Overview of HCS12 Parallel Ports

- A HCS12 device may have from 48 to 144 pins arranged in 3 to 12 I/O ports and packaged either a *quad flat pack* (QFP) or a *low-profile quad flat pack* (LQFP) package.
- The pins of an I/O port are often multiplexed with one or more peripheral functions (e.g., timer or serial peripheral interface).
- When a peripheral function is enabled, that pin may not be used as a general-purpose I/O pin.
- General-purpose I/O pins can be considered the simplest of peripherals. They allow the microcontroller to monitor and control other devices.

Overview of HCS12 Parallel Ports *Cont'd …*

- I/O ports do not have the same number of pins. The number of pins available in each I/O port for the HCS12 MCU is listed below:

Port Name	**No. of Pins**	**Pin Name**
A	8	PA7~PA0
B	8	PB7~PB0
E	8	PE7~PE0
H	8	PH7~PH0
J	4	PJ7~PJ0
K	7	PK7~PK0
M	8	PM7~PM0
P	8	PP7~PP0
S	8	PS7~PS0
T	8	PT7~PT0
PAD1,PAD0	16	PAD15~PAD0
L	8	PL7~PL0
U	8	PU7~PU0
V	8	PV7~PV0
W	8	PW7~PW0

Overview of HCS12 Parallel Ports *Cont'd …*

- Each I/O port has several *associated registers* to support its operation and these registers must be dealt with when performing I/O operations by specifying the address of a register in order to read from it or write into it.
- A better way is to use the EQU directive (in assembly language) to equate a *symbolic name* to the address of a given register so that the *symbolic name* can be used to access it.
- For example:

```
PORTA       EQU       $00
            MOVB      #$FF,PORTA
```

Overview of HCS12 Parallel Ports *Cont'd …*

- The file *Reg9S12.h* that contains the **EQU** directive for all peripheral registers has been provided on the *example* folder of DRAGON12.
- Include this file with your program, then you will be able to use symbolic names to access all peripheral registers.
- For example:

```
#include        C:\DARGON12\examples\Reg9S12.h
                ORG             $1000
                 ⋮
```

Port Registers

- All I/O ports (except PAD0 & PAD1) have an associated *data direction register* (DDR) and a *data register*.
- To configure a pin for output, write a 1 to the corresponding bit in the *data direction register*.
- To configure a pin for input, write a 0 to the corresponding bit in the *data direction register*.

```
MOVB  #$00,DDRA     ;configure port A for input
LDAA  PORTA         ;reads a byte into A from port A
MOVB  #$FF,DDRB     ;configure port B for output
MOVB  #$A5,PORTB    ;output $A5 to port B
```

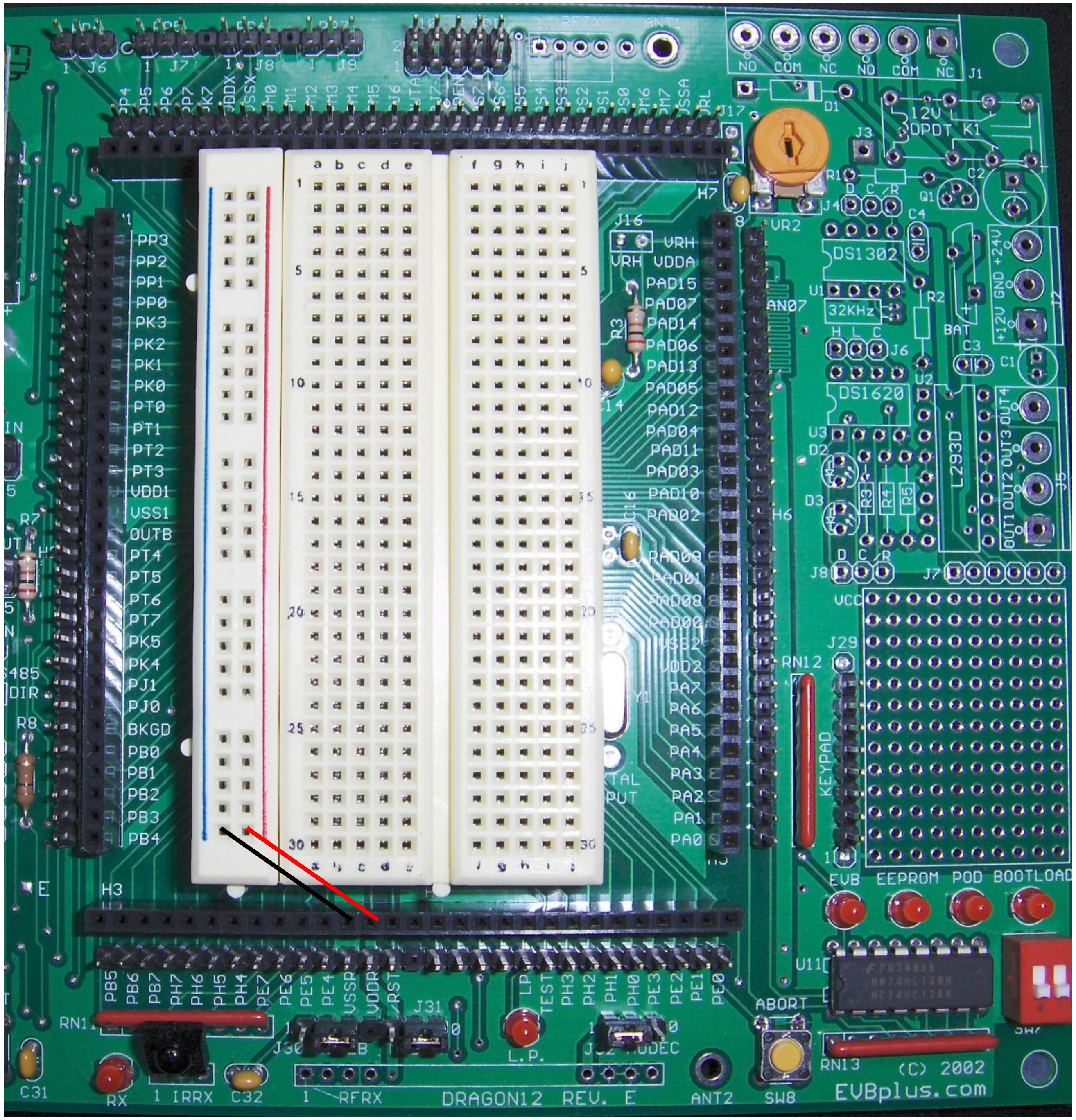
Port Registers *Cont'd …*

- A complete list of *symbolic* names for port data registers is shown in the following table:

Port Name	**Data Register Name**
A	PORTA[1]
B	PORTB[1]
E	PORTE[1]
H	PTH
J	PTJ
K	PORTK[1]
M	PTM
P	PTP
S	PTS
T	PTT
PAD1,PAD0	PORTAD1,PORTAD0
L	PTL[2]
U	PTU[2]
V	PTV[2]
W	PTW[2]

[1] PORTA, PORTB, PORTE, and PORTK are also referred to as PTA, PTB, PTE, and PTK in the *Reg9S12.h* file.

[2] Port L, U, V, & W are available in H-family devices only.



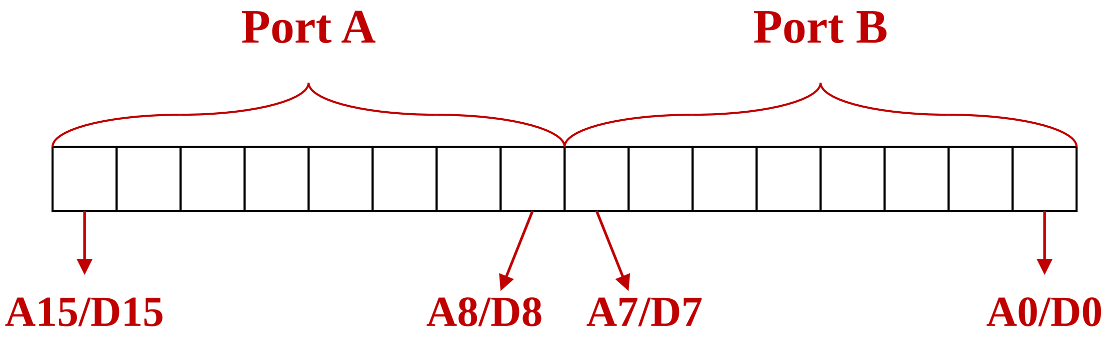
Port A & B:

➢ when configured in *single-chip mode*, these two ports are used as general-purpose I/O ports.

➢ Every single pin can be configured as input or output.

➢ In *expanded mode*, both port A & B are used as time-multiplexed address/data pin.

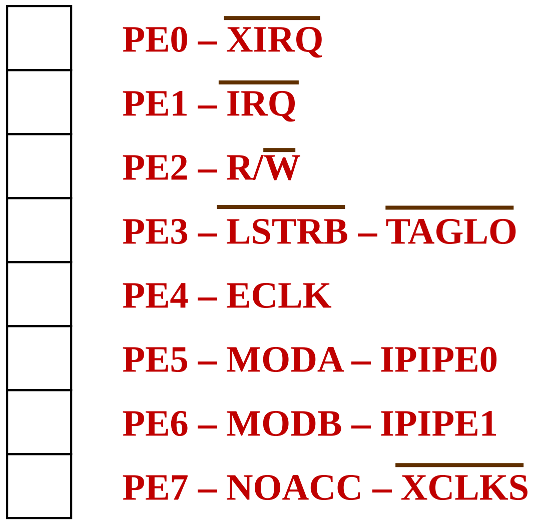
➢ In *expanded mode*, port A carries the time-multiplexed upper address and data signals, whereas port B carries the time-multiplexed lower address and data signals.



Port E:

- Port E pins are used for *bus control* and *interrupt* service request signals.
- When the pin is not used for one of these special functions, it can be used as a general-purpose I/O.
- Two of the port E pins (PE[1:0]) can only be used for input, and the states of these pins can be read in the *port data register* even when they are used for IRQ and XIRQ.

Port E pins and their alternate functions


PE0 – $\overline{\text{XIRQ}}$
PE1 – $\overline{\text{IRQ}}$
PE2 – R/$\overline{\text{W}}$
PE3 – $\overline{\text{LSTRB}}$ – $\overline{\text{TAGLO}}$
PE4 – ECLK
PE5 – MODA – IPIPE0
PE6 – MODB – IPIPE1
PE7 – NOACC – $\overline{\text{XCLKS}}$

Port E *Cont'd …*

The PE7 pin serves three functions:

1. A general-purpose I/O pin
2. A signal that indicates the MCU is not accessing external memory
3. A signal that selects between the *external clock* or *crystal oscillator* signal as the clock input to the HCS

The PE6 and PE5 pins serve three functions:

1. A general-purpose I/O pins (in single-chip mode)
2. Signals that set the operation mode of the HCS12 after reset
3. Instruction queue tracking signals (in expanded mode)

The PE4 pin has two functions:

1. A general-purpose I/O pin (in single-chip mode)
2. The E clock output (in expanded mode)

Port E *Cont'd …*

The PE3 pin has three functions:

1. A general-purpose I/O pin (in single-chip mode)
2. The LCD front-plane segment driver output pin (in H family)
3. The low-byte strobe signal to indicate the type of access (in expanded mode)

The PE2 pin has two functions:

1. A general-purpose I/O pins (in single-chip mode)
2. A signal to indicate whether the current bus cycle is a read cycle or a write cycle (in expanded mode)

The PE1 pin has two functions:

1. A general-purpose I/O pin
2. The $\overline{\text{IRQ}}$ input

The PE0 pin has two functions:

1. A general-purpose I/O pin
2. The $\overline{\text{XIRQ}}$ input

Port E Registers:

In addition to the DDRE and PORTE registers, port E also has the following registers:

- Port E assignment register (PEAR)
- Mode register (MODE)
- Pull-up control register (PUCR)
- Reduced drive register (RDRIV)
- External bus interface control register (EBICTL)

Port E Assignment Register (PEAR)

In expanded mode, this register assigns the function of each port E pin. The PEAR register is not accessible for reads or writes in peripheral mode. The pin assignment for PEAR register is as follows:

Port E Registers *Cont'd …*

7	6	5	4	3	2	1	0
NOACCE	0	PIPOE	NECLK	LSTRE	0	RDWE	0

NOACCE: *No Access output enable.* Can be read/written any time.

0 = PE7 is used as general-purpose I/O pin.

1 = PE7 is output and indicates whether the cycle is a CPU free cycle.

PIPOE: Pipe signal output enable.

In normal modes, write once. Special modes, write anytime except the first time. This bit has no effect on single-chip mode.

0 = PE[6:5] are general-purpose I/O.

1 = PE[6:5] are outputs and indicate the state of the instruction queue.

NECLK: *No External E clock.* Can be read any time.

In expanded modes, write to this bit has no effect. E clock is required for de-multiplexing the external address. NECLK can be written once in normal single-chip mode and can be written anytime in special single-chip mode.

0 = PE4 is the external E clock.

1 = PE4 is a general-purpose I/O pin.

Port E Registers *Cont'd …*

LSTRE: *Low strobe (LSTRB) enable*. Can be read any time.

In normal modes, write once. Special modes, write anytime. This bit has no effect on single-chip modes or normal expanded narrow modes.

0 = PE3 is a general-purpose I/O pin.

1 = PE3 is configured as the LSTRB bus-control output, provided that the HCS12 is not in single-chip or normal expanded narrow modes.

RDWE: *Read/Write enable*. Can be read any time.

In normal modes, write once. Special modes, write anytime. This bit has no effect in single-chip modes.

0 = PE2 is a general-purpose I/O pin.

1 = PE2 is configured as the R/W pin. In single-chip modes, RDWE has no effect and PE2 is a general-purpose I/O pin.

R/W used for external writes. After reset in normal expanded mode, it is disabled. If needed, it should be enabled before any external writes.

MODE Register

This register establishes the operation mode and other miscellaneous functions (i.e., internal visibility and emulation of ports E and K). The pin assignment for MODE register is as follows:

Port E Registers *Cont'd …*

7	6	5	4	3	2	1	0
MODC	MODB	MODA	0	IVIS	0	EMK	EME

MODC, MODB, MODA: mode select bits.

