HCS12 FAMILY

EMBEDDED C USING CODEWARRIOR

INTERFACING WITH THE PORTS

Featuring Bit Twiddling

TECHNOLOGICAL ARTS, INC.
Toll-free: 1-877-963-8996 (USA and Canada)
Phone: +1 (416) 963-8996 Fax: +1 (416) 963-9179

www.technologicalarts.com

BEFORE READING THIS GUIDE . . .

The source code herein is written in Embedded C using the Metrowerks CodeWarrior 3.1 IDE.

If you are new to CodeWarrior 3.1 or embedded C, you can get a free copy of Technological Arts' *Embedded C Using CodeWarrior - Getting Started Manual* from our website. This manual will show you how to start a project, where to write your source code, compile the code, and finally how to download it to your HCS12 board!

The Embedded C code written herein applies to the 9S12C derivatives of the HCS12 family, however . . .

The general port concepts are similar for all HCS12 derivatives. The embedded C coding structure for the ports mentioned is similar for all HCS12s plus/minus a few registers. We will let you know where you should be concerned with your own HCS12 derivative.

PORT AHEAD….. AYE AYE CAPTAIN!

If you want to program your HCS12 to communicate with the outside world (your keypad, LCD and other peripherals) then you've come to the right place.

Ports are the HCS12's way to allow your peripherals to communicate with the HCS12 core.

The purposes for this guide are to:

- Describe the function and reason for using ports
- Describe general port functions and the basis for designating port names
- Introduce Port T and Port AD
- Develop an algorithm for communicating through Ports T and AD
- Define Port T and AD's registers
- Introduce bit twiddling
- Provide a simple code combining port programming and bit twiddling
- Provide a circuit diagram to correspond with the code
- Provide a resource for finding additional port information

WHAT ARE PORTS ?

Ports are the HCS12's built in, programmable interface devices that allow your peripherals to communicate with the HCS12 core. **Figure 1** shows a simplified view of how the ports physically connect the peripherals to the core.

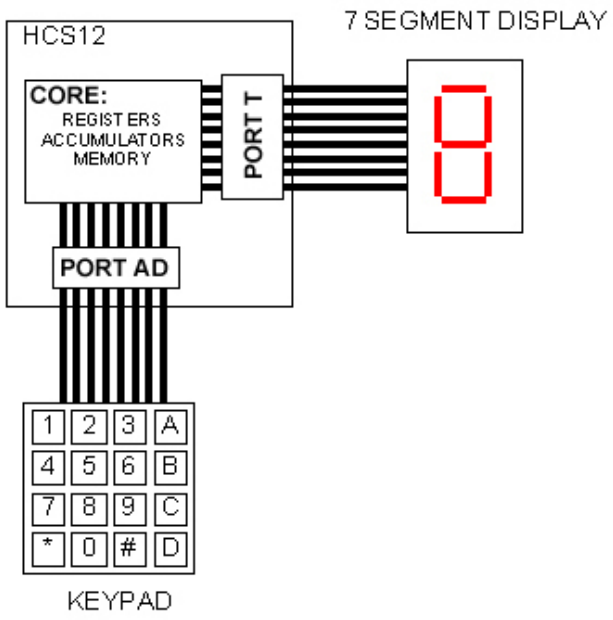

FIGURE 1

WHAT ARE THE PURPOSES OF PORTS?

Ports can be programmed to serve several purposes including:

- Controlling data flow
- Controlling data direction

Controlling data flow

External peripherals can transmit and receive data at different speeds. Most ports can be programmed to control the flow of data to prevent **data over run** - where a peripheral is receiving data from the HCS12 faster than it can process.
Figures 2a, 2b and **2c** illustrate a simplified flow control process for sending data to an output device:

Figure 2a – 1. If the HCS12's core has data to send to port A, it first checks if the port is ready to accept data. This is to prevent the core's data from overwriting data currently being transmitted from port A to the LCD.

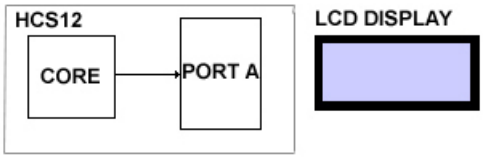

FIGURE 2a

Figure 2b – 2. If port A is ready to accept data, the core will transmit the data to the port via the data bus. If the port is not ready, the core can be programmed to either periodically check for the port's readiness, or continue with other tasks until the port issues an interrupt signal, indicating that it is ready to accept data.

