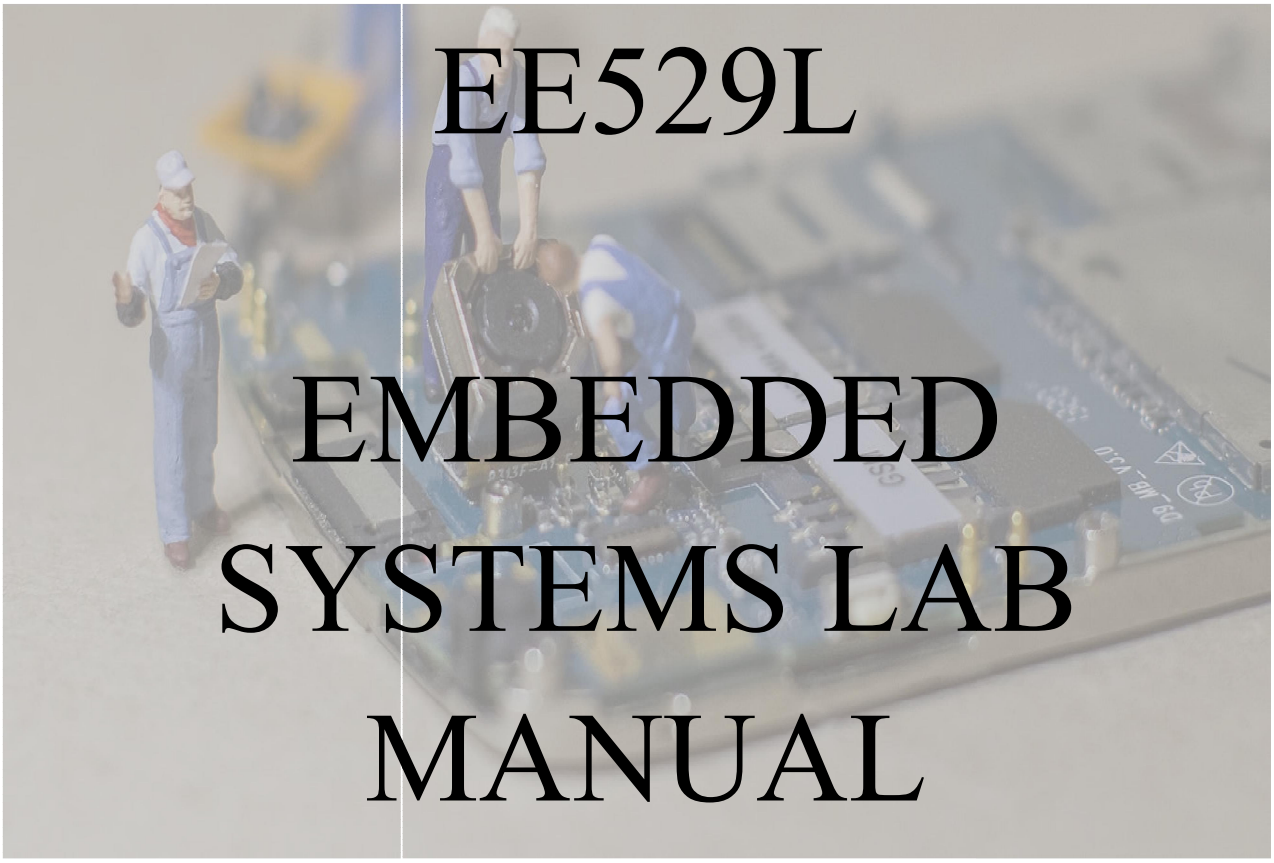
Democratic And Popular Republic Of Algeria
Ministry Of Higher Education And Scientific Research
University M'hamed Bouguerra Of Boumerdes

Institute of Electrical and Electronic Engineering



EE529L

EMBEDDED SYSTEMS LAB MANUAL

Dr. Hadjira BELAIDI
Senior Lecturer (MCB),
Departement of Electronic
Institute of Electrical and Electronics Engineering (IGEE)
University M'hamed Bougara of Boumerdes (UMBB)
35000 IGEE/ UMBB, Boumerdes, Algeria
Tel: +21391724584
E-mail: hadjira983@yahoo.fr

December, 2018.

CONTENTS

About this LAB manual

This Laboratory manual is intended for second year Master's students and computer engineers who are interested in practical embedded systems programming and problem solving. It offers insight into HCS12 microcontroller based systems implementation and programming.

Introduction

Often embedded systems are designed to be small, low weight, low power, low energy, low cost, realtime, distributed, reliable, durable, safe, and secure. In general this means the simplest systems are usually preferred and whatever hardware and software that is unnecessary is taken away from the system and its development process. Therefore, embedded engineers frequently face limited resources and deal with low-level programming as well as bit-operations on small processors and microcontrollers. Many embedded systems are bare-machine, meaning there is no operating system. Such systems are often programmed in assembly and C. The concepts that do not need to be practiced on a bare-machine, are performed on the host-machine.

This lab manual drives the student through the steps to create assembly programs using CodeWarrior software and emulate them; then, implement them in an HCS12 based educational board. In the following we start by giving an overview about the T-board used in this lab manual.

An introduction of the codeWarrior integrated development environment is given in LAB1 which will familiarize you with the general compile and download process. LAB 2 focuses on the use of both the flowchart and the algorithm procedure to describe the solution to a problem the parts of this lab are given in form of exercises. LAB 3 focuses on HCS12 parallel ports. At first, it explains how the LEDs connected to port B on the T-board can be used to post some data. Then, it describes how LEDs and switches can be connected to port A and how to write the appropriate software to define them as outputs and inputs. Lab 4 is a comprehensive exercise for HCS12 parallel port and how 7-segment displays can be connected and details also the delay subroutine creation. Lab 5 explains maskable and non-maskable interrupt based I/O operations. Lab 6 deals with the timer Input Capture and Timer Output Compare functions. Lab 7 clarifies the SCI Serial Communications module, and shows the basic steps needed to set up the HCS12 for asynchronous serial communications.

Overview

The HCS12 T-Board is a universal evaluation and training board for Motorola's advanced HCS12 16-bit microcontroller family. It provides a low-cost development platform and helps reducing development time and cost. It is a versatile tool for rapid prototyping and educational purposes.

The HCS12 T-Board is equipped with a MC9S12DP512 microcontroller unit (MCU). It contains a 16-bit HCS12 CPU, 512KB of Flash memory, 14KB RAM, 4KB EEPROM. The memory map of the microcontroller is initialized by the TwinPEEKs monitor as follows:

Begin	End	Resource
$0000	$03FF	Control Registers
$0400	$07FF	1KB(of total 4KB) EEPROM
$0800	$3FFF	14KB RAM
$4000	$7FFF	16KB Flash
$8000	$BFFF	16KB Flash page $20
$C000	$FFFF	16KB Flash

Note:
Due to a mask set erratum of the MC9S12DP512 Mask Set 4L00M (and earlier) not only the monitor code in page $3F is write protected, but also an additional area starting at $B000 up to $BFFF in page $3B. Consequently, the monitor cannot download user code to this region. However, the whole Flash memory (including the write protected areas) can be programmed using a BDM tool at any desired time.

