Conveyor Belt & Sunlight Monitoring
Embedded System Design, Lab 3

Ben Lorenzetti

October 1, 2015

Contents

1 Objectives and Problem Descriptions

2 Procedure

2.1 Conveyor Belt Controller

The circuit for the conveyor belt controller is shown in figure 1. There are two LEDs and two pushbuttons. A green LED indicates if the phototransistor is detecting a box in the line of sight. A red LED is used as a blinking alarm in case of jam. The two pushbuttons are for indicating that a jam has been manually cleared and for indicating the end of a shift.

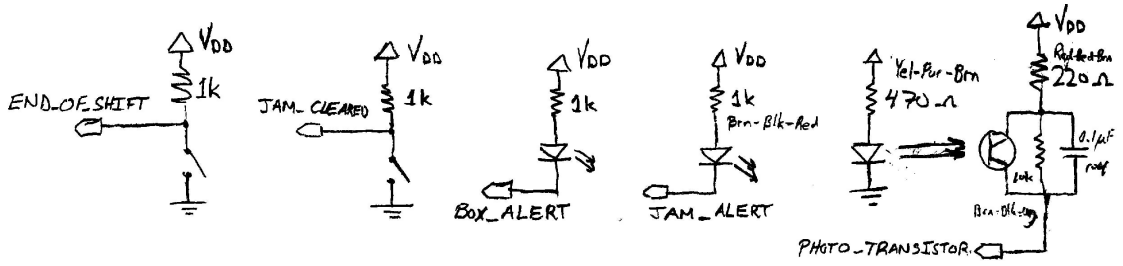

Figure 1: Conveyor Belt Controller Circuit

The conveyor belt specifications had a lot of requirements, including status upon reset and non-volatile memory. My approach to dealing with all of these requirements was to create a state diagram and a main loop model where the main loop has the ability to change states once per loop and takes actions based on the current state.

The program flow is shown in the implementation flowchart in figure 2 and the transistions for the state variable are shown in figure 3. I think these diagrams show my approach to the problem more concisely than could be explained in words.

