

Traffic Light Controller & Jeopardy Game

Embedded System Design, Lab 1

Ben Lorenzetti

September 17, 2015

Contents

1	Objectives and Problem Description	1
1.1	Traffic Light Controller	1
1.2	Jeopardy Style Game	2
2	Procedure	2
2.1	Traffic Light Controller	2
2.2	Jeopardy Style Game	3
3	Expected Results	3
4	Experiment and Design Revisions	4
4.1	Traffic Light Controller	4
4.2	Jeopardy Game	4
5	Observations	4
6	Discussion	5
7	Exercises	5
8	Programs	8
8.1	Traffic Light Controller	8
8.2	Jeopardy Style Game	9

1 Objectives and Problem Description

The objectives of this lab were to learn the basics of the Stamp 2 board, setup the programming+debugging toolchain, and complete two projects with the simplest actuators available: LEDs and pushbutton switches.

1.1 Traffic Light Controller

The first project is the traffic light controller, which models two traffic lights at an intersection between a major and minor road. There are 6 LEDs—2 red, 2 yellow, 2 green—and a pushbutton switch representing a magnetic presence sensor on the minor road. The major road should always

remain green unless the minor road pushbutton is triggered. This trigger causes a transistion following normal traffic signals, with 5 second delays between each signal permutation and 10 seconds green duration on the minor road.

1.2 Jeopardy Style Game

The second project has two pushbutton representing Jeopardy buzzers and some indicator LEDs. When the go LED turns on, two players compete to buzz in the quickest. The winner is displayed via LEDs and the winning player's reaction time is printed to the debug terminal. To make the game more difficult and competitive, the go LED is delayed by a random amount of time.

2 Procedure

2.1 Traffic Light Controller

The lab said to go through the first 3 chapters of the book and do the activities, so I did this.

I built the circuit shown in figure 1 on the Basic Stamp breadboard. It has the 6 LEDs in active low configuration with $1k\Omega$ current limiting resistors. The pushbutton switch was also configured active low for consistency. It also has a $1k\Omega$ current limiting resistor.

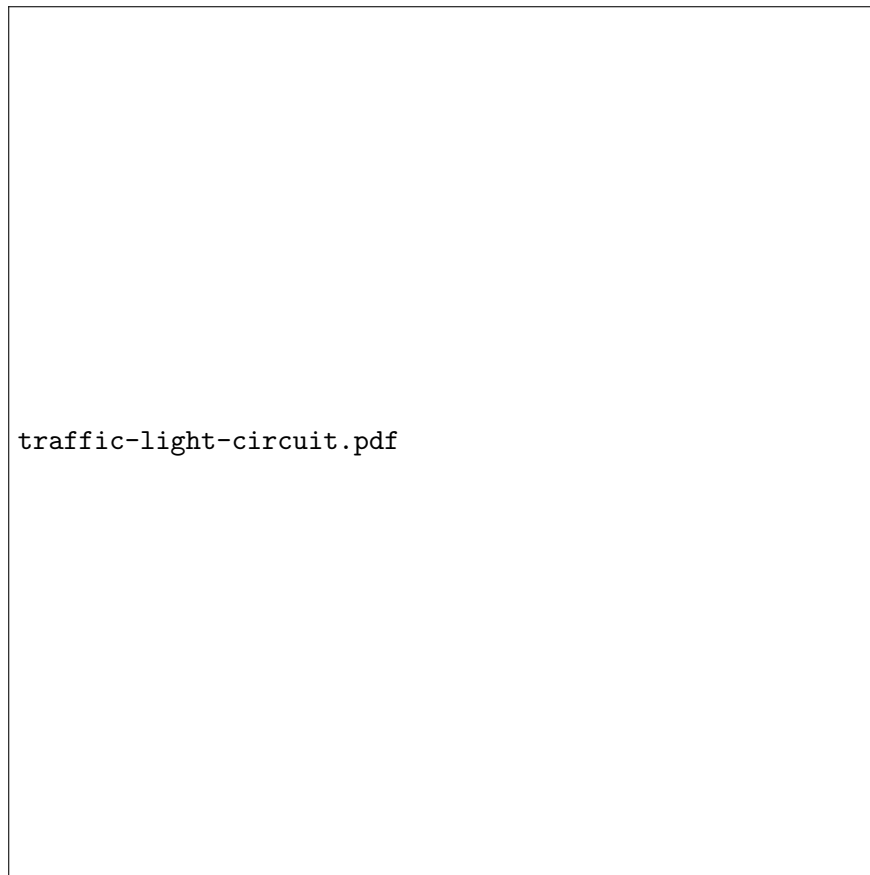



Figure 1: Traffic Light Controller Circuit Diagram

The control flow chart for the PBasic program is shown in figure 2.



traffic-light-flowchart.pdf

Figure 2: Traffic Light Controller Flowchart

2.2 Jeopardy Style Game

I designed the circuit for the Jeopardy game similarly to the traffic light controller. Each LED or pushbutton switch has a $1k\Omega$ current limiting resistor and is configured active low. The circuit diagram for the Jeopardy game is shown in figure 3.

