ENGG364: Microcomputer Interfacing

Lab 2: Parallel Port Programming & General Interfacing

School of Engineering, University of Guelph
Fall 2012

Start Date: *Wednesday (Week#4)*
Due Date: *No Report is Required*

1 Objectives:

The purpose of this lab is to:

- Develop fundamental concepts associated with programming parallel ports available on the MC68HC12 micro-controller.
- Design and construct a system which displays the 8-bit binary (ASCII) code of a pressed PC keyboard on LED's.
- Design an equivalent software decoder to display BCD numbers on a 7-segment display.

2 Introduction

Data transfer between the I/O device and the interface chip can proceed bit-by-bit (serial) or in multiple bits (parallel). Data are transferred serially in low-speed devices such as modems and low-speed printers. Parallel data transfer is mainly used by high-speed I/O devices. In this lab we will only be concerned with parallel I/O interfacing.

After you become familiar with the on-chip ports, we are going to perform several practical experiments to learn how to interface the M68HC12 to the real world. In the previous lab you became familiar with programming techniques using the assembler directives. In real life, the MCU deals with outside devices using its internal registers, memory, timer, A/D, serial ports, and I/O ports. Usually there is an electronic interfacing circuit between the MCU and the outside world. The outside device could be input or output devices. The input devices could be digital or analog. Digital input devices include switches, keyboards, digital integrated sensors that send 0V (logic 0) or 5V (logic 1), and so on. There are many analog input devices — for example, all types of transducers. A transducer is an analog device that converts some form of energy to an electrical signal (this will be covered later on in the course).

3 Preparation

The preparation for this lab involves the familiarity with :

- I/O parallel ports of the 68HC12 and programming them.
- The Light Emmitting Diodes (LEDs) on the Dragon12-Plus2.
- The Seven Segment Display on the Dragon12-Plus2.

3.1 MC68HC12 I/O Ports

The 68HC812A4 has 91 I/O pins arranged in **Twelve** I/O ports. All of these pins serve multiple functions depending on the operation mode and data in the control registers. Ports **A** through **J** and ports **S** and **T** are used as general-purpose I/O pins under the control of their associated data direction registers. Port AD is fixed as either inputs for the A/D converter or as logic inputs and does not have a data direction register. Each pin of the bidirectional ports has an associated bit in a specific port data register and another in a data direction register. The data direction register bits are used to specify the primary direction of data for each I/O pin. When an output pin is read, the value at the input to the pin driver is returned. When a pin is configured as an input, that pin becomes a **high-impedance** input. If a write is attempted to an input pin, its value does not affect the I/O pin but is stored in an internal latch. When the pin becomes an output, this value appears at the I/O pin. Data direction register bits are **cleared** by reset to configure I/O pins as inputs.

So to configure a pin in a certain port as an **output pin**, a **'1'** should be written to the corresponding data direction register, else a '0' should be written. The following example illustrates how to configure port A on the MC68HC12.

3.1.1 Example of Configuring a Port

Problem: Write an instruction sequence to set **port J** as an output port, and output the value $CD to it.
Solution: Since **port A** is bi-directional, we need to configure it as an output port. Figure 1 shows the **Port A** Data Register and Data Direction Registers. Notice that in the code below the label PORTA was set to $0000 which is the address of Port A data register and the label DDRA was set to $0002 which is the address of the 'A' data direction register.

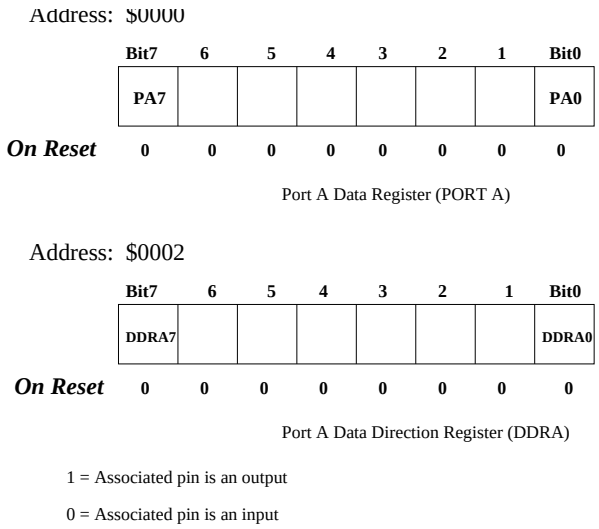

Figure 1: Port A, Data/Direction Registers

```
REGBAS    EQU    $0000   ; base address of all I/O registers
PORTA     EQU    $0000   ; offset of PORTA from REGBAS
DDRA      EQU    $0002   ; offset of DDRJ from REGBAS

          LDX    #REGBAS
          LDAA   #$FF    ; Set all pines of PORTA (all output)
          STAA   DDRA,X  ; set PORTA as an output port
          LDAA   #$CD    ; value to output to PORTA
          STAA   PORTA,X ; output the data
          END
```

