

This lecture aims at giving you a more thorough understanding of how certain VHDL code structures map to hardware. We will not introduce any new language concepts and instead focus on practically applying what we already know.

HWMod
WS25

C2C

Introduction
Circuit \rightarrow VHDL
VHDL \rightarrow Circuit

Hardware Modeling [VU] (191.011) – WS25 – From Circuits to Code and Back

Florian Huemer & Sebastian Wiedemann & Dylan Baumann

WS 2025/26

Similar to how a system-level programmer must have a good understanding of how the written program maps to the available machine instructions on a given architecture, in order to produce efficient software, you – as a hardware developer – should have a similar intuition about the mapping between VHDL code and circuits. Hence, with this lecture we want to help you to develop such an intuition, as we believe that this is a key skill in becoming a great hardware developer!

Introduction

HWMoD
WS25

C2C
Introduction
Circuit → VHDL
VHDL → Circuit

Lecture Goal

Develop an intuition for how VHDL code maps to hardware and the associated circuit complexity.

- Two examples
 - Circuit diagram → Derive VHDL code
 - VHDL code → Derive circuit diagram

To achieve this goal we present and work through two circuit modeling examples. In the first example we take a circuit diagram and derive a VHDL entity/architecture pair that models the behavior of the given circuit. For the second example, we perform the reverse operation, taking a simple VHDL design and deriving a circuit diagram from it. We will thus effectively perform the task of the synthesis tool ourselves.

Introduction

HWMoD
WS25

C2C
Introduction
Circuit → VHDL
VHDL → Circuit

Lecture Goal

Develop an intuition for how VHDL code maps to hardware and the associated circuit complexity.

- Two examples
 - Circuit diagram → Derive VHDL code
 - VHDL code → Derive circuit diagram

Note that, besides other topics, this lecture heavily builds upon the videos about sequential circuit elements and behavioral modeling.

Introduction

HWMod
WS25

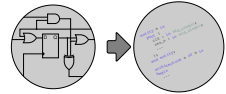
C2C
Introduction
Circuit → VHDL
VHDL → Circuit

Lecture Goal

Develop an intuition for how VHDL code maps to hardware and the associated circuit complexity.

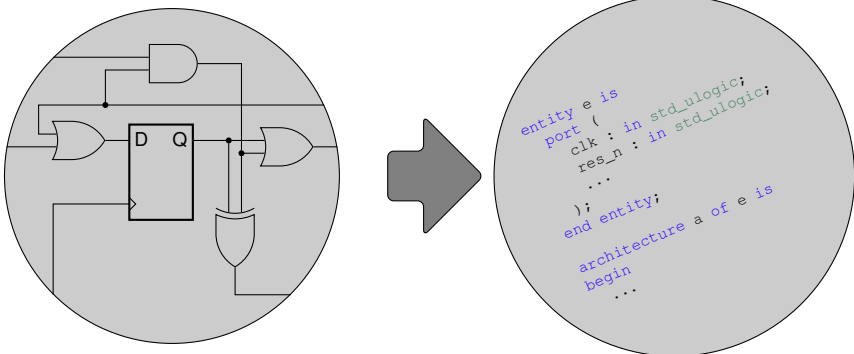
- Two examples
 - Circuit diagram → Derive VHDL code
 - VHDL code → Derive circuit diagram
- Important VHDL concepts used in the lecture
 - Sequential circuit elements
 - Behavioral modeling
 - Array-types and arithmetic functions

- └ From Circuits to Code and Back
 - └ Circuit → VHDL
 - └ **Example 1: Circuit → VHDL Code**



Let's start with the first example, where we take a circuit diagram and derive an appropriate VHDL design from it.

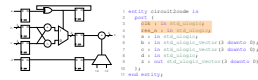
Example 1: Circuit → VHDL Code



HWMMod
WS25

C2C
Introduction
Circuit → VHDL
Circuit
Entity
Architecture
Restructured
Architecture
VHDL → Circuit

From Circuits to Code and Back
 Circuit → VHDL
Example 1 - Entity

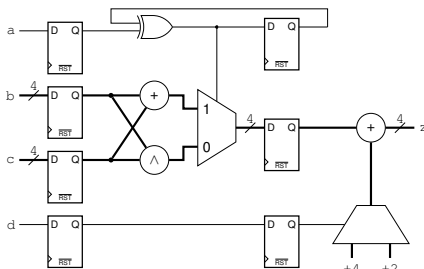


First, we add a clock and reset input, as we will need them to implement the registers. As always, we use the `std_ulogic` type for both of these signals. Note that the symbols used for the flip-flops in the circuit indicate an active-low reset.

Example 1 - Entity

HWMod
WS25

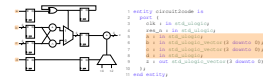
C2C
 Introduction
 Circuit → VHDL
 Circuit
 Entity
 Architecture
 Restructured
 Architecture
 VHDL → Circuit



```

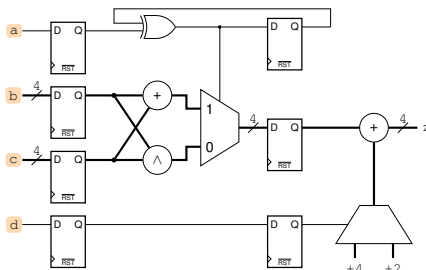
1 entity circuit2code is
2   port (
3     clk : in std_ulogic;
4     res_n : in std_ulogic;
5     a : in std_ulogic;
6     b : in std_ulogic_vector(3 downto 0);
7     c : in std_ulogic_vector(3 downto 0);
8     d : in std_ulogic;
9     z : out std_ulogic_vector(3 downto 0)
10  );
11 end entity;
  
```

- └ From Circuits to Code and Back
 - └ Circuit → VHDL
 - └ **Example 1 - Entity**



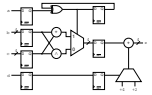
Next, we list all the input signals in the order of their appearance. For the signals `b` and `c` we could also have gone with the signed or the unsigned data-type. However, as the signedness of these input values does not change the way they need to be handled by the circuit, we simply settled on the `std_ulogic_vector` type.

Example 1 - Entity



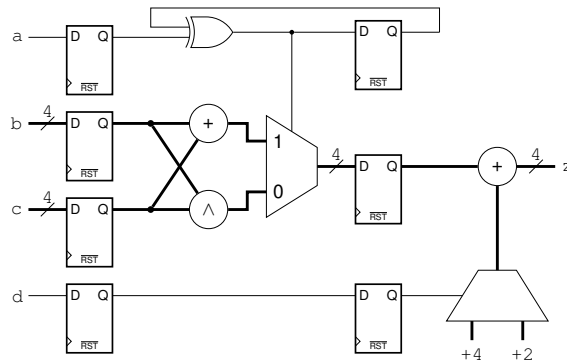
```

1 entity circuit2code is
2   port (
3     clk : in std_logic;
4     res_n : in std_logic;
5     a : in std_logic;
6     b : in std_logic_vector(3 downto 0);
7     c : in std_logic_vector(3 downto 0);
8     d : in std_logic;
9     z : out std_logic_vector(3 downto 0)
10  );
11 end entity;
  
```

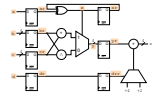



With the entity in place, we can now turn our attention to the architecture, which contains the description of the actual circuit behavior. Before we can start coding, we have to label some of the internal signals of our circuit, such that we can then refer to these signals using appropriate identifiers in the VHDL design. Let us therefore again look at the circuit.

Example 1 - Architecture



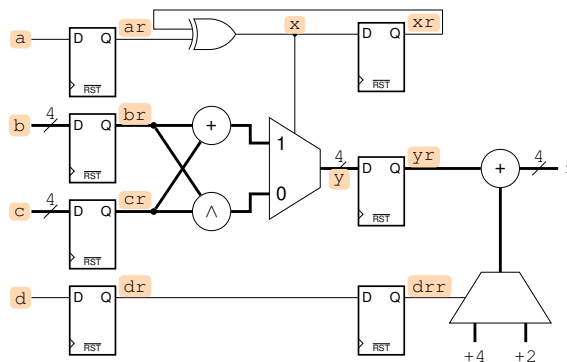
- └ From Circuits to Code and Back
 - └ Circuit → VHDL
 - └ **Example 1 - Architecture**



Circuit Preparation
Label all register inputs and outputs and other important signals!

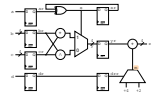
In particular, we make sure that the inputs and outputs of all sequential elements have proper names. Therefore, for all inputs and outputs of such elements, which are not directly connected to an entity port signal, we need to introduce a proper signal name. For this example we use the following naming scheme: The output of a sequential element gets the name of its input extended by the letter *r*, referring to the signal being "registered". Furthermore, depending on the exact way we express the combinational parts of the circuit as VHDL code, it can also make sense to label further signals.

Example 1 - Architecture



Circuit Preparation

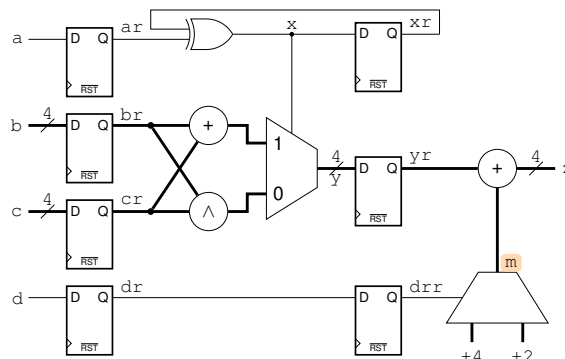
Label **all** register inputs and outputs and other important signals!



Circuit Preparation
 Label all register inputs and outputs and other important signals!

One candidate for such a signal is the output of the multiplexer in the lower right corner, which we simply label *m*.

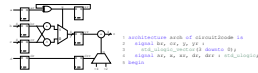
Example 1 - Architecture



Circuit Preparation

Label **all** register inputs and outputs and other important signals!

