Домашнее задание №3

Домашнее задание №3 состоит из 7 упражнений:

- Первое в папке `03_01_signed_add_with_overflow`
- Второе в папке `03_02_signed_add_with_saturation`
- Третье в папке `03_03_signed_or_unsigned_mul`
- Четвёртое в папке `03_04_four_ways_of_doing_shift`
- Пятое в папке `03_05_circular_shifts`
- Шестое в папке `03_06_arithmetic_shift_or_signed_divide_by_power_of_2`
- Седьмое в папке `03_07_sort_floats`

В большинстве упражнений есть секция `Example` с модулем для примера,
и секция `Task` с описанием задания и местом, где необходимо описать ваше решение.

Предисловие

В процессе работы с упражнениями 1-6, возможно запустить проверку решения только одного задания с помощью команды
`iverilog -g2005-sv *.sv && vvp a.out` в папке задания.

В файле Testbench любого из заданий можно убрать комментарий у строк `$dumpfile;` и `$dumpvars;` для генерации `dump.vcd` файла. В файле будут содержаться текстовые описания временной диаграммы, описывающей изменения на всех проводах и регистрах во время симуляции.

Можно воспользоваться командой `gtkwave dump.vcd` для просмотра файла, либо добавить опцию `--wave` или `-w` к скрипту `run_`.
Так же, возможно использовать более современную программу Surfer для просмотра временных диаграмм.
Surfer доступен на системах Linux, Windows и macOS, а так же в качестве расширения редактора VS Code.

Упражнение 1. Знаковое сложение с переполнением

Задание:

Реализуйте модуль, который складывает два числа со знаком и детектирует переполнение.
Под "знаковым числом" мы подразумеваем числа представленные в "дополнительном коде" (two's complement).

Выходной бит "переполнения" должен быть равен 1, если сумма двух входных аргументов превышает максимальное положительное или отрицательное знаковое число в четырёхбитной кодировке.
В противном случае значение "переполнения" должно быть 0.

Упражнение 2. Знаковое сложение с насыщением

Задание:

Реализуйте модуль, который складывает два знаковых числа с насыщением.
"Сложение с насыщением" означает:

- Если результат не помещается в 4 бита, а аргументы положительные, результирующая сумма должна стать максимальным положительным числом.
- Если результат не помещается в 4 бита, а аргументы отрицательные, результирующая сумма должна стать минимальным отрицательным числом.

Упражнение 3. Знаковое и беззнаковое умножение

Задание:

Реализуйте параметризованный модуль, который выдает результат умножения
с учётом знаков или без учёта знаков в зависимости от входного бита 'signed_mul'.

Упражнение 4. Четыре способа побитового сдвига

Задание:

Реализуйте параметризованный модуль, который сдвигает беззнаковое входное число на `S` бит вправо
используя четыре разных способа: логический сдвиг вправо, конкатенацию, цикл `for` внутри `always_comb`,
и цикл `for` внутри `generate`.

Упражнение 5. Циклический побитовый сдвиг

Задание:

Реализуйте модуль, который сдвигает входящие `S` битов вправо циклически, по кругу, используя только оператор побитовой конкатенации (фигурные скобки `{` и `}`) и срезы (slice, оператор квадратные скобки `[` и `]`).

"Циклически" означает `ABCDEFGH` -> `FGHABCDE`, когда `N = 8` и `S = 3`.

Во второй подзадаче используйте только следующие операции:
логический сдвиг вправо (`>>`), логический сдвиг влево (`<<`), "или" (`|`) и константы.

Упражнение 6. Арифметический сдвиг или деление на степень двойки

Задание:

Реализуйте арифметический сдвиг вправо тремя разными способами.
Арифметический сдвиг вправо (`>>>`) отличается от логического сдвига (`>>`) заполнением начальных бит правильным значением, в зависимости от знака.

Например:
`-4` равно `8'b11111100` в дополнительном коде, а значит `-4 >>> 2` будет равно `-1` = `8'b11111111`.

Предисловие к упражнению 7

Для успешного выполнения упражнения, необходимо на базовом уровне ознакомиться с представлением вещественных чисел (floating-point numbers) в компьютерах и в двоичном формате. Упражнение основывается на стандарте IEEE 754.

В данном упражнении для работы с вещественными числами используется блок (FPU) из открытого процессора CORE-V Wally. Данный процессор основан на стандарте RISC-V и разрабатывается группой исследователей во главе с Дэвидом Харрисом.

Для упрощения работы с вещественными числами, блок FPU из процессора обёрнут в более простые модули обёртки. Каждый модуль-обёртка специализирован для выполнения одной конкретной операции. К примеру, модуль `f_less_or_equal` вычисляет, является ли первое число меньше или равно второму, а модули `f_add` и `f_sub` выполняют операции сложения и вычитания двух вещественных чисел соответственно.

Все модули-обёртки находятся в папке `common/wally_fpu`. Исходные коды самого процессора находятся в папке `import/preprocessed/cvw` и, при отсутствии, должны быть импортированы через запуск скрипта `run_linux_mac.sh`.

Константа `FLEN` объявляется в файле `import/preprocessed/cvw/config-shared.sv` и обозначает длину вещественного числа в битах. В текущем упражнении длина всех вещественных чисел подразумевает 64 бита, однако в целях совместимости настоятельно рекомендуется использовать константу `FLEN` вместо численного указания длины.

Константа `NE` (Number of Exponent bits) и константа `NF` (Number of Fraction bits) обозначают количество бит используемое для хранения показателя степени и дробной части соответственно. Так же, первый бит вещественного числа обозначает знак (Sign).

Упражнение 7. Комбинационная сортировка трёх вещественных чисел

Необходимо ознакомиться с примерами сортировки двух чисел `a` и `b` отдельно, а так же с сортировкой массива `unsorted` из двух элементов.

Задание:

В файле `03_07_sort_floats.sv`, имплементировать модуль для сортировки трёх вещественных чисел с использованием нескольких модулей `f_less_or_equal`.
Решение должно быть комбинационным. При обработке входящих чисел, модуль должен выставлять флаг `err` в логическую единицу, если любой из внутренних модулей `f_less_or_equal` детектирует числа `NaN`, `+Inf` или `-Inf` и выставляет флаг `err`.