



High Speed Development

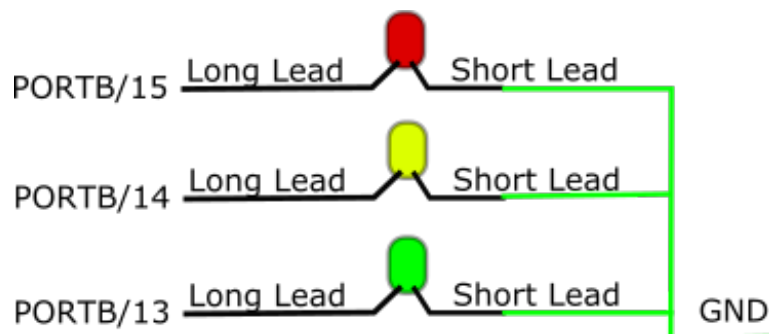
Traffic Lights

(** Please complete [introduction](#) first **)

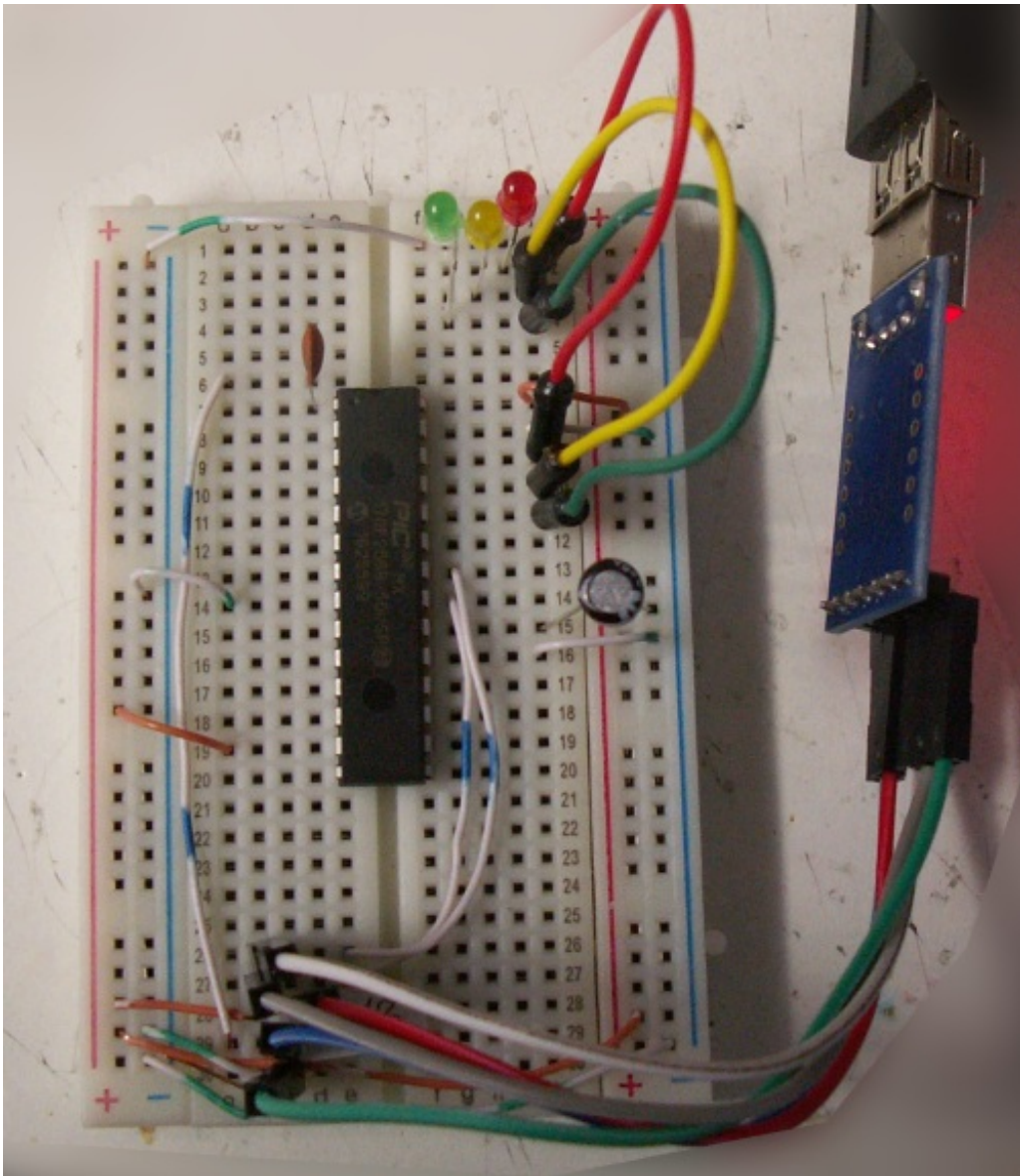
Here we use input output to simulate traffic lights using three leds of appropriate colour. The leds are connected to pins B15(RED), B14(AMBER) and B13(GREEN)

