Managing dynamic reconfiguration on MIMO Decoder

Hongzhi Wang, Jean-Philippe Delahaye, Pierre Leray and Jacques Palicot

IETR/Supelec
Campus de Rennes
Av. de la Boulais, CS 47601
35576 CESSON-SEVIGNE, France
{hongzhi.wang, Jean-Philippe.Delahaye, pierre.leray, jacques.palicot}@supelec.fr

Abstract

This paper is about the implementation of a MIMO V-BLAST (Vertical Bell Laboratories Layered Space-Time) square root decoder in a FPGA using dynamic partial reconfiguration. The decoder architecture is based on four CORDIC (COordinate Rotation DIgital Computer) Units. Among these CORDIC units, three are used in rotation mode and the fourth one is used in vectoring mode. The design implementation aims power saving and area efficiency allowing dynamically changing the interconnections between the fixed modules in the reconfigurable modules. This MIMO square root design method shows the configuration time improvement, area efficiency and flexibility of the decoder by using the dynamic partial reconfiguration method.

1 Introduction

Dynamically reconfigurable FPGAs offer new design space with a variety of benefits: flexibility and reusability at run time. The dynamic reconfiguration is closely related to partial reconfigurability of FPGA. Indeed, the partial reconfigurability allows to selectively change segments of the FPGA functionality without suspending operations of the remaining parts. There are several benefits of partial reconfiguration. It reduces the configuration time and saves memory as the partial reconfiguration files (bitstreams) are smaller than full ones.

Reconfigurable computing [3] and [6] has been proposed in a large range of signal processing applications in order to improve high performance, flexibility and adaptability. The development of wireless communication systems has indicated the need to dynamically adapt systems architectures at the hardware level, as in Software Radio system [8].

One of the most promising technologies to enhance the wireless communications performances is Multiple-Input Multiple-Output (MIMO). MIMO is an attractive technology for future wireless systems because of their huge bandwidth capacity. It is well known that an extraordinary spectral efficiency can be achieved in MIMO system [5]. In various MIMO detection algorithms, square root decoder is an interesting tradeoff to obtain a high performance with reasonable complexity.

In our previous work [9], we have implemented a reconfigurable architecture MIMO decoder with various number of CORDIC. It adapts to different number of antennas, different signal constellations and different throughputs for wireless communications.

We introduce in this paper the dynamic reconfiguration into the MIMO V-BLAST decoder architecture. We especially detail our design experiments to integrate the control of configuration into the processing algorithm during the MIMO decoding. Dynamic partial reconfiguration is used to change the interconnection between the processing modules.

The rest of the paper is organized as follows. The square root algorithm and block diagram are briefly described in section 2, details further in [9]. The reconfigurable architecture for square root decoder is detailed in section 3. Section 4 deals with the configuration management. Section 5 presents the design methodology. The experimental results are provided in section 6. The conclusions and a look at future research will be stated in section 7.

2 Decoding algorithm

The V-BLAST square root algorithm is proposed in [4], which successfully avoids the repeated pseudo inverse and matrix inverse computations by using unitary trans-

1-4244-0910-1/07/$20.00 ©2007 IEEE

formations. The computational cost is reduced effectively from $O(M^4)$ to $O(M^3)$ without degradation in BER performance, where M is the number of transmit antennas. The whole algorithm is described in the following steps:

A) Compute $P^{1/2}$ and Q_a : for i= 1, 2,. . . ,N:

$$\begin{bmatrix} 1 & (H)_i P_{i-1}^{M\times M} \\ 0^{M\times 1} & P_{i-1}^{M\times M} \\ -e_i^{N\times 1} & Q_{i-1}^{N\times M} \end{bmatrix} \Theta_i = \begin{bmatrix} \times & 0^{1\times M} \\ \times & P_i{}^{M\times M} \\ \times & Q_i^{N\times M} \end{bmatrix} \quad (1)$$

In this relation, $P_0^{1/2} = \beta I$, $Q_0 = 0^{N\times M}$, e_i is the i-th unit vector of dimension N, Θ_i is any unitary transformation that block lower triangularizes the pre-array and $\times$ are the results ignored. After N steps, we obtain: $P^{1/2} = P_N^{1/2}$ and $Q_\alpha = Q_N$.

