1. Overview
The goal of the term project is to develop a digital system, named Polytechnic Playing Machine, Ppm. It consists of digital circuits which allow a human player to play the game against the machine. The Ppm players play digits on four position displays after which they earn points. A player may skip a play, i.e. the player does not have to play the digit. No points are subtracted. Player 1 and Player 2 play the game until one wins the game. Then, the game is restarted by resetting.

There are three versions of the game at course web site: (i) version 1, which is the term project allowing a human player (Player 1) to play against the “machine” (Player 2), (ii) version 2, which allows two human players to play against each other and (iii) version 3, allows two machine players to play against each other. Below, the play description is given for the first version, the human versus machine version. These will be followed by short playing descriptions of the other two versions.

Ppm 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. Player 1 may also 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 Player 2: play or skip.

Playing the random digit on a position display is either playing it directly 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, a position display digit (PD_k) is added the random digit and the result, (PD_k + RD), is placed on the same display. But, the maximum value that can be played on a position display is (15)_{10}. That is, the sum of RD and the position display value cannot be greater than (15)_{10}. If the player has a sum that exceeds (15)_{10}, this situation is called display overflow. If the player tries to play it, the digit played is (PD_k - RD) - (16)_{10}. For example, if PD_2 is (F)_{10} and RD is 9, after the addition PD_2 is placed 8 = 9 + 15 - 16. Afterwards, the display blinks at a high rate, signalling there is a display overflow.

If a player plays the random digit directly, the player earns RD points. If a player adds two digits and there is no display overflow, the player earns the sum result points or more. If there is a display overflow, the player still earns points. The amount of points earned depends on the number of similar adjacent digits in a row on the position displays! The more adjacencies, the larger the points earned. The game ends when a player exceeds the maximum points which is (255)_{10}. All four displays blink, indicating there is a points overflow situation. The game is restarted by resetting Ppm. This will clear the displays and the players’ points (they become zero). The human player can reset the game anytime even if it is not an overflow situation.

In summary, the game for each player is about playing the random digit on a position display to earn points, more than the opponent. If a player chooses, the play can be skipped to give the turn to the opponent without losing points. The opponent receives a new random digit. Thus, winning the game is dependent on both chance and thinking.

2. Black-Box View and the Input/Output Relationship (Operation) of the Ppm
The term project circuit is a digital system! It allows a human player to play a game against the machine player. The black-box view of Ppm is shown in Figure 1. It has 13 inputs and 19 outputs. The inputs and outputs are connected to the Input/Output (I/O) devices on the FPGA board. One of them is the 25.175 MHz clock signal generated on the board. All the remaining I/O devices are switches, push buttons, 7-Segment displays, and LED lights and are used to enter decisions and to visualize the game situation. The Ppm I/O devices are shown in Figure 2. The detailed view of the Ppm inputs and outputs is shown in Figure 3.

Figure 1. The Ppm black box view.
All zero when the FPGA is downloaded/reset

7-segment displays

A display blinks fast if display overflow/subtraction

All displays blink if points limit exceeded

Figure 2. FPGA Board Input/Output device utilization of the Ppm Term Project.

Figure 3. Inputs and outputs of the Ppm term project.
2. The Input/Output Devices of the Ppm System for Version 1, the Term Project

The human player plays the game by using switches and push buttons on the FPGA board and the status of the game is shown by four 7-Segment displays, and eight LED lights. These are the Input/Output (I/O) devices needed for entering decisions and visualizing the game situation. The description of the inputs and outputs of the Ppm are shown on Table 1 and Table 2, respectively.

<table>
<thead>
<tr>
<th>Input device</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW1 - SW4</td>
<td>Player 1 position display select. Each switch selects a position display to play a digit</td>
</tr>
<tr>
<td>SW5 - SW7</td>
<td>Player 2 random digit input. Together with SW8, a BCD digit is input for the machine player</td>
</tr>
<tr>
<td>SW8</td>
<td>Player 1 add. When it is turned on, the current RD is added to the selected position display. In state 3, it can be used as the least significant bit of the random digit input to the machine player</td>
</tr>
<tr>
<td>BTN1</td>
<td>Player 1 wants to play. When pressed, it means the human player is ready to play</td>
</tr>
<tr>
<td>BTN2</td>
<td>Player 2 can play. When pressed, it means the human player wants the machine player to play. In two situations it is pressed. The human player presses it after examining the situation and wants the machine player to play. Otherwise, the human player presses it to skip the play, so it is machine player’s turn to play</td>
</tr>
<tr>
<td>BTN3</td>
<td>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</td>
</tr>
<tr>
<td>BTN4</td>
<td>Show players’ points. When pressed, Player 1 and Player 2 points are shown on the four displays. It can be pressed any time</td>
</tr>
<tr>
<td>Clock</td>
<td>The clock signal. It is a 25.175 MHz clock signal generated by an FPGA board circuit</td>
</tr>
</tbody>
</table>

Table 1: Input devices of the Ppm term project.

