RTL Synthesis System Using FPGA Macro Block Capabilities

Marie-Claude BERTRAND, Anne MIGNOTTE, Omar KHALIL, Gabrièle SAUCIER
Instut National Polytechnique de Grenoble / CSI
46, avenue Félix Viallet 38031 GRENOBLE cedex, FRANCE
Phone: (33) 76.57.86.87 Fax: (33) 76.50.34.21
E-mail: Marie-Claude.Bertrand@imag.fr

Abstract

For any technology targets, it is very important to make a consistent use of library macro blocks. It is obvious that the FPGA vendors designed carefully the basic blocks and that for some targets placement and routing is pretty well handled or prepared for these blocks. Therefore, and especially for FPGA targets, the resynthesis of arithmetic operators would lead to poor results compared to designs where the library macro blocks are used. It seems very important to realize a good link from a VHDL description to fully use library macro block capabilities.

In addition, most of the designers use a meet-in-the-middle design methodology. This means that they design some basic, dedicated, efficient basic blocks and they assemble or use them in a more complex design. Therefore it is very important to support a VHDL description using the functionalities of the predesigned basic blocks.

The ASYL RTL VHDL synthesis system presented in this paper provides facilities for both library macro blocks handling and predesigned designer's blocks.

Introduction

Application specific circuits use commonly arithmetic blocks which have not to be handled as glue logic during synthesis. Most of the FPGA vendors including Actel and Xilinx offer such library macro blocks [ACTa92, ACTd92, XILI91]. The difficult point is to integrate efficiently these library macro blocks in a synthesis flow starting from a high level description language as VHDL [BCM92].

This paper aims at showing how an efficient use of library macro blocks, parameterized generators and predesigned blocks is done in the ASYL RTL synthesis system [MBCF92, MBCR93] starting from a VHDL description design.

In a VHDL description, a macro block can be associated to a VHDL component instantiation, or a VHDL operator, or a VHDL function or procedure. The use of VHDL function or procedure is interesting because it allows the call to sophisticated operations performed by complex library macro blocks which cannot be called by a simple VHDL operator. Particularly, VHDL functions or procedures allow to call predesigned blocks in a meet-in-the-middle approach. Macro blocks and parameterized generators handling or resynthesis through VHDL component instantiations, or VHDL operators, or VHDL functions and procedures is explained in part 1. Illustrative examples are also.

The macro blocks and parameterized generators declaration is explained in part 2. The format is adapted to the use of library macro blocks in architectural synthesis [MiCr92].

Part 3 gives an example using the Actel and Xilinx libraries.
1. Macro blocks and parameterized generators handling or resynthesis

To declare operations in ASYL RTL VHDL, 3 types of specifications can be used: a component instantiation in a structural VHDL description, a VHDL operator (+, *, >, =, etc.) or a VHDL function or procedure in a data flow or behavioral VHDL style. These 3 types will be commented and illustrated below. The user can give synthesis directives in a directive file and the synthesis is done according to these directives. A synthesis directive is associated to one or several operations specifying or not a corresponding VHDL file line describing the operation.

Using ASYL RTL VHDL synthesis system, the user can have a full control for each component instantiation in a structural VHDL description; he can ask for:
- instantiation of a library macro block or parameterized generator or predefined block through a directive file,
- synthesis according to user defined VHDL specifications,
- instantiation of black boxes.

The user may also have a full control for each operator or function or procedure call; he can ask for:
- instantiation of a library macro block or parameterized generator or predefined block through a directive file,
- synthesis according to user defined VHDL specifications,
- resynthesis according to ASYL RTL pre-stored specifications.

1.1. Component instantiation in a structural VHDL style

1.1.1. Search and selection in a macro block library

For each component instantiation, a directive file controls the macro block library selection. The macro block or user predefined block is instantiated and connected. In this case, the macro block library has to be described according to ASYL RTL format and must be specified in the directive file.

1.1.2. Synthesis according to user defined VHDL specifications

If there is a configuration for the component, the corresponding VHDL entity-architecture description expressing the designer specification is synthesized. The synthesis is done according to speed/area trade-offs. No directive file is needed.

1.1.3. Black box instantiation

If the component is not configured in the specification, a black box is instantiated. No directive file is needed. In this case, the user has to replace in the final netlist, the name of the created black box by the name of the chosen macro block.

