Signals & Systems

A Reconfigurable Architecture of Software-Defined-Radio for Wireless Local Area Networks

M.Sc. Thesis

Ajay Kapoor

University of Twente
Department of Electrical Engineering,
Mathematics & Computer Science (EEMCS)
Signals & Systems Group (SAS)
P.O. Box 217
7500 AE Enschede
The Netherlands

Report Number:	SAS04-048
Report Date:	March 1, 2005
Period of Work:	1/5/2004 – 7/3/2005
Thesis Committee:	Prof. Dr. ir. C.H. Slump
	Dr. ir. S.H. Gerez
	ir. F.W. Hoeksema
	Dr. ir. R. Schiphorst

Abstract

The Software-Defined-Radio (SDR) project at the University of Twente aims at combining two different WLAN standards, Bluetooth and HiperLAN2, on one common flexible hardware platform. A functional architecture SDR baseband receiver has already been derived which is capable of receiving both OFDM and phase-modulated signals [39]. The scope of this MSc. project is to design and implement an ASIC-like reconfigurable hardware for a part of this architecture.

This project involves the estimation of computational complexities of various subblocks of the two receivers. These results are used for the identification of sub-blocks with similar computational complexities in the two receivers. The FIR and FFT blocks, for Bluetooth and HiperLAN2 respectively, are identified as the most computationally intensive parts and have been further analyzed for computational requirements and hardware implementation in the two receivers. A coarse-grained, dynamically reconfigurable, tile-based hardware architecture is proposed to implement the algorithms. There are nine autonomous tiles (data processing elements) in the system. The autonomous nature of a tile allows easy scalability and testability of the system. The architecture implementation and algorithms mapping is done using SystemC via Synopsys CoCentric System Studio. The design is done using 16-bit fixed-point data format and is compared with the floating point software implementation. Synthesis results show that design consumes 0.59 mm^2 area and can run at 188 MHz maximum frequency in 0.18μ UMC CMOS process.

The proposed implementation is compared with the implementation on the Montium tile processor [26], designed under the Chameleon project [1], in terms of speed and area. This comparison shows an area reduction of about 15 times in our design compared to the Montium TP based implementation. This reduction comes at the expense of limited flexibility. The FFT implementation in this thesis is also compared with various other FFT implementations. This comparison shows a performance/flexibility trade-off between these implementations.

An area reduction of about 25-30 percent can be made in the combined implementation compared to the individual implementations of the two receivers. The datapath of the Bluetooth receiver can be used for the OFDM system without much overhead. The memory and the memory-bandwidth of the OFDM system can be used in the Bluetooth receiver without any overhead. These results can be used to estimate the overhead required to accommodate the Bluetooth receiver in the HiperLAN2 system.

Acknowledgements

The work leading to this thesis was done during my stay at the Signals and Systems (SAS) research group at the University of Twente (UT). The effort that has gone into this thesis has been thoroughly enjoyable due to the healthy interactions I had with my supervisors and other colleagues.

To each of my supervisors, Ir. Fokke Hoeksema, Dr. ir. Roel Schiphorst and Dr. Ir. Sabih Gerez, I owe a great debt of gratitude for their patience and inspiration. So, first of all I want to thank them for their support and encouragement during the work. At the same time, I want to thank the head of the SAS group Prof. Dr. ir. C.H. Slump for allowing me to join his research group in the first place and let me work flexibly.

I would also like to give special thanks to Dr. ir. Paul Heysters of Computer Architecture Design and Test for Embedded Systems (CADTES) group for providing me lot of information about the reconfigurable hardware design concept and ir. Gerard Rauwerda for the discussions about the mapping of algorithms on the Montium TP.

I also want to take the opportunity to thank the staff members of SAS group for the pleasant research atmosphere. Of these especially, to ir. Johan Wesselink for practical tips about tools and methodologies that I followed, ing. Geert Jan Laanstra for system support and Anneke van Essen-Rekers for support on administrative issues.

Finally, I would like to thank my friends Sisir and Praveen for their support during my study time and to Amol and Raajaa for reminding me about the coffee breaks.

This was great fun to do. Thank you everyone.

Contents

List of Figures

List of Tables

1

Introduction

The wireless communication industry is facing new challenges due to constant evolution of new standards (2.5G, 3G, and 4G), existence of incompatible wireless network technologies in different countries inhibiting deployment of global roaming facilities and problems in rolling-out new services/features due to wide-spread presence of legacy subscriber handsets. Software-defined-radio(SDR) technology promises to solve these problems by implementing the radio functionality on a generic hardware platform. Further, multiple modules, implementing different standards can be present in the radio system and the system can take up different personalities depending on the module being used [33].

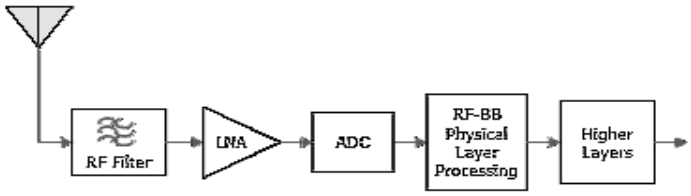

Figure 1.1: SDR architecture

1.1 Background

A software radio transceiver, in its widest meaning, defines a general Transmitter/Receiver architecture that can be completely reconfigured to support multiple services and communication protocols, directly operating on a radio frequency (RF) digitized information stream. Because of the analog nature of the air interface, a radio receiver will always have an analog front end. In an ideal software radio design, a single reconfigurable front end takes care of all the analog interface requirements. Analog processing is limited

at the RF front-end, where a pass-band image-rejection filter selects a large spectrum portion containing the desired services. After Low-noise-amplifier (LNA), an Analog-to-digital converter (ADC) converts the signal with the precision required by the system specifications. The digital RF stream is then fed to a RF baseband(BB) physical layer DSP subsystem (see Figure 1.1 [19]). In that case, the analog-to-digital and digital-to-analog (AD/DA) converters can be positioned directly after the antenna and all the signal processing can be done in digital domain. So, an ideal SDR front end would receive different RF signals through a single reconfigurable antenna and then directly convert them to baseband. But, such an implementation is not feasible due to the power that such device would consume and other physical limitations. It is therefore, a challenge to design a system that preserves most properties of the ideal software radio while being realizable with current-day technology [16].

In analog design, new ways are sought to place the AD/DA blocks closer to RF antenna. This is motivated by the advent of new IC processes which permit the integration of more functionality in the digital domain. The above idea results in implementing more and more functionality digitally in baseband processing, and increases the algorithm complexity in digital domain. The main functions of BB processing are:

- Centers the received signal spectrum to the band of services of interest.
- Lowers the sampling frequency of the digital stream down to the minimum rate required by the standard specifications.
- Operates the necessary filtering in order to reject the unwanted adjacent signals.
- Demodulates channel- and source-decodes the symbol flow and supplies the information bit-stream, for subsequent processing, to higher layers hardware and software.

To realize the complex digital domain supporting multiple demodulation algorithms, an obvious choice can be software implementation to allow easy configurability (using a general purpose processor, GPP). But, a GPP unit will not only require more hardware than needed but also consume much more power than a dedicated hardware unit. The second option is to design a baseband demodulator for each SDR algorithm separately and connect it to single analog front end. This is motivated by the advancement in technology, which allows integration of billion of transistors on a single chip. This implementation, though, saves energy but will increase hardware enormously. Lot of hardware will be unused at any given time.

The third option is to design a reconfigurable system which reuses some or most of the hardware to support different services. This is an exciting

opportunity for computer architects and designers to come up with system designs that efficiently use the huge transistor budget and meet the requirements of future SDR applications. The development of personal mobile devices will give an extra dimension, because these devices have a very small energy budget, are small in size but require a performance that exceeds the levels of current desktop computers. The functionality of these mobile computers will be limited by the required energy consumption for communication and computation. This will require choosing the demodulation algorithms with similar computations and then design a reconfigurable hardware to implement those algorithms. This requires and allows implementation of SDRs in terms of dedicated, but reconfigurable hardware.

In September 2000, the Signals and Systems group started one such software-defined radio (SDR) project. In order to keep the complexity of the project realistic, it was decided to concentrate on a platform that would be able to support two standards: HiperLAN2 and Bluetooth. In the first part of this SDR project, a functional architecture SDR baseband receiver has been derived which is capable of receiving both OFDM and phase-modulated signals [39]. The basis for these designs were the performance requirements and the compatibility between the two demodulators. To verify the functionality and performance of these designs, an implementation on a notebook PC(GPP) was done. Successful communication was proven in a demonstrator that included two PCs, some dedicated digital hardware and a suitable analog front end that was also designed as part of the project. In this setup, most of the signal processing is done on the Pentium-IV processor [47]. This implementation of the algorithms was based on floating-point arithmetic.

1.2 Assignment

In this second part of the SDR project, an efficient hardware implementation of the demodulation algorithms is sought for. This graduation project investigates the *design and implementation of flexible hardware architecture* for a part of the developed SDR receiver.

In the SDR receiver, the most computationally intensive parts are Fast-fourier-transform(FFT) for HiperLAN2 and channel selection and matched filtering for Bluetooth. The main focus of this thesis is to design and implement an efficient, reconfigurable architecture for these parts.

This thesis mainly deals with the following issues:

- Understanding the SDR architecture and identification of parts with similar computations and computational load .

- Architecture design to satisfy the contradictory requirements of reconfigurability, hardware, efficiency and real time performance. This is the central issue of the project.

- Implementation of chosen algorithms and performance evaluation after mapping of algorithms.

- Performance evaluation with respect to floating point implementation.

- Hardware overhead estimation in HiperLAN2 due to Bluetooth functionality.

The above investigations have lead to a prototype implementation. The main tools that are used for this project are: Synopsys CoCentric System Studio, for algorithmic design (e.g. for the modeling of the environment outside the hardware such as the analog front-end, the channel, etc.) and architectural design in SystemC; Synopsys Design Compiler for the synthesis from SystemC/VHDL/Verilog to gates from a standard-cell library; SystemC to verilog converter from open design cores [3]. The technology used for synthesis is 0.18μ UMC CMOS process.

1.3 Organization

This thesis is organized into the following sections:

1. Chapter 2 starts with the basic introduction to Bluetooth and HiperLAN2 physical layer. It also provides the basic receiver architecture for both standards [39].

2. Chapter 3 discusses the sections of baseband demodulation algorithms of our SDR, along with their computational complexity. The channel-selection algorithm for the Bluetooth receiver and the OFDM algorithm for HiperLAN2 are identified as most computationally demanding algorithms in the two receivers. These algorithms are implemented in this thesis.

3. Chapter 4 analysis the computational schemes for algorithms of interest. This helps us in identifying the datapath computations and control schemes for our hardware.

4. Chapter 5 provides an introduction to the concept of reconfigurable architecture and main features of various contemporary reconfigurable architectures. This study helps us identifying the main considerations for reconfigurable DSP hardware design. A comparison of various design approaches is also part of discussion in this chapter.

5. Chapter 6 explains the proposed architecture that is developed and implemented in this thesis. Its main features are highlighted.

6. Chapter 7 explains the mapping of SDR algorithms on the proposed design. The discussion here helps us in understanding the complete dataflow and real-time performance requirements in our design.

7. Chapter 8 evaluates the synthesis results of our design and compares it with the performance of state-of-art Montium tile processor (TP) recently designed at the University of Twente (UT) [26]. A quick comparison with some other FFT implementations is also provided there.

8. Chapter 9 summarizes our design flow and architecture design approach, It concludes this thesis with final conclusions and future research possibilities of the system.

9. Appendix-A provides the schematic overview of our system.

10. Appendix-B provides the SNR degradation in fixed point finite precision implementation compared to floating point implementation.

11. Appendix-C gives a brief introduction to SystemC design methodology and Synopsys CoCentric System Studio for algorithmic and architectural design.

2

WLAN standards- HiperLAN2 and Bluetooth

SDR project at Signals and Systems (SAS) group, aims to combine two different types of standards -Bluetooth and HiperLAN2, on one common hardware platform. HiperLAN2 is a high speed Wireless LAN (WLAN) standard [21, 22], whereas Bluetooth is a low-cost and low-speed Personal Area Network (PAN) standard [41]. Table 2.1 provides the physical layer overview of both standards. As can be seen from the table, these standards differ with each other in several aspects and pose an interesting challenge for an SDR platform.

System	Bluetooth	HiperLAN2
Frequency Band	2.4-2.4835 GHz	5.150-5.300 GHz, 5.470-5.725 GHz
Access Method	CDMA	TDMA
Duplex Method	TDD	TDD
Modulation	GFSK	OFDM
Max. Data Rate	1 Mbps	54 Mbps
Channel Spacing	1 MHz	20 MHz
Max Power Peak	100 mW	200 mW -1 W

Table 2.1: Physical Layer Overview

This chapter gives a brief introduction to Physical layer of HiperLAN2 and Bluetooth and also suggests the generic transmitter, receiver model. The model will provide an insight in the demodulation functions that are necessary in HiperLAN2 and is used for determining channel selection and computational requirements for the SDR project.

2.1 HiperLAN2

HiperLAN2 is a high-speed WLAN standard [21] using Orthogonal Frequency Division Multiplexing (OFDM) modulation in the 5 GHz frequency band. It has been developed by the European Telecommunications Standard Institute (ETSI). The physical layer is very similar to the American Institute of Electrical and Electronics Engineers (IEEE) 802.11a standard. The transmission format on the physical layer is a burst, which consists of a preamble and a data part. The frequency spectrum available to HiperLAN2 is divided into 19 so called channels, which are referred as radio channels. Each of those radio channels has a bandwidth of 20 MHz. Orthogonal frequency division multiplexing (OFDM) has been chosen as modulation technique in HiperLAN2. OFDM is a special kind of multicarrier modulation. This modulation technique divides the high data rate information in several parallel bit streams and each of those bit streams modulates a separate subcarrier. The physical layer transmits 52 subcarriers in parallel per radio channel. Four of the 52 subcarriers are used to transmit pilot tones. Those pilots assist the demodulation in the receiver. A HiperLAN2 MAC frame consists of 5 parts and has a maximal duration of 2 ms.

2.1.1 Transmitter

The HiperLAN2 transmitter [39] starts with mapping raw bits on QAM symbols (BPSK, QPSK, 16 QAM or 64-QAM symbols). In the next step, the QAM symbols are mapped on data carriers and an OFDM symbol is constructed by adding pilot carriers, applying an inverse FFT (for OFDM) and adding an prefix, which results in a 20 MSPS signal. MAC bursts are then created by adding special symbols, preambles, to the start of the MAC burst. The PHY layer provides transportation mechanisms of bits between the DLC layer in transmitter and receiver. The standard defines seven functions in the transmitter, namely,

- Scrambling of the binary input stream.
- Forward Error Correction (FEC) coding.
- Interleaving.
- QAM Mapping.
- Modulation using OFDM.
- Physical burst generation.
- Transmitting of the burst.

Figure 2.1 shows the block diagram of HiperLAN2 transmitter.

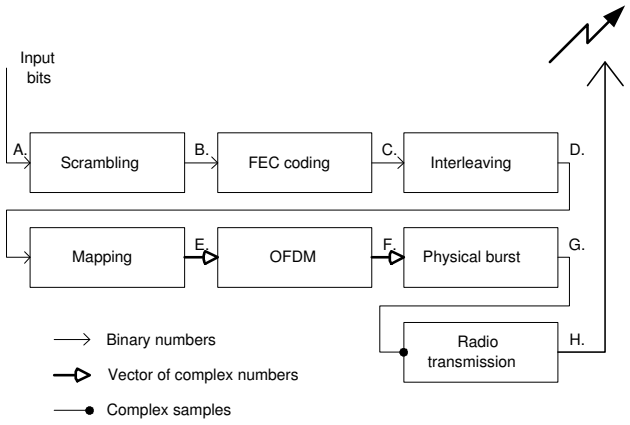

Figure 2.1: Transmitter block diagram for HiperLAN2

2.1.2 Receiver

The receiver not only has to convert the received signal to data bits by performing the inverse of the transmitter, but also has to try to compensate for the distortions caused by the radio channel. The HiperLAN2 receiver [39] can roughly be divided into two parts, a time domain part and a frequency domain part. In the first stage of the receiver, signal functions will be time domain functions. In the second stage of the receiver, signal functions will be frequency domain functions. Most of the operations can be performed in time domain and in frequency domain. The location of the functions in the receiver architecture is based upon a trade-off between the necessary resolution that must be reached for a certain correction and the solution with the minimum number of operations. One also tried to keep the corrections independent of each other by deciding the execution order of the functions. The HiperLAN2 receiver starts by searching for the start of a MAC burst. If found, it estimates the frequency offset and channel parameters. After these steps the data OFDM symbols can be demodulated by first correcting the frequency offset, performing an FFT, correcting the channel and detecting and correcting the phase offset by using the pilot tones. The outputs are QAM symbols, which have to be de-mapped into raw bits. A HiperLAN2 receiver should at least perform the following functions at physical layer:

- Synchronization and parameter estimation function.
- Frequency offset corrector.
- Phase offset corrector.
- Channel equalizer.

- Inverse OFDM.
- De-mapping.
- De-interleaving.
- Viterbi-decoder.
- De-scrambling.

Figure 2.2 shows the block diagram of HiperLAN2 receiver.

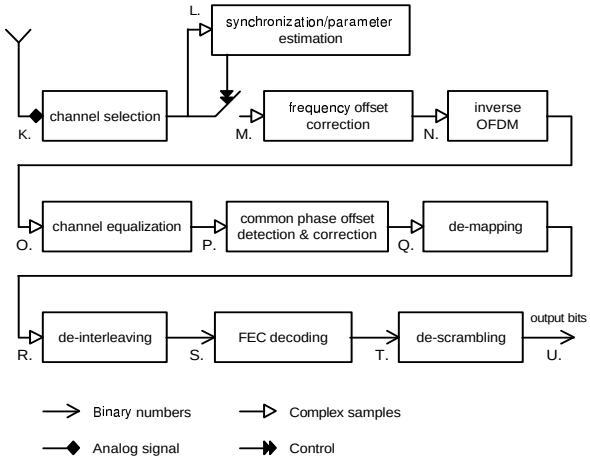

Figure 2.2: Receiver block diagram for HiperLAN2

2.2 Bluetooth

The frequency spectrum available to Bluetooth [41] is positioned in an unlicensed radio band that is globally available. This band, the Industrial, Scientific, Medical (ISM) band, is centered on 2.45 GHz. In most countries, free spectrum is available from 2400 MHz to 2483.5 MHz. The frequency spectrum is divided into 79 so called channels, which are referred as radio channels. Each of those radio channels occupies a bandwidth of 1 MHz. For robustness, a binary modulation scheme was chosen. With the mentioned bandwidth restriction, the data rates are limited to about 1 Mbps. Bluetooth uses Gaussian shaped frequency shift keying (GFSK) modulation with a nominal modulation index of h = 0.32. Logical ones are sent as positive frequency deviations, logical zeros as negative frequency deviations. The channel is a hopping channel with a nominal hop dwell time of 625 μs. The Bluetooth system uses packet-based transmission: the information stream is fragmented into packets. In each slot, only a single packet can be sent. All packets have the same format, starting with an access code, followed by a packet header, and ending with the user payload.

2.2.1 Transmitter

In the PHY layer of the Bluetooth transmitter, the first step [39] is to embed the raw bits into MAC bursts, which are then BPSK modulated at 1 Mbit/s. The BPSK symbols are filtered by a Gaussian low pass filter and the filtered output is connected to a VCO that translates the amplitude variation into frequency variations. Its functional architecture is shown in Figure 2.3. The architecture contains a physical burst, which creates packets from a bit stream. These packets contain besides the payload, a packet header and a device-specific access code. After packet generation, the packet will be modulated using GFSK modulation. The output of the GFSK modulation function is a complex baseband signal (with carrier frequency of 0 Hz). The final step in the transmitter is to convert the baseband signal to RF frequencies.

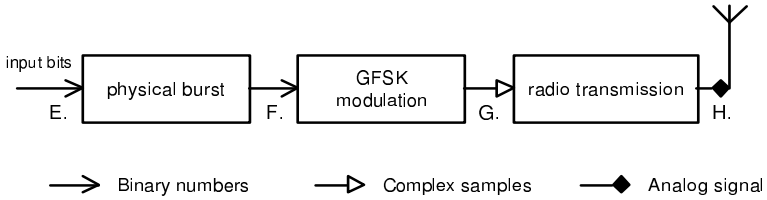

Figure 2.3: Block diagram for Bluetooth Transmitter

2.2.2 Receiver

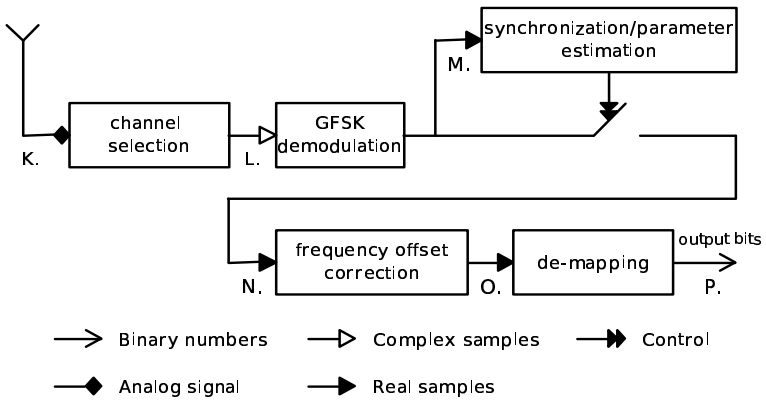

Figure 2.4: Block diagram for Bluetooth Receiver

Figure 2.4 shows the functional architecture of the Bluetooth receiver [39]. In order to test the SDR receiver functionality, the transmitter is implemented from point E to H, the whole PHY layer.

At the receiver side [39], the first step is to select the wanted Bluetooth channel and suppressing all others, which is performed both digitally and by the analog front-end. This is achieved by mixing the wanted channel to zero IF and applying a low-pass filter. The next step is to demodulate the FM signal using MAP receiver. This receiver requires an orthogonal vector space, which is given by the Laurent decomposition [32]. This Laurent decomposition describes the GFSK signal by a sum of linear, orthogonal, Pulse Amplitude-Modulated (PAM) waveforms. Demodulation using MAP receiver requires first passing the signal through low pass filter [38]. This filter also acts as matched filter for input signal. Then the signal is frequency corrected and decoded using Viterbi decoding. The synchronization/parameter estimation entity uses this signal to detect the start of a MAC burst (time/symbol synchronization) and estimates the frequency offset. A frequency offset introduces a Direct Current (DC) value in the AM signal and therefore it has to be corrected before bit decision.

