PIC 16F84A Microcontroller:

Traffic Light System (embedded C)

Aren Tyr
(Updated version Spring 2019)

Contents

List of Figures

1 Introduction

This demonstration project is concerned with the development of software for an embedded development board (the PIC 16F84A) to simulate a pedestrian crossing system.

2 Statement Of Requirements

2.1 Overall Requirements

A pedestrian crossing system, as with any other system, has certain specific demands.

The object of a pedestrian crossing system is that it should enable people to safely cross a given road. Therefore it has the following overall requirements:

1. Reliability and consistency. The crossing system *must* always function correctly – people's lives depend on it – and it must do so consistently, every time. Its behaviour should always be predictable and consistent.

2. Accurate. The timing of the traffic lights, crossing lights, and other functions must be accurate if the system is going to be consistent.

3. Intuitive and logical. The crossing system should function as expected, and be obvious to use. The only user input is the push button, and this must function correctly and accordingly depending on the state of the system – i.e. only light the wait light when appropriate, initiate the crossing sequence after an appropriate time delay, etc.

We can view (from an algorithmic perspective) the pedestrian crossing system as being composed of three core parts:

1. Traffic lights

2. Pedestrian lights (Red man, Green man, Wait light)

3. Crossing buzzer

Each of the three core parts will now be analysed in more detail below.

2.2 Traffic Light Requirements

1. The light sequence must be correct:

   (a) **Traffic active to traffic halted**
      i. Dim green light
      ii. Delay
      iii. Illuminate amber light
      iv. Delay
      v. Dim amber light
      vi. Delay
      vii. Illuminate red light

   (b) **Traffic halted to traffic active**
      i. Dim red light
      ii. Delay
      iii. Flash amber light

    iv. Delay
    v. Illuminate green light

2. The timing of the lights must be correct

2.3 Pedestrian Light Requirements

1. The light sequence must be correct when the crossing sequence has been initiated:
   (a) **Crossing inactive to crossing active**
      i. Dim red man light
      ii. Delay
      iii. Illuminate green man light
   (b) **Crossing active to crossing inactive**
      i. Flash green man light
      ii. Dim green man light
      iii. Delay
      iv. Illuminate red man light
2. The timing of the lights must be correct
3. The "Wait light" must be correct lit when needed. The conditions that determine whether it should be lit are:
   (a) Has the button been depressed?
   (b) Is a crossing sequence already in progress?

   When a crossing is still in progress, then the wait light in a crossing system does not illuminate when depressed. The system completely ignores the request. After the crossing sequence has completed, further inputs will be accepted and the wait light will illuminate. There is a set delay after a crossing has occurred before another one may occur.

2.4 Buzzer Requirements

1. The buzzer timing/sequence must be correct:
   (a) **Crossing inactive to crossing active**
      i. Green man light active
      ii. Delay
      iii. Activate buzzer
   (b) **Crossing active to crossing inactive**
      i. Delay
      ii. Suppress buzzer
2. The buzzer pitch and tone must be correct (as realistically possible)

3 Test Plan

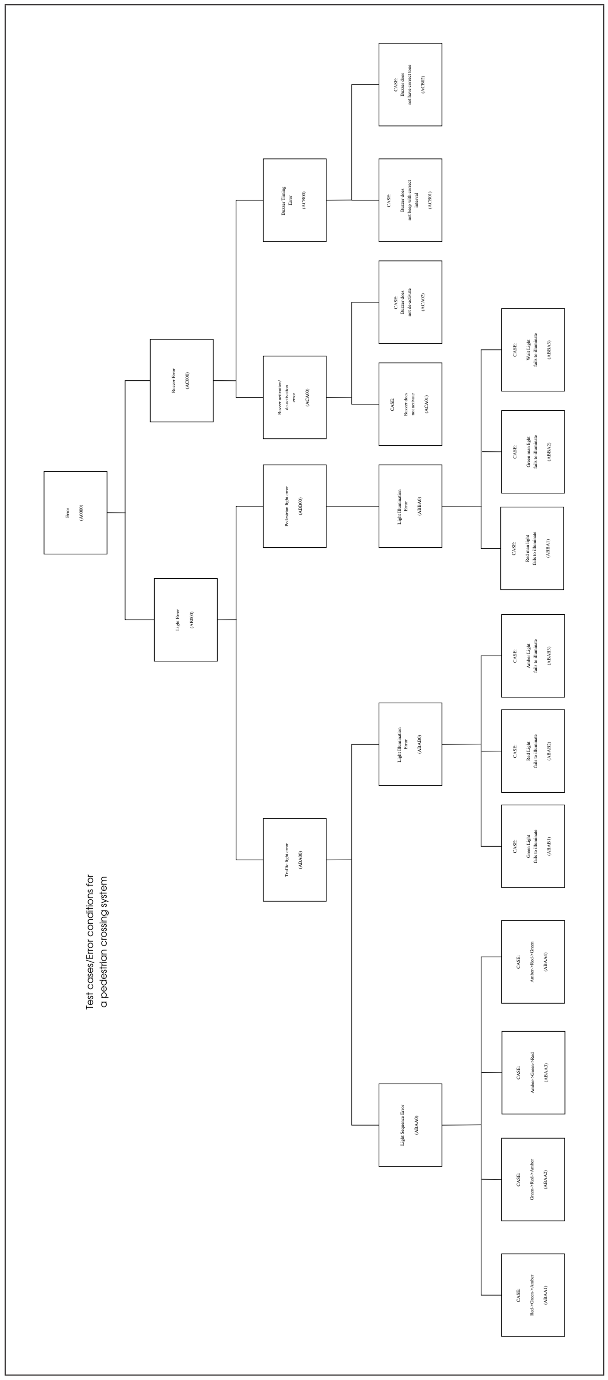
3.1 Test hierarchy

The test cases can be ordered according to the type of error they could potentially generate. Figure 1 shows a hierarchical structure for the test cases with respect to the type of error condition they can generate. Due to the size of the tree, Figures 2, 3, and 4 show individual sections of the tree (subtrees) enlarged.

3.2 Test strategy

Figures 2, 3, 4 all illustrate the various test cases. The testing strategy is one of simply exhaustively testing each possible case in turn, to see if the system functions in the correct manner.

Figure 1: Test Case/Error condition hierarchy (Overview)



