

EXPERIMENT 3 ADDPD3RD DIGITAL CIRCUIT

1. Goals

Develop a combinational circuit which will be a portion of the **Compare** Subsubblock of Block 4, the **Play Check Block** of the term project and reinforce the following concepts studied the first four lab sessions:

- logic design on paper,
- schematic design on Xilinx software,
- how to simulate and verify logic circuits,
- the Digilent XLA FPGA board, and
- how to test circuits on FPGA chips.

2. Summary

A combinational circuit that adds, subtracts and compares numbers will be designed. The circuit called “**ADDPD3RD**” will be a part of Block 4 of the term project. The term project has 6 blocks : Block 1, the Control Unit ; Block 2, the Input/Output Block ; Block 3, the Human Play Block ; Block 4, the Play Check Block ; Block 5, the Points Calculation Block ; Block 6, the Machine Play Block. Below, we will describe the development of the ADDPD3RD circuit on paper : the black-box view, input-output relationship and its partitioning. The remaining work will be done by students themselves.

The ADDPD3RD circuit is in the Compare Subsubblock of Block 4. The block, the Play Check Block, performs operations for both players. For example, based on Block 4 results, (i) certain plays are not allowed by the Control Unit, (ii) the machine player decides how to play RD and (iii) the Points Calculation Block calculates reward points. The Compare Subsubblock in Block 4 adds/subtracts the random digit, RD, to/from position displays PD0, PD1, PD2 and PD3. The ADDPD3RD adds/subtracts to/from PD3. In the schematic of Block 4 at the course web site, the ADDPD3RD circuit is implemented by **macro M1**. Students will delete the macro and place their own circuit there.

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 will be 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

The 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 *change of the design* based on test results. If the circuit is simple one can completely design it, i.e. the components, on paper and then place the same components on the computer screen. For ADDPD3RD, it is not the case. So, we first obtain the black-box view and initial block partitioning of the circuit 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 digital circuit is viewed as a black box where only the inputs and outputs are considered. Then, the outputs are related to the inputs. The ADDPD3RD circuit has 12 inputs and 11 outputs. The black box view and the detailed view of the inputs and outputs 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. Black-box view of the ADDPD3RD circuit, macro M1.

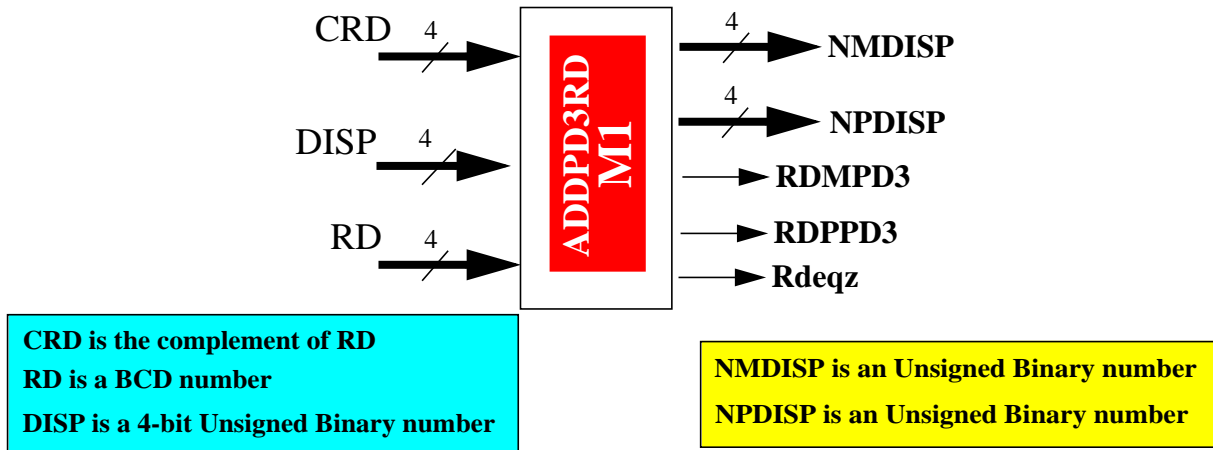


Figure 2. Inputs and outputs of the ADDPD3RD circuit, macro M1.

Input	Number of lines	Description
CRD	4	The complement of the random digit
DISP	4	Position Display 3 (PD3). DISP stands for display. A 4-bit Unsigned Binary number
RD	4	Random digit. A BCD number

Table 1: Inputs of the ADDPD3RD circuit.

