EE128: Data Acquisition, Instrumentation and Process Control
University of California, Riverside

Lab 2: Simple Parallel I/O

Fall 2017

Objective

To become familiar with a development environment for 9S12DG256-based systems; to learn hardware/software techniques for developing, testing and debugging microcontroller-based digital systems.

References

- Dragon12-JR Trainer User's Manual
- Freescale MC9S12DG256 Device User Guide
- Freescale HCS12 Core User Guide
- Reference Guide for D–Bug12 Version 4.x.x

Equipment

- PC running MS Windows
- Digital Multi-Meter (DMM)
- Dragon12-JR 9S12DG256 evaluation board

Parts

- 2 each 8-bit LED bars
- 1 each 4-bit DIP switch (piano type)
- 2 each 470Ω, 10-pin bussed resistor network (SIP)

Software

- Freescale CodeWarrior for HC12 v5.1 (C cross compile & programming environment)
- RealTerm (terminal emulation program)

Lab Computing Environment

Freescale CodeWarrior and RealTerm are installed on all the EE128 computers. Reference documents are available at the course website.

You should create any project on your ECE account drive (aka "Z drive"). By doing so, your files are backed up automatically. DO NOT PLACE YOUR PROJECT FOLDERS ON THE C: DRIVE. Files left on the C drive may be removed at any time.

Background

Dragon12-JR 9S12DG256 Evaluation Board

The Dragon12-JR board is a low-cost, entry-level evaluation board (EVB) that allows for complete access to the 9S12DG256 microcontroller signals. The board allows for all four modes of operation (e.g., EVB, Jump-to-EEPROM, etc.) through jumper settings. For this lab, the jumpers should be set up for EVB. The board also has the DBug-12 monitor program installed in flash memory, allowing for communication between the microcontroller and the development PC. Applications for single chip operation can be downloaded into the internal static RAM or flash memory.

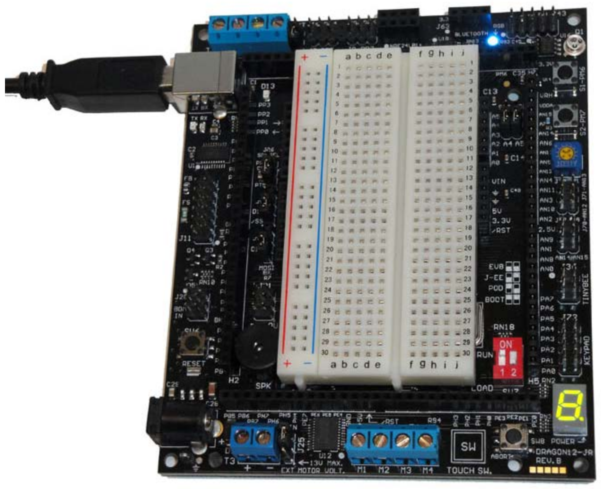


Figure 1. Dragon12-JR 9S12DG256 EVB

Board Features (see the EVB manual for detailed information)

- Motorola 9S12DG256 microcontroller
- 256K Byte Flash Memory, 12K Byte RAM and 4K byte EEPROM
- 3 SPIs, 2 SCIs, 3 CANs, I2C interface, 8 16-bit timers, 8 PWMs and 16-channel 10-bit A/D converter
- Bus speed up to 25 MHz
- 112 Pins, up to 91 I/O-Pins
- 16 MHz crystal, 8 MHz default bus speed and up to 25MHz bus speed via PLL
- Four operating modes: EVB, Jump-to-EEPROM, POD and Boot loader
- On-board USB interface based on the FT231XS for programming and debugging code
- Automatic sensing/switching circuit to select power input from USB or AC adapter
- LED operating mode indication (E,J,P,B) on the 7-segment display during power-up
- 3 digit diagnostic code on the 7-segment LED during power up to aid troubleshooting.
- On-board BDM-in connector to be connected with a BDM from multiple venders for debugging.
- 2-position DIP switch for selecting one of 4 operating modes with D-Bug12 monitor or RUN/LOAD mode with Serial Monitor
- 2 user pushbutton switches, Reset and program abort buttons
- Light and temperature sensors, Potentiometer trimmer pot for analog input, Speaker
- Small breadboard area

In this lab, we will primarily use Ports A and B.

DBug-12 Monitor

A program named DBug-12 is pre-installed in the 9S12DG256 flash memory, which occupies the very low end of the memory. Normally, the system boots off this program after reset. DBug-12 is a monitor, which is a primitive version of a simple operating system. It can support many basic operations for embedded system development. You should know at least the following commands:

```
>    md <start-addr> [<end-addr>]
```

The md command displays a block of EVB memory beginning at <start-addr> and continuing to <end-addr>. The <start-addr> parameter must be supplied, however, the <end-addr> parameter is optional. 9 lines of 16 bytes are displayed if one or more addresses are omitted.

```
>    mm <addr>
```

This command allows the user to examine/modify contents of EVB memory at specific locations in an interactive manner.

```
>    bf <start-addr> <end-addr> <data>
```

This command allows user to fill a block of memory (from <start-addr> to <end-addr>) with a specific byte given in <data>.

```
>    load
```

This command allows downloading, from a host computer, a user program in Motorola S-record format into the EVB through its terminal port.

```
>    call <addr>
```

The call command executes a user subroutine starting from the given <addr>.

```
>    g <addr>
```

This command allows execution of a program starting from address <addr>. For the EVB, a user program is typically loaded at address 0x2000.

```
>    help
```

This command displays all commands for quick reference.

