

## EXPERIMENT 4

### BLOCK 4 OF THE TERM PROJECT

#### 1. Goal

Develop Block 4 of the term project by week 8 and introduce/reinforce concepts below :

- > logic design on paper,
- > schematic design on Xilinx software,
- > block-based design,
- > core-based design,
- > how to simulate and verify logic circuits,
- > the Digilent XLA FPGA board, and
- > how to test circuits on FPGA boards.

#### 2. Summary

We will develop Block 4 of the term project. Currently, Block 4 is partially core and also a small portion of it, macro M1, has been developed in Experiment 3. The remaining non-core circuits, macros M2, M3, M4, and M5 are the target of this experiment. Block 5 and Block 6 macros, M6, M7, M8, and M9 will be kept as macros in this experiment and worked on later. The term project will be described in a handout titled *Term Project*. The term project will be completed with Experiment 6. Experiment 6 is the final design, its outcome is Ppm.

Block 4 is responsible for a) keeping track of position displays, b) determining addition/subtraction results between position displays and the random digit, and c) determining which position displays are the same as the played digit. Block 4 circuits are shared by both players, meaning the Human Play Block and the Machine Play Block send values to Block 4 and use outputs of Block 4. By sharing Block 4, the amount of hardware is reduced. We will call the overall circuit of Block 4 **Play Check** since, play checking starts with this block. The subblock that keeps track of position displays is given to students as a core.

Overall, students will design macros, M2, M3, M4, and M5 in Experiment 4. Students will be allowed to revise their Block 4 design in Experiment 5 and Experiment 6 as they will learn more advanced circuits. Below, we will first describe the development of the Play Check (Block 4) circuits on paper : the black-box view, input-output relationship and initial partitionings. Then, the remaining work will be done by students themselves. Nevertheless, we will advise students in which order to develop Block 4.

#### 3. Development Cycle on Paper/Computers

Follow the first two major digital product development cycles to develop a *new chip* given below. These two development cycles are the **development cycle on computers**, and the **development cycle on breadboards with FPGA chips**. These two and the last development cycle are described in detail in the handout titled *Digital Product Development*. Also, the *Development Cycle on Breadboards with FPGA Chips* handout describes how to do this cycle on the Xilinx Foundation software package. The development cycle on computers consists of three steps :

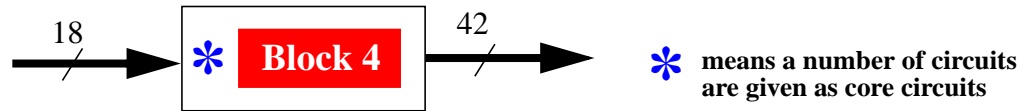
- > logic design
  - > input/output relationship
  - > implementation (the schematic design)
- > test
- > modify

Logic design consists of obtaining the precise input-output relationship and implementation. The test consists of the simulation of the circuit on computers and modification is the simple change of the design based on test results. If the circuit is simple one can completely design it on paper and then place the components on the computer screen. For Experiment 4, this is not the case. So, we obtain the black-box view and initial block partitioning of the block on paper and then move the design to the computer to continue with additional partitionings and component determination.

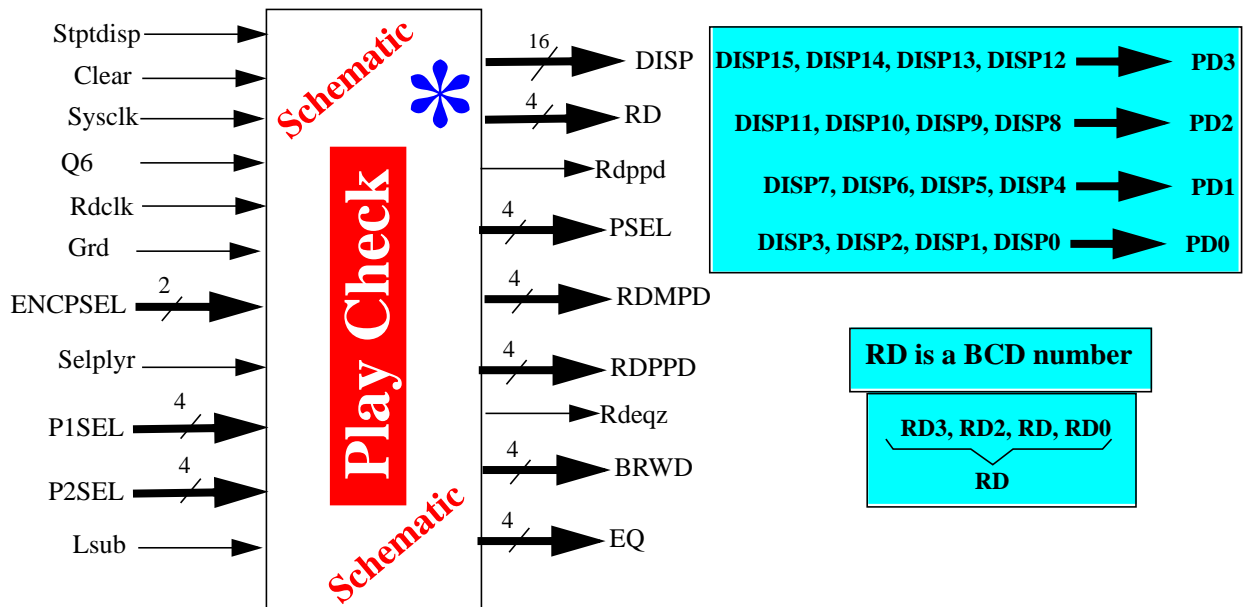
### 3.1. Logic Design

#### 3.1.1. Input-Output Relationship

The input/output relationship means the digital circuit is viewed as a black box where only inputs and outputs are considered. Then, the outputs are related to the inputs. Block 4 has 18 inputs and 42 outputs. The black-box view and the detailed view of the inputs and outputs of Block 4 are shown in Figure 1 and Figure 2, respectively. The description of the inputs and outputs are given on Tables 1 and 2, respectively.



**Figure 1.** The black-box view of the Experiment 4 block, Block 4. It is partially core. Also, a portion of Block 4 has been designed in Experiment 3.



