Register Transfer Level VHDL Models without Clocks

Matthias Mutz (MMutz@sican–bs.de)
SICAN Braunschweig GmbH, Digital IC Center
D–38106 Braunschweig, GERMANY

Abstract

Several hardware compilers on the market convert from so-called RT level VHDL subsets to logic level descriptions. Such models still need clock signals and the notion of physical time in order to be executable. In a stage of a top-down design starting from the algorithmic level, register transfers are considered, where the timing is not controlled by clock signals and where physical time is not yet relevant. We propose an executable VHDL subset for such register transfer models.

1 Introduction

The paper introduces a VHDL subset for modeling register transfer behavior at a more abstract level than usual in today's synthesis tools. First of all, we want to motivate why VHDL can play a central role for synthesis at several levels of abstraction. VHDL has been designed to support hardware models at all levels of abstraction. We are aiming at a systematic way of performing an incremental design by adding information to a hardware description. VHDL is also suited to model hardware at more abstract timing levels. In addition to *physical time*, VHDL provides the concept of *delta time* counting successive simulation cycles which do not increase physical time (so called *delta cycles*). We explicitly use delta time and the corresponding simulation semantics to model hardware timing at higher levels. Furthermore, we need user-defined resolution functions, which are used to combine values assigned to a signal at the same time. Other languages only support implicit resolution functions for buses. At higher levels, we also have multiple sources for other signals, e.g. module and register ports.

In [1] it is already mentioned that there should be a systematic but general way based on VHDL subsets to deal with VHDL in order to create efficient link with hardware synthesis results. We use our own VHDL subset for modeling the register transfer level, because known subsets only cover models using clock and control signals. We model register transfer level timing based on *control steps*. Clock and control signals with physical timing implementing the timing based on control steps are introduced in a succeeding synthesis step. Therefore, we use delta time instead of physical time.

Subset selections could be based on designer's guides for using VHDL [2, 3]. In such guides, behavior at the most abstract level is based on clocking schemes with definite clock periods, which models behavior at lower levels than the ones we are aiming at. Such guidelines lead to VHDL subsets for synthesis tools. For example, a subset for logic synthesis is given in [4]. Also subsets for higher levels are proposed, e.g. [5] for high level synthesis, but still involving explicit clocks. An European working group [6] is going to define standards for VHDL subsets for synthesis. There are efforts in defining a subset in order to constitute a standard subset of VHDL for synthesis application, which will allow description portability between tools as well as design reusability [7].

If we look for formal verification, other aspects become more important. For example, [8] introduces a verification oriented subset of VHDL. It has already been shown that it makes sense to deal with more abstract levels of behavior than supported by current subsets. [9] describes a methodology for verifying VHDL descriptions of processors using a computer algebra simplification tool relating instruction level and register transfer level. The most important similarity to our approach is, that VHDL descriptions don't use explicit delay times and explicit clocks. Instead, abstract state transitions are inferred from successive VHDL simulation cycles not increasing simulation time.

Most efforts concentrate on clocked circuits. We want to go towards standards for higher timing levels. Section 2 introduces the VHDL subset for the more abstract register transfer level. It first explains the basic model. Then basic component descriptions are given. Finally, the construction of register transfer models is described. Section 3 gives an example for the application of our subset. In section 4, we summarize

and conclude.

2 Register Transfer Models

2.1 The basics

We here only discuss the base model, which can be easily extended for different register transfer timings and different data types to be performed. The model uses structural components, the so-called *resources*, and covers the usage of the resources at a behavioral level by so-called *transfers*. The model supports a specific high-level architecture not based on clocks, which can be mapped to several low-level architectures defining how transfers are implemented based on clocks.

A register transfer model is defined by a set of registers, a set of modules performing arithmetical and logical operations, a set of buses used for transfers of values between modules and registers, and the timing of transfers. Registers and modules are called functional units. Register transfers are embedded in a control step scheme. 1 illustrates a concrete register transfer of an example.

