# Notes - Unit 5

## HARDWARE AND SOFTWARE DEVELOPMENT TOOLS FOR THE HCS12

- **Hardware**: Dragon12-Light Board pre-loaded with Freescale Serial Monitor for CodeWarrior. The board includes the MC9S12DG256BCPV (112 LQFP package). The Dragon12-Light Board Manual contains a lot of information on the components connections to the microprocessor. Among many other components, the Dragon12-Light Board includes:
  - ✓ Two 10-bit DAC for testing SPI interface and generating analog waveforms
  - ✓ Four 7-segment displays
  - ✓ Eight LEDs
  - ✓ An eight-position DIP Switch
  - ✓ Four push button switches
  - ✓ Speaker to be driven by timer, DAC, or PWM
  - ✓ 4x4 keypad (made of push button switches)
  - ✓ CAN Port
  - ✓ RGB color LED

  The crystal frequency is 8 MHz and this results in a 4 MHz bus speed. However, we can boost the bus speed to 24 MHz by configuring the PLL.

- **Software**: CodeWarrior for HCS12(X) Microcontrollers (Classic) v. 5.1 (Special Edition). We can create projects in purely Assembly Code, in C, or in mixed C/Assembly. We will use the software in two modes:
  - ✓ Full Chip Simulation: The Board is not required to be connected. The register and memory data are simulated values.
  - ✓ HCS12 Serial Monitor: The Board is required to be connected. The registers and memory data are the actual ones.

## I/O REGISTERS

The Device User Guide shows the detailed register map (Section 1.6) of the MC9S12DG256 from `$0000` to `$03FF` (1KB). The entire memory map of the device is shown in Figure 1.2. Here we will use some of the registers that are linked to the most common components in the Dragon12-Light Board. The `mc9s12dg256.inc` file contains all the numeric equivalent of the port numbers (and memory positions too):

### PORTA

This is General Purpose I/O (GPIO) register located at `$0000`. In the Board, it is connected to the keypad. If we use the keypad, we must configure PORTA so that the bits 7 to 4 are outputs and the bits 3 to 0 are inputs. This is done via the DDRA register located at `$0002`.
For Keypad: `DDRA ← $F0 = 1111000`. '1' means output, '0' means input

### PORTB

This is General Purpose I/O (GPIO) register located at `$0001`. In the Board, it is connected to the LEDs. If we use the LEDs, we must configure PORTB so that the all the bits are outputs. This is done via the `DDRB` register located at `$0003`.
For LEDs and 7-segment displays: `DDRB ← $FF`

### PORTP

This is General Purpose I/O (GPIO) register located at `$0258`. In the Board, the bits 3 to 0 are connected to the cathodes of the 7-segment displays. The bits 4 to 6 are connected to the RGB LED. To use it with the 7-segment display and the RGB LED, we must configure `PTP` so that the bits are outputs. This is done via the DDRP register located at `$025A`.
For 7-segment displays and RGB LED: `DDRP ← $FF`

### PORTH

This is General Purpose I/O (GPIO) register located at `$0260`. In the Board, it is connected to the DIP Switch and Push Buttons (the last 4 bits). To use the DIP Switch and the Push Buttons, we must configure `PTH` so that the bits are inputs. This is done via the `DDRH` register located at `$0262`.
For DIP Switches: `DDRH ← $00`

### PORTK

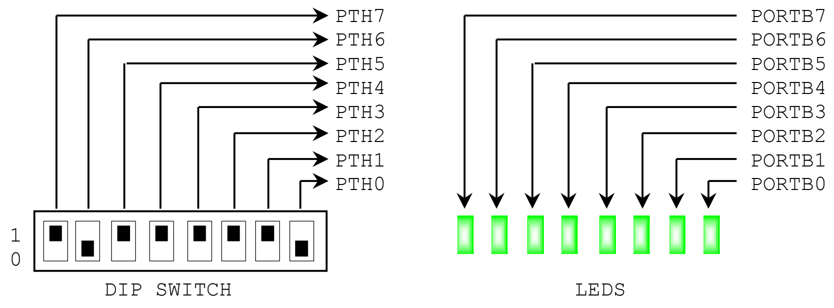
This is General Purpose I/O (GPIO) register located at `$0032`. In the Board, it is connected to the LCD controller. To use the LCD Controller, we must configure `PORTK` so that some bits are outputs and other inputs. This is done via the `DDRK` register located at `$0033`.