**Figure 2.** The detailed view of the input and output signals of the Play Check Block.

| Signal name | Number of lines | Description   |
|-------------|-----------------|---|
| Stptdisp    | 1               | Store the played digit on a position display. When it is 1, the played digit is stored on its register. From the Control Unit                           |
| Clear       | 1               | Clear registers, counters and flip-flops. It is used to store zeros on registers, counters and FFs during reset. From the Control Unit                  |
| Sysclk      | 1               | The system clock at 6 Hz. It is used to synchronize operations on sequential circuits. From the Input/Output Block                                      |
| Q6          | 1               | A clock signal at 3 Hz. It is used in the clear circuit of the register that keeps the random digit for the current player. From the Input/Output Block |
| Rdclk       | 1               | The random digit clock at 768Hz. It is used by the counter whose output is stored on a register as the random digit. From the Input/Output Block        |

**Table 1: Inputs of the Play Check Block.**

| Signal name | Number of lines | Description  |
|-------------|-----------------|--|
| Grd         | 1               | Get the random digit. When it is one, the register is stored the outputs of the counter as the random digit. From the Control Unit.  |
| ENCPSEL     | 2               | Encoded position select. The two bits indicate the played position in Unsigned Binary. It is used by both players. From the Points Calculation Block   |
| Selplyr     | 1               | Select players for the current mode. When it is 1 it indicates Player 2 is the current player which means the system is in one of states 4, 5 and 6. Otherwise, it means Player 1 is the current player and the system is in one of states 1, 2 and 3. From the Control Unit |
| P1SEL       | 4               | Player 1 position select signals. One bit per position indicates a position selected by the human player when its bit is 1. Each bit is connected to a switch on the FPGA board. From the Input/Output Block   |
| P2SEL       | 4               | Player 2 position select signals. One bit per position indicates a position selected by the machine player when its bit is 1. From the Machine Play Block  |
| Lsub        | 1               | Latched subtract signal. When it is 1, it means the current player has decided to subtract the random digit from a position display. It cannot be 1 if the result of the subtraction is negative. From the Control Unit  |

**Table 1: Inputs of the Play Check Block.**

| Signal name | Number of lines | Description   |
|-------------|-----------------|---|
| DISP        | 16              | Position display bits. The four position display digits in Unsigned Binary. To the Input/Output Block   |
| RD          | 4               | The current random digit. In states 1, 2, and 3, it is the human player random digit. In states 4, 5 and 6, it is the machine player random digit. To the Input/Output Block  |
| Rdppd       | 1               | Position display plus the random digit. When it is 1, it indicates there is a display overflow. That is, the sum is greater than $(15)_{10}$ . To the Input/Output Block and the Points Calculation Block   |
| PSEL        | 4               | Player position select. Each bit selects a position to play when it is 1. It is used by both players. In states 1, 2 and 3, they are equal to Player 1 position selection signals, P1SEL. In states 4, 5 and 6, they are equal to Player 2 position selection signal, P2SEL. To the Input/Output Block and the Points Calculation Block |
| RDMPD       | 4               | Position display minus the random digit. Each bit is for one position and when it is 1 it means the result of the subtraction is negative for that position. To the Human Play Block and the Machine Play Block   |
| RDPPD       | 4               | Position display plus the random digit. Each bit is for one position and when it is 1 it means the result of the addition exceeds $(15)_{10}$ for that position. That is, if a bit is 1, there is a display overflow. To the Human Play Block and the Machine Play Block  |
| Rdeqz       | 1               | Random digit is equal to zero. When it is 1, it means RD is equal 0. To the Machine Play Block  |
| BRWD        | 4               | Basic reward. It is the digit played on a position. It is also the minimum points a player earns if there is no adjacency and no subtraction. To the Points Calculation Block   |
| EQ          | 4               | The basic reward digit is equal to the position display digit. Each bit is for a position. When a bit is 1, it indicates, the corresponding position is equal to the basic reward digit. It is used by both players. To the Point Calculation Block   |

**Table 2: Outputs of the Play Check Block.**

Experiment 4 completes the development of Block 4, the Play Check Block. A portion of the block has already been developed in Experiment 3 : the ADDPD3RD circuit. In addition, Block 4 has core circuits provided with students. Thus, the amount of development in this Experiment is **not** large. Students are suggested that they refresh their memory on the playing rules and input/output devices of the Ppm system by reviewing the handout titled *Playing Rules of*

*the Ppm Game.* Note that the circuits of Experiment 3 will be used immediately in Experiment 4. Nevertheless, students can revise their implementation by changing the logic and/or using new gates and new Xilinx Design Blocks in this experiment and Experiments 5 and 6.

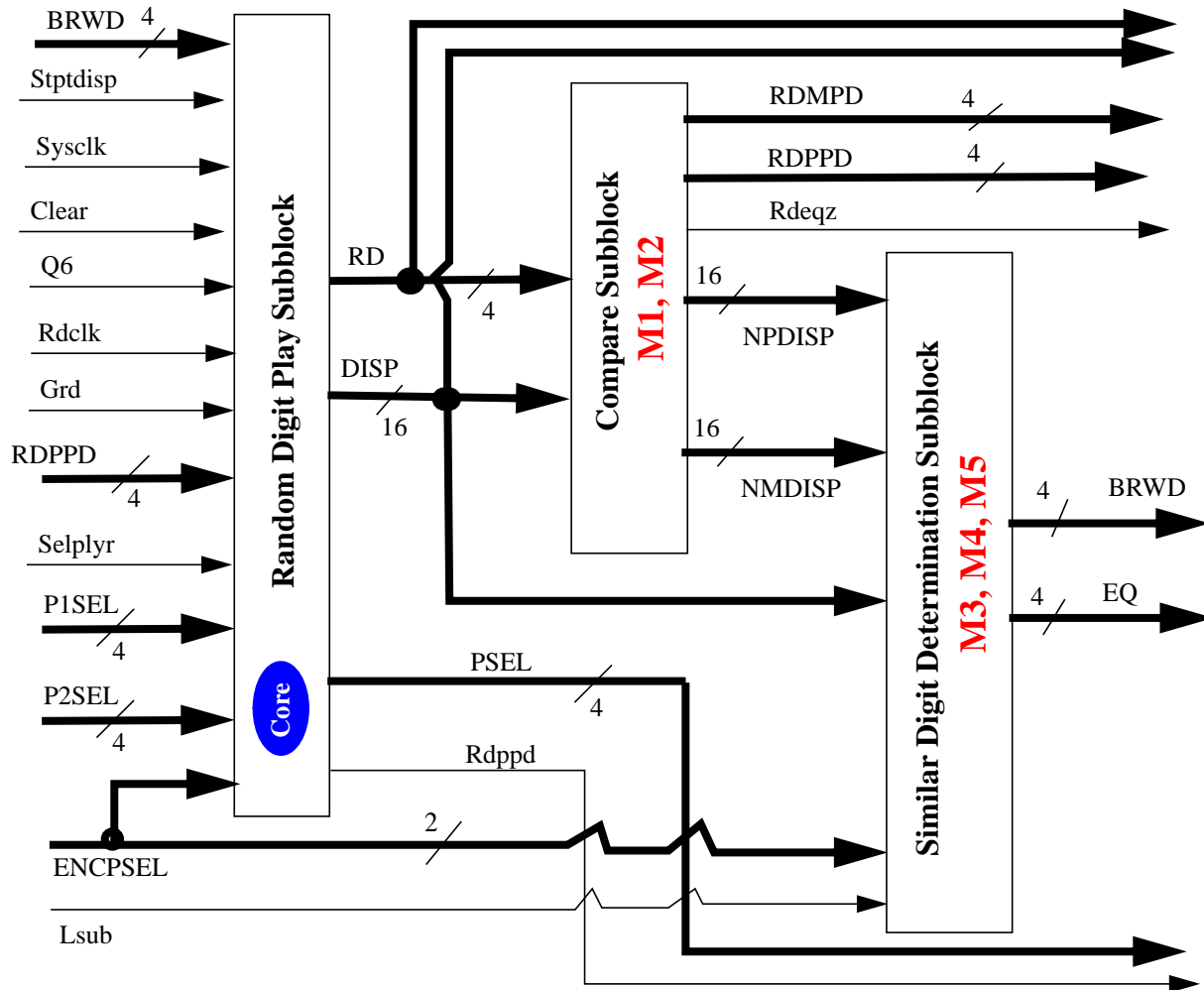
Block 4 is responsible for the position displays and operations on them. Both players share this block. The specific functions of Block 4 are to keep track of position displays, play possibilities and determining similar digits next to the digit played. Since there are 18 inputs to Block 4, we cannot obtain a detailed input output relationship, the truth table. Therefore, we have to divide the circuit into simpler blocks. We already know that a circuit is partitioned into blocks based on major operations (specified on the operation table), design goals and available components. Major operations on the operation table indicate the blocks that perform them. The available components indicate how many components to use for a block (how easy a block will be implemented).