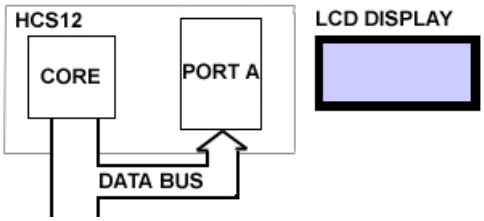

FIGURE 2b

Figure 2c- 3. Port A transmits the data to the LCD

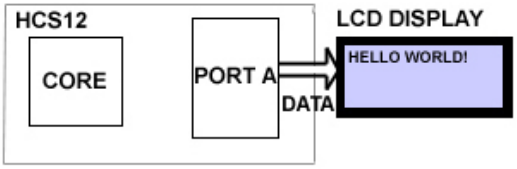

FIGURE 2c

Figure 3a, **3b** and **3c** illustrate the simplified flow control process for receiving data from an input device.

Figure 3a- 1. If port A is ready to receive data from the keypad, it will wait for a key to be entered.

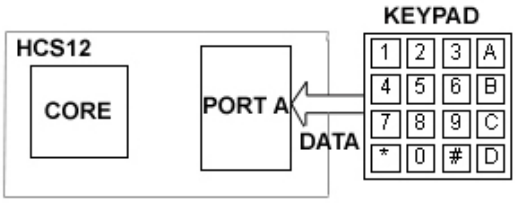

FIGURE 3a

Figure 3b- 2. Once port A receives data from the keypad, it will not accept more data until the current data has been read by the HCS12's core. The core can be programmed to either periodically check the port for data, or continue with other tasks until the port issues an interrupt indicating that it is has data to be read.

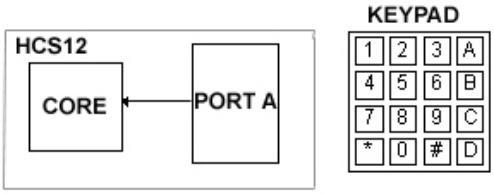

FIGURE 3b

Figure 3c- 3. The core reads the data from port A. Port A is now free to accept data from the keypad.

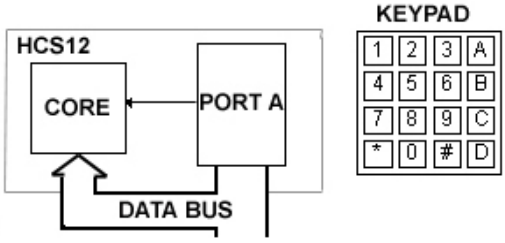

FIGURE 3c

Controlling Data Direction

Most ports can be programmed to allow data to flow in a particular direction. Some ports are limited to input or output function only. **Figure 4a**, **4b**, **4c** illustrate the possible programmable directions of data flow for most ports.

Figure 4a - direction of all port lines are programmed for output

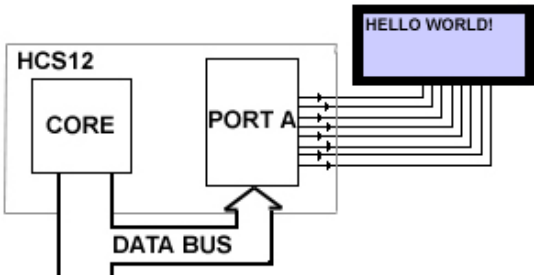

FIGURE 4a

Figure 4b – direction of all port lines are programmed for input

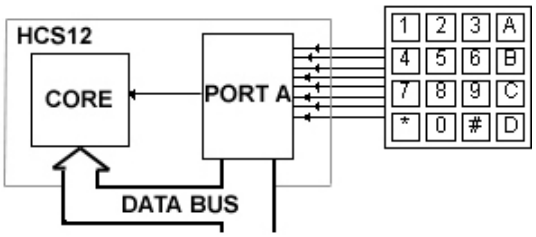

FIGURE 4b

Figure 4c- some port lines are programmed for input while others are programmed for output.