The HCS12 T-Board is equipped also with a large amount of peripheral function blocks, such as SCI, SPI, CAN, IIC, Timer, PWM, ADC and General-Purpuse-I/Os. The MC9S12DP512 has full 16-bit data paths throughout. An integrated PLL-circuit allows adjusting performance vs. current consumption according to the needs of the user application.

In addition to the on-chip controller functions, the HCS12 T-Board module provides a number of useful peripheral components, such as RS232 and CAN transceivers, indicator elements (optical/acoustical), input devices (DIP switch, potentiometer) and a voltage regulator.

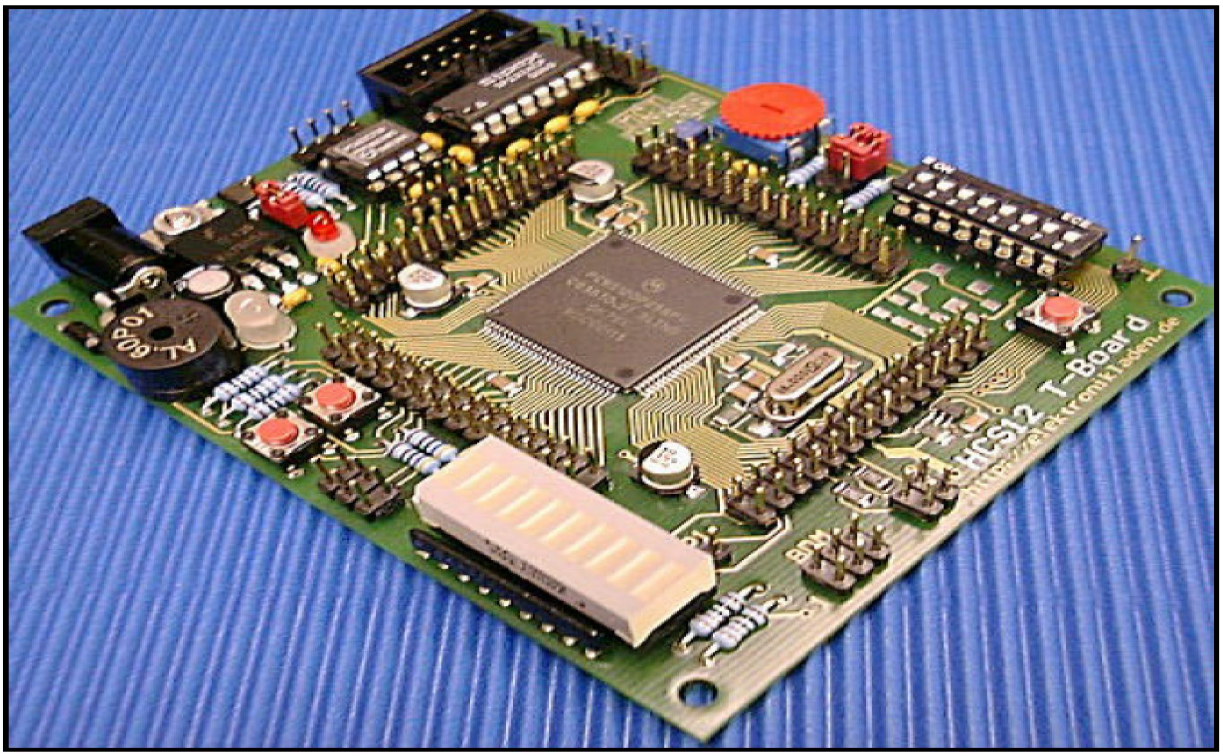
Technical Data

- MCU MC9S12DP512 with LQFP112 package (SMD)
- HCS12 16-bit CPU, uses same programming model and command set as the HC12
- 16 MHz crystal clock, up to 25 MHz bus clock using PLL

- Memory: 512KB Flash, 4KB EEPROM, 14KB RAM
- 2x SCI - asynch. serial interface (e.g. RS232, LIN)
- 3x SPI - synch. serial interface
- 1x IIC - Inter-IC-Bus
- 5x msCAN-Module (CAN 2.0A/B-compatible), one channel equipped with on-board high-speed physical interface driver
- 8x 16-Bit Timer (Input Capture/Output Compare)
- 8x PWM (Pulse Width Modulator)
- 16-channel 10-bit A/D-Converter
- BDM - Background Debug Mode Interface, std 6-pin connector
- Special LVI-circuit (reset controller)
- Serial interface with RS232 transceiver (for PC connection)
- Second serial port for IF-Modules (RS232, RS485, LIN...)
- 8x Indicator-LED, one Bi-color LED (adjustable via PWM)
- Sound transducer (buzzer)
- Reset Button
- 8x DIP switch, two push button switches
- analog input potentiometer
- up to 85 free general-purpose I/Os
- all MCU signals brought out on four header connectors around the MCU, arrangement compatible with Motorola EVB
- On-board voltage regulator generates 5V operating voltage, current consumption 50 mA typ. (plus LEDs etc.)
- Mech. Dimensions: 80mm x 95mm

Note

For more information you can refer to the "HCS12 T-Board, Hardware Version 1.00" User manual published in June 9th, 2008. The figure below is a picture of the HCS12educational T-Board.



HCS12educational T-Board

Lab 1: Creating and Building an S12(X) Project

Introduction

This lab deals with the correct way to use CodeWarrior. CodeWarrior is an IDE (Integrated Development Environment) designed to support the software development for all microcontroller products manufactured by Freescale. CodeWarrior allows the user to debug his/her software using the following three approaches:

1. Running the program using the simulator;
2. Running the program on the target hardware programmed with serial monitor (the HCS12 MCU is programmed with the serial monitor e.g. “Occonsole” or any other one);
3. Running the program on the target hardware connected to a BDM-based debug adapter (the HCS12 provides a background debug module “BDM” that allows the user to perform software debug activities (set break-point, trace program, execute program to a breakpoint or a certain location, etc) via the serial interface.

CodeWarrior has a built-in simulator that can be used by the user to debug her/his soft-ware. CodeWarrior can support software debugging via the **serial monitor**. Using this approach, the user needs to connect the demo board to the COM port of the PC using a serial cable. CodeWarrior can work with the following BDM-based adaptors to debug users’ software:

1. P&E Multilink/CyclonePro BDM adaptor
2. TBDML adaptor
3. Abatron BDI adaptor
4. Softee’s inDART debugger…

1. Purpose

This lab will guide you through the general use of the CodeWarrior™ integrated development environment and will familiarize you with the general compile and download process.