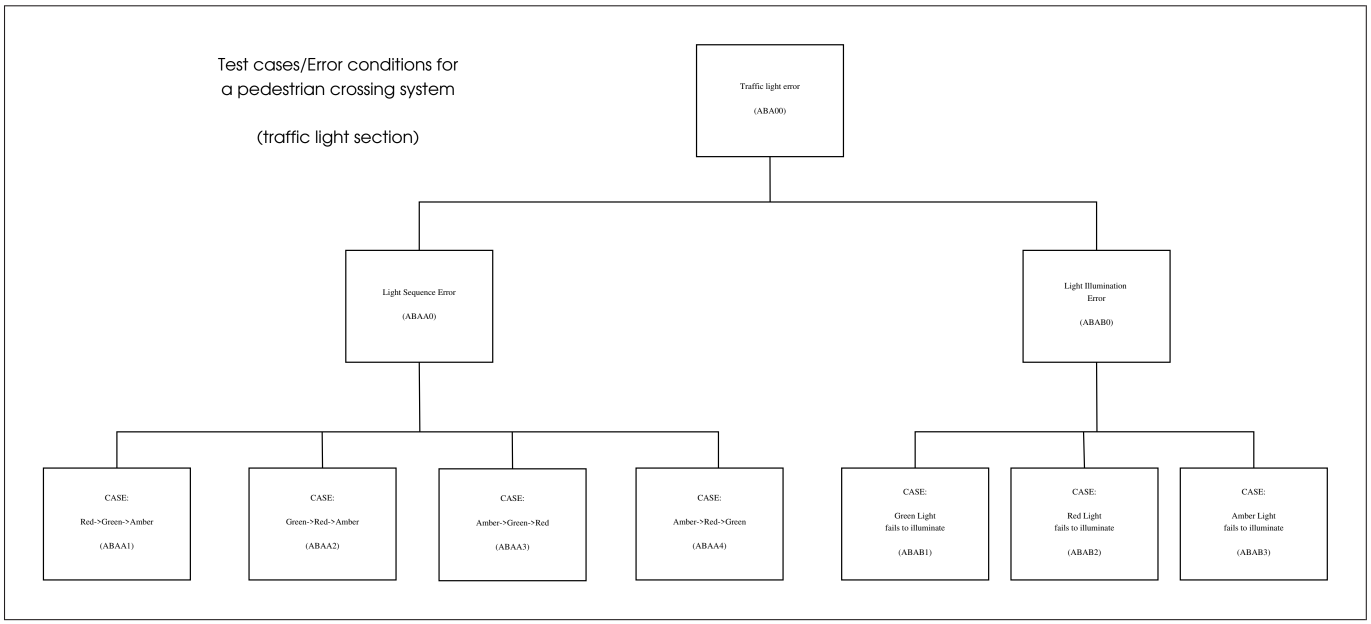
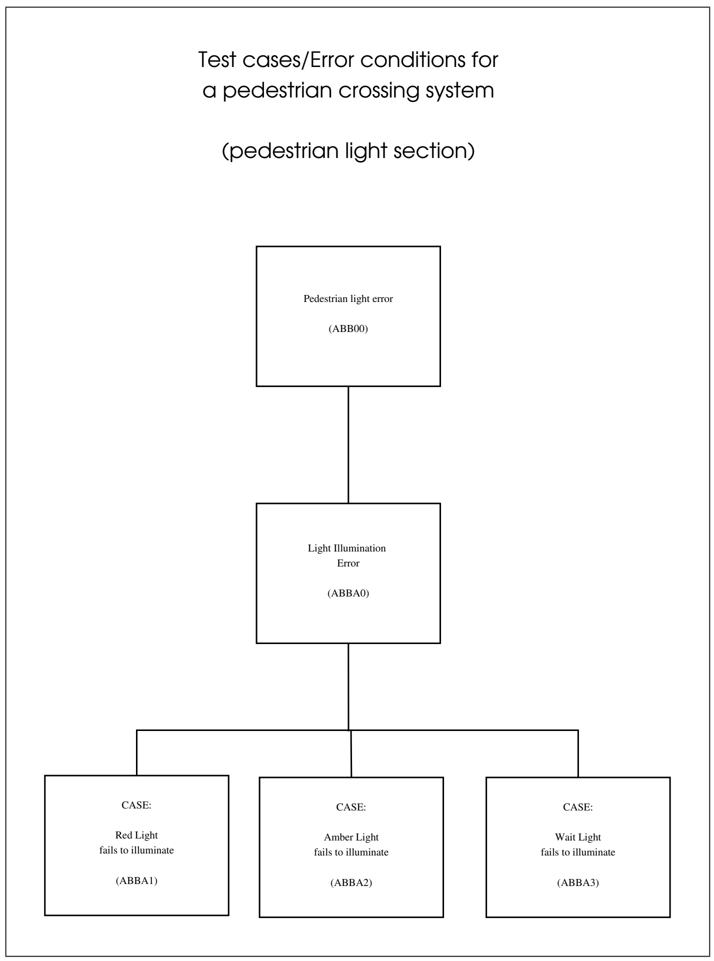


Figure 2: Test Case/Error condition hierarchy (Traffic Light Section)

Figure 3: Test Case/Error condition hierarchy (Pedestrian Light Section)



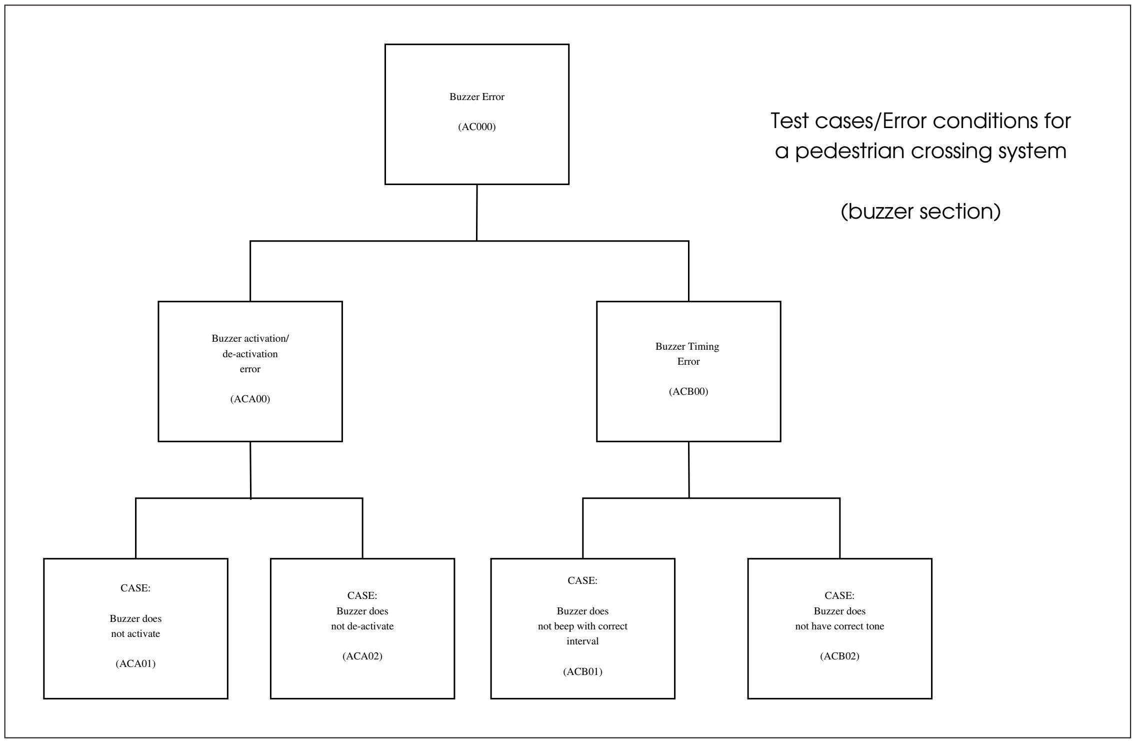


Figure 4: Test Case/Error condition hierarchy (Buzzer Section)

4 Design

4.1 Modelling decisions

Among other functions, the Millennium development board has six LEDs, a buzzer and a selection of push buttons. For this traffic light system, the options naturally presented themselves; out of the six LEDs (of which there are two of each colour), three could be used to represent the traffic light, and out of the remaining three, two could represent the red and green man lights for the pedestrian, leaving the remaining LED for the wait light. It was logical to use the corresponding LED colour for the particular light, since the three colours happen to be red, green, and amber/yellow as per a conventional traffic light system.

4.2 Timing Data

The table below illustrates some actual timing data recorded from a typical UK pedestrian crossing. The timing is immediately started after pushing the button. All timing is approximate, which is sufficient for our purposes here.

