A Flexible Bufferless H-ARQ Processor Based on Dataflow Scheduling

Pierre-Henri Horrein, Christine Hennebert
CEA,LETI,Minatec
Grenoble,France
Email: pierre-henri.horrein@cea.fr

Frédéric Pétrot
TIMA Labs
CNRS,Grenoble-INP,UJF
Grenoble,France
Email: Frederic.Petrot@imag.fr

***Abstract*—Flexible radio is a challenging way to implement communication standards. In theses standards, Hybrid ARQ (H-ARQ) is admitted as a usual error control protocol. H-ARQ is a cross-layer protocol, offering a number of different possible versions, with multiple instantiations running concurrently. In this context, designing a flexible H-ARQ component is a necessity. This paper presents an H-ARQ processor able to cope with the possible versions of the protocol and any number of instantiations. Based on a modified hardware/software partitioning, it is able to dynamically reconfigure its operation mode. Data representation is used to create codewords, associated to an operation. A new buffer management scheme, based on software buffers and independent of the number of protocol instantiations, is proposed. The resulting architecture is able to process any H-ARQ protocol with no throughput penalty, in MIMO or SISO environments, and to support concurrent standards.**

***Keywords*-H-ARQ; flexible radio; reconfigurable hardware.**

I. Introduction

In the last few years, the flexible radio concept has become one of the major research field in wireless networks. Rapidly evolving standards, increased mobility, and technology advances are calling for adaptable and flexible solutions.

Traditional radio devices, while very efficient for those computational intensive applications, can not cope with the growing number of connection methods offered by the standards. These different standards evolve very quickly. This imposes a flexibility, to keep pace with the advances included in the standards (update flexibility). Telephone applications are very representative of this two-dimensional expansion. Smartphones usually offer personal, local and wide area network connection (PAN, LAN, WAN), through different standards. All these standards are also evolving, which, with traditional devices, means redesigning a new platform to support the updates.

At the same time, the increasing requirements for quality in wireless networks demand that error control through retransmission (in addition to Forward Error Correction - FEC) be an unavoidable part of new standards. The aim of error control is to guarantee a controlled error probability during transmissions. Hybrid Automatic Repeat and reQuest (H-ARQ), further described in Section II, is now used in many wireless standards, such as 3GPP HSPA, IEEE WiMax or 3GPP LTE.

Implementations of H-ARQ in a traditional (not flexible) radio device has been described in several patents. Yet, the design of a flexible H-ARQ processor usable in a flexible radio context has, to the best of the authors' knowledge, not been described before. The aim of this article is to describe a solution to implement such a H-ARQ processor.

This paper is organized as follows. First, H-ARQ and traditional solutions, along with the challenges due to flexibility are presented in Section II. Then, the concepts underlying the proposed solutions are presented in Section III, while details on the actual solution are given in Section IV. Finally, use cases and practical results are presented in Section V, and the study concluded in Section VI.

II. H-ARQ details

The aim of this study is to describe a solution for flexible implementation of H-ARQ. This section presents usual H-ARQ schemes and existing implementations as well as challenges for a flexible implementation.

A. H-ARQ processing

H-ARQ is an error control protocol based on two types of error control methods:

- *proactive* methods, acting before transmission, basically using FEC, integrated in the PHY layer;
- *reactive* methods, taking place after errors occur, using Automatic Repeat reQuest (ARQ) protocols integrated in the MAC layer (or between MAC and PHY).

The use of these two types of techniques in a single protocol allows for enhanced error correction and reduced effect on performance. The original protocol, described in [1], was simply an ARQ protocol operating on FEC encoded packets. FEC was used as a solution to lower the retransmission probability, thus reducing the mean latency of a transmission, while ARQ was used to reduce the overhead of FEC methods, since it is not necessary to provide sufficient error correction for the worst case.

While the aim of the original H-ARQ protocol was to cope with the respective problems of FEC and ARQ, the cooperation of both methods is far more powerful. Several enhancements have been proposed, which make use of this cooperation. Incremental Redundancy [2] (IR) is a method used to further reduce the code overhead. In the first transmission, only the raw packet or a very high-rate code is sent. If errors occur, additional parity bits from the code are sent in the retransmissions, for error correction. This method is mainly used with punctured codes instead of the straightforward partitioning.

