CHALMERS
Dept. of Computer Science and Engineering

2005-08-31

Introduction to Microcomputer Systems

Creating your application in XCC12

Overview

XCC12 is a development suite where you can develop the software for your HCS12 processor in assembler and/or C. You can compile and link the result, simulate it and download the code to the target processor for debugging and execution. All of this is done within one single framework.

A design can consist of several files that can be a mixture of assembler and C files and you hold them together as a project. Every project will have a separate folder on the hard drive in your computer but all the files in the project does not necessarily have to reside in that folder so you can have files that are shared between several projects. The project folder, with its sub structure, will be created when you create your project.

The basic framework in XCC12 is a so called workspace where you can gather a collection of related projects, for example all designs (projects) in this laboration course. The projects does not have to reside in the same base folder.

To create a workspace

The projects you develop have to reside within a workspace. You create a workspace by choosing **File/New Workspace** within the XCC12 framework and use the navigation window to go to the folder where you want your workspace to reside. You give a name to the workspace and a workspace file with a `.w12`-extension will be created. The new workspace will be visable in the file tree to the left within your XCC12 framework.

After this you are ready to continue and create your project. If you like you can insert an already creted project into the new workspace instead so you can customize your project structure the way you like.

To create a project

You create the new project by choosing **Project/Insert New Project into Workspace** and give the project a name in the file navigator. You will get a couple of dialog boxes where you can customize your project but for now we can leave it the way it is and just continue through the dialogs without changing anything.

You can place the project wherever you want but in most cases it is most convenient to keep it under the base folder where you created your workspace. When you create the project a new folder with some subfolders will be created for your project.
The new project will be active and visible in the file tree to the left within your XCC12 framework.
Your new project will contain a startup file (`_startup.s12`), a assembler file that will be linked at the top of your application to configure the processor. In our case we will have to make a small change in the startup file.

Customizing the startup file

In your applications you will use Port S on the processor to communicate with the bargraph on your laboration card. In the startup file this port is configured to use serial port 0 through this port. The serial port will not be used in your applications but it will stop you from using all the bits in the port for communication with the bargraph so you have to deactivate the serial port. You do this by opening the file `_startup.s12` by double clicking on it in the file tree. In the editor that shows you move down the file until you find the line

```
bsr  sciinit
```

which is a branch to the subroutine that activates the serial port. Deactivate this line by changing it into a comment. You do this by putting a star (`*`) at the beginning of the line.

```
*  bsr  sciinit
```

Save the file. This file is placed in the XCC program folder and not in your project so the change you make here will affect all the projects we will create in the laboration course, that is you only have to do the editing once.
Now you are ready to develop your application files.

Creating application files

Your application files can be assembler files, C files or header files (h files). You create a new file by choosing **File/New** and giving the file an appropriate name. Use the correct extention, `.s12` for assembler file, `.c` for C files and `.h` for header files. A text editor will show and you can start writing your code. We will get back to the file layout in the different cases in a short while.
When you have created your application file you have to include it into the project. It is not enough to save the file in the project folder. You include the file by selecting **Project/Insert Files into Project** and select the relevant file.

Supporting header files

The processor HCS12 have a lot of configuration registers and they are all memory mapped which means that they are placed at addresses within the processors memory space. It is not that easy to remember all this addresses so it is practical to use a header file where the registers have been given appropriate symbolic names instead of the numeric addresses. You can download such a header file from the homepage of the course (**Laborations/h-file with symbolic names for the different configuration registers**).The file is not complete but contains all the registers you will be needing. The file is called `ITuniv.h`.
When you address a register that has a fix memory address from C code you will have to use a pointer of type

```
*((unsigned char *)(register_address))
```

to address a 8-bit register. For a 16-bit register you need a similar pointer with an argument of type unsigned short. It is a little unpractical to write this code for every addressing of a register so the pointers are wrapped in macros to make it easier. These macros are placed in a file that also can be downloaded from the homepage (**Laborations/h-file with macros to address the configuration registers**). The file is called `regmacro.h`.
There are three different macros (REG8, REG16 and REG32) that can be used to address 8-, 16- and 32 bit registers respectively. Instead of writing the pointer expression above you use the macros in the following way. If you want to read a register you write (for a 8-bit register)

```
X=REG8(register_address);
```

where `X` is the variable where you want to place the result.
If you want to write to the register instead you write (for a 16-bit register)

```
REG16(register_address)=X;
```

where once again `X` is a variable you want to write to the register.
Now you can combine the two header files. The header file with register addresses contain for example the symbolic name `PTS` for the address 0x0248, which is the address to Port S, so to write to this port you can write

```
REG8(PTS)=X;
```

When you have downloaded these header files it is practical to save them in the top folder of your workspace so that you can easily access the files from all your projects. The files will have to be included into one of the program files in your project. The syntax will differ slightly if you create your application as a assembler file or a C file (our simple applications will only contain one file, not counting the startup file and the header files).

The structure of an assembler file

The assembler file is a ordinary text file but it will be to your advantage if you save it with a `.s12` extention and not a `.txt` extension.
The file have three separate columns. You use tabs to separate the columns. The first column is for labels, next is a column for the assembler code with its argument(s) and lastly a column for comments. Start a comment in the third column with a semicolon (`;`).
As we said earlier the linking of the application will start with the startup file. When this file is executed it calls the main function of your project which means if you want to use a assembler file you will have to have a main label at the entry point of the code. This label will have to have the name `_main`. The reason for this is that labels (function names) from C code gets a underscore (_) at the beginning when they are converted to assembler code and in this case we already have assembler code so we have to place the underscore there our selfs. To make the label `_main` visable in the rest of the project you have to include the line

```
DEFINE _main
```

in your assembler file.

If you want to use the header file with all the symbolic register names you will have to include it in your assembler file, preferable at the top of the file. You do this by the command

```
USE ..\ITuniv.h
```

This example assumes that the header file is placed in the folder where you created your workspace, that is one level up from your project folder.
The regmacro file is not used in the assembler file.

The structure of a C file

The C structure as such assumes that your file contains a main function so here you don´t need to do anything more than to make sure that your C code starts in the main function.
To use the header files you have to include them in the C file. The syntax for this is

```
#include "..\ITuniv.h"
#include "..\regmacro.h"
```

Once again we assume that the files are placed in your workspace folder.

Compiling and linking your application

When you have created your application files you compile and link the application by choosing **Build/Build All**. The process will probably generate some errors the first time around. You will get error messages in the window at the bottom of the framework. If you double click on a line with an error message the offending line in the code will be high lighted. Correct the code and build the application again.

Simulating the application

In the limited time we have for this course we will not be able to get into simulation of the application. If you like you can read about simulation in the help system within the framework (**Help/Help on XCC**).
We will go directly to downloading the application to the target processor.

Downloading the application to a target processor

To download the application to your target system you need to connect the target system to your PC. You do this through a cable connected to the serial port on your PC. This serial cable should be connected to a so called BDM-loader (BDM = Background Debug Mode) at the other end. This takes care of the transfer of code from the PC the code to the target processor. To do this the BDM-loader have to be connected to the target system. We will go into more detail on how to connect the parts when we get into the laboratory.
To begin the downloading process you start by choosing **Build/Debug Options** and choose **POD10 – BDM Adapter** in the dialog window.
After this you choose **Build/Start Debug/Source Level/Start at "main"** and the code will be downloaded and the startup file will be executed. If the application is a assembler file the application will start running. If the source file is a C file the execution will halt at the `_main` label and wait for your command.
We will talk more about debugging and running our applications when we are into the laboratory.