An Embedded Platform for Scientific Radio Applications

Marcus Leech, *Science Radio Laboratories, Inc*

Abstract

We describe a computing and SDR-receiver platform for scientific radio applications such as small-scale radio astronomy, riometry, forward-scatter meteor observations, etc.

Introduction

The emergence of extremely cost-effective, low-power, embedded compute platforms, largely based on multiple-CPU ARM system-on-chip (SOC), developed for the smart-phone industry has afforded some interesting new approaches for small-scale scientific-radio applications.

In parallel, developments in the Software-Defined-Radio (SDR) ecosystem have allowed multiple-receiver arrays to be realized very cheaply.[1]

This paper focuses on a particular hardware platform, the *Odroid U3*, manufactured by a Korean company called HardKernel[2]. The device has some interesting features, including:

- Exynos 4412 SOC, providing 4 ARM Cortex(tm) A9 CPU cores at 1.7GHz
- 2GB RAM
- 3 USB-2.0 ports
- uSD and eMMC ports
- 10/100Mbit Ethernet port

The platform is able to run various Linux distributions, including ArchLinux, and Ubuntu.

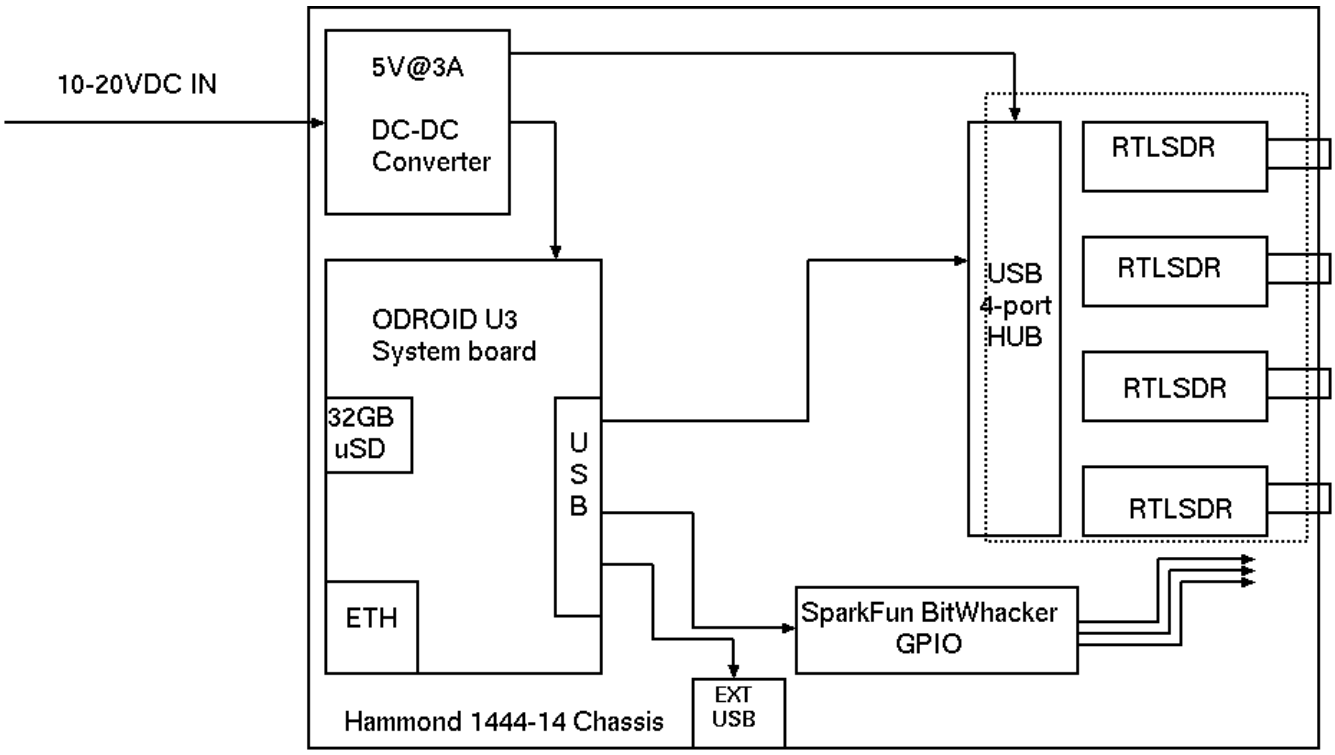
System Configuration

Projects like this are largely exercises in so-called *systems integration*, in which various hardware subsystems are assembled together to form a whole, without necessarily designing and implementing any new hardware. The author chose to house the assembled system in a Hammond 1444-series chassis-style enclosure, since it afforded a reasonably compact packaging, and was easy to work with.

The diagram below shows the gross architecture of the assembled system.

1 http://sdr.osmocom.org/trac/wiki/rtl-sdr

2 http://www.hardkernel.com/main/products/prdt_info.php?g_code=G138745696275



The system as shown above provides a very flexible and generic platform, even for schemes that don't involve scientific radio at all, for example spectrum monitoring and SIGINT.

System Software

The system runs the ARM version of *Arch Linux*, with various additional packages added to improve its utility for scientific radio experiments including:

- Gnu Radio 3.7.4
- gr-osmosdr with UHD, RTLSDR, HackRF, and BladeRF support
- gr-ra_blocks
- simple_ra
- pyephem
- pyserial (to control the SparkFun *BitWhacker*)

Additionally, many standard system-software features are enabled, including both Python2 and Python3, samba, nfs, ntpd, etc.

The current configuration includes a 32GB uSD card for storage, allowing for installation of many different relevant packages, and the storage of data on the platform itself.

Also, an external USB port allows one to insert a USB flash drive to move data off the device conveniently, which would be useful for autonomous operation where the unit may be visited occasionally for data transfer.

Software

The platform as-described is extremely flexible, and various types of software may be developed for the platform, and used in scientific-radio data collection.

Since the platform has Gnu Radio 3.7.4 installed on it, a simple flow-graph was constructed that provides for several different kinds of experiments that are commonly used by small-scale radio astronomy observers.

This flow-graph takes I/Q sample streams from 4 RTLSDR receivers, at 2Msps each, and computes both a high resolution (2048 bins by default) FFT and detected power for each receiver. Both data items are lightly averaged (via single-pole IIR integration), and then sent over a TCP connection at modest rates—a few Hz (5 by default) for the FFT data, and 1kHz for the detected power.

This straightforward architecture allows a variety of useful experiments using the TCP-based data streams.

In addition, the flow-graph knows how to talk to the attached *BitWhacker* GPIO module, and arranges to raise the lower four bits of the RB output “high” for 45 seconds at the top of every hour—this can be used to drive a calibration source. The calibration state is recorded in a “phantom” fifth detector channel in the TCP data—always either 0.0 or 1.0, with 1.0 indicating “calibration on”.

The resulting TCP streams can be processed with simple *Python* scripts, and test scripts have been written to intercept the data streams and log the resulting summary data. In fact, the scripts can run on the *Odroid* platform itself, with the log files simply accumulating on the local *uSD* filesystem. Another approach is to have the files written to a network filesystem—either NFS-based for *Unix/Linux/BSD* file-servers or CIFS-based, for *Windows(tm)* network file-shares.