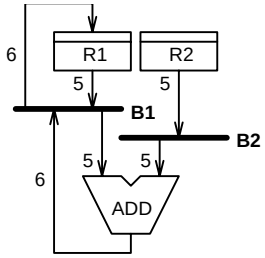

Figure 1: a concrete register transfer

Fig. 1 only shows the resources affected by the register transfer denoted by the tuple

(R1,B1,R2,B2,5,ADD,6,B1,R1)

The tuple is interpreted in the dollowing way. In control step 5, the value at the output port of register R1 is transferred to the left input port of the module ADD via bus B1. Also in control step 5, the value at the output port of register R2 is transferred to the right input port of the module ADD via bus B2. In control step 6, the value of the output port of the module ADD is transferred to the input port of register R1 via bus B1.

It is to mention, that nothing is said about the implementation of transfers and the implementation of control step timing. It is also just said, that transfer occur via buses, it is not said how ports are connected to buses. There still are several degrees of freedom for further synthesis steps. This is a typical situation in a top-down design, where resources are allocated and register transfers are scheduled to implement operations. At this stage abstract level of timing resource conflicts can be detected. This is *better* than usual RT models in the sense that the more abstract model can be mapped to several usual RT models. It is *worse* than usual RT models in the sense that it is not directly synthesizable, i.e. an additional transformation is needed to obtain a usual RT description that can be performed by current commercial synthesis tools.

The transfers given by 9-tuples define when values are transferred between buses and ports of the functional units. The functional units combine values at the input ports available in a control step and provide the resulting value at the output port at the same or a later control step. The scheduling task is to determine the register transfers and to properly embed them into the control step scheme observing the timing of the functional units.

2.2 Timing

A control step is partitioned into six successive phases, which occur in a cyclic manner as shown in fig. 2.

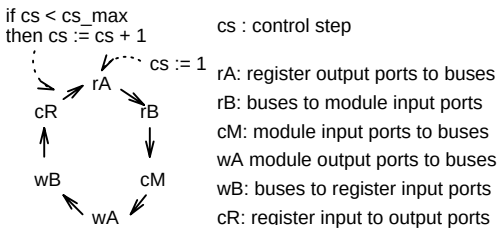

Figure 2: timing of register transfers

Control step phases are introduced as VHDL enumeration type:

```
type Phase is (rA,rB,cM,wA,wB,cR);
```

and it is represented by a signal PH: Phase. The control step number is represented by a natural number signal CS: Natural. The cyclic occurrence of phases with increasing control step number is driven by a controller process:

```
entity CONTROLLER is
  generic (CS_MAX: Natural);
  port (CS: inout Natural := 0;
```

```
    PH: inout Phase := Phase'High); -- Phase'High = cR
end CONTROLLER;

architecture transfer of CONTROLLER is
begin
  process (PH)
  begin
    if (PH = Phase'High) then
      if (CS < CS_MAX) then
        CS <= CS+1;
        PH <= Phase'Low; -- Phase'Low = rA
      end if;
    else
      PH <= Phase'Succ(PH); -- Phase'Succ(cM) = wA
    end if;
  end process;
end;
```

For a concrete register transfer model, one controller process is instanciated, where the maximum number of control steps CS_MAX is given. As it can be seen from the source code, the phase changes with delta delay, i.e. no physical delay is modeled. So, the simulation of each control step takes 6 delta simulation cycles. The complete simulation takes CS_MAX $\times 6$ delta simulation cycles.

An additional synthesis step leading to a synthesizable RT description, which can be performed by commercial synthesis tools, would define, how control steps are implemented by clock signals and clock cycles. Of course, there are different ways to implement control steps. The choice of a specific control step implementation also influences the implementation of registers and modules.

2.3 Values, buses and ports

Modules combine values and compute new values, registers store values, and values are transferred via buses. We here consider natural number values (of course, we are also dealing with other types, but the mechanism for this is beyond the scope of this paper). We need two extra values representing "no value" and "illegal". Therefore, signals representing ports of functional units and buses are of type integer. Natural number values are regular values. The special values are introduced by:

```
constant DISC: Integer := -1;
constant ILLEGAL: Integer := -2;
```

Input ports of functional units and buses are modeled by resolved signals, because they are the sinks of transfers from several sources. The resolution function combining a list of integer values computes to DISC, if all integers in the list are DISC. It computes to ILLEGAL, if at least one integer is ILLEGAL or if at least two integers are not DISC. In this manner, it only computes to a natural number if exactly one natural number is in the list and all other values are DISC.

