A Novel Algorithm for Reconfigurable Architecture for Software-Defined Radio Receiver on Baseband Processor for Demodulation

H. D. Nataraj Urs, R. Venkata Siva Reddy, Raveendra Gudodagi, K. M. Sudharshan, and B. N. Aravind

1 Introduction

The wireless communication industry is facing new challenges due to constant evolution of new standards (2.5G, 3G, 4G and 5G), 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 widespread 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 [2] (Fig. 1).

2 Background

2.1 Baseband Demodulation

In the current implementation, algorithms for demodulation are implemented on GPP (general purpose processor) hardware [3], and the analog front end of SDR is already made to be flexible and reconfigurable [4]. This work focuses

H. D. Nataraj Urs (✉) · R. Venkata Siva Reddy · R. Gudodagi · K. M. Sudharshan
School of Electronics & Communication Engineering, REVA University, Bengaluru, India
e-mail: natarajurs.hd@reva.edu.in; venkatasivareddy@reva.edu.in; raveendra.g@reva.edu.in; sudharshankm@reva.edu.in

B. N. Aravind
Department of Electronics and Communication Engineering, Rajeev Institute of Technology, Hassan, India

S. Awasthi et al. (eds.), *Sustainable Computing*,
https://doi.org/10.1007/978-3-031-13577-4_11

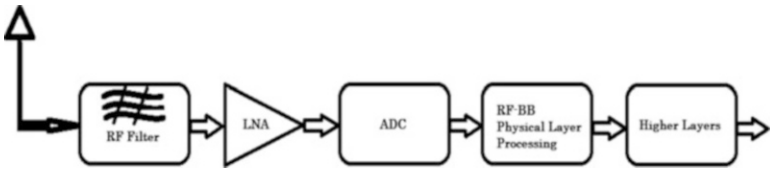

Fig. 1 SDR architecture

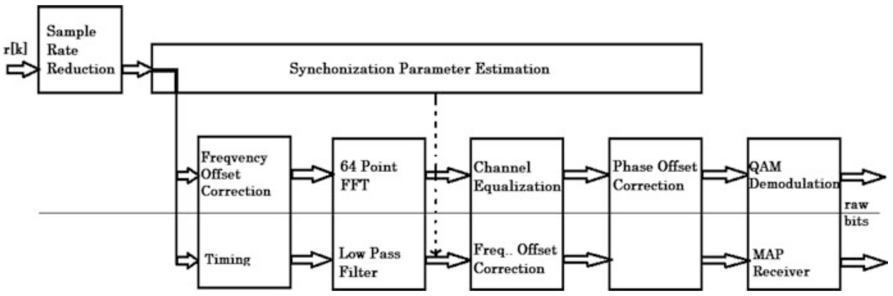

Fig. 2 Functional architecture of the receiver

on the hardware implementation of digital baseband part of the receiver (PHY (physical layer) only). Input data is coming in the BB receiver after the analog front end (including ADC (analog-to-digital converter)) at the rate of 80 MSPS (megasamples-per-second). The digital baseband part consists of a sample rate reduction block followed by digital demodulator block. The output from sample rate reduction block is fed to the digital demodulator part which demodulates the data stream digitally.

2.2 OFDM

After frequency offset correction, the first step is the inverse OFDM as shown in Fig. 2. The inverse OFDM is same as fast Fourier transform (FFT) operation. An OFDM symbol has a duration of 80 complex samples. Only 64 samples of them are needed for the FFT operation. 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 [1, 5], we chose to use radix-2 FFT in our implementation. Radix-2 FFT is performed using radix-2 butterflies and requires $64 \times \log_2(64)$ complex multiplications.

2.3 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 [6].

2.4 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.

2.5 QAM (Quadrature Amplitude Modulation) Demapping

The final step in demodulation is demapping. There are four constellations available: BPSK (binary phase-shift keying), QPSK (quadrature phase-shift keying), 16-QAM and 64-QAM. Each of these constellations has a different number of bits per complex symbol. Demapping can be done using lookup table. In the lookup table, all possible subcarrier values for a certain mapping scheme are defined [7]. For BPSK, two 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 (Table 1).