From Circuits to Code and Back
 Circuit → VHDL
Architecture - Signal Declarations

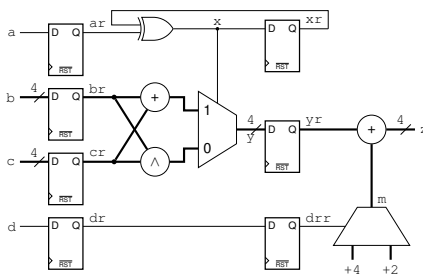


Now we have everything we need to start implementing the architecture. We begin with the declarations for the internal signals, that we have identified and labeled in the previous step. Note that we don't declare a signal for m here. We will see why we don't do this shortly.

Architecture - Signal Declarations

HWMoD
 WS25

C2C
 Introduction
 Circuit → VHDL
 Circuit
 Entity
Architecture
 Restructured
 Architecture
 VHDL → Circuit



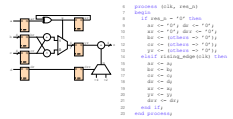
```

1 architecture arch of circuit2code is
2     signal br, cr, y, yr :
3         std_ulogic_vector(3 downto 0);
4     signal ar, x, xr, dr, drr : std_ulogic;
5 begin
  
```

From Circuits to Code and Back

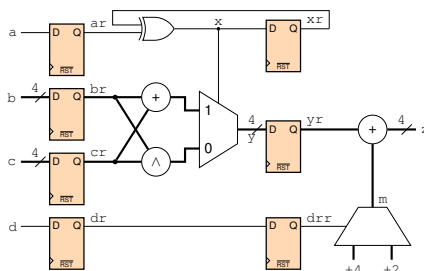
Circuit → VHDL

Architecture - Registers



After we declared the required internal signals, we can start to implement the behavior of our circuit within the architecture body. A good place to start are the sequential elements. Hence, we create a synchronous process by using the appropriate pattern presented in the sequential circuit elements lecture. Inside this process we write to all signals in our design that represent the output of a register. For our example these are the signals whose names end with an `r`.

Architecture - Registers



```

6 process (clk, res_n)
7 begin
8   if res_n = '0' then
9     ar <= '0'; dr <= '0';
10    xr <= '0'; drr <= '0';
11    br <= (others => '0');
12    cr <= (others => '0');
13    yr <= (others => '0');
14  elsif rising_edge(clk) then
15    ar <= a;
16    br <= b;
17    cr <= c;
18    dr <= d;
19    xr <= x;
20    yr <= y;
21    drr <= dr;
22  end if;
23 end process;

```

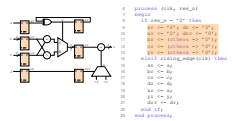
HWMoD
WS25

C2C
Introduction
Circuit → VHDL
Circuit
Entity
Architecture
Restructured
Architecture
VHDL → Circuit

From Circuits to Code and Back

Circuit → VHDL

Architecture - Registers

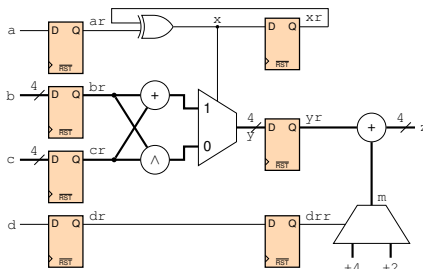


As always in our course, we use an asynchronous reset. Furthermore, as already noted, the circuit diagram indicates that the reset is active-low. In addition to that, because the flip-flops in the circuit feature a reset rather than a set input, we initialize all of them to zero.

Architecture - Registers

HWMoD
WS25

C2C
Introduction
Circuit → VHDL
Circuit
Entity
Architecture
Restructured
Architecture
VHDL → Circuit

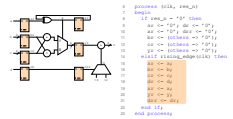


```

6 process (clk, res_n)
7 begin
8     if res_n = '0' then
9         ar <= '0'; dr <= '0';
10        xr <= '0'; drr <= '0';
11        br <= (others => '0');
12        cr <= (others => '0');
13        yr <= (others => '0');
14    elsif rising_edge(clk) then
15        ar <= a;
16        br <= b;
17        cr <= c;
18        dr <= d;
19        xr <= x;
20        yr <= y;
21        drr <= dr;
22    end if;
23 end process;

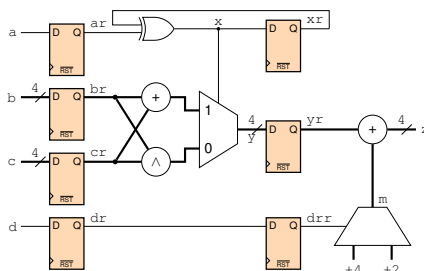
```

From Circuits to Code and Back
 Circuit → VHDL
 Architecture - Registers



On rising clock edges we simply assign each input signal of a sequential element to the appropriate output signal. For example, the signal `a` is assigned to `ar`. This process is a quite straight forward step that doesn't demand much thought and is more or less independent of the actual circuit.

Architecture - Registers

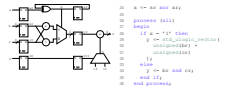


```

6 process (clk, res_n)
7 begin
8   if res_n = '0' then
9     ar <= '0'; dr <= '0';
10    xr <= '0'; drr <= '0';
11    br <= (others => '0');
12    cr <= (others => '0');
13    yr <= (others => '0');
14  elsif rising_edge(clk) then
15    ar <= a;
16    br <= b;
17    cr <= c;
18    dr <= d;
19    xr <= x;
20    yr <= y;
21    drr <= dr;
22  end if;
23 end process;

```

From Circuits to Code and Back
 Circuit → VHDL
Architecture - Combinational Logic I

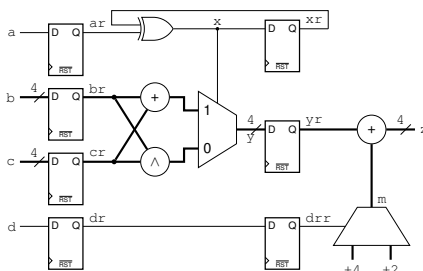


With the sequential circuit elements in place, we can now add the combinational logic between them that defines most of our circuits specific behavior.

Architecture - Combinational Logic I

HWMoD
WS25

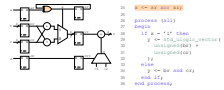
C2C
Introduction
Circuit → VHDL
Circuit
Entity
Architecture
Restructured
Architecture
VHDL → Circuit



```

24  x <= ar xor xr;
25
26  process (all)
27  begin
28    if x = '1' then
29      y <= std_ulogic_vector(
30        unsigned(br) +
31        unsigned(cr)
32      );
33    else
34      y <= br and cr;
35    end if;
36  end process;
  
```

From Circuits to Code and Back
 Circuit → VHDL
Architecture - Combinational Logic I

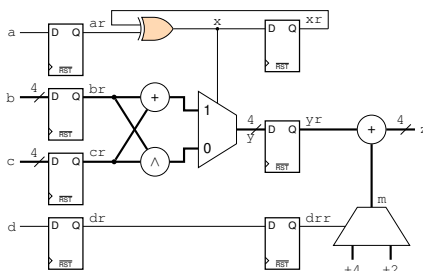


Let's start with the XOR gate in the upper part of the circuit that produces the x signal. While there exist multiple ways to describe this logic, we can very easily express such simple gates using concurrent signal assignments. An assignment like the one highlighted on the slide typically directly maps to a single gate of the respective type for single bit signals, and to multiple such gates for multi-bit signals.

Architecture - Combinational Logic I

HWMoD
 WS25

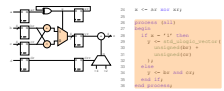
C2C
 Introduction
 Circuit → VHDL
 Circuit
 Entity
Architecture
 Restructured
 Architecture
 VHDL → Circuit



```

24  x <= ar xor xr;
25
26  process (all)
27  begin
28    if x = '1' then
29      y <= std_ulogic_vector(
30        unsigned(br) +
31        unsigned(cr)
32      );
33    else
34      y <= br and cr;
35    end if;
36  end process;
  
```

From Circuits to Code and Back
 Circuit → VHDL
Architecture - Combinational Logic I

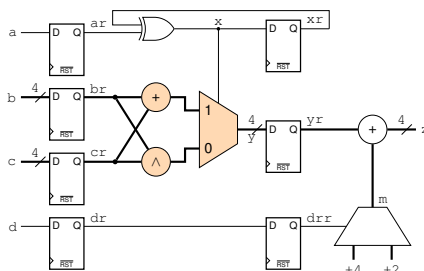


For the logic that decides whether to forward the "bitwise-AND" or the sum of the signals `br` and `cr` to the signal `y` we use a separate process. As you already know, multiplexers can be expressed using if-else statements. Hence, we test the signal `x` and either assign the result of the "bitwise-AND" or the addition to `y`. Of course this is not the only possibility how we can describe this sub-circuit in VHDL. One alternative would be to use a concurrent signal assignment with an appropriate when/else expression. However, operations comprising multiple circuit elements quickly become quite confusing using such assignments, and we therefore usually recommend the use of a process in such cases. Please note that, we would get the exact same circuit, if we would put the concurrent signal assignment for `x` into the process for `y`.

