In Proceedings of Design of Integrated Circuits and Systems Conference (DCIS), November 1996

Register Transfer Level VHDL Block Generation

Paulo Flores *Horácio Neto*

INESC/IST
Instituto de Engenharia de Sistemas e Computadores
Rua Alves Redol, 9 - 1100 Lisboa - Portugal
Tel: +351.1.3100000 Fax: +351.1.525843
E-mail: pff@inesc.pt

Abstract

System-level design usually begins with a high level description that is refined to get a register transfer (RT) level netlist of the circuit. An important task of this process is the component selection, that can be executed automatically by high level synthesis tools or manually by the designer. Typically this selection is based on a limited set of components available on the selected technology.

In this paper, we present a set of RT-level VHDL generators that can be used as standard component library. Each block generator is a technology independent VHDL description that can be customized, at instantiation time, to fit a particular application.

For some real circuits descriptions we compare, in terms of area of the synthesized circuit, the use of VHDL block generators versus dedicated RT-level descriptions. We conclude that the area penalty which can occur in some cases is well compensated by the advantages of having a more compact description using a set of well defined and previously tested component generators.

1 Introduction

The continuous development of VLSI technology has made available increasing gate count and performance. To deal with the increasing amount of information needed to manage the complexity of actual integrated circuits designs, new design methodologies have been developed over the last few years. Basically, the designer can start the circuit specification at an higher level of abstraction, without the need to spend critical design time with the low level details of implementation.

Standard hardware description languages, such as the VHDL language, offer a machine, vendor and technology independent method of describing, documenting and simulating complex digital integrated circuits and systems. VHDL can be used to describe circuits at different levels of abstraction: logic, RT-level and behavioral.

Today's commercial synthesis tools can produce a gate level net-list, mapped to a specific technology library, from the VHDL specification. However, some restrictions are imposed in the description with respect to the language sub-set supported and to the accepted style of description. Currently most of the synthesis tools require a description style that only supports logic and RT-level descriptions.

However, the flexibility of VHDL enables the description of generic RT-level block generators that can be parameterized and programmed according to the needs of a specific application. These generators can be simulated and synthesized with different customizations, therefore providing an efficient mechanism to high-level specification.

In the following section the advantages of using efficient RT-level block generators, in the existing synthesis design methodologies, are presented. In section 3 the complete library of developed generators is described. Application examples of the use of VHDL block generators and results obtained with real circuits are presented in section 4. Finally, in the last section

some conclusions are presented.

2 Using RTL Generators with Synthesis Methodologies

The system level design process typically starts with an high level design specification. This specification is then refined down to the level where each hardware system component is described as a block diagram or abstract netlist of RT-level components [1].

The refinement process can be done "automatically", using an high level synthesis tool, or manually by the designer. Both methodologies can profit by the use of RT-level block generators.

Using a high level synthesis tool, the RT-level netlist is obtained from the behavioral description of the circuit, through a set of tasks that deal with:

- selection of the components from a library
- scheduling all the operations in the description into control steps
- binding the operations in the description to the selected components
- design optimization

The library from which the components are selected plays an important role in the context of synthesis. Typically, RT-level libraries offer a limited set of common components (such as adders, registers, multiplexers, etc) and/or specific components from a given technology. These libraries may be very well characterized in terms of area and delay, but they miss the flexibility of being tailored individually to each instantiation of a component.

The use of generic RT-level block generators practically defines an "unlimited" set of components due to the considerable amount of parameterizations and programming options allowed. Each component results from the customization of a block generator to the specific requirements of each applications. This customization can involve the parameterization and programming of the blocks generators and/or the attachment of some attributes that will be used during logic optimization.

The use of standard VHDL to specify generic RT-level block generators preserves technology, EDA vendor and platform independence, and reduces the cost of developing and maintaining such libraries of components. Technology independence also allows the use of block generators as a "standard" library, which facilitates unambiguous documentation, communication and design re-use.

An important requirement for system-level design is the capability to predict technology-specific design characteristics. Using existing VHDL simulators, RT-level block generators can be used to rapidly explore architectural alternatives in terms of overall system functional performance. Combined with existing logic synthesis tools, an accurate exploration of architectural alternatives, in terms of area, speed and power, can be rapidly performed by synthesizing to a selected technology.

If the design methodology is not based on high-level synthesis tools then the designer has to manually obtain a RT-level specification for the system. In this case the RT-level block generators acts as a very efficient generic library whose components can be instantiated by the designer and adjusted to fit the specific application.

The use of this library increases the designer productivity by reducing the amount of work to get a specification and time dedicated to verify the design, given that each component has been previously well verified and documented.

3 IC-Blocks Library

The parameterized/programmed generic RT block generators set presented has been named *IC-Blocks Library* .