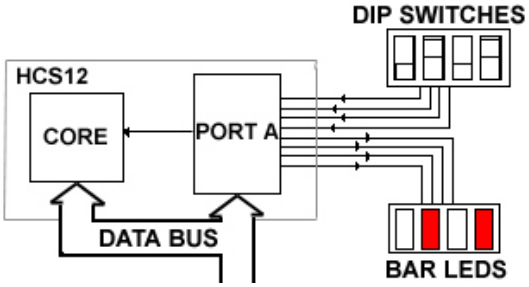

FIGURE 4c

GENERAL PORTS FUNCTIONS AND DESIGNATIONS

The number of programmable ports and the functionality of each available port are dependent on your HCS12's derivative.

A port can be designed to serve one of the three functions:

- dedicated input
- dedicated output
- input and output

Some ports, in addition to I/O are designed to serve a unique function eg- analog to digital conversion, pulse width modulation etc.

In general, each available port on your HCS12 is a designated letter that identifies the port's function and the number of data lines connected to it.

NOTE: The Motorola document xxxPIMVx.pdf, found on the Technological Art's resource CD or on the website at www.technologicalarts.com, provides the port names associated with your HCS12 derivative and their descriptions including register names and their programmable functions.

Depending on your HCS12's derivative, some or most port lines will be physically connected to the HCS12's external pins. Note: Technological Art's HCS12 modules + docking stations provide easy access to these pins. Refer to your board's datasheet, under the table H1, for more information.

On the majority of HCS12 derivatives, ports T and AD are connected to the external HCS12 pins.

We will now introduce ports T and AD, show you how to program their registers, and create a simple program to communicate between a set of DIP switches and bar LEDs. This will get you started with programming ports.

WHAT DO PORT T AND PORT AD DO?

Port T can be programmed for two functions:

- Providing a gateway for connecting the pulse width modulator and/or the timer to the corresponding peripherals
- To serve as general I/O

For our program we will be interested in the latter.

Port AD can also be programmed for two functions:

- Providing an 8 channel analog to digital converter
- To serve and general input and output (output available on most HCS12 derivatives)

Again, for our program, we will be interested in the latter.

THE ALGORITHM FOR OUR SIMPLE PROGRAM

Before we dive into programming the port registers we will briefly discuss the operation of our program.

Figure 5 shows the general interface between the peripherals and the HCS12.

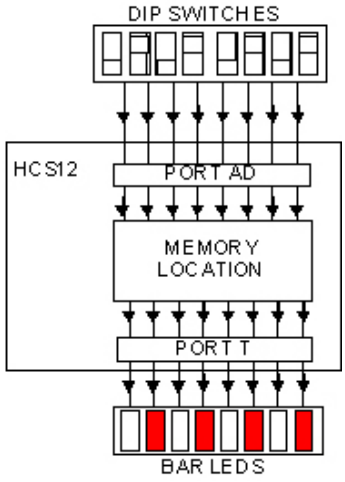

FIGURE 5

THE ALGORITHM

1. Port AD accepts an 8 bit combination from the DIP switches
2. The HCS12 will read the value from Port AD and store it in memory
3. The HCS12 will perform a couple **bit twiddling** operations on the value
4. The result from bit twiddling will be sent to port T
5. Port T will send the value to the bar LEDs which will display the result

PROGRAMMING THE PORT T AND AD REGISTERS

From the program guidelines, we need to program port Ts registers to serve as general output and PORT AD to serve as general input. We will start with port T since it is easier.

For port T, there are only two registers we need to be concerned with:

- The data direction register → DDRT
- The transmit data register → PTT

Register DDRT is an 8 bit data direction register for port T. Each bit corresponds to the direction of the data line connected to port T .
1= OUTPUT 0= INPUT
We need to configure port T for 8 bit output we will assign 1s to all the DDRT's bits. Ie DDRT=0xFF

Register PTT is the 8 bit transmit register that we will use to send data to the bar LEDS.
Example: PTT = somevalue;

For Port AD, another 8 bit register, we will be interested in the following three registers:

- ATDCTL23
- ATDCTL45
- ATDDIEN
- PTAD

The detail behind these registers is extensive and can be found on the Analog to Digital conversion guide at www.technologicalarts.com . Just know that setting the ATDCTL23 and ATDCTL45 registers to 0 and ATDDIEN to FF

```
ie-  ATDCTL23=0x00;
     ATDCTL45=0x00;
     ATDDIEN = 0xFF;
```

will disable these registers' normal analog to digital functions, and allow the ports to be used at general input only. PTAD is Port AD's receive register where the expected data arrives.

