‘C’ PreProcessor Tricks

Passing Multiple Parameters Values in a Single Macro Parameter

Contents

Use Case

Everyone knows how to pass multiple parameters to a pre-processor macro.

What is not so obvious is how to pass them all in a single parameter. I can hear you asking – why wouldn’t you just use multiple parameters? Well here is an example of why this is important.

Working on micro-controllers requires accessing SFRs (special function registers) to control things like port pins. Typically there are multiple SFRs relating to a specific port/pin that have to be accessed to implement functionality on a specific port/pin (ex: pin data, pin direction, pin pull-up, …)

There are two specific issues that this trick addresses:

1. Code can quickly get confusing when there are multiple port/pins required to implement a function and they are all referred to by their numeric definition (ex: TRISCbits.TRISC2) versus a more mnemonic name (ex: TRIS(MASTERCLK)).
2. It can be a real nightmare to have to later modify working code to move a port/pin functionality from one pin to another. Inevitably, a reference is missed, or changed incorrectly resulting in hours of wasted debug time.

Problem Addressed

The ‘C’ pre-processor does not have an obvious way to ‘pick apart’ a macro parameter which is why this isn’t usually done.

However you can take advantage of the fact that the pre-processor makes multiple passes to resolve name definitions to handle this.

Solution

Two macros must be used for each case as follows:

```
#define     TRIS(a)     XTRIS(a)
#define     XTRIS(a,b)  TRIS ## a ## bits.TRIS ## a ## b
```

By invoking the first macro in your code with multiple parameters as follows:

```
TRIS(C,2)
```

The pre-processor will pass the literal string “C,2” to the second macro XTRIS. This will work in the usual manner to provide multiple parameters to XTRIS.

This, of course, assumes that you separate your parameters with commas. It will not work otherwise because the pre-processor expects macro parameters to be comma separated.

The last step is an obvious one of replacing the “C,2” string with a macro as follows.

```
#define     MASTERCLK   C,2
```

Then you can do things like the following:

```
TRIS(MASTERCLK) = 1;

If (TRIS(MASTERCLK) == 1) do something
```

Which the pre-processor will reduce to the following:

```
TRISCbits.TRISC2 = 1;
if (TRISCbits.TRISC2 == 1) do something
```

Example

The following code shows how the code is cleaned up by using these macros (see full definitions in MicroChip 16F Family Port Specific Macros below.)

Source History

```
16
17  #define USE_LED_TO_DEBUG    0
18
19  // define which ports pins are used for this functionality (requires PortBitMacros.h)
20  #define MASTERCLK   C,2
21  #define MASTERDATA  A,5
22
23  #if USE_LED_TO_DEBUG
24  #define LEDPORT     B,7
25  #endif
26
27  void Init_2WireLink_Master(void){
28
29  #if USE_LED_TO_DEBUG
30      TRIS(LEDPORT) = 0;
31      PORT(LEDPORT) = 0;
32  #endif
33
34      // set clk/data outputs high/low
35      PORT(MASTERCLK) = 1;
36      PORT(MASTERDATA) = 0;
37
38      // set data port to open-collector so either master/slave can drive it low
39      // OCDON open-collector=1, normal=0
40      ODCON(MASTERCLK) = 0;
41      ODCON(MASTERDATA) = 1;
42
43      // set clock to be a hard output
44      // default to data is an input so that weak pull-up drives line high
45      // TRIS input=1, output=0
46      TRIS(MASTERCLK) = 0;
47      TRIS(MASTERDATA) = 1;
48
49      // enable weak pull-ups on data
50      WPU(MASTERCLK) = 0;
51      WPU(MASTERDATA) = 1;
52  }
```

MicroChip 16F Family Port Specific Macros

Here is the implementation for a specific chip set (16F15244) port registers.

```
// this set is required to force the preprocessor to handle two levels of indirection
// these should not be used in your code
#define XPORT(a,b)      PORT ## a ## bits.R ## a ## b
#define XTRIS(a,b)      TRIS ## a ## bits.TRIS ## a ## b
#define XLAT(a,b)       LAT ## a ## bits.LAT ## a ## b
#define XANSEL(a,b)     ANSEL ## a ## bits.ANS ## a ## b
#define XWPU(a,b)       WPU ## a ## bits.WPU ## a ## b
#define XODCON(a,b)     ODCON ## a ## bits.ODC ## a ## b
#define XSLRCON(a,b)    SLRCON ## a ## bits.SLRC ## a ## b
#define XINLVL(a,b)     INLVL ## a ## bits.INLVL ## a ## b
#define XIOCP(a,b)      IOC ## a ## Pbits.IOC ## a ## P ## b
#define XIOCN(a,b)      IOC ## a ## Nbits.IOC ## a ## N ## b
#define XIOCF(a,b)      IOC ## a ## Fbits.IOC ## a ## F ## b

// these macros should be used in your code to facilitate easily moving functionality
// from one port pin to another later.
#define PORT(a)     XPORT(a)        // PORT pin level: high=1, low=0
#define TRIS(a)     XTRIS(a)        // TRIS input=1, output=0 (1=default)
#define LAT(a)      XLAT(a)         // LAT output latch level: high=1, low=0
#define ANSEL(a)    XANSEL(a)       // ANSEL analog=1, digital=0 (1=default)
#define WPU(a)      XWPU(a)         // WPU weak pull-up=1, no pull-up=0 (0=default)
#define ODCON(a)    XODCON(a)       // OCDON open-collector=1, normal=0 (0=default)
#define SLRCON(a)   XSLRCON(a)      // SLRCON pin slew rate: limited=1, max=0 (1=default)
#define INLVL(a)    XINLVL(a)       // INLVL pin input level: Schmitt trigger=1, TTL=0 (1=default)
#define IOCP(a)     XIOCP(a)        // IOCP rising edge IOC interrupt enable
#define IOCN(a)     XIOCN(a)        // IOCN falling edge IOC interrupt enable
#define IOCF(a)     XIOCF(a)        // IOCF interrupt occurred
```