UN0RICK: OPEN-SOURCE ULTRASOUND FPGA BOARD

A POLYGLOT DOCUMENTATION FILE

Luc Jonveaux*
Tinkerer, Milly le Meugon, France
contact@un0rick.cc

August 14, 2019

ABSTRACT

Non destructive testing and imaging ultrasound have been around since the '50s. Many ultrasound open-source projects are emerging, mostly focusing on image processing - while hardware has been left behind. Several teams have produced successful designs to be used on commercial US scanners, but they are not cheap, and are difficult to access. I couldn't find designs to play with, that would be affordable or open, so I decided to make one for makers, researchers and hackers.

This PDF is also a ZIP that contains the sources to the hardware and some data too, don't hesitate to have a look. Just rename the file from .PDF to .ZIP and you're ready to go .

Keywords open-source · ultrasound · hardware · ice40 · fpga

1 Overview

This wonderful board has been designed to provide a curious tinkerer with the basis to play with, and understand, ultrasound NDT and imaging bases.

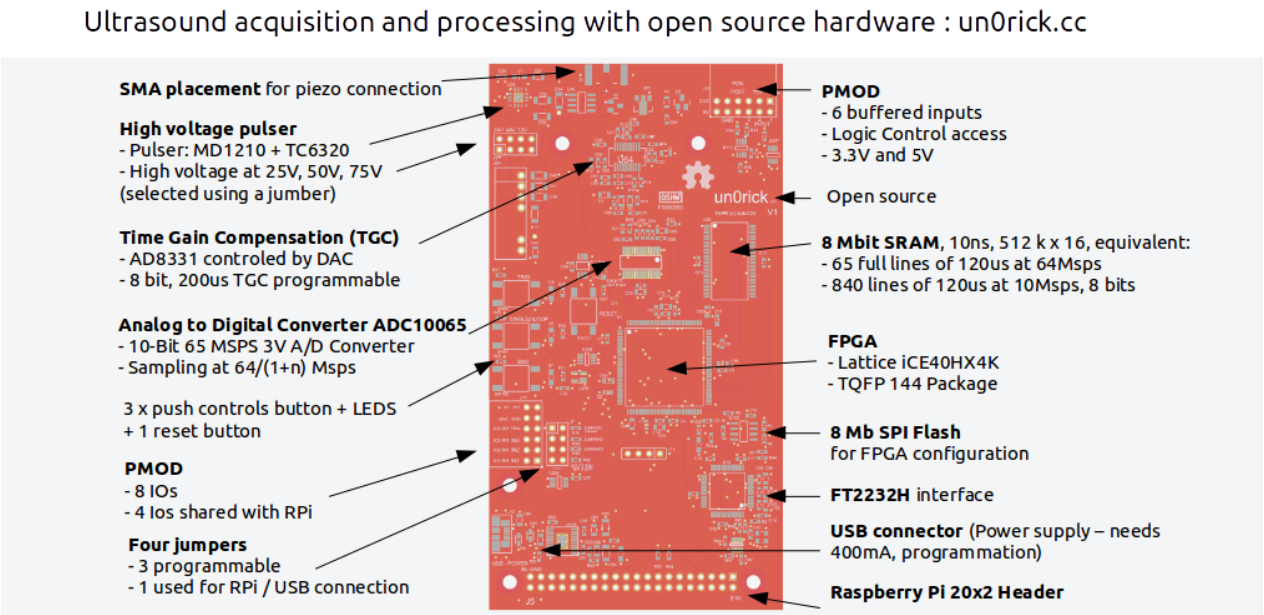


Figure 1: What is on the board ?

*More on the website http://un0rick.cc. This paper has its on Zenodo DOI 10.5281/zenodo.3364559

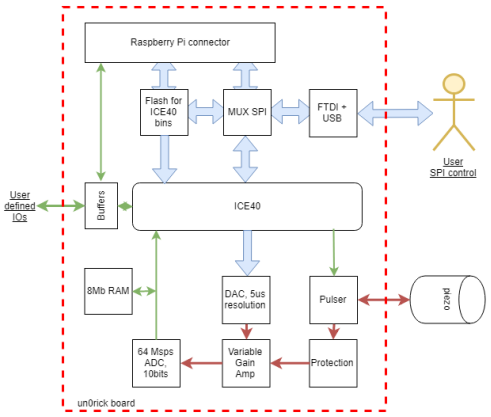
It has not been designed with cost-efficiency, size optimization and having the lightest BOM - instead, it tries to be an understandable assembly of the different blocks that constitute a pulse-echo system : a High Voltage (HV) source, a pulser, a protection, a Time Compensation Amplifier, a good ADC, and, at the heart, a cheap but interesting FPGA to provide with all the controls and interfaces. There are also a couple of IOs, and some space for a Raspberry header - so that you can use with your favorite card-sized computer, but also with any chip with an available SPI bus.

1.1 Concept

That said, the board is based on a HX4K/HX8K FPGA, with plenty of IOs broken out. It communicates with the outside world via an SPI bus, available either through USB with the on-board FTDI.

The FPGA is really at the heart of the platform, controlling pulser, ADC, TGC, interfacing with the high-speed RAM and the different IOs.

A benchmarking exercice showing that 50-80Msps ADCs were the norm, especially for piezos between 3 to 10MHz, I had to doubt puttign a 65Msps ADC onboard. Its 10 bits mean that I can use remaining bits on the 16-bit words to add in some information - always handy.
This platform has been used with NDT piezos, with regular piezos, and with vintage mechanical probes. It seemed to work with all - just be aware you may want to work on impedance matching sometimes, somehow.



1.2 What about the specifications of the board ?

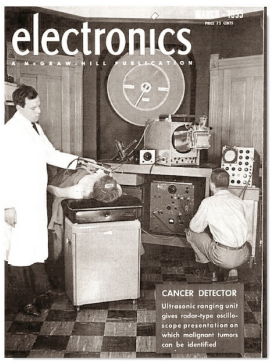


Figure 2: Old news

FPGA: Lattice iCE40HX4K - TQFP 144 Package: use for quite a number of IOs, though no RAM, and compatibility with Clifford Wolf's Yosys Open SYnthesis Suite [2].

Memory 8 Mbit SRAM, 10ns, 512 k x 16 (equivalent 65 full lines of 120us at 64Msps or 840 lines of 120us at 10Msps, 8 bits), as well as 8 Mb SPI Flash for FPGA configuration.

Ultrasound processing: A VGA (AD8331) controled by DAC, with a pulser: MD1210 + TC6320, and a 65Msps ADC (ADC10065).

Extensibility: 2 x PMOD connectors, two SMA plug for the piezos (with capacity to separate the TX and RX paths) as well as a general header for RPi GPIO

User Interfaces: 2 x PMOD for IOs, 3 push button (with software noise debouncing) and jumpers for high voltage selection

Input Voltage: 5 V from RPi or USB, uses 350mA-450mA at 5V with a raspberry on. The FPGA and logic operate at at 3.3 V, still with high voltage at 25V, 50V, 75V

2 Where to find the latest sources

The latest sources of the hardware as well as software are available at https://github.com/kelu124/un0rick/. However, this PDF also doubles as an archive (you can rename the .pdf as a .zip, and you'll see), and contains, in short: a set of gerbers and BOM, some VHDL code, a basic FPGA binary ready to be used, and a python library to operate the board from a Raspberry Pi. There may be some other stuff there, but I forgot what I put there.

The hardware file is copied on GitHub, feel free to check it on UpVerter though - thanks David!


Figure 3: NDT

3 Operation

Okay, we know the board now. But how is it operated? The usual flow of operation is to first flash the FPGA with the right "server", configure the acquisition details, get the signal, and to process it. Let's see how to do it in more details.

3.1 Programming

The first thing is to program the FPGA with its binary. The recommended binary is attached in this archive (with a .bin extension). First is to use a computer with any FPGA programming capacity, such as diamond programmer or IceProg for

Linux. Don't forget to setup the the jumper (see below) for FPGA update before connecting to the USB. The FTDI should appear in you device list. You're then ready to program using your tool. The LED XXX should then be ready. The onboard flash will keep the binary in memory, so that's something that needs to be done only once.

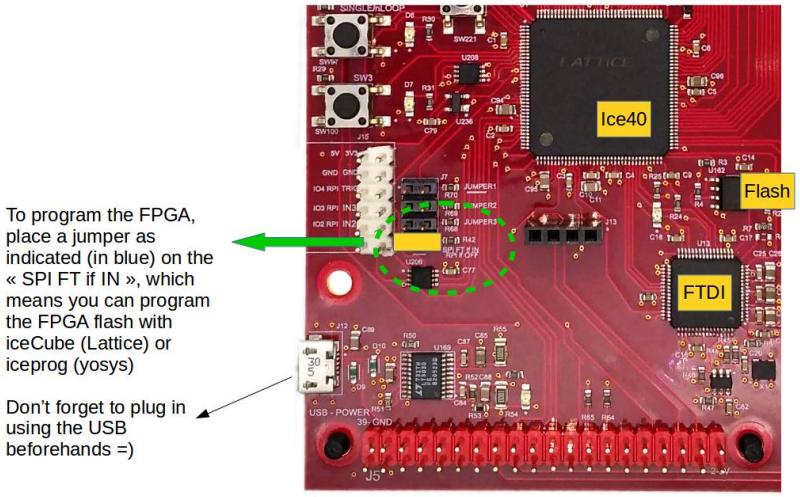


Figure 4: Programming the board

3.2 Preparing

The platform can be set up with the SPI bus, to configure the different acquisition values as well as the TGC gain. It using the following addresses and protocol (SPI is only 8bits per CS). For each address, one needs to send 0xAA 0xXX 0xYY, with XX the register address, and YY the data one wishes to send. Register addresses start at 0xE0 and end at 0xD0 on the default bitstream. The following can be programmed:

Figure 5: Robotic arms

- Type of acquisition (one line / set of lines)
- Number of lines
- Length of lines acquisitions
- Delay between acquisitions
- Pulse width
- Delay between pulse and beginning of acquisitions
- 200us time-gain-compensation programmable (8 bits, from 0 to Max), every 5us

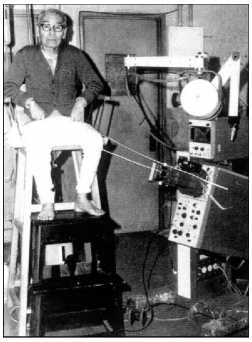


Figure 6: With an old endo-cavity setup

For the TGC, the addresses start at 0x10 and end at 0x38. This corresponds to t = 0us for 0x10, and t = 200us for 0x38. Values that can be saved range from 0 to 8 bits for full amplification.

The LED 6 is linked to the multi/single line status. Therefore, it is easy to check if the FPGA is ready to shoot: one can toggle the register to check if the LED blinks. It is done in pyUn0 with the `test_spi` from the `us_spi` class, by typing `python pyUn0.py test`.

The pyUn0 lib for example provides a wrapper to easily manipulates an acquisition set of parameters with a Raspberry (one just needs to leave the jumper on from Fig 3), you can also find code for an exercice of using an ESP32-based M5Stack board (check out the `esp32_prog.ino` file in the archive).

The python lib also provides access to objects that can be manipulated, including the time, all metadata, the signal itself, and allows for preprocessing the data according to the metadata (for example filtering the signal given the frequency and bandwidth of the element used).

3.3 The registers

Address	bitsize	Description	Default value	Unit	Value
0xE0	8	Lengh of Pon	0x14	1/128us	200 ns
0xD0	8	Lengh between Pon and Poff	0x0A	1/128us	100 ns
0xE1	8	Lengh of Poff MSB			
0xE2	8	Lengh of Poff LSB	0xC8	1/128us	2000 ns
0xE3	8	Delay between Poff and Acq (MSB)	0x02BC	1/128us	7000 ns
0xE4	8	Delay between Poff and Acq (LSB)			
0xE5	8	Lengh of acquisition MSB	0x32C8	1/128us	130 us
0xE6	8	Lengh of acquisition LSB			
0xE7	8	Period of one cycle MSB	0x186A0	1/128us	1 ms
0xE8	8	Period of one cycle 15 to 8			
0xE9	8	Period of one cycle LSB			
0xEA	1	Software Trig : Auto clear	0	N/A	
0xEB	1	0: single mode 1 continuous mode	0	N/A	
0xEC	8	Voltage gain control: 0V to 1V	0x11	0,004	68 mV
0xED	8	Frequency of ADC acquisition	0x03	64/(1+f) MHz	16 MHz
0xEE	8	How many cycles in continuous mode	0x0A	cycles	10 cycles
0xEF	1	Sofware memory reset: set to 1; auto clear	0		

Table 1: The different registers in the current .bin binary

3.4 Acquiring and reading

If the master reads more data than the board has acquired, you will have old or incorrect values: the master has to know how many measures were done. The data is stored in RAM after the acquisition, for up to a 500k points depths. Again, the SPI bus is used to read the ADC value., by sending for example 0x00. If the FPGA is doing an acquisition, the returned value is 0xAA, else the value is on 2x8 bits, as follows:

- MSB: 8th bit: MSB ID: set to 1
- MSB: 6-7th bit: cycle bits: to identify a line in a sequence
- MSB: 4-5th bits: the two inputs (TopTurn 1 and TopTurn2) in case a counter is used.
- MSB: 1st-3th bits: ADC 7 to 9: 3 remaining bits of acquisition
- LSB: 8th bit: LSB ID: set to 0
- LSB: 1st to 7th bits: ADC 7-0: 7 first bits of acquisition

3.5 State of the art

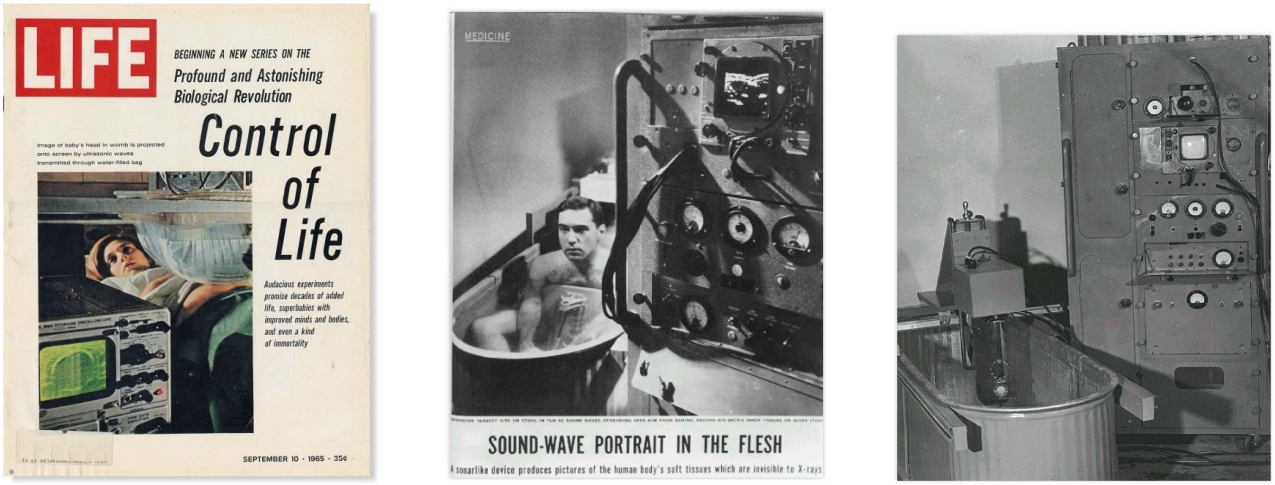


Figure 7: The latest ultrasound news

On the first image, note the water bag for acoustic matching. On the last image, the tube holding the focused transducer ad a fairly narrow beam and a broad bandwidth signal response which allowed for high axial and lateral resolution. The transducer head could be in moved linearly back and forth or it could oscillate back in a polar rotation over about 120 degree angle, allowing visualization of biological structures that where in the field of view of the sound beam.

3.6 Processing using the pyUn0 python library

The objective of this last step is to transform the raw json, which contains the data as well as the metadata of the acquisition, into something that can be processed further - usually using Python.

Attached in this archive as well is a pyUn0.py file, as well as the data of an experiment, if you want to try. An example of this is shown in this video. The FPGA was already programmed. I simply used the pyUn0 wrapper by typing `python pyUn0.py single` to acquire a single line, followed by `python pyUn0.py process`, to process it. The result is as follows:

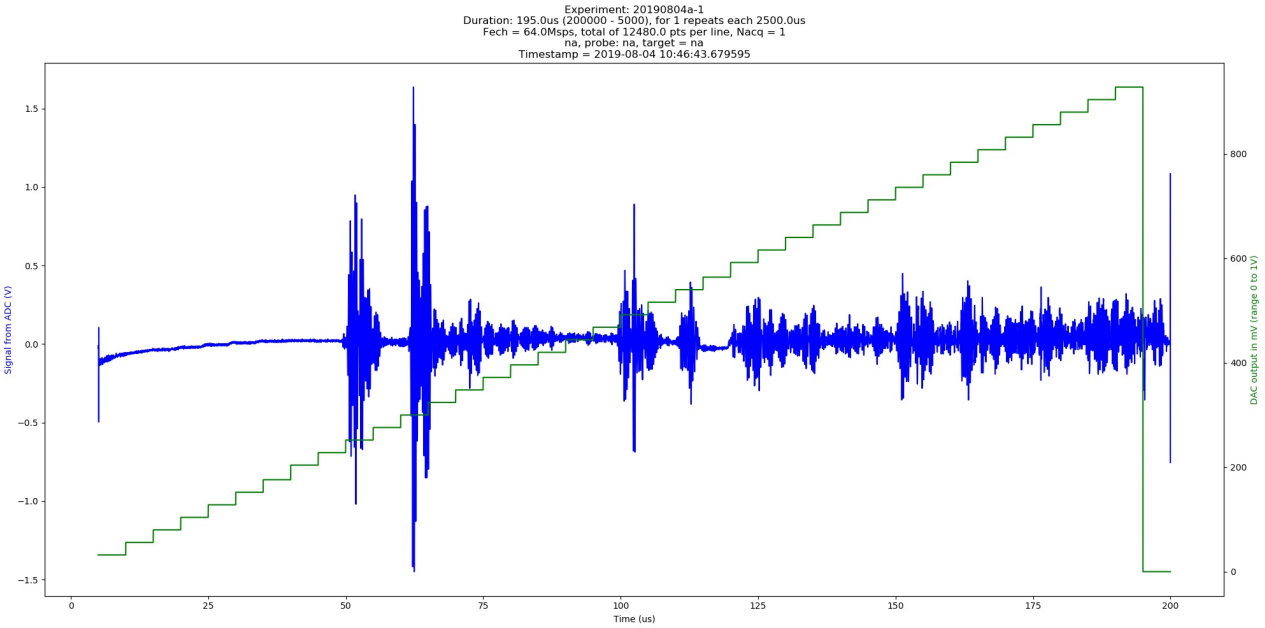


Figure 8: Results of the 20190804a experiment

The experiment files are also in the present archive - feel free to play with those.

3.7 Bonus

Interestingly, it is possible to reach a signal that is sampled at 128Msps - double of the maximum speed. Do you have an idea how this is achieved? Mail me and let me know!