2.3 Summary

This chapter very briefly discusses Bluetooth and HiperLAN2 standards. A comprehensive summary has been given in [39]. In the next chapter, we will discuss the computational complexity of baseband demodulation algorithms for our SDR.

3

Baseband Demodulation

In the SDR project at UT, the basic thinking was that the HiperLAN2 hardware is that complex compared to the Bluetooth hardware that Bluetooth capability may be added to the HiperLAN2 platform at limited cost [47]. So, it was not the demand for flexibility (one front-end for all signals), but the idea of providing added functionality nearly "for free" was the main motivating factor. From a software-radio perspective the issues were to determine which functions can be identical for both standards, which functions were different (and should be switch able at the time instant a particular standard is selected) and which functions can be parameterizable (identical functions with parameters depending on the selected standard).

In the current implementation, algorithms for demodulation are implemented on GPP hardware [39] and the analog front-end of SDR is already made to be flexible and reconfigurable [46].

This thesis focuses on the hardware implementation of digital baseband (BB) part of the receiver (PHY layer only). This chapter discusses how various building blocks of baseband demodulation has been designed in software to combine the two receivers. Later, we will also estimate the computational complexity of these blocks to realize them in hardware. For all parts, we assume that 16-bit fixed point calculations are sufficient [27].

Input data is coming in the BB receiver after the analog front end (including ADC) at the rate of 80 MSPS. The digital baseband part consists of a a sample rate reduction block followed by digital demodulator block. The sample rate reduction block performs sample-rate reduction from 80 MSPS to 20 MSPS and selects the channel corresponding to one HipereLAN2 channel. This channel is of 10 MHz bandwidth. The output from sample rate reduction block is fed to the digital demodulator part which demodulates the data stream digitally.

As described in chapter 2, in HiperLAN2, QAM mapped symbols are modulated by OFDM, while in Bluetooth, BPSK symbols are modulated using GFSK. For realizing both kinds of demodulators on one common

hardware, similar algorithms have been developed to demodulate the signals. The functional architecture of the Bluetooth receiver and the HiperLAN2 receiver for SDR receiver has been described in [39] in detail. Figure 3.1 shows the functional architecture of the Bluetooth enabled HiperLAN2 receiver.

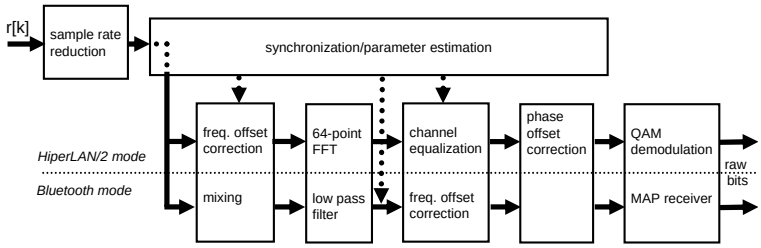

Figure 3.1: Functional architecture of the Bluetooth enabled HiperLAN2 receiver

3.1 HiperLAN2

Input data rate for BB demodulator is 20MSPS. This data signal consists of OFDM symbols. One OFDM symbol has a duration of 4 μs (80 complex samples) with 48 data and 4 pilot carriers. A MAC frame consists of 5 parts. For estimating computational requirements [37], all parts having equal duration and demodulation requirement of 2 parts (one common and one user part) are assumed . These part have a duration of $(2000/5) * 2 = 800\mu$s (i.e., 200 OFDM symbols). Thus, number of transmitted OFDM symbols per second are $(1/2e - 3) * 200 = 100000$ symbols. In the text below, we will estimate the computational complexity of various building blocks of HiperLAN2 baseband demodulator.

3.1.1 OFDM

After frequency offset correction, the first step is inverse OFDM in HiperLAN2 demodulator as shown in Figure 3.1. The inverse OFDM is same as Fast-Fourier-transform (FFT) operation. An OFDM symbol has duration of 80 complex samples. Only 64 samples of them are needed for the FFT. The remaining 16 samples are used as cyclic prefix to reduce inter symbol interference (ISI) and synchronization. So, the first step in the receiver is to pass the data through 64-point FFT block. After examining various FFT algorithms [2,34,45,48], we chose to use radix-2 FFT in our implementation. The reason for choosing this algorithm will become clear in the chapter 4.

Radix-2 FFT is performed using radix-2 butterflies and requires $64 * \log_2(64)$ complex multiplications. So, the requirements are 384 16-bit complex mul-

tiplications for each OFDM symbol. Data will be coming out from FFT at $(64/80) * 20 = 16$ MSPS (see Figure 3.2).

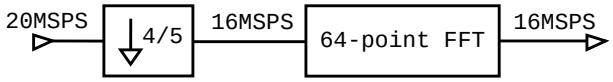

Figure 3.2: Inverse OFDM in HiperLAN2 receiver

3.1.2 Channel equalization

After FFT, the channel equalizer block has to compensate the channel for the carriers. The estimation of the channel is done by comparing the known preamble and the received subcarrier values. This equalization should be done for 52 subcarriers. So, it will require 52 complex multiplications per OFDM symbol. Channel equalization block works at $(52/64) * 16 = 13$ MSPS (see Figure 3.3).

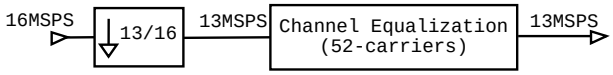

Figure 3.3: Channel equalization in HiperLAN2 receiver

3.1.3 Phase offset correction

At the front-end of the receiver, frequency-offset correction is implemented by calculating only the values of the frequency offset for the first symbol and these values are subsequently reused for other symbols. This saves (computational-intensive) instructions (cos and sin) but also introduces a phase offset. This phase offset can be corrected by using the pilot carriers in the OFDM symbol. This requires 48 complex multiplications. Thus, phase offset block works at $(48/52) * 13 = 12$ MSPS (see Figure 3.4).

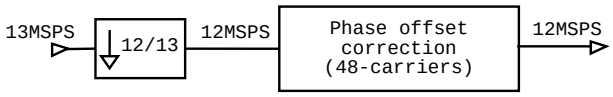

Figure 3.4: Phase offset correction in HiperLAN2 receiver

3.1.4 QAM Demapping

Final step in demodulation of HiperLAN2 receiver is demapping. In HiperLAN2 there are four constellations available: BPSK, QPSK, 16-QAM and 64-QAM. Each of these constellation has a different number of bits per complex symbol. Demapping can be done using look up table. In the lookup

Function	DataRate	Number of multiplications	Number of additions
64 point FFT	16	153.6e6	76.8e6
Channel equalization	13	20.8e6	10.4e6
Phase offset correction	12	19.2e6	10.4e6
64-QAM demapping	12	9.6e6	9.6e6

Table 3.1: Computational requirements for HiperLAN2 receiver

table, all possible subcarrier values for a certain mapping scheme are defined. For BPSK, 2 subcarrier values are stored in the lookup table; for QPSK, 16-QAM and 64-QAM there are 4, 16 and 64 subcarrier values stored, respectively. The largest constellation used is 64-QAM. A 64-QAM symbol has $2^3 = 8$ possible values for both the real and imaginary part. Demapping can be implemented by generating an index for a table. So demapping requires 2 comparisons (border checking), 1 addition, 1 multiplication and 1 table lookup.

The computational complexity of the building blocks of HiperLAN2 baseband demodulator is summarized in Table 3.1 [37].

3.2 Bluetooth

The Bluetooth symbol duration is 1 μs. The symbols are modulated using GFSK modulation scheme. Data is transmitted in time slots with duration of 625 μs [41]. As in HiperLAN2 input data in the BB receiver is coming at 20 MSPS. This data is of 10 MHz bandwidth. But, each channel of Bluetooth has bandwidth of 1 MHz. So, input data consists of lot of redundant and undesired information.

The first step in Bluetooth receiver is to select the information corresponding to desired channel and reduce the incoming data rate to remove redundant computations in subsequent blocks. This corresponds to mixing and low pass filtering steps shown in Figure 3.1.

To demodulate the GFSK signal, the SDR receiver uses Maximum A Posteriori Probability (MAP) receiver algorithm [38] in the Bluetooth system. For this purpose, GFSK signal is described by a sum of linear, orthogonal, pulse amplitude modulated (PAM) waveforms using the Laurent decomposition [32]. It has enabled us to represent GFSK signal by orthogonal vector space which is a requirement for MAP receiver [38]. In the (MAP) receiver, there are two steps performed. The first step is to perform matched filtering and second step is to perform Viterbi decoding (see Figure 3.5).

From the implementation point of view, the matched filtering is similar to low pass filtering step. So, these two steps are combined together and performed after mixing step in the actual implementation. This will become

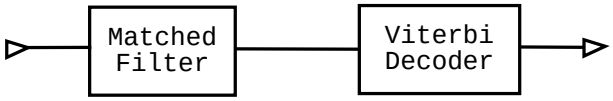

Figure 3.5: MAP receiver

clear in chapters 4 and 7. In this way, low pass filtering is combined with matched filtering and only Viterbi decoding is done in MAP receiver stage of our receiver.

For estimating computational requirements, we assume maximal transfer rate. In this mode, Bluetooth uses a packet, which spans 5 time slots, and 1 time slot is used for uplink communication.

3.2.1 Mixing

After Analog front end (including ADC), input data is coming in baseband demodulator is coming at 20 MSPS. This data is first converted into baseband by mixing. This requires one complex multiplication (i. e. 4 multiplication and 2 additions per input sample). This will require $(20 * 4) = 80$ 16-bit multiplications per second and $(2 * 20) = 40$ 16-bit additions per second. This step is shown in Figure 3.6.

Figure 3.6: Mixing

3.2.2 Sample rate reduction

The incoming data rate for this block is 20 MSPS. So, the first step is reduce this data rate. This is performed using two halfband filters each decimating the input stream by a factor two. Each halfband filter is of 7^{th} order and have linear phase. So, A decimation factor of 4 is applied to reduce the data rate to 5 MSPS. A one-to-one implementation of this step will require $(2 * 7 * 20 + 2 * 7 * 10) = 420$ 16-bit multiplications per second and $(2 * 6 * 20 + 2 * 6 * 10) = 360$ 16-bit additions per second. These computations are an upper estimate and can be reduced by exploiting linear phase and halfband property of the filters. This step is shown in Figure 3.7.

3.2.3 Low pass filtering

As explained before, this low pass filter block selects the desired channel and perform the matched filtering for MAP receiver block. Input and output

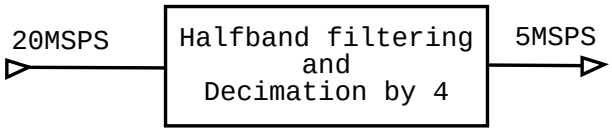

Figure 3.7: Sample rate reduction

data rate for this block is 5 MSPS. Low pass filter used here is of 17^{th} order linear phase filter. This will require $(2 * 17 * 5) = 170$ 16-bit multiplications per second and $(2 * 16 * 5) = 160$ 16-bit additions per second. Again, linear phase property can be used to reduce the number of multiplications by two. Figure 3.8 shows the data flow for this block.

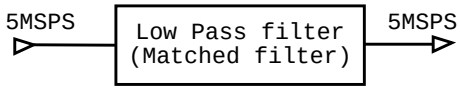

Figure 3.8: Low pass filtering to select the desired channel in Bluetooth

3.2.4 Frequency offset correction

The MAP receiver has a very good performance but it requires a very precise knowledge of signal properties such as phase offset, frequency offset and modulation index. This precise knowledge is required because these effects influence the position of the states in the trellis diagram. Moreover, the receiver uses the history of all received signals, and therefore small estimation errors will already result in bit errors. So, the next step in receiver is frequency offset correction of input signal.

The frequency offset is estimated by the synchronization/parameter estimation part and corrected in the frequency-offset correction part of the receiver. It requires one complex multiplication per sample. Moreover the influence of the frequency offset on each symbol/sample has to be calculated, which requires 2 multiplications and 2 table lookups. The input sample rate for this block is: $(5/6) * 5 = 4.15$ MSPS. A factor of 5/6 is used because 1 out of 6 time slot is used for uplink. Synchronization and parameter estimation block ensures correct timing information and output data rate for this block is reduced to 0.83 MSPS. Input and output data rate for this block are shown in Figure 3.9.

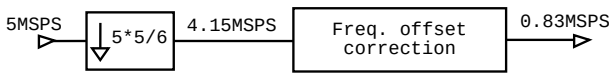

Figure 3.9: Frequency offset correction in Bluetooth

Function	DataRate	Number of multiplications	Number of additions
Mixing	20/20	80e6	40e6
Decimation/Halfband	20/5	420e6	360e6
Matched filter	5/5	170e6	160e6
Freq. offset correction	4.15/0.83	20e6	8.5e6
Viterbi	0.83	29.9e6	21.6e6

Table 3.2: Computational requirements for Bluetooth receiver

3.2.5 MAP receiver

The matched filtering corresponding to MAP receiver is already done in low pass filtering block. So, the MAP receiver consists of a 2-state Viterbi algorithm. This algorithm has to calculate 2 branches for each state and select the best branch. The state with the highest values determines the detected bit. Each branch requires 2 or 3 complex multiplications. In total, the Viterbi algorithm requires 9 complex multiplications, 4 complex additions and 3 comparisons (36 multiplications, 26 additions and 3 comparisons) for each sample. The Viterbi algorithm block operates at 0.83 MSPS (See Figure 3.10). So, total number of multiplications per second and additions for this stage are 29.9e6 and 21.6e6 respectively.

Figure 3.10: Viterbi decoding in Bluetooth

The computational complexity of various building blocks of Bluetooth baseband demodulator is shown in Table 3.2.

3.3 Summary

In this chapter, the architecture, various algorithm steps for demodulation and the functionality of various building blocks of SDR receiver has been explained. This has helped us in estimation of the computational complexity of various blocks of SDR receiver.

It is clear from this analysis that the OFDM block in HiperLAN2 and the matched filter along with halfband filtering blocks in Bluetooth are the most computationally intensive blocks in the two demodulators. Therefore, the main aim of this thesis is to design a reconfigurable hardware for these two blocks. The algorithms corresponding to these two steps will be further analyzed in chapter 4.

Implementation of our design is done using SystemC in Synopsys Co-Centric System Studio. A brief introduction to SystemC and Synopsys Co-Centric System Studio for algorithmic and architectural design is provided in appendix-C.

4

Algorithms analysis

The algorithm domain of the SDR project includes baseband demodulation algorithms for HiperLAN2 and Bluetooth. Detailed description of these algorithms can be found in [39]. A brief description along with the assessment of the computational complexity of these algorithms is provided in the chapter 3. In this thesis, we are dealing with the hardware implementation of the channel-selection block of Bluetooth receiver and OFDM block of HiperLAN2 receiver. (The halfband filter block and matched filter block are combined together into one channel-selection block in the Bluetooth receiver). For this purpose, our first step is to perform the dataflow analysis in various computations of these algorithms.

This chapter begins with the analyzing the algorithms in channel-selection (for Bluetooth) and FFT (for HiperLAN2) sections of the baseband demodulator. Next, it discusses the corresponding signal flow graph and the dominant kernels for each algorithm. This helps us in designing the datapath and control sections of our hardware realization.

4.1 Dataflow for Channel-selection/FFT

1. The first block in baseband demodulation of HiperLAN2 receiver is 64 point FFT block. This block is used for OFDM demodulation. The data from the sample rate reduction block is coming at 20 MSPS. This data is arranged in blocks of 80 samples each. Due to OFDM scheme, last 16 samples are same as first the 16 samples in each block. So, we need to take 64 samples out of these 80 samples. A simple schematic for FFT section of HiperLAN2 is shown in Figure 4.1.

Figure 4.1: FFT of HiperLAN2

2. The first block in the baseband demodulation of Bluetooth receiver is channel-selector/Low pass filter (LPF). This is required to select the desired 1 MHz bandwidth(BW) channel. As explained in the previous chapter, the complexity and data computation unit of FFT block is similar to LPF section of Bluetooth. So, in our implementation, we propose to combine FFT with LPF. But, direct implementation of LPF is computationally intensive. This is against the original thinking of SDR project (HiperLAN2 is complex and Bluetooth can be implemented without much additional costs). So, a one-to-one mapping for LPF is not useful. Actually, LPF is similar to matched-filter in MAP receiver part (of Bluetooth). Matched filter also needs to select the data in 1 MHz BW. So, matched-filtering operation is moved from MAP receiver part to channel selection part. Also, input data stream into demodulator block is of 20 MSPS. Doing Bluetooth demodulation on 20 MSPS will involve lot of redundant computations and will require a very high order matched filter. So, input data is first passed through two linear phase half band filters. Each half band filters decimates data by factor 2. These half band filters help in reducing the order of matched filter. Also, matched filter can be designed to be linear phase. In this way, number of computations can be reduced further. A simple schematic for channel-selector section of Bluetooth is shown in Figure 4.2.

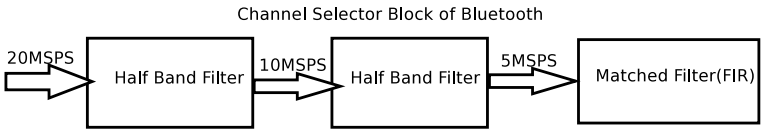

Figure 4.2: Channel-selector section of Bluetooth.

4.2 Signal flow graph for FIR/FFT

The signal flow graphs and basic building blocks corresponding to half band filter, matched filter and FFT (Butterfly) are described below.

4.2.1 Halfband filter

Input data stream in Bluetooth is filtered through halfband filters before doing low pass filtering. There are two halfband filters. Each halfband filter is of 7^{th} order. To simplify the computations, main points to remember about this building block are: linear phase, halfband and decimation. By using linear phase property, we can reduce the number of multiplications by a factor 2. Halfband property means number of multiplications (corresponding

to amount of zeros in filter coefficient) can be reduced further. Also, using a polyphase representation, decimation can be used to reduce the speed of computation. A basic 7^{th} order FIR filter can be represented as in equation:

$$H(z) = a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + a_4 z^{-4} + a_5 z^{-5} + a_6 z^{-6} \tag{4.1}$$

Its critical path contains one multiplier and six adders. A direct form implementation of such filter is shown in Figure 4.3.

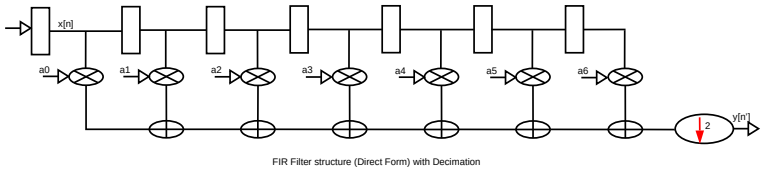

Figure 4.3: Direct form FIR filter

The transposed form of above filter is shown in Figure 4.4. Its critical path contains one multiplier and one adder only.

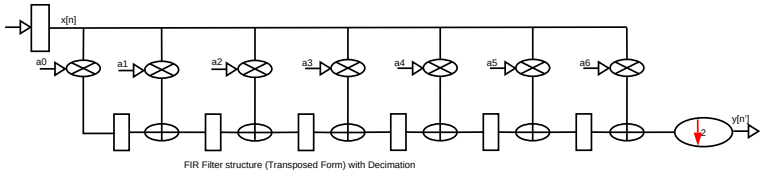

Figure 4.4: Transposed form FIR filter

The halfband property of the filter implies that a_1 and a_5 have zero value and can be omitted to reduce the number of multiplications required. Also, the linear phase property implies that $a_2 = a_4$ and $a_0 = a_6$. So, the multiplications in first half of the filter are identical to the multiplications in other half. Thus, equation 4.1 can be rewritten as:

$$H(z) = a_0 + a_2 z^{-2} + a_3 z^{-3} + a_2 z^{-4} + a_0 z^{-6} \tag{4.2}$$

By using polyphase representation, decimation by 2 can be used to reduce the speed of computations (if needed). Thus, equation 4.2 can be written in polyphase form as:

$$H(z) = (a_0 + a_2 z^{-2} + a_2 z^{-4} + a_0 z^{-6}) + z^{-1}(a_3 z^{-2}) \tag{4.3}$$

The simplified structure, which is computationally most efficient in terms of speed of operation and in terms of amount of datapath computations, is shown in Figure 4.5.

In this way, number of multiplications can be reduced by a factor of 3/7 from direct form halfband filter. Also, each computation unit can work at half of the incoming data rate.

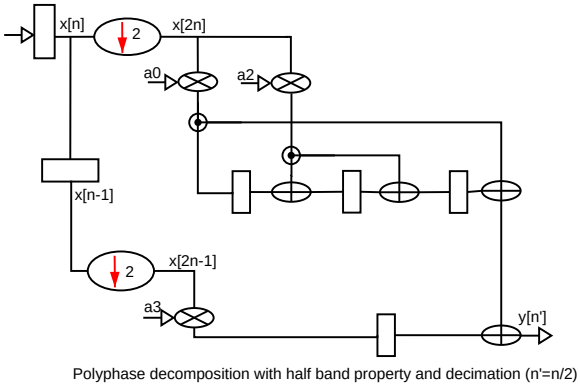

Figure 4.5: Filter structure simplification

Moreover, it is important to notice that the filter structure above has a basic computation unit (shown in Figure 4.6). The repetitive use of this unit realizes the filter. The basic operation can be described as multiply and add.

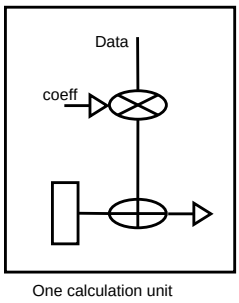

Figure 4.6: Filter calculation unit

4.2.2 FIR (Matched filter)

After halfband filtering, the input data (decimated by 4) is fed to matched filter block. The output of this block is the data corresponding to desired channel. The matched filter used in SDR project is of 17^{th} order. The transposed form representation is shown in Figure 4.7. The basic computation unit is the same the one for half band filters (shown in Figure 4.6). Polyphase decomposition for efficient decimation and half band properties are not applicable for this stage. So, filter structure is corresponding to transposed form structure with linear phase. This means that number of multiplications can be reduced by 2.

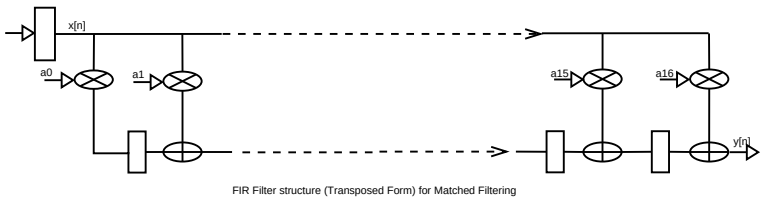

Figure 4.7: Transposed Form LPF for Matched Filtering

4.2.3 FFT

In HiperLAN2, data from ADC block is demodulated by using OFDM demodulator. AN OFDM demodulator consists of a FFT block.

