A Middleware Aided Robust and Fault Tolerant Dynamic Reconfigurable Architecture

Yannick Dadji, Björn Osterloh, Harald Michalik

Institute of Computer and communication Network Engineering, TU Braunschweig
Hans-Sommer-Str.66, 38106 Braunschweig, Germany
y.dadji-foyet@tu-bs.de
b.osterloh@tu-bs.de
michalik@ida.ing.tu-bs.de

Abstract— *Dynamic reconfiguration enhances embedded system with at run-time adaptive functionality and is an improvement in terms of resource utilization and system adaptability. SRAM-based FPGAs provides a dynamic reconfigurable platform with high logic density. The requirements for such an embedded high flexible system based on FPGAs are robustness and reliability to prevent operation interrupts or even system failures. The complexity of a dynamic reconfigurable system with adaptive processing module demands high effort for the user. Therefore a high level abstraction of the communication issues is required to support application development by an appropriate middleware. To achieve such a flexible embedded system we present our Network-on-Chip (NoC) approach System-on-Chip Wire (SoCWire) and outline its performance and suitability for robust dynamic reconfigurable systems. Furthermore we introduce a suitable embedded middleware concept to support the system reconfiguration and the software application development process.*

Index Terms— *SoCWire, Network-on-Chip, Virtex, FPGA, Middleware*

1. INTRODUCTION

The need of flexibility and adaptability in the development process of embedded system for industrial applications e.g. robotics, automotive, aerospace and space drastically increased in the last decade. Traditionally, three main options are available to implement embedded systems: microcontroller platform (inclusive DSP systems), FPGAs, and ASICs. ASICs provide the highest performance, because their design can be optimally configured with respect to the application's requirements. However ASICs have a fixed configuration that cannot be adapted when the application requirements change. On the other hand microcontroller-based solutions provide only suboptimal processing speed due to the execution of software programs. Between these two extremes, FPGAs provide a compromise, which is suitable for many applications. FPGAs have a run-time adaption capability. Special processing modules could be requested on demand which is an improvement in terms of resource utilization and system adaptability. Dynamic reconfigurable systems provide these enhancements. Available dynamic reconfigurable devices, e.g. the Xilinx Virtex-4 family, provide a platform to the user with high logic density. The requirements for such an enhanced architecture are: the system needs to be robust and reliable for the industrial environment (temperature, shock) and even fault tolerant, e.g. for a harsh space environment (Single Event Effects (SEEs), a bit-flip in the configuration memory or a transient error due radiation). Furthermore, the system qualification has to be guaranteed after a module update or during the dynamic reconfiguration process to prevent operational interrupts or even system failures. Such an enhanced dynamic reconfigurable system demand deep knowledge of the system architecture and the dynamic reconfiguration process by the user. The user needs to know the allocation of the processing modules, the reconfiguration mechanism and has to prove the functional correctness of the module after update to guarantee system qualification. In order to support the application development process, the system complexity should be made transparent as far as feasible to the application developer. To reduce the system complexity for the user, the approach is to introduce a high level abstraction layer of the communication issues (hiding system details like communication links and module location to the user) as typically realized by communication middleware. Furthermore the complexity of the configuration and reconfiguration mechanisms shall also be encapsulated in simple high level functions. Consequently, the need arises to deploy an appropriate middleware for system development support. A middleware has the advantage to reduce the application development effort and thus the time to market. The deployment of a middleware on an embedded system implies additional resource utilization. This might become a design problem, since many real-time and embedded systems have tight constraints on memory footprint due to cost, power consumption, or weight restrictions [1]. Therefore, the middleware must be designed as lean as possible in term of memory utilization. Consequently, an adequate middleware must be tailored to meet the special needs of the application field implemented on the corresponding embedded system. This eliminates the possibility of adapting Commercial off-the-shelf (COTS) middleware, which is typically a generic solution without optimization of memory utilization and, in the case of our

SoCWire architecture, will not cover the special configuration and reconfiguration issues.

