PARTIAL RECONFIGURATION CONCEPT IN A SCA APPROACH

Michel Sarlotte (Thales, Colombes, France; michel.sarlotte@fr.thalesgroup.com)
Bruno Counil (Thales, Colombes, France; bruno.counil@fr.thalesgroup.com)
Paul Gelineau (Thales, Cholet, France; paul.gelineau@fr.thalesgroup.com)
Remy Chau(Thales, Colombes, France; remy.chau@fr.thalesgroup.com)
Daniel Maufroid (Thales, Colombes, France; daniel.maufroid@fr.thalesgroup.com)

ABSTRACT

Continuous services improvement for Software Defined Radio system leads to use non Corba capable devices such as DSP and FPGA to meet the data rate requirements. The current version of the SCA (Software Communication Architecture) which provides solution for GPP implementation does not covered properly FPGA platform even if latest releases introduces MHAL [1] concept. This paper describes some results achieved in using partial reconfiguration Virtex capabilities. We focus especially on the technologies to manage partial reconfigurability on non Corba enabled devices.

1. INTRODUCTION

JTRS program since the end of the 90's tried to define common approach to facilitate the portability of waveform and the reconfigurability of a SDR receiver. The objective is to be able to port any waveforms onto any platform. This leads to the SCA [2] and Core framework concepts suitable for pure SW component such as GPP. Application of this approach to DSP is under progress and preliminary analysis and proposal has been raised for FPGA and several upstream programs work on this topics. FPGA technology provides intrinsic reconfiguration capability but full configuration is time consuming and in most of the cases does not meet the system availability requirement.

The paper focuses on the partial reconfiguration technology and process itself and not on the portability features. It presents briefly the targeted architecture and the applied design flow to use partial reconfiguration. HW and SW architecture implemented within the SoPC are described. We conclude by the perspective offered by this technology and by an identification of complementary upstream studies to be held to solve the remaining key issues.

2. PLATFORM OVERVIEW

The demonstrator platform is based upon a ML405 evaluation board delivered by Xilinx composed by a Virtex-4 FX20 component and several transceiver components for connectivity purpose such as USB, Ethernet, … .The board provides also audio and video facilities. An overview of the physical demonstrator and its environment is provided below (fig 1).

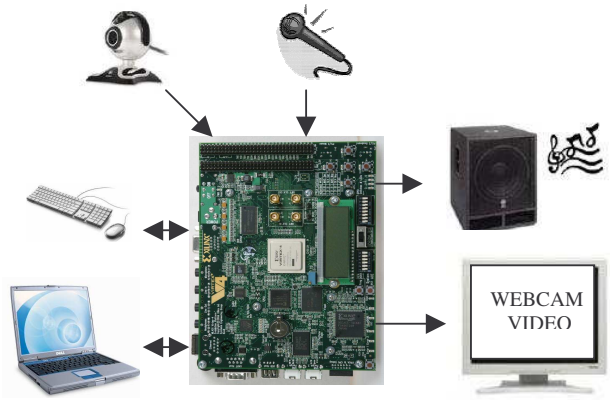

Figure 1: Demonstrator overview

2.1. Hardware architecture

The HW architecture is build around the PPC405 embedded within the FPGA FX20 devices. Classical interfaces are directly managed by the core and the embedded SW. The reconfigurable module is plugged onto the OPB bus and fully drives the audio interface.

The FPGA is the main component of the demonstrator. Indeed, the Virtex-4 FX includes a PowerPC 405 that controls:

- The partial dynamic reconfiguration through the Internal Configuration Access Port (ICAP).
- The non-stop display of a video controlled by the PPC 405, through the camera connected on the USB controller and the TFT screen connected on the VGA chipset.
- The dialog with the GUI residing on the laptop by Ethernet link.
- The SystemAce chip which allows the PowerPC 405 to access to a Compact Flash memory which hosts the Operating Environment (OS, …).
- Audio processing modules

Concerning the audio processing part of the demonstrator, the FPGA includes a controller which manages the AC97 Codec chip. A microphone and a speaker are connected to this chip. The AC97 Codec is used as a ADC/DAC. The audio processing module resides on a dynamic reconfigurable region of the FPGA so the user can change audio effects on the voice by activating the partial reconfiguration.

