

```
//*****
*
**
** Source name: C:\Users\Bill
Swann\Desktop\FlowCode\2_AxisController\Az_EI_Calculation\AzEI_I_ThinkItWorks.fcx
** Title:
** Description:
** Device: AVR.ATMEGA.ATMEGA328P
**
** Generated by: Flowcode v6.1.4.0
** Date: Saturday, December 24, 2016 11:00:52
** Users: 1
** Registered to: Bill Swann
** Licence key: EXKDOW
**
**
** http://www.matrixsl.com
**
//*****
*
```

```
#define MX_AVR
#define MX_CAL_AVR
#define MX_CLK_SPEED 800000
#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\eprom.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_DEGTORAD (0.017453279346227646)

#define FCV_FALSE (0)

#define FCV_RADTODEGREE (57.295825958251953)

#define FCV_TRUE (1)

MX_GLOBAL MX_SINT16 FCV_MONTH = (1);

MX_GLOBAL MX_SINT16 FCV_COUNT = (1);

MX_GLOBAL MX_SINT16 FCV_HOUR = (0);

MX_GLOBAL MX_SINT16 FCV_DAYOFMONTH = (0);

MX_GLOBAL MX_SINT16 FCV_DAYOFYEAR = (1);

MX_GLOBAL MX_FLOAT FCV_ELEVATION = (0.0);

MX_GLOBAL MX_FLOAT FCV_DECLINATION = (0.0);

MX_GLOBAL MX_FLOAT FCV_AZIMUTH = (0.0);

MX_GLOBAL MX_FLOAT FCV_LATITUDE = (48.816665649414063); // Value is negative west of
Greenwich

MX_GLOBAL MX_FLOAT FCV_LONGITUDE = (0.0);

MX_GLOBAL MX_SINT16 FCV_MINUTE = (0);

MX_GLOBAL MX_FLOAT FCV_EOT = (0.0);

MX_GLOBAL MX_FLOAT FCV_OMEGA = (0.0);

MX_SINT16 FCM_DayOfYear(MX_UINT8 FCL_MONTH, MX_UINT8 FCL_DAYOFMONTH);

void FCM_SolarHourAngle();

void FCM_Elevation();

MX_FLOAT FCM_Declination();

void FCM_Azimuth();

void FCM_EoT();
```

```

/*=====*\
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 :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];
```

```
FCL_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, B, 0x1, 0x0, 0);
```

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

```
FCP_SET(B, B, 0x4, 0x2, 0);
```

```
FCP_SET(B, B, 0x8, 0x3, 0);
```

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

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

```
#if (0)
```

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

```
// #else
```

```
#endif
```

```
FCL_NIBBLE = (FCL_DATA >> 4);
```

```
FCP_SET(B, B, 0x1, 0x0, (FCL_NIBBLE & 0x01));
```

```
FCL_NIBBLE = FCL_NIBBLE >> 1;
```

```
FCP_SET(B, B, 0x2, 0x1, (FCL_NIBBLE & 0x01));
```

```
FCL_NIBBLE = FCL_NIBBLE >> 1;
```

```
FCP_SET(B, B, 0x4, 0x2, (FCL_NIBBLE & 0x01));
```

```
FCL_NIBBLE = FCL_NIBBLE >> 1;
```

```
FCP_SET(B, B, 0x8, 0x3, (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, 0x10, 0x4, 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, 0x20, 0x5, 1);

FCI_DELAYBYTE_US(100);

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

FCI_DELAYBYTE_US(100);
```

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

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

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

```
FCP_SET(B, B, 0x4, 0x2, 0);
```

```
FCP_SET(B, B, 0x8, 0x3, 0);
```

