1

THE UNIVERSAL SOFTWARE RADIO PERIPHERAL (USRP) FAMILY OF LOW-COST SDRs

MATT ETTUS AND MARTIN BRAUN
Ettus Research, USA

1.1 OVERVIEW

1.1.1 Software Defined Radio and Opportunistic Spectrum Access

With the incredible growth of devices that use the RF spectrum, the pressure to fit more users into the finite spectrum has pushed ever more efficient technologies to the fore. This growth is expected to continue at a pace faster than spectral efficiency measures are projected to grow. Despite this growth, there are still underutilized pieces of spectrum, but these pieces are often disjointed and geographically or temporally variable. At the same time, spectrum allocation is a slow and expensive process, so any technology that can lend flexibility to this process is of great value.

One piece of the puzzle may be opportunistic spectrum access (OSA), the subject of this book. A key enabling technology for OSA is software defined radio (SDR). SDR can allow one general-purpose hardware device to be used for many different types of communication systems simply by changing out the software, which implements the specific modulation, coding, and protocols. This is distinguished from previous radio systems where these functions were normally conducted by fixed-function

Opportunistic Spectrum Sharing and White Space Access: The Practical Reality, First Edition.
Edited by Oliver Holland, Hanna Bogucka, and Arturas Medeisis.

hardware with only minimal reprogrammability. With SDR, a device that implements OSA can survey the spectrum, determine which pieces are free and which are in use, and adapt to the conditions present at that instant. It can communicate with legacy devices and bridge between different systems.

While much of the complexity in a SDR is, of course, in the software, the flexibility it allows for places additional demands on the hardware devices. In particular, the ability to operate in the presence of (and often in between) strong adjacent signals becomes paramount in crowded spectrum. Similarly, a requirement to not interfere with other users of the spectrum means that the transmitter must have exceptionally low emissions in adjacent spectrum, especially if there are incumbent and/or primary users. In order to meet these requirements, a radio must have high linearity, and this is often at odds with the need for low power consumption. The design of systems for use in OSA applications is still an area of open research.

1.1.2 Principles of SDR

While there is large variation in the design and components of SDR-based communication devices, all share certain basic features. The fundamental purpose of an SDR hardware device is to accurately capture and digitize RF signals from an antenna on the receive side and faithfully produce an analog version of the provided samples on the transmit side.

The simplest, most ideal embodiment of this concept is what we refer to here as the direct sampling radio, as shown in Figure 1.1. The basic concept is to have almost no analog components and instead to connect the data converters almost directly to the antenna. In a direct sampling radio, nearly all functions (frequency conversion, filtering, etc.) are performed in the sampled digital domain.

There have been some examples of successful direct sampling radios, but widespread adoption has not happened yet. This is due to a number of factors that work against this architecture, particularly on the receiver side. Since all channel filtering is performed in the digital domain, the entire spectrum that the radio covers needs to be sampled, not just the desired portion, in order to avoid aliasing. This leads to very high sample rates. The power consumed by analog-to-digital converters (ADCs) tends to be proportional to sample rate, so it is very difficult to make a low power radio with this architecture. At the same time, having a wide open receiver

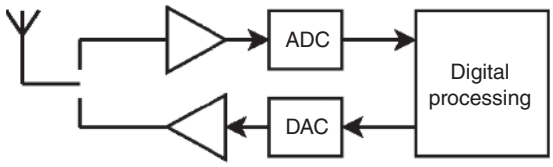

Figure 1.1 Direct sampling architecture.

means that any strong signals in the whole band can get into the ADC and cause clipping. The wider the band, the more likely there will be signals that are strong enough to cause these problems, so these types of radio are simply not yet practical for use in crowded spectrum, particularly in mobile devices.

Direct sampling radios (not to be confused with direct conversion) have seen considerable success in specialized areas like radio astronomy and high-frequency (HF, the spectrum between 3 and 30 MHz) radio. Radio astronomy observatories are often built in very remote locations to ensure that there will be very little terrestrially generated interference. That, combined with purpose-built frontend filters and generally noise-like signals, helps to limit the dynamic range of signals at the input to the radio, making direct sampling practical.

A more traditional radio architecture is the superheterodyne. While capable of high performance, these systems are harder to integrate into semiconductors due to the need for high-quality filtering at high frequencies. While there have been some superheterodyne SDR systems, they have largely fallen out of favor except in narrow band applications or in very high quality measurement receivers.

The direct conversion radio architecture has become the most popular for SDR as well as for most modern digital radio systems in general. The essential concept behind these is the concept of quadrature up- and down-conversion. In a receiver, the signals are handled at RF only to amplify and possibly apply a band filter before being converted to quadrature (or complex) baseband signals. The majority of the analog channel filtering and digitization is thus done at baseband where it can be easily integrated into semiconductor processes.

The advantage of direct conversion is in its simplicity, especially when used in a very wideband application. A superheterodyne system with more than an octave of bandwidth often entails multiple stages of frequency conversion and filtering, while a multioctave direct sampling radio would be exposed to large numbers of strong interfering signals, with corresponding dynamic range impact. A direct conversion radio, on the other hand, often only needs simple wide RF filters on the front to cover multiple octaves, with the rest of the radio handling the full range of frequencies.

The direct conversion radio is not without its disadvantages. For one, dual ADCs and digital-to-analog converters (DACs) are needed, as well as dual signal paths for the two components (I and Q) of the baseband. Direct conversion radios also suffer from a number of impairments [1], but most of these can be compensated for digitally. The most well known of these is quadrature gain and phase imbalance, which results from small mismatches in the mixers and baseband signal chains. Additionally, because the baseband signal paths must cover down to 0 Hz, small DC offsets in the chains can result in spurious signals centered on the band of interest. Due to the popularity of the direct conversion receiver, many OFDM[1]-based standards like LTE

[1] Orthogonal Frequency Division Multiplexing.

and 802.11a/g/n/ac actually avoid using the center section of the band for precisely this reason.

