A Reconfigurable Tile-Based Architecture to Compute FFT and FIR Functions in the Context of Software-Defined Radio

Ajay Kapoor,* Sabih H. Gerez,+ Fokke W. Hoeksema and Roel Schiphorst

Signal and Systems (EWI-SAS) Group, Department of Electrical Engineering, University of Twente, The Netherlands

E-mail: s.h.gerez@utwente.nl

Abstract— *Software-defined radio* **(SDR) is the term used for flexible radio systems that can deal with multiple standards. For an efficient implementation, such systems require appropriate reconfigurable architectures. This paper targets the efficient implementation of the most computationally intensive kernels of two significantly different standards, viz. Bluetooth and HiperLAN/2, on the same reconfigurable hardware. These kernels are FIR filtering and FFT. The designed architecture is based on a two-dimensional arrangement of 17 tiles. Each tile contains a multiplier, an adder, local memory and multiplexers allowing flexible communication with the neighboring tiles. The tile-base data path is complemented with a global controller and various memories. The design has been implemented in SystemC and simulated extensively to prove equivalence with a reference all-software design. It has also been synthesized and turns out to outperform significantly other reconfigurable designs with respect to speed and area.**

Keywords—**Software-defined radio (SDR), architectures for digital signal processing**

I. Introduction

The many standards used in radio systems and the tendency of standards to change over time created the necessity for radio architectures that have some degree of flexibility. The concept of *software radio* was proposed by Mitola [1] about one decade ago. In its ideal form, the received signal at the antenna is directly fed into an *analog-to-digital converter* (ADC) and processed entirely in the digital domain. Such an implementation is infeasible due to the power that such device would consume and other physical limitations [2, 3]. 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. Such a system is called a *software-defined radio* (SDR). An SDR has hardware that can be reconfigured to be used for different radio systems. It should at least support reconfigurability on behalf of a standard update. From a commercial point of view, SDR is interesting because of a shorter time-to-market, the added value of multiple-standard device with upgrade possibility and the wider set of customers that can be served with the same device.

The *analog front-end* of an SDR system, the part between the antenna and the ADC, will consist of some low-noise amplifier, a mixer and filters each having some degree of configurability. The *digital back-end*, that converts the output of the ADC to a received stream of bits, can be designed in many different ways [4]. In one extreme, the use of a general-purpose microprocessor offers a high degree of flexibility at the expense of area and power inefficiency. In the other extreme, dedicated back-ends can be built for each standard to be supported. This may be optimal from a speed and power point of view, but implies a large area overhead. In between these extremes, one can think of many architectures that offer opportunities of *hardware sharing* for the processing of multiple standards while minimizing the area and power overhead. Such architectures are the topic of this paper.

The research was carried out in the context of the *University of Twente SDR Project* which is presented in Section II. The receiver block diagrams, as given in Section III, were the starting point for architecture design as elaborated in Section IV. Implementation details are finally discussed in Section V.

II. The University of Twente SDR Project

In the years 2000-2004, the Signals and Systems Group took part in Project TES.5177 entitled *Development of a software-radio-based embedded mobile*

*Ajay Kapoor is currently with the Embedded Systems Architectures on Silicon (ESAS) Group, Philips Research, The Netherlands.

+The main affiliation of Sabih Gerez is with SiTel Semiconductor, Design Center Hengelo, The Netherlands.

ProRISC 2005 16th Annual Workshop on Circuits, Systems and Signal Processing, Veldhoven, The Netherlands, November 2005.

terminal funded by the *PROGram for Research on Embedded Systems & Software* (PROGRESS) of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the technology foundation STW focusing on the digital back-end [5] while the analog front-end was developed in parallel by the IC Design group [6].

To create a research challenge, it was decided to select two standards that are significantly different, Bluetooth [7] and HiperLAN/2 [8], and to try to reconcile both within an SDR framework. The relevant characteristics of the two standards are summarized in Table I.

A testbed consisting of a reconfigurable analog front-end and a digital back-end was built to prove the feasibility of the idea. The digital back-end is a Pentium 4 processor that receives its data at a rate of 20 MSPS from the front-end via an analog-to-digital converter and a programmable down converter connected to a PCI card.

A Pentium 4 processor offers a high degree of flexibility: it can implement any demodulation algorithm provided that the algorithm's computational requirements can be met by the processor. The price paid for this flexibility, however, is high: both the area and the power consumption are orders of magnitude higher than dedicated ASIC solutions for either of the standards chosen. It was therefore decided to investigate more efficient hardware architectures that still were sufficiently flexible to support the two standards mentioned. First results of this research are reported in this paper.