We have been developing special application specific communication middleware in the scope of the SFB562 Project at the Technical University Braunschweig. This project deals with new concepts for the control of parallel robots for handling and assembly tasks. In this scope we developed a PC based control architecture with the communication middleware MiRPA-X ([2][3][4]) as key component MiRPA-X supports the modular development of the control software components and encapsulates all Inter Process Communication (IPC) features. One requirement for the middleware was real time capability of the communication services with time constraint in microseconds range in order to enable a reliable control of the high dynamic parallel robot structure. Therefore, MiRPA-X obeys an application specific design and implements performance optimized synchronous and asynchronous communication.

In this paper we will propose a robust and fault tolerant dynamic reconfigurable architecture with application development aided by a specific middleware.

First, we will outline the dynamic partial reconfiguration process in Xilinx Virtex-4 and the limitation of a bus structure. Furthermore we will introduce our SoCWire architecture for dynamic reconfigurable systems and the essentials for a NoC approach. Then we will introduce the main features of the middleware MiRPA-X and discuss integration of some of these features to support the application development, the automatic configuration and reconfiguration mechanism on the SoCWire architecture.

2. DYNAMIC PARTIAL RECONFIGURATION IN VIRTEX-4 FPGA

Xilinx provides the Virtex-4 family at different qualifications level from commercial, industrial, military and even for space applications. In contrast to earlier Virtex families, e.g. Virtex-II, the internal configuration of the hardware architecture has changed. In previous FPGAs the CLBs (Configurable Logic Block) were surrounded by a ring of IOBs (Input-Output Buffer). The IOBs are now organized in columns. Additionally the FPGA is divided in clock regions, each comprising 16 CLBs. These clock regions have significant influence on the configuration process of the FPGA. Xilinx FPGAs are customized by loading configuration data into the internal configuration memory. The configuration memory is arranged in frames that are tiled about the device. These frames are the smallest addressable segment of the configuration memory space. One frame comprises 16 CLBs and therefore one clock region [5]. This architecture with clock regions and IOB structures has the advantage to overcome the limitation of partial reconfiguration in the Virtex-II architecture.

Partial Reconfigurable Modules (PRMs) do not have to occupy the full height of the device and IOBs above the top edge and below the bottom edge of the module are not part of the module resources. Therefore the logic resources left, right, top and bottom of a PRM can be used for the static area. The Virtex-4 family provides now a 32Bit data word width configuration interface (SelectMap) running at 100 Mhz which significantly decrease reconfiguration time by a factor of 8 compared to Virtex-II.

For communication between modules (static and partial reconfigurable area) Xilinx provides new unidirectional Bus-Macros in the Virtex-4 family which can connect modules horizontal and vertically. These Bus-Macros are suitable for handshaking techniques and bus standards like AMBA or Wishbone. Dedicated processing tasks, e.g. image processing, can be typically structured as a macro-pipeline with pre- and post-processing steps and require high data rate point-to-point communication as depicted in Fig. 1.

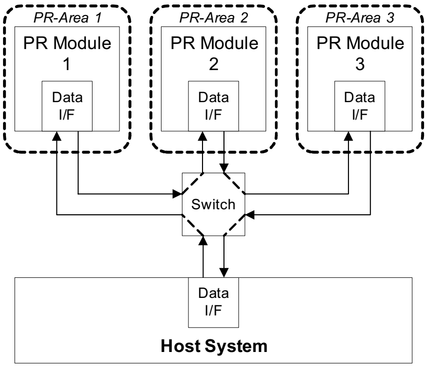

Fig. 1 Macro-pipeline system

To realize this architecture in a bus structure, multi master and bus arbitration are needed. Also bus structures are limited in the Xilinx hardware architecture. In a partial reconfigurable system a bus requires wires that distribute signals across the device. The Virtex family provides bidirectional vertical, horizontal long lines that span the full height and width of the device and 3-State buffered horizontal long lines that span the full width of the device. But these long lines are limited resources in the device: 24 bidirectional horizontal, vertical and four 3-State buffer long lines per column CLB [6]. Furthermore, the dynamic partial reconfiguration process does not have an explicit activation. New frames become active as they are written. If bits are identical to the current value, the bits will not momentarily glitch to some other value. But if the bits change, those bits could glitch when the frame write is processed. Furthermore, some selections (e.g. the input multiplexers on CLBs) have their control bits split over multiple frames and thus do not change atomically.

