CS 2204 DIGITAL LOGIC & STATE MACHINE DESIGN FALL 2017

PPM PROJECT

1. Overview

The goal of the term project is to enhance a game project, named *Polytechnic Playing Machine*, ppm. Students are also allowed to modify the project provided that they design a digital system !

The ppm project has three different versions. The ppm project to be enhanced is named machine vs. machine : **ppmvsm**. The other two versions of the ppm game at the course web site are : (i) **a** human player playing against a machine player, human vs machine : **ppmhvsm**, and (ii) two human players play against each other, human vs human : **ppmhvsh**. This handout describes the ppmhvsm project. That is, it describes the second version, the human versus machine version in detail. Short descriptions of the other two versions are given at the end.

The ppm players, human and machine, play digits on four position displays to earn points. They play the game until one wins the game. Then, the game is restarted by **resetting**. The players take turns to play : Player 1 plays, then Player 2 plays, then, Player 1 plays, etc. Always, Player 1 starts the game when the FPGA is **downloaded** or reset. Player 1 receives a **pseudo-random digit** (simply random digit, RD) to start with. Player 1 may choose to play the digit or skip the play. Whether Player 1 plays or skips, when Player 2 gets its turn it receives a new random digit. It may play the random digit or skip the play. If Player 2 skips the play, Player 1 gets a new random digit and has the same two options as before : play or skip. If a player skips a play **no** points are subtracted. Also, both players are allowed to see the **next two random digits** which can help them plan their moves better. In summary, the game for each player is about playing a random digit on a position display to earn points, more than the opponent.

The random digit is a BCD digit, i.e. it is 0 through 9. Playing the random digit on a position display is either playing it directly (overwriting) or adding it to a display. If it is a direct play, the position is stored the random digit : RD. If it is an addition, the position is added the random digit and the result is placed on the same display. In either case, if the player has an adjacency, i.e. the digit played has an identical neighbor, the player earns more points. A 2-digit code which is not visible to the players and is on the rightmost two displays can help player earn even more points.

The largest value to play on a display is F, i.e. $(15)_{10}$. That is, the sum of RD and the position display value cannot be greater than $(15)_{10}$. If the sum exceeds $(15)_{10}$, this situation is called **display overflow**. If the player plays it, the digit played is the sum minus $(16)_{10}$. For example, if a position has E and RD is 9, after the addition the position is played 14 + 9 - 16 = 7. Then, the display blinks at a high rate, signalling there is a display overflow. If the player has an adjacency, the player earns more than 7 points. If the digit played is identical to the code digit on that position, the player earns additional points. Thus, winning the game is dependent on both chance and thinking.

The ppm circuit is a **digital system**. A digital system consists of digital circuits. Today's digital systems are numerous. Examples of digital systems are microprocessors, computers, DVD players and iPhones. They are also complex. Consequently, special emphasis is given to the coverage of digital systems in computer science and computer engineering curricula. For example, digital systems is a major topic of **CS 2214**, as well as **EE 4313**.

The ppm digital system consists of six blocks on six schematic sheets. Blocks 1, 2, 4, 5 and 6 are **core** blocks : They are already designed and provided with on schematics 1, 2, 4, 5 and 6. Block 3 on schematic 3 is the Machine Play Block and students will enhance it. Schematics 4 and 6 have black boxes or **macros**. Students cannot see their schematic implementations, but only VHDL ones. The machine vs machine or ppmmvsm project students enahnce has two additional schematic sheets to help student enhance their project better.

The experiments during the semester are planned as follows : Experiment 1 through 3 are for learning the hardware and software fundamentals of the lab as well understanding the term project. Experiment 4 completes the term project and affects the term grade. In Experiment 4, teams finalize how they enhance ppmmvsm project. In Experiment 4, students may instead elect to develop their own project as long as the project involves a digital system. Experiment 4 is collected in Lab 14 of each lab section. The experiments follow the class coverage where initial circuits are combinational circuits. Then, blocks with sequential circuits are designed. Below, we first describe the black box view of the ppm and then the individual blocks. The descriptions are in the context of digital systems. The description is also based on the Digitlent NEXYS-4 DDR FPGA board.

2. Black-Box View and the Input/Output Relationship (Operation) of the Ppm

The ppm project is a digital system. Starting this page, the the human versus machine version is described. The black-box view of this ppm is shown in Figure 1. It has 13 inputs and 19 outputs. One of the inputs is a clock signal generated on the board. All the remaining signals are connected to I/O devices such as **switches**, **push buttons**, **7-Segment displays**, and **LED lights** which are used to enter decisions and visualize game situations.

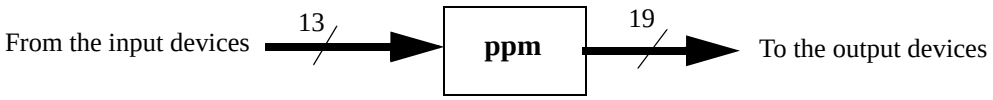


Figure 1. The ppm black box view.

2.1. The Ppm Input/Output Devices

This handout describes the use of the rightmost 4 7-segment displays, rightmost eight LED lights and the rightmost eight switches of theboard. The other input/output devices can be used by the students to do hardware debugging. The ppm I/O devices used, other than the clock are shown in Figure 2. The detailed view of the ppm inputs and outputs is shown in Figure 3. The description of the inputs and outputs of the ppm are shown on Table 1 and Table 2, respectively. Eight inputs are from the switches and four inputs are from push buttons. 11 outputs are for the 7-segment displays and eight are for the LED lights. One signal that is not connected to an I/O device is the **Clock** signal. It is at 100MHz and used to time and synchronize operations in the digital system. Some of the input and output devices have multiple purposes as explained below.

The switches are used to input decisions and a new random digit. If the switch handle is closer to the human player, it generates logic 0, otherwise it generates logic 1. Switches, SW4-SW7, are the Player 1 display select switches to play the random digit on a display. For example, switch, SW7 selects the leftmost position display, **PD3** ; SW6 selects the next display **PD2**, etc. **SW0** is used to indicate an addition by the human player. If it is 1, it means the human player wants to add the random digit. Finally the rightmost four switches, **SW15** through **SW12** are used to input a random digit for the machine player to speed up the design of the machine player as described below. To indicate that a random digit is being input, one of SW15 - SW12 must be turned on.

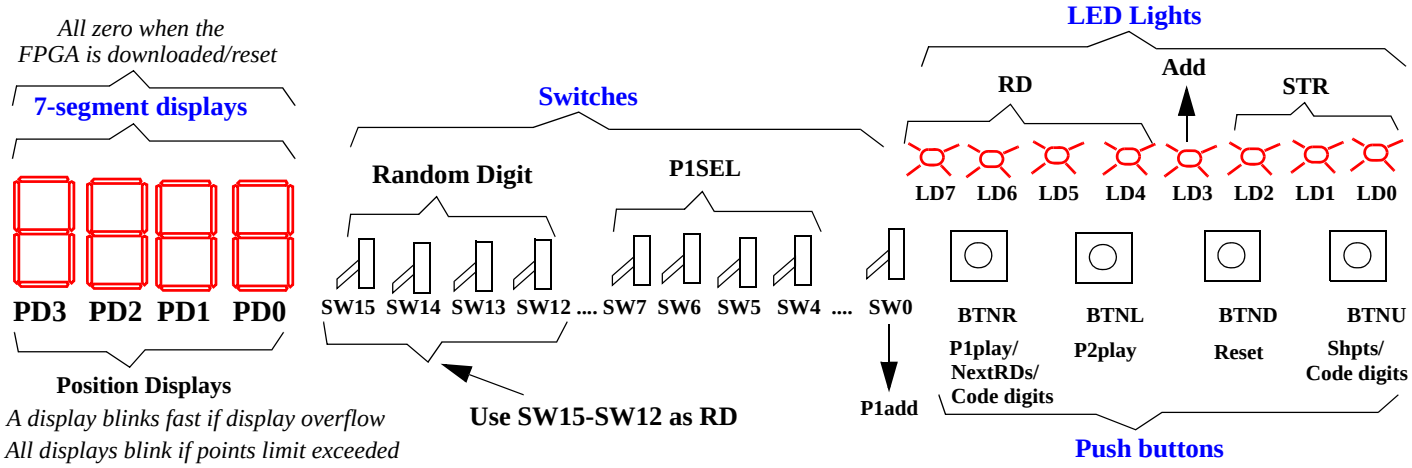


Figure 2. FPGA Board Input/Output device utilization of the ppm.

The four push buttons on the right, BTNR, BTNL, BTNU and BTND, are used by the human player to enter decisions or to get game status. Normally, when they are not pressed, they generate logic 0. As long as they are pressed, they generate logic 1. Push button 3, **BTNR**, has *four* uses. One is to give the turn to Player 1. When it is pressed, it means Player 1 wants to play. Another use is that it is pressed right after reset four (4) times in a row to get three (3) RDs and then to give the first turn to Player 1. The third use of BTNR is that before or after Player 1 plays, it can be pressed to see the next two RDs. The final use is that when pressed together with BTNU, the code digits discovered by the machine player are shown. Push button 2, **BTNL**, is used to give the turn to Player 2. Push button 1, **BTND**, resets the system, clearing all displays and points. When **BTNU** is pressed, players' points are shown on the four displays. When it is pressed together with BTNR, the code digits are shown.

11 outputs are connected to the four 7-segment displays named from left to right, PD3, PD2, PD1 and PD0. The dis-

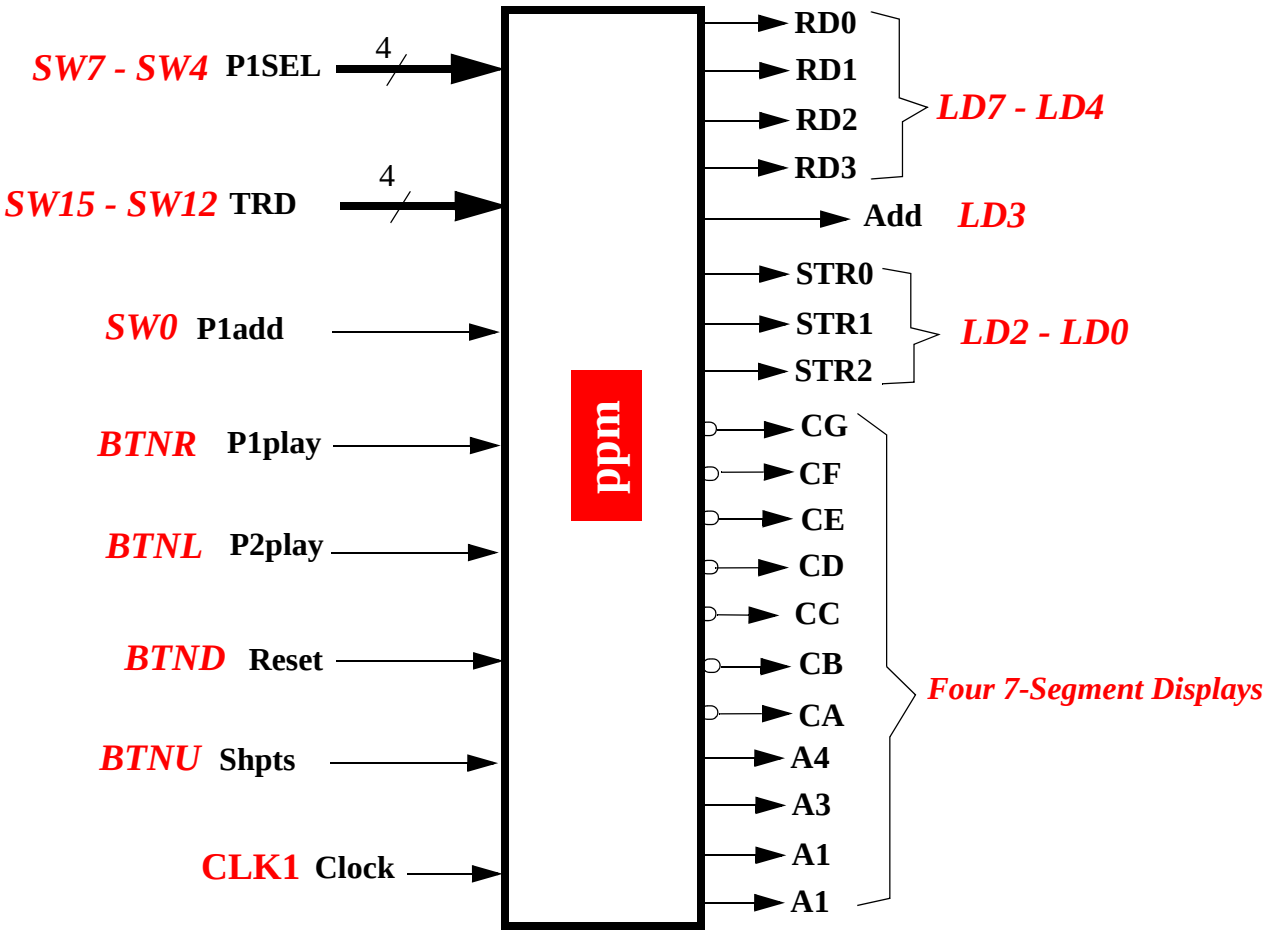


Figure 3. Inputs and outputs of the ppm.

plays show the four position display digits **by default**. They show players' points when BTNU (Shpts) is pressed : (PD1, PD0) show Player 1 points and (PD3, PD2) show Player 2 points. If BTNR pressed when Player 1 has to play, the rightmost two displays show the next two RDs. PD0 shows the next RD and PD1 shows the following RD. If BTNR and BTNU are pressed together, PD1 and PD0 show the code digits discovered by the machine player. All four displays blink if a player wins the game, i.e. when there is a points overflow. The 11 outputs are named **CA** through **CG** and **A4** through **A1**. Outputs, CA - CG, are **active-low**.

Parallel to the push buttons are eight LED lights, LD0 through LD7. All eight LED lights are used to show the game status. The current random digit, RD, is shown on LED lights **LD7 - LD4**. The random digit is a BCD number. LED light 3, **LD3**, emits light if an addition is performed on a display by the current player. LED lights **LD2 - LD0** show the current **state** the ppm system is in. That is, they indicate which step of the game ppm is in. The values are between 0 through 6. The state concept is explained in detail below.

Input device	Description
SW7 - SW4	Player 1 position display select. Each switch selects a position display to play a digit. Also, if one of the switches is 1 when Player 2 gets the turn, a random digit is input from SW3 - SW0
SW15 - SW12	Random digit input. A BCD random digit is input for the player
SW0	Player 1 add. When it is turned on, the current RD is added to the selected position display

Table 1: Inputs of the ppm.

Input device	Description
BTNR	It is used for four different reasons : (**i**) Player 1 wants to play. When pressed, it means the human player is ready to play ; (**ii**) Pressed four times after reset to get three RDs and allow Player 1 to play ; (**iii**) Pressed before or after Player 1 plays, it shows the next two RDs : PD0 shows the next RD and PD1 shows the following RD ; (**iv**) Pressed together with BTNU, the code digits discovered by the machine player are shown on PD1 and PD0.
BTNL	Player 2 can play. When pressed, it means the human player wants the machine player to play. In three situations it is pressed : (**i**) The human player presses it after playing on a position and wants the machine player to play ; (**ii**) The human player presses it to skip the play, so the machine player plays ; (**iii**) Pressed after the machine player plays with an adjacency.
BTND	Reset. When pressed, the game is reset such that all the displays, lights and points are cleared. It is as if the FPGA chip has just been downloaded the bit file. It can be pressed anytime
BTNU	It is used for two reasons : (**i**) Pressed by itself, players' points are shown on the four displays ; (**ii**) Pressed together with BTNR, it shows the code digits discovered by the machine player on PD1 and PD0.
Clock	The clock signal. It is a 100 MHz clock signal generated by an FPGA board circuit

Table 1: Inputs of the ppm.

Output device	Description
LD7 - LD4	Random Digit LED light outputs. The random digit is in BCD. They show the random digit of the current player
LD3	The current player adds. If it is on, it means the current player is adding RD to a position display
LD2 - LD0	The **state** of the game. They indicate where the game stands. The value shown is between 0 and 6. State 0 is the reset state. States 1, 2 and 3 are for Player 1 and states 4, 5 and 6 are for Player 2.
Four 7-Segment displays	By default, the displays show position displays, PD3-PD0, on which random digits are played. If BTNU is pressed, they show Player 2 points in Positional Hex on the left two displays and Player 1 points in Positional Hex on the right two displays. All four displays blink when there is a points overflow. A single display blinks fast if there is a display overflow. If a display is played with an addition and there is no display overflow, it blinks slowly. If BTNR is pressed before and after Player 1 plays, it shows the next two random digits on PD1 and PD0. If BTNR and BTNU are pressed at the same time, PD1 and PD0 show the code digits discovered by the machine player.

Table 2: Outputs of the ppm.

2.2. The Ppm Input/Output Description

The input/output relationship is the operation description which includes the game rules. A good understanding of the input/output relationship is needed to design the ppm digital system correctly. Formally, the input/output relationship should be given by an **operation diagram** since the ppm has sequential circuits. However, in order to simplify the discussion of the ppm system, we will first describe the playing rules in detail and then give the operation diagram.

2.2.1. The Ppm Playing Description

The two players, Player 1 (human) and Player 2 (machine), play until one wins the game. Then, the game is started by pressing the reset button (BTNR). Players take turns : Player 1 plays, then Player 2 plays, then, Player 1 plays, and so on. Always, Player 1 starts the game when the FPGA is downloaded or reset. The game for players is as follows :

- Play a random digit, RD, ($0 \leq$ RD $\leq (9)_{10}$), directly on display k, PD_k, $0 \leq k \leq 3$, or add the random digit to a display, PD_k, and store the result on the same display to earn points.
  - Players try to earn points more than the opponent, by looking for **adjacent** identical digits and code digits.
    - After the play, if the player has at least one adjacent identical digit, the player plays again.

- If a player chooses, the play can be skipped to give the turn to the opponent without losing points. A new random digit is given to the opponent. The opponent can play the random digit or skip the play.

2.2.1.1. An overview

After the FPGA chip is downloaded, all four 7-segment displays show zero, since by default, the four 7-segment displays show position displays, i.e. the digits played. Both players start with zero points. All LED lights are blank. If BTNU is pressed the displays show zero as players' points after downloading. BTNR is pressed four times so that human player plays first. The rightmost two displays show the next two RDs when BTNR is pressed again. The game involves guessing and thinking. Players play RD on a display and earn points depending on the value played on the display and the adjacency and code digit situations. The game ends if a player's points exceeds $(255)_{10}$. This points overflow condition is shown by blinking all four displays. The game is restarted by resetting. A player can skip a play, giving the turn to the opponent. The human player can reset the game any time.

When the FPGA is downloaded or reset, a 2-digit **code** is generated on the rightmost two displays, PD1 and PD0 : **Code**. The code is not changed until the game is over. A new code is generated once the FPGA chip is downloaded or the game is reset. The code is hidden from the players. They cannot see its digits. To determine the digits, the players have to analyze the game, especially the points earned after each play. However, playing on the rightmost two displays generate code digit rewards. Playing on the other two positions would not result in code digit rewards.

Once RD is played the reward points is calculated. A reward consists of a regular reward and a code reward. The regular reward depends on the adjacency. If there is no adjacency, the player earns what the display is stored. The player earns more if there are adjacent identical digits : If there is one adjacent digit, the earned points is **two** times the result digit played. Two adjacent identical digits means the player earns **four** times the result digit played. In the best case where there are three adjacent identical digits, the player earns **eight** times the result played. In addition, the player plays one more time. Displays on the two sides are **not** considered adjacent.

If the player plays a code digit on one of the rightmost two displays, the players earns the code reward. If the other display does not have the code digit, the player earns **eight** times the code digit as the code reward. In the best case, if the other display already has the code digit, then player earns the 2-digit code as the code reward.

2.2.1.2. The Ppm Play Sequence and Rules of Playing a Random Digit on Position Displays

This section lists the rules of the game. Although, the critical point in the game is after a random digit is received by a player, below we list all the rules in the order starting from **reset** :

1) When the FPGA chip is downloaded, all position displays, players' points and LED lights are zero. Note that the FPGA chip can be downloaded anytime, for example when the game is going on.

2) When BTNR is pressed four times in a row after downloading or a reset, three RDs are generated and then the human player gets the turn to play and receives a random digit automatically. The random digit is shown on the leftmost four LED lights.

3) After receiving the random digit following downloading or a reset, the human player has **two** options : play the digit on a display or skip. If the human player skips, he/she does **not** lose points. Whether the human player plays or skips, the machine player receives a new random digit automatically when it is its turn. The human player can press BTNR to see the next two RDs where PD0 shows next RD (R1D) and PD1 shows the following RD (R2D).

- Playing the random digit on a position display means either playing it directly on a display (overwriting it) or adding the random digit to a display and storing the result there.
  - Playing the random digit directly on a display is the **default** case. To play the random digit directly, the human player turns on a switch that corresponds to the display intended, then turns off the same switch. For example, to play on position 3 (the leftmost position) with direct playing, SW7 (the leftmost switch) is turned on then off ; to play on position 2, SW6 is turned on and off, and so on.
  - In order to add the random digit to a display, the human player first turns on SW0, then turns on the switch that corresponds to the display intended then turns off the position selection switch and then turns off SW0. For example, to play on position 0 (the rightmost position) with an addition, SW0 is turned on, then SW4 is turned on and off and then SW0 is turned off.
- If the human player wants to skip, BTNL is pressed. Since the human player is expected to play, pressing BTNL signals the human player wants the machine player to play, giving the turn to the machine player. The machine player receives a **new** random digit.

4) If the human player plays the random digit, RD, on position display PD_k, reward points are added to the human player points. The earned reward points depend on the digit played, the amount of adjacency and the code digits :

- If the human player plays the random digit on display PD_k directly, the result is $\mathbf{PD_k = RD}$ and the human player earns points which is the sum of regular and code reward points :
  - The regular reward :
    - **No** adjacent RD : RD points

- **One** adjacent RD : [(2*(RD)) plus another play]
- **Two** adjacent RD digits in a row : [(4*(RD)) plus another play]
- **Three** adjacent RD digits in a row : [(8*(RD)) plus another play]

- The code reward if the digit played is on display 1 or 0 :
  - The other display does **not** have its code digit : [(8*(RD))]
  - The other display does **have** its code digit : Code

- If the human player adds the random digit to display PD_k the played display gets the result : **(PD_k + RD)** :
  - If (PD_k + RD) < $(16)_{10}$, the human player earns points which is the sum of regular and code reward points :
    - The regular reward :
      - **No** adjacent (PD_k + RD) : (PD_k + RD)
      - **One** adjacent (PD_k + RD) : [(2*(PD_k + RD)) plus another play]
      - **Two** adjacent (PD_k + RD) digits in a row : [(4*(PD_k + RD)) plus another play]
      - **Three** adjacent (PD_k + RD) digits in a row : [(8*(PD_k + RD)) plus another play]
    - The code reward if the digit played is on display 1 or 0 :
      - The other display does **not** have its code digit : [(8*(PD_k + RD))]
      - The other display does **have** its code digit : Code
  - If (PD_k + RD) $\geq$ $(16)_{10}$, there is a display overflow. The display blinks at a high rate. The human player plays digit (PD_k + RD - $(16)_{10}$), and earns points which is the sum of regular and code reward points :
    - The regular reward :
      - **No** adjacent (PD_k + RD - $(16)_{10}$) : (PD_k + RD - $(16)_{10}$) points
      - **One** adjacent (PD_k + RD - $(16)_{10}$) : [(2*(PD_k + RD - $(16)_{10}$))) plus another play]
      - **Two** adjacent (PD_k + RD - $(16)_{10}$) digits in a row : [(4*(PD_k + RD - $(16)_{10}$))) plus another play]
      - **Three** adjacent (PD_k + RD - $(16)_{10}$) digits in a row : [(8*(PD_k + RD - $(16)_{10}$))) plus another play]
    - The code reward if the digit played is on display 1 or 0 :
      - The other display does **not** have its code digit : [(8*(PD_k + RD - $(16)_{10}$))]
      - The other display does **have** its code digit : Code

After the play, if there is a display overflow, the played position blinks at a high rate. Note that if a 0 is played on a display and there are adjacent 0s, the human player earns 0 points but plays again. Also, if 0 is a code digit and the other display has the other code digit, the player earns Code points.

