From ASIC to ASIP in RF Transceivers

Stefanie Castillo
Johannes Kepler University
Linz, Austria

***Abstract*—The fast evolution of Radio Access Technologies (RAT) in the wireless communication industry demands a flexible and adaptable design of the Radio Frequency (RF) Transceiver. In the current State-of-the-Art (SoA), the transceiver includes two types of processing elements (PEs). These are Application-Specific Integrated Circuits (ASICs) on the one hand which are controlled by one or many General Purpose Processors (GPPs) on the other hand. In the PE spectrum Application-Specific Integrated Circuits (ASICs) have the greatest performance and area/energy efficiency, nevertheless they lack of flexibility. Due to the need of flexibility in the RF Transceiver (TRX) we propose to re-evaluate the PE partition in the TRX, which implies re-evaluating the Hardware (HW)/Software (SW) partition by investigating other types of PE in its architecture. In this paper paper we take the fist step towards the flexibility road in the PE spectrum and compare performance and area/energy efficiency of ASICs and Application-Specific Instruction-Set Processors (ASIPs) implementing a typical Digital Signal Processing (DSP) algorithm of the TRX. We show how measurements match to the proposed PE spectrum regarding area/energy efficiency and performance.**

I. Introduction

The rapid evolution of the wireless Radio Access Technologies (RAT) in today's industry requires flexible, scalable, cost-effective and low time-to-market solutions for embedded Radio Frequency (RF) systems [1], [2]. The latter is quite a challenge for RF modems, since they must support fast-changing multi-band, multi-mode radio architectures. With shrinking Time-To-Market (TTM) and shortened Time-In-Market (TIM) of products, RF modems must meet, not only the area and energy efficiency requirements but also the right flexibility, re-usability and scalability requirements in their design.

Due to this need of flexibility the Hardware (HW)/Software (SW) co-design has been posed for re-evaluation in RF modems. New ideas have been flourishing for quite some years, e.g. the concept of a Software Defined Radio (SDR). Since its inception in the radio industry [3], SDR technology promised to solve the high level of dependency on dedicated hardware infrastructure of classic RF modems. The concept of SDR makes reference to a radio where the majority of the functionality is defined in SW/Firmware (FW) [4]. It promises to provide flexible, upgradeable and longer lifetime RF modems. Nevertheless, the more programmability we bring to the device, the more memory and processing capabilities we have to provide in its architecture, this results in less area and energy efficiency. Applying the concept of SDR in battery powered radio devices is quite challenging, since these devices usually run on limited energy resources. SDR challenges the paradigm of hardwired RF modems, and proposes to re-evaluate the HW/SW co-design of the RF modem to transition to flexible programmable architectures. The balance between HW and SW/FW must be exploited to satisfy the design constraints of an RF modem, such as cost, performance and energy/area efficiency [5].

If we want to bring more flexibility to the design of the RF Transceiver (TRX), we'll have to introduce programmable elements in its architecture. We must re-evaluate the HW/SW partition to bring more flexibility to the RF modem design. The design flow of wireless communication embedded systems starts with the system specification, partitioning, functionality design, algorithm design, architecture design and ends with the HW and SW implementation [6]. When it comes to mapping a bit-true algorithm to an architecture we have different architectural options within the processing element (PE) spectrum. A summary of the State-of-the-Art (SoA) PE spectrum will be carried out in Section III. As we will show, there is a variety of choices when it comes to processing elements available as today's SoA. We aim to evaluate architectures specifically suitable to map Digital Signal Processing (DSP) algorithms in the RF modem. DSP algorithms usually exploit parallelism regarding Data-Level Parallelism (DLP) and Instruction-Level Parallelism (ILP). Hence, the architectures we're looking for must exploit both DLP and ILP.

A key point is to chose the right PE architecture. To make a good choice we've decided first to investigate the area/energy efficiency and performance of different architectures of the SoA PE spectrum. The spectrum can go from dedicated HW architectures: Application-Specific Integrated Circuit (ASIC), to entirely general purpose processing architectures: General Purpose Processors (GPPs). If we move from ASIC towards more flexible architectures along the spectrum, our first step are the Application-Specific Instruction-Set Processor (ASIP) architectures. In this paper we intend to investigate how ASIP based design maps against the classical ASIC design, regarding area/energy efficiency and performance. We've chosen a typical algorithm in the Digital Front-End (DFE) of the TRX and designed/implemented two ASICs and three ASIPs. We present and compare synthesis results in nano-technology of performance and area/energy efficiency of these showing how these map to the PE spectrum.

