12/06/23

Time: 7.5h

Monday, June 12, 2023 8:04 AM

Start 8:00AM

On Friday while I was trying to get the wiringPi library to work for the GPIO pins, I was seeing online that the library was no longer being maintained. For this reason, it may be better to try to find a new library.

Lgpio

- GPIO interaction
- I2C
- SPI
- Serial
- PWM
- http://abyz.me.uk/lg/lgpio.html

Pigpio

- GPIO
- PWM
- Serial
- http://abyz.me.uk/rpi/pigpio/

This lgpio library seems to have most of the functionality I am looking for.

Installing that and using it could be something I look into today. I would like to know I can interface with the hardware so I can start working on the higher level stuff

I went back to trying to install the Lucam Driver on the Pi. I thought there was an issue with the with a file from the package being missing. When I looked closer at the error message, I found that the file that was missing was not internal to the library, it was looking under a different folder. Turns out that the file has something to do with the "linux headers"

This is something that was mentioned in the readme but I did not know what it meant.

I researched this and found some commands that are supposed to install the files. However after running them I still have the same issues

12:30 ouch
1:00 back

Looking under the folder path that the installer was looking for, I can see that one of the sub folders "build" was not present. I have found that in a different version folder that build file is present. I may be able to redirect the installer to the version that has the paths its looking for.

I found where the build is referenced in the Makefile and changed it to the version that had a build folder. I got further this time but the installation of the lucam sdk still failed.

RPI Zero 2W Distributers

https://www.amazon.ca/Raspberry-Zero-Wireless-Bluetooth-2021/dp/B09LH5SBPS/ref=d_pd_di_sccai_cn_sccl_3_2/132-6606787-8620330?pd_rd_w=HkwZX&content-id=amzn1.sym.d6674fdf-bd00-4d07-8317-6dfd6c498cdf&pf_rd_p=d6674fdf-bd00-4d07-8317-6dfd6c498cdf&pf_rd_r=PYSA00W8DZ40XFCR83FP&pd_rd_wg=QgnON&pd_rd_r=79853c59-38a0-42f0-8eb5-965c5fb9e7ef&pd_rd_i=B09LH5SBPS&psc=1

~$160

It appears that the 2W is largely out of stock from most distributers. Most places only have them listed for $25, but they are all sold out. The amazon distributer is charging 6x the cost but does have some in stock.

I noticed that the RPI Zero W is much more available. I am not really sure what the main differences are. Maybe the RPI Zero W would also work.
https://www.pishop.ca/product/raspberry-pi-zero-w/?src=raspberrypi

I found that the 2W got a large increase in computing power but it seems like the power consumption and interfaces are similar.

Camera Distributor

I emailed the sales contact for the camera I found to inquire about the price and availability/leadtime for the camera.

MPLAB X IDE

Nick had mentioned that has a microchip board that has a similar chip to the one I had selected for the ADC. He mentioned that MicroChip has an IDE used for programming these chips.

I searched for MicroChip IDE and the MPLAB X IDE lab was the result. I downloaded the software.

Lgpio Library

I installed the new gpio library using the following instructions:
http://abyz.me.uk/lg/download.html

I created a super simple script that lets me control the IO pins. I was able to get a pin to turn on and off. This is more success than I had with the other library

Leave 4:00

13/06/23

Time: 7.5h

Tuesday, June 13, 2023 8:00 AM

Start 8:00 AM

At the end of the day yesterday, I was able to get the GPIO interfacing working using the lgpio library.

This morning I wanted to try some of the other aspects of the library for interfacing hardware.

One of those is the PWM interface. Reading the documentation, the pwn takes a lot more arguments than I am used to.

int lgTxPwm(int handle, int gpio, float pwmFrequency, float pwmDutyCycle, int pwmOffset, int pwmCycles)

This starts software timed PWM on an output GPIO.

```
      handle: >= 0 (as returned by lgGpiochipOpen)
        gpio: the GPIO to be pulsed
pwmFrequency: PWM frequency in Hz (0=off, 0.1-10000)
pwmDutyCycle: PWM duty cycle in % (0-100)
   pwmOffset: offset from nominal pulse start position
   pwmCycles: the number of pulses to be sent, 0 for infinite
```

If OK returns the number of entries left in the PWM queue for the GPIO.

On failure returns a negative error code.

Each successful call to this function consumes one PWM queue entry.

PWM is characterised by two values, its frequency (number of cycles per second) and its duty cycle (percentage of high time per cycle).

Another PWM command may be issued to the GPIO before the last has finished.

If the last pulse had infinite cycles then it will be replaced by the new settings at the end of the current cycle. Otherwise it will be replaced by the new settings when all its cycles are complete.

Multiple PWM settings may be queued in this way.

lgpioTest.cpp

```cpp
#include <lgpio.h>
#include <stdio.h>
#include <unistd.h>

using namespace std;

int main(){
    int h;

    h = lgGpiochipOpen(0);
    printf("Handle: %d\n", h);


    printf("Toggle LED\n");
    lgGpioWrite(h, 14, 1);
    sleep(2);
    lgGpioWrite(h, 14, 0);

    sleep(3);

    printf("PWM\n");
    lgGpioWrite(h, 12, 0);
    for(int i = 0; i < 100; i++){
        lgTxPwm(h, 12, 1000.0, 1, 0, 0);
        sleep(1);
        printf("Duty Cycle: %d\n", i);
    }
    lgGpioWrite(h, 12, 0);



    lgGpiochipClose(h);

    return 0;
}
```

There is the frequency, duty cycle (float), offset, and # of cycles. For my test program, I used the values 1000, 0-100, 0, 0.

I was able to get the PWM functioning, however I found that the pin needs to be written low before the PWM will work. This should not be a big deal, I just need to remember to always initialize the pin to low before attempting to use PWM.

Motor Drivers

Previously I was confused about what the motor drivers here were. I asked Jakob and he said that they are some Faulhaber motor drivers and found be the product page on their website.
https://www.faulhaber.com/en/products/series/mcbl-3002-p/

This is definitely the motor driver that I have sitting in front of me.

