

MSC12xx Programming with SDCC

Charles Repetti

Data Acquisition Product – Microsystem

Charlie@Voter-Guide.com

ABSTRACT

The MSC12xx is a “system on a chip” which embeds an 8051 microcontroller, a 24 bit analog-to-digital converter (ADC), 4-32K bytes of flash memory, and a set of peripherals in one package. Programming the 8051 is an important part of a complete product development effort. “SDCC” [1] is one of the options for 8051 software development. SDCC is available to anyone free of charge, and is delivered with all of its source code. The entire tool flow runs under Microsoft Windows™.

Contents

INTRODUCTION.....	2
INSTALLATION.....	2
CYGWIN.....	2
SDCC.....	4
SOURCE CODE PREPARATION.....	5
C SOURCE CODE.....	5
ASSEMBLY LANGUAGE.....	7
LIBRARIES.....	7
RUNNING THE “HELLO WORLD” PROGRAM.....	8
MAKEFILE.....	8
DOWNLOADING AND RUNNING.....	8
EXERCISING THE ANALOG TO DIGITAL CONVERTER.....	9
USING FLOATING POINT IN SDCC.....	9
USING THE ADC.....	10
RUNNING THE PROGRAM.....	12
CONCLUSION.....	13
END NOTES.....	14

Introduction

SDCC is a compelling choice for software development on the msc12xx for at least two reasons. First, it is available free of charge. Perhaps more importantly, it is delivered with all of the source code for the tools and the libraries, as well as for numerous helpful examples. This means that if a problem arises, the developer may fix it using the supplied source code.

We note the differences between the formats of the source and object code TI supplies and the format our GNU-Linux expects, and provide an automatic method for converting between them. Finally, we run an example on an MSC12xxEVM.

Installation

Before beginning, it is a good idea to install and test the downloader TI ships on the CD supplied with your MSC12xxEVM [2]. We will use the downloader later to run our first program.

We first install “CYGWIN” [3], which runs as a guest under Microsoft Windows. We install this as a binary, then install the GNU [4] compiler (known as “GCC”), and then use it to build “SDCC”. SDCC is bundled with “ASXXX” and “ASLINK”, which are a freeware assembler and linker, respectively. While SDCC binaries are available for Microsoft Windows, we use a number of the other GNU tools in this application note, so we work with CYGWIN.

Microsoft Windows is not shipped with any of the GNU tools installed, so a bit of work is required to simply prepare CYGWIN. This is conveniently accomplished with the advanced tools which are a part of the CYGWIN package, but downloading them is time consuming. Of course CYGWIN is not simply useful for developing on the MSC12xx, so the effort may have additional benefits.

Beyond the base CYGWIN package, it is necessary to install the CYGWIN developer’s kit to get the GCC compiler. Then SDCC must be obtained from Sourceforge.

Finally, the TI downloader must be installed. This may be obtained either from the MSC12xxEVM CD or from the TI web site mentioned in the End Notes.

CYGWIN

The home page of CYGWIN, <http://www.cygwin.com>, is the place to start. A program named “setup.exe” must be copied to your computer. This allows for CYGWIN components to be installed onto your computer. The “Base” category may be installed first, and the “Devel” category added next. If there are problems with timeouts over your network connection, it may be

necessary to install the components piecemeal, or perhaps to download the components before installing them.

