МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»
(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова»)

БГТУ.СМК-Ф-4.2-К5-01

Факультет И (шифр) Информационные и управляющие системы (наименование)

Кафедра И4 (шифр) Радиоэлектронные системы управления (наименование)

Дисциплина Проектирование цифровых систем на ПЛИС

КУРСОВАЯ РАБОТА

на тему

Проектирование гирлянды на ПЛИС

Выполнил студент
группы И4М31 _____
Гаврилова Ю. И.
Фамилия И.О.

РУКОВОДИТЕЛЬ

_____ _____
Фамилия И.О. Подпись

Оценка _____

«_____» _____ 2018г.

САНКТ-ПЕТЕРБУРГ

2018 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

По мере развития цифровых микросхем возникло противоречие между возможной степенью интеграции и номенклатурой выпускаемых микросхем. Экономически оправдано было выпускать микросхемы средней интеграции, таких как регистры, счетчики, сумматоры. Более сложные схемы приходилось создавать из этих узлов. Разместить более сложную схему на полупроводниковом кристалле не было проблем, но это было оправдано либо очень большой серийностью аппаратуры, либо ценой аппаратуры (военная, авиационная или космическая). Заказные микросхемы не могли удовлетворить возникшую потребность в миниатюризации аппаратуры. Решение могло быть только одним — предоставить разработчикам аппаратуры возможность изменять внутреннюю структуру микросхемы (программировать).

История развития программируемых логических интегральных схем (ПЛИС) начинается с появления программируемых постоянных запоминающих устройств. Первое время программируемые ПЗУ использовались исключительно для хранения данных, однако вскоре их стали применять для реализации цифровых комбинационных устройств с произвольной таблицей истинности. В качестве недостатка подобного решения следует отметить экспоненциальный рост сложности устройства в зависимости от количества входов. Это не позволяет реализовать многовходовые комбинационные цифровые схемы.

Для реализации цифровых комбинационных устройств с большим числом входов были разработаны программируемые логические матрицы. В иностранной литературе они получили название. Именно программируемые логические матрицы можно считать первыми программируемыми логическими интегральными схемами. Программируемые логические матрицы получили широкое распространение в качестве первых универсальных микросхем большой интеграции.

Целью курсовой работы является проектирование цифровой системы на ПЛИС.

ОБЩИЕ СВЕДЕНИЯ О ПЛИС

Программируемая логическая интегральная схема— электронный компонент (интегральная микросхема), используемый для создания конфигурируемых цифровых электронных схем. В отличие от обычных цифровых микросхем, логика работы ПЛИС не определяется при изготовлении, а задаётся посредством программирования. Для программирования используются программатор и IDE (отладочная среда), позволяющие задать желаемую структуру цифрового устройства в виде принципиальной электрической схемы или программы на специальных языках описания аппаратуры: Verilog, VHDL, AHDL и др.

Типы ПЛИС

Ранние ПЛИС

В 1970 году компания «TI» разработала маскируемые (то есть, программируемые с помощью маски) интегральные схемы, основанные на ассоциативном ПЗУ фирмы «IBM». Эта микросхема называлась TMS2000 и программировалась чередованием металлических слоёв в процессе производства интегральной схемы. TMS2000 имела до 17 входов и 18 выходов с 8 JK-триггерами в качестве памяти. Для этих устройств компания «TI» ввела термин PLA — программируемая логическая матрица.

PAL

PAL — программируемый массив (матрица) логики. В СССР PLA и PLM не различались и обозначились как ПЛМ (программируемая логическая матрица). Разница между PLA и PLM состоит в доступности программирования внутренней структуры (матриц).

GAL

GAL — это ПЛИС, имеющие программируемую матрицу «И» и фиксированную матрицу «ИЛИ».

CPLD

CPLD (сложные программируемые логические устройства) содержат относительно крупные программируемые логические блоки — макроячейки, соединённые с внешними выводами и внутренними шинами. Функциональность CPLD кодируется в энергонезависимой памяти, поэтому нет необходимости их перепрограммировать при включении. Может применяться для расширения числа входов/выходов рядом с большими кристаллами, или для предобработки сигналов (например, контроллер COM-порта, USB, VGA).