Signal name	Number of lines	Description
NMDISP	4	New value of Position Display 3 (PD3) after the random digit is subtracted which is $DISP - RD = PD3 - RD$. It is a 4-bit Unsigned Binary Number. If the result of the subtraction is less than 0, the RDMPD3 output described below is 1. Otherwise, RDMPD3 is 0
NPDISP	4	New value of Position Display 3 (PD3) after the random digit is added . It is the result which is $DISP + RD = PD3 + RD$. It is a 4-bit Unsigned Binary Number. If the result of the addition is greater than $(15)_{10}$, the RDPPD3 output described below is 1. Otherwise, RDPPD3 is 0
Rdeqz	1	RD is equal to 0. When it is 1, it indicates that RD is equal to zero
RDMPD3	1	“Position Display 3 minus RD.” When it is 1, it indicates $PD0 - RD < 0$
RDPPD3	1	“Position Display 3 plus RD.” When it is 1, it indicates $RD + PD0 > (15)_{10}$

Table 2: Outputs of the ADDPD3RD circuit.

The random digit, RD, is a BCD digit. Its 1's Complement is CRD, standing for Complemented RD. CRD is used to subtract RD from PD3 in the 2's Complement system. The four DISP lines in this experiment represent PD3 (Position Display 3). The display on the leftmost side is named PD3 and its lines are named DISP[12] through DISP[15] which are input to the ADDPD3RD circuit.

NMDISP is the subtraction of RD from DISP : PD3 - RD. The subtraction is done in the 2's Complement system, and the subtraction is converted to an addition. That is, NMDISP is obtained by performing CRD + DISP + 1 which is PD3 + CRD + 1. RDMPD3 is 1, if PD3 - RD is negative. NPDISP is a simple addition of RD and DISP : RD + DISP. RDPPD3 is 1, if there is a display overflow when PD3 and RD are added. That is, the result of the addition is greater than $(15)_{10}$. Rdeqz is a result of the comparison between RD and 0. If RD is equal to 0, Rdeqz is 1. Therefore, the ADDPD3RD circuit performs two additions, and three comparisons :

- PD3 - RD = DISP - RD = DISP + CRD + 1 = NMDISP and if the sum result < 0, RDMPD3 is 1
- PD3 + RD = DISP + RD = NPDISP and if the sum result > $(15)_{10}$, RDPPD3 is 1
- RD is compared with zero and if RD = 0, Rdeqz is 1

Now that we know the top-level view of the system, i.e. the black box view and the input/output relationship, we will try to implement it. First, we check to see if a Xilinx component immediately implements ADDPD3RD. A search on the Xilinx component library reveals that there is no component that implements ADDPD3RD. Therefore, we have to implement it ourselves, by exposing more details of the ADDPD3RD circuit. Since the truth table of the ADDPD3RD circuit is large, as there are 12 inputs, requiring 2^{12} , or 4096, rows on the table, we have to partition the black box into blocks. After we get the blocks, if there are no available components implementing them immediately, we will continue to divide them until either they are implemented immediately or they are small enough to be designed quickly, i.e. small enough to get the truth tables and the gate networks.

How can we partition ADDPD3RD ? We know that a circuit is partitioned into subcircuits based on major operations on the **operation table**, design goals and available components. Also, these subcircuits need to be of **equal** size, if no practical or logical restrictions apply. An operation table has much less detail than a truth table. It is textual, but still precise. The operation table indicates explicitly or implicitly the major operations that make up the input/output relationship. These major operations indicate the blocks. The operation table of ADDPD3RD can be obtained by studying Tables 1 and 2 above and converting them to a summary. How does the ADDPD3RD operation table look like ?

First, there is **no** one, single operation table for a circuit. One can obtain different operation tables for the same circuit. Second, this operation table, the summary, is technology dependent. Based on which components are available, one picks the major operations so that a minimum number of components are used. Thus, knowing low-level details influences the high-level decisions. Clearly, this is **not** exactly top-down design. However, top-down design has the inherent disadvantage that successive partitionings of a block can result in impractical circuits if one does **not** consider (or at least have a feeling of) lower levels. For example, if one completely ignores the physical attributes of components until the last moment, the circuit may be too large, too expensive, too much power consuming, etc.

