Microcontrollers

ApNote AP0829

☐ or ☑ additional file
AP082902.EXE available

LCD Control Using the C505L

This Application Note shows how the C505L can be used to control a Liquid Crystal Display. The example provided uses the C505L LCD controller and A/D Converter to implement a simple voltmeter.

Author: M. Copeland – S.M. Wong / Microcontroller Applications

05.99, Rel. 02

AP0829 ApNote - Revision History		
Actual Revision : Rel.02	Previous Revision: Rel.01	
Page of actual Rel.	Page of prev. Rel.	Subjects changed (since last release)
		Real Time Clock Description, References and Code Removed

1 Introduction

In many applications it is necessary for a microcontroller to display some information to a user. Information can be displayed using Light Emitting Diodes (LEDs), segmented LED displays, dot matrix Liquid Crystal Displays (LCDs), or segmented LCDs.

Segmented LCDs are popular because they use less power than segmented LED displays, convey more information than individual LEDs, and require fewer I/O lines than dot matrix LCD displays.

This Application Note focuses on interfacing Segmented LCDs to the Infineon C505L 8-bit microcontroller. The example software demonstrates the configuration of the LCD controller and Analog-to-Digital converter for use as a simple voltmeter.

2 Theory of Liquid Crystal Displays

Liquid Crystal Displays are a inexpensive, low power means to display information. Unfortunately, the structure and control schemes of the displays are usually complicated. LCD control becomes even more complicated when the structure is designed to reduce the display pin count.

2.1 Structure of Liquid Crystal Displays

Figure 1 shows the cross-section of a simple LCD. An LCD is made of a liquid crystal sandwiched between two thin, grooved filaments. Transparent electrodes are placed above and below this structure. The placement of the electrodes defines the segments of the display. The “electrode, filament, liquid crystal” structure is then sandwiched between two layers of glass. Polarizing filters are then placed above the top piece of glass and below the bottom piece of glass. Beneath the bottom filter is usually a reflector. Sometimes the bottom polarizing filter and the reflector are combined into a single “polarizing reflector”.

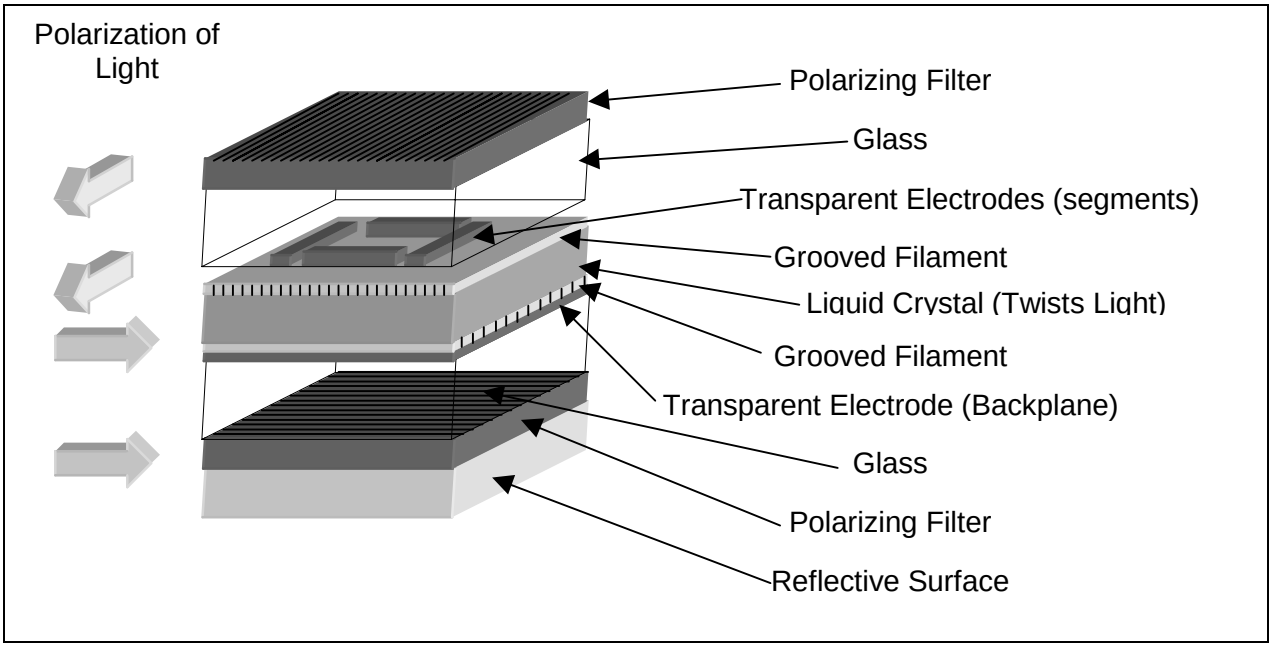


Figure 1:
Structure of an LCD

As light enters the top filter, it gets polarized. The polarized light is not effected by the glass, or the transparent electrodes.

The liquid crystal molecules align themselves with the grooves in the grooved filaments. The grooves of the bottom filament are rotated 90 degrees with respect to the top filament. This causes the liquid crystal molecules to twist like a helix. As the polarized light passes through the liquid crystal, it remains polarized, and twists 90 degrees, as shown in Figure 2.

The light then passes through the bottom transparent electrode and glass. The bottom polarizing filter is rotated 90 degrees with respect to the top filter. Since the light that reaches the bottom filter has been twisted, it passes through the filter unaffected. The light is then reflected back through the structure in the same way.

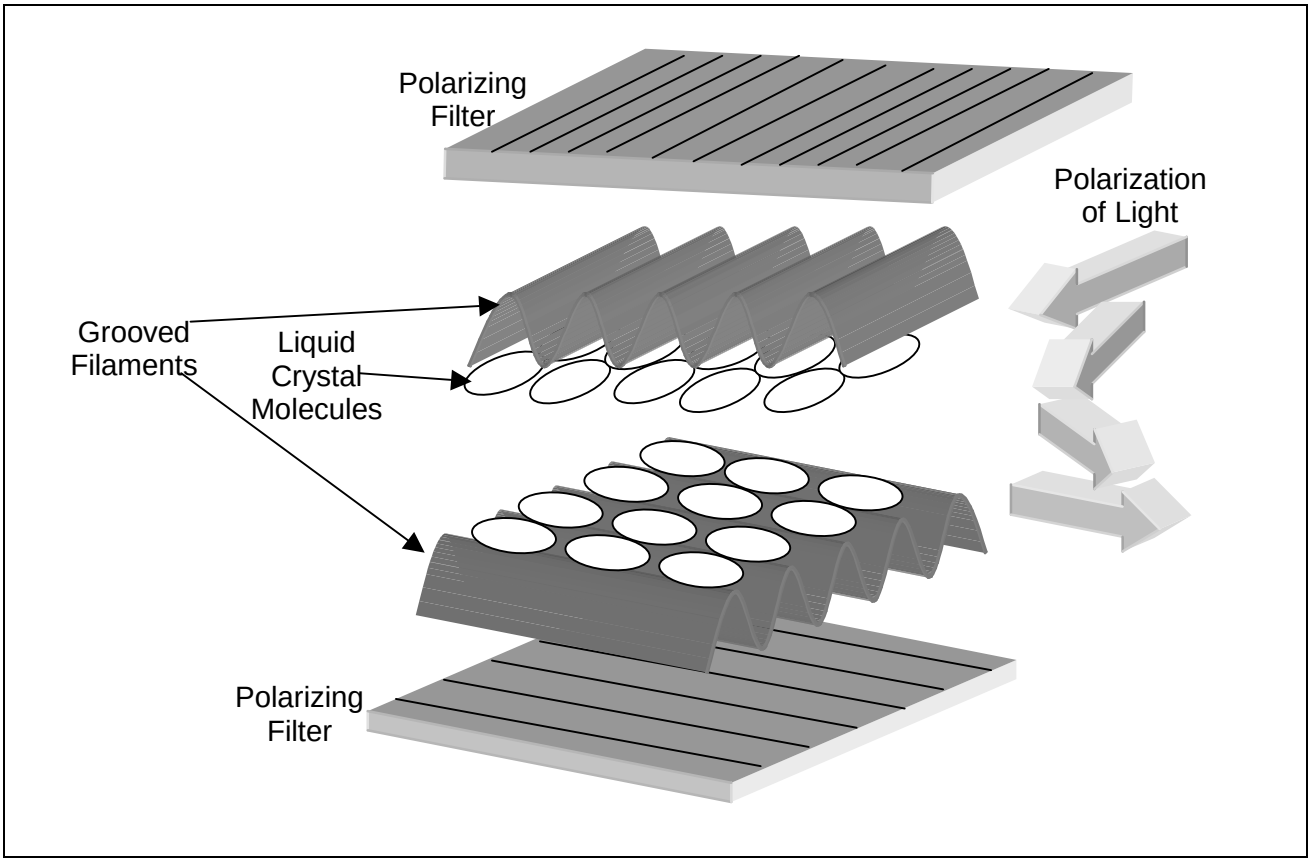


Figure 2:
Liquid Crystal Twisting Polarized Light

If there is a potential difference between the top electrode and the bottom electrode, the molecules of the liquid crystal that are between the electrodes align themselves with the electric field that is generated. This means that the liquid between the electrodes will no longer twist the light that passes through it.

