**Проектирование систем на кристалле с программируемой архитектурой**

**Лабораторная работа №2**

Цели работы:

1. Изучить особенности описания modport интерфейсов в HDL SystemVerilog;
2. Изучить дополнительные конструкции HDL SystemVerilog (package, typedef, enum);
3. Изучить алгоритм работы последовательного CRC-16;
4. Дополнить HDL модуль из Л.Р. №1 регистровой картой APB3 с использованием изученных доп. конструкций HDL SystemVerilog;
5. Имплементировать расчёт CRC-16 в соответствии с вариантом в модуль из HDL Л.Р. №1.

**Теоретическая часть**

**Modport**

Modport – конструкция языка описания аппаратуры SystemVerilog, описывающая направления сигналов внутри интерфейса в зависимости от типа используемого интерфейса (Master, Slave или др.). Данная конструкция необходима для явного определения направления сигналов для модуля, использующего интерфейс, поскольку все сигналы, находящиеся внутри SV-интерфейса без modport по-стандарту являются типом inout.

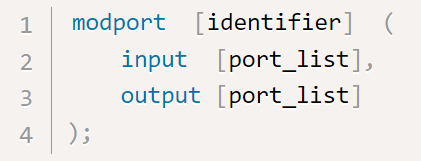


Рисунок 1 – Синтаксис modport (описывается внутри необходимого интерфейса)

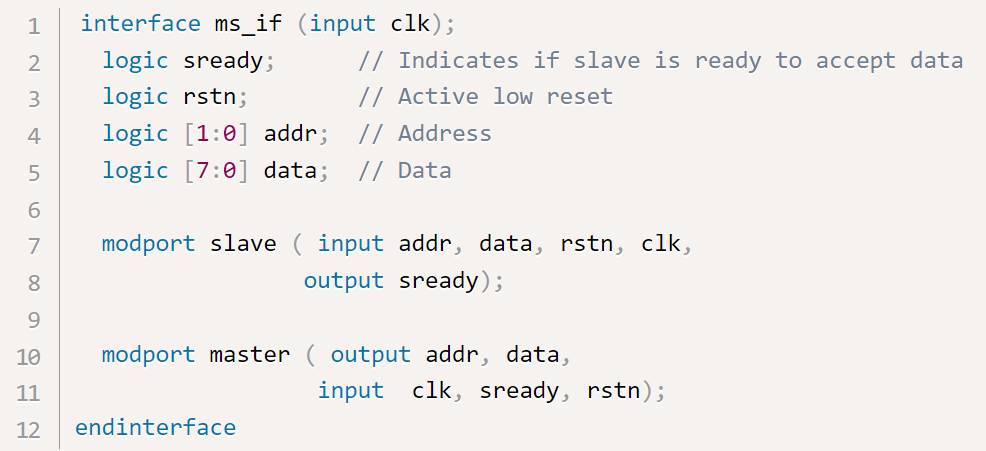


Рисунок 2 – Пример определения интерфейса с использованием modport

Для назначения конкретного modport инстанцируемого модуля используется следующая конструкция (рисунок 3).

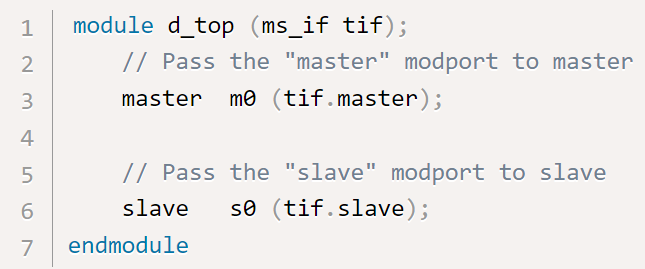


Рисунок 3 – Назначение modport интерфейса для подключаемого модуля

**Struct, typedef**

Конструкция struct в SystemVerilog позволяет нам создать группу из нескольких переменных (даже разного типа). На всю группу можно ссылаться как на одно целое, либо на отдельные её части можно ссылаться по имени, используя «иерархическое» обращение (как и в интерфейсах). Это очень удобно при написании RTL-кода, когда у вас есть набор сигналов, которые вам нужно передать по всему дизайну вместе, но вы хотите сохранить читаемость и доступность каждого отдельного сигнала.