1.1.3 The USRP Story

In 2001, Eric Blossom started the GNU Radio project.[2] Its goal has always been to provide a framework for building SDR applications with Free Software.[3] It has attracted a large community of users and developers from around the world, and has become the design environment of choice for the Dynamic Spectrum Access research community.

While early work with GNU Radio was very successful, even producing a working ATSC HD television receiver, it was hampered by a lack of usable low-cost hardware. Most of the work was done with data acquisition systems and cobbled-together RF frontends using connectorized modules and evaluation boards. Early on, it became clear that to truly realize the potential of open source and SDR, low-cost hardware was necessary to give researchers and students access to the RF spectrum. In 2003, Matt Ettus started work on the Universal Software Radio Peripheral (USRP) to help lower the barrier of entry to SDR. In 2004, Ettus Research was started to develop, produce, and support the USRP, which was first released in 2005.

Since then the USRP has developed into a large family of related products, spanning a wide range of capabilities. USRPs have been used on all seven continents and over 106 countries. They have been used for research, teaching, experimentation, and deployed production-use systems. Applications have ranged from the common uses of RF spectrum to the rare and esoteric, including all of the following:

- Wireless networking
- Spectrum monitoring
- Dynamic spectrum access
- GSM, WCDMA, and LTE mobile telephony base stations
- RADAR
- Radio astronomy and RADAR astronomy
- Wildlife tracking
- RF test equipment
- Magnetic resonance imaging (MRI)
- Motion tracking
- Radio navigation, GPS, and GNSS
- Satellite communications
- RFID
- Wireless security research.

[2]http://gnuradio.org.
[3]http://www.fsf.org.

1.2 THE USRP FAMILY OF SDR SYSTEMS

1.2.1 USRP System Architecture

A block diagram for a full USRP system is shown in Figure 1.2.

There are three main hardware components in a USRP-based SDR system:

- A computer with a general-purpose processor (GPP)
- One or more USRP motherboards
- One or more RF daughterboards

The computer typically runs a standard (i.e., non-real-time) operating system (OS) like Linux, Microsoft Windows, or Apple's OS X. The user's SDR application software runs as a process on this OS and interfaces with the USRP hardware via the USRP Hardware Driver (UHD) library. This application software is often designed using an off-the-shelf SDR framework like GNU Radio, Simulink, or SCA, but it does not need to be. The application sets up and controls the whole system, and performs processing that has not otherwise been offloaded to the FPGA on the USRP motherboard.

The motherboard is the bridge between the computer and the analog world. The USRP motherboard connects to the computer via a high-speed asynchronous interface, such as Ethernet, USB, or PCI. All timing, control, sample rate conversion, and high-speed signal processing happen in the FPGA on the motherboard. Quadrature baseband digital samples are exchanged between the FPGA and the ADCs and DACs on the motherboard. The analog signals from the data converters, along with the control signals, are sent to the RF daughterboard. The capabilities of the various motherboards are summarized in Table 1.1 and described more fully in the following sections.

The RF daughterboard converts between analog baseband signals and RF. In Figure 1.2, everything between the antenna connections on the left up to and including the lowpass filters is contained on the RF daughterboards. This includes quadrature up- and down-converters, amplifiers, local oscillators (LOs), filters, and gain control.

1.2.2 USRP1

While the USRP1 has been superseded by newer designs from the second and third generations of USRPs, it is valuable to examine it to see the trajectory of technology over time in the area of SDR.

The primary design goal of the original USRP1, which was released in 2005, was to provide an extremely low-cost way to get access to RF spectrum from an off-the-shelf computer running a general-purpose OS. It connected to a host computer via USB 2.0 and was able to transport up to 16 MHz of RF bandwidth. It was capable of full duplex 2×2 MIMO operation. A picture is shown in Figure 1.3.

In the USRP1, digital processing and control functions are handled by an Altera Cyclone I FPGA. Because the Cyclone I is relatively small and has no "hard" multiplier units, the amount of digital signal processing (DSP) that can be performed in

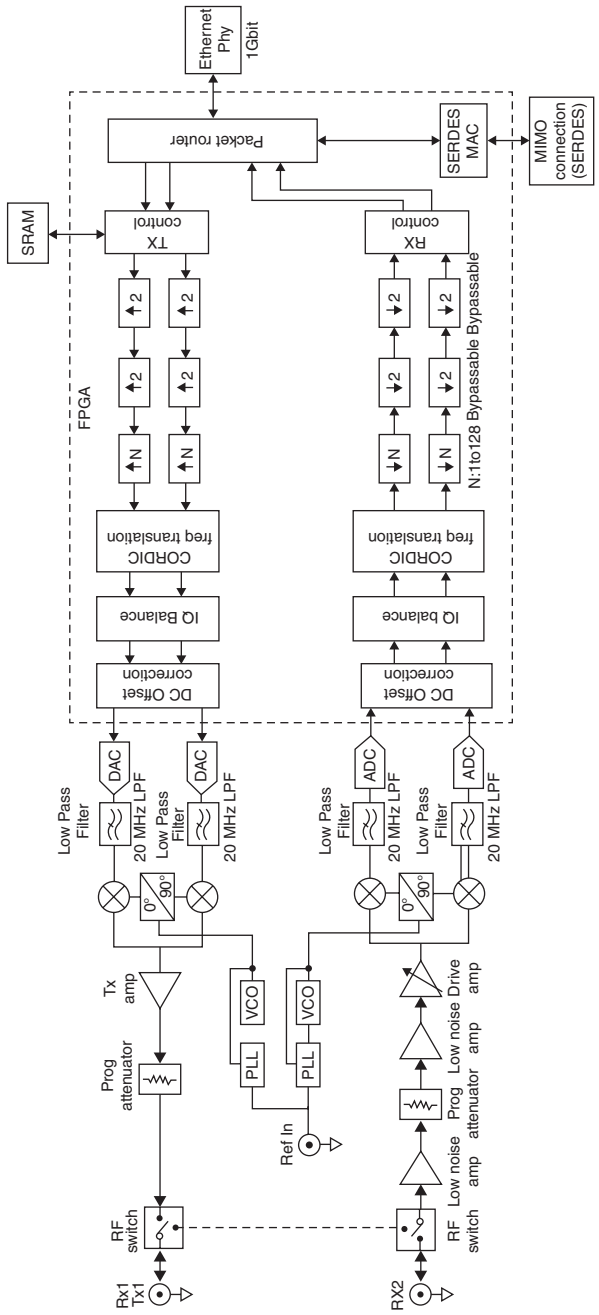

Figure 1.2 USRP N200 with transceiver daughterboard.

