# 802.15.3 Transmitter: A fast design cycle using OFDM framework in Bluespec

Teemu Pitkänen[1], Vesa-Matti Hartikainen[1], Nirav Dave[2], and Gopal Raghavan[3]

[1] Tampere University of Technology, P.O. Box 553, FIN-33101 Tampere, Finland
[teemu.pitkanen, vesa-matti.hartikainen]@tut.fi
[2] Massachusetts Institute of Technology, Cambridge, USA
ndave@csail.mit.edu
[3] Nokia Research Center Cambridge
Nokia Corporation
gopal.raghavan@nokia.com

**Abstract.** Orthogonal Frequency-Division Multiplexing (OFDM) has become the preferred modulation scheme for both broadband and high bitrate digital wireless protocols because of its spectral efficiency and robustness against multipath interference. Although the components and overall structure of different OFDM protocols are functionally similar, the characteristics of the environment for which a wireless protocol is designed often result in different instantiations of various components. In this paper we present a new baseband processing transmitter case, namely 802.15.3 (WUSB), to existing OFDM framework which consists highly parametrized code in Bluespec for two different wireless baseband processing cases, namely 802.11a (WiFi) and 802.16 (WiMAX). The design cycle for transmitter of WUSB took only six week's for two designers which were not familiar with Bluespec, WUSB protocol or the OFDM framework.

## 1 Introduction

Wireless systems are experiencing rapid development as more applications call for mobile and distributed use. To effectively meet the vastly varying application requirements (*e.g.*,power, bitrate, and flexibility) a variety of different wireless protocols have been designed. In recent years, Orthogonal Frequency-Division Multiplexing (OFDM) has become preferred modulation scheme for both broadband and high bitrate digital wireless protocols because of its spectral efficiency and robustness against multipath interference. These protocols are sufficiently similar that many of the component blocks in transceivers across protocols could be described using the same parametric module with different parameters in the various forms including bitsizes, default values, pipelining strategies and combinational functions.

Despite the capability for sharing and the significant time pressure on designers to ship designs quickly, in practice engineers still write each design from scratch ignoring possible reuse between designs. Much of this is due to the fact that while most hardware description languages (HDLs) like Verilog and VHDL provide the ability for parameterization, only very low-level parameterization is supported (*e.g.*,values and bit-sizes) leaving many important parameterizations very hard to describe.

Recently, Ng et. al. developed a parameterized suite for quickly generating OFDM baseband transceivers [1] in Bluespec SystemVerilog (BSV), a high-level hardware description language which can be compiled mechanically in to efficient high-quality RTL code [2]. This suite consists of a number of highly parameterized OFDM component blocks which can be reused across multiple designs. These parametric designs cause no additional hardware overhead, as the Bluespec compiler can remove all static parameterization during design elaboration.

The OFDM framework provides specific baseband implementations for both the 802.11a (WiFi) and 802.16 (WiMAX) protocols. Using this as a starting point we add the design of a 802.15.3 (WUSB) transmitter. This work took very little time, taking only six weeks for two designer unfamiliar with both BSV and OFDM protocols to complete.

## 2 OFDM Framework

The OFDM Framework used has been developed as part of the ARMO project by Nokia Research Center and Massachusetts Institute of Technology [3]. The project started out focusing on studying the cost-area tradeoffs possible in the RTL design of a 802.11a transmitter [5]. As the project progressed it became clear that many of the key blocks in both the transmitter and reciever, the receiver size, it then became clear that many of the key blocks, while complex, were reusable across multiple OFDM-based protocols. This framework has been released to the public under the MIT license [4].

