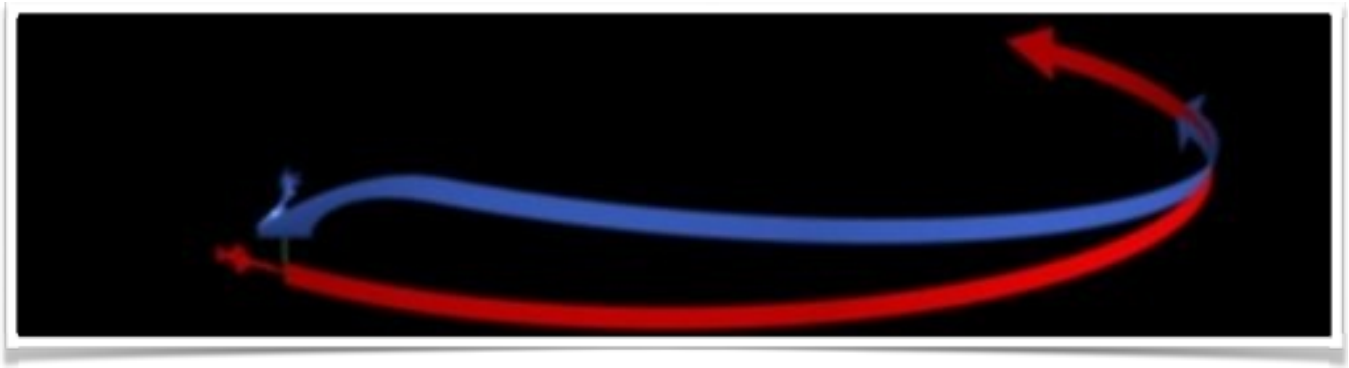


On Resolution

Everybody wants more resolution for their controllers. What good is it? What is it? How much is enough?

We come from the 8 bit resolution controllers, to 10bit controllers of the the Cougar era, then to today's 12bit resolution in the shadow of Warthog. Each improvement gave us real tangible benefits that we can feel. With each newer generation of micro controller units, the resolution of the built-in Analog Digital Converters keep improving their resolution, where is the point that we humans can no longer tell the difference when it comes to joystick controllers?

You want more resolution? How much is enough?

Wait… Warthog is 16bit, they said, "16-bit resolution (65536 x 65536 values)", http://www.thrustmaster.com/en_UK/products/hotas-warthog. Not exactly. Read on.

In this article, I will not give you rigorous arguments or evidences worthy of academic journal nor as evidences in court. Instead, I will give you arguments and reasonings for you to think for yourself.

Digital Resolution

> The resolution of the converter indicates the number of discrete values it can produce over the range of analog values. — http://en.wikipedia.org/wiki/Analog-to-digital_converter

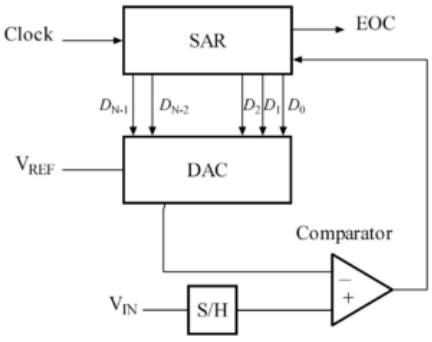
Your average computers do not process analog signals, period. They only process binary digital signals, either 1 or 0. There are many good reasons for computers to do that, I will not go into too much details.

So, in order for your computer, or in our case a MicroController Units (MCUs) to process analog signals, they must be converted to digital first. This is the job for Analog Digital Converters (the other way around is the Digital Analog Converter, DAC, to convert digital signals to analog, the way your MP3 works).

How Some of the ADCs work

There are several methods of constructing Analog Digital Converters, we will only discuss one of those called Successive approximation ADCs.

The successive approximation Analog to digital converter circuit typically consists of four chief subcircuits:



1. A sample and hold circuit to acquire the input voltage (Vin).
2. An analog voltage comparator that compares Vin to the output of the internal DAC and outputs the result of the comparison to the successive approximation register (SAR).
3. A successive approximation register subcircuit designed to supply an approximate digital code of Vin to the internal DAC.
4. An internal reference DAC that, for comparison with VREF, supplies the comparator with an analog voltage equal to the digital code output of the SARin.

http://en.wikipedia.org/wiki/Successive_Approximation_ADC.