If one takes into account available Xilinx components for the ADDPD3RD circuit, the operations table, hence the major operations of ADDPD3RD can be as shown on Table 3. There are two additions and three compares ! Two of the three compares are related to the additions and can be combined with them. Does it mean then that we partition the ADDPD3RD circuit into **three** subcircuits : two additions with compares and one separate compare ? The answer is “**Yes**” since these subcircuits are **immediately** implementable by Xilinx components. Thus, the ADDPD3RD circuit is partitioned into the three subcircuits as shown in Figure 3.

Operation
NMDISP = PD3 - RD = DISP + CRD + 1 and RDMPD3 = 1 if DISP - RD < 0
NPDISP = PD3 + RD and RDPPD3 = 1 if DISP + RD > $(15)_{10}$
Rdeqz = 1 if RD = 0

Table 3: Operation table of the ADDPD3RD circuit.

We are done with the design of the ADDPD3RD circuit 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 ADDPD3RD 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**) and then start the design on computers. We will practice it often this semester.

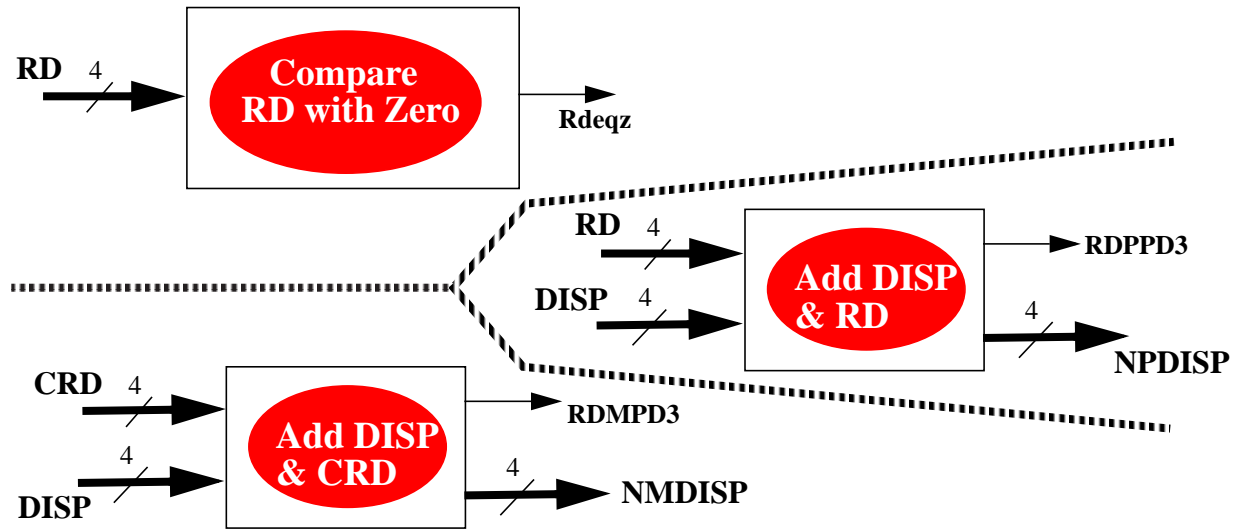


Figure 3. The partitioning of the ADDPD3RD circuit, macro M1.

3.1.2. Implementation

The implementation means determining the components based on the major operations, components and given product 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 subcircuit 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 subcircuit. 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 of the three subcircuits of ADDPD3RD, 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. ADDPD3RD 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.

First task before starting a new project :

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

- By using "**My Computer**" create a new directory named "**exp3**" 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 ADDPD3RD circuits :

- Copy the "**ppm**" folder and the "**ppm**" Xilinx pdf file from the "**termproject**" folder to the "**exp3**" 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 **exp3** directory.

In order to make sure there was no problem with the copying of the project to the exp3 folder, we will do a Xilinx Implementation and see that the Implementation Log File has six "no-load" warnings and no error messages :

- Do a Xilinx implementation of the exp3 project.
- Starting with the "**Project Manager**" window and by following "**Reports**" -> "**Implementation Log Files**," browse through the file for warnings and errors. The six warnings are in the beginning of the file.
 - ⇨ If there are errors or a different number of warnings 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.
- 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.
- Select schematic 4 or ppm4.sch to view Block 4, the Play Check Block. Concentrate on the subblock named Compare in the middle of the schematic. It has two black boxes or macros, named M1 and M2. M1 implements the ADDPD3RD circuit. **Your goal in this project is to implement M1.**
 - ⇨ M1 is a macro, but not an XDB. That is it is not designed by Xilinx. M1 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. 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, M1, shown on top of the Spartan components (gates, flip-flops and XDBs) under the title "**BLK4AA**."

