A CUSTOM BUS MACRO GENERATOR FOR FPGA DYNAMIC PARTIAL RECONFIGURATION.

Fabio Cancarè

email: fabio.cancare@microlab-mi.net

ABSTRACT

This work aims to improve the Caronte Hardware Architecture by introducing a new kind of Bus Macro, called Custom Bus Macro, instead of the Xilinx Bus Macro currently used. Such new hardware architecture is called YaRA, Yet another Reconfigurable Architecture, and actually represents an open DRESD research topic.
The main benefits introduced by this Custom Bus Macro are design facilitation and the possibility of go on with all possible communications during an area reconfiguration (Obviously communications that involve this area are excluded).

1. STATE OF THE ART

Caronte is a methodology for implementing dynamic reconfigurable systems on FPGA. It's based on standard development kit, such as EDK. The Caronte flow, reported in Figure 2, is basically composed in three phases:

HW-SSP Phase

In this initial phase the goal is to identify all the necessary EDK systems description that are going to be placed onto the FPGA for a certain time. HW-SSP stands for HardWare Static System Photo Phase.

Design Phase

In this second phase the goal is to prepare all the designs in order to compute all the bitstream to physically implement the embedded reconfiguration of the FPGA. In particular during this phase the Xilinx Bus Macro are placed in order to allow communication between modules.

Bitstream Creation Phases

In this final phase all the bitstreams needed to implement the system description are computed. The system is now ready to be put onto FPGA exploiting Dynamic Partial Reconfiguration.

Caronte Hardware Architecture

Caronte Hardware Architecture is constituted by several elements such as a processor, memories, a set of reconfigurable devices and a reconfiguring device. The reconfigurable devices are placed in the Reconfigurable part of the hardware architecture while the remaining elements constitutes the Fixed part.
Actually Caronte Harware Architecture relies on Xilinx Bus Macro for enabling communication between two reconfigurable modules or between one fix module and one reconfigurable module. This situation is depicted in Figure 1.

Fig. 1. Xapp290 scenario. Communication channels requires one or more Bus Macros

In order to enable this kind of communication it's necessary to respect the constraints specified in [2]. In particular:

- The reconfigurable module height must al-

ways be the full height of the device.

- The Reconfigurable module width must range from a minimum of four slices to a maximum of the full-device width, in four-slice increments.
- Horizontal placement must always be on a four-slice boundary; the leftmost placement being x = 0, 4, 8,...
- If a signal ”passes through” a reconfigurable module connecting the two modules on either side of the reconfigurable module, bus macros must be used to make that connection. This effectively requires creation of an intermediate signal that is defined in the reconfigurable module. The signal cannot be actively used during the time the reconfigurable module is being configured.

As can be seen in Figure 1, communication between non adjacent modules is hard to achieve, and needs intermediate signals that cannot be used while reconfiguring an area among the two modules.
However, the actual Caronte Hardware Architecture, widely described in [1], relies on Xilinx Bus Macro. The Caronte Flow, also described in [1], is reported in Figure 2.

2. CURRENT WORK

The analysis of the state of the art has shown that using Xilinx Bus Macro has several negative effects. First of all, while reconfiguring a reconfigurable area, all the communication channels crossing that area must be disabled, as specified in [2]. Moreover, using Xilinx Bus Macro implies that, in order to enable communication between two non adjacent modules, as many Bus Macros as the number of boundaries Reconfigurable Area-Reconfigurable Area and Reconfigurable Area-Fix Area crossed by the communication channel are needed. This obviously results in a greater design complexity, as shown in Figure 1. Aim of this work is creating a new type of Bus Macro, by now referred as Custom Bus Macro, to be inserted