An FFT represents set of algorithms to compute discrete Fourier transform (DFT) of a signal efficiently. An N-point DFT corresponds to the computation of N samples of the Fourier transform at N equally spaced frequencies, $\omega_k = 2\pi k/N$, i.e., at N-points on the unit circle in the z-plane. The DFT of a finite-length sequence of length N is

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \cdots \forall k \in \{0, 1, ...N-1\} \tag{4.4}$$

where, $W_N^{kn} = e^{-j2\pi/N}$.The idea behind almost all FFT algorithms is based upon divide and conquer strategy and establishes the solution of a problem by working with a group of subproblems of the same type and smaller size. In general, each algorithm can be represented either as *decimation in time (DIT)* or *decimation-in-frequency (DIF)*. These two can be thought of as transposed form of each other. An elaborate description of various FFT algorithms can be found in [2, 34, 45, 48].

An objective choice for the best DFT algorithm can not be made without knowing the constraints imposed by the environment in which it has to operate. The main criteria for choosing the most suitable algorithm are amount of required arithmetic operations (costs), and regularity of structure. Several other criteria (e.g. latency, throughput, scalability, control) also play major role in choosing a particular FFT algorithm. We have chosen radix-2 DIF FFT implementation for our system because it has advantages in terms of regularity of hardware, ease of computation and number of processing elements. Also, the basic butterfly corresponding to radix-2 can be combined easily with filter processing element (of our implementation). This facilitates the similar datapath computations in two receivers and simple control structure for HiperLAN2 receiver.

Radix-2 FFT

As mentioned above, OFDM is implemented using radix-2 FFT in our implementation. We have chosen to implement DIF version of radix-2 FFT. This gives us the option of omitting the bit reversal step in the receiver and transmitter of HiperLAN2. The computations in DIF radix-2 FFT are shown in following equations.

$$X[k] = \sum_{n=0}^{N-1} x[n]W_N^{kn}, k = 0, 1, ...N-1 \tag{4.5}$$

which can be expressed as

$$X[2r] = \sum_{n=0}^{N/2-1} (x[n] + x[n+N/2])W_{N/2}^{rn} \cdots \forall r \in \{0..N/2-1\} \tag{4.6}$$

and,

$$X[2r+1] = \sum_{n=0}^{N/2-1} (x[n] - x[n+N/2])W_{N/2}^{rn}W_N^n \cdots \forall r \in \{0..N/2-1\} \tag{4.7}$$

Thus, on the basis of above equations, with $g[n] = x[n] + x[n+N/2]$ and $h[n] = x[n] - x[n+N/2]$, the DFT can be computed by first forming the sequences g[n] and h[n], then computing h[n]W_N^n, and finally computing the N/2-point DFTs of these two sequences to obtain the even-numbered output points and the odd-numbered output points respectively. Proceeding in the manner similar to above, we note that N/2 point DFTs can be computed by computing the even and odd numbered output poins separately and so on. This procedures is illustrated for the case of an 8-point DFT in Figure 4.8.

If N is a power of 2, then eventually we are left with the computations of 2 point DFTs. These 2 point DFT are the elementary computation unit of radix-2 DIF FFT computation. A single 2 point DFT (also known as radix-2 butterfly) can be calculated by the following equations.

$$A_{re} = a_{re} + b_{re} \tag{4.8}$$

$$A_{imag} = a_{im} + b_{im} \tag{4.9}$$

$$B_{re} = (a_{re} - b_{re})W_{re} - (a_{im} - b_{im})W_{im} \tag{4.10}$$

$$B_{imag} = (a_{im} - b_{im})W_{re} + (a_{re} - b_{re})W_{im} \tag{4.11}$$

where, subscripts "re" and "im" represents real and imaginary part of data respectively, and $W = e^{-j2\pi k/N}$. The corresponding signal flow graph is shown in Figure 4.9 and is decomposed further in Figure 4.10. So, a single butterfly computation requires 4 multipliers and six adder/subtrator blocks. Different inputs and outputs of this butterfly structure can also be seen from the Figure. In an N-point FFT, there are $\log_2$N stages and N/2 butterflies.

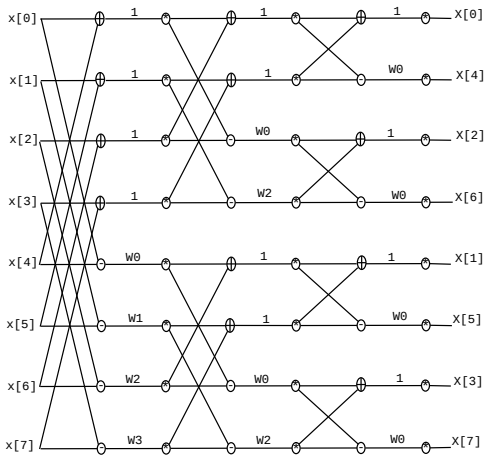

Figure 4.8: Flow graph of DIF decomposition of 8-point, radix-2 FFT

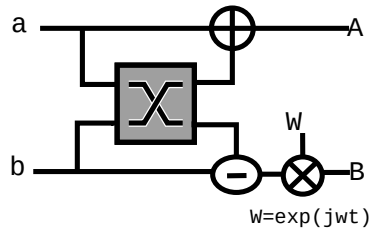

Figure 4.9: Radix-2 butterfly structure

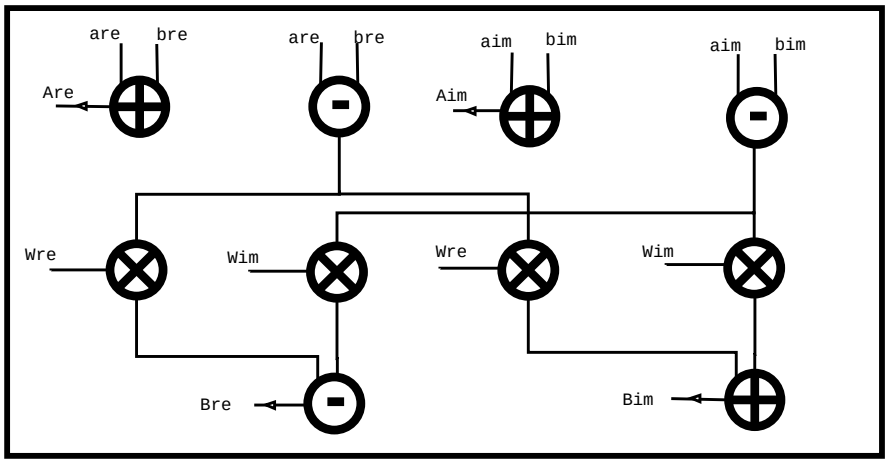

Figure 4.10: Radix-2 butterfly computation

4.3 Summary

In this chapter, we have discussed the various algorithms that need to be implemented in hardware. The hardware should be reconfigurable so as to

choose between Bluetooth and HiperLAN2. In the next chapter we will analyze this concept of reconfigurable hardware design. Following points can be summarized based on the discussion so far.

- The channel-selection block in the Bluetooth receiver requires more computations in the datapath than the OFDM section in the HiperLAN2 receiver. However, a single computation unit of the channel-selection/FIR block (a MAC unit) is simpler than the single computation unit of the the FFT (radix-2 butterfly).
- The control structure of OFDM is more complex than the one of FIR filter. This is due to the address calculation and butterfly-structure combining needed in each stage of the FFT.
- The FFT computation requires more memory-datapath bandwidth than the FIR section due to larger number of operands in a single butterfly.
- The datapath computations in both receivers require multiply-and-accumulate (MAC) unit in hardware.
- The FFT requires extra memory resources compared to the FIR filter to store twiddle data.
- The FIR filter operates on a single input sample while the FFT operates on a block of N samples. So, Bluetooth demodulation is stream-based while HiperLAN2 is block-based.

5

Reconfigurable Architectures - A survey

5.1 A quick glance so far

The SDR project at the UT aims to combine Bluetooth and HiperLAN2 on one common platform. In the previous chapters we have already discussed the basic building blocks, for baseband demodulation, of the two receivers. The data flow for each receiver was also part of that discussion. Our discussion, so far, was limited to the analysis of the algorithms. These algorithms help us in determining the complexity of major building blocks in the SDR receiver. The motivation for this MSc. project was to explore a reconfigurable architecture for a subset of SDR algorithms. Now, we are moving our attention towards hardware mapping of these chosen algorithms.

This chapter begins with the basics of reconfigurable hardware architectures for DSP algorithms. Section 5.4 elaborates on some of the contemporary design projects. Section 5.5 provides the architectural considerations for DSP design. Section 5.6 compares the various architecture-design approaches. Section 5.7 concludes this chapter.

5.2 Design spectrum

The conventional design spectrum of data processing elements ranges from usage of general purpose processors (GPPs) to application specific integrated circuits (ASICs). Fully programmable architectures, like GPPs, can be used to compute virtually any algorithm. In classical system design, GPPs are used for computational purposes. The performance of a GPP was defined in terms of its clock frequency. These GPPs occupy a substantial amount of die area and are far less energy efficient than custom application-specific devices. The cause of this inefficiency is the manner in which flexibility is achieved in conventional processors. Computations are performed on general-purpose

functional units that are designed to implement a wide variety of arithmetic and logic functions. As a result, these functional units are large and complex, and their granularity is not always well-matched with the data types and the computations required by target algorithms. Data operands are stored in general-purpose memory units that are large, centralized structures. The tasks performed by these hardware resources during every execution cycle are specified by a stream of instructions that must be fetched from the instruction memory and then decoded and dispatched by the instruction controller. The net result is that a great deal of energy and timing overhead is attached to every basic computational step. This kind of solution involves fixed resources and dynamic algorithms. The other extreme is ASIC based implementation to achieve low area and high energy efficiency. High performance can be achieved because an ASIC architecture is designed to exploit the parallelism along with optimization for power, speed and area. But, this implementation comes at the cost of increased effort/time for design and less flexibility. This limits the use of ASICs where requirements are dynamic changing environment or low volume or small life time of the product. These two extremes of implementation differ with respect to ease of implementation, reusability, power efficiency and flexibility. In between these two extremes there lie different possibilities of implementations (see Figure 5.1). In each of these, we exchange flexibility, design costs, area, energy, upgradability at the cost of one another.

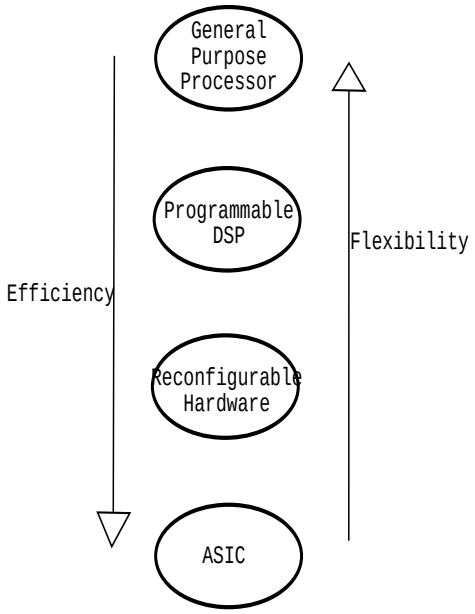

Figure 5.1: Design Domain

5.3 Reconfigurable Architectures

In recent times, the system which just computes the algorithms is of limited use. A usable system must compute the algorithms in an efficient way. The efficiency of system can be defined in terms of cost metrics. Cost metrics encompass all costs in design and production. These includes amount of hardware, design time, time to market, power consumption, and speed. This means that design should be optimum in terms of above-mentioned costs. This, for example, means a high speed system offers limited incentive, if it uses large number of hardware resources and/or uses excessive power and/or needs lot of design time.

The basic difference among modern DSP architectures lies in the amount of flexibility they offer for a given algorithm domain. Some architectures require more hardware resources and simply *map the algorithm on a powerful GPP*. This implementation, although verifies the validity of design algorithm, is hardly conceivable for a cheap mass-produced hardware. Next to this approach lies a *domain-specific-processor based design*. In this implementation, algorithms are mapped on a processor which is designed for the particular algorithmic domain. A digital signal processor (DSP) based implementation is an example of such an implementation. This allows reconfigurability but still consumes lot of power and wastes hardware resources. Additionally, it has an overhead in fetching, executing the instruction and storing the result. The parallelism of algorithm cannot be exploited fully. Another approach is to *put hardware resources in parallel* to meet the speed requirements of an algorithm. A superscalar architecture is based on this approach. Although, this increases the speed of computation and also allows flexibility to map different algorithms, this also suffers from hardware and energy wastage. Then there is the approach concerning *mapping the algorithm on FPGAs*. The biggest advantage with this approach is that functionality is defined after the fabrication, by the end user. But, the configuration is done for each bit. This means it is a very fine grained architecture and requires long time before a configuration is achieved. To overcome this, one can use *vector architectures*, in which reconfigurability is defined on a vector instead of a bit. In this approach, an architecture is defined which is reconfigurable (on vector operation) for full algorithmic domain. A field programmable function array (FPFA) is an example of such an implementation. In all these approaches, reconfiguration can be viewed as an extra layer between programming and hardware. But the major disadvantage with all these approaches is that reconfigurability is defined in a very general manner. None of these approaches can compete with state of the art dedicated ASICs in claiming hardware resources or energy requirements (see Figure 5.1). But, ASICs are non-reconfigurable and require a lot of design effort.

This thinking has led the way for reconfigurable computing in present

day integrated circuits (ICs). The idea behind reconfigurable computing is to design systems that can be flexible for their application domains only. The performance of these systems are optimized to suit the various requirements within the application-domain. This means that a reconfigurable system offers a compromise between the performance advantages of fixed-functionality hardware and the flexibility of GPPs. Like ASICs, these systems are distinguished by their ability to implement the specialized computation directly in hardware. Additionally, like a GPP based design, reconfigurable systems contain functional resources that may be easily reconfigured in response to changing parameters and data sets.

5.3.1 Domain-Specificity

An application domain determines the set of algorithms that are of of interest to the designer of a system and have similar data processing operations. One example of application domain can be DSP algorithms. It is generally observed that DSP algorithms exhibit lot of *spatial* and *temporal concurrency*. Spatial concurrency implies that multiple identical computations occur in parallel while temporal concurrency indicates the repetition of identical computations in time [26]. A regular and repetitive computation operation in an algorithm is called a *computation kernel*. When a computation kernel requires extensive computation and claims lot of resources during execution, it is called a *dominant kernel*. Algorithms belonging to the same algorithm domain have similar kernels. For example, the kernel of filter based algorithm domain can be a multiply-and-accumulate (MAC) operation. By optimizing the circuit corresponding to the dominant kernels of a domain specific architecture, high performance can be obtained. This makes domain-specific architectures both efficient and flexible within their algorithm domain.

5.3.2 Reconfigurability

Reconfigurability of a system indicates the extent of programmability incorporated into the reconfigurable system which, in turn, determines its flexibility. It can be classified in terms of *static* and *dynamic* reconfigurability. Coarsely speaking, a non-frequent reconfiguration is called static reconfiguration, while, frequent reconfiguration is known as dynamic reconfiguration. The term "frequent" in the above definition varies depending on the system requirements. An ideal dynamically reconfigurable system must be able to reconfigure without any delay. But in real-time applications, systems which consume insignificant time for reconfiguration compare to the timing budget allocated for various operations in reconfiguration, is also considered as dynamically reconfigurable system.

Reconfiguration of a system is also characterized by the overhead and

the granularity level of reconfigurable parts. For dynamic reconfiguration, it is important that the amount of configuration data required to reconfigure a chip is small. Dynamic reconfiguration allows time-sharing of hardware resources by pipelining the algorithms.

5.3.3 Granularity

Granularity of a system is determined by the width of the components in its datapath. For example, an FPGA based reconfiguration system is fine-grained system because functionality and reconfigurability is available at bit-level. Consequently, FPGA based reconfigurable systems have lot of reconfiguration overhead and are generally slow to be reconfigured. This is in contrast to a coarse-grained system, where reconfiguration is done on a collection of bits (word-level). Coarse-grained systems require less configuration data and are easy to reconfigure. But, they are not suited for complex bit-level operations.

5.3.4 Scalability

Scalability of a system is defined in terms of effort required to extend the data processing capabilities of the system. A nice approach to adopt scalability is to have *tile based systems* (see Figure 5.2). A tile can be thought of

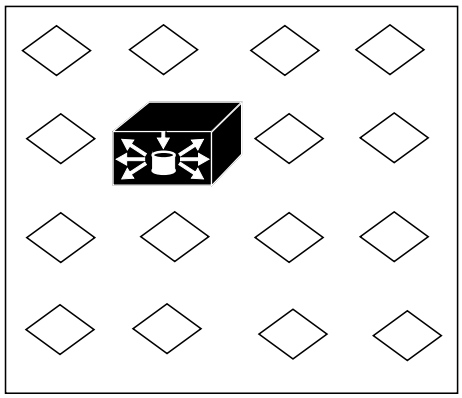

Figure 5.2: Tiled Architecture

as a basic data processor which acts as basic building block to realize the complex functionality in the system. In a tile based system, these basic data processing entities are replicated and arranged in highly regular fashion. The connection between various entities in such systems forms a network-on-chip (NOC). These tiles can be of the same kind or of different kinds. To extend the capabilities of system one can add more tiles. Another advantage of these tile based systems is the regularity of design. The regularity or parallelism in an algorithm can easily be exploited in these tile based systems. Also,

test time of the system is simplified because one needs to test the same kind of tile only once. It will be clear in the following section that latest reconfigurable DSP architectures have a highly regular organization of basic data processor(s).

5.4 Reconfigurable Architecture Examples

There are many reconfigurable architectures proposed in recent times. In this section we will discuss their approaches briefly.

5.4.1 Pleiades Architecture

The Pleiades architecture [9,10,13] relies on a heterogeneous network of processing elements, optimized for a given domain of algorithms, that can be reconfigured at runtime to execute the dominant kernels of the given domain. The Pleiades architectural template, developed at University of California at Berkeley, is a reusable architecture template for ultra low power high performance multimedia computing. It is shown in Figure 5.3). This template is

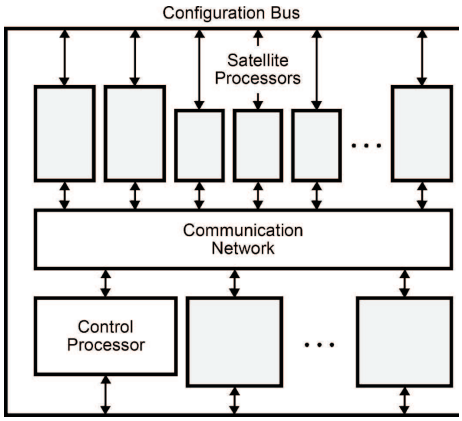

Figure 5.3: The Pleiades architecture template

reusable and can be used to create an instance of a domain-specific processor, which can then be programmed to implement a variety of algorithms within the given domain of interest. All instances of this architecture template share a fixed set of control and communication primitives. The type and number of processing elements in a given domain-specific instance, however, can vary and depend on the properties of the particular domain of interest. The architecture template consists of a control processor, a general-purpose microprocessor core, surrounded by a heterogeneous array of autonomous, special-purpose satellite processors. All processors in the system communicate over a reconfigurable communication network that can be configured to create the required communication patterns. All computation and

communication activities are coordinated via a distributed data-driven control mechanism. The dominant, energy-intensive computational kernels of a given DSP algorithm are implemented on the satellite processors as a set of independent, concurrent threads of computation. Most satellite processors are dedicated to performing specific tasks but some satellite processors might support a higher degree of flexibility to allow the implementation of a wider range of kernels. The parts of the algorithms, which are not computation intensive and tend to be control-oriented, are executed on the control processor. The computational load on the control processor is thus relatively light, as the bulk of the computational work is done by the satellite processors. In addition to executing the non-compute-intensive and control-oriented sections of a given algorithm, the control processor is responsible for spawning the dominant kernels as independent threads of computation, running on the satellite processors.

5.4.2 Montium:Coarse-Grained Reconfigurable processor

In the Chameleon project [1, 23], the Chameleon system on chip (SoC) template is proposed as a solution for the contradicting requirements of mobile handheld devices. In a Chameleon SoC template, heterogenous processing tiles are connected by a network-on-chip. In the template four processing tile types are distinguished.

- general purpose (i.e., GPPs and DSPs).
- fine-grained reconfigurable (i.e., FPGAs).
- Coarse-grained reconfigurable (i.e., DSRAs).
- Application specific (i.e., ASICs).

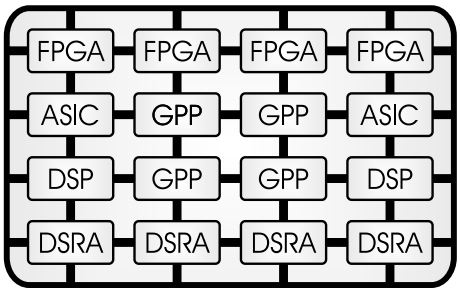

Figure 5.4: The Chameleon architecture template

An example of chameleon SoC that contains 16 processing tiles is given in Figure 5.4. At the University of Twente, a domain specific reconfigurable accelerator is designed which can be incorporated in Chameleon SoC as a

DSRA tile . This DSRA is called Montium Tile processor (TP) [24, 26, 36] and its algorithm domain consists of 16 bit DSP algorithms that contain MAC operations.

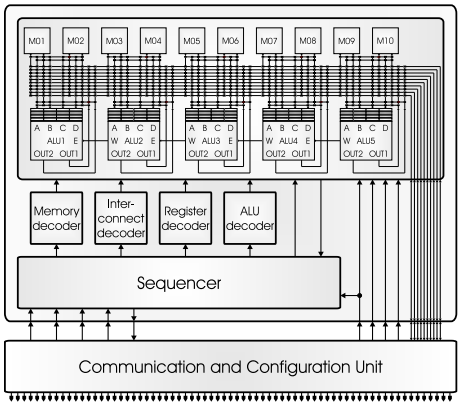

Figure 5.5: The Montium Processing Tile: A Tile Processor and a Communication and configuration Unit

In Figure 5.5, a Montium processing tile is depicted. The upper part of the Figure shows the processor part: the Montium TP. The lower part shows the NoC interface : the communication and configuration unit (CCU). The TP acts as DSRA and CCU acts as interface with the world outside the processing tile. The Montium TP is controlled by a sequencer. The sequencer selects instructions from configurable decoders. The part of the Montium TP that is responsible for the datapath processing is called the processing part array (PPA). The PPA has a regular VLIW like architecture.

