```
//*****************************************************************************
*

//**

//**  Title:       1 Axis tracker Code.

//**  Description:   Program Description

//**              The program uses a real time clock for time keeping. The motion of the array is
controlled by 4 time settings, defining when moves occur. The move times are expressed as minutes
after midnight.

//**              For example one is called EightAM, and is set to 8*60= 480.

//**              The 1st move, at 8AM, is 45 degrees to the east. The second move is defined by a
setting, say NineThirtyAM, and it is the first of 24 moves to the west, ending at ThreePM.

//**              The array movement dwells for 2 hours, before returning to horizontal at FivePM.

//**              The hardware consists of 4 components. 1.) An Arduino UNO clone. 2.) An LCD display
and 5 button keypad from DFRobot. This board plugs directly into the UNO. 3.) A solid state relay board
from SainSmart.

//**              This board gets Vcc and Ground from the UNO. The board has 2 relays, which connect to
D2 and D3 on the LCD board. 4.) A Real Time Clock board.

//**              This board has a button Lithium Cell that keeps the clock running when power is absent. It
communicates to the UNO thru an I2C connection. The actuator motor (110 volts AC) is one that rotates
72 RPM, then thru a 9:1 reduction.

//**              It could be a DC actuator at 12 volts. In this case, the relay is a different style.

//**

//**              Main and Subroutine description:

//**              1. Main is a while 1 loop with a switch statement. The three elements of the switch
statement are Stop, Run, and Set Time. Choose the Up, Down, or Right pushbutton to select Run or Stop
or Set Time.

//**              2. Subroutines -

//**                Version - Displays software version upon startup.

//**                Instructions ? Displays operating instructions.
```

//** Timer 1 Overflow - The Real Time Clock maintains the time over the long haul. The hour and minute are updated daily. But over a 24 hour period, an interrupt is used to maintain the minute.

//** The interrupt is called Timer 1 Overflow. The target chip clock speed is 15,999,552 Hz. This is divided by 8. When a 16 bit register fills 1831 times, a minute variable is incremented.

//** But this method is in accurate to the tune of 2 minutes per 24 hours. Hence, the real time clock is used, as its accuracy is better than 1 minute per year.

//** Move1Axis - This switches 110 volts AC to the actuator motor. 2 relays are used. One sets the motor direction; the other applies power to the motor. There are 3 wires to the motor. See the schematic.

//** Button Press - an interrupt is called when a button press occurs. The routine sets a variable that determines where to be in the switch statement, or in the case of setting the time, increments the variables.

//** Currently this is hours and minutes. In the future, a 2 axis software package will use the day of the year.

//** Run - The Run subroutine is divided into 2 sections. Tracking, or stored horizontally when not tracking.

//** MinutesToHour_Minute - converts a global variable Minutes to Hours:Minutes.

//** PowerFromBattery - The real time clock can be set to where it gets its power from the button battery. This does that.

//** GetAndDisplayTime ? Obtains the time from the real time clock, at 12:01, each day.

//** SetTime - Does that to the real time clock. This occurs once or twice during the lifetime of the system, when the Lithium button cell needs to be replaced.

//**

//** A note about time keeping. In version one of the single axis controller, time was approximated by using a light sensor to count dark minutes during the night time.

//** Minutes, as measured from midnight, was simply Dark minutes / 2 plus a correction factor. Version 2 of the controller uses an accurate real time clock (RTC).

