Measuring and Displaying Angular Displacement in Real-Time Design Document

Contents

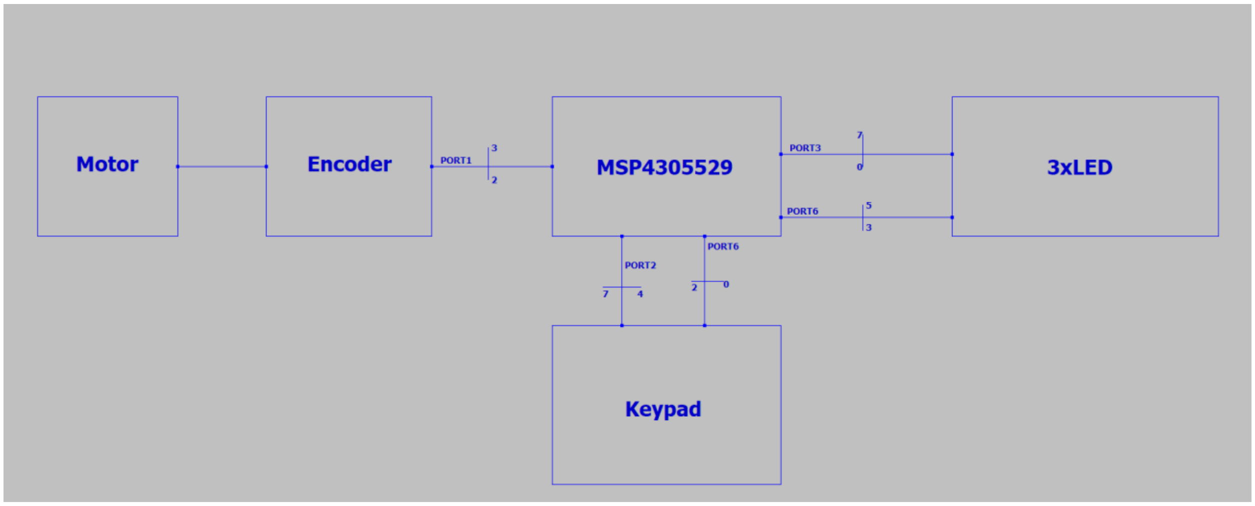
1 System Functional Description and Specification

The system functionality and hardware and software specifications are described below.

1.1 System Functional Description

This embedded system measures and displays, in real-time, the angular position (+/- 360 degrees) of a flywheel attached to the bench-top DC motors. The flywheel will be manually rotated by a user. The system will be configured and controlled through a matrix keypad.

1.2 System Block Diagram



1.3 System Operation

When the system is powered on it will default to persistently display 0 (single digit only) to the display with the current flywheel position. Clockwise rotation of flywheel (user facing the flywheel) will cause the display to update with the angular position in degrees up to a maximum of 360 degrees. Counter-clockwise rotation will update the display with negative angular displacement up to – 360 degrees . Negative angles will be denoted by using the decimal point of the most significant display digit. Multi-turn events will cause the display to roll over to 0. The display is persistent at power on unless a new mode is requested by the user or the power is turned off.

In addition to basic operation the system can be configured into numerous modes by a user through a matrix keypad. They are listed below:

1. ***PWR***. PWR can be toggled on/off.
2. ***CLR***. The display can be overwritten with a single 0.
3. ***FLASH***. The display can be flashed at a rate of 1 Hz.

The following are bonus modes:

B1. TBD or proposed by designers

1.4 System Specifications

System Specifications	
Display Angular Resolution	1 degree
System Angular Resolution	Calculated by User using Motor Name Plate data and Encoder
Encoder CHA Freq (max)	??
Display Refresh Rate	33.33Hz
Display Refresh Duty Cycle	33%
Display Segment Luminosity	5 mcd
Display Colour	red

1.5 Hardware Specification

1.5.1 Components

The following HW is permitted. No other components or IC's are permitted in the design.

- 1 MSP-EXP430F5529LP EVM
- 3 Digit Seven Segment Led Display. No integrated display drivers are permitted.
- Resistors (preferably resistor arrays used for busses or LED segment interfacing).
- Three transistors (BJT or FET type).
- Connection wires
- On-board proto-type wires must be solid core wire.
- Bench top DC motor with gear box and encoder.