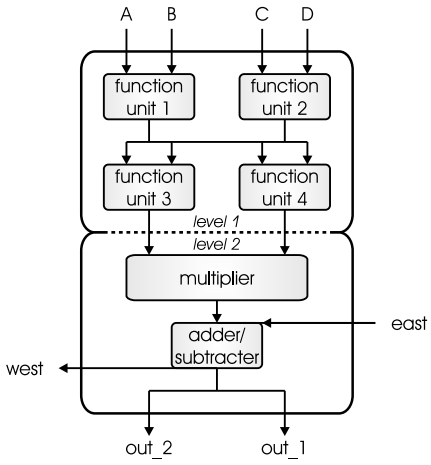

Figure 5.6: Montium Arithmetic and Logic unit

The five ALUs (ALU1-ALU5) in a PPA (Figure 5.5) can exploit spatial concurrency to enhance performance. A vertical segment that contains one ALU together with its associated input register files, a part of interconnect and two local memories is called a processing part (PP). The ALU in each PP is tailored to DSP algorithms. A simplified schematic of a Montium ALU is depicted in Figure 5.6. Each input of an ALU has its own local storage in the form of a register file. In addition, each PP also has two local storage memories. The upper level (level 1) in the ALU contains four function units for general arithmetic and logic operations. The lower level (level 2) contains the MAC units.

5.4.3 PACT's extreme processor platform (XPP)

An XPP processor [18, 40, 44], developed by PACT XPP Technologies, is built from a coarse-grained homogenous array of reconfigurable ALU processing elements, RAMs and communication channels. It uses only a handful of different functional blocks: ALU processing-array-elements (PAEs) perform the basic calculations, RAM together with an ALU for address calculation, and I/O objects that provide access to external streaming channels and external RAMs. All these elements are integrated with the communication channels of the array, providing point-to-point connections with data handshaking. Figure 5.7 shows an XPP with two identical processing array clusters.

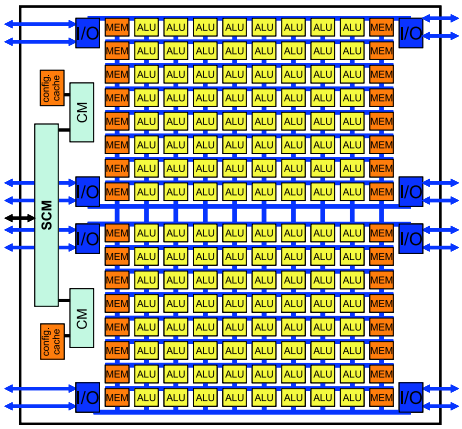

Figure 5.7: The XPP containing two identical processing array clusters

The array elements can be configured to execute their operations when triggered by an event signal indicating that new data is available at the input ports. A new output can be produced every clock cycle and the result constitutes a data output and an event signal indicating that data is ready on the output port. The ALU-PAE comprises a data path, with two inputs and two outputs, and two vertical routing resources. The vertical routing

resources can also perform some arithmetic and control operations. One of the two is used for forward routing and the other for backward routing. The forward routing resource (FREG) is, besides for routing, also used for control operations such as merging or swapping two data streams. The backward routing resource (BREG) can be used both for routing and for some simple arithmetic operation between the two inputs. There are also additional routing resources for event signals which can be used to control PAE execution. The RAM-PAE is exactly the same as the ALU-PAE except that the datapath is replaced by a static RAM. The RAM-PAE can be configured to act either as a dual-ported memory or as a FIFO.

The configuration manager (CM), a small microcontroller block, configures the function of the ALUs and the connections between them. Various CMs are organized in a hierarchical tree, enabling the concurrent configuration of processing array clusters.

5.4.4 Adaptive System-on-a-Chip (aSoC)

Adaptive System-On-a-Chip (aSOC) [12, 28, 29], is a modular communications architecture developed at University of Massachusetts (UMASS). aSOC is primarily an interconnect architecture, based on static scheduling of virtual interconnects onto a highly characterized and regular physical interconnect fabric. The basis for this architecture is that on chip coordination of communication between cores in data intensive applications can be predicted on per application basis and can be statistically scheduled. This approach emphasized hardware minimization and interconnect performance at the cost of some flexibility.

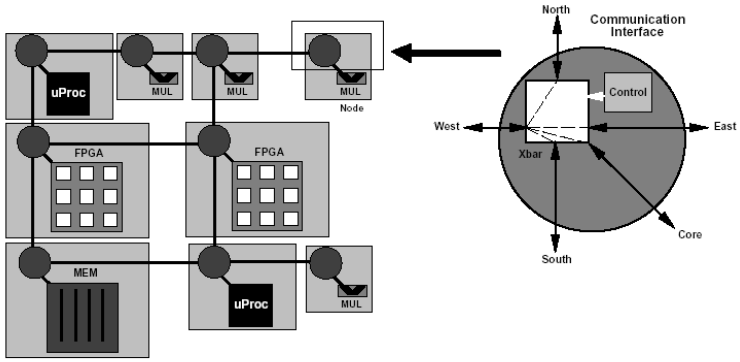

Figure 5.8: Adaptive System on-a-Chip (aSOC)

As shown in Figure 5.8, an aSOC device contains a two-dimensional mesh of computational tiles. Each tile consists of a core and an associated communication interface. The interface design can be customized based on core datawidths and operating frequencies to allow for efficient use of

resources. The core interface manages communications through each tile and synchronizes global communications. Communication between nodes takes place via pipelined, point-to-point connections. By limiting inter-core communication to short wires with predictable performance, high-speed communication can be achieved.

5.4.5 Quicksilver's adaptive computing machine (ACM)

Quicksilvers's ACM technology [25, 35] is based on the development of heterogenous systems on chip. The ACM is essentially a collection of adaptive heterogeneous algorithmic engines, called nodes, which are connected via an adaptable network. The structure of the ACM is completely scalable. An embedded controller configures a network of nodes in such a way that they represent a data flow graph instantiated in hardware. This configuration can be changed every clock cycle, if need be, at minimal cost. Figure 5.9 illustrates the two basic components of the Adapt2400 ACM architecture.

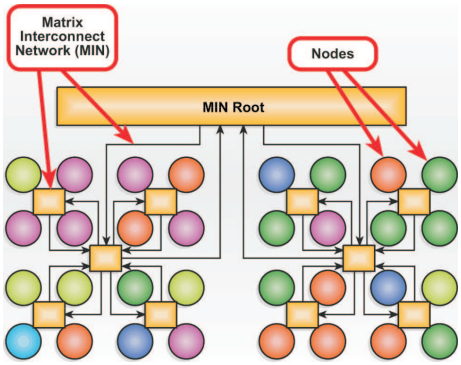

Figure 5.9: ACM architecture

There are two basic components in the ACM architecture.

1. Nodes: Nodes are the computing resources in the ACM architecture that perform the actual work. Nodes are heterogeneous by design, each being optimized for a given class of problems. Each node is self-contained with its own controller, memory, and computational resources. As such, a node is capable of independently executing algorithms that are downloaded in the form of binary files, known as Silverware

2. Matrix Interconnect Network (MIN): Tying the heterogeneous nodes together, the MIN is a homogeneous network that carries data, Silverware, and control information between ACM nodes, as well as between nodes and the outside world. This network is hierarchical in structure, providing high bandwidth between adjacent nodes for close

> coupling of related algorithms, while facilitating easy scaling of the ACM at low silicon overhead. Each connection between blocks within the MIN structure simultaneously supports 32 bits of data payload in each direction. Data within the MIN is transported in single 32-bit word packets, with addressing carried separately. Each 32-bit transfer within the MIN can be routed to any other node or external interface, with the MIN bandwidth fully shared between all the nodes in the system.

ACM nodes are configured/programmed using a binary file called SilverWare, which is much smaller than that of a typical FPGA configuration file, and is comparable to the program size of a DSP or RISC processor. The smaller binary file size, combined with hardware specifically designed to adapt on the fly, allows the function of a node to change in as little as a few clock cycles. Nodes are constructed of three basic components: The node Wrapper, nodal memory, and the algorithmic engine. The node Wrapper has two major functions: 1) To provide a common interface to the MIN for the heterogeneous Algorithmic Engines; 2) To make available a common set of services associated with inter-node communication and task management. Each node is nominally equipped with 16 kilobytes of nodal memory organized as four 1k x 32 bit blocks. Each heterogeneous node type is distinguished by its algorithmic engine. The computational resources of each node type is closely matched and optimized to satisfy a finite range of algorithms at a specific performance/price/power consumption level demanded by image processing and communications systems.

The MIN also differs from the interconnects of conventional reconfigurable IC designs in that the concept of dedicated wires does not exist. Each word of data to be transferred between nodes is routed individually, on a clock-cycle-by-clock-cycle basis.

5.4.6 Reconfigurable Communications Processor (RCP)

Chameleon System's RCP [26] was intended for wireless base stations. It addresses not only the computational demands but also programmability and run time adaptability. It is a SOC that includes a 32-bit arithmetic reconfigurable fabric (RPF), a 32-bit GPP and programmable I/O. The main Architectural blocks of RCP are shown in Figure5.10.

The interface between the main blocks of the system is the 128 bit 'Road Runner' bus. For off-chip communication, it incorporates a PCI bus, a 64-bit memory bus and 160 pins for programmable I/O. The RPF consists of data processing units, local storage and interconnect structure. It has a two-dimensional array of identical processing tile as shown in Figure 5.11. A slice is the basic unit of reconfiguration and can be reconfigured independently of other slices. The RCP configuration system consists of a configuration

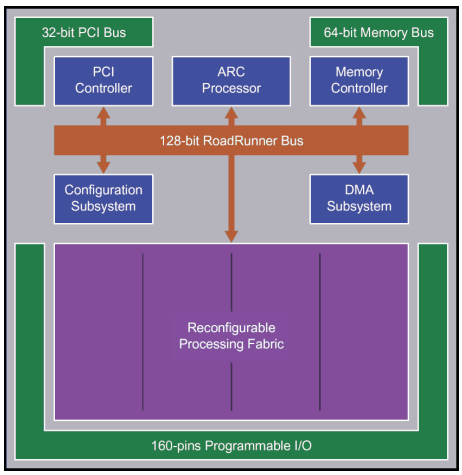

Figure 5.10: RCP architecture

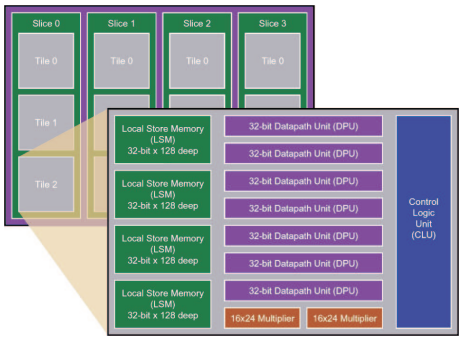

Figure 5.11: Reconfigurable processing fabric and tile architecture

controller and two configuration planes. The active configuration controls the RPF and the background plane can hold another configuration. The RPF can be configured in one clock cycle by switching the background and the active plane.

5.4.7 Universal Communications Coprocessor (UCC)

The Universal Communications Coprocessor (UCC) [14] has been developed specifically for use in SoC designs where SDR functions are required. It contains configurable signal processing blocks which perform functions that are common to the majority of radio standards, such as frequency correction, sample rate conversion, filtering and error correction. It uses a programmable SIMD processor with an instruction set tailored to complex vector processing for demodulation and modulation and behaves as an intelligent peripheral, communicating with the host processor using interrupts and mailboxes. This approach allows the host processor to interact with the

UCC in a flexible manner, without imposing a heavy load on the host.

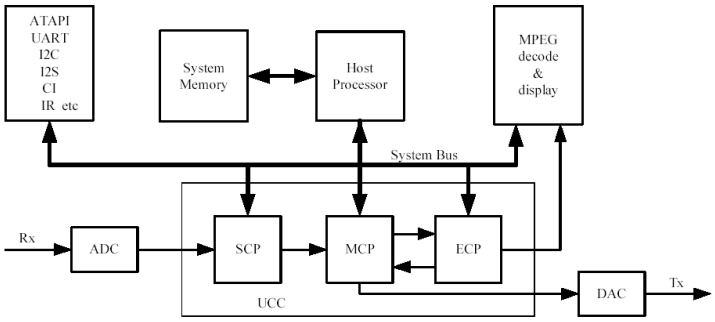

Figure 5.12: A SoC design incorporating the UCC

The UCC can handle real-time operations with very low response latency, and can communicate with the host processor in a relaxed fashion by exchanging messages on a schedule defined by the host.An example of a SoC design that uses the UCC is shown in Figure 5.12. The UCC contains three processing blocks, each with processing capabilities appropriate to a set of SDR tasks. The Signal Conditioning processor (SCP) performs quadrature mixing, digital resampling, interference rejection filtering and decimation. The Modulation and Coding processor (MCP) is a programmable processor with an instruction set optimized to the processing of vectors of complex data. The Error Correction processor (ECP) is to support the channel coding schemes used in many standards.

5.4.8 Dynamically Reconfigurable Architecture (DReAM)

DReAM [11], developed at Darmstadt University of Technology, consists of an array of concurrently operating coarse-grained Reconfigurable Processing Units (RPUs). Each RPU is designed for executing all required arithmetic data manipulations for the data-flow oriented mobile application parts, as well as to support necessary control-flow oriented operations. The complete DReAM array architecture is shown in Figure 5.13. It connects all RPUs with reconfigurable local and global communication structures. In addition, the architecture provides efficient and fast dynamic reconfiguration possibilities for the RPUs as well as for the interconnection structures, e.g. only partly and during run-time while other parts of the reconfigurable architecture are active. This architecture consists of a scalable array of RPUs that have 16-bit fast direct local connections between neighboring RPUs, whereas each subarray of four RPUs shares one common Configuration Memory Unit (CMU). The CMU holds configuration data for performing fast dynamic reconfiguration for each of these four RPUs and is controlled by one responsible Communication Switching Unit (CSU). Each RPU consists of two dynami-

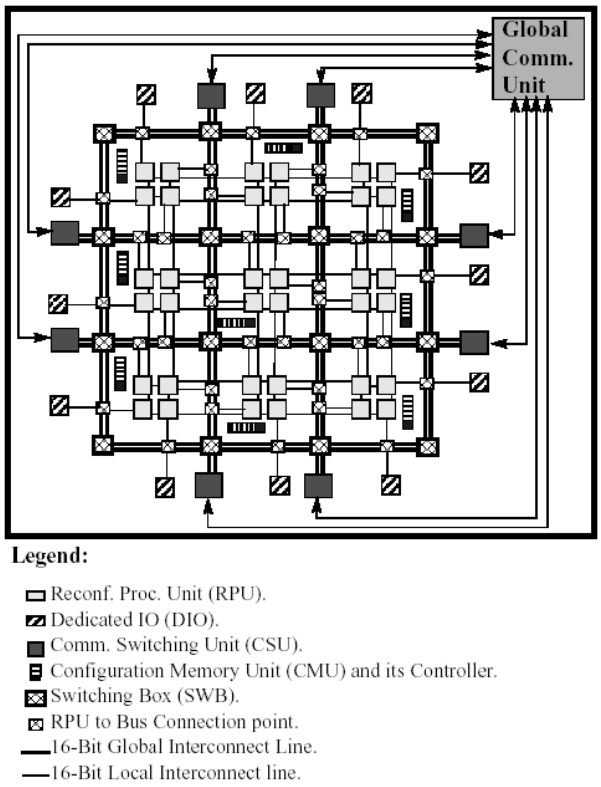

Figure 5.13: Hardware Structure of the DReAM Architecture

cally reconfigurable 8-bit data paths, (Reconfigurable Arithmetic Processing Units, RAPs), one Spreading Data Path (SDP), one RPU-controller, two dual port RAMs, and one Communication Protocol Controller.

5.4.9 RAW Processor

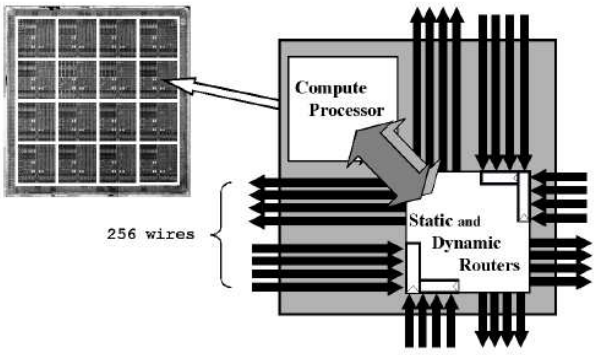

Figure 5.14: Raw microprocessor die photo and tile diagram

RAW Processor architecture [20, 31], developed at MIT, is in principle a chip multiprocessor, constituting an array of 16 full scale MIPS-like RISC processors called tiles (see Figure 5.14). Large portions of a RAW tile constitute instruction fetch and decode units, a floating point unit and units for packet oriented interconnection routing. The tiles are connected by four 32-bit point-to-point, on-chip, pipelined, mesh interconnection networks: two static and two dynamic. To reduce power consumption in the Raw processor, a toggle-suppression strategy is followed. This ensures that wires in the system do not toggle unless they are actually computing something useful. Characteristic for RAW is the high I/O bandwidth distributed around the chip edges and the dynamic and static networks that tightly interconnect the processors, directly accessible through read and write operations in the register file of the processors. The Raw I/O port is a high-speed, simple (a three-way multiplexed I/O port has 32 data and five control pins for each direction), and flexible word-oriented abstraction that lets system designers proportion the quantities of I/O devices according to the application domains needs. Memory intensive domains can have up to 14 dedicated interfaces to DRAM. Other applications may not have external memory : a single ROM hooked up to any I/O port is sufficient to boot Raw so that it can execute out of the on-chip memory. In addition to transferring data directly to the tiles, off-chip devices connected to Raw I/O ports can route through the on-chip networks to other devices to perform direct memory accesses (DMAs).

5.4.10 A Medium-grain Reconfigurable Cell Array

This architecture [30] tries to bridge the gap between fine-grain and coarse-grain reconfigurable devices. In this approach, the device contains an a 44 matrix of reconfigurable elements (see Figure 5.15), each of which handles a small portion of the overall operation. Users can tailor the device to the processing task at hand by controlling the word length, number of parallel functional units, and functional unit connectivity. More generally, exposing these hardware resources to software management allows for more efficient parallelism via the tradeoff of temporal and spatial utilization of the device. Sixteen 4-bit busses connect each cell to its eight neighbors.

Each cell contains four main components : The processing core performs the 4-bit operations necessary for DSP computations; The switch routes data between the cell and its neighbors; The interface contains buffers and pipeline latches to improve performance; Finally, the control circuitry buffers the global clock signal and manages the reconfiguration process.

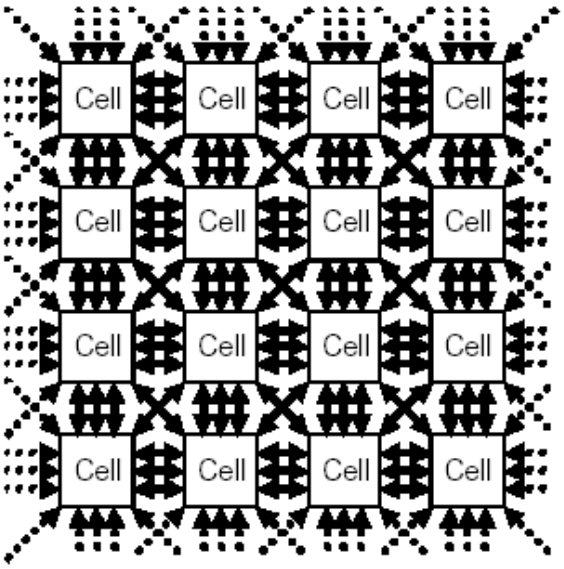

Figure 5.15: Portion of reconfigurable cell array

5.5 Architectural considerations for DSP design

From the discussion in the beginning of this chapter, it is clear that the ongoing trend of enriching more and more functionality in the hardware requires novel architectures in the present day SOC designs. Lot of work is currently being done to meet the conflicting demands of flexibility and efficiency in real time DSP applications. Too much flexibility leads to a larger chip area, whereas, too little flexibility limits the algorithm domain. For proper design choices one need to consider area, performance, power figures, design efforts and various other factors like scalability, design automation etc.

In the architectures mentioned above, more flexible solutions than ASICs and more efficient solutions than GPPs are sought for. The common design approach is to identify domains of applications and design an architecture that supports this entire domain rather than going for the acceleration of one specific calculation. This enables building domain specific architectures rather than application specific architectures. Further, by the use of reconfigurable hardware it is possible to accelerate larger parts of the algorithms than with a fixed computer architecture.

There are lot of projects going on or has been done recently in the industry and academics to explore the various possibilities for reconfigurable DSP hardware. A single reconfigurable processing element, called tile, is the minimal processing entity in these projects. These designs incorporate parallel, reconfigurable array structures consisting of homogeneous or

heterogenous tiles connected in an optimal way to obtain flexibility and performance. Various points that can be learnt from above mentioned projects are enlisted below. These points are starting point of our architecture design (next chapter).

- Architectures that target a smaller set of applications can be more efficient than general-purpose devices and must be pursued. In the above architectures, flexibility is recommended to the extent that is needed in these designs, unlike GPPs.
- Most of the above approaches recommend coarse-grained datapath design and discard the fine grained design approach (as in FPGAs), because of excessive programming overhead.
- All of the above mentioned systems are tile-based systems. This allows easy scalability and upgradability, datapath optimization and testing.
- The interconnection network that links the memories and the processing elements must support high data rates and must be flexible enough to support the required communication patterns that are commonly seen in DSP kernels.
- Communication between various units should be optimized and must be reconfigurable.
- The control structure that is used to coordinate computational activities within multiple parallel processors and memories must be efficient and scalable.
- Some algorithms can be mapped on hardware in one-to-one manner. Some other algorithms can be optimized further for easy implementation. This may involve retiming, pipelining and other optimization tricks for optimal use of hardware resources.
- Unwanted glitches wastes lot of energy. So, register the inputs, if possible.

5.6 Comparison of different approaches

The Data processing elements in the architectures mentioned above span a quite wide spectrum from pipelined full scale processors [20, 35] to simple ALUs [40]. Accordingly, these coarse-grained architectures paradigms range from one end with chip multiprocessor with dedicated control and program sequencing circuitry (the RAW processor developed at MIT) to the other end with an ALU array (the XPP processor from PACT XPP Technologies) with resources only for pure dataflow processing.

