04/07/23

Time: 7.5h

July 4, 2023 8:06 AM

Start 8:00am

Last week I emailed our Ximea contact for a formal quote as Nick had requested.
He replied last night and the email was forwarded to both Nick and Grant.

230703_Q...

The quote was ~$2500USD and the time to ship was ~4 days, however I believe that is when it leaves them not when it arrives.

For this week:

- Figure out the I2C on the PIC
- Figure out timer interrupts on the PIC
- Once Digi key order arrives, setup a system on the Zero 2W and install the camera drivers.

Timer Interrupts

24.6 Register Summary - Timer0

Address	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00 ... 0x0317	Reserved									
0x0318	TMR0L	7:0	TMR0L[7:0]							
0x0319	TMR0H	7:0	TMR0H[7:0]							
0x031A	T0CON0	7:0	EN		OUT	MD16	OUTPS[3:0]			
0x031B	T0CON1	7:0	CS[2:0]			ASYNC	CKPS[3:0]			

Looking at the timer module, it appears to be very simple to configure. The EN bit just enables the timer, the OUT bit is just an output of the timer, MD16 is to select 16bit or 8bit mode, OUTPS is the output post scale value (the post scale of the timer is what sets the interrupt). CS is the clock source, ASYNC is to choose synchronous or asynchronous, and CKPS is the clock prescaler value. I am wondering if both the prescaler and the postscaler are capable of changing the interrupt time. The datasheet directly said that the postscaler was used, but If the prescaler for the timer changes what the post scaler is scaling, than both should be able to set this.

Bits 3:0 – OUTPS[3:0] TMR0 Output Postscaler (Divider) Select

Value	Description
1111	1:16 Postscaler
1110	1:15 Postscaler
1101	1:14 Postscaler
1100	1:13 Postscaler
1011	1:12 Postscaler
1010	1:11 Postscaler
1001	1:10 Postscaler
1000	1:9 Postscaler
0111	1:8 Postscaler
0110	1:7 Postscaler
0101	1:6 Postscaler
0100	1:5 Postscaler
0011	1:4 Postscaler
0010	1:3 Postscaler
0001	1:2 Postscaler
0000	1:1 Postscaler

Bits 3:0 – CKPS[3:0] Prescaler Rate Select

Value	Description
1111	1:32768
1110	1:16384
1101	1:8192
1100	1:4096
1011	1:2048
1010	1:1024
1001	1:512
1000	1:256
0111	1:128
0110	1:64
0101	1:32
0100	1:16
0011	1:8
0010	1:4
0001	1:2
0000	1:1

Bits 7:5 – CS[2:0] Timer0 Clock Source Select

Value	Description
111	CLC1_OUT
110	SOSC
101	MFINTOSC (500 kHz)
100	LFINTOSC
011	HFINTOSC
010	$F_{OSC}/4$
001	Pin selected by T0CKIPPS (Inverted)
000	Pin selected by T0CKIPPS (Non-inverted)

There is no option for the oscillator clock on this, unless CLC1_OUT is tied to the oscillator.

Timer 1 is able to use the Oscillator as the clock source.

25.13.3 TxCLK

Name: TxCLK
Address: 0x317,0x328

Timer Clock Source Selection Register

Bit	7	6	5	4	3	2	1	0
				CS[4:0]				
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

Bits 4:0 – CS[4:0] Timer Clock Source Selection

Table 25-4. Timer Clock Sources

CS	Clock Source	
	Timer1	Timer3
11111 – 10001	Reserved	
10000	CLC4_OUT	
01111	CLC3_OUT	
01110	CLC2_OUT	
01101	CLC1_OUT	
01100	TMR3_OUT	Reserved
01011	Reserved	TMR1_OUT
01010	TMR0_OUT	
01001	CLKREF_OUT	
01000	EXTOSC	
00111	SOSC	
00110	MFINTOSC (32 kHz)	
00101	MFINTOSC (500 kHz)	
00100	LFINTOSC	
00011	HFINTOSC	
00010	F_{OSC}	
00001	$F_{OSC}/4$	
00000	Pin selected by T1CKIPPS	Pin selected by T3CKIPPS

I have been trying to get a test script going for the timer interrupt. It does not appear that the timer is counting properly.

I have found that the High register is staying at 0xFF and the Low register is staying at 0x00.
I set it up using the SOSC clock which I don't really know what that is. Maybe it would be good to use the FOSC/4 since I at least know that is set/available. Maybe that is the issue it is not counting.

I changed the clock source and now the timer is actually counting properly.

The interrupt is working right now. It appears to be occuring every ~1.5ms.

I had put a for loop of 100 NOP();s to try to give some delay. With these pre/post scale values, the time in the interrupt is causing it not to occur at the proper times. Cause it is still in the in ISR when the next flag is set.

I removed the NOP(); so the ISR is only turning the pin on and off.

It is occuring every ~6us

I with 1MHz on the timer clock,

1/1e6=1e-6 = 1us

I think I have the timer interrupt happening too fast.

If I change it to be a postscale of 16, I should be able to see it happening every 16us reliably

This did not change the interrupt period at all, it is still the ~6us…

Maybe if I change the prescale it will have an effect?

I just went back to the datasheet, and the interrupt occurs when the timer counter rolls over, not changes.

Meaning that even with no prescale it should have to complete 2^16 cycles to shift, which would make the period dramatically slower

65536 cycles would be needed to trigger it, meaning a period of ~65ms should be seen.

The scale is too low to see the short time of the on/off command, so I need my for loop again to add a delay

I included 1000 NOP(); commands between on and off so that I would be able to see it at a larger scale. I found the time between the rising edges of the interrupts was ~66ms, which is what I expected.

I used the measure tool instead of cursor and it gave a period of 65.6ms, which is even more accurate.

Now I want to change the post scaler to 2 to see if that doubles.

That changed the period to 131.2ms
65.6*2=131.2

Now I want to reset the postscaler to 1 and change the prescaler to 2 to see if that has the same effect. It does.

This means that the pre and post scalers can both be used to set the period. The prescaler has a lot larger range with less accuracy, while the post scaler has a lower range with higher accuracy

So if I want 1s period,
The base period is 65536us

So I need 1/65536e-6=15.2588 as my total scaling.

I could use postscale 16 to get close. That got me 1.049s

Maybe I could use an 8bit timer to get a more accurate time.