<table>
<thead>
<tr>
<th>Output device</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD1 - LD4</td>
<td>Random Digit LED light outputs. The random digit is BCD. They show the random digit of the current player</td>
</tr>
<tr>
<td>LD5</td>
<td>The current player adds. If it is on, it means the current player is adding RD to a position display</td>
</tr>
<tr>
<td>LD6 - LD8</td>
<td>The state of the game. They indicate where players stand. 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</td>
</tr>
<tr>
<td>Four 7-Segment displays</td>
<td>By default, the displays show position displays, PD3-PD0, on which digits are played. If Shpts is pressed, they show Player 2 points in Hex coding on the left two displays and Player 1 points in Hex coding on the right two displays. All four displays blink when there is a points overflow. A single display blinks fast if there is an addition or a display overflow. If a display is played with an addition and there is no display overflow, it blinks slowly</td>
</tr>
</tbody>
</table>

Table 2: Output devices of the Ppm term project.

Eight of the 13 inputs are from switches on the board: SW1 through SW8. Five of them are used by the human player to input decisions. The leftmost four switches, SW1-SW4, are the Player 1 position select switches to select a display to play the random digit. For example, the leftmost switch, SW1 is for which selects the leftmost position display, PD3; SW2 selects the next display PD2, etc. Note that if the switch handle is closer to you, it generates logic 1 and if the handle is away from you, it generates logic 0. The rightmost switch, SW8, is used to indicate an addition by the human player. Note that the rightmost four switches, SW5 through SW8 are also used to input a random digit, TRD3-TRD1, Pfadd, for the machine player to speed up the design of the machine player as described below.

To the right of the switches are four push buttons, labelled BTN1 through BTN4. All four push buttons are used by the human player to enter decisions or to see the game status. Push button 1, BTN1, is used to give the turn to Player 1. When it is pressed, it means Player 1 wants to play. Push button 2, BTN2, is used to give the play turn to Player 2. Push button 3, BTN3, resets the system, clearing all displays and points. Finally, when BTN4 is pressed, players’ points are shown on the four displays. Normally, when they are not pressed, they generate logic 0. As long as they are pressed, they generate logic 1.
Finally, the **Clock** input is a clock signal at 25.175MHz generated on the FPGA board. The clock is used to time and synchronize operations.

11 of the 19 outputs are connected to the four 7-segment displays named from left to right, PD3, PD2, PD1 and PD0. They show the four position display digits by default in Hexadecimal coding. They show players’ points in Hexadecimal coding when BTN4 (Shpts) is pressed. All four displays blink if a player wins the game, i.e. when there is a points overflow. These 11 outputs are CA through CG and A4 through A1. Note that Digilent calls the displays from left to right, A1, A2, A3 and A4 which is the opposite of our convention. Outputs, CA-CG, are active-low. The *Advanced Xilinx and Digilent Features* handout has details of the FPGA board 7-segment displays.

Parallel to the push buttons are eight LED lights, LD1 through LD8. All eight LED light are used to show the game status. The current random digit is shown on LED lights LD1 - LD4. The random digit is a BCD number. LED light 5, LD5, shows if an addition is wanted by the current player where LD5 emits light if it is wanted. LED lights LD6 - LD8 show the current state the circuit is in. That is, they indicate which step of the game the Ppm is in. The values are between 0 through 6. The state concept is explained in detail below.

### 3. The Ppm Playing Description

The two players, Player 1 (human) and Player 2 (machine), play until one wins the game (a points overflow). Then, the game is started by resetting. 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 each player is about:

- playing a random digit on a position display, which means playing directly/adding the random digit on/to a position display and storing the result on the same position display to earn points, more than the opponent, by seeking for adjacent identical digits.

- 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.

#### 3.1. An overview

When the FPGA chip is downloaded, all four 7-segment displays show zero. Both players start with zero points. All LED lights are blank. By default, the four 7-segment displays show position displays, i.e. the digits played. They show players’ points in Hex coding if BTN4 is pressed. The leftmost four LED lights show the random digit of the current player. Each player earns points depending on the digit value played on the display and the adjacency situation. The game ends if a player’s points exceeds \((255)\text{_{10}}\). This points overflow condition is shown by blinking the four displays. The game is restarted by resetting. A player can skip a play, giving the turn to the opponent. A display blinks at a high rate after a display overflow and after a subtraction. The human player can reset the game any time.

