BLAST: classes and xml files for block models and implementations

Stéphane Domas
Laboratoire d'Informatique de l'Université de Franche-Comté,
BP 527,
90016 Belfort CEDEX, France

June 29, 2016

1 Models XML description

2 Models class hierarchy

bla bla ...

2.1 parameters classes

There are four classes of parameters for a block:

- user,
- generic,
- port,
- wishbone.

Four attributes are common to all:

- `owner`: a pointer to the block instance (reference/functional) that "owns" this parameter,
- `name`: the name of the parameter (NB: cannot be changed by the user of BLAST),
- `type`: the type of the value,
- `value`: the default value of the parameter, given as a `QString` but stored as a `QVariant`.
- wishbone.

`type` value (int) must be set with one of those in `ParamType` enum (cf. `BlockParameter.h`):

```
enum ParamType { Undefined = -1, Expression = 1, Character,
                 String, Bit, BitVector, Boolean,
                 Integer, Natural, Positive, Real, Time};
```

Except the two first, these values correspond to predefined types in VHDL. `Expression` correspond to an arithmetic expression and must be used only for port and wishbone parameters. Thus, its syntax will be presented in the associated sections (2.1.3 and **??**).

Whatever the case, parameters will be used during VHDL generation but at different phases, depending on the class.

2.1.1 user parameters

User parameters have a type equals to `String`. A default value must be given at construction but a `userValue` can be defined via setters.

A user parameter is only used when generating the VHDL code of the architecture section of a block (cf. section 3). Each time the generator encounters an escape sequence: `@val{user_parameter_name}`, it replaces it with `userValue` if defined, or with the default value.

CAUTION: No validity check are done by BLAST on default and user value strings. Thus, they can be anything and can lead to incorrect VHDL.

2.1.2 generic parameters

Generic parameters have a type equals to any predefined VHDL type (i.e. all defined above except `Undefined` and `Expression`). A default value must be given at construction but a `userValue` can be defined via setters.

A generic parameter is used during the generation of:

- the entity section,
- the component section when the owner block is used within another block,
- the generic map of when instanciating owner block is used within another block,
- the architecture section.

In the two first cases, it leads to lines like
`d_width :  integer := 16;`, using the `name`, `type` and default value.
In the third case, it leads to lines like
`d_width => 10,`, using the `name` and `userValue`, or default value if not defined.
`d_width => d_width,`, using only the `name`. This case occurs when the owner is instanciated within a block that has a generic parameter with the same name.
In the last case, each time the generator encounters an escape sequence: `@val{generic_parameter_na`
it replaces it with `userValue` if defined, or with the default value.

IMPORTANT: a block that defines wishbone parameters will be generated with 2 generic parameters with predefined names: `wb_data_width` and `wb_addr_width`. They correspond to the width of the address and data buses of the wishbone and are used during the generation of the controller of the block (cf. 2.1.4).

2.1.3 port parameters

A port parameter can be used to obtain a value associated to an interface of the block. Indeed, it is possible to create several instances of the same interface, when its multiplicity given in the reference model is greater than 1. For example, it is used for block that may have a variable number of inputs/outputs, like a multiplexer. Instead fo creating a bunch of multiplexers with 2, 3, 4, ... data inputs, it is possible to generate one for any amount. In BLAST, this amount is given by the number of instances of the data input the user has created. It implies that the selector input has a variable range and thus needs a variable number of bits to be expressed. If N is the number of instances of the data input, then the selector size is $log_2(N)$. A port parameter is used to express such a value.

A port parameters have a supplementary attribute `ifaceName` that must correspond to an existing interface of the block. `type` is equal to `Expression` and `value` contains this expression, using variables with a predefined name `$if_nb` and `$if_width` that will be respectively replace during VHDL generation by the number of instances of `ifaceName`, and its width.

A port parameter is used during the generation of:

- the entity section,
- the component section when the owner block is used within another block,
- the architecture section.

In every case, each time the generator encounters an escape sequence: `@val{port_parameter_name}`, it replaces it with the computed value of the parameter using the expression and the interface name.

2.1.4 wishbone parameters

A wishbone parameter corresponds to a register that can be read/write via the wishbone bus. Nevertheless, the GUI allows the user to disable the wishbone access and replace it by a fixed value or a port that is assign to the register.

Since a register has a width, `type` gives its type. Valid types are:

- `boolean` if the register is an `std_logic`,
- `natural` if the register has a fixed width and is a `std_logic_vector`,
- `expression` if the register has a variable width and is a `std_logic_vector`,

The width is given in a supplementary attribute `width`. Note that:

- if the type is boolean, width should be equal to 1 but in fact is not used.
- if the type is natural and width is equal to 1, it leads to `std_logic_vector(0 downto 0)`, which is sometimes usefull for memory accesses.

- if the type is an expression, width must contains an expression using only +,-,*, numbers and generic parameter references (i.e. parameter name prepend with a).$Nocheckisdonethus, themodelm$

  In the second case, the expression may use predefined names `wb_data_width` and `wb_addr_width`.

  Whatever the case, during VHDL generation, the final width will be compared to the wishbone data bus width. If the bus width is lesser than the register width, then several wishbone accesses are needed to read/write the register. For example, if wishbone width is 16 and a register has a width of 20, we need to define two addresses, one to access to bits 0 to 15, and one for bits 16 to 19. The total number of needed addresses gives the minimal width of the address bus (i.e. log2(nb addr)).

  The value is a initialization value. If not provided, it is 0 by default.

  Three other attributes are declared:

  - `wbAccess`: indicates if the register is written or read by the wishbone bus. Thus, if it is written, the register is an input of the block and if it is read, the block provides its value.
  - `wbValue`: it may be a natural, a boolean, or the word data. In the two first cases, the given value is affected to the register as soon as the register is accessed in writing. In the third case, the register is affected with the value that is provided on the wishbone data bus.
  - `wbDuration` : indicates if the affectation is permanent or just a trigger. In the second case, the value is set for just one clock cycle and then reset to the initialization value given by the `value` attribute.

  A wishbone parameter is used during the generation of:

  - the controller of the block (NB: this generation is done at the block level),
  - the entity section, because register values are in fact read/write by the block via input/output ports,
  - the component section when the owner block is used within another block.

3 Implementations XML description

4 Implementations class hierarchy