Event	**Time Elapsed (seconds)**
Wait light lit; Green light dims; transition	0.75
Amber light illuminated	2.00
Transition delay	0.75
Red light illuminated; transition	0.75
Wait light dims; red man light dims; transition	0.75
Green man light lit; delay	2.00
Buzzer activated; delay	9.50
Red light dims; green man & amber light flashed	8.00
Transition delay	0.75
Red man lit; transition	0.75
Green light lit; delay	10.00

4.3 Data flow

Figure 5 illustrates data flow through the pedestrian crossing system.

4.4 State Transitions

Figure 6 illustrates the various states that the pedestrian crossing system goes through.

4.5 Program Code Design

Figures 7, 8, 9, 10 and 11 illustrate the design of the various functions through the program.

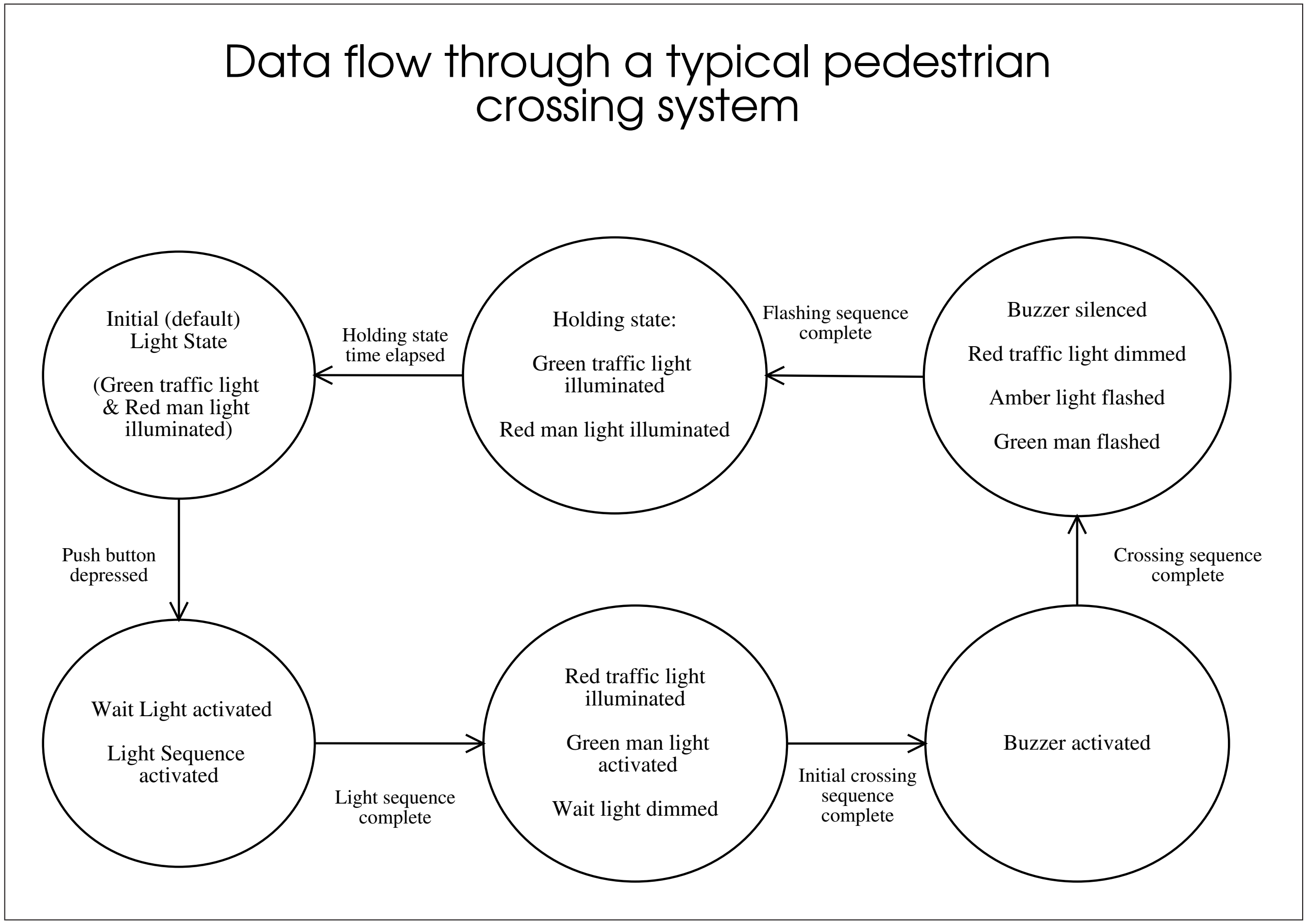