5) After the human player plays RD, he/she is given a chance to review the situation. For example, the human player can press BTNR to see the next two random digits or BTN4 to see players' points. Then, the human player presses BTNL to give the turn to the machine player. The machine player receives its random digit automatically.

6) The machine player has **two** options : play the random digit on a display or skip. If the machine player skips, it does **not** lose points and the human player receives a **new** random digit. If the machine player plays the random digit, the human player also receives a new random digit. The machine can also see the next two RDs as the human player does. In summary :

- If the machine player plays its random digit, it earns points as described in (4).
- If the machine player skips, it does not lose points. The human player gets a new random digit.

7) After the machine player plays or skips, the human player is given a chance to review the situation and check the players' points by pressing BTNU. Then, the human player presses BTNR to play next. But, if the machine player has an adjacency, the human player presses BTNL to allow the machine player to play. The machine player plays as described in (6) above.

8) The human player has **two** options : play the digit on a display or skip. The human player can press BTNR to see the next two RDs on PD0 and PD1 or BTNR and BTNU to see the discovered code digits. If the human player skips, points is **not** lost. A **new** random digit given to the machine player. Again, we see that :

- If the human player plays its random digit, he/she earns points as described in (4).
- If the human player skips, no points is lost. The machine player gets a new random digit.

9) The game continues in this fashion until one of the players exceeds the points limit, $(255)_{10}$. Then, all four displays blink slowly, the game stops and the player is the winner. The game is restarted from the beginning by pressing BTND (Reset). The game can be reset anytime to restart. When reset, all position displays, players' points and LED lights are zero. The game can be reset even when the game is going on. The machine player **cannot** reset the system.

10) Player points can be seen by pressing BTNU at any time. On the **left** two displays Player 2 points is shown in Positional Hex and on the **right** two displays Player 1 points is shown in Positional Hex. The maximum player points is FF or $(255)_{10}$. The minimum is 0.

Figure 4 gives a number of random digit playing cases, one following another with respect to **time** right after the r**eset**. In this sequence, the machine player is an imaginary one, not the one that is at the course web site.

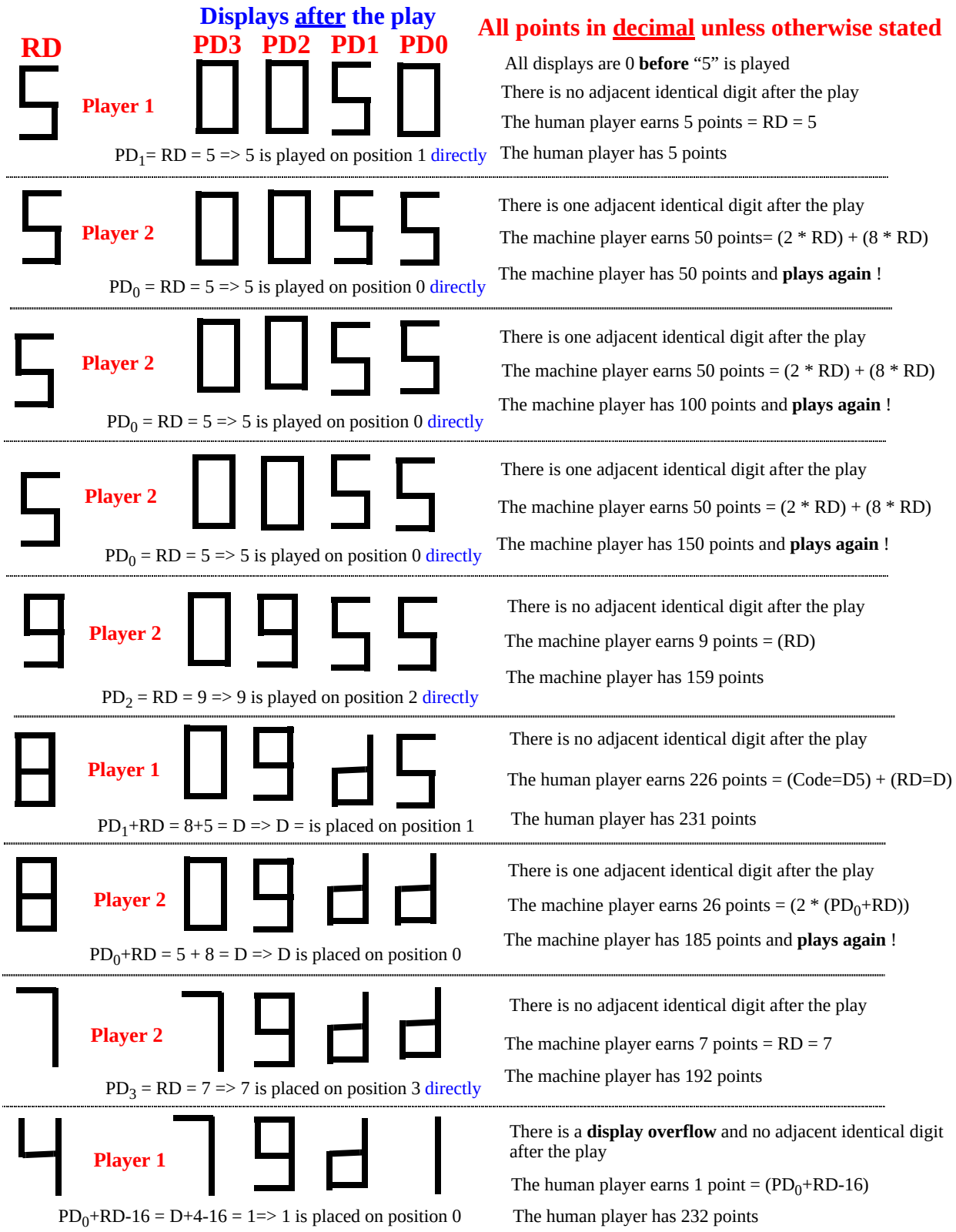


Figure 4. Examples of consecutive digit plays after a reset. Assume that the code is D5

2.2.2. Ppm Playing Modes

The operation diagram of the ppm is shown in Figure 5. The operation diagram shows the operations graphically with respect to time. In order to clearly describe the operations, i.e. which player does what and when, the operation diagram is given in terms of **modes** and **states**. The ppm system is always in a specific mode at any time. There are three modes : the **Reset** mode, the **Player 1 play** mode and the **Player 2 play** mode.

A mode consists of submodes (steps or **states**). A circle is a state in the operation diagram. In each state, the system performs specific operations. Each state takes at least one clock period. To refer to the states easily, numbers are assigned to the states. The assignment is from top to bottom, though, in real life, this is done in a way to reduce the hardware. Each line connects one state to another and is taken if there is a label next to the line and the label is true. For example, if a state has a circled line directed at itself and another line with a label directed at another state, then the system stays in the state until the condition in the label is satisfied. That is, it waits for certain input/event (a push button press/the machine player completes its thinking) indicated by the label to occur to move to the next state. States 0, 1, 3, 4 and 6 are as such. In some other states, there is no label and so a move to another state happens without waiting. Also, certain state transitions change the mode to another mode. In the next section, the operation diagram is used to obtain the major operations and blocks of the ppm. The concept of state is discussed in detail when sequential circuits are covered in class.

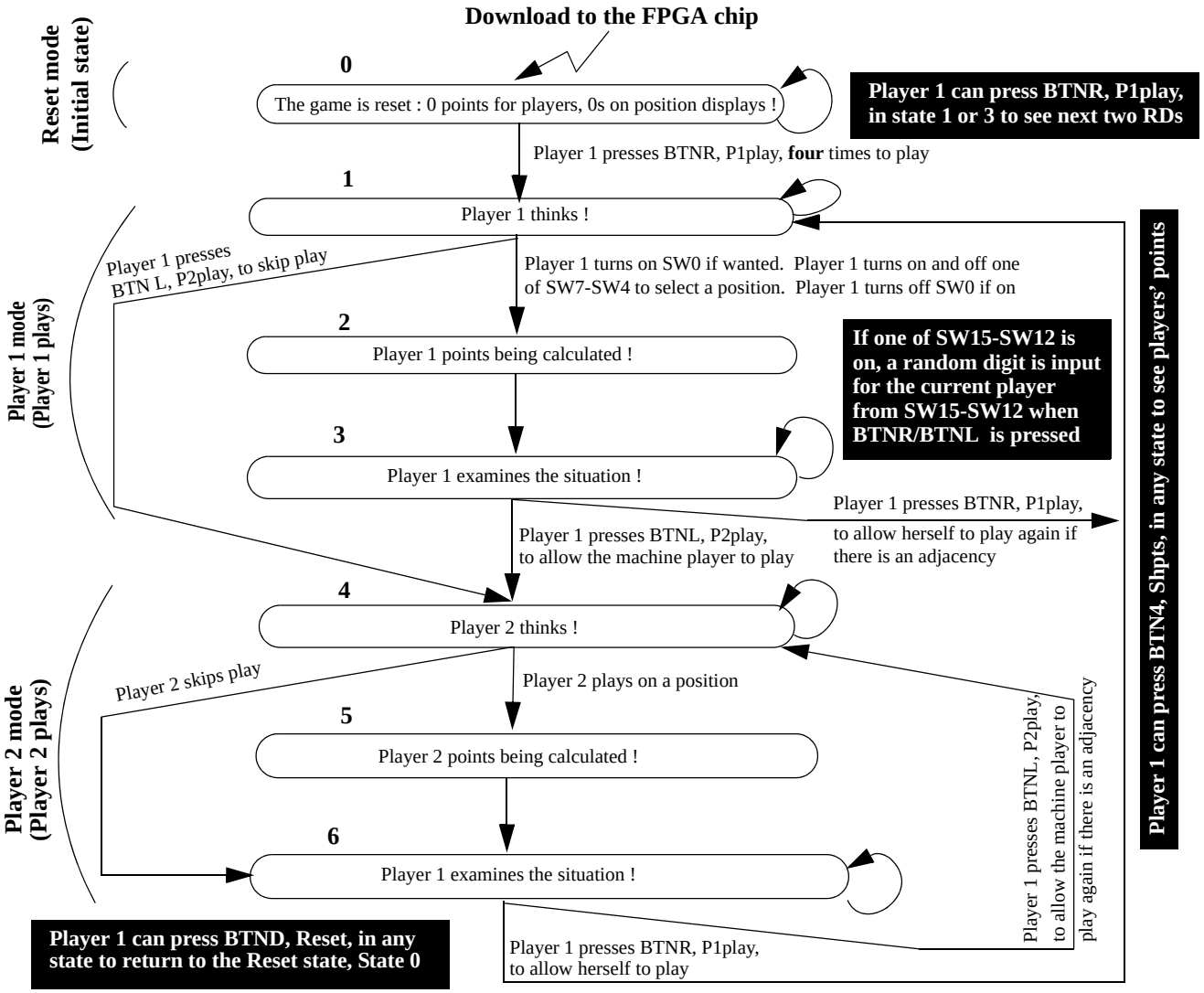


Figure 5. The operation diagram of ppm.

At any moment, the current state number is shown on the rightmost three LED lights : LD2 - LD0 (see Figure 3). Thus, if it is the Reset state, the lights show 0. If the human player is thinking, they show 1, if human player points is

being calculated they show 2 and finally when the human player is examining the situation they show 3. If the machine player is thinking, they show 4, if machine player points is being calculated, they show 5 and when the human player is examining the play again they show 6. These lights are also useful in determining which player has won the game. We know that when a player exceeds $(255)_{10}$, the four displays blink slowly. If BTNU (Shpts) is pressed the points also blink slowly. By looking at the points one cannot determine which player has won the game. If LED lights are checked, they would indicate the winning player : If the lights show 3, the human player has won the game and if they show 6, the machine player has won the game.

2.3. Ppm Black box Partitioning

We know the rules of the ppm game and the black-box view. The ppm has 13 inputs, 19 outputs and is **sequential**. Because of a large number of inputs, it has to be partitioned into blocks. How can we go about partitioning it ? We have experimented with partitioning complex sequential circuits into blocks : We obtain a list of major operations it performs by studying the operation diagram and also consider the design goals and technology available. Then, we create a block for each major operation. However, thinking of these major operations is not simple. It requires considerable logic design experience. Also, the list of major operations is not fixed, two people can come up with two different lists and therefore two different block partitionings.

By studying the ppm operation diagram in Figure 5, we determine the major operations. For example, state 4 indicates Player 2 (the machine player) thinks. This has to be a major operation for which a block is needed ! In state 1, Player 1 thinks. A major operation is needed and so a block to implement this major operation. In states 2 and 5, we determine the new player points. Thus, calculating new player points has to be a major operation and we need to have a block for that. A major operation not explicit in the operation diagram, but needed is interfacing the ppm to the I/O devices on the FPGA board (switches, displays, etc.). So, we need a major operation and so a block for that. Another implied major operation is playing the random digit on a position display and the associated work such as storing position displays and players' points and generating the random digit. We have to have a block for that. Now, we have five blocks !

Will we have just five blocks ? No ! We need a way to make sure the game proceeds as state 0, state 1, state 2, etc. so that the right operation happens at the right time : A **controller** is needed to keep track of the states and indicate what to do when. We will then have one more major operation, the controlling operation and a block, the Control Unit Block. Then, overall, we have six major operations and so six blocks in our ppm digital system (Figure 6).

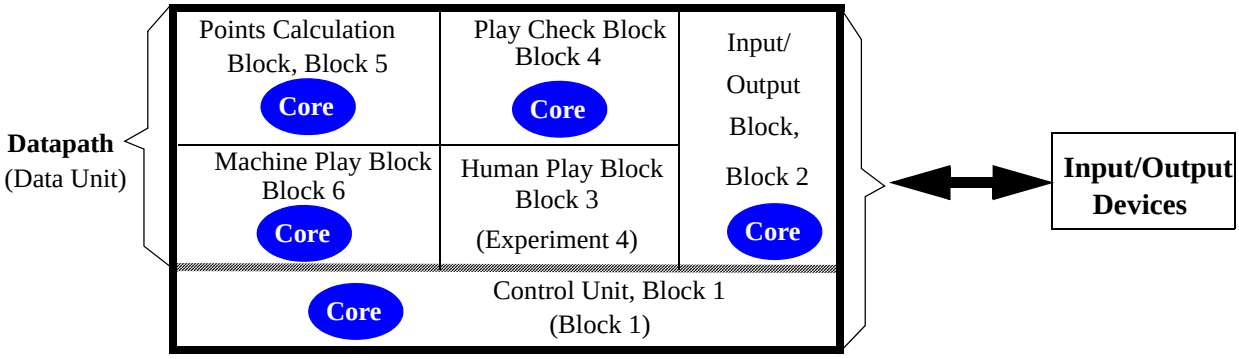


Figure 6. Block partitioning of the ppm.

In general, if a circuit has a Control Unit, then it is a **digital system**, since a digital system performs a series of operations and needs to have a circuit to keep track of operations and determine what is to be done next, based on inputs and past operations. This circuit is the Control Unit. So, **the ppm is a digital system** that plays a game for which it follows the game rules and responds to inputs and past events !

In digital system terminology, all the blocks other than the Control Unit are placed in a super block called **datapath** (Data Unit). Then, the first partitioning of **any** digital system always results in a Control Unit and a Data Unit. The partitioning of the Data Unit follows it and is based on major operations it has to perform. These major operations are the ones we specify above : interacting with input/output devices on the FPGA board, handling human player's plays, handling the play of the random digit, calculating new points and playing for the machine. Below, first we discuss digital systems in general and then the six ppm blocks.

3. Digital Systems Overview

This section presents an overview of digital systems and relates to current practices and tools. It also discusses digital system coverage at New York University.

3.1. Introduction to Digital Systems

A digital system consists of digital circuits. It performs sequences of simple operations (**microoperations**). An example of a digital system is the Central Processing Unit (the **CPU** or processor) of a computer. Every computer has a CPU which is on the microprocessor chip. For example, if one buys a PC with a Pentium 4 microprocessor, the microprocessor chip has the Intel Pentium 4 CPU (in addition to cache memories). A computer is a collection of digital systems ! DVD controllers, calculators are digital systems as well.

The first partitioning of a digital system results in **data** and **control** units (Figure 7). The control unit determines which microoperation happens when. The data unit performs the microoperations. The Control Unit generates *control signals* to determine the timing of microoperations in the datapath. The control unit receives *status signals* from the datapath informing where the data unit stands. The next partitioning of the digital system is on the data unit, dividing it into smaller blocks.

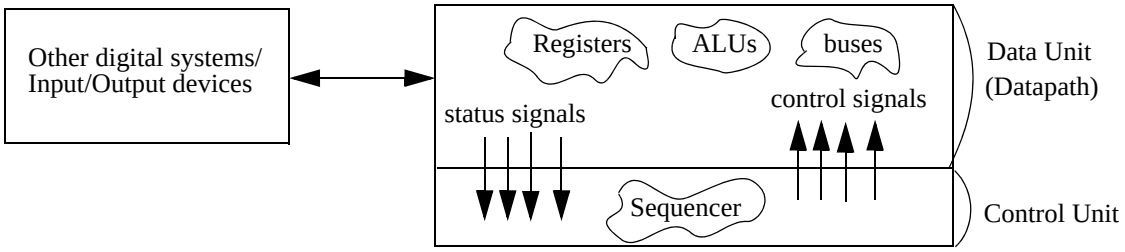


Figure 7. A large scale view of a digital system.

3.2. Digital System Design

Today's digital systems, such as CPUs, have hundreds of millions of transistors. Developing such a digital system starts at a high, abstract level and ends with the transistor circuit layout. This development cycle is automated to handle the complexity. A vast array of tools and techniques are used for effective design to allow engineers to move freely between different levels of abstraction : superblocks, blocks, subblocks, gates/flip-flops, chips and transistors.

There are a number of techniques used to design a digital system, such as the **Finite State Machine** (**FSM**) and **Hardware Description Language** (HDL) techniques and a combination of them. They lend themselves to computer usage for speedy design. Verifying that the design works is done by obtaining correct results from the **simulations** of the design. Such simulations allow designers to complete the prototype quickly and confidently. Thus, we can move from the high-level abstract specification of the machine to its implementation fast.

The FSM technique is often used for educational purposes where **a FSM state diagram describes a digital system**. The FSM technique treats the digital system as a "state machine" going through states based on the sequence of inputs/events. A state diagram with a finite number of states is drawn to show which microoperation happens when and how. In each state, the value of every signal is either known or determinable, allowing to account for every clock period. Some of these signals are, in fact, control signals that the Control Unit generates to control the datapath. The overall goal is to describe the flow of microoperations, the input-output relationship with respect to time precisely. CS2204 uses the FSM technique which is why the name of the course is "Digital Logic and State Machine Design."

The digital system design approach now in use by industry is *Hardware Description Language- (**HDL**-) based*. In this technique, **an HDL program describes a digital system**. The HDL technique is used for such digital systems as CPUs where there is a very large number of states and so using the FSM technique, i.e. working on a state diagram is not efficient. Most commonly used HDLs are **VHDL** and the **Verilog HDL**. VHDL is developed in the early 1980s and its syntax looks like the 1980's software language ADA. The Verilog HDL is developed earlier than the VHDL and looks like the C language. Knowing one of the languages helps learning the other one quickly.

An HDL program has statements (text lines) to describe hardware : The hardware is abstracted in text. The CAD tool converts the HDL text to hardware. Designers work on HDL programs and delay building the prototype until they are sure their HDL programs are correct. HDLs also enable companies to license their circuits as cores (IP) easily, since they do not have to send a PCB nor a chip, but a layout of the design derived from the HDL program.

Whichever method is used, the starting point in the digital system design is the operation diagram. It shows major operations and **implies** microoperations that implement the major operations with respect to time precisely. It shows the response of the digital system for expected and unexpected inputs/events. Each state on the diagram has a meaning, a summary of past events ! A state is arrived based on a sequence of inputs/events up to that point. The system can stay in a state until an input/event takes place after which it moves to another state and performs actions of that state and so on. At other times, the system does not wait for an input/event to move to another state.

If the digital system is simple, such as the ppm, the operation diagram can be immediately converted to a finite state diagram, leading to the FSM technique. Both diagrams would have the same number of states. If the digital system is more complex, each operation diagram state is converted to a number of state diagram states and implement the system. This may not be acceptable for CPU design where the number of states is very high. One would convert operation diagram states to HDL statements to handle the complexity.

3.2.1. The Data Unit

The data unit performs microoperations : additions, subtractions, AND, OR, shift, compare, etc. In order to perform these microoperations, it needs to have three major kinds of hardware : **registers**, Arithmetic Logic Units (**ALU**s) and **buses** (Figure 7). Registers are needed to store data temporarily. ALUs perform additions, subtractions, shifts, ANDs, ORs, etc. Buses transfer information. Buses interconnect registers and ALUs. Registers contain flip-flops to store bits. ALUs are often combinational circuits. Buses are bundles of wires with additional control logic.

The data unit has the bulk of the logic, i.e. the most number of gates and flip-flops. But, it is easier to design than the control unit since the data unit is highly **regular**. The data unit has pieces of hardware repeated. For example, an ADDer is a repetitive set of smaller addition blocks, the multiplier is similar and so are the registers. Another example is a 4-bit comparator that can be repeated 8 times to compare 32-bit numbers. Design, test, manufacture, upgrade of regular hardware is easier. Therefore, the data unit design is straightforward compared with the control unit.

3.2.2. The Control Unit

The control unit controls the data unit. It determines microoperation timing in the data unit : Which microoperation happens when. The control unit may seem to perform rather an easy job, just controlling the data unit. It could indeed be so if we used ideal components : no gate delays, no power consumption and fan-in restrictions, etc. However, the reality is the opposite. For example, a circuit that logically works may not work due to a **glitch** created by gate delays. And when one thinks of the data unit of a real digital system with many blocks of circuits, the enormous complexity of determining the timing, locally and globally becomes clear.

The control unit has status signals as inputs and control signals as outputs (Figure 8). The hardware of the control unit is called **sequencer**, since it is the one that determines the sequence of microoperations. It is the sequencer that goes through the states. The sequencer can “jump over” states also. It knows exactly which step it is in at any moment by using a state register. Based on the state register and status signals, control signals are generated for the data unit and the next state value for the state register. The sequencer is often an **irregular** circuit whose design is complex. The irregularity is due to the large amount of **random logic** (gate networks and flip-flops).