Chase Combining (or Code Combining, CC) [3] aims at a better efficiency of retransmissions, using confidence information on the previously received packet (from previous transmissions). In this scheme, no additional information is sent in retransmissions, but the receiver combines all the retransmissions, to obtain the most likely packet. This can be done at different levels. In this study, a slightly different combining scheme using soft bits average is used.

Finally, Partial Incremental Redundancy is a scheme using both approaches. Incremental Redundancy using punctured codes is the most complex but the most efficient version of the protocol, but the retransmissions are not self-decodable. In Partial IR, the raw packet is always included in the transmissions, with only the parity bits changing.

Enhancements are also possible on the ARQ side but not necessarily thanks to the cooperation. H-ARQ benefits from the classical ARQ protocols, which mainly impact the way retransmissions are scheduled. This study focuses on the most commonly used protocol, N-Stop and Wait (N-SAW). SAW protocols are "send and wait" protocols, in which the transmitter is in a waiting state while waiting for the acknowledgement from the receiver. N-SAW protocols are SAW protocols using multiple instances, meaning that several SAW protocols can run concurrently. This increases the required buffering capacity, as well as the packet ordering complexity, but the available bandwidth is better used.

B. *Integration and implementation*

H-ARQ is a cross-layer operation. It must be integrated in the transmission layer of a communication chain (Layer 1), to implement the combining methods, as well as in the protocol layer (Layer 2), to manage acknowledgement transmission and retransmission scheduling. This study focuses on the layer 1 integration.

Two main operations must be implemented in the PHY layer part of the protocol: the packet rebuilding based on the different transmissions, and the buffering of received packets. Integration of H-ARQ in a baseband processing chain is presented in Figure 1. The H-ARQ block processes soft-bits (estimated value, along with confidence information) from the deinterleaving block, before decoding. Data processed in the decoder is the enhanced results from the H-ARQ protocol.

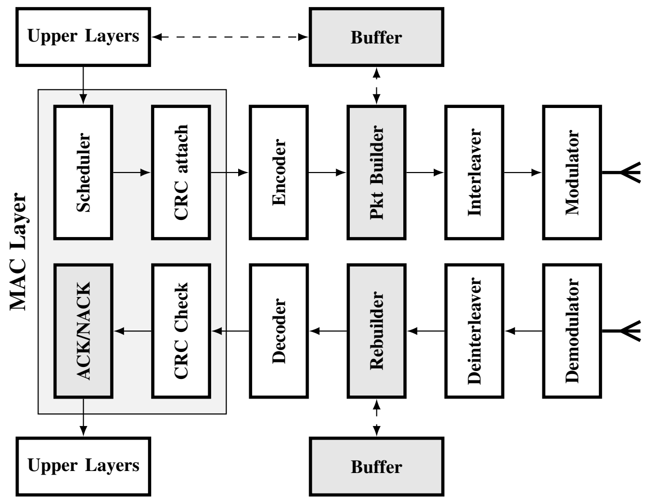

Figure 1. H-ARQ integration in a usual communication system

The rebuilding process uses several basic operations:

- a depuncturer to recover the punctured codes,
- a bit combiner for Code Combining, which can operate on retransmissions, on transmissions from multiple channels, or on repeated bits of a single transmission,
- retransmission concatenation for Incremental Redundancy, the most complex operation.

Possible enhancements to the protocol may require other operations to be implemented. These operations are simple. The problem when dealing with the implementation of H-ARQ comes from the scheduling of these operations. Yet, traditional device usually implements a single version of the protocol. This reduces the scheduling complexity, since it is predefined, allowing the H-ARQ to be implemented as a part of the decoder, using hard-wired control to manage the scheduling.

The real problem in these architectures lies in the buffer management. In a N-SAW protocol, the H-ARQ processor is required to process up to N packets concurrently. This means that it must be able to store the N received packets and select the buffer corresponding to the packet being received. Several, mostly patented, solutions have been proposed to solve this issue. [4] presents a method based on dynamic computation of addresses in a buffer to manage memory, along with a method to decide whether a received packet must be combined with an already received packet or not. [5] presents a global method to process H-ARQ in a traditional communication device.