Figure 5: Data Flow diagram for a pedestrian crossing system

Figure 6: State transition diagram for a pedestrian crossing system

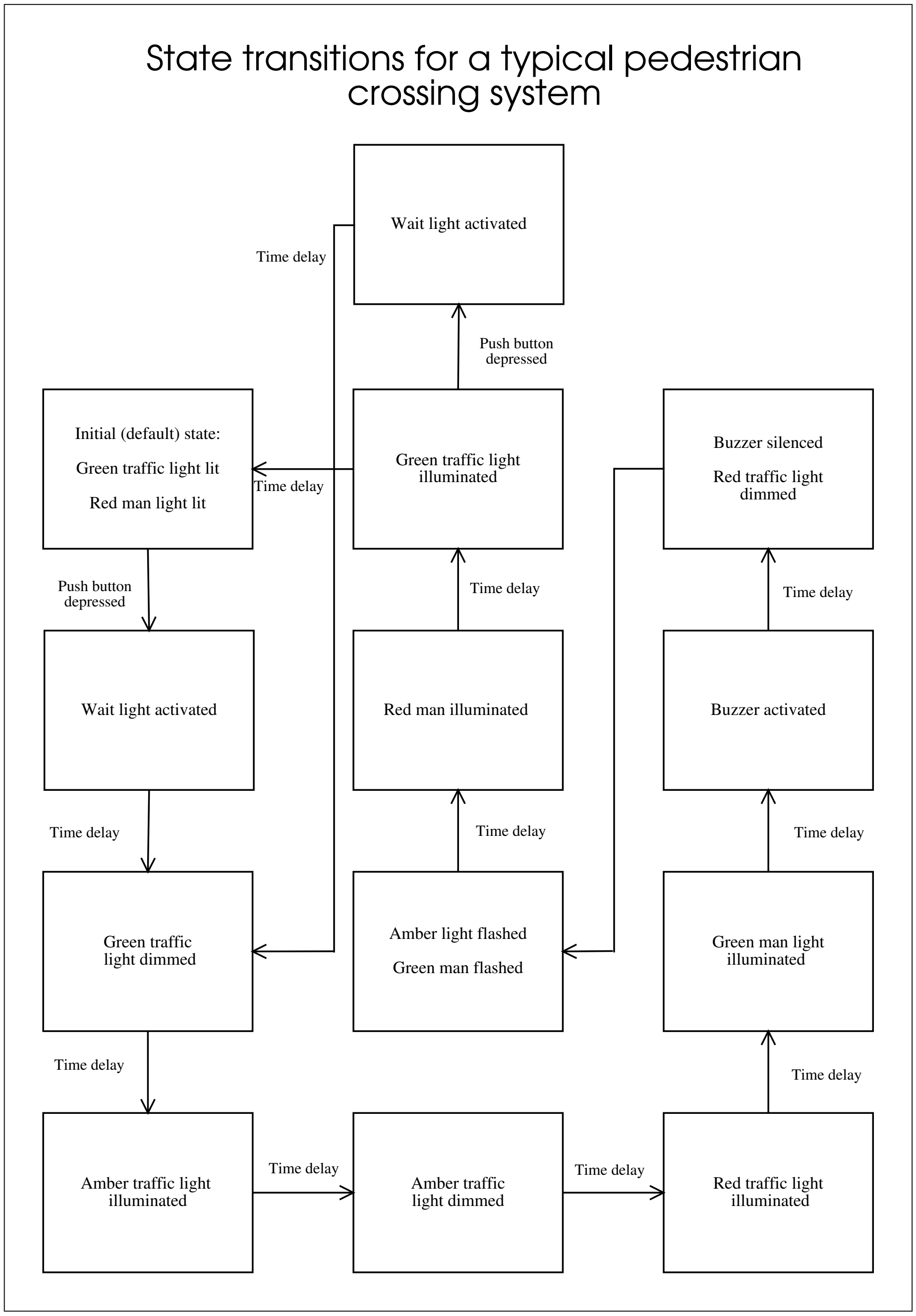


Figure 7: Default Light State

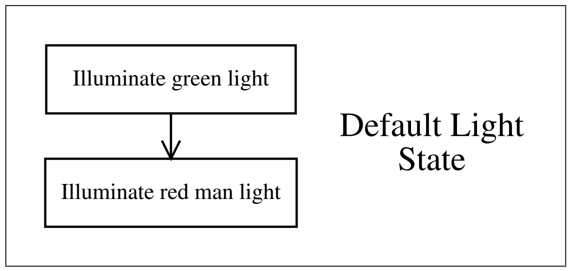


Figure 8: Light Sequence

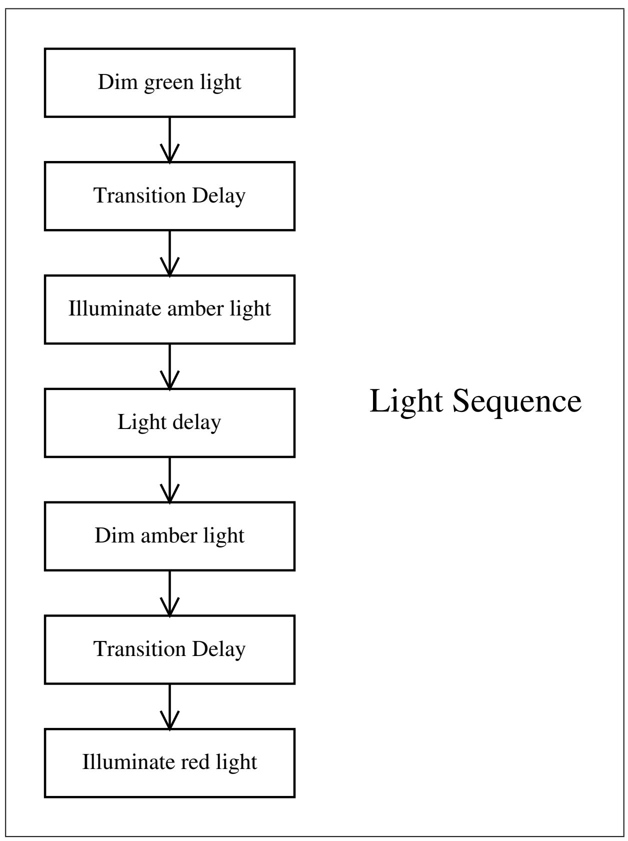
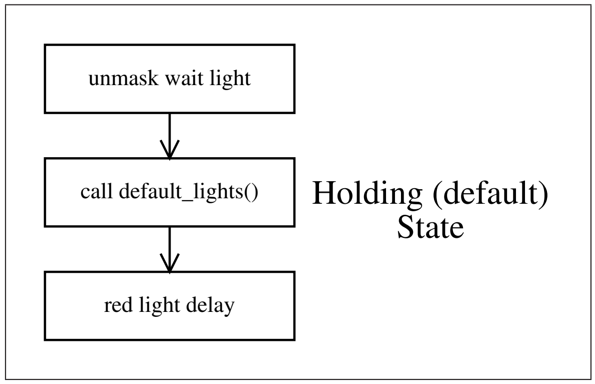


Figure 9: Holding Light State



4.6 Development Strategy

The development strategy I pursued was one of rapid, evolutionary prototyping. The code was successively "evolved" as each of the necessary functions, in turn, were developed. The objective was to have a working system as fast as possible, and to isolate errors to particular functions, as the code gradually increased in size.