3.1.2 Dragon12-Plus2 Port Allocation

It should be noted that the ports on the Dragon12-Plus2 are multiplexed, that is they can have one of two or more possible functions.

The following table provides a summary:

Port and Pin	I/O Usage	Comments
PAD0	Selects EVB or alternate execution mode	Set to 0 for EVB Mode
PAD1	Selects EVB or alternate execution mode	Set to 0 for EVB Mode
PORT A	Connected to Keypad on Dragon12-Plus2	Can be used as General Purpose I/O
PORT B	Conncected to LEDs on Dragon12-Plus2	Can be used as General Purpose I/O
PORT E	External Signals (E-clk, IRQ, XIRQ, etc)	Not to be used
PORT H	Connected to DIP Switches and Push Buttons	Can be used as General Purpose I/O
PORT J	Used to Enable LEDs on Dragon12-Plus2	Not to be used
PORT P	Used to Enable 7-Segment Displays	Not to be used as General Purpose I/O
PORT H	Connected to LCD Module on Dragon12-Plus2	Can be used as General Purpose I/O

Please refer to Appendix A for detailed description of the MCU connectors H1-H8 i.e. how to access the I/O ports on the Dragon12-Plus2 Board and Appendix B for the address of some registers utilized in certain ports.

3.2 Simple Interface to LED's

The light emitting diode (LED) is one of the most often used output devices in an embedded system. In many embedded systems, LEDs are simply used to indicate that the system is operating properly.

An LED can illuminate if it is forward-biased and has sufficient current flowing through it. The current required to light an LED may range from a few to more than 10 mA. The voltage drop across the LED when it is forward-biased can range from 1.6V to more than 2.2V.

LED indicators are easy to interface with micro-controllers. All you need is enough current to drive the LED and a series resistor to absorb the voltage drop (limit the current to some acceptable value!). The circuit in Figure 2 is often used to interface with an LED. Using a 5-volt supply and assuming that the LED has a 1.7 volt drop across it, a suitable resistor will limit the current to 10 mA. An I/O port pin of a micro-controller generally does not have enough drive to supply the current. So an inverter is often used as a switch to turn the LED on and off. When the inverter's output is low (close to 0V), the diode has a 2.0V voltage drop, and by ohm's law: $5V = 2.0V + I_{rx} \times R_x$. When setting the I_{rx} to 10mA, the resistor R_x is solved to be 300Ω.

3.3 Seven Segment Decoder

Digital readouts found in electronic calculators and digital watches use LEDs. Each digit of the readout is formed from seven segments, each consisting of one LED that can be illuminated by digital signals. A BCD-

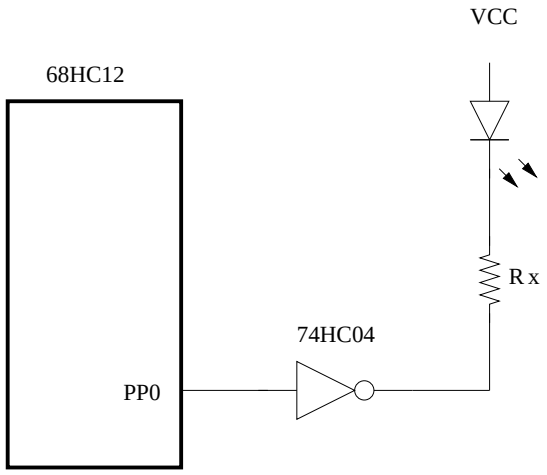

Figure 2: An LED connected to an inverter

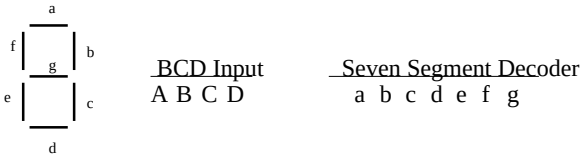

Figure 3: Seven Segment Display.

to-seven-segment decoder is a combinational circuit that accepts a decimal digit in BCD and generates the appropriate outputs for the selection of segments that display the decimal digit. In this lab you will attempt to interface the MPU to a seven segment display located on the Dragon12-Plus2 Board using software.

