Signal generator log

13/8/20

I’ve been getting into RF stuff a little bit of late, and building up some basic bench gear to mess with it. Relatedly, I would also like to work with analog electronics. One common piece of equipment these hobbies share is a function generator.

Sure, I could buy one. But where is the fun in that?

I found a post on the EEVblog forum

( <https://www.eevblog.com/forum/projects/idea-for-a-tiny-function-generator/msg2507355/#msg2507355> )

to use as inspiration, and started having a crack.

The function generator section used the same basic wave generator IC (DDS chip, AD9834) and amplitude/offset IC (DAC chip, MCP4812) as the forum post to begin with, for ease of prototyping. It is entirely possible that these chips will be replaced as the project matures. In particular, I would like to try using a DAC built in to the main microcontroller to replace the MCP4812.

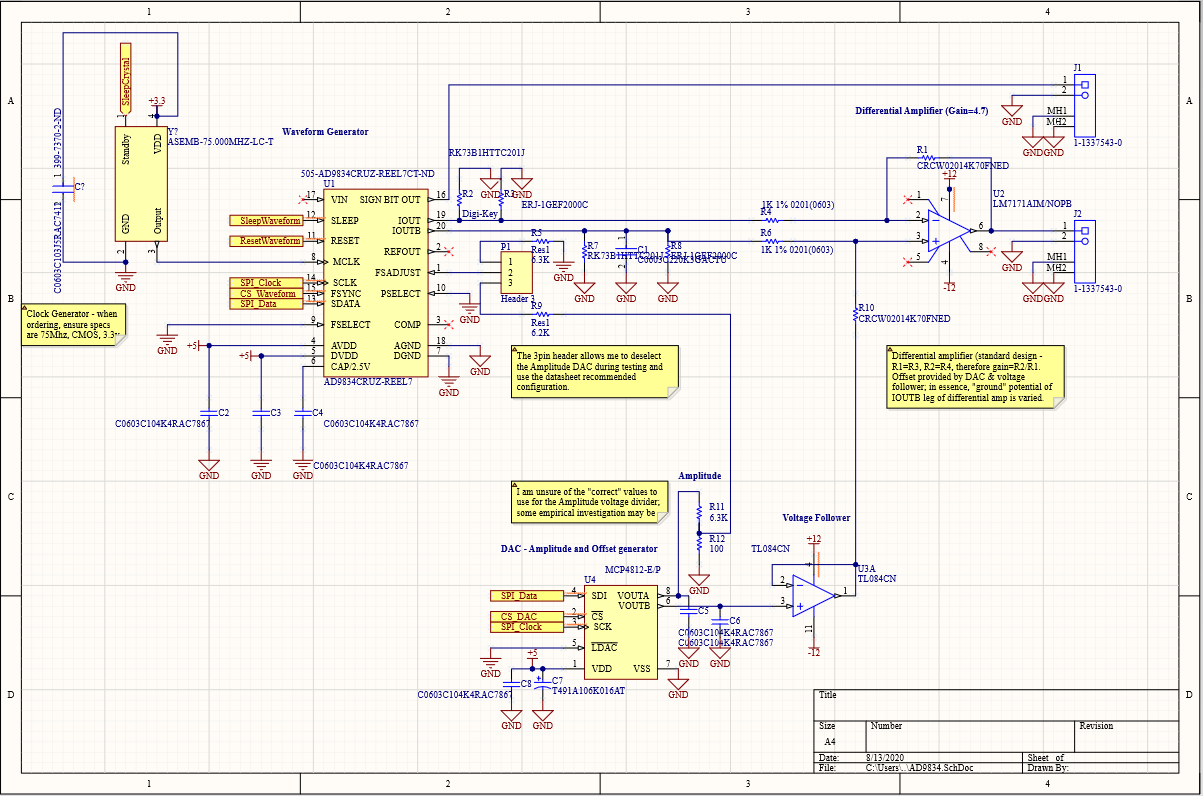


Figure Initial schematic

The majority of the schematic flows from the application circuits in the datasheets. One item I am unsure of is the voltage divider labelled Amplitude (grid C3). FSadjust is used to set the base amplitude the DDS outputs (this amplitude is then embiggened by the Differential Amplifier, grid B4). FSAdjust should have a nominal 6.3K resistor to ground, to set the amplitude of the generated waveform.

