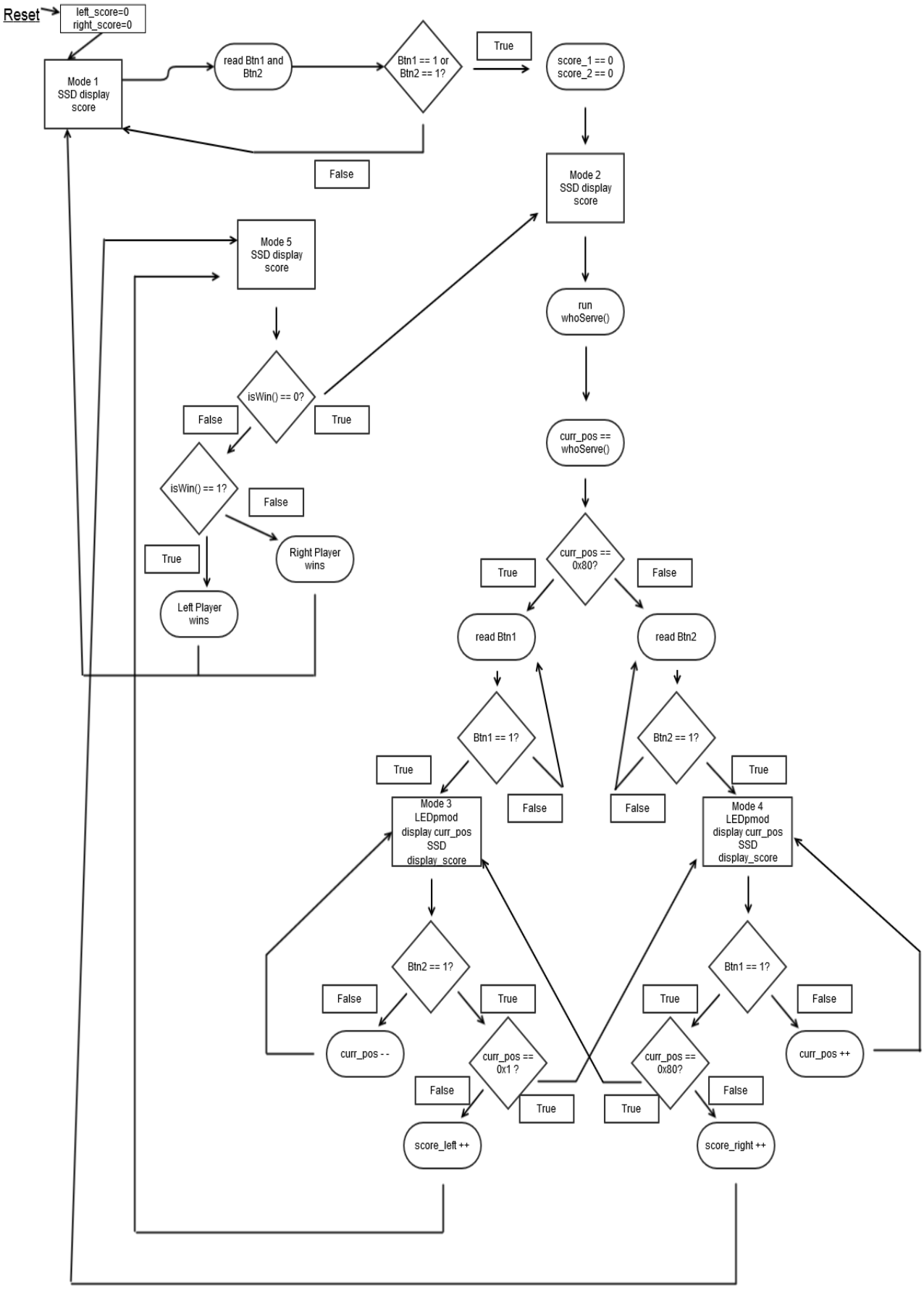
CPEG 222
Project 1 Flow Chart

Katie Black

Byron Lambrou


Reset
left_score=0
right_score=0
Mode 1
SSD display
score
read Btn1 and
Btn2
Btn1 == 1 or
Btn2 == 1?
True
False
score_1 == 0
score_2 == 0
Mode 2
SSD display
score
run
whoServe()
curr_pos ==
whoServe()
curr_pos ==
0x80?
True
False
read Btn1
read Btn2
Btn1 == 1?
Btn2 == 1?
True
False
Mode 3
LEDpmod
display curr_pos
SSD
display_score
Mode 4
LEDpmod
display curr_pos
SSD
display_score
Btn2 == 1?
Btn1 == 1?
curr_pos - -
curr_pos ++
curr_pos ==
0x1 ?
curr_pos ==
0x80?
score_left ++
score_right ++
Mode 5
SSD display
score
isWin() == 0?
isWin() == 1?
Right Player
wins
Left Player
wins

Variables:

-score_left, score_right → Integer value that represents the score from the user, increases by one each time the user hits the ball back at the correct time.
-curr_pos → Integer value that represents the current position of the ball. The light furthest to the left represents position 0, and the position furthest to the right represents position 7.