Notice that Port P is used to multiplex and enable the 4 units of 7-Segment Display.

- PP0 is connected to Digit 3 of the 7-Segment display.
- PP1 is connected to Digit 2 of the 7-Segment display.
- PP2 is connected to Digit 1 of the 7-Segment display.
- PP3 is connected to Digit 0 of the 7-Segment display.

Refer to Appendix C for more details of how to utilize the 7-Segment Displays on the Dragon12-Plus2 board.

4 Requirements

You are required to accomplish the following for this lab:

1. Use the example provided to you in Section 3.1.1 to familiarize yourself with configuring I/O ports and apply the same technique to **port B** instead of **port A**.
2. The user will type characters on the PC keyboard whose values will be sent to the PC's serial (COM) port by software on the PC. The PC serial port will be connected to the serial port on the 'HC12. The software on-board the 'HC12 will monitor its serial port for the 8-bit character codes and latch new codes on one of its parallel output ports. **Use Port B on the Dragon12-Plus2 board since this port is**

already connected to the LEDs.
Please refer to:

(a) Appendix D in this handout for information on utility routines that can help you get a char from the keyboard and printing a char to the terminal.

(b) Your text book Section 4.8 page 173 for using the D-Bug12 functions to perform I/O operations.

3. Seven Segment 'Software' Decoder:

   - Input a number via the PC keyboard using the Debug12 utility functions.
   - Display the number read on Digit0 of 7-Segment display on the Dragon12-Plus2 board.
   - Use Port P (since it is connected to the 7-Segment Displays).

5 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.**

6 Appendix A - MCU Connections

Eight pin header connectors (H1 .. H8), surrounding the breadboard provide access to the MCU's I/O and bus lines as seen in Figure 4. These connectors are located adjacent to the prototype area.

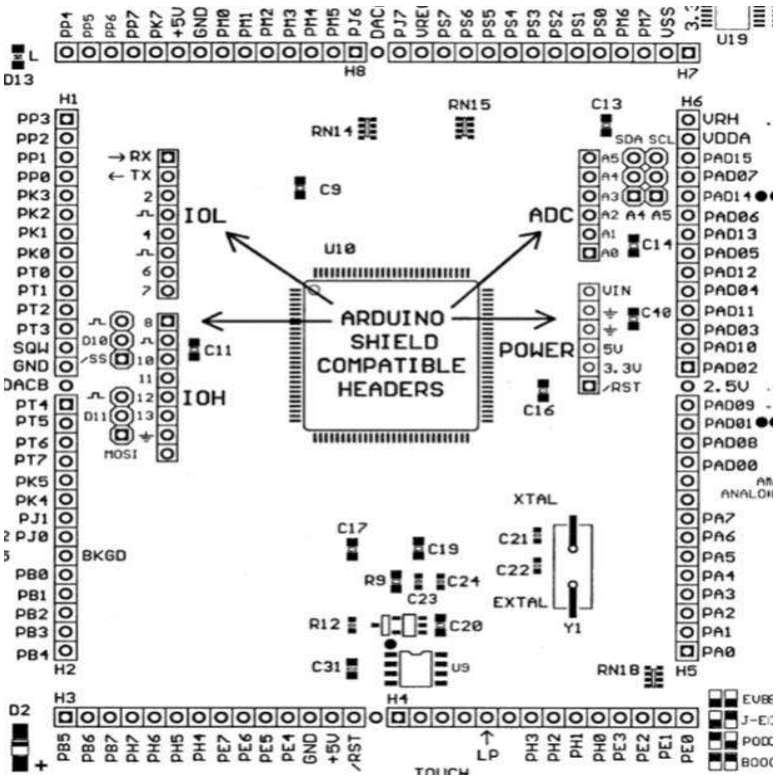

Figure 4: Main Pin Headers on the Dragon12-Plus2 Board

1. Connector H1 located at upper/left of breadboard.
2. Connector H2 located at lower/left of breadboard (below H1).
3. Connector H3 located at bottom/left of breadboard.
4. Connector H4 located at bottom/right of breadboard.
5. Connector H5 located at lower/right of breadboard.
6. Connector H6 located at upper/right of breadboard.
7. Connector H7 located at top/right of breadboard.
8. Connector H8 located at top/left of breadboard.