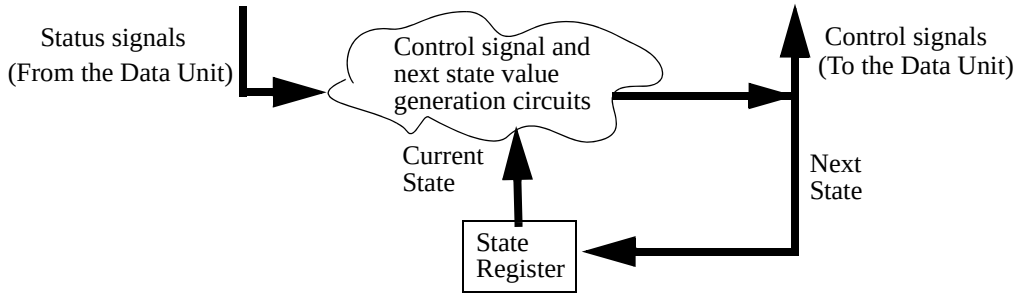


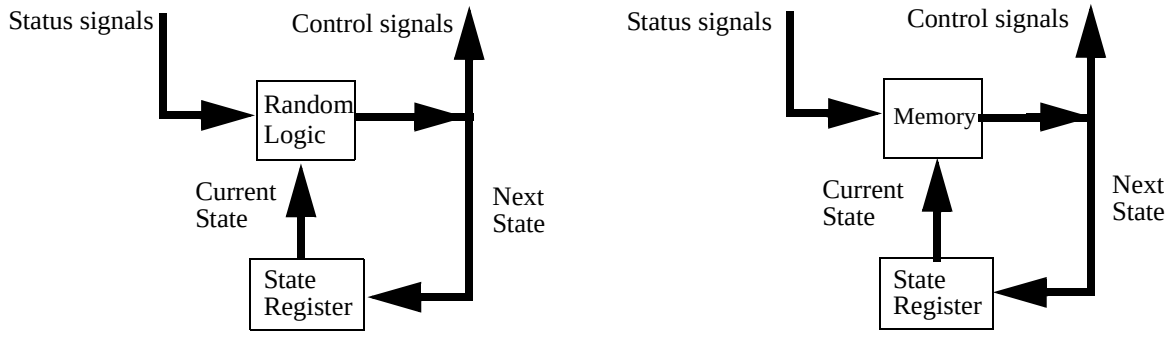
Figure 8. A general view of the sequencer.

3.2.3. Control Unit Design

Today, the sequencer is implemented by using **hardwiring** or **microprogramming** or a **mixture** of both. Whether it is hardwiring or microprogramming or a mixture, or the FSM or the HDL technique, the first task in the design of a sequencer is to lay out the input-output relationship of the sequencer, i.e. based on the status signals and the current

state, values of control signals and the next state are shown at any moment. This input-output relationship must be precise, which can be an exhaustive job. Also, the control unit has to be free from timing problems itself (glitch-free), and also make sure that the data unit is free from timing problems. For example, microoperations should not start too early nor too late. To generate such signals, the Control Unit has to be implemented so that gate delays are accounted for. Thus, a control unit is partitioned only once to Control Signal Generation and Next State Generation Subblocks to make sure delays are under check during the design.

Hardwiring generates control signals and next state values by using gates and flip-flops. For each signal, there are gates and flip-flops interconnected in a random looking way, Figure 9(a), hence **random logic**. Hardwiring leads to a large number of gates and flip-flops and so their development is time consuming. The design, test, modification, manufacture and upgrade of a large amount of random logic is monumentally difficult. Since the ppm digital system is not too complex, its control unit is implemented by hardwiring. One can see schematic 1 of the ppm project where gates and flip-flops are used to generate the control signals and next state values.



a. A general view of a hardwired sequencer. **b. A general view of a microprogrammed sequencer.**

Figure 9. Digital system sequencers.

A microprogrammed sequencer generates control signals and next state values by storing them in a special memory in the control unit with little additional random logic, Figure 9(b). Thus, microprogramming simplifies the control unit development. However, microprogrammed control units are **slower** than hardwired control units. Some digital systems today mix hardwiring and microprogramming in their control unit to have a higher speed and less complexity.

3.3. Digital System Design at Tandon

Digital system design is substantially emphasized in academia. For example, at New York University, fundamentals of digital system design are distributed to CS2204 and CS2214 (Computer Architecture and Organization). CS2214 describes digital system design by focusing on how to design a small computer, including a CPU on a microprocessor. It covers both control and data units. While Data Unit design is equally covered in CS2204 and CS2214, Control Unit design is heavily emphasized in CS2214.

These two courses and others in CIS and ECE departments at Tandon, such as CS6143, EE4313, EL5493, introduce additional techniques and tools for handling the complexity : they allow engineers to move freely between different levels of abstraction. The FSM technique is used in CS2214. EE4313 and EL 5493 introduce VHDL.

4. The Ppm Digital System

This section presents the ppm digital system in a top-down project development style where initial design steps involve simple concepts with few details. Targets are large blocks, not gates and flip-flops. This is because students design a digital system. Since a digital system is complex, designing it all at once is prohibitive as there are many details to deal with at any moment. While having a part of the digital system (the control unit) control the rest of the digital system (datapath) simplifies the design, using the top-down design technique simplifies it further.

Students are cautioned to proceed in even smaller steps when they design the blocks. Implementing an entire block at once can result in many errors. They are suggested that they study blocks and as blocks become clearer partition a block into subblocks, and subsubblocks and so on. Then, design and test each (sub)block separately. Then piece by piece, attach tested (sub)blocks together. It is left to students which (sub)block to be designed after which, which (sub)blocks to be attached to which for a larger test and finally how to eliminate redundant circuits in case two or more (sub)blocks have the same circuit. Eventually, gates and flip-flops are considered for the implementation. As

mentioned above, the ppm consists of six blocks. These blocks are fairly complex and so must be divided into subblocks, subsubblocks and so on. Thus, overall designing the digital system in smaller pieces, piece by piece and systematically improves the progress of the design.

Finally, CS2204 experiments incorporate the core-based design technique. Due to time-to-market constraints, hardware engineers "license" a part of the circuit, a block, called the "core" from another company under a license agreement. The core may be modified slightly if necessary, then the remaining parts of the circuit are designed and attached to the core. Engineers save time since they design and attach only the remaining parts of the circuit. The name is "core" because the other simpler, blocks seem to "surround" it. In CS2204, students do not design all ppm blocks. They are given core circuits and develop the remaining circuits. The core circuits are blocks 1, 2, 4, 5 and 6. Thus, the description in this section includes the partitioning of the ppm digital system into blocks and subblocks and the incorporation of the non-core blocks to the core blocks. Students are strongly advised that they use the *General Design Rules* and *Digital Product Development* Handouts during the ppm development.

4.1. Ppm Control Unit

The ppm Control Unit block design uses the **Finite State Machine** technique mentioned above. The block requires considerable attention : Its design must ensure that the ppm operation diagram is conformed with all the time. The operation diagram is useful as it allows us to concentrate on simpler concepts in the initial phases of top-down design. Each state in the operation diagram describes a major operation and implies what happens in the data unit at the moment and what the control unit has to do.

The FSM technique requires that we derive a FSM **state diagram**, describing both the Control Unit and the Data Unit, detailing the values of *control signals* precisely at any moment. We obtain the FSM diagram from the operation diagram. The ppm state diagram states are identical to the operation diagram states since the ppm is a simple system. The state diagram shows the same information, but is more detailed. On the state diagram, each circle is again a state that takes at least one clock period which we will call the system clock period (the Sysclk period). A line indicates a transition from one state to another. State changes happen automatically or are triggered by inputs/events.

Two types of state diagrams are obtained from the operation diagram : The high-level state diagram and the low-level state diagram. The high-level state diagram is written in terms of microoperations, leading to the datapath. After determining the datapath, the high-level state diagram is converted to a low-level state diagram where each state is specified in terms of **active** control signals. This way, the control unit knows which control signals should be active and so which microoperation(s) have to be performed. Both the operation diagram and FSM state diagrams of ppm are based on the **Mealy machine** model where the outputs (of the control unit) in any state are a function of the current inputs and the current state. These diagrams are described in detail in lab sessions. Also, interested readers are referred to Chapters 7 and 8 of the textbook. As an exercise, students can develop the ppm state diagram to show which control signal is active in which state, by studying the ppm Control Unit circuit (schematic 1).

4.1.1. Ppm States

As mentioned above, either automatically or because of an input/event the digital system moves from one state to another state and performs the operations of that state. In order to derive the states of the operation diagram, we carefully determine all the signals that trigger responses from the digital system. Then, determine the operations in each state conforming to the input/output behavior of the circuit, i.e. the input/output relationship. Table 3 lists the seven states of the ppm digital system and relate to the operation diagram given in Figure 5.

State	Description
Reset	Reset to the downloading-to-the-FPGA state. All position displays are 0. All LED lights are blank. Both players' points are zero. The move to the next state, P1Think, happens when BTND is pressed four times in a row
P1Think	Player 1 takes time to think and then selects a position to play. Player 1 can also skip the play. Player 1 plays right after downloading or reset or after Player 2 plays without an adjacency. If BTNR is pressed next two RDs are shown on the rightmost two displays. If BTNU is pressed, current players' points are shown. If BTNR and BTNU are pressed, machine player discovered code digits are shown
P1Calculation	The digit played is stored on the selected position display. New Player 1 points is calculated and stored. If Player 1 exceeds the maximum points limit, the position displays blink slowly and the game ends

Table 3: ppm states.

State	Description
P1Examine1	Player 1 is given time to examine the situation, such as checking the points by pressing BTNU or the next two random digits by pressing BTNR. If BTNR and BTNU are pressed, machine player discovered code digits are shown. In addition, for testing purposes, Player 1 can turn on one of SW7-SW4 to input a random digit to Player 2. The random digit is obtained from SW3-SW0
P2Think	Player 2 takes time to think and then selects a position to play. Player 2 can also skip the play. Player 2 plays after Player 1 plays without an adjacency or skips. If BTNU is pressed, current players' points are shown
P2Calculation	The digit played is stored on the selected position display. New Player 2 points is calculated and stored. If Player 2 exceeds the maximum points limit, the position displays blink slowly and the game ends
P1Examine2	Player 1 is given time to examine the situation, such as checking the points by pressing BTN4. If BTNR and BTNU are pressed, machine player discovered code digits are shown

Table 3: ppm states.

4.1.2. Ppm Control Unit Implementation

The control unit of the ppm has 20 inputs and 23 outputs (Figure 10). 11 of the 20 inputs are status signals. 20 of the 23 outputs the Control Unit generates are control signals. The fact that the number of inputs is close to the number of outputs is a coincidence. On Tables 4 and 5, we list the inputs and outputs, respectively. The Control Unit of the ppm is implemented by using **hardwiring** ! Block 1 is a core block and so has already been designed.

Figure 10. Inputs and outputs of the ppm control unit.

Input name	Number of lines	The source	Definition
P1playsynch	1	Block 2	Synchronized Player 1 play derived from BTNR. If it is 1 in state 0 four times in a row, it means Player 1 wants to play after downloading or reset. If it is 1 in state 1 or 3, it means Player 1 wants to see the next two RDs. If it is 1 in state 6, it means Player 1 has examined the play and wants to play. If it is 1 together with Shpts in states 1, 3 and 6, it means Player 1 wants to see the machine discovered code digits
P1played	1	Block 3	Player 1 has played. When it is 1, it means Player 1 has turned on one of SW7 - SW4
P1skip	1	Block 3	Player 1 skips the play. It is 1 when P2playsynch is 1. P2playsynch is 1 when the human player presses push button BTNL, P2play. P1skip is separated from P2playsynch so that in the future another machine player can be attached as Player 1 instead of the human player
P2playsynch	1	Block 2	Synchronized Player 2 play signal derived from BTNL. Player 1 signals Player 2 to play. When BTNL is pressed in state 1, it signals Player 1 wants to skip the play. When BTNL is pressed in state 3, it means Player 1 has examined the play and wants the machine player to play
P2played	1	Block 6	Player 2 has played. When it is 1, it means Player 2 has selected a position to play
P2skip	1	Block 6	Player 2 skips the play. When it is 1, it indicates Player 2 skips the play
P1add	1	Block 2	Player 1 wants to add. It is 1 when the human player turns on SW0 to add the random digit to a display
P2add	1	Block 6	Player 2 wants to add. When it is 1, it signals the machine player adds the random digit to a display

Table 4: Inputs of the ppm Control Unit.

Input name	Number of lines	The source	Definition
Lptovf	1	Block 2	Latched points overflow. It is the latched (stored) value of the Ptovf signal. When Ptovf is 1, it means the current player points has exceeded $(255)_{10}$. The Ptovf signal is latched in states 2 and 5 when the current player points is calculated
Shpts	1	Block 2	Show players' points. When BTNU is pressed it becomes 1, meaning the human player wants to see the players' points. If it is 1 together with P1playsynch in states 1, 3 and 6, it means Player 1 wants to see the machine discovered code digits
Lpdprd	1	Block 2	The latched display overflow signal that is generated when a position is added the random digit. If Pdprd is 1, it means the current play generated a display overflow. It is latched is states 2 and 5
NSD	2	Block 5	Number of similar digits. It gives the adjacency of the position played. It is in Unsigned Binary
PSEL	4	Block 4	Position Select signals. They show which position is selected by the current player
Sysclk	1	Block 2	The system clock at 6 Hz. It is used to time and synchronize events in the digital system
P2clk	1	Block 2	The Player 2 clock signal at 48 Hz. It is used in the debounce circuit of control signal Clrrdregs
Reset	1	Block 2	Reset ppm. When BTND is pressed, it becomes 1 and resets the ppm system

Table 4: Inputs of the ppm Control Unit.

Output name	Number of lines	The Destination	Definition
Clear	1	Block 2, 4, 6	Clear all registers, counters and flip-flops. It is used to store zero on all registers, counters and flip-flops when the system is reset. It is active high and active in state 0
P2sturn	1	Block 6	It is Player 2's turn to play. It is active in state 4
STDISP	4	Block 4	Store on a position display. They are active high and active in states 2 and 5. Only one of them is 1, indicating the position played
Selplyr	1	Block 4, 5	Select a player for play. It is 1 to select Player 2 in states 4, 5 and 6, and 0 in states 0, 1, 2 and 3
Clearp2ffs	1	Block 6	Clear Player 2 flip-flops. It clears Player 2 flip-flops and registers to prepare for the next play by the machine player. It is active in state 1 and when it is state 6 with an adjacency
DISPSEL	2	Block 2	Select a value for the displays. Its four combination determine what to show on displays. If 00 in all states, then position displays, 01 in all states then players' points, 10 in states 1 and 3 then next two random digits on the rightmost two displays and 11 in states 1, 3 and 6 then the discovered code digits on the rightmost two displays
Clrrdregs	1	Block 4	Clear random digit registers. Three registers that keep the three random digits are cleared at reset
Add	1	Block 2, 4	Add. It is P1add in state 1 when the human player plays by turning on SW0 and P2add when the machine player plays in state 4. If it is 1 in states 2 and 3 or in states 5 and 6, it means a addition was done by the human or machine player, respectively
Stp1pt	1	Block 2, 4	Store Player 1 points. It is active high and active in state 2
Stp2pt	1	Block 2, 4, 6	Store Player 2 points. It is active high and active in state 5
Grd	1	Block 4	Get a new random digit. It signals a new random digit is to be generated for the next player. It is active-high and active in states 0, 1, 3 and 6

Table 5: Outputs of the ppm Control Unit.

Output name	Number of lines	The Destination	Definition
Getcode	1	Block 4	Get the code for the game. It signals a new code is to be generated for the game. It is active-high and active in state 0 after Player 1 presses BTNR four times
Calcpts	1	Block 2, 4	Calculate the code reward points. It is active and 1 in states 2 and 4, indicating to the Code Reward Calculation circuits to output the code reward for the current player
Bpds	1	Block 2	Blink position display slowly. It is active high. When it is 1, it blinks slowly (at 1.5 Hz) the position display played, provided that the game is not over, there is no display overflow and also Shpts is not pressed. It blinks slowly all the displays if the game is over. The states that can have this signal 1 are 2, 3, 5 and 6
Bpdf	1	Block 2	Blink position display fast. It is active high. When it is 1, it blinks fast (at 6 Hz) the position display played because of a display overflow, provided that the game is not over and Shpts is not pressed. The states that can have this signal 1 are 2, 3, 5 and 6
Clff	1	Block 2	Clear flip-flops. It is used to store zeros on latches in Block 2
STR	3	Block 2	The state register output. It shows the current state the digital system is in. It is output to the rightmost three LED displays via Block 2

Table 5: Outputs of the ppm Control Unit.

The Control Unit has storage circuits to store past events. One of them, Add, remembers if an addition has been performed by the current player. This output, Add, is shown on LD5 to give feedback to the human player. Another circuit stores the number of times BTNR is pressed to create three random digits and the code after reset. Other circuits store if the current player has an adjacency so that the Control Unit gives another turn to the current player.

4.2. *P*pm Data Unit

This section describes the core Data Unit blocks of the ppm which are Block 2, 4, 5 and 6. Block 3 is assigned to the students as a part of their term project. It also describes how the non-core block, Block 3, is "interfaced" to the core blocks.

4.2.1. Ppm Core Data Unit Blocks, the Input/Output Block, Block 2

Block 2, the Input/Output Block is responsible for receiving inputs from the switches and push buttons, outputting to the LED lights and the 7-segment displays and generating timing signals. It has 85 inputs and 39 outputs (Figure 11). It interacts with all the blocks. On Table 6, we list the inputs and on Table 7 we list the outputs of Block 2. Block 2 is partitioned into three subblocks : I/O Buffer Subblock, Timing Subblock, and Display Subblock. Block 2 is a core block and so has already been designed.

The I/O Buffer Subblock contains input and output buffers for the FPGA pins. It also has two Global Clock Buffers for reliable distribution of two signals used for clocking across the FPGA chip. One is the 100MHz Clock signal. The other is P1play buffered for reliable clocking of a flip-flop. The global buffers ensure the signals are received without delay, known as **skew**, by all the areas on the FPGA chip. Students are referred to the Xilinx FPGA data sheets for a detailed description of the global clock buffer.

The Timing Subblock consists of three subsubblocks. The Clock Signal Generation Subsublock contains a "*frequency divider*" which is a series of counters to divide the 100 MHz FPGA board clock to low frequencies. The following outputs are used in the system : A 192 Hz signal (Q0) used as the random digit generation clock (Rdclk) and to multiplex digits on 7-segment displays, a 48 Hz signal (Q2) used as the Player 2 clock (P2clk), a 6 Hz signal (Q5) used as the system clock (Sysclk) and also to blink a display fast when there is a display overflow, a 1.5 Hz signal (Q7) to blink the display just played at a slow rate and to blink all four displays slowly if there is a points overflow (a player has won the game). Q7 is also used in Block 4 to generate the rightmost bit of the random digit. The Sysclk, P2clk and Rdclk signals are distributed throughout the chip after they are buffered by Global Clock Buffers in this subblock.

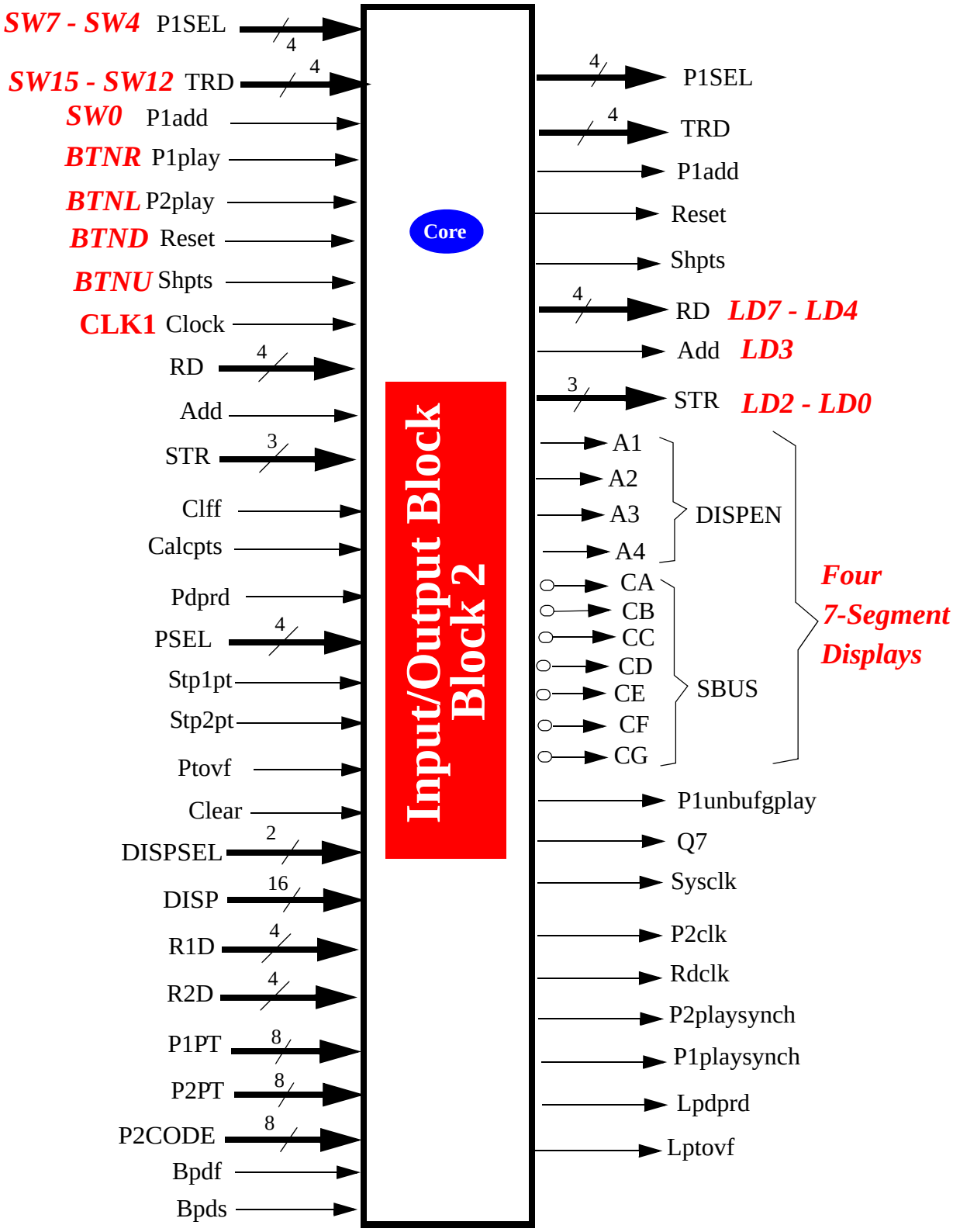


Figure 11. The input and output signals of the Input/Output Block.

