

# GCCHCS12 Project Final Report

## CSE4080: Computer Science Project Faculty of Science & Engineering York University

Course Director: Dr. Uyen T Nguyen  
Department of Computer Science & Engineering  
Office: Computer Science and Engineering Building, CSE 2024  
Phone: (416) 362100 x33274  
Email: [utn@cse.yorku.ca](mailto:utn@cse.yorku.ca)

Project Director: Dr. Mokhtar Aboelaze  
Department of Computer Science & Engineering  
Office: Computer Science and Engineering Building, CSE 2026  
Phone: (416) 362100 x40607  
Email: [aboelaze@cse.yorku.ca](mailto:aboelaze@cse.yorku.ca)

Student: Navid Mohaghegh – 206238984  
Email: [CS231381@cse.yorku.ca](mailto:CS231381@cse.yorku.ca)

## **DISCLAIMER**

Although all care has been taken to obtain correct information and accurate test results, we cannot be liable for any sort of incidental or consequential damages (including damages for loss of business, profits or the like) arising out of the use of the information provided in this report.

## **ABSTRACT**

Since the introduction of 68HC812A4 chips in 1996, there has been an increasing demand for information about HC12 development tools, software, sample projects and technical resources. There are many modular hardware implementations that brings HCS12 features for engineers and educators. However the current HCS12 Integrated Development Environment offered from Freescale, known as CodeWarrior Studio, is a proprietary software and is only available on Microsoft Windows systems. Unfortunately, current open source attempts are highly experimental and developers' time are being wasted by reading unorganized documentations instead of focusing on the project itself. GCC HCS12 project tries to provide GPL type development environment with various examples and guidelines for easily developing HCS(X)12 applications.

# TABLE OF CONTENTS

DISCLAIMER.....	I
ABSTRACT .....	II
BACKGROUND INFORMATION.....	1
EMBEDDED SYSTEMS.....	1
DIFFICULTIES WITH EMBEDDED SYSTEMS.....	1
FREE SOFTWARE FOUNDATION.....	1
FREESCALE SEMICONDUCTOR, INC.....	2
HCS12 AS A GOOD OPTION FOR EMBEDDED SYSTEMS.....	4
ABOUT GCC-HCS12 PROJECT .....	5
GNU DEVELOPMENT TOOL CHAIN FOR HCS12.....	6
GCC HC1X - HANDLING REGISTER SHORTAGES .....	6
GCC HC1X - STACK FRAME LIMITATIONS .....	6
GCC HC1X - MEMORY MODELS .....	7
GCC HC1X - RUN-TIME LIBRARY (LIBGCC) .....	7
GCC HC1X - STANDARD LIBRARY (LIBC ETC.) .....	7
GCC HC1X - SOURCE S.....	7
BUILDING GNU DEVELOPMENT TOOL CHAIN FROM SOURCE FOR HC1X.....	8
1 - COMPILING GNU BINUTILS FOR HC1X.....	8
2. COMPILING GCC FOR HC1X.....	9
3. COMPILING GDB FOR HC1X.....	10
4. COMPILING NEWLIB FOR HC1X.....	11
USING BINARIES OF GNU DEVELOPMENT TOOL CHAIN FOR HC1X.....	12
GNU HC1X DEVELOPMENT TOOL CHAIN DOCUMENTATION.....	12
MINICOM.....	13
MINICOM INSTALLATION.....	13
MINICOM GENERAL NOTES.....	13
ECLIPSE FRAME WORK.....	14
ECLIPSE C/C++ DEVELOPMENT TOOLS PROJECT .....	14
INSTALLATION OF ECLIPSE CDT.....	14
USING ECLIPSE CDT IDE FOR HCS12.....	16
LINKING EXTERNAL HELPER SCRIPTS TO ECLIPSE CDT .....	18
USEFUL HINTS ON ECLIPSE CDT .....	18
PROJECT EXAMPLE – ADVANCE PWM ON HCS12.....	19
ADVANCE PWM ON HCS12 – GENERAL NOTES .....	19
ADVANCE PWM ON HCS12 – IMPLEMENTATIONS.....	20
PWM MODULE.....	21

LCD MODULE.....	22
KEYPAD MODULE.....	22
SCI MODULE.....	22
MENU SUBSYSTEM.....	22
JOB SUBSYSTEM.....	22
RTI MODULE.....	22
SERIAL TERMINAL SUBSYSTEM.....	23
PC-HCS12 COMMUNICATION .....	23
ADVANCE PWM ON HCS12 – COMPILATION USING GCC.....	28
ADVANCE PWM ON HCS12 AND ECLIPSE.....	29
PROJECT MILESTONES .....	30
PROJECT INITIATION .....	30
PROJECT DEFINITION.....	30
PROJECT PLANNING .....	30
PROJECT EXECUTION.....	30
REFERENCES.....	31
APPENDIX A – SOURCE CODE OF SAMPLE PWM PROJECT.....	33

# **BACKGROUND INFORMATION**

## ***EMBEDDED SYSTEMS***

An embedded system, as oppose to general purpose computer, is a special purpose system in which the computer is completely encapsulated by or dedicated to the device or system it controls. Unlike a general-purpose computer, such as a personal computer, an embedded system performs one or a few predefined tasks, usually with very specific requirements. Since the system is dedicated to specific tasks, design engineers can optimize it, reducing the size and cost of the product. Embedded systems are often mass-produced, benefiting from economies of scale.

## **DIFFICULTIES WITH EMBEDDED SYSTEMS**

Usually intermediate developers face the following issues while working on embedded platforms :

- Finding affordable hardware platform
- Finding easy-to-follow and organized documentations
- Finding open source and customizable compilers and assemblers
- Finding simple and affordable Integrated Development Environment (IDE)
- Difficulties with testing, transferring and debugging codes on target platform
- Finding simple and understandable examples for hardware modules
- Need of electrical knowledge and general understanding of target platform
- Need of high tech tools and expensive equipments (i.e. Oscilloscopes and logic analyzers)

## ***FREE SOFTWARE FOUNDATION***

Proprietary software is software with restrictions on using, copying and modifying as enforced by the proprietor. Restrictions on use, modification and copying is achieved by either legal or technical means and sometimes both. Technical means include releasing machine-readable binaries to users and withholding the human-readable source code. Legal means can involve software licensing, copyright, and patent law.

Exclusive legal rights to software by a proprietor are not required for software to be proprietary, since public domain software and software under a permissive license can become proprietary software by distributing compiled versions of the program without offering the source code. Proprietary software's restrictions make it an antonym of free software

For free software, the same laws used by proprietary software are used to preserve the freedoms to use, copy and modify the software. Proprietary software includes free-ware and shareware. It can be

commercial software, but public domain and all other free software can also be sold for a price and be used for commercial purposes.

According to the Free Software Foundation (FSF), proprietary software is any software that does not meet its definitions of free software or semi-free software. The term's literal meaning covers software that has an owner who exercises control over what users can do with it. One license of the FSF's, the GNU General Public License (GPL), asserts that the restrictions of free software offer computer users freedom while the restrictions of other software benefit only the owner and are unethical.

Proponents of proprietary software, like Microsoft, argue that innovation is driven more quickly when it is lucrative. They claim that the best way to ensure this motivation is to tie revenue to innovation. The proprietor uses a temporary monopoly with copyright and sometimes software patents that makes the software more expensive. A dependency on future versions and upgrades can make the monopoly permanent without the emergence of a competing software package, a situation termed "vendor lock-in". Proprietary software is said to create greater commercial activity over free software, especially in regard to market revenues.

A variety of activation or license management systems are emerging in proprietary software that prevent copyright infringement and determine how the software is used. If the proprietor ceases to exist or for any other reason does not provide keys for activation or to unlock discontinued products, legitimate users can be unable to re-activate existing software or use other hardware.

If the proprietor of a software package should cease to exist, or decide to cease or limit production or support for a proprietary software package, recipients and users of the package can be left at a disadvantage and have no recourse if problems are found with the software. Proprietors can fail to improve and support software because of business problems. Companies also end their support for a product for business and organizational planning purposes. The consequence is also tied to enticing more to upgrade and pay for newer versions.

## ***FREESCALE SEMICONDUCTOR, INC.***

Freescale Semiconductor, Inc. is an semiconductor manufacturer created by the divestiture of the Semiconductor Products Sector of Motorola in 2004. Freescale focuses on the automotive and embedded and communications markets for their semiconductor products. Freescale is among the Worldwide Top 20 Semiconductor Sales Leaders.

The 68HC12 (6812 or HC12 for short) is a 16-bit micro-controller family from Freescale Semiconductor. Originally introduced in 1994, the architecture is an enhancement of the previous

Freescale 68HC11. Programs written for the HC11 are usually compatible with the HC12, which has a few extra instructions. The first 68HC12 derivatives had a maximum bus speed of 8MHz and flash memory sizes up to 128kbytes.

Like the 68HC11, the 68HC12 has two 8-bit accumulators A and B (sometimes referred to as a single 16 bit register D), also two 16-bit registers X and Y, a 16-bit program counter, a 16-bit stack pointer and an 8 bit Condition Code Register.

Motorola has launched the new HCS12 (also known as MC9S12) product line in 2000. The bus speed was improved up to 25MHz and flash sizes up to 512kbytes. The MC9S12NE64 was introduced by Freescale in September 2004, claiming to be the "industry's first single-chip fast-Ethernet Flash micro-controller." It features a 25 MHz HCS12 CPU, 64K bytes of FLASH EEPROM, 8K bytes of RAM, and an Ethernet 10/100 Mbit/s controller. Since the HCS12 was a direct upgrade to the existing HC12 family, most of the links and information provided here were suitable for both lines.

Latest addition in 2004 was the advanced HCS12X, providing even more features, including the XGATE DMA co-processor. HCS12X is fully backward-compatible with HCS12 CPU. The S12X family utilizes the latest process technology. It boasts higher speed (40 MHz), more functionality, reduced power consumption and cost, and enhanced performance with the new XGATE on-chip memory-management and DMA module. The XGATE peripheral co-processor allows for some tasks to be offloaded from the CPU also allows several new instructions to increase performance.

Freescale announced the MC9S12XEP100 in May 2006 to further extend the S12X family to 50MHz bus speed and add a Memory protection unit (based on segmentation) and a hardware scheme to provide Emulated EEPROM.

## **HCS12 AS A GOOD OPTION FOR EMBEDDED SYSTEMS**

Freescall Semiconductor's HCS12 family of microcontrollers (MCUs) is the next generation of the highly successful 68HC12 architecture. Using Freescall's industry-leading 0.25  $\mu$ s Flash, the HCS12 is part of a pin-compatible family that scales from 32 KB to 512 KB of Flash memory. HCS12 provides an upward migration path from Freescall's 68HC08, 68HC11 and 68HC12 architectures for applications that need larger memory, more peripherals and higher performance. Also, with the increasing number of CAN-based electronic control units (ECUs), its multiple network modules support this environment by enabling highly efficient communications between different network buses.

HC1x series have been around from 1990s which makes them affordable, well tested and stable candidates for embedded projects. Documentations can be downloaded free of charge from Freescall Semiconductor's website. There are also some third party projects with source codes available free of charge on the internet. GNU tool chain has been ported and tested on HC1x MCUs from 1993. BDM features of HC12 series provide ease of transfer, testing and debugging codes to the actual HC12 platform.

## ABOUT GCC-HCS12 PROJECT

Since the introduction of 68HC812A4 in 1996, there has been an increasing demand for information about HC12 development tools, software, sample projects and technical resources. There are many modular hardware implementations that brings HCS12 features for engineers and educators.

However the current HCS12 Integrated Development Environment offered from Freescale, known as CodeWarrior Studio, is a proprietary software which is only available on Microsoft Windows systems. In addition, due to licensing issues it is not possible to compile more than 32KB of C or 1KB of C++ code in this IDE under academic license. These limitations highly restrict academic users to either buy expensive commercial versions of CodeWarrior or restrict their code size development.

Unfortunately, the current open source attempts are absolutely experimental and users are completely by their own. And usually users' time are being wasted by reading tremendous amount of unorganized documentation instead of focusing on the project itself.

In this project, we focus on providing GPL type development environment with various examples and guidelines for easily developing HCS12 applications. GNU compilers will be introduced and explained how they can be patched and be ready for HCS12 platform. At the end, we demonstrate a simple real-time operating system example for HCS12 platform. Our goal is to focus on development process and provide step by step and quick manual for each task. Below are the hardware target platforms will be covered in this project:

- HC11series (outdated and will not be covered)
- HC12 series (covered briefly as they are replacing by new HCS12 series)
- S12 series (covered in details)
- S12X series (covered briefly due to budget limitations)

The following training and prototyping boards are preferred:

- DRAGON12 Development Board (based on S12 series: MC9S912DP256 chip)
- Adapt9S12XDP512M2 Module (based on S12X series: MC9S12XDP512 chip)
- S12X T-Board with extra 256KB RAM Add-On-Board (based on S12X series: MC9S12XDP512 chip)

## **GNU DEVELOPMENT TOOL CHAIN FOR HCS12**

In 1993 Real-Time Systems Inc. port GCC to HC1x. They were looking for reliable, affordable and open source compiler for HC1x processor series.

Main goals were to have:

- ease of context switching
- good performance in code speed
- minimal code size
- the ability to handle 32-bit integers and single-precision floats
- similarity to desktop environments.

Some things they did not care about were:

- 64-bit integers and double-precision floats
- completeness of the “C” and math libraries
- the ability to compile standard Unix applications

### ***GCC HC1X - HANDLING REGISTER SHORTAGES***

GCC really works best on machines with many general-purpose registers, but the HC1x series have only three or four registers, depending on the processor. Several approaches are possible here:

- Limit the data types to chars and 16-bit ints, avoiding entirely the use of longs or floats
- Use RAM locations (in the base page) as "pseudo-registers", increasing the number of registers that the GCC code generator sees
- Force the code generator to use stack slots in place of hardware registers.

Fortunately, the GCC code generator already uses stack slots, even on larger processors, when it runs out of hardware registers. The biggest problem was coercing the code generator to keep longs and floats entirely on the stack. This was done by telling the compiler that it should keep longs and floats only in 32-bit registers, then telling it that there were never any such registers free. This trick caused all long and float variables and intermediates to end up on the stack. While it would be straightforward, 64-bit ints and double-precision floats never implemented.

### ***GCC HC1X - STACK FRAME LIMITATIONS***

The X register was used to point to the stack frame where temporaries, local variables, and function arguments are stored by the compiler. Luckily, in HC12 we have 16-bit indexed addressing

modes.

## ***GCC HC1X - MEMORY MODELS***

GCC HC12 port supports two models:

- The GCC option '-ms' chooses a 'small' 64 kbyte space.
- The GCC option '-mm' chooses a 'medium' model which has 64 kbytes of data space but which uses the bank switching hardware in some HC12 devices to extend the text space to 4 Mbytes.

Both the small and medium models use 16-bit pointers for efficiency. In order to have 16-bit function pointers in the medium models, small "thunk" functions were generated in the data space, one for each real function in the larger text space.

These thunk functions each contain a long call instruction to the real function out in the text space, followed by a short return instruction. The address of the thunk function then gives us something we can point to with a 16-bit pointer.

## ***GCC HC1X - RUN-TIME LIBRARY (LIBGCC)***

The support routines for non-inlined arithmetic ended up separate from the compiler source tree. In the normal GCC way of doing things, these routines should be in a library called 'libgcc.a', which the compiler would be configured to find automatically.

## ***GCC HC1X - STANDARD LIBRARY (LIBC ETC.)***

Due to the stack frame limitations, we try not to use the whole standard libraries such as glibc. Usually, just a few needed routines are used or re-written which helps to squeeze out unnecessary features.

## ***GCC HC1X - SOURCE S***

Sources consist of patch files to apply to a original distribution of binutils, gcc, gdb and newlib.

## ***BUILDING GNU DEVELOPMENT TOOL CHAIN FROM SOURCE FOR HC1X***

This description focuses on building the complete tool chain by hand. The source codes can be obtained from the download section of <http://gcc-hcs12.com> (Linux - [Source Codes](#))

Before you start, you will need:

- The GNU make utility
- GCC Compiler (3.2/3.3)
- GNU assembler, linker and utilities (binutils 2.14/2.15)
- The patch program (version 2.5 at least)
- The Texinfo documentation suite (to produce the documentation)
- Several Unix utilities (sh, cmp, cp, test, ...)

### **1 - COMPILING GNU BINUTILS FOR HC1X**

Make sure you have the followings:

- GNU Binutils 2.15: [binutils-2.15.tar.gz](#)
- M68HC1x fixes (20040801): [binutils-2.15-m68hc1x-20040801.diffs.gz](#)

Type the following commands:

```
gzip -d binutils-2.15-m68hc1x-20040801.diffs.gz
tar xvzf binutils-2.15.tar.gz
mv binutils-2.15 binutils-2.15-m68hc1x
cd binutils-2.15-m68hc1x
patch -p1 < ../binutils-2.15-m68hc1x-20040801.diffs
```

When you build the GNU Binutils, you will get the support for 68HC11 and 68HC12 at the same time. The assembler and linker have a default cpu target based on the option you used during configuration.

```
sh ./configure --target=m6812-elf \
  --program-prefix=m6812-elf-
```

Then, build everything:

```
make
```

This will create: ld, as, ar, ranlib and many other tools. Typing make install will install the binutils files:

```
make install
```

## 2. COMPILING GCC FOR HC1X

Make sure you have the followings:

- Compiler GCC 3.3.5: gcc-3.3.5.tar.gz
- M68HC1x fixes: gcc-3.3.5-m68hc1x-20050515.diffs.gz

GCC comes with a C, C++, Objective C, Fortran, Java and Ada compiler. You'll get the cross compiler for all these languages. However, only the C compiler was really ported at this time. The C++ compiler is known to work a little. Other compilers are not checked at all.