000 = special single-chip mode

001 = emulation narrow mode

010 = special test mode

011 = emulation wide mode

100 = normal single-chip mode

101 = normal expanded narrow mode (external data bus memory is 8-bit)

110 = special peripheral mode

111 = normal expanded wide mode (external memory data bus is 16-bit)

IVIS: internal visibility.

0 = no visibility of internal bus operations on external bus.

1 = internal bus operations are visible on external bus.

EMK: emulate port K.

0 = PORTK and DDRK are in the memory map and port K can be used for general I/O.

1 = if in any expanded mode, PORTK and DDRK are removed from memory map.

Port E Registers *Cont'd …*

EME: emulate port E.

0 = PORTE and DDRE are in the memory map and port E can be used for general I/O

1 = in any expanded mode or special peripheral mode, PORTE and DDRE are removed from memory map, which allows the user to emulate the function of these registers externally.

Pull-Up Control Register (PUCR)

This register is used to select the pull-up resistors for the pins associated with the core part. The MC9S12DP256 has ports A, B, E, and K in its core part. This register can be written at any time. The pin assignment for PUCR register is as follows:

7	6	5	4	3	2	1	0
PUPKE	0	0	PUPEE	0	0	PUPBE	PUPAE

PUPKE: pull-up port K enable.

0 = pull-up resistors of port K are disabled.

1 = pull-up resistors of port K are enabled.

PUPEE: pull-up port E enable.

0 = pull-up resistors of port E are disabled.

1 = pull-up resistors of port E are enabled.

Port E Registers *Cont'd* …

PUPBE: pull-up port B enable.

0 = pull-up resistors of port B are disabled.

1 = pull-up resistors of port B are enabled.

PUPAE: pull-up port A enable.

0 = pull-up resistors of port A are disabled.

1 = pull-up resistors of port A are enabled.

Reduced Drive Register (RDRIV)

This register is used to select drive for the pins associated with the core ports, which gives reduced power consumption and reduced RFI with a slight increase in transition time, a feature used on ports that have a light loading. This register is not in the memory map in expanded mode. The pin assignment for RDRIV register is as follows:

7	6	5	4	3	2	1	0
RDPK	0	0	RDPE	0	0	RDPB	RDPA

Port E Registers *Cont'd …*

RDPK: reduced drive of port K.

0 = all port K pins have full drive enabled.

1 = all port K pins have reduced drive enabled.

RDPE: reduced drive of port E.

0 = all port E pins have full drive enabled.

1 = all port E pins have reduced drive enabled.

RDPB: reduced drive of port B.

0 = all port B pins have full drive enabled.

1 = all port B pins have reduced drive enabled.

RDPA: reduced drive of port A.

0 = all port A pins have full drive enabled.

1 = all port A pins have reduced drive enabled.

External Bus Interface Control Register (EBICTL)

Only the bit 0 (ESTR) of this register is implemented. It controls the stretching of the external E clock. When the ESTR bit is set to 0, the E clock is free-running and does not stretch (lengthen). When the HCS12 is interfacing with a slower memory drive, then the E clock can be lengthened (stretch its high interval) by setting this bit.

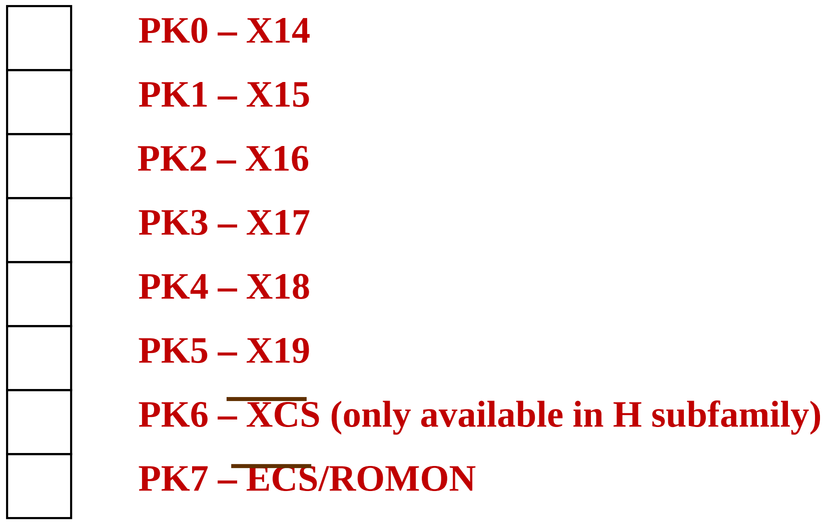
Port K:

➢ Port K has a data register (PORTK) and a data direction register (DDRK).

➢ In expanded mode, port K carries the expanded address XADDR14~XADDR19, emulated chip select (ECS), and external chip select (XCS) signals.

➢ At the rising edge of the RESET signal, the state of the PK7 pin is latched to the ROMON bit of the MISC register. If this bit is 1, the on-chip flash memory will be enabled in the extended mode. This bit is forced to 1 in the single-chip modes.

Port K pins and their alternate functions



PK0 – X14
PK1 – X15
PK2 – X16
PK3 – X17
PK4 – X18
PK5 – X19
PK6 – $\overline{\text{XCS}}$ (only available in H subfamily)
PK7 – $\overline{\text{ECS}}$/ROMON

Port T:

- ➢ Port T has a port I/O register (PTT), port data direction register (DDRT), port input register (PTIT), reduced drive register (RDRT), pull device enable register (PERT), and port polarity select register (PPST).
- ➢ The PTIT register allows one to read back the status of port T pins. This register can also be used to detect overload or short circuit conditions on output pins.

The **RDRT** register configures the drive strength of each port output pin as either full or reduced current. If a port pin is used as input, this bit is ignored. The reduced drive feature for each pin can be enabled individually.

RDRT7	RDRT6	RDRT5	RDRT4	RDRT3	RDRT2	RDRT1	RDRT0
0	0	0	0	0	0	0	0

RDRT[7:0]: reduced drive port T.

0 = full drive strength at output.

1 = associated pin devices at about 1/3 of the full drive strength.

Port T *Cont'd …*

The **PERT** register configures whether a pull-up or pull-down device is enabled if the port is used as input. Each pin's pull-up or pull-down device can be enabled individually. No pull-up or pull-down device is enabled out of reset.

PERT7	PERT6	PERT5	PERT4	PERT3	PERT2	PERT1	PERT0
0	0	0	0	0	0	0	0

PERT[7:0]: pull device enable port T.

0 = pull-up or pull-down is disabled.

1 = either pull-up or pull-down is enabled.

The **PPST** register selects whether a pull-down or pull-up device is connected to the pin. This register has an effect on input pins only.

PPST7	PPST6	PPST5	PPST4	PPST3	PPST2	PPST1	PPST0
0	0	0	0	0	0	0	0

Port T *Cont'd …*

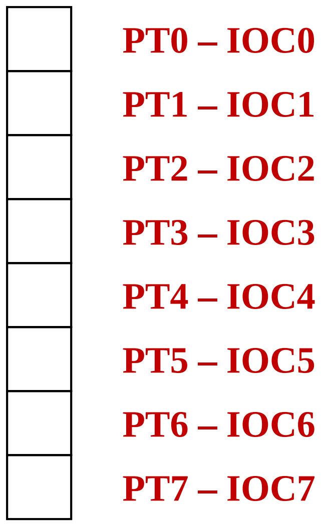
PPST[7:0]: pull polarity select port T.

0 = a pull-up device is connected to the associated port T pin, if enabled by the associated bit in register PERT and if the port is used as input or as wired-or output.

1 = a pull-down device is connected to the associated port T pin, if enabled by the associated bit in register PERT and if the port is used as input.

The **pin function of port T are shown below. In addition to being used as general I/O pins, port T pins can also be used as *input capture* or *output compare* action pins.**

Port T pins and their alternate functions



Port S:

➢ Port S has a port S Wired-OR mode register (WOMS) in addition to all the registers associated with port T (PTS, DDRS, PTIS, RDRS, PERS, and PPSS).

➢ Each bit of WOMS register configures the associated output pin as wired-OR. The contents of this register is shown below.

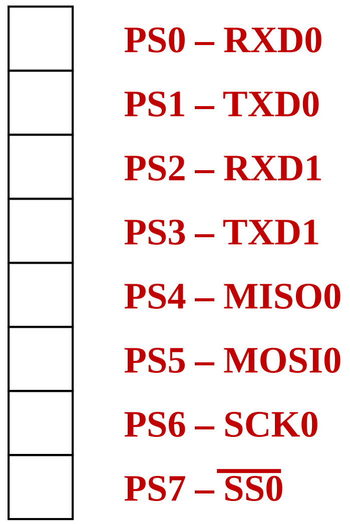
WOMS7	WOMS6	WOMS5	WOMS4	WOMS3	WOMS2	WOMS1	WOMS0
0	0	0	0	0	0	0	0

WOMS[7:0]: wired-OR mode port S.

0 = output buffers operate as push-pull output.

1 = output buffers operate as open-drain output.

The contents of PTIS, RDRS, PERS, and PPSS are identical to those of port T. **Port S** pins can be used as general I/O pins or serial interface signals.



Ports H, J, and P:

- These three I/O ports have the same set of registers associated with them.
- All of the pins associated with these three ports have *edge-triggered* interrupt capability in the wired-OR fashion.
- The SPI function pins can be routed to ports H and P.
- The routing of SPI functions done by programming the MODRR register.
- Each of these three ports has eight associated registers:
    1. Port I/O register (PTH, PTJ, PTP)
    2. Port input register (PTIH, PTIJ, PTIP)
    3. Port data direction register (DDRH, DDRJ, DDRP)
    4. Port reduced drive register (RDRH, RDRJ, RDRP)
    5. Port pull device enable register (PERH, PERJ, PERP)
    6. Port polarity select register (PPSH, PPSJ, PPSP)
    7. Port interrupt enable register (PIEH, PIEJ, PIEP)
    8. Port interrupt flag register (PIFH, PIFJ, PIFP)

Ports H, J, & P *Cont'd …*

All except the last two registers have been discussed. The contents of the port H Interrupt Enable register and port H Interrupt Flag register are shown below:

PIEH7	PIEH6	PIEH5	PIEH4	PIEH3	PIEH2	PIEH1	PIEH0
0	0	0	0	0	0	0	0

PIEH[7:0]: interrupt enable port H.

0 = interrupt is disabled.

1 = interrupt is enabled.

PIFH7	PIFH6	PIFH5	PIFH4	PIFH3	PIFH2	PIFH1	PIFH0
0	0	0	0	0	0	0	0

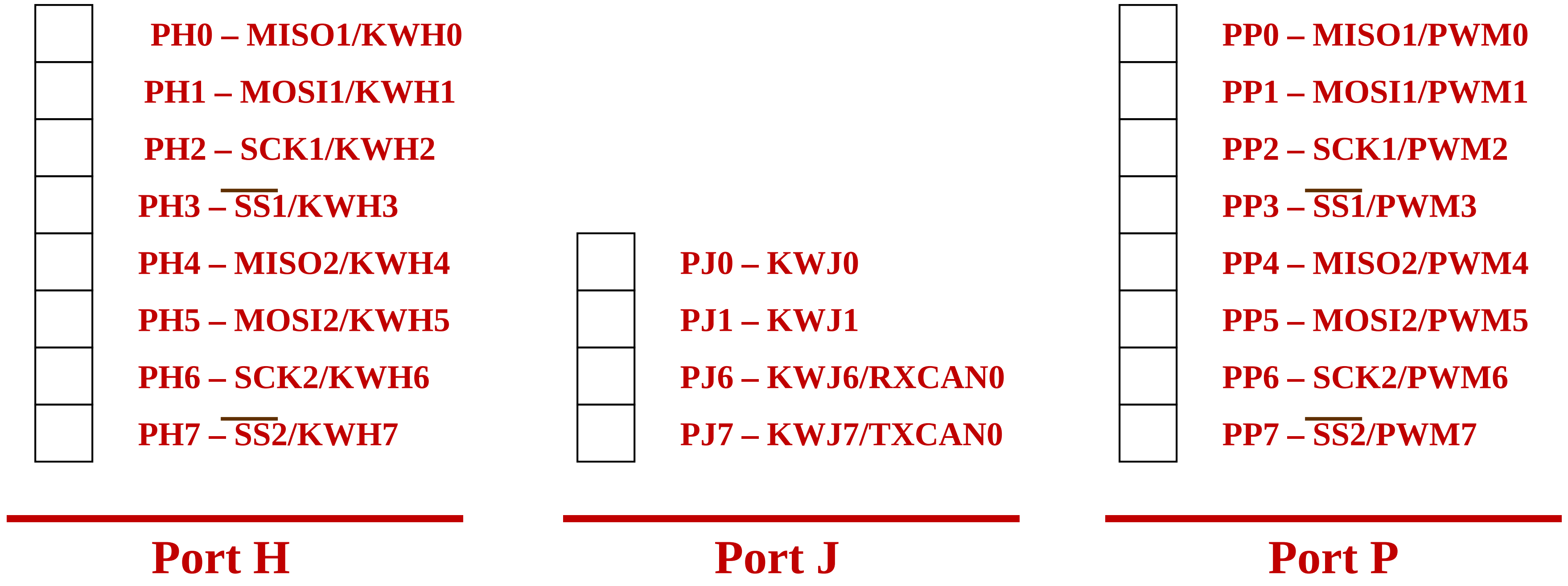
PIFH[7:0]: interrupt flag port H.

0 = no active edge pending.

1 = active edge has occurred (writing a '1' clears the associated flag).

Ports H, J, & P *Cont'd …*

- The interrupt enable as well as the sensitivity to rising or falling edges can be individually configured on a per pin basis.
- If a pin's pull-down device is enabled, then the interrupt is rising-edge triggered. Otherwise, it is falling-edge triggered.
- All 8-bits/pins of the port share the ***same interrupt vector***. Interrupts can be used with the pin configured as inputs or outputs.



Ports AD0 and AD1:

- HCS12 have implemented two 8-channel A/D converters.
- These two ports are analog input interface to the analog-to-digital subsystem.
- When analog-to-digital functions are not enabled, these two ports are available for general-purpose I/O.
- Since these ports can not be used as output, there are no data direction registers associated with them.
- The ports have no resistive input loads and no reduced drive controls.
- Corresponding data registers for these two ports are called PTAD0 and PTAD1.
- To be able to use these ports as digital input, each port has a ATD digital input enable register (ATD0DIEN and ATD1DIEN).
- In order to use an A/D pin as a digital input, its associated bit in this register needs to be set to 1.