//**

//** The code has been composed with the aid of Flowcode ? www.matrixtls.com

//** Device: AVR.ATMEGA.ATMEGA328P

//**

```
//**  Generated by:  Flowcode v6.1.4.0
//********************************************************************************
*



#define MX_AVR
#define MX_CAL_AVR
#define MX_CLK_SPEED 15999522
#define FCP_NULL Unconnected_Port
#define MX_UART_ID
#define MX_UART_UCSRC

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <avr\io.h>
#include <avr\interrupt.h>
#include <avr\eeprom.h>
#include <avr\wdt.h>

//Configuration Start
//Configuration End

/*=========================================================================*\
  Use :Include the type definitions
\*=========================================================================*/
```

```
#include "C:\Program Files (x86)\Flowcode 6\CAL\internals.c"



/*============================================================================*\
  Use :panel
    :Variable declarations
    :Macro function declarations
\*============================================================================*/
#define FCV_FALSE (0)
#define FCV_EIGHTAM (480) // First move to east
#define FCV_FOURTHIRTYPM (990) // Last move to Horizontal
#define FCV_THREEFIFTEENPM (915) // Last Move to west
#define FCV_NINETHIRTYAM (570) // First move to west
#define FCV_TRUE (1)
MX_GLOBAL MX_SINT16 FCV_TIMERSCALECOUNTER = (0); // In tracking minutes, when the Timer 1 overflow occurs, this has to happen 1831 times before minute advances. See Timer 1 Overflow Interrupt.
MX_GLOBAL MX_SINT16 FCV_MENUPOSITION; // Variable depicting the position in the switch statement in Main. Set to Stop when booting.
MX_GLOBAL MX_SINT16 FCV_MINUTEFROMMIDNITE = (0); // Minute of day, beginning at midnight. 24 x 60 = 1440 minutes per 24 hours.
MX_GLOBAL MX_BOOL FCV_PERFORMONCE = (1);


void FCM_Run();
void FCM_GetTimeFromRTC();
void FCM_ButtonPress();
```

```
void FCM_Move1Axis(MX_UINT8 FCL_LR, MX_SINT16 FCL_MOVETIME);

void FCM_PowerFromBattery();

void FCM_SetTime();

void FCM_Timer1_Overflow();

void FCM_MinutesToHour_Minute();

void FCM_Instructions();

void FCM_Version();


/*=========================================================================*\

  Use :InjectorBase1

     :Variable declarations

     :Macro function declarations

\*=========================================================================*/


/*=========================================================================*\

  Use :dash_IO_flasher

     :Variable declarations

     :Macro function declarations

\*=========================================================================*/


/*=========================================================================*\

  Use :fcdhelper

     :Variable declarations

     :Macro function declarations

\*=========================================================================*/
```

```
/*===========================================================================*\
  Use :cal_i2c
     :Variable declarations
     :Macro function declarations
\*===========================================================================*/
#define MX_I2C_SDA_PORT_1 portc
#define MX_I2C_REF1
#define MX_I2C_1
#define MX_I2C_BMODE_1 (0)
#define MX_I2C_SDA_TRIS_1 trisc
#define MX_I2C_SCL_PIN_1 (5)
#define MX_I2C_SCL_PORT_1 portc
#define MX_I2C_STOPDEL_1 (1)
#define MX_I2C_SDA_PIN_1 (4)
#define MX_I2C_SCL_TRIS_1 trisc
#define MX_I2C_BAUD_1 (100000)
#define MX_I2C_CHANNEL_1 (1)



/*=---------------------------------------------------------------------=*\
  Use :cal_i2c
     :Supplementary defines
\*=---------------------------------------------------------------------=*/
#define MX_MI2C
```

```
MX_GLOBAL MX_UINT32 FCV_07da1_cal_i2c__CONSOLE;


void FC_CAL_I2C_Slave_Uninit_1();
void FCD_07da1_cal_i2c__Prv_TextConsole(MX_CHAR *FCL_STR, MX_UINT16 FCLsz_STR);
void FC_CAL_I2C_Master_Stop_1();
void FC_CAL_I2C_Slave_Init_1(MX_UINT8 FCL_ADDRESS, MX_UINT8 FCL_MASK);
void FC_CAL_I2C_Master_Uninit_1();
MX_UINT8 FC_CAL_I2C_Slave_Status_1();
MX_UINT8 FC_CAL_I2C_Slave_TxByte_1(MX_UINT8 FCL_DATA);
MX_UINT8 FC_CAL_I2C_Slave_RxByte_1(MX_UINT8 FCL_LAST);
void FC_CAL_I2C_Master_Init_1();
void FC_CAL_I2C_Master_Start_1();
MX_UINT8 FC_CAL_I2C_Master_TxByte_1(MX_UINT8 FCL_DATA);
void FC_CAL_I2C_Master_Restart_1();
MX_UINT8 FC_CAL_I2C_Master_RxByte_1(MX_UINT8 FCL_LAST);


/*==========================================================================*\
  Use :I2C_Master1
     :Variable declarations
     :Macro function declarations
\*==========================================================================*/
MX_UINT8 FCD_005f1_I2C_Master1__ReceiveByte(MX_UINT8 FCL_LAST);
void FCD_005f1_I2C_Master1__Restart();
void FCD_005f1_I2C_Master1__Stop();
MX_UINT8 FCD_005f1_I2C_Master1__ReceiveByteTransaction(MX_UINT8 FCL_DEVICE_ID, MX_UINT8
FCL_ADDRH, MX_UINT8 FCL_ADDRL);
```

```
MX_UINT8 FCD_005f1_I2C_Master1__TransmitByte(MX_UINT8 FCL_DATA);

void FCD_005f1_I2C_Master1__SendByteTransaction(MX_UINT8 FCL_DEVICE_ID, MX_UINT8
FCL_ADDRH, MX_UINT8 FCL_ADDRL, MX_UINT8 FCL_DATA);

void FCD_005f1_I2C_Master1__Start();

void FCD_005f1_I2C_Master1__Initialise();


/*============================================================================*\

  Use :fcdhelper

     :Variable declarations

     :Macro function declarations

\*============================================================================*/


/*============================================================================*\

  Use :cal_adc

     :Variable declarations

     :Macro function declarations

\*============================================================================*/

#define ADC_1_MX_ADC_ACTIME 40

#define MX_ADC_REF

#define ADC_1_MX_ADC_VREFVOL 500

#define MX_ADC_CHANNEL_0

#define MX_ADC_TYPE_2

#define ADC_1_MX_ADC_VREFOP 0

#define ADC_1_MX_ADC_CONVSP 3

#define MX_ADC_BITS_10

#define ADC_1_MX_ADC_CHANNEL 0
```

```
#define FCV_0aae1_cal_adc__FALSE (0)

#define FCV_0aae1_cal_adc__TRUE (1)


void FC_CAL_ADC_Disable();

void FC_CAL_ADC_Enable(MX_UINT8 FCL_CHANNEL, MX_UINT8 FCL_CONV_SPEED, MX_UINT8
FCL_VREF, MX_UINT8 FCL_T_CHARGE);

MX_UINT16 FC_CAL_ADC_Sample(MX_UINT8 FCL_SAMPLE_MODE);


/*=========================================================================*\

  Use :adc_base

    :Variable declarations

    :Macro function declarations

\*=========================================================================*/

MX_UINT16 FCD_08f41_adc_base__RawSampleInt();

MX_UINT8 FCD_08f41_adc_base__RawAverageByte(MX_UINT8 FCL_NUMSAMPLES, MX_UINT8
FCL_DELAYUS);

void FCD_08f41_adc_base__GetString(MX_CHAR *FCR_RETVAL, MX_UINT16 FCRsz_RETVAL);

MX_UINT8 FCD_08f41_adc_base__GetAverageByte(MX_UINT8 FCL_NUMSAMPLES, MX_UINT8
FCL_DELAYUS);

MX_SINT16 FCD_08f41_adc_base__RawAverageInt(MX_UINT8 FCL_NUMSAMPLES, MX_UINT8
FCL_DELAYUS);

MX_UINT16 FCD_08f41_adc_base__GetAverageInt(MX_UINT8 FCL_NUMSAMPLES, MX_UINT8
FCL_DELAYUS);

MX_FLOAT FCD_08f41_adc_base__GetVoltage();

void FCD_08f41_adc_base__RawEnable();

MX_UINT8 FCD_08f41_adc_base__RawSampleByte();
```

```
MX_UINT16 FCD_08f41_adc_base__GetInt();
void FCD_08f41_adc_base__RawDisable();
MX_UINT8 FCD_08f41_adc_base__GetByte();

/*====================================================================*\
  Use :LCD_PB
     :Variable declarations
     :Macro function declarations
\*====================================================================*/
#define FCD_0d511_LCD_PB__RawSampleInt FCD_08f41_adc_base__RawSampleInt
#define FCD_0d511_LCD_PB__RawAverageByte FCD_08f41_adc_base__RawAverageByte
#define FCD_0d511_LCD_PB__GetString FCD_08f41_adc_base__GetString
#define FCD_0d511_LCD_PB__GetAverageByte FCD_08f41_adc_base__GetAverageByte
#define FCD_0d511_LCD_PB__RawAverageInt FCD_08f41_adc_base__RawAverageInt
#define FCD_0d511_LCD_PB__GetAverageInt FCD_08f41_adc_base__GetAverageInt
#define FCD_0d511_LCD_PB__GetVoltage FCD_08f41_adc_base__GetVoltage
#define FCD_0d511_LCD_PB__RawEnable FCD_08f41_adc_base__RawEnable
#define FCD_0d511_LCD_PB__RawSampleByte FCD_08f41_adc_base__RawSampleByte
#define FCD_0d511_LCD_PB__GetInt FCD_08f41_adc_base__GetInt
#define FCD_0d511_LCD_PB__RawDisable FCD_08f41_adc_base__RawDisable
#define FCD_0d511_LCD_PB__GetByte FCD_08f41_adc_base__GetByte

/*====================================================================*\
  Use :ctrl_lcd
     :Variable declarations
```

```
    :Macro function declarations
\*=========================================================================*/


/*=========================================================================*\
  Use :LCD1
    :Variable declarations
    :Macro function declarations
\*=========================================================================*/
void FCD_04071_LCD1__Clear();
void FCD_04071_LCD1__PrintString(MX_CHAR *FCL_TEXT, MX_UINT16 FCLsz_TEXT);
void FCD_04071_LCD1__PrintAscii(MX_UINT8 FCL_CHARACTER);
void FCD_04071_LCD1__PrintNumber(MX_SINT16 FCL_NUMBER);
void FCD_04071_LCD1__RAMWrite(MX_UINT8 FCL_INDEX, MX_UINT8 FCL_D0, MX_UINT8 FCL_D1,
MX_UINT8 FCL_D2, MX_UINT8 FCL_D3, MX_UINT8 FCL_D4, MX_UINT8 FCL_D5, MX_UINT8 FCL_D6,
MX_UINT8 FCL_D7);
void FCD_04071_LCD1__ClearLine(MX_UINT8 FCL_LINE);
void FCD_04071_LCD1__Cursor(MX_UINT8 FCL_X, MX_UINT8 FCL_Y);
void FCD_04071_LCD1__Command(MX_UINT8 FCL_INSTRUCTION);
void FCD_04071_LCD1__PrintFormattedNumber(MX_UINT32 FCL_NUMBER, MX_BOOL FCL_FORMAT);
void FCD_04071_LCD1__ScrollDisplay(MX_UINT8 FCL_POSITION, MX_UINT8 FCL_DIRECTION);
void FCD_04071_LCD1__RawSend(MX_UINT8 FCL_DATA, MX_BOOL FCL_TYPE);
void FCD_04071_LCD1__Start();


/*=========================================================================*\
  Use :Include the chip adaption layer
\*=========================================================================*/
```

```
#include "C:\Program Files (x86)\Flowcode 6\CAL\includes.c"



/*============================================================================*\
  Use :InjectorBase1
     :Macro implementations
\*============================================================================*/

/*============================================================================*\
  Use :dash_IO_flasher
     :Macro implementations
\*============================================================================*/

/*============================================================================*\
  Use :fcdhelper
     :Macro implementations
\*============================================================================*/

/*============================================================================*\
  Use :cal_i2c
     :Macro implementations
\*============================================================================*/
/*=---------------------------------------------------------------------=*\
  Use :Send text to the console
     :
```

```
      :Parameters for macro Prv_TextConsole:
      :  str[20] : MX_CHAR (by-ref)
\*=----------------------------------------------------------------------=*/
void FCD_07da1_cal_i2c__Prv_TextConsole(MX_CHAR *FCL_STR, MX_UINT16 FCLsz_STR)
{




}



/*=========================================================================*\
   Use :I2C_Master1
      :Macro implementations
\*=========================================================================*/
/*=----------------------------------------------------------------------=*\
   Use :Receives a byte from the I??C bus.
      :
      :Parameters for macro ReceiveByte:
      :  Last : Used to signify the last byte when streaming incoming data. 0=Not last byte, 1=Last Byte
      :
      :Returns : MX_UINT8
\*=----------------------------------------------------------------------=*/
MX_UINT8 FCD_005f1_I2C_Master1__ReceiveByte(MX_UINT8 FCL_LAST)
{
    //Local variable definitions
```

```
    MX_UINT8 FCR_RETVAL;



    FCR_RETVAL = FC_CAL_I2C_Master_RxByte_1(FCL_LAST);

    return (FCR_RETVAL);

}

/*=-------------------------------------------------------------------=*\
   Use :Outputs a restart condition onto the I??C bus.
\*=-------------------------------------------------------------------=*/
void FCD_005f1_I2C_Master1__Restart()
{

    FC_CAL_I2C_Master_Restart_1();

}

/*=-------------------------------------------------------------------=*\
   Use :Outputs a stop condition onto the I??C bus.
\*=-------------------------------------------------------------------=*/
void FCD_005f1_I2C_Master1__Stop()
{
```

```
   FC_CAL_I2C_Master_Stop_1();


}


/*=-----------------------------------------------------------------------=*\
```