Существуют структуры типа packed и unpacked. При объявлении структуры без присваивания конкретного типа она по умолчанию будет представлена как unpacked. Синтезируемой конструкцией является только лишь структура типа packed!

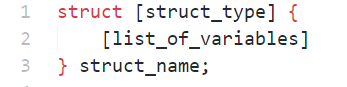


Рисунок 4 – Синтаксис конструкции struct

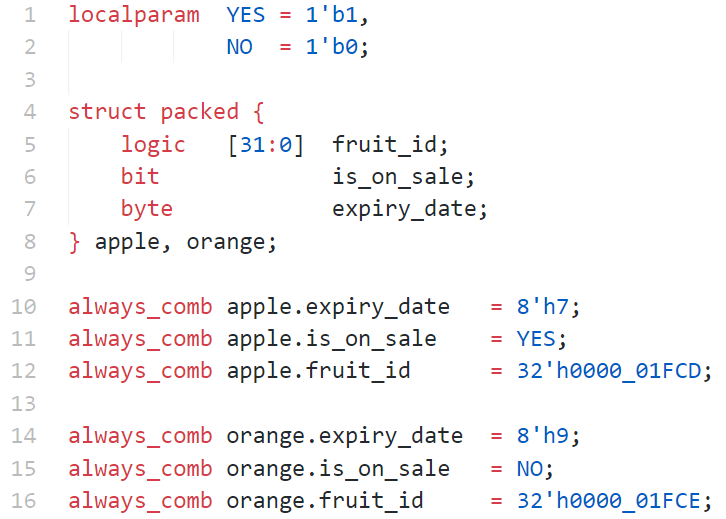


Рисунок 5 – Пример объявления структуры и использования её параметров.

В языке SystemVerilog можно создавать новые типы данных. Для этого используется конструкция typedef. В большинстве случаев мы просто используем typedef для присвоения имени объявлению типа, которое мы хотим использовать в нескольких местах вашего кода. Это полезно, так как мы можем создавать довольно сложные типы данных в SystemVerilog. Когда мы используем typedef вместо повторения сложного объявления типа, мы упрощаем наш код для понимания и поддержки.

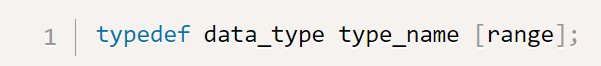


Рисунок 6 – Синтаксис конструкции typedef

Крайне удобным решением является объявление определённых регистров проектируемого модуля через typedef struct. Пример показан на рисунке 7.

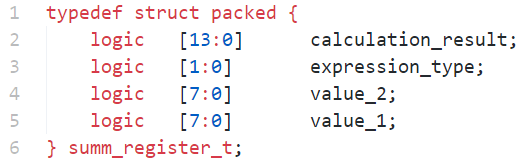


Рисунок 7 – Пример использования typedef для определения структуры регистра

**Package**

Package реализуют механизм хранения и передачи структур, методов, переменных, параметров и других конструкций HDL SystemVerilog между модулями для дальнейшего использования.