IEN7	IEN6	IEN5	IEN4	IEN3	IEN2	IEN1	IEN0
0	0	0	0	0	0	0	0

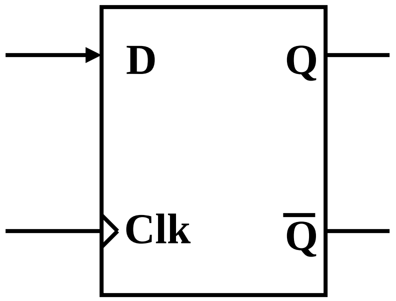
IENx[7:0]: ATD digital input enable on channel x.

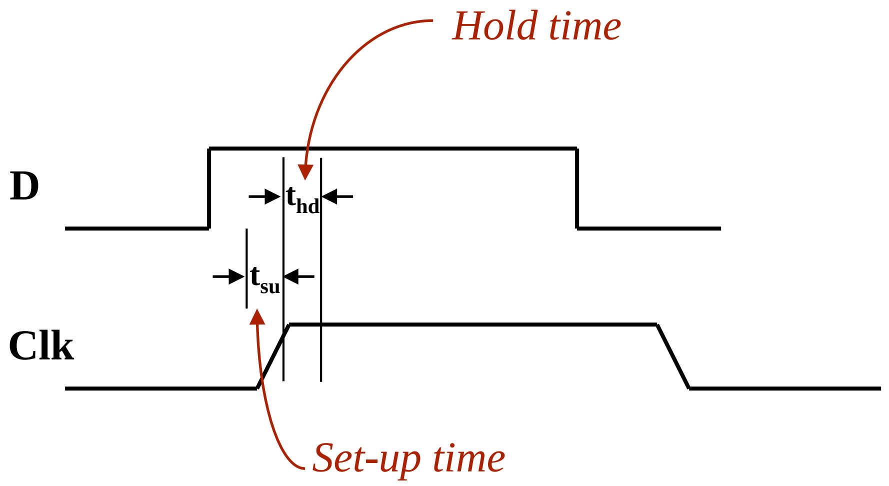
0 = disable digital input buffer to PTADx pin.

1 = enable digital input buffer to PTADx pin.

Electrical & Timing Considerations for I/O Interfacing:

- Voltage-level compatibility.
- Current Drive compatibility.
- Timing compatibility.



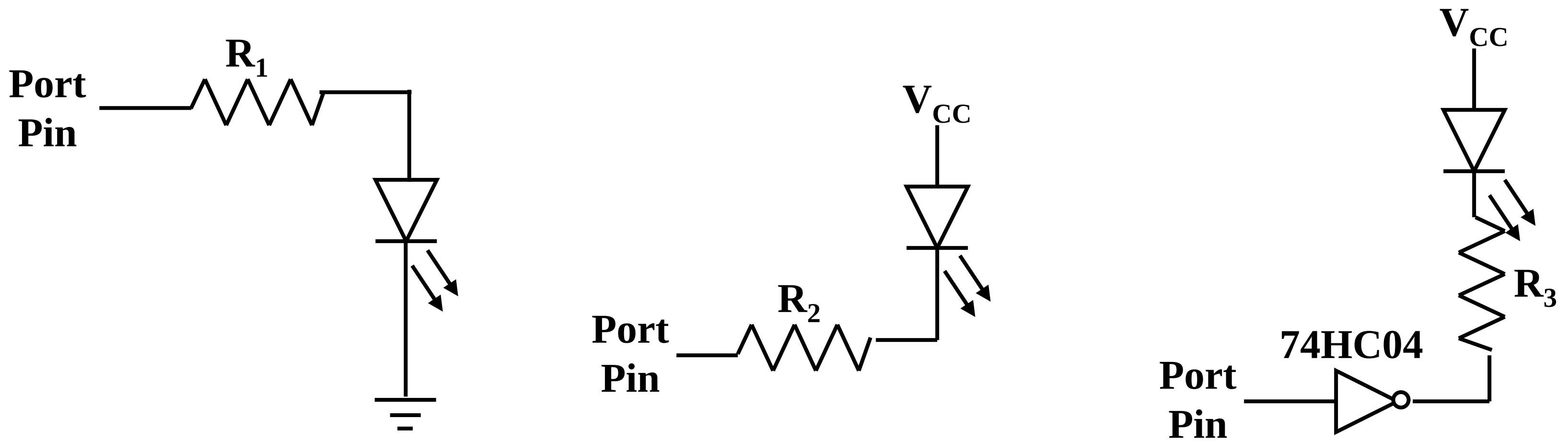


Interfacing with Simple I/O Devices

Interfacing with LEDs:

- The LEDs are often used to indicate the system operation mode, whether power is turned on, whether system operation is normal, and so forth.
- An LED can illuminate when it is forward biased and has sufficient current flowing through it.
- The current required to light a LED may range from a few to more than 10 mA.
- The voltage drop across the LED can range from 1.6 V to more than 2.2 V.
- There are three different methods for interfacing with LEDs.
- The first two methods are only recommended for use with LEDs that need only 1 to 2 mA to produce enough brightness.

Interfacing with Simple I/O Devices *cont'd …*



1. Positive Direct Drive **2. Inverse Direct Drive** **3. Buffered Drive**

An LED connected to a CMOS inverter through a current

- The circuit 3 is recommended for use with LEDs that need larger current to light.
- Dragon12 has provided 8 LEDs which are connected to Port B.
- Bit 1 of port J is used to enable and disable these connections. If it is set to 0 will enable this function.

```
*************************************************************************************
*   Function: Makes port B as a binary counter and displays it on LEDs.   *
*************************************************************************************
#include  reg9s12.h
          org       $1000              ; program code
start     ldaa      #$FF
          staa      ddrb               ; make port B an output port
          staa      ddrp               ; make port P an output port
          staa      ptp                ; turn off 7-segment LED display
          bset      ddrj,$02           ; make bit 1 of port j output
          bclr      ptj,$02            ; clear bit 1 of port j to enable LEDs
back      inca                         ; create a binary counter with 100 ms
          staa      portb              ; delay between counts
          jsr       d_100ms             ;
          jmp       back                ;

d_100ms   ldab      #100               ; delay 100 ms
dly1      ldy       #6000              ; 6000 x 4 = 24,000 cycles = 1ms
dly       dey                          ; 1 cycle
          bne       dly                ; 3 cycles
          dbne    b,dly1                         ; continue till it is 100 ms
          rts

          end
```

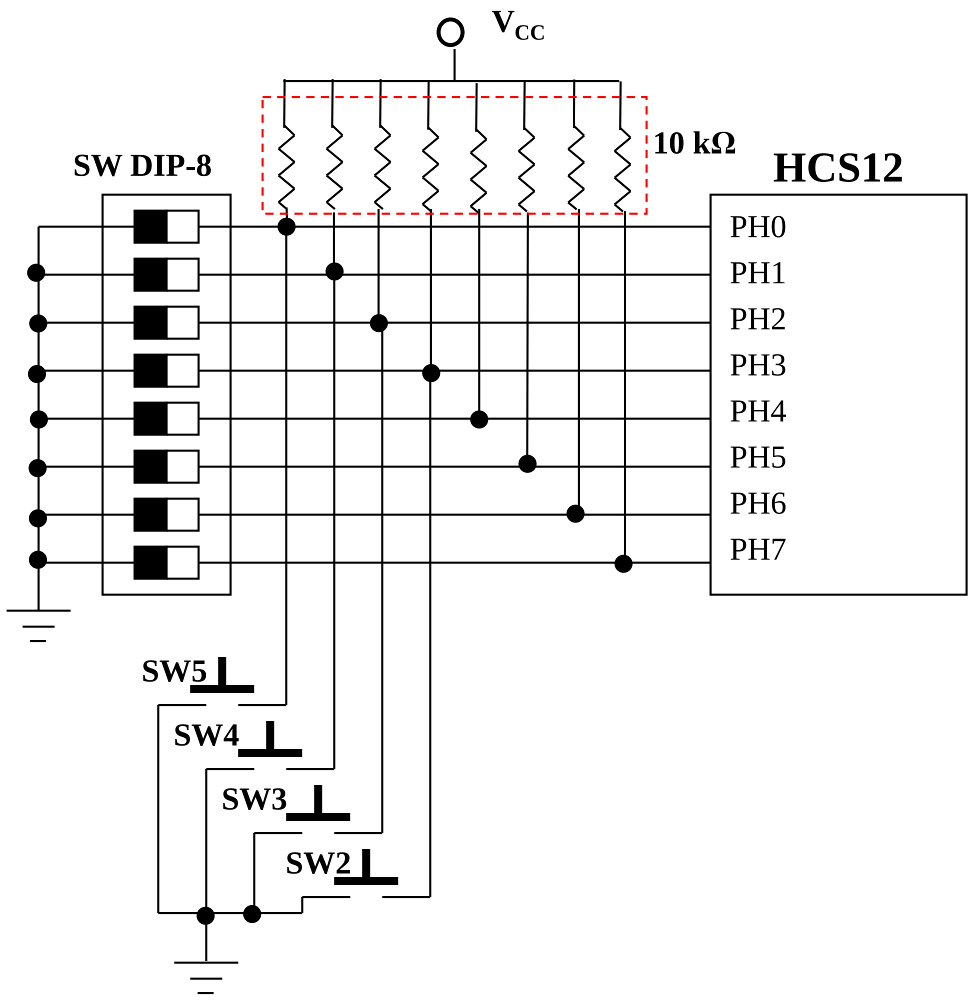
```
*********************************************************************************
*      Function: Turns LEDs ON and OFF one at a time every 100 ms.      *
*********************************************************************************
#include            reg9s12.h

            org         $1000               ; program code
start       ldaa        #$ff
            staa        ddrb                ; make port B an output port
            staa        ddrp                ; make port P an output port
            staa        ptp                 ; turn off 7-segment LED display
            bset        ddrj,$02            ; make bit 1 of port j output
            bclr        ptj,$02             ; clear bit 1 of port j to enable LEDs
            movb        #1,portb            ; start with most right LED
repeat      ldaa        #7                  ; shift left 7-times with100 ms
bb          jsr         d_100ms              ; delay between moves
            lsl         portb                 ;
            dbne        a,bb                   ;
            ldaa        #7                  ; shift right 7-times with100 ms
bb1         jsr         d_100ms               ; delay between moves
            lsr         portb                  ;
            dbne        a, bb1                  ;
            jmp         repeat              ; repeat process
```

```
*********************************************************************************
*       Function: Turns LEDs ON and OFF one at a time every 100 ms.        *
*********************************************************************************
#include                reg9s12.h

            org         $1000                   ; program code
start       ldaa        #$ff
            staa        ddrb                    ; make port B an output port
            staa        ddrp                    ; make port P an output port
            staa        ptp                     ; turn off 7-segment LED display
            bset        ddrj,$02                ; make bit 1 of port j output
            bclr        ptj,$02                 ; clear bit 1 of port j to enable LEDs

repeat      ldy         #led_tbl                ; create a moving light with 100 ms
back        movb        1,y+,portb              ; delay between moves
            pshy                                 ;
            jsr         d_100ms                   ;
            puly                                   ;
            cpy         #led_tbl+14                 ;
            bne         back                         ;
            bra         repeat                        ;
*
```

```
d_100ms ldab     #100                ; delay 100 ms
dly1    ldy      #6000               ; 6000 x 4 = 24,000 cycles = 1ms
dly     dey                          ; 1 cycle
        bne      dly                 ; 3 cycles
        dbne     b,dly1              ; continue till it is 100 ms
        rts

led_tbl fcb      $80,$40,$20,$10,$08,$04,$02,$01
        fcb      $02,$04,$08,$10,$20,$40
        end
```

Interfacing with DIP Switches:

- A switch is probably the simplest input device we can find.
- To make input more efficient, a set of eight switches are organized as a *dual-in-package* (DIP) is often used.
- A DIP package can be connected to any input port with eight pins.
- Dragon12 board uses port H to connect the DIP switches.
- When the switch is closed the associated port H input is 0.



Schematic of Push Buttons and DIP Switches on Dragon12 Board

Example: Write a program that displays a binary counter on the 8-LEDs. Program should control the speed of the counter through DIP switches and terminate any time push button SW5 is pressed.