TABLE 1.1 USRP Motherboard Comparison

	N200/N210	B200/B210	X300/X310	E300
Interface to host computer	Gigabit ethernet	USB 3.0	1G/10G ethernet PCI Express x4	AXI4-MM interface to an embedded dual-core ARM Cortex-A9 processor
RF frontend	USRP daughterboards	Integrated RFIC	USRP daughterboards	Integrated RFIC
Instantaneous bandwidth	25–50 MHz	56 MHz	160 MHz	56 MHz
RF frequency coverage	DC to 6 GHz (determined by daughterboard)	70 MHz to 6 GHz	DC to 6 GHz (determined by daughterboards)	70 MHz to 6 GHz
MIMO	1×1 per unit, up to 8×8 using multiple units	B200: 1×1 B210: 2×2	2×2 per unit, up to 128×128 using multiple units	2×2
Full duplex	Yes	Yes	Yes	Yes
ADC	Dual 14-bit 100 MS/s	Dual/quad 12-bit 61.44 MS/s	Quad 14-bit 200 MS/s	Dual/quad 12-bit 61.44 MS/s
DAC	Dual 16-bit 400 MS/s	Dual/quad 12-bit 61.44 MS/s	Quad 16-bit 800 MS/s	Dual/quad 12-bit 61.44 MS/s
FPGA (see Table 1.3)	Xilinx Spartan 3A DSP	Xilinx Spartan 6	Xilinx Kintex 7	Xilinx Zynq 7
RFNoC compatible	No	No	Yes	Yes
Notes	Optional GPS-disciplined oscillator (GPSDO)	Optional GPSDO	Optional GPSDO	Battery powered, portable 9-Axis IMU, GPS, audio in/out

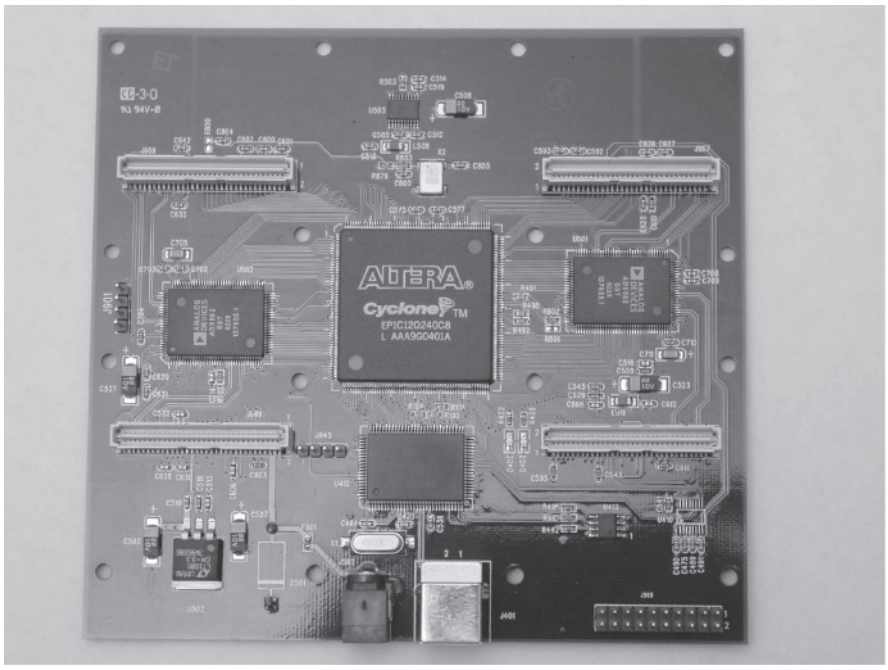

Figure 1.3 USRP1 motherboard. *(See color insert for representation of this figure.)*

the FPGA is relatively limited. Most of the DSP is done with algorithms that can be efficiently implemented without multipliers, like the Cascaded Integrator–Comb (CIC) filter for decimation and interpolation, and the CORDIC algorithm (COordinate Rotation Digital Computer) for frequency translation. In order to provide a flatter passband, in most cases the CIC filter roll-off effects are minimized by cascading them with halfband filters which use a minimum of multipliers, as those multipliers must be built with general logic resources.

1.2.3 Gen 2 USRP

In 2008, the first of several devices in the second generation of USRPs, the USRP2 was released (pictured in Figure 1.4 with an SBX daughterboard). It contained a Xilinx Spartan 3 FPGA, a gigabit Ethernet interface, and a connection to the standard USRP daughterboards (see Table 1.2). It has since been followed by the B100 (connected via USB 2.0) and the E100/E110 which has an embedded ARM processor making for a fully standalone system. The USRP2 was superseded by the USRP N200/N210 with a larger FPGA, a Spartan 3A-DSP from Xilinx. A block diagram of the N200 with RF daughterboard is shown in Figure 1.2. The structure of the other second-generation devices is similar.