Figure 10: Green man & buzzer

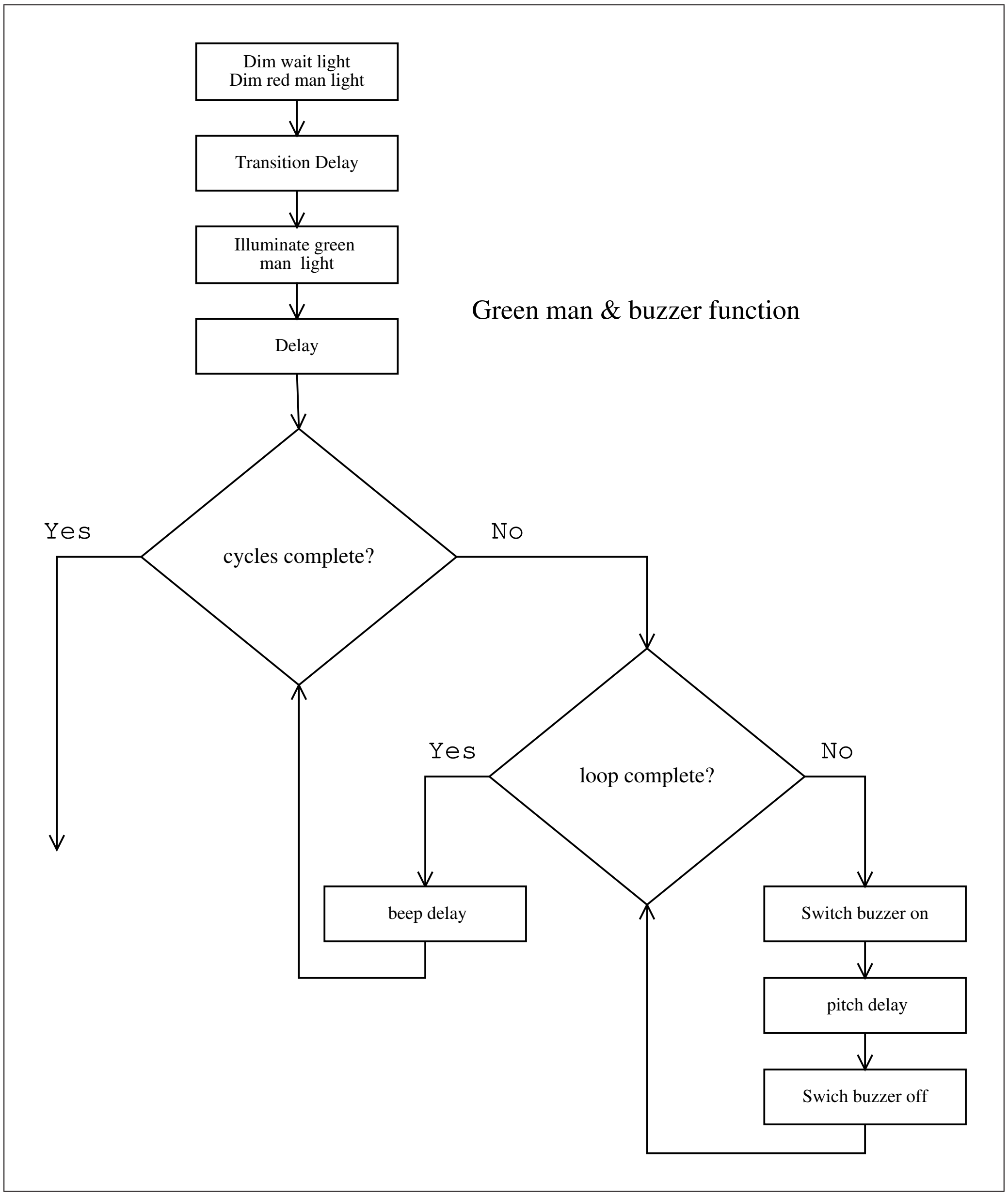
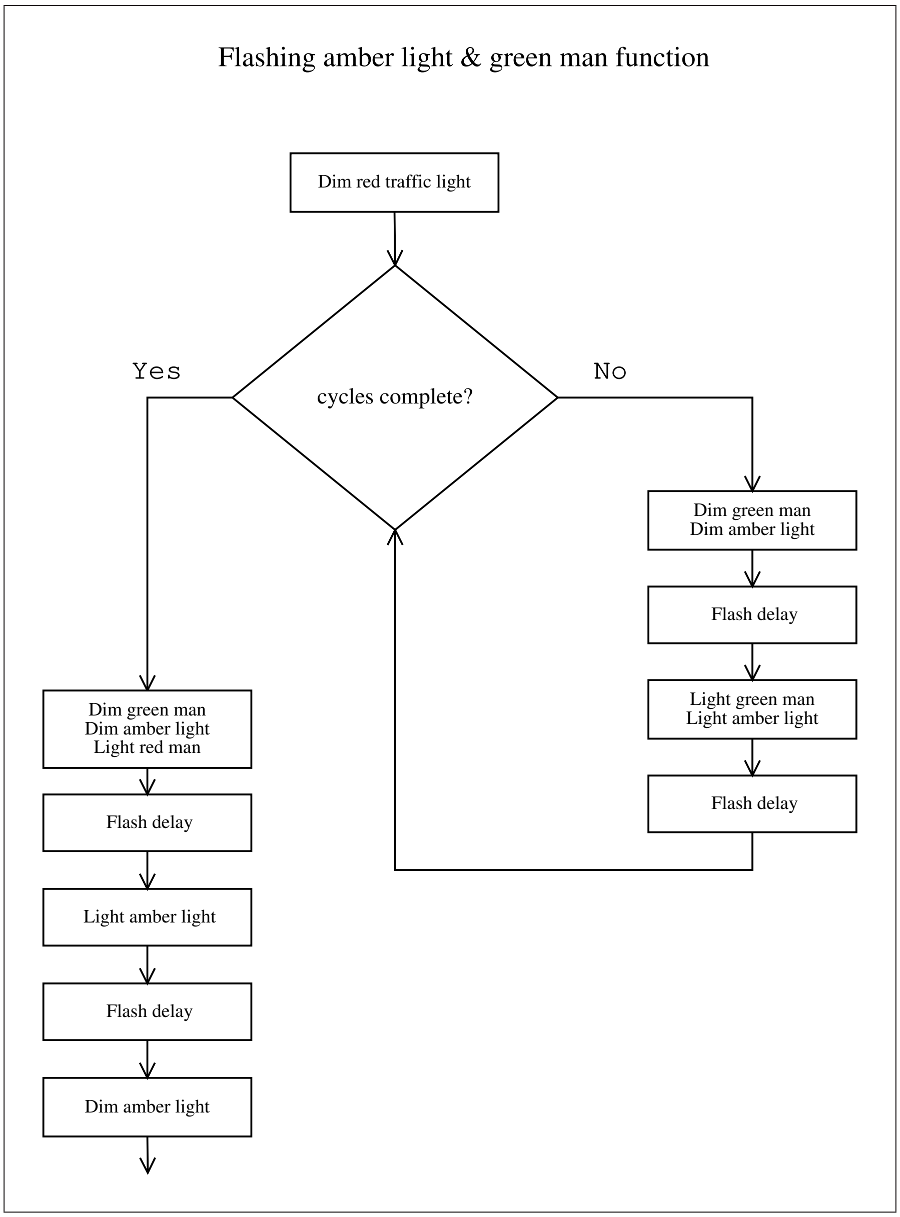


Figure 11: Flashing green man & amber light



5 Source Code

The source code presented here has extensive comments and should be reasonably self–documenting.

5.1 Development History

As described above, the source code was developed with an evolutionary strategy; the source code presented here offers four snapshots of that developmental process. Version 0.8 is the latest version and was found to be functionally stable and sufficiently complete (as per the intial program specification).

5.2 pedes-0.2.c

The fundamental shape and form of the code was established by this stage; all of the traffic light functions, and the pedestrian green and red lights were essentially complete. The successive evolutions of the code were concerned with developing the buzzer and wait light, which are more complex.

```
/*************************************************************
 *  Filename:       pedes-0.2.c                              *
 *  Author:         Aren Tyr                                 *
 *  Version:        0.2                                      *
 *  Description:    Initial version of pedestrian program.   *
 *                  This version has a fully functioning     *
 *                  traffic light sequence operated by a     *
 *                  push button. Buzzer and wait light not   *
 *                  implemented yet.                         *
 *                                                           *
 *-----------------------------------------------------------*
 *  Pin mapping:                                             *
 *                                                           *
 *     PIN_B0:  Push Button                                  *
 *     PIN_B1:  Green traffic light                          *
 *     PIN_B2:  Amber traffic light                          *
 *     PIN_B3:  Red traffic light                            *
 *     PIN_B4:  Green man light                              *
 *     PIN_B5:  Red man light                                *
 *                                                           *
 *************************************************************/

// Pre-processor directives section
#if defined(__PCM__)
    #include <16F84.h>
    #fuses XT, NOWDT, NOPROTECT, PUT
    #use delay(clock = 4000000)
#endif

#byte port_b = 6
```

```
#define trans_delay 750
// 0.75 second light transition delay

#define amber_delay 2000
// 2 second amber light delay

#define red_delay 10000
// 10 second red light delay

#define amber_flash 500
// 0.5 second amber flash delay

#define amber_cycles 10
// do 10 cycles


// forward declarations of functions
void default_lights();
void hold_state_no_input();
void light_sequence();
void green_man_buzzer();
void green_man_amber();


// default traffic light state function:
// 1. Green traffic light illuminated
// 2. Red man light illuminted
void default_lights()
{
    output_high(PIN_B1);
    output_high(PIN_B5);
}

// Prevent any the light sequence from being
// initiated for 10 seconds
void hold_state_no_input()
{
    default_lights();
    // don't allow any inputs for a while
    delay_ms(red_delay);

    // ...now return to main() loop
}

// Main traffic light sequence (green->amber->red)
void light_sequence()
{
    // light the LEDs
    output_low(PIN_B1);
```

```
    delay_ms(trans_delay);
    output_high(PIN_B2);
    delay_ms(amber_delay);
    output_low(PIN_B2);
    delay_ms(trans_delay);
    output_high(PIN_B3);
    delay_ms(trans_delay);

    // call the green man & buzzer function
    green_man_buzzer();
}

// light green man and start buzzer (not implemented yet)
void green_man_buzzer()
{
    // suppress red man, light green man
    output_low(PIN_B5);
    delay_ms(trans_delay);
    output_high(PIN_B4);

    // wait for a specified length of time
    delay_ms(red_delay);

    green_man_amber();
}

// flash green man light and amber light simultaneously
void green_man_amber()
{
    int i = 0;

    // unlight the red traffic light
    output_low(PIN_B3);

    // flash the green man and amber light
    for(i = 0; i < amber_cycles; i++)
    {
        output_low(PIN_B4);
        output_low(PIN_B2);
        delay_ms(amber_flash);
        output_high(PIN_B4);
        output_high(PIN_B2);
        delay_ms(amber_flash);
    }

    output_low(PIN_B4);
    output_high(PIN_B5);
    delay_ms(amber_flash);
    output_high(PIN_B2);
    delay_ms(amber_flash);
```

```
    output_low(PIN_B2);

    // go to holding state
    hold_state_no_input();
}

void main()
{
    // illuminate the initial (default) lights
    default_lights();

    setup_counters(RTCC_INTERNAL, WDT_18MS);

    do
    {
        // if button has been pressed
        if(input(PIN_B0))
        {
            while(input(PIN_B0))
                delay_ms(1); // user held down button

            // start off the light sequence
            light_sequence();
        }

    }while (TRUE);
}
```