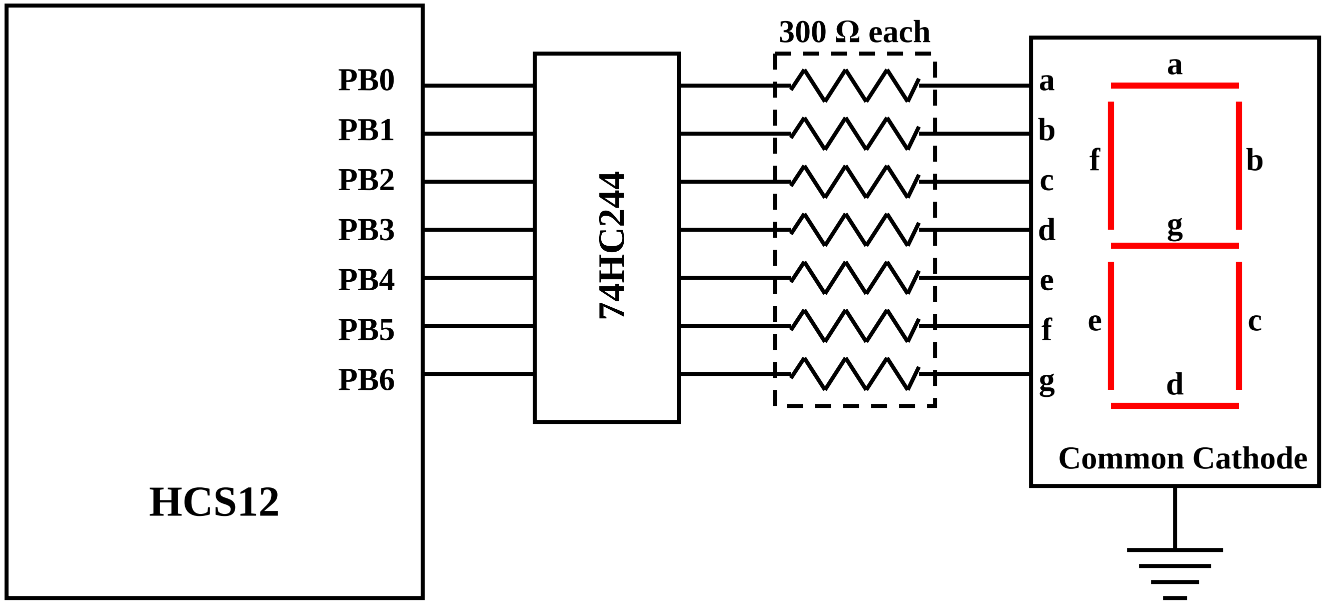
```
*****************************************************************************************************
*Program creates binary counter and displays it on LEDs.  The speed of the counter is controlled*
*        by DIP switches. The program will terminates as soon as push button SW5 is pressed.        *
*****************************************************************************************************

#include    reg9s12.h
            org         $1000                    ; program code

start       ldaa        #$ff
            staa        ddrb                     ; make port B an output port
            staa        ddrp                     ; make port P an output port
            staa        ptp                      ; turn off 7-segment LED display
            bset        ddrj,$02                 ; make bit 1 of port j output
            bclr        ptj,$02                  ; clear bit 1 of port j to enable LEDs
            clr         ddrh                     ; make port H an input port

back        inca                                 ; create a binary counter with as many ms
            staa        portb                     ; delay between counts as DIP switches are
            jsr         d_100ms                    ; indicating. Stop when push button
            brset       pth,$01,back                ; SW5 is pressed.
            swi
```

```
d_100ms  ldab    pth                 ; read DIP switches for no. of ms delay

dly1     ldy     #6000               ; 6000 x 4 = 24,000 cycles = 1ms
dly      dey                         ; 1 cycle
         bne     dly                 ; 3 cycles
         dbne    b,dly1              ; continue till it is 100 ms
         rts

         end
```

Interfacing with Seven-Segment Displays:

- Seven-segment displays are often used when the embedded product needs to display only a few digits.
- Seven-segment displays are mainly used to display digits and small subset of letters.
- Although a HCS12 device has enough current to drive a seven-segment display, it is not advisable to do so when the controller needs to drive many other devices.

- Port B derives a common-cathode seven-segment display through the buffer chip 74HC244.

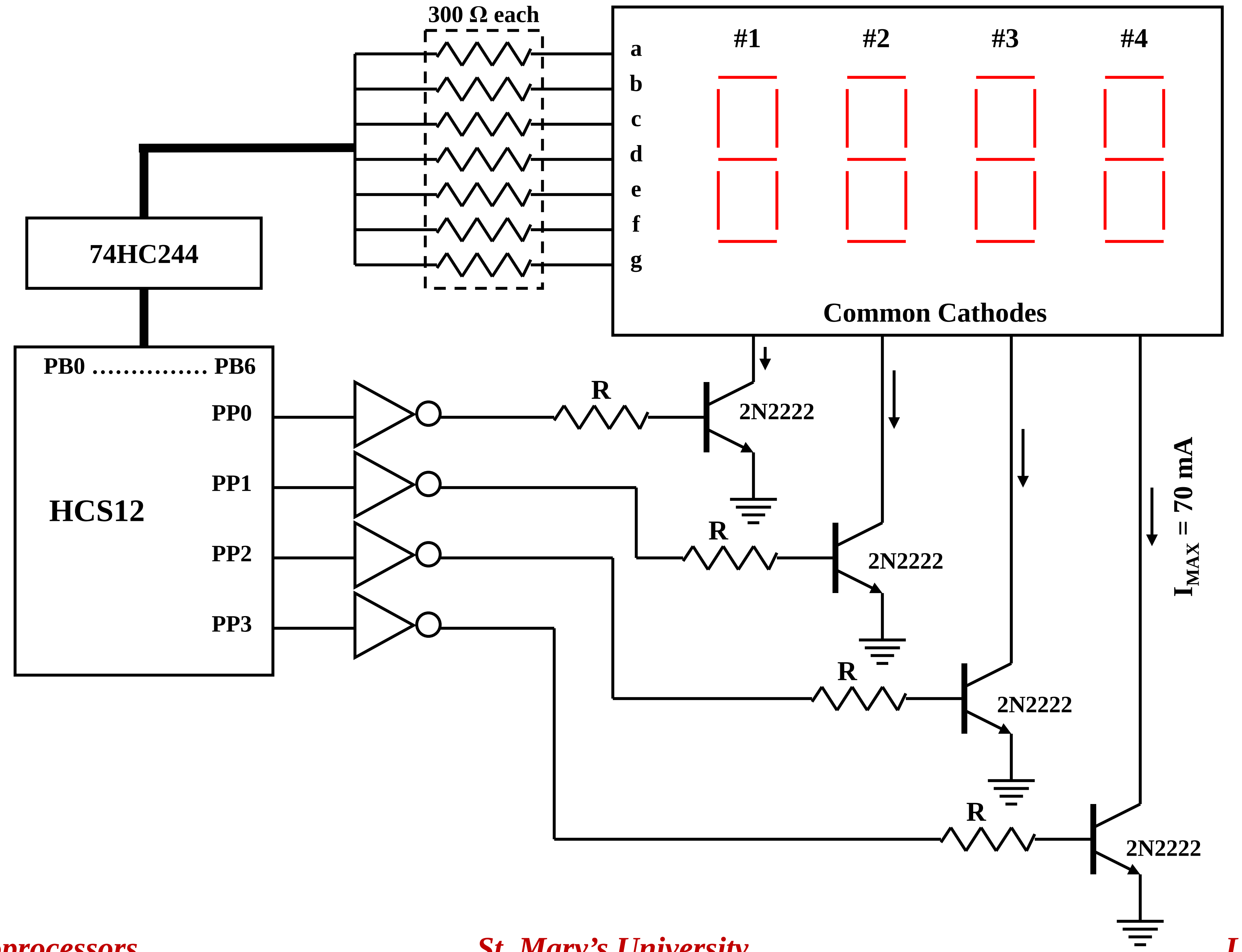


- The light patterns corresponding to ten BCD (decimal) digits are shown in the following table.
- The numbers in table require that segments a~g be connected from PB0~PB6 of the output port B.

BCD Digit	g Pb6	f pb5	e pb4	d pb3	c pb2	b pb1	a pb0	Corresponding Hex Number
0	0	1	1	1	1	1	1	$3F
1	0	0	0	0	1	1	0	$06
2	1	0	1	1	0	1	1	$5B
3	1	0	0	1	1	1	1	$4F
4	1	1	0	0	1	1	0	$66
5	1	1	0	1	1	0	1	$6D
6	1	1	1	1	1	0	1	$7D
7	0	0	0	0	1	1	1	$07
8	1	1	1	1	1	1	1	$7F
9	1	1	0	1	1	1	1	$6F

- When an application needs to display multiple BCD digits, the *time-multiplexing technique* is used.
- An example of the circuit that displays 4 BCD digits is shown in next slide.
- The common-cathode of the display is connected to the collector of an NPN transistor.

Interfacing with Simple I/O Devices *cont'd …*



- When a port P pin voltage is low, the connected NPN transistor will be driven into saturation region.
- The common cathode of the display then will be pulled-down to about 0.1 volt, allowing the display to be lighted.
- By turning ON and OFF these four transistors multiple time per second, multiple digits can be displayed.
- When all segments of a display is ON they draw about 70 mA of current that 2N2222 transistor is capable of sinking that current.
- To ensure the saturation mode for the transistor, a value of several hundred to 1 kΩ resistor is needed for 'R'.
- The following two programs will demonstrate the use of the seven-segment module on Dragon12.

Interfacing with Simple I/O Devices *cont'd …*

```
*******************************************************************************
*                     Set third 7-segment LED to value 4                      *
*******************************************************************************
#include  reg9s12.h                           ; include register equates
          org       $1000
          ldaa      #$ff       ; turn off 7-segment display
          staa      ddrb       ; portb = output
          staa      ddrp       ; portp = output
          staa      ptp        ; portp = 11111111 - all 7-segments are off

          ldaa      #$66         ; seven segment code for digit '4'
          staa      portb
          bset      ptp,$01    ; turn off digit 1
          bset      ptp,$02    ; turn off digit 2
          bclr      ptp,$04                   ; turn on digit 3
          bset      ptp,$08    ; turn off digit 4
          swi

***    segment pattern     ***
zero      fcb       $3f,$06,$5b,$4f,$66,$6d,$7d,$07              ; 0-7
*                     0,   1,   2,  3,   4,   5,   6,   7
          fcb       $7f,$6f,$77,$7c,$39,$5e,$79,$71              ; 8-$0f
*                     8,  9,   A,  b,   C,   d,  E,   F
          end
```

```
**************************************************************************************
*                        Set 7-segment module to display '6125'                      *
**************************************************************************************

#include  reg9s12.h                                     ; include register equates

          org      $1000
          ldaa     #$ff                  ; [A] = 11111111
          staa     ddrb                  ; portb = output
          staa     ddrp                  ; portp = output
          staa     ptp                   ; turn off all 7-segment displays

repeat    ldaa     #$7d                  ; seven segment for digit '6'
          staa     portb                 ; send it to 7-segment
          bclr     ptp,$01               ; turn on digit 1
          jsr      d_1ms                 ; keep it ON for 1 ms
          bset     ptp,$01               ; turn off digit 1
          ldaa     #$06                  ; seven segment for digit '1'
          staa     portb                 ; send it to 7-segment
          bclr     ptp,$02               ; turn on digit 2
          jsr      d_1ms                 ; keep it ON for 1 ms
          bset     ptp,$02               ; turn off digit 2
          ldaa     #$5b                  ; seven segment for digit '2'
```

Interfacing with Simple I/O Devices *cont'd …*

```
        staa    portb           ; send it to 7-segment
        bclr    ptp,$04         ; turn on digit 3
        jsr     d_1ms           ; keep it ON for 1 ms
        bset    ptp,$04         ; turn off digit 3
        ldaa    #$6d            ; seven segment for digit '5'
        staa    portb           ; send it to 7-segment
        bclr    ptp,$08         ; turn on digit 4
        jsr     d_1ms           ; keep it ON for 1 ms
        bset    ptp,$08         ; turn off digit 4
        jmp     repeat

d_1ms   ldy     #6000           ; 6000 x 4 = 24,000 cycles = 1ms
dly     dey                     ; 1 cycle
        bne     dly             ; 3 cycles
        rts

* segment pattern
zero    fcb     $3f,$06,$5b,$4f,$66,$6d,$7d,$07         ; 0-7
*                 0,  1,  2,  3,  4,  5,  6,  7
        fcb     $7f,$6f,$77,$7c,$39,$5e,$79,$71         ; 8-$0f
*                 8,  9,  A,  b,  C,  d,  E,  F
        end
```

Interfacing with LCD *cont'd …*

Write an assembly program to set the 7-segment module to count 0 to F in HEX while scrolling to right and wrapping around. The program should terminate once push button SW5 is pressed.

```
*****************************************************************************
* Set 7-seg module to Count 0 to F in Hex while scrolling to right and *
*wrapping around it will terminate once push button SW5 is pressed. *
*****************************************************************************
#include      reg9s12.h            ; include register equates

            org      $1000
            ldaa     #$ff          ; turn off 7-segment display
            staa     ptp           ; portp = 11111111
            bset     ddrj,$02      ; bit 2 of port j is output
            bset     ptj,$02       ; turn off LEDs
            staa     ddrb          ; portb = output
            staa     ddrp          ; portp = output
repeat      ldy      #zero         ; start at digit 0 - 3
```

```
new     movb    0,y,dig1        ; load initial values into
        movb    1,y,dig2         ; four 7-segments
        movb    2,y,dig3          ;
        movb    3,y,dig4           ;

        ldab    #25             ; repeat 25 times
redo    ldaa    dig1            ; get unit digit
        staa    portb                       ; send it to 7-segment
        bclr    ptp,$01         ; turn on digit 4
        jsr     d_1ms           ; delay 1 ms
        bset    ptp,$01           ; turn off digit 4
        ldaa    dig2            ; get 10 digit
        staa    portb                       ; send it to 7-segment
        bclr    ptp,$02         ; turn on digit 3
        jsr     d_1ms           ; delay 1 ms
        bset    ptp,$02           ; turn off digit 3
        ldaa    dig3            ; get 100 digit
        staa    portb               ; send it to 7-segment
        bclr    ptp,$04               ; turn on digit 2
        jsr     d_1ms                   ; delay 1 ms
        bset    ptp,$04               ; turn off digit 2
        ldaa    dig4                ; get 1000 digit
        staa    portb                               ; send it to 7-segment
        bclr    ptp,$08               ; turn on digit 1
        jsr     d_1ms                               ; delay 1 ms
        bset    ptp,$08               ; turn off digit 1
        dbne    b,redo

        iny                         ; scroll down by one
        cpy     #zero+16            ; is pointer out of range?
        bne     new                 ; if no, then continue
        jmp     repeat              ; if yes, then re-initialize
done    swi
```

Interfacing with LCD *cont'd …*

```
d_1ms  brclr   pth,$01,done
       ldx     #6000                ; 6000 x 4 = 24,000 cycles = 1ms
dly    dex                          ; 1 cycle
       bne     dly                  ; 3 cycles
       rts

***  segment pattern   ***
zero   fcb    $3f,$06,$5b,$4f,$66,$6d,$7d,$07            ; 0-7
*              0,  1,   2,   3,  4,   5,   6,   7
       fcb    $7f,$6f,$77,$7c,$39,$5e,$79,$71            ; 8-$0f
*              8,  9,  A,   b,  C,   d,  E,   F
       fcb    $3f,$06,$5b
*              0,   1,   2
dig1   rmb     1
dig2   rmb     1
dig3   rmb     1
dig4   rmb     1

       end
```

Write an assembly program to set the 7-segment module to count up to '9999'.

```
*******************************************************************************
*                    Set 7-segment module to Count up to '9999'              *
*******************************************************************************
#include      reg9s12.h    ; include register equates

              org       $1000
              ldaa      #$ff                  ; [A] = 11111111
              staa      ddrb                  ; portb = output
              staa      ddrp                  ; portp = output
              staa      ptp                   ; turn off 7-segment displays

              ldy    #zero
              sty    ad3
              sty    ad2
              sty    ad1
              movb   0,y,dig4
              movb   #0,dig3
              movb   #0,dig2
              movb   #0,dig1
```