FPGA

FPGA содержат блоки умножения-суммирования, которые широко применяются при обработке сигналов, а также логические элементы (как правило, на базе таблиц перекодировки — таблиц истинности) и их блоки коммутации. FPGA обычно используются для обработки сигналов, имеют больше логических элементов и более гибкую архитектуру, чем CPLD. Программа для FPGA хранится в распределённой памяти, которая может быть выполнена как на основе энергозависимых ячеек статического ОЗУ — в этом случае программа не сохраняется при исчезновении электропитания микросхемы, так и на основе энергонезависимых ячеек flash-памяти или перемычек antifuse — в этих случаях программа сохраняется при исчезновении электропитания. В первом случае при каждом включении питания микросхемы необходимо заново конфигурировать её при помощи начального загрузчика, который может быть встроен и в саму FPGA.

Некоторые ведущие мировые производители ПЛИС

- Achronix
- Actel
- Altera
- Atmel
- Lattice semiconductor
- Xilinx

ПЛИС ФИРМЫ ALTERA СЕМЕЙСТВА FPGA

В курсовой работе использовалась ПЛИС фирмы Altera Cyclone V SoC dual -core ARM Cortex-A9 5CSEMA5F31C6N.

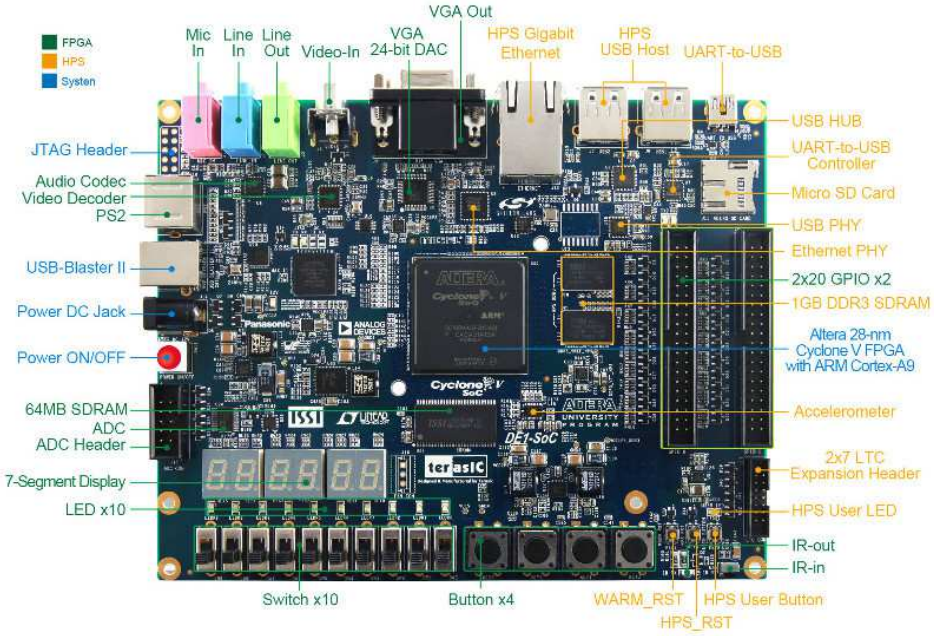

Рисунок 1 – ПЛИС Altera Cyclone V SoC 5CSEMA5F31C6N

Структурная схема платы показана на рисунке 2.