FAULHABER

Motion Controllers

V2.5, 4-Quadrant PWM
with RS232 or CAN interface

MCBL 3002 P

Values at 22°C	MCBL 3002 P	
Power supply electronic	U_N/U_N	8 ... 30 V DC
Power supply motor [1]	$-/U_N$	0 ... 30 V DC
PWM switching frequency	f_{PWM}	78,12 kHz
Efficiency electronic	η	95 %
Max. continuous output current	I_{cont}	2 A
Max. peak output current [2]	I_{max}	3 A
Standby current for electronic (at U_N=24V)	I_d	0,04 A
Operating temperature range		-25 ... +85 °C
Mass		7 g

[1] Only available for option 3085 (separate power supply)
[2] S2 mode for max. 5s

Interfaces	MCBL 3002 P RS	MCBL 3002 P CO
Interface	RS232	CAN (CiA)
Protocol	FAULHABER - ASCII	CANopen

Basic features

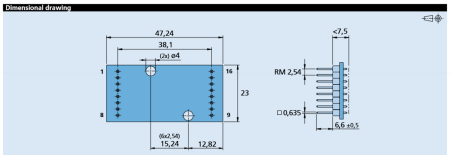
- Supported sensor systems: analog Hall sensors
- Positioning resolution when using analog Hall sensors as position encoder: 3000 increments per revolution
- Max. 3 digital inputs, max. 1 digital output, 1 analog input. Not all I/Os available depending on wiring
- Setpoint specification via fieldbus, quadrature signal, pulse and direction or analog inputs
- Optional stand-alone operation via application programs with the RS232 interface version

Range of functions

Operating modes (RS Versions)	Position, speed and torque control with setpoint specification via interface or analog. Position control with Gearing Mode or stepper motor operation. Operation as Servo Amplifier in voltage controller mode
Operating modes (CO Version)	Profile Position Mode (PP), Profile Velocity Mode (PV), Homing Mode. Cyclic Synchronous Position Mode (CSP) acc. to IEC 61800-7-201 or IEC 61800-7-301.
Speed range for brushless motors with number of pole pairs 1	5 min^{-1} ... 30 000 min^{-1} (with sinus commutation)
Application programs	Available in versions with RS232 interface
Additional functions	Overload protection for electronics and motor, self-protection from overheating, over-voltage protection in generator mode.
Indicator	Trace as logger
Motor types	Brushless DC-motors with analog Hall sensors and number of pole pairs 1 or 2

FAULHABER

Dimensional drawing



MCBL 3002 P

Options and connection information

Example product designation: MCBL 3002 P RS 3085

Option	Type	Description
3085	Supply	Separate power supply for motor and electronics

Connection

No.	Function	No.	Function
1	TxD / CAN_H	9	Sensor A
2	RxD / CAN_L	10	Sensor B
3	AGND	11	Sensor C
4	Fault	12	U_{Hall}
5	A_{VAN}	13	SGND
6	U_N	14	Motor A
7	GND	15	Motor B
8	3.3	16	Motor C

Note: For details on the connection assignment, see device manual MC.

Product combination

Brushless DC-Motors	Cables / Accessories
0620 ... B 1218 ... B 1226 ... B 1628 ... B 2036 ... B 2232 ... BX4 2250 ... BX4 2444 ... B 3056 ... B 3242 ... BX4	To view our large range of accessory parts, please refer to the „Accessories" chapter.

The datasheet does not provide a while lot of information.

No description of the pins is available aside from the abbreviated name. It Jakob mentioned that the motor driver was programmed using some software on his PC. I am not sure how to connect to the motor driver to do this... The driver has Rx and Tx pins for UART

I downloaded the software but I don't really have anything to connect the driver to my PC to actually interface with it.

I thought more about the PWM test script I made before. Allowing the adjustable frequency makes me think that maybe the PWM is being run off the CPU and not a hardware circuit.

I changed the pin to GPIO14 which is not a PWM pin. I found that the software was still functional, which makes me believe it is run off the CPU. This is not the most efficient... It will work for a proof of concept however another PWM method should be used before launch.

Went back to trying to get the Lucam SDK installed. I tried to go further with my work around of chosing a different kernel that has the build directory. I was able to get further but it seems that everytime I fix one issue another one is present.

I reread the README and saw that it specifies it is for kernel 5.15. The kernel version that is installed is 6.1. I am trying to install the 5.15 kernel and then try the first step of my work around to get the Lucam Makefile to check the 5.15 kernel instead of the current kernel.

https://www.cyberciti.biz/tips/compiling-linux-kernel-26.html

I tried following this guide for installing 5.15. It worked good until getting to the point where I copy the .config file

There is no config file in the directory they tell me to copy from, which means I cant complete that step, and I cant do the next step until I have this config file

Ximea Software

https://www.ximea.com/support/wiki/apis/XiAPI

From this, it appears that some example code is available for the setup and basic image acquisition from the camera. I am curious where the image actually ends up. Is it just stored in the RAM variables while the script runs? How do I save it to a file?

```
// image buffer
XI_IMG image;
memset(&image,0,sizeof(image));
image.size = sizeof(XI_IMG);
```

I guess it gets saved there. I know memset is used in C to manage memory.

So I think that it would be getting stored to the storage location addressed my image

I notice the & symbol is used infornt of image, which I am pretty sure means to use the memory location and not the value of image
12:00-12:30 lunch

I got an email back from the Ximea people. The camera will cost ~$2k. I began looking further into the software. I found that the Ximea drivers need a 64bit OS meaning that Raspbian can not be used.

It does not appear that the RPI Zero W supports 64bit OS, but the RPI Zero 2 W does support 64-bit. This means that the 2W would be needed.

I found Josh O'Niells Thesis and Nick recommended that I read through it. It should have more detailed on the ADCS systems and the theory behind them which is one thing I wanted to read more about. I spent some time reading it, however it is 150 pages so I will not get through all of it today.

