A Software Defined OFDM Modulator

by

Jeffrey M. Fifield

B.S., University of Michigan, 2000

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Master of Science

Department of Computer Science

2006
This thesis entitled:
A Software Defined OFDM Modulator
written by Jeffrey M. Fifield
has been approved for the Department of Computer Science

Prof. Dirk Grunwald

Prof. Douglas Sicker

Prof. Timothy Brown

Date _______________

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.
Fifield, Jeffrey M. (M.S., Computer Science)

A Software Defined OFDM Modulator

Thesis directed by  Prof. Dirk Grunwald

A software defined radio (SDR) is a radio system that allows a great degree of reprogrammability. In a SDR, all signal processing is performed by or closely controlled by software. Because of this, proponents of SDR promise dynamic and intelligent protocols for wireless networks. Orthogonal Frequency Division Multiplexing (OFDM) has emerged as a popular modulation technique for such networks. Consequently, it is desirable to make software defined OFDM available. If possible, such a radio system should be based on the PC platform. Existing solutions for OFDM on a PC based platform lack the performance or the flexibility needed for true software defined OFDM.

This thesis presents the design of an OFDM transmitter for software defined radio. The transmitter is implemented on an FPGA as part of a PC based software radio platform. To demonstrate the programmability of the FPGA design, IEEE 802.11a and several other applications are demonstrated.
Dedication

To Ullr.
Contents

Chapter

1 Introduction 1

2 Related Work 5
  2.1 FPGA Software Radio Platforms 5
  2.2 Software OFDM Implementations 6

3 Orthogonal Frequency Division Multiplexing 8
  3.1 OFDM Details 8
    3.1.1 Modulation 9
    3.1.2 Channel Coding 11
  3.2 OFDM Applications 11

4 Design 14
  4.1 Platform 15
  4.2 Design Decisions 17
  4.3 A General Purpose OFDM Modulator 19
    4.3.1 Overview 19
    4.3.2 Implementation 23

5 Example Application: IEEE 802.11a 25
  5.1 The 802.11a Physical Layer 25
5.2 Software 802.11a Modulation

6 Other Applications

7 Conclusion

Bibliography
Tables

Table

4.1 The performance of software IPFT compared to an example OFDM application (2.2GHz AMD Athlon 64 Processor) ........................................ 18
4.2 OFDM modulator parameters ................................................................. 20
5.1 Data rates and related parameters for IEEE 802.11a .................................. 26
5.2 IEEE 802.11a PHY timing parameters ................................................... 26
5.3 Required rate vs. achieved rate for a software implementation of a 802.11a modulator (100 and 1500 byte packets, 2.2Ghz AMD Athlon 64 Processor) 28
5.4 Required rate vs. achieved rate for a software implementation of a 802.11a modulator without subcarrier modulation, IFFT, or guard interval insertion (100 and 1500 byte packets, 2.2Ghz AMD Athlon 64 Processor) . . 29
Figures

3.1 Typical OFDM Transmit Processing ........................................... 9

4.1 Software Radio Platform .......................................................... 15
4.2 Software Radio Platform - FPGA Design ....................................... 16
4.3 Example OFDM modulator input packet ....................................... 22
4.4 Screenshot of the OFDM modulator design .................................... 22

5.1 IEEE 802.11a Test Setup .......................................................... 30

6.1 Non-contiguous transmission of two 802.11a frames ....................... 33
6.2 Waterfall plot of frequency agility. The X axis is frequency and the Y
axis is time. Signal strength is indicated by color .............................. 33
Radio devices have traditionally been implemented using only analog hardware components. Once built, the functionality of such a system is fixed and can not be easily changed. With the advent of powerful digital signal processors (DSP), field programmable gate arrays (FPGA) and general purpose processors (CPU), it is now possible to implement many signal processing tasks in software. This has given rise to the field of Software Defined Radio (SDR). A software defined radio can be defined as a device that is substantially defined in software and whose physical layer behavior can be significantly altered through changes to its software [1]. The advantage of SDR is that changes to the radio require mainly software modifications and few, if any, hardware modifications. Taken to the extreme, a SDR consists of a programmable processor directly connected to analog-to-digital (A/D) and digital-to-analog (D/A) converters which are in turn connected to an antenna. In such a radio, all signal processing is performed in software and the software is, in effect, directly connected to the antenna.

Due to constraints such as cost, performance and ease of programming, purely software radio systems are not always possible. As a result, many SDR systems end up being some combination of CPUs, DSPs, FPGAs and custom hardware. These systems are often powerful, but are not always easy to program. Arguably the easiest platform to program is the CPU based personal computer (PC) with its vast body of programming tools and expertise. Enabling SDR on the PC has been the goal of a number of projects.
Although the performance of the PC continues to increase, so do the demands of modern signal processing applications. Thus, the PC platform using only CPUs is not powerful enough for many applications. In this work, we augment the CPU with a FPGA based signal processing peripheral connected to the PCI bus of a standard PC. By providing appropriate FPGA designs one can increase the overall signal processing capability of the PC platform while maintaining programmability and ease of use.