Finally the emergence of Multiple Input Multiple Output (MIMO) solutions, as opposed to Single Input Single Output (SISO) solutions has led to research on the use of H-ARQ in such environments. MIMO devices use several antennas for emission and reception, leading to truly time-concurrent transmissions. Two main possibilities are used for operation with H-ARQ. The first one uses the multiple transmission possibility to send several retransmissions concurrently. The second one sends different packets on each channels, thus increasing the throughput when compared to SISO solutions. The theoretical study of the different methods is out of the scope of this study.

C. *Flexible radio constraints*

A lot of flexible radio platforms have been described in the literature. This paper does not focus on pure software radio, but on hybrid architectures, based on reconfigurable hardware and on software. The study focuses on a dynamically reconfigurable implementation of the H-ARQ (the hardware does not need to be deployed with each configuration change). Figure 2 presents a possible architecture of a flexible system, which will be used as a reference in this study. A generic processor is available, used to process high layers algorithms (especially layer 2). The processing chain is implemented as a sequence of processing units, controlled through a configuration controller, connected to the system bus, and addressable by the processor.

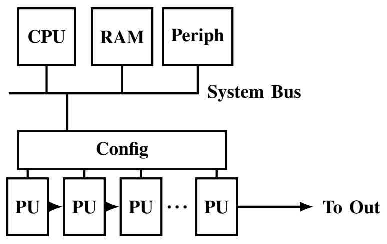

Figure 2. Reference system architecture

Flexible devices must meet constraints that can be avoided when dealing with more traditional, dedicated devices: in the development process, the design of the device is done before a standard is targeted, since the standard is dynamically chosen. While in traditional devices, it is possible to optimize the processing chain since the precise implementation details are known, flexible devices must take into account all the possibilities.

In the case of a flexible H-ARQ processor, two main constraints appear.

- Any version of the protocol can be used, as well as mixed version, and they can exist concurrently. The design must be able to process any protocol based on a sequence of known operations, regardless of the sequence, and to switch from one to the other efficiently, meaning with minimal reconfiguration overhead. It must also be able to stand MIMO operations;
- Buffers can quickly become a bottleneck. While in traditional devices, the maximum number of buffers (usually around 8) is known before the implementation, in a flexible environment, this upper bound can not be known. Selecting the highest possible number is inefficient, since it can evolve with standard updates. Furthermore, flexible radio is also used to implement different standards concurrently, in order to reduce the required silicon area in multiply connected devices.

Resolving these constraints is unavoidable in order to use H-ARQ in a flexible radio environment. The next sections present an efficient solution based on hardware/software partitioning and configurable hardware.

III. Design of the flexible H-ARQ processor

A. *Hardware/Software partitioning*

In order to meet the two flexibility constraints, the solution proposed as a result of this study follows the flexible radio philosophy:

- definition of an efficient hardware/software partitioning and
- design of an efficient reconfigurable hardware part to support the processing.

In a traditional implementation, H-ARQ partitioning between hardware and software usually follows the PHY/MAC partitioning. All the PHY elements are processed in hardware, offering pipelining capabilities and high processing power, and all the MAC protocols processed in software, offering the high level approach best fitted to MAC operations. The need for flexibility in the PHY elements (packet rebuilding) means that a new partitioning may be benefiting.

First of all, the management of the content of the transmissions is easier in software. While in a single standard, the sequence of retransmissions is usually based on a predefined sequence, which can be hard wired in the hardware part, this is not true in the flexible case, since each standard has its own sequence.

Secondly, buffer management also becomes a software part. In the presence of a single known standard, it is possible to manage the buffers through a simple round-robin algorithm. When several standards can be present at the same time, the algorithm is not so simple anymore, since the association between the standards and the buffers must be stored and controlled.

Thirdly, and to cope with the possibly high number of buffers, buffering itself is easier in software: any software system has memory. There are two main advantages to software buffering, apart from cost. Management of buffers is easier, since it is possible to use all the software mechanisms for buffers management. And secondly, locating the buffers in the software memory would make these buffers accessible to the processor, opening the door to more complex and possibly more efficient H-ARQ protocols.

