An Architectural Analysis of Software-defined Radios

Tiago Rogério Mück*, Roberto de Matos† and Antônio Augusto Fröhlich*
Federal University of Santa Catarina (UFSC)
Florianópolis - Brazil
*Laboratory for Software and Hardware Integration (LISHA)
Email: {tiago,guto}@lisha.ufsc.br
†Department of Automation and Systems (DAS)
Email: rmatos@das.ufsc.br

***Abstract*—Nowadays there is a lot of wireless communication protocols being used, and this results in a series of difficulties for developers of systems that interacts with devices which can use different communication protocols. Software-defined radios (SDR) aims to solve this problem by using a software-based approach to provide flexibility on the implementation of the protocols physical layer. In this paper, an analysis of SDR implementation approaches is presented, and an architecture for SDR implementation is proposed. The architecture uses hybrid HW/SW components and programmable logic devices in order to enable the efficient implementation of software-defined radios from high level models. The results show that the architecture imposes a deterministic overhead to the SDR processing chain.**

I. Introduction

Nowadays there is a lot of wireless communication protocols being used, and this yields several difficulties for developers of systems that interacts with devices which can use different communication protocols. Providing adaptability to the communication protocols is a goal to be achieved. However, the hardware-based architecture of traditional radios imposes several limitations in providing this adaptability.

In traditional radios, each element of the receive/transmit hardware chain has a specific function, and the components are designed to work according to a fixed protocol. When some protocol's parameters change, the information is not transmitted correctly by the radio anymore. Hardware components needs to be replaced so the system can work with new standards. This lack of flexibility leads to higher development costs and time-to-market.

Software-defined radios (SDR) follow a software-based approach to eliminate the limitations imposed by the traditional radios. The idea behind SDRs is to use a multi-band system that allows transmitting and receiving in the frequencies of interest, called RF front-end, and does all the modulation and demodulation of the signal in software instead of hardware, thus allowing more flexibility on implementing the physical layer of the communication protocols.

The use of general purpose systems is very popular for implementing SDRs. Frameworks like the GNU Radio [1] can be used to easily implement SDRs from high level models on common PCs. Moreover, most of the protocols require a big amount of processing power to be implemented, and the requirements of cost, size and power consumption of most applications do not allow the use of a powerful PC to implement the system.

Usually, more specific solutions are used to implement SDRs, such as DSPs, GPPs with SIMD co-processors and programmable logic devices (PLD) like FPGAs. However, the use of this hybrid systems may present high project risks due to the difficulties in translating a high level model of the SDR to an implementation. To overcome these issues this paper proposes a new architecture for the

implementation of SDRs.

Following the *Application-driven Embedded System Design* (ADESD) methodology [2] and using the concept of *hybrid HW/SW components* [3], the proposed architecture allows the efficient implementation of SDRs in FPGAs by mapping parts of a high level functional model of the SDR directly to components that can migrate between the hardware and software domains in a transparent way.

II. SDR IMPLEMENTATION APPROACHES

There are many ways of implementing SDRs. Overall, these implementations can be generalized into three basic approaches: using general purpose processors, using programmable signal processing hardware, and using dedicated hardware.

The general purpose processor approach consists in the general SDR architecture. In this model, all the processing is made using a GPP, achieving the highest flexibility and ease of development. The GNU Radio [1] framework provides means of implementing SDRs on common PCs and is an example of an architecture that follows this approach. However, using general purpose hardware usually increases the costs of the system, and the overhead imposed by general purpose operating systems can forbid the efficient implementation of time-strict protocols [4].

In the second approach, the processing is made by programmable devices designed for specific functions. These devices includes: *digital signal processors* (DSP), GPPs with SIMD co-processors, and devices for specific functions like dedicated *digital up/down converters* (DUC/DDC). The hardware project on this approach tends to be more complicated and, typically, a FPGA is used to route the signals among the possible ways in the processing chain. In the last few years many architectures have been proposed based on this approach [5] [6] [7] [8]. The disadvantages of this architectures is the complexity of the software implementation. On SODA [5], for example, the synchronization between the GPP and the SIMD co-processors and the optimization of the signal processing functions were made manually in assembly code.

In the dedicated hardware approach, most of the processing chain is hard-coded on PLDs. The implementation of the desired protocol is made in hardware. This approach may look like the traditional radios way, but, since the hardware can be reconfigured, the radio can be considered a SDR. From all approaches, this one yields the better trade-off between the hardware cost, size and performance. However, the unfriendly environment for developing the signal processing functions and the lack of software support for controlling the data flow on the hardware may present high project risks and time-to-market.