A 64-QAM symbol has $2^3 = 8$ possible values for both the real and imaginary parts. Demapping can be implemented by generating an index for a table. So demapping requires two comparisons (border checking), one addition, one multiplication and one lookup table.

Table 1 Computational requirements for receiver

Function	Data rate	Number of multiplications	Number of additions
64-point FFT	16	153.6e6	76.8e6
Channel equalization	13	20.8e6	10.4e5
Phase shift correction	12	19.2e6	10.4e6
64-QAM demapping	12	9.6e6	9.6e6

3 Algorithm Analysis

The algorithm domain of the SDR includes baseband demodulation algorithms. In this work, we are dealing with the hardware implementation of the channel selection block of receiver and OFDM block. (The halfband filter block and matched filter block are combined together into one channel selection block in the receiver.) For this purpose, our first step is to perform the dataflow analysis in various computations of these algorithms.

3.1 Dataflow for Channel Selection/FFT

The first block in the baseband demodulation receiver is a 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 the first 16 samples in each block. So we need to take 64 samples out of these 80 samples.

The first block in the baseband demodulation receiver is a channel selector/low-pass filter (LPF). This is required to select the desired 1 MHz bandwidth (BW) channel. As we analysed, the complexity and data computation unit of FFT block are similar to LPF section [8]. So in our implementation, we propose to combine FFT with LPF. But direct implementation of LPF is computationally intensive. The input data is first passed through two linear phase halfband filters. Each halfband filter decimates data by factor 2. These halfband filters help in reducing the order of matched filter. Also matched filter can be designed to be linear phase. In this way, the number of computations can be reduced further. A simple schematic for channel selector section is shown in Fig. 3.

3.2 Signal Flow Graph for FIR/FFT

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

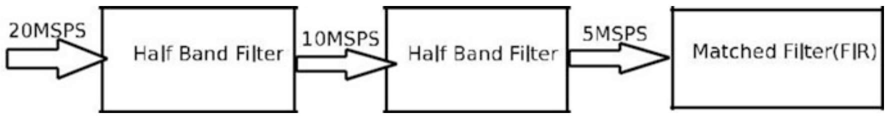

Fig. 3 Channel selector section of Bluetooth

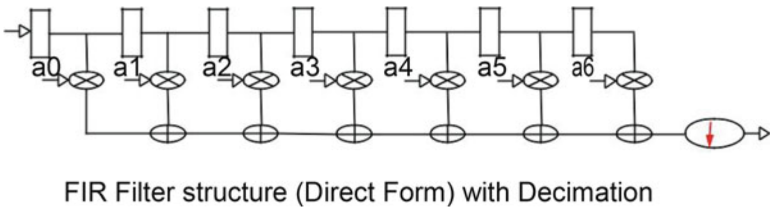

Fig. 4 Direct form FIR filter

Halfband Filter

Input data stream is filtered through halfband filters before doing low-pass filtering. There are two halfband filters. Each halfband filter is of seventh 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 that the number of multiplications (corresponding to the 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 seventh-order FIR filter can be represented as in equation:

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

Its critical path contains one multiplier and six adders. A direct form implementation of such filter is shown in Fig. 4.

The transposed form of above filter is shown in Fig. 4. Its critical path contains one multiplier and one adder only.

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, Eq. 1 can be rewritten as follows:

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

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

$$H_{(z)} = \left(a_0 + a_2\, z^{-2} + a_2\, z^{-4} + a_0\, z^{-6}\right) + Z^{-1}\left(a_3\, z^{-2}\right) \tag{3}$$

The simplified structure, which is computationally most efficient in terms of speed of operation and in terms of the amount of data path computations, is shown in Fig. 5.