Signal name	Number of lines	Description
P1SEL	4	Player 1 position display select. From SW7-SW4 on the FPGA board. Each switch selects a position display to play. If one these switches is 1 when BTNL is pressed, the value represented by SW3-SW0 is input as RD to the machine player
TRD	4	Thefour bits of RD used to debug players. They are labelled as TR3-TRD0. Input from SW15-SW12
P1add	1	Player 1 add. From SW0 on the board. The human player turns it on to add RD to a position display. It is also used as the least significant bit of RD input to the machine player manually. Finally, if it is 1 when Shpts is 1, discovered code digits are shown on the rightmost two displays
P1play	1	Player 1 wants to play. Input from BTNR. When it is pressed in state 0, it means Player 1 wants to play after reset. When pressed in state 6, it means Player 1 has examined the play and wants to play
P2play	1	Player 2 can play. From BTNL. When pressed, it means the human player wants the machine to play. If the human player presses it after playing on a position in state 3, it means the human player has ended the checking of the play and wants the machine player to play. If the human player wants to skip the play, it is pressed in state 1, so it is machine's turn to play
Reset	1	Reset. Input from BTND. When pressed by the human player, the whole digital system is reset where the displays, lights and registers are cleared. It is as if the FPGA chip has just been downloaded the bit file. It can be pressed anytime
Shpts	1	Show players' points. Input from BTNU. When pressed, the Player 1 and Player 2 points are shown on the four displays. The Player 2 points is shown on the leftmost two displays in Hex coding and the Player 1 points is shown on the rightmost two displays in Hex coding. If it is 1 when P1play is 1, discovered code digits are shown on the rightmost two displays
Clock	1	The clock signal. It is a 100 MHz clock signal generated by an FPGA board circuit
RD	4	The random digit of the current player. It is in BCD. From the Play Check Block
Add	1	Add. It is P1add in state 1 when Player 1 plays by turning on SW0 and P2add when Player 1 plays in state 4. If it is 1 in states 2 and 3 or in states 5 and 6, it means a addition was done by the human or machine player, respectively. From Block 1
STR	3	The state register. It shows the current state the digital system is in. From Block 1
Clff	1	Clear flip-flops. It is used to store zeros on latches. From Block 1
Calcpts	1	Store signals. It is active high and active in states 2 and 5. Block 2 uses it to store Pdprd and PSEL signals. From Block 1
Pdprd	1	The display overflow signal that is generated when a position is added the random digit. If Pdprd is 1, it means the current play generated a display overflow. From Block 4
PSEL	4	Position Select signals. They show which position is selected by the current player. From Block 4
Stp1pt	1	Store Player 1 points. When it is 1, player 1 points is stored in state 2. From Block 1
Stp2pt	1	Store Player 2 points. When it is 1, player 2 points is stored in state 5. From Block 1
Ptovf	1	Points overflow. If it is 1, the current player points is above $(255)_{10}$. From Block 5
Clear	1	Clear all registers, counters and flip-flops. It is used to store zero on all registers, counters and flip-flops when the system is reset. It is active high and active in state 0. From Block 1
DISPSEL	2	Select a value for the displays. Its four combination determine what to show on displays. If 00 in all states, then position displays, 01 in all states then players' points, 10 in states 1 and 3 then next two random digits on the rightmost two displays and 11 in states 1, 3 and 6 then the discovered code digits on the rightmost two displays. From Block 1
DISP	16	Position display bits. The four position display digits in Unsigned Binary. From Block 4
R1D	4	Next random digit. From Block 4

Table 6: Inputs of the Input/Output Block.

Signal name	Number of lines	Description
R2D	4	The random digit after the next random digit. From Block 4
P1PT	8	Player 1 points. The Player 1 points in Unsigned binary. From Block 4
P2PT	8	Player 2 points. The Player 2 points in Unsigned binary. From Block 4
P2CODE	8	Code digits discovered by the machine player. Two 4-bit Hex coded digits. From Block 6
Bpdf	1	Blink position display fast. If it is 1, the position display played blinks fast because of a display overflow, provided that there is no points overflow and Shpts is not pressed. The states that have this signal 1 are 2, 3, 5 and 6. From Block 1
Bpds	1	Blink position display slowly. When it is 1, it blinks slowly the position display played, if there is no point limit exceeded, no display overflow and also Shpts is not pressed. It is 1 in states 2, 3, 5 and 6. From Block 1

Table 6: Inputs of the Input/Output Block.

Signal name	Number of lines	Description
P1SEL	4	Player 1 position select. Each bit selects a position to play a digit. It is used by the human player. To Block 3 and 4
TRD	4	The four bits of RD used to debug the machine player. They are labelled as TR3-TRD0. To Block 4
P1add	1	Player 1 wants to add. It is 1 when the human player turns on SW0 which signals the player wants to add the random digit from a display. It is also used as the least significant bit of RD input to the machine player manually. To Block 1 and 4
Reset	1	Reset the digital system. When active at 1, it returns the state to state 0. To Block 1
Shpts	1	Show players' points. When it is 1, it means the human player wants to see the players' points. To Block 1
RD	4	Random Digit. It is in BCD and is the random digit of the current player. To LED lights LD7 - LD4
Add	1	Add. It is P1add in state 1 when the human player plays by turning on SW0 and P2add when the machine player plays in state 4. If it is 1 in states 2 and 3 or in states 5 and 6, it means a addition was done by the human or machine player, respectively. It is activated in these states if the game is not over. To LD3 on the FPGA board
STR	3	State register. It shows the state the digital system is in. The value of STR is between 0 and 6. To LED lights LD2 - LD0
CA - CG & A1 - A4	11	The 7-segment display signals. By default, they show position displays on which digits are played. Otherwise, players' points on all displays or next two random digits or discovered code digits are shown on the rightmost two displays
P1unbufgplay	1	The P1play signal not buffered by the global buffer. To Block 1
Q7	1	Clock signal at 1.5 Hz. To Block 4
Sysclk	1	The system clock at 6 Hz. It is used to synchronize operations. To Block 1, 4 and 6
P2clk	1	The Player 2 clock signal at 48 Hz. To Block 2 and 6
Rdclk	1	The random digit generation clock at 192 Hz. It is the random digit generation counter clock. To Block 4
P2playsynch	1	Synchronized Player 2 play. Player 1 signals Player 2 to play. When BTNR is pressed in state 1, Player 1 wants to skip the play. When pressed in state 3, it means Player 1 has examined the play. To Block 1 and 3
P1playsynch	1	Synchronized Player 1 play signal. Player 1 wants to play. When BTNR is pressed in state 0, it means Player 1 wants to play after reset. When pressed in state 6, it means Player 1 has examined the play and wants to play. To Block 1

Table 7: Outputs of the Input/Output Block.

Signal name	Number of lines	Description
Lpdprd	1	Latched position display plus random digit. If Pdprd is 1, it means the current play generated a display overflow. It is latched is states 2 and 5 when the current player points is calculated. To Block 1
Lptovf	1	Latched points overflow. It is the latched Ptovf. When Ptovf is 1, it means the current player points has exceeded $(255)_{10}$. Ptovf is latched in states 2 and 5 when the current player points is calculated. To Block 1

Table 7: Outputs of the Input/Output Block.

The Debouncing Subsubblock of the Timing Subblock contains circuits to prevent the debouncing of push buttons 1 and 2. A push button is an electromechanical device and has an unwanted feature : **bouncing**. Bouncing means when we press a push button, more than one contact is made unintentionally and so multiple "*pulses*" are generated. That is, the signal becomes 1, 0, 1, 0,... until finally stabilizing at 1. It stays at 1, until the button is released. Since when we press P2play, we want to signal to Player 2 to play only once, we have to prevent the push button from bouncing. The P2playsynch **debouncing** circuit does that. It has a flip-flop network so that the Control Unit would function correctly. There is a similar debouncing circuit for P1play to generate P1playsynch.

The Latching Subsubblock has a circuit to latch (store) Pdprd for the Control Unit. The latched signal, Lpdprd, is used to generate the Bpdf signal when a display exceeds $(15)_{10}$. It is latched in states 2 and 5 and cleared in states 0 (after reset), 1 (after Player 1 skips), 3 (after Player 1 plays) and 6 (after Player 2 plays). The PSEL lines are latched also in states 2 and 5 to be used by the Play Check Block in states 3 and 6. The LPSEL signals are cleared similarly. The last latching circuit latches the Ptovf signal in state 2 (after Player 1 plays) and in state 5 (after Player 2 plays) to generate Lptovf. It is used by the Control Unit in states 3 and 6. It is also used by the Input/Output Block to blink the displays when there is a points overflow in states 3 and 6. The flip-flop is cleared when the system is reset.

The Display Subblock is composed of four subsubblocks. One of them, the 7-Segment Digit Content Selection Subsubblock, multiplexes among position displays, next two RDs, players' points and the discovered code digits. If BTNR and BTNU are not pressed, it shows position displays. If BTNR is pressed in state 1 the next two RDs are shown on the rightmost two displays and the leftmost two displays show PD2 and PD3. When BTNU is pressed, Shpts is 1, and it selects players' points so that the displays show players' points in Positional Hex : Player 2 points on the leftmost two displays and Player 1 points on the rightmost two displays. If BTNR and BTNU are pressed at the same time, the discovered code digits are shown on the leftmost two displays. The four-digit output of this subsubblock is connected to the 7-Segment Digit Selection Subsubblock.

The 7-Segment Digit Selection Subsubblock is responsible for selecting one digit out of four digits in a round-robin fashion as explained in the *Advanced Xilinx and Digilent Features* Handout. The FPGA chip has only 11 pins assigned for the 7-segment displays, not 32 pins needed for the four displays. One needs to output digits one at a time in a round robin fashion to give the impression that all four displays are shown all the time. The digit is selected by using MC1 and MC0 signals generated by the 7-Segment Digit Control Subsubblock. The single digit output from this subsubblock is connected to the Hex-to-7-Segment Converter Subsubblock

The Hex-to-7-Segment Converter Subsubblock is responsible for converting the single Hex digit output to the 7-Segment format. This is needed to turn on and off individual segments of each display to show Hex digits. The subsubblock implements the conversions by means of seven **Read-Only Memory** (**ROM**) blocks. Students are referred to the *Advanced Xilinx and Digilent Features* Handout for a detailed description of how to use ROMs to implement functions. Later in the semester students are shown how to use ROMs to implement gate networks in general.

The 7-Segment Digit Control Subsubblock generates multiplexer select signals MC1 and MC0. They are used in the 7-Segment Digit Selection Subsubblock to select one of four digits. The rate of selecting one digit out of four is 192 Hz (Q0). The subsubblock is also responsible for blinking the position played at 1.5 Hz (Q7), which signals to the human player that the digit has been successfully played on that position. The subsubblock blinks faster a position display if there is a display overflow at the rate of 6 Hz (Q5), signalling the situation. Finally, the subsubblock blinks all four displays when the points limit is exceeded (Lptovf) at the rate of 1.5 Hz (Q7).

4.2.2. *The Ppm Non-Core Data Unit Block, the Human Play Block, Block 3*

Block 3 completes the human play. The completion is either to signal that RD has been played (P1played = 1) or the play has been skipped (P1skip = 1). Block 3 has five inputs and two outputs (Figure 12). Tables 8 and 9 describe the inputs and outputs of this block, respectively. Block 3 is a non-core block, but has been designed as a human palyer.

Block 3 is a simple block with two circuits that generate the two outputs to indicate that the human player has concluded the play. One output is P1played. It is 1 if the human player plays the random digit which happens when the player selects a position by turning on one of SW7 - SW4. The circuit checks if two or more switches are turned on. If yes, P1played stays at 0. A MUX circuit implements the gate network to check for the number of switches turned on. Students are referred to the classroom discussion on how to use a MUX to implement a gate network. The other output is the P1skip which is 1 if the human player does not want to play RD : The human player wants to skip. P1skip is determined by the P2playsynch signal derived from BTNL. A buffer is used to rename the P2playsynch signal as P1skip. Note that only one of P1skip and P1played can be 1 at a time. They cannot be 1 at the same time.

Figure 12. The detailed view of the input and output signals of the Human Play Block.

Signal name	Number of lines	Description
P1SEL	4	Player 1 position select signals. One bit per position indicates the position selected by the human player when that bit is 1. From SW7-SW4 via the Input/Output Block
P2playsynch	1	Synchronized version of P2play. When the P2play signal from BTNL, synchronized to reduce bouncing, becomes 1, it means Player 1 wants Player 2 to play. It is normally 0. When BTNL is pressed in state 1, it means Player 1 wants to skip the play. When pressed in state 3, it means Player 1 has examined the play. From Block 2

Table 8: Inputs of the Human Play Block.

Signal name	Number of lines	Description
P1played	1	Player 1 has played. When it is 1, it means Player 1 switched on one of SW7 - SW4. P1played is 0 if two or more switches are turned on. To Block 1
P1skip	1	Player 1 skips the play. It is 1 when P2playsynch is 1. To Block 1

Table 9: Outputs of the Human Play Block.

Block 3 is a simple block with two circuits that generate two outputs to indicate that the human player has concluded the play. One output is P1played. It is 1 if the human player plays the random digit which happens when the player selects a position by turning on one of SW7 - SW4. The circuit checks if two or more switches are turned on. If yes, P1played stays at 0. A MUX circuit implements the gate network to check for the number of switches turned on. Students are referred to the classroom discussion on how to use a MUX to implement a gate network. The other output is the P1skip which is 1 if the human player does not want to play RD : the human player wants to skip. P1skip is determined by the P2playsynch signal derived from BTNL. P1skip is separated from P2playsynch so that in the future another machine player can be attached as Player 1 in place of the human player. A buffer is used to rename the P2playsynch signal as P1skip. Note that only one of P1skip and P1played can be 1 at a time. They cannot be 1 at the same time.

Is this block needed ? Its two circuits can be a part of Block 2 or Block 4 ! However, recalling our discussion on *General Design Rules*, we realize we need to have additional room for future upgrades. Having a separate block for the human player, allows us to have the machine vs. machine version easily in the future. Similarly, one can replace the Machine Play Block, Block 6, with a Human Play Block to have two human players to play against each other.

4.2.3. The Ppm Core Data Unit Block, the Play Check Block, Block 4

Block 4 controls the displays and operations on them. It has sequential circuits to keep players' points and generate the random digit. It multiplexes signals between the two players. It generates results in case the random digit is added to the four displays. **Both players share Block 4**. It has 37 inputs and 63 outputs (Figure 13). Tables 10 and 11 list the inputs and outputs of Block 4, respectively. Block 4 is a core block and so has already been designed.

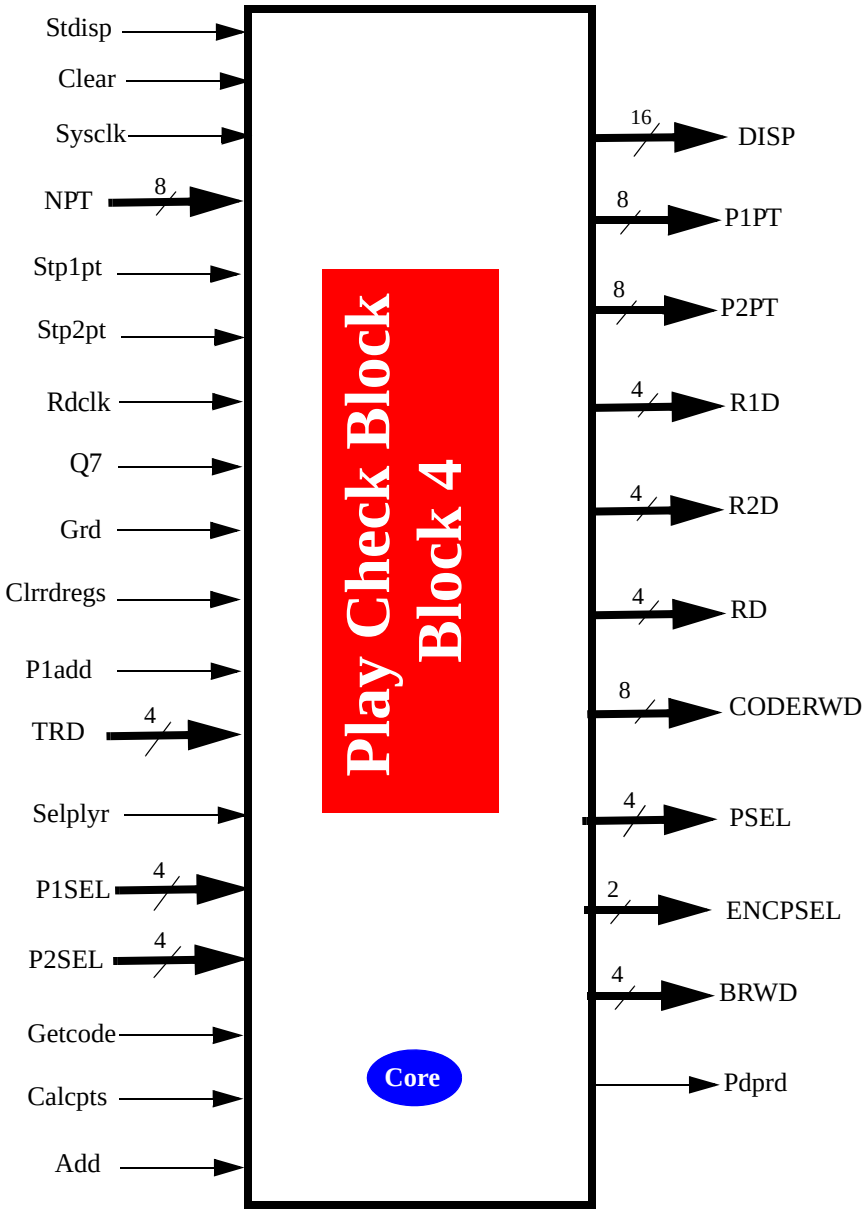


Figure 13. The detailed view of the input and output signals of the Play Check Block.

Name	# of lines	Description
STDISP	4	Store on a position display. They are active high and active in states 2 and 5. Only one of them is 1, indicating the position played. When it is 1, the played digit is stored on its register. From Block 1

Table 10: Inputs of the Play Check Block.

Name	# of lines	Description
Clear	1	Clear registers, counters and flip-flops. It stores zeros on registers, counters and FFs at reset. From Block 1
Sysclk	1	The system clock at 6 Hz. It is used to synchronize operations on sequential circuits. From Block 2
NPT	8	New player points. The new points calculated for the current player. From Block 5
Stp1pt	1	Store Player 1 points. It is 1 in state 2 & the Player 1 points register is stored the new points. From Block 1
Stp2pt	1	Store Player 2 points. It is 1 in state 5 & the Player 2 points register is stored the new points. From Block 1
Rdclk	1	The random digit clock at 192 Hz. It is used by the random digit counter. From Block 2
Q7	1	Clock signal at 1.5 Hz. It is used as the least significant bit of the random digit. From Block 2
Grd	1	Get a new random digit. When it is 1, it signals a new random digit is to be generated for the next player. It is 1 if the game is in states 0, 1, 3 and 6. When it is 1, the random digit register is stored the outputs of the counter as the random digit. From Block 1
Clrrdregs	1	Clear random digit registers. Three registers keep 3 random digits and are cleared at reset From Block 1
P1add	1	Player 1 wants to add. It is 1 when the human player 1 turns on SW0 to add the random digit from a display. It is also used as the least significant bit of RD input to the machine player manually. From Block 2
TRD	4	The four bits of the random digit used to debug players. They are labelled as TR4-TRD0. From Block 2
Selplyr	1	Select a player. It is 1 to select Player 2 in states 4, 5 and 6, and 0 in states 0, 1, 2 and 3. From Block 1
P1SEL	4	Player 1 position display select. Each switch selects a position display to play. From Block 2
P2SEL	4	Player 2 position display select. Each line selects a position display to play. From Block 6
Getcode	1	Get the code for the game. It signals a new code is to be generated for the game. It is active-high and active in state 0 after Player 1 presses BTNR four times. From Block 1
Calcpts	1	Calculate the code reward points. It is active and 1 in states 2 and 4, indicating to the Code Reward Calculation circuits to output the code reward for the current player. From Block 1
Add	1	Add. It is the P1add in state 1 when the human player plays and P2add when the machine player plays in state 4. If it is 1 in states 2 and 3 or in states 5 and 6, it means an addition was done during the play by the human or machine player, respectively. It is activated in these states if the game is not over. From Block 1

Table 10: Inputs of the Play Check Block.

Name	# of lines	Description
DISP	16	Position display bits. Four position display digits in Unsigned Binary. To Block 2, 5 and 6
P1PT	8	Player 1 Points. Player 1 points in Unsigned binary. To Block 2, 5 and 6
P2PT	8	Player 2 Points. Player 2 points in Unsigned binary. To Block 2, 5 and 6
R1D	4	Next random digit. Output to Block 2 and 6
R2D	4	The random digit following next random digit. Output to Block 2
RD	4	The current random digit. In states 1, 2, and 3, it is the human player RD. In states 4, 5 and 6, it is the machine player RD. To Block 2 and 6

Table 11: Outputs of the Play Check Block.

Name	# of lines	Description
CODERWD	8	The reward if the player plays on display 0 or 1 and the digit played is identical to a code digit. It is output in states 2 and 5. Output to Block 6
PSEL	4	Player position select. Each bit selects a position to play for the current player. In states 1, 2 and 3, they are Player 1 position selections. In states 4, 5 and 6, they are Player 2 position selection signals. All bits are output to Block 2. Bits 1 and 0 are output to Block 6
ENCPSEL	2	Encoded position select. The bits indicate the played position in Unsigned Binary. To Block 5
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 addition. To Block 5 and 6
Pdprd	1	Position display plus random digit. If Pdprd is 1, the current play generated a display overflow : The sum of the position played and the random digit is greater than $(15)_{10}$. It is the display overflow signal for the human player in states 1, 2 and 3 and for the machine player in states 4, 5 and 6. To Block 2

Table 11: Outputs of the Play Check Block.

Because of 36 inputs to Block 4, we cannot obtain a detailed input/output relationship. An operation diagram is needed since the block has sequential circuits. But, to simplify our discussion we obtain an operation table (Table 12). The table has three rows, one for each major operation. We then have a subblock for each row : **Position Displays, Points Storage and Random Digit Generation** Subblock, **Display Add and Compare** Subblock and **Played Digit Determination** Subblock (Figure 14). Each subblock is partitioned based on their major operations, design goals and available components. These three subblocks allow a better sharing of Block 4 by the human and machine players.

Operation
Store BRWD on one of the four position displays • 16 DISP signals show displays Store Player 1 points and Player 2 points on their registers • 8 P1PT signals show the Player 1 points • 8 P2PT signals show the Player 2points Generate new random digits when Grd indicates so • Three sets of 4 signals show three random digits. Also, if one of SW15-SW12 is active, input a random digit from SW15-SW12
Code storage and code reward calculation • Keep the 8-bit Code • Calculate the code reward after each play Select current displays or next displays and compute (PD_k + RD) to generate intermediate signals : • 16 NPDISP signals as the result of four additions between the four displays and RD • Four PDPRD signals, indicating if PD_k + RD is greater than $(15)_{10}$, that is if the addition generates a display overflow
Generate various signals to be used by both players : • Select one of Player 1 or Player 2 position selects signals (P1SEL or P2SEL) based on Selplyr and output as PSEL • Generate the 2-bit ENCPSEL lines from PSEL lines that encode the number of the position played • Generate the Pdprd signal (display overflow) of the position played by using ENCPSEL Determine the new set of display bits : • Determine the four NPSELDISP signals from 16 NPDISP signals by using ENCPSEL Determine the digit to be played on a position display from NPSELDISP and RD, by using Add : • BRWD3, BRWD2, BRWD1 and BRWD0 indicate the digit to be played (stored), also the minimum points to earn

Table 12: Operation table of the Play Check Block.