Step 1: Wire the device to the three leds

Because this applies to all PCB's including the IC a simplified diagram is here:

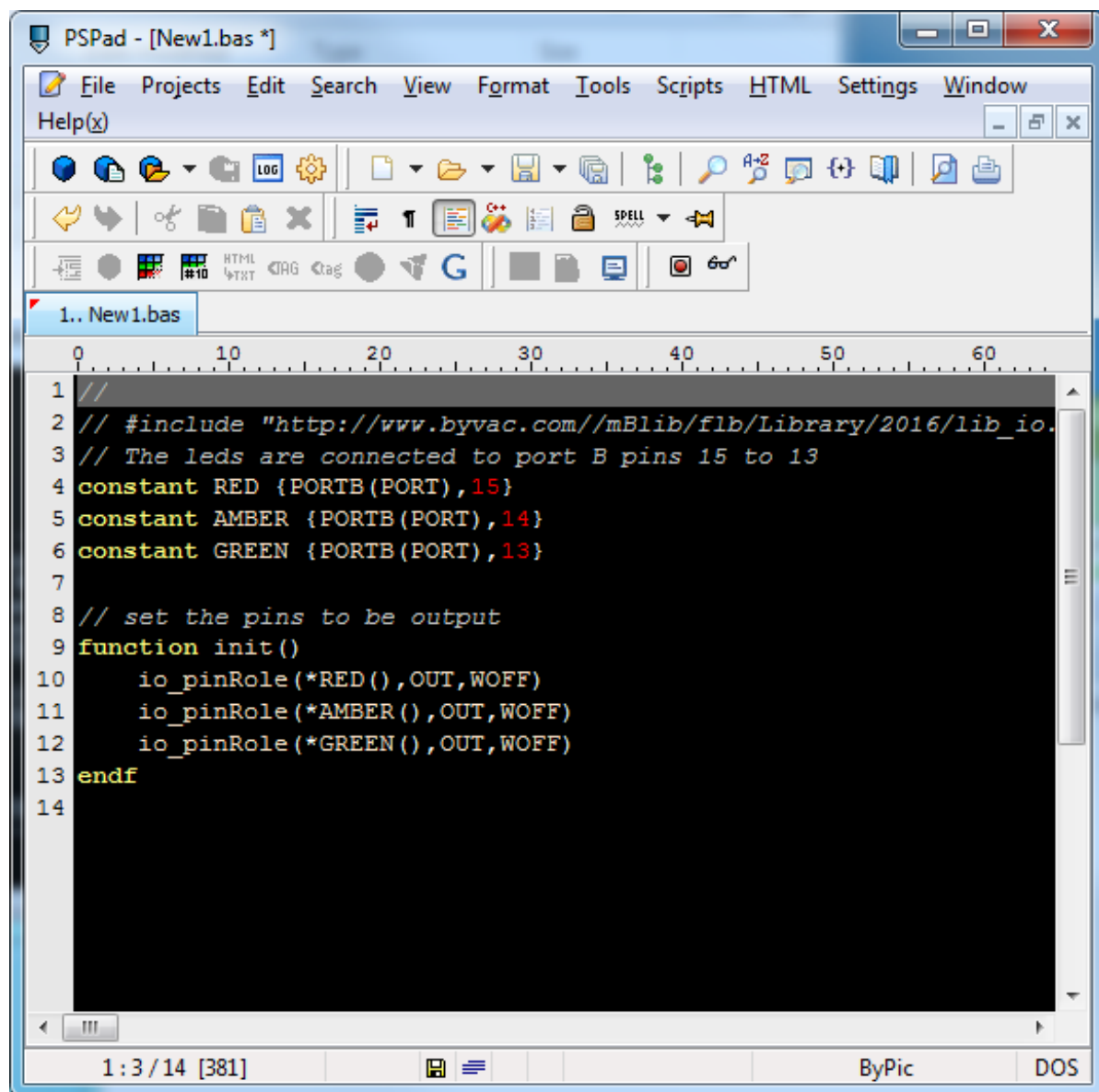


On a bread board it will look something like this



Step 2: Cut and past the following into the editor:

```
//  
// #include "http://www.byvac.com/mBlib/flb/Library/2016/lib_io.bas"  
//  
// The leds are connected to port B pins 15 to 13  
constant RED {PORTB(PORT),15}  
constant AMBER {PORTB(PORT),14}  
constant GREEN {PORTB(PORT),13}  
  
// set the pins to be output  
function init()  
  io_pinRole(*RED(),OUT,WOFF)  
  io_pinRole(*AMBER(),OUT,WOFF)  
  io_pinRole(*GREEN(),OUT,WOFF)  
endf
```



```
1 //
2 // #include "http://www.byvac.com//mBlib/flb/Library/2016/lib_io.
3 // The leds are connected to port B pins 15 to 13
4 constant RED {PORTB(PORT),15}
5 constant AMBER {PORTB(PORT),14}
6 constant GREEN {PORTB(PORT),13}
7
8 // set the pins to be output
9 function init()
10     io_pinRole(*RED(),OUT,WOFF)
11     io_pinRole(*AMBER(),OUT,WOFF)
12     io_pinRole(*GREEN(),OUT,WOFF)
13 endf
14
```

The first thing to notice is the `// #include` at the top of the program, this is a library file which makes input output easier. The 'include' will direct the IDE to place the contents of this file into flash and so will be there semi-permanently.

Lines 10 to 12 relate the hardware to a meaningful name. The syntax may be a little strange but on RED you can see PORTB and 15 making it reasonably obvious. The only function so far is `init()` which will set the lines to output.

Step 3: Send the text over to the device using F4