The game involves both guessing and thinking. Players try to play RD or PDk + RD. If there is no adjacency after an addition or subtraction, the player earns the result points. The player earns more if there are adjacent identical result digits as follows. If there is one adjacent digit, the earned points is \((8)\text{_{10}}\) plus two times the result digit played. If there are two adjacent identical digits, the player earns \((16)\text{_{10}}\) plus four times the result digit played. If the best case happens where there are three adjacent identical digits, the player earns \((32)\text{_{10}}\) plus eight times the result digit played. Figure 4 gives a number of random digit playing cases, one following another with respect to time right after the reset. In this sequence, the machine player is an imaginary one, not the one that is at the course web site.

The point earning system is such that if players add to the displays constantly, they would earn large points. But, eventually players would get display overflows, hence smaller reward points. A general playing strategy could be looking for adjacencies, not for large reward points all the time to gradually increase the display values while more adjacencies are obtained. Nevertheless, looking for adjacencies requires more effort on both players. Especially, for a machine player that means considerable amount of additional hardware.

An example of keeping displays small while maintaining adjacencies is the following: If the displays are \((A744)\), adding RD = 3 on position 2 would result in one adjacency and \((20)\text{_{10}}\) points: \((AA44)\). Adding RD = 3 on position 1 results in one adjacency and \((14)\text{_{10}}\) points: \((A774)\) and the displays are kept low and there would be more adjacency opportunities until there are display overflows. In summary, if players do not keep display digits small, displays would have large digits quickly. There would be frequent display overflows, meaning small digits on displays and small points earned! Thus, with extra points, players are encouraged to keep digits small, even if thinking about these requires more effort.
<table>
<thead>
<tr>
<th>RD</th>
<th>PD3</th>
<th>PD2</th>
<th>PD1</th>
<th>PD0</th>
<th>All points in <strong>decimal</strong> unless otherwise stated</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>There is no adjacent identical digit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The human player earns 5 points = (PD₁+RD)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The human player has 5 points</td>
</tr>
</tbody>
</table>

Player 1

| New | PD₁+RD = 5+0 = 5 is placed on position 1 |

Player 2

| New | Play RD on PD₀ : 5 is placed on position 0 |

<table>
<thead>
<tr>
<th>Player 1</th>
<th>Player 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td></td>
</tr>
<tr>
<td>PD₂+RD = 5+0 = 5 is placed on position 2</td>
<td>Play 5 on PD₃ : 5 is placed on position 3</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Player 1</td>
<td>Player 2</td>
</tr>
<tr>
<td>New</td>
<td></td>
</tr>
<tr>
<td>PD₃+RD = 9+5 = E is placed on position 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Player 1</th>
<th>Player 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td></td>
</tr>
<tr>
<td>PD₂+RD = 8+5 = D is placed on position 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Player 1</th>
<th>Player 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td></td>
</tr>
<tr>
<td>PD₁+RD = 5 + 8 = D is placed on position 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Player 2</th>
<th>Player 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td></td>
</tr>
<tr>
<td>PD₁+RD = D+0 = D is placed on position 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Player 1</th>
<th>Player 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td></td>
</tr>
<tr>
<td>PD₂+RD-16 = D+6-16 = 3 is placed on position2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4. Examples of digit plays after a reset.**
3.2. Ppm Play Sequence and Rules of Playing a Random Digit on Position Displays

This section lists the rules of the game. Naturally, the critical point in the game is when a random digit is considered for a play on position displays. However, we will list all the rules in an order starting from the reset. The following are the rules:

1) When the FPGA chip is downloaded, all position displays, players’ points and random digit lights are zero. The FPGA chip can be downloaded anytime, including when the game is going on.

2) The game can be reset anytime to restart the game. It can be reset even when the game is on. If a player exceeds the points limit, (255)_{10}, all four displays blink and the game stops. The game is restarted by resetting. When reset, all position displays, players’ points and random digit lights are zero. To reset the game, BTN3 (Reset) is pressed.

3) Player points can be seen by pressing BTN4 (Shpts) any time. On the left two displays Player 2 points is shown in Hex coding and on the right two displays Player 1 points is shown in Hex coding. The maximum player points is FF or (255)_{10}. The minimum limit is 0. If limit FF is exceeded all four displays blink until the Reset button is pressed.

4) When BTN1 (P1play) is pressed after the reset, the human player gets the turn to play. The human player receives a random digit automatically.

5) After receiving the random digit following 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.