In this case, the light will pass through the top filter and get polarized. It will then be unaffected by the glass, the electrodes, and the liquid crystal. The polarized light will not pass through the bottom filter because it is rotated 90 degrees with respect to the top filter as shown in Figure 3. This causes the area between the two plates to appear dark. The dark area is referred to as a segment.

Since the electrodes in the display do not physically touch each other, the LCD uses very little current. The current that is used is due to the capacitance of the electrodes.

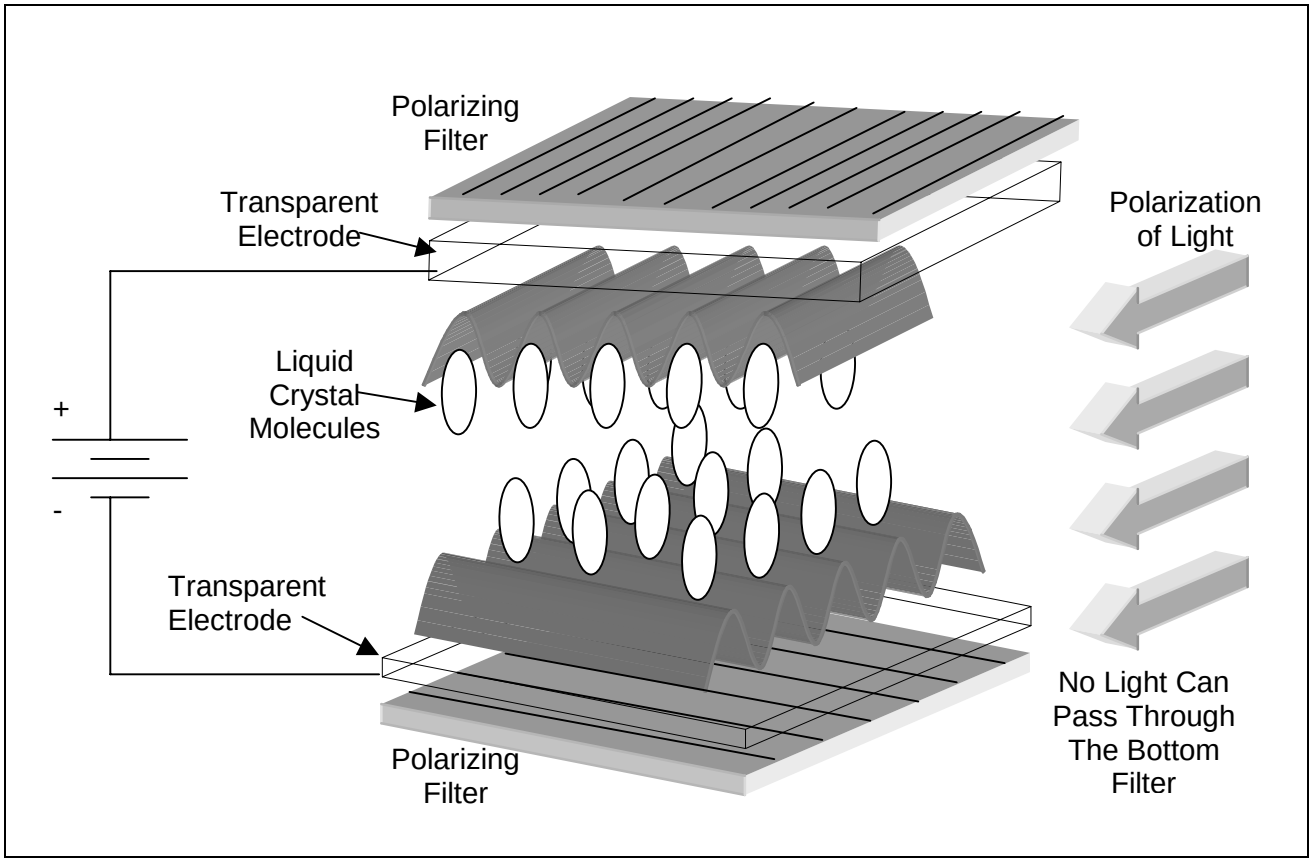


Figure 3:
Liquid Crystal with Externally Applied Potential Difference

2.2 Multiple Backplanes to Reduce the Pin Count

For the simple LCD shown in Figure 1, the bottom electrode covers the entire surface of the LCD. This electrode is referred to as a backplane. The segments are defined by the shape of the top electrodes in Figure 1. So, in order to individually control each segment, a wire (and a pin) must be connected to each electrode on the top surface, and one additional pin will be needed for the backplane. For displays with a large number of segments, this solution is not practical.

To reduce the number of pins required to drive the display, often more than one back-plane is used. When there is more than one backplane, more than one electrode (on the top layer) can be connected to the same pin as shown by the electrode configurations in Figure 4. When 4 backplanes are used each new pin can control 4 segments.

The pins connected to the electrodes on the top surface will be referred to as *column* pins. The backplane pins will be referred to as *row* pins. Having multiple row (backplane) pins allows the row and column pins to form a matrix, so that each segment can be individually controlled by selectively applying voltage to the appropriate row and column pin.

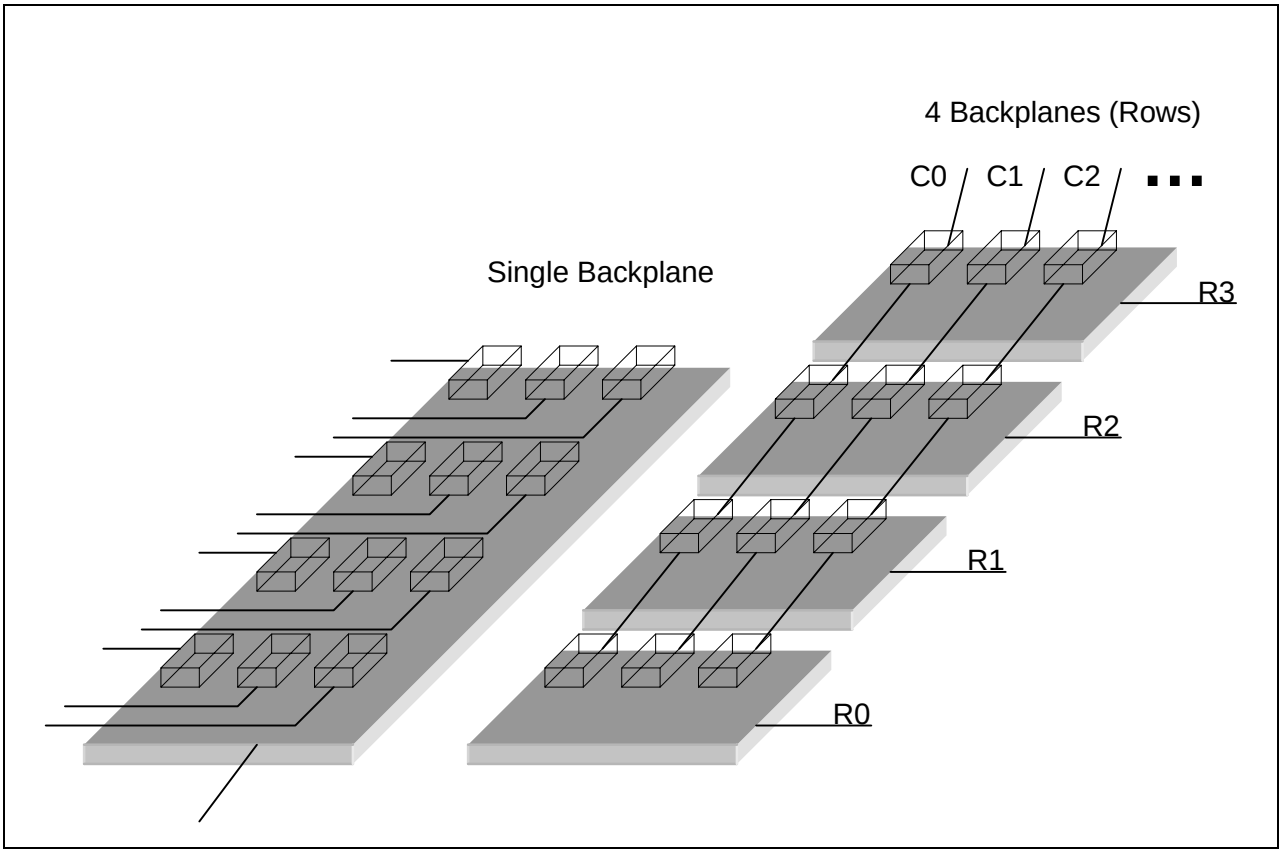


Figure 4:
Electrode Configuration of Single and Multiple Backplane LCDs

Unfortunately, when an LCD has multiple backplanes, it is impossible to control the segments using static voltages. For example, consider the multiple backplane LCD in Figure 4. Let's assume that one wishes to activate the segments in column 1 row 1 and column 1 row 2. To do this, static voltages can be applied to the C1, R1, and R2 pins. Suppose in addition to these two segments, one wishes to activate the segment in column 2 row 1. The only way to do this is to apply a static voltage to pin C2, but this will also cause the segment in column 2 row 2 to become active. So to control any combination of segments the row and column signals must be time multiplexed (there are other reasons why static voltages cannot be used on LCDs).

