ENGG364: Microcomputer Interfacing
Lab 0: M68HC12A Evaluation Board and Software

School of Engineering,University of Guelph
Fall 2009

Start Date: *Thur (Week #2)*
Due Date: *No Report is Required*

1 Objectives:

- Learn how to configure and setup the Evaluation Board.
- Understand the usage of the D-Bug12 commands.

2 Introduction

The M68HC12A4EVB[1] is an economical tool for designing and debugging code for, and evaluating the operation of, the MC68HC12 MCU family. By providing the essential MCU timing and I/O circuitry, the EVB simplifies user evaluation of prototype hardware and software.

2.1 Startup

The following startup procedure includes a checklist of configuration and setup items. To begin operating the M68HC12A4EVB, follow these steps:

1. Connect the EVB to the external power supply.
2. Connect the EVB to the PC using COM1.
3. Start MiniIDE
4. Configure the terminal communication interface to use COM1 and 9600 baud rate (terminal → options, terminal tab)
5. Check terminal → connected.
6. Reset the EVB by pressing and releasing the on-board reset switch.
7. Please check the "MiniIDE Tutorial" on the web for further detailed instructions.

Since the EVB is configured to execute D-Bug12 upon reset you will see the following message:

```
 >D-Bug12 v1.0.4
Copyright 1995 - 1996 Motorola Semiconductor
For Commands type "Help"
```

If the prompt does not appear, check all connections and verify that startup steps 1 through 5 above have been performed correctly. When the prompt appears, D-Bug12 is ready to accept commands from the terminal.

[1]All information in this document are borrowed from the 68HC12 Evaluation Board User's Manual by Motorola

2.2 Using D-Bug12 Commands

D-Bug12, the EVB's firmware-resident monitor program, provides a self-contained operating environment that allows writing, evaluation, and debugging of user programs.

Commands are typed on the terminal's D-Bug12 prompt line and executed when the carriage return (ENTER) key is pressed. D-Bug12 then displays either the appropriate response to the command or an error indication.

The D-Bug12 command-line prompt is the greater sign (>). Table 1 in Appendix A summarizes the D-Bug12 commands. In the following section we will just look at a few of these commands[2]. **Please check**:

1. The tutorial on the "MiniIDE" in Lab0 Section on the web for more information on how to utilize the D-Bug12 commands.

2. Your text book Section 3.6 on page 84 for using the D-Bug12 commands.

2.2.1 LOAD "Load S-Record File"

The LOAD command is used to load S-Record object files into memory from an external device. The address offset, if supplied is added to the load address of each S-Record before its data bytes are placed in memory.
syntax

```
Load [<AddressOffset>]
```

To load a program, type "load" at the prompt, then press the download button in the MiniIDE GUI and select your program s-record (*.s19)

```
> LOAD 1000
  **********************
>
```

2.2.2 G "Go Execute a User Program"

The G command is used to begin the execution of user code. The program will begin execution at the address supplied. The location of the program in memory can be set using assembler directive (e.g. ORG). Execution of the user program continues until a user breakpoint is encountered, a CPU exception occurs, or the EVB's reset or program-abort switch is pressed.
syntax

```
G  [<Address>]
```

The following example will illustrate the GO operation.

```
> G 800
User Breakpoint  Encountered

PC      SP     X     Y     D = A:B      CCR= SXHI NZVC
0820   0A00   057C  0000     00:00           1001 0000
0820   08           INX
>
```

[2]Appendix B gives some more examples

2.2.3 MD "Memory Display"

The memory Display command displays the contents of memory as both hexadecimal bytes and ASCII characters, 16-bytes on each line. The $< StartAddress >$ parameter must be supplied, the $< EndAddress >$ parameter is optional.
syntax

```
MD  <StartAddress> [<EndAddress>]
```

2.2.4 MM "Memory Modify"

This command allows us to examine and modify the contents of memory location one byte at a time.
syntax

```
MM  <StartAddress> [<Data>]
```

If no optional data is provided, then D-bug12 enters the *"interactive memory modify mode"*. In this mode, each byte is displayed on a separate line following the address of data. Once the memory modify command has been entered, **single-character** sub-commands are used for the modification and verification of memory contents. These sub-commands have the following format:

- $[< Data >] < CR >$: Optionally update current location and display the next loc.
- $[< Data >]$ <=>: Optionally update current location and redisplay the same loc.
- $[< Data >] < - >$: Optionally update current location and display the previous loc.
- $[< Data >] < . >$: Optionally update current location and exit Memory Modify.