This library was developed considering its use in an HDL-based synthesis methodology as described in the previous section. A technology independent description of each generator was obtained using a synthesized VHDL subset and an appropriate description style.

The ports of all generators are declared as `std_logic` or `std_logic_vector` types, as defined by the IEEE package `std_logic_1164`. In some cases, the VHDL description of the generators takes advantage of the packages

`std_logic_arith` and `std_logic_unsigned`. These packages were developed by SYNOPSYS, INC. and can be freely used and distributed. The declaration and body of these packages are available, so they can be exported to any VHDL system.

Tailoring a block generator so that the synthesis output is adjusted to a particular problem is achieved by two methods:

Parameterization - Most of the generators can be parameterized through the use of the *generic* VHDL construct. For example, in most of the generators, the input/output buses width can be parameterized through the value of the `WIDTH` generic.

Programming - Several generators have input ports that can be used to control the execution of some operations. These ports can be used as normal control ports or as programming ports. In the latter case, they are connected explicitly to a fixed logic value, enabling or disabling definitely some operations.

Note that, while the parameterization defines the number of resources to allocate for the specific component, programming "will force" an optimization of the generated component during the logic synthesis step (see examples in section 4)

The generators developed, so far, for the *IC-Blocks Library* are listed in table 1. The 29 generators presented can be divided in three classes:

- Combinational generators - This class of generators will only yield combinational circuits and can be sub-divided in:
  - arithmetic generators - ADD, ADSB, ALU, EQ, GLE, SUB
  - code converts generators - BCD_BIN, BIN_BCD, BIN_SEGD, BIN_SEGH, COD, DECOD, ENC
  - multiplexers/demultiplexers generators - DEMUX, DEMUXW, MUX, MUXW
- Sequential generators - This class yields circuits with memory elements (latches or flip-flops) and can be sub-divided in:
  - data storage generators - FIFO, FIFOD, FIFO_S, MEM, MEMD, REGFF, REGL
  - application generators - COUNTER, SR
- Special purpose generators - This class includes generators developed to fit more particular solutions which usually have low parameterization and programming capabilities - DATA_BUS, HDB3_NRZ, NRZ_HDB3.

IC BLOCK	FUNCTIONALITY
ADD	Binary adder with carry
ADSB	Adder-subtractor
ALU	Arithmetic and Logic Unit
BCD_BIN	BCD to binary converter
BIN_BCD	Binary to BCD converter
BIN_SEGD	Binary to decimal segment converter.
BIN_SEGH	Binary to hexadecimal segment converter.
COD	Coder to binary.
COUNTER	Generic counter.
DATA_BUS	Bus interface.
DECOD	Decoder
DEMUX	Demultiplexer for single lines.
DEMUXW	Demultiplexer for bus lines.
ENC	Priority encoder
EQ	Equality comparator
FIFO	Dual port, single clocked FIFO
FIFOD	Dual port, dual clocked FIFO
FIFO_S	Synchronous FIFO for asynchronous bus.
GLE	Comparator with greater than, less than and equal to output conditions
HDB3_NRZ	High density bipolar 3 (HDB3) to non-return to zero (NRZ) encoder
MEM	Dual-port static RAM memory with single output data bus
MEMD	Dual-port static RAM memory with double output data bus
MUX	Multiplexer for single lines.
MUXW	Multiplexer for bus lines.
NRZ_HDB3	Non-return to zero (NRZ) to high density bipolar 3 (HDB3) encoder
REGFF	Register using D-type Flip-Flops elements.
REGL	Register using Latch elements.
SR	Shift Register.
SUB	Subtractor

Table 1: Generators in the *IC-Blocks Library*

All blocks generators are documented in the *IC-Blocks Reference Manual* [2], which includes, for each generator a data-sheet containing the

following information:

- Name of the generator.
- Features enumerating the main functionality of the generator.
- Detailed description of the functionality and of the configuration possibilities of the generator. Two tables are included, one for the ports description, and other for the description of the generics and their default values.
- VHDL description of the generator.
- Default synthesis results for a general technology library.
- Application examples with synthesis results to quickly help the designer understand the generator usage and potentials.

4 Application Examples

To exemplify the use of RT-level block generators the REGFF block is described below. This generator is a "simple" register whose memory elements are D type flip-flops. Figure 1 presents the symbol of the REGFF block.