Go back to the main folder and run the following commands:

```
gzip -d gcc-3.3.5-m68hc1x-20050515.diffs.gz
tar xvzf gcc-3.3.5.tar.gz
mv gcc-3.3.5 gcc-3.3.5-m68hc1x
cd gcc-3.3.5-m68hc1x
patch -p1 < ../gcc-3.3.5-m68hc1x-20050515.diffs
```

To configure GCC, you must use the same option you have used during configuration of the GNU Binutils.

```
sh ./configure --target=m6812-elf \
  --program-prefix=m6812-elf- \
  --enable-languages=c,c++
```

Then, build everything:

```
make
```

This will create: xgcc, cpp, cc1 and libgcc.a (as well as many other stuff). Move to the gcc subdirectory and type make install to install the compiler files. They are normally installed under /usr/local/bin, and /usr/local/lib/gcc-lib/m6812-elf. The GNU binutils are assumed to be installed in /usr/local/m6812-elf.

```
cd gcc
make install
```

### 3. COMPILING GDB FOR HC1X

Make sure you have the followings:

- GDB sources: `gdb-6.2.tar.gz`
- M68HC1x fixes and improvements (20040829): `gdb-6.2-m68hc1x-20040829.diffs.gz`

Go back to the main folder and run the following commands:

```
gzip -d gdb-6.2-m68hc1x-20040829.diffs.gz
tar xvzf gdb-6.2.tar.gz
mv gdb-6.2 gdb-6.2-m68hc1x
cd gdb-6.2-m68hc1x
patch -p1 < ../gdb-6.2-m68hc1x-20040829.diffs
```

Configure and build GDB for 68HC11/68HC12:

```
sh ./configure --target=m6811-elf \
  --program-prefix=m6812-elf-
make
```

Typing `make install` will install everything.

```
make install
```

This will install `m6811-elf-gdb` and `m6811-elf-run` in the `/usr/local/bin` directory by default. Note that GDB recognizes automatically when the program is for a 68HC11 or for a 68HC12. The simulator is also configured automatically according to the program ELF file.

## 4. COMPILING NEWLIB FOR HC1X

Make sure you have the followings:

- Newlib sources: newlib-1.12.0.tar.gz
- M68HC1x port (20040801): newlib-1.12.0-m68hc1x-20040801.diffs.gz

Go back to the main folder and run the following commands:

```
gzip -d newlib-1.12.0-m68hc1x-20040801.diffs.gz
tar xvzf newlib-1.12.0.tar.gz
mv newlib-1.12.0 newlib-1.12.0-m68hc1x
cd newlib-1.12.0-m68hc1x
patch -p1 < ../newlib-1.12.0-m68hc1x-20040801.diffs
```

The NEWLIB 1.12.0 must be built in a separate directory from the sources. Create such directory before configuring and building NEWLIB. For example, you can type the following commands:

```
cd ..
mkdir build-newlib
cd build-newlib
sh ../newlib-1.12.0-m68hc1x/configure \
  --disable-newlib-io-float --disable-newlib-multithread \
  --target=m6812-elf --program-prefix=m6812-elf-
make CFLAGS="-g -Os -Wall"
```

This will create a sub-directory m6812-elf that will contain the libc, libm and libbcc libraries. These libraries are compiled several times so that you will get 2 sets of 4 versions of them. One set for 68HC11 and one set for 68HC12, each set being composed of libraries compiled for:

- 32-bit integers, 64-bit double
- 32-bit integers, 32-bit double (-fshort-double)
- 16-bit integers, 64-bit double (-mshort)
- 16-bit integers, 32-bit double (-mshort -fshort-double)

Note if you don't specify the CFLAGS options at make time, the default makefile will use `-g -O2 -W -Wall`. You can also avoid the support for source level debugging by using: `CFLAGS="-Os -Wall"`. Also do not pass any `-mshort -m68hc11 -m68hc12 -mlong-calls` or `-fshort-double` option because this will break the multi-lib support.

Typing 'make install' will install the libraries and the includes in `/usr/local/m6812-elf/lib` and in `/usr/local/m6812-elf/include`.

make install

## **USING BINARIES OF GNU DEVELOPMENT TOOL CHAIN FOR HC1X**

If you do not feel comfortable to compile everything from scratch, you have the option to use pre-compiled and ready binaries of GNU Development Tool Chain. Please use below to see which one suits you the most and refer to download section of <http://gcc-hcs12.com>.

- Linux - DEB Based (Debian, Ubuntu, Knoppix ...)
  - [Have super user access and can install packages](#)
  - [Don not have super user access](#)
- Linux - RPM Based (Redhat, Fedora, CentOS, Suse ...)
  - [Have super user access and can install packages](#)
  - [Don not have super user access](#)
- Linux - Other (Gentoo, Arch ...)
  - Use Alien program along with Linux RPM
  - Or use related distributer package manager (Apt, Yum, Portage ... )
- FreeBSD:
  - Use Linux Emulation along with Linux binaries
- [Windows](#)

## **GNU HC1X DEVELOPMENT TOOL CHAIN DOCUMENTATION**

Please refer to <http://gcc-hcs12.com> ([Official Documentation of GNU Development Tools](#))

# MINICOM

Minicom is a text-based modem control and terminal emulation program for Unix-like operating systems, originally written by Miquel van Smoorenburg, and modeled after the popular MS-DOS program TeliX. Minicom includes a dialing directory, full ANSI and VT100 emulation, an (external) scripting language, and other features. Minicom is a menu-driven communications program. It also has an auto zmodem download. It can easily be used as a replacement of Microsoft HyperTerminal in Unix like operating systems.

## MINICOM INSTALLATION

In Linux, you need to download the source code of Minicom under the Downloads section of <http://gcc-hcs12.com> and run the followings:

- `tar -zxvf minicom-2.2.tar.gz; cd minicom-2.2;`
- `./configure`
- `make; make install`

## MINICOM GENERAL NOTES

To start Minicom, open a shell and type 'minicom'. A help menu screen with the various Minicom commands can be brought up at any time by typing CTRL-A Z. This means hold down the control key and type A, then release the control key and type Z (the a and z are case insensitive). To exit, type CTRL-A X. To set your options type CTRL-A O. To send a file type CTRL-A S.

```
-----+-----
                Minicom Command Summary
                Commands can be called by CTRL-A <key>

                Main Functions                Other Functions
-----+-----+-----
| Dialing directory..D | run script (Go)...G | Clear Screen.....C |
| Send files.....S   | Receive files.....R | cOnfigure Minicom..0 |
| comm Parameters...P | Add linefeed.....A | Suspend minicom...J |
| Capture on/off....L | Hangup.....H       | eXit and reset....X |
| send break.....F   | initialize Modem...M | Quit with no reset.Q |
| Terminal settings..T | run Kermit.....K   | Cursor key mode...I |
| lineWrap on/off...W | local Echo on/off..E | Help screen.....Z   |
| Paste file.....Y   |                    | scroll Back.....B   |
-----+-----+-----

                Select function or press Enter for none.█

                Written by Miquel van Smoorenburg 1991-1995
                Some additions by Jukka Lahtinen 1997-2000
                118n by Arnaldo Carvalho de Melo 1998
-----+-----
CTRL-A Z for help | 9600 8N1 | NOR | Minicom 2.2 | VT102 | Offline
```

## **ECLIPSE FRAME WORK**

Eclipse is an open-source software framework written primarily in Java. In its default form it is a Java Integrated Development Environment (IDE), comprising the Java Development Tools (JDT) and compiler (ECJ). Users can extend its capabilities by installing plug-ins written for the Eclipse software framework, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules.

### ***ECLIPSE C/C++ DEVELOPMENT TOOLS PROJECT***

Commonly known as CDT (Eclipse CDT), this is one of the most popular project under the auspices of the Eclipse Foundation. It adds C/C++ project structure and syntax recognition to the Eclipse platform. However, CDT does not include its own set of build tools but uses the GNU tool chain instead. On UNIX-like operating systems, these tools are typically supplied as part of the distribution and might not require the user to perform additional configuration for Eclipse. On Windows, however, users must supply their own build tools. This can be done through a GNU port such as Minimalist GNU for Windows (MinGW) or cygwin, or possibly by supplying tools from another IDE such as Visual Studio.

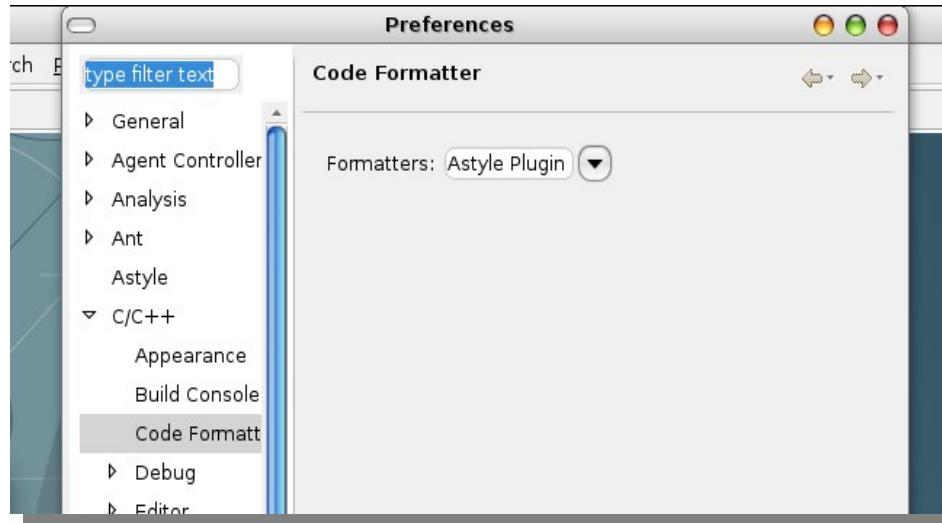
Being able to change the compiler command and its settings in an Eclipse IDE provides the possibility to develop codes almost for any processor and platforms using Eclipse CDT. In GCC-HCS12 project, Eclipse CDT is being used along with HC12 GNU Tool Chain Suit to provide an IDE for HCS12 platforms. In addition, capability of Eclipse in supporting external plugins provides the option to easily make external scripts and link them to Eclipse CDT IDE. These scripts can be used to ease the process of uploading and debugging the code on the HC12 platforms. We will explain how mentioned procedures can be done below. For more details, please refer to tutorial section of <http://gcc-hcs12.com>.

## **INSTALLATION OF ECLIPSE CDT**

Please visit <http://www.eclipse.org> and refer to 'Downloads' section and try to download Eclipse IDE for C/C++ Developers. Remember in Linux, a '.tar.gz' file can be opened using 'tar -zxvf filename.tar.gz'. Usually the executable file is './eclipse' once you are in Eclipse CDT directory.

If your Eclipse CDT does not have a source code formatter and beautifier, you can use <http://astyleclipse.sourceforge.net>. This plugin can be easily installed using Eclipse update mechanism. To do so, you need to refer to Software Update section which is commonly located under Help menu. You can use Find and Install option to add a new channel (<http://astyleclipse.sourceforge.net/update>) and

follow from there. After installation make sure your Eclipse is using Astyle plugin by going to Preferences section which is commonly located under Window menu bar. Once you are in Preferences, you should check the options for your C/C++ code formatter and make sure its using Astyle.

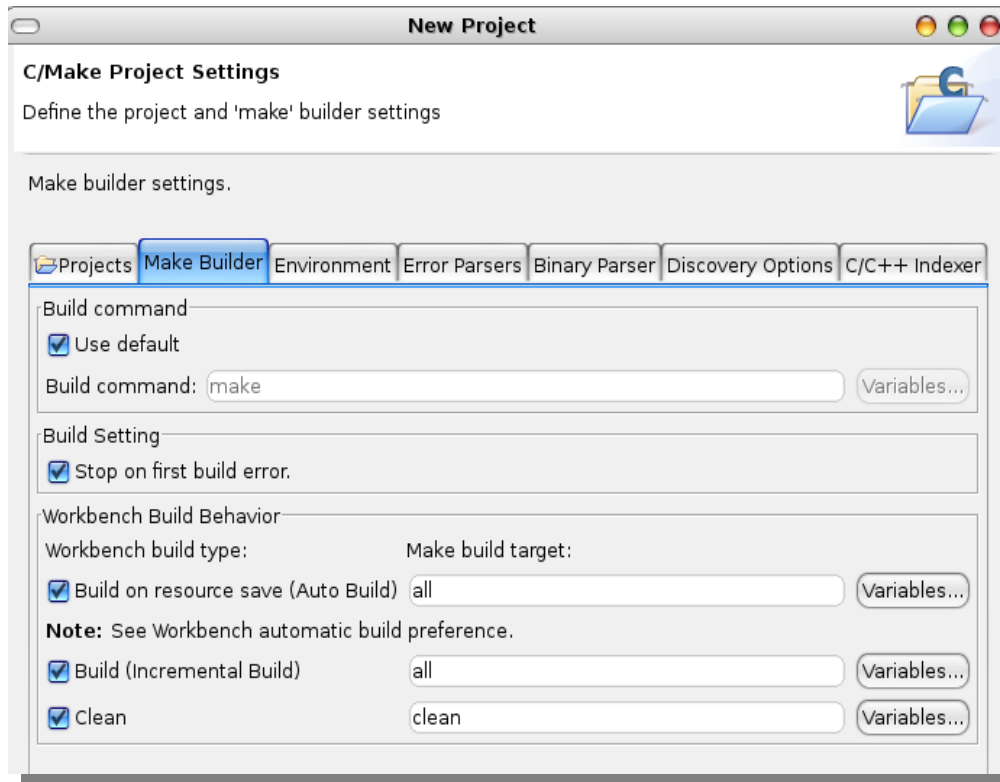


(ASTYLE PLUGIN FOR ECLIPSE)

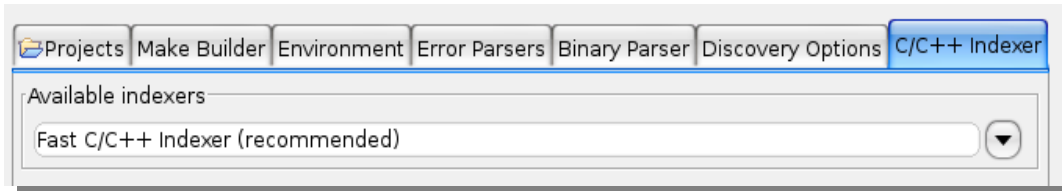
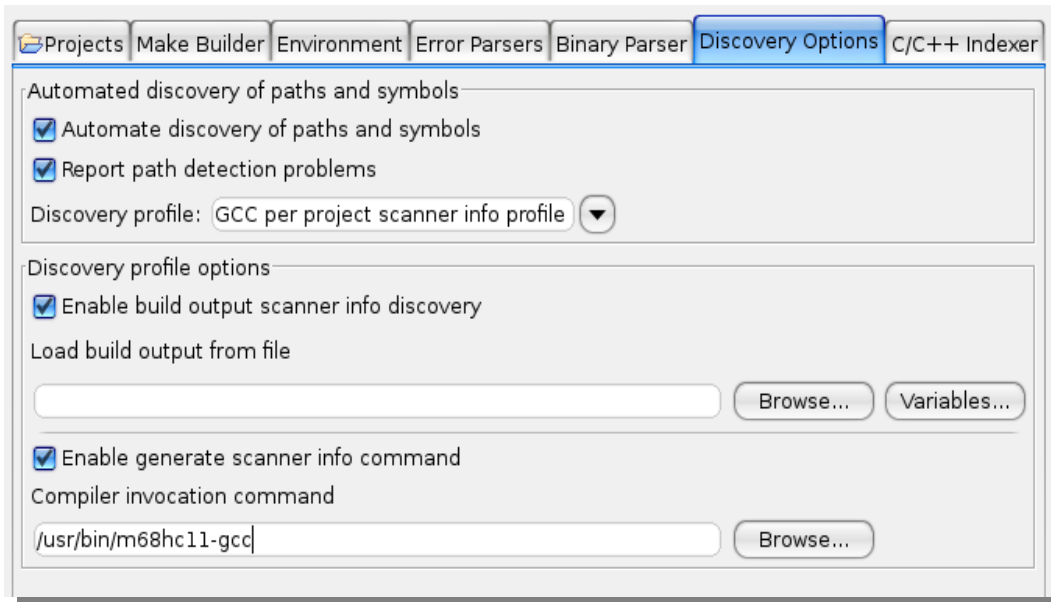
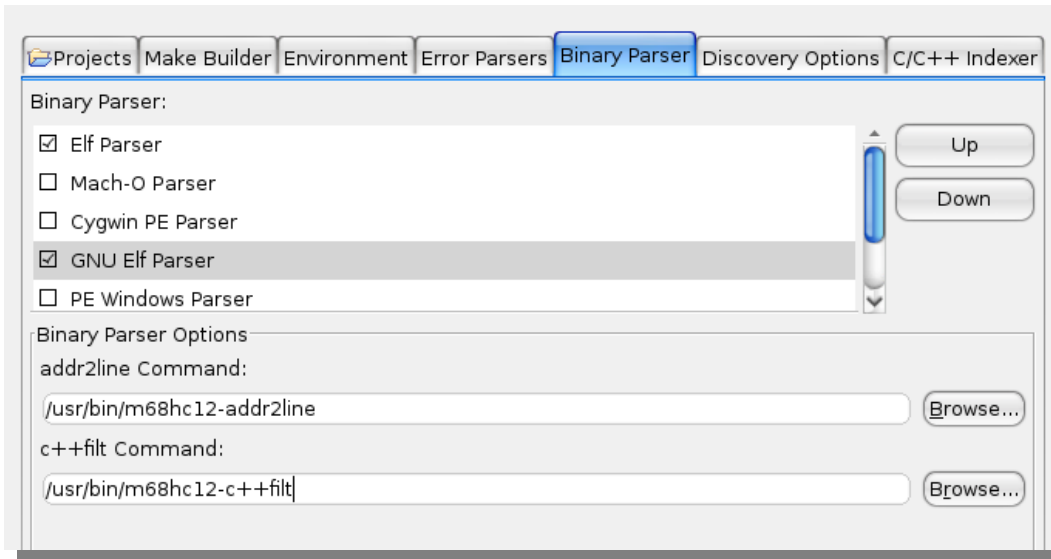
## USING ECLIPSE CDT IDE FOR HCS12

Once you install and run Eclipse CDT, you can make a new project by referring to File->New->Project. You should choose Standard C Make Project or any option that lets you manually control the Make file. Make file should not be controlled automatically as you are going to change the compiler commands name and options to GCC for HC12 later.