## EXAMPLES

- Read DIP Switches and place the result on the LEDs

```
DDRH ← 00
DDRB ← FF
while (1)
     A ← PORTH
     PORTB ← A
end
```

**ASM Code**: `unit5a.asm`

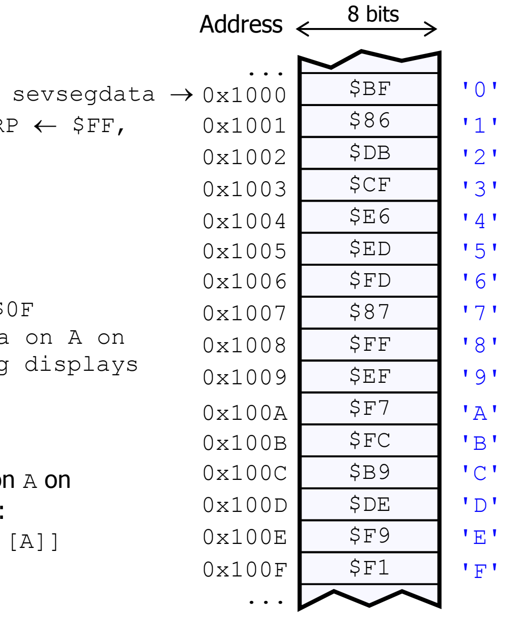


- Read 4-bit data from DIP Switches (the last 4 LSBs) and display the hexadecimal value on all 7-segment displays.

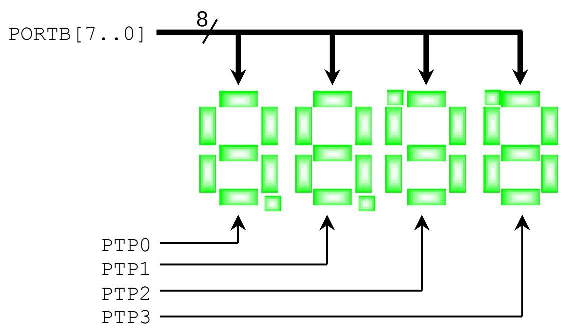
```
DDRB ← $FF, DDRP ← $FF,
DDRH ← $00
X ← sevsegdata
PTP ← $00
while (1)
     A ← PORTH
     A ← A AND $0F
     Display data on A on
        all 7-seg displays
     PORTB ← B
end
```

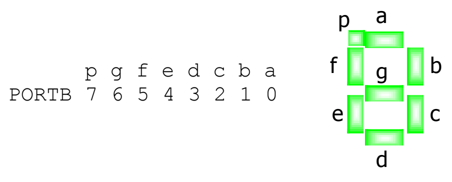
* To Display data on `A` on all 7-seg displays:

```
     B ← [[X] + [A]]
     PORTB ← B
```


Address
8 bits

| | Address | 8 bits | |
|---|---|---|---|
| | ... | | |
| sevsegdata → | 0x1000 | $BF | '0' |
| | 0x1001 | $86 | '1' |
| | 0x1002 | $DB | '2' |
| | 0x1003 | $CF | '3' |
| | 0x1004 | $E6 | '4' |
| | 0x1005 | $ED | '5' |
| | 0x1006 | $FD | '6' |
| | 0x1007 | $87 | '7' |
| | 0x1008 | $FF | '8' |
| | 0x1009 | $EF | '9' |
| | 0x100A | $F7 | 'A' |
| | 0x100B | $FC | 'B' |
| | 0x100C | $B9 | 'C' |
| | 0x100D | $DE | 'D' |
| | 0x100E | $F9 | 'E' |
| | 0x100F | $F1 | 'F' |
| | ... | | |

