Clock Gating on RT-Level VHDL

Pieter J. Schoenmakers
<tiggr@ics.ele.tue.nl>
Eindhoven University of Technology,
Eindhoven, the Netherlands

J. Frans M. Theeuwen
<ftheeuwe@natlab.research.philips.com>
Philips Research Laboratories
Eindhoven, the Netherlands

Clock gating is a technique that can be used to reduce the power dissipated by the clock net. In this paper we describe the operation of a tool that performs clock gating on RT-level VHDL by transforming VHDL descriptions before they are processed further by logic synthesis.

1 Introduction

In synchronous digital circuits the clock net is responsible for a significant part of the power dissipation (up to 40% [1]). Clock gating tries to reduce the activity on parts of the clock net by disabling the clock on flip flops when they are clocked only to retain their value.

Though gated clocks can be introduced into a design manually, tools exist to perform clock gating automatically. In [2] a sequential circuit is modeled as a finite state machine (FSM). If the FSM does not change state, the clock on the flip flops can be disabled. For realistic circuits however, that situation does not occur frequently. Also the size of the FSM transition graph can be too large for this technique to be applicable.

In [3] a clock gating tool is presented that operates on gate-level net lists. This modus operandi has several drawbacks: due to the low level of the description, higher level entities such as multi-bit busses are difficult or impossible to recognize; the output of the transformation is not necessarily a very efficient implementation; and timing problems may arise.

In this paper, we describe a tool named Vault, that performs clock gating on RT-level VHDL descriptions. Transforming the description at this higher level means that logic synthesis of the resulting RTL description may optimize the generated expressions and enforce the timing restrictions.

In the following sections, the method used by the tool will be explained, followed by an example, results, and concluding remarks.

2 Method

Clock gating is a transformation that is performed on a synchronous digital circuit. For each flip flop in the circuit, the hold condition is determined, i.e. the condition under which the value that is clocked into the flip flop is identical to its current value. Under this condition, a transition on the clock input of the flip flop can be suppressed without changing the circuit's behavior. Such a modified clock is called a gated clock.

Since gating a clock involves a latch, and thus area overhead and extra power dissipation, flip flops with similar hold conditions are grouped to be clocked by the same gated clock. Power reduction is achieved if a gated clock governs enough flip flops with a combined hold condition that is often true.

Vault performs clock gating on an RT-level VHDL architecture. This can be the architecture of a full design or, if non-locality of the generated clocks poses a problem, on architectures that reside at a lower level in the hierarchy.

Vault functions as follows: First, the top architecture is instantiated, fully expanded, and the nets are flattened. Instantiated processes contain a description in terms of so-called operations, that are simple representations of the equivalent VHDL sequential statements. The operations within a process instance are private to that instance and can be removed or otherwise modified without affecting other instances of the same process.

Using this fully expanded description, the assigned values are computed for every net, by computing the corresponding BDDs. For this purpose, no other values than `true` and `false` (or `'1'` and `'0'`) are discerned, and every assignment to a net must be performed under a condition that does not overlap with the combined condition of any other assignments to the same (partial) net (this restriction is also posed on VHDL code to be synthesizable).

Vault must handle combinatorial logic, as well as level and edge triggered flip flops and latches, including variations with synchronous or asynchronous set and reset inputs. For this purpose, the assignments to each net are modeled after the VHDL code in figure 1. The `comb_condition` is the condition that the non-clocked assignment `s <= v1` is performed; similarly, the `clk_condition` governs the clocked assignment `s <= v2`.

```
IF comb_condition = true THEN
  s <= v1;
ELSIF clk'event AND clk = '1' THEN
  IF clk_condition = true THEN
    s <= v2;
  END IF;
END IF;
```

Figure 1: Model of signal assignments.

When the BDDs for all nets have been computed, they are grouped in hold domains using the same algorithm as in [3]. Each hold domain will be governed by its own gated clock. For every architecture instance that is involved in at least one assignment to a net that is to be gated, a new architecture is created. For a gated clock `gclk` with assignment condition `cond` (i.e., the inverse of the hold condition), two signals and their concurrent assignments are generated, which are synthesizable using a latch and an AND-gate (figure 2):

```
SIGNAL gclk, gclk_in: std_logic;

gclk_in <= cond WHEN clk = '0'
                ELSE gclk_in;
gclk <= clk and gclk_in;
```

In the generated architectures, copies are made of each process that contains assignments to nets that are to be gated. Each of these copies is specialized on one of the gated clocks involved. A clocked assignment to a net is only retained in the process that is specialized on the clock governing that net.

One final hurdle to be taken is preserving behavior of the circuit given the semantics of a

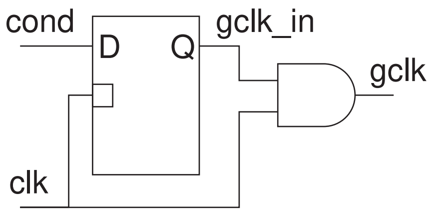

Figure 2: Synthesized gated clock.