2. Building a software project using CodeWarrior

CodeWarrior can be started by clicking on its icon.

2.1. Project Setup

Step 1: Create a new project by pressing the **File menu** and select **New**…. A popup dialog box will appear to allow you to select the HCS12 device.

Step 2: Select the HCS12 device (**MC9S12DP512**) and the click on **Next**.

Step 3: Select the set of languages to be supported initially. Choose **absolute assembly** for this Lab. Enter the **project name** and the **project directory** and then click on **Next**.

Step 4: Don't select any file to be added to the new project, click on **Next** directly.

Step 5: Select **None** for the Rapid Application Development Options. No device initialization code is generated. Only generates startup code.

Step 6: Click on Finish to complete the project setup. The resultant Screen is shown in Figure 1.

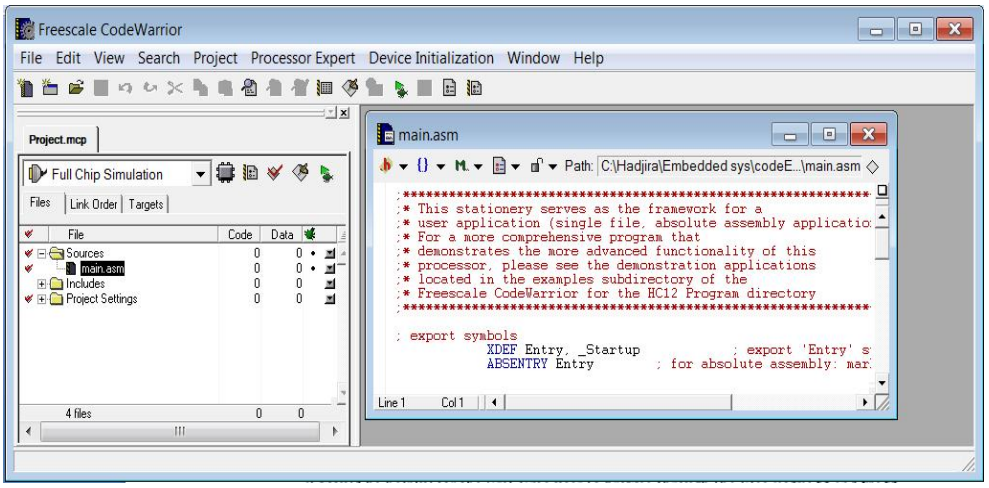


Figure 1: CodeWarrior screen after project setup.

2.2. Source Code Entering

♣ During the project creation process, CodeWarrior also creates the required files for the project and put them under different directory names (Sources, includes, Project Settings …).

♣ The user can display the file names under these directories by clicking on the '+' character to their left.

2.2.1. **Two Methods for Entering a Program**

1. Modify the main.asm program

2. Enter the program with a different name and add it to the project

2.2.2. **Example program**

Open the main.asm program (by double clicking on its name) and replace it with the following program.

```
1            XDEF          Start      ; export symbols
2            ABSENTRY      Start      ; required for absolute assembly
3            ; ORG         $0900      ; section for data in RAM
4 ;Sum:      DS.B          1          ; put the sum value here
5            ORG           $0800      ; section for code, constants
6  Start:    LDS           #$0900     ; initialize the stack pointer
7            LDAA          #$1A       ; get a value
8            ADDA          #$45       ; add to another
9            ;STAA         Sum        ; store the sum
10           END
```

2.2.3. **Adding a File into a Project**

♣ Press the right mouse button on Sources and select the file name from the directory that contains the file.

2.3. Project Build

♣ A project can be built by pressing **Project menu** and selecting **Make** or pressing the F7 function key on the keyboard.

♣ Nothing will be displayed when the project has no error.

♣ A project with errors will be displayed as shown in Figure 2.

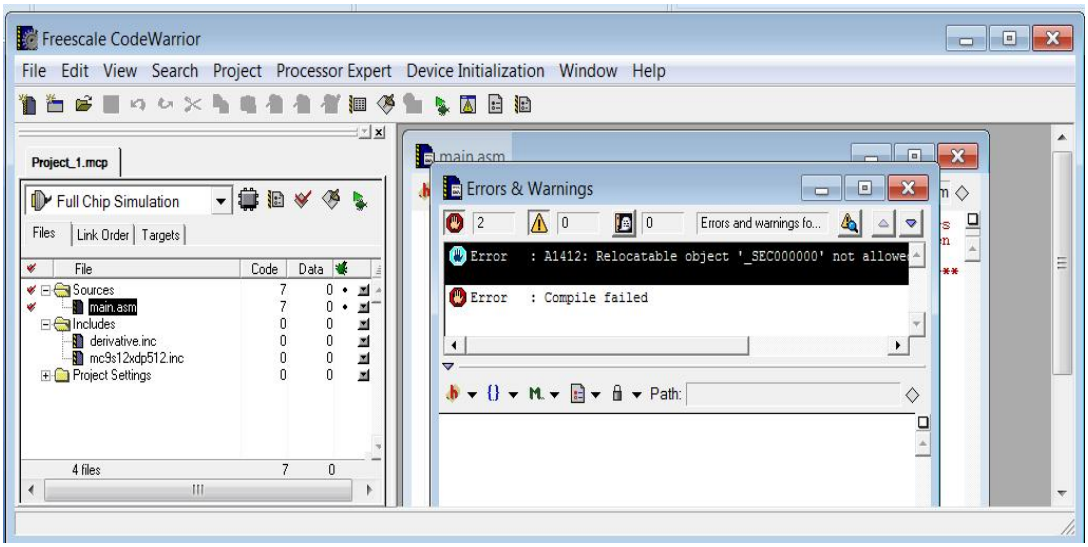


Figure 2: CodeWarrior display error messages when there are errors in the program.

- Press the function key F5 or select **Debug** from the **Project menu**.
  - What are the different viewer windows shown in the debugger? Explain their roles.
- Click on the **Step over** button once.
  - What do you remark?
- Click on the **Step over** button one more time.
  - What do you remark?
- Press the **Start/Continue** to execute the rest of the program.
  - What does this program do?
- Remove the semi-colon from the begging of the lines 3, 4 and 9 of the main program.
  - Where is the result stored?
  - Change the program to store the result in memory location 1000hex then in 1100hex.
  - Can you store the result in memory location 800hex? Explain?

3. Setting-up the T-board

- Connect the T-Board via RS232 to the lab PC using the flat ribbon cable.
- On the PC, start the **OC-Console** terminal program (can be found in the directory C:/HCS12).
- Connect a power supply to X1 power connection.
- Once powered up, the TwinPEEKs Monitor program will start, displaying a welcome message and waiting your commands.
- The OC-Console terminal program should show you a prompt that displays the current program page as follow: 20>

- Type the command H and press Enter.
- Type the command I and press Enter.
    - Explain the meaning of the information that you get.