Table 1 provides a brief description of the signals.

I/O Pin Usage

Many I/O pins of the MC9S12DG256 on the Dragon12-Plus2 board are used by on-board peripherals, here are examples:

1. PORT A: Keypad.

Connector	Signal Name	Port	Description
H1	PP3, PP2, PP1, PP0	Port P(0..3)	Gen Purpose I/O
H1	PK3, PK2, PK1, PK0	Port K(0..3)	Gen Purpose I/O
H1	PT0, PT1, PT2, PT3	Port T(0..3)	NOT USED
H2	PT4, PT5, PT6, PT7	Port T(4..7)	NOT USED
H2	PK5, PK4	Port K(4..5)	Gen Purpose I/O
H2	PJ1, PJ0	Port J(0..1)	NOT USED
H2	PB0, PB1, PB2, PB3, PB4	Port B(0..4)	Gen Purpose I/O
H3	PB5, PB6, PB7	Port B(5..7)	Gen Purpose I/O
H3	PH7, PH6, PH5, PH4	Port H(4..7)	Gen Purpose I/O
H3	PE7, PE6, PE5, PE4	Port E(4..7)	NOT USED
H4	PH3, PH2, PH1, PH0	Port H(0..3)	Gen Purpose I/O
H4	PE3, PE2, PE1, PE0	Port E(0..3)	NOT USED
H5	PA0, PA1, PA2, PA3, PA4, PA5, PA6, PA7	Port A(0..7)	Gen Purpose I/O
H5	PAD00, PAD08, PAD01, PAD09	Port AD0(0,1,8,9)	Analog-to-Dig
H6	PAD02, PAD03, PAD04, PAD05, PAD06, PAD07	Port AD0(2..7)	Analog-to-Dig
H6	PAD10, PAD11, PAD12, PAD13, PAD14, PAD15	Port AD1(0..5)	Analog-to-Dig
H7	PS0, PS1, PS2, PS3, PS4, PS5, PS6, PS7	Port S(0..7)	Serial Comm
H7	PM6, PM7	Port M(6,7)	NOT USED
H8	PM0, PM1, PM2, PM3, PM4, PM5	Port M(0..5)	NOT USED
H8	PP4, PP5, PP6, PP7	Port P(4..7)	Gen Purpose I/O

Table 1: MCU Connectors J8/J9 Pin Assignments

2. PORT B: Light Emmitting Diodes (LEDs)
3. PORT H: DIP and Pushbutton Switches
4. PORT K: Liquid Crystal Display (LCD)
5. PORT P: 7-Segment Display
6. PORT AD: Light Sensor, Temp Sensor, Trimmer Pot

IMPORTANT:
Fortunetly, it's unlikely that all on-board peripherals will be used by one application program. So the I/O pins on unused peripheral devices can still be used by your circuits on the breadboard. For instance, if you don't touch the 4x4 on-board keypad, the entire port A will be available to your circuits.

7 Appendix B - Port Registers

Each port within the M68HC12 has several registers that are used to control its functionality and also give the status of the port following a read/write transaction. Table 2 provides a brief description of the regular I/O ports and their associated registers. Ports A, B, E, K can be used for Input/Output based on how the "Data Direction" register has been programmed.

Port	Register	**Description**	**Address**	**Current Usage**
PORT A	PORTA	Data Register **A**	$0000	Keypad
	DDRA	Data Direction Register **A**	$0002	
PORT B	PORTB	Data Register **B**	$0001	LED
	DDRB	Data Direction Register **B**	$0003	
PORT P	PORTP	Data Register **P**	$0258	7-Seg Display
	DDRP	Data Direction Register **P**	$025A	
PORT H	PORTH	Data Register **H**	$0260	DIP Switches
	DDRH	Data Direction Register **H**	$0262	
PORT E	PORTE	Data Register **E**	$0008	IRQ, XIRQ
	DDRE	Data Direction Register **E**	$0009	
PORT K	PORTK	Data Register **K**	$0032	LCD
	DDRK	Data Direction Register **K**	$0033	

Table 2: M68HC12 I/O Ports and Associated Registers

Table 3 provides a brief description of the I/O ports (H,J) that have interrupt capability (Wakeup). Port P is very similar to ports H and J in functionality.

