

Application Note

Caitlin Slicker

April 3rd, 2015

Abstract:

Team 10 is implementing a system to monitor tool life in a 6-spindle automatic lathe based on the frequency output of an accelerometer. As the frequency of the accelerometer output increases, tool life is diminishing. To analyze the frequency of the signal, Team 10 has opted to use a programmable band-pass filter. This application note details how to use an MSP430 to program the MAX262 adjustable band pass filter IC, how the output of the filter will be compared to determine if there has been a diminished gain, and how to trigger an interrupt service routine based on this output to reprogram the filter.

Keywords: tool monitoring, microcontroller, embedded C, MSP430, programmable filter, MAX262, ISR, interrupt service routine, GPIO

Team 10 has been tasked with developing a tool monitoring system for use with a 6-spindle automatic New Britain lathe. The system utilizes an industrial accelerometer to measure vibration on the tool arm while machining a part. The frequency of the output will vary with tool life. To process this signal, an adjustable filter IC will be utilized to determine the frequency range of the output. The primary focus of this application note will be to discuss the operation of the programmable filter and the microcontroller utilized to interface with it and the final output of the system.

MAX262

The MAX262 is an adjustable analog filter manufactured by Maxim Integrated. This chip has two independent, second-order switched capacitor filters, both of which have adjustable center frequency, Q, and filter operating modes.

Calculations To create a filter, a center frequency and Q value, and input clock must be specified. f_{CLK} is the input clock for the filter. f_0 is the center frequency, which is a function of the input clock rate, 6-bit f_0 control word, and operating mode. In mode 1, $\frac{f_{CLK}}{f_0} = \frac{(26*N)\pi}{2}$. Therefore, from the input clock and desired center frequency, N can be calculated. This value can then be used to determine, via. a table in the datasheet, the correct 6-bit code (F5:F0) to program to set the desired center frequency.

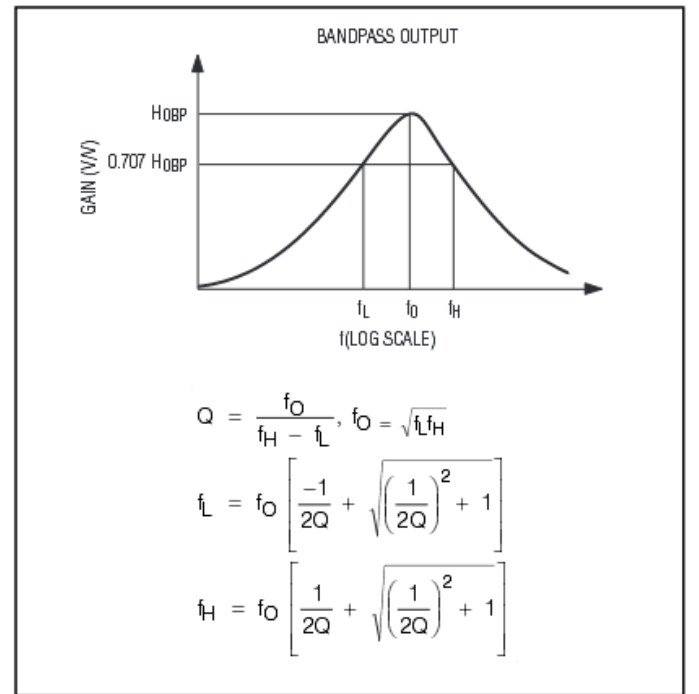


Figure 1: Diagram of center frequency and Q

Finding a Q value is done similarly. First, if the corners of the desired bandpass are known, Q can be calculated using the equations in Figure 1. Next, the program code to set the Q value can be found. In mode 1, $= \frac{64}{128-N}$. After solving for N, the 7-bit program code (Q6:Q0) can be determined via a table in the data sheet.

Programming Once the program codes for f_0 and Q have been determined, the chip can be programmed. Rather than a serial communication bus, the MAX262 chip is programmed through a series of input pins and a write enable line. Inputs include two data bits (D0:D1), four address bits (A0:A3), and an input clock. Parameters for each of the two filters can be set by writing data to various specified address locations. Figure 2 shows the pinout of the MAX262 chip and Figure 3 specifies the addresses each bit of data should be sent to to create the desired filter.