B. *Operation sequence representation*

Given this modified partitioning when compared to the usual implementation, it is necessary to define the flexible hardware component as well as the control required to use it, and the integration of software buffers. The flexible H-ARQ processing presented here relies on a per-bit operation based on data representation.

In a practical communication protocol, different operations can be applied on the information. On the transmission part, the packet is encoded, and then prepared for the H-ARQ protocol before modulation for transmission. This preparation, made according to different parameters such as the current transmission index, the current estimated channel quality, or the H-ARQ protocol being used, is also called rate-matching. The rate-matching algorithm can:

- puncture, meaning delete information from the packet. The punctured information will not be transmitted.
- repeat, meaning send the same information any number of time in a single packet. This is often done to fill unused bandwidth.
- resend, meaning that information already sent in previous transmissions will be resend in this transmission.

Rate-matching can be a destructive operation, which needs to be reversed. To be able to use the received transformed packet, the receiver needs this packet, as well as information on the meaning of this packet: which operation has been applied to which part of the packet. This representation is sent along with the packet.

The representation can take different forms. It defines the association of each bits in the current transmission with the bits of the complete packet being transmitted. One of the most used representation is the puncturing pattern of the code. It is a binary matrix shared between the transmitter and the receiver, and in which a 1 means that the bit is present in the transmission, and a 0 means that it has been punctured. Another method is to use a shared parameterized algorithm, and to send only the parameters. Regardless of the method used, each information bit must be associated to the operation that was applied to it. When no repetition of bits in a single packet is possible, a single bit is sufficient, to signal whether a bit is present or not in the packet. If repetition is possible, several bits are required, to give the number of times the bit is sent. If a bit was punctured a neutral value must be inserted to replace the deletion. This neutral value usually is a "not trustworthy" value (meaning a soft-bit with the lowest possible confidence value).

This representation is the key to H-ARQ operation scheduling. It defines the sequence of operations to be executed by the H-ARQ component. This is better shown with the exemple of an Incremental Redundancy component, as in Figure 3. The control sequence is the concatenation of the matrices representing the current (new) and previous (prev.) transmissions. A neutral value is also possible if none of the transmissions contain a bit. If the scheme is Full Incremental Redundancy (no combination of bits), when the matrix for the new transmission gives a "1", data is consumed from the new transmission data source, else when the matrix for the previous transmissions gives a "1", data is consumed from the buffered data source, else when both matrices give a "0", data is consumed from the neutral data source.

This concatenation method can be extended as a complete scheduling method: when both matrices give a "1", data is read

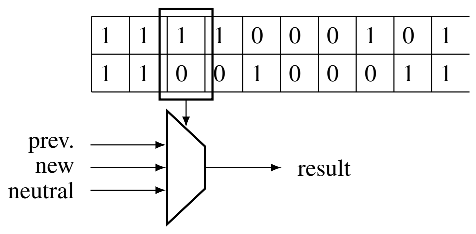

Figure 3. Dataflow operation

from both buffered and new data source, combined, and output. The combination of the representations of all transmissions of a packet creates a codeword, defining which operation must be applied to recover a given information bit.

Using this representation of packets, implementation of H-ARQ does not need prior knowledge of the version used. All inputs become data flows, as well as outputs. The representation is consumed by the component, which enables consumption of some input data, and production of some output data. For each possible representation value, a given input/output mapping is used.

The H-ARQ processor resulting from this hardware/software partitioning and from the per-bit operation principle presented is divided in three main parts, as shown on Figure 4, further presented in the next sections:

- a control part, which computes the operation associated to the current bit,
- an operating part processing data according to the operation associated to the current bit,
- a buffer part to store the transmissions associated to each instance of the protocol.

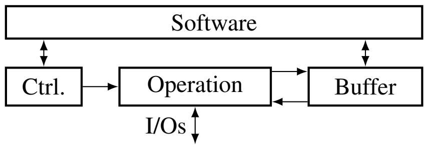

Figure 4. Dataflow scheduling architecture for H-ARQ

IV. Implementation details

A. Operating part

The operating part described in the previous section is responsible for the bit processing according to the control part instructions. Since the atomic operations used by the H-ARQ component are simple, each operation is implemented separately and a multiplexer is used for control. A simple mux is not sufficient, because of the possible number of inputs and outputs (especially in MIMO operation). A specific configurable data switch has been designed to implement the operation mux. It associates a specific input/output mapping to a given codeword (transmitted by the control part). The mappings may be dynamically modified if some standards require different operations. The modification is done by writing values to a look-up table, with a low latency. This latency is too high for modification during processing of a packet, but is negligible for reconfiguration between two packets.