2.1. Software architecture

The software architecture of the demonstrator aims to provide an architecture compliant with SCA and which enables to take advantage of the Xilinx Virtex 4 partial reconfiguration technology. The Operating Environment is based on the Linux Monta Vista Operating System, MICO ORB the Thales CoreFramework.

The SCA enables applications (typically radio waveforms) to be deployed dynamically into a hardware platform. The hardware platform is abstracted by a set of software components named logical devices in the SCA specifications. These logical devices have two functions: to enable applications to access to hardware resources in an independent way of their implementation, and to enable to load and to execute software on processors of the platform (GPP, DSP, FPGA). Management of application deployment is realized by another software module named Core Framework (CF). When the CF has to deploy an application, it relies on the logical devices present in the platform to load and to execute components of the application on the different processors of the platform.

Architecture of the demonstrator focalizes on the definition and the realization of an SCA compliant logical device which enables a CF to load partial bitstreams in FPGA reconfigurable areas. First element is to define the granularity of device(s) compared to the number of reconfigurable areas. It is possible to make one SCA logical device to reconfigure all areas, or a device for each area. The second option was chosen for the demonstrator. This option enables to define a more flexible architecture. It is easier to adapt the architecture when the number of reconfigurable areas changes, it just has to be added or removed an instance of the device.

Software architecture of the demonstrator is composed of FPGA Area Devices to load dynamically partial bitstreams in reconfigurable areas, a GPP Device to load and to execute software on PPC 405 and of a CF software which manages deployment of applications on reconfigurable areas and on PPC 405 GPP.

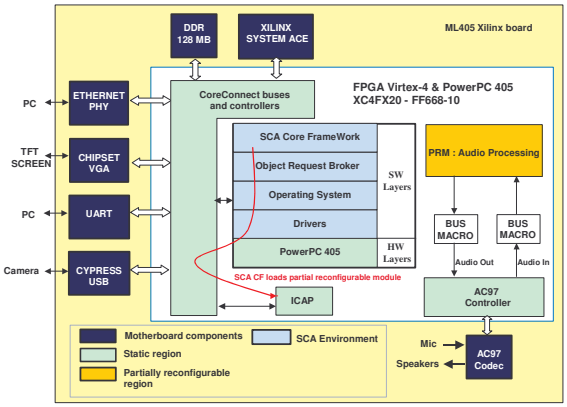

Figure 2: Detailed HW/SW architecture

In the demonstrator, only one reconfigurable area was defined, so only one FPGA Area Device was instantiated. It enables the CF to load dynamically the content of its corresponding reconfigurable area, using the ICAP driver located in the operating system.

3. DESIGN FLOW

Standard design flow is dedicated to generate complete bitstream not taking into account partial reconfiguration. To succeed in using this innovative technology dedicated approach has to be applied. The main issue is related to the physical view of the module and more precisely the "footprint" i.e. the physical interface of the partial reconfigurable module must be identical for the different versions and the static design could be considered as a socket.

In this paragraph, we will present the hardware workflow and provide general guidelines to develop reconfigurable platform. The figure below shows the hardware design flow and the tools used at each step:

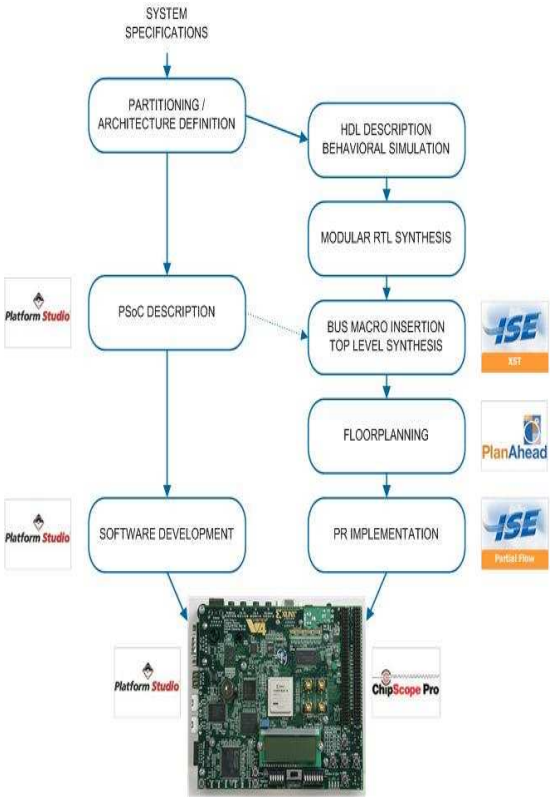

Figure 3: Design flow

Step 1 – HDL description & behavioral simulation
Partial reconfiguration requires a hierarchical design approach that must be strictly followed during the HDL coding process. Global logic such as IO buffers, clock buffers and DCMs must be in the top-level design unit. Multiple base design modules and partially reconfigurable modules are instantiated as black boxes in the top-level module. In our case, the system integrates a PPC405 sub-system generated by EDK tools from Xilinx [3], [4] and completed by our own content. However, the instantiated DCM required by the processor and the SDRAM controller have been displaced from the intermediate level to the top-level.The reconfigurable modules cannot contain any clock-related primitives.

All PR modules for a given reconfigurable region must be pin compatible with each other, i.e. have the same port definitions and entity names. Consistent naming allows each of the PR modules to be linked from the same top-level description.

At this time, the complete VHDL model is available to perform the behavioral simulation and to verify that the functional requirements are met. The reconfigurability aspect is not managed here; all the versions of the reconfigurable modules have to be validated separately.

Step 2 – Modular RTL synthesis
The PR flow requires separated netlists for each module and preserved hierarchy at the top-level.All the modules instantiated at the top-level are synthesized one at a time in macro mode (no automatic IO insertion) using standard synthesis tool. The various IP used is generated by EDK or the Coregen utility from ISE.

Step 3 – Bus macros insertion & top-level synthesis
As previously explained, the physical footprint of the module must be unchanged for the various configurations of the module. Specific cell, called Bus macros (BM) , have been created by Xilinx in order to lock the routing between the PR modules and the static part of the design. BMs are slice-based pre-placed and pre-routed hard macros that can handle up to 8 bits of data in a fixed direction and offer an enable control. These bus macros are inserted on every signal of the PR modules interfaces, except for the global signals such as clocks. The type of bus macro is mainly chosen depending on topological criteria (edge of the PR region, signals direction).

The structural top-level HDL description – a collection of black boxes instantiations – is synthesized.

Step 4 – Floorplanning
The PR implementation flow is tightly driven by placement constraints provided in the User Constraints File (UCF). PlanAhead is a graphical floorplanning tool able to create these constraints.

A PlanAhead project is created for each version of the PR modules. The netlists are imported in the tool just as the initial IO location constraints.

PlanAhead brings the notion of “Pblock“. A Pblock gathers one or more design units on which a set of attributes may be assigned. The PR modules are Pblocks characterized by an area range and a “RECONFIG” mode. The range allocated to a PR Pblock must encompass all the logic resources needed for the module. PlanAhead is able to estimate the amount of required/available resources in a given Pblock. All the static modules belong to the same Pblock which does not have any area range attribute.

All the Virtex-4 primitives instantiated in the top-level netlist have their location to be locked. It is particularly the case for the clock primitives (DCMs, BUFG) and the bus

macros. In PlanAhead, this task is performed by dragging and dropping the primitives onto the floorplan view.

At this stage, PlanAhead performs a design rule check in order to identify potential violations hard to fix during the P&R. Afterwards, PlanAhead exports all the data required by the place & route step. The "Partial Reconfig. Mode" has to be used to perform this export. All the information are stored in the final UCF file.

Step 5 – Non-PR implementation & timing verification
This step is crucial for design debug, aids initial timing and placement analysis, and helps in determining the best area range and bus macros locations.

The place & route process can be driven by PlanAhead after the design has been exported in "Area mode", or performed by means of Xilinx ISE tool. Because the PR flow does not provide any simulation model, a post place & route simulation can only be achieved at that time.