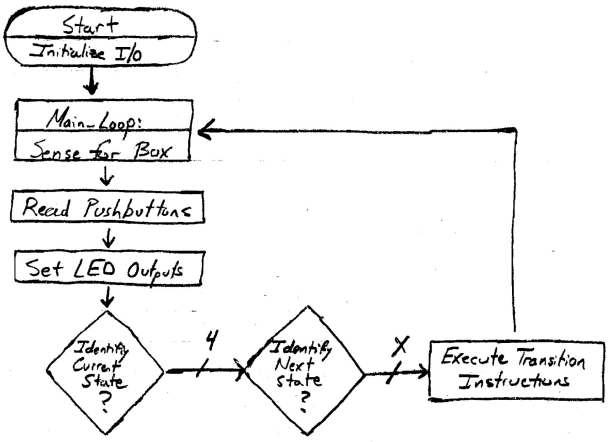

Figure 2: Conveyor Belt Implementation Flowchart

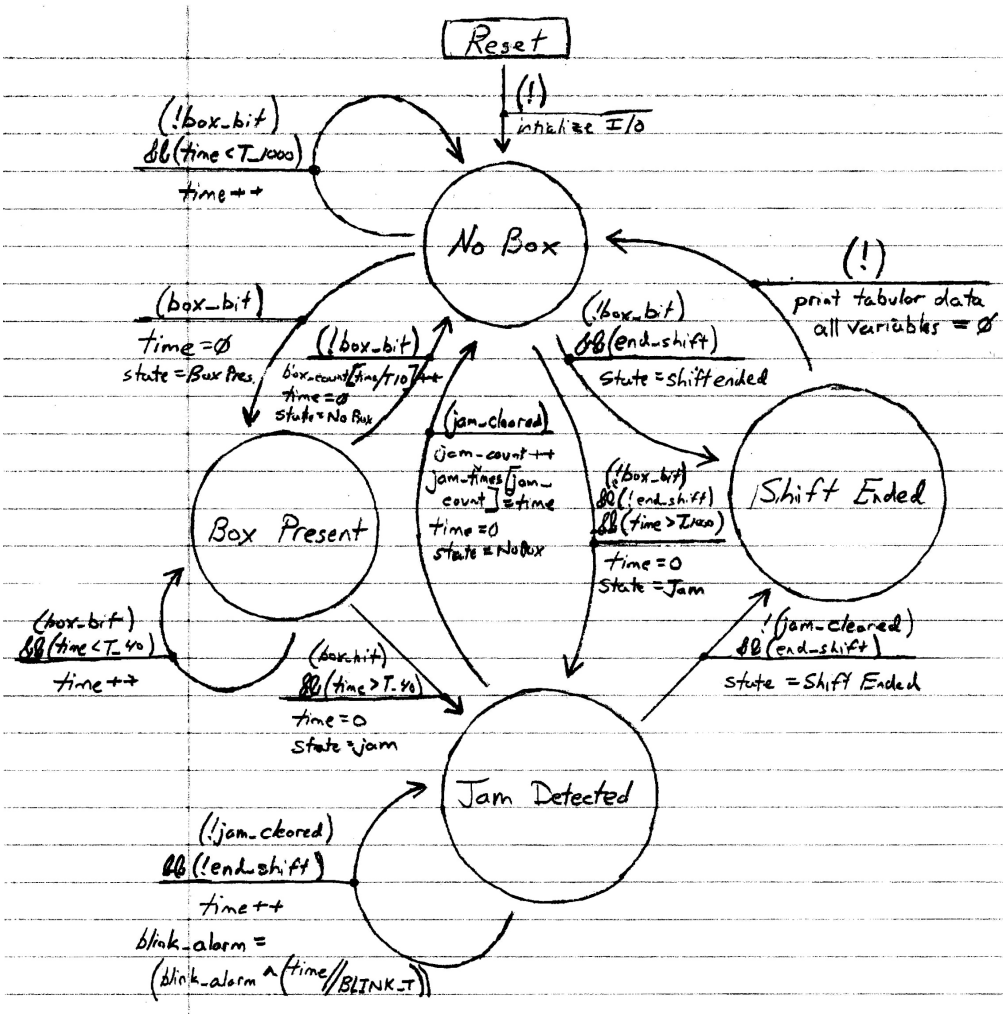

Figure 3: State Diagram for Conveyor Belt Controller

2.2 Sunlight on Planet Alpha Controller

The circuit for the sunlight sensor is shown in figure 4. An LED is used in reverse bias as a photosensor: greater light intensity causes an increase in the reverse bias leakage current. `RCTIME` can be used without a capacitor because the diode and an appropriate junction capacitance for the 2μs time resolution of the microcontroller. The 220 Ω resistor limits the charging current from damaging the microcontroller.

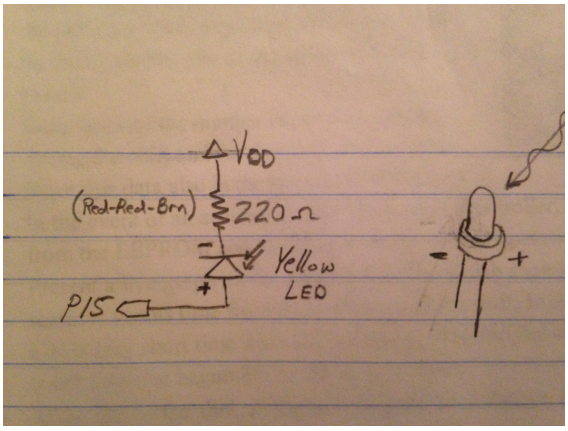

Figure 4: Sunlight Sensing Circuit

The programming for the sunlight sensing controller was also easy. Upon reset, the controller attempts to dump all its recorded data to a Debug channel and also prompts the user the current data should be discarded. Importantly, the default is no so that data is not lost if the Debug connection is not active.

Then, the microcontroller takes one measurement every 15 minutes and saves the data to EEPROM. In between it sleeps to conserve power. The program flowchart is shown in figure 5.