Students are strongly advised they perform functional and timing simulations of the ADDPD3RD black box, **M1** or **BLK4AA**, given by the professor extensively, before starting the 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. Hence, students are suggested that they follow the following steps for this experiment :

- Study macro, M1 (BLK4AA), 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 BLK4AA, M1, macro, without deleting its wires in order to use them later for your circuit.
- On the Schematic Editor implement the ADDPD3RD 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.**
 - ↪ The circuit is beautified.
 - ↪ The schematic files are saved.

To do the schematic design (to implement the subcircuits of Figure 3 by using components), we will apply the three implementation steps given in the beginning of Section 3.1.2. We know that the ADDPD3RD circuit has three major operations as shown on Table 3 : two additions with compares and a separate compare. The three subcircuits for the operations are Add DISP and CRD, ADD DISP and RD and Compare RD with Zero, as shown in Figure 3. We also know that these subcircuits are **easily** implementable by available Xilinx components.

First, we will obtain an input/output relationship in the form of an operation table for each subcircuit on Table 3. The operation table for the Add DISP and RD is on Table 4 :

Operation	Situation
$NPDISP = DISP + RD < (16)_{10}$	NPDISP is correct (no display overflow) & RDPPD3 = 0
$NPDISP = DISP + RD > (15)_{10}$	NPDISP is not correct (display overflow) & RDPPD3 = 1

Table 4: Operation Table of the Add DISP and RD subcircuit.

Then, we will implement the operation table. We try if step (i) of Section 3.1.2 is applicable for the subcircuit. We search the Xilinx component library and see if there is a single component that implements the subcircuit. The answer is “Yes,” meaning we no longer have to partition it. Which component is it ? It is the Xilinx 4-bit ADDer, **ADD4**. The ADD4 Xilinx component is identical to the 4-bit ADDer studied in class with the exception that the ADD4 has an additional output called **OFL** which stands for overflow. Obviously, the “**CO**” output is the carry out output used for both Unsigned and 2’s Complement additions. If the CO output is 1 in Unsigned Binary additions, it signals an overflow. If the OFL output 1, it signals an overflow for 2’s Complement additions

Students will place the ADD4 XDB in place of the BLK4AA macro on the screen, paste some of the wires that were attached to macro M1. If those wires are deleted, new wires are drawn on the screen and then labelled. Students note that the Xilinx **GND** device is used as Logic 0 and is **not** labelled as a component. A close analysis of Table 4 and the ADD4 macro indicates that when $DISP + RD$ is greater than $(15)_{10}$, C_{out} is 1, so is RDPPD3 ! Similarly, when $DISP + RD$ is less than or equal to $(15)_{10}$, C_{out} is 0, so is RDPPD3 ! This operation table (Table 4) is implying that the ADD4 also indicates if there is a display overflow. Thus, there is **no** need to use a separate comparator ! That is, both major operations can be implemented by a single component ! This is a good example where using digital logic techniques and components carefully reduces hardware, cost and power consumption, without sacrificing the speed and reliability ! Students now just need to name the CO output of ADD4 as RDPPD3 ! This will complete the Add DISP and RD subcircuit. We are done one out of three subcircuits.

The operation table for the Add DISP and CRD is on Table 5. Again a single Xilinx component, the ADD4 implements the Add DISP and CRD subcircuit. Students will place this ADD4 on the screen next to the other ADD4 component, copy and paste some of the wires that were attached to macro M1. If those wires are deleted, new wires are drawn on the screen and then labelled. The CO output is labelled as RDMPD3. Also, the Xilinx VCC device is used as Logic 1 and is **not** labelled as a component. This will complete the Add DISP and CRD subcircuit.

Operation	Situation
$NMDISP = DISP + CRD + 1 = DISP - RD > 0$	NMDISP is positive & RDMPD3 = 1
$NMDISP = DISP + CRD + 1 = DISP - RD < 0$	NMDISP is negative & RDMPD3 = 0

Table 5: Operation Table of the Add DISP and CRD subcircuit.