If it is 8bit, than the base period would only be 255
1/255e-6=3,921.5686

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
4	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120	128
8	64	128	192	256	320	384	448	512	576	640	704	768	832	896	960	1024
16	1024	2048	3072	4096	5120	6144	7168	8192	9216	10240	11264	12288	13312	14336	15360	16384
32	32768	65536	98304	131072	163840	196608	229376	262144	294912	327680	360448	393216	425984	458752	491520	524288
64	2097152	4194304	6291456	8388608	10485760	12582912	14680064	16777216	18874368	20971520	23068672	25165824	27262976	29360128	31457280	33554432
128	2.68E+08	5.37E+08	8.05E+08	1.07E+09	1.34E+09	1.61E+09	1.88E+09	2.15E+09	2.42E+09	2.68E+09	2.95E+09	3.22E+09	3.49E+09	3.76E+09	4.03E+09	4.29E+09
256	6.87E+10	1.37E+11	2.06E+11	2.75E+11	3.44E+11	4.12E+11	4.81E+11	5.5E+11	6.18E+11	6.87E+11	7.56E+11	8.25E+11	8.93E+11	9.62E+11	1.03E+12	1.1E+12
512	3.52E+13	7.04E+13	1.06E+14	1.41E+14	1.76E+14	2.11E+14	2.46E+14	2.81E+14	3.17E+14	3.52E+14	3.87E+14	4.22E+14	4.57E+14	4.93E+14	5.28E+14	5.63E+14
1024	3.6E+16	7.21E+16	1.08E+17	1.44E+17	1.8E+17	2.16E+17	2.52E+17	2.88E+17	3.24E+17	3.6E+17	3.96E+17	4.32E+17	4.68E+17	5.04E+17	5.4E+17	5.76E+17
2048	7.38E+19	1.48E+20	2.21E+20	2.95E+20	3.69E+20	4.43E+20	5.17E+20	5.9E+20	6.64E+20	7.38E+20	8.12E+20	8.85E+20	9.59E+20	1.03E+21	1.11E+21	1.18E+21
4096	3.02E+23	6.04E+23	9.07E+23	1.21E+24	1.51E+24	1.81E+24	2.12E+24	2.42E+24	2.72E+24	3.02E+24	3.32E+24	3.63E+24	3.93E+24	4.23E+24	4.53E+24	4.84E+24
8192	2.48E+27	4.95E+27	7.43E+27	9.9E+27	1.24E+28	1.49E+28	1.73E+28	1.98E+28	2.23E+28	2.48E+28	2.72E+28	2.97E+28	3.22E+28	3.47E+28	3.71E+28	3.96E+28
16384	4.06E+31	8.11E+31	1.22E+32	1.62E+32	2.03E+32	2.43E+32	2.84E+32	3.25E+32	3.65E+32	4.06E+32	4.46E+32	4.87E+32	5.27E+32	5.68E+32	6.08E+32	6.49E+32
32768	1.33E+36	2.66E+36	3.99E+36	5.32E+36	6.65E+36	7.98E+36	9.3E+36	1.06E+37	1.2E+37	1.33E+37	1.46E+37	1.6E+37	1.73E+37	1.86E+37	1.99E+37	2.13E+37

I used the prescale of 16 and the post scale of 4 to get close to the 3921, but it is giving me a period of 16.3ms. I am wondering if setting it to 8bit does not change it like I assumed.

65536e-6*4096=268.4355

256e-6*4096=1.0486

This doesn't even get me any better resolution...

This is also not the correct time.

Is there a condition that is not a multiplier? Having the only multipliers of even numbers does not give a whole lot of resolution.
Something like the PWM where there is a counter would give wayyyy better resolution. Surely there is a way to get better than this cause this sucks.
50ms off for a second is a ton.

Im not seeing anything in the data sheet.

Although there is a different clock sources, maybe one of them would work better for a base

I2C

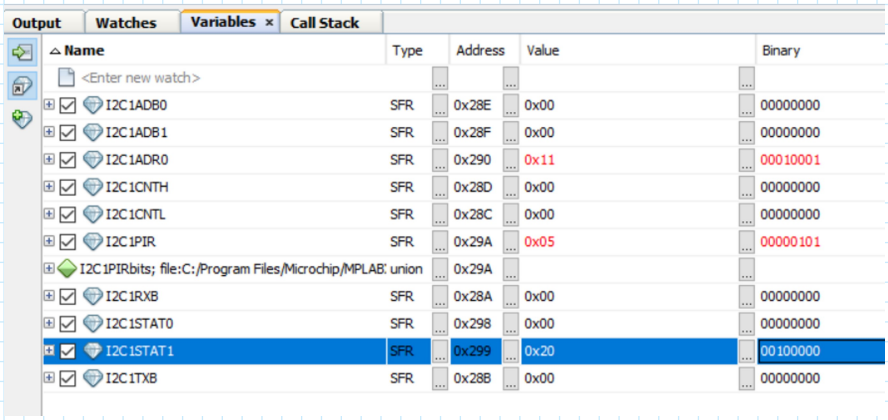
```
void __interrupt(irq(I2C1)) ISR(void){
    if(I2C1PIRbits.SCIF == 1){
        i2cStart();
    }
    return;
}
```

The ISR seems to be triggering properly today.

I still have the issue of the Stop bit condition being set.

I tried changing the Stop flag enable bit to 0 but the flag is still set. I believe the enable only controls whether it triggers the I2C1 interrupt

There is still nothing getting loaded into the address buffers



Name	Type	Address	Value	Binary
<Enter new watch>				
I2C1ADB0	SFR	0x28E	0x00	00000000
I2C1ADB1	SFR	0x28F	0x00	00000000
I2C1ADR0	SFR	0x290	0x11	00010001
I2C1CNTH	SFR	0x28D	0x00	00000000
I2C1CNTL	SFR	0x28C	0x00	00000000
I2C1PIR	SFR	0x29A	0x05	00000101
I2C1PIRbits; file:C:/Program Files/Microchip/MPLAB	union	0x29A		
I2C1RXB	SFR	0x28A	0x00	00000000
I2C1STAT0	SFR	0x298	0x00	00000000
I2C1STAT1	SFR	0x299	0x20	00100000
I2C1TXB	SFR	0x28B	0x00	00000000

I found that there was a bit in the I2CSTAT1 register that was set.

36.5.5 I2CxSTAT1

Name: I2CxSTAT1
Address: 0x0299

I2C Status Register 1

Bit	7	6	5	4	3	2	1	0
	TXWE		TXBE		RXRE	CLRBF		RXBF
Access	R/W/HS		R		R/W/HS	R/S		R
Reset	0		1		0	0		0

Bit 7 – TXWE Transmit Write Error Status(1)

Value	Description
1	A new byte of data was written into I2CxTXB when it was full (*must be cleared by software*)
0	No transmit write error occurred

Bit 5 – TXBE Transmit Buffer Empty Status(2)

Value	Description
1	I2CxTXB is empty (*cleared by writing to the I2CxTXB register*)
0	I2CxTXB is full

Bit 3 – RXRE Receive Read Error Status(1)

Value	Description
1	A byte of data was read from I2CxRXB when it was empty (*must be cleared by software*)
0	No receive overflow occurred

Bit 2 – CLRBF Clear Buffer(3)

Value	Description
1	Setting this bit clears/empties the receive and transmit buffers, causing a Reset of RXBF and TXBE Setting this bit clears the I2CxRXIF and I2CxTXIF interrupt flags

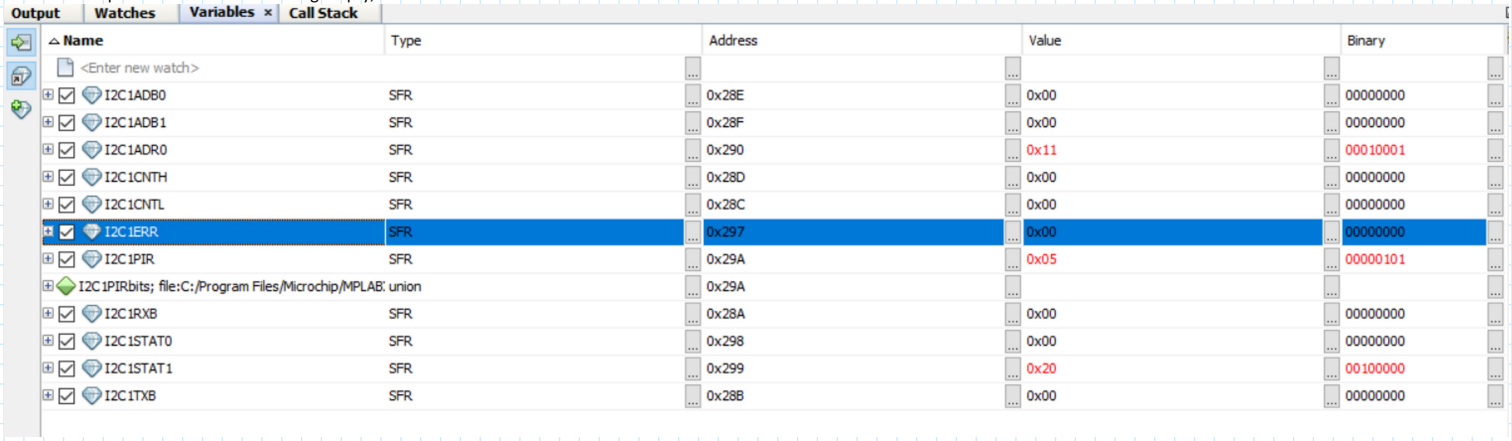
Bit 0 – RXBF Receive Buffer Full Status(2)