III. Receiver Block Diagrams

The full software implementation of the digital back-end was based on specific functional partitionings of receiver functions for both standards [5]. They are presented below together with the transmitter functions, which help to understand the receiver functionality.

A. Bluetooth Block Diagrams

The frequency spectrum available to Bluetooth 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 channels, each of them occupying a bandwidth of 1 MHz. GFSK (Gaussian frequency shift keying) is used to transmit data in a channel. This is accomplished

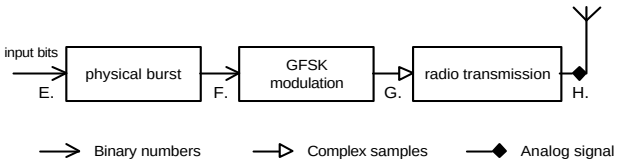

Fig. 1. Bluetooth transmission block diagram.

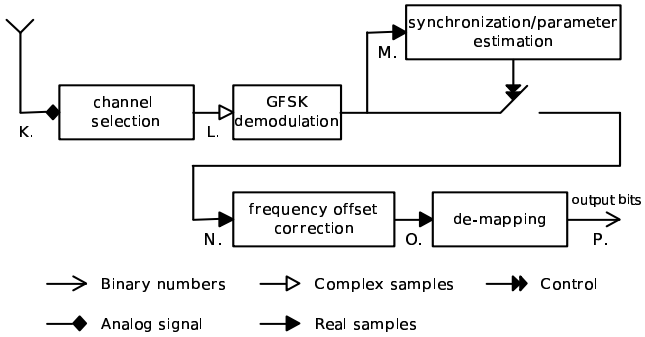

Fig. 2. Bluetooth receiver block diagram.

by letting the Gaussian-filtered bit pattern control the *voltage-controlled oscillator* (VCO) of the transmitter. This idea is illustrated in Figure 1.

The receiver is somewhat more complex as shown in Figure 2. The first step is to select the wanted Bluetooth channel and suppressing all others. This is necessary as a consequence of the fact that the analog front-end passes a large-bandwidth signal to the digital back-end in order to satisfy the HiperLAN/2 requirements. Channel selection is achieved by digitally mixing the wanted channel to zero IF followed by appropriate filtering and decimation. The next step is to demodulate the FM signal using a *maximum a-posteriori probability* (MAP) receiver based on the *Laurent decomposition*. Demodulation using the MAP receiver requires first passing the signal through a low-pass filter. This filter also acts as a matched filter for the 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.

B. HiperLAN/2 Block Diagrams

The transmitter block diagram for HiperLAN/2 is shown in Figure 3. The bit stream to be transmitted is fed to a *forward-error correction* (FEC) *coding* function after having been scrambled. Interleaving is followed by mapping the bits onto complex QAM (quadrature amplitude modulation) symbols. These

ProRISC 2005 16th Annual Workshop on Circuits, Systems and Signal Processing, Veldhoven, The Netherlands, November 2005.

TABLE I
COMPARISON OF THE PHYSICAL-LAYER CHARACTERISTICS OF BLUETOOTH AND HIPERLAN/2

	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
Maximal data rate	1 Mbps	54 Mbps
Channel spacing	1 MHz	20 MHz
Maximal power peak	100 mW	200 mW -1 W

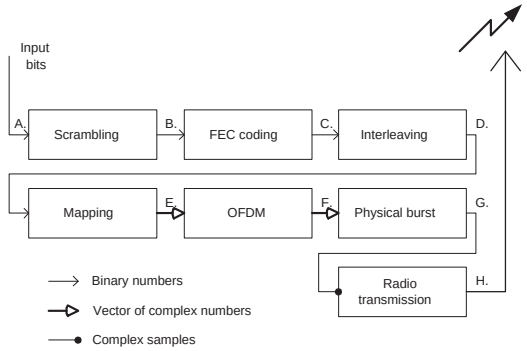

Fig. 3. HiperLAN/2 transmission block diagram.