There are a few different ways to achieve the reprogrammability required by SDR in an FPGA. The brute force method is to provide a static FPGA design for every configuration that might be encountered by the radio. If the number of possible configurations is small, then this may be a viable approach. For true general purpose SDR, however, enumerating every possible design is intractable. Another problem is that reconfiguration can take several milliseconds to several seconds depending on the FPGA device and its connection to the rest of the system. This is too slow for many applications. A similar solution to this kind of FPGA problem is called partial reconfiguration. Using partial reconfiguration, portions of the FPGA device can be reconfigured while the device is running. This is an attractive option because one could design a number of small signal processing blocks with well defined interfaces and connect them together at runtime on the FPGA. This is similar to the design style of software systems like GNU Radio where small units of functionality are logically connected together at runtime. While partial reconfiguration is more general and makes more sense than the brute force method, reconfiguration still takes some amount of time and samples need to be buffered while reconfiguration takes place. This makes it difficult to reconfigure a FPGA signal processing application on the fly without introducing interruptions into the data stream. Yet another approach is to design FPGA processing blocks that are themselves programmable. In band or out of band signaling can then be used to change the behavior of the processing block while it is processing data. This kind of processing block could still be used in conjunction with partial reconfiguration, but such reconfiguration
would become necessary only for significant changes to the processing pipeline.

A typical programming model for SDR is the data flow model. In this model, functional blocks are nodes of a graph with streams of data flowing along the edges of the graph. This is also representative of the way that signal processing is often visualized (e.g. Figure 3.1) or implemented in custom hardware. Although this architecture is suitable for many applications, some people who want to use SDR are interested in complex packet radio applications. It is difficult to express packet based applications in the stream based data flow model. For example, the 802.11 protocols often require very different modulation during different portions of the transmission of a single packet. In data flow systems it can be difficult to tell the runtime system where such modulation changes should occur in the data stream and how to reconfigure the flow graph appropriately. An alternative to the stream based data flow model is a hierarchical frame based model. In this software architecture, entire packets of data are passed between processing elements along with meta-data describing how that data should be processed. Processing elements can then process portions of the packet as required by the meta-data or pass subsets of the packet to children processing elements in the hierarchy before producing another packet as its output. There is an effort underway in the GNU Radio project to implement such a processing model on top of the current data flow model [5, 6].

Orthogonal Frequency Division Multiplexing (OFDM) is a popular modulation choice for wireless networks. Therefore, research involving such networks will benefit from software defined OFDM. Today most implementations of OFDM realize the IEEE 802.11a/g standards for local area networks. These transceivers are often based on application specific hardware and allow for little modification of modulation parameters. Application specific hardware is used because the computational requirements of OFDM are more than most general purpose systems can provide. For a software defined OFDM platform, a more programmable solution is needed. A goal of this thesis is to build a
programmable OFDM transmitter for the PC platform. Because of the computational costs, some portion of OFDM must be moved to application specific hardware. By carefully designing both the hardware and the hardware/software interface, it is possible to retain the programmability of a purely software system while gaining hardware acceleration.

We present a hybrid CPU and FPGA design for the implementing OFDM transmitters on the PC platform. The FPGA design is general purpose and programmable with a simple application programming interface (API) for communication between the CPU and FPGA portions of the design. With this design, we improve the performance of the PC platform in the OFDM design space without significant extra cost. This is accomplished while maintaining complete software control over modulation and the relative ease of programming a CPU based system. Furthermore, we have designed our API such that the design is compatible with both the stream based data flow programming model and the frame based hierarchical programming model. Conceptually, our API is most like the frame based design style. Structured meta-data is attached to the data stream describing the modulation that will be used with that data.

The rest of this thesis is organized as follows. First, Chapter 2 describes work related to this one. In Chapter 3 we discusses OFDM, its applications, and its processing requirements in greater detail. Chapter 4 describes the software radio platform and FPGA based OFDM modulator design developed for this thesis. In Chapter 5 we review the 802.11a physical layers and present a validation of our design in the form of a working 802.11a transmitter. Two more examples of the flexibility of our platform are shown in Chapter 6. Finally, Chapter 7 concludes.
Chapter 2

Related Work

This section reviews previous work on FPGA based software radio platforms and software radio systems for OFDM.

2.1 FPGA Software Radio Platforms

There have been a number of FPGA and software solutions reported for software defined radio and OFDM systems. Cummings et al[7] and Reed [1] report on the architecture of FPGAs and the general applicability of FPGAs to software defined radio.

Reves et al[8, 9] focus on a FPGA hardware abstraction layer for software defined radios. This middleware, called the Platform and Hardware Abstraction Layer, places a standard API between radio software and any algorithm kernels placed in the FPGA. The API aims to provide flexibility and portability by defining an interface common across different applications and hardware.

Some FPGAs have the ability to be reconfigured while they are running. This is known as partial reconfiguration. Coulton et al[10, 11] seek to enable fine grained partial reconfiguration in FPGA’s for software radio platforms. The authors demonstrate their system by implementing a partial 802.11a transmitter where the puncturing step of the convolutional coder can be reconfigured on the fly.

Work by Dick et al[12] demonstrated an OFDM physical layer implemented in an FPGA. Their design implements OFDM modulation/demodulation and receiver syn-
chronization algorithms for 802.11a. As in this work, the Simulink and System Generator toolkits were used as fast implementation tools.

Jameson et al[13] and Di Stefano et al[14] also present FPGA based platforms for software defined radio. The former specifically targets OFDM, while the latter has been demonstrated by implementing 802.11b and 802.15.4, which are both PSK based modulation standards.

The WARP project at Rice University [15] is another FPGA based development platform for software defined radio. The project has chosen OFDM as the physical layer of choice, and they have implemented a full OFDM transmit and receive chain in the FPGA. They also make use of the PowerPC processors embedded in the Xilinx FPGAs. The WARP platform is programmed using Simulink and System Generator.

Harada reported a FPGA based system for W-CDMA and 802.11a in [16]. The prototype includes a FPGA board with four FPGAs, a CPU board with two 240MHz CPUs and a radio board operating at 2.4GHz or 5GHz.