IMPORTANT wait until it is completed, this will take a little longer the first time as the library is being loaded into flash. Subsequent loads will detect the presence of the library and will not load it again. Note the numbers at the top of the terminal.

```

bserial.exe - Shortcut
[[lib_io.bas](32) constant: RC1
[[lib_io.bas](33) constant: RC2
[[lib_io.bas](34) constant: RC3
[[lib_io.bas](35) constant: RC4
[[lib_io.bas](36) constant: RC5
[[lib_io.bas](37) constant: RC6
[[lib_io.bas](38) constant: RC7
[[lib_io.bas](39) constant: RC8
[[lib_io.bas](40) constant: RC9
[[lib_io.bas](54) function: io_pinRole(*pin(),ionorout,pull)
[[lib_io.bas](71) function: io_pinSet(*pin(),value)
[[lib_io.bas](79) function: io_pinGet(*pin())
[[lib_io.bas](90) function: io_getPort(*port())
[[lib_io.bas](99) function: io_setPort(*port(),value)
[[lib_io.bas](106) function: io_setPortH(*port(),value)
[[lib_io.bas](110) function: io_setPortL(*port(),value)
RAM - New1.bas - RAM
[[New1.bas](3) constant: RED
[[New1.bas](4) constant: AMBER
[[New1.bas](5) constant: GREEN
[[New1.bas](8) function: init()
Done
ok

```

Step 4: type `init()`

Step 5: type `io_pinSet(*RED(),1)` (note the * before the `RED()`)

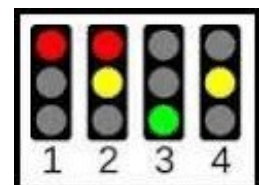
This will illuminate the RED led if not illuminated already, if not check the wiring, do the same for the AMBER and GREEN, `io_pinSet(*RED(),0)` will switch it off. (*HINT Use the up/down arrow keys for command history.*)

Traffic Light States

There are just 4 states that a set of traffic lights can be in that is:

Fig. 6 Traffic light states

We can model this is a function.