We want the operation table to *explicitly* show what the major operations are ! What are the major operations on the table ? Can one can be sure to say “k” major operations in the block, by just looking at the table ? It turns out two experienced people can have two different tables, therefore two different sets of major operations ! Also, unless a person has the experience of estimating how much logic (the number of components) these major operations need, one cannot be sure about the number of major operations. Thus, determining major operations on the operation table is hard. Even experience sometimes cannot help in which case one would first partition the block based on intuition, then start designing blocks and then revise block partitionings until the whole circuit is completed. Table 3 shows the operation table of Block 4.

| Operation  |
|--|
| Store BRWD on one of the four position displays, by using PSEL<br>> 16 DISP signals show the new set of displays<br>Generate a new random digit if Grd indicates so<br>> 4 RD signals show the new random digit<br>Select one from several<br>> Rdppd selected from four RDPPD signals<br>> PSEL selected from P1SEL and P2SEL   |
| Compute $(PD_k + RD)$ to generate intermediate signals :<br>> 16 NPDISP signals as the result of four additions<br>> four RDPPD signals, indicating if $PD_k + RD$ is greater than $(15)_{10}$<br>Compute $(PD_k - RD)$ to generate intermediate signals :<br>> 16 NMDISP signals as the result of four additions<br>> four RDMPD signals, indicating if $PD_k - RD$ is negative<br>RDEQZ signal, indicating if $RD = 0$   |
| Determine the new set of display bits based on Lsub :<br>> 16 NDISP signals are generated from NPDISP signals if an addition is performed<br>> 16 NDISP signals are generated from NMDISP signals if a subtraction is performed<br>Determine the digit to be stored on a position display from NDISP, by using ENCPSEL :<br>> BRWD3, BRWD2, BRWD1 and BRWD0 indicate the digit to be played (stored).<br>Determine which position digits are identical to BRWD in case it is played on a position :<br>> EQ0, EQ1, EQ2, EQ3 outputs indicate the number of digits identical to BRWD on position displays |

**Table 3: Operation table of the Play Check Block.**

The operation table have three major rows each with similar amount of complexity. We can then have a subblock for each row of the table. Hence, three subblocks : Random Digit Play Subblock, Compare Subblock and Similar Digit Determination Subblock (Figure 3). Among the subblocks the Random Digit Subblock, is a core subblock and provided with the students. This subblock corresponds to the top major row of Table 3. Students will design the remaining portions of this block : Compare Subblock and Similar Digit Determination Subblock. Also, a portion of the Compare Subblock has been developed in Experiment 3. Finally, it will be clear later that the above three subblocks do also allow a better sharing of Block 4 by human and machine players.



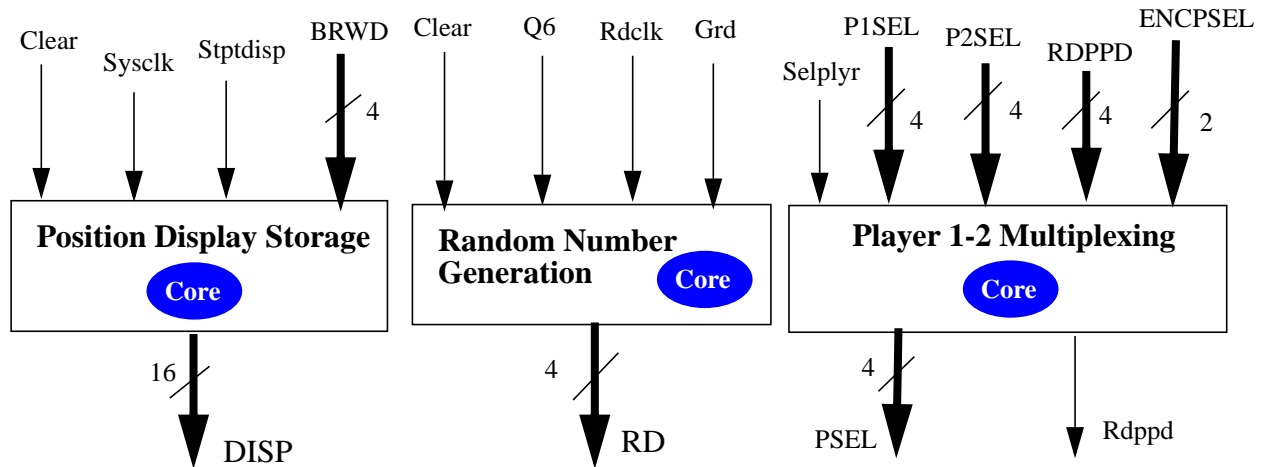
**Figure 3. The Play Check Block partitioning.**