One of the most popular software radio systems today is the GNU Radio system [4]. The hardware most often used with this system is the Universal Software Radio Peripheral (USRP) [17]. The USRP is motherboard containing a FPGA, A/D and D/A converters. It has four expansion slots accepting two transmit and two receive daughterboards configured for different frequency ranges. The current version of the USRP connects to a PC via USB 2.0 and provides a small amount of processing on the FPGA. Future versions will contain larger FPGAs that can support additional processing and will faster gigabit ethernet to connect to a PC [5].

2.2 Software OFDM Implementations

There have been several efforts to implement OFDM in software for execution on a CPU or DSP.

Tariq et al[18] reports on an implementation of 24Mbit/sec 802.11a on a system
composed of a PC plus an accelerator board containing a DSP processor. Although their system could achieve 24Mbit/sec during burst transmissions, it could only achieve a sustained rate of 1.7Mbit/sec at the application layer.

Meeuwsen et al[19] were the first to report a full rate implementation of an 802.11a transmitter. Full rate is defined as being able to modulate data at or above the data rate of the modulation being used (e.g. > 54Mbit/sec for 54Mbit 802.11a). Their software is targeted to run on a novel DSP architecture called an Asynchronous Array of simple Processors (AsAP).

Another novel processor architecture for DSP called SODA was reported by Lin et al[20]. SODA is a low power software radio specific processor architecture targeting mobile applications. Using this design, the authors were able to implement an IEEE 802.11a transmitter and receiver running at 24Mbit/sec.

Previous work has demonstrated the feasibility of FPGA based SDR, FPGA based OFDM and SDR based OFDM. The work presented in this thesis is a significant extension of prior work in several ways. Our FPGA design is generalized and programmable, able to implement a wide variety of OFDM protocols, not just one example. We also retain software control over the modulation process on a general purpose CPU, using a well defined and simple API, despite the fact that modulation occurs on the FPGA. Finally, our example 802.11a software is of high quality and we know of no other publicly available implementation written for general purpose processors. We believe it will port cleanly to GNU Radio when that package has sufficient support for frame based computation.
All wireless data communication systems use some sort of modulation technology to encode digital data onto analog radio waves for transmission over the air. At the receiver, demodulation is performed to recover the original information. Typically data is encoded by changing one or more characteristic of the carrier radio signal such as frequency, amplitude or phase. One such modulation technique is orthogonal frequency division multiplexing (OFDM). In OFDM, the data to be transmitted is split up into some number of parallel data streams (subcarriers). These subcarriers are individually modulated using a low rate modulation technique and transmitted in parallel using carriers at different frequencies. Although the subcarriers are individually carrying data at a low rate, the aggregate throughput of all subcarriers can be high. With appropriate spacing, the subcarriers can be placed very close together with no interference between them.

### 3.1 OFDM Details

Figure 3.1 shows a typical OFDM transmitter. The functions of the OFDM transmitter can be divided into two categories; channel coding and modulation. **Cource coding** refers to all processing that occurs before individual subcarriers are modulated. This includes actions such as data framing, forward error correction coding and data scrambling. **OFDM Modulation** refers to the modulation of individual subcarriers,
the inverse fast fourier transform operation and guard interval insertion. While channel coding can vary significantly between standards, OFDM modulation generally includes the same 3 components. This section first explains OFDM modulation and some other OFDM constructs such as preambles and pilot tones. We then describe channel coding and its purpose. Finally, some applications of OFDM are discussed.

3.1.1 Modulation

The first step in OFDM modulation is subcarrier modulation. One or more bits from the channel coding step are assigned to each subcarrier then modulated using a simple technique such as phase shift keying (PSK) or quadrature amplitude modulation (QAM). The number of bits used for each subcarrier depends on the modulation technique being used. For example, QPSK requires two bits per subcarrier. The number of subcarriers and the modulation to use for each subcarrier is defined by the protocol or standard being used.

At the heart of an OFDM modulator is the Discrete Fourier Transform (DFT) in its common implementation the Fast Fourier Transform (FFT) algorithm. The DFT is a function that takes time series data as its input and returns a set of coefficients representing frequencies found in that data. The OFDM modulator performs the inverse DFT operation using the inverse FFT (IFFT) algorithm. As its name implies, this operation is the opposite of the DFT, transforming frequency information into time series data. Both the FFT and the IFFT have a size, $N_{\text{fft}}$, associated with them. This
is the amount of input and output produced by one execution of the algorithm and is an important design parameter for OFDM systems. The sample rate of the system and $N_{fft}$ determine the spacing of the subcarriers. IFFT is ideal for OFDM because it is has efficient hardware and software implementations and because it produces subcarriers that are ideally spaced. Subcarriers are in fact orthogonal in that they are spaced so that they do not interfere with each other. The IFFT found in OFDM takes modulated subcarrier data as input and produces time series data as output. This time domain signal will contain the sum of each of the $N_{fft}$ subcarrier signals. One **OFDM Symbol** consists of the output of one IFFT execution representing $N_{fft}$ subcarriers.

Since the OFDM symbol contains many subcarriers representing many bits, the duration of the OFDM symbol can be relatively long. This reduces the effect of intersymbol interference (ISI) caused by multipath reflections at the OFDM receiver. Multipath reflections occur when a signal reflects off different surfaces along multiple paths from the transmitter to the receiver. Since any of these reflected paths will be longer than the direct path from transmitter to receiver, the multipath signals will be delayed relative to the direct path signal and will overlap with the next OFDM symbol. Since the OFDM symbol period is long, the amount of overlap can be small when compared with a serial modulation scheme where the amount of overlap can span several short symbols. To completely eliminate ISI, a guard interval can be inserted at the start of each OFDM symbol. During the guard interval, the receiver might still be receiving delayed copies of the previous symbol caused. The receiver then ignores the guard interval portion of the transmission. In many OFDM systems the guard interval is a cyclic extension of the OFDM symbol. A cyclic extension is a copy of the last portion of the OFDM symbol appended to the start of the same symbol.

