EE 271

Final Design Project

Implementing and Analyzing 64-bit Conditional Sum and Carry Select Adders

I. Project Description

The purpose of this project is to implement, optimize and analyze performance of 2 different implementations of 64-bit adders (an adder performs both addition and subtraction instructions): (1) Conditional-Sum Adder (CSuA), (2) Carry Selected Adder with 8-bit groups (CSeA-8bG). Each adder will take a number of clocks to complete the calculation and the designs must be modeled at RTL level. The number of clocks can be defined based on your design and implementation of the pre- and post-calculation (if any) so that you can get best performance (binary addition is a combinational circuit). Analysis must be performed for three important metrics: timing performance, circuit area, and power consumption. The addition/subtraction process can be summarized as shown in the table below:

Operation	Signs of Operands	Process
A + B	A & B are both positive	– Binary add normally – Overflow if sign-bit of "sum" is different from sign-bit of the operands (Negative "sum" means overflow) – *There will be NO carry-out (carry-out = 0)*
	A & B are both negative	– Binary add normally – Overflow if sign-bit of "sum" is different from sign-bit of the operands (Positive "sum" means overflow) – *There will be carry-out (carry-out = 1)*
	A & B are different in sign	– Binary add normally – The "sum" can be positive or negative – There will be NO overflow – *If carry-out = 1, then magnitude of positive operand is larger than magnitude of negative operand and so the sum is positive* – *If carry-out = 0, then magnitude of positive operand is smaller than magnitude of negative operand and so the sum is negative*
A - B		– Take two's complement of B (call B') – Perform A + B' (same as A + B above with B now is B')

II. Design Specifications and Circuit Interface

The **exact** interface ports of all adder circuits must be as below:

- `ope1[63:0]:` 64-bit operand #1 in 2's complementary format
- `ope2[63:0]:` 64-bit operand #2 in 2's complementary format
- `addsub:` 0: ope1+ope2, 1: ope1-ope2
- `clock`
- `reset:` 0 = reset
- `sum [63:0]:` 64-bit sum or difference in in 2's complementary format
- `cout:` 1-bit carry-out or borrow-in
- `overf:` 1-bit overflow flag (1: overflow, 0: NOT overflow)

#

#

#

- `start`: You can implement the start signal as either positive-edge active or high-level active.
  Positive-edge active: The circuit is initially idle and will start at the positive-edge of the `start` signal. The circuit restarts whenever the `start` signal once again changes from 0 to 1
  High-level active: The circuit is initially idle and will start whenever the `start` signal is high (logic 1). For each clock, the circuit checks the `start` signal and restarts whenever the `start` signal is high (logic 1). So in this case, the `start` signal should not active for more than 1 clock period
- `complete`: Active-high output signal from the circuit to inform the exterior that the calculation is completed (64-bit sum is available). The `complete` signal becomes inactive (`0`) at the positive-edge of the `start` signal and become active (`1`) whenever the calculation was completed.

```
(overf)' AND (complete)   = 1: sum is valid
                          = 0: sum is invalid
```

All your work, including the details of your implementation, test, measurement, analysis, design options, EDA tool setting and configurations must be clearly described as stated in the project report template. Note that for this project, **the analysis is very important and should be documented clearly and neatly.** The report must include exact information as requested. Do NOT skip the information and do NOT report extra information. Grading will be based on the quality and completeness of your project and report.

III. Libraries and EDA Tools

- Use Toshiba library at **`/apps/toshiba/sjsu/synopsys/tc240c/`** for your synthesis and analysis. This is a technology library with 250 nm technology operating at 2.5V. For best and worst delays (hold and setup time violation checks), use **`tc240c.db_BCCOM25`** and **`tc240c.db_WCCOM25`**, respectively. Note that in using Toshiba library, the `tc240c.workview.sdb` is the symbol library that you should use.
- RTL-level simulations can be performed on any simulator but gate-level (netlist) simulations must be performed with either **Synopsys VCS** or **Cadence NCVERILOG**. Synthesis must be performed with **Synopsys Design Vision** (or **Design Compiler**)
- All simulations must be performed at both RTL and gate (netlist) levels

Other technology libraries available on the system that you can try include the 0.18 μm and 0.25 μm technologies from TSMC and 0.35 μm and 0.5 μm technologies from AMI as listed below:

- `/apps/cadence/OSU_LIB/osu_stdcells/lib/tsmc018`
- `/apps/cadence/OSU_LIB/osu_stdcells/lib/tsmc025`
- `/apps/cadence/OSU_LIB/osu_stdcells/lib/ami035`
- `/apps/cadence/OSU_LIB/osu_stdcells/lib/ami05`

Or default LSI libraries available from Synopsys (`lsi_7k.db, lsi_9k.db, lsi_10k.db`) at `/apps/synopsys/SYNTH/libraries/syn`

IV. Design Implementation and Testing

#

#

The implementation must be done at RTL level in order for you to control the architecture and structure of your designs. You are **NOT** allowed to use Verilog arithmetic operators +, -, *, and / for hardware implementation (but of course you can use these operators for programming purposes and in your testbenches.) You must make sure that the final optimized circuits that you implemented have structures as you intended to implement.