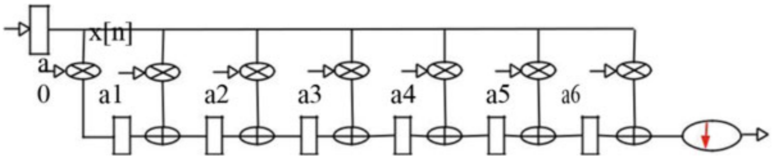

Fig. 5 Transposed form FIR filter

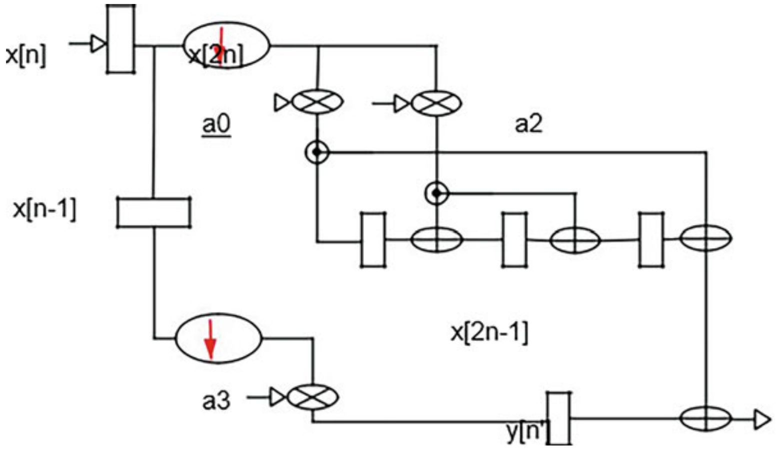

Fig. 6 Filter structure simplification

In this way, the 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 rates.

Moreover, it is important to notice that the filter structure above has a basic computation unit (shown in Fig. 6). The repetitive use of this unit realizes the filter. The basic operation can be described as multiply and add (Fig. 7).

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 here is of 17th order. The transposed form representation is shown in Fig. 8. The basic computation unit is the same as the one for halfband filters. Polyphase decomposition for efficient decimation and halfband properties are not applicable for this stage. So filter structure is corresponding to transposed form structure with linear phase. This means that the number of multiplications can be reduced by 2.

FFT
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

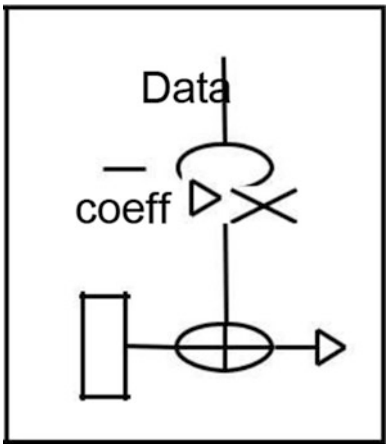

Fig. 7 Filter calculation unit

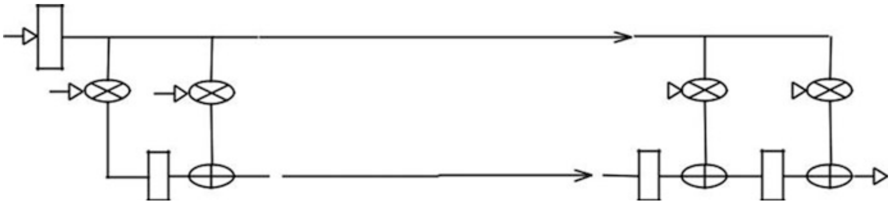

Fig. 8 Transposed form LPF for matched filtering

N-point DFT corresponds to the computation of N samples of the Fourier transform at N equally spaced frequencies, $\omega_k = 2\pi k/N$, that is, at N-points on the unit circle in the z-plane. The DFT of a finite-length sequence of length N is

$$X\,[k] = \sum_{n=0}^{N-1} x\,[n]\,{W_N}^{kn} \cdots \forall\ \ k \in \{0, 1, \ldots N-1\} \tag{4}$$

where ${W_N}^{kn} = \mathrm{e}^{-j2\pi/N}$. 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.

An objective choice for the best DFT algorithm cannot 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 the amount of required arithmetic operations (costs) and regularity of structure. Several other criteria (e.g. latency, throughput, scalability, control) also play a 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 data path computations in two receivers and simple control structure for receiver.