Step 6: Add the text below to the editor under the `init()` function, don't forget the blank line at the end:

```

function light_state(n)
  select(n)
  case(1)
    io_pinSet(*RED(),1)
    io_pinSet(*AMBER(),0)
    io_pinSet(*GREEN(),0)
  case(2)
    io_pinSet(*RED(),1)
    io_pinSet(*AMBER(),1)
    io_pinSet(*GREEN(),0)
  case(3)
    io_pinSet(*RED(),0)
    io_pinSet(*AMBER(),0)
    io_pinSet(*GREEN(),1)

```

```

    case(4)
        io_pinSet(*RED(),0)
        io_pinSet(*AMBER(),1)
        io_pinSet(*GREEN(),0)
    endselect
endf

```

Step 7: Send to the device using F4

Step 8: type:

- init()
- light_state(1)
- light_state(2)
- light_state(3)
- light_state(3)

This will test all of the states

Timing

There are a few options we could use here, lets say we want the following: (real timing takes too long)

- (1) RED: 15 seconds
- (2) RA: 2 seconds
- (3) GREEN 15 seconds
- (4) AMB 4 seconds

Method 1 a simple delay

Step 8: Copy and paste text below into the editor below the last function, then press F4

```

function method1()
    init()
    print "Running\r\n"
    while comkey?(2) = 0
        light_state(1) // red for 15 seconds
        wait(15000) // 1000 = 1 second
        light_state(2) // red and amber for 2
        wait(2000)
        light_state(3) // green for 15 seconds
        wait(15000)
        light_state(4) // amber 10 seconds
        wait(10000)
    wend
endf

```

The above uses simple a simple wait to go onto the next state. The 'comkey?(2)' is reading the input from UART2 to which the terminal is connected, when a key is received from the

terminal comkey?(2) will NOT be zero anymore and so the loop will finish. NOTE that this is only read at the start of the sequence so pressing a key at the terminal will only stop the loop when the sequence has finished, this could take up to 42 seconds given the above timings.

The big disadvantage with this method is that the device can't do anything else as the wait() function will tie the device up, known as a blocking function. If the device was needed to do other things then another method is needed.

Method 2 Using Tasks

Built into ByPic is a task scheduling system that is very useful for handing of a job so that we can get on with something else. A task will run a function on a timed interval which is more or less what is needed. We could set up a task for each light however if the timing drifts a bit off then the lights will get out of sequence so we create a function called traffic_ir.

Step 9: copy the traffic_ir function below and add to the end of the existing code.

```
dim current_seconds

function traffic_ir()
  if current_seconds > 40 then
    current_seconds = 0
  endif

  if current_seconds = 0 then // red
    light_state(1)
  endif
  if current_seconds = 15 then // red+amber
    light_state(2)
  endif
  if current_seconds = 20 then // green
    light_state(3)
  endif
  if current_seconds = 35 then // amber
    light_state(4)
  endif
  current_seconds = current_seconds + 1 // update count
endf
```

Note that there is also a variable defined called current_time, this will keep track of the number of seconds that have passed, see as a particular value is reached it changes the state of the traffic lights.

Step 10: Add the code below to the end of the file

```
function method2()
  init()
  current_seconds = 0
```

```
taskadd("traffic_ir()",1,1000,1)
endf
```

After loading the code with F4 type `method2()` This uses the built in scheduler to run the sequence. Note that you can still type into the terminal and even disrupt the sequence. Try `print current_seconds` to see where it is up to.

To stop the sequence type `taskclr()`

Method 3 Using a Timer

This is a bit more complex but will allow other operations when the program is running, similar to the task. Times are more accurate than the task and so should be reserved for accurate timings. This method uses a timer and interrupt, the timer will trigger the interrupt every second and a global variable will keep track of the seconds

Step 9: Add the two lines below to the top of the file, **above** the `lib.io.bas` include file.

```
// #include "http://www.byvac.com/mBlib/flb/Library/2016/lib_tmr.bas"
// #include "http://www.byvac.com/mBlib/flb/Library/2016/lib_ir.bas"
```

And this text to the end of the current program

```
function method3()
  init()
  current_seconds = 0
  print "Running\r\n"
  tmr_init(*TIMER23(),7,312500) // 1 second
  ir_set(*TIMER23(),3,"traffic_ir")
endf
```

Step 10: IMPORTANT type `reset` or `taskclr()` before loading the above program. This will clear the schedule as it will still try to do its scheduling whilst downloading is taking place and this will interfere with the download process. Use |A to send the whole thing over again.

Step 11: Type `method3()`