B) Determine the optimal ordering and nulling vectors: for i=M, M-1,. . . ,1:

B_1)Find the minimum length row of $P^{1/2}$ and permute it to be the last (Mth) row. Permute s accordingly.

B_2) Find a unitary Σ to block upper triangularize $P^{1/2}$:

$$P_i{}^{1/2}\Sigma_i = \begin{bmatrix} P_{i-1}{}^{1/2} & \times^{i-1\times 1} \\ 0 & p_i \end{bmatrix} \quad (2)$$

B_3)Update Q_a to $Q_a\Sigma_i$, the nulling vector for the i-th signal is given by

$$w_i = p_i q_{\alpha,i}^* \quad (3)$$

where $q_{\alpha,i}^*$ is the i-th column of $Q_a{}^*$.

B_4)Compute $y_i = w_i r$, and then the i-th transmitted signal in s is detected as the closest point in the signal constellation.

B_5) Cancel the interferences of the detected signal in the remaining received signal s:

$$r = r - s_i(H)_i \quad (4)$$

B_6) Go back to the step B_1, but now with $P_{i-1}^{1/2}$ and $Q_{\alpha,i-1}$ (the first i-1 columns of Q_a).

The Block diagram of the MIMO square root decoder is illustrated in figure 2. It consists of 6 processing modules. The values of matrix channel H and messages r are assumed to have been pre-calculated. The three first modules(M_1,M_2,M_3) use unitary transformations to compute $P^{1/2}$(Step A), Q_a(Step A), p_i(Step B_2) and $q_{a,i}^*$(Step B_3) by employing various numbers of CORDIC. The following module(M_4) calculates the optimal ordering and nulling vectors w_i. Module M_5 compute the transmitted symbol vector. The last module (M_6) performs interferences cancellation.

The three unitary transformation modules have a similar architecture. In these modules, unitary transformations

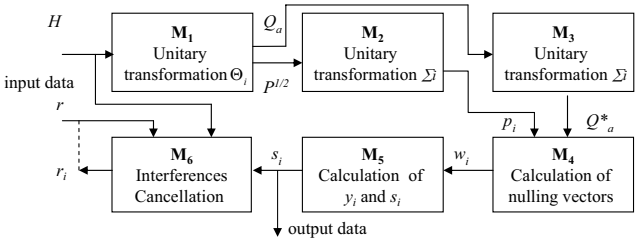

Figure 1. Block diagram of square root decoder

are used instead of the conventional QR triangular array which employs a too high number of processors [6]. Unitary transformations are performed by a sequence of numerically stable complex Givens rotations which are suitable for implementation because the hardware elementary is based on CORDIC in which only shifters and adders are involved [1]. It reduces the computational complexity significantly.

In the module M_1, the elements of equation (1) are passed by column to operator CORDIC which performs Givens rotations. Then the products are stored in the buffer waiting to be passed to operator CORDIC again. This completes an iteration. After N iterations, the module output becomes $P^{1/2}$ and Q_a. The modules M_2, M_3 perform the same calculation but with the elements of equation (2) and (3). Because there is no room here to explain the internal details of CORDIC, the reader can see the reference [1]. The last three modules (M_4, M_5, M_6) are based on PE (Processor Elementary). Every PE unit consists of a multiplier-accumulation unit, a adder-substractor and a buffer. The module can improve the throughput by paralleling several modules.

3 Dynamically reconfigurable architecture

3.1 Dynamic Reconfiguration

The dynamic reconfiguration is used to share a group of configurable elements between several processing contexts. This is a resource multiplexing. We can distinguish two types of dynamic reconfiguration which are illustrated in figure 2.