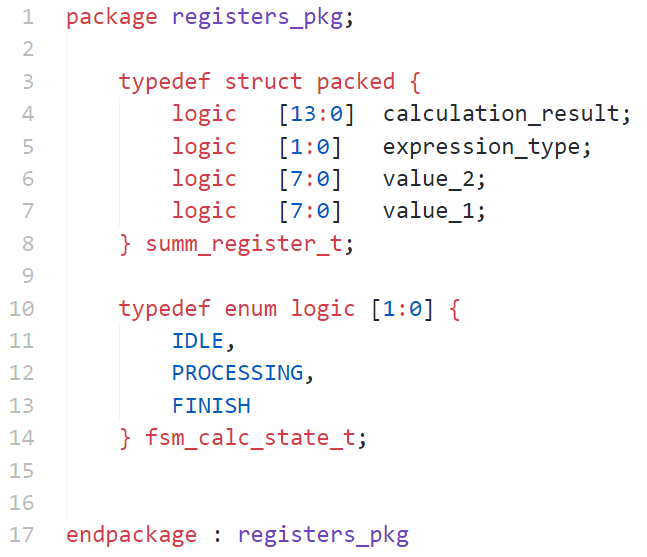


Рисунок 8 – Пример объявления package с перечнем необходимых конструкций для определённого регистра

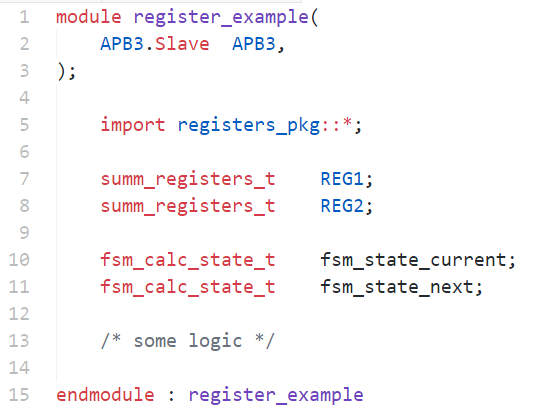


Рисунок 9 – Подключение package внутри модуля и использование его переменных

**CRC Алгоритм**

Cyclic Redundancy Code (Циклически избыточный код) – Одна из разновидностей контрольной суммы, необходимая для проверки целостности данных. Алгоритм вычисления подразумевает передачу пакета данных через устройство, реализующее мат. функцию (полином). На рисунке 10 представлен пример HDL кода обработки данных для CRC-16-CCITT с использованием полинома x16 + x12 + x5 + 1 (0x1021).

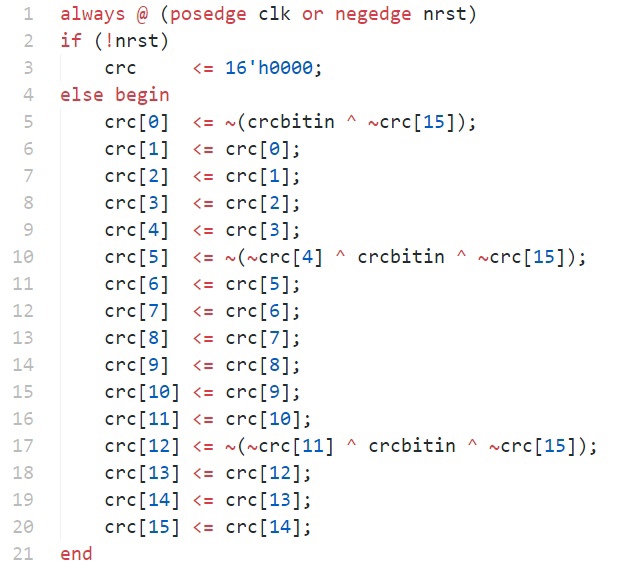


Рисунок 10 – Пример HDL кода вычисления последовательного CRC для CRC-16-CCITT (полином 0х1021)