6) If the human player plays the random digit, RD, \(0 \leq RD \leq (9)_{10}\) on position display \(k\), \(PD_k\), where \(0 \leq k \leq 3\) from right to left, points are added to the human player points:

- If the human player plays the random digit on a display directly, the result is that \(PD_k = RD\):
  - If \(PD_k = RD = 0\), the human player earns 0 points, even if there are adjacent zeros
  - Otherwise, the human player earns
    - RD points if there is no adjacent RD
    - \((2^R(D))\) if there is one adjacent RD
    - \((4^R(D))\) if there are two adjacent RD digits in a row
    - \((8^R(D))\) if there are three adjacent RD digits in a row

- If the human player adds the random digit to the display, the result is \((PD_k + RD)\):
  - If \((PD_k + RD)\) is 0, the human player earns 0 points, even if there are adjacent zeros
• If \((PD_k + RD) < (16)_{10}\), the human player earns
  • \((PD_k + RD)\) points if there is \textbf{no} adjacent \((PD_k + RD)\)
  • \((2 \cdot (PD_k + RD))\) if there is \textbf{one} adjacent \((PD_k + RD)\)
  • \((4 \cdot (PD_k + RD))\) if there are \textbf{two} adjacent \((PD_k + RD)\) digits in a row
  • \((8 \cdot (PD_k + RD))\) if there are \textbf{three} adjacent \((PD_k + RD)\) digits in a row

• If \((PD_k + RD) \geq (16)_{10}\), there is a display overflow. The played display blinks at a high rate. The human player plays digit \((PD_k + RD - (16)_{10})\), and earns
  • \((PD_k + RD - (16)_{10})\) points if there is \textbf{no} adjacent \((PD_k + RD - (16)_{10})\)
  • \((2 \cdot (PD_k + RD - (16)_{10}))\) if there is \textbf{one} adjacent \((PD_k + RD - (16)_{10})\)
  • \((4 \cdot (PD_k + RD - (16)_{10}))\) if there are \textbf{two} adjacent \((PD_k + RD - (16)_{10})\) digits in a row
  • \((8 \cdot (PD_k + RD - (16)_{10}))\) if there are \textbf{three} adjacent \((PD_k + RD - (16)_{10})\) digits in a row

After earning the points, the player points can exceed the limit of \((255)_{10}\), in which case there is a points overflow and all four displays blink. The game stops! To restart the game, it has to be reset as explained in (2) above.

7) After the human player plays the random digit, the player is given a chance to review the situation and check the players’ points. Then, the human player presses BTN2 (P2play), and the machine player gets the turn. The machine player receives its new random digit automatically.

8) After the human player skips the play or plays the random digit, the turn is given to the machine player. It has \textbf{two} options: play the digit on a display or skip. If the machine player skips, it does \textbf{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. In summary:
  • If the machine player plays its random digit, it earns points as described in point 6. The human player gets a new random digit when it is the turn.
  • If the machine player skips, it does not lose points. Again, the human player gets a new random digit.

9) After the machine player plays or skips, the human player is again given a chance to review the situation and check the players’ points. Then, the human player presses BTN1 (P1play), to play.

10) After the machine player skips the play or plays the random digit, the turn is given to the human player. The human player has \textbf{two} options: play the digit on a display or skip. If the human player skips, points is \textbf{not} lost. A new random digit given to the machine player. Again, we see that:
  • If the human player plays its random digit, it earns points as described in point 6. The machine player gets a new random digit when it is its turn.
  • If the human player skips, no points is lost. Again, the machine player gets a new random digit.

11) The game continues in this fashion until one of the players exceeds the points limit, \((255)_{10}\). The game is restarted from the beginning by pressing BTN3 (Reset).

### 3.3. Ppm Playing Modes

In order to clearly specify which player does what and when, we will introduce the concept of Ppm play \textbf{modes} and states. We will say that the Ppm is always in a specific mode at any time. There are three modes: the \textbf{Reset} mode, the \textbf{Player 1 play} mode and the \textbf{Player 2 play} mode. A mode can consist of submodes (steps or \textbf{states}). In each state, the system performs specific actions and waits for certain inputs/events (a push button press/the machine player completes its thinking) to occur to move to the next state. In some states, a move to another state happens without waiting. Certain state transitions change the mode to another mode.
We will show the modes on an operation diagram. The diagram shows the operation steps graphically with respect to time (Figure 5). In the diagram, each circle is a specific step (state) in the operation of the circuit. Each line that connects one circle to the other is taken when the input specified next to the line is true. This transition results in a number of operations in the new state. To refer to the steps easily, numbers are assigned to the steps. The assignment is from top to bottom. If a state has a 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. For example, states 0, 1, 3, 4 and 6 are such.