Use :Function to perform a generic I2C read transaction. The 7-bit device ID is automatically shifted up by one bit to make room for the read/write bit. Returns the data from the location specified.

```
      :
      :Parameters for macro ReceiveByteTransaction:
      :  Device_ID : 7-bit Device Address ID
      :  AddrH : Internal Address High Byte
      :  AddrL : Internal Address Low Byte
      :
      :Returns : MX_UINT8
\*=-----------------------------------------------------------------------=*/
MX_UINT8 FCD_005f1_I2C_Master1__ReceiveByteTransaction(MX_UINT8 FCL_DEVICE_ID, MX_UINT8 FCL_ADDRH, MX_UINT8 FCL_ADDRL)
{
   //Local variable definitions
   MX_UINT8 FCR_RETVAL;



   FCL_DEVICE_ID = FCL_DEVICE_ID << 1;


   FC_CAL_I2C_Master_Start_1();
```

```
    FC_CAL_I2C_Master_TxByte_1(FCL_DEVICE_ID);

    FC_CAL_I2C_Master_TxByte_1(FCL_ADDRH);

    FC_CAL_I2C_Master_TxByte_1(FCL_ADDRL);

    FC_CAL_I2C_Master_Restart_1();

    FCL_DEVICE_ID = FCL_DEVICE_ID | 0x01;

    FC_CAL_I2C_Master_TxByte_1(FCL_DEVICE_ID);

    FCR_RETVAL = FC_CAL_I2C_Master_RxByte_1(0x01);

    FC_CAL_I2C_Master_Stop_1();

    return (FCR_RETVAL);

}

/*=-----------------------------------------------------------------------=*\
   Use :Sends a byte on the I??C bus. Returns the acknowledge if any.
       :0 represents that data was acknowledged and 1 represents no acknowledge was detected.
       :
       :Parameters for macro TransmitByte:
```

```
      :  Data : Data byte to send on the I??C bus.
      :
      :Returns : MX_UINT8
\*=----------------------------------------------------------------------=*/
MX_UINT8 FCD_005f1_I2C_Master1__TransmitByte(MX_UINT8 FCL_DATA)
{
   //Local variable definitions
   MX_UINT8 FCR_RETVAL;



   FCR_RETVAL = FC_CAL_I2C_Master_TxByte_1(FCL_DATA);


   return (FCR_RETVAL);


}


/*=----------------------------------------------------------------------=*\
  Use :Function to perform a generic I2C Write transaction. The 7-bit device ID is automatically shifted up by one bit to make room for the read/write bit.
      :
      :Parameters for macro SendByteTransaction:
      :  Device_ID : 7-bit Device Address ID
      :  AddrH : Internal Address High Byte
      :  AddrL : Internal Address Low Byte
      :  Data : Data Byte
\*=----------------------------------------------------------------------=*/
```

```
void FCD_005f1_I2C_Master1__SendByteTransaction(MX_UINT8 FCL_DEVICE_ID, MX_UINT8
FCL_ADDRH, MX_UINT8 FCL_ADDRL, MX_UINT8 FCL_DATA)

{


   FCL_DEVICE_ID = FCL_DEVICE_ID << 1;


   FC_CAL_I2C_Master_Start_1();


   FC_CAL_I2C_Master_TxByte_1(FCL_DEVICE_ID);


   FC_CAL_I2C_Master_TxByte_1(FCL_ADDRH);


   FC_CAL_I2C_Master_TxByte_1(FCL_ADDRL);


   FC_CAL_I2C_Master_TxByte_1(FCL_DATA);


   FC_CAL_I2C_Master_Stop_1();


}


/*=---------------------------------------------------------------------=*\

  Use :Outputs a start condition onto the I??C bus.

\*=---------------------------------------------------------------------=*/

void FCD_005f1_I2C_Master1__Start()

{
```

```
    FC_CAL_I2C_Master_Start_1();


}


/*=-----------------------------------------------------------------=*\
  Use :Enables the I??C hardware and performs some initialization.
      :Should be called at the start of the program or at least before any of the other I??C functions are called.
\*=-----------------------------------------------------------------=*/
void FCD_005f1_I2C_Master1__Initialise()
{


    FC_CAL_I2C_Master_Init_1();


}




/*=================================================================*\
  Use :fcdhelper
      :Macro implementations
\*=================================================================*/


/*=================================================================*\
  Use :cal_adc
      :Macro implementations
\*=================================================================*/
```

```
/*============================================================================*\
  Use :adc_base
     :Macro implementations
\*============================================================================*/
/*=---------------------------------------------------------------------=*\
  Use :Background call to read the ADC at full bit depth
     :Call Enable() first
     :
     :Returns : MX_UINT16
\*=---------------------------------------------------------------------=*/
MX_UINT16 FCD_08f41_adc_base__RawSampleInt()
{
   //Local variable definitions
   MX_UINT16 FCR_RETVAL;



   FCR_RETVAL = FC_CAL_ADC_Sample(1);


   return (FCR_RETVAL);


}


/*=---------------------------------------------------------------------=*\
  Use :Background call to read the ADC as a byte average sample over time
```

```
      :Call Enable() before this
      :
      :Parameters for macro RawAverageByte:
      :  NumSamples : MX_UINT8
      :  DelayUs : Number of micro seconds in between taking each sample
      :
      :Returns : MX_UINT8
\*=----------------------------------------------------------------------=*/
MX_UINT8 FCD_08f41_adc_base__RawAverageByte(MX_UINT8 FCL_NUMSAMPLES, MX_UINT8
FCL_DELAYUS)
{
   //Local variable definitions
   MX_UINT16 FCL_AVERAGE = (0x0);
   MX_UINT8 FCL_COUNT = (0x0);
   MX_UINT8 FCR_RETVAL;



   if (FCL_DELAYUS > 0)
   {


      while (FCL_COUNT < FCL_NUMSAMPLES)
      {


         FCL_AVERAGE = FCL_AVERAGE + FC_CAL_ADC_Sample(0);
         FCL_COUNT = FCL_COUNT + 1;
```

```
            FCI_DELAYBYTE_US(FCL_DELAYUS);



        }

    } else {

        while (FCL_COUNT < FCL_NUMSAMPLES)
        {

            FCL_AVERAGE = FCL_AVERAGE + FC_CAL_ADC_Sample(0);
            FCL_COUNT = FCL_COUNT + 1;



        }

    }

    FCR_RETVAL = FCL_AVERAGE / FCL_COUNT;

    return (FCR_RETVAL);

}

/*=----------------------------------------------------------------------=*\
```

```
   Use :Reads the ADC as a direct voltage and returns as as tring
       :
       :Returns : MX_CHAR
\*=----------------------------------------------------------------------=*/
void FCD_08f41_adc_base__GetString(MX_CHAR *FCR_RETVAL, MX_UINT16 FCRsz_RETVAL)
{
   //Local variable definitions
   MX_FLOAT FCL_SAMPLE;



   FCL_SAMPLE = FCD_08f41_adc_base__GetVoltage();


   FCI_FLOAT_TO_STRING(FCL_SAMPLE, FCV_PRECISION, FCR_RETVAL,20);


}


/*=----------------------------------------------------------------------=*\
   Use :Function call to read the ADC as a byte average sample over time
       :
       :Parameters for macro GetAverageByte:
       :  NumSamples : MX_UINT8
       :  DelayUs : Number of micro seconds in between taking each sample
       :
       :Returns : MX_UINT8
\*=----------------------------------------------------------------------=*/
```

```
MX_UINT8 FCD_08f41_adc_base__GetAverageByte(MX_UINT8 FCL_NUMSAMPLES, MX_UINT8
FCL_DELAYUS)

{

   //Local variable definitions

   MX_UINT8 FCR_RETVAL;



   FC_CAL_ADC_Enable(0, 3, 0, 40);


   FCR_RETVAL = FCD_08f41_adc_base__RawAverageByte(FCL_NUMSAMPLES, FCL_DELAYUS);


   FC_CAL_ADC_Disable();


   return (FCR_RETVAL);


}


/*=-----------------------------------------------------------------------=*\

  Use :Background call to read the ADC as a full width average sample over time

      :Call Enable() before this

      :

      :Parameters for macro RawAverageInt:

      :  NumSamples : MX_UINT8

      :  DelayUs : MX_UINT8

      :

      :Returns : MX_SINT16
```

```
\*=----------------------------------------------------------------------=*/

MX_SINT16 FCD_08f41_adc_base__RawAverageInt(MX_UINT8 FCL_NUMSAMPLES, MX_UINT8
FCL_DELAYUS)

{

   //Local variable definitions

   MX_UINT32 FCL_AVERAGE = (0x0);

   MX_UINT8 FCL_COUNT = (0x0);

   MX_SINT16 FCR_RETVAL;



   if (FCL_DELAYUS > 0)

   {


      while (FCL_COUNT < FCL_NUMSAMPLES)

      {


         FCL_AVERAGE = FCL_AVERAGE + FC_CAL_ADC_Sample(1);

         FCL_COUNT = FCL_COUNT + 1;


         FCI_DELAYBYTE_US(FCL_DELAYUS);



      }


   } else {
```

```
        while (FCL_COUNT < FCL_NUMSAMPLES)
        {

           FCL_AVERAGE = FCL_AVERAGE + FC_CAL_ADC_Sample(1);
           FCL_COUNT = FCL_COUNT + 1;



        }

    }

    FCR_RETVAL = FCL_AVERAGE / FCL_COUNT;

    return (FCR_RETVAL);

}

/*=-------------------------------------------------------------------=*\
   Use :Function call to read the ADC as a full width average sample over time
       :
       :Parameters for macro GetAverageInt:
       :  NumSamples : MX_UINT8
       :  DelayUs : Number of micro seconds in between taking each sample
       :
       :Returns : MX_UINT16
```

```
\*=----------------------------------------------------------------------=*/

MX_UINT16 FCD_08f41_adc_base__GetAverageInt(MX_UINT8 FCL_NUMSAMPLES, MX_UINT8
FCL_DELAYUS)

{

