Project 5
FSMs
Keerthi Radhakrishnan

The Goal

- Implement a finite state machine that successfully debounces a switch.
- That is, implement a software timer based finite state machine that “knows” if the button should transition between high and low, or vice versa.

The Solution

- We implemented a state table that would map out the behavior of a debouncing function targeted at the switch.
- We encapsulated these behaviors within an ANSI C struct. This allows for more portable, reusable code.
- We essentially copied the transitions for the next state and output functions as per the document.

The Implementation

- **Poll the real-time status of the switch:**

```
SwitchStatus GetSwitch(SwitchDefine *Switch)
{
    if(*Switch->SwitchPort & Switch->SwitchPortBit)
    {
    return Low;
    }
    else
    {
    return High;
    }
}
```

The Implementation (cont'd)

- Get the variables X1, X0 to figure out "where" in the state machine the debouncing mechanism is.

Getting Location

```
// First, determine the current inputs, X1 and X0.
CurrentSwitchReading = GetSwitch(Switch);
if(CurrentSwitchReading==Low)
{
    X0 = FALSE;
}
else
{
    X0 = TRUE;
}

//calculate elapsed time from last event.
ElapsedTime = (unsigned int)(g1msTimer - Switch->EventTime);

if((Switch->CurrentState == DbValidateHigh && ElapsedTime > Switch->HoldTime) || (Switch->CurrentState == DbValidateLow && ElapsedTime > Switch->ReleaseTime) )
{
    X1 = TRUE;
}
else
{
    X1 = FALSE;
}
```

The Implementation (cont'd)

- Determine the next state transition (and thus the next set of behaviors to follow).

Next State

```
switch (Switch->CurrentState) {
    case DbExpectHigh:
        if(X0==1) //X0=1
        {
            NextState = DbValidateHigh;
        }
    break;
    case DbValidateHigh:
        if(X0==0) //X0=0
        {
            NextState = DbExpectHigh;
        }
        else //X0=1
        {
            if(X1==1) //X1=1,X0=1
            {
                NextState = DbExpectLow;
            }
        }
    break;
    case DbExpectLow:
        if(X0==0) //X0=0
        {
            NextState = DbValidateLow;
        }
    break;
    case DbValidateLow:
        if(X0==1) //X0=1
        {
            NextState = DbExpectLow;
        }
        else //X0=0
        {
            if(X1==1) //X1=1,X0=1
            {
                NextState = DbExpectHigh;
            }
        }
    break;
    default: NextState = DbExpectHigh;
}
```

Performing the Output

- **Figure out if debouncing is necessary given the input state.**

```
// Perform the output function based on the inputs and current state.
switch (Switch->CurrentState) {
    case DbExpectHigh:
        DebouncedSwitchStatus = Low;
        if(X0 == TRUE){
            Switch->EventTime = g1msTimer;
        }
        SET_DEBUG1_PIN_LOW; SET_DEBUG0_PIN_LOW; //assign 0 0 to current state
    break;
    case DbValidateHigh:
        if(X0 == TRUE && X1 == TRUE)
        {
            DebouncedSwitchStatus = High;
        }
        else
        {
            DebouncedSwitchStatus = Low;
        }
        SET_DEBUG1_PIN_LOW; SET_DEBUG0_PIN_HIGH; //assign 0 1
    break;
    case DbExpectLow:
        DebouncedSwitchStatus = High;
        if(X0 == FALSE){
            Switch->EventTime = g1msTimer;
        }
        SET_DEBUG1_PIN_HIGH; SET_DEBUG0_PIN_LOW; // assign 1 0
    break;
    case DbValidateLow:
        if(X0 == FALSE && X1 == TRUE)
        {
            DebouncedSwitchStatus = Low;
        }
        else
        {
            DebouncedSwitchStatus = High;
        }
        SET_DEBUG1_PIN_HIGH; SET_DEBUG0_PIN_HIGH; // assign 1 1
    break;
}
```

Generating Debug Data

- Enable the Port 2 pins and return the binary representation of the “DbStatus” enum.
- This gives us insight into the decision making of the code.

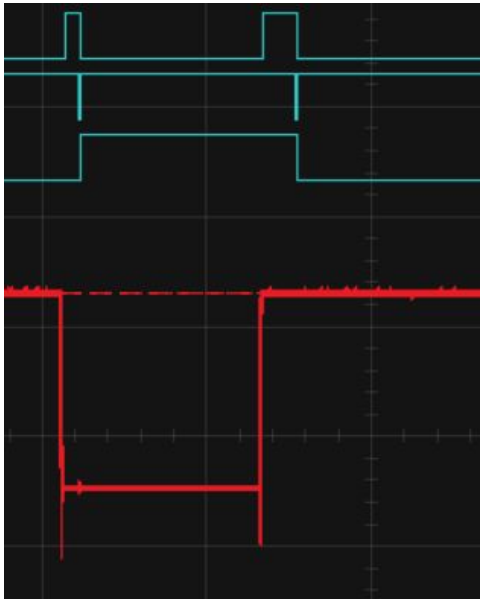
Turning on LEDs

- **Generate the LED response from the output that the decision making algorithm returns.**

```
while(1) {
    ManageSoftwareTimers();
    PushButtonStatus = Debouncer(&PushButton);
    if (PushButtonStatus == Low){ // Switch is inactive
        TURN_OFF_LED1;
    }
    else {
        TURN_ON_LED1;
    }
}
```

Verification of the Debounce

- Curiously, despite what I think is correct FSM implementation, there seems to be a spike right when the output hits its minimum, but not on the transition itself. The edges seem to be noise free. The issue seems to be that the second graph (X1) is not resetting to zero.



Fin