4. Loading the *.s19 record on the T-board

- The CodeWarrior produces an output file with a ***.s19** (i.e. machine code) extension which can be loaded into and run on the HCS12. The monitor loads and records the ***.S19** into the HCS12 memory.
- Once you have successfully tested the functionality of your code using the debugger, you can now load the Project ***.abs.s19** file which contains the machine code into the HCS12 memory. (The **Project.abs.s19** file is generated in the **/bin** directory) of your project.
    - Open the ***.abs.s19** file using notepad text editor. You can observe the machine code of your assembly application in addition to its starting address. Compare the content of your ***.s19** file with the machine code you see in the Debugger Memory viewer window.
- In the **OC-Console** terminal prompt type the command **L** and press **Enter**. You should get the following: *20>L*

  loading ...
- In the top menu of the OC-Console click the **Download** icon; a file window appears so you can select the **Project.abs.s19** from the **/bin** directory. Then click on **Download**. The *.s19 machine code should now be loaded into the HCS12 memory.
- Now you can start executing your machine code just loaded by typing the following command:

  20>G 0800
    - What is the starting address of your program in memory?
    - How can you verify that your program is working correctly?
    - What do you expect to see on the memory where the result is stored? Verify it?
    - Modify the previous program to sum the two numbers 25 and 90. Store your program in memory location 850hex and the result must be stored in memory location 970hex.

Note: If you get an error about writing to some **Flash address**, just type the command **X** then press **Enter**, then press **Y**. This command will erase the flash before you load your program into it. This is because you cannot write to a Flash memory block unless it is erased (i.e. it has no data from before). Once the Flash is erased, you can load your program again and execute it as explained above.

Lab 2: Flowchart and pseudocode implementation (To solve arithmetic problems)

Introduction

Embedded system designers must spend a significant amount of time on software development; hence, a serious look at some software development issues is needed. Software development starts with *problem definition*. The problem presented by the application must be fully understood before any program can be written. Once the problem is known, the programmer can begin to lay out an overall plan of how to solve the problem. The plan is also called an *algorithm*. Informally, an algorithm is any well defined computational procedure that takes some value, or set of values, as input, and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transforms the input into the output. We can also view an algorithm as a tool for solving a well-specified computational problem. The statement of the problem specifies in general terms the desired input/output relationship. The algorithm describes a specific computational procedure for achieving that input/output relationship.

An algorithm is expressed in *pseudocode* that is very much like C or PASCAL. What separates pseudocode from "real" code is that in pseudocode, we employ whatever expressive method that is most clear and concise to specify a given algorithm. An algorithm provides not only the overall plan for solving the problem but also documentation to the software to be developed.
An earlier alternative for providing the overall plan for solving software problems was the use of flowcharts. A flowchart shows the way a program operates. It illustrates the logic flow of the program. Therefore, flowcharts can be a valuable aid in visualizing programs.

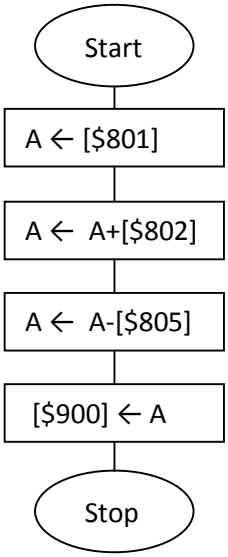
1. Purpose

The purpose of this lab is the use of both the flowchart and the algorithm procedure to describe the solution to a problem (during this lab, we deal with arithmetic operations). The lab is presented in form of exercises which must be prepared before the lab session.

After being satisfied with the algorithm or the flowchart, it must be *converted into a source code in the assembly language*. Each statement in the algorithm (or each block of the flowchart) will be converted into one or multiple assembly instructions.

Exercise N°1

1) Explain what does the program described in the following flowchart do?
2) Translate this flowchart into an assembly program.
3) Run the program in the simulator then on the T-board.



Exercise N°2

Write HCS12 assembly language code to implement the following pseudo code module.

Assume that N, NCOUNT, and PCOUNT are 8-bit variables that should be defined in memory.

IF (N < 0) THEN
Increment NCOUNT
ELSE
Increment PCOUNT
ENDIF

Exercise N°3

Write a flowchart, a pseudo-code then an assembly program to add two 16-bit numbers that are stored at $800~$801 and $802~$803, and store the sum at $900~$901.
Extend the previous program to add two 4-byte numbers that are stored at $800~$803 and $804~$807, and store the sum at $810~$813.

Exercise N°4

Develop a flowchart then write an instruction sequence to divide the signed 16-bit number stored at memory locations $805~$806 by the 16-bit unsigned number stored at memory locations $820~$821, and store the quotient and remainder at $900~$901 and $902~$903, respectively.

Lab 3: Using the HCS12 Parallel Ports (Ports A and B)

Introduction

Ports A and B are the easiest HCS12 parallel ports to understand and use. In expanded mode, both Port A and Port B are used as time-multiplexed address or data pins. When configured in single-chip mode, these two ports are used as general-purpose I/O ports. Each Port A or Port B pin can be configured as an input or output pin. When the HC12 is configured in expanded mode, Port A carries the time-multiplexed upper address and data signals (A15/D15 ~ A8/D8), whereas Port B carries the time-multiplexed lower address and data signals (A7/D7 ~A0/D0).

1. Purpose

In this lab, you will write an assembly-language program to display various patterns on the LEDs of your T-board. You will use the HCS12's Port B as an output port to display the LED patterns.

For this lab, you will create programs to write to Port B. In the first part, you will test your programs by 8 LEDs connected to Port B. In the second part, you will vary the four pins of Port A output by changing the switch settings connected to the other four pins of the same port.

2. Part 1

2.1. Pre-LAb

1) Write a program to set up Port B as an 8-bit output port.
2) Write a program to count the number of zeros contained in memory locations $900~$901 and save the result at memory location $905. Then, post the result on the LEDs.
3) Write a program to shift the 32-bit number stored at $910~$913 to the right four places. And Post the result of the least significant byte on the LEDs if it is odd; otherwise post FF on the LEDs.

Write the program before coming to lab. Be sure to write the program using structured, easy-to read code.

Note:
The address of Port B Data Register (PORTB) is $0001
The address of Port B Data Direction Register (DDRB) is $0003

2.2. The Lab

I)

1) Run your program on the simulator.
2) After your program works on the simulator, load it into your HCS12.

II) The HCS12DP512 has EEPROM (Electrically Erasable Programmable Read Only Memory) between 0x0400 and 0x0FFF. If you put your program into EEPROM the program will remain there when you turn off power. To put your program into EEPROM, all you need to do is change the starting address of your program to 0x0400. Note that you will want to store the array which has the turn signal patterns in EEPROM (so the array will not disappear when you turn off power). To do this, include any tables of constant patterns in the CODE section of your program. You will want variables which will change as the program is executed to be placed in RAM, say at location 0x0900, as usual.