From the AD9834 datasheet:

“The full-scale current of the AD9834 is as follows:

IFULLSCALE = 18 × (FSADJUST/RSET)

This circuit uses a specified load of 200 Ω and maximum

full-scale current to achieve voltage output without exceeding

the compliance range of the DAC.

IFULLSCALE = 18 × (1.15/6800 Ω) = 3 mA,

VOUT = 3 mA × 200 Ω = 0.6 V”

14/8/20

In choosing a microcontroller, I started with some admitted bias. In the past I have used a variety of ‘development boards’ (Arduino, Teensy, Esp32) mounted into larger pcbs, but I’ve only designed a few ‘real’ boards where the IC (SiLabs efm32, in those cases) is mounted directly and I provide all the electronic wizardry required. For this project I wanted to try the STMicro ecosystem.

Being of a lazy disposition, I went on Digikey and did some filtered searching. First I filtered for STMicro microcontrollers which were in stock and had datasheets and cad models available. 511 products returned.

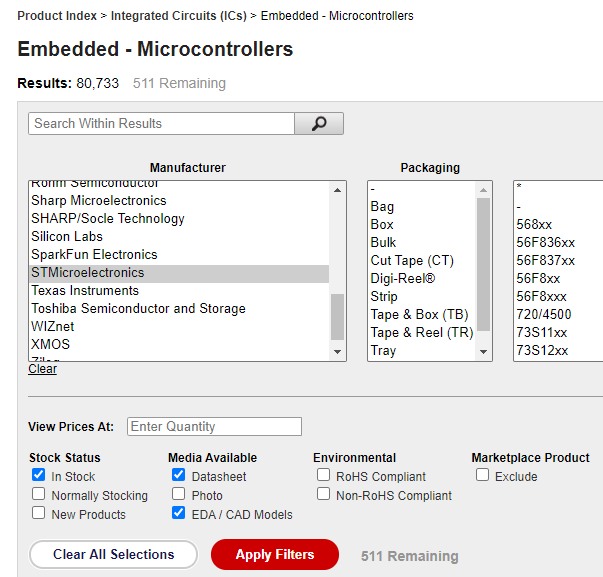


Figure Digikey product filtering

A quick look at how many pins I need, both for general input/output as well as interfacing with my DAC and signal generator led me to select for ICs with at least 20 pins, plus a few more for future expansion – let’s say 30-60 pins. 223 products returned.

32bit, active part status, and a voltage supply range including 3.3v – 178 remaining.

Only packages which I am confident I can hand solder (QFP, so excluding any BGA or QFN etc) – 150 products remaining.

Filtering for speed can be difficult for applications where you’re not entirely certain what you can achieve. But using basic theorems like Nyquist for a rough guide can be helpful; ie, if I expect to be able to generate frequencies up to say 30Mhz, I might want a core frequency of at least twice that if I wish to read back what I have generated for some variety of closed loop. As such, filtering out any ICs with a core frequency lower than 60Mhz, 90 items remain.

I mentioned earlier I might want to use an inbuilt DAC. So filtering for these, 54 remain.

Returning to the packages – I already filtered for QFP, but now let’s minimise the number of pins to solder. A 48 pin LQFP is smaller than a 64 pin LQFP, and sounds a tad easier. 9 results remain.

For connectivity, I need SPI. I2C might be nice to connect to a display of some kind. A UART for debugging too. … This was not helpful, as all of the chips remaining have these connectivity methods (And more that I do not care about)

Program size – This one is more difficult than speed for beginners like myself. A rule of thumb I have used so far is to count the drivers I will need and assume as a worst case that each will require 50kB of ram. In this case, User Interface (input and output); WaveGen; Amplitude/Offset. So ~150kB minimum would be nice. 5 products remaining!

At this point it’s often worth looking at the datasheets for any trip hazards or bonus features. In my particular case, I really cannot be bothered. Lets get arbitrary. I have 3 options for core frequency – 72, 80, and 170Mhz. 80 seems like a nice guy, lets go with that. Et voila! The STM32L433CCT6 is the last one standing.

