EFIS PFD
Nathaniel Robinson

Introduction

The Electronic Flight Information System is the main system that integrates the PFD (Primary Flight Display) the EFIS also includes other instruments such as engine readings, fuel levels, flap and trim, and many other instruments. The PFD which is an artificial horizon is set on a blue and brown background the blue part which represents the sky is the top part and there is a middle between the two presenting the horizons, then the bottom part which is brown represents the ground. Also on the PFD is little segment of lines, the segment of lines means how fast you are ascending or descending in either feet or metres. For example, if you were taking off your rate of climb would be 20 feet or more. Also, the artificial horizon also roles of the plane therefore letting you know how steep your turn is a this the left or right.
The EFIS must have the instruments that have been said by law, law means you must have the ASI which is the airspeed indicator, the altitude, the artificial horizon, roll and yaw, heading (which is like a compass) and of area which is for the rates of climb and descend.
Another board that is set to be put in place as of this year for light aircraft is the installation of the G metre, the G metre which is the measurement of the gravitational pull is already an instrument which is installed in larger aeroplanes by law. This law has come in because of pilots not knowing how much G they are pulling during turns therefore blacking out and inevitably dying because they have blacked out.
With analogue versions, you can only get the artificial horizon, however with the electronic version which is the electronic flight information system it has the speed of the aircraft and the altitude of the aircraft display the same time, with some more expensive models you can have engine readings and other instruments of your choice displayed on the screen to.
The reason for me deciding to make her PFD is that the market for making these PFD's is extraordinarily expensive. The cheapest that you can get which is the basic model which is usually just the artificial horizon and may be altitude and speed depending whether it has GPS or not, is £750. The more expensive models vary from prices of £5000-£30,000, some models may be more expensive than this due to the complexity of them.
With my project, I am aiming to spend no more than £250. This means that if someone would want to buy a cheaper one they would have the ability to do so through my project.

Table of Contents

Potential Solutions

Hardware solution
The potential solution that you could use to make the EFIS PFD is that use hardware, this way of making it would be a vastly expensive project. This is because you must use mechanical parts to measure the readings accurately, this is because you would not be able to program the sensors that are required electronically easily, without using programming. Some of the mechanical parts are extortionate prices, for example just for the airspeed indicator would cost well over £300 this is because it comes as one unit and this unit is a very important part of the plane therefore companies will make them expensive. Other instruments that would be needed are the altitude indicator, a yaw and roll indicator and the artificial horizon. This method would be an inappropriate method because it would not be meeting the requirements to be electronic, this is mainly because it is expensive to convert the mechanical readings to a digital format.

Software solution
The other potential solution is to make the project is using software, there are already many software versions out there which can be seen in Microsoft flight simulator or in explain and many others. This method would be a waste of time, this would be because I don't have the level of expertise that the computer programmers had to program the flight simulators. Another reason for using this method is that it would require a graphics heavy computer meaning that it needs have reasonable graphics and to accommodate this it will require a good CPU and plentiful RAM, which would cost a lot more than £250.

Final solution
The other solution, which is the method that I will be using, is the mixed modules and circuit building. This method uses modules and some circuit building to meet the requirements for the project. This allows me to use already made sensors that I can integrate into a circuit that could be already built board that I could build myself. It also allows it to be a more cost-effective way which allows it to be more efficient and less time-consuming. In addition to this I will be able to build the project module by module which allows me to progressively build the project. This design also allows me to learn how to code the different modules.

Specification

The basic operation of the device – the basic operation of the device is to gather data from sensors and process the data and output the relevant values that can be understood, this could be shown on either the computer screen or on an LCD screen. The input data may need to be processed, this means that a calculation may be involved to work out the end value. The output data to come under the following titles; speed, altitude, roll, pitch, yaw.

Source of supply – the source of supply will be through a USB cable; this USB cable could be connected to a battery which would have a converter which will be adjusted so that it outputs the correct voltage needed.

Outputs, inputs and advanced features –

Outputs;

- Must have an interface
- Should have an external display, such as a LCD display showing numbers and characters
- Could have a Colour LCD display showing a graphic artificial horizon

Inputs;

- Must have; Accelerometer + gyro to track the pitch, roll, yaw, speed
- Could have; A temperature sensor to show the pilot the current temperature.
- Could have; Compass to display which direction the plane is heading
- Should have; Altitude/ barometric sensor to detect the altitude the plane is at.
- Should have; Light sensor, like an ambient light sensor to adjust the screen brightness.
- Should have; Buttons so that the pilot can interact with the device.

Advanced features

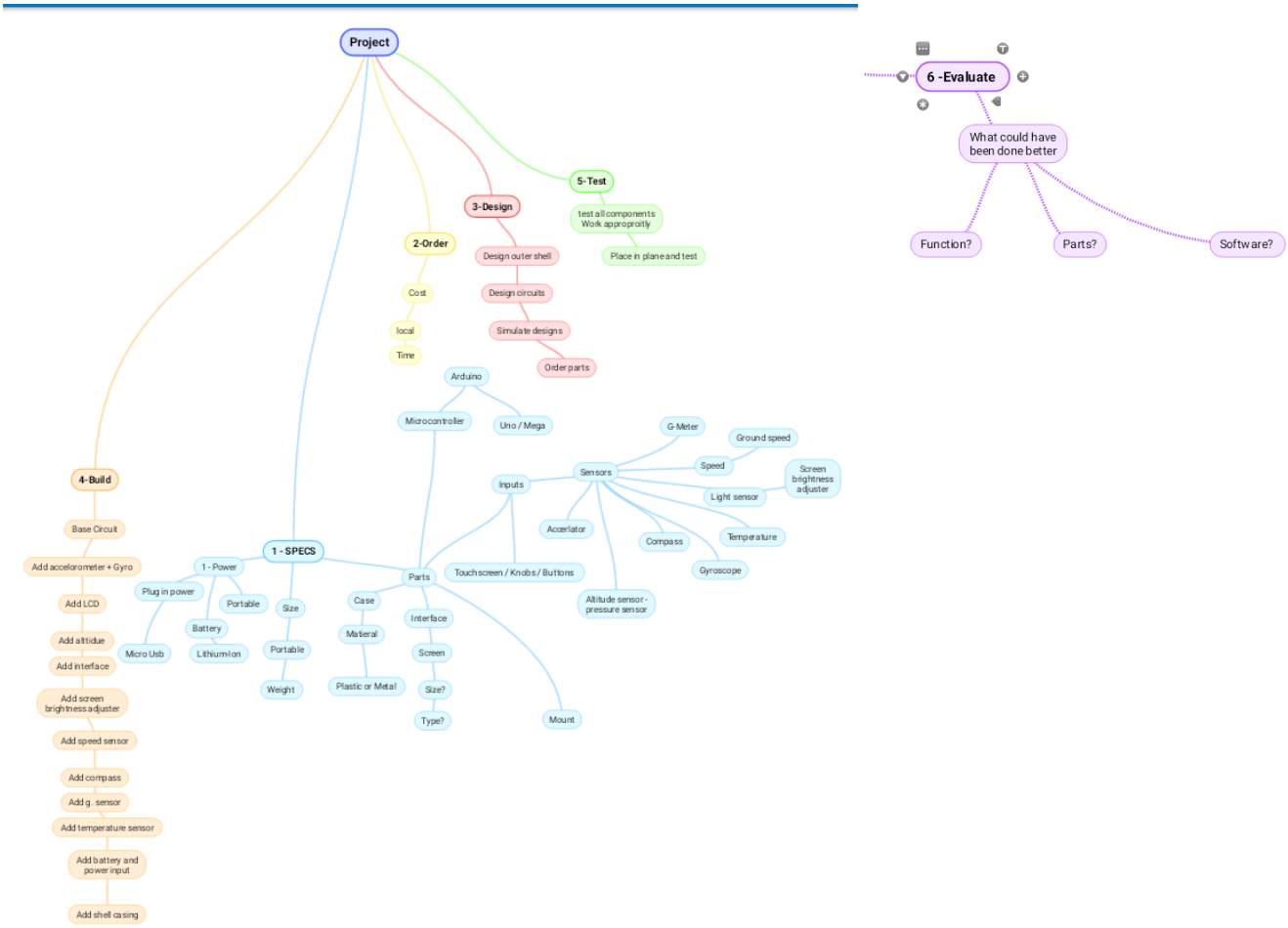
- Could have a WI-FI module so that you could connect to the internet to be able to access the plane database
- Could have a speaker, this will allow for the system to tell you when you're at low altitude, at high speed or if you're pitch is too high, and if a speed sensor was to be used then an alert system for stalling would be introduced. This would work by having an Internet database that it could go to find the user is plain that they have entered.

Overall size – It must be portable, so it must be less than 10 inches. So, the device can be between 4 inches to 10 inches.

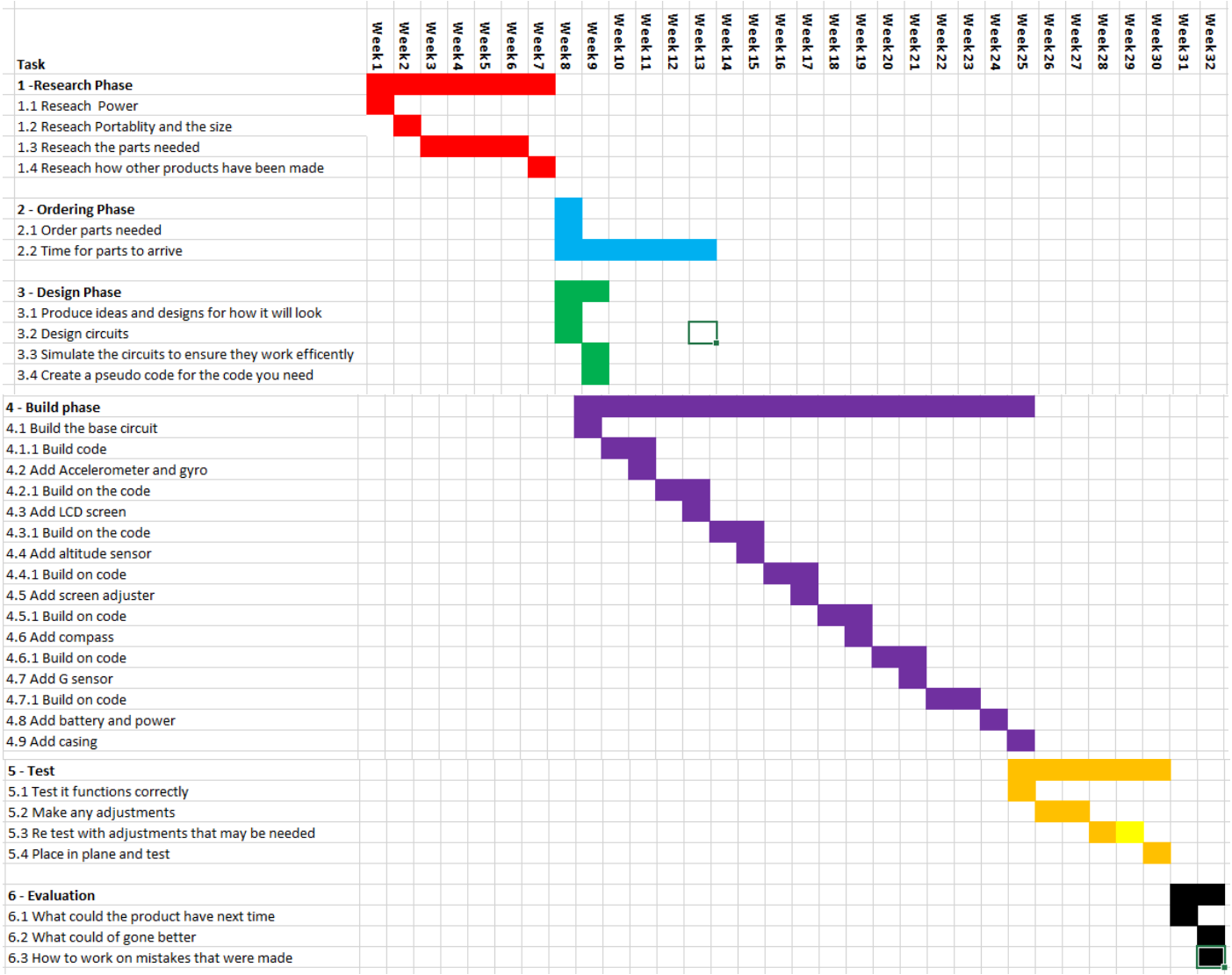
The initial price forecast:
Accelerometer sensor – £ 8
Barometer/pressure sensor – £15
Ambient light sensor – £3.50
Compass - £12
All in one sensor - £38
LED display - £30 - £70
Arduino - £15
Battery - £10

