AVR340: Direct Driving of LCD Using General Purpose IO

8-bit AVR® Microcontrollers

Application Note

Features

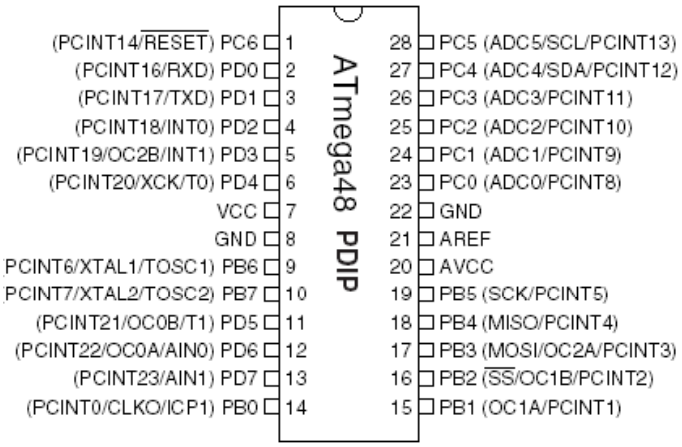
- **4K byte self-programming Flash Program Memory.**
- **512 byte SRAM, 256 Byte EEPROM.**
- **8 Channel 10-bit A/D-converter (TQFP/MLF).**
- **debugWIRE On-chip Debug System.**
- **Up to 20 MIPS throughput at 20 MHz.**
- **23 to 28 I/O lines, depending on package.**
- **PDIP (shown), TQFP and QFN/MLF packages available.**

1 Introduction

Many products require a Liquid Crystal Display (LCD) Interfaced to a microcontroller (MCU). This application note describes the operation of a Multiplexed LCD. Also discussed are electrical waveforms and connections needed by a LCD, as well as a C-program to operate the LCD. The result is an excellent low cost combination and a starting point for many products.

Although multiplexed LCDs are initially more complex to get operational, they are the lowest cost displays and require the lowest number of I/O pins of all glass LCDs (glass refers to the lack of an additional on-board LCD driver chip).

Figure 1-1. ATmega48 PDIP package



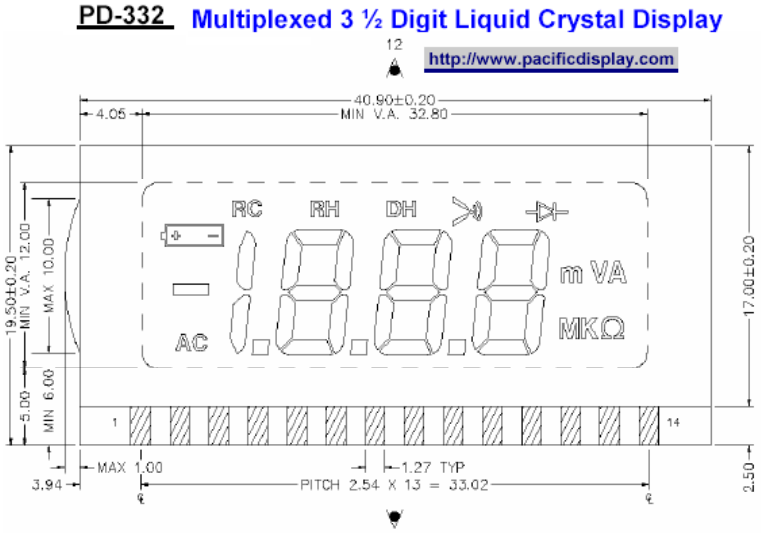
Rev. 8103A-AVR-09/07

2 Details on the LCD

The LCD described within is Glass-only device (no interface IC on the LCD). The LCD has 4 COM inputs and 8 segment inputs, requiring 12 I/Os on the ATmega48. There are 11 to 16 additional I/Os still available on the ATmega48, depending on the package chosen.