So we have now configured port T and AD for output and input interfacing respectively. Now we need to focus on the intermediate part of bit twiddling!

WHAT IS BIT TWIDDLING?

Bit twiddling is the process of twiddling bits.

Okay - bit twiddling is the process of checking and/or manipulating programmer selected bit(s) of a byte or word. The results of these operations do not change state of the non selected bits of the byte or word.

Consider the byte in **figure 6**. In this case we have chosen to check if the logic state of bit 6 is set. We will use the embedded C instruction:

```
if((PORTAD &  0x40) = =1)  /* checking if bit  6 is set */
```

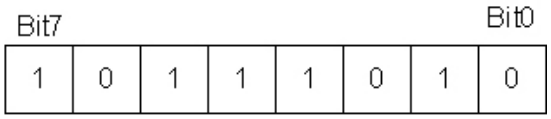

FIGURE 6

Now consider the byte in **figure 7a**. This time we have chosen to perform bit manipulation on bit 6 and bit 2. We have chosen to set bit 6 (to logic 1) and XOR bit 2 using the following embedded C instructions:

```
hexvalue = 0xBA;    /*assign  hexadecimal value to variable  hexvalue*/
hexvalue |= 0x40;   /* setting bit 6 of hexvalue*/
hexvalue ^= 0x02;   /* XOR (with 1) bit 2 of hexvalue*/
```

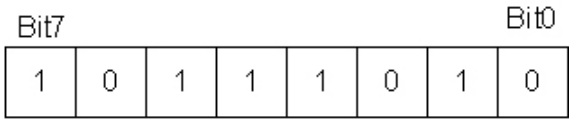

FIGURE 7a

The result of these two operations results in the byte in **figure 7b**.

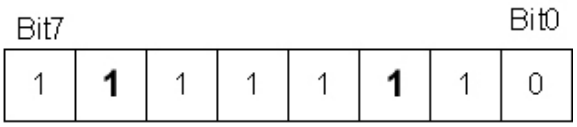

FIGURE 7b

UPDATING OUR ALGORITHM WITH BIT TWIDDLING

1. Port AD accepts an 8 bit combination from the DIP switches
2. The value from Port AD is stored in a variable
3. Set the bit 6 of the variable
4. XOR bit 2 of the variable
5. The resultant value of the variable will be sent to port T
6. Port T will send the value to the bar LEDs which will display the result

THE CIRCUIT DIAGRAM

Figure 8 shows the circuit diagram of two peripherals – The DIP switches, and the bar LEDs - that our program is designed to interface to the HCS12 core.

Build the circuit in **Figure 8** connecting the bar LEDs and DIP switches

NOTE: Refer to your board's datasheet to determine the actual pin numbers for port T and port AD.