The resulting architecture is presented in Figure 5. The operations are implemented between two data switches. The input switch is sufficient to schedule any sequence of operations. It selects its inputs (buffered data, a given input stream if several are possible, or a neutral value), and maps them to the output corresponding to the desired operations. The output switch is used to manage the buffers, and the output streams if several are possible (for example, a Viterbi decoder or a block code decoder). As a result, all dataflow management, and thus all the configuration process, is performed using the switches.

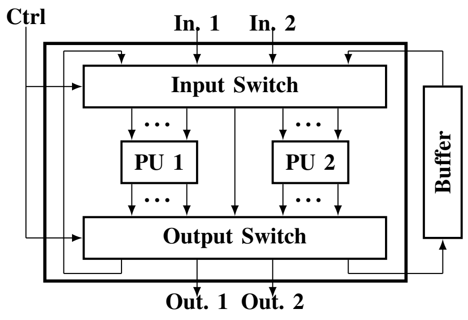

Figure 5. Dataflow scheduling architecture for H-ARQ

The configuration process only consists in giving the I/O mapping information to the switch, *ie.* associating a codeword (runtime control associated to each byte) to a I/O mapping.

B. Control

The second part in the component design is the control part. Control of H-ARQ processing, in this component, means sending to the operating part the sequence of operations needed to build the packet from all its transmissions. Different designs have been envisioned for this part.

Two main operations must be processed when controlling H-ARQ processing. First, recover the transmission representation, meaning the information required to recover the received packet based on the transmissions. This operation can be either very straightforward (known puncturing patterns, which are applied to the transmission), or they may require some computation (parameterized algorithm) in order to recover the complete transmission.

The second operation is the effective control of the operating part, meaning the processing of the representations in order to create the commands to be sent to the operating part. Once again, this can be very straightforward, like in the example in Section III-B, or it can require some computation when the rate-matching process becomes more complicated. For example, if repetition of information in a single transmission is allowed, the concatenation of representations may not be the best choice, since it would be necessary to add support for N-bits repetition in the operating part. It is better to have only a 1 bit repetition operation, and to let the control iterate the operation N times (iteration is typically control).

In order to support both control operations, different hardware/software limit have been experimented.

- An "all hardware" solution, in which everything is processed in hardware. This is an efficient solution for simple H-ARQ schemes, since it frees the processor, and since it avoids the need for synchronization between both targets. Yet, it substantially reduces the flexibility, since updates in processing

becomes difficult, and it limits the component capabilities to the implemented algorithms.
- A mixed solution, in which one operation is implemented in software, and the other in hardware. This is inefficient, since to avoid excessive data exchange between both domains, this solution makes sense only if the hardware operation is the second one, which benefits more from a software implementation.
- The most flexible solution is the "all software" solution. In this solution, everything is processed in software, and the only hardware element is an interface with all the operations to be processed by the operating part. This interface is a small memory, and the size of the sequence being stored. If the sequence is smaller than the packet being processed, the control part iterates the sequence (no control, only a counter), otherwise, the packet is fully represented in memory.

These experimentations are not shown in Section V, but the preferred solution is the software solution in terms of flexibility. The required processing is not time consuming for the processor, while it is complex in hardware. The all hardware solution is useful for a fast implementation of a non-flexible component.

C. *Bufferless design*

The previous design meets the first flexibility constraint, the protocol-independence. But the second constraint is still not met. The last part of the H-ARQ component is the buffer part. The goal of the buffer part is to manage the storage, as long as required, of received transmissions, which may still be used. This management means that, along with the required buffer space, a control of which information is stored, and where, is also required.

The buffer part, in an environment where the number of possible concurrent transmissions is known, can be implemented in hardware. The existing implementations of H-ARQ mostly uses small and fast memory, with the control responsible for managing the memory, and associating an address with a transmission number. In this study, experimentations have been conducted with FIFOs affected to a specific transmission through the use of a mux. The resulting simplification of multiple buffers management comes at the expense of increased single buffer size.