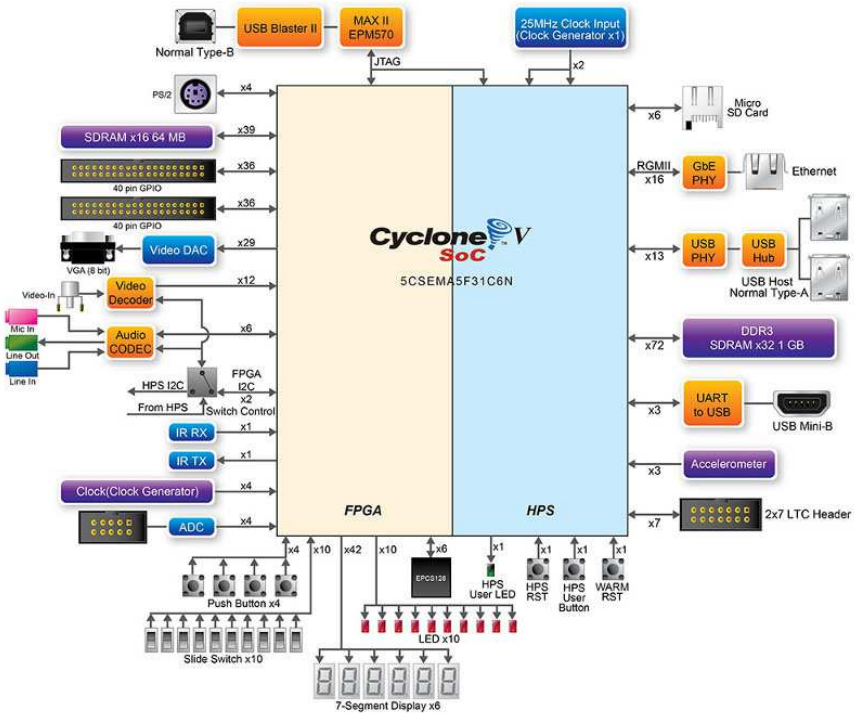

Рисунок 2 – Структурная схема платы Altera Cyclone V SoC 5CSEMA5F31C6N

Характеристики платы показаны в таблице 1.

Таблица 1 –Характеристики платы

RAM	1GB DDR3 SDRAM (HPS) + 64MB SDRAM (FPGA)
Ethernet	10/100/1000
Размер	354 x 130 мм
Другие возможности	24-bit VGA DAC Audio 24-bit CODEC TV decoder (NTSC/PAL/SECAM) ADC 500 KSPS x 12 bits x 8 каналов

ЯЗЫК ПРОГРАММИРОВАНИЯ VERILOG HDL

Для описания и моделирования электронных систем используются различные языки описания аппаратуры. В данной курсовой работе использовался язык Verilog HDL.

Verilog имеет препроцессор, очень похожий на препроцессор языка C, и основные управляющие конструкции «if», «while» также подобны одноимённым конструкциям языка C. Все вышеперечисленное упрощает усвоение языка.

Описание аппаратуры, написанное на языке Verilog принято называть программами, но в отличие от общепринятого понятия программы как последовательности инструкций, здесь программа задает структуру системы.

История создания

Verilog был создан в 1983—1984 годах в фирме Automated Integrated Design Systems (с 1985 года Gateway Design Automation) как язык моделирования аппаратуры. В 1990 году Gateway Design Automation была куплена Cadence Design Systems. Компания Cadence имеет права на логические симуляторы Gateway’s Verilog и Verilog-XL simulator.

Verilog-95

Во время увеличивающейся популярности языка VHDL, Cadence приняла решение добиться стандартизации языка. Cadence передала Verilog в общественное достояние. Verilog был послан в IEEE и принят как стандарт IEEE 1364—1995 (часто называемый Verilog-95).

Verilog 2001

Verilog-2001 является значительно обновленным по сравнению с Verilog-95. Во-первых, он добавил поддержку знаковых переменных (в формате дополнительного кода). Прежде авторам кода приходилось реализовывать знаковые операции с использованием большого количества битовых логических операций. Та же функциональность на Verilog-2001 описывается встроенными операторами языка. Был улучшен файловый ввод-вывод. Для улучшения читаемости кодов был немного изменен синтаксис, например always @*, переопределение именованных параметров, объявление заголовков функций, задач и модулей в стиле Си.

Verilog-2001 является самым часто используемым диалектом языка и поддерживается в большинстве коммерческих САПР для электроники.

Verilog 2005

Verilog 2005 (стандарт IEEE 1364—2005) добавил небольшие исправления, уточнения спецификаций и несколько новых синтаксических конструкций, например, ключевое слово uwire.

Отдельная от стандарта часть, Verilog-AMS, позволяет моделировать аналоговые и аналого-цифровые устройства.

