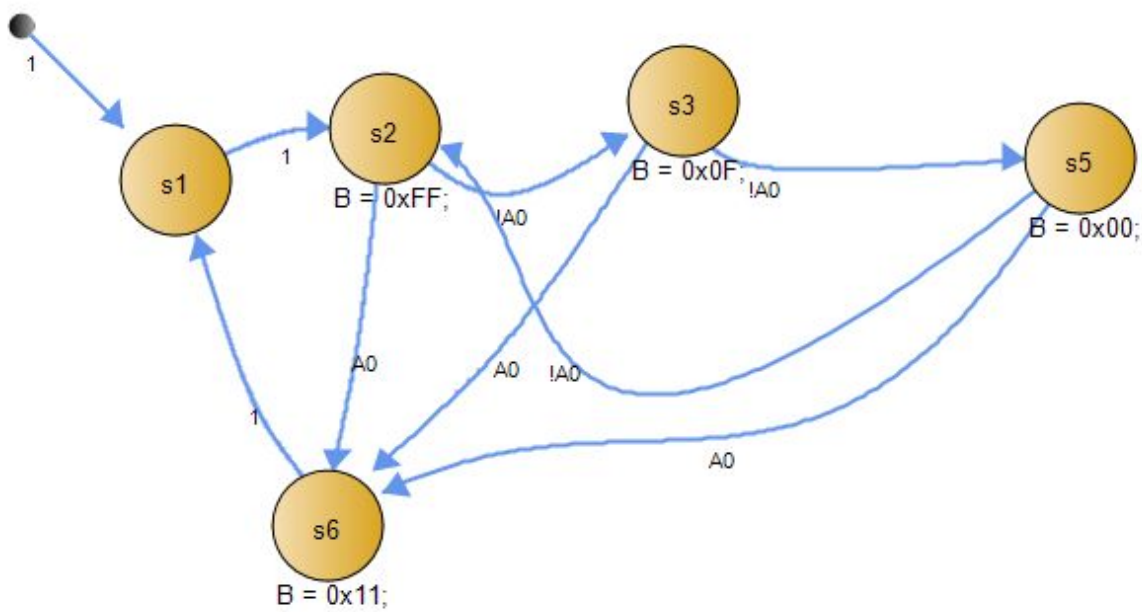


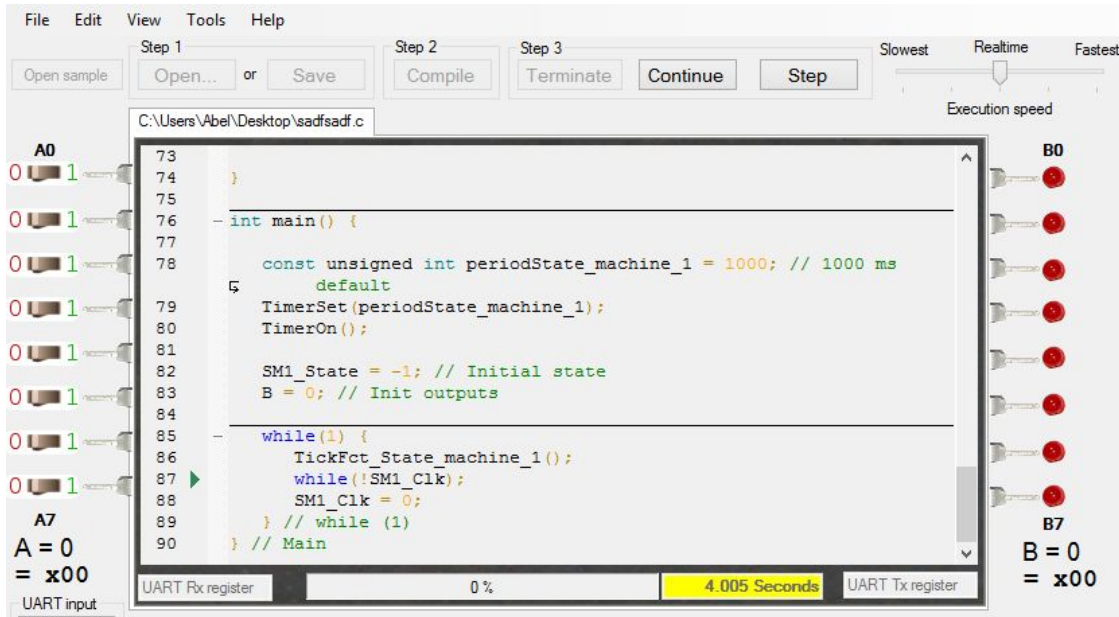
## State Machine Programming Tutorial

This tutorial will show you how to light up some lights in a specific pattern using state machines.