Second-generation USRP devices all share a common core FPGA design which facilitates code reuse across the family. In addition to signal processing functions like

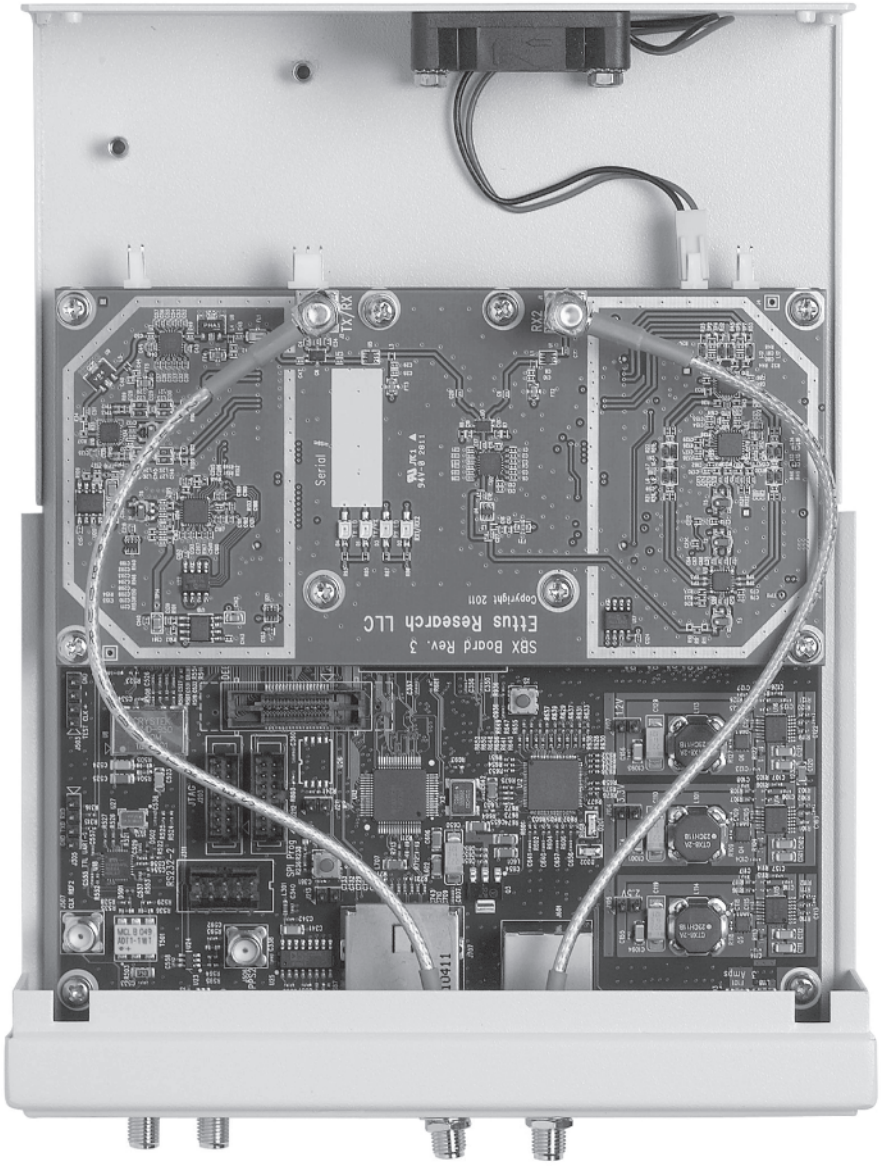

Figure 1.4 USRP2 with SBX daughterboard. *(See color insert for representation of this figure.)*

frequency translation and sample rate conversion, it contains control logic and space for user-specified DSP logic. The control logic communicates directly with the driver (UHD) and application on the host computer and allows for precisely timed streaming and control. While these devices each support only one simultaneous receiver and

TABLE 1.2 RF Transceiver Daughterboards

	WBX	SBX	CBX	UBX
Frequency range	25 MHz to 2.2 GHz	400 MHz to 4.4 GHz	1.2–6 GHz	10 MHz to 6 GHz
Full duplex	Yes	Yes	Yes	Yes
RF bandwidth	40 or 120 MHz	40 or 120 MHz	40 or 120 MHz	40 or 160 MHz
TX output power	17–20 dBm	16–20 dBm	14–17 dBm	15–18 dBm

transmitter, multiple devices can be tied together to implement MIMO operations. The USRP2, N200, and N210 have a "MIMO port," which is used to create a 2×2 MIMO system with two devices and one cable. With the addition of external clock sources and distribution system users can make MIMO systems as large as 16×16.

1.2.4 Gen 3 USRP family

The third generation of USRP devices is focused on high bandwidth, scalability for Massive MIMO applications, and easier development of FPGA-based processing. The first two devices in the family are the X300 series and the E300, released in 2013 and 2014, respectively. All devices in the third generation support the use of RFNoC, which is described further in Section 1.4.

1.2.4.1 X300-series The X300 and X310 (which differ only in the size of their FPGA) (Fig. 1.4) have dual 10 Gigabit Ethernet interfaces for high bandwidth, but they can also drop down to 1 Gigabit Ethernet for applications which don't require as much bandwidth. The dual interfaces can be channel-bonded to double bandwidth, or the second interface can be used to daisy-chain the devices, allowing for scaling beyond 2×2 MIMO without a need for Ethernet switches or other external hardware. Alternatively, the PCI Express x4 link allows for direct connection of the X300 over a cable to the bus of a host computer with extremely low latency (Figure 1.5).

The X300 series was designed with scalability in mind. It is already being used in 100×100 MIMO systems, with nearly all of the OFDM and MIMO processing being performed in the FPGAs on the individual radios.

Figure 1.5 USRP X300-series device.

