

# Implementation of Wideband Multicarrier and Embedded GSM

Thomas Y. Tsou

Thesis submitted to the Faculty of the  
Virginia Polytechnic Institute and State University  
in partial fulfillment of the requirements for the degree of

Master of Science  
in  
Electrical and Computer Engineering

Jeffrey H. Reed, Co-Chair  
T. Charles Clancy, Co-Chair  
Robert W. McGwier

September 11, 2012  
Blacksburg, Virginia

Keywords: Wireless, Cellular Communications, Open Source Software, Software-Defined  
Radio, Multicarrier, Multirate, Digital Signal Processing

Copyright © 2012 by Thomas Y. Tsou

# Implementation of Wideband Multicarrier and Embedded GSM

Thomas Y. Tsou

## ABSTRACT

The Global System for Mobile (GSM) cellular standard, having been in existence for over two decades, is the most widely deployed wireless technology in the world. While third generation networks and beyond, such as Universal Mobile Telecommunications System (UMTS) and Long Term Evolution (LTE), are undergoing extraordinary growth and driving a large share of current cellular development, technologies and deployments based on GSM are still dominant on a global scale and, like more recent standards, continue to evolve very rapidly.

The software-defined radio (SDR) base station is one technology that is driving rapid change in cellular infrastructure. While commercial vendors have now embraced SDR, there is another movement that has recently gained prominence. That movement is the convergence of open source software and hardware with cellular implementation. OpenBTS, a deployable implementation of the GSM radio air interface, and the Universal Software Radio Peripheral (USRP), a RF hardware platform, are two primary examples of such open source software and hardware products. OpenBTS and the USRP underlie three GSM features that are implemented and presented in this thesis.

This thesis describes the extension of the OpenBTS software-defined radio transceiver in the three critical areas of user capacity, transmit signal integrity, and the embedded small form factor. First, an optimized wideband multicarrier implementation is presented that substantially increases the capacity beyond that of a single carrier system. Second, the GSM modulator is examined in depth and extended to provide performance that exceeds standards compliance by a significant margin. Third, operation of the GSM transceiver on an E100 embedded platform with ARM and fixed point DSP processors will be explored, optimized, and tested.

This work received support from the Communications-Electronics Research, Development and Engineering Center (CERDEC); Research, Development, and Engineering Command (RDECOM), United States Army.

# Acknowledgments

I would like to thank my advisory committee, fellow students, and countless members of the open source community. For it is through their guidance, support, and contributions that this work was made possible.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Contributions . . . . .	2
<b>2</b>	<b>Multicarrier GSM</b>	<b>3</b>
2.1	Polyphase Filterbanks . . . . .	5
2.1.1	Resampling . . . . .	5
2.1.2	Analysis Filter Bank . . . . .	8
2.1.3	Synthesis Filter Bank . . . . .	10
2.2	Implementation . . . . .	12
2.2.1	SIMD Optimization of Polyphase Filters . . . . .	14
2.2.2	DFT Computation on Non-Contiguous Data . . . . .	17
2.3	Results and Discussion . . . . .	17
2.4	Optional Features . . . . .	21
<b>3</b>	<b>Improved Signal Integrity with Linearized GMSK</b>	<b>24</b>
3.1	GSM Modulation . . . . .	24
3.2	Sampling Effects . . . . .	25
3.3	Continuous Phase Modulation . . . . .	28
3.4	Laurent Decomposition of CPM Signals . . . . .	29
3.4.1	GMSK Laurent Decomposition for L=3 . . . . .	30
3.5	Implementation . . . . .	31

3.6	Results and Discussion . . . . .	32
<b>4</b>	<b>Embedded BTS</b>	<b>36</b>
4.1	Hardware Platform Description . . . . .	36
4.1.1	Interprocessor Communication . . . . .	37
4.2	GSM Transceiver . . . . .	38
4.2.1	Transmitter Structure . . . . .	39
4.2.2	Receiver Structure . . . . .	39
4.3	Implementation . . . . .	40
4.3.1	ARM SIMD . . . . .	40
4.3.2	Texas Instruments C64x+ DSP . . . . .	42
4.4	Results and Discussion . . . . .	46
<b>5</b>	<b>Concluding Remarks</b>	<b>51</b>
	<b>Bibliography</b>	<b>52</b>

# List of Abbreviations

CIC	Cascaded integrator-comb
CPM	Continuous phase modulation
DSP	Digital signal processor
FFT	Fast Fourier Transform
FPGA	Field programmable gate array
GMSK	Gaussian minimum-shift keying
GPIO	General purpose input/output
GPMC	General purpose memory controller
ISI	Intersymbol interference
MCBTS	Multicarrier Base Transceiver Station
SIMD	Single instruction, multiple data
TCH	Traffic channel
TCH/H	Half rate traffic channel
TRX	Transceiver
USRP	Universal Software Radio Peripheral
VAMOS	Voice over Adaptive Multi-user channels on One Slot

# List of Figures

2.1	Traditional multi-transceiver BTS structure . . . . .	4
2.2	Multicarrier BTS with wideband power amplifier and baseband processing . . . . .	4
2.3	Direct form I/Q sample rate conversion . . . . .	5
2.4	Direct form resampler with combined image rejection and anti-aliasing filter . . . . .	6
2.5	Polyphase resampler . . . . .	8
2.6	Direct form channelizer . . . . .	9
2.7	Polyphase channelizer . . . . .	10
2.8	Direct form synthesis filter . . . . .	11
2.9	Polyphase synthesis filter . . . . .	12
2.10	OpenBTS uplink channelizer . . . . .	13
2.11	Tapped delay line for complex FIR filtering . . . . .	14
2.12	Intel SSE3 SIMD inner product summation for complex-real convolution with 4 tap FIR filter. Usage of single, double, and quad registers is labeled. . . . .	15
2.13	Comparison of SSE assembly intrinsics (blue) vs. GCC generated (red) filter operations. Displayed completion times are based on filtering of 10,000 GSM bursts of 2, 4, and 8 samples-per-symbol. . . . .	16
2.14	Vector optimization of 4x4 DFT on non-contiguous data . . . . .	18
2.15	Multiple handset uplink channels allocated on separate carriers, span = 4 MHz . . . . .	19
2.16	Spectrogram showing timeslots of 8 carrier MCBTS traffic, 4 MHz span . . . . .	20
2.17	MCBTS downlink (with activated secondary beacons), 2 MHz span . . . . .	20
2.18	Synthesis filter output with 45 GSM channels, 20 MHz span . . . . .	21
2.19	Spectrogram with time synchronized signals, 10 MHz span . . . . .	22

2.20	2x oversampled synthesis filter implementation . . . . .	23
2.21	Non-contiguous spectrum usage with 10 MHz LTE signal between composite (left) and GSM (right) signals, 16 MHz span . . . . .	23
3.1	GSM reference signal, phase error 0.29° RMS, 0.85° peak . . . . .	25
3.2	OpenBTS default configuration, phase error 4.98° RMS, 14.95° peak . . . . .	26
3.3	Oversampled OpenBTS spectrum with 8 samples-per-symbol, 1 MHz span . . . . .	27
3.4	Undersampled OpenBTS spectrum with 1 samples-per-symbol, 1 MHz span . . . . .	27
3.5	OpenBTS signal, 4 samples per symbol, phase error 1.97° RMS, 5.08° peak . . . . .	28
3.6	Laurent C0, C1, and C2 pulses for $BT = 0.30$ . . . . .	31
3.7	Laurent C0, C1, and C2 pulses for $BT = 0.20$ . . . . .	32
3.8	Laurent C0, C1, and C2 pulses for $BT = 0.40$ . . . . .	32
3.9	Linearized GMSK symbol mapper with single pulse . . . . .	32
3.10	Linearized GMSK symbol mapper with C0 and C1 pulses . . . . .	33
3.11	Combined tapped delay line for linearized GMSK pulses C0 and C1 . . . . .	33
3.12	OpenBTS–USRP1 phase error, 1 sample per symbol, C0 pulse, 4.98° RMS, 14.95° peak . . . . .	35
3.13	OpenBTS–USRP1 phase error, 4 samples per symbol, C0 pulse, 1.92° RMS, 5.44° peak . . . . .	35
3.14	OpenBTS–USRP1 phase error, 4 samples per symbol, C0 and C1 pulses, 1.66° RMS, 4.53° peak . . . . .	35
3.15	OpenBTS–USRP2 phase error, 4 samples per symbol, C0 and C1 pulses, 1.14° RMS, 2.97° peak . . . . .	35
4.1	E100 multi-processor platform . . . . .	37
4.2	C64x shared memory transport . . . . .	38
4.3	OpenBTS L1/PHY transmit chain . . . . .	39
4.4	OpenBTS L1/PHY receive chain . . . . .	40
4.5	NEON quad-register 8 element multi-lane load for deinterleaving I and Q data . . . . .	41
4.6	NEON multiply-accumulate instruction VMLA causes read after write (RAW) hazard . . . . .	43



4.7	NEON 8 tap convolution product-sum with pipeline compatible operations (only first sum is shown) . . . . .	44
4.8	Texas Instruments C64x DSP . . . . .	45
4.9	NEON <code>vmul/vsum</code> vs. <code>vmla</code> comparison . . . . .	47
4.10	C64x+ DSP complex convolution (blue) vs. static analysis (red) . . . . .	48
4.11	C64x+ DSP complex-real convolution vs. ARM . . . . .	49
4.12	C64x+ DSP complex-real convolution vs. ARM . . . . .	49

# List of Tables

3.1	Measured Phase Error . . . . .	34
4.1	Number of Supported TCH Channels on E100 . . . . .	50

# Chapter 1

## Introduction

In recent years, cellular implementations based on open source models have become viable alternatives to commercial products for certain conventional and unconventional deployments and applications. Conventional roles consist of mobile infrastructure, often for experimental or private use, where the open source approach may provide advantages of reduced cost and accessibility to source code and implementation details. In unconventional use cases, open source implementations have filled the role of niche applications in addressing needs not met by existing industry products; some examples include non-operator applications such as student education, security research, and tools for curious hobbyists.

OpenBTS is an open source software implementation of the GSM radio air interface, which is known as the *Um* in specification documents published by the 3rd Generation Partnership Project, the standardization body for GSM. For this thesis, the most important part of OpenBTS is the radio transceiver. The transceiver is notable because the physical layer is implemented in software with RF access provided by products such as the Universal Software Radio Peripheral, a class of flexible RF hardware devices. Because of this software radio approach, the transceiver design is extremely flexible and extensible to new capabilities.

Within the broader cellular industry, there is a continuing push for wider bandwidths, increased capacity, and greater performance out of ever smaller packages. In this context, OpenBTS is no different, and these driving factors provide the motivation to add features and extend the limits of OpenBTS and related open source implementations.

### 1.1 Motivation

The existing OpenBTS and USRP transceiver combination is capacity limited by a single carrier implementation. Given the relatively narrow bandwidth of GSM signals – hundreds of kHz vs. MHz for more recent systems – there should be sufficient capability to utilize higher

rates. But, an implementation that allows additional carrier use, and higher capacity, is not readily available to test this assumption. In addition, cellular industry practices mandate a high degree of testing throughout the air interface stack. At the physical signal level, specific tests and procedures are formalized by 3GPP. Users expect the signal integrity of OpenBTS to meet these requirements, but, for the most part, performance measurements currently remain unknown. These assumptions and unknown factors need to be answered.

In another area, the rise of increasingly capable smartphones – and the processors that drive them – has created a permanent shift in the world of computing. As an offshoot to the unceasing Moore's Law, the drive towards mobile has created a new class of embedded processors and devices. What role do these processors have for software-defined radio and, more specifically, cellular baseband implementation? This is the third issue that this thesis attempts to address.

## 1.2 Contributions

This work examines three limitations of the OpenBTS transceiver in the areas of capacity, transmit signal integrity, and physical form factor. Taken individually, the constituent parts of this thesis provide little advancement in terms of fundamental algorithms or breakthrough signal processing. The basic GSM formula is, of course, a well established standard with the first GSM phone call being placed over 20 years ago. This thesis does, however, create an operational cellular implementation that is unique in that it leverages a combination of established signal processing concepts, SDR implementation methodologies, and a substantial base of existing open source projects and code.

The contributions of this thesis are presented in the following three chapters. Chapter 2 presents a path from multirate theory to optimized real time implementation of a high capacity multicarrier GSM base station. Chapter 3 evaluates and quantifies transmit signal performance of OpenBTS along with a thorough investigation of GSM modulation. Chapter 4 describes the extension of OpenBTS to an embedded heterogeneous processor platform.

# Chapter 2

## Multicarrier GSM

A single carrier GSM cell site is capacity limited to 8 physical channels. With one channel dedicated to beacon transmission and other control purposes, seven remaining channels are available for traffic channel (TCH) use. As a result, this typical configuration is limited to seven simultaneous full rate voice connections. For OpenBTS systems, this is the default and most common setup. The goal of this chapter is to increase the capacity of an OpenBTS system by addressing this single carrier limitation.

There are currently three standardized methods that can be used to increase user capacity of a GSM system. These include the following 3GPP specified approaches:

1. Half-rate channels (TCH/H) [1]
2. Voice over Adaptive Multi-user channels on One Slot (VAMOS) [2]
3. Multiple transceivers (TRX)

The use of half-rate traffic channels doubles voice capacity by reducing the number of time slots available to a user by half. The consequence, though, is a reduction in bit rate and degradation in voice quality. VAMOS is a 3GPP Release 9 item that aims to double voice capacity through the use of orthogonal subchannels on a single time slot - this recent specification has not been largely deployed. In this chapter we focus on the third and most fundamental approach to increasing capacity - bandwidth.

The traditional approach to raising capacity of GSM deployments is the use of multiple transceivers where each additional unit supplies a carrier and 8 new timeslots (physical channels). These transceivers take the form of standalone components feeding into a RF combiner and single antenna as shown in Figure 2.1. A limitation of this traditional approach is the replication of RF components and the need for physical installation for each each new unit. Furthermore, this cases suffers from physical losses, which are a result of RF combining and the overhead of maintaining separate power amplifiers.

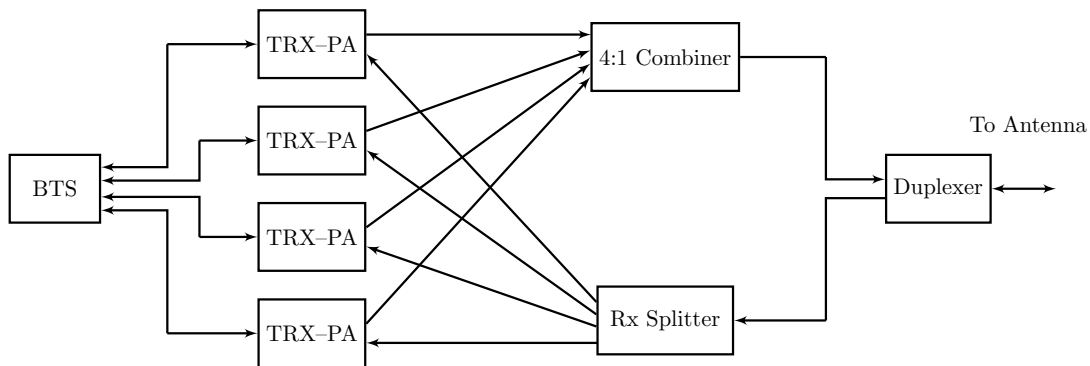


Figure 2.1: Traditional multi-transceiver BTS structure

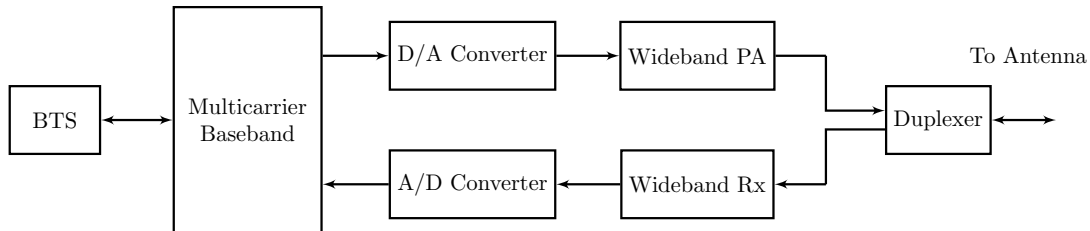


Figure 2.2: Multicarrier BTS with wideband power amplifier and baseband processing

Recently, two primary issues have been driving the base station industry away from the independent transceiver model towards alternative GSM implementations that are more efficient and more flexible. These factors are, first, the ability to handle incremental increases and reductions in GSM capacity and, second, efficient upgrade paths to more recent UMTS and LTE standards. These factors are, of course, part of larger efforts to reduce capital and operating expenses. In addition, there are economic and regulatory pressures to move toward more flexible spectrum utilization. In response to these concerns, 3GPP Release 8 introduced the standardization of the *multicarrier BTS* (MCBTS) [3].

Enabled by the availability of wideband amplifiers, high speed baseband processing units, and the application of new and established multirate signal processing techniques, MCBTS implements multiple transceivers with a single baseband unit and wideband RF front end. These factors lead us to the integrated design shown in Figure 2.2. Development of wideband RF components and high bandwidth baseband capabilities has been spurred by deployments of wideband 3G technology and the continuing advancement of Moore's Law respectively [4]. When combined with the final aspect, the enabling aspect of advanced signal processing algorithms, the MCBTS becomes an appropriate model for OpenBTS and software-defined radio implementation. The theory, operations, and implementation of a MCBTS system based on OpenBTS is the topic for the remainder of this chapter.