Architecture - Combinational Logic I

HWMoD
 WS25

C2C
 Introduction
 Circuit → VHDL
 Circuit
 Entity
Architecture
 Restructured
 Architecture
 VHDL → Circuit



```
24 x <= ar xor xr;
```

25

26

27

28

29

30

31

32

33

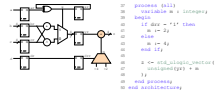
34

35

36

```
process (all)
begin
  if x = '1' then
    y <= std_ulogic_vector(
      unsigned(br) +
      unsigned(cr)
    );
  else
    y <= br and cr;
  end if;
end process;
```

From Circuits to Code and Back
 Circuit → VHDL
Architecture - Combinational Logic II

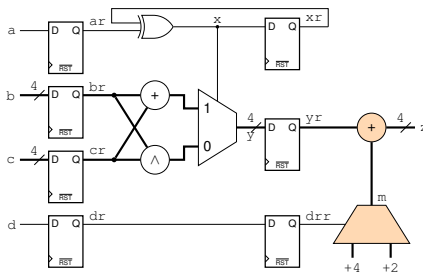


The last thing we need to do is to implement the logic that produces the output signal z . For that we use a separate process, although this is not strictly required. It would be semantically equivalent to simply add this code to the other combinational process that produces the y signal. However, it is often preferable to restrict processes to related parts of a circuit in order to give your code more structure and to improve its readability and maintainability.

Architecture - Combinational Logic II

HWMoD
 WS25

C2C
 Introduction
 Circuit → VHDL
 Circuit
 Entity
Architecture
 Restructured
 Architecture
 VHDL → Circuit

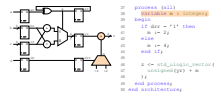


```

37 process (all)
38     variable m : integer;
39 begin
40     if drr = '1' then
41         m := 2;
42     else
43         m := 4;
44     end if;
45
46     z <= std_ulogic_vector(
47         unsigned(yr) + m
48     );
49 end process;
50 end architecture;

```

From Circuits to Code and Back
 Circuit → VHDL
Architecture - Combinational Logic II

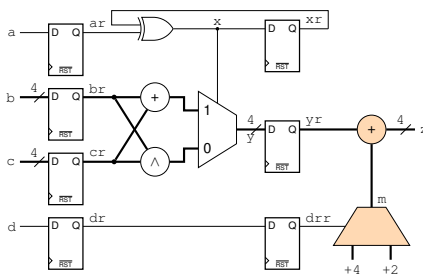


We start with the declaration of the intermediate `m` as a variable in the process. Since the output of the second multiplexer is only required within this process, it makes sense to restrict its scope in order to reduce needless cluttering of the architecture's name space. We select the `integer` data type for `m`. However, `std_ulogic_vector`, or `unsigned` would also be valid options.

Architecture - Combinational Logic II

HWMoD
WS25

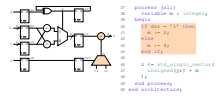
C2C
Introduction
Circuit → VHDL
Circuit
Entity
Architecture
Restructured
Architecture
VHDL → Circuit



```

37 process (all)
38   variable m : integer;
39 begin
40   if drr = '1' then
41     m := 2;
42   else
43     m := 4;
44   end if;
45
46   z <= std_ulogic_vector(
47     unsigned(yr) + m
48   );
49 end process;
50 end architecture;
  
```

From Circuits to Code and Back
 Circuit → VHDL
Architecture - Combinational Logic II

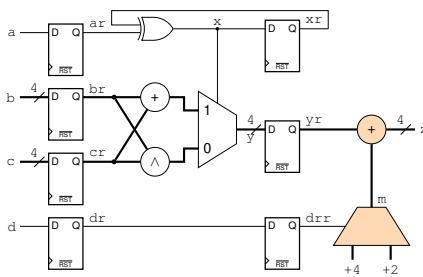


Next we implement the multiplexer that selects between the constants two and four based on the value of `drr` and writes to the variable `m`.

Architecture - Combinational Logic II

HWMod
WS25

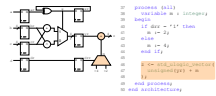
C2C
Introduction
Circuit → VHDL
Circuit
Entity
Architecture
Restructured
Architecture
VHDL → Circuit



```

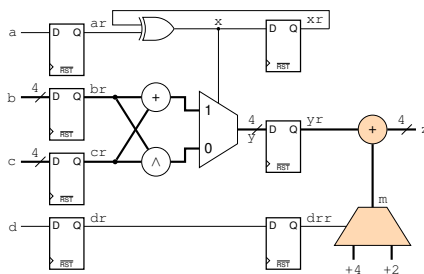
37 process (all)
38     variable m : integer;
39 begin
40     if drr = '1' then
41         m := 2;
42     else
43         m := 4;
44     end if;
45
46     z <= std_ulogic_vector(
47         unsigned(yr) + m
48     );
49 end process;
50 end architecture;
    
```

From Circuits to Code and Back
 Circuit → VHDL
Architecture - Combinational Logic II



Finally, we produce the output z by adding yr to m . This completes our architecture and thus also our VHDL design.

Architecture - Combinational Logic II



```

37 process (all)
38   variable m : integer;
39 begin
40   if drr = '1' then
41     m := 2;
42   else
43     m := 4;
44   end if;
45
46   z <= std_ulogic_vector(
47     unsigned(yr) + m
48   );
49 end process;
50 end architecture;
  
```

From Circuits to Code and Back

Circuit → VHDL

Restructured Architecture

```

1 process (clk, res_n)
2 begin
3     if res_n = '0' then
4         ar <= '0'; dr <= '0';
5         xr <= '0'; drr <= '0';
6         br <= (others => '0');
7         cr <= (others => '0');
8         yr <= (others => '0');
9     elsif rising_edge(clk) then
10        ar <= a;
11        br <= b;
12        cr <= c;
13        dr <= d;
14        xr <= x;
15        yr <= y;
16        drr <= dr;
17    end if;
18 end process;

```

Before we continue with the next example, we want to show you that there usually exist multiple different VHDL descriptions of the exact same circuit. Therefore, let us quickly discuss some example modifications to our previous VHDL code that do not change the resulting circuit. The reason why we discuss that, is to really show you the different ways to describe the exact same circuit.

Restructured Architecture

HWMMod
WS25

C2C
Introduction
Circuit → VHDL
Circuit
Entity
Architecture
Restructured
Architecture
VHDL → Circuit

```

1  process (clk, res_n)
2  begin
3      if res_n = '0' then
4          ar <= '0'; dr <= '0';
5          xr <= '0'; drr <= '0';
6          br <= (others => '0');
7          cr <= (others => '0');
8          yr <= (others => '0');
9      elsif rising_edge(clk) then
10         ar <= a;
11         br <= b;
12         cr <= c;
13         dr <= d;
14         xr <= x;
15         yr <= y;
16         drr <= dr;
17     end if;
18 end process;
19
20
21 process (all)
22 begin
23     if x = '1' then
24         y <= std_ulogic_vector(
25             unsigned(br) +
26             unsigned(cr)
27         );
28     else
29         y <= br and cr;
30     end if;
31 end process;

```

From Circuits to Code and Back
 Circuit → VHDL
 Restructured Architecture

```

1 process (clk, res_n)
2 begin
3   if res_n = '0' then
4     ar <= '0'; dr <= '0';
5     xr <= '0'; drr <= '0';
6     br <= (others => '0');
7     cr <= (others => '0');
8     yr <= (others => '0');
9   elsif rising_edge(clk) then
10    ar <= a;
11    br <= b;
12    cr <= c;
13    dr <= d;
14    xr <= x;
15    yr <= y;
16    drr <= dr;
17  end if;
18 end process;
  
```

One thing we could do is merge the concurrent signal assignment and the process that produces the y signal into the synchronous process. This change would reduce the overall length of our architecture. However, it arguably also makes the VHDL code less structured, as we no longer have dedicated processes for synchronous and combinational parts of the circuit.

Restructured Architecture

- HWMod
- WS25
- C2C
- Introduction
- Circuit → VHDL
- Circuit
- Entity
- Architecture
- Restructured Architecture
- VHDL → Circuit

```

1  process (clk, res_n)
2  begin
3    if res_n = '0' then
4      ar <= '0'; dr <= '0';
5      xr <= '0'; drr <= '0';
6      br <= (others => '0');
7      cr <= (others => '0');
8      yr <= (others => '0');
9    elsif rising_edge(clk) then
10     ar <= a;
11     br <= b;
12     cr <= c;
13     dr <= d;
14     xr <= x;
15     yr <= y;
16     drr <= dr;
17   end if;
18 end process;
  
```

```

19 x <= ar xor xr;
20
21 process (all)
22 begin
23   if x = '1' then
24     y <= std_ulogic_vector(
25       unsigned(br) +
26       unsigned(cr)
27     );
28   else
29     y <= br and cr;
30   end if;
31 end process;
  
```

From Circuits to Code and Back
 Circuit → VHDL
Restructured Architecture

```

1 process (clk, res_n)
2 begin
3   if res_n = '0' then
4     ar <= '0'; dr <= '0';
5     xr <= '0'; drr <= '0';
6     br <= (others => '0');
7     cr <= (others => '0');
8     yr <= (others => '0');
9   elsif rising_edge(clk) then
10    ar <= a;
11    br <= b;
12    cr <= c;
13    dr <= d;
14    xr <= x;
15    yr <= y;
16    drr <= dr;
17  end if;
18 end process;
19
20
21 process (all)
22 begin
23   if x = '1' then
24     y <= std_ulogic_vector(
25       unsigned(br) +
26       unsigned(cr)
27     );
28   else
29     y <= br and cr;
30   end if;
31 end process;
  
```

Let's start with the process. The goal is to take the shown sequence of code and replace the assignment in the synchronous process with it. However, care must be taken because the combinational process writes to the signal y , while the synchronous process writes to y_r . To get semantically equivalent code, we therefore need to replace all occurrences of y by y_r in the part we take from the combinational process.

