In this two part, two week lab we used Pulse Width Modulation via the Output Compare peripheral on our PIC32 to control the speed of the DC motors. These motors control the two wheels on our robot and so we need two PWMs. We also made use of a timer peripheral to control the how often the pulse output was in the PWM. We set the period to 1000 which resulted in about 70% of the PWM duty cycle for proper operation and variability in speed.

**PART 1**

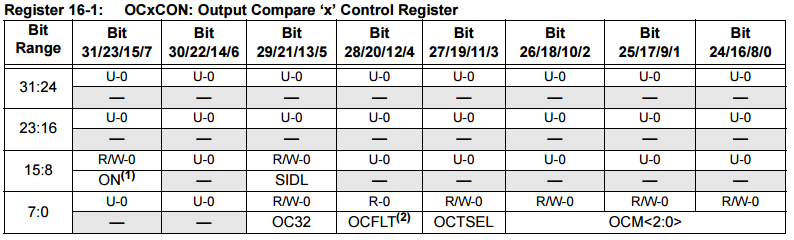
In part 1, we have almost the same code as one can be found below in part 2 report, but without the interrupt service routine and the algorithm for the robot to follow the line (i.e. the if-else statement) was in the main function inside the while(1) loop. So the only thing we added for part 1 is these lines of code:

OpenOC2(OC\_ON | OC\_IDLE\_STOP | OC\_TIMER\_MODE16 | OC\_TIMER2\_SRC | OC\_PWM\_FAULT\_PIN\_DISABLE, 500, 500); // Left Wheel

OpenOC3(OC\_ON | OC\_IDLE\_STOP | OC\_TIMER\_MODE16 | OC\_TIMER2\_SRC | OC\_PWM\_FAULT\_PIN\_DISABLE, 500, 500); // Right Wheel

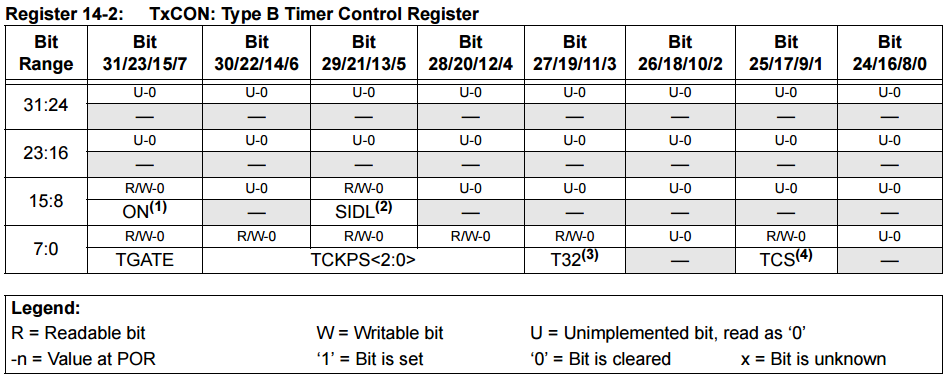
OpenTimer2(T2\_ON | T2\_IDLE\_STOP | T2\_GATE\_OFF | T2\_PS\_1\_1, 0x000003E8);

We referred to the PIC32 Family Reference Manual Section 12 (IO Ports), 14 (Timers), and 16 (Output Compare) when doing the lab. We simply called the output compare functions as seen above. To better understand how we wrote the code, here’s a table of the Output Compare control register:

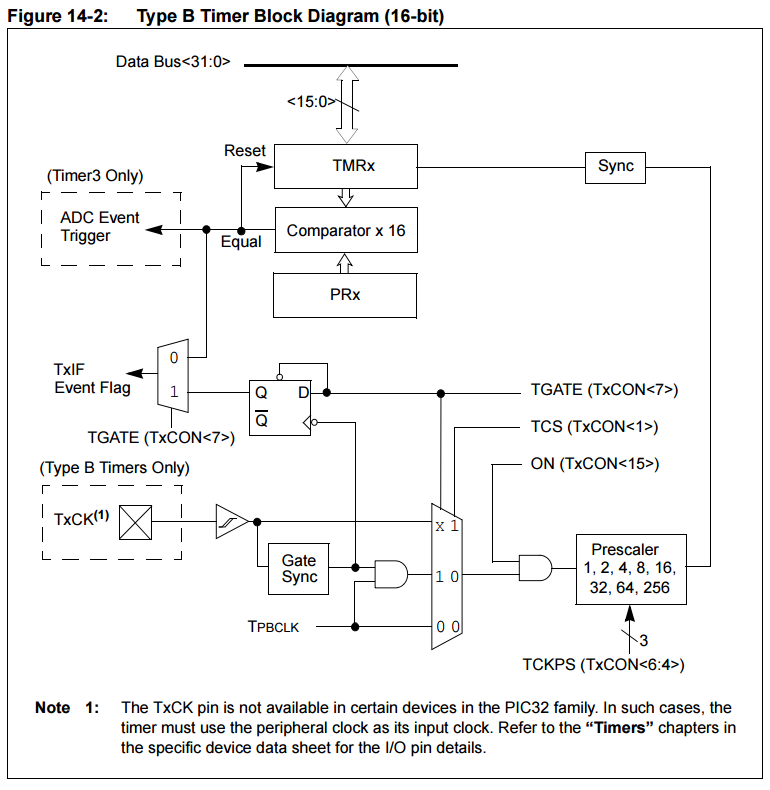


We used the OC2 and OC3 to *replace* pins RD1 and RD2 which enable the wheels to turn. To enable the output compare peripheral, we set the ON bit (bit 15) to ‘1’ simply by writing OC\_ON. Then, we want to discontinue operation when CPU enters Idle mode, so we used SIDL (Stop in Idle Mode) bit or bit 13 by setting it to ‘1’ by writing OC\_IDLE\_STOP. We also used the OCTSEL (Output Compare Timer Select) bit or bit 3 to set the clock source by selecting OC\_TIMER\_MODE16, then setting OCTSEL to ‘0’ to use Timer2, or by simply writing OC\_TIMER2\_SRC. Then we disabled the fault pin by simply writing OC\_PWM\_FAULT\_PIN\_DISABLE.

For the timer, we used Type B timer. The timer control register for Type B is seen below:



We used the timer T2, which is written onto bit 15 (as the Timer On bit; ‘1’ is to enable or if we just call the function OpenTimer2, then we can just write T2\_ON). Then, we also used the SIDL bit (bit 13) to discontinue operation when device enters Idle mode (set to ‘1’, or using OpenTimer2 function, we just write T2\_IDLE\_STOP). Based on the schematic, we should also turn the gate off so that it doesn’t have the ability to stop the timer; we did this by setting the TGATE bit (bit 7) to ‘0’, or if using OpenTimerT2, we simply write T2\_GATE\_OFF. Lastly, we set the prescale value to be 1 to 1 and the period to be 1000 (3E8 in HEX).



**PART 2**

We used an interrupt service routine (ISR) in this part to control movement of the motors. Code that had previously been used in the while loop within the main function to control motors has been modified and put into the ISR. Notes about the ISR are found in the code below.

From part 1, we added these lines to our main function:

/\* setup the change notice options \*/

mCNOpen(CN\_ON | CN\_IDLE\_STOP, CN3\_ENABLE | CN5\_ENABLE ,CN\_PULLUP\_DISABLE\_ALL );

/\* read port(s) to clear mismatch \*/

int value = mPORTBRead();

/\* clear change notice interrupt flag \*/

ConfigIntCN(CHANGE\_INT\_ON | CHANGE\_INT\_PRI\_2);

/\* enable multi-vector interrupts \*/

INTEnableSystemMultiVectoredInt();

/\* Initial Setup \*/

SetDCOC3PWM(450);

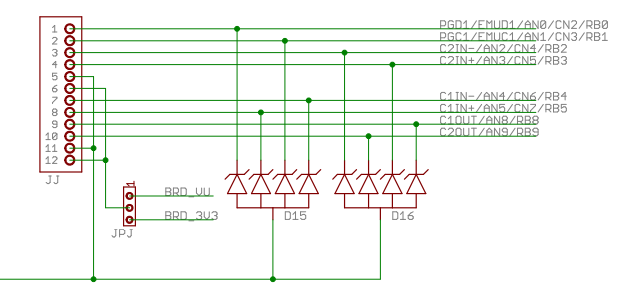
SetDCOC2PWM(450);