The Sample & Hold circuit could be a simple as a capacitor (sample) front ended by a switch (hold). So that when the switch is closed the analog signal is allowed in and charge up the capacitor. Charging a capacitor takes time. A capacitor's voltage does not change instantaneously. When the switch is opened, the charge/voltage stay unchanged so the main ADC circuits can measure the voltage without being disturbed.

If you put on a 16 channel multiplexer (think of it as a 16 to 1 selector connecting one of the 16 inputs to the capacitor) in front of the S/H circuit, then you got a 16 channel ADC that can sample 16 input lines (not all at once, of course).

The important thing about this ADC talk is that it takes time to setup the multiplexer, the S/H, and then the process of successive approximation to run to completion. You should notice from the above Wiki description of the the Successive Approximation, that it needs to tell the DAC to generate one voltage at a time, compare it, then generate another voltage and compare, until it finds the final correct voltage that match the incoming signal. There are much faster methods, but they still take a long time in the view of a fast hundreds MHz CPU core. The time the ADC needs to setup, sample, compare, and generate the digital value is directly related to the sampling rate of an ADC.

That is, when the firmware is running, you have to tell the ADC to start on a channel (or more), wait for the ADC to complete its process, and then read the converted digital value(s) from the ADC. The ADC module on the SAM4S chip is quite a fast one. It is capable of 1000ksps. That is one million sampling cycles. Let's say you tell it to do all 16 channels in one shot. It will probably take a little bit more than 16µs. Let's say you are using a slow CPU running at 24MHz. What are you going to tell the CPU to do during that 24x16 = 384 CPU cycles? Do nothing but wait? Or do you tell it to check whether the ADC is done 384 times? The simplest way to do is the later, and a lot of programmers actually do that.

What if your CPU is running at 120MHz? The CPU cycles to wait for the 16µs is about 1920 cycles!!! You could have done a lot of other things during this 1920 cycles! Now, run the ADC conversion 1000 times a second. Meaning, you waste 192000 cycles per second, just waiting for the ADC, Bubba!

That's called a busy loop. One of the worst things you could do in programming. Although a busy loop is one of the worst things you could do in programming, sometimes, it's the only way. But before

you use it, you better search for all the possible alternatives and see if you can avoid it.

A variation of this dumb busy loop checking the ADC complete status is to do a busy loop checking on time elapsed. You basically write a busy loop to check whether the time tick has passed the specified period of time. So, instead of busy polling the ADC, you find the time ADC needed to complete its conversion process and you do a busy loop waiting for the time to pass. A lot of the Arduino crowd does this….

It's not better, in fact, it's worse. What happens when you estimated the time incorrectly? Or somebody adjusted the overall system clock speed? In other words, the later Arduino crowd approach is timing dependent. It's just not smart!

How Much Resolution Is Good Enough?

That's a long way to get to what we really want to talk about. How much resolution is good enough and what else do we need other than resolution? Let's get to the resolution first.

We know, from experience, the upgrade from 10bit to 12bit makes a huge difference in flight controls, i.e. your joystick, throttle, and rudder. Why is it so? Let's compute.

Let's say, the pivot point of the gimbal to the top of the stick is 10" (about there for the Warthog), and the swing angle of the stick is 25º each side, so we have total of 50º, and the angle of the stick movement from the center is θ.

Then, we know the arc length of the tip is as the following, where θ is in radian unit.

$$\text{length(arc)} = r\theta$$

At the resolution of 10bit. You have maximum 1024 steps. So, we have,

$$50 \div 1024 = 0.0488^\circ .$$

Here, we are assuming that the mechanical linkage would rotate the potentiometer the full 50º when the stick moves 50º.

Plug this into the equation.

$$10” \times 0.0488 \times \pi / 180 = 0.0085” = 0.216mm$$

Now, let’s increase the resolution to 12bit, i.e. 4096. We get this.

$$10” \times (50 \div 4096) \times \pi / 180 = 0.00213” = 0.0541mm$$

Now, let’s ratchet it up again to 14bit. We get this.

$$10” \times (50 \div 16384) \times \pi / 180 = 0.000533” = 0.0135mm$$

At 16bit, we get this.

$$10” \times (50 \div 65536) \times \pi / 180 = 0.000133” = 0.00338mm$$

At 14bit, that’s about 0.5 mil, and 14 micron. If you tell me that you can tell any difference below that resolution by bumping it up to 16bit, I have to either call you a liar or the SuperMan!