5.3 pedes-0.4.c

```c
/***************************************************************
 *  Filename:        pedes-0.4.c                              *
 *  Author:          Aren Tyr                                 *
 *  Version:         0.4                                      *
 *  Description:     Initial attempt for buzzer for           *
 *                   pedestrian program. This version has an  *
 *                   incorrect but partially functioning      *
 *                   buzzer. Wait light not                   *
 *                   implemented yet.                         *
 *                                                            *
 *------------------------------------------------------------*
 *  Pin mapping:                                              *
 *                                                            *
 *     PIN_B0:  Push Button                                   *
 *     PIN_B1:  Green traffic light                           *
 *     PIN_B2:  Amber traffic light                           *
 *     PIN_B3:  Red traffic light                             *
 *     PIN_B4:  Green man light                               *
 *     PIN_B5:  Red man light                                 *
 *     PIN_B6:  Buzzer                                        *
 *                                                            *
 **************************************************************/

// Pre-processor directives section
#if defined(__PCM__)
    #include <16F84.h>
    #fuses XT, NOWDT, NOPROTECT, PUT
    #use delay(clock = 4000000)
#endif

#byte port_b = 6

#define trans_delay 750
// 0.75 second light transition delay

#define amber_delay 2000
// 2 second amber light delay

#define red_delay 10000
// 10 second red light delay

#define amber_flash 500
// 0.5 second amber flash delay

#define amber_cycles 10
// do 10 cycles
```

```
#define sound_delay 122000 //1906

// flow control booleans
int delayCount = 0;
boolean soundOn = 0;
int sOn = 2500;

// forward declarations of functions
void default_lights();
void hold_state_no_input();
void light_sequence();
void green_man_buzzer();
void green_man_amber();
void sound_buzzer();

// default traffic light state function:
// 1. Green traffic light illuminated
// 2. Red man light illuminted
void default_lights()
{
    output_high(PIN_B1);
    output_high(PIN_B5);
}

// Prevent any the light sequence from being
// initiated for 10 seconds
void hold_state_no_input()
{
    default_lights();
    //don't allow any inputs for a while
    delay_ms(red_delay);
}

// Main traffic light sequence (green->amber->red)
void light_sequence()
{
    // light the LEDs
    output_low(PIN_B1);
    delay_ms(trans_delay);
    output_high(PIN_B2);
    delay_ms(amber_delay);
    output_low(PIN_B2);
    delay_ms(trans_delay);
    output_high(PIN_B3);
    delay_ms(trans_delay);

    // call the green man & buzzer function
    green_man_buzzer();
}
```

```
// light green man and start buzzer
void green_man_buzzer()
{
    // suppress red man, light green man
    output_low(PIN_B5);
    delay_ms(trans_delay);
    output_high(PIN_B4);

    // switch the buzzer sound on
    soundOn = 1;

    // wait for a specified length of time
    delay_ms(red_delay);

    green_man_amber();
}

// flash green man light and amber light simultaneously
void green_man_amber()
{
    int i = 0;

    // switch the buzzer sound off
    soundOn = 0;

    // unlight the red traffic light
    output_low(PIN_B3);

    // flash the green man and amber light
    for(i = 0; i < amber_cycles; i++)
    {
        output_low(PIN_B4);
        output_low(PIN_B2);
        delay_ms(amber_flash);
        output_high(PIN_B4);
        output_high(PIN_B2);
        delay_ms(amber_flash);
    }

    output_low(PIN_B4);
    output_high(PIN_B5);
    delay_ms(amber_flash);
    output_high(PIN_B2);
    delay_ms(amber_flash);
    output_low(PIN_B2);

    hold_state_no_input();
}

// send an output to the buzzer
```

```
void sound_buzzer()
{
    output_high(PIN_B6);
    delay_us(5);
    output_low(PIN_B6);
}

// Interrupt service routine serviced
// every time the real time clock overflows
#int_rtcc
clock_isr()
{
    // if the sound is on
    if(soundOn == 1)
    {
        while(--sOn > 0)
            sound_buzzer();

        sOn = 2500;
        delay_ms(250);

    }

}

void main()
{
    // initialize real time clock
    set_rtcc(0);
    setup_counters(RTCC_INTERNAL, WDT_18MS);

    // enable real time clock interrupt
    enable_interrupts(INT_RTCC);
    // enable global interrupts (make RTCC interrupt active)
    enable_interrupts(GLOBAL);

    // illuminate the initial (default) lights
    default_lights();
    delayCount = sound_delay;

    do
    {
        if(input(PIN_B0))
        {
            while(input(PIN_B0))
                delay_ms(1); // user held down button

            // start off the light sequence
            light_sequence();
        }
```

```
    }while (TRUE);
}
```

