Лабораторная работа №2.

Язык Verilog/SystemVerilog.
Необходимо выполнить все задания.
Задания можно выполнять и сдавать в любом порядке.

Для каждого задания необходимо:

- Написать синтезируемый модуль, который решает поставленную задачу.
- Написать модуль-тестбенч (testbench) к этому модулю, используя несинтезируемые конструкции.
- Провести симуляцию в ModelSim (или аналогичном симуляторе). Убедиться, что модуль корректно работает.
- Сделать проект в Quartus'е. Собрать синтезируемый модуль, убедится, что модуль собирается, и не возникает латчей, комбинационных петель и пр. Симуляцию в Quartus'е (как в первой лабе) проводить не надо.

Если по уровню сложности, то они располагаются примерно так (по возрастанию):
A6, A5, A7, A2, A3, A1, A4.

Считаем, что все входные сигналы в заданиях A1-A4 синхронны с синхроимпульсом (clk). Никакой дополнительной пересинхронизации делать не надо.

A1. Часы

Необходимо сделать эмуляцию часов: на выход модуля выдается «текущее» время: часы, минуты, секунды, милисекунды. Синхроимпульс, который заходит в модуль, имеет частоту 50 МГц. Необходимо предусмотреть возможность сброса времени в ноль, а так же предустановку какого-то из значений.

A2. Сериализатор

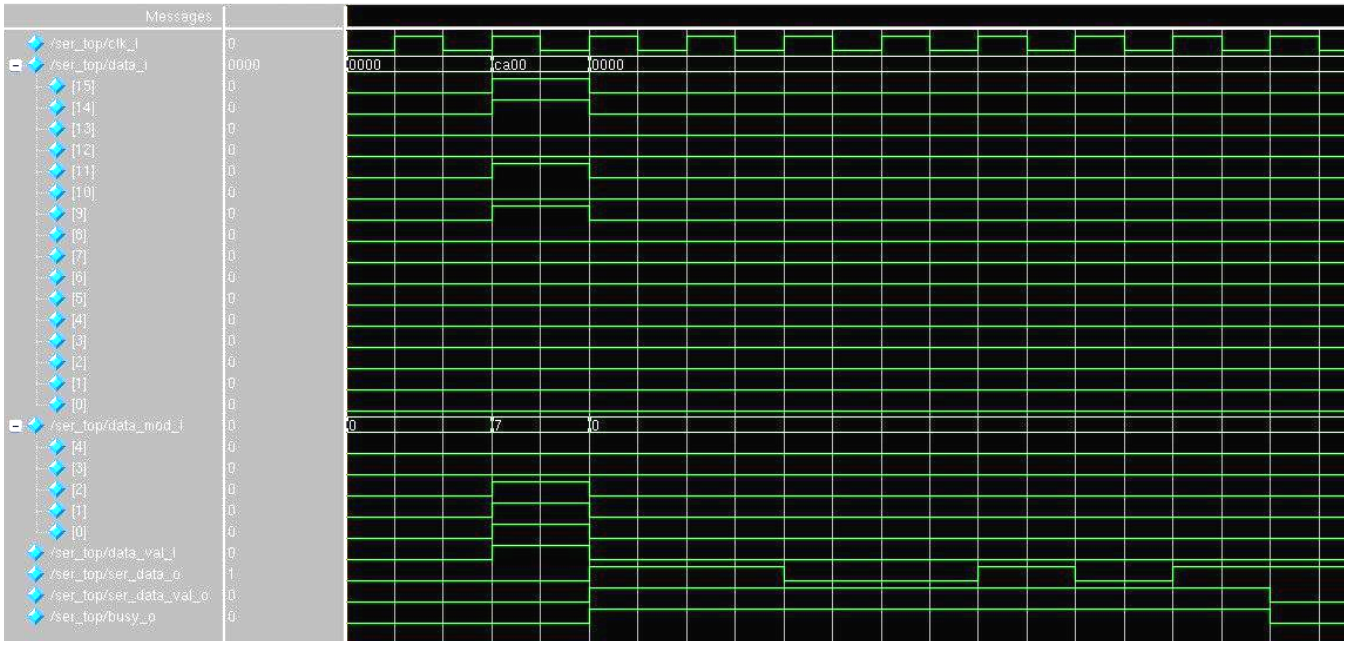
Сериализатор на вход получает «широкие» данные — длинное слово, и отправляет по одному биту эти данные на выход.

Интерфейс модуля:

clk_i	Синхроимпульс

rst_i	Асинхронный сброс
data_i[15:0]	Входные данные
data_mod_i[4:0]	Число показывает сколько бит в входных данных валидно. К примеру, если там стоит 5, то валидны биты [15:11], и значит только пять бит будет «отправлено» дальше. Минимальное валидное число равно 3. Если mod меньше, то эта посылка игнорируется.
data_val_i	Сигнал, подтверждающий, что data_i и data_mod_i валидны, держится один такт.
ser_data_o	«Сериализованные» данные.Будем считать, что первым битом выходит самый старший бит из data_i.
ser_data_val_o	Подтверждает валидность ser_data_o.
busy_o	Если 1, то это говорит о том,что модуль сериализации занят, и вышестоящий модуль не пошлет новые data_i данные в тот такт, когда висит busy_o.

Времянки его работы:



A3. Десериализатор

Десериализатор выполняет функцию обратную сериализатору. Интерфейс модуля «похож», поэтому я его приводить не буду.

A4. Светофор

Требуется написать модуль для управления светофором, который будет зажигать красный, желтый и зеленый фонари во всем известной последовательности: красный, красный и желтый, зеленый, зеленый моргает, желтый, красный.

Интерфейс модуля:

clk_i	Синхроимпульс
rst_i	Асинхронный сброс
red_o	Если 1, то горит соответствующий цвет, если 0 — то он не горит
yellow_o	
green_o	

Параметры, задающие время горения фонарей светофора и период мигания зеленого фонаря являются параметрами модуля. Время задается в количестве тактов синхросигнала clk_i. **Внимание**: необходимо использовать конечный автомат для решения этого задания.

A5. Крайние единицы

Модуль должен в входном N-битном числе найти самую правую и самую левую единицу и оставить только ее.

Пример:
На входе: → На выходе
5'b00110 → 5'b00100; 5'b00010
5'b11111 → 5'b10000; 5'b00001
5'b00000 → 5'b00000; 5'b00000

A6. Количество единиц и нулей

Модуль должен в входном N-битном числе посчитать количество единиц и нулей.

A7. Антидребезг

Решить проблему дребезга кнопки, который может происходить при нажатии на нее.

Интерфейс модуля:

clk_i	Синхроимпульс
key_i	Сигнал с кнопки. 0 - кнопка нажата, 1 - не нажата.
key_pressed_stb_o	Строб (импульс) на один такт, сообщающий, что кнопка была нажата