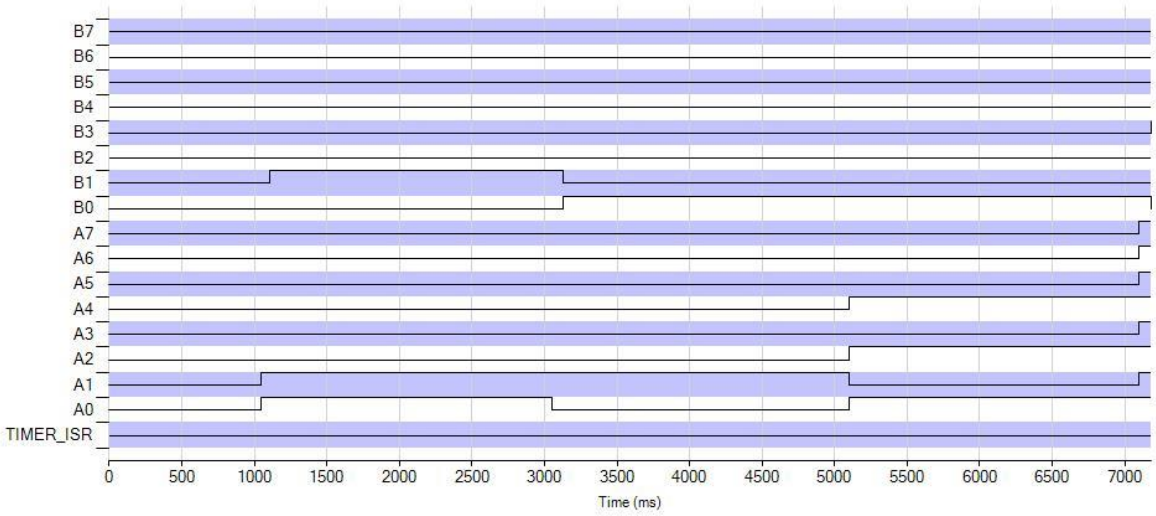
Souradeep Bhattacharya

EE120B HW 1

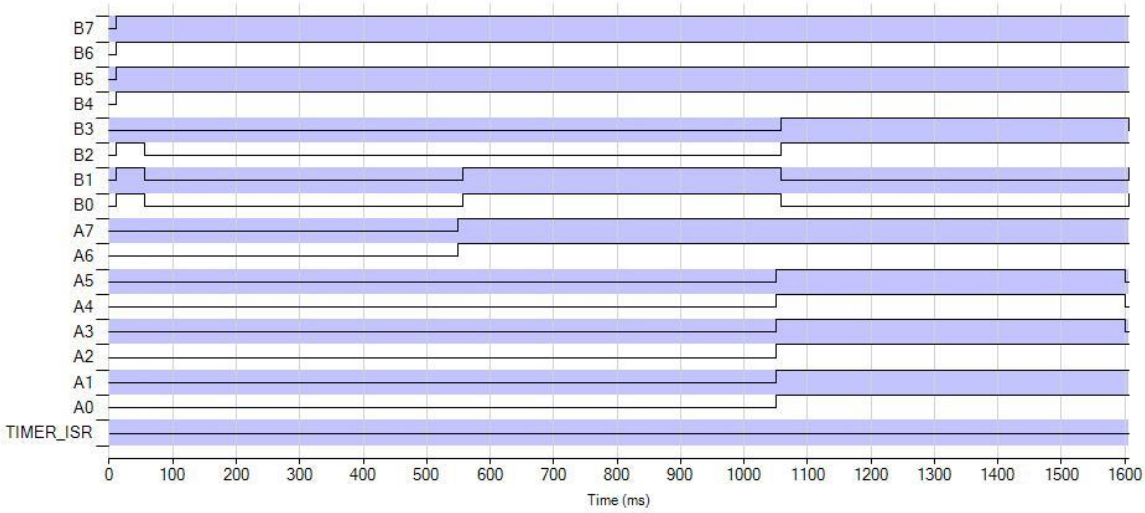
Problem 1

```
#include "rims.h"

void main()
{
   while (1)
   {
      char input = A;
      char count = 0;
      char maxCount = 0;
      char bitMask = 1;
      char i = 0;
      for(; i < 8; i++)
      {
          if((input & bitMask) == bitMask)
          {
              count++;
          }
          else
          {
              if (count > maxCount)
              {
                  maxCount = count;
              }
              count = 0;
          }
          bitMask = bitMask << 1;
      }
      if(count > maxCount)
      {
          maxCount = count;
      }
      B = maxCount;
   }
}
```