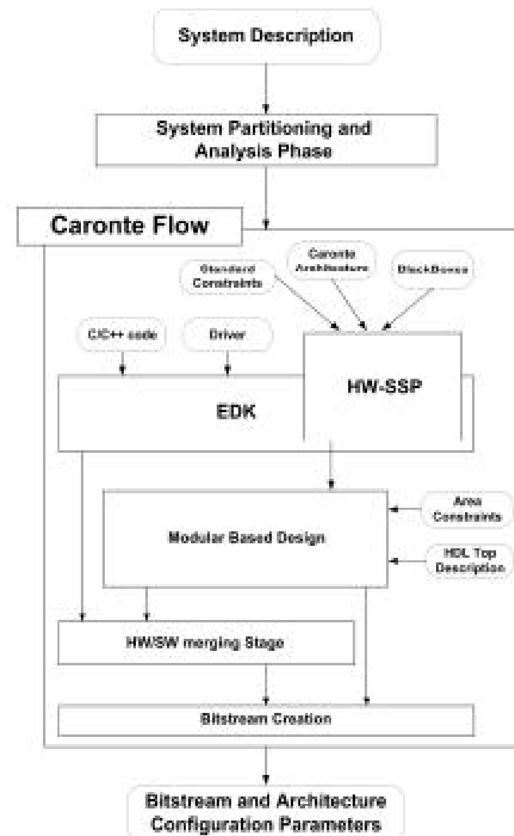

Fig. 2. Reconfiguration Design Methodology Flow.

into Caronte flow, that goes over the previously described negative effects and simplifies the design of Dynamic Partial Reconfigurable Systems.
An experimental, handmade Bus Macro has already been created by the DRESD Team for a specific test architecture for a Xilinx Virtex II Pro FPGA on an Avnet evaluation board that implements a point-to-point communication between a Fix area and a Reconfigurable area. However, this application specific Bus Macro is not what this work is looking for, because it is not flexible with respect to the number of areas and the width of each area and enables only point-to-point communication. Instead a flexible, multipoint Custom Bus Macro is needed, which allows the creation of architectures like the one shown in Figure 3.

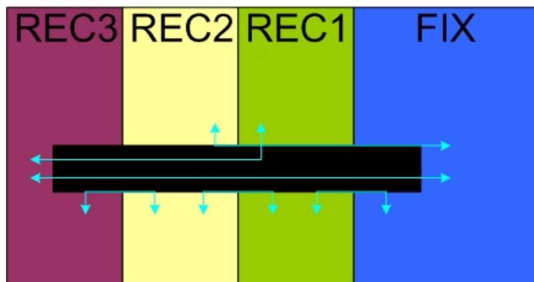

Fig. 3. YaRA scenario. A single Bus Macro enables all the possible communication channels

The Custom Bus Macro has also to be compliant with the Wishbone specification [3], as stated in a previous DRESD analysis.
In order to bring as much flexibility as possible a Custom Bus Macro generator must be created. Such generator must take as input parameters the number of reconfigurable area, the width of each area, the name of the Custom Bus Macro and the name of each reconfigurable area.
As output this Custom Bus Macro generator must render the Hard Macro(To be inserted into the appropriate Caronte directory) and the VHDL description (To be inserted into the top design VHDL file) of the needed Bus Macro, as describe in Figure 4.

While the VHDL description is quite straightforward, the Hard Macro generation is a tricky process. In order to obtain this Hard Macro, sev-

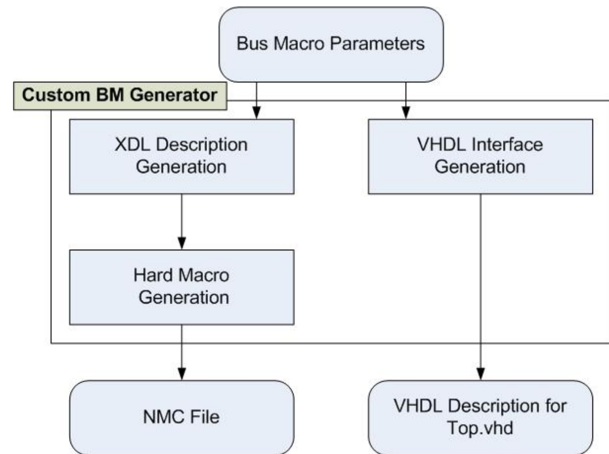

Fig. 4. Custom Bus Macro Generator Flow

eral approaches are possible:

- Starting from a VHDL file of a generic Bus Macro, performing synthetization, mapping, placement, routing and then modifying through FPGA Editor the logical design file obtained;
- Starting from an empty XDL file, adding all the necessary information in order to build a Bus Macro and then converting in NMC (Hard Macro) File.

