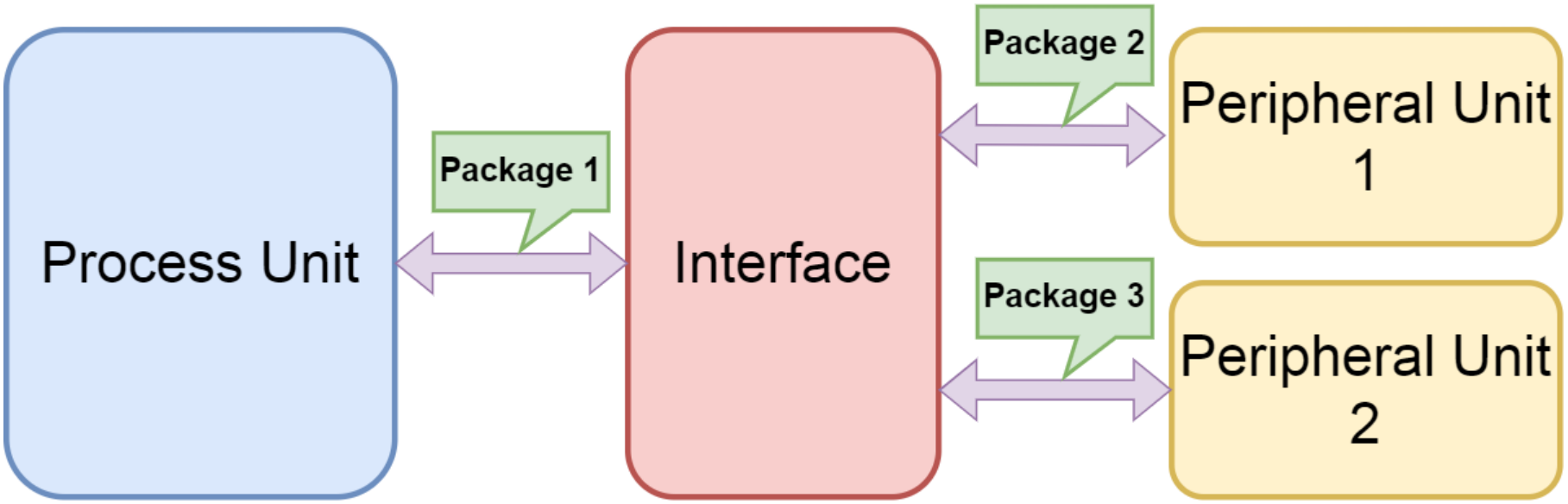
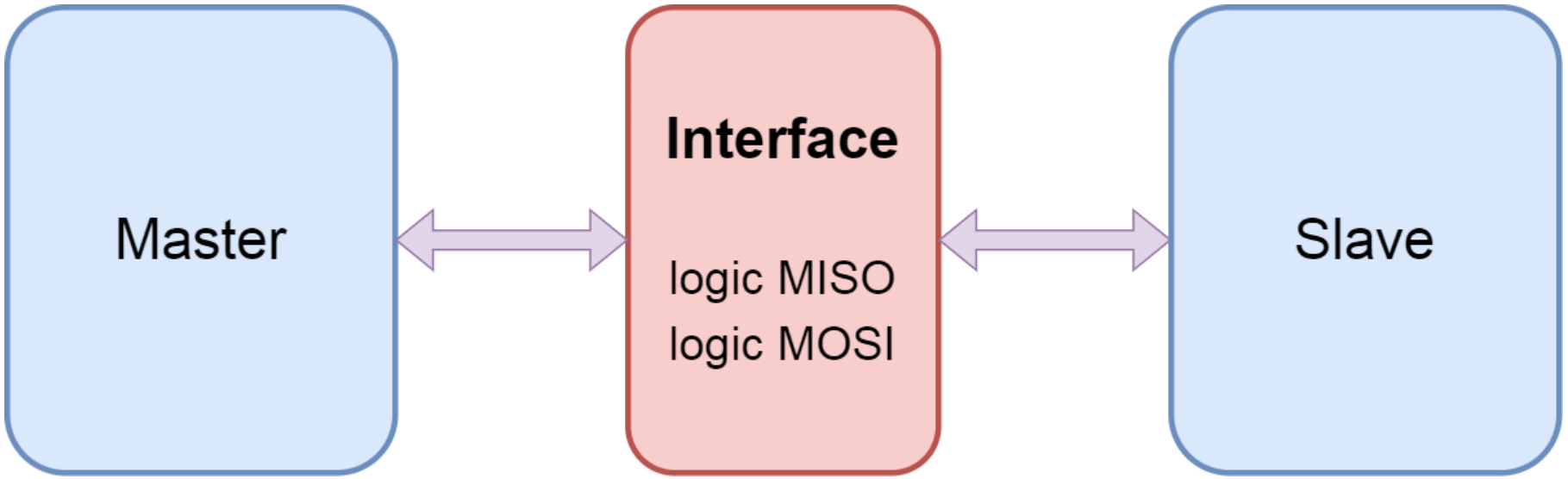
ПРОЕКТИРОВАНИЕ СНК С ПРОГРАММИРУЕМОЙ АРХИТЕКТУРОЙ