Leave: 4:00pm

14/06/23

Time: 7.5h

Wednesday, June 14, 2023 7:59 AM

Start 8:00AM

At the end of the day yesterday, Nick mentioned that the camera I had selected did not have an operating temperature range. He also brought down brouchure for a Blackfly S camera. I know that I had seen this in my search, however I do not remember what I did not like about it. It may have been something I found later on and thought that cameras with similar specs were already on the list.

https://www.flir.ca/products/blackfly-s-board-level/?model=BFS-GE-50S5C-BD2&vertical=machine+vision&segment=iis

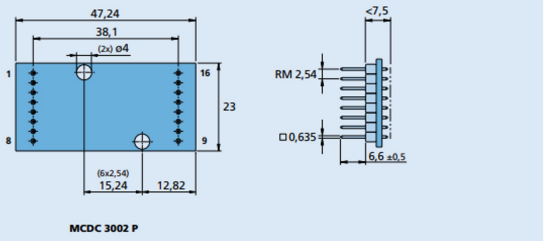
Overall the Camera does seem to be a good fit. It does have the operating temperatures listed as 0-50C however the CubeSat requirements say that a range of -20-50C is required…

I just went back to check the original camera (Python 5000) and it did meet this requirement.

Also looking at the software package from the camera, it is listing only 64bit OS's even though the specs said it works on 32bit as well. I am not sure but it may just be a better idea to go with the 2W board to make sure there are no issues and have the extra computing power.

Yesterday Nick brought me a cable to connect to the motor driver to program it. This motor driver has the worst documentation I have ever seen. There is a 2 page datasheet that doesn't even go into what the pins are. There is just a abbreviated name like *Us*. How am I supposed to wire this when I don't even know what the pins are.

Dimensional drawing



MCDC 3002 P

Options and connection information

Example product designation: **MCDC 3002 P RS 3085**

Option	Type	Description
3085	Supply	Separate power supply for motor and electronics

Connection

No.	Function	No.	Function
1	TxD / CAN_H	9	4. In
2	RxD / CAN_L	10	Ch A
3	AGND	11	Ch B
4	Fault	12	Ucc
5	AnIn	13	SGND
6	Us	14	Mot +
7	GND	15	Mot -
8	3. In	16	5. In

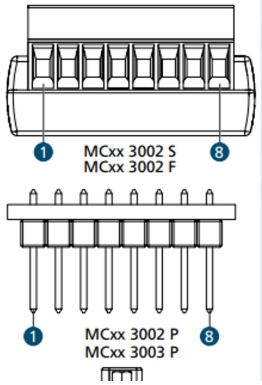
Note: For details on the connection assignment, see device manual MC.

This is essentially all the information I have to wire and program this device.

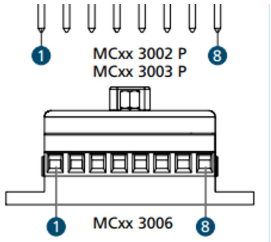
https://www.faulhaber.com/fileadmin/Import/Media/EN_7000_05038.pdf

4.2.3 Connections

4.2.3.1 Connections on the supply side (MCxx 3002/3003/3006)



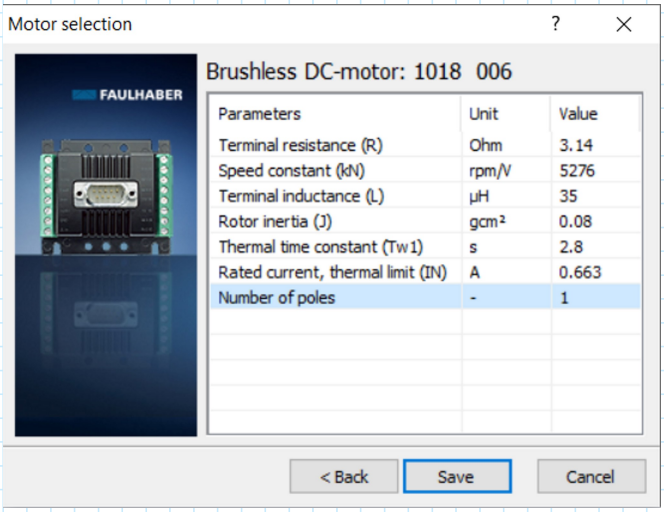
Pin	Designation	Meaning
1	TxD/CAN_H	RS232/CAN interface
2	RxD/CAN_L	RS232/CAN interface
3	AGND	Analog Ground
4	Fault	Fault output
5	AnIn	Analog input
6	U_B	Power supply for controllers
7	GND	Ground connection for controllers
8	3. In	3. Input / opt. separated power supply



In a completely separate document I found this which at least gives a little information about the pins

I was able to communicate with the Device in the Motion software,

I just needed to connect the Us and GND to a power supply with 10V and the Rx/Tx to the UART connecter, then join a ground from the power supply to the UART connector.



I am noticing that there is no pin on the driver to control the direction of the motor…

I connected the analog pins from my previous GPIO PWM experiment on the Pi and I was able to adjust the speed of the motor using PWM. It appears that instead of a direction pin, the PWM is varied to 0-100 duty cycle is -15k to + 15k rpm. I will have to check documentation to see if this is how it is supposed to work.

The motor also seems to run really rough at low RPM's. I can feel a lot more vibrations.

Nick had mentioned that the motor really needs to have the current controlled. This is because the current is proportional to the torque. If the inertia of the load is known, then the torque would be related to the inertia and the acceleration. It may be hard to control the acceleration with the change in velocity as there is going to be some time involved. We could make the time relatively slow to allow for control over the change in speed, but this is obviously slow.

Range of functions	
Operating modes (RS Versions)	Position, speed and torque control with setpoint specification via interface or analog. Position control with Gearing Mode or stepper motor operation. Operation as Servo Amplifier in voltage controller mode

The shitty datasheet says that the RS Versions have torque control, which this device is being detected as the RS model by the Motion software.

I cant not find in any documentation how to actually use this torque control. Everything just says that it exists. Maybe when I go back to Josh's thesis this will become more clear.

I made it through more of the Thesis

Here is the installation instructions for the Blackfly S camera on a Linux
https://www.flir.ca/support-center/iis/machine-vision/application-note/using-spinnaker-on-arm-and-embedded-systems/

I: 0 → Imax
PWM 0 → 100

I → PWM

$\frac{100.0}{I_{max}} (I)$

12:00 lunch

12:30 back

Went back to Josh's thesis

Meeting with Nick

Focus on selecting Camera
He likes the Ximea camera, wants to know if it will work
I know it can work with the RPI4, but nothing is listed about other boards
RPI Zero 2W can run the same OS, so I would assume it works but its hard to be completely sure.

