- **Dragon12 LCD Displays**
- Hantronix_CLD.PDF data sheet (Dragon12 CD-ROM)
  - Using the Dragon12 LCD display

Dragon12 LCD Display

• The Dragon12 board has a 16 character x 2 line display

• Each character is a 5x7 bit matrix

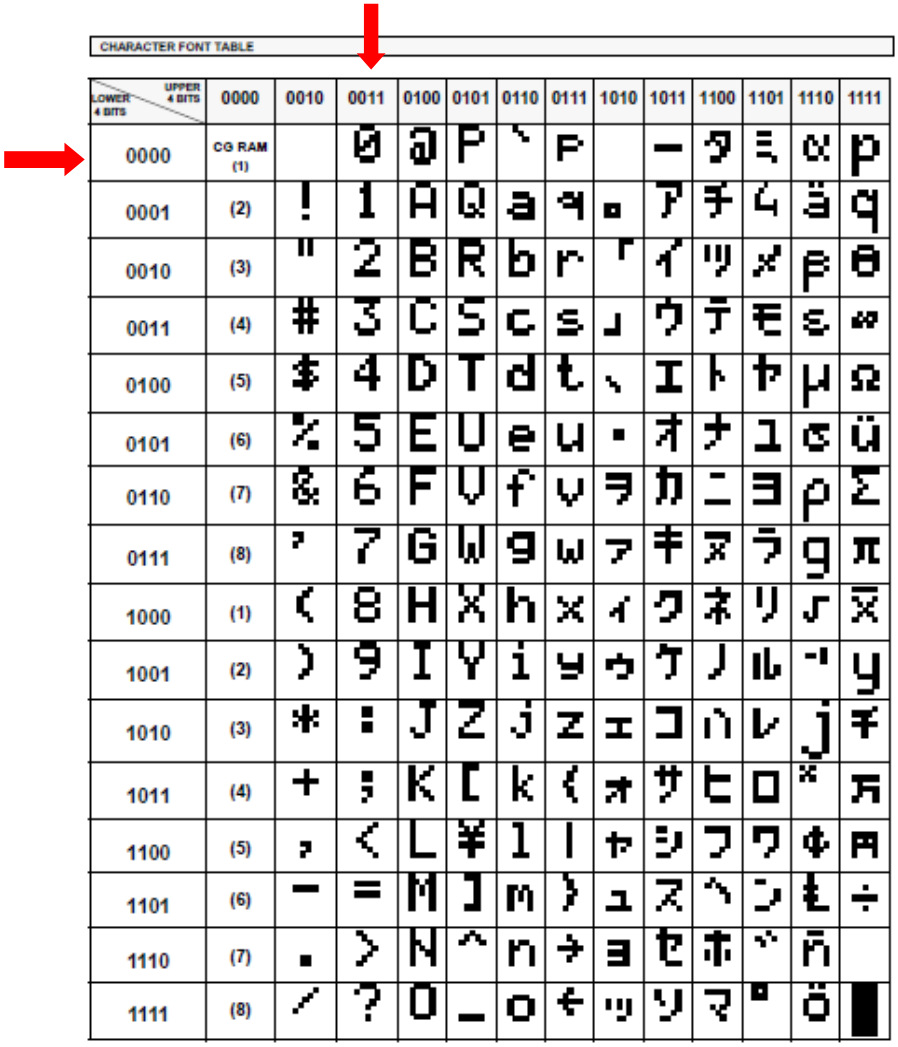
• A controller chip (Hitachi HD44780) converts ASCII characters to 5x7 bit image

• The controller chip is connected to **Port K** of the MC9S12

- Bit 0 of Port K (PK0) selects command (0) or data (1)
- Bit 1 of Port K (PK1) enables the data transfer
- Bits 5 through 2 Port K (PK5-2) contain the data
- Bit 7 of Port K (PK7) can be used to select read or write. The LCD on the Dragon12 board is set up for write only; you need to cut a trace to be able to read from the LCD.