Лабораторная работа №2

Интерфейс


Process Unit
Package 1
Interface
Package 2
Peripheral Unit 1
Package 3
Peripheral Unit 2

Конструкция интерфейса modport



Конструкция интерфейса modport

```
interface SPI(
    input clk,
    input rst
);

logic MOSI;
logic MISO;
logic SS;

modport master(
    input   clk,
    input   rst,
    input   MISO,
    output  SS,
    output  MOSI
    );

modport slave(
    input   clk,
    input   rst,
    input   SS,
    input   MOSI,
    output  MISO
    );

endinterface : SPI
```

Пример описания

Без определения modport будут как inout

```
modport  [identifier]  (
    input  [port_list],
    output [port_list]
);
```

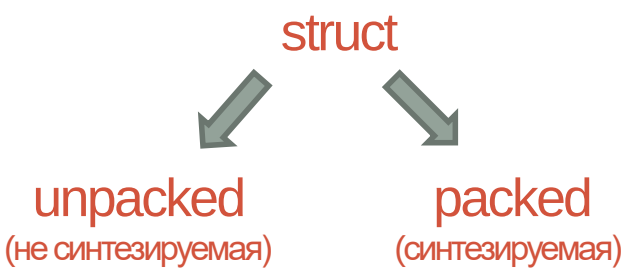
Синтаксис

```
module Slave(
    SPI.slave   spi_if
);

    /* some logic */

endmodule


module TB;

/* some logic */

SPI my_spi(clk, rst);
Slave udb_slave(spi_if.slave);

endmodule
```

Пример инстанцирования

Конструкция struct

Struct (структура) - 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 [struct_type] {
    [list_of_variables]
} struct_name;
```

Синтаксис конструкции

```
localparam  YES = 1'b1,
            NO  = 1'b0;

struct packed {
    logic   [31:0]  fruit_id;
    bit             is_on_sale;
    byte            expiry_date;
} fruit_t;

fruit_t apple;
fruit_t orange;

always_comb apple.expiry_date   = 8'h7;
always_comb apple.fruit_id      = 32'h0000_01FCD;
always_comb apple.is_on_sale    = YES;
```

Пример использования

Конструкция typedef

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.

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

Синтаксис конструкции