The current state is shown on the rightmost three LED lights on the FPGA board (see Table 2). Thus, if it is the Reset state, the lights show 0. If it is human player’s turn, the lights show 1-3. 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. When it is machine player’s turn, the LED lights show 4-6. 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 they show 6. These LED lights are useful in determining which player has won the game. When a player exceeds \((255)_{10}\), the four displays blink. If BTN4 (Shpts) is pressed the points do also blink. However, by looking at the points one cannot determine which player won the game easily. If LED lights are checked at this point, they would

**Figure 5. The operation diagram of Ppm.**

Player 1 can press BTN4, Shpts, in any state to see players’ points.

Player 1 can press BTN3, Reset, in any state to return to the Reset state, State 0.
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.

4. 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, depending on how much into the future it can project, can distribute the burden of a single (difficult) play to several plays. Students who are familiar with chess playing can relate to this concept.

One can think of a number of playing strategies for the Ppm game. Students need to select the strategy first before designing the machine player. They also need to keep in mind that all project circuits they will design, whether they are for the machine player or not, are related to the machine player. As they design these circuits, they need to think how they can be used for the machine player. Their experience during this time will help them select a number of strategies to implement by the time they start to design the machine player.

An intelligent machine player is the one that determines the largest adjacencies and earnings to win the game against the human player. One can come up with various types of intelligent players. A strategy that can be used by an intelligent machine player is the **aggressive playing strategy** where the player always plays on the position that results in largest earnings (reward points). The reason why this strategy can be for an intelligent machine player is that the player has to look for adjacencies for all the positions with direct playing and additions, which is not a simple task.

The drawback of this strategy is that looking for largest points now sometimes means not thinking about future plays. That is, the player plays aggressively, even though future earnings may be less due to display overflows. Thus, an aggressive player may earn large points initially, but may then be forced to earn fewer points in the future.

A strategy that is easy to implement is the **random playing strategy** where the player randomly selects positions and decides to add and subtract randomly as well. Thus, the random playing strategy can result in missing adjacencies. The **fixed playing strategy** means the player always plays in a certain fixed way, regardless of the situation, making its plays predictable. For example, the player plays in a round robin way on position displays with additions then subtractions. This strategy would also miss adjacencies often. It is clear that the random and fixed playing strategies are not intelligent machine player strategies. One can think of other less intelligent strategies. For example, the player plays on the rightmost position.

An intelligent machine player can be the one that plays on a position that results in the largest adjacency and also thinks about the future. This player would try not to play large numbers on displays, unless necessary, to avoid display overflows. Keeping display values as low as possible leads to adjacencies of large digits in the future. Furthermore, a sophisticated machine player can have several playing strategies, using each one at different times based on a **super strategy**. The super strategy indicates which strategy will be used when. For example, a machine player can have two strategies such that if it is ahead, it plays randomly and when it is behind, it plays aggressively. The course web site machine player uses what we can call a **calculated super playing strategy** whose goal is that the display digits remain small as much as possible. Therefore, it tries to maximize its earnings by going for largest adjacencies, not for largest earnings. As shown below, not always largest adjacencies mean largest earnings.

If the machine player is not so intelligent, it can have fewer and simpler strategies. Students must note that a single strategy may not be enough. At least two playing strategies might be needed to deal with situations when there are two or more equally good positions to play. For example, if playing on two positions results in the same reward points, which one will be selected? One solution is that the secondary strategy is the fixed strategy that always the leftist of these two positions is selected. Another solution is that the secondary strategy would randomly select one of these two positions.

Finally, students are suggested that they determine and think about their machine player strategy now, not when they design the machine player later in the semester. As mentioned before, all term project circuits that will be designed are usable by the machine player and thinking how those circuits can be used can help choose and design the strategy with ease later on.

5. The Course Web Site Machine Player Playing Strategies
The course web site machine player is an intelligent one with four playing strategies. It thinks ahead and plays so that its earnings currently and in the future are maximized as much as possible. Its primary playing strategy is looking for largest adjacencies when it is ahead. When, there are zeros on the displays and it is ahead, it tries to play on zero
positions for future adjacencies. When it is behind, it plays aggressively, going for largest points (not largest adjacencies). Finally, if there are equally good playing possibilities, it uses the fixed playing strategy where it plays on the rightmost of these playable positions (Figure 6).

The course web site machine player first determines if the displays are zero (perhaps because the game has just started as the displays would have zeros due to the reset) and RD is zero:

- If yes, it skips the play.
- If no, it checks if RD is 0.
  - If yes, it checks to see if there are adjacencies and it plays on the position with the largest adjacency. If there are several positions with the same largest adjacency, it plays on the rightmost of these largest

Figure 6. The playing strategy of Player 2 machine player at the course web site.
adjacency positions. If playing directly and adding result in the same largest adjacency, playing directly is chosen (Action 3 in Figure 4).