The LCD must be supplied with AC signals, typically square waves. LCDs will be damaged if supplied with static DC signals, and irreversible plating of the segments will occur. Figure 2-1 shows an example LCD footprint.

Figure 2-1. LCD footprint



The LCD segments will become visible if the voltage difference between a COM input and a segment input is typically 3.5 VAC.

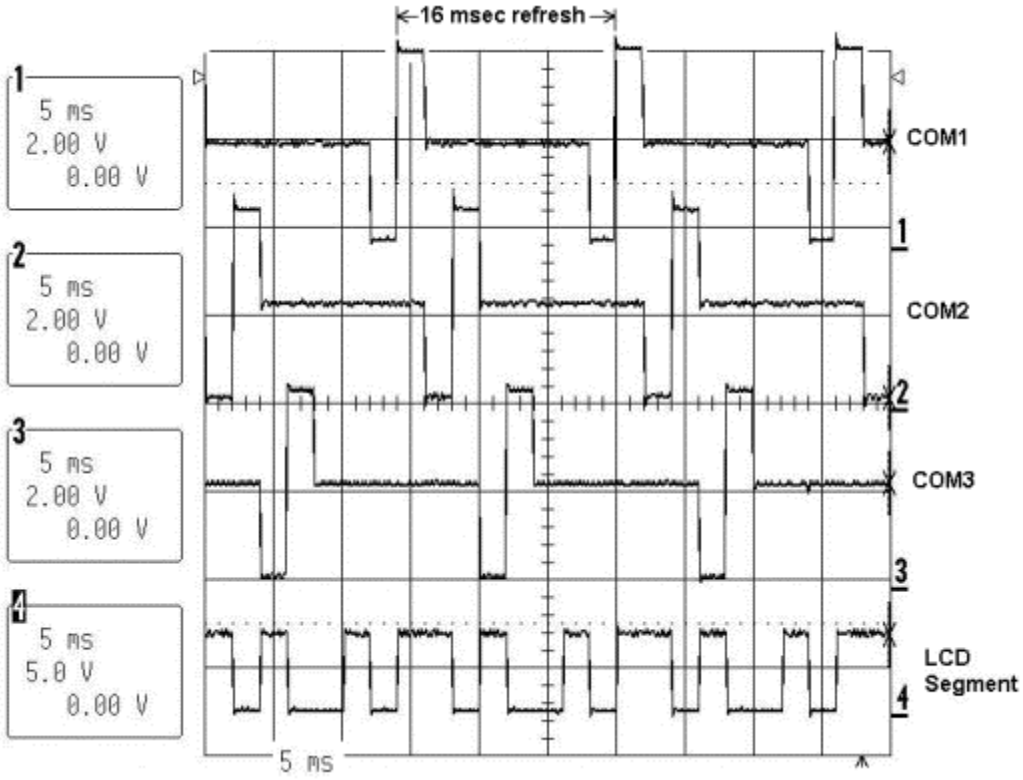
COM Inputs: In this example, the 4 COM inputs are each biased at ½ VCC. This is done to allow selected segments to be visible. Figure 2-2 shows 3 of the 4 COM waveforms. All 4 COMs are sequentially enabled to produce one AC cycle each. The total time for all 4 COMs to cycle is approximately 16 msec, which refreshes the LCD at about a 60 Hz rate. This is fast enough so no flicker will be noticed, and still slow enough to prevent ghosting of the ‘off’segments.

The scope image shows that each COM signal goes from ½ VCC to VCC for 2 msec, then to GND for 2 msec, then back to ½ VCC. This is referrd to as “half VCC excitation”. More complex LCDs often use “one third excitation”, but this is beyond the scope of this ap note.

For additional information on LCD excitation, see the STK502 User’s Guide, available at www.avr.atmel.com .

Each COM cycle is active for 4 msec, then returning to ½ VCC, the inactive state. This adds up to a total time for all 4 COMs to be 16 msec.