A quick look at the datasheet doesn’t reveal any major issues; not a huge requirement for supporting components, just a couple of caps. Has 2 DACs, 64K of ram, and most pins are 5v tolerant. Not that I intend to blitz them but it’s a nice touch. Altium has a prebuilt footprint and 3d model for it too, so that’s a win.

Downloading the STM32 ide (“cube”? ok, cool…) is seamless.

Assigning functionality to the pins is also pretty nice; this was annoyingly crap in the SiLabs eclipse ide, as the gui appeared to work just fine… until it had no effect at all and I had to bring the chip up by hand. I had a quick look at the code generated by the cubemx gui and it looks… actually really nice?

Little things like combining output pins from the same port into the same block of code, it’s exactly what I would have done if I was writing it by hand. Good stuff.

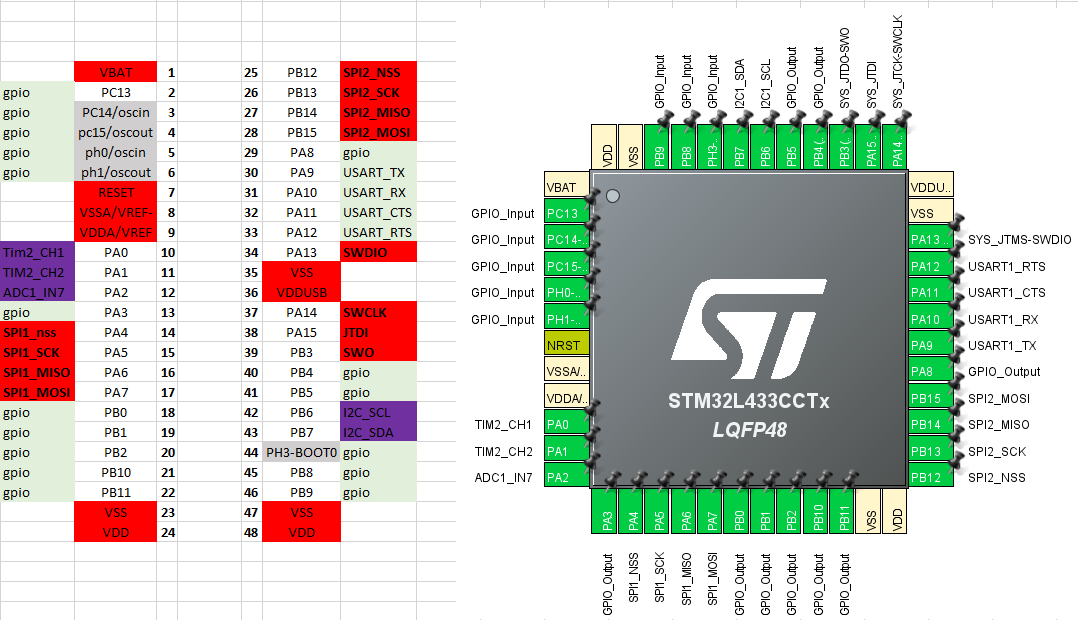


Figure Assigning functions to pins

17/8/20

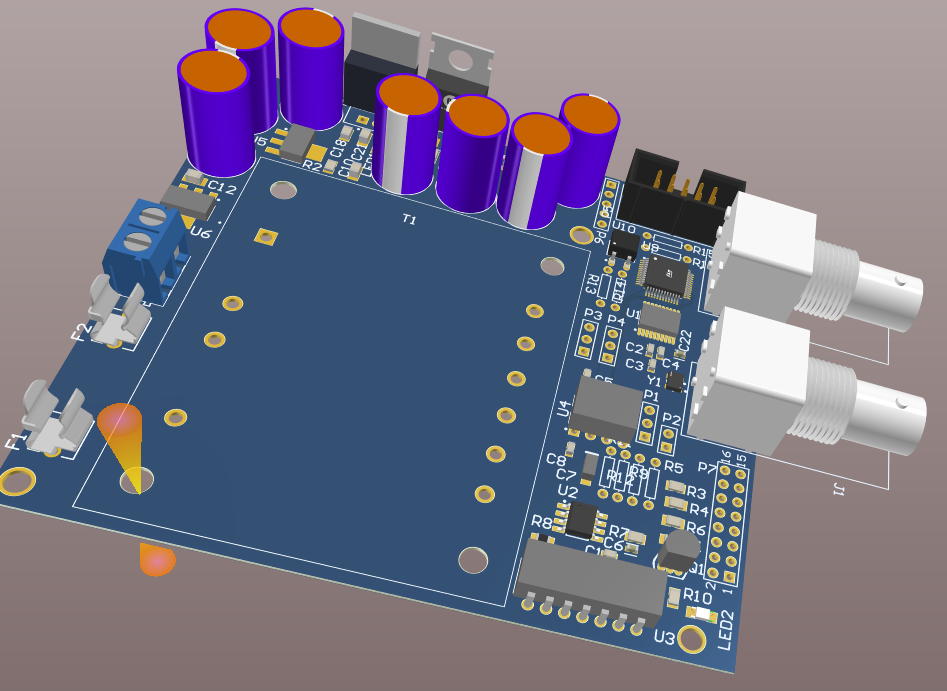


Figure 3d model of pcb

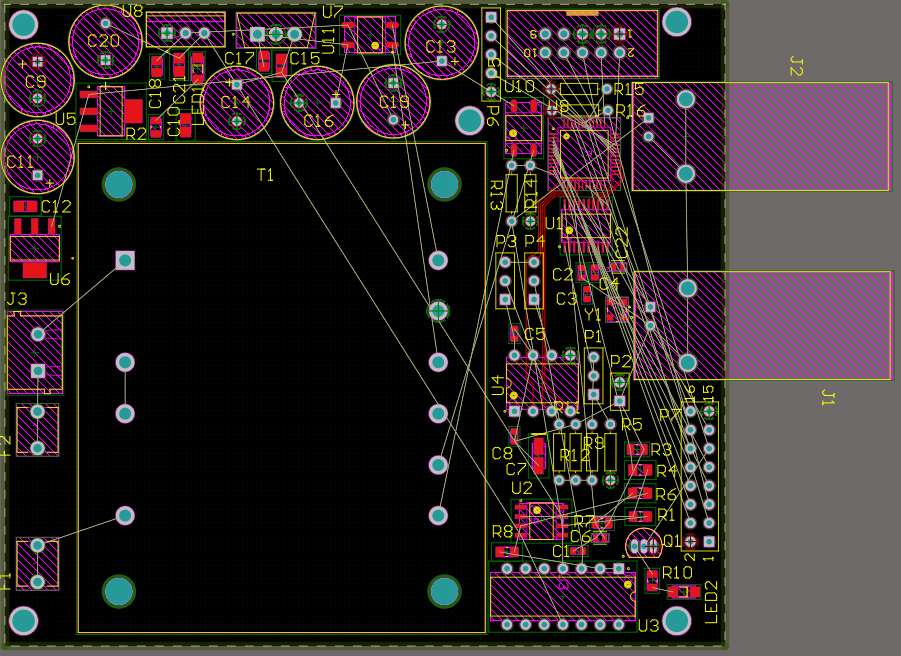


Figure Initial component layout

Moving the components around on the board took some time, and I won’t bore you with multiple screenshots of this process. Suffice to say the board ended up being less than 88x99mm, to make fabrication costs slightly cheaper. I could make the board smaller yet by putting the transformer and other tall through-hole components on the underside.

I have a section devoted to power, and the rest of the board is for the actual signal generation section. In this way I have tried to keep the majority of noise producing components away from the areas where interference might be an issue. The microcontroller and SPI lines are still close to the analog section of necessity, however I am hopeful that I will not need to be continuously sending data back and forth, and may even be able to make use of the mcu sleep state if necessary.

My intent is to try and route SPI and analog particularly (the signal path) traces first and power afterwards.

8/10/20

Finally placed the order at jlcpcb – I’d been waiting for some other projects to be finalised so I could combine shipping.

10/10/20

Sadly I did not take screen captures of the pcb layout process, but the finished product is visible in Figure 6.