1.2. Use of VHDL operators

1.2.1. Search and selection in a macro block library

The system searches and selects in a library a macro block according to user specified speed, area or area/speed trade-offs requirements. This requirement is a part of the synthesis script. The macro block library has to be described according to the ASYL RTL format and must be specified in the directive file. The user can also control the selection. In this case, the name of the macro block must be specified in the directive file. Therefore, a directive file is necessary.

a) Automatic selection in a macro block library:

The operation is specified by a VHDL operator which is performed by a macro block in the library. The name of the macro block library must be specified in the directive file. An optimization criterion can be given. In the example of figure 1, for every "-", the synthesis system will search in the library BLKLIB the smallest macro block performing a subtraction (Otype SUB) and will instantiate it with a given width.
b) User controlled selection or direct binding in a macro block library:

The operation is specified by a VHDL operator corresponding to a macro block in the library. The name of the library must be specified in the directive file. The name of the selected macro block can also be specified. In the example of figure 2, for every "-", the synthesis system will search in the library BLKLIB the macro block named SUBTRACTER and will instantiate it with a given width.

1.2.2. Synthesis according to user defined VHDL specifications

Synthesis according to a user specified VHDL description is implemented by an overloading of the VHDL operator. This means that the designer knows that he wants some specific features (adder with a special carry handling, etc ...).

In the example of figure 3, the operation is specified by an overloaded VHDL operator. The user has redefined a VHDL "+" operator describing a special addition. The synthesis system will synthesize the VHDL description of the overloading function. No directive file is needed.
1.2.3. Resynthesis according to ASYL RTL prestored specifications

In ASYL RTL, Boolean equations are prestored (adders, subtracters, incrementers, ...) and synthesized according to the synthesis script. A directive file can be optionally used to specify synthesis criteria.

In the example of figure 4, the operation is specified by a VHDL operator which is not performed by any library block and which is not overloaded by the user. It is described in a predefined ASYL RTL package. It will then be resynthesized using basic cells. According to the user requirements specified in the directive file, specific Boolean equations are called and resynthesized in an optimized way on basic cells. In the example below, all the "+" operations will be resynthesized using the AREA optimization criterion.
1.3. Use of function and procedure calls

The function or procedure calls refer to a function or procedure "body". It allows to address directly library macro blocks which do not correspond to basic VHDL operators, and also to call user predesigned blocks in a meet-in-the-middle approach.

1.3.1. User controlled selection or direct binding in a macro block library

The macro block or the user predesigned block is instantiated and connected. The macro block library has to be described according to ASYL RTL format and must be specified in the directive file. Therefore, a directive file is necessary. A "forced" selection of a macro block is performed and the name of the selected macro block must be specified in the directive file. This means that the designer selects any type of block predesigned or stored in a library.

In the example of figure 5, for each function named MYFUNC, the synthesis system will search in the library MYBLKLIB the macro block named MYBLOCK and will instantiate it. The macro block named MYBLOCK can be a predesigned block which has been previously described by the user in the library MYBLKLIB or a founder specific function.

![Diagram showing VHDL function, Block Library, and Directive File](image)

Figure 5: Functions & Procedures: user controlled selection in a macro block library

1.3.2. Synthesis according to user defined VHDL specifications

The synthesis is performed according to the VHDL function or procedure body description. The designer has defined his own description and asks for synthesis. This synthesis is performed with the synthesis traditional script for area, speed or area/speed trade offs.

In the example of figure 6, the operation is specified by a VHDL function or procedure which has been described by the user. The system synthesizes the VHDL function or procedure using basic cells. No directive file is needed is this case.
1.3.3. Resynthesis according to ASYL RTL prestored specifications

The resynthesis can be performed according to VHDL prestored descriptions of the function or procedure body (prestored package). When the functions or procedures are well known functions or procedures (Square Root Extractor for instance), a corresponding prestored body description is available. No directive file is needed in this case.

The operation is specified by a VHDL prestored function or procedure. The resynthesis is done according to the prestored equations. In the example of figure 7, all the "SQRT" functions will be resynthesized.