Initial Project Plan

Work Breakdown Structure



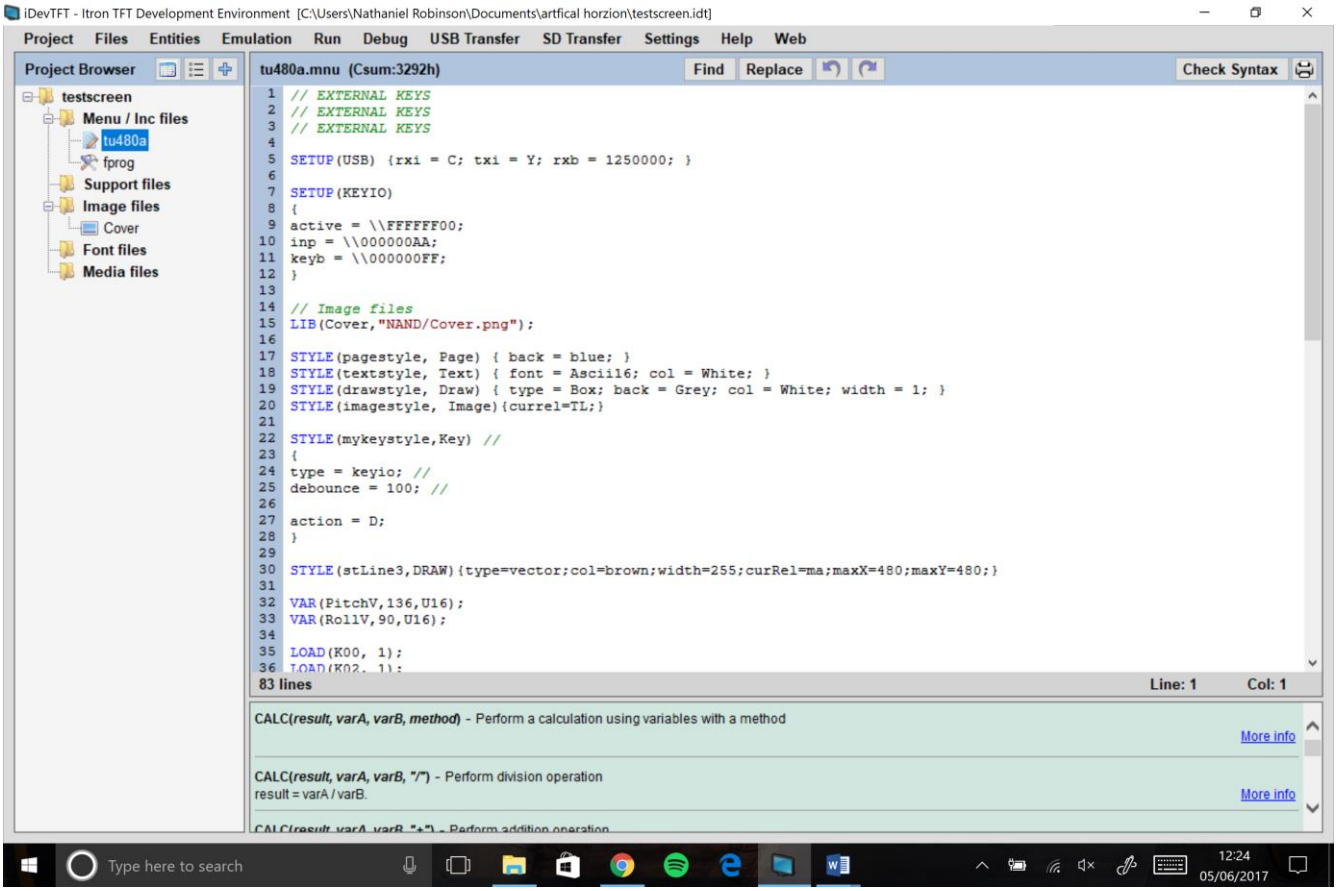
This shows clearly what order the tasks are set, and what steps I will be taking to progress the whole project. The structure also includes a mind map to show the various parts that could be used.



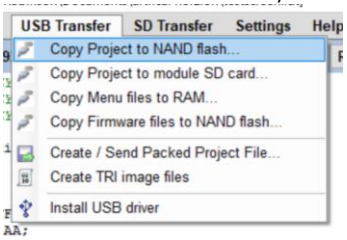
This GANTT chart shows how I have spilt up the tasks into different time slots, and giving the design phase the largest amount of time as I am giving extra time, as it is extremely likely many problems will occur during the project.

Software Design

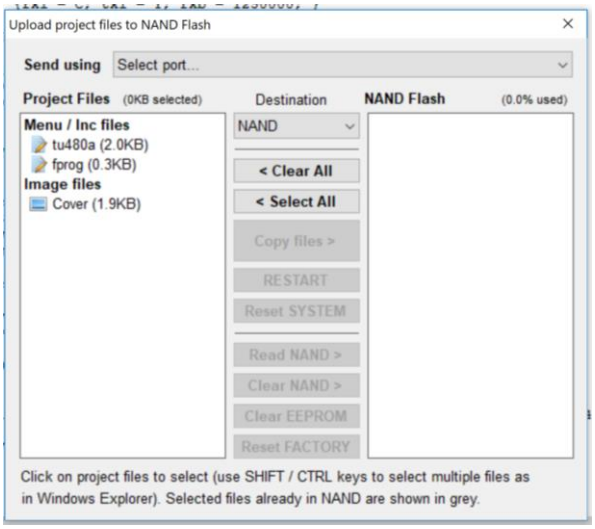
The software I used was IdevOS, it was one that was must be used with the TFT screen:



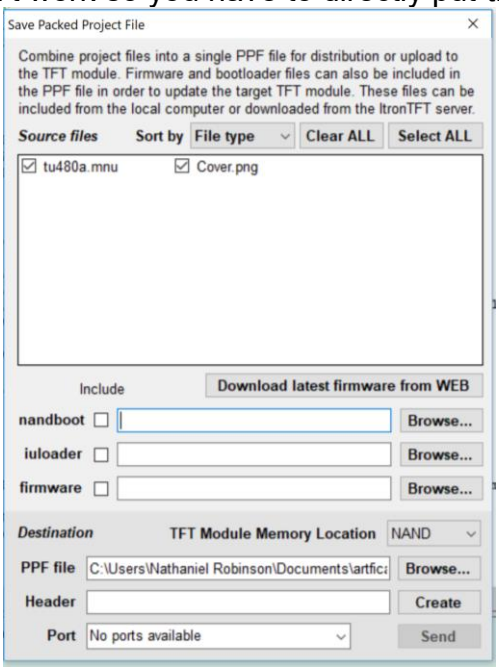
Its layed-out similaly to other programs like this, it has the folders on the left with the sub-files, then theres the code on the right and at the bottom you can highlight a bit of code and it will jump to that word in its index and will give you a short description of the code.
In this program you can do other functions like add images that go in the image folder.
Other functions are; transfer of files via the USB



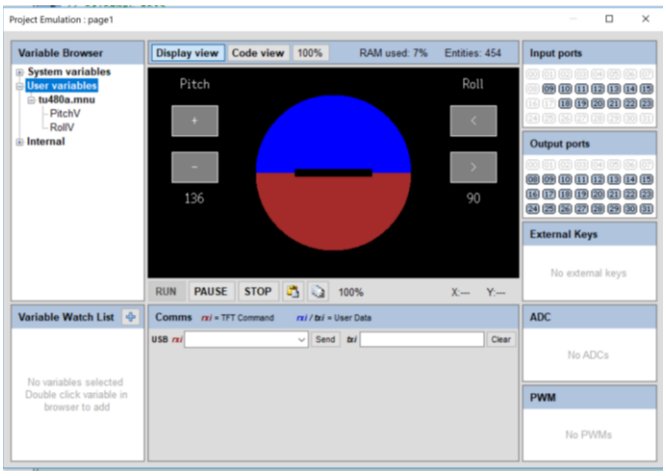
Usually I use the top one (copy project to NAND flash)



However, sometimes it doesn't work so you have to directly put the files on the memory via:



Theres also an emulation function that allows you to test the program before uploading to the screen. Therefore you can see if theres any bugs in the program which will save you from an error occouring on the screen which you would have to use the SD module to reset the screen.



http://www.noritake-itron.com/NewWeb/TFT/iDev%20Programing%20Guide%20(incomplete).pdf

Recommendations

I have learnt from this experience that not everything goes to plan as you expect it to, for example I had planned for parts of the project to go quicker than they did, for example being able to get the TFT screen to show graphically. To which I presumed it would take a week or two however it took well over a month.
Another experience I have learnt from is to start from scratch and take each part one at a time rather than throwing myself in at the deep end. From what I have encountered during the project if I had taken it one step at a time I might have got a lot further.
However, I enjoy challenging and pushing myself as I cope better when under pressure which is the reason why I threw myself it to the deep end.