VHDL simulator. With the VHDL delta delay model, the original clock signal must be delayed before being used in the generated architectures. Otherwise, the following sequential VHDL code:

```
IF clk'event and clk = '1' THEN
  a <= NOT a;
  IF b = '1' THEN
    c <= a;
  END IF;
END IF;
```

will not show the same behavior as the following two statements (in the same or different processes, with the proper sensitivity):

```
IF clk'event AND clk = '1' THEN
  a <= NOT a;
END IF;

IF gclk'event AND gclk = '1' THEN
  IF b = '1' THEN
    c <= a;
  END IF;
END IF;
```

Therefore, to clock assignments that are not gated, a delayed version of the not-gated clock is used instead of the original clock:

```
SIGNAL dclk: std_logic;

dclk_gen: dclk <= clk and '1';
```

3 Handling complexity

Using BDDs to compute the next value of a net has the disadvantage that the program may need huge amounts of resources: even a simple 16*16 bit multiplication will have it run out of memory fast. To cope with this problem, a net can be made *complex*, either as specified by the user, or as the BDDs needed to represent its value grow beyond a user-defined threshold. If the BDDs of a partial net exceed this threshold, the whole net is marked complex, and its assignment condition becomes the condition that any of the subnets is assigned. Combinatorial complex nets are handled like primary inputs, clocked nets, and combinatorial nets that are conditionally assigned, in that they are never expanded in expressions using them. As long as the condition of a clocked assignment (`clk_condition` in figure 1) to a net is not complex, the clock of the flip flops in the net can still be gated.

4 Example

Given the entity and architecture of figure 3, which constitute an 8 bit conditional counter. Without limits on the maximum BDD size, the LSB of the counter does not make it into the hold domain consisting of the other bits, because with the small size of its assignment condition, `en`, the hold condition of the domain would become too weak, clocking the other flip flops in the domain far too often.

In the resulting output, the process `p1` is duplicated: one instance (`p1`) operates under the normal clock; the other (`p1_gclk_0`) is specialized on the gated clock (figure 4).

With a BDD size limit of 10 nodes, the expression for the next value of `a` becomes too complex, and all bits of `a` end up having the same apparent assignment condition, `en`. Even though this condition is often true,

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY incr1 IS
  PORT (clk, en: std_logic;
        a: OUT unsigned (7 DOWNTO 0));
END;

ARCHITECTURE struct OF incr1 IS
BEGIN
  p1: PROCESS (clk)
  BEGIN
    IF clk'EVENT AND clk = '1' THEN
      IF en = '1' THEN
        a <= a + to_unsigned (1, 8);
      END IF;
    END IF;
  END PROCESS;
END;
```

Figure 3: Sample architecture and its entity.

gating 8 flip flops with it is interesting, and all bits end up in the same hold domain (figure 5).

5 Results

The experiments conducted with Vault so far show run times in the order of seconds. For a 267kB VHDL description of a design containing 4041 nets, the run time is 26.84 seconds on a 180MHz HP K260. In this example, the average BDD size of the values of the nets is 4.64, the largest BDD is 5306, and the run time is not bound by any particular part of the program.

On a trivial example containing an 8*8 bit multiplication, the 8.52 second run time is mostly spent in computing the BDDs. With a limited BDD size, making the nets of the multiplication result complex, the run time is 4.15 or 1.92 seconds, for a BDD size limit of 1000 and 100 nodes, respectively.

```
ARCHITECTURE struct_gated_0 OF incr1 IS
  SIGNAL dclk: std_logic;
  SIGNAL gclk_0_in: std_logic;
  SIGNAL gclk_0: std_logic;
BEGIN
  dclk_gen: dclk <= clk and '1';
  gclk_0_in <= a(0) and en WHEN clk = '0'
                           ELSE gclk_0_in;
  gclk_0 <= clk and gclk_0_in;
  p1: PROCESS (dclk)
    VARIABLE a_0: unsigned(7 DOWNTO 0);
  BEGIN
    IF dclk'event and dclk = '1' THEN
      IF en = '1' THEN
        a_0 := a + to_unsigned (1,8);
        a(0) <= a_0(0);
      END IF;
    END IF;
  END PROCESS;
  p1_gclk_0: PROCESS (gclk_0)
    VARIABLE a_1: unsigned(7 DOWNTO 0);
  BEGIN
    IF gclk_0'event and gclk_0 = '1' THEN
      a_1 := a + to_unsigned (1,8);
      a(7 DOWNTO 1) <= a_1(7 DOWNTO 1);
    END IF;
  END PROCESS;
END struct_gated_0;
```

Figure 4: Gated without BDD limit.

Vault can generate the same gated clocks as those generated by the Hold tool described in [3].

6 Conclusion

The tool Vault presented in this paper can clock-gate large designs at the RT-level before logic synthesis. Though it employs BDDs to describe the actual or next-state values of signals, it can cope with a design containing constructs that are notorious for their BDD size,

```
ARCHITECTURE struct_gated_1 OF incr1 IS
  SIGNAL gclk_0_in: std_logic;
  SIGNAL gclk_0: std_logic;
BEGIN
  gclk_0_in <= en WHEN clk = '0'
                  ELSE gclk_0_in;
  gclk_0 <= clk and gclk_0_in;
  p1_gclk_0: PROCESS (gclk_0)
  BEGIN
    IF gclk_0'event and gclk_0 = '1' THEN
      a <= a + to_unsigned (1,8);
    END IF;
  END PROCESS;
END struct_gated_1;
```

Figure 5: Gated with a BDD limit of 10 nodes.

such as large (e.g., $>$ 16 bit) multipliers. This is the result of Vault's capability to abstract multi-bit signals that are too complex to be described by BDDs. Even when the next-state value is too difficult to describe, the condition under which that value is assigned can still be administered, and the assignment can be governed by a gated clock.

References

[1] D. Dobberpuhl, R. Witek, "A 200MHz 64b dual-issue CMOS microprocessor," IEEE International Solid-State Circuits Conference, 1992, pp 106–107.

[2] L. Benini, G. De Micheli, "Transformation and synthesis of FSMs for low-power gated-clock implementation," International Symposium on Low Power Design, 1995.

[3] Frans Theeuwen, Eric Seelen, "Power reduction through clock gating by symbolic manipulation," VLSI: Integrated Systems on Silicon, R. Reis, L. Claesen (eds), pp 389–399.