5.4 pedes-0.6.c

This version of the code has substantial sections commented out since extensive alterations were being made to the sections of code responsible for the buzzer. These sections are still present here to illustrate the development of the code.

```
/****************************************************************
 *  Filename:       pedes-0.6.c                                 *
 *  Author:         Aren Tyr                                    *
 *  Version:        0.6                                         *
 *  Description:    Slightly improved attempt for buzzer.       *
 *                  This buzzer beeps, periodically, but the    *
 *                  timing is inconsistent and the pitch        *
 *                  incorrect. The buzzer timing is no longer   *
 *                  controlled via an interrrupt.  The wait     *
 *                  light is partially implemented.             *
 *                                                              *
 *--------------------------------------------------------------*
 *  Pin mapping:                                                *
 *                                                              *
 *     PIN_B0:  Push Button                                     *
 *     PIN_B1:  Green traffic light                             *
 *     PIN_B2:  Amber traffic light                             *
 *     PIN_B3:  Red traffic light                               *
 *     PIN_B4:  Green man light                                 *
 *     PIN_B5:  Red man light                                   *
 *     PIN_B6:  Buzzer                                          *
 *     PIN_B7:  Wait notification light                         *
 *                                                              *
 ****************************************************************/

// Pre-processor directives section
#if defined(__PCM__)
    #include <16F84.h>
    #fuses XT, NOWDT, NOPROTECT, PUT
    #use delay(clock = 4000000)
#endif

#byte port_b = 6

#define trans_delay 750
// 0.75 second light transition delay

#define amber_delay 2000
// 2 second amber light delay

#define red_delay 10000
// 10 second red light delay

#define amber_flash 500
```

```
// 0.5 second amber flash delay

#define amber_cycles 10
// do 10 cycles

#define sound_delay 122000 //1906

int delayCount = 0;
boolean soundOn = 0;
boolean soundControl = 0;

long soundCycles = 50000;

// forward declarations of functions
void default_lights();
void hold_state_no_input();
void light_sequence();
void green_man_buzzer();
void green_man_amber();
// void sound_buzzer();

// default traffic light state function:
// 1. Green traffic light illuminated
// 2. Red man light illuminted
void default_lights()
{
    output_high(PIN_B1);
    output_high(PIN_B5);
}

// Prevent any the light sequence from being
// initiated for 10 seconds
void hold_state_no_input()
{
    long k;

    default_lights();

    // holding loop that attempts to allow the
    // wait notification light to be illuminated
    // whilst the appropriate delay is still
    // maintained
    for(k = 0; k < 200; k = k + 0.5)
    {

        if(input(PIN_B0))
        {
            while(input(PIN_B0))
                delay_ms(1); // user held down button
```

```
            output_high(PIN_B7);
        }
    }

}

// Main traffic light sequence (green->amber->red)
void light_sequence()
{
    // light the LEDs
    output_low(PIN_B1);
    delay_ms(trans_delay);
    output_high(PIN_B2);
    delay_ms(amber_delay);
    output_low(PIN_B2);
    delay_ms(trans_delay);
    output_high(PIN_B3);
    delay_ms(trans_delay);


    // call the green man & buzzer function
    green_man_buzzer();
}

// light green man and start buzzer
void green_man_buzzer()
{
        // local loop counters
    int i, j = 0;

    // suppress red man, light green man
    output_low(PIN_B5);
    delay_ms(trans_delay);
    output_high(PIN_B4);

    output_low(PIN_B7);

    // switch the buzzer sound on
    soundOn = 1;

    // beep the buzzer using a nested loop
    // (also indirectly delays)
    for(j=0; j<100; j++)
    {
        for(i = 0; i<100; i++)
        {
            output_high(PIN_B6);
            delay_us(200);
            output_low(PIN_B6);
            --soundCycles;
```

```
        }

        delay_ms(100);
    }


    green_man_amber();
}

// flash green man light and amber light simultaneously
void green_man_amber()
{
    int i = 0;

    // switch the buzzer sound off
    soundOn = 0;

    // unlight the red traffic light
    output_low(PIN_B3);

    // flash the green man and amber light
    for(i = 0; i < amber_cycles; i++)
    {
        output_low(PIN_B4);
        output_low(PIN_B2);
        delay_ms(amber_flash);
        output_high(PIN_B4);
        output_high(PIN_B2);
        delay_ms(amber_flash);
    }

    output_low(PIN_B4);
    output_high(PIN_B5);
    delay_ms(amber_flash);
    output_high(PIN_B2);
    delay_ms(amber_flash);
    output_low(PIN_B2);

    hold_state_no_input();
}

/*
   DEPRECATED: Now part of green_man_buzzer() function

void sound_buzzer()
{
    int i;

    soundControl = 1;
```

```
    for(i = 0; i<100; i++)
    {
        output_high(PIN_B6);
        delay_us(300);
        output_low(PIN_B6);
        --soundCycles;

    }

    delay_ms(100);
    soundControl = 0;
}
*/

/*
  DEPRECATED: Control structure is becoming too complex
              too make any guarantees or control over
              timing.
#int_rtcc
clock_isr()
{
    if(soundOn == 1 && soundControl == 0)
        sound_buzzer();

    if(soundOn == 1 && soundControl == 1)
    {
        sound_buzzer();
        soundOn = 0;
        soundCycles = 50000;

    }

    if(soundCycles <= 0 && soundOn == 1)
    {
        soundCycles = 50000;
        soundOn = 0;
    }

    if(soundCycles <= 0 && soundOn == 0)
    {
        soundCycles = 50000;
        soundOn = 1;
    }

    --soundCycles;
}
*/
```

```
void main()
{
        // initialize real time clock
    set_rtcc(0);
    setup_counters(RTCC_INTERNAL, WDT_18MS);

    // setup_counters(RTCC_INTERNAL, RTCC_DIV_256);
    // enable real time clock interrupt
    // enable_interrupts(INT_RTCC);
    // enable global interrupts (make RTCC interrupt active)
    // enable_interrupts(GLOBAL);

    // illuminate the initial (default) lights
    default_lights();
    delayCount = sound_delay;

    do
    {
        if(input(PIN_B0))
        {
            while(input(PIN_B0))
                delay_ms(1); // user held down button

            // illuminate the wait notification light
            output_high(PIN_B7);

            // start off the light sequence
            light_sequence();
        }

    } while (TRUE);
}
```

5.5 pedes-0.8.c

```
/***************************************************************
 *  Filename:       pedes-0.8.c                               *
 *  Author:         Aren Tyr                                  *
 *  Version:        0.8                                       *
 *  Description:    All major functions implemented. The push  *
 *                  button now generates an interrupt, and the *
 *                  wait light correspondingly illuminates     *
 *                  correctly. The buzzer correctly beeps, via *
 *                  a nested loop. Only outstanding issues are *
 *                  ones of precise timing demands and code    *
 *                  abstraction.                              *
 *                                                            *
 *------------------------------------------------------------*
 *  Pin mapping:                                              *
 *                                                            *
 *     PIN_A0:  Green traffic light                           *
 *     PIN_A1:  Amber traffic light                           *
 *     PIN_A2:  Red traffic light                             *
 *     PIN_A3:  Green man light                               *
 *                                                            *
 *     PIN_B0:  Buzzer                                        *
 *     PIN_B1:  Red man light                                 *
 *     PIN_B2:  Wait notification light                       *
 *     PIN_B7:  Push Button                                   *
 *                                                            *
 ***************************************************************/

// Pre-processor directives section
#if defined(__PCM__)
    #include <16F84.h>
    #fuses XT, NOWDT, NOPROTECT, PUT
    #use delay(clock = 4000000)
#endif

#define trans_delay 750
// 0.75 second light transition delay

#define amber_delay 2000
// 2 second amber light delay

#define red_delay 10000
// 10 second red light delay

#define amber_flash 500
// 0.5 second amber flash delay

#define amber_cycles 10
```

```
// do 10 cycles

// boolean used for masking the wait notification light
boolean allowLight = 0;

// boolean regulates the light sequence (non-essential, though)
boolean startLights = 0;

// forward declarations of functions
void default_lights();
void hold_state_no_input();
void light_sequence();
void green_man_buzzer();
void green_man_amber();
void start_program();

// default traffic light state function:
// 1. Green traffic light illuminated
// 2. Red man light illuminted
void default_lights()
{
    output_high(PIN_A0);
    output_high(PIN_B1);
}

// Prevent any the light sequence from being
// initiated for 10 seconds
void hold_state_no_input()
{
    // allow the wait notification light
    allowLight = 1;

    default_lights();

    delay_ms(red_delay);

}

// Main traffic light sequence (green->amber->red)
void light_sequence()
{
    // boolean not strictly necessary, since the hardware
    // protects against recursion
    startLights = 0;

    // light the LEDs
    output_low(PIN_A0);
    delay_ms(trans_delay);
    output_high(PIN_A1);
    delay_ms(amber_delay);
```

```
    output_low(PIN_A1);
    delay_ms(trans_delay);
    output_high(PIN_A2);
    delay_ms(trans_delay);


    // call the green man & buzzer function
    green_man_buzzer();
}

void green_man_buzzer()
{
    // local loop counters
    int i, j = 0;

    // don't allow the wait light here
    allowLight = 0;

    // switch off the wait notification light
    output_low(PIN_B2);

    // suppress red man, light green man
    output_low(PIN_B1);
    delay_ms(trans_delay);
    output_high(PIN_A3);
    delay_ms(amber_delay);

    // beep the buzzer using a nested loop
    // (also indirectly delays)
    for(j=0; j<100; j++)
    {
        for(i = 0; i<100; i++)
        {
            output_high(PIN_B0);
            delay_us(200);
            output_low(PIN_B0);

        }

        delay_ms(100);
    }

    // just to ensure the buzzer is actually off
    output_low(PIN_B0);

    green_man_amber();
}

// flash green man light and amber light simultaneously
void green_man_amber()
```

```
{
    // local loop iterator
    int i = 0;

    // unlight the red traffic light
    output_low(PIN_A2);

    // flash the green man and amber light
    for(i = 0; i < amber_cycles; i++)
    {
        output_low(PIN_A3);
        output_low(PIN_A1);
        delay_ms(amber_flash);
        output_high(PIN_A3);
        output_high(PIN_A1);
        delay_ms(amber_flash);
    }

    output_low(PIN_A3);
    output_high(PIN_B1);
    delay_ms(amber_flash);
    output_high(PIN_A1);
    delay_ms(amber_flash);
    output_low(PIN_A1);

    hold_state_no_input();
}

// interrupt service routine for push button
// will set the light sequence off and
// wait light (where appropriate)
#int_RB
button_isr()
{
    if(input(PIN_B7))
    {
        while(input(PIN_B7))
            delay_ms(1); // user held down button

        // wait light
        if(allowLight == 1)
            output_high(PIN_B2);

        // light sequence boolean
        startLights = 1;
    }

}

// the main part of the program
```

```
void start_program()
{
    do
    {
        if(startLights == 1)
            light_sequence();


    }
    while (TRUE);

}


void main()
{
    // initialize real time clock
    set_rtcc(0);
    setup_counters(RTCC_INTERNAL, WDT_18MS);

    // enable interrupts for Pins B4-B7 (push button is on B7)
    enable_interrupts(int_RB);
    // enable global interrupts (make push button interrupt active)
    enable_interrupts(GLOBAL);

    // illuminate the initial (default) lights
    default_lights();

    // allow the light sequence initially
    allowLight = 1;

    // start the program off
    start_program();

}
```