2.4 Transfers

A specific register transfer is partitioned into transfers from register output ports to buses, transfers from buses to module input ports, transfers from module output ports to buses, and transfers from buses to register input ports.

Buses and input ports of the functional units (sinks) are represented by resolved signals. Output ports of functional units (sources) are represented by regular signals. A transfer process assigns a source value to a sink signal at a specific phase:

```
entity TRANS is
  generic (S: Natural; P: Phase);
  port (CS: in Natural;
    PH: in Phase;
    InS: in Integer;
    OutS: out Integer := DISC);
end TRANS;

architecture transfer of TRANS is
begin
  process
  begin
    wait until CS=S and PH=P;
    OutS <= InS;
    wait until CS=S and PH=Phase'Succ(P);
    OutS <= DISC;
  end process;
end transfer;
```

An instance of a transfer process is activated at phase P of control step S and at the succeeding phase. In the first activation it assigns the source value to the sink signal. In the second phase, it assigns the DISC value to the sink value, which indicates that the transfer process no longer provides a value for the sink signal.

2.5 Registers

A register fetches a new value in each phase a transfer process is assigning a value to the register input port. If no transfer assigns to the input port, the old value is kept.

```
entity REG is
  port (PH: in Phase;
    R_in: in Integer;
    R_out: out Integer := DISC);
end REG;

architecture transfer of REG is
```

```
begin
  process
  begin
    wait until PH=cR;
    if R_in /= DISC then
      R_out <= R_in;
    end if;
  end process;
end transfer;
```

A register process always drives its output port as soon as the first value is assigned to its input port. Registers compute in the cR phase.

2.6 Modules

We here consider pipelined arithmetical modules, which can fetch operands in each control step and provide the results in the next control step. This can be easily extended to other delay characteristics. Here is the module process for the adder module used in the example:

```
entity ADD is
  port (PH: in Phase;
    M_in1, M_in2: in Integer;
    M_out: out Integer := DISC);
end ADD;

architecture transfer of ADD is
begin
  process
    variable M: Integer := DISC;
  begin
    wait until PH=cM;
    M_out <= M;
    if M /= ILLEGAL then
      if M_in1=DISC and M_in2=DISC then
        M := DISC;
      elsif M_in1 /= DISC and M_in2 /= DISC then
        M := M_in1 + M_in2;
      else
        M := ILLEGAL;
      end if;
    end if;
  end process;
end transfer;
```

This model assumes that either both operand values are natural values or both are DISC. The pipeline mode is established by means of the variable M.

The timing of module behavior is based on control steps. So, every combinational aspect must be covered in the variable assignment based sections of a module description. It is not possible to deal with cascades of combinational circuits linked via signals, because this would involve the usage of delta cycles for other reasons than control step phase changes. If we want to introduce several combinational levels then procedures, functions, and blocks can be used to group variable assignments associated with specific combinational parts.

2.7 Concrete register transfer models

A concrete register transfer model consists of the signal declarations of the control step and phase signals, the signal declarations for the ports of functional units and the buses, register, module and transfer processes. The design entity just introduces the external signals. We give a partial description for the example given in fig 1:

```
entity example is
  port (x_in,y_in,z_in: in Integer;
    x_out,y_out: out Integer := DISC);
end;

architecture transfer of example is
  -- timing signals
  signal CS: Natural;
  signal PH: Phase;
  -- module ports
  signal ADD_in1,ADD_in2: resolved_Integer;
  signal ADD_out: Integer;
  ...
  -- register ports
  signal R1_in,R2_in: resolved_Integer;
  signal R1_out,R2_out: Integer;
  ...
  -- buses
  signal B1: resolved_Integer;
  ...
begin
  -- modules
  ADD_proc:
    ADD port map (PH,ADD_in1,ADD_in2,ADD_out);
  ...
  -- registers
  R1_proc: REG port map (PH,R1_in,R1_out);
  R2_proc: REG port map (PH,R2_in,R2_out);
  ...
  -- transfers
  R1_out_B1_5: TRANS
    generic map (5,rA) port map (CS,PH,R1_out,B1);
  B1_ADD_in1_5: TRANS
    generic map (5,rB) port map (CS,PH,B1,ADD_in1);
  R2_out_B2_5: TRANS
    generic map (5,rA) port map (CS,PH,R2_out,B2);
  B2_ADD_in2_5: TRANS
    generic map (5,rB) port map (CS,PH,B2,ADD_in2);
  ADD_out_B1_6: TRANS
    generic map (6,wA) port map (CS,PH,ADD_out,B1);
  B1_R1_in_6: TRANS
    generic map (6,wB) port map (CS,PH,B1,R1_in);
  -- controller
```

```
  CONTROL: CONTROLLER
    generic map (7) port map (CS,PH);
end transfer;
```