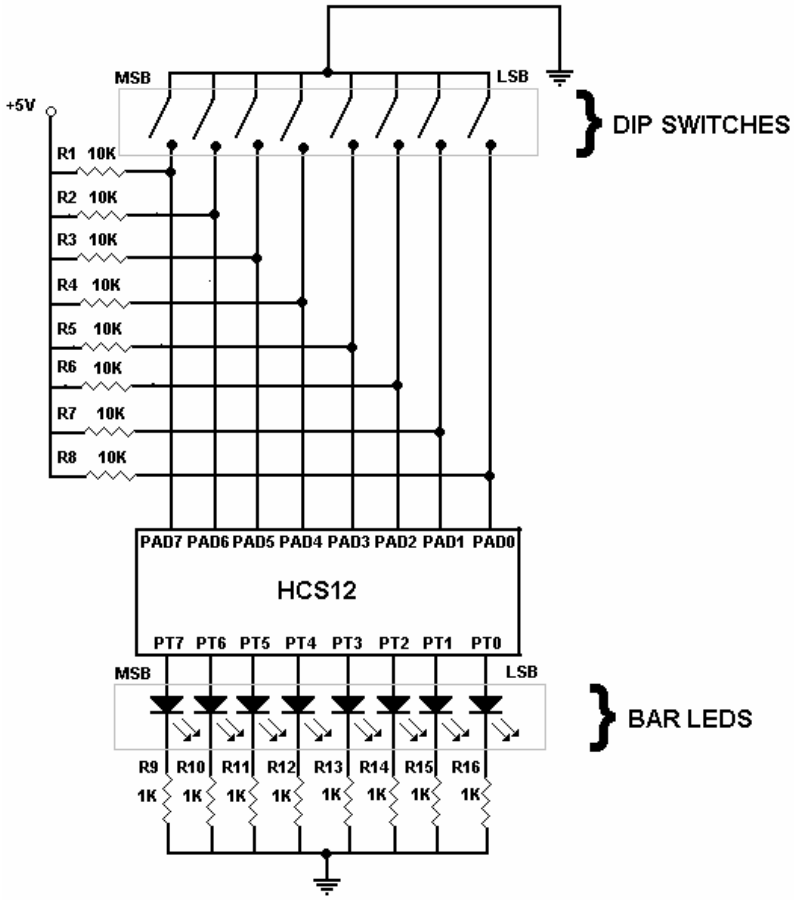

FIGURE 8

THE SOURCE CODE

```
//---------------------- Start of code – fully commented -------------------------------

#include <hidef.h>      /* common defines and macros */
#include <mc9s12c32.h>     /* derivative information */


#pragma LINK_INFO DERIVATIVE "mc9s12c32"

void main(void)
{
            // variable declaration
            unsigned char hexvalue;

            // Enabling DDRT for general output
              DDRT=0xFF;

            // Enabling Analog to digital register for general Input
            ATDCTL23=0x00;
            ATDCTL45=0x00;
            ATDDIEN = 0xFF;

            // Assign the value at the dip switches, via PTAD to hexvalue
               hexvalue=PTAD;

            // Set bit 6 of hex value
               hexvalue |= 0x40;  /* setting bit 6 of hexvalue*/

            // XOR bit 2 of hex value with 1
                 hexvalue ^= 0x04;  /* XOR (with 1) bit 2 of hexvalue*/

            // Send hexvalue to Port T
                  PTT=hexvalue;
}
//------------------------------ End of Source Code ------------------------------------------
```

TESTING YOUR CIRCUIT

Alright, time for a little quality assurance testing ie- making sure that your project works! Make the source code and download it to your HCS12.

1. With your program successfully downloaded, we want to first make sure that the DIP switch representing MSB of the input corresponds to the LED representing the MSB of the output.
    a. Set DIP switch representing the MSB into a position and note if the LED is ON or OFF. Change the position of the dipswitch and make sure the LED changes into the opposite state. **The switch position that causes the MSB of the bar LEDS to turn on corresponds to your logic 1 position.**
    b. Verify that the LSB of the DIP switch corresponds to the LSB of bar LEDs. Also make sure the switch positions for turning the LEDs on and off correspond to those from step 1a.

2. Now configure the DIP switches for 00000000
   Since the bit twiddling operations of our program cause bit 6 to be set and bit 2 to be inverted (XORed by logic1), the output on the LEDs should correspond with 01000100

3. Try a few of your own dipswitch combinations and verify the output of your Operation.

SUMMARY

- Ports are the HCS12's built in, programmable interface devices that allow your peripherals to communicate with the HCS12
- Ports can be programmed to serve several purposes including:
    - Controlling data flow
    - Controlling data direction
- The number of programmable ports and the functionality of each available port are dependent on your HCS12's derivative
- Depending on your HCS12's derivative, some or most port lines will be physically connected to the HCS12's external pins
- Port T and AD were introduced in this guide included the general function of each port and how to program each port's registers
- Bit Twiddling was introduced in this chapter
- **For more information on ports consult the following resource:**
    - The Motorola document xxxPIMVx.pdf found on the Technological Art's resource CD or at www.technologicalarts.com the provides port names of your HCS12 derivative and their descriptions, including register names and their programmable functions.