An implementation study with respect to processing efficiency in FFT computation has been done in the [17] on these two extreme ends of coarse-grained paradigms. The implementation study shows that a considerable part of the resources are used for implementing the control structures in the FFT implementation. Computations shows that using about 15 percent of the peak performance, for computations in an application, would still mean that the ALU array is competitive compared to the more traditional chip multiprocessor architecture. Theoretically, this means that up to 85 percent of the ALU resources, can be used to implement control functions for complex algorithms, still the ALU array could be more effective than a chip multiprocessor, when comparing performance-area. It can easily be concluded that the peak performance of XPP is much higher than RAW, but it is difficult to program XPP and most of the time full processing power cannot be completely realized in the implementation. The results of implementation study show that although, the ALU-array is more efficient than chip multiprocessor, still the control operations occupies about 75-percent of total processing power. So, only 25-percent of processing power is used for actual computations.

5.7 Conclusion

In this chapter, we have elaborated the need for domain specific processors for DSP applications. These processors improve the performance of SOC designs compare to GPP based systems. But, as discussed in the previous section, these architectures have their own limitations because these designs are always a trade-off between performance and flexibility. For real-time high computation applications, there is still a big question on the performance of domain specific processors. In fact, when we consider the performance requirements of our SDR system and the performance of recently designed Montiun TP based system, we can easily conclude that one Montium TP based system will not be able to meet the data processing demands. This point will become more clear later on in chapter 8. So, we need multiple TPs in the system. This not only requires extra resources but also wastes lot of energy due to extra peripheral communication between two any two processors.

This thesis proposes an ASIC-like reconfigurable design for the chosen set of SDR algorithms. This design is explained in the next chapter.

6

Architecture Design

The study done in the previous chapter has motivated us to have an ALU-array like architecture instead of a simple multiprocessor architecture for our implementation. The reason being the ALU- array makes more efficient use of the silicon area than a multi processor [17]. Still, this means that lot of computational resources will be wasted in the control elements, if we try to implement control on the tiles from XPP like architectures as is done in the afore-mentioned implementation study. The main reason for this wastage lies in the fact that control operations are generally quite different from data processing operations. So, a normal ALU-like element designed for data processing cannot perform the computations corresponding to control operations very efficiently. To overcome this bottleneck, we propose to keep the data processing ALU array-like architecture for data processing parts of algorithms and design separate hardware for control operations. In this way, control operations are implemented on the hardware that is optimally designed for it. This should improve the computational efficiency of the system. In a way, our approach is to separate the data processing operations and control operations of algorithm domain first and then design the hardware separately. This is due to the fact that most of the DSP algorithms can be characterized by highly parallel dataflow and repeated calculations. Thus, it should be possible to allocate most of the hardware resources to the parallel datapath and only minor parts to the control of the computations. Since the control steps of our algorithm can easily be described in a state-machine, so the hardware corresponding to them can easily be designed in a state machine following the predetermined schedule of operations.

6.1 Design approach

In this thesis, we propose a solution which is optimized for our specific algorithmic domain. Our algorithm domain is limited to the DSP algorithms for each stage of SDR receiver. In the proposed architecture, the basic approach

is to limit the flexibility of design to the algorithms of interest (OFDM and channel-selection). This limited flexibility requirement will result in only moderate-degradation of the ASIC performance. This is in contrast to various designs discussed in previous chapter, where the approach is to incorporate the sufficient flexibility to support the application domain. So, our approach is to design a flexible ASIC-like system for specific algorithms only. Our design approach has four main steps.

1. In the first step, we are identifying the dominant kernels of our algorithm domain. This step is similar to any domain-specific design mentioned previously and requires careful reviewing of the tailored application's area requirements.

2. In the second step, we have designed the optimal control hardware for our algorithm domain. This is in contrary to various hardware- design approaches mentioned in the previous chapter because all those approaches put their attention towards the dominant data processing operations only. But as indicated above, lot of resources are wasted, if we used the normal data processing elements for control operations. One can argue to use a GPP for control operations, but this would again mean one will use hardware not tailored to perform the desired operation and also waste extra resources, due to the GPP structure. This will also mean energy wastage and other disadvantages corresponding to instruction-fetch, decode and execute in the control part of the system. Our algorithm domain was limited to HiperLAN2 and Bluetooth only. So, a simple state-machine description of control has been implemented. The scalability of this control is limited because of non-programmability. But, the control scheme of the overall system is scalable in the sense that one can always add extra state machines or control hardware in the hardware template for schedules that are not implemented or addressed in the current schemes. Additionally, one can add limited programmability, if the flexibility of control is the major concern. This approach is valid, if algorithms are more or less identified. But, if we want to keep changing the algorithms, the additional control hardware for each algorithm will make the above scheme quiet inefficient. Then one should implement the control using separate control processor(s) as is done in [9, 26]. This is especially important if design is in the feasibility-analysis phase.

3. In the third step, we have identified the communication patterns in our algorithm domain as recommended in the [29]. This has helped us in designing the optimal communication network in the system. Only those parts of communication are programmable which are really needed. As far as possible, global busses are minimized to reduce

capacitance and cross-talk effects. So, point-to-point and local communication is preferred in our the proposed architecture.

4. In the fourth step, we have identified the memory requirements for our systems. In this step, we have identified things like how much RAM, ROM memory are needed, what are the memory bandwidth requirements, is it better to reuse the memory by using in-place computations etc.

It must be emphasized that all the above steps are interlinked and final outcome is achieved by iteratively following these four steps. In these iterations, various steps may or may not be executed in the same order. The main components of the proposed design are detailed in this chapter. In later chapters we will evaluate the performance of our system.

The proposed architecture is a coarse-grained architecture. Basically, we have identified the parts with similar complexity (in both receivers) and designed an architecture for them. Also, we have configured the dataflow among these parts to match the receiver functionality. The proposed architecture comprises of nine homogenous data-processing tiles , two 128x16 bit memory (RAM) tiles, one 64x16 bit ROM, a configuration unit to configure the data and communication network, and a control section in the form of a state machine to execute algorithm steps sequentially. The control section also controls the data transfer from datapath elements to memories through the communication network. It also generates the control signals for the configuration unit. The architecture view[1,2] of the system is shown in Figure 6.1. Detailed architecture is shown in Figure A.1. Main components of the proposed architecture are discussed further in this chapter.

6.2 Granularity

The proposed architecture is a coarse-grained architecture. The datapath is 16-bit wide. This bit-width is calculated based on the SNR required in our algorithms [27]. Data is taken in (16,10) fixed-point format. Simulation results shows that overflow and quantization errors are within tolerable limits.(see appendix-B).

6.3 Scalability

The proposed design is based on tiled-architecture approach. A tiled architecture in which various tiles are connected by an on-chip network has a very modular design. The design of a single processing tiles is relatively

[1] Actual datapath contains nine processing elements, only 8 are shown in Figure 6.1
[2] CU denotes configuration unit

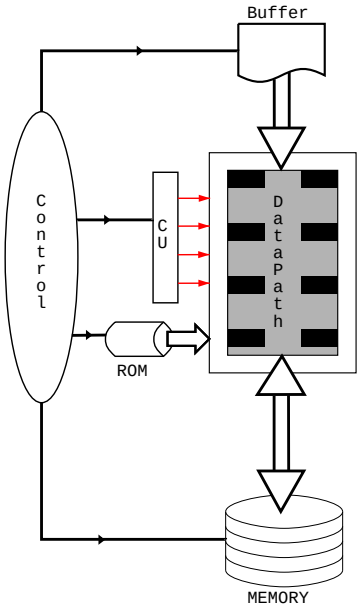

Figure 6.1: Architecture

simple and allows extra effort for power optimizations at physical level. To increase or decrease the processing power of our system, we can easily add or remove tiles. A simplified view of our tiled network is shown in Figure 6.2.

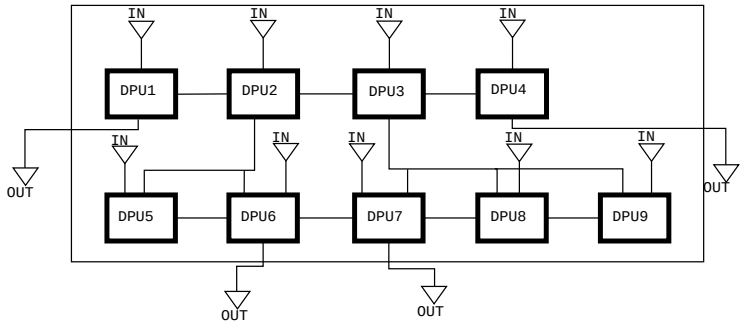

Figure 6.2: Tiled architecture

6.4 Reconfigurability

The proposed design is reconfigurable within one clock cycle and supports the chosen subset of the SDR algorithms [39]. So, the algorithm domain of our design includes FIR filter, half-band filters and radix2-FFT. These algorithms are also the most common algorithms used to benchmark a DSP

system [10, 15]. The dynamic reconfigurability allows time-sharing of hardware resources by pipelining the algorithms. This minimizes the total hardware resources required to implement the complete system. Also, almost all of the WLAN systems use either phase-modulation or OFDM based modulation. So, the suitability of our system for phase-modulated and OFDM based receivers implies that our design can be used in number of WLAN systems.

6.5 Datapath

In the proposed design, datapath consists of 9 homogenous 16-bit data processing tiles called data processing units (DPUs). The detailed view of our datapath is shown in Figure A.2. A single DPU is depicted in Figure 6.3. The design of a DPU can be divided into four parts: The processing part, the storage part, the configuration part and the communication interface. These parts are shown as arithmetic unit, registers, configuration part and various input/output ports, respectively, in Figure 6.3. .

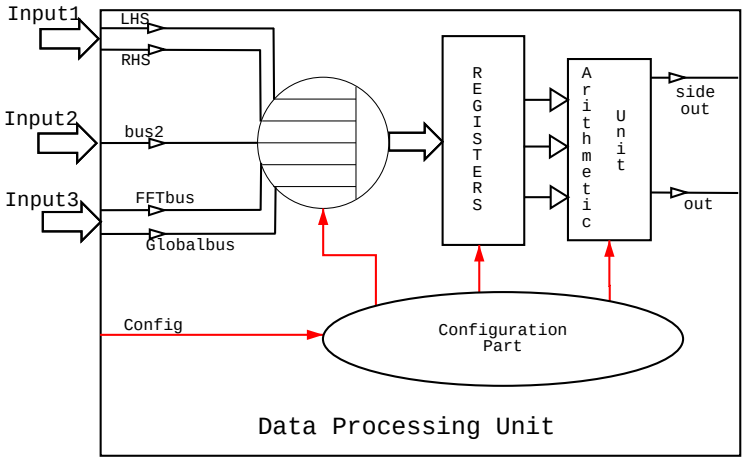

Figure 6.3: A Data processing unit (DPU)

6.5.1 The communication interface

The communication interface of each DPU supports the use of heterogenous processing occupying one or more tiles. This interface manages the communication through each tile and synchronizes the global communication. Each DPU has 3 sets of 16-bit inputs.

- Input1-set is used to read data either from left or from right neighbor into the registers. The ports corresponding to these inputs are named as 'LHS' and 'RHS'.

- Input2-set (bus2) is used to read data from globalbus of the system. There are two global busses in our system. Each global bus is providing the input to one row of DPUs.
- Input3-set is connected to two point-to-point buses of the system. The ports corresponding to these inputs are named as 'FFTbus' and 'Globalbus'.

Each DPU has two 16 bit outputs.

- First output ('sideout') is used to communicate with the adjacent left and right side neighbors.
- Second output ('out') is used to communicate data over the system communication buses. To avoid bus arbitration output 'out' is a tristate output.

6.5.2 The processing part

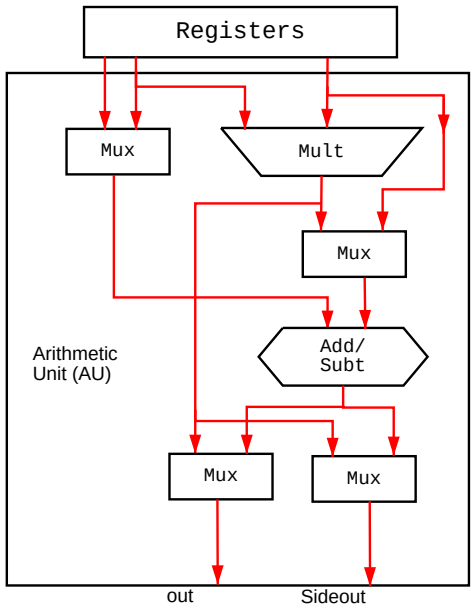

Figure 6.4: Arithmetic unit (AU) of DPU

The data processing capabilities of DPU are attributed to a 16-bit arithmetic unit (AU). A functional representation of the AU is shown in Figure 6.4. An AU is purely combinational and is capable of doing the basic 16-bit arithmetic operations namely add, subtract, multiply, multiply and add, and multiply and subtract. The input to AU is from internal registers and outputs are provided on the output ports.

6.5.3 The storage part

Each DPU comprise of a set of 11 local data registers of 16 bit each. These registers can be used to store intermediate data variables as required in FIR data structure. This way of having local registers is far more efficient than one centralized set of registers [15]. These registers are used to read data from input ports and to provide data to ALU. In this way, inputs are always registered, thus minimizing the excessive glitches. Another reason for having registered inputs is to allow pipelining between various datapath units. This not only allows the reduction of critical path delay, but also allows a straightforward implementation of transposed form FIRs.

6.5.4 The configuration part

Each DPU has a local configuration section called "configuration part", which provides the configuration signals to various entities within the DPU. This configuration section is part of the control hierarchy of the system to reduce the control overhead significantly [26]. The input to this section comes from the main configuration unit of the architecture.

6.6 Control section

In the proposed architecture, the control section is implemented as a state machine corresponding to each algorithm. This is motivated by the fact that data flow is determined at the design time itself. In the normal operation, the control system loops through the set of algorithms steps called a schedule. To compute an algorithm, first the control section is activated with the corresponding wake-up call. The control section responds by generating the series of control signals to memory and to the configuration part, thus controlling the data-operations in the system. In this way, we avoid the common bottleneck (corresponding to fetch and decode an instruction before execution) find in normal processor like architecture. This scheme has obvious disadvantage that each new algorithm needs to be implemented separately. So, if algorithm is subject to change, one should incorporate the programming facility in the control.

6.7 Configuration unit

In the proposed architecture reconfigurability is achieved by reconfiguration of the datapath and reconfiguration of the communication network. These configuration signals are generated in the Configuration unit (CU). The input of the CU comes from control section in the form of control signals. The CU decodes these control signals and provide input to local configuration

sections of various DPUs. The configuration of the datapath and communication network is achieved within one clock cycle. This allows dynamic and static reconfigurations in the proposed architecture. To compute an algorithm, first step is to activate the centralized control section. This control section then activates the CU on a per clock cycle basis. The CU provides the input to local configuration of each DPU. Each local configuration part responds by configuring the corresponding subsection of datapath. This way distributed control is achieved in the proposed architecture. This is shown in Figure 6.5. This facilitates high operating speeds and time shar-

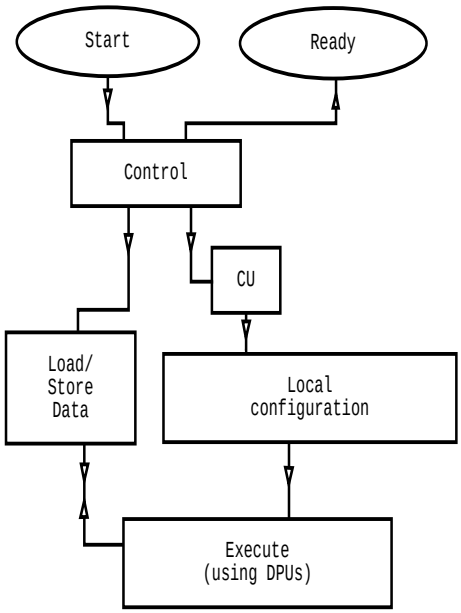

Figure 6.5: Control Scheme

ing of data and communication network. The low-overhead and dynamic reconfiguration allows time multiplexing of the processing part.

6.8 Communication network

The communication network consists of two parts: communication within each DPU and communication between all the entities (DPUs, memory, CU, control). The communication within each DPU and the communication of each DPU (via input and output ports) with all other entities in the system is controlled by the local configuration section of DPU. The collective communication interface of all DPUs makes the data transfer part of communication network. This communication network is designed based on the communication pattern of the algorithms of interest. Dynamic determination of data-source and data-destinations at run-time is not supported. In

the proposed architecture, the system connects various tiles , via data and control busses, based on the predictable communication pattern of algorithm. The number of data-busses are optimized to reduce the capacitive load. Various registers at each input allow data between neighboring DPUs to move in a communication pipeline as shown in Figure 6.6. This facilitates high operating speeds and time sharing of data and communication network [29]. The Communication network showing communication between all the entities of the system can be seen in Figure A.1 and Figure A.2.

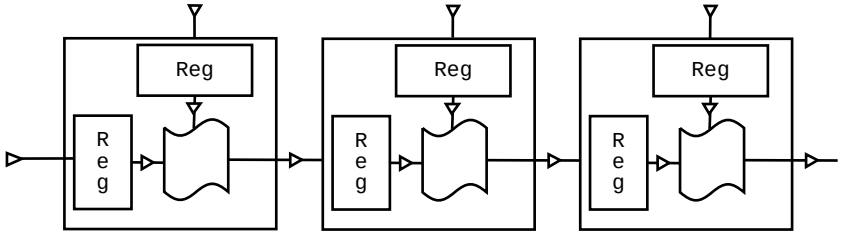

Figure 6.6: Communication Pipeline

6.9 Conclusion and Summary

In this chapter, we have elaborated the design of our reconfigurable architecture. In the next chapter, we will show how the proposed architecture can be used to implement the chosen subset of SDR algorithms. The main features of our implementation are given below. Some of these points will become more clear after reading chapter 7.

- The proposed architecture is a regular tile-based system for easy scalability and testability.
- Our architecture is statically and dynamically reconfigurable. Reconfiguration is achieved within one clock cycle. So, the datapath does not remain idle at any moment of time during computations.
- Basic DSP algorithms such as FFT and FIR, which are part of our SDR project, can easily be mapped onto it. These algorithms are the two most important algorithms for benchmarking DSP systems.
- Local communication between tiles is limited to left and right neighbors.
- I/O Bandwidth is optimized to meet the demands of the chosen SDR algorithms.
- Datapath width is fixed to 16 bits.

- All data inputs are registered to avoid glitches.
- All DPUs are autonomous. The structure of DPU incorporates functionality and performance for SDR algorithms. To save energy, it is possible to switch-off tiles that are not used.
- Vertical programming in the the configuration path reduces the control signals from control to datapath.
- Control is designed as separate hardware in the form of state-machine.
- Time-multiplexing can be used easily to reduce hardware and improve performance.
- On-chip communication resources are allocated just to suit our applications (as recommended in [29]).

7

Algorithm Mapping

The proposed architecture (see chapter 6) is designed to implement a chosen subset of SDR algorithms (see chapter 3).

In this chapter, the mapping of these algorithms on the proposed architecture is discussed. This includes the elaboration of the data connectivity, allocation and the scheduling of our design. The performance of these computations will be evaluated in the next chapter. For explanation purposes, the DPUs are numbered from 1 to 9.

7.1 Mapping of a half-band filter

Incoming data for this block is coming in two separate streams. One stream is corresponding to the real part of the data and the other stream is corresponding to the imaginary part of the data. The filter coefficients and the computations on real and imaginary data are the same. We allocate DPU1-DPU4 for the real data and DPU6-DPU9 for the imaginary data. DPU5 is put to sleep mode. This is shown in Figure 7.1. Each half-band

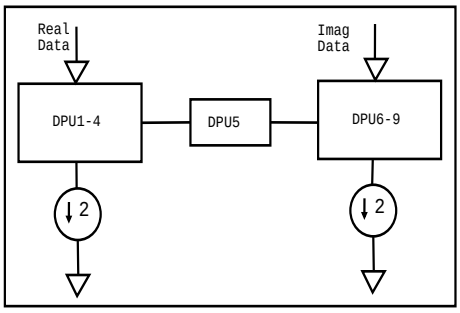

Figure 7.1: DPU allocation scheme for Real and Imaginary Data

filter is of 7th order. So, we need 7 basic computations equivalent to a MAC operation (shown in Figure 4.6). But, we have allocated only four DPUs for

each filter. This means that we will need two clock cycles to filter one data sample. The following steps are performed on DPU1-DPU4 during these two clock cycles to perform the computation for one sample of the real data stream (Imaginary data is calculated in similar way on DPU6-DPU9).

- Load the real data sample from memory into the globalbus connecting DPU1-DPU4.
- Each AU is configured for muliply-and-add.
- Read data from global bus input into a data register.
- Read intermediate data value from LHS input into a data register.
- Configure multiplier inputs of the AU: input1 is from stored data corresponding to global bus input and input2 is from 'coeff0' value stored in another register within the DPU.
- Configure adder inputs of the AU: input1 is from multiplier output and input2 is from intermediate value corresponding to LHS input stored in a data register.
- Put adder output into side-out output.
- Tri-state the main output.

The resultant structure is shown in Figure 7.2.

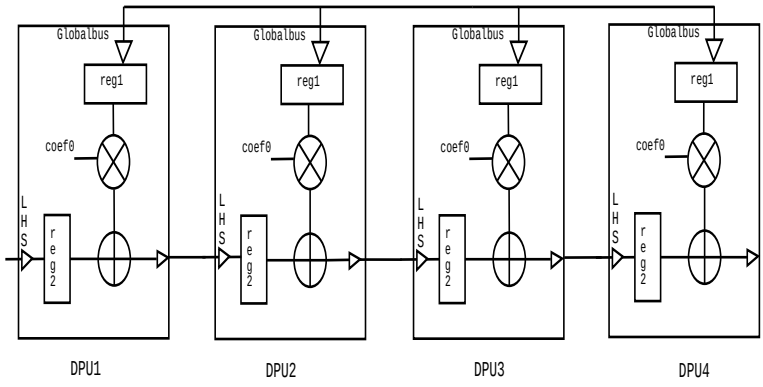

Figure 7.2: First clock cycle in half-band mapping

In the second clock cycle, all steps, except the following, are the same as above.

- Read intermediate data value from RHS input into a data register. In all operations in the first clock cycles, LHS is replaced by RHS.

- In DPU1, Put adder output on to the main output. This is filtered output from half-band filter. Store this output into memory for next stage.