Ideally, architectures that keep the high level of abstraction for the SDR developers should be used, for example the GNU Radio. However, as it was explained earlier, most of the applications cannot be implemented using PCs. Architectures that uses dedicated hardware may meet the applications requirements, but offer many difficulties to translate a high level model of the SDR to an implementation. Therefore, SDR developers should try to follow an approach that combines both the good trade-off of programmable hardware and the flexibility of architectures like GNU Radio.

III. AN ARCHITECTURE FOR SDR IMPLEMENTATION

Given the difficulties showed on the previous sections, this paper proposes a new architecture for SDR development. This new architecture uses the ADESD methodology [2] and hybrid hardware/software components [3] to enable the implementation of SDRs using a dedicated hardware approach. A framework allows the easy translation of a high level model of the SDR to an implementation.

The architecture is based on the *Synchronous Data Flow* (SDF) model, that is a very common model for designing digital signal processing applications. In this model, the SDR processing chain is abstracted as a flow graph, where the node represents processing blocks and the edges represents the data flow between the blocks. Figure 1 presents an overview of the proposed architecture. The processing blocks have their functions implemented using

hybrid components, so the processing blocks can be in hardware, software or both, in a transparent way to the SDR developers. The blocks are connected through FIFO channels, that are allocated by the architecture's framework in hardware or software, depending on the implementation of the blocks that it connects. An entity called *Flow Controller* controls the data flow between the blocks.

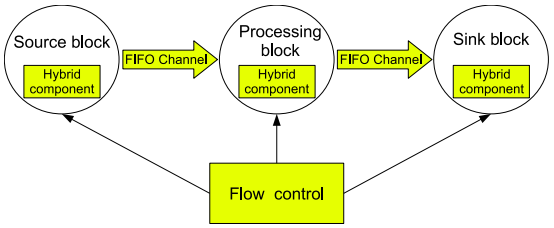

Fig. 1. Overview of the proposed architecture

As depicted in figure 1, the architecture defines three different kinds of blocks: the *source blocks* are responsible for the insertion of data on the flow graph. They usually abstract input devices like RF front-ends and ADCs. The *processing blocks* are responsible for the signal processing. Finally *sink blocks* consumes the data from the flow graph. It can abstracts output devices like a DAC and a RF front-end, or provide a callback mechanism for higher protocols layers.

The blocks have an interface that defines its number of inputs and outputs. For each execution of the block function, the interface defines the number of data elements consumed and generated. These informations are used by the flow controller to correctly connect the blocks and calculate the FIFO's size.

The flow controller is also responsible for controlling the data flow between the blocks. This is accomplished by creating a thread for each block, where its function is executed. The threads are synchronized using semaphores associated to the FIFO channels. They remain locked on the semaphores associated to the channels connected to the block's inputs. Every time an element is added to a channel, the *v()* method of its associated semaphore is called, unlocking the threads that consume the data from the channels.

Since this software control mechanism could be a bottleneck if both blocks are in hardware, the architecture allows hardware blocks communicate directly without software intervention. This is done through the deployment of HW FIFO channels that is supported by the hardware structure showed in figure 2. This structure has read ports, write ports and a set of FIFOs. The interconnection between this elements is defined by registers that are configured by software. All the blocks that are in hardware have its inputs connected to the structure's read ports, and its outputs connected to the write ports. This way any hardware block can be directly connected to any other hardware block through a hardware channel. The structure is also connected to the system bus, so hardware and software blocks can exchange data with each other.

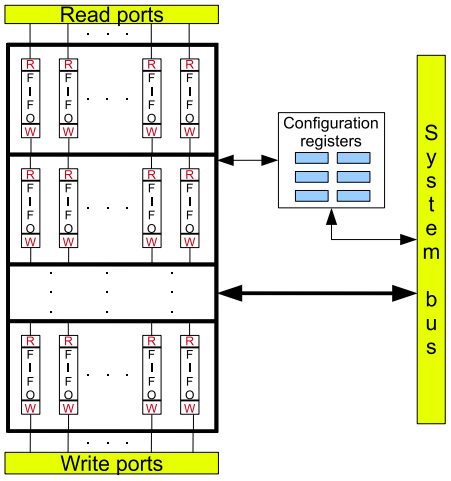

Fig. 2. Flow controller structure for HW channel allocation

IV. Results

Tests were made to evaluate the overhead imposed over the data flow by the control structures. The three structures shown in figure 3 were implemented to evaluate the increase of the overhead in terms of the number of blocks in serial, the number of blocks in parallel and the number of inputs/outputs of a block, respectively.