- If there are no adjacencies:
  - It checks if it has more points than the human player.
    - If yes, it skips.
    - If no, it plays on the position with the maximum points, i.e., it plays aggressively. If there are several positions with the same largest amount of 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 2).

- If RD is not 0:
  - If the machine player is ahead of the human player:
    - It checks if there is at least one adjacency. If yes, it plays on the position with the largest adjacency. If there are several positions 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 3).
    - If no adjacency, it checks if there are two one-digit adjacencies or one three-digit adjacency:
      - It skips if there are two one-digit adjacencies or one three-digit adjacency. That is, it skips if one of the following two is true:
        - If display 0 is equal to display 1 and display 2 is equal to display 3.
        - If all displays are equal to each other.
      - Otherwise, it checks if all displays are non-zero:
        - If, a display is zero, it plays there. If several displays are zero, it plays on the rightmost of these zero displays by playing directly (Action 0).
        - Otherwise, it plays on the smallest display position with an addition so that adjacencies on the displays are not disturbed. Note that, not disturbing the adjacencies by checking for equalities of displays 0 and 1 and displays 2 and 3 and by checking the equality of all the displays is done so that these adjacencies can be used in the future. This is possible, if for example, the random digit is 0 next time, in which case, adding it to a display with an adjacency would result in a lot of points (Action 1).
  - If the machine player is behind the human player in terms of points, it tries to earn the maximum points possible, even if it means the display digits become large. It compares the earning from the largest adjacency with the largest earning without any regard to adjacency. Sometimes, the largest adjacency may not give the largest points earned for a play. For example, if the displays are 55AF and the random digit is 5, playing on position 1 with an addition will result in (30)_{10} points with a single adjacency: 55FF. This earning is larger that the play on position 1 by playing directly which gives (20)_{10} points with two adjacencies: 555F. It plays on the largest earning position. If there are several positions 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 3).

6. The Human vs. Human Playing Description
This Ppm version is version 2 where two human players play against each other. Both human players use SW1-SW4 to select a position, and SW8 to add the random digit. The rules of the game are identical to the human vs. machine game rules.

7. The Machine vs. Machine Playing Description
This Ppm version is version 3 where two machine players play against each other. The switches used by the human player SW1-SW4, and SW8 are not used. The human player presses BTN1 to let the first machine player to play and BTN2 to let the second machine player to play. BTN3 and BTN4 are used as before. The rules of the game are identical to the human vs. machine game rules.

The Player 1 machine player uses the playing strategy described in Figure 6. The Player 2 machine player uses the playing strategy shown in Figure 7. Player 2 plays aggressively, always seeking for the largest reward points. It does
The decision to not to skip, is to fit the two machine players and the other blocks on the FPGA chip. Finally, when there are positions with equal maximum earnings, the rightmost of these positions is selected, which is a fixed playing strategy.

**Figure 7. The playing strategy of Player 1 machine player at the course web site.**

At the end of the semester, students will replace this machine player (Player 3) with their own machine player and submit their project. If their machine player wins against the machine player designed by the professor (Player 1), they will earn 10% of their lab grade.

### 8. The Ppm Digital System

We have studied the black-box view of the Ppm circuit. We know that there are 13 inputs and 19 outputs and also some of the circuits are **sequential** circuits. Since it has a large number of inputs and so needs to be partitioned into pieces. How can we go about the partitioning of the Ppm system? We have discussed how to partition a complex circuit into blocks: we obtain a list of major operations the hardware has to perform, by also considering the design goals. Then, we create a block for each major operation.

We know that one of the blocks will be the machine play block for the machine play major operation. The others? The others can be the following: a block for points calculation, a block for the human player, a block to interface to the Input/Output devices and a block to check the play. Note that thinking of these major operations requires 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.

Will we have just five blocks? No! Because the Ppm term project circuit is a **digital system**! A digital system performs a complex series of operations based on its inputs and needs to have a circuit that keeps track operations and determines what to do next given the inputs and past operations. This special circuit is the control unit that controls the rest of the digital system! Our Ppm system is digital system that plays a game for which it follows the predetermined game rules and responds to inputs and past events! Therefore, we will have one more block, the control unit.

In digital system terminology, all the block other than the control unit placed in a super block called **datapath** (data unit). Then, the first partitioning of any digital system must result in a Control Unit and a Data Unit. The partitioning of the Data Unit is based on major operations performed. These major operations are the one we specified above on this page: interacting with input/output devices on the FPGA board, handling human player’s plays, checking the correctness of the play, points calculation and machine playing. Before we discuss these partitioning in detail, first we study digital systems in general.