The LCD controller of the C505L has the ability to control 4 backplanes and up to 32 columns. This means that 128 segments can be controlled.

3 Control of LCDs

Unfortunately, if a voltage difference between two electrodes is maintained for too long, the liquid crystal may become permanently aligned in one direction, thus ruining the display. To ensure that the liquid does not become permanently aligned, the voltage difference between the two plates must be toggled between positive and negative. Since

the voltages that are applied to the crystal must be toggled periodically, the control scheme becomes even more complicated.

When the LCD controller of the C505L is activated, it applies the voltages shown in Figure 5 to the row (backplane) pins. These signals are periodic and are generated by the LCD controller with **no CPU intervention**.

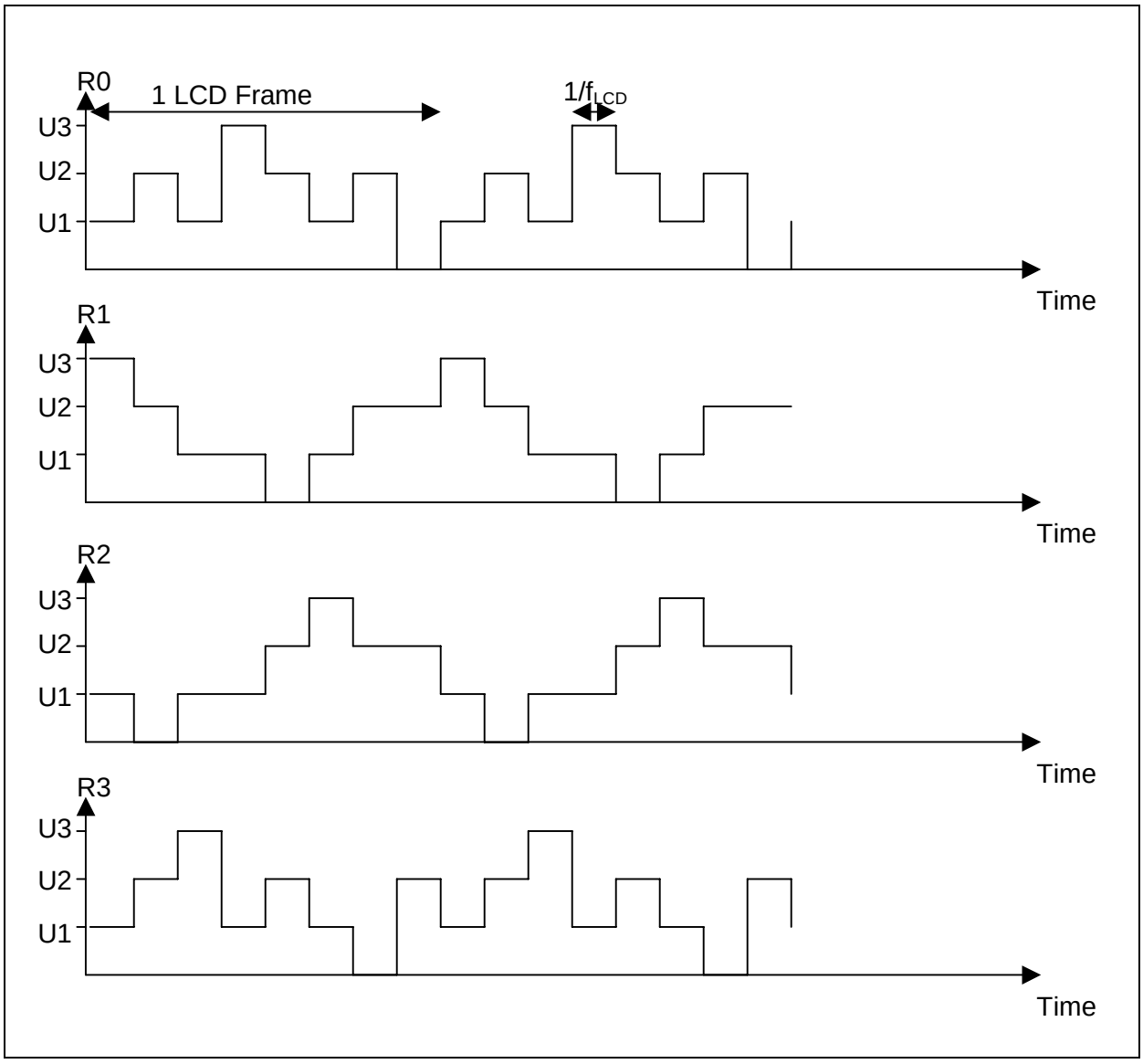


Figure 5:
Voltage Applied to the Row Pins

The voltage levels, U1, U2, and U3, are created by the C505L's on board programmable D/A converter. U3 is programmable using the on-chip D/A converter, and the voltages for U1 and U2 are always U3 ÷ 3 and (2÷3)U3, respectively. Since the D/A converter is programmable, the C505L can be used on a wide variety of LCDs. The value of U3 can also be changed "on the fly" to adjust the contrast of the display.

The timing of the row signals in Figure 5 is also programmable. The LCD controller has a dedicated timer unit which is used to provide the clock signal for the controller. The clock frequency (f_{LCD}) is programmable and can be up to 360 Hz. The LCD controller can also be clocked by the on-board Real Time Clock unit (this allows the LCD to remain active even during power-down mode).

Since the row signals vary according to Figure 5, a constant voltage on the column pins would cause all of the segments of the LCD to become active. To stop the segments from becoming active, the voltage shown in Figure 6 (solid line) is applied to each column signal.