The remainder of this paper is structured as follows: Section II gives an overview of the TRX, explains the subsystems and shows the PE distribution within the SoA TRX, finally it shows the need to adopt flexible architectures in the TRX design; in Section III an overview of the SoA of

processing architectures will be presented; in Section IV a review of related work regarding ASIP architectures is carried-out, and the chosen ASIP architecture for our investigations, the Transport Triggered Architecture (TTA) architecture is presented; Section V presents the targeted DSP algorithm within the DFE, the design methodology and architectures of the designed ASICs and ASIPs architectures; Section VI exposes the experimental results of area/energy efficiency and performance of the designed ASICs and ASIPs architectures. Finally Section VII concludes this paper and presents future work.

II. RF Transceiver

Usually an RF modem consist of three main interconnected blocks: the Baseband (BB) Integrated Circuit (IC), the RF Front-End and the RF IC (also referred to as RF Transceiver or TRX). Figure 1 shows an illustrative block diagram of a TRX. The main focus in our research is the TRX IC. The main subsystems within the TRX are the Transmit (TX) sub-system, the Receive (RX) sub-system, the RF digital interface (DigRF), the Phase Locked Loop (PLL) and the RF Control Architecture (RF-CA). Both, the TX and RX sub-systems have a digital part (DFE) and an analog part Analog Front-End (AFE).

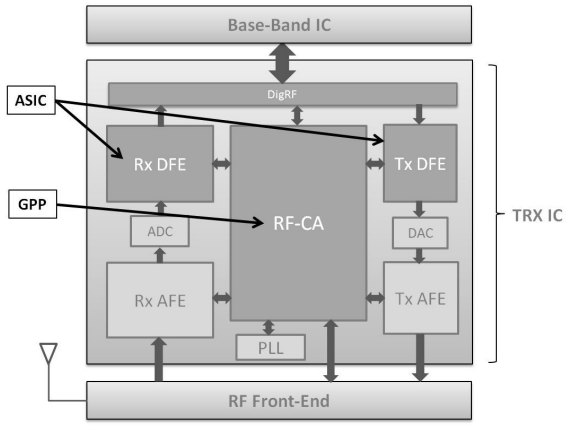

Figure 1. *TRX* system block diagram

Figure 1 shows how processing elements are usually distributed within the TRX. Most of the TRX is designed on dedicated HW, that is ASIC architectures, and the RF-CA is the one that holds the one or many GPPs to control the system. Regarding the HW/SW partition, the current approach is quite drastic: the RF-CA holds the SW and the rest of the chip is built on dedicated HW. The HW/SW partitioning is quite extreme as the two corners of the processing element spectrum are chosen. We detect there is a need to re-evaluate HW/SW partition in the TRX to include other types of processing architectures in its design. We can identify the great benefit this would bring to the flexibility of the digital part of the TRX. These benefits include: reusable HW being able to run various algorithms, ease the management of complexity being moved from the BB IC to the TRX, ease the adaptability of the TRX to adopt changes in communication standards or across different wireless platforms, enable runtime adaptability of the TRX, in between others. Hence our research goal is to re-evaluate the HW/SW partition of the TRX in order to incorporate as much programmability as possible, having in mind that the area/energy efficiency, performance, flexibility and accuracy requirements are met. In order to embark on this quest, let us first explore the SoA PE spectrum in the next section.

III. Processing Element Spectrum

In this section we will make a brief overview of the SoA PE spectrum. We're interested in the DSP aspect of the TRX and hence we make reference to digital signal processing architectures. By PE we make reference to a HW architecture which computes digital signal processing algorithms. Figure 2 shows the PE spectrum discussed in this section. We have based the diagram shown in Figure 2 on [7] and [6], but we have further extended it to incorporate other notions to be considered on PE.

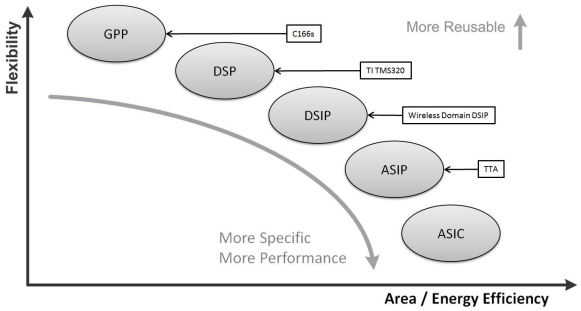

Figure 2. *Processing Element Spectrum*, based on [7]

We follow the definition of [8] when referring to *flexibility*, and adopt the correlation between *flexibility* and *re-usability*. With the difference that *re-usability* refers to the degree in which a PE can be used across different applications, not only behaviors. Greater *flexibility* and *re-usability* of a PE ensures shorter TTM and longer TIM. The more tailored a HW is, the more performance we can gain out of it as it is designed to execute a specific behavior and it can maximize its performance to the cost of loosing *flexibility*. We will measure *performance* in *Mega Samples Per Second (Msps)*, where each sample comprises of a pair of In-phase (I) and Quadrature (Q) symbols. On the horizontal axis, we show both energy and area efficiency growing rightwards, as we have seen that these two correlate.

