Project Assignment in TSTE85 Low Power Electronics

2019

Henrik Ohlsson, Mark Vesterbacka, and Erik Backenius

[The figure on the front page is from “Designing Low-Power Circuits: Practical Recipes” by L. Benini, G. De Micheli, and E. Macii, *IEEE Circuits and Systems Magazine*, vol. 1, 2001]

1 Assignment

1.1 Objective

A 16x16 multiplier with a throughput of 100 Msamples/s should be designed for low power. You should solve the six tasks below in a group consisting of two students.

There are some important changes in the project. Please read about them in the end under the sub-section for each tool. These instructions are vital to achieve comparable results.

1.2 Tasks

1.2.1 Task 1

Design a two's complement multiplier consisting of a partial product generator, a carry-save adder tree, and a final carry propagate adder according to Fig. 1. Verify the functionality in Modelsim. Estimate the propagation delay (power supply voltage 3.3 V) and the area in Design Compiler. Use Nanosim for estimation of power consumption.

A time margin (called "slack" in the Design Compiler) of 10% of the current clock period is here required to guarantee a proper operation of the multiplier. This time margin should be considered for all tasks in this project assignment. For example, for a clock frequency of 100 MHz, the required time margin is 1.0 ns.

Scale the supply voltage so that the maximum throughput equals 100 Msamples/s. Estimate the power consumption after voltage scaling. Use this multiplier as a reference when you solve the other tasks.

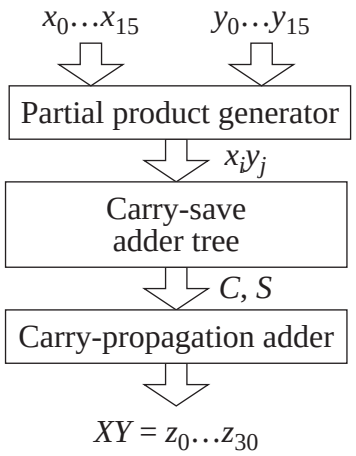

Figure 1: Reference multiplier.

1.2.2 Task 2

Pipeline the multiplier in task 1 by introducing a register along a cut A-A' in the adder-tree (see Fig. 2). Try to place the cut so that the paths in the adder tree are divided into two equally long parts with respect to propagation time. Estimate the power consumption and the area. Scale the supply voltage so that the maximum throughput equals 100 Msamples/s. Estimate the power consumption after voltage scaling.

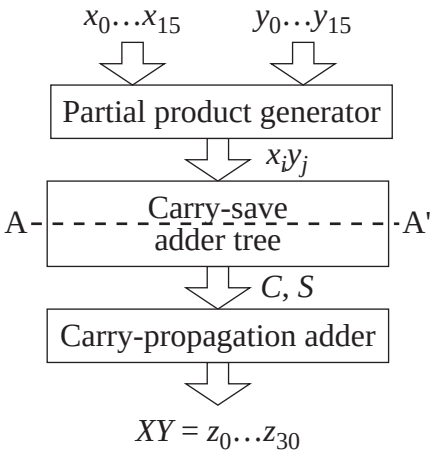

Figure 2: Pipelined multiplier.

1.2.3 Task 3

Repeat task 2 with two pipeline registers along cuts B-B' and C-C' in the adder-tree according to Fig. 3. Try to place the cuts so that the paths in the adder tree are divided into three equally long parts.

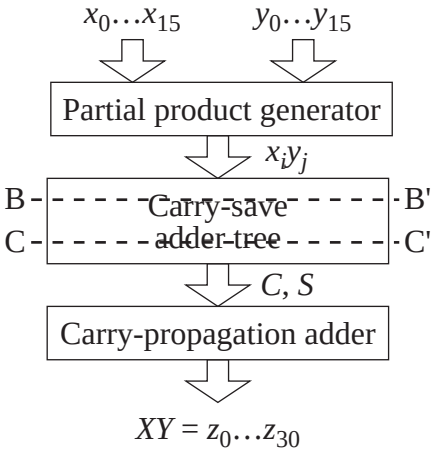

Figure 3: Multiplier with two pipeline registers.