Eclipse usually use a Workplace directory to save the projects files and settings. This folder can be easily changed to whatever you prefer. Also make sure that on the new project wizard, you choose Build on Resource Save and also Stop on First Build Error options. As you will see later, we use 'make all' to build our projects and 'make clean' to clean and erase all the compiled files.



In Binary Parser section make sure that you have chosen GNU ELF Parser and you IDE is pointing to HC11/12 addr2line and c++filt commands. Also make sure your Compiler Invocation Command is pointing to GCC for HC11/12. Since GCC for HC11/12 is a uniform compiler, you can decide your target platform through the compiling flags in Makefile and the command name of the compiler itself does not play any role there. Keep in mind, it is recommended to use Fast C/C++ Indexer option.



## **LINKING EXTERNAL HELPER SCRIPTS TO ECLIPSE CDT**

Once you made a proper C project that can control the Make file manually, you can edit the Make file according to your needs and be able to compile your codes for HC12 platforms. But somehow you need to transfer, upload and debug the results. Obviously, Eclipse CDT does not have any feature on HC12, but you can link your helper scripts using External Tools options.

External Tools which is usually located under Run menu will give the option to be able to specify a command or script and set its working directory. You can choose that Eclipse will refresh your project files after running the scripts. Also you have the option to run the script in background. You can find helper scripts in 'bin' folder of each examples on <http://gcc-hcs12.com>.

In Linux, you can set the baud rate of a terminal by issuing 'stty' command. Also it is possible to echo a sentence (set of characters) to a character device like serial lines or even use 'cat' command to transfer the whole or part of a file to serial device. Sleep command is another useful tool to make sure your serial device got the command you sent to it. Most scripts that you see in the examples are using the above commands.

For each project you have to double check your helper scripts on External Tools section to make sure External Tools is pointing to the current project scripts and working directory. You can add as many scripts as you would like. Scripts can be organized using Organize Favorites option. Please refer to <http://gcc-hcs12.com> to see tutorial clips for more details.

## **USEFUL HINTS ON ECLIPSE CDT**

One of the most useful features of Eclipse IDE is syntax and code assistance. By pressing CTRL+SPACE or ALT+\ you will get code completion. The other feature is source code formatter and beautifier which can be done using CTRL+SHIFT+F. By going to Window->Preferences you can change the editor options such as colors, font size, line numbering and many more.

## **PROJECT EXAMPLE – ADVANCE PWM ON HCS12**

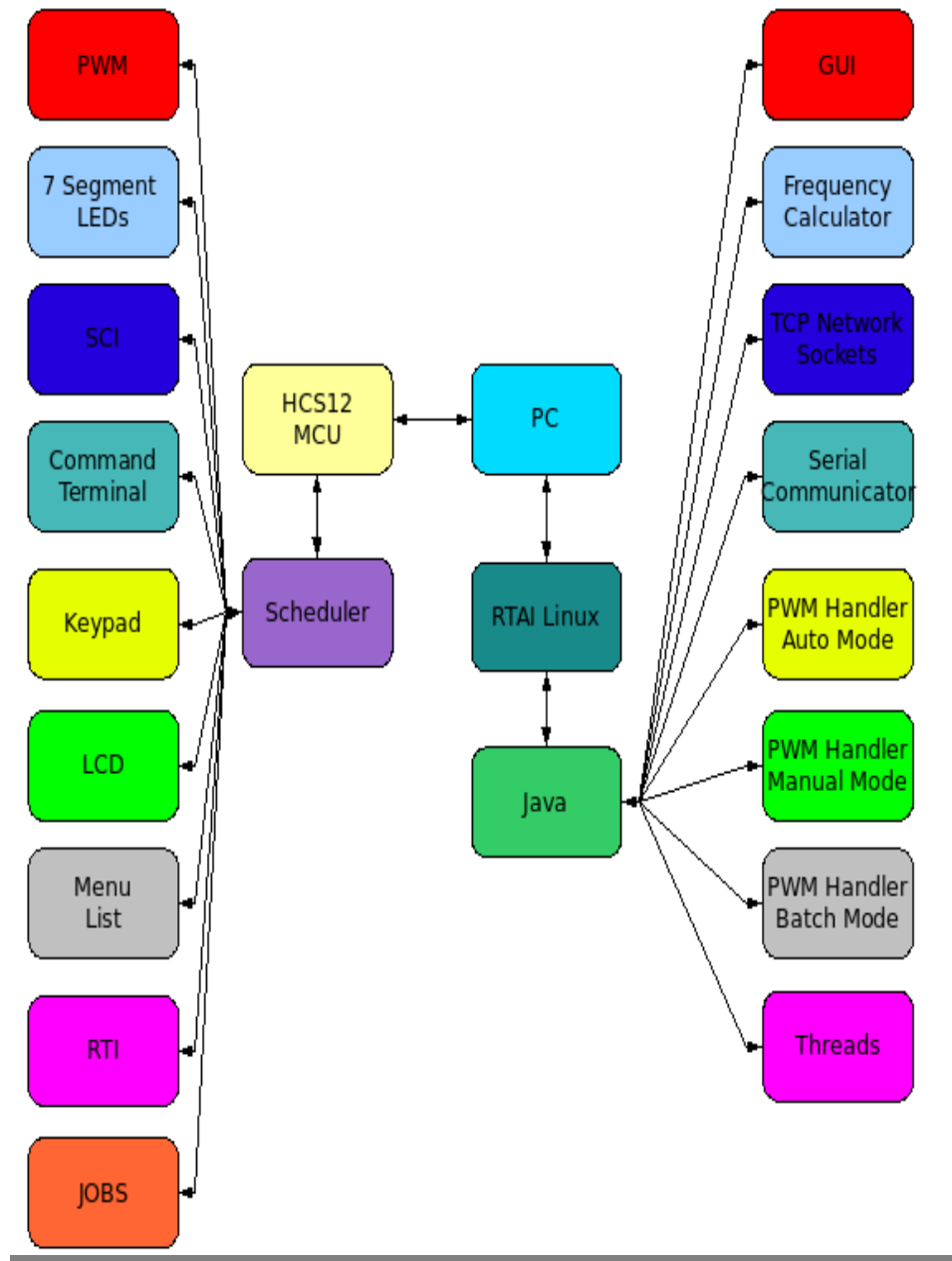
As an example, we are going to program HCS12 (MC9S12D series to be precise) to make a real-time changeable pulse-width modulation system. HCS12 should be able to get the commands from a keypad or an external serial device (i.e. personal computer via serial line).

### ***ADVANCE PWM ON HCS12 – GENERAL NOTES***

For user to be able to control the system via keypad, we need to display the settings' questions to user via a LCD and get the answers via a keypad. We need to use the following hardware modules of HCS12:

- Pulse-width modulation (PWM)
  - This module is actually making the pulses after setting the related settings' registers.
- Serial communication interface (SCI)
  - This is the module we use to communicate to external serial devices (i.e PC)
- General purpose input/output (IO)
  - PORTA will be used to scan keypad events
    - Keypad will be used to get the inputs from the user who is interacting directly with HCS12
  - PORTB will be used to display small 4 character message on 7-segment LEDs
  - PORTK will be used to communicate to LCD module
    - LCD will be used to display questions and messages to the user who is interacting directly with HCS12
- Real-time interrupts (RTI)
  - RTI will be used to help multi-tasking in the system. It acts as a clock which announce the time to the scheduler module in software section.

We also need to have a software subsystems which get and validate the commands from external serial device, display different menus on LCD and get and validate the answers from user. And more importantly, have a scheduler to run all these different tasks simultaneously.



(ADVANCE PWM ON HCS12 – GENERAL OVERVIEW)

## ***ADVANCE PWM ON HCS12 – IMPLEMENTATIONS***

The source code along with comments is in appendix A. You can also download this project and its related scripts through <http://gcc-hcs12.com>. Brief description of each module is as follows:

## PWM MODULE

This is the actual unit which makes the pulses. To use this module we need to know the followings:

- Mode of operation (8bit PWM or 16bit PWM. 16bit provide more adjustable and flexible pulses.)
- PWM channel (a number between zero to seven inclusive in 8bit mode and a number between zero to three inclusive in 16bit mode.)
- Alignment of the pulse (left or center aligned)
- Polarity (how the pulse will behave at start of the period? high or low)
- What is the value for main divider register? (a number between zero to seven inclusive)
- Do we want to use scaled divider on top of the main divider? If yes, what is the value for scaled divider? (a number from zero to 255 which zero counts as 256)
- What is the value of period register? (in 16bit mode, it can be a number from zero to  $2^{16}-1$  and for 8bit mode the limit will be  $2^8-1=255$ )
- What is the value of duty cycle register? (in 16bit mode, it can be a number from zero to  $2^{16}-1$  and for 8bit mode the limit will be  $2^8-1=255$ )

If pulse polarity is high at start: 
$$PWM\ duty\ cycle = \frac{DutyCycleReg}{PeriodReg} \times 100$$

If pulse polarity is low at start: 
$$PWM\ duty\ cycle = 100 - \frac{DutyCycleReg}{PeriodReg} \times 100$$

If pulse is left aligned: 
$$PWM\ frequency = \frac{CpuFreq}{2^{MainDividerReg} \times 2 \times ScaledDivider \times PeriodReg}$$

If pulse is center aligned: 
$$PWM\ frequency = \frac{CpuFreq}{2^{MainDividerReg} \times 2 \times 2 \times ScaledDivider \times PeriodReg}$$

Please notice, in 16bit mode channel zero will use the corresponding pin of channel one in 8bit mode. The reason is that two 8bit channel zero and one will be combined to make this 16bit channel. Similarly 16bit channel one will use the corresponding pin of 8bit channel three as two 8bit channel two and three are combined to make this 16bit channel and so on.

## **LCD MODULE**

This module helps to display or scroll a message on the two line sixteen characters LCD of Dragon12 development board. Currently the max number of characters that can be scrolled is 200 chars which easily can be changed to support longer messages. Scroll feature need to periodically be called from a job. The more frequent the job be executed, the faster the message will be scrolled on LCD.

## **KEYPAD MODULE**

This module helps to scan a 4x4 matrix keypad of Dragon12 development board. Like LCD scroll routine, scan keypad procedure need to be called periodically in order to capture all the events user is trying to input.

## **SCI MODULE**

This module will help us to read or write characters, string or numbers to HCS12 serial port. If the serial port is connected to PC, the serial line can be monitored and result can be displayed on the monitor of PC.

## **MENU SUBSYSTEM**

In order to interact with the user to get the PWM settings, we need to show a number of questions and get the corresponding answers. This subsystem facilitate LCD and Keypad modules to provide a suite of easily changeable menu system. Each menu can have a dedicated function to do any specific task related to that menu. In addition, each menu will have some basic routines such as get the user answer, redisplay the menu, erase the last character that user inputed and many more.

## **JOB SUBSYSTEM**

In this projects many tasks and procedure should be executed simultaneously. While system is scrolling a message on the LCD, user should be able to input something via keypad or serial line. To achieve this level of multi tasking, we have defined a job queue that when a real-time interrupt happens, the queue will be dispatched to run any ready functions. Each job in the queue have a priority, period and a function which corresponds to it. If a new job wants to be added to the system, all should be done is to define a job and its corresponding function and add the job to queue.

## **RTI MODULE**

This module provides an adjustable real-time clock which is used by Job subsystem. The faster the RTI is set the faster the JOB subsystem works.

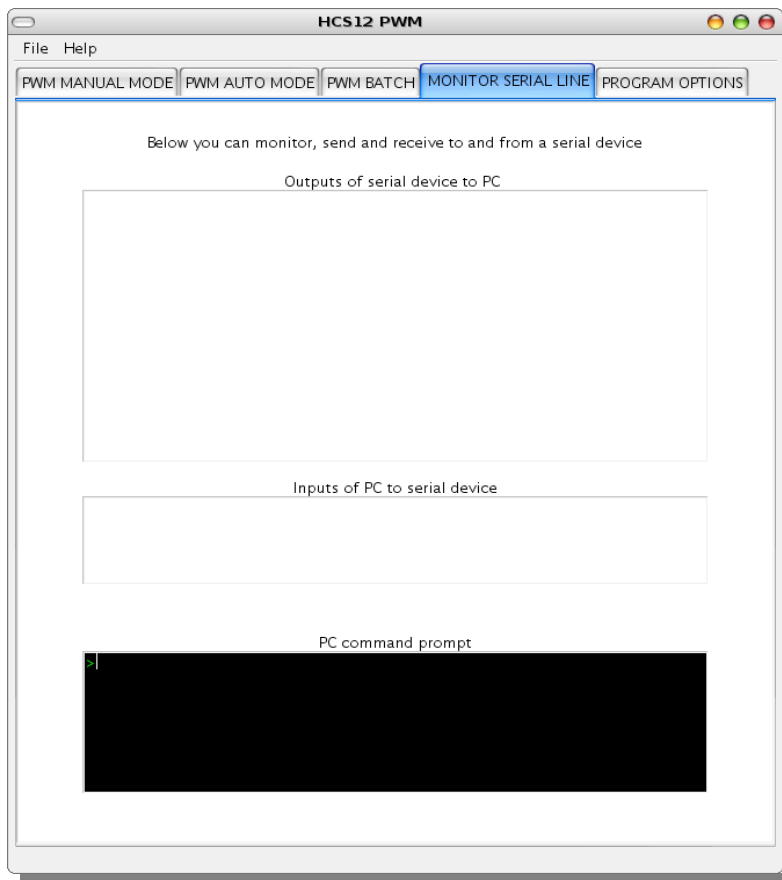
## SERIAL TERMINAL SUBSYSTEM

This section will take care of getting and validating the commands form PC or any external serial device. Communication syntax is “pc:op\_code&argument\_1&argument\_2&” while op\_code is an unsigned 8bit number and the two arguments are 16 bit unsigned numbers. For more information please refer to PC-HCS12 communication section.

## PC-HCS12 COMMUNICATION

On the PC side, we have Linux with low latency kernel. PWM graphical user interface has been written in Java Swing and using RXTX, Threads and Sockets to connect serially to HCS12. PC which runs the GUI can directly be connected to HCS12 (GUI will be in standalone mode) or can use TCP network to connect to another PC which is connected to HCS12 (GUI will be in client mode of operation).

GUI can provide a direct ASCII communication to HCS12. This communication is just like any serial communication program (i.e. Minicom or HyperTerminal) which can be used for debugging or manually sending commands to HCS12. Notice while PWM GUI is running, the serial port is locked and no other program can take over so it is very need to monitor the serial line we are we are blocking.