Restructured Architecture

```

1  process (clk, res_n)
2  begin
3    if res_n = '0' then
4      ar <= '0'; dr <= '0';
5      xr <= '0'; drr <= '0';
6      br <= (others => '0');
7      cr <= (others => '0');
8      yr <= (others => '0');
9    elsif rising_edge(clk) then
10     ar <= a;
11     br <= b;
12     cr <= c;
13     dr <= d;
14     xr <= x;
15     yr <= y;
16     drr <= dr;
17   end if;
18 end process;
19
20
21 process (all)
22 begin
23   if x = '1' then
24     y <= std_ulogic_vector(
25       unsigned(br) +
26       unsigned(cr)
27     );
28   else
29     y <= br and cr;
30   end if;
31 end process;
  
```

- HWMMod
- WS25
- C2C
- Introduction
- Circuit → VHDL
- Circuit
- Entity
- Architecture
- Restructured Architecture
- VHDL → Circuit

From Circuits to Code and Back
 Circuit → VHDL
Restructured Architecture

```

1 process (clk, res_n)
2 begin
3   if res_n = '1' then
4     ar <= '0'; xr <= '0';
5     dr <= '0'; drr <= '0';
6     br <= (others => '0');
7     cr <= (others => '0');
8     yr <= (others => '0');
9     elsif rising_edge(clk) then
10      ar <= a;
11      br <= b;
12      cr <= c;
13      dr <= d;
14      xr <= ar xor xr;
15      if x = '1' then
16        yr <= std_ulogic_vector(
17          unsigned(br) +
18          unsigned(cr)
19        );
20      else
21        yr <= br and cr;
22      end if;
23      drr <= dr;
24    end if;
25  end process;
26
27 x <= ar xor xr;
28

```

This slide shows the result of this modification.

Restructured Architecture

HWMod
WS25

C2C
Introduction
Circuit → VHDL
Circuit
Entity
Architecture
Restructured
Architecture
VHDL → Circuit

```

1  process (clk, res_n)
2  begin
3    if res_n = '1' then
4      ar <= '0'; xr <= '0';
5      dr <= '0'; drr <= '0';
6      br <= (others => '0');
7      cr <= (others => '0');
8      yr <= (others => '0');
9      elsif rising_edge(clk) then
10       ar <= a;
11       br <= b;
12       cr <= c;
13       dr <= d;
14       xr <= ar xor xr;
15       if x = '1' then
16         yr <= std_ulogic_vector(
17           unsigned(br) +
18           unsigned(cr)
19         );
20       else
21         yr <= br and cr;
22       end if;
23       drr <= dr;
24     end if;
25  end process;
26
27 x <= ar xor xr;
28

```

From Circuits to Code and Back

Circuit → VHDL

Restructured Architecture

```

1 process (clk, res_n)    is
2 begin
3   if res_n = '1' then
4     ar <= '0'; xr <= '0';
5     dr <= '0'; drr <= '0';
6     br <= (others => '0');
7     cr <= (others => '0');
8     yr <= (others => '0');
9     elsif rising_edge(clk) then
10      ar <= a;
11      br <= b;
12      cr <= c;
13      dr <= d;
14      xr <= x;
15
16      if x = '1' then
17        yr <= std_ulogic_vector(
18          unsigned(br) +
19          unsigned(cr)
20        );
21      else
22        yr <= br and cr;
23      end if;
24      drr <= dr;
25    end if;
26  end process;
27
28  x <= ar xor xr;

```

Notice that the synchronous process only writes to the signal `yr` and that the signal `y` is no longer needed at all. We can therefore remove its declaration altogether.

Restructured Architecture

```

1  process (clk, res_n)
2  begin
3    if res_n = '1' then
4      ar <= '0'; xr <= '0';
5      dr <= '0'; drr <= '0';
6      br <= (others => '0');
7      cr <= (others => '0');
8      yr <= (others => '0');
9    elsif rising_edge(clk) then
10     ar <= a;
11     br <= b;
12     cr <= c;
13     dr <= d;
14     xr <= x;
15
16     if x = '1' then
17       yr <= std_ulogic_vector(
18         unsigned(br) +
19         unsigned(cr)
20       );
21     else
22       yr <= br and cr;
23     end if;
24     drr <= dr;
25   end if;
26 end process;
27
28 x <= ar xor xr;

```

From Circuits to Code and Back
 Circuit → VHDL
 Restructured Architecture

```

1 process (clk, res_n) is
2 begin
3   if res_n = '1' then
4     ar <= '0'; xr <= '0';
5     dr <= '0'; drr <= '0';
6     br <= (others => '0');
7     cr <= (others => '0');
8     yr <= (others => '0');
9     elsif rising_edge(clk) then
10      ar <= a;
11      br <= b;
12      cr <= c;
13      dr <= d;
14      xr <= ar xor xr;
15    end if;
16  end process;
17 end architecture;

```

Finally, we can also remove the signal `x` from our design, by directly calculating the XOR function in the synchronous process as well.

Restructured Architecture

```

1 process (clk, res_n)
2 begin
3   if res_n = '1' then
4     ar <= '0'; xr <= '0';
5     dr <= '0'; drr <= '0';
6     br <= (others => '0');
7     cr <= (others => '0');
8     yr <= (others => '0');
9     elsif rising_edge(clk) then
10      ar <= a;
11      br <= b;
12      cr <= c;
13      dr <= d;
14      xr <= ar xor xr;
15    end if;
16  end process;
17 end architecture;

```

From Circuits to Code and Back
 Circuit → VHDL
 Restructured Architecture

```

1 process (clk, res_n)
2 begin
3     if res_n = '1' then
4         ar <= '0'; xr <= '0';
5         dr <= '0'; drr <= '0';
6         br <= (others => '0');
7         cr <= (others => '0');
8         yr <= (others => '0');
9     elsif rising_edge(clk) then
10        ar <= a;
11        br <= b;
12        cr <= c;
13        dr <= d;
14        xr <= ar xor xr;
15        if (ar xor xr) = '1' then
16            yr <= std_ulogic_vector(
17                unsigned(br) +
18                unsigned(cr)
19            );
20
21        else
22            yr <= br and cr;
23        end if;
24        drr <= dr;
25        end if;
26    end process;
27
28    process (all)
29        variable m : integer;
30    begin
31        if drr = '1' then
32            m := 2;
33        else
34            m := 4;
35        end if;
36        z <= std_ulogic_vector(
37            unsigned(yr) + m);
38    end process;

```

Here, we see the complete final resulting statement part of our architecture.

Restructured Architecture

```

1  process (clk, res_n)
2  begin
3      if res_n = '1' then
4          ar <= '0'; xr <= '0';
5          dr <= '0'; drr <= '0';
6          br <= (others => '0');
7          cr <= (others => '0');
8          yr <= (others => '0');
9      elsif rising_edge(clk) then
10         ar <= a;
11         br <= b;
12         cr <= c;
13         dr <= d;
14         xr <= ar xor xr;
15         if (ar xor xr) = '1' then
16             yr <= std_ulogic_vector(
17                 unsigned(br) +
18                 unsigned(cr)
19             );
20
21         else
22             yr <= br and cr;
23         end if;
24         drr <= dr;
25         end if;
26     end process;
27
28     process (all)
29         variable m : integer;
30     begin
31         if drr = '1' then
32             m := 2;
33         else
34             m := 4;
35         end if;
36         z <= std_ulogic_vector(
37             unsigned(yr) + m);
38     end process;

```

From Circuits to Code and Back
 Circuit → VHDL
 Restructured Architecture

```

1 process (clk, res_n)
2 begin
3     if res_n = '1' then
4         ar <= '0'; xr <= '0';
5         dr <= '0'; drr <= '0';
6         br <= (others => '0');
7         cr <= (others => '0');
8         yr <= (others => '0');
9     elsif rising_edge(clk) then
10        ar <= a;
11        br <= b;
12        cr <= c;
13        dr <= d;
14        xr <= ar xor xr;
15        if (ar xor xr) = '1' then
16            yr <= std_ulogic_vector(
17                unsigned(br) +
18                unsigned(cr)
19            );
20        else
21            yr <= br and cr;
22        end if;
23
24        drr <= dr;
25        end if;
26    end process;
27
28    process (all)
29        variable m : integer;
30    begin
31        if drr = '1' then
32            m := 2;
33        else
34            m := 4;
35        end if;
36        z <= std_ulogic_vector(
37            unsigned(yr) + m);
38    end process;
  
```

Please note that it is NOT possible to also move the last remaining combinational process generating z into the synchronous process as well. This is because the output z is not directly provided by a register, but generated combinationaly out of the signals yr and drr.

Restructured Architecture

```

1  process (clk, res_n)
2  begin
3      if res_n = '1' then
4          ar <= '0'; xr <= '0';
5          dr <= '0'; drr <= '0';
6          br <= (others => '0');
7          cr <= (others => '0');
8          yr <= (others => '0');
9      elsif rising_edge(clk) then
10         ar <= a;
11         br <= b;
12         cr <= c;
13         dr <= d;
14         xr <= ar xor xr;
15         if (ar xor xr) = '1' then
16             yr <= std_ulogic_vector(
17                 unsigned(br) +
18                 unsigned(cr)
19             );
  
```

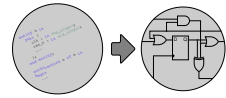
```

20         else
21             yr <= br and cr;
22         end if;
23
24         drr <= dr;
25         end if;
26     end process;
27
  
```

```

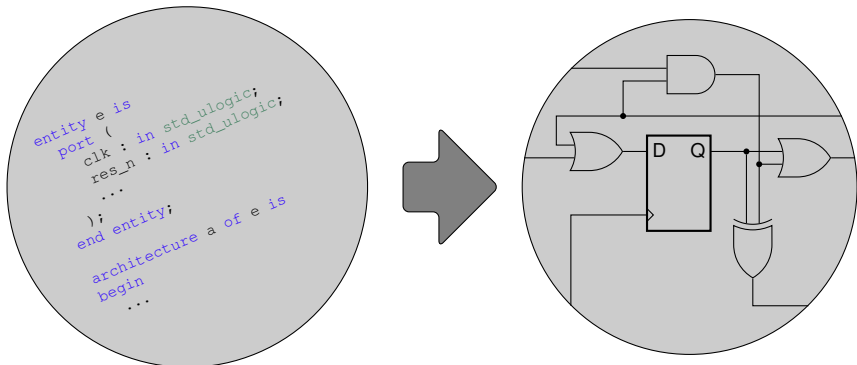
28    process (all)
29        variable m : integer;
30    begin
31        if drr = '1' then
32            m := 2;
33        else
34            m := 4;
35        end if;
36        z <= std_ulogic_vector(
37            unsigned(yr) + m);
38    end process;
  
```

└ From Circuits to Code and Back
└ VHDL → Circuit
└ **Example 2: VHDL Code → Circuit**



Let us now move to the second example, where we take a VHDL design and derive a circuit diagram from it.