A fault tolerant bus structure with hot-plug ability is necessary to guarantee data integrity. With these limitations a bus structure based system would encounter the following disadvantages:

- No dedicated Bus-Macros are provided by Xilinx to access long lines which lead to manual time consuming routing.
- Failure tolerant bus structure (high efforts) with hot-plug ability is necessary to guarantee data integrity
- Dynamic reconfiguration of a PRM could block the bus and stop the system
- Limited long lines resources restrict the bus structure in data word width

The major disadvantage of a bus structure is the unpredictable behavior of the dynamic reconfiguration process in the system which could lead to block the bus and stop the system. Therefore the PRMs need to be isolated from the host system physically and logically. Our framework to physically isolate the PRM from the host system and therefore to retain the system qualification, is to subdivide the system into a static area and Partial Reconfigurable Areas (PR-Areas) which can be updated during operation. The static area remains unchanged and comprises all critical interfaces (processor, communication interfaces and memory controller). This offers the advantage that only the updated module has to be qualified in a delta-qualification step.

Furthermore to logically isolate the PRMs from the host system and with the bus structure limitation issued before we consider instead a networked architecture with a Network-on-Chip (NoC) approach providing:

- Reconfigurable point-to-point communication
- Support of adaptive macro-pipeline
- High speed data rate
- Hot-plug ability to support dynamic reconfigurable modules
- Easy implementation with standard Xilinx Bus-Macros

In order to achieve these requirements we have developed our own NoC architecture: System-on-Chip Wire (SoCWire).

3. SYSTEM-ON-CHIP WIRE (SOCWIRE)

Our approach for the NoC communication architecture, which we have named SoCWire, is based on the ESA SpaceWire interface standard [7]. SpaceWire is a well established standard in the space community, providing a layered protocol (physical, signal, character, exchange, packet, network) and proven interface for space applications. It is an asynchronous communication, serial link, bi-directional (full duplex) interface including:

- Link initialization
- Credit based flow control
- Detection of Link Errors
- Link Error Recovery
- Hot-plug ability
- Automatic reconnection after link disconnection

Thus, SpaceWire meets all requirements for a fault-tolerant NoC approach. A further advantage is that SpaceWire requires significantly small resource utilization, only.

3.1. SpaceWire

SpaceWire uses Data Strobe (DS) encoding. DS consists of two signals: Data and Strobe. Data follows the data bit stream whereas Strobe changes state whenever the Data does not change from one bit to the next. The clock can therefore be recovered by a simple XOR function. The performance of the interface depends on skew, jitter and the implemented technology. Data rates up to 400 Mb/s can be achieved. The SpaceWire character level protocol is based on the IEEE Standard 1355-1995 with additional Time-Code distribution. The character level protocol includes data character, control character and control codes. A data character (10bit length) is formed by 1 parity bit, 1 data-control flag and 8 data bits and includes data to be transmitted, as shown in Fig. 2.

The data-control flag indicates, if the current character is a data (0) or control character (1). Control characters (4-bit length) are used for flow control: A flow control token (FCT), end of packet markers (EOP or EEP) and an escape character (ESC) are used to form higher level control codes (8-14bit length) e.g. NULL (ESC+FCT) and Time-Code (ESC + Data character).

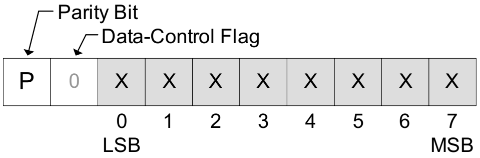

Fig. 2 . Data character

3.2. SoCWire CODEC

As mentioned before, SpaceWire is a serial link interface and the performance of the interface depends on skew, jitter and the implemented technology. For our NoC approach we are in a complete on-chip environment. The maximum character length in the SpaceWire standard without time code, which is not needed in our NoC, is 10bit (data character). Therefore we have modified the SpaceWire interface to a 10bit parallel data interface [8].

