Practical Assignment №5

Practical assignment №5 consists of 10 exercises: 6 are mandatory and 4 are optional for extra points.

These exercises are divided into 3 subjects — FIFO, pipelined computational units, and schoolRISCV processor core.

The assignment has the following directory structure:

- `05_01_fifo_with_counter_baseline` - baseline example FIFO for examination
- `05_02_fifo_pow2_depth` - first exercise on FIFO
- `05_03_fifo_empty_full_optimized` - second exercise on FIFO
- `05_04_fifo_with_reg_empty_full` - third exercise on FIFO
- `05_05_a_plus_b_using_fifos_and_double_buffer` - exercise with `a + b` formula
- `05_06_sqrt_formula_pipe` - exercise with formula 2 from assignment №4 and `isqrt` module
- `05_07_cpu_baseline` - baseline schoolRISCV processor example for examination
- `05_08_cpu_with_comb_mul_instr` - exercise on adding multiplication instruction
- `05_09_cpu_mul_with_latency` - optional exercise
- `05_10_cpu_with_b_instr` - optional exercise
- `05_11_cpu_fetch_with_latency` - optional exercise
- `05_12_three_cpus_sharing_instr_memory` - optional exercise

Every mandatory exercise has an example section starting with `// Example` comment, and you should write your solution below the `// Task` comment.

Every exercise also has `tb.sv` file, which does a minimal check of your solution.

Preface

While working on solutions, it is possible to check each separate exercise with `run_using_iverilog` script in each directory. It is also possible to run the script in the root directory of the assignment to check all exercises.

In `tb.sv` file of any task, you can uncomment `$dumpvars;` line for generation of `dump.vcd` file. Then you can use `gtkwave dump.vcd` command to view the waveform, or uncomment the according line in `run_all_using_iverilog` script.

Exercise 1. FIFO with a depth of power of 2

> Exercise: Mandatory
>
> Directory: `05_02_fifo_pow2_depth`

Task: Implement missing logic for extended read pointer and `empty` signal to make a fully functional FIFO.

Exercise 2. Optimized FIFO

> Exercise: Mandatory
>
> Directory: `05_03_fifo_empty_full_optimized`

FIFO optimization is implemented by omitting the depth dependent counter, but there are 2 additional bits to determine relative pointer position.

Task: Implement logic to update read pointer and circle parity, and form `full` signal according to `equal_ptrs` and/or `same_circle`.

Exercise 3. FIFO with empty and full registers

> Exercise: Mandatory
>
> Directory: `05_04_fifo_with_reg_empty_full`

In this exercise, empty and full signals are registers, inner signals are formed combinationally, and then they are written in according registers.

Task: Implement logic for combinational signal `rd_ptr_d`, and also for signals `empty_d` and `full_d` when `pop` is active.

Exercise 4. a + b formula with FIFO

> Exercise: Mandatory
>
> Directory: `05_05_a_plus_b_using_fifos_and_double_buffer`

This exercise implements the scheme that is described in "FIFO for the little ones" (Article is in Russian) in example 3. Two FIFOs on inputs of addition operation align operands in time, and the result is put into a double buffer from formula output.

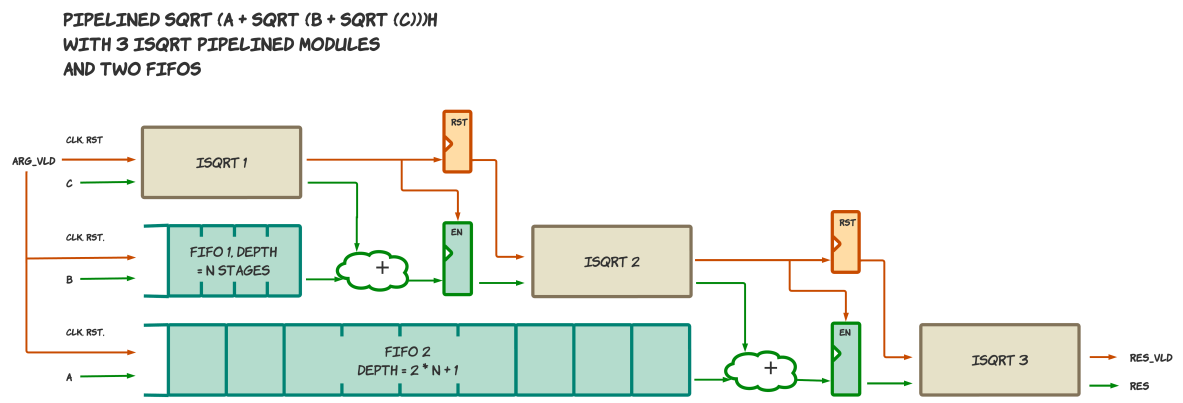
Task: Connect all the inner modules of the exercise, using external valid/ready signals as well. The start of each connection (assign) is listed under `Task` comment, you need to implement the logic after `=`