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

```
FCL_NIBBLE = (FCL_DATA & 0xf);
```

```
FCP_SET(B, B, 0x1, 0x0, (FCL_NIBBLE & 0x01));
```

```
FCL_NIBBLE = FCL_NIBBLE >> 1;
```

```
FCP_SET(B, B, 0x2, 0x1, (FCL_NIBBLE & 0x01));
```

```
FCL_NIBBLE = FCL_NIBBLE >> 1;
```

```
FCP_SET(B, B, 0x4, 0x2, (FCL_NIBBLE & 0x01));
```

```
FCL_NIBBLE = FCL_NIBBLE >> 1;
```

```
FCP_SET(B, B, 0x8, 0x3, (FCL_NIBBLE & 0x01));
```

```
if (FCL_TYPE)
```

```
{
```

```
    FCP_SET(B, B, 0x10, 0x4, 1);
```

```
// } else {
```

```
}
```

```
FCI_DELAYBYTE_US(100);
```

```
FCP_SET(B, B, 0x20, 0x5, 1);
```

```
FCI_DELAYBYTE_US(100);
```

```
FCP_SET(B, B, 0x20, 0x5, 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, B, 0x1, 0x0, 0);
```

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

```
    FCP_SET(B, B, 0x4, 0x2, 0);
```

```
    FCP_SET(B, B, 0x8, 0x3, 0);
```

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

```
    FCP_SET(B, B, 0x20, 0x5, 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 :Parameters for macro DayOfYear:
    : Month : MX_UINT8
    : DayOfMonth : MX_UINT8
    :
    :Returns : MX_SINT16
\*=-----=*/  

MX_SINT16 FCM_DayOfYear(MX_UINT8 FCL_MONTH, MX_UINT8 FCL_DAYOFMONTH)
{
    //Local variable definitions
    #define FCL_DEGTORAD (0.017453279346227646) // Multiply by this to convert Degrees to Radians
    #define FCL_RADTODEG (57.295825958251953) // Multiply by this to convert Radians to Degrees
    MX_SINT16 FCL_DAYOFYEAR = (0);

```

```
#define FCLsz_STRINGS 6

MX_CHAR FCL_STRINGS[FCLsz_STRINGS];

MX_FLOAT FCL_DECLINATION = (0.0);

MX_FLOAT FCL_EQUATIONOFTIME = (0.0);

MX_FLOAT FCL_FRACTIONALYEAR = (0.0); // (day of year + hour/24)*360/365.25

MX_FLOAT FCL_SOLARHOURANGLE = (0.0);

MX_FLOAT FCL_SOLARZENITHANGLE;

MX_FLOAT FCL_SZA1;

MX_FLOAT FCL_SZA2;

MX_FLOAT FCL_SZA3;

MX_FLOAT FCL_SZA4;

MX_FLOAT FCL_SZA5;

MX_FLOAT FCL_ELEVATION = (0.0);

MX_SINT16 FCR_RETVAL;

// Switch

// Switch: .Month?

switch (FCL_MONTH)

{

    case 1:

        {

            // January

            // Calculation:

            // .DayOfYear = .DayOfMonth
```

```
FCL_DAYOFYEAR = FCL_DAYOFMONTH;

break;
}
case 2:
{
    // Feb
    // Calculation:
    // .DayOfYear = .DayOfMonth + 31
    FCL_DAYOFYEAR = FCL_DAYOFMONTH + 31;

    break;
}
case 3:
{
    //Comment:
    //Add 1 if Leap Year.

    // March
    // Calculation:
    // .DayOfYear = .DayOfMonth + 59
    FCL_DAYOFYEAR = FCL_DAYOFMONTH + 59;

    break;
}
```

case 4:

```
{  
    //Comment:  
    //Add 1 if Leap Year.  
  
    // April  
    // Calculation:  
    // .DayOfYear = .DayOfMonth + 90  
    FCL_DAYOFYEAR = FCL_DAYOFMONTH + 90;  
  
    break;  
}
```

case 5:

```
{  
    //Comment:  
    //Add 1 if Leap Year.  
  
    // May  
    // Calculation:  
    // .DayOfYear = .DayOfMonth + 120  
    FCL_DAYOFYEAR = FCL_DAYOFMONTH + 120;  
  
    break;  
}
```