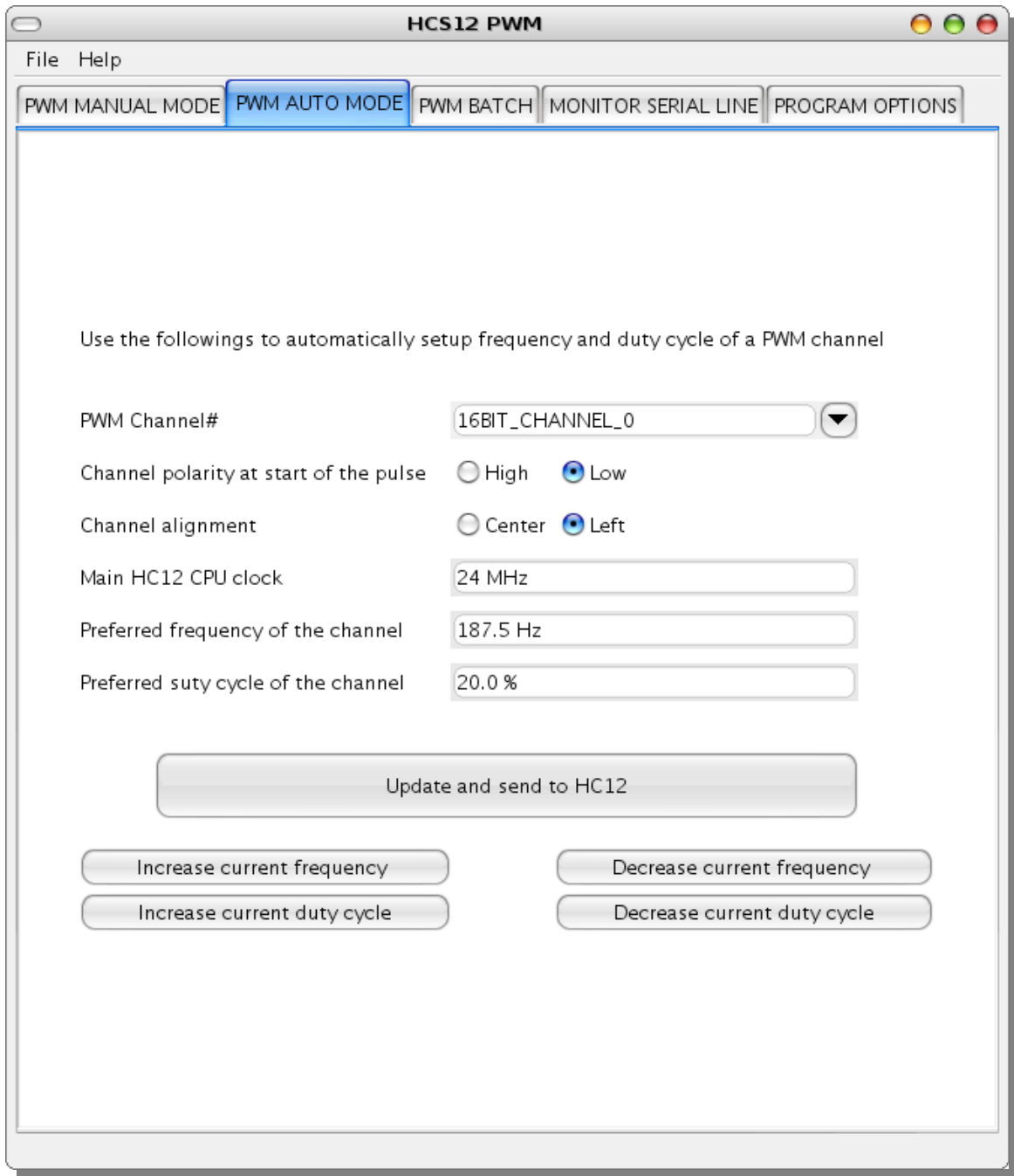


(PWM Serial Monitor)

PWM GUI can provide three different way of communication with HCS12:

- Automatic mode:

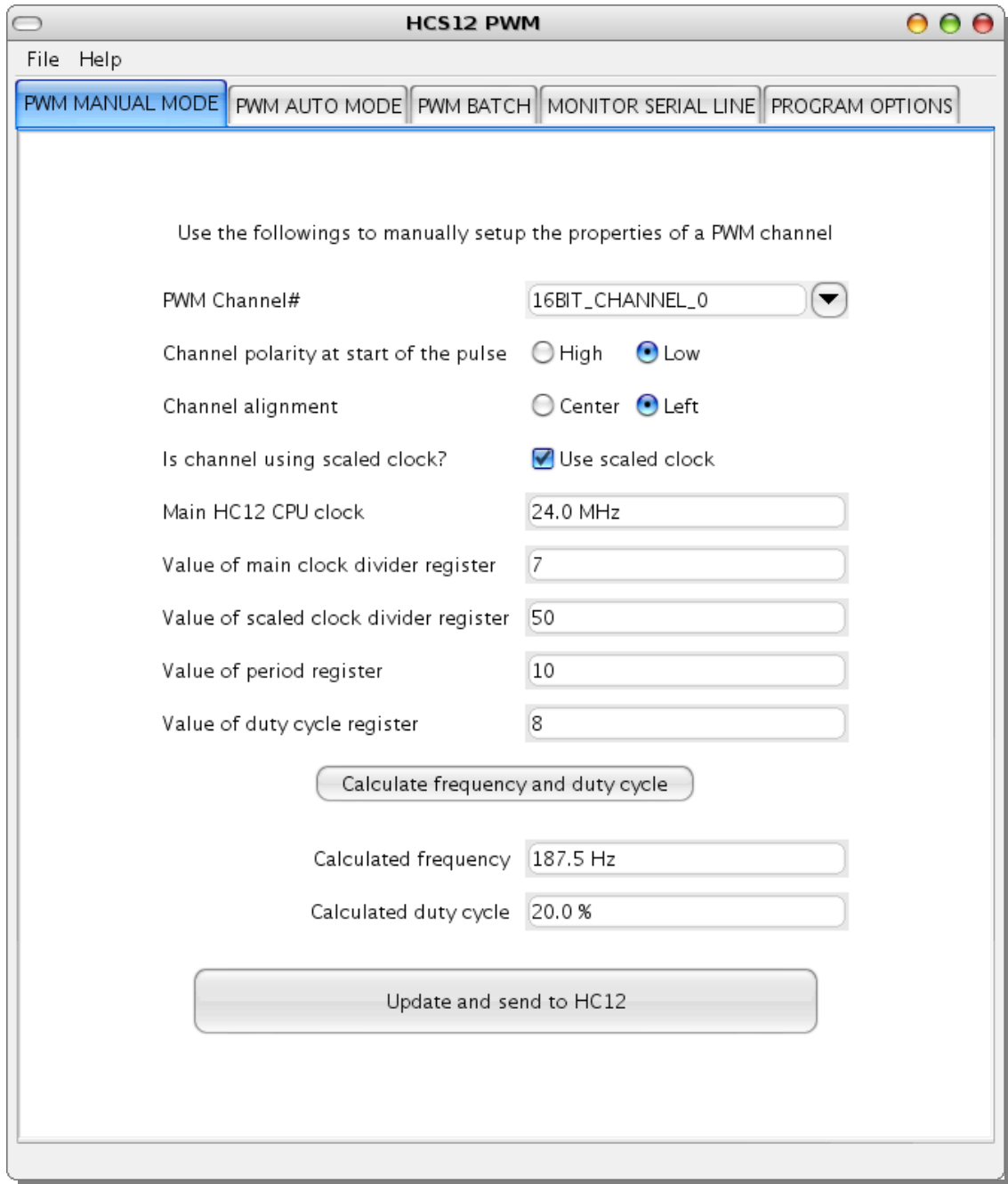
In this mode PWM channel, pulse frequency, pulse duty cycle, pulse alignment and polarity should be specified. PWM will be automatically generated accordingly.



(PWM Automatic Mode)

- Manual mode

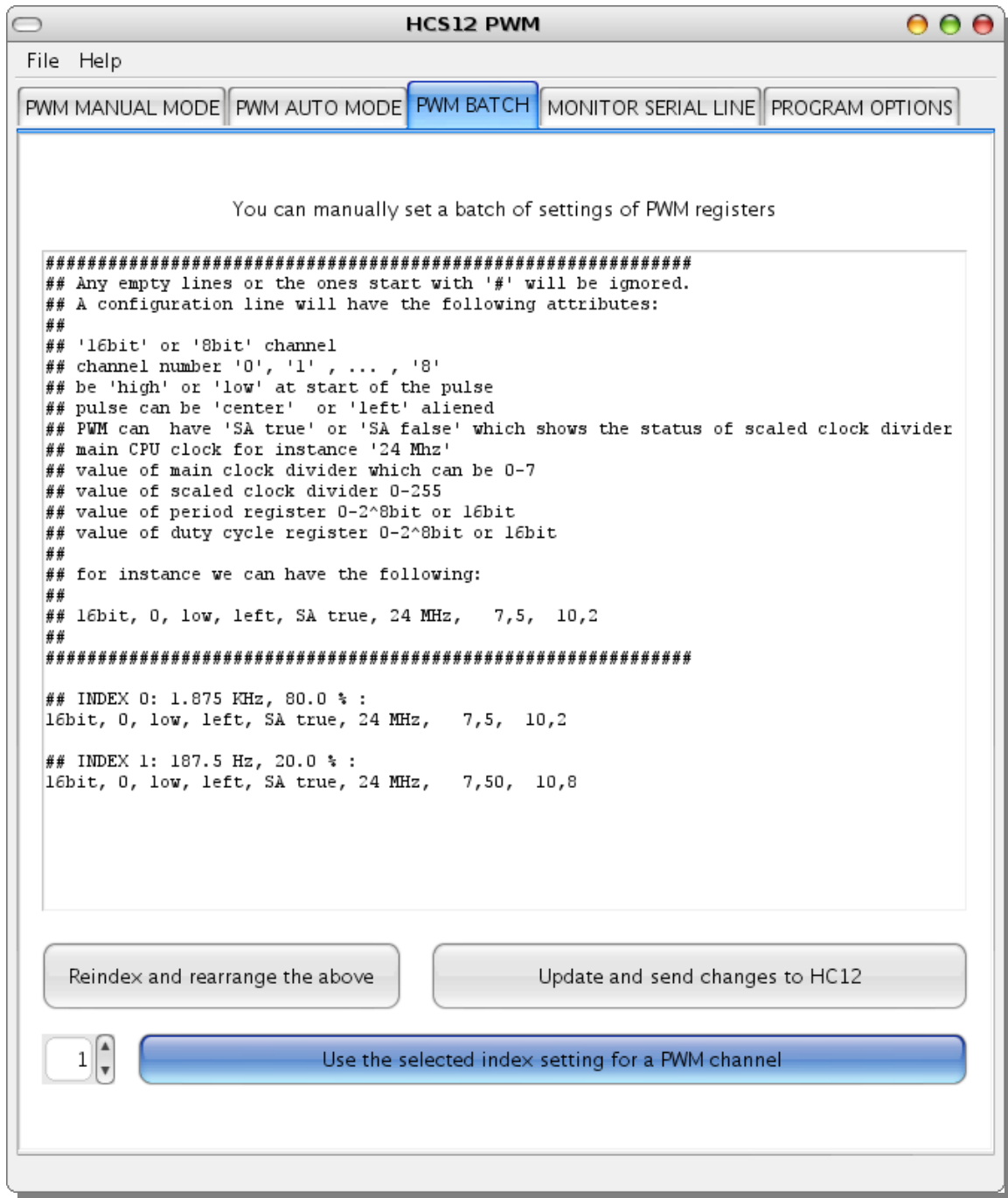
This is useful when we know the value of each HCS12 PWM registers and want to set the settings manually by hand.



(PWM Manual Mode)

- Batch mode

Calculating the corresponding PWM registers value through automatic mode is time and CPU consuming. We may want to specify a batch of pre-calculated PWM settings (register values) to be able to reload the settings from a batch and change the settings quickly without any delay caused by calculation.



(PWM Batch)

PC or any serial device can communicate to HCS12 using the following set of commands:

Category	OP Code	First Argument What register/subsystem?	Second Argument Corresponding value of subsystem
PWM channel settings	111	0 = for 8bit mode 1 = for 16bit mode	0-7 = channel number (8bit) 0-3 = channel number (16bit)
PWM flag settings	122	0 = for polarity 1 = for alignment 2 = is using scaled	0 = no or low or left 1 = yes or high or center
PWM clock settings	133	0 = for main clock divider reg 1 = for scaled clock divider reg 2 = for period register 3 = for duty cycle register	Value of the specified register
PWM update settings	144	the only value acceptable is 1	the only value acceptable is 1
PWM batch settings	155	1 = for update the batch array 2 = for load a setting from batch	index of the batch

For instance if the external serial device (PC in this case) want to have 187.5 Hz left aligned, low at start with the duty cycle of 80% it should send the following ASCII commands to HCS12:

“pc:111&1&0&” which means use 16bit mode, on channel 16bit zero

“pc:122&0&0&” which means the polarity is low at start

“pc:122&1&0&” which means the alignment is left

“pc:122&2&1&” which means we want to use scaled mode

“pc:133&0&7&” which means the main divider register is 7

“pc:133&1&50&” which means the scaled clock divider is 50

“pc:133&2&10&” which means the period register value is 10

“pc:133&3&2&” which means the duty cycle register value is 10

“pc:144&1&1&” which means update and effect the received settings

Alternatively, if the user change any settings through keypad module on HCS12 itself, the board will notify the connect external serial device through the same convention as above.

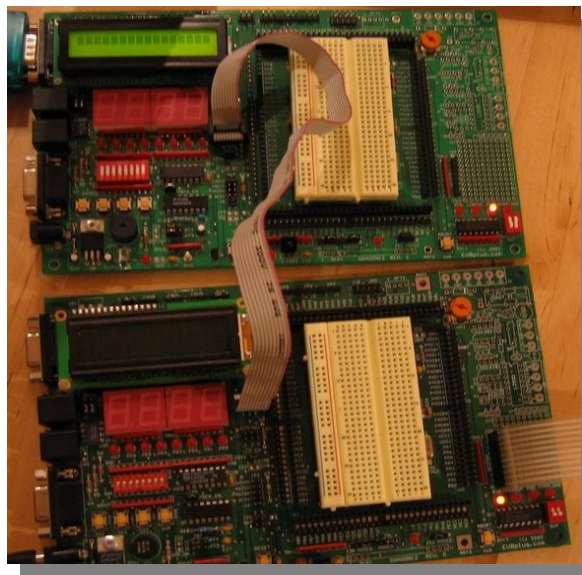
## **ADVANCE PWM ON HCS12 – COMPILATION USING GCC**

First please notice since project is using many hardware modules and also involves single precision floating point operations, the compiled code will be large and you have to burn the compiled project to flash area of HCS12 MCU as it does not fit into 12KB of RAM.

In order to compile the project, you need to have GNU tool chain installed on your system (refer to GNU DEVELOPMENT TOOL CHAIN FOR HCS12 section). The next step will be to download the project from <http://gcc-hcs12.com> and uncompress the project zip file.

You can edit the `Makefile` of the project and make sure that your Make script is pointing to proper files. If you do not have GNU tool chain path included in your `PATH` environmental variable you can use commands like `SETENV` to do so. You can clean the project which removes all the compiled files by issuing `make clean` command. The project can be built by using `make` command. As mentioned before, since you need to program the FLASH area of your MCU, you need a POD device. Please refer to <http://gcc-hcs12.com> for more information.

After you build the project, you can manually transfer the S record file (called `000.s19`) to HCS12 using Minicom (refer to MINICOM section). Or you can use the provided script in bin directory of the project to transfer the file automatically (called `upload_to_hcs12`). Edit the script to make sure it is pointing to proper serial device.



(POD CONFIGURATION – MASTER/SALVE BOARDS)

## ***ADVANCE PWM ON HCS12 AND ECLIPSE***

To import the project to Eclipse CDT 3.2, you first need to download the project from <http://gcc-hcs12.com> and follow the steps mention in USING ECLIPSE CDT IDE FOR HCS12 section.

