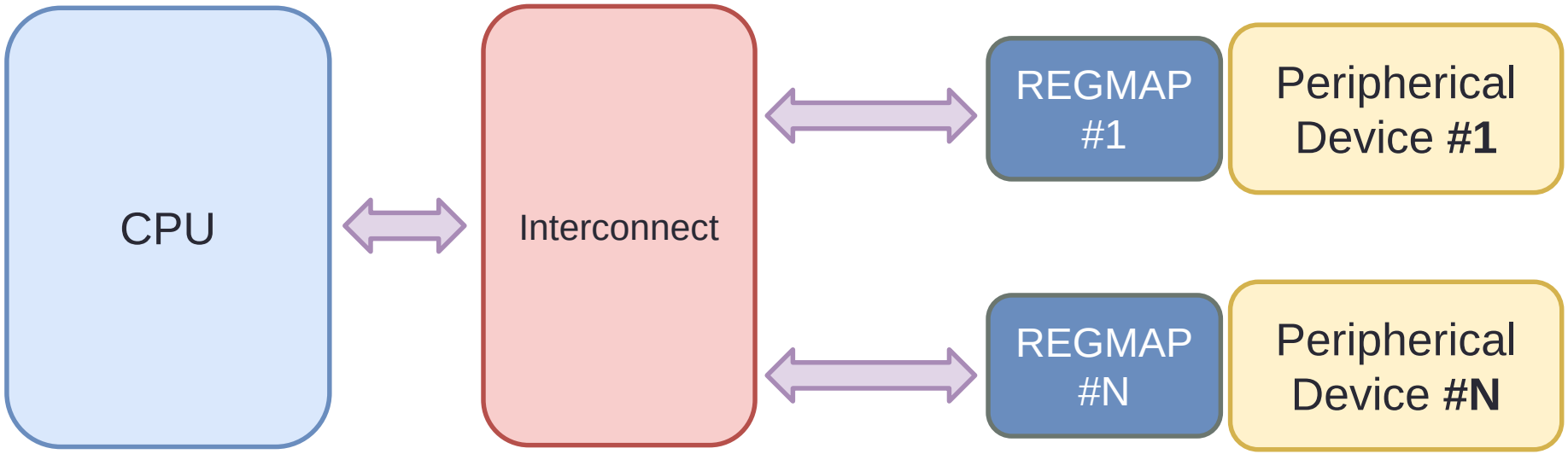
ПРОЕКТИРОВАНИЕ РЕГИСТРОВОЙ КАРТЫ УСТРОЙСТВА, ОБЛАДАЮЩЕГО СИСТЕМНЫМ ИНТЕРФЕЙСОМ

Курс «Проектирование СнК с программируемой архитектурой»

Авторы:
Любавин Кирилл Дмитриевич
Кузьмин Павел Андреевич

Регистровые карты



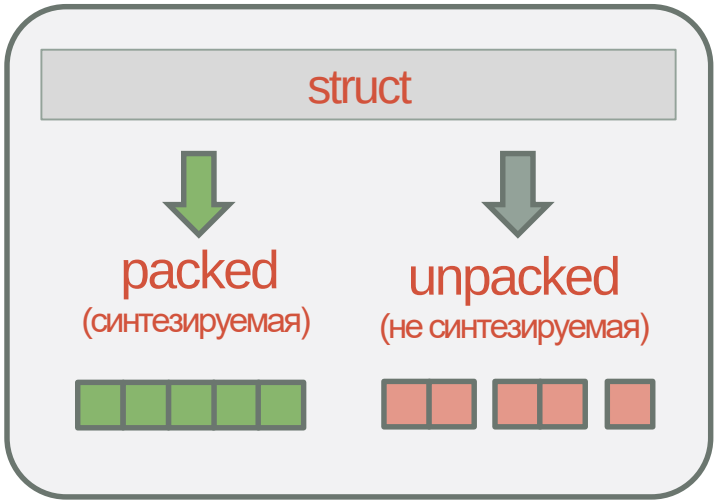
Регистровые карты (в совокупности с системными интерфейсами) являются одним из основных методов управления периферийными устройствами. Через них может производится как конфигурация устройства, так и их непосредственное управление. Управление происходит через передачу как исходных данных для проведения логических операций, так и через предоставление устройством информации о результатах выполненной работы или же служебной информации (версия или ревизия устройства, жестко закрепленные параметры и так далее).

Конструкция struct

Struct (структура) 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. **Struct** делится на два типа: **packed** (синтезируемая) и **unpacked** (не синтезируемая)

Синтаксис конструкции

```
struct [тип структуры
(packed/unpacked)] {
    список переменных]
} [название структуры];
```



Пример использования конструкции struct

```
1  localparam  YES = 1'b1,
2              NO  = 1'b0;
3
4  struct packed {
5      logic   [31:0]  fruit_id;
6      bit             is_on_sale;
7      byte            expiry_date;
8  } apple, orange;
9
10 always_comb apple.expiry_date   = 8'h7;
11 always_comb apple.is_on_sale    = YES;
12 always_comb apple.fruit_id      = 32'h0000_01FCD;
13
14 always_comb orange.expiry_date  = 8'h9;
15 always_comb orange.is_on_sale   = NO;
16 always_comb orange.fruit_id     = 32'h0000_01FCE;
```

Конструкция typedef

Typedef в SystemVerilog – позволяет создавать новые типы данных. Когда мы используем **typedef** вместо повторения сложного объявления типа, мы упрощаем наш код для понимания и поддержки.