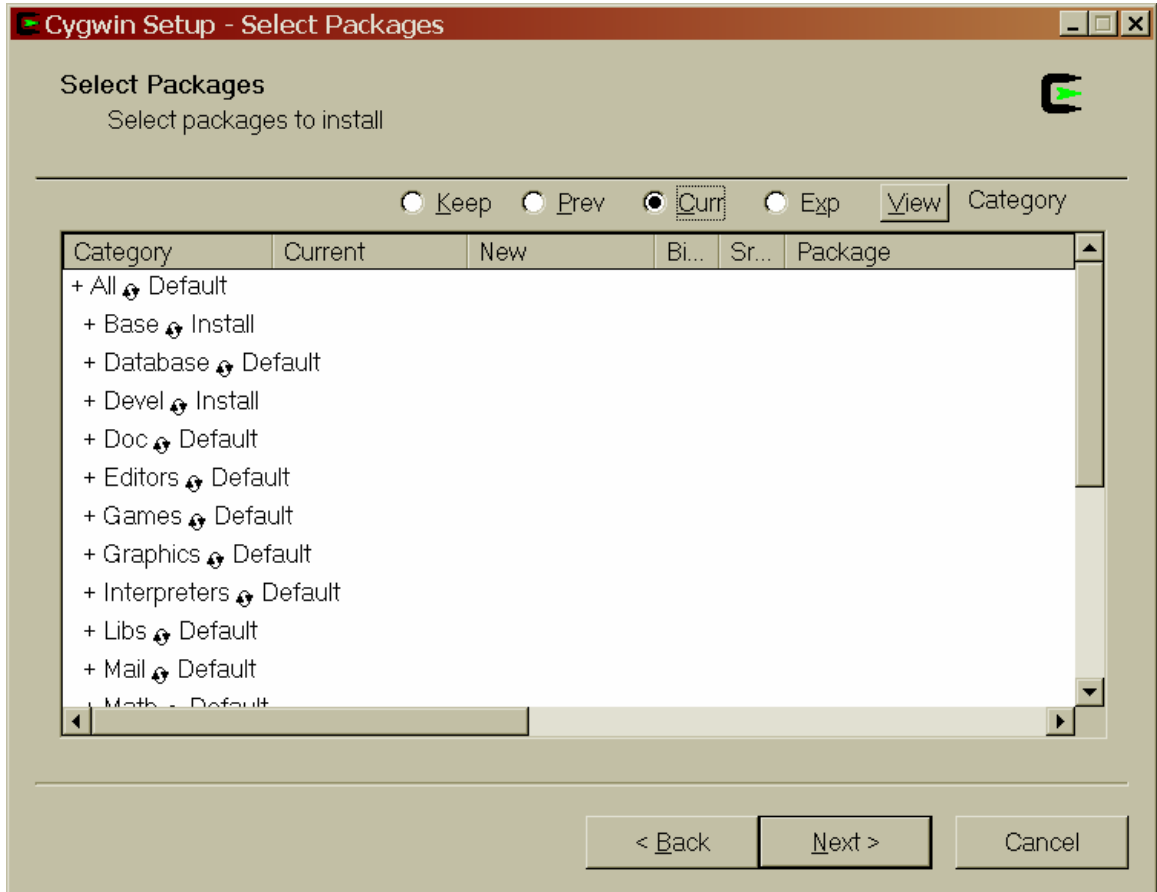
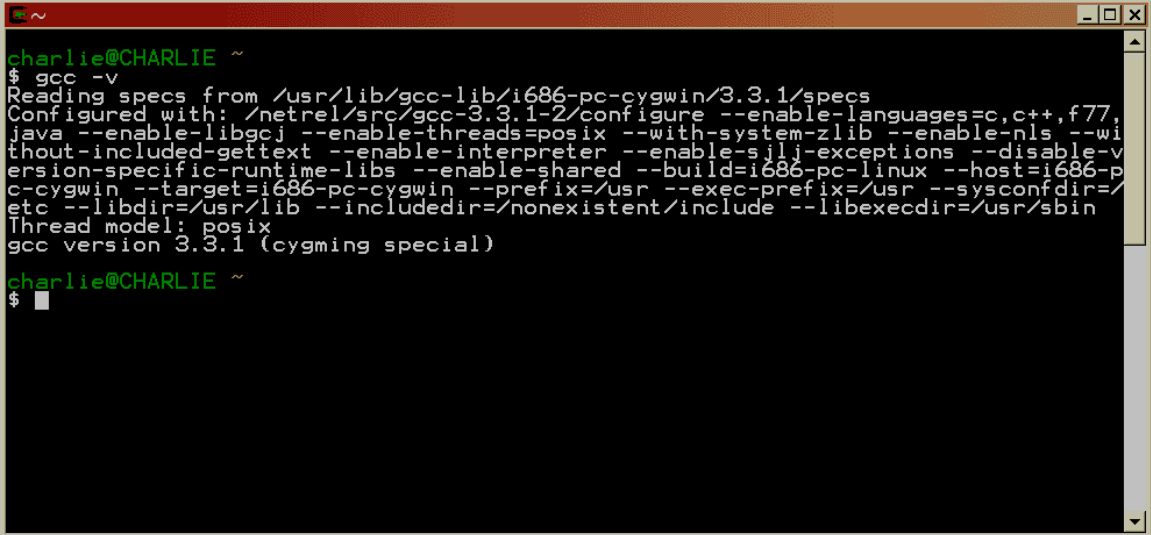