Example 2: VHDL Code → Circuit



HWMod
WS25

C2C
Introduction
Circuit → VHDL
VHDL → Circuit
Entity
Architecture
Circuit

```
1 entity code2circuit is
2     generic (
3         N, M : positive
4     );
5     port (
6         clk : in std_ulogic;
7         res_n : in std_ulogic;
8         a : in std_ulogic;
9         b : in std_ulogic_vector(N-1 downto 0);
10        c : in std_ulogic_vector(M-1 downto 0);
11        d : in std_ulogic_vector(M-1 downto 0);
12        x : out std_ulogic;
13        y : out std_ulogic_vector(M-1 downto 0)
14    );
15 end entity;
```

This slide shows the entity declaration of our example circuit. As with the previous example the design does not serve any particular purpose, other than being an example.

Example 2: Entity

HWMoD
WS25

C2C
Introduction
Circuit → VHDL
VHDL → Circuit
Entity
Architecture
Circuit

```
1 entity code2circuit is
2     generic (
3         N, M : positive
4     );
5     port (
6         clk : in std_ulogic;
7         res_n : in std_ulogic;
8         a : in std_ulogic;
9         b : in std_ulogic_vector(N-1 downto 0);
10        c : in std_ulogic_vector(M-1 downto 0);
11        d : in std_ulogic_vector(M-1 downto 0);
12        x : out std_ulogic;
13        y : out std_ulogic_vector(M-1 downto 0)
14    );
15 end entity;
```

```
1 entity code2circuit is
2     generic (
3         N, M : positive
4     );
5     port (
6         clk : in std_ulogic;
7         res_n : in std_ulogic;
8         a : in std_ulogic;
9         b : in std_ulogic_vector(N-1 downto 0);
10        c : in std_ulogic_vector(M-1 downto 0);
11        d : in std_ulogic_vector(M-1 downto 0);
12        x : out std_ulogic;
13        y : out std_ulogic_vector(M-1 downto 0)
14    );
15 end entity;
```

The entity has two generics M and N that define the widths of the inputs b , c and d as well as of the output y .

Example 2: Entity

HWMoD
WS25

C2C
Introduction
Circuit → VHDL
VHDL → Circuit
Entity
Architecture
Circuit

```
1 entity code2circuit is
2     generic (
3         N, M : positive
4     );
5     port (
6         clk : in std_ulogic;
7         res_n : in std_ulogic;
8         a : in std_ulogic;
9         b : in std_ulogic_vector(N-1 downto 0);
10        c : in std_ulogic_vector(M-1 downto 0);
11        d : in std_ulogic_vector(M-1 downto 0);
12        x : out std_ulogic;
13        y : out std_ulogic_vector(M-1 downto 0)
14    );
15 end entity;
```

From Circuits to Code and Back

VHDL → Circuit

Example 2: Entity

```
1 entity code2circuit is
2   generic (
3     N, M : positive
4   );
5   port (
6     clk : in std_ulogic;
7     res_n : in std_ulogic;
8     a : in std_ulogic;
9     b : in std_ulogic_vector(N-1 downto 0);
10    c : in std_ulogic_vector(M-1 downto 0);
11    d : in std_ulogic_vector(M-1 downto 0);
12    x : out std_ulogic;
13    y : out std_ulogic_vector(M-1 downto 0)
14  );
15 end entity;
```

It also has a clock and reset input, indicating that it contains some sequential circuit elements.

Example 2: Entity

HWMoD
WS25

C2C
Introduction
Circuit → VHDL
VHDL → Circuit
Entity
Architecture
Circuit

```
1 entity code2circuit is
2   generic (
3     N, M : positive
4   );
5   port (
6     clk : in std_ulogic;
7     res_n : in std_ulogic;
8     a : in std_ulogic;
9     b : in std_ulogic_vector(N-1 downto 0);
10    c : in std_ulogic_vector(M-1 downto 0);
11    d : in std_ulogic_vector(M-1 downto 0);
12    x : out std_ulogic;
13    y : out std_ulogic_vector(M-1 downto 0)
14  );
15 end entity;
```

From Circuits to Code and Back

VHDL → Circuit

Example 2: Entity

```
1 entity code2circuit is
2     port (
3         clk : in std_ulogic;
4         res_n : in std_ulogic;
5         a : in std_ulogic;
6         b : in std_ulogic_vector(N-1 downto 0);
7         c : in std_ulogic_vector(M-1 downto 0);
8         d : in std_ulogic_vector(M-1 downto 0);
9         x : out std_ulogic;
10        y : out std_ulogic_vector(M-1 downto 0);
11    );
12 end entity;
```

Finally, the entity has an input *a* and an output *x*, both being single-bit signals.

Example 2: Entity

HWMod
WS25

C2C
Introduction
Circuit → VHDL
VHDL → Circuit
Entity
Architecture
Circuit

```
1 entity code2circuit is
2     generic (
3         N, M : positive
4     );
5     port (
6         clk : in std_ulogic;
7         res_n : in std_ulogic;
8         a : in std_ulogic;
9         b : in std_ulogic_vector(N-1 downto 0);
10        c : in std_ulogic_vector(M-1 downto 0);
11        d : in std_ulogic_vector(M-1 downto 0);
12        x : out std_ulogic;
13        y : out std_ulogic_vector(M-1 downto 0);
14    );
15 end entity;
```

From Circuits to Code and Back

VHDL → Circuit

Example 2: Architecture

```
1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c' subtype;
4   temp : std_logic;
5   process (clk, res_n)
6   begin
7     if res_n = '0' then
8       r0 <= '0'; r1 <= '0'; x <= '0';
9       r2 <= (others => '0');
10    elsif rising_edge(clk) then
11      r0 <= a xor r1;
12      r1 <= r0;
13      x <= r1;
14      r2 <= std_ulogic_vector(
15        signed(c) + signed(d)
16      );
17    end if;
18  end process;
19 end arch;
```

Let us now look at a given architecture for this entity. We can see that it consists of two processes.

Example 2: Architecture

```
1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c' subtype;
4 begin
5   process (clk, res_n)
6   begin
7     if res_n = '0' then
8       r0 <= '0'; r1 <= '0'; x <= '0';
9       r2 <= (others => '0');
10    elsif rising_edge(clk) then
11      r0 <= a xor r1;
12      r1 <= r0;
13      x <= r1;
14      r2 <= std_ulogic_vector(
15        signed(c) + signed(d)
16      );
17    end if;
18  end process;
19
20   process(all)
21   variable temp : std_ulogic;
22   begin
23     temp := '0';
24     for i in b'range loop
25       temp := temp or b(i);
26     end loop;
27     y <= r2;
28
29     if temp and r0 then
30       y <= std_ulogic_vector(
31         to_unsigned(3, y'length)
32       );
33     end if;
34   end process;
35
36 end architecture;
```

From Circuits to Code and Back

VHDL → Circuit

Example 2: Architecture

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c'subtype;
4 begin
5   process (clk, res_n)
6   begin
7     if res_n = '0' then
8       r0 <= '0'; r1 <= '0'; x <= '0';
9       r2 <= (others => '0');
10    elsif rising_edge(clk) then
11      r0 <= a xor r1;
12      r1 <= r0;
13      x <= r1;
14      r2 <= std_ulogic_vector(
15        signed(c) + signed(d)
16      );
17    end if;
18  end process;
19 end architecture

```

The process on the left describes sequential circuit elements, as indicated by the usual code pattern for D flip-flops.

Example 2: Architecture

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c'subtype;
4 begin
5   process (clk, res_n)
6   begin
7     if res_n = '0' then
8       r0 <= '0'; r1 <= '0'; x <= '0';
9       r2 <= (others => '0');
10    elsif rising_edge(clk) then
11      r0 <= a xor r1;
12      r1 <= r0;
13      x <= r1;
14      r2 <= std_ulogic_vector(
15        signed(c) + signed(d)
16      );
17    end if;
18  end process;
19
20 process(all)
21   variable temp : std_ulogic;
22 begin
23   temp := '0';
24   for i in b'range loop
25     temp := temp or b(i);
26   end loop;
27   y <= r2;
28
29   if temp and r0 then
30     y <= std_ulogic_vector(
31       to_unsigned(3, y'length)
32     );
33   end if;
34 end process;
35
36 end architecture;