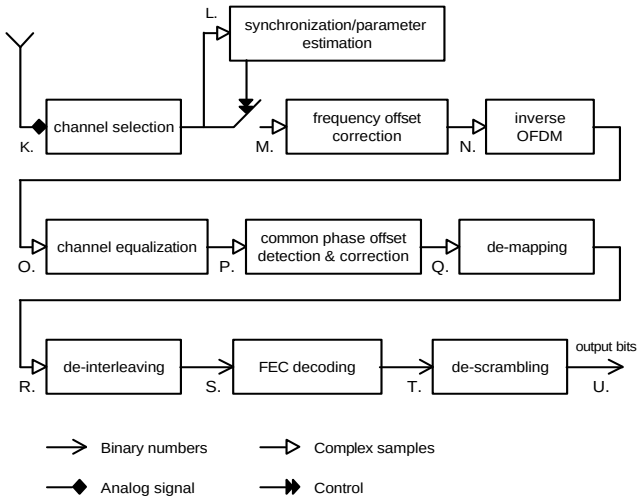

Fig. 4. HiperLAN/2 receiver block diagram.

symbols are modulated by means of *orthogonal frequency division multiplexing* (OFDM).

As shown in Figure 4, the receiver needs the inverse of the transmitter functions as well as functions for synchronization and parameter estimation, frequency-offset correction, phase-offset correction and channel equalization.

IV. Architecture Design

A. Bottleneck Identification

The first step in the design of a reconfigurable but efficient hardware architecture for both standards was to identify the most computationally intensive tasks (the remaining tasks can remain being implemented in software). This was done by analyzing the complexity of the functions in the receiver block diagrams presented above. The analysis is based on simply counting multiplications and additions taking parameter values for filter orders and sample rates from the full software implementation [5]. Details on this analysis can be found in [9]. Here, only the results for both standards are presented in Tables II and III. For each of the most important receiver functions, the tables give the (input and output) sample rates (in megasamples per second) and the number of multiplications and additions.

After an inspection of the tables it can be seen that (FIR) filtering related to sample-rate reduction and channel filtering is the most computationally intensive function of the Bluetooth receiver, while the 64-point FFT (OFDM demodulation) is the HiperLAN/2 function with the highest requirements. Not surprisingly, for both standards the bottleneck occurs in the first stages of the receiver chain. This implies that FIR filtering and FFT computations are likely candidates to share the same data path in a reconfigurable SDR architecture. For this reason, it was actually decided to focus on such a data path.

B. Architectural Alternatives

Reconfigurable architectures can be classified according to a number of criteria:

- *Granularity.* Devices with a small granularity allow

ProRISC 2005 16th Annual Workshop on Circuits, Systems and Signal Processing, Veldhoven, The Netherlands, November 2005.

TABLE II
COMPUTATIONAL COMPLEXITY OF THE BLUETOOTH RECEIVER

Function	**Data rate** [in/out MSPS]	**Number of multiplications**	**Number of additions**
Mixing	20/20	80e6	40e6
Decimation/Halfband filtering	20/5	420e6	360e6
Matched filter	5/5	170e6	160e6
Frequency offset correction	4.15/0.83	20e6	8.5e6
Viterbi decoder	0.83	29.9e6	21.6e6

TABLE III
COMPUTATIONAL COMPLEXITY OF THE HIPERLAN/2 RECEIVER

Function	**Data rate** [MSPS]	**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

reconfiguration at the level of single bits and elementary logic functions. They are known as *field-programmable gate arrays* FPGAs (see e.g. [10]). They are very flexible in the sense that almost any hardware can be built with them, but the overhead in area and speed of bit-level reconfiguration is high (both the function of elementary Boolean units as their interconnections is programmable). Their flexibility makes FPGAs very suitable for a prototyping platform. However, for efficient reconfigurability a more coarse-grained approach, where the unit of computation is a (simple) processor rather than an elementary Boolean function, may be more interesting. Examples of such systems are the *field-programmable processor array* of Nussbaum et al. [11] and the *field-programmable function array* of Jaap Smit [12] that evolved into the Montium platform [13].