2.2.5 RD "Register Display"

The Register Display command is used to display the CPU12's registers.
syntax

```
RD
```

The following example will illustrate the RD operation.

```
> RD
PC     SP     X     Y         D = A:B     CCR  =  SXHI  NZVC
0206   03FF   1000  3700        27:EF             1001  0001
```

2.2.6 Help "On Screen Help Summary"

The HELP command is used to display a summary of the D-Bug12 command set. Each command is shown with its command line format and a brief description of its function.

3 Academic Misconduct

The policy for this course is zero tolerance for any form of academic misconduct. Consultation with other students is encouraged especially on design issues. However, directly copying another student's work or copying portions of code for example assembly language code) is an honour code violation and will result in a failing grade and may result in a failing grade in the course. Students will automatically be referred to the Director of the School and Dean of the college for action. **Please refer to the regulations outlined in the student handbook regarding academic misconduct.**

4 Appendix A - D-Bug12 Commands "Table"

Command	**Description**
ASM $< address >$	Single line assembler/disassembler
BAUD $< BAUDRate >$	Set the SCI communication baud rate
BF $< StartAddress >< EndAddress >$ [$< Data >$]	Block Fill user memory with data
BR [$< Address >< Address >$ ..]	Set/display user breakpoints
BULK	Bulk erase on-chip EEPROM
CALL [$< Address >$]	Execute a user subroutine; return to D-Bug12
G [$< Address >$]	**Go – begin execution of user program**
GT $< address >$	Go Till – set a temp breakpoint and begin execution of user program
HELP	Display D-Bug12 command set and command syntax
LOAD [$< AddressOffest >$]	Load user program in S-record format
MD $< StartAddress >$ [$< EndAddress >$]	Memory Display - display memory contents in hex bytes/ASCII format
MDW $< StartAddress >$ [$< EndAddress >$]	Memory Display Word - display memory contents in hex bytes/ASCII format
MM $< Address >$ [$< data >$]	Memory Modify - interactively examine/change memory contents
MMW $< Address >$ [$< data >$]	Memory Modify Word - interactively examine/change memory contents
MOVE $< StartAddress >< EndAddress >$ $< DestAddress >$	Move a block of memory
NOBR [$< Address >< Address >$ ..]	Remove individual user breakpoint
RD	Register Display – display the CPU registers
RM	Register Modify – interactively examine/change CPU register contents
T [$< Count >$]	Trace – execute an instruction, disassemble it and display the CPU registers
UPLOAD $< StartAddress >< EndAddress >$	Display memory contents in S-Record format
VERF [$< AddressOffset >$]	Verify memory contents against S-Record Data

Table 1: D-Bug12 Command Set Summary

5 Appendix B - D-Bug12 Commands “Useful Commands”

BF “Block Fill”

The Block Fill command is used to place a single 8-bit value into a range of memory locations.
syntax

```
BF <StartAddress> <EndAddress> [<Data>]
```

The following example will fill memory from location 4000 to 4100 with zeros.

```
> BF 4000 4100 0
```

CALL “Call Subroutine”

The CALL command is used to execute a subroutine and return to the D-Bug12 monitor program when the final RTS of the subroutine is executed. When control is returned to D-Bug12, the CPU register contents are displayed.
syntax

```
CALL  [<Address>]
```

The following example will illustrate the CALL operation.

```
> CALL 820
Subroutine Call Returned

PC      SP     X     Y     D = A:B      CCR= SXHI NZVC
0820   0A00   057C  0000     0F:F9           1001 0000
>
```

BR “Breakpoint Set”

The BR command is used to set a software breakpoint at a specified address or to display any previously set breakpoints. The function of a breakpoint is to halt user program execution when the program reaches the breakpoint address. When a breakpoint address is encountered, D-Bug12 disassembles the instruction at the breakpoint address, prints the CPU12’s register contents, and waits for a D-Bug12 command to be entered by the user.
syntax

```
BR [<Address> <Address> ..]
```

The following example will illustrate the BR command.

```
> BR 35ec 2f80 c592
Breakpoints: 35EC 2F80 C592
> BR
Breakpoints: 35EC 2F80 C592
>
```

NOBR “Remove Breakpoints”

The NOBR command can be used to remove one or more previously entered breakpoints.
syntax

```
NOBR [<Address> <Address> ..]
```

The following example will illustrate the BR command.

```
> BR 800 810 820 830
Breakpoints: 0800 0810 0820 0830
> NOBR 810 820
Breakpoints: 0800 0830
>
```

T “Trace”

The Trace command is used to execute one or more user program instructions beginning at the current Program Counter (PC) location. As each program instruction is executed, the CPU12’s register contents are displayed and the next instruction to be executed is displayed. The following example will illustrate the Trace command.

```
> T
 PC     SP     X     Y         D = A:B     CCR  =  SXHI  NZVC
0803   09FE   057C  0000          10:00            1001  0000
0803    830001            SUBD   #$0001
```