How about if you mount it on the floor, as a center stick? Ok. Let’s say that your pivot point to the top of the stick is about 2 feet, i.e. 24”. Then at 14bit resolution, you get this for 12bit, 14bit, and 16bit.

$$24” \times (50 \div 4096) \times \pi / 180 = 0.0051” = 0.12mm$$

$$24” \times (50 \div 16384) \times \pi / 180 = 0.00128” = 0.032mm$$

$$24” \times (50 \div 65536) \times \pi / 180 = 0.00032” = 0.008mm$$

Ahha! There, you can probably tell the difference between 12bit and 14bit. 0.1mm is in the realm of possibility to detect by your hand, but not 0.03mm. 0.1mm may be just that little nudge to get the bead on that bandit. But 0.03mm is like you breath a bit harder. 0.008mm? Your heart in the vein of your hand probably causes more vibration than that!

So, by the above back-of-the-envelope calculations, you know there is no point of increasing the stick resolution over 14bit. Anything beyond that is waste of time. Tell those who say otherwise to bug off, or provide calculations and proofs to dispute the above calculations.

Remember, though, we are talking about stick resolution. We are not talking about other uses of ADCs. But, the most resolution hungry “thing” in your pit is probably the stick, unless you get into motion control of using servo motors to move your entire cockpit. That is a completely different story.

Now, you know why I limit the resolution of the Hempstick to 14bit even though I have enough ADC sampling rate and CPU cycles to increase it to 16bit.

However, one of the big assumptions we have is that your mechanical linkage moves the pot 50º when the stick moves 50º. What if your pot is a 360º range pot, but your only move 50º? That's almost like a 1 to 8 gear reduction. In that case, it's equivalent of losing 3 bits resolution. Also, you cannot drive a pot from 0 to Vcc. If you drive the pot to Vcc, it means 0 Ω and that's a short circuit. Sometime will burn or the MCU may crash/brownout…. bad things will happen.So, either you never drive the full range or you put a little extra current limiting resistor in series of the pot. Whatever it is, you lose a bit of the pot range.

This problem can be solved by using a programmable Hall Effect Sensor like the MLX90316 or MLX90333 that are capable of rail-to-rail radiometric output. What this means is that it outputs voltage from 0 to Vcc at the ratio of the range your program in. For instance, if I program the MLX90316 to between 0 to 120º range, 0º outputs 0V, and 120º outputs Vcc (3.3V or 5V, does not matter), then at the mid-point, it outputs 1/2Vcc. This way, you can drive the Hall Sensor hard to the wall and it will not cause a short. You just have to design the mechanical linkage to drive the Hall Sensor as much to the programmed extremes.

The lessons learned here is that if you design your sensor and mechanical linkage correctly, you don't need anything beyond 14bit. You will only need 16bit if your mechanical design is not ideal so you are forced to waste resolution.

Sampling Rate

Before we get into disputing Warthog's 16bit resolution claim, we must talk about the other important factor in using ADC, the sampling rate.

No matter how high your resolution is, if your sampling rate is horrible, the resolution does not matter. Let's take it to the extreme, 1 second per sample, it's completely useless for a joystick even if you have 16bit resolution. It means you can only change your direction once every second. Dude, one second is ling enough to eat three missiles up your ass!

Clearly, by guessing, 500KHz is not necessary (X gets 500ksps, and Y gets 500ksps, as we have 1000ksps on the SAM4S chip). How much then?

The average human reaction time from seeing something, the signal gets to the brain, the brain reacts to it and command signals get send to the muscles and move the muscles, takes about on average 200ms, http://www.humanbenchmark.com/tests/reactiontime/.

Some elite athletes have been tested at about 80ms. But according to the above URL, they say it's about 130ms. Let's be generous, and give it 100ms.

This 100ms is a straight see-something-and-react. There is also the factor of the "in-between" anticipated actions. That is, you can through practice, anticipate the movements, or tracking, by moving your hands faster or slower in a smooth fluid continuous motion to get in between the 100ms. How much more? I don't know.

A rule of thumb in electronics is to give it 10x factor. That's 10ms. Because we have no idea how humans are capable of anticipating the in-between motions, let's give it another 10x safety factor. We get 1ms. Of course, no self-respecting fighter jocks are going to admit that they are in the 200ms average crowd. I give you 1ms for your ease of mind.