The figure 2.1 shows the case of the dynamic reconfiguration with data dependency between both functions T_2 , T_3 that share the same configurable resources. In this case, it is necessary to ensure the data temporary storage. The figure 2.2 shows the use of dynamic reconfiguration without data dependency between two functions. The second case could occur in applications when the transmission environment changes [7], (for example, signal-to-noise ratio,

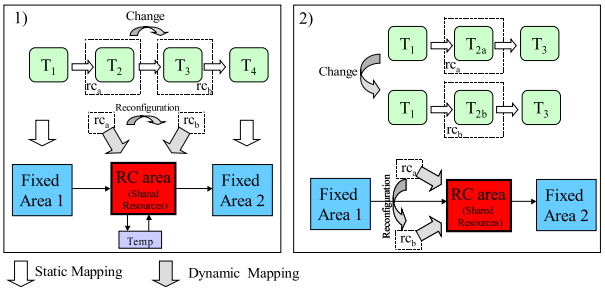

Figure 2. Different types of dynamic reconfiguration

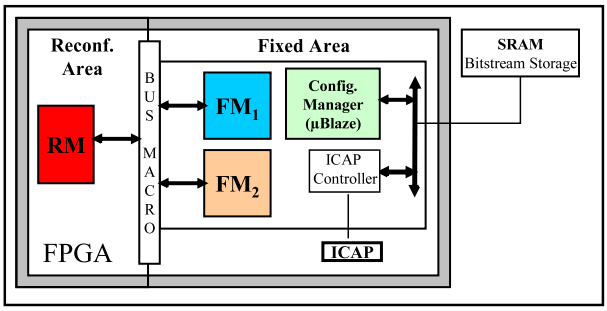

Figure 4. Dynamic reconfigurable architecture

available power, etc.), or when a more efficient function or a less power consuming one is required.

The tasks scheduling of both type of dynamic reconfiguration are illustrated in figure 3. These simple examples show that in the first case, figure 3.1, the reconfiguration task should be controlled by the processing to avoid data loss. In the first case, T_2 should wait for the end of processing of T_1 and the reconfiguration. Whereas in the second case, the dynamic reconfiguration could be performed at any time except when the reconfigurable block is under data processing.

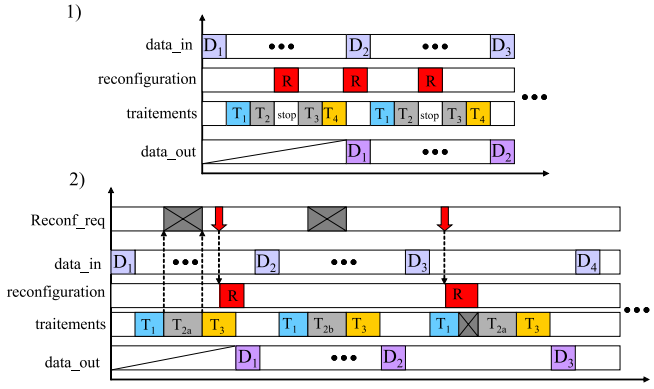

Figure 3. Temporal representation of dynamic reconfiguration

The first case occurs during the processing when processing blocks of a large function are multiplexed. This could be useful to reduce the area of the function. We focus in this paper on this first type of dynamic reconfiguration. Particularly, we tackle this issue to reconfigure point-to-point interconnections, to change the datapath between processing elements of a MIMO decoder.

In the dynamic reconfigurable architecture, shown in figure 4, the configuration manager is an embedded processor (like Xilinx MicroBlaze) and the configuration interface is the Internal Configuration Access Port (ICAP) of the Xilinx Virtex-II FPGA. The logic resources are divided into two parts: the fixed area and the dynamically reconfigurable area. Each of these two areas can contain several modules. For example, there are two modules FM_1 and FM_2 in the fixed area and one module RM in the reconfigurable area. The processing elements T_1 and T_4 will be performed in two fixed modules FM_1 and FM_2. The context of reconfigurable module RM can be changed between T_2 and T_3.

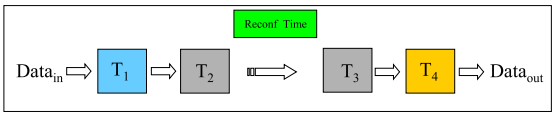

Figure 5. Datapath of first type of dynamic reconfiguration

The datapath of first type of dynamic reconfiguration is shown in figure 5. There is data dependency between two configurations T_2 and T_3. The processing time depends on the reconfiguration time of two context (T_2 and T_3) of reconfigurable module RM. On the contrary, the processing is not interrupted by the reconfiguration for every data in the second type (figure 2.2). In this paper, the first type of dynamic reconfiguration is used in the MIMO decoder and the reconfiguration is executed as one part of the decoding process. The partial reconfiguration allows to reduce the reconfiguration time and saves the storage memory as the size of partial reconfiguration files are smaller than the full ones.

