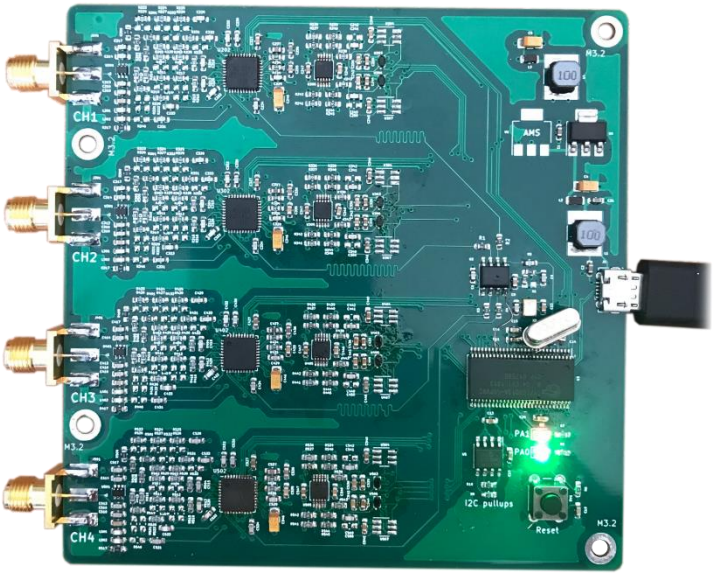


Development of a Four-Channel Narrowband Software-Defined Radio

(a 2020 Monash ECSE FYP)

William Woods
Supervisor: Dr. Emanuele Viterbo

Significant Contributions

The original work in this project covers the following:

- **Selection and demonstration of a novel set of components** which have not previously been demonstrated to work together.
- **Synthesis of a circuit design from multiple references**. For low-level design, these sources were datasheets, application notes and reverse engineering existing devices and code. For high-level architecture, sources were primarily lecture notes on RF frontends.
- **Simulation of subsections of the design** to verify that reference designs would operate as expected, especially given that some reference designs (e.g. the PLL loop filters) were transplanted and combined from devices other than those used in the project.
- **Layout of a 4-Layer printed circuit board** for the project, with consideration of recommended design practices around high-speed digital signals and weak RF being on the same board.
- Hand assembly of surface-mount devices on said PCB.
- **Adaptation and extension of an existing microcontroller firmware** used for high speed data acquisition (an open source USB logic analyser firmware for the same microcontroller as used in this project was the starting point). The adaptations include:
    - Increasing data bus frequency from 24 MHz to 48 MHz
    - Implementing the control signals needed to control the sampling of 8 ADCs simultaneously
    - Implementing a custom USB vendor command to allow the microcontroller to forward messages from the host PC to the tuners via SPI
- **Coding of a basic host-side utility to control the tuners and collect ADC samples via USB**. Note, however, that this did leverage an existing open source driver to determine how to set the tuners' registers for a certain frequency, bandwidth, amplifier gain settings, etc.
- **Troubleshooting** across a fairly complex stack of hardware and software, which has led to the development of several test setups and utilities.

Four-Channel Narrowband SDR Receiver

William Woods
Supervisor: Dr. Emanuele Viterbo

The Motivation Behind the Project

Monash's High Powered Rocketry team (HPR) came to desire a groundstation receive which could listen to a variety of low-rate telemetry transmitters. The team's solution did not involve Software Defined Radio (SDR), however I thought: wouldn't it be cool if it did? We could leverage all the flexibility that comes with Software Defined Radio! The famous low-cost RTL-SDRs might work well enough, but in truth I just wanted to design and build a similarly-spec'd SDR from scratch to learn a bit about what goes on beneath the surface in order to pull tiny electromagnetic signals out of thin air and decode them on your computer. Anyway, I had to find a justification for the undertaking, so this SDR design is intended to do a few neat things the trusty RTL-SDRs cannot.

At the start of 2020, with a vague idea in mind, I approached Dr. Emanuele Viterbo of the ECSE faculty, who graciously agreed to supervise this project. He has watched the project's development and provided valuable guidance (and reminded me to keep the scope focused!) so I owe him many thanks.

Design Specifications

Channels: **4 x I/Q**

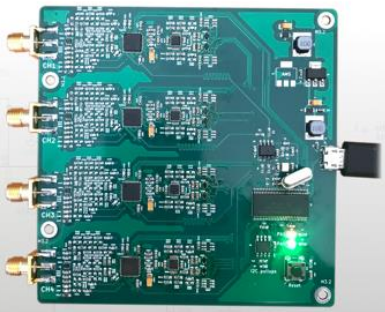
Sampling bandwidth (per channel): **3 MHz**

RF bandwidth: **1.5 MHz**

ADC resolution: **12 bits**.

Frequency range: **~150 kHz to ~2 GHz** (with a bandgap from ~300 MHz to ~470 MHz).

Max power gain: **~70dB**



Applications

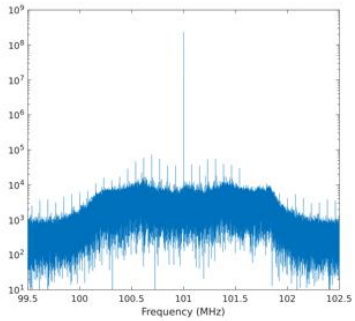
In addition to the usual flexibility of software-defined radio (SDR), this receiver opens up some more intriguing possibilities, such as:

Trunked radio: With the ability to tune into four independent frequencies, it is easily possible to listen to a control channel and, at the same time, follow along with data/voice channels.

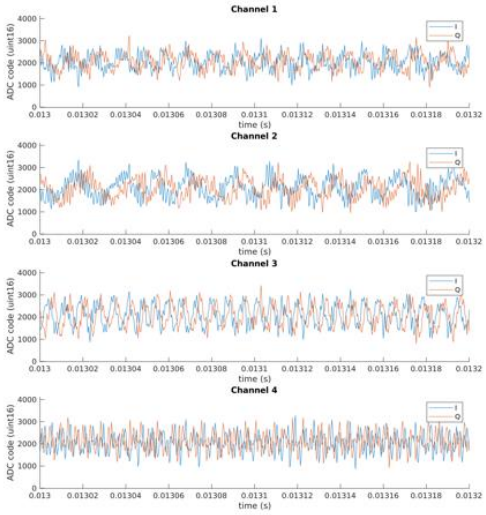
Directional reception: By connecting an antenna array to the SDR and tuning each channel identically, it is possible to detect the phase shift between the antennas of the array because each tuner shares the same reference oscillator. The phase shifts between received signals can be used to pick out signals from different directions.

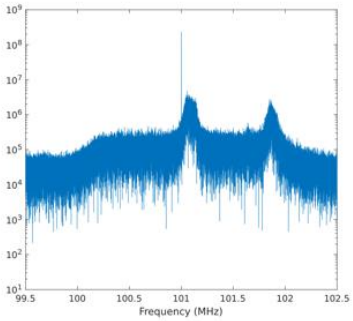
Passive radar: Rather than relying on sending out any RF, it is possible to listen to the way existing high-power transmissions (such as TV and FM radio) are reflected off objects and the environment. By having multiple coherent receive channels available, one can be dedicated to a directional antenna pointed at the source (e.g. TV transmission towers), while the other channels can have highly directional antennas pointed at targets. Information on the doppler shift of reflections from targets can be gained this way, and by correlating the "reference" signal with the reflected signal, a time delay may also be found.

Diversity: When concentrating on receiving a single signal well, it can be beneficial to have multiple antennas, as this allows either the antenna with the best signal strength to be selected, or for some better combination of the received signals to be created.

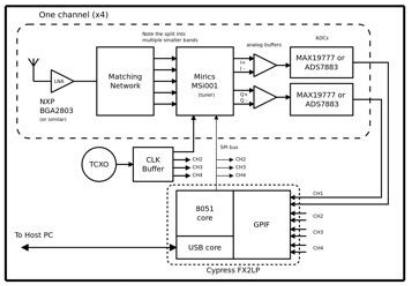


An FFT captured from a channel with no antenna attached, to give an indication of noise floor.





An FFT captured from a channel with an antenna attached, receiving FM broadcasts.



Architecture

Hardware:
Each channel features a 50-Ohm SMA input which feeds into an NXP BGA2303 Low-Noise Amplifier (LNA), and then into a matching and filtering network which splits the wideband RF input into smaller bands for the appropriate inputs of a Mirics MSi001 tuner. The baseband (or low IF) pseudo-differential analogue outputs of the tuner are then buffered and converted to single-ended signals by an op-amp circuit before being sampled by a 12-bit ADC (either a Maxim MAX19777 or TI ADS7883).

On the digital side, a Cypress FX2LP uses its fast General Programmable Interface (GPIF) to pull the samples off the ADCs and pass them along to the host computer via USB2.0 bulk transfers at a datarate of ~300 Mbps. In the other direction, the FX2LP also receives commands from the host computer to configure the tuners, which are interfaced via SPI.

Software:
At present, the firmware written for the FX2LP is functionally complete; it samples from all eight ADCs at 3 Msps, can pass this data back to the host over USB and can pass commands from the driver running on the host back to the tuners.

The user-space driver running on the host (which is written around libusb and libmirisdr) is still in an experimental state, however. Its present functionality allows configuration and tuning of the MSi001 tuners via the command line and allows the ADC samples to be stored to a raw binary file.

Further driver development will involve integration into GNU Radio as a source block (via gr-osmosdr) and potentially other parts of the open source SDR ecosystem.

Results (Pictured)

The four time domain plots above show simultaneous reception (of FM bands) on the four channels, each tuned to a different centre frequency and showing a complex baseband signal output with sensible in-phase and quadrature components.

The leftmost FFT is taken from a single channel, without any antennas connected to the SDR. It gives an indication of the noise floor and shows a roll-off corresponding to the ~1.5 MHz bandwidth option selected on the tuner. It does show a DC offset spike, as DC offset correction is not performed.

The rightmost FFT is also taken from a single channel, but now with a short stub of wire connected as an antenna. The noise floor is noticeably higher, and peaks corresponding to two FM stations are visible within the 1.5 MHz bandwidth. Due to the use of complex sampling, the FFT can cover the full 3 MHz sampling bandwidth and yet have no images present.

The State of the Project so Far...

At present, the project has produced a working PCB with all tuners and ADCs able to operate simultaneously. The ADCs (without any output from the tuner) appear to have about 10 useable bits, with the other two least significant bits lost to noise. The PLLs of the tuners appear able to maintain a stable lock with the present loop filter design (for at least a few frequencies tested so far).

The project also includes a completed firmware to run on the SDR and a host utility able to stream data from the SDR at the rate required for full speed sampling (3M 12-bit samples-per-second from 8 ADCs). The host utility is also able to successfully control the tuners, as demonstrated by successfully receiving multiple FM broadcasts.

Design Tools, Software Toolchains, Libraries and Parent Projects

KiCAD -- used for schematic and PCB design.

LTSpice -- used for designing the op-amp circuit, PLL loop filter and some very basic matching network design.

Cypress GPIF designer -- allowed for graphical design of the state machine and dedicated state machine hardware used to interface the ADCs at high speed.

fx2lib and SDCC -- small device C compiler and FX2 headers/helper functions used for developing the firmware on the FX2LP.

The Sigrok project (particularly fx2lafw) -- the starting point for both the firmware and the userspace drivers.

libusb -- a host-side library which made it easiser to communicate over USB.

libmirisdr -- a handy piece of code which took care of implementing the calculations for register values of the MSi001 in order to configure it and tune to a specific frequency, set filter bandwidths, etc.

Areas of Future Work and Research

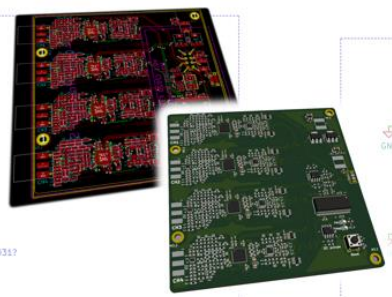
- Integration with GNU Radio and real-time MATLAB processing.

- More extensive testing of reception in the different bands of the tuner.

- Characterisation of the capabilities and performance of the the SDR in a lab under test conditions, and subsequent adjustment of the RF frontend matching/filtering network.

- Implement calibration between the channels and try out some of the ideas listed under *Applications*!

- Version 2 printed circuit board with small errors fixed.



MONASH University
Engineering

github.com/BillyWoods/Narrowband-SDR

Executive Summary

This final year project entailed the development of a software-defined radio (SDR) receiver from low-cost components. SDRs can be thought of as a kind of universal radio receiver, with complex protocol decoding and signal processing offloaded onto a computer connected to the SDR. To increase the availability of multi-channel, phase-coherent receivers available to hobbyists and educators, the device developed in this project uses a selection of low-cost components to build four identical and synchronised channels. That the channels are synchronised ought to open novel possibilities for experiments with applications such as passive doppler radar, direction finding and directional reception.

This project started with research into the architecture of existing SDRs and basic RF theory, which revealed that similar devices do exist, but that in the low-cost end of the market a multi-channel SDR (such as this project) could bring a benefit if its channels could be more easily synchronised and kept in synchronisation. This informed a major choice in the architecture for this project, which was whether to combine four existing low-cost SDRs onto a single board and include a USB hub, as at least one existing design does, thus saving a lot of development effort, or whether to develop a new low-cost multi-channel baseband sampling architecture which would guarantee synchronised sampling, at the least.

The final prototype (and supporting software and firmware) created in this project has been demonstrated to have all four channels functional and simultaneously sampled, without any major issues, so the development of a well-synchronised baseband sampling system has been successful, as has interfacing with the tuners. However, the phase-coherence/synchronisation of the tuners appears, under a brief investigation so far, to need more work.

Acknowledgements

I would like to thank Dr Emanuele Viterbo, of Monash University's ECSE faculty, who has supervised the project since its inception and assisted in developing its scope.

I also wish to acknowledge a large body of open source software that has been drawn upon to provide a starting point for the firmware and host software developed as part of this project. The ability to peer into the inner workings of these projects and cross reference against manufacturer datasheets provided insight into the capabilities of certain hardware and is partly why the architecture of the SDR could be developed before all components had been purchased or a circuit board manufactured.

The open source Sigrok project must be acknowledged for serving as an example of how to use the same MCU at the core of this SDR for high speed data acquisition. Similarly, the Osmocom project, which provides a driver for the tuners used in this SDR, has assisted with interfacing and configuring the tuners used in this project.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ADC	Analog to digital converter
ADF4351	A frequency synthesiser manufactured by Analog Devices
ADS7883	An analog to digital converter manufactured by Texas Instruments.
BGA	Ball grid array
CMOS	Complementary metal oxide semiconductor
EDA	Electronic design automation
EEPROM	Electrically erasable programmable read-only memory
EMI	Electromagnetic interference
EP	USB endpoint
FFT	Fast Fourier transform
FIFO	First-in first-out buffer
FM	Frequency modulation
FPGA	Field programmable gate array
FX2LP	A USB microcontroller chip made by Cypress Semiconductor
FYP	Final year project
GPIF	General purpose interface
GPIO	General purpose input/output
IBIS	Input/output buffer information specification

LDO	Low-dropout Regulator
LED	Light-emitting diode
LNA	(RF) Low-noise Amplifier
MAX19777	An analog to digital converter manufactured by Maxim Integrated.
MCU	Microcontroller unit
MSB	Most significant bit
MSi001	An integrate RF tuner manufactured by Mirics
NB3L553-D	A clock buffer manufactured by ON Semiconductor
PCB	Printed circuit board
PLL	Phase locked loop
PSRR	Power supply ripple rejection
RAM	Random access memory
RF	Radio frequency
SDCC	Small Device C Compiler
SDR	Software-defined radio
SMD	Surface mount device
SPI	Serial peripheral interface
TCXO	Temperature-compensated crystal oscillator

1. Introduction

This report describes the design of a four-channel software-defined radio receiver and the low-level software and firmware written to operate it. It also explains the development of the architecture of the device and why certain design choices have been made. It details some of the pitfalls encountered during development and solutions to these problems. A brief set of results have been gathered through limited testing and are included to demonstrate the outcome of the project: that the design operates successfully with all four channels of the receiver functioning to receive RF signals independently.

Areas for future work are also scoped in this report, and since technical details of the design and novel components of the codebase are discussed, this report should also serve as a helpful companion to anyone exploring the design files and codebase of the project, which have both been made freely available in the hope of further open source development.

The basic approach to development so far, after learning about the necessary components of a radio receiver, was to start with a list of possible components that might fit together and then see if they could be arranged to meet a list of design requirements and constraints. The list of possible configurations was simplified by the choice of an integrated tuner chip, and as the process went on the list of other component choices was whittled down and the details of possible designs refined. Often, a key factor in the choice of components for this project has been availability of working open source software/hardware utilising said components. The result has been that by the time a selection of hardware was chosen and ordered along with PCBs there was a level of confidence that the design would function.

2. Background

The hobbyist SDR scene exploded almost 10 years ago with the discovery that a class of TV receiver dongles (now dubbed RTL-SDRs) all share a chipset which can provide raw baseband I/Q samples, not just a DVB TV stream. It further turns out that the tuners used in these devices can cover a large useful range from ~20 MHz to ~2 GHz. Since a mere $20 could buy a whole lot of radio receiver, an ecosystem of software to go along with these devices emerged, and existing SDR software also got an influx of interest and new development (these days even MATLAB has an interface for the RTL-SDRs!) This same hobby market is where the project detailed in this report is intended to take a spot, by providing more access to the applications that are possible with multiple channels, preferably coherent channels.

To be worthwhile, the requirements of this project were also chosen to exceed the capabilities of this RTL-SDR and place it as a "next step up" option for those already interested in SDR as a hobby or to learn. This project has produced an SDR with a higher channel count, slightly higher sampling bandwidth and greater ADC resolution than the RTL-SDR. This project also synchronises its multiple channels to a single frequency reference, which should enable it to measure phase and frequency differences between the channels and allow for experimentation with exciting applications such as directional reception or passive doppler radar when software processing is applied.

A commercially-available SDR called the KerberosSDR has almost the same features as this project, however it is essentially four RTL-SDRs with a common clock attached to a USB hub, whereas the SDR in this project presents as a single USB device, and baseband ADC samples are guaranteed to be synchronised across all channels. Note, however, that whether this gives any real advantage over the KerberosSDR remains to be ascertained as part of future development, because the RF frontends may still need to be calibrated together.

Other commercially available multi-channel SDRs exist at a higher price point in the market and have capabilities far exceeding this project; often they can also transmit. Presently, examples include the LimeSDR and BladeRF ranges and the SDRPlay RSPduo; at the very high end there is also the National Instruments USRP range.