Value	Description
1	I2CxRXB is full (*cleared by reading the I2CxRXB register*)
0	I2CxRXB is empty

Notes:

1. This bit, when set, will cause a NACK to be issued.
2. Used as a trigger source for DMA operations.
3. This bit is special function; it can only be set by user software and always reads '0'.

This bit corresponds to the TX buffer being empty, which it should be. No issues there.



Name	Type	Address	Value	Binary
<Enter new watch>				
I2C1ADB0	SFR	0x28E	0x00	00000000
I2C1ADB1	SFR	0x28F	0x00	00000000
I2C1ADR0	SFR	0x290	0x11	00010001
I2C1CNTH	SFR	0x28D	0x00	00000000
I2C1CNTL	SFR	0x28C	0x00	00000000
I2C1ERR	SFR	0x297	0x00	00000000
I2C1PIR	SFR	0x29A	0x05	00000101
I2C1PIRbits; file:C:/Program Files/Microchip/MPLAB	union	0x29A		
I2C1RXB	SFR	0x28A	0x00	00000000
I2C1STAT0	SFR	0x298	0x00	00000000
I2C1STAT1	SFR	0x299	0x20	00100000
I2C1TXB	SFR	0x28B	0x00	00000000

There is no bits being set in the error register.

```
76          * Address is compared to the address saved on this client
77          * If there is a match, the SMA bit is set by hardware to activate c
78          * Data bit D is cleared by hardware to indicate last byte was addre
79          *
80          */
            while(I2C1STAT1bits.RXBF == 0);
            NOP();
```

I added this in the function that triggers when the ISR triggers. The software got stuck in this loop, meaning that nothing was loaded into the RX buffer. I was concerned that maybe I was haulting the program too fast to load the registers.

The closest thing to a clue I have as to why this isnt working is that the STOP flag is always set. But everything I have done to try to clear it has not worked and from what I have read in Client mode, the only way to set that bit is for there to be a stop condition on the line. Which there is not.

I am technically defining the SDA and SCL pins before setting the mode. So if it is defaulting into another mode than client, and I am defining the pins in a certain order, it may happen that the stop condition appears to occur but doesn't. I am not enabling the I2C until after this tho, so it is unlikely,. I moved the pin definition below the I2C setup and nothing changed.

I noticed that I never explicitly disabled the I2C before setting it up. I would assume the default is disabled but. I tried explicitly disabling it first and it did not work.

Leave 3:30

05/07/23

Time: 7.5h

Wednesday, July 5, 2023 8:05 AM

Start 8:00AM

36.3.13.6 7/10-Bit Client Reception

When address buffers are enabled (ABD = 0), client hardware loads I2CxADB0/1 with the matching address, while all data is received by I2CxRXB. Once the client loads I2CxRXB with a received data byte, hardware sets I2CxRXIF, which triggers the DMA to read I2CxRXB. The DMA will continue to read I2CxRXB whenever I2CxRXIF is set.

When address buffers are disabled (ABD = 1), the client loads I2CxRXB with the matching address byte(s) as they are received. Each received address byte sets I2CxRXIF, which triggers the DMA to read I2CxRXB. The DMA will continue to read I2CxRXB whenever I2CxRXIF is set.

Maybe the DMA has something to do with the buffers being empty
Not sure where it would be moved too...

During operation, the client device waits until module hardware detects a Start condition on the bus. Once the Start condition is detected, the client waits for the incoming address information to be received by the receive shift register. The address is then compared to the addresses stored in the I2C Address 0/1/2/3 registers (I2CxADR0, I2CxADR1, I2CxADR2, I2CxADR3), and if an address match is detected, client hardware transfers the matching address into either the I2CxADB0/I2CxADB1 registers or the I2CxRXB register, depending on the state of the Address Buffer Disable (ABD) bit. If there are no address matches, there is no response from the client.

This is saying the address is compared, then moved to RXB
Not received by RXB then compared like I assumed...

In 7-bit Address mode, the received address byte is compared to all four I2CxADR registers independently to determine a match. The R/$\overline{W}$ bit is ignored during address comparison. If a match occurs, the matching received address is transferred from the receive shift register to either the I2CxADB0 register (when ABD = 0) or to the I2CxRXB register (when ABD = 1), and the value of the R/$\overline{W}$ bit is loaded into the Read Information (R) bit.

The recieve register is NOT the same as RXB as I had assumed

I am not seeing anywhere that I can access this "receive register" directly, but from what I am reading the behavior I am seeing is correct IF the address received is not matching the address of the client.

The address is technically 7bit and I am using a byte to store it on each device
I would assume the 7 Lsb would be used but maybe there is an issue there...

I am going to try an address of 0b11111111 on each device

Also, I should be able to use the i2cdetect function on the PI to avoid this issue -- as that function calls every device address and logs which ones respond to determine what devices are available on the bus. This way, there will certainly be a matching address received.

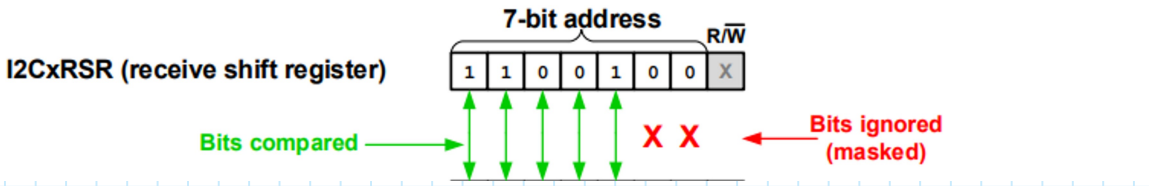
Although the host generates a restart signal instead of a stop between each transmission, meaning that I need something to trigger when the matching address is received.. Is there a bit on the PIR register or another register that will tell me when a matching address is received?

Bit 3 – ADRIF Address Interrupt Flag (*used only when* MODE = 0xx *or MODE* = 11x)(1)

Value	Description
1	Address detected, set on the 8th falling SCL edge for a matching received address byte
0	Address not detected

```
    */
    if(I2C1PIRbits.ADRIF == 1){
        NOP();
    }
```

I added this in the function that gets called when the start condition is detected but it never triggers. So there is never a matching address received. Even through the bus is having every address checked…



I2C1RSR looks like the receive register. Maybe I can monitor that register.

I cant monitor it in the debugger like the other registers. I also can't read the value using I2C1RSR like I can with other registers. The above image is also the only instance of I2CxRSR in the whole datasheet. I2C1RSR is never mentioned.