Figure 1, CYGWIN Setup. Click on “Default” to change it to “Install”

Once the CYGWIN base is installed and the development components are added, it should be possible to start a CYGWIN window on your computer. This will be an MS-DOS box with a command shell running a “bash” shell. Kernel calls map through to the CYGWIN DLL. Once the installation is complete, run a quick check of your work, as follows:

A terminal window with a black background and green text. The prompt is 'charlie@CHARLIE ~'. The user enters '\$ gcc -v'. The output shows the GCC version and configuration details for the i686-pc-cygwin target. The prompt returns to '\$' after the output.

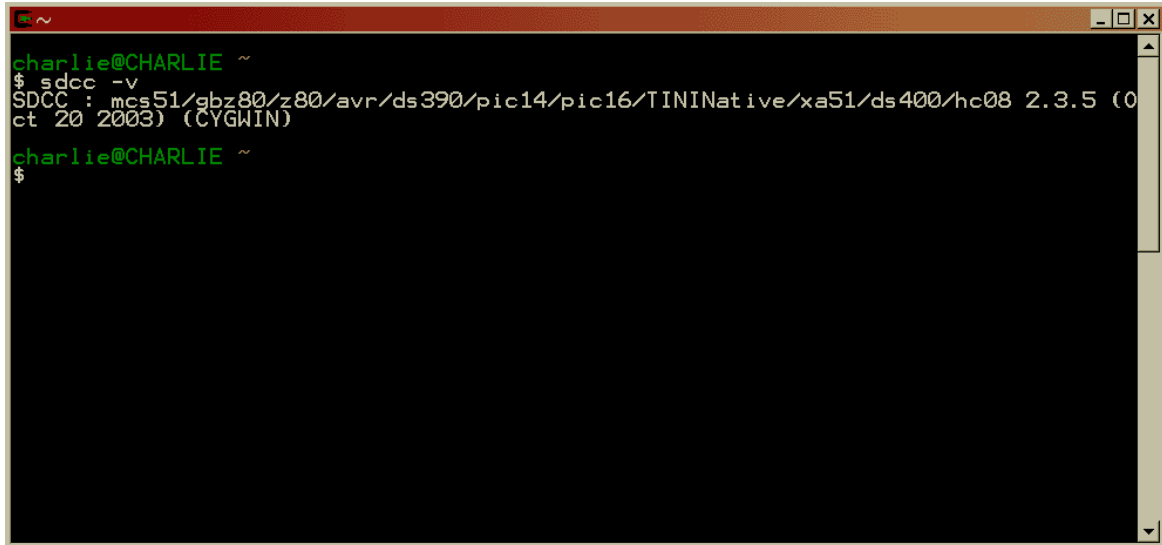
```
charlie@CHARLIE ~
$ gcc -v
Reading specs from /usr/lib/gcc-lib/i686-pc-cygwin/3.3.1/specs
Configured with: /netrel/src/gcc-3.3.1-2/configure --enable-languages=c,c++,f77,
java --enable-libgcj --enable-threads=posix --with-system-zlib --enable-nls --wi
thout-included-gettext --enable-interpreter --enable-sjlj-exceptions --disable-v
ersion-specific-runtime-libs --enable-shared --build=i686-pc-linux --host=i686-p
c-cygwin --target=i686-pc-cygwin --prefix=/usr --exec-prefix=/usr --sysconfdir=/
etc --libdir=/usr/lib --includedir=/nonexistent/include --libexecdir=/usr/sbin
Thread model: posix
gcc version 3.3.1 (cygming special)
charlie@CHARLIE ~
$
```

Figure 2, checking the CYGWIN/GCC installation

Note that we have not only checked the fact that the CYGWIN shell runs, but we have also checked our GCC installation by starting the compiler as “gcc -v”. We had no trouble installing “X-Window” and running it as well, for those who want to use one of the GUI based IDE’s.