Next is the input/output relationship of the three subblocks. Each subblock has many inputs so we do need to partition them, again based major operations, design goals and available components. The operation table of the Random Digit Play Subblock is shown on Table 4. This subblock is a core subblock. Students will not change any circuit in it. The three subsubblocks of this subblock are shown in Figure 4. Given the three rows on the table, one concludes that the subblock a) keeps (stores) the four position displays, and b) generates a new random digit and c) multiplexes between (selects one of several) values. Therefore, this subblock consists of three subsubblocks : Position Displays Storage Subsubblock, Random Number Generation Subsubblock and the Player 1-2 Multiplexing Subsubblock.

| Operation  |
|--|
| If Stptdisp is active, store BRWD on a display pointed by PSEL. If Clear is active, zero the displays. |
| If Grd is active, get a new random digit   |
| By using Selplyr, select PSEL from P1SEL and P2SEL. By using ENCPSEL generate Rdppd from RDPPD         |

**Table 4: Operation table of the Random Digit Play Subblock.**

The Position Displays Storage Subsubblock core circuit has four 4-bit registers to keep the four position display digits. The registers are Xilinx FD4CE components. We know that a register is a sequential circuit and used as a temporary storage circuit. A position display register is stored a digit when the Stptdisp signal is 1. This happens in state 2



**Figure 4. The Random Digit Play Subblock partitioning.**

for Player 1 and in state 5 for Player 2. The AND of the selected position signal (PSEL<sub>x</sub>) and Stptdisp enable the clock input of the position register played. The Sysclk from Block 2 is used as the clock signal (the positive edge) for the registers. All four registers are cleared (stored 0) **asynchronously** by the Clear signal. Clear is active when the FPGA is downloaded (state 0), the Ppm is reset (state 0).

The Random Number Generation Subsubblock has a freely running counter at 768Hz. The counter is the Xilinx CD4CE. The counter output is latched to (stored on) a register, a Xilinx FD4CE, when the Grd input is active. The register is cleared when the FPGA chip is downloaded or when there is a reset. The Clear signal is used to clear the register. But, the Clear is not directly applied to the register in order to make sure that it does not interfere with the storage of the random digit on the register. A back-to-back combination of two flip-flops is used to generate a signal from the Clear signal. The Clear is 1 in state 0 and 0 when Player 1 turns on P1play to play. The FF combination signal is a pulse whose duration is shorter than the Clear signal and determined by the Q6 signal which is at 3Hz. Thus the register is applied a clear signal for short duration and is ready to accept the first random digit when the human player turns on P1play.

The Player 1-2 Multiplexing Subsubblock multiplexes two sets of values : a) selects PSEL from either Player 1 position select signals (P1SEL) and Player 2 position select signals (P2SEL) and b) selects Rdppd from four RDPPD signals. For the first selection it uses the Selplyr as the select signal to select P1SEL when Selplyr is 0 since Player 1 is the current player. Otherwise, P2SEL is selected and output as PSEL. A 4-bit 2-to-1 Xilinx MUX, X74\_157, is used to implement circuit. To generate the Rdppd signal a 4-to-1 Xilinx MUX, M4\_1E, is used. The MUX is input four RDPPD signals and the output is equal to the RDPPD signal that is for the played position. The position number indicated by the ENCPSEL signals.

The Compare Subblock computes new display values. The subblock is implemented by macros M1, and M2 which perform three sets of compares. One is comparing RD with zero, outputting the Rdeqz signal. The second is comparing  $PD_k + RD$  with  $(15)_{10}$ , outputting four compare result values, RDPP3, RDPP2, RDPP1 and RDPP0. The subsubblock also outputs four  $PD_k + RD$  values on 16 wires named NPDISP. The last one is comparing  $PD_k - RD$  with 0, outputting four compare results, RDMP3, RDMP2, RDMP1, and RDMP0. In addition 16 NMDISP signals are output to represent four  $PD_k - RD$  values.

Since the subblock has 20 inputs, we need to get its operation table to implement it. First, our intuition tells us that the operation table of the Compare Subblock must be a super set of the operation table of Experiment 3. This is because the circuit of Experiment 3 will be used in the Compare Subblock directly. Second, the operation table of the Compare Subblock is shown on Table 5. Third, we know from Experiment 3 that the subtractions are implemented as additions. Thus, we would obtain a new operation table, the final one shown on Table 6. Accordingly, there are 10 subsubblocks, one for each row as shown in Figure 5. A portion of the Compare Subblock, M1, is designed in Experiment 3 as the ADDPD3RD circuit. The ADDPD3RD circuit generates Rdeqz, NPDISP15-NPDISP12, RDPPD3, NMDISP15-NMDISP12, and RDMPD3. Thus, M1 implements three subsubblocks (rows) of the Compare Subblock (operation table).

| Operation   |
|---|
| Add PD3 and RD to generate NPDISP15-NPDISP12 and then compare => RDPPD3 = 1 if $PD3 + RD > (15)_{10}$ , |
| Add PD2 and RD to generate NPDISP11-NPDISP8 and then compare => RDPPD2 = 1 if $PD2 + RD > (15)_{10}$ ,  |
| Add PD1 and RD to generate NPDISP7-NPDISP4 and then compare => RDPPD1 = 1 if $PD1 + RD > (15)_{10}$ ,   |
| Add PD0 and RD to generate NPDISP3-NPDISP0 and then compare => RDPPD0 = 1 if $PD0 + RD > (15)_{10}$ ,   |
| Subtract RD from PD3 to generate NMDISP15-NMDISP12 and then compare => RDMPD3 = 1 if $PD3 - RD < 0$     |
| Subtract RD from PD2 to generate NMDISP11-NMDISP8 and then compare => RDMPD2 = 1 if $PD2 - RD < 0$      |
| Subtract RD from PD1 to generate NMDISP7-NMDISP4 and then compare => RDMPD1 = 1 if $PD1 - RD < 0$       |
| Subtract RD from PD0 to generate NMDISP3-NMDISP0 and then compare => RDMPD0 = 1 if $PD0 - RD < 0$       |
| Compare RD with 0 => RDEQZ = 1 if $RD = 0$  |