3. Overview

At least one example of a multi-channel SDR device, the KerberosSDR, avoids the need for much firmware and driver development by including a USB hub on the PCB [1], which allows multiple TV reception dongles to be duplicated on the one PCB. When such a device is plugged into a computer, it is as if four TV dongles have been plugged in, and the host knows nothing about the particular setup and that the four tuners share a clock, etc.

This approach utilising a USB hub avoids firmware development because the baseband/USB bridge chip used in these TV dongles can still be used for a multi-channel solution; the Realtek RTL2832 is used in the KerberosSDR, but the MSi2500 (which is designed as a companion to the MSi001) would be the natural choice if this project was to take the same approach. This approach saves on development time because, just like the tuners, these chips do not generally require or allow any custom firmware to be written, and furthermore, existing drivers can be used on the host as each "channel" simply appears as a new single-channel device.

On the other hand, the approach this project uses does away with these traditional TV baseband sampling/decoding chips which only support one set of I and Q inputs. Instead, after the tuners, 8 ADCs are triggered and read by a single device (an FX2LP) and are guaranteed to be sampled simultaneously. Further, a single data stream is sent back over USB, keeping the simultaneously taken samples in sync. This entailed major work on firmware and drivers, but was done with the intention of investigating whether this could simplify the synchronisation process between channels, as the only source for synchronisation issues with this approach would be different frequency responses in the frontends and differences in the tuners.

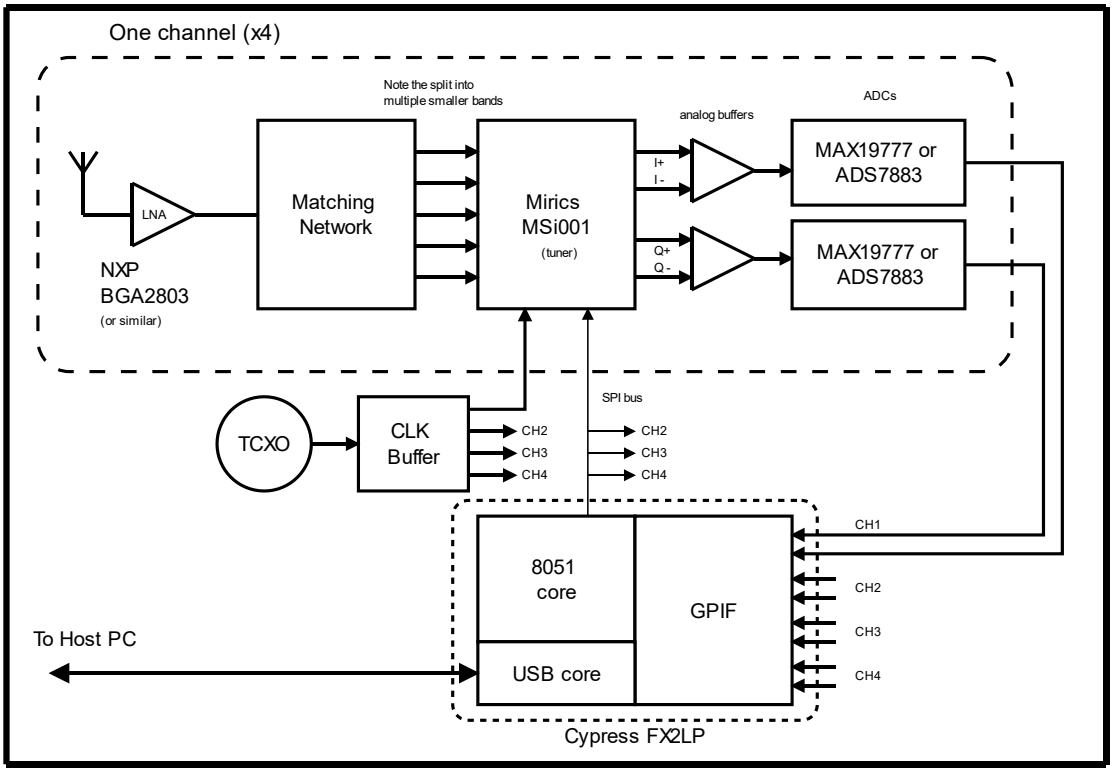


Figure 1: High-level architecture of the SDR hardware implementation. Note that a single channel is draw as all four channels of the device are identical.

3.1. RF Frontend

Directly behind the RF input port of each channel is a wideband low-noise amplifier (LNA), which feeds into a matching network. This LNA is included to provide the option of selecting an LNA with a lower noise figure than the LNAs internal to the MSi001. The LNA is also useful for attempting to offset a lossy matching network.

The matching network is necessary to split the wideband output of the LNA into smaller bands which are accepted by the MSi001; rather than having a single RF input, the MSi001 has multiple inputs for separate bands, as it internally has separate LNAs for each.

The MSi001 is also connected to an SPI bus (shared with all the other tuners) over which its internal registers for things such as calibration, gain and tuning can be set.

Note that although the nearest reference frequency the MSi001 datasheet allows for is 24.576 MHz, a 24 MHz reference has been demonstrated to work instead in many devices, including this project. Being able to use a 24 MHz reference is convenient because the FX2LP also has a 24 MHz clock, and thus part count can be reduced (theoretically), however in the final design the FX2LP was found to have issues accepting an oscillator input, and needed its own crystal instead—more on this in section *4.1.7.2. 24 MHz Reference for the* FX2LP.

3.2. Baseband and ADCs

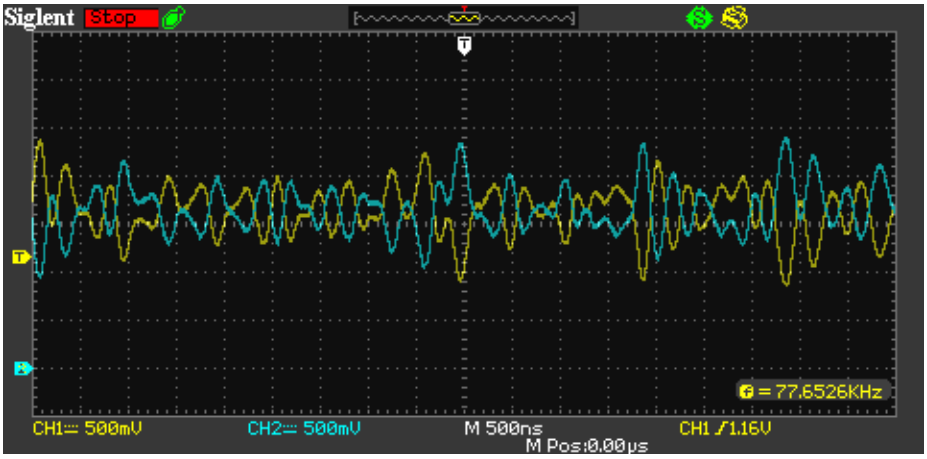


Figure 2: A differential analog output from the MSi001.

The MSi001 outputs two baseband (or low-IF) analog signals: one for the in-phase component and another for the quadrature component. Pictured in Figure 2 is an example of one such differential analog output. Because either half of these differential analog outputs are specified to have a maximum load resistance of 5 kΩ to ground and a maximum load capacitance of 20pF to ground [2, p. 9], op-amps are included to buffer these outputs before being supplied to the ADCs. Also, since the ADCs have a single-ended input, the op-amps serve to convert the differential signals.

The inclusion of the op-amps is mostly precautionary however, as the MAX19777 ADC has an input capacitance of 20pF on its analog input when sampling [3, p. 3], and the single-ended conversion may not be necessary because one half of the differential pair has been observed

to fit nicely within the input range of the ADC (0V up to the 3.3V supply). Whether the op-amp can be forgone has yet to be tested, however.

The ADCs were chosen to have a sample rate of 3 MSPS to match the lowest bandwidth of the MSi001 tuner when operated in zero-IF mode--1.535 MHz. Applying the Nyquist sampling criterion, 3 MSPS was thought to be appropriate. However, the particulars of complex sampling were not properly understood at the time of the choice, and this means a sample rate of 3 MSPS is excessive over the minimum by about two times.

There are two options for the ADC; footprints are included on the board for either the MAX19777 or the ADS7883 for reasons of part availability and concerns about ease of assembly. Both are equivalent in terms of sample rate, resolution, input range and serial interface protocol, however the most pertinent difference is that the ADS7883 is available in a SOT package which lends itself to hand soldering, while the MAX19777 comes only in an 8-pin BGA smaller than a grain of rice, but can still be hand assembled with some patience. The ADS7883 also costs more and has better performance, while the MAX19777 includes an input mux for selecting between two channels (however only one is permanently selected and used in this project).

3.2.1. An Option for Reducing the Number of ADCs

As an aside, for possible future development, if there was a desire to further reduce part count, the second input of the MAX19777 could be explored as a means to perhaps read both I and Q from the tuner with a single ADC, at the cost of halving the sample rate per channel.

This path to reducing ADC count with a halved sample rate of 1.5 MSPS might work quite well with the zero-IF 1.5 MHz double-sided bandwidth option on the MSi001 tuner because the double sided bandwidth is specified at RF, and so when down converted to baseband by the tuner it actually comes out as (f_{tune} - 1.5 MHz/2, f_{tune} + 1.5 MHz/2) of RF spectrum folded onto (0, 1.5 MHz/2) seen by the ADC on any one baseband output. This folding is allowable because complex sampling allows negative and positive frequencies to be separated out: i.e. I (0, 1.5 MHz/2) and Q (0, 1.5 MHz/2) → (-1.5MHz, 1.5MHz/2).

However, with sampling being interleaved, it should be considered that some correction would need to be applied, and the differences between connecting Ch. 1's I and Ch. 1's Q to an interleaved-sampling ADC versus Ch. 1's I and Ch. 2's I should be researched—i.e. interleaving the complex components of a channel versus interleaving two whole channels.

3.3. USB Data Bridge (FX2LP)

Connected to the 8 ADCs is the FX2LP. The GPIF in the FX2LP handles the 48 MHz SPI-like protocol required to read from the ADCs at their full rate of 3 MSPS. More discussion on the particulars of using the parallel bus of the GPIF to read from 8 serial devices can be found in the sections *4.3.2. GPIF* Design and *4.2.1. Re-structuring the USB Packets/GPIF FIFO data into ADC* Samples.

The main purpose of the FX2LP is to act as a bridge to the host PC; it is responsible for all USB communication, responding to requests and identifying the device to the host. By presenting

a USB endpoint and configuring (and triggering) its internal GPIF, the FX2LP can provide ADC samples to the host when polled for them. In the other direction, the FX2LP reads commands sent by the host, and these are used to configure the tuners on the board.

3.4. PC Host Software

Not pictured in Figure 1, but still an important part of the project, is the PC host software. It takes care of communication with the SDR board via USB, sending commands to tune and configure it, while reading, reassembling and recording samples sent from the SDR. It can also provide the data from the SDR in a format helpful for debugging.

Future work on real-time integration of this SDR with GNU Radio or MATLAB could use this piece of software as a starting point.

4. Detailed Discussion

This section contains more information on technical details of key parts of the firmware and software, provides instructions useful for the assembly of the project and building of software, and also includes simulations and calculations which informed the choice of components.

4.1. Circuit Design and PCB Layout

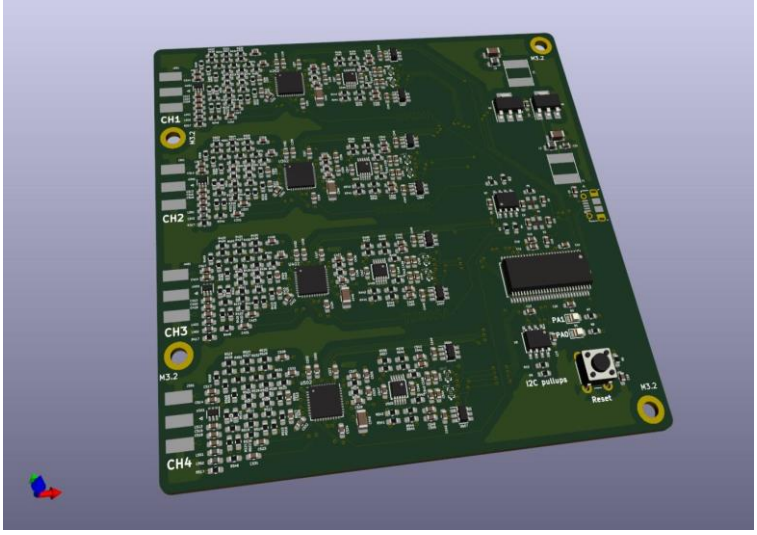


Figure 3: 3D rendering of the final PCB design in KiCad.

4.1.1. PCB Stack-up and Properties

The PCB produced in this project has four layers, with signals being preferentially routed on the top layer, an uninterrupted ground plane on the second layer, a split power plane on the third layer (split into an area for the digital power rail and another for the analog power rail) and a bottom layer for aiding signal routing and fitting some decoupling capacitors.

The PCB is laid out with trace widths intended for the manufacturer JLCPCB's four-layer JLC2313 controlled impedance board profile [4], which was chosen for its short spacing between top and inner topside layers (0.1mm). The choice of this profile is intended to reduce the reach of any fringing fields from transmission lines. One minor downside, however, is that a 50 Ω microstrip transmission line has a calculated width of only 0.147mm, which is narrower than the 0603 components used in the RF frontend matching network--often designs appear to keep trace widths and component widths similar so that signals do not see a change in impedance when travelling into a component pad.

For the calculation of trace widths, although KiCad and many other EDA tools provide transmission line design tools which can work with board properties provided by the manufacturer, the manufacturer's own basic calculator [5] has been relied upon where results differ.

4.1.2. Power Rail Configuration

All power for the device is drawn from USB, as the device consumes less than 500mA. All components on the board are powered with 3.3 V, and a linear regulator is used to step the 5 V rail provided by USB down to 3.3 V. A linear regulator was chosen for simplicity, rather

than efficiency, and to avoid the need to successfully implement a switching converter with adequate filtering and EMI performance.

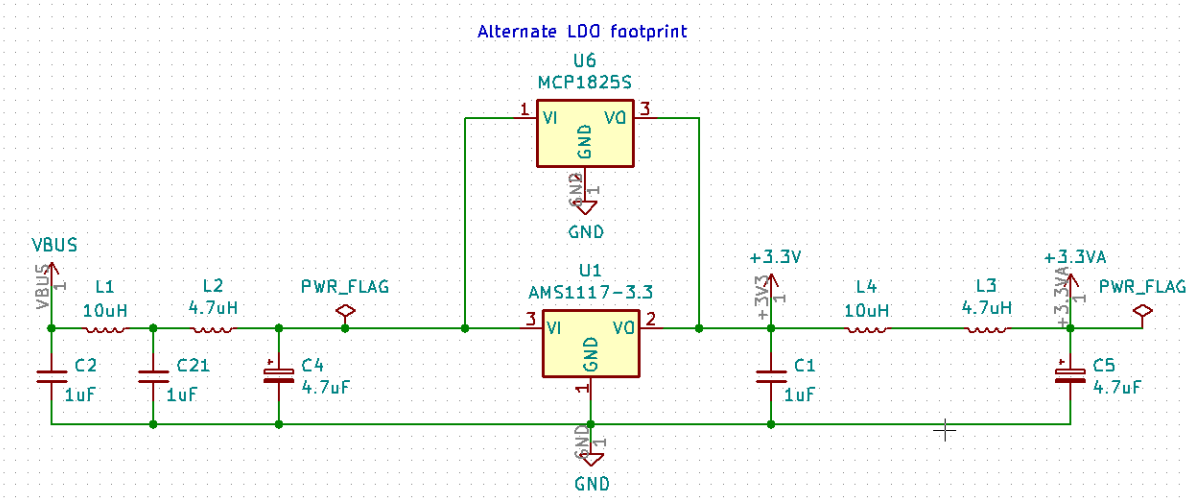


Figure 4: Schematic of power rail configuration and filtering.

The 5 V VBUS rail is passed through an LC lowpass filter before entering a linear regulator (MCP1825S). This was intended to stop high frequency content entering the board or escaping from the board onto the USB cable and radiating, however a ferrite choke on the device end of a USB cable would serve a similar purpose.

The direct 3.3 V output of the linear regulator is used as a digital supply, supplying the FX2LP, the EEPROM, a digital voltage supply pin on the MSi001 tuners and, notably, the oscillator and clock buffer. The oscillator and clock were placed on the digital supply out of concern that despite appropriate decoupling these devices would still pollute their supply rail to a degree that the ADCs should not share the same rail. To create an analog rail for the ADCs, LNAs, op-amps and tuners, the digital rail is filtered with another LC lowpass filter.

The power supply is designed under the assumption that the only source of low frequency ripples is the USB supply, and that the power supply noise generated on the digital part of the SDR board would be high frequencies associated with producing sharp digital signal edges, partly but not completely cleaned up by decoupling capacitors. Under this assumption, the analog rail derived from the digital rail only needs to have high frequency content blocked, while the linear regulator on the USB supply takes care of low frequency ripple for the whole board.

Where possible, circuits in this project are designed to use as few different values of components as possible to reduce BoM count, however the inductors in the power supply needed to be split up into separate values to overcome the fact that for a given SMD component size, larger value inductors tend to have a lower self-resonant frequency and higher resistance. In the LC filters of Figure 4, the smaller value inductors are intended to attenuate higher frequencies which may leak through the larger inductors via parasitic capacitive coupling.

The two LC filters, which are similar, are designed somewhat arbitrarily for a cut-off frequency of about 20 kHz, roughly balancing a desire for as low a cut-off frequency as possible against increasing the part count, or having the size and cost of capacitors and inductors increase too

much. There is, however, a potential gap in this configuration: Figure 5 shows that the PSRR of the linear regulator used in the design is poor between approximately 1 kHz and 100 kHz, so most of the ripple on the USB power rail in this range will pass through the regulator and the LC filters relatively easily. To give more flexibility to solve this issue (should it have eventuated) the LC filter on the input side has additional shunt branches, with the intention being to have spare footprints on the PCB to try out different capacitors to improve filtering.

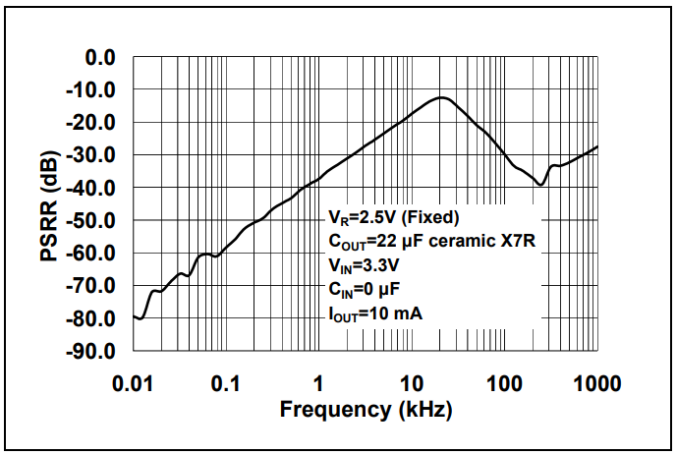


Figure 5: Power supply ripple rejection (PSRR) vs. frequency for the MCP1825S. Reproduced from the device's datasheet [6]. For indicative purposes only, as PSRR here is measured at a different current and regulated voltage.