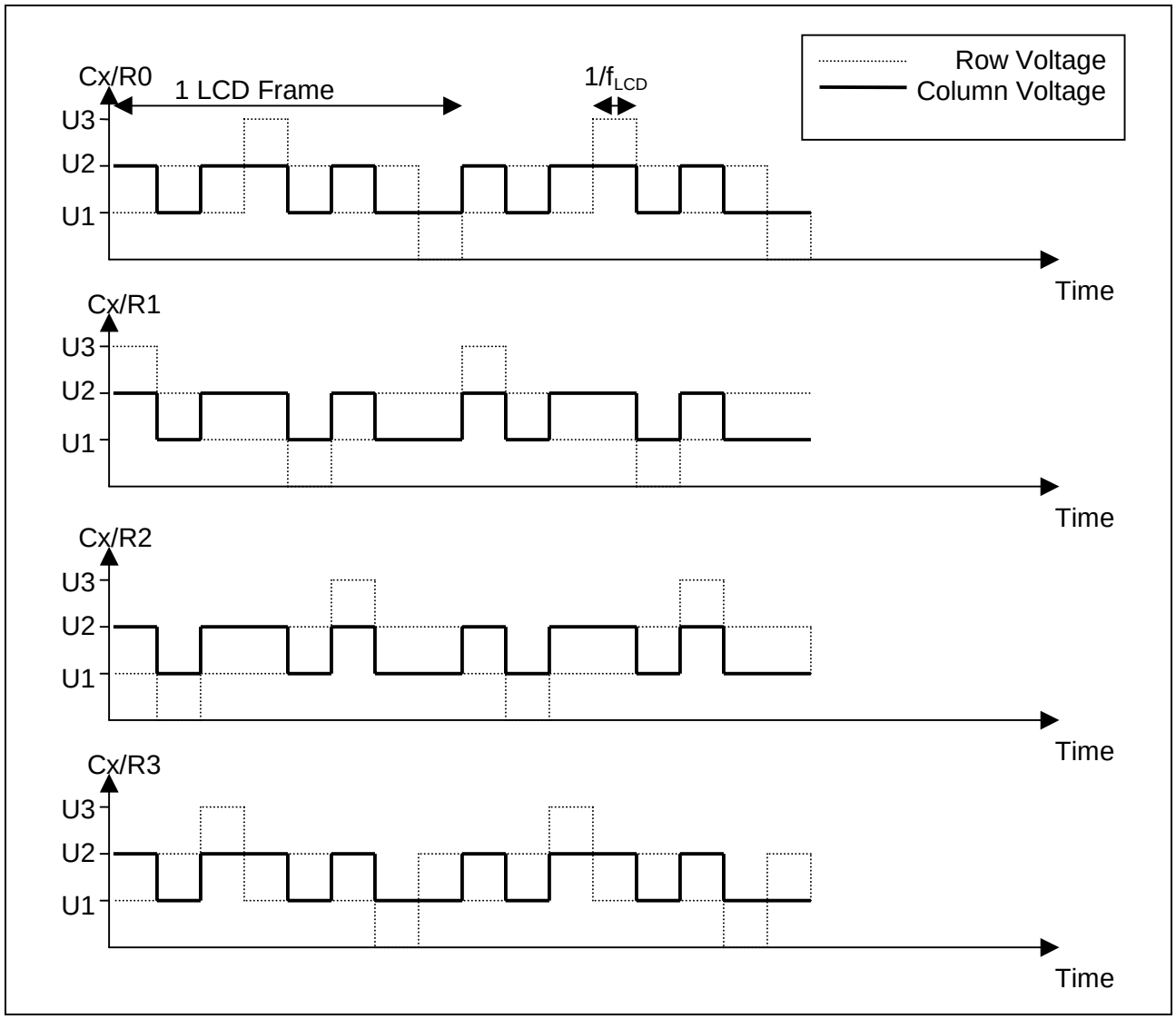


Figure 6:
Column Pin Voltage for LCD Segments to remain inactive

Figure 6 shows the column signal that should be applied to ensure that no LCD segment is activated. The dotted lines in Figure 6 represent the row signals from Figure 5. As

Figure 6 indicates, the voltage difference between the row and column pins is never more than 1U, which is not enough to activate a segment. Like the row signals, the column signals are generated without CPU intervention.

To activate a segment, the column pin voltages should be varied so that a larger voltage is applied. In Figure 7, the voltages are varied so that two segments will be activated and two segments will remain inactive.

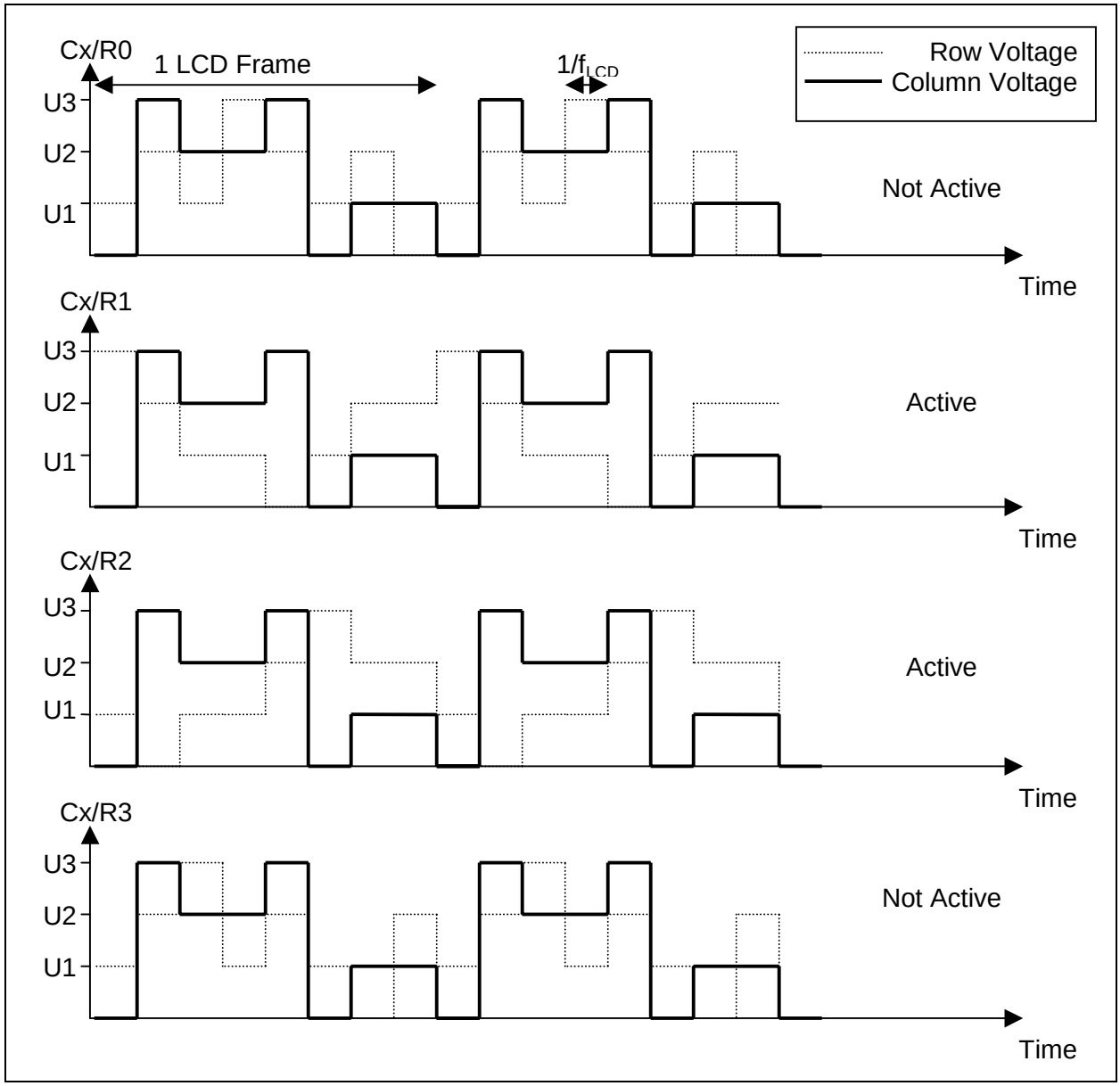


Figure 7:
Activation of LCD Segments

As Figure 7 indicates, if the column pin was connected to four segments, two of the segments would be activated and two of the segments would remain inactive. The

segment is active when the voltage difference of 3U is present on the LCD pins. As Figure 7 shows, the voltage difference alternates between +3U and –3U on the active segments. This keeps the liquid crystal molecules from becoming permanently aligned in one direction.

As indicated by Figure 7, the activated segments only receive the full 3U voltage difference during 2 of the eight $1/f_{LCD}$ periods that make up an LCD frame. Since the segments are only active for 2/8 of the LCD frame, this control scheme is referred to as the ¼ driving method.

Since each column controls 4 segments, 2 to the power of 4 = 16 different voltage sequences are required on each of the 32 column pins in order to fully control the 128 segments in an LCD. The LCD controller of the C505L has the ability to do this without CPU intervention.

4 LCD Controller of the C505L

The LCD controller of the C505L makes it possible to ignore all of the details of LCD structure and control presented in the two previous sections. Figure 8 shows a block diagram of the LCD controller of the C505L.

