[2]}]
#set_property PACKAGE_PIN D17 [get_ports {vgaGreen[3]}]
	#set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[3]}]
#set_property PACKAGE_PIN P19 [get_ports Hsync]
	#set_property IOSTANDARD LVCMOS33 [get_ports Hsync]
#set_property PACKAGE_PIN R19 [get_ports Vsync]
	#set_property IOSTANDARD LVCMOS33 [get_ports Vsync]


##USB-RS232 Interface
#set_property PACKAGE_PIN B18 [get_ports RsRx]
	#set_property IOSTANDARD LVCMOS33 [get_ports RsRx]
#set_property PACKAGE_PIN A18 [get_ports RsTx]
	#set_property IOSTANDARD LVCMOS33 [get_ports RsTx]


##USB HID (PS/2)
#set_property PACKAGE_PIN C17 [get_ports PS2Clk]
	#set_property IOSTANDARD LVCMOS33 [get_ports PS2Clk]
	#set_property PULLUP true [get_ports PS2Clk]
#set_property PACKAGE_PIN B17 [get_ports PS2Data]
	#set_property IOSTANDARD LVCMOS33 [get_ports PS2Data]
	#set_property PULLUP true [get_ports PS2Data]


##Quad SPI Flash
##Note that CCLK_0 cannot be placed in 7 series devices. You can access it using the
##STARTUPE2 primitive.
#set_property PACKAGE_PIN D18 [get_ports {QspiDB[0]}]
	#set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[0]}]
#set_property PACKAGE_PIN D19 [get_ports {QspiDB[1]}]
	#set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[1]}]
#set_property PACKAGE_PIN G18 [get_ports {QspiDB[2]}]
	#set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[2]}]
#set_property PACKAGE_PIN F18 [get_ports {QspiDB[3]}]
	#set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[3]}]
#set_property PACKAGE_PIN K19 [get_ports QspiCSn]
	#set_property IOSTANDARD LVCMOS33 [get_ports QspiCSn]
```

Unused Code

```
`timescale 1ns / 1ps
/* File: UART_Test.sv
 * Init: 11/17/2018 03:34:53 AM
 * Author: Alex Goldstein
 * Desc: Test for UART_Trans & UART_Rec on a single board.
 *      Can be configured to send via wired JA ports, or using interconnects.
 *           --> To do this, comment out the JA ports, and replace their port mappings with
the wires t_bs, t_sig
                Also, you will need to change propDelay=1 for board, =2 for sim. (Not sure
why.)
 */
module UART_Test(
    input clk,
    input [15:0] dataIn, //value to send
    input sendBtn, // Push this button when ready to send.
                 // Sending repeats until btn is released, so make sure to release it!
    output [15:0] dataOut, //value received

    input JAbsI, JAsigI, // JA pin inputs (bitstream, receiving signal)
    output JAbsO, JAsigO // JA pin outputs (bitstream, sending signal)
    );
logic t_bs, t_sig; //used for internal wiring, in place of JA pins (eg. for simulation, JA pins
cannot be used)

/* The parameters for the modules are a predetermined agreement.
   #([packetSize, cycleDiv, propDelayOffset]), where
       packetSize = # bits sent/received per bitstream (these must agree)
       cycleDiv = clock division for sclk. Period of sclk = (1/100)clk (these must agree)
       propDelayOffset = Rec has delay from the time it receives the "sending" signal, to
the time it can begin accepting bits from bitstream.
              Only a parameter for Trans.
              Arguments:
                        for basys3: delay=1
                        for sim:    delay=2
*/

// Don't forget to change propDelay!
UART_Trans #(16,100,1) trans(.clk, .data(dataIn), .sendBtn, .bsOut(JAbsO),
.sendSig(JAsigO));
UART_Rec #(16,100) rec(.clk, .bsIn(JAbsI), .recSig(JAsigI), .data(dataOut));
endmodule
```

```
`timescale 1ns / 1ps

/* File: ClockDiv.sv
 * Init: 11/16/2018 04:54:54 PM
 * Author: Alex Goldstein
 * Description: Simple clock divider.
 */

// CYCLES: # of clk cycles to equal one sclk cycle
module ClockDiv #(parameter CYCLES=1)(
   input clk,
   output logic sclk = 1 //doesn't seem to matter if this is 1 or 0 initially, but necessary for
sim
   );

   logic [29:0] count = 0;

   always_ff @ (posedge clk)
   begin
      count = count + 1;
      if (count == (CYCLES)/2) //Flip state at 1/2 period == 1/2 CYCLES
      begin
         count = 0;
         sclk = ~sclk;
      end

   end

endmodule
```

```
`timescale 1ns / 1ps

/* File: UART_Trans.sv
 * Init: 11/15/2018 09:46:48 AM
 * Author: Alex Goldstein
 * Description: Transmitter for UART system. Converts input data to bitstream, outputs the
bitstream to Receiver.
 */