Detailed information on DBug-12, including its source code, can be found in the DBug-12 manual.

Freescale CodeWarrior

Freescale CodeWarrior IDE (integrated development environment) will be used to edit, compile, link and simulate your programs during the development of a 9S12DG256-based digital system.

RealTerm

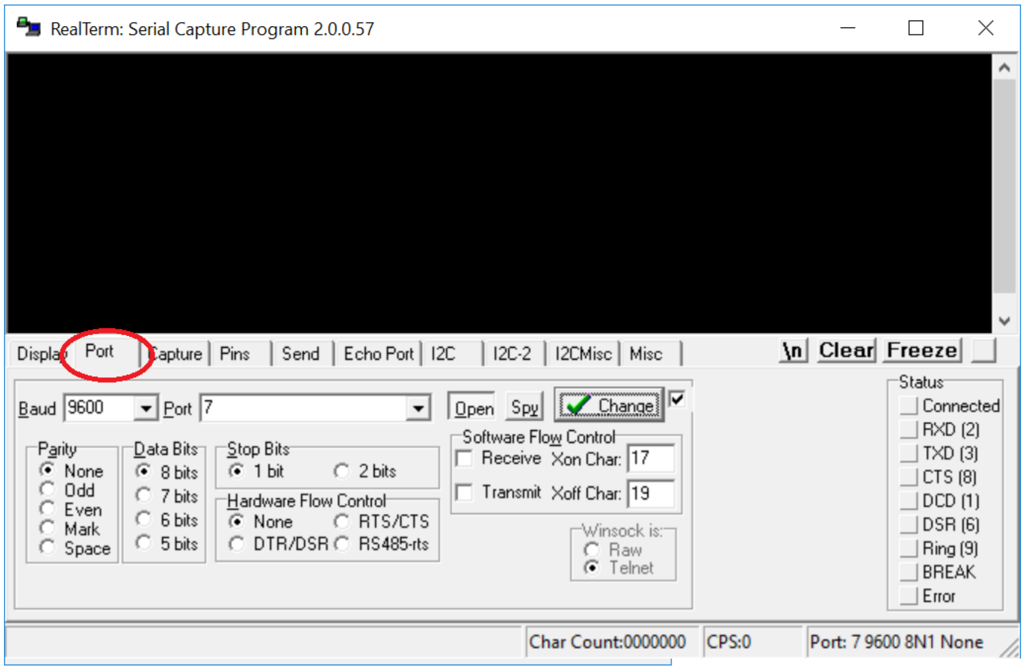
RealTerm will be used as a terminal emulation program to interact with the DBug-12 monitor of the EVB. It will also be used to download user programs in Motorola S-record format to the EVB.

PART 1.

Part 1 is a step-by-step tutorial to learn how to (i) connect the Dragon12-JR evaluation board (EVB) to the development PC, (ii) use RealTerm to interact with the EVB, and (iii) develop a program in Freescale CodeWarrior. The lab report is *not* required for Part 1. However, you should complete all the steps below before starting Part 2.

Procedures

1. Make sure that the **two DIP switches of SW7** of the EVB are set in the "**low**" positions. This is for the EVB to operate in "EVB mode".
2. Connect the EVB and the development PC with a **USB cable**. This USB connection powers up the EVB, so the external AC adapter is not needed most of the time. Note that some applications (e.g., motor control) may need more power, and in such a case, an AC power adapter or external power supply needs to be used.
3. Whenever the EVB is powered up (or reset by the reset button of the board), the speaker should chirp once and the 7-segment LED should display the letter "**E**" momentarily. If not, check the DIP switches again.
4. The Dragon12-JR board uses a USB-to-Serial interface. In other words, the USB connection will be recognized as a COM port on the development PC. Find out which COM port number is assigned to the USB connection (Control Panel > Systems > Device Manager > Ports).
5. Launch **RealTerm**, and click on the **Port** tab (circled in RED in the figure below).



