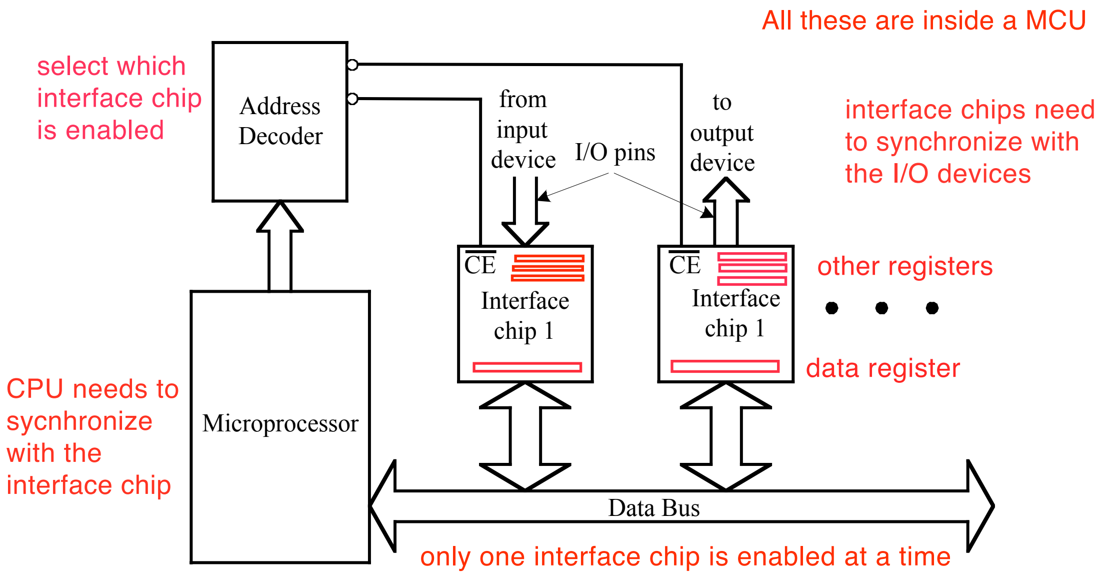
Chapter 7: Parallel Ports

- I/O devices are also called peripheral devices.
- I/O devices are pieces of equipment that exchange data with a computer.
- Examples include switches, LEDs, keyboards, and computer screen.

Interface (Peripheral) Chips


All these are inside a MCU
select which interface chip is enabled
Address Decoder
from input device
I/O pins
to output device
interface chips need to synchronize with the I/O devices
CE
Interface chip 1
CE
Interface chip 1
other registers
data register
CPU needs to sycnhronize with the interface chip
Microprocessor
Data Bus
only one interface chip is enabled at a time

- Resolve the differences, such as speed and electrical properties, between the CPU and I/O devices.

- Synchronize (the timing of) data transfer between the CPU and I/O devices.

- Consist of four kind of registers:
  1. ***Control registers*** allow us to set up the parameters for the desired I/O operations.
  2. ***Status registers*** report the progress and status of the I/O operations.
  3. ***Data direction registers*** allow us to select the data transfer direction for each (bidirectional) I/O pin.
  4. ***Data registers*** hold the data to be sent to the output device (from the CPU) or hold the new data placed by the input device (unitl they are read by the CPU).

- An interface chip has pins that are connected to the CPU and I/O port pins that are connected to the I/O devices.

- In a large system, many I/O devices are attached to the data bus via interface chips. Only one device is allowed to drive data to data bus at a time. Otherwise, data bus contention can result.

- Address decoder makes sure that each time one and only one peripheral device responds to the CPU's I/O request.

- Each interface chip has a chip enable input which, when asserted, allow the interface chip to react to the data transfer request.

- Data transfer between an I/O device and the CPU can be proceeded bit-by-bit (***serial***) or in multiple bits (***parallel***).

I/O Addressing

- An address must be assigned to each of the I/O registers in an interferace chip for accessing.

- Two ways of accessing the I/O registers:
  1. **Isolated I/O scheme**
     - CPU has dedicated instructions and addressing modes for I/O operations.
     - CPU has a separate address space for I/O devices.

  2. **Memory-mapped I/O scheme** (current trend; used in HCS12)
     - CPU uses the same instruction set and addressing modes to perform memory accesses and I/O operations.
     - I/O devices and memory components (e.g., EEPROM, Flash, SRAM) are resident in the same memory space.

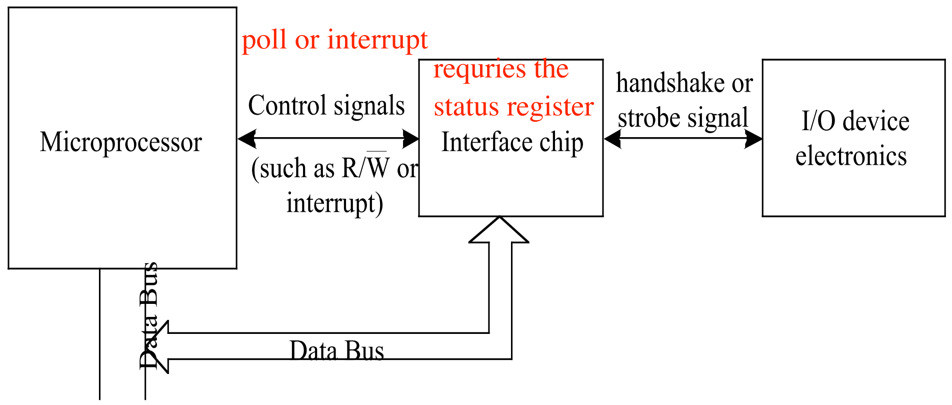
I/O Synchronization

- When inputting data, the CPU reads data from the interface chip. So, there must be a mechanism to make sure that data is valid when the CPU reads it.

- When outputting data, the CPU writes data into the interface chip. Again, there must be a mechanism to make sure that the output device is ready to accept data when the CPU outputs it.

- These mechanisms need I/O synchronization, which is a 2-step process:
  1. Synchronizing data transfer between the CPU and interface chip.
  2. Synchronizing data transfer between the interface chip and I/O device.

CPU canot talk to the I/O devices directly!



1. Synchronizing the CPU and Interface Chip

To obtain valid data from an input device, the CPU must make sure that the interface chip has latched the data to the data register from the input device.

Polling method

- For input –
  1. The CPU checks a ***status bit*** of the interface chip to find out if the interface chip has received new data from the input device.

data from interface chip => CPU

  2. The CPU keeps checking the status bit until it indicates that the data register is filled.
  3. Then, the CPU reads the data from the data register.

  - *Con*: The CPU is tied up and cannot do anything else while waiting for the data.
  - *Pro*: A very simple method to implement and is often used when the CPU has nothing else to do while waiting for completion of the input process.

- For output –
  1. The interface chip uses a ***status bit*** to indicate whether the data register of the interface chip is empty for accepting new data from the CPU.

data from CPU => interface chip

  2. The CPU keeps checking the status bit of the interface chip until it indicates that the data register is empty.
  3. Then, the CPU writes data into the data register.

Interrupt-driven method

- For input –

data from interface chip => CPU

  1. The interface chip interrupts the CPU whenever the interface chip has received new data from the input device.

  2. The CPU then executes the associated interrupt service routine to read the data from the data register.

- For output –

data from CPU => interface chip

  1. The interface chip interrupts the CPU whenever the data register is empty and ready to accept new data from the CPU.

  2. The CPU then executes the associated interrupt service routine to output the data to the data register.

2. Synchronizing the Interface Chip and I/O Devices

The interface chip is responsible for making sure that data are properly transferred to and from I/O devices.

Brute-force method is useful when the timing of data is not important. This method is used to test voltage level of a signal, set the voltage of an output pin to high or low, or drive LEDs.

voltage from input device => interface chip

For input -- nothing special is done. The CPU reads the interface chip and the interface chip returns the voltage levels on the input port pins to the CPU.

voltage from interface chip => output device

For output -- nothing special is done. The interface chip places the data that it received from the CPU directly on the output port pins.

Strobe method uses a strobe signal to indicate that data are stable on I/O port pins.

input device --> strobe signal --> interface chip

- For input – the input device asserts a strobe signal when the data are stable on the input port pins. The interface chip stores the data into its data register using the strobe signal.

interface chip --> strobe signal --> output device

- For output -- the interface chip places the data on output port pins that it received from the CPU and asserts the strobe signal. The output device fetches the data from the data register using the strobe signal.

- None of the HCS12 parallel ports support this method.

Handshake method is used when the timing of data is crucial. This method guarantees correct data transfer between an interface chip and an I/O device.

- For example, it takes a much longer time to print a character than it does to send a character to the printer electronics. So, data shouldn't be sent to the printer if it is still printing.

- Two handshake signals are used to synchronize the data transfer. One signal, say H1, is asserted by the interface chip. The other signal, say H2, is asserted by the I/O device.

- Two handshake modes are available -- ***pulse mode*** and ***interlocked mode***.

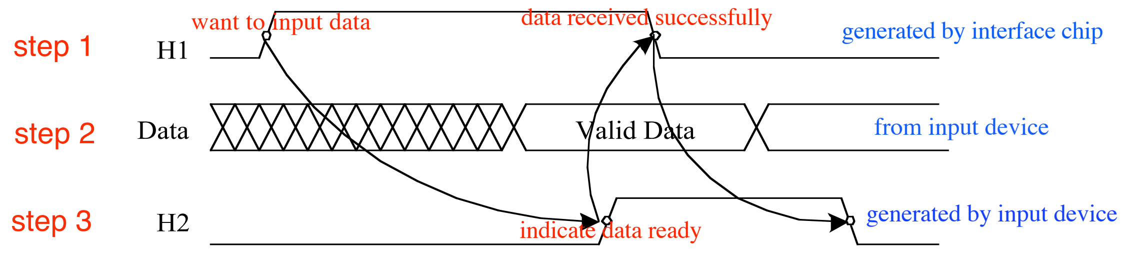
Input Handshake Protocol (from input device to interface chip)

Step 1: The interface chip asserts (or pulses) H1 to indicate its intention to input data.

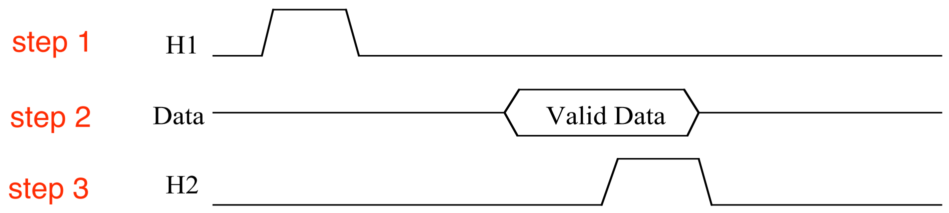
Step 2: The input device puts data on the data port pins and also asserts (or pulses) H2.

for interlock only

Step 3: The interface chip latches the data and de-asserts H1. After some delay, the input device also de-asserts H2.