A better solution than what has been thrown together here is no doubt possible and presents an opportunity for future development and optimisation, however this solution has been adequate so far.

4.1.3. Clock Network Design and Distribution

As all four tuners of the SDR share a clock source, a 1:4 clock buffer (NB3L553-D) is used to supply them. This is necessary because the TCXO used on the board can drive only 15 pF, and each clock input on each tuner and the FX2LP are allowed a capacitance of 5 pF as a rule of thumb, not even accounting for transmission lines to get to these devices. It may be possible to find a TCXO with just enough strength to drive everything, however the clock buffer may still be a preferable solution because it simplifies distribution.

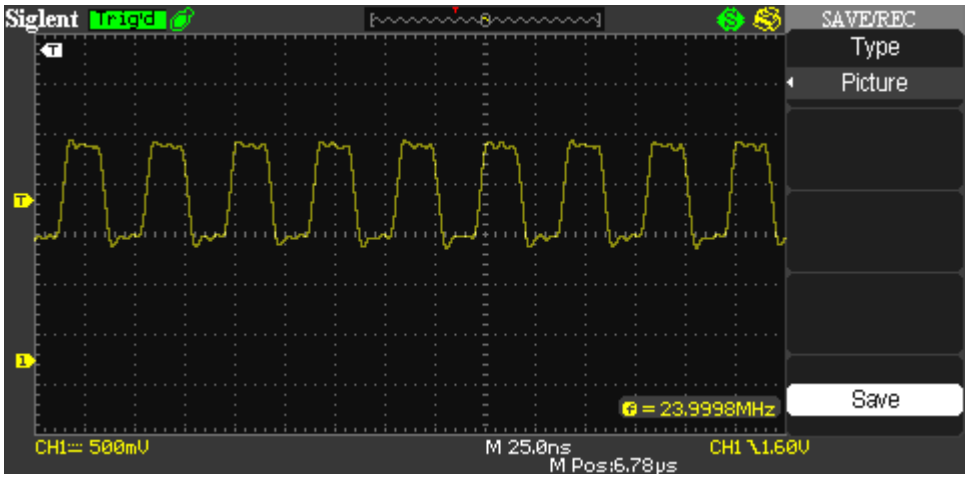


Figure 6: 24 MHz TCXO output, as measured in-circuit, showing a clipped sine wave swinging from approx. 0v to 1V.

By using a clock buffer to split the clock signal rather than a shared clock bus, reflections on the transmission lines become less of a challenge to deal with. Series termination resistors (~25 Ω) are included on the transmission lines at the clock buffer end in anticipation of some reflections coming back from the load (tuner) end, but being successfully absorbed at the clock buffer by allowing the output impedance of the clock buffer (~25 Ω) to be matched to the 50 Ω transmission line. That transmission line impedance was chosen to keep everything uniform with the RF frontend but is somewhat arbitrary.

Figure 7: Output of the clock buffer, showing some potential ringing and a swing from approx. 0v to 3.3v.

These 24 MHz reference clock lines are also length matched because clean, consistent and synchronised edges were understood to be important for synchronising the tuners. For the ADC clocks and nCS lines however, which are driven at 48 MHz by the FX2LP GPIF to control both sampling and data transfer, less care has been taken. In this case transmission lines are used, but no concern is given for impedance matching, splitting and length matching. This is justified by these lines being for a digital bus, which can tolerate more skew and jitter, and because the sample clocks (at baseband) do not need to be as precisely synchronised in absolute terms as the local oscillators within the tuners.

4.1.4. PLL Loop Filter

A correctly designed loop filter is crucial to the operation (or not) of the tuners; it can affect the frequencies that can be locked onto and will also affect the phase noise and frequency stability of the tuner. However, despite requiring an external loop filter, the MSi001 does not provide a reference design or even an indication of the bandwidth such a filter (typically a low-pass filter) should have. Therefore guessing, of the informed type where possible, was unavoidable, but a filter topology with room for modification of the loop filter gain and bandwidth seemed like a safe bet to move the project forward towards manufacture; this way operation could be modified if required.

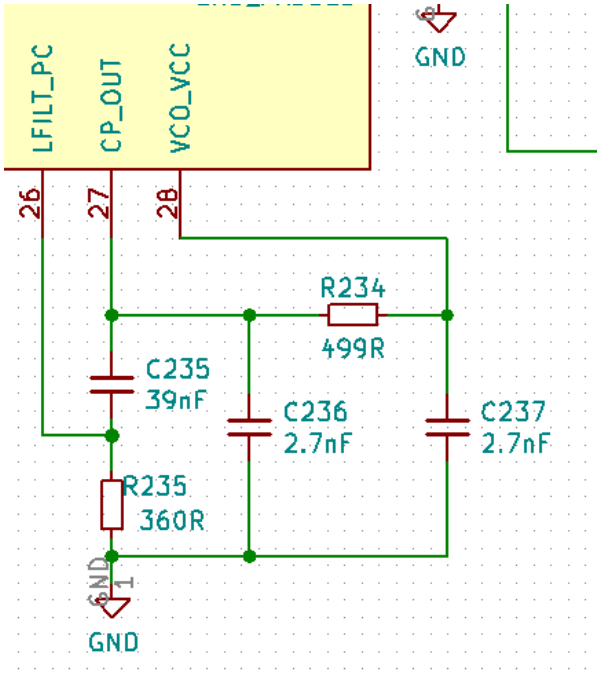


Figure 8: Schematic of the PLL loop filter used for each MSi001 on the board. Based on a reference design in the ADF4351 datasheet [7, p. 25].

The only clues to go by in the datasheet of the MSi001 are the names of the relevant pins and their descriptions: “Synth Loop filter pre-charge,” “Synth Charge Pump OP” and “VCO2 bias decouple” [2, p. 4]. This last one was puzzling, as there appears to be only a single VCO in the MSi001. Further, it was guessed that this was not simply a pin to which an external decoupling capacitor should be connected, rather it was more likely to be the VCO voltage input upon inspection of photos of a disassembled TV dongle PCB using the MSi001 found online [8]. A more enlightening description of this pin would be “VCO voltage input (make sure this is smooth!)” The charge pump output was determined to most likely be the output of the phase detector in the MSi001’s PLL and be a current source (capable of both sourcing and sinking pulses of current) [9]; hence in was apparent the loop filter would need to be between the CP_OUT and VCO_VCC pins.

At this point, a reference design for the Analog Devices ADF4351 frequency synthesiser [7, p. 25] was felt to be an appropriate candidate for a loop filter, because this device covers a comparable range of frequencies to the frequency synthesiser within the MSi001, and will accept comparable reference clock frequencies.

The synth loop filter pre-charge pin was assumed have a function similar to the SW pin on the ADF4351, which is an open-drain-to-ground pin which is activated for “fast-lock” functionality [7, p. 23] and works by bypassing the damping resistor (R235 in Figure 8) to increase the loop bandwidth temporarily when changing synthesiser frequency. The photos of the disassembled MSi001 TV dongle [8] were found to support this hypothesis. It should be noted that the SW pin on the ADF4351 fast-lock reference design [7, p. 23] is connected via a resistor whereas this is omitted in this project’s design (Figure 8) because the MSi001 TV dongle used for reverse engineering [8] appears to omit it.

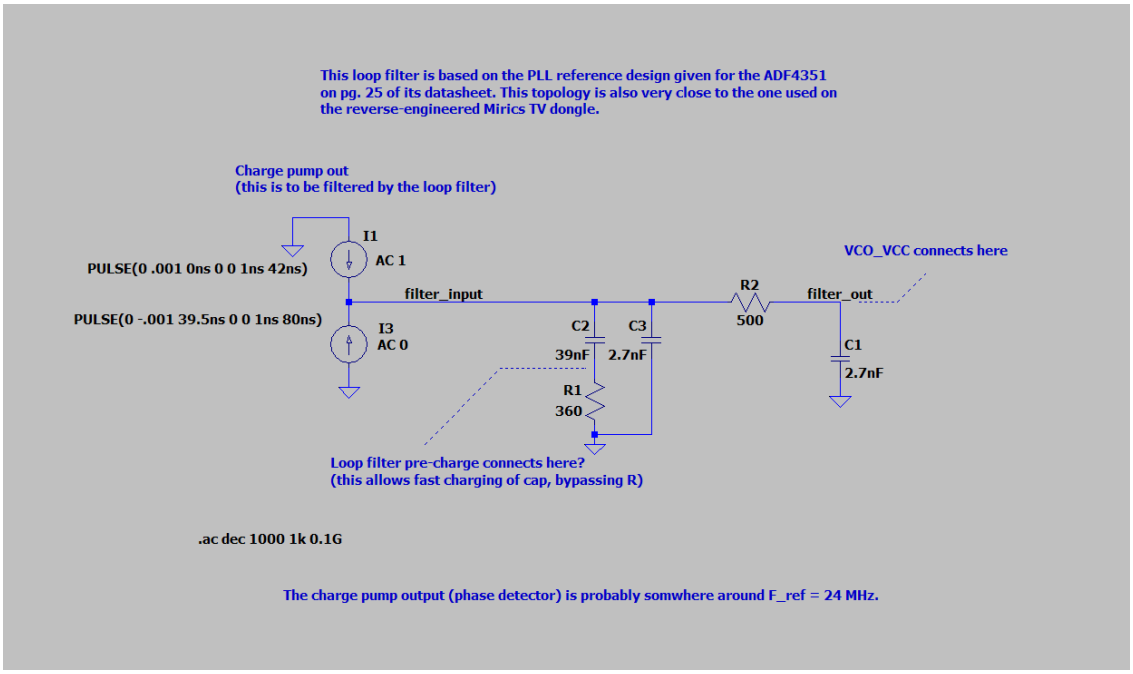


Figure 9: LTSpice circuit used to characterise the PLL loop filter.

Finally, after the topology of the loop filter was chosen, the component values used in the PLL loop filter were changed slightly to reduce the distinct part count, and found in LTSpice to give a bandwidth of about 100 kHz, when measuring from loop filter input voltage to loop filter output voltage (VCO voltage). This may not be the correct way to measure loop filter bandwidth given that the input to the loop filter is a current source rather than a voltage source and the filter is part of a whole closed loop, but this was thought to be a worst-case way to determine bandwidth, and 100 kHz is in the ballpark of typical figures seen in the ADF4351 datasheet (60 kHz) and in line with recommendations to keep loop bandwidth below a tenth of the phase detector output frequency [10, p. 15]. Note the assumption was made that the phase detector output frequency would be somewhere close to the reference frequency (24 MHz).

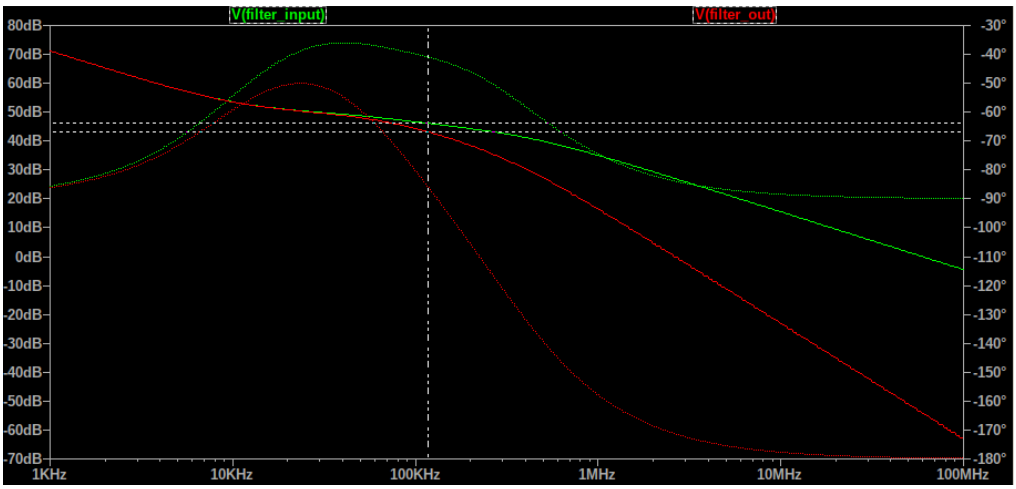


Figure 10: 3dB bandwidth (marked with cursors) of the designed loop filter is ~100 kHz.

4.1.5. Baseband Differential Op-amp Buffer Circuit

As stated in *3.2. Baseband and* ADCs, this buffer circuit was added as a precaution due to the weak driving capabilities of the MSi001's differential outputs.

The circuit in Figure 11 is a differential amplifier, with a feedback resistor chosen to provide a small amount of gain, which was thought to be beneficial as it was not understood at design-time that each half of the differential outputs of the MSi001 can individually go almost from rail-to-rail. In the design in Figure 11, the AC component of both differential inputs see a resistance of 20 KΩ to ground. This was considered adequately high given the MSi001's datasheet specifying a maximum resistive load of 5 KΩ to ground on any analog output.

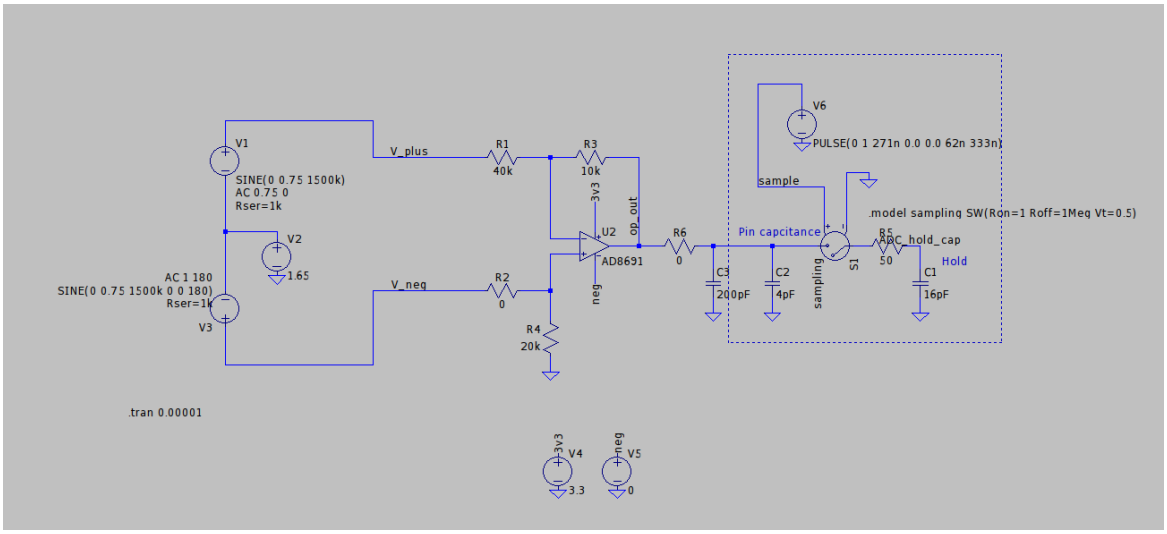


Figure 11: LTSpice model of the baseband differential op-amp buffer circuit and a model of the ADC sample and hold circuitry. The simulation model omits parasitic inductances.

The differential outputs of the MSi001 have been modelled in LTSpice as having a series resistance of 1 KΩ. This value seemed reasonable given the maximum resistive load of the baseband outputs and that an output impedance of anywhere from 20Ω to 2 KΩ is not atypical on the output pins of CMOS devices. However, without IBIS files (or similar) for the MSi001, this is just a guess. It should also be pointed out that the op-amp used in this LTSpice simulation is not the same model as the one used in the design (the op-amp used for simulation appears to have a lower bandwidth), but they are roughly similar.

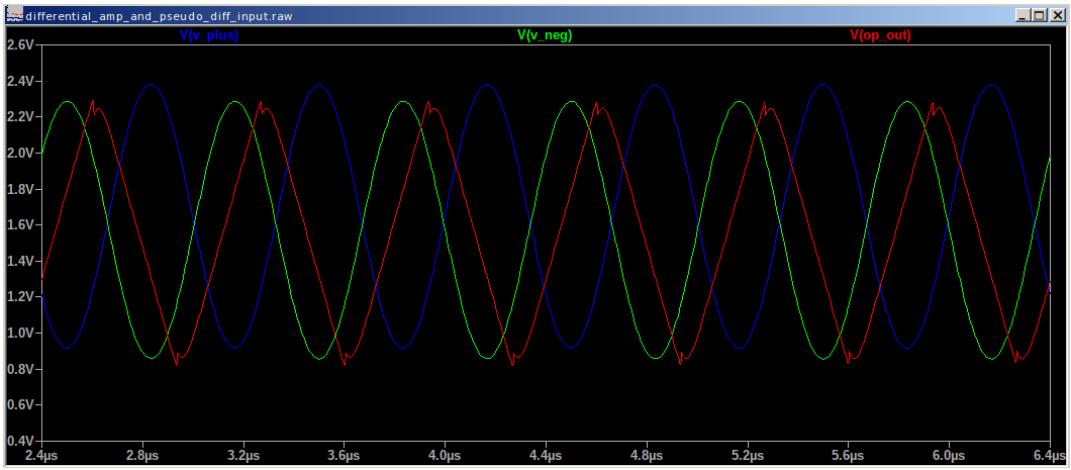


Figure 12: Simulation results of baseband differential op-amp buffer circuit comparing differential input pair and single-ended output with an input at the Nyquist frequency of the ADC (1.5 MHz).

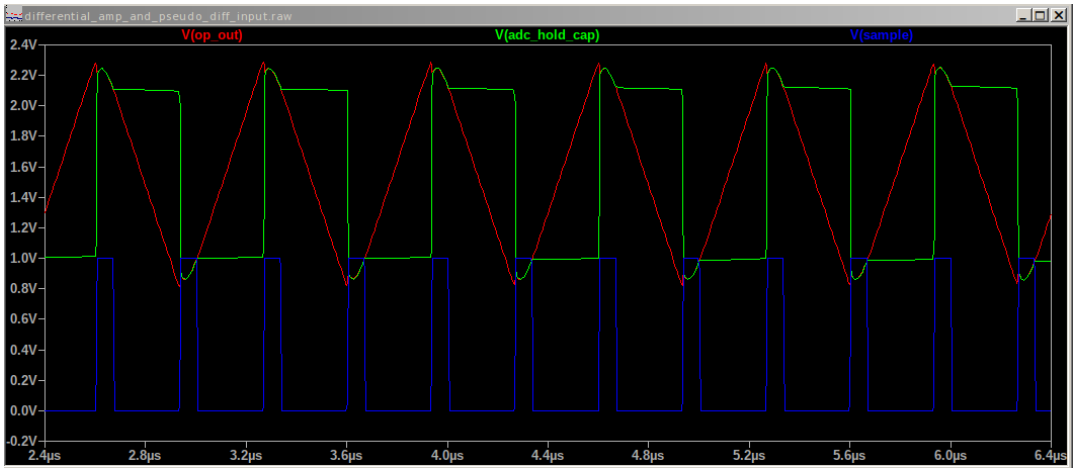


Figure 13: Simulation results of ADC sampling of buffered baseband signal at the Nyquist frequency of the ADC (1.5 MHz).