6 Testing & Test Results

This section documents a complete listing of test results through all versions of the source code. Please refer to Figures 1, 2, 3 and 4 for the meanings of the particular test cases/error codes.

6.1 pedes-0.2.c

Test Case	Importance	Result
ABAA1	Critical	PASS
ABAA2	Critical	PASS
ABAA3	Critical	PASS
ABAA4	Critical	PASS
ABAB1	Critical	PASS
ABAB2	Critical	PASS
ABAB3	Critical	PASS
ABBA1	High	PASS
ABBA2	High	PASS
ABBA3	Low	FAIL
ACA01	Medium	FAIL
ACA02	Medium	FAIL
ACB01	Low	FAIL
ACB02	Low	FAIL

6.2 pedes-0.4.c

Test Case	Importance	Result
ABAA1	Critical	PASS
ABAA2	Critical	PASS
ABAA3	Critical	PASS
ABAA4	Critical	PASS
ABAB1	Critical	PASS
ABAB2	Critical	PASS
ABAB3	Critical	PASS
ABBA1	High	PASS
ABBA2	High	PASS
ABBA3	Low	FAIL
ACA01	Medium	PASS
ACA02	Medium	FAIL
ACB01	Low	FAIL
ACB02	Low	FAIL

6.3 pedes-0.6.c

Test Case	Importance	Result
ABAA1	Critical	PASS
ABAA2	Critical	PASS
ABAA3	Critical	PASS
ABAA4	Critical	PASS
ABAB1	Critical	PASS
ABAB2	Critical	PASS
ABAB3	Critical	PASS
ABBA1	High	PASS
ABBA2	High	PASS
ABBA3	Low	FAIL
ACA01	Medium	PASS
ACA02	Medium	PASS
ACB01	Low	FAIL
ACB02	Low	FAIL

6.4 pedes-0.8.c

Test Case	Importance	Result
ABAA1	Critical	PASS
ABAA2	Critical	PASS
ABAA3	Critical	PASS
ABAA4	Critical	PASS
ABAB1	Critical	PASS
ABAB2	Critical	PASS
ABAB3	Critical	PASS
ABBA1	High	PASS
ABBA2	High	PASS
ABBA3	Low	PASS
ACA01	Medium	PASS
ACA02	Medium	PASS
ACB01	Low	PASS
ACB02	Low	PASS

6.5 Testing Development

The testing stage was extremely important for this project, particularly for the development of the buzzer. Getting the exact timing for the buzzer proved quite difficult when I had it controlled via an interrupt, until I controlled it's beeping via a nested `for` loop instead.

The other major complication that arose was interrupt debugging. During the transition from `pedes-0.6.c` to `pedes-0.8.c`, I decided that it should be the actual push button itself that is serviced via an interrupt (not the buzzer), since a button press could occur at any time and the wait light needs to be able to lit immediately, if appropriate. It was the *users* input that provided the dynamic, unpredicatable element in the system, so logically that was best controlled via an interrupt. However, the wait light was not actually lighting when it should have been from the interrupt service routine. I first tried controlling it more indirectly through use of a masking boolean in the button interrupt service routine, and by putting test instructions in the code to see whether those booleans were actualy getting set. I tried testing it on a different LED to see whether that would make it function correctly. That, too, failed.

The solution was to simply use a different pin for the wait light. On a different pin, all the problems simply disappeared and the program functioned as expected. The likely cause of the complication was that my program was using default I/O, so the compiler was "`tris`ing" the pins for me, with some unexpected results. A more elegant solution would probably have been to use fast I/O, and manually `tris` the pins as needed.

7 User Guide

Installation of the simulation system is simple:

1. Fit the programmed PIC chip into the board.
2. Wire PIN_A2 to the top right red LED (pin 6).
3. Wire PIN_A1 to the right middle amber LED (pin 5).
4. Wire PIN_A0 to the bottom right green LED (pin 4).
5. Wire PIN_A3 to the bottom left green LED (pin 3).
6. Wire PIN_B2 to the middle left amber LED (pin 2).
7. Wire PIN_B1 to the top left red LED (pin 1).
8. Wire PIN_B7 to the first push button (pin 5).
9. Wire PIN_B0 to the speaker (see "spk").

The left bank of LEDs now represents the green man, red man, and wait lights; the right-hand bank represents the traffic light[s].

8 Summary

Progress with the project was initially very rapid. Within a couple of hours I had a programmed chip that had a fully working traffic light sequence, together with green and red man lights, controlled via a button.

Unfortunately, the progress rapidly slowed when it came to development of the buzzer! I found that getting the buzzer to actually emit a signal at roughly the right pitch, with the correct beeping interval, considerably more problematical. The code went through several iterations, and a design alteration, before I finally had a program that functioned as I desired. Originally the buzzer was controlled via an interrupt. The problem with this, however, is that as you add more logic to the program, so the number of machine instructions increases, and so the timing alters.

The solution was to control the buzzer via a nested loop structure, and instead have the push button generating the interrupt. With the button generating the interrupt, it is easy to control when the wait light should be lit (notwithstanding standard I/O issues, see above), and initiate the crossing sequence. It was only half way through the development process that this design revelation hit me; however, because the program code was well modularised, it did not necessitate any major alterations to the other code.

Before this project I had scant experience of any interrupt programming; one of the valuable lessons I have learnt from this project is just how difficult interrupt related timing issues can be. The buzzer went from being an increasingly complex, buggy, inconsistent, interrupt service routine to simply a nested for loop. As with everything, simplicity is always a virtue in programming.