(a) Interlocked



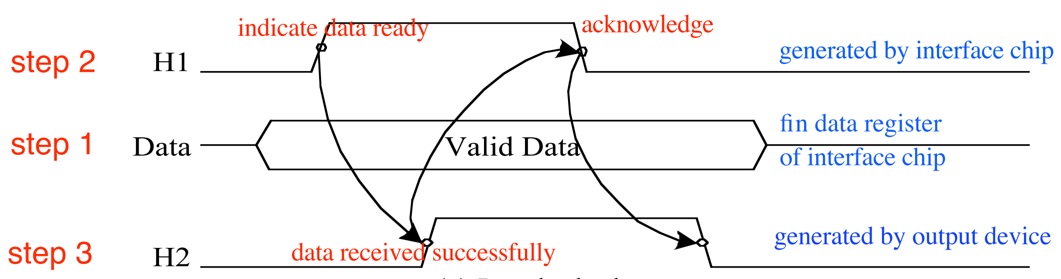
(b) Pulse mode

Output Handshake Protocol (from interface chip to output device)

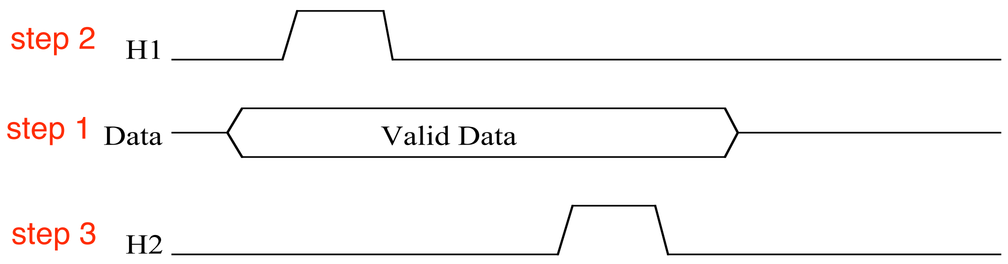
Step 1: The interface chip places data on the port pins and asserts (or pulses) H1 to indicate that it has valid data to be output.

Step 2: The output device latches the data and asserts (or pulses) H2 to acknowledge the receipt of data.

for interlock only Step 3: The interface chip de-asserts H1 following the assertion of H2. The output device then de-asserts H2.



(a) Interlocked



(b) Pulse Mode

HCS12 Parallel Ports

- The HCS12 members have 48 to 144 I/O pins arranged in 3 to 12 ports and packaged in a quad flat pack (QFP) or low profile flat pack (LQFP).

- All I/O port pins serve multiple functions. When a peripheral function is enabled, its associated pins cannot be used as I/O pins.

- In general, each HCS12 I/O port has several registers to support its operation. Registers related to I/O ports have been assigned a mnemonic name and the user can use these names to refer to them. (See the reg9s12.h file.)

- A ***data register*** (refered to with the prefix "Port" or "PT") is used to hold the data (for output) to be applied to the port pins or the current signal levels (for input) of the port pins.

- A ***data direction register*** (refered to with the prefix "DDR") is used to select the I/O direction (input or output).
  - To configure a pin for output, write a 1 to the associated bit in the DDR.
  - To configure a pin for input, write a 0 to the associated bit in DDR.

```
movb    #$FF, DDRA      ; configure all 8 pins in port A for output
movb    #$37, PortA     ; output  0011 0111  to Port A

movb    #00, DDRB       ; configure all 8 pins in port B for input
ldaa    PortB           ; read data from port B into accumulator A

bset    DDRK, $81       ; configure port K pin 7 and 1 for output
                        ; but other pins for input ($81 = 1000 0001)
```

Pull-Up Control Register (PUCR)

A good design practice is to tie unused input pins to either a high or low logic level. This register selects the pull-up resistors for the pins associated with Port A, B, E, and K.

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

PUPKE: pull-up Port K enable
  0 = Port K pull-up resistors are disabled
  1 = Port K pull-up resistors are enabled
PUPEE: pull-up Port E enable
  0 = Port E input pins 7 and 4-0, pull-up resistors are disabled
  1 = Port E input pins 7 and 4-0, pull-up resistors are enabled
PUPBE: pull-up Port B enable
  0 = Port B pull-up resistors are disabled
  1 = Port B pull-up resistors are enabled
PUPAE: pull-up Port A enable
  0 = Port A pull-up resistors are disabled
  1 = Port A pull-up resistors are enabled

Figure 7.8 Pull-Up control register

Reduced Drive Control (RDRIV)

- A large capacitive load at an output pin requires high drive (i.e., amount of current) to achieve fast switching speed.
- Higher drive also means higher power consumption and possible radio frequency interference.
- This register selects reduced drive for the pins associated with Port A, B, E, and K.

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

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

Figure 7.9 Reduced Drive Register(RDRIV)

Port A ($0000) and Port B ($0001)

- In general, these two ports are used for ***general-purpose I/O*** ports.
- The directions of port A (PortA at memory location $0000) pins are configured by setting or clearing the associated bits in the DDRA ($0002) data direction register.
- The directions of port B (PortB at $0001) pins are configured by setting or clearing the associated bits DDRB ($0003).
- When setting to 1, a pin is configured for output. Otherwise, it is configured for input.

```
PortA       equ     $0000         ; address of port A
DDRA        equ     $0002         ; address of DDR A
            …                     ; main program with other insructions
            movb    #$FF,DDRA     ; configure all 8 pins of port A for output
            staa    PortA         ; output the content of A to port A
            …                     ; the rest of the main program
```

Port E ($0008)

- PortE ($0008) can also be used for ***bus control*** and ***interrupt service request*** signals. When a pin is not used for one of these functions, it can be used as a **I/O** pin.

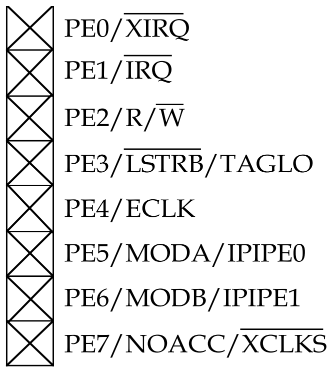

PE0/$\overline{\text{XIRQ}}$
PE1/$\overline{\text{IRQ}}$
PE2/R/$\overline{\text{W}}$
PE3/$\overline{\text{LSTRB}}$/TAGLO
PE4/ECLK
PE5/MODA/IPIPE0
PE6/MODB/IPIPE1
PE7/NOACC/$\overline{\text{XCLKS}}$

Figure 7.5 Port E pins and their alternate functions

- DDRE ($0009) determines whether a pin is input or output when it is used for general-purpose I/O. As PE[1:0] are input pins, only DDRE[7:2] have effects.

- PortE[7] serves three functions: i) a general-purpose I/O pin; ii) a signal that indicates the MCU is not accessing external memory; and iii) a siganl that selects between the external clock or crystal oscillator signal as the clock input to the HCS12.

- PortE[6:5] serve three functions: i) general-purpose I/O pins; ii) signals that set the operation mode of the HCS12 after reset; and iii) instruction queue tracking signals (in expanded mode).

- PortE[4] serves two functions: i) a general-purpose I/O pin; and ii) the E-clock output (in expanded mode).

- Pin 3 serves three functions: i) a general-purpose I/O pin; ii) the LCD front-plane segment driver output pin; and iii) the low-byte strobe signal to indciate the type of access (in expanded mode).

- PortE[2] serves two functions: i) a general-purpose I/O pin; and ii) a signal to indicate whether the current bus cycle is a read or write cycle (in expanded mode).

- PortE[1] serves two functions: i) a general-purpose I/O pin; and ii) the $\overline{\text{IRQ}}$ interrupt input.

- PortE[0] serves two functions: i) a general-purpose I/O pin; and ii) the $\overline{\text{XIRQ}}$ interrupt input.

- PortE[1:0] can only be used for input. (The states of these two pins can be read in the data register even when they are used for $\overline{\text{IRQ}}$ and $\overline{\text{XIRQ}}$ interrupts.)

Port K ($0032)

Used for general-purpose I/O in a single-chip mode. Port K has a data register (PortK at $0032) and a direction register (DDRK at $0033).

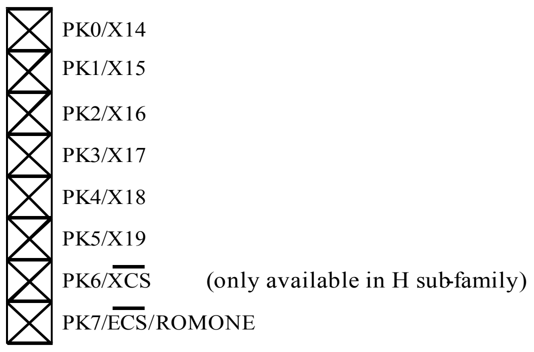

PK0/X14
PK1/X15
PK2/X16
PK3/X17
PK4/X18
PK5/X19
PK6/$\overline{\text{XCS}}$
(only available in H sub-family)
PK7/$\overline{\text{ECS}}$/ROMONE

Figure 7.10 Port K pins and their alternate functions

Port T ($0240)

- Port T has a data register (PTT at $0240), a direction register (DDRT at $0242), an input register (PTIT at $0241), a reduced drive register (RDRT at $0243), a pull device enable register (PERT at $0244), and a port polarity select register (PPST at $0245).

- Port T pins are also used as timer input capture/output compare pin.

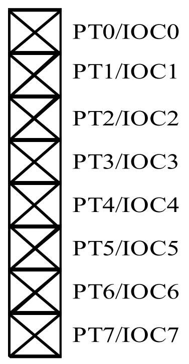


Figure 7.14 Port T pins and their alternate functions

- PTIT input register allows the user to read back the status of Port T pins.
- RDRT reduced drive register can configure the current ouptut strength of each port pin as full or reduced load.
- PERT pull device enable register is used to enable input pins for pull devices.
- PPST port polarity select register selects whether a pull-down or pull-up device is connected to the pin.

	7	6	5	4	3	2	1	0
	RDRT7	RDRT6	RDRT5	RDRT4	RDRT3	RDRT2	RDRT1	RDRT0
reset:	0	0	0	0	0	0	0	0

RDRT[7:0]: Reduced drive Port T
0 = full drive strength at output
1 = associated pin drives at about1/3 of the full drive strength

Figure 7.11 Port T Reduced Drive register (RDRT)

	7	6	5	4	3	2	1	0
	PERT7	PERT6	PERT5	PERT4	PERT3	PERT2	PERT1	PERT0
reset:	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

Figure 7.12 Port T Pull Device Enable register (PERT)

	7	6	5	4	3	2	1	0
	PPST7	PPST6	PPST5	PPST4	PPST3	PPST2	PPST1	PPST0
reset:	0	0	0	0	0	0	0	0

PPST[7:0]: pull device enable Port T
0 = A pull-up device is connected to the associated ort 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

Figure 7.13 Port T Polarity Select register (PPST)

Port S ($0248)

- Port S is the 8-bit interface to the standard serial interface consisting of serial communication interface (SCI) and serial peripheral interface (SPI).
- Port S is also available for general-purpose I/O when serial interfaces are not enabled. It has a data register (PTS at $0248) and a direction register (DDRS at $024A).

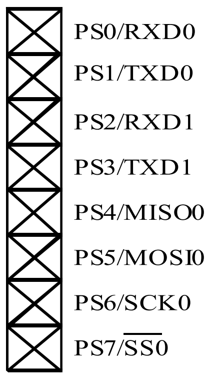


Figure 7.16 Port S pins and their alternate functions

Port M ($0250)

- Port M has all the equivalent registers that Port S has. It also has a module routing register (MODRR), which configures the rerouting of CAN0, CAN4, SPI0, SPI1, and SPI2 on defined port pins.
- Port M has a data register (PPM at $0250) and a direction register (DDRM at $0252).