Port	Register	**Description**	**Address**
PORT H	PTH	Data Register **H**	$0260
	DDRH	Data Direction Register **H**	$0262
	PTIH	Input Register **H**	$0261
	RDRH	Reduced Drive Register **H**	$0263
	PERH	Pull Devide Enable **H**	$0264
	PPSH	Polarity Select Register **H**	$0265
	PIEH	Interrupt Enable Register (Wakeup)	$0266
	PIFH	Interrupt Flag Register **H**	$0267
PORT J	PTJ	Data Register **J**	$0268
	DDRJ	Data Direction Register **J**	$026a
	PTIJ	Input Register **J**	$0269
	RDRJ	Reduced Drive Register **J**	$026b
	PERJ	Pull Device Enable **J**	$026c
	PPSJ	Polarity Select Register **J**	$026d
	PIEJ	Interrupt Enable Register **J**	$026e
	PIFJ	Interrupt Flag Register **J**	$026f

Table 3: I/O Ports with Interrupt Enable Capability

Table 4 provides a brief description of other I/O ports and registers that are used for "Real Time Interrupt" (Section 6.7 in Text Book), "Analog-to-Digital Conversion" (Section 12.3 in Text Book), and "Serial Communication".

Port	Register	**Description**	**Address**
RTI	RTICTL	Real Time Interrupt Control Register (RTR0:RTR6)	$003B
	CRGFLG	The CRG Flag Register (RTIF)	$0037
	CRGINT	The CRG Interrupt Enable Register (RTIE)	$0038
PORT ATD	ATD0CTL0	Control Register 0 (Reserved)	$0080
	ATD0CTL1	Control Register 1 (Reserved)	$0081
	ATD0CTL2	Control Register 2 (ADPU)	$0082
	ATD0CTL3	Control Register 3 (S8C, S4C, S2C, S1C)	$0083
	ATD0CTL4	Control Register 4 (PRS4:PRS0)	$0084
	ATD0CTL5	Control Register 5 (SCAN, MULT, CC, CB, CA)	$0085
	ATDSTAT0	Status Register 0 (SCF)	$0086
	ATDSTAT1	Status Register 1 (Reserved)	$0087
	PORTAD0	Data Input (Bit7:Bit0)	$008F
	ATD0DR0H	Result Register 0 (High Byte)	$0090
	ATD0DR7H	Result Register 7 (High Byte)	$009E
PORT SCI 0	SCI0BDH	Baud Rate Register High	$00C8
	SCI0BDL	Baud Rate Register Low	$00C9
	SCI0CR1	Control Register 1	$00CA
	SCI0CR2	Control Register 2	$00CB
	SCI0SR1	Status Register 1	$00CC
	SCI0SR2	Status Register 2	$00CD
	SCI0DRH	Data Register High	$00CE
	SCI0DRL	Data Register Low	$00CF

Table 4: M68HC12 Ports and Registers

8 Appendix C - Evaluation Board "Displays of Dragon12-Plus2 Board"

The Dragon12-Plus2 Board has several means to display information by the MCU12. Here are some examples:

1. Light Emmitting Diodes, (LEDs) (connected to Port B)
2. Seven-Segment Displays, (connected to Port P)
3. Liquid Crystal Display, (connected to Port K).

Each peripheral is connected to a specific port as demonstrated above. These ports can still be used as general purpose ports if the peripherals are not used.

1. If you don't use the LCD or just unplug the LCD, the **Port K** will be available as well.
2. **Port B** drives LEDs, but if you ignore the status of the LEDs, the port B can drive any other I/O devices on the breadboard.

Please refer to the "Dragon12-Plus2 Trainer, User's Manual" for more details.

LEDs:

Each port B line is monitored by an LED. In order to turn on port B LEDs, the PJ1 (port J pin 1) must be programmed as output and set for logic zero.

7-Segment LED multiplexing

There are 4 digits of 7-Segment LEDs on the Dragon12-Plus2 board. The type of the 7-Segment display on board is called common cathode. in an individual digit, all anodes are driven individually by an output pot and all cathodes are internally connected together. Before sending a number to a 7-Segment LED, the number must be converted to its corresponding 7-segment code depending how the 7-segment display is connected to an output port.