Interfacing with Simple I/O Devices *cont'd …*

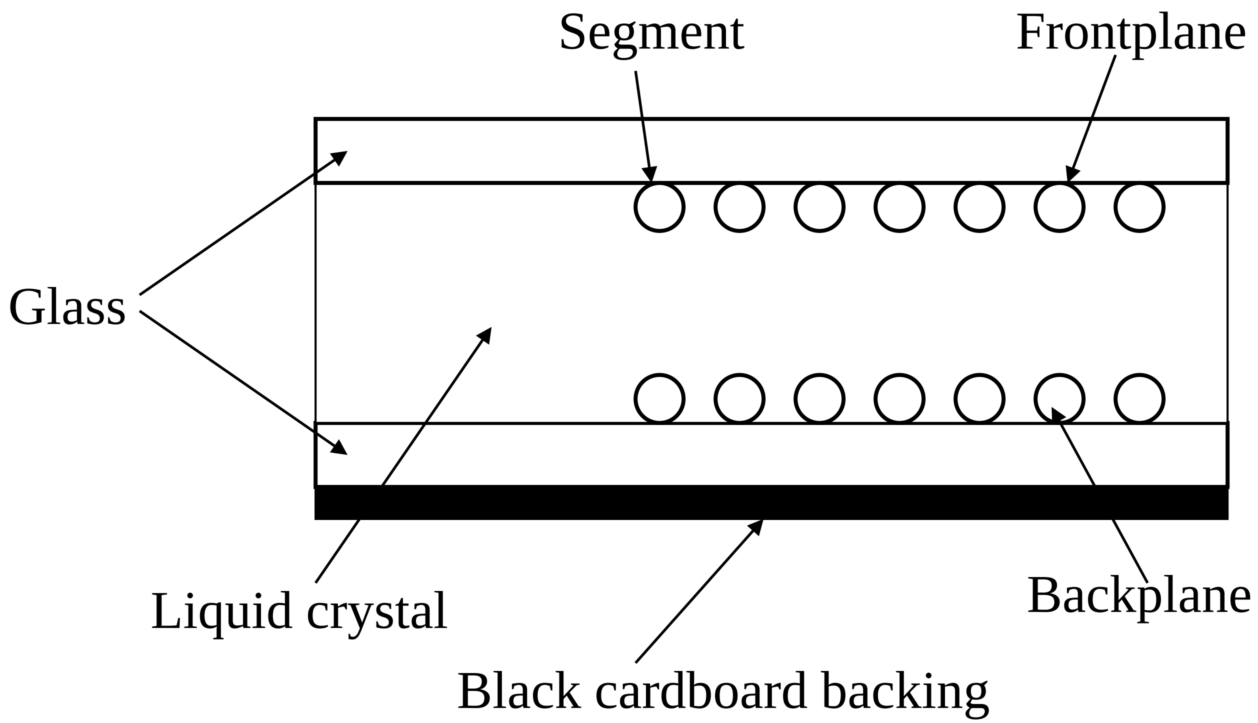
```
repeat  ldab   #10
redo    ldaa   dig4          ; get unit digit
        staa   portb         ; send it to 7-segment
        bclr   ptp,$08       ; turn on digit 4
        jsr    d_1ms          ; delay 1 ms
        bset   ptp,$08       ; turn off digit 4
        ldaa   dig3          ; get 10th digit
        staa   portb         ; send it to 7-segment
        bclr   ptp,$04       ; turn on digit 3
        jsr    d_1ms          ; delay 1 ms
        bset   ptp,$04       ; turn off digit 3
        ldaa   dig2          ; get 100th digit
        staa   portb         ; send it to 7-segment
        bclr   ptp,$02       ; turn on digit 2
        jsr    d_1ms          ; delay 1 ms
        bset   ptp,$02       ; turn off digit 2
        ldaa   dig1          ; get 1000th digit
        staa   portb         ; send it to 7-segment
        bclr   ptp,$01       ; turn on digit 1
        jsr    d_1ms          ; delay 1 ms
        bset   ptp,$01       ; turn off digit 1
        dbne   b,redo
```

```
            iny                     ; increment unit digit
            cpy    #zero+10         ; as long as not 10, continue refreshing
            bne    cont4             ;
            ldy    ad3              ; get 10th digit
            iny                     ; increment 10th digit
            cpy    #zero+10         ; as long as not 10, re-initialize pointer
            bne    cont3             ; for unit digit and continue refreshing
            ldy    ad2              ; get 100th digit
            iny                     ; increment 100th digit
            cpy    #zero+10         ; as long as not 10, re-initialize pointer
            bne    cont2             ; for unit & 10th digits, continue refreshing
            ldy    ad1              ; get 1000th digit
            iny                     ; increment 1000th digit
            cpy    #zero+10         ; as long as not 10, re-initialize pointer for
            bne    cont1             ; unit, 10th, & 100th digits, continue refreshing
            swi
cont1       movb   0,y,dig1         ; update 1000th digit
            sty    ad1              ; save 1000th digit pointer
            ldy    #zero            ; update digit 100th to zero
cont2       movb   0,y,dig2          ;
            sty    ad2              ; save 100th digit pointer
            ldy    #zero            ; update digit 10th to zero
cont3       movb   0,y,dig3          ;
            sty    ad3              ; save 10th digit pointer
            ldy    #zero            ; update unit digit to zero
cont4       movb   0,y,dig4          ;
            jmp    repeat           ; go back to refresh all 4 digits
```

```
d_1ms   ldx     #6000                           ; 6000 x 4 = 24,000 cycles = 1ms
dly     dex                                     ; 1 cycle
        bne     dly                             ; 3 cycles
        rts

* segment pattern
zero    fcb     $3f,$06,$5b,$4f,$66,$6d,$7d,$07         ; 0-7
*                0,  1,  2,  3,  4,  5,  6,  7
        fcb     $7f,$6f,$77,$7c,$39,$5e,$79,$71         ; 8-$0f
*                8,  9,  A,  b,  C,  d,  E,  F
dig1    rmb   1
dig2    rmb   1
dig3    rmb   1
dig4    rmb   1
ad3     rmb   2
ad2     rmb   2
ad1     rmb   2

        end
```

Interfacing with Liquid Crystal Displays (LCDs):

- Although seven-segment displays are easy to use, they are bulky and quite limited in the set of characters that they can display.
- Liquid Crystal Displays come in handy when the application requires the display of many characters.
- LCD has the following advantages:
  - High contrast
  - Low power consumption
  - Small footprint
  - Ability to display both characters and graphics
- The basic construction of an LCD is shown on next slide. The most common type of LCD allows light to pass through when activated.
- When a voltage is applied across the segment, an electrostatic field is set up that aligns the crystal in the liquid.

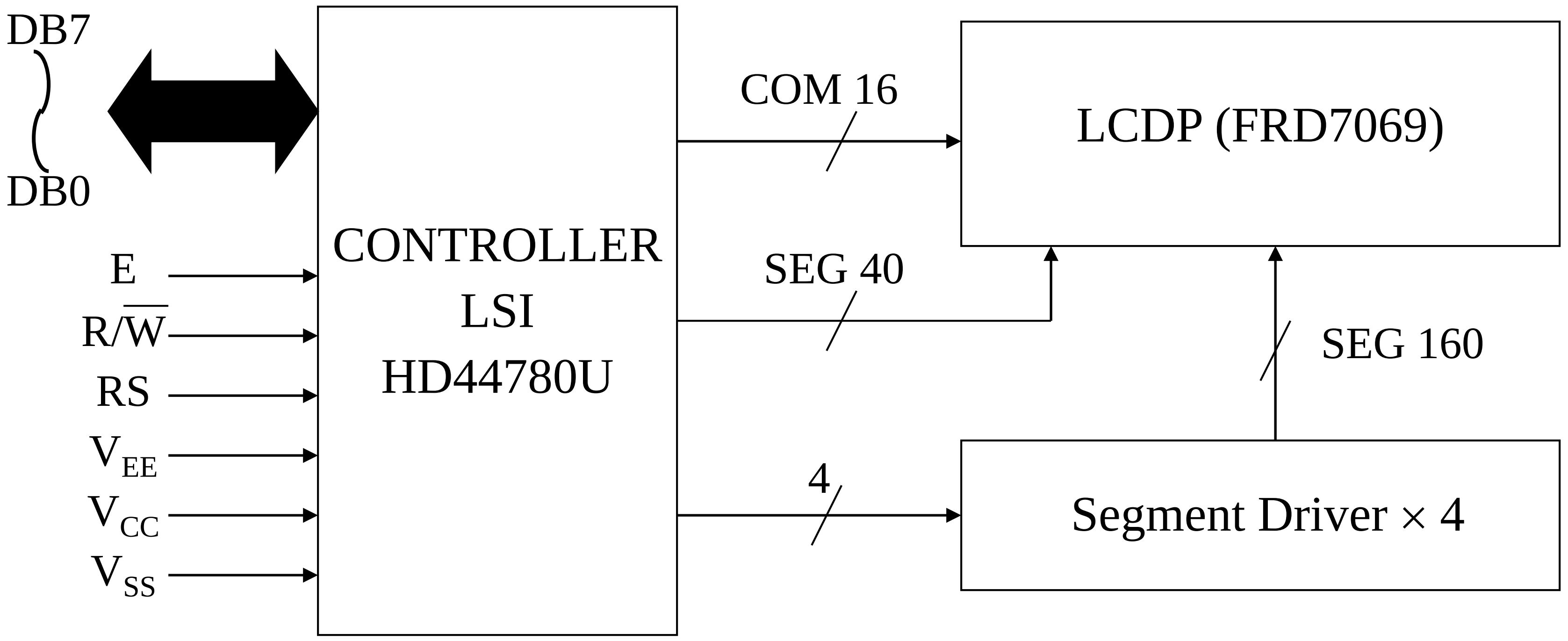


A Liquid Crystal Display (LCD)

- Although LCDs can display graphics and characters, only character-based LCD controller is used on Dragon12 board.
- The Hitachi HD44780, one of the most popular LCD display controllers, is used on Dragon12.

The HD44780U LCD Controller:

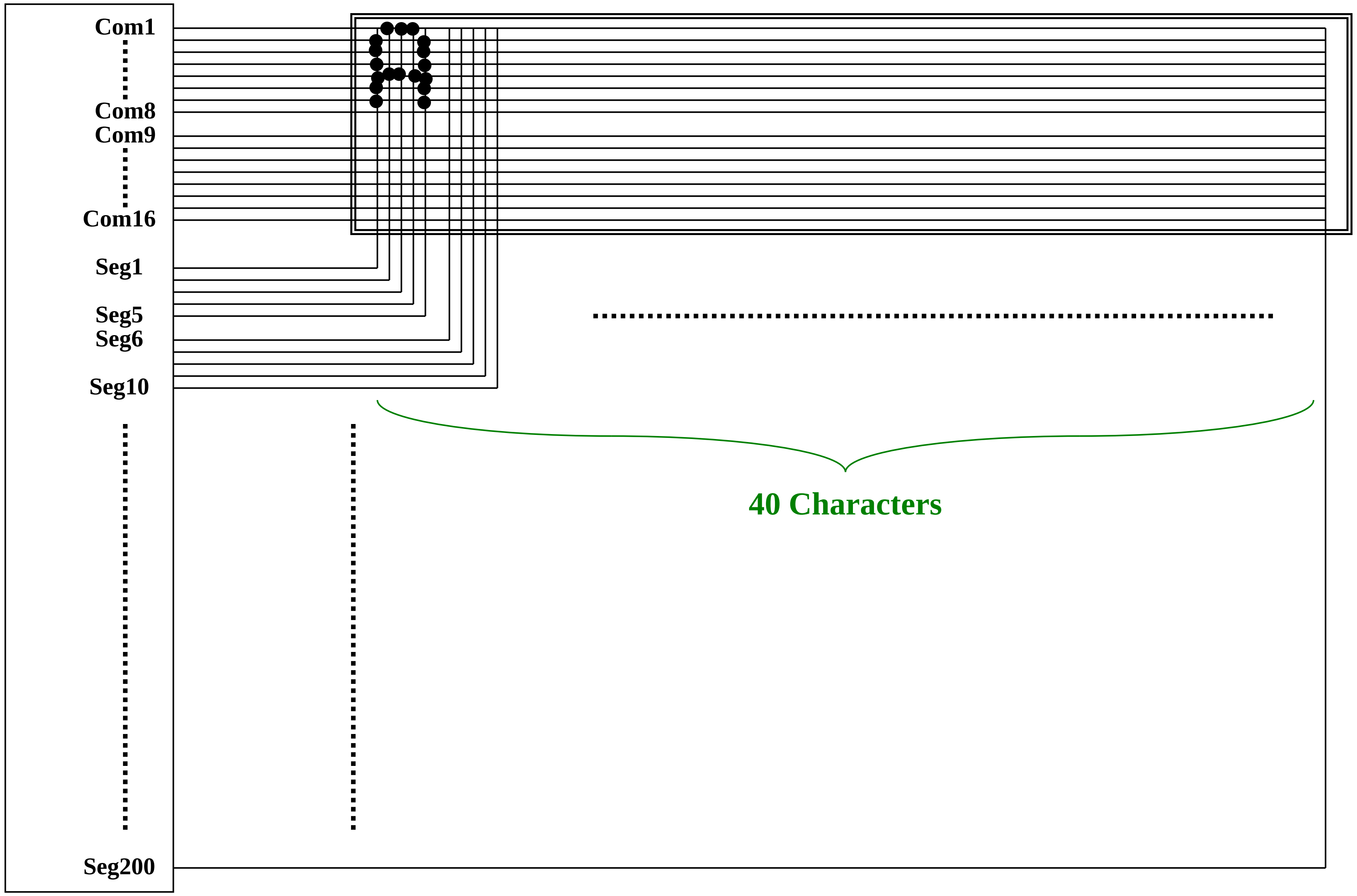
- The block diagram of an LCD kit that incorporates the HD44780U controller is shown.