Figure 2-2. Scope screen dump



To energize a segment, the waveform to that segment must be 180 degrees out of phase with its COM waveform. The voltage difference between the segment and COM signals will therefore be typically 5 Volts AC, which causes the segment to become visible after 300-400 msec of refresh cycles.

To de-energize (turn off) a segment, the COM and segment waveforms should be in phase with each other while that segment's COM is active, that is, not at 2.5 volts. So, a LCD with all segments OFF will have the segment inputs IN PHASE with each COM input. (COM3 was omitted from Figure 2-2, since only 4 channels were available.)

3 Schematic Descritpion

The schematic in Figure 3-1 shows the simple interface between the ATmega48 and PD-383 LCD. Notice the pull up resistors on the COM signals: These resistors supply the ½ VCC voltage, which is necessary when a COM signal goes to the Inactive state. The ATmega48 simply sets the respective output pin to an input, and the resistors pull the signal to ½ VCC.

On the LCD, the 8 Segment connections have been re-labeled in this ap note as 1A, 1B, 2A, 2B, and so forth, to match the data within the C program's segment table.

Figure 3-1. Connections overview

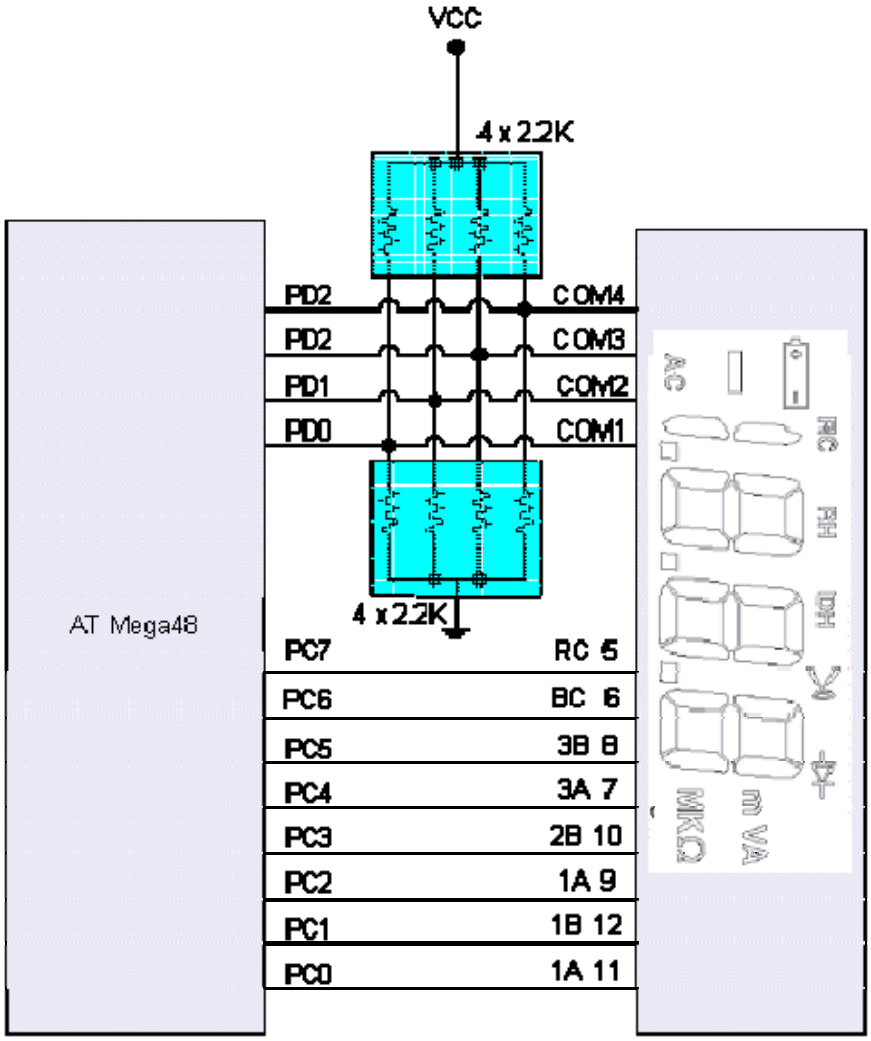
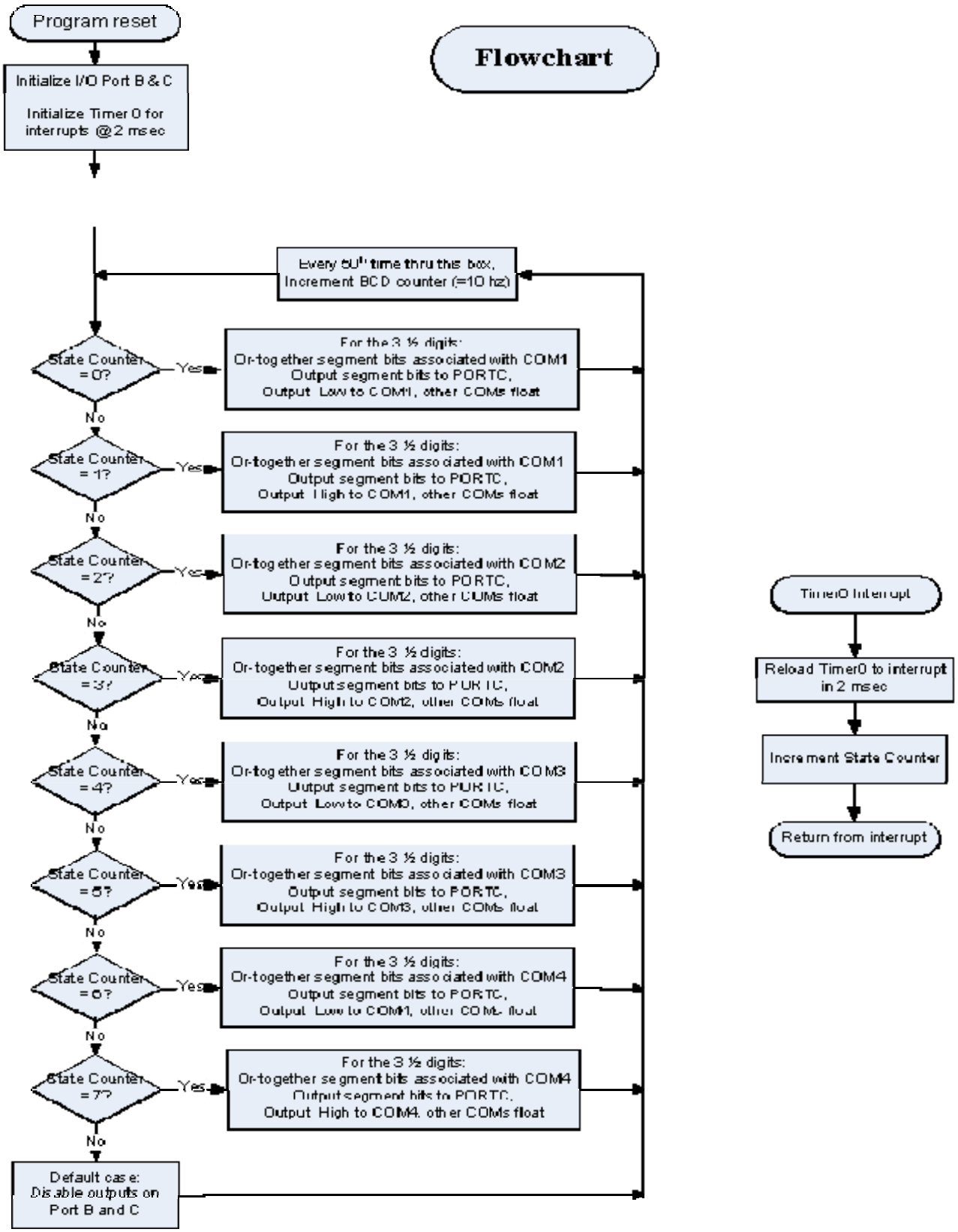


Figure 3-2. Program flowchart.