**Table 5: Operation table of the Compare Subblock**

| Operation   |
|---|
| Add PD3 and RD to generate NPDISP15-NPDISP12 and then compare => RDPPD3 = 0 if $PD3 + RD > (15)_{10}$ , |
| Add PD2 and RD to generate NPDISP11-NPDISP8 and then compare => RDPPD2 = 0 if $PD2 + RD > (15)_{10}$ ,  |
| Add PD1 and RD to generate NPDISP7-NPDISP4 and then compare => RDPPD1 = 0 if $PD1 + RD > (15)_{10}$ ,   |
| Add PD0 and RD to generate NPDISP3-NPDISP0 and then compare => RDPPD0 = 0 if $PD0 + RD > (15)_{10}$ ,   |
| Add CRD to PD3 to generate NMDISP15-NMDISP12 and then compare => RDMPD3 = 1 if $PD3 + CRD + 1 < 0$      |
| Add CRD to PD2 to generate NMDISP11-NMDISP8 and then compare => RDMPD2 = 1 if $PD2 + CRD + 1 < 0$       |
| Add CRD to PD1 to generate NMDISP7-NMDISP4 and then compare => RDMPD1 = 1 if $PD1 + CRD + 1 < 0$        |
| Add CRD to PD0 to generate NMDISP3-NMDISP0 and then compare => RDMPD0 = 1 if $PD0 + CRD + 1 < 0$        |
| Obtain CRD from RD by complementing it  |
| Compare RD with 0 => Rdeqz= 1 if $RD = 0$   |

**Table 6: Final operation table of the Compare Subblock**

The first four subsubblocks are identical, except input and output names ! Each is a copy of the “Add DISP & RD” circuit of Experiment 3. Similarly, the next set of four rows are identical, except input and output names. Each is a copy of the “Add DISP & CRD circuit” of Experiment 3. Thus, to implement the Compare Subsubblock, students will copy and paste the Add RD & DISP circuit of Experiment 3 and then change input and output names. They will also copy and paste the Add DISP & CRD circuit and then change input and output names. The conversion of RD to CRD is straightforward with four inverters. The last subsubblock, Compare RD with Zero is the “Compare RD with Zero” circuit of Experiment 3.

The fact that the Compare Subblock consists of identical circuits that differ in only inputs and outputs is **not** unusual. In fact, many real-life circuits have such **replicated** hardware, simplifying the design considerably. If a circuit has this property of replicated hardware, then we say the circuit is “**regular**.” We will encounter such regular circuits throughout the semester and have a discussion on this topic later. In general, the **datapath** of a digital system is highly regular, making it easier to design than the Control Unit.

The last subblock, the Similar Digit Determination Subblock, determines the digit played on a position and which position displays are the same as the digit played. In order to determine the digit played it first selects either NPDISP lines for an addition or NMDISP lines for a subtraction. Then determines the digit played, BRWD, by using the played position number, ENCPSEL. Finally, it compares BRWD with the position displays to determine the equal digits. The subblock has five sets of inputs : (i) the 16 NPDISP lines, (ii) the 16 NMDISP lines, (iii) the 16 DISP lines, (iv) the two encoded PSEL number (ENCPSEL), and (v) the Lsub line as shown in Figure 3. It has two sets of outputs : (i) the digit played (BRWD) and (ii) four lines (EQ) indicating which position is equal to BRWD. Based on this discussion, the operation table of the Similar Digit Determination Subblock is given on Table 7. The block partitioning based on Table 7 is shown in Figure 6.

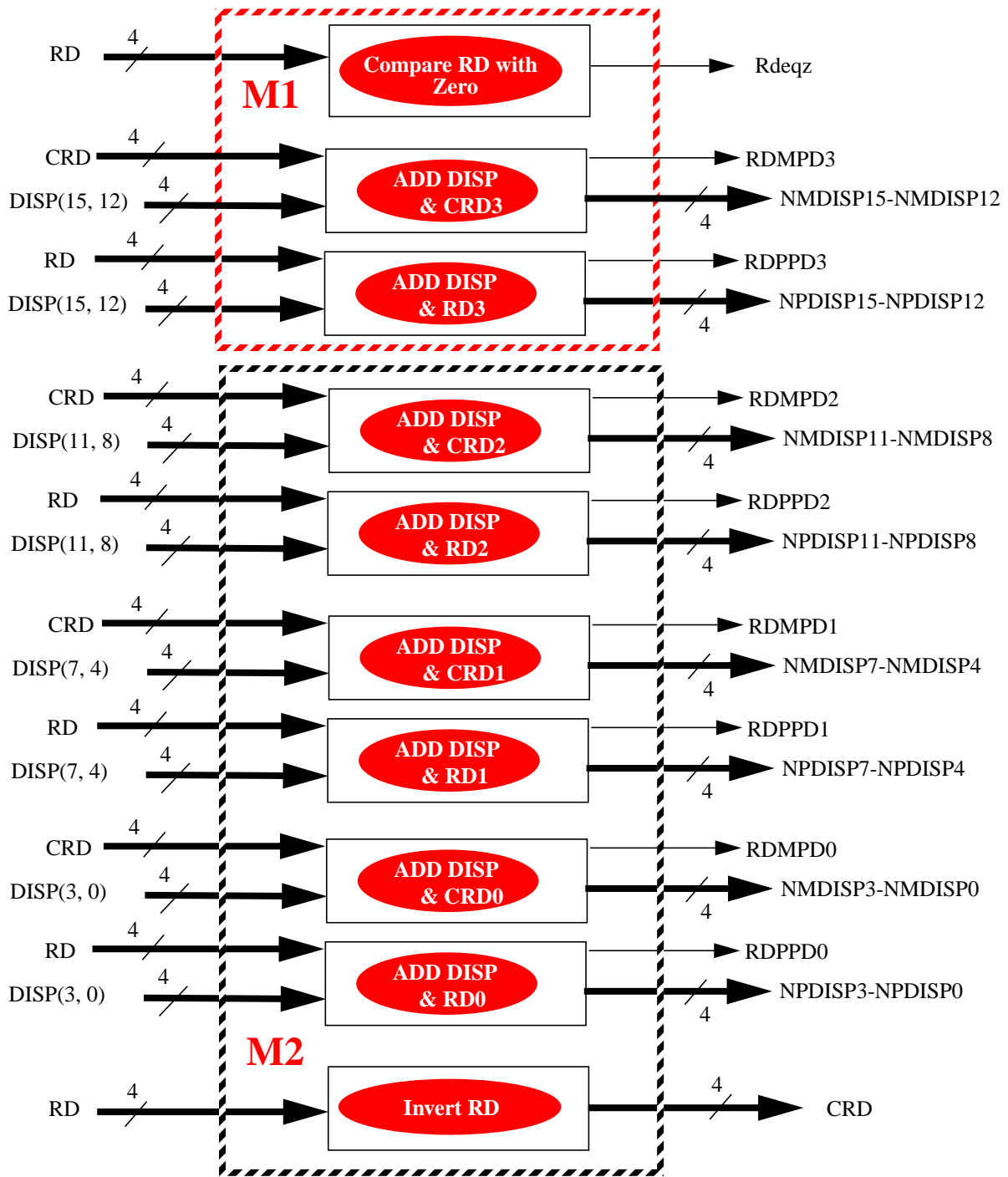
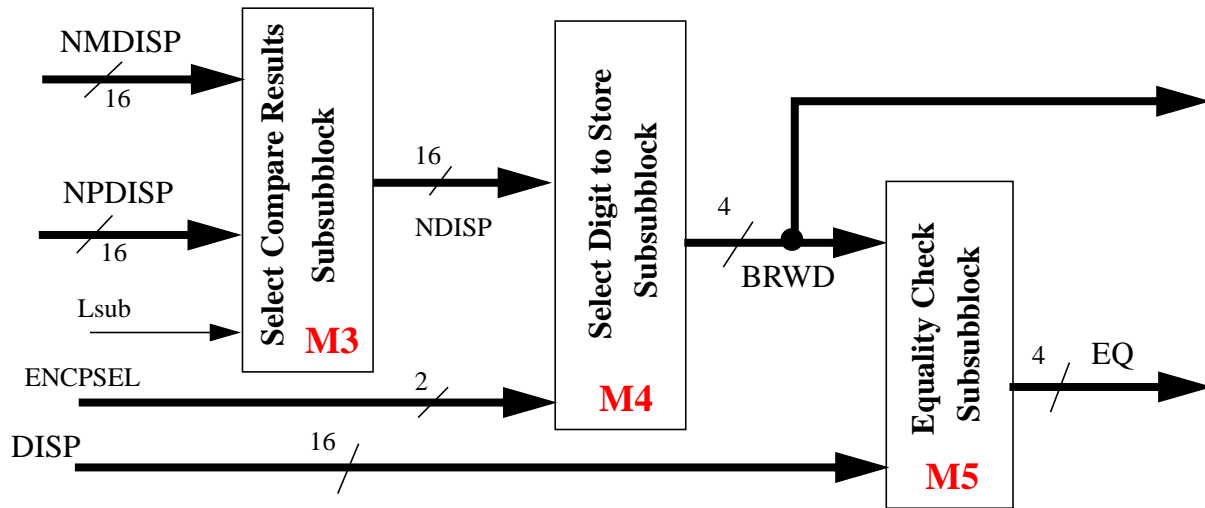