But with FIFOs or with addressable memory, the infinite buffer constraints is still not met. In order to offer to the device as much memory as required to store numerous transmissions, the hardware buffer has been replaced with a software buffer, located in the processor (software) memory, and managed by the software. This offers multiple advantages:

- more available memory for the component.
- easier to manage. Memory management is an important task in software, and all kernels offer a memory management API to the applications
- accessible buffer. More efficient H-ARQ algorithms could be used, through the use of software, since the buffer is made visible to the software.

This extended flexibility has a cost: data transfer to and from software must not be a blocking point. Figure 6 shows the buffer part details. FIFOs are inserted at the input and at the output to avoid stalling the transfer, and thus be able to use the burst capacity of most RAM. These FIFOs are used as a local cache for

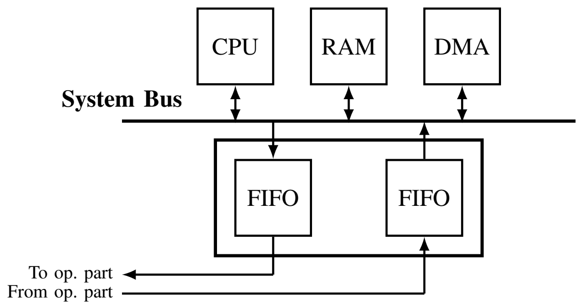

Figure 6. HW/SW interface for bufferless H-ARQ

buffer data. In order to avoid overloading the processor, a DMA is required to implement the software buffers. The size of the FIFOs depends on the bus throughput.

V. Practical results

A. *Environment*

Validation of the proposed architecture means validation of the flexibility as well as validation in terms of performance (impact on the global system, and reconfiguration overhead). In order to experiment on both aspects, the proposed architecture has been implemented and integrated in a complete communication chain, and compared with non-flexible devices implementations of Chase Combining and Incremental Redundancy.

The targeted communication environment is the Magnet high data-rate communication chain [6], supporting up to 40 Mbps. The hardware platform is based on an ATMEL AT91 (a System on Chip integrating an ARM processor), and a Xilinx Virtex 4 (XC4VSX55). The maximum packet size allowed has been fixed to 2kb per packet. The convolutive code used for FEC is a rate $\frac{1}{3}$ code, leading to up to 6kb per encoded packet. The reception chain uses soft bits to add confidence information to the received packet. In the considered implementation, a soft bit is coded on 3 bits words, meaning 4 levels of confidence for each possible hard bit value. All these choices are coherent with existing standards, even if the size is rather smaller than usual.

Based on this implementation, two main lines have been followed when conducting the experiments. First, the flexibility of the platform is studied. While it is difficult to prove that any H-ARQ variation can be implemented, extensive experimentations are conducted in order to validate at least some flexibility. The second point studied is the cost of flexibility. Both points are studied in the next sections.

B. *Flexibility: configuration examples*

The implementation of the proposed architecture uses two possible processing units:

- a combiner, which combines the information coming from two inputs selected according to the codeword. This combiner outputs the resulting combined symbols.
- a voter, which outputs the most probable value among the two selected inputs.

The voter has been implemented in order to validate the feasibility of multiple dataflow operations management in the device. When combining was required in a version of the protocol, both operations could be used. Based on these two units, several configurations have been produced in order to process the different versions.

When operating in Chase Combining mode, the output stream is the result of the combination of previous and current transmissions, which adds the need for a combining operation, applied on previous and current transmissions. Incremental Redundancy operation adds complexity, since data can be present in current or in previous transmissions only. Combining of multiply received bits is optional in Incremental Redundancy, but it comes at no cost using the proposed component.

The most challenging operation in this architecture is the repetition of bits. Two possibilities are used. In the first one, a loop is introduced after the first switch, and after the last one, thus adding two possible inputs for the component. Repetition of a bit is then just a matter of selecting the corresponding input. In this case, a delay is introduced during the processing if a bit is repeated more than once, since the two first repetition must be combined before combining with the third one. The second method uses a special processing unit, able to accumulate the inputs, as long as the output is not read.