Why is that a single ADD4 accomplishes Table 5? To answer it, let's consider the two cases (negative and not negative) in Figure 4. We see that when we do the subtraction, the second number, RD, is negative (its sign bit, the leftmost bit is 1) and adding it to the sign bit of DISP which is 0, will give 1, meaning a negative number. Thus, the result is not negative if the leftmost bit position gets a carry from the ADD4 output. However, if the ADD4 carry out is 1, its addition to the sign bits generates a 0 for the sign bit of the result which means the result is positive. Thus, if the result is positive, CO of ADD4 is 1 and if negative CO is 0.

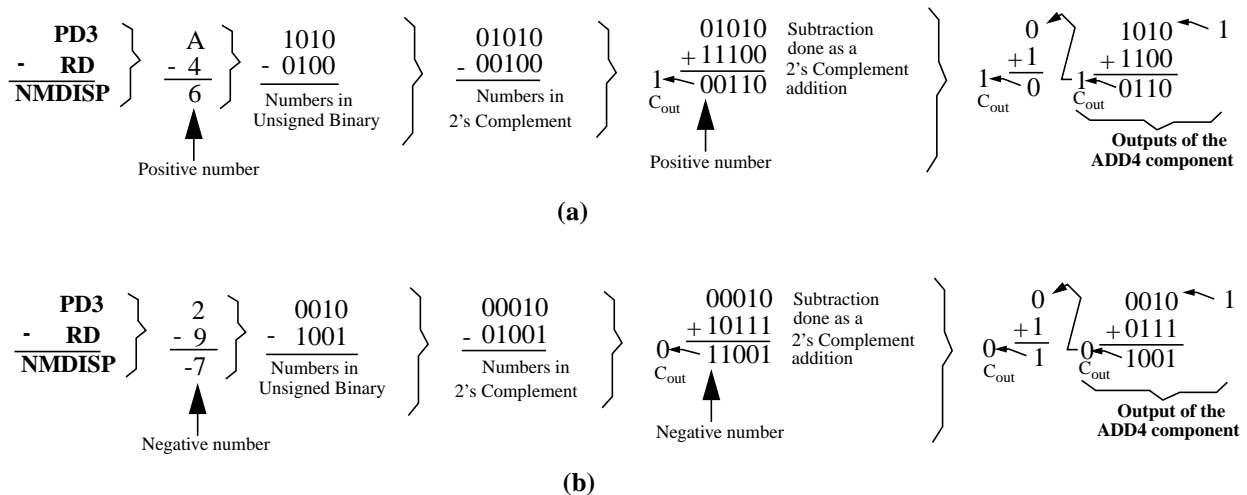


Figure 4. Determining if the subtraction results in a negative number : (a) the result is positive, and (b) the result is negative.

The last subcircuit, the Compare RD with Zero Subblock, operation table is given on Table 6. We apply step (i) above to see if there is an XDB that implements this subcircuit immediately. The answer is "Yes." We can use the 4-bit Magnitude (Unsigned Binary) Comparator XDB, **COMP4**, which checks for the "equal to" (EQ) case. However, as we think about this subcircuit, we realize that the COMP4 is more than what we need as we are comparing a number with a constant! We are **not** comparing two numbers. There is, in fact, a simpler component that performs the same function! It is a 4-bit NOR, **NOR4** that outputs a 1 if all of its four inputs are 0. That is, we skip step (i) and switch to step (ii) for this subcircuit! By using this single simpler component, we save space on the FPGA chip! Students must note that in general, if a circuit uses constants, it can be simplified.

Situation	Operation
$RD = 0$	$Rdeqz = 1$
$RD \neq 0$	$Rdeqz = 0$

Table 6: The operation table of Compare RD with Zero subcircuit.

Note that this last subcircuit is another example of using digital logic techniques and components to our advantage to reduce hardware (and so, the cost, power, size and weight), without reducing the speed ! Also, this is a good example of how knowing available components can help us partition circuits better ! However, as noted above, this style of design is not exactly top-down design ! Nevertheless, reality forces engineers to consider lower levels while they design the higher levels. Students will practice this often in CS2204, after this semester and after graduation. This style of design is called “**middle-out.**”

We have designed each subcircuit, so, we stop the partitioning of the ADDPD3RD circuit ! We also know that **all the subcircuits contain combinational circuits, i.e. gate networks and XDBs with gate networks.** Before starting the simulation, students need to remember to :

- ⇒ Perform an **integrity test** 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 the problem in the circuit.
- ⇒ Make sure the schematic files are saved.