The jeopardy game was more complicated than the traffic light controller, but only from a software perspective—not hardware.

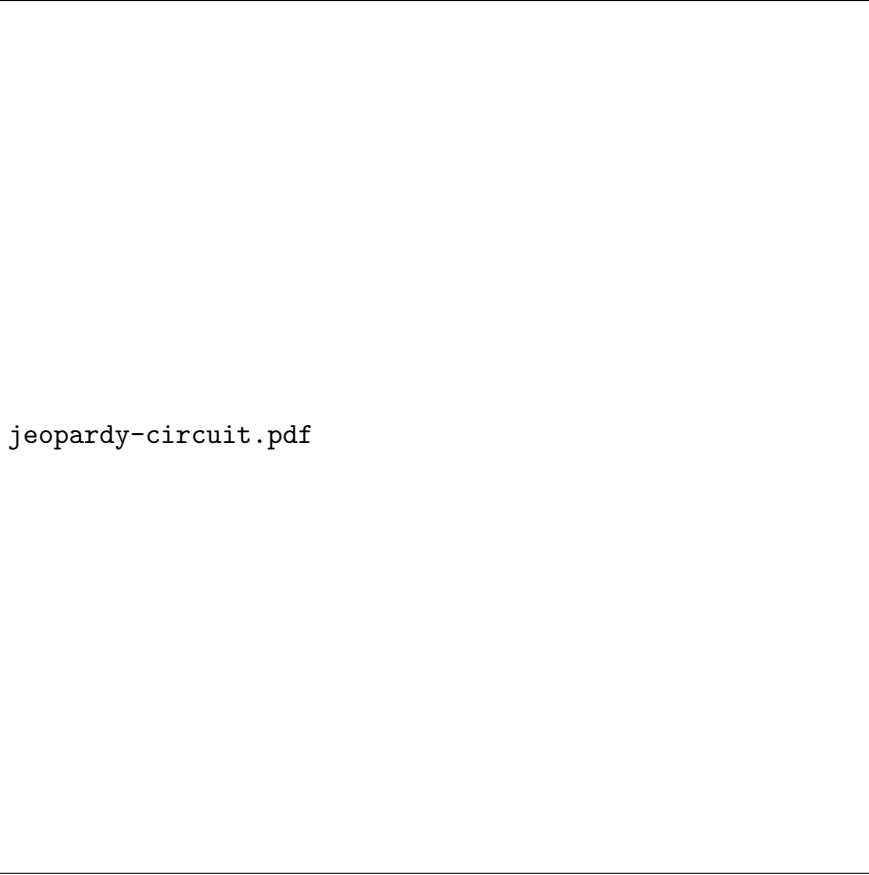
I used the random function to generate a random delay time before the game begins, passing in the previous game’s reaction time as a seed value so that the same pseudorandom sequence is not repeated every game.

Once the game begins the microcontroller executes a tight loop that does checks the pushbuttons for a victor and then increments a `react_cycles` variable. When the first player buzzes in, the appropriate LED lights up and their reaction time is calculated from the number of cycles and the closest integer to the loop iteration time. This value is printed to the debug terminal.

The control flow chart for the PBasic program is shown in figure 4.

3 Expected Results

I expected both circuits to behave according to the problem statements.



jeopardy-circuit.pdf

Figure 3: Jeopardy Circuit Diagram

4 Experiment and Design Revisions

4.1 Traffic Light Controller

This circuit and code worked like a charm. No revisions had to be made, but I did restyle the code to look more like C with boolean algebra.

4.2 Jeopardy Game


I really did not want to have two sets of Debug statements, depending on the winner. However, using only one Debug statement and storing the winner in a bit variable proved more difficult than I anticipated.

It was difficult to generate the boolean algebra statements for turning on the appropriate LEDs, mostly because PBasic sucks and doesn't even have a complete set of boolean operators.

So reducing to one set of Debut statements and only a single goto location proved more difficult, but I think it was worth the change for code brevity.