3.2 Square Root Decoder Overview

In our previous work, we have implemented an reconfigurable architecture MIMO decoder with various numbers of CORDIC [9]. It adapts to a different number of antennas and different throughputs for wireless communication.

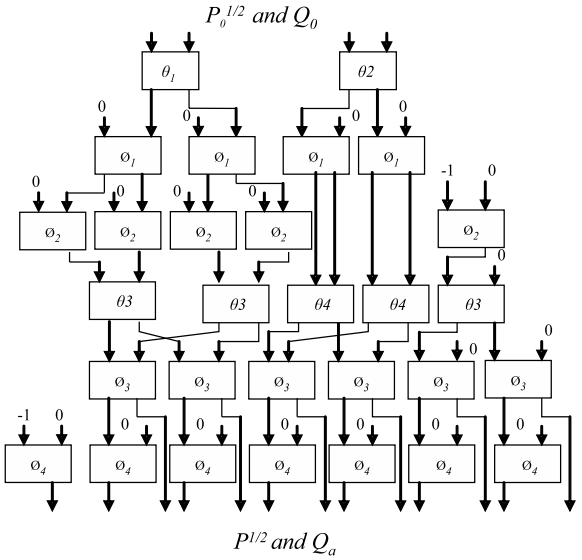

Figure 6. Calculation of $P_1^{1/2}$ and Q_a in the module M_1 in total parallel structure

The calculation of $P_1^{1/2}$ and Q_a in module M_1 is shown in figure 6. This example illustrates the design of the computatitional architecture of M_1. This calculation requires 29 CORDIC in rotation mode and its sequence is shown in figure 6. They use different angles ($\theta 1$, $\theta 2$, $\phi 1$, $\phi 2$, $\theta 3$, $\theta 4$, $\phi 3$, $\phi 4$) that are pre-calculated by a CORDIC in vectoring mode (not shown in figure 6). This total parallel structure may lead to a waste of computational capabilities, since the channel data changes slower than the received symbol data. Therefore the iterative use of several CORDIC operators can optimize the resources. So we have implemented a decoder with a iterative structure that uses three parallel CORDIC operators to replace 29 CORDIC operators in the total parallel structure.

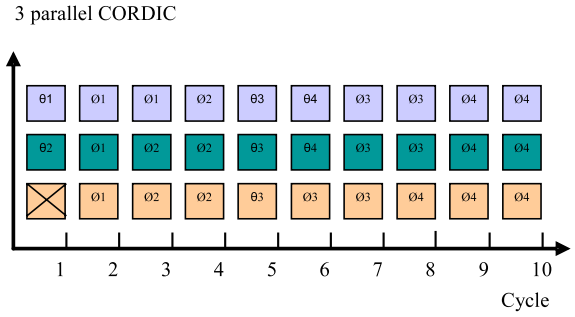

Figure 7. Temporal representaion of calculation of $P_1^{1/2}$ and Q_a in the module M_1

In the iterative architecture of module M_1, we have used only 3 CORDIC operators in parallel (see figures 7 and 8). In the first cycle, we use 2 CORDIC operators to perform 2 Givens rotations with angles $\theta 1$ and $\theta 2$. Then in the next cycle, 3 rotations with the different angles (shown in figure 8) are performed by 3 CORDIC operators. The operators are the same as in the first cycle and are re-used in the second cycle, and so on for the following cycles. After each iteration, the producted datas are used in a different way. So the interconnections between operations should change every cycle. The whole of the processing is performed by 3 CORDIC operators in 10 cycles.