There are many things I would do differently if I was to do this again.

1) I would use a different screen company, I should have used AndersDX as they work with schools and universities to make sure that anyone can use their screens, plus you can use an Arduino with it
2) I would take my time in the different stages
3) I would change the way I approach how I did things, for example I should have worked on making a battery pack for the device too and a low voltage indicator
4) Another recommendation would be to take each step one at a time, this means working on each stage progressive so when I compete one thing to then add on to that. For example, with the Arduino I shouldn't have taken the giant leap from the Arduino system to the TFT screen system. If I had done it step by step I wouldn't have encounter the problems that I did.
   This recommendation would be appropriate because it would reduce the amount of problems for one as I would be encountering, to which I had them all at once and I would be learning more from these problems as they are part of the progress of a project. Therefore, the time taken may have been reduced which could have produced a working project.
5) An additional recommendation would be an alternative method, this alternative method is a mechanical method this could work by having a rolling ball on a platform which would be controlled by two motors/servos which control the tilt and pitch. And then have two screens showing the other information such as altitude and speed and such like. This would be an easier way in terms of programming, however the build would be much harder.
   This recommendation would be appropriate as it would be an easier approach as it would be visual representation of the graphical visual, this would be better as it would have taken less time to program and it would have been an easier process to progress with, which with my project it could have been easy to progress but my project should have the graphical display to regulate with EFIS PFD CAA (Civil Aviation Authority). The mechanical method would have been cheaper to build and make, in addition I would have been able to make a circuit and circuit schematics.

Conclusion

The project overall may not have been as successful as I had of hoped however I am pleased with how far I got with it considering that it should be an RTOS (Real Time Operating System) and considering the amount of difficulties I had with the project, the screen code and the accelerometer being the biggest of troubles. I am also pleased with what I have learnt from the process. Yes, I wasn't successful but I have learnt what I did wrong and how I could change them in the future. To which I am hoping I can come back to this project in a years' time with more experience after my first year at university in Aerospace Systems Engineering and hope to be successful with.

My initial plan was to spend under £250, the table below show the total amount spent.

Parts	**Product**	**Source**	**Price**	
Microcontroller	Arduino Uno	Brought before hand so no cost	£-	
Three-axis accelerometer	ADXL 335	Brought before hand so no cost	£-	
7 segment display		Brought before hand so no cost	£-	
LCD module	1602 LCD	Brought before hand so no cost	£-	
Temp and humidity sensor	DHT11	Brought before hand so no cost	£-	
3 x breadboards		Brought before hand so no cost	£-	
jump cables		Brought before hand so no cost	£-	
USB cable		Brought before hand so no cost	£-	
4x buttons		Brought before hand so no cost	£-	
9 DOF sensor module	LSM9DSO	Amazon UK	£39.00	
Integrated Circuit LCD	TU480	Noritake Itron	£101.00	
Three-axis accelerometer I2C	ADXL 345	Amazon UK	£6.99	
Three-axis accelerometer replacement	ADXL 335	Amazon UK	£6.29	
LCD module replacement I2C	LCD 20x4	Amazon UK	£9.99	
			£163.27	**Total**

EFIS PFD Project Diary

By Nathaniel Robinson

Week 1-7

Research
For the first 7 weeks, I intended on spending this time researching how the project will work best and finding the various solutions to make the product.
I used the research time to find out what products were already on the market, ranging from the more expensive models to the cheap models. This research meant I could form ideas on how it should work, look and how it would sit in the cockpit. Many of the cheaper models were ones you could attach to the cockpit panel via an attachment so that it can be taken from plane to plane or there were permanently attached into the cockpit panel.

http://www.gps.co.uk/dynon-d2-pocket-panel--portable-efis/p-0-1468/?gclid=google&utm_medium
This image is of one of the cheapest portable EFIS PFD it cost's £1020 the feature that this device has;

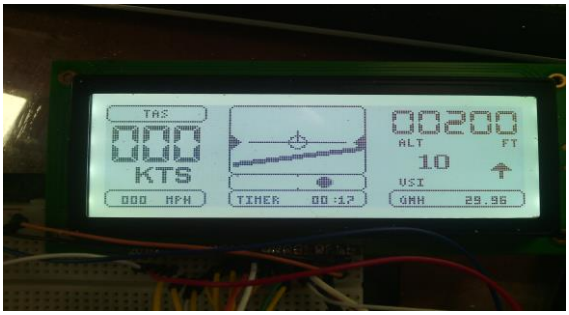
- artificial horizon
- Internal Li-Ion battery and GPS
- Versatile portable mounts: RAM suction mount and 3.125" panel hole "pinch" mount
- GPS ground speed and track (heading)
- GPS altitude and vertical speed
- Dimmable screen for night flight
- Wi-Fi connectivity

The higher priced end of the market has many more features than the product above and the product itself looks more premium.

AF-4500 (8.4" Display)

https://www.pilotshop.com/pages/av/efis_0browse/af3400_efis_6_5.php

As you can see from the image above it is a more advanced product compared to the last product. The product is integrated into the cockpit panel and isn't portable, but there are many differences between the products as with this one it has a 3D map of the horizon so you are able to see where you are in accordance of the runway, altitude, speed, engine readings and many other features. This product costs £2,856 for the basic EFIS system with no other functions, for the other functions such as engine display it would start from £8,554.32 and the highest spec costs £ 26,619.45. The market for such devices is a growing one therefore I am aiming to make a budget market product so that for flying enthusiasts such as myself can't afford such products.



Other people have made projects like the one I am doing, as see in the picture above however most of them aren't fully graphic.

During the research process, I had to discuss my project ideas with other pilots to see what they thought of my ideas so far and whether they had other ideas on how to make it simple but still meeting the demand of the market. From discussing these ideas, I generated an interest from this pilots that were keen to see if I could make one or not. In addition to this I had 2 pilots that requested that if I was to be successful that they would be interested to buy one to be placed in their own aircrafts. From this research, I formed specifications and requirements for the project.

Once I was satisfied with the outcome of the research I reorganized the mind map so that it would form the WBS. Once I had formed the WBS I started on the GANTT chart so that I would know the time frame I had to make the project.

Week 8

Accelerometer coding
Once I had the specification, I could start to order parts and start to prototype the build. The first step I took was to learn how to get a 7-segment display to work, this I did with the use if the college parts as it was a part that I wouldn't need for the final build. I did this so that I could see how it would look to have number and letters displayed on the units, to which it would have been to slow to show the required data on the 4 7-segment displays. To be able to meet the must dos of the project, specific data must be shown, it doesn't have to be graphical it can be numeral or character based, but must show it as an all in one on the screen or two slides on the screen for the information. Whereas if I was to use the 7 segment displays it will take many slides/lines to show the specific information. Such as roll degrees either negative = left and positive = right, and for pitch it was negative for down and positive for up.

Week 9

This week I started coding a LCD screen, a small one that the college had, I got it to show the same information I used for the 7-segment display. It was a step up from the 7 segments but the writing is too small to be seen by a pilot and isn't bright enough. I

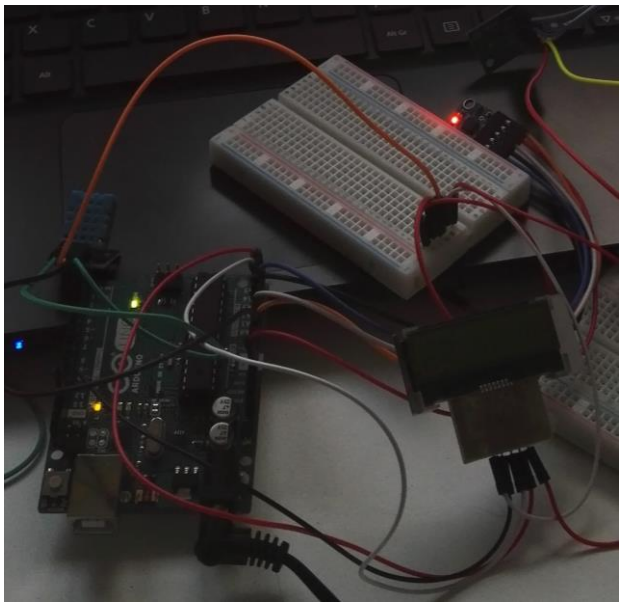
```
#include <LiquidCrystal.h>  // this includes a library

LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

#include <Wire.h>

void setup()   {

          lcd.begin(16, 2); // setting the LCD up so that it is 16 characters by 2 lines

         }

void loop() {

         lcd.clear();

         lcd.print ("EFIS PFD testing");

         delay (2000); //delay by 2 seconds

         }
```

The next step I took was to add the accelerometer, the accelerometer was just a basic sensor:

As you can see from the image it has 3 outputs and power and ground, I used this sensor to start off with as it was the simplest to start with, as you only have the 3 outputs to handle. My plan is to start simple and gradually make it harder as I gain more knowledge.

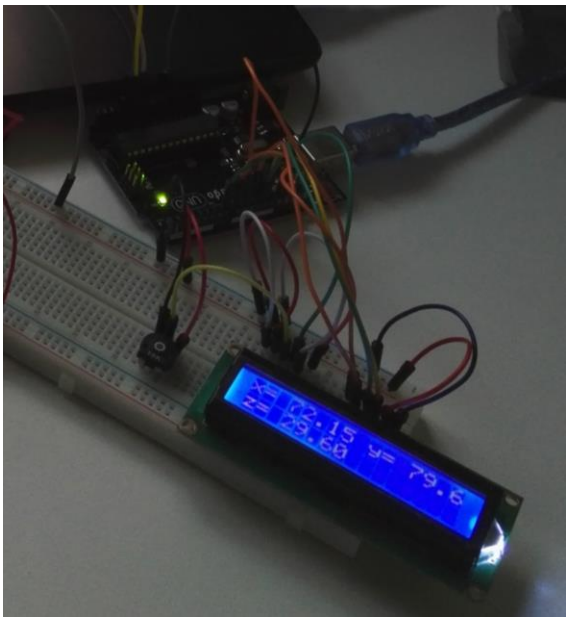
The display was originally coded to update every second but after testing the code I soon realised that this was a short amount of time, so I adjusted it to 3 seconds which made it easier to read the screen.