5 Observations

Both circuits worked as expected, and photos of the working boards are shown in figure 5 and figure 6. A screenshot of the Debug terminal for the Jeopardy board is shown in figure ??.



jeopardy-flowchart.pdf

Figure 4: Jeopardy-Like Game Flowchart

6 Discussion

This lab really revealed a major flaw of the Stamp Basic board: the fact that running an interpreter prevents you from knowing exact timing. If we were writing code directly for PIC, we could know the clock rate and write some assembly inside the reaction time loop—and thereby know the exact number of clock cycles and the clock rate.

I have a low opinion of the Stamp Basic board right now, at least compared to other micro-controllers. This week I complain about not having control of clock cycles or there being a clock variable that is accessible. Next week's dissatisfaction preview: no ADC on board.

7 Exercises

There were no exercises given.

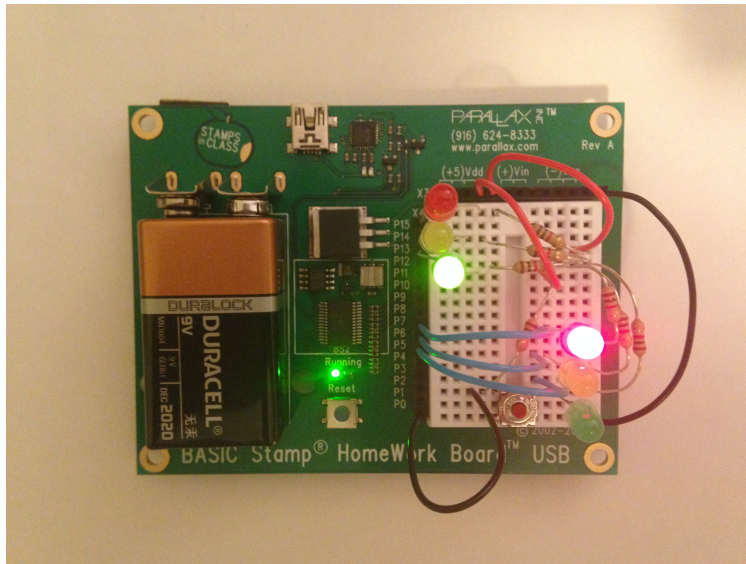


Figure 5: Traffic Light Controller Circuit

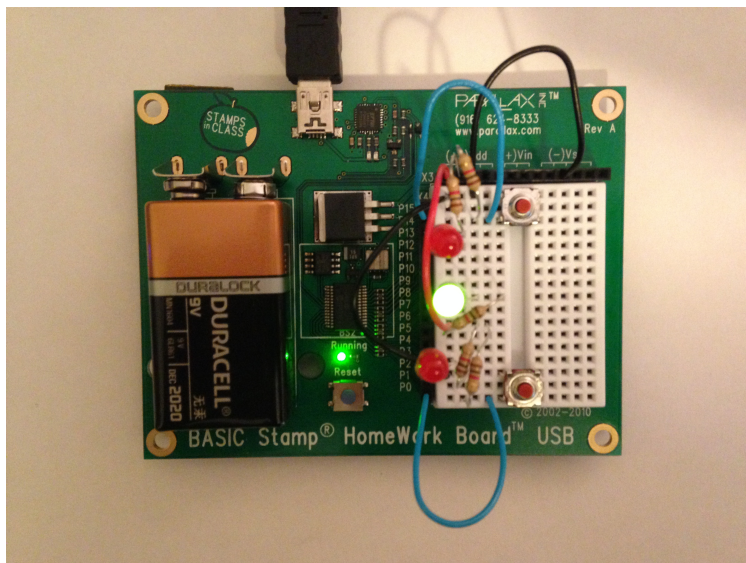


Figure 6: Jeopardy Game Circuit

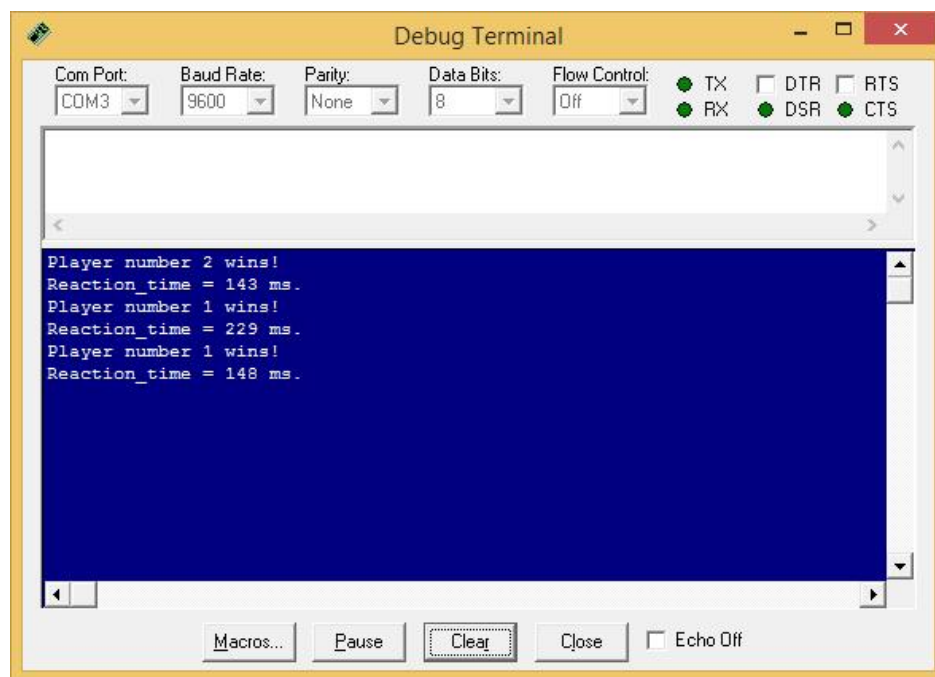


Figure 7: Screenshot of Jeopardy Debug Terminal with Reaction Times

8 Programs

8.1 Traffic Light Controller

```
' {$STAMP BS2}
' {$PBASIC 2.5}
' Traffic_Controller.bs2

RED CON $3
YELLOW CON $2
GREEN CON $1
HIGHWAY CON $C
FARM_ROAD CON $4

' Define I/O Directions: 1110|0000|1110|0000
DIRS = $EOEO
' Set Initial Output Conditions:
OUTS = $EOEO
TOGGLE (HIGHWAY | GREEN)
TOGGLE (FARM_ROAD | RED)

DO
  IF (INO = 1) THEN Continue
  State_Transitions:
  PAUSE 5000
  HIGH (HIGHWAY | GREEN)
  LOW (HIGHWAY | YELLOW)
  PAUSE 5000
  HIGH (HIGHWAY | YELLOW)
  LOW (HIGHWAY | RED)
  PAUSE 5000
  HIGH (FARM_ROAD | RED)
