

## 1 Introduction

This laboratory will guide you through the software development process that will be used in this course. This process uses the CodeWright IDE, Metrowerks CodeWarrior (command-line only), and the Noral BDM Flex Debugger. CodeWright is used as the IDE to organize projects, edit files, and run GNU make. CodeWarrior, which is executed through the CodeWright Build button, compiles, assembles, and links the code. In addition it generates the IEEE DWARF2 file to be loaded into the debugger. The Noral debugger will be used to download the programs, program the flash, and debug your programs. This development process allows for modular programming with C source and/or assembly source files.

In this lab, you will not write any code. You will create a project directory, collect the required files from the WWU course directory, edit the files, build the project, and test the resulting code on your development board. The text material covering this process is Chapter 12, Modular and C Code Construction.

### Reference Material:

'Embedded Microcontrollers', Chapter 12  
CodeWarrior 68HC12 Manuals  
Noral Flex Debugger Manual

### 1.1 First Time CodeWright and Noral Setup

In order to use the CodeWright IDE and the Noral debugger in the ET340 lab, you will need to perform some setup procedures. These procedures will allow you to customize these applications to your work environment. Follow these procedures carefully!

**Noral Setup.** Before you try to start the Noral debugger, you must add a registry key. To do this, go to *Y:\Noral* and double-click on the file *Noralreg.reg*. It should respond with something like: 'Your registry has been updated'. Once you do this, it is safe to start the debugger from the start menu.

If the Noral debugger still does not work, it is most likely because a previous Noral key already existed and needs to be removed. To do this go to *Start->Run...* and enter *regedit*. Go to *HKEY\_CURRENT\_USER -> Software*. Select *Noral* and select *Edit->Delete*.

Be VERY careful when editing the registry. You can really mess up your account settings.

**CodeWright Setup.** Before you start using CodeWright, you must perform the following steps. This will allow you to customize CodeWright for your projects and will set it up for building HC12 projects.

1. Create a subdirectory in your home directory called *CW*.
2. Copy the files (except *RegBugFix*) from *Y:\cwnet\* to your *CW* directory.

From this point on, you should use the CodeWright shortcut on the desktop to start CodeWright. As long as you use the shortcut, CodeWright will startup in the same state as when you last closed it.

### 1.2 Creating a CodeWright Project Space

CodeWright uses the concepts of 'Project Spaces' and 'Projects' to help organize and automate code development. It is important that you understand these concepts.

**Project Space.** Before a CodeWright project can be created, you must create a project space. The project space defines CodeWright setups that contain editor options and build options for a type of code development. For example, you may have a project space for 9S12 development and another for HTML development. Each of these spaces will have different settings that are specific to the type of development you are doing. This is one of the concepts that make CodeWright a very powerful development tool. You can create project spaces for different compilers, different languages, or different target processors. In this class – and the senior project, you will only need one project space for 9S12 development.

#### Procedure for creating a CodeWright Project Space.

- 1) On the bottom-left side of the CodeWright window, select *Open*, and browse to your new *CW* directory.
- 2) Select *New...* under *Project* → *Project Space*.
- 3) Enter a project space name that makes sense such as *9S12Dev* a for 9S12 development.

This will create a file in your *CW* directory called *9S12Dev.psp*, which will contain all of your project information.

### 1.3 Creating a Project Directory

When developing modular programs your complete 'project' consists of many files. Because of this, you will need to create a separate project directory for each project or lab. It is also convenient to use the CodeWright project capabilities to help organize a project or projects, quickly jump between different projects, and automatically set build configurations when creating a new project.

**Project Sources.** The *emicros* directory on the server contains the source files for this lab. In this lab, you will copy all of the required files for the lab into a new project directory. These files include source files, a *Makefile*, a linker command file, an environment file, and a startup file. There will be minimal editing required to build your project.

#### Procedure for Creating your Project Directory.

- 1) Using windows explorer, create a new directory for this lab – most likely in an *ETec454* directory.
- 2) Using windows explorer, copy the following files into the new project directory:  
*Y:\emicros\ProjS12\Example Projects\Demo1 Source\pulse2.c*  
*Y:\emicros\ProjS12\Example Projects\Demo1 Source\STAR12\includes.h*  
*Y:\emicros\ProjS12\Example Projects\Demo1 Source\STAR12\msdelay.a12*  
*Y:\emicros\ProjS12\Makefiles\Makefile*  
*Y:\emicros\ProjS12\Linker Files\9S12DP256sm.prm*  
*Y:\emicros\ProjS12\Startup Files\9S12DP256Start.c*  
*Y:\emicros\ProjS12\Vector Files\s12dp256vecs.s12*  
*Y:\emicros\ProjS12\ProjEnv Files\project.ini*
- 3) Rename the startup file in your project directory to ' *ProjStart.c*' where *Proj* is the name of your project.

**CodeWright Project.** Now, to have CodeWright set up a project environment for your project, you will have to create a new CodeWright project. Make sure that the project space you created above is open (Checked under '*Project*' → '*Project Space*').