When CPU starts running, it checks the state of pins AD0 and AD1. If these are both low, it runs normally. If, however, AD0 is high and AD1 is low, the CPU will immediately jump to your program in EEPROM. Thus, you can run a program without having to have the HCS12 connected to a serial port to receive a command. In order for the HCS12 to run properly, however, your program must do some hardware setup which is normally done by TwinPEEKs. To set up the hardware properly, your program needs to execute the following instructions. The reasons for doing this will be discussed later in the course.

```
        ldaa   #$55         ; Reset COP Timer Arm/Reset Register
        staa   $003f        ; COP Timer Arm/Reset Register
        coma
        staa   $003f
        clr    $003c              ; Turn off COP Control Register
        ldab    #$11        ; Map RAM into proper location
        nop
        stab   $0010        ; Initialization of Internal RAM Position Register

        ldab   #$00         ; Set clock reference divider to 0
        stab   $0035
        ldab   #$05         ; Set PLL to multiply oscillator clock by 6
        stab   $0034
        nop                 ; wait
        nop
        nop
        nop
l1:     brclr  $0037,#$08,l1 ; Wait for PLL to lock
```

```
bset    $0039,#$80 ; Switch to PLL clock
```

Add the above instructions as the very first instructions in your program. Change the address of your CODE section to 0x0400. Load the program as you have done in the past. Use the ASM command of Oc-console to verify that the code has been loaded into address 0x0400.
Be sure that a resistor is connected from GND to AD1 and a resistor from +5 V to AD0. Verify that your program runs correctly without reloading after cycling the power on your HCS12.

3. Part2

3.1. Pre-lab

Write an assembly program which allows you to read the input from the upper 4-bits (pins) of port A and post (output) their value to the lower 4-bits (pins) of port A.

3.2. The Lab

Use four pins from Port A as inputs, and the remaining four pins as outputs. Connect the inputs to switches (connect them as it is illustrated in the figure 1 bellow), and the outputs to LEDs (as shown in figure 2).

Run the program (written in the pre-lab) which allows you to read the input from the switches and post their value to the output.

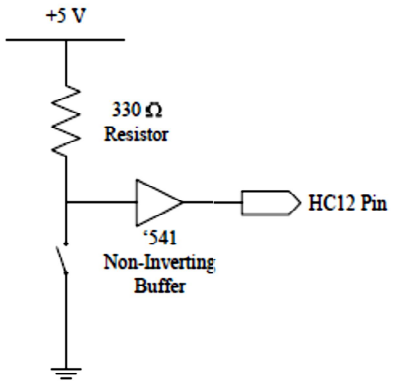


Figure 1. Switch Connection.

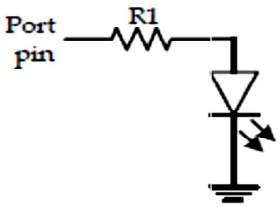


Figure 2. LED connection (R1=1kΩ)

Lab 4: 7-segment displays drive

Introduction

Freescale's HCS12 Controller Family is an upgrade from the existing 16-bit product line HC12. Compared to the HC12, the new types are faster and more flexible, while the programming model and the instruction set remain the same.

The MCU MC9S12DP512 has **25MHz bus clock rate**.

1. Purpose

This lab is a comprehensive exercise for HCS12 parallel port (7-segment displays) and delay subroutine creation.

2. Pre-Lab

1. Program an assembly code to display a sequence of BCD values on the seven-segment display. Each digit must be posted for 1s.
2. Program an assembly code to display two digits (Let's say 53) on two seven-segment displays.

3. The LAB

1. Connect your hardware as it is shown in Figure 1 bellow.
2. Connect the two output enable pins (/1OE and /2OE) to the ground.
3. Load your program to post a sequence of digits (from 0 to 9) on the 7-segments.