  LOW (FARM_ROAD | GREEN)
  PAUSE 10000
  HIGH (FARM_ROAD | GREEN)
  LOW (FARM_ROAD | YELLOW)
  PAUSE 5000
  HIGH (FARM_ROAD | YELLOW)
  LOW (FARM_ROAD | RED)
  PAUSE 5000
  HIGH (HIGHWAY | RED)
  LOW (HIGHWAY | GREEN)
  Continue:
LOOP
```


8.2 Jeopardy Style Game

```
' {$STAMP BS2}
' {$PBASIC 2.5}
' Jeopardy.bs2
' Declare Mapping for I/O Signals

BUZZER_1 CON $0001
BUZZER_2 CON $4000
P1_LED CON $0008
P2_LED CON $2000
GO_LED CON $0100

' Set I/O Directions (Only Set Outputs, default 0=input)
DIRS = (P1_LED | P2_LED | GO_LED)
' Set Output Initial Values (HIGH is off)
OUTS = (P1_LED | P2_LED | GO_LED)

' Declare Constants
CYCLE_PERIOD CON 1
' Note: CYCLE_PERIOD should be << 268 ms (average human reaction time)

' Declare Variables
react_cycles VAR Word
react_time VAR Word
winner VAR Bit

' Run Main Game Loop Forever
DO
  ' Blink All LEDs to Indicate Game Started
  OUTS = OUTS & (~(P1_LED | P2_LED | GO_LED))
  PAUSE 500
  OUTS = OUTS | (P1_LED | P2_LED | GO_LED)
  ' Random Wait Time--Use Prior Results as Seed
  RANDOM react_time
  PAUSE (react_time/10)
  ' Go - Count Cycles Until a Buzzer is Pressed
  react_cycles = 0
  OUTS = OUTS & ~(GO_LED)
  DO
    winner = 0
    IF (0 = (INS & BUZZER_1)) THEN Break
    winner = 1
    IF (0 = (INS & BUZZER_2)) THEN Break
    react_cycles = react_cycles + 1
  LOOP
  Break:
  ' Indicate the Winner by LED and turn off GO_LED
  OUTS = OUTS & (~ (P1_LED * (1 & (~winner))))
  OUTS = OUTS & (~ (P2_LED * (1 & winner)))
  OUTS = OUTS | (GO_LED)
  ' Calculate Reaction Time and Display Results
  react_time = react_cycles / CYCLE_PERIOD
  DEBUG "Player number ", DEC (winner + 1), " wins!", CR
  DEBUG "Reaction_time = ", DEC react_time, " ms.", CR
  ' Pause, then continue to the next game
  PAUSE 2000
LOOP
```