

Практическое Задание №3

Практическое задание №3 состоит из 8 упражнений:

- Первое в папке `03_01_detect_sequence_using_fsm`
- Второе в папке `03_02_detect_sequence_using_shift_reg`
- Третье в папке `03_03_serial_divisibility_using_fsm`
- Четвертое и пятое в папке `03_04_05_sqrt_formula_fsms`
- Шестое в папке `03_06_sort_floats`
- Седьмое в папке `03_07_sort_floats_using_fsm`
- Восьмое в папке `03_08_float_discriminant`

В большинстве упражнений есть секция `Example` с модулем для примера, секция `Task` с описанием задания и местом, где необходимо описать ваше решение, а так же файл `Testbench`, который осуществляет минимальную проверку работоспособности вашего решения.

Предисловие

В процессе работы с упражнениями 1-3, возможно запускать проверку вашего решения отдельно 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 при запуске. В файле будет содержаться текстовое описание временной диаграммы, описывающей изменения на всех проводах и регистрах во время симуляции работы модуля.

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. Распознавание бинарной последовательности с помощью FSM

Ознакомиться с примером детектирования 4-х битной последовательности.

Задание: Реализовать модуль для детектирования 6-ти битной последовательности `110011` используя конечный автомат.

Упражнение 2. Распознавание бинарной последовательности с помощью Shift Register

Ознакомиться с примером детектирования 4-х битной последовательности.

Задание: Реализовать модуль для детектирования 6-ти битной последовательности `110011` используя сдвиговый регистр.

Упражнение 3. Последовательная проверка делимости числа

Ознакомиться с примером детектирования делимости числа на 3.

Ниже приведён пример работы модуля, вывод и внутреннее состояние в процессе. На вход модуля подаётся лишь самый правый бит:

binary number	Div by 3	State
0	yes	mod_0
01	no	mod_1
011	yes	mod_0
0110	yes	mod_0
01101	no	mod_1
011010	no	mod_2
0110100	no	mod_1
01101001	yes	mod_0

Задание: Реализовать модуль последовательного детектирования делимости числа на 5 используя конечный автомат.

Упражнения 4 и 5. Вычисление формулы с помощью КА

Введение

Директория `03_04_05_sqrt_formula_fsms` содержит примеры, тестбенчи, заготовки решений и вспомогательный код для 4-го и 5-го упражнений.

Для выполнения упражнений, необходимо использовать готовый модуль `isqrt.sv` в качестве чёрного ящика и написать FSM для вычисления двух формул.

Модуль `isqrt.sv` вычисляет целочисленный квадратный корень (integer square root) с фиксированной латентностью (временем в тактах между поступлением аргумента на вход и получением результата на выходе).

Модуль начинает вычисление при выставлении сигнала `x_vld`, и сообщает о готовности (валидности) результата выставляя сигнал `y_vld`.

Модуль `isqrt` находится в директории `common/isqrt/`:

```
common/black_boxes // Готовый модуль isqrt
├─ isqrt.sv
├─ isqrt_slice_comb.sv
└─ isqrt_slice_reg.sv
```

Структура директории упражнений:

```
├─ testbenches
│   └─ formula_tb.sv // Основной код тестбенча
│   └─ isqrt_fn.svh  // Математическая формула isqrt для верификации
│   └─ tb.sv         // Запуск трёх тестбенчей для разных формул
├─ formula_1_fn.svh  // Эталонная формула 1 (используется для верификации)
└─ formula_1_impl_1_fsm.sv // Пример реализации формулы 1
```

```
|— formula_1_impl_1_fsm_style_2.sv // Альтернативная реализация формулы 1
|— formula_1_impl_1_top.sv
|— 03_04_formula_1_impl_2_fsm.sv // файл с упражнением 4
|— formula_1_impl_2_top.sv
|— formula_2_fn.svh // Эталонная формула 2 (используется для верификации)
|— 03_05_formula_2_fsm.sv // файл с упражнением 5
|— formula_2_top.sv
|— run_all_using_iverilog_under_linux_or_macos_brew.sh
|— run_all_using_iverilog_under_windows.bat
```

Замечание: Создавать инстансы модуля `isqrt` самостоятельно запрещается.

Необходимо работать с модулем через входы и выходы `isqrt_x` и `isqrt_y` модуля упражнения.

Упражнение 4

Ознакомиться с формулой в файле `formula_1_fn.svh` и примером конечного автомата для последовательного вычисления этой формулы в файле `formula_1_impl_1_fsm.sv` или `formula_1_impl_1_fsm_style_2.sv`.

Задание:

В файле `formula_1_impl_2_fsm.sv`, имплементировать вычисление Формулы 1 используя два модуля `isqrt` одновременно. Вычислить два из трёх значений параллельно. Далее, вычислить оставшееся значение и предоставить результат суммы.

Упражнение 5

Ознакомиться с формулой в файле `formula_2_fn.svh`.

Задание:

В файле `formula_2_fsm.sv`, имплементировать последовательное вычисление Формулы 2 используя один модуль `isqrt`.

Упражнения 6, 7 и 8. Вещественные числа

Введение

Для успешного выполнения упражнений, неизбежно на базовом уровне ознакомиться с представлением вещественных чисел (floating-point numbers) в компьютерах и в двоичном формате. Упражнения основываются на стандарте IEEE 754.

В данной группе упражнений 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 обернут 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. Во всех упражнениях данного практического задания, длина 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 6. Комбинационная сортировка вещественных чисел

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

Задание:

В файле `03_06_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`.

Упражнение 7. Сортировка вещественных чисел с помощью КА

Задание:

В файле `03_07_sort_floats_using_fsm.sv`, имплементировать модуль для сортировки трёх вещественных чисел с использованием FSM и внешнего модуля сравнения двух вещественных чисел.

В данном задании **запрещается** создание любых инстансов модулей. Необходимо использовать

сигналы `f_le_a`, `f_le_b`, `f_le_res`, `f_le_err` для общения с внешним комбинационным модулем.

Общая латентность модуля (от момента выставления `valid_in` до момента выставления `valid_out`) не должна превышать 10 тактов.

При обнаружении ошибки сравнения чисел, то есть `f_le_err` равно логической единице, необходимо прервать работу FSM и в текущем, либо следующем такте выставить сигнал `valid_out` вместе с сигналом об ошибке `err`. В этом случае значения вывода `sorted` могут быть произвольными и будут игнорироваться тестирующим окружением.

Упражнение 8. Вычисление вещественного дискриминанта

Ознакомиться с модулями-обёртками для умножения (`f_mult`), сложения (`f_add`) и вычитания (`f_sub`) вещественных чисел.

Задание:

В файле `03_08_float_discriminant.sv`, имплементировать модуль для вычисления дискриминанта квадратного уравнения. Вычисление должно использовать общепринятую формулу $D = b^2 - 4ac$.

При обработке входящих чисел, модуль должен выставить флаг `err` в логическую единицу, если любой из внутренних инстансов модулей детектирует числа `NaN`, `+Inf` или `-Inf` и выставляет флаг `err`. В этом случае значения вывода `res` может быть произвольными и будет игнорироваться тестирующим окружением.

В качестве константного вещественного числа 4 вы можете использовать следующее объявление:

```
localparam [FLEN - 1:0] four = 64'h4010_0000_0000_0000;
```