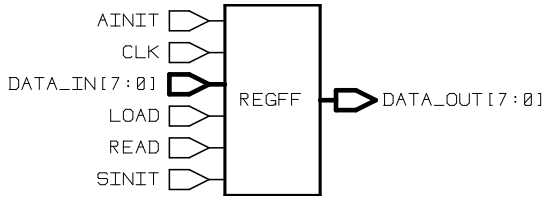


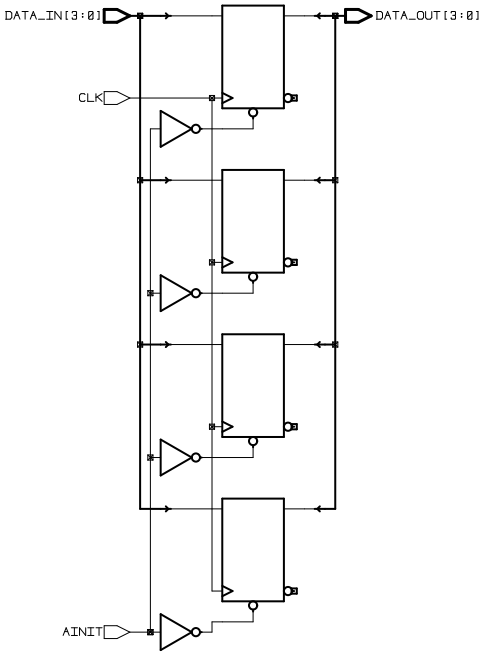
Figure 1: REGFF symbol

The ports of this generator are described in table 2. As expected, there is a clock line (CLK) and buses to write data and read it from the register (DATA_IN and DATA_OUT, respectively). The LOAD and READ signals control the capability to execute a synchronous parallel load and activation of the output bus (DATA_OUT) with the register value, respectively. The initialization of the register can be implemented asynchronously, synchronously, both or none, using the right combination of AINIT and SINIT signals.

The generics of REGFF are described in table 3. This generator can be parameterized regarding to: the width of the register and data buses (WIDTH); the asynchronous and synchronous initialization values (AINITIAL and SINITIAL, respectively); and the values of the register output data bus, when the READ signal is not active (NO_READ).

Figures 2, 3 and 4 present synthesized circuits[1], originated by the REGFF block generator using different parameterization and programming values.

Figure 2 represents a simple 4-bits register with asynchronous initialization to "0000". This circuit results from parameterizing the generator REGFF to a width of 4-bits (generic WIDTH=4) and connecting the "programming ports" LOAD and READ to logic '1', and SINIT to logic '0'. The VHDL code to generate this register is a simple instantiation of the library component REGFF customized as described above.




```
library IEEE;
library ICBLOCKS;

use IEEE.std_logic_1164.all;
use ICBLOCKS.GENERIC_COMPONENTS.all;

entity ex1_regff is
    port(CLK, AINIT: in std_logic;
         DATA_IN : in std_logic_vector(3 downto 0);
         DATA_OUT: out std_logic_vector(3 downto 0));
end ex1_regff;

architecture example of ex1_regff is
    signal ZERO, ONE: std_logic;
begin
    ZERO <= '0';
    ONE <= '1';

    U1: REGFF
        generic map(WIDTH => 4)
        port map(AINIT => AINIT, CLK => CLK,
                 DATA_IN => DATA_IN,
                 LOAD => ONE, READ => ONE,
                 SINIT => ZERO,
                 DATA_OUT => DATA_OUT);
end example;
```

Figure 2: A 4-bits register.

In figure 3 the REGFF generator was config-

[1] Synthesized circuits for a general technology library.

Port Name	Type	Description
CLK	Input	Positive edge trigger clock.
LOAD	Input	Parallel load of the register with DATA_IN value.
READ	Input	DATA_OUT takes the value of the registers when READ = 1.
AINIT	Input	Asynchronously register initialization with AINITIAL generic value.
SINIT	Input	Synchronously register initialization with SINITIAL generic value.
DATA_IN	Input	Register input data bus.
DATA_OUT	Output	Register output data bus.

Table 2: REGFF ports

Generic	Default	Description
WIDTH	8	Register and buses dimension.
AINITIAL	0	Asynchronous initialization value.
SINITIAL	0	Asynchronous initialization value.
NO_READ	2	When READ = 0, DATA_OUT assumes a value according to the following values of NO_READ: 0 – DATA_OUT is filled with zeros; 1 – DATA_OUT is filled with ones; 2 – DATA_OUT goes to high impedance; 3 – don't cares.

Table 3: REGFF Generics

ured as in the previous example, except that the width was parameterized for 2-bits and the LOAD programming port was left as a input signal. Note that leaving the LOAD port as a signal implies extra logic (a multiplexer) before each flip-flop. In the previous example this logic was removed (optimized) by the synthesis tools, because the LOAD control signal had a constant value.

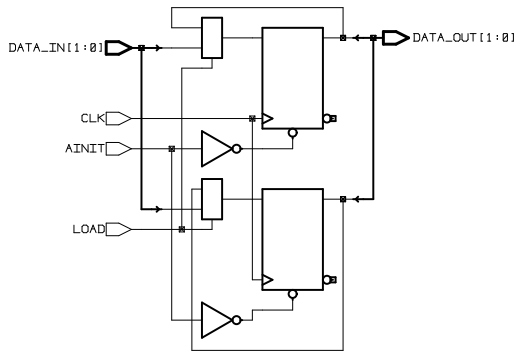


Figure 3: A 2-bit register with parallel load.