case 6:

```
{  
    //Comment:  
    //Add 1 if Leap Year.  
  
    // June  
    // Calculation:  
    // .DayOfYear = .DayOfMonth + 150  
    FCL_DAYOFYEAR = FCL_DAYOFMONTH + 150;  
  
    break;  
}  
case 7:  
{  
    //Comment:  
    //Add 1 if Leap Year.  
  
    // July  
    // Calculation:  
    // .DayOfYear = .DayOfMonth + 180  
    FCL_DAYOFYEAR = FCL_DAYOFMONTH + 180;  
  
    break;  
}  
case 8:  
{
```



```
//Comment:
//Add 1 if Leap Year.

// August
// Calculation:
// .DayOfYear = .DayOfMonth + 212
FCL_DAYOFYEAR = FCL_DAYOFMONTH + 212;

break;
}
case 9:
{
//Comment:
//Add 1 if Leap Year.

// Sept
// Calculation:
// .DayOfYear = .DayOfMonth + 243
FCL_DAYOFYEAR = FCL_DAYOFMONTH + 243;

break;
}
case 10:
{
//Comment:
```

```
//Add 1 if Leap Year.

// October
// Calculation:
// .DayOfYear = .DayOfMonth + 273
FCL_DAYOFYEAR = FCL_DAYOFMONTH + 273;

break;
}
// default:

}

// Switch
// Switch: .Month?
switch (FCL_MONTH)
{
case 11:
{
//Comment:
//Add 1 if Leap Year.

// November
// Calculation:
// .DayOfYear = .DayOfMonth + 304
```

```
FCL_DAYOFYEAR = FCL_DAYOFMONTH + 304;

break;
}
case 12:
{
//Comment:
//Add 1 if Leap Year.

// December
// Calculation:
// .DayOfYear = .DayOfMonth + 334
FCL_DAYOFYEAR = FCL_DAYOFMONTH + 334;

break;
}
// default:

}

// Calculation
// Calculation:
// .Return = .DayOfYear
FCR_RETVAL = FCL_DAYOFYEAR;
```

```

return (FCR_RETVAL);

//Local variable definitions

#undef FCL_DEGTORAD // Multiply by this to convert Degrees to Radians
#undef FCL_RADTODEG // Multiply by this to convert Radians to Degrees
#undef FCLsz_STRINGS
}

/*-----*\

Use :15*(Ts-12) = Omega

:Ts = Solar Time

:The Longitudinal Correction for Houston is ( -95.33 + 90) / 15 =

:-5.33/15 = -0.35

\*-----*/

void FCM_SolarHourAngle()
{
//Local variable definitions

MX_FLOAT FCL_SOLARTIME = (0.0); // Ts

// Calculation

// Calculation:

// .SolarTime = Hour + Minute / 60.0 + EoT / 60.0 - 0.35 //timezone offset,95w vs 90w

// Omega = 15.0 * .SolarTime - 180

FCL_SOLARTIME = flt_sub(flt_add(flt_add(flt_fromi(FCV_HOUR), flt_div(flt_fromi(FCV_MINUTE),
60.0)), flt_div(FCV_EOT, 60.0)), 0.35);

```

```

FCV_OMEGA = flt_sub(flt_mul(15.0, FCL_SOLARTIME), 180);

}

/*=-----=*/

Use :Seems to work if I follow the book. That is. Local Clock Time is the watch.

:Solar Time = LCT+EoT/60 -1/15(Long_Houston-Long_NO)

:Combine with Omega = 15 * Ts - 150

\*=-----=*/

void FCM_Elevation()
{
//Local variable definitions

MX_FLOAT FCL_ELEV1;

MX_FLOAT FCL_ELEV2;

MX_FLOAT FCL_ELEV3;

MX_FLOAT FCL_ELEV4;

MX_FLOAT FCL_ELEV5;

#define FCLsz_STRINGS 20

MX_CHAR FCL_STRINGS[FCLsz_STRINGS];

// Calculation

// Calculation:

// //Elevation = sin (Declination * DegToRad) * sin (Latitude * DegToRad) + cos (Declination *
DegToRad) * cos (SolarHourAngle * DegToRad) * cos (Latitude * DegToRad)

// //Elevation = RadToDegree * asin (Elevation)