The structure of OFDM implementations described in this framework is shown in Figure 1. To aid comprehension, we briefly discuss the high-level functionality of each blocks:

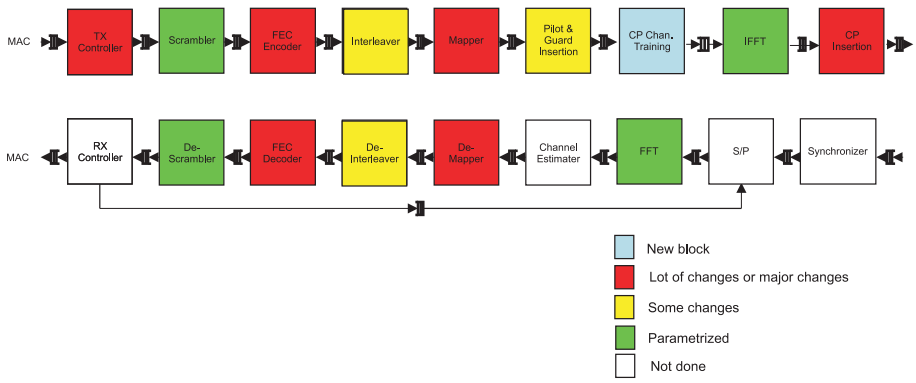


**Fig. 1.** Structure of the OFDM Framework and changes made

### 2.1 Transmitter

**TX Controller:** Receives information from the MAC. Adds header data before actual payload and generates control for all the subsequent blocks.

**Scrambler:** Randomizes the data stream to remove repeated patterns.

**FEC Encoder:** Encodes and adds some redundancy to data making it possible for the receiver to detect and correct errors. The encoded data is punctured to reduce the transmitted number of bits.

**Interleaver:** Interleaves bit stream to provide robustness against burst errors.

**Mapper:** Passes interleaved data through a serial to parallel converter, mapping groups of bits to separate carriers, and encoding each bit group by frequency, amplitude, and phase. The output of the Mapper contains the values of data subcarriers for an OFDM symbol.

**Pilot/Guard Insertion:** Adds the values for pilot and guard subcarriers to OFDM symbols.

**IFFT:** Converts OFDM symbols from the frequency domain to the time domain.

**CP Insertion:** Copies some samples from the end of the symbol to the front to add some redundancy to the symbols to avoid Inter-Symbol Interference. The block also adds a preamble before the first transmitted symbol.

After CP insertion, OFDM symbols are outputted to a DAC, which converts them to analog signals which can them be transmitted.

### 2.2 Receiver

The receiver roughly applies the transmitter transformations in reverse. However, it requires some additional feedback to help synchronize to to the expected phase.

**Synchronizer:** Detects the starting position of an incoming packet based on preambles.

**Serial to Parallel (S/P):** Removes the cyclic prefix (CP) and then aggregates samples into symbols before passing them to the FFT. It also propagates the control information from the RX Controller to subsequent blocks.

**FFT:** Converts OFDM symbols from the time domain into the frequency domain.

**Channel Estimator:** Compensates for frequency-dependent signal degradation based on pilots and corrects the errors caused by multi-path interference.

**Demapper:** Demodulates data and converts samples to encoded bits.

**Deinterleaver:** Reverses the interleaving and restores the original arrangement of bits.

**FEC Decoder:** Uses the redundant information to detect and correct errors occurred during transmission.

**Descrambler:** Reverses the scrambling.

**RX Controller:** Based on the decoded data, the RX Controller generates the control feedback to S/P block.

## 3 The WUSB Transmitter

The OFDM Framework provided a very good starting point for WUSB implementation. It included almost all of the functionality needed, and most changes were just changes to parameters of the framework. Only few bigger changes to the framework were needed. Figure 1 illustrates the structure of OFDM Framework and changes necessary for WUSB. In this chapter we discuss some of the specific changes and how they were represented.

### 3.1 Parameterization

Many of the modifications needed in the WUSB design are captured by the component module parameterization. Table 1 lists some of the parameter settings of each protocol.