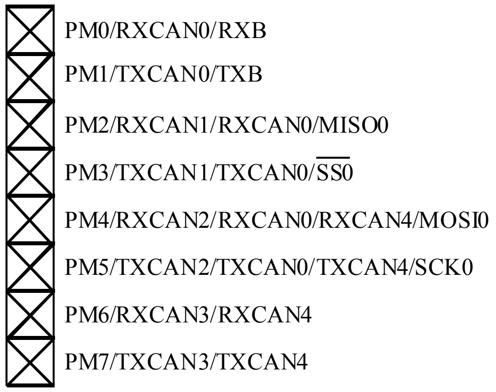

PM0/RXCAN0/RXB
PM1/TXCAN0/TXB
PM2/RXCAN1/RXCAN0/MISO0
PM3/TXCAN1/TXCAN0/$\overline{SS0}$
PM4/RXCAN2/RXCAN0/RXCAN4/MOSI0
PM5/TXCAN2/TXCAN0/TXCAN4/SCK0
PM6/RXCAN3/RXCAN4
PM7/TXCAN3/TXCAN4

Figure 7.18 Port M pins and their alternate functions

Port P ($0258), H ($0260), and J ($0268)

- These three I/O ports have the same set of registers:
  - Port I/O register [PTP ($0258), PTH ($0260), PTJ ($0268)]
  - Port Input Register (PTIP, PTIH, PTIJ)
  - Port Data Direction Register [DDRP ($025A), DDRH ($0262), DDRJ ($026A)]
  - Port Reduced Drive Register (RDRP, RDRH, RDRJ)
  - Port Pull Device Enable Register (PERP, PERH, PERJ)
  - Port Polarity Select Register (PPSP, PPSH, PPSJ)
  - Port Interrupt Enable Register (PIEP, PIEH, PIEJ)
  - Port Interrupt Flag Register (PIFP, PIFH, PIFJ)

- These ports have edge-triggered interrupt capability by writing to the port interrupt enable registers. The interrupt edges can be rising or falling and are programmed through port pull device enable register and port polarity select register.

- The SPI function pins can be rerouted to Port H and P.

	7	6	5	4	3	2	1	0
	PIEH7	PIEH6	PIEH5	PIEH4	PIEH3	PIEH2	PIEH1	PIEH0
reset:	0	0	0	0	0	0	0	0

PIEH[7:0]: Interrupt enable Port H
0 = interrupt is disabled
1 = interrupt is enabled

Figure 7.19 Port H Interrupt Enable Register (PIEH)

	7	6	5	4	3	2	1	0
	PIFH7	PIFH6	PIFH5	PIFH4	PIFH3	PIFH2	PIFH1	PIFH0
reset:	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)

Figure 7.20 Port P Interrupt Flag Register (PIFH)

Port AD0 ($008F) and AD1 ($012F)

- These two ports are analog input interfaces to the analog-to-digital converter subsystem (10 bits, 8 channels).
- When A/D function is disabled, these two ports can be used as general-purpose input ports.
- Pins are inputs only. No data direction register. PORTAD0 ($008F) and PORTAD1 ($012F) are data registers.
- Each ATD module has a Digital Input Enable Register. In order to use an A/D pin as a digital input (such as general-purpose input), its associated bit in this register needs to be set to 1.

	7	6	5	4	3	2	1	0
	IEN7	IEN6	IEN5	IEN4	IEN3	IEN2	IEN1	IEN0
reset:	0	0	0	0	0	0	0	0

IENx: ATD digital input enable on channel x
0 = disable digital input buffer to PTADx pin
1 = enable digital input buffer to PTADx pin

Figure 7.24 ATD Input enable register(ATD0DIEN and ATD1DIEN)

Pin Name	Pin #	I/O Usage
PA0 (output)	Pin 57	Col_0 of keypad
PA1 (output)	Pin 58	Col_1 of keypad
PA2 (output)	Pin 59	Col_2 of keypad
PA3 (output)	Pin 60	Col_3 of keypad
PA4 (input)	Pin 61	Row_0 of keypad
PA5 (input)	Pin 62	Row_1 of keypad
PA6 (input)	Pin 63	Row_2 of keypad
PA7 (input)	Pin 64	Row_3 of keypad
PB0 (output)	Pin 24	LED0 or H-bridge
PB1 (output)	Pin 25	LED1 or H-bridge
PB2 (output)	Pin 26	LED2 or H-bridge
PB3 (output)	Pin 27	LED3 or H-bridge
PB4 (output)	Pin 28	LED4
PB5 (output)	Pin 29	LED5
PB6 (output)	Pin 30	LED6
PB7 (output)	Pin 31	LED7
PE0 (input)	Pin 56	Abort switch SW8
PE1	Pin 55	not used
PE2 (output)	Pin 54	Relay
PE3 (output)	Pin 53	Opto-coupler
PE4	Pin 39	not used
PE5	Pin 38	not used
PE6	Pin 37	not used
PE7	Pin 36	not used
PH0 (input)	Pin 52	DIP switch 1 or pushbutton switch SW5
PH1 (input)	Pin 51	DIP switch 2 or pushbutton switch SW4 (input)
PH2 (input)	Pin 50	DIP switch 3 or pushbutton switch SW3 (input)
PH3 (input)	Pin 49	DIP switch 4 or pushbutton switch SW2 (input)
PH4 (input)	Pin 35	DIP switch 5 (input)
PH5 (input)	Pin 34	DIP switch 6 (input)
PH6 (input)	Pin 33	DIP switch 7 (input)
PH7 (input)	Pin 32	DIP switch 8 (input)
PJ0 (output)	Pin 22	DIR of RS485
PJ1 (output)	Pin 21	LED enable
PJ6	Pin 99	SDA for DS1307(U11) or external I2C (J2)
PJ7	Pin 98	SCL for DS1307(U11) or external I2C (J2)
PK0 (output)	Pin 8	RS of LCD module
PK1 (output)	Pin 7	EN of LCD module
PK2	Pin 6	DB4 of LCD module (bi-directional)
PK3	Pin 5	DB5 of LCD module (bi-directional)
PK4	Pin 20	DB6 of LCD module (bi-directional)
PK5	Pin 19	DB7 of LCD module (bi-directional)
PK7 (output)	Pin 108	R/W of LCD module

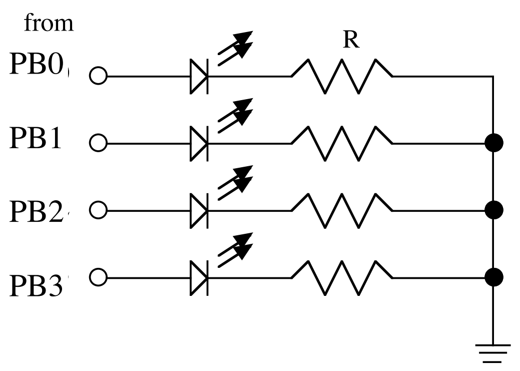
Table 1-1: I/O pin usage list 1

Pin Name	Pin #	I/O Usage
PM0	Pin 105	CAN0
PM1	Pin 104	CAN0
PM2	Pin 103	Write Enable for SD memory
PM3	Pin 102	Card detect for SD memory
PM4	Pin 101	CS of SD memory
PM5	Pin 100	not used
PM6	Pin 88	CS of LTC1661 (DAC)
PM7	Pin 87	I/O for external SPI (J10)
PP0 (output)	Pin 4	Digit 3 of 7-segment display or EN12 of H-bridge
PP1 (output)	Pin 3	Digit 2 of 7-segment display or EN34 of H-bridge
PP2 (output)	Pin 2	Digit 1 of 7-segment display
PP3 (output)	Pin 1	Digit 0 of 7-segment display
PP4 (output)	Pin 112	Servo motor 1 or RGB LED
PP5 (output)	Pin 111	Servo motor 2 or RGB LED
PP6 (output)	Pin 110	Servo motor 3 or RGB LED
PP7 (output)	Pin 109	Servo motor 4
PS0	Pin 89	SCI0 for PC communication, RECV (DB9 connector P1)
PS1	Pin 90	SCI0 for PC communication, XMIT (DB9 connector P1)
PS2	Pin 91	SCI1 for user applications, RECV, selected by J23
PS3	Pin 92	SCI1 for user applications, XMIT
PS4	Pin 93	MISO for LTC1661, SD memory interface and external SPI (J10)
PS5	Pin 94	MOSI for LTC1661, SD memory interface and external SPI (J10)
PS6	Pin 95	SCLK for LTC1661, SD memory interface and external SPI (J10)
PS7	Pin 96	I/O for external SPI (J10)
PT0 (input)	Pin 9	Rotary encoder
PT1 (input)	Pin 10	Rotary encoder
PT2	Pin 11	not used
PT3 (input)	Pin 12	IR RECV when jumpers on J27 set for PT3 and PT4
PT4 (output)	Pin 15	IR XMIT when jumpers on J27 are set for PT3 and PT4
PT5 (output)	Pin 16	Speaker (output)
PT6 (output)	Pin 17	BDMout reset (used in POD mode only)
PT7	Pin 18	BDMout data line (bi-directional, used in POD mode only)
PAD0	Pin 67	D-bug12 mode select, SW7
PAD1	Pin 69	D-bug12 mode select, SW7
PAD2	Pin 71	Alarm trigger1, analog or digital input
PAD3	Pin 73	Alarm trigger2, analog or digital input
PAD4	Pin 75	Light sensor (phototransistor Q1)
PAD5	Pin 77	Temperature sensor (U14, MCP9701A)
PAD6	Pin 79	Not Used
PAD7	Pin 81	Trimmer pot VR2
PAD8	Pin 68	X axis input for Wytec accelerometer or ADC input for GP12D2
PAD9	Pin 70	Y axis input for Wytec accelerometer or ADC input for GP12D2
PAD10	Pin 72	Z axis input for Wytec accelerometer or ADC input for GP12D2
PAD11	Pin 74	not used
PAD12	Pin 76	not used
PAD13	Pin 78	not used
PAD14	Pin 80	not used
PAD15	Pin 82	not used

Table 1-2: I/O pin usage list 2

Light Emitting Diodes

- LEDs must be forward biased and has enough current (a few to 10 mA) flowing through it in order to be lighted.
- The forward voltage across the LED is from 1.6V to about 2.3V. We use 2.0V for calculation purpose.
- If Vcc = 5V, then $5V = 2.0V + I_{Rx}R$. If I_{Rx} = 10 mA, then R = 300 ohms.



Each port B pin is connected to a LED on Dragon12-plus2. To select the LEDs, pin 1 of port J (PTJ1) needs to be set to 0, as required in Dragon12-plus2.

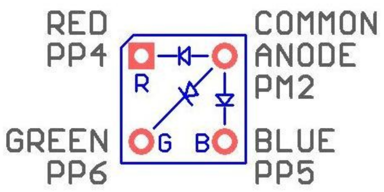
To turn an LED on and off, we need to output appropriate values to port B. A 2-layer loop is used to create time delay to control the LED blinking rate.

```
#include "reg9s12.h"              ; include this file in the directory

led0:       equ     $01           ; 00000001 => PB0=1 for LED0
led1:       equ     $02           ; 00000010 => PB1=1  for LED1
led2:       equ     $04           ; 00000100 => PB2=1  for LED2
led3:       equ     $08           ; 00001000 => PB3=1  for LED3

            org     $2000
            movb    #$0F, DDRB    ; set bits 0-3 of port B as output
            bset    DDRJ, $02     ; set port J bit 1 =1 for output (in Dragon12+)
            bclr    PTJ, $02      ; set port J bit 1 =0 to allow LEDs
            movb    #$FF, DDRP    ; set port P as output
            movb    #$0F, PTP     ; turn off 7-segement displays (in Dragon12+)


; clear all bits to turn off LED0-3 off by PortB*(1's complement of 00001111)
            bclr    PortB,led0+led1+led2+led3
            jsr     delay                   ; generate the desired delay
```

```
main        bset    PortB, led0         ; LED 0 on;
            jsr     delay               ; generate the desired delay
            bclr    portB, led0         ; LED 0 off;
            jsr     delay

            bset    PortB, led1         ; LED 1
            jsr     delay
            bclr    PortB, led1
            jsr     delay

            bset    PortB, led2         ; LED 2
            jsr     delay
            bclr    PortB, led2
            jsr     delay

            bset    PortB, led3         ; LED 3
            jsr     delay
            bclr    PortB, led3
            jsr     delay

            jmp     main                ; start over

; delay subroutine; use 2 loops to set the desired time delay
delay:      ldab    #$40                ; adjust the value to change blinking rate
delay1:     ldx     #$FFFF
delay2:     dbne    x,delay2
            dbne    b,delay1
            rts
            end
```

RGB LEDs

Bits 4~6 of port P are connected to the red, green, and blue (RGB) LEDs.