ASIC are hardwired HW architectures, designed specifically to execute a very specific behavior/algorithm. ASIP are application specific processing architectures designed for a set of specific algorithms. These architectures are designed to execute a specific set of algorithms, all with more or less similar operations, by time-sharing the PE. An example of an ASIP architecture is the TTA architecture [9]. Domain-Specific Instruction set Processor (DSIP) is a domain specific processing architecture [10]. That means that it does not target

a set of algorithms, but ideally could execute any algorithm for a specific domain. The DSIP for the wireless communication domain was recently published in [6]. A Digital Signal Processor (DSPr) comes, in principle, packed with all possible computational needs of a DSP algorithm. A classic DSPr that has been proven successfully in the market for quite some years now is the TMS320 [11] of Texas Instruments (TI). A General Purpose Processor (GPP) is a general purpose oriented processor. The Infineon C166s processor [12] is a GPP that has been used to control the TRX.

A summary of the main benefits of dedicated architectures vs. programmable architectures is shown in Table I. We must re-evaluate the type of PE we have in the TRX in order to have the right balance to meet and satisfy all requirements. The first step on the ladder towards increasing flexibility are the ASIP architectures, and so let us explore related work regarding these architectures in the next section.

Table I
BENEFITS OF ASIC VS. PROGRAMMABLE ARCHITECTURES

ASIC	Programmable architectures
performance	flexibility
energy efficiency	short Time-To-Market
area efficiency	longer Time-In-Market
	adaptability
	re-usability

IV. RELATED WORK

Within the processing element spectrum, we've chosen first to investigate how ASIP architectures map against the classical ASIC approach. To give a full overview of the ASIP landscape is not in the scope of this paper, therefore we refer the reader to [13], [14], [15], [7] for further reference. Within the ASIP architectures, one that suits our needs is the TTA architecture. Since its fist publication in 1991 [9], the TTA architecture has been of great interest to the industry and research community. The TTA processor architecture can be described as an exposed data path Very Long Instruction Word (VLIW) architecture, and supports both DLP and ILP. It is organized as a set of Functional Units (FUs) with an interconnection network. Opposed to the classical operation-based programming-approach, the TTA architecture is programmed by specifying data transports within functional units of the processor and computation occurs as a side effect of data transports. The TTA is statically schedule at compile time.

One of the major bottlenecks in VLIW architectures is the register file. With the TTA approach one can overcome this, since in the TTA architecture one can move data from one FU to another without having to go trough the register file. This introduces an extra level in the Data Memory Hierarchy (DMH) of the architecture. Usually TTA instructions are quite big, since they must specify move operations, including source and destination/opcode ports, for each of the buses in the processor and hence can commonly be hundreds of bits wide. This is the case of most VLIW architectures and one of its biggest drawbacks. As the TTA architectures are a sub-set of the VLIW architectures, they support both DLP and ILP, making them suitable to easily map DSP algorithms. Vast work has been done with the TTA architecture for DSP applications [16], [17], [18], [19], [20]. This makes it a good anchor point for our investigations. We've chosen the TTA architecture to do our ASIP investigations. Figure 3 shows a diagram of a TTA architecture.