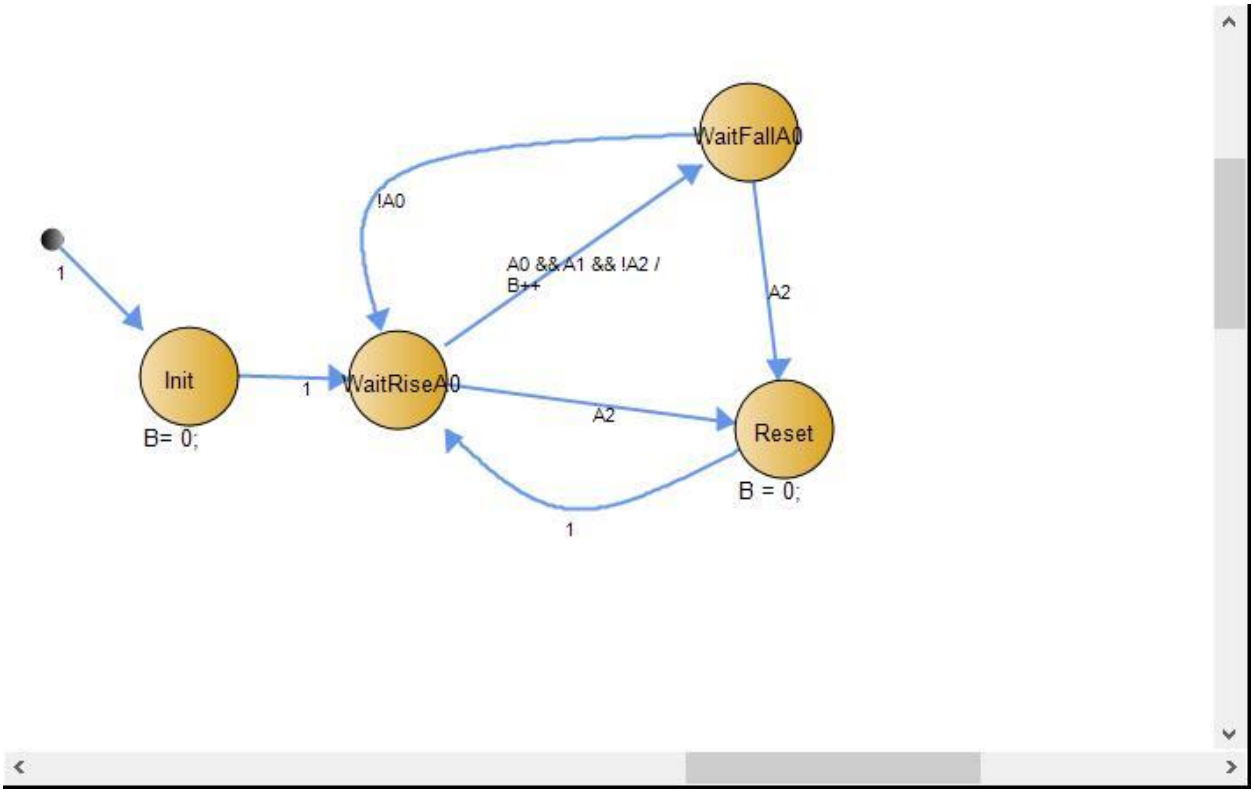

B7
B6
B5
B4
B3
B2
B1
B0
A7
A6
A5
A4
A3
A2
A1
A0
TIMER_ISR
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
5500
6000
6500
7000
Time (ms)

Problem 2

```
#include "rims.h"


void main()
{
   while (1) {

      char sum = 0;
      char A1 = A & 0x03;
      char A2 = (A & (0x03 << 2)) >> 2;
      char A3 = (A & (0x03 << 4)) >> 4;
      char A4 = (A & (0x03 << 6)) >> 6;
      sum = A1 + A2 + A3 + A4;
      B = 0xF0 | sum;
   }
}
```