(Note: the orientations of the diodes are reversed. PM2 is common cathode, which needs to set to logic 0 to enable the LEDs.)

```
#include “reg9s12.h”                          ; include this file in the directory

red:        equ     $10                       ; 0001 0000 => PP4 = 1 for red LED
blue:       equ     $20                       ; 0010 0000 => PP5 = 1 for blue LED
green:      equ     $40                       ; 0100 0000 => PP6 = 1 for green LED

            org     $2000
            movb    #$70, DDRP                ; set bits 4-6 of port P as output
            bclr    PTP,red+blue+green        ; clear bits 4-6 of port P
            bset    DDRM, $04                 ; set port M bit 2 as output (required step)
            bclr    PTM, $04                  ; set PM2=0 to enable RGB LEDs
            jsr     delay                     ; for signals to settle down

main:       bset    PTP, red                  ; red on
            jsr     delay
            bclr    PTP, red                  ; red off
            jsr     delay

            bset    PTP, blue                 ; blue on
            jsr     delay
            bclr    PTP, blue                 ; blue off
            jsr     delay

            bset    PTP, green                ; green on
            jsr     delay
            bclr    PTP, green      ;          green off
            jsr     delay

            jmp     main                      ; start over

; delay subroutine; use 2 loops to set the desired delay
delay:      ldab    #$40                      ; adjsut the value to change blinking rate
delay1:     ldx     #$FFFF
delay2:     dbne    x, delay2
            dbne    b, delay1
            rts
            end
```

Buzzer

The buzzer is a 5V audio transducer, connected to pin 5 of port T (PT5).

```
#include “reg9s12.h”                          ; include this file in the directory
buzzer:     equ     $20                       ; 0010 0000 => PT5=1 for buzzer
```

```
                org     $2000
                bset    DDRT, buzzer            ; set PT 5 as output for buzzer
                jsr     delay

main:           bset    PTT, buzzer             ; buzzer on
                jsr     delay
                bclr    PTT, buzzer             ; buzzer off
                jsr     delay

                jmp     main                    ; start over

; delay subroutine; use 2 loops to set the desired delay
delay:          ldab    #$40                    ; adjust the value to change tone
delay1:         ldx     #$FFFF
delay2:         dbne    x, delay2
                dbne    b, delay1
                rts
                end
```

Relay

A relay is connected to pin 2 of port E (PE2) on Dragon12-plus2.

```
#include "reg9s12.h"                            ; include this file in the directory
relay:          equ     $04                     ; 0000 0100 => PE2=1 for relay

                org     $2000
                bset    DDRE, relay             ; set port E bit 2 as output for relay
                jsr     delay

main:           bset    PortE, relay            ; relay on (hear clicking sound)
                jsr     delay
                bclr    PortE, relay            ; relay off (hear clicking sound)
                jsr     delay

                jmp     main                    ; start over

; delay subroutine; use 2 loops to set the desired delay
delay:          ldab    #$40            ; adjust the value to change toggleing rate
delay1:         ldx     #$FFFF
delay2:         dbne    x, delay2
                dbne    b, delay1
                rts
                end
```

DIP Switches and Pushbuttons

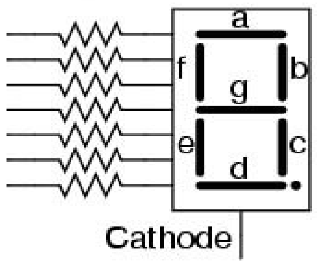
- Eight switches in a dual-inline package (DIP) are connected to port H.
- When a DIP switch is up, the associated input is 1. Otherwise, the input is 0.
- Four pushbuttons share the connected with the lower four DIP switches.
- When port H is programmed as an output port, the DIP switches and pushbuttons are ignored. (For the best result, all eight DIP switches should be at the down position.)

```
#include "reg9s12.h"                ; include this file in the directory

            org     $2000           ;
            movb    #$FF, DDRB      ; set port B as output for LEDs
            bset    DDRJ, $02       ; set port J bit 1 as output (required by Dragon12+)
            bclr    PTJ, $02        ; turn off port J bit 1 to enable LEDs
            movb    #$FF, DDRP      ; set port P as output
            movb    #$0F, PTP       ; turn off 7-segement displays (in Dragon12+)

            movb    #$00, DDRH      ; set port H as input for DIP switches

main        ldaa    PTH             ; read the status of DIP switches
            jsr     delay           ; generate the deired delay (optional)
            staa    PortB           ; output to LEDs
            jmp     main            ; start over

; delay subroutine; use 2 loops to set the desired delay
delay:      ldab    #$40                    ; adjsut the value to change the time delay
delay1:     ldx     #$FFFF
delay2:     dbne    x,delay2
            dbne    b,delay1
            rts
            end
```

Seven-Segment LED Displays

- There are four digits of seven-segment displays on Dragon12+.
- The type of these seven-segment displays is "**common cathode**." In each digit, all anodes are driven indiviually by the associated pins of an output port, while all cathodes are internally connected together.

- A segment is lighted when a high voltage is applied at the segment.

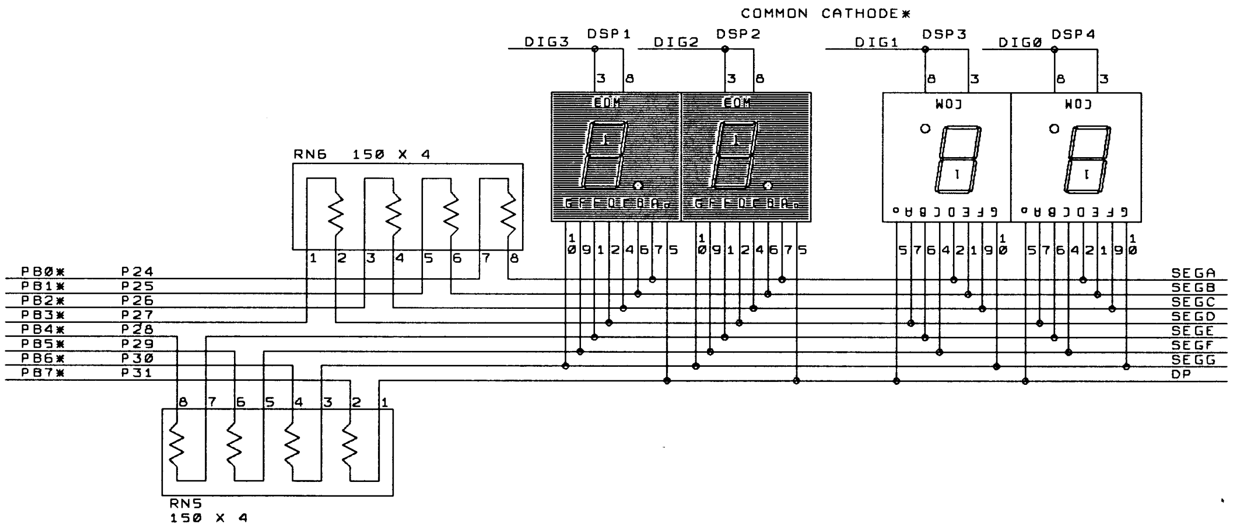


(Note: To enabe each 7-segement display, its cathode needs to set to logic 0.)

- Port B is used to drive the 7-segment anodes. (PB0 = a, PB1 = b, PB2 = c, PB3 = d, PB4 = e, PB5 = f, PB6 = g, PB7 = decimal point.)

port B	7	6	5	4	3	2	1	0	
character	DP	g	f	e	d	c	b	a	hex
0	0	0	1	1	1	1	1	1	$3F
1	0	0	0	0	0	1	1	0	$06
2	0	1	0	1	1	0	1	1	$5B
3	0	1	0	0	1	1	1	1	$4F
4	0	1	1	0	0	1	1	0	$66
5	0	1	1	0	1	1	0	1	$6D
6	0	1	1	1	1	1	0	1	$7D
7	0	0	0	0	0	1	1	1	$07
8	0	1	1	1	1	1	1	1	$7F
9	0	1	1	0	1	1	1	1	$6F
A	0	1	1	1	0	1	1	1	$77
b	0	1	1	1	1	1	0	0	$7C
C	0	0	1	1	1	0	0	1	$39
d	0	1	0	1	1	1	1	0	$5E
E	0	1	1	1	1	0	0	1	$79
F	0	1	1	1	0	0	0	1	$71
G	0	0	1	1	1	1	0	1	$3D
H	0	1	1	1	0	1	1	0	$76
h	0	1	1	1	0	1	0	0	$74
J	0	0	0	1	1	1	1	0	$1E
L	0	0	1	1	1	0	0	0	$38
n	0	1	0	1	0	1	0	0	$54
o (upper)	0	1	1	0	0	0	1	1	$63
o (lower)	0	1	0	1	1	1	0	0	$5C
P	0	1	1	1	0	0	1	1	$73
r	0	1	0	1	0	0	0	0	$50
t	0	1	1	1	1	0	0	0	$78
U	0	0	1	1	1	1	1	0	$3E
u	0	0	0	1	1	1	0	0	$1C

y	0	1	1	0	1	1	1	0	$6E
- (bottom)	0	0	0	0	1	0	0	0	$08
- (middle)	0	1	0	0	0	0	0	0	$40
blank	0	0	0	0	0	0	0	0	$00
- (top)	0	0	0	0	0	0	0	1	$01
= (lower)	0	1	0	0	1	0	0	0	$48
= (upper)	0	1	0	0	0	0	0	1	$41
=	0	0	0	0	1	0	0	1	$09
=	0	1	0	0	1	0	0	1	$49



- **Time-multiplexing** technique is often used to drive multiple displays in order to save I/O pins. Port P pins 3, 2, 1, and 0 are used to drive the common cathodes in each display in order to turn on DSP 1, 2, 3, and 4, respectively.

- Each display is turned on and off sequentially by at least 30 times within a second. Like motion picture, the persistence of vision makes us feel that all displays are turned on simultaneously.