To start, efficient multirate signal processing techniques that enable multicarrier operation

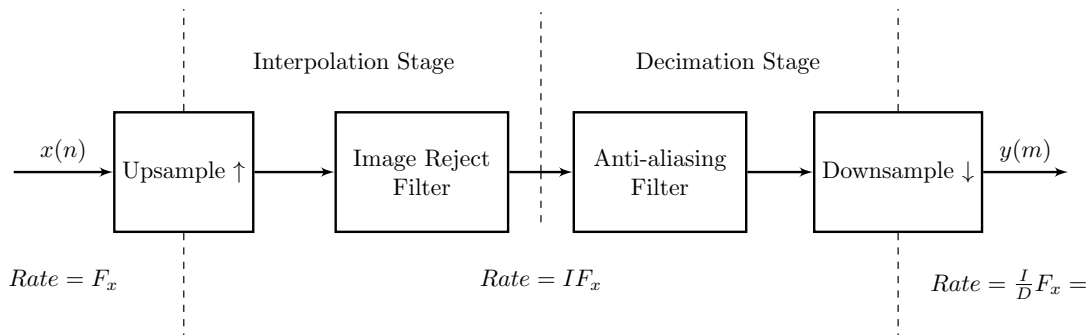


Figure 2.3: Direct form I/Q sample rate conversion

are presented and examined with respect to simpler, but inefficient, direct form approaches. The mathematical analysis is followed by the description of a real time multicarrier implementation that builds on the OpenBTS software radio transceiver and USRP hardware. Finally, the end result, a substantial increase in user capacity and much greater spectrum flexibility, is discussed along with deployment implications.

## 2.1 Polyphase Filterbanks

There are three primary applications of multirate polyphase filters applicable to the cellular applications described in this chapter. We start with the process of sample rate conversion using the basic operations of interpolation and decimation. Second is the analysis filter bank - also referred to as a channelizer - used to demultiplex individual receive signals from the single wideband base station uplink. Third, opposite of the channelizer is the synthesis filter bank that combines multiple independent narrowband streams into a single wideband output. In all cases, we will progress from simple designs to more sophisticated forms that are appropriate for high bandwidth SDR implementation. The analysis provided in this section is largely drawn from the texts of Crochiere and Rabiner [5] and fred harris [6].

### 2.1.1 Resampling

The task of sample rate conversion by a rational factor of I over D, that is upsampling by a factor of I followed by downsampling by a factor of D, is perhaps the most fundamental pair of operations in multirate applications. We begin exploring this task by introducing a direct form implementation and following with a reconstruction that leads to the computationally efficient, but still mathematically equivalent, polyphase variant.

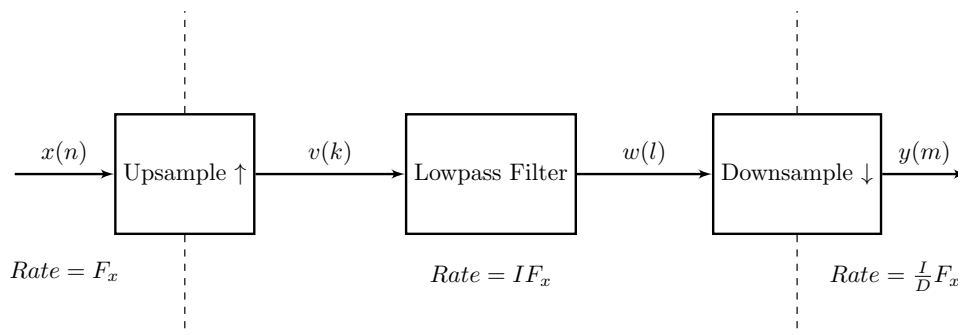


Figure 2.4: Direct form resampler with combined image rejection and anti-aliasing filter

### Direct Form Resampler

The basic I/Q resampling process can be described by the two separate processes of interpolation and decimation as shown in Figure 2.3. We find a upsampling stage - zero insertion - followed by by an FIR filter to remove the resulting images that appear at Nyquist intervals in the zero-padded signal. The interpolated output is followed by a decimation stage, which includes anti-aliasing filtering prior to downsampling.

The obvious inefficiency in the above structure is the redundancy of back-to-back filtering operations. A combined filter design that encapsulates both image rejection and anti-aliasing requirements can replace the dual filters as shown in Figure 2.4.

Through we have removed the redundancy of filters, this design is still inefficient due to the arrangement of the upsampling and downsampling operations. Note that the anti-aliasing filter multiplies primarily zeros and, furthermore,  $1/D$  of these computed samples are discarded in the downsampling process. Therefore, the direct form resampler implementation computes many unnecessary samples given that a large portion of these samples are unused in the final output.

In order to find a more efficient solution, we now examine the inputs and outputs of the above structure.

The upsampler output is given by

$$v(l) = \begin{cases} x(l/I), & \text{for } l = 0\pm, \pm I, \pm 2I, \dots \\ 0, & \text{otherwise} \end{cases}$$



with the following lowpass filter output

$$\begin{aligned}\omega(l) &= \sum_{k=-\infty}^{\infty} h(mD - k)v(k) \\ &= \sum_{k=-\infty}^{\infty} h(mD - kI)x(k)\end{aligned}\quad (2.1)$$

Once the filtered output passes through the downsampler, we are left with

$$\begin{aligned}y(m) &= \omega(mD) \\ &= \sum_{k=-\infty}^{\infty} h(mD - kI)x(k)\end{aligned}\quad (2.2)$$

At this point, we introduce an alternate formulation with a change of variables. Let

$$r = \left\lfloor \frac{m}{L} \right\rfloor - n \quad (2.3)$$

where  $\lfloor n \rfloor$  is the integer floor of  $n$ . This gives us output of

$$y(m) = \sum_{n=-\infty}^{\infty} h\left(mD - \left\lfloor \frac{mD}{I} \right\rfloor I + nI\right)x\left(\left\lfloor \frac{mD}{I} \right\rfloor - n\right) \quad (2.4)$$

with the key characteristic of cyclic operation in that

$$mD - \left\lfloor \frac{mD}{I} \right\rfloor I = mD \bmod I \quad (2.5)$$

Applying the above modulo expression gives us our final expression

$$y(m) = \sum_{n=-\infty}^{\infty} h(nI + mD \bmod I)x\left(\left\lfloor \frac{mD}{I} \right\rfloor - n\right) \quad (2.6)$$

This leads to the critical result of the preceding resampling analysis - the output of the interpolation and decimation pair is fully described by the convolution of the input sequence and a time varying cyclic filter. The properties of this time varying filter provide the basis for which we can construct an equivalent representation using *polyphase filters*.

## Polyphase Resampler

From Eq. (2.6) we have a set of periodic filters. It can be shown that these filters can be represented by the set of impulse responses formed by

$$\begin{aligned}p_k(n) &= h(k + nI) & k &= 0, 1, 2, \dots, I - 1 \\ & & n &= 0, 1, 2, \dots, K - 1\end{aligned}\quad (2.7)$$

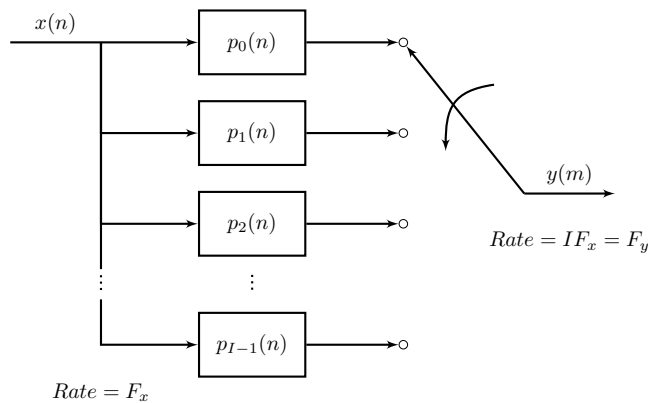


Figure 2.5: Polyphase resampler

This process of constructing a set of smaller filters of this form from an original *prototype filter* is known as *polyphase decomposition*. Similarly, these new filters are called polyphase filters.

For the case of interpolation only, where  $D = 1$ , the selection of filters per successive output is cyclic in  $m$  over the period  $I$ . That is, each output is generated from one of the polyphase filters, which are selected incrementally from the generated set; after the  $I - 1$  filter, we wrap around to the first filter. This filter selection process is conveniently represented by a rotating output commutator as shown in Figure 2.5. The commutator representation draws from the rotary commutator unit found in electric motors. At this point, we can apply the decimation factor  $D$  by manipulating the output commutator. For example, a decimation rate  $D = 2$  leads to the commutator moving two filter positions for every generated output sample. At the boundary filters,  $p_0$  and  $p_{M-1}$ , the commutator rotation wraps with respect to the modulo expression in Eq. (2.6).

Given this structure, we are ready for resampler implementation, which will be shown in later sections. Before we explore actual real time operation, however, we need to examine the other critical polyphase components that will work in conjunction with the resampler inputs and outputs.

### 2.1.2 Analysis Filter Bank

The analysis filter bank performs the process of signal decomposition; the term *analysis* in this case refers to the analysis of a signal in terms of its constituent sub-band components. For the purposes of multicarrier GSM, the analysis filter bank extracts individual narrowband channels from a much larger wideband signal. The analogous operation where a composite signal is reconstructed from smaller subbands is known as *synthesis* and will be examined

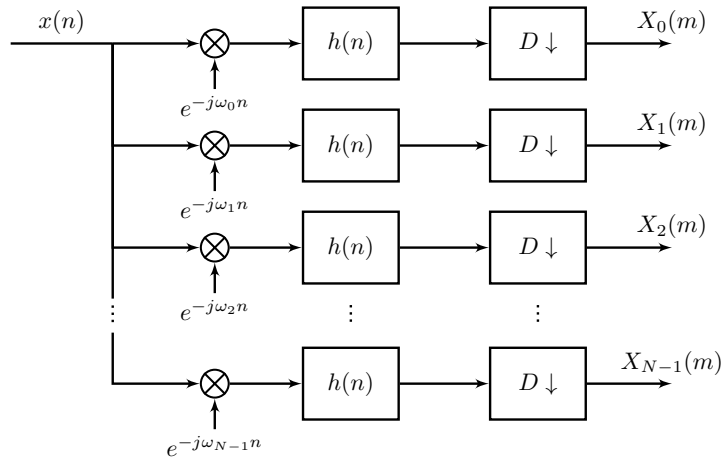


Figure 2.6: Direct form channelizer

in the next section. For the remainder of this chapter, we use the terms analysis filter bank and channelizer interchangeably.

As in the case of sample rate conversion, we start from an intuitive direct form approach and progress to an efficient polyphase implementation. The basic form of our desired application is shown in Figure 2.6. In this simple approach, each individual channel is mixed to baseband by the multiplication of a complex exponential, low pass filtered to isolate the channel of interest, and finally downsampled to the desired frequency.

We assume for this discussion that the target frequencies are equally spaced and that the individual channel bandwidth is  $1/M$  of the total aggregate bandwidth. That is, the channelizer operates in a *maximally decimated* configuration. This assumption is appropriate for the case of MCBTS since the channel spacing for GSM is consistent at 200 kHz – we will later utilize 400 kHz channel separation given that adjacent channels are overlapping.

With no change in functionality, we can rearrange the order of operations shown in Figure 2.6. In this case, the lowpass filters are replaced with bandpass counterparts and downconversion is performed after decimation. Given this construction, the filters now have the following impulse responses

$$h_k(n) = h(n)e^{j2\pi nk/N}, \quad k = 0, 1, \dots, N-1 \quad (2.8)$$

with the filter bank outputs

$$X_k(m) = \left[ \sum_n x(n)h(mD-n)e^{j2\pi k(mD-n)/N} \right] e^{-j2\pi mkD/N} \quad (2.9)$$

Now, let us define the set of polyphase filters with the following impulse responses.

$$p_k(m) = h(nN-k), \quad k = 0, 1, \dots, N-1 \quad (2.10)$$

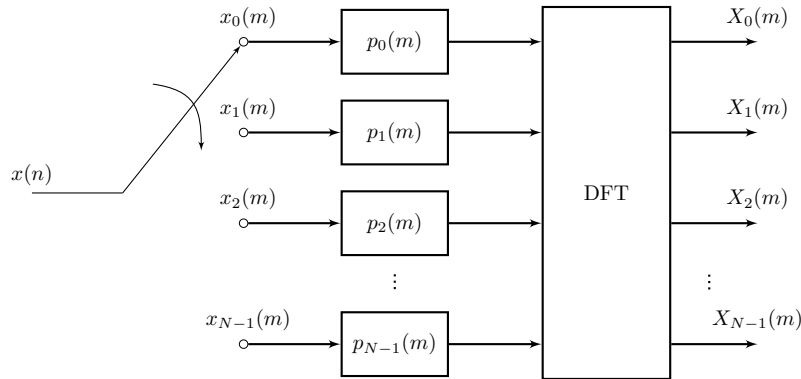


Figure 2.7: Polyphase channelizer

Using a commutator to decimate the input sequence, as in the previous section, the sequence that feeds each respective polyphase filter is represented by

$$x_k(n) = x(nN + k), \quad k = 0, 1, \dots, N - 1 \quad (2.11)$$

This definition assumes clockwise rotation of the input commutator or, alternatively, top-to-bottom loading of the polyphase paths. The previously described structure can now be combined with the polyphase partitions to arrive at the following form

$$X_k(m) = \sum_{n=0}^{N-1} \left[ \sum_l p_n(l) x_n(m-l) \right] e^{-j2\pi nk/N} \quad (2.12)$$

The inner summation can be represented as  $N$  inner convolution operations. For the outer summation, recall that discrete Fourier transform (DFT) is defined by

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j2\pi kn/N} \quad (2.13)$$

Consequently, we arrive at the critical result – the analysis filter bank is represented by a set of convolutions and a  $M$ -point DFT of the convolved filter outputs. Similar to the resampler representation, the input to the channelizer is analogous to a rotating commutator. The final representation is shown in Figure 2.7. This resulting design is extremely important because the substitution of the DFT allows us to use highly efficient Fast Fourier Transform (FFT) algorithms and implementations instead of a costly bank of complex exponential multiplications.

### 2.1.3 Synthesis Filter Bank

Opposite of the analysis filter is the synthesis filter, which reconstructs the downlink wide-band signal from constituent narrowband streams. Given these analogous operations, we

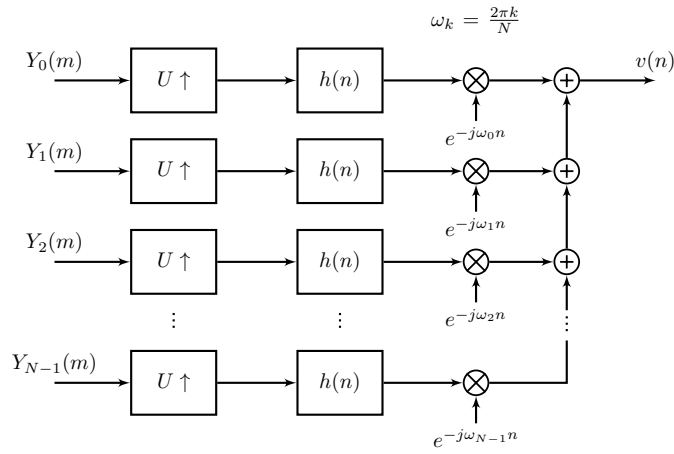


Figure 2.8: Direct form synthesis filter

start with a similar representation as used for the synthesis filter bank, which is shown in Figure 2.8. Once again, we can rearrange order of operations with no change in functionality.

The transformation of the basic synthesis filter with output  $v(n)$  is described by

$$\begin{aligned}
 v(n) &= \frac{1}{N} \sum_{k=0}^{N-1} e^{j2\pi nk/N} \left[ \sum_m Y_k(m) h(n - mI) \right] \\
 &= \sum_m h(n - mI) \left[ \frac{1}{N} \sum_{k=0}^{N-1} Y_k(m) e^{j2\pi nk/N} \right] \\
 &= \sum_m h(n - mI) y_n(m)
 \end{aligned}$$

Taking the same approach as the analysis filter bank, but inverted, we can rewrite the filtering model with a polyphase decomposition given by

$$p_k(n) = h(nN + k) \quad (2.14)$$

with the corresponding output of each partitioned polyphase filter

$$v_k(n) = v(nN + k) \quad (2.15)$$

Now, with a counterclockwise commutator representation, we can substitute the polyphase filters and rearrange the  $l$ th filter output as

$$v_l(n) = \sum_m p_l(n - m) \left[ \frac{1}{N} \sum_{k=0}^{N-1} Y_k(m) e^{j2\pi kl/N} \right] \quad (2.16)$$

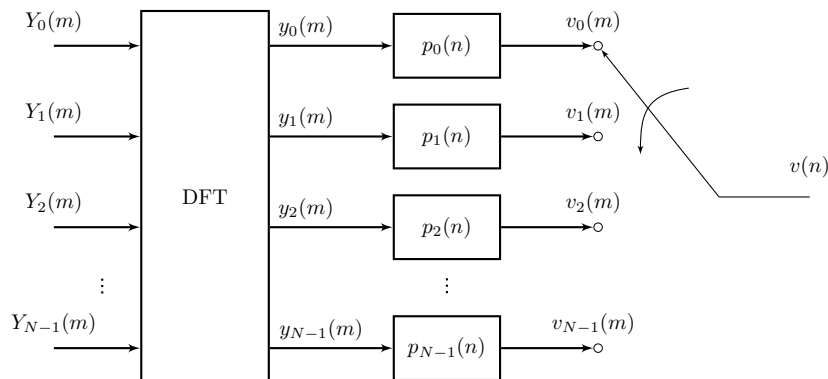


Figure 2.9: Polyphase synthesis filter

Again, recalling the inverse DFT

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{j2\pi kn/N} \quad (2.17)$$

leads us to the synthesis filter counterpart of the previously described analysis filter bank. Compared to the analysis filter, we have a reverse in signal flow and a shift in the location of the commutator and reverse DFT. The effective polyphase synthesis filter structure is shown in Figure 2.9.