#### Procedure for Creating a CodeWright Project

- 1) Make sure you are in your new project directory.
- 2) Select '*Add New Project...*' under '*Project*' → '*Project Space*'. Name the project something like '*lab1*'.
- 3) Add all of the files you copied into your directory in the '*Members*' tab of the project properties dialog.

This should create a new file in you project directory with a *pjt* suffix. It contains all of the settings and state information that are specific to that project.

### 1.4 Edit the Makefile

The Makefile controls the build process by executing CodeWarrior command lines. You will need to enter the C source files (CSRCS), the assembly source files (SRCS), and the desired linker command file (LD), as described in the Makefile comments. For this lab, you should include the following:

```
CSRCS = pulse2.c Lab1Start.c
SRCS = msdelay.a12 s12dp256vecs.a12

PRM = 9S12DP256sm.prm
```

This is the last time you should have to touch the Makefile for this lab. In future labs, you will have to edit the Makefile to list all of your source files.

## 1.5 Building the Project

The next step is to build the project. This is as simple as hitting the build icon on the CodeWright toolbar. The commands executed by *make* will be shown in the **Build** window. If there are errors, they will also show up in the build window. Simply double click on the error message and CodeWright will take you to the location of the error in the correct file.

## 1.6 Connect and Open the Noral Debugger

You will use the Noral BDM Flex debugger to download and run your program. First, carefully connect the Flex debugging pod to your target board (BDM IN) using the following procedure:

For the Technological Arts Adapt9S12DP256 Board:

1. Make sure the EVB is not connected to power.
2. Connect the GND lead to ground at J5, Pin 10.
3. Connect the ECLK lead to J5, Pin 9, MODA to JB1, Pin 3, and MODB to JB1, pin 4.
4. Connect the 6-pin BDM connector to the BDM connector, J5, pins 1-6. Make sure the black wire on the POD connector is connected to pin 1.
5. Power up the EVB

Now you can open the Flex debugging program. Select the **RESET(Hard)** button and then the **LOAD** button. Browse to your project directory and select the DWARF2 file, *Proj.abs*. This should load the program source into the debugger source window.

## 1.7 Flash Programming

The **LOAD** button on the Noral window does not automatically program the Flash ROM for you. It only loads the debugger. To program the Flash ROM, you will have to follow these directions:

**9S12DP256 Flash Programming.** Go to **View->BDM->Flash 0 Banks \$3C-\$3F**. Then select **Erase Flash**, to erase the Flash ROM. Then select **Capture File** and browse to find your *project.abs* file. Make sure the flash banks are defined as shown in Figure 1. Then select **Capture** and finally **Program Flash**. Remember to do this every time you change your code and reload it into the debugger.

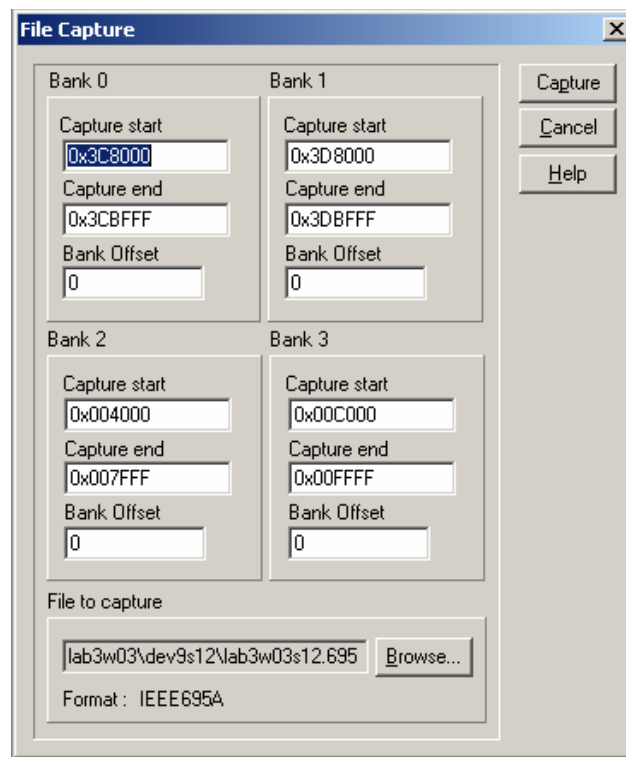


Figure 1 – Noral Flash Bank Options

## **ETec454 Lab #1 - Modular Software Development for C and Assembly (cont)**

---

### **1.8 Run the program**

You can either connect an active-low LED to PORTP, bit-0 or look at it with an oscilloscope. The program will generate 60 active-low pulses.

To run the program, select the **GO** button. Practice using the debugger. Set breakpoints, use the watch windows, etc. To run the program from the start again, hit **RESET(Hard)** and select **Go** again.

### **1.9 Submit Your Work**

This is an in-lab experiment only. When you have completed the procedure, have the instructor check your work.

### **Due dates**

**Lab Demonstration:** January 15, 2008