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 ADDPD3RD circuit. Note that for the complete ADDPD3RD 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 M1 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 4.3.
- Perform **functional simulations** as described in Section 4.2, 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 3 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 show your project and receive comments.**

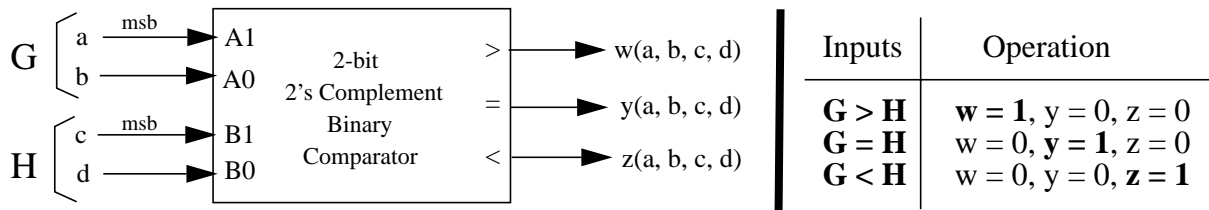
6. Exercises

1) Develop a 2-bit **2’s Complement Binary** Comparator. In order to design the 2-bit 2’s Complement comparator, study the Logic design initial implementation steps of the 2-bit 2’s Complement comparator below :

Logic Design :

The Input-Output Relationship :

The 2-bit 2’s Complement Comparator has **four** inputs and **three** outputs. It accepts two 2-bit 2’s Complement numbers, G and H, and generates outputs : w, y and z. When G is greater than H, w is 1 (high, active) while the other two are 0 (low, inactive). When G is less than H, only z is high. In the last case, only y is high. A 2-bit 2’s Complement Comparator can have its inputs negative ! We can have negative numbers, for example, (ab) = (11) = (-1)₁₀. We now give the black box view, the operation table, and the truth table below :



**G and H are 2-bit 2’s
Complement Binary numbers**

	G	H	a	b	c	d	w	y	z
0	0	0	0	0	0	0	0	1	0
1	0	1	0	0	0	1	0	0	1
2	0	-2	0	0	1	0	1	0	0
3	0	-1	0	0	1	1	1	0	0
4	1	0	0	1	0	0	1	0	0
5	1	1	0	1	0	1	0	1	0
6	1	-2	0	1	1	0	1	0	0
7	1	-1	0	1	1	1	1	0	0
8	-2	0	1	0	0	0	0	0	1
9	-2	1	1	0	0	1	0	0	1
10	-2	-2	1	0	1	0	0	1	0
11	-2	-1	1	0	1	1	0	0	1
12	-1	0	1	1	0	0	0	0	1
13	-1	1	1	1	0	1	0	0	1
14	-1	-2	1	1	1	0	1	0	0
15	-1	-1	1	1	1	1	0	1	0

$$w(a,b,c,d) = \sum m(2,3,4,6,7,14)$$

$$y(a,b,c,d) = \sum m(0,5,10,15)$$

$$z(a,b,c,d) = \sum m(1,8,9,11,12,13)$$

The implementation of the 2-bit 2's Complement Binary Comparator :

We check if there is a 2-bit 2's Complement XDB. There is none. Therefore, we implement it. We decide to use three gate networks for which three switching expressions are determined. We will obtain minimal SOP expressions by using the K-map technique. Students are asked to get also the minimal POS expressions and compare the amount of complexities. The expression for output w is below. The other two minimal SOP expressions follow similarly:

$w(a,b,c,d) = \sum m(2,3,4,6,7,14)$

$w(a, b, c, d) = \bar{a}c + bcd\bar{d} + \bar{a}b\bar{d}$

1 2 3
epi epi epi
②③⑦ ⑭ ④

2-bit 2's Complement Binary Comparator Circuit Design on the Computer (Xilinx Software) :

First task before starting a new project :

Keep the 2-bit 2's complement comparator project in the "exercise1" folder of the exp3 folder on the S drive.

Create a New Project :

- On the Project Manager, start a new project named "tsc2cmp," standing for 2-bit 2's Complement Comparator in the exercise1 folder.