SDCC

Even if you receive a version of SDCC with your CYGWIN release, you will still want to update SDCC. The version of SDCC we received with our CYGWIN did not work at all! As of this writing, the SDCC group is doing nightly builds, and we had no trouble with the latest stable release. Visit <http://sdcc.sourceforge.net/> and update your installation. Build SDCC from the source, using your GCC compiler. The steps to do this are clearly outlined in the SDCC documentation.



```

charlie@CHARLIE ~
$ sdcc -v
SDCC : mcs51/gbz80/z80/avr/ds390/pic14/pic16/TINative/xa51/ds400/hc08 2.3.5 (Oct 20 2003) (CYGWIN)
charlie@CHARLIE ~
$
    
```

Figure 3, checking the SDCC installation

Once done, check your work by exercising the compiler at the command line. Again, note that versions of SDCC prior to 2.3.5 will probably not work.

Source Code Preparation

SDCC accepts ANSI C, and includes a number of language extensions specific to the 8051. While the syntax of SDCC is slightly different than the one used by TI in its examples, it is a simple matter to automatically translate from TI "C" to SDCC "C."

C Source Code

In order to compile programs for the MSC12xxEVM, it is helpful to have "msc12xx.h" available for complete register definitions. Unfortunately, some TI files use slightly different language extensions for special register and special bit definitions. Some of these cannot be remedied with "#define" style macros. For example, a TI header file might read as:

MSC12xx.h

```

/*--
MSC12xx.H
Header file for TI MSC12xx microcontroller.
All rights reserved.
--*/
sfr P0 = 0x80;
sbit TF1 = TCON^7;
    
```

This is not immediately useful for SDCC both because the syntax of the "sfr" and "sbit" language extensions is ordered differently, and because the SDCC preprocessor does not accept bit level operators for "sbit." But since we are using Linux, we have many easy ways to fix this. Rather than rewrite

the compiler to accept TI's syntax, which might take some effort, we simply write a quick preprocessor to do the trick. We use Perl [5], because it is quick and easy:

sdcc_header.pl

```
#!/usr/bin/perl

# Perl script to translate TI 8051 Headers for use with SDCC

# Open the header and treat each line one at a time
open(FH, "msc12xx.h") || die "No File $!";
while(<FH>) {

    # if we find a "sfr" definition, save it for later
    if (/^sfr(16)?\s+(\S+)\s+=\s+(\S+);/) {
        print "sfr at $3 $2;\n";
        $sfr{$1} = $2;

    # if we have an "sbit" as a "sfr" definition, use our record
    } elsif (/^sbit\s+(\S+)\s+=\s+([\^\^]+)\.(\S+);/) {
        printf("sbit at 0x%X %s;\n", hex($sfr{$2})+hex($3), $1);

    # Any other lines are just echoed
    } else {
        print $_;
    }
}
close(FH)
```

This program is included, and as supplied simply dumps its output to the screen. Invoking it as "parse.pl >ti.h", though, would write a new file named "ti.h" with the proper definitions. The above header might translate as follows:

ti.h

```
/*--
MSC12xx.H
Header file for TI MSC12xx microcontroller.
All rights reserved.
--*/
sfr at 0x80 P0;
sbit at 0x88 TFL;
```

We copy this file to the SDCC include directory. Now we write up a quick "Hello World" program in "C":