## 2.2 Implementation

The previous section provided the analytical building blocks for efficient construction of a multicarrier GSM system. This section translates the previous constructs into a deployable software radio implementation of a multicarrier BTS. For the initial implementation, we focus on the maximally packed configuration and design for 400 kHz channel spacing. As an extension, we will also discuss a more flexible 200 kHz approach.

For brevity, we focus primarily on wideband signal reception on the BTS uplink ; the down-link transmitter follows a reciprocal design. The overall channelizer and receiver design is shown in Figure 2.10. Sample rate conversion blocks are utilized in two different areas. Resampling blocks are located on each channel to accommodate the difference between the 400 kHz channel spacing and the 270.833 kHz symbol rate of the individual streams; the effective resampling ratio is therefor 65/96. Alternatively, resampling can be performed internally within the channelizer. Given the ratio of 65/96, however, this approach and the complexity of the necessary state machine quickly becomes intractable while providing limited benefit. The outputs of the low rate resamplers feed into a bank of OpenBTS receivers. Each receiver instance is identical to that of a default single channel configuration.

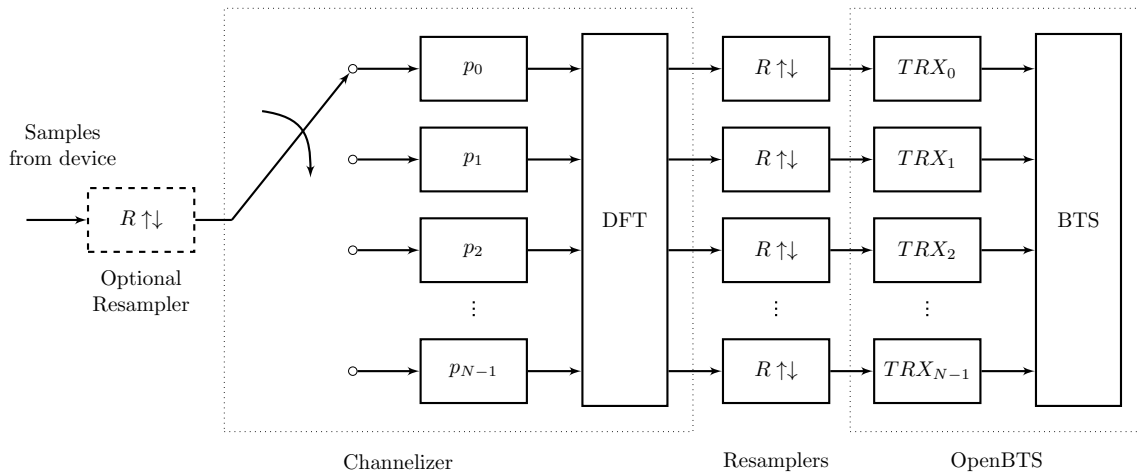


Figure 2.10: OpenBTS uplink channelizer

Optionally, a resampler block can be placed between the device receive signal and channelizer input. This approach is undesirable because the signal processing occurs on the high rate side of the channelizer, incurring higher computational cost, but may be useful under two situations. First, the device may not support the desired channelizer rate,  $M * 400kHz$  in our case, which makes the block mandatory. Second, certain sample rates may have better frequency domain characteristics than others. For example, the USRP uses different configurations of half-band and CIC filters on board the FPGA, which result in different frequency response characteristics as the sample rate is changed. With resampling of the device signal, a rate that minimizes pass-band distortion from the device can be selected.

Given the above design, the next step is to carry over the computational efficiency objectives from the previous section to hardware specific software optimizations. The target processor architecture for this section is Intel/AMD x86 (embedded instances for ARM and Texas Instruments DSP are examined in Chapter 4). The processor specification of the test systems is as follows:

```
processors      : 4
model name     : AMD Phenom(tm) 9850 Quad-Core Processor
cpu MHz       : 2500
cache size    : 512 KB
```

Relatively recent processors are emphasized in that the presence of SSE3, Streaming SIMD Extensions 3, instructions are assumed. The implementation carries out optimizations in the two critical tasks of filter bank convolutions and DFT computation.

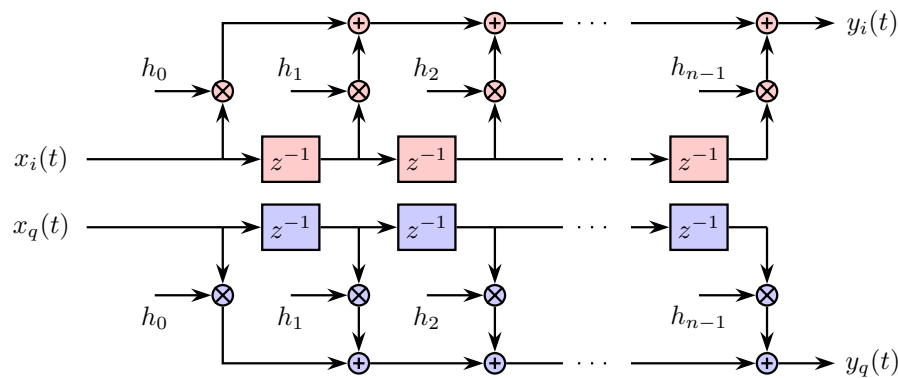


Figure 2.11: Tapped delay line for complex FIR filtering

### 2.2.1 SIMD Optimization of Polyphase Filters

One implementation advantage of polyphase filters is that large prototype filters are broken up into smaller polyphase partitions. The reduction in the number of filter taps has the advantage that more aggressive code optimizations can be performed with regard to CPU register loading and unloading. To illustrate, Figure 2.11 shows the convolution operation that would be used to compute a complex valued input sequence filtered with real valued filter coefficients. The operations are represented in the form of a paired tapped delay line with separate in-phase and quadrature processing paths. This structure is used repeatedly in the polyphase resampler, channelizer, and synthesis filter components.

For optimized performance, the implementation takes the underlying hardware into consideration by using instruction set specific operations. More specifically the implementation makes full use out of single instruction, multiple data (SIMD) instructions to exploit *data level parallelism* inherent to filtering and other signal processing tasks.

The Intel SSE register file provides 16 128-bit registers that each hold 4 32-bit floating point values [7]. For best performance, we restrict convolutions lengths to multiples of 4 between 4 and 20 to maximize efficiency and reduce overhead when using 128-bit registers. The SIMD implementation takes the form of the sequence of multiply and sum operations shown in Figure 2.12. Note that there is no dedicated multiply-accumulate operation in the x86 SSE3 instruction set, so multiplies are and product sums are carried out in separate operations.

Benchmark results based on filtering of 156-length symbol sequences, the length of a one GSM time slot burst, are shown in Figure 2.13 for oversampling rates of 2, 4, and 8 samples-per-symbol. These data points for SSE implementation are compared against the compiled C code from GCC. The overall metric is completion time for filtering 10,000 sequences using filter lengths between 4 and 20.



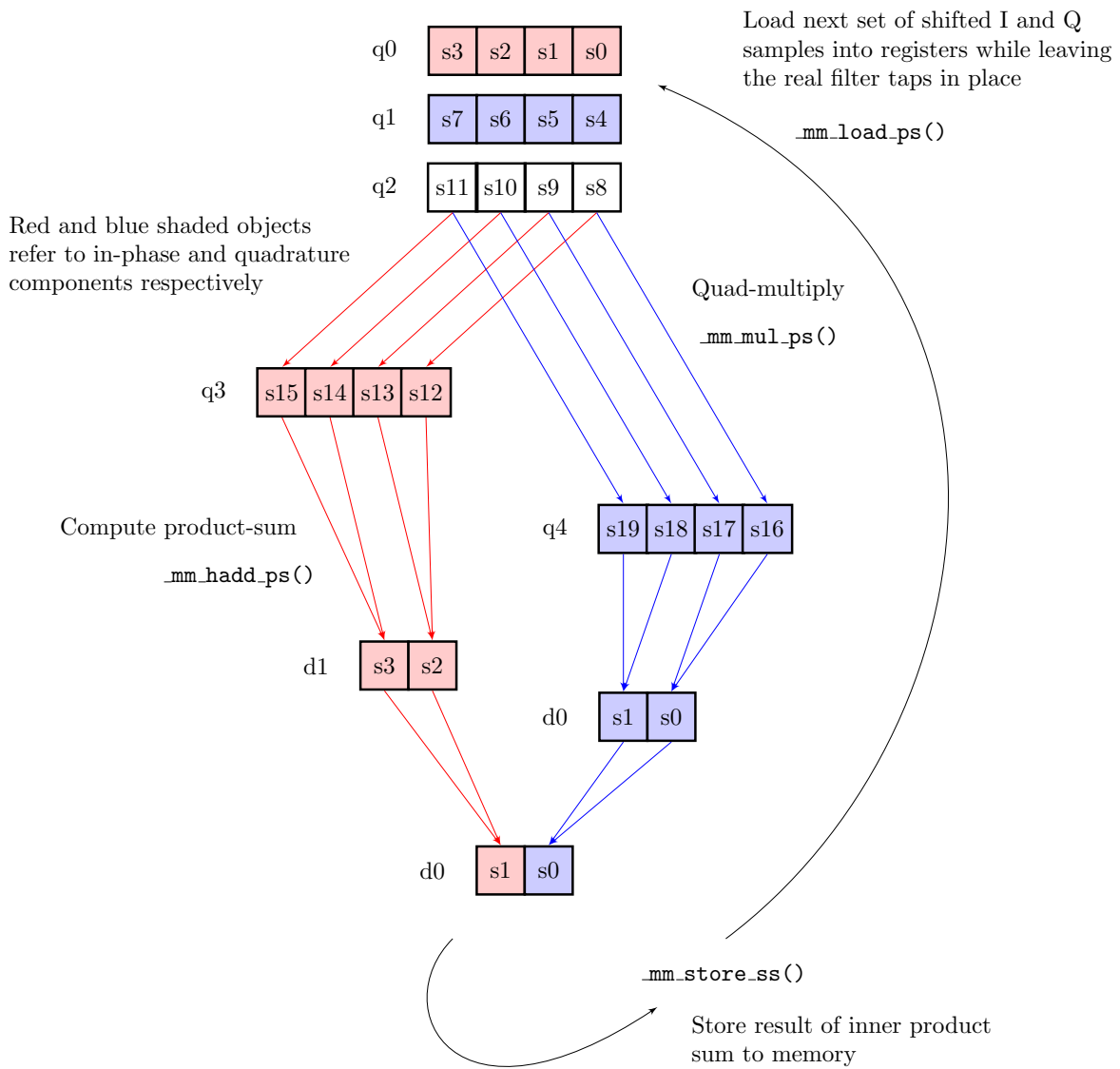


Figure 2.12: Intel SSE3 SIMD inner product summation for complex-real convolution with 4 tap FIR filter. Usage of single, double, and quad registers is labeled.

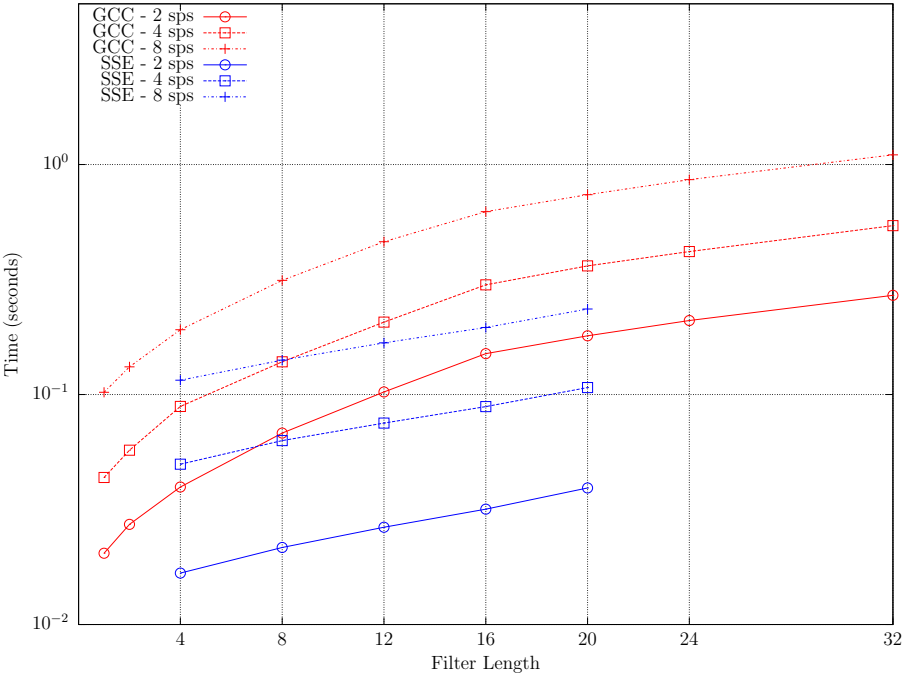


Figure 2.13: Comparison of SSE assembly intrinsics (blue) vs. GCC generated (red) filter operations. Displayed completion times are based on filtering of 10,000 GSM bursts of 2, 4, and 8 samples-per-symbol.

## 2.2.2 DFT Computation on Non-Contiguous Data

A key motivation for using the polyphase channelizer and synthesis filter implementation is that we can replace as large number of multiplication operations with a single DFT. This gives us the advantage that very computationally efficient FFT methods can be used. For the implementation we utilize “Fastest Fourier Transform in the West” (FFTW), an open source software package for computing discrete Fourier transforms. FFTW is well known as the fastest freely available implementation of the FFT algorithm. Like the above SIMD optimizations, FFTW also uses similar efficient SSE instructions internally for architecture specific optimization.

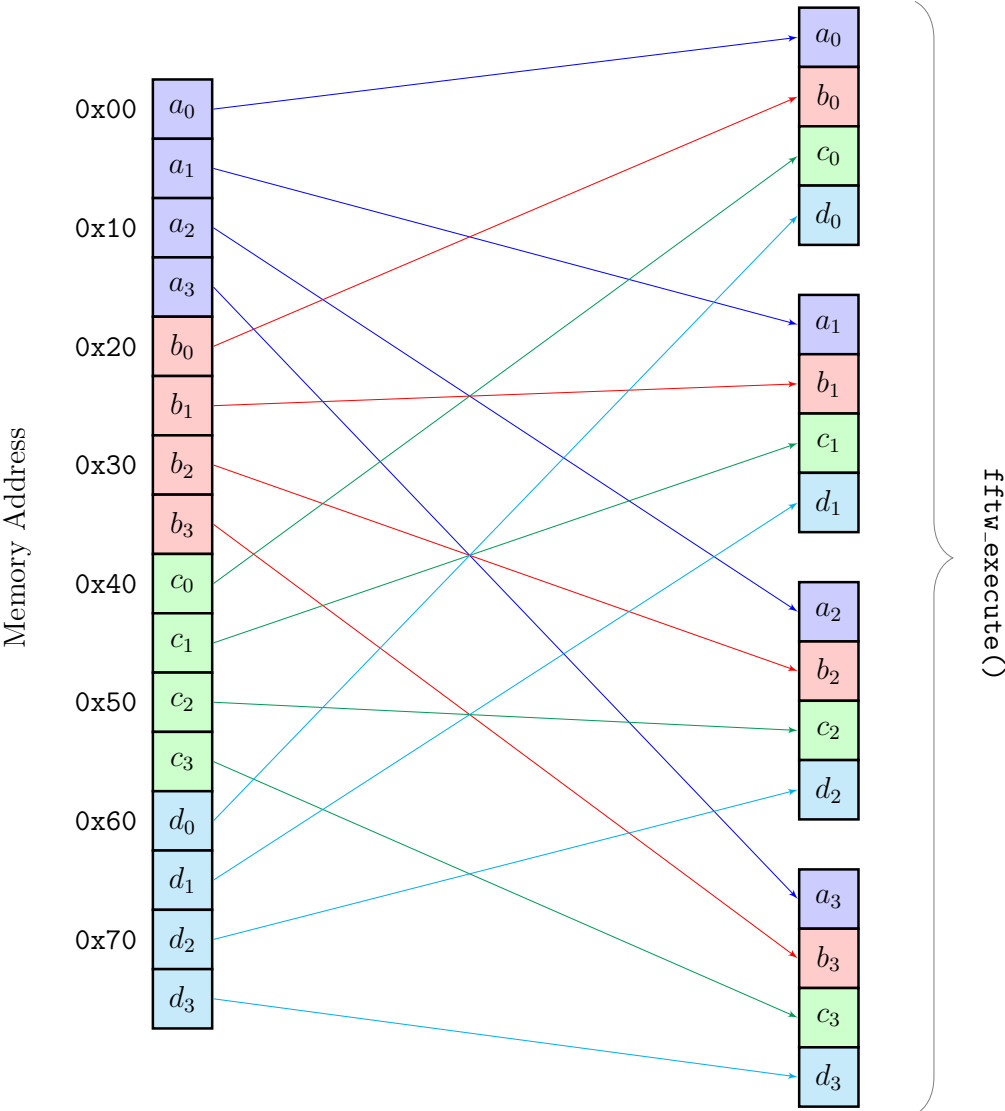
Furthermore, FFTW elegantly handles the complex task of manipulating data storage when dealing with the filter bank inputs and outputs. The critical issue here is that each filter bank output writes data into individual continuous buffers. The DFT operates on a single sample from each partition buffer such that the data accesses are across non-contiguous memory storage. There are two approaches to handle this situation. First, the data can be interleaved into continuous blocks for the FFT computation or, second, the FFT can be computed on discontinuous data. For the latter each sample would be drawn with an offset, known as the stride count, related to the number of polyphase partitions.