The resultant structure is shown in Figure 7.3. It is important to note

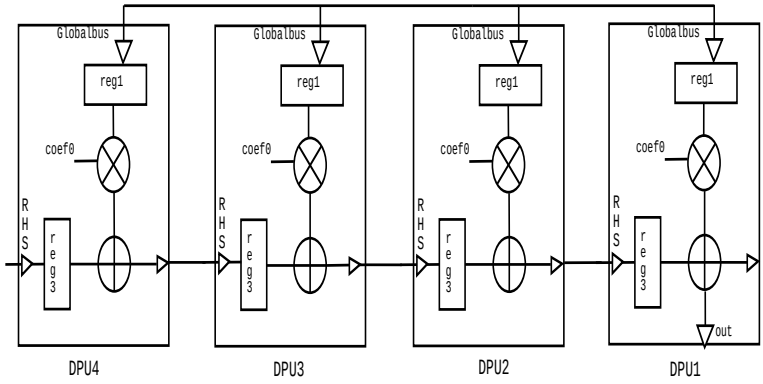

Figure 7.3: Second clock cycle in half-band mapping

that, we are not preforming any multiplication in the second clock cycle. So, we are reducing the multiplications by 2, because of the linear phase property. The polyphase representation, discussed in chapter 4, is not used here because we cannot gain much by reducing the speed of operations in AU. This speed of operations in AU is determined by other steps. Moreover, a polyphase implementation will also make the control more complex.

7.2 Mapping of matched FIR filter

The input data after half-band filtering and decimation is processed into 17^{th} order matched FIR filter. This means that we need 17 basic computations equivalent to a MAC operation (shown in Figure 4.6). For each sample, our implementation can range from using one DPU, i.e., 17 clock cycles for one computation to seventeen DPUs ,i.e., one clock cycle computation. We propose to use an intermediate solution which uses 2 clock cycles for one computation of real or imaginary data. Data processing of real and imaginary parts are done in alternate cycles. This means that there will be 4 clock cycles of computation for each data input. For this solution we need 9 DPUs. This decision is the main determining factor for choosing 9 DPUs in the proposed architecture. Scheduling corresponding to real part is discussed in next few lines. Imaginary part will be calculated in the same way.

- Load data sample from memory into the global bus connecting DPU1-DPU4 and into globalbus connecting DPU5-DPU9.

- Each AU is configured for muliply-and-add.
- Read data from global bus input into a data register.
- Read intermediate data value from LHS input into a data register.
- Configure multiplier inputs of the AU: input1 is from stored data input corresponding to global bus input and input2 is from 'coeff1' value stored in another register within the DPU.
- Configure adder inputs of the AU: input1 is from multiplier output and input2 is from intermediate value corresponding to LHS input stored in a data register.
- Put adder output into 'sideout' output (Data is flowing from left to right).
- Tri-state the main output of each DPU.

Dataflow in this clock period is shown in Figure 7.4. Similar, to the

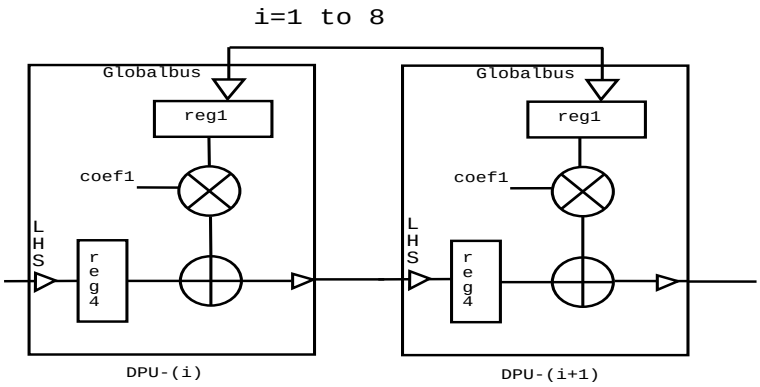

Figure 7.4: First clock cycle in FIR mapping

half-band filtering step, in the second clock cycle, only the following steps are different.

- Read intermediate data value from RHS input into a data register. In all operations in the first clock cycle, LHS is replaced by RHS.
- In DPU1, Put adder output on to the main output. This is the filtered output from FIR filter. Store this output into memory for the next stage.

Dataflow in this clock period is shown in Figure 7.5. This implementation allows us to use linear phase property and hence, number of multipliers in hardware are reduced by half. Also, the speed of multiplication and addition in the AU is corresponding to the critical path delay of the system.

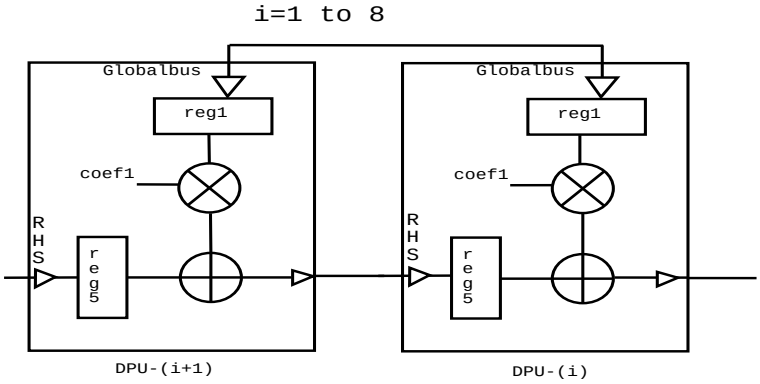

Figure 7.5: Second clock cycle in FIR mapping

7.3 Complete dataflow mapping for Bluetooth

It is clear from the above that individual steps from the Bluetooth algorithms can be mapped on to the proposed architecture. For the complete dataflow, we have two options: 1) Take three instances of the proposed architecture and do static scheduling, or, 2) Do time multiplexing and use the same datapath for all three steps of the demodulation. It turns out that for FFT we need only one instance of the proposed architecture. So, for Bluetooth also, we should use only one instance of the proposed architecture. This means that we need to do time-multiplexing of our datapath and perform the dynamic scheduling. As already indicated, our reconfigurable hardware need only one clock cycle for reconfiguration. So, we can easily perform this reconfiguration in real-time. To achieve this, our first step step is to convert the incoming Bluetooth data stream into data blocks. For this purpose, we divide the input data into blocks of 32 samples each and perform the computations. This is shown in Figure 7.6. The size of sample blocks (=32 samples) is a compromise between latency(real-time) requirements of the system and energy spent in the frequent reconfiguration of the system. Very small data size will have good latency performance, but it will require extra energy due to frequent reconfigurations of the system. Very large data size will perform poorly with respect to latency of the system.

7.4 Mapping of FFT

The HiperLAN2 receiver uses a 64-point FFT for OFDM. The heart of the FFT is the butterfly computation. As already discussed, we use radix-2 butterfly for regularity and ease of computation. This means that we will have 32 butterflies and six stages of computation. The basic butterfly was shown in Figure 4.10. From the Figure, it is clear that the real and the imaginary part of a butterfly have a similar structure. For hardware

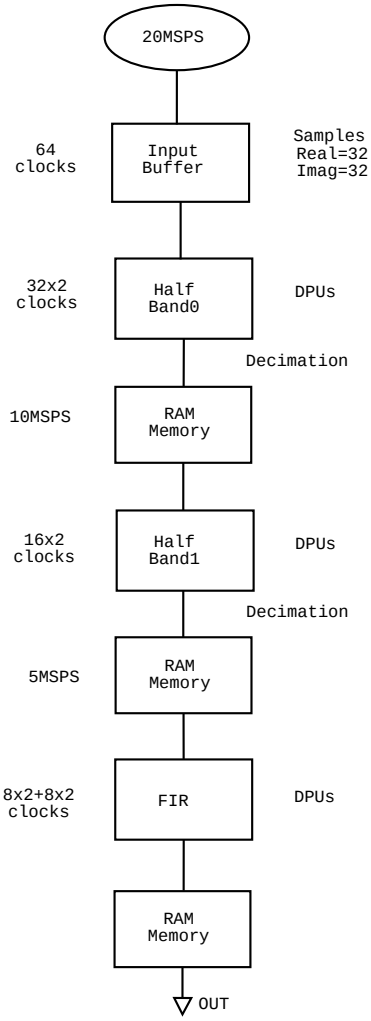

Figure 7.6: Dataflow mapping for Bluetooth

mapping, we need two ROMs for storing real and imaginary parts of twiddle factors ($= e^{-j2\pi k/N}$). There are 2 memory (RAM) units required for storing real and imaginary part of data of one stage. In the next few lines, we will discuss the mapping corresponding to real part of butterfly. This mapping needs four DPUs each for real and imaginary part of butterfly. So, we will need use DPU1-DPU8. This means that throughput of our design will be one butterfly per clock cycle. Therefore, we will need 32 clocks to compute one stage of FFT. In total we will need $32 * 6 = 192$ clocks of computations. Configuration of each DPU is described below and is also shown in Figure 7.7.

- Configure DPU1 for addition; Read data from FFTbus input and bus2 input; Put the AU output into the A_{re} memory.

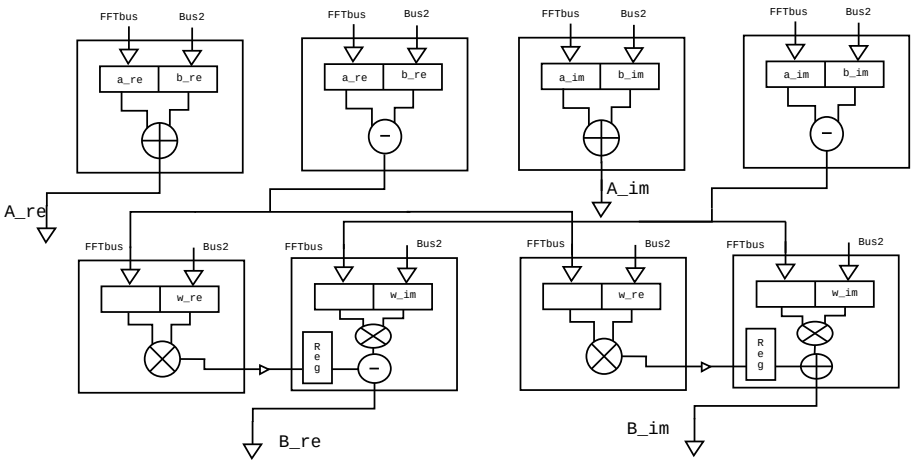

Figure 7.7: One butterfly Mapping

- Configure DPU2 for subtraction; Read data from FFTbus input and bus2 input; Put the AU output onto the FFTbus input of DPU5 and DPU7.

- Configure DPU3 for addition, Read data from FFTbus input and bus2 input. Put the AU output into the A_{im} memory.

- Configure DPU4 for subtraction; Read data from FFTbus input and bus2 input; Put the AU output onto the FFTbus input of DPU6 and DPU8.

- Configure DPU5 for multiplication; Read data from FFTbus input and bus2 input; Put the AU output onto the sideout.

- Configure DPU6 for multiply and subtract; Read data from FFTbus input and bus2 input into multiplier; Put the multiplier output and LHS input into the subtractor. Put the AU output into the B_{re} memory.

- Configure DPU7 for multiplication; Read data from FFTbus input and bus2 input; Put the AU output onto the sideout.

- Configure DPU8 for multiply and add; Read data from FFTbus input and bus2 input into multiplier; Put the multiplier output and LHS input into the adder. Put the AU output into the B_{im} memory.

- Configure DPU9 for Sleep mode.

This implementation is slightly different from basic butterfly computation. This is because, we are registering the data output of DPU2. This will cause one clock latency with respect to Figure 4.10.

7.5 Complete dataflow mapping for HiperLAN2

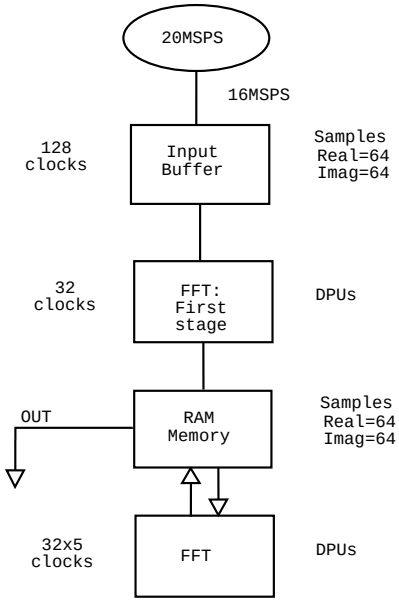

Figure 7.8: Dataflow mapping for FFT

The input data is stored in the input buffer and then used for the computation of the first stage of FFT. After the first stage, RAM memory is used to store the data. So, all the stages (except the first stage) are doing the in-place computations. Complete dataflow mapping for HiperLAN2 is shown in Figure 7.8.

7.6 Discussion

In this chapter we have demonstrated the mapping of SDR algorithms onto the proposed architecture. But, the separation of buffer and memory has not yet been discussed. Basically, the buffer is used to read the data from the AD block. This data is coming on a sample-by-sample basis. But, our operations are performed on blocks of data (block of 80 samples for FFT and block of 32 samples for Bluetooth). So, we need to buffer the input data first and convert it into a block of appropriate size. While the input buffer is filled by AD block, our data processing path should not remain idle. This is because of real-time application requirements, which needs as low latency as possible. Hence, we introduce pipelining into our archutecture. We fed the data block from the input buffer for the first stage of SDR algorithms and store the intermediate results into RAM memory unit. Subsequent computations will access data from RAM. This means that AD block can feed the input buffer with further data and when the computations on one

block of data is finished, datapath can immediately start processing new block of data. In this way, datapath will not remain idle at any time during algorithm execution.

The realization of above concepts are done using a SystemC RTL description. Datapath and control are synthesized in 0.18μ technology. The synthesization results and performance evaluation of our system is done in the next chapter.

8

Synthesis and Evaluation

The proposed architecture and the mapping of chosen algorithms on the proposed architecture has been described in previous chapters. This chapter elaborates on the synthesis results and evaluates the design after hardware realization. Section 8.1 discusses the minimum speed requirements that the design must fulfill to meet the SDR receiver requirements. In section 8.2, the synthesis results for the proposed design are presented. The synthesis results for each receiver (Bluetooth and HiperLAN2) individually, are also discussed there. Section 8.3 summarizes the performance of Montium TP, when the chosen SDR algorithms are mapped onto it. Section 8.4, compares the performance of the proposed system with the performance of Montium TP for the chosen SDR algorithms. Section 8.5 compares the proposed design and implementation, with respect to the FFT computation, with some other designs .

8.1 Performance requirements

In this section, the estimation of the minimum speed requirements of the proposed system is given.

8.1.1 Speed requirements for the OFDM datapath

As shown in Figure 7.8, input data is coming at 20 MSPS. One OFDM symbol contains 80 complex input samples. The first 16 samples of each OFDM symbol are the same as the last 16 samples (OFDM cyclic shift property). So, a useful data-input to the OFDM demodulator is 16 MSPS. In the proposed implementation, we can perform one butterfly computation in each clock cycle. For a 64-point FFT, using radix-2 computation, we need[1] $64/2 * 6 = 192$ clock cycles. This would mean that at least 192 clock cycles are needed in 4 μs duration. So, minimum clock frequency required is

[1] $N/2*\log_2 N$

48 MHz. This means that the system computes one OFDM symbol every 4 μs, when running at 48 MHz. But, for real-time operation, we need to reduce this latency. Hence, in the actual system we need to compute the FFT at higher frequency than 48 MHz. The actual speed will be determined by the overall complexity and the latency requirements of the complete receiver.

8.1.2 Speed requirements for the Bluetooth datapath

As shown in Figure 7.6, a block of 32 complex samples will require 64 + 32 + 32 = 128 clock cycles for channel selection. Again, the input data rate is 20 MSPS. Thus, the time for each sample will be 50 ns. This means that 32 samples should be processed within $32 * 50 = 1600$ ns. This means that the minimum clock frequency required will be 80 MHz. Assuming, input buffering requires 64 clock cycles to buffer 32 complex samples, this means that the latency will be $128 + 64 = 192$ clock cycles. At 80 MHz clock frequency, this is 2.4 μs. If input buffering requires 32 clock cycles, then the latency will be 2 μs.

8.1.3 Overall speed requirements

The control section, configuration section and memory should be able to run at the frequency corresponding to the maximum of *minimum datapath frequency required* of the two receivers. This means that if minimum datapath frequency is 80 MHz, then control, configuration and memory blocks should also be able to run at least at 80 MHz. In that case, the overall speed requirement of the system will be 80 MHz. This should not be a problem as the control part performs relatively simple computations.

It is clear from the discussions in this section that the speed of datapath operations will be determined by the minimum frequency of operations needed to meet the latency/real-time requirements of the overall system.

8.2 Synthesis results

The design has been synthesized using the 0.18μ UMCL18U250 CMOS technology. This process has a density of 82 kgates/mm^2. For ASIC synthesis, worst case conditions (Vcc= 1.65V and Temperature = 125^0C) are assumed. The area estimated by the synthesis tool does not include area due to wiring/routing. Additional area needed for wiring is assumed to be 10 percent of the total area (a realistic figure according to Philips Research experts [26]). This area is also included in the total area estimation of the system.

8.2.1 Synthesis results for the SDR receiver

The results of synthesis are shown in Table 8.1. These results indicate that the proposed system approximately requires 0.6 mm^2 of silicon area and has a critical path length of 5.3 ns. Thus, the maximum operating frequency of the system is 188 MHz, which is well above the minimum operating frequency estimated in the previous section. This gives us enough room to play with the latency requirements of the overall system. The

Component	Area[μm^2]	Critical Path[ns]
DPU(x9)	510000	5.3
Control	26000	3.8
CU	1300	-
Wiring	62700	-
Resultant	600000	5.3

Table 8.1: Synthesis results for SDR receiver

results of synthesis are used as an indicator to evaluate the performance of our system. It is important to note that we have not included the area required due to various memories (RAM, ROM, Buffer) in the system. In the proposed design, we need two RAMs of 128x16 size each and one ROM of 64x16 size. From the above results, it is clear that majority of area is consumed by the datapath of the system. The control part consumes less than 5 percent of the total area.

In the sub-sections below, an *estimation* of the area requirements of each receiver, when designed individually on the separate hardware, is given.

8.2.2 Synthesis results for the Bluetooth receiver

If only Bluetooth receiver needs to be designed, then we will need all nine DPUs. But, the area corresponding to multiplexing due to various modes of DPU will be reduced. In this case all of the DPUs will need to have modes corresponding to different modes for Bluetooth operations only. We

Component	Area[μm^2]	Critical Path[ns]
DPU(x9)	480000	5.3
Control	18000	2.4
CU	1300	-
Wiring	50700	-
Resultant	550000	5.3

Table 8.2: Synthesis results for Bluetooth receiver

estimate that there will be less than 5 percent of area gain in each DPU. This will mean that one DPU will need an area of approximately 53300 μm^2. Further, the CU will also require smaller area. But, this reduction will also be insignificant compared to the total area of the system. The control section (i.e., FSM), in this case, will need 18000 μm^2 area. So, total area required will be 550000 μm^2. These results are shown in Table 8.2. Also, there will not be any ROM required in the system and RAM memory requirement will also be reduced to one RAM of 64x16 size and another RAM of 32x16 size. Additionally, the memory bandwidth requirement will be reduced. This will result in further reduction of wiring area. The critical path for DPUs will be the same as in the current implementation, but the critical datapath length for the control section will be reduced to 2.4 ns. So, the maximum operating frequency of the system will remain at 188 MHz.

8.2.3 Synthesis results for the HiperLAN2 receiver

In the HiperLAN2 mode, we need only 4 multipliers and 6 adder/subtractor blocks for each butterfly computation (see Figure 4.10). Also, the DPU modes corresponding to various filter stages are not required. This means that there will be a reduction of at least 55 percent in the datapath area. Secondly, the control section, in this case, will require 12000 μm^2 area. The

Component	**Area[μm^2]**	**Critical Path[ns]**
DPU(x9)	230000	5.3
Control	12000	2.3
CU	1300	-
Wiring	47700	-
Resultant	290000	5.3

Table 8.3: Synthesis results for HiperLAN2 receiver

CU part will remain more or less the same. So, total area required will be 290000 μm^2. These results are shown in Table 8.3. Also, the memory and memory bandwidth requirements will remain same. The critical path length in control section will be reduced to 2.3 ns, but the critical path length in datapath will remain the same. So, maximum operating frequency of the system will remain at 188 MHz.

8.3 Performance of Montium TP

As mentioned in chapter 5, Monitum TP has been designed recently. A Montium TP requires approximately 2 mm^2 silicon area in 0.12μ Philips CMOS technology. The results in this section are directly taken or derived from [26].

8.3.1 Montium mapping : OFDM

An OFDM symbol can be decoded using a single TP. The maximum frequency of operation in this mode is 100 MHz and it will take 204 clock cycles to perform the FFT. The configuration time is 473 clock cycles, which will be required in the initialization phase. Streaming in the input data will require 64 clock cycles more and to read 52 samples at output will need 52 extra clock cycles. So, in a sequential scenario, where the input is loaded, the algorithm is executed and data is retrieved, a total of 320 clock cycles are needed. Consequently, the tile processor has to run at 80 MHz to perform the FFT within a 4 μs time window. If the FFT algorithm is implemented in a streaming fashion, then the communication time can be neglected and a clock frequency of 51 MHz would suffice but at the moment communication and configuration unit (CCU) are the limiting factors for this.

8.3.2 Montium mapping : Bluetooth

Montium TP can run at 140 MHz maximum clock frequency in FIR configuration. In a single clock, a Montium TP can compute five taps of a filter. This means that a single half-band filter of 7th order will require two clock cycles to compute one sample (either real or imaginary input stream). Similarly, a matched filter of 17th order will require four clock cycles. Also, in a sequential scenario, input data must be stored in the local memories before computation can start. This will take one clock cycle each to store the one sample of input data and one clock each to retrieve one sample of output data. Configuration time for FIR filter varies from 50 to 200 clock cycles and will be required in the initialization phase.

The input data stream (complex) is providing samples at 20 MSPS. So, each sample must be processed in 50 ns time duration. The first halfband filter will require 2 clock cycles for real data and 2 clock cycles for imaginary data. The second half-band filter stage will also need 2 clock cycles for real and 2 clock cycles for imaginary data samples. But, data samples are decimated by 2 after first half-band filter. So, each complex input data sample will require $(2+2)+(2+2) \div 2 = 6$ clock cycles in this stage. Then the next stage corresponding to the FIR filter will require 4 clock cycles for real data and 4 clock cycles for imaginary data. Since the input to the FIR is decimated by 4 from original input stream, so the FIR, on an average, will require $(4+4) \div 4 = 2$ clock cycles for each complex input data sample. Therefore, total of 8 clock cycles are required by one bluetooth sample on Montium TP. This will mean that Montium TP need to run at 8÷50e-9 = 160 MHz. This is more than the maximum clock frequency of the TP. Therefore, mapping of Bluetooth will require at least 2 Montium TPs. The communication between these two processors will go through peripherals and will consume additional energy. Also, the CCU needs to be improved

to allow the above mentioned real-time data operations.