State machines are really important because they allow you to visualize your system. This can make programming them really easy. Let's look at the following state machine:



In this state machine, you can see how we have our starting state is s1. From that state, we go to s2. In s2 we set B = 0xFF. This means we are lighting up everything. We then go on to state s3 where we do B = 0x0F; We only light up half the lights this time. We then go on to s5 where we do not light up anything. From here, the state machine goes to s2. If at anytime A0 is turned on, we go to s6 and light up the first and 5th light.



As you can see from the picture above, this is the RIMS environment which simulates a simple microprocessor. We have our inputs A0 - A7 and our outputs B0 - B7. We are using a timer of 1 second so each state will have 1 second time for outputs.

Let's take a look at some of the code:

```

- TickFct_State_machine_1() {
-     switch(SM1_State) { // Transitions
-         case -1:
-             SM1_State = SM1_s1;
-             break;
-         case SM1_s1:
-             if (1) {
-                 SM1_State = SM1_s2;
-             }
-             break;
-         case SM1_s2:
-             if (!A0) {
-                 SM1_State = SM1_s3;
-             }
-             else if (A0) {
-                 SM1_State = SM1_s6;
-             }
-             break;

```

This code above is from our state machine logic that decides which state should we be in next. You can see in case SM1\_s2, we go to SM1\_S3 if A0 is not pressed. If it is pressed, we go to SM1\_s6 which is another state. The other states all follow this same approach. You simply tell the microprocessor which state to go to next depending on the given inputs.

Let's now take a look at the code for doing something once we are in a state:

```
switch(SM1_State) { // State actions
    case SM1_s1:
        break;
    case SM1_s2:
        B = 0xFF;
        break;
    case SM1_s3:
        B = 0x0F;
        break;
    case SM1_s5:
        B = 0x00;
        break;
    case SM1_s6:
        B = 0x11;
        break;
    default: // ADD default behaviour below
        break;
} // State actions
```

In this code above, you can see the actions that take place while we are in each state. Some states have no action while others are sending output B a specific pattern.

Let's now take a look at the output at important states:

S2:

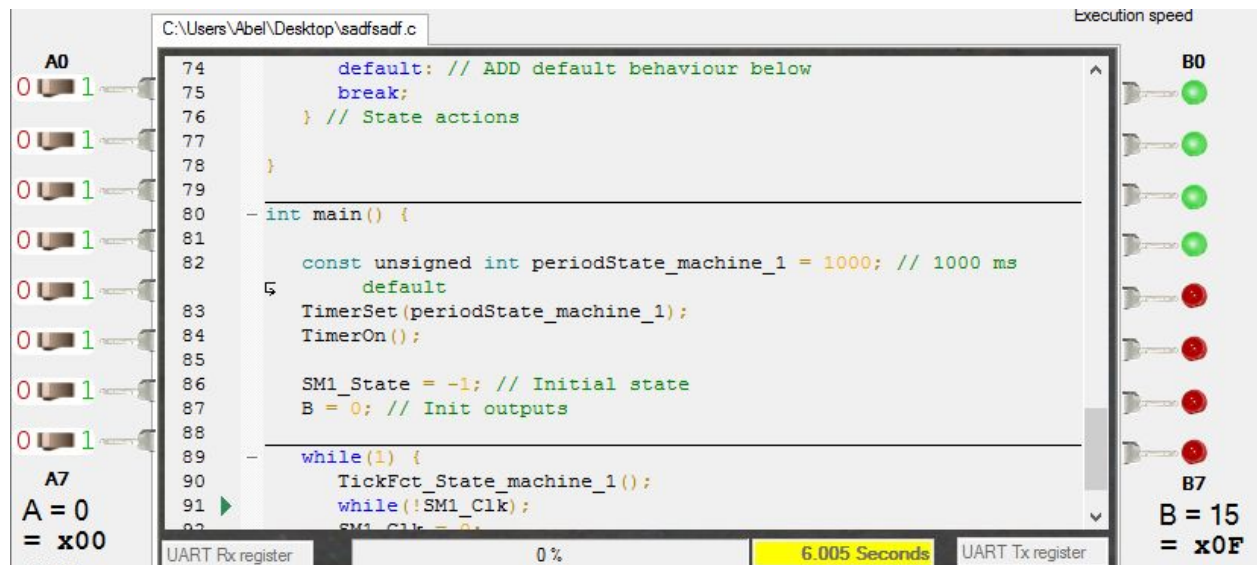
The screenshot displays a code editor with a state machine implementation. The code includes a switch statement for state actions and a main function that initializes the state machine and enters a loop. The hardware monitor on the right shows the output B (B7) as 255 (0xFF) at 5.005 seconds. The UART Rx register shows 0% and the UART Tx register shows 0%.

```
74         default: // ADD default behaviour below
75             break;
76     } // State actions
77
78 }
79
80 - int main() {
81
82     const unsigned int periodState_machine_1 = 1000; // 1000 ms
83     default
84     TimerSet(periodState_machine_1);
85     TimerOn();
86
87     SM1_State = -1; // Initial state
88     B = 0; // Init outputs
89
90     while(1) {
91         TickFct_State_machine_1();
92         while(!SM1_Clk);
93         SM1_Clk = 0;
```