```
1  typedef struct packed {
2      logic    [13:0]      calculation_result;
3      logic    [1:0]       expression_type;
4      logic    [7:0]       value_2;
5      logic    [7:0]       value_1;
6  } summ_register_t;
```

Пример использования

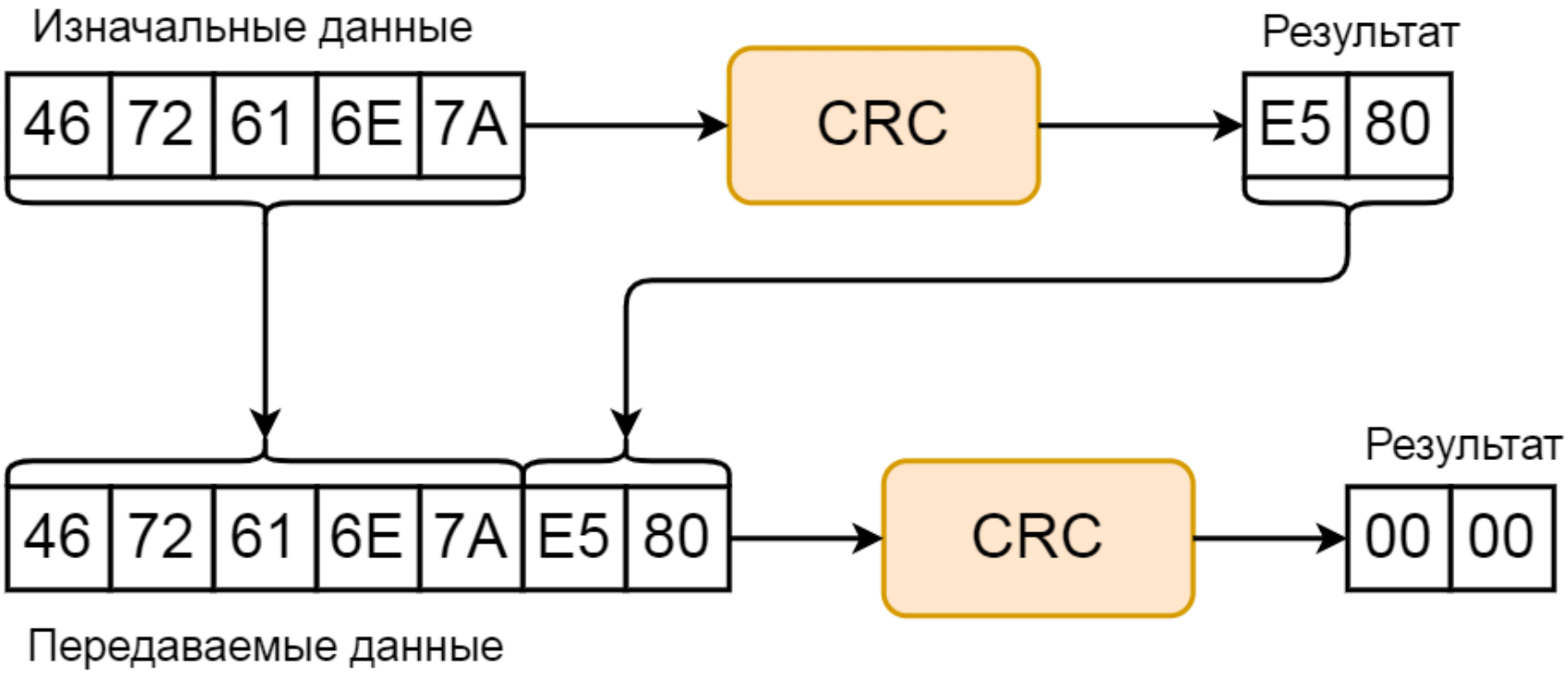
Конструкция package

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