Software

Operating system

It is possible to install the XIMEA software package on a system running the official **Raspberry Pi OS**, but there are some restrictions.
At the time of testing only the 32bit version of this OS has been officially released.
The XIMEA CamTool is not supported on 32bit operating systems, so it couldn't be used.
Ubuntu 21.04 (64 bit) is also available for the Raspberry Pi 4.
All components of the XIMEA software package including CamTool can be used on a Raspberry Pi when using Ubuntu 21.04 (64 bit).

Other distribution may also work with XIMEA cameras, but it was not tested.

Raspberry Pi 4 - new module

All current XIMEA USB cameras are compatible with the Raspberry Pi 4 Model B.
The XIMEA software package for Linux can be installed on the Raspberry Pi, but (depending on the choice of the operating system) the XIMEA CamTool may not be available.
Performance on the Raspberry Pi depends on the camera model as well as on the application.
It may not be possible to reach the advertised frame rate of a camera.

Aside from Camera, Nick is going to give me a microcontroller to try to program to control the pwm and read the ADC on the microcontroller. Also to communicate with the microcontroller over I2C.

This way the hardware can be controlled using the microcontroller which will have better pwm and ADC than the Pi would.

End 4:00

15/06/23

Time: 7.5

June 15, 2023 8:07 AM

Start 8:00AM

Look into datasheet for microchip

ADC

40.1.1 Port Configuration

The ADC will convert the voltage level on a pin whether or not the ANSEL bit is set. When converting analog signals, the I/O pin will be configured for analog by setting the associated TRIS and ANSEL bits. Refer to the **"I/O Ports"** chapter for more information.

Important: Analog voltages on any pin that is defined as a digital input may cause the input buffer to conduct excess current.

40.1.2 Channel Selection

The ADPCH register determines which channel is connected to the Sample-and-Hold circuit for conversion. When switching channels, it is recommended to have some acquisition time (ADACQ register) before starting the next conversion. Refer to the ADC Operation section for more information.

Important: To reduce the chance of measurement error, it is recommended to discharge the Sample-and-Hold capacitor when switching between ADC channels by starting a conversion on a channel connected to V_{SS} and terminating the conversion after the acquisition time has elapsed. If the ADC does not have a dedicated V_{SS} input channel, the V_{SS} selection through the DAC output channel can be used. If the DAC is in use, a free input channel can be connected to V_{SS}, and can be used in place of the DAC.

40.1.3 ADC Voltage Reference

The PREF bits provide control of the positive voltage reference. The NREF bit provides control of the negative voltage reference. Refer to the ADREF register for the list of available positive and negative sources.

40.1.4 Conversion Clock

The conversion clock source is selected with the CS bit. When CS = 1 the ADC clock source is an internal fixed-frequency clock referred to as ADCRC. When CS = 0 the ADC clock source is derived from F_{OSC}.

Important: When CS = 0, the clock can be divided using the ADCLK register to meet the ADC clock period requirements.

The time to complete one bit conversion is defined as the T_{AD}. Refer to Figure 40-2 for the complete timing details of the ADC conversion.

For correct conversion, the appropriate T_{AD} specification must be met. Refer to the ADC Timing Specifications table in the **"Electrical Specifications"** chapter for more details. The table below gives examples of appropriate ADC

40.7.1 ADCON0

Name: ADCON0
Address: 0x3F3

ADC Control Register 0

Bit	7	6	5	4	3	2	1	0
	ON	CONT		CS		FM		GO
Access	R/W	R/W		R/W		R/W		R/W/HC/HS
Reset	0	0		0		0		0

Bit 7 – ON ADC Enable

Value	Description
1	ADC is enabled
0	ADC is disabled

Bit 6 – CONT ADC Continuous Operation Enable

Value	Description
1	GO is retriggered upon completion of each conversion trigger until ADTIF is set (if SOI is set) or until GO is cleared (regardless of the value of SOI)
0	ADC is cleared upon completion of each conversion trigger

Bit 4 – CS ADC Clock Selection

	GO is cleared (regardless of the value of SOI)
0	ADC is cleared upon completion of each conversion trigger

Bit 4 – CS ADC Clock Selection

Value	Description
1	Clock supplied from ADCRC dedicated oscillator
0	Clock supplied by F_{OSC}, divided according to ADCLK register

Bit 2 – FM ADC Results Format/Alignment Selection

Value	Description
1	ADRES and ADPREV data are right justified
0	ADRES and ADPREV data are left justified, zero-filled

Bit 0 – GO ADC Conversion Status(1,2)

Value	Description
1	ADC conversion cycle in progress. Setting this bit starts an ADC conversion cycle. The bit is cleared by hardware as determined by the CONT bit

40.7.8 ADPCH

Name: ADPCH
Address: 0x3EC

ADC Positive Channel Selection Register

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

Bits 5:0 – PCH[5:0] ADC Positive Input Channel Selection

PCH	ADC Positive Channel Input
111111	Fixed Voltage Reference (FVR) Buffer 2(1)
111110	Fixed Voltage Reference (FVR) Buffer 1(1)
111101	DAC1 output(2)
111100	Temperature Indicator(3)
111011	V_{SS} (Analog Ground)
111010	DAC2 output(2)
111001 – 011000	Reserved. No channel connected.
010111	RC7/ANC7(4)
010110	RC6/ANC6(4)
010101	RC5/ANC5
010100	RC4/ANC4
010011	RC3/ANC3
010010	RC2/ANC2
010001	RC1/ANC1
010000	RC0/ANC0
001111	RB7/ANB7(4)
001110	RB6/ANB6(4)
001101	RB5/ANB5(4)
001100	RB4/ANB4(4)

This looks like how I assign what pin gets read. Also there is a byte to connect the ADC to Vss to ground it between samples.

40.7.15 ADRES

Name: ADRES
Address: 0x3EA

ADC Result Register

Bit	15	14	13	12	11	10	9	8
	RES[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

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

Bits 15:0 – RES[15:0] ADC Sample Result

Notes: The individual bytes in this multibyte register can be accessed with the following register names:

- ADRESH: Accesses the high byte ADRES[15:18]
- ADRESL: Accesses the low byte ADRES[7:0]

- ADRESL: Accesses the low byte ADRES[7:0]

This is how the result is accessed.