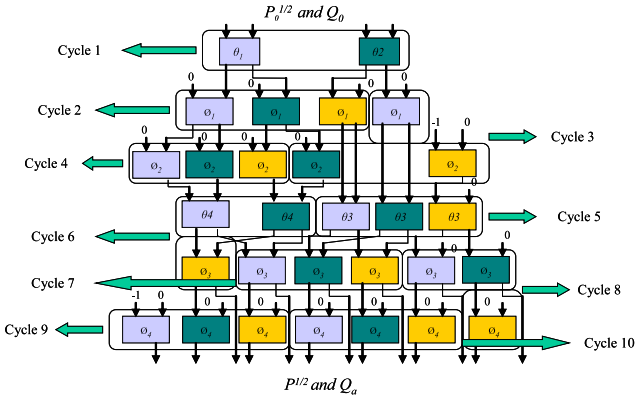

Figure 8. Calculation of $P_1^{1/2}$ and Q_a in the module M_1 in iterative structure

All iterations of CORDIC algorithm are performed in parallel, using a 20 steps pipelined structure. The input data of the CORDIC periodically changes and static implementation of the interconnections frameworks uses a great number of multiplexers to switch from one interconnection context to the next one. They take a lot of surface of FPGA and lead to waste of power consumption. Nevertheless, these multiplexers remain in the same state during 20 steps of CORDIC operations. The only difference between every 20 steps are the interconnections. This fact lets inspire the implementation on reconfigurable hardware, as shown in figure 9.

Our approach splits the processing into a static hardware skeleton which is composed of decoding processing elements and a reconfigurable part that evolves at run-time depending on the step of processing to perform. In this calculation the processing elements, three CORDIC operators, are implemented in the fixed part and the interconnections between processing elements are implemented in the reconfigurable module (shown in figure 9). Thus the multiplexers are changed by certain number of reconfigurations which are determined by decoding processing. Every reconfiguration represents one state of multiplexer. It is suitable that

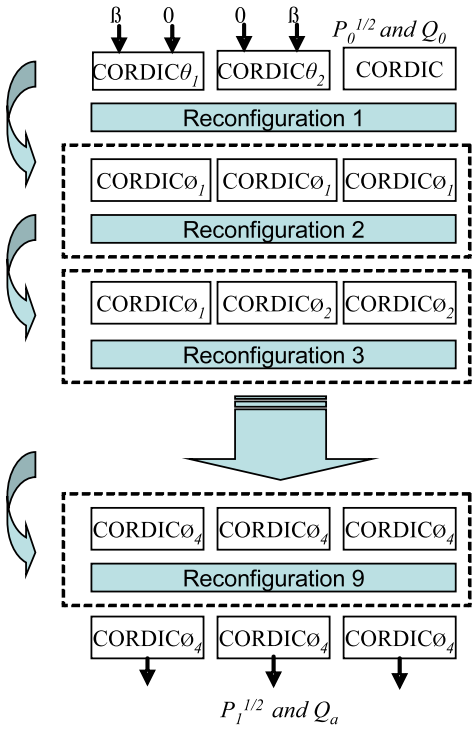

Figure 9. Calculation of $P_1^{1/2}$ and Q_a with reconfiguration

the reconfigurations are used in the decoder, because the main processing is preformed in the CORDIC operator and only the interconnnetions between them are changed at certain regulary moment. Moreover, it is possible to reduce power consumption because in the reconfigurable module only wire resources are used.

3.3 Interconnection Multiplexing

Concerning reconfigurable multiplexer hardware, some researches have been focused on reconfiguration of the I-Mux (Input Multiplexers) or the LUT (look-up table). For example, a reconfigurable crossbar switch by reconfiguring I-Mux is presented in [11]. The switch is implemented on an FPGA using partial configuration to modify routing resources during operation. The reconfigurable 3×1 LUT, shown in figure 10.2, in each logic cell of the FPGA is used to perform a multiplexer in [4].

Both of these approaches use RTL description level. We use here a system level description to make the reconfigurable interconnection switch, showing in figure 10.3. The interconnections of processing elements are defined as a module which is implemented in the reconfigurable part. The input and output of interconnection module is connected to the output and input of fixed part processing elements. The granularity of reconfiguration is taken up from the LUT and I-Mux level to module level.