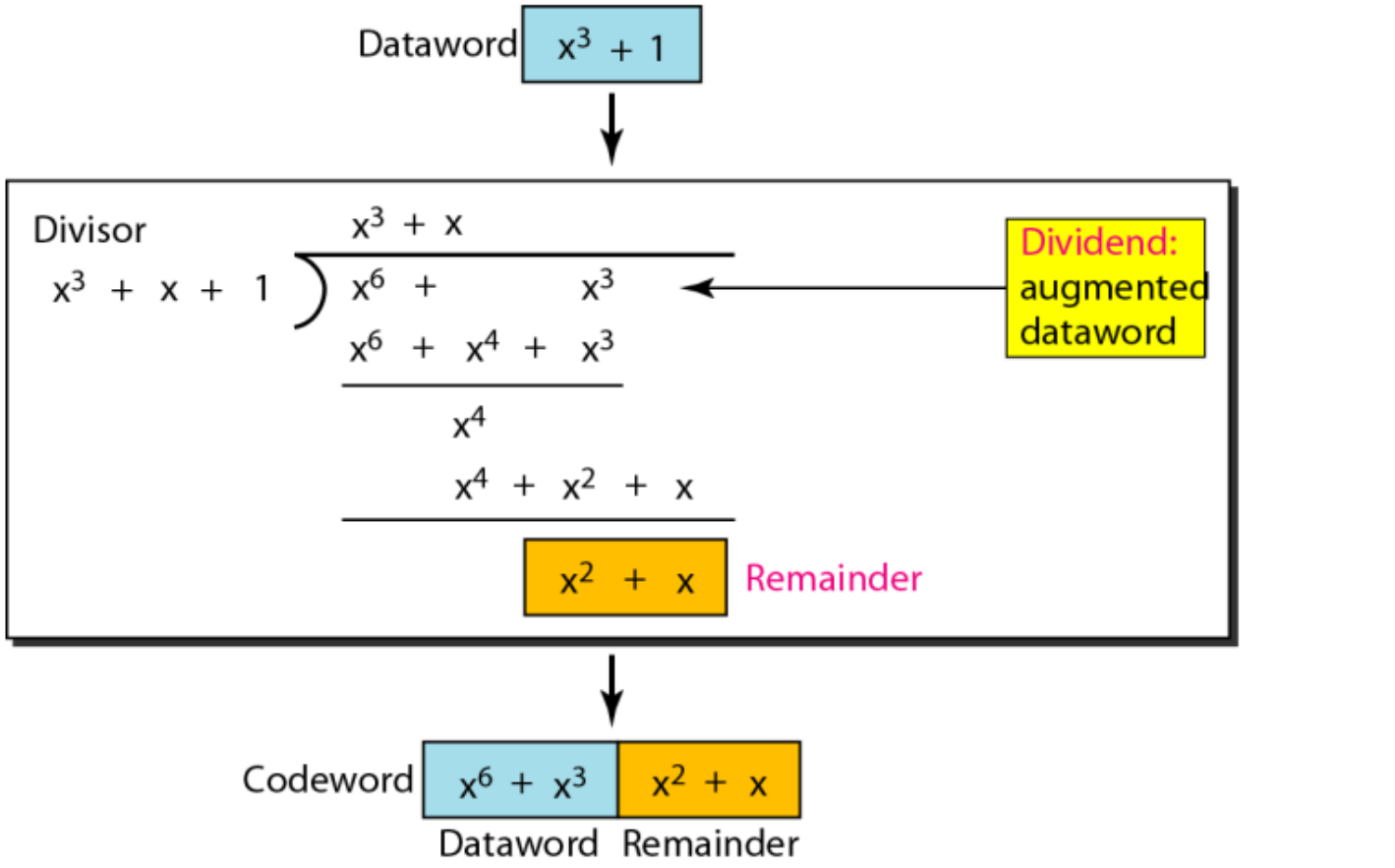
```
1  package registers_pkg;
2
3      typedef struct packed {
4          logic    [13:0]      calculation_result;
5          logic    [1:0]       expression_type;
6          logic    [7:0]       value_2;
7          logic    [7:0]       value_1;
8      } summ_register_t;
9
10     enum logic {
11    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

