

Programming the MSP430 for Data Acquisition and Processing

Application Note

Nick Mancuso

Overview

The MSP430 microcontroller is a 16-bit low power solution for data acquisition and processing. By properly utilizing a combination of digital I/O, analog I/O, and on chip timers, a user can capture and record a series of phenomena. This application note is highly code based, with multiple in line comments to relate the code back to the section summaries. This application note is designed to be used in conjunction with the TI MSP430 User Guide, which details many of these registers and code ideas in a technical manner. The User Guide supplies very few actual code examples, making this Application Note extremely useful as a quick reference for working code.

Digital Inputs and Outputs

The most basic form of data acquisition is digital input and digital output. To setup a pin as a digital input or output, one must configure that pin to accept or produce a voltage. In the following table, Figure 1, each of the important registers are listed alongside their functions. In the code that follows, it should be noted that BIT5 denotes a '1' on the registers 5th bit, and ~BIT5 denotes a '0' on the registers 5th bit.

[1]

Register	0	1
PxSEL	Input/Output	Peripheral Module
PxIN	Input is '0'	Input is '1'
PxOUT	Output is '0'	Output is '1'
PxDIR	Pin is Input	Pin is Output

Figure 1. Input/Output Registers

In the following example, pin P1.5 is configured as an input. A loop is constantly checking the pin for an input voltage. For an input to read '1', the voltage must be between 2 and 5 Volts. Between 0 and 1.3 Volts the input will be read as '0'. Between 1.3 and 2 Volts, the digital value of the bit will not change.

```
void main(void) {
    P1SEL = ~BIT5; //P1.5 is I/O
    P1DIR = ~BIT5; //P1.5 is input

    //main loop
    while (1) {
        if (P1IN & BIT5 == BIT5) {
            //input detected, execute logic
        }
        else {
            //input not detected, execute logic
        }
    } //end while
} //end main
```

Figure 2. Basic Input [1]

In the following example, pin P1.6, an on-board LED, is configured as an output. This is in addition to pin P1.5 as an output. The P1.6 LED will light up if the input at pin P1.5 is high.

```
void main(void) {
    P1SEL = ~BIT5; //P1.5 is I/O
    P1DIR = ~BIT5; //P1.5 is input
    P1SEL = ~BIT6; //P1.6 is I/O
    P1DIR = BIT6; //P1.6 is output
    P1OUT = ~BIT6; //P1.6 is off by default

    //main loop
    while (1) {
        _delay_cycles(100); //delay loop between input checks
        if (P1IN & BIT5 == BIT5) {
            //input detected, turn on LED
            P1OUT |= BIT6;
        }
        else {
            //input not detected, turn off LED
            P1OUT &= ~BIT6;
        }
    } //end while
} //end main
```

Figure 3. Basic Input and Output with LED [1]

Using the 10-bit Analog-to-Digital Converter

If the voltage supplied to a pin is actually going to be a dynamic signal, or you want to be able to execute different logic based on the signal strength at an input pin, using the ACD10 register and reading input values as a voltage can be a powerful way to manage Inputs.

In order to use the ADC10 on the MSP430, you must configure your clocks, set up your input pins, with PxSEL and PxDIR, and assign the appropriate values to the ADC10 control registers. The following code shows an example configuration with input on pin P1.5, and executing logic based on the voltage being above 1.5 Volts. When the voltage is above 1.5 Volts, the green LED is off and the red

LED is on. When the voltage is below 1.5 Volts, the red LED is off and the green LED is on.

```
#define LED0          BIT0 //red LED
#define LED1          BIT6 //green LED
unsigned int value=0;
float input_voltage = 0;

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;          // Disable Watchdog timer
    BCSCCTL1 = CALBC1_16MHZ;           // Set Clock Frequencies (16 MHz)
    DCOCTL = CALDCO_16MHZ;
    P1DIR |= LED0 + LED1;              //
    P1SEL |= BIT5;                     //ADC Input pin P1.5
    P1OUT &= ~(LED0 + LED1);

    ADC10CTL1 = INCH_5;                // Channel 5
    ADC10CTL0 = SREF_1 + ADC10SHT_3 + ADC10ON + ADC10IE + REFON + REF2_5V;
    ADC10AE0 |= BIT5;                  //P1.5 ADC option

    value=0;
    while(1)
    {
        __delay_cycles(2000);          // Delay for ADC
        ADC10CTL0 |= ENC + ADC10SC;     // Sampling and conversion start
        value = ADC10MEM;
        input_voltage = (float)value*2.5/1023; //Convert back to voltage
        if (input_voltage > 1.5)
        {
            P1OUT &= ~LED1;
            P1OUT |= LED0;
        }
        else //input_voltage less than or equal to 1.5 V
        {
            P1OUT &= ~LED0;
            P1OUT |= LED1;

            __delay_cycles(10000);
        }
    } //end while
} //end main
```

Figure 4. Analog-to-Digital Conversion Example [1]

MSP430 Timers and Timing Applications

Timing is very important in data processing. The MSP430 provides counting timers that can count clock cycles. By setting the clock speed and writing the interrupt vectors, you can easily keep track of events or phenomena with relative time-stamps. In the following example, a timer will count up until 31249, and then reset. Upon reset, the interrupt vector will be called and a time tick will be added. By counting the ticks, and multiplying by the time unit, you can easily create a real-time clock and timing system. In this case, the time unit is .25 seconds. By selecting a clock frequency and counter constant, you can customize the clock's accuracy.

```
float totalSeconds =0;
long ticks =0;
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;           // Disable Watchdog timer
    BCSCCTL1 = CALBC1_1MHZ;             // Set Clock Frequencies (1 MHz)
    DCOCTL = CALDCO_1MHZ;

    P1DIR |= BIT2;                      // P1.2 to output
    P1SEL |= BIT2;                      // P1.2 to TA0.1

    TACTL = TASSEL_2 + ID_3 + MC_1 + TACLRL; // select SMCLK/8, up mode
    TACCTL0 = CCIE; // enable interrupts
    TACCR0 = 31249; // .25s cycle with 1MHZ clk and /8 divider
    _enable_interrupt();

    while(1) {
        float totalSeconds = .25 * tickCount; //update real-time timer with .25s accuracy
    } //end while

} //end main

// timerA interrupt for CCR0
#pragma vector = TIMERA0_VECTOR
__interrupt void CCR0_ISR (void) {
    ticks++; // increment ticks
} //end timer vector
```

Figure 5. Real-Time Clock Example [1]

Conclusion

By combining digital I/O, analog I/O, and timing systems, you can setup any MSP430 to acquire and process data. Design Team 3's Ballistic Chronograph project utilizes all three of these systems, with clock frequencies and registers selected specifically for the application, to accurately detect and time projectiles crossing through the chronograph. The references section includes the MSP430 User Guide, as well as a collection of the MSP430 tutorials that may prove useful to the reader. [1] [2]

References

[1] Texas Instruments, ed. *MSP430x2xx Family User's Guide - SLAU144J*. Rev. J ed. N.p.: Texas Instruments, n.d. June-July 2013. Web. 26 Mar. 2014. <<http://www.ti.com/lit/ug/slau144j/slau144j.pdf>>.

[2] "MSP430 LaunchPad Tutorials." - *Texas Instruments Wiki*. Texas Instruments, n.d. Web. 28 Mar. 2014. <http://processors.wiki.ti.com/index.php/MSP430_LaunchPad_Tutorials>.