# **PROJECT MILESTONES**

## ***PROJECT INITIATION***

- I1. Research on different project scenarios (2007-04-27)
- I2. Meetings with potential supervisors (2007-05-01)

## ***PROJECT DEFINITION***

- D1. Defining scope of the project (2007-05-02)
- D2. Determine deliverables, constraints (2007-05-02)

## ***PROJECT PLANNING***

- P1. Brainstorming and project plans (2007-05-02)
- P2. Distribute works (2007-05-05)
- P3. Contract proposal and project scope due - (2007-05-11)
- P4. Detailed work breakdown and scheduling (2007-05-11)

## ***PROJECT EXECUTION***

- E1. Purchase target development platforms (2007-05-11)
- E2. Make the GCC-HCS12 website (2007-05-12)
- E3. Develop an IDE for GCC-HCS12 project (2007-06-16)
- E4. Develop small documents and guidelines for different component of HCS12 (2007-06-13)
- E5. Develop sample projects for different component of HCS12 (2007-06-23)
- E6. Develop small real-time OS (2007-07-10)
- E7. Introduction to FREE-RTOS for HCS12 platform (2007-07-20)

## REFERENCES

[0] GCC-HCS12 Website

<http://gcc-hcs12.com>

[1] MC9S12DP256 based BDM with D-bug12:

[http://www.evbplus.com/Dragon\\_BDM/dragon\\_bdm.html](http://www.evbplus.com/Dragon_BDM/dragon_bdm.html)

[2] DRAGON12 Development Board:

[http://www.evbplus.com/dragon12\\_hc12\\_68hc12\\_9s12\\_hcs12.html](http://www.evbplus.com/dragon12_hc12_68hc12_9s12_hcs12.html)

[3] MiniDRAGON+ Development Board:

[http://www.evbplus.com/minidragonplus\\_hc12\\_68hc12\\_9s12\\_hcs12.html](http://www.evbplus.com/minidragonplus_hc12_68hc12_9s12_hcs12.html)

[4] Oliver Thamm's HC12 Web:

<http://elmicro.com/hc12web/index.html>

[5] HCS12X T-Board:

<http://elmicro.com/en/hcs12tb.html>

[6] Technological Arts Adap9S12XDP512M2 XGATE MCU Module:

[http://www.technologicalarts.ca/catalog/product\\_info.php?cPath=50\\_154\\_155&products\\_id=367](http://www.technologicalarts.ca/catalog/product_info.php?cPath=50_154_155&products_id=367)

[7] Tom Almy's Free Simulator for 68HC12:

<http://hcs12text.com/freesim.html>

[8] Linux Devices:

<http://linuxdevices.com>

[9] GCC tool chain for 68HC12:

[http://m68hc11.serveftp.org/m68hc11\\_inst\\_ptc.php](http://m68hc11.serveftp.org/m68hc11_inst_ptc.php)

[10] GCC tool chain for 68HC12-FAQ:

<http://m68hc11.serveftp.org/wiki/index.php/FAQ:Link>

[11] Developing Embedded Software in C Using ICC11/ICC12/Metrowerks:

<http://users.ece.utexas.edu/~valvano/embed/toc1.htm>

[12] CodeWarrior for HCS12(X) Microcontrollers:

[http://www.freescale.com/webapp/sps/site/prod\\_summary.jsp?code=CWS-H12-STDED-CX](http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=CWS-H12-STDED-CX)

[13] EmbeddedGNU IDE for Windows:

<http://www.ericengler.com/EmbeddedGNU.aspx>

[14] Freescale S12X chip:

[http://www.freescale.com/webapp/sps/site/prod\\_summary.jsp?code=S12XE&nodeId=0162468636bJwn](http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=S12XE&nodeId=0162468636bJwn)

[15] More resources on 68HC12:

<http://www.mgtek.com/miniide/resources/hc12.aspx>

[16] Wikipedia, the free encyclopedia:

<http://en.wikipedia.org>

[17] Real Time Systems Inc.

<http://www.realtime.bc.ca>

[18] Eclipse Framework

<http://www.eclipse.org>

[19] Astyle plugin for eclipse

<http://astyleclipse.sourceforge.net>

# APPENDIX A – SOURCE CODE OF SAMPLE PWM PROJECT

```
#include "hcs12.h"

/*****7-segment LED START*****/

/**
 * 7 segments LED decoder
 * 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,G,H
 *
 * Example: if you want to show "8" on LED segments
 * you should do the following:
 *
 * DDRB = 0xff;
 * PORTB = segment_decoder[8];
 */
unsigned int segment_decoder[]={
    0x3f,0x06,0x5b,0x4f,0x66,
    0x6d,0x7d,0x07,0x7f,0x6f,
    0x77,0x7c,0x39,0x5e,0x79,
    0x71,0x3d,0x76
};

/*
 * timer to set when we want show 'donE' on LEDs
 *
 * the bigger the timer, the more longer donE will be showed
 * on LEDs
 */
unsigned int display_donE_on_7segment_timer = 0;

/*
 * here we set the value of display_donE_on_7segment_timer
 *
 * since 'donE' is has four chars, if we like to show donE 4 times
 * we should set the timer counter to 4*4+1 = 17
 */
void enable_and_show_donE_on_7segment(void);
void enable_and_show_donE_on_7segment(void)
{
    display_donE_on_7segment_timer = 17; //4times shows donE ... 4*4+1
}

/*
 * Just a variable to keep track to see what is the current index on LEDs
 * I mean which segment LED ...
 */
int message_index_on_7segment_LEDs = 0;

/*
 * notice we are using pwm in this project here and
 * we can not play PTP ... means we have no control over which segmnet
 * of the LEDs we are going to choose ...
 *
 * If this procedure is being called, each time one of the chars
 * of 'donE' will be diplayed ...
 *
 * so if we call it 16 times, 'donE' will be diplayed 4 times ...
 */
void FLAT_FAR display_donE_on_7segment(void);
void FLAT_FAR display_donE_on_7segment(void)
```

```

{
  DDRB = 0xff; // PortB is set to be output.

  if (message_index_on_7segment_LEDs == 0)
  {
    PORTB = 0x5e; //d
  }
  else if (message_index_on_7segment_LEDs == 1)
  {
    PORTB = 0x5c; //o
  }
  else if (message_index_on_7segment_LEDs == 2)
  {
    PORTB = 0x54; //n
  }
  else if (message_index_on_7segment_LEDs == 3)
  {
    PORTB = 0x79; //E
  }

  message_index_on_7segment_LEDs++;

  if (message_index_on_7segment_LEDs > 3) //means we reach the end of 4 segmnets LEDs we
have
  message_index_on_7segment_LEDs = 0;
}

/*
 * here we take a look at the timer counter (above), to see is it possible to
 * show 'donE' on LEDs or not ... we manually set the timer whenever we want to show
 * donE on LEDS
 *
 * the procedure is being conatanlty called from JOBS section ...
 */
void display_donE_on_7segmnet_for_few_momnets__if_possible(void);
void display_donE_on_7segmnet_for_few_momnets__if_possible(void)
{
  if (display_donE_on_7segment_timer > 0)
  {
    display_donE_on_7segment();
    display_donE_on_7segment_timer--;

    if (display_donE_on_7segment_timer == 0)
    {
      DDRB = 0xff;
      PORTB = 0;
      message_index_on_7segment_LEDs = 0;
    }
  }
}

/*****7-segment LED END*****/
/*****PWM STRATS*****/

/*
 * below are some temp. vars. to save the values we get from PC or KEYPAD ...
 */
char pwm_mode, pwm_channel_number, pwm_polarity, pwm_alignment, pwm_is_using_scaled_clock,
pwm_main_clock_divider;
unsigned int pwm_scale_clock_divider, pwm_period_reg_value, pwm_duty_reg_value;

/*
 * bit value of enable and disable flags in PWME

```

```

*/
#define ENABLE 1
#define DISABLE 0

/*
 * bit value of polarity
 * polarity can be high at start of the pulse or can be low
 */
#define PWM_POLARITY_IS_HIGH_AT_BEGINNING_OF_THE_PULSE 1
#define PWM_POLARITY_IS_LOW_AT_BEGINNING_OF_THE_PULSE 0

/*
 * alignment: pulse can be left or center aligned
 */
#define PWM_ALIGNMENT_CENTER_ALIGN 1
#define PWM_ALIGNMENT_LEFT_ALIGN 0

/*
 * 16bit or 8bit of operation
 */
#define PWM_8BIT_OPERATION_MODE 8
#define PWM_16BIT_OPERATION_MODE 16

/*
 * channels
 */
#define PWM_8BIT_CHANNEL_0 0
#define PWM_8BIT_CHANNEL_1 1
#define PWM_8BIT_CHANNEL_2 2
#define PWM_8BIT_CHANNEL_3 3
#define PWM_8BIT_CHANNEL_4 4
#define PWM_8BIT_CHANNEL_5 5
#define PWM_8BIT_CHANNEL_6 6
#define PWM_8BIT_CHANNEL_7 7
#define PWM_16BIT_CHANNEL_0 0 //combined 8bit channel1 and 0, CH1 is used for output pin
#define PWM_16BIT_CHANNEL_1 1 //combined 8bit channel3 and 2, CH3 is used for output pin
#define PWM_16BIT_CHANNEL_2 2 //combined 8bit channel5 and 5, CH5 is used for output pin
#define PWM_16BIT_CHANNEL_3 3 //combined 8bit channel7 and 6, CH7 is used for output pin

/*
 * we have two clock source ... A and B
 * some channels use A and some use B
 */
#define PWM_IS_USING_CLOCK_A 0
#define PWM_IS_USING_CLOCK_B 1

/*
 * are we going to use scale our clock source further or not
 */
#define PWM_IS_USING_SCALED_CLOCK 1
#define PWM_IS_NOT_USING_SCALED_CLOCK 0

/**
 * cannal: 8 channel in 8bit or 4 channel in 16 bit
 * status: enable, disable ...
 * mode: 8bit mode or 16bit
 * polarity: hight at start or low at start
 * alignment: left or center aligned
 * is_using_scaled_clock: SA or SB ...

```

```

*/
typedef struct pwm_struct
{
    char channel;
    char status;
    char mode;
    char polarity;
    char alignment;
    char is_using_scaled_clock;
    char main_clock_divider_reg_value;
    int scaled_clock_divider_reg_value;
    unsigned int duty_reg_value;
    unsigned int period_reg_value;
}
PWM;

/*
 * We can precalculate and preset the setting for diffrenet Freq.
 * and whenever we want just call the related index in batch to simply run that
 * settings ...
 *
 * here we can preload 50 diffrent settings for channels.
 */
PWM pwm_batch[50];

/**
 * helps us to see which of clock A or B this givven channle is using ...
 * channel: 0-7 for 8bit and 0-3 for 16bit modes
 * mode: is 16bit or 8bit
 */
char FLAT_FAR pwm_is_using_which_clock(char channel, char mode);
char FLAT_FAR pwm_is_using_which_clock(char channel, char mode)
{
    if (mode == PWM_8BIT_OPERATION_MODE)
    {
        if (channel == PWM_8BIT_CHANNEL_0 ||
            channel == PWM_8BIT_CHANNEL_1 ||
            channel == PWM_8BIT_CHANNEL_4 ||
            channel == PWM_8BIT_CHANNEL_5
        )
            return PWM_IS_USING_CLOCK_A;
        else
            return PWM_IS_USING_CLOCK_B;
    }
    else // mode == PWM_16BIT_OPERATION_MODE
    {
        if (channel == PWM_8BIT_CHANNEL_0 ||
            channel == PWM_8BIT_CHANNEL_2)
            return PWM_IS_USING_CLOCK_A;
        else
            return PWM_IS_USING_CLOCK_B;
    }
}

/*
 * get a port and set the coresponding port number to 1 or 0
 */
void FLAT_FAR helper__bit_setter(volatile unsigned char * PORT, char bit_number, char
bit_value);
void FLAT_FAR helper__bit_setter(volatile unsigned char * PORT, char bit_number, char

```

```

bit_value)
{
    if (bit_value == 1)
    {
        *PORT |= 1 << bit_number;
    }
    else if (bit_value == 0)
    {
        *PORT &= ~(1 << bit_number);
    }
}

/**
 * set the priod and duty cycel registers for a channel
 */
void FLAT_FAR pwm_set_period_and_duty_registers(char channel, char mode, unsigned int
period_value, unsigned int duty_value );
void FLAT_FAR pwm_set_period_and_duty_registers(char channel, char mode, unsigned int
period_value, unsigned int duty_value )
{
    if (mode == PWM_8BIT_OPERATION_MODE)
    {
        char new_period_value = (char) period_value;
        char new_duty_value = (char) duty_value;

        switch (channel)
        {
            case 0:
            {
                PWMPER0 = new_period_value;
                PWMDTY0 = new_duty_value;
            }
            case 1:
            {
                PWMPER1 = new_period_value;
                PWMDTY1 = new_duty_value;
            }
            case 2:
            {
                PWMPER2 = new_period_value;
                PWMDTY2 = new_duty_value;
            }
            case 3:
            {
                PWMPER3 = new_period_value;
                PWMDTY3 = new_duty_value;
            }
            case 4:
            {
                PWMPER4 = new_period_value;
                PWMDTY4 = new_duty_value;
            }
            case 5:
            {
                PWMPER5 = new_period_value;
                PWMDTY5 = new_duty_value;
            }
            case 6:
            {
                PWMPER6 = new_period_value;
                PWMDTY6 = new_duty_value;
            }
        }
    }
}

```

```

        case 7:
        {
            PWMPER7 = new_period_value;
            PWMDTY7 = new_duty_value;
        }
    }
}
else
{
    switch (channel)
    {
        case 0:
        {
            PWMPER01_16BIT = period_value;
            PWMDTY01_16BIT = duty_value;
        }
        case 1:
        {

            PWMPER23_16BIT = period_value;
            PWMDTY23_16BIT = duty_value;
        }
        case 2:
        {
            PWMPER45_16BIT = period_value;
            PWMDTY45_16BIT = duty_value;
        }
        case 3:
        {

            PWMPER67_16BIT = period_value;
            PWMDTY67_16BIT = duty_value;
        }
    }
}
}

/**
 * PWME - PWM Enable Register
 * To enable PWM on channel0 and 1 you should write PWME = 0x03 ...
 * Once concatenated mode is enabled (CONxx bits set in PWMCTL register) then
 * enabling/disabling the corresponding 16-bit PWM channel is controlled by
 * the low order PWMEx bit.
 *
 * status is enabled or disabled ...
 * mode is 16bit or 8bit
 */
void FLAT_FAR pwm_set_status(char pwm_channel_number, char mode, char status);
void FLAT_FAR pwm_set_status(char pwm_channel_number, char mode, char status)
{

    if (mode == PWM_8BIT_OPERATION_MODE)
    {
        helper__bit_setter(&PWME, pwm_channel_number, status);
    }
    else
    {
        helper__bit_setter(&PWME, pwm_channel_number+pwm_channel_number+1, status);
    }
}

```

```

}

/*
 * if we like to change the settings for a PWM we have to reset the
 * configurations first by writing something to PWM counter to reset
 * the counter ...
 */
void FLAT_FAR pwm_reset_settings(char channel, char mode);
void FLAT_FAR pwm_reset_settings(char channel, char mode)
{
    if (mode == PWM_8BIT_OPERATION_MODE)
    {
        switch (channel)
        {
            case 0:
            {
                PWCNT0 = 1;
                pwm_set_status(channel,mode, DISABLE);
                PWCNT0 = 1;
            }
            case 1:
            {
                PWCNT1 = 1;
                pwm_set_status(channel,mode, DISABLE);
                PWCNT1 = 1;
            }
            case 2:
            {
                PWCNT2 = 1;
                pwm_set_status(channel,mode, DISABLE);
                PWCNT2 = 1;
            }
            case 3:
            {
                PWCNT3 = 1;
                pwm_set_status(channel,mode, DISABLE);
                PWCNT3 = 1;
            }
            case 4:
            {
                PWCNT4 = 1;
                pwm_set_status(channel,mode, DISABLE);
                PWCNT4 = 1;
            }
            case 5:
            {
                PWCNT5 = 1;
                pwm_set_status(channel,mode, DISABLE);
                PWCNT5 = 1;
            }
            case 6:
            {
                PWCNT6 = 1;
                pwm_set_status(channel,mode, DISABLE);
                PWCNT6 = 1;
            }
            case 7:
            {
                PWCNT7 = 1;
                pwm_set_status(channel,mode, DISABLE);
            }
        }
    }
}

```

```

        PPMCNT7 = 1;
    }
}
else
{
    switch (channel)
    {
        case 0:
        {
            PPMCNT01_16BIT = 1;
            pwm_set_status(channel,mode, DISABLE);
            PPMCNT01_16BIT = 1;
        }
        case 1:
        {
            PPMCNT23_16BIT = 1;
            pwm_set_status(channel,mode, DISABLE);
            PPMCNT23_16BIT = 1;
        }
        case 2:
        {
            PPMCNT45_16BIT = 1;
            pwm_set_status(channel,mode, DISABLE);
            PPMCNT45_16BIT = 1;
        }
        case 3:
        {
            PPMCNT67_16BIT = 1;
            pwm_set_status(channel,mode, DISABLE);
            PPMCNT67_16BIT = 1;
        }
    }
}
}
}

```

```

/**
 * PWMPOL - PWM Polarity Register
 * If the polarity bit is one, the PWM channel output is high at the beginning
 * of the cycle and then goes low when the duty count is reached. Conversely,
 * if the polarity bit is zero, the output starts low and then goes high when
 * the duty count is reached.
 */
void FLAT_FAR pwm_set_polarity(char pwm_channel_number, char mode, char polarity);
void FLAT_FAR pwm_set_polarity(char pwm_channel_number, char mode,char polarity)
{
    if (mode == PWM_8BIT_OPERATION_MODE)
    {
        helper__bit_setter(&PWMPOL, pwm_channel_number, polarity);
    }
    else
    {
        helper__bit_setter(&PWMPOL, pwm_channel_number+pwm_channel_number+1, polarity);
    }
}