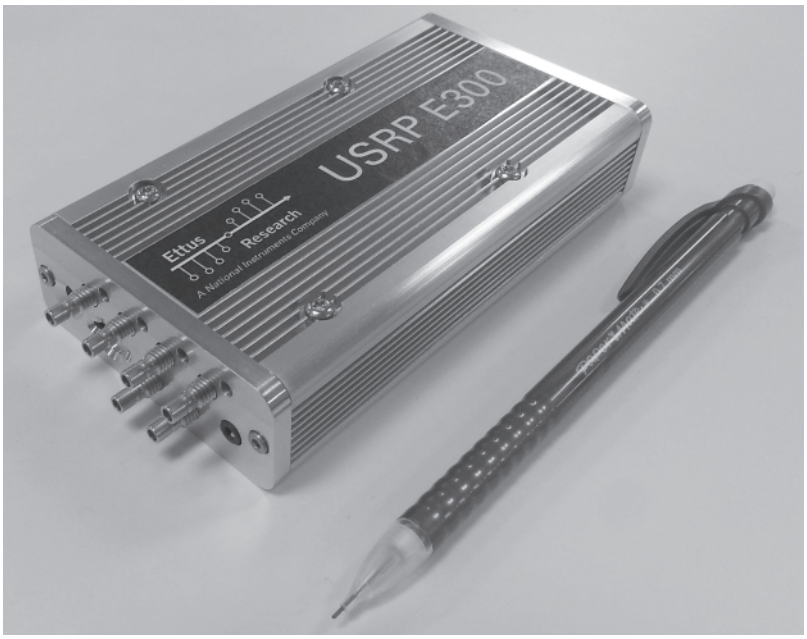

Figure 1.6 E300 (with pencil for size comparison).

1.2.4.2 E300 The USRP E300 integrates a 2×2 MIMO radio system with a complete embedded computer running Linux in a cellphone-sized form factor. Thus, it is fully self-contained, and there is no need for a separate computer. It is based on the Zynq 7020 FPGA from Xilinx, which includes a dual-core ARM Cortex-A9 processor with a high-speed interface to the FPGA fabric. It uses a standard UHD driver to allow it to work with the same applications as any of the other USRPs, and it has RFNoC compatibility to augment the processors' computational capacity.

In addition to the radio, the device also includes a number of peripherals to enable interesting applications. These include a GPS receiver (for timing, position, and frequency accuracy), a full 10-axis inertial measurement unit (IMU, including accelerometers, gyroscopes, magnetometers, and altimeter), an audio codec, and USB ports for external peripherals (Figure 1.6).

1.2.5 RF Daughterboards

The USRP1 established the standard interface for interchangeable daughterboards which also work with the second- and third-generation devices. In the early days of USRP1, wideband integrated quadrature mixers and voltage-controlled oscillators (VCOs) were not available off the shelf, and so changing bands often meant changing daughterboards. The newer daughterboards that are available today (WBX, SBX, CBX, and UBX, see Table 1.2) are based on newer components, and so are able to cover much wider bands. For most applications, the user should never need to switch daughterboards.

1.3 UHD

Since the intended use of a USRP is to control it from a host PC and use it from any application, the software controls (or drivers) for these devices also play an important role in the overall USRP architecture. The first generations of USRPs came with software APIs tailored specifically toward these devices, but it quickly became evident that this approach would not scale well with future development, and it also meant software had to be changed when switching between device types, which contradicts the ideal of a SDR.

To solve these problems, the UHD was developed. UHD is the software package which is used to control all of the devices in the USRP family of products. It is an open source library which runs on Linux (including embedded), Windows, and Mac OS X, implemented in C++. UHD is licensed both under the GNU General Public License (GPL v3), and is also offered under an alternative license for situations in which the GPL is not appropriate. It provides an intuitive API which has all of the controls and operations necessary to use all of the features of the devices, while at the same time abstracting away the low-level implementation details of the hardware. Despite their widely varying capabilities, bandwidths, interconnection methods, and form factors, all of the devices share this same API, and so in most cases programs which work with one type of USRP will work with the others without significant changes. These capabilities have helped to make UHD a de facto standard in the industry.

Numerous applications directly target UHD to interact with USRPs, including OpenBTS (an open-source GSM base station system), Amarisoft LTE100 (a full LTE eNodeB base station application), and SDRX (a ham radio application). Additionally, many application frameworks have been linked with UHD, allowing USRPs to be used from within those development systems. The most commonly used are GNU Radio and LabVIEW, but Matlab, Simulink, IRIS, ALOE, OSSIE, and RedHawk also support UHD.

Beyond addressing and connecting to devices, the most important operations provided by UHD fall into two classes: control and streaming (Figure 1.7). Control operations are used to set frequencies, sample rate, gains, and any other settings required. All functionality of the device can be accessed through control methods, and

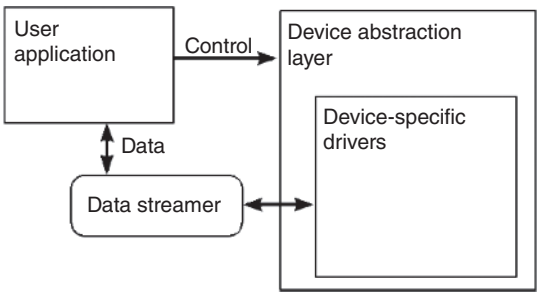

Figure 1.7 A simplified overview of a UHD application setup.

```
/mboards/0/fw_version
/mboards/0/eeprom
/mboards/0/link_max_rate
/mboards/0/fpga_version
/mboards/0/name
/mboards/0/codename
/mboards/0/rx_codec/
/mboards/0/rx_codec/A
/mboards/0/rx_codec/A/name
/mboards/0/rx_codec/A/gains
/mboards/0/tx_codec
/mboards/0/tx_codec/A
/mboards/0/tx_codec/A/name
/mboards/0/tx_codec/A/gains
/mboards/0/time
/mboards/0/time/cmd
/mboards/0/time/now
/mboards/0/time/pps
/mboards/0/auto_tick_rate
```

Figure 1.8 Sub-section of the property tree for a USRP. Individual settings such as revision, gains etc. can be queried and set through this interface.

there are mechanisms to extend these capabilities through a *property tree* interface, allowing for new features on future devices. This property tree is a hierarchical list of components and settings on a USRP, with an interface to change or query values (Figure 1.8).

Streaming operations are done through a *streamer interface*, which allows applications to connect to the sample streams of USRPs through an abstraction layer. UHD takes care of streaming details such as flow control and conversion to a specific data type, which allows the developer to focus on simply passing and receiving sample data from the UHD object. If the user wants to handle samples as complex floating-point types, the streamer object will take care of this, regardless of what type the device uses internally (most devices describe samples as 16-bit integers for each in-phase and quadrature part). The same is true for MIMO operations: Using a streamer object, UHD guarantees that sample streams will arrive in a time-aligned fashion.