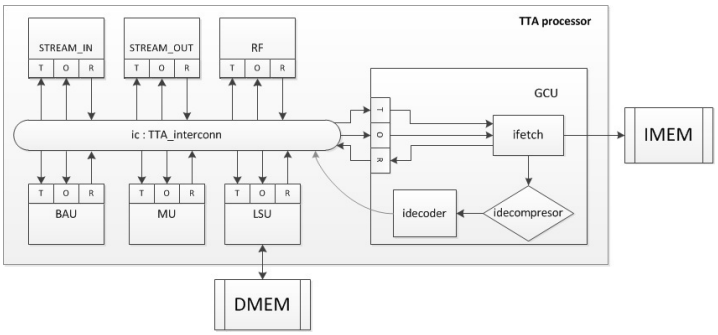

Figure 3. *Transport Triggered Architecture*

A TTA is composed of a set of FUs with an interconnection network and is controlled by the TTA General Control Unit (GCU). Two memories can be accessed from the TTA, that is the Instruction Memory (IMEM) and the Data Memory (DMEM). The gate-way to access the IMEM is trough the GCU FU, and to access the DMEM one must address the Load and Store Unit (LSU) FU. The GCU, LSU and even the register file are all treated as all other FUs within the TTA processor, that means that they're addressed the same way the computational units are addressed. The GCU fetches the instruction from the IMEM, decompresses them, decodes them and accordingly manages the transport of data across the FUs. The ports in the GCU can be used to write to the Program Counter (PC), to jump or branch in the program. The TTAs GCU implements a three stage execution pipeline: a fetch, a decode and a move stage. Each FU can execute different operations specified by the opcode in the destination port field of the instruction. An FU in the TTA architecture has at least three ports: the *operation* or *O* input port, *trigger* or *T* input port and the *result* or *R* output port. Each FU executes an operation, specified by the opcode. Operands should be moved to the *O* and *T* ports of an FU. The FU will only start to operate the very moment an operand is written to the *T* port.

In our ASIP implementations we've decided to choose a set of fixed FUs to design with. This limits the architecture design space and simplifies the design of the ASIP architectures. We've followed the vast amount of research carried out in [6], when selecting the types of FU we could include in our architecture. In [6] after analyzing many DSP oriented algorithms, they came to specify a fixed set of FUs that are sufficient to implement DSP algorithms. Of those functional units we've selected the ones shown in Table II to include in

our ASIP designs.

The Basic Arithmetic Unit (BAU) supports basic arithmetic operations such as: addition, subtraction and shift. The Multiplier Unit (MU) supports multiplications with variable operators. The Look-Up Table Unit (LUTU) supports trigonometric operations and other special operations with a Look-Up Table (LUT) approach. The Logic Unit (LU) unit supports logic and comparison operations.

Table II
BASIC FU FOR DSP ALGORITHMS

FU	Operations
BAU	add, subtract, shift
MU	multiplication variable operator
LUTU	trigonometric and special
LU	logic and comparison

V. ON TO FLEXIBILITY ORIENTED RF TRANSCEIVER DESIGN: FROM ASIC TO ASIP

The TRXs requires flexible architectures in its design to cope with the fast evolving wireless communication industry. The current SoA transceivers have two kinds of processing elements in their architecture, that is ASICs and GPPs. Withing the presented SoA PE spectrum, we can characterize these two as extreme opposites. The one, ASIC is highly energy and area efficient and has the best performance, but has the least flexibility and re-usability. On the other hand we have the GPPs which have great flexibility and re-usability but have the least area and energy efficiency. When moving towards higher flexibility up the ladder of the PE spectrum, the first step is the ASIP architectures, which are our focus of research in this paper.

We target to bring flexibility in to the DSP in the TRX. Within the low data rate of the DFE, we target a *notch filter* which runs at about 6.5 Msps. The notch filter is composed of four 2^{nd} order Infinite Impulse Response (IIR) filters and it has a direct form 2 structure. In order to evaluate and have a quantitative grasp of what it is to bring flexibility to DSP blocks within the TRX we've designed and implemented this filter in 2 ASIC versions and 3 ASIP versions. Each design targeted to give the maximum throughput it could with the given resources, and hence each of the implementations has a different performance. Our main goal is to compare the performance and area/energy efficiency across these.

ASIC_0 is a straight forward design of the algorithm, that means that we have used no optimization techniques. The architecture uses multipliers, adders and shifters to implement the functionality. This design is mostly combinatorial and has no control mechanism, hence the parallelism in this architecture is at its highest. This design executes the *notch filter* functionality at a specific and set frequency, and hence no other frequency can be filtered with this architecture. This is the least flexible architecture of all, as the behavior of the HW is tightly hardwired and cannot be changed. This is a good example of a dedicated-non-configurable ASIC architecture. The data-path of this architecture is 36-bit and it has a settling time of about 9 clock cycles.

In *ASIC_1* we aimed to include a little more flexibility in the design, and hence we've designed this HW such that the pass-band of the *notch filter* can be reconfigured. This architecture, even though is still an ASIC architecture, has a little reconfigurability, as the pass-band of the *notch filter* can be changed by register configuration. To enable the reconfigurability feature, this architecture has much more HW than the first ASIC architecture. In this architecture we've used a technique called Canonical-Signed-Digit (CSD) [21] to represent the weights in the filter, to improve the energy efficiency of the architecture. Furthermore, this architecture has no multipliers, and that means that all multiplications are carried out by shift and add operations, which also is expected to improve the energy efficiency of the architecture. Regarding the register file, we've used a circular buffer approach [22]. This architecture has a minimal control architecture that executes a fixed but register reconfigurable hardwired algorithm. The data-path of this architecture is 36-bit and it has a 16 clock-cycle latency.

A summary of the characteristics of each ASIC design is shown in Table III.