HelloWorld.c

```

//
// Copyright 2002 Texas Instruments
//
// MSC12xx Hello World Program
#include <8052.h>
#include <ser.h>
#include <stdio.h>

extern void autobaud(void);

// We define a "putchar" for "printf" to use
void putchar(char ch) {
    ser_putc(ch);
}
// Begin the program..
void main(void) {

    // Press <Enter> for auto baudrate adjust
    autobaud();

    // SDCC requires these setup calls
    ser_init();
    EA=1;

    // Print to the terminal
    printf("Hello World\r\n");
}

```

Note that the “autobaud” that we use is from the SDCC library, and not the TI supplied ROM. No matter, it functions the same way, timing the signal wavelength on the input channel and setting the baud rate on the 8051 appropriately. Also note that we used the SDCC serial library, being careful to enable interrupts with “EA=1” before trying to write to the port.

We also make use of the standard “C” library, as implemented by SDCC. By defining a “putchar” which simply writes to our terminal, we may now use the library function “printf” (which resolves, after formatting, to our “putchar”) to write a message to the host screen.

Assembly Language

SDCC’s output is a well commented assembly language listing. This can be used for reference. Also, the ASXXX assembler can accept hand written code. The command “asx8051 -los myfile.asm” will assemble “myfile.asm” to “myfile.rel”. The output file may be linked along with any output of SDCC to form a program downloadable to the MSC12xxEVM.

Libraries

The entire source listing for the SDCC libraries is included with the distribution. The area of library transparency and adaptability is one where the “open source” movement really shows its advantages. Naturally, tools for creating one’s own libraries are also included.

It is considered good form to contribute any useful code you might produce back to the SDCC project.

Running the “Hello World” Program

All that remains is to run the program. As we mentioned, it is necessary to install the TI downloader. Instructions for doing this are included on the CD which was shipped with your MSC12xxEVM. It is a good idea to test the installation with a TI supplied example before approaching SDCC.

Makefile

There is one last foible of our tool chain we must address. A CYGWIN file uses a different “end of line” sequence than does a Microsoft Windows[™] file. This is such a common nuisance that a utility to remedy the issue is included with CYGWIN. It is the “unix2dos” command that is included in the “make file”. “Make” is a utility that has been around for years, and is a favored method for automating a tool chain’s flow:

Makefile

```
all:      HelloWorld.ihx

HelloWorld.ihx: HelloWorld.rel
    sdcc HelloWorld.rel
    unix2dos HelloWorld.ihx
    download /FHelloWorld.ihx /X11 /P1 /T /B19200

HelloWorld.rel: HelloWorld.c
    sdcc -c HelloWorld.c

clean:
    rm *.rel
```

Downloading and Running

Typing “make” automatically looks for a file in the current directory named “Makefile”, as shown above, and follows the rules it contains. This should result in the building and running of the “HelloWord” program we have been discussing in this application note.