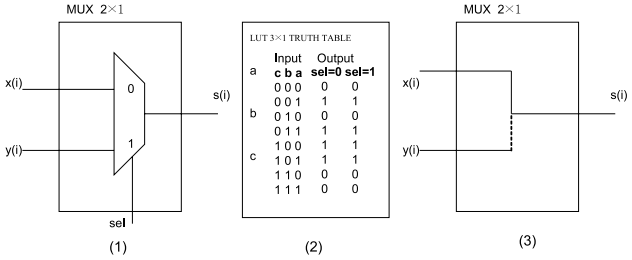

Figure 10. Different multiplexer. static version (1), LUT-based dynamic version (2) versus module level dynamic version (3)

3.4 Dynamically Reconfigurable Decoder

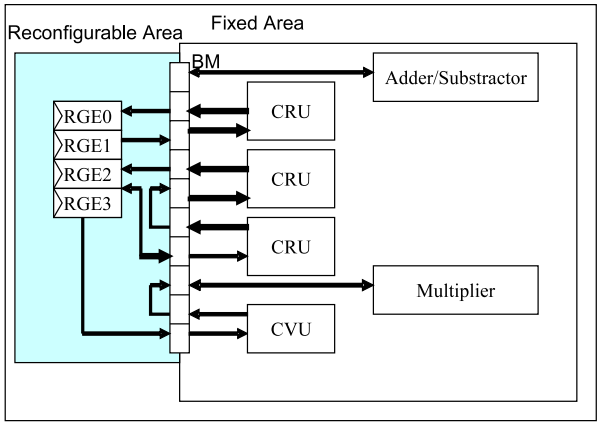

Figure 11. Placement of fixed modules and reconfigurable module (interconnections and registers)

Our architecture contains two parts (see figure 11). The first one is a fixed part that contains four CORDIC units (3 CRU: CORDIC Rotation Unit and 1 CVU: CORDIC Vectoring Unit), a multiplier and a adder or substractor. In the CORDIC unit, our design makes use of an iterative bit-parallel CORDIC architecture. The second one is the reconfigurable one, allowing the implementation of interconnections between fixed part modules. In the reconfigurable part, only routing and registers are implemented. The same register is placed and routed in the same place of FPGA in order to retain the information stored in the register after a reconfiguration. It makes possible to share internal data between two configurations. The reconfigurable part is

connected to the modules of fixed part through LUT-based Bus Macro provided by Xilinx to ensure the right place and routing crossing over partial reconfigurable area.

4 Configuration Management

The first stage of the FPGA configurations is its initialization. The full bitstreams and the partial bitstreams are stored in the Host memory (see in figure 12). The initial configuration, a full bitstream, is downloaded by the Host controller. This FPGA configuration contains the MicroBlaze and the MIMO decoder (fixed part and initial context of the reconfigurable part), which is shown in figure 12. Next, the reconfigurations are only partial and are performed during the processing to change the interconnections between the fixed processing elements. Even if a reset occurs, we call it a *soft_reset* and it implies that a partial bitstream composed of the basic CORDIC units, multiplier is automatically loaded by reconfiguration controller into the FPGA.

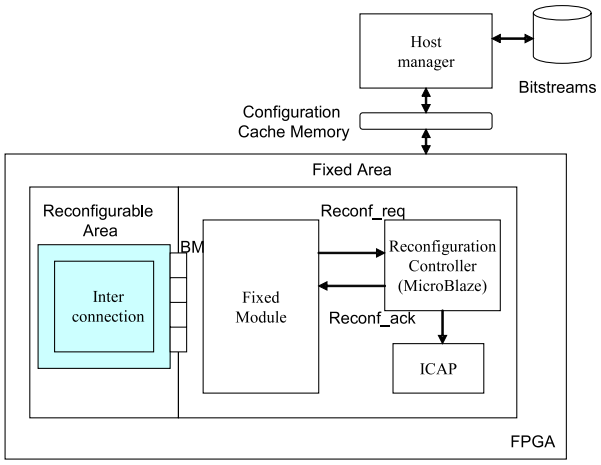

Figure 12. System architecture

Reconfiguration during decoding processing: During the partial reconfiguration process all the modules can continue working except for the reconfigured module. Thus in our design, the reconfiguration tasks are considered as one part of the decoding process. The reconfiguration sequence is predefined by the decoding proceeding (as shown in figure 13).