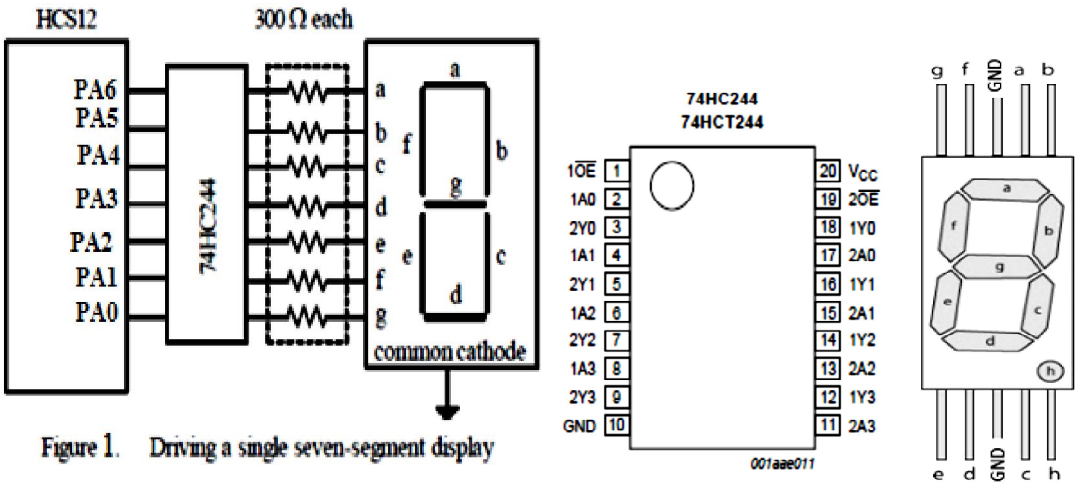


Figure 1. Driving a single seven-segment display

4. Modify your hardware to design the circuit shown in figure 2.

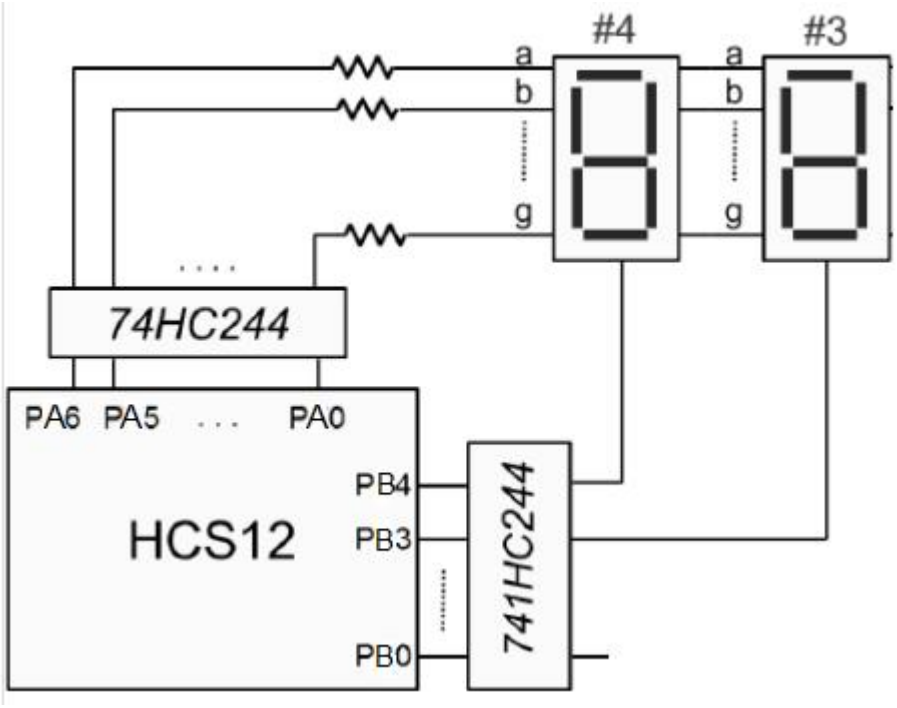


Figure 2. Connecting two 7-segments display to display two digits.

5. Modify your software so that you will be able to display two digits on two 7-sigment displays (for example display 43)

Lab 5: Interrupt (IRQ, XIRQ & Port H interrupts)

Introduction

The interrupt vectors of the HCS12 are located at the end of the 64KB memory address range, which falls within the protected monitor code space. Therefore, the application program cannot modify the interrupt vectors directly. To provide an alternative way, the monitor redirects all vectors (except the reset vector) to RAM.

The application program can set the required interrupt vectors during runtime (before global interrupt enable!) by placing a jump instruction into the RAM pseudo vector. The following example shows the steps to utilize the IRQ interrupt:

```
ldaa #$06 ; JMP opcode to
staa $3FEE ; IRQ pseudo vector
ldd #isrFunc ; ISR address to
std $3FEF ; IRQ pseudo vector + 1
```

The original vector addresses as well as the redirected addresses in RAM are given at the end of the LAB.

1. Purpose

The purpose of this lab is to understand the concept of interrupts in embedded systems. So, by doing this lab assignment, you will learn:

1. To use IRQ maskable interrupt
2. To use XIRQ non-maskable interrupt
3. Port H interrupt.

2. Pre-Lab

Part I: IRQ interrupt

1. Assume that the IRQ pin of HCS12 is connected to a push button and PortB is connected to eight LED's.
2. Write a program to configure PortB for output and enable the IRQ interrupt to respond to a falling edge.
3. Write the service routine for the IRQ interrupt. Initially $FF should be posted on PortB. Service routine for the IRQ interrupt should simply output $A1 to PortB.

Part II: XIRQ interrupt

Repeat the same steps done in part I; however, by using the XIRQ pin connected to a 1-Hz digital waveform.

Part III: PORT H interrupt

1. Port H (PTH) is connected to 8 dual-switches. Write a program to count from $00 to FF and post the count on PORTB.
2. Each number should be posted for 100ms, and so on.
3. Once the switch connected to PTH0 is toggled (rising edge), the count should stop where it is and post its value on PORTB.

3. The LAB

1. Connect IRQ pin to a push button, then load the program of part I.
2. Connect XIRQ pin to a push button, then load the program of part II.
3. Load the program of part III and test it.

4. Redirected vector addresses

The following assembly listing is part of the monitor program. It shows the original vector addresses (1st column from the left) as well as the redirected addresses in RAM (2nd column):

```
FF80  :  3F43  dc.w TP_RAMTOP-189   ; reserved
FF82  :  3F46  dc.w TP_RAMTOP-186   ; reserved
FF84  :  3F49  dc.w TP_RAMTOP-183   ; reserved
FF86  :  3F4C  dc.w TP_RAMTOP-180   ; reserved
FF88  :  3F4F  dc.w TP_RAMTOP-177   ; reserved
FF8A  :  3F52  dc.w TP_RAMTOP-174   ; reserved
FF8C  :  3F55  dc.w TP_RAMTOP-171   ; PWM Emergency
                                    ; Shutdown
FF8E  :  3F58  dc.w TP_RAMTOP-168   ; Port P
FF90  :  3F5B  dc.w TP_RAMTOP-165   ; CAN4 transmit
FF92  :  3F5E  dc.w TP_RAMTOP-162   ; CAN4 receive
FF94  :  3F61  dc.w TP_RAMTOP-159   ; CAN4 errors
FF96  :  3F64  dc.w TP_RAMTOP-156   ; CAN4 wake-up
FF98  :  3F67  dc.w TP_RAMTOP-153   ; CAN3 transmit
FF9A  :  3F6A  dc.w TP_RAMTOP-150   ; CAN3 receive
FF9C  :  3F6D  dc.w TP_RAMTOP-147   ; CAN3 errors
FF9E  :  3F70  dc.w TP_RAMTOP-144   ; CAN3 wake-up
FFA0  :  3F73  dc.w TP_RAMTOP-141   ; CAN2 transmit
FFA2  :  3F76  dc.w TP_RAMTOP-138   ; CAN2 receive
FFA4  :  3F79  dc.w TP_RAMTOP-135   ; CAN2 errors
FFA6  :  3F7C  dc.w TP_RAMTOP-132   ; CAN2 wake-up
```

```
FFA8  :   3F7F   dc.w TP_RAMTOP-129   ; CAN1 transmit
FFAA  :   3F82   dc.w TP_RAMTOP-126   ; CAN1 receive
FFAC  :   3F85   dc.w TP_RAMTOP-123   ; CAN1 errors
FFAE  :   3F88   dc.w TP_RAMTOP-120   ; CAN1 wake-up
FFB0  :   3F8B   dc.w TP_RAMTOP-117   ; CAN0 transmit
FFB2  :   3F8E   dc.w TP_RAMTOP-114   ; CAN0 receive
FFB4  :   3F91   dc.w TP_RAMTOP-111   ; CAN0 errors
FFB6  :   3F94   dc.w TP_RAMTOP-108   ; CAN0 wake-up
FFB8  :   3F97   dc.w TP_RAMTOP-105   ; FLASH
FFBA  :   3F9A   dc.w TP_RAMTOP-102   ; EEPROM
FFBC  :   3F9D   dc.w TP_RAMTOP-99    ; SPI2
FFBE  :   3FA0   dc.w TP_RAMTOP-96    ; SPI1
FFC0  :   3FA3   dc.w TP_RAMTOP-93    ; IIC
FFC2  :   3FA6   dc.w TP_RAMTOP-90    ; BDLC
FFC4  :   3FA9   dc.w TP_RAMTOP-87    ; Self Clock Mode
FFC6  :   3FAC   dc.w TP_RAMTOP-84    ; PLL Lock
FFC8  :   3FAF   dc.w TP_RAMTOP-81    ; Pulse Accu B Overflow
FFCA  :   3FB2   dc.w TP_RAMTOP-78    ; MDCU
FFCC  :   3FB5   dc.w TP_RAMTOP-75    ; Port H
FFCE  :   3FB8   dc.w TP_RAMTOP-72    ; Port J
FFD0  :   3FBB   dc.w TP_RAMTOP-69    ; ATD1
FFD2  :   3FBE   dc.w TP_RAMTOP-66    ; ATD0
FFD4  :   3FC1   dc.w TP_RAMTOP-63    ; SCI1
FFD6  :   3FC4   dc.w TP_RAMTOP-60    ; SCI0
FFD8  :   3FC7   dc.w TP_RAMTOP-57    ; SPI0
FFDA  :   3FCA   dc.w TP_RAMTOP-54    ; Pulse Accu A Input Edge
FFDC  :   3FCD   dc.w TP_RAMTOP-51    ; Pulse Accu A Overflow
FFDE  :   3FD0   dc.w TP_RAMTOP-48    ; Timer Overflow
FFE0  :   3FD3   dc.w TP_RAMTOP-45    ; TC7
FFE2  :   3FD6   dc.w TP_RAMTOP-42    ; TC6
FFE4  :   3FD9   dc.w TP_RAMTOP-39    ; TC5
FFE6  :   3FDC   dc.w TP_RAMTOP-36    ; TC4
FFE8  :   3FDF   dc.w TP_RAMTOP-33    ; TC3
FFEA  :   3FE2   dc.w TP_RAMTOP-30    ; TC2
FFEC  :   3FE5   dc.w TP_RAMTOP-27    ; TC1
FFEE  :   3FE8   dc.w TP_RAMTOP-24     ; TC0
FFF0  :   3FEB   dc.w TP_RAMTOP-21    ; RTI
FFF2  :   3FEE   dc.w TP_RAMTOP-18    ; IRQ
FFF4  :   3FF1   dc.w TP_RAMTOP-15    ; XIRQ
FFF6  :   3FF4   dc.w TP_RAMTOP-12    ; SWI
FFF8  :   3FF7   dc.w TP_RAMTOP-9     ; Illegal Opcode
FFFA  :   3FFA   dc.w TP_RAMTOP-6     ; COP Fail
FFFC  :   3FFD   dc.w TP_RAMTOP-3     ; Clock Monitor Fail
FFFE  :   F000   dc.w main            ; Reset
```