PORTSetBits(IOPORT\_D, BIT\_1 | BIT\_2 | BIT\_7); // Both Wheel Enabled, Right Direction Forward

PORTClearBits(IOPORT\_D, BIT\_6); // Left Direction Forward

The first line mCNOpen(CN\_ON | CN\_IDLE\_STOP, CN3\_ENABLE | CN5\_ENABLE ,CN\_PULLUP\_DISABLE\_ALL ); is basically saying we want to observe the change in CN3 and CN5, and to stop operation when device is in Idle mode. CN3 and CN5 refers to the middle-right and left most sensors; so we’re saying that if there’s a change in reading (i.e. from black to light or vice versa) then we go to our interrupt service routine. mPORTBRead() is to read value from port B so we have a fresh start (kind of resetting it so we don’t have mismatch). ConfigIntCN(CHANGE\_INT\_ON | CHANGE\_INT\_PRI\_2); is to clear CN interrupt flag and set it to priority 2 (we can set it to priority 1 too if preferred). Then, lastly we call INTEnableSystemMultiVectoredInt();

The lines of code below the first part is simply the initial setup. We set both PWM (for left and right wheels) to go at 450. Then we enable both wheels to move forward (‘0’ for the left wheel to move forward and ‘1’ for the right wheel to move forward).



As we can see from the schematic above, the sensors now use CN2 through CN5 instead of RB0 through RB3. We used the middle-left and left most sensors as explained in the previous paragraphs so we only used CN3 and CN5.

After that, all we did is move our reading lines algorithm (our if-else statement) from the while loop to our ISR function as seen below:

void \_\_ISR(\_CHANGE\_NOTICE\_VECTOR, ipl2) MyInterruptServiceRoutine()

{

if ((PORTReadBits(IOPORT\_B, BIT\_3)) == 0) // if leftmost sensor S4 detects black

// turn left

{

SetDCOC2PWM(300); // Left wheel slower

SetDCOC3PWM(700); // Right wheel faster

}

else if ((PORTReadBits(IOPORT\_B, BIT\_1)) == 0) // if middle-right sensor S2 detects black

// turn right

{

SetDCOC2PWM(700); // Left wheel faster

SetDCOC3PWM(300); // Right wheel slower

}

else

{

SetDCOC2PWM(450); // Go straight

SetDCOC3PWM(450);

}

}

The ISR is called by a change in either CN3 or CN5. The algorithm is simple:

* If the left most sensor S4 detects black, then turn left (right wheel is faster at 700 while left wheel is at 300)
* If the middle-right sensor S2 detects black, then turn right (left wheel is faster at 700 while right wheel is at 300)
* If neither of the two sensors detect a black line, then both go straight at 450

Below is our full and final code for Lab #3:

/\*

\* File: Blink.c

\* Author: petrusn & ahtouw

\*

\* Created on September 26, 2016, 8:56 AM

\*/

// Code built off of prior labs

#include <stdio.h>

#include <stdlib.h>

#include <plib.h>

/\* all ports have names: \*/

/\* e.g. IOPORT\_A IOPORT\_B, etc. \*/

/\* all port bits have names: \*/

/\* e.g. BIT\_0, BIT\_1,... BIT\_14, BIT\_15, etc. \*/

/\* led 1 port RB10 \*/

/\* led 2 port RB11 \*/

/\* led 3 port RB12 \*/

/\* led 4 port RB13 \*/

/\*

\* This program is to control the motors on the MRK-LINE Robot, using feedback

\* from the optical sensors in the line-following accessory.

\* Here are the pertinent port connections:

\* Right wheel (assuming sensors are on front bumper)

\* DIR - RD6 (0: forward)

\* EN - RD2 --> OC3 active high

\* SA - RD10 SA and SB are quadrature encoded feedback

\* SB - RC2

\* Left wheel

\* DIR - RD7 (1: forward)

\* EN - RD1 --> OC2

\* SA - RD9

\* SB - RC1

\* Optical Sensors

\* S1 - RB0 S1 is rightmost sensor (CN2)

\* S2 - RB1 S2 is middle right sensor (CN3)

\* S3 - RB2 S3 is middle left sensor (CN4)

\* S4 - RB3 S4 is leftmost sensor (CN5)

\*/

// This ISR controls the motors to follow the black line using the optical sensors

// The ISR is activated any time either of the optical sensors changes from their current states