2. Library declaration

The library can contain fixed blocks and parameterized generators. The description of these two types of blocks is explained below. VHDL models have also to be associated with the block library for simulation purpose. The VHDL models of the operations are given in a VHDL package. Note that the VHDL models of the operations are not synthesized. These models are only used for simulation [BCM92].
2.1. Fixed blocks

Fixed blocks are blocks whose characteristics are entirely defined. The description of a fixed block is introduced by the key word "resource". An example of a fixed block description is given in figure 8: it is a 16 bits adder/subtractor. Fixed blocks are characterized by their I/O ports, the operations they implement and physical characteristics (area and timing). The description will be illustrated on the adder-subtractor of figure 8.

The name of the block is given after the name "resource". Each block has a type (ResType field). It can be an operator (OPE), a register (MEM_D), a multiplexer (MUX), ... The area is specified to be constant.

The ports are listed, each port having a name, a type (Port Type field) and a width. The width is a constant.

The operations performed by the block are described in the operation list (OpList field). A block (for example ADDSUB16 in figure 8) is able to perform one or several specific operations (e.g. ADDC, SUBC). These operations will be associated with the VHDL functions or VHDL operators used in the VHDL specification. Each operation has a name, a constant delay and is associated with a type (the OpType field) which gives the semantics of the operation. This OpType allows to establish the link with VHDL operators. For each operation, if necessary, a command port list is given indicating the values needed to apply on the command port to perform the operation. In the example of figure 8, the ADD_SUB command port must have the value 1 to perform the ADDC operation. A list of data port is also specified. The commutativity of input ports can be given. This information is used during the synthesis process to optimize the design. Here, a default value can also be given to an input port; this default value will be used if the port is unconnected. The declaration is similar to the EDIF format.

(Resource ADDSUB16
   (ResType OPE)
   (Area 120)
   (PortList
      (PortDesc A (PortType INPUT) (PortWidth 16))
      (PortDesc B (PortType INPUT) (PortWidth 16))
      (PortDesc ADD_SUB (PortType COMMAND) (PortWidth 1))
      (PortDesc S (PortType OUTPUT) (PortWidth 16))
      (PortDesc COUT (PortType OUTPUT) (PortWidth 1))
   )
   (OpList
      (OpDesc ADDC # --> this operation name is the name of
         VHDL specification function #
         (OpType ADD) # --> link with the VHDL operator "+" #
         (Delay 15)
         (OpPortCommandList
            (OpPortCommandDesc ADD_SUB (PortDefaultValue 1))
         )
         (OpPortDataList
            (PortDataDesc A (PortCommutList B))
            (PortDataDesc B (PortCommutList A))
            (PortDataDesc S)
            (PortDataDesc COUT)
         )
      )
      (OpDesc SUBC (OpType SUB)
         (Delay 15,6)
         (OpPortCommandList
            (OpPortCommandDesc ADD_SUB (PortDefaultValue 0))
         )
         (OpPortDataList
            (PortDataDesc A)
            (PortDataDesc B)
            (PortDataDesc S)
            (PortDataDesc COUT)
         )
      )
   )
)

Figure 8: Description of a 16 bit ADDER-SUBTRACTER block
The figure 9 gives the VHDL description of the function "ADDC" which is associated with the operation "ADDC" performed by the adder-subtractor described before in figure 8. This function must be described in the ASYLRTL package.

```vhdl
package ASYLRTL is
...
function ADDC (A,B : BIT_VECTOR) return BIT_VECTOR;
...
end ASYLRTL;

package body ASYLRTL is
...
function ADDC (A,B : BIT_VECTOR) return BIT_VECTOR is
variable ...
begin
...
  SUM(I) := A(I) xor B(I) xor CARRY(I);
...
end ADDC;
...
end ASYLRTL;
```

Figure 9: ASYLRTL package

### 2.2. Parameterized blocks

Parameterized blocks are blocks which are not completely defined. Some informations are not known (as for example the number of bits of the operands of the block) and must be fixed by the synthesis system during the synthesis process. These informations are extracted from the input VHDL specification or from the user requirements. The description of a parameterized generator is introduced by the key word "generic". An example of parameterized adder-subtractor is given in figure 10.

As for fixed blocks, parameterized generators are characterized by their I/O ports, the operations they implement and physical characteristics (area and timing). In addition, a parameter list must also be given. The parameters are declared in the parameter list (ParamList field). Each parameter has a name, a type and a range or a set of values. In the example of figure 10, the parameter is the number of bits of the adder-subtractor which name is Nbits. This parameter can take a value from 1 to 32. If necessary, other parameters can be described as the implementation style of the block, ... A default value for a parameter can be specified.