The register of figure 4 was synthesized using the same generator, REGFF. The READ port was left as a control signal which combined with the default value of the NO_READ generic, will result in a register with three-state outputs. The generic AINITIAL was parameterized with the value 1, so that an asynchronous initialization of the register will yield the value "01".

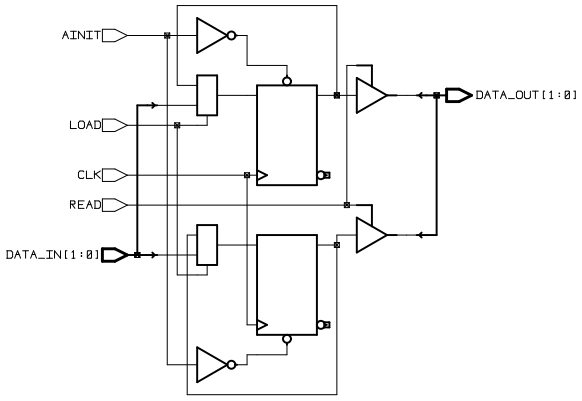


Figure 4: A 2-bit register with three-state output.

Note that the use of the VHDL block generators does not introduce any penalty, in terms of area, speed or power. For the same functionality, the synthesized circuits from the generators blocks, have a gate count and structure equivalent to the circuits synthesized from typical dedicated VHDL code. Of course, the results are highly dependent on the "quality" of the logic synthesis tool used.

Results got from using the VHDL block generators at our CAD environment are presented

in table 4. This table compares the area of three circuits, described in VHDL, at a RT-level, with and without block generators.

Circuit	With Gen. (Area)	Without Gen. (Area)
Up-Counter	183	183
Register-Bank	1198	1046
USART	8575	8492

Table 4: Results from using generators

The first circuit (Up-Counter) is a simple 4-bit up-counter with parallel load, which can be implemented directly using the COUNTER generator configured correctly. The second circuit (Register-Bank) is a set of four different types registers with a common output data bus. The third circuit (USART) is an Universal Synchronous Asynchronous Receiver Transmitter used in a Smart Card Interface developed internally.

These examples show that the area of the synthesized circuit using VHDL block generators can be slightly higher than the one synthesized from a custom-made description. The small area penalty results from the fact that the generator makes the optimization phase of the synthesis tool harder, thus, resulting in a less optimized circuit.

The "quality" of the synthesis tool used may influence these results, but with the current development of synthesis technology, the area differences will tend to disappear.

5 Conclusions

The fast development of VLSI technology has lead to the evolution of the design methodologies, pushing the capture of design specifications to higher levels of abstraction.

In this paper we presented a library of RT-level generators (*IC-Blocks Library*) suitable for use in high level design methodologies. Each library generator is a VHDL description that can be instantiated automatically by a tool, during the component selection phase of high level synthesis, or manually by the designer. Due to the capability to configure each instantiation of a generator, by parameterizing generics and/or programming port values, it is possible to tailor it for a particular application. This kind of flexibility can not be achieved by common RT-level libraries in which the components are definitely defined and bound to a particular technology.

The increasing of the designer productivity, by significantly reducing the time to get a functional correct RT-level description is a major advantage of using block generators.

A small area penalty can be incurred by using the block generators instead of using customized code for the RT-level description. However, this penalty is compensated by the benefits of using well verified and tested components, in terms of reduced design and debug time of a circuit.

References

[1] Daniel Gajski, Loganath Ramachandrn, Peter Fung, Frank Vahid, and Sanjiv Narayan, "Towards achiving an 100-hour design cylcle: A test case", Technical report #94-08, Dept. of Information and Computer Science, University of California, Irvine, Februay 1994.

[2] Leonel Simões and Paulo Flores, *IC-Blocks Reference Manual*, INESC, Version 2.0 - April 1995.

[3] Scott Powell and Thomas Cesear, "Rapid design and exploration of signal processing systems using a VHDL generator based paradigm", *VHDL Times*, vol. 4, no. 3, 1995.

[4] M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, and A. Sangiovanni-Vincentelli, "A formal specification model for hardware/software codesign", in *International Workshop on Hardware-Software Codesign*, October 1993.

[5] Pradip Jha, Nikil Dutt, and Daniel Gajski, "An evaluative study of RT component libraries", Technical report #93-11, Dept. of Information and Computer Science, University of California, Irvine, March 1993.

[6] Synopsys, Inc., *VHDL Compiler Reference Manual*, November 1992, Version 3.0.

[7] Institute of Electrical and Electronics Engineers, *IEEE Standard VHDL Language Reference Manual.*, March 1988, IEEE Std 1076-1987.