In order to validate the flexibility of the architecture, experimentations have been conducted on the physical platform. The purpose of these experimentations is to evaluate the ability of this new component to deal with numerous, concurrent and independent instantiations of the H-ARQ protocol. For a given instantiation, either a known version of the protocol is used, from existing standard, or versions built using randomly generated puncturing/repeating patterns may be chosen. While this is not sufficient to guaranty that the proposed component is usable for any possible H-ARQ scheme, it offers a good coverage of the possible cases.

This scenario has been used for big files transfers (1GB each). The switch time from one instance to the others corresponds to the time of two memory write. The component is able to stand high data rates (up to 40Mb/s in simulation, the platform is not able to stand more than 20Mb/s), including reconfiguration latency between each packet. These results make this component fit for real-time applications.

C. Performance and configuration overhead

The precedent section showed that the flexibility constraint is met by the architecture. But it is important to study the cost of the architecture, since flexibility usually means overhead.

In terms of slices cost in the FPGA, the addition of the H-ARQ component and its control interface adds around 300 slices, which means about 1% additional slices in the reference reception chain. As a comparison, this cost is similar to the cost of an Incremental Redundancy only component. In terms of buffer space, the proposed H-ARQ component requires 64 bits of local buffer. With a total available memory of 32MB, and a usable DMA engine, the platform has been used to implement a 10-SAW adaptive H-ARQ protocol. This implementation requires 18kb of free memory for each protocol instance, but this memory is located in the system memory, and no buffers are required in the hardware device, leading to an important gain in silicon area requirements.

In terms of performance, the component processes 1 input bit per clock cycle with no repetition. When there is repetition in the scheme, with the accumulator solution, the throughput stays the same, but control becomes more complicated because of the need for synchronization at the output. If the loopback solution is used and if a bit can be repeated only once, there is no overhead either. If more repetition can occur, a delay is introduced in the processing. This delay is equal to the length of the pipeline, *ie.* 6 clock cycles, once every two repetitions. The configuration time, which is the time required to change the codeword/mapping association, is negligible, since it is only a write to a look-up table.

VI. Conclusion and Future Work

An implementation of a H-ARQ processor fit for flexible radio devices has been presented in this paper. Thanks to its design, this component is able to cope with current and future envisioned versions of the protocol. The buffer is virtually unlimited, and the control management is flexible. The choice of operations, implemented between the switches in the operative part, is open. Managing multiple inputs or outputs, when parallel transmissions are possible (OFDMA, MIMO, ...) is intuitive thanks to the dataflow view.

The component is based on a specific software/hardware partitioning:

- memory and scheduling of required operations are managed through software,
- actual computation is done in hardware, using configurable codewords to describe the operation to be processed.

The codewords used to describe the different operations are based on the representation of the different dataflow. The component is able to use the sequence of operations applied on the original packet and on the already received transmissions, and associate to each combination an operation to process.

The resulting component is fast, small (about the same size as other dedicated H-ARQ devices used for a single version of the protocol), and flexible enough that any H-ARQ protocol can be configured and processed. In the continuation of this work, power consumption measures are planned, as well as a study of new protocols using the flexibility property to enhance the error correction capacity.

References

[1] J. Wozencraft and M. Horstein, "Coding for Two-Way Channels," Research Laboratory of Electronics, MIT, Tech. Rep., 1961.

[2] D. M. Mandelbaum, "An Adpative-Feedback Coding Scheme Using Incremental Redundancy," *IEEE Transactions on Information Theory (corresp.)*, vol. 33, pp. 388–389, 1974.

[3] D. Chase, "Code Combining - A Maximum-Likelihood Decoding Approcach for Combining an Arbitrary Number of Noisy Packets," *IEEE Transactions on Communications*, vol. Vol. COM-33, pp. 385–393, 1985.

[4] B. Franovici, "Method and system for memory management in a HARQ communication system," European Patent 2 093 921 A1, 2009.

[5] M.-C. Tsai, W. Chowdiah, and W. Jing, "Packet decoding for H-ARQ transmission," World Patent 2009/108 516 A2, 2009.

[6] R. Prasad, *My personal Adaptive Global NET (MAGNET)*. Springer, 2010.