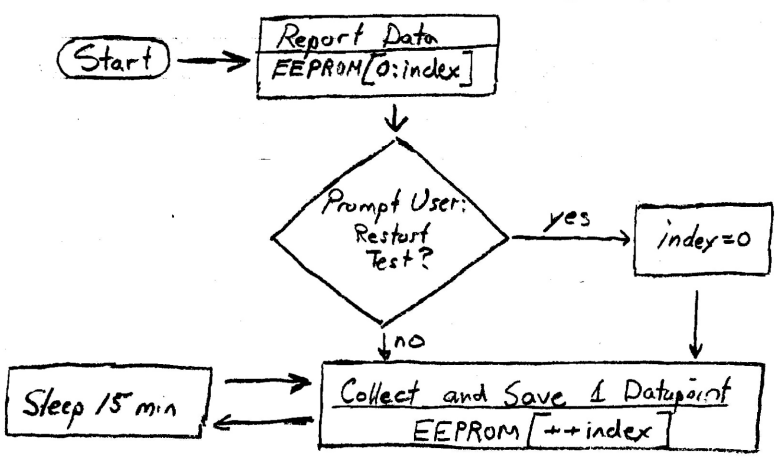

Figure 5: Sunlight on Planet Alpha Implementation Flowchart

3 Expected Results

For the conveyor controller, I expected results that matched the specifications–minus probably not having the timing for the main loop correct (`T_10_INCHES`).

For the sunlight data, a low number indicates strong light and an high number indicates dark conditions.

4 Experiment and Design Revisions

The conveyor circuit worked as expected, using R and C values from the prior lab.

For the sunlight sensor, I ran several test experiments to get the best divisor term to ensure the data was always withing 0-255. One interesting thing I noted was that if `RCTIME` timed out (i.e. if the room was completely dark), then the time returned was zero instead of the maxium $2^{16} - 1$.

5 Observations

The sunlight sensor was placed in the window of my room for $\tilde{2}4$ hours. Afterwards, I retrieved the data with the Debug terminal and imported it into excel as a space-separated list. I changed all zero values to 255 manually based on what I learned in experimentation/design revisions.

Finally, I plotted the inverse of the times measured to show sunlight intensity over the course of the day. This is shown in figure 6.