Baud 9600 Port 7

Parity: [✓ None] Odd Even Mark Space

Data Bits: [✓ 8 bits] 7 bits 6 bits 5 bits

Stop Bits: [✓ 1 bit] 2 bits

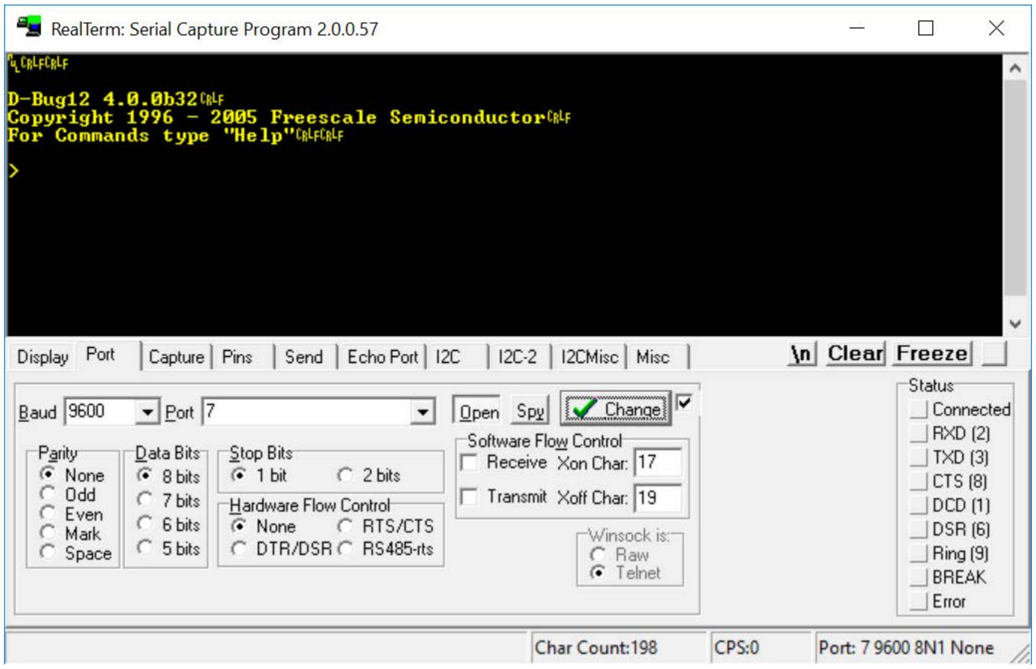
Hardware Flow Control: [✓ None] RTS/CTS DTR/DSR RS485-rts

Software Flow Control: Receive Xon Char: 17 Transmit Xoff Char: 19

Char Count:0000000 CPS:0 Port: 7 9600 8N1 None

Set the **Baud** rate to **9600** and the **Port** number to the COM port number found in Step 4 (e.g., COM1 as in the figure). Make the rest of your settings identical to those shown in the figure (Data bits: **8**, Stop bits: **1**, Parity and Flow control: **None**). Then, click the **Change** button to apply the changes you made.

6. Your EVB is preconfigured with a boot loader and D-BUG12. When you reset the board, you should see:



RealTerm: Serial Capture Program 2.0.0.57

```
CRLFCRLF

D-Bug12 4.0.0b32CRLF
Copyright 1996 - 2005 Freescale SemiconductorCRLF
For Commands type "Help"CRLFCRLF

>
```

Baud 9600 Port 7

Parity: [✓ None] Odd Even Mark Space

Data Bits: [✓ 8 bits] 7 bits 6 bits 5 bits

Stop Bits: [✓ 1 bit] 2 bits

Hardware Flow Control: [✓ None] RTS/CTS DTR/DSR RS485-rts

Software Flow Control: Receive Xon Char: 17 Transmit Xoff Char: 19

Winsock is: Raw [✓ Telnet]

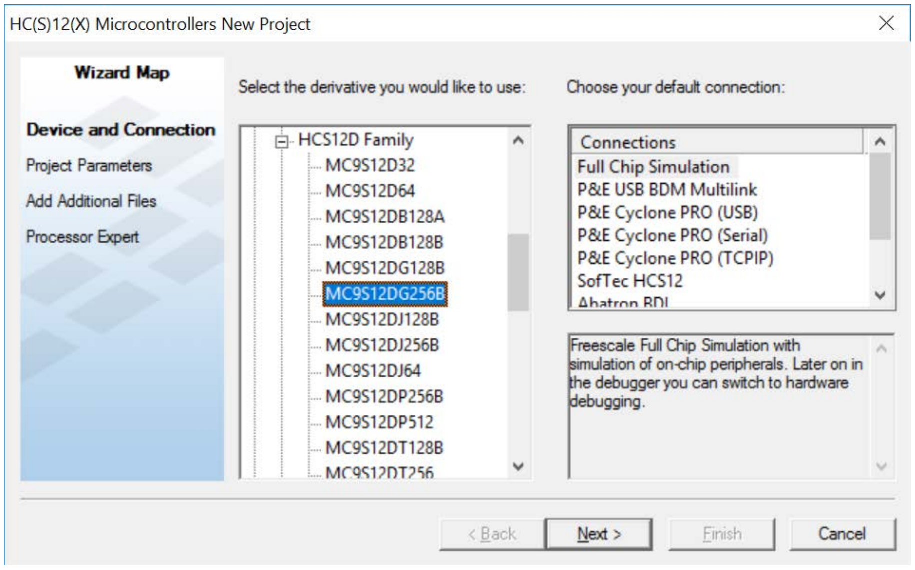
Status: Connected RXD (2) TXD (3) CTS (8) DCD (1) DSR (6) Ring (9) BREAK Error

Char Count:198 CPS:0 Port: 7 9600 8N1 None

7. Type "help" then <enter> will display a list of available commands.

   Hitting any other key will display the remainder of the commands. For further information on the DBug-12 monitor program, refer to the Background section of this handout or the DBug-12 reference document.