8.4 Comparison of proposed design with Montium TP

It is clear from the previous section that it will be very difficult for a single Montium TP to satisfy the real time requirement of the parts of HiperLAN2 receiver we chose to implement. In the case of Bluetooth receiver, even if we use the Montium TP with maximum operating frequency, still we will need 2 TPs to realize the various filter stages. It is very difficult to exploit the linear phase property of the filters because FIR matched filter requires 4 clock cycles. Also, the more general bus network in Montium TP implies more energy wastage in charging and discharging of redundant capacitances. The configuration time of a Montium TP varies depending on the algorithm e.g. a 64-point FFT needs 473 clock cycles and an FIR filter of 20^{th} order needs 270 clock cycles.

The Montium TP occupies 2 mm^2 area in CMOS12 process from Philips [4]. The maximum clock frequency for Montium TP, is according to the synthesis tool, about 40 MHz. It is estimated that the Montium TP ASIC realization can implement an FIR filter at about 140 MHz and an FFT at about 100 MHz. The CMOS12 process has a gate density of 200 kgate/mm^2. So, if we normalize our synthesis results to this process, our implementation will need 0.24 mm^2 area (approximately 8 times smaller than one Montium TP). But, it is important to notice that in the Montium TP, approximately 0.5 mm^2 area is occupied by RAM memory. In our system, we need a RAM of 256x16 size and a ROM of 64x16 size, which will occupy an additional area of approximately 30000 μm^2 in our system.

On the other hand, the Montium TP has much more flexibility and is suitable to implement a number of DSP algorithms [26]. In the design space, our system is closer to the ASIC implementation than the Montium TP (which is a domain-specific reconfigurable accelerator for the chameleon SoC).

8.5 FFT Implementation on other architectures

In this section we will discuss briefly some other designs with respect to area and speed for FFT computation. The results are taken directly from [26].

8.5.1 FASRA

An FFT algorithm-specific reference architecture (FASRA) was developed in order to compare the Montium TP with both an ASIC and an FPGA. The FASRA is a dedicated FFT processor capable of computing upto 1024-point

FFTs. It can be thought of as an algorithm specific instruction processor (ASIP). The datapath of FASRA is shown in Figure 8.1. It can compute one radix-2 butterfly per clock cycle. A FASRA ASIC with 16-bit datapath

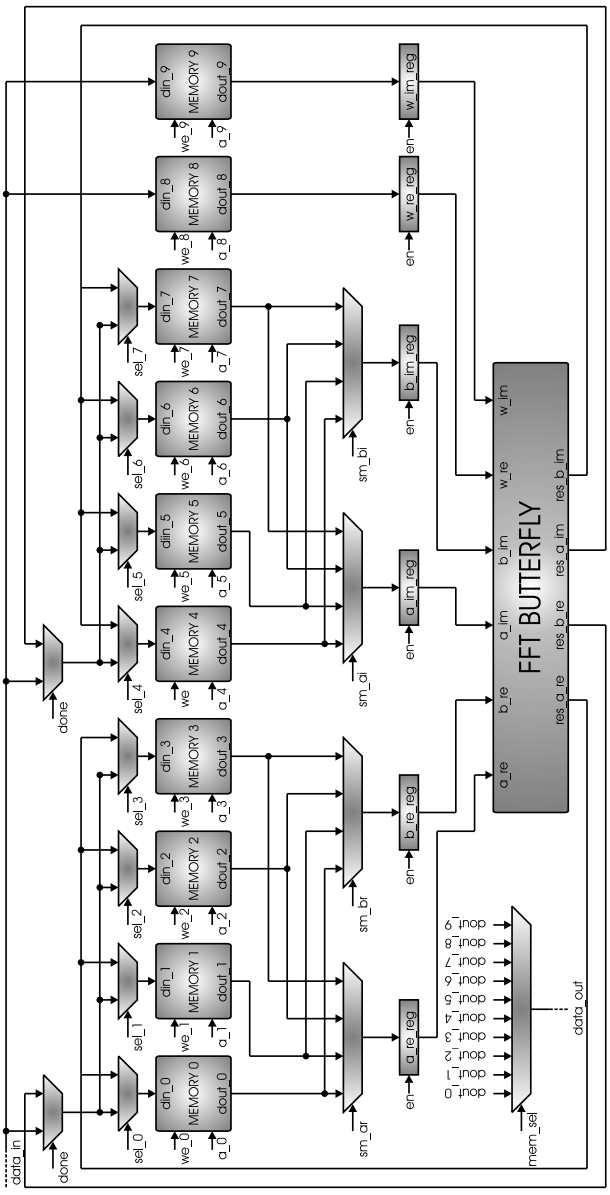

Figure 8.1: FASRA datapath architecture

was designed in VHDL and synthesized in CMOS12 process from Philips. The area (excluding wires) of the resulting ASIC is 0.63 mm^2. The area of the datapath is 0.62 mm^2. There are ten data memories of 512x16 size each in the datapath. These memories will consume an area of about 0.46 mm^2.

So, the effective area used by computational part in the datapath is 0.16 mm^2. The area of the controller is 0.01 mm^2. The maximal clock frequency for the FASRA ASIC is 120 MHz.

On a Xilinx Virtex-II Pro FPGA (CMOS12 process), with smallest device (XC2VP2) for syntheis, the maximum clock frequency of FASRA is 63 MHz.

8.5.2 Avispa

The Avispa block accelerator from Silicon Hive [5] has been developed for efficient and reconfigurable acceleration of DSP algorithms such as OFDM. The Avispa has in total 75 function units, which include four 16x16 multipliers. In CMOS12 process, it occupies 6.5 mm^2 area and the maximum clock frequency is 150 MHz. It can compute one butterfly per clock cycle.

8.5.3 ARM920T

The ARM920T is a 32-bit GPP. In 0.13μ technology, it has an area of 4.7 mm^2 area and a maximal clock frequency of 250 MHz. A single butterfly computation on it takes 21 clock cycles. so, the FFT butterfly frequency of an ARM running at 250 MHz is 12 MHz only.

8.5.4 Comparison of different implementations

Table 8.4 depicts a quick comparison (for butterfly computation) of different designs mentioned above. We have chosen to compare FFT (butterfly), because we know that in FFT we have about 50 percent of redundant hardware in our implementation (see section 8.2.3).

Architecture type	**Architecture name**	**Area [mm^2]**	**Speed [MHz]**
ASIP	FASRA (ASIC)	0.63	120
DSRA	Avispa	6.5	150
DSRA	Montium TP	2	100
GPP	ARM920T	4.7	12
Reconfigurable ASIC	Our design (excl. RAM and ROM)	0.24	188

Table 8.4: Comparison of different architectures for butterfly computation

It is important to note that all these designs, except ours, have lot of data memories to store data operands and intermediate and final results. For example, in the FASRA-ASIC, approximately 0.5 mm^2 area is occupied by the RAM memory. For our system, we need an additional area corresponding

to RAM (256x16) and ROM (64x16). This area will approximately be equal to 0.03 mm^2 in CMOS12 Philips process.

9

Summary and Conclusions

This chapter summarizes the work done in this project regarding realization of the chosen SDR algorithms on reconfigurable hardware. The design flow is explained first. In the section 9.2, our architecture design approach is explained. In section 9.3, the performance results of our system are discussed. The chapter ends with the conclusions and suggestions for future work.

9.1 Design flow

This project started with acquiring the basic understanding of the SDR receiver architecture and estimation of the computational complexity of the SDR algorithms. The channel-selection block in the Bluetooth receiver and the OFDM block in the HiperLAN2 receiver are the most computationally demanding parts in our SDR receiver. So, these blocks were chosen to be implemented in reconfigurable hardware in the project. The algorithms to be implemented in hardware were further analyzed later on.

The algorithm-analysis step was followed by learning about available tools and design language for our design. We have used a SystemC RTL description in Synopsys CoCentric System Studio in this project. The main reason for using SystemC is that the algorithms were already proven in a software environment. So, it is logical to use the hardware environment which resembles that software environment. The RTL description, which typically leads to a more optimal realization of hardware than a behavioral description, is used to describe our code. The design methodology of SystemC is described in appendix-C). The SystemC description is independent of the target technology. This means that a synthesizable SystemC code can be mapped to virtually any FPGA or ASIC technology. The main drawback is that efficiency of synthesized code is largely dependent on tooling. So, the synthesis results are not optimal, but still a good indicator of the actual design.

In the next step, we have studied and evaluated the various contemporary

design approaches to implement a reconfigurable system. Based on these approaches, we have proposed a reconfigurable architecture for our system and analyzed the mapping of chosen SDR algorithms on it. Various design decisions with respect to implementation issues had also been taken in this step.

Following this, we have implemented the architecture and mapped the chosen algorithms on it. For design description, 'divide-and-conquer' approach is followed. Using this approach, we have divided our system into basic building blocks and the basic building blocks are defined in terms of separate sub-modules. The correctness of these sub-modules and basic building blocks has been tested individually. Finally, the complete system is tested using a testbench environment for each algorithm. In this testing, the tester generates a set of input sequences that initialize the hardware and start the computation on a predefined input data sequence. The computed results are evaluated against the software computations (floating-point) from [39]. Finally, the error due to finite precision of hardware is calculated to ensure that error is within the acceptable limits.

In the next step of this project, we have synthesized our design using the 0.18μ UMC CMOS technology. The results of synthesis are used to estimate the performance of our system with respect to the SDR receiver requirements.

Finally, our synthesis results are compared with the implementation on a DSRA (Montium TP-recently designed at the UT). The performance of our architecture with respect to various other architectures is also analyzed.

9.2 Architecture design

It was shown in [17] that a multiprocessor system (RAW) can be programmed easily in a fashion that uses its peak performance, while an ALU-array like architecture (XPP) has difficulty in achieving peak performance. But, the peak performance of XPP is much higher than RAW. In fact, using about 15 percent of the peak performance for computations in an application would still mean that the ALU-array is competitive when compared with the more traditional multiprocessor architecture.

In our architecture design, we have used the conclusion above to implement our datapath as an ALU-array. But, as indicated above, the ALU-array architecture has difficulties in achieving peak performance. The main bottleneck for this is the inefficient implementation of control operations. So, to overcome this performance bottleneck, we have separated the datapath operations from control operations and designed the dedicated hardware for each separately. In this way, we have combined the ALU-array and multiprocessor approaches to *realize and achieve* the peak performance in our system. As indicated in the mapping section, the datapath resources (vari-

ous DPUs) can be used completely for datapath computations in our system, if required. This shows that it is relatively easy to achieve the peak performance in our system. This can be attributed to the limited hardware resources required by the control part of our system. This is in contrast to various approaches mentioned in the chapter 5, where peak performance is difficult to achieve due to resources claimed by operations corresponding to the control parts of the algorithms.

Also, in our architecture, we have used the fact that all of the communication in our system is predetermined. The chosen SDR algorithms do not require dynamic determination of communication patterns. So, exploiting the fact that the communication patterns in our system are predictable, we have optimized the interconnect network of our system. This approach of interconnect optimization is the basis of [29]. Our architecture has been developed with the belief that most, if not all, communication in data-intensive applications can be determined at design-time. This approach emphasized hardware minimization and interconnect performance at the cost of some flexibility. It is shown in [28] that this approach gives significant gains in performance compare to a hierarchical bus-based system-on-chip approach.

Further, to optimize the performance of the system we have incorporated various design techniques like using:

- smaller busses for capacitance minimization,
- local registers instead of central register schemes (locality of reference),
- registered inputs for datapath to reduce unwanted glitching,
- distributed control instead of a central control,
- preference of short distance communication over long distance communication,
- pipelining in butterfly computation,
- parallel processing, and,
- facility of sleep mode in DPUs.

9.3 Conclusions

This section concludes the work and summarizes the achievements and lessons learnt through this project.

- In our SDR receiver, the Bluetooth channel selection algorithm requires more datapath resources than the HiperLAN2 OFDM demodulation. On the other hand, HiperLAN2 demodulation needs more memory and memory bandwidth.

- By incorporating limited flexibility in our system, we are able to reduce the total hardware required to implement the SDR receiver compared to the implementation in which each receiver is implemented individually. This is shown in Table 9.1. It can be concluded that an area reduction of about 25-30 percent can be made in the combined implementation compared to the individual implementations of the two receivers.

- Dynamic reconfiguration in our system allows time-sharing of hardware resources by pipelining algorithms, thus, increasing the performance of overall system at the cost of some latency.

- For state-of-the-art designs, an ASIC implementation with minimal flexibility can easily outperform the flexible implementation. The results of our ASIC-like implementation were shown to be superior to the implementation on more flexible systems. A GPP (ARM920T) based implementation requires 20 times more area and computes 15 time slower than our ASIC-like implementation. A domain specific processor like Montium TP requires 15 times more area than our implementation to meet the SDR computational requirements. On the other hand, flexible solutions like the Montium TP and GPP are superior to our design in terms of suitability for different algorithms and ease of implementation. So, a design decision based on the performance requirements and implementation costs needs to be taken before deciding on the platform and methods for the final implementation of a DSP system. It can be concluded that the performance of ASIC > ASIP; ASIP >DSRA; DSRA > GPP, while the flexibility of ASIC < ASIP; ASIP < DSRA; DSRA < GPP.

- By introducing pipelining in the datapath, we are able to perform computations at higher speed than a non-pipelined datapath (Table 8.4).

- The 16-bit datapath performs satisfactorily for the chosen SDR algorithms.

Component	**Sum of Separate Implementations**	**Combined Implementation**
Computation area[μm^2]	840000	600000
RAM	352x16	256x16
ROM	64x16	64x16

Table 9.1: Area requirements of SDR receiver

- A high-level description language, like SystemC, can be used to design VLSI systems. The benefits are in timely and easily realization of a design. The main drawback is that efficiency of synthesized code is largely dependent on the tools.

- Almost all of the WLAN systems use either phase-modulation or OFDM-modulation. So, the suitability of our system for phase-modulated (Bluetooth) and OFDM (HiperLAN2) receivers implies that our design can be used in a number of WLAN systems.

9.4 Future work

The last step of this MSc. project was the hardware synthesis. Due to shortage of time, we were not able to validate the synthesis results. This will be the next step for the remaining work for the hardware implementation of the SDR project. Also, in our FFT implementation, we have not performed the bit-reversing operation on the output. This should be taken into consideration in the next stage of the receiver implementation while reading the data from the memory. Also, the datapath may be changed to heterogenous DPUs to reduce the area. The control section can be optimized further. The butterfly computations in the last stage of FFT can be simplified to simple addition-subtraction operations. The overflow and underflow conditions need to be incorporated in the complex multiplication and addition functions. Also, extensive power consumption analysis in the system still needs to be done.

From the results, it is clear that Bluetooth looks more complex than HiperLAN2, which is contrary to the initial assumption of the SDR project. The computational complexity of Bluetooth receiver can be simplified by reducing the order of filters, or increasing the decimation. Currently, the decimation factor is 4, which gives data rate of 5 MSPS for 1 MHz Bluetooth channel. If we change the decimation factor to 6, the data rate will be 3.33 MSPS for 1 MHz channel (a theoretically sufficient number). Also, the sample rate reduction block after ADC block may also be modified.

In the broader context, the design was made as a subsystem of SDR transceiver system. Also, the other blocks of the SDR receiver need to be implemented in hardware. The SDR transmitter needs to be designed and implemented as well.

A

Appendix A - Architecture View

The architecture view of the complete system along with the test-pattern generator is shown in Figure A.1. It is shown here to indicate the connections between various entities of the system.

The architecture view of the complete datapath is shown in Figure A.2. It is shown here to indicate the connections between various datapath (9 DPUs) entities of the system.

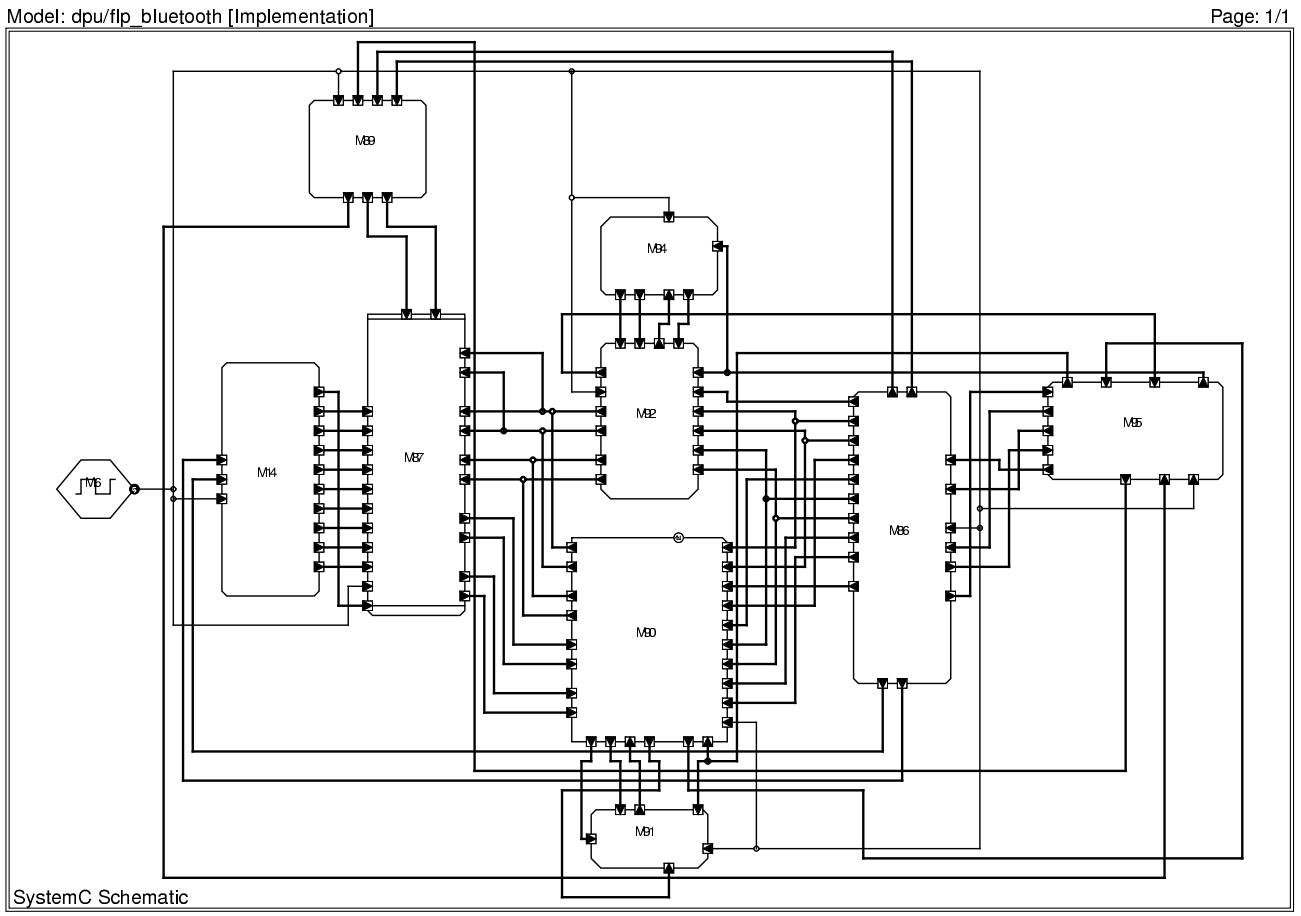

Figure A.1: Architecture view of the system

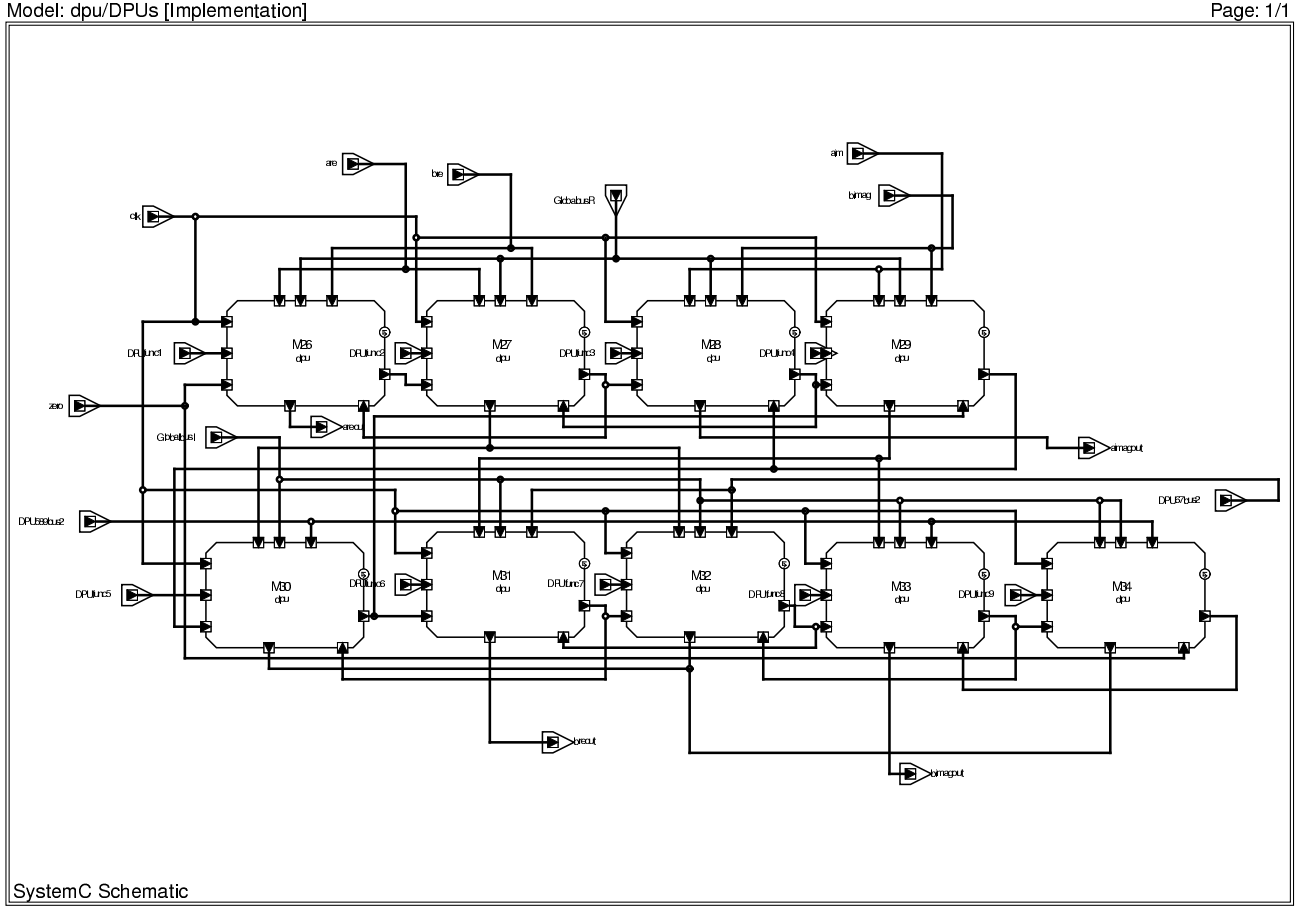

Figure A.2: Architecture view of the datapath

B

Appendix B - Floating point Vs Fixed point system

In this section, the errors due to finite precision, fixed-point datapath in our hardware implementation are evaluated. For this purpose, we are testing our hardware implementation against the floating-point software implementation. The test-vectors used for this comparison are the same test-vectors which were used to validate the software implementation of the SDR receiver.