   //Local variable definitions

   MX_UINT16 FCR_RETVAL;



   FC_CAL_ADC_Enable(0, 3, 0, 40);


   FCR_RETVAL = FCD_08f41_adc_base__RawAverageInt(FCL_NUMSAMPLES, FCL_DELAYUS);


   FC_CAL_ADC_Disable();


   return (FCR_RETVAL);


}


/*=----------------------------------------------------------------------=*\

  Use :Reads the ADC as a direct voltage

     :

     :Returns : MX_FLOAT

\*=----------------------------------------------------------------------=*/

MX_FLOAT FCD_08f41_adc_base__GetVoltage()

{

   //Local variable definitions
```

```
    MX_UINT16 FCL_SAMPLE;
    MX_FLOAT FCR_RETVAL;



    FC_CAL_ADC_Enable(0, 3, 0, 40);


    FCL_SAMPLE = FC_CAL_ADC_Sample(1);


    FCR_RETVAL = flt_mul(flt_fromi(FCL_SAMPLE), 0.004883);


    FC_CAL_ADC_Disable();


    return (FCR_RETVAL);


}


/*=-----------------------------------------------------------------------=*\
   Use :Enables and configures the ADC channel to be an analogue input.
      :Only one ADC channel can be enabled at a time. Any RAW functions will reference the last enabled channel only.
\*=-----------------------------------------------------------------------=*/
void FCD_08f41_adc_base__RawEnable()
{


    FC_CAL_ADC_Enable(0, 3, 0, 40);
```

```
}

/*=-----------------------------------------------------------------=*\
  Use :Background call to read the ADC as a byte
     :Call Enable() before this
     :
     :Returns : MX_UINT8
\*=-----------------------------------------------------------------=*/
MX_UINT8 FCD_08f41_adc_base__RawSampleByte()
{
   //Local variable definitions
   MX_UINT8 FCR_RETVAL;



   FCR_RETVAL = FC_CAL_ADC_Sample(0);


   return (FCR_RETVAL);


}

/*=-----------------------------------------------------------------=*\
  Use :Blocking call to read the ADC at full bit depth
     :
     :Returns : MX_UINT16
\*=-----------------------------------------------------------------=*/
```

```
MX_UINT16 FCD_08f41_adc_base__GetInt()
{
   //Local variable definitions
   MX_UINT16 FCR_RETVAL;



   FC_CAL_ADC_Enable(0, 3, 0, 40);


   FCR_RETVAL = FC_CAL_ADC_Sample(1);


   FC_CAL_ADC_Disable();


   return (FCR_RETVAL);


}


/*=-----------------------------------------------------------------=*\
  Use :Disables the previously enabled ADC channel and converts back to digital mode.
\*=-----------------------------------------------------------------=*/
void FCD_08f41_adc_base__RawDisable()
{


   FC_CAL_ADC_Disable();


}
```

```
/*=-----------------------------------------------------------------=*\
  Use :Blocking call to read the ADC as a byte
     :
     :Returns : MX_UINT8
\*=-----------------------------------------------------------------=*/
MX_UINT8 FCD_08f41_adc_base__GetByte()
{
  //Local variable definitions
  MX_UINT8 FCR_RETVAL;


  FC_CAL_ADC_Enable(0, 3, 0, 40);

  FCR_RETVAL = FC_CAL_ADC_Sample(0);

  FC_CAL_ADC_Disable();

  return (FCR_RETVAL);

}


/*=================================================================*\
  Use :LCD_PB
```

```
    :Macro implementations
\*=========================================================================*/

/*=========================================================================*\
  Use :ctrl_lcd
    :Macro implementations
\*=========================================================================*/

/*=========================================================================*\
  Use :LCD1
    :Macro implementations
\*=========================================================================*/
/*=-----------------------------------------------------------------------=*\
  Use :Clears the entire contents of the display.
\*=-----------------------------------------------------------------------=*/
void FCD_04071_LCD1__Clear()
{




   FCD_04071_LCD1__RawSend(0x01, 0);


   FCI_DELAYBYTE_MS(2);
```

```
   FCD_04071_LCD1__RawSend(0x02, 0);


   FCI_DELAYBYTE_MS(2);


}


/*=-----------------------------------------------------------------------=*\
   Use :Breaks down a string of text and sends it to the LCD via the private RawSend(byte, mask) macro
      :
      :Parameters for macro PrintString:
      :  Text[20] : Enter the text or variable to print to the LCD
\*=-----------------------------------------------------------------------=*/
void FCD_04071_LCD1__PrintString(MX_CHAR *FCL_TEXT, MX_UINT16 FCLsz_TEXT)
{
   //Local variable definitions
   MX_UINT8 FCL_IDX = (0x0);
   MX_UINT8 FCL_COUNT;




   FCL_COUNT = FCI_GETLENGTH(FCL_TEXT, FCLsz_TEXT);


   while (FCL_IDX < FCL_COUNT)
   {
```

```
      FCD_04071_LCD1__RawSend(FCL_TEXT[FCL_IDX], 0x10);

      FCL_IDX = FCL_IDX + 1;


   }


}

/*=-----------------------------------------------------------------------=*\
   Use :Takes the ascii value for a character and prints the character
      :
      :Parameters for macro PrintAscii:
      :  character : Holds an ascii value.
\*=-----------------------------------------------------------------------=*/
void FCD_04071_LCD1__PrintAscii(MX_UINT8 FCL_CHARACTER)
{

   FCD_04071_LCD1__RawSend(FCL_CHARACTER, 0x10);



}
```

```
/*=-----------------------------------------------------------------------=*\
  Use :Based on v5 macro, will allow you to print a number. This is limited to a signed-INT, -32768 to
32767
     :
     :Parameters for macro PrintNumber:
     :  Number : Enter the number or variable to print to the LCD
\*=-----------------------------------------------------------------------=*/
void FCD_04071_LCD1__PrintNumber(MX_SINT16 FCL_NUMBER)
{
   //Local variable definitions
   #define FCLsz_S 10
   MX_CHAR FCL_S[FCLsz_S];



   FCI_TOSTRING(FCL_NUMBER, FCL_S,10);


   FCD_04071_LCD1__PrintString(FCL_S, FCLsz_S);


   //Local variable definitions
   #undef FCLsz_S
}


/*=-----------------------------------------------------------------------=*\
  Use :Modifies the internal memory of the LCD to allow for up to 8 customised characters to be created
and stored in the device memory
     :
```

```
      :Parameters for macro RAMWrite:
      :  Index : Values 0 to 7
      :  d0 : MX_UINT8
      :  d1 : MX_UINT8
      :  d2 : MX_UINT8
      :  d3 : MX_UINT8
      :  d4 : MX_UINT8
      :  d5 : MX_UINT8
      :  d6 : MX_UINT8
      :  d7 : MX_UINT8
\*=----------------------------------------------------------------------=*/
void FCD_04071_LCD1__RAMWrite(MX_UINT8 FCL_INDEX, MX_UINT8 FCL_D0, MX_UINT8 FCL_D1,
MX_UINT8 FCL_D2, MX_UINT8 FCL_D3, MX_UINT8 FCL_D4, MX_UINT8 FCL_D5, MX_UINT8 FCL_D6,
MX_UINT8 FCL_D7)
{

   FCD_04071_LCD1__RawSend(64 + (FCL_INDEX << 3), 0);

   FCI_DELAYBYTE_MS(2);

   FCD_04071_LCD1__RawSend(FCL_D0, 0x10);

   FCD_04071_LCD1__RawSend(FCL_D1, 0x10);

   FCD_04071_LCD1__RawSend(FCL_D2, 0x10);
```

```
   FCD_04071_LCD1__RawSend(FCL_D3, 0x10);

   FCD_04071_LCD1__RawSend(FCL_D4, 0x10);

   FCD_04071_LCD1__RawSend(FCL_D5, 0x10);

   FCD_04071_LCD1__RawSend(FCL_D6, 0x10);

   FCD_04071_LCD1__RawSend(FCL_D7, 0x10);

   FCD_04071_LCD1__Clear();

}

/*=-----------------------------------------------------------------------=*\
  Use :Clears a single line on the display and then moves the cursor to the start of the line to allow you to start populating the line with data.
      :
      :Parameters for macro ClearLine:
      :  Line : The line to clear, zero being the first (top) line of the display
\*=-----------------------------------------------------------------------=*/
void FCD_04071_LCD1__ClearLine(MX_UINT8 FCL_LINE)
{
   //Local variable definitions
   MX_UINT8 FCL_X;
```

```
if (FCL_LINE < 2)
{
   FCD_04071_LCD1__Cursor(0, FCL_LINE);
   FCL_X = 0;
   while (FCL_X < 16)
   {
      FCD_04071_LCD1__RawSend(' ', 0x10);
      FCL_X = FCL_X + 1;
   }
   FCD_04071_LCD1__Cursor(0, FCL_LINE);
// } else {
}
```

```
}

/*=-----------------------------------------------------------------------=*\
  Use :Moves the cursor on the LCD Display
     :
     :Parameters for macro Cursor:
     :  x : Set the cursor position in the X plane, 0 is the left most cell
     :  y : Set the cursor position in the Y plane, 0 is the top most cell
\*=-----------------------------------------------------------------------=*/
void FCD_04071_LCD1__Cursor(MX_UINT8 FCL_X, MX_UINT8 FCL_Y)
{



  #if (0) // 2 == 1

  //Code has been optimised out by the pre-processor
  // #else

  #endif

  #if (1) // 2 == 2

    if (FCL_Y == 0)
    {
```

```
        FCL_Y = 0x80;

    } else {

        FCL_Y = 0xC0;

    }

// #else

//Code has been optimised out by the pre-processor

#endif

#if (0) // 2 == 4

//Code has been optimised out by the pre-processor

// #else

#endif

FCD_04071_LCD1__RawSend(FCL_Y + FCL_X, 0);

FCI_DELAYBYTE_MS(2);
```

```
}

/*=-----------------------------------------------------------------=*\
```