4 Architecture Design Approach

In this work, we propose a solution which is optimized for our specific algorithmic domain. Our algorithm domain is limited to the DSP (digital signal processing) 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) [9]. 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 as follows:

(a) 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.
(b) In the second step, we have designed the optimal control hardware for our algorithm domain. This is in contrary to various regular available hardware design approaches that put their attention towards the dominant data processing operations only.
(c) In the third step, we have identified the communication patterns in our algorithm domain as recommended. 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 buses are minimized to reduce capacitance and crosstalk effects. So point-to-point and local communication is preferred in our proposed architecture.
(d) In the fourth step, we have identified the memory requirements for our systems. In this step, we have identified things like how much RAM (random-access memory) and ROM (read-only memory) are needed, what are the memory bandwidth requirements, is it better to reuse the memory by using in-place computations, etc.

The proposed architecture comprises of nine homogenous data processing tiles, two 128 $\times$ 16-bit memory (RAM) tiles, one 64 $\times$ 16-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 data path elements to memories through the communication network. It also generates the control signals for the configuration unit. The architecture view of the system is shown in Fig. 9.

The proposed design is based on tiled architecture. A tiled architecture in which various tiles are connected by an on-chip network has a very modular design [10]. The design of a single processing tile is relatively simple and allows extra effort for power optimizations at physical level. To increase or decrease processing power of

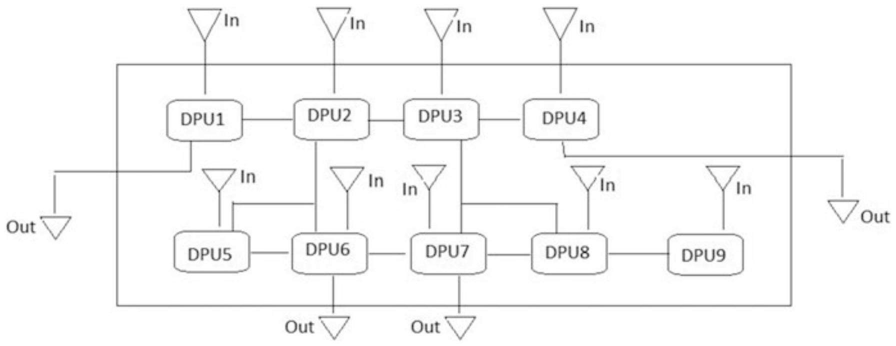

Fig. 9 Tiled architecture

our system, we can easily add or remove tiles. A simplified view of our tiled network is shown in Fig. 9.

4.1 Reconfigurability

The proposed design is reconfigurable within one clock cycle and supports the chosen subset of the SDR algorithms. So the algorithm domain of our design includes FIR filter, halfband filters and radix-2 FFT. These algorithms are also the most common algorithms used to benchmark a DSP system [9]. 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 (wireless local area network) systems use either phase modulation or OFDM-based modulation [11]. 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.

4.2 Data Path

In the proposed design, data path consists of nine homogeneous 16-bit data processing tiles called data processing units (DPUs). The detailed view of our data path is shown in Fig. 10. A single DPU is depicted in Fig. 10. 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 Fig. 10.