- *Static vs. dynamic reconfigurability.* A system is called dynamically reconfigurable, if the transition from the execution of one function to another is (almost) smooth, if the system does not need to be stopped and restarted after reconfiguration. In some FPGAs this is e.g. achieved by reprogramming part of the array while another part is involved in an operational computation; when the reprogramming is ready, the new part takes over the computation. A system is called to be statically reconfigurable if it needs to be stopped for reconfiguration.
- *Application domain.* If the reconfigurable system is to be used for specific applications, its building blocks may be designed to optimally support such an application. A *multiply accumulate* (MAC) unit is e.g. often encountered in platforms that are targeting digital signal processing. One may also encounter architectures with an embedded RISC processor to complement the functionality of data-path oriented building blocks with a part that can handle control-dominated computations. Studies show that systems based on arrays of ALUs make a more efficient use of silicon area than systems based on arrays of microprocessors for applications in the signal-processing domain [14].
- *Homogenous vs. heterogeneous structure.* Homogeneous platforms in which the same building blocks are repeated in one or two dimensions have the advantage of simplicity and regularity. On the other hand, heterogeneous systems in which RISC processors, FPGAs, custom data paths, etc. are combined, may offer the best opportunity to efficiently map a wide range of applications. Rabaey was one of the first to advocate such heterogeneous systems [15]. A similar setup can also be found in the *adaptive system-on-chip* architecture developed at the University of Massachusetts, Amherst [16], where each building block is called a *tile* and tiles have a standardized interface for communication with their neighbors.
- *Scalability.* An architecture is called "scalable" if the computational power of the system can be in-

ProRISC 2005 16th Annual Workshop on Circuits, Systems and Signal Processing, Veldhoven, The Netherlands, November 2005.

creased by increasing the number of regular building blocks in the architecture.

When it comes to hardware that should efficiently support the entire physical layer of both Bluetooth and HiperLAN/2, a heterogeneous platform is the best choice. However, if the design is limited to hardware support for the computationally most expensive parts, viz. FIR filtering and FFT, one can make the domain-specific design considerations that are presented below. The resulting architecture can more or less be qualified as coarse-grained homogeneous, scalable, dynamically reconfigurable and application specific.

C. Design Considerations

Once the most computationally intensive parts of the receiver have been identified, the next step is to analyze these parts in more detail. The local structures in their data flow hint at the data path to be constructed. The overall succession of computations leads to insight about the requirements to local and global controllers.

As mentioned earlier, the *halfband filtering* related to decimation and the *matched filter* that are the most computationally intensive parts of the Bluetooth receiver, are implemented by means of *finite impulse response* (FIR) filters. If the transposed form [17] of a FIR filter is implemented, the *computational kernel* can be expressed as:

$$w_k[n] = h_k x[n] + w_{k-1}[n]$$

where $x[n]$ is the filter input data stream, the h_k are the filter coefficients, and the $w_k[n]$ represent the data streams at the internal registers (the filter output $y[n]$ equals $w_n[n]$). Obviously, the computational kernel consists of a multiplication followed by an addition. For complex-valued signals that occur in the Bluetooth demodulator, the kernel does not change: the real and imaginary components of the signals can be separately filtered as the filter coefficients are real valued.

The fast Fourier transform (FFT) that is the most computationally intensive part of the HiperLAN/2 receiver, has the following radix-2 *butterfly* as computational kernel when the *decimation in frequency* decomposition [17] is chosen:

$$X_k[p] = X_{k-1}[p] + X_{k-1}[q]$$
$$X_k[q] = (X_{k-1}[p] - X_{k-1}[q])W$$

In these equations, all numbers are complex valued, including the *twiddle factor* W which is a number on the unit circle whose value depends on the butterfly stage. This means that the total number computations in this kernel consists of four multiplications (one complex multiplication), three additions (one complex addition shown above plus one addition in the complex multiplication) and three subtractions (one complex one and one in the multiplication).

In the design of hardware to support both kernels, the following considerations have been made:

- For the sake of scalability (e.g. support for other filter orders), the hardware should consist of uniform *tiles*.
- Both the FIR kernel as the complex multiplication of the FFT have a data flow in which a multiplication is followed by an addition (or subtraction which is easily computed on an adder/subtractor). It is therefore wise to have a multiplier followed by an an adder in the data path.
- In order to keep the tiles simple, the subtraction that precedes the multiplication in the FFT, could be executed on a neighboring tile, rather than on the tile itself. This consideration can be combined with the pipelining of the butterfly implementation.
- Apart from the tiles that carry out the computations, there is a necessity for memories in the architecture. They are necessary for storing the input and output data (notice that the computation for the FFT are block based rather than stream based), as well as the coefficients and twiddle factors.
- Each tile will need some local storage in order to avoid that data is sent multiple times to the same unit.
- Both intra-tile as inter-tile communication should be reconfigurable up to a certain degree. This implies the use of multiplexers.
- The system needs a central controller that takes care of reconfiguring for either Bluetooth or HiperLAN/2 in a dynamic way (a switch from one standard to the other should not cost more than a few clock cycles). Actually, dynamic reconfiguration is essential for the Bluetooth implementation alone as different FIR filter structures have to be executed after each other in the same data stream.