B.1 OFDM

Figure B.1 shows the SNR degradation in the real part of the fixed-point hardware implementation compared to the floating point software implementation for the OFDM block of the HiperLAN2 receiver.

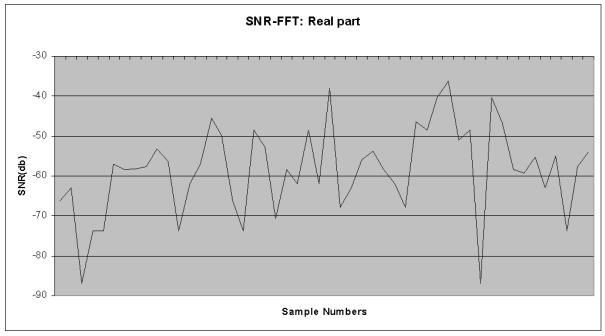

Figure B.1: SNR degradation in Real part of the OFDM block

Figure B.2 shows the SNR degradation in the imaginary part of the fixed-point hardware implementation compared to the floating point software implementation for the OFDM block of the HiperLAN2 receiver.

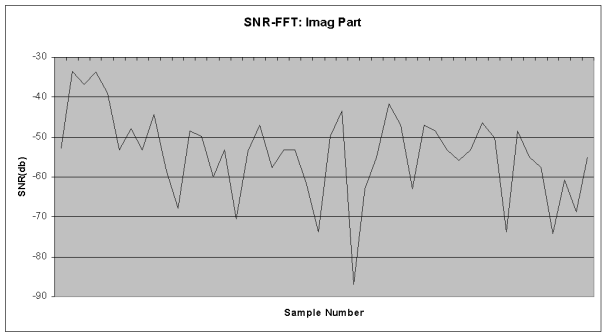

Figure B.2: SNR degradation in Imaginary part of the OFDM block

The maximum SNR degradtion can be seen to -33 dB, which is well below the critical SNR (-26 dB).

B.2 FIR

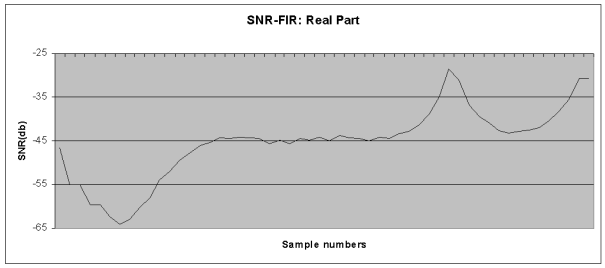

Figure B.3: SNR degradation in Real part of the channel-selector block

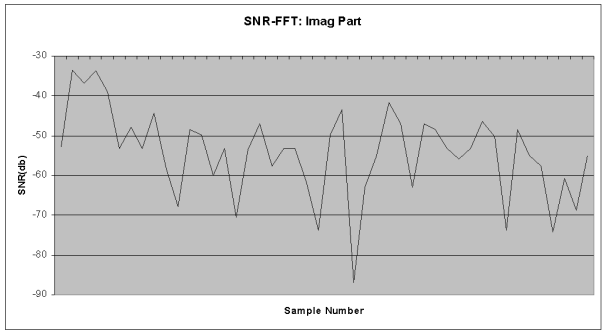

Figure B.4: SNR degradation in Imaginary part of the channel-selector block

Figure B.3 shows the SNR degradation in the real part of the fixed-point hardware implementation compared to the floating point software implemen-

tation for the OFDM block of the HiperLAN2 receiver. Figure B.4 shows the SNR degradation in the imaginary part of the fixed-point hardware implementation compared to the floating point software implementation for the OFDM block of the HiperLAN2 receiver.

The maximum SNR degradtion can be seen to -28 dB, which is well below the critical SNR (-21 dB).

The above results show that 16-bit fixed point datapath provides sufficient accuracy for the SDR receiver.

C

Appendix C - An Introduction to SystemC

The standard C and C++ languages lack the constructs necessary for modelling system architecture such as hardware timing, concurrency, and reactive behavior. SystemC [6, 42, 43] is a C++ class library that can be used to create a cycle-accurate model of software algorithms, a hardware architecture, for interfacing of SoC (System-on-Chip) and for system-level designs. In this way, SystemC and standard C++ development tools can be used to create a system-level model, quickly simulate to validate and optimize the design, explore various algorithms, and provide the hardware and software development team with an executable specification of the system. The testbench used to test the executable specification can be refined or used as to test the implementation of the specification. This can provide tremendous benefits to implementers and drastically reduce the time for implementation verification.

In the next section, the main features of the SystemC Class Library are explained. Later on a brief introduction to Synopsys CoCentric System Studio for algorithmic and architectural design is provided.

C.1 SystemC

SystemC [42, 43] supports the description of hardware, software, and interfaces in a C++ environment. It can be understood as the design and verification language that spans the full development path from concept engineering to implementation in hardware and software. The SystemC design approach offers many advantages over the traditional approach for system level design. The traditional system design methodology starts with a system engineer writing a C or C++ model of the system to verify the concepts and algorithms at the system level. After the concepts and algorithms are validated, the parts of the C/C++ model to be implemented in hardware are

manually converted to a VHDL or Verilog description for actual hardware implementation. This is shown in Figure C.1. This process is very tedious and error prone due to manual conversion and multiple system tests.

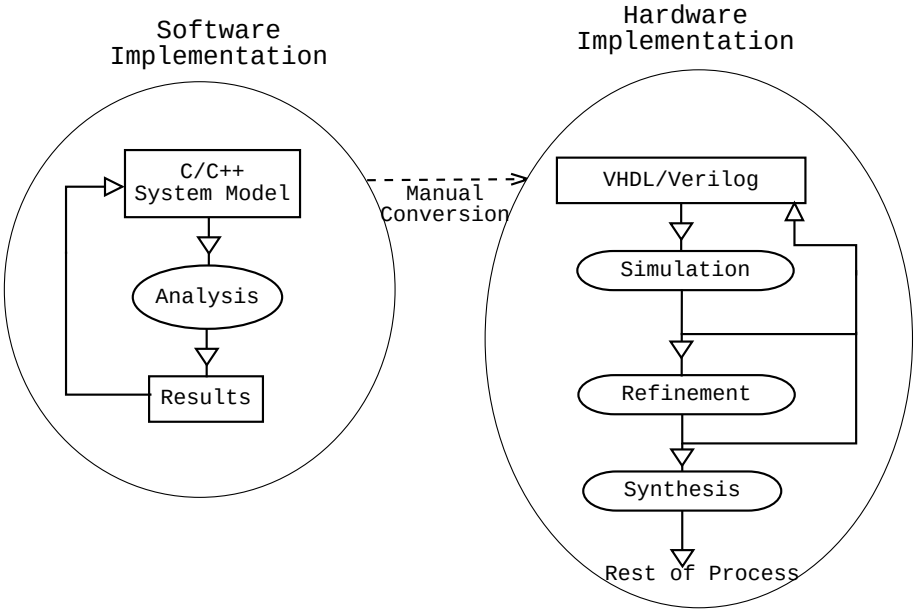

Figure C.1: Traditional Design Methodology

With the SystemC approach, the design is not converted from a C-level description to an HDL in one large effort. The design is slowly refined in small sections to add the necessary hardware and timing constructs to produce a good design. Using this refinement methodology, the designer can more easily implement design changes and detect bugs during refinement. The SystemC design methodology for hardware is shown in the Figure C.2.

In this way, it can used to support hardware-software co-design and the description of the architecture of complex systems consisting of both hardware and software components. The following features of SystemC allow it to be used as a co-design language:

C.1.1 Modules

SystemC has a notion of a container class called a module. Modules are the basic building blocks within SystemC for partitioning a design. They allow complex systems to be broken into smaller, more manageable pieces. Modules are hierarchical and can have other modules or processes contained in it.

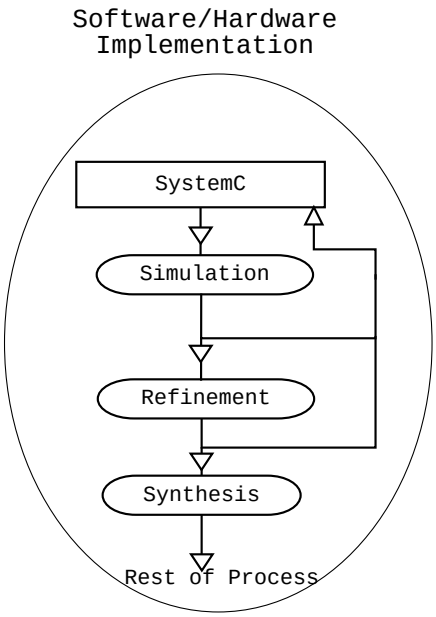

Figure C.2: SystemC Design Methodology

C.1.2 Processes

Processes are functions that are identified to the SystemC simulator to describe the functionality of the system. Processes are contained inside modules and are called whenever the signals that these processes are sensitive to change their value. Some processes behave just like functions; these processes are started when called and return execution to the calling mechanism when it has completed. Other processes are called only once at the beginning of a simulation and are either actively executing or suspended waiting for a condition to be true. This condition can be a clock edge, a signal expression, or a combination of the two. Processes are not hierarchical, so they cannot directly call other processes. Processes can however call methods and functions that are not processes.

C.1.3 Channels

A channel implements one or more interfaces, and serves as a container for communication functionality. A channel may be connected to more than two modules. Different interfaces can also be created by refining predefined interface types.

C.1.4 Ports

A port is an object through which a module, and hence its processes, can access a channels interface. The ports of a module are the external interface

that are used to pass information to and from a module, and trigger actions within the module. SystemC supports unidirectional and bidirectional ports.

C.1.5 Signals

A signal is a primitive channel that connects a port of one module to a port of another module. SystemC supports resolved and unresolved signals. Resolved signals can have more than one driver (a bus) while unresolved signals can have only one driver. To support modeling at different levels of abstraction, from the functional to the RTL, SystemC supports a rich set of port and signal types. This is different than languages like Verilog that only support bits and bit-vectors as port and signal types. SystemC supports both two-valued and four-valued signal types.

C.1.6 SystemC Data Types

SystemC has a rich set of data types that includes all standard C++ data types as well as unique SystemC data types to model systems. The fixed precision data types allow for fast simulation, the arbitrary precision types can be used for computations with large numbers, and the fixed-point data types can be used for DSP applications. It supports both two-valued ('0' or '1') and four-valued data types('0' or '1' or 'X' or 'Z').

C.1.7 Clocks

SystemC has the notion of clocks (as special signals). Clocks are the time-keepers of the system during simulation. Multiple clocks, with arbitrary phase relationship, are also supported.

It is clear from the above that SystemC provides the necessary constructs for system-level modeling. Some of the advantages of using SystemC for system-leve modelling are given below.

- Hardware models can be compiled and simulated in any C++ environment without needing any other simulator.
- Most of the programmers already know C/C++ language. So, they do not need to use any new language.
- Lot of algorithms are already available in the form of C/C++ programs. So, these programs can be used easily in system design.
- Higher abstraction level implies faster implementation, thereby reducing the time-to-market.

It is for this reason we assume that SystemC modeling capabilities can be used for our SDR implementation.

C.2 Synopsys CoCentric System Studio

The System Studio product [7, 8] is a SystemC simulator and specification environment for the joint verification and analysis of algorithmic, architectural, hardware, and software models at multiple levels of abstraction. It consists of tools, methodologies, and libraries that facilitates the design and simulation of systems-on-a-chip. The modeling paradigms supported can be hierarchically mixed at all levels. Cosimulation of SystemC and HDL blocks with the System Studio simulator and HDL simulators through import or export mechanisms is also allowed.

System Studio models are divided into two distinct domains, algorithmic and architectural. Architectural and algorithmic designs can be mixed together in one architectural design and can be simulated the mixed design using System Studios common simulation kernel.

C.2.1 Architectural Design support

Architectural design is design of timed and untimed SoC architectures at multiple abstraction levels from transaction-level modeling (TLM) to register transfer level (RTL) modeling. An SoC architecture contains processing elements (CPUs, DSPs), interconnection elements (buses), storage elements (memories, caches), and other peripherals (address generators, multiply-accumulators, I/O). System Studio supports transaction-level modeling for designing and verifying architectures. Using the transaction-level modeling capability, it is possible to achieve significant simulation performance speedups compared to traditional RTL-based methods. Designers can create and import pin-level models that can be simulated together with transaction-level models, enabling the verification of synthesizable models in the system context simulating at high-speed. The System Studio simulator supports event-driven simulation for simulating SystemC designs.

C.2.2 Algorithmic Design support

Algorithmic design is design of untimed and implicitly timed algorithms and behavioral models at various levels of accuracy from floating-point to fixed-point representations. System Studio supports data flow and finite state machine graphical semantics. The benefit of using this modeling style is ease of modeling, graphical model views, and most important the specific optimized simulation engine. The System Studio compiles control and data-flow models into static or dynamic simulation executables. Its simulator supports compiled, static, and dynamic data-flow simulation.

C.2.3 System-Level Simulation support

System Studio provides a compiled simulation kernel that optimizes parts of system model. The compiled simulation can be statically scheduled and linked with the dynamically scheduled parts of the system. It contains a full stream-driven simulation engine for the accurate execution stream driven simulator models. It also has a debug mode that produces an efficient, yet fully debuggable simulation. Debug mode, allows pausing or single-stepping the simulation, setting breakpoints, and examining the state of the simulation. The DAVIS data visualization tool can be used to monitor any stream of data.

C.3 SystemC to synthesizable description

As mentioned previously, SystemC is a powerful language that allows the designer to develop a complete system description of his design. From this system level design is necessary to get down to a RT synthesizable description that allows a physical implementation of the model. For this purpose we have used SystemC to Verilog translator [3]. In this way, we can obtain from a SystemC description (written following some rules a synthesizable), a Verilog description supported by most of the synthesis tools available in the market.

Bibliography

[1] http://chameleon.ctit.utwente.nl.

[2] http://www.fftw.org.

[3] http://www.opencores.org/cvsweb.shtml/sc2v.

[4] http://www.semiconductors.philips.com.

[5] http://www.silicon-hive.com.

[6] http://www.systemc.org.

[7] Getting started with system studio, 2003.

[8] System studio user guide, 2003.

[9] M. Wan G. Varghese V. Prabhu A. Abnous, H. Zhang and J. Rabaey. *The Application of Programmable DSPs in Mobile Communications*, chapter The Pleiades Architecture, pages 327–360. Ed., Wiley, 2002.

[10] Y. Ichikawa M. Wan J. Rabaey A. Abnous, K. Seno. Evaluation of a low-power reconfigurable dsp architecture. *Proceedings of Parallel and Distributed Processing. SPDP '98 Workshops*, pages 55–60, March 1998.

[11] J. Becker; G. Manfred A. Ahmad and J. Starzyk. A dynamically reconfigurable soc architecture for future mobile digital signal processing. *European Signal Processing Conference, EUSIPCO*, 2000.

[12] P. Jain N. Weng W. Burleson A. Laffely, J. Liang and R. Tessier. Adaptive system on a chip (asoc) for low-power signal processing. *Asilomar Conference on Signals, Systems, and Computers,*, pages 1217–1222, November 2001.

[13] A. Abnous. *Low-Power Domain-Specific Processors for Digital Signal Processing*. PhD thesis, University of California, Berkeley, 2001.

[14] D. McBrien A.J. Anderson. An architecture for software defined radio systems. *GSPx Conference Proceedings*, 2003.

[15] M.J.G. Bekooij. *A constraint Driven Operation Assignment for retargetable VLIW Compilers*. PhD thesis, Eindhoven University of Technology, 2004.

[16] E. Buracchini. The software radio concept. *IEEE transactions on communications*, 38(9), September 2000.

[17] J. Bengtsson D. Johnsson and B. Svensson. Two-level reconfigurable architecture for high-performance signal processing. *The 2003 International Conference on VLSI (VLSI'03)*, June 23-26 2003.

[18] P. Mancini E. Schueler and G. Martinelli. Silicon implementation of a reconfigurable processor array. December 2004 www.eedesign.com/article/showArticle.jhtml?articleId=17408264 .

[19] F. Frescura P. Antognoni E. Sereni, G. Baruffa. A software reconfigurable architecture for 3g and wireless systems. *Proceedings of 3G Wireless 2002*, February 2002.

[20] M.B. Taylor et al. The raw microprocessor:a computational fabric for software circuits and general-purpose programs. *IEEE Micro*, March-April 2002.

[21] Broadband radio access networks (bran) ETSI. Hiperlan type 2 : system overview. Technical Report ETSI TR 101683 V1.1.1.1 (2000-02), February 2000.

[22] Broadband radio access networks (bran) ETSI. Hiperlan type 2 : physical (phy) layer. Technical Specification ETSITS 101 475 V1.2.2 (2001-02), February 2001.

[23] B. Molenkamp G.J.M. Smit, P.M. Heysters. The chameleon project in retrospective. *Proceedings of the 5th PROGRESS symposium on embedded systems*, pages 177–180, October 2004.

[24] P.J.M. Havinga G.J.M. Smit, P.M. Heysters. Exploring energy-efficient reconfigurable architectures for dsp algorithms. *Proceedings of the 1st PROGRESS symposium on embedded systems*, pages 37–46, October 2000.

[25] G. Heidari and K. Lane. Introducing a paradigm shift in the design and implementation of wireless devices. *White Paper*, 2003 www.quicksilvertech.com.

[26] P.M. Heysters. *Coarse Grained domain specific Processor*. PhD thesis, University of Twente, Enschede, 2004.

[27] L.F.W. Hoesel. Design and implementation of hiperlan/2 physical layer models for simulation purposes. Master's thesis, University of Twente, Enschede, August 2002.

[28] S. Srinivasan J. Liang, A. Laffely and R. Tessier. An architecture and compiler for scalable on-chip communication. *IEEE Transaction on very large scale integration(VLSI) systems*, 2004.

[29] S. Swaminathan J. Liang and R. Tessier. asoc: A scalable, single-chip communication architecture. *Proceedings of the IEEE International Conference on Parallel Architectures and Compilation Techniques*, October 2000.

[30] F.L. Anderson J.G. Delgado-Frias, M.J. Myjak and D.R. Blum. A medium-grain reconfigurable cell array for dsp. *International Conference on Circuits, Signal and Systems (CSS-2003)*, pages 231–236, May 2003.

[31] J. Miller J.S. Kim, M.B. Taylor and D. Wentzlaff. Energy characterization of a tiled architecture processor with on-chip networks. *International Symposium on Low Power Electronics and Design, ISLPED*, August 2003.

[32] P.A. Laurent. Exact and appropiate construction of digital phase modulateds by superposition of amplitude modulated pulses (amp). *IEEE transactions on communications*, 34(2), February 1986.

[33] J. Mitola. The software radio architecture. *IEEE transactions on communications*, 33(5), May 1995.

[34] A.V. Oppenheim and R.W. Schafer. *Discrete-Time SignalProcessing.* Prentice Hall, Inc., 2002.

[35] B. Plunkett and J. Watson. Adapt2400 acm - architecture overview. *White Pape*, 2004
www.quicksilvertech.com.

[36] J. Smit G.J.M. Smit P.J.M. Havinga P.M. Heysters, H. Bouma. Reconfigurable system design: The control part. *Proceedings of the 2nd PROGRESS workshop on Embedded Systems*, pages 89–93, October 2001.

[37] F.W. Hoeksema R. Schiphorst and C.H. Slump. A flexible wlan receiver. *14^{th} proRISC workshop on Circuits, Systems and Signal Processing*, November 2003.

[38] F.W. Hoeksema R. Schiphorst and C.H. Slump. A (simplified) bluetooth maximum aposteriori probability (map) receiver. *Proceedings of IEEE SPAWC2003*, June 2003.

[39] R. Schiphorst. *Software-Defined Radio for Wireless Local-Area Networks*. PhD thesis, University of Twente, Enschede, 2004.

[40] E. Schueler. Smart media processing with xpp. *White Paper*, April 2003
www.pactcorp.com.

[41] Bluetooth SIG. Specification of the bluetooth system - core. Technical Specification Version 1.1, February 2001.

[42] S.Swan. An introduction to system level modeling in systemc 2.0, May 2001.

[43] G. Martin S. Swan T. Grotker, S. Liao. *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[44] The XPP team. The xpp: A technical perspective. *White Paper*, March 2002
www.pactcorp.com.

[45] R. Tolimieri and M. An. *Lesser Known FFT Algorithms*. Kluwer Academic Publishers, 2001.

[46] F.W. Hoeksema E.A.M. Klumperink B. Nauta V.J. Arkesteijn, R. Schiphorst and C.H.Slump. A combined receiver front-end for bluetooth and hiperlan/2. *3rd PROGRESS workshop on Embedded systems and Software*, October 2003.

[47] F.W. Hoeksema E.A.M. Klumperink B. Nauta V.J. Arkesteijn, R. Schiphorst and C.H. Slump. A software defined radio test-bed for wlan front ends. *3rd PROGRESS workshop on Embedded systems and Software*, October 2002.

[48] P.T. Wolkotte. Realization of a demonstrator for smallband jammer detector in a wideband radar signal. Master's thesis, University of Twente, Enschede, August 2003.