The advantage of this parallel data transfer interface is that we can achieve significantly higher data rates as compared to the SpaceWire standard. Additionally, we have implemented a scalable data word width (8-128bit) to support medium to very high data rates. On the other hand we keep in our implementation the advantageous features of the SpaceWire standard including flow control and hot-plug ability. Also the error detection is still fully supported making it suitable even for an SEE sensitive environment. For a parallel data transfer the Flow Control Token (FCT) need be included in the parallel data transfer. After initialization phase, every eighth data character is followed by one FCT to signal the readiness of the destination to receive data. The maximum data rate for a bi-directional (full-duplex) transfer can therefore be calculated by:

$$DRate_{Bi}\left[\frac{Mb}{s}\right] = f_{Core(MHz)} \times DWord\ Width - \frac{f_{Core(MHz)} \times DWord\ Width}{8}$$

For a unidirectional data transfer the flow control characters are processed in parallel and the maximum data rate can be calculated by:

$$DRate_{Uni}\left[\frac{Mb}{s}\right] = f_{Core(MHz)} \times DWord\ Width$$

Fig. 3 shows data rates for different data word width, unidirectional and bi-directional (full-duplex) data transfer at a core clock frequency of 200 MHz.

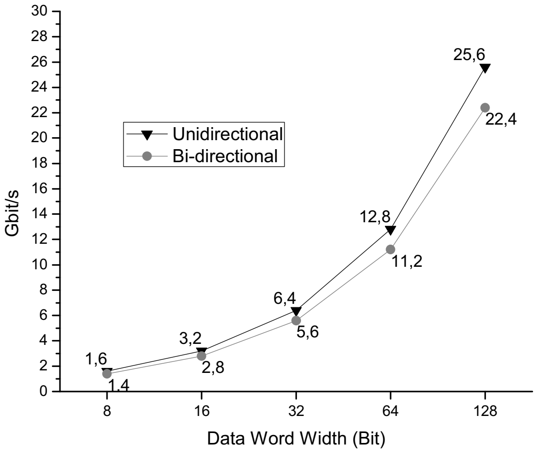

Fig. 3 SoCWire CODEC data rates at core clock frequency 200 MHz

The SoCWire CODEC has been implemented and tested in Xilinx Virtex-4 LX60-10. Figure 4 shows the occupied area, absolute values and maximum clock period.

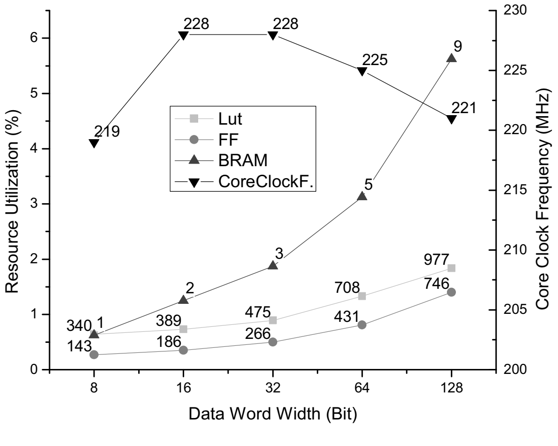

Fig. 4 SoCWire CODEC synthesis report

Fig. 3 and Fig. 4 show that high data rates are achieved with small resource utilization occupied by the SoCWire CODEC.

3.3. SoCWire Switch

To build up a network, a switch and a packet oriented protocol is needed. The switch enables the transfer of packets arriving at one link interface to another link interface on the switch, and then sending out from this link. The SoCWire Switch and its packet format are again based on the SpaceWire standard. The packet level comprises: destination address + cargo + end of packet. The destination address includes the destination identifier to support routing of packets. The cargo contains the data characters that need to be transferred from source to destination. It can be in any packet format, e.g IP packets. The end of packet marker can be either EOP for normal end of packet or alternatively EEP for exceptional end of packet as an indication of an error in the packet.