```

HWMMod
WS25

C2C
Introduction
Circuit → VHDL
VHDL → Circuit
Entity
Architecture
Circuit

From Circuits to Code and Back

VHDL → Circuit

Example 2: Architecture

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : std_ulogic;
4   process (clk, res_n)
5   begin
6     if res_n = '0' then
7       r0 <= '0'; r1 <= '0'; x <= '0';
8       r2 <= (others => '0');
9     elsif rising_edge(clk) then
10      r0 <= a xor r1;
11      r1 <= r0;
12      x <= r1;
13      r2 <= std_ulogic_vector(
14        signed(c) + signed(d)
15      );
16    end if;
17  end process;
18 end architecture

```

The only really noteworthy thing is the declaration of the `r2` signal. Here the `subtype` attribute is used on the input `c` to extract its fully-constrained type information. This is a neat way to declare a signal, variable or constant with the same type as an already declared object and saves us from re-specifying the whole type information. Unfortunately this construct is not supported by all synthesis tools, but works fine in simulation. We still wanted to include it, such that you have seen this feature once.

Example 2: Architecture

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c' subtype;
4 begin
5   process (clk, res_n)
6   begin
7     if res_n = '0' then
8       r0 <= '0'; r1 <= '0'; x <= '0';
9       r2 <= (others => '0');
10      elsif rising_edge(clk) then
11        r0 <= a xor r1;
12        r1 <= r0;
13        x <= r1;
14        r2 <= std_ulogic_vector(
15          signed(c) + signed(d)
16        );
17      end if;
18    end process;
19
20    process(all)
21    variable temp : std_ulogic;
22    begin
23      temp := '0';
24      for i in b'range loop
25        temp := temp or b(i);
26      end loop;
27      y <= r2;
28
29      if temp and r0 then
30        y <= std_ulogic_vector(
31          to_unsigned(3, y'length)
32        );
33      end if;
34    end process;
35
36  end architecture;

```

From Circuits to Code and Back

VHDL → Circuit

Example 2: Architecture

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c'subtype;
4   process (clk, res_n)
5   begin
6     if res_n = '0' then
7       r0 <= '0'; r1 <= '0'; x <= '0';
8       r2 <= (others => '0');
9     elsif rising_edge(clk) then
10      r0 <= a xor r1;
11      r1 <= r0;
12      x <= r1;
13      r2 <= std_ulogic_vector(
14        signed(c) + signed(d)
15      );
16    end if;
17  end process;
18 end architecture

```

Pause the video at this point and think about how the circuit can look like. You can also make a quick paper sketch.

Example 2: Architecture

- HWMMod
- WS25
- C2C
- Introduction
- Circuit → VHDL
- VHDL → Circuit
- Entity
- Architecture
- Circuit

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c'subtype;
4 begin
5   process (clk, res_n)
6   begin
7     if res_n = '0' then
8       r0 <= '0'; r1 <= '0'; x <= '0';
9       r2 <= (others => '0');
10    elsif rising_edge(clk) then
11      r0 <= a xor r1;
12      r1 <= r0;
13      x <= r1;
14      r2 <= std_ulogic_vector(
15        signed(c) + signed(d)
16      );
17    end if;
18  end process;
19
20  process(all)
21  variable temp : std_ulogic;
22  begin
23    temp := '0';
24    for i in b'range loop
25      temp := temp or b(i);
26    end loop;
27    y <= r2;
28
29    if temp and r0 then
30      y <= std_ulogic_vector(
31        to_unsigned(3, y'length)
32      );
33    end if;
34  end process;
35
36 end architecture;

```

From Circuits to Code and Back

VHDL → Circuit

Example 2: Circuit - Registers

```
1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c'subtype;
4 begin
5   process (clk, res_n)
6   begin
7     if res_n = '0' then
8       r0 <= '0'; r1 <= '0'; x <= '0';
9       r2 <= (others => '0');
10    elsif rising_edge(clk) then
11      r0 <= a xor r1;
12      r1 <= r0;
13      x <= r1;
14      r2 <= std_ulogic_vector(
15        signed(c) + signed(d)
16      );
17    end if;
18  end process;
```

Recall that when going from a circuit diagram to VHDL code, we started by implementing the sequential circuit elements. This is also our starting point for deriving a circuit from VHDL code.

Example 2: Circuit - Registers

```
1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c'subtype;
4 begin
5   process (clk, res_n)
6   begin
7     if res_n = '0' then
8       r0 <= '0'; r1 <= '0'; x <= '0';
9       r2 <= (others => '0');
10    elsif rising_edge(clk) then
11      r0 <= a xor r1;
12      r1 <= r0;
13      x <= r1;
14      r2 <= std_ulogic_vector(
15        signed(c) + signed(d)
16      );
17    end if;
18  end process;
```

HWMMod
WS25

C2C
Introduction
Circuit → VHDL
VHDL → Circuit
Entity
Architecture
Circuit

From Circuits to Code and Back

VHDL → Circuit

Example 2: Circuit - Registers

```
1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c'subtype;
4 begin
5   process (clk, res_n)
6   begin
7     if res_n = '0' then
8       r0 <= '0'; r1 <= '0'; x <= '0';
9       r2 <= (others => '0');
10    elsif rising_edge(clk) then
11      r0 <= a xor r1;
12      r1 <= r0;
13      x <= r1;
14      r2 <= std_ulogic_vector(
15        signed(c) + signed(d)
16      );
17    end if;
18  end process;
19 end arch;
```

Hence, we first identify all processes that describe sequential circuit elements. As already mentioned, in our example there is only one such process.

Example 2: Circuit - Registers

```
1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c'subtype;
4 begin
5   process (clk, res_n)
6   begin
7     if res_n = '0' then
8       r0 <= '0'; r1 <= '0'; x <= '0';
9       r2 <= (others => '0');
10    elsif rising_edge(clk) then
11      r0 <= a xor r1;
12      r1 <= r0;
13      x <= r1;
14      r2 <= std_ulogic_vector(
15        signed(c) + signed(d)
16      );
17    end if;
18  end process;
```

HWMMod
WS25

C2C
Introduction
Circuit → VHDL
VHDL → Circuit
Entity
Architecture
Circuit

From Circuits to Code and Back

VHDL → Circuit

Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c'subtype;
4   begin
5     process (clk, res_n)
6     begin
7       if res_n = '0' then
8         r0 <= '0'; r1 <= '0'; x <= '0';
9         r2 <= (others => '0');
10      elsif rising_edge(clk) then
11        r0 <= a xor r1;
12        r1 <= r0;
13        x <= r1;
14        r2 <= std_ulogic_vector(
15          signed(c) + signed(d)
16        );
17      end if;
18    end process;

```



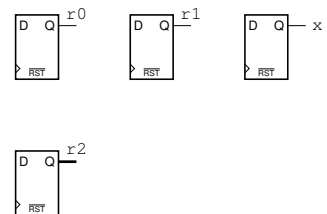
Looking at this process we first identify all the signals the process writes to, as these signals represent the outputs of flip-flops. We add registers with these signals as their outputs to our circuit diagram. If you perform such a conversion on a piece of paper, be sure to leave enough space between the elements, such that you can add in the combinational logic in the next step.

Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c'subtype;
4   begin
5     process (clk, res_n)
6     begin
7       if res_n = '0' then
8         r0 <= '0'; r1 <= '0'; x <= '0';
9         r2 <= (others => '0');
10      elsif rising_edge(clk) then
11        r0 <= a xor r1;
12        r1 <= r0;
13        x <= r1;
14        r2 <= std_ulogic_vector(
15          signed(c) + signed(d)
16        );
17      end if;
18    end process;

```



HWMoD
WS25

C2C
Introduction
Circuit → VHDL
VHDL → Circuit
Entity
Architecture
Circuit

From Circuits to Code and Back

VHDL → Circuit

Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c' subtype;
4 begin
5   process (clk, res_n)
6   begin
7     if res_n = '0' then
8       r0 <= '0'; r1 <= '0'; x <= '0';
9       r2 <= (others => '0');
10    elsif rising_edge(clk) then
11      r0 <= a xor r1;
12      r1 <= r0;
13      x <= r1;
14      r2 <= std_ulogic_vector(
15        signed(c) + signed(d)
16      );
17    end if;
18  end process;

```



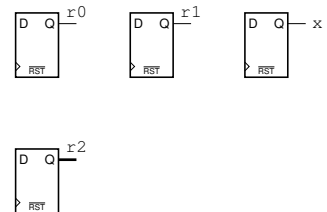
The reset code shows that the reset is active-low and that everything must be initialized to zero. As with the other example we don't draw the clock nor the reset signal, as not to clutter the figure with too many connections. With the flip-flops in place, we can now connect their inputs to the appropriate sources.

Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c' subtype;
4 begin
5   process (clk, res_n)
6   begin
7     if res_n = '0' then
8       r0 <= '0'; r1 <= '0'; x <= '0';
9       r2 <= (others => '0');
10    elsif rising_edge(clk) then
11      r0 <= a xor r1;
12      r1 <= r0;
13      x <= r1;
14      r2 <= std_ulogic_vector(
15        signed(c) + signed(d)
16      );
17    end if;
18  end process;

```



From Circuits to Code and Back

VHDL → Circuit

Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : std_ulogic;
4   signal clk, res_n;
5   begin
6     r0 <= '0'; r1 <= '0'; x <= '0';
7     r2 <= (others => '0');
8     if rising_edge(clk) then
9       r0 <= a xor r1;
10      r1 <= r0;
11      x <= r1;
12      r2 <= std_ulogic_vector(
13        signed(c) + signed(d)
14      );
15    end if;
16  end process;

```



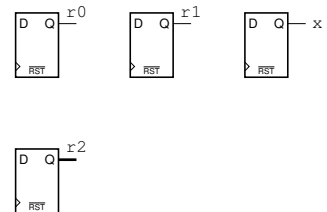
Let's start with `r1`. The respective assignment in the synchronous process specifies that this signal gets assigned the value of `r0`.

Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : std_ulogic;
4 begin
5   process (clk, res_n)
6   begin
7     if res_n = '0' then
8       r0 <= '0'; r1 <= '0'; x <= '0';
9       r2 <= (others => '0');
10    elsif rising_edge(clk) then
11      r0 <= a xor r1;
12      r1 <= r0;
13      x <= r1;
14      r2 <= std_ulogic_vector(
15        signed(c) + signed(d)
16      );
17    end if;
18  end process;