**Convolutional Encoder:** One of the simplest examples of parameterization we encountered was in the convolutional encoder. In this design, we needed to generate a 3-bit output for each 1-bit input using a moving history of 8 input bits. To represent this change the input and output sizes to match the expected rates (8 and 24 respectively) and pass in three values representing the individual polynomial for each output bit. The computation necessary for each output bit can be described by a single polynomial in $\mathbb{Z}_2$. These are represented as $8-bit$ values. Thus we need to pass in 3 8-bit values to the parameterized module.

Due to a restriction in the current Bluespec compiler, to generate a separate Verilog module for this block, the Bluespec module be non-parameterized. This requires us to add a small wrapper module to restrict the type and provides the modules arguments to make the module self-contained.

```
typedef 8 ConvEncoderInDataSz;
typedef TMul#(3,ConvEncoderInDataSz) ConvEncoderOutDataSz;
module mkConvEncoderInstance(ConvEncoder#(TXGlobalCtrl,
                                          ConvEncoderInDataSz,
                                          ConvEncoderOutDataSz));
  ConvEncoder#(TXGlobalCtrl, ConvEncoderInDataSz
                           , ConvEncoderOutDataSz) convEncoder
        <- mkConvEncoder(convEncoderG1, convEncoderG2,convEncoderG3);
  return convEncoder;
endmodule
```

**Puncturer:** A slightly more interesting parameterization can be found in the puncturer. Puncturing is a feature of the FEC encoder which allows the transmitter to reduce the number of bits being sent. For higher transmission rate, in low-noise channels, the encoded data is punctured by deleting bits before transmission and replacing them with fixed values on reception. This reduces the number of bits to be carried over the channel as we can depend on the error correction in the receiver to correctly reconstruct the data.

The WUSB protocol specifies 7 separate puncturing modes, of which 5 are already described the previous framework. To add a new puncturing mode, we define the new

functions `punctureHalf` which takes 3 bits and returns 2 bits and `puncturerElevenThirtySecond` which takes 33 bits and returns 32 bits.

```
function Bit#(2) punctureHalf (Bit#(3) x);
   return {x[2], x[0]};
endfunction
function Bit#(32) puncturerElevenThirtySecond (Bit#(33) x);
  return x[31:0];
endfunction
```

Each function is then extended to the apply to the input size using the `parFunc` function. These new functions, along with the other functions corresponding to the other modes along with a function which determines which function corresponds to which mode (`puncturerMapCtrl`).

```
puncturer <- mkPuncturer(puncturerMapCtrl,
                         parFunc(f0_sz,punctureHalf),
                         parFunc(f1_sz,puncturerTwoThird),
                         ...,
                         parFunc(f6_sz,puncturerFiveEigth) );
```

Using functions as parameters is possible because functions are considered first-order objects in Bluespec.

| Blocks | Parameters | WiFi | WiMAX | WUSB |
|---|---|---|---|---|
| Scrambler | Generator Polynomial | $X^7+X^4+1$ | $X^{15}+X^{14}+1$ | $X^{15}+X^{14}+1$ |
| Convolutional | Generator Polynomials | 133oct & 171oct | 133oct & 171oct | 133oct, 165oct & 171oct |
| Interleaver | Coding Rate | 1/2, 2/3, 3/4 | 1/2, 2/3, 3/4, 5/6 | 1/2,11/32, 5/8, 3/4 |
| | No. Stages | 2 | 2 | 3 |
| Mapper | Modulation Schemes | BPSK, QPSK, 16-QAM, 64-QAM | BPSK, QPSK, 16-QAM, 64-QAM | QPSK |
| Pilot & Guard Insertion | No. Pilot Subcarriers | 4 | 8 | 12 |
| | No. Guard Subcarriers | 12 | 56 | 10 |
| FFT/IFFT | Size | 64 | 256 | 128 |
| Cyclic Prefix Insertion | Size | 1/4 | 1/32, 1/16, 1/8, 1/4 | N/A |