```

```

// .ELEV1 = sin (Declination * DegToRad)
FCL_ELEV1 = sin(flt_mul(FCV_DECLINATION, FCV_DEGTORAD));

// Calculation
// Calculation:
// .ELEV1 = sin (Declination * DegToRad)
FCL_ELEV1 = sin(flt_mul(FCV_DECLINATION, FCV_DEGTORAD));

// Calculation
// Calculation:
// .ELEV2 = sin (Latitude * DegToRad)
FCL_ELEV2 = sin(flt_mul(FCV_LATITUDE, FCV_DEGTORAD));

// Calculation
// Calculation:
// .ELEV3 = cos (Declination * DegToRad)
FCL_ELEV3 = cos(flt_mul(FCV_DECLINATION, FCV_DEGTORAD));

// Calculation
// Calculation:
// .ELEV4 = cos (-1 * Omega * DegToRad)
// //.ELEV4 = cos (-5.23 * DegToRad) // gets to 82 degrees on the elevation
// //.ELEV4 = cos (-.034) // from hand calculations
FCL_ELEV4 = cos(flt_mul(flt_mul(-1, FCV_OMEGA), FCV_DEGTORAD));

```

```

// Calculation

// Calculation:

// .ELEV5 = cos (Latitude * DegToRad)
FCL_ELEV5 = cos(flt_mul(FCV_LATITUDE, FCV_DEGTORAD));

// Calculation

// Calculation:

// Elevation = RadToDegree * (asin (.ELEV1 * .ELEV2 + .ELEV3 * .ELEV4 * .ELEV5))
FCV_ELEVATION = flt_mul(FCV_RADTODEGREE, (asin(flt_add(flt_mul(FCL_ELEV1, FCL_ELEV2),
flt_mul(flt_mul(FCL_ELEV3, FCL_ELEV4), FCL_ELEV5))))));

//Local variable definitions

#undef FCLsz_STRINGS
}

/*=-----=*
Use :There is an error here compared to
: http://www.esrl.noaa.gov/gmd/grad/solcalc/azel.html
:Not sure of the severity.
:
:Returns : MX_FLOAT
\*=-----=*
MX_FLOAT FCM_Declination()
{
//Local variable definitions
#define FCLsz_STRINGS 20

```

```

MX_CHAR FCL_STRINGS[FCLsz_STRINGS];

MX_FLOAT FCR_RETVAL;

// Calculation

// Calculation:

// Declination = RadToDegree * asin (0.39795 * cos (0.98563 * (DayOfYear - 173) * DegToRad))

FCV_DECLINATION = flt_mul(FCV_RADTODEGREE, asin(flt_mul(0.39795, cos(flt_mul(flt_mul(0.98563,
flt_fromi((FCV_DAYOFYEAR - 173))), FCV_DEGTORAD))))));

#if 0 // Disabled code

// Calculation

// Calculation:

// .StringS = FloatToString$ (Declination)

FCI_FLOAT_TO_STRING(FCV_DECLINATION, FCV_PRECISION, FCL_STRINGS,20);

#endif // #if 0: Disabled code

#if 0 // Disabled code

// Call Component Macro

// Call Component Macro: LCD1::PrintString(.StringS)

FCD_04071_LCD1__PrintString(FCL_STRINGS, FCLsz_STRINGS);

#endif // #if 0: Disabled code

#if 0 // Disabled code

// Delay

// Delay: 4 s

```



```

FCI_DELAYBYTE_S(4);

#endif // #if 0: Disabled code

#if 0 // Disabled code

// CLS

// Call Component Macro: LCD1::Clear()

FCD_04071_LCD1__Clear();

#endif // #if 0: Disabled code

return (FCR_RETVAL);

//Local variable definitions

#undef FCLsz_STRINGS
}

/*-----*\

Use :

\*-----*/

void FCM_Azimuth()

{

//Local variable definitions

#define FCLsz_STRINGS 4

MX_CHAR FCL_STRINGS[FCLsz_STRINGS];

MX_FLOAT FCL_AZ1;

MX_FLOAT FCL_AZ2;