Example 40-2. ADC Conversion (C)

```
/*This code block configures the ADC
for polling, VDD and VSS references,
ADCRC oscillator and AN0 input.
Conversion start & polling for completion
are included.
 */
   void main() {
       //System Initialize
       initializeSystem();

       //Setup ADC
       ADCON0bits.FM = 1;         //right justify
       ADCON0bits.CS = 1;         //ADCRC Clock
       ADPCH = 0x00;              //RA0 is Analog channel
       TRISAbits.TRISA0 = 1;      //Set RA0 to input
       ANSELAbits.ANSELA0 = 1;    //Set RA0 to analog
       ADACQ = 32;                //Set acquisition time
       ADCON0bits.ON = 1;         //Turn ADC On

   while (1) {
       ADCON0bits.GO = 1;      //Start conversion
       while (ADCON0bits.GO); //Wait for conversion done
       resultHigh = ADRESH;    //Read result
       resultLow = ADRESL;     //Read result
   }
}
```

These 2 are still unclear

19.14.3 TRISx

Name: TRISx

Tri-State Control Register

Bit	7	6	5	4	3	2	1	0
	TRISx7	TRISx6	TRISx5	TRISx4	TRISx3	TRISx2	TRISx1	TRISx0
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 0, 1, 2, 3, 4, 5, 6, 7 – TRISxn Port I/O Tri-state Control

Value	Description
1	PORTx output driver is disabled. PORTx pin configured as an input (tri-stated)
0	PORTx output driver is enabled. PORTx pin configured as an output

Important:

- The TRIS bit associated with the $\overline{MCLR}$ pin is read-only and the value is '1'
- Refer to the **"Pin Allocation Table"** for details about $\overline{MCLR}$ pin and pin availability per port
- Unimplemented bits will read back as '0'

19.14.4 ANSELx

Name: ANSELx

Analog Select Register

Bit	7	6	5	4	3	2	1	0
	ANSELx7	ANSELx6	ANSELx5	ANSELx4	ANSELx3	ANSELx2	ANSELx1	ANSELx0
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 0, 1, 2, 3, 4, 5, 6, 7 – ANSELxn Analog Select on RX Pin

Value	Description
1	Analog input. Pin is assigned as analog input. Digital input buffer disabled.
0	Digital I/O. Pin is assigned to port or digital special function.

Important:

- When setting a pin as an analog input, the corresponding TRIS bit must be set to Input mode to allow external control of the voltage on the pin
- Refer to the "**Pin Allocation Table**" for details about pin availability per port
- Unimplemented bits will read back as '0'

28.4.2 Setup for PWM Operation

The following steps illustrate how to configure the CCP module for standard PWM operation:

1. Select the desired output pin with the RxyPPS control to select CCPx as the source. Disable the selected pin output driver by setting the associated TRIS bit. The output will be enabled later at the end of the PWM setup.
2. Load the selected timer TxPR period register with the PWM period value.
3. Configure the CCP module for the PWM mode by loading the CCPxCON register with the appropriate values.
4. Load the CCPRx register with the PWM duty cycle value and configure the FMT bit to set the proper register alignment.
5. Configure and start the selected timer:
   - Clear the TMRxIF Interrupt Flag bit of the PIRx register. See the Note below.
   - Select the timer clock source to be as $F_{OSC}/4$. This is required for correct operation of the PWM module.
   - Configure the TxCKPS bits of the TxCON register with the desired timer prescale value.
   - Enable the timer by setting the TxON bit.
6. Enable the PWM output:
   - Wait until the timer overflows and the TMRxIF bit of the PIRx register is set. See the Note below.
   - Enable the CCPx pin output driver by clearing the associated TRIS bit.

28.5.1 CCPxCON

Name: CCPxCON
Address: 0x0342

CCP Control Register

Bit	7	6	5	4	3	2	1	0
	EN		OUT	FMT	MODE[3:0]			
Access	R/W		R	R/W	R/W	R/W	R/W	R/W
Reset	0		x	0	0	0	0	0

Is this the output pin?

Bit 7 – EN CCP Module Enable

Value	Description
1	CCP is enabled
0	CCP is disabled

Bit 5 – OUT CCP Output Data (read-only)

Bit 4 – FMT CCPxRH:L Value Alignment (PWM mode)

Value	Condition	Description
x	Capture mode	Not used
x	Compare mode	Not used
1	PWM mode	Left aligned format
0	PWM mode	Right aligned format

The analog setup seems to be pretty straight forward. The pwm does not seem to bad except right now I am unclear for to assign the PWM output to a pin on the controller

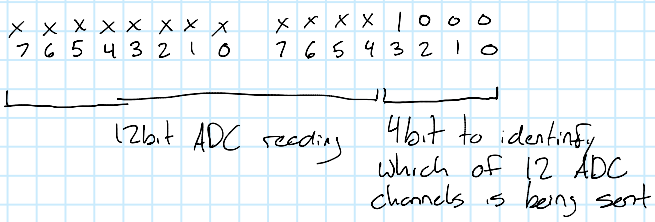
The analog setup seems to be pretty straight forward. The pwm does not seem to bad except right now I am unclear for to assign the PWM output to a pin on the controller

Sun Sensor I2C

The I2C is much less clear to me and I think will need to have some trail and error involved to figure out what is going on.

I2C Client

- recieves signal to send sunsensor data
- send 2 bytes / sensor

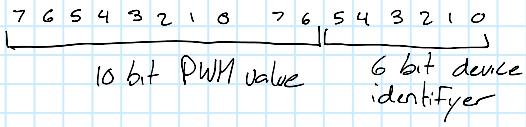


PWM I2C

I2C Client

- Send command to control PWM

2 Bytes / command



- Parse recieved bytes
- Use identifyer bits to determine which PWM output to write to
- Write 10 bit PWM value to the correct duty cycle register

According to the brief at the beginning of the data sheet, there are 3 PWM generators that each have independent dual outputs

Introduction

The PIC18-Q40 microcontroller family is available in 14/20-pin devices for real-time control applications. This family features a 12-bit ADC with Computation (ADCC) automating Capacitive Voltage Divider (CVD) techniques for advanced capacitive touch sensing, averaging, filtering, oversampling and threshold comparison and two 8-bit DAC modules. The family showcases a 16-bit Pulse-Width Modulator (PWM) module which provides dual independent outputs on the same time base. Additional features include vectored interrupt controller with fixed latency for handling interrupts, system bus arbiter, Direct Memory Access (DMA) capabilities, UART with support for asynchronous, DMX, Digital Addressable Lighting Interface (DALI) and Local Interconnect Network (LIN) protocols, Serial Peripheral Interface (SPI), I^2C and a programmable 32-bit Cyclic Redundancy Check (CRC) with memory scan. This family also includes memory features such as Memory Access Partition (MAP) to support users in data protection and bootloader applications and Device Information Area (DIA), which stores factory calibration values to help improve temperature sensor accuracy.