Table III
ASICS CHARACTERISTICS

ASIC_0	ASIC_1
no optimization techniques	CSD weights representation
pipelined architecture	circular-buffer register file
no control mechanism	pipelined with control mechanism
no reconfigurability	reconfigurable pass-band
36-bit data-path	36-bit data-path
latency of 9 clock cycles	latency 16 clock cycles

In the ASIP architectures we used two FUs to input and output data to the machine, these are the *stream_in* and the *stream_out* FUs. The *register file* in the architecture was used for two purposes: the one is to store partial results of computations and the other is to store the past samples (z^{-1}). Since the targeted DSP algorithm only requires basic arithmetic computations and multiplications, only the BAUs and MUs were used in the ASIP designs.

ASIP_0 has a TTA architecture implementation with minimum computational resources: 1 BAU, 1 MU and one bus. The one bus limits the architecture from fully utilizing its computation resources, as only one move operation can be carried out in a single clock cycle. That just throws away the benefit of parallelism of this type of architecture, this is basically a sequential machine as only one instruction can be executed per clock cycle. The data-path of this architecture is 32-bit long, the register file is 8 and the architecture has a 91 clock-cycle latency. It has an instruction size of 38-bit and an IMEM length of 88 rows. Figure 4 shows the architecture of ASIP_0.

ASIP_1 is a TTA architecture implementation, once again

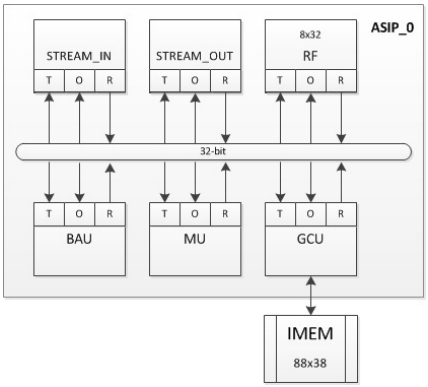

Figure 4. *ASIP_0* architecture

with minimum computational resources: 1 BAU and 1 MU, but in this case the architecture has four buses. The four buses in the architecture allow to do DLP and ILP, since four instructions can be executed with different data in a single clock cycle. The data-path of this architecture is 32-bit long, the register file size is 8 with 2 read/2 write ports and it has a 30 clock-cycle latency. The register file size is the same as in the previous design, but in this case the number or read and write ports of the register file were doubled to manage the parallelism of the architecture. It has an instruction size of 153-bit and an IMEM length of 27 rows. Figure 5 shows the architecture of ASIP_1.

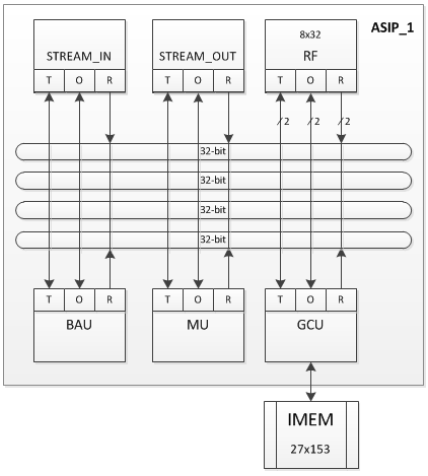

Figure 5. *ASIP_1* architecture

ASIP_2 is a TTA architecture implementation with more computational resources and interconnection capabilities: 2 BAUs, 2 MUs and eight buses. Hence this architecture exploits the best not only ILP but also DLP, for this algorithm. The data-path of this architecture is 32-bit long, the register file size is 12 with 2 read/2 write ports and it has a 21 clock-cycle latency. The register file had to grow in size compared to previous architectures, due to the parallelism of this architecture. It has a instruction size of 312-bit and an IMEM length of 18 rows. Figure 6 shows the architecture of ASIP_2.

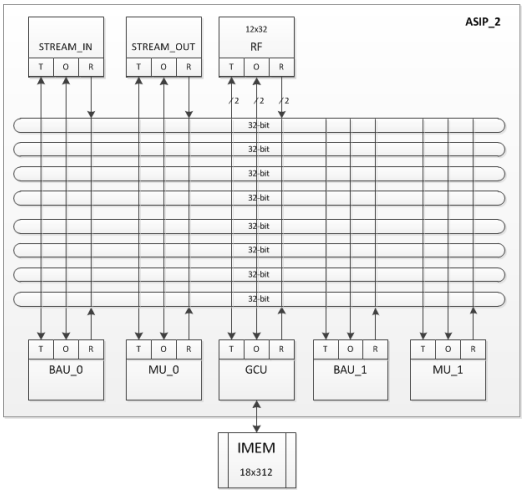

Figure 6. *ASIP_2* architecture

A summary of the computational resources, latency and memory utilization of each ASIP design is shown in Table IV.