- Setting PTP3=1, PTP2=1, PTP1=1, and PTP0=0 turns on DISP1 (leftmost digit). Setting PTP3=1, PTP2=1, PTP1=0, and PTP0=1 turns on DISP2. Setting PTP3=1, PTP2=0, PTP1=1, and PTP0=1 turns on DISP3. Setting PTP3=0, PTP2=1, PTP1=1, and PTP0=1 turns on DISP4 (rightmost digit).

```
movb    #$0E, PTP    ; turn on DISP1 (leftmost); $0E = 0000 1110
movb    #$0D, PTP    ; turn on DISP2; $0D = 0000 1101
movb    #$0B, PTP    ; turn on DISP3; $0B = 0000 1011
movb    #$07, PTP    ; turn on DISP4 (rightmost) ); $07 = 0000 0111
```

The following program displays “1234” simultaneously.

```
#include “reg9s12.h”                ; include this file in the directory

            org     $2000           ;
            movb    #$FF, DDRB      ; set port B as output for 7-segment displays
            movb    #$FF, DDRP      ; set port P as output

main:       movb    #$06, PortB     ; display “1”
            movb    #$0E, PTP       ; turn on DISP1 (leftmost) and off others
            jsr     delay           ; generate the desired delay

            movb    #$5B, PortB     ; display “2”
            movb    #$0D, PTP       ; turn on DISP2 and off others
            jsr     delay

            movb    #$4F, PortB     ; display “3”
            movb    #$0B, PTP       ; turn on DISP3 and off others
            jsr     delay

            movb    #$66, PortB     ; display “4”
            movb    #$07, PTP       ; turn on DISP4 (rightmost) and off others
            jsr     delay

            jmp     main            ; start over

; delay subroutine; use 2 loops to set the desired delay
delay:      ldab    #$01            ; adjust the value to change refresh rate
delay1:     ldx     #$FFFF
delay2:     dbne    x,delay2
            dbne    b,delay1
            rts
            end
```

Example: Display the word “HELP” on the 7-segment LED displays with refreshing time of 1 ms per digit.

```
#include "reg9s12.h"
            org     $1000
select      rmb     1
d1ms_flag   rmb     1
disp_data   rmb     4
disptn      rmb     4
```

```
segm_ptrn:     ; segment pattern for characters in the conversion table
               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     $3d,$76,$74,$1e,$38,$54,$63,$5c       ; 10-17
;                      G,  H,  h,  J   L   n   o   o
               fcb     $73,$50,$78,$3e,$1c,$6e,$08,$40       ; 18-1f
;                      P,  r,  t,  U,  u   Y   -   -
               fcb     $00,$01,$48,$41,$09,$49               ; 20-23
;                      blk, -,  =,  =,  =,  =

; Segment conversion table:
;
; Character position in table:  0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
; Characters to display:        0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
;
; Character position in table:  $10,$11,$12,$13,$14,$15,$16,$17
; Characters to display:        G    H    h    J   L    n    o    o
;
; Character position in table:  $18,$19,$1A,$1B,$1C,$1D,$1E,$1F,$20
; Characters to display:        P    r    t    U    u    y    _    --  Blank


               org     $2000
               lds     #$2000
               ldx     #0
               movb    #$FF,DDRB              ; set Port B as output
               movb    #$FF,DDRP              ; set Port P as output
               movb    #$FF,DDRJ              ; set port J as output
               movb    #$02,PTJ               ; set PJ1=1 to disable LEDs
               movb    #$0F,PTP               ; turn off 7-segment and RGB LEDs

               movb     #$80,TSCR             ; enable timer
               movb     #$40,TIOS             ; select T6 as an output compare
               movb     #$40,TMSK1
               cli

begin          ldab    #$11                   ; position of "H" in the look-up table
               stab    disp_data
               ldab    #$0E                   ; position of "E"
               stab    disp_data+1
               ldab    #$14                   ; position of "L"
               stab    disp_data+2
               ldab    #$18                   ; position of "P"
               stab    disp_data+3
```

```
                ldx     #disp_data          ; X as address pointer
                jsr     move                ; conversion subroutine
                jsr     sel_digit           ; display subroutine

wait            tst     d1ms_flag           ; wait for 1 ms interrupt
                beq     wait
                clr     d1ms_flag           ; restart TC6
                jmp     begin               ; refresh the displays forever

move            ldy     #disptn             ; convert the char to segment pattern
next            ldaa    0,X                 ;
                jsr     seven_segment       ; convert A to segment patten, bit 7= DP
                staa    0,Y
                inx
                iny
                cpy     #disptn+4
                bne     next
                rts

seven_segment:                          ; match the digit position with segment table
                pshx
                pshb
                ldx     #segm_ptrn          ; X = segment table starting address
                psha
                anda    #$3F                ; clear the rightmost 3 bits
                tab
                abx
                ldaa    0,X                 ; get segment pattern for the displaying digit
                pulb
                andb    #$80                ; add DP
                aba                         ; A contains the segment pattern
                pulb
                pulx
                rts

sel_digit       inc     select              ; multiplexing displays one digit at a time
                ldab    select
                andb    #3
                tstb
                beq     digit3
                decb
                beq     digit2
                decb
                beq     digit1
digit0          ldaa    disptn+3
                staa    PortB
                bclr    PTP,$08             ; turn on digit 0
                bset    PTP,$04             ; turn off all other digits
                bset    PTP,$02
                bset    PTP,$01
                rts
digit1          ldaa    disptn+2
                staa    PortB
                bset    PTP,$08
                bclr    PTP,$04             ; turn on digit 1
                bset    PTP,$02             ; turn off all other digits
                bset    PTP,$01
                rts
digit2          ldaa    disptn+1
                staa    PortB
                bset    PTP,$08
                bset    PTP,$04
                bclr    PTP,$02             ; turn on digit 2
                bset    PTP,$01             ; turn off all other digits
                rts
digit3          ldaa    disptn
                staa    PortB
                bset    PTP,$08
                bset    PTP,$04
                bset    PTP,$02
                bclr    PTP,$01             ; turn on digit 3
                rts

timer6          inc     d1ms_flag
                ldd     #24000              ; reload the count for 1 ms time base
                addd    TC6
                std     TC6
                movb    #$40,TFLG1   ; clear flag of TC6
                rti

                org     $3E62
                fdb     timer6

                end
```

Possible project: Display a 4-character message on the 7-segment displays at the beginning. Then, scan the keypad and display the key "value" on the 7-segment display whenever a key is pressed.

Liquid Crystal Displays (LCDs)

- LCDs have several advantages over 7-segement displays, such as high constrast, low power consumption, small footprint, displaying many characters, and ability to display graphics.

- Most common type of LCDs allows the light to pass through when activated.

- An LCD segment is activated when a low-frequency (30Hz~1KHz) bipolar signal is applied. When voltage is placed across the segment, it sets up an electrostatic field that aligns the crystals in the liquid, allowing the light to pass through the segment.

- If no voltage is applied across a segment, the crystals are randomly aligned and appear to be opaque.

- In a digital watch, the segments (dots) appear to darken when they are activated because light passes through the segment to a black cardboard backing that absorbs all light. The area surrounding the activated segment appears brighter in color because the randomly aligned crystals reflect much of the light.

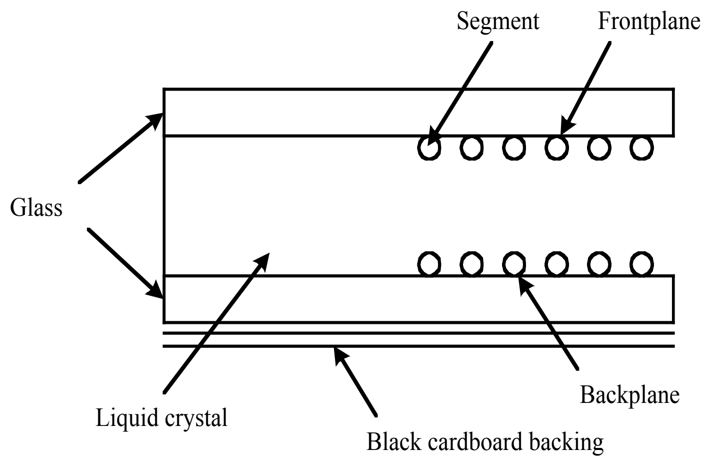


Figure 7.15 A liquid crystal display (LCD)

- In a backlit computer display, the segment (dot) appears to grow brighter because of a light placed behind the display; the light is allowed to pass through the segment when it is activated.

- In color LCD displays, three segments (dots) for each picture element (pixel) are used, and the three dots are filtered so that they can pass red, blue, and green lights. By varying the number of dots and the amount of time each dot is active, just about any color and intensity can be display. For example, white light consists of 59% green, 30% red, and 11% blue light.

HD44780U LCD Controller

- Hitachi HD44780 is the most popular LCD controller with the following features:
  - Display capability: 4 (rows) x 20 (characters).
  - Pins DB7~DB0 are used to exchange data with the MCU.
  - E is an enable input to the controller.
  - $R/\overline{W}$ input determines the direction of data transfer (R=read; W=write).
  - RS input selects the register for instruction (RS=0) or data (RS=1).
  - V_{EE} input is used to control the brightness of the display and is often connected to a variable resistor.
  - HD44780 can be configured to display 1-line, 2-line, and 4-line information.
  - Expandable the character-based LCD module with more than 80 characters

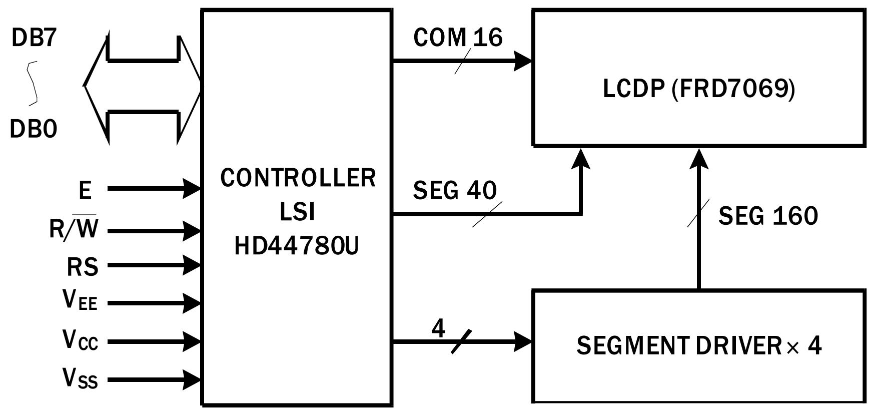


Figure 7.27 Block diagram of a HD 44780U-based LCD kit

(This table is for the instruction set on next page.)

Table 7.6 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

Table 7.5 HD44780U instruction set

Instruction	Code										Description	Execution Time
	RS	R/$\overline{W}$	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 without changing DDRAM contents Also returns display being shifted to the original position	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	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	1	S/C	R/L	*	*		Sets cursor-move or display-shift (S/C), shift direction(R/L). DDRAM contents remain unchanged	40 µs
Function set	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 and 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 operation)	0 µs
Write CGRAM or DDRAM	1	0	write data								Writes data to CGRAM or DDRAM	40 µs
Read from CGRAM or DDRAM	1	1	read data								Reads data from CGRAM or DDRAM	40 µs

- HD44780 has a Display Data RAM (DDRAM) to store data to be displayed on the LCD with the following address ranges of DDRAM for 1-line, 2-line, and 4-line.

Table 7.7a DDRAM address usage for a 1-line LCD

Display Size	Visible	
	Character Positions	DDRAM Addresses
1 * 8	00..07	0x00..0x07
1 * 16	00..15	0x00..0x0F
1 * 20	00..19	0x00..0x13
1 * 24	00..23	0x00..0x17
1 * 32	00..31	0x00..0x1F
1 * 40	00..39	0x00..0x27

Table 7.7b DDRAM address usage for a 2-line LCD

Display Size	Visible	
	Character Positions	DDRAM Addresses
2 * 16	00..15	0x00..0x0F + 0x40..0x4F
2 * 20	00..19	0x00..0x13 + 0x40..0x53
2 * 24	00..23	0x00..0x17 + 0x40..0x57
2 * 32	00..31	0x00..0x1F + 0x40..0x5F
2 * 40	00..39	0x00..0x27 + 0x40..0x67

Registers of HD44780

- The HD44780 has two 8-bit user accessible registers: instruction register (IR) and data register (DR).

- IR stores instruction and command codes, such as display clear and cursor move, and address information for DDRAM and CGRAM. The MCU writes commands into IR to set up the LCD operation parameters.

- To write data into DDRAM or CG RAM, the MCU writes into DR first. (The address of the DDRAM should first be set up with a previous instruction.)

- DR is also used for data storage when reading data from DDRAM or CGRAM.

- HD44780 has a Busy Flag (BF) that is output to the DB7 pin when RS=0 and $R/\overline{W}$ =1 (in instruction mode). BF=1 indicates that the HD44780 is still busy with an internal operation.

- HD44780 uses a 7-bit Address Counter (AC) to keep track of the address of the next DDRAM or CGRAM location to be accessed. When an instruction is written into IR, the address information contained in the instruction is transferred to AC. The content of AC is output to DB6-0 when RS=$R/\overline{W}$ =0 and DB7=1 (in instruction mode).