The next step is to increase the accuracy as the sensor doesn't calibrate properly and sometimes reads strange angles, then the next step is to add a larger LCD screen



Week 10

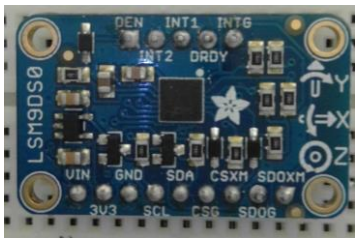
The objective for this week is to add a larger LCD screen and to improve the accuracy of the accelerometer and adding the temperature and humidity sensor.
I brought a larger back lit LCD which can be read more easily than the last LCD screen and the backlight can be adjusted depending on the brightness of the environment. I tested the screen with the previous code and the accelerometer, which worked better as the words and numbers could be read more clearly. I also decided to add the temperature and humidity sensor, this is a good senor to add as it could be used in two ways, 1) to show the pilot the current temperature and humidity of the cockpit, which would be essential for a glider pilot when you are at high altitude so that you can be aware of how cold, the 2nd would be that you could have the sensor put on the outside of the plane so that you could be aware of the outside temperature. The accuracy of the sensor was more accurate than I thought it would be considering it was a rather cheap sensor. Which the accelerometer was also, however not quite as accurate.
Because the accelerometer isn't as accurate as I would have hoped I set out to order a new accelerometer. The one I order is called 9 DOF sensor, this means it is a 9-axis sensor. The sensor is made up of an accelerometer sensor, magnetic field sensor, gyroscope sensor, temperature sensor and a linear acceleration sensor. I brought this specific one as it has the linear acceleration sensor which could be used to calculate the acceleration of the aircraft and calculate the speed from that.
The coding for this sensor is more difficult than it previously was as there are more sensors to code for, hence why I have chosen to do one at a time then once I have a hang of it I will bundle all of them together.



Week 11

9-Axis senor coding

During this week, I started to code the accelerometer 9 DOF:



The code as I have stated in last week's log is more difficult but I was wrong after reading the datasheet and seeing what the examples were it wasn't as hard as I once thought. The sensor is an I2C, which makes it a lot easier to call the sensors and read them. This therefore makes it quicker to respond and is a lot more accurate than the previous sensor.

Due to a solder error with the LCD screen I had to order a replacement, to which I decided I would get a larger one than worked on I2C to.

But in the meanwhile, I have been using the much smaller LCD screen and programming the sensor much further.

The new LCD screen arrived, to which I started to program straight away. Although I originally thought it would be much easier to use than using the previous LCD screen it became clear than I was undoubtedly wrong. Because the screen in larger and can show 20 characters on 4 lines it should be done via coordinates to show where each line and word will start. Which has defiantly used more time than I had anticipated.

The datasheet for the 9-DOF:

Figure 1. Block diagram

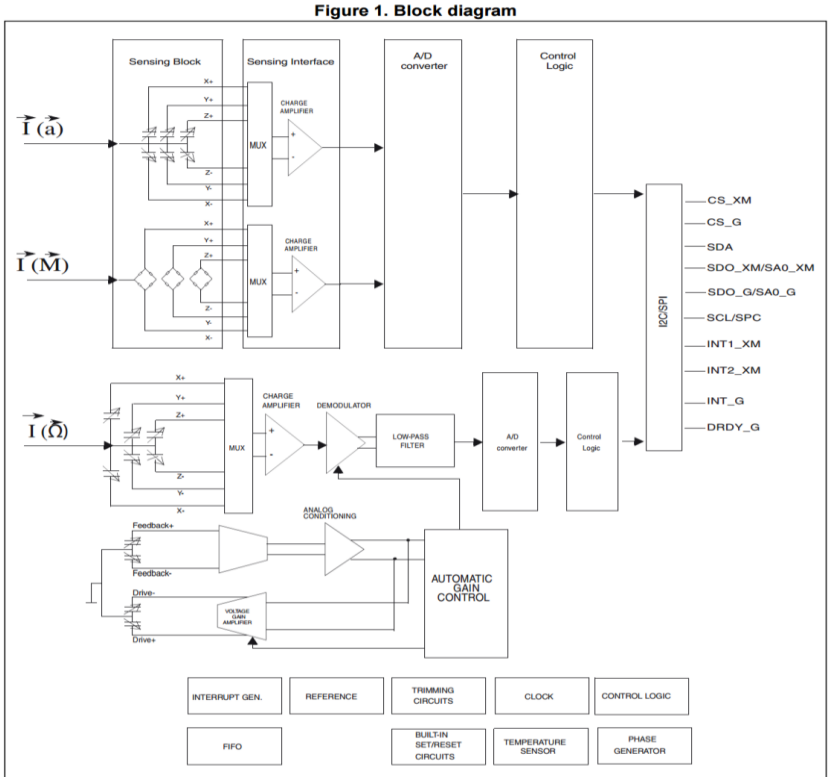


Table 2. Pin description

Pin#	Name	Function
1	Reserved	Leave unconnected
2	Reserved	Connect to GND
3	Reserved	Connect to GND
4	Reserved	Connect to GND
5	GND	0 V supply
6	GND	0 V supply
7	C1_XM	Capacitor connection (C1)
8	SETC_XM	S/R capacitor connection (C2)
9	SETP_XM	S/R capacitor connection (C2)
10	DEN_G	Gyroscope data enable
11	INT_G	Gyroscope programmable interrupt
12	DRDY_G	Gyroscope data ready
13	INT1_XM	Accelerometer and magnetic sensor interrupt 1
14	INT2_XM	Accelerometer and magnetic sensor interrupt 2
15	Vdd	Power supply
16	Vdd	Power supply
17	Vdd	Power supply
18	Vdd_IO	Power supply for I/O pins
19	CS_G	Gyroscope I^2C/SPI mode selection 1: SPI idle mode / I^2C communication enabled 0: SPI communication mode / I^2C disabled
20	CS_XM	Accelerometer and magnetic sensor SPI enabled I^2C/SPI mode selection 1: SPI idle mode / I^2C communication enabled 0: SPI communication mode / I^2C disabled
21	SCL SPC	I^2C serial clock (SCL) SPI serial port clock (SPC)
22	SDO_G SA0_G	Gyroscope serial data output (SDO) Angular rate sensor I^2C less significant bit of the device address (SA0)
23	SDO_XM SA0_XM	Accelerometer and magnetic sensor SPI serial data output (SDO) Accelerometer and magnetic sensor I^2C less significant bit of the device address (SA0)
24	SDA	I^2C serial data (SDA)

Table 3. Sensor characteristics

Symbol	Parameter	Test conditions	Min.	Typ.(1)	Max.	Unit
LA_FS	Linear acceleration measurement range(2)			±2		g
				±4		
				±6		
				±8		
				±16		
M_FS	Magnetic measurement range			±2		gauss
				±4		
				±8		
				±12		
G_FS	Angular rate measurement range			±245		dps
				±500		
				±2000		
LA_So	Linear acceleration sensitivity	Linear acceleration FS = ±2 *g*		0.061		mg/LSB
		Linear acceleration FS = ±4 *g*		0.122		
		Linear acceleration FS = ±6 *g*		0.183		
		Linear acceleration FS = ±8 *g*		0.244		
		Linear acceleration FS = ±16 *g*		0.732		
M_GN	Magnetic sensitivity	Magnetic FS = ±2 gauss		0.08		mgauss/ LSB
		Magnetic FS = ±4 gauss		0.16		
		Magnetic FS = ±8 gauss		0.32		
		Magnetic FS = ±12 gauss		0.48		
G_So	Angular rate sensitivity	Angular rate FS = ±245 dps		8.75		mdps/ digit
		Angular rate FS = ±500 dps		17.50		
		Angular rate FS = ±2000 dps		70		
LA_TCSo	Linear acceleration sensitivity change vs. temperature	From -40 °C to +85 °C		±1.5		%
M_TCSo	Magnetic sensitivity change vs. temperature	From -40 °C to +85 °C		±3		%

Table 3. Sensor characteristics (continued)

Symbol	Parameter	Test conditions	Min.	Typ.(1)	Max.	Unit
G_SoDr	Angular rate sensitivity change vs. temperature	From -40 °C to +85 °C		±2		%
LA_TyOff	Linear acceleration typical zero-g level offset accuracy(3)(4)			±60		mg
G_TyOff	Angular rate typical zero-rate level	FS = 245 dps		±10		dps
		FS = 500 dps		±15		
		FS = 2000 dps		±25		
LA_TCOff	Linear acceleration zero-g level change vs. temperature	Max delta from 25 °C		±0.5		mg/°C
G_TCOff	Zero-rate level change vs. temperature			±0.05		dps/°C
M_EF	Maximum exposed field	No perming effect on zero reading			10000	gauss
M_DF	Magnetic disturbing field	Sensitivity starts to degrade. Automatic S/R pulse restores the sensitivity(5)			20	gauss
LA_ST	Linear acceleration self-test positive difference (6)(7)	±2 *g* range, X, Y, Z-axis AST1:0 = 01 see *Table 74*	60		1700	mg
G_ST	Angular rate self-test output change (8)(9)	FS = 245 dps	20		250	dps
		FS = 500 dps	70		400	
		FS = 2000 dps	150		1000	
Top	Operating temperature range		-40		+85	°C

Week 12

As using the new LCD screen has become more time consuming during the programming and it doesn't ideally meet my specifications I have chosen to look for an alternative screen, this would be more ideal and much more like the current product out there.
This screen is a non-touch TFT screen, this means it would be a graphic screen. Therefore, I would be able to show an artificial horizon which would portray the data from the sensor much like the commercial products out there.
I have spent some time looking at different companies within the UK such as AndersDX and Noritake Itron. Both companies have given me quotes, which was based on my specifications and requirements. I have decided to purchase my screen from Noritake Itron, the screen is a 4.3" inch screen which has its own IC making it easier to use as I should be able to communicate between it and an Arduino board.
The 9-DOF can be plugged onto it or plugged to the Arduino.
Although it is expensive it should be worth it as the other TFT screen I have brought for Arduino in the past have never worked.
I ordered the product so the screen should arrive next week.

The datasheet for TFT screen:

Product Overview

* 4.3 inch TFT	* Single 5VDC Supply	* 3V3/5V Logic	* 6 ADCs
* 480x272 pixels	* ARM9 CPU	* 3 Async UART, I2C, SPI	* 4 PWMs
* 16 million colours	* 64M byte RAM	* RS232, RS485 / RS422	* AC97 Audio Bus
* LED backlighting	* 128M byte NAND	* Up to 32 user I/O	* Real Time Clock / Alarm
	* 8K EEPROM	* USB 2.0 Device	* iDevOS

Applicable Products

Part Number	Touch	RS232	RS485	RS422	AS1, I2C, SPI	CN8	OpSystem
TU480X272C-K611A1N	None	Yes	Yes	Yes	3V3 logic	No	iDevOS
TU480X272C-K611A1NU	None	Yes	Yes	Yes	3V3 logic	Yes	iDevOS
TU480X272C-K611A1NUS	None	Yes	Yes	Yes	5V/3V3 logic	Yes	iDevOS
TU480X272C-K612A1N	None	Yes	No	No	3V3 logic	No	iDevOS
TU480X272C-K612A1NU	None	Yes	No	No	3V3 logic	Yes	iDevOS
TU480X272C-K612A1NUS	None	Yes	No	No	5V/3V3 logic	Yes	iDevOS
TU480X272C-K618A1NU	None	Yes	Yes	No	3V3 logic	Yes	iDevOS

2 – Product Description

This module includes a 480X272 pixel TFT panel mounted on a printed circuit board with low profile construction.
Each pixel has red, green and blue striped elements with 24 bit colour control and 8 bit alpha blending.