1.2.4 Task 4

Repeat task 2 with three pipeline registers along cuts D-D', E-E' and F-F' in the adder-tree according to Fig. 4. Try to place the cuts so that the paths in the adder tree are divided into four equally long parts.

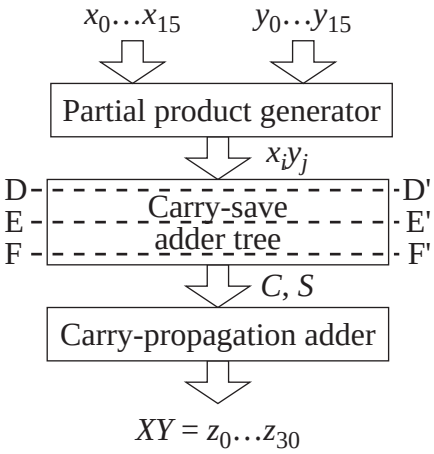

Figure 4: Multiplier with three pipeline registers.

1.2.5 Task 5

Design an interleaved version of the original multiplier by the use of two multipliers of the type used in task 1 (see Fig. 5). By the use of a complementary clock the registers available in the project directory can be used. Choose a clock frequency that corresponds to a throughput of 100 Msamples/s. Estimate the power consumption and the area. Scale the supply voltage so that the combined maximal throughput equals 100 Msamples/s. Estimate the power consumption after voltage scaling.

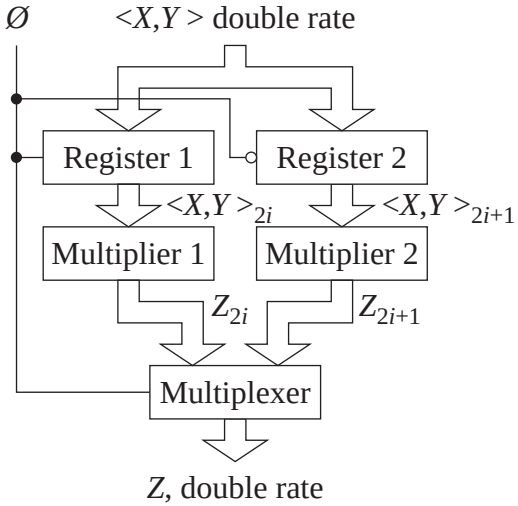

Figure 5: Two parallel multipliers.

1.2.6 Task 6

Repeat task 5, but interleave two pipelined multipliers of the type used in task 2.

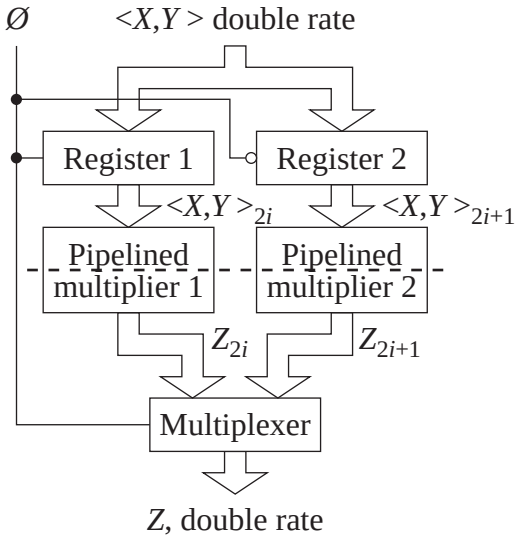

Figure 6: Two parallel pipelined multipliers.

1.3 Reporting Your Work

Write a well-written lab report where you explain and make conclusions of your results. The assumed target reader of the report is a student that has not yet taken “Low Power Electronics”. Hence you have to be clear about what you write. Hand in a copy of your report as a pdf file to your laboratory assistant no later than the announced deadline (see http://www.isy.liu.se/en/edu/kurs/TSTE85/projekt/). Assignments are due at midnight on the due date.

For all cases the following parameters have to be included.
- Critical path prior to voltage scaling.
- Scale factors, i.e., new V_{DD}, time scaling factor, and relative power dissipation.
- Total average power dissipation.
- Total design area.
- Motivation for placement of pipeline registers. Predicted critical path after insertion of each pipeline cut.