Проверка целостности, CRC

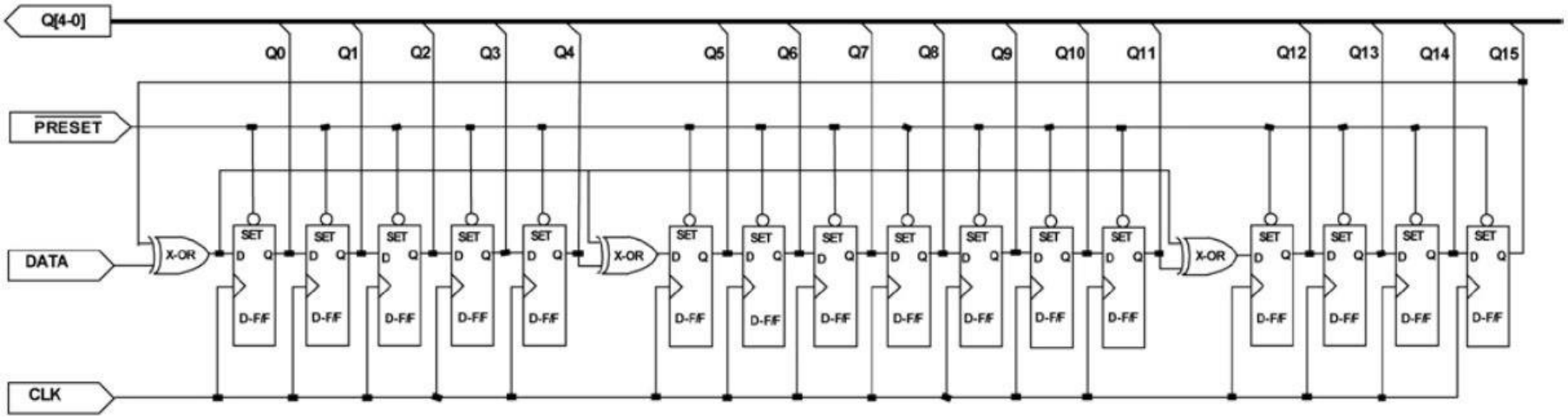


Проверка целостности, CRC



CRC, реализация

Полином: $X^{16} + X^{12} + X^5 + 1$



CRC, реализация

Полином: $X^{16} + X^{12} + X^5 + 1$

```
1  always @ (posedge clk or negedge nrst)
2  if (!nrst)
3      crc     <= 16'h0000;
4  else begin
5      crc[0]  <= ~(crcbitin ^ ~crc[15]);
6      crc[1]  <= crc[0];
7      crc[2]  <= crc[1];
8      crc[3]  <= crc[2];
9      crc[4]  <= crc[3];
10     crc[5]  <= ~(~crc[4] ^ crcbitin ^ ~crc[15]);
11     crc[6]  <= crc[5];
12     crc[7]  <= crc[6];
13     crc[8]  <= crc[7];
14     crc[9]  <= crc[8];
15     crc[10] <= crc[9];
16     crc[11] <= crc[10];
17     crc[12] <= ~(~crc[11] ^ crcbitin ^ ~crc[15]);
18     crc[13] <= crc[12];
19     crc[14] <= crc[13];
20     crc[15] <= crc[14];
21 end
```

Цели лабораторной работы

1) Изучить вышеописанные конструкции, применить их в дальнейших задачах;

2) Дополнить модуль, написанный в л/р1 логикой адресов регистров, привязать к ним вычисление CRC-16 в соответствии с вариантом;

3) Сделать тестбенч для вашего модуля, продемонстрировать процессы записи и чтения, вычисления CRC, корректную работу флагов.

Проверить результаты вычисления CRC-16 можно тут: https://crccalc.com
Требования по адресной карте (адреса регистров и их содержание) смотрите в LabWork2.doc

Задание на Л/Р (Варианты CRC-16)

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