**Table 1.** Algorithmic settings for WiFi, WiMAX and WUSB

### 3.2 Further Changes

Since the frame header is protocol specific, the transmission controller also needs to be changed. Figure 2 illustrates the WUSB frame format. While, this format is similar

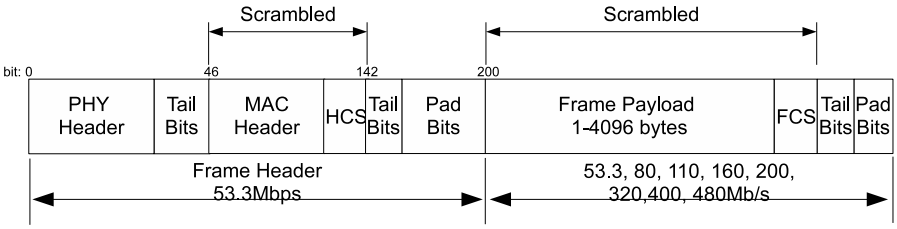


**Fig. 2.** PLCP Frame Format for WUSB

to both the WiFi and WiMAX protocols, these differences are not well-suited to parameterization, since the description complexity required to express the controller in a parametric way is worth the cost of writing a new module.

The biggest change in the controller was the addition of a side scrambler. In WiFi all header data is scrambled. In WiMAX all header data is sent unscrambled. In WUSB we had to change scrambling of the headers, since a combination of the MAC header and header checksum (HCS) needed to be scrambled and the PHY header should not be scrambled. Since the scrambled parts of the header do not fit the byte alignment, we either needed to change the scrambler to support non-byte aligned scrambling, add a new module to the main pipeline for adding tail bits after the PHY header or add a side scrambler that is used only for encoding scrambled part of the header. We choose to implement the later two options. The two options we considered are illustrated in Figure 3 We decided to use a separate side scrambler. This allows use the library scrambler implementation more easily. The side scrambler is only used for headers; the payload is still scrambled by the main scrambler instance in the pipeline.

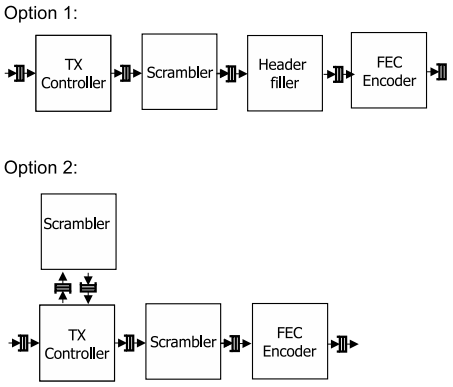


**Fig. 3.** Implementation Options for Header Scrambling. Option 1 adds a “header filler” module to add tail bits. Option 2 uses a separate side-scrambler for side scrambling.

**Interleaver:** In WiFi and WiMAX the data interleaving is done only inside symbol. In WUSB, the interleaver must support interleaving of data across symbol triplets. This change did not require changing the parameteric interleaver, only a new interleaving function using three symbols as an input, not one.

**Preamble Generation:** In WUSB, there are 4 different preambles added to symbols in both the time and frequency. The choice for these is not specified. As a result we only implemented the first choice. Augmenting the system to add the other preambles can be easily done.

Previous OFDM transmitters did not need to support frequency-domain preambles. To add this functionality, we need to add a new block into our pipeline: the CP Channel Training Block. This block adds 6 prefixed sequences to the input of the IFFT separately. The system sends this frequency preamble before sending the symbol header and payload.

**Mapping Values:**

In WiFi and WiMAX the mapping always remains the same in WUSB there are two different mappings. At data rates below 110 Mbps we encode 100 input bits to 50 complex numbers and calculate complex conjugate of the numbers and append it to end to form a single OFDM Symbol. At faster rates this redundancy is removed and an OFDM symbol is formed directly of 200 input bits. Other change is contents of the guard bits at the edges of frequency. WiFi and WiMAX uses zeros as contents of guard bits. Instead in WUSB we replicate outermost data bits and use them as content of the guard bits.