Fortunately, FFTW is capable of operating on discontinuous data, so the second approach was chosen to manipulate disjoint data directly and avoid unnecessary data interleaving or shuffling. Figure 2.14 demonstrates the data mapping and computation of a simplified  $M = 4$  configuration. For this configuration, a single call into FFTW performs 4 FFT operations of size 4 on disjoint data. The buffer depth is also chosen to be 4. On the left side of the figure there is a linear arrangement of continuous data buffers. The right side of the figure shows the element mapping for the FFT operation. This example implementation computes 4 FFT operations of length 4 with a single function call.

## 2.3 Results and Discussion

We now provide overall operational results of the multicarrier BTS implementation. Figures 2.15 and 2.16 demonstrate OpenBTS operation with 8 carriers on 8 MHz of combined uplink and downlink bandwidth. The spectrum display shows uplink signals from 8 handsets on separate carrier. Recall that each carrier has 8 physical channels, so this configuration has maximum capacity of 63 full rate voice channels after accounting for the broadcast beacon. For this example, handsets were allocated to separate carriers for demonstrative purposes. Deployed systems may use other channel assignment schemes.

Note that the signals shown on the spectrum display have a faint image because the handsets are not continuously transmitting as they operate with Tx/Rx switching. Since each traffic channel occupies a single physical TDMA channel, or 1/8th of the complete frame time, the



DFT computation on non-contiguous complex transform data using multi-dimensional `fftwf_plan_many_dft()` operation and stride count

Figure 2.14: Vector optimization of 4x4 DFT on non-contiguous data

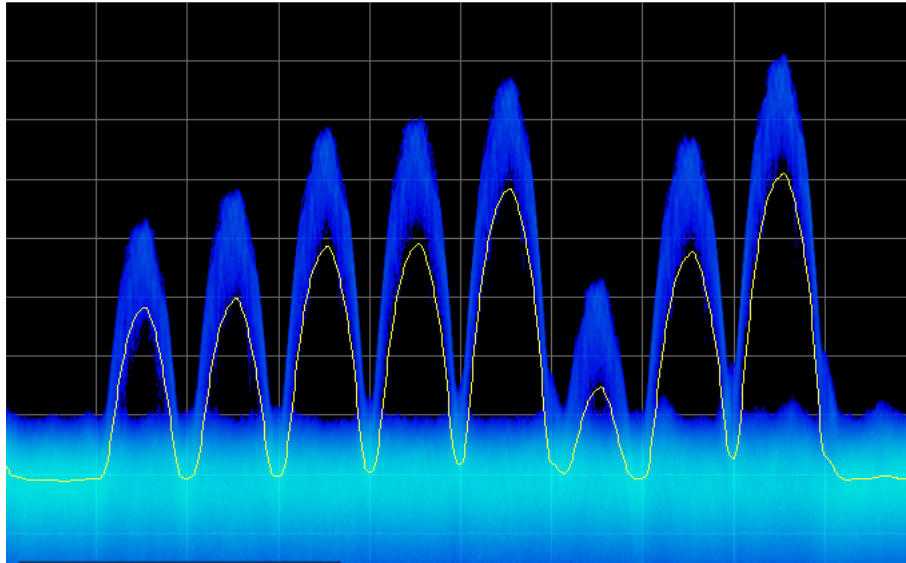


Figure 2.15: Multiple handset uplink channels allocated on separate carriers, span = 4 MHz

effective duty cycle of the mobile transmitter is below 15% after guard intervals are considered. Also note that the spectrogram construction uses overlapped measurement windows, so the exact placement of bursts in time (Y-axis) relative to each other is not accurate. A more precise representation would show subsequent bursts evenly spaced at the GSM frame period of 4.615 ms.

Figure 2.17 shows the downlink with a narrower span of 2 MHz. The beacon channel – with the noticeable FCCH peak right of the center frequency – is the leftmost signal. For illustrative purposes, beacon signals are also activated on the non-primary channels - for normal operation only the primary carrier would be broadcasting a continuous signal on all time slots.

For performance comparisons, operational testing also consisted of single direction benchmarks in which only the downlink signal was enabled while using the highest available bandwidth possible. Figure 2.18 shows maximum supported bandwidth in the downlink only test. The synthesis filter is operating at the device limited bandwidth of 20 MHz in a 50 channel x 400 kHz configuration. The USRP2 support 25 MHz with 1 Gbps Ethernet, however, the OpenBTS transceiver requires a single receive to timestamp delivery and synchronization of with the USRP sample clock. Consequently, this receive channel limits higher sample rates although CPU utilization may allow higher rates and more channels. Here, the transceiver is limited to a reduced maximum rate of 20 MHz.

Figure 2.19 shows the spectrogram output and that all channels are time synchronized. Internal to the software radio implementation, all transceivers are locked to the same GSM frame clock, which is an incremental time slot counter. Because all transmitters originate from a single baseband source, time synchronization is a trivial matter and significantly

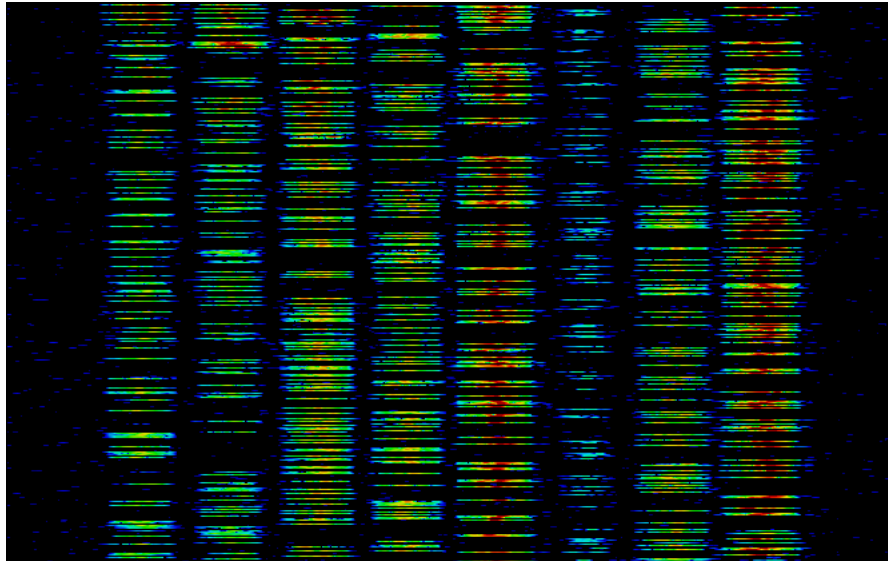


Figure 2.16: Spectrogram showing timeslots of 8 carrier MCBTS traffic, 4 MHz span

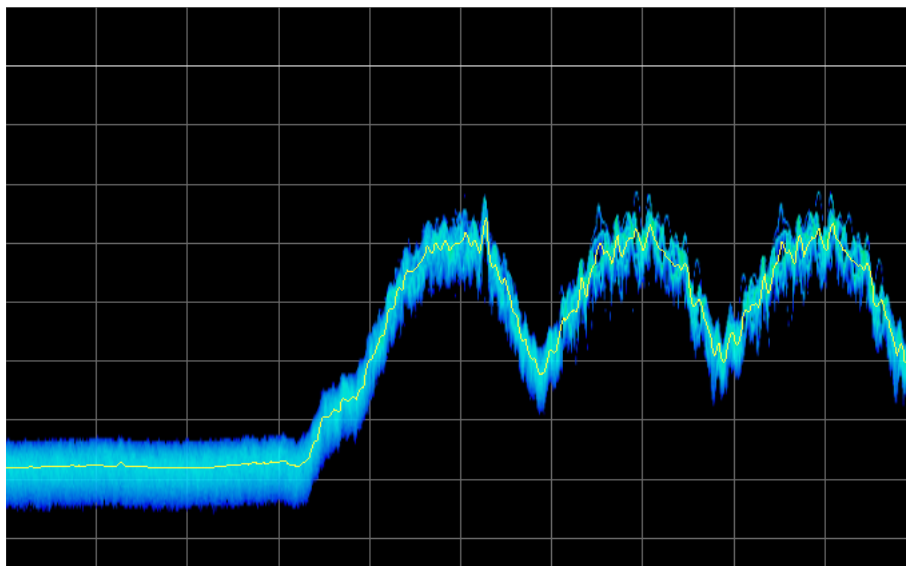


Figure 2.17: MCBTS downlink (with activated secondary beacons), 2 MHz span

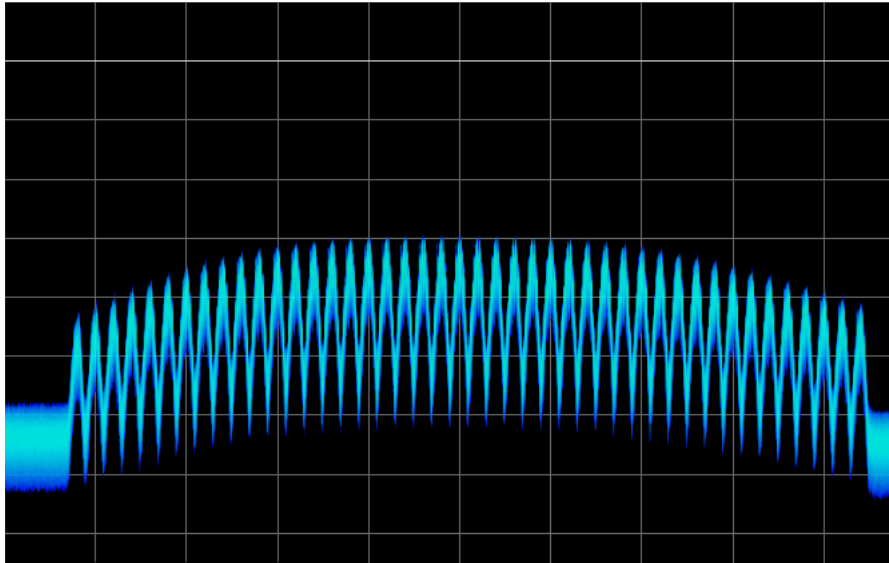


Figure 2.18: Synthesis filter output with 45 GSM channels, 20 MHz span

simpler than an implementation using physically separated transceivers.

## 2.4 Optional Features

For prior sections, the multicarrier operation was setup for 400 kHz channel separations, however, GSM specified spacing is actually 200 kHz with overlapping channels. For a maximally packed GSM configuration, there is no requirement for supporting 200 kHz channel placements because these channels would be self-interfering. There are other situations, however, where more flexible selection of frequencies is desired. In order to allow more flexibility in channel assignments, we now extend the existing design to allow any GSM channel selection.

In order to operate at 200 kHz instead of 400 kHz spacing, we need to double the number of channels in the filter bank. To maintain the 400 kHz sampling rate, rather than halving the channelizer output, the individual channels are oversampled by a factor of two.

We primarily focus on the synthesis filter used to aggregate independent channels on the base station downlink direction and, for brevity, we describe the implementation without derivation. Analysis of the oversampled filter bank can be found in [6].

Figure 2.20 shows the oversampled synthesis filter implementation for oversampling factor of two. Existing texts typically represent the oversampled filter bank with a cyclic, or serpentine, shift of the polyphase filters and the corresponding commutator phase. These shifts would then occur on alternating samples, or every other sample for a oversampling

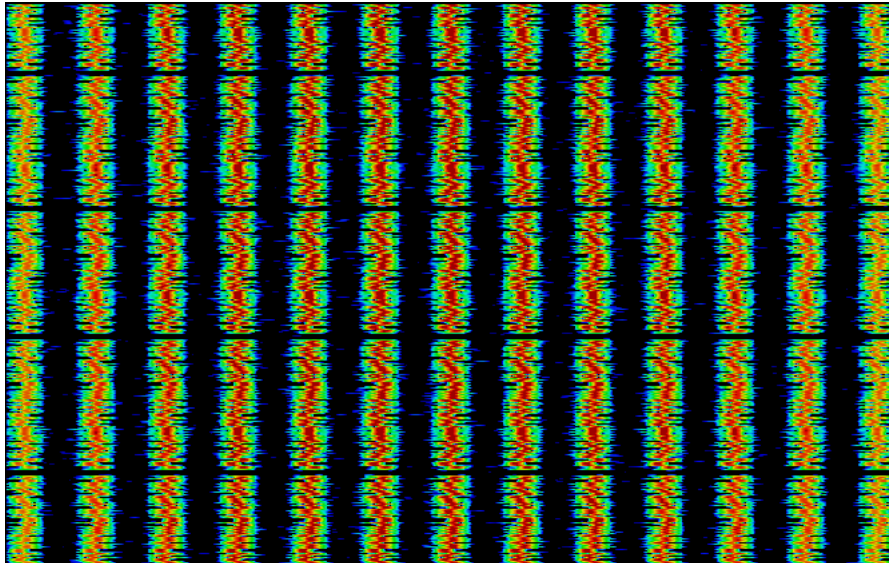


Figure 2.19: Spectrogram with time synchronized signals, 10 MHz span

rate of two.

For the OpenBTS implementation we address the cyclic shift by deinterleaving even and odd samples into separate filter banks with pre-shifted polyphase partitions. Note that in the actual implementation, the shifts are arranged with memory pointer swaps. The commutator phase difference is absorbed into a second commutator on the second filter bank. The overall effect is that the second filter bank is offset, or delayed, with respect to the first.

The end result is a great deal of flexibility including the capability to precisely place and overlap GSM channels. At first glance, this feature may not seem particularly useful. When we consider that the channelizing filter bank may be extended to interaction with other non-GSM signals, there is a high potential for use in more general applications. For example, Figure 2.21 shows an example of a composite signal constructed from 200 kHz channels and a GSM channel. With a 20 MHz filter bank, we assign roughly 10 MHz separation between the two different signals.

As the figure shows, this configuration has the capability of running two distinct waveforms, which both originate from a single time synchronized baseband in conjunction with an existing 10 MHz LTE signal. In this case, the LTE signal is a commercial over-the-air signal at 751 MHz. In an isolated test environment, the filter bank output transmits above and below the LTE signal without interfering. As the spectrum capture shows, given enough baseband bandwidth, the filter bank gives us the ability to operate with a great deal of flexibility on non-contiguous spectrum assignments.



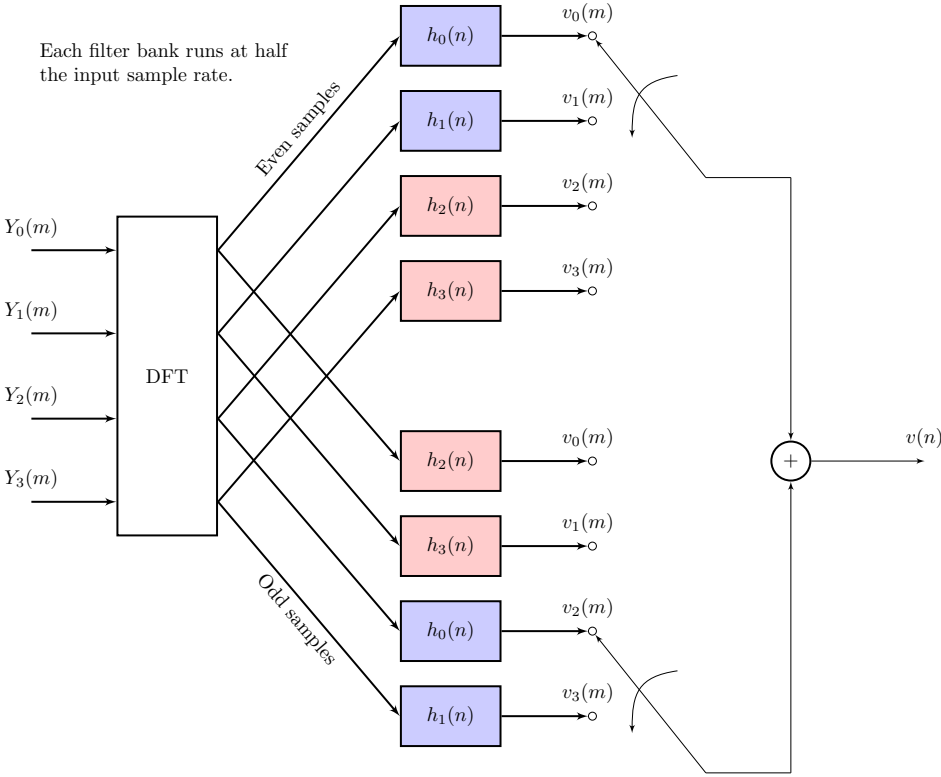


Figure 2.20: 2x oversampled synthesis filter implementation

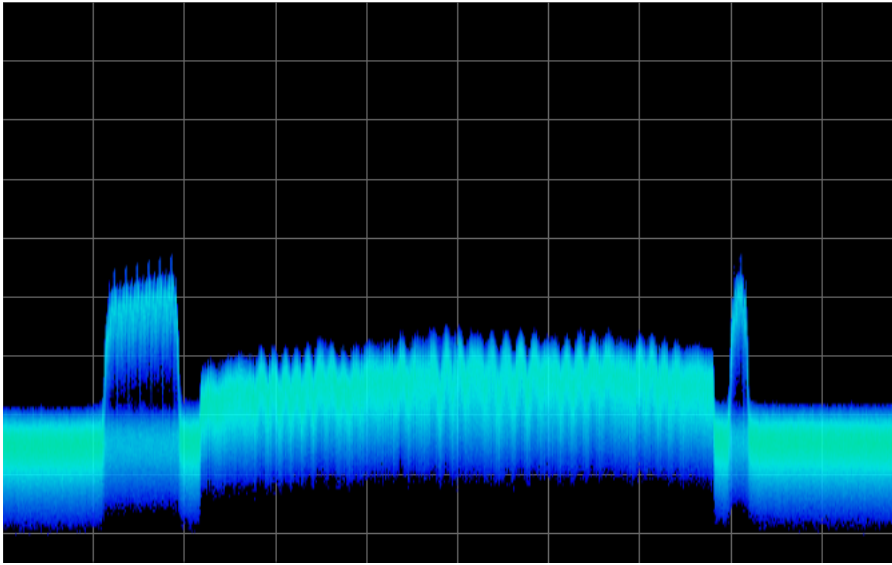


Figure 2.21: Non-contiguous spectrum usage with 10 MHz LTE signal between composite (left) and GSM (right) signals, 16 MHz span

# Chapter 3

## Improved Signal Integrity with Linearized GMSK

The GSM standard specifies Gaussian minimum-shift keying (GMSK) as the modulation scheme, which determines the metrics that are used for characterizing signal integrity and performance [8]. In a operational system, such as OpenBTS, there are many factors that affect transmit signal performance prior to reaching the power amplifier: DC offset, IQ imbalance, sampling, and phase noise among others. But, certain factors are hardware related and others are inherent to the design and implementation of the digital transmitter itself; the focus of this chapter is on the latter.