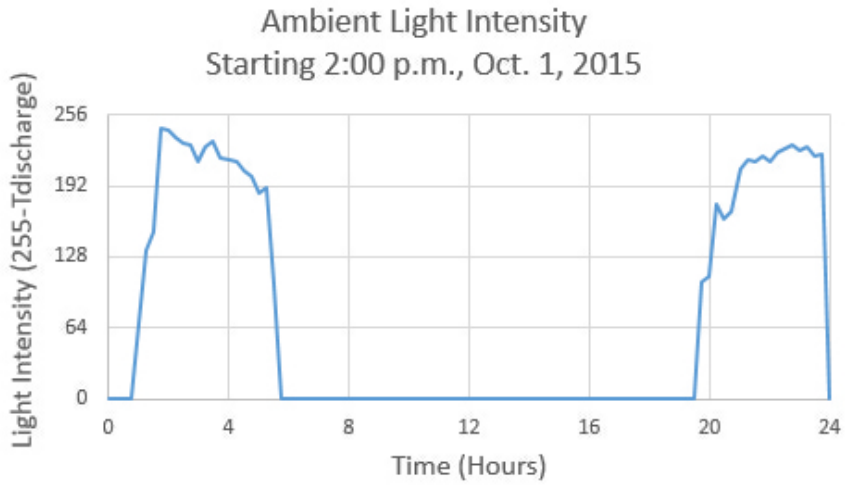

Figure 6: Ambient Lighting of My Bedroom Window

6 Implementation Code

6.1 Converyor Belt

```
' Conveyor-Belt.bs2
' {$STAMP BS2}
' {$PBASIC 2.5}
'----------------------Declare Constants---------------------------'
SHIFT_END  CON 0
CLEAR_JAM  CON 4
BOX_ALERT  CON 11
JAM_ALERT  CON 9
ACTIVE_LEVEL      CON 0
PHOTO_TRANSISTOR  CON 7
T_LIGHT_THRESHOLD CON 80
' (0.1uF || 10kOhm || photo-trans.) has max RC discharge time ~ 600 *(2us)
T_RC_CHARGE       CON 1
T_10_INCHES       CON 50
' Number of main loop iterations per 10 inch conveyor displacement
T_BLINK           CON 20
MAX_BOX_LENGTH    CON 4
MAX_GAP_LENGTH    CON 100
' MAX_ lengths are relative to the 10 inch unit length
MAX_EEPROM_SIZE   CON 4
NO_BOX       CON 0
BOX_PRESENT  CON 1
JAM_DETECTED CON 2
SHIFT_ENDED  CON 3

'------------------------Declare RAM Variables----------------------'
state VAR Nib
time  VAR Word
jam_cleared  VAR Bit
end_shift VAR Bit
box_sense  VAR Bit
ptran_time   VAR Word
blink_alarm VAR Bit

'------------------------------Start--------------------------------'
state = (1 << NO_BOX)
OUTS = $FFFF * (~ACTIVE_LEVEL)
DIRS = (1 << BOX_ALERT) | (1 << JAM_ALERT)

Main_Loop:
  '-------------------------Sense for Box----------------------------'
  ' Charge the capacitor for light measurement
  DIRS = DIRS | (1 << PHOTO_TRANSISTOR)
  OUTS = OUTS | (ACTIVE_LEVEL * (1 << PHOTO_TRANSISTOR))
  OUTS = OUTS & (~ ( (ACTIVE_LEVEL ^ 1) * (1 << PHOTO_TRANSISTOR)))
  PAUSE T_RC_CHARGE
  ' Discharge the capacitor and make binary light on/off decision
  RCTIME PHOTO_TRANSISTOR, ACTIVE_LEVEL , ptran_time
  box_sense = ptran_time / T_LIGHT_THRESHOLD
```

```
  '--------------------Read Pushbutton Inputs-----------------------'
  jam_cleared = ~(ACTIVE_LEVEL ^ ( (INS & (1 << CLEAR_JAM)) >> CLEAR_JAM))
  end_shift =   ~(ACTIVE_LEVEL ^ ( (INS & (1 << SHIFT_END)) >> SHIFT_END))

  '----------------------Set LED Outputs----------------------------'
  temp VAR Word
  temp = OUTS ^ (~($FFFF * ACTIVE_LEVEL))
  temp = temp | ((((state >> JAM_DETECTED)&1) << JAM_ALERT) * blink_alarm)
  temp = temp & (~(((((~state) >> JAM_DETECTED)&1) << JAM_ALERT))* blink_alarm)
  temp = temp |    (box_sense << BOX_ALERT)
  temp = temp & (~((~box_sense) << BOX_ALERT))
  OUTS = temp ^ (~($FFFF * ACTIVE_LEVEL))

  '--------------------Identifty the Current State------------------'
  No_Box_State:
    '---------------Identifty the Next State Transition-------------'
    IF (state <> (1 << NO_BOX)) THEN Box_Present_State
    IF (box_sense = 1) THEN No_Box_to_Box_Present
    IF (end_shift = 1) THEN No_Box_to_Shift_Ended
    IF (time > (T_10_INCHES * MAX_GAP_LENGTH)) THEN No_Box_to_Jam_Detected
    GOTO Continue
  Box_Present_State:
    '---------------Identifty the Next State Transition-------------'
    IF (state <> (1 << BOX_PRESENT)) THEN Jam_Detected_State
    IF (box_sense = 0) THEN Box_Present_to_No_Box
    IF (time > (T_10_INCHES * MAX_BOX_LENGTH)) THEN Box_Present_to_Jam_Detected
    GOTO Continue
  Jam_Detected_State:
    '---------------Identifty the Next State Transition-------------'
    IF (state <> (1 << JAM_DETECTED)) THEN Shift_Ended_State
    IF (jam_cleared = 1) THEN Jam_Detected_to_No_Box
    IF (end_shift = 1) THEN Jam_Detected_to_Shift_Ended
    IF (0 = (time // T_BLINK)) THEN Jam_Detected_to_Jam_Detected
    GOTO Continue
  Shift_Ended_State:
    '---------------Next State Transition is Automatic--------------'
    DEBUG "End of Shift Status Report:", CR
    READ 0, temp
    DEBUG "10 inch boxes: ", DEC temp, CR
    READ 4, temp
    DEBUG "Number of Jams: ", DEC temp, CR
    GOTO Shift_Ended_to_No_Box
'-----------------Iterate Time for Next Loop Iteration--------------'
Continue:
  time = time + 1
GOTO Main_Loop


'--------------------State Transition Subroutines-------------------'
No_Box_to_Box_Present:
  time = 0
  state = 1 << BOX_PRESENT
GOTO Main_Loop
```

```
No_Box_to_Shift_Ended:
Jam_Detected_to_Shift_Ended:
  state = 1 << SHIFT_ENDED
GOTO Main_Loop

No_Box_to_Jam_Detected:
Box_Present_to_Jam_Detected:
  DEBUG "Jam Detected, time=", DEC time, CR
  time = 0
  state = 1 << JAM_DETECTED
GOTO Main_Loop

Box_Present_to_No_Box:
  READ (time / T_10_INCHES), temp
  WRITE (time / T_10_INCHES), (temp + 1)
  time = 0
  state = 1 << NO_BOX
GOTO Main_Loop

Jam_Detected_to_No_Box:
  DEBUG "Jam Cleared by User", CR
  READ 4, temp
  WRITE 4, (temp + 1)
  IF (temp > MAX_EEPROM_SIZE) THEN Protect_Program_Memory
    WRITE (5 + (2*(temp + 1))), time
  Protect_Program_Memory:
  time = 0
  state = 1 << NO_BOX
GOTO Main_Loop

Shift_Ended_to_No_Box:
  FOR temp = 0 TO MAX_EEPROM_SIZE
    WRITE temp, 0
  NEXT
  time = 0
  state = 1 << NO_BOX
GOTO Main_Loop

Jam_Detected_to_Jam_Detected:
  blink_alarm = (blink_alarm ^ 1)
GOTO Continue
```