It so happened that the Full-Speed USB has a maximum polling rate of 1000Hz, i.e. 1ms. That's it. 1ms it is. I would say, anything higher than 1ms is waste of time.

The SAM4S chip is capable of High-Speed USB. That is, it's capable of sending USB reports at much higher rates. But, I personally think that above arguments should have convinced you that sub-millisecond report rate is not necessary, just plain old waste of MCU cycles and your host computer cycles.

Oversampling & Decimation

It is possible to increase resolution by oversampling. Meaning, if yo u have excessive sampling rate, you can sample more and use it to increase resolution, http://www.atmel.com/images/doc8003.pdf.

Basically, the formula looks like this.

$f_{samplilng} = 4^n \times f_{nyquist}$

where n = the number of bits to increase.

That basically means, if you have a sampling rate you want to have, $f_{nyquist}$, and you wish to increase the resolution by n bits, you need to sample it 4^n times faster.

In our case, we want 1KHz, because that's what USB report rate we aim for, so that each report we send to the host computer contains the latest ADC measurements.

So, to increase 2 bits to 14bit, we need to run the ADC sampling at 16KHz. But we only have one physical ADC module that is capable of 1000ksps, and 16 channels.

$$16 \times 16K = 256K$$

We are still well under the 1000ksps.

But how about pushing it up to 16bits? Let's do a little different calculation this time. One channel at 1K, requires the following.

$$4^4 \times 1K = 256K$$

$$1000K \div 256K = 3.09026$$

The answer is that we can only run 3 channels at 16bit. 3 x 256K = 768K. That's still well under 1000ksps.

Warthog Is 16bit?

Parameter	Symbol	Test Condition	Min	Typ	Max	Units
Sampling Rate	C_t	Slow mode Fast mode		600 200	1000 330	µs µs

16bit at what sampling rate? 16bit at 1KHz? Really?

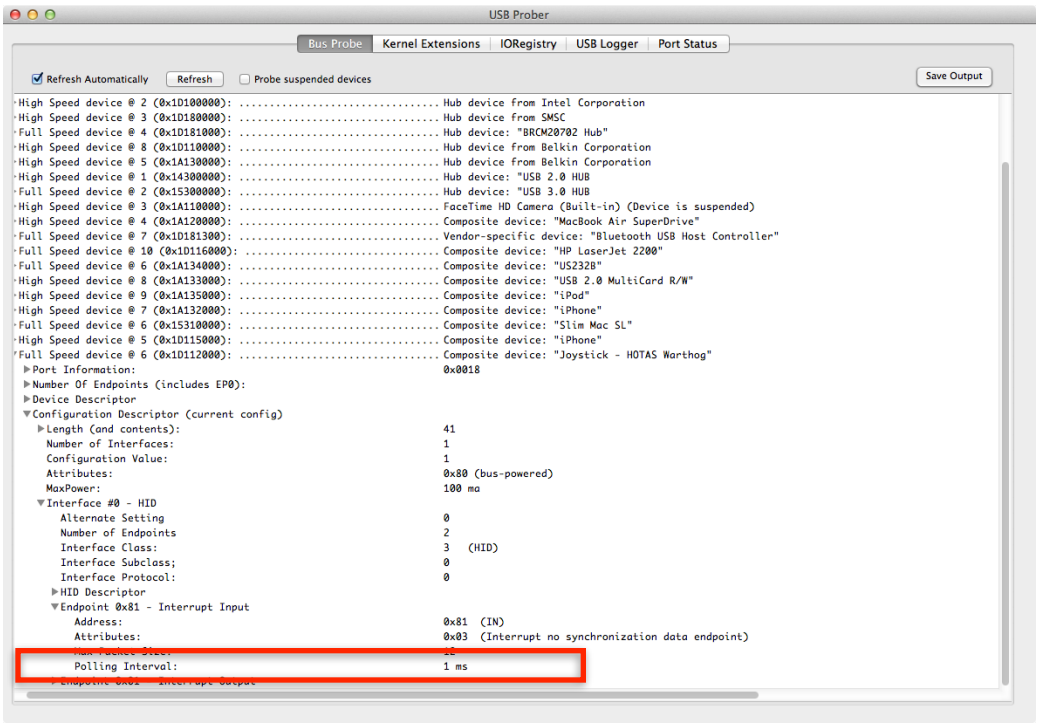
Parameter	Symbol	Test Condition	Min	Typ	Max	Units
ADC Resoluton on the raw signals X, Y, and Z	R_{ADC}	Slow mode Fast mode		15 14		bits bits

The Warthog uses the MLX90333 chip running at digital mode, in a 3-wire SPI configuration.