D. Tile-Based Architecture

Considering the dimensions of the filters and FFT, an arrangement of 9 data-processing tiles turns out to be sufficiently powerful to meet the requirements of the two standards. This network is shown in Figure 5. Each tile is called a *data-processing unit* (DPU). As can be seen in the figure, the DPUs are arranged in

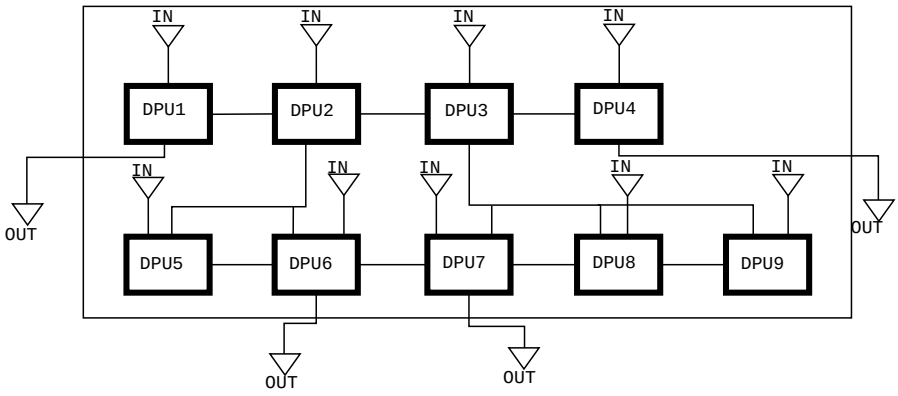

Fig. 5. The tile-based architecture.

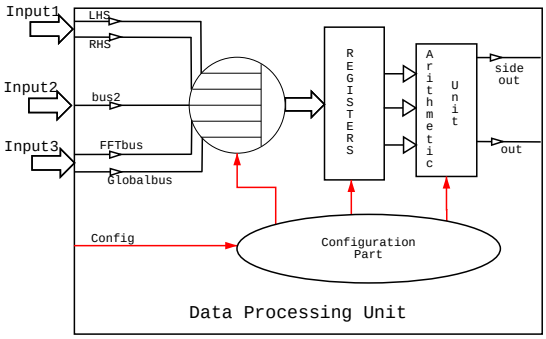

Fig. 6. Structure of a single data-processing unit.

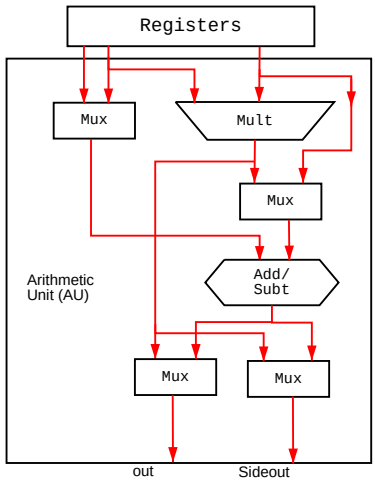

Fig. 7. The contents of the arithmetic unit.

two rows. Within each row, left-to-right and right-to-left communication is possible. In addition, some dedicated connections run from the top row to the bottom row.

The structure of a DPU is shown in Figure 6. It mainly consists of a register file, an arithmetic unit and a multiplexer to take data either from a neighboring DPU or a global bus (connected to memory not shown in Figure 5. It also has a local controller that is connected to the global controller.

The arithmetic unit, depicted in Figure 7, contains a multiplier, an adder/subtractor and a set of multiplexers. It is possible to configure the multiplexers in such a way that either the adder/subtractor or the multiplier is bypassed.