```

```

/**
 * PWMCAE - PWM Center Align Enable Register
 * If the CAEx bit is set to a one, the corresponding PWM output will be center
 * aligned. If the CAEx bit is cleared, the corresponding PWM output will be left
 * aligned
 */
void FLAT_FAR pwm_set_alignment(char pwm_channel_number, char mode, char alignment);
void FLAT_FAR pwm_set_alignment(char pwm_channel_number, char mode, char alignment)
{
    if (mode == PWM_8BIT_OPERATION_MODE)
    {
        helper__bit_setter(&PWMCAE, pwm_channel_number, alignment);
    }
    else
    {
        helper__bit_setter(&PWMCAE, pwm_channel_number+pwm_channel_number+1, alignment);
    }
}

/*
 * helps us to set the 16bit or 8bit mode of operation for PWM channel.
 */
void FLAT_FAR pwm_set_operation_mode_flags(char pwm_channel_number, char mode);
void FLAT_FAR pwm_set_operation_mode_flags(char pwm_channel_number, char mode)
{
    if (mode == PWM_16BIT_OPERATION_MODE)
    {
        // why 4?? just an offset to reach CONxx bits
        helper__bit_setter(&PWMCTL, (pwm_channel_number+4), ENABLE);
    }
    else if (mode == PWM_8BIT_OPERATION_MODE)
    {
        helper__bit_setter(&PWMCTL, ((pwm_channel_number/2) +4), DISABLE);
    }
}

/*
 * set the main clock settings for a channel
 */
void FLAT_FAR pwm_set_main_clock(char channel, char mode, char is_using_scaled);
void FLAT_FAR pwm_set_main_clock(char channel, char mode, char is_using_scaled)
{
    if (mode == PWM_8BIT_OPERATION_MODE)
    {
        if (is_using_scaled)
        {
            helper__bit_setter(&PWMCLK, channel, PWM_IS_USING_SCALED_CLOCK);
        }
        else
        {
            helper__bit_setter(&PWMCLK, channel, PWM_IS_NOT_USING_SCALED_CLOCK);
        }
    }
    else // mode == PWM_16BIT_OPERATION_MODE
    {
        char bit = channel+channel+1; //to reach proper bit

        if (is_using_scaled)
        {

```

```

        helper__bit_setter(&PWMCLK,bit, PWM_IS_USING_SCALED_CLOCK);
    }
    else
    {
        helper__bit_setter(&PWMCLK,bit, PWM_IS_NOT_USING_SCALED_CLOCK);
    }
}

/*
 * set the main clock divider which can be 0-7
 *
 * 2^0 or 2^1 ... 2^7
 */
void FLAT_FAR pwm_set_main_clock_divider(char channel, char mode, char divider);
void FLAT_FAR pwm_set_main_clock_divider(char channel, char mode, char divider)
{
    //Selects prescale clock source for clocks A and B independently: XXX PCKB2 PCKB1 PCKB0
    XXX PCKA2 PCKA1 PCKA0
    if (pwm_is_using_which_clock(channel, mode) == PWM_IS_USING_CLOCK_A)
    {
        PWMPRCLK &= 0xf0; //reset the pins for A
        PWMPRCLK |= divider;
    }
    else
    {
        PWMPRCLK &= 0x0f; //reset the pins for B
        PWMPRCLK |= divider << 4;
    }
}

/*
 * set the scaled clock divider register, IF we are using scaled mode
 */
void FLAT_FAR pwm_set_scale_clock_divider(char channel, char mode, char is_using_scaled,
char divider );
void FLAT_FAR pwm_set_scale_clock_divider(char channel, char mode, char is_using_scaled,
char divider )
{
    if (is_using_scaled)
    {
        if (pwm_is_using_which_clock(channel, mode) == PWM_IS_USING_CLOCK_A)
        {
            PWMSCLA = 0;
            PWMSCLA |= divider;
        }
        else
        {
            PWMSCLB = 0;
            PWMSCLB |= divider;
        }
    }
}

/*****PWM ENDS*****/
/*****SCI STARTS*****/

/*128 Byte circular FIFO output sci1_buffer*/
#define BUFSIZE 128
unsigned char sci0_buffer[BUFSIZE] ;

```

```

unsigned char * volatile sci0_bufin = sci0_buffer ;
unsigned char * volatile sci0_bufout = sci0_buffer ;
volatile unsigned char sci0_charin = 0 ;

unsigned char sci1_buffer[BUFSIZE] ;
unsigned char * volatile sci1_bufin = sci1_buffer ;
unsigned char * volatile sci1_bufout = sci1_buffer ;
volatile unsigned char sci1_charin = 0 ;

#define BAUD_115200 13 //you will have around 5% error!!!!
#define BAUD_57600 26
#define BAUD_38400 39
#define BAUD_19200 78
#define BAUD_9600 156
#define BAUD_4800 313

void INTERRUPT sci0_isr( void );
unsigned char FLAT_FAR sci0_read_char(void);
void FLAT_FAR sci0_write_char(unsigned char val);
void FLAT_FAR sci0_init(unsigned char baud);
void FLAT_FAR sci0_write_string(const char *msg);
void FLAT_FAR sci0_print_integer(int num);

void INTERRUPT sci1_isr( void );
unsigned char FLAT_FAR sci1_read_char(void);
void FLAT_FAR sci1_write_char(unsigned char val);
void FLAT_FAR sci1_init(unsigned char baud);
void FLAT_FAR sci1_write_string(const char *msg);
void FLAT_FAR sci1_print_integer(int num);

/**
 * SCI baud rate = SCI module clock / (16 x BR),
 *
 * SCI Baud Rate Registers:
 * SCI BDH/L:
 *
 * SBR07 SBR06 SBR05 SBR04 SBR03 SBR02 SBR01 SBR00
 * X X X SBR12 SBR11 SBR10 SBR09 SBR08
 *
 * 13(115200bps),
 * 26(57600bps),
 * 39(38400bps),
 * 78(19200bps),
 * 156(9600bps),
 * 313(4800bps)
 *
 * Interuppt service routine for SCIO
 */
void sci0_isr( void )
{
    if(SCIOSR1 & 0x20)
    {
        /* RDRF is set */
        sci0_charin = SCI0DRL ;
    }
    if(SCIOSR1 & 0x80)
    {
        /* TDRE is set */
        if ( sci0_bufin == sci0_bufout )

```

```

        {
            /* Done -> Disable transmitter interrupt */
            SCI0CR2 &= ~0x80 ;
        }
        else
        {
            SCI0DRL = *sci0_bufout++ ;
            if(sci0_bufout == sci0_buffer + BUFSIZE)
            {
                sci0_bufout = sci0_buffer ;
            }
        }
    }
}

```

```

/*
 * Retrieves a character from the serial port
 */

```

```

unsigned char sci0_read_char(void)
{
    unsigned char result ;
    while((result = sci0_charin) == 0)
    {
        wait_for_interrupts() ;
    }
    sci0_charin = 0 ;
    return result ;
}

```

```

/*
 * Put a character onto serial port
 */

```

```

void sci0_write_char(unsigned char val)
{
    do
    {
        int used = sci0_bufin - sci0_bufout ;
        if ( used < 0 )
        {
            used += BUFSIZE ;
        }
        if(used < BUFSIZE - 1)
        {
            break ;
        }
        wait_for_interrupts() ;
    }
    while(1) ;
    *sci0_bufin++ = val ;
    if(sci0_bufin == sci0_buffer + BUFSIZE)
    {
        sci0_bufin = sci0_buffer ;
    }
    // Re-enable transmitter interrupt
    SCI0CR2 |= 0x80 ;
}

```

```

/*
 * Initialize serial line with 'baud' rate
 */

```

```

void sci0_init(unsigned char baud)

```

```

{
    SCIOBDL = baud ;
    /* Set RIE, TE and RE bits */
    SCIOCR2 = 0x2c ;
}

/*
 * Print a string onto the serial line
 */
void sci0_write_string(const char *msg)
{
    while (*msg != 0)
    {
        unsigned char val = *msg++;

        do
        {
            int used = sci0_bufin - sci0_bufout ;
            if ( used < 0 )
            {
                used += BUFSIZE ;
            }
            if(used < BUFSIZE - 1)
            {
                break ;
            }
            wait_for_interrupts();
        }
        while(1) ;
        *sci0_bufin++ = val ;
        if(sci0_bufin == sci0_buffer + BUFSIZE)
        {
            sci0_bufin = sci0_buffer ;
        }
        /* Re-enable transmitter interrupt */
        SCIOCR2 |= 0x80 ;
    }
}

/*
 * Interuppt service routine for SCI1
 */
void scil_isr( void )
{
    if(SCI1SR1 & 0x20)
    {
        /* RDRF is set */
        scil_charin = SCI1DRL ;
    }
    if(SCI1SR1 & 0x80)
    {
        /* TDRE is set */
        if ( scil_bufin == scil_bufout )
        {
            /* Done -> Disable transmitter interrupt */
            SCI1CR2 &= ~0x80 ;
        }
        else
        {

```

```

        SCI1DRL = *scil_bufout++ ;
        if(scil_bufout == scil_buffer + BUFSIZE)
        {
            scil_bufout = scil_buffer ;
        }
    }
}

/*
 * Retrieves a character from the serial port
 */
unsigned char scil_read_char(void)
{
    unsigned char result ;
    while((result = scil_charin) == 0)
    {
        wait_for_interrupts() ;
    }
    scil_charin = 0 ;
    return result ;
}

/*
 * Put a character onto serial port
 */
void scil_write_char(unsigned char val)
{
    do
    {
        int used = scil_bufin - scil_bufout ;
        if ( used < 0 )
        {
            used += BUFSIZE ;
        }
        if(used < BUFSIZE - 1)
        {
            break ;
        }
        wait_for_interrupts() ;
    }
    while(1) ;
    *scil_bufin++ = val ;
    if(scil_bufin == scil_buffer + BUFSIZE)
    {
        scil_bufin = scil_buffer ;
    }
    // Re-enable transmitter interrupt
    SCI1CR2 |= 0x80 ;
}

/*
 * Initialize serial line with 'baud' rate
 */
void scil_init(unsigned char baud)
{
    SCI1BDL = baud ;
    /* Set RIE, TE and RE bits */
    SCI1CR2 = 0x2c ;
}

/*
 * Print a string onto the serial line

```

```

*/
void scil_write_string(const char *msg)
{
    while (*msg != 0)
        scil_write_char(*msg++) ;
}

/*
 * Print an unsigned number to serial port 0
 */
void FLAT_FAR sci0_print_unsigned_integer(unsigned int num);
void sci0_print_unsigned_integer(unsigned int num)
{
    char digits[17];

    int i = 0;

    // zero as special case
    if(num == 0)
    {
        sci0_write_char('0');
        return;
    }

    for (i = 0; i <= 16; i++)
    {
        digits[i] = num % 10;
        num /= 10;
    }

    char should_skip = 1;
    for (i = 16; i >= 0; i--)
    {
        if (digits[i] == 0 && should_skip)
            continue;
        else
            should_skip = 0;

        sci0_write_char('0' + digits[i]);
    }
}

/*
 * Print a signed number to serial port 0
 */
void scil_print_integer(int num)
{
    char digits[6];

    int i = 0;

    // zero as special case
    if(num == 0)
    {
        scil_write_char('0');
        return;
    }
}

```

```

    }

    if(num < 0)
    {
        scil_write_char('-');
        num = -num;
    }

    for (i = 0; i <= 5; i++)
    {
        digits[i] = num % 10;
        num /= 10;
    }

    char should_skip = 1;
    for (i = 5; i >= 0; i--)
    {
        if (digits[i] == 0 && should_skip)
            continue;
        else
            should_skip = 0;

        scil_write_char('0' + digits[i]);
    }
}

/*****SCI DONE*****/
/*****COMMANDS FROM PC START*****/

/**
 * pc can send commands to micro controller. Format will be
 * pc:command_code&arg1&arg2&
 * comand code can be from 1 to (2^16 -1)
 * args can be from 1 to (2^16 -1)
 * sample:
 * "pc->mcu:125&0&10"
 */
void FLAT_FAR process_the_retrieved_command_from_pc(int command, unsigned int arg1,
unsigned int arg2);
void FLAT_FAR process_the_retrieved_command_from_pc(int command, unsigned int arg1,
unsigned int arg2)
{
    // these series of commands will control the PWM section
    if (command > 100 && command < 200)
    {
        /*
         * command for pwm channel////////////////////////////////////
         *
         * examples:
         * pc:111&0&1& means use PWM 8bit on channel 1
         * pc:111&1&0& means use PWM 16bit on channel 0
         */
        if (command == 111)
        {
            //16bit or 8bit
            if (arg1 == 0)
                pwm_mode = PWM_8BIT_OPERATION_MODE;
            else if (arg1 == 1)
                pwm_mode = PWM_16BIT_OPERATION_MODE;

            pwm_channel_number = (char) arg2;
        }
    }
}

```

```

/*
 * set the boolean flags //////////////////////////////////////
 *
 * examples:
 * pc:122&0&1& means use polarity high at start
 * pc:122&1&0& means use left aligned for the pulse
 */
else if (command == 122)
{
    if (arg1 == 0) //polarity
    {
        if (arg2 == 0)
            pwm_polarity = PWM_POLARITY_IS_LOW_AT_BEGINNING_OF_THE_PULSE;
        else if (arg2 == 1)
            pwm_polarity = PWM_POLARITY_IS_HIGH_AT_BEGINNING_OF_THE_PULSE;
    }
    else if (arg1 == 1) //align
    {
        if (arg2 == 0)
            pwm_alignment = PWM_ALIGNMENT_LEFT_ALIGN;
        else if (arg2 == 1)
            pwm_alignment = PWM_ALIGNMENT_CENTER_ALIGN;
    }
    else if (arg1 == 2) //is using SA clock
    {
        if (arg2 == 0)
            pwm_is_using_scaled_clock = PWM_IS_NOT_USING_SCALED_CLOCK;
        else if (arg2 == 1)
            pwm_is_using_scaled_clock = PWM_IS_USING_SCALED_CLOCK;
    }
}
else if (command == 133) ///values //////////////////////////////////////
{
    if (arg1 == 0) //main divider value
        pwm_main_clock_divider = arg2;
    else if (arg1 == 1) //scaled divider value
        pwm_scale_clock_divider = arg2;
    else if (arg1 == 2) //period value
        pwm_period_reg_value = arg2;
    else if (arg1 == 3) //duty value
        pwm_duty_reg_value = arg2;
}
else if (command == 144) ///UPDATE SETTINGS////////////////////////////////////
{
    if (arg1 == 1 && arg2 == 1)
    {
        pwm_reset_settings(pwm_channel_number, pwm_mode);

        pwm_set_polarity(pwm_channel_number, pwm_mode, pwm_polarity);
        pwm_set_alignment(pwm_channel_number, pwm_mode, pwm_alignment);
        pwm_set_operation_mode_flags(pwm_channel_number, pwm_mode);
        pwm_set_main_clock(pwm_channel_number, pwm_mode, pwm_is_using_scaled_clock);
        pwm_set_main_clock_divider(pwm_channel_number, pwm_mode,
pwm_main_clock_divider);
        pwm_set_scale_clock_divider(pwm_channel_number, pwm_mode,
pwm_is_using_scaled_clock, pwm_scale_clock_divider);
        pwm_set_period_and_duty_registers(pwm_channel_number, pwm_mode,
pwm_period_reg_value, pwm_duty_reg_value);

        pwm_set_status(pwm_channel_number, pwm_mode, ENABLE);

        enable_and_show_donE_on_7segment();
    }
}

```

```

    }
    else if (command == 155) ///UPDATE the BATCH PWM////////////////////////////////////
    {
        if (arg1 == 1) //update batch array
        {
            pwm_batch[arg2].alignment = pwm_alignment;
            pwm_batch[arg2].channel = pwm_channel_number;
            pwm_batch[arg2].duty_reg_value = pwm_duty_reg_value;
            pwm_batch[arg2].is_using_scaled_clock = pwm_is_using_scaled_clock;
            pwm_batch[arg2].main_clock_divider_reg_value = pwm_main_clock_divider;
            pwm_batch[arg2].mode = pwm_mode;
            pwm_batch[arg2].period_reg_value = pwm_period_reg_value;
            pwm_batch[arg2].polarity = pwm_polarity;
            pwm_batch[arg2].scaled_clock_divider_reg_value = pwm_scale_clock_divider;

            enable_and_show_donE_on_7segment();
        }
        else if (arg1 == 2) //load a pwm settings from batch array
        {
            pwm_alignment = pwm_batch[arg2].alignment ;
            pwm_channel_number = pwm_batch[arg2].channel;
            pwm_duty_reg_value = pwm_batch[arg2].duty_reg_value ;
            pwm_is_using_scaled_clock = pwm_batch[arg2].is_using_scaled_clock ;
            pwm_main_clock_divider = pwm_batch[arg2].main_clock_divider_reg_value ;
            pwm_mode = pwm_batch[arg2].mode ;
            pwm_period_reg_value = pwm_batch[arg2].period_reg_value ;
            pwm_polarity = pwm_batch[arg2].polarity ;
            pwm_scale_clock_divider = pwm_batch[arg2].scaled_clock_divider_reg_value ;

            pwm_reset_settings(pwm_channel_number, pwm_mode);
            pwm_set_polarity(pwm_channel_number, pwm_mode, pwm_polarity);
            pwm_set_alignment(pwm_channel_number, pwm_mode, pwm_alignment);
            pwm_set_operation_mode_flags(pwm_channel_number, pwm_mode);
            pwm_set_main_clock(pwm_channel_number, pwm_mode, pwm_is_using_scaled_clock);
            pwm_set_main_clock_divider(pwm_channel_number, pwm_mode,
pwm_main_clock_divider);
            pwm_set_scale_clock_divider(pwm_channel_number, pwm_mode,
pwm_is_using_scaled_clock, pwm_scale_clock_divider);
            pwm_set_period_and_duty_registers(pwm_channel_number, pwm_mode,
pwm_period_reg_value, pwm_duty_reg_value);
            pwm_set_status(pwm_channel_number, pwm_mode, ENABLE);

            enable_and_show_donE_on_7segment();
        }
    }
}

}

}

/*
* pc can send command to micro controller. Format will be
* pc:command_code&arg1&arg2
* comand code can from 1-255
* args can be from 1-255
* sample:
* "pc:125&0&10&"
* pc:125&0&10&