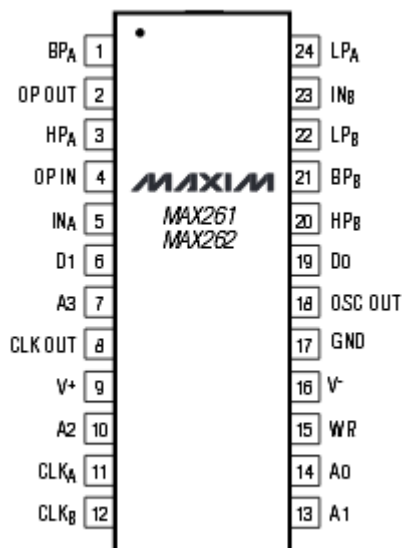


Figure 2: MAX262 Pinout

DATA BIT		ADDRESS				LOCATION
D0	D1	A3	A2	A1	A0	
FILTER A						
M0A	M1A	0	0	0	0	0
F0A	F1A	0	0	0	1	1
F2A	F3A	0	0	1	0	2
F4A	F5A	0	0	1	1	3
Q0A	Q1A	0	1	0	0	4
Q2A	Q3A	0	1	0	1	5
Q4A	Q5A	0	1	1	0	6
Q6A		0	1	1	1	7
FILTER B						
M0B	M1B	1	0	0	0	8
F0B	F1B	1	0	0	1	9
F2B	F3B	1	0	1	0	10
F4B	F5B	1	0	1	1	11
Q0B	Q1B	1	1	0	0	12
Q2B	Q3B	1	1	0	1	13
Q4B	Q5B	1	1	1	0	14
Q6B		1	1	1	1	15

Figure 3: Table of data and address locations for each filter

In Figure 3, M1:M0 refer to the mode, F5:F0 refer to the center frequency, and Q6:Q0 refer to the Q value for each respective filter. Two bits of data can be sent in each transaction and Figure 4 shows the timing of each transaction. t_{WR} refers to the $\sim WR$ pulse width, t_{AS} the address setup time, t_{AH} the address hold time, t_{DS} the data set time, and t_{DH} the data hold time. To program the chip, 7 GPIO pins on a microcontroller are needed. Each of the data and address bits are set, then the select line is set low for a minimum of $t_{WR}=250\text{ns}$.

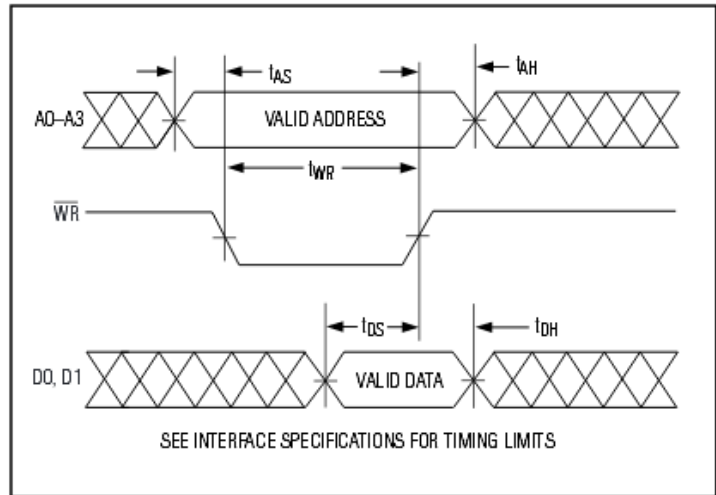


Figure 4: Timing Diagram for MAX262

MSP430

To program the chip, interpret the filter output, and output a signal to the PLC based on the tool life. Team 10 has chosen to use the MSP430, a low cost, low power microcontroller produced by Texas Instruments. The MSP430 can be programmed using Code Composer Studio, an Eclipse based IDE produced by Texas Instruments to be used with their embedded products. Through this IDE, there are libraries included that allow the user to address registers directly while writing embedded C. Figure 5, shows the pinout of the MSP430G2553 20-pin dip

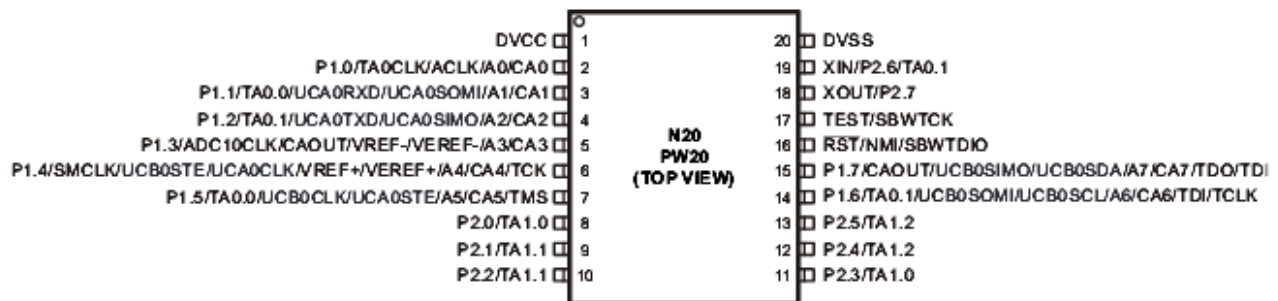


Figure 5: MSP430G2553 20-pin Dip Package Pinout

package. Each pin has a number of programmable functions that include PWM outputs, two serial data busses, and a 10-bit ADC, among other things. Alternatively, they can be configured as GPIOs, which is required for the programming of the MAX262 chip.