Of interest in Figure 12 is a slight offset introduced between what should be symmetrical differential inputs (blue and green), due to mismatched loading upon their DC components, but this has not been observed in the actual system, so the MSi001's baseband outputs are likely stronger than modelled. It is also visible in the output of the buffer (red) that when the ADC comes out of holding and begins tracking the analog input there is a sharp transient dip. This dip is being mitigated by the inclusion of the 200pF output following capacitor.

Figure 13 gives an indication that the analog input tracking duration (t_{ACQ} [3]) of the ADCs (the period where blue is high) is long enough for the holding capacitor within the ADC (green) to match the output voltage of the op-amp (red) and further, that the transient in the op-amp output initially caused by sampling has decayed by the time the ADC begins holding.

4.1.6. RF Inputs and Matching Network

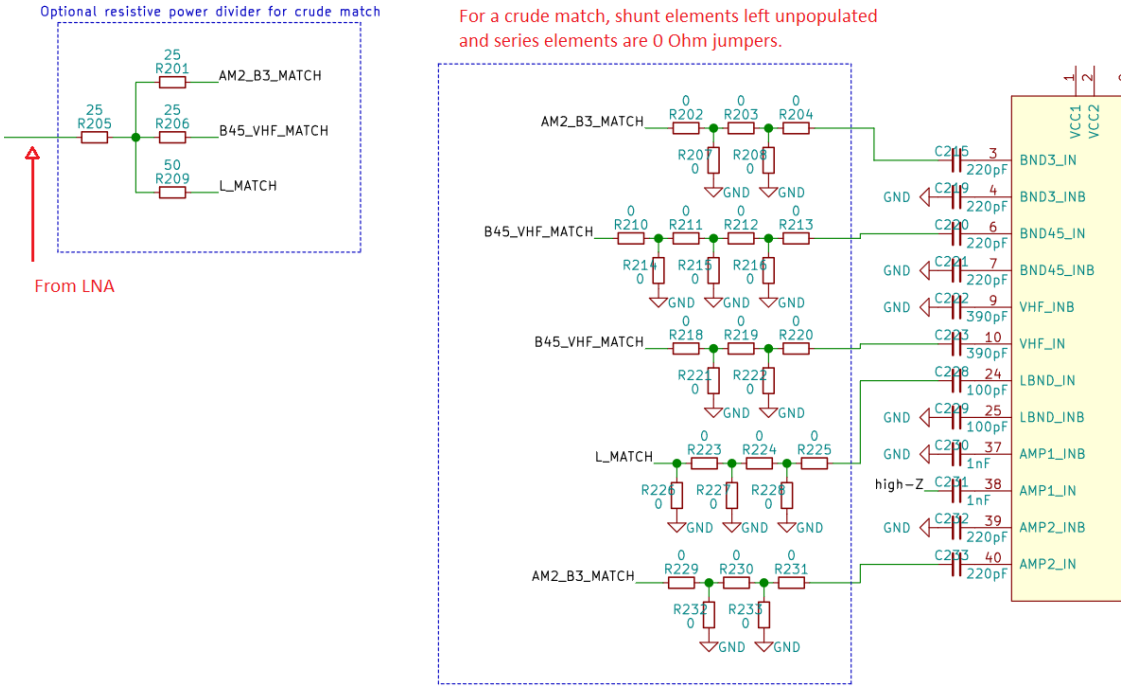


Figure 14: Schematic of the matching network used for each tuner. Note the use of 0 Ω jumpers as placeholders. The resistive divider is in the top left corner.

The MSi001 has multiple RF inputs, each intended to capture a certain band because internally the MSi001 uses a different LNA for each input. These inputs can be used in a single-ended or balanced configuration, and Figure 14 shows that single-ended mode is used in this design because the RF input port of the SDR and the wideband LNA are also single-ended.

Because there is a need to split up the wideband signal from the antenna and LNA, a network must be used for splitting and filtering. Due to underestimating how long it would take to learn and attempt this design task, and yet needing to get hardware manufactured, the circuit design and PCB include unpopulated footprints which a matching network can be built on. The inputs with wider bandwidths have allowances for longer matching networks.

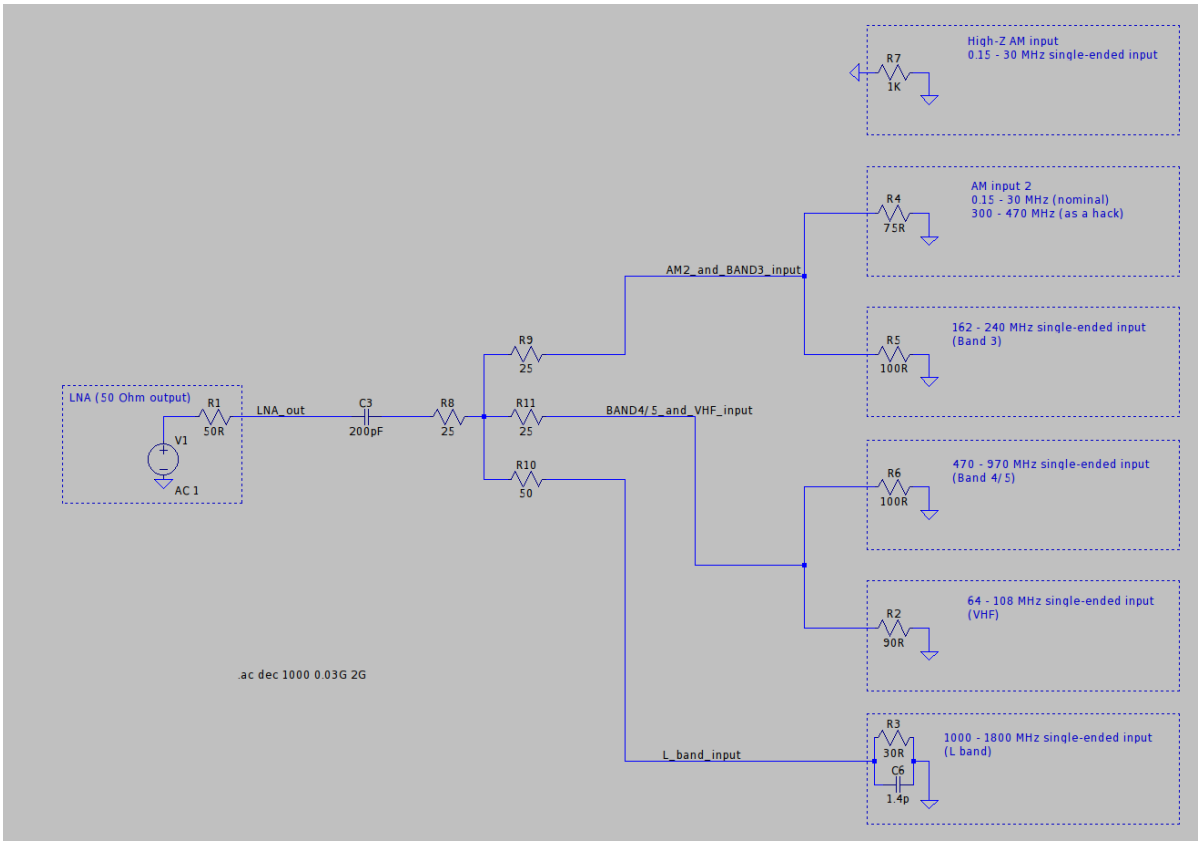


Figure 15: LTSpice simulation setup for a low-effort resistive power divider/matching network. AC coupling capacitors not modelled.

Presently, the through-path of the matching network simply contains jumpers, and a resistive divider is used to ensure that the LNA sees 50 Ω on its output and the amount of power delivered into each port is approximately equal, even if the amount of power loss is dismal. Table 1 shows that the worst performing branch of the divider/input to the MSi001 has an power transfer (S_{21}) of only -13.21 dB. This almost wipes out the gain of the wideband LNA (~20 to 30 dB).

On the upside, the resistive power divider does have very stable characteristics over a wide range of frequencies, with phase and S_{21} barely changing.

Table 1: Power transfer calculations for the resistive power splitter network.

Note: dB current and voltage relative to 1V and 1A

	V (dB)	I (dB)	Power (dB)	Power (dBm)	Power (mW)
Branch 1	-16.21	-53.71	-34.96	-4.96	0.319
Branch 2	-16.21	-56.21	-36.21	-6.21	0.239
Branch 3	-15.89	-55.89	-35.89	-5.89	0.258
Branch 4	-15.89	-54.98	-35.435	-5.435	0.286
Branch 5	-20.74	-50.26	-35.5	-5.5	0.282
Total captured power					1.384
Theoretical max power transfer	-6	-40	-23	7	5.012

Note: this is the max power able to be captured in load (1v 50 Ohm source into 50 Ohm load). With proper matching we'd see this in each branch at its design frequency.

Worst S_21 on any branch	-13.21 dB

4.1.7. Errata on the v0.1 PCB

After manufacture and assembly, the following issues with the circuit design were discovered and fixes subsequently developed.

4.1.7.1. FX2LP "WAKEUP" and "RESERVED" Pins

These pins have been left floating on the v0.1 PCB. For proper booting, it is recommended to connect them per Table 2.

Table 2: Recommended config. of WAKEUP and RESERVED pins on the FX2LP.

Name	Pin Number (on the 56-pin package used in this project)	Connection on PCB v0.1	Recommended Connection
WAKEUP	51	Floating	Tied high via 10kΩ-100kΩ resistor (so the resistor can be removed in case wakeup functionality is to be used in future).
RESERVED	21	Floating	Connected to ground.

4.1.7.2. 24 MHz Reference for the FX2LP

The FX2LP needs a 24 MHz crystal between the XTALIN and XTALOUT pins, with a 12pF load capacitor from XTALIN to ground and another 12 pF load capacitor from XTALOUT to ground. For the v0.1 PCB, this can be added in per Figure 16; be sure to leave C16 unpopulated to apply this fix. According to Cypress' *Guide to a Successful EZ-USB® FX2LP™ Hardware Design* "Two crystals used successfully with the FX2LP are the eCera FX2400026 and the Ecliptek EC-12-24.000M. Load capacitors of 12 pF are required between each crystal pin and ground," [11, p. 21].

a) b)

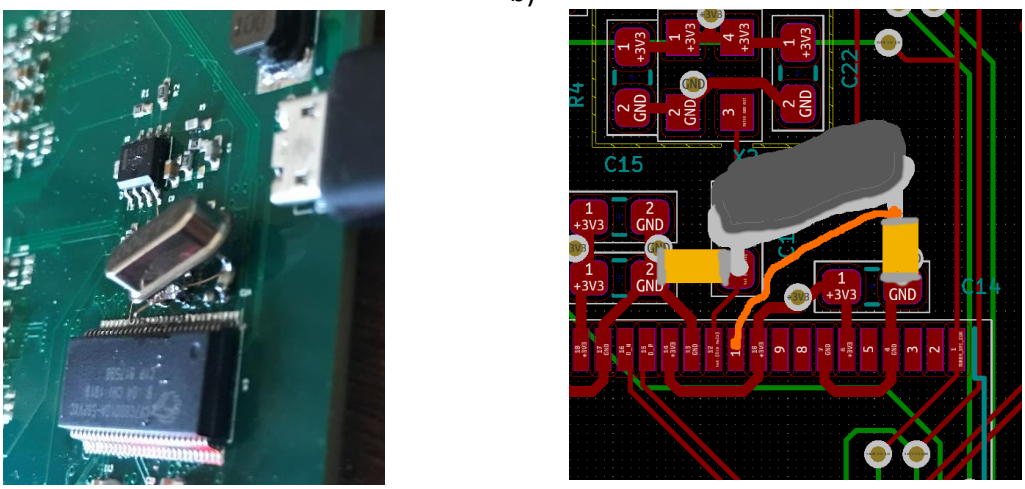


Figure 16: FX2LP 24 MHz crystal bodge for the v0.1 PCB; recommended workaround. a) Photo of implementation. b) View of the workaround on the PCB layout.

This error came about because the v0.1 PCB design aimed to use a single reference on the board and thus feed the FX2LP with the output of a shared oscillator rather than providing the FX2LP an external crystal for its internal Pierce oscillator circuitry. Per Cypress' recommendations, the v0.1 PCB leaves the XTALOUT pin of the FX2LP floating. On the v0.1 PCB the 24 MHz TCXO output is injected into the FX2LP's XTALIN pin via a 1nF AC coupling capacitor. Unfortunately, this setup has not yet allowed the FX2LP to successfully boot, despite attempting many combinations of inputs into the XTALIN port: 0-1V swing output of the TCXO, AC coupled, DC coupled; 0 to 3.3V swing output of the clock buffer, AC coupled, DC coupled. Feeding the XTALOUT pin with these same combinations has not worked either.

It would have been expected that at least one of these combinations should work, as Cypress documentation on the FX2LP [12, p. 26] does explicitly state "it is also correct to drive XTALIN with an external 24-MHz square wave derived from another clock source. When driving from an external source, the driving signal should be a 3.3-V square wave," and of the XTALOUT pin it is stated: "if an external clock is used to drive XTALIN, leave this pin open."

4.1.7.3. TCXO Choice, Clock Buffer Input Bias and Clock Swing

The v0.1 PCB DC couples the TCXO output directly to the input of the NB3L553-D clock buffer. This clock buffer, when supplied with V_{DD} = 3.3V, as on this SDR board, has its clock input low level (V_{IL}, I_{CLK}) at V_{DD} /2 – 0.7 and the high level (V_{IL}, I_{CLK}) at V_{DD} /2 + 0.7 [13, p. 4]. Depending on the choice of crystal oscillator, what its output swing is and whether its output is centred on V_{DD} /2, there may be issues getting the clock buffer to detect the clock input from the TCXO, particularly when the swing of the TCXO is not very large, as is the case for the TCXO chosen for this board (manufactured by TXC Corporation, part number 7Q-24.000MBN-T). This TCXO is not a good choice as it has a typical output swing of 0.8 V_{p-p}—too small for the selected clock buffer—and its output is not centred on V_{DD} /2, rather it outputs from ~0 - 0.8V as shown in Figure 6.

There are two possible workarounds here. The best workaround is to find a TCXO which matches the clock buffer input more closely, however for the v0.1 PCB this low output swing

TCXO was able to work in a pinch (the clock buffer specifications must be conservative) by AC coupling the TCXO to the clock buffer and biasing it. Figure 17 shows a way to implement this workaround, with biasing resistors hacked into a spare TCXO footprint which was originally included on the PCB in the event the tuners and FX2LP needed different reference frequencies, as would be the case if the use of a 24 MHz reference with the MSi001 (an undocumented mode of operation) did not work.

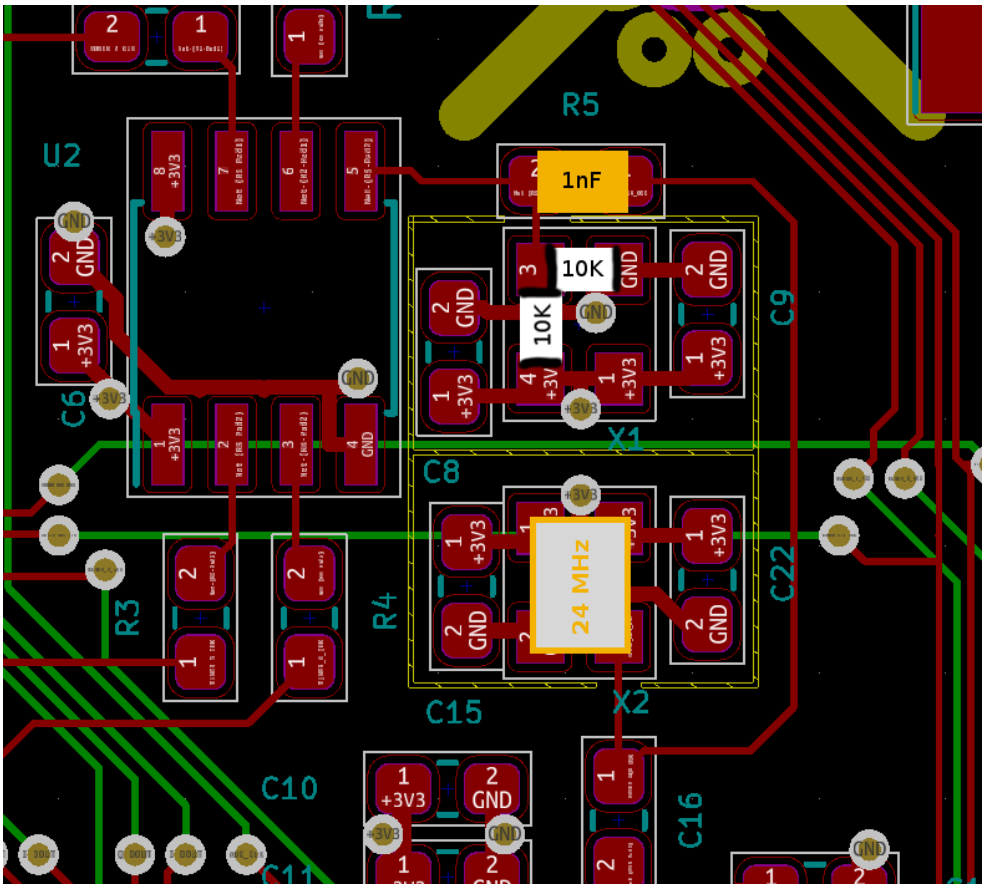


Figure 17: One possible workaround to AC couple the TCXO to the clock buffer on the v0.1 PCB.

4.1.7.4. Baseband Differential Op-amp Buffer Self Oscillation

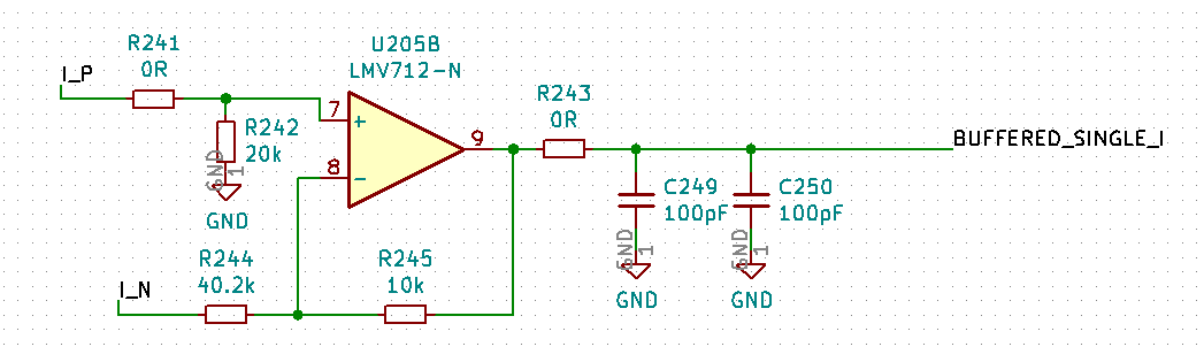


Figure 18: Differential op-amp buffer circuit as originally designed.

When assembled with the original 10 KΩ feedback resistors and 200 pF of output following capacitance, the buffer circuits (see an example in Figure 18) were found to self-oscillate (Figure 19) despite the datasheet for the LMV712-N proclaiming it can drive up to 200 pF. This

problem was solved by reducing the output following capacitance to 100 pF and reducing the differential gain by replacing the feedback resistor with a 2.2 KΩ feedback resistor.

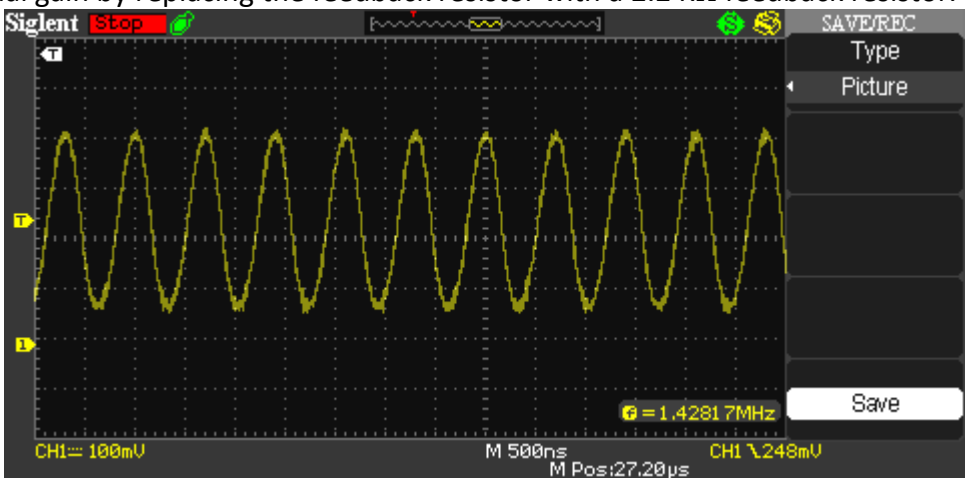


Figure 19: Op-amp self-oscillation with 200pF output loading. Solved by reducing loading to 100pF.

4.1.8. Assembling and Testing the PCB

The PCB has almost as many places it can go wrong as it does components. A simple short circuit hidden away, or a tiny chip placed backwards, can lead to much headache. For example, during assembly of this board a single short to ground, hidden under a component, meant needing to de-solder a chip at a time until the culprit was found. For this reason, there is a recommended order for assembly and testing of the prototype board, at least when assembly is done by hand. One should also get in the habit of checking for obvious shorts after doing a batch of work on the PCB—e.g. power to ground, or lines on a digital bus getting shorted together.

1. The USB connector and power supply, (including large value power filtering capacitors and inductors) should be soldered on. Then it should be verified there are no obvious shorts to ground on the power rails, before plugging the board in and verifying the presence of a 5 V rail and a 3.3 V rail.
2. The main TCXO and clock buffer should be soldered on, along with their associated decoupling capacitors, termination resistors, AC coupling caps, etc. After the board is checked again for shorts then powered, the presence of stable TCXO output and output from the clock buffer should be confirmed. If the TCXO has no output, experimentation with local power supply decoupling may help. If the TCXO has an output, but the clock buffer does not, then it should be verified that the oscillator swing seen by the clock buffer is sufficient and properly DC biased.
3. The FX2LP, crystal/clock source jumpers, indicator LEDs, I2C pullup resistors (but not the I2C EEPROM [11]), reset button and RC reset network should be soldered on, along with the associated decoupling capacitors. If one wishes to try using the TCXO or a clock buffer output to supply the FX2LP, then that should be routed in by inserting the appropriate AC coupling capacitors, 0 Ohm jumpers and/or bodge wires. However, as discussed in *4.1.7.2. 24 MHz Reference for the* FX2LP, the only source shown to work with the FX2LP so far is a separate crystal (not oscillator), which must be bodged onto the v0.1 PCB. Once this is done, the board should enumerate properly as a USB device, with Cypress as the manufacturer, and it should be possible to upload this project's

firmware, which will light the indicator LEDs (with one LED blinking) when it runs successfully.

4. The ADCs, differential op-amp buffer circuits and their decoupling capacitors can be added to the board, and it should be possible to read from them using the PC host application and see that the reading values match whatever test signals or voltages are applied.
5. The RF matching networks, LNAs and tuners can be soldered on. It may be prudent to leave the capacitors on the back side of the board off at this point in case rework of the front side needs to be done. Thus far the tuners and LNAs have been found to work OK without these components on the back side (they are mainly decoupling capacitors), so this is fine for testing functionality. At this point, the PC host software should be able to tune the SDR to a particular frequency and set its gain; it will be apparent when this has worked because any signals picked up on an antenna should now be visible in an FFT of the ADC samples. If there is no difference whatsoever from the previous step where the ADCs were tested (the noise floor should at least increase upon tuning, even if the tuner has no input signal), then this provides a clue that there may be an issue with the tuner or its soldering.
6. Solder on the capacitors on the back side.
7. Solder on the EEPROM if you wish to keep a copy of the firmware permanently stored (optional and not recommended because a bad firmware on the EEPROM may block booting and flashing, requiring the EEPROM to be taken off and wiped or re-flashed externally).

A reference of use for debugging issues getting the FX2LP up and running is Cypress' *Guide to a Successful EZ-USB® FX2LP™ Hardware Design* [11], particularly the section *Bringing Up the Board*.

4.2. PC Host Application

The host application was written to send USB commands to the board to configure the tuners and to retrieve and assemble the data captured from the ADCs by the FX2LP. The host application has been written and tested on Linux but should port easily to any environment with libusb. Its only dependencies are gcc and libusb-1.0 (the libusb-1.0 dev headers are needed too).

The USB commands sent between the host and the board are detailed further in section *4.3.4. USB* Vendor Commands. The payloads put in the USB command for tuner configuration are the SPI register writes detailed in the Mirics MSi001 datasheet [2].

To retrieve ADC readings, this utility makes USB bulk transfer requests to the FX2LP's endpoint 2 (IN), which is set up in the FX2LP firmware as the destination for ADC readings. The host application uses multiple asynchronous threads to saturate the USB bus with bulk transfer requests and so keep the buffers on the FX2LP well-serviced. If the FX2LP GPIF's FIFO buffers are allowed to fill completely, ADC reads are paused, and so sampling becomes irregular. This technique of spamming bulk transfer requests was picked up from the Sigrok project's host-side code [14] (see lines 674 to 694).

4.2.1. Re-structuring the USB Packets/GPIF FIFO data into ADC Samples

The packets from the board require some reassembly on the part of the host application to be put into the format of unsigned 16-bit I/Q samples. This is because the FX2LP's GPIF reads in a whole byte per clock via a parallel interface, whereas each ADC only has a single data line and shifts data out serially, one bit per clock. The obvious solution in hardware was to connect the total of 8 serial data lines from the ADCs to the 8-bit wide parallel GPIF bus and then clock the bus at the serial data rate of the ADCs (48 MHz for 3 M 12-bit samples/second). However, the result is that the data in the USB packets needs every set of ADC samples to be transposed and padded out to 16 bits.

There is a naïve solution which does this bit matrix transposition bit-by-bit, and it runs just fast enough on a dedicated thread of a mid-range PC to handle the data coming from the board at ~300 Mbps, however a neat solution for bit matrix transposition worth mentioning, and which is implemented in the PC host application, is an algorithm found in *Hacker's Delight* [15, pp. 108-111]. This algorithm performs well on a 64-bit computer, where rather than shifting and setting single bits per instruction (as in the naïve implementation), whole 64-bit registers are masked and shifted in place. Using this algorithm, throughput on a single thread was increased up to 3-4 Gbps on the same PC.

4.2.2. Usage of the PC host application

This section details some useful ways the PC host application can be used and how to call it (from the terminal).

4.2.2.1. Tuning and Setting Gain

Presently, this functionality is hard-coded, and the PC host application sets this fixed gain, centre frequency and bandwidth upon start-up. This is not because of any technical limitation, but rather a severe lack of time before the writing of this report had to commence! Currently, to change frequency, etc. constants in the source code are modified and the PC host application rebuilt. The host application is easily rebuilt by calling `make` in the top-level directory of the PC host application's source code.

4.2.2.2. Raw Binary Output of ADC Samples (to file)

```
./main -t > <path-to-file.bin>
```

When called with the -t flag, the PC host application will stream data from the SDR board to stdout, which can easily be piped into a file for later analysis by other software; for example, MATLAB or Octave (in which case the `fread` function is helpful).

Table 3: The multi-channel raw binary I/Q data output format of the PC host application. Actual output contents are shaded.

ADC Sample Index	Offset	uint16 (little endian)	uint16 (little endian)	uint16 (little endian)	uint16 (little endian)	uint16 (little endian)	uint16 (little endian)	uint16 (little endian)	uint16 (little endian)
0	0x0000	CH1 I	CH1 Q	CH2 I	CH2 Q	CH3 I	CH3 Q	CH4 I	CH4 Q
1	0x0010	CH1 I	CH1 Q	CH2 I	CH2 Q	CH3 I	CH3 Q	CH4 I	CH4 Q
…	…	…	…	…	…	…	…	…	…

4.2.2.3. Raw Binary Output of USB Packet Contents

To print USB packet contents straight to stdout (messy and not very useful):

```
./main -s
```

To store for these contents for later analysis:

```
./main -s > <path-to-file.bin>
```

For analysis of the USB packet contents as a stream of human-readable hex values:

```
./main -s | xxd | less
```

Sometimes it is favourable to view the raw payload contents of the USB packets; this allows pretty much what the exact contents of the FIFO buffers on the FX2LP were to be inspected, as read in by the GPIF, and prior to any padding or transposition occurring on the host side. This has been useful when fixing issues such as the FIFO buffers on the FX2LP being overwritten by an overflow, losing samples or ADC samples becoming misaligned to different offsets from packet to packet. It was also invaluable to the design and modification of the GPIF waveforms running on the FX2LP.

4.2.2.4. Human-Readable Channel Output

```
./main
```

When the PC host application is called without arguments, it defaults to printing a human-readable table of ADC samples from the channels, printed as hex. This is particularly useful during debugging of the FX2LP firmware or when assembling the PCB and needing to check that all ADCs are operating properly. To view live changes, the watch command can be used, like so:

```
watch -n 0.1 ./main
```

```
Every 0.1s: ./main                                              ante

Read in 504 bytes

n       CH0     CH1     CH2     CH3     CH4     CH5     CH6     CH7
0:      089     070     5b0     084     1eb     19e     1c7     1d4
1:      1ae     1a6     5c1     1b5     072     083     07a     06e
2:      08d     086     5ce     087     1d2     188     1b6     1b3
3:      1a2     188     5e1     1aa     074     093     07b     07c
4:      090     09e     5ee     089     1c6     179     1ad     194
5:      18e     16b     601     199     075     0a2     07b     092
6:      0a0     0bc     609     095     1bd     168     1a3     16e
7:      16f     14d     615     181     077     0b0     07c     0b6
8:      0be     0db     624     0aa     1b3     15c     198     146
9:      14e     132     62e     161     07b     0be     081     0e2
10:     0e5     0fb     63d     0cb     1a5     152     185     11c
11:     126     11a     64f     13e     083     0ca     08e     112
12:     112     119     65f     0f5     191     148     170     0f8
13:     102     103     671     11a     091     0d4     0a1     13e
14:     13d     133     67c     121     17b     140     155     0dc
15:     0e4     0f0     68c     0f7     0a5     0de     0bb     15f
16:     15e     14b     694     14b     163     13b     13d     0c5
17:     0cc     0de     69e     0db     0bd     0e5     0d8     172
18:     172     15f     6ad     16c     14e     133     122     0b2
19:     0ba     0d2     6b7     0c4     0d6     0ed     0f7     17e
20:     17e     16a     6c5     17e     137     12e     10d     0aa
21:     0b3     0c5     6cb     0b6     0f0     0f3     113     17c
22:     17b     173     6d4     189     122     12a     0f7     0a8
23:     0b4     0bd     6dd     0a9     108     0fa     12d     172
```

Figure 20: PC host application called without arguments outputs human-readable data from each channel.

4.3. Cypress FX2LP Firmware

The firmware has two main components: the GPIF design/configuration, which controls the sampling and data acquisition from the ADCs; and the C code, which handles USB commands, configures the tuners over SPI and initialises and triggers the GPIF.

4.3.1. Uploading the FX2LP Firmware

To build and upload the firmware, the following dependencies are required on the build system:

- make
- gcc
- libusb-1.0
- SDCC
- python 2.7
- swig

Uploading of the firmware binary to the FX2LP relies on the fx2load utility provided by fx2lib. This utility is a Python script with dependencies which are built as part of fx2lib's build process, which can be completed by running `make` in the fx2lib folder of the project. Once fx2lib is built, fx2load can be found in the path `fx2lib/examples/fx2/scripts/` and needs to be added to the search PATH of the system.

After this, the SDR's FX2LP firmware is built and loaded into the FX2LP's RAM, via USB, by navigating to the project's FX2LP firmware folder (`fx2fw/fx2sdr/`) and running:

```
make load
```

At present, each time the SDR board has power removed, it loses the firmware, as fx2load loads firmware into RAM rather than EEPROM. This is great for development as it makes it simple to wipe a bad firmware image which freezes USB communication (and thus prevents further firmware uploads), for example. However, now that the firmware is less experimental it could be time to flash the EEPROM instead.

4.3.2. GPIF Design

The GPIF is a part of the FX2LP which has the ability, when configured and triggered by the firmware, to push masses of data directly over USB without intervention; at the hardware end, it can implement a flexible range of protocols on a bank of GPIO pins as determined by a user-programmable state machine.

As part of this project, a GPIF "waveform" (Cypress' terminology for one of these state machines) was created to drive the sampling of the ADCs, provide them a clock and collect the (serial) data from them. The ADCs used in this project—either the MAX19777 or ADS7883—follow the protocol shown in Figure 21. Note this protocol has 16 clocks per 12-bit ADC conversion, so the GPIF must clock 16 times per sample but only needs to read on 12 of them. The GPIF also must supply a chip select signal (nCS) which controls sampling.

The GPIF waveform shown in Figure 22 is actually intended to read data on 13 out of 16 clock cycles (data read states are indicated on the magenta coloured "Data" signal) so that the fixed

zero at the start of a conversion is read; this helps with detecting sample alignment issues further up the software stack, at the cost of a bit of bandwidth.

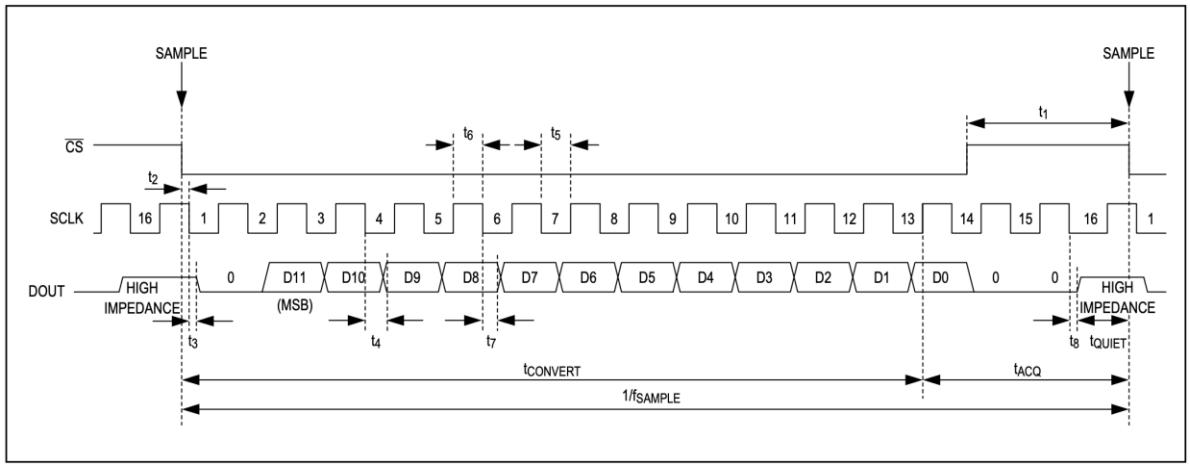


Figure 21: ADC interface signals for maximum throughput. Reproduced from the Maxim MAX19777 datasheet [3].

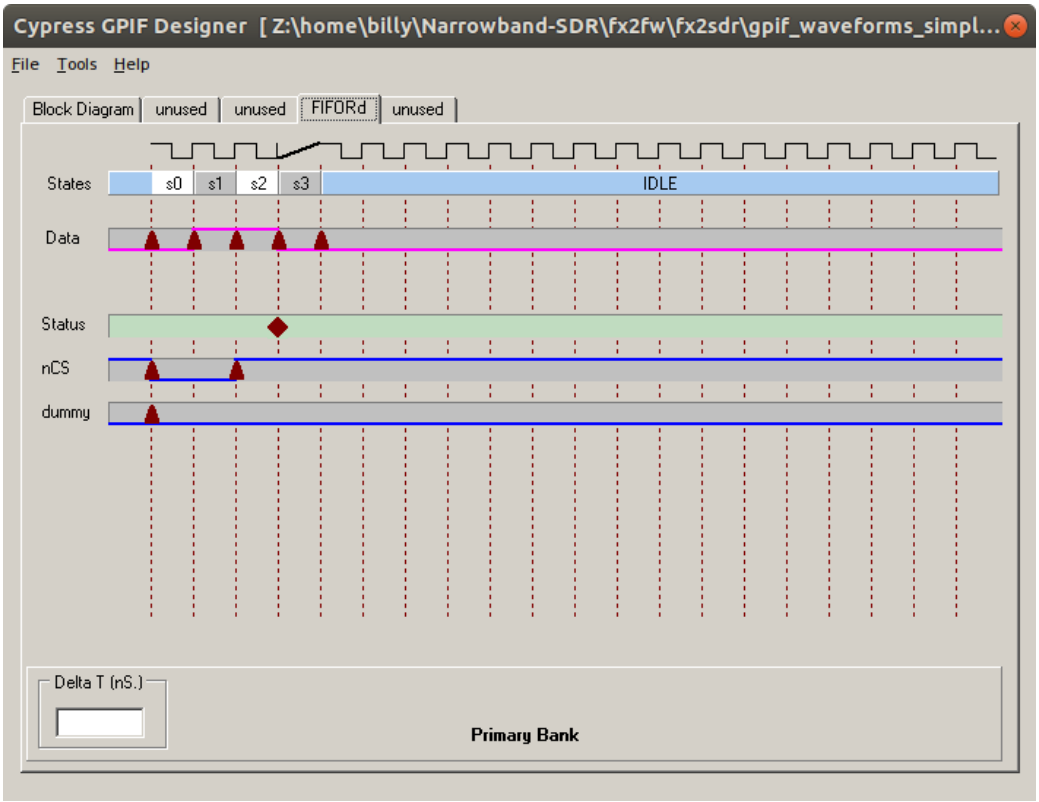


Figure 22: the basic GPIF "waveform" used to interface the ADCs. Not shown: s0 lasts for two clocks; s1 for 12; s2 for one; s3 also for one. The decision point (maroon diamond) considers if the FIFO buffer is full or a count of transactions has been depleted; if so, the idle state is entered after s3; otherwise, the GPIF loops back to re-execute from s0.

4.3.2.1. Hacking the GPIF Flow State

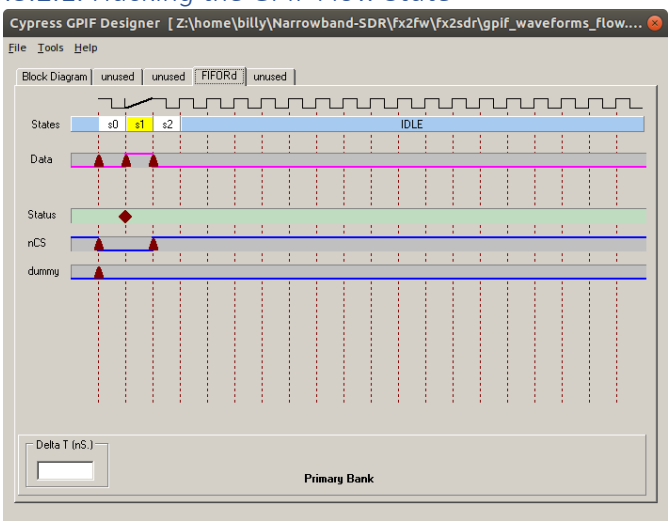


Figure 23: A separate GPIF waveform utilising a "flow state"; s1 is in yellow as it is set as the flow state.

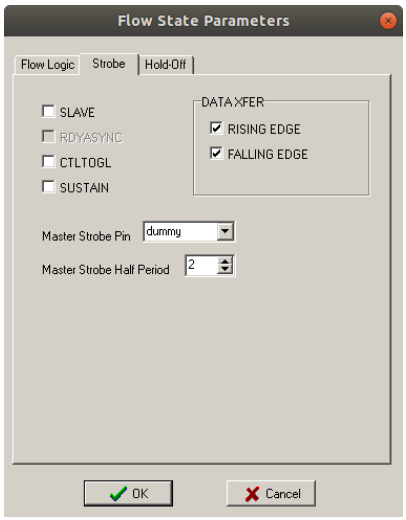


Figure 24: crucial flow state settings.

The crucial part of creating the GPIF configuration used in this project is that the C files exported from the *Cypress GPIF Designer* for both GPIF waveforms (i.e. the basic waveform shown in Figure 22 and the one with the flow state shown in Figure 23) need to be manually combined in a text editor. This manual combination of the two waveforms allows s1 to be a flow state and yet terminate after a fixed number of clock cycles, as in the basic waveform. To do this, in the C file for the basic waveform, the data in the `FlowStates[]` array needs to be replaced with the same section from the waveform which utilises the flow state.

This is necessary because the s1 state in the basic waveform, although lasting for 12 clock cycles and having the flag set for storing data, will only store the data from the first clock edge and then sit idle for the rest of the remaining 11 clock cycles, whereas this application needs data read in on all 12 rising clock edges. Since each state, regardless of duration, can make one transfer at most, and since the GPIF only has 7 programmable states available, enough additional states cannot be simply inserted. A flow state is the way around this, as they last for an indefinite period and can transfer on every clock period, terminating only when certain programmable conditions and flags are set. However, the documentation on the GPIF does not detail any way to use a simple number of clock cycles as the terminating condition for a flow state (to the best of my knowledge) and the *Cypress GPIF Designer* explicitly disables the option of setting the state duration of a flow state. It appears to be an undocumented feature that the flow state can have a duration (manually) set using the workaround detailed here.

Without being able to leverage this hidden feature, the GPIF would either be unable to create the pulsed nCS signal which the ADCs use as a trigger for sampling, or the GPIF would have to drop data.

Note that in Figure 24, under the "Hold-Off" tab (the contents of which are not pictured), hold-off is disabled by unchecking the "HOCTL pin is asserted when Not Ready (HOSTATE)" checkbox. Similarly, in the "Flow Logic" tab, the conditional does not matter, as long as none of the control pins are set to change. This can all be seen in the project files.

4.3.3. GPIF Initialisation and Triggering

The source code of the project is the best template on this topic, rather than this document, however there are a few things worth mentioning which have caused headaches:

- `gpif_fifo_read(0);` Will actually start reading into endpoint 2, which corresponds to **index 0** of the endpoint FIFOs.
- The FIFO auto-commit packet size, which determines how many bytes make up the payload of a USB packet from the FIFO, should be a multiple of the number of bytes the GPIF stores per ADC conversion cycle—this ensures an ADC sample is not split across two packets, and that a GPIF decision point check can get hit before an overflow; either of these not being the case can make alignment or data loss a concern. As mentioned previously, the current GPIF design takes 13 bytes per ADC conversion cycle (instead of the minimum 12) to ensure zeros are read in for alignment issue detection when debugging. Since the max USB 2.0 bulk transfer packet size is 512 bytes, the nearest allowable auto-commit packet size under these conditions is 507 bytes (39 ADC samples, each with a padding zero).

4.3.4. USB Vendor Commands

Table 4: Custom vendor commands implemented on the fx2sdr firmware.

Endpoint	**bRequest**	**wValue (16 bits)**	**wIndex**	**data**
EP0 OUT (control transfer) (vendor command)	0xB0	Tuner CS lines […, X, X, b3, b2, b1, b0] e.g. …0001 to select Ch. 1, or …0011 to select Ch. 1 and Ch. 2	<ignored>	Bytes for the FX2LP to write directly over SPI. MSB first. SPI mode 0.

4.3.5. Low Speed Software-Implemented SPI

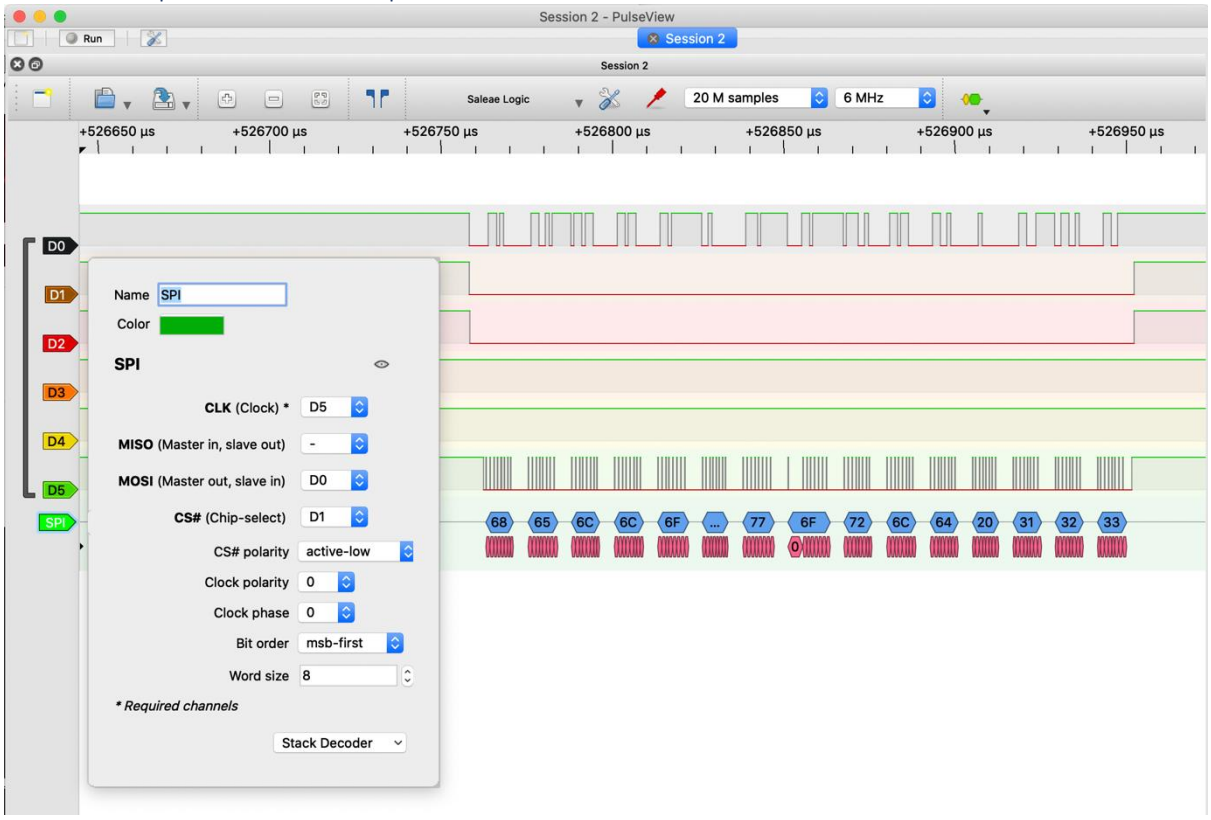


Figure 25: Software-implemented SPI; notice the clock (on channel D5) is irregular.

The FX2LP does not include a hardware SPI module, so a software implementation provided by Cypress [16] is used to write the bits out over the GPIO ports to the MSi001 tuners at ~500 kHz, following the SPI clock polarity and phase given in the MSi001 datasheet [2]. It is possible to get away with this simple solution as a low speed SPI bus is sufficient for occasionally writing configuration data to the tuners.

However, the particular GPIO ports used for this low speed SPI connection have also been chosen from the spare GPIF ports; this was done in case the software SPI were to turn out too slow for the tuners (some devices have a minimum clock rate), in which case a GPIF routine could be created to implement an SPI master at 30 MHz or 48 MHz.

4.4. Test Setup for Debugging the Firmware and PC Host Software

In the early stages of the project, with an FX2LP development board on hand, but without access to the ADCs or tuners, the setup shown in Figure 26 served as the development environment for the FX2LP firmware and the PC host software. An FPGA was loaded with a design intended to emulate the serial protocol of the ADCs, as given in the MAX19777 datasheet and reproduced in Figure 21. The mock ADC samples supplied by the FPGA were simply the value of a 12-bit counter which incremented on each sample, and each of the 8 ADC channels was set to increment at a different rate, so as to be distinguishable from one another.

This test setup was revived again when encountering issues making the SDR board work and needing to develop the firmware and host software further, so it does appear to have value for debugging and development when wanting to isolate a problem from the other components of the SDR.

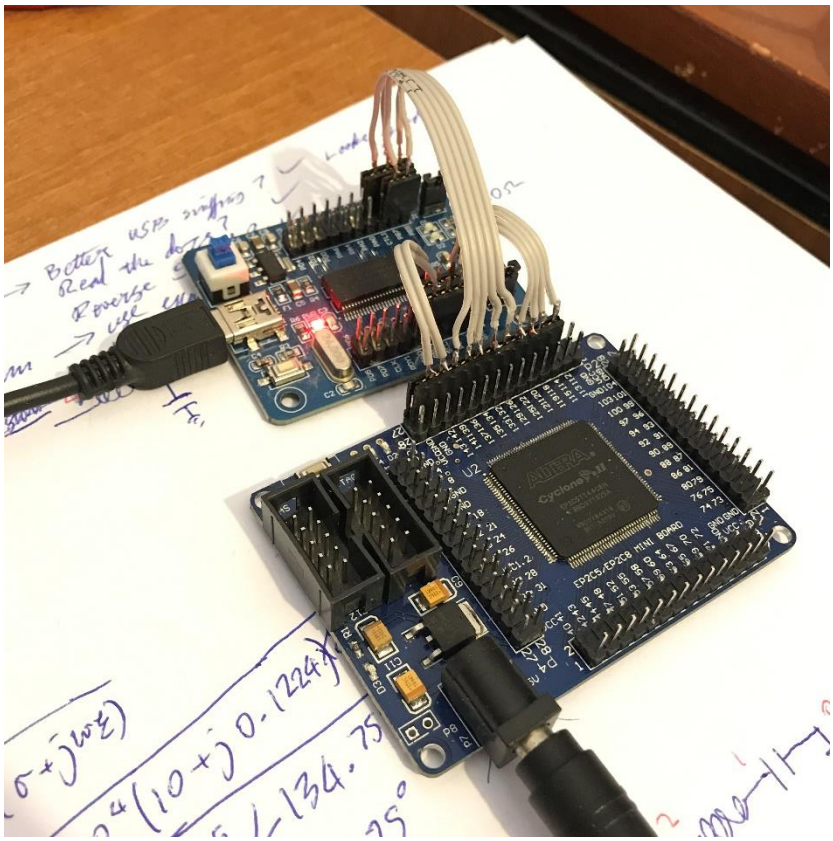


Figure 26: Software and firmware test setup, which consists of an FX2LP development board interfaced with an FPGA configured as 8 counters with serial interfaces. Signal integrity was suspected to be marginal at 48 MHz.

The expected behaviour of the test setup when the firmware and the PC host software are working correctly is to see a sawtooth wave pattern with a different frequency on each channel, counting between 0 and 4095 in the increment of the particular channel's counter. Figure 27 was as close as the test setup got to this goal, and evidence suggestive of signal integrity issues is present (unfortunately equipment was not available to investigate). Note that the SDR board does not show any evidence of the same.

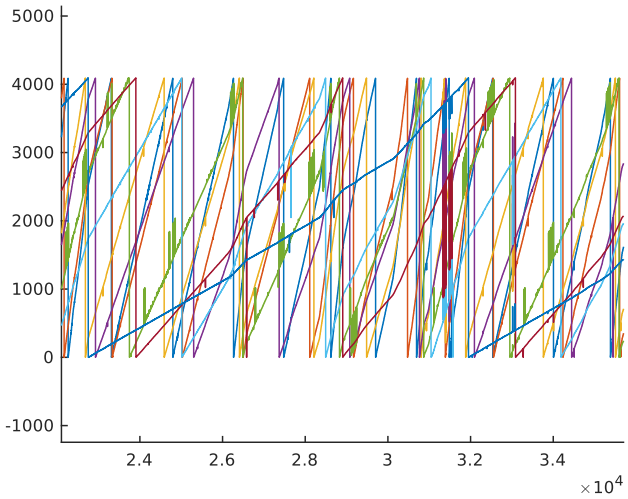


Figure 27: The incrementing of all 8 counters, as received by the PC host software.

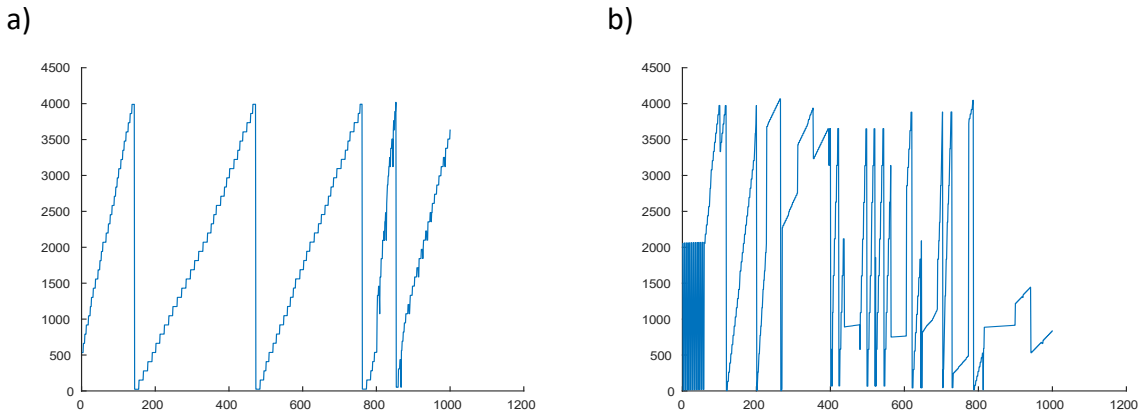


Figure 28: Count value, as seen on one channel by the PC host software, a) showing unexpectedly granular incrementing, b) showing unexpected jumps and unexpected changes in incrementation velocity.

Figure 28 shows examples of different issues in the data path, e.g. left shifted and missing samples in Figure 28a.

5. Results

At the conclusion of the project, all software and hardware components have been confirmed to be working, up to the point of receiving FM broadcast transmissions and being able to plot an FFT which shows clear reception of distinct signals. However, the exact performance of the device, such as noise figure across different bands, intermodulation products, phase noise, etc. and the exact extents of its operating range (i.e. sensitivity, reception bands/gaps) has not been determined due to a lack of access to appropriate test equipment and a lack of time.

For the plots in this section of the report, MATLAB has been used to process the raw binary I/Q data saved by the PC host software.

5.1. ADC Noise Floor

a) b)

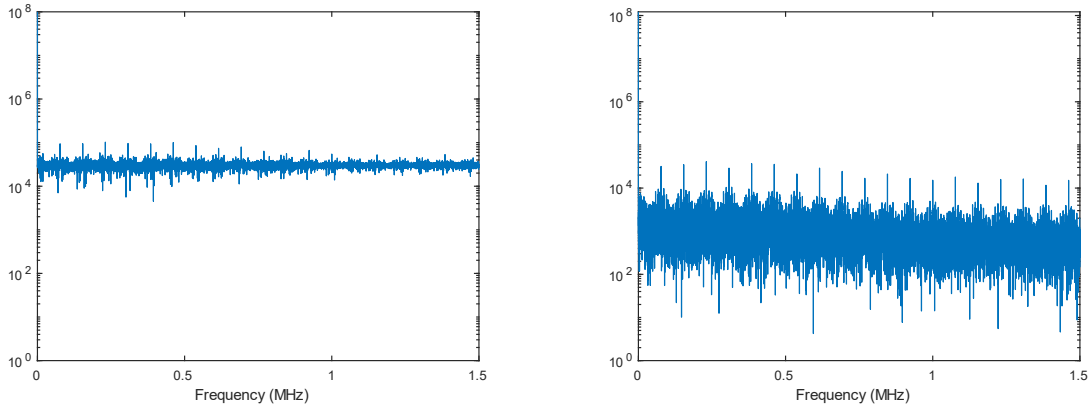


Figure 29: Comparative indication of noise floor on the two options for ADCs installed on the board. Measurements were taken with the tuner uninitialised, i.e. with its outputs off. a) MAX19777. b) ADS7883.

Figure 29 compares the noise floor of samples from the two different options for ADCs, tested while connected onboard the SDR, with the op-amp buffer included in circuit, but with the MSi001 left un-initialised and so producing a constant output. Keep in mind that the ADS7883 is marketed with notably higher performance specifications, and as expected its noise floor is seen to be considerably lower. However, the same spurs (with a similar magnitude) are clearly visible in the samples from both ADCs, which does indicate some source of interference on the board.

5.2. Integration Testing

The integration test that has been performed so far is to prove reception of FM signals on all four channels of the receiver and be able to store and plot this data. Successful completion of this test proves that: the tuners are functional and there is an RF path to them; the ADC and differential buffer circuits are working; the FX2LP is collecting and transferring samples reliably; the PC host software is polling the USB bus frequently and consistently enough to achieve the required data rate; and the PC host software is correctly restructuring and storing the baseband samples.

a) b)

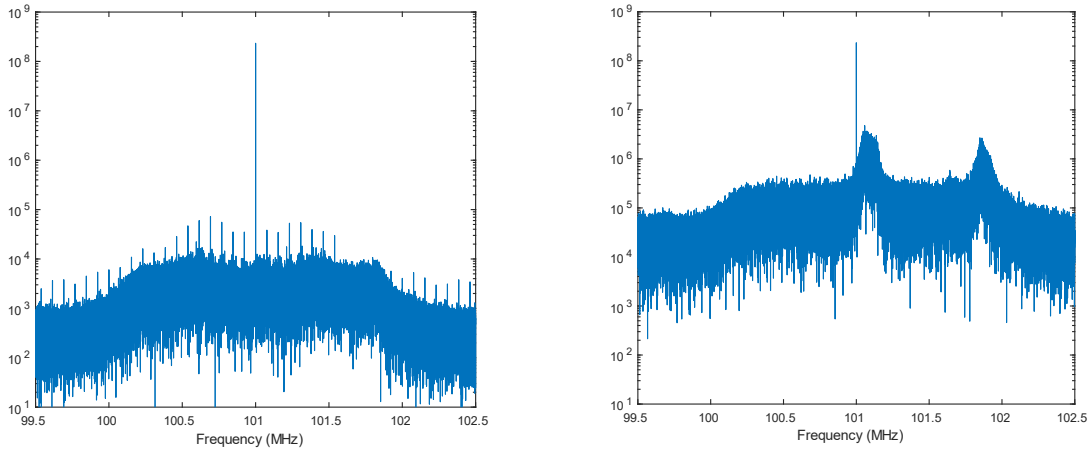


Figure 30: FFT of I/Q samples from a channel of the SDR tuned to 101 MHz, sampled with the ADS7883 and with 1.536 MHz double-sided bandwidth set on the tuner. a) No antenna attached. b) Antenna attached.

Figure 30a shows an FFT of the sampled baseband output of the tuner when set for exactly the same frequency and gain settings as in the Figure 30b, but without an antenna attached. That there is a clear absence of the two FM channels seen in Figure 30b indicates that RF is not simply leaking into the board, but rather is travelling through the intended RF frontend circuitry.

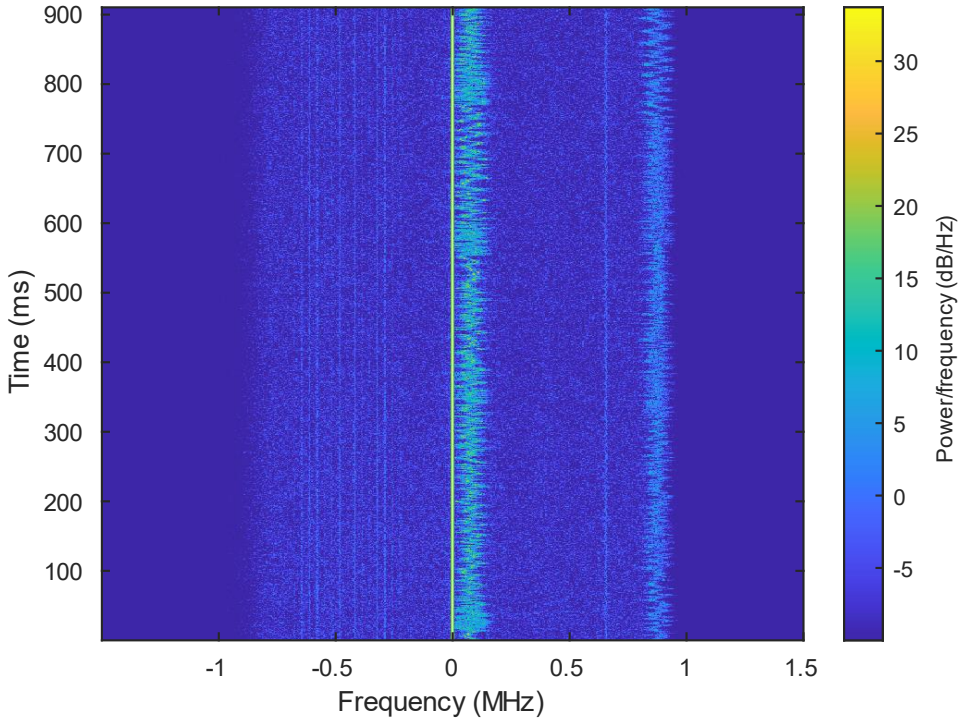


Figure 31: Corresponding spectrogram of the baseband FM signals for Figure 30b.

Further, it has been verified (see Figure 32) that with an antenna attached to only one channel, and all tuners configured the same, the other channels without antennas do not see these signals, which would be the case if there was unwanted crosstalk or some kind of coupling between the receiver channels.

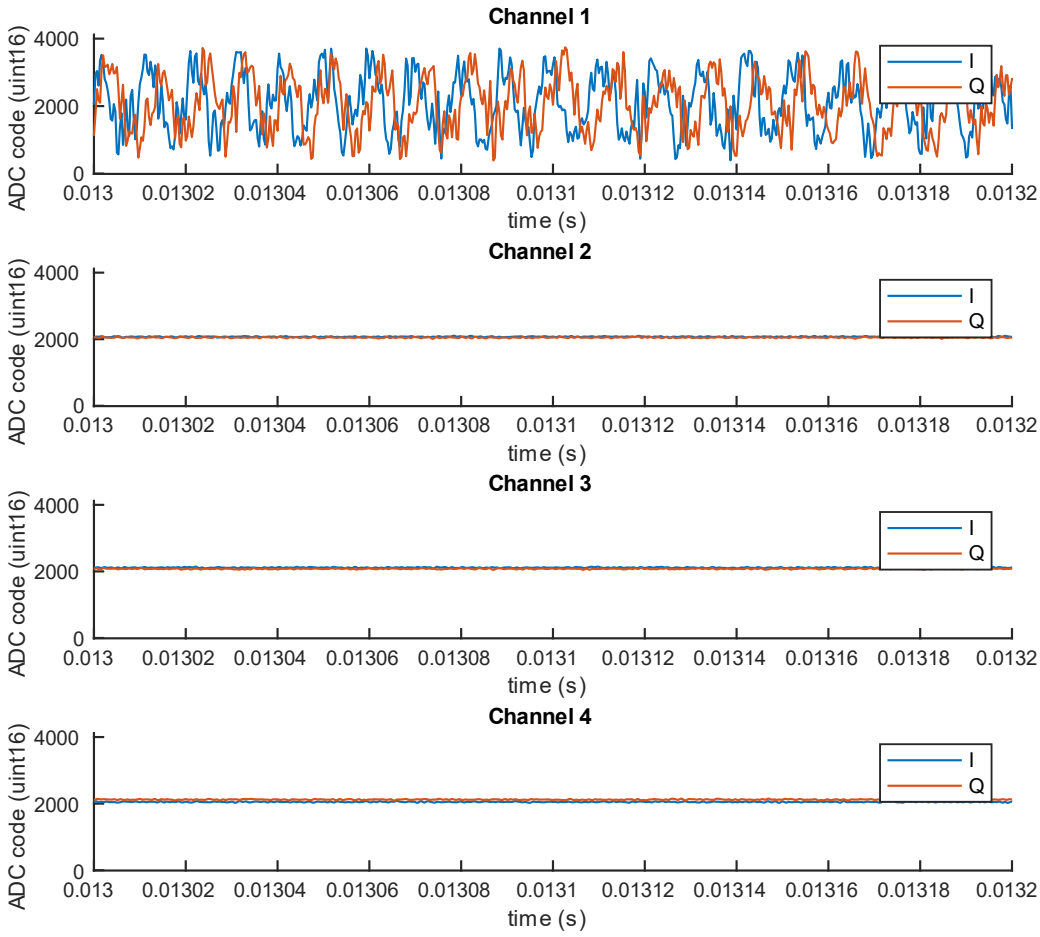


Figure 32: I/Q samples in the time domain. All channels are identically configured but an antenna is only connected on channel 1.

It should be noted that the SDR currently has a noticeable DC offset spike at the tuned centre frequency. This spike comes about because the baseband signals from the differential buffer are centred on $V_{CC}/2$, as they have to be, given the single rail supply available for the analog circuitry. This is common for almost all SDRs and is usually filtered out in software [17]. Also of note in the no-antenna test (Figure 30a) is the presence of spurs; since they match those seen in Figure 29, it would appear these are not produced by the RF frontend but are most likely caused by noise in the differential op-amp buffer or ADC, as previously discussed in *5.1. ADC Noise Floor.* Nevertheless, with an antenna connected, the combined noise floor of the received signal and RF frontend appears to be far more dominant than this conjectured ADC noise floor, as the spurs are hidden.

Figure 30 also suggests that the bandwidth setting of the MSi001 tuner is being properly set, as clear roll off is seen outside the 1.535 MHz double sided bandwidth that has been selected. The actual bandwidth appears a bit larger than 1.535 MHz, but regardless there is still much sampling of empty spectrum. The next widest selectable bandwidth is 4.6 MHz [2, p. 10], and this has been tested, but the expected aliasing is present.

Figure 33 is included as a demonstration of the successful simultaneous operation of all channels, showing the complex baseband output of all channels.

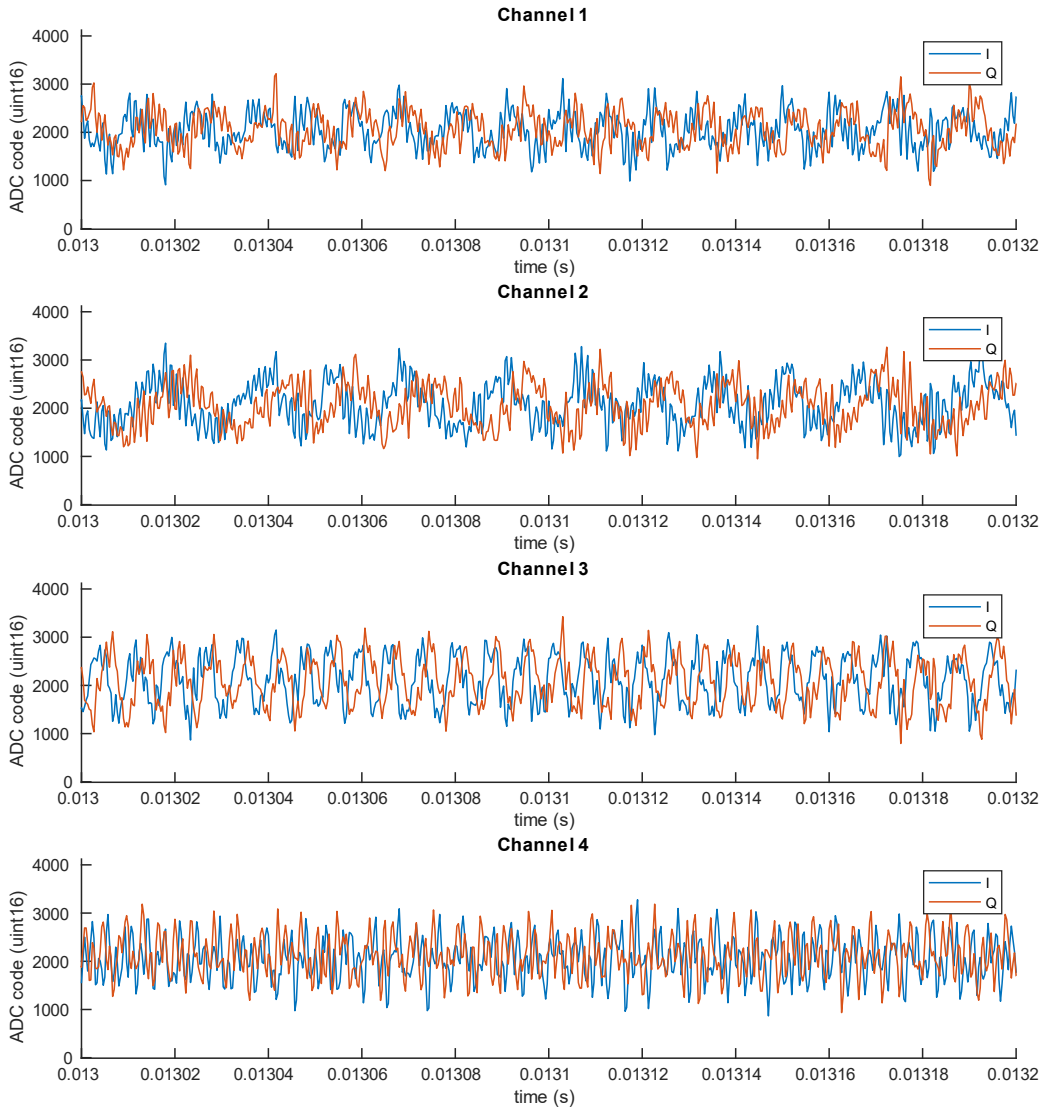


Figure 33: I/Q samples in the time domain gathered from all four channels of the receiver simultaneously, with each channel tuned to a different centre frequency.

6. Conclusions

The project has broadly met its initial objectives, except for the characterisation of the performance of the finished device and demonstration of phase-coherence between the channels. The original design specification proved to be a well-matched selection of parts and software architecture, only requiring a departure in the choice of ADCs. Room still likely exists for optimisation of areas of the circuit design where hand-waving estimates have been made or additional footprints expressly set aside for future experimentation currently sit populated with quick solutions. Nevertheless, the architecture has been demonstrated successfully.

This project has also resulted in a useful repository of designs and code which allows the recreation of the work detailed in this report or the creation of derivatives, thus giving back to the open source software community which provided the framework and tools to enable this project. This means the project can serve as a working reference design for implementing and interfacing the Mirics MSi001 tuner chip or simultaneously sampling from 8 high speed serial ADCs with the Cypress FX2LP, which may assist those interested in the use of these chips. Similarly, the discovery made in this project of hidden functionality in the Cypress FX2LP's GPIF, and an example of how to use it, may be of use for similar reasons.

With further development of host-side software, it is hoped that the affordable hardware created in this project will open a range of experiments and applications to hobbyists and educators alike. It has certainly educated me, if only on a small fraction of a much larger body of knowledge, and has provided me exposure to new design techniques and a new selection of components out of the endless and incredible choices available.

7. Future Work

The source and design files for the project can be found at:
https://github.com/BillyWoods/Narrowband-SDR.

GNU Radio Source Block

Popular for building real-time SDR processing chains, a GNU Radio source block would be a logical next step for processing data, rather than the current static solution of collecting a file, loading it into MATLAB, and then processing it there.

Channel Calibration

Presently the SDR is fine for operation as if it were four separate receivers, but the end goal is to make use of the common clock and simultaneous sampling. Coherent reception has not been successfully demonstrated yet, and investigation into the possible calibration of the tuners appears to be necessary. Further, it is possible that the differences between the components of each channel is significant enough to require the channels to be correlated and calibrated in software.

Performance Characterisation

So far characterisation of the device has been rudimentary i.e.: things are working, because an FM signal is present well above the noise floor and the time domain plots appear correct (see *5.2.* Integration). However, to test with a properly calibrated signal generator would allow the sensitivity, noise floor and intermodulation products to be measured and understood over the whole input range of the SDR.

RF Frontend Optimisation

Since the current matching network is a rudimentary resistive power divider, as discussed in section *4.1.6. RF Inputs and Matching Network*, a logical next step, especially if performance can be more quantitatively measured, would be the design and implementation of a more efficient matching network.

ADC Noise Reduction

The spurs discussed in section *5.1. ADC Noise Floor,* although seemingly well hidden under the RF front-end noise floor in some conditions (and so ignored for the moment) should ideally not be present. As part of future work, the source of these spurs could be tracked down and an improvement made.

PCB v0.2

If another run of PCBs is made, this would be an opportunity to address *4.1.7. Errata on the v0.1* PCB.

8. References

[1] KerberosSDR, “About - KerberosSDR,” [Online]. Available: http://kerberossdr.com/about/. [Accessed 5 November 2020].

[2] Mirics Semiconductor Inc., “MSi001 Multi-mode Tuner,” [Online]. Available: http://www.cqham.ru/forum/attachment.php?attachmentid=145189&d=1373880883. [Accessed 4 November 2020].

[3] Maxim Integrated Products, Inc., “MAX19777 3Msps, Low-Power, Serial 12-Bit ADC,” 2017. [Online]. Available: https://datasheets.maximintegrated.com/en/ds/MAX19777.pdf. [Accessed 4 November 2020].

[4] JLCPCB, “Multilayer high precision PCB’s with impedance control,” JLCPCB, [Online]. Available: https://cart.jlcpcb.com/impedance. [Accessed 6 November 2020].

[5] JLCPCB, “Impedance Calculation,” [Online]. Available: https://cart.jlcpcb.com/impedanceCalculation. [Accessed 6 November 2020].