The SoCWire Switch determines from the destination address where the packet is to be routed to. Direct port addressing (packets with a port address are routed directly to one of the output ports) with header deletion has been implemented. As soon as the destination port of a packet is determined and the port is free the packet is routed immediately to that output port. The port is marked as busy and can not be accessed until the end of the packet. This is also known as wormhole routing which reduces buffer space and latency. Our SoCWire Switch is a fully scalable design supporting data word width (8-128bit) and 2 to 32 ports. It is a totally symmetrical input and output interface with direct port addressing including header deletion. The SoCWire Switch has been implemented and tested in a Xilinx Virtex-4 LX60-10. Table 1 shows the occupied area and maximum clock frequency for a 4 port switch are dependent on the data word width.

Table 1 SoCWire Switch (4 Ports) synthesis report

DWord Width	Max. f_{Core} (MHz)	Area	
		LUT	FlipFlops
8	190	1736	668
32	170	2540	1169

The SoCWire Switch basically consists of a number of SoCWire CODECs according to the number of ports and additional fully pipelined control machines. The maximum data rate is therefore equivalent to the SoCWire CODEC.

3.4. SoCWire Test and Results

We have implemented four SoCWire CODECs, one in the Host system, three in the PRMs and one SoCWire Switch in a dynamic reconfigurable macro-pipeline system, see Fig. 5. The Host system and SoCWire Switch where placed in the static area and the PRMs in the partial reconfigurable areas. All SoCWire CODECs where configured with an 8 bit data word width. The implementation of the system with reconfigurable areas could be easily implemented with the standard unidirectional Xilinx Bus-Macros. Fig. 5 shows a cut out of the placed and routed SoCWire macro-pipeline system: the PRMs (PRM1, PRM2 and PRM3) and Bus-Macros in a Virtex-4 LX 60. The static area is distributed over the FPGA.

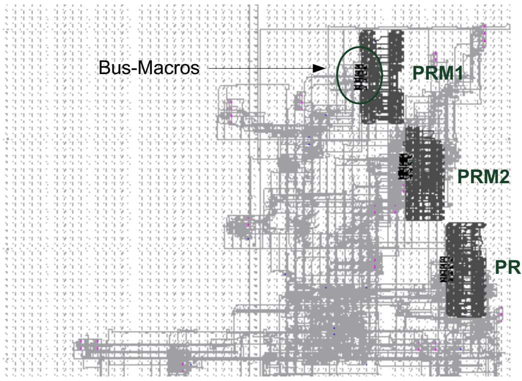

Fig. 5. SoCWire Macro-Pipeline System

The PRMs were configured as packet forwarding modules. We have tested different configuration of packet forwarding e.g. between modules, through the whole macro-pipeline system, under the condition of parallel communication between nodes. The system runs at 100MHz and the maximum data rates of the simulation could be validated to be 800 Mbps according to the selected 8-bit data word width. We dynamically reconfigured one PRM in the system. During the reconfiguration process the communication between the PRM and SoCWire Switch was interrupted, the other PRMs connections were still established. After the reconfiguration process was completed the communication between the two nodes was built up automatically within 400 ns and without any further external action (e.g. reset of node or switch). This makes the system ideal for dynamic reconfigurable systems. The Partial Reconfiguration Time (PRT) can be calculated by:

The size of one PRM was 37912 Bytes (64 Bytes command + 37848 Bytes data) and therefore the PRT 758µs (SelectMap, 8Bit data word width at 50 Mhz). For this test system the area for one PRM was set to utilize 0.6 % of the logic resources.

4. ROBUST DYNAMIC RECONFIGURABLE SYSTEM

SoCWire meets all requirements for a robust and fault tolerant dynamic reconfigurable architecture. The architecture provides system robustness and reliability. Programming a dynamic reconfigurable system is without design aids an intensive task for the user. The user needs detailed knowledge of the architecture, the processing modules, the dynamic reconfiguration process and the requalification of the system after a module update.

To simplify the application development process by hiding the system complexity to the user we propose to integrate a middleware, which is derived from the existing middlware approach MiRPA-X. Thus, we will give a short description of the MiRPA-X in the first part of this section. Then we will show how we intend to integrate the middleware in the SoCWire architecture.