The Dragon12-Plus2 board uses port B to drive 7-segment anodes and used PP0-PP3 (i.e., Port P) to drive common cathodes. In the next few paragraphs we will explain how to multiplex 7-segment by displaying the number 1234 on the display. By convention, the 7-Segments are called segment a, b, c, d, e, f and g, as shown in Figure 5.

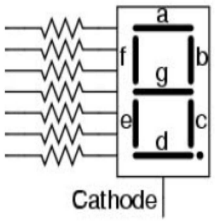

Figure 5: The Dragon12 7-Segment

The segment a, b, c, d, e, f, g and Decimal Point are driven by (PORT B) PB0, PB1, PB2, PB3, PB4, PB5, PB6, PB7, respectively. The hex value of the segment code is shown in the following table:

The schematic for multiplexing 4 digits is shown in Figure 7.

Number	DP	G	F	E	D	C	B	A	Hex Value
1	0	0	0	0	0	1	1	0	$06
2	0	1	0	1	1	0	1	1	$5B
3	0	1	0	0	1	1	1	1	$4F
4	0	1	1	0	0	1	1	0	$66

Figure 6: Hex Value of the numbers '1, 2,3, 4'

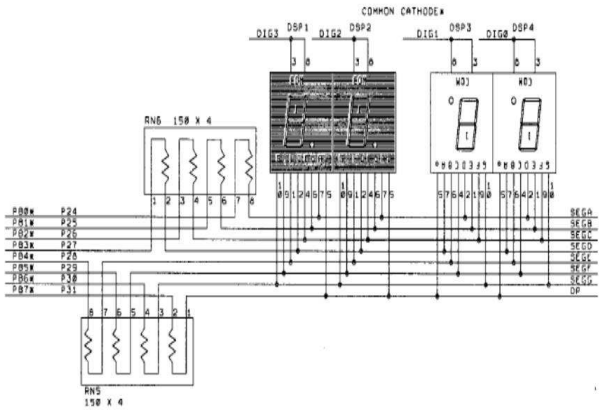

Figure 7: Multiplexing the 7-Segment Displays on the Dragon12-Plus2 Board

The digits **3, 2, 1 and 0** are driven by PP0, PP1, PP2 and PP3, respectively (PORT P). The 7-Segment LED is turned on one at a time at 250 HZ refresh rate. It's so fast that our eyes will perceive that all digits are turned on at the same time. To display the number '1234' on the 7-Segment display, the following steps should be taken:

1. Output $06 to Port B, set PP0 low and PP1, PP2 and PP3 high. The number 1 is shown on the digit 3 (the leftmost digit) but other 3 digits are turned off.
2. Delay 1ms.
3. Output $5B to Port B, set PP1 low and PP0, PP2 and PP3 high. The number 2 is shown on the digit 2 but other 3 digits are turned off.
4. Delay 1ms.
5. Output $4F to Port B, set PP2 low and PP0, PP1 and PP3 high. The number 3 is shown on the digit 1 but other 3 digits are turned off.
6. Delay 1ms.
7. Output $66 to Port B, set PP3 low and PP0, PP1 and PP2 high. The number 4 is shown on the digit 0 (the rightmost digit) but other 3 digits are turned off.
8. Delay 1ms.
9. Go back to step 1.

9 Appendix D - Evaluation Board "Using D-Bug12 Routines"

The D-Bug12 monitor provides a few subroutines to support I/O operations. One can utilize these I/O routines to facilitate program development. The D-Bug12 currently provides access to 18 different utility routines through a table of 16-bit pointers beginning at address $EE80 as shown in Table 5. This appendix will provide you with some information that will allow you to utilize internal D-Bug12 routines.

Please refer to Page 173, Section 4.8 in your text book "Using the D-Bug12 Functions to Perform I/O Operations for detailed examples.