Block diagram of a HD44780U-based LCD kit

Interfacing with LCD *cont'd …*

HD44780U



Pin No.	Symbol	I/O	Function
1	V_{SS}	--	Power supply (GND)
2	V_{CC}	--	Power supply (+5 V)
3	V_{EE}	--	Contrast adjust
4	RS	I	0 = instruction input, 1 = data input
5	R/W	I	0=write to LCD, 1=read from LCD
6	E	I	Enable signal
7	DB0	I/O	Data bus line 0
8	DB1	I/O	Data bus line 1
9	DB2	I/O	Data bus line 2
10	DB3	I/O	Data bus line 3
11	DB4	I/O	Data bus line 4
12	DB5	I/O	Data bus line 5
13	DB6	I/O	Data bus line 6
14	DB7	I/O	Data bus line 7

Pin assignment for displays with no more than 80 characters

Interfacing with LCD *cont'd …*

Pin No.	Symbol	I/O	Function
1	DB7	I/O	Data bus line 7
2	DB6	I/O	Data bus line 6
3	DB5	I/O	Data bus line 5
4	DB4	I/O	Data bus line 4
5	DB3	I/O	Data bus line 3
6	DB2	I/O	Data bus line 2
7	DB1	I/O	Data bus line 1
8	DB0	I/O	Data bus line 0
9	E1	I	Enable signal row 0 and 1
10	R/W	I	0=write to LCD, 1=read from LCD
11	RS	I	0 = instruction input, 1 = data input
12	V_{EE}	--	Contrast adjust (GND)
13	V_{SS}	--	Power supply (+5 V)
14	V_{CC}	--	Power supply
15	E2	I	Enable signal row 2 and 3
16	N.C.	--	

Pin assignment for displays with more than 80 characters

Interfacing with LCD *cont'd …*

- The HD44780U provides a set of instructions for user to setup the LCD parameters.
- The operations performed by these instructions are summarized on a table and will be presented in upcoming slides.
- The HD44780U can be configured to control 1-line, 2-line, and 4-line LCDs.
- The mappings of the character positions on the LCD screen and the Display Data RAM (DDRAM) addresses are not sequential and are shown in a table in upcoming slides.
- To better understand these tables, one needs to get familiar with memory allocations and different registers on board of the LCD controller.

Display Data RAM (DDRAM) – stores display data represented in 8-bit character codes. Its extended capacity is 80 × 8 bits, or 80 characters.

Character Generator ROM (CGROM) – The character generator ROM generates 5 × 8 or 5 × 10 dot character patterns from 8-bit character codes. It can generate 208 5 × 8 dot character patterns and 32 5 × 10 dot character patterns.

Character Generator RAM (CGRAM) – The user can rewrite character patterns into the CGRAM by programming. For 5 × 8 fonts, eight character patterns can be written, and for 5 × 10 fonts, four character patterns can be written.

Instruction Register (IR) – The IR is an 8-bit register which stores *instruction codes*, such as "display clear" and "cursor move", and *address information* for DDRAM and CGRAM.

Data Register (DR) – The DR is an 8-bit register that is used to transfer data from microcontroller into LCD controller. The IR and DR registers are distinguished by the RS signal.

Instruction	Code										Description	Execution time
	RS	R/W	B7	B6	B5	B4	B3	B2	B1	B0		
Clear display	0	0	0	0	0	0	0	0	0	1	Clears display and returns cursor to the home position (address 0).	1.64 ms
Cursor home	0	0	0	0	0	0	0	0	1	*	Returns cursor to home position. Also returns display being shifted to the original position. DDRAM content remains unchanged.	1.64 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction (I/D); specifies to shift the display (S). These operations are performed during data read/write.	40 μs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets on/off of all display (D), cursor on/off (C), and blink of cursor position character (B).	40 μs
Cursor/display shift	0	0	0	0	0	1	S/C	R/L	*	*	Sets cursor-move or display-shift (S/C), shift direction (R/L). DDRAM content remains unchanged.	40 μs
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display line (N), and character font (F).	40 μs
Set CGRAM address	0	0	0	1	CGRAM address						Sets the CGRAM address. CGRAM data are sent and received after this setting.	40 μs
Set DDRAM address	0	0	1	DDRAM address							Sets the DDRAM address. DDRAM data are sent and received after this setting.	40 μs
Read busy flag & address counter	0	1	BF	CGRAM/DDRAM address							Reads busy flag (BF) indicating internal operation being performed and reads CGRAM or DDRAM address counter contents (depending on previous instruction.	0 μs
Write CGRAM or DDRAM	1	0	write data								Write data to CGRAM or DDRAM	40 μs
Read from CG/DDRAM	1	1	read data								Read data from CGRAM or DDRAM	40 μs

Busy Flag (BF) – The HD44780U has a busy flag to indicate whether the current internal operation is complete. When BF is 1, the controller is still busy with an internal operation. When RS = 0 and R/W = 1, the busy flag is output to the DB7.

Address Counter (AC) – The HD44780U uses a 7-bit address counter to keep track of the address of the next DDRAM or CGRAM location to be accessed. When an instruction is written into the IR register, the address information contained in the instruction is transferred to the AC register. After writing or reading of DDRAM or CGRAM, the content of the AC register is automatically incremented by 1.

Instruction Description:

Clear Display – Clears the whole DDRAM and sets the address counter to 0. It reconfigures the LCD to its initial mode and sets I/D to 1 (increment mode).

Return Home – sets DDRAM address 0 into the address counter and returns to its original status if it was shifted. The DDRAM contents are not changed.

Entry Mode Set – The I/D bit of the instruction controls increment/decrement which in turn controls cursor going to right or left. The S bit controls the shifting of the LCD display. S=1→ display shifts, S=0 → display does not shift.

Display ON/OFF Control – This instruction has three bit parameters: D, C, and B. D=1→ display is turned ON, C=1 → cursor is turned on, B=1 → cursor blinks.

Cursor or Display Shift – This instruction shifts the cursor position to the right or left without writing or reading display data. Shifting is controlled by two bits as its shown.

S/C	R/L	Operation
0	0	Shifts the cursor position to the left. (AC is decremented by 1).
0	1	Shifts the cursor position to the right. (AC is incremented by 1).
1	0	Shifts the entire display to the left. The cursor follows the display shift.
1	1	Shifts the entire display to the right. The cursor follows the display shift.

Function Set – This instruction allows the user to set the interface data length, select the number of display lines, and select the character fonts. There are three bit variables in this instruction:

DL = 1 → data length 8-bit, DL = 0 → data length 4-bit (DB7 – DB4)

N = 0 → 1-line display is selected, N = 1 → 2-line display is selected

F = 0 → 5 × 8 font is selected, F =1 → the 5 × 10 font is selected.

Set CGRAM Address – This instruction contains the CGRAM address to be set into the address counter.

Set DDRAM Address – This instruction allows the user to set the address of the DDRAM (in address counter).

Read Busy Flag and Address – This instruction reads the busy flag (BF) and the address counter. The BF flag indicates whether the LCD controller is still executing the previously received instruction.

LCD Instruction bit names

Bit Name	Settings	
I/D	0=decrement cursor position	1=increment cursor position
S	0 = no display shift	1 = display shift
D	0 = display off	1 = display on
C	0 = cursor off	1 = cursor on
B	0 = cursor blink off	1 = cursor blink on
S/C	0 = move cursor	1 = shift display
R/L	0 = shift left	1 = shift right
DL	0 = 4-bit interface	1 = 8-bit interface
N	0 = 1/8 or 1/11 duty (1 line)	1 = 1/16 duty (2 lines)
F	0 = 5×8 dots	1 = 5×10 dots
BF	0 = can accept instruction	1 = internal operation in progress

The next slide represents the DDRAM address usage both for 1-line display as well as 2-line display.

Interfacing with LCD *cont'd …*

Display Size	Visible	
	Character Positions	DDRAM Addresses
1 × 8	00..07	0 ×00..0 ×07
1 × 16	00..15	0 ×00..0 ×0F
1 × 20	00..19	0 ×00..0 ×13
1 × 24	00..23	0 ×00..0 ×17
1 × 32	00..31	0 ×00..0 ×1F
1 × 40	00..39	0 ×00..0 ×27

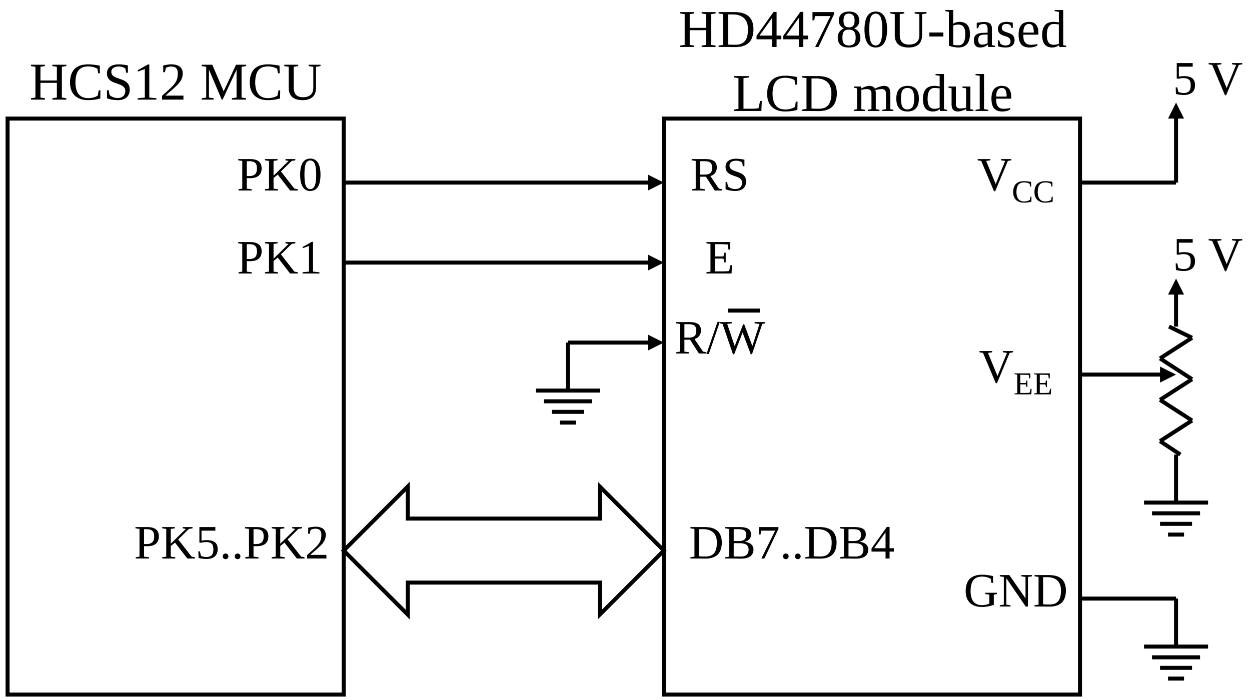
DDRAM address usage for a 1-line LCD

Display Size	Visible	
	Character Positions	DDRAM Addresses
2 × 16	00..15	0 ×00..0 ×0F + 0 ×40..0 ×4F
2 × 20	00..19	0 ×00..0 ×13 + 0 ×40..0 ×53
2 × 24	00..23	0 ×00..0 ×17 + 0 ×40..0 ×57
2 × 32	00..31	0 ×00..0 ×1F + 0 ×40..0 ×5F
2 × 40	00..39	0 ×00..0 ×27 + 0 ×40..0 ×67

DDRAM address usage for a 2-line LCD

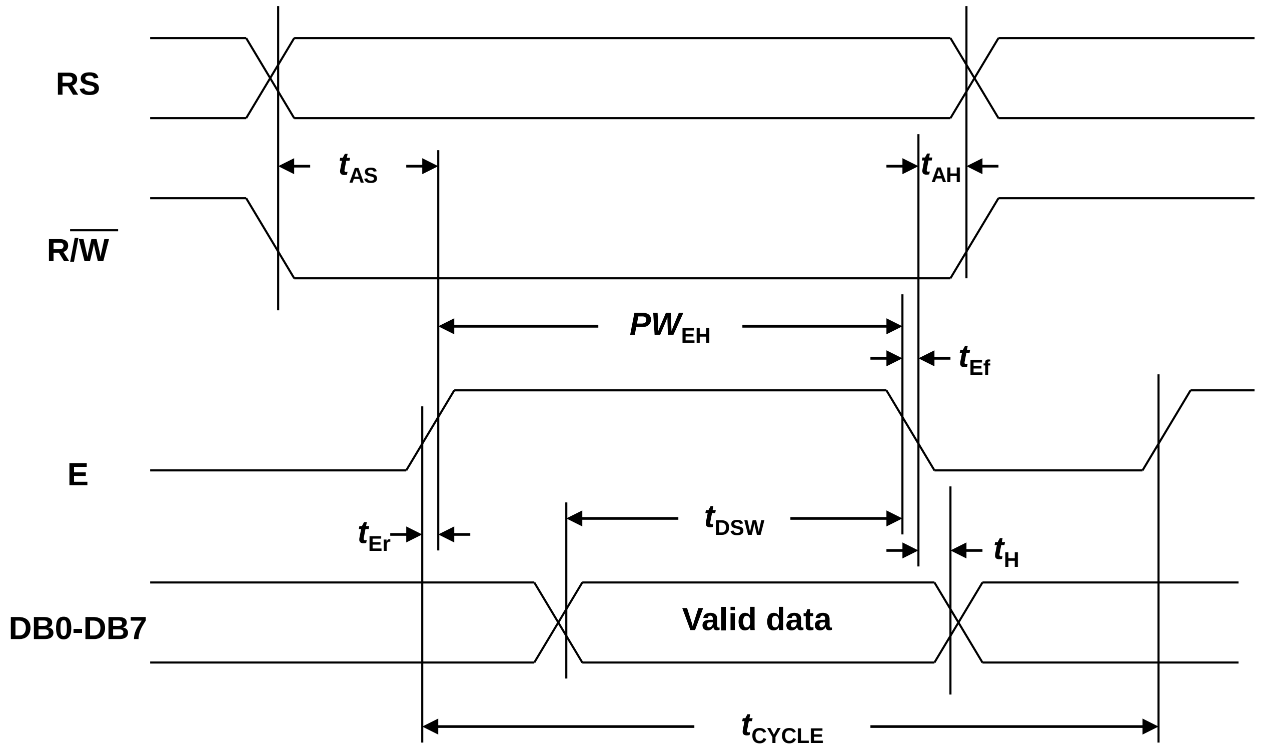
Interfacing the HD44780U to Dragon12

- It uses 4-bit data transfer. First the upper four bits are sent over DB7 ~ DB4 followed immediately by the lower four bits.
- The R/W signal to the LCD kit in the Dragon12 demo board is grounded, which prevents the user from polling the BF flag to determine whether the LCD internal operation has been completed.