4.1. MiRPA-X

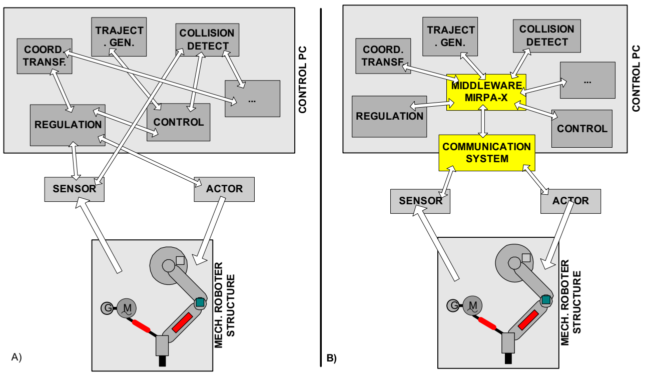

Fig. 6 PC based software control architecture: a) implementation without middleware, IPC issues require deeper system knowledge from the user, and b) implementation with middleware, no system knowledge required from IPC issues.

MiRPA-X is an object server, which transparently handles data and procedure requests. Its lean design is optimized for high communication performance. It supports the client/server implementation pattern, therefore control software components are registered as client or server in the MiRPA-X environment. One of the key features of MiRPA-X is the ability to hide the identity of servers from clients; so every data transfer is content-based, not structure-based. This enables modifying the whole control system structure dynamically without recompilation or restart of system components in general. Fig. 6 shows exemplarily the realization of a PC based control system consisting of multiple software components exchanging data with each other to perform a specific control task. On the left side a), a software component is directly linked to all components it communicates with. The user must know all the communication links available on the system. A dynamic reconfiguration of the system is quasi impractical. On the right side b) the middleware MiRPA-X is introduced as the exclusive communication partner for all software components. The connection to the object server of the middleware is encapsulated in high level API functions. Therefore the user must not know any system information while developing the control components. MiRPA-X supports the IPC; this can be realized in both synchronous and asynchronous way. The synchronous way consists of blocking request/reply pattern. The asynchronous way consists of non blocking command messages send over the middleware to the corresponding target component. Additionally MiRPA-X supports IPC over shared memory region. This is designated for software components with tight data coupling. Since the shared memory access has no inherent synchronization mechanism, access conflicts and data integrity violation may appear when many components simultaneously access the same shared memory. To solve this problem, MiRPA-X introduces a real-time sequence control engine, the token manager. The token manager defines a high priority token cycle that includes all registered components. To ensure data integrity, the token manager enforces a sequential shared memory access of the registered components.

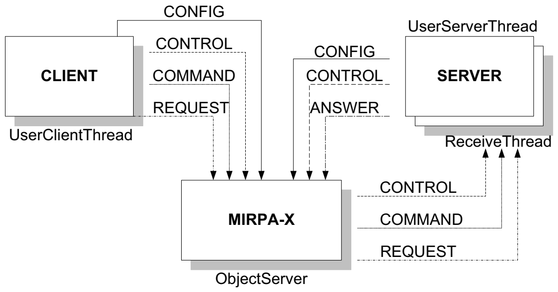

Fig. 7 Message types in the MiRPA-X environment

Fig. 7 shows the general message types, which are transmitted in the MiRPA-X environment. All requests are passed to MiRPA-X, which then routes the requests to their ultimate targets and back, if an answer is required. MiRPA-X supports four different message types:

- REQUEST / ANSWER – transmission of service requirement and waiting for service delivery.
- COMMAND – transmission of non blocking procedure call
- CONFIG – service and resource (shared memory) registration
- CONTROL - request of statistical information about resource and service delivery status