Multipath reflections also cause attenuation or nulls to occur at different frequencies within an OFDM symbol. This may effect one or more subcarrier frequencies while not impacting the other subcarriers. To measure these fades and other channel condi-
tions, it is common to insert what are known as pilot subcarriers into the OFDM symbol. Since the symbols carried by the pilots are known to the receiver, the receiver can use the pilots to estimate the channel conditions across all the subcarriers and compensate appropriately. This is sometimes called pilot symbol assisted modulation (PSAM).

In frame based OFDM systems, it is common to start each frame with some number of preamble symbols. There are a number of reasons for this. One reason is that the receiver must be able detect the start of a transmission and adjust its gain control circuits. The receiver can also synchronize its timing and fine tune its clock frequency to match that of the receiver. Finally, the preamble allows some channel estimation and equalization to occur.

3.1.2 Channel Coding

To improve the performance of OFDM, systems will employ some sort of coding layer. Forward error correction (FEC) introduces redundancy into the data stream so that errors caused by fading or interference in only a few subcarriers can be corrected using error free data from the other subcarriers. The performance of some coding schemes can be improved in OFDM with an interleaving step. Interleaving mixes up the bits so that consecutive bits in the data stream are assigned to different subcarriers in the OFDM symbol. This means that errors in consecutive subcarriers won’t result in bursts of errors in the deinterleaved data stream. FEC is one of the most important channel coding operations in OFDM. Other things that can be done are scrambling to whiten the data stream, data compression to reduce the size of the data stream, and framing to format and add control information to the data stream.

3.2 OFDM Applications

The idea of OFDM has been around for about 50 years, but only in the last 10 years has its use become widespread. Many popular standards and protocols now use
some version of OFDM. Examples are digital audio and video broadcasting (DAB,DVB-T), IEEE 802.11a/g, IEEE 802.16 (WiMax), and Asymmetric Digital Subscriber Line (ADSL).

There are two broad categories of OFDM standards and protocols; frame based and stream based. The distinction between frame based and stream based protocols is similar to the distinction between packet switched and circuit switched networks. Stream based OFDM standards like digital radio and TV transmit a continuous flow of data. Packet based protocols like 802.11a transmit data as short bursts so that multiple stations can share the wireless medium. The burst nature of frame based protocols means that the receiver must perform synchronization at the start of each frame. Since the receiver sees a continuous signal in stream based protocols, it can synchronize more slowly and maintain that synchronization over time.

As software defined radio and other flexible OFDM platforms become possible, new uses of OFDM are possible. One possibility is to use OFDM as a multiple access scheme, where different transmitters and receivers use different subcarriers to share the frequency spectrum. This idea already used as part of the 802.16 standards where the base station divides the subcarriers among its receiver stations [21]. A similar idea is wireless wormhole networks [22] where paths through a large network are preallocated using a subset of the subcarriers at each hop. This enables fast switching as well as a traffic management framework.

Individual control of the subcarrier enables a number of optimization possibilities. Traditional protocols like 802.11a change the subcarrier modulation used depending on channel conditions. The problem with this approach is that all subcarriers use the same modulation scheme. Since multipath fading and interference may impact only a few of the subcarriers, a better approach may be to adapt individual subcarriers to channel conditions [23]. A similar technique to avoid incumbent users of spectrum is non-continuous OFDM (NC-OFDM) [24]. In NC-OFDM, the radio senses existing users
and turns off subcarriers that would interfere with those users.
A goal of this work is to provide a general purpose OFDM modulator for software defined radio. Such a design should support existing and future protocols. A general purpose OFDM modulator should:

1. Provide access to a suitably large chunk of spectrum.
2. Provide support for stream based and frame based protocols.
3. Provide fine grained subcarrier modulation control.
4. Provide guard interval (cyclic extension) control.
5. Provide variable IFFT width
6. Provide variable sampling rate and subcarrier spacing

The design presented here fulfills all but the last of these goals. Furthermore, we provide software control over these parameters. The modulator supports 40MHz of bandwidth, which is wider than the 35MHz provided by our radio. The CPU/FPGA interface and the OFDM modulator support both frame based and stream based protocols. The modulation of subcarriers is individually controlled and can be changed between consecutive OFDM symbols. The guard interval can also be changed on a symbol by symbol basis. Finally, by disabling subcarriers, software can effectively change the size of the IFFT and create non-continuous waveforms.
4.1 Platform

Our software defined radio platform is shown in Figure 4.1. The platform is composed of a Xilinx XtremeDSP development kit manufactured by Nallatech and a 2.4GHz radio transceiver manufactured by Fidelity Comtech, Inc. The XtremeDSP development kit features a Nallatech Benadda Pro PCI/USB card that includes a Vertex II Pro FPGA, 4 MByte of RAM and two sets of A/D and D/A converters. Although the picture shows the device in its USB configuration, we typically use it as a PCI card in a standard PC.

Nallatech has provided a number of example FPGA designs that can be used with the XtremeDSP kit. One of these designs, the SV_IFACE, is a simplified interface to the PCI bus. Using this design, we have developed a basic SDR platform using VHDL [25]. A diagram illustrating the features of this SDR platform is shown in Figure 4.2. Control and status registers are not shown.