```

```

*/
void FLAT_FAR get_commands_from_pc(void);
void FLAT_FAR get_commands_from_pc(void)
{
    while (TRUE)
    {
        //commands from computer will start with "pc->mcu:"
        if (sci0_read_char() != 'p')
            continue;
        if (sci0_read_char() != 'c')
            continue;
        if (sci0_read_char() != ':')
            continue;

        //get the command code portion
        unsigned char temp = '0';
        int code = 0;
        int can_pass = TRUE;
        while (temp != '&')
        {
            int value = temp-'0';
            if (value <0 || value >9)
            {
                can_pass = FALSE;
                break;
            }
            code = 10*code + value;
            temp = sci0_read_char();
        }
        if (!can_pass)
            continue;

        //get the first argument
        temp = '0';
        int arg1 = 0;
        while (temp != '&')
        {
            int value = temp-'0';
            if (value <0 || value >9)
            {
                can_pass = FALSE;
                break;
            }
            arg1 = 10*arg1 + value;
            temp = sci0_read_char();
        }
        if (!can_pass)
            continue;

        //get the second argument
        temp = '0';
        int arg2 = 0;
        while (temp != '&')
        {
            int value = temp-'0';
            if (value <0 || value >9)
            {
                can_pass = FALSE;
                break;
            }
            arg2 = 10*arg2 + value;
            temp = sci0_read_char();
        }
    }
}

```

```

    }
    if (!can_pass)
        continue;

    //now we have everything
    process_the_retrieved_command_from_pc(code, arg1, arg2);
}

}
/*****COMMANDS FROM PC DONE*****/
/*****KAYPAD MODULE STARTS*****/
/*
 * Dragon12 and MiniDragon12+ Keypad (4x4 Key Pad Support)
 * Connected to PORTA 8 bits used.
 *
 * Port Set Up:
 * DDRA = 0x0f;          pa0-pa3 are outputs, pa4-pa7 are inputs
 * PUCR = PUCR | 1;     Init Port A enable pullups on porta for Key Pad.
 */

#define KBDEBOUNCE      1      // How many times the kb must be equal.
#define ROW_MASK        0xf0   // Mask for the row data.
#define COL1            0xe0
#define COL2            0xd0
#define COL3            0xb0
#define COL4            0x70

/*
 * Keypad - These tables map row/cols to numbers 0-15.
 * define this if you want to plug the keypad in the other way.
 */
#ifndef INV_KEYPAD
byte _kbdecoder[4][4]={
    {0xf,0xb,0x7,0x3},
    {0xe,0xa,0x6,0x2},
    {0xd,0x9,0x5,0x1},
    {0xc,0x8,0x4,0x0},
};
#else
byte _kbdecoder[4][4]={
    {0x0,0x1,0x2,0x3},
    {0x4,0x5,0x6,0x7},
    {0x8,0x9,0xa,0xb},
    {0xc,0xd,0xe,0xf},
};
#endif

/*
 * Keypad - Private variables used for the keypad.
 */
int FLAT_FAR ScanKeyPad(void);
int FLAT_FAR ScanKeyPad(void)
{
    byte d;
    byte rowsel = 0xf7; //pa3=low, pa0-pa2=high
    byte i;
    byte input;
    byte j;
    static volatile byte last_digit = -1;
    static volatile byte kbdebounce = 0;

```

```

DDRA = 0x0f; // pa0-pa3 are outputs, pa4-pa7 are inputs

/*
 * Scan the keypad by lowering one row line at a time.
 * then checking if a col went low.
 */
for( i = 0, rowsel = 0xf7 ; i < 4 ; ++i)
{
    PORTA = rowsel;
    for( d = 0 ; d < 16 ; ++d ) // small delay to settle.
        ;
    input = PORTA & ROW_MASK;

    if( input != ROW_MASK ) // a line was lo, so key pressed
        break;
    rowsel >>= 1; // select the next row...
}
d = 255;
if( i < 4 ) // Did we find any lows?
{
    // if 2 keys are pressed, then none will match.
    switch(input)
    {
        case COL1:
            j = 3;
            break;
        case COL2:
            j = 2;
            break;
        case COL3:
            j = 1;
            break;
        case COL4:
            j = 0;
            break;
        default:
            j = 255;
            ;
    }
    // Have a match ?
    if( j != 255 )
        d = _kbdecoder[i][j];
}
// Did anything decode ?
if( d != 255 )
{
    if( last_digit == d )
    {
        if(kbdebounce != 0xff)
            ++kbdebounce;
    }
    else
    {
        kbdebounce = 0;
        last_digit = d;
    }

    // Valid key press detected ?
    if(KBDEBOUNCE == kbdebounce )
    {
        kbdebounce = 0xff; // means we have seen this key press
        return d;
    }
}

```

```

    }
}
else
{
    // Nothing found reset debounce.
    last_digit = -1;
    kbdebounce = 0;
}

return -1;
}

#ifdef miniDragon
#define SW1 0x10
#define SW2 0x08
#define SW3 0x20
#define SW4 0x40
#define SWMASK (SW1 + SW2 + SW3 + SW4)
#define SWPORT PORTAD0
#else
#define SW2 0x8
#define SW3 0x4
#define SW4 0x2
#define SW5 0x1
#define SWMASK 0xff
#define SWPORT PTH

#define DIPSW8 0x80
#define DIPSW7 0x40
#define DIPSW6 0x20
#define DIPSW5 0x10
#define DIPSW4 0x8
#define DIPSW3 0x4
#define DIPSW2 0x2
#define DIPSW1 0x1
#endif
volatile byte switches;

int FLAT_FAR ScanSwitchPad(void);
int FLAT_FAR ScanSwitchPad(void)
{
    switches = SWPORT;
    return switches;
}

/*****KAYPAD MODULE ENDS*****/
/****LCD MODULE STARTS*****/

#define NUMBER_OF_LCD_LINES 2

#define LCD_FIRST_LINE 0
#define LCD_SECOND_LINE 1

/*
 * since we mentioned 200 chars, each of our lines on LCD scroll section
 * can have up to 200 chars ... you can increase this though ...
 */
#define LCD_MAX_BUFFER_LINE_LENGTH 200

typedef struct lcd_line_struct

```

```

    {
        char session_id;
        char line_number;
        char buffer[LCD_MAX_BUFFER_LINE_LENGTH];
        int current_index;
        int current_length;
        char original_string[LCD_MAX_BUFFER_LINE_LENGTH];
        int original_string_length;
        int block_backspace_below_this_index;
    }
LCD_LINE;

/*
 * Dragon12 LCD Module is in 4-bit mode. Port K bits:
 *
 * 7 = unused
 * 6 = unused
 * 5 = Module DB7
 * 4 = Module DB6
 * 3 = Module DB5
 * 2 = Module DB4
 * 1 = EN (pulse 1 to write)
 * 0 = RS (0=command, 1=data)
 */

// Delay constants - 24MHz CPU clock
#define ENBIT (0x02)
#define DELAY40US (2000L)
#define DELAY4_1MS (220000L)
#define DELAY100US (5000L)
#define LCDWIDTH (16)

/*
 * dummy delay function
 */
void FLAT_FAR lcd_delay(unsigned long constant);
void FLAT_FAR lcd_delay(unsigned long constant)
{
    volatile unsigned long counter;

    for(counter = constant; counter > 0; counter--)
        ;
}

LCD_LINE lcd_lines[NUMBER_OF_LCD_LINES];

/*
 * Write LCD module in 8-bit mode
 *
 * data: to be written, lower 4 bits are ignored
 * rs: register select, only bit 0 is significant
 *
 * Handles the shifting into place and the EN pulsing
 * This is only used at the start of the init sequence
 */
void FLAT_FAR write_to_lcd_8bit_mode(unsigned char data);
void FLAT_FAR write_to_lcd_8bit_mode(unsigned char data)
{
    unsigned char temp;

```

```

// shift upper nibble to data bits in portK
temp = (data >> 2); // rs is always 0

// Now do the EN pulsing
PORTK = temp;           // write with EN=0
PORTK = temp | ENBIT;  // write with EN=1
PORTK = temp;           // write with EN=0

// allow instruction to complete
lcd_delay(DELAY40US);
}

/*
 * Write LCD module in 4-bit mode
 *
 * data: to be written, 8 bits are significant
 * rs: register select, only bit 0 is significant
 *
 * Does two consecutive writes, high nibble, then low
 * Handles the shifting into place and the EN pulsing
 * This is can be used at any time (init and display)
 */
void FLAT_FAR write_to_lcd_4bit_mode(unsigned char data, unsigned char rs);
void FLAT_FAR write_to_lcd_4bit_mode(unsigned char data, unsigned char rs)
{
    unsigned char hi, lo;

    /*
     * split byte into 2 nibbles and shift to line up
     * with data bits in port K
     */
    hi = ((data & 0xf0) >> 2) | (rs & 0x01) ;
    lo = ((data & 0x0f) << 2) | (rs & 0x01) ;

    // do write pulses for upper, then lower nibbles
    PORTK = hi;           // write with EN=0
    PORTK = hi | ENBIT;  // write with EN=1
    PORTK = hi;           // write with EN=0
    PORTK = lo;           // write with EN=0
    PORTK = lo | ENBIT;  // write with EN=1
    PORTK = lo;           // write with EN=0

    // allow instruction to complete
    lcd_delay(DELAY40US);
}

/*
 * Init LCD module
 */
void FLAT_FAR lcd_init(void);
void FLAT_FAR lcd_init(void)
{
    //init the data direction register
    DDRK = 0xff;

    write_to_lcd_8bit_mode(0x30); // tell it once
    lcd_delay(DELAY4_1MS);
    write_to_lcd_8bit_mode(0x30); // tell it twice
    lcd_delay(DELAY100US);
    write_to_lcd_8bit_mode(0x30); // tell it thrice
}

```

```

// last write in 8-bit mode sets bus to 4 bit mode
write_to_lcd_8bit_mode(0x20);

// Now we are in 4 bit mode, write upper/lower nibble
write_to_lcd_4bit_mode(0x28, 0); // last function set: 4-bit mode, 2 lines, 5x7 matrix
write_to_lcd_4bit_mode(0x0c, 0); // display on, cursor off, blink off
write_to_lcd_4bit_mode(0x01, 0); // display clear
write_to_lcd_4bit_mode(0x06, 0); // cursor auto-increment, disable display shift
}

```

```

void FLAT_FAR lcd_reconfigure_a_lcd_line( char which_lcd_line, char session_id, char*
content);
void FLAT_FAR lcd_reconfigure_a_lcd_line( char which_lcd_line, char session_id, char*
content)

```

```

{
    LCD_LINE* lcd_line = & lcd_lines[which_lcd_line];

    lcd_line->session_id = session_id;
    lcd_line->line_number = which_lcd_line;

    int i = 0;
    for (i=0; i < LCD_MAX_BUFFER_LINE_LENGTH ; i++)
    {
        char temp = *content;

        lcd_line->buffer[i] = temp;
        lcd_line->original_string[i] = temp;

        if (temp == '\0')
            break;
        else
            content++;
    }

    lcd_line->current_length = i;
    lcd_line->original_string_length = i;
    lcd_line->block_backspace_below_this_index = i;

    lcd_line->current_index = 0;
}

```

```

/*
 * Write a line to the LCD
 *
 * string is a pointer to an array of char to be sent. It must be 16 long.
 * line tells which line to display (0=top line, 1=bottom line). Defaults to top.
 *
 * If the string is shorter, we will write garbage to the LCD.
 * If it's longer, it gets truncated.
 */

```

```

void FLAT_FAR lcd_write_line(char *string, int line);
void FLAT_FAR lcd_write_line(char *string, int line)
{
    int charCount;
    unsigned char instruction;

    /* Set address in LCD module */
    if( line == LCD_SECOND_LINE)
        instruction = 0xc0; // write bottom line
    else
        instruction = 0x80; // write top line
}

```

```

write_to_lcd_4bit_mode( instruction, 0); // rs=0 means command

/* blast out 16 bytes */
char is_null_has_been_seen = 0;
for( charCount = 0; charCount < LCDWIDTH; charCount++)
{
    if (string[charCount] == '\0')
    {
        is_null_has_been_seen = 1;
        break;
    }
    else
    {
        write_to_lcd_4bit_mode( string[charCount], 1); // rs=1 means data
    }
}

if (is_null_has_been_seen)
{
    int i = 0;
    for (i = 0; i < LCDWIDTH; i++)
    {
        write_to_lcd_4bit_mode( ' ', 1); // rs=1 means data
    }
}

}

/*
 * refresh and redisplay a line ...
 */
void FLAT_FAR lcd_redisplay_line(char which_lcd_line);
void FLAT_FAR lcd_redisplay_line(char which_lcd_line)
{
    LCD_LINE* lcd_line = &lcd_lines[which_lcd_line];
    lcd_write_line(lcd_line->buffer, lcd_line->line_number);
}

/*
 * try to scroll a LCD line
 */
void FLAT_FAR lcd_scroll(char which_lcd_line);
void FLAT_FAR lcd_scroll(char which_lcd_line)
{
    char * line;

    LCD_LINE* lcd_line = &lcd_lines[which_lcd_line];

    line = & lcd_line->buffer[lcd_line->current_index];

    if (lcd_line->current_index < lcd_line->current_length)
        lcd_line->current_index ++;
    else
        lcd_line->current_index = 0;

    lcd_write_line(line, lcd_line->line_number);
}

}

/*
 * ad a char to end of the line of LCD

```

```

*/
void FLAT_FAR lcd_append_to_end_of_line(char which_lcd_line, char to_add);
void FLAT_FAR lcd_remove_from_end_of_line(char which_lcd_line)
{
    LCD_LINE* lcd_line = &lcd_lines[which_lcd_line];

    if (lcd_line->current_length < LCD_MAX_BUFFER_LINE_LENGTH-1)
    {
        lcd_line->buffer[lcd_line->current_length] = to_add;
        lcd_line->current_length++;
        lcd_line->buffer[lcd_line->current_length] = '\\0';
    }

    lcd_redisplay_line(which_lcd_line);
}

/*
 * remove a char from the end of LCD line...
 * you may have a LCD line which shows "INPUT> " and you may not
 * want the use to be able to delete "INPUT >" part ... so you set the
 * limit for remove char by setting the block_backsapce_below_this_index
 */
void FLAT_FAR lcd_remove_from_end_of_line(char which_lcd_line);
void FLAT_FAR lcd_remove_from_end_of_line(char which_lcd_line)
{
    LCD_LINE* lcd_line = &lcd_lines[which_lcd_line];

    if (lcd_line->current_length > lcd_line->block_backsapce_below_this_index)
    {
        lcd_line->current_length--;
        lcd_line->buffer[lcd_line->current_length] = '\\0';
    }

    lcd_redisplay_line(which_lcd_line);
}

/*
 * each LCD line will start we the string that we want to diplay ...
 * later it can be altered by user, for instance user input something ...
 * we may want to reset the LCD to its original diplay ... soe we use this
 * method
 */
void FLAT_FAR lcd_reset_the_lcd_line_to_original( char which_lcd_line);
void FLAT_FAR lcd_reset_the_lcd_line_to_original( char which_lcd_line)
{
    LCD_LINE* lcd_line = & lcd_lines[which_lcd_line];

    int i = 0;
    for (i=0; i < LCD_MAX_BUFFER_LINE_LENGTH ; i++)
    {
        char temp = lcd_line->original_string[i];

        lcd_line->buffer[i] = temp;
        lcd_line->original_string[i] = temp;

        if (temp == '\\0')
            break;
    }

    lcd_line->current_length = i;
    lcd_line->block_backsapce_below_this_index = i;
}

```

```

    lcd_line->current_index = 0;

    lcd_write_line(lcd_line->buffer, lcd_line->line_number);
}

/*
 * if we append chars to LCD line, using this method we can see what user has
 * addd to the line
 */
char* FLAT_FAR lcd_get_the_appended_string_on_lcd_line( char which_lcd_line);
char* FLAT_FAR lcd_get_the_appended_string_on_lcd_line( char which_lcd_line)
{
    LCD_LINE* lcd_line = & lcd_lines[which_lcd_line];
    return & lcd_line->buffer[lcd_line->original_string_length];
}

/*
 * helps us to find a what is the numerical equivalent for string: "16"
 * which will be 16. ... just like atoi you used to use ...
 */
unsigned int FLAT_FAR string_to_int__atoi(char* input);
unsigned int FLAT_FAR string_to_int__atoi(char* input)
{
    char MAX_NUMBER_OF_BITS = 32;

    unsigned int result = 0;

    int i = 0;
    for (i=0; i < MAX_NUMBER_OF_BITS ; i++)
    {
        char temp = *input;

        if (temp == '\0')
            break;

        int value = temp - '0';

        if (temp == 'Y') //if user input YES
            return 1;
        else if (temp == 'N') //if user input NO
            return 0;
        else if( value >= 0 && value < 10 )
        {
            if (result > 6553) //overflow
                return 65535;
            else if (result == 6553 && value > 5) //overflow
                return 65535;
            else
                result = (unsigned int) result*10 + value;
        }

        input++;
    }

    return result;
}

/*****LCD MODULE ENDS*****/
/*****MENU LIST STARTS*****/

// Max. Number of menus
#define MAX_NUMBER_OF_MENUS 15