The Mapper for WUSB only needs to support QPSK-modulation, and therefore input and output sizes are less constrained than other schemes where other modulation schemes more restrict the choices, meaning the provided parameterized mapper did not cover all cases for the protocol. To generate a WUSB mapper we had to manually strip modulation and generate a specialized mapper. A more parameterized mapper must be designed to capture this implementation as well.

## 4 Implementation Results

In the following section we describe the result of synthesis of the WUSB transmitter and evaluate the value of the OFDM framework.

### 4.1 Technical Results

The WUSB transmitter is synthesized with Synopsys Design Compiler to 130 nm technology with 1.5 V operating voltage and power dissipation is acquired through gate-level simulation at 100 MHz.

The results of the synthesis are presented in the Table 2. Most of the area and power are consumed by the 128-point IFFT block. We compare our designs to two comparable FFT implementations compatible with our design.

To meet the frequency requirements of the protocol, we must be able to complete an IFFT in 312.5 ns. We use a folded pipeline design using the same Bluespec pipelining

| Component | # of Gates (K) | Power (mW) | Component | # of Gates (K) | Power (mW) |
|---|---|---|---|---|---|
| TX Controller | 2.8 | 1.18 | Scrambler | 0.5 | 0.085 |
| FEC Encoder | 7.5 | 2.31 | Interleaver | 15.9 | 4.81 |
| Pilot & Guard Insertion | 47.5 | 14.4 | CP Channel Training | 46.2 | 10.2 |
| IFFT | 718.2 | 36.2 | CP Insertion | 23.6 | 0.45 |
| Mapper | 57.8 | 11.3 | Total | 920 | 92.3 |

**Table 2.** Area and Power Dissipation of WUSB Transmitter

framework [5]. Thus it is easy to quickly change the area/performance tradeoff. Our final IFFT design uses 32 radix-2 butterflies and requires 14 cycles of computation per input.

To match the performance requirements for our IFFT block needs to run at 44.8 MHz. In fact, the critical path 9.5 ns which allows the block to be clocked at 105 MHz.

Other FFT implementations in the literature have similar results. Mathew et. al [6], use a pseudo parallel datapath structure to calculate a 128-point FFT in 10 cycles that can be clocked at 275 MHz. The architecture used 3 butterfly stages, and with each butterfly stage containing 8 separate datapaths. Power figures for this design are measured using a clock speed of 33.3 MHz.

Chen et. al [7] present a different 128-point FFT core, with four radix-$2^2$ and four radix-$2^2/2$ butterflies. The first two and last two stages each use a separate sets of butterflies requiring 32 cycles to compute one input. The design also used six eight-bank single ported RAMs, two coefficient ROMs, and two address generators. While the authors were only able to run the system at 66 MHz, scaling down the technology would give comparable results to ours.

The comparison FFT presented here is shown in Table 3. Area and power consumption numbers is normalized to 130 nm technology to give a fair comparison between designs. the maximum clock describes how fast each design can run, required clock describes how fast the design must run to achieve the performance requirement. The parameterized BSV code requires approximately 50% more area and slightly more power compared to [6], and 3.5 times more area compared [7]. From experience much of this area overhead is due to our choices of using radix-2 butterflies as the base block in our design. Larger radices would improve the design, though they would require the FFT either be partitioned into two parts, or the inputs changed so that the $2^7$ size input is naturally factor by the radix size.