4 Look Up Table Operation

The Look Up Table (LUT), see Table 4-1 for an example, is a 10-byte long list of bytes that describe which segment to activate for characters 0-9. More characters could be defined requiring one byte per new character.

The basic operation is this:

- In the main program, 4 RAM locations contain the 4 characters to be displayed. These RAM locations contain a binary number from 0-9, and could be later modified to contain ASCII characters.
- For each character position, there are COMA and COMB inputs. When each of the 4 COM inputs are active, the COMA and COMB inputs are the result of a C language instruction where each character's LUT values are OR'd together.
- As an example, to display a "5" in the right-most character position, examine the last row entry of the LUT.

1. An interrupt occurs each 2 msec, which causes the Switch statement to be accessed.
2. When COM1 is active (case 0), the program accesses the LUT at row "5" and the masks off all but the 2 right-most bits. These are added to the "segs_out" variable, an AVR® RAM location.
3. Similarly, while COM1 is active, the program accesses the LUT three more times to OR the first 2 bits of the remaining three display characters into the segs_out variable.
4. A total of 2*4 = 8 bits are OR'd together into the segs_out variable, and then they are written to the PORTC output port.
5. Two msec later the Switch statement case 1 will be accessed. This time the segs_out data is XOR'd with 0xFF to reverse the polarities of all bits, which is required as AC waveforms are required by the LCD. These XOR'd (complimented) results are sent to PORTC.

- After another 2 msec has elapsed, the above sequence occurs again, this time while COM2 is active. Switch statement case 2 is executed. This time, data for each displayed character is accessed from the LUT and assembled in segs_out, then sent to PORTC in time with the COM2 active signal on PORTD.
- The sequence is duplicated for COM3 and COM4 signals. The entire 4 COM sequence takes 16 msec, that is, 2 msec for each of the 8 cases in the Switch statement.

Consult Figure 3-2 for further details on program flow.