Exercise 5. Formula 2 with FIFO

> Exercise: Mandatory
>
> Directory: `05_06_sqrt_formula_pipe`

The exercise directory structure is the same as in exercises with formulas in assignment 4. It is recommended to read the article by Yuri Panchul "What American students can and what cannot write in SystemVerilog for ASIC and FPGA" in FPGA-Systems Magazine (Article is in Russian).

Task: Implement one of the last cases described in the article — calculation of Formula 2 using pipelined `isqrt` module and already implemented module from `flip_flop_fifo_with_counter` file.



Exercise 6. New multiplication instruction in the processor

> Exercise: Mandatory
>
> Directory: `05_08_cpu_with_comb_mul_instr`

In this exercise, you have to examine schoolRISCV processor from `05_07_cpu_baseline` directory. In this and following exercises, the baseline structure of the processor is similar, and is being extended depending on the exercise. So it's advised to examine `30_schoolriscv` lab in `basics-graphics-music` repository as well.

It's also useful to take a look at RISC-V Specification (RV32I User-Level ISA and RV32M extension) and extend the processor according to the specification.

Task: Add the multiplication instruction `mul` for the correct execution of `program.s` program, that calculates the factorial. For that, you have to extend `sr_cpu.svh` by adding correct constants, and also extend `sr_alu.sv`, or create a new module `sr_mdu.sv`.

Exercise 7. Multiplication instruction with latency

> Directory: `05_09_cpu_mul_with_latency`

Task: Based on previous exercise, implement `mul` instruction with the latency of 2 (or parameterized latency `N`), thus the result would occur 2 (`N`) clock cycles after the instruction.

Note: Multiple variants are allowed - from very simple, where the whole processor stalls while waiting for the result from the multiplication module, to building a simple pipelined processor with bypasses (forwarding). Bypasses could be done between multiplications and additions (and other operations of combinational ALU), and also between 2 separate multiplications, which provides maximal throughput. Two multiplication operations could follow one another and should be processed in a multiplication module simultaneously. From experience, a student who implemented this variant as homework, after the interview got an offer from a big electronics company.

Exercise 8. New unconditional jump instruction `b`

> Directory: `05_10_cpu_with_b_instr`

Task: Add unconditional jump `b` instruction. To do that, it is necessary to extend `sr_control.sv` and `sr_decode.sv` modules, as well as to add logic to update `pcNext` and save the result in the register.

Exercise 9. Instruction memory module with latency

> Directory: `05_11_cpu_fetch_with_latency`

Task: Modify the processor to work with instruction memory which has a read latency of 1 clock cycle.

You can assume either non-pipelined or pipelined memory. In the first case you just need to track whether instruction memory data is valid for the current clock cycle. In the second case you can try to implement a pipelined CPU design with a rudimentary branch prediction ("predict no branch").

Exercise 10. Three processors and an arbiter

> Directory: `05_12_three_cpus_sharing_instr_memory`

Examine the arbiter in file `round_robin_arbiter_8.sv`

Exercise: Implement a processor cluster in `cpu_cluster.sv` file consisting of 3 instances of schoolRISCV core sharing one instruction memory.

Advanced variant: Implement instruction memory using so-called banks, which would provide simultaneous access to multiple parts of address space in most cases (when there's no "bank conflict").