4.2. Middleware Aided Dynamic Reconfiguration

Fig. 8 shows an example of a dynamic reconfigurable architecture. It consists of a System-On-Chip (SoC) design with additional external resources. The on-Chip design is organized in a static and a Partial Reconfigurable (PR) area. The static area comprises the host system, the SoCWire switch, the I/O controller and the memory controller. On the host system, a PowerPC is installed and the operating system QNX is running. The I/O controller enables the integration of additional peripheral devices. The memory controller enables the integration of additional external application memory. To avoid a bottleneck by the memory access, the memory controller implements a multi port (0..n) connection to the SoCWire switch. On the host system, the middleware MiRPA-XE (additional extension E for “Embedded”) is deployed. The PR area can be dynamically reconfigured on demand with application dependant hardware modules at run time. In Fig. 8 the PR area is exemplarily configured with the modules PRM 1, 2, und 3. The external resources consist of the configuration memory where all hardware modules are stored, the shared memory region (which could also be on-Chip for small systems) for data exchange between software applications and PRMs, the I/O driver and the application memory space used by the PRM processing. To support the reconfiguration process, the middleware needs to know the corresponding basic configuration information of every PRM. This configuration information consists of the module name, a unique module identifier, a list of services provided and a global module status. To publish the configuration information each module implements a dedicated memory location which contains the configuration information. At the start up, the middleware will extract all configuration information and set up a configuration database and a Look-Up table (LUT) for a dynamic run time mapping of service request into module allocation.

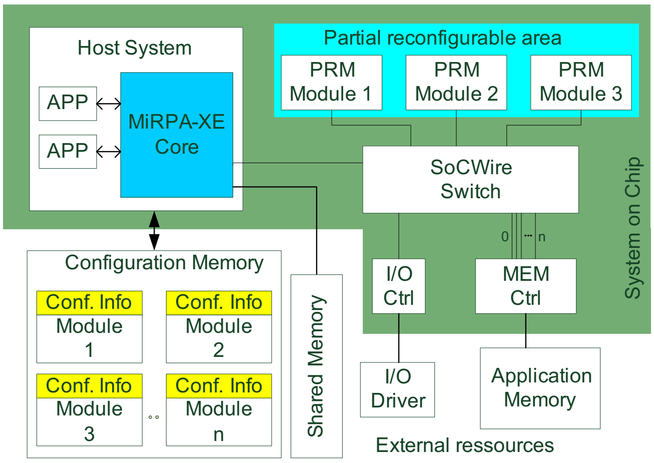

Fig. 8 Dynamic reconfigurable architecture

The middleware offers a high level API function to support the hardware reconfiguration process. The API function encapsulates all the reconfiguration process, therefore transparency is provided since the user needs not to handle with the complex reconfiguration mechanism. The parameter needed for the reconfiguration process is the name of the PRM to configure. The API function resumes the module reconfiguration process and loads the corresponding hardware code from the configuration memory into the PR areas (e.g. through the Xilinx ICAP interface). After the module reconfiguration process is completed the SoCWire system automatically builds up a link connection to the module. The module signals its activity by sending a message to the host system. The Middleware then operates certain tests (test vectors, timing constraints) to validate the correct behavior of the module and to guarantee system qualification. After a successful validation the module is ready for processing. The Middleware sends the results of the reconfiguration process (reconfiguration successful or failed) to the user.

4.3. Middleware Aided Application Development Process

Beside the reconfiguration process of the PRM the middleware also supports the application development on the host system. For this purpose three basic high level functionalities of MiRPA-X will be adapted to the embedded environment: the shared memory access management (analogue to 4.1), the processing request (derived from the MiRPA-X request /reply communication model) and the task activation service (derived from the MiRPA-X non Blocking COMMAND messages). Each processing required by an application is coded in the MiRPA-XE environment as a service identified by a unique name within the system. In this way transparency is provided, since the user does not need to specify the PRM or the software application that should process the service. If a software component performs a PRM processing request service, it sends a request message containing the service name to the middleware. To figure out which PRM is concerned by a service request, the middleware uses its internal configuration database. Subsequently the middleware issues a service request packet and forwards it via the SoCWire system to the corresponding PRM, when the latter is active. Upon reception of the packet the PRM starts processing. The data to be processed can either be compiled in the request packet (synchronous processing) or passed through a shared memory (asynchronous processing). By a synchronous processing, the software component blocks until it receives the processing result. When the PRM completes the processing it sends back the processing results via the SoCWire system to the middleware. The middleware then compiles the result in a reply message and sends it back to the former software component. If for any reason the PRM does not reply to the request (i.e. the PRM is out of order), then calling process will remain blocked. This may lead to the collapse of the overall application. To avoid this, timeout information will be appended to each blocking service request. This way, a calling process will unblock with the corresponding timeout error if the target PRM does not answer a request within the specified time.