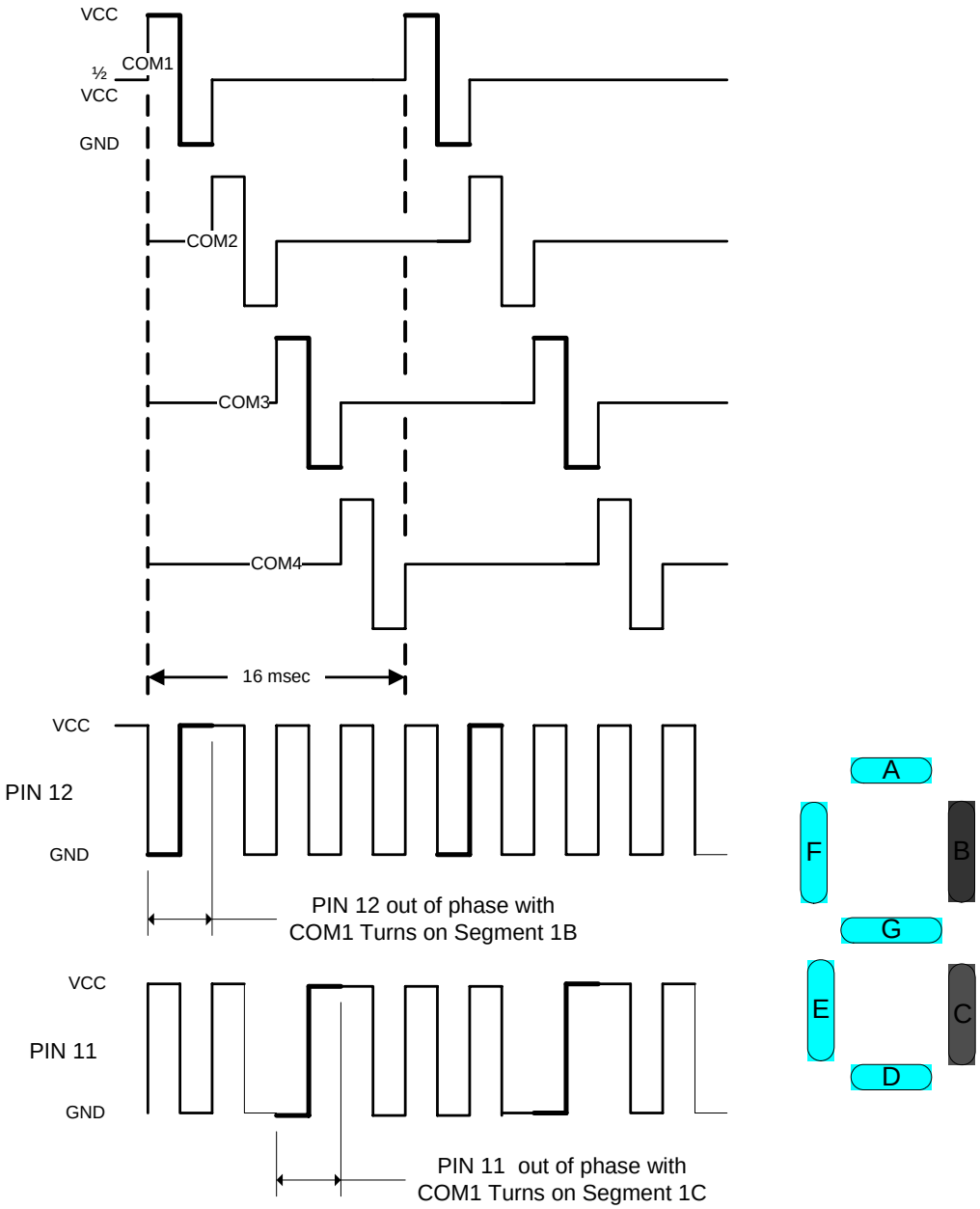
Table 4-1. Look up table (LUT).

	Segment activated								
	DP	**D**	**C**	**E**	**G**	**F**	**B**	**A**	
Character	**COM4 B**	**COM4 A**	**COM3 B**	**COM3 A**	**COM2 B**	**COM2 A**	**COM1 B**	**COM1 A**	**HEX**
0	0	1	1	1	0	1	1	1	77
1	0	0	1	0	0	0	1	1	22
2	1	1	0	1	1	0	1	1	DB
3	1	0	0	1	0	1	1	1	97
4	0	0	1	0	1	1	1	0	2E
5	0	1	1	0	1	1	0	1	6D
6	0	1	1	1	1	1	0	0	7C
7	0	0	1	0	0	0	1	1	23
8	1	1	1	1	1	1	1	1	FF
9	0	0	1	0	1	1	1	1	2F

5 Diagram of COM1-COM4 Signals

Figure 5-1 shows how the AC waveform is generated for two LCD segments. The COM signals are generated by the C program shown section 6. To activate an LCD segment, the opposite polarity waveform must be applied to the desired A or B input pins of the LCD. There are A and B inputs for each of the 4 character positions.