More specifically, this chapter traces signal integrity in the form of 3GPP specified phase error from the hardware output to the core design of the GMSK transmitter. For this process, definitive measurements are made with a GSM specific signal analyzer. We will show that the GSM requirements for GMSK signal quality are not particularly strict and that there is significant margin for improvement beyond minimum compliance values.

### 3.1 GSM Modulation

A GSM test signal generated by a commercial cellular signal generator (Agilent E4438C) is shown in Figure 3.1. The receiving and measuring device is a corresponding cellular signal analyzer (E4406A) of the same manufacturer. We note two defining characteristics of the GSM modulation. First, there is the very clear constant envelope and circular phase path. Second, the sampling points (in yellow) form small cluster of three distinct locations; the clustering is the result of intersymbol interference, ISI, of the partial response Gaussian pulse, which will be explained in following sections. Unsurprisingly for calibrated and certified commercial test equipment, measured phase error – the RMS and peak phase deviation compared to an ideal signal – is very low at below  $1^\circ$  and demonstrates an extremely accurate

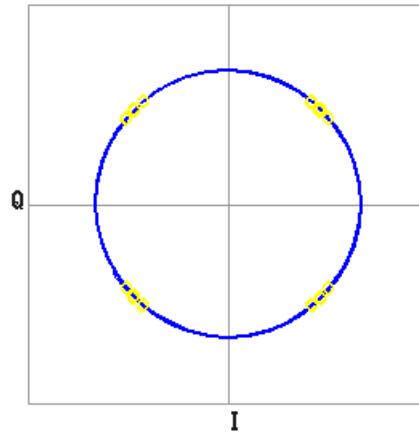


Figure 3.1: GSM reference signal, phase error  $0.29^\circ$  RMS,  $0.85^\circ$  peak

signal.

Figure 3.2 shows the default output from OpenBTS with USRP1 hardware. In contrast to our reference signal, the OpenBTS output shows non-constant amplitude and irregular phase trajectory. The measured phase error is much higher, but we also note that this signal is still compliant to the GSM specification. The specification 3GPP TS 45.005 [3] states,

The RMS phase error (difference between the phase error trajectory and its linear regression on the active part of the time slot) shall not be greater than 5 with a maximum peak deviation during the useful part of the burst less than 20.

The RMS measurement is very close to the limit, but the specification tolerates an additional 5 degrees of phase deviation in the peak value. From these observations, we can conclude, first, that the GSM specification is extremely tolerant with regards to phase error and, secondly, that there is substantial room for signal improvement in the OpenBTS transmitter. For the goals presented in this chapter, we seek to reduce the large, distressing gap between reference and observed OpenBTS signals.

## 3.2 Sampling Effects

The default configuration of OpenBTS runs the transceiver with sampling at 1 complex sample-per-symbol. This factor provides the most immediate source of distortion given that the sampling rate is below the Nyquist rate. The Nyquist–Shannon sampling theorem shows that a bandlimited signal can be perfectly reconstructed if sampled at a rate that exceeds twice the bandwidth of the signal. Consequently, default complex sampling at 1 sample-per-symbol – or 2 samples-per symbol when real sampling is considered – only accommodates up to a 270.833 kHz signal, which is insufficient for adequate reconstruction.

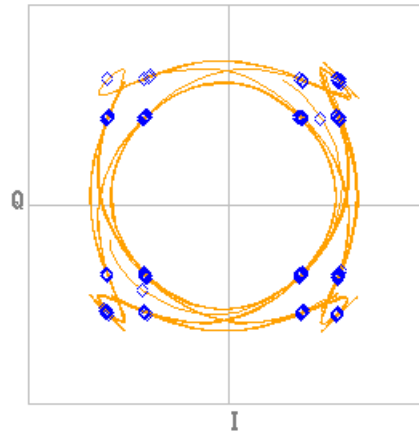


Figure 3.2: OpenBTS default configuration, phase error  $4.98^\circ$  RMS,  $14.95^\circ$  peak

Figures 3.3 and 3.4 show the spectrum of oversampled and undersampled OpenBTS signals respectively. Each capture shows 1 MHz of spectrum with 10 dB increments on the Y-axis. Note that in the oversampled spectrum the occupied bandwidth (roughly 300 kHz at 99% of total signal power) exceeds the available complex sampling bandwidth of the 1 sample-per-symbol case. This undersampling is further compounded by downstream filtering by the USRP hardware, which creates additional aliasing. Consequently, given the observable spectrum distortion, we can conclude that the signal shown in Figure 3.2 suffers from the detrimental effects of undersampling.

Therefore, increasing the sample rate is the first step to improving transmit signal quality; the same OpenBTS signal sampled at 4 samples-per-symbol is shown in Figure 3.5. Though 2 samples per symbol would be sufficient from a standalone DSP perspective, these examples are further oversampled to 4 samples-per-symbol for the purpose of reducing filter distortion generated on the USRP devices. The CIC (cascaded integrator-comb) filters located on the device apply a sinc shaped frequency response to the input signal, which we can reduce by oversampling.

Given sufficient sampling rate, we now observe a much cleaner phase trajectory and more accurate sampling points. Phase error is now well below the requirement for GSM compliance. Note, though, that there are remaining phase artifacts that were clearly not present in the reference signal of Figure 3.1. Resolving these phase path effects, which are not a result of sampling, requires a much deeper examination into the design of the OpenBTS modulator. The phase artifacts are a result of the fact the transmitter does not use an exact, direct form GMSK modulator, but instead uses a linearized approximation of GMSK. In order to further pursue the source of these effects, we need to understand the theory and representations of GMSK and the broader class of continuous phase modulations.

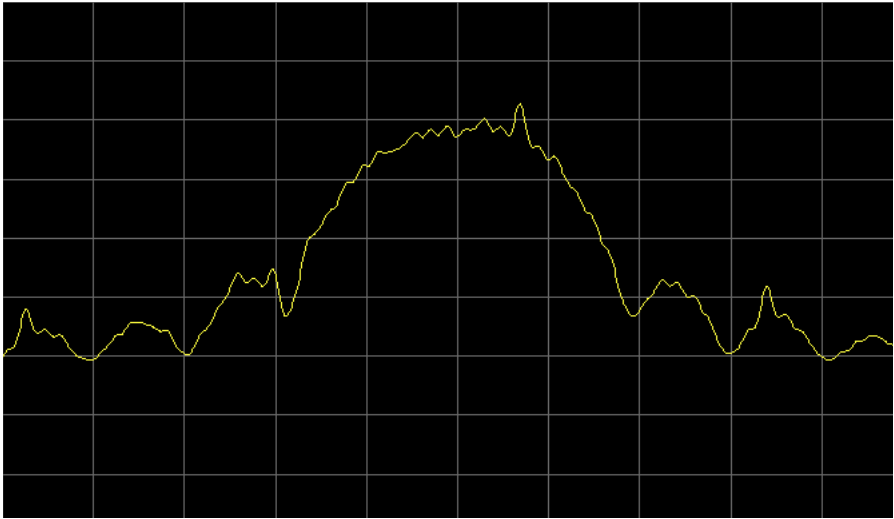


Figure 3.3: Oversampled OpenBTS spectrum with 8 samples-per-symbol, 1 MHz span

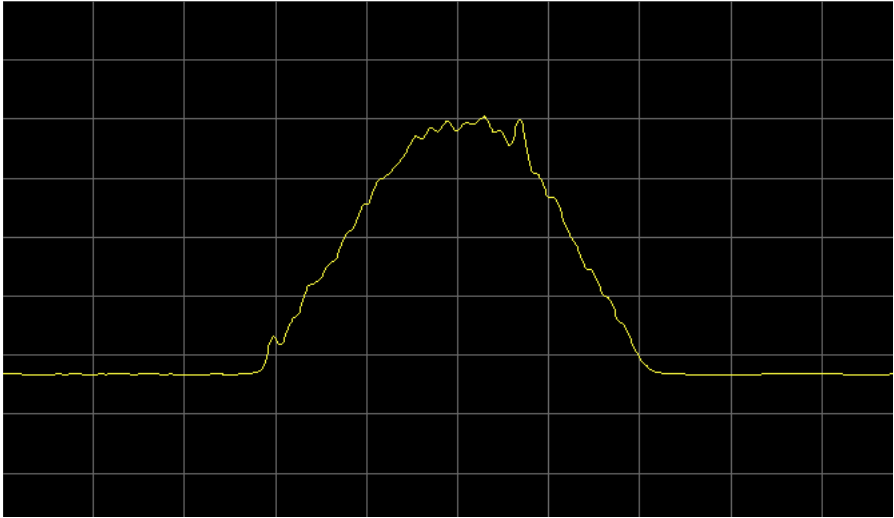


Figure 3.4: Undersampled OpenBTS spectrum with 1 samples-per-symbol, 1 MHz span

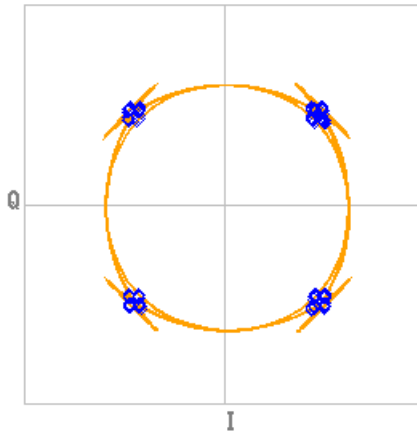


Figure 3.5: OpenBTS signal, 4 samples per symbol, phase error  $1.97^\circ$  RMS,  $5.08^\circ$  peak

### 3.3 Continuous Phase Modulation

Continuous Phase Modulation (CPM) belongs to a class of modulation techniques where the phase is constrained to be continuous [9]. This characteristic results in a non-linear system with memory. In the general case, amplitude need not be constrained, however, we assume that information symbols affect the phase only and do not consider the more general case of multi-amplitude CPM.

We begin with the following representation of the carrier modulate CPM signal

$$s(t) = \exp j [2\pi f_c t + \phi(t; I) + \phi_0]$$

where  $f_c$  is the modulated carrier frequency and  $\phi(t; I)$  represents the time varying carrier phase.  $\phi_0$  is the arbitrary initial phase of the carrier, which we will ignore from this point on. The time varying phase is defined by

$$\phi(t; I) = 2\pi \sum_{k=-\infty}^n I_k h_k q(t - kT), \quad nT \leq t \leq (n+1)T$$

where  $I_k$  is a sequence of information symbols and  $h_k$  is a sequence of modulating indices.  $q(t)$  is the normalized waveform shape, which is generally represented as the integral of some pulse shape.

$$q(t) = \int_0^t g(\tau) d\tau$$

For simplicity and applicability to GSM signals, we restrict the information sequence  $I_k$  to binary symbols  $|I_k| = 1$ . Furthermore, the modulation index can be fixed for all symbols at a value of  $h = \frac{1}{2}$ . The use of the latter yields a special form of CPM called Minimum Shift

Keying (MSK). We can rewrite the phase of MSK as

$$\begin{aligned}\phi(t; I) &= \frac{1}{2}\pi \sum_{k=-\infty}^{n-1} I_k + \pi I_n q(t - kT), \quad nT \leq t \leq (n+1)T \\ &= \theta_n + \frac{1}{2}\pi I_n \left( \frac{t - nT}{T} \right)\end{aligned}$$

Here  $\theta_n$  is the accumulation of previous phase shifts up to symbol  $n$ . Consequently, we have a representation where the current phase at time  $t$  is represented by summed previous phase shifts and a positive or negative  $\frac{\pi}{2}$  shift for the current symbol. From this point forward, we primarily consider the MSK case where  $h = \frac{1}{2}$ .

### 3.4 Laurent Decomposition of CPM Signals

As an alternate representation, CPM can be constructed as a linear combination of a finite sequence of pulses. This second approach provides an additional method that can be used in the modulation or demodulation process. We will discuss in later implementation sections as to why such an approach is desirable. But, for now, the linearized approach has the benefit of simpler and more convenient implementation and implications towards modulations used in more recent standards. Originally described by Pierre Laurent in 1986 [10], we now provide the linear representation of CPM and follow with a representation for 0.30 BT GMSK that is used for GSM.

We begin with the previous low-pass representation and ignoring the initial phase offset.

$$s(t) = \exp j[\phi(t; I)]$$

with the time varying phase described by

$$\phi(t; I) = 2\pi \sum_{k=-\infty}^n I_k h_k q(t - kT), \quad nT \leq t \leq (n+1)T$$

which we can use to rewrite the signal as

$$e^{\phi(t; I)} = \exp \left( j\pi h \sum_{k=-\infty}^{n-L} \right) \prod_{k=0}^{L-1} \exp [j2\pi h I_{n-k} q(t - (n-k)T)]$$

Laurent then introduces the *generalized phase pulse function* that is derived from the phase shift of the signal. This important function has non-zero values from  $0 \leq t \leq 2LT$  and is specified by

$$s_0(t) = \frac{\sin(\psi(t))}{\sin\pi h}$$

where

$$\begin{aligned}\psi(t) &= 2\pi h q(t) & t < LT \\ \psi(t) &= \pi h - 2\pi h q(t - LT) & LT \leq t\end{aligned}$$

This function serves as the basis for constructing the pulse series that will constitute the complete linear representation of the signal. The pulses  $c_k(t)$  for  $0 \leq k \leq 2^{L-1}$  are defined by

$$c_k(t) = s_0(t) \prod_{n=1}^{L-1} s_0(t + (n + La_{k,n})T), \quad 0 \leq t \leq T \min_n [L(2 - a_{k,n}) - n]$$

Each pulse is weighted by a complex coefficient where

$$\begin{aligned}A_{k,n} &= \sum_{m=-\infty}^n I_m - \sum_{m=1}^{L-1} I_{n-m} a_{k,m} \\ k &= \sum_{m=1}^{L-1} 2^{m-1} a_{k,m}, \quad k = 0, 1, 2, \dots, 2^{L-1} - 1 \\ a_{k,m} &= 0 \text{ or } 1\end{aligned}$$

Given the above phase pulses, substituting and reorganization of terms in the signal function yields the final result

$$e^{\phi(t;I)} = \sum_n \sum_{k=0}^{2^{L-1}-1} e^{j\pi h A_{k,n}} c_k(t - nT)$$

which is the CPM signal represented by finite sum of weighted pulses.

### 3.4.1 GMSK Laurent Decomposition for L=3

As an supplementary example to the mathematical representation, the effects of Laurent decomposition can be shown through graphical comparison. We examine the case for  $L = 3$ , which can be easily shown to be sufficient for GMSK with a 0.30 bandwidth time product. The first three generated pulses (named C0, C1, and C2) for  $BT = 0.30$  are shown in Figure 3.6

From the preceding analysis, we can construct our CPM signal with a finite number of pulses, but Figure 3.6 shows that the practical number of pulses may be an even smaller number than the mathematical representation specifies. In this case the plot shows that only the first two pulses are of any significant magnitude, while the third pulse, C2, is barely visible. Furthermore, the primary pulse, C0, is overwhelmingly dominant. In fact, the C0 pulse alone is sufficient for a linearized approximation of the GMSK modulation, which is used by



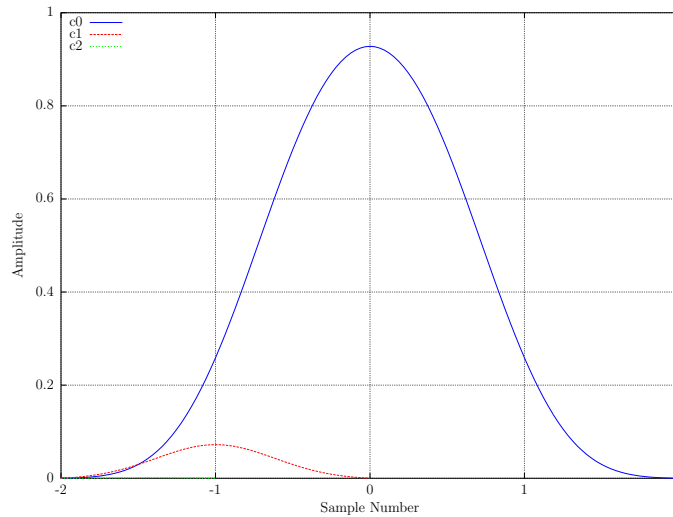


Figure 3.6: Laurent C0, C1, and C2 pulses for  $BT = 0.30$

OpenBTS and measured in Figure 3.5. But, as we observed, there are remaining artifacts due to the missing C1 pulse, which, though small, are not insignificant.