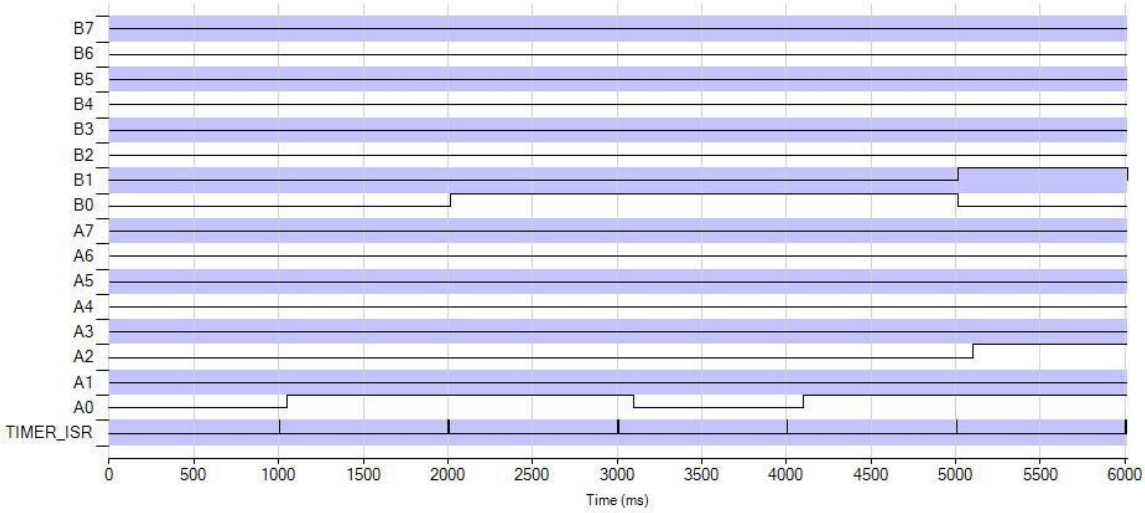

B7
B6
B5
B4
B3
B2
B1
B0
A7
A6
A5
A4
A3
A2
A1
A0
TIMER_ISR
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
Time (ms)

Problem 3

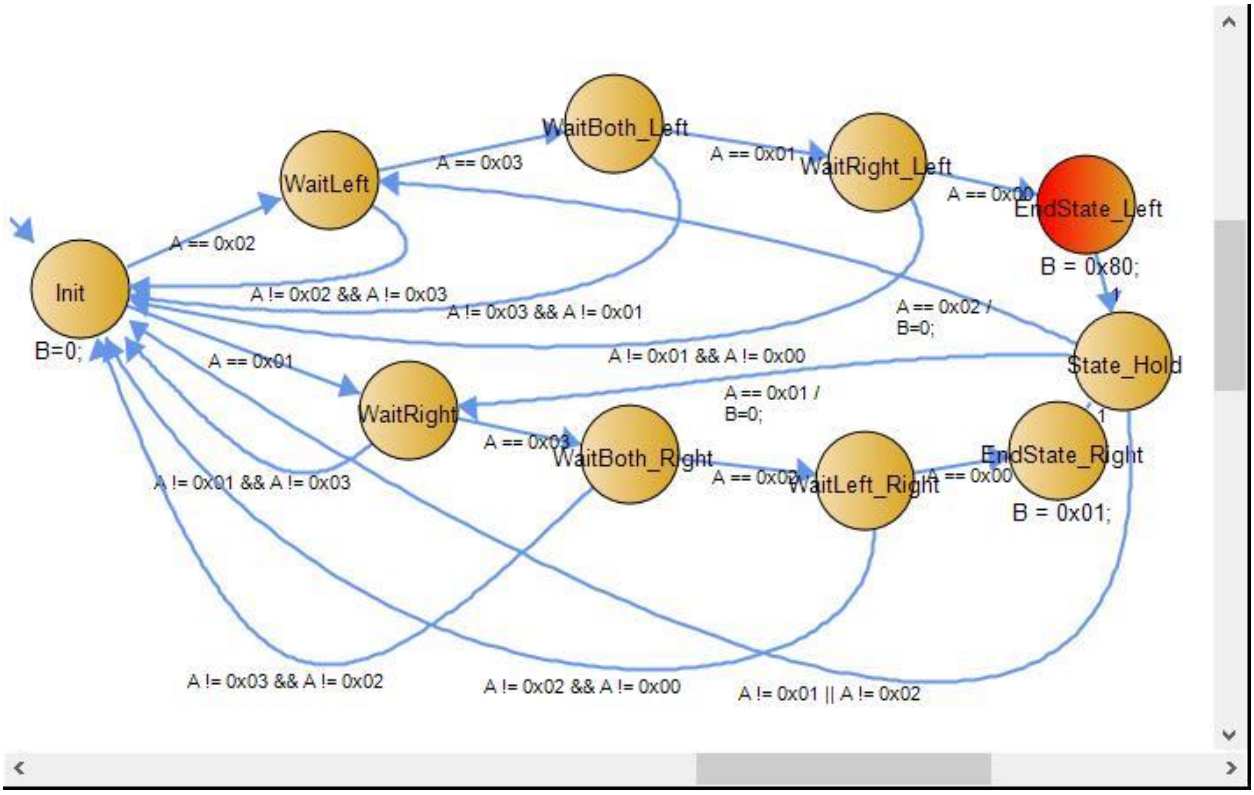


```
#include "rims.h"


enum SM1_States { SM1_Init, SM1_WaitRiseA0, SM1_WaitFallA0, SM1_Reset }
SM1_State;

TickFct_State_machine_1() {
   switch(SM1_State) { // Transitions
      case -1:
         SM1_State = SM1_Init;
         break;
         case SM1_Init:
         if (1) {
            SM1_State = SM1_WaitRiseA0;
         }
         break;
      case SM1_WaitRiseA0:
         if (A0 && A1 && !A2) {
            SM1_State = SM1_WaitFallA0;
            B++;
         }
         else if (A2) {
            SM1_State = SM1_Reset;
         }
         break;
      case SM1_WaitFallA0:
         if (A2) {
            SM1_State = SM1_Reset;
         }
         else if (!A0) {
            SM1_State = SM1_WaitRiseA0;
         }
         break;
      case SM1_Reset:
         if (1) {
            SM1_State = SM1_WaitRiseA0;
         }
         break;
      default:
         SM1_State = SM1_Init;
   } // Transitions

   switch(SM1_State) { // State actions
      case SM1_Init:
         B= 0;
         break;
      case SM1_WaitRiseA0:
         break;
      case SM1_WaitFallA0:
         break;
      case SM1_Reset:
         B = 0;
         break;
      default: // ADD default behaviour below
```

```
      break;
   } // State actions

}

int main() {

   SM1_State = -1; // Initial state
   B = 0; // Init outputs

   while(1) {
      TickFct_State_machine_1();
   } // while (1)
   return 0;
} // Main
```