Figure 5-1. Phase details and AC wave form



6 Application code

```
//  File name:  LCX_App_Note_5_3_05.c  Demo C code to run Pacific
Displays' PPD-332 3 1/2 digit LCD
//  Compiled with CodeVision AVR, version 1.24.4a Evaluation
version, available from http://www.hpinfotech.ro
//  This program displays 4 incrementing decimal digits on the LCD.
LCD has 4 COMs and 8 segment connections (12 total)
// COM1-COM4 outputs on PORTD
//Segment Outputs on PORTC, called 1A, 1B, 2A, 2B, 3A, 3B to match
A&B values in segment_table array
```

```
// The PD-332 has identical LCD wiring for each of 3 digits; 2
segment pins per digit, labeled A and B
#include <mega48.h>
unsigned char segs_out = 0;
unsigned char state_counter = 8;
unsigned char output_change = 0;
unsigned char LCD_d_1       =0;    // counter starts at "000"
unsigned char LCD_d_2       =0;
unsigned char LCD_d_3       =0;
unsigned char LCD_d_4       =0;
unsigned char Pt_1_sec      =0; // 0.1 second counter

// Prototypes defined below
void initialization(void);

#define debug 0
#define LCD_Driver 1

//  Look Up Table (LUT) for 3 1/2 digit/4 COM LCD by Pacific
Displays, #PD-332
// The following table has 10 entries to display chars 0-9. Hex
values are COM1-COM4 for LCD inputs A & B.
const unsigned char segment_table[] =
{0x77,0x22,0xDB,0x97,0x2E,0x6D,0x7C,0x23,0xFF,0x2F}; //display
numbers 0-9

//************************Timer 0 overflow interrupt service
routine***********************
//
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
// Re-load Timer 0 value
TCNT0=5; //Timer0 period =  0.125 usec = 8MHZ/64.  2msec = 8
usec*250 5 = 255-250 5/4/05
        state_counter++;
       output_change = 1;  // This is a flag for main loop
       if (state_counter > 7) state_counter = 0;
}
//*********************End of Interrupt Service
Routine************************************

//***********************Main Begins
here*****************************************************
       void main(void) {
//* * * * * Call Initialization function * * * * * *see function
defined below* * * * * * *
 initialization();
```

```
//* * * * * * * Eanble Global enable interrupts* * * * * * * * ** *
#asm("sei")

//**********The following infinate While Loop contains the Switch
statement for LCD refresh******
#if LCD_Driver
while (1)
      {
        if(output_change){
        output_change = 0;

//  The following state_counter generates the 4 COM output
waveforms via PORTD, each with HI and LOW outputs
       switch (state_counter) {

    case 0:     {
                segs_out = (segment_table[LCD_d_1]& 0x03);  //get
digit_1's A & B bits
                segs_out = segs_out | ((segment_table[LCD_d_2]&
0x03)*4);  //get digit_10's A & B bits
                segs_out = segs_out | ((segment_table[LCD_d_3]&
0x03)*16); //get digit_100's A & B bits
                segs_out = segs_out | ((segment_table[LCD_d_4]&
0x03)*64); //get digit_1000's A & B bits
                DDRD = 0;
                PORTD = 0x00;
                PORTC = segs_out;
                DDRC  = 0xFF; // always on
                DDRD  = 0x01; //COM1 asserted LOW


                }
    break;
    case 1:     {
                PORTD = 0x01;
                PORTC = segs_out ^ 0xFF; // Compliment segment
outputs
                DDRC  = 0xFF; // always on
                DDRD  = 0x01;  //COM1 asserted High
               }
    break;
    case 2:     {
                segs_out = (segment_table[LCD_d_1]& 0x0C)/4;  //get
digit_1's A & B bits
                segs_out = segs_out | (segment_table[LCD_d_2]&
0x0C);//get digit_10's A & B bits
                segs_out = segs_out | ((segment_table[LCD_d_3]&
0x0C)*4); //get digit_100's A & B bits
```

```
                segs_out = segs_out | ((segment_table[LCD_d_4]&
0x0C)*16); //get digit_1000's A & B bits

              DDRD  = 0;
                PORTD = 0x00;
                PORTC = segs_out;
                DDRC  = 0xFF; // always on
                DDRD  = 0x02;  //COM2 asserted LOW
                }
    break;
    case 3:     {
                PORTD = 0x02;