8. Launch the **CodeWarrior IDE** and click **Create New Project** in the initial popup window, or click **File > New Project**. Then, you will see a New Project wizard like the figure below.



HC(S)12(X) Microcontrollers New Project

Select the derivative you would like to use:

- HCS12D Family
  - MC9S12D32
  - MC9S12D64
  - MC9S12DB128A
  - MC9S12DB128B
  - MC9S12DG128B
  - **MC9S12DG256B**
  - MC9S12DJ128B
  - MC9S12DJ256B
  - MC9S12DJ64
  - MC9S12DP256B
  - MC9S12DP512
  - MC9S12DT128B
  - MC9S12DT256

Choose your default connection:

Connections
- **Full Chip Simulation**
- P&E USB BDM Multilink
- P&E Cyclone PRO (USB)
- P&E Cyclone PRO (Serial)
- P&E Cyclone PRO (TCPIP)
- SofTec HCS12
- Abatron BDI

Freescale Full Chip Simulation with simulation of on-chip peripherals. Later on in the debugger you can switch to hardware debugging.

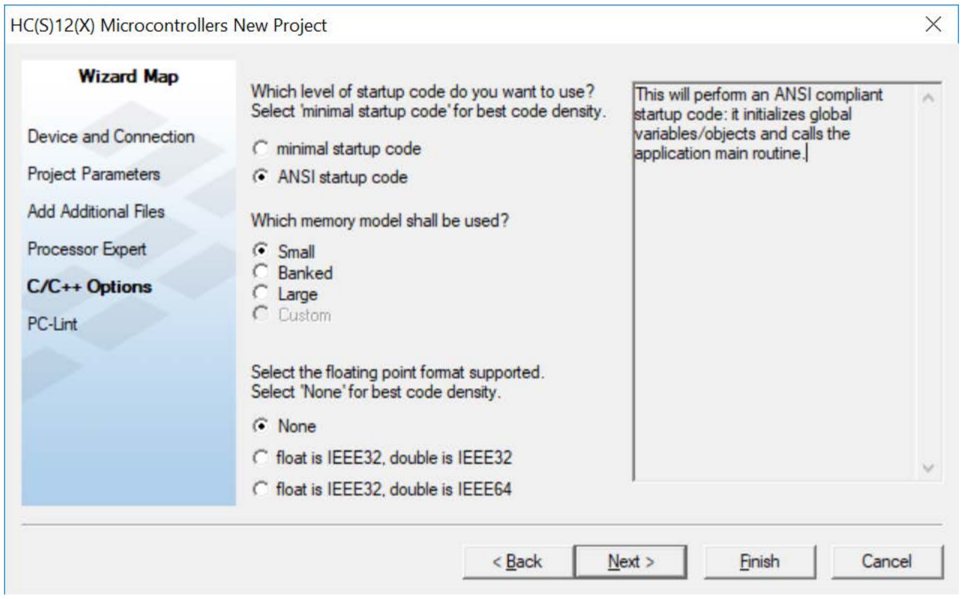
Choose **HCS12 > HCS12D Family > MC9S12DG256B** and click **Next**.

9. In the **Project Parameters** tab, make sure to select **"C"** for your programming language. Set your project name (e.g., Part1.mcp) and location (e.g., Z:\EE128\Lab2\) appropriately. Then, click **Next.**

10. There is nothing to set in the **Add Additional Files** tab. Click **Next**.

11. In the **Processor Expert** tab, check **None** and click **Next**.

12. In the **C/C++ Options** tab, check **ANSI startup code, Small** memory model, and **None** for floating point. Then, click **Finish**.



Which level of startup code do you want to use?
Select 'minimal startup code' for best code density.

minimal startup code [✓ ANSI startup code]

Which memory model shall be used?