Синтаксис конструкции

```
typedef [исходный_тип_данных]
[новый_тип_данных];
```

```
1  typedef data_type type_name [range];
```

Поскольку данная конструкция не создаёт новых типов данных, а лишь используется для создания псевдонимов уже существующих типов данных, возможность синтеза кода с использованием структуры typedef зависит исключительно от типов данных, для которых создаются их псевдонимы.

Пример использования конструкции typedef

```
1   typedef enum logic [1:0] {
2       ADD = 2'd0,
3       SUB = 2'd1,
4       MUL = 2'd2,
5       DIV = 2'd3
6   } exp_type_t;
7
8   typedef struct packed {
9       logic       [13:0]  calculation_result;
10      exp_type_t          expression_type;
11      logic       [7:0]   value_2;
12      logic       [7:0]   value_1;
13  } summ_reg_t;
14
15  summ_reg_t  summ_reg;
16
17  always_ff@(posedge clk or negedge nrst)
18  if(!nrst)
19      summ_reg.calculation_result <= '0;
20  else case(summ_reg.expression_type)
21
22      ADD:    summ_reg.calculation_result <= summ_reg.value_1 + summ_reg.value_2;
23      SUB:    summ_reg.calculation_result <= summ_reg.value_1 - summ_reg.value_2;
24      MUL:    summ_reg.calculation_result <= summ_reg.value_1 * summ_reg.value_2;
25      DIV:    summ_reg.calculation_result <= summ_reg.value_1 / summ_reg.value_2;
26
27  endcase
```

Конструкция package

Package в SystemVerilog - реализует механизм хранения и передачи структур, методов, переменных, параметров и других конструкций между объектами дизайна для дальнейшего использования.

Синтаксис конструкции

```
package [название];
    [содержимое/контент];
endpackage
```

Пример использования конструкции package

```
1  package registers_pkg;
2
3      typedef struct packed {
4          logic    [13:0]  calculation_result;
5          logic    [1:0]   expression_type;
6          logic    [7:0]   value_2;
7          logic    [7:0]   value_1;
8      } summ_register_t;
9
10     typedef enum logic [1:0] {
11         IDLE,
12         PROCESSING,
13         FINISH
14     } fsm_calc_state_t;
15
16
17 endpackage : registers_pkg
```

```
1  module register_example(
2      APB3.Slave  APB3,
3  );
4
5      import registers_pkg::*;
6
7      summ_registers_t    REG1;
8      summ_registers_t    REG2;
9
10     fsm_calc_state_t    fsm_state_current;
11     fsm_calc_state_t    fsm_state_next;
12
13     /* some logic */
14
15 endmodule : register_example
```

Методика проектирования регистровых карт

Регистровая карта каждого устройства **состоит из N регистров** (количество задается разработчиком, в зависимости от необходимого функционала), **каждый из которых в свою очередь состоит из полей**. Количество полей может быть от 1 до K, где K – количество бит в регистре.

Каждое поле обладает своим типом доступа и атрибутом доступа

Тип доступа	Описание
Read/Write (R/W)	Разрешены чтение и запись
Read Only (R/O)	Разрешено только чтение
Write Only (W/O)	Разрешена только запись

Атрибуты доступа	Описание
Write 1 to Set (W1S)	Разрешен переход из 0 в 1 путём записи 1.
Write 1 to Clear (W1C)	Разрешен переход из 1 в 0 путём записи 1.
Write 1 to Toggle (W1T)	Разрешен переход в инверс. значение путём записи 1.
Write 0 to Set (W0S)	Разрешен переход из 0 в 1 путём записи 0.
Write 0 to Clear (W0C)	Разрешен переход из 1 в 0 путём записи 0.
Write 0 to Toggle (W0T)	Разрешен переход в инверс. значение путём записи 0.

Методика проектирования регистровых карт

```
package device_regmap_pkg;

    typedef struct packed {
        logic   [15:0]  field2;     // RO
        logic   [15:0]  field1;     // RW
    } reg1_t;


    typedef struct packed {
        logic   [15:0]  field2;     // RW
        logic   [15:0]  field1;     // RW
    } reg2_t;

    typedef struct packed {
        logic   [7:0]  field4;      // RO
        logic   [7:0]  field3;      // RW
        logic   [7:0]  field2;      // RO
        logic   [7:0]  field1;      // RW
    } reg3_t;


    typedef struct packed {
        reg3_t          reg3;
        reg2_t          reg2;
        reg1_t          reg1;
    } regmap_t;

endpackage : device_regmap_pkg
```

```
module test;

import device_regmap_pkg::*;

regmap_t    regmap;

always_ff@(posedge clk or negedge nrst)
if(!nrst) begin
    regmap      <= '0;
end else if(wr_en) begin

    case(addr)

        0: begin
            regmap.reg1.field1      <= data_in;
        end

        1: begin
            regmap.reg2             <= data_in;
        end

        2: begin
            regmap.reg3.field1      <= data_in[7:0];
            regmap.reg3.field3      <= data_in[23:16];
        end

        default: /* do nothing */

    endcase

end

endmodule
```

Лабораторное задание

В лабораторной работе необходимо:

1. Составить регистровую карту устройства;
2. Описать тип доступа каждого регистра и его полей;
3. Реализовать полученную регистровую карту на базе APB3-ведомого устройства с использованием конструкций typedef, struct, package.

! За основу желательно использовать модуль устройства, разработанного в Л/Р №1.