The TFT backlight power supply requirements are achieved using a 1Mhz switching boost circuit with constant current control.
Brightness level is controlled by a PWM output from the CPU configured by the application program.

Power and data interfacing is achieved through optional PCB connections.
Many data lines have RC or LC filters to reduce effects due to noise and static discharge.
Pre-regulation of the supply permits a wide operating supply voltage to allow for voltage drop in a long supply cable.

The CPU with ARM9 core, 64Mbyte SDRAM and 128Mbyte NAND flash memory provide control of data processing, font generation, display scanning and peripheral control. 8K bytes of EEPROM provide non-volatile memory for system and user parameter storage. The SD card connector allows loading of the original operating system and other ports can also provide updates depending on the operating system used.

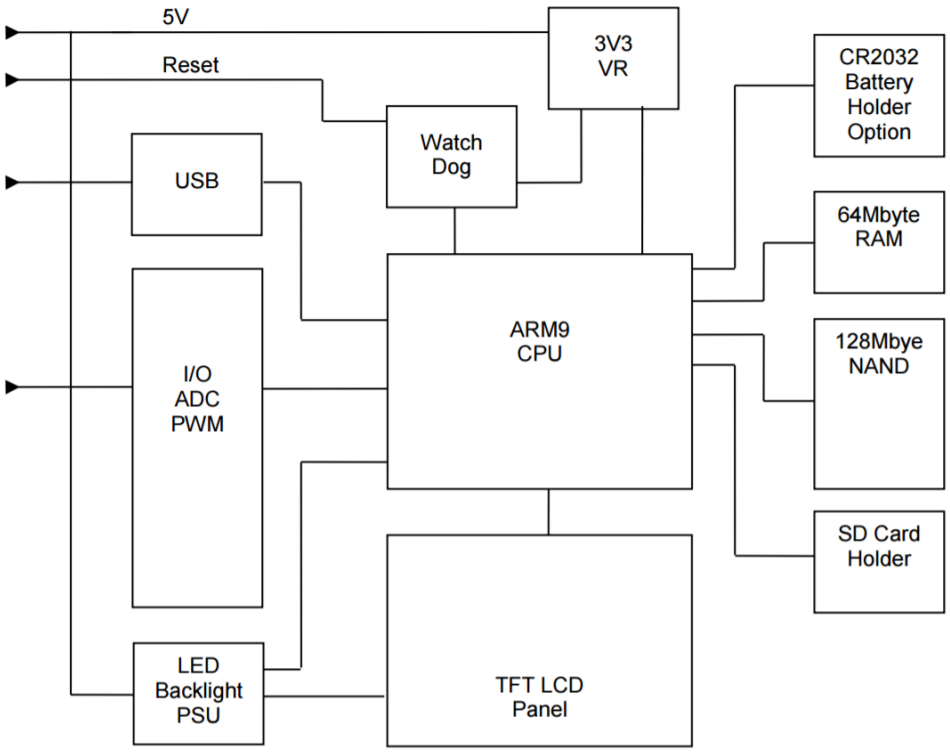
The CPU has an adjustable peripheral and core clock frequency from 80MHz/160MHz to 92MHz/184MHz in 2MHz steps.
This can be adjusted by firmware to reduce noise within specific frequencies critical to certain applications.

An internal independent watchdog chip is reset by the main CPU on a typical 600ms cycle. In the event of a malfunction, the watchdog resets the internal and external 3V3 supply forcing a cold boot for the CPU, memory and any external peripherals connected to the module. The /RESET input on CN3 connects directly to the watchdog circuit and turns off the 3V3 supply when held low.

A CR2032 battery in the optional holder or battery supply connection provide backup power for the Real Time Clock.

This module is designed to be RoHS compliant with sub class A EMI emission and 2kV human body contact model for touch.

3 –Circuit Block Diagram



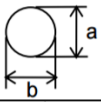
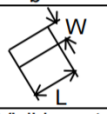
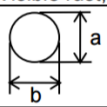
4 – Electrical Characteristics

Section	Parameter	Symbol	Min	Typ	Max	Unit	Condition
5V Input Power Supply	Supply Voltage	$V_{CC}1$	4.5	5.0	5.5	VDC	GND = 0V
	Supply Current	$I_{CC}1$	340	360	390	mA	Vcc1=5V - All pixels ON
		Icc2	120	140	170	mA	Vcc1=5V - LED backlight off
		Icc3	50	60	70	mA	Vcc1=5V – Reset LOW
3V3 Output Power Supply	Supply Voltage	$V_{CC}2$	3.2	3.3	3.4	VDC	GND = 0V
	Supply Current	$I_{CC}2$	-	-	200	mA	$V_{CC}1$=5V
Data Interfaces and I/O Ports	Logic Input Low	V_{IL}	0	-	0.5	VDC	$V_{CC}2$=3V3
	Logic Input High	V_{IH}	2.0	-	Vcc2	VDC	K0-K30, SDHC, ADC
	Logic Output Low	V_{OL}	0	-	0.7	VDC	Maximum sink current 10mA per port
	Logic Output High	V_{OH}	3.0	-	3.4	VDC	Total sink current 70mA
RS232 interface (RX)	Logic Input Low	V_{IL}	-15.0	-	0.6	VDC	$V_{CC}2$=3V3
	Logic Input High	V_{IH}	2.0	-	+15.0	VDC	$V_{CC}2$=3V3
RS232 interface (TX)	Logic Output Low	V_{OL}	-	-3.0	-2.0	VDC	$V_{CC}2$=3V3
	Logic Output High	V_{OH}	4.0	7.0	-	VDC	$V_{CC}2$=3V3
/RESET	Logic Input Low	V_{IL}	0		1.2	VDC	Vcc1=5V
	Logic Input High	V_{IH}	2.2		3.4	VDC	Vcc1=5V
Backup Battery	Sustain Voltage	Vbb	1.5	3V	3.6	VDC	Vcc1=0V – 180mAH ~ 360 days

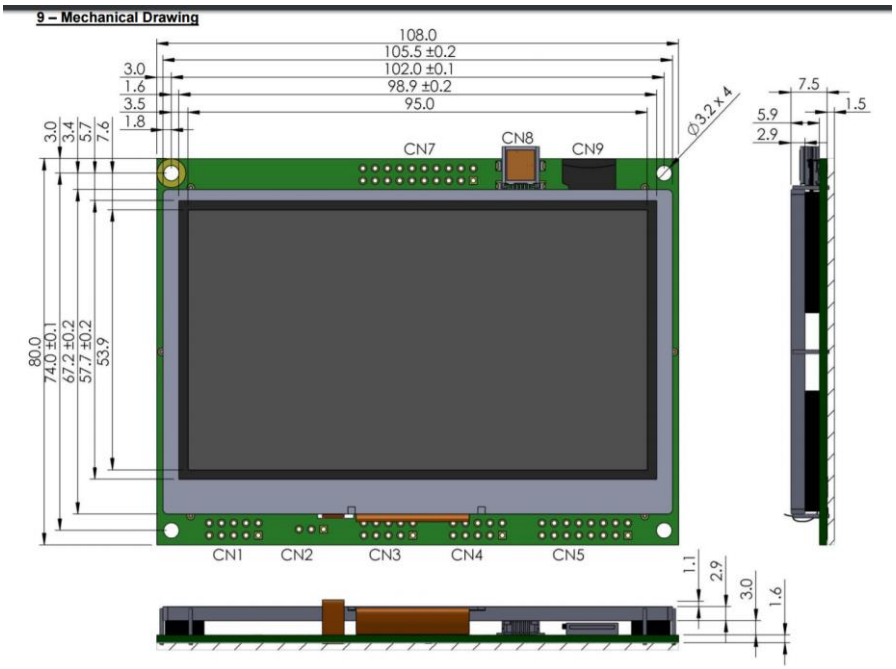
If data signals are applied before the power supply stabilizes, the module CPU may not start correctly until a watchdog timeout.

5 – Optical Characteristics

Visual Parameter		Value					
Display Area (X x Y mm)		95 x 53.9 – 4.3inch diagonal					
Display Format (X x Y)		480 x 272 pixels					
Dot Size/Pitch (X x Y mm)		0.066 x 0.198					
RGB Colours		16,777,216					
Display Type		Transmissive					
Prime Viewing Angle		12 o'clock (colour inversion at 6 o'clock)					
Visual Parameter		**Symbol**	**Min.**	**Typ.**	**Max.**	**Unit**	**Condition**
Contrast Ratio		CR	400	500	-	-	At optimized viewing angle
Color Chromaticity	White	Wx	0.24	0.29	0.34	-	Θ=0° Φ=0°
		Wy	0.26	0.31	0.36	-	Θ=0° Φ=0°
Viewing Angle	Hor.	ΘR	60	70	-	Deg.	CR≥10
		ΘL	60	70	-	Deg.	CR≥10
	Ver.	ΦT	40	50	-	Deg.	CR≥10
		ΦB	60	70	-	Deg.	CR≥10
Brightness		-	350	-	500	cd/m²	Center of Display
LED Backlight Lifetime		-	50,000	-	-	Hours	50% of brightness @ 25°C

Visual Parameter	Defect Criteria	Size (mm)	Acc. Qty
TFT Black / White spot, Foreign material, Pinholes Stain, Particles inside cell. (Minor defect)	 φ= (a + b) /2 2 defects allowed more than 3mm apart Defects outside display area allowed	0.10 < φ ≤ 0.15	2
		0.15 < φ ≤ 0.25	1
		0.25 < φ	0
TFT Black and White line, Scratch, Foreign material (Minor defect)	 W: Width L: Length L<0.25 2 defects allowed more than 3mm apart Defects outside display area allowed	0.03 < W ≤ 0.05	3
		0.05 < W ≤ 0.10	2
		0.1 < W	0
Bezel	Visible rust, distortion, fingerprints or stains	D ≥ 0.1	0
Polarizer – Bubble or Dent (Minor defect)	 φ= (a + b) /2 2 defects allowed more than 3mm apart Defects outside display area allowed	0.20 < φ ≤ 0.30	4
		0.30 < φ	0

9 – Mechanical Drawing



Dimensions are in mm.

11 – Connector Assignment

Pin 1 is a square pad on the PCB. DIL pin 2 is opposite to pin 1 on the other row.