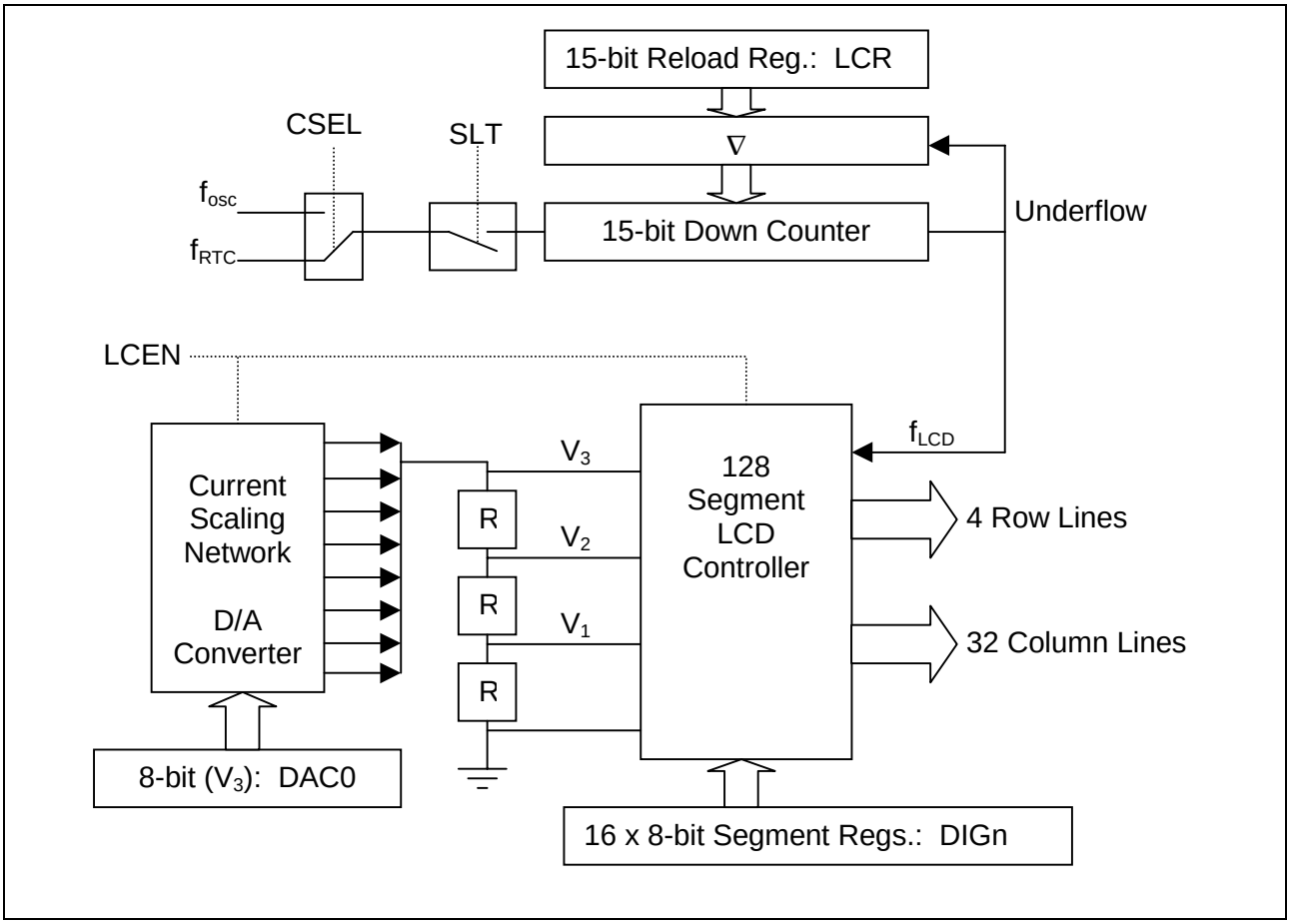


Figure 8:
Block Diagram of LCD Controller

The C505L allows a great deal of flexibility in choosing an LCD. The maximum voltage applied to the LCD is controlled by the DAC0 register. The LCD frequency (f_{LCD}) is controlled by the 15-bit reload value placed in the LCRH and LCRL registers. The 15-bit down counter can be clocked by the external system oscillator, or by the Real Time Clock oscillator.

The C505L generates up to 4 row outputs and up to 32 column outputs. If fewer than 32 column outputs are needed, 8 or 16 of the pins can be used as digital I/O pins.

There are 16 x 8-bit registers that are used to individually control the 128 segments. Each bit in these registers represents a segment. Setting a bit activates a segment by automatically applying the appropriate voltages and timing to the column pins. Setting multiple bits activates multiple segments. Each of the 16 registers controls 2 consecutive column outputs as shown in Figure 9.

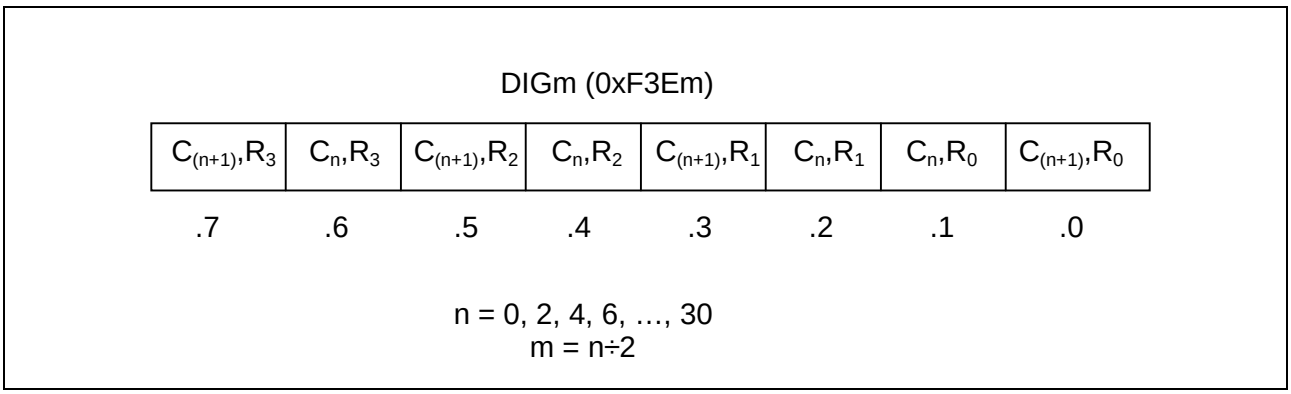


Figure 9:
LCD Segment Registers

5 Connecting an LCD to the C505L

When connecting an LCD to the C505L, it is important to consider the order in which the column pins are connected to the LCD. For example, consider an LCD that is made up of 8 identical characters each with 8 segments. Each character is controlled by 2 column pins as shown in Figure 10.

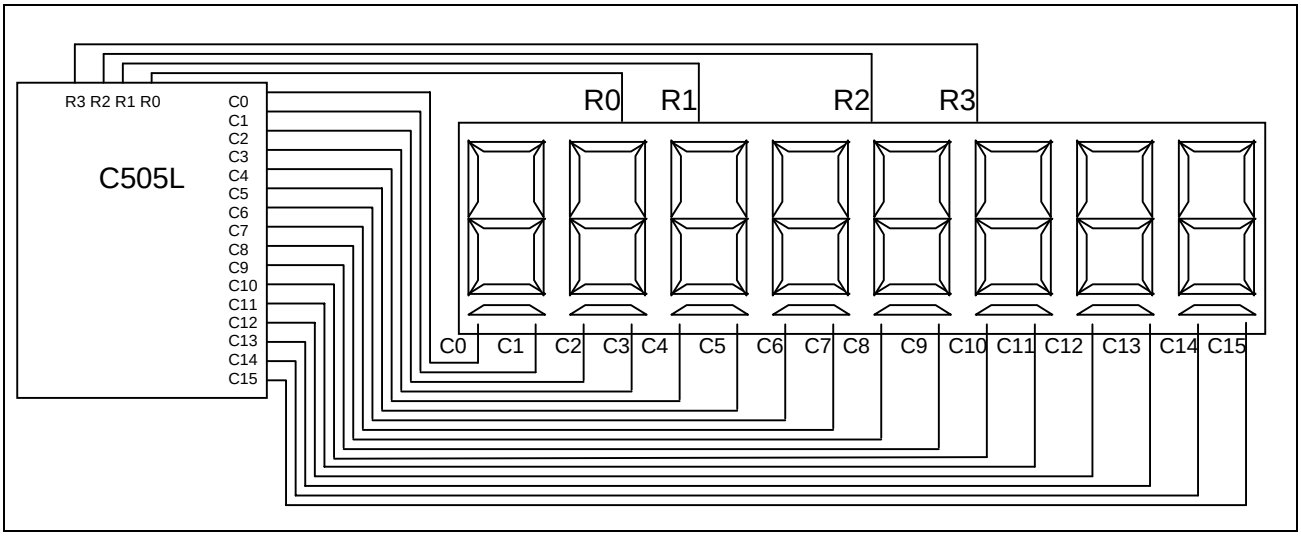


Figure 10:
Connection of LCD

For the type of display shown in Figure 10 it would be more convenient (but not necessary) if the LCD is connected such that the first character of the display is controlled by the C0 and C1 pins, and the second character is controlled by the C2 and C3 pins, etc. When the display is connected in this manner, each DIGm register controls a character on the LCD. So if writing a 0x15 to DIG0 makes a “7” appear in the first character of the display, writing a 0x15 to DIG1 will make a “7” appear in the second character of the display. This type of consistency makes working with the LCD much easier.

The evaluation board for the C505L Starter Kit uses an LCD that has eight characters. Each character contains 16 segments. The LCD is constructed such that each character is controlled by 4 column pins. As the schematic of the evaluation board shows, each LCD character is controlled completely by two consecutive DIG registers. This is convenient when programming in C because the two DIG registers that control each character can be defined as a single 16 bit INT variable since they are located in consecutive memory addresses. Figure 11 shows how the first character of the LCD on the Starter Kit evaluation board is connected to the C505L. The other seven characters of the display are connected in a similar manner.