1.5.2 Display HW Specification.

Not all LED segment displays are created equally. Look at the datasheet!!

Per Segment Luminosity	Relative Luminosity of 0.5
Refresh Frequency, $f_{refresh}$[1]	33.33Hz
Refresh Duty Cycle	33%

This is also listed in the system specification. Digit 3 is the most significant decimal digit, Digit 1 is the least significant digit.

[1] The refresh frequency of a **single** display.

1.5.3 Mode Control with Keypad

The systems modes can be controlled by the keypad. The table below describes the mode keys.

Mode	***Key***	***Description***
PWR	*	Toggle display power on/off. Angular measurement still active.
CLR	0	Display 0 to least significant digit. All other digits are off.
FLASH	#	Toggles the display in and out of flash mode. The Flash rate is 1 Hz and the angular measurement is still active.

1.5.4 MSP430F5529 EVM HW Specification.

A maximum of 15 output pins and 6 input pins of the MSP430F5529 can be used to interface the encoder, keypad and display.

f_{MCLK}	~ 1MHz
V_{CC}	3.3V
Maximum MSP430 Output Current, I_{OHMAX}[2]	+/- 125mA

1.5.5 MSP430F5529 EVM I/O Mapping

Interface	***MSP430 I/O PORT***	***Description***
SEG<7:0>	P3<7:0>	7 segment output port <a:g> plus dp.
DIGIT<2:0>	P6<5:3>	Anode/Cathode control outputs
CHAB<1:0>	P1<3:2>	Encoder CHA, CHB inputs
COL<3:0>	P2<7:4>	Column output port. Can be <2:0>
ROW<3:0>	P6<2:0>	Row input port. Can be <2:0>

[2] See notes 2,3 in Tables 5.10 and 5.11 of MSP430 Data Sheet. This is a spec for ALL digital OUTPUT currents combined. It is not average current, it is instantaneous current. Why have I had to increase above 125mA.

1.6 Firmware Specification

The FW driver responds to the state change of the inputs, updates the global *sysState* variables and other global variables and consequently updates the display. ALL inputs must activate interrupts (adhering to priority). See the discussion of Global Variables below for more detail.

1.6.1 Target Language

The firmware will be implemented in C and targeted for the MSP430F5529 MCU.

1.6.2 C Module Requirement

All source code must be partitioned into C modules. The modules are defined below.

Within the .h file of each module you must declare any #define constants and macros. Hard coding literal constants is not permitted.

1.6.3 sevenSegDisplay Module

This C module will:

- Output the multiplexed display signals at the required refresh rate and duty cycle
- Implement the display modes PWR, HOME and FLASH.
- Decode the decimal angular position into 7 segment codes.

1.6.3.1 Module Constants and Macros

These belong in *sevenSegDisplay.h* before the structure declaration.

#define FRQ_RFRSH 100 // refresh frequency of single display

#define SEG P3OUT // Segment Port for 7 segment interface

#define DIGIT P5OUT // you choose an output port for the digit display control.

#define DISP1 BIT0 // display 1 on, display 2 off

#define DISP2 BIT1

#define DISP3 BIT2

// there's probably others

1.6.3.2 SEVEN_SEG_DISP Structure

Control and updates of the seven segment display must use the structure `SEVEN_SEG_DISP`.

```
typedef struct SEVEN_SEG_DISP {
        char decDigit;  // base 10 decimal digit -9 to + 9
        unsigned char binSegCode; // 7 segment code <a:g> plus dp in BIT7
        unsigned char ctrl  // single bit indicating on/off state of display

    } SEVEN_SEG_DISP; // define this in the sevenSegDisplay .h file

SEVEN_SEG_DISP sevenSegDisp[3]; //for three displays create an array in the client file.
```

1.6.3.3 sevenSegDispInit function

/* **

* Initialize sevenSegDisp variable by setting display-> decDigit to 0, display-> binSegCode to 0x0. SEG and DIGIT are initialized to outputs. Displays are turned off. display->ctrl is not affected

* arguments:

* `display` – address of one of the three displays

* return: *void*

* Author:

* Date:

***/

sevenSegDispInit (`SEVEN_SEG_DISP display)`

1.6.3.4 decTo7Seg function

/* **

* Convert display->decDigit to display->binSegCode

display->ctrl is not affected

* arguments:

* `display` – address of one of the three displays

* return: *void*

* Author:

* Date:

***/

void *decto7Seg* (`SEVEN_SEG_DISP *` ***display*)**

1.6.3.5 dispRefresh function

/* **

* Output SEG port with display->binSegCode and assert DIGIT port with display->ctrl. display->ctrl is updated.

* arguments:

* `display` – address of one of the three displays

* `dispIndex` – index 0 to 2 refers to display 1 to 3. Used to update correct bit within DIGIT port

* return: *void*

* Author:

* Date:

**/

The refresh rate is interrupt driven and is controlled by updating the *REFRESH bit within the sysState global variable.*

void *dispRefresh*(`SEVEN_SEG_DISP *` ***display,*** *unsigned char* ***dispIndex)***

1.6.3.6 dispRefresh pseudo code

```
If (Refresh.bit == 1) {
    SEG |= display[dispIndex]->binSegCode;
    DIGIT |= display[dispIndex]->ctrl;
    if (!(display->cntr & DISP3)) {
            display->ctr << 1;
    }
    else {
        display->cntr = DISP1;
    }
    REFRESH.BIT = 0;
}
```

There are some other function(s, ality) that needs to be defined here. For example, how does the display flash? How does it turn off? How does the display zero? This is all fairly straight forward once we have the basic functions defined above.

1.6.4 quadEncDecode Module

quadEncDecode module contains the C functions to decode a quadrature encoder into a 16 bit counter. It does NOT compute the angle.

1.6.4.1 Module Constants and Macros

These belong in *quadEncDecode.h* before the structure declaration.

#define CHAB_PORT PxIN // figure it out

#define CHA BIT1 // CHA bit position

#define CHB BIT0 // CHB bit position

1.6.4.2 QUAD_ENC_DECODER structure

```
typedef struct QUAD_ENC_DECODER {
        unsigned char channelState[4]  // current and previous channel A,B states
       // channelState[0] > CHB_PREV, channelState[1] > CHA_PREV
       // channelState[2] > CHB_PREV, channelState[3] > CHA_PREV

        int postCount; // 16 bit signed position counter updated every quadrature event


    } QUAD_ENC_DECODER; // define this in the quadEncDecode.h file
```

QUAD_ENC_DECODER qEdecoder; // will need to be global, make this declaration in quadEncDecode.h.

See below in Global Variables for how to link global variables across multiple C modules.

1.6.4.3 qEncDecode function

/* **

* qEncDecode will be called from within an ISR when the encoder triggers an interrupt on CHAB_PORT.

It updates the channelState array and increments or decrements posCount.

* arguments:

* none. It accesses the global variable qEdecoder and updates the states and posCount

* return: *void*

* Author:

* Date:

***/

void qEncDecode ();