CN1: RS232

Pin	Signal	Pin	Signal
1	NC,Tx+	2	DTR,Rx-
3	TXD	4	CTS
5	RXD	6	RTS
7	DSR,Rx+	8	NC,Tx-
9	GND	10	Vcc1*

*when J47 soldered

CN2: POWER

Pin	Signal
1	Vcc1
2	/PZ*
3	GND

*O/C 20V FET

CN3:AS1 / I2C / SPI + I/O Ports

Pin	Signal	Pin	Signal
1	Vcc1/Vcc2*	2	SCL,SCK,K24
3	AS1 RX,/SS,K25	4	SDA,MOSI,K26
5	GND	6	/IRQ*[I],MISO,K27
7	AS1 TX,/IRQ*[S],K28	8	/RESET
9	MB,K29	10	HB,K30

*selectable via jumper in front, next to CN3
*[I]IRQ for I2C *[S]IRQ for SPI

CN4: ADC / PWM / AUDIO + I/O Ports

Pin	Signal	Pin	Signal
1	ADC1,K16	2	ADC2,K17
3	GND	4	Vcc1,Vcc2*
5	PWM1,K18	6	PWM2,K19
7	ATX,K20	8	ARX,K21
9	ACK,K22	10	AFS,K23

*selectable via jumper in front, next to CN4

CN5: USB / SDHC Expansion

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	DA2	2	DA3	9	GND	10	CD
3	CDA	4	Vcc2	11	GND	12	Vcc1
5	CK	6	GND	13	USB-	14	USB+
7	DA0	8	DA1	15	K31,CNX	16	GND

Enable USB on CN5 or J6 by linking J1 and J5 on back of PCB.

CN6: DBG

Pin	Signal
1	Vcc2
2	GND
3	DRXD
4	DTXD

CN7: I/O Ports

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	Vcc1	2	GND	11	K6	12	K7
3	Vcc2	4	GND	13	K8	14	K9
5	AS2 TX,K0	6	AS2 RX,K1	15	K10	16	K11
7	K2	8	K3	17	K12	18	K13
9	K4	10	K5	19	K14	20	K15

CN12: RS4

Pin	Signal
1	TXDO
2	GND
3	RXDI
4	Vcc1

3v3 logic

CN13: ADC

Pin	Signal
1	GND
2	ADT0
3	ADT3
4	ADT1/2
5	ADT1/2

CN15: USB

Pin	Signal
1	Vcc1
2	USB
3	USB
4	K31,CNX
5	GND

12 - Connector Function

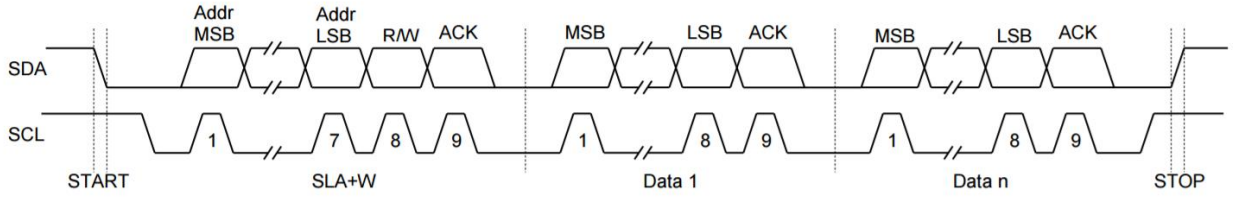
Pin	Signal	Function
CN1-1	NC, Tx+, TxRx+	Not connected in -K612 version. RS422 -K611 versions - transmit positive. RS485 TxRx+
CN1-2	DTR, Rx-	RS232 flow control output, RS422 -K611 versions - receive negative
CN1-3	TXD	RS232 transmit output
CN1-4	CTS	RS232 flow control input CTS - Clear to Send
CN1-5	RXD	RS232 receive input
CN1-6	RTS	RS232 flow control output RTS - Request to Send
CN1-7	DSR, Rx+	RS232 flow control input, RS422 -K611 versions - receive positive
CN1-8	NC, Tx-, TxRx-	Not connected in -K612 versions, RS422 -K611 versions - transmit negative. RS485 TxRx-
CN1-9	GND	0V
CN1-10	Vcc1	5V input / output when J47 is soldered
CN2-1	Vcc1	5V input / output
CN2-2	/PZ	Open drain buzzer output
CN2-3	GND	0V
CN3-1	Vcc1/Vcc2	5V input / output or 3V3 output depending on jumper J27
CN3-2	SCL, SCK, K24	SCL I2C clock, SCK SPI clock, K24 user I/O
CN3-3	AS1 RX, /SS, K25	Asynchronous receive input AS1, Slave Select SPI, K25 user I/O
CN3-4	SDA, MOSI, K26	SDA I2C data, MOSI SPI data, K26 user I/O
CN3-5	GND	0V
CN3-6	/IRQ, MISO, K27	/IRQ I2C interrupt request, MISO SPI data, K27 user I/O
CN3-7	AS1 TX, /IRQ, K28	Asynchronous transmit output AS1, /IRQ SPI, K28 user I/O
CN3-8	/RESET	Master reset - active LOW
CN3-9	MB, K29	Module busy output (AS1), K29 user I/O
CN3-10	HB, K30	Host busy input (AS1), K30 user I/O

Pin	Signal	Function
CN7-1	Vcc1	5V input / output
CN7-2	GND	0V
CN7-3	Vcc2	3V3 output only
CN7-4	GND	0V
CN7-5	AS2 TX, K0	Asynchronous transmit output AS2, K0 user I/O
CN7-6	AS2 RX, K1	Asynchronous receive input AS2, K1 user I/O
CN7-7	K2	K2 User I/O
CN7-8	K3	K3 User I/O
CN7-9	K4	K4 User I/O
CN7-10	K5	K5 User I/O
CN7-11	K6	K6 User I/O
CN7-12	K7, PWM4	K7 User I/O, PWM4 with link J52 and J6 pins 1 and 2 causing backlight level to be fixed at 100%
CN7-13	K8	K8 User I/O
CN7-14	K9, PWM3	K9 User I/O, PWM3
CN7-15	K10	K10 User I/O
CN7-16	K11	K11 User I/O
CN7-17	K12	K12 User I/O
CN7-18	K13	K13 User I/O
CN7-19	K14	K14 User I/O
CN7-20	K15	K15 User I/O

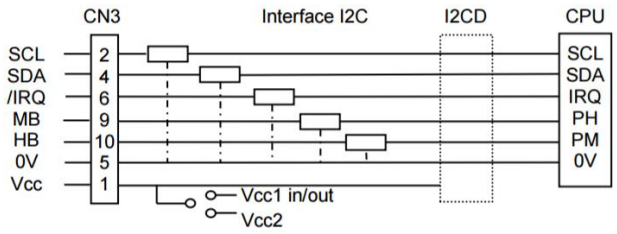
I will be using CN7 and CN3

19 – I2C Communication

The I²C interface operates in slave or master mode as a two wire interface. The maximum speed is 400K bits per second subject to inter-connection. Optional buffers can be fitted for connection to systems where open collector drive is required at 3V3 or 5V.



A START condition is signaled by driving SDA low while SCL is high. A STOP condition is signaled by driving SDA high while SCL is high. After a START condition is detected by the slave followed by 'SLA+W' with the 7 bit address and Read/Write from the master, the slave signals acceptance by raising the ACK bit. When a STOP condition is detected the data received is processed. The /IRQ can be used by a slave device to signal the host that data is available for read. When not active, the port is set to input and when active the port is set to output low.



I2CD
Open collector I2C bus drivers are fitted as suffix option W or S (e.g. -K611TUS) These have pull ups to 5V or 3V3 according to the setting of Vcc

I will also be using I2C.

Link below is for the full datasheet:
http://www.noritake-itron.com/site2017/images/Specs/TFT/TU480X272C-K61XA1NU-FS.pdf

Week 13-14

TFT screen from Noritake Itron
Screen arrived - 2nd December

All parts are acquired for the must do's and should have's.

The screen I originally thought was going to be an easy job, however anything too easy is too good to be true. The screen has its own embedded system which I was going to use to communicate with the Arduino, but through trial and error it is starting to become harder and harder for the two to communicate easily. So, after discussion with my tutor we have come to the conclusion that it would be more practical to run everything off the screen system. Which makes things slightly easier, but it means learning a whole new code from scratch. To which I thought it was C that was uses as the main code on the screen, but I have realized the code is one that has been made from certain aspects of C but is its own code made by the developers of the screen so I must learn a new language quickly and accurately.

Week 15

So far, I have been able to use an example and display the date and time, using the wrong date and time:

Up to this point it hasn't been simple to get to grips with the language as it's meant to be "easier" to use as the company foretold, but it is far from this. The way the code is laid out is that you must setup the USB communication first allowing it to transfer files to the screen's IC then you can do the normal setup like in Arduino for setting integers or such like. Then you can "style" objects for example making a box on the page, next is making variables and telling the program how many bits it needs and the ascii code/hexadecimal code. Next is telling it how to layout the page like a website page using HTML, then you can do your code, then another part for calculations or keys or such like and then telling the screen to show what has been programmed.
So, for something like this to be "simpler" it is quite the opposite.

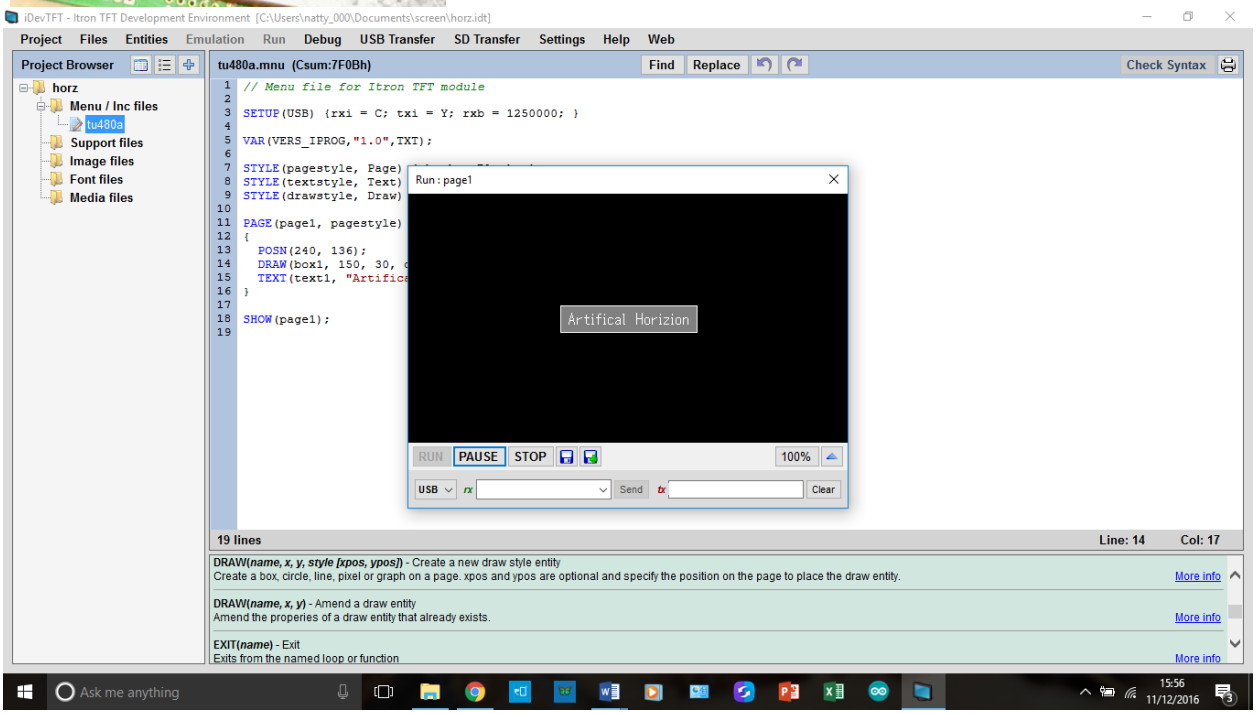
Week 16-17

I have crashed the screen and after discussions with the makers of it they said to do a physically reset between Pins 3 and 4 on CN6 (Port). This didn't work to which they were suggesting sending it back for a reset. But I uploaded a reset program on a SD card and placed it in the SD card module, and by magic it worked. Since then I have managed to display text and the next step is to display an image that doesn't crash the system.