On the transmit path, data is first stored in one of the two banks of ZBT memory found on the PCI card. This memory contains 512K 32 bit entries and can be used in two ways; it can serve as a simple FIFO between the interface FIFO and the rest of the
transmit path or one can buffer data in the memory then later trigger a transmission of the buffer contents. Data is read out of the ZBT memory by a configurable memory read step, a simple block that controls the number of cycles between memory reads. This is important for interpolation, where one input sample produces multiple outputs, or when multiple samples are packed into the 32 bit ZBT word. Data produced by the memory read block is sent to the transmit logic block. This is where our OFDM modulator resides. In the ”stock” configuration, this block contains an interpolation filter. Finally, samples are written to the D/A converters.

The receive path is not used in this work but it will be briefly described. Samples are read from the A/D converters into a DC offset removal block. This block sends its output to a downconversion step that multiplies one of the input channels by a DDS generated complex signal producing I and Q outputs. It can also be disabled in which case one or both of the input channels are sent on to the decimation block. The configurable decimation block simply throws away some portion of the input stream. Samples are then buffered in a large FIFO where they wait for a software triggered DMA transfer over the PCI bus.

Although it is not used here, we have also produced a GNU Radio interface to this design. As GNU Radio matures, we will port our software to it.
4.2 Design Decisions

In order to implement an OFDM waveform on our platform, we had to first partition the design between the host processor and the FPGA. Our base FPGA design simply transfers samples from the PCI bus to the D/A converters on the card. There is enough space remaining on the FPGA to implement significant DSP pipelines. Thus we could implement all processing on the host, all processing on the FPGA, or choose a hybrid design somewhere in between.

One of our goals is to provide a platform that can utilize a large chunk of spectrum and that can control as much of the transmission process as possible. Clearly an all software design can achieve these goals given enough computational power and I/O bandwidth to the A/D converters. Furthermore, given the ease of programming software when compared to designing hardware, it is desirable to keep as much in software as possible.

The FFT/IFFT algorithm is one of the most computationally and I/O intensive portions of OFDM. To understand the requirements of OFDM modulation, we ran some simple benchmarks using FFTW [26], a heavily optimized FFT library. Since the IFFT is only a part of the receiver, this will give us an upper bound on performance. We benchmark the IFFT operation assuming a fixed subcarrier spacing of .3125MHz and a fixed OFDM symbol period of $3.2 \mu$sec. These parameters correspond to the requirements of IEEE 802.11a. We also assume that the IFFT produces complex 16-bit inputs and outputs, a data size representative of typical digital to analog converter inputs. Table 4.1 shows the performance of FFTW and the real time performance required by the example OFDM application as the IFFT size is increased. To maintain a constant OFDM symbol period, the sample rate must increase with the IFFT size. Thus, we see that as IFFT size doubles, so does the amount of data it produces. We also observe that the performance per sample (subcarrier) decreases as the IFFT size increases. For the
<table>
<thead>
<tr>
<th>IFFT size</th>
<th>FFTW MBytes/sec</th>
<th>Peak PCI MBytes/sec</th>
<th>802.11a Requirement MBytes/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>336</td>
<td>133</td>
<td>80</td>
</tr>
<tr>
<td>128</td>
<td>268</td>
<td>133</td>
<td>160</td>
</tr>
<tr>
<td>256</td>
<td>232</td>
<td>133</td>
<td>320</td>
</tr>
<tr>
<td>512</td>
<td>212</td>
<td>133</td>
<td>640</td>
</tr>
<tr>
<td>1024</td>
<td>188</td>
<td>133</td>
<td>1280</td>
</tr>
</tbody>
</table>

Table 4.1: The performance of software IFFT compared to an example OFDM application (2.2GHz AMD Athlon 64 Processor)

example application, the CPU cannot keep up with processing requirements for IFFT sizes greater than 128. Furthermore, the PCI bus with a theoretical peak throughput of 133MByte per second, can only keep up with the smallest of the IFFT sizes tested.

As CPU performance continues to increase, multiple cores and acceleration devices like GPUs become integrated into PC processors and better I/O standards like PCI express are introduced into software radio platforms, some of the performance constraints presented will be met. For the foreseeable future, however, we will continue to desire more performance than our PC based platforms can provide. A 4096 FFT, for example, will require more I/O throughput than 16 lane PCI Express operating at 4GByte/sec can provide. Furthermore, we would like to make use of the platforms we have today instead of waiting around for the future. These factors all point to a hybrid CPU/FPGA design. By partitioning the design correctly between software and hardware, computational and I/O requirements can be reduced.

We believe that the correct partitioning of an OFDM transmitter is to place bit manipulation operations such as framing, coding, interleaving and subcarrier assignment in host software while subcarrier modulation, the IFFT operation and guard interval insertion are performed on the FPGA. The modulator presented here takes this approach. To retain software control over these functions, the OFDM modulation on the FPGA must be as general and configurable as possible. This partitioning puts enough functionality on the FPGA to significantly reduce computation and I/O subsystem re-
quirements but only places functions on the FPGA that tend to be universal across OFDM protocols. For example, different protocols may have very different data framing, error coding, and preamble schemes, but they will all eventually divide frames into subcarriers and run an IFFT on them. To see why the I/O requirements are reduced, note that the input to the subcarrier modulators is as small as one bit for BPSK or as large as six bits for 64-QAM. However, the output of the IFFT is a pair of 16 bit numbers. So if we send unmodulated subcarriers to the FPGA as bytes then the I/O requirements of the system are reduced by a factor of four.