```

```
MX_FLOAT FCL_AZ3;
```

```
MX_FLOAT FCL_AZ4;
```

```
MX_FLOAT FCL_AZ5;
```

```
MX_FLOAT FCL_AZ6;
```

```
// Calculation
```

```
// Calculation:
```

```
// .Az1 = sin (Declination * DegToRad)
```

```
FCL_AZ1 = sin(flt_mul(FCV_DECLINATION, FCV_DEGTORAD));
```

```
// Calculation
```

```
// Calculation:
```

```
// .Az2 = cos (Latitude * DegToRad)
```

```
FCL_AZ2 = cos(flt_mul(FCV_LATITUDE, FCV_DEGTORAD));
```

```
// Calculation
```

```
// Calculation:
```

```
// .Az3 = cos (Declination * DegToRad)
```

```
FCL_AZ3 = cos(flt_mul(FCV_DECLINATION, FCV_DEGTORAD));
```

```
// Calculation
```

```
// Calculation:
```

```
// .Az4 = cos (Omega * DegToRad)
```

```
FCL_AZ4 = cos(flt_mul(FCV_OMEGA, FCV_DEGTORAD));
```

```

// Calculation

// Calculation:

// .Az5 = sin (Latitude * DegToRad)
FCL_AZ5 = sin(flt_mul(FCV_LATITUDE, FCV_DEGTORAD));

// Calculation

// Calculation:

// .Az6 = cos (Elevation * DegToRad)
FCL_AZ6 = cos(flt_mul(FCV_ELEVATION, FCV_DEGTORAD));

// Calculation

// Calculation:

// Azimuth = RadToDegree * acos ((.Az1 * .Az2 - .Az3 * .Az4 * .Az5) / .Az6)
FCV_AZIMUTH = flt_mul(FCV_RADTODEGREE, acos(flt_div((flt_sub(flt_mul(FCL_AZ1, FCL_AZ2),
flt_mul(flt_mul(FCL_AZ3, FCL_AZ4), FCL_AZ5))), FCL_AZ6)));

// Decision

// Decision: sin (Omega * DegToRad) > 0?
if (flt_gt(sin(flt_mul(FCV_OMEGA, FCV_DEGTORAD)), 0))
{

// Calculation

// Calculation:

// Azimuth = 360.0 - Azimuth
FCV_AZIMUTH = flt_sub(360.0, FCV_AZIMUTH);

```

```
// } else {  
  
}  
  
// "Month-Azim-Elev"  
// Call Component Macro: LCD1::PrintString("Month-Azim-Elev")  
FCD_04071_LCD1__PrintString("Month-Azim-Elev", 16);  
  
// Cursor 0,1  
// Call Component Macro: LCD1::Cursor(0, 1)  
FCD_04071_LCD1__Cursor(0, 1);  
  
// Call Component Macro  
// Call Component Macro: LCD1::PrintNumber(Count)  
FCD_04071_LCD1__PrintNumber(FCV_COUNT);  
  
#if 0 // Disabled code  
// Calculation  
// Calculation:  
// .StringS = ToString$ (Month)  
FCI_TOSTRING(FCV_MONTH, FCL_STRINGS,4);  
  
#endif // #if 0: Disabled code  
  
#if 0 // Disabled code
```

```
// Printr Azimuth

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

#endif // #if 0: Disabled code

// "-"

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

// Calculation
// Calculation:
// .StringS = FloatToString$ (Azimuth)
FCI_FLOAT_TO_STRING(FCV_AZIMUTH, FCV_PRECISION, FCL_STRINGS,4);

// Print Month
// Call Component Macro: LCD1::PrintNumber(Azimuth)
FCD_04071_LCD1__PrintNumber(flt_toi(FCV_AZIMUTH));

// "-"

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

// Calculation
// Calculation:
// .StringS = FloatToString$ (Elevation)
```

```

FCI_FLOAT_TO_STRING(FCV_ELEVATION, FCV_PRECISION, FCL_STRINGS,4);

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

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

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

//Local variable definitions
#undef FCLsz_STRINGS
}

/*=====*\
Use :Compared to the NOAA calculation, this is close, but not dead on.
:Considering that it is minutes, and no more than 15, the error is tolerable.
:PowerFromTheSun has a more accurate implementation.
\*=====*/

void FCM_EoT()
{