### 8.1. Introduction

A digital system consists of digital circuits. It performs sequences of simple operations (called **microoperations**). It is organized into **data** and **control** units (Figure 8). The control unit determines which microoperation happens when. The data unit performs the microoperations. An example of a digital system is the Central Processing Unit (the **CPU** or processor) of a computer. Note that every computer today has a CPU and the microprocessor chip contains the CPU. For example, if one buys a PC with a Pentium IV microprocessor, the microprocessor chip has the Intel Pentium IV CPU (in addition to cache memories, buffers, etc.).

**Figure 8. A large scale view of a digital system.**
Digital system based design can be done top-down and also block-based where initial design steps are simple with a few details: the targets are large blocks, not gates and flip-flops. In addition, digital system design can be core-based where a part of the circuit, the “core,” can be “licensed” 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.

8.2. 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 kinds of hardware: registers, Arithmetic Logic Units (ALUs) and buses (Figure 8). Registers are needed to keep data temporarily, ALUs perform additions, subtractions, shifts, ANDs, ORs, etc. and 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.

While the data unit has the bulk of the logic, i.e. the most number of gates and flip-flops, it is easier to design than the control unit since the data unit is highly regular. That is, a data unit has pieces of hardware repeated many times. For example, an ADDer is a repetitive set of smaller addition blocks, the multiplier is similar and so are registers. Another example is that a 4-bit compare circuit 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.

8.3. The Control Unit
The control unit controls the data unit. It controls 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, no fan-in restrictions, etc. However, the reality is just the opposite. A simple example to show the point is that a circuit that logically works may not work due to a Static-1 or Static-0 hazard (glitch) created by gate delays. And when one thinks of the data unit of a real digital system with many blocks of circuits, the task of determining the timing, locally and globally is enormous.

The control unit has status signals as inputs and control signals as outputs (Figure 9). The circuit that generates control signals in the control unit is called “sequencer.” It is the one that determines the sequence of microoperations, hence the name, sequencer. The sequencer goes through steps, known as states. The sequencer may “jump over” steps also. But, it knows exactly which step it is in by using a state register. That is, the state register indicates the state the digital system is in at any moment. Based on the state 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 quite complex. The irregularity is due to the large amount of random logic (gate networks and flip-flops).

The control unit not only has to be free from timing problems itself (glitch-free), but also has to make sure that the data unit operates free from timing problems. To be able to generate such signals, it has to be implemented in such a way that as many gate delays as possible are accounted for. Thus, a control unit is partitioned only once to Control Signal Generation and Next State Generation Subblocks to make sure that all timing problems are always under constant check during the design.

![Figure 9. A general view of the sequencer.](image-url)
8.4. Control Unit Hardware Design

The sequencer is implemented by using **hardwiring** and **microprogramming**. Hardwiring generates control signals by gate networks and flip-flops, that is for each control signal, there are gates and flip-flops, Figure 10(a). Gate networks are also needed to generate next state signals. Overall, hardwiring leads to a tremendous number of gates and flip-flops. Since gate networks are random logic, their development is very time consuming. The design, test, modification, manufacture and upgrade of huge amounts of random logic is monumentally difficult.

![Diagram of a hardwired sequencer](image1.png)

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

![Diagram of a microprogrammed sequencer](image2.png)

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

In order to reduce the physical size of the gate networks, today one or more Programmable Logic Array (PLA) chips are used. Note that, since the Ppm playing engine digital system is not too complex, the control unit of Ppm is implemented by hardwiring. One can see schematic 1 of the Ppm project that gates and flip-flops are used to generate the control signals.

A microprogrammed sequencer generates control signals by using a special memory in the control unit, with very little accompanying random logic as shown in Figure 10(b). Thus, microprogramming simplifies the control unit development substantially. Today, Read Only Memory (ROM) chips are used as the special memory in the sequencer. Whether it is hardwiring or microprogramming, the first task in the design of a sequencer is to lay out the input-output relationship of the sequencer. That is, based on the status signals (inputs to the control unit), control signals (outputs from the control unit) are generated. This input-output relationship must be precise, which can be an exhaustive job. Two techniques are used to describe the sequencer precisely: the FSM and HDL techniques.

The FSM technique treats the sequencer as a “state machine” going through states based on the sequence of past inputs. Every state precisely describes which control output is 1 at any moment. Accounting for every clock period is accomplished by indicating which microoperation happens when (in which state) exactly. An introduction to the state concept, the FSM technique and the description of the sequencer of the Ppm digital system is given in the Term Project handout.