Table 7.8 Register selection

RS	$R/\overline{W}$	Operation
0	0	IR write as an internal operation (display clear, etc.).
0	1	Read busy flag (DB7) and address counter (DB0 to DB6).
1	0	DR write as an internal operation (DR to DDRAM or CGRAM).
1	1	DR read as an internal operation (DDRAM or CGRAM to DR).

HD44780 Instructions

Clear display

- Writes $20 (= space character) to all DDRAM locations.
- Sets 0 into the address counter (return cursor to upper left corner of the LCD).
- Sets I/D=1 (increment mode) in the entry mode.

Return Home

- Sets DDRAM address counter to 0 (return cursor to upper left corner of the LCD).
- DDRAM contents are not changed.

Entry Mode Set

- Sets I/D=1 to increment or I/D=0 to decrement the DDRAM or CGRAM address. The cursor will be moved to the right (I/D=1) or left (I/D=0).
- Controls the shifting of the display: characters shift if S=1.

Display On/Off Control

- Turns on/off display when D=1/0.
- Turns on/off cursor when C=1/0.
- Turns on/off cursor blinking when B=1/0.

Cursor or Display Shift

- Shifts the cursor position to the right or left without writing or reading display data.
- Cursor shifting is controlled by two bits.
- When the cursor gets to the end of a line, it will move to the next line.
- When the displayed data is shifted repeatedly, each line of data moves only horizontally. The second line does not shift into the first row.

Table 7.9 LCD Shift function

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

- Sets the data length to be 4 bits (using DB7-4 when DL=0) or 8 bits (using BD7-0 when DL=1).
- Selects the number of display lines to be one line (when N=0) or two lines (when N=1).
- Selects character font to be 5×8 (when F=0) or 5×10 (when F=1).

Set CGRAM Address

- Contains the CGRAM address to be written into the address counter.

Set DDRAM Address

- Set the starting DDRAM address in the address counter to display information in a certain position on the LCD screen.

Read Busy Flag and Address

- User can use the busy flag (BF) to determine whether the LCD controller is ready to accept another command.
- User can use the address counter (AC) to control where to start displaying data.

Setting Up HD44780U LCD Controller

- Can treat the LCD as an I/O device that uses an I/O port for data transfer and several other I/O pins as control signals.

- The interface can be 4 bits (DB7-4) or 8 bits (BD7-0). When in the 4-bit mode, the upper 4 bits of the data are sent over DB7-4 first and followed immediately by the lower 4 bits (also to DB7-4).

- Write commands to the instruction register in order to set up the LCD controller before it can be used.

- To read or write the LCD successfully, one must satisfy the timing requirements of the LCD. An LCD command takes much longer timer to execute than a normal HCS12 instruction. No new command should be sent to the LCD before the current command completes.

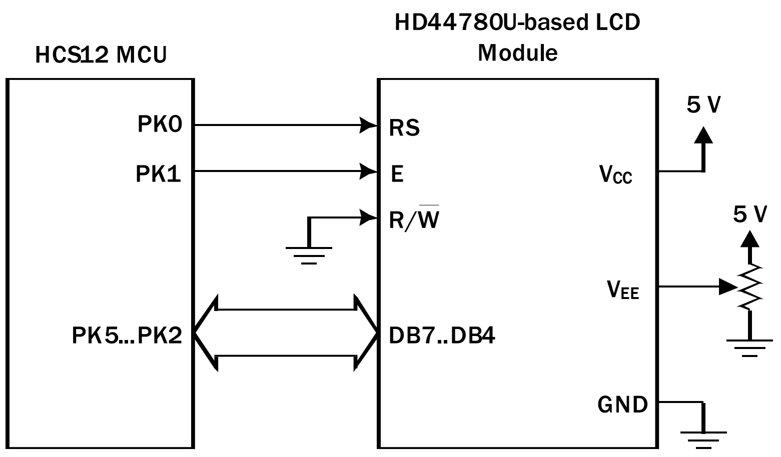


Figure 7.28 LCD interface example (4-bit bus, used in Dragon 12)

Procedure to send a command to the LCD instruction register

Step 1: Pull RS to low (for instruction mode) and E to low (to disable LCD module).
Step 2: Pull $R/\overline{W}$ to low (to write). (It is already grounded in Dragon12-plus2)
Step 3: Pull E to high (to enable the LCD module).
Step 4: Output instruction to the output port attached to the LCD data bus. (One needs to configure the I/O port for output before writing data to the LCD module.)
Step 5: Pull E to low and make sure that the internal operation has time to complete.

Procedure to write a byte to the LCD data register

Step 1: Pull RS to high (for data mode) and E to low (to disable LCD controller).
Step 2: Pull $R/\overline{W}$ to low (to write). (already grounded in Dragon12-plus2)
Step 3: Pull E to high (to enable the LCD module).
Step 4: Output data to the output port attached to the LCD data bus.
Step 5: Pull E to low and make sure that the internal operation has time to complete.

(These five steps need to be repeated again once for an LCD kit with 4-bit mode.)

Sending a command to the LCD controller

Most LCD commands are completed in 40 μs. If the function waits for 40 μs after performing the specified operation, then most commands will be completed when the function returns.

The following assembly subroutine is for the 8-bit instruction.

```
lcd_dat     equ    PortK          ; LCD data port (use PK5~2)
lcd_dir     equ    DDRK           ; LCD data direction register
lcd_E       equ    $02            ; E signal pin for PTK1
lcd_RS      equ    $01            ; RS signal pin for PTK0

; the command is contained in A when calling this subroutine from main program
cmd2LCD     psha                          ; save the command in stack
            bclr   lcd_dat, lcd_RS        ; set RS=0 for IR => PTK0=0
            bset   lcd_dat, lcd_E         ; set E=1 => PTK=1
            anda   #$F0                   ; clear the lower 4 bits of the command
            lsra                          ; shift the upper 4 bits to PTK5-2 to the
            lsra                          ; LCD data pins
            oraa   #$02                   ; maintain RS=0 & E=1 after LSRA
            staa   lcd_dat                ; send the content of PTK to IR
            nop                           ; delay for signal stability
            nop                           ;
            nop                             ;
            bclr    lcd_dat,lcd_E           ; set E=0 to complete the transfer

            pula                            ; retrieve the LCD command from stack
            bset    lcd_dat, lcd_E          ; set E=1 => PTK=1
            anda    #$0F                    ; clear the upper four bits of the command
            lsla                            ; shift the lower 4 bits to PTK5-2 to the
            lsla                            ; LCD data pins
            oraa    #$02                    ; maintain E=1 to PTK1 after LSLA
            staa    lcd_dat                 ; send the content of PTK to IR
            nop                             ; delay for signal stability
            nop                             ;
            nop                             ;
            bclr    lcd_dat,lcd_E           ; set E=0 to complete the transfer

            ldy     #1                      ; adding this delay will complete the internal
            jsr     delay50us               ; operation for most instructions
            rts
```

Configurating the LCD before displaying characters

```
openLCD     movb    #$FF,lcd_dir    ; configure Port K for output
            ldy     #2              ; wait for LCD to be ready
            jsr     delay100ms      ;        "
            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 instruction)
            jsr     cmd2lcd         ;       "
            ldaa    #$01            ; clear display screen and return to home position
            jsr     cmd2lcd         ;       "
            ldy     #2              ; wait until clear display command is complete
            jsr     delay1ms        ;       "
            rts
```

Outputing a character to the LCD

; The character to be output is in accumulator A.

```
putcLCD     psha                            ; save a copy of the chasracter
            bset    lcd_dat,lcd_RS          ; set RS=1 for data register => PK0=1
            bset    lcd_dat,lcd_E           ; set E=1 => PTK=1
            anda    #$F0                    ; clear the lower 4 bits of the character
```

```
        lsra                    ; shift the upper 4 bits to PTK5-2 to the
        lsra                    ; LCD data pins
        oraa    #$03            ; maintain RS=1 & E=1 after LSRA
        staa    lcd_dat         ; send the content of PTK to DR
        nop                     ; delay for signal stability
        nop                     ;
        nop                     ;
        bclr    lcd_dat,lcd_E   ; set E=0 to complete the transfer

        pula                    ; retrieve the character from the stack
        anda    #$0F            ; clear the upper 4 bits of the character
        lsla                    ; shift the lower 4 bits to PTK5-2 to the
        lsla                    ; LCD data pins
        bset    lcd_dat,lcd_E   ; set E=1 => PTK=1
        oraa    #$03            ; maintain RS=1 & E=1 after LSLA
        staa    lcd_dat         ; send the content of PTK to DR
        nop                     ; delay for signal stability
        nop                     ;
        nop                     ;
        bclr    lcd_dat,lcd_E   ; set E=0 to complete the transfer

        ldy     #1              ; wait until the write operation is complete
        jsr     delay50us       ;
        rts
```

Output a string terminated by a NULL character

; The string to be output is pointed to by index register X (= address pointer).

```
putsLCD ldaa    1,X+        ; get one character from the string
        beq     donePS      ; reach NULL character?
        jsr     putcLCD
        bra     putsLCD
donePS  rts
```

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

hello world!
I am ready!

```
#include "reg9s12.h"
lcd_dat equ     PortK       ; LCD data pins (PK5~PK2)
lcd_dir equ     DDRK        ; LCD data direction port
```

```
lcd_E        equ    $02              ; E signal pin
lcd_RS       equ    $01              ; RS signal pin

             org    $2000
             lds    #$2000           ; set up stack pointer
             jsr    openLCD          ; initialize the LCD
             ldx    #msg1            ; point to the first line of message
             jsr    putsLCD          ; display in the LCD screen
             ldaa   #$C0             ; move to the 2nd row (Tabl 7.5: $C0 = 1100 0000
             jsr    cmd2LCD          ;  = DDRAM address $40)
             ldx    #msg2            ; point to the second line of message
             jsr    putsLCD
             swi

msg1         dc.b   "hello world!",0
msg2         dc.b   "LCD is working!",0


delay1ms     movb   #$90,TSCR                ; enable TCNT & fast flags clear
             movb   #$06,TMSK2               ; configure prescale factor to 64
             bset   TIOS,$01                 ; enable OC0
             ldd    TCNT
again0       addd   #375                     ; start an output compare operation
             std    TC0                      ; with 1 ms time delay
wait_lp0     brclr  TFLG1,$01,wait_lp0
             ldd    TC0
             dbne   y,again0
             rts

delay50us    movb   #$90,TSCR                ; enable TCNT & fast flags clear
             movb   #$06,TMSK2               ; configure prescale factor to 64
             bset   TIOS,$01                 ; enable OC0
             ldd    TCNT
again1       addd   #15                      ; start an output compare operation
             std    TC0                      ; with 50 us time delay
wait_lp1     brclr  TFLG1,$01,wait_lp1
             ldd    TC0
             dbne   y,again1
             rts

delay100ms   movb   #$90,TSCR                ; enable TCNT & fast flags clear
             movb   #$06,TMSK2               ; configure prescale factor to 64
             bset   TIOS,$01                 ; enable OC0
             ldd    TCNT
again2       addd   #37500                   ; start an output compare operation
             std    TC0                      ; with 100 ms time delay
```

```
wait_lp2      brclr   TFLG1,$01,wait_lp2
              ldd     TC0
              dbne    y,again2
              rts

              end
```