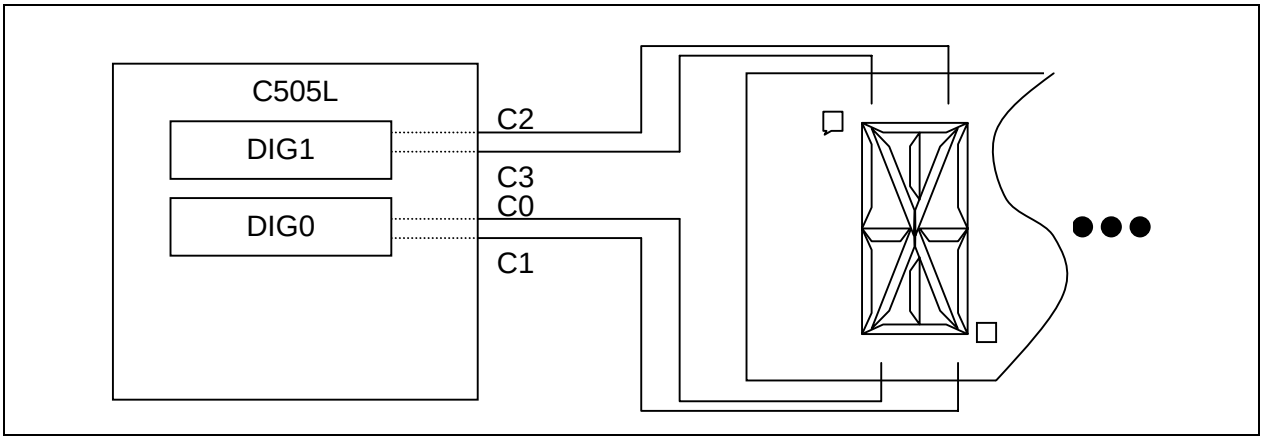


Figure 11:
Connection of LCD on Evaluation Board

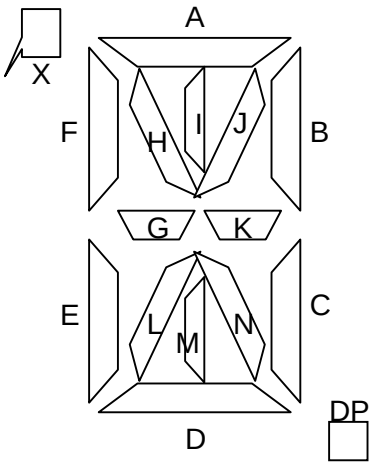
It should be noted that connecting an LCD as shown in Figure 10 and Figure 11 is not necessary, but it may make displaying text easier. Displays that contain symbols and other non-text information, or displays that have a small number of text characters may be connected in any way that suites the circuit designer and programmer.

6 The Application

To demonstrate the configuration and use of the C505L LCD controller, an example program has been created. The example application uses the LCD controller and A/D converter to implement a simple volt meter.

6.1 Hardware and Connections

The application is designed to run easily with the C505L Starter Kit. The LCD on the C505L starter kit is the VL Electronics VIM-878 128 segment 8 character display. Figure 12 shows how each character of the display is configured. The configuration shown in Figure 12 makes it easy to control each character by associating the character with a 16 bit integer variable. So the bits in each variable control the segments in each character.



Segment	Column	Row
A	n+2	0
B	n+2	1
C	n+2	2
D	n	3
E	n	2
F	n	1
G	n+3	1
H	n+3	0
I	n+1	0
J	n+1	1
K	n+1	2
L	n+3	2
M	n+3	3
N	n+1	3
DP	n+2	3
X	n	0

n = 0, 4, 8, …, 28

DIGn (0xF3En)								DIGn+1 (0xF3En+1)							
N	D	K	E	J	F	X	I	M	DP	L	C	G	B	A	H

n = 0, 2, 4, 6, 8, … , 14

Figure 12:
Configuration of LCD on Evaluation Board

This application requires the use of the onboard analog-to-digital converter of the C505L. To use the A/D converter an external reference voltage and ground (V_{AREF} and V_{AGND}) must be applied. For this example V_{AREF} should be connected to V_{CC} (5 V) and V_{AGND} should be connected to V_{SS} (0 V). These connections are already made on the starter kit board. This example will use A/D channel 4 (P1.4) to read the externally applied analog voltage.

6.2 Software Algorithms

The example software for this application note implements a voltmeter. The voltmeter is implemented very simply using the A/D converter and the LCD controller of the C505L. A Timer 0 interrupt is used to initiate the conversion to ensure an even distribution of sampling points. Figure 11 shows the structure of the software.

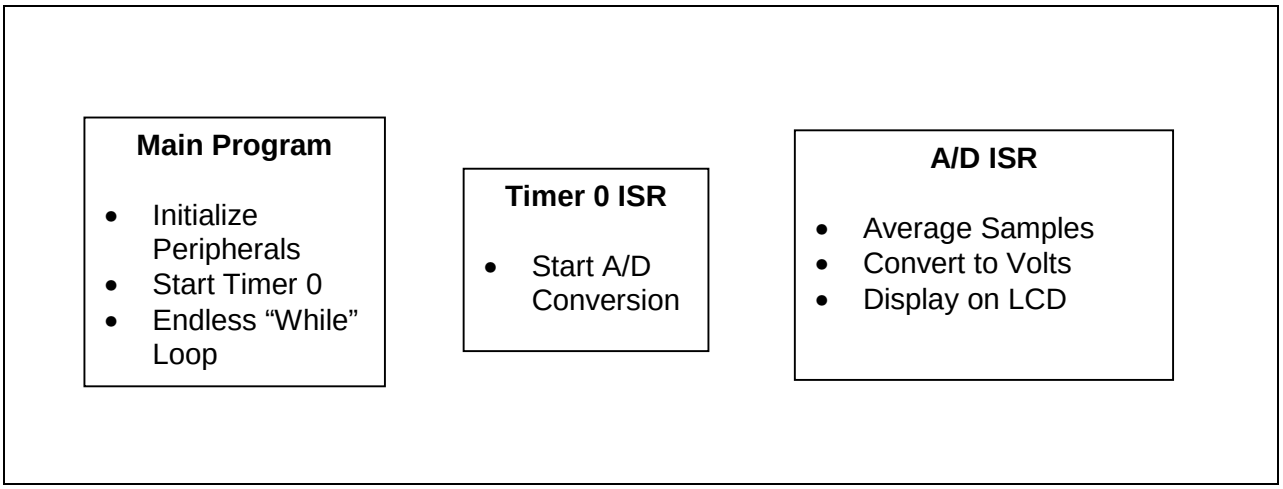


Figure 11:
Structure of Voltmeter Software

Below is a brief description of how the peripherals are configured:

Timer 0 -
Timer 0 is placed in Mode 0 (8 bit timer with divide by 32 prescaler). This causes an interrupt to occur every (8192)* 6 / f_{osc} seconds (4.096 ms with a 12 MHz external clock). In the Timer 0 Interrupt Service Routine (ISR), the A/D conversion is started.

A/D Converter -
The A/D converter is placed in single conversion mode (bit ADM is set to 0). When the conversion is complete, an interrupt is generated. The A/D ISR reads 9 bits of the 10 bit A/D result. This provides an accuracy of approximately 10 mV, so two digits to the right of the decimal are displayed on the LCD.

128 results are averaged together. Averaging 128 results ensures a consistent result and still allows the sum of all the results to be contained in a 16 bit integer variable.

The average of the 128 samples is then converted into volts * 100 by multiplying by 91/93. This provides the closest approximation that can be accomplished using simple 16 bit integer math (the exact conversion is 5*100/511). The resulting 16 bit value is then displayed in decimal format on the LCD and a decimal point is added after the second digit.

Appendix A shows the C code for the application. The code was created for use with the Keil C51 compiler.