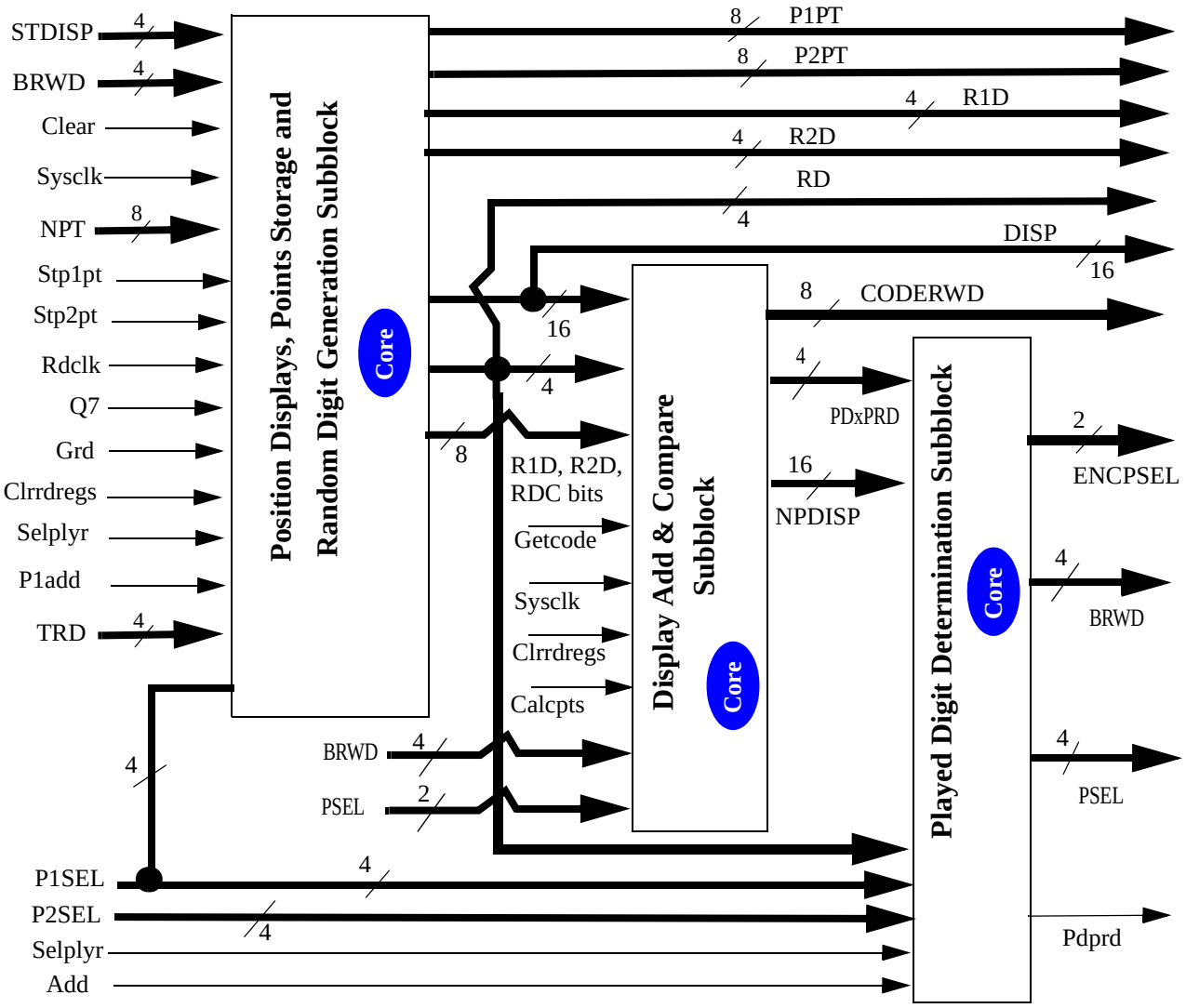


Figure 14. The Play Check Block partitioning.

The Position Displays, Points Storage and Random Digit Generation Subblock has 34 inputs. It also has sequential circuits. Therefore, we have to get an operation diagram. But, to simplify our discussion, we will obtain an operation table shown on Table 13. Given the three rows on the table, one concludes that the subblock a) keeps (stores) the four position displays, b) stores current player's points and c) generates next three random digits. Therefore, this subblock consists of three subsubblocks : Position Displays Storage Subsubblock, the Players' Points Storage Subsubblock and Random Digit Generation Subsubblock. The three subsubblocks are shown in Figure 15.

Operation
If Sysclk has a positive edge, store BRWD on a display pointed by STDISP. If Clear is 1, zero the displays
If Stp1pt/Stp2pt is active and Sysclk has a positive edge, store the new player points, NPT, on the respective player points register P1PT/P2PT. If Clear is 1, zero the registers
If Grd is active and Sysclk has a positive edge, get another random digit so that overall there are three random digits. If Clear is 1, zero the registers that keep the random digits. If one of SW15-SW12 is 1, input SW15-SW12 as a random digit

Table 13: Operation table of the Position Displays, Points Storage and Random Digit Generation Subblock.

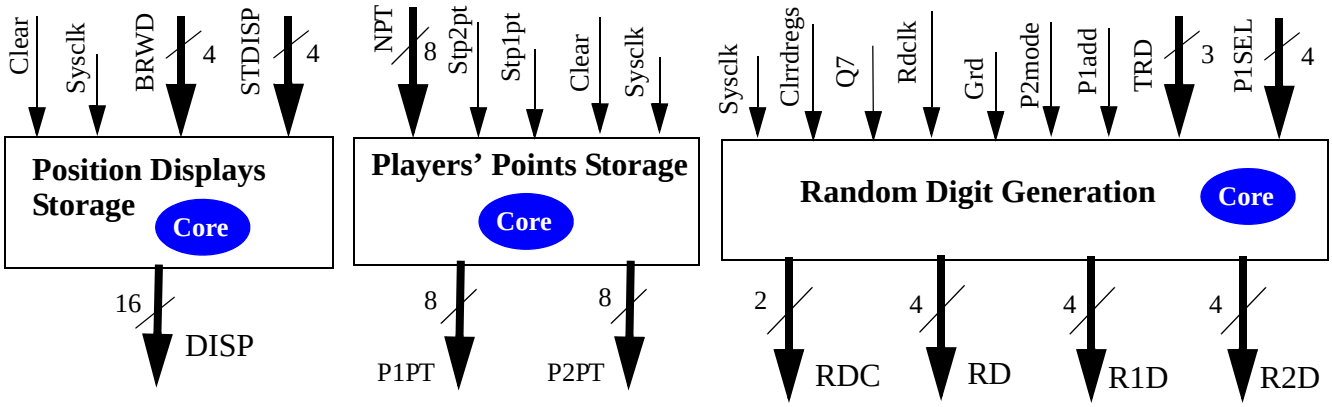


Figure 15. The Position Displays, Points Storage and Random Digit Generation Subblock partitioning.

The Position Displays Subsubblock 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 temporary storage. A position display register is stored a digit when its STDDISP signal is 1 and the clock input has a positive edge. These happen in state 2 for Player 1 and in state 5 for Player 2. The Sysclk is used as the clock signal (the clocking 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) and when the ppm is reset (state 0).

The Players' Points Storage Subsubblock is responsible for keeping the points of the players. It stores the new Player 1 points indicated by NPT, at the **end** of state 2 when Stp1pt is 1 and there is a positive edge on Sysclk. It stores the new Player 2 points indicated by NPT, at the **end** of state 5 when Stp2pt is 1 and there is a positive edge on Sysclk. The new points is stored if previous plays did **not** generate any points overflow (Lptovf must be 0). If the current play generates such a situation (Ptovf is 1 now), the new Player 1 or Player 2 points is still stored on the 8-bit register. Once Ptovf is 1, it is latched as Lptovf. From that moment on, no points storage is allowed for both players and the game is stopped only to be restarted if the reset push button (BTND) is pressed.

In order to understand how the points is stored on the two registers, the operation table of the circuit that stores the Player 2 points is shown on Table 14. The design follows the sequential circuit design conventions as follows : the storage of Player 2 points happens in two situations. One, when the machine is reset (state 0) to store zero and second, when Player 2 earns points (state 5). The Clear input stores 0 and the Stp2pt stores the new points. The Clear has higher priority than Stp2pt as Clear is asynchronous, and so the Player 2 points register is cleared (stored 0) even if the Stp2pt input is active. The subsubblock stores new points only when Stp2pt is active and Clear is inactive, and the Sysclk has a positive edge. Figure 16 shows the timing sequence when Player 2 earns points in state 5 (S5 is 1). The Sysclk period duration is sufficiently long so that the register can receive the correct addition result (NPT) from the adder in Block 5. That is, the Sysclk period is longer than the addition time for NPT : $t_{npoints}$.

Clear	Stp2pt	Sysclk	Operation
1	x	x	Clear Player 2 register immediately (**asynchronously**)
0	1	↑	Store new points (NPT) on the Player 2 Points register. Note that once the Ptovf is 1, no **further** storage on the points register happens
0	x	0	Not Stored

Table 14: The operation table of the Player 2 Points Storage circuit.

If the subsubblock is correct but the register stores incorrect results, this might be due to two reasons. One is a longer points calculation time than the Sysclk period. That is, $t_{npoints}$ in Figure 16 is longer than the Sysclk period. If yes, the new points calculation logic in Block 5 must be changed to shorten $t_{npoints}$. Otherwise, the Xilinx implementation may not be optimal and so students need to change the optimization level of the Xilinx implementation.

The Random Digit Generation Subsubblock has a freely running counter, a Xilinx CD4CE, at 192 Hz. There are also three registers connected in a cascaded fashion to form a shift register. The registers keep three random digits. In

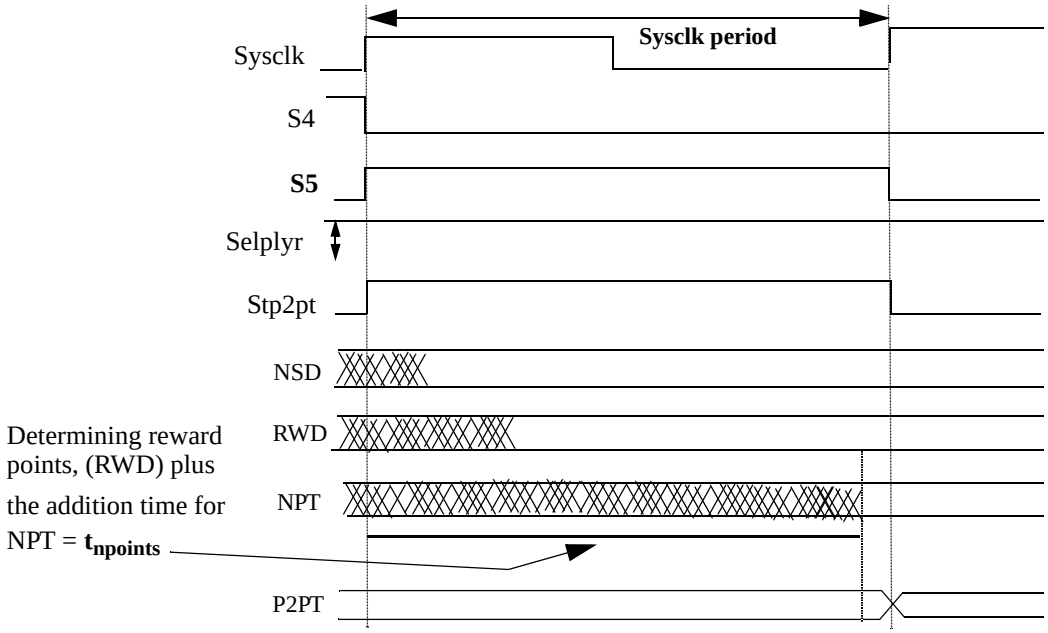


Figure 16. The timing of storage for Player 2 in Block 4 in state 5.

reset, BTND is pressed three times in a row, making Grd 1 three times generating three random digits. The registers are cleared by Clrrdregs in state 0 at reset. Note that the least significant output of the counter is **not** stored on the register. If it is stored, the register output, the random digit, happens to be always odd. This is due to the Xilinx software that during the conversion of the schematic to the bit file an error occurs. To solve it, the Q7 line which is a 1.5 Hz signal is connected to the least significant input of the register. Another circuit in this subsubblock allows a random digit to be input from SW15-SW12. If one of SW15-SW12 is one 1, then the digit coming from the register is bypassed and the value from the switches is output as RD.

The Display Add and Compare Subblock has only sequential and combinational circuits. Since it has 38 inputs, we need to get its operation diagram to start its design. However, to simplify its discussion, we will obtain its operation table as shown on Table 15. There are two rows for two major operations on the table, therefore there need to be two subsubblocks. One subsubblock is for the code generation and code reward calculation and the other one is Adding RD to the Displays.

Operation
Generate and keep the code bits and calculate the code reward after every play •By using bits RDC3, RDC1, R2D2, R2D1, R1D3, R1D1 and R1D0 obtain the eight code bits when Getcode is 1 and Sysclk has a positive edge. If Clrrdregs is 1 which happens at reset, clear the code registers. •After a player plays, calculate the code reward points if the player plays on display 1 or 0 and the played digit is identical to the code digit on that display
Generate 16 NPDISP and 4 PDkPRD lines from DISP and RD •Select between DISP15-DISP12 and TDISP15-TDISP12 and add it to RD to generate NPDISP15-NPDISP12 and compare => PD3PRD = 1 if PD3 + RD > $(15)_{10}$, •Select between DISP11-DISP8 and TDISP11-TDISP8 and add it to RD to generate NPDISP11-NPDISP8 and compare => PD2PRD = 1 if PD2 + RD > $(15)_{10}$, •Select between DISP7-DISP4 and TDISP7-TDISP4 and add it to RD to generate NPDISP7-NPDISP4 and compare => PD1PRD = 1 if PD1 + RD > $(15)_{10}$, •Select between DISP3-DISP0 and TDISP3-TDISP0 and add it to RD to generate NPDISP3-NPDISP0 and compare => PD0PRD = 1 if PD0 + RD > $(15)_{10}$

Table 15: Operation table of the Display Add and Compare Subblock

The Code Storage and Code Reward Subsubblock internal design is not available and is given to students as a black box, a macro, named M1. It generates the code by using RDC, R2D and R1D1 bits and keeps it as the 8-bit code. The generation happens after the last BTND press when the game is started. At this moment the Getcode control signal is 1, instructing the Subsubblock to generate the code bits. The code bits are pseudo-random and do not change until a player wins the game. The two 4-bit code digits are never non-zero. The code reward calculation circuits compare the digit played (BRWD) with the two 4-bit code digits that reside on the rightmost two displays. If there is a match, a non-zero reward is output in states 2 and 5. There are two different possibilities for the code reward :

- The digit played (BRWD) matches one of the two code digits on one of the rightmost two displays and the other code digit is not equal to the digit is **not** equal to the display digit at the moment : The player earns 8 times BRWD.
- The digit played (BRWD) matches one of the two code digits on one of the rightmost two displays and the other code digit is not equal to the digit is **equal** to the display digit at the moment : The player earns a reward whose value is equal to the Code value..

The Adding RD to the Displays Subsubblock internal design is visible to the students. There are four subrows on Table 15 and accordingly, four major operations and four subsubblocks (Figure 17). The four subsubblocks are identical to each other, i.e. they are replicated circuits : Four identical circuits that differ in only input and output names. This is **not** unusual and many real-life circuits have such **replicated** hardware, simplifying the design considerably. We will encounter such regular circuits throughout the semester as we focus on the datapath and in general, the **datapath** of a digital system is highly regular, making it easier to design than the Control Unit.

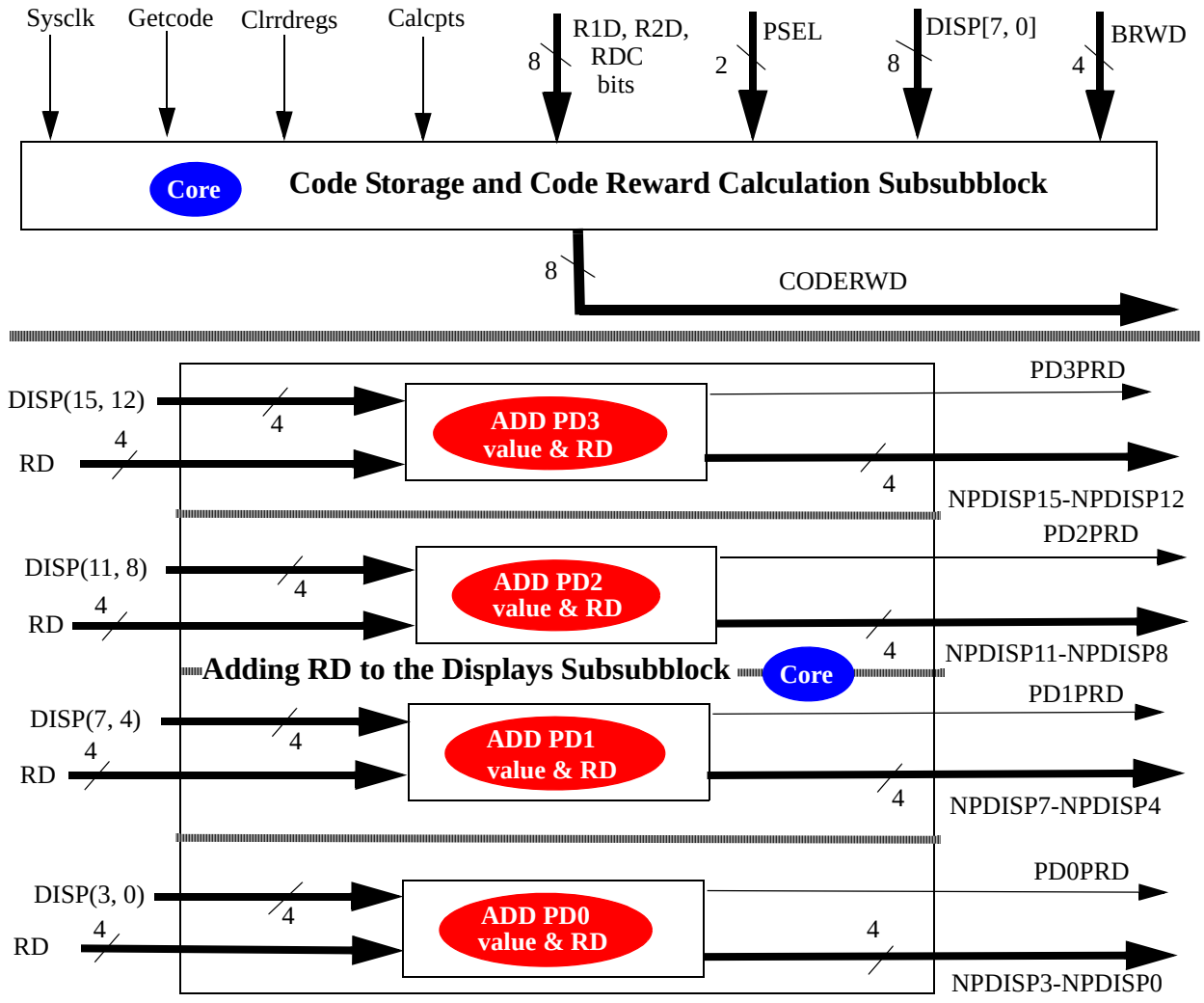


Figure 17. The Display Add and Compare Subblock partitioning.

The Display Add and Compare Subsubblock computes new display values and compares them with $(15)_{10}$. It has four identical circuits. Each circuit multiplexes between current (DISP) and next (TDISP) display values and adds the selected value to RD. The subblock outputs four addition results on 16 wires (NPDISP) and four PDkPRD values, by using four Xilinx 4-bit 2-to-1 MUXes (X74_157) and four 4-bit Xilinx ADDers (ADD4). In this subblock, comparing the addition result with $(15)_{10}$ does not require a comparator ! The c_{out} output of the 4-bit Xilinx ADDer (ADD4) is used as the comparator output, since in Unsigned Binary addition, c_{out} indicates if the addition has generated an overflow. In this case, the overflow happens if the sum is greater than $(15)_{10}$.

The Played Digit Determination Subblock multiplexes between Player 1 and Player 2 signals and determines the digit played on a position (BRWD). To generate BRWD, it first selects between P1SEL and P2SEL to indicate which position is selected by the current player. It then select the four appropriate NPDISP lines. For example, if the position played is 3, it outputs NPDISP lines for position 3 : NPDISP (15, 12) and outputs as NPSELDISP. Then it determines BRWD by selecting either RD or NPSELDISP, by using Add. If the player has played directly, it outputs RD otherwise, it outputs NPSELDISP. The operation table of the Similar Digit Determination Subblock is given on Table 16. Since there are three rows on the table, we divide the subblock into three subsubblocks as shown in Figure 18.

Operation
Generate PSEL, ENCPSEL and Pdprd lines from P1SEL, P2SEL, Selplyr and PDkPRD lines
Based on ENCPSEL select four lines out of 16 NPDISP and output as NPSELDISP
Based on Add select either NPSELDISP or RD and output as BRWD

Table 16: The operation table of the Played Digit Determination Subblock.

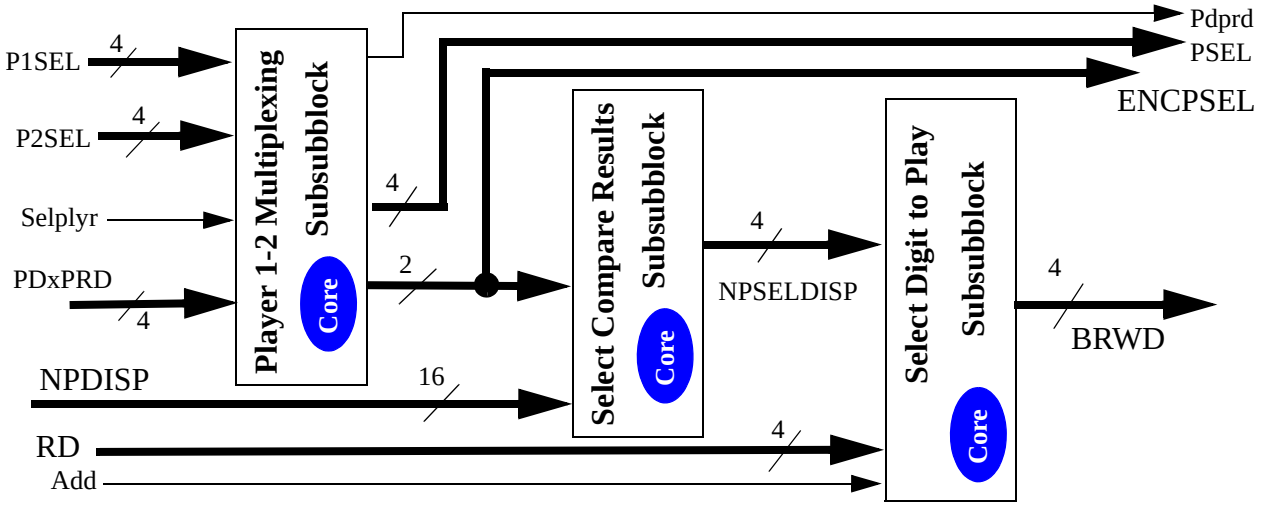


Figure 18. The Played Digit Determination Subblock partitioning.

The Player 1-2 Multiplexing Subsubblock outputs PSEL by selecting either Player 1 position select signals (P1SEL) or Player 2 position select signals (P2SEL). It uses the Selplyr as the select signal, selecting 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. The subsubblock encodes the four PSEL lines as ENCPSEL by using a gate network. ENCPSEL shows the played position number in Unsigned Binary. For example, if PSEL is 0100, position 2 is selected, the ENCPSEL output should be 10. The main reason for generating these two signals is that multiplexers in this block and Block 5 need select signals to select one out of four. To select one out of four, the two ENCPSEL lines are used. Table 17 shows how the ENCPSEL values are determined.

ENCPSEL is generated from PSEL, requiring a **4-to-2 non-priority binary encoder**. It is non-priority since no more than one PSEL line is 1 at a time. When implementing this encoder not all four PSEL signals are needed, but just three. Why three PSEL inputs would suffice to generate the ENCPSEL inputs becomes clear with Homework 5

PSEL	ENCPSEL
0001	00
0010	01
0100	10
1000	11

Table 17: The generation of the Encoded Position Number.

which has a question on a non-priority 10-to-4 encoder. Finally, the subsubblock selects one of the four compare values (Pd3prd, Pd2prd, Pd1prd and Pd0prd) and outputs as Pdprd by using ENCPSEL. To generate the Pdprd signal a 4-to-1 Xilinx MUX, M4_1E, is used. When Pdprd is 1, it indicates a display overflow for the played position. Which position played on is indicated by the ENCPSEL signals : they are the select signals for the 4-to-1 MUX.