Function	Description	Vector Table Address
far main()	Start of D-Bug12	$EE80
getchar()	Get a char from SCI0 or SCI1	$EE84
putchar()	send a char out SCI0 or SCI1	$EE86
printf()	Formatted Output - binary to char	$EE88
far GetCmdLine()	Obtain a line of input from the user	$EE8A
far sscanhex()	Convert and ASCII hex to binary int	$EE8E
isxdigit()	Checks for membership [0..0,a..f,A..F]	$EE92
toupper()	Converts lower case char to upper case	$EE94
isalpha()	Checks for membership in [a..z,A..Z]	$EE96
strlen()	Returns the length of a string	$EE98
strcpy()	Copies a null terminated string	$EE9A
far out2hex()	Displays 8-bit number as 2 ASCII chars	$EE9C
far out4hex()	Displays 16-bit number as 4 ASCII chars	$EEA0
SetUserVector()	Setup user interrupt service routine	$EEA4
far WriteEEByte()	Write a data byte to on-chip EEPROM	$EEA6
far EraseEE()	Bulk erase on-chip EEPROM	$EEAA
far ReadMem()	Read data from M68HC12 memory map	$EEAE
far WriteMem()	Write data to M68HC12 memory map	$EEB2

Table 5: D-Bug12 Utility Routines Summary

Here are a set of rules that you can follow to utilize these utility routines:

1. Calling a function from assembly language is simple.
   - First, **push the parameters onto the stack** in the proper order, loading the **first or only function parameter** into the D Accumulator. In other words if you have two parameters or arguments then one will be passed by the D Accumulator and the other pushed onto the stack.
   - Then call the function with a JSR instruction.
   - The code following the JSR instruction should remove any parameters pushed onto the stack. If a single parameter is stacked, a PULX or PULY instruction is one of the most efficient ways to unstack it.
2. All 8-bit and 16-bit function results are returned in the D Accumulator. Char values returned in the D accumulator are located in the 8-bit B Accumulator.
3. Boolean function results are zero for False and non-zero values for True.

Example #1:
To output a single character to the control terminal SCI (i.e., screen) you can use the **int putchar(int);** function whose "Pointer Address" is located at $EE86. If the control SCI's transmit data register is full when the function is called, **putchar()** will wait until the transmit data register is empty before sending the character.

Adding the following instruction sequence in your program will output the character **"A"** to the control SCI:

```
putchar         equ     $EE86
                ...
                ldd     #'A'
                jsr     [putchar,PCR]
                ...
```

The addressing mode used in the jsr instruction of this example is a form of **indexed indirect addressing** that uses the program counter as an index register. In reality, the HCS12 does not support PCR. Instead, the PCR mnemonic is used to instruct the assembler to calculate an offset to the address specified by the label **putchar**. The MiniIDE software supports this syntax. However, if you are using an assembler that does not support program-counter relative indexed addressing the following two instruction sequence can be used instead:

```
    ldx     putchar     ; load the address of putchar
    jsr     0,x         ; call the subroutine
```

Important: If the name of a library function is preceded by the keyword *far*, then it is located in the expanded memory and must be called by using the **call** instruction.

Example #2:
To retrieve a single character from the control terminal SCI use the **int getchar(void);** function whose pointer address is located in "$EE84". If an unread character is not available in the receive data register when this function is called, it will wait until one is received. Because the character is returned as an integer, the 8-bit character is placed in accumulator B.

Adding the following instruction sequence in your program will read a character from the SCI0 port

```
getchar         equ     $EE84
                ...
                jsr     [getchar,PCR]
                ...
```

Example #3:
The following is a sample code for utilizing the getchar() and printf() utility routines Notice that there are two org statements. The first org statement is to set the starting address of the variables and constants at location $1000. The second org statement is to set the start address of the main program to be executed to location $1100. When you attempt to run your program you need to issue the command g $1100 in the miniIDE environment.

```
CR          EQU   $0D                      ; ASCII Code for Carriage Return
LF          EQU   $0A                      ; ASCII Code for Line Feed
getchar     EQU   $EE84                    ; Address of Debug12 Utility Function getchar()
printf      EQU   $EE88                    ; Address of Debug12 Utility Function printf()

            ORG   $1000                    ; Start Address of variables/constants
```

```
msg         db    "char pressed is %c",CR,LF,0

            ORG   $1100                       ; Start Address of main program
getnewval   LDX   getchar                     ; get the character from the keyboard
            JSR   0,X

            CMPB  #$1B                        ; Check if the char is an ESC
            BEQ   terminate                   ; jump to end of program

            PSHD                              ; push char on stack (parameter for printf)
            LDD   #msg                        ; second parameter in ACC D
            LDX   printf                      ; Load Index Reg X with printf address
            JSR   0,X                         ; print message on terminal

            BRA   getnewval                   ; repeat getting new chars
terminate   SWI                               ; end of program (return to D-Bug12)
```