B7
B6
B5
B4
B3
B2
B1
B0
A7
A6
A5
A4
A3
A2
A1
A0
TIMER_ISR
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
5500
6000
Time (ms)

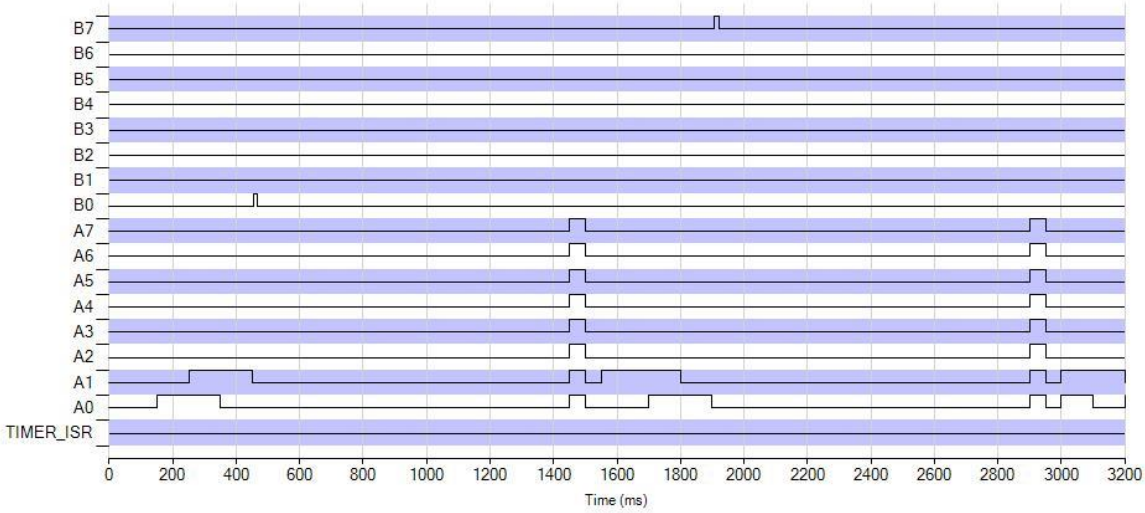
Problem 4


Init
B=0;
WaitLeft
WaitBoth_Left
WaitRight_Left
EndState_Left
B = 0x80;
State_Hold
WaitRight
WaitBoth_Right
WaitLeft_Right
EndState_Right
B = 0x01;
A == 0x02
A == 0x03
A == 0x01
A == 0x00
A != 0x02 && A != 0x03
A != 0x03 && A != 0x01
A == 0x02 /
B=0;
A != 0x01 && A != 0x00
A == 0x01 /
B=0;
A == 0x01
A == 0x03
A == 0x02
A == 0x00
A != 0x01 && A != 0x03
A != 0x03 && A != 0x02
A != 0x02 && A != 0x00
A != 0x01 || A != 0x02

```
#include "rims.h"

enum SM_States { SM_Init, SM_WaitLeft, SM_WaitRight, SM_WaitBoth_Right,
SM_WaitBoth_Left, SM_WaitRight_Left, SM_WaitLeft_Right, SM_EndState_Left,
SM_EndState_Right, SM_State_Hold } SM_State;

TickFct_State_machine_1() {
   switch(SM_State) { // Transitions
      case -1:
         SM_State = SM_Init;
         break;
         case SM_Init:
         if (A == 0x02) {
            SM_State = SM_WaitLeft;
         }
         else if (A == 0x01) {
            SM_State = SM_WaitRight;
         }
         break;
      case SM_WaitLeft:
         if (A == 0x03) {
            SM_State = SM_WaitBoth_Left;
         }
         else if (A != 0x02 && A != 0x03) {
            SM_State = SM_Init;
         }
         break;
      case SM_WaitRight:
         if (A == 0x03) {
            SM_State = SM_WaitBoth_Right;
         }
         else if (A != 0x01 && A != 0x03) {
            SM_State = SM_Init;
         }
         break;
      case SM_WaitBoth_Right:
         if (A == 0x02) {
            SM_State = SM_WaitLeft_Right;
         }
         else if ( A != 0x03 && A != 0x02) {
            SM_State = SM_Init;
         }
         break;
      case SM_WaitBoth_Left:
         if (A == 0x01) {
            SM_State = SM_WaitRight_Left;
         }
         else if (A != 0x03 && A != 0x01) {
            SM_State = SM_Init;
         }
         break;
      case SM_WaitRight_Left:
         if (A == 0x00) {
            SM_State = SM_EndState_Left;
         }
```

```
         else if (A != 0x01 && A != 0x00) {
            SM_State = SM_Init;
         }
         break;
      case SM_WaitLeft_Right:
         if (A == 0x00) {
            SM_State = SM_EndState_Right;
         }
         else if (A != 0x02 && A != 0x00) {
            SM_State = SM_Init;
         }
         break;
      case SM_EndState_Left:
         if (0) {
            SM_State = SM_Init;
         }
         else if (1) {
            SM_State = SM_State_Hold;
         }
         break;
      case SM_EndState_Right:
         if (0) {
            SM_State = SM_Init;
         }
         else if (1) {
            SM_State = SM_State_Hold;
         }
         break;
      case SM_State_Hold:
         if (A == 0x01) {
            SM_State = SM_WaitRight;
         }
         else if (A == 0x02) {
            SM_State = SM_WaitLeft;
         }
         else if (A != 0x01 || A != 0x02) {
            SM_State = SM_Init;
         }
         break;
      default:
         SM_State = SM_Init;
   } // Transitions

   switch(SM_State) { // State actions
      case SM_Init:
         B=0;
         break;
      case SM_WaitLeft:
         break;
      case SM_WaitRight:
         break;
      case SM_WaitBoth_Right:
         break;
      case SM_WaitBoth_Left:
         break;
      case SM_WaitRight_Left:
         break;
```

```
      case SM_WaitLeft_Right:
         break;
      case SM_EndState_Left:
         B = 0x80;
         break;
      case SM_EndState_Right:
         B = 0x01;
         break;
      case SM_State_Hold:
         break;
      default: // ADD default behaviour below
      break;
   } // State actions

}

int main() {

   SM_State = -1; // Initial state
   B = 0; // Init outputs

   while(1) {
      TickFct_State_machine_1();
   } // while (1)
} // Main
```


B7
B6
B5
B4
B3
B2
B1
B0
A7
A6
A5
A4
A3
A2
A1
A0
TIMER_ISR
0
200
400
600
800
1000
1200
1400
1600
1800
2000
2200
2400
2600
2800
3000
3200
Time (ms)