LCD Interface Example (4-bit bus, used in Dragon12)

- Certain timing parameters must be satisfied in order to access the LCD successfully.
- The write timing diagram is shown below and the values of timing parameters are given in a table on next slide.



HD44780U LCD controller write timing diagram

HD44780U LCD bus timing parameters (1 MHz operation)

Symbol	Meaning	Min.	Typ.	Max.	Unit
t_{CYCLE}	Enable cycle time	1000	--	--	ns
PW_{EH}	Enable pulse width (high level)	450	--	--	ns
t_{Er}, t_{Ef}	Enable rise and decay time	--	--	25	ns
t_{AS}	Address setup time, RS, R/W, E	60	--	--	ns
t_{DDR}	Data delay time	--	--	360	ns
t_{DSW}	Data setup time	195	--	--	ns
t_H	Data hold time (write)	10	--	--	ns
t_{DHR}	Data hold time (read)	5	--	--	ns
t_{AH}	Address hold time	20	--	--	ns

- In the next few slides we will try to develop a set of subroutines that will perform certain functions for the LCD on the Dragon12 demo board such as ***cmd2lcd***, ***openlcd***, ***putc2lcd***,and ***puts2lcd***.

cmd2lcd:

- Write a subroutine that sends the command **cmd** to the LCD kit.
- The procedure for sending a command to the IR register of the LCD is as follows:
  1. Pull the RS and the E signals to low.
  2. Pull the R/$\overline{\text{W}}$ signal to low.
  3. Pull the E signal to high.
  4. Output data to the output port attached to the LCD data bus. One needs to configure port K for output before writing data to the LCD kit.
  5. Pull the E signal to low and make sure that the internal operation is complete.
- Note that with exception of (*clear display* & *cursor home*), every other operation takes 40 μs to be completed. For those two commands one needs extra delay loop.

```
cmd2lcd     PSHA                    ; save the command in stack
            BCLR    PORTK,RS        ; select the instruction register
            BSET    PORTK,E         ; pull the E signal high
        {   ANDA    #$F0            ; clear the lower 4 bits
        {   LSRA                    ; match the upper four bits
        {   LSRA                    ;  with the LCD data pins
        {   ORAA    #E              ; Maintain the E signal value
450 ns  {   STAA    PORTK           ; send the command along w/ RS & E
        {   NOP                     ; extend the duration of E pulse
        {   NOP                     ;
        {   NOP                     ;
            BCLR    PORTK,E         ; pull the E signal low
            PULA                    ; retrieve the LCD command
            ANDA    #$0F            ; clear the upper four bits
            LSLA                    ; match the lower four bits
            LSLA                    ;  with the LCD data pins
            BSET    PORTK,E         ; pull the E signal high
            ORAA    #E              ; maintain the E signal value
            STAA    PORTK           ; send the lower 4-bit w/RS & E
            NOP                     ; extend the duration of E pulse
            NOP                     ;
            NOP                     ;
```

```
                BCLR    PORTK,E             ; clear E signal to comp. the operation
*       Add 40 μs delay to complete the internal operation
                LDY     #240                ; 240 x 4 = 960 cycles = 40 μs
dly             DEY                         ; 1 cycle
                BNE     dly                 ; 3 cycles
                RTS

RS              EQU     $01
E               EQU     $02
```

- Before using the LCD, one must configure it properly. The configuration of the LCD involves at least the following for LCD instructions:
  1. Entry Mode Set – The common setting for this instruction is to move the cursor to the right after reading or writing a character from/to the LCD.
  2. Display ON/OFF – The common setting for this instruction is to turn ON the display, cursor, and blinking.

3. Function Set – This instruction sets the number of rows for display, the font size, and the width of the interface data (4 or 8 bits).
4. Clear Display – Before outputting any data, it is always a good idea to clear the LCD screen and move the cursor to the home position (upper left corner).

The following subroutine performs the LCD configuration:

```
openlcd         MOVB   #$FF,DDRK       ; configure port K for output
                JSR    d_10ms          ; wait for LCD to be ready
                LDAA   #$28            ; set 4-bit data, 2-line display,  5×8 font
                JSR    cmd2lcd          ;
                LDAA   #$0F            ; turn on display, cursor, and blinking
                JSR    cmd2lcd          ;
                LDAA   #$06            ; move cursor right (entry mode set
                JSR    cmd2lcd          ; instruction)
                LDAA   #$01            ; clear the screen and return to home
                JSR    cmd2lcd          ; position
```

Interfacing with LCD *cont'd …*

```
*    Wait until "clear display" command is complete
             LDY     #10000          ; 10000 x 4 = 40,000 cycles = 1.67ms
dly1         DEY                     ; 1 cycle
             BNE     dly1            ; 3 cycles

             RTS

d_10ms       LDY     #60000          ; 60000 x 4 = 240,000 cycles = 10ms
dly          DEY                     ; 1 cycle
             BNE     dly             ; 3 cycles
             RTS
```

- The procedure for writing a byte to the LCD data register is as follows:
  1. Pull the RS signal to high.
  2. Pull the R/W signal to low.
  3. Pull the E signal to high.
  4. Output data to the I/O port attached to the LCD data bus.
  5. Pull the E signal to low and make sure that the internal operation is complete.

Interfacing with LCD *cont'd …*

```
putc2lcd        PSHA                    ; save the command in stack
                BSET    PORTK,RS        ; select LCD data register
                BSET    PORTK,E         ; pull the E signal high
                ANDA    #$F0            ; clear the lower 4 bits
                LSRA                    ; match the upper four bits
                LSRA                     ; with the LCD data pins
                ORAA    #$03            ; keep E & RS signals unchanged
                STAA    PORTK           ; send the upper 4-bit along w/ RS & E
                NOP                     ; extend the duration of E pulse
                NOP                      ;
                NOP                       ;
                BCLR    PORTK,E         ; pull the E signal low
                PULA                    ; retrieve the LCD command
                ANDA    #$0F            ; clear the upper four bits
                LSLA                    ; match the lower four bits
                LSLA                     ; with the LCD data pins
                BSET    PORTK,E         ; pull the E signal high
                ORAA    #$03            ; keep E & RS signals unchanged
                STAA    PORTK           ; send the lower 4-bit w/RS & E
                NOP                     ; extend the duration of E pulse
                NOP                      ;
                NOP                       ;
                BCLR    PORTK,E              ; clear E signal to comp. operation
*       Add 40 µs delay to complete the internal operation
                LDY     #240                 ; 240 x 4 = 960 cycles = 40 µs
dly             DEY                          ; 1 cycle
                BNE     dly                  ; 3 cycles
                RTS
```

The subroutine that outputs a NULL-terminated string pointed to by the index register X is as follows:

```
puts2lcd        LDAA    1,X+            ; get a character from string
                BEQ     done_puts       ; reach NULL character?
                JSR     putc2lcd        ; if not, output to screen
                JMP     puts2lcd        ; continue till whole string is outputed
done_puts               RTS
```

Interfacing with LCD *cont'd …*

Write an assembly program to test the previous four subroutines by displaying the following messages on two lines:

Hello world!

I am ready!

```
#include        "C:\Dragon12\examples\Reg9s12.h"
                ORG     $1000
                JSR     openlcd         ; initialize the LCD
                LDX     #msg1_lcd       ; point to 1st message
                JSR     puts2lcd        ; output it to LCD screen
                LDAA    #$C0            ; move the cursor to 2nd row
                JSR     cmd2lcd          ;
                LDX     #msg2_lcd       ; point to 2nd message
                JSR     puts2lcd         ;
                SWI

msg1_lcd        FCC     'Hello world!'
                DC.B    0
msg2_lcd        FCC     'I am ready!'
                DC.B    0               ; clear the upper four bits
```

```
*****************************************************************************************
*     Write a program that as you type on keyboard it appears on LCD and scrolls     *
*****************************************************************************************
#include        reg9s12.h                 ; include register equates
                ORG     $1000
                JSR     openlcd           ; initialize the LCD
                LDAA    #$80              ; start at home position
                JSR     cmd2lcd
                LDAA    #$0C              ; turn off cursor and blinking
                JSR     cmd2lcd            ;
nexchar         LDX     getchar           ; get next character
                JSR     0,X
                STAB    endmsg            ; store it at the end of buffer
                LDX     #msg              ; scroll buffer by 1
                LDAB    #8                 ;
shift           MOVW    1,X,2,X+            ;
                DBNE    B,shift              ;
                CLR     0,X               ; put null at the end of buffer
                LDX     #msg              ; point to 1st message
                JSR     puts2lcd          ; output it to LCD screen
                LDAA    #$80              ; start at home position
                JSR     cmd2lcd
                JMP     nexchar           ; get next character
```

```
cmd2lcd PSHA                         ; save the command in stack
        BCLR    PORTK,RS             ; select the instruction register
        BSET    PORTK,E              ; pull the E signal high
        ANDA    #$F0                 ; clear the lower 4 bits
        LSRA                         ; match the upper four bits
        LSRA                          ; with the LCD data pins
        ORAA    #E                   ; Maintain the E signal value
        STAA    PORTK                ; send the command along w/ RS & E
        NOP                          ; extend the duration of E pulse
        NOP                           ;
        NOP                            ;
        BCLR    PORTK,E                          ; pull the E signal low
        PULA                         ; retrieve the LCD command
        ANDA    #$0F                 ; clear the upper four bits
        LSLA                         ; match the lower four bits
        LSLA                          ; with the LCD data pins
        BSET    PORTK,E                          ; pull the E signal high
        ORAA    #E                   ; maintain the E signal value
        STAA    PORTK                ; send the lower 4-bit w/RS & E
        NOP                          ; extend the duration of E pulse
        NOP                           ;
        NOP                            ;
        BCLR    PORTK,E              ; clear E signal to complete the operation
```

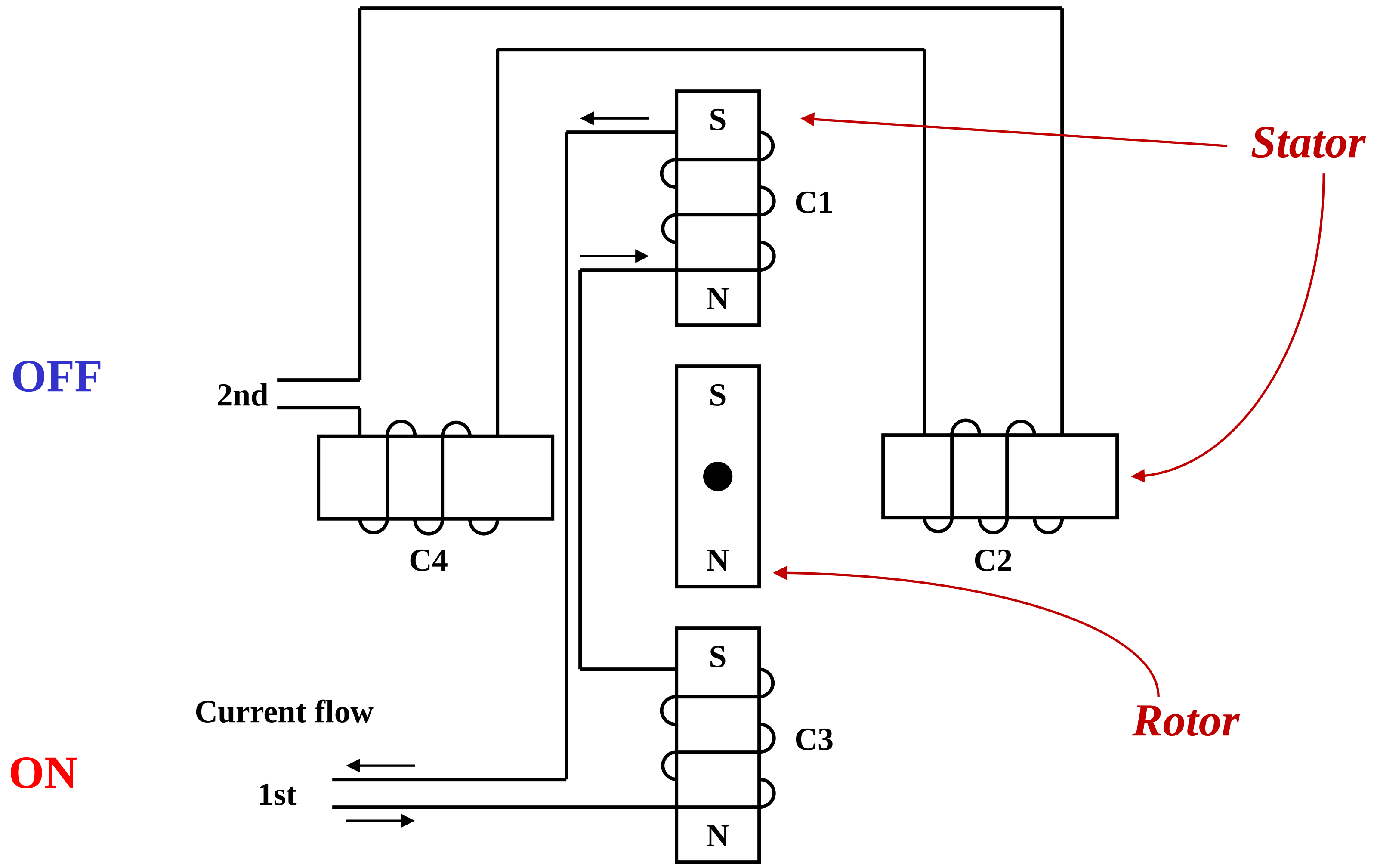
```
*          Add 40 ?s delay to complete the internal operation
           LDY     #240                 ; 240 x 4 = 960 cycles = 40 micro_s
sdly       DEY                          ; 1 cycle
           BNE     sdly                 ; 3 cycles
           RTS

openlcd    MOVB    #$FF,DDRK            ; configure port K for output
           JSR     d_10ms               ; wait for LCD to be ready
           LDAA    #$28                 ; set 4-bit data, 2-line display, 5x8 font
           JSR     cmd2lcd               ;
           LDAA    #$0F                 ; turn on display, cursor, and blinking
           JSR     cmd2lcd               ;
           LDAA    #$06                 ; move cursor right (entry mode set
           JSR     cmd2lcd               ; instruction)
           LDAA    #$01                 ; clear the screen and return to home
           JSR     cmd2lcd               ; position
*    Wait until “clear display” command is complete
           LDY     #10000               ; 10000 x 4 = 40,000 cycles = 1.67ms
dly1       DEY                          ; 1 cycle
           BNE     dly1                 ; 3 cycles
           RTS
```

```
d_10ms  LDY     #60000          ; 60000 x 4 = 240,000 cycles = 10ms
dly     DEY                     ; 1 cycle
        BNE     dly             ; 3 cycles
        RTS

putc2lcd PSHA                   ; save the command in stack
        BSET    PORTK,RS                ; select LCD data register
        BSET    PORTK,E                 ; pull the E signal high
        ANDA    #$F0            ; clear the lower 4 bits
        LSRA                    ; match the upper four bits
        LSRA                     ; with the LCD data pins
        ORAA    #$03            ; keep E & RS signals unchanged
        STAA    PORTK           ; send the upper 4-bit along w/ RS & E
        NOP                     ; extend the duration of E pulse
        NOP                      ;
        NOP                       ;
        BCLR    PORTK,E                 ; pull the E signal low
        PULA                    ; retrieve the LCD command
        ANDA    #$0F            ; clear the upper four bits
        LSLA                    ; match the lower four bits
        LSLA                     ; with the LCD data pins
        BSET    PORTK,E                 ; pull the E signal high
```