The internals of the device driver are geared specifically toward the available devices. Different devices use different connections (such as USB, Ethernet, or PCIe), and their initialization also differs. These hardware-specific details are hidden away by the abstraction layer, which is how UHD-based applications can work with different devices without major revision. On a side note, in an RFNoC-capable device (see the following section), UHD also allows access to individual computation engines implemented inside the FPGA.

As an example, consider how GNU Radio uses UHD: It ships with a subcomponent called `gr-uhd`, which maps the UHD API into the GNU Radio component model. This exposes all of the controls and features of the device to users of GNU Radio, including when used with the GNU Radio Companion graphical design environment.

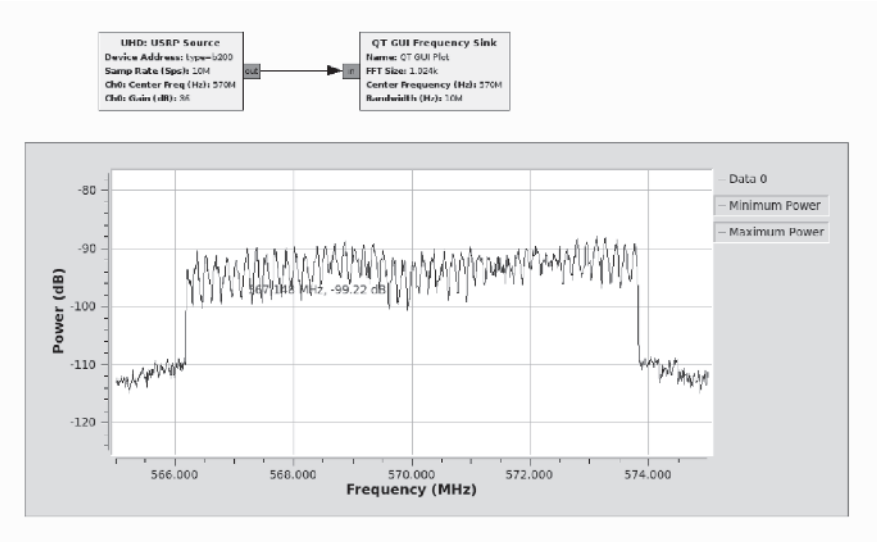

Figure 1.9 A simple GNU Radio flow graph with a spectrum analysis output, operating on a DVB-T channel.

Figure 1.9 shows a simple example of such a setup, where GNU Radio's components are used to present a simple spectrum analysis tool. In GNU Radio terminology, a "USRP Source Block" is a component which interfaces with a UHD-capable device to introduce samples into a GNU Radio application, from where it can be streamed to other components. Note that the Source block only has few settings, such as frequency and gain, which set up the block regardless of the actual device connected to the host PC, as long as the connected device supports the chosen values (Figure 1.9).

Users of GNU Radio thus don't even have to know about any details of UHD, making the access to USRPs even simpler. However, the open source nature of UHD allows anyone to easily incorporate UHD into their own applications, and an active community around UHD facilitates its usage further.

1.4 RFNoC

1.4.1 Introduction to RFNoC

Digital computational resources for communication systems, including those found in FPGAs, have scaled roughly in line with Moore's Law, as expected, and thus ever more complex systems may be realized with them. This is illustrated in Table 1.3, which shows the relative FPGA resources available on the various USRP devices over time. At the same time, the complexity of designing those systems has been rising as well. Certainly, human and financial resources for development are not scaling as quickly, and so some means of dealing with this gap must be found.

TABLE 1.3 USRP-Family FPGA Size Comparison

Generation	First	Second	Third	Third
Device	USRP1	USRP N210	USRP E300	USRP X310
Year introduced	2005	2010	2014	2013
FPGA	Altera Cyclone	Xilinx Spartan 3A	Xilinx Zynq	Xilinx Kintex 7
Logic cells	12K	53K	85K	406K
Memory	26 KB	252 KB	560 KB	3180 KB
Multiply units	0	126	220	1540
Clock rate	64 MHz	100 MHz	200 MHz	250 MHz
Total RF BW	8 MHz	50 MHz	128 MHz	640 MHz

In the software world, this increase in capability and complexity has traditionally been handled with a combination of design tools (like higher level languages and automatic code generation) and code reuse (through shared libraries and the like). The FPGA design world has seen numerous efforts at the former (System Verilog, LabVIEW FPGA, Bluespec, etc.), but the latter has not been as successful. Although there have been efforts, like various intellectual property (IP) interconnection standards, FPGA vendor libraries, and the OpenCores project, the reality is that much of FPGA-based design involves reinventing the wheel over and over again. It is precisely this problem that led us to create RFNoC.

RFNoC (RF Network-on-Chip) is a system aimed at creating a design process in the FPGA which is similar to the block diagram-oriented GNU Radio software development methodology. In particular, RFNoC is organized around the concept of a distributed network of individual computational elements ("CEs") and radio frontends ("radios"), which communicate using a network that (despite the NoC name) can be both on-chip and between devices across a larger switching fabric, such as 10 Gigabit Ethernet or PCI Express. All CEs and radios share a common interface that facilitates reuse and automatic reconfiguration. RFNoC borrows some concepts from RapidIO, including the addressing system, but is much more aimed at small CEs and highly efficient implementation in an FPGA.

The end goal of RFNoC is to look at large FPGAs and other computational resources as a distributed system of flexibly interconnected components, which can be used as needed. By getting rid of fixed routing, one can get beyond treating FPGAs as rewritable ASICs.

1.4.2 RFNoC Principles

Computational and radio elements in RFNoC are known as "blocks." The abstraction used for communication between blocks is the FIFO, so RFNoC is essentially an implementation of Kahn Process Networks [2]. The infrastructure of RFNoC enables the building of large-scale Process Networks across real, practical networks, both within and between chips.