SystemVerilog

SystemVerilog является надмножеством Verilog-2005, с многими новыми возможностями для верификации и моделирования разработок.

Конструкции языка

Типы данных

Verilog содержит два базовых типа данных: wire и reg. Оба эти типа могут принимать 4 возможные значения при симуляции Verilog программы:0,1,X («неизвестное значение» - это значение используется только для симуляции, в реальной аппаратуре будет 0 или 1.) и Z («состояние высокого сопротивления», то есть отсутствие сигнала.)

Тип wire используется для описания цепей, reg для регистров и переменных. Оба эти типа могут также быть использованы при описании многобитовых данных:

Переменные типа reg имеют начальное значение 'X'. Цепи передают значения между регистрами. Если цепь не присоединена ни к какому регистру, она будет иметь значение 'Z'.

Verilog также содержит массивы, которые позволяют моделировать память:

Кроме этого, Verilog содержит еще следующие типы данных:

- integer — то же самое, что «reg[31:0]», при этом в операциях учитывается знак (старший бит)
- real
- time
- realtime

Список приложений, поддерживающих Verilog

- Quartus II -среда моделирования и отладки;
- icarus verilog — приложение для моделирования и синтеза;
- vcs - среда моделирования и отладки;
- logicsim - среда моделирования и отладки;
- incisive hdl - среда моделирования и отладки;
- modelsim - среда моделирования и отладки;
- veritak - редактор, интегрированный компилятор/симулятор, транслятор с vhdl в verilog;
- verilator - open-source высокопроизводительный компилятор verilog.
- verilog-perl - набор perl-модулей для предобработки и построения других инструментов.
- и другие.

В курсовой работе использовалась среда моделирования и отладки Quartus II.

ГИРЛЯНДА НА ПЛИС

В работе гирлянды будет использоваться 4 разных режима переключения светодиодов.

В имеющейся плате есть встроенный генератор, который будет необходим при задании счетчика, а так же 8 светодиодов.

В первую очередь необходимо задать входные и выходные параметры:

```
module k1(
input wire clk,     //Вход тактового генератора
output wire [7:0] leds   //8 выходов на светодиоды
);
```

Далее необходимо реализовать счетчик:

```
reg [31:0] cnt; initial cnt <= 32'd0;          //Счетчик, задаем первоначальное значение
always @(posedge clk) cnt <= cnt + 1'b1;  //Увеличение счетчика с каждым тактом
```

Далее будет реализовываться логика управления светодиодами. Рассмотрим 4 варианта переключения: 1) последовательное чередование, 2) бегущий огонек, 3) последовательное чередование двух светодиодов, 4) двойные бегущие огоньки.

1) **Последовательное чередование.** В данном случае последовательно будут включаться все четные и нечетные светодиоды:

```
wire [7:0] var1=(cnt[24]) ? 8'b01010101 : 8'b10101010;
```

2) **«Бегущий» огонек.** Здесь будет реализовано последовательное переключение светодиодов друг за другом:

```
wire [7:0] var2 =
        (cnt[24:22] == 3'd0) ? 8'b11111110 :
        (cnt[24:22] == 3'd1) ? 8'b11111101 :
        (cnt[24:22] == 3'd2) ? 8'b11111011 :
        (cnt[24:22] == 3'd3) ? 8'b11110111 :
        (cnt[24:22] == 3'd4) ? 8'b11101111 :
        (cnt[24:22] == 3'd5) ? 8'b11011111 :
        (cnt[24:22] == 3'd6) ? 8'b10111111 :
        (cnt[24:22] == 3'd7) ? 8'b01111111 : 8'b11111111;
```

3) **Последовательное чередование двух светодиодов.** Повторяется тот же алгоритм, что и в первом случае, но переключаются два соседних светодиода:

```
wire [7:0] var3 = (cnt[24]) ? 8'b00110011 : 8'b11001100;
```