On the cover to the report these items have to be listed.
- Full name of each student in the group.
- Personal id for each student in the group.
- Student id for each student in the group.

2 Instructions

2.1 Setting up the Project

Open a terminal window and start by logging onto the server `naum` via the intermediate server `ssh`.

```
ssh -X ssh.edu.liu.se
ssh -X naum.ad.liu.se
```

Create a project directory in your low power course directory.

```
cd TSTE85
mkdir project
```

Copy the directories (`vhdl-files`, `DesignCompiler`, `Nanosim` and `Modelsim`) to your project directory.

```
cd project
cp -r /coop/e/eks/course/TSTE85/project/*  .
```

You should be in the `Modelsim` directory when you run Modelsim, in the `DesignCompiler` directory when you run Design Compiler, and in the `Nanosim` directory when you run Nanosim.

2.2 The Building Blocks

Read the vhdl-part of the compendia of laboratory 1. Here, you will find some useful information of how to build a new component of existing components. Use the file `multiplier_i` (found in `.../project/vhdl-files`) for the corresponding task *i* (*i* = 1, 2, 3, 4, 5, 6).

2.2.1 Partial Product Generator, ppgen.vhdl

The block **ppgen** generates the partial products given by the coefficient. Each output is either zero, corresponding to a zero in that specific position of the coefficient, or the input shifted right, with the number of shifts corresponding to the significance of the bit in the coefficient.

2.2.2 Carry-Save Adder, csa32.vhdl

The carry-save adder is a very fast adder for adding three operands. The result of the operation is given as two operands, one sum and one carry vector. This adder is often used when we are to add several operands together, as in a multiplier. The adder we will use, **csa32**, is 32 bit wide.

2.2.3 Carry-Save Adder Tree, csatree.vhdl

In **csatree**, carry-save adders are used to form an adder tree that adds the partial products together. The tree has a minimum number of levels in order to compute the sum as fast as possible. To start with there are no pipeline registers inside the tree, but it should be rather straightforward to introduce pipeline registers in the tree (this is described in section 2.7).

2.2.4 Carry Look-Ahead Adder, cla32.vhdl

The output from the carry-save tree is two operands, a sum and a carry vector. To form the final result in one operand a fast carry propagation adder is needed. We will use a pipelined 32-bit carry look-ahead adder, **cla32**. The adder has been divided in block of four bit each. Between these blocks pipeline registers have been inserted. Each block is implemented using 4-bit carry look-ahead adder.

2.2.5 Registers and 2-to-1 Multiplexers

reg16 is a positive edge triggered 16-bit register (`reg16.vhdl`).
reg32 is a positive edge triggered 16-bit register (`reg32.vhdl`).
mux16 is a 2-to-1 multiplexer for 16-bit data (`mux16.vhdl`).
mux32 is a 2-to-1 multiplexer for 32-bit data (`mux32.vhdl`).

2.3 Design Compiler

In the Design Compiler you do the same steps as in the first laboratory. You should define a clock period of 1 ns before you compile the design. This makes the tool aim for a short propagation time. However, the tool will not succeed to meet this timing requirement, but it will give you a circuit containing high-speed standard cells.

Before you run report_timing you may change the clock period to the intended (e.g. `10n` or `20n`), **do not recompile**. Take a look into the timing report and note from which time point the propagation delay is counted. In the interleaving cases this time point may NOT be 0 ns. The total area of the design is also of interest.

When you save your design as a Verilog-file (to be used with Nanosim), a lot of warnings will probably pop up. If your multiplier did work as intended in Modelsim, you do not have to care about the warnings.

If you save the designs as db-files do not place them in the `DesignCompiler` directory.

After “compile” in Design Compiler you can look into the different circuits and see which standard cells that have been chosen. It may be interesting just to see what kind of standard cells the vhdl-files have resulted in.