RFNoC has been carefully designed to be simple and efficient, yet highly scalable and stable. All components in RFNoC may communicate with each other

asynchronously, and common clocks are replaced with time stamps to allow for precisely synchronized events. The network is packet-switched and not circuit-switched, allowing for dynamic data-dependent routing. It is able to carry any form of data, not just timed samples, and both control and data are carried over the same paths. All endpoints are considered equal, and there is not a need to have a designated "master" or "host" to coordinate. Any block can control any other block simply by sending the appropriately formatted control packet.

In large networks of distributed asynchronous systems, it can be hard to predict latency and congestion, and to provide guarantees of real-time performance. It is for this reason that we have added explicit end-to-end flow control to the network. It works on the principle that no endpoint may send data into the network unless the destination endpoint has enough buffer space to hold it. This guarantees that packets will not get stuck in the network fabric itself while waiting to be consumed. This explicit flow control mechanism in combination with other design elements allows one to accurately predict worst-case latency and guarantee no congestion in a properly designed system. This allows for automated reasoning that can provide a check that a design is practical.

1.4.3 RFNoC Components

There are four main types of components in an RFNoC system: the CE, the Radio, the Radio Transport Router, and the External Interface (EI). The CEs are what provide the impetus for using RFNoC, so we will discuss those first. A CE can be thought of as an independent processing device, and while most will be in FPGAs, it is also possible to create CEs in general purpose processors, GPUs, or anything else which is able to communicate via the RFNoC protocol. The CE will receive some number of streams of packets in and produce a number of output streams. What the user does inside is not constrained, so one could imagine building FFTs, FIR filters, equalizers, AGC state machines, protocol state machines, error-correction codes, and the like. Because of latency and overhead issues, it is not very efficient to create very simple CEs like adders or multipliers, so there is a soft lower bound on the minimum complexity one would want in a CE. On the other end of the spectrum, while one could implement an entire OFDM receive chain inside a single CE for example, that would not be leveraging the strengths of RFNoC, such as code reuse.

The Radio component of RFNoC contains all digital aspects of the interface to the actual hardware radio. In the case of a USRP X300, this would include the connections to the ADCs and DACs and the hardware controls like gain and switching. Everything which must happen in the clock domain of the radio hardware must happen within this block, so all time-stamping and time-critical controls are performed here. Received samples from the ADC are processed with IQ balance and DC offset corrections, possibly filtered and decimated, and packed into time-stamped packets before being sent into the RFNoC network for further processing. On transmit the reverse happens, with time-stamped packets of samples coming in, unpacked, processed, and sent to the DACs at the precise time specified.

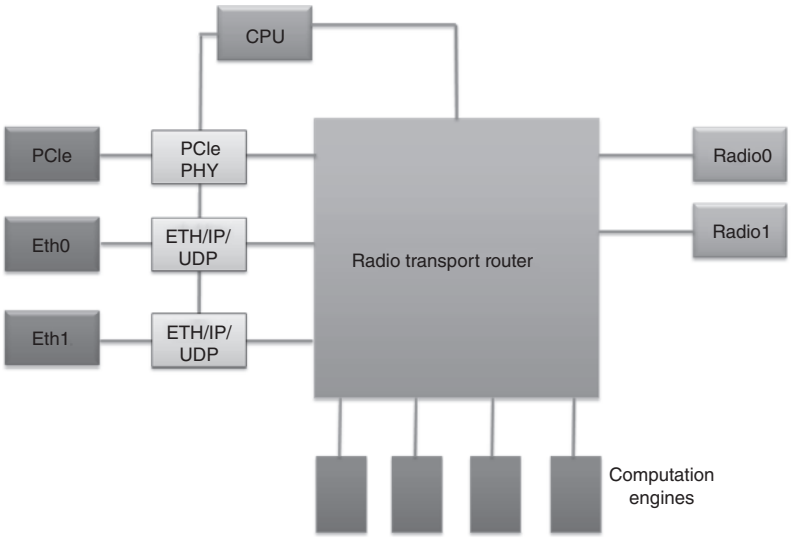

Figure 1.10 A typical RFNoC-enabled device, the X300-series.

The Router is the part of the system that connects all of the other blocks together. Based on the address of a packet, it decides to which port it should be sent. It is implemented as a full crossbar, which can be efficiently implemented with up to 16 ports in a Xilinx 7-Series FPGA. If more ports within an FPGA are needed, multiple Routers can be instantiated. When more than one source port has traffic for the same destination port, packets are accepted from each source in a round-robin fashion. When building larger multi-FPGA designs, the individual Routers can be connected together either by direct Ethernet connections between the USRP devices (known as daisy-chaining, as shown in Figure 1.11), or an Ethernet switch can be used.

The EIs allow RFNoC to bridge to other components outside the FPGA, and their purpose is to pass RFNoC protocol packets. The Ethernet EI, on packet ingress, takes incoming packets and strips off Ethernet, IP, and UDP protocols, and passes pure RFNoC packets into the Router. On egress, the EI looks up the destination RFNoC address of each packet and adds the appropriate Ethernet, IP, and UDP headers to get it there. Packets which have contents that are not RFNoC protocol are passed to a small auxiliary CPU to handle auxiliary tasks like ARP, Ping, and device configuration. EIs for Ethernet, PCI Express, USB3, and Zynq/ARM interfacing have been built, but one could imagine others to bridge to other interconnects like RapidIO or other memory buses.