• Use of the display is discussed in the **Hatronix_LCD2.pdf** datasheet which is on the CD-ROM which came with the Dragon12 board.

Port K	Signal	LCD pin	Pin function
0	RS	4	Reg select: 0=instruction, 1=data
1	Transfer	6	Enable data transfer
2	DB4	11	Data bus
3	DB5	12	Data bus
4	DB6	13	Data bus
5	DB7	14	Data bus
6	-	-	Not used
7	Write	-	Select read/write



• You can send commands or data to the controller chip to control the LCD display.

• These commands and data are detailed in the Hatronix_LCD2.pdf datasheet.

• The commands are:

- Clear display and return cursor to home (upper left)
- Cursor home (don't clear display)
- Entry mode (move cursor left or write)
- Display on/off – turns display on or off, cursor on or off, cursor blink
- Cursor/display shift – move cursor or shift display, which direction
- Function set – bus data width (8 or 4), number of display lines (1 or 2), font size 6x8 or 5x7)
- Set CG RAM address – set address of CG (Character Generation) RAM to generate your own characters
- Set DD RAM Address – set address of DD (Data Display) RAM to display characters
- Read busy flag and address (we can't use)
- Write data to DD RAM or CG RAM
- Read data from DD RAM or CG RAM (we can't use)

Dragon12 LCD Display

• The LCD display can use either 8-bit or 4-bit data bus. The Dragon12 board uses a **4-bit bus**, **so it takes two transfers to send one command**

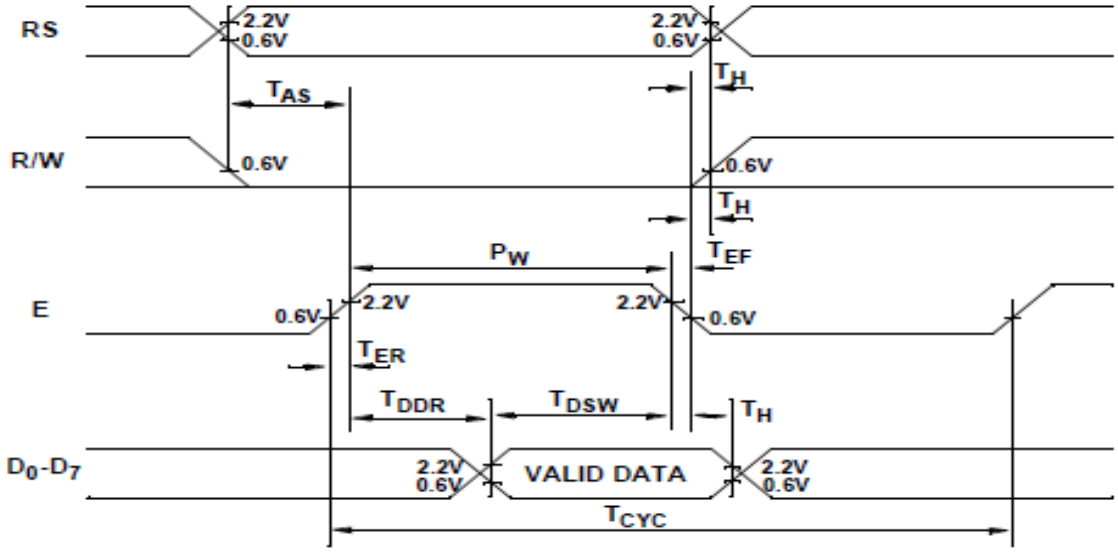
• The Dragon12 board is set so that you cannot read from the display; you can only write to it.