To configure pins for a particular function, a series of registers must be set. On this microcontroller, there are two ports, each with 8 pins. Accordingly, there are two sets of configuration registers. Each register, when referenced in the data sheet, is called, for example, PxSEL. The x is a placeholder for the port that is being referenced. Each register is 8-bits long, with each bit referring to one pin. The naming of each pin in Figure 5 assists in this process. For example, a pin called P1.2 is Port 1, pin 2. Therefore, to set a register with regards to this pin, the x mentioned earlier would be 1, as in P1SEL, and the second bit in the register would be changed.

To configure the pins as GPIOs, first registers PxSEL and PxSEL2 must be set. These two registers allow the user to select between a pin's many possible functions. As the table in Figure 6 demonstrates, both registers must be set to 0 for all pins to be GPIOs.

PxSEL2	PxSEL	Pin Function
0	0	I/O function is selected.
0	1	Primary peripheral module function is selected.
1	0	Reserved. See device-specific data sheet.
1	1	Secondary peripheral module function is selected.

Figure 6: MSP430 PxSEL and PxSEL2 Functions

Next, it is necessary to define the pins as either inputs or outputs. To program the MAX262, 7 pins must be set as outputs. This is done using the PxDIR register. In this register, if a bit=0, the corresponding pin is an input, and if a bit=1, the corresponding pin is an output. For example, to set P1.1 to an output and all other pins on the port to inputs, the user would set P1DIR=0x01. To read from or write to a pin, PxIN and PxOUT are used. To read the value of an

input pin, the user needs only to check the value of PxIN. To set an output, write that value to PxOUT.

Programming MAX262 To program the MAX262, six output pins must be written to, then the seventh write enable line driven low. A small delay must be added that is at least t_{WR} long, then the write enable line can be brought back high. After this is completed, another transaction can begin. Implemented in a function, this series of events could be implemented as follows:

```
void set_filter(int D0, int D1, int A3, int A2, int A1, int A0)
{
    P2OUT = (D0 << 5) + (D1 << 4) + (A3 << 3) + (A2 << 2) + (A1 << 1) + A0;
    P1OUT &= ~BIT6;
    _delay_cycles(10);
    P1OUT |= BIT6;
}
```

Data bits D1:D0 and address bits A3:A0 are taken as parameters. Shifting each bit and adding them has the effect of concatenation. For example, if all bits equaled 1, adding them in a normal way would produce 6, but in this case produces 111111. This example code assumes that P2.0 through P2.5 are output pins that are wired to the appropriate pins on the MAX262 chip. After setting the desired bits to the output pins, the write enable line can be brought low. This code assumes that P1.6 is being used as the write enable line, so setting P1OUT to ~BIT6, sets that bit low. BIT6 is a part of the MSP430 library which is equivalent to an 8-bit number with only the 6th bit high. Setting a register equal to BIT6 is equivalent to 0100 0000, or 0x40. Additionally, a technique called bit-masking is being used here. Particularly when implementing an operation in a function, it is important to not overwrite more data in a register than is necessary. Instead of setting P1OUT = BIT6, which consequentially sets all other bits on P1OUT low, |= or &= can be used. &= is useful for setting a single bit low without affecting other values, while |= is useful for

setting a single bit high without affecting other values. After setting P1.6 low, a small delay is implemented with the function `_delay_cycles()`, then the line is brought high again, completing the transaction. Because only two bits can be sent at a time, eight transactions are necessary to set all of the values for one filter, as shown in Figure 3.

Filter Output Analysis To interpret the signal output of the filter without using a more complex DSP solution, Team 10 will be setting the band pass filter to a certain frequency range, then using a comparator that will check the amplitude of the filter output against a constant voltage equal to 3dB less than the maximum filter output. The circuit for this system is shown in Figure 7. The accelerometer that has

been chosen has an output of 0-10V, which will need to be stepped down before being filtered. The suggested supply voltage for the MAX262 chip is $\pm 5V$, and the maximum input signal voltage is 0.3V over the