Task activation services can be used in collaboration with shared memory communication. When a software component wants to perform task activation, it sends a non-blocking task activation message to the middleware. The middleware forwards the message via the SoCWire system to the corresponding PRM. Upon reception of the activation The PRM starts the task processing. It processes data contained in a prior specified shared memory region and write the processing results a second shared memory region. To communicate the processing results, the PRM may use two different ways. By a small data set of processing result, it may pass a message to the middleware. By a large data set, the PRM may write the processing result in a shared memory region and set a task completion flag. Then it would send a task completion signal to the middleware and the latter would forward the signal to the corresponding software component.

The MIRPA-XE is realized as a soft- and hardware module. The software module uses the QNX services to implement the API functions relative to the processing request and task activation services. In the hardware module, a SoCWire interface converts the message passing data into SoCWire packets and vice versa.

With the assistance of the middleware the complex hardware reconfiguration process is made transparent for the user. Also the communication mechanisms derived from MiRPA-X ease the software application development and to reduce the development time.

5. CONCLUSION

Dynamic reconfiguration enhances embedded system with at run-time adaptive functionality and is an improvement in terms of resource utilization and system adaptability. To meet these requirements, an improved communication architecture with NoC approach is required to guarantee system qualification and to prevent operational interrupts. Furthermore a suitable framework is needed to support the complex system reconfiguration mechanism and the application development.

In this paper we presented our NoC approach SoCWire. SoCWire meets all requirements for a high speed dynamic reconfigurable architecture. High data rates are achieved with significantly small implementation efforts. Hardware errors detection, hot-plug ability and support of adaptive macro-pipeline are provided. The implemented test application demonstrates the suitability of SoCWire for robust dynamic reconfigurable systems. Furthermore, we introduce a middleware concept to support the complex system reconfiguration process and the application development at the user level. Through the middleware support the configuration process is made transparent for the user. Finally we presented three high level communication functionalities which reduce the complexity of the communication issues by the software development process on our SoCWire system.

ACKNOWLEDGMENT

The authors highly appreciate the support given by the German Research Foundation (DFG) within SFB562 and by QNX Software Systems providing free software licenses.

REFERENCES

[1] R. Joost and R. Salomon, “Advantages of FPGA-Based Multiprocessor Systems in Industrial Applications”, *Industrial Electronics Society*, 31st Annual Conference of IEEE, 2005. IECON 2005.

[2] Y. Dadji, H. Michalik, T. Moeglich, J. Steiner, “Performance optimized Communication system for high-dynamic and real-time Robot Control Systems”, *CD-ROM proceedings of the 16th. International Workshop on Robotic in Alpe-Adria-Danube Region,* 7-9 June 2007, Ljubljana, Slovenia.

[3] N. Kohn, J.-U. Varchmin, J. Steiner, U. Golz, “Universal communication architecture for high-dynamic robot systems using

QNX", *8th International Conference on Control, Automation Robotics and Vision*, Kunming, China, pp. 205- 210, 2004.

[4] Y. Dadji et al., "Networked Architecture for Distributed PC-based Robot Control Systems", *International Conference on Automation, Robotics and ControlSystems*, 7-10 July 2008, Orlando, Florida

[5] Xilinx, *Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Functional Description*, www.xilinx.com, September 2005

[6] Xilinx, *Virtex-4 Configuration Guide*, www.xilinx.com, October 2007

[7] ECSS, *Space Engineering: SpaceWire–Links, nodes, routers, and networks*, ESA-ESTEC, Noordwijk Netherlands, January 2003, ECSS-E-50-12A

[8] B. Osterloh, H. Michalik, B. Fiethe, K. Kotarowski. "SoCWire: A Network-on-Chip Approach for Reconfigurable System-on-Chip Designs in Space Applications." In NASA/ESA Conference on Adaptive Hardware and Systems, Volume (AHS-2008), pp 51-56, Noordwijk, June 2008