Inputs:
-btn1, btn2 onboard the microcontroller board

Outputs:
-Pmod 8 LED display
-2 Pmod SSD displays
-4 onboard LED display

Modes

1. Mode 1 – Initial/Game Over
    a. Waits to start the game
    b. Displays score from previous game, or 00 for both if just reset
2. Mode 2 – Ready to Serve
    a. Determines which player should serve, shows the ball at their end
    b. Waits for a player to hit the button to serve the ball
3. Mode 3 – Right shift
    a. Ball moves from left to right.
    b. The right player must hit the ball when it is at the right spot
4. Mode 4 – Left shift
    a. Ball moves from right to left
    b. The left player must hit the ball when it is at the right spot
5. Mode 5 – Violation
    a. Determine if any player has won the game.
    b. If neither player has won, go back to mode 2.

Questions to Address:

Question 1: Please implement a function “isWin()” in C code. The function determines, according to the winning policy, whether there is a winner and who is the winner. It takes the scores of both players as inputs and returns a value: 0 indicates “no winner yet”, 1 indicates “left side wins”, and 2 indicates “right side wins”.

```c
int isWin( unsigned int score_left, unsigned int score_right){
  //Function to decide if one of the players has won the game.
  //The rule for winning is as follows
  //If either player has a score of 11, and the other player has a score of <10,
  //this player wins the game.
  //If both players have a score of 10 or more, then if either player has a score
  //which is 2 points greater than the other, they win the game.
  //Return 0 if neither player has won, 1 if left side has won, 2 if right side has won
  int decision=0;
  if (score_left<11 && score_right<11){
  decision=0;
  }
  if(score_left==11 && score_right<10){
  decision=1;
  }
  if(score_right==11 && score_left<10){
    decision=2;
  }
  if (score_left>=10 && score_right>=10){
    if((score_left-2)>=score_right){
      decision=1;
    }
    else if((score_right-2)>=score_left){
      decision=2;
    }
    else{
      decision=0;
    }
  }
  return decision;
}
```

Question 2: Please implement a function “whoServe()” in C code. The function determines, according to the service rule, who should serve the current round. It takes the scores of both players as inputs and returns a value representing Pmod8LD display status: “0x80” means the left player should serve and “0x1” means the right player should serve.

```c
int whoServe(unsigned int score_left, unsigned int score_right){
    //Function to determine who serves the ball next.
    //Returns, according to the services rules, who serves.
    //Returns 0x80 if the left player should serve, and
    //returns 0x1 if the right player should serve, which correspond to
    //the pmod8LED status for each serve case.
    //The left player always serves for the first 2 turns, then the right
    //player for two turns, until the players both have ten points or more,
    //then each player serves the ball for only one point in their turn.
    int sum = score_right+score_left;
    int to_serve;
    if((sum)<20){
        if(((sum/2)%2)==0){
            //then left player serves
            //On turn 1,2, sum=0,1 respectively, while left serves,
            //and ((sum/2)%2)==0.
            //On turn 3,4, sum=2,3 respectively, while right serves,
            //and ((sum/2)%2)==1
            //On turn 5,6, sum=4,5 respectively, while left serves,
            //and ((sum/2)%2)== 0.
            //Thus, when ((sum/2)%2)==0, left serves and vice versa.
            to_serve=0x80;
        }
        else{
            //then right player serves
            to_serve=0x1;
        }
    }
    if((sum)>=20){
        if(sum%2==0){
            to_serve=0x80;
        }
        else{
            to_serve=0x1;
        }
    }
    return to_serve;
}
```

Question 3: Each Pmod SSD is able to display two digits at the same time. For each digit, seven inputs are required to control all the seven segments. So it seems that fourteen inputs are needed for displaying two digits. However, if you check the pins of your Pmod SSD, you will find that there are only 8 inputs available for each Pmod SSD (despite GND and VCC). Using these limited 8 inputs, how to display both digits at the same time? Please write down your solution and provide a brief explanation.

The SSD board's has seven anode pins, which are shared, meaning that the anode for segment 1 corresponds to the LED for the first SSD's anode 1, and to the LED for the second SSD's anode 1. The SSD's LEDs are all tied together in a "common cathode" circuit configuration. This common cathode means that the for one value of the cathode pin, the pin inputs for the 7 anodes go to the first SSD while the second SSD is off, and that for the other value of the cathode pin, the pin inputs for the 7 anodes go to the second SSD while the first SSD is off. So, in the span of one cycle, we can set the pins to be equal to our desired value for SSD 1 and set the cathode to correspond to SSD 1, then set the pins to be equal to our desired value for SSD 2 and set the cathode to correspond to SSD 2. By rapidly doing this, in a repeating, continuous fashion, it will give the illusion to the human eye that both digits are constantly illuminated with the correct values so long as the rate at which it is done is greater than 60Hz. This setup forms a multiplexing display for the SSDpmod.