The Select Compare Results Subsubblock outputs four added display values, NPSELDISP, based on ENCPSEL and NPDISP. For example, if ENCPSEL indicates position 0 is played on, then position 0 NPDISP bits, (NPDISP(3, 0)), are output as NPSELDIP. The operation table of this subsubblock is shown on Table 18. The operation table clearly indicates that this subsubblock is implemented by using a 4-bit 4-to-1 MUX. Xilinx does not have such large MUXes. Its largest 4-to-1 MUX is the 2-bit 4-to-1 MUX, which is the X74_153 component. Therefore, the 4-bit 4-to-1 MUX is built by using two X74_153 components in this subsubblock.

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

Table 18: The operation Table of the Select Compare Results Subsubblock.

The Select Digit to Play Subsubblock outputs the digit played BRWD. BRWD is the digit played and stands for Basic Reward. It is Basic Reward since if there is no adjacency the player earns BRWD points. The operation table of the Select Digit to Store Subsubblock is as shown on Table 19. The circuits in this subsubblock selects either RD or NPSELDISP based on Add. If Add is 0, it means the player decided to play directly on a display, therefore RD lines are selected and output as BRWD. If Add is 1, the player has decided to add RD to a display and so NPSELDISP lines are selected and output as BRWD. The operation table clearly indicates that this selection circuit is implemented by using a 4-bit 2-to-1 Xilinx MUX, which is the X74_157 component.

Add	Operation
0	BRWD = RD
1	BRWD = NPSELDISP

Table 19: The operation table of the Select Digit to Store Subsubblock.

4.2.4. The Ppm Core Data Unit Block, Points Calculation Block, Block 5

Block 5 calculates new player points. It also determines if there is a points overflow and which positions are similar to the digit played. Block 5 is shared by both human and machine players, reducing hardware. It is a completely combinational block. The block has 47 inputs and 19 outputs (Figure 19). Tables 20 and 21 list the inputs and outputs of Block 5, respectively.

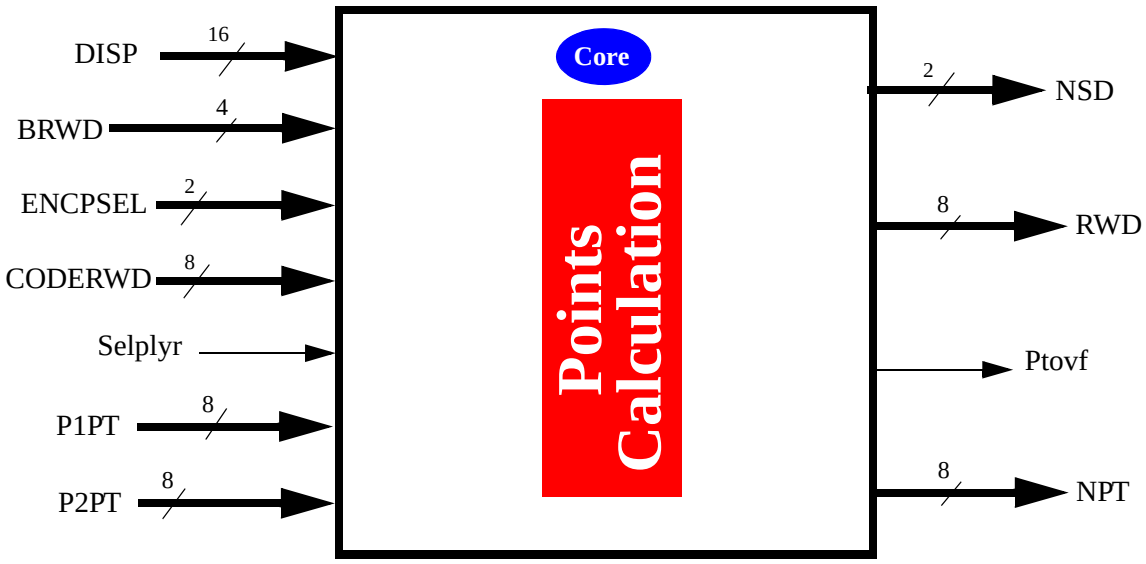


Figure 19. The input and output signals of the Points Calculation Block.

Signal name	Number of lines	Description
DISP	16	Position display bits. Four position display digits in Unsigned Binary. From Block 4
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. From Block 4
ENCPSEL	2	Encoded position select. They indicate the played position number in Unsigned Binary. From Block 4
CODERWD	8	The reward if the player plays on display 0 or 1 and the digit played is identical to a code digit. It is output in states 2 and 5. From Block 4
Selplyr	1	Select players. It is 1 to select Player 2 in states 4, 5 and 6, otherwise, 0 in states 0, 1, 2 and 3. From Block 1
P1PT	8	Player 1 Points. The Player 1 points in Unsigned binary. From Block 4
P2PT	8	Player 2 points. The Player 2 points in Unsigned Binary. From Block 4

Table 20: Inputs of the Points Calculation Block.

Signal name	Number of lines	Description
NSD	2	Number of Similar Digits. It is the adjacency of the position played in Unsigned binary. To Block 1 and Block 6
RWD	8	Reward points. It is in Unsigned Binary. To Block 6
Ptovf	1	Points overflow. If it is 1, the current player points is above $(255)_{10}$. To Block 2
NPT	8	New player points. The sum of the addition of the current player points and the reward points. It is in Unsigned Binary. To Block 4

Table 21: Outputs of the Points Calculation Block.

Block 5 has 47 inputs. We cannot obtain a detailed input/output relationship and so we have to obtain an operation

table. We realize that to calculate the new player points for the current player, we need to obtain the adjacency information from which we calculate the reward points. Thus, Block 5 needs to determine (i) how many adjacent identical digits there are next to the digit played (the adjacency), (ii) the reward points, and (iii) new player points and points overflow. The operation table is as shown on Table 22. Based on it, there are three subblocks as shown in Figure 20.

Situation	Operation
New DISP and/or ENCPSEL and/or BRWD	Based on DISP, ENCPSEL and BRWD, determine the number of adjacent identical digits next to the position played : NSD (number of similar digits). NSD has two bits for four possible adjacency situations : 0 adjacency, 1, 2 and 3
New NSD and/or BRWD	Based on NSD, BRWD, determine the amount of regular reward points for the current player. The regular reward points, RWD, has eight bits to reward up to $(255)_{10}$ points
New Selplyr and/or P1PT and/or P2PT and/or RWD and/or CODERWD	Based on RWD, CODERWD and current player points (P1PT or P2PT), determine the new points. If Selplyr is 0, the current player is human, otherwise, the machine. Also, determine if there is a points overflow

Table 22: The operation table of Block 5, the Points Calculation Block.

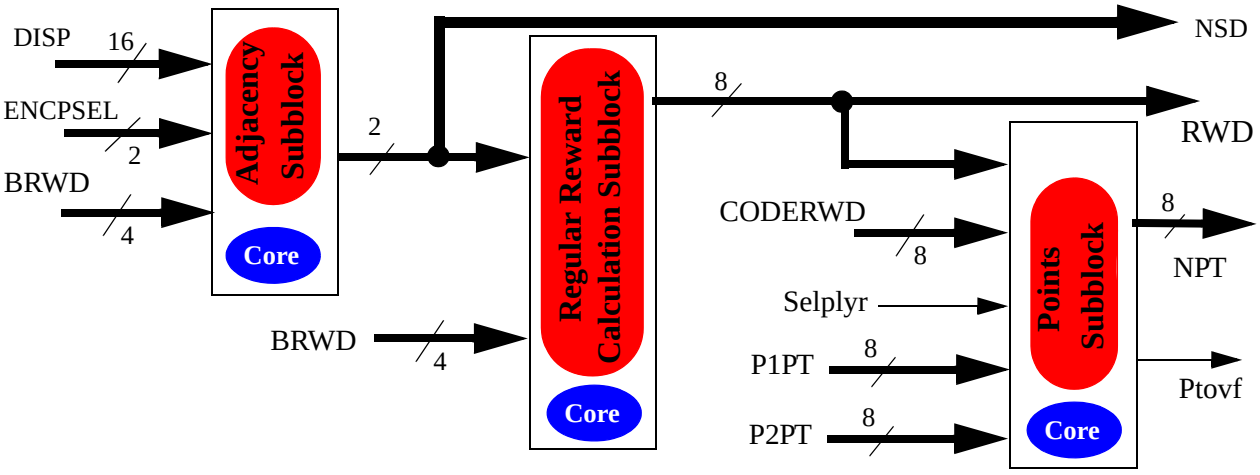


Figure 20. The Points Calculation Block partitioning.

We stress that one can get different operation tables for the same block. Also, given an operation table, one can get different major operations. These are true for even experienced people ! Also, a person needs the experience of estimating how much logic (the number of components) a major operation needs before deciding about the number of major operations. Often, one has to iterate a number of times so that the subblocks have similar amount of logic : They need to be balanced.

As mentioned above, the partitioning is hard. But, it is actually harder since not only we have to get an operation table with acceptable major operations, but also we have to consider the design goals and available technology. While major operations indicate the subblocks that perform them, the available technology indicate how many components to use for a block (how easy a block will be implemented). Even experience may not 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. The design becomes even more iterative !

Finally, note that the last **two** major operations for the ppm system *at this point*, calculating new points (Block 5) and playing for the machine (Block 6) seem to be independent of each other and can be implemented separately. This is the case with the ppm project at the course web site. It implements them separately, by using the Points Calculation Block and the Machine Play Block. However, the two major operations can be tightly coupled too : Adjacencies and reward points are needed for an intelligent machine player to decide how to play. In fact, for a real (commercial) game chip, they could be tightly integrated !

4.2.4.1. The Adjacency Subblock

The subblock determines the adjacency, NSD. It has 22 inputs and two outputs. Outputs NSD, indicate the amount of adjacency for the played position. It has two bits to indicate four adjacency possibilities : 0, 1, 2 and 3, based on the DISP and ENCPSEL inputs. The DDISP inputs are compared with BRWD to determine which positions are equal to the digit played. The ENCPSEL inputs indicate the played position. Then, the operation table of the Adjacency Subblock is shown on Table 23.

Situation	Operation
BRWD is played on **PD3** **(**ENCPSEL = 11)	• No adjacency : NSD = **00** • One adjacent digit (PD2 = BRWD : NSD = **01** • Two adjacent digits (PD3 = PD1 = BRWD) : NSD = **10** •Three adjacent digits (PD2 = PD1 = PD0 = BRWD) : NSD = **11**
BRWD is played on **PD2** (ENCPSEL = 10)	• No adjacency : NSD = **00** • One adjacent digit (PD3 = BRWD **or** PD1 = BRWD) : NSD = **01** • Two adjacent digits (PD3 = PD1 = BRWD **or** PD1 = PD0 = BRWD) : NSD = **10** • Three adjacent digits (PD3 = PD1 = PD0 = BRWD) : NSD = **11**
BRWD is played on **PD1** (ENCPSEL = 01)	• No adjacency : NSD = **00** • One adjacent digit (PD2 = BRWD **or** PD0 = BRWD) : NSD = **01** • Two adjacent digits (PD2 = PD0 = BRWD **or** PD2 = PD3 = BRWD) : NSD = **10** • Three adjacent digits (PD3 = PD2 = PD0 = BRWD) : NSD = **11**
BRWD is played on **PD0** (ENCPSEL = 00)	• No adjacency : NSD = **00** • One adjacent digit (PD2 = BRWD **or** PD0 = BRWD) : NSD = **01** • Two adjacent digits (PD2 = PD0 = BRWD **or** PD2 = PD3 = BRWD) : NSD = **10** • Three adjacent digits (PD3 = PD2 = PD0 = BRWD) : NSD = **11**

Table 23: Operation Table of the Adjacency Subblock.

Table 23 has four major rows, one for each position. We could go for four major operations for this subblock. However, we realize that all operations are the same for all four positions. We could get a different set of major operations to save hardware. What we can do is that we get adjacencies for the four positions first, choose the one played on and output it. There are still four major operations and four subsubblocks : Equality Check Subsubblock, Adjacent Similar Digits Subsubblock, Unencoded Adjacency Subsubblock and Encoded Adjacency Subsubblock (Figure 21). Note that this subblock is an unusual case where the four major operations are not explicit, but implied on Table 23.

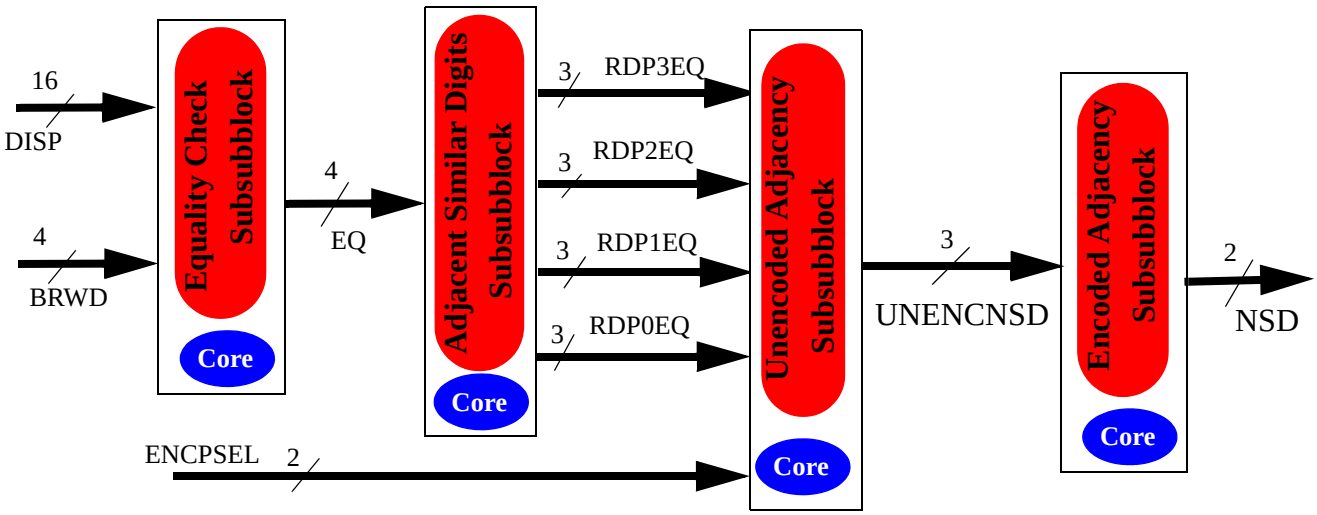


Figure 21. The Adjacency Subblock partitioning.

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 to determine the adjacency and 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 24. Since BRWD and PD_k digits are 4-bit unsigned numbers and we check only for the equality case, we need four 4-bit Unsigned Binary Comparators. Xilinx has two such comparators. One is the COMP4 component and the other one is the X74L_85 component. For simplicity reasons, the circuit is implemented by four COMP4 components.

BRWD	Positions **before** placement	Outputs
2	7 2 0 4	EQ = 0100
2	7 1 0 4	EQ = 0000
9	0 9 0 9	EQ = 0101
3	3 0 0 3	EQ = 1001
3	3 0 0 1	EQ = 1000
6	6 6 6 0	EQ = 1110
6	6 6 6 6	EQ = 1111

Table 24: Equality cases for BRWD.

The Adjacent Similar Digits Subsubblock generates adjacency information for the four positions. It has 4 inputs (EQ) and 12 outputs (four sets of three bits). One set for one position : RDP3EQ (position 3), RDP2EQ (position 2), etc. We use three lines for each position as there can be up to three identical digits in a row on the displays. If there is only one adjacent identical digit, only one of the three bits is 1. It does **not** matter which one it is : 001 or 010 or 100. If there are two adjacent digits **in a row**, two of the three bits are 1 and again it does not matter which bits are one : 110 or 011 or 101. If there are three similar adjacent digits, the bits are 111. To simplify the design, we do **not** assume PD3 is adjacent to PD0. Tables 25 and 26 give examples of adjacencies from different perspectives.

BRWD	Display positions **before** placement	Adjacent Similar Digits Outputs	Unencoded Adjacency Outputs UNENCNSD	Encoded Adjacency Outputs NSD
2 **ENCPSEL** = 01	F 2 0 A **EQ** = 0100	RDP3EQ = 001 or 010 or 100 RDP2EQ = 000 RDP1EQ = 001 or 010 or 100 RDP0EQ = 000	**RDP1EQ** selected 001 or 010 or 100	01
5 **ENCPSEL** = 11	0 C 0 5 **EQ** = 0001	RDP3EQ = 000 RDP2EQ = 000 RDP1EQ = 001 or 010 or 100 RDP0EQ = 000	**RDP3EQ** selected 000	00
7 **ENCPSEL** = 00	0 7 7 F **EQ** = 0110	RDP3EQ = 110 or 011 or 101 RDP2EQ = 001 or 010 or 100 RDP1EQ = 001 or 010 or 100 RDP0EQ = 110 or 011 or 101	**RDP0EQ** selected 110 or 011 or 101	10
3 **ENCPSEL** = 01	3 3 E 3 **EQ** = 1101	RDP3EQ = 001 or 010 or 100 RDP2EQ = 001 or 010 or 100 RDP1EQ = 111 RDP0EQ = 000	**RDP1EQ** selected 111	11

Table 25: Independent adjacency examples.

BRWD	Display positions **before** the play	Outputs
2	7 2 0 4	**RDP3EQ** = 100 or 010 or 001 ; **RDP2EQ** = 000 ; **RDP1EQ** = 100 or 010 or 001 ; **RDP0EQ** = 000
5	7 1 0 4	**RDP3EQ** = 000 ; **RDP2EQ** = 000 ; **RDP1EQ** = 000 ; **RDP0EQ** = 000
9	0 9 0 9	**RDP3EQ** = 100 or 010 or 001 ; **RDP2EQ** = 000 ; **RDP1EQ** = 110 or 011 or 101 ; **RDP0EQ** = 000
9	0 5 0 9	**RDP3EQ** = 000 ; **RDP2EQ** = 000 ; **RDP1EQ** = 100 or 010 or 001 ; **RDP0EQ** = 000
3	3 7 0 3	**RDP3EQ** = 000 ; **RDP2EQ** = 100 or 010 or 001 ; **RDP1EQ** = 100 or 010 or 001 ; **RDP0EQ** = 000
6	6 6 6 0	**RDP3EQ** = 110 or 011 or 101 ; **RDP2EQ** = 110 or 011 or 101 ; **RDP1EQ** = 110 or 011 or 101 ; **RDP0EQ** = 111
8	8 0 8 0	**RDP3EQ** = 000 ; **RDP2EQ** = 110 or 011 or 101 ; **RDP1EQ** = 000 ; **RDP0EQ** = 100 or 010 or 001

Table 26: Additional adjacency examples.

Since this subsubblock has 4 inputs, one can immediately obtain a truth table with 16 rows, 12 output columns, and by using the K-map technique, 12 minimal expressions. We notice then that six outputs do **not** need any logic. These outputs are directly connected to six inputs. An input line is an output line. But, when one tries to implement this on Xilinx, the software does **not** allow it. The software does **not** allow assigning more than one name to a wire. In order to rename the wires without using any logic, Xilinx BUF components (buffers) are used in this subsubblock.

Consider the RDP3EQ outputs that indicate how many adjacent digits there are next to PD3. If there is only **one** adjacent digit, it must be in position PD2 : EQ2 must be 1. One of the three RDP3EQ bits is output 1. It does not matter which RDP3EQ bit is 1. If there are two adjacent similar digits **in a row** next to PD3, they must be on PD2 and PD1: EQ2 and EQ1 must be 1. Two of the three RDP3EQ bits are 1. The order of 1's again does not matter. If there are three adjacent identical digits next to PD3, they must be on PD2, PD1 and PD0 : EQ2, EQ1 and EQ0 must be 1. All three RDP3EQ lines are 1 : 111. Figure 22 shows the circuit to generate RDP3EQ signals. One can reason about the other position displays in a similar fashion to obtain the circuits, remembering that the adjacency of similar digits must be **continuous**, i.e. in a **row**. Also, this checking is done **before** BRWD is stored on the position display.

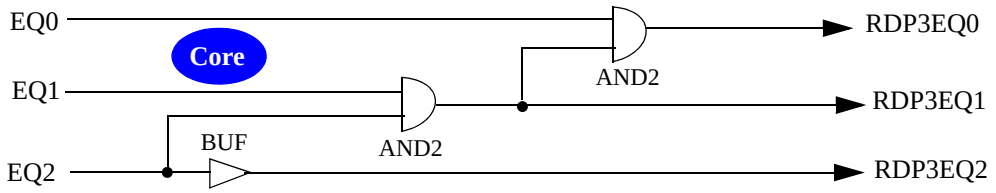


Figure 22. The RDP3EQ circuit in the Adjacent Similar Digits Subsubblock.

The Unencoded Adjacency Subsubblock has 14 inputs and three outputs named UNENCNSD, meaning unencoded number of similar digits. It selects one of the four sets of outputs from the Adjacent Similar Digits Subsubblock depending on the position played (ENCPSEL). Thus, UNENCNSD is equal to the three bits of the selected position. For example, if ENCPSEL is 01, position 1 is selected. Then, UNENCNSD is equal to RDP1EQ. Table 27 shows the operation table of this subsubblock.

ENCPSEL	UNENCNSD
00	RDP0EQ
01	RDP1EQ
10	RDP2EQ
11	RDP3EQ

Table 27: Operation table of the Unencoded Adjacency Subsubblock.

In order to understand the circuit implementation in the context of the *Digital Product Development* Handout, here we describe the thinking process for this subsubblock : We go through the list of questions in the Handout. The subsubblock selects (multiplexes) one out of four sets of lines : RDP3EQ, RDP2EQ, RDP1EQ and RDP0EQ ! It is a 3-bit 4-to-1 MUX. The first question on the list is "Is there a single (or a few) Xilinx non-programmable component(s) that implement(s) this MUX ?" Does Xilinx have such a MUX ? No ! Its largest 4-to-1 MUX is the 2-bit 4-to-1 MUX : X74_153. But two MUXes can implement the subsubblock. The answer is "Yes." We use an X74_153 MUX and a (1-bit) 4-to-1 MUX, the M4_1E to implement the 3-bit 4-to-1 MUX !

The Encoded Adjacency Subsubblock has three inputs, UNENCNSD, and two outputs named NSD (number of similar digits). NSD is the adjacency of the played position in Unsigned Binary. To get the adjacency, the subsubblock counts how many 1's there are on the UNENCNSD lines. For example, if UNENCNSD is 110, then NSD is 10, meaning $(2)_{10}$ since there are two 1's on "110." Since there are three inputs, one can immediately obtain a truth table with two output columns, then two K-maps and eventually two minimal expressions. The truth table shows that the encoding of the three inputs is nothing but adding them. The implementation is then a 1-bit ADDer, a Full ADDer !

4.2.4.2. The Regular Reward Calculation Subblock

The subblock calculates the regular reward points for the current player. It has six inputs and eight outputs. The outputs are for the regular reward points named RWD. When a new NSD or BRWD is supplied, a new regular reward points is calculated. The black box view of the subblock is shown in Figure 23 and the operation table is shown on Table 28. If the operation table is analyzed it is difficult to figure out the major operations. We realize we need to get a **different** operation table that can help us determine the major operations quickly. This new operation table of the subblock is shown on Table 29. This table is a good example of the fact that for a circuit there can be more than one operation table ! On the table, we see that RWD is determined by BRWD and NSD.