4.3 A General Purpose OFDM Modulator

4.3.1 Overview

A summary of the fixed and configurable parameters of our OFDM modulator is shown in Table 4.2. We choose a 256 point IFFT and a sample rate of 80MHz for our design. Although the sample rate can be changed by altering the FPGA design, it is viewed as fixed by software. We picked these values so that the subcarrier spacing of the design is compatible with 802.11a receivers. The 256 subcarriers are individually modulated before they are sent to the inputs of the IFFT. A subcarrier can be disabled or modulated using BPSK, QPSK, 16-QAM, or 64-QAM. By enabling or disabling appropriate subcarrier modulators, software can effectively reconfigure the IFFT size to be anywhere between one and 256. The length of the cyclic extension for the guard interval insertion step can be configured to be between 0 and 100 percent of the OFDM symbol duration.

Subcarrier bits and configuration data are sent from software to hardware using a packet-like API. Packets are subdivided to contain zero or more control blocks and zero or more data blocks. A block starts with a block identifier followed by block specific data. Figure 4.3 shows an example input packet. The first part of this packet
contains some control blocks to configure the modulator. Following the control blocks are two data blocks that will be modulated to form two OFDM symbols. It is not always necessary to include control blocks with data blocks. If the OFDM modulator is already configured appropriately, only the data blocks need to be sent. Similarly, control blocks do not need to be accompanied by data blocks. Each entry in a packet is a single byte. Although this limits the range of values allowed, it gives a very compact representation. For IFFT sizes larger than 256, it may be necessary to allow data types larger than one byte. There are four block types:

- The NCARRIERS block tells the hardware how many of the IFFT inputs software will use. The block identifier is followed by the number of IFFT inputs. Specifically, this variable tells the hardware how many subcarriers will be consumed by the subcarrier modulation unit during each OFDM symbol period.

- The MOD block is used to configure the individual subcarrier modulators. The block identifier is followed by the number of data items contained in the block. Each data item, consisting of an address byte and a configuration byte, configures two subcarrier modulators. The address byte gives the address of the first modulator to be configured and the configuration byte contains 4 bits of configuration information for each subcarrier. The control bits enable or disable the subcarrier and also select the subcarrier modulation scheme to be used with that subcarriers. If a subcarrier is disabled, a zero is inserted into the

<table>
<thead>
<tr>
<th>OFDM Parameter</th>
<th>Allowable Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>sample rate</td>
<td>80 MHz</td>
</tr>
<tr>
<td>IFFT size</td>
<td>1-256</td>
</tr>
<tr>
<td>subcarrier spacing</td>
<td>.3125 MHz</td>
</tr>
<tr>
<td>OFMD symbol period</td>
<td>3.2 µs</td>
</tr>
<tr>
<td>subcarrier modulation</td>
<td>BPSK, QPSK, 16-QAM, 64-QAM</td>
</tr>
<tr>
<td>guard interval</td>
<td>0-3.2 µs</td>
</tr>
</tbody>
</table>

Table 4.2: OFDM modulator parameters
corresponding IFFT input and no subcarrier data will be consumed for that subcarrier.

- The GUARD block is used to configure the size of the guard interval. The block identifier is followed by the size of the guard interval.

- The DATA block contains subcarrier bits to be modulated into OFDM symbols. The block identifier is followed by the number of subcarriers in the block. The format of the subcarrier data byte is defined by the modulation to be used and is generally just the appropriate number of bits for the modulation type.

This packet based API fits well with both stream and frame based software architectures. In frame structured programs, a functional block can attach meta-data to a data packet describing the coding or modulation to be performed on different parts of the data. When a functional block later consumes the meta-data it will remove it from the data packet. In software, a programmer has great flexibility in how meta-data is attached to data and passed between functional blocks. In our system we would like to minimize complex signaling paths between hardware processing blocks and other processing blocks, whether they are in software or hardware. Thus, we have designed a modulator where the only input is a structured stream of data. Before being sent to hardware, meta-data is serialized and placed inline with the appropriate parts of the data. In data programs, the processing paths and functionality of the various processing blocks are configured at the start of program execution and generally remain static for the remainder of the execution. Thus in our model we only need to send some configuration blocks to the hardware at the start of execution. Data then only needs to be divided up into DATA blocks before being sent to hardware.
Figure 4.3: Example OFDM modulator input packet

Figure 4.4: Screenshot of the OFDM modulator design
### 4.3.2 Implementation

The OFDM modulator was implemented using Simulink [27] and Xilinx System Generator [28]. A screen shot of the Simulink design is shown in Figure 4.4. The system generator tool takes this design and transforms it into a NGC file that is then instantiated in the VHDL design described in Section 4.1. This design flow makes it possible to quickly develop and simulate a FPGA design within the Simulink/Matlab framework, then incorporate it into a larger design. There are four major parts to the design:

**Controller**  The controller controls the flow of control signals and data to the rest of the modulator. Because of the packet format described in the previous section, it is easy to parse the input data stream using a simple state machine. In fact, this state machine and all the control signals for the design are described in less than 100 lines of m-code. M-code is Matlab’s native programming language and a subset of m-code can be automatically translated to a hardware description by System Generator.

**Subcarrier Modulation**  The modulation of each subcarrier is controlled by four bits of control data stored in the 256x4 subcarrier control memory. This memory is written by the MOD block described in the previous section. When a DATA block is encountered by the controller, the data within that block is written to a small FIFO sitting between the controller and the subcarrier modulation logic. At the same time, the controller starts issuing reads of each entry of the subcarrier control memory in order. The top bit of each control entry enables or disables the subcarrier. If the subcarrier is enabled, data is read from the FIFO and modulated using the modulation scheme indicated by the rest of the control entry. Otherwise, the subcarrier is disabled and the FIFO is not read. Instead, a zeros are inserted for that subcarrier. Modulated subcarriers are sent on to the IFFT.
**IFFT** We use a prepackaged FFT/IFFT block from Xilinx. When processing a DATA block, the controller only has to signal the start of an IFFT operation at the appropriate time. The IFFT reads and writes serial data streams.