For comparison purposes, we can also examine the generated Laurent pulses for varying widths of the original partial response Gaussian shape. Figures 3.7 and 3.8 show the computed pulses for  $BT = 0.20$  and  $BT = 0.40$  respectively. In all cases we find negligible significance of pulses beyond C0 and C1. We also find an increase in significance of the C1 pulse as the  $BT$  product, and ISI, decreases. This is expected since as the  $BT$  product approaches infinity, the partial response GMSK signal becomes a normal full response MSK signal, which can be fully represented by linear representation equivalent to offset-QPSK with a half-sinusoidal pulse shape.

## 3.5 Implementation

Given the linearized representation of CPM, we can now proceed to the actual software implementation. In this section, we directly translate the analytical work of Laurent into a fully operational real time GMSK modulator. The initial OpenBTS design, which uses single C0 pulse implementation is shown in Figure 3.9. The implementation consists of an input bit sequence mapped to either a positive or negative phase shift of  $\pi h = \frac{\pi}{2}$ . The phase shift is then accumulated and sent to the pulse shaping filter. Note that GSM employs differential encoding, which is not shown.

In order to add the second C1 pulse, Laurent's decomposition translates to the structure of Figure 3.10. While there is an expanded structure to accommodate the effect of one delayed signal, the operations for MSK,  $h = \frac{1}{2}$ , are simply bit inversions in the in-phase or quadrature

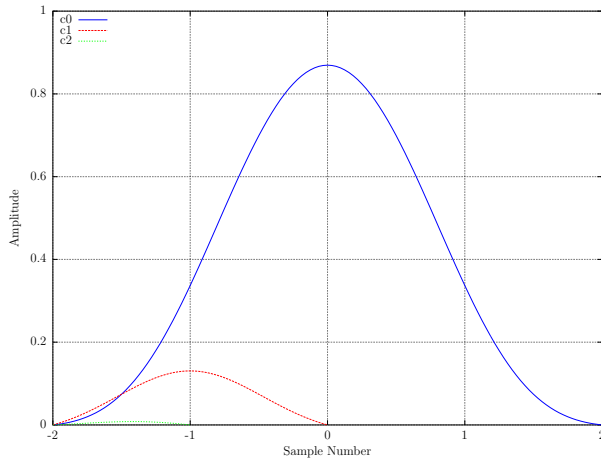


Figure 3.7: Laurent C0, C1, and C2 pulses for  $BT = 0.20$

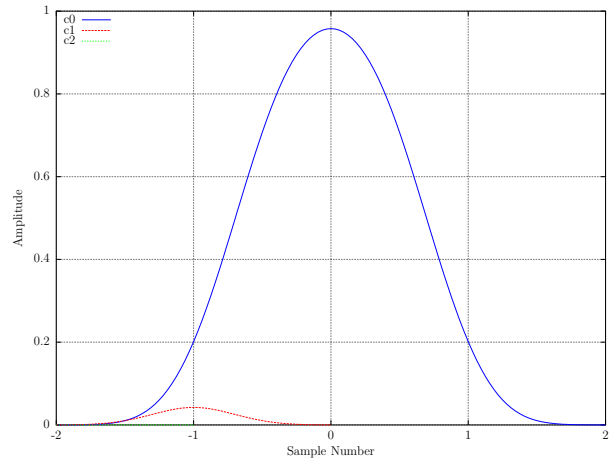


Figure 3.8: Laurent C0, C1, and C2 pulses for  $BT = 0.40$

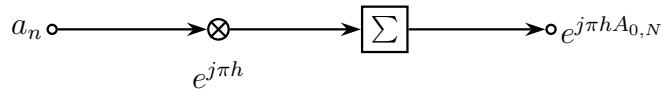


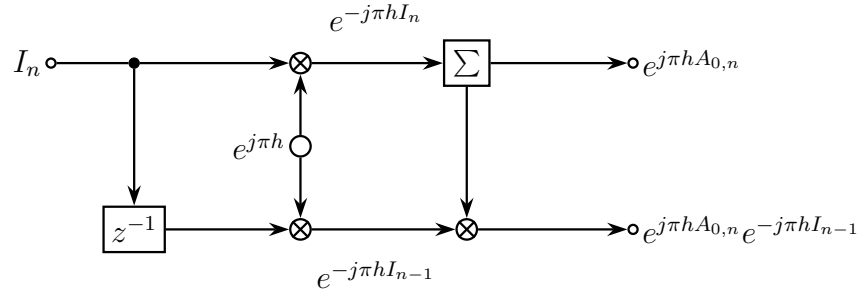
Figure 3.9: Linearized GMSK symbol mapper with single pulse

channels. Filtering and summing with the C0 and C1 pulse shapes can be implemented with a structure that resembles two conjoined tapped delay lines as shown in Figure 3.11. The tap lengths in this case represent computed pulse filters at 4 samples-per-symbol.

### 3.6 Results and Discussion

With the advantages of the linearized approach, our pulse simulations showed that we can easily ignore all pulses except for C0 and C1. That led us to the construction given in Figures 3.11 and 3.10. We also know that we can only use the C0 pulse for a very simple implementation, but then suffer from the phase trajectory artifacts shown at the beginning of this chapter. The USRP1 output of our final construction is shown in Figure 3.14 alongside the original single pulse implementation at 1 and 4 samples per symbol in Figure 3.12 and Figure 3.13 respectively.

Given the modified GMSK modulator of Figure 3.14, we can clearly observe the absence of phase artifacts present in the original modulator, Figure 3.13. Obviously, both cases show substantial improvements from the sub-Nyquist samples case of Figure 3.2. While modifying the OpenBTS modulator shows clearly visible improvements in the magnitude error near the sampling points, the improvement in measured phase error is quite modest. To find out why,



$$s(t) = \sum_{n=-\infty}^{\infty} e^{j\pi h A_{0,n}} [c_0(t - nT) + e^{-j\pi h I_{n-1}} c_1(t - nT)]$$

$$A_{0,n} = \sum_{m=-\infty}^n I_m$$

$$A_{1,n} = A_{0,n} - I_{n-1}$$

$$I_n = 1, -1$$

Figure 3.10: Linearized GMSK symbol mapper with C0 and C1 pulses

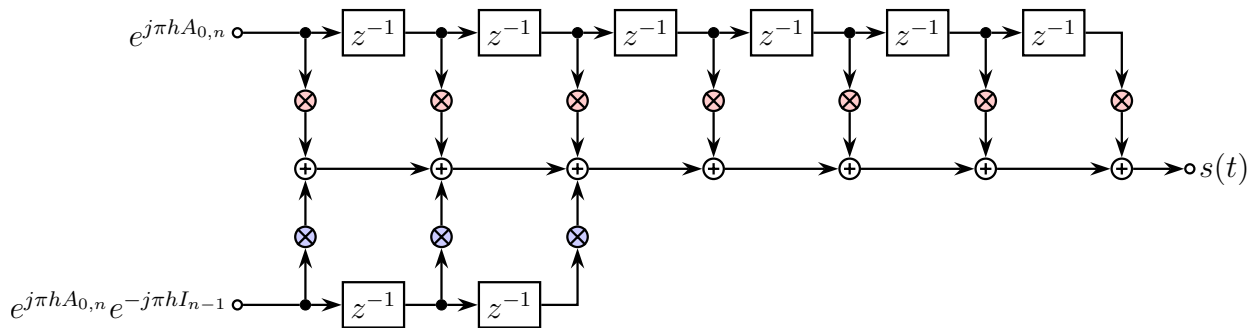


Figure 3.11: Combined tapped delay line for linearized GMSK pulses C0 and C1

Table 3.1: Measured Phase Error

	Samples-per-symbol	RMS	Peak
3GPP TS 45.005	N/A	5.00	20.00
USRP1, C0	1	4.89	14.25
USRP1, C0	4	1.92	5.44
USRP1, C0 and C1	4	1.66	4.53
USRP2, C0 and C1	4	1.14	2.97

we can measure the same modulator implementation through an USRP2, which is displayed in Figure 3.15.

The USRP2 shows superior phase accuracy – recall that the GSM compliance phase error is  $5^\circ$  RMS and  $20^\circ$  peak. The summary of various modulator and hardware combinations discussed in this chapter are shown in Table 3.1. In the USRP2 case, the primary difference between USRP1 and USRP2 measurements are due to the availability of DC offset and IQ imbalance corrections on the USRP2. Both combinations use the same WBX daughterboard from Ettus Research, but only the USRP2 supports automatic daughterboard calibration. Consequently, we can conclude that, to a large extent, we are no longer primarily observing GMSK modulator irregularities, but the effects of uncompensated RF hardware differences.

The linear modulation approach described in this chapter, at first glance, seems unnecessary and cumbersome given that a basic GMSK implementation with a pulse shaped bit sequence driving a phase accumulator is not particularly complex. There are, however, three significant advantages of the linearized approach. First, the modulator can be implemented using only standard multiply-accumulate operations. This contrasts to the direct implementation of MSK that drives a frequency modulator using trigonometric methods, which may be difficult to implement or optimize depending on the processor architecture. Second, while GMSK modulation is constant amplitude, the TDMA based GSM signal is not. Time slots in GSM are separated by guard intervals with specified amplitude ramp-up and ramp-down requirements [2]. Consequently, the implementation does need to consider amplitude changes, and the linear approach can provides a simpler implementation. Finally, there are strong implications for the linearized approach because of developments in later specifications. For example, the linearized Gaussian pulse shape is explicitly specified for use in 8-PSK EDGE [8]. Furthermore, the same pulse shape is used again with a modified QPSK modulation in the more recent Release 9 specification for orthogonal subchannels and voice, VAMOS [4]. Given these concerns, the linearized approach to GMSK turns out to be very reasonable for GSM targeted implementations.

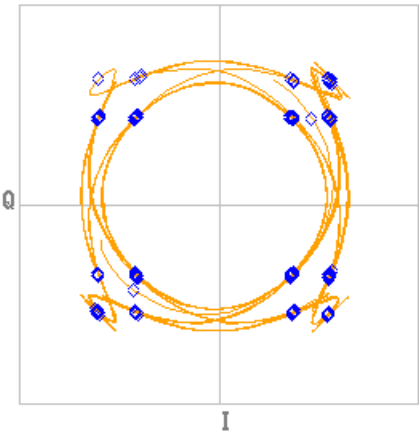


Figure 3.12: OpenBTS–USRP1 phase error, 1 sample per symbol, C0 pulse, 4.98° RMS, 14.95° peak

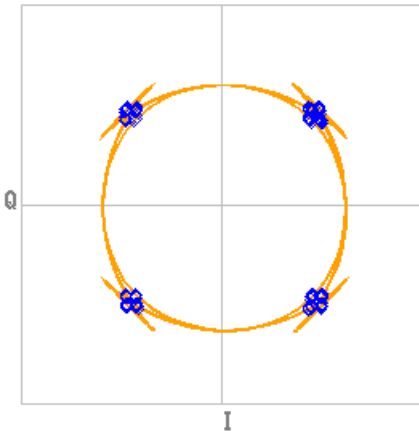


Figure 3.13: OpenBTS–USRP1 phase error, 4 samples per symbol, C0 pulse, 1.92° RMS, 5.44° peak

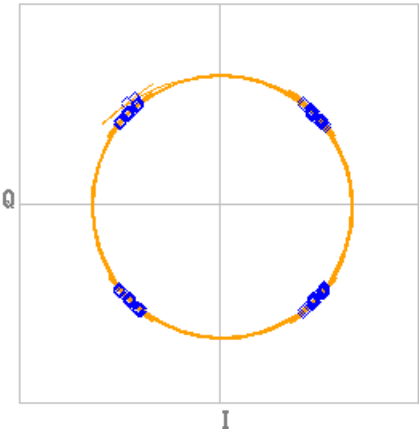


Figure 3.14: OpenBTS–USRP1 phase error, 4 samples per symbol, C0 and C1 pulses, 1.66° RMS, 4.53° peak

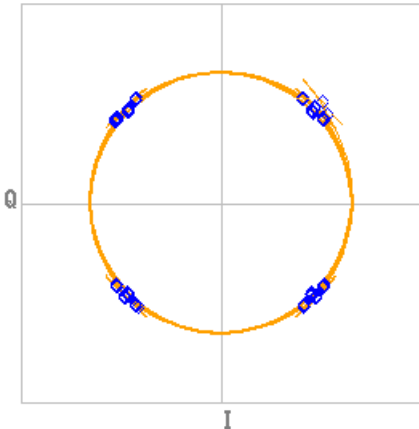


Figure 3.15: OpenBTS–USRP2 phase error, 4 samples per symbol, C0 and C1 pulses, 1.14° RMS, 2.97° peak

# Chapter 4

## Embedded BTS

Chapter 2 explored the task of maximizing OpenBTS user capacity with a wideband multi-carrier implementation. In Chapter 3, we increased OpenBTS signal integrity substantially beyond requirements set forth by 3GPP specifications. In this chapter, rather than pushing any upper limits, we attempt to reduce certain properties by running OpenBTS on small, embedded form factor.

The overall system assumption of previous chapters consisted of a comparatively powerful x86 host machine interfacing with a USRP radio device. Now, we consider the case of much smaller, and less power hungry, processor combinations. Specifically, an ARM application processor and DSP processor from Texas Instruments will be compared. We will introduce and discuss the E100 hardware platform from Ettus Research that encompasses these processors and follow with comparative results based on filtering metrics similar to those presented in Chapter 2. Our final result will evaluate single carrier OpenBTS operation on the embedded platform with a NEON optimized and fixed point DSP transceiver implementation.

### 4.1 Hardware Platform Description

The E100 platform is a USRP variant that includes three reconfigurable processors suitable for different aspects of software radio and cellular use: Xilinx Spartan-3A FPGA, ARM CortexA8, and Texas Instruments fixed point C64x+ DSP. The general purpose ARM processor (GPP) and C64x+, as components of the TI OMAP3530 application processor, are collocated on a removable Gumstix computer-on-module board [11]. The FPGA is mounted directly on the main board and interfaces with the other processors through a combination of General Purpose Input / Output (GPIO) pins and the OMAP3530 General Purpose Memory Controller (GPMC). The overall processor structure and connections are shown in Figure 4.1. Since we are primarily interested in signal processing and optimized computation, the remainder of this chapter focuses on the ARM and DSP elements.

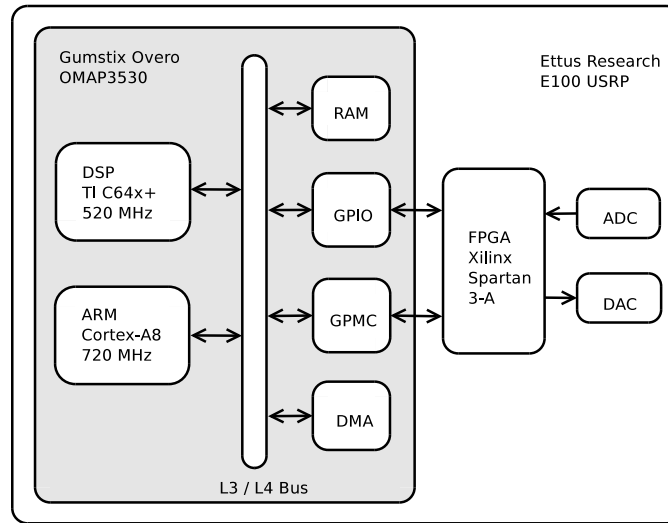


Figure 4.1: E100 multi-processor platform

### 4.1.1 Interprocessor Communication

Distribution of base station functionality across the set of multiple, heterogeneous processors greatly improves efficiency and increases the capabilities of the device, but introduces the added task of managing interprocessor communication across each of the different cores. Because of disparate physical locations, interconnects, and the unique characteristics of each processor, no unified software architecture for interprocessor communication exists for the E100 configuration. Rather, interaction between the specialized DSP and FPGA cores and the GPP occurs through separate, independently implemented software interfaces. Also, no direct FPGA–DSP line of interaction is implemented. Consequently, all communication flows through the ARM processor. While less optimal than a directly mapped FPGA–DSP interface, this approach results in a far less complex implementation and significantly reduced development time.

ARM to C64x+ transfers are implemented using TI DSP/BIOS Link (DSPLINK), which provides an interface and abstraction layer for a shared memory and hardware interrupt based transport. The overall GPP–DSP message queue transport is shown in Figure 4.2. During initialization, asynchronous message buffers are allocated from a shared memory pool. Through DSPLINK, OpenBTS uses message queues for transferring shared memory pointers to and from the DSP; these memory pointers reference buffers containing the bursts received from the FPGA. Either side of the connection can access the shared buffer contents by translating the pointer address between the ARM and DSP virtual address spaces. The data in memory is then read or written by explicit cache invalidations or cache writes respectively.

