SiLabs IDE, SDCC, and driver Installation and Configuration

Introduction

The SiLabs IDE is a convenient way to edit, compile, and download source code written for the microcontroller. While SiLabs provides a nice interface for making source code changes and easily downloading them to the development boards, it lacks the actual compiler portion, which converts C code to hex files, the common format used by the 8051. To do this, a free and widely used open source tool called Small Device C Compiler (SDCC) is used. SDCC compiles the C code written, and automatically optimizes and converts it to hex. Due to its popularity, support for SDCC in the SiLabs IDE comes standard, making it easy and convenient to use.

Installing SDCC

To install the current supported released SDCC, follow the simple instructions below:

1. Open the following URL: http://sourceforge.net/projects/sdcc/files/. Click on **sdcc-win64** (middle of the page), select a version (3.5.0 was the latest on 6/2015), select **3.x.x** then **sdcc-3.x.x-x64-setup.exe** link for Windows to download it and click on it to install. Files are also on LMS: **Course Resources→Software & Drivers**. (Older laptops require **sdcc-win32** if not 64bit.)
2. The installation wizard that was just downloaded and executed will step you through the process of installing SDCC on your computer. It is recommended that you leave all of the configuration options and installation paths the same as suggested by the installation wizard (C:\Program Files (x86)\SDCC), so that it will be easier to assist you should problems arise.
3. To ensure SDCC was properly installed, click on the **Start Menu**, click on **All Programs**, and ensure that **SDCC** shows up on your programs list. If it does, you have successfully installed SDCC. 64bit laptops use C:\Program Files\SDCC and 32bit use C:\Program Files (x86)\SDCC.

Installing SiLabs (Silicon Laboratories) IDE

To install the SiLabs IDE, follow the simple instructions below:

1. Open the following URL: http://www.silabs.com/products/mcu/Pages/8-bit-microcontroller-software.aspx#ide. Click on **Download Software** under "Silicon Labs IDE". Version 4.90 was released on 7/18/2014. (Later you may want to get **Configuration Wizard**.)
2. Open the executable that was just downloaded and allow the installation wizard to step you through the installation process. Once again, it is recommended that you leave everything at the default settings. **If this is a second install – check the "Maintain or update the instance of this application …"**
3. To ensure the IDE was properly installed, click on the **Start Menu**, click on **All Programs**, and see that **Silicon Laboratories** shows up on your programs list. If it does, you have successfully installed the SiLabs IDE.

Configuring SiLabs IDE with SDCC

(***This section is the source of almost all installation problems. Follow the directions exactly.***)

To configure SDCC as the compiler in the SiLabs IDE, follow the instructions below:

1. SDCC now supports both 32-bit and 64-bit versions of the compiler. Depending on which one was installed you may or may not need the **(x86)** tag on the **Program Files** folder name used below. Check **C:\Program Files** and **C:\Program Files (x86)** to see where the **SDCC** folder is.
2. Click on the **Start Menu→All Programs→Silicon Laboratories**, and open the **Silicon Laboratories IDE**.
3. Once open, locate the **Projects** menu at the top of the program, and select **Tool Chain Integration** from the menu.
4. You will see a *Preset Name* dropdown box under **Tool Definition Presets**. Select *SDCC 3.x* from this menu (after the 1st try you will see another popup window, but select **No** in **Save Modified Presets)**. The values under **Tool Definition** should all change to the SDCC defaults, but some need to be further edited.
5. Verify that the paths for the Assembler, Compiler and Linker are correct. Click on the tab for each and check that the directory paths are consistent, as detailed in the following (do **not** close the window after verifying the paths):
   i) Assembler: The **Assembler** tab is selected by default. If necessary, click on **Browse** to tell SiLabs where SDCC is installed. If you kept all the default paths, as recommended, the path should be:
   C:\Program Files (x86)\SDCC\bin\sdas8051.exe
   ii) Compiler: After the assembler path is configured, check the compiler path. To do this, click on the **Compiler** tab. Once again, if necessary, click **Browse** and select the path to where SDCC is installed. If you kept all the default paths, the full path should be: *C:\Program Files (x86)\SDCC\bin\sdcc.exe*
   iii) Linker: Finally, to configure the linker, select the **Linker** tab. Click **Browse** and select the path where SDCC is installed if needed. The full path for a default installation should be: *C:\Program Files (x86)\SDCC\bin\sdcc.exe*
6. Set the command line flags. The compiler and linker flags must be set correctly. If you don't start projects correctly, these flags will be reset to the defaults (which are not correct). Not setting these correctly is the most frequent source of compiler errors. (do **not** close the window after setting the flags)
   i) Compiler. Click the **Compiler** tab. In the *command line flags* box, delete the section that says (the smart quotes " " are not part of the string) "-l"C:SiLabs/MCU/inc"". Add the flag, (again don't include the smart quotes) "--nogcse". The double dash is part of the string. Your command line flags should be:
   -c --debug --use-stdout -V --nogcse.
   ii) Linker: Click the **Linker** tab. Verify that the command line flags are (if not, correct them):
   --debug --use-stdout -V
7. Once you have made these changes, click on **Save As** under **Tool Definition Presets**, enter *LITEC* as the *Preset Name* and click **OK**. Now click **OK** in the Tool Chain Integration panel. It is very important that you do not change anything outside of what has been mentioned above, especially the command line flags passed to SDCC. Unfortunately, whenever a new project is created the program defaults to one of the other presets (see below), you must select *LITEC* as the *Preset Name* and select **No** for **Save Modified Presets**. This seems to be a bug in the current IDE. Double-check all settings above (they may reset to their defaults if changes are made in the wrong order) & exit.
8. You have successfully set up SiLabs to use SDCC!

Header files

1. The projects require a header file that initializes the microcontroller. The file name is c8051_SDCC.h. It is available on the course RPILMS site.
2. Go to the course site on RPILMS and save the header file, **c8051_SDCC.h**, to your computer in the following folder: **C:\Program Files (x86)\SDCC\include\mcs51**. You may need to use a different browser other than IE, such as Safari or FireFox and you may need to perform the move in 2 steps: first save it to the desktop, and then move it to the folder.
3. Later in the course you will need to add a second header file, **i2c.h**, to this same location, but you will be told when it will be required. The include statement for i2c.h must be the last one in your program.

Creating a Project

NOTE: project files have the extension .wsp. Clicking on this file will start up the SiLabs IDE. It is highly recommended that all new project folders be place in the folder **C:\SiLabs** for consistency.

1. To create a new project, click on **Project→New Project…**, select C8051F02x under **Select Device Family:**, enter a name under **Project name:**, leave **Project Type:** as *Blank Project*, and browse to a desired **Location:**. This will create an empty project.
2. Move a saved C file (ex. LITEC-HW1.c) to the folder selected as the project location in step 1. REMEMBER: C file names must not contain the '#' character, only alphanumerics, '_', and '-' (and a single '.' before the extension). A new folder for each new project is recommended.
3. Add this new C file to the project by clicking on the **Project→Add Files to Project**.
4. Right click on the filename under the project folder on the left side of the screen and select **Add *filename.c* to build** (ex. Add LITEC-HW1.c to build). Your code will not build if you forget to perform this step, another frequent mistake.
5. Double click *filename.c* to open it for editing in the IDE window.
6. The file will now be part of the project, and can be compiled. It is suggested that you delete the Source Files and Header Files folders by right clicking on them and selecting **Remove group Source/Header Files from project** to minimize the confusion.
7. Unfortunately, you will need to specify *LITEC* in **Tool Definition Presets** (in **Tool Chain Integration**) for every new project; the defaults can't seem to be changed on installations.
8. The file will now be part of the project, and can be compiled. Save the project by clicking on: **Project→Save Project**
9. Also, for every new project, you will need to go to **Project→Target Build Configuration…** , and in the **Absolute OMF file name:** field add *.omf* at the end of file name (no spaces). If you don't do this you will receive an error: "Cannot find file C:/..." when attempting to download.