// packetSize: @ bits/bitstream
// cycleDiv: # clk cycles / sclk cycle
// propDelayOffset: The UART_Rec has a delay between the time it receives the "sending"
signal and when it can first accept bits from the bitstream. This delay causes Trans to wait # sclk
cycles.
module UART_Trans #(parameter packetSize=16, cycleDiv=100, propDelayOffset=0)(
   input [packetSize-1:0] data,
   input clk, sendBtn,
   output .bsOut(dOut[0]), // maps .dOut[0](bsOut) <-- except that the syntax is incorrect. Indices
only allowed inside (), so the mapping is done here instead of in the ShiftReg module
instantiation.
   output sendSig //brief high signal that alerts UART_Rec to begin accepting bitstream data
   );

//Interconnects
logic t_sclk;
logic [1:0] t_regShift; //0=none, 1=right, 2=left
logic t_regLD; //1: LD-enable ShiftReg

//Modules
ClockDiv #(cycleDiv) uartClk(.clk(clk), .sclk(t_sclk));

logic [packetSize-1:0] dOut; //for mapping ShiftReg(.dOut(dOut))
ShiftReg #(packetSize) shift_reg(.clk(t_sclk), .dIn(data), .LD(t_regLD), .shift(t_regShift), .dOut );

TransFSM #(packetSize, propDelayOffset) transFSM(.clk(t_sclk), .ifSend(sendBtn),
.regShift(t_regShift), .regLD(t_regLD), .sendSig(sendSig) );

endmodule
```

```
`timescale 1ns / 1ps

/* File: ShiftReg.sv
 * Init: 11/16/2018 06:23:30 PM
 * Author: Alex Goldstein
 * Description: Register for UART_Trans. Takes in register dIn, shifts right on each clk.
 */

//packetSize: # bits/bitstream
module ShiftReg #(parameter packetSize=4)(
    input clk,
    input LD, //load-enable
    input [1:0] shift, //0=hold, 1=right shift, 2=left shift
    input [packetSize-1:0] dIn,
    output logic [packetSize-1:0] dOut = 0
    );

always_ff @(posedge clk)
    begin
    //Load data (reg)
    if(LD) dOut = dIn;

    //Bit shift
    if(shift==1) dOut = {1'b0, dOut[packetSize-1:1]};
    else if(shift==2) dOut = {dOut[packetSize-2:0], 1'b0};
    else dOut = dOut;

    end
endmodule
```

```
`timescale 1ns / 1ps

/* File: TransFSM.sv
 * Init: 11/16/2018 04:42:38 PM
 * Author: Alex Goldstein
 * Description: FSM for UART_Trans. Holds register value unless button input (ifSend) is
received, then begins bitshift and sends transmission signal to Receiver.
 */

module TransFSM #(parameter packetSize=4, propDelayOffset=0)(
   input clk,
   input ifSend, //if send button is pressed
   output logic regLD=0,  //start state doesn't let register load
   output logic sendSig=0,
   output logic [1:0] regShift=1 //shift right (initially, this makes sure we are clearing the
register if there isn't anything in it)
   );

// States
localparam [3:0] rest=0, sendFirst=1, sendRemaining=2;

logic [3:0] PS = rest;
logic [3:0] NS = rest;

// FSM Logic
logic [15:0] bitsSent = 0;
logic [5:0] propDelayCounter = 0; //Due to propogation delays, we need to delay sending
the bitstream by a clock cycle or two. Use this to adjust.
always_ff @ (posedge clk)
   begin
   PS = NS; //advance to next state on clock edge
   if(bitsSent == packetSize) PS=rest; //unless we've sent all the bits in a packet; in this
case, return to rest

   case(PS)
      rest:
         begin
         regShift = 1; regLD = 0; //don't load the register, continually shift (to clear) the
register
         sendSig = 0;
         bitsSent = 0; //reset bitsSent from last run
         propDelayCounter = 0; //reset propDelayCounter from last run
         if(ifSend) NS = sendFirst; //go to sendFirst when btn is pressed
```