Figure 5. The Compare Subblock partitioning.

The Select Compare Results Subsubblock, macro M3, outputs new display values, NDISP, based on the Lsub input. If Lsub is 0, it means the player decided to add a display and RD, therefore NPDISP lines are selected and output as NDISP. If Lsub is 1, the player has selected to subtract RD from a display and so NMDISP lines are selected and output as NDISP. The operation table of this subsubblock is shown on Table 8. The operation table clearly indicates that this subsubblock is implemented by using a 16-bit 2-to-1 MUX. Xilinx does not have such MUX components. Its largest 2-to-1 MUX is the 4-bit 2-to-1 MUX, which is the X74\_157 component. However, one can build a 16-bit 2-to-bit MUX by using four X74\_157 components.

| Operation  |
|--|
| Select either NPDISP (if Lsub is 0) or NMDISP (if Lsub is 1) and output as NDISP                 |
| Based on ENCPSEL lines select four lines out of 16 NDISP lines as the digit to played : BRWD     |
| Determine if the BRWD digit is equal to the position display digits (DISP lines) : four EQ lines |

**Table 7: The operation table of the Similar Digit Determination Subblock.**



**Figure 6. The Similar Digit Determination Subblock partitioning.**

| Lsub | Operation      |
|------|----------------|
| 0    | NDISP = NPDISP |
| 1    | NDISP = NMDISP |

**Table 8: The operation table of the Select Compare Results Subsubblock.**

The Select Digit To Store Subsubblock, macro M4, outputs the digit played BRWD. BRWD stands for Basic Reward. It is Basic Reward since if there is no adjacency, no display overflow and no subtraction, the player earns BRWD points. The circuits in this subsubblock need multiplexers to select one out of four sets. The multiplexers, in turn, need select inputs which are two signals : ENCPSEL. The operation table of the Select Digit to Store Subsubblock is as shown on Table 9. As it is the case with the previous subsubblock, this one also uses a MUX whose size is 4-bit 4-to-1 MUX. Xilinx does not have such a MUX. Its largest 4-to-1 MUX is the X74\_153 which is a 2-bit 4-to-1 MUX. As, before one can build a 4-bit 4-to-1 MUX by using two of these X74\_153 components.

The Equality Check Subsubblock compares the played digit, BRWD, and the four position display digits to output four compare results : EQ3 - EQ0, indicating which position displays are identical to BRWD. The result is then used in Block 5 to determine the reward points. This comparison happens **before** the BRWD digit is stored, **not** after. Determining the EQ bits can be better understood with examples on Table 10.

Since BRWD and  $PD_k$  digits are 4-bit unsigned numbers and we will check only for the equality case, we need a 4-bit Unsigned Binary Comparator that checks for equality. Xilinx has two such comparators. One is the COMP4 component and the other one is the X74L\_85 component.

| ENCPSEL | Operation           |
|---------|---------------------|
| 00      | BRWD = NDISP(3-0)   |
| 01      | BRWD = NDISP(7-4)   |
| 10      | BRWD = NDISP(11-8)  |
| 11      | BRWD = NDISP(15-12) |

**Table 9: The operation Table of the Select Digit to Store Subsubblock.**

| BRWD | Positions <b>before</b> placement | Outputs   |
|------|-----------------------------------|-----------|
| 2    | 7 2 0 4                           | EQ = 0100 |
| 2    | 7 1 0 4                           | EQ = 0000 |
| 9    | 0 9 0 9                           | EQ = 0101 |
| 9    | 0 5 0 9                           | EQ = 0001 |
| 3    | 3 0 0 3                           | EQ = 1001 |
| 3    | 3 0 0 1                           | EQ = 1000 |
| 6    | 6 6 6 0                           | EQ = 1110 |
| 6    | 4 6 6 0                           | EQ = 0110 |
| 6    | 6 6 6 6                           | EQ = 1111 |

**Table 10: Equality cases for BRWD.**

We are done with the design of Block 4 on paper and now ready to move the design to the computer. That is, we completed the work on the input/output relationship and initial partitioning of the circuit. We can start its implementation which will be in line with the standard implementation cycles described in the *Digital Product Development* handout. Note again that for large circuits, we do not do a complete design on paper. Only a few initial design steps are done on paper where the black box is partitioned into smaller and smaller subblocks (**block partitioning**).

### 3.1.2. Implementation

The implementation means determining the components based on the major operations, available components and design goals : speed, cost, power consumption, reliability, size, weight, etc. For the sake of simplicity, throughout the semester, we will aim at a minimal implementation, meaning minimal number of components in the circuit.