36.5.16 I2CxADR0

Name: I2CxADR0
Address: 0x0290

I2C Address 0 Register

Bit	7	6	5	4	3	2	1	0
	ADR[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1

Bits 7:0 – ADR[7:0] I2C Client Address 0

Condition	Description
7-bit Client/Multi-Host modes (MODE = *00x* or *11x*):	**ADR[7:1]:** 7-bit client address **ADR[0]:** Unused; bit state is 'don't care'
10-bit Client modes (MODE = *01x*):	**ADR[7:0]:** Eight Least Significant bits of first 10-bit address

It looks like I do need to shift the address over, meaning the address it checks is actually the 7Msb not 7Lsb
So the address should be 0b11111110 instead of 0b1111111

I changed the address and still am not getting an a matching address…

But one more error off the list.

I tried using the MCC tool to make a script that would use I2C and just let me see what the registers are doing. But this also does not work properly. I have no Idea what the problem is, but I have no more ideas today.

Timer Interrupts

I went back to the datasheet and found that there is a match option on some of the timers, including timer 0, but only in 8 bit mode. This might give me better resolution for the timer interrupt.

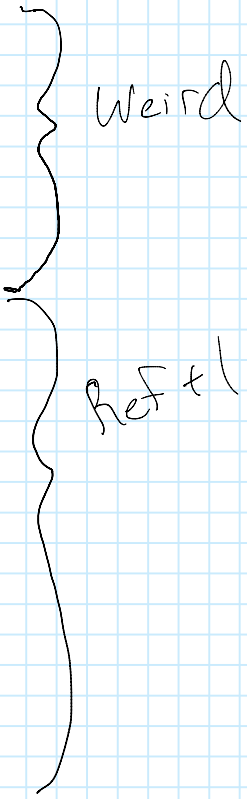
I changed it to 8bit, set the reference (high register) to 1, and removed all prescaling. This gives a period of 8us.

1/1e6=1e-6 = 1us. Not sure why it is 8…

If I set the reference to 2, the period changes to 9

If I set the reference to 3, the period changes to 8 again…

Ref	Time (us)
0x00	0 (no interrupt)
0x01	8
0x02	9
0x03	8
0x04	10
0x05	12
0x06	7
0x07	8
0x08	9
0x09	10
0x0A	11
0x0B	12
0x0C	13
0x0D	14
0x0E	15
0x0F	16
0x10	17



There seems to be a pattern once you get past the first few bits, not sure why it isnt the same for those

How do I go from a desired period to a setup for this timer?

Period = base_period * prescaler * postscaler * cnt

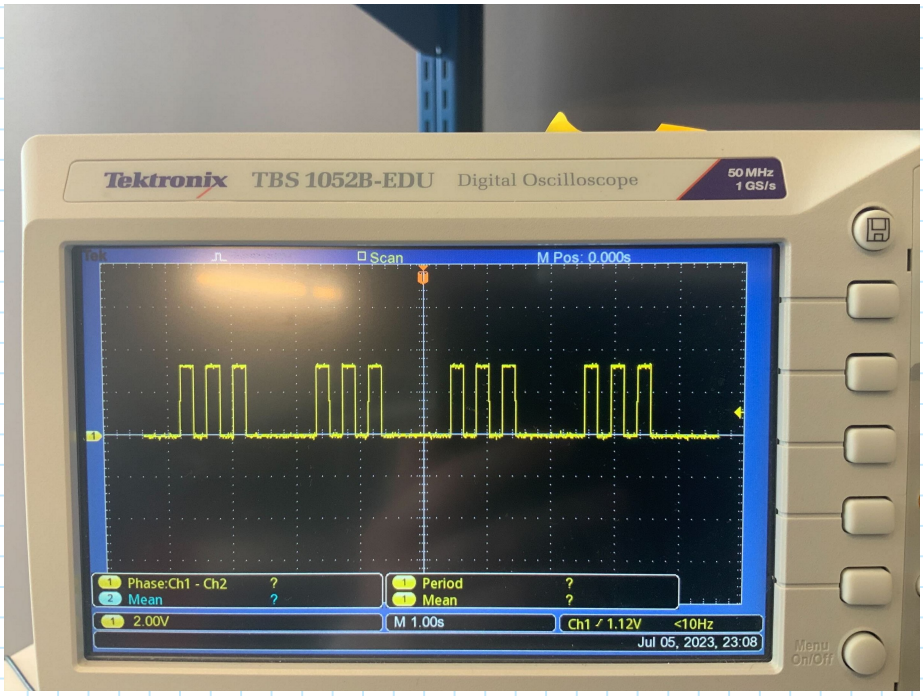
Base_period = 1us
Prescaler = 1, 2, 4, 8, 32, …, 32786
Postscaler = 1:16
Cnt = 7-256

If period = 1s

1.00000000/32786=3.05E-5 = 30.5us

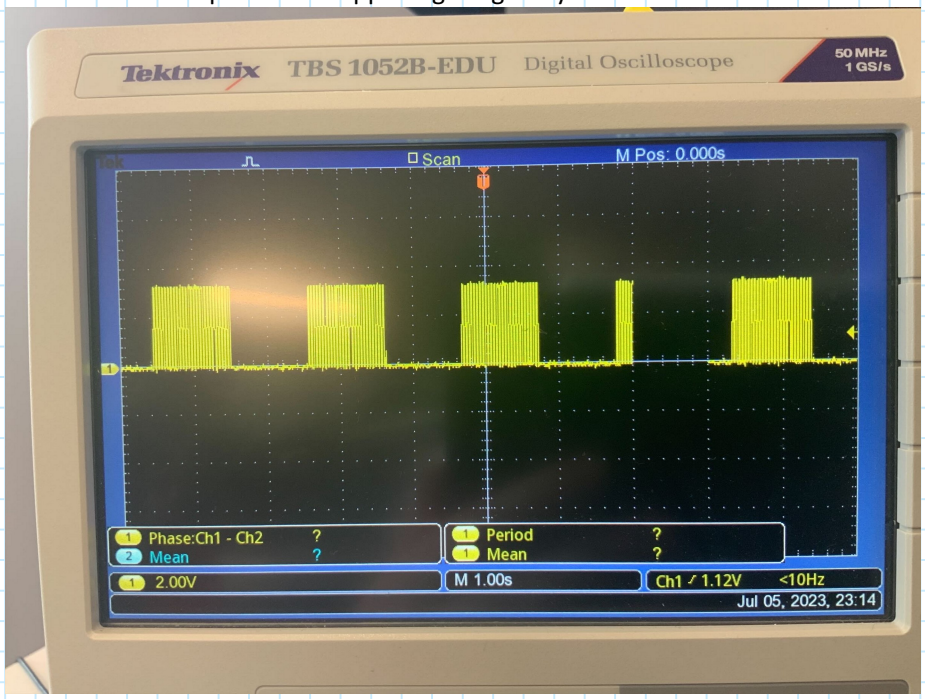
So setting CNT to 30 should make it 31us

31e-6*32786=1.0164



I am getting this weird behaviour where the signal is ocsillating. I am using a flag to cycle the pin every interrupt, so I should be getting a 50% duty cycle all the time, regardless of the interrupt time.

Unless the interrupt itself is happening irregularly…



Changing the reference number in the high register seems to change the amount of oscillations in the signal, but does not change the overall period.

Irregularity appears to be induced when using the prescaler. It is completely regular when only the post scaler is used. However I cant get a very large time delay with only the post scaler. The largest is 8.2ms

I did some more experimenting, it seems that the prescaler is stable but only when the value is kept below 8 bits.