// Both wheels require power even when turning so that the turns are smooth and not too sharp

void \_\_ISR(\_CHANGE\_NOTICE\_VECTOR, ipl2) MyInterruptServiceRoutine()

{

if ((PORTReadBits(IOPORT\_B, BIT\_3)) == 0) // if leftmost sensor S4 detects black

// turn left

{

SetDCOC2PWM(300); // Left wheel slower

SetDCOC3PWM(700); // Right wheel faster

}

else if ((PORTReadBits(IOPORT\_B, BIT\_1)) == 0) // if middle-right sensor S2 detects black

// turn right

{

SetDCOC2PWM(700); // Left wheel faster

SetDCOC3PWM(300); // Right wheel slower

}

else

{

SetDCOC2PWM(450); // Go straight

SetDCOC3PWM(450);

}

}

int main(int argc, char\*\* argv) {

INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);

INTEnableInterrupts();

//Configure ports for onboard LEDs as outputs

PORTSetPinsDigitalOut(IOPORT\_B, BIT\_10 | BIT\_11 | BIT\_12 | BIT\_13);

PORTSetPinsDigitalOut(IOPORT\_D, BIT\_6 | BIT\_7 | BIT\_1 | BIT\_2);

PORTSetPinsDigitalIn(IOPORT\_C, BIT\_2 | BIT\_1);

// Useful functions: (see PeripheralLibraries pdf file for more)

// PORTSetPinsDigitalOut(IOPORT\_B, BIT\_10 | BIT\_11 | BIT\_12 | BIT\_13);

// PORTSetPinsDigitalIn(IOPORT\_B, BIT\_10 | BIT\_11 | BIT\_12 | BIT\_13);

// PORTClearBits(IOPORT\_B, BIT\_10 | BIT\_11 | BIT\_12 | BIT\_13); // clear all bits

// PORTSetBits(IOPORT\_B, BIT\_10 | BIT\_11 | BIT\_12 | BIT\_13); // set all bits

// PORTToggleBits(IOPORT\_B, BIT\_10 | BIT\_11 | BIT\_12 | BIT\_13); // toggle state of the bits

// Configure built in buttons as inputs

// On ProMX4 buttons are on RA6 and RA7

PORTSetPinsDigitalIn(IOPORT\_B, BIT\_0 | BIT\_1 | BIT\_2 | BIT\_3);

PORTSetPinsDigitalIn(IOPORT\_A, BIT\_6 | BIT\_7);

// Useful function: see DigitalIO project for more

// PORTReadBits(IOPORT\_A, BIT\_6); // read the state of button on RA6

// calling functions for output compares

OpenOC2(OC\_ON | OC\_IDLE\_STOP | OC\_TIMER\_MODE16 | OC\_TIMER2\_SRC | OC\_PWM\_FAULT\_PIN\_DISABLE, 500, 500); // Left Wheel

OpenOC3(OC\_ON | OC\_IDLE\_STOP | OC\_TIMER\_MODE16 | OC\_TIMER2\_SRC | OC\_PWM\_FAULT\_PIN\_DISABLE, 500, 500); // Right Wheel

OpenTimer2(T2\_ON | T2\_IDLE\_STOP | T2\_GATE\_OFF | T2\_PS\_1\_1, 0x000003E8); // period = 1000

/\* setup the change notice options \*/

mCNOpen(CN\_ON | CN\_IDLE\_STOP, CN3\_ENABLE | CN5\_ENABLE ,CN\_PULLUP\_DISABLE\_ALL );

/\* read port(s) to clear mismatch \*/

int value = mPORTBRead();

/\* clear change notice interrupt flag \*/

ConfigIntCN(CHANGE\_INT\_ON | CHANGE\_INT\_PRI\_2);

/\* enable multi-vector interrupts \*/

INTEnableSystemMultiVectoredInt();

/\* Initial Setup \*/

SetDCOC3PWM(450);

SetDCOC2PWM(450);

PORTSetBits(IOPORT\_D, BIT\_1 | BIT\_2 | BIT\_7); // Both Wheel Enabled, Right Direction Forward

PORTClearBits(IOPORT\_D, BIT\_6); // Left Direction Forward

while (1) // continuous loop

{

// left intentionally blank

}

return (EXIT\_SUCCESS);

}