After I worked out what I had gone wrong I started from scratch and started simple. I started by just programming it to show text:

```
SETUP(USB) {rxi = C; txi = Y; rxb = 1250000; }
```

```
LIB (background,"SDHC/background.jpg"); // this inserts a JPEG file
STYLE(pagestyle, Page) { back = black; } // makes the pages
background black
STYLE(textstyle, Text) { font = Ascii16; col = White; } // the
writing will be ASCII 6 which is its style and size and that it will
be white
STYLE(drawstyle, Draw) { type = Box; back = Grey; col = White; width
= 1; }// the size of the box
STYLE(imageStyle, Image) { curRel=TL;}

PAGE(page1, pagestyle)
{
  POSN(240, 136); // the position of the box on the page
  DRAW(box1, 150, 30, drawstyle);
  TEXT(text1, "Artifical Horizion", textstyle);
  IMG(imageBackground,background,imageStyle,0,0);
}
SHOW(page1);
```



Week 18-19

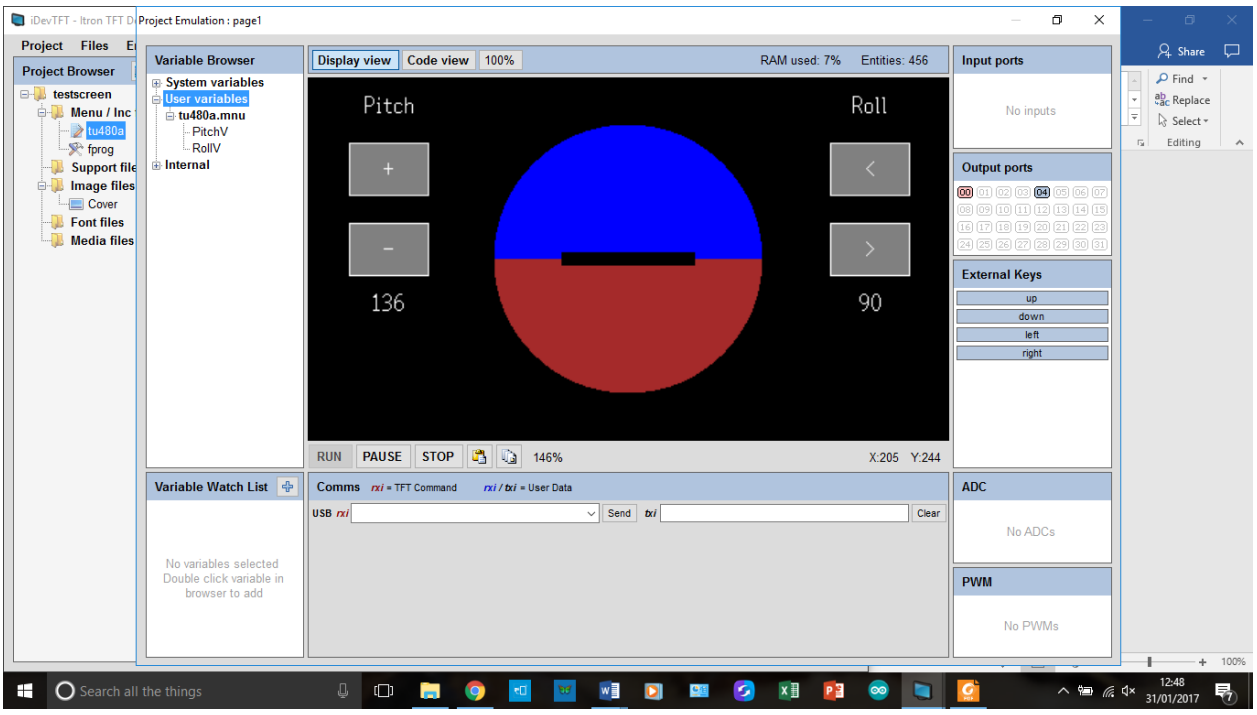
Wall hit

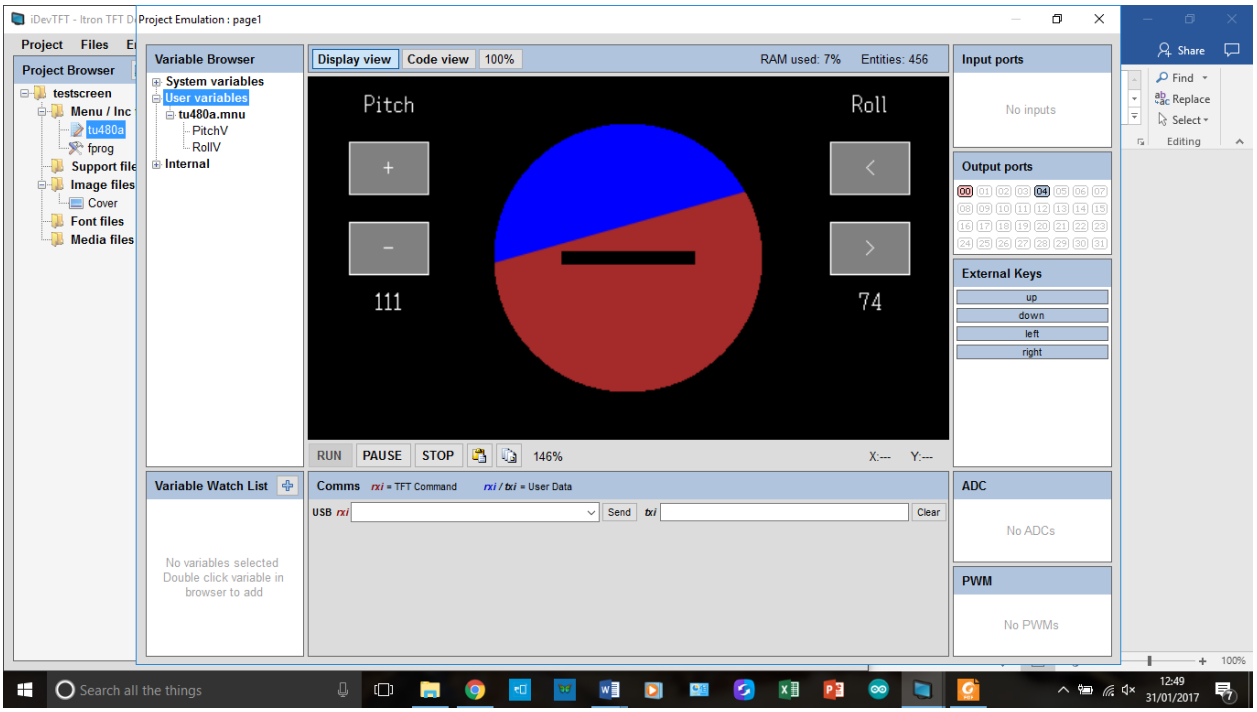
After a month of trying to work out the code, I have hit a wall. I just can't grasp the concept of the code and how it works. Therefore, I have reached out to Noritake Itron for help to which they are happy to do so. They will be helping me understand how to use the program and the screen with my accelerometer by giving me a basic platform to work off. This will help me interface it all and get it running. Once I have this I can continue to improve the quality and the then make a case for it. And if possible the altitude and g meter will be introduced to the project if time allows.
The code will likely be given to me next week.

Week 20-21

Wall overcome slightly

I got some code from the engineers at Noritake Itron, so it now does the roll and pitch graphically. However, it can only be operated by buttons so far on the simulation on the computer. Currently I am working out how to make an external keyboard.





- At first, I tried straight out if statements, but this didn't work.

- So, then I tried loading in the values. But still doesn't work.

- So, I am currently trying out making variable names then assigning the external buttons to the variables then creating an if statement to try and connect the functions.

Just coming to realise that this project (even though I've done 50% of it) is really challenging. And is hitting walls is a constant, and overcoming them takes much longer for what I have planned for. Although I will continue it's very challenging.

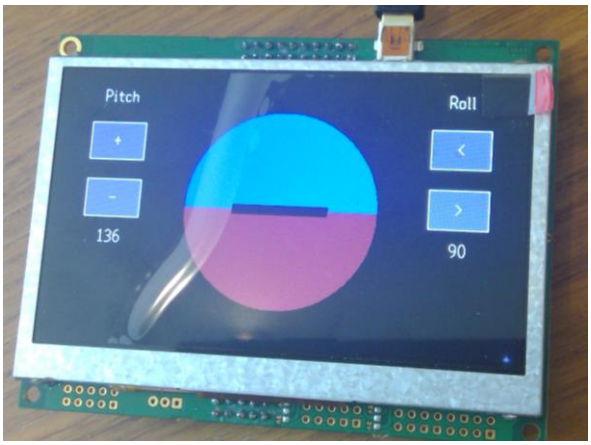
Week 22-23

Screen works!!

Since the last entry I had external keys working on the emulation and the accelerometer working in the emulation but my code had some errors that I wasn't aware of that meant that nothing would work externally. So, the IC was picking reading the data but not showing it on the screen.

After working closely with the tech/support team at Noritake Itron they told me what I had done wrong, and I have learnt from this. They also taught me about the code that I now have and how it works. So, as it stands it works with external keys. Which is cool to see something working after the long hauling months of hard work and constant stress and problems. So, the next step is to get the accelerometer working and try and get a temperature reading and shown on the screen.