There is no condition in the datasheet that states that the prescaler will act differently depending on whether the timer is in 8bit or 16bit mode. However I know that for Timer 2 (which is only 8 bit) the prescaler only goes from 1-128

I am wrong, I was writing in binary so the first to digits are bits not 4 bits (HEX). So it is stable until the 3rd bit is used.

It seems like all the three variables effect how the output is looking…

Even changing back to the 16bit mode, the system is not stable like I would assume, some cycles are longer than others etc.

IMU

https://github.com/dtornqvist/icm-20948-arduino-library/blob/master/src/ICM20948.h

https://invensense.tdk.com/wp-content/uploads/2016/06/DS-000189-ICM-20948-v1.3.pdf

There are some libraries available for the IMU, however they are written in Arduino, meaning that they are going to use the I2C library for Arduino in all of their functions. Depending on how dependent the library is on the Arduino library, I may be able to just change the references to the i2c library to my i2c library any use the rest.

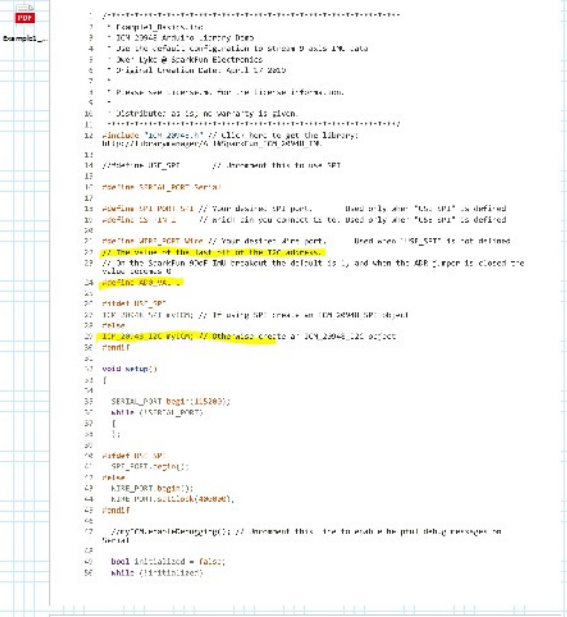
Leave - 3:30

06/07/23

Thursday, July 6, 2023 8:01 AM

Start Coding

~\Downloads\Example1_Basics.ino



This library appears to just be build as a wrapper for another library. This is getting really in the weeds into details and files referencing other files and stuff.

I doubt that I really need this level of control. Like I could probably get away with setting up the required registers statically instead of having a function that can change every single option.

The library being referenced here is 2.7k lines of code. That might be a bit much to go through...

```
  printPaddedInt16b(agmt.acc.axes.y);
  SERIAL_PORT.print(", ");
  printPaddedInt16b(agmt.acc.axes.z);
  SERIAL_PORT.print(" ], Gyr [ ");
  printPaddedInt16b(agmt.gyr.axes.x);
  SERIAL_PORT.print(", ");
  printPaddedInt16b(agmt.gyr.axes.y);
  SERIAL_PORT.print(", ");
  printPaddedInt16b(agmt.gyr.axes.z);
  SERIAL_PORT.print(" ], Mag [ ");
  printPaddedInt16b(agmt.mag.axes.x);
  SERIAL_PORT.print(", ");
  printPaddedInt16b(agmt.mag.axes.y);
  SERIAL_PORT.print(", ");
  printPaddedInt16b(agmt.mag.axes.z);
  SERIAL_PORT.print(" ], Tmp [ ");
  printPaddedInt16b(agmt.tmp.val);
  SERIAL_PORT.print(" ]");
  SERIAL_PORT.println();
}
```

```

void printFormattedFloat(float val, uint8_t leading, uint8_t decimals)
{
  float aval = abs(val);
  if (val < 0)
  {
    SERIAL_PORT.print("-");
  }
  else
  {
    SERIAL_PORT.print(" ");
  }
  for (uint8_t indi = 0; indi < leading; indi++)
  {
    uint32_t tenpow = 0;
    if (indi < (leading - 1))
    {
      tenpow = 1;
    }
    for (uint8_t c = 0; c < (leading - 1 - indi); c++)
    {
      tenpow *= 10;
    }
    if (aval < tenpow)
    {
      SERIAL_PORT.print("0");
    }
    else
    {
      break;
    }
  }
  if (val < 0)
  {
    SERIAL_PORT.print(-val, decimals);
  }
  else
  {
    SERIAL_PORT.print(val, decimals);
  }
}

#ifdef USE_SPI
void printScaledAGMT(ICM_20948_SPI *sensor)
{
#else
void printScaledAGMT(ICM_20948_I2C *sensor)
{
#endif
  SERIAL_PORT.print("Scaled. Acc (mg) [ ");
  printFormattedFloat(sensor->accX(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(sensor->accY(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(sensor->accZ(), 5, 2);
  SERIAL_PORT.print(" ], Gyr (DPS) [ ");
```

```
  printFormattedFloat(sensor->gyrX(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(sensor->gyrY(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(sensor->gyrZ(), 5, 2);
  SERIAL_PORT.print(" ], Mag (uT) [ ");
  printFormattedFloat(sensor->magX(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(sensor->magY(), 5, 2);
  SERIAL_PORT.print(", ");
  printFormattedFloat(sensor->magZ(), 5, 2);
  SERIAL_PORT.print(" ], Tmp (C) [ ");
  printFormattedFloat(sensor->temp(), 5, 2);
  SERIAL_PORT.print(" ]");
  SERIAL_PORT.println();
}

```

IMU Datasheet

Looking at the datasheet, there are clear registers that hold the readings of the sensors. There are labelled like ACCEL_YOUT_L and ACCEL_YOUT_H.

And the datasheet outlines how registers can be read using I2C.

But I am confused as to when these registers are readable. Cause surely they have to be updated at some point. So I cant read it while it is updating. Maybe these registers are latched and wait for a read before loading the next value in. Or there is a separate way to tell the registers to update so that I can read it once that is done. I see that there is a "Data ready" bit, but this seems to be associated with the FIFO, which that whole module is much less clear to me.

```
  if (pdev->_serif->write == NULL)
  {
    return ICM_20948_Stat_NotImpl;
  }
  return (*pdev->_serif->write)(regaddr, pdata, len, pdev->_serif->user);
}
```

I know in the data sheet there is a description of how to interface with the module using I2C. I am thinking it may be easier to start from scratch than to try to modify this library.

Burst Write Sequence

Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

To read the internal ICM-20948 registers, the master sends a start condition, followed by the I²C address and a write bit, and then the register address that is going to be read. Upon receiving the ACK signal from the ICM-20948, the master transmits a start signal followed by the slave address and read bit. As a result, the ICM-20948 sends an ACK signal and the data. The communication ends with a not acknowledge (NACK) signal and a stop bit from master. The NACK condition is defined such that the SDA line remains high at the 9th clock cycle. The following figures show single and two-byte read sequences.