```

```

//Local variable definitions

MX_FLOAT FCL_ANGLEFORDAYOFYEAR = (0.0);

#define FCLsz_STRINGS 20

MX_CHAR FCL_STRINGS[FCLsz_STRINGS];

// Calculation

// Calculation:

// .AngleForDayOfYear = 360.0 * (DayOfYear - 1.0) / 365.242

// EoT = 0.258 * cos (.AngleForDayOfYear * DegToRad) - 7.416 * sin (.AngleForDayOfYear *
DegToRad) - 3.648 * cos (2 * .AngleForDayOfYear * DegToRad) - 9.228 * sin (2 * .AngleForDayOfYear *
DegToRad)

FCL_ANGLEFORDAYOFYEAR = flt_div(flt_mul(360.0, (flt_sub(flt_fromi(FCV_DAYOFYEAR), 1.0))),
365.242);

FCV_EOT = flt_sub(flt_sub(flt_sub(flt_mul(0.258, cos(flt_mul(FCL_ANGLEFORDAYOFYEAR,
FCV_DEGTORAD))), flt_mul(7.416, sin(flt_mul(FCL_ANGLEFORDAYOFYEAR, FCV_DEGTORAD)))),
flt_mul(3.648, cos(flt_mul(flt_mul(2, FCL_ANGLEFORDAYOFYEAR), FCV_DEGTORAD))), flt_mul(9.228,
sin(flt_mul(flt_mul(2, FCL_ANGLEFORDAYOFYEAR), FCV_DEGTORAD))));

#if 0 // Disabled code

// Calculation

// Calculation:

// .StringS = FloatToString$ (Declination)

FCI_FLOAT_TO_STRING(FCV_DECLINATION, FCV_PRECISION, FCL_STRINGS,20);

#endif // #if 0: Disabled code

#if 0 // Disabled code

// Pring Declination

```

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

#endif // #if 0: Disabled code

#if 0 // Disabled code

// Delay

// Delay: 3 s

FCI_DELAYBYTE_S(3);

#endif // #if 0: Disabled code

#if 0 // Disabled code

// CLS

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

#endif // #if 0: Disabled code

//Local variable definitions

#undef FCLsz_STRINGS
}

//Seems to work. Needs verification.

/*=====*\

Use :Main
```



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

```
int main()
```

```
{
```

```
    MCUSR=0x00;
```

```
    // LCD Start
```

```
    // Call Component Macro: LCD1::Start()
```

```
    FCD_04071_LCD1__Start();
```

```
    // Calculation
```

```
    // Calculation:
```

```
    // DayOfMonth = 6 //15
```

```
    // Month = 1
```

```
    // Hour = 9
```

```
    // Minute = 23
```

```
    // Latitude = 29.75 //48.0 + 49.0 / 60.0
```

```
    // Longitude = -95.33 //-1 * (2.0 + 17.0 / 60.0 + 23.0 / (60.0 * 60.0))
```

```
    FCV_DAYOFMONTH = 6;
```

```
    FCV_MONTH = 1;
```

```
    FCV_HOUR = 9;
```

```
    FCV_MINUTE = 23;
```

```
    FCV_LATITUDE = 29.75;
```

```
    FCV_LONGITUDE = -95.33;
```

```
// Loop
// Loop: While Count > 12
while (!(FCV_COUNT > 12))
{

    // CLS

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

    // Call Day Of Year
    // Call Macro: DayOfYear=DayOfYear(Month, DayOfMonth)
    FCV_DAYOFYEAR = FCM_DayOfYear(FCV_MONTH, FCV_DAYOFMONTH);

    // Call EoT
    // Call Macro: EoT()
    FCM_EoT();

    // Call Declination
    // Call Macro: Declination()
    FCM_Declination();

    // Call SolarHourAngle
    // Call Macro: SolarHourAngle()
    FCM_SolarHourAngle();
```

```
// Call Elevation
// Call Macro: Elevation()
FCM_Elevation();

// CallAzimuth
// Call Macro: Azimuth()
FCM_Azimuth();

// Calculation
// Calculation:
// Count = Count + 1
// //Month = Month + 1
// Hour = Hour + 1
FCV_COUNT = FCV_COUNT + 1;
FCV_HOUR = FCV_HOUR + 1;

#if 0 // Disabled code
// Delay
// Delay: 2 s
FCI_DELAYBYTE_S(2);

#endif // #if 0: Disabled code

}
```

```
mainendloop: goto mainendloop;  
}
```

```
/*=====*\nUse :Interrupt\n\*=====*/
```