E. Entire Reconfigurable System

The complete reconfigurable architecture is shown in Figure 8. Apart from the the tile-based data path that was discussed above, it consists of:

- An *input buffer*. This is the memory in which the data samples produced by the analog front-end are stored after down conversion.
- A central *controller*. For the sake of simplicity,

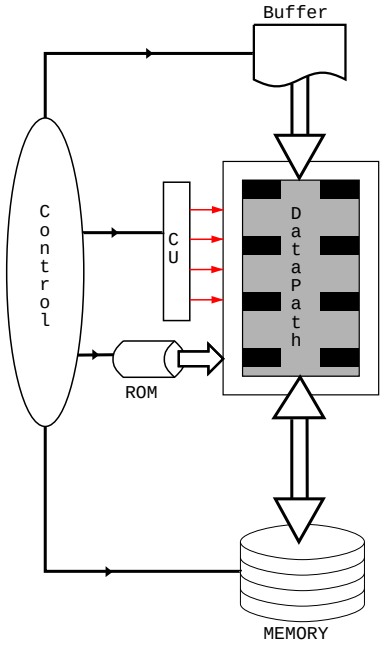

Fig. 8. The entire reconfigurable architecture.

the controller is supposed to be implemented as a state machine (it is not difficult to replace the state machine by a programmable controller). Note that none of the computations envisaged (FIR filtering or FFT) contain data-dependent calculations. Therefore, the data path does not need to generate status signals.

- A *configuration unit* (CU). This unit takes care of translating the controller state into detailed control instructions for each of the tiles in the data path.
- A *read-only memory* (ROM) used for the storage of FIR-filter coefficients and twiddle factors.
- A *random-access memory* (RAM). This memory is used for the storage of filtered data streams or the result of the FFT.

F. Algorithm Mapping

The halfband filters of the Bluetooth receiver proposed in [5] have order 7. When mapped on the architecture of Figure 5, the 4 DPUs in the top row are used for the real signal, and 4 (out of 5) DPUs of the bottom row for the imaginary signal. Then, the filter can be implemented using 2 clock cycles. In the first cycle data travels from left to right, as shown in Figure 9. In the second cycle, data travels from right to left. In this way, the data on the rightmost processor can remain local (it does not need to be transferred to the leftmost processor). In addition, no multiplications need to be performed in the second cycle as the filter has symmetric coefficients and the multiplication result is still locally stored in the tile. This results in power savings.

The 17th-order matched filter used for Bluetooth is mapped to be executed in 2 two clock cycles on the entire set of 9 processors. Because both the real and the imaginary parts of the signal need to be filtered, the total number of clock cycles for one invocation of this filter amounts to 4. This filter has symmetric coefficients too which means that the multipliers can remain inactive in the second clock cycle of a FIR-filter computation.

As far as the FFT is concerned, the design strategy was such that one butterfly would execute in a single clock cycle. The mapping of the butterfly on the architecture is shown in Figure 10. One should note that there are tiles that are configured to only perform an addition or multiplication. For the computation of the entire 64-point FFT, 32 executions of the butterfly are required.

V. Implementation Results

The design has been modeled in synthesizable SystemC and verified by extensive simulations using Synopsys CoCentric System Studio. Using reference input data streams, identical results were obtained as with the existing full-software implementation (except for the transition from floating-point numbers in the software to the fixed-point numbers in the reconfigurable hardware).

Synthesis took place in two steps. First, the SystemC descriptions were translated into Verilog using the public-domain tool `sc2v` [18]. Then, the Synopsys Design Compiler took care of the logic synthesis with a 0.18 micron ASIC technology. Actually, three different variants were built:

- The dynamically reconfigurable version with 9 DPUs described above; it has an estimated area of 0.60 mm^2.
- A dedicated 9-DPU version for Bluetooth only; unused connections between DPUs were removed in this version of the design and the controller was replaced by a controller that only supports Bluetooth; it has an estimated area of 0.55 mm^2.
- A dedicated DPU version that only supports HiperLAN/2 using the data path shown in Figure 10 and an appropriate controller; it has an area of 0.29 mm^2. The main gain comes from the fact that unused multipliers and adders in the DPUs do not contribute to the total area.

The area numbers above do not include the RAM and ROM areas.

Because there are many ways to implement hardware on silicon, one can only cautiously conclude that adding HiperLAN/2 functionality to the Bluetooth design, costs about 10% extra area. Putting the dedicated HiperLAN/2 receiver on chip would incur an overhead of some 50%. These numbers show that hardware sharing is really interesting.

Given the rates of the input data streams and the reuse factor of the data path (the number of clock cycles per data sample), one can derive that the system needs to operate at least at 80 MHz. Synthesis results for all three variants indicate that clock speeds up to 180 MHz are feasible with the design.