Single-Byte Read Sequence

Master	S	AD+W		RA		S	AD+R			NACK	P	
Slave			ACK		ACK			ACK	DATA			

Burst Read Sequence

Master	S	AD+W		RA		S	AD+R			ACK		NACK	P	
Slave			ACK		ACK			ACK	DATA		DATA			

6.4 I²C TERMS

SIGNAL	DESCRIPTION
S	Start Condition: SDA goes from high to low while SCL is high
AD	Slave I²C address
W	Write bit (0)
R	Read bit (1)
ACK	Acknowledge: SDA line is low while the SCL line is high at the 9th clock cycle
NACK	Not-Acknowledge: SDA line stays high at the 9th clock cycle
RA	ICM-20948 internal register address
DATA	Transmit or received data
P	Stop condition: SDA going from low to high while SCL is high

Table 15. I²C Terms

I thought this was pretty clear. Until I found that the different register banks can have the same internal addresses. For example, 0x05 is in Bank 0 and Bank 2

7.1 USER BANK 0 REGISTER MAP

ADDR (HEX)	ADDR (DEC.)	REGISTER NAME	SERIAL I/F	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
00	0	WHO_AM_I	R	WHO_AM_I[7:0]							
03	3	USER_CTRL	R/W	DMP_EN	FIFO_EN	I2C_MST_EN	I2C_IF_DIS	DMP_RST	SRAM_RST	I2C_MST_RST	-
05	5	LP_CONFIG	R/W		I2C_MST_CYCLE	ACCEL_CYCLE	GYRO_CYCLE	-			
06	6	PWR_MGMT_1	R/W	DEVICE_RESET	SLEEP	LP_EN	-	TEMP_DIS	CLKSEL[2:0]		
07	7	PWR_MGMT_2	R/W	-		DISABLE_ACCEL			DISABLE_GYRO		
0F	15	INT_PIN_CFG	R/W	INT1_ACTL	INT1_OPEN	INT1_LATCH_INT_EN	INT_ANYRD_2CLEAR	ACTL_FSYNC	FSYNC_INT_MODE_EN	BYPASS_EN	-

7.3 USER BANK 2 REGISTER MAP

ADDR (HEX)	ADDR (DEC.)	REGISTER NAME	SERIAL I/F	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
00	0	GYRO_SMPLRT_DIV	R/W	GYRO_SMPLRT_DIV[7:0]							
01	1	GYRO_CONFIG_1	R/W	-		GYRO_DLPFCFG[2:0]			GYRO_FS_SEL[1:0]		GYRO_FCHOICE
02	2	GYRO_CONFIG_2	R/W	-		XGYRO_CTEN	YGYRO_CTEN	ZGYRO_CTEN	GYRO_AVGCFG[2:0]		
03	3	XG_OFFS_USRH	R/W	X_OFFS_USR[15:8]							
04	4	XG_OFFS_USRL	R/W	X_OFFS_USR[7:0]							
05	5	YG_OFFS_USRH	R/W	Y_OFFS_USR[15:8]							
06	6	YG_OFFS_USRL	R/W	Y_OFFS_USR[7:0]							
07	7	ZG_OFFS_USRH	R/W	Z_OFFS_USR[15:8]							
08	8	ZG_OFFS_USRL	R/W	Z_OFFS_USR[7:0]							
09	9	ODR_ALIGN_EN	R/W								ODR_ALIGN_EN
10	16	ACCEL_SMPLRT_DIV_1	R/W	-				ACCEL_SMPLRT_DIV[11:8]			
11	17	ACCEL_SMPLRT_DIV_2	R/W	ACCEL_SMPLRT_DIV[7:0]							
12	18	ACCEL_INTEL_CTRL	R/W	-						ACCEL_INTEL_EN	ACCEL_INTEL_MODE_INT
13	19	ACCEL_WOM_THR	R/W	WOM_THRESHOLD[7:0]							
14	20	ACCEL_CONFIG	R/W	-		ACCEL_DLPFCFG[2:0]			ACCEL_FS_SEL[1:0]		ACCEL_FCHOICE
15	21	ACCEL_CONFIG_2	R/W	-			AX_ST_EN_REG	AY_ST_EN_REG	AZ_ST_EN_REG	DEC3_CFG[1:0]	
52	82	FSYNC_CONFIG	R/W	DELAY_TIME_EN		WOF_DEGLITCH_EN	WOF_EDGE_INT	EXT_SYNC_SET[3:0]			
53	83	TEMP_CONFIG	R/W	-					TEMP_DLPFCFG[2:0]		
54	84	MOD_CTRL_USR	R/W	-							REG_LP_DMP_EN
7F	127	REG_BANK_SEL	R/W	-		USER_BANK[1:0]					

So sending address of 0x05 does not really tell the device what register I want. But I think that 7F in every register corresponds to the REG_BANK_SEL register which lets you select a bank. So I am assuming that I just need to make a write command to that register to switch to the right bank, and then use the I2C outlined above to read/write a specific register.

The other thing that I am concerned about is how to do another start condition without a stop condition on the RPI's I2C.

```
ioctl(file, I2C_FUNCS, unsigned long *funcs)
  Gets the adapter functionality and puts it in *funcs.

ioctl(file, I2C_RDWR, struct i2c_rdwr_ioctl_data *msgset)
  Do combined read/write transaction without stop in between.
  Only valid if the adapter has I2C_FUNC_I2C.  The argument is
  a pointer to a

  struct i2c_rdwr_ioctl_data {
      struct i2c_msg *msgs;  /* ptr to array of simple messages */
      int nmsgs;             /* number of messages to exchange */
  }

  The msgs[] themselves contain further pointers into data buffers.
  The function will write or read data to or from that buffers depending
  on whether the I2C_M_RD flag is set in a particular message or not.
  The slave address and whether to use ten bit address mode has to be
  set in each message, overriding the values set with the above ioctl's.
```

I am still unclear as how this works. Cause before I was using the ioctl to define what device I was going to interact with, and then used the read() and write() functions after. But this ioctl command seems to need to have the message when I all it. So it doesn't seem like I use the read() or write() independently. Also what is the I2C_M_RD flag and where do I set it? Is that contained in the message?

```
// Gets one or more variables from the Tic (without clearing them).
// Returns 0 for success, -1 for failure.
int tic_get_variable(int fd, uint8_t address, uint8_t offset,
  uint8_t * buffer, uint8_t length)
{
  uint8_t command[] = { 0xA1, offset };
  struct i2c_msg messages[] = {
    { address, 0, sizeof(command), command },
    { address, I2C_M_RD, length, buffer },
  };
  struct i2c_rdwr_ioctl_data ioctl_data = { messages, 2 };
  int result = ioctl(fd, I2C_RDWR, &ioctl_data);
  if (result != 2)
  {
    perror("failed to get variables");
    return -1;
  }
  return 0;
}
```

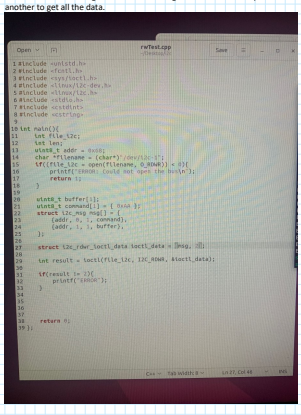
https://www.pololu.com/docs/0J71/12.8

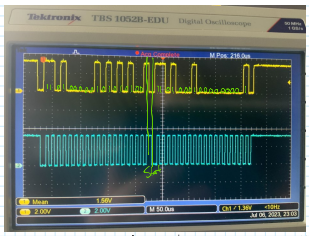
This makes thing a bit more clear. There is a i2c_msg structure that holds the format of the message.
So for this, I should need:

Struct i2c_msg messages[] = {
  {address, 0, 1, REGADDR},
  {address, I2C_M_RD, 1, buffer},

};

And this would do a single write and then single read without a stop condition inbetween. I would likely do these one after another to get all the data.





This appears to be working — It wrote the

Although with this method, all 9 sensor readings is going to require me to send the address again, which is not very time efficient. It would be better if I would get all the reading to come in the same read command. That was I only have to send the address twice as opposed to 10 times.