Besides testing each module during the project development, you are required to develop final test cases to comprehensively test the operations of the designs and display (by using both `$display` system function and waveforms) the input operands and output results together with timing data to represent the circuit delay performance. Final test cases must be able to perform pre-synthesis (RTL) functional verification and **post-synthesis (netlist) functional and timing** verifications. Select a set of test data such that you can re-verify the static timing (during the synthesis) with dynamic timing (during post-synthesis simulation). Select the data point that you want to output in order for you to exam the dynamic timing delays as function of the input operands.

In order to run VCS with `netlist`*, you need to point to the Verilog source of the target library. As an example of synthesizing with Toshiba library, the VCS command for simulating your netlist named* `adder64_netlist` *should be as below:*

```
vcs +v2k -debug_all -gui -y /apps/toshiba/sjsu/verilog/tc240c
+libext+.tsbvlibp adder64_TB.v adder64_netlist.v
```

Moreover, you can use Cadence simulator (`ncverilog`) as below:

```
ncverilog -y /apps/toshiba/sjsu/verilog/tc240c +libext+.tsbvlibp
+access+r adder64_TB.v adder64_netlist.v
```

where `.tsbvlibp` is the suffix of Verilog source files of `tc240c` library and `/apps/toshiba/sjsu/verilog/tc240c` is the library directory

Since tc240c library has nanosecond time scale and 10 picosecond time-precision, the timescale directive below should be included in the Verilog testbenches

```
`timescale 1 ns / 10 ps
```

Note that some Verilog files in this technology library are encrypted (protected) by Cadence EDA tool and VCS may not be able to decrypt them. If that is the case, you will then get error messages. Using `ncverilog` for netlist simulation should be fine since the library was provided to us by Cadence. If anyway you get error message about file protection, try to find out in your netlist which encrypted Verilog files that are used and then try to replace them with non-encrypted files. Below is the list of encrypted files.

- `tsbLD2SFprim.tsbvlibp`
- `tsbSCK2prim.tsbvlibp`
- `tsbMUXXprim.tsbvlibp`
- `tsbSCK1prim.tsbvlibp`
- `tsbCTLprim.tsbvlibp`
- `tsbFD2BASICprim.tsbvlibp`
- `tsbFD4BASICprim.tsbvlibp`
- `tsbCHKprim.tsbvlibp`

#

- `tsbRCK1prim.tsbvlibp`
- `tsbCLD3SFprim.tsbvlibp`
- `tsbCLD4SFprim.tsbvlibp`
- `tsbMAJprim.tsbvlibp`
- `tsbLDP1prim.tsbvlibp`
- `tsbFD1BASICprim.tsbvlibp`
- `tsbCFD3BASICprim.tsbvlibp`

V. Project Report and Report Submission

EE271 final project is an individual project. Each student will work on the project independently and no information should be shared among students. Students are free to share ideas at high level approaches with each other, but sharing of Verilog code (or code fragments) and any part of the report is NOT allowed. Students are NOT allowed to work on the code, or any code portion, as a group. Code comparison tool will be used to analyze the student work and violation will NOT be tolerated.

Each student is required to submit on class

- A full **report in PDF format**
- A Verilog **source code file in text format** (put all Verilog files in one file, including testbenches)

Students are responsible for providing completed information such that grader can re-simulate and re-synthesize the design for grading purposes. **Be aware that the submitted report will be plagiarism checked with turn-it-in and students will NOT see the similarity results but only the graders.**

The project report must be in the right format and must include completed information about the project as shown in the report template. Download the report template on the class canvas and edit your report directly from the template. Do not change the format, fonts, and page setup of the report template but just fill-in and writing the requested information. Do NOT skip the information and do NOT report extra information. Grading will be based on the quality and completeness of your project and report.

The report and Verilog file names MUST be in the formats as below:

Report: SJSUID_Lastname_Report.pdf
Verilog file: SJSUID_Lastname_SourceCode.v

#

#