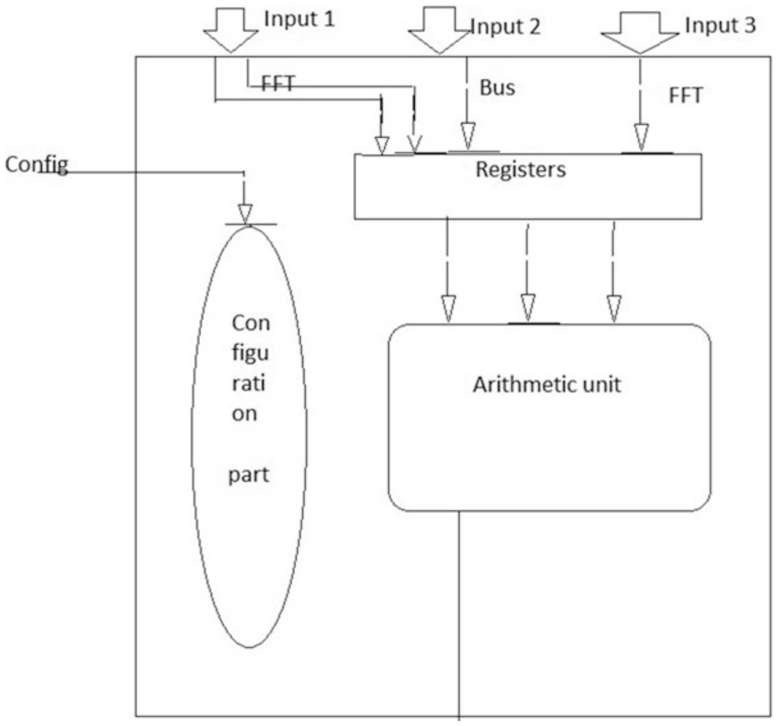

Fig. 10 A data processing unit (DPU)

4.3 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 three sets of 16-bit inputs:

- Input 1 set is used to read data either from left or from right neighbour into the registers. The ports corresponding to these inputs are named as ‘LHS’ and ‘RHS’.
- Input 2 set (bus 2) is used to read data from global bus of the system. There are two global buses in our system. Each global bus is providing the input to one row of DPUs.
- Input 3 set is connected to two point-to-point buses of the system. The ports corresponding to these inputs are named as ‘FFT bus’ and ‘global bus’.

Each DPU has the following two 16-bit outputs:

- First output (‘sideout’) is used to communicate with the adjacent left and right side neighbours.

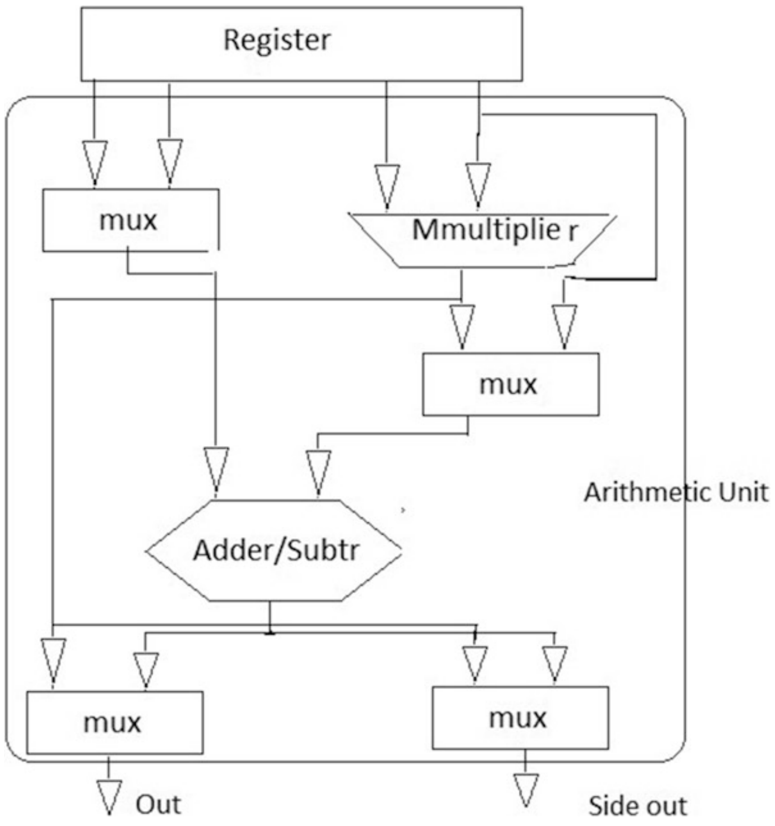

Fig. 11 Arithmetic unit (AU) of DPU

- Second output ('out') is used to communicate data over the system communication buses. To avoid bus arbitration output, 'out' is a tri-state output.

4.4 The Processing Part

The data processing capabilities of DPU are attributed to a 16-bit arithmetic unit (AU). A functional representation of the AU is shown in Fig. 11. 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.

4.5 The Storage Part

Each DPU comprises a set of 11 local data registers of 16 bits 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 data path units. This not only allows the reduction of critical path delay but also allows a straightforward implementation of transposed form FIRs.

4.6 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 [12]. The input to this section comes from the main configuration unit of the architecture.