I can get the Accel and Gyro to go to the FIFO register which supports burst reading. However I do not see a way to get the Mag to go there. And ultimately I do not think the accelerometer actually needs to be used, so trying to setup the FIFO would only help reduce the Gyro form 4 addresses to 2.

I think I am going to make it resent the address every time for now, at least to get something going. It can be optimized later if it is an issue.

```
73  int i2c::writeReadBus(uint8_t address, int length, uint8_t *command, bool *R_W){
74      sturct i2c_msg msg[] = {
75      };
76
77      for(int i = 0; i < length; i++){
78          msg[i] = {address, R_W[i], 1, command[i]};
79      }
80
81      struct i2c_rdwr_ioctl_data ioctl_data = {msg, length};
82
83      if(ioctl(file_i2c, I2C_RDWR, &ioctl_data) != length){
84          printf("ERROR:\tFailed to complete read/write command");
85          return 1;
86      }
87      return 0;
88  }
```

I added this funciton to my I2C class. The idea is that you give it the address of the slave, and then an array containing the commands you want to write/ a buffer for the read commands.

I am not sure if I can append the msg structure as I did... but there is only one way to find out.

I am also not sure if I can use the length parameter in the declaration of the other parameters.

I guess I will have to make a test script to see if it works and send it to the PI to test.

The for look does not work. I am getting errors when I try to compile.

To-Do list

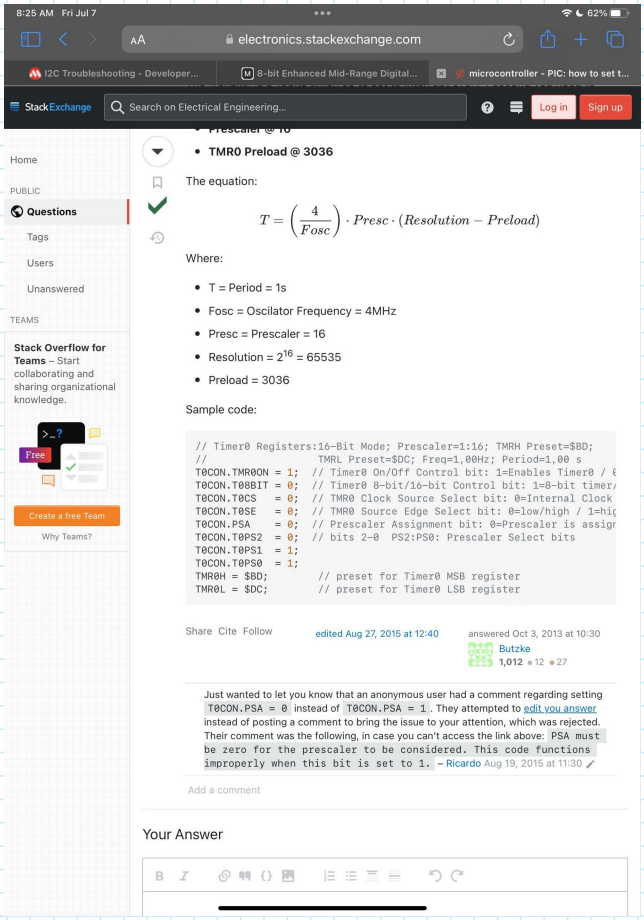
- Get I2C working on the PIC controller
- Get the I2C for the IMU working on the RPI
- Get more reliable and accurate control of the timers on the PIC
- Deal with the Zero 2 W issue

07/07/23

Friday, July 7, 2023 8:19 AM

Start 8:15 AM

https://electronics.stackexchange.com/questions/84164/pic-how-to-set-timer0-to-give-an-interrupt-at-every-1-sec



- **TMR0 Preload @ 3036**

The equation:

$$T = \left(\frac{4}{Fosc}\right) \cdot Presc \cdot (Resolution - Preload)$$

Where:

- T = Period = 1s
- Fosc = Oscilator Frequency = 4MHz
- Presc = Prescaler = 16
- Resolution = 2^{16} = 65535
- Preload = 3036

Sample code:

```
// Timer0 Registers:16-Bit Mode; Prescaler=1:16; TMRH Preset=$BD;
//                  TMRL Preset=$DC; Freq=1,00Hz; Period=1,00 s
T0CON.TMR0ON = 1;  // Timer0 On/Off Control bit: 1=Enables Timer0 / 0
T0CON.T08BIT = 0;  // Timer0 8-bit/16-bit Control bit: 1=8-bit timer/
T0CON.T0CS   = 0;  // TMR0 Clock Source Select bit: 0=Internal Clock
T0CON.T0SE   = 0;  // TMR0 Source Edge Select bit: 0=low/high / 1=hig
T0CON.PSA    = 0;  // Prescaler Assignment bit: 0=Prescaler is assign
T0CON.T0PS2  = 0;  // bits 2-0  PS2:PS0: Prescaler Select bits
T0CON.T0PS1  = 1;
T0CON.T0PS0  = 1;
TMR0H = $BD;       // preset for Timer0 MSB register
TMR0L = $DC;       // preset for Timer0 LSB register
```

Share Cite Follow edited Aug 27, 2015 at 12:40 answered Oct 3, 2013 at 10:30 Butzke 1,012 • 12 • 27

Just wanted to let you know that an anonymous user had a comment regarding setting `T0CON.PSA = 0` instead of `T0CON.PSA = 1`. They attempted to edit you answer instead of posting a comment to bring the issue to your attention, which was rejected. Their comment was the following, in case you can't access the link above: `PSA must be zero for the prescaler to be considered. This code functions improperly when this bit is set to 1.` – Ricardo Aug 19, 2015 at 11:30

Add a comment

Your Answer

This person is ignoring the post scaler and setting a preload into the timer register

https://www.engineersgarage.com/specific-time-delay-using-timers-of-pic-microcontroller/

This person is employing a similar strategy

So if I want 1s in 16bit mode

$\frac{4}{4MH_2} = 1\mu s$

Prescaler = 32

$TP = 1\mu s \cdot 32 = 32\mu s$

$\frac{1s}{32\mu s} = \frac{1}{32} \times 10^6 > 65536$ ✗

Prescaler = 4096

TP = 4.096ms

$\frac{1}{4.096 \times 10^{-3}} = 244.14$ if I round down to 244

$P = \frac{4}{4Mhz} \cdot 4096 \cdot (65536 - 244)$ $\quad 65536 - 244 = 65292$

$1 \times 10^{-6} \cdot 4.096 \times 10^{3}$

$4.096 \times 10^{-3} \cdot 65.292 \times 10^{3}$

Something here is not right

$4.096 \times 10^{-3} \times 244 = 999.424ms$

so I need 244 counts, meaning I want to preload the timer to 65292

The guy on the website used 65536 − n instead of n but I guess maybe he's counting down on the timer instead of up.