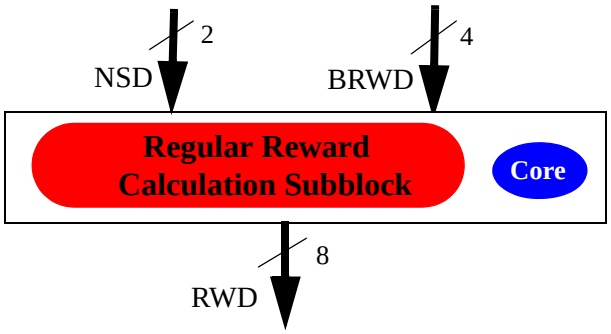


Figure 23. The Regular Reward Calculation Subblock.

Situation	Operation
New NSD and/or BRWD	Based on NSD, BRWD, determine the amount of reward points for the current player. The reward points, RWD, has eight bits to reward up to $(255)_{10}$ points

Table 28: The operation table of the Regular Reward Calculation Subblock.

NSD	RWD
00	BRWD
01	2 * BRWD
10	4 * BRWD
11	8 * BRWD

Table 29: A Different Operation Table for the Regular Reward Calculation Subblock.

Now, the major operations ! Are they just multiplication and multiplexing ? Multiplication is needed for adjacencies and multiplexing is needed to select one of the results ? **No** ! There are several reasons ! The first reason is about the multiplication major operation. There is a peculiar point in binary arithmetic that to multiply a number by 2, by 4 and by 8, a multiplier is **not** needed. As we discuss in class, one can multiply a number by 2 by shifting it left by one bit. Multiplying by 4 is a shift left by two bits, and so on. As an example if the number is 0011 which is three in decimal and it is shifted left by one bit, it becomes 0110 which is six in decimal. We see we do **not** need a multiplier to do two times the number, four times the number, etc.

A shifting circuit that attaches zeros to the right of the number is needed. Second, do we need the shift and multiplexing major operations ? No ! On paper, we see that attaching a zero to the right of a number does **not** need any logic ! It just requires an arrangement of wires such that wire values are either 0 or the bits of the number shifted. Then, there is **no** shift major operation ! The only major operation is multiplexing ! In summary then, the subblock outputs an 8-bit Unsigned Binary number, the regular reward points (RWD), based on NSD and BRWD by performing the multiplications via shifting as shown on Table 29. It has a multiplexing circuit to select one possibility for RWD.

Implementing the multiplexing circuit is **not** trivial and has to be started on paper. How large is it ? How many choices are there ? Inputs of the multiplexer must be determined such that the wire arrangement accomplishes the shift operation. Table 29 shows that there are four choices to select from. That is, the multiplexing selects a value from the following choices on Table 29 : BRWD or 2*BRWD or 4*BRWD or 8*BRWD. So, it is a 4-to-1 multiplexer and since we output an 8-bit number, it is an 8-bit multiplexer. Then, the size of the multiplexer is 8-bit 4-to1. Does Xilinx have such MUXes ? No ! Its largest 4-to-1 MUX is a 2-bit one : X74_153. Then, the 8-bit 4-to-1 MUX is implemented by using three X74_153 Xilinx components and a gate.

4.2.4.3. The Points Subblock

This subblock adds the regular reward points and the Code reward to the current player points to calculate the new player points. It also determines if there is a points overflow. The subblock has 29 inputs and nine outputs (Figure 20). The operation table is shown on Table 30. Is it possible to determine the major operations on the operation table? Yes ! Each row of the table is a major operation : A select and an add. Thus, the subblock has two subsubblocks : Select Player Points and Points Addition (Figure 24).

Input	Operation
New Selplyr and/or P1PT and/or P2PT	Select the current player points from P1PT and P2PT by using Selplyr and output as PT
New RWD or CODERWD or PT	Add to the points of the current player (PT), the regular reward (RWD) and the code reward (CODERWD) to generate the new player points (NPT). Generate the points overflow bit (Ptovf) which is 1 if PT + RWD+ CODERWD is greater than $(255)_{10}$

Table 30: The operation table of the Points Subblock.

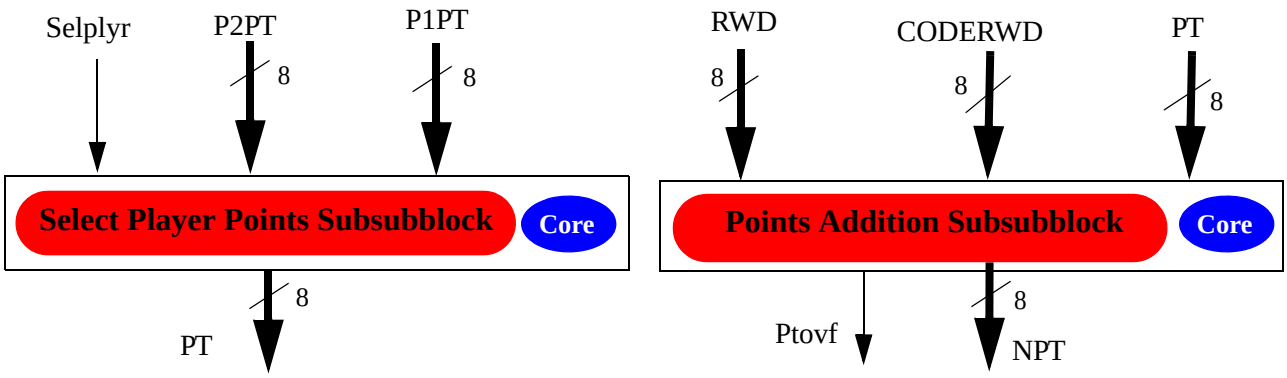


Figure 24. The Points Subblock partitioning.

The Select Player Points Subsubblock selects one of Player 1, Player 2 points based on Selplyr and outputs as PT. The operation table is shown on Table 31. The major operation is an 8-bit multiplexing operation where it outputs Player 1 points (P1PT) if Selplyr is 0, otherwise outputs Player 2 points (P2PT). The multiplexer is an 8-bit 2-to-1 multiplexer which is implemented by two Xilinx 4-bit 2-to-1 multiplexers.

Selplyr	PT
0	P1PT
1	P2PT

Table 31: Operation table of the Select Player Points Subsubblock.

The Points Addition Subsubblock adds the selected points value (PT) to the 8-bit regular reward (RWD) and the 8-bit code reward (CODERWD). The result is the new Player points (NPT) and the points overflow bit (Ptovf). The operation table is shown on Table 32. The major operation is an 8-bit addition of three numbers. This addition is implemented by two Xilinx 8-bit adders. Output Ptovf is 1 if the result of the addition, (PT + CODERWD + RWD) is greater than $(255)_{10}$. Note that the Ptovf output should be generated based on the fact that PT, RWD and CODERWD are Unsigned Binary numbers.

Operation
• NPT = RWD + CODERWD + PT • Ptovf = 1 if RDW +CODERWD + PT > $(255)_{10}$

Table 32: The operation table of the Points Addition Subsubblock.

4.2.5. The Ppm Core Data Unit Block, Machine Play Block, Block 6

Block 6, the Machine Play Block, is the machine player. It plays against the human player. Block 6, is needed in state 4 of the ppm operation diagram given in Figure 5. In that state the machine player thinks and makes a decision which is either playing on a position or skipping the play. Currently, Block 6 is implemented by macros M2, M3 and M4 in addition to a core circuit.

4.2.5.1. Machine Player Playing Strategies

While exceeding $(255)_{10}$ points before the human player does is the main goal of the machine player, accomplishing it can be done in different ways, i.e. by using different playing strategies. A strategy is a high-level game playing plan that is needed especially when there are alternative plays. The strategy determines how much into the future the machine player can project or how much it can distribute the burden of a single (difficult) play to several plays. Students who are familiar with chess playing can relate to this discussion.

One can think of a number of playing strategies for the ppm game. A strategy that is easy to implement is the **random playing strategy** where the player (i) randomly selects a position and (ii) randomly decides whether to play directly or with an addition. It does not skip. The random playing strategy can result in missed adjacencies and code rewards and so the machine player would lose games against the human player most of the time. A slightly more complex strategy, the **fixed playing strategy** is that the player always plays in a fixed way, regardless of the situation. For example, the player plays in a round robin way on position displays with direct playing then additions, then direct playing again and so on, making its plays very predictable. This strategy would also miss adjacencies and code rewards and lose to the human player many times. One can think of other similar less intelligent machine player strategies. For example, the player checks from left to right which position has a **single** adjacency, and then plays there. If there is no adjacency, the player plays on the rightmost position. This strategy would miss two adjacency cases and code rewards and lose games often.

There are a number of more intelligent playing strategies. Overall, for such strategies, the machine player needs to gather information and then decide what to do. For example, it would gather reward points information for all positions and then decide. In this **points oriented playing strategy** the player always plays on the position that earns the largest regular reward points. The reason why this strategy can be for an intelligent machine player is that the player has to look for regular reward points for all the positions with direct playing and with additions. This is not a simple

task. The drawback of this strategy is that looking for largest points at the moment means not thinking about adjacencies that can eventually result in more points soon. Because, we know that not always largest adjacencies mean largest earnings. However, going for adjacencies is favorable since the player is given another chance to play and earn more points while the opponent waits for the turn. Therefore, even a more intelligent player would gather adjacency and regular reward points information for all positions and then decide. But, this intelligent can also lose games since it does not keep track of code digits which can provide additional points besides the regular points.

Another intelligent machine player strategy can be the one that checks regular reward points, adjacencies and also thinks about the future, that is at least the next random digit. This player would try not to play large numbers on displays all the time, unless necessary. By keeping display values low would lead to more adjacency situations before display overflows. This strategy implies that the machine player has different plays for the same combination of display and random digit values. For example, a machine player plays for adjacencies if it is ahead and when it is behind, it plays for largest regular reward points. Again, this machine player would also lose games since it does keep work on the code digits.

A more intelligent machine player checks regular reward points, adjacencies and code digits. We know that the code digits are not visible, but can be determined by game situations. That is, by observing how many points after a play, the player can determine the code digit on that position. For a machine player, it means it has to work not only in state 4 of the ppm operation diagram, but also in states 2 and 5 during which human player and machine player points are calculated. Only in states 2 and 5 the CODERWD bits can be non-zero hence the need to be active in states 2 and 5. As we see below, the course web site machine player uses such a strategy.

Students note that their machine player strategy needs to deal with situations when there are two or more equally good positions to play given an intelligent strategy. For example, if playing on two positions results in the same regular reward points and no code reward, which one will be selected ? One solution is that the fixed strategy is used where always the rightmost of these two positions is selected. Another solution is that one of these two positions is randomly chosen. Therefore, students will have **two** strategies, one **primary** strategy that deals with most of the situations and a **secondary** one that handles equally playable situations.

4.2.5.2. The Machine Player at the Course Web Site

The Machine Play Block at the course web site is implemented by Macros M2, M3 and M4 as well as a core circuit. It has 70 inputs and 15 outputs (Figure 25). Tables 33 and 34 list the inputs and outputs of the macro, respectively. Students note that the number of inputs is not absolute : there can be more inputs for more machine intelligence. Because, the player needs to collect more information to make a better decision. As students decide how much "thinking" or intelligence the block will have, they will also determine the inputs. The number of outputs is absolute though. It will **not** change ! Students have to generate the top **seven** outputs in Figure 25, otherwise, the machine player will **not** play properly. The other eight outputs on the bottom of Figure 25 can be output as 0 all the time.

The Course Web Site Machine Player Playing Strategy is a relatively intelligent one. It thinks ahead and plays so that its earnings currently and in the future are maximized as much as possible. It checks for a number of situations and consequently performs one of six (6) actions when it is its turn. When it encounters a specific case, it decides to skip. In any situation, if there are equally good playing possibilities, it uses the fixed playing strategy where it plays on the rightmost of the playable positions. There is an exception to that in case there is code digit playing possibility. There are other details about the strategy that are shown in Figure 26.

Overall, first the machine player tries to win the game with the current random digit. Otherwise, it tries to keep displays small unless there is a code digit play possibility on displays 1 or 0. If no, it goes for largest adjacencies, not for largest regular reward points when it is ahead. If there are zeros on the displays in the beginning of the game, it plays on zero positions for future adjacencies. When there is no code digit possibility and it is not ahead, it plays for the largest regular reward points. It also checks the next RD (R1D) to see if it is equal to a code digit. Note that the strategy in Figure 26 indicates what the machine player does in states 2, 4 and 5 of the ppm operation diagram.

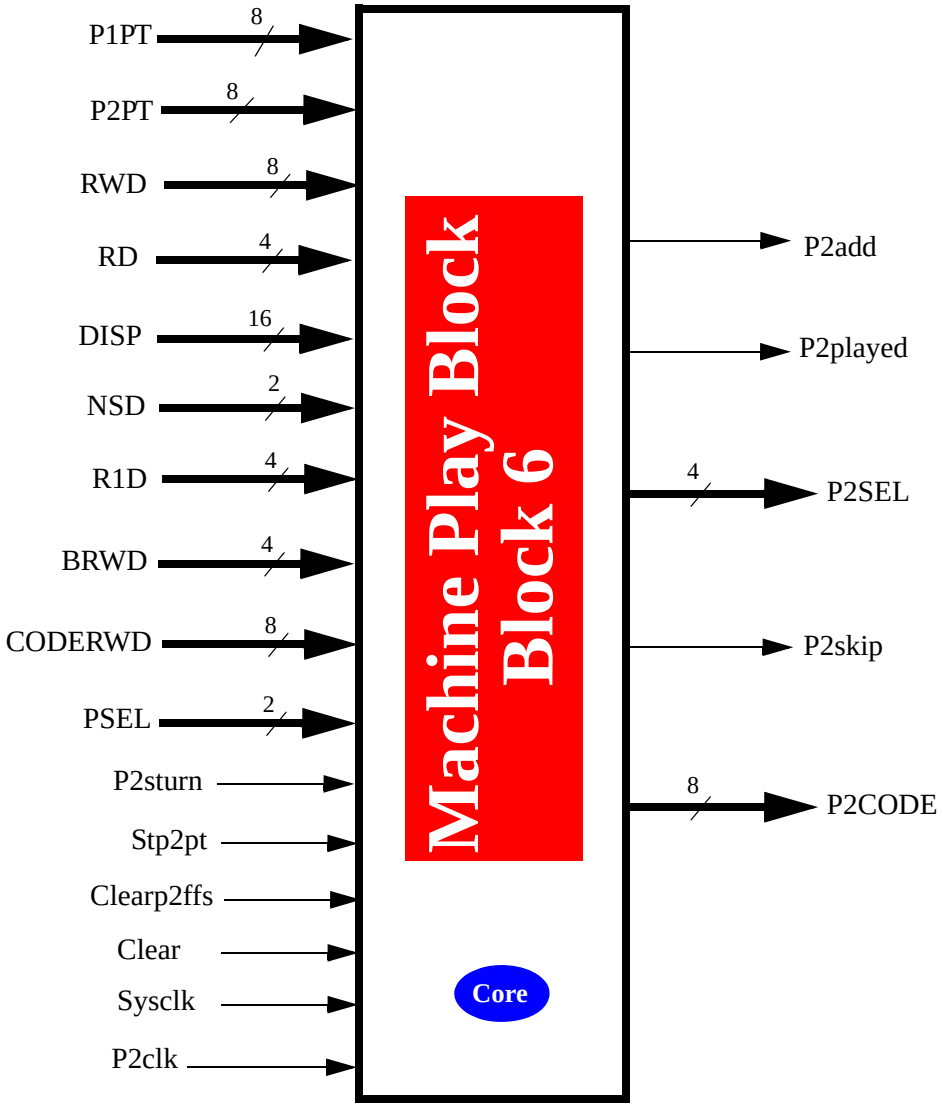


Figure 25. The input and output signals of the course web site Machine Play Block.

Signal name	Number of lines	Description
P1PT	8	Player 1 points. It is in Unsigned Binary. From Block 4
P2PT	8	Player 2 points. It is in Unsigned Binary. From Block 4
RWD	8	Reward points. It is in Unsigned Binary. From Block 5
RD	4	The random digit. It is in BCD. From Block 4
DISP	16	Position Display bits. 16 bits represent the four position display bits. From Block 4
NSD	2	Number of Similar Digits. The adjacency of the position played. From Block 5

Table 33: Inputs of the course web site Machine Play Block.

Signal name	Number of lines	Description
R1D	4	Next random digit. It is in BCD. From Block 4
BRWD	4	Basic Reward. It is also the digit played. It is n Hex. From Block 4
CODERWD	8	The reward if the player plays on display 0 or 1 and the digit played is identical to a code digit. It is output in states 2 and 5. From Block 4
PSEL	2	Player position select. Only bits 1 and 0 are input for code digit determination. Each bit selects a position to play for the current player. In states 1, 2 and 3, they are Player 1 position selections. In states 4, 5 and 6, they are Player 2 position selection signals.
P2sturn	1	Player 2's turn. When the ppm is in state 4, it is 1. From Block 1
Stp2pt	1	Store Player 2 points. It is 1 in state 5 & the Player 2 points register is stored the new points. From Block 1
Clearp2ffs	1	It clears Player 2 registers, counters and FFs. It happens **after** Player 2 plays. From Block 1
Clear	1	Clear all registers, counters and flip-flops. It is used to store zero on all registers, counters and flip-flops when the system is reset. It is active high and active in state 0. From Block 1
Sysclk	1	The system clock at 6 Hz. It is used to synchronize operations on sequential circuits. From Block 2
P2clk	1	The Player 2 clock at 48 Hz. It is used to synchronize operations. From Block 2

Table 33: Inputs of the course web site Machine Play Block.

Name	Number of lines	Description
P2add	1	Player 2 wants to add. When it is 1, the machine player wants to add the random digit to a display. To Block 1
P2played	1	Player 2 has played. When it is 1, it means Player 2 has selected a position to play. To Block 1
P2SEL	4	Player 2 position select. Each line selects a position to play a digit. To Block 4
P2skip	1	Player 2 skip. When it is active, 1, it means Player 2 skips the play. To Block 1
P2CODE	8	Code digits discovered by the machine player. Two 4-bit Hex coded digits. To Block 2

Table 34: Outputs of the course web site Machine Play Block.

Let's analyze the course web site machine player strategy in detail. The strategy consists of condition checks and actions. The machine player first determines if it is close to winning the game, a condition. That is, if the current machine player points plus the largest regular reward points it can get exceeds $(255)_{10}$, then it is close to winning.

Therefore, it plays for the most regular reward points, an action. Otherwise, it checks in the order of (i) playing a code digit, (ii) if the next RD is a code digit, (iii) if it is early in the game and (iv) if it is ahead of the human player. If it is ahead, it goes for adjacencies and plays for the future. If the machine player is not ahead, it plays for largest reward points. Below we describe first the actions and then the conditions :

- Action 0 : Play on the (rightmost) largest regular reward points position (directly if equal)
  - The machine player collects regular reward points information for each position for both cases of the position is played directly and with an addition. It selects the largest reward position. If there are several positions with the same largest reward points, it plays on the rightmost of these positions. If playing directly and adding result in the same largest points, it chooses playing directly .
- Action 1 : Play on the (rightmost) largest adjacency position (directly if equal)
  - The machine player collects adjacency information for each position for both cases of the position is played directly and with an addition. It selects the largest adjacency position. If there are several posi-

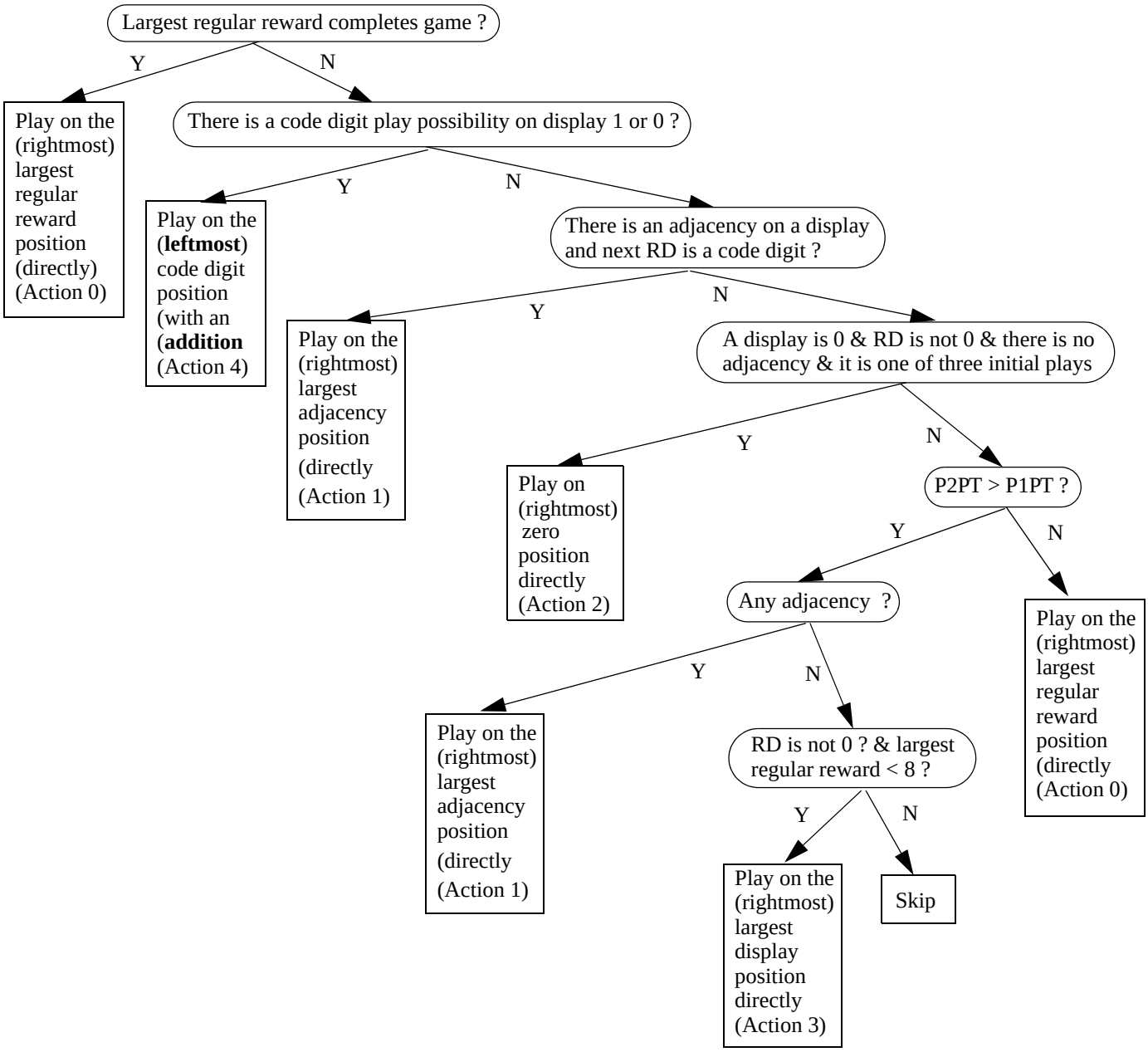


Figure 26. The playing strategy of the machine player at the course web site.

tions with the same largest adjacency, it plays on the rightmost of these largest adjacency positions. If playing directly and adding result in the same largest adjacency, playing directly is chosen.