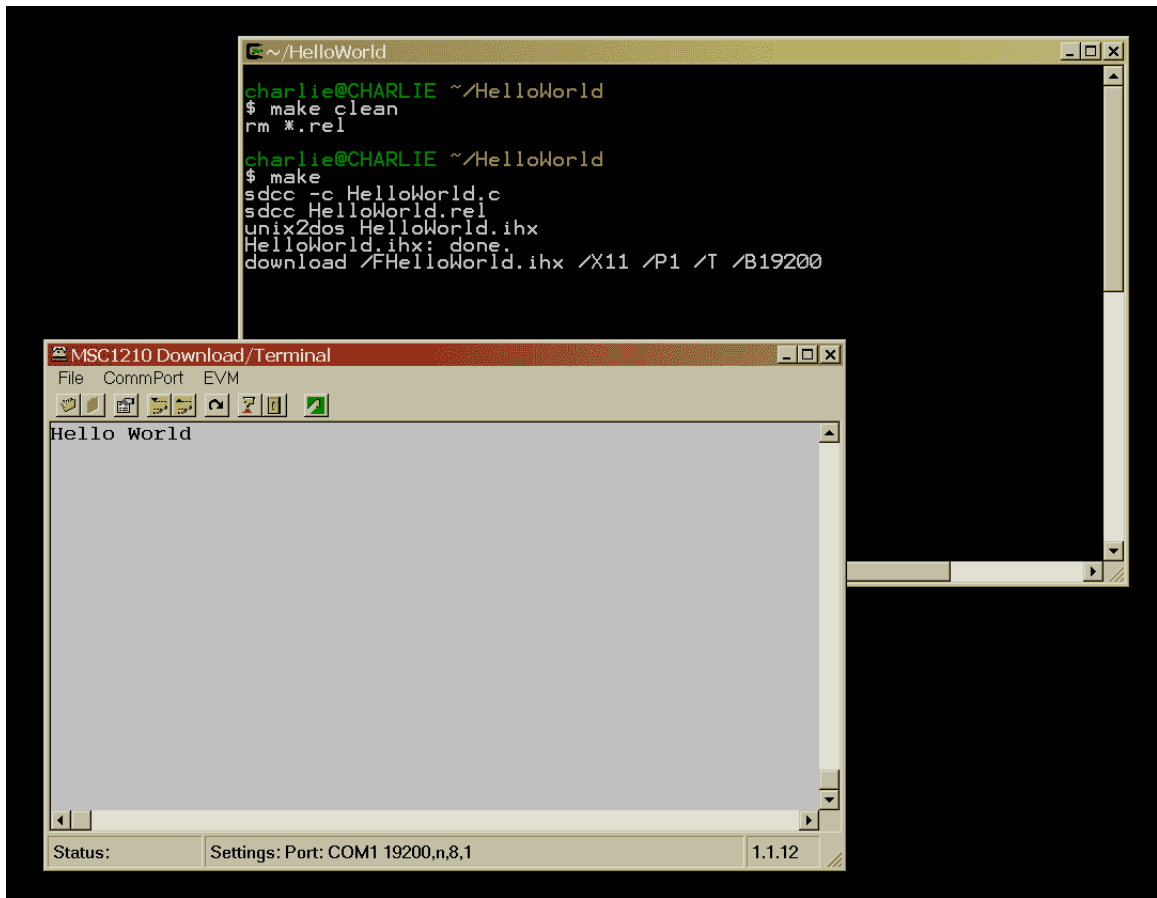


Figure 4, Running the Program

The figure above shows the screen after the TI downloader has run, and you have pressed “enter” in your keyboard. The “HelloWorld” shows its output in the terminal screen, demonstrating our success!

Exercising the Analog to Digital Converter

The Analog to Digital Converter (the “ADC”) is the heart of the msc12xx . Before we set up and use the ADC, though, we will want to be sure we can print the results. Because we want to easily be able to analyze our results without overflow or underflow problems [6], we need to do a little work first.

Using Floating Point in SDCC

SDCC includes floating point support by default. However, SDCC currently ships with the floating point support for the “printf” library function turned off. This is easy to fix. We go back to the place where we first downloaded and built SDCC and make a small change to the “vprintf” function. This function should be in “/usr/share/sdcc/device/lib/vprintf.c”.

vprintf.c

```

/*-----
vprintf.c - formatted output conversion
Written By - Martijn van Balen aed@iae.nl (1999)
Added %f By - johan.knol@iduna.nl (2000)

This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the
Free Software Foundation; either version 2, or (at your option) any
later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

In other words, you are welcome to use, share and improve this program.
You are forbidden to forbid anyone else to use, share and improve
what you give them. Help stamp out software-hoarding!
-----*/

/* this module uses some global variables instead function parameters, so: */

#ifdef SDCC_STACK_AUTO
#warning "this module cannot yet be use as a reentrant one"
#endif

// We comment out the switch, so "USE_FLOATS" is unconditionally true
// #if defined(__ds390)
#define USE_FLOATS 1
// #endif

```

Note at the end of the file where we have set "USE_FLOATS" to true. This applies only to the "printf" function; floating point match is enabled by default in SDCC. We then rebuild and reinstall SDCC as we did when we first installed it.