A source block generates samples that contain timestamps of when they were generated. The

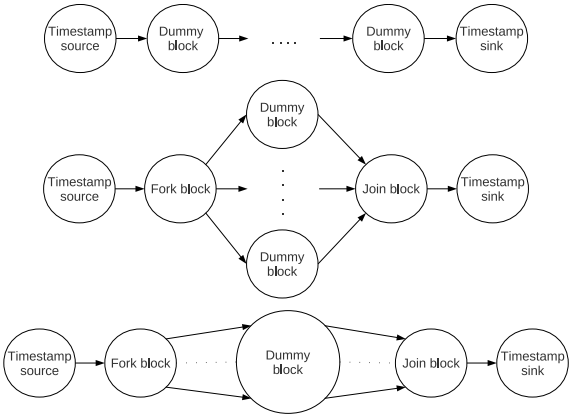

Fig. 3. SDRs implemented for the overhead evaluation

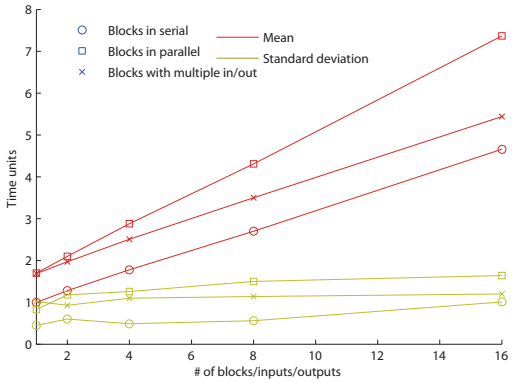

Fig. 4. Mean and standard deviation of the architecture overhead

dummy blocks are empty blocks that just propagate its inputs to its outputs. When the samples arrive at the sink block, the timestamps are compared with the current time, obtaining the time the sample took to go through the block chain. Since the blocks are empty, this time is the overhead imposed by the architecture to the data flow.

Figure 4 shows the results. Since absolute time values would be strongly related to the platform processing power, all the results were normalized in relation to the simplest case, where there is just one *dummy block* between the source and sink blocks. This way, more general results can be shown. The results show that, for all structures, the overhead grows linearly in relation to the increase of the number of blocks and the number of inputs and outputs. The standard deviation of the average overhead remained low compared to the mean values. This low standard deviation is especially important when implementing time-strict protocols, like TDMA based protocols. This is not the case of GNU Radio for example, where the standard deviation of the time a sample takes to get to the processing chain after being generated is higher then the average time [4].

V. Conclusion

This paper presented an analysis of SDR implementation approaches and proposed a new architecture for SDR development. This new architecture enables SDR development using PLDs by directly mapping components of a high level model to implementable components. The results show that the architecture imposes a very deterministic overhead over the SDR processing chain.

References

[1] GNU FSF project, "The GNU Radio," 2009, [Online; accessed 08-November-2009]. [Online]. Available: http://www.gnu.org/software/gnuradio

[2] A. A. Fröhlich, "Application-Oriented Operating Systems," Ph.D. dissertation, Technical University of Berlin, Berlin, 2001.

[3] H. Marcondes and A. A. Fröhlich, "On Hybrid Hw/Sw Components for Embedded System Design," in *Proceedings of the 17th IFAC World Congress*, Seoul, Korea, 2008, pp. 9290–9295.

[4] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, and P. Steenkiste, "Enabling MAC Protocols Implementation on Software-defined Radios," in *Networked Systems Design and Implementation*, 2009.

[5] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "SODA: A Low-power Architecture For Software Radio," in *ISCA '06: 33rd International Symposium on Computer Architecture*, 2006, pp. 89–101.

[6] J. Glossner, E. Hokenek, and M. Moudgill, "The Sandbridge Sandblaster Communications Processor," in *3rd Workshop on Application Specific Processors*, 2004, pp. 53–58.

[7] K. van Berkel, F. Heinle, P. P. E. Meuwissen, K. Moerman, and M. Weiss, "Vector processing as an enabler for software-defined radio in handheld devices," *EURASIP: Journal on Applied Signal Processing*, vol. 2005, pp. 2613–2625, 2005.

[8] Steven Kelem, Brian Box, Stephen Wasson, Robert Plunkett, Joseph Hassoun, and Chris Phillips, "An Elemental Computing Architecture for SD Radio," in *SDR '07: 2007 Software Defined Radio Technical Conference*, 2007.