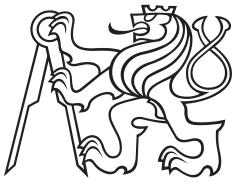
Bachelor Project



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Electromagnetic field

Processing of FMCW 24/120 GHz radar signals

and the architecture of FPGA-free fast data acquisition

Emil Jiří Tywoniak

Supervisor: Associate Prof. Přemysl Hudec
Field of study: Electrical Engineering
August 2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Tywoniak** Jméno: **Emil Jiří** Osobní číslo: **474266**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra radioelektroniky**

Studijní program: **Otevřené elektronické systémy**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Zpracování signálů FMCW radaru 24/120GHz

Název bakalářské práce anglicky:

Processing of FMCW 24/120GHz Radar Signals

Pokyny pro vypracování:

Prostudujte funkci radarového modulu Siliconradar SiRAD EASY 24/120/L7 s pracovními frekvencemi 24 a 120 GHz. Dále prostudujte metody zpracování signálů FMCW radarů pro pohybující se cíle a různé druhy frekvenčních čirpů. K předmětnému modulu navrhněte a realizujte jednotku zpracování signálů s ADC a DSP napojitelnou přes USB k PC. USB připojení k PC by mělo umožňovat i nastavování provozních parametrů radarového modulu z PC. Navrhněte a realizujte SW umožňující nastavování modulu, zobrazování a ukládání praktických měření v rovině vzdálenost-rychlost.

Seznam doporučené literatury:

[1] Suleymanov S.: Design and Implementation of an FMCW Radar Signal Processing Module for Automotive Applications, master thesis, university of Twente, 2016.
[2] Stove A. G.: Linear FMCW Radar Techniques, IEE Proceedings-F, Vol. 139, No.5., October 1992, pp. 343-350.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Přemysl Hudec, CSc., katedra elektromagnetického pole FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **31.01.2020** Termín odevzdání bakalářské práce: **14.08.2020**

Platnost zadání bakalářské práce: **30.09.2021**

doc. Ing. Přemysl Hudec, CSc.	doc. Ing. Josef Dobeš, CSc.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Acknowledgements

I thank the CTU for proving quite a fun playground.
I thank my colleagues at eForce FEE Prague Formula for advice and inspiration.
I thank my coworkers at esc Aerospace s.r.o. for advice and inspiration.
I thank the members of Digital Design HQ for osmotic learning of FPGA pitfalls.
I thank the free software and open source communities for sharing decades of progress with the world.
For this reason, the body of this work, that is, without its attached files, is licensed under a Creative Commons Attribution 4.0 International License as CC-BY. Read the license at `https://creativecommons.org/licenses/by/4.0/`

The attached files and related public git repository are mostly licensed under LGPL v3.0, for reasons of future compatibility with closed-source code provided by Silicon Radar GmbH. Other licenses may exist in subdirectories or headers of files and should be considered to override the LGPL license.

Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Emil Jiří Tywoniak V Praze, 14. August 2020
I hereby declare, that I have carried out the presented work individually, and that I have listed all used literature in compliance with the applicable laws and requirements. Emil Jiří Tywoniak Prague, 14. August 2020

Abstract

This thesis documents the design and performance of baseband sampling hardware, its data acquisition, and USB transmitting firmware, and PC-side reception software, designed for the Silicon Radar Easy FMCW radar module evaluation kit. Major design decision explanations are provided with comparisons to alternatives to provide context and ease further development. Range and velocity detection and I/Q imbalance equalization are briefly discussed, as well as the minimum system requirements for fully embedded signal processing.

Keywords: STM32, ADC, data acquisition, radar, FMCW, chirp, USB HS, libusb

Supervisor: Associate Prof. Přemysl Hudec
Faculty of Electrical Engineering, CTU, Technická 2,
166 27 Prague 6 - Dejvice

Abstrakt

Tato práce dokumentuje návrh, realizaci a funkcionalitu zařízení vzorkující základní pásmo radaru, jeho firmware k akvizici dat a přenos přes USB a programové vybavení pro osobní počítač. Systém byl navrženo pro testovací sadu FMCW radaru Silicon Radar Easy. Důležitá rozhodnutí provedená v procesu návrhu jsou poskytnuty se srovnáním s altenativami pro poskytnutí kontextu a usnadnění vývoje dalších zařízení. Detekce vzdálenosti a rychlosti a ekvalizace I/Q větví jsou krátce probrány, spolu s minimálními systémovými požadavky na zcela zabudované zpracování radarových signálů.

Klíčová slova: STM32, ADC, akvizice dat, radar, FMCW, chirp, USB HS, libusb

Překlad názvu: Zpracování signálů FMCW radaru 24/120 GHz — a architektura rychlé akvizice bez FPGA

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Radar fundamentals

Measuring distances and velocities of objects is crucial for controlling complex systems. With good reason does the mammal brain use an adaptively trained mix of heuristics and approximations to this end [6]. We could consider the earliest attempts at converting distance measurements, a typically empirical problem solved with a long piece of string, to a more scalable one through theory, to be t hose of Eratosthenes who trigonometrically computed the diameter of the Earth with an error of 1.4% using a short stick [7]. Since the third century BC, humanity has progressed somewhat, and has uncovered a handful of reliable laws, such as that of a finite speed of light [8], later generalized to a finite speed of information propagation [9]. This has taught us that we could measure distances from reflective objects if we could send a pulse of light, and measure its time of flight. Due to implementation difficulties beyond the scope of this work, this technique, named lidar, was developed only as a secondary tool derived from the radio frequency equivalent, radar, which uses the fact that light is a special case of electromagnetic radiation [10].

1.2 Frequency accuracy

We also have at our disposal incredibly precise methods of measuring frequencies and time intervals, using atom oscillations [11], to which we have affixed our very notion of time. More affordably and reliably, rather than every device being equipped a caesium oscillator, we can use off-the-shelf crystals with frequency errors under 10 parts-per-million, receive radio-distributed accurate time signaling, provided by Global Navigation Satellite Systems, lock onto these with a phase-locked loop either in hardware on software, calibrate the crystal, and eliminate the temperature-caused shorter-term frequency drift with some polynomial compensation. This gives us scalable, accurate frequency measurement on small devices.

1.3 Field-programmable gate array (FPGA) fundamentals

To understand the allure of FPGAs, we must first remind ourselves of the shortcomings of processors. A processor is a device which executes instructions from memory. It operates on them sequentially, a small number of operations at a time. It is a finite state machine, but from the perspective of a human, the number of possible states is infinite, it is very hard to guarantee any sort of latency. If one manages to guarantee some latency in software, it is always at the cost of mean latency, and performance loss. In short, processors are great at executing arbitrary algorithms in a sufficiently small number of clock cycles.

On the other hand, FPGAs behave exactly like hard-wired logic. Arbitrary digital systems can be built out of their configurable logic blocks, which are connected by a "fabric", a fancy name for a complex, undocumented crossbar-style digital multiplexer. They can be used for designs either asynchronous or synchronous to internal clocks and typically include

- fast distributed RAM
- large block RAM
- digital look-up tables for implementing arbitrary logical functions with efficient silicon area usage
- digital signal processing blocks for arithmetics
- digital input-output interfaces to interface with the world with high speed protocols

It is very common to combine the best of both approaches and synthesize simple processor cores on FPGAs, since processors fully consist from logic gates and some memory. Also, Systems on Chip (SoC), that integrate processor cores and FPGA fabrics are becoming popular for applications such as metrology or motor control.

However, at the time of writing, implementing algorithms in FPGAs is limited to niche applications. Deterministic latency is required in critical systems in aerospace[12], defense[13] and medical[14] applications, as well as high end media production equipment[15]. Some of the reasons for this are for example the lower complexity of developing embedded hardware with microcontrollers, and where such generic silicon lacks in performance or latency, application specific integrated circuits and application specific processors are more efficiently manufactured at scale, and can be tuned more finely for low power consumption and clock speeds.

1.4 Microcontroller fundamentals

Microcontrollers are simple integrated circuits with a processor core or multiple cores, on-board memory both volatile and non-volatile, and a set of peripherals typically mapped onto a bus or multiple. For simpler embedded devices, performance is a lower priority over reliability. For this reason, microcontrollers are most commonly programmed without any sort of operating system. If some sort of multi-threading is desired, real time operating systems (RTOS) which, unlike personal computer oriented systems such as linux, provide simpler means of switching contexts, for the sake of small code size and determinism. Peripherals often found on microcontrollers include timers, direct memory access (DMA) controllers, serial communication interfaces such as SPI and I2C, mixed signal blocks such as analog-to-digital and digital-to-analog converters, and debug functionality for loading and debugging firmware.

1.5 Range and velocity target detection

This section will use the formalism of Bandiera, Coluccia, Dodde, Masciullo, Ricci 2016 [16] with modifications for more explicit assumptions and description of multi-chirp operation.

The first chirp from a FMCW radar, such as that in our system, is transmitted as a frequency ramp:

$$f(t) = f_0 + \alpha t,\ 0 \leq t < T \tag{1.1}$$

f_0 being the carrier frequency, t time, T the duration of the chirp, T_c the time between chirp starts, α the chirp "rate". Let us enumerate chirps with k up to a chirp count of M:

$$f(t) = f_0 + \alpha\left(t - (k-1)\,T_c\right), \tag{1.2}$$

$$(k-1)T_c \leq t < (k-1)T_c + T \tag{1.3}$$

Let us call the time since the start of chirp k $t_k = t - (k-1)T_c$. The phase of the first chirp is simply frequency integrated over time:

$$\phi(t) = 2\pi \int_0^t f(\tau)d\tau = 2\pi f_0 t + \pi\alpha t^2 \tag{1.4}$$

For chirp k, in a voltage-controlled oscillator or digital equivalents, the phase is continuous, even at a jump in frequency, hence:

$$\phi[k](t) = 2\pi \int_0^t f(\tau)d\tau = 2\pi f_0 t_k + \pi\alpha t_k^2 + k\phi(T) \tag{1.5}$$

For clarity, let us ignore $k\phi(T)$, arbitrary phase offsets in chirps eliminate themselves later either way. The radar generates in its local oscillator a

real-valued signal, which is radiated by the TX antenna:

$$l[k](t) = A_L \cos(\phi[k](t)) \tag{1.6}$$

Assuming that targets are moving at constant speeds, the distance r and time of signal flight τ can be expressed:

$$r(t) = r_0 + vt = r_0 + vkT_c + vt_k \tag{1.7}$$

$$\tau(t) = \frac{2r(t)}{c(1 + \frac{v}{c})} \approx \frac{2r(t)}{c} \tag{1.8}$$

where c is the speed of light. We assume the targets's speed to be significantly slower than the speed of light from this step onward. The reflected signal is the delayed radiated signal, attenuated:

$$s[k](t) = A_R \cos\left(\phi[k]\left(t - \tau(t)\right)\right) + w(t) \tag{1.9}$$

where $w(t)$ is AWGN. This is downmixed with the local oscillator signal, and the local oscillator signal phase-shifted by $\frac{\pi}{2}$, to generate the I/Q signals. At this point, inaccuracy is introduced in the form of a gain error g_e between the I and Q signal, as well as an error in the phase shift ϕ_e of the local oscillator. The local oscillator I/Q signals have the following forms in each chirp:

$$l_I[k](t) = \frac{A_L}{\sqrt{2}} \cos(\phi(t)) \tag{1.10}$$

$$l_Q[k](t) = g_e \frac{A_L}{\sqrt{2}} \sin(\phi(t) + \phi_e) \tag{1.11}$$

Downmixing consists in multiplication of (assumed harmonic) signals in time-domain, and low-pass filtering all resulting terms at frequencies other than that of the difference term. Assuming an ideal low-pass, after this mix we arrive at the following "baseband" signal pair:

$$b_I[k](t) = 2a' \cos(2\pi f_T t_k + \theta) + n_I(t) \tag{1.12}$$

$$b_Q[k](t) = 2a' g_e \sin(2\pi f_T t_k + \theta + \phi_e) + n_Q(t) \tag{1.13}$$

where $f_T = 2\alpha\frac{r_0}{c}, \theta = 4\pi\frac{f_0(r_0 + vkT_c)}{c}, a' = \frac{A_L A_R}{4\sqrt{2}}$ and $n_I(t)$, $n_Q(t)$ are possibly correlated low-pass filtered white Gaussian noise, but assumed uncorrelated zero-mean for the purposes of I/Q calibration. f_T also has a Doppler-effect element of $2\frac{v}{c}f_0$, not expressed above, however, with our FMCW parameters, this turns out to be smaller than the distance-delay element of $2\alpha\frac{r_0}{c}$ by a factor of 10^{-3}. We are going to be extracting the velocity data by using the difference in range over our multiple chirps instead. At this point, let's assume we are receiving a perfect baseband signal, that is, $g_e = 1, \phi_e = 1$. We can now see that the baseband signal pair consists of a real and imaginary part of a complex-valued signal:

$$\hat{b}[k](t) = 2a' \exp(2\pi \mathrm{j} f_T t_k + \theta) \tag{1.14}$$

Now we sample the baseband signal pair at a sufficiently high frequency of f_s, the sampling period being T_s. Afterwards, if we perform a discrete Fourier transform (DFT) of each chirp along the sampled chirp time (t_k) axis, we can detect a peak at f_T. Let's hide uninteresting multiplicative constants into a'' and for simplicity's sake perform a continuous Fourier transform of one period:

$$\hat{b}[k](\omega) = a'' exp\left(\mathrm{j}k\left(4\pi f_0 T_c \frac{v}{c}\right)\right)\delta\left(\omega - 2\alpha\frac{r_0 + kT_c v}{c}\right) \tag{1.15}$$

If we now perform a DFT along the chirp index axis:

$$\hat{b}[\Omega'](\omega) = a''\delta\left(\Omega' - \left(4\pi f_0 T_c \frac{v}{c}\right)\right)\delta\left(\omega - 2\alpha\frac{r_0 + kT_c v}{c}\right) \tag{1.16}$$

periodic with regards to Ω'. That being said, in implementation, the untransformed signal is discrete along both axes; furthermore, the FFT only gives us a single period of the true DFT. With trivial manipulations, we find that we can unambiguously detect targets at distances of up to $r_{max} = \frac{cf_s}{4\alpha}$ and velocities of up to $v_{max} = \frac{cf_0}{4T_c}$. The accuracy of r,v, from DFT characteristics, becomes $\Delta r = \frac{2r_{max}}{f_s T}$, $\Delta v = \frac{2v_{max}}{M}$. We can see, that theoretically, if we let the chirp count and sampling frequency approach infinity, we can perfectly resolve distances and velocities of arbitrarily far-away objects moving at constant subluminal velocities, which fits our intuitions of "the better the hardware and the more information, the better the resulting data". Curiously, this seems to violate the Fourier-transform logic behind the Heisenberg uncertainty principle; however, we are not using a single measurement here, but an infinite series of measurements.

1.6 The USB Protocol

The USB protocol[18] provides guaranteed in-order delivery between a bus host and a connected device. An endpoint is a virtual construct, the functions of which are typically implemented in hardware to various degrees. The STM32 USB peripherals, for example, treat endpoints as configurable FIFOs where packets are stored and sent or received by the USB core logic, without involvement of the processor core, triggering interrupts for the processor to come pick up its received data or queue more data for transmission. Personal computer peripherals treat them similarly, but an application does not have to work directly with FIFOs, since such operations are handled by the operating systems in privileged modes by kernel modules (DMA) and drivers. In USB protocol terminology, raw data flows through "pipes", which are virtual connections between two endpoints. The developer uses this terminology, opens a pipe and request data on it without worrying about packet structure. There are two types of pipes: unidirectional "stream pipes" directed by either the host or device, and bidirectional host-directed "message pipes". The directions are called, from the perspective of the bus host, IN and OUT.

The protocol has multiple transfer types, which occur between endpoints. Bidirectional endpoint 0 is reserved for control transfers, which convey the state of the bus, including data requests, over message pipes. The transfers which use stream pipes to convey data are interrupt, isochronous, and bulk transfers. Interrupt transfers are non-periodic and handled with higher priority than the rest, isochronous transfers are periodic and get assigned a special portion at the start of USB data frames, and bulk transfers are designated for large amounts of data. In reality, all but control and bulk are considered deprecated. Isochronous transfers seem to have been designed with extremely simple devices with application-specific integrated circuit controllers, and have not been designed to scale well with bandwidth increases such as USB microframes. Isochronous transfers have no error detection. Bulk transfers have the limitation of being 8, 16, 32, or 64 bytes on USB-FS devices, 256 or 512 on USB-HS, and up to 1024 bytes in USB-SS.

When a device is connected to a host, the presence of the device is detected as a load on the 5V VBUS. For this reason, the capacitance of a USB device at the time of connection is limited to a range of 1 µF to 10 µF. It is not clear from the standard whether capacitively loading the VBUS line further after connection is acceptable.

The USB protocol provides multiple data speeds. Most ubiqutously implemented in desktop peripherals is Full Speed, which provides 12Mbps and has rather lenient accuracy requirements. USB 2.0 adds High Speed at 480Mbps, a bandwidth sufficient for carrying live video.

1.7 Future of radar

Among the most notable directions the development of radar technology has been taking is certainly accessibility. Using modern MMIC (microwave integrated circuit) production capabilities, the cost of designing and bringing to market a device with an oscillator, rat-race and mixer has declined. Passenger cars are starting to use radars as a part of their sensor suite for collision avoidance, paving the road for full autonomy. Medical uses of radar have been proposed, such as breathing rhythm detection.

Significant advances have been made through advanced digital signal processing work on beam-steering and beam-shaping, in conjunction with the development of modern RF communication standards. Increases in resolution beyond the traditional wavelength limitation have been achieved. MIMO (multiple input, multiple output) systems operate at the ever-increasing limits of embedded computing, using mathematical tricks to reduce ambiguity and inaccuracy, at the cost of computational copmlexity.

Last but not least, there is an increasing potential of applying machine learning to object detection. Detecting and classifying objects in video feeds is a highly popular and productive field of study, greatly surpassing classical algorithms in adaptability and increasingly also in low computational complexity once trained.

Chapter 2

Requirements

The quantifiable requirements for the device were as follows: The device was

Unambiguous distance	100 m
Chirp time	100 µs
Chirp count	128
Sampling frequency	5 MHz
Voltage range	0–3.3V
Bit depth	≥12-bit

Table 2.1: Requirements for the device

to be connected to a computer for displaying the range-velocity plot via USB. In order to sample two 12-bit channels at 5 MHz, a communication bandwidth of 120Mbps is required. For this reason, USB High Speed was selected. As for the form factor, the morpho connector layout, as can be found on STM32 Nucleo boards, has been replicated, to fit the Silicon Radar Easy evaluation kit.

Chapter 3

Design

3.1 Hardware

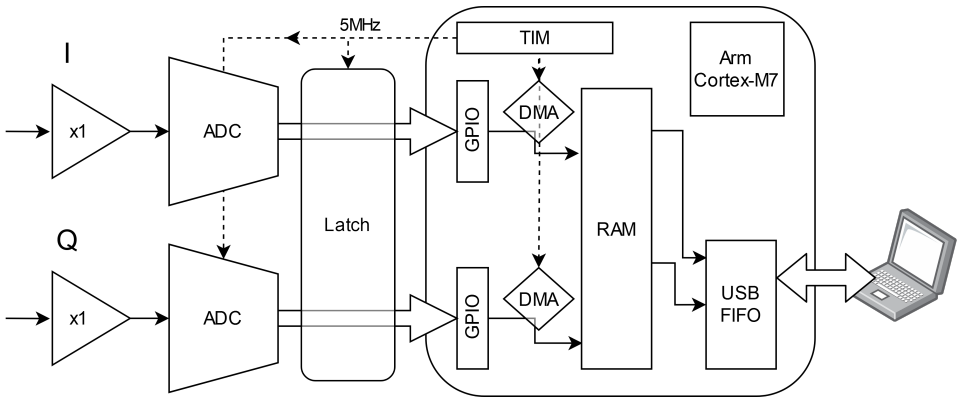


Figure 3.1: Hardware architecture

3.1.1 Inputs

The device features two 50 Ω BNC inputs, coherently sampled, to be used as I/Q feeds or separate input channels. These are AC-coupled, DC-biased to the 2.048 V ADC voltage reference. This allows for an input voltage range of 0–4.096V, that is, $2.048\mathrm{V}_{pp}$. In the next step, they are buffered, and filtered, before being sampled by the ADC. This is done to minimize the internal capacitance dynamical switching of the ADC from generating noise on the signal. The evaluation kit's baseband I/Q outputs are biased to 1.65 V, and clipped to $\langle 0; 3.3 \rangle$V. Therefore this output range is fully covered by our input range.

The operation amplifiers, used as unity gain followers, were initially chosen to be LMH6628MAX. However, since these are grossly not rail-to-rail, and would require adding a negative voltage source to prevent clipping or distorting the signal, they have been replaced in v1.1 by a single rail-to-rail TLV3542ID.

The latter is designed for lower voltage ranges, with specification equal or superior to the LMH6628MAX. Note that LMH6628MAX amplifiers were initially chosen due to being explicitly recommended by the ADC12010 datasheet, are in fact dual channel, but on v1 there are two on the board with their second channels ignored, due to the crosstalk of the channels being above the noise floor of the ADC. The TLV amplifiers have sufficiently low crosstalk in order for a single one to be used.

3.1.2 USB

Since a USB High Speed connection was required, to simplify the design, the STM32F733ZI microcontroller has been selected for its on-board USB High Speed physical layer. Other parts require an external interface integrated circuit connected over the "UTMI Low Pin Interface", the data and control lines of which are mapped out to seemingly random pins on STM32s, and require either a second on-board crystal, or careful use of the PLL output functionality of the STM32.

3.1.3 Sampling

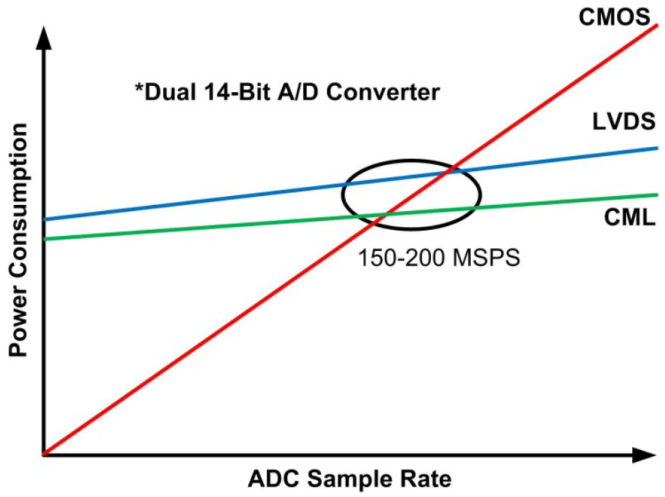


Figure 3.2: Voltage vs. current-driven signaling in ADC outputs [1]

Let us first consider the output bus, on which the digital data is to be carried to STM32F733ZI, the chosen microcontroller. If we look at available parts that meet our sampling rate and bit depth requirements, we find that the following options: classic voltage-driven parallel, serial such as I2C and SPI including its variants such as QSPI and Microwire, and high bandwidth parallel low voltage differential signaling (LVDS) layers such as JESD204. LVDS allows for lower power consumption at ridiculously high sampling rates, as demonstrated in Figure 3.2. There are no LVDS transceivers on the STM32

line of microcontrollers, an external LVDS physical would be connected in parallel, and the LVDS would be run for only 2 cm on the board, defeating its purpose. On the STM32 F7 series, for this application, I2C is out of the question with its 1Mbps [17] bandwidth limit, as well as SPI, with its limit of 54Mbps. At 54Mbps, getting a 12-bit ADC value would take 222 ns, which not just exceeds a reasonable timing window within the period of the 5 Msps sampling rate, it outright exceeds the period. For these reasons, a parallel bus has been selected. The STM32F7 GPIOs propagate input voltage to the internal registers at speeds exceeding 100 MHz.

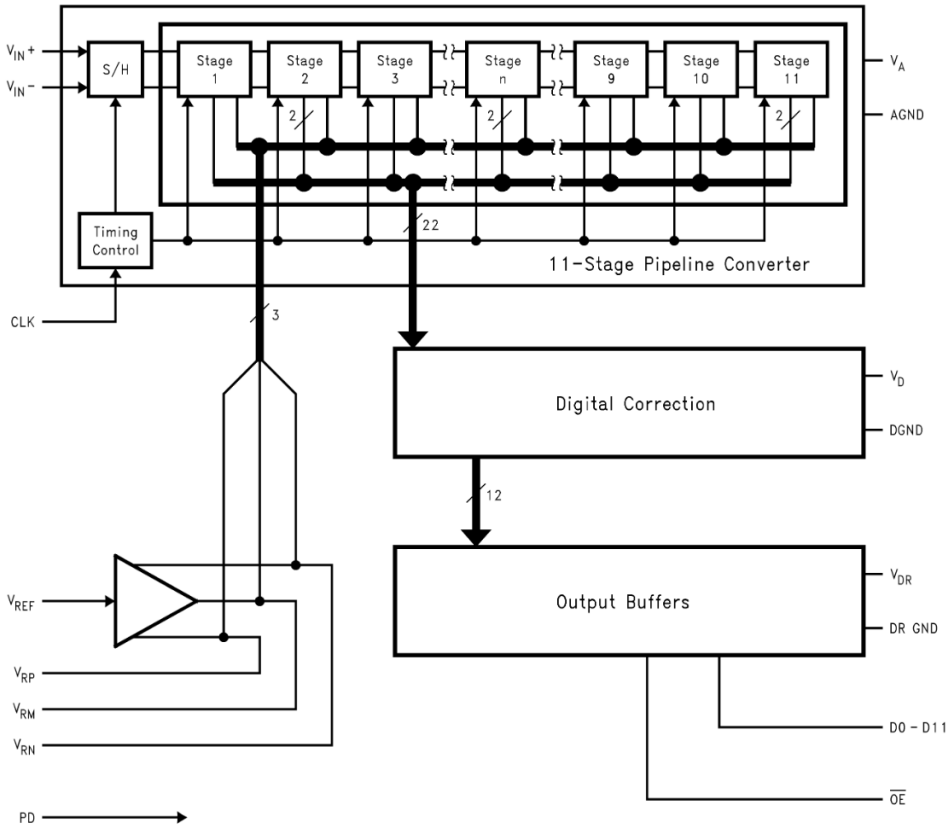


Figure 3.3: ADC12010 block diagram [2]

After the output was decided, a part that fulfills requirements was selected, the ADC12010, a 10 MSPS single channel, differential input, general purpose ADC. [2] It features a pipelined architecture, converting input voltage to 22-bit with an internal sample and hold input stage, built-in unspecified digital correction to 12-bit, and a non-latching digital output buffer. The output is again buffered with a set of high-speed non-inverting octal buffers. Note that in the design, ACT-series drivers have been used in error, operating below their specified 4.5V-5.5V power supply range – LVC series parts should be used instead, i.e. the 74LVC574ADB. Missed ADC codes may occur, since propagation delay is unspecified outside the operating voltage range, or rather,

propagation isn't guaranteed at all. At 3.3V, a 3.5ns propagation delay is specified for 74LVC574Axx. Similarly, LVC-family Schmitt triggers are used to provide a high slew rate transition on clock inputs of the ADC and latches.

3.1.4 Power consumption

With its typical power draw of up to 500 mA, this device violates the USB standard, due to unimplemented link power management and excessive capacitive load on VBUS. ADCs are specified to draw 160 mW each at 10 Msps and 25 mW on power-down, which is unimplemented due to a possible bug in the microcontroller. Three different STM32F733ZI have been observed to be unable to set or read correctly configured GPIO pins on some of the banks, luckily not GPIOE and GPIOF, which are used as parallel buses for ADC data input. Power efficiency is also reduced by the use of a linear 3.3 V regulator, selected for simplicity of design and lower noise over switch-mode supplies.

3.1.5 Physical realization

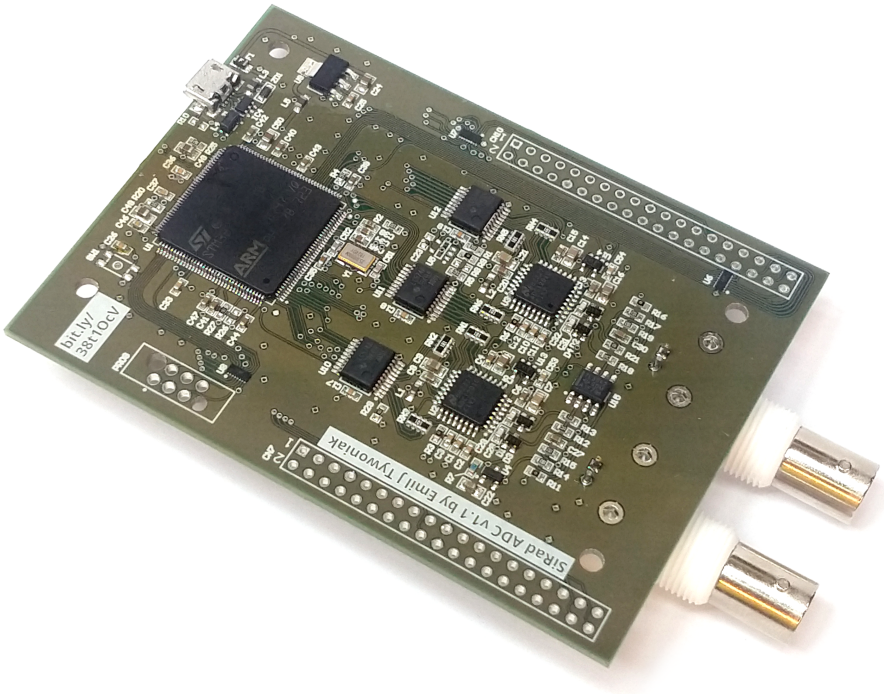


Figure 3.4: Hardware v1.1

As simple as the design is, it has been designed as a four-layer board with care taken to minimize signal crossing and a an internal all-GND layer as well as an internal all-power layer. USB-HS was length-matched to within

2 mm, easily meeting the maximum skew of 100 ps[18]. All layout guidelines have been followed as specified by the documentation for the ADC and microcontroller. Ferrite beads were used to minimize high slew rate current changes from the digital driving circuitry from causing propagation of voltage spikes. Digital connections to the system, with the exception of USB, have been protected with EMI5208MU eight-channel pi-filters, providing 100 Ω in-line DC resistance, EMI filters with a 250 MHz cutoff frequency, and ESD protection. Note that due to a layout error in v1 hardware, a surge pulse isn't forced to travel through the surge protector. USB has been protected with the USBLC6-2 ESD and surge protection package, optimized for USB 2.0. Basic USB EMI/EMC reduction has been achieved with a standard common-mode choke.

3.2 Firmware

The full firmware for the self-developed hardware is provided in the attached git repository under `my SW/v1`. It can be built on any platform with the ARM GCC toolchain with no EABI. A Makefile is provided. It can be imported into any IDE which supports Makefile projects. Peripherals have been configured to desired settings by the STM32CubeMX free utility, which emits C code complete with a linker script and startup script. Included is detection of functionality conflicts, for which the user would otherwise have to meticulously read the entirety of each relevant section of the reference manual. The generated code uses the manufacturer-provided hardware abstraction layer (HAL), a maximalist set of libraries provided in source file form. It can also provide middleware code, such as the USB Device library, or CMSIS DSP modules.

3.2.1 USB communication

This device implements a custom communication protocol, outside of the range of standard device classes. The USB HS peripheral is configured to behave as a device, and in its internal memory (not mapped into processor-accessible space), each used endpoint gets an allocated FIFO. A single-purpose DMA is provided for pushing Tx packets into the FIFOs or pulling Rx data out without the involvement of the processor core, but despite all instructions in the reference manual being followed, it would not copy data and the address of the datum it was to copy never incremented. There is minimal documentation for this feature, likely because the USB-HS peripheral itself is not often used by STM32 developers. This means, that the peripheral has to be served by the processor.

The configurations of the USB device can be found in `Src/usbd_custom.c` The vendor and product ID (VID, PID) have been left to ST Microelectronics' default values, since neither the designer nor his institution has a vendor

ID. It can not be left unsaid, that USB is not a free standard, and the USB Implementers' Forum forces individuals and small companies to violate the standard by using VID/PID pairs they do not own, since they can not afford the fee for owning the entire PID range for a single VID. The device has a single USB interface with three endpoints, all of these sized 512 bytes. The majority of the descriptor structs are left as-is from the USB HID example code provided by ST, only modified to accommodate the higher power draw of the device, and its endpoints.

Recorded ADC samples are sent over the ADC_I and ADC_Q endpoints. There is no header or packing being done. These endpoints directly send the 16-bit half words, serialized into bytes. There simply isn't enough time to spare on using macros to convert to a protocol-specific byte order either. There is also a configuration OUT endpoint, and a notification IN endpoint. Once powered on, the device waits for an empty string on the configuration endpoint. After this, it waits for a rising edge on the external trigger pin. Then, it sends a string `"chirp:0xSamples;0xChirps"`, notifying the PC to request the correct amount of data, enables timer output compare channels, and lets the DMA transfer complete callbacks request to be written into USB FIFOs until the desired amount of samples is transmitted. There is no stall-prevention.

3.2.2 Clock generation

The clock for the ADC is generated by TIM1, a timer peripheral. In the PWM or output compare mode, the timer counts up to the ARR value and resets and continues incrementing from zero. The output GPIO pin is set once the timer's value hits the value in the capture/compare register, and reset when the timer resets. The prescaler on TIM1 is set to 0, so its auto-reload and output compare (corresponding to period and on-time) can be changed in rather coarse increments of one period of the 216 MHz core clock. The ADC is clocked on the rising edge of the timer output pin, and the flip-flop buffer (called Latch in the system architecture diagram) captures the ADC digital outputs on the falling edge of the timer output pin. This means, that manipulating the duty cycle of the timer changes the time available for the ADC to provide a stable output code. This comes with the limitation, that the ADC used on this device is specified at clock duty cycles of 30-70%. Adding a second output with a different phase on the same timer to trigger the flip-flop buffer is not a problem, since a single timer peripheral can have multiple compare channels. It should be noted, that even though the ADC digital outputs are set very quickly after the clock rising edge, the pipelined architecture of the part takes six clock cycles for the sampled voltage to convert and are presented on the output parallel bus.

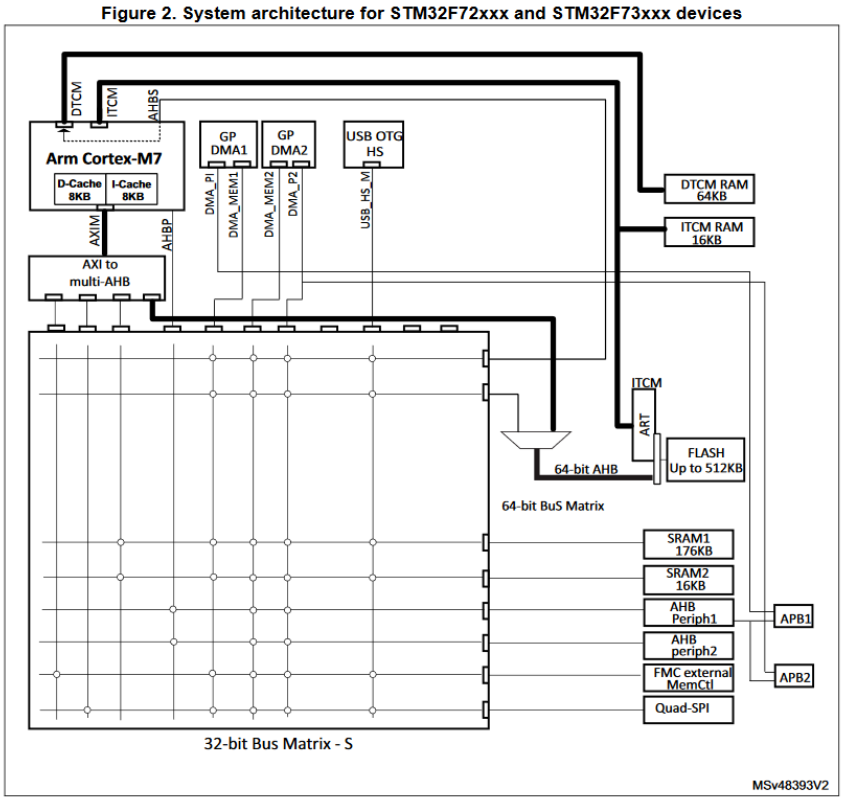


Figure 3.5: System architecture of the used device

3.2.3 Digital readout

Two DMA controllers are provided on the microcontroller, DMA1 and DMA2. They operate on a subset of the same memory mapping as the processor core. Each controller can serve one "stream" at a time, and each stream has multiple "channels". Each controller can access a different set of device section, as shown in the system architecture diagram in figure 3.5. DMA transfers are started on request. This request can be generated by writing into the correct registers with firmware, or by a peripheral generating an event that is set up to trigger the request. The design uses the latter. Two timer output compare channels, one of which is controlling an output pin generating the clock as described earlier, the other only in "timing" mode. Together, they generate DMA requests for two DMA channels. The DMA channels are configured to the same stream, because each combination of controller, stream, and channel has a typically unique set of peripherals it can access, some of which are even empty (that is, these can only perform memory-to-memory transfers). To summarize:

- Controller is DMA2 for both ADC channels, because it has access to AHB1 peripherals such as GPIOs
- (Stream, channel) is (1,6) for ADC I and (2,6) for ADC Q, since these

are requested by TIM1 CH1 and TIM1 CH2 respectively

The entire multi-chirp sequence can not be recorded into the microcontroller's 256kB SRAM, so the device has to continually write to, read out of, and overwrite a set of buffers. For high data flow applications, such as video rendering and transfer, the "double buffering" technique is often used. Each data source gets two buffers. It writes at its own pace into one of them, and once done, it starts writing to the other, while the reading process operates on the full buffer. This means that reading always happens on the same size of data in the same place, which is practical for transferring large amounts of data. Compare this with a ring buffer, where the reading process is continually trying to catch up with the writing process, and has minimal latencies. On the other hand, this tends to create a large amount of smaller reading requests, and requires handling of the wraparound. Wraparound is the moment the writing or reading process reaches the end of the buffer and has to continue from the start. The device used also has a circular mode, which handles this in hardware.

3.3 Software

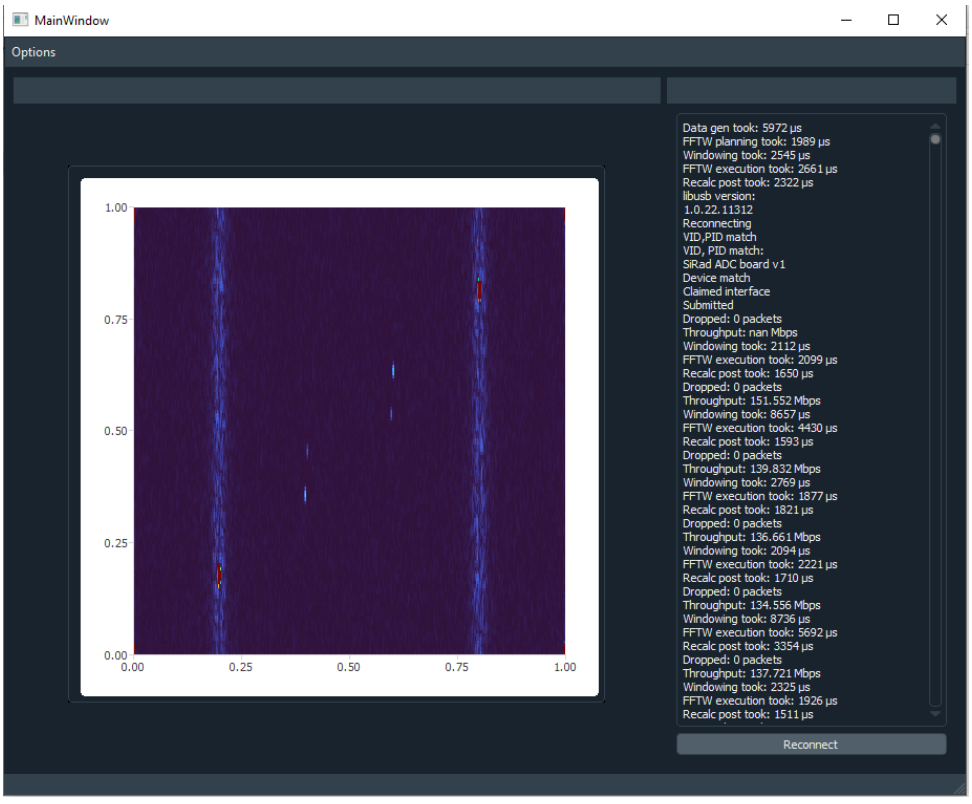


Figure 3.6: Custom developed viewer application, showing the 2D spectrum of a 1MHz, 100mV (peak-to-peak) signal sampled at 5MHz on one of the device channels

3.4 User interface

Qt is a feature-rich multi-platform framework for creating visual applications. It is written in C++, however, many language bindings exist for it, including Python, Rust, Go, Java, and C#. C++ was selected due to the author's prior experience and interest. The user should be warned that the application is experimental work, and is not very extendable, configurable, or memory-safe.

3.5 USB communication

For portable application, the most widely used low-level USB connection library is libusb. Its kernel-level driver is available by default on both Windows and Linux. This means that a user does not have to install new device drivers, however, on Windows, the driver has to be assigned to the USB device via either Device Manager (not recommended, since Windows hides driver filenames in the Update Driver driver selection dialog), or the Zadig utility [19]. In userspace, functionality is provided by a dynamically-loaded library binary file, and a single C-header. Wrappers exist for libusb to be used with languages other than C/C++. The libusb library supports all major functionality from USB 1.0 to USB 3.1.

The API provided by the header file has two sets of functions: the synchronous (blocking) and asynchronous modes. Synchronous is recommended as the default for most applications, as it is simpler to understand and implement. The asynchronous interface is superior in performance by allowing multiple bulk requests to be processed simultaneously. Even though the USB protocol has only one data line, this means, that the host side is always ready to receive data. In testing, naive use of the synchronous interface can not use the entire available USB-HS throughput, only receiving up to 120Mbps. This could be sufficient for the radar purposes of the device. However, after porting the application to the asynchronous I/O, running a handler in a loop with a delay in a separate thread inspired by Qt code in Jakub Vodseďálek's thesis [20], samples are read at the 160 Mbit/s speed, equivalent to the speed at which the samples are being recorded. This means, that the system can be used for continuous 5MHz sampling.

There are multiple Windows backends for libusb: WinUSB by Microsoft (not to be confused with WinUSB, the portable Windows bootable drive creation tool), libusbK, libusb-win32, and usbdk by Red Hat. [21] Each of these comes with a set of limitations. WinUSB is closed source, the libusbK project has not received updates since 2014, libusb-win32 only supports libusb version 0.1 (i.e. no asynchronous interface), and usbdk is not included as a suggested libusb-compatible driver in Zadig, despite being supported as a libusb backend. An interesting feature of usbdk is its ability to hide a device from Windows, once assigned to a particular device [22]. It is also the most actively worked-on libusb backend at the time of writing. For unknown

reasons, libusbK is the only backend tested to function with the provided device.

3.6 Data processing

The received data are interpreted as a 2D array of complex double-precision floating-point values. This array is windowed with a Blackman-Harris window to reduce aberrations in the spectrum, using generic code from Stack Overflow [23]. After this, a two-dimensional discrete Fourier transform is performed on the array, the log of absolute value of which is displayed in a Google Turbo colormap [24] window. The FFT library used is FFTW. This data processing sequence takes under 20 ms on a modern personal computer.

3.7 Performance evaluation

3.7.1 ADC characteristics

From specification of the original Nucleo F303 solution and specifications of the external ADC on the custom design, typical values are provided in Table 3.1. With both input channels left floating, low noise of the sampling system has been observed, as can be seen in the raw dump of the 2D spectrum plot in Figure 3.7. Compare with the bright, easily discernible signals in Figure 3.6. A shortcoming of hardware version 1.1 is its strong attenuation of low frequency signals - even going from 1 to 2MHz produces a visible increase in signal strength. This can easily be fixed with the input AC-coupling capacitor value being reduced.

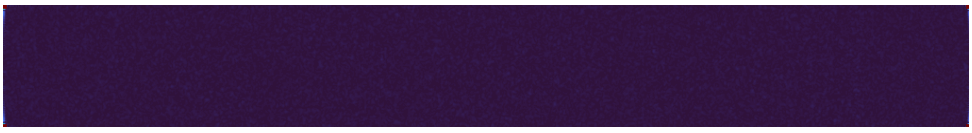


Figure 3.7: 2D spectrum plot of a batch of samples, input channels left floating, image unscaled

3.7.2 Comparison with existing solutions

This section introduces a few off-the-shelf data acquisition systems, to provide context for the capabilities of the SiRad ADC v1 board.

The ubiquitous **Saleae logic analyzer clones** [25] [26] [27], popular for debugging digital buses, provides 8 channels, sampled at up to 24 MHz over a 192Mbps USB-HS connection. It holds no firmware, as it is loaded with a program from the host PC. Its microcontroller is a low-cost legacy 8051 part

System	Nucleo F303	SiRad ADC v1
Nominal bit-depth	12-bit	12-bit
Offset error	±1 LSB	-0.1% FS
Gain error	±3 LSB	±0.2% FS
Differential linearity error	±1 LSB	±0.3 LSB
Integral linearity error	±1.5 LSB	±0.5 LSB
Effective number of bits	10.8 bits	11.4 bits
Signal-to-noise and distortion ratio	68 dB	70 dB
Signal-to-noise ratio	67 dB	70 dB
Total Harmonic Distortion	-76 dB	-86 dB
Intermodulation Distortion	unspecified	-75 dBFS

Table 3.1: Comparison of specified sampling performance, typ. values

with scarce peripherals. It has no detection of dropped frames, that could be caused by the host PC's USB controllers introducing latency, and its data buffer is, at most, 4 KiB+8.5 KiB. Its core runs at 48 MHz maximum and as such can not be used for reliable logic analysis, or radar baseband sampling, unless dropped frame detection is built into PC software.

Beginner radio frequency enthusiasts' choice, the **RTL2832U**, is an integrated solution for DVB-T broadcast television reception. It is bundled with various external tuner ICs, which it controls. It samples the analog downconverted intermediate frequency signal, further downconverts it digitally into an I/Q signal pair, which it downsamples and serves over USB High Speed. Its theoretical maximum sampling rate is 3.2 Msps, but typically only 2.4 Msps can be achieved due to packet loss. With 8 bit per I and Q channel each, the theoretical maximum USB throughput of the device is 51.2Mbps. Packet loss can be checked with rtl_test, a utility in the librtlsdr toolkit [28], and implemented in host code. However, no dropped frame detection has been found in the rtl_sdr backend used by utility scripts and applications, or GUI applications using various backends such as SDR# – in operation, it is assumed one has tested the connection and sampling rate used, and that these conditions remain the same.

Modern digital oscilloscopes operate at bandwidths ranging from 50 MHz to 100 GHz [29]. However, they are not designed to stream captured data in real time at their specified sampling rates to an external machine. Higher-end models certainly are built on full-featured desktop computers with ample processing power, but are sold as monolithic devices that are highly impractical for use outside of the laboratory. However, their fast data acquisition core components can be found as PCIe cards or modules for PCIe-like systems in the price range of 1300-30000EUR. A subversive exception to this is **ScopeFun** [30], an open source oscilloscope with two 10-bit channels sampled at 250 Msps each. If only one channel is used, it delivers 500 Msps over its USB 3.0 SuperSpeed, saturating the bus 5Gbps bandwidth. Its functionality relies on a Xilinx Artix-7 FPGA with 512MB RAM, and an external Cypress FX3 USB 3.0 controller.

System	Sampling rate	Throughput	ADC	Cost
Saleae clone	24 MHz	192 Mbps	no	10 EUR
RTL-SDR	3.2 MHz	52.2 Mbps	8-bit	16 EUR
SiRad ADC v1	5 MHz	160 Mbps	12-bit	70 EUR*
ScopeFun	500 MHz	5 Gbps	10-bit	660 EUR / 300 EUR*

Table 3.2: Comparison of acquisition systems. * cost of manufacture in quantity of 5

3.7.3 Required processing for on-board 2D FFT

Detecting targets from M chirps (of N samples each) on an embedded system requires:

- pre-processing
- M x FFT of size N
- transposition in RAM
- N x FFT of size M
- post-processing

The Cortex-M7 core in the STM32F7 series implements the Armv7-M instruction set as a part of the Armv7E-M microarchitecture, which features a two-way superscalar architecture with integer SIMD (single instruction, multiple data) operations on integers among its digital signal processing capabilities. It also has access to a floating-point unit (FPU) implementing the FPv5 instruction set extension.[31] We can take a look at how this works in practice in a report by ST Microelectronics for Cortex-M7 with a comparable device, the STM32F746[3]. Its processing power differs from the STM32F733 part used in this device only in that it has half the instruction and data cache. This benchmark uses ARM CMSIS-DSP library functions, which are considered the canonical implementations of mathematical functionality, making use of SIMD capabilities, presumably optimally. The results show that a Cortex-M7 is able to compute a 1024-point FFT in 339 µs in 32-bit floating-point representation, and in 265 µs in the Q15 fractional format which does not require the FPU, see Figure 3.8. On the slightly lower clock speed, non-superscalar Cortex M4 part, a 50% increase in cycle count per operation has been observed 3.9. Floating-point is preferable due to its high possible dynamic range and lower error propagation risk.

From the requirements for this device, $N = 1024$ and $M = 128$. The on-board device will take at least 43 ms to compute our Mx N-point FFT. If we extrapolate from the available data the processor time for a 128-point FFT by determining the square root of the ratio between 256 and 64-point, we get an estimate of 35 µs, adding up to 35 ms for the Nx M-point FFT operation.

Matrix transposition is a costly operation. Note that we can't store and operate on our data in TCM due to DMA access limitations. The 8 kB data cache is not useful for SRAM accesses located on non-incrementing addresses; a matrix transposition is going to require copying into/from an incrementing 1024 bytes-long array from/into addresses offset by 1024 bytes. This means that just about every operation will be at SRAM speeds, which does not have specified access times. In reality, the situation is even somewhat more complicated due to complex FFT requiring interleaved real and imaginary parts of the values.

We can already see that just the raw FFTs are going to take 78 ms. With the transposition and additional overhead, it would be challenging to achieve just 10 Hz, which is poor for control-theoretical applications as well as human operation. Additional overhead would come from the required memory operations left out of these calculations, as they can not be easily specified without being implemented and benchmarked.

Thankfully, there is a wide range of embedded computing systems available. If we limit ourselves to the realm of ARM, the next performance tier are the lower-end Cortex-A cores. The Nvidia Jetson Nano development kit was used to benchmark FFT performance of its modern quad-core ARMv8-A 64-bit out-of-order superscalar Cortex-A57 processor running at 1.5 GHz, featuring a VFPv4 vector FPU for each core, and plentiful two-layer caching. Using the NE10 library, optimized for ARM vector compute [32], a 1024-point FFT to take 17 µs in a single-threaded task, at 1.5 GHz with a power consumption under 3 W. The benchmark program is modified F32 complex FFT sample code from the NE10 library and is provided in the attached project files.

In comparison with a modern PC system: 1024-point complex FFT took 2.6 µs on a 4.3 GHz on a single core of the Intel i7-9750H (microcode update 0xCA) laptop CPU, driving the processor's power consumption 23 W above its idle power consumption of 4 W using the FFTW library under Windows 10.0.18363 wrapped in pyFFTW, run with python 3.7.4. Furthermore, compiled native C code using single-core FFTW executing 1024-point complex DFTs has been measured against parallelized CUDA with the CuFFT library.

CUDA [33] is a high core-core count architecture, primarily designed for graphical processing on vectored floats, but extensively used for short integer operations for machine learning, and other GPGPU (general purpose GPU) usecases. Taking a task and offloading it to the external resource, the GPU, typically brings latency overhead, and the total execution time is dependent not on the average unit execution time, but the worst unit execution time, so parallelizing operations comes with its own set of dangers. The overhead and lower per-core performance of CUDA caused CuFFT to be approximately half as fast as FFTW at 1024-point, and only gain an upper hand at 4096-point and larger. However, CUDA allows us to perform many operations in parallel.

On the personal computer system, CuFFT on the 896-core GTX 1650 performed 128 1024-point FFTs in a time equivalent to 172 ns of sequential execution, outperforming the straight-C FFTW time of 2.14 µs. On the Jetson

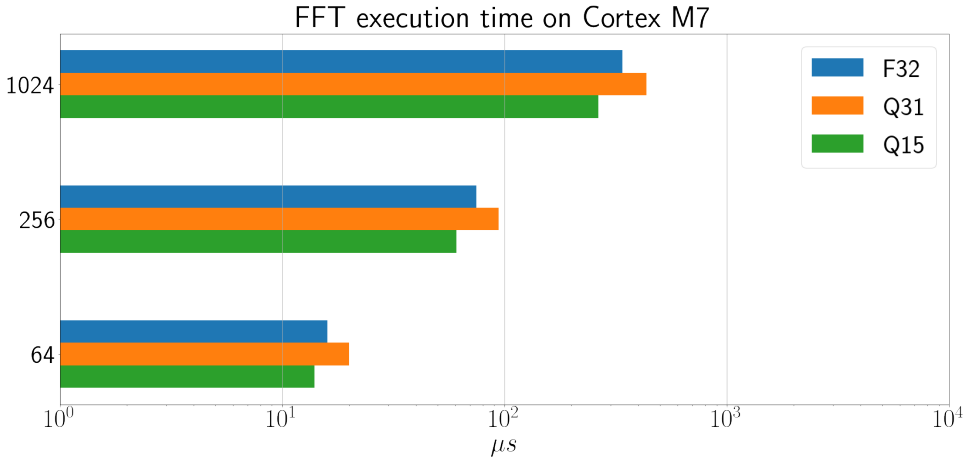


Figure 3.8: FFT computation time on a Cortex-M7 device at 216MHz with instruction and data caches enabled [3]

Nano, CuFFT on the 128-core GPU performed 128 1024-point FFTs in a sequential equivalent time of 1.5 µs, outperforming the single-threaded NE10 at 17 µs. This demonstrates that many-core solutions are a potentially highly power-efficient fit for radar systems. All used code is provided for replication in the attachments in FFT and fftw-vs-cufft subdirectories of the testing directory. FFTW vs CuFFT is a modified version of a free benchmarking setup by Github user moznion. [34]

If we wish to identify targets, peak detection has to be implemented. A popular option is CFAR (constant false alarm rate algorithm), a stateless detector, the simplest instance of which triggers when a value in a "cell" is higher than the cell value average with an added threshold. When peaks tend to be wider than one cell, a "guard cell" pattern is introduced, which prevents points close to a peak from registering as peaks.

Let us evaluate the computational complexity of CFAR without guard cells. With the resulting range-velocity plot still at a size of $M \times N$, let our cells have the size of $P \times P$ points. To get cell averages, we do $M \times N$ mostly fusable sums and P^2 divisions. At a Cortex-M7 core clock of 216 MHz, assuming 1-3 clocks per each of these instructions, this creates a latency of under 5 ms, negligible in comparison with the FFTs.

It should be noted, that ARM Helium, a low power DSP instruction set extension, will be available for cores implementing the upcoming Armv8.1-M microarchitecture. [35] Out of these cores, only one, the Cortex-M55, has been announced. The floating-point version (MVE-F) of Helium will bring, among other things, support for half-precision float vector operations, and most likely noticable speedup of DSP loops due to removed overhead. This might somewhat change the playing field of high complexity, very low power computation in embedded systems.

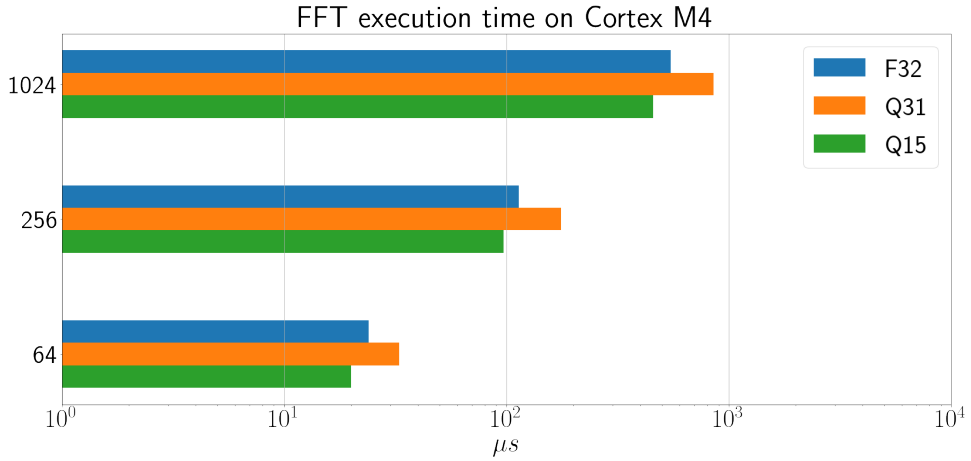


Figure 3.9: FFT computation time on a Cortex-M4 device at 180MHz [3]

The original evaluation kit firmware gets the frequency spectrum of the incoming data from each chirp, averages the chirp spectra, and detects peaks with a CFAR algorithm with configurable parameters. This is in practice grossly insufficient for velocity estimation.

3.7.4 DMA request latency

The latency of the DMA transfer itself can be estimated using manufacturer-povided documentation [36] to be 5 AHB clocks in the case of an AHB peripheral such as GPIO, which comes out to be 23 ns with the maximum core and AHB clock of 216 MHz. This for copying the IDR of a single GPIO bank. We are copying two, sequentially, and there is a risk of this trigger coinciding with an interrupt, adding further cycles, up to an estimate of 13 cycles until both registers are copied into SRAM, corresponding to 60 ns. With a 5 MHz sampling rate, our latched ADC codes are constant on the GPIO inputs for 100 ns. Our transfers fit into this timeframe.

However, this documentation does not cover the latency of the transfer being triggered. For this reason, before the device was built, a simple feasibility test was conducted to determine how fast and reliably can a DMA transaction be triggered by a timer event on an empty bus, since it was not clear if there was a need for the DMAMUX peripheral, which only available in newer STM32 parts. It was only needed to establish an upper bound, so some liberties in inaccuracy have been taken. A Nucleo board with a STM32F756ZG microcontroller was used, with the following configuration:

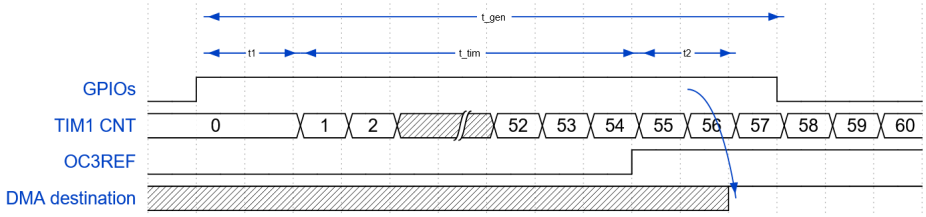


Figure 3.10: Generated pulse long enough

- Rohde & Schwarz HAMEG HMF2550 50MHz arbitrary generator, generating a variable-length 3.3V pulse every 100 ms,
- GPIOF with a single input pin connected to the signal generator output
- GPIOE with a single input pin, which corresponds to the TIM1 channel 1, used as timer input 1
- TIM1 timer peripheral running at 216 MHz (i.e. max bus clock) with 0 prescaler (no division)
- TIM1 channel 1 set up as TI1FP1 (timer input 1) no filter, rising edge, linked to:
- TIM1 channel 3 in one pulse mode (OPM) PWM mode 2, slave mode set to trigger, generating an OC3REF output compare event on capture at CNT = CCR3 = 54 cycles, corresponding to 250 ns, reloading arbitrarily at 432 cycles (2 µs).
- DMA2 peripheral, stream 6 channel 0, circular mode (to prevent interrupt handler from resetting the transfer finished interrupt enable bit)
- DMA2 source: GPIO bank F input data register
- DMA2 destination: an allocated word in RAM with the GNU C Compiler "unused attribute" macro
- DMA2 trigger: the OC3REF event
- Breakpoint in said DMA's transfer complete interrupt handler

The exact used source code is provided in the DMAReqTest subfolder of the repository and attachment. The sequence of events in each run is as presented in Figures 3.10, 3.11:

The generator output goes high. It is connected to the GPIOE pin, sets TI1, which triggers the timer 1. Timer 1 starts counting up, until its CNT register reaches CCR3. At that point, OC3REF is set. This triggers the DMA transfer, which copies the state of the GPIOF pin into RAM. This sequence takes $t_{seq} = t_1 + t_{timer} + t_2$, where we perfectly know t_{timer} with a precision equal to that of the microcontroller crystal oscillator. t_2 includes

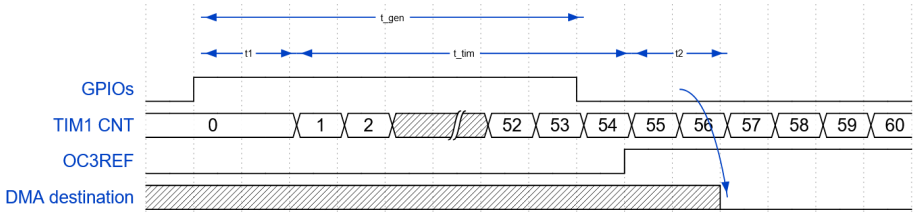


Figure 3.11: Generated pulse too short

the DMA request latency. From the timing diagram, we can see, that when the generator pulse time t_{gen} is lesser than t_{seq}, we read a zero on the GPIOF pin, and a one if it is greater. By varying t_{gen}, we can deduce, that the bit flip happens repeatably (N=50) at $t_{gen} = t_{timer} \pm 40\,\text{ns}$. The precision of the measurement was limited by the limited slew rate of the generator's output, and its pulse width being possible to change only in 5 ns increments. Symmetricity of the pulse edges was verified informally with an Agilent DSO3102A 100MHz digital oscilloscope, eliminating most of the distrust in the absolute value of the measurement. Regardless, this has proven that a DMA can read parallel data from GPIO buses reliably on an empty bus without, within the timeframe of 200 µs that a 5 Msps ADC requires, without the DMAMUX peripheral.

3.7.5 I/Q equalization

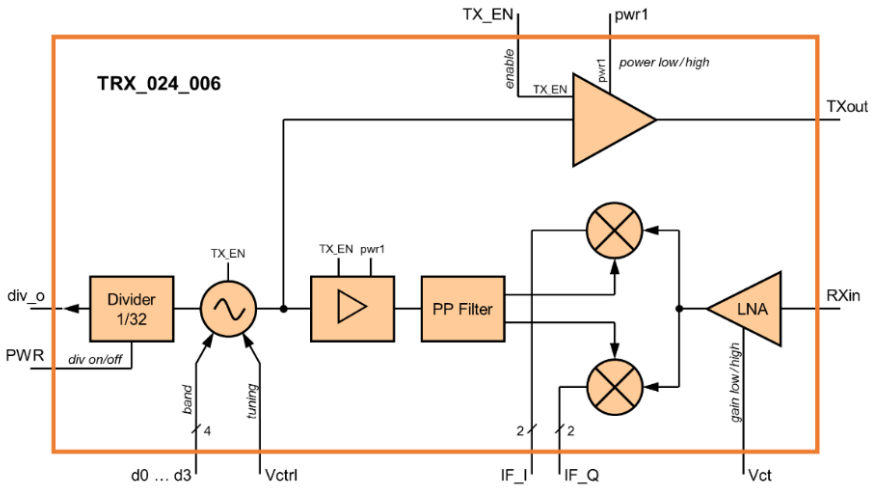


Figure 3.12: Block diagram of the 24GHz transceiver [4]

The core part of the 24GHz RF frontend, the TRX_024_006 IQ Transceiver, generates a sine wave in the 24GHz band offset by a frequency by the Vctrl analog input, see Figure 3.12. It then generates an I/Q pair from this signal to downconvert the received signal with. However, it does not do so ideally,

the part is specified to have a gain error of ±1 dB, and a phase error of ±10°.

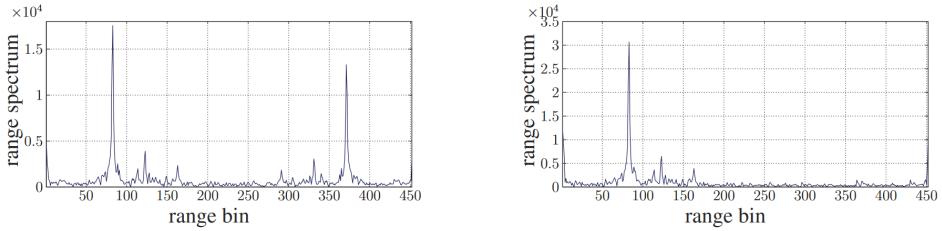


Figure 3.13: I/Q imbalance creates ghost targets. a) uncalibrated, b) calibrated. From Ulrich, Yang 2017 [5]

For example, easily implementable IQ gain and phase calibration introduced in Zekkari, Djendi, Guessoum 2019 [37] conclude that the following estimators noticably improve SNR in digital communications:

$$\hat{g}_I = \sqrt{\frac{E\left\{y_I^2(n)\right\}}{E\left\{r_Q^2(n)\right\}}} = \sqrt{\frac{E\left\{y_I^2(n)\right\}}{E\left\{r_I^2(n)\right\}}} \tag{3.1}$$

$$\hat{\phi}_Q = \arcsin \frac{E\left\{y_I(n) y_Q(n)\right\}}{\sqrt{\frac{E\left\{y_I^2(n)\right\}}{E\left\{r_I^2(n)\right\}} \sqrt{\frac{E\left\{y_Q^2(n)\right\}}{E\left\{r_I^2(n)\right\}} E\left\{r_I^2(n)\right\}}}} \tag{3.2}$$

$$= \arcsin \frac{E\left\{y_I(n) y_Q(n)\right\}}{\sqrt{E\left\{y_I^2(n)\right\}} \sqrt{E\left\{y_Q^2(n)\right\}}} \tag{3.3}$$

where y is local oscillator output and r is received signal. We can eliminate the unknown received signal power, since we only care about the ratio $\frac{g_I}{g_Q}$. This method can continually calibrate for slowly changing error values, and should be considered in embedded real-time target detecting systems.

Chapter 4

Conclusions

This work demonstrates, that accessible industrial microcontrollers can be used to perform radar baseband sampling and passing onto a USB-HS bus, and that current existing ARM Cortex-M cores are on the edge of being insufficient for executing the spectral analysis needed for target detection. It also demonstrates that existing heterogenous embedded systems such as the Nvidia Jetson Nano can perform the digital signal processing required at rates exceeding the typical sampling rates while fitting into tight power budgets. Bus latencies involved are calculated and empirically verified, motivating development of high sampling rate applications with FPGA or ASIC technology. The self-developed hardware, firmware, and software is usable as-is for generic sampling of FMCW radar basebands, however, further work should be done on configurability of the sampling and USB protocol, and compatibility with the evaluation kit it has been designed for. This project has been slowed down considerably by the selection of Qt framework, which is not the best fit for simpler scientific applications, where iterating code is prioritized over graphical interfaces and pre-compiled binary performance. The author would like to suggest the use of Julia with calls to libusb Python bindings for similar projects in the future.

Bibliography

[1] Analog Devices. Ms-2374: Jesd204b survival guide, 2014. URL https://www.analog.com/media/en/technical-documentation/technical-articles/JESD204B-Survival-Guide.pdf.

[2] Texas Instruments. Adc12010 datasheet, document code snas185b, 2018. URL https://www.ti.com/lit/ds/symlink/adc12010.pdf.

[3] STMicroelectronics. Pm0253 rev5: Stm32f7 series and stm32h7 series cortex®-m7 processor programming manual, 2019. URL https://www.st.com/content/ccc/resource/technical/document/application_note/group0/c1/ee/18/7a/f9/45/45/3b/DM00273990/files/DM00273990.pdf/jcr:content/translations/en.DM00273990.pdf.

[4] Silicon Radar GmbH. Trx_024_006 24-ghz highly integrated iq transceiver datasheet version 2.2, 2019.

[5] M. Ulrich and Bin Yang. Iq and array calibration for fmcw radar. In *2017 18th International Radar Symposium (IRS)*, pages 1–10, 2017.

[6] Ian P. Howard. *Perceiving in Depth, Volume 1: Basic Mechanisms*. Oxford Psychology Series. Oxford University Press, 1 edition, 2012. ISBN 019976414X,9780199764143.

[7] Eratosthenes of Cyrene. *Geographika / Erastothenes' Geography*. Assumed Library of Alexandria / Princeton University Press, Alexandria / United Kingdom: Princeton university Press, 6 oxford street, woodstock, oxfordshire ox20 1tw, 1st edition, approx. 265 BC / 2010 AD. ISBN 9780691142678.

[8] A. Einstein. Zur elektrodynamik bewegter körper. *Annalen der Physik*, 322(10):891–921, 1905. doi: 10.1002/andp.19053221004. URL https://doi.org/10.1002/andp.19053221004.

[9] Peter Milonni. *Fast light, slow light, and left-handed light*. Institute of Physics, Bristol Philadelphia, 2005. ISBN 978-0-7503-0926-4.

[10] James Clerk Maxwell. VIII. a dynamical theory of the electromagnetic field. *Philosophical Transactions of the Royal Society of London*, 155: 459–512, December 1865. doi: 10.1098/rstl.1865.0008. URL https://doi.org/10.1098/rstl.1865.0008.

[11] L. ESSEN and J. V. L. PARRY. An atomic standard of frequency and time interval: A cæsium resonator. *Nature*, 176(4476):280–282, August 1955. doi: 10.1038/176280a0. URL https://doi.org/10.1038/176280a0.

[12] J. P. Cobos Carrascosa, B. Aparicio del Moral, J. L. Ramos Mas, M. Balaguer, A. C. López Jiménez, and J. C. del Toro Iniesta. Scientific computing and fault mitigation on fpga aboard the solar orbiter phi instrument. In *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 1–8, 2015.

[13] W. Weiguo, Y. Suochang, Z. Jianguo, and S. Mingliang. The design of test system for launcher equipment of one air defence missile based on fpga. In *2011 First International Conference on Instrumentation, Measurement, Computer, Communication and Control*, pages 622–625, 2011.

[14] K. Khandagle and S. S. Rathod. Implementation of mri gradient generation system and controller on field programmable gate array(fpga). In *2018 International Conference on Communication information and Computing Technology (ICCICT)*, pages 1–4, 2018.

[15] Y. Wei, L. Chen, R. Xie, L. Song, X. Zhang, and Z. Gao. Fpga based video transcoding system with 2k-4k super-resolution conversion. In *2019 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–2, 2019.

[16] F. Bandiera, A. Coluccia, V. Dodde, A. Masciullo, and G. Ricci. Crlb for i/q imbalance estimation in fmcw radar receivers. *IEEE Signal Processing Letters*, 23(12):1707–1711, 2016.

[17] STMicroelectronics. Stm32f732xx, stm32f733xx datasheet, ds11854 rev 4, 2018. URL https://www.st.com/resource/en/datasheet/stm32f732ie.pdf.

[18] USB Implementers' Forum. High-speed inter-chip usb specification 2.0 / iec 62680-2-1:2015 - part 2-1: Universal serial bus specification, revision 2.0, 2000.

[19] Pete Batard. Zadig - usb driver installation made easy, 2020. URL https://web.archive.org/web/*/https://zadig.akeo.ie/.

[20] Jakub Vodseďálek. Měřicí kamera pro výukové laboratoře, 2019. URL https://dspace.cvut.cz/handle/10467/83303.

[21] mcuee. libusb windows github wiki page, 2020. URL https://github.com/libusb/libusb/wiki/Windows.

[22] SPICE Project. Usbdk at a glance, 2015. URL https://www.spice-space.org/download/windows/usbdk/UsbDk_at_a_Glance.pdf.

[23] Goz. Blackman-harris in c, a stack overflow thread, 2012. URL https://www.analog.com/media/en/technical-documentation/technical-articles/JESD204B-Survival-Guide.pdf.

[24] Anton Mikhailov. Turbo, an improved rainbow colormap for visualization, 2019. URL https://web.archive.org/web/*/https://ai.googleblog.com/2019/08/turbo-improved-rainbow-colormap-for.html.

[25] J.W. Andrews. Saleae logic analyser clone teardown & reprogramming, 2011. URL https://web.archive.org/web/20190913145643/http://www.jwandrews.co.uk/2011/12/saleae-logic-analyser-clone-teardown-and-reprogramming/.

[26] Dave Jones. Eevblog #436 - saleae usb logic analyser review & teardown, 2013. URL https://www.youtube.com/watch?v=7OCPWCdg2ys.

[27] sigrok community. Sigrok wiki: Noname saleae logic clone, 2020. URL https://web.archive.org/web/20200709091635/https://sigrok.org/wiki/Noname_Saleae_Logic_clone.

[28] Osmocom team. librtlsdr website, 2020. URL https://web.archive.org/web/20191126170721/https://osmocom.org/projects/rtl-sdr/wiki.

[29] Teledyne Technologies. Labmaster 10-100zi 100 ghz oscilloscope, 2019. URL https://web.archive.org/web/20190415051112/https://teledynelecroy.com/100ghz/.

[30] David Kosenina Dejan Priversek. Scopefun - open source instrumentation, 2020. URL https://web.archive.org/web/20190509173644/https://www.scopefun.com/.

[31] STMicroelectronics. An4841 rev2: Digital signal processing for stm32 microcontrollers using cmsis, 2018. URL https://www.st.com/content/ccc/resource/technical/document/programming_manual/group0/78/47/33/dd/30/37/4c/66/DM00237416/files/DM00237416.pdf/jcr:content/translations/en.DM00237416.pdf.

[32] Project Ne10. An open optimized software library for the arm architecture, 2019. URL https://web.archive.org/web/*/https://projectne10.github.io/Ne10/.

[33] NVIDIA Corporation. Cuda faq, 2020. URL https://web.archive.org/web/*/https://developer.nvidia.com/cuda-faq.

[34] moznion. Fftw vs cufft, 2015. URL https://github.com/moznion/fftw-vs-cufft.

[35] Arm Limited. Ddi0553: Armv8-m architecture reference manual, twelfth release, 2020. URL https://developer.arm.com/documentation/ddi0553/bl/.

[36] STMicroelectronics. An2548 rev7: Using the stm32f0/f1/f3/gx/lx series dma controller, 2020. URL https://www.st.com/content/ccc/resource/technical/document/application_note/47/41/32/e8/6f/42/43/bd/CD00160362.pdf/files/CD00160362.pdf/jcr:content/translations/en.CD00160362.pdf.

[37] Chahrazed Zekkari, Mohamed Djendi, and Abderrezak Guessoum. IQ imbalance estimation and compensation in receiver system. In *2019 International Conference on Advanced Electrical Engineering (ICAEE)*. IEEE, November 2019. doi: 10.1109/icaee47123.2019.9014803. URL https://doi.org/10.1109/icaee47123.2019.9014803.

.0.1 Notable abbreviations

- ADC - analog to digital converter
- AWGN - additive white gaussian noise
- CMOS - complementary metal oxide semiconductor; in this context, transistors with traditional voltage-driven logic
- CRLB - Cramer-Rao Lower Bound, a characteristic of best possible estimator quality
- DFT - discrete Fourier transform
- DMA - direct memory access, a microcontroller peripheral for CPU-free byte copy
- ESD - electro-static discharge, a common failure mechanism of semiconductors, caused by low-energy, high-voltage discharges between insulated objects
- FFT - fast Fourier transform, a performant implementation of DFT
- FIFO - first in first out memory, equivalent to "buffer"
- FMCW - frequency modulated continuous wave, a type of radar
- FPGA - field-programmable gate array, a reprogrammable arbitrary logic device

- FS - full scale of a value range
- GPIO - general purpose input output, a microcontroller peripheral for mapping signals to electrical connections
- HAL - hardware abstraction layer
- IRQ - interrupt request, an event which triggers immediate unconditional code execution, unless pre-empted by a higher priority interrupt request
- I/Q - in-phase and quadrature-phase components of a complex signal
- LSB - least significant bit, lowest resolution unit of an ADC
- LVDS - low voltage differential signaling, a type of high bandwidth current-driven digital bus physical layer
- Msps - million samples per seconds, interchangable with MHz
- MIMO - multiple input, multiple output. Used in digital communications and radar
- MVE - M-Profile Vector Extension, a general label for ARM Cortex-M vector math instruction set extensions
- RF - radio frequency
- TCM - tightly coupled memory, directly accessible by the processor core with no bus arbitration overhead
- TIM - timer, in the context of STM32 microcontroller peripherals

All digital information units are intended to be read as per JEDEC convention of 1KB (read one kilobyte) = 1024B.

.0.2 Attached files

The attached files include all hardware design files, their manufacturing outputs, all source code, a Windows x64 binary executable of the desktop software viewer, and an .elf and .hex firmware binary files. All of these, including .tex sources for this thesis are also available at `https://gitlab.com/tywonemi-school-stuff/silicon-radar-fun`.