Using the ADC

Now we can write a "C" program which will read the ADC and output the results to the terminal:

adc.c

```

#include <ti.h>
#include <stdio.h>
#include <ser.h>

// We will use these to scale our ADC's
#define VREF 2.5
#define GND 0.0
#define FULL_SCALE 16777215.0

// function declarations
extern void autobaud ( void );
extern unsigned long unipolar ( void );
extern void calibrate ( void );

```

adc.c (continued)

```

// global variables for convenience
// note: SDCC places automatics on the heap,
// unless keyword "reentrant" is used
int i, n = 0, decimation = 1920, samples = 10;
unsigned long adc, adc_buffer;
float volts;

// Our entry point
void main ( void ) {

    // Set up the terminal
    autobaud();
    ser_init();
    EA=1;

    // Prepare for ADC use
    calibrate();

    // Go in circles...
    while (1) {

        // Take a few samples for noise free operation
        adc = 0;
        for ( i = 0 ; i < samples ; i++ ) {

            // wait for the next result
            while (!(AIE&0x20));

            // accumulate the results quickly, without overflow,
            // but with a bias against older samples.
            adc_buffer = unipolar() >> 1;
            adc = (adc >> 1) + adc_buffer;
        }

        // calculate the exact voltage we are seeing...
        volts = ((float) adc / FULL_SCALE) * (VREF - GND);

        // ...and print out the results
        printf("Sample %3d: V=%f      \r", ++n, volts);
    }

}

//
// Set up the ADC and throw away a few a samples as the device settles
//
void calibrate ( void ) {

    printf ("Unipolar mode (AIN0)\r\n");

    // Timer Setup
    USEC   = 10;                // 11 MHz Clock
    ACLK   = 8;                 // ACLK = 11,059,000 / 9 = 1.2288MHz
                                // modclock = 1.2288MHz / 64 = 19,200 Hz

    // Setup ADC
    PDCON &= 0xF7;             // turn on ADC
    ADMUX = 0x08;              // Select Analog Channel 1

    // VRefOn, VRef Hi, Burnout Detect Off, PGA = 1
    ADCON0 = 0x30;

    // unipolar auto, self calibration, offset, gain
    ADCON1 = 0x41;