Use :Use this method/macro to send a specific command to the LCD. Refer to the Matrix Multimedia EB006 datasheet for a list of supported instructions. For Non-Matrix LCD's refer to the manufacturers datasheet.

:

:Parameters for macro Command:

: instruction : Send a defined command to the LCD Screen. See datasheet for supported commands.

```
\*=-----------------------------------------------------------------=*/

void FCD_04071_LCD1__Command(MX_UINT8 FCL_INSTRUCTION)

{


   FCD_04071_LCD1__RawSend(FCL_INSTRUCTION, 0);


   FCI_DELAYBYTE_MS(2);


}


/*=-----------------------------------------------------------------=*\
```

Use :Will allow you to print a number up to 32-bits with signed or unsigned formatting.

:Signed = -2147483648 to 2147483647

:Unsigned = 0 to 4294967295

:

:Parameters for macro PrintFormattedNumber:

: Number : Enter the number or variable to print to the LCD

```
      :  Format : 0=Signed, 1=Unsigned
\*=----------------------------------------------------------------------=*/
void FCD_04071_LCD1__PrintFormattedNumber(MX_UINT32 FCL_NUMBER, MX_BOOL FCL_FORMAT)
{
   //Local variable definitions
   #define FCLsz_S 15
   MX_CHAR FCL_S[FCLsz_S];



   if (FCL_FORMAT == 1)
   {

      FCI_UTOS32(FCL_NUMBER, FCL_S,15);

   } else {

      FCI_ITOS32((MX_SINT32)(FCL_NUMBER), FCL_S,15);

   }

   FCD_04071_LCD1__PrintString(FCL_S, FCLsz_S);

   //Local variable definitions
   #undef FCLsz_S
}
```

```
/*=-----------------------------------------------------------------------=*\
   Use :Scrolls the display left or right by a number of given positions.
      :
      :Parameters for macro ScrollDisplay:
      :  Position : Holds the number of positions to shift the display
      :  direction : 0 = left, 1 = right
\*=-----------------------------------------------------------------------=*/
void FCD_04071_LCD1__ScrollDisplay(MX_UINT8 FCL_POSITION, MX_UINT8 FCL_DIRECTION)
{
   //Local variable definitions
   MX_UINT8 FCL_CMD = (0x0);



   FCL_CMD = 0;


   switch (FCL_DIRECTION)
   {
      case 0:
      {
         FCL_CMD = 0x18;



         break;
      }
```

```
case 'l':
{
   FCL_CMD = 0x18;



   break;
}
case 'L':
{
   FCL_CMD = 0x18;



   break;
}
case 1:
{
   FCL_CMD = 0x1C;



   break;
}
case 'r':
{
   FCL_CMD = 0x1C;
```

```
      break;
    }
    case 'R':
    {
      FCL_CMD = 0x1C;



      break;
    }
    // default:

}

if (FCL_CMD != 0 && FCL_POSITION != 0)
{

    while (FCL_POSITION != 0)
    {

      FCD_04071_LCD1__RawSend(FCL_CMD, 0);

      FCL_POSITION = FCL_POSITION - 1;
```

```
        }


    // } else {


    }


}


/*=-----------------------------------------------------------------------=*\
   Use :Sends data to the LCD display
       :
       :Parameters for macro RawSend:
       :  data : The data byte to send to the LCD
       :  type : A boolean to indicate command type: true to write data, false to write a command
\*=-----------------------------------------------------------------------=*/
void FCD_04071_LCD1__RawSend(MX_UINT8 FCL_DATA, MX_BOOL FCL_TYPE)
{
    //Local variable definitions
    MX_UINT8 FCL_NIBBLE;



    //Comment:
    //Output upper nibble of the byte


    #if (1) // 0 == 0
```

```
    FCP_SET(B, D, 0x10, 0x4, 0);
    FCP_SET(B, D, 0x20, 0x5, 0);
    FCP_SET(B, D, 0x40, 0x6, 0);
    FCP_SET(B, D, 0x80, 0x7, 0);
    FCP_SET(B, B, 0x1, 0x0, 0);
    FCP_SET(B, B, 0x2, 0x1, 0);

    #if (0)

    //Code has been optimised out by the pre-processor
    // #else

    #endif

    FCL_NIBBLE = (FCL_DATA >> 4);
    FCP_SET(B, D, 0x10, 0x4, (FCL_NIBBLE & 0x01));
    FCL_NIBBLE = FCL_NIBBLE >> 1;
    FCP_SET(B, D, 0x20, 0x5, (FCL_NIBBLE & 0x01));
    FCL_NIBBLE = FCL_NIBBLE >> 1;
    FCP_SET(B, D, 0x40, 0x6, (FCL_NIBBLE & 0x01));
    FCL_NIBBLE = FCL_NIBBLE >> 1;
    FCP_SET(B, D, 0x80, 0x7, (FCL_NIBBLE & 0x01));

// #else
```

```
//Code has been optimised out by the pre-processor
#endif

//Comment:
//Output byte to pins

#if (0) // 0 == 1

//Code has been optimised out by the pre-processor
// #else

#endif

//Comment:
//Output byte to port

#if (0) // 0 == 2

//Code has been optimised out by the pre-processor
// #else

#endif

if (FCL_TYPE)
```

```
{

    FCP_SET(B, B, 0x1, 0x0, 1);

// } else {

}

FCI_DELAYBYTE_US(100);

//Comment:
//Set Enable high, pause then set low
//to acknowledge the data has been
//submitted.

FCP_SET(B, B, 0x2, 0x1, 1);

FCI_DELAYBYTE_US(100);

FCP_SET(B, B, 0x2, 0x1, 0);

FCI_DELAYBYTE_US(100);

#if (1) // 0 == 0
```

```
FCP_SET(B, D, 0x10, 0x4, 0);

FCP_SET(B, D, 0x20, 0x5, 0);

FCP_SET(B, D, 0x40, 0x6, 0);

FCP_SET(B, D, 0x80, 0x7, 0);

FCP_SET(B, B, 0x1, 0x0, 0);


FCL_NIBBLE = (FCL_DATA & 0xf);

FCP_SET(B, D, 0x10, 0x4, (FCL_NIBBLE & 0x01));

FCL_NIBBLE = FCL_NIBBLE >> 1;

FCP_SET(B, D, 0x20, 0x5, (FCL_NIBBLE & 0x01));

FCL_NIBBLE = FCL_NIBBLE >> 1;

FCP_SET(B, D, 0x40, 0x6, (FCL_NIBBLE & 0x01));

FCL_NIBBLE = FCL_NIBBLE >> 1;

FCP_SET(B, D, 0x80, 0x7, (FCL_NIBBLE & 0x01));


if (FCL_TYPE)

{


   FCP_SET(B, B, 0x1, 0x0, 1);


// } else {


}


FCl_DELAYBYTE_US(100);
```

```
        FCP_SET(B, B, 0x2, 0x1, 1);

        FCI_DELAYBYTE_US(100);

        FCP_SET(B, B, 0x2, 0x1, 0);

        FCI_DELAYBYTE_US(100);

    // #else

    //Code has been optimised out by the pre-processor
    #endif

}

/*=----------------------------------------------------------------------=*\
   Use :Startup routine required by the hardware device.
       :Automatically clears the display after initialising.
\*=----------------------------------------------------------------------=*/
void FCD_04071_LCD1__Start()
{
```

```
#if (1) // 0 == 0

   FCP_SET(B, D, 0x10, 0x4, 0);

   FCP_SET(B, D, 0x20, 0x5, 0);

   FCP_SET(B, D, 0x40, 0x6, 0);

   FCP_SET(B, D, 0x80, 0x7, 0);

   FCP_SET(B, B, 0x1, 0x0, 0);

   FCP_SET(B, B, 0x2, 0x1, 0);

// #else

//Code has been optimised out by the pre-processor

#endif

#if (0) // 0 == 1

//Code has been optimised out by the pre-processor

// #else

#endif

#if (0) // 0 == 2

//Code has been optimised out by the pre-processor

// #else
```

```
#endif

#if (0)

//Code has been optimised out by the pre-processor
// #else

#endif

FCI_DELAYBYTE_MS(12);

FCD_04071_LCD1__RawSend(0x33, 0);

FCI_DELAYBYTE_MS(2);

FCD_04071_LCD1__RawSend(0x33, 0);

FCI_DELAYBYTE_MS(2);

#if (0) // 0 > 0

//Code has been optimised out by the pre-processor
#else
```

```
    FCD_04071_LCD1__RawSend(0x32, 0);

    FCI_DELAYBYTE_MS(2);

    FCD_04071_LCD1__RawSend(0x2c, 0);

#endif

FCI_DELAYBYTE_MS(2);

FCD_04071_LCD1__RawSend(0x06, 0);

FCI_DELAYBYTE_MS(2);

FCD_04071_LCD1__RawSend(0x0c, 0);

FCI_DELAYBYTE_MS(2);

FCD_04071_LCD1__RawSend(0x01, 0);

FCI_DELAYBYTE_MS(2);

FCD_04071_LCD1__RawSend(0x02, 0);

FCI_DELAYBYTE_MS(2);
```

```
   FCD_04071_LCD1__Clear();


}



/*=========================================================================*\
  Use :panel
     :Macro implementations
\*=========================================================================*/
/*=-----------------------------------------------------------------=*\
  Use :
\*=-----------------------------------------------------------------=*/
void FCM_Run()
{


   // Decision
   // Decision: (MinuteFromMidnite >= EightAM) AND (MinuteFromMidnite <= FourThirtyPM)?
   if ((FCV_MINUTEFROMMIDNITE >= FCV_EIGHTAM) & (FCV_MINUTEFROMMIDNITE <=
FCV_FOURTHIRTYPM))
   {


      //Comment:
      //Tracking


      // "Track to "
```

```
// Call Component Macro: LCD1::PrintString("Track to Min")
FCD_04071_LCD1__PrintString("Track to Min", 13);

// Stop Time
// Call Component Macro: LCD1::PrintNumber(FourThirtyPM)
FCD_04071_LCD1__PrintNumber(FCV_FOURTHIRTYPM);

// Cursor 0,1
// Call Component Macro: LCD1::Cursor(0, 1)
FCD_04071_LCD1__Cursor(0, 1);

// Convert X:XX
// Call Macro: MinutesToHour_Minute()
FCM_MinutesToHour_Minute();

// Delay
// Delay: 5 s
FCI_DELAYBYTE_S(5);

// CLS
// Call Component Macro: LCD1::Clear()
FCD_04071_LCD1__Clear();

// Delay
// Delay: 1 s
```

```
FCI_DELAYBYTE_S(1);


// Decision

// Decision: MinuteFromMidnite = EightAM // 8 AM?

if (FCV_MINUTEFROMMIDNITE == FCV_EIGHTAM)

{


   // Initial Move

   // Call Macro: Move1Axis(2, 600)

   FCM_Move1Axis(2, 600);


// } else {


}


//Comment:

//This happens 24 times.

//Keep this relationship-> 24 * 50 = 1200

//If 50 became 60, then the initial move should be 60. (24*60 = 720)


// Decision