```

```

/*
 * session id: just an ID to keep track of the session for next and previous menu
 * message: is the question string that we want to ask
 * also we have a function for each menu if we like ... which will be run before going to
next menu
 */
typedef struct menu_struct
{
    int session_id;
    char* message;
    unsigned int answer_from_user;
    int next_session_id;
    int previous_session_id;
    int (*menu_item__function_pointer)(int session_id);
}
MENU;

MENU menus[MAX_NUMBER_OF_MENUS];
// what is the current menu on the LCD
int current_menu_session_id = 0;

/*
 * dummy function to just fill the function prototypes ...
 */
int FLAT_FAR menu_foobar__do_nothing(int session_id);
int FLAT_FAR menu_foobar__do_nothing(int session_id)
{
    return 0;
}

/*
 * answers from user for each menu
 */
unsigned int user_answers[MAX_NUMBER_OF_MENUS];

/*
 * will be executed after use answer all the menu questions ...
 *
 * this function will execute the pwm setting gotten from user
 */
int FLAT_FAR menu_set_the_user_information_function(int session_id);
int FLAT_FAR menu_set_the_user_information_function(int session_id)
{
    MENU* menu = &menus[session_id];

    user_answers[session_id] = menu->answer_from_user;

    if ( session_id == MAX_NUMBER_OF_MENUS-1 && menu->answer_from_user == 1) //user said save
the changes
    {
        //16bit or 8bit
        char pwm_mode = PWM_16BIT_OPERATION_MODE;
        if (user_answers[2] == FALSE)
            pwm_mode = PWM_8BIT_OPERATION_MODE;

        char pwm_channel_number = user_answers[3];

        //is high at start of the pulse or is low
        char pwm_polarity = PWM_POLARITY_IS_LOW_AT_BEGINNING_OF_THE_PULSE;
        if (user_answers[4] == TRUE)
            pwm_polarity = PWM_POLARITY_IS_HIGH_AT_BEGINNING_OF_THE_PULSE;

        // is center alined or left alined

```

```

char pwm_alignment = PWM_ALIGNMENT_LEFT_ALIGN;
if (user_answers[5] == TRUE)
    pwm_alignment = PWM_ALIGNMENT_CENTER_ALIGN;

char pwm_is_using_scaled_clock = PWM_IS_USING_SCALED_CLOCK;
if (user_answers[7] == FALSE)
    pwm_is_using_scaled_clock = PWM_IS_NOT_USING_SCALED_CLOCK;

char pwm_main_clock_divider = user_answers[6];
int pwm_scale_clock_divider = user_answers[7];
int pwm_period_reg_value = user_answers[8];
int pwm_duty_reg_value = user_answers[9];

pwm_reset_settings(pwm_channel_number, pwm_mode);

pwm_set_polarity(pwm_channel_number, pwm_mode, pwm_polarity);
pwm_set_alignment(pwm_channel_number, pwm_mode, pwm_alignment);
pwm_set_operation_mode_flags(pwm_channel_number, pwm_mode);
pwm_set_main_clock(pwm_channel_number, pwm_mode, pwm_is_using_scaled_clock);
pwm_set_main_clock_divider(pwm_channel_number, pwm_mode, pwm_main_clock_divider);
pwm_set_scale_clock_divider(pwm_channel_number, pwm_mode, pwm_is_using_scaled_clock,
pwm_scale_clock_divider);
pwm_set_period_and_duty_registers(pwm_channel_number, pwm_mode, pwm_period_reg_value,
pwm_duty_reg_value);

pwm_set_status(pwm_channel_number, pwm_mode, ENABLE);

    //send the changes to pc:
    //16bit, 0, low, left, SA true, 24 MHz, 7,5, 10,2

    if (pwm_mode == PWM_16BIT_OPERATION_MODE) sci0_write_string("16bit, ");
    else sci0_write_string("8bit, ");

    sci0_print_unsigned_integer(pwm_channel_number);
    sci0_write_string(", ");

    if (pwm_polarity == PWM_POLARITY_IS_HIGH_AT_BEGINNING_OF_THE_PULSE)
sci0_write_string("high, ");
    else sci0_write_string("low, ");

    if (pwm_alignment == PWM_ALIGNMENT_CENTER_ALIGN) sci0_write_string("center,
");
    else sci0_write_string("left, ");

    if (pwm_is_using_scaled_clock == PWM_IS_USING_SCALED_CLOCK)
sci0_write_string("SA true, ");
    else sci0_write_string("SA false, ");

    sci0_write_string("24 MHz, ");

    sci0_print_unsigned_integer(pwm_main_clock_divider);
    sci0_write_string(", ");

    sci0_print_unsigned_integer(pwm_scale_clock_divider);
    sci0_write_string(", ");

    sci0_print_unsigned_integer(pwm_period_reg_value);
    sci0_write_string(", ");

    sci0_print_unsigned_integer(pwm_duty_reg_value);

```

```

        sci0_write_string("\r\n");

        enable_and_show_donE_on_7segment();

    }

    return 0;
}

/*
 * init a menu
 */
void FLAT_FAR build_a_menu(int session_id, char* message, int previous_session_id, int
next_session_id, int (*menu_item_function_pointer)(int session_id));
void FLAT_FAR build_a_menu(int session_id, char* message, int previous_session_id, int
next_session_id, int (*menu_item_function_pointer)(int session_id)
{
    MENU* menu = &menus[session_id];
    menu->message = message;
    menu->session_id = session_id;
    menu->previous_session_id = previous_session_id;
    menu->next_session_id = next_session_id;
    menu->answer_from_user = 0;
    menu->menu_item_function_pointer = menu_item_function_pointer;
}

/*
 * init the menu system
 */
void FLAT_FAR build_menus(void);
void FLAT_FAR build_menus(void)
{
    build_a_menu(0, "Please press enter to start ...", 0, 1, &menu_foobar__do_nothing);
    build_a_menu(1, "Please press enter to start ...", 0, 2, &menu_foobar__do_nothing);

    build_a_menu(2, "Should we use a 16BIT PWM channel?" , 1, 3 ,
&menu_set_the_user_information_function);
    build_a_menu(3, "What is the channel number?" , 2, 4 ,
&menu_set_the_user_information_function);
    build_a_menu(4, "Is polarity high at starts?" , 3, 5 ,
&menu_set_the_user_information_function);
    build_a_menu(5, "Is the pulse center aligned?" , 4, 6 ,
&menu_set_the_user_information_function);
    build_a_menu(6, "What is the main clock divider value (0-7)?" , 5, 7 ,
&menu_set_the_user_information_function);
    build_a_menu(7, "What is the scaled clock divider? (say NO if don't need scaled clock or
input 0-255)" , 6, 8 , &menu_set_the_user_information_function);
    build_a_menu(8, "What is the value for period register?" , 7, 9 ,
&menu_set_the_user_information_function);
    build_a_menu(9, "What is the value for duty cycle register?" , 8, MAX_NUMBER_OF_MENUS-1 ,
&menu_set_the_user_information_function);

    build_a_menu(MAX_NUMBER_OF_MENUS-1, "We are done, do you want to save the changes?" , 9, 1 ,
&menu_set_the_user_information_function);
}

/*
 * load the next menu
 *
 * also save the answer from the current menu session for duture use ...
 */

```

```

void FLAT_FAR load_the_next_menu(unsigned int user_answer_from_current_menu);
void FLAT_FAR load_the_next_menu(unsigned int user_answer_from_current_menu)
{
    MENU* menu = &menus[current_menu_session_id];
    MENU* next_menu = &menus[menu->next_session_id];

    menu->answer_from_user = user_answer_from_current_menu;

    lcd_reconfigure_a_lcd_line(LCD_FIRST_LINE, next_menu->session_id, next_menu->message);
    lcd_reconfigure_a_lcd_line(LCD_SECOND_LINE, next_menu->session_id, "INPUT> ");
    lcd_redisplay_line(LCD_FIRST_LINE);
    lcd_redisplay_line(LCD_SECOND_LINE);

    int result_from_menu_item_function_pointer = (* (menu->menu_item_function_pointer) )
(menu->session_id);
    result_from_menu_item_function_pointer = result_from_menu_item_function_pointer +0;

    current_menu_session_id = next_menu->session_id;
}

/*
 * on each menu there are some certain job that will repeat ...
 *
 * Examples:
 * BACK SPACE: If you input something wrong, you like to remove it ...
 * Or hitting enter to going to next menu
 */
void FLAT_FAR menu_helper(int input_from_keypad);
void FLAT_FAR menu_helper(int input_from_keypad)
{
    if (input_from_keypad >=0 )
    {
        //DB12FNP->printf("Input from keypad = %d\r\n", input_from_keypad);

        if (input_from_keypad == 0x0C) //mapped to backsapce - C key
        {
            lcd_remove_from_end_of_line(LCD_SECOND_LINE);
        }
        else if (input_from_keypad == 0x0A) //mapped to YES - A key
        {
            lcd_append_to_end_of_line(LCD_SECOND_LINE, 'Y');
            lcd_append_to_end_of_line(LCD_SECOND_LINE, 'E');
            lcd_append_to_end_of_line(LCD_SECOND_LINE, 'S');
        }
        else if (input_from_keypad == 0x0B) //mapped to NO - B key
        {
            lcd_append_to_end_of_line(LCD_SECOND_LINE, 'N');
            lcd_append_to_end_of_line(LCD_SECOND_LINE, 'O');
        }
        else if (input_from_keypad == 0x0D) //reset to original - D key
        {
            lcd_reset_the_lcd_line_to_original(LCD_SECOND_LINE);
        }
        else if (input_from_keypad == 0x0F) //ENTER - F key
        {
            unsigned int user_answer_from_current_menu =
string_to_int_atoi(lcd_get_the_appended_string_on_lcd_line(LCD_SECOND_LINE));
            load_the_next_menu(user_answer_from_current_menu);
        }
        else if (input_from_keypad < 10)
        {

```

```

        lcd_append_to_end_of_line(LCD_SECOND_LINE, '0' + input_from_keypad);
    }

}

}

/*****MENU LIST ENDS*****/
/*****JOB LIST STARTS*****/

/*
 * how many jobs we have?
 * remeber, if you define a job and make the function, you also
 * need to increase this number
 */
#define JOB_LIST_SIZE 5

/**
 * A periodic job will have the followings:
 * name (i.e. job0-pwm)
 * its period (i.e. every 30ms)
 * a counter to check the time
 * a pointer to a function related to this job
 * functions should ONLY be "void function(void);"
 *
 * I will avoid a linklist style jobs ... just to make the code smaller and
 * easier to follow
 */
typedef struct job_struct
{
    unsigned char* name;
    unsigned int period;
    unsigned int counter;
    void (*job_function_pointer)(void);
}
JOB;

int job_list_size = JOB_LIST_SIZE;
JOB job_list [JOB_LIST_SIZE];

/**
 * init the job list and set all the pointers and vars to 0
 */
void FLAT_FAR init_the_jobs(JOB job_list[] , int job_list_size);
void FLAT_FAR init_the_jobs(JOB job_list[] , int job_list_size)
{
    JOB temp_job = {0,0,0,0};
    int i = 0;
    for (i = 0; i < job_list_size; i++ )
    {
        job_list[i] = temp_job;
    }
}

/**
 * allocated the memory for each variable in the JOB struct and assign given values
 */
void FLAT_FAR malloc_this_job(JOB* job, unsigned char* name, unsigned int period, unsigned
int counter, void (*job_function_pointer)(void));
void FLAT_FAR malloc_this_job(JOB* job, unsigned char* name, unsigned int period,

```

```

unsigned int counter, void (*job_function_pointer)(void))
{
    job->name = name;
    job->period = period;
    job->counter = counter;
    job->job_function_pointer = job_function_pointer;
}

/**
 * helper function for "execute_the_jobs"
 */
void FLAT_FAR execute_a_job(JOB* job);
void FLAT_FAR execute_a_job(JOB* job)
{
    if (job->counter == job->period)
    {
        job->counter = 0;
        (* (job->job_function_pointer) )();
    }
    else
    {
        job->counter = job->counter + 1;
    }
}

/**
 * execute the whole job list IF possible
 * a job will be executed if its period has been reached
 */
void FLAT_FAR execute_the_jobs(JOB job_list[] , int job_list_size);
void FLAT_FAR execute_the_jobs(JOB job_list[] , int job_list_size)
{
    int i = 0;
    for (i = 0; i < job_list_size; i++ )
    {
        execute_a_job(&job_list[i]);
    }
}

void FLAT_FAR job0_function(void);
void FLAT_FAR job0_function(void)
{
}

void FLAT_FAR job1_function(void);
void FLAT_FAR job1_function(void)
{
    menu_helper(ScanKeyPad());
}

void FLAT_FAR job2_function(void);
void FLAT_FAR job2_function(void)

```

```

{
    lcd_scroll(LCD_FIRST_LINE);
}

void FLAT_FAR job3_function(void);
void FLAT_FAR job3_function(void)
{
    display_donE_on_7segmnet_for_few_momnets__if_possible();
}

void FLAT_FAR job4_function(void);
void FLAT_FAR job4_function(void)
{
    DDRB = 0xff;
    if (PORTB == 7) PORTB = 0;
    else PORTB = 0x07;
    sci0_write_string("JOB4 running ...\\r\\n");
}

/*****JOB LIST DONE*****/
/*****RTI STARTS*****/
/*
 * here are the methods and variables for real-time-interrupt
 */
volatile unsigned int counter_for_real_time_interrupt;
unsigned int counter_for_real_time_interrupt_limit;

/**
 * when a real-time interrupt happens, HC12 will go through this method
 */
void INTERRUPT rti_isr( void );
void INTERRUPT rti_isr( void )
{
    //clear the RTI - don't block the other interrupts
    CRGFLG = 0x80;

    //for instance if limit is "10", every 10 intrupts do something ...
    if (counter_for_real_time_interrupt == counter_for_real_time_interrupt_limit)
    {
        //reset the counter
        counter_for_real_time_interrupt = 0;

        //do some work
        execute_the_jobs(job_list , job_list_size);
    }
    else
        counter_for_real_time_interrupt ++;
}

/**
 * initalize the rti: rti_ctl_value will set the prescaler ...
 */
void FLAT_FAR rti_init(unsigned char rti_ctl_value, unsigned int counter_limit);
void FLAT_FAR rti_init(unsigned char rti_ctl_value, unsigned int counter_limit)
{
    /**
     * set the maximum lmit for the counter:
     * if max set to be 10, every 10 intrupts some work will be done
     */
}

```

```

counter_for_real_time_interrupt_limit = counter_limit;

/**
 * RTICTL can be calculated like:
 * i.e: RTICTL == 0x63 == set rate to 32.768ms:
 * The clock divider is set in register RTICTL and is:  $(N+1) * 2^{(M+9)}$ ,
 * where N is the bit field RTR3 through RTR0
 * and M is the bit field RTR6 through RTR4.
 * 0110 0011 = 0x63 ==>  $1 / (4\text{MHz} / 4 * 2^{15}) == 4 * 2^{15} / 4,000 = 32.768\text{ms}$ 
 * 0111 1111 = 0x7F ==>  $1 / (4\text{MHz} / 16 * 2^{16}) == 2^{20} / 4,000 = 262.144\text{ms}$ 
 */
RTICTL = rti_ctl_value;

// How many times we had RTI intruppts
counter_for_real_time_interrupt = 0;

// Enable RTI interrupts
CRGINT |= 0x80;
// Clear RTI Flag
CRGFLG = 0x80;
}
/*****RTI DONE*****/

int main ( void )
{
    //init real time intruppts
    rti_init(0x10, 1);

    // init job lists
    init_the_jobs(job_list , job_list_size);
    malloc_this_job(&job_list[0], "job0", 1, 0, &job0_function);
    malloc_this_job(&job_list[1], "job1", 100, 0, &job1_function); //scan keypad
    malloc_this_job(&job_list[2], "job2", 400, 0, &job2_function); //lcd scroll
    malloc_this_job(&job_list[3], "job3", 800, 0, &job3_function); //display done messge if it
    // is possible
    malloc_this_job(&job_list[4], "job4", 2000, 0, &job4_function); //on and off PORTB

    // init serail ports
    sci0_init(BAUD_115200);
    sci1_init(BAUD_115200);

    // init lcd
    lcd_init();

    // build the menus and load menu 0 to lcd
    build_menus();
    load_the_next_menu(0);

    enable_interrupts();

    sci0_write_string("=== STARTING ===\r\n");

    // get the commands coming from PC and process them
    get_commands_from_pc();

    // never should reach here ...
    return 0;
}

```