Compiling a Project

1. To compile your project, first verify that all of your code is correct. If it's not, don't worry; the compiler will catch your mistakes. There are 5 icons that simplify the compile, build, and download process. Below the menu bar, they start with the 8th icon from the left with the name "Assemble/Compile current file (Ctrl+F7)" when the mouse is over it.
2. Select the **Assemble/Compile current file** icon. The window at the bottom of the IDE will alert you of any errors or warnings it finds in your code. If there are errors, correct them first

before moving on to the next step. If there are warnings, you may or may not want to correct them, depending on the nature of the warning.

3. Select the next icon to the right, the **Build/Make project** icon.
4. After successfully building your project, you will want to download the executable code to the development board. In order to perform the next steps (4-7) you need to plug the USB cord on the car connected to the Debug Adapter toward the middle-back of the car into your laptop (not the serial USB cord on the side of the car). Before doing this, you will need to configure the adapter used to download the executable code. To do this, click on the **Options** menu and select **Connection Options…**. Once the window opens, select the *USB Debug Adapter*, make sure that *JTAG* is selected under Debug Interface (not *C2*), and click **OK**. Be sure the *RS232 Serial Adapter* option is **not** selected.
5. You are now ready to download the code. To do this, click on the **Debug** menu, and select **Connect** (or select the **Connect** icon)**.** This will connect the IDE to the development board through the USB Debug Adapter.
6. Select the **Download code** icon (to the right of the **Connect** icon). This will download the code you just compiled and built to the development board. If you forgot to add the ".omf" in step 9 under **Creating a Project** you will get an error!
7. Finally, you are ready to execute your code. But before you do you need to make sure you have **SecureCRT** installed so that any output from your program may be observed. For this you need to complete the steps below under **Terminal Emulator Program**. With that installed and the USB/serial adapter wire from the side of the car plugged into your laptop you may return to the **IDE** and execute your code (run it), by clicking on the green **Go** icon.

Note: to reset a program after stopping it, go to **Debug→Reset** (or **Ctrl+R**)-faster than a 2nd download.

Optimization bug (repeated from Configuring SiLabs IDE with SDCC, step 5)

This will be discussed in lecture 3. The compiler sometimes mistakenly concludes that a while loop will never end and code after the loop can't be reached. In this case is doesn't compile the remaining code. A warning is generated that a branch statement was modified.

To avoid this, make sure the "--nogcse" option was added to the compiler in **Configuring SiLabs IDE with SDCC** above and -l"C:SiLabs/MCU/Inc" was removed.

Terminal Emulator Program

SecureCRT is the recommended terminal emulator program for use in this course. It is part of the student build. Other programs such as HyperTerminal and Putty are acceptable, but don't expect technical assistance from the staff if you use something other than SecureCRT. The older 5.5 version is strongly suggested and is available on LMS under **Course Resources→Software & Drivers**. (Any newer version should be avoided as unnecessary.)

The following instructions are for setting up version, (similar setting for all versions).

1. Start SecureCRT.
2. Create a new session with **File > Quick Connect…** or **(Alt+Q)**
3. Change the **Protocol:** to *serial*,
4. Change the **Port:** number to the correct COM port number from Device Manager above

5. Set **Baud rate**: to *38400*, **Data bits:** to *8*, **Parity**: to *None*, **Stop bits**: to *1*. These *should* be the default settings, but you never know.
6. **IMPORTANT**: Uncheck **all** (e.g. **RTS/CTS**) boxes under **Flow Control** and click **Connect**

The following steps 9. - 11. are only needed if you want to save a lot of previous screen lines.