The schematic design

- On the Schematic Editor draw the three gate networks for the three outputs where

⇔ There are **single-rail** inputs,

- ↔ Determine the pin assignments for the inputs and outputs.
- 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.
- Make sure to perform an **integrity test** to see if there is any wiring/component problem. If yes, follow the error messages to correct the problems in the circuit.
- Make sure the schematic file is saved.
- Make sure the circuit is **beautified**. This means components that do similar work form horizontal and vertical lines, line tanglings are minimized and there is enough space between components to identify subcircuits.

Remaining Steps :

- Perform functional simulations on the complete circuit where all 16 combinations of the truth table are tried.
- Modify the schematic if necessary and continue with functional simulations. If no more errors are discovered, continue with the step below.
- Development Cycle on Breadboards with FPGAs :
 - Perform the **Xilinx Implementation** on the circuit to obtain the bit file.
 - If there is a problem, the Implementation will be stopped. Read the error messages in the "**Implementation Log File**."
 - Perform a timing simulation **Xilinx Verification** on the circuit.
 - 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 all 16 input combinations and observing the outputs

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 when errors are encountered. That is, modify the schematic design.
- Perform the **functional simulation** step above after an integrity test and proceed accordingly.
- Functional simulations on the complete circuit where all 16 combinations of the truth table are tried.

2) Implement a 2-bit unsigned binary comparator, by using just **two** Xilinx Design Blocks. Assume single-rail inputs. Give the name "**exp3exer2**" to this project. Keep it in the **exercise2** folder.

3) Implement the 2-bit unsigned binary comparator by using a Xilinx Design Block and a gate. Assume single-rail inputs. Give the name "**exp3exer3**" to this project. Keep it in the **exercise3** folder.

4) Implement a 4-bit unsigned binary comparator, by using the block-based approach where the 4-bit comparator is implemented by two blocks and if necessary a few gates. Create an **exercise4** folder and keep the projects below in it. There are two ways for now to implement the 4-bit unsigned comparator :

- o Each block is the one designed in Exercise 2 above : Assume single-rail inputs. Give the name "**exp3exer41**" to this project. Keep it in **exercise4**.
- o Each block is the one designed in Exercise 3 above : Assume single-rail inputs. Give the name "**exp3exer42**" to this project. Keep it in **exercise4**.

CONTACTS :

1) Students can see the professor and teaching adjuncts (TAs), about the lectures, homework, and lab experiments.

2) Professor's contact information :

Room : 114 LC (718) 260-3101 Fax : (718) 260-3609 haldun@photon.poly.edu

Open-door policy to see the professor. If the door is closed, he might be in the lab.

Present in the lab : Mondays (4-6), Wednesdays (3-5) and Fridays (2-4)

3) Below are the contact information and assignments of the TAs :

Nikhil Joshi : 001LC, (718) 260-4011, njoshi01@utopia.poly.edu

- Present in lab session : Tuesday : 3-5:50 (B)

Sapan Shenoy : 102RH, (718) 260-3731, ssheno01@utopia.poly.edu

- Present in lab sessions : Tuesday : 6-8:50 (C), Thursday : 6-8:50 (A)

Jeff Tao : 233LC, (718) 260-3420, jefftao@photon.poly.edu

- Present in lab session : Wednesday : 6-8:50 (D)

Bo Yang : 001LC, (718) 260-4011, yangbo@photon.poly.edu

- Present in lab session : Wednesday : 6-8:50 (D)

- Grading the homework

Peng Yao : 004LC, (718) 260-3975, yaopeng@utopia.poly.edu

- Present in lab sessions : Tuesday : 3-5:50 (B), Thursday : 6-8:50 (A)

4) All handout and lab files are at the course web site : <http://cis.poly.edu/cs2204>

5) Students are asked to give feedback about the announced lab hours in case the hours are not sufficient and/or need to be rescheduled.

6) When short-term problems are encountered in PC labs, students are advised to contact : help@duke.poly.edu or (718) 260- 3123 or go to Room : 337 RH.

For CS2204 lab related issues and to have the lab open, students need to contact the CIS lab supervisor Mr. Keni Yip at (718) 260-3023, keni@poly.edu. His office is 225RH

For longer-term problems in PC labs and **any other matter**, students should not hesitate to contact the professor and the TAs.