dsPIC Helper Library 1.0

Dean Ferreyra
dean@octw.com
http://www.bourbonstreetsoftware.com/

November 14, 2008

Contents

Chapter 1

Introduction

The dsPIC Helper Library is a small collection of macros and functions written in C that I have found useful while developing and prototyping Microchip dsPIC-based embedded software. It includes a 16-bit CRC-CCITT implementation, a stand-in for the missing `<stdint.h>` header file for defining types like `int16_t`, a stand-in for the missing `<stdbool.h>` header file for defining the Boolean type, and various low-level macros emitting special instructions, like `CLRWDT` to clear the watchdog timer.

I have released this code under the GNU General Public License with the hope that others might find it useful.

Chapter 2

Building From Sources

To build the dsPIC Helper Library, you need to have already installed the Microchip MPLAB C30 cross-compiler. You also need a basic Unix-like build environment with programs like Make and Sed. Under Windows I personally install and use Cygwin.

To build the dsPIC Helper Library, first unpack the source tree. You can do this with the following command:

```
$ tar -xzf dspic-helper-1.0-src.tgz
```

Next, change directory to the head of the source tree that was just unpacked and run the `make` command:

```
$ cd dspic-helper-1.0
$ make
```

This should build the dsPIC Helper Library for the supported dsPIC microcontrollers. This includes the dsPIC30F6015, dsPIC30F6010A, and the dsPIC30F6010, though you can easily modify the `Makefile` to build for other targets as well.

2.1 Library Types

The build process actually builds two libraries for each microcontroller type; one for the default code size, and one using the `-mlarge-code` option. The libraries are built in separate directories named after the corresponding microcontroller and which option was enabled. For example, for the dsPIC30F6015, the libraries can be found here:

- `src/30F6015/libdspic-helper.a`
- `src/30F6015-large-code/libdspic-helper.a`

Chapter 3

Boolean Type

The Microchip MPLAB C30 cross-compiler does not come with the C99 standard header `<stdbool.h>`. The dsPIC Helper Library provides the `boolean.h` to take its place. It defines the type `bool` and the `true` and `false` manifest constants.

3.1 Usage

To use these types, you can use the `-I` compiler option to add the dsPIC Helper Library source directory to the compiler command line, and then include the header file like so:

```
  #include <boolean.h>
```

Chapter 4

Integer Types

The Microchip MPLAB C30 cross-compiler does not come with the C99 standard header `<stdint.h>`. The dsPIC Helper Library provides the `inttypes.h` to take its place. It defines the following signed integer types:

- `int8_t`
- `int16_t`
- `int32_t`
- `int64_t`

and the following unsigned integer types:

- `uint8_t`
- `uint16_t`
- `uint32_t`
- `uint64_t`

It also defines the corresponding minimum and maximum limits, for example `INT16_MIN` and `INT16_MAX`.

4.1 Usage

To use these types, you can use the `-I` compiler option to add the dsPIC Helper Library source directory to the compiler command line, and then include the header file like so:

```
#include <inttypes.h>
```

Chapter 5

Interrupt Macros

The `dspic-utility.h` header file defines a number of macros for manipulating the interrupt priority level (IPL) of the microcontroller, and for using the `DISI` instruction.

5.1 Usage

To use these macros, you can use the `-I` compiler option to add the dsPIC Helper Library source directory to the compiler command line, and then include the header file like so:

```
#include <dspic-utility.h>
```

5.2 Macros

`CLI()` ***macro***

This macro raises the IPL to 7, disabling all maskable interrupts. The previous IPL is saved in a global variable for use by `STI()`.

`STI()` ***macro***

This macro restores the IPL to the value saved by the `CLI()` macro.

`SET_CPU_IPL_MAX()` ***macro***

This macro raises the IPL to 7, disabling all maskable interrupts. Unlike `CLI()`, the current IPL value is not saved.

`DISI_SET(int n)` ***macro***

This macro emits the `DISI` instruction with the given count. This disables interrupts for $\mathtt{n} + 1$ cycles.

`DISI_MAX()` ***macro***

This macro emits the `DISI` instruction to disable interrupts for the maximum number of cycles. This macro can be used with `DISI_CLEAR()` to disable interrupts for a reasonably short, but unknown number of instructions.

`DISI_CLEAR()` ***macro***

This macro clears the `DISICNT` register to re-enable interrupts after a call to `DISI_MAX()` or `DISI_SET()`.

Chapter 6

Low-level Instructions

The `dspic-utility.h` header file defines a number of macros for emitting special instructions, like `CLRWDT` to clear the watchdog timer.

6.1 Usage

To use these macros, you can use the `-I` compiler option to add the dsPIC Helper Library source directory to the compiler command line, and then include the header file like so:

```
#include <dspic-utility.h>
```

6.2 Macros

`NOP()` ***macro***

Emits the `NOP` instruction; i.e., the "no operation" instruction.

`RESET()` ***macro***

Emits the `RESET` instruction to force a software reset of the microcontroller.

`CLRWDT()` ***macro***

Emits the `CLRWDT` instruction to clear the watchdog timer.

Chapter 7

X and Y Data Spaces

The `dspic-utility.h` header file defines two macros that tell the compiler to allocate storage in the dsPIC X and Y data spaces.

7.1 Usage

To use these macros, you can use the `-I` compiler option to add the dsPIC Helper Library source directory to the compiler command line, and then include the header file like so:

```
#include <dspic-utility.h>
```

7.2 Macros

`XSPACE` ***macro***

Expands to a compiler attribute to tell the compiler to allocate storage in the X data space. For example, the following code will allocate the data for the array in the X data space:

```
int16_t abcCoeffs[3] XSPACE;
```

`YSPACE` ***macro***

Expands to a compiler attribute to tell the compiler to allocate storage in the Y data space. For example, the following code will allocate the data for the array in the Y data space:

```
int16_t controlHistory[3] YSPACE;
```

Chapter 8

16-bit CRC-CCITT

The `dspic-utility.h` header file defines two functions for calculating the 16-bit CRC-CCITT value of a stream of bytes.

These functions are based on an article by Joe Geluso, originally located at `http://www.joegeluso.com/software/articles/ccitt.htm`. Unfortunately, it seems that that site no longer exists, but a copy of the article can be found in the Wayback Machine at `http://web.archive.org/web/20071229021252/http://www.joegeluso.com/software/articles/ccitt.htm`.

8.1 Usage

To use these functions, you can use the `-I` compiler option to add the dsPIC Helper Library source directory to the compiler command line, and then include the header file like so:

```
  #include <crc-ccitt.h>
```

8.2 Functions

`uint16_t crc_ccitt(uint16_t crc, char byte)` ***function***

Call this function with the first byte in the stream, using an initial `crc` value of `CRC_CCITT_INITIAL_VALUE`. The function returns the `crc` value to use in the call for the next byte in the stream. Once all the bytes have been processed, call `crc_ccitt_normalize()` to retrieve the actual CRC value.

`uint16_t crc_ccitt_normalize(uint16_t crc)` ***function***

Once all the bytes have been processed, call this function with `crc` set to the last value returned by `crc_ccitt()` to retrieve the actual CRC value.

8.3 Example

Here's a simple example that calculates the 16-bit CRC-CCITT value for the string "123456789". A call to test() will return value 0xE5CC:

```
#include <crc-ccitt.h>

uint16_t test(void)
{
    const char* str = "123456789";
    uint16_t crc = CRC_CCITT_INITIAL_VALUE;
    while (*str)
        crc = crc_ccitt(crc, *str++);
    return crc_ccitt_normalize(crc);
}
```

Index