• When you write a command, you need to wait until the command has been executed by the LCD controller. The Busy Flag (from Read Busy Flag command) tells when the command is done. **We cannot read Busy Flag, so we have to wait specified time before proceeding.**

• To write to the controller, we need to:

1. Set RS (PK0) to 0 for command, 1 for data
2. Set R/$\hat{W}$ (On Dragon12, R/$\hat{W}$ tied low for write only)
3. Put 4 MSB on Port K bits 5-2
4. Bring E (PK1) high for at least 230 ns
5. Bring E (PK1) low
6. Put 4 LSB on Port K bits 5-2
7. Bring E (PK1) high for at least 230 ns
8. Bring E (PK1) low
9. Wait specified amount of time for execution to complete

DATA WRITE



TIMING CHARACTERISTICS

ITEM	SYMBOL	MAX.	MIN.	UNIT
ENABLE CYCLE TIME	T_{CYC}		500	nS
ENABLE PULSE WIDTH	P_W		230	nS
ENABLE RISE/FALL TIME	T_{ER}, T_{EF}	20		nS
RS, R/W SET UP TIME	T_{AS}		40	nS
DATA DELAY TIME	T_{DDR}	360		nS
DATA SETUP TIME	T_{DSW}		60	nS
HOLD TIME	T_H		10	nS

Dragon12 LCD Display

• To use LCD display

1. Give command **0x28**: Tell controller our display uses 4-bit data, 2-line display, 5x7 font

2. Give command **0x0F**: Turn on display, use cursor, blink cursor

3. Give command **0x06**: Move cursor to right after writing a character

4. Give command **0x01**: Clear screen, move cursor to home (upper left character)

5. Wait for at least **1.64 ms**

• After display is set up, you can write characters to display

Handling LCD's & LCD Modules

• Do not touch display are with bare hands
• Do not touch exposed polarizer with hard objects
• Do not expose the CMOS IC's to static electricity
• Avoid exposing the module to excessive shock or pressure
• Do not allow the storage temperature to exceed the specified range

File lcd.h:

```
#define LCD_DAT PORTK      /*Port K drives LCD data pins, E, and RS */
#define LCD_DIR DDRK       /*Direction of LCD port */
#define LCD_E 0x02         /*LCD E signal */
#define LCD_RS 0x01        /*LCD Register Select signal */

#define CMD 0              /*Command type for put2lcd */
#define DATA 1             /*Data type for put2lcd */
```

/* Prototypes for functions in lcd.c */

```
void openlcd(void);                /* Initialize LCD display */
void put2lcd(char c, char type); /*Write command  or data to LCD  */
void puts2lcd (char *ptr);         /* Write a string to the LCD display */
void delay_50us(int n);            /* Delay n 50 microsecond intervals */
void delay_1ms(int n)      ;       /* Delay n 1 millisecond intervals */
```