[6] Microchip Technology Inc, “MCP1825/MCP1825S 500 mA, Low Voltage, Low Quiescent Current LDO Regulator,” [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/22056b.pdf. [Accessed 6 November 2020].

[7] Analog Devices, Inc., “ADF4351 - Wideband Synthesizer,” [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ADF4351.pdf. [Accessed 6 November 2020].

[8] A. Palosaari, “Logitec LDT-1S310U/J teardown!,” [Online]. Available: http://blog.palosaari.fi/2013/10/naked-hardware-13-logitec-ldt-1s310uj.html. [Accessed 6 November 2020].

[9] Wikipedia, “Charge pump,” [Online]. Available: https://en.wikipedia.org/wiki/Charge_pump#Terminology_for_PLL. [Accessed 6 November 2020].

[10] D. Bannerjee, “PLL Fundamentals Part 3: PLL Design,” Texas Instruments (National Semiconductor), [Online]. Available: https://www.ti.com/lit/ml/snap003/snap003.pdf?ts=1604657358494. [Accessed 6 November 2020].

[11] R. S. K. Vakkantula, “AN15456 - Guide to a Successful EZ-USB® FX2LP™ Hardware Design,” Cypress Semiconductor Corporation, [Online]. Available: https://www.cypress.com/file/135006/download. [Accessed 6 November 2020].

[12] Cypress Semiconductor Corporation, “EZ-USB® FX2LP™ USB Microcontroller,” [Online]. Available: https://www.cypress.com/file/138911/download. [Accessed 6 November 2020].

[13] Semiconductor Components Industries, LLC (ON Semiconductor), “NB3L553 - Clock / Data Fanout Buffer, 2.5 V / 3.3 V / 5.0 V 1:4,” August 2019. [Online]. Available: https://www.onsemi.com/pub/Collateral/NB3L553-D.PDF. [Accessed 6 November 2020].

[14] The Sigrok Project, “[libsigrok.git] / src / hardware / fx2lafw / protocol.c,” [Online]. Available:

https://sigrok.org/gitweb/?p=libsigrok.git;a=blob;f=src/hardware/fx2lafw/protocol.c;h=963d0336e59c4000e24a0f51542776cbda9678b4;hb=HEAD. [Accessed 6 November 2020].

[15] H. S. Warren, “Hacker's Delight,” Boston, MA, USA, Pearson Education, 2003, pp. 108-111.

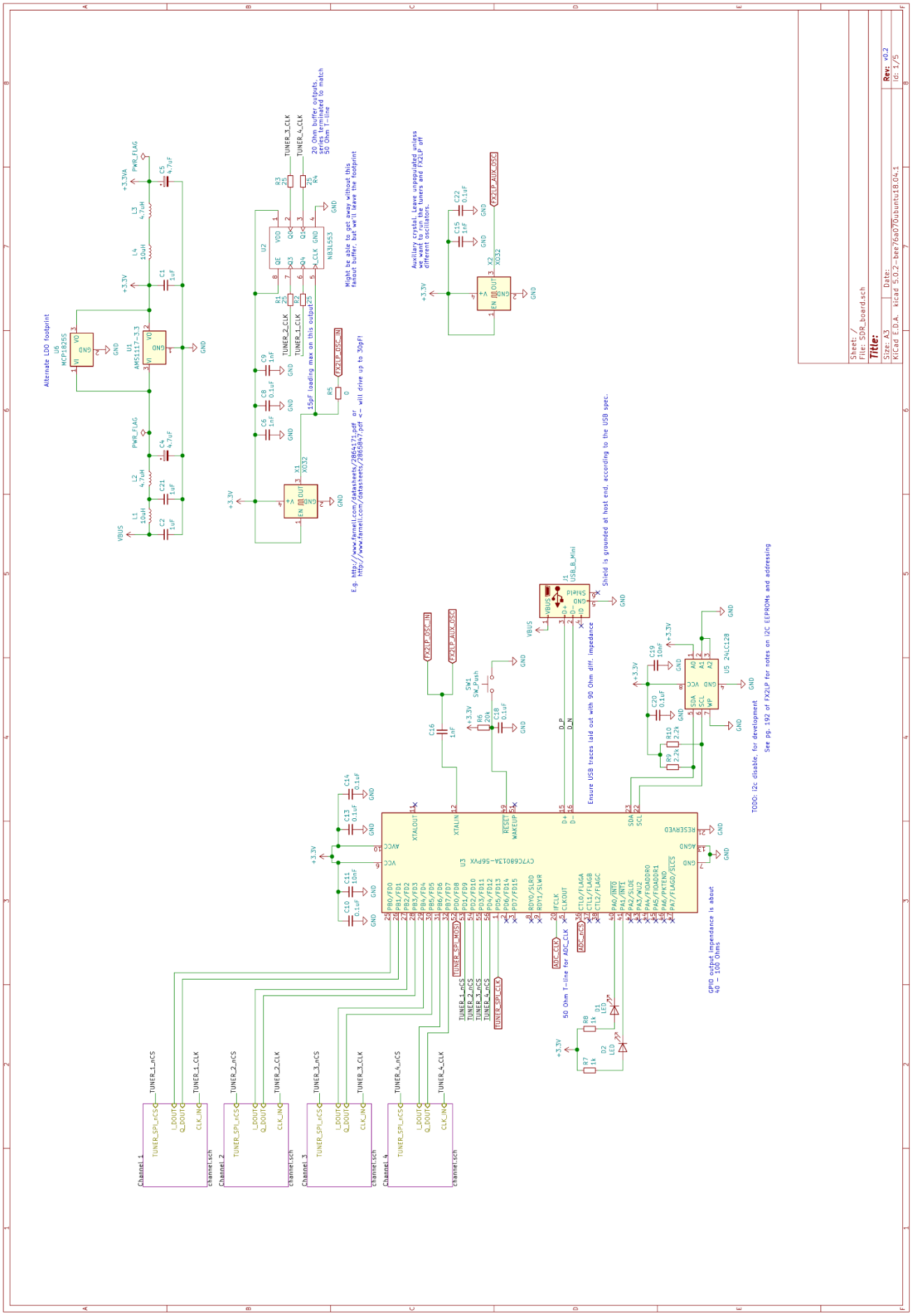
[16] Cypress Semiconductor Corporation, “Implementing an SPI Master on EZ-USB® FX2LP™ (AN14558),” 2014. [Online]. Available: https://www.cypress.com/file/125791/download. [Accessed 4 November 2020].

[17] RTL-SDR.com, “REMOVING THAT CENTER FREQUENCY DC SPIKE IN GNURADIO THE EASY WAY,” 3 February 2017. [Online]. Available: https://www.rtl-sdr.com/removing-that-center-frequency-dc-spike-in-gnuradio-the-easy-way/. [Accessed 7 November 2020].

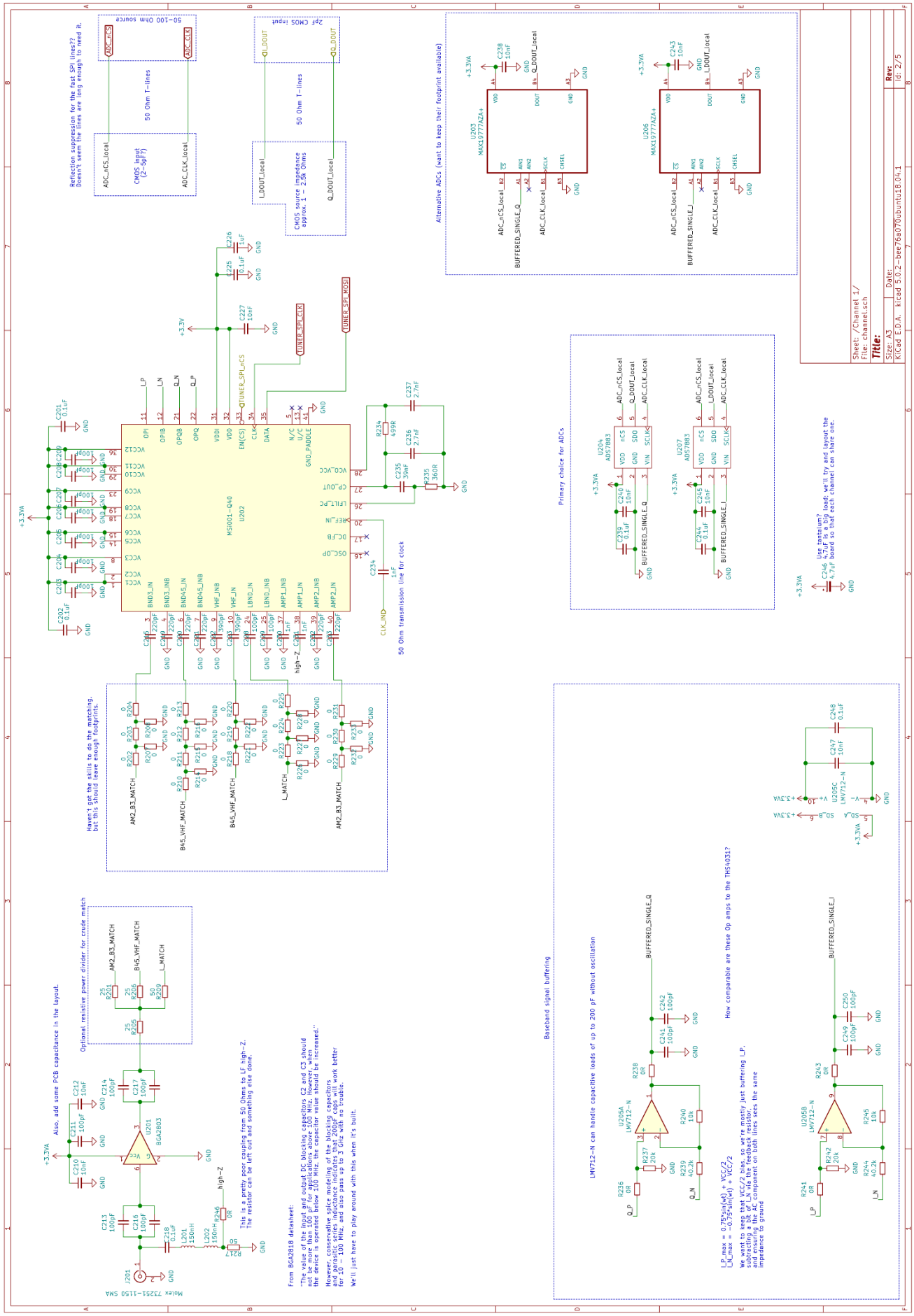
[18] Microchip Technology Inc., “MCP1825/MCP1825S 500 mA, Low Voltage, Low Quiescent Current LDO Regulator,” [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/22056b.pdf. [Accessed 6 November 2020].

Appendices

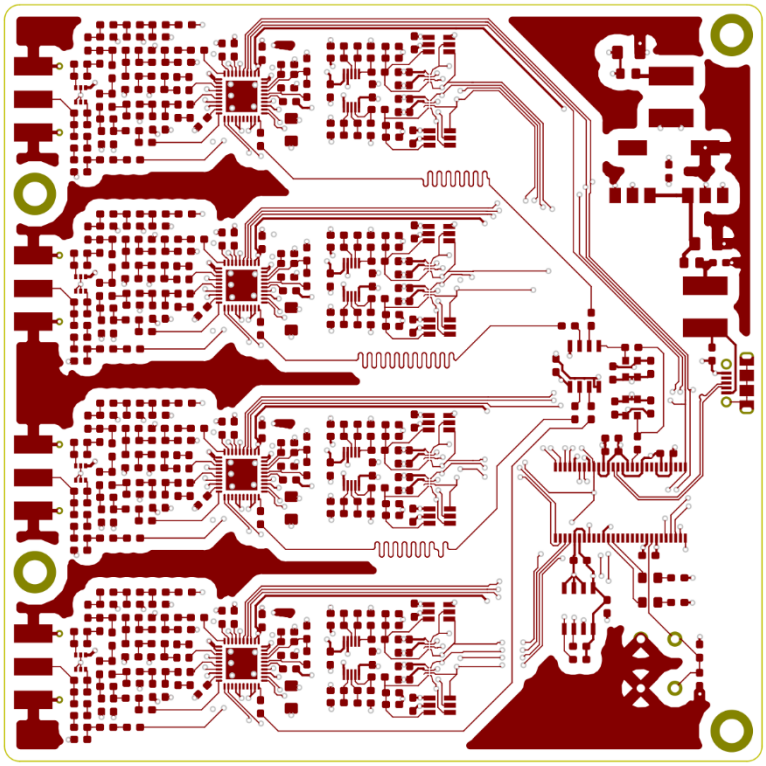
Appendix 1: Digital Circuit Schematic

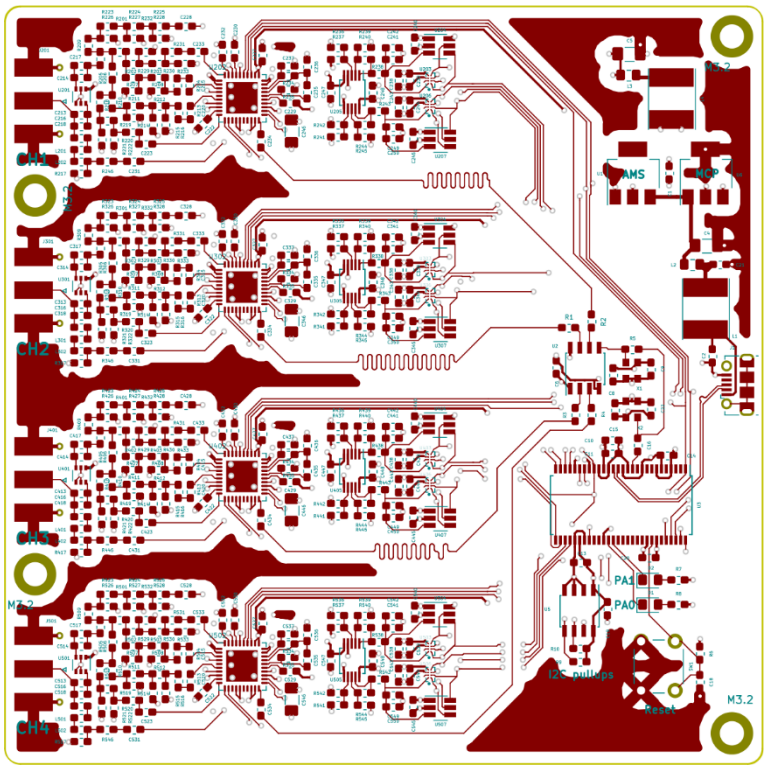


Appendix 2: RF and Baseband Schematic (for One Channel)

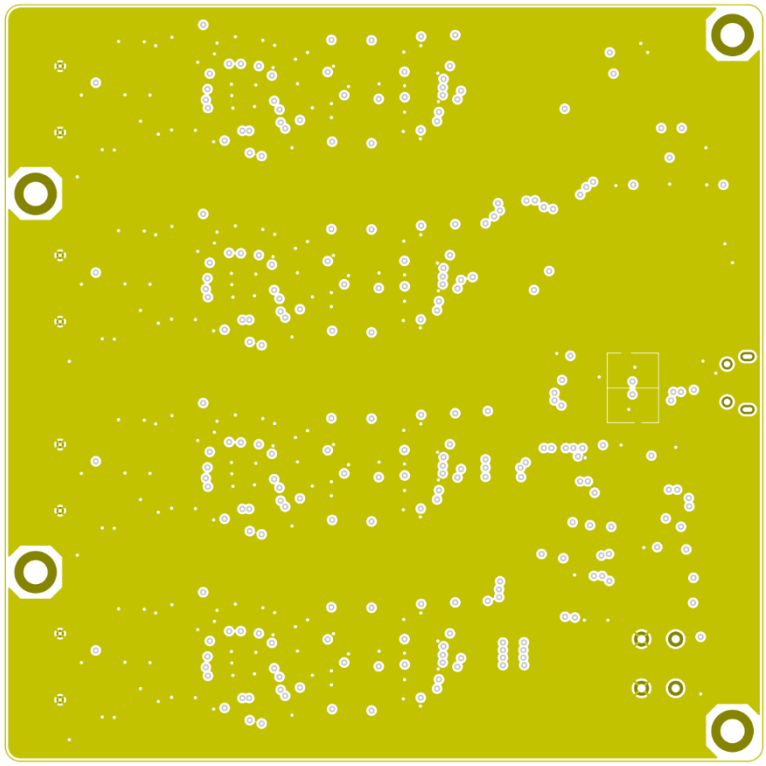


Appendix 3.1: PCB Layout (Layer 1—top with and without silkscreen)

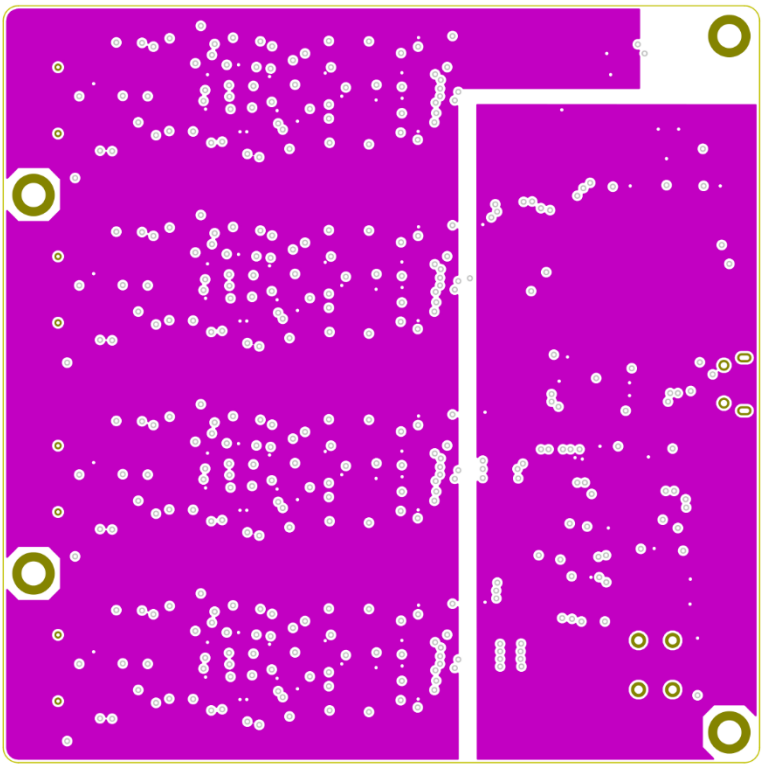




Appendix 3.2: PCB Layout (Layer 2—Top Side Inner)



Appendix 3.3: PCB Layout (Layer 3—Bottom Side Inner)



Appendix 3.4: PCB Layout (Layer 4—bottom with and without silkscreen)