```
            else NS = rest;
            end
        sendFirst: //send first bit by loading it data into register
            begin
            regLD = 1; regShift = 0;
            sendSig = 1; //notify receiver to receive bitstream
            NS=sendRemaining;
            bitsSent = 1; //we've sent the first bit
            end
        sendRemaining:
            begin
            if(propDelayCounter < propDelayOffset) propDelayCounter++; //don't send the
second bit until propDelayOffset sclk cycles have passed. This only applies to hold the
first bit; the remaining bits don't get delayed.
            else
                begin
                regLD = 0; regShift = 1; //Don't let the reg load anymore, just shift on each sclk
edge
                sendSig = 0; //sendSig is brief, it only needs to be HIGH to notify for the
beginning.
                bitsSent = bitsSent+1; //if all packets are sent, the if(bitsSent==packetSize) will
kick us back to the rest state.
                NS = sendRemaining; //otherwise keep looping; we have more bits to send
                end
            end
        endcase
    end
endmodule
```

```
`timescale 1ns / 1ps

/* File: RecFSM.sv
 * Init: 11/16/2018 09:06:46 PM
 * Author: Alex Goldstein
 * Description: Triggers BitStreamReg shiftReg to begin/end accepting bits from bitstream.
 *     1. Receives ifRecSig, indicates Trans is going to send 3 sclk cycles later
 *     2. Tells shiftReg to accept bits for [packetSize] sclk cycles
 *     3. Rest state causes shiftReg to stop, hold value.
 */

// packetSize: # bits per bitstream
module RecFSM #(parameter packetSize=4)(
    input clk, // clk receives from sclk
    input ifRecSig,
    output logic [1:0] regShift, //controls for shiftReg
    output logic regLD
    );

// Set states
localparam [3:0] rest=0, receiving=1;
logic [3:0] PS = rest, NS = rest;

// FSM Logic
logic [15:0] bitsReceived = 0;
always_ff @(posedge clk)
    begin
    PS = NS; //advance to next state at each clock edge
    //except we need to return to rest when all bits in a packet have been received
    if(bitsReceived == packetSize)
        begin
        PS=rest;
        bitsReceived=0;
        end

    case(PS)
        rest: // do nothing. Reg holds value.
            begin
            regShift=0;
            regLD=0;
            if(ifRecSig) NS=receiving; //when we receive the signal from Trans
            else NS=rest;
            end
        receiving: //This state occurs when ifRecSig==1, even when ifRecSig goes back to zero.
            begin
            regShift=1; //start shifting, loading the register (reg shifts, then loads, so we can turn
both off simultaneously, and it will not shift the last bit
            regLD=1;
            bitsReceived += 1;
            NS=receiving; // Stay in this state until bitsReceived==packetSize.
            end
        endcase

    end

endmodule
```

```
`timescale 1ns / 1ps
/* File: BitStreamReg.sv
 * Init: 11/16/2018 06:23:30 PM
 * Author: Alex Goldstein
 * Desc: Receiver module for UART, receives a bitstream and assembles it as a register value.
 *       Works in conjunction with UART_Trans module.
 */

// packetSize: # bits / bitstream
// cyclesDiv: clock divider. sclk = (1/100)clk. While this doesn't use a clock divider, it needs to
know what division Trans uses, so it can calculate the midpoint of each state.
module BitStreamReg #(parameter packetSize=4, cycleDiv=100)(
   input clk,
   input LD, msbLD, //LD=load enable. msbLD = load incoming LSB into MSB slot (because
Trans sends LSB first, and this right-shifts on each sclk)
   input [1:0] shift, //0=none, 1=right shift, 2=left shift
   input [packetSize-1:0] dIn,
   output logic [packetSize-1:0] dOut = 0,
   output logic HERE //dummy output, spikes show when reg is taking samples from the input
   );

logic [29:0] cycleCounter = 0; //tracks board clk, resets to 0 @ cycleDiv
always_ff @(posedge clk)
   begin
   cycleCounter++;
   HERE=0;
   if(cycleCounter == 20) //at the halfway point
      begin
      HERE = 1;

      //Bit shift
      if(shift==1) dOut = {1'b0, dOut[packetSize-1:1]};
      else if(shift==2) dOut = {dOut[packetSize-2:0], 1'b0};
      else dOut = dOut;

      //Load newest bit from bitstream
      if(LD && msbLD) dOut[packetSize-1] = dIn[0];
      else if(LD && ~msbLD) dOut[0] = dIn[0];
      end
   if(cycleCounter == cycleDiv) cycleCounter=0;
   end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO CPE 133
// Engineer: Darian Primer
//
// Create Date: 11/20/2018 01:22:30 AM
// Design Name: Checking that there is only a one bit defference between the two inputs.