                PORTC = segs_out ^ 0xFF; // Compliment segment
outputs
                DDRC  =0xFF;
                DDRD  = 0x02;  //COM2 asserted High
                }
    break;
     case 4:    {
                segs_out = (segment_table[LCD_d_1]& 0x30)/16;
//get digit_1's A & B bits
                segs_out = segs_out | ((segment_table[LCD_d_2]&
0x30)/4);//get digit_10's A & B bits
                segs_out = segs_out | (segment_table[LCD_d_3]&
0x30);//get digit_100's A & B bits
                segs_out = segs_out | ((segment_table[LCD_d_4]&
0x30)*4);//get digit_1000's A & B bits

                DDRD = 0;
                PORTD = 0x00;
                PORTC = segs_out;
                DDRC  = 0xFF;
                DDRD  = 0x04;  //COM3 asserted LOW
                }
    break;
     case 5:    {
                PORTD = 0x04;
                PORTC = segs_out ^ 0xFF; // Compliment segment
outputs
                DDRC  = 0xFF;
                DDRD  = 0x04;  //COM3 asserted High
                }
    break;
    case 6:     {
                segs_out = (segment_table[LCD_d_1]& 0xC0)/64;
//get digit_1's A & B bits
                segs_out = segs_out | ((segment_table[LCD_d_2]&
0xC0)/16); //get digit_10's A & B bits
```

```
                segs_out = segs_out | ((segment_table[LCD_d_3]&
0xC0)/4); //get digit_100's A & B bits
                segs_out = segs_out | (segment_table[LCD_d_4]&
0xC0); //get digit_1000's A & B bits

                DDRD = 0;
                PORTD = 0x00;
 //               PORTC = 0x00;//LCD_d_3 ^ 0xFF; // Compliment
segment outputs
                PORTC = segs_out;
                DDRC  = 0xFF;
                DDRD  = 0x08;  //COM4 asserted LOW
                }
    break;
    case 7:     {
                PORTD = 0x08;
                PORTC = 0x55;
 //                 PORTC = 0xFF; //LCD_d_3;
                PORTC = segs_out ^ 0xFF; // Compliment segment
outputs
                DDRC  = 0xFF;
                DDRD  = 0x08;  //COM4 asserted High
                }
    break;

   default:     DDRC = 0;
                DDRD = 0;       // COM1-COM4 float
    }
// Increment a counter to measure out 0.1 sec
        Pt_1_sec++;
        if (Pt_1_sec >=50){//.1 sec
                Pt_1_sec = 0;
                LCD_d_1++;      // 3 1/2 digit ripple BCD counter
for LCD digits
                if (LCD_d_1 >=10){
                    LCD_d_1 = 0;
                    LCD_d_2++;}
                if (LCD_d_2 >=10){
                    LCD_d_2 = 0;
                    LCD_d_3++;}
                if (LCD_d_3 >=10){
                    LCD_d_3 = 0;
                    LCD_d_4++;}
                      }// end .1 sec

                }  // ***************End of Switch
Statement************************
```

```
}
#endif
//***********************End of infinite While
loop*********************************
}
//***********************End of
Main*************************************************

//***********************Initialization function defined
here**********************
void initialization(void) {

// Declare your local variables here

// Crystal Oscillator division factor: 1
CLKPR=0x80;
CLKPR=0x00;

//DDRC=0x7F;  //7 Segment outputs

// Timer/Counter 0 initialization
TCCR0A=0x00;
TCCR0B=0x03;   // = 8MHz/64   3/22/05
TCNT0=0xC1;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// Interrupt on any change on pins PCINT0-7: Off
// Interrupt on any change on pins PCINT8-14: Off
// Interrupt on any change on pins PCINT16-23: Off
EICRA=0x00;
EIMSK=0x00;
PCICR=0x00;
// Timer/Counter 0 Interrupt(s) initialization
TIMSK0=0x01;
// Timer/Counter 1 Interrupt(s) initialization
TIMSK1=0x00;
// Timer/Counter 2 Interrupt(s) initialization
TIMSK2=0x00;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture Dy Timer/Counter 1: Off
//************************End of Initialization function
*************************
                }
```

7 References

Data sheet and more detail at http://www.atmel.com/products/AVR/

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature