# Saturn Development Activities

This document describes the development steps required

# Project Overview

“Saturn” is a new FPGA radio project which will couple a Xilinx FPGA for the ADC/DAC and immediate DSP functions, together with a Raspberry Pi4 compute module for the data handling, protocol code and local SDR application. It will be possible to use Saturn in several ways. Other radios achieve each of these individually, but Saturn will be able to implement all of them.

|  |  |
| --- | --- |
| Diagram  Description automatically generated | The processor module is used simply to move data using protocol 1 – like the Red Pitaya’s processor. Low demand on the processor. Wired ethernet connection to PC running Thetis or other app. |
| Diagram  Description automatically generated | The processor module is used simply to move data using protocol 2. This will have higher demand because the data rate is higher. Wired ethernet connection to PC running Thetis or other app. |
| Diagram  Description automatically generated | The processor executes an SDR app such as Pihpsdr or linhpsdr. No PC required, and high quality display outputs are available. |
| Diagram  Description automatically generated | The processor execute Pihpsdr and has an attached 7” RPi touchscreen display. Possibly single ADC or 14 bit ADC version, with Apollo-like RF module. This could be a lower cost small radio, but there may be no market for it now. |

Saturn uses a Raspberry Pi4 Compute Module as its local processor. The unique element to the Pi4 CM is a single lane PCI Express interface. This provides the connection to the FPGA for both low bandwidth data settings and high bandwidth data transfer.

Saturn is the project name for the board accommodating ADC/DAC, FPGA and Raspberry Pi. It does not (yet, at least!) describe radios using it. Saturn does have some moons, which might be appropriate for that purpose when the time comes.

# Development Required

To make Saturn happen there are several aspects requiring development:

1. The FPGA itself – being coded using Vivado and making substantial use of Xilinx pre-defined IP modules.
2. The PCB. This has been laid out and bare boards have been manufactured; a 1st prototype awaits arrival of the
3. A “protocol 2 app” for the Raspberry pi; this will accept and send protocol 2 packets to a remote SDR client.
4. Subsequently, a port of Pihpsdr will be needed to provide local processing.
5. Utility code for the Raspberry pi. This is for functions such as the bool loader, to program the FPGA configuration memory a simple GUI app has already been created.
6. Documentation is always required!

# Development Plan

## FPGA

* Largely complete; the remainder is only a couple of weeks work.
* I’m working on the data interface between the PCI express interface and each DDC and TX, Codec. I needed a bug in Vivado 2020.2 to be fixed, and it is fixed in the new 2021.1 release.
* I need to repackage the schematic slightly, putting the DDC and DUC elements into their block design pages. This will allow them to be simulated and tweaked in isolation.

## PCB

* Manufacture prototypes; this is delayed because of component availability.
* Add the Raspberry pi4 compute module and test the “Pi4” interfaces (USB, ethernet, HDMI).
* Check the FPGA responds to JTAG; program the config prom through JTAG
* Use a simple “test FPGA” build to test the ADC/DAC/Codec/Alex Interface. This will route and ADC straight to a DAC for example, for testing with signal source & spectrum analyser.
* Add the full FPGA, and test its main elements. The initial focus will be on establishing that it is “about right” with detail to be ironed out later.

## Protocol 2 App

* Still to be written from scratch, but an outline of what it needs to do is known & documented. Essentially a single thread will probably service pairs of DDC; another the TX; another the Codec etc.
* Write code for discovery response, sending & receiving each message; and stopping & starting correctly.
* Expressions of interest welcome!

## Pihpsdr

* Later in the process. Take the P2 app and code something similar directly into Pihpsdr.

## Utility Code

* We already have a GUI and a command line config prom writer (like the bootloader app)
* We have a register reader/writer app
* Bit others to support debugging could be needed

## Documentation

* We could write a Saturn handbook now. That would describe the connectors and their functions, and the programming interface to the FPGA.

## Finishing

* Optimise the DSP from an initial “good enough” version
* Beta test the unit

# Things to Do

This is an incomplete list. It could be better to host this in a commercial “to do” list such as Jira.

|  |  |  |
| --- | --- | --- |
| **Item** | **Kit / Environment** | **Description** |
| RX data FIFO | Vivado | Each DDC needs its own data FIFO  Ideally, a multiplexed DDC0/DDC1 FIFO |
| Data transfer documentation | Document | Document the API for data transfers, after doing final tests of FIFO read/write IP |
|  |  |  |
|  |  |  |
|  |  |  |
| Test FPGA | Vivado, for Saturn | Create a “Test FPGA” block design. ADC1 or ADC 2 routed to DAC; Codec mic input to codec speaker output; a way to test 122.88MHz clock generation; XDMA with some simple registers; I2C connection to config prom; Alex driving code. Essentially enough to do rapid test of h/w once PCB arrives. |
| Litefury FPGA | Litefury | This is to permit writing of data transfer s/w before final h/w available.  Add DDS and datapath FIFOs to simulate RX DDC  Potentially loop back TX FIFO to an RX FIFO  Loop back speaker codec to mic codec  The sample rates need to be nearly right but not perfect. |
| TX datapath | Vivado | Widen TX datapath to 24 bits data |
| Test plan | documentation | Work out how we will test this robustly! |
| Alex Test jig | PCB design | Make a simple “alex emulator” with LEDs for each data bit |
| DSP performance | Vivado | Improve DSP performance to that required for final radio. Likely to need a working connection to pihpsdr or Thetis. |
| Pi test data transfer s/w | Pi / litefury | Write code to manager reading or writing one FIFO using DMA, so we achieve a constant stream. |
| Pi protocol 2 | Pi / litefury?? | Write a complete protocol 2 data transfer app. If I can emulate the DDC & DUC well enough, we might be able to do this using the litefury before final h/w is available. |
| Desktop github | Pi | Find linux equivalent of github desktop |
| Refactor block design into IP | Vivado | Create formal IP blocks for DDC, DUC, CW keyer, codec interface. Probably this only makes sense after the functions are substantially complete.  Vivado added a new feature “block design containers” in release 2020.2 – ability to have proper nested block designs each with its own testbed. |
| Make CW TX ramp programmable duration | Vivado |  |
| pihpsdr | pi | Modify pihpsdr to have direct data transfers, allowing a “front panel” radio operation in the same RPi |