// Decision: MinuteFromMidnite MOD 15 = 0 && MinuteFromMidnite >= NineThirtyAM &&
MinuteFromMidnite <= ThreeFifteenPM && PerformOnce?

if (FCV_MINUTEFROMMIDNITE % 15 == 0 && FCV_MINUTEFROMMIDNITE >= FCV_NINETHIRTYAM
&& FCV_MINUTEFROMMIDNITE <= FCV_THREEFIFTEENPM && FCV_PERFORMONCE)

{
```

```
    // Move One Delta
    // Call Macro: Move1Axis(1, 50)
    FCM_Move1Axis(1, 50);

    // Calculation
    // Calculation:
    //  PerformOnce = 0
    FCV_PERFORMONCE = 0;

// } else {

}

// Decision
// Decision: MinuteFromMidnite MOD 15 = 1?
if (FCV_MINUTEFROMMIDNITE % 15 == 1)
{

    // Calculation
    // Calculation:
    //  PerformOnce = 1
    FCV_PERFORMONCE = 1;

// } else {
```

```
    }

    // Decision
    // Decision: MinuteFromMidnite = FourThirtyPM // 4:30 PM?
    if (FCV_MINUTEFROMMIDNITE == FCV_FOURTHIRTYPM)
    {

       // InitialMove
       // Call Macro: Move1Axis(2, 600)
       FCM_Move1Axis(2, 600);

    // } else {

    }

} else {

    //Comment:
    //"Rest-Horizontal"

    // "Rest to Min "
    // Call Component Macro: LCD1::PrintString("Rest to Min ")
    FCD_04071_LCD1__PrintString("Rest to Min ", 13);
```

```
    // Start Time
    // Call Component Macro: LCD1::PrintNumber(EightAM)
    FCD_04071_LCD1__PrintNumber(FCV_EIGHTAM);

    // Cursor 0,1
    // Call Component Macro: LCD1::Cursor(0, 1)
    FCD_04071_LCD1__Cursor(0, 1);

    // Convert X:XX
    // Call Macro: MinutesToHour_Minute()
    FCM_MinutesToHour_Minute();

    // Delay
    // Delay: 5 s
    FCI_DELAYBYTE_S(5);

    // CLS
    // Call Component Macro: LCD1::Clear()
    FCD_04071_LCD1__Clear();

    // Delay
    // Delay: 1 s
    FCI_DELAYBYTE_S(1);

}
```

```
}

/*=-----------------------------------------------------------------------=*\
  Use :
\*=-----------------------------------------------------------------------=*/
void FCM_GetTimeFromRTC()
{
   //Local variable definitions
   MX_UINT8 FCL_HOURFROMRTC;
   MX_UINT8 FCL_SECONDFROMRTC;
   MX_UINT8 FCL_MINUTEFROMRTC;



   //Comment:
   //Start
   //TransmitByte(208) - Write Mode
   //TransmitByte(1st Register address)
   //TransmitByte(value 1st register )
   //Restart
   //TransmitByte(209) - Read Mode
   //RecieveByte(value 1st register ) (Last Byte = 0)
   //RecieveByte(value 2nd register ) (Last Byte = 0)
   //RecieveByte(value final register )  (Last Byte = 1)
   //Stop
```

```
// I2C Start
// Call Component Macro: I2C_Master1::Start()
FCD_005f1_I2C_Master1__Start();

// Write mode
// Call Component Macro: I2C_Master1::TransmitByte(208)
FCD_005f1_I2C_Master1__TransmitByte(208);

// Transmit 0
// Call Component Macro: I2C_Master1::TransmitByte(0)
FCD_005f1_I2C_Master1__TransmitByte(0);

#if 0 // Disabled code
// I2C Stop
// Call Component Macro: I2C_Master1::Stop()
FCD_005f1_I2C_Master1__Stop();

#endif // #if 0: Disabled code
// Restart
// Call Component Macro: I2C_Master1::Restart()
FCD_005f1_I2C_Master1__Restart();

// Read Mode
// Call Component Macro: I2C_Master1::TransmitByte(209)
```

```
FCD_005f1_I2C_Master1__TransmitByte(209);


// Read Seconds
// Call Component Macro: .SecondFromRTC=I2C_Master1::ReceiveByte(0)
FCL_SECONDFROMRTC = FCD_005f1_I2C_Master1__ReceiveByte(0);


// Read Minutes
// Call Component Macro: .MinuteFromRTC=I2C_Master1::ReceiveByte(0)
FCL_MINUTEFROMRTC = FCD_005f1_I2C_Master1__ReceiveByte(0);


// Read Hour
// Call Component Macro: .HourFromRTC=I2C_Master1::ReceiveByte(0)
FCL_HOURFROMRTC = FCD_005f1_I2C_Master1__ReceiveByte(0);


// Convert: BCD to DEC
// Calculation:
//  .SecondFromRTC = (.SecondFromRTC >> 4) * 10 + (.SecondFromRTC & 15)
//  .HourFromRTC = (.HourFromRTC >> 4) * 10 + (.HourFromRTC & 15)
//  .MinutefromRTC = (.MinuteFromRTC >> 4) * 10 + (.MinuteFromRTC & 15)
//  MinuteFromMidnite = .HourFromRTC * 60 + .MinuteFromRTC
FCL_SECONDFROMRTC = (FCL_SECONDFROMRTC >> 4) * 10 + (FCL_SECONDFROMRTC & 15);
FCL_HOURFROMRTC = (FCL_HOURFROMRTC >> 4) * 10 + (FCL_HOURFROMRTC & 15);
FCL_MINUTEFROMRTC = (FCL_MINUTEFROMRTC >> 4) * 10 + (FCL_MINUTEFROMRTC & 15);
FCV_MINUTEFROMMIDNITE = FCL_HOURFROMRTC * 60 + FCL_MINUTEFROMRTC;
```

```
   #if 0 // Disabled code
   // I2C Stop
   // Call Component Macro: I2C_Master1::Stop()
   FCD_005f1_I2C_Master1__Stop();

   #endif // #if 0: Disabled code
}

/*=-----------------------------------------------------------------------=*\
  Use :The analog voltage is from a divider network, used on the LCD, to sense which pushbotton has been pressed.
\*=-----------------------------------------------------------------------=*/
void FCM_ButtonPress()
{
   //Local variable definitions
   MX_FLOAT FCL_PB_VOLTAGEREAD;
   MX_SINT16 FCL_PB_VOLTAGEREAD_MV_INT;


   // Read PB Voltage
   // Call Component Macro: .PB_VoltageRead=LCD_PB::GetVoltage()
   FCL_PB_VOLTAGEREAD = FCD_0d511_LCD_PB__GetVoltage();

   // ConvertTo MV
   // Calculation:
   //  .PB_VoltageRead_MV_Int = .PB_VoltageRead * 1000
```

```
FCL_PB_VOLTAGEREAD_MV_INT = flt_toi(flt_mul(FCL_PB_VOLTAGEREAD, 1000));

// Right
// Decision: .PB_VoltageRead_MV_Int > 25 AND .PB_VoltageRead_MV_Int <= 300?
if (FCL_PB_VOLTAGEREAD_MV_INT > 25 & FCL_PB_VOLTAGEREAD_MV_INT <= 300)
{

   //Comment:
   //Right Pressed

   // Calculation
   // Calculation:
   //  MenuPosition = 3
   FCV_MENUPOSITION = 3;

   // Goto Connection Point
   // Goto Connection Point: [A]: A
   goto FCC_ButtonPress_A;

// } else {

}

// Up-1093 mv
// Decision: .PB_VoltageRead_MV_Int > 300 AND .PB_VoltageRead_MV_Int <= 1400?
```

```
if (FCL_PB_VOLTAGEREAD_MV_INT > 300 & FCL_PB_VOLTAGEREAD_MV_INT <= 1400)
{

   //Comment:
   //Up Pressed

   // Calculation
   // Calculation:
   //  MenuPosition = 2
   FCV_MENUPOSITION = 2;

   // Goto Connection Point
   // Goto Connection Point: [A]: A
   goto FCC_ButtonPress_A;

// } else {

}

// Down- 2065 mv
// Decision: .PB_VoltageRead_MV_Int > 1400 AND .PB_VoltageRead_MV_Int <= 2500?
if (FCL_PB_VOLTAGEREAD_MV_INT > 1400 & FCL_PB_VOLTAGEREAD_MV_INT <= 2500)
{

   //Comment:
```

```
      //Down pressed - Stop


      // Calculation
      // Calculation:
      //  MenuPosition = 1
      FCV_MENUPOSITION = 1;


      // Goto Connection Point
      // Goto Connection Point: [A]: A
      goto FCC_ButtonPress_A;


   // } else {


   }


   // Connection Point
   // Connection Point: [A]: A
FCC_ButtonPress_A:
;


}


/*=----------------------------------------------------------------------=*\
   Use :Facing south, LR takes on values of 1-Left/East, 2-Right/West.
      :If thge rotation direction is incorrect, remove power and swap 2 wires.
```

```
      :Motor omega = 10.4 x sun's omega. Implys that duty cycle is 0.095
      :Do not know why I have to double the move time.
      :
      :Parameters for macro Move1Axis:
      :  LR : 1 or 2, to move the array left or right.
      :  MoveTime : MX_SINT16
\*=----------------------------------------------------------------------=*/
void FCM_Move1Axis(MX_UINT8 FCL_LR, MX_SINT16 FCL_MOVETIME)
{

   // Switch
   // Switch: .LR?
   switch (FCL_LR)
   {
      case 1:
      {
         //Comment:
         //Left Winding

         // "  -Move West-"
         // Call Component Macro: LCD1::PrintString("  -Move West-")
         FCD_04071_LCD1__PrintString("  -Move West-", 14);

         // Cursor 2,1
         // Call Component Macro: LCD1::Cursor(2, 1)
```

```
FCD_04071_LCD1__Cursor(2, 1);

// Print Move Time
// Call Component Macro: LCD1::PrintNumber(.MoveTime)
FCD_04071_LCD1__PrintNumber(FCL_MOVETIME);

// Print Seconds
// Call Component Macro: LCD1::PrintString(" Seconds")
FCD_04071_LCD1__PrintString(" Seconds", 9);

// Output
// Output: 1 -> D2
FCP_SET(B, D, 0x04, 2, (1));
// Delay
// Delay: .MoveTime s
FCI_DELAYINT_S(FCL_MOVETIME);

// Output
// Output: 0 -> D2
FCP_SET(B, D, 0x04, 2, (0));
// CLS
// Call Component Macro: LCD1::Clear()
FCD_04071_LCD1__Clear();

break;
```

```
}
case 2:
{
   //Comment:
   //Right Winding

   // "  -Move East-"
   // Call Component Macro: LCD1::PrintString("  -Move East-")
   FCD_04071_LCD1__PrintString("  -Move East-", 14);

   // Cursor 2,1
   // Call Component Macro: LCD1::Cursor(2, 1)
   FCD_04071_LCD1__Cursor(2, 1);

   // Print Move Time
   // Call Component Macro: LCD1::PrintNumber(.MoveTime)
   FCD_04071_LCD1__PrintNumber(FCL_MOVETIME);

   // Print Seconds
   // Call Component Macro: LCD1::PrintString(" Seconds")
   FCD_04071_LCD1__PrintString(" Seconds", 9);

   // Output
   // Output: 1 -> D3
   FCP_SET(B, D, 0x08, 3, (1));
```