**Guard Interval** When the controller processes a GUARD block, it writes the guard value to a register. Later, when the controller signals the start of an IFFT operation, it also writes the value of the guard register to a small FIFO. As the IFFT operation completes, this FIFO is read by the guard insertion logic. The output of the IFFT is written to two buffers. The first stores the entire output of the IFFT (i.e. a OFDM symbol) and the second stores just the number of samples needed for the guard. To complete the cyclic extension operation, the appropriate number of guard samples are read from the guard buffer followed by 256 samples from the symbol buffer.
Chapter 5

Example Application: IEEE 802.11a

To demonstrate the usability of the OFDM modulator design, a software 802.11a transmitter was implemented in C. After reviewing the 802.11a physical layer, our 802.11a software and its performance is described.

5.1 The 802.11a Physical Layer

The IEEE 802.11a Physical Layer (PHY) was defined in [29] for the 5GHz band and later extended to the 2.4GHz band in [30]. This specification defines OFDM modulation parameters such as subcarrier spacing, subcarrier modulation, FFT width, and guard interval duration. It also defines framing, scrambling, forward error correction coding and interleaving operations that are performed before OFDM modulation. These parameters and operations are combined to provide the eight data rate choices shown in table 5.1. Table 5.2 lists the timing related PHY parameters.

Figure 3.1 shows a block diagram of a 802.11a modulator. Data from the MAC layer is first padded with zero bits to aid coding and to ensure that the number of bits to be modulated is a multiple of the number of data bits per OFDM symbol. These bits are then scrambled by the function

\[ S(x) = x^7 + x^4 + 1 \]  

(5.1)

to randomize the data. At the receiver, the same equation is used to descramble the data. The scrambled data is then encoded using a 1/2 rate convolutional code. For
<table>
<thead>
<tr>
<th>Data Rate (Mbits/s)</th>
<th>Modulation</th>
<th>Coding Rate</th>
<th>Coded Bits per Subcarrier</th>
<th>Coded Bits per OFDM Symbol</th>
<th>Data Bits per OFDM Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>BPSK</td>
<td>1/2</td>
<td>1</td>
<td>48</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>BPSK</td>
<td>3/4</td>
<td>1</td>
<td>48</td>
<td>36</td>
</tr>
<tr>
<td>12</td>
<td>QPSK</td>
<td>1/2</td>
<td>2</td>
<td>96</td>
<td>48</td>
</tr>
<tr>
<td>18</td>
<td>QPSK</td>
<td>3/4</td>
<td>2</td>
<td>96</td>
<td>72</td>
</tr>
<tr>
<td>24</td>
<td>16-QAM</td>
<td>1/2</td>
<td>4</td>
<td>192</td>
<td>96</td>
</tr>
<tr>
<td>36</td>
<td>16-QAM</td>
<td>3/4</td>
<td>4</td>
<td>192</td>
<td>144</td>
</tr>
<tr>
<td>48</td>
<td>64-QAM</td>
<td>2/3</td>
<td>6</td>
<td>288</td>
<td>192</td>
</tr>
<tr>
<td>54</td>
<td>64-QAM</td>
<td>3/4</td>
<td>6</td>
<td>288</td>
<td>216</td>
</tr>
</tbody>
</table>

Table 5.1: Data rates and related parameters for IEEE 802.11a

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of FFT Points</td>
<td>64</td>
</tr>
<tr>
<td>Number of Data Subcarriers</td>
<td>48</td>
</tr>
<tr>
<td>Number of Pilot Subcarriers</td>
<td>4</td>
</tr>
<tr>
<td>Sample Rate</td>
<td>20MHz</td>
</tr>
<tr>
<td>FFT Symbol Period</td>
<td>3.2 µs</td>
</tr>
<tr>
<td>Guard Interval Duration</td>
<td>0.8 µs</td>
</tr>
<tr>
<td>OFDM Symbol Duration</td>
<td>4.0 µs</td>
</tr>
</tbody>
</table>

Table 5.2: IEEE 802.11a PHY timing parameters
each input bit, two output bits are generated by the functions

\[ g_0(x) = x^5 + x^4 + x^2 + x^1 + 1 \]  

and

\[ g_1(x) = x^5 + x^2 + x^1 + x + 1 \]  

To achieve higher rates, a puncturing step removes some of the coded bits from the data stream. After puncturing, an interleaving step ensures that adjacent bits are mapped onto nonadjacent subcarriers. The bits are now ready to be modulated.

The 802.11a standard defines an OFDM modulation like that described in chapter 3. The scrambled, coded, interleaved bits are divided up into OFDM symbols of 48 subcarriers where 1, 2, 4 or 6 bits are mapped to each subcarrier depending on the subcarrier modulation used; BPSK, QPSK, 16-QAM and 64-QAM are available. These subcarriers are individually modulated resulting in one complex number per subcarrier. Four pilot tones and 12 zero subcarriers are added for a total of 64 subcarriers per OFDM symbol. These vectors of 64 subcarriers are the input to the IFFT operation and 64 complex time domain samples are the output. The guard interval is added by copying the last 16 samples of each IFFT output to the start of the output giving 80 complex time domain samples per OFDM symbol.