The structure of a typical RFNoC-enabled device, the X300-series, is shown in Figure 1.10. All links shown are bidirectional FIFOs, each 64 bits wide, and the FIFOs carry streams of packets rather than streams of samples. These FIFO interfaces use the AXI4-Stream format, an open standard developed by ARM and adopted widely in the industry by companies like Xilinx. This allows for relatively easy integration

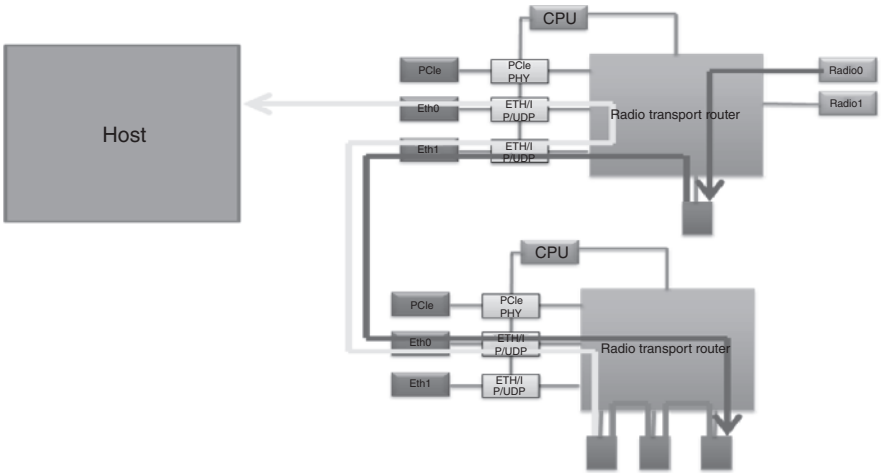

Figure 1.11 X300 RFNoC FPGA structure.

of existing IP blocks like those found in the Xilinx Coregen tool, which includes FIR filters, FFT units, error correction coding blocks, and the like.

1.4.4 Example RFNoC Application for OSA

In this section, we illustrate how one might build an application for OSA using RFNoC. A basic OSA radio implementation would be required to sense the spectrum, detect active signals, tune the radio to a free channel, and then transmit and receive OFDM with its communication partner. To do this, we can put together a library of the following blocks in the FPGA:

- Variable length FFT/IFFT
- Schmidl and Cox OFDM synchronization
- Finite impulse response (FIR) filter
- Vector windowing
- PSK symbol mapping
- PSK symbol demodulation
- Forward error correction (FEC) encode
- FEC decode
- Sample rate converter
- Energy detector
- Embedded processor.

When performing spectrum sensing, the flow of data would be

radio → vector windowing → FFT → energy detector → embedded processor.

TABLE 1.4 RFNoC Addressing

Bits	Function
15-8	Subnetwork number
7-4	Crossbar port
3-0	Logical block port

The embedded processor, upon seeing a free channel would send control packets to the transmit side of the radio to tune to a free channel and then would send control to the FIR filter, sample rate converter, and FFT to set up for transmission parameters which fit in the channel. Transmission flow would then be

processor → FEC encode → PSK symbol mapping → IFFT → vector windowing → FIR filter → sample rate conversion → radio.

For reception, flow would be

radio → sample rate conversion → FIR filter → FFT → PSK symbol demodulation → FEC decode → embedded processor.

As can be seen from this example, the RFNoC architecture provides for a lot of flexibility and allows for a lot of code and resource reuse. In particular, the FFT/IFFT, vector windowing, FIR filter, and sample rate conversion are all used more than once. Additionally, multiple different FEC blocks could be present and chosen dynamically based on the application.

1.4.5 RFNoC Addressing

In order to route packets in the RFNoC network, flow endpoints are given 16-bit addresses. A data flow is identified by the 32-bit tuple of (source, destination). This facilitates automatic responses and flow control, since an endpoint can simply reverse the tuple to send a packet back to the source.

The 16-bit addresses are further broken down hierarchically in order to simplify routing, as shown in Table 1.4. The upper eight bits specify the network switch to which the endpoint is connected, the next four specify to which port on that switch the endpoint is connected, and the lowest four are used within the endpoint as a logical port. By having up to 16 multiple logical ports in a single endpoint, one can differentiate various types or purposes of data, similar to the multiple ports that each GNU Radio block can use. For example, different logical ports in a block can be used to select different filters in a generic FIR engine block (Table 1.4).

1.4.6 Compressed Header (CHDR) Packet Format

The compressed header format (CHDR) considers characteristics of existing protocols, as well as the benefits and limitations imposed by a 64-bit AXI4-Stream.

TABLE 1.5 Compressed Header – 64-bits

Bits	Function
15-0	Destination address
31-16	Source address
47-32	Packet size (bytes)
59-33	Sequence number
60	1 = end-of-burst
61	1 = has time
63-62	Packet type 00 = data 01 = flow control 10 = control/command 11 = response/error

Through judicious allocation of bits, it is possible to get all the information necessary in the header into a single 64-bit word. This header (see Table 1.5) can express the stream source and destination, packet length, type, and provide a frame count (Table 1.5).

Expressing this information with single, 64-bit words allows muxes, demuxes, and other elements to easily parse and produce data without complex state machines. This improves throughput and facilitates timing closure in the FPGA build process.

For those packets that contain them, the timestamp is a single 64-bit word immediately following the header. The remainder of the packet is the data payload. Since the format of the payload is not constrained by RFNoC, it is up to the application designer to ensure that each block gets the data format it is expecting on each of its ports. The most common data type is I/Q data tuples where both elements are of 16-bit two's complement type.

1.5 CONCLUSION

Research in dynamic and OSA is opening new and exciting doors to increased spectrum efficiency and better utilization of scarce resources. The USRP family of products is aimed at providing the capabilities needed to enable this research and to deploy these systems in the real world. The third generation of USRP devices provides a path to high-bandwidth, low-latency MIMO communication systems, while simplifying development. The combination of GNU Radio, UHD, and RFNoC allows users to quickly implement complex modulation and protocol systems without needing to deal with the low-level details of hardware control and FPGA design, allowing them to concentrate on their respective areas of expertise.

REFERENCES

1. Loke A, Ali F. Direct conversion radio for digital mobile phones-design issues, status, and tends. IEEE T Microw Theory Tech 2002;50(11):2422–2435.
2. Kahn G, The semantics of a simple language for parallel programming, In Proceedings of the IFIP Congress 74. North-Holland Publishing Co., 1974.