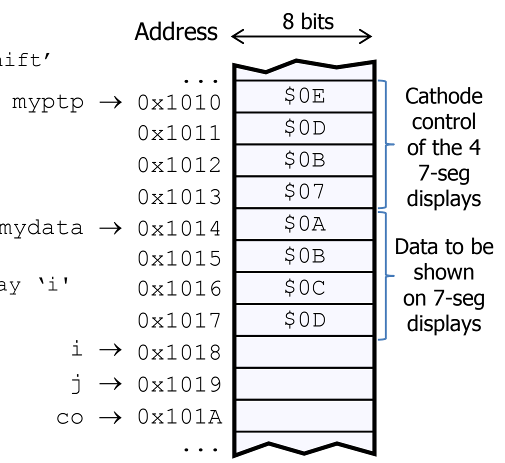




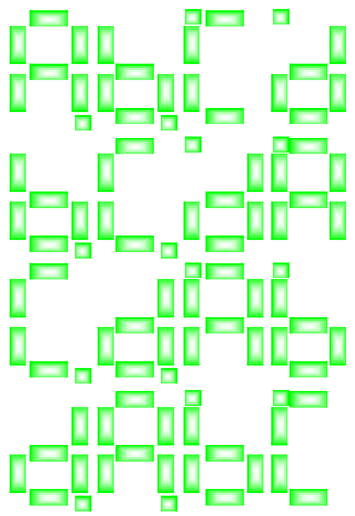
**ASM Code**: `unit5b.asm`

- Read 4 bytes of data from memory and display the hexadecimal values on the four 7-segment displays. Then, rotate the 4 digits to the left every 1 second. For visual persistence, display each digit for only 1 ms. This delay was computed using a bus speed of 24 MHz (this requires an extra piece of code, this code does not seem to run on the Debugger step by step).

```
X ← sevsegdata
Y ← myptp
co ← 0 ; This one indicates the 'shift'
while (1)
   for j = 0 to 249
      for i = 0 to 3
         idx ← i + co
         if i+co ≥ 4 then
            idx ← i+co-4
         end
         Display data on
               'idx+mydata' on display 'i'
         Delay by 1 ms
         end
   end
   co ← co+1
   if co ≥ 4 then
      co ← 0
   end
end
```


Address
8 bits
Cathode control of the 4 7-seg displays
Data to be shown on 7-seg displays

| | Address | 8 bits |
|---|---|---|
| | ... | |
| myptp → | 0x1010 | $0E |
| | 0x1011 | $0D |
| | 0x1012 | $0B |
| | 0x1013 | $07 |
| mydata → | 0x1014 | $0A |
| | 0x1015 | $0B |
| | 0x1016 | $0C |
| | 0x1017 | $0D |
| i → | 0x1018 | |
| j → | 0x1019 | |
| co → | 0x101A | |
| | ... | |

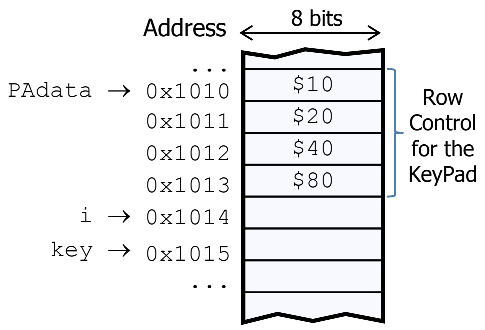


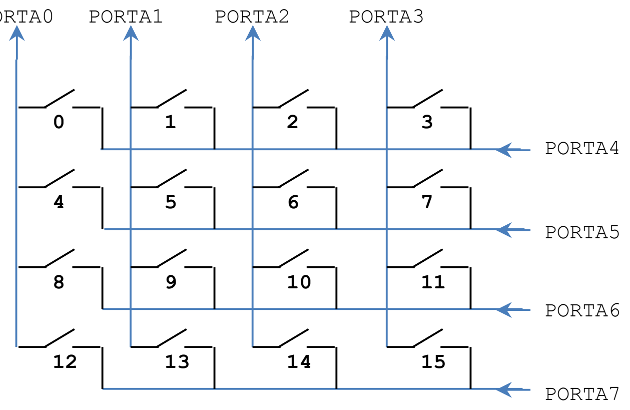
**ASM Code**: `unit5c.asm`

- Read the key pad, and display the hexadecimal value on all 7-segment displays.