Before transmitting the frame, two pieces of information are added. To help with signal detection, synchronization and channel estimation, a 16μs preamble begins each frame. The preamble consists of 10 short training symbols and two long training symbols. Following the preamble is the signal symbol encoding the length of the data portion of the frame and the data rate at which the data is encoded. The signal symbol is always modulated using the 6Mbit/sec rate.

An 802.11a receiver basically carries out the reverse of the above operations to decode a frame. The most significant additional steps required by a receiver are synchronization and equalization.
5.2 Software 802.11a Modulation

It was shown in [31] that there are smart ways to implement a software 802.11a transmitter such that the performance of the system is much better than a straightforward implementation of the processing blocks. For example, the convolutional coding, puncturing and interleaving steps can be combined into one step to avoid unnecessary data copies between processing elements. Our implementation uses similar techniques. We also precompute as much information as possible so that operations such as scrambling and convolutional coding become fast table lookups.

To verify our design partitioning, we also implement the entire 802.11a transmitter in software. For the IFFT operation, we again use FFTW. The output of the software transmitter is a stream of complex I and Q samples representing the preamble symbols, signal symbol and data symbols of the frame sampled at 20MHz. This does not account for any interpolation or filtering that might need to be performed.

Table 5.3 shows the performance of our software implementation on a desktop computer. The required rate column shows the 802.11a rate at which the data is being encoded and the software rate columns show the rate achieved by the software when processing 100 and 1500 byte packets. For all but the highest rates, the modulator reaches the required performance. In table 5.4, the performance of the software without
Table 5.4: Required rate vs. achieved rate for a software implementation of a 802.11a modulator without subcarrier modulation, IFFT, or guard interval insertion (100 and 1500 byte packets, 2.2Ghz AMD Athlon 64 Processor)

<table>
<thead>
<tr>
<th>required rate</th>
<th>software rate</th>
<th></th>
<th>software rate</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100 byte packets</td>
<td>1500 byte packets</td>
<td>1500 byte packets</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>38.9</td>
<td>43.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>43.7</td>
<td>49.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>43.8</td>
<td>49.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>48.1</td>
<td>55.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>47.2</td>
<td>54.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>50.9</td>
<td>59.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>47.9</td>
<td>60.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>51.5</td>
<td>60.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
the OFDM modulation step (i.e. without subcarrier modulation, IFFT, guard insertion) is shown. In this case, the CPU can keep up with the computational requirements of the all the rates. This suggests that the partitioning we choose is the correct one for 802.11a.

The operation of the hybrid 802.11a transmitter can be verified using an off the shelf 802.11g wireless adapter to receive transmitted packets. The complete setup is shown in figure 5.1. Packets are received from an arbitrary network application using a Linux tap interface. We use the zulu [32] program since it produces valid 802.11 MAC layer packets. The packet is then encoded using our 802.11a software. We package the resulting frame using the packet format described in section 4.3.1 and send it over the PCI bus to the FPGA. The FPGA processes the data and an analog signal is sent to the radio and over the air. The 802.11g adapter receives the frame and we can verify its correctness using standard network sniffing software. Using this process we verified our 802.11a design.

Modulating an 802.11a frame exercises all the features of our design. It requires subcarriers to be reconfigured in the middle of a frame (between the preamble and payload), it requires that subcarriers are modulated differently within a single OFDM symbol (pilots subcarriers vs. data subcarriers), it requires the guard interval to be reconfigured in the middle of a frame (again, between preamble and payload), and it requires only a portion of the 256 subcarriers available. Since we are able to code,
modulate, transmit and subsequently receive such a frame over the air, we conclude that the OFDM modulator design works.
Chapter 6

Other Applications

The flexibility of our platform is reinforced with two more examples. The first shows non-continuous transmission and the second shows the frequency agility of the platform.

Since the modulator can utilize 40MHz of spectrum and an 802.11a signal requires less than 20MHz, one could imagine transmitting on two 802.11a channels at once. Figure 6.1 shows the spectrum of such a transmission during a Simulink simulation. In this example, one 802.11a transmission is centered at zero and another is centered at roughly ± 20MHz. This is an example of non-contiguous transmission similar to what NC-OFDM requires. Because our radio cannot handle a signal of this width, this application could not be tested over the air.

To demonstrate the frequency agility of the platform, a small application was written to send OFDM frames using a random number of subcarriers at a random offset in the frequency space of the modulator. Figure 6.2 shows a waterfall plot of the application’s transmissions as captured over the air by a vector signal analyzer. The plot shows bursts of a few thousand packets each over a time period of about 60 seconds. The width of the plot is 20MHz.
Figure 6.1: Non-contiguous transmission of two 802.11a frames

Figure 6.2: Waterfall plot of frequency agility. The X axis is frequency and the Y axis is time. Signal strength is indicated by color.
Chapter 7

Conclusion

This thesis has presented a unique OFDM transmitter design for hybrid CPU/FPGA software defined radio platforms. In doing so, we have also presented a design style for the implementation of programmable SDR accelerators. In this model, data and control information (meta-data) are communicated together to hardware using a well defined and structured data stream. This type of API enables easy parsing of the data stream by hardware while moving control of the hardware into software. In some ways, this is not unlike the stream of opcode and operand data presented to a CPU. It is also similar to the structure of data passing through packet data networks.

The OFDM modulator itself provides enough processing power and generality to implement a range of OFDM applications. We have demonstrated this by implementing an IEEE 802.11a transmitter on the platform. We have also shown the applicability of our design to non-continuous and frequency agile OFDM applications.

In the future, we will continue this work by applying the same design principles to an OFDM receiver. We will also be porting our work, particularly our 802.11a software, to the popular GNU Radio software platform. This will allow others to enjoy the benefits of this work.
Bibliography