Example: Display YES on the first row of the LCD displays.

```
#include "reg9s12.h"

lcd_data      equ     portK           ; LCD data register
lcd_ctrl      equ     portK           ; LCD instruction register
RS            equ     %00000001       ; mask for RS input
EN            equ     %00000010       ; mask for EN input

d1            equ     $1001           ; for delay loop
d2            equ     $1002           ; for delay loop
d3            equ     $1003           ; for delay loop
temp          equ     $1200

              org     $2000           ;
              lds     #$2000          ; define stack location
              movb    #$FF, DDRK      ; port K as output

;             ldaa    #$33            ; 8 bit, 1 line, 5x8 dots
;             jsr     comwrt4         ; output the setting to IR
;             jsr     delay
;             ldaa    #$32            ; 8 bit, 1 line, 5x8 dots
;             jsr     comwrt4         ;
;             jsr     delay

              ldaa    #$28            ; 4 bit, 2 lines, 5x8 dots
              jsr     comwrt4         ; output the command
              jsr     delay
              ldaa    #$0E            ; display on; cursor on; no blink
              jsr     comwrt4
              jsr     delay
              ldaa    #$01            ; clear display; set cursor to 0 position
              jsr     comwrt4
              jsr     delay
              ldaa    #$06            ; auto increment address counter; no shift
              jsr     comwrt4
              jsr     delay
```

```
                ldaa    #$80            ; DDRAM address = 000 0000
                jsr     comwrt4
                jsr     delay

                ldaa    #'Y'            ; character to display
                jsr     datwrt4
                jsr     delay
                ldaa    #'E'            ; character to display
                jsr     datwrt4
                jsr     delay
                ldaa     #'S'           ; character to display
                jsr     datwrt4
                jsr     delay
forever         bra     forever

comwrt4         staa    temp            ; shift upper 4 bits right to PK5-2
                anda    #$F0
                lsra
                lsra
                staa    lcd_data        ; output to LCD DB7-4
                bclr    lcd_ctrl, RS    ; select instruction register
                bset    lcd_ctrl, EN    ; transfer to instruction register
                nop
                nop
                nop
                bclr    lcd_ctrl, EN
                ldaa    temp            ; shift lower 4 bits left to PK5-2
                anda    #$0F
                lsla
                lsla
                staa    lcd_data        ; output to LCD DB7-4
                bclr    lcd_ctrl, RS    ; select instruction register
                bset    lcd_ctrl, EN    ; transfer to instruction register
                nop
                nop
                nop
                bclr    lcd_ctrl, EN
                rts

datwrt4         staa    temp            ; shift upper 4 bits right to PK5-2
                anda    #$F0
                lsra
                lsra
                staa    lcd_data        ; output to LCD DB7-4
                bset    lcd_ctrl, RS    ; select data register
                bset    lcd_ctrl, EN    ; transfer to data register
                nop
                nop
                nop
                bclr    lcd_crtl, EN

                ldaa    temp            ; shift lower 4 bits left to PK5-2
                anda    #$0F
                lsla
                lsla
                staa    lcd_data        ; output to LCD DB7-4
                bset    lcd_ctrl, RS    ; select data register
                bset    lcd_ctrl, EN    ; transfer to data register
                nop
                nop
                nop
                bclr    lcd_ctrl, EN
                rts

delay           psha                    ; save A on stack
                ldaa    #1
                staa    d3
; E-Clock = 24MHz; (1/24MHz) x 10E x 240 x 100 = 1 msec delay (approx).
loop3           ldaa    #100
                staa    d2
loop2           ldaa    #240
                staa    d1
loop1           nop                     ; loop 1 gives 10 E-cycle
                nop                     ; 1 E-cycle
                nop
                dec     d1              ; 4 E-cycles
                bne     loop1           ; 3 E-cycles
                dec     d2              ;
                bne     loop2
                dec     d3
                bne     loop3
                pula
                rts

                end
```

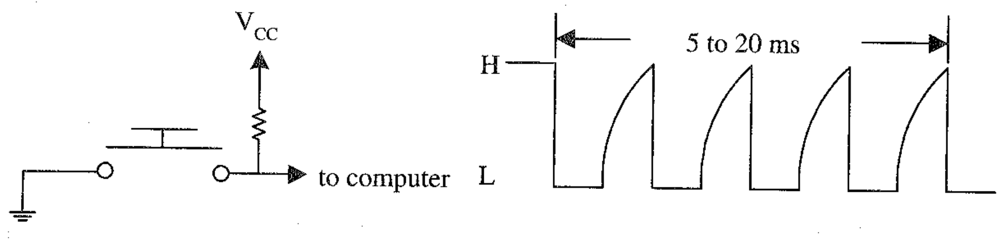
Possible Project: Show a 2-line (long) message on the LCD displays and allow shifting.

Keypad

- Arranged as an array of switches, which can be mechanical, membrane, capacitors, or Hall-effect in construction.

- Mechanical switches are most popular for keypad and keyboard due to lower cost and strength of construction. Two metal contacts are brought together to complete an electric circuit.

- Mechanical switches have a problem called ***contact bounce***. Closing a mechanical switch generates a series of pulses because the switch contacts do not come to rest immediately.



- Human cannot type more than 50 keys in a second (or 20 ms per key stoke). Reading the keyboard more than 50 times a second will read the same keystroke (caused by bouncing pulses) too many times. A debouncing mechanism (in software or hardware) is needed.

- A keyboard input is divided into three steps:
  1. Scan the keypad to discover which key has been pressed.
  2. Debounce the keypad to determine if a key is indeed pressed.
  3. Lookup the ASCII table to find out the ASCII code of the pressed key.

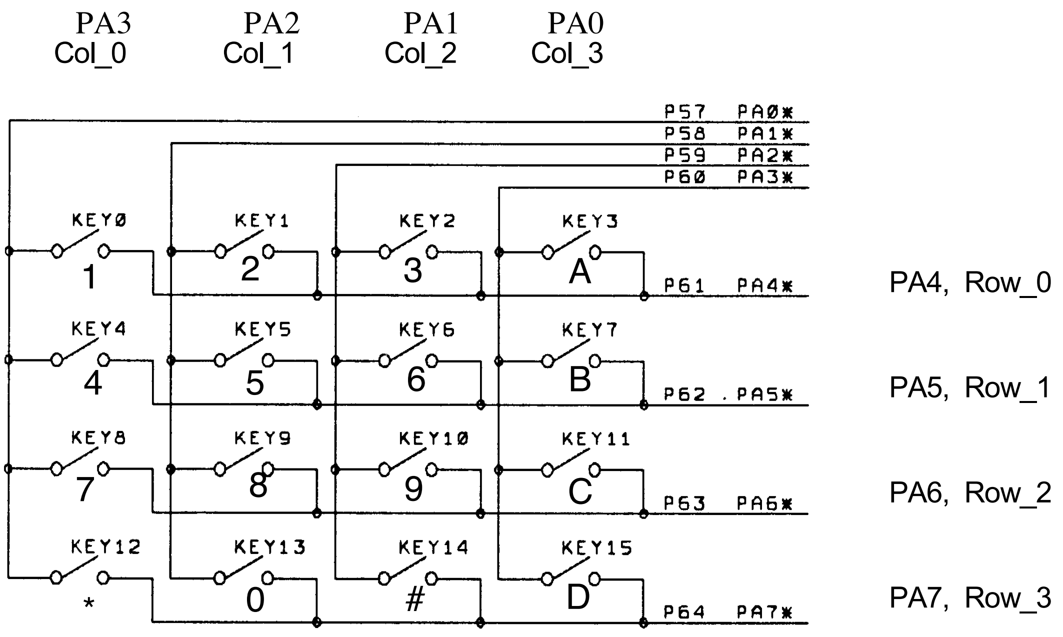
Keypad Scanning Techniques

- A keypad consisting of 12 to 24 keys is adequate for many applications.

- Arranged as a matrix switches that uses two decoding and selecting devices to determine which key was pressed by coincident decoding of the row and column of the key.

- A 16-key keypad can be easily interfaced to one parallel port, such as port A.

- A scanning routine is invoked to search for the pressed key, and then the debouncing routine is executed to verify that the key is closed.



- PA7~4 are configured for output. PA3~0 are configured for input.
- Whenever a key is pressed, the corresponding row and column are shorted together.
- To distinguish the row being scanned from those not being scanned, the row being scanned is driven high, whereas the other rows are driven low.

scan patterns				pressed key when one of PA3~0 = 1			
PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
0	0	0	1	1	2	3	A
0	0	1	0	4	5	6	B
0	1	0	0	7	8	9	C
1	0	0	0	*	0	#	D

Keyboard Debouncing

- Contact bounce is due to the dynamics of a closing contact. Signal falls and rises a few times within a period of about 5 ms as a contact bounces.

- A human being cannot press and release a switch in less 20 ms. Thus, a debouncer is designed to recognize that the switch is closed after the voltage is low for about 10 ms and that the switch is open after the voltage is high for another 10 ms.

Hardware Debouncing Techniques

1. SR latches.
2. Non-inverting CMOS gates with high impedance.
3. Integrating RC-circuit debouncer.

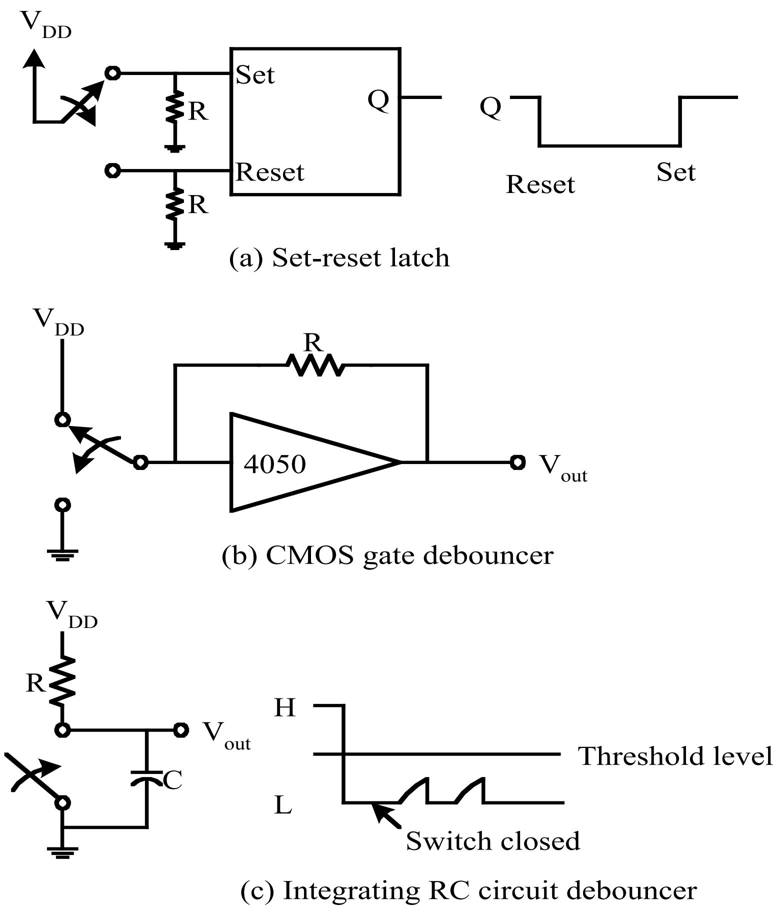


Figure 7.22 Hardware debouncing techniques

Software Debouncing Technique

The most popular and simple one has been the **wait-and-see** method. Routine simply waits for about 10 ms and re-examine the same key again to see if it is still pressed.