After that if I had time I would like to get a portable power source and a case to test it in an aircraft, once the accelerometer works. Then I would like to add a pressure sensor to get altitude and get the yaw working on the accelerometer.



https://drive.google.com/file/d/0ByvPvFxqipN7YXZCVzFmRUhKcFE/view?usp=sharing

Video in the link above showing the external keys working.

```
SETUP(USB) {rxi = C; txi = Y; rxb = 1250000; encode = s;  }

SETUP(keyio) {active=\\00000F; inp=\\00000F; trig=\\00000F; edge=\\00000F;}

// Image files
LIB(Cover,"NAND/Cover.png");

STYLE(pagestyle, Page) { back = blue; }
STYLE(textstyle, Text) { font = Ascii16; col = White; }
```

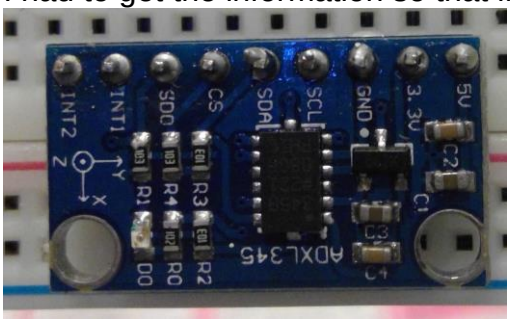
```
STYLE(imagestyle, Image){currel=TL;}

STYLE(stLine3,DRAW){type=vector;col=brown;width=255;curRel=ma;maxX=480;maxY=480;}

VAR(PitchV,136,U16);
VAR(RollV,90,U16);

INT(PUINT,K00,pitchuf);
INT(PDINT,K01,pitchdf);
INT(RLINT,K02,rolluf);
INT(RRINT,K03,rolldf);

PAGE(page1, pagestyle)
{
POSN(240,PitchV); DRAW( box1, 400, 90, 0, stLine3 );
POSN(0,0);IMG(img1,Cover,imagestyle);

POSN(60,20); TEXT(pitcht,"Pitch",textstyle);

POSN(60,168);TEXT(pitchvt,PitchV,textstyle);

POSN(420,20); TEXT(rollt,"Roll",textstyle);

POSN(420,168);TEXT(rollvt,RollV,textstyle);
}

FUNC(pitchuf){CALC(PitchV,PitchV,5,"+");POSN(240,PitchV,box1);TEXT(pitchvt,PitchV);
;}
FUNC(pitchdf){CALC(PitchV,PitchV,5,"-
");POSN(240,PitchV,box1);TEXT(pitchvt,PitchV);;}
FUNC(rolluf){CALC(RollV,RollV,2,"-");DRAW(box1,400,RollV,0);TEXT(rollvt,RollV);;}
FUNC(rolldf){CALC(RollV,RollV,2,"+");DRAW(box1,400,RollV,0);TEXT(rollvt,RollV);;}
SHOW(page1);
```

```
INT(PUINT,K00,pitchuf);
INT(PDINT,K01,pitchdf);
INT(RLINT,K02,rolluf);
INT(RRINT,K03,rolldf);
```

These integers are setting up the names with specific keys, so PUINT is the Pitch Up button so that it is set to KEYIO-00 and that it corresponds with pitchuf.

Week 24-26

So, after getting the keys to work the next task has been to get the accelerometer to work.
So, after a lot of trial and error I brought a cheaper more basic accelerometer. With the code, so far all I have been to do is turn the sensor on. This is because you must program all the functions from scratch whereas with Arduino you can use a library.
So, from the Arduino library's I've been trying to interpret the data and write it in the IdevOS language, which is much harder than it sounds.
Also with the current accelerometer it is hard to tell if the device is on with the new one it has an LED on it that tells you when the interface is on between the screen and accelerometer. As it took at least a week to do. As I had to go through all the libraries for the two accelerometers I had to get the information so that if I needed to I could use the more advance accelerometer.



The week following, I have been trying to get the raw data and convert it into useful data which is also extremely hard, as you must use the date sheet to try and work out the exact hexadecimal code for the I2C address for the sensor to get the raw data for that specific axis reading.
So, far taken two weeks to get this far and for me stay within my planned time will take even longer if I don’t find a solution. Thus, I have decided to go with a simpler method, that took time, thought and discussion with my tutor. Although it was sad not to continue with the more advanced accelerometer it was taking too long.

In this new method, I will just be moving a circle around the screen which will show the axis data. Which therefore shows that it still works and still meets the must do's. From there I could leave it like that and then carry on or I could work out how to turn that data into the pitch and roll and get a proper display going.
Only big issue is the company has just updated the program software and it doesn't allow you to upload the data to the screen.

Week 27

Final week of project

After talking to a friend, who's an ex-commercial helicopter pilot, he pointed out that my project was a hard and enduring one, even though I have got 60% of it if you take into consideration what he has talked about to be it would knock it down to 10% or a little more. This is because he got me to realize that the project needed to work as an RTOS (Real Time Operating System) This is because when it is cloudy and a pilot has the license and equipment to fly through cloud, then they would completely rely on the device. To which my project doesn't update nearly quick enough to meet this standard.
In addition, I have hit a big brick wall with the project, this is because I haven't been able to get the simplest of accelerometer to work, compared to an Arduino its extremely difficult. This is because my interpretations of trying to convert the Arduino libraries for the accelerometer didn't cross over easily, nor did it work.

Even if my I was to continue with the project I would have the various problems that I currently face with the accelerometer. The current problems are:

- The both accelerometers can be turned on, however there seems to be a break down in the connection, as no data seems to be coming through. Even trying to get the raw data is hard enough and that isn't coming through properly. Therefore, no information can be displayed to the screen.

Final Code
The code below is the last piece of code I did which turned on the accelerometer but for some unknown reason didn't communicate with the accelerometer.

```
SETUP(USB) {rxi = C; txi = Y; rxb = 1250000; encode = s;  }

SETUP(i2c)
{
  addr = \\1E;
  end = \\FF;
  active = m;
  proc = all;
  speed = 100;
  encode = sd;
  txi = y;
  rxi = y;
}

VAR(X1, pitchuf, U8);
VAR(Y1, rolluf, U8);
VAR(X0, pitchdf, U8);
VAR(Y0, rolldf, U8);

// Image files
LIB(Cover,"NAND/Cover.png");
```

```
STYLE(pagestyle, Page) { back = blue; }
STYLE(textstyle, Text) { font = Ascii16; col = White; }

STYLE(imagestyle, Image){currel=TL;}

STYLE(stLine3,DRAW){type=vector;col=brown;width=255;curRel=ma;maxX=48
0;maxY=480;}

VAR(PitchV,136,U16);
VAR(RollV,90,U16);

PAGE(page1, pagestyle)
{
POSN(240,PitchV); DRAW( box1, 400, 90, 0, stLine3 );
POSN(0,0);IMG(img1,Cover,imagestyle);

POSN(60,20); TEXT(pitcht,"Pitch",textstyle);

POSN(60,168);TEXT(pitchvt,PitchV,textstyle);

POSN(420,20); TEXT(rollt,"Roll",textstyle);

POSN(420,168);TEXT(rollvt,RollV,textstyle);
}

FUNC(pitchuf){CALC(PitchV,PitchV,5,"+");POSN(240,PitchV,box1);TEXT(pi
tchvt,PitchV);;}
FUNC(pitchdf){CALC(PitchV,PitchV,5,"-
");POSN(240,PitchV,box1);TEXT(pitchvt,PitchV);;}
FUNC(rolluf){CALC(RollV,RollV,2,"-
");DRAW(box1,400,RollV,0);TEXT(rollvt,RollV);;}
FUNC(rolldf){CALC(RollV,RollV,2,"+");DRAW(box1,400,RollV,0);TEXT(roll
vt,RollV);;}

SHOW(page1);
```

Explanation of the code

```
SETUP(USB) {rxi = C; txi = Y; rxb = 1250000; encode = s;  }
```

This is the setup command that allows for the computer and screen to communicate
The USB is a mini-B USB.

rxi = C sets the receiver so that it can interface with the USB device port as a command process source, this parameter is default.

txi = Y this enables the transmit interface of the USB device port

rxb = 1250000 this is a specific size of the receive buffer in bytes

encode = s this the sets the data encode to 8 bit ASCII data, hexadecimals are converted to ASCII

I2C

```
SETUP(i2c)
{
  addr = \\1E;
  end = \\FF;
  active = m;
  proc = all;
  speed = 100;
  encode = sd;
  txi = y;
  rxi = y;
}
```

Addr = \\1E = this is the 7-bit address of the accelerometer in Hexadecimal (slave)
end = \\FF; = this is the 7-bit address for the TFT screen in Hexadecimal (Master)
active = m; = sets the TFT module as the Master
proc = all; = this sets the trigger to receive all characters from the slave
speed = 100; = sets the speed of the transmission in kbs
encode = sd; = sets the data encoder to 8-bit raw data bytes
txi = y; = this enables the transmission of the device to the I2C device
rxi = y; = this enables the receiver of the device to the I2C device

Variables

```
VAR(X1, pitchuf, U8);
VAR(Y1, rolluf, U8);
VAR(X0, pitchdf, U8);
VAR(Y0, rolldf, U8);
```

This part of the code is setting up the variables, the first letter and number, X1, is the corresponding axis and whether it is up or down. The Y axis,1 is for up and 0 is for down or on the X-axis 1 is for right and 0 left.
The name after is its variable name and the U8 is the variable data style, this style is an unsigned 8-bit integer.

```
VAR(PitchV,136,U16);
VAR(RollV,90,U16);
```

This is very much the same to the previous variable however the first bit is also being used as a name, whereas with the X1 and so on wasn't used later in the code. In addition, the data in the middle is the default angle for the variable.
The variable data type is longer as this variable will have the pitch data and the roll data which will have longer/ more complex numbers.

Function

```
FUNC(pitchuf){CALC(PitchV,PitchV,5,"+");POSN(240,PitchV,box1);TEXT(pi
tchvt,PitchV);;}
FUNC(pitchdf){CALC(PitchV,PitchV,5,"-
");POSN(240,PitchV,box1);TEXT(pitchvt,PitchV);;}
```

```
FUNC(rolluf){CALC(RollV,RollV,2,"-
");DRAW(box1,400,RollV,0);TEXT(rollvt,RollV);;}
FUNC(rolldf){CALC(RollV,RollV,2,"+");DRAW(box1,400,RollV,0);TEXT(roll
vt,RollV);;}
```

CALC(Destination Variable, Operand 1, Operand2, "Method");

The first function uses the PitchV to store the final value once it has been used to calculated the current value. So, the value in PitchV is having 5 added to it so that it pitches to an appropriate angle, therefore you can notice the change. Then it is telling the program that PitchV is in a specific position in box1 so that it changes the specific location so others aren't moved. Then it is telling it to update the angle that is being shown in the text box to the new value that has been stored in that variable.
It's the exact same for the other functions however there minus values for the down and left movements.