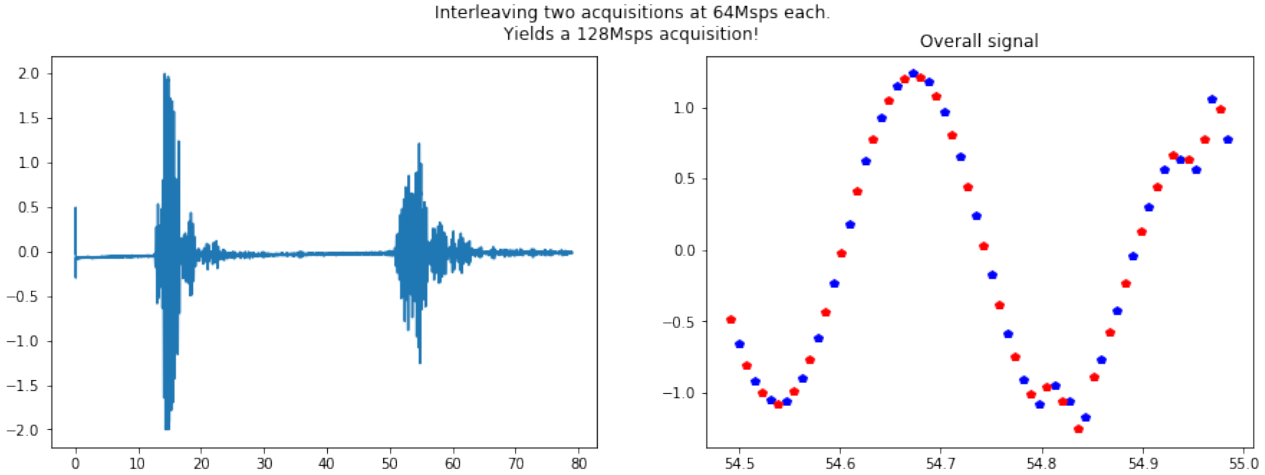


Figure 9: Programming the board

4 Last details

License: This work is based on a previous TAPR project, the echOmods project. The un0rick project and its boards are open hardware and software, developed with open-source elements.

Disclaimer This project is distributed WITHOUT ANY EXPRESS OR IMPLIED WARRANTY, INCLUDING OF MERCHANTABILITY, SATISFACTORY QUALITY AND FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. Also:

- This is not a medical ultrasound scanner! It's a development kit that can be used for pedagogical and academic purposes - possible use as a non-destructive testing (NDT) tool too, for example in metallurgical crack analysis.
- As in all electronics, be careful, especially with high-voltage.
- This is a learning by doing project, I never did something related: it's all but a finalized product.
- Ultrasound raises questions. In case you build a scanner, use caution and good sense!

Certification This piece of hardware is also open-hardware certified, under ID FR000005.

5 Links to go further

- Come and chat : join the Slack channel
- The full GitHub Repo with more ongoing works : also a messy braindump with all experiments
- The board's Tindie shop to get it
- The project Hackaday page with more logs
- wlmeng11's SimpleRick for a analog part board. Clever use of RTL-sdr hardware for the acquisition !
- Check out the OSHWA certification link under reference FR000005
- Check out my previous work on the topic of ultrasound modules [1] and its dataset on Zenodo.

6 Next steps

Plenty to do on the next steps! Let me know if you'd like to contribute. The current shopping list (non-exhaustive) may include:

- Improving the documentation, and prepare the work of the small sibling, lit3rick.
- Work on BOM costs and overall hardware design.
- Increase the high voltage source, and have it settable via an on-board, and ideally have a bipolar design. Including a new pulser chip as being tested on the lit3rick board, an up5k design that's getting ready.
- Improving the features of the onboard firmware.. and try to develop a VGA output !
- Work on the FTDI - so I have only used the RPi, and write something to program the flash from the RPi.
- Develop a small server on the host RaspberryPi to manage the board and deliver to several possible users.

References

[1] Luc Jonveaux 2017. Arduino-like development kit for single-element ultrasound imaging. In *Journal of Open Hardware, 1(1), p.3*. DOI: 10.5334/joh.2

[2] Clifford Wolf, Johann Glaser. Yosys - A Free Verilog Synthesis Suite. Website at *http://www.clifford.at/yosys/*