Table IV
COMPUTATIONAL RESOURCES AND MEMORY UTILIZATION IN ASIPS

	ASIP_0	ASIP_1	ASIP_2
BAU	1	1	2
MU	1	1	2
32-bit buses	1	4	8
32-bit register file	8	8	12
instruction size (bit)	38	153	312
IMEM length	88	27	18
latency *(clk-cyc)*	91	30	21

VI. EXPERIMENTAL RESULTS

All architectures: ASIC_0,1 and ASIP_0,1,2 were designed and synthesized in nano-technology. The performance of the architectures was measured in Msps, with the assumption that the same amount of operations were needed per sample, as the algorithm is the same across samples. The experiment utilizes a 150 MHz clock and a targets a sample rate of 6.5 Msps. Each design gives the maximum throughput it can with the given resources, and hence each of the implementations has a different throughput. The aim is to match the 6.5 Msps or above. Each sample comprises of a set of I and Q symbols. The IMEM in the ASIPs is a Read-Only Memory (ROM) memory. Regarding area and energy, only relative comparisons are of interest and are shown in our results. For these relative comparisons the smallest value was set to relative 1 and the rest of the results were scaled accordingly.

Figure 7 depicts the performance of the architectures. We're looking for a HW architecture for a *notch filter* in the DFE running at a sample rate of 6.5 Msps. That means that any architecture that runs below the 6.5 Msps threshold does

not meet the performance requirement. ASIC_0 had the best performance, this was expected since it is the most dedicated HW architecture of our designs. Nevertheless, the behavior of this architecture cannot be changed or modified in any way, it is set to filter a specific pass-band and no other, hence it lacks of flexibility. The second design ASIC_1 shows less performance, but still good enough to meet the performance requirement of the filter. This diminished performance is easily explained due to the fact that this architecture has the benefit of reconfigurability and hence has a control mechanism, this accounts for the diminished performance. This control mechanism is deeply hardwired in the architecture and cannot be changed, but the the pass-band frequency of the filter can be reconfigured. ASIP_0 has the least of all performances and it is not good enough to meet the performance requirements of the filter. This architecture has little computational resources, only one BAU and one MU, but the bottleneck in this architecture are not its lack of computational resources, but the one bus. Even if it has the ability to execute parallel operations in the BAU and MU in the same clock cycle, it cannot exploit this as it has only one bus. This shows us how not only the amount of computational resources defines an architecture, but also its interconnection. A register file size of 8 was sufficient in this architecture, as there was no need for data storage due to the lack of parallelism of this architecture. This architecture takes 91 clock cycles to filter a given sample, and hence it clearly fails the 6.5 Msps threshold. ASIP_1 increases performance as the number of buses in this architecture allows it to leverage DLP and ILP, but even so it doesn't get past the 6.5 Msps threshold. This is due to too little amount of computational resources for the target algorithm. Once again a register file size of 8 was sufficient for this architecture. Finally, in ASIP_2 we doubled the number of computational resources but also the number of buses, to meet the right performance requirements to implement this filter. The register file size in this architecture was enlarged to 12. A register file size of 8 sufficed the previous ASIP_0 and ASIP_1 architectures, but in order to be able to handle the amount of parallelism of ASIP_2 we had to enlarge the size of the register file to 12. This shows that the increased parallelism in an architecture also demands the increase of the storage capabilities of the machine. Overall we can see that only ASIC_0, ASIC_1 and ASIP_2 meet the performance requirements of the filter, as these pass the 6.5 Msps threshold. An outcome of these investigations is that ASIC architectures have greater performance than ASIP architectures and we have shown this quantitatively.

We have measured the cell area of the synthesized netlist of each design. Figure 8 shows the relative comparisons of area efficiency of our architectures. ASIC_0 shows the best area efficiency of all, as this architecture is entirely designed to execute a specific algorithm only and it was designed with the exact amount of needed resources, hence it shows the best area efficiency. ASIC_1 has less area efficiency than the previous ASIC architecture. This is because it has more HW to enable the reconfigurability feature. ASIP_0 shows the least area efficiency of all, and hence has been scaled to relative