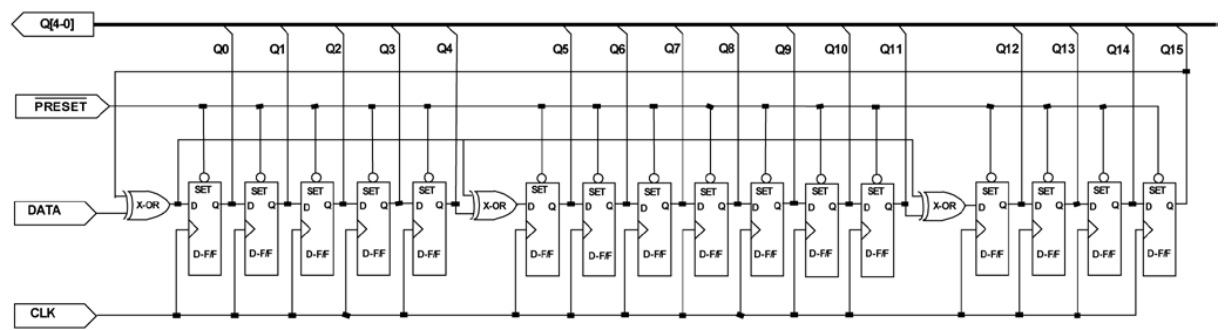


Рисунок 11 – Пример схемотехнической реализации последовательного расчёта CRC-16-CCITT

**Варианты заданий**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | CRC-16 Тип | Полином | Начальное значение | RefIn | RefOut | XorOut |
| 1 | CRC-16/MAXIM | 0x8005 | 0x0000 | TRUE | TRUE | 0xFFFF |
| 2 | CRC-16/CCITT-FALSE | 0x1021 | 0xFFFF | FALSE | FALSE | 0x0000 |
| 3 | CRC-16/ARC | 0x8005 | 0x0000 | TRUE | TRUE | 0x0000 |
| 4 | CRC-16/AUG-CCITT | 0x1021 | 0x1D0F | FALSE | FALSE | 0x0000 |
| 5 | CRC-16/BUYPASS | 0x8005 | 0x0000 | FALSE | FALSE | 0x0000 |
| 6 | CRC-16/CDMA2000 | 0xC867 | 0xFFFF | FALSE | FALSE | 0x0000 |
| 7 | CRC-16/DDS-110 | 0x8005 | 0x800D | FALSE | FALSE | 0x0000 |
| 8 | CRC-16/DECT-R | 0x0589 | 0x0000 | FALSE | FALSE | 0x0001 |
| 9 | CRC-16/DECT-X | 0x0589 | 0x0000 | FALSE | FALSE | 0x0000 |
| 10 | CRC-16/DNP | 0x3D65 | 0x0000 | TRUE | TRUE | 0xFFFF |
| 11 | CRC-16/EN-13757 | 0x3D65 | 0x0000 | FALSE | FALSE | 0xFFFF |

* RefIn — порядок поступления битов из буфера данных: false — начиная со старшего значащего бита (MSB first), true – LSB first;
* RefOut – признак инвертирования порядка битов на выходе: true – инвертировать.
* XorOut – значение, с которым необходимо произвести XOR результата вычисления до сохранения в регистр.

Результаты вычислений можно сверить на <https://crccalc.com>

Если адрес не совпадает с регистровой картой, необходимо поднимать PSLVERR.

Регистровая карта должна выглядеть следующим образом (все регистры 32 бита).

|  |  |  |
| --- | --- | --- |
| Название регистра | Адрес регистра (сдвиг) | Функционал |
| Input Data | 0х0 | Исходные данные для вычисления (32 бита) |
| Result | 0х4 | Результат вычислений (16 бит) |
| Flags | 0х8 | Флаги событий. 0 бит – начать вычисление (выставляет master-устройство). 15 бит – вычисление закончено (задаётся slave-устройством). |
| Status | 0хС | Необходим для дальнейшей отладки в Л.Р. 3 |

**Используемая литература**

1. SystemVerilog Modport. Режим доступа: <https://www.chipverify.com/systemverilog/systemverilog-modport>
2. CRC Calculation. Режим доступа: <https://www.crccalc.com>
3. Нововведения языка SystemVerilog. Режим доступа: <https://habr.com/ru/post/221265/>