```
        // Delay
        // Delay: .MoveTime s
        FCI_DELAYINT_S(FCL_MOVETIME);

        // Output
        // Output: 0 -> D3
        FCP_SET(B, D, 0x08, 3, (0));
        // CLS
        // Call Component Macro: LCD1::Clear()
        FCD_04071_LCD1__Clear();

        break;
      }
      // default:

   }

}

/*=---------------------------------------------------------------------=*\
  Use :Macro to run RTC from button cell.
\*=---------------------------------------------------------------------=*/
void FCM_PowerFromBattery()
{
```

```
    // I2C Start
    // Call Component Macro: I2C_Master1::Start()
    FCD_005f1_I2C_Master1__Start();

    // Write Mode
    // Call Component Macro: I2C_Master1::TransmitByte(208)
    FCD_005f1_I2C_Master1__TransmitByte(208);

    // Select Register
    // Call Component Macro: I2C_Master1::TransmitByte(0x0E)
    FCD_005f1_I2C_Master1__TransmitByte(0x0E);

    // Write register bitmap, bit 7 is /EOSC
    // Call Component Macro: I2C_Master1::TransmitByte(0b00011100)
    FCD_005f1_I2C_Master1__TransmitByte(28);

    // I2C Stop
    // Call Component Macro: I2C_Master1::Stop()
    FCD_005f1_I2C_Master1__Stop();

}

/*=-----------------------------------------------------------------------=*\
  Use :So far, this just gets values and writes them, via I2C to the RTC.
      :Need to Initialize, and call this in main.
```

```
\*=----------------------------------------------------------------------=*/
void FCM_SetTime()
{
   //Local variable definitions
   #define FCL_DISPLAYDELAY (700)
   MX_UINT8 FCL_HOURFROMRTC = (0x1);
   MX_UINT8 FCL_MINUTEFROMRTC = (0x1);
   MX_UINT8 FCL_SECONDFROMRTC = (0x1);
   MX_FLOAT FCL_BUTTONVOLTAGE;
   MX_SINT16 FCL_BUTTONVOLTAGE_MV;


   //Comment:
   //Need to get some starting point for the time.
   //The only time the time needs to be set is with a battery cahnge.
   //And I think it default to 0. So I am going to set the time to 1's,
   //except the year

   //Comment:
   //Comment

   // "To Change Minute,"
   // Call Component Macro: LCD1::PrintString("To Change Minute,")
   FCD_04071_LCD1__PrintString("To Change Minute,", 18);
```

```
// Cursor 0,1
// Call Component Macro: LCD1::Cursor(0, 1)
FCD_04071_LCD1__Cursor(0, 1);

// "Press Up"
// Call Component Macro: LCD1::PrintString("Press Up")
FCD_04071_LCD1__PrintString("Press Up", 9);

// Delay
// Delay: 2 s
FCI_DELAYBYTE_S(2);

// CLS
// Call Component Macro: LCD1::Clear()
FCD_04071_LCD1__Clear();

// "Now!"
// Call Component Macro: LCD1::PrintString("Now!")
FCD_04071_LCD1__PrintString("Now!", 5);

// Delay
// Delay: 2 s
FCI_DELAYBYTE_S(2);

// CLS
```

```
   // Call Component Macro: LCD1::Clear()
   FCD_04071_LCD1__Clear();

   // Connection Point
   // Connection Point: [A]: A
FCC_SetTime_A:
;

   // Get Voltage from AN0
   // Call Component Macro: .ButtonVoltage=LCD_PB::GetVoltage()
   FCL_BUTTONVOLTAGE = FCD_0d511_LCD_PB__GetVoltage();

   // Calculation
   // Calculation:
   //  .ButtonVoltage_MV = .ButtonVoltage * 1000
   FCL_BUTTONVOLTAGE_MV = flt_toi(flt_mul(FCL_BUTTONVOLTAGE, 1000));

   // Decision
   // Decision: .ButtonVoltage_MV > 1000 AND .ButtonVoltage_MV < 2000?
   if (FCL_BUTTONVOLTAGE_MV > 1000 & FCL_BUTTONVOLTAGE_MV < 2000)
   {

      // Calculation
      // Calculation:
      //  .MinuteFromRTC = .MinuteFromRTC + 1
```

```
FCL_MINUTEFROMRTC = FCL_MINUTEFROMRTC + 1;


// Decision
// Decision: .MinuteFromRTC > 60?
if (FCL_MINUTEFROMRTC > 60)
{

   // Calculation
   // Calculation:
   //  .MinuteFromRTC = 1
   FCL_MINUTEFROMRTC = 1;

// } else {

}

// Print Minute
// Call Component Macro: LCD1::PrintNumber(.MinuteFromRTC)
FCD_04071_LCD1__PrintNumber(FCL_MINUTEFROMRTC);

// Delay
// Delay: .DisplayDelay ms
FCl_DELAYINT_MS(FCL_DISPLAYDELAY);

// CLS
```

```
    // Call Component Macro: LCD1::Clear()
    FCD_04071_LCD1__Clear();

    // Goto Connection Point
    // Goto Connection Point: [A]: A
    goto FCC_SetTime_A;

// } else {

}

// "ToChange Hour,"
// Call Component Macro: LCD1::PrintString("ToChange Hour,")
FCD_04071_LCD1__PrintString("ToChange Hour,", 15);

// Cursor 0,1
// Call Component Macro: LCD1::Cursor(0, 1)
FCD_04071_LCD1__Cursor(0, 1);

// "Press Up or Down"
// Call Component Macro: LCD1::PrintString("Press Up or Down")
FCD_04071_LCD1__PrintString("Press Up or Down", 17);

// Delay
// Delay: 2 s
```

```
    FCI_DELAYBYTE_S(2);

    // CLS
    // Call Component Macro: LCD1::Clear()
    FCD_04071_LCD1__Clear();

    // "Now!"
    // Call Component Macro: LCD1::PrintString("Now!")
    FCD_04071_LCD1__PrintString("Now!", 5);

    // Delay
    // Delay: 2 s
    FCI_DELAYBYTE_S(2);

    // CLS
    // Call Component Macro: LCD1::Clear()
    FCD_04071_LCD1__Clear();

    // Connection Point
    // Connection Point: [B]: B
FCC_SetTime_B:
;

    // Get Voltage from AN0
    // Call Component Macro: .ButtonVoltage=LCD_PB::GetVoltage()
```

```
FCL_BUTTONVOLTAGE = FCD_0d511_LCD_PB__GetVoltage();


// Calculation

// Calculation:

//  .ButtonVoltage_MV = .ButtonVoltage * 1000

FCL_BUTTONVOLTAGE_MV = flt_toi(flt_mul(FCL_BUTTONVOLTAGE, 1000));


// Decision

// Decision: .ButtonVoltage_MV > 1000 AND .ButtonVoltage_MV < 2000?

if (FCL_BUTTONVOLTAGE_MV > 1000 & FCL_BUTTONVOLTAGE_MV < 2000)

{


   // Calculation

   // Calculation:

   //  .HourFromRTC = .HourFromRTC + 1

   FCL_HOURFROMRTC = FCL_HOURFROMRTC + 1;


   // Decision

   // Decision: .HourFromRTC > 23?

   if (FCL_HOURFROMRTC > 23)

   {


      // Calculation

      // Calculation:

      //  .HourFromRTC = 1
```

```
      FCL_HOURFROMRTC = 1;


    // } else {


    }


    // Print Minute
    // Call Component Macro: LCD1::PrintNumber(.HourFromRTC)
    FCD_04071_LCD1__PrintNumber(FCL_HOURFROMRTC);


    // Delay
    // Delay: .DisplayDelay ms
    FCI_DELAYINT_MS(FCL_DISPLAYDELAY);


    // CLS
    // Call Component Macro: LCD1::Clear()
    FCD_04071_LCD1__Clear();


    // Goto Connection Point
    // Goto Connection Point: [B]: B
    goto FCC_SetTime_B;


// } else {


}
```

```
// Calculation

// Calculation:

//  MinuteFromMidnite = .HourFromRTC * 60 + .MinuteFromRTC

FCV_MINUTEFROMMIDNITE = FCL_HOURFROMRTC * 60 + FCL_MINUTEFROMRTC;


// DEC to BCD

// Calculation:

//  .SecondFromRTC = (.SecondFromRTC / 10 << 4) + (.SecondFromRTC MOD 10)

//  .MinuteFromRTC = (.MinuteFromRTC / 10 << 4) + (.MinuteFromRTC MOD 10)

//  .HourFromRTC = (.HourFromRTC / 10 << 4) + (.HourFromRTC MOD 10)

FCL_SECONDFROMRTC = (FCL_SECONDFROMRTC / 10 << 4) + (FCL_SECONDFROMRTC % 10);

FCL_MINUTEFROMRTC = (FCL_MINUTEFROMRTC / 10 << 4) + (FCL_MINUTEFROMRTC % 10);

FCL_HOURFROMRTC = (FCL_HOURFROMRTC / 10 << 4) + (FCL_HOURFROMRTC % 10);


// I2C Start

// Call Component Macro: I2C_Master1::Start()

FCD_005f1_I2C_Master1__Start();


// 208 = write mode

// Call Component Macro: I2C_Master1::TransmitByte(208)

FCD_005f1_I2C_Master1__TransmitByte(208);


// Transmit 0

// Call Component Macro: I2C_Master1::TransmitByte(0)
```

```
FCD_005f1_I2C_Master1__TransmitByte(0);

// Transmit Seconds
// Call Component Macro: I2C_Master1::TransmitByte(.SecondFromRTC)
FCD_005f1_I2C_Master1__TransmitByte(FCL_SECONDFROMRTC);

// Transmit Minutes
// Call Component Macro: I2C_Master1::TransmitByte(.MinuteFromRTC)
FCD_005f1_I2C_Master1__TransmitByte(FCL_MINUTEFROMRTC);

// Transmit Hours
// Call Component Macro: I2C_Master1::TransmitByte(.HourFromRTC)
FCD_005f1_I2C_Master1__TransmitByte(FCL_HOURFROMRTC);

// I2C Stop
// Call Component Macro: I2C_Master1::Stop()
FCD_005f1_I2C_Master1__Stop();

// Calculation
// Calculation:
//  MenuPosition = 1
FCV_MENUPOSITION = 1;

//Local variable definitions
#undef FCL_DISPLAYDELAY
```

```
}

/*=----------------------------------------------------------------------=*\
  Use :How to set the mC Hz to configure the interrupt so I get the minute correct:
     :Hz = TimerScaleCounter * 8 * (2 to the16 power) / 60.
     :So for 1831 the Hz is 15,999,522.
\*=----------------------------------------------------------------------=*/
void FCM_Timer1_Overflow()
{

  // Decision
  // Decision: TimerScaleCounter = 1831?
  if (FCV_TIMERSCALECOUNTER == 1831)
  {

    // Calculation
    // Calculation:
    //  TimerScaleCounter = 0
    FCV_TIMERSCALECOUNTER = 0;

    // Calculation
    // Calculation:
    //  MinuteFromMidnite = MinuteFromMidnite + 1
    FCV_MINUTEFROMMIDNITE = FCV_MINUTEFROMMIDNITE + 1;
```

```
    } else {

        // Calculation