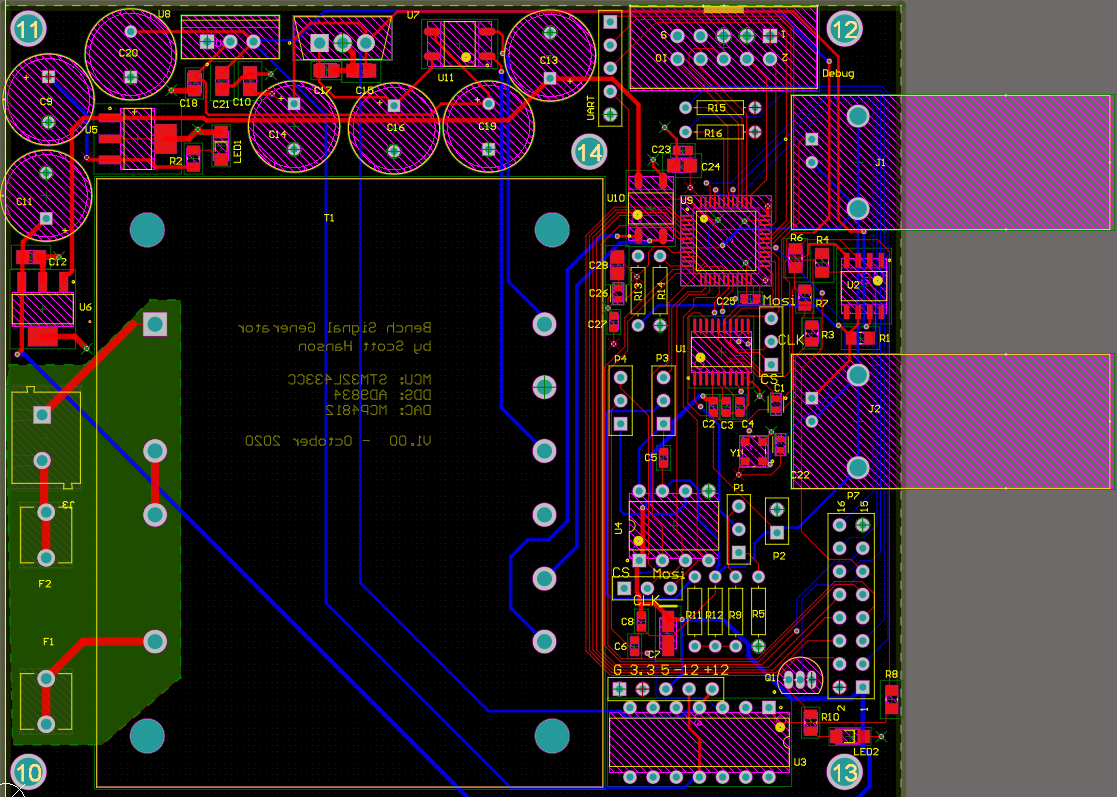


Figure Finished pcb layout

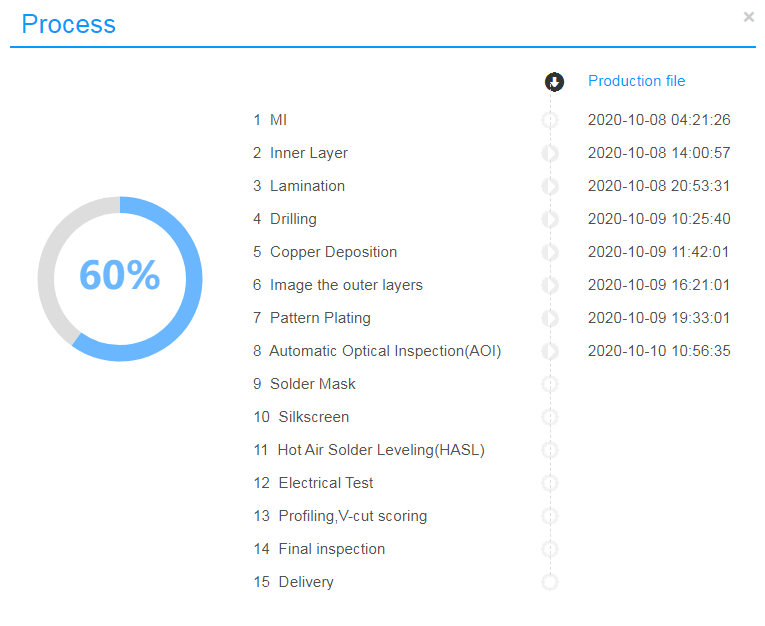


Figure JLCPCB progress

11/10/20

I thought I’d take some time to write out how I intend the device to function.