- Action 2 : Play on the (rightmost) zero position directly
  - The machine player checks each position if it is zero. It selects the position with a zero and plays the random digit directly. If there are several positions with zero, it plays on the rightmost of these zero positions.
- Action 3 : Play on the (rightmost) largest display position directly
  - The machine player checks each position to see which has the largest value. It selects the position with the largest value and plays the random digit directly. If there are several positions with the same largest display value, it plays on the rightmost of these positions.

- Action 4 : Play on the (leftmost) code digit position (with an addition if equal)
  - The machine player checks positions 0 and 1 with direct playing and additions to see if any results in a code digit that it discovered. If yes, it plays on that position. If both positions result in a code digit, it selects the leftmost of these two position. If direct playing and adding gives code digits, then it selects adding.
    - In order to discover the code digits, the Action 4 circuitry checks the value of CODERWD in states 2 and 5 of the ppm operation diagram. We know that the CODERWD is valid only in these two states when the new player points is calculated. If the CODERWD value is non-zero, then the Action 4 circuitry stores the digit played and its position number on two registers.

- Action 5 : Skip the play and give the turn to the opponent.

- Largest regular reward completes game ?
  - Does the sum of the largest regular reward and the current machine player points exceeds $(255)_{10}$?

- There is a code digit play possibility on display 1 or 0 ?
  - Does playing on position 0 or 1 with direct playing or adding generate a code digit ?

- There is an adjacency on a display and next RD is a code digit ?
  - Does playing the random digit on any position results in an adjacency and the next random digit if played directly is equal to one of the discovered code digits ?

- A display is 0 & RD is not 0 & there is no adjacency & it is one of three initial plays
  - Has the machine player played not more than three times yet (the game has just started) and there is no adjacency on displays and RD is not zero and a display is zero ?

- P2PT > P1PT ?
  - Is the machine player ahead in terms of game points ?

- Any adjacency ?
  - Is there any adjacency on the displays if the machine player plays the random digit either directly or with an addition ?

- RD is not 0 ? & largest regular reward < 8 ?
  - Is the random digit non-zero and the largest regular reward is less than 8 ?
    - The machine player considers this condition when it is ahead and there is no code digit possibility nor adjacency. Its goal is to reduce the largest display value so that the human player does not earn a lot of points.

A note about keeping the displays small is that if players constantly add to the displays they would earn large points. But, eventually display overflows occur and they earn smaller points. A general playing strategy can be looking for adjacencies and code digit (not for large reward points) to gradually increase the display values to have more adjacency opportunities before display overflows. For example, if the displays are (A644), adding RD = 4 on position 2 would result in one adjacency and $(20)_{10}$ points : (AA44). Playing RD on position 2 directly results in two adjacencies with lower points $(16)_{10}$: (A444). But this second play keeps the displays low so that there would be more adjacency opportunities until there are display overflows. However, looking for adjacencies and code digits requires more thinking effort. For the machine player that means considerable amount of additional hardware.

There is even more thinking effort needed if a player considers next two RDs to earn more points faster. For example, if the current displays are (A644), RD = 4 and next RD = 6, then playing as (AA44) may be better since the player plays again to get another adjacency with RD = 6 : (AAA4). Another example is that the displays can be (FCFE), RD = 2 and next RD = 1, then playing the 2 as (FEFE) would give a good opportunity to the opponent since the 1 can be to get (FFFE). The opponent would have an adjacency situation ! The course web site machine player does not check for these two situations though. The reason is that the FPGA space is exceeded if any other feature is added.

The Course Web Site Machine Play Block Partitioning is needed since the machine player is complex. How can we do the partitioning to implement the strategy in Figure 26 ? We know that the partitioning is done based on the major operations ! What are the major operations ? There are at least two major operations. One is gathering information about the current situation, such as the displays, player points, random digits, etc. : The **information gathering** major operation. The other is decision making based on the information gathered and the strategy : The **decision making** major operation. Thus, we need a subblock for information gathering and a subblock to make a decision.

If one decides to gather the information and make a decision at the same time, only these two subblocks are needed.

But, gathering the information requires a substantial amount of hardware. Gathering all at the same time (in parallel) requires even more. The FPGA chip space limit is exceeded. If, on the other hand, one gathers information sequentially (step by step) and then makes a decision, the hardware is reduced since we use the same hardware for different purposes : We do not need parallel hardware. But, a **controller** (a sequencing subblock) is needed to step through the sequence of information gathering steps. There are **three** major operations then ! Since the course web site machine player gathers information sequentially and then decides, it has a Sequencing Subblock, a small Control unit, as the controller. This Machine Player is in fact a tiny digital system with its own Control Unit and Datapath (Figure 27). Its operation diagram is shown in Figure 28(b) which is used in state 4 of the ppm operation diagram in Figure 28(a).

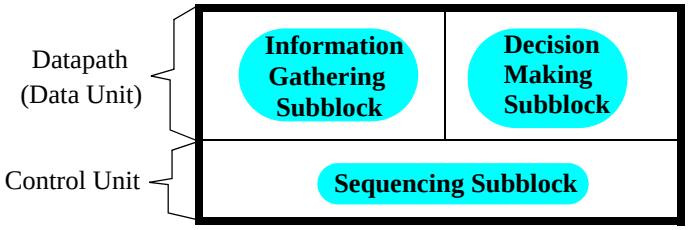
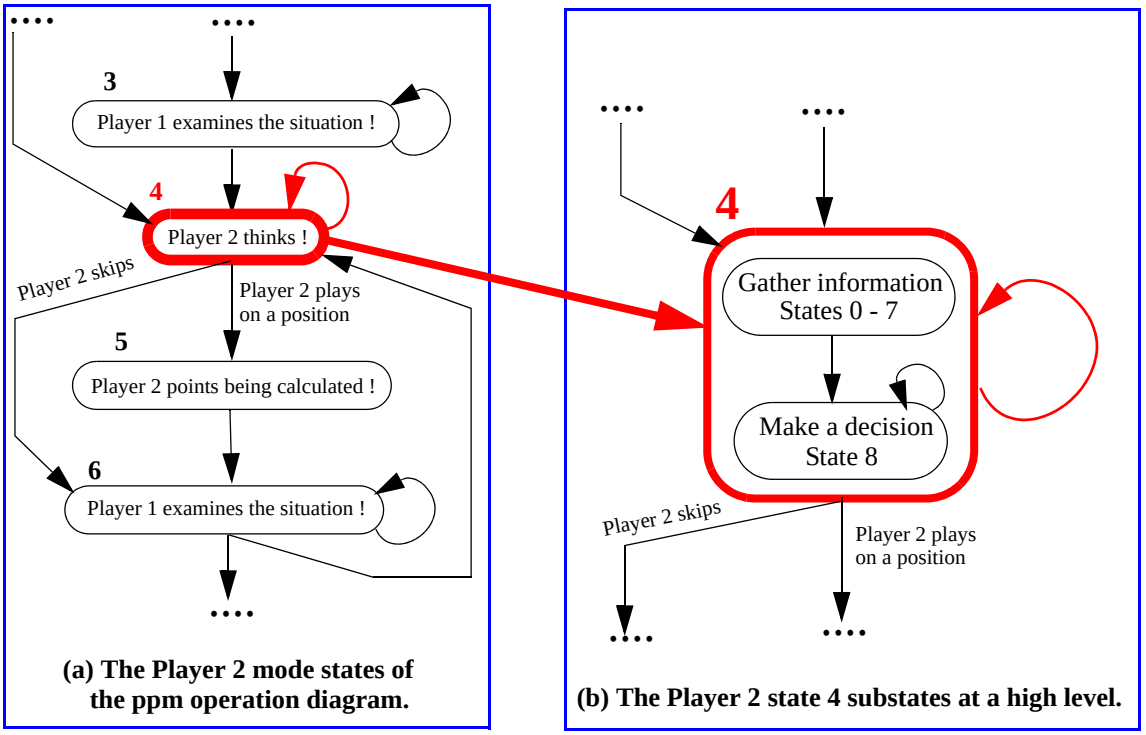


Figure 27. The course web site Machine Player partitioning.



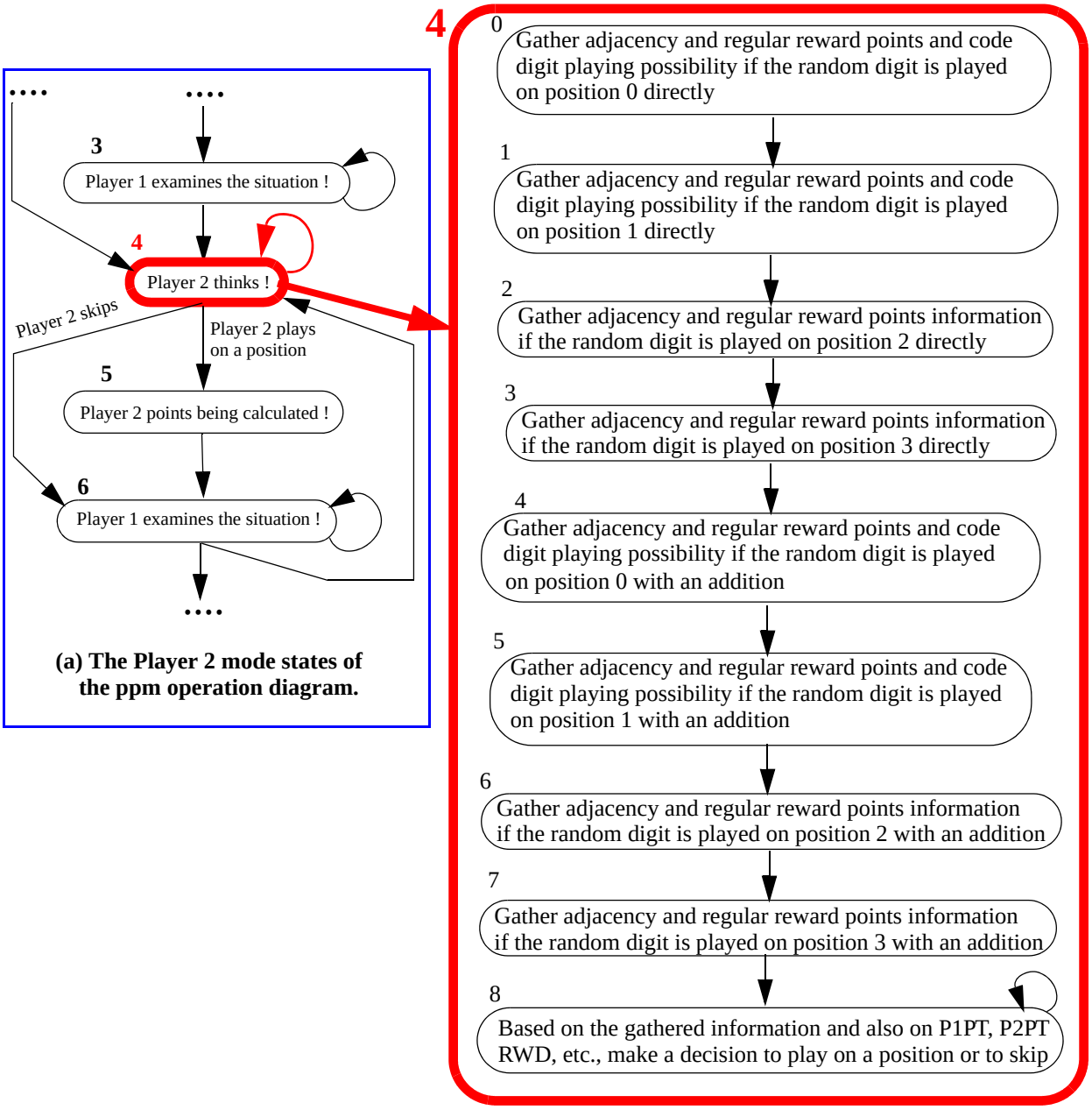
(a) The Player 2 mode states of the ppm operation diagram.

(b) The Player 2 state 4 substates at a high level.

Figure 28. The course web site Machine Player operation diagram.

The information gathering step in Figure 28(b) consists of 8 steps. For each display position it spends 2 clock periods, one for direct playing and one for adding to the display. The decision making step is just one clock period. Then, the course web site Machine Player has 9 steps shown in Figure 29(b), happening in state 4 of ppm of Figure 29(a). Notice the similarity of this discussion to section 2.3 where the Control Unit, Block 1, is used to step through the states of the ppm operation diagram. Note also that the machine player is active in states 2 and 5 of the ppm operation state diagram in order to discover the code digits.

Is it possible to calculate the number of clock periods it takes the machine player to play given how deep the play is checked ? Yes ! Here the depth is how many random digits are checked for all possibilities of RWD before making a decision. For example, the machine player at the course web site checks only the current random digit for all RWD



(a) The Player 2 mode states of the ppm operation diagram.

(b) The Player 2 substates.

Figure 29. The Detailed Machine Player operation diagram.

possibilities. The relation between the depth and the clock periods is as follows : For a random digit there are eight different RWD possibilities : four RWDs when the random digit is played directly on the four displays and four RWDs when the random digit is played on the four displays with an addition. Collecting these eight RWDs takes eight clock periods. If RWD is collected for the next random digit, we need 64 additional clock periods to collect 64 RWDs. Why 64 RWDs ? It is because once the current random digit is played on a position, we have eight new RWDs possible by using the next random digit. Since we have eight possibilities for the current random digit, then we collect 8 * 8 = 64 new RWDs. Thus, when the depth is two (RD and R1D), the number of RWDs is $8 + 8^2 = 72$. That is the search space is 72.

What if the machine player had a depth of three random digits (RD, R1D, R2D) ? How many different RWDs have to

be collected to make a decision ? In a more formal way, what is the size of the search space to make a decision ? Again, we collect eight RWDs for the current random digit then 64 RWDs for the next random digit. Finally, we collect 512 RWDs for the following random digit (R2D) ! The total number of RWDs is 8 + 64 + 512 = 584. This is what is often referred as the exponential search space for decision making. Such exponential search space sizes are common in game theory, including playing chess. It is also the reason why the implementation of many artificial intelligence problems is hard today !

It must be noted that the machine is active also in states 2 and 5 to determine if the CODERWD (code reward points) is non-zero. If non-zero, then the machine player stores the code digit value and the position number to be used after these states. Since it is the Information Gathering Subblock that collects such information, then it is this Subblock that is active in states 2 and 5.

The Sequencing Subblock controls the thinking and playing of the machine player. It goes through the 9 substates in the Machine Player operation diagram in Figure 29(b). Each substate corresponds to one P2clk period. We know that P2clk is at 48 Hz. Then a P2clk period is 1/48 = 0.02 seconds. Then, the thinking time in ppm state 4 is 9 * 0.02 = 0.18 seconds. One might wonder if the Player 2 substates can be combined with the ppm operation diagram such that state 4 of the ppm is replaced by 9 states. Is this plausible ? Yes, in a real game chip implementation, one would do exactly that. Overall, the ppm would have 15 states. But, we will not do that to simplify our discussion on the machine player.

The main circuit of this subblock is a counter/decoder combination similar to the counter/decoder combination in the Control Unit of the ppm. Figure 30 shows the counter/decoder combination of the Sequencing Subblock. The subblock goes through substates 0 through 8 at the rate of P2clk when it is in ppm state 4 (P2sturn is 1) **and** P2played is 0 **and** P2skip is 0. That is, as long as the Machine Player has not played nor skipped and it is ppm state, 4, the substates are traced.

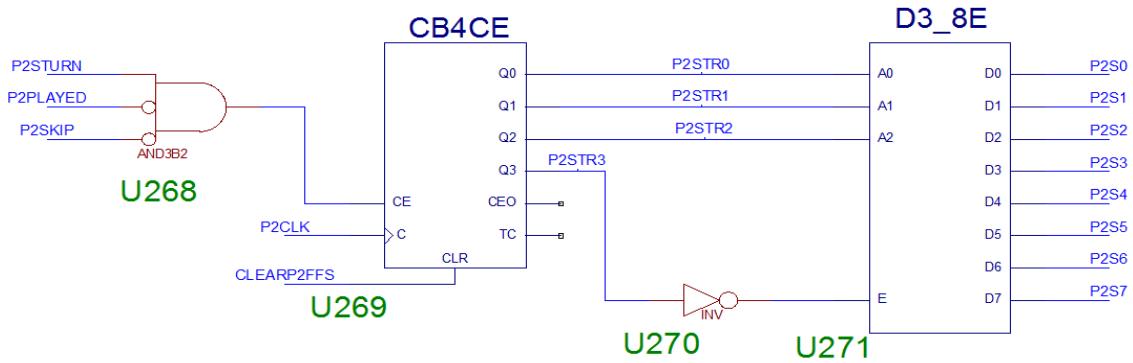


Figure 30. The counter decoder combination of the Sequencing Subblock.

The substate bits from the counter are named P2STR (Player 2 state register). The counter is reset (cleared) in ppm state 1 and in state 6 so that when it is machine player's turn next time the counter starts at 0. There are is a decoder to decode the counter outputs. The decoder is a 3-to-8 decoder, uses P2STR0, P2STR1 and P2STR2 and outputs P2S (Player 2 States) signals. Each output is for one of the top eight circles in Figure 29(b). For example, if P2STR0, P2STR1, P2STR2 and P2STR3 are 0110, P2S6 is 1 and the P2S signals are 0.

The Information Gathering Subblock gathers players points, regular reward points, adjacencies, displays, BRWD, RD, R1D, CODERWD information. During the first 4 substates, it gathers information position 0 through 3 by direct playing. Then, during the next 4 substates, it gathers information about position 0 through 3 by playing with an addition. The subblock also outputs eight P2CODE signals that represent the eight code digit bits discovered by the machine player. The circuit that discovers the code digits and implements Action 4 operations is in macro 4, M4, in Block 6. The circuit is a sequential circuit that is active in states 2, 4 and 5 of the ppm operation diagram. If students do not include a circuit like this, P2CODE *outputs must be 0 permanently.* That is, students connect GND to these eight outputs permanently.

The Decision Making Subblock determines the final values of the remaining seven outputs of Block 6 in substate 8. Especially, it turns on one of P2played and P2skip signals since the ppm Control Unit waits for one of these two sig-

nals to be 1. When the ppm Control Unit detects one of them is 1, it moves the ppm from state 4 to either 5 if a position is played or to 6 if the play skipped (Figure 29(a)). It is important that the P2played and P2skip outputs are kept low during the information gathering steps, 0 - 7. Because, the machine player is allowed to think as long as it wants if these two signals are kept low. Thus, it is critical for this subblock to wait until it is in substate 8 and then activate one of P2played/P2skip.

When it is time to decide, the subblock determines how to play. The possibilities are playing on display PD_k directly or playing PD_k + RD, or skip. If the machine player plays on a position, the subblock activates the corresponding position selection output (a P2SEL line) which then raises the P2played signal to 1, meaning the machine has played on a position. When P2played is 1, only one P2SEL line is 1 so that only one position is played on. If, on the other hand, the machine player decides to skip, it raises P2skip to 1. Overall, P2played must be 1 when the thinking is over **and** only one P2SEL line is 1. If P2played is 0, then P2skip can be turned on. If P2skip is raised to 1, the P2SEL signals are ignored.

5. The Machine vs. Machine Playing Description

This ppm version is the one where two machine players play against each other. It is used for the term project. The switches used by the human player SW7-SW4 and SW0 are **not** needed. The human player presses BTNR to allow the first machine player to play and BTNL to allow the second machine player to play. BTND and BTNU are used as before. The rules of the game are identical to the human vs. machine game rules. This version of the ppm has the same four macros as the human vs. machine project has : M1, M2, M3 and M4. Both players can be input a random digit from switches SW15-SW12 to debug the two player circuits.

The first step in the design is that students determine the machine player strategy, then designing the schematics of the machine player. They also need to keep in mind that schematics 1, 2, 4 and 5 are related to and used by the machine player. As they analyze and design these schematic circuits, students need to think how they can be used for the machine player. Students' experience during this time will help them select a strategy by the time they start the work on the machine player. Thus, students are suggested that they determine and think about the machine player strategy now, not when they start the design of the machine player later in the semester.

The **Player 2** machine player uses the playing strategy described in Figure 26, since it is the course web site machine player. The first machine player implemented by Block 3 has a different playing strategy. It uses the playing strategy shown in Figure 31. Player 1 always seeks for the largest regular reward points. If there are positions with identical largest regular reward points, it plays on the rightmost of these. If playing directly and with an addition results in the same regular largest reward points, playing directly is chosen. It never skips. Overall, Player 1 is much simpler than Player 2, the course web site machine player.

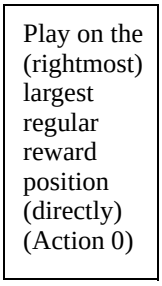


Figure 31. The playing strategy of Player 1 machine player at the course web site.

6. The Human vs. Human Playing Description

This ppm version is the one where two human players play against each other. Both human players use SW7-SW4 to select a position and SW0 to add the random digit. The rules of the game are identical to the human vs. machine game rules. A random digit can be input to either player manually. There is only **one** macro, M1, in this project.

7. Completion of the Term Project :

Students will complete the term project in Experiments 1 through 3. They will have a folder for each experiment, exp1 through exp3. **Once students are sure about their design, they will submit their term project with the machine vs. human project in their exp3 folder.** Students may also enhance ppmmvsh project, as long as it is a digital system. A TA will copy the exp3 folder of one of the team members to a memory stick.

Block 3 is worked on in Experiment 3. Experiment 3 completes the term project. Students will design Block 3 completely ! That is, the design of the machine player is left to students, including its intelligence level and playing strategy, provided that the machine player plays according to the rules, generates at least seven outputs described below and the block is completed by the last day of classes.

An intelligent machine player requires a large amount of logic to make a playing decision. Designing and testing such a machine player would take a considerable amount of time and so students are cautioned that they develop their machine player incrementally to make sure they complete it on time. The required logic would be even much larger if students design such that more operations are done in parallel (as opposed to sequentially). Because, doing more in parallel means more components are employed to perform different operations at the same time. In addition, students would run out of space on schematic sheet 3, since students are not allowed to create macros to hide encapsulate sub-circuits. Therefore, students are strongly cautioned that they do not perform many operations in parallel to reduce the component count, hence the development time.

When students implement the machine player, they will first decide how much "thinking" or "intelligence" the block will have and the strategy and then partition Block 3 into subblocks and subsubblocks. They will also decide if the block gathers information in parallel or in sequence, by keeping in mind that the FPGA space is limited. Students will determine if they have inputs such as what is given in Figure 25 : There can be more inputs for more intelligence. **Students are reminded that they NOT change the outputs to be able to complete the project.** Students are advised that they design the subblocks of Block 3 starting with the easiest one, becoming more familiar with each subblock and completing the design with the hardest one.

How can students develop this block once the strategy and partitioning are decided about ? First, they must use all their knowledge on the term project, the tools (the Xilinx software and Digilent hardware) and techniques (top-down, team-based, core-based design) to develop the block. They must study their lecture notes and handouts. Students are suggested they design each assigned circuit first on paper then on their computer : Draw the schematic on paper, draw the schematic on the computer, simulate it and then download the project and play it. Then, start the work on the sub-circuit. Also, students are reminded that they **not** be afraid of iterating several times to get the final circuit on paper.

Note that machine vs. machine ppm project uses all eight displays and all 16 LED lights. That is, four aditional displays and eight additional LED lights are used compared with the human vs. human and human vs. machine projects. What is shown on these outputs is as follows :

- Displays 4 and 5 show Player 1 points
- Displays 6 and 7 show Player 2 points

- LED lights 8 to 11 show the next random digit
- LED lights 12 to 15 show the following random digit

In addition, students will use the keypad and 7-Segment Pmods to enhance the game and also debug their circuits faster. Finally, students can input a random digit from switches SW15-SW12 to also debug their circuits faster.