Digital Peripherals

- Three 16-Bit Pulse-Width Modulators (PWM):
  - Dual outputs for each PWM module
  - Integrated 16-bit timer/counter
  - Double-buffered user registers for duty cycles
  - Right/Left/Center/Variable-Aligned modes of operation
  - Multiple clock and Reset signal selections
- Three 16-Bit Timers (TMR0/1/3)

With 12 ADC pins, 5PWMpins, 2I2C pins and VSS and VDD, we are one pin to many to put everything on one IC.

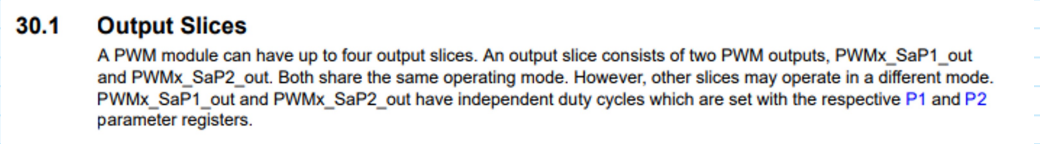
Instead there could be one controller dedicated to giving the sun sensor data, and another IC that is responsible for sending the PWM signals to the hardware.

All of these could interface to the RPI over one I2C interface.

Nick and I had discussed and when I get the IC, I could try to impliment a PWM signal. This can be easily validated on the oscilloscope in the lab. Once I know that is working, I could use a potentiometer to give a analog signal and use that to power the PWM. That way I could test the functionality of the analog read

The last step would be getting the I2C communication working.

Looking further into the datasheet, I see that 4 16bit timers are available. However from what I am seeing the PWM can only be controlled over the even numbered timers (Timer2 and Timer4).



30.1 Output Slices

A PWM module can have up to four output slices. An output slice consists of two PWM outputs, PWMx_SaP1_out and PWMx_SaP2_out. Both share the same operating mode. However, other slices may operate in a different mode. PWMx_SaP1_out and PWMx_SaP2_out have independent duty cycles which are set with the respective P1 and P2 parameter registers.

This is stating that each PWM module can have four output slices. Each slice can have 2 independent duty cycle outputs.

This is confusing.

I launched the IDE I had downloaded for Microchip and it prompted me that no compilers were installed.

I downloaded the compiler for the PIC18's and installed it.
https://www.microchip.com/en-us/development-tool/SW006011

I am getting errors that I need the XC8 Compiler so I downloaded and installed that
https://www.microchip.com/en-us/tools-resources/develop/mplab-xc-compilers/downloads-documentation#XC8

The basic template does not work. I am not dealing with this im just going to get Nick to give me his working example for go from there.

I created a new project and selected the Chip that will be used. Once making a main.c file, it will build. I started going through the data sheet to figure out how to control the PWM. I got through a bit but will need to finish tomorrow

Leave 3:30

16/06/23

Friday, June 16, 2023 8:03 AM

Time: 6.75h

Start 8.00

Nick is supposed to be bringing me the microchip MCU to program today

Josh's Thesis

Kelmer

Quaternions

$$q_1 = \cos\frac{\alpha}{2}$$

$$\vec{q}_{2:4} = [q_2 \quad q_3 \quad q_4]^T = \hat{a}\sin\frac{\alpha}{2}$$

In knowing the rotation of frame ***B*** as an axis-angle pair with respect to frame ***A***, the result would be $\boldsymbol{q}_{AB}$, or the quaternion from frame ***A*** to frame ***B***. Note that the result defines a unit quaternion, meaning that it is subject to the following constraint:

Much like rotation matrices, quaternions can also be inversed to get the reverse rotation and multiplied to combine rotations. As seen in the equations below, the inverse quaternion is found by simply negating the sign of either the scalar part or the vector part.

$$\boldsymbol{q}_{AB} = [q_1 \quad \vec{q}_{2:4}]^T = \left([q_1 \quad -\vec{q}_{2:4}]^T\right)^{-1} = \left([-q_1 \quad \vec{q}_{2:4}]^T\right)^{-1} = \boldsymbol{q}_{BA}^{-1} \tag{2.14}$$

Here, $\boldsymbol{i}$, $\boldsymbol{j}$, and $\boldsymbol{k}$ are three different imaginary numbers. Writing the multiplication with respect to this notation would be as follows:

$$q_A * q_B = (q_{A,1} + q_{A,2}\boldsymbol{i} + q_{A,3}\boldsymbol{j} + q_{A,4}\boldsymbol{k}) * (q_{B,1} + q_{B,2}\boldsymbol{i} + q_{B,3}\boldsymbol{j} + q_{B,4}\boldsymbol{k}) \tag{2.16}$$

Expanding and rearranging the right-hand side reveals the following equation:

$$\begin{aligned} q_A * q_B = {} & q_{A1}q_{B1} - q_{A2}q_{B2} - q_{A3}q_{B3} - q_{A4}q_{B4} \\ & + (q_{A1}q_{B2} + q_{A2}q_{B1} + q_{A3}q_{B4} - q_{A4}q_{B3})\,\boldsymbol{i} \\ & + (q_{A1}q_{B3} - q_{A2}q_{B4} + q_{A3}q_{B1} + q_{A4}q_{B2})\,\boldsymbol{j} \\ & + (q_{A1}q_{B4} + q_{A2}q_{B3} - q_{A3}q_{B2} + q_{A4}q_{B1})\,\boldsymbol{k} \end{aligned} \tag{2.17}$$