9. Click **Options → Session Options…** and select "Emulation" under "Terminal".
10. Under Scrollback, change the value in **Scrollback buffer**: to 3000.
11. Click **OK**.

For HyperTerminal, ProComm plus, or Putty, menu items will be different but the important settings for all direct serial communication modes are: baud rate = 38400, data bits = 8, parity = none, stop bit = 1, flow control = none, and the COM port is whatever number your laptop has assigned to the USB/serial bridge (as seen in Windows Device Manager under Ports (COM & LPT).

Later in the semester you will be using SecureCRT to collect data from the blimp for plotting. In order for all the data printed on the screen to be collected into a file, you must invoke another feature. After making the connection to the C8051 from SecureCRT, select **File → Log Session** and change the name and location of **File name:** from *session.log* if you desire and click **Save**. When you quit SecureCRT after running your program, the file will be saved with all the data written to the terminal. The maximum number of lines saved in the file is changed by going to **Options → Session Options…** and selecting **Emulation** under **Terminal** on the left side. Then change the value in the box labelled **Scrollback buffer** to what you need it to be. You first must be connected to a comm port to do this.

The remaining parts of these installation instructions are not used until midway though the course and should be skipped for now.

Drivers for USB/serial adapter (if it doesn't install automatically when USB/serial is plugged in)

A driver is required for the USB-to-serial connector used in this course. If it doesn't auto-install, then:

1. http://www.prolific.com.tw/US/supportDownload.aspx?FileType=56&FileID=133&pcid=85&Page=0 download the PL2303_Prolific_DriverInstaller_v1.7.0.zip file, unzip and run the installer. This driver works for XP, VISTA, and Windows 7. Use "GUEST" for the Account and Password, if required. The zip file of the installer can also be found on the course LMS page:
   Course Resources→Software & Drivers→PL2303_Prolific_DriverInstaller_v1.zip
2. When you plug in the USB/serial adapter, your machine should find the hardware and it will create a serial port. After connecting to the adapter, do the following (this only works if you have the USB/serial adapter cable plugged into one of your USB slots):
   a. Right click on **Start → Computer**
   b. Click on "Properties"
      i. Select the Hardware tab
      ii. Click on Device Manager
      iii. Expand Ports (COM & LPT)
      iv. A port called "Prolific USB-to-Serial Comm Port" should exist
      v. Note the port number – such as COM4 or COM6. You need to know this port number for the terminal emulator program, (see below)
   c. Close the windows
3. A special note for Windows 8 and up users: you **must use the black one-piece USB/serial adapter cable**. The two-piece blue adapters are incompatible with Windows 8.

Drivers for USB RF link serial adapter (wireless serial connection)

Another driver is required for the wireless serial connector used with the blimps and turntables to allow the remote C8051 to communicate with your laptops running SecureCRT.

1. The RF link connects directly to a USB port of your laptop. It requires a driver available at http://www.ftdichip.com/Drivers/VCP.htm. If it exists, download and run the "installation executable" which links to: http://www.ftdichip.com/Drivers/CDM/Win2000/CDM_Setup.exe Otherwise, it that doesn't work, download the zip file (for your OS under the column heading **Driver Version**) and the instructions under **Installation Guide**. If automatic installation is enabled Windows 7 may be able to find, download, and install the driver by itself.

2. The RF link runs at a max 38400 baud rate, which is the same as what we have been using.

3. The link sends data in packets. It ignores any new communication while it is sending a packet. So printf statements must:
   i. Be shorter than 92 bytes
   ii. Be spaced out in time. Only printing during each new range reading should work.

4. The RF links use channel numbers, so don't try to move them from gondola to gondola. It won't work.

Common Problems

Hardware

1. Make sure all ribbon cables are securely attached to the protoboard, C8051 board, and buffer board. Make sure +5V and ground have been properly routed to chips and modules.