2.4 Modelsim

Verify the functionality of the multiplier in Modelsim for each design task. Use file `multiplier_i` for task *i* (*i* = 1, 2, 3, 4, 5, 6). A do-file can be useful in Modelsim. The content (e.g., force commands) in the file filename.do is read and executed with the command

```
do filename.do
```

2.5 Nanosim

In the Nanosim folder there is a test vector file (`test.vec`) that you may use. The transition activity of the inputs is 0.5 (random input). Use for example the following command for the first task (it is a long command — do not break lines).

```
nanosim -nvlog multiplier_1.v -m multiplier_1 -nspi spice.sp
-L /coop/e/eks/course/TSTE85/c35_CORELIB -nvec test.vec -c cfg
 -t 1000.00 -o multiplier_1
```

Simulate for at least 1000 ns (`-t 1000.00`). You may simulate up to 2000 ns (i.e., `-t 2000.00`) since the test vector file contains 200 input vectors. Use the same simulation time for each design task to make them comparable.

In the file `spiceNNN.sp` the power supply voltage and the clock signal are defined. Do not forget to do the changes in this file for the different tasks. Use `spice50.sp` for 50 MHz clock and `spice100.sp` for 100 MHz clock. Do not forget to change `par_vdd` when you scale the power supply voltage.

You may get more wasted current in the interleaved cases (which is not expected). The reason for this is that the inverter for the inverted clock is by default small (with the current set-up file). If the design should be manufactured, the small inverter would have been substituted with a chain of inverters.

For low power supply voltages some logic blocks may get no power consumption at all, which is a little bit too good to be true. The reason is that Nanosim (in this project) can not properly handle power supply voltages below 1.33 V. Therefore, keep the power supply voltage on or above 1.33 V, then the result will be reliable.

2.6 Supply Voltage Scaling

In Design Compiler it is possible to scale the supply voltage. But, the models that are provided for the standard cells that we use in this project are only accurate in timing for the default power supply voltage of 3.3 V. For this reason we use results from Hspice simulations, where the accuracy is high.

The process (AMS 0.35 µm) that we use in this project was also used in the second laboratory (Interleaving of a FIFO register). The effect of voltage scaling on propagation delay was briefly studied with the tool CosmosScope (that displays the result of a Hspice simulation). You will use the same graph in the project as in the laboratory when the power supply voltage should be scaled and the new propagation delay should be estimated. First, estimate the propagation delay (@ 3.3V) of your design with the use of Design Compiler, then use the graph to estimate the lowest possible power supply voltage.

The power consumptions you estimate with the use of Nanosim. When you estimate the power for a lower supply voltage, change the parameter for the power supply voltage (in `spice.sp`) in the same way as you did in the second lab.

2.7 Introducing Pipelining in the Carry-Save Adder Tree

Below is an example given on how pipeline registers are introduced in the adder tree. In the example, registers are introduced between the outputs from the second-level adders and the inputs of the third-level adders in the tree.

```
PROCESS(clk) BEGIN
if rising_edge(clk) then level_3_in_0 <= level_2_out_0;
level_3_in_1 <= level_2_out_1; level_3_in_2 <= level_2_out_2;
level_3_in_3 <= level_2_out_3; level_3_in_4 <= level_2_out_4;
level_3_in_5 <= level_2_out_5; level_3_in_6 <= level_2_out_6;
level_3_in_7 <= level_2_out_7;
end if;
end PROCESS;
```

2.8 Interleaving

When interleaving the multiplier, define an internal clock signal that is the inverted version of the original clock signal (e.g. `clk_inv <= not(clk);`). Do not define new registers that triggers on the falling edge. This prevents some risk of synthesize problems in Design Compiler.

Do not forget to change the clock frequency (in `spice.sp`) when running Nanosim for the designs using interleaving.

2.9 Compile the VHDL-files

Compile the vhdl-files in the right order. For example, `csa32.vhdl` is used in `csa_tree.vhdl`. Hence, `csa32.vhdl` must be compiled before `csa_tree.vhdl`. This is of importance for both Modelsim and Design Compiler.

2.10 Extra Task (Optional)

This is **not** a mandatory task, and it will not give you a higher grade, but it will probably give you some challenge and knowledge. Modify the partial product generator for lower power consumption (see Lesson 5, Exercises 6 and 7). Redo the first task with the new partial product generator.