The reconfiguration tasks are inserted in the decoding processing when a change of interconnection is required. Each reconfiguration, during decoding, is requested by the fixed part of the decoding. The fixed part sends the request for reconfiguration by the signal *reconf_req* (figure 12) to configuration controller. After the negotiation with the Host, the MicroBlaze retrieves the partial bitstream from

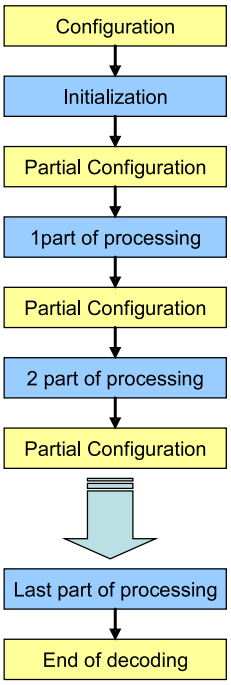

Figure 13. Partial configurations Sequence

the Host memory. Then the MicroBlaze downloads the bitstream through the ICAP interface [2]. ICAP is a primitive which allows the Virtex-II FPGAs to access to its configuration memory.

The fixed module should wait for the reconfiguration acknowlege signal *reconf_ack* sent from the configuration Controller. This signal confirms the end of configuration and enables the data availability for the next step of decoding proceeding. The program run by the reconfiguration controller for the run-time reconfiguration management is shown in figure 14.

```
Send  full bitstream to FPGA
Begin decoding processing
If  processing requires a reconfiguration:
    send a signal to Host ask for a partial reconfiguration bitstream
    if end of partial reconfiguration
      continue the processing
    else
      wait for end of partial reconfiguration
else
    continue the processing.
```

Figure 14. Partial reconfiguration bitstream manager structure

5 Design Methodology

The partial bitstreams for the IP modules are generated following the methodology developed by Xilinx. The design flow is based on Early Access Partial Reconfiguration (EAPR) [10] with the use of the new design tool, PlanAhead. It provides a hierarchical floorplanning, block-based, modular, and incremental design methodology. It allows changing only part of the design and leaving placement of the remaining intact. The PlanAhead floorplanner allows to handle LUT-based Bus Macro and placed during the floorplanning phase. All of these shorten design iterations, even while making frequent changes.

PlanAhead does not require the user to perform all the operations in ISE. The designer only needs to synthesize top level and module in ISE. All other operations (floorplanning, P&R, bitstreams generation) are directly performed in PlanAhead which launches automatically ISE tools.

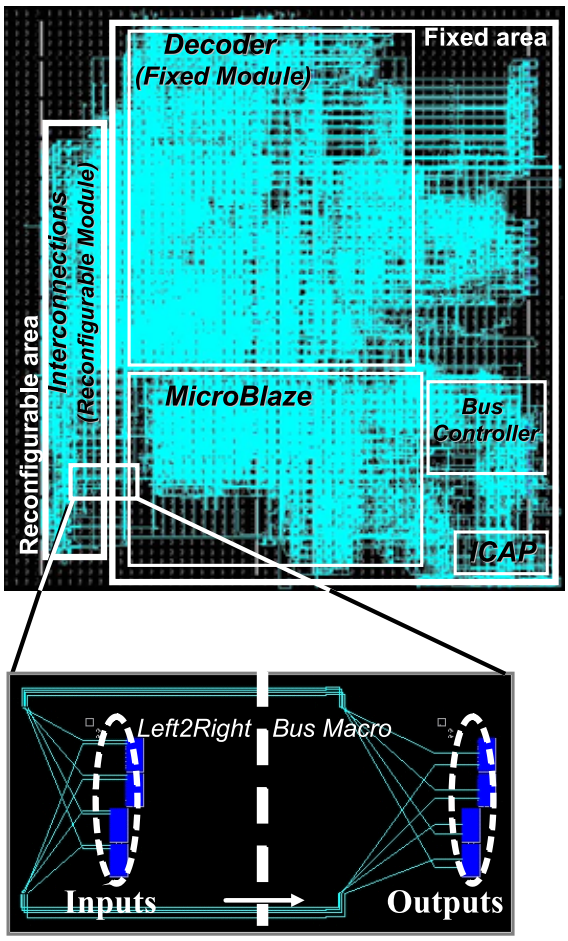

Figure 15. PAR map of square root decoder

Table 1. Synthesis results of MIMO square root decoder

Target FPGA Xilinx Virtex	slices	Flip Flops	Reconf time
Without Reconf	4505 (40%)	5927 (27%)	16ms
Fixed part	2857 (26%)	4766 (22%)	12ms
Reconf part 1	85	148	0.40ms
Reconf part 2	85	148	0.40ms

6 Experimental results

The MIMO decoder for 2 antennas system with QPSK signal constellation is designed in VHDL, simulated with Modelsim. The decoder is implemented and tested on a Virtex-II xc2v-2000 from Xilinx. Figure 15 shows a post-PAR (placement and routing) diagram of dynamically reconfigurable decoder. The reconfigurable modules are designed in whole column rectangular to minimize the size of Bitstream, as the reconfiguration on Virtex-II is done by entire column of CLB.