File lcd.c:

```
#include "derivative.h"
#include "lcd.h"

void openlcd(void)
{
        LCD_DIR = 0xFF;         /* configure LCD_DAT port for output */
        delay_1ms(100);         /* Wait for LCD to be ready */
        put2lcd(0x28,CMD);      /* set 4-bit data, 2-line display, 5x7 font */
        put2lcd(0x0F,CMD);      /* turn on display, cursor, blinking */
        put2lcd(0x06,CMD);      /* move cursor right */
        put2lcd(0x01,CMD)       /* clear screen, move cursor to home */
        delay_1ms(2);           /* wait until "clear display" command */
                                /* complete */
}

void puts2lcd (char *ptr)
{
        while (*ptr) {                  /* While character to send */
                put2lcd(*ptr,DATA);     /* Write data to LCD */
                delay_50us(1);          /* Wait for data to be written */
                ptr++;                  /* Go to next character */
        }
}

void put2lcd(char c, char type)
{
        char c_lo, c_hi;

        c_hi = (c & 0xF0) >> 2;          /* Upper 4 bits of c */
        c_lo = (c & 0x0F) << 2;          /* Lower 4 bits of c */
        if (type == DATA) LCD_DAT |= LCD_RS; /* select LCD data */
                                                /*register */
```

```
        else LCD_DAT &= (~LCD_RS);      /* select LCD command */
                                        /* register */
        if (type == DATA)
                LCD_DAT = c_hi|LCD_E|LCD_RS; /* output upper 4 bits, */
                                             /* E, RS high */
        else
                LCD_DAT = c_hi|LCD_E;   /* output upper 4 bits, E, */
                                        /* RS low */
        LCD_DAT |= LCD_E;               /* pull E signal to high */
        __asm(nop);                     /* Lengthen E */
        __asm(nop);
        __asm(nop);
        LCD_DAT &= (~LCD_E);            /* pull E to low */
        if (type == DATA)
                LCD_DAT = c_lo|LCD_E|LCD_RS; /* output lower 4 bits, E,
                                             RS high */
        else
                LCD_DAT = c_lo|LCD_E; /* output lower 4 bits, E, RS low *

        LCD_DAT |= LCD_E;               /* pull E to high */
        __asm(nop);                     /* Lengthen E */
        __asm(nop);
        __asm(nop);
        LCD_DAT &= (~LCD_E); /* pull E to low *
        delay_50us(1);                  /* Wait for command to */
                                        /* execute */
}

#define D50US 133                       /* Inner loop takes 9 cycles; */
void delay_50us(int n)                  /* need 50x24 = 1200 cycles */
{
        volatile int c;

        for (;n>0;n--)
                for (c=D50US;c>0;c--) ;
        }
```

```
void delay_1ms(int n)
{
        for (;n>0;n--) delay_50us(200);
}
```

File main.c:

```
#include <hidef.h>              /* common defines and macros */
#include "derivative.h"         /* derivative-specific definitions */
#include "lcd.h"

void main(void) {
        char *msg1 = "Hello, World!";
        char *msg2 = "From Bill";
        openlcd();              // Initialize LCD display
        puts2lcd(msg1);         // Send first line
        put2lcd(0xC0,CMD);      // move cursor to 2nd row, 1st column
        puts2lcd(msg2);         // Send second line
        __asm(swi);
}
```