Such a register transfer VHDL model has a clear structure and is easy to understand in the sense that there is a strightforward way of identifying register transfers. Because of the close relationship of control step phases to the VHDL simulation delta cycle, simulation results allow easily to locate design errors leading to resource conflicts: it would result to ILLEGAL values of resolved signals in specific simulation cycles associated with a specific phase of a specific control step.

Execution is very fast, because we need not to deal with asynchronous handshake, as it is often be used for exchanging values between modules when more abstract timing is modeled by means of VHDL without introducing physical time. The subset is quite useful for simulating designs at a very early stage of register transfer level design.

Last but not least, formal register transfer models can be easily translated to the VHDL register transfer model and vice versa. Our main intention when developing the VHDL subset for the register transfer level was to map formal timing abstraction mechanisms to transformations on VHDL subsets. For formal verification methods it is very important to keep behavioral models as easy as possible. The subset introduced here perfectly supported our demands. For example, let's consider the tuple from fig.1 representing a register transfer. The transfer process instances are derived in a straightforward manner (the important part of the tuple for a transfer process instance is underlined):

(R1,B1,R2,B2,5,ADD,6,B1,R1) → R1_out_B1_5
(R1,B1,R2,B2,5,ADD,6,B1,R1) → B1_ADD_in1_5
(R1,B1,R2,B2,5,ADD,6,B1,R1) → R2_out_B2_5
(R1,B1,R2,B2,5,ADD,6,B1,R1) → B2_ADD_in2_5
(R1,B1,R2,B2,5,ADD,6,B1,R1) → ADD_out_B1_6
(R1,B1,R2,B2,5,ADD,6,B1,R1) → B6_R1_in_6

Vice versa, if we know the transfer process, the tuples can be easily constructed

R1_out_B1_5, B1_ADD_in1_5 → (R1, B1, -, -, 5, ADD, -, -, -)
R2_out_B2_5, B2_ADD_in2_5 → (-, -, R2, B2, 5, ADD, -, -, -)
ADD_out_B1_6, B6_R1_in_6 → (-, -, -, -, -, ADD, 6, B1, R1)

These easy mappings lead to simple formal semantics, which form the basis for automatic verification tools, which compare register transfer level descriptions with either more abstract descriptions or more concrete descriptions. The close relationship of the register transfer model to th VHDL simulation delta cycle allows to prove the consistency of the dedicated semantics sketched above with VHDL simulation semantics.

3 Application

We have used our subset to model several designs at the abstract register transfer level. As an example, we have modeled resources (called MACC, multiplier/accumulator and cordic core) and register transfers of an IKS (inverse kinematics solution in robotics) chip [10], where a VHDL description only was given at the logic level. The logic level description contains a lot of details not relevant for the register transfer level. First, we have identified the functional units and the buses. Fig.3 shows the resources and used transfer paths (with some simplifications).

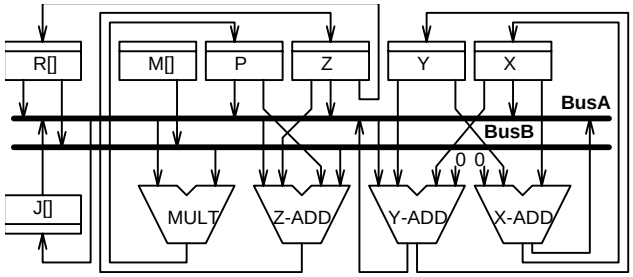

Figure 3: RT structure of the IKS chip [10]

The adders are not pipelined. The multiplier is a 2-stage pipelined unit. There are transfers via BusA and BusB, but there are also transfers between registers and adders via direct links (e.g. from register P to the right input port of Z-ADD) and transfers from registers to registers via direct links (from the Z register to the dual port R register file). Our model is already suited to model the scheme of the IKS chip. For example, for the direct link from register P to module input port Z_ADD a bus P_Z_ADD_in2 is introduced. For the direct link from Z to the register file R two extra buses and one extra module, which just copies the input to the output, are introduced. To keep the model – especially the timing – as simple as possible, it is better to model more resources than to extend the VHDL subset.