4.7 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 dataflow 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 (correspondingly to fetch and decode an instruction before execution) found 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.

4.8 Configuration Unit

In the proposed architecture, reconfigurability is achieved by reconfiguration of the data path and reconfiguration of the communication network.

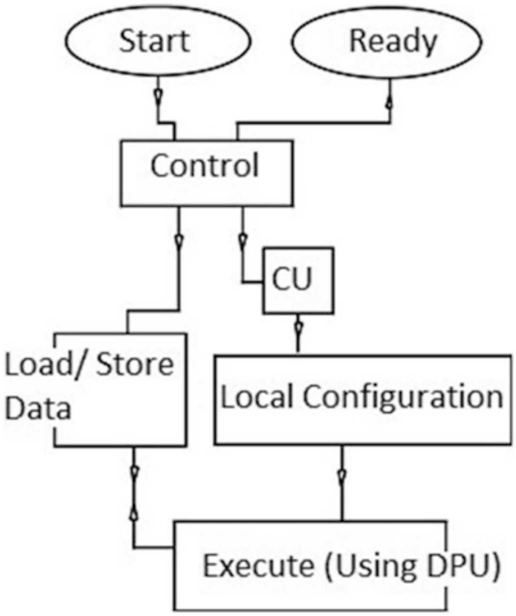

Fig. 12 Configuration unit block diagram

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 provides input to local configuration sections of various DPUs. The configuration of the data path and communication network is achieved within one clock cycle. This allows dynamic and static reconfigurations in the proposed architecture. To compute an algorithm, the 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 data path. This way, distributed control is achieved in the proposed architecture.

This is shown in Fig. 12. This facilitates high operating speeds and time-sharing of data and communication network. The low-overhead and dynamic reconfiguration allows time multiplexing of the processing part.

5 Algorithm Mapping

5.1 Mapping of Matched FIR Filter

The input data after halfband filtering and decimation is processed into 17th-order matched FIR filter. This means that we need 17 basic computations equivalent to a MAC operation (shown in Fig. 13). For each sample, our implementation can range

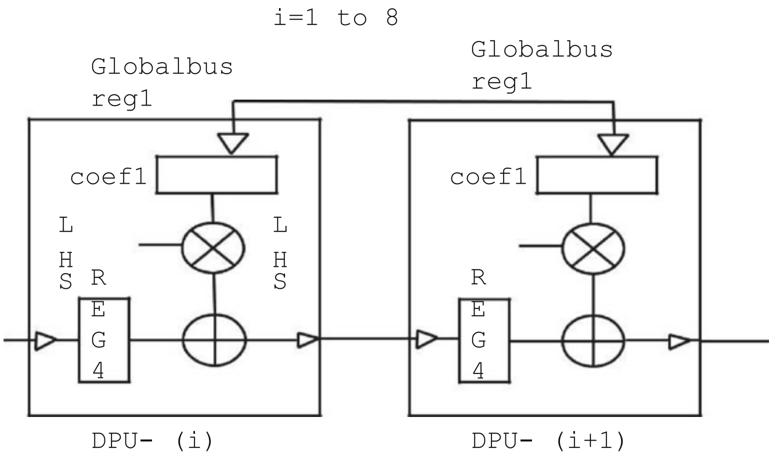

Fig. 13 First clock cycle in FIR mapping

from using 1 DPU, that is, 17 clock cycles for 1 computation, to 17 DPUs, that is, 1 clock cycle computation. We propose to use an intermediate solution which uses two clock cycles for one computation of real or imaginary data. Data processing of real and imaginary parts is done in alternate cycles. This means that there will be four clock cycles of computation for each data input. For this solution, we need nine DPUs. This decision is the main determining factor for choosing nine 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 global bus connecting DPU5–DPU9.
- Each AU is configured for multiply 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: input 1 is from stored data input corresponding to global bus input and input 2 is from ‘coef1’ value stored in another register within the DPU.
- Configure adder inputs of the AU: input 1 is from multiplier output and input 2 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 Fig. 13.

Similar to the halfband 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 onto 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 Fig. 13. This implementation allows us to use linear phase property, and hence, a 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.