| | Output | Input | | | |
|---|---|---|---|---|---|
| | PORTA[7..4] | PORTA[3..0] = 0001<br>idx = 0 | PORTA[3..0] = 0010<br>idx = 1 | PORTA[3..0] = 0100<br>idx = 2 | PORTA[3..0] = 1000<br>idx = 3 |
| i=0 | 0001 | KEY0 | KEY1 | KEY2 | KEY3 |
| i=1 | 0010 | KEY4 | KEY5 | KEY6 | KEY7 |
| i=2 | 0100 | KEY8 | KEY9 | KEY10 | KEY11 |
| i=3 | 1000 | KEY12 | KEY13 | KEY14 | KEY15 |

⇒ KEY = 4*i + idx





| Main Routine | getidx subroutine. Input: A(3..0) |
|---|---|
| see below | see below |

```
Main Routine
DDRA ← $F0, DDRB ← $FF, DDRP ← $FF, DDRH ← $00
X ← sevsegdata
PTP ← $00
Y ← PAdata
while (1)
   for i = 0 to 3
      PORTA ← [PAdata + i]
      Delay 1 ms
      A ← PORTA ; Read PORTA
      idx = getidx (A(3..0))
      If idx = 15 then
         Display blank on all 7-seg displays
      else
         key ← i*4 + idx ;
         Display hex value of 'key' on all 7-seg displays
      end
   end
end
```

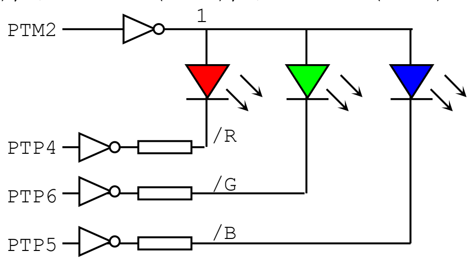
```
getidx subroutine. Input: A(3..0)

if A(3..0) = 1000
   idx = 3
elsif A(3..0) = 0100
   idx = 2
elsif A(3..0) = 0010
   idx = 1
elsif A(3..0) = 0001
   idx = 0
else
   idx = 15
end if
```

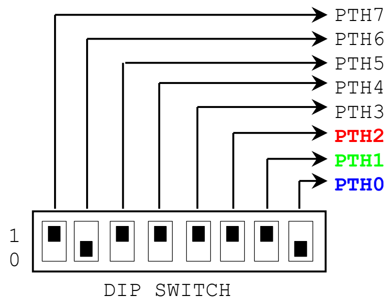
**ASM Code**: unit5d.asm

- RGB LED control via DIP switch. PORTP4 = RED LED cathode, PORTP5 = BLUE LED cathode, PORTP6 = GREEN LED cathode. The anodes of these 3 LEDs are connected to PTM2. All the pins that control the LEDs are in **negative logic**.

Input: PTH[2..0]. PTH2 = Red, PTH1 = Green, PTH0 = Blue
Output: PTP[6..4]. /R = not (PTP4), /G = not (PTP6), /B = not (PTP5).

| R | G | B | Color |
|---|---|---|---|
| 0 | 0 | 0 | Black |
| 0 | 0 | 1 | Blue |
| 0 | 1 | 0 | Green |
| 0 | 1 | 1 | Cyan |
| 1 | 0 | 0 | Red |
| 1 | 0 | 1 | Magenta |
| 1 | 1 | 0 | Yellow |
| 1 | 1 | 1 | White |





```
DDRH ← $00, DDRM ← $FF, PTM ← $00; PTM2 = 0: Sets the anodes to '1'
while (1)
     A ← PTH
     A ← A AND $07; A stores only the 3 LSBs or PORTH
     temp(4) ← A(2), temp(6) ← A(1), temp(5) ← A(0); Rewires the bits of PTH[2..0] into PTP[6..4]
     PTP ← temp ; PTP controls the cathodes. If a bit in PTP is '1', then that cathode is '0'
end
```

**ASM Code**: unit5e.asm