        // Calculation:
        //  TimerScaleCounter = TimerScaleCounter + 1
        FCV_TIMERSCALECOUNTER = FCV_TIMERSCALECOUNTER + 1;

    }

    // Decision
    // Decision: MinuteFromMidnite > 1440?
    if (FCV_MINUTEFROMMIDNITE > 1440)
    {

        // Calculation
        // Calculation:
        //  MinuteFromMidnite = 0
        FCV_MINUTEFROMMIDNITE = 0;

    // } else {

    }

}
```

```
/*=-----------------------------------------------------------------------=*\
  Use :Converts Minute to Hour:Minute.
\*=-----------------------------------------------------------------------=*/
void FCM_MinutesToHour_Minute()
{
   //Local variable definitions
   MX_SINT16 FCL_HOUR = (0);
   MX_SINT16 FCL_MINUTES = (0);
   #define FCLsz_STRINGS 20
   MX_CHAR FCL_STRINGS[FCLsz_STRINGS];


   // Calculation
   // Calculation:
   //  .Minutes = (MinuteFromMidnite) MOD 60
   FCL_MINUTES = (FCV_MINUTEFROMMIDNITE) % 60;

   // Calculation
   // Calculation:
   //  .Hour = (MinuteFromMidnite / 60)
   FCL_HOUR = (FCV_MINUTEFROMMIDNITE / 60);

   // Decision
   // Decision: .Minutes < 10?
   if (FCL_MINUTES < 10)
```

```
{

   // Calculation
   // Calculation:
   //  .StringS = ToString$ (.Minutes)
   //  .StringS = "0" + .StringS
   FCI_TOSTRING(FCL_MINUTES, FCL_STRINGS,20);
   FCI_SHEAD("0",2, FCL_STRINGS,FCLsz_STRINGS, FCL_STRINGS,20);

} else {

   // Calculation
   // Calculation:
   //  .StringS = ToString$ (.Minutes)
   FCI_TOSTRING(FCL_MINUTES, FCL_STRINGS,20);

}

// "Time(Std)->"
// Call Component Macro: LCD1::PrintString("Time(Std)->")
FCD_04071_LCD1__PrintString("Time(Std)->", 12);

// Print Hour
// Call Component Macro: LCD1::PrintNumber(.Hour)
FCD_04071_LCD1__PrintNumber(FCL_HOUR);
```

```
   // ":"
   // Call Component Macro: LCD1::PrintString(":")
   FCD_04071_LCD1__PrintString(":", 2);

   // Print Minutes
   // Call Component Macro: LCD1::PrintString(.StringS)
   FCD_04071_LCD1__PrintString(FCL_STRINGS, FCLsz_STRINGS);

   //Local variable definitions
   #undef FCLsz_STRINGS
}

/*=---------------------------------------------------------------------=*\
  Use :As was advertized by Apple, with a 1 buttom mouse, it is practilly impossible to press the wrong button. This is not quite that simple, as there are 4 buttons that respond.
\*=---------------------------------------------------------------------=*/
void FCM_Instructions()
{
   //Local variable definitions
   #define FCL_CLS_DELAY (4) // Delay for displaying instructions.

   // "Run->Press up PB"
   // Call Component Macro: LCD1::PrintString("Run->Press up PB")
   FCD_04071_LCD1__PrintString("Run->Press up PB", 17);
```

```
// Cursor 0,1
// Call Component Macro: LCD1::Cursor(0, 1)
FCD_04071_LCD1__Cursor(0, 1);

// " Stop->Down PB."
// Call Component Macro: LCD1::PrintString(" Stop->Down PB.")
FCD_04071_LCD1__PrintString(" Stop->Down PB.", 16);

// Delay
// Delay: .CLS_Delay s
FCl_DELAYBYTE_S(FCL_CLS_DELAY);

// CLS
// Call Component Macro: LCD1::Clear()
FCD_04071_LCD1__Clear();

// "To change Time"
// Call Component Macro: LCD1::PrintString("To change Time")
FCD_04071_LCD1__PrintString("To change Time", 15);

// Cursor 0,1
// Call Component Macro: LCD1::Cursor(0, 1)
FCD_04071_LCD1__Cursor(0, 1);
```

```
// "Press Right PB."
// Call Component Macro: LCD1::PrintString(" Press Right PB.")
FCD_04071_LCD1__PrintString(" Press Right PB.", 17);

// Delay
// Delay: .CLS_Delay s
FCI_DELAYBYTE_S(FCL_CLS_DELAY);

// CLS
// Call Component Macro: LCD1::Clear()
FCD_04071_LCD1__Clear();

// "Eight-FourThirty"
// Call Component Macro: LCD1::PrintString("Eight-FourThirty")
FCD_04071_LCD1__PrintString("Eight-FourThirty", 17);

// Delay
// Delay: .CLS_Delay s
FCI_DELAYBYTE_S(FCL_CLS_DELAY);

// CLS
// Call Component Macro: LCD1::Clear()
FCD_04071_LCD1__Clear();

//Local variable definitions
```

```
   #undef FCL_CLS_DELAY // Delay for displaying instructions.
}

/*=---------------------------------------------------------------------=*\
  Use :
\*=---------------------------------------------------------------------=*/
void FCM_Version()
{

   // Start
   // Call Component Macro: LCD1::Start()
   FCD_04071_LCD1__Start();

   // Delay
   // Delay: 1 s
   FCI_DELAYBYTE_S(1);

   // "FTS_1 Axis"
   // Call Component Macro: LCD1::PrintString("  FTS_1 Axis")
   FCD_04071_LCD1__PrintString("  FTS_1 Axis", 13);

   // Cursor 0,1
   // Call Component Macro: LCD1::Cursor(0, 1)
   FCD_04071_LCD1__Cursor(0, 1);
```

```
   // Key In Version
   // Call Component Macro: LCD1::PrintString("  1Axis_1_16")
   FCD_04071_LCD1__PrintString("  1Axis_1_16", 13);

   // Delay
   // Delay: 2 s
   FCI_DELAYBYTE_S(2);

   // CLS
   // Call Component Macro: LCD1::Clear()
   FCD_04071_LCD1__Clear();

}

//The pot is for a voltage divider network, used on the LCD, to sense which pushbotton has been pressed. See ButtonPress Subroutine.

/*================================================================*\
  Use :Main
\*================================================================*/
int main()
{
   MCUSR=0x00;
```

```
// LCD Start
// Call Component Macro: LCD1::Start()
FCD_04071_LCD1__Start();

// I2C Initialize
// Call Component Macro: I2C_Master1::Initialise()
FCD_005f1_I2C_Master1__Initialise();

// Call Version
// Call Macro: Version()
FCM_Version();

// Call Instructions
// Call Macro: Instructions()
FCM_Instructions();

// Delay
// Delay: 1 s
FCI_DELAYBYTE_S(1);

// PowerFromBattery
// Call Macro: PowerFromBattery()
FCM_PowerFromBattery();

// Menu Position = Stopped
```

```
// Calculation:
//  MenuPosition = 1
FCV_MENUPOSITION = 1;

// Interrupt for incrementing minutes
// Interrupt: Enable TMR1
TCCR1B &= 0xf8;
TCCR1B |= 0x02;
sei();
TIMSK1 |= (1 << TOIE1);

// Interrupt for Button Press
// Interrupt: Enable IOC1
sei();
PCMSK1=0x1;
PCICR |= (1 << PCIE1);

// Delay
// Delay: 1 s
FCI_DELAYBYTE_S(1);

// Call Get Time from RTC
// Call Macro: GetTimeFromRTC()
FCM_GetTimeFromRTC();
```

```
// Loop
// Loop: While 1
while (1)
{

   // ReadRTC and set minute
   // Decision: MinuteFromMidnite = 60 && PerformOnce?
   if (FCV_MINUTEFROMMIDNITE == 60 && FCV_PERFORMONCE)
   {

      // Call Get Time from RTC
      // Call Macro: GetTimeFromRTC()
      FCM_GetTimeFromRTC();

      //Comment:
      //Could have a set time here. As well as in the switch statement.

      // Calculation
      // Calculation:
      //  PerformOnce = 0
      FCV_PERFORMONCE = 0;

   // } else {

   }
```

```
// Decision
// Decision: MinuteFromMidnite = 70?
if (FCV_MINUTEFROMMIDNITE == 70)
{

   // Calculation
   // Calculation:
   //  PerformOnce = 1
   FCV_PERFORMONCE = 1;

// } else {

}

// Switch
// Switch: MenuPosition?
switch (FCV_MENUPOSITION)
{
   case 1:
   {
      //Comment:
      //Stop

      // "Stopped"
```

```
    // Call Component Macro: LCD1::PrintString("Stopped")
    FCD_04071_LCD1__PrintString("Stopped", 8);

    // Cursor 0,1
    // Call Component Macro: LCD1::Cursor(0, 1)
    FCD_04071_LCD1__Cursor(0, 1);

    // Convert X:XX
    // Call Macro: MinutesToHour_Minute()
    FCM_MinutesToHour_Minute();

    // Delay
    // Delay: 5 s
    FCI_DELAYBYTE_S(5);

    // CLS
    // Call Component Macro: LCD1::Clear()
    FCD_04071_LCD1__Clear();

    // Delay
    // Delay: 1 s
    FCI_DELAYBYTE_S(1);

    break;
}
```

```
    case 2:
    {
        //Comment:
        //Run

        // Call Run
        // Call Macro: Run()
        FCM_Run();

        break;
    }
    case 3:
    {
        //Comment:
        //Set time is going here.

        // Call Set Time
        // Call Macro: SetTime()
        FCM_SetTime();

        break;
    }
    // default:

}
```

```
   }


   mainendloop: goto mainendloop;
}




/*=====================================================================*\
  Use :Interrupt
\*=====================================================================*/


   //Handler code for [TMR1]
   #ifndef MX_TIMER1OVF_HANDLER
   #define MX_TIMER1OVF_HANDLER
   ISR(TIMER1_OVF_vect)
   {
      FCM_Timer1_Overflow();
   }
   #else
   #warning The <TIMER1 OVF> interrupt has previously been enabled, so the macro <Timer1_Overflow> may never get called.
   #endif
```

```
//Handler code for [IOC1]
#ifndef MX_PCINT1_HANDLER
#define MX_PCINT1_HANDLER
ISR(PCINT1_vect)
{
   FCM_ButtonPress();
}
#else
#warning The <PORTC Change> interrupt has previously been enabled, so the macro <ButtonPress> may never get called.
#endif
```