It is interesting to compare the performance of the design with other approaches. The FFT is a computation that has been widely implemented and therefore a good vehicle for comparison. Various FFT implementations are already compared with the Montium implementation in [13]. It is not surprising that our design outperforms is an order of magnitude smaller in

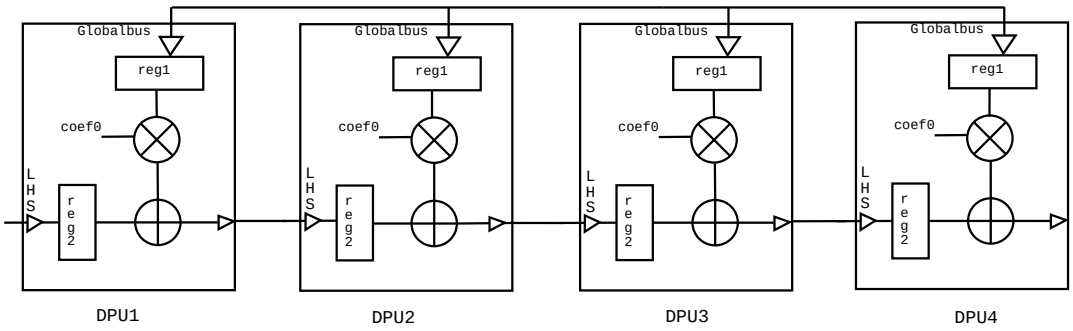

Fig. 9. Half of the 7th order FIR filter mapped on one row of the architecture.

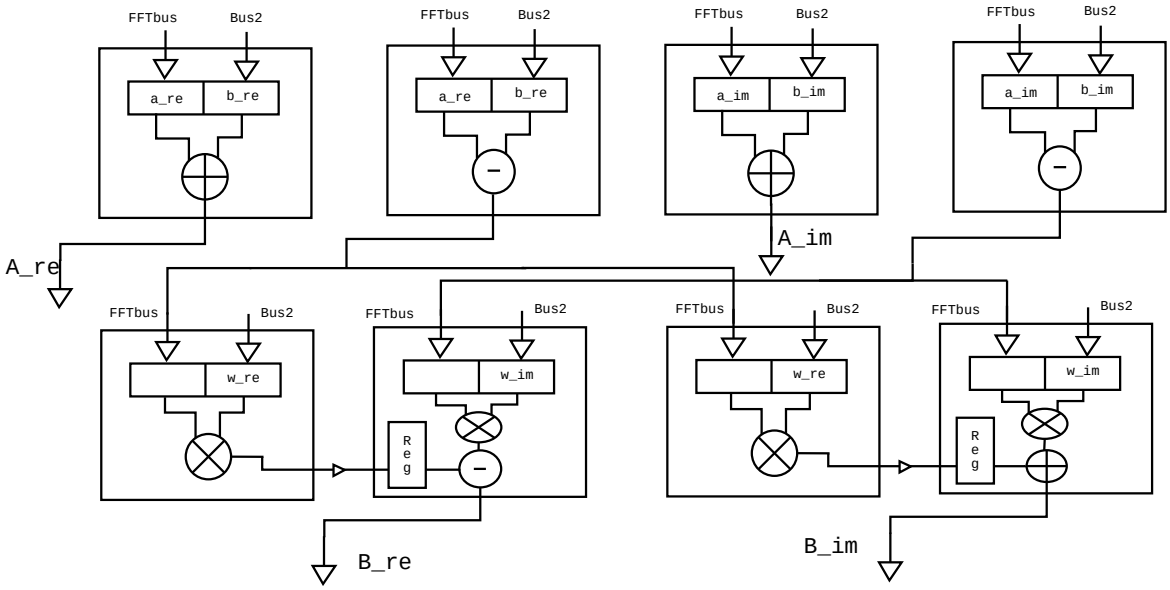

Fig. 10. Mapping of a single FFT butterfly on the tiled architecture.

area than implementations on the Montium or Avispa [19] platforms, as both have been designed to support a larger class of algorithms than FFTs and FIR filters. Our design is also somewhat faster. More details on performance comparisons can be found in [9].

VI. Conclusions

Starting point for the research presented in this paper was the all-software implementation for the Bluetooth and HiperLAN/2 receivers in the digital back-end of the Twente University SDR project. The goal was to provide a reconfigurable hardware solution such that the digital back-end could be implemented more efficiently from an area and power-consumption point of view while preserving support for the two significantly different standards.