2. Check to be sure the power switch has been turned on.

3. Buffer chips (74365) need power and ground in addition to pins 1 & 15 being grounded to enable the chip's output.

4. Connections to the C8051 bus must be checked by both partners to insure that the correct pin numbers have been wired. Particularly troublesome are the connections to I2C SCL & SDA lines or the CEX0 to CEX4 lines whose position and pin numbers are directly dependent on the Crossbar 0 settings. These move around from example to example and exercise to exercise.

5. Using the wrong SYSCLK input to the PCA for PWM control of the motors will NOT work. In some cases the steering servomotor may react by turning to one limit but the drive motor will appear to be completely dead. SYSCLK/12 must be used and the PCA period must be 20ms.

6. On the car and gondola, if the steering control does not seem to be working, make sure the compass is calibrated and check it readings as the car or gondola are rotated a full 360° turn. Make sure the value read is increasing steadily for a 0° to 360° rotation.

7. When wiring up the I2C devices, all the SDA pins must be tied together and all the SCL pins must be tied together. Then a single pull-up resistor connects all the SDA pins to +5V and a single pull-up resistor connects the SCL pins to +5V.

SDCC and C8051F020 Quirks & Problems

1. #define values shouldn't end with a ";" Doing so will create problems in some C statements. For example:

```
int test;
#define VALUE = 1234;      //This should be: #define VALUE = 1234
…
test = 4321;
if (test < VALUE)...       //This will not work, it is equivalent to "if (test < 1234;)…"
//Also, #define variables may not be used to set the values of s-bits.
```

2. To create an sbit or sfr with a specific name, SDCC uses the format:

```
__sbit __at 0xHH bit_name;    // There should not be a space between the 2 '_' characters.
                              // Some fonts may connect them together even though there
                              // are 2 of them 0xHH is the hex value from the sbit table in
                              // the SFR & SBIT DEFINITIONS table starting on page 142,
                              // bit_name is the label you choose for your variable
__sfr __at 0xHH sfr_name;     // NOTE: you still must be on the correct SFR page!
```

   Again, sbit values are dependent on the SFR page; make sure you are on the correct page. The latest SDCC compiler will complain about missing '__' before the 'sbit', 'sfr' and 'at', if you leave them off. Some of the header files may not use them and can result in a long list of warnings. It is strongly recommended that they are always used to keep the compiler output messages free of clutter and help you spot actual errors immediately.

3. To print floating point numbers, use:

```
printf_fast_f(" %f10.5", 3.14159);  //Don't use printf(), it will not work!
```

   To print long integers use:

```
printf(" %ld", long_integer); //Don't use %d or %i, they will not work!
```

4. Type conversion, especially implicit, may not work correctly. To extract numerical values of characters use atoi( ) & itoa( ) in stdlib.h.

5. To define data in external RAM at a given address use:

```
xdata at 0x8EED int my_variable;          // Will appear at address 0x8EED
```

   The C8051F020 only allocates 128 bytes for program variable space. Creating too many variables will result in an error concerning limited memory. Moving some variables to external RAM can fix this error. If you don't care about the address, just use `xdata int my_variable;`

6. Using too many variables, too large arrays, or too many large numerical types (long int, float, double) may result in a linker error "Could not get nn consecutive bytes in internal RAM for area DSEG". To correct this the number and/or size of the variables must be reduced. Reuse variables in multiple calculations, reduce integers to short int or char if the maximum numerical value will fit in the smaller type and reduce the size of arrays if fewer elements will work. Also consider moving data into external RAM (as in 5. above).

7. There is no implementation of the function scanf( ) in SDCC and there is no simple substitution. Any input must be done character by character using getchar( ) and extra processing by the program. Similarly, putchar( ) and getchar( ) are not directly available under SDCC. As mentioned in the first lab exercise, include c8051_SDCC.h in all programs to retain these functions.