```



HWMoD
WS25

C2C
Introduction
Circuit → VHDL
VHDL → Circuit
Entity
Architecture
Circuit

From Circuits to Code and Back
 └─ VHDL → Circuit
 └─ Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c'subtype;
4 begin
5   process (clk, res_n)
6   begin
7     if res_n = '0' then
8       r0 <= '0'; r1 <= '0'; x <= '0';
9       r2 <= (others => '0');
10    elsif rising_edge(clk) then
11      r0 <= a xor r1;
12      r1 <= r0;
13      x <= r1;
14      r2 <= std_ulogic_vector(
15        signed(c) + signed(d)
16      );
17    end if;
18  end process;
  
```

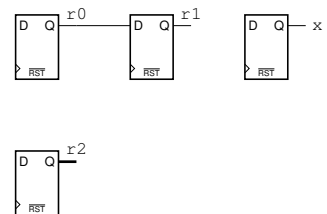


Hence, we simply add an appropriate connection to our circuit diagram.

Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c'subtype;
4 begin
5   process (clk, res_n)
6   begin
7     if res_n = '0' then
8       r0 <= '0'; r1 <= '0'; x <= '0';
9       r2 <= (others => '0');
10    elsif rising_edge(clk) then
11      r0 <= a xor r1;
12      r1 <= r0;
13      x <= r1;
14      r2 <= std_ulogic_vector(
15        signed(c) + signed(d)
16      );
17    end if;
18  end process;
  
```



From Circuits to Code and Back
 VHDL → Circuit
Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c'subtype;
4 begin
5   process (clk, res_n)
6   begin
7     if res_n = '0' then
8       r0 <= '0'; r1 <= '0'; r2 <= '0';
9       r2 <= (others => '0');
10    elsif rising_edge(clk) then
11      r0 <= a xor r1;
12      r1 <= r0;
13      x <= r1;
14      r2 <= std_ulogic_vector(
15        signed(c) + signed(d)
16      );
17    end if;
18  end process;
  
```

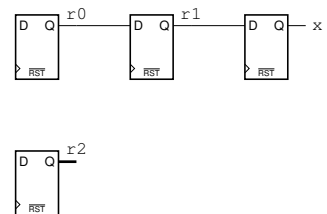


The signal *x* is handled in the exact same way.

Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c'subtype;
4 begin
5   process (clk, res_n)
6   begin
7     if res_n = '0' then
8       r0 <= '0'; r1 <= '0'; x <= '0';
9       r2 <= (others => '0');
10    elsif rising_edge(clk) then
11      r0 <= a xor r1;
12      r1 <= r0;
13      x <= r1;
14      r2 <= std_ulogic_vector(
15        signed(c) + signed(d)
16      );
17    end if;
18  end process;
  
```



HWMoD
WS25

C2C
Introduction
Circuit → VHDL
VHDL → Circuit
Entity
Architecture
Circuit

From Circuits to Code and Back

VHDL → Circuit

Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : std_ulogic;
4   signal clk, res_n;
5   process (clk, res_n)
6     if res_n = '0' then
7       r0 <= '0'; r1 <= '0'; r2 <= '0';
8     else if rising_edge(clk) then
9       r0 <= a xor r1;
10      r1 <= r0;
11      r2 <= signed(c) + signed(d);
12    end if;
13  end process;

```



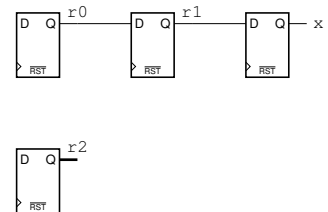
For `r0` the situation is slightly different, as here the right-hand side of the assignment contains an XOR-operation of two signals. However, the respective part of the circuit is still quite straight-forward to implement.

Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c' subtype;
4 begin
5   process (clk, res_n)
6     begin
7       if res_n = '0' then
8         r0 <= '0'; r1 <= '0'; x <= '0';
9         r2 <= (others => '0');
10      elsif rising_edge(clk) then
11        r0 <= a xor r1;
12        r1 <= r0;
13        x <= r1;
14        r2 <= std_ulogic_vector(
15          signed(c) + signed(d)
16        );
17      end if;
18    end process;

```



From Circuits to Code and Back

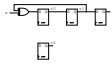
VHDL → Circuit

Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : std_ulogic;
4   signal clk, res_n;
5   signal a, b;
6   signal x;
7   signal c, d;
8   signal r0_q, r1_q, r2_q;
9   signal r0_d, r1_d, r2_d;
10  signal r0_rst, r1_rst, r2_rst;
11  signal r0_en, r1_en, r2_en;
12  signal r0_clr, r1_clr, r2_clr;
13  signal r0_set, r1_set, r2_set;
14  signal r0_ena, r1_ena, r2_ena;
15  signal r0_enb, r1_enb, r2_enb;
16  signal r0_ena, r1_ena, r2_ena;
17  signal r0_ene, r1_ene, r2_ene;
18  signal r0_eneb, r1_eneb, r2_eneb;
19  signal r0_eneb, r1_eneb, r2_eneb;
20  signal r0_eneb, r1_eneb, r2_eneb;
21  signal r0_eneb, r1_eneb, r2_eneb;
22  signal r0_eneb, r1_eneb, r2_eneb;
23  signal r0_eneb, r1_eneb, r2_eneb;
24  signal r0_eneb, r1_eneb, r2_eneb;
25  signal r0_eneb, r1_eneb, r2_eneb;
26  signal r0_eneb, r1_eneb, r2_eneb;
27  signal r0_eneb, r1_eneb, r2_eneb;
28  signal r0_eneb, r1_eneb, r2_eneb;
29  signal r0_eneb, r1_eneb, r2_eneb;
30  signal r0_eneb, r1_eneb, r2_eneb;
31  signal r0_eneb, r1_eneb, r2_eneb;
32  signal r0_eneb, r1_eneb, r2_eneb;
33  signal r0_eneb, r1_eneb, r2_eneb;
34  signal r0_eneb, r1_eneb, r2_eneb;
35  signal r0_eneb, r1_eneb, r2_eneb;
36  signal r0_eneb, r1_eneb, r2_eneb;
37  signal r0_eneb, r1_eneb, r2_eneb;
38  signal r0_eneb, r1_eneb, r2_eneb;
39  signal r0_eneb, r1_eneb, r2_eneb;
40  signal r0_eneb, r1_eneb, r2_eneb;
41  signal r0_eneb, r1_eneb, r2_eneb;
42  signal r0_eneb, r1_eneb, r2_eneb;
43  signal r0_eneb, r1_eneb, r2_eneb;
44  signal r0_eneb, r1_eneb, r2_eneb;
45  signal r0_eneb, r1_eneb, r2_eneb;
46  signal r0_eneb, r1_eneb, r2_eneb;
47  signal r0_eneb, r1_eneb, r2_eneb;
48  signal r0_eneb, r1_eneb, r2_eneb;
49  signal r0_eneb, r1_eneb, r2_eneb;
50  signal r0_eneb, r1_eneb, r2_eneb;
51  signal r0_eneb, r1_eneb, r2_eneb;
52  signal r0_eneb, r1_eneb, r2_eneb;
53  signal r0_eneb, r1_eneb, r2_eneb;
54  signal r0_eneb, r1_eneb, r2_eneb;
55  signal r0_eneb, r1_eneb, r2_eneb;
56  signal r0_eneb, r1_eneb, r2_eneb;
57  signal r0_eneb, r1_eneb, r2_eneb;
58  signal r0_eneb, r1_eneb, r2_eneb;
59  signal r0_eneb, r1_eneb, r2_eneb;
60  signal r0_eneb, r1_eneb, r2_eneb;
61  signal r0_eneb, r1_eneb, r2_eneb;
62  signal r0_eneb, r1_eneb, r2_eneb;
63  signal r0_eneb, r1_eneb, r2_eneb;
64  signal r0_eneb, r1_eneb, r2_eneb;
65  signal r0_eneb, r1_eneb, r2_eneb;
66  signal r0_eneb, r1_eneb, r2_eneb;
67  signal r0_eneb, r1_eneb, r2_eneb;
68  signal r0_eneb, r1_eneb, r2_eneb;
69  signal r0_eneb, r1_eneb, r2_eneb;
70  signal r0_eneb, r1_eneb, r2_eneb;
71  signal r0_eneb, r1_eneb, r2_eneb;
72  signal r0_eneb, r1_eneb, r2_eneb;
73  signal r0_eneb, r1_eneb, r2_eneb;
74  signal r0_eneb, r1_eneb, r2_eneb;
75  signal r0_eneb, r1_eneb, r2_eneb;
76  signal r0_eneb, r1_eneb, r2_eneb;
77  signal r0_eneb, r1_eneb, r2_eneb;
78  signal r0_eneb, r1_eneb, r2_eneb;
79  signal r0_eneb, r1_eneb, r2_eneb;
80  signal r0_eneb, r1_eneb, r2_eneb;
81  signal r0_eneb, r1_eneb, r2_eneb;
82  signal r0_eneb, r1_eneb, r2_eneb;
83  signal r0_eneb, r1_eneb, r2_eneb;
84  signal r0_eneb, r1_eneb, r2_eneb;
85  signal r0_eneb, r1_eneb, r2_eneb;
86  signal r0_eneb, r1_eneb, r2_eneb;
87  signal r0_eneb, r1_eneb, r2_eneb;
88  signal r0_eneb, r1_eneb, r2_eneb;
89  signal r0_eneb, r1_eneb, r2_eneb;
90  signal r0_eneb, r1_eneb, r2_eneb;
91  signal r0_eneb, r1_eneb, r2_eneb;
92  signal r0_eneb, r1_eneb, r2_eneb;
93  signal r0_eneb, r1_eneb, r2_eneb;
94  signal r0_eneb, r1_eneb, r2_eneb;
95  signal r0_eneb, r1_eneb, r2_eneb;
96  signal r0_eneb, r1_eneb, r2_eneb;
97  signal r0_eneb, r1_eneb, r2_eneb;
98  signal r0_eneb, r1_eneb, r2_eneb;
99  signal r0_eneb, r1_eneb, r2_eneb;
100 signal r0_eneb, r1_eneb, r2_eneb;

```



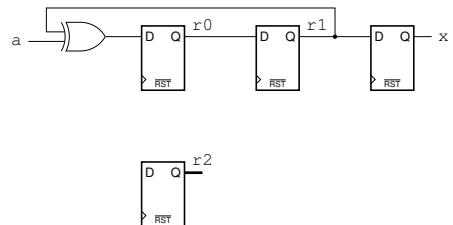
We simply add an XOR gate with the appropriate input signals to the input of the flip-flop that drives r0.

Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c' subtype;
4 begin
5   process (clk, res_n)
6     begin
7       if res_n = '0' then
8         r0 <= '0'; r1 <= '0'; x <= '0';
9         r2 <= (others => '0');
10      elsif rising_edge(clk) then
11        r0 <= a xor r1;
12        r1 <= r0;
13        x <= r1;
14        r2 <= std_ulogic_vector(
15          signed(c) + signed(d)
16        );
17      end if;
18    end process;