To implement a circuit we search the Xilinx library of components of the schematic editor and see if there is any component that immediately implements the subcircuit : the component implements the textual input-output relationship/ the truth table of the circuit. The Xilinx library contains three types of components : **gates**, **flip-flops** and Xilinx Design Blocks, **XDBs**. An XDB is black box, a macro, containing a number of gates and if necessary flip-flops. An XDB is a circuit designed by Xilinx. On the screen, it is shown as a black-box with inputs and outputs. Since an XDB already implements a circuit, using it reduces the number of components and saves time. Thus, we have to try to use XDBs as much as possible. Note that here we imply that we have confidence in XDBs that they are as good as we can design and also they satisfy our product goals.

For each circuit, we first obtain the input/output relationship and then try to implement it by following a list of Implementation steps (three steps). The complete list of Implementation steps (five steps) are described later in the *Digital Product Development* handout :

i) Search the Xilinx library of components (gates, flip-flops and Xilinx Design Blocks) for one or a few XDBs that can immediately implement a subcircuit, while satisfying the product goals. These XDBs are standard blocks, therefore, they are known. That is, we immediately know their input-output relationship. If several XDBs are needed their interconnection is straightforward. There might be a few gates here and there which is fine. Their logic expressions can be very simple. If there are alternative XDBs, choose the ones that satisfy the product goals the best. After the XDBs/gates are chosen move on to the TEST step of the subcircuit implemented.

ii) If the answer is no, check to see if this subcircuit is simple enough to be implemented by a few gates while satisfying the product goals. That is, the number of inputs is less than **five**. If yes, select the right gates. After the gates are chosen move on to the TEST step of the subcircuit implemented.

iii) If all the answers are no, you will need to divide the circuit into subcircuits based on their functions, design goals and availability of components. After the partitioning, you will return to the input/output relationship step for each new subcircuit.

### ***3.1.2.1. Experiment 4 Circuit Design on the Computer :***

Follow the steps given below to develop the circuit. Note that throughout the semester, the steps to go through will be indicated by using the “bullet” symbol “>” in the beginning of a line. We will show the key to press or the mouse selection to make in bold. Also, when we say “click the mouse,” we mean clicking the **left** mouse button. The right mouse click will be explicitly mentioned.

Also, throughout the semester, the handouts are prepared by assuming that students use their **S drive** as their project storage space. If students cannot access the S drive, for example, in a place where no network connection is possible, they should use the default Xilinx project directory on their computer which is the path “...\Fndtn\active\projects.” Later, students need to move the project to the S drive when they establish the connection. Before we discuss the development of Block 4 we will give the steps to set up the Experiment 4 project on the computer.

#### ***First task before starting a new project :***

In order to keep our designs organized, for the current experiment, we will do the following :

- > By using “**My Computer**” create a new directory named “**exp4**” under cs2204.

#### ***Copying the Term Project from the termproject Folder***

We will copy the term project from the “**termproject**” folder so that we can implement the Block 4 circuit :

- > Copy the “**ppm**” folder and the “**ppm**” Xilinx pdf file from the “**termproject**” folder to the “**exp4**” folder.

#### ***Starting the Xilinx software***

- > Double click on the “**Project Manager**” icon on the screen to start the Foundation software.
- > Open the “**ppm**” project in the **exp4** directory.

In order to make sure there was no problem with the copying of the project to the exp4 folder, we will do a Xilinx Implementation and see that the Implementation Log File has six “no-load” warnings, no error messages and 80% FPGA chip utilization :

- > Do a Xilinx implementation of the exp4 project.
- > Starting with the “**Project Manager**” window and by following “**Reports**” -> “**Implementation Log Files**,” browse through the file for warnings, errors and utilization. The six warnings are in the beginning of the file. The utilization figure is seen by scrolling down a little.
  - ⇔ If there are errors or a different number of warnings or a different utilization number after the Xilinx Implementation, request help from the professor or TAs.

#### ***Sourcing the Schematic from Experiment 3***

We will **source** the schematic file of Experiment 3 to Experiment 4. That is, we will copy the Block 4 or ppm4.sch of Experiment 3 to Experiment 4 in a **special** way. The reason why we source the schematic, instead of straight copying and pasting is that sourcing keeps the component labels. This way, we save time for not entering the component

labels again. In addition, errors can happen when we enter component labels. This means the circuits that worked during Experiment 3, do not work during Experiment 4. Also, copying and pasting do not keep pad names and pin numbers. One has to enter them again.

- On the Project Manager window, click on “**Project**” and select “**Add Source File(s)...**” The Project Manager will show the content of the **ppm** folder in the “**Add Document**” window. Change the folder to the **exp3/ppm** folder which contains the schematic files of Experiment 3. In the folder select the fourth Ppm schematic file, ppm4.sch and then click “**Open.**” The project manager will give a warning that the current fourth schematic in **exp4/ppm** will be overwritten. Click “**Yes**” for the overwrite question.

In order to make sure there was no problem with the sourcing of the schematic, we will do another Xilinx Implementation and see that the Implementation Log File has six “no-load” warnings, no error messages and 80% FPGA chip utilization. But, first, we will clear the implementation data by following “**Project**” -> “**Clear Implementation Data**” on the Project Manager window so that the old Xilinx Implementation data is not used for the current Xilinx implementation. Clearing deletes the Implementation Log File. Otherwise the log file is appended the messages of the new Xilinx Implementation. Here is the new Xilinx Implementation :

- Do a Xilinx implementation of the exp4 project.
- Starting with the “**Project Manager**” window and by following “**Reports**” -> “**Implementation Log Files,**” browse through the file for warnings, errors and utilization.
  - ⇨ If there are errors or a different number of warnings or a different utilization number after the Xilinx Implementation, request help from the professor or TAs.

### Schematic design

- Click on the “**Schematic Editor**” button of the Flow panel of the Project Manager to view the schematic sheets.

As the schematic editor opens the schematic sheets, you will see these are **not** project files. They are labelled **Non-Project** :

- Close all of these six schematic files one by one, then the schematic editor. Then, on the project manager window, double click on each schematic file name one at a time to open the project schematics.
- Select schematic 1 or ppm1.sch to enter the team information.
  - ⇨ Enter a teammate’s name on the first line of the lower right table.
  - ⇨ Enter the other teammate’s name on the second line of the lower right table.
  - ⇨ Enter teammates’ section on the third line of the lower right table.