As a TDMA based standard, reception and transmission of GSM signals is heavily dependent on segmentation of continuous data streams into independent, synchronized time slots, or

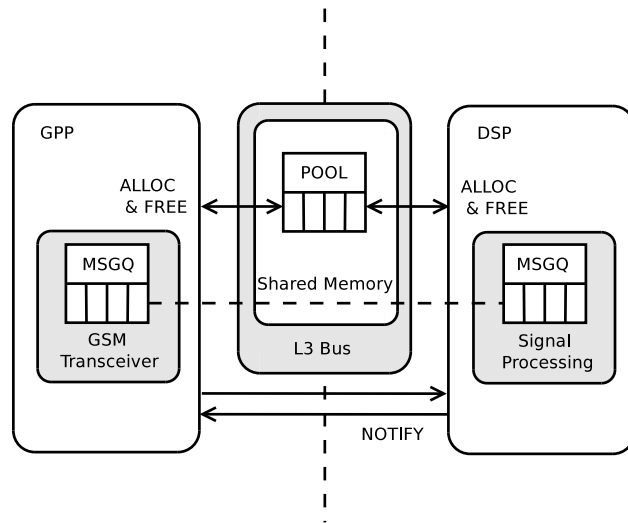


Figure 4.2: C64x shared memory transport

bursts. Consequently, due to time slot use, the message and queuing mechanism facilitates an efficient and fitting abstraction for communicating sampled GSM data across the GPP–DSP barrier.

## 4.2 GSM Transceiver

The embedded BTS implementation partitions base station functionality across multiple processors in order to support different aspects of GSM operation in an efficient manner. The precise timing necessary to meet the strict, real-time requirement of GSM specific TDMA access is managed through coordinated use of the ARM and FPGA cores. The FPGA configuration is mandatory since no other processor on the device has the necessary hard timing capability. The task of partitioning functionality between the ARM and C64x DSP cores is somewhat less clear. As an embedded application processor, the ARM is heavily constrained compared to the larger x86 based systems. Therefore, maximizing use of the DSP is the preferred approach, but, frequent data transactions between the two cores incur transport and signalling penalties that must be considered.

One approach to transceiver partitioning is to split transmit and receive chains across different processors. With this approach, there are no dependencies as each signal path is separately handled. In addition, the previous chapter demonstrated that the GSM modulator can be constructed with a simplified linearized GMSK implementation; we showed that the transmitter design consists of a symbol mapper and modified pulse shaping filter. In contrast, the OpenBTS receiver structure, which we have not yet examined, must handle more complicated tasks of burst detection, timing recovery, and demodulation. As a re-



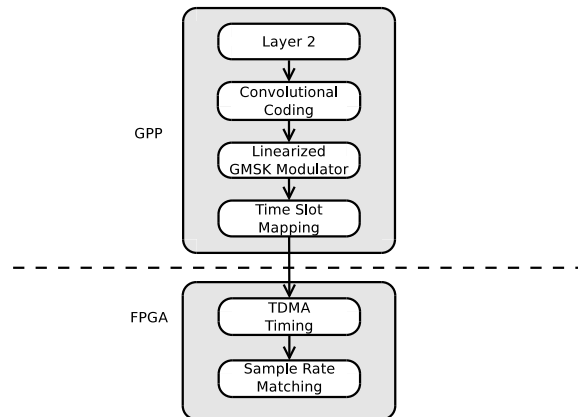


Figure 4.3: OpenBTS L1/PHY transmit chain

sult of these differences, allocating the receiver chain to the DSP and performing transmit modulation on the ARM creates an appropriate partitioning.

### 4.2.1 Transmitter Structure

The linearized GMSK (LGMSK) modulator from Chapter 3 can be implemented through filtering and therefore the successive use of multiply-accumulate operations. Consequently, we can modify the implementation with SIMD approaches not unlike those performed for the polyphase multicarrier implementation from Chapter 2. These changes will be described in the later SIMD section.

Figure 4.3 shows the more general transmit chain beyond GMSK modulator. Within the transmit chain, the overall burden of the modulation process is quite small. Note that in OpenBTS, forward error correction is implemented separately from the transceiver program, so we do not consider convolutional coding or decoding processes in this thesis.

### 4.2.2 Receiver Structure

The OpenBTS receiver chain is shown in Figure 4.4 and encompasses all portions of the GSM physical layer. Our focus in this section is primarily on *porting* the GSM receiver to the C64x DSP. Since we maintain the high-level receiver design, thorough discussion of the design itself is beyond the scope of this thesis. The essential elements, however, will be briefly introduced. As with the transmitter, we note that forward error correction is considered separately from the receiver.

Given the discontinuous nature of GSM TDMA communications – there is no continuity in sample timing or phase between bursts – timing and synchronization are heavily based on

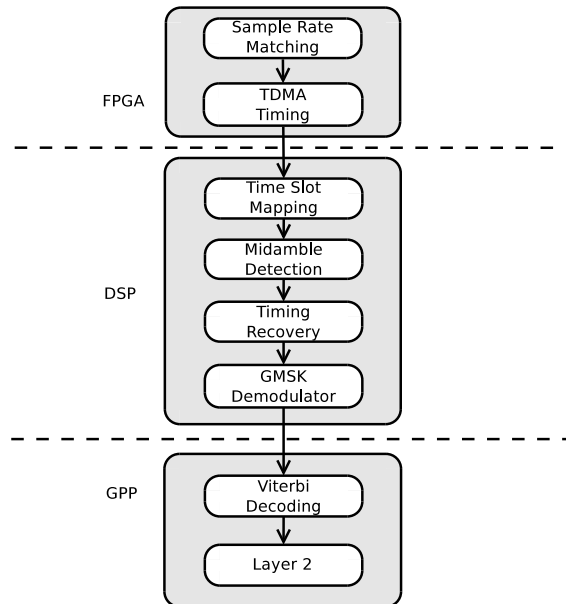


Figure 4.4: OpenBTS L1/PHY receive chain

the use of embedded training sequences [12]. These training sequences exist as midambles centered within the GSM burst structure. As Figure 4.4 shows, the output of the midamble detection, a correlation operation, is used for timing recovery, which consists of fractional delay filtering, and phase correction. While we do not discuss these operations in further detail, the critical characteristic of all these processes is that they can be implemented using some form of polyphase filter bank or convolution method.

For example, midamble detection is implemented through cross correlation, which is analogous to convolution. Also, fractional delay filtering is an interpolation process that is efficiently implemented with a polyphase filter bank. Consequently, the task of implementation and optimization of the receiver is highly reflective of the previous approaches and analysis provided in Chapter 2.

## 4.3 Implementation

### 4.3.1 ARM SIMD

In comparing ARM to the x86 SIMD architecture presented in Chapter 2, there are very close similarities such as the use of quad-word operations and a register bank of 16 128-bit registers [13]. Again, OpenBTS uses single precision 32-bit floating point values. There are, however, significant variations that can be traced to fundamental differences in the x86 and ARM architectures. The ARM processor is a load/store architecture in that data stored in

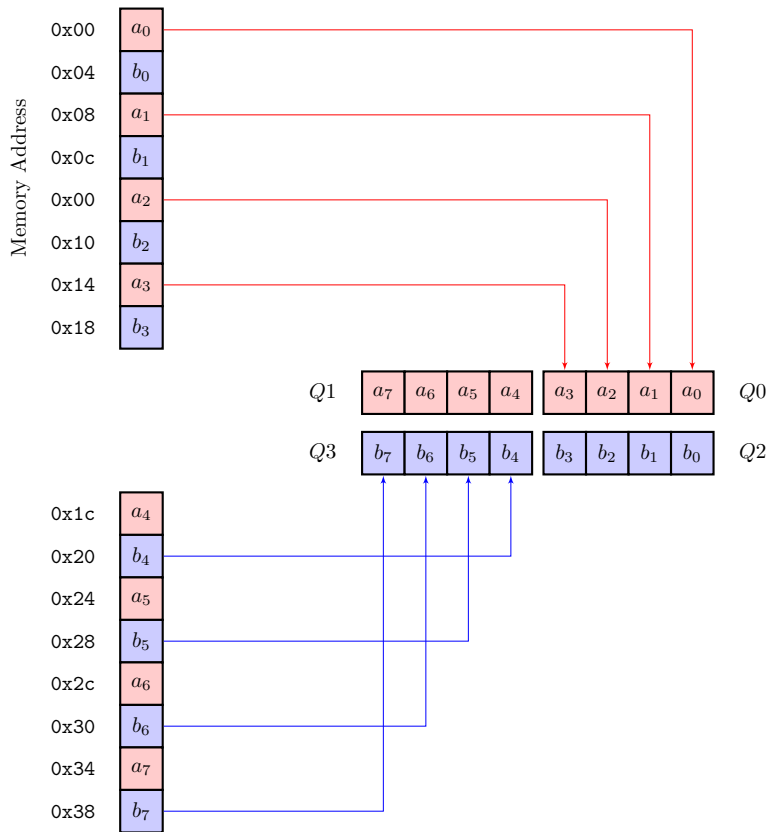


Figure 4.5: NEON quad-register 8 element multi-lane load for deinterleaving I and Q data

memory is never accessed directly, but always imported and exported to and from registers with explicit load and store instructions. This approach contrasts with the x86 architecture that allows instructions to directly or indirectly access memory values without an explicit load/store instruction. Given this characteristic, the ARM instruction set provides some additional flexibility with loading stored sampled data.

### Deinterleaving Register Load

One advantage of the ARM instruction set is that register loads can simultaneously deinterleave from memory. This feature is beneficial for software-defined radio because samples are commonly stored in interleaved I and Q format. For example, the USRP, GNU Radio, and OpenBTS all follow this convention of storing sampled data in alternating I and Q samples. Figure 4.5 demonstrates this deinterleaving capability with an 8-element complex load sequence. In contrast, to perform the equivalent operation, the x86 architecture requires either independent, separate loads, or a linear load followed by in-register shuffling of data.

## NEON Multiply–Accumulate

Unique to the NEON instruction set is the availability of dedicated multiply–accumulate instructions. The `VMLA` instruction performs a 4-element floating point multiply–accumulate, which, upon first glance, appears to be highly advantageous for use in SDR signal processing. With further examination, however, this instruction is not without significant performance limitations. Figure 4.6 illustrates a successive complex multiply–accumulate as would be performed for a FIR filter or other convolution based operation.

The limitation with the ARM `VMLA` instruction is due to idle delays in instruction pipelining of the ARM Cortex–A8 architecture. More specifically, Cortex–A8 uses a 13 stage pipeline with no out-of-order execution of instructions [14]. The critical limitation arises when two `VMLA` instructions are used back-to-back – as would be performed for SDR filtering purposes. This condition causes a read-after-write (RAW) hazard, which leads to a pipeline stall of multiple cycles. The duration of the stall depends on the subsequent instructions. For `VMLA` the effect is a rather significant 8 instruction cycles. Given that out-of-order execution is not supported, this behavior effectively reduces the advantages of hardware instruction pipelining when the `VMLA` instruction is used.

For comparison, the instruction path shown in Figure 4.7, which is very similar to the previous x86 SSE case, does not have any dependencies through the first sequence of multiply operations. Consequently, this sequence, which does not utilize the hardware multiply–accumulate instruction, computes the same function with more instructions, yet completes faster by a notable margin.

### 4.3.2 Texas Instruments C64x+ DSP

In comparison to previously discussed ARM and x86 processors, the C64x DSP is unique in that it is not used as a standalone processor. Rather than running a general purpose multitasking operating system like Linux, the DSP runs a true real time operating system (RTOS) from Texas Instrument called DSP/BIOS. Then there is the difference in that the C64x is a fixed point processor where any floating point operations are performed strictly through software emulation.

As one would expect, the architecture differences of the specialized DSP processor are also substantial. Figure 4.8 shows the basic architecture of a C6000 series DSP of which the C64x is a member. Like the ARM processor, the DSP is a load/store architecture in that all memory accesses are performed by explicit load or store instructions. Unlike the ARM – or more specifically the Cortex–A8 – the C64x is a pipelined architecture with the following functional units that can execute in parallel with no interdependencies.

- `.D` (`.D1` and `.D2`) data load and store

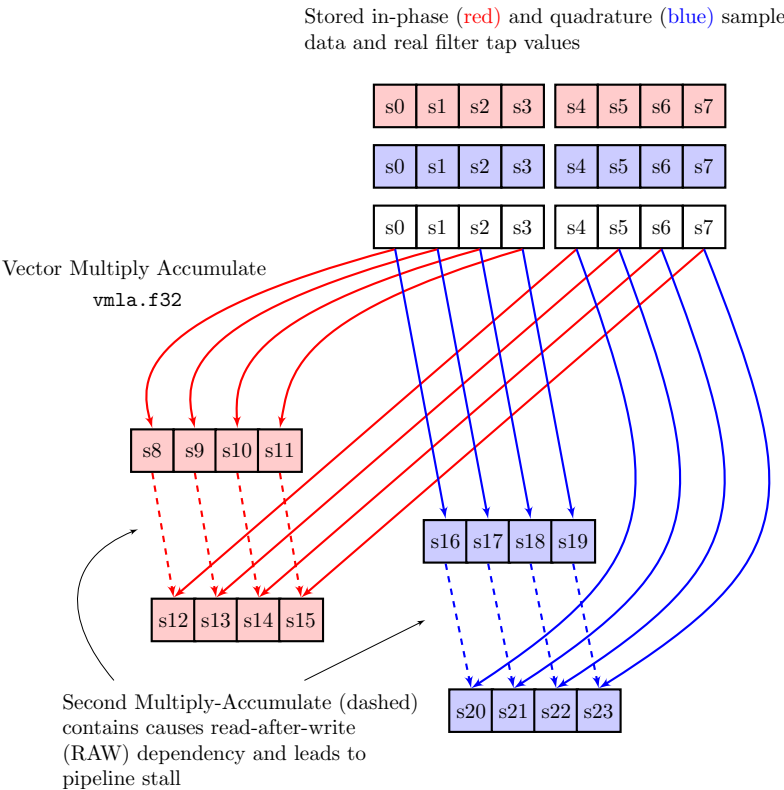


Figure 4.6: NEON multiply-accumulate instruction VMLA causes read after write (RAW) hazard

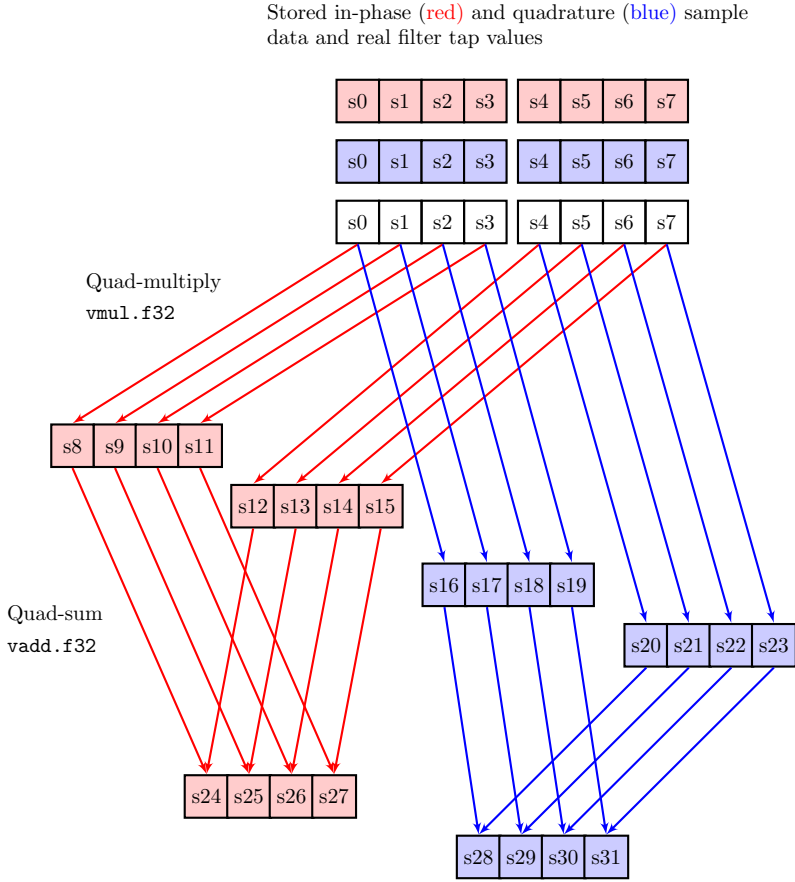


Figure 4.7: NEON 8 tap convolution product-sum with pipeline compatible operations (only first sum is shown)

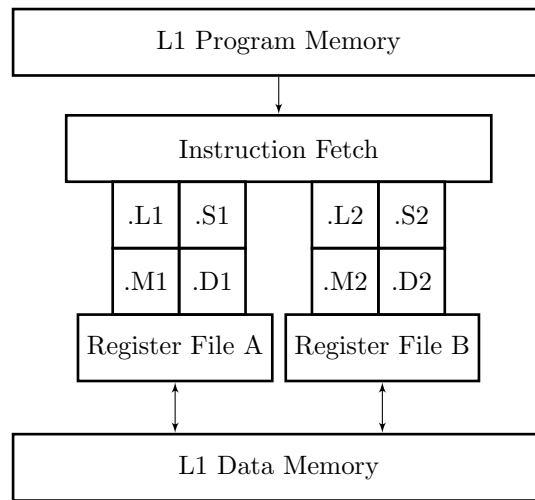


Figure 4.8: Texas Instruments C64x DSP

- .S (.S1 and .S2) shift, branch, or compare
- .M (.M1 and .M2) multiply
- .L (.L1 and .L2) logic and arithmetic

Practically speaking, we see significant differences in the handling of iterative multiply–accumulate operations. Whereas the VMLA operation would essentially seize the Cortex–A8 pipeline, the DSP is capable of processing two multiply-and-accumulate operations in a single cycle. Now, to be fair, we use 32-bit floating point values on the ARM and 16 and 32 bit fixed point values on the DSP, so architectural comparison is an inexact process. Illustrating these differences does, however, provide insight into the appropriate methods to approaching signal processing tasks.