//////////////////////////////////////////////////////////////////////////////////


module InputChecker(
    input [15:0] A, B,
    input clk,
    output OK
    );

    logic t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16;
    logic
nott1,nott2,nott3,nott4,nott5,nott6,nott7,nott8,nott9,nott10,nott11,nott12,nott13,nott14,nott15,nott
16;
    assign nott1 = ~t1;
    assign nott2 = ~t2;
    assign nott3 = ~t3;
    assign nott4 = ~t4;
    assign nott5 = ~t5;
    assign nott6 = ~t6;
    assign nott7 = ~t7;
    assign nott8 = ~t8;
    assign nott9 = ~t9;
    assign nott10 = ~t10;
    assign nott11 = ~t11;
    assign nott12 = ~t12;
    assign nott13 = ~t13;
    assign nott14 = ~t14;
    assign nott15 = ~t15;
    assign nott16 = ~t16;

    logic [3:0] t17, t18, t19, t20, t21, t22, t23, t24, t25, t26, t27, t28, t29,
              t30, t31, t32, t33;

    Comparator #1 comp1(.A(A[0]), .B(B[0]), .EQ(t1), .LT(), .GT());
    Comparator #1 comp2(.A(A[1]), .B(B[1]), .EQ(t2), .LT(), .GT());
    Comparator #1 comp3(.A(A[2]), .B(B[2]), .EQ(t3), .LT(), .GT());
    Comparator #1 comp4(.A(A[3]), .B(B[3]), .EQ(t4), .LT(), .GT());
    Comparator #1 comp5(.A(A[4]), .B(B[4]), .EQ(t5), .LT(), .GT());
    Comparator #1 comp6(.A(A[5]), .B(B[5]), .EQ(t6), .LT(), .GT());
    Comparator #1 comp7(.A(A[6]), .B(B[6]), .EQ(t7), .LT(), .GT());
    Comparator #1 comp8(.A(A[7]), .B(B[7]), .EQ(t8), .LT(), .GT());
    Comparator #1 comp9(.A(A[8]), .B(B[8]), .EQ(t9), .LT(), .GT());
    Comparator #1 comp10(.A(A[9]), .B(B[9]), .EQ(t10), .LT(), .GT());
    Comparator #1 comp11(.A(A[10]), .B(B[10]), .EQ(t11), .LT(), .GT());
    Comparator #1 comp12(.A(A[11]), .B(B[11]), .EQ(t12), .LT(), .GT());
    Comparator #1 comp13(.A(A[12]), .B(B[12]), .EQ(t13), .LT(), .GT());
    Comparator #1 comp14(.A(A[13]), .B(B[13]), .EQ(t14), .LT(), .GT());
    Comparator #1 comp15(.A(A[14]), .B(B[14]), .EQ(t15), .LT(), .GT());
    Comparator #1 comp16(.A(A[15]), .B(B[15]), .EQ(t16), .LT(), .GT());

    RippleCarryAdder RCA1(.A({3'b0,nott1}), .B({3'b0,nott2}), .Sub('0), .S(t17), .Co());
    RippleCarryAdder RCA2(.A({3'b0,nott3}), .B({3'b0,nott4}), .Sub('0), .S(t18), .Co());
    RippleCarryAdder RCA3(.A({3'b0,nott5}), .B({3'b0,nott6}), .Sub('0), .S(t19), .Co());
    RippleCarryAdder RCA4(.A({3'b0,nott7}), .B({3'b0,nott8}), .Sub('0), .S(t20), .Co());
    RippleCarryAdder RCA5(.A({3'b0,nott9}), .B({3'b0,nott10}), .Sub('0), .S(t21), .Co());
    RippleCarryAdder RCA6(.A({3'b0,nott11}), .B({3'b0,nott12}), .Sub('0), .S(t22), .Co());
    RippleCarryAdder RCA7(.A({3'b0,nott13}), .B({3'b0,nott14}), .Sub('0), .S(t23), .Co());
    RippleCarryAdder RCA8(.A({3'b0,nott15}), .B({3'b0,nott16}), .Sub('0), .S(t24), .Co());

    RippleCarryAdder RCA9(.A(t17), .B(t18), .Sub('0),  .S(t25));
    RippleCarryAdder RCA10(.A(t19), .B(t20), .Sub('0),  .S(t26));
    RippleCarryAdder RCA11(.A(t21), .B(t22), .Sub('0), .S(t27));
    RippleCarryAdder RCA12(.A(t23), .B(t24), .Sub('0), .S(t28));

    RippleCarryAdder RCA13(.A(t25), .B(t26), .Sub('0), .S(t29));
    RippleCarryAdder RCA14(.A(t27), .B(t28), .Sub('0), .S(t30));

    RippleCarryAdder RCA15(.A(t29), .B(t30), .Sub('0), .S(t31));

    Comparator #4 comp17(.A(t31), .B(4'b0001), .EQ(OK), .LT(), .GT());


    endmodule
```