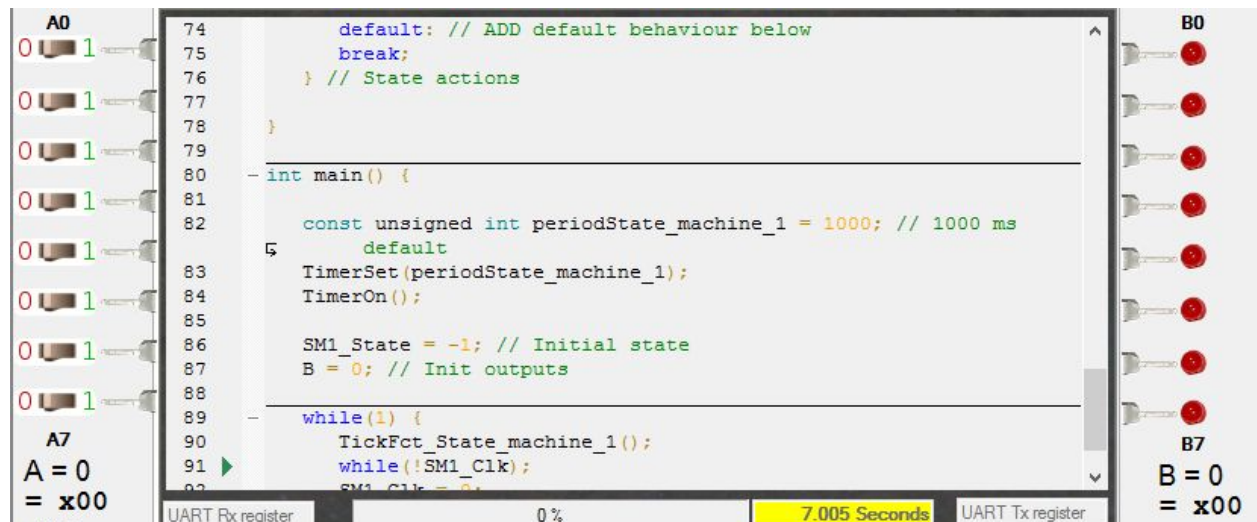
UART Rx register: 0%  
5.005 Seconds  
UART Tx register: 0%

B7  
B = 255  
= xFF

S3:



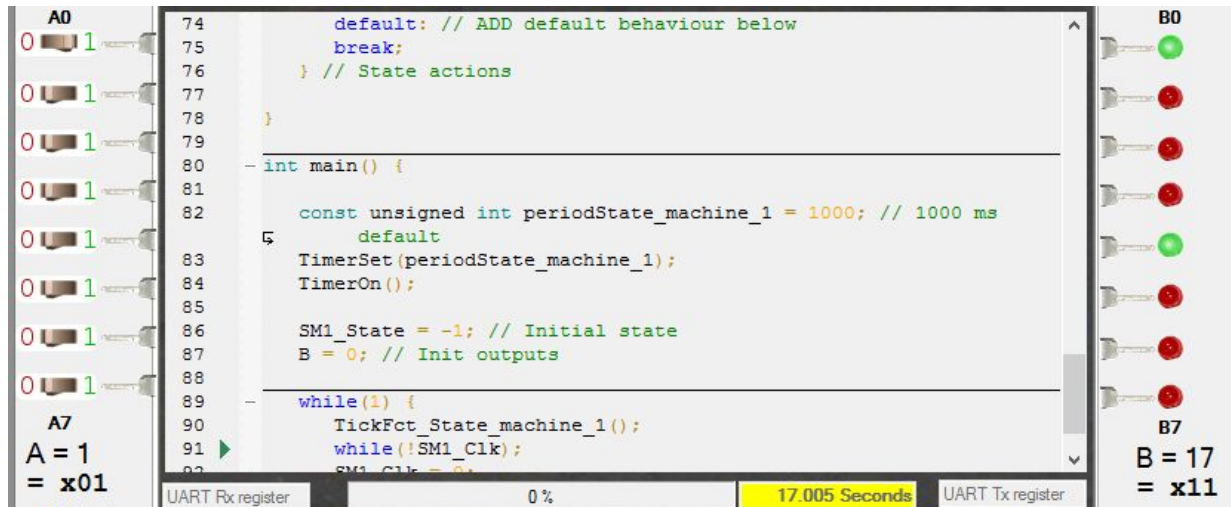
s5:



:

s6:

This state only happens when A0 is engaged. You can look in the top left corner of the picture and see that the switch was pushed.



As you guys can see from the pictures, the right outputs are on B.

This tutorial has given you a basic understanding of how to program state machines. It is important to make sure that your state machine should never be able to go to two different states at any given time.