It would probably be choose a smaller prescaler to use more of the timer and get a better resolution

Prescale 128

$\frac{1 \times 10^{-6}s}{128} = 0.128 \times 10^{-3}s$ $\quad \frac{1}{0.128 \times 10^{-3}} = 7.8125 \times 10^{3} = 7812.5$ round down 7812

$0.128ms \cdot 7812 = 999.936ms$ even better

there will be some time required to reset the timer so under is better

I have been trying to put this into a script but it is acting super weird. I have it set to toggle a flag on every interrupt, and then constantly write the flag to a IO pin. But it is not running a 50% duty cycle as I would expect. I tired using another pin to tie to the output of timer0 to see when it is triggering and for how long relative to the flag. But the two pins are tied together -- they act the exact same way. Even when I define of as an output and only set it to 0, no other command, it still going high with the TMR out pin…

```
#include <xc.h>
#include <stdio.h>
#include <stdbool.h>

#define _XTAL_FREQ 4000000

bool flag = 0;
void setup(){
  /*Config Output pin for test*/
  ANSELBbits.ANSELB6 = 0; //
  TRISBbits.TRISB6 = 0;   //Set B6 to output
  LATBbits.LATB6 = 0;     //Set B6 to 0;

  ANSELAbits.ANSELA0 = 0; //
  TRISAbits.TRISA0 = 0;   //Set B6 to output
  RA0PPS = 0x23;

  /*Config Timer0*/
  T0CON0bits.EN = 0;     //Disable timer
```

```
    T0CON0bits.MD16 = 1;    //Set timer to 16bit mode
    T0CON0bits.OUTPS = 0b0000;  //Set post scaler to 16

    T0CON1bits.CS = 0b010;     //Set clock source to Fosc/4
    T0CON1bits.ASYNC = 0;      //Clock is synchronized to Fosc/4
    T0CON1bits.CKPS = 0b0111;   //Prescaler set to 1

    T0CON0bits.EN = 1;         //Enable timer

    /*Config Interrupt*/
    INTCON0bits.GIE = 1;    //Enable interrupts
    INTCON0bits.IPEN = 1;   //Enable Interrupt Priorities

    PIR3bits.TMR0IF = 0;    //Clear interrupt flag
    PIE3bits.TMR0IE = 1;    //Enable timer0 interrupts
  }

  void __interrupt(irq(TMR0)) ISR(void){
    //PIR3bits.TMR0IF = 0;
    //flag = !flag;
    //T0CON0bits.EN = 0;
    //TMR0L = 0x84;
    //TMR0H = 0x1E;
    //T0CON0bits.EN = 1;
  NOP();

  }

  void loop(){
    //LATBbits.LATB6 = flag;
    NOP();
  }

  void main(void){
    setup();
    while(1){
      loop();
    }
    return;
  }
```

This script is causing the A0 and B6 pins to act identical…

But I am never setting the B6 pin to anything in the script…

The pins are in completely different registers and should not be tied together….

I commented out the ISR completely and that caused the A0 line to stay low. Now the B5 line has a lot of noise spikes but is generally low.

I cant figure out why the pins are tied together and its hard to debug because I can only see the interrupt

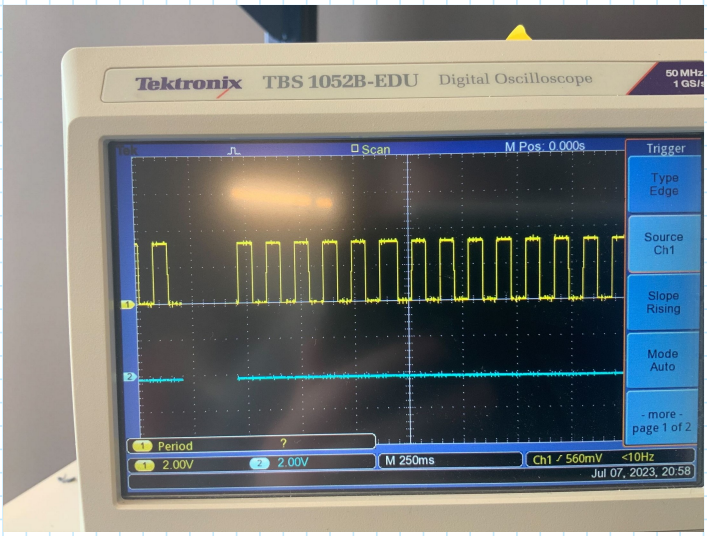
I am seeing that the OUT bit of the timer toggles, meaning that every time the High register rolls over OUT = !OUT

I cant clear OUT as it is read only.

```
#include <xc.h>
#include <stdio.h>

#define _XTAL_FREQ 4000000

void setup(){
  /*Config Output pin for test*/
  TRISBbits.TRISB6 = 0;   //Set B6 to output

  /*Config Timer0*/
  T0CON0bits.EN = 0;     //Disable timer
  T0CON0bits.MD16 = 1;    //Set timer to 16bit mode
  T0CON0bits.OUTPS = 0b0000;  //Set post scaler to 16
```

```
  T0CON1bits.CS = 0b010;     //Set clock source to Fosc/4
  T0CON1bits.ASYNC = 0;      //Clock is synchronized to Fosc/4
  T0CON1bits.CKPS = 0b0000;  //Prescaler set to 1

  T0CON0bits.EN = 1;         //Enable timer

  /*Config Interrupt*/
  INTCON0bits.GIE = 1;   //Enable interrupts
  INTCON0bits.IPEN = 1;  //Enable Interrupt Priorities

  PIR3bits.TMR0IF = 0;   //Clear interrupt flag
  PIE3bits.TMR0IE = 1;   //Enable timer0 interrupts
}

void __interrupt(irq(TMR0)) ISR(void){
  NOP();
}

void loop(){
  NOP();
}

void main(void){
  setup();
  while(1){
    loop();
  }
  return;
}
```

Setting a pin to an an output is automatically tying it to the ISR and it toggles… WTF is this.



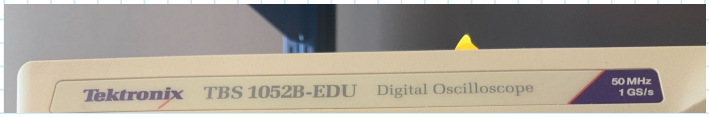
CH1 is on RB6. I also tried setting it to RB5 just to see if RB6 was defaulted to the timer out or what.

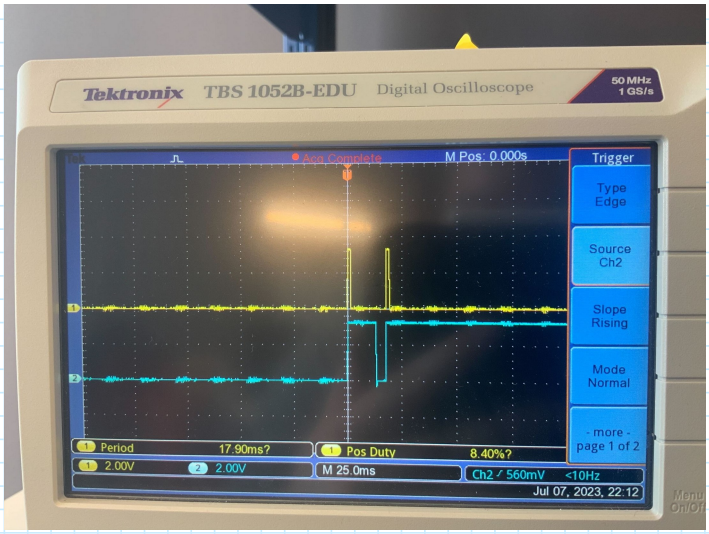
Same issue persists.

I got rid of all the ISR, anything to do with the timer aside from enabling it, and it is causing interference on my output pin.

This is so dumb.

I cycled power to the chip and the interference is gone…





That is my problem. There is a double trigger happening on the rising edge of the OUT bit. Which is causing the ISR to trigger twice rapidly. I need to get rid of this.

Nick removed the Arduino shield with CPLD from the PIC board. This seemed to address the issues with the random trigger of the timer. However this was not tested exhaustively.

We broke something with the PIC to where I can no longer program it…

Looking up the error code it looks like there could be damage to the memory on the PIC that is preventing code from being uploaded.

I would like to see if the timers are working but I cant do that right now. I think all I can really do is look into the