As a first step the most computationally intensive kernels of the two receivers were identified by counting arithmetic operations. They turned out to be FIR filtering and FFT respectively. Then, the two kernels were analyzed and found to be implementable on a two-dimensional arrangement of 17 tiles each having a multiplier, an adder, local memory and flexible interconnection structures to communicate with their neighbors. This tile-based data path was connected to a global controller and global memories that act as input and output buffers.

The entire design was implemented in SystemC. After confidence in the correct functioning was assured by extensive simulations, the design was synthesized for 0.18 μm process. It turns out to occupy 0.6 mm^2 and to be able to run at 180 MHz, more than twice as fast as required by the system constraints. The performance of the design has been compared with other reconfigurable hardware platforms and shows to be both faster and smaller in area. The design also proves that hardware sharing for the support of multiple standards is feasible: the area for the recon-

figurable hardware is smaller than the sum of the areas for separate dedicated implementations of the two standards.

Acknowledgments

The authors would like to thank Paul Heysters and Gerard Rauwerda from the Montium research project at the University of Twente for the discussions on mapping radio algorithms on the Montium platform.

References

[1] J. Mitola. The software radio architecture. *IEEE Communications Magazine*, pages 26–38, May 1995.

[2] B. Nauta and C.H. Slump. On the design of a front-end subsystem for software-radio applications. In *PROGRESS 2000 Workshop on Embedded Systems*, Utrecht, The Netherlands, October 2000.

[3] R.H. Walden. Performance trends for analog-to-digital converters. *IEEE Communications Magazine*, pages 96–101, February 1999.

[4] J. Reed. *Software Radio: A Modern Approach to Engineering*. Prentice Hall PTR, Upper Saddle River, New Jersey, 2002.

[5] R. Schiphorst. *Software-Defined Radio for Wireless Local-Area Networks*. PhD thesis, University of Twente, Department of Electrical Engineering, September 2004.

[6] V.J. Arkesteijn, R. Schiphorst, F.W. Hoeksema, E.A.M. Klumperink, B. Nauta, and C.H. Slump. A combined analogue+digital software defined radio receiver front-end for Bluetooth and Hiperlan/2. In *5th Progress Symposium on Embedded Systems*, Nieuwegein, The Netherlands, October 2004.

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

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

[9] A. Kapoor. A reconfigurable architecture of software-defined radio for wireless local area networks. Master's thesis, Signals and Systems Group, Department of Electrical Engineering, Mathematics and Computer Science, University of Twente, March 2005. SAS04-048.

[10] J.O. Hamblen and M.D. Furman. *Rapid Prototyping of Digital Systems*. Kluwer Academic Publishers, Boston, 2000.

[11] P. Nussbaum, B. Girau, and A. Tisserand. Field programmable processor arrays. In M. Sipper, D. Mange, and A. Perez-Uribe, editors, *Evolvable Systems: From Biology to Hardware*, pages 311–322. Springer, Berlin, 1998. Lecture Notes in Computer Science, No. 1478.

[12] P.M. Heysters, J. Smit, G.J.M. Smit, and P.J.M. Havinga. Mapping of DSP algorithms on field programmable function arrays. In *10th International Conference on Field-Programmable Logic and Applications, FPL 2000*, pages 400–411, Villach, Austria, August 2000.

[13] P.M. Heysters. *Coarse-Grained Reconfigurable Processors, Flexibility Meets Efficiency*. PhD thesis, Department of Electrical Engineering, Mathematics and Computer Science, University of Twente, September 2004.

[14] D. Johnsson, J. Bengtsson, and B. Svensson. Two-level reconfigurable architecture for high-performance signal processing. In *International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA'04*, pages 177–183, Las Vegas, Nevada, June 2004.

[15] J.M. Rabaey. Reconfigurable processing: The solution to low-power programmable DSP. In *International Conference on Acoustics, Speech and Signal Processing*, 1997.

[16] J. Liang, A. Laffely, S. Srinivasan, and R. Tessier. An architecture and compiler for scalable on-chip communication. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(7):711–726, July 2004.

[17] A.V. Oppenheim and R.W. Schafer. *Discrete-Time Signal Processing*. Prentice-Hall International, London, second edition, 1999.

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

[19] J. Leijten, G. Burns, J. Huisken, E. Waterlander, and A. van Wel. Avispa: A massively parallel reconfigurable accelerator. In *International Symposium on System-on-Chip*, pages 165–168, 2003.