The latter approach is the one followed in this work, while the former is the approach that has led the DRESD Team to build the experimental Bus Macro described before. Obviously the XDL approach is more feasible, the script must simply generates an ASCII file with XDL extension and then converts this file into an NMC file using a shell command provided by Xilinx. However, creating XDL files is not as straightforward as it could seems. In fact, Xilinx does not provide information about XDL - Xilinx Definition Language - probably because XDL is used as an internal, intermediate representation in the ISE flow.
A careful analysis of available XDL files was then performed, in order to understand the syntax and the semantic of Xilinx Definition Language. Moreover, due to some compatibility problems between the ISE version used during this work, 8.1i, and XDL files created with previous version, additional examples of XDL file were needed. They were

obtained using FPGA Editor, a Xilinx utility provided with ISE pack.
After the analysis the XDL syntax and semantic became clear. In the following the XDL constructs used by the Custom Bus Macro Generator are briefly explained. The Custom Bus Macro layout desired is reported in Figure 5.

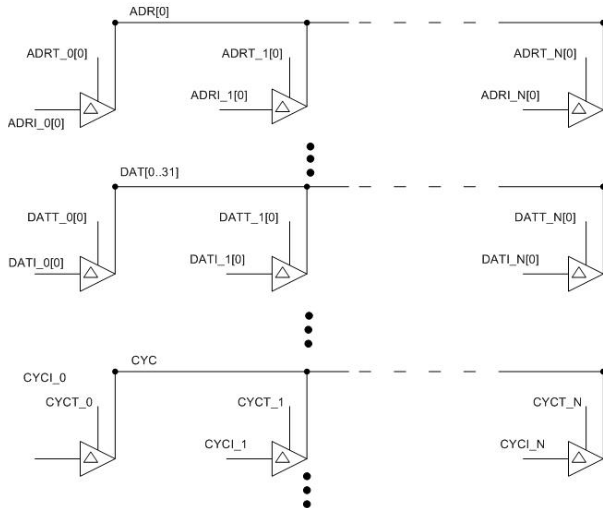

Fig. 5. Custom Bus Macro Layout

```
design "__XILINX_NMC_MACRO" xc2vp7ff896-5;
module "moduleName" "m0t0" ,
cfg "_SYSTEM_MACRO::FALSE" ;
```

The code snippet above is the XDL file's header. In the first row, the string xc2vp7ff896-5 indicates the FPGA model referred by the XDL file, while in the second row moduleName is the name of the XDL file and m0t0 is the reference component's name of the hard macro (this component has to be instantiated after with all the others ones).

```
inst "m0t0" "TBUF" , placed R40C3 TBUF_X4Y0,
cfg "IINV::I TINV::T _SUPERBEL::TRUE"
;
inst "m0t1" "TBUF" , placed R40C3 TBUF_X4Y1,
cfg "IINV::I TINV::T _SUPERBEL::TRUE"
;

...

inst "m1t0" "TBUF" , placed R40C7 TBUF_X28Y0 ,
cfg "IINV::I TINV::T _SUPERBEL::TRUE"
;
...
```

The construct reported above lets the components' instantiation. All the components instantiated within the Custom Bus Macro are tristates. The first number of the component's name refers to the module's number while the second numbers refers to the tristate number within the specific module.

```
port "adrI0(0)" "m0t0" "I" ;
port "adrT0(0)" "m0t0" "T" ;
port "adr(0)" "m0t0" "O" ;
port "adrI0(1)" "m0t1" "I" ;
port "adrT0(1)" "m0t1" "T" ;
port "adr(1)" "m0t1" "O" ;
port "adrI1(0)" "m1t0" "I" ;
port "adrT1(0)" "m1t0" "T" ;
port "adr(0)" "m1t0" "O" ;
```

This code snippet simply allows the naming of the tristate's ports. The rows reported are relative to the components instantiated in the example before. As can be seen, the output signal's name is the same for every tristate belonging to the same Custom Bus Macro line.

```
net "adrI0(0)_I" ,
inpin "m0t0" I ,
;
net "adrT0(0)_T" ,
inpin "m0t0" T ,
;
net "adrI0(0)_I" ,
inpin "m1t0" I ,
;
net "adrT0(0)_T" ,
inpin "m1t0" T ,
;
net "adr(0)_O" ,
cfg "_NET_PROP::IS_BUS_MACRO:" ,
outpin "m0t0" O ,
outpin "m1t0" O ,
;
```

This final XDL code snippet is relative to the nets instantiation. Above are reported the nets that refer to the tristates m0t0 and m1t0. With these nets, the Bus Macro line creation is complete (For sake of simplicity the nets relative to m1t0, a component reported in the example before, are omitted).
Once that the XDL file is generated, it's necessary to convert it in a macro hardware file (NMC). This is achieved, as said before, using a Xilinx bash command, xdl, as shown below:

```
%Folder%\xdl -xdl2ncd moduleName.xdl
```

Once that this operation is done, the hard macro is ready to be put into the appropriate Caronte folder. Before starting the Caronte flow (Acheronte) it's however necessary to modify the top design file (Usually named top.vhd) with the vhdl description of the bus macro. This description is also generated by the Custom Bus Macro Generator, however this is a trivial function and do not need further explanations. Finally, in Figure 6 is shown where the Custom Bus Macro Generator interacts with the Caronte flow.