Code to convert integer to character (courtesy of Riley Myers):

```
int main(void) {
        int value, data;
        unsigned char i, j;
        char msg[8] = {0};

        .
        .
        .

        value = hex2bcd(data);

        /* This will place the number in value in the first 4 characters of the
         * character buffer msg */

        for (i = 0, j = 3; j >= 0; i++, j--) {
                /* This takes the jth nybble of value, ands it with 0xf to
                 * isolate it, and then converts it to ascii by adding 0x30, the
                 * ascii value of zero. It then stores this in the ith element
                 * of msg. */
                msg[i] = 0x30 + ((value >> (4 * j)) & 0x0f);
        }
}
```

```
/**
 * @file lcd.c
 * @author Hector Erives; Revisions By Kelsey J. Harvey.
 * @date April 8, 2017.
 * @brief Functions used for outputting to the LCD display of the MC9S12
 * Micro-Controller.
 * @bugs None Yet Discovered.
 */

#include "lcd.h"

/**
 * Initialises the necessary bits and ports for LCD output
 * @param       None (void).
 * @return      None (void).
 */
void open_lcd(void)
{
        MOV_BYTE(LCD_DIRECTION, 0xFF);  /* configure LCD_DAT port for output */
        delay_1ms(100);                 /* Wait for LCD to be ready */
        put_to_lcd(0x28, COMMAND);  /* set 4-bit data, 2-line display, 5x7 font */
        put_to_lcd(0x0F, COMMAND);      /* turn on display, cursor, blinking */
        put_to_lcd(0x06, COMMAND);      /* move cursor right */
        put_to_lcd(0x01, COMMAND);      /* clear screen, move cursor to home */
        delay_1ms(2);           /* wait until "clear display" command complete */
 }

/**
 *      Sends either character data or command data to the LCD Display.
 *      @param  entry,  The data to be sent to the LCD Display.
 *      @param  type,   The type of data being sent (character or command).
 *      @return None, (void).
 */
void put_to_lcd(char entry, char data_type)
{
       char entry_hon  = (entry & 0xF0) >> 2;  /** The Higher-Order Nyble of 'entry'. **/
       char entry_lon  = (entry & 0x0F) << 2;  /** The Lower-Order Nyble of 'entry'.  **/

       /** Determines if the High-Order Nyble is a Command or Character. **/
       LCD_ENTRY = data_type ? entry_hon|LCD_SEND_CHARACTER : entry_hon|LCD_SEND_COMMAND;

       delay_230ns();  /** Delay for 230 nano-seconds. **/

       CLR_BITS(LCD_ENTRY, LCD_ENABLE_TRANSFER);       /** Clear Port K for new data. **/
       /** Determines if the Low-Order Nyble is a Command or Character. **/
       LCD_ENTRY = data_type ? entry_lon|LCD_SEND_CHARACTER : entry_lon|LCD_SEND_COMMAND;

       delay_230ns();  /** Delay for 230 nano-seconds. **/

       CLR_BITS(LCD_ENTRY, LCD_ENABLE_TRANSFER);       /** Clear Port K for new data. **/
       delay_50us(1);  /** Delay for 50 micro-seconds. **/
 }


 /**
  *     Sends a string to the LCD Display.
  *     @param  output_str,     The character string to be outputted to the display.
  *     @return None, (void).
  */
void puts_to_lcd (char *output_str)
{
        while (*output_str) {  /* While not at end of the string. */
        put_to_lcd(*output_str, CHARACTER);  /* Write character to LCD Display. */
        delay_50us(1);         /* Wait for said data to be written.    */
```

```
          output_str++;         /* Go to the next character with Pointer Arithmetic. */
        }
 }


/**
 *      Delays for 50 micro-seconds.
 *      @param  n,      The number of 50 us increments to delay for.
 *      @return None, (void).
 */
void delay_50us(int n) /* need 50x24 = 1200 cycles */
{
        volatile int c;
        while (n--){c = D50US; while (c--);}
 }


/**
 *      Delays for 1 milli-second.
 *      @param n, The number of milli-seconds to delay for.
 *      @return None, (void).
 */
void delay_1ms(int n)
{
        while (n--) delay_50us(200);
 }

/**
 *      Delays for 230 nano-seconds.
 *      @param None, (void).
 *      @return None, (void).
 */
void delay_230ns(void)
{
     _asm(NOP);
     _asm(NOP);
     _asm(NOP);
 }
```

```
/**
* @file lcd.h
* @author Hector Erives ; Revisions By Kelsey J. Harvey.
* @date April 8, 2017.
* @brief Function Prototypes for functions defined in lcd.c and Macro Definitions for user utilization.
* @bugs None Yet Discovered.
*/

#define LCD_ENTRY                *(volatile unsigned char*)(0x32)          /** Port K Redefinition **/
#define LCD_DIRECTION            *(volatile unsigned char*)(0x33)          /** DDRK Redefinition          **/
#define LCD_ENABLE_TRANSFER      0x02      /** Enable the transfer of Character/Command data when used on Port K. **/

#define LCD_SEND_COMMAND         0x02      /** Set Port K so as to send Command Data.                     **/
#define LCD_SEND_CHARACTER       0x03      /** Set PORT K so as to send Character Data.          **/

#define COMMAND                  0
#define CHARACTER                1

#define SET_BITS(BYTE, MASK)     BYTE |=  MASK
#define CLR_BITS(BYTE, MASK)     BYTE &= ~MASK
#define MOV_BYTE(BYTE, VALU)     BYTE  =  VALU

#define D50US 133 /* Inner loop takes 9 cycles; */

void open_lcd(void);
void put_to_lcd(char c, char type);
void puts2lcd (char *ptr);
void delay_50us(int n);
void delay_1ms(int n) ;
void delay_230ns(void);
```