Converting this back to the matrix notation, it can be determined that:

$$\boldsymbol{q}_A \otimes \boldsymbol{q}_B = \begin{bmatrix} q_{B1} & -q_{B2} & -q_{B3} & -q_{B4} \\ q_{B2} & q_{B1} & q_{B4} & -q_{B3} \\ q_{B3} & -q_{B4} & q_{B1} & q_{B2} \\ q_{B4} & q_{B3} & -q_{B2} & q_{B1} \end{bmatrix} \begin{bmatrix} q_{A1} \\ q_{A2} \\ q_{A3} \\ q_{A4} \end{bmatrix} = \begin{bmatrix} q_{A1} & -q_{A2} & -q_{A3} & -q_{A4} \\ q_{A2} & q_{A1} & -q_{A4} & q_{A3} \\ q_{A3} & q_{A4} & q_{A1} & -q_{A2} \\ q_{A4} & -q_{A3} & q_{A2} & q_{A1} \end{bmatrix} \begin{bmatrix} q_{B1} \\ q_{B2} \\ q_{B3} \\ q_{B4} \end{bmatrix} \tag{2.18}$$

Where $\otimes$ is the standard operator for denoting the multiplication of quaternions when using matrix

Table 2-3: Methods of combining rotation quaternions.

	Compounding	Simple
$q_{n0} =$	$q_{n(n-1)} \otimes \dots \otimes q_{21} \otimes q_{10}$	$q_{10} \otimes q_{21} \otimes \dots \otimes q_{n(n-1)}$
$q_{0n} =$	$q_{01} \otimes q_{12} \otimes \dots \otimes q_{(n-1)n}$	$q_{(n-1)n} \otimes \dots \otimes q_{12} \otimes q_{01}$

For any unit quaternion, the following operation can be interpreted as the rotation of a vector from the ***A*** frame to the ***B*** frame.

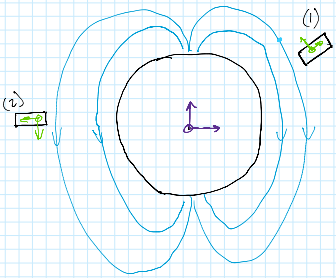
$$[0 \quad {}^B\vec{v}]^T = \boldsymbol{q}_{AB}^{-1} \otimes [0 \quad {}^A\vec{v}]^T \otimes \boldsymbol{q}_{AB} \tag{2.19}$$

Where $[0 \quad \vec{v}]^T$ is known as the vector quaternion [10].

Microchip Scripts

ADC

```
void main(void) {

    /*Configure Pin*/
    ADPCH = 0x00;       //RA0 analog channel
    TRISAbits.TRISA0 = 1;   //Set RA0 to input
    ANSELAbits.ANSELA0 = 1; //Set RA0 to analog

    /*Configure ADC*/
    ADCON0bits.FM = 1;   //Right justify
    ADCON0bits.CS = 1;   //ADCRC Clock
    ADACQ = 32;             //Set acquisition time
    ADCON0bits.ON = 1;      //Turn on ADC


    while(1){
        /*Read ADC*/
        ADCON0bits.GO = 1;  //Start acquisition
        while(ADCON0bits.GO); //Wait until done
        uint8_t resultHigh = ADRESH;
        uint8_t resultLow = ADRESL;

        /*Convert 2 8bit to 1 16bit*/
        uint16_t result = resultHigh;
        result = result << 8;
        result = result + resultLow;
    }
    return;
}
```



Where the satellite is is required to determine the correct quaternion is

The correct quaternion would be very different in positions (1) and (2). The quaternion could be determined entirely from a model.

The IMU will read earths magnetic field in body coordinates. The earths field in Earth centered coordinates is known meaning the body coordinates can be converted to earth coordinates. The rotation to do this transformation is the quaternion as measured by the satellite

need to know where satellite is vector in ECI changes with position

As for the sun sensors IF the sun is supposed to be behind the satellite, the vector to the sun can be measured and the vector from earth to the sun can be known from this the quaternion between the vectors

The accelerometer can also be used to point to the center mass of earth. IF the position is known, the expected gravitational field can be used to determine the quaternion

So, a model, needs to be used to determine position of the satellite. With a known position, the target quaternion can be defined

With the position known, the current quaternion can be defined using the Magnometer, accelerometer and maybe sun sensor

Would need to hold constant speed on reaction wheels and turn off magnetorquers while taking measurements

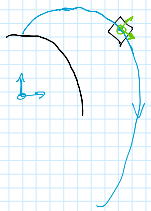
I know the SGP4 model takes a TLE file and approximates position how to go from position to ECI magnetic field, gravitational field and sun vector what does the propogator actually output? That may help me understand

```
>>> print(r)  # True Equator Mean Equinox position (km)
(-6102.44..., -986.33..., -2820.31...)
```

This is from the SGP4 python library — this suggests that a vector from ECI coordinates can be output from the SGP4

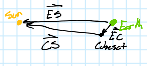
The accelerometer should give the same unit vector as the SGP4 output (as both are the vector from the Sat to the center of earth)

Not sure how the magnetic field vector is defined



As the SGP4 output is fixed to the stars, when the unit vector is within a certain range, the sun would be in range of the sun sensors

The sens unit vector should be constant, but it is not as easy to be a known reference cause it introduces a new vector



$\vec{ES}$ is constant

$\vec{EC}$ is output by SPG4

$\vec{CS}$ is the reference for sun sensor measurements

$\vec{CS} = \vec{ES} - \vec{EC}$

Now I understand the Sun and accelerometer Getting the reference magnetic field is still unclear

$$B(r) = \frac{\mu_o}{4\pi}\left[\frac{3\hat{r}(\hat{r}\cdot\vec{m}) - \vec{m}}{r^3}\right]$$

The magnetic field vector B as location r can be defined by knowing the vector of r, the permeability and the magnetic moment of the magnet

https://www.kjmagnetics.com/blog.asp?p=dipole

The only variable value in terms of earth is r which is output by the SGP4 propagator.

This means the SGP4 output can give a reference vector for the accelerometer, manometer and sun sensor using no new information (just system parameters)

This way the spacecrafts quaternion from its body center coordinates and ECI coordinates can be calculated.