Step 6 – PR implementation
PlanAhead is used again to manage the PR implementation after the design has been exported in "Partial Reconfig. mode". For each step of the PR flow, the designer has to customize some parameters and PlanAhead launches the Xilinx implementation commands in batch mode.

The Budgeting step builds a skeleton of the top-level design.
The Static Logic Implementation step performs the place & route of the non reconfigurable part of the design, including the PPC405 sub-system and the ICAP primitive used in the reconfiguration of the FPGA. The PR Module Implementation step performs the place & route of ONE version of the reconfigurable modules.

The final Assemble step merges the static part and all the PR modules to create bit files for configuring the FPGA. The PR flow provides a full bitstream for the initial configuration of the FPGA and, for each PR region, a partial bitstream corresponding to the implemented module plus a blank bitstream intended to "clean" the PR region.

Once the PR flow parameterization is well settled, PlanAhead can generate script files which could be used to generate new version of the design

Step 7 – Platform integration
This step consists in finalizing all the files to load onto the target board. The various operations are managed by script files based on standard Xilinx commands (data2mem, promgen). The boot software is incorporated into the full bitstream which will be stored in the configuration memory connected to the FPGA. The application software is programmed in the flash memory. At start up, the boot software copies the application software to the SDRAM and launches its execution. The partial bitstreams are arranged in a dedicated table located in the flash memory. The application software reads the reconfiguration data in the flash memory and sends them to the ICAP.

The design flow described above is operational and allow to develop reconfigurable devices even if some limitations still exist in the various tools due to their youth. Strong improvement has been achieved during the last years.

4. RESULTS AND PERSPECTIVES

The feasibility of dynamic on the fly reconfiguration has been assessed by this demonstrator leading to really flexible solution. Issues are related to the complete design flow and mainly the SW version consistency. Indeed, the reconfigurability is managed by the PPC and the SW stack running on it. The ICAP driver generated by the HW design flow should be compliant with the OS, CoreFramework and the ORB running on the power PC. Furthermore, stack layer for standard IP such as Ethernet, USB provides also some constraints in term of SW configuration. Prospects relative to the SDR demonstrator with FPGA partial reconfiguration are multiple.

The main result of the study is the proof of the concept, i.e. it is possible to use the FPGA partial reconfiguration technology in an SDR environment.

Integration of this technology in SCA-compliant architectures enables to reduce time necessary to reload FPGA firmware. This is important, as requirements for time of boot and waveform switching is very constrained in SDR terminals, especially when it is compared with previous legacy terminals. It also enables to reduce size of memory necessary to store the firmware, because only firmware which is loaded dynamically has to be stored with other waveform parts. The optimization of the FPGA size leads to a static power consumption reduction compared to a complete

This solution also offers the possibility to reduce the Bill-of-material as the size of the FPGA could be optimized embedding only the treatment used at each time. A multi-waveform terminal could take advantages of such technology which allows to download transparently one business code in parallel of the current execution of another one.

To extend down to hardware the SCA approach of portability, interoperability and reconfigurability of components based waveform applications, a unified specification/design flow and execution environment have to be provided to developers by SDR platforms.

System and user services provided and required between HW and SW components ports should be defined independently of target programming languages thanks to IDL or Xilinx Specific Language (DSL approach).

In a more conceptual view, the possibility to load dynamically and partially a firmware in an FPGA can be considered as a first step to design this firmware in a more global software approach. For example, design of firmware can be integrated in a Model Driven Architecture (MDA)/ Model Driven Engineering (MDE) or even in an approach based on a component framework.

5. CONCLUSION

Dynamic partial reconfiguration provides really flexible approach to deploy SDR terminal. Remaining actions have to be performed to stabilize the SW environment and to achieve a consistent design environment including HW-SW tools and drivers). Second main issue is related to the connectivity to achieve a transparent deployment i.e. extend the SW abstraction approach down to the HW.

6. REFERENCES

[1] JPEO JTRS, Modem Hardware abstraction Layer (MHAL API, version 2.11.1, May 2007.

[2] JPEO JTRS SCA v2.2

[3] VIRTEX-4 user guide

[4] VIRTEX-4 Configuration guide