Appendix

A Main.c

```
#include <Reg505L.h>   // C505L SRFs
#include <LCD.h>   // Configuration of LCD
#include <stdio.h> // Contains printf

////////////////////////////////////////////////////////////
//  Application Example for ApNote 8029 "LCD Control
//  Using the C505L"
////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////
//  Main Routine
////////////////////////////////////////////////////////////
void main(void)
{
    /////////////// Functions ///////////////
        void Init(void);


    Init();         // Initialize LCD, T0, A/D,
    EA = 1;         // Enable All Interrupts

    TR0 = 1;        // Start Timer 0

    while (1);      // endless loop
}


////////////////////////////////////////////////////////////
// Initialization Function
// Initializes LCD controller, Timer 0, and
// A/D Converter
////////////////////////////////////////////////////////////
void Init(void)
{
    //// Initialize the LCD Controller ////////////////////
    SYSCON = 0x20;     // ENABLE ACCESS TO LCD/RTC/XRAM
    LCON = 0xC1;       // ENABLE ALL LCD OUTPUTS
                       // ENABLE LCD CONTROLLER
                       // USE SYSTEM CLOCK FOR LCD TIMER
    DAC0 = LCD_Volt;   // SET VALUE OF U3
    LCRL = 0x1B;       // SET VALUES FOR fLCD
    LCRH = 0xC1;       // START TIMER

    //// Initialize Timer 0 ////////////////////////////
    TMOD = 0x00;       // Set T0 to mode 0
    ET0 = 1;           // Enable T0 overflow interrupt

    //// Initialize the A/D Converter /////////////////////
    SYSCON |= 0x10;    // ENABLE ACCESS TO A/D P1ANA REG
```

```
    EAN4 = 0;           // EANBLE CHANNEL 4 FOR A/D
    SYSCON &= 0xEF;     // DISABLE ACCESS TO A/D P1ANA REG
    ADCON0 = 0x00;      // SINGLE CONVERSION
    ADCON1 = 0x44;      // fADC = fOSC/8
                        // SELECT CHANNEL 4 (P1.4)
    EADC = 1;           // ENABLE ADC INTERRUPT

    IP0 = 0x02;         // Set Interrupt Priority Levels
    IP1 = 0x01;         // A/D has higher priority than T0

}

//////////////////////////////////////////////////////
// Timer 0 Overflow ISR
// In this ISR the A/D conversion is started
//////////////////////////////////////////////////////
void T0ISR (void)  interrupt 1
{

    ADDATL = 0x01; // AD conversion is started by
                   // writing a dummy value in
                   // ADDATL
}


//////////////////////////////////////////////////////
// A/D Conversion Complete ISR
// In this ISR, 128 A/D converter readings are averaged.
// The A/D converter results are converted to Volts.
// The Volt value is converted into digits for display
// on the LCD.
//////////////////////////////////////////////////////
void ADC_ISR (void)    interrupt 8
{

    /////////// Variables ///////////////
    static unsigned int count = 0;        // Counter for Averaging A/D reads
    static unsigned int sum = 0;
    unsigned int avg;
    unsigned int volt;
    unsigned int temp;

    count ++;

    sum = sum + ((((int)ADDATH) << 1) | (ADDATL >> 7));  // Use only 9 digits

    if (count == 128)
    {
        TR0 = 0;                  // Stop the timer 0
        avg = sum >> 7;           // Calculate average voltage

        temp = avg * 91;          // convert to volts*100
        volt = temp / 93;

        printf("\n%03u %s", volt, "VOLT");
        DIG[0] = DIG[0] | decimal_point;      // Set the Decimal Point
```

```
        sum = 0x0000;              // reset the sum for the next average
        count = 0x00;
        TR0 = 1;                   // start timer 0
    }
    IADC = 0;                      // Clear A/D converter interrupt request flag
}
```

B LCD.c

```
//////////////////////////////////////////////////////
//  THIS FILE CONVERTS THE 16 DIG REGISTERS OF THE
//  C505L INTO A SINGLE ARRAY OF TYPE DigType (DEFINED
//  IN LCD.H).
//////////////////////////////////////////////////////

#include <LCD.h>
xdata DigType DIG[DispLen -1] _at_ 0xF3E0; // Define DIG registers as an array
```

C Putchar.c

```
/*************************************************************************/
/*  This file is part of the C51 Compiler package                        */
/*  Copyright (c) 1995-1996 Keil Software, Inc.                          */
/*************************************************************************/
/*                                                                       */
/*  PUTCHAR.C:  This routine is the general character output of C51.     */
/*                                                                       */
/*  To translate this file use C51 with the following invocation:        */
/*                                                                       */
/*     C51 PUTCHAR.C <memory model>                                      */
/*                                                                       */
/*  To link the modified PUTCHAR.OBJ file to your application use the    */
/*  following BL51 invocation:                                           */
/*                                                                       */
/*     BL51 <your object file list>, PUTCHAR.OBJ <controls>              */
/*                                                                        */
/*  THIS FILE HAS BEEN MODIFIED BY MIKE COPELAND OF INFINEON.             */
/*  THIS FILE DIRECTS OUTPUTS TO THE LCD CONTROLLER OF THE C505L.         */
/*  THE FILES "LCD.H" AND "LCD.C" ARE NEEDED.  SINCE THE DISPLAY IS       */
/*  ONLY 1 LINE THE "/n" WILL CLEAR THE DISPLAY.  THE "printf"            */
/*  FUNCTION OF THE KEIL COMPILER USES THIS FUNCTION.7/98                 */
/*                                                                       */
/*************************************************************************/


#include <LCD.h>

char putchar (char c)  {

DigType code ASCII_TABLE[128] =
               {let_X, let_X, let_X, let_X, let_X, let_X, let_X, let_X,
               let_X, let_X, let_X, let_X, let_X, let_X, let_X, let_X,
               let_X, let_X, let_X, let_X, let_X, let_X, let_X, let_X,
               let_X, let_X, let_X, let_X, let_X, let_X, let_X, let_X,
```

```
                space, let_X, let_X, let_X, dollar_sign, let_X, let_X,
apostrophe,
                let_X, let_X, asterisk, plus_sign, let_X, minus_sign,
            decimal_point, forward_slash,
                num_0, num_1, num_2, num_3, num_4, num_5, num_6, num_7,
                num_8, num_9, let_X, let_X, let_X, let_X, let_X, let_X,
                let_X, let_A, let_B, let_C, let_D, let_E, let_F, let_G,
                let_H, let_I, let_J, let_K, let_L, let_M, let_N, let_O,
                let_P, let_Q, let_R, let_S, let_T, let_U, let_V, let_W,
                let_X, let_Y, let_Z, let_X, back_slash, let_X, let_X, underscore,
                let_X, let_A, let_B, let_C, let_D, let_E, let_F, let_G,
                let_H, let_I, let_J, let_K, let_L, let_M, let_N, let_O,
                let_P, let_Q, let_R, let_S, let_T, let_U, let_V, let_W,
                let_X, let_Y, let_Z, let_X, or_sign, let_X, let_X, let_X};

unsigned char i;
unsigned long j;
static unsigned char place;


    if (c == '\n')
    {
        for (i = 0; i < DispLen; i++)     // clear the display
            DIG[i] = space;               // instead of CR
        place = 0;
    }
    else
    {
        if (place == DispLen)
        {
            for (j = 0; j<0x00010000; j++);   // delay if scrolling
            place = DispLen -1;
            for (i = 0; i<DispLen-1; i++) // scroll
                DIG[i] = DIG[i+1];
        }

        DIG[place] = ASCII_TABLE[c];

        place ++;

    }
    return c;
}
```

D LCD.h

```
////////////////////////////////////////////////////////////////////////
// This header file defines commonly used characters for the LCD used on
//  the KitCON 505L
// Change this header file to use a different LCD
////////////////////////////////////////////////////////////////////////

#define let_A  0x341E  // Letters
#define let_B  0x741E
#define let_C  0x5402
#define let_D  0x1421
#define let_F  0x340A
#define let_G  0x741A
#define let_H  0x341C
#define let_J  0x5014
#define let_K  0x9C08
#define let_P  0x340E
#define let_Q  0xD416
#define let_R  0xB40E
#define let_U  0x5414
#define let_W  0x9434
#define let_X  0x8821
#define let_Y  0x0881
#define let_Z  0x4822
#define let_S  0x641A
#define let_I  0x4182
#define let_E  0x740A
#define let_M  0x1C15
#define let_N  0x9415
#define let_V  0x1C20
#define let_O  0x5416
#define let_L  0x5400
#define let_T  0x0182
#define num_1  0x0014  // Numbers
#define num_2  0x700E
#define num_3  0x601E
#define num_4  0x241C
#define num_5  0x641A
#define num_6  0x741A
#define num_7  0x0016
#define num_8  0x741E
#define num_9  0x641E
#define num_0  0x5C36
#define decimal_point  0x0040  // Decimal Point
#define apostrophe 0x0200      // Apostrophe
#define space  0x0000          // No Segments active
#define dollar_sign    0x659A
#define asterisk   0xA9A9
#define plus_sign  0x2188
#define minus_sign 0x2008
#define forward_slash  0x0820
#define back_slash 0x8001
#define underscore 0x4000
#define or_sign    0x0180
#define smile  0x5811          // Smiley face
```

```
#define DispLen0x08           // Number of digits on the LCD
#define LCD_Volt    0xA0      // Voltage applied to the LCD

typedef unsigned int DigType; // Some displays will use char type
                              //  instead of int type

extern xdata DigType DIG[DispLen -1] ; // Define DIG registers as an array
```