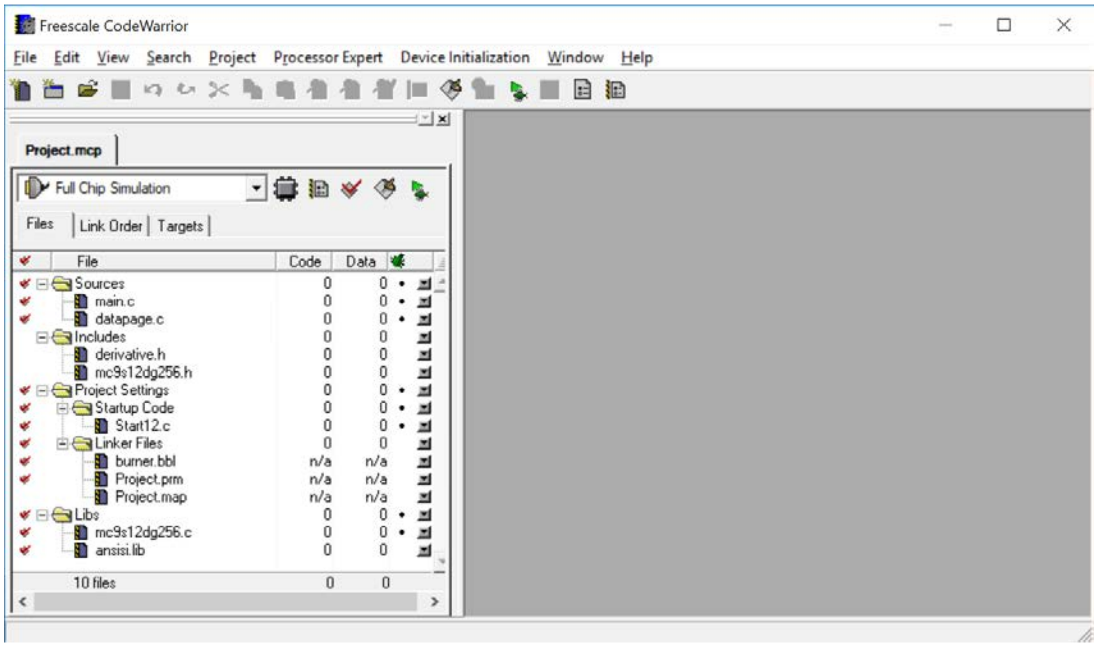
[✓ Small] Banked Large Custom

Select the floating point format supported.
Select 'None' for best code density.

[✓ None] float is IEEE32, double is IEEE32 float is IEEE32, double is IEEE64

This will perform an ANSI compliant startup code: it initializes global variables/objects and calls the application main routine.

13. Your project is now created. You will see a project window. There are several folders shown here, such as Sources, Includes, Project Settings. In this project window, you can add and remove files in their respective folders.


Freescale CodeWarrior
File Edit View Search Project Processor Expert Device Initialization Window Help
Project.mcp
Full Chip Simulation
Files Link Order Targets

File	Code	Data
Sources	0	0
main.c	0	0
datapage.c	0	0
Includes	0	0
derivative.h	0	0
mc9s12dg256.h	0	0
Project Settings	0	0
Startup Code	0	0
Start12.c	0	0
Linker Files	0	0
burner.bbl	n/a	n/a
Project.prm	n/a	n/a
Project.map	n/a	n/a
Libs	0	0
mc9s12dg256.c	0	0
ansisi.lib	0	0
10 files	0	0

14. Open "**main.c**" from the project window. Modify the source code of main.c as follows:

```
#include <hidef.h>      /* common defines and macros */
#include <mc9s12dg256.h>

void main(void) {
  /* put your own code here */
  unsigned long i;

  DDRH=0xff;
  PTH=0xff;

  for(;;) {
    for (i=0;i<150000;i++);
    if (PTH == 0xff) PTH = 0;
    else PTH = 0xff;
  } /* loop forever */

  /* please make sure that you never leave main */
}
```

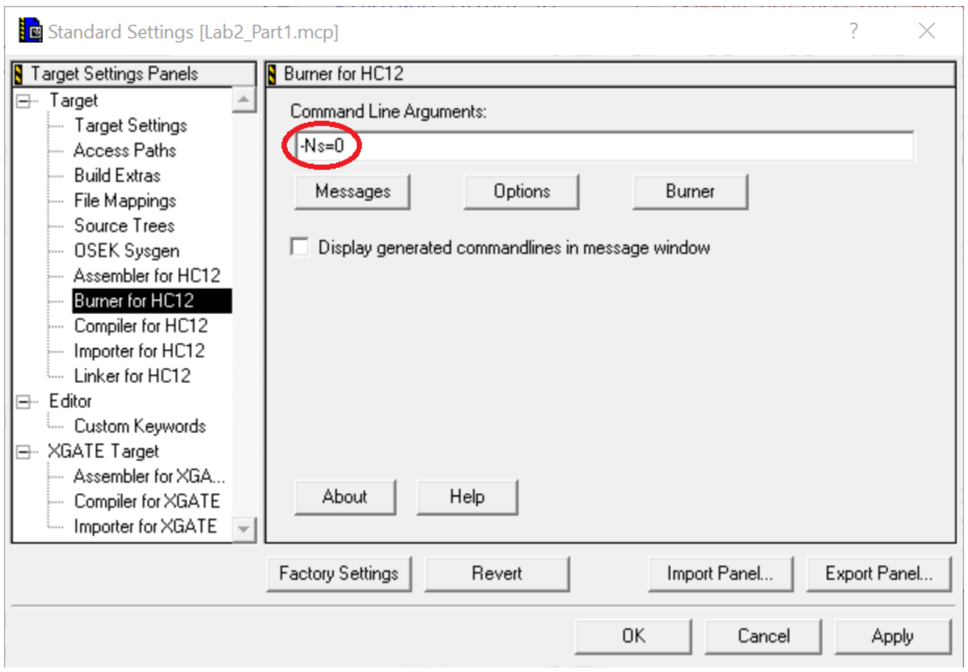
This code blinks the 7-segment LED of the EVB.

Click the **Make** button ( ) of the project window, or **Project** > **Make** to build the code. If there is any error, check if your source code has any typo.

If successful, it will create the executable as an **s19 file** (**<your_project>\bin\Project.abs.s19**), which represents your program in HEX following the Motorola S-record format.

15. The s19 file is the one to be downloaded to your EVB. Since we will use D-BUG12 to download the program, we need to make sure the s19 file is in a form compatible with D-BUG12.

Click the **Standard Settings** button ( ) of the project window, and select the "**Burner for HC12**" option under the "Target" tree. Then, type "**-Ns=0**" in the Command Line Arguments, as shown in the following figure:



Burner for HC12

Command Line Arguments: -Ns=0

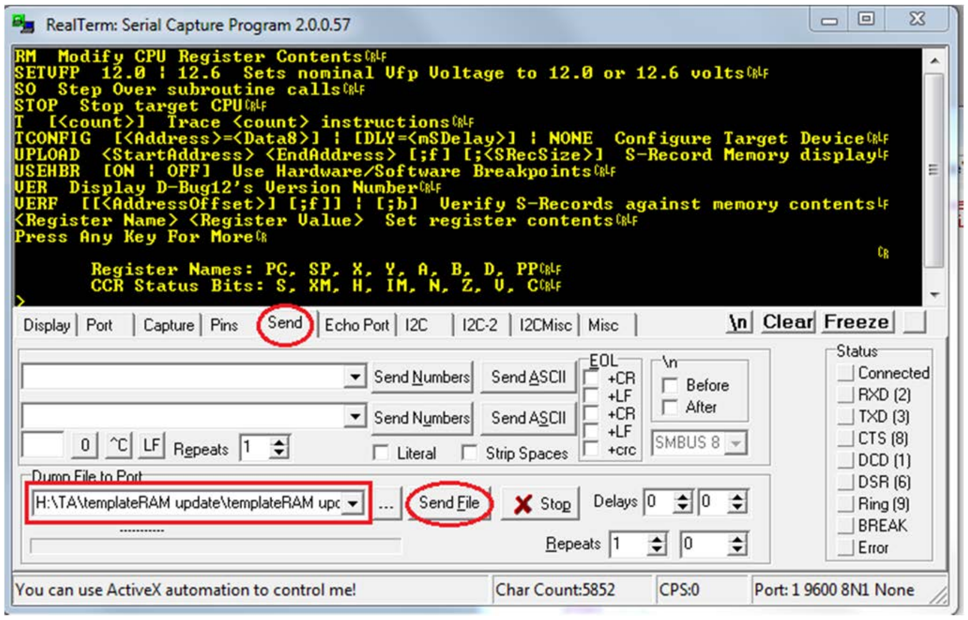
☐ Display generated commandlines in message window

16. We will test this program by loading it into RAM of the EVB, because it is faster and more convenient than programming the flash memory. To do so, the linker parameters of this project must be changed. In the project window, open the **"Project.prm"** file in the **Project Settings > Linker Files** folder. Delete all the code from Project.prm, and copy and paste the below code:

```
NAMES END

SECTIONS
    MY_RAM = READ_WRITE 0x1000 TO 0x1FFF; /* 4095 bytes of data */
    MY_PSEUDO_ROM = READ_ONLY 0x2000 TO 0x3FFF; /* 8191 bytes of code */
END

PLACEMENT
    _PRESTART,
    STARTUP,
    ROM_VAR,
    STRINGS,
    DEFAULT_ROM, NON_BANKED,
    COPY INTO MY_PSEUDO_ROM;
    DEFAULT_RAM INTO MY_RAM;
END

STACKSIZE 0x100
```

Build the code again to apply the linker parameter change.

17. Now let's test your program on the EVB. Switch back to the **RealTerm** window and type "**load**" at the command prompt. Then, click the **Send** tab which is circled in RED as shown in the following figure.


RealTerm: Serial Capture Program 2.0.0.57

```
RM  Modify CPU Register Contents
SETVFP  12.0 | 12.6  Sets nominal Vfp Voltage to 12.0 or 12.6 volts
SO  Step Over subroutine calls
STOP  Stop target CPU
T  [<count>]  Trace <count> instructions
TCONFIG  [<Address>=<Data8>] | [DLY=<mSDelay>] | NONE  Configure Target Device
UPLOAD  <StartAddress> <EndAddress> [;f] [;<SRecSize>]  S-Record Memory display
USEHBR  [ON | OFF]  Use Hardware/Software Breakpoints
VER  Display D-Bug12's Version Number
VERF  [[<AddressOffset>] [;f]] | [;b]  Verify S-Records against memory contents
<Register Name> <Register Value>  Set register contents
Press Any Key For More

        Register Names: PC, SP, X, Y, A, B, D, PP
        CCR Status Bits: S, XM, H, IM, N, Z, V, C
>
```

Dump File to Port: H:\TA\templateRAM update\templateRAM upc

Delays: 0 0

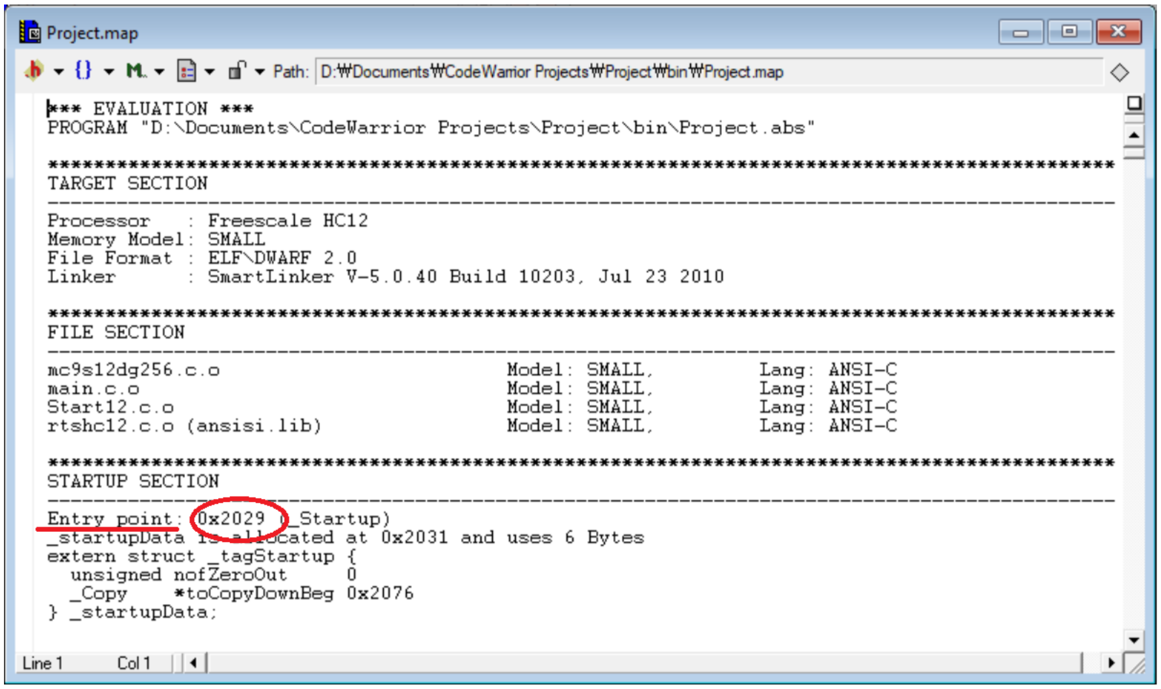
Repeats: 1 0

You can use ActiveX automation to control me! | Char Count:5852 | CPS:0 | Port: 1 9600 8N1 None

We will then select the **s19 file** (<your_project >\bin\ Project.abs.s19) and transmit it to the EVB by clicking the **Send File** button.

18. We have loaded the program into RAM. How do we execute it? We know that "g" is the DBug-12 command to begin execution at a specific address. This means that, if we know the start address of our program code, we can execute it.

From your project window in CodeWarrior, open the **Program.map** file located in **Project Settings > Linker Files** folder.



Project.map
Path: D:₩Documents₩CodeWarrior Projects₩Project₩bin₩Project.map
Line 1
Col 1

```
*** EVALUATION ***
PROGRAM "D:\Documents\CodeWarrior Projects\Project\bin\Project.abs"

*********************************************************************************************
TARGET SECTION
---------------------------------------------------------------------------------------------
Processor   : Freescale HC12
Memory Model: SMALL
File Format : ELF\DWARF 2.0
Linker      : SmartLinker V-5.0.40 Build 10203, Jul 23 2010

*********************************************************************************************
FILE SECTION
---------------------------------------------------------------------------------------------
mc9s12dg256.c.o                         Model: SMALL,         Lang: ANSI-C
main.c.o                                Model: SMALL,         Lang: ANSI-C
Start12.c.o                             Model: SMALL,         Lang: ANSI-C
rtshc12.c.o (ansisi.lib)                Model: SMALL,         Lang: ANSI-C

*********************************************************************************************
STARTUP SECTION
---------------------------------------------------------------------------------------------
Entry point: 0x2029 (_Startup)
_startupData is allocated at 0x2031 and uses 6 Bytes
extern struct _tagStartup {
  unsigned nofZeroOut     0
  _Copy    *toCopyDownBeg 0x2076
} _startupData;
```

Bring your attention to the "STARTUP SECTION". You will notice that the Entry point of your program is 0x2029.

Switch back to RealTerm and type “g 2029”. Then, your program will start properly, and the 7-segment LED of your EVB will start blinking.

These are the basics for creating and downloading a program. There are many more features in the CodeWarrior IDE, such as debugging tools, adding libraries, etc. For further details on this, please refer to the on-line CodeWarrior help manual.

PART 2.