Lab 6: I/O Timer Functions

Introduction

The purpose of the timer module is to allow for time critical operations to be handled mostly by hardware, instead of entirely in software. For example, generating or measuring waveforms can be done with minimal input from the processor using the timer module.

The 68HC12 Standard Timer Module consists of a 16-bit programmable timer that is driven by a programmable prescaler mechanism. It also has eight 16-bit input capture/output compare channels, and two pulse accumulators. This hardware is explained in its datasheet.

Before anything happens in the Standard Timer Module, the software on the CPU must enable the timer system by setting bits in the appropriate registers. Table 1 summarizes the studied MC9S12DP512 microcontroller timer register addresses.

1. Purpose

In this lab you will work with the timer Input Capture and Timer Output Compare functions. To demonstrate their usage you will design a program to measure the period of a signal and you will generate a 2 KHz waveform. Thus, you will:

1- Learn and understand how to use timer functions and the underlying mechanism for setting up real time counters.
2- Understand the basic idea of interrupts.

2. Pre-Lab

Part I: Input Capture

To measure the period of an unknown signal, the input-capture function should be configured to capture the timer values corresponding to two consecutive rising or falling edges.

The circuit connection for the period measurement is shown in Figure 1. Give the logic flowchart, and write an assembly program for the period measurement. (Capture the timer values corresponding to two consecutive rising edges).

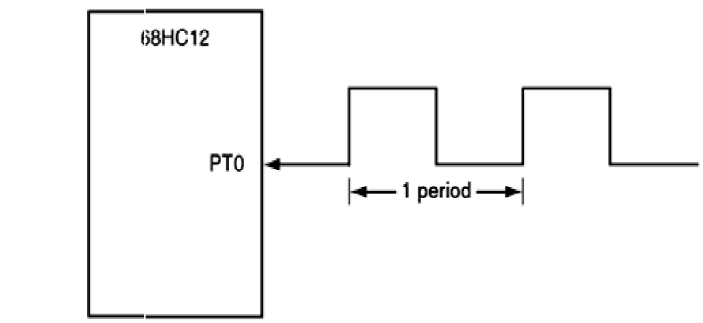


Figure 1. Period measurement signal connection.

Part II: Output compare

Suppose we want to generate a 2 KHz digital waveform with a 40 percent duty cycle on the PT0 pin. Assume that the E-clock frequency is 25 MHz.

1. By setting the prescale factor to 8. What will be the period of the generated signal?
2. What is the high interval of one period?
3. What is the low interval?
4. Use OC0 to generate the 2 KHZ waveform. Initially, write the program without using interrupt.
5. Rewrite the program by using interrupt-driven approach so that the CPU can perform other operations.

3. The LAB

1. Connect PT0 to a digital-waveform generator as shown in figure 1. Set its frequency to a random value; then, load the program of part I to measure the period of the sensed signal and post its value on PORTB.

2. Connect PT0 pin to an oscilloscope. Load the program of part II to generate a 2 KHz waveform without and with interrupt.

Table 1. MC9S12DP512 microcontroller timer register addresses

$0040 - $007F ECT (Enhanced Capture Timer 16 Bit 8 Channels)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0040	TIOS	Read:	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0
		Write:								
$0041	CFORC	Read:	0	0	0	0	0	0	0	0
		Write:	FOC7	FOC6	FOC5	FOC4	FOC3	FOC2	FOC1	FOC0
$0042	OC7M	Read:	OC7M7	OC7M6	OC7M5	OC7M4	OC7M3	OC7M2	OC7M1	OC7M0
		Write:								
$0043	OC7D	Read:	OC7D7	OC7D6	OC7D5	OC7D4	OC7D3	OC7D2	OC7D1	OC7D0
		Write:								
$0044	TCNT (hi)	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0045	TCNT (lo)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0046	TSCR1	Read:	TEN	TSWAI	TSFRZ	TFFCA	0	0	0	0
		Write:								
$0047	TTOV	Read:	TOV7	TOV6	TOV5	TOV4	TOV3	TOV2	TOV1	TOV0
		Write:								
$0048	TCTL1	Read:	OM7	OL7	OM6	OL6	OM5	OL5	OM4	OL4
		Write:								
$0049	TCTL2	Read:	OM3	OL3	OM2	OL2	OM1	OL1	OM0	OL0
		Write:								
$004A	TCTL3	Read:	EDG7B	EDG7A	EDG6B	EDG6A	EDG5B	EDG5A	EDG4B	EDG4A
		Write:								
$004B	TCTL4	Read:	EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A
		Write:								
$004C	TIE	Read:	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I
		Write:								
$004D	TSCR2	Read:	TOI	0	0	0	TCRE	PR2	PR1	PR0
		Write:								
$004E	TFLG1	Read:	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F
		Write:								