*I have made allowance for several buttons, a simple display, and an encoder wheel. The buttons are labelled Select, Power, Enter, Phase, Amplitude, Wavetype, and Frequency. The wheel is unlabelled.*

*There is one led on the pcb labelled “DebugLed”.*

*When the unit is powered on, a sine wave of 200 hertz and an amplitude of 1v is output.*

*The display will state these values.*

*Pressing Amplitude will allow the user to change signal amplitude by rotating the encoder wheel. The wheel value shall be displayed on the screen; when the value is satisfactory, the user can save this value by pressing select. I am unsure whether the signal output should change prior to the user pressing select; this could be a separate mode.*

*Similarly, pressing Frequency will allow the user to change the frequency of the signal output, Phase will allow the user to change the phase, and pressing Wavetype will allow the user to select between square, triangle, sine, and sawtooth waves.*

*Power should wake or put to sleep the signal generator and display.*

From this description, some modules jump out at me.

Display is obvious – the program should either display the signal being output, or indicate which value is being altered.

Generate is another- the program should set the amplitude, frequency, phase and wavetype.

Input will need to account for the various pushbuttons, and potentially the encoder wheel also if that functionality is not complex enough for its own module.

…and that is pretty much it at this stage, as the device is intended to be simple. I will not force an arbitrary limit on myself though – if more modules are required, so be it.

My thinking is, every loop the device should :

* check whether it is in a “Change Value” or “Generation” state;
* check for input;
* display;
* request the signal be generated.

The ‘state’ lends itself to a simple state machine, likely implemented in the main loop.

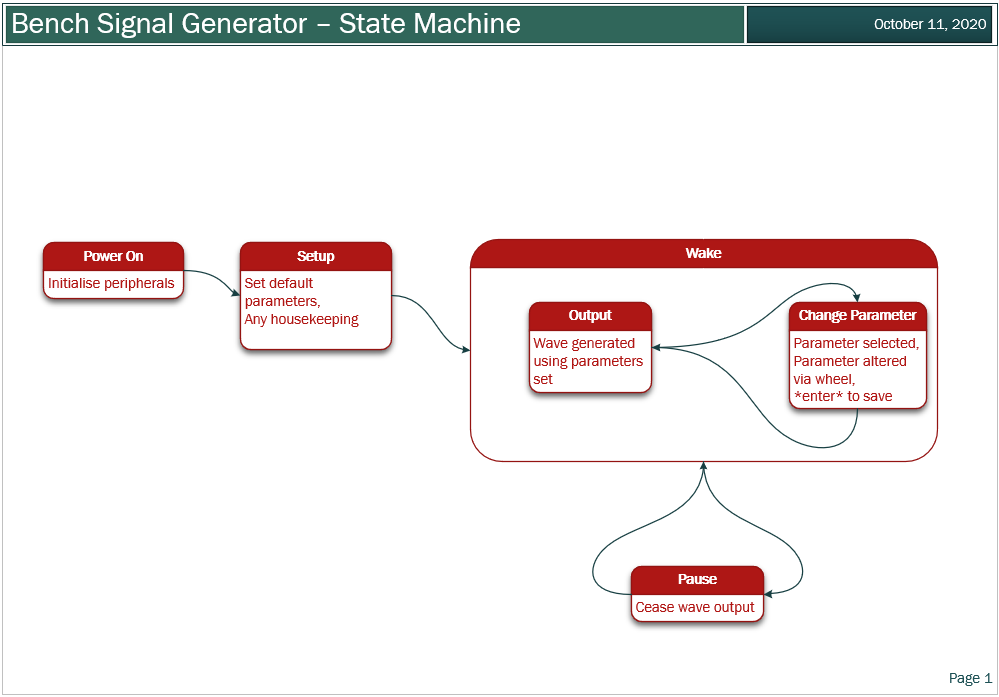


Figure State Machine