Table 1 shows some synthesis results of fixed part and reconfigurable part. In this table, the transmission time from Host to FPGA is not counted. The implemented MIMO decoder uses the dynamic reconfiguration to change the interconnections between the processing elements. It can save about 36% slice compared to the multiplexer based iterative decoder. In table I, the reconfiguration times are calculated by the size of Bitstreams and ICAP frequency. In this design, the time of the application was not precisely precalculated in order to ensure that the timing requirements are met with partial reconfiguration. This is because the Host-to-device transmission time is much longer than the decoder processing.

7 Conclusion and Future work

In this paper, we present a reconfigurable MIMO square root decoder design using dynamic partial reconfiguration, which has area efficiency, flexibility and configuration time advantage. The proposed method produces a reduction in hardware cost and allows performing partial reconfiguration, where only the interconnection of the modules are reconfigured. It is attractive for the future wireless applications, supporting different antenna sizes, different modulation and throughputs.

Currently the performance of the reconfigurable decoder is reduced due to the reconfiguration time and Host-to-device transmission time. However, these are mainly technological issues. This paper proves the management of reconfigurable decoder during data processing. The time of

reconfiguration is limited by the throughput of the serial UART interface. This time could be reduced by implementing in the FPGA a mechanism of direct access (DMA transfert) to transport the partial reconfiguration bitstreams from an internal or external memory to ICAP primitive. Moreover, the performance of reconfigurable decoder will be improved by using Xilinx FPGA Virtex-4. This is the next step of our design work. The reconfiguration of Virtex-4 is indeed more efficient than the Virtex-II (The Bitstream could be smaller due to a new structure of the Virtex-4 configuration memory based on block of configuration frames rather than whole column frames of the Virtex-II). The reconfiguration time can be speeded-up, as the bandwidth of Virtex-4 ICAP is also 8 times greater than the ICAP bandwidth of Virtex-II.

References

[1] R. Andraka. A survey of cordic algorithms for fpgas. *FPGA '98. Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*, 34(2):191–200, Feb. 22-24 1998.

[2] A. Blodget, P. James-Roxby, E. Keller, S. McMillan, and P. Sundararajan. A self-reconfiguring platform. pages 565–574, 2003.

[3] K. Compton and S. Hauck. Reconfigurable computing: a survey of systems and software. *ACM Comput. Surv.*, 34(2):171–210, 2002.

[4] F. Fons, M. Fons, E. Cant, and M. Lpez. Trigonometric computing embedded in a dynamically reconfigurable cordic system-on-chip. *International Workshop on Applied Reconfigurable Computing*, March, 1-3 2006.

[5] G. J. Foshini. Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas. *Bell Labs Technical Journal*, pages 41–57, Autumn 1996.

[6] R. Hartenstein. A decade of reconfigurable computing: A visionary retrospective. *date*, 00:0642, 2001.

[7] J. Liang, R. Tessier, and D. Goeckel. A dynamically-reconfigurable, power-efficient turbo decoder. In *FCCM*, pages 91–100. IEEE Computer Society, 2004.

[8] M.Cummings and S.Haruyama. Fpga in the software radio. *IEEE Communications Magazine*, 37(2):108–112, Feb 1999.

[9] H. Wang, P. Leray, and J. Palicot. A reconfigurable architecture for mimo square root decoder. *Reconfigurable Computing: Architectures and Applications*, pages 317–322, 2006.

[10] Xilinx. Early access partial reconfiguration user guide, ug208. 2006.

[11] S. Young, P. Alfke, C. Fewer, S. McMillan, B. Blodget, and D. Levi. A high i/o reconfigurable crossbar switch. In *FCCM*, pages 3–10. IEEE Computer Society, 2003.