If the parameter used is the number of bits, the length of the I/O ports must be given according to this parameter. As shown in figure 10, the length of the input ports A and B and the output port S is on Nbits bits. Note also that physical informations (area and delay) can be given by a formula or a function call which allows to know the value according to the parameters value. In figure 10, an example of formula is given for the area and an example of function call is given for the delay.
3. Illustrative examples

To illustrate this approach one example has been synthesized at the Register Transfer Level on Actel ACT1 library [ACTa92, ACTd92] and on Xilinx XC4000 library [XIL91]. It is a serial multiplier described in [MBCF92]. It contains 8 states, an addition and a shifting operation.

3.1. Actel ACT1 library

ACT1 library contains 8 and 16 bits adders and 8 bits registers. The serial multiplier was described using 8 bits and 16 bits operands. Therefore, the 8 bits serial multiplier will be synthesized using ACT1 macro blocks and the 16 bits serial multiplier will be synthesized resynthesizing the 16 bits registers. This will illustrate the necessity to resynthesized some blocks when they do not exist in the library.

3.2. Xilinx XC4000 library

Xilinx XC4000 library contains 8 bits registers and adders. The serial multiplier was described using 8 bits operands. Therefore, all the RTL operations of the initial specification can be mapped on XC4000 macro blocks.
3.3. Results

The serial multiplier was synthesized using either the macro block library or only the basic cell library to compare the results between blocks (illustrating the meet-in-the-middle design approach) and resynthesis (illustrating the top-down design approach). The example has been synthesized using either area main criterion (Area criterion columns) or speed main criterion (Speed criterion columns). The area is given by the number of FPGA modules to be used. On ACT1, the critical path of the design is estimated. On XC4000 the critical path is given after place and route.

<table>
<thead>
<tr>
<th></th>
<th>Area Criterion</th>
<th></th>
<th>Speed criterion</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area (# modules)</td>
<td>Critical Path (ns)</td>
<td>Area (# modules)</td>
<td>Critical Path (ns)</td>
</tr>
<tr>
<td>8 bits serial multiplier</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blocks</td>
<td>198</td>
<td>41</td>
<td>198</td>
<td>41</td>
</tr>
<tr>
<td>Resynthesis</td>
<td>143</td>
<td>112</td>
<td>222</td>
<td>90</td>
</tr>
<tr>
<td>16 bits serial multiplier</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blocks</td>
<td>256</td>
<td>65</td>
<td>310</td>
<td>48</td>
</tr>
<tr>
<td>Resynthesis</td>
<td>219</td>
<td>183</td>
<td>811</td>
<td>139</td>
</tr>
</tbody>
</table>

Table 1: Serial multiplier example using ACT1 library

Concerning speed main criterion, results with resynthesis are really worst than the ones using predefined blocks of the library. Therefore, efficient use of library blocks is really indispensable.

Concerning the area main criterion, the results resynthesizing operators are better (about 25%) but they increase dramatically (more than 180%) the critical path.

<table>
<thead>
<tr>
<th></th>
<th>Area Criterion</th>
<th></th>
<th>Speed criterion</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area (# CLBs)</td>
<td>Critical Path (ns)</td>
<td>Area (# CLBs)</td>
<td>Critical Path (ns)</td>
</tr>
<tr>
<td>8 bits serial multiplier</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blocks</td>
<td>32</td>
<td>60</td>
<td>32</td>
<td>62</td>
</tr>
<tr>
<td>Resynthesis</td>
<td>43</td>
<td>64</td>
<td>54</td>
<td>79</td>
</tr>
</tbody>
</table>

Table 2: Serial multiplier example using XC4000 library

Concerning XC4000 library, when using area main criterion, the results using the macro blocks are better in terms of number of modules (about 30%) than the ones using resynthesis. Concerning the speed main criterion, the resulting critical path using the macro blocks is better (about 25%) than the one using resynthesis.
Conclusion

This paper has presented a VHDL synthesis tool that allows to use efficiently library and predesigned blocks. The library manager fits the designer criterion (area or speed) and the designer explicit operator selection. Moreover, it builds non-existing macro blocks by resynthesis on basic cells. This approach was illustrating on a simple example using Actel ACT1 and Xilinx XC4000 FPGA libraries.

References