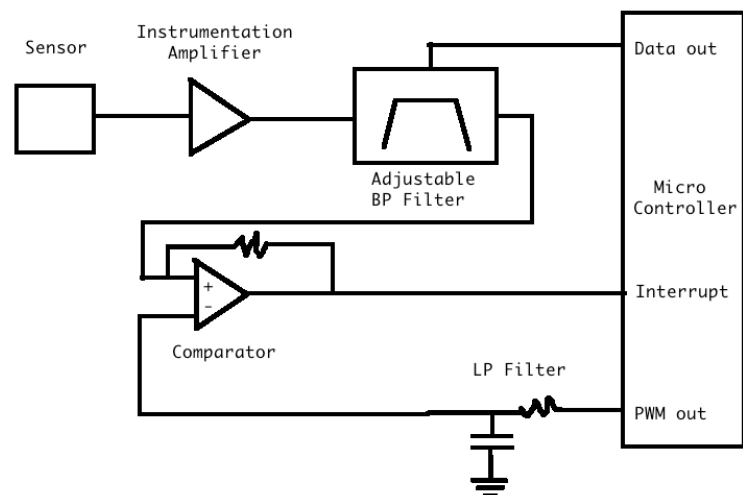


Figure 7: MAX262 Output Analysis

supply voltage on either the positive or negative bound, or $\pm 5.3V$ range. The output signal swing of the MAX262 chip is $\pm 4.75V$ when a 5V supply is used. A 3dB loss of signal would be equal to $\pm 4.75 \times 0.707 = \pm 3.36V$. If a microcontroller with a higher power supply is used, such as 5V voltage rail, it would be possible to produce a PWM output that more precisely obtained this value. However, the MSP430 is powered from a 3.3V voltage rail. 3.3V is reasonably similar to the 3dB loss of the filter output so in this case, the filter output will simply be compared to MSP430 power supply. Additionally, the output of the filter will need to be rectified in order to obtain a DC voltage to be compared.

The positive terminal of the comparator will be connected to the output of the rectified filter output and the negative terminal of the comparator will be connected to 3.3V. The comparator will have a voltage swing of 3.3V to ground, to produce an output appropriate for the MSP430. In this configuration, when the amplitude of the filter output is less than 3.3V, the output of the comparator will be 0V, and when the output of the filter surpasses 3.3V, the output of the comparator will be pulled to 3.3V.

Interrupts When the output of the comparator is pulled high, it should be understood that the sensor output has reached a certain frequency. As a result, the filter should be reprogrammed to test for another, higher frequency range. This is done with the use of an interrupt service routine, or ISR.

On the MSP430, each pin on ports P1 and P2 have interrupt capability, but all source a single interrupt vector for that port. The PxIFG register can be tested to determine for a particular port which pin is the source of the interrupt vector.

To configure a particular pin to have interrupt capability on the MSP430, first the pin must be set to an input and the register PxIE, or Port x Interrupt Enable, must be set. To set pin P1.4 as an input that triggers an interrupt, it can be set to an input by setting $P1DIR \&= \sim \text{BIT4}$, which sets the fourth bit to 0 (or alternatively, not setting it high in the first place). Next, the fourth bit of P1IE will need to be set to 1 to enable the interrupt on that pin, or $P1IE |= \text{BIT4}$.

Second, interrupts are only triggered on transitions, not static values. Register PxIES allows the user to configure the type of transition to trigger the interrupt on for the corresponding I/O pin. On the MSP430, setting a bit in PxIES equal to 0 will trigger an interrupt on a low-to-high transition, and setting it equal to 1 will trigger on a high-to-low transition. For this

application, it will be necessary to trigger an interrupt on P1.4 when a low-to-high transition occurs, so the user should set `P1IES &= ~BIT4`, or setting the fourth bit equal to 0. If the user is interested in triggering an interrupt on a high-to-low transition, it should be noted that they should also enable a pull-up resistor for that pin using `PxREN`. Setting of configuration registers should occur within the main function and, for this purpose, should look as follows:

```
P1IE |= BIT4;           // P1.4 interrupt enabled
P1IES &= ~BIT4;         // P1.4 lo/hi edge
P1IFG &= ~BIT4;         // P1.4 IFG cleared
```

Now that P1.4 is interrupt enabled for a low-to-high transition, when this event occurs, it will set a flag in `PxIFG`, as mentioned earlier. When this flag is set, the interrupt service routine for Port 1 will begin. This register is not read explicitly by the user's code in order to begin the interrupt, when a flag is set, the ISR will begin automatically. However, this register can be written to to trigger an interrupt in software. Outside of the main function, the ISR interrupt vector should be defined and should look similar to the following code:

```
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(//new filter parameters)
{
    set_filter(//new filter parameters);
}
```

The ISR vector is similar to a function in that the user can pass parameters. In this case, it will be necessary to reprogram the filter. The user can pass a pointer to the ISR vector and then call the function to reprogram the filter using these values. When the program returns the main function, the values stored at the pointer address can be changed again to be ready for the next calling of the ISR.

References

1. MSP430G2553 Data Sheet: <http://www.ti.com/lit/ds/symmlink/msp430g2553.pdf>
2. MSP430x2xx Family User Guide: <http://www.ti.com/lit/ug/slau144j/slau144j.pdf>
3. MAX262 Data Sheet: <http://datasheets.maximintegrated.com/en/ds/MAX260-MAX262.pdf>