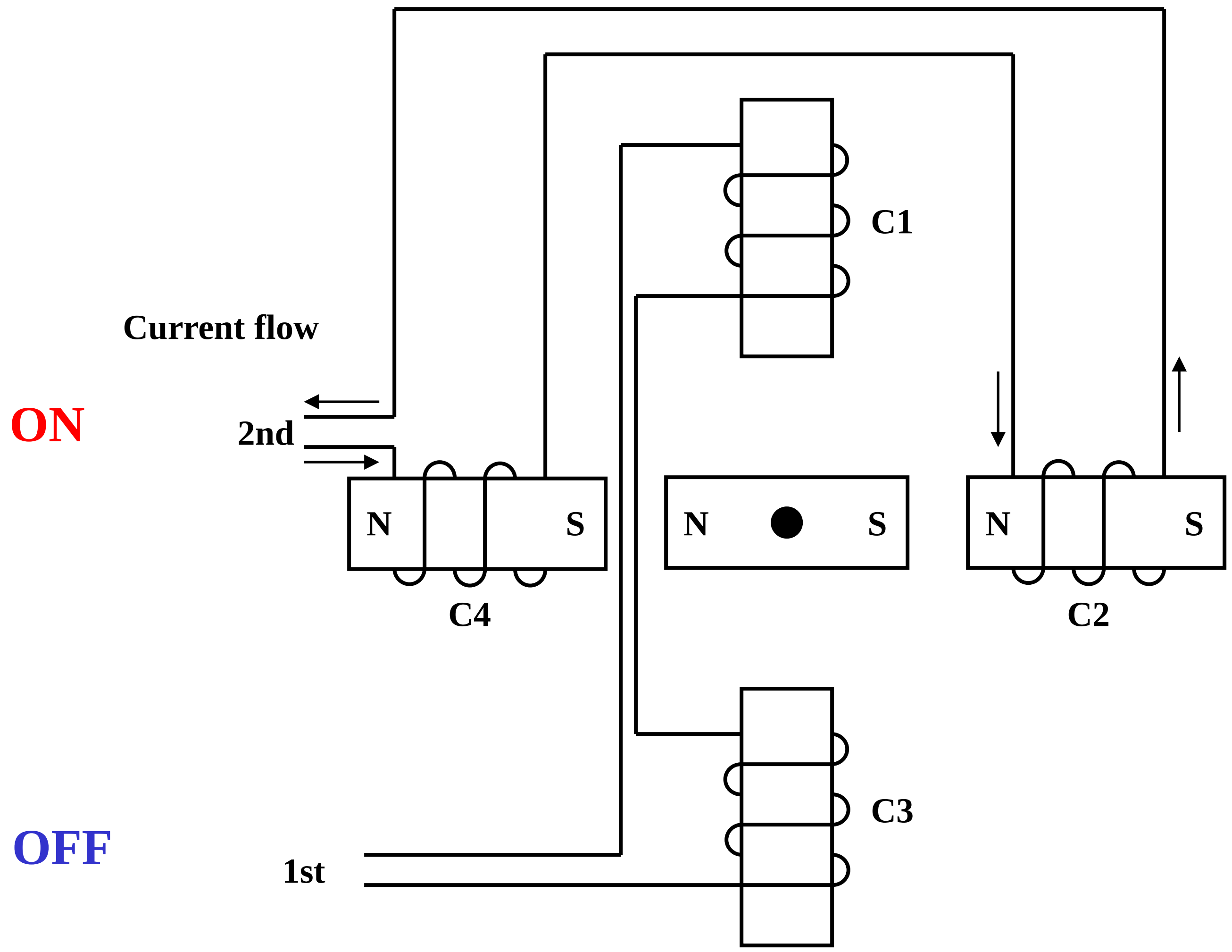
```
ORAA    #$03                    ; keep E & RS signals unchanged
        STAA    PORTK           ; send the lower 4-bit w/RS & E
        NOP                     ; extend the duration of E pulse
        NOP                     ;
        NOP                     ;
        BCLR    PORTK,E         ; clear E signal to complete the operation
*       Add 40 ?s delay to complete the internal operation
        LDY     #240            ; 240 x 4 = 960 cycles = 40 ?s
dly2    DEY                     ; 1 cycle
        BNE     dly2            ; 3 cycles
        RTS
puts2lcd        LDAA    1,X+            ; get a character from string
                BEQ     done_puts       ; reach NULL character?
                JSR     putc2lcd        ; if not, output to screen
                JMP     puts2lcd        ; continue till whole string is done
done_puts       RTS
getchar         EQU    $EE84
RS              EQU     $01
E               EQU     $02
msg             FCC     '                ‘      ; 16 space characters
endmsg          DB      0
                END
```

Stepper Motor Control

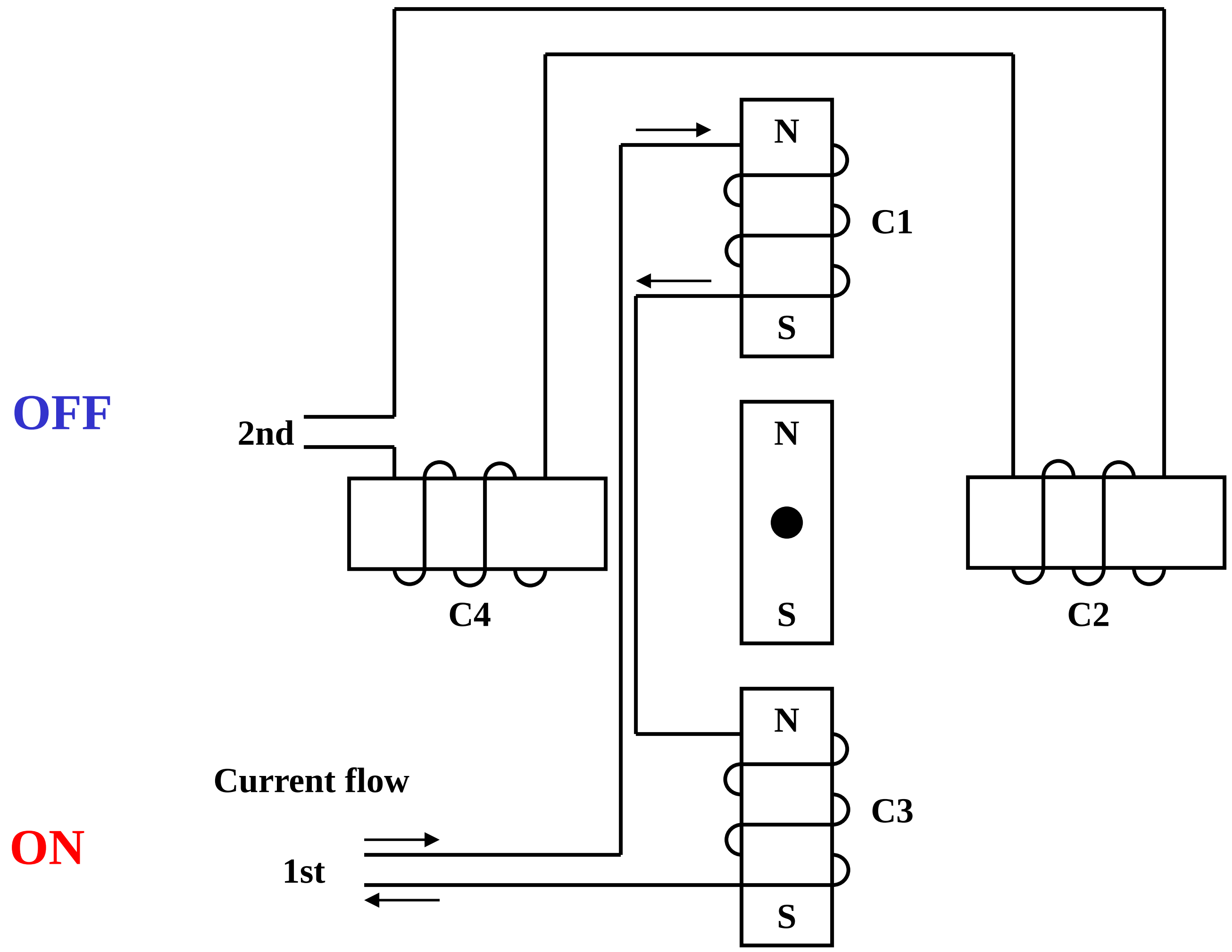
- Stepper motors are digital motors. They are convenient for applications where a high degree of positional control is required.
- In its simplest form, a stepper motor has a permanent magnet ***rotor*** and a ***stator*** consisting of two coils.
- The rotor aligns with the stator coil that is energized.
- The direction of the current determines the polarity of the magnetic field, and thus the angular position of the rotor.
- For two sets of stators, the rotor will rotate in 90^o angles.
- The next few slides will show this operation.



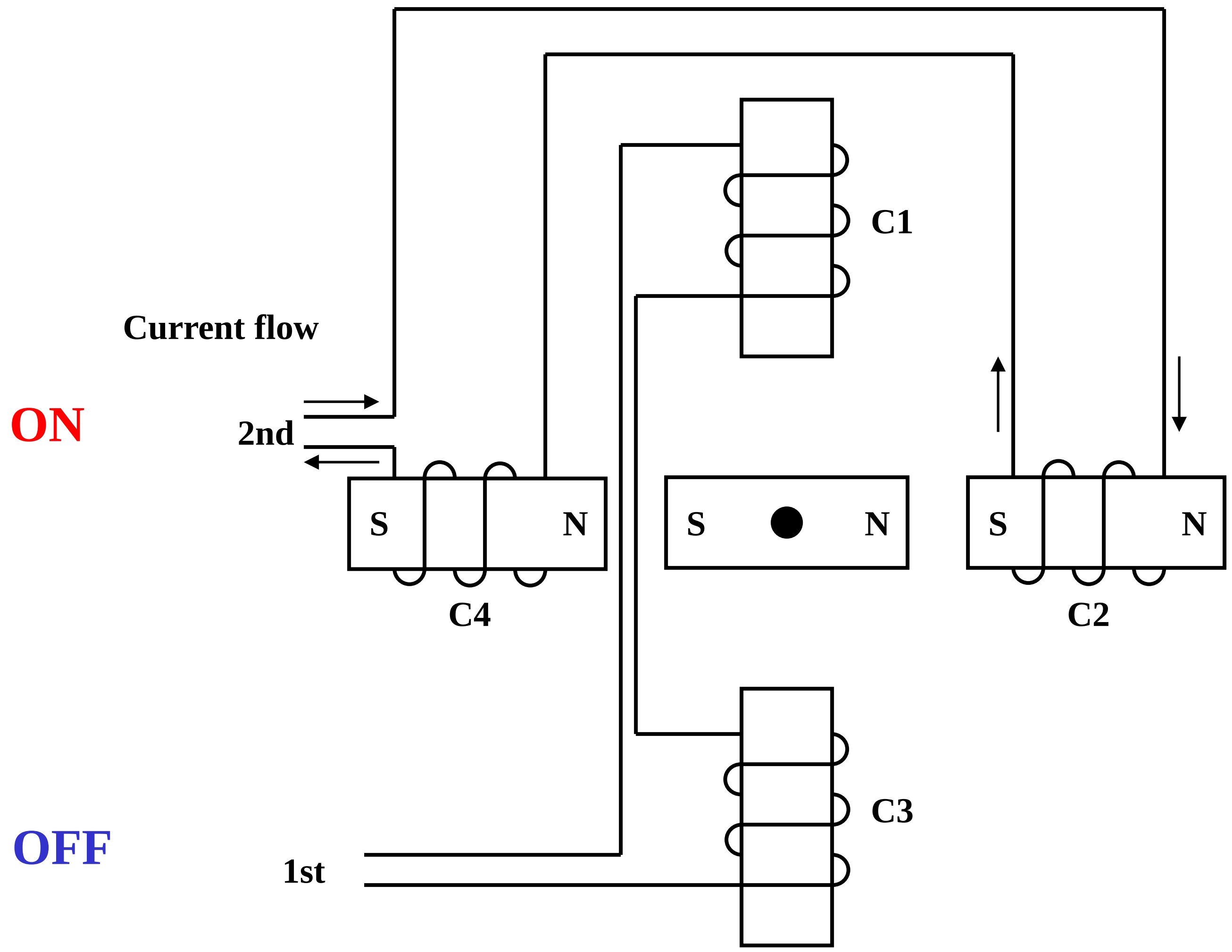
Stepper Motor full step 1



Stepper Motor full step 2



Stepper Motor full step 3



Stepper Motor full step 4