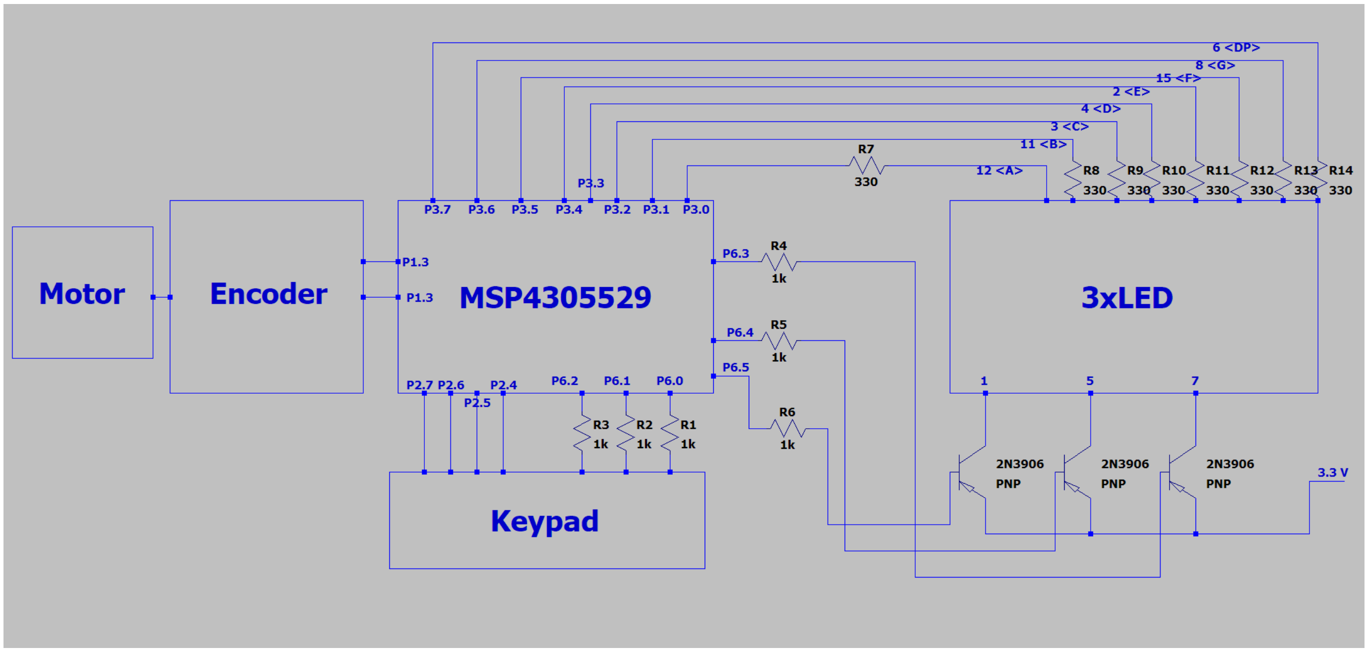
1.6.4.4 qEncDecode pseudo code

```
qEncDecode

int channelCntr = 0; // integer to control the position of
channelState
Psedo code
if (P1IN & CHA1) {   // channel A is being triggered
    qEdecoder.channelState[3] = qEdecoder.channelState[1];    //
previous state into the last position
    qEdecoder.channelState[1] = CHA;
    channelCntr++;
    P1IFG &= CHA | CHB;
}
if (P1IN & CHA2) { // channel B is being triggered
    qEdecoder.channelState[2] = qEdecoder.channelState[0];
    qEdecoder.channelState[0] = CHB;
    channelCntr++;
    P1IFG &= CHA | CHB;
}
if (cahnnelCntr == 3) {     // encoder has made 4 counts
    qEdecoder.postCount++;
}
```

2 Appendix. Schematic and Source Code

2.1 Schematic



2.2 Display Driver Calculations

1) Calculations for luminocity of a diode

Assume:

$V_{diode} = \ 1.6\,V\ (red\ diode), and\ I_{diode} = 5\,mA$

$V_{cc} = 3.3\,V$

$$\therefore R_F = \frac{(V_{CC} - V_{diode})}{I_{diodee}} = \frac{(3.3\,V - 1.6\,V)}{5\,mA} = 340\,\Omega.\text{Choose } 330\,\Omega$$

We choose 560 Ω, because it is one of the E12 resistors

2) Calculations for a segment current

The total current through a segment is $I_{total} = I_C = 40\,mA$, where I_C is the collector current.

To make sure that the BJT transistor is in the satureation mode, we use assume that the bas current is

$$I_B > \left(\frac{I_C}{\beta}\right) \times 1.2$$

and β is minimum.

The 1.2 factor is an additional 20% to make 100% sure we are in the saturation region.

So,

$$I_B > \frac{40\ mA}{30} \times 1.2 > 1.6\ mA$$

$\mathrm{I_{OH}}\ at\ V_{cc} = 3.0\ V$ is 6 mA which is more than enough to turn the transistor on.

$$R_{EB} = \frac{(V_{CC} - 0.6\ V - V_{BE})}{I_B} = \frac{(3.0\ V - 0.6\ V - 0.7\ V)}{1.6\ mA} = 1\ \mathrm{k\Omega}.$$