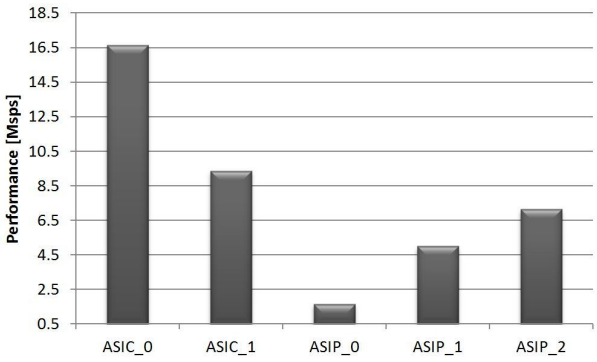

Figure 7. *Performance Comparison*

1 for comparison purposes. This architecture has not many resources, it has only one BAU, one MU and a single bus; hence it has not much the area, but it has little performance and hence the two combined show how little area-efficient this architecture is. ASIP_1 has greater area, even though the amount of computational resources stay put, the increase of area comes from the increase amount of buses. As much more DLP and ILP is exploited, the performance increases and hence so does the area efficiency of this architecture. Finally ASIP_2 shows less area efficiency than the previous ASIP. This is because in order to exploit the right amount of parallelism to meet the performance requirement of the design, much more computational resources and buses were included. Overall we can perceive that ASIC implementations outperform ASIP implementations regarding *area efficiency* as we had accurately shown in the PE spectrum of Section III.

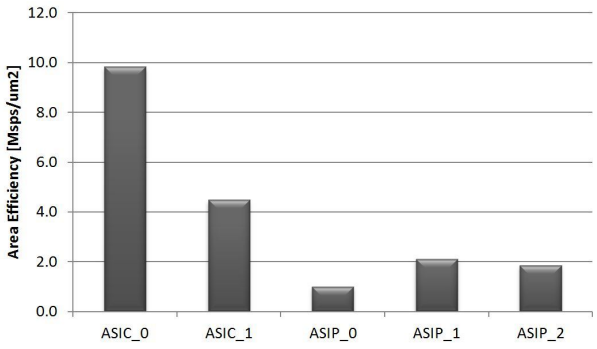

Figure 8. *Relative comparison of area efficiency*

Figure 9 shows the relative *energy efficiency* comparison of our architectures. ASIC_0 shows the greatest energy efficiency of all, also it had shown the greatest area efficiency, and it outperforms the other architectures by far. This clearly shows that the best approach to design area and energy efficient HW is to make it entirely customized to implement a very specific behavior, of course this implies having no flexibility at all. ASIC_1 shows less energy efficiency than the first ASIC due to the fact that this one has re-reconfigurability capabilities, this means more HW that dissipates more energy. ASIP_0 has the least of all energy efficiencies, and it has been scaled to

relative 1. In this architecture, even though the area is not that different compared to the resources utilized in other ASIC architectures, this architecture has a huge latency and hence bad performance, this accounts for it's low energy efficiency. ASIP_1 improves the energy efficiency compared to the first ASIP design, but the performance of this architecture is still not enough to meet our performance requirements. Finally, ASIP_2 has comparable energy efficiency to ASIC_1 and still meets the performance requirements. The measurements clearly show how ASIC architectures outperform the ASIP architectures regarding *energy efficiency*, we have shown this quantitatively. We can also recognize a correlation between area and energy efficiency as the PE spectrum had shown in Section III.

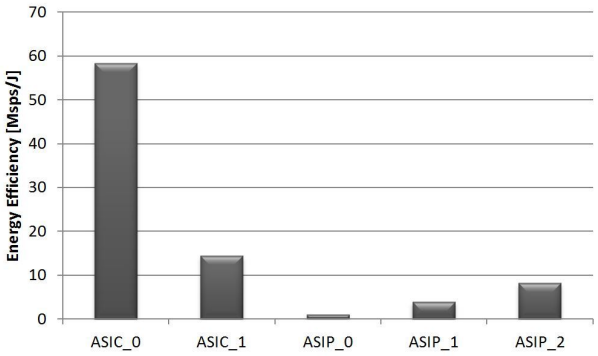

Figure 9. *Relative comparison of energy efficiency*

A summary of all relative comparisons is depicted in Table V.

Table V
PERFORMANCE AND RELATIVE AREA/ENERGY EFFICIENCY COMPARISON

	asic_0	asic_1	asip_0	asip_1	asip_2
Performance (Msps)	16.6	9.4	1.6	5	7.1
Area efficiency ($\frac{Msps}{um^2}$)	9.8	4.5	1	2.1	1.9
Energy efficiency ($\frac{Msps}{J}$)	58.3	14.4	1	3.9	8.2

VII. CONCLUSION

In this paper we've investigated how the first step towards higher flexibility in the PE spectrum: the ASIP architectures, map against the classical ASIC architectures. With this purpose we've designed two ASIC architectures and three ASIP architectures, to implement a typical digital filter in the DFE of the TRX. Our results show that ASIC architectures, as these are more tailored to execute a specific behavior, have the greatest performance and area/energy efficiency, ASIC architectures have greater performance than ASIP architectures, we've corroborated a correlation between area and energy efficiency as the PE spectrum had shown in Section III and we have shown all this quantitatively.