```

```

// Note that if decimation is too low, noise will appear...
ADCON2 = decimation & 0xFF;           // LSB of decimation
ADCON3 = (decimation>>8) & 0x07;     // MSB of decimation

printf ( "Calibrating. . .\r\n");

for ( i = 0; i<4; i++ ) {

    // Wait for four conversions for filter to settle after calibration
    while (!(AIE & 0x20));

    // dummy read to clear ADCIRQ
    unipolar();
}

//
// This is for unsigned 24 bit results
//
unsigned long unipolar ( void )
{

    // Storage for in-place conversion
    union {
        char BitMap[4];
        long LongInteger; } DoubleWord ;

    // Move the register contents piecemeal to "little endian" "long" storage
    DoubleWord.BitMap[3] = 0x00;
    DoubleWord.BitMap[2] = ADRESH;
    DoubleWord.BitMap[1] = ADRESM;
    DoubleWord.BitMap[0] = ADRESL;

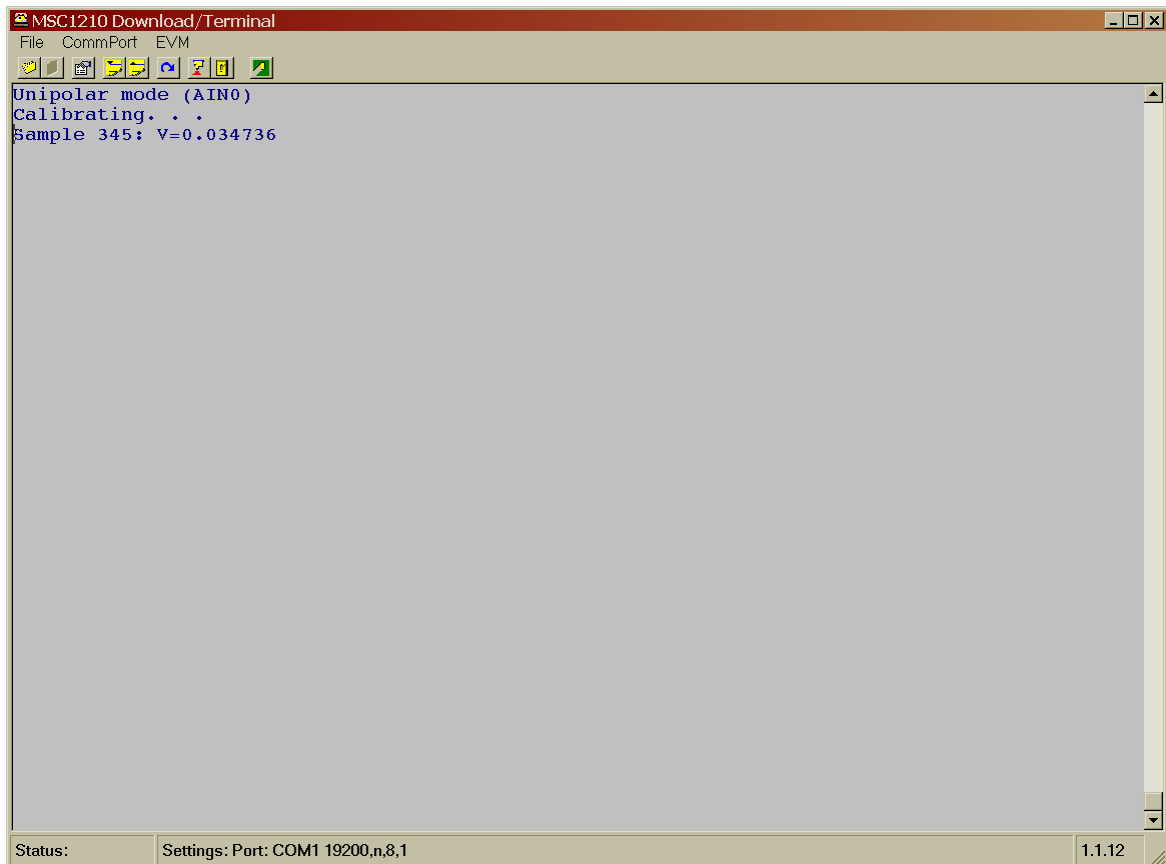
    // return the result as a "long"
    return DoubleWord.LongInteger;
}

//
// As in "HelloWorld", we need this for "printf"
//
void putchar(char ch) {
    ser_putc(ch);
}

```

Running the Program

With the MSC12xxEVM powered up, ground AGND, and place a voltage source across ANC and AIN0 of less than 2.5 Volts. Now, using a "Makefile" like the one we used for "Hello World", we build and run "ADC." The result should be a screen like the one we saw for "HelloWorld", but with a precise read on the voltage we place on the input pin of our msc12xxEVM showing on the terminal screen.



```

MSC1210 Download/Terminal
File  CommPort  EVM
Unipolar mode (AIN0)
Calibrating. . .
Sample 345: V=0.034736
Status: Settings: Port: COM1 19200,n,8,1 1.1.12
  
```

Figure 5, Running "adc.c"

Conclusion

SDCC is a viable option for software development on the TI MSC12xx family of Microsystems. The software is available free of charge, and is "open source." Extending the Libraries is always an option, and the Developer has the added security of knowing that software tool reliability is under his or her control.

End Notes

- [1] SDCC is currently maintained by the Source Forge at <http://sdcc.sourceforge.net/>.
- [2] <http://www-s.ti.com/sc/psheets/sbac018a/sbac018a.zip> is a location for obtaining the TI downloader.
- [3] The home page for Cygwin is <http://www.cygwin.com>.
- [4] Important information is available at <http://www.gnu.org>. Note that SDCC and CYGWIN are also released under the GPL.
- [5] Perl is a widely used for text processing, particularly for Internet applications. It is included in the CYGWIN distribution. More information is available at <http://www.perl.com>, as well as on many other Internet sites.
- [6] Many thanks to Russell Anderson at TI for suggesting this topic.