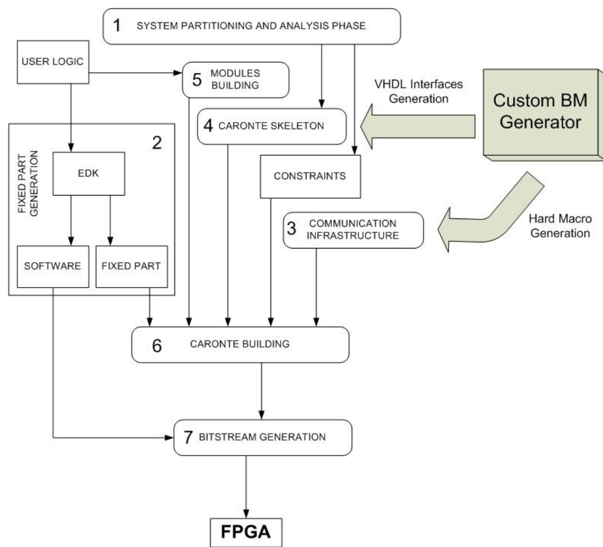

Fig. 6. Custom Bus Macro Generator and Caronte Flow

3. TESTING

In order to test the correctness of bus macro instances created with Custom Bus Macro generator a dummy test architecture was created for the Xilinx Virtex II Pro FPGA on an Avnet evaluation board. This simply architecture consist of two areas, one fix and one reconfigurable. Both the modules that are placed on these areas implement a simple finite state machine. Obviously communication is achieved using a bus macro instances created with the generator.
The system works in the following way: Fix sends to Rec two 32bit arrays and then waits for a Rec response. Rec computes a function with two input 32bit arrays and one single output 32bit array. Meanwhile, also Fix computes the same function. Rec sends the result to Fix, that compares his result with Rec's one. If the computation is correct (meaning that the communication channel works properly), all the board's leds are turned on, else only some leds are turned on. The test was successfully.
Moreover, a bus macro instances has also been inserted in place of the old, experimental one within the Caronte folders of a current DRESD thesis project. The Caronte flow worked well, and, as one output, has given the design file reported also in figure 7.

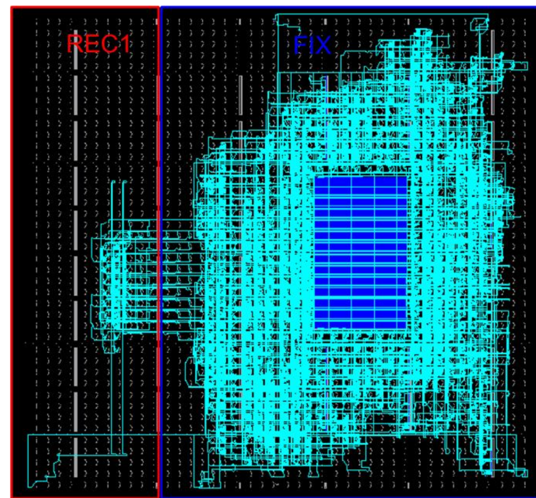

Fig. 7. A design produced by Acheronte using a generated Custom Bus Macro

4. CONCLUSIONS

In conclusion is possible to said that the Custom Bus Macro Generator effectively creates functional Bus Macros. Moreover, it allows to solve the problems individuated during the state of the art analysis.
However, a lot of works is still needed and hopefully will be done in the next DRESD projects. For example it's necessary to integrate the generator with an FPGA database in order to place correctly the bus macro line avoiding the possible obstacles (e.g. PowerPCs).

5. REFERENCES

[1] Alberto Donato, Fabrizio Ferrandi, Massimo Redaelli, Marco D. Santambrogio, Donatella Sciuto
Caronte: a methodology for the implementation of partially dynamically self-reconfiguring systems on FPGA platforms

[2] Xilinx Inc.
Two flows for partial reconfiguration: Module Based or Difference Based (September 9, 2004)

[3] OpenCores.org, Silicore
WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores (September 7, 2002)