## 5 Development Experience

The WUSB design was done by two engineers as a six-week project. Only one had any previous hardware design experience (i.e. VHDL). The pair spent approximately two weeks learning the language, the OFDM framework, and the WUSB specification

| Name | Butter-flies | Radix of Butterflies | # of Gates (K) | Power (mW) | Tech | max.clk (MHz) | req. clk (MHz) |
|---|---|---|---|---|---|---|---|
| BSV | 32 | 2 | 718.2 | 36.2 | 130 nm | 105 | 44.8 |
| [6] | 24 | 8, 2 | 968 | 60.6 | 180 nm | 275 | 31.7 |
| | | normalized | 504.9 | 31.6 | | | |
| [7] | 8 | 4, 2, | $5.85mm^2$ | n/a | 250 nm | 66 | 102.3 |
| | | approx. | 760.3 | | | | |
| | | normalized | 205.6 | | | | |

**Table 3.** comparison between FFT architectures

before starting on the design. The remaining four weeks to complete the transmitter design and the much of the receiver design. We estimate that it would take another 1-2 weeks to complete the receiver design.

One of the keys points which makes the OFDM framework so effective is the rich parameterization properties of Bluespec. Bluespec's rich type structure, parameterized types, and higher-order functions, made expressing much of the parameterized designs natural. Functions do not need to be represented as bit-vectors to be a parameter. Most of the work involved in using the library was simply understanding what block was desired.

The OFDM systems modular decomposition also proved to be highly valuable. because all modules were expected to be latency insensitive and have FIFO buffering, adding new stages to the pipeline and setting up complete testbenches were both easy. One can handle each input or output to a module separately.

Other Bluespec language features also proved to be helpful in a number of minor ways. Bluespec's static elaboration allowed the design to be expressed recursively; the system elaborates the description into non-recursive description automatically. Type provisos which represent the assumptions needed to use a function of module provided both useful documentation as well guarantees that designs are being parameterized in legal ways.

## 6 Conclusions

The primary goal of this work was not to see how much time and effort was needed in generating a new protocol; it was to see how effectively high-level design language ideas could be leveraged by engineers not already steeped in the language. In our views, this work has been a stunning success. The OFDM framework, though well structured, represents a fairly sophisticated system with significant parameterization. For inexperienced engineers to be able to understand, use, and even augment the system in so short a time argues for how natural the system is represented.

Our experience suggests that in the hands of experienced designers, our findings will be even more magnified, leading to greater focus on much larger design choices and hopefully leading to better implementation results.

## References


1. Ng, M.C., Vijayaraghavan, M., Dave, N., Arvind, Raghavan, G., Hicks, J.: From WiFi to WiMAX: Techniques for high-level ip reuse across different OFDM protocols. In: Proc. IEEE MEMOCODE, Nice, France (2007) 71–80
2. Arvind, Nikhil, R.S., Rosenband, D.L., Dave, N.: High-level Synthesis: An Essential Ingredient for Designing Complex ASICs. In: Proceedings of ICCAD'04, San Jose, CA (2004)
3. http://www.research.nokia.com/projects/armo
4. http://opensource.nokia.com/projects/armo-open-source-hardware/index.html
5. Dave, N., Pellauer, M., Gerding, S., Arvind: 802.11a Transmitter: A Case Study in Microarchitectural Exploration. In: Proceedings of Formal Methods and Models for Codesign (MEMOCODE), Napa, CA (2006)
6. Mathew, J., Maharatna, K., Pradhan, D., Vinod, A.P.: Exploration of power optimal implementation technique of 128-pt fft/ifft for wpan using pseudo-parallel datapath structure. In: Proc. IEEE ICCS, Singapore (2006) 1–5
7. Chen, S., Yu, Y., Chen, B., Lai, S., Zeng, Y., Zhang, Y., Wang, C.: Design of a 128-point fourier transform chip for uwb applications. In: Proc. IEEE ICSICT, Shanghai, China (2006) 1957 – 1959
8. IEEE: 802.15 High Rate Alternative PHY Task Group (tg3a) for Wireless Personal Area Networks, Multi-band OFDM physical layer proposal for IEEE 802.15 Task Group 3a, IEEE P802.15-03