$004F	TFLG2	Read: Write:	TOF	0	0	0	0	0	0	0
$0050	TC0 (hi)	Read: Write:	Bit 15	14	13	12	11	10	9	Bit 8
$0051	TC0 (lo)	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$0052	TC1 (hi)	Read: Write:	Bit 15	14	13	12	11	10	9	Bit 8
$0053	TC1 (lo)	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$0054	TC2 (hi)	Read: Write:	Bit 15	14	13	12	11	10	9	Bit 8
$0055	TC2 (lo)	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$0056	TC3 (hi)	Read: Write:	Bit 15	14	13	12	11	10	9	Bit 8
$0057	TC3 (lo)	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$0058	TC4 (hi)	Read: Write:	Bit 15	14	13	12	11	10	9	Bit 8
$0059	TC4 (lo)	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$005A	TC5 (hi)	Read: Write:	Bit 15	14	13	12	11	10	9	Bit 8
$005B	TC5 (lo)	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$005C	TC6 (hi)	Read: Write:	Bit 15	14	13	12	11	10	9	Bit 8
$005D	TC6 (lo)	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$005E	TC7 (hi)	Read: Write:	Bit 15	14	13	12	11	10	9	Bit 8
$005F	TC7 (lo)	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0

Lab 7: Serial Communication Interface

Introduction

There are two types of serial communications modules on the HCS12 chips: the SPI module and the SCI module. The Serial Peripheral Interface (SPI) is a synchronous (clocked) module used for high-speed communications over short distances with other devices that have an SPI interface. We might use the SPI to communicate with a high-speed, high-resolution A/D converter, for instance. Because of signal degradation and critical timing issues with clock skew, the SPI is not suitable for long-distance communications. Usually the SPI is used for devices on the same printed circuit board as the microcontroller only 1 to 10 inches away.

The Serial Communications Interface (SCI) module is a relatively slow, asynchronous communication port that is widely used to communicate with other embedded systems and devices. The RS-232 interface to the T-board, for instance, uses an SCI interface. The SCI is more widely used than the SPI, and can communicate over longer distances than the SPI. We will use the SCI module in this Lab. The functional diagram of the Serial Communications Interface (SCI) is given in figure 1.

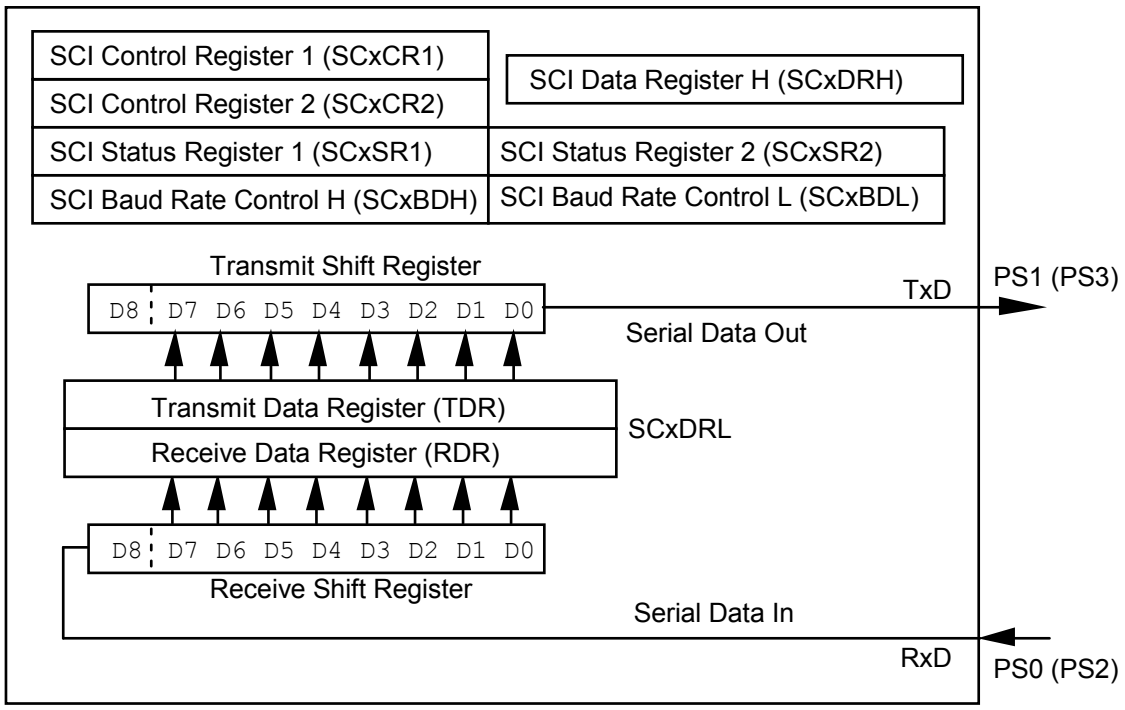


Figure 1. Functional diagram of the Serial Communications Interface (SCI)

1. Purpose

This lab is based on using the SCI Serial Communications module, and will show the basic steps needed to set up the HCS12 for asynchronous serial communications.

You will write a simple program that continuously transmits a string of characters from a T-board, and then add some code so that you can receive and store the characters as well.

2. Pre-Lab

1) Draw the waveform you would see when sending the character 'A' out of the transmit SCI port. Show the whole frame including start, stop and data bits, and include a time axis (assume we are transmitting at 19200 baud).

2) Write an instruction sequence that configures SCI0 for 19200 baud, 8-data bit, 1 stop bit, no parity with the 25-MHz E–clock.

3) Enable the RX register (and also the TX register of course).

4) Write a subroutine to transmit characters of the string 'Code Warrior' from SCI0 in an infinite loop.

5) Write a subroutine to input a string from SCI0. The string is terminated by the carriage return character and must be stored in a buffer pointed to by index register X. The register addresses are given in the following table (Table 1).

Table 1. SCI Registers in the MC9S12DP512

Name	Register Addr	Description
SC0BDH	00C8	SCI Baud Rate Control Register High
SC0BDL	00C9	SCI Baud Rate Control Register Low
SC0CR1	00CA	SCI Control Register 1
SC0CR2	00CB	SCI Control Register 2
SC0SR1	00CC	SCI Status Register 1
SC0SR2	00CD	SCI Status Register 2
SC0DRH	00CE	SCI Data Register High
SC0DRL	00CF	SCI Data Register Low

3. The LAB

Run the pre-written programs and verify their functionality.