I think I understand this, I want to explain all of this to Nick to make sure im understanding.

```
        result = result << 8;
        result = result + resultLow;
    }
    return;
}
```

PWM

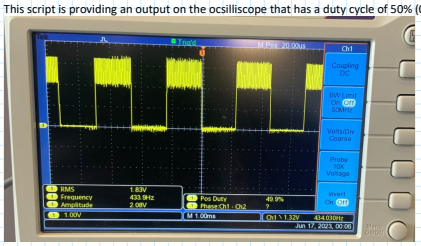
```
void main(void) {
    /*Setup Pins*/
    RA0PPS = 0x09;  //Assigns RA0 to CCP1 output
    TRISAbits.TRISA0 = 1;   //Disable output driver

    /*Configure PWM*/
    T2PR = 0xFF;            //Set timer pre-scaler for 10bit res
    CCP1CONbits.EN = 1;     //Enable CCP
    CCP1CONbits.FMT = 0;    //Right aligned PWM
    CCP1CONbits.MODE = 0b1100;  //Set mode to PWM

    /*Set Duty cycle*/
    uint16_t duty_cycle;
    duty_cycle = 0x0000;    //Define 16bit Duty cycle (only 10 relevant bits)
    CCPR1 = duty_cycle;     //Write duty cycle to register

    PIR3bits.TMR2IF = 0;       //Clear interrupt flag bit
    T2CLKbits.CS = 0b00001;     //Set the timer clock source to Fosc/4
    T2CONbits.CKPS = 0b11;      //Set timer pre-scaler value??
    //T2CONbits.RD16 = 1;       //Supposed to configure to 16bit or 8bit
    T2CONbits.T2ON = 1;

    /*Enable PWM*/
    TRISAbits.TRISA0 = 0;

    return;
}
```

12:00 lunch
12:45 back

PWM

```
void PWM(){
    /*Setup Pins*/
    RB7PPS = 0x09;  //Assigns RB7 to CCP1 output
    TRISBbits.TRISB7 = 1;   //Disable output driver

    /*Configure PWM*/
    T2PR = 0xFF;            //Set timer pre-scaler for 10bit res
    CCP1CONbits.EN = 1;     //Enable CCP
    CCP1CONbits.FMT = 0;    //Right aligned PWM
    CCP1CONbits.MODE = 0b1100;  //Set mode to PWM

    /*Set Duty cycle*/
    uint16_t duty_cycle;
    duty_cycle = 0x0200;    //Define 16bit Duty cycle (only 10 relevant bits)
    CCPR1 = duty_cycle;     //Write duty cycle to register

    PIR3bits.TMR2IF = 0;       //Clear interrupt flag bit
    T2CLKbits.CS = 0b00001;     //Set the timer clock source to Fosc/4
    T2CONbits.CKPS = 0b11;      //Set timer pre-scaler value??
    //T2CONbits.RD16 = 1;       //Supposed to configure to 16bit or 8bit
    T2CONbits.T2ON = 1;

    /*Enable PWM*/
    TRISBbits.TRISB7 = 0;

    return;
}
```

This script is providing an output on the ocsilliscope that has a duty cycle of 50% (0x0200 = 512/1023=50%)



```
void PWM(){
    /*Setup Pins*/
    RB7PPS = 0x09;  //Assigns RB7 to CCP1 output
    TRISBbits.TRISB7 = 1;   //Disable output driver

    /*Configure PWM*/
    T2PR = 0xFF;            //Set timer pre-scaler for 10bit res
    CCP1CONbits.EN = 1;     //Enable CCP
    CCP1CONbits.FMT = 0;    //Right aligned PWM
    CCP1CONbits.MODE = 0b1100;  //Set mode to PWM

    /*Set Duty cycle*/
    uint16_t duty_cycle;
    duty_cycle = 0x02FF;    //Define 16bit Duty cycle (only 10 relevant bits)
    CCPR1 = duty_cycle;     //Write duty cycle to register

    PIR3bits.TMR2IF = 0;       //Clear interrupt flag bit
    T2CLKbits.CS = 0b00001;     //Set the timer clock source to Fosc/4
    T2CONbits.CKPS = 0b11;      //Set timer pre-scaler value??
    //T2CONbits.RD16 = 1;       //Supposed to configure to 16bit or 8bit
    T2CONbits.T2ON = 1;

    /*Enable PWM*/
    TRISBbits.TRISB7 = 0;

    return;
}
```

0X02FF is 3/4*1023 which should be 75%


```
void PWM(){
    /*Setup Pins*/
    RB7PPS = 0x09;  //Assigns RB7 to CCP1 output
    TRISBbits.TRISB7 = 1;   //Disable output driver

    /*Configure PWM*/
    T2PR = 0xFF;            //Set timer pre-scaler for 10bit res
    CCP1CONbits.EN = 1;     //Enable CCP
    CCP1CONbits.FMT = 0;    //Right aligned PWM
    CCP1CONbits.MODE = 0b1100;  //Set mode to PWM

    /*Set Duty cycle*/
    uint16_t duty_cycle;
    duty_cycle = 0x0155;    //Define 16bit Duty cycle (only 10 relevant bits)
    CCPR1 = duty_cycle;     //Write duty cycle to register

    PIR3bits.TMR2IF = 0;       //Clear interrupt flag bit
    T2CLKbits.CS = 0b00001;     //Set the timer clock source to Fosc/4
    T2CONbits.CKPS = 0b11;      //Set timer pre-scaler value??
    //T2CONbits.RD16 = 1;       //Supposed to configure to 16bit or 8bit
    T2CONbits.T2ON = 1;

    /*Enable PWM*/
    TRISBbits.TRISB7 = 0;

    return;
}
```

Finally 33%

Leave 3:30

Task List

Monday, June 12, 2023 8:37 AM

Week of June 12-16 tasks/goals

- Successfully interface with gpio and PWM pins on Pi ✓ 13/06/23
- Use Pi to capture images in software (C++ Script)
- Obtain better understanding of ADCS and what background theory/math is involved
- Find supplier for RPI & Camera
- Figure out how to control torque using motor driver
- Learn to program PIC 18 chip

Questions

- What subsystem should be my focus?
-