For implementation and code, the DSP is also unique in the tools and optimization process. Unlike the ARM where we relied on manual optimization with NEON instructions, DSP development is heavily dependent on the tool chain, compiler, and static analysis with less emphasis low level operations. This concept is easily illustrated with the following code example and compiler output.

```

#pragma MUST_ITERATE(4, 16, 4)
for (i = 0; i < count; i++) {
    prod = a[i] * b[i];
    sum += prod;
}

*-----*
*   SOFTWARE PIPELINE INFORMATION
*
*   Loop Unroll Multiple           : 2x
*   Known Minimum Trip Count      : 2
*   Known Maximum Trip Count      : 8
*   Known Max Trip Count Factor   : 2
*   Loop Carried Dependency Bound(^) : 0
*   Unpartitioned Resource Bound  : 1
*   Partitioned Resource Bound(*) : 2
*   Resource Partition:
*
*           A-side   B-side
*   .L units           0     0
*   .S units           0     1
*   .D units           1     1
*   .M units           1     0
*
*   Total cycles (est.) : 9 + trip_cnt * 2 = Between 13 and 25
*-----*

```

The code example consists of an iterative multiply and accumulate. A `pragma` directive is also inserted to signify that iteration values range from 4 to 16 with multiples of 4. The optimizing C6000 compiler output provides us with notable characteristics about the generated code such as unrolling and an estimate of the total cycle count. These cycle counts will later provide a point of comparison for evaluating DSP transport and overhead.

## 4.4 Results and Discussion

### NEON Benchmarks

Benchmarking the different embedded approaches took the same form as the SIMD tests used in Chapter 2. All measurements show averaged completion times for 10,000 convolutions on random GSM burst data. Bursts are 156 symbols and sampled at 2, 4, and 8 samples-per-symbol with varying



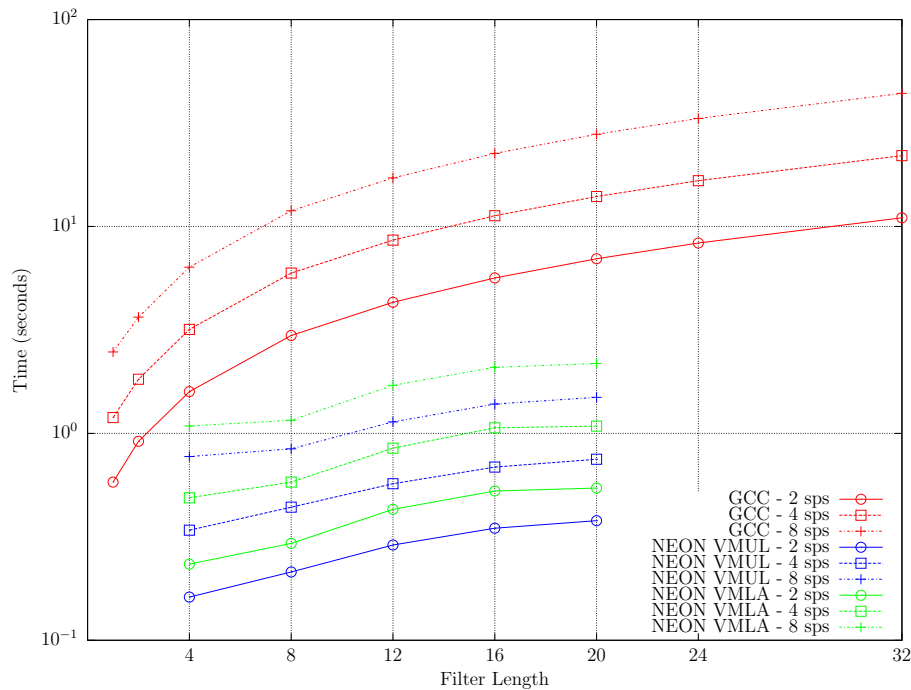
Figure 4.9: NEON `vmul/vsum` vs. `vmla` comparison

Figure 4.9 shows the comparison between using the GCC compiled C code output, multiply–accumulate instruction (VMLA), and pipeline optimized NEON approach. The comparison shows large differences – an order of magnitude – between both versions of NEON code and the compiled C code. The SIMD speedups are significantly larger than those seen on x86 and also much higher than a factor of 4 that may be expected from quad-vector operations. This disparity is mostly attributed to GCC, which is very limited when generating NEON code as discovered by examination of the generated assembly.

Within the two NEON implementations, the differences are more modest, but clearly beneficial. We conclude that for our application with OpenBTS, the existence of the dedicated multiply–accumulate operation in the ARM instruction set provides little to no benefit. Furthermore, the use of NEON optimization is essential due to poor compiled C code performance.

## DSP Benchmarks

Comparing DSP output is more complicated than NEON assembly because of the added overhead of interprocessor communication. Figure 4.10 shows completion times compared to static analysis, or theoretical values. A constant DSP message size of 32 kB, equal to the maximum size of the L1 data cache, was used for all tests and implementations. There are two notable characteristics in this comparison related to interprocessor communication

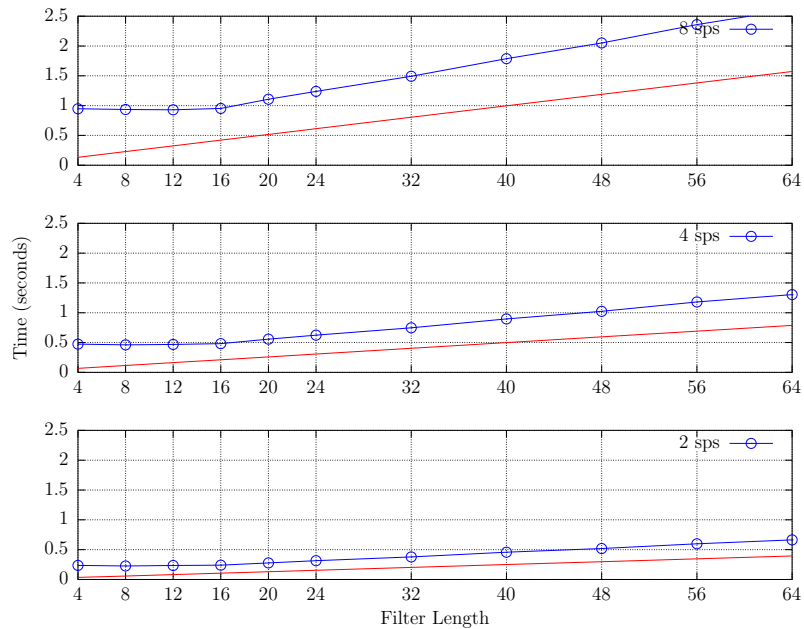


Figure 4.10: C64x+ DSP complex convolution (blue) vs. static analysis (red)

and message size. First, the gap between theoretical and observed values widens as we increase the samples-per-symbol (and therefore the burst size). This is a result of using a fixed message size, which results in fewer GSM bursts per DSP transport message as the samples-per-symbol is increased. The second characteristic is that elapsed time is flat for smaller filter lengths. In this case, we are seeing the effect of interprocessor communication dominating the measurements. From these observations, the overall conclusion is that the target DSP operation must be sufficiently complex in order to take advantage of the architecture. Otherwise, the efficiency becomes lost due to transport and communication overhead.

Figure 4.11 and Figure 4.12 show the observed DSP completion times against the raw GCC and optimized NEON instructions. For smaller tap lengths (and reduced computation), we can see that the DSP overhead nullifies any benefits and performs worse than similarly optimized NEON. Given our previous conclusions about DSP use, it follows that the more complicated and intensive the signal processing task, the more benefit the DSP processor will yield to the overall system.

## Runtime Testing

Given the above benchmarks, large improvements are expected when moving to from a benchmark test application to the OpenBTS transceiver implementation. For baseline per-

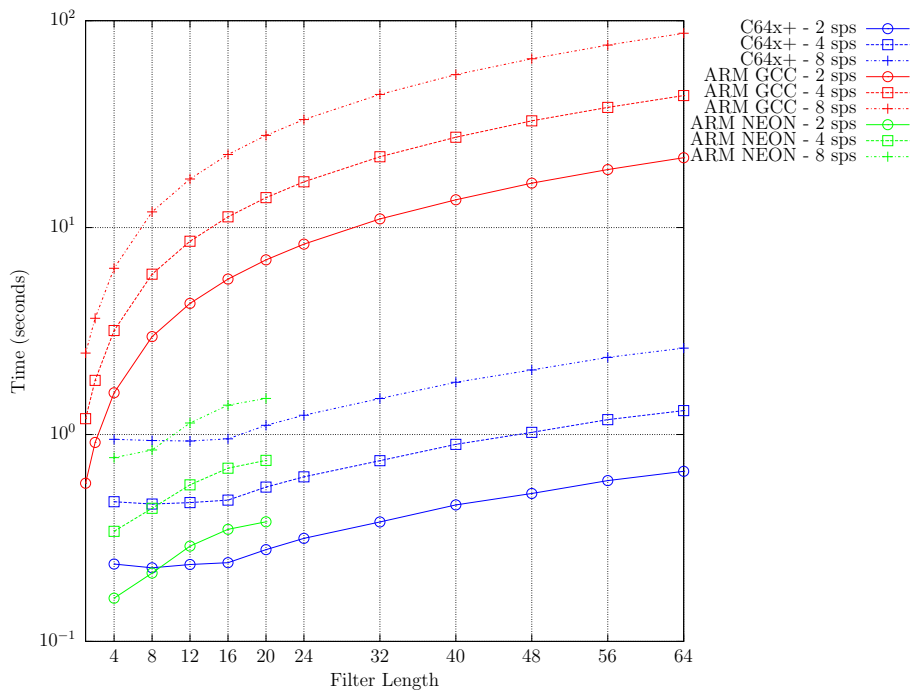


Figure 4.11: C64x+ DSP complex-real convolution vs. ARM

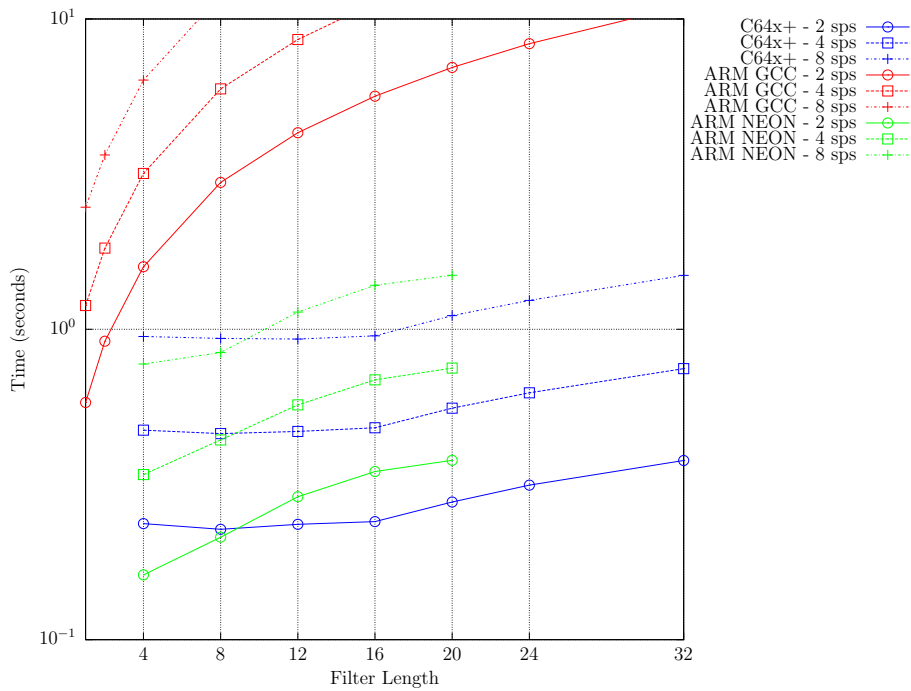


Figure 4.12: C64x+ DSP complex-real convolution vs. ARM

Table 4.1: Number of Supported TCH Channels on E100

	TCH Channels
ARM GCC	2
ARM NEON	3
ARM GCC & DSP	4
ARM NEON & DSP	5

formance, OpenBTS does run on the E100 platform with no optimization beyond GCC. This configuration is quickly limited to two simultaneous full rate TCH (voice) channels. Recall from Chapter 2 that a GSM carrier supports 8 physical channel; with one channel dedicated to control purposes, 7 TCH channels is the target value. NEON and DSP optimizations each yield incremental improvements with the final combination supporting up to 5 voice calls – two short of our target value.

Further examination reveals that the optimized implementation is not particularly limited by signal processing, but also by other factors not considered in this thesis. For example, we previously mentioned that convolutional decoding was not considered part of the radio transceiver and was excluded from all optimization efforts. The Viterbi implementation used for decoding does indeed consume a high amount of ARM processing, which increases with the number of TCH channels. In addition, we considered the overhead of DSP interprocessor communication, but we did not consider the overhead of communication within OpenBTS. For example, the performance implications of I/O mechanisms such as sockets and certain thread safe queues remains to be evaluated.

In summary, the approach of partitioning transceiver functionality across the heterogeneous processors yielded significant improvements in embedded small cell efficiency and overall capacity – in unit tested and system level cases. But, additional development is necessary to achieve our target of a fully utilized single carrier femtocell. To achieve this goal, future software development needs to investigate non-transceiver areas of the code base, such as Viterbi decoding and I/O performance, in addition to signal processing concepts such as convolution and filtering.

# Chapter 5

## Concluding Remarks

Multicarrier and embedded GSM implementations were examined as the first and final topics of this thesis respectively. In both cases the topics were considered independently, but there is no fundamental restriction on combining the two development areas. Chapter 3 showed that SIMD and DSP optimizations alone were insufficient for running a fully loaded OpenBTS configuration on one particular embedded platform. Looking forward, however, this limitation already has solutions.

The ARM Cortex-A8 processor was tested in this thesis, however, the successor processor is readily available. With rapid processor advancements occurring in embedded, the Cortex-A9 is a multicore processor with significantly higher performance in floating point efficiency, latency, and overall throughput than the A8 edition. Combined with an appropriate RF interface, there is no doubt that such a configuration could serve as part of an effective future SDR platform for higher performance embedded and multicarrier cellular implementations.

Further out on the horizon, advanced signal processing architectures with multiple DSP cores will provide larger increases in capabilities that are sufficient for not just multicarrier GSM, but multi-standard implementations as well. Chapter 2 introduced the concept of extending multirate signal processing techniques to handle arbitrary signal bandwidths and channel spacings. This level of system flexibility will become more and more relevant as future generation cellular technologies, such as LTE and LTE-Advanced, are deployed and operated concurrently alongside existing networks. As these network combinations become more commonplace, in addition to use of heterogeneous small and large cells, the need for reconfigurable components that can scale for capacity, radio technology, and operating frequency will only increase.

While this thesis examined multirate and embedded techniques in the scope of GSM systems, the topics discussed are extensible to new air interface technologies. Because of this applicability, the motivation for advancing the material in this thesis toward future multi-mode systems is very strong and serves as a topic worthy of future study.

# Bibliography

- [1] 3GPP, “Half Rate Speech Transcoding,” 3rd Generation Partnership Project (3GPP), TS 06.20, Nov. 2000.
- [2] —, “Physical layer on the radio path; General description,” 3rd Generation Partnership Project (3GPP), TS 45.001, Sep. 2008.
- [3] —, “Radio transmission and reception,” 3rd Generation Partnership Project (3GPP), TS 45.005, Sep. 2008.
- [4] M. Saily, G. Sebire, E. Riddington, *GSM/Edge: Evolution and Performance*. Wiley, 2011.
- [5] R. Crochiere and L. Rabiner, *Multirate Digital Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1983.
- [6] f. harris, *Multirate Signal Processing for Communication Systems*. Upper Saddle River, NJ: Prentice Hall, 2004.
- [7] Intel, “Intel 64 and IA32 Architectures Software Development Manual,” TS, May 2012.
- [8] 3GPP, “Modulation,” 3rd Generation Partnership Project (3GPP), TS 45.004, Sep. 2008.
- [9] J. Proakis, *Digital Communications*. McGraw-Hill, 1995.
- [10] P. Laurent, “Exact and Approximate Construction of Digital Phase Modulations by Superposition of Amplitude Modulated Pulses (AMP),” *Communications, IEEE Transactions on*, vol. 34, no. 2, pp. 150 – 160, Feb 1986.
- [11] Gumstix, “Overo COMS,” [Online] Available: <http://www.gumstix.com>, April 2012.
- [12] 3GPP, “Radio subsystem synchronization,” 3rd Generation Partnership Project (3GPP), TS 05.10, Sep. 2003.
- [13] ARM, “ARM Architecture Reference Manual: ARMv7-A and ARMv7-R edition,” ARM DDI 0406C, 2011.

[14] —, “Cortex-A8: Technical Reference Manual,” ARM DDI 0344D, 2010.