Here's what the spec. sheet of MLX90333 says.

In fast mode, MLX90333 can do 14bit at max 5000 samples per second (1 / 200 µs). In slow mode, MLX90333 is capable of doing 15bit, but at max 1666 (1 / 600 µs)samples per seconds.

The Warthog's USB report rate is 1000Hz (i.e. 1ms), see the screenshot below.



So, if they use slow mode, 14bit, to bump up 2 bits, at 1KHz report rate, they would need 4^2 x 1KHz = 16KHz, but there is only 5000KHz available. Not possible!

If they use fast mode, 15bit, to bump up 1 bit, at 1KHz report rate, they would need 4 x 1KHz = 4KHz, but there is only at best 1666Hz available. Not possible again!

The spec. sheet also says its Serial Output Resolution is Theoretical — jitter free 16 bits. How?

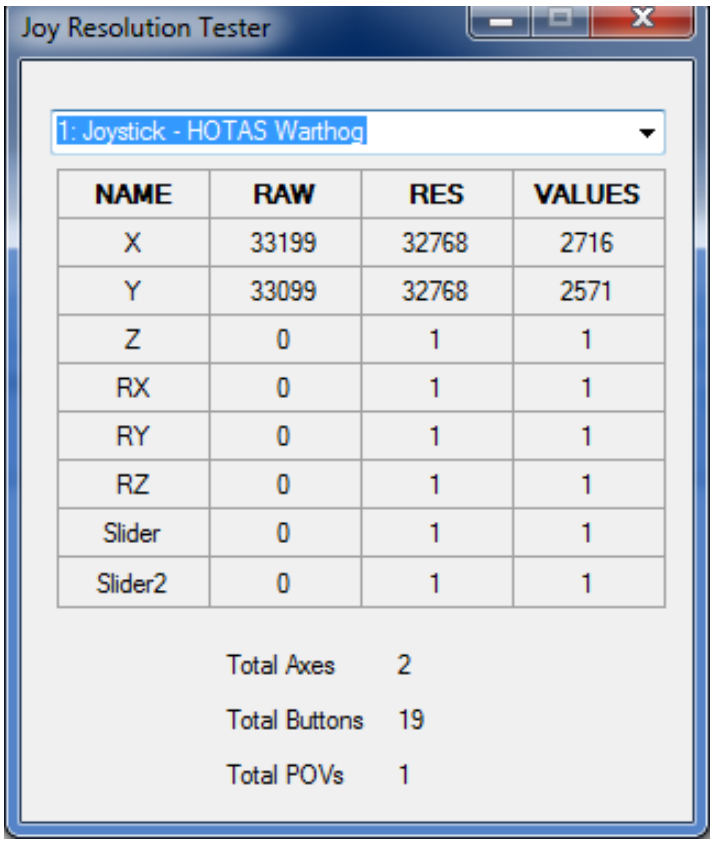
Also note that we are calculating the sampling rate from the typical case, we are not even use the worst case scenario (the max column of values).

The SPI mode digital output of MLX90333 indeed outputs 16bit, 2 bytes. But that does not necessarily mean that all 16bit are meaningful. If you don't believe me, try this.

Open your Device Analyzer in Target. And slowing move the stick and closely watch one of the axis value and see if you can get every value from 0 to 65535. I tried that, I could never get odd numbers except 65535 (or cannot get even numbers).

What does that mean? It means most likely somebody got 15bit values (or even 14bits) and shifted left to 16bit (i.e. multiply it by 2), so the last bit is always 0, hence always even numbers.

Still don't believe me? Ok, go find the JoyResTest.exe program and run it against Warthog and see what you get. Below is what I got.



See? Told you, it's at best 15bits!

I said the same thing on SimHQ when Warthog just came out, nobody could dispute it, yet Thrustmaster continues to claim 16bit resolution and people continue to quote Warthog is 16bit.

I said my piece, you be the judge.