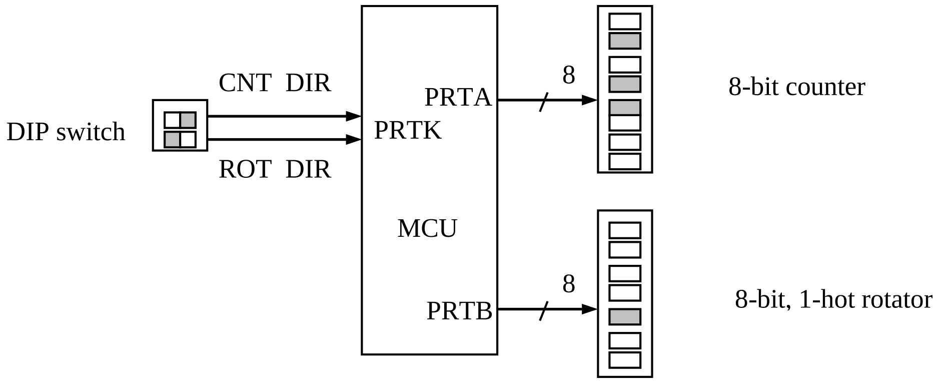
In this experiment, we design and implement a 9S12DG256-based digital system that contains

- an 8-bit, bi-directional, binary counter; and
- an 8-bit, bi-directional, one-hot rotator.

Both run at approximately 1 Hz by software-generated delay. The counting direction (upward or downward) for the counter, and the rotating direction (left or right) for the rotator, are given respectively by logic signals provided by a 2-bit DIP switch:

- CNT_DIR: 0 for upward, and 1 for downward;
- ROT_DIR: 0 for left, and 1 for right.

The state of the counter, and that of the rotator are displayed with two 8-bit LED bars.



Design Considerations

Resource Allocation

It is recommended that you use Port A as the counter, and Port B as the rotator. Port K can be used for signals CNT_DIR, and ROT_DIR, respectively. Make sure that you configure the ports for general purpose I/O and that you get the data directions correct (e.g., Port A should be configured as an 8-bit output port)

Programming

Symbolic names should be used wherever possible to improve your program's readability. Portability should also be taken into account. The following symbolic names should be used.

- PRTA, PRTB, PRTK
- CNT_DIR, ROT_DIR

For your convenience, use the 9S12DG256 include file (mc9s12dg256.h) which defines symbolic names for all registers and ports. You can reuse the Part 1 project file(s).

Make sure your program is accurate and concise, and is an implementation of the following pseudo-code:

```
Configure Port A to be an 8-bit output port;
Configure Port B to be an 8-bit output port;

Initialize Port A to 0;
Initialize Port B such that only 1 bit is ON;

while (1) {
      delay for the effect of soft clock;

      read Port K;

      if (CNT_DIR is 0)
            increment Port A;
      else
            decrement Port A;

      if (ROT_DIR is 0)
            left-rotate Port B;
      else
            right-rotate Port B;
}
```

Pre-lab

Make sure you know how to connect an LED bar with a bussed current limiting resistor network; and how to connect a DIP switch to provide a valid logic signal. Use Digital Multi-Meter whenever needed.

Detailed schematic drawings using given parts are required (use Cadence-Capture CIS).

Procedures

1. Build the circuit of the system. Make sure all connections are correct.
2. Compile and link your program to generate the required Motorola S-record file (s19 file).
3. Run the terminal emulator program (RealTerm) on the PC. Make sure that the COM port parameters are set correctly.
4. Turn on the power of the EVB, which also provides power for your peripheral circuit (LEDs, DIP switches, etc.). DBug-12 should start and prompt for a command.
5. Load your S-record file into RAM:
   - first using DBug-12 command "load";
   - then transmit the S-record file to the EVB through the com port.
6. Verify that your program is indeed loaded at the specified location in RAM by using the DBug-12 command "md".
7. Run your program by commanding DBug-12 using the "g" command. If it does not seem to work, debug hardware/software, and repeat from Step 1.

   Possible errors include a wrong connection on LEDS and/or DIP switches, reversed polarity on LED bars, etc.

   You may have to tweak the delay time in your program to obtain a "clock" of approximately 1 Hz.
8. Using your implemented system, find **answers** to the questions shown below.

9. When all the experiments are done, **<u>demo</u>** your system to the TA, with a brief overview of your circuit and source code

Questions

1. 1) What is the size of your S-record file? 2) What is the *exact size* of your loaded program? 3) Why are they different? (You may have to use DBug-12 to find it out)

2. Do the counter and rotator change their respective states at the same time? Justify your answer with *experimental evidence* – you may wish to use a logic analyzer to find it out.

3. When the software delay is removed, how fast can your program run -- in terms of the frequency in which the counter and rotator are updated? Answer with *logical or experimental evidence*.

Report

Make sure you include the following in your report:

- A short abstract (one paragraph stating the objectives and accomplishments)
- Experiment system specification (what is to be designed and implemented)
- Block diagram and hardware design
- Detailed schematic diagram
- High level description of the software
- Program listing (including comments)
- Technical problems encountered, and how they are solved
- Answers to the questions
- A very brief conclusion