E Reg505l.h

```
//===========================================================
//       SYMBOL Definition File for C505L
//===========================================================
//       Author     : R.Schmid, HL MC PD
//       Rev./ Date : 1.0 / 17.10.97
//===========================================================
//       Revision History
//       V1.0     : Original version
//===========================================================
//DEV     C505L
//
//       Special Function Registers
//
sfr     P0 =   0x80;
sfr     SP =   0x81;
sfr     DPL =  0x82;
sfr     DPH =  0x83;
sfr     WDTREL =   0x86;
sfr     PCON = 0x87;
sfr     TCON = 0x88;
#define PCON1   TCON
sfr     TMOD = 0x89;
sfr     TL0 =  0x8A;
sfr     TL1 =  0x8B;
sfr     TH0 =  0x8C;
sfr     TH1 =  0x8D;
sfr     P1 =   0x90;
#define P1ANA  P1
sfr     XPAGE =0x91;
sfr     DPSEL =0x92;
sfr     SCON = 0x98;
sfr     SBUF = 0x99;
sfr     P2 =   0xA0;
sfr     IEN0 = 0xA8;
sfr     IP0 =  0xA9;
sfr     SRELL =0xAA;
sfr     P3 =   0xB0;
sfr     SYSCON =   0xB1;
sfr     IEN1 = 0xB8;
sfr     IP1 =  0xB9;
sfr     SRELH =0xBA;
sfr     IRCON =0xC0;
sfr     CCEN = 0xC1;
sfr     CCL1 = 0xC2;
sfr     CCH1 = 0xC3;
sfr     CCL2 = 0xC4;
```

```
sfr     CCH2 = 0xC5;
sfr     T2CON =0xC8;
sfr     CRCL = 0xCA;
sfr     CRCH = 0xCB;
sfr     TL2 =  0xCC;
sfr     TH2 =  0xCD;
sfr     PSW =  0xD0;
sfr     ADCON0 =   0xD8;
sfr     ADDATH =   0xD9;
sfr     ADDATL =   0xDA;
sfr     ADCON1 =   0xDC;
sfr     ACC =  0xE0;
sfr     P4 =   0xE8;
sfr     B =0xF0;
sfr     P5 =   0xF8;
sfr     VR0 =  0xFC;
sfr     VR1 =  0xFD;
sfr     VR2 =  0xFE;
//
//       Bitaddressable Special Function Register Bits
//
//SFR   TCON & PCON1
sbit     IT0 = TCON^0;
sbit     IE0 = TCON^1;
sbit     IT1 = TCON^2;
sbit     IE1 = TCON^3;
sbit     TR0 = TCON^4;
sbit     TF0 = TCON^5;
sbit     TR1 = TCON^6;
sbit     TF1 = TCON^7;
#define  EPWD  TF1

//SFR    P1
sbit     INT4 =P1^1;
sbit     INT5 =P1^2;
sbit     T2EX =P1^5;
sbit     CLKOUT =  P1^6;
sbit     T2 =  P1^7;

//SFR    P1ANA
sbit     EAN0 =P1ANA^0;
sbit     EAN1 =P1ANA^1;
sbit     EAN2 =P1ANA^2;
sbit     EAN3 =P1ANA^3;
sbit     EAN4 =P1ANA^4;
sbit     EAN5 =P1ANA^5;
sbit     EAN6 =P1ANA^6;
sbit     EAN7 =P1ANA^7;

//SFR    SCON
sbit     RI =  SCON^0;
sbit     TI =  SCON^1;
sbit     RB8 = SCON^2;
sbit     TB8 = SCON^3;
sbit     REN = SCON^4;
sbit     SM2 = SCON^5;
sbit     SM1 = SCON^6;
```

```
sbit     SM0 = SCON^7;

//SFR    IEN0
sbit     EX0 = IEN0^0;
sbit     ET0 = IEN0^1;
sbit     EX1 = IEN0^2;
sbit     ET1 = IEN0^3;
sbit     ES =  IEN0^4;
sbit     ET2 = IEN0^5;
sbit     WDT = IEN0^6;
sbit     EA =  IEN0^7;

//SFR    P3
sbit     RXD = P3^0;
sbit     TXD = P3^1;
sbit     INT0 =P3^2;
sbit     INT1 =P3^3;
sbit     T0 =  P3^4;
sbit     T1 =  P3^5;
sbit     WR =  P3^6;
sbit     RD =  P3^7;

//SFR    IEN1
sbit     EADC =IEN1^0;
sbit     EX3 = IEN1^2;
sbit     EX4 = IEN1^3;
sbit     EX5 = IEN1^4;
sbit     EX6 = IEN1^5;
sbit     SWDT =IEN1^6;
sbit     EXEN2 =   IEN1^7;

//SFR    IRCON
sbit     IADC =IRCON^0;
sbit     IEX3 =IRCON^2;
sbit     IEX4 =IRCON^3;
sbit     IEX5 =IRCON^4;
sbit     IEX6 =IRCON^5;
sbit     TF2 = IRCON^6;
sbit     EXF2 =IRCON^7;

//SFR    T2CON
sbit     T2I0 =T2CON^0;
sbit     T2I1 =T2CON^1;
sbit     T2CM =T2CON^2;
sbit     T2R0 =T2CON^3;
sbit     T2R1 =T2CON^4;
sbit     I3FR =T2CON^6;
sbit     T2PS =T2CON^7;

//SFR    PSW
sbit     P =   PSW^0;
sbit     F1 =  PSW^1;
sbit     OV =  PSW^2;
sbit     RS0 = PSW^3;
sbit     RS1 = PSW^4;
sbit     F0 =  PSW^5;
sbit     AC =  PSW^6;
```

```
sbit     CY =  PSW^7;

//SFR    ADCON
sbit     MX0 = ADCON0^0;
sbit     MX1 = ADCON0^1;
sbit     MX2 = ADCON0^2;
sbit     ADM = ADCON0^3;
sbit     BSY = ADCON0^4;
sbit     CLK = ADCON0^6;
sbit     BD =  ADCON0^7;
//
//       Special Function Register of the LCD Module
//
#define    LCON(* ((unsigned char volatile xdata *)  0xF3DD))
#define    LCRL (* ((unsigned char volatile xdata *) 0xF3DE))
#define    LCRH (* ((unsigned char volatile xdata *) 0xF3DF))
#define    DAC0 (* ((unsigned char volatile xdata *)    0xF3DC))

//  These registers can be defined as a global array (LCD.c)
#define    DIG0 (* ((unsigned char volatile xdata *) 0xF3E0))
#define    DIG1 (* ((unsigned char volatile xdata *) 0xF3E1))
#define    DIG2 (* ((unsigned char volatile xdata *) 0xF3E2))
#define    DIG3 (* ((unsigned char volatile xdata *) 0xF3E3))
#define    DIG4 (* ((unsigned char volatile xdata *) 0xF3E4))
#define    DIG5 (* ((unsigned char volatile xdata *) 0xF3E5))
#define    DIG6 (* ((unsigned char volatile xdata *) 0xF3E6))
#define    DIG7 (* ((unsigned char volatile xdata *) 0xF3E7))
#define    DIG8 (* ((unsigned char volatile xdata *) 0xF3E8))
#define    DIG9 (* ((unsigned char volatile xdata *) 0xF3E9))
#define    DIG10 (* ((unsigned char volatile xdata *)    0xF3EA))
#define    DIG11 (* ((unsigned char volatile xdata *)    0xF3EB))
#define    DIG12 (* ((unsigned char volatile xdata *)    0xF3EC))
#define    DIG13 (* ((unsigned char volatile xdata *)    0xF3ED))
#define    DIG14 (* ((unsigned char volatile xdata *)    0xF3EE))
#define    DIG15 (* ((unsigned char volatile xdata *)    0xF3EF))
//
//
//       Special Function Registers of the RTC
//
#define    RTCON (* ((unsigned char volatile xdata *)    0xF3F0))
#define    RTCR0 (* ((unsigned char volatile xdata *)    0xF3F1))
#define    RTCR1 (* ((unsigned char volatile xdata *)    0xF3F2))
#define    RTCR2 (* ((unsigned char volatile xdata *)    0xF3F3))
#define    RTCR3 (* ((unsigned char volatile xdata *)    0xF3F4))
#define    RTCR4 (* ((unsigned char volatile xdata *)    0xF3F5))
#define    CLREG0 (* ((unsigned char volatile xdata *)   0xF3F6))
#define    CLREG1 (* ((unsigned char volatile xdata *)   0xF3F7))
#define    CLREG2 (* ((unsigned char volatile xdata *)   0xF3F8))
#define    CLREG3 (* ((unsigned char volatile xdata *)   0xF3F9))
#define    CLREG4 (* ((unsigned char volatile xdata *)   0xF3FA))
#define    RTINT0 (* ((unsigned char volatile xdata *)   0xF3FB))
#define    RTINT1 (* ((unsigned char volatile xdata *)   0xF3FC))
#define    RTINT2 (* ((unsigned char volatile xdata *)   0xF3FD))
#define    RTINT3 (* ((unsigned char volatile xdata *)   0xF3FE))
#define    RTINT4 (* ((unsigned char volatile xdata *)   0xF3FF))
```