6.2 Sunshine Monitoring

```
' Sunshine-on-Planet-Alpha.bs2
' {$STAMP BS2}
' {$PBASIC 2.5}

SLEEP_TIME CON  900
CHARGE_TIME CON 10
MAX_DATAPOINTS CON 96
LED_PIN CON 15

DIRS = (1 << LED_PIN)

rc_time VAR Word
ram_index VAR Byte
eeprom_index VAR Word
restart_test VAR Byte

' Print Data from EEPROM
READ MAX_DATAPOINTS, eeprom_index
FOR ram_index = (eeprom_index + 1) TO (MAX_DATAPOINTS - 1)
  READ ram_index, rc_time
  DEBUG DEC3 (ram_index - (eeprom_index + 1)), 9, DEC3 rc_time, CR
NEXT
FOR ram_index = 0 TO eeprom_index
  READ ram_index, rc_time
  DEBUG DEC3 (ram_index + (MAX_DATAPOINTS - eeprom_index)), 9, DEC3 rc_time, CR
NEXT

' Ask User: Restart Test?
DEBUG "Restart Test? (y/n): "
DEBUGIN STR restart_test \1
DEBUG CR
IF (restart_test <> 121) THEN Break
  WRITE MAX_DATAPOINTS, 0
Break:

DO
  LOW LED_PIN
  PAUSE CHARGE_TIME
  RCTIME LED_PIN, 0, rc_time
  rc_time = (rc_time >> 8)
  READ MAX_DATAPOINTS, eeprom_index
  DEBUG "Writing ", DEC rc_time, " to EEPROM ", DEC (eeprom_index // MAX_DATAPOINTS), CR
  WRITE (eeprom_index // MAX_DATAPOINTS), rc_time
  WRITE MAX_DATAPOINTS, ((eeprom_index + 1) // MAX_DATAPOINTS)

  SLEEP SLEEP_TIME
LOOP
```