4) **«Двойные» бегущие огоньки.** Повторяется второй случай, но «двигаются» два соседних светодиода:

```
wire [7:0] var4 =
        (cnt[24:23] == 2'd0) ? 8'b11001100 :
        (cnt[24:23] == 2'd1) ? 8'b01100110 :
        (cnt[24:23] == 2'd2) ? 8'b00110011 :
        (cnt[24:23] == 2'd3) ? 8'b10011001 : 8'b11111111;
```

Далее реализован вывод вышеперечисленного алгоритма на реальную плату:

```
always @(posedge clk)
begin
    if ( but[0] == 0 ) begin
        leds = (cnt[29:27] == 3'd1) ? var1 : var1;
    end
    if ( but[1] == 0 ) begin
        leds = (cnt[29:27] == 3'd1) ? var2 : var2;
    end
    if (but[2] == 0 ) begin
        leds = (cnt[29:27] == 3'd1) ? var3 : var3;
    end
    if ( but[3] == 0 ) begin
        leds = (cnt[29:27] == 3'd1) ? var4 : var4;
    end
end
```

В Приложении А представлен полный код данного алгоритма.

ЗАКЛЮЧЕНИЕ

В данной работе были рассмотрены ПЛИС на архитектуре FPGA, была описана архитектура ПЛИС, так же основные отличительные особенности. Одним из основных производителей ПЛИС на данной архитектуре является фирма ALTERA. Так же был рассмотрен язык описания аппаратуры: Verilog и спроектирована цифровая система на этом языке.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) http://www.altera.com

2) https://www.terasic.com.tw/en/ – Terasic Inc.

3) В.В. Соловьев Основы языка проектирования цифровой аппаратуры

4) Максфилд Клайв – Проектирование на ПЛИС. Архитектура, средства и методы. Курс молодого бойца

5) Соловьев В., Климович А. «Введение в проектирование комбинационных схем на ПЛИС

6) http://fpga.in.ua/ – Проектирование ПЛИС в Altera Quartus II

7) Стешенко В. «Школа разработки аппаратуры цифровой обработки сигналов на ПЛИС

8) https://parallel.ru – Основные производители современных ПЛИС-компьютеров и комплектующих к ним – Лаборатория Параллельных информационных технологий НИВЦ МГУ

ПРИЛОЖЕНИЕ А

```
module k1(
input wire [3:0] but,
input wire clk,
output reg [7:0] leds
);
reg [31:0] cnt;
initial cnt <= 32'd0;
always @(posedge clk) cnt <= cnt + 1'b1;
wire [7:0] var1=(cnt[24]) ? 8'b01010101 : 8'b10101010;
wire [7:0] var2 =
                (cnt[24:22] == 3'd0) ? 8'b11111110 :
                (cnt[24:22] == 3'd1) ? 8'b11111101 :
                (cnt[24:22] == 3'd2) ? 8'b11111011 :
                (cnt[24:22] == 3'd3) ? 8'b11110111 :
                (cnt[24:22] == 3'd4) ? 8'b11101111 :
                (cnt[24:22] == 3'd5) ? 8'b11011111 :
                (cnt[24:22] == 3'd6) ? 8'b10111111 :
                (cnt[24:22] == 3'd7) ? 8'b01111111 : 8'b11111111;
wire [7:0] var3 = (cnt[24]) ? 8'b00110011 : 8'b11001100;
wire [7:0] var4 =
                (cnt[24:23] == 2'd0) ? 8'b11001100 :
                (cnt[24:23] == 2'd1) ? 8'b01100110 :
                (cnt[24:23] == 2'd2) ? 8'b00110011 :
                (cnt[24:23] == 2'd3) ? 8'b10011001 : 8'b11111111;
wire [7:0] var5 = 8'b11111111;
always @(posedge clk)
begin
if ( but[0] == 0 ) begin
```

```
leds = (cnt[29:27] == 3'd1) ? var1 : var1;
end
if ( but[1] == 0 ) begin
leds = (cnt[29:27] == 3'd1) ? var2 : var2;
end
if (but[2] == 0 ) begin
leds = (cnt[29:27] == 3'd1) ? var3 : var3;
end
if ( but[3] == 0 ) begin
leds = (cnt[29:27] == 3'd1) ? var4 : var4;
end
end
endmodule
```