8. A program that resets and restarts randomly, resets shortly after the motors are turned on, or seems to lose the value of variables may be a result of a low battery voltage, both on the gondola and car (if the charger has been unplugged). Make sure the battery charger cable is attached if you are working at a stationary point. On the gondolas a fresh battery must be installed. On the car in particular, driving the servomotor past the point the steering linkages allow movement, draws a very high current through the stalled motor and usually resets the program due to the large battery voltage drop.

9. Unreachable code warnings may show up as a result of temporarily disabling sections of code in the process of debugging programs. These may be ignored. If you get a strange quote about "so says Evelyn, the modified dog" in your warning, it means you haven't completed step 5 on page 2 properly. It could also mean you have a conditional value that can't change, such as `if (var = 0)` or `while (var = 0)` when you meant to use "`==`".

10. Sbit variables can't be initialized when they are defined. This will generate a lot of random assembler errors. Set them equal to 0 or 1 on a separate line inside main(). Likewise arrays that are attempted to be initialized by a statement such as "`char Data[2] = 0;`" will generate unrelated error messages complaining about structures that don't exist and the flagged lines may move about randomly. Use "`char Data[2] = {0, 0};`". On occasion initializing declared variables in xdata has been troublesome. Be aware that these might not be initialing properly.

11. Bit-wise logical operations on sbits such as: `if (PB0 & PB1) LED0 = 0;`
    results in a fatal compiler error (error 9: FATAL Compiler Internal Error in file) that does not indicate the offending statement correctly. It may be very difficult to track down the problem. Make sure byte-wise operators are used (`PB0 && PB1`).

12. All local variables must be declared at the beginning of the routine (main or function). Implicit declaration at the time of use is not allowed

13. File names cannot include the '#' or ',' characters and possibly other special characters. For some reason it tries to use the assembler on the C source code and generates many errors.

14. As with most C compilers, the order of the included header files at the top of a program is important. Basic headers defining objects used in other headers MUST be included first. Most user created headers should be included last.

15. Starting with version 4.10 of the IDE and 2.9.0 of SDCC there are a number of new issues. If multiple .c & .h files are used in a project and added to the build, it is possible to confuse the compiler and linker so that it doesn't know which file contains `main( )`, and it may actually try to run some other function as main, yielding very unexpected results. When adding files to the build, make sure the icon changes to "the-file-to-be-built" icon for the file containing `main( )`.

16. Programs that crash and stop responding 10 to 20 seconds after starting are usually reading or writing to the SMB (I2C) bus too frequently. Check all calls to the I2C functions and make sure none are happening more frequently than every 20 ms. Remember the keypad/LCD is another I2C device in addition to the compass and ultrasonic ranger. None of these routines can be called repeatedly inside the "`while (1)`" loop. For practical purposes the LCD should not be written to more than 100 ms or longer.

17. Sound travels at about 1 foot/ms. Rangers that only seem to work at close distances are usually having a problem with the software not pinging the ranger immediately after reading the last value. If the software waits 20ms before pinging, that could result in any distance larger than 10 feet (round trip travel of 20 feet) being measured incorrectly or returning a value close to 0xFFFF.

18. In order for both the steering and altitude control loops to function correctly they depend on regular and predictable timing periods on the sampled data values of compass heading and altitude. Any delays inserted in the critical timing loops will cause stability problems or erratic behavior.

19. Control calculations must be done with signed integers for the values that can be either positive or negative (error, previous error) and long types to be sure no overflows occur during calculations.

20. The LCD panel has room for only 80 characters. If you send more than 80 characters in a single lcd_print( ) function you will overwrite the buffer and unintentionally (and unknowingly) wipe out other variables in your program.

21. When using the wireless RF links, pay attention to item 5. on page 6 above, under the USB RF link description. Rapidly sending lines of text, even if they are less than 80 characters, will lock up the transmission link and print nothing on the SecureCRT terminal.

22. F0 and F1 are defined sbit names. The compiler will not flag it when you meant to use 0xF0 or 0xF1.