Example: Press a key and the ASCII code of the key is shown on port B LEDs.

```
#include "reg9s12.h"

d15mh       equ     17
d15ml       equ     250
cols        equ     PortA
rows        equ     PortA
led         equ     PortB
row0        equ     %00010000           ; mask for scan patterns for the 4 rows
row1        equ     %00100000
row2        equ     %01000000
row3        equ     %10000000
colm        equ     %00001111           ; mask for PA3-0 (the 4 columns)
rowm        equ     %11110000           ; mask for PA7-4 (the 4 rows)

            org     $1000               ; keypad look-up table
kcode0      fcb     $31,$32,$33,$41     ; "123A"
kcode1      fcb     $34,$35,$36,$42     ; "456B"
kcode2      fcb     $37,$38,$39,$43     ; "789C"
kcode3      fcb     $2A,$30,$23,$44     ; "*0#D"

            org     $1100               ; delay variables
dr15mh      rmb     1                   ; dr15mh & dr15ml for two
dr15ml      rmb     1                   ; loops for a total of 15ms delay
pdelay      rmb     1

            org     $2000               ;
            lds     #$2000              ; stack
            movb    #$FF, DDRB          ; set port B as output for LEDs
            movb    #$02, DDRJ          ; enable LEDs
            movb    #$00, PTJ           ;
            movb    #$0F, DDRP          ;
            movb    #$0F, PTP           ; turn off 7-segment displays
            movb    #$00, PortB         ; turn off LEDs

            movb    #$F0, DDRA          ; set PA7-4 as output & PA3-0 as input

check1:     ; make sure that no button is pressed in the beginning of the program
            ldaa    #rowm               ; set PA7-4 =1111 for the 4 rows
            staa    rows                ; check if PA3-0 = 0000 for the 4 columns
            ldaa    cols                ; read port A
            anda    #colm               ; get the values of PA3-0 (the 4 columns)
            cmpa    #$00                ; if PA3-0 = 0, → no button is pressed
            bne     check1              ; don't move on until NO button is pressed
```

```
check2:     ; wait for a button to be pressed
            jsr     delay           ; call for 15ms delay
            ldaa    cols            ; read port A
            anda    #colm           ; get the values of PA3-0 (the 4 columns)
            cmpa    #$00            ; if PA3-0 ≠ 0, a button is pressed
            bne     debounce        ; then jump to debounce routine
            bra     check2          ; keep checking until a button is pressed

debounce:   jsr     delay           ; debounce delay
            ldaa    cols            ; read port A
            anda    #colm           ; get the value of PA3-0 (the 4 columns)
            cmpa    #$00            ; check if button is pressed after debounce
            bne     scan            ; if pressed, determine which row
            bra     check2          ; if not pressed, wait for a button again

scan:       ldaa    #row0           ; set PA4 (row of 123A) =1, PA7-5=0
            staa    rows            ; output to port A
            movb    #$08, pdelay    ; short delay for stability
p1:         dec     pdelay          ;
            bne     p1              ;
            ldaa    cols            ; read port A
            anda    #colm           ; get the values of PA3-0 (the 4 columns)
            cmpa    #$00            ; if not 0, the presesed button is in row0
            bne     r0              ; determine which column of row0

            ldaa    #row1           ; if = 0, set PA5 (row of 456B) =1,
            staa    rows            ; PA7,6,4=0
            movb    #$08, pdelay    ;
p2:         dec     pdelay          ;
            bne     p2              ;
            ldaa    cols            ; read port A
            anda    #colm           ; get the values of PA3-0 (the 4 columns)
            cmpa    #$00            ; if not 0, the presesed button is in row1
            bne     r1              ; determine which column of row1

            ldaa    #row2           ; if = 0, set PA6 (row of 789C) =1,
            staa    rows            ; PA7,5,4=0
            movb    #$08, pdelay    ;
p3:         dec     pdelay          ;
            bne     p3              ;
            ldaa    cols            ; read port A
            anda    #colm           ; get the values of PA3-0 (the 4 columns)
            cmpa    #$00            ; if not 0, the presesed button is in row2
            bne     r2              ; determine which column of row2

            ldaa    #row3           ; if = 0, set PA7 (row of *0#D) =1,
            staa   rows              ; PA6-4=0
            movb   #$08, pdelay      ;
p4:         dec    pdelay            ;
            bne    p4                ;
            ldaa   cols              ; read port A
            anda   #colm             ; get the values of PA3-0 (the 4 columns)
            cmpa   #$00              ; if not 0, the presesed button is in row3
            bne    r3                ; determine which column of row3

            bra    check2            ; if no row is found, wait for a button again

r0:         ldx    #kcode0           ; load adress of array of row "123A"
            bra    find              ; jump to find ASCII code of pressed key
r1:         ldx    #kcode1           ; load adress of array row "456B"
            bra    find              ; jump to find ASCII code of pressed key
r2:         ldx    #kcode2           ; load adress of array row "789C"
            bra    find              ; jump to find ASCII code of pressed key
r3:         ldx    #kcode3           ; load adress of array row "*0#D"
            bra    find              ; jump to find ASCII code of pressed key

find:       ; identify which column
            anda   #colm             ; read PA3-0 to check which column is high
            coma                     ; invert the content of PA3-0
shift:      lsra                     ; shift the righmost bit of PA to C of CCR
            bcc    match             ; if C = 0, the column is found and jump to
            inx                      ; match the ASCII code
            bra    shift             ; if C=1, continue to shift column until C=0

match:      ; assign ASCII code for the pressed key
            ldaa   0,X               ; load ASCII code row array
            staa   led               ; ouptut ASCII code to port B for LEDs
            lbra   check1            ; wait for next button to be pressed

delay:      ldaa   #d15mh            ; load delay count of the outer loop
            staa   dr15mh            ;
d2:         ldaa   #d15ml            ; load sdelay count of the inner loop
            staa   dr15ml            ;
d1:         dec    dr15ml            ;
            nop
            nop
            bne    d1                ;
            dec    dr15mh            ;
            bne    d2                ;
            rts         ;

            end
```

Digital-to-Analog Converter (DAC)

- A DAC converts a binary number (digial code) into an (analog) electric voltage or current. The output of a DAC is a linear function of the input.

- Application examples: digital gain and offset adjustment, programmable voltage and current sources, programmable attenuators, digital audio processing, robotics.

- Most MCUs need to use an off-chip DAC.

Factors to Consider a DAC

- Resolution. This is the number of possible output levels, which is equal to the 2^n with n as the number of bits in use. An 8-bit DAC can represent up to 256 levels.

- Dynamic range of the output. This is a measurement of the difference between the largest and smallest signals the DAC can reproduce. Usually in decibel (dB).

- Number of channels. A DAC may have multiple output channels to satisfy the needs of the application that require more than one channel.

- Type of output: voltage or current.

- Monotonicity. This refers to the ability of the DAC's analog output to increase linearly with digital code.

AD7302 DAC

- A dual-channel 8-bit DAC (two DACs on the same chip) from Analog Devices that has a parallel interface with the MCU.

- Converts an 8-bit digital value into an analog voltage.

- Set $\overline{\text{CS}} = 0$ for the DAC to work.

- On the rising edge of $\overline{\text{WR}}$, the values on D7-D0 are latched into the input register.

- When $\overline{\text{LDAC}} = 0$, the data in the input register will be transferred to the DAC register and a new D/A conversion is started.

- AD7302 needs a reference voltage to operate. The reference voltage can be external one (from the REFIN pin) or the internal V_{DD}.

- $\overline{A}/B$ selects the channel (A or B) to perform the D/A operation.

- Each conversion takes about 2 μs to complete.

- PD puts the AD7301 in the power-down mode, reduces power consumption to 1 μW.

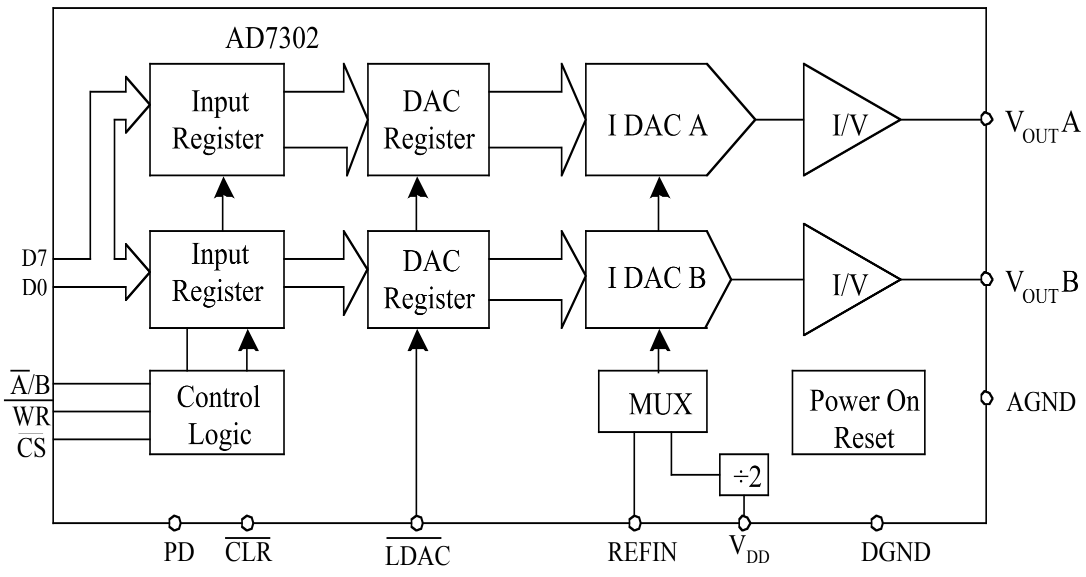


Figure 7.24 Functional block diagram of the AD7302

The output voltage ($V_{OUT}A$ or $V_{OUT}B$) from either DAC is given by

$$V_{OUT}A \text{ or } V_{OUT}B = \mathbf{2 \times V_{REF} \times (N/256)}$$

- N is the digital value (ranging from 0 to 255) to be converted.
- V_{REF} is the voltage applied to the external REFIN pin or $V_{DD}/2$ when the internal reference is selected.

Using the AD7302 to Generate a Waveform

- Both the $\overline{CS}$ and $\overline{LDAC}$ can be tied to ground permanently.

- Configure PortB7~0 and PJ1~0 as output.

- Output the digital value (from 0 to 255) to be converted to the AD7302 via a parallel port (connect to pins D7~D0), and repeat.

- For each value, pull PJ0 to low and then high so that the value on pins PortB7~0 can be transferred to the AD7302. Pull PJ1 to low to select A output during the process.

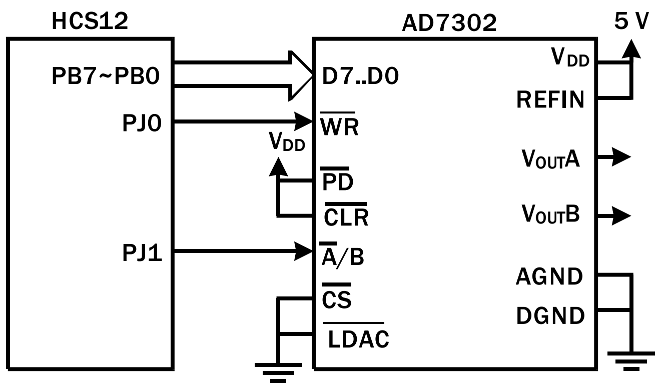


Figure 7.36 Circuit connection between the AD 7302 and the HCS12

Example: Generate a sawtooth waveform from V_{OUT}A pin.

```
#include "reg9s12.h"                    ; include this file in the directory

                org     $2000           ;
                movb    #$FF, DDRB      ; set port B as output
                bset    DDRJ, $03       ; set port J bits 1 and 0 as output
                bclr    PTJ, $02        ; select VoutA as output
loop:           inc     PortB           ; increase the output voltage by 1 step
                bclr    PTJ, $01        ; generate a rising edge on PJ0
                bset    PTJ, $01
                bset    PTJ, $01        ; add 9 more bset to provide 2 us delay
                bset    PTJ, $01        ; for D/A conversion to complete
                bset    PTJ, $01
                bset    PTJ, $01
                bset    PTJ, $01
                bset    PTJ, $01
                bset    PTJ, $01
                bset    PTJ, $01
                bset    PTJ, $01
                bra     loop
                end
```

End of Chapter 7