Future work will include to investigate how area and energy utilization are distributed within the building blocks of the TTA ASIP architecture, building blocks such as the control unit, computational FUs, register file, communication network, interface and memories. We will further continue our investigations towards bringing more flexibility to the TRX by investigating not only other architectures in the PE spectrum but considering also other subsystems of the TRX, aside from the DFE.

REFERENCES

[1] ITRS, "International technology roadmap for semiconductors," *ITRS 2011*, vol. Design, 2011. [Online]. Available: http://www.itrs.net/

[2] F. Luo, W. Williams, R. Rao, R. Narasimha, and M.-J. Montpetit, "Trends in signal processing applications and industry technology [in the spotlight]," *IEEE Signal Processing Magazine*, vol. 29, no. 1, pp. 184–174, January 2012.

[3] J. Mitola, "Software radios: Survey, critical evaluation and future directions," *IEEE Aerospace and Electronic Systems Magazine*, vol. 8, no. 4, pp. 25–36, 1993.

[4] T. Ulversoy, "Software defined radio: Challenges and opportunities," *IEEE Communications Surveys Tutorials*, vol. 12, no. 4, pp. 531–550, May 2010.

[5] J. Teich, "Hardware/Software codesign: The past, the present, and predicting the future," *Proceedings of the IEEE Special Centennial Issue*, vol. 100, pp. 1411–1430, May 2012.

[6] R. Fasthuber, "An energy-efficient architecture template for wireless communication systems," Ph.D. dissertation, Katholieke Universiteit Leuven, Belgium, 2012.

[7] K. Karuri and R. Leupers, *Application Analysis Tools for Asip Design: Application Profiling and Instruction-Set Customization*. Springer, June 2011.

[8] C. Haubelt, J. Teich, K. Richter, and R. Ernst, "System design for flexibility," in *Proceedings of the conference on design, automation and test in Europe, (DATE'02)*. Washington, DC, USA: IEEE Computer Society, 2002, p. 854.

[9] H. Corporaal and H. Mulder, "MOVE: a framework for high-performance processor design," in *Proceedings of the 1991 ACM/IEEE conference on Supercomputing (SC'91)*. New York, USA: ACM, 1991, pp. 692–701.

[10] F. Catthoor, P. Raghavan, A. Lambrechts, M. Jayapala, A. Kritikakou, and J. Absar, *Ultra-Low Energy Domain-Specific Instruction-Set Processors*, 1st ed. Springer, 2010.

[11] K.-S. Lin, G. Frantz, and J. Simar, R., "The TMS320 family of digital signal processors," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1143 –1159, September 1987.

[12] Infineon, "C166S v2, 16-bit synthesizable microcontroller," February 2006.

[13] K. Keutzer, S. Malik, and A. Newton, "From ASIC to ASIP: the next design discontinuity," in *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD'02)*, 2002, pp. 84–90.

[14] M. Gries and K. Keutzer, *Building ASIPs: The Mescal Methodology*, 1st ed. Springer, June 2005.

[15] O. Schliebusch, H. Meyr, and R. Leupers, *Optimized ASIP Synthesis from Architecture Description Language Models*. Springer, February 2007.

[16] P. Salmela, T. Jarvinen, J. Takala, and T. Sipila, "Scalable FIR filtering on transport triggered architecture processor," in *Proceedings of the International Symposium on Signals, Circuits and Systems (ISSCS'05)*, vol. 2, July 2005, pp. 493–496.

[17] T. Pitkanen, R. Makinen, J. Heikkinen, T. Partanen, and J. Takala, "Low-power, high-performance TTA processor for 1024-point fast fourier transform," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, ser. Lecture Notes in Computer Science, W. Vassiliadis and Hamalainen, Eds. Springer Berlin / Heidelberg, 2006, vol. 4017, pp. 227–236.

[18] V. Guzma, S. Bhattacharyya, P. Kellomaki, and J. Takala, "An integrated ASIP design flow for digital signal processing applications," in *Proceedings of the First International Symposium on Applied Sciences on Biomedical and Communication Technologies (ISABEL'08)*, October 2008, pp. 1–5.

[19] J. Haikkinen, "DSP applications on transport triggered architectures," Department of Electrical Engineering, Tampere University of Technology, Tampere, Finland, March 2001.
[20] P. Salmela, "Implementations of baseband functions for digital receivers," Dissertation, Tampere University of Technology, 2009.
[21] E. Backenius and E. Sall. (2005) Two's complement conversion to minimal signed digit code. Department of Electrical Engineering, Linkoping University, Sweden.
[22] S. W. Smith, *Scientist and Engineer's Guide to Digital Signal Processing*. Bertrams, January 1997.