There are four (4) macros to be implemented. Students are suggested that implement them in the order of M2, M3, M4 and M5. Students are strongly advised they perform functional and timing simulations of each black box, macro, extensively, before starting its schematic design. Students have enough time to complete the schematic, move the design to the FPGA board and test on it. This way, they can fix and understand errors and warnings faster. Students are suggested that they follow the following steps given for macro M2 for all the other macros to be replaced :

- Select schematic 4 or ppm4.sch to view Block 4, the Play Check Block. Zoom into the subblock named Compare on the schematic. It has now one macro M2. **Your goal is to implement M2.**
  - ⇨ M2 is a macro, but not an XDB. That is it is not designed by Xilinx. M2 is a **user design block**, designed by the professor. There is a convention for a macro that the inputs are on the left side and the outputs are on the right side. However, one can rotate symbols (macros) like XDBs. This macro has been rotated to fit the area for this subblock. One can also change the size of the symbol and the position of the inputs and outputs by right clicking on the symbol and selecting the “**Symbol Editor**” from the menu. The symbol editor has an easy-to-use interface and with the help of the mouse one can make changes on the symbol. One can create a macro from a schematic and also from a VHDL program. The macros on this sheet are schematic macros. One can convert a schematic to a **netlist** and then ask from the Project Manager to convert it to a macro. The Project Manager uses the information in the netlist to convert the schematic to a black box (a symbol) with inputs and outputs whose names the schematic

designer indicates. This macro definition would be placed on the Symbols Toolbox library and is visible only to this project. You will see the current user macro, M2, shown on top of the Spartan components (gates, flip-flops and XDBs) under the title “**BLK4AB**.”

Students are suggested that they follow the following steps for this experiment :

- Study macro, M2 (BLK4AB), by doing functional and timing simulations.
- Download the project to the FPGA chip and do extensive testing on the FPGA board. That is, play the game !
- **Delete** the BLK4AB, M2, macro, **without** deleting its wires in order to use them later for your circuit.
- On the Schematic Editor implement the M2 circuit where
  - ⇨ There are single-rail inputs, and
  - ⇨ An **integrity test** is performed on each subcircuit to see if there is any wiring/component problem. If yes, follow the error messages on the bottom of the Project Manager window to correct them in the circuit.
  - ⇨ **Each subcircuit is designed and functionally simulated separately, before starting the next subcircuit. To do functional simulations of each subcircuit, see Section 3.2 (testing) below. If there is an error, change the schematic design as described in Section 3.3 (modifying) below.**
  - ⇨ Make sure the circuits are **beautified**. This means components that do similar work form horizontal and vertical lines, line tanglings are minimized and there is enough space between components and sub-blocks to identify them.
  - ⇨ The schematic files are saved.

To do the schematic design (to implement the circuits of M2 by using components), we will apply the three implementation steps given in the beginning of Section 3.1.2. Teams are reminded of the following points :

- **Do not show your schematic to other teams.**
- **Block 4 schematic must be on one sheet each.**
- Team members must first discuss how to collaborate on major lab steps, and then proceed. **During the design try to save hardware and time !**

### 3.2. Test (Functional Simulation) :

- Perform **functional simulations** on, individual subcircuits. Then, connect them one at a time to each other and perform additional simulations. Finally, perform simulations on the complete circuit. Note that for the complete circuit, the number of inputs is large, so you cannot try all possible input combinations. Thus, carefully think about which input combinations (**test vectors**) to apply with your partner. Make sure to write down the test vectors and what you observe so that you can figure out the cause of an error quickly and also not to apply a test vector again and again unnecessarily.
- ⇨ If you realize you can minimize the circuit while testing, note what it is. Do the minimization **after** you think it is highly unlikely you would catch any more errors. After the minimization, continue doing the testing (simulation) until you are sure the minimization is correct and does not introduce new errors.
- ⇨ If you are not sure about the circuit response, open the ppm project in the termproject folder and observe the response of macro M2 for the same error generating case.

### 3.3. Modify the Circuit :

If an output value is not correct, you will need to go back to your schematic design to correct the error :

- Modify the circuit when errors are encountered. That is **modify the schematic design**. Then, go back to the Functional Simulation step above after an integrity test.

#### 4. Development Cycle on Breadboards with FPGAs :

- Perform the **Xilinx Implementation** on the circuit to obtain the bit file as described in the handout titled *Development Cycle on Breadboards with FPGA Chips*. Here is a brief summary :
  - ⇨ The Xilinx implementation has to be optimized so that the FPGA resources are utilized efficiently and time delays are reduced. For that, students will change the implementation options before they click **“Run”** when they are implementing the first time (or they cleared the implementation data by following **“Project”** -> **“Clear Implementation Data”** on the Project Manager window). The selected options will require the Xilinx software to generate a *faster and more efficient* design. If they have already done the Xilinx implementation, then changing the options is as follows :
    - ➔ Go through the following menu selections **“Implement Design...”** -> **“Options...”**
      - Change the **“Place and Route Effort Level”** to the maximum : **“High Effort.”**
      - Go through the following menu selections starting with the same window. Locate the **“Program Options”** area of the window. Click **“Edit Options...”** of the **“Implementation :”**
      - On the **“SPARTAN Implementation Options: Default”** window, click on **“Place and Route.”**
      - Locate the **“Router Options”** area and then change the **“Routing Passes”** level from **“Auto”** to **“5”** and the **“Delay-Based Cleanup Passes”** level from **“0”** to **“5.”**
      - Click **“OK”** twice to close both windows.
    - ⇨ Start the Implementation and allow it to complete to generate the bit file named **“ppm.bit.”**
  - If there is a problem, the Implementation will be stopped. Read the error messages in the **“Implementation Log File”** accessible from the **“Reports”** selection of the Project Manager window.
  - Perform extensive timing simulations on the circuit, by trying the input combinations used during the functional simulations and observing the outputs.
  - Make sure to observe the outputs in the **“Glitch”** mode to see if there are timing hazards.
  - Perform a **Xilinx Programming**. That is, **download** the bit file to the FPGA chip.
  - Test the circuit on the FPGA board, by trying the input combinations used during the functional and timing simulations and observing the outputs. That is, play the game extensively to make sure you test your design as much as possible.

If an output does not match your expected result, you will need to go back to your schematic design to correct it :

- **Modify** the schematic design on the computer if errors are encountered. That is, modify the schematic design as described in Section 3.3.
- Perform **functional simulations** as described in Section 3.2, after an integrity test and proceed accordingly.

#### 5. Remaining Step :

- Make sure again that the circuits are **beautified**, meaning components doing similar work form horizontal and vertical lines, line tanglings are minimal and there is space between components and subblocks to identify them.
- Make sure that you continuously look at the Experiment 4 Check List handout to remember and follow design conventions and avoid errors.
  - ⇨ Make sure the final Xilinx Implementation has been done. That is, on the Project Manager window, all schematic files have a check mark next to their names.
- **When done, signal to a TA to submit your project. Your project will be copied to a zip disk.**