5.2 Mapping of FFT

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 6 stages of computation. The basic butterfly was shown in Fig. 14. From the figure, it is clear that the real and the imaginary parts of a butterfly have a similar structure. For hardware mapping, we need two ROMs for storing real and imaginary parts of twiddle factors ($= \mathrm{e}^{-j2\pi k/N}$). There are two memory (RAM) units required for storing real and imaginary parts 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 [12]. So we will need to 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 $32 \times 6 = 192$ clocks of computations. Configuration of each DPU is described below and is also shown in Fig. 14:

- Configure DPU1 for addition; read data from FFTbus input and bus2 input; put the AU output into the A_{re} memory.
- Configure DPU2 for subtraction; read data from FFTbus input and bus2 input; and 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; and put the AU output onto the FFTbus input of DPU6 and DPU8.
- Configure DPU5 for multiplication; read data from FFTbus input and bus2 input; and put the AU output onto the sideout.
- Configure DPU6 for multiply and subtract; read data from FFTbus input and bus2 input into multiplier; and 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; and put the multiplier output and LHS input into the adder. Put the AU output into the B_{im} memory.
- Configure DPU9 for sleep mode.

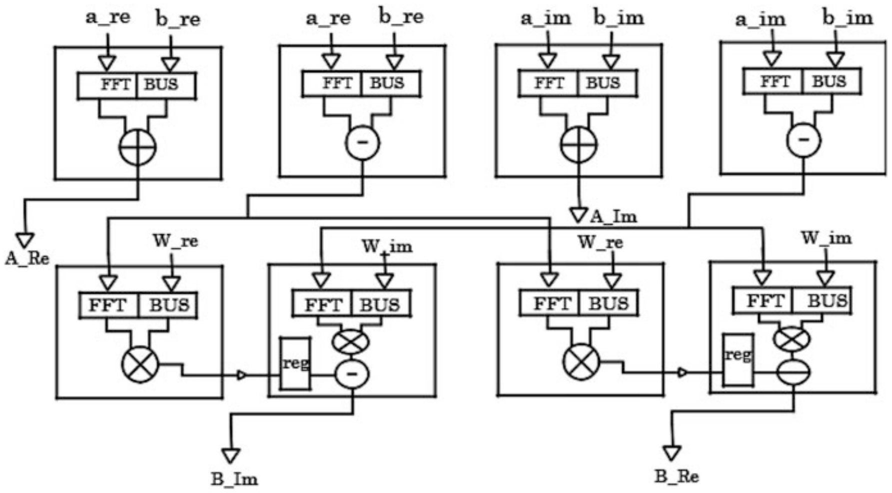

Fig. 14 One butterfly mapping

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.

6 Synthesis and Evaluation

This section elaborates the synthesis results and evaluates the design after hardware realization. The control section discusses the minimum speed requirements that the design must fulfil to meet the SDR receiver requirements [13]. The synthesis results for the proposed design are presented, and it summarizes the performance of Montium TP, when the chosen SDR algorithms are mapped onto it. The performance of the proposed system is compared with the performance of the Montium TP system for the chosen SDR algorithms.

6.1 Synthesis Results for the SDR Receiver

The results of synthesis are shown in Table 2. 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.

Table 2 Synthesis results for SDR receiver

Component	Area [μm^2]	Critical path [ns]
DPU (x9)	510,000	5.3
Control	26,000	3.8
CU	1300	–
Wiring	62,700	–
Resultant	600,000	5.3

The 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 [7, 14]. In the proposed design, we need two RAMs of 128 $\times$ 16 size each and one ROM of 64 $\times$ 16 size. From the above results, it is clear that the majority of area is consumed by the data path of the system. The control part consumes less than 5% of the total area.

6.2 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 two TPs to realize the various filter stages [15]. It is very difficult to exploit the linear phase property of the filters because FIR matched filter requires four 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, for example, a 64-point FFT needs 473 clock cycles and an FIR filter of 20th order needs 270 clock cycles.

The Montium TP occupies 2 mm^2 area in CMOS12 process from Philips. 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 eight 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 256 $\times$ 16 size and a ROM of 64 $\times$ 16 size, which will occupy an additional area of approximately 30,000 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 [10]. 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).