9. The Ppm Partitioning

We are now ready to partition the Ppm digital system into blocks (Figure 11):

- The Control Unit Block, controlling the datapath, the five blocks below : Block 1
- The Input/Output Block, interfacing to the Input/Output devices of the FPGA board : Block 2
- The Human Play Block, handling human player’s play : Block 3
- The Play Check Block, controlling the display operations based on the rules : Block 4
- The Point Calculation Block, calculating the new points for both players : Block 5
- The Machine Play Block, determining the machine player play : Block 6

The Control Unit design starts with detailing the values of signals, including “control signals” precisely in every state of the operation diagram. We know that the operation diagram describes the flow of the game which is the input-out-
put relationship with respect to time. The diagram shows the response of our digital system for expected and unexpected inputs/events at each state. It shows the correct input sequences and states together graphically where in each state, the value of every signal is either known or determinable. Some of these signals are, in fact, control signals that the Control Unit generates to control the datapath. The Control Unit is then partitioned into Control Signal Generation and Next State Generation subblocks as discussed above.

In order to determine control signal values, hence the Control Unit circuits, we will use the Finite State Machine (FSM) technique, not the longer textual (English) description nor the HDL technique. The FSM technique requires that we derive a FSM state diagram, which not only describes the operation of the machine, but also the hardware needed and which microoperation happens when. Thus, we move from the specification (of the operation of the machine) phase to the implementation (of the machine) phase fast. The FSM design and FSM diagram will not be covered in this course. However, there is more coverage of the Ppm Control Unit in the Term Project Handout.

Students are reminded that throughout the semester, they are guided for a top-down project development style where initial design steps involve simple concepts with a few details. Targets are large blocks, not gates and flip-flops. As blocks become clearer, more details are dealt with: blocks are partitioned into smaller blocks. Eventually, gates and flip-flops are considered for implementation. As mentioned above, the playing engine consists of six blocks. These blocks are fairly complex and so must be divided into subblocks, subsubblocks and so on.

Additionally, students will be guided to design the playing engine as a digital system. A digital system is a complex circuit. It performs microoperations that generate the outputs shown in Figure 10. To design the complex playing engine all at once is prohibitive as there are many details to deal at any moment. Having a part of the digital system (the control unit) controlling the rest of the digital system (microoperations in the data unit, or datapath) simplifies its design. Readers are referred to the Appendix section of this handout for a discussion of digital systems.

Finally, CS2204 Experiments will have incorporated the core-based design technique. Due to time-to-market constraints, hardware engineers license a part of the circuit, called the core (since it is often the most complex part) from another company under a license agreement. They save time since they design and attach only the remaining parts of the circuit. In that context, students will not design all the blocks. They will be given core circuits and develop the remaining circuits. The core circuits are blocks 1, 2, 3, 4 and a portion of Block 5.

**Figure 11. Block partitioning of the Ppm term project.**

10. Digital System Design at Polytechnic

Digital system based design is substantially emphasized in academic curricula. For example, at Polytechnic University, fundamentals of digital system design are distributed to CS2204 and CS2214 (Computer Architecture and Organization). CS2214 emphasizes digital system design by introducing how to design a small computer, including its (micro)processor. It covers both control and data units of digital systems. While the 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 Polytechnic (CS613, EL549, among others) introduce other design techniques and tools as well for handling the complexity. They allow engineers to move freely between different levels of abstraction: superblocks, blocks, subblocks, subsubblocks, chips, gates/flip-flops, and transistors.

One technique often used for educational purposes is the finite-state machine (FSM) technique, described above. A “state diagram” with finite number of “states” is drawn to show which microoperation happens when and how. Thus, a FSM state diagram describes a digital system. In CS2204, the FSM technique is used, which is also why the
name of the course is “Digital Logic and State Machine Design.” The FSM technique is also used in CS2214. Note that these FSM states are the same states mentioned in Figure 5. The concept of state will be introduced in detail in the second half of the semester when sequential circuits are discussed.

The design approach now in use by industry is Hardware Description Language- (HDL-) based. An HDL has statements (text lines) to describe hardware in detail: the abstraction of hardware. In this technique, an HDL program describes a digital system. Designers work on HDL programs and delay building the prototype until they are sure their HDL programs are correct. Most commonly used HDLs are VHDL (introduced in EL549) and the Verilog HDL. HDLs 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.

FSM and HDL 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 start the prototype-on-breadboards phase quickly and confidently. Thus, students are strongly urged to understand this handout and the Term Project Handout not only to complete the project but also to learn an industry method and prepare for CS2214.