One extensions of the basic model became necessary. The adders may perform several arithmetical operations. Therefore, a register transfer also defines the operation to be performed by the module.

We have extracted the register transfers from the microcode for computing the IKS given in [10]. This could be easily automated. We have written a C program, that translates the microcode tables given in [10] to transfer process instances. As an example, we give the table entry for microprogram store address 7:

addr	cycle	opc1	opc2	m	J	R1	M/R2
1	1	3	4	0	6		

For opc1 and opc2 code maps exists. We give the code maps for opc1=20 and opc2=2:

opc1	Bus src	A dst	Bus src	B dst	Reg J	Reg R
3	J	y2			r	

zang	z1	z2	x1	x2	y1	y2
0	0	0	0	Y	0	A

opc2	Z	X	Y	setf
4	+	+	+	1

From these table entries, the transfers from registers to buses (J[6],BusA,y2,1), (Y,*direct*,x2,1) and the module operations Z := 0 + 0, X := 0 + Rshift(x2,i), Y := 0 + y2, F := 1 are derived.

4 Summary and Conclusions

We have introduced a VHDL subsets for register transfer models not depending on clock signals and physical delay. The simulation of the models is very fast because of their simple timing of transfers of values between the resources.

This IKS chip was an application, where the register transfers are derived from the microcode level. This register transfer level description is to be verified against a description at the algorithmic level. This is some kind of bottom-up evaluation of low level descriptions in order to find a link to more abstract descriptions.

Another application of our subset is high level synthesis, where the result of scheduling and allocation is given as a register transfer model. High level synthesis results are translated into our subset and can then be simulated at a high level before the next synthesis steps translate to a more concrete implementation. We are using this method in order to verify the correctness of high level synthesis results at an early stage. Formal semantics of initial algorithmic description and resulting register transfer level description are defined. An automatic proving procedure has been implemented, that performs the verification task. In this manner, our subset has been turned out be practicable for applications dealing with a more abstract register transfer level.

The timing of the abstract RT model is based on control steps and control step phases. There are several ways to translate a control step scheme into a clock scheme based on clock signals. The transformation into a usual synthesizable RT description based on clock signals can be performed automatically. We are now developing such automatic translation rules especially aiming at their formal correctness. The verification process is directly supported by the regular structure of our high-level RT model.

References

[1] M. Mastretti. VHDL quality: Synthesizability, complexity and efficiency evaluation. In EURO-DAC'95 [11], pages 482–487.

[2] P. J. Ashenden. *The designer's guide to VHDL.* Morgan Kaufmann Publishers, Inc., 1996.

[3] J.-M. Bergé, A. Fonkona, S. Maginot, and J. Rouillard. *VHDL Designer's Reference.* Kluwer Academic Publishers, 1992.

[4] M.S. Abrahams and A. Rushton. Translation of VHDL for logic synthesis. *Microprocessors and Microsystems*, 19(8):459–467, October 1994.

[5] A. Postula. *VHDL Specific Issues in High Level Synthesis*, pages 117–134. Kluwer Academic Publishers, 1992.

[6] European VHDL Synthesis Working Group. *Level-0 VHDL Synthesis Subset.* EVSWG, 1994.

[7] M. Selz, W. Ecker, and E. Villar. VHDL synthesis description: The need for level synthesis subsets. *Journal of System Architecture*, 42:105–116, 1996.

[8] D. Déharbe and D. Borrione. Semantics of a verification-oriented subset of VHDL. In *Proc. CHARME'95*, LNCS 987, pages 293–310. Springer, 1995.

[9] L. Arditi and H. Collavizza. Towards verifying VHDL descriptions of processors. In EURO-DAC'95 [11], pages 414–419.

[10] S. S. Leung and M. A. Shanblatt. *ASIC System Design with VHDL: A Paradigm.* Kluwer Academic Publishers, 1989.

[11] *Proc. EURO-DAC'95 with EURO-VHDL'95*, Brighton (Great Britain), 1995.