Table 3 Comparison of different architectures for butterfly computation

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**	0.24	188
	(excl. RAM and ROM)		

6.3 Comparison of Different Implementations

Table 3 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% of redundant hardware in our implementation.

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 (256 $\times$ 16) and ROM (64 $\times$ 16). This area will approximately be equal to 0.03 mm^2 in CMOS12 Philips process.

7 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 data path 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 3. It can be concluded that an area reduction of about 25–30% 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-

Table 4 Area requirements of SDR receiver

Component	Sum of separate implementations	Combined implementation
Computation area [μm^2]	840,000	600,000
RAM	352x16	256×16
ROM	64x16	64×16

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 and DSRA > GPP, while the flexibility of ASIC < ASIP, ASIP < DSR and DSRA < GPP.
- By introducing pipelining in the data path, we are able to perform computations at higher speed than a non-pipelined data path (Table 4).

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

A high-level description language, like SystemC, can be used to design VLSI (very large-scale integration) 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 systems use either phase modulation or OFDM modulation [5]. So the suitability of our system for phase-modulated and OFDM receivers implies that our design can be used in a number of systems.

8 Future Work

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 data path 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.

The computational complexity of 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.

References

1. A. Kapoor, A Reconfigurable Architecture of Software-Defined-Radio for Wireless Local Area Networks, 2005
2. A. Kumar Kaushik, A Comparative Study of Software Defined Radio and Cognitive Radio Network Technology Security, pp. 104–110
3. M.B. Blanton, An FPGA software-defined ultra wideband transceiver. Master Sci. (2006)
4. X. Zhang, J. Ansari, M. Arya, P. Mähönen, Exploring parallelization for medium access schemes on many-core software defined radio architecture. Proc. Second Work. Softw. Radio Implement. Forum - SRIF **13**, 37 (2013)
5. F. Buchali, F. Steiner, G. Böcherer, L. Schmalen, P. Schulte, W. Idler, Rate adaptation and reach increase by probabilistically shaped 64-QAM: An experimental demonstration. J. Lightwave Technol. **34**(7), 1599–1609 (2016)
6. C. Zhang, Dynamically reconfigurable architectures for real-time baseband processing, no. May. 2014
7. L. Zhao, H. Shankar, A. Nachum, 40G QPSK and DQPSK modulation, Inphi Corporation, 2008
8. P. Dong, C. Xie, L. Chen, L.L. Buhl, Y.-K. Chen, 112-Gb/s monolithic PDM-QPSK modulator in silicon. Opt. Express **20**(26), B624–B629 (2012)
9. H.D. Nataraj Urs, V.S. Reddy, Implementation and analysis of low frequency transceiver for SDR platforms. J. Adv. Res. Dyn. Control Syst. **10**(04–Special Issue), 1–9 (2018)
10. C.Y. Chen, F.H. Tseng, K. Di Chang, H.C. Chao, J.L. Chen, Reconfigurable software defined radio and its applications. Tamkang J. Sci. Eng. **13**(1), 29–38 (2010)
11. P. Suarez-Casal, A. Carro-Lagoa, J. A. Garćia-Naya, L. Castedo, A multicore SDR architecture for reconfigurable WiMAX downlink, in Proceedings of 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD) 2010, no. Cc, pp. 801–804, 2010
12. H. Goyal, J. Saxena, S. Dewra, Performance evaluation of OWC using different modulation techniques. J. Opt. Commun. **37**(4), 33–35 (2016)
13. V. Kumar, H.D. Nataraj Urs, R.V.S. Reddy, Software defined radio: Advancement to cognitive radio and basic challenges in spectrum sensing. Asian J. Eng. Technol. Innov. **4**(7), 166–169 (2016)
14. C. Chaitra, H.D. Nataraj Urs, Performance of SDR transceiver using different modulation techniques. Int. J. Adv. Eng. Res. Sci. **3**(5) (2016). ISSN: 2349-6495
15. M.S. Karpe, A.M. Lalge, S.U. Bhandari, Reconfiguration challenges & design techniques in software defined radio. Int. J. Adv. Comput. Res. (2013)