```



HWMMod
WS25

C2C
Introduction
Circuit → VHDL
VHDL → Circuit
Entity
Architecture
Circuit

From Circuits to Code and Back

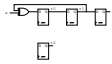
VHDL → Circuit

Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : std_ulogic;
4   signal clk, res_n;
5   begin
6     r0 <= '0'; r1 <= '0';
7     r2 <= (others => '0');
8     r0 <= a xor r1;
9     r1 <= r0;
10    r2 <= signed(c) + signed(d);
11  end architecture;

```



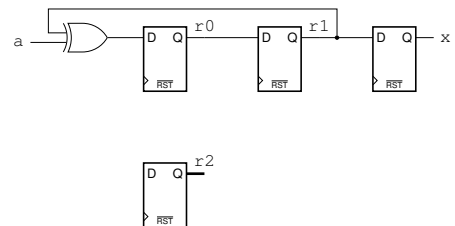
Finally, we have to deal with `r2`. However, this is essentially handled in the exact same way as the XOR gate.

Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c' subtype;
4 begin
5   process (clk, res_n)
6     begin
7       if res_n = '0' then
8         r0 <= '0'; r1 <= '0'; x <= '0';
9         r2 <= (others => '0');
10      elsif rising_edge(clk) then
11        r0 <= a xor r1;
12        r1 <= r0;
13        x <= r1;
14        r2 <= std_ulogic_vector(
15          signed(c) + signed(d)
16        );
17      end if;
18    end process;

```



From Circuits to Code and Back

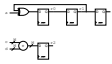
VHDL → Circuit

Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : std_ulogic;
4   signal clk, res_n;
5   process (clk, res_n)
6     if res_n = '0' then
7       r0 <= '0'; r1 <= '0'; r2 <= '0';
8     else if rising_edge(clk) then
9       r0 <= a xor r1;
10      r1 <= r0;
11      r2 <= signed(c) + signed(d);
12    end if;
13  end process;

```



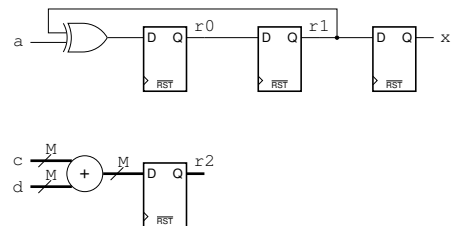
We simply add an adder with the appropriate inputs to the register that drives `r2`. Notice that the adder produces an M-bit wide signal.

Example 2: Circuit - Registers

```

1 architecture arch of code2circuit is
2   signal r0, r1 : std_ulogic;
3   signal r2 : c' subtype;
4 begin
5   process (clk, res_n)
6     begin
7       if res_n = '0' then
8         r0 <= '0'; r1 <= '0'; x <= '0';
9         r2 <= (others => '0');
10      elsif rising_edge(clk) then
11        r0 <= a xor r1;
12        r1 <= r0;
13        x <= r1;
14        r2 <= std_ulogic_vector(
15          signed(c) + signed(d)
16        );
17      end if;
18    end process;

```

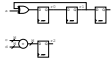


From Circuits to Code and Back
 VHDL → Circuit
Example 2: Circuit - Combinational Process

```

19 process(all)
20   variable temp : std_ulogic;
21 begin
22   temp := '0';
23   for i in b'range loop
24     temp := temp or b(i);
25   end loop;
26
27   y <= r2;
28
29   if temp and r0 then
30     y <= std_ulogic_vector(
31       to_unsigned(3, y'length)
32     );
33   end if;
34 end process;
35
36 end architecture;

```



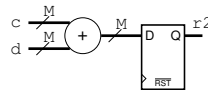
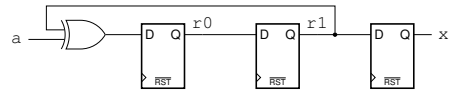
Now let's turn to the second process, which – as already mentioned – only contains combinational logic.

Example 2: Circuit - Combinational Process

```

19 process(all)
20   variable temp : std_ulogic;
21 begin
22   temp := '0';
23   for i in b'range loop
24     temp := temp or b(i);
25   end loop;
26
27   y <= r2;
28
29   if temp and r0 then
30     y <= std_ulogic_vector(
31       to_unsigned(3, y'length)
32     );
33   end if;
34 end process;
35
36 end architecture;

```



HWMoD
WS25

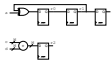
C2C
Introduction
Circuit → VHDL
VHDL → Circuit
Entity
Architecture
Circuit

From Circuits to Code and Back
 VHDL → Circuit
Example 2: Circuit - Combinational Process

```

19 process(all)
20   variable temp : std_ulogic;
21 begin
22   temp := '0';
23   for i in b'range loop
24     temp := temp or b(i);
25   end loop;
26
27   y <= r2;
28
29   if temp and r0 then
30     y <= std_ulogic_vector(
31       to_unsigned(3, y'length)
32     );
33   end if;
34 end process;
35
36 end architecture;

```



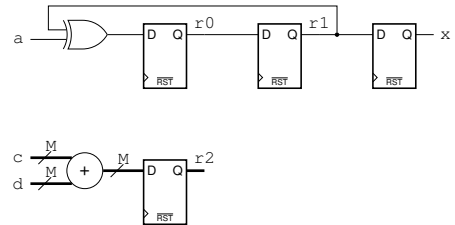
The process initially sets the temporary single-bit variable `temp` to zero. It then uses a for-loop to go over all elements of the input `b` and calculates the disjunction with `temp`, which is then again assigned to the `temp` variable. Hence, if any of the elements of `b` is one, `temp` is set to one as well and will stay at that value until the end of the loop.

Example 2: Circuit - Combinational Process

```

19 process(all)
20   variable temp : std_ulogic;
21 begin
22   temp := '0';
23   for i in b'range loop
24     temp := temp or b(i);
25   end loop;
26
27   y <= r2;
28
29   if temp and r0 then
30     y <= std_ulogic_vector(
31       to_unsigned(3, y'length)
32     );
33   end if;
34 end process;
35
36 end architecture;

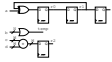
```



From Circuits to Code and Back
 VHDL → Circuit
Example 2: Circuit - Combinational Process

```

19 process(all)
20   variable temp : std_ulogic;
21 begin
22   temp := '0';
23   for i in b'range loop
24     temp := temp or b(i);
25   end loop;
26
27   y <= r2;
28
29   if temp and r0 then
30     y <= std_ulogic_vector(
31       to_unsigned(3, y'length)
32     );
33   end if;
34 end process;
35
36 end architecture;
  
```



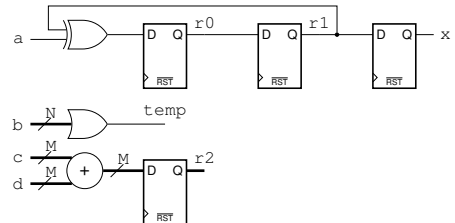
Notice how this effectively describes an N-input OR gate with `temp` as its output signal.

Example 2: Circuit - Combinational Process

HWMod
WS25

```

19 process(all)
20   variable temp : std_ulogic;
21 begin
22   temp := '0';
23   for i in b'range loop
24     temp := temp or b(i);
25   end loop;
26
27   y <= r2;
28
29   if temp and r0 then
30     y <= std_ulogic_vector(
31       to_unsigned(3, y'length)
32     );
33   end if;
34 end process;
35
36 end architecture;
  
```



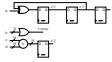
C2C
Introduction
Circuit → VHDL
VHDL → Circuit
Entity
Architecture
Circuit

From Circuits to Code and Back
 VHDL → Circuit
Example 2: Circuit - Combinational Process

```

19 process(all)
20   variable temp : std_ulogic;
21 begin
22   temp := '0';
23   for i in b'range loop
24     temp := temp or b(i);
25   end loop;
26
27   y <= r2;
28
29   if temp and r0 then
30     y <= std_ulogic_vector(
31       to_unsigned(3, y'length)
32     );
33   end if;
34 end process;
35
36 end architecture;

```



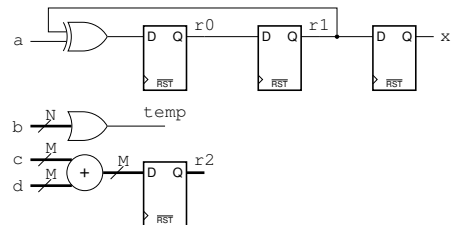
The next part of the process contains an if-statement. First `y` is assigned the value of `r2`. If the if-condition is true this value is then overridden by the constant three. Notice that it would be semantically equivalent to perform the `r2` assignment in the else-branch of the if statement. We already know that if-statements can be expressed using multiplexers. However, before we can do that we have to generate a signal that we can feed into the control input of this multiplexer.

Example 2: Circuit - Combinational Process

```

19 process(all)
20   variable temp : std_ulogic;
21 begin
22   temp := '0';
23   for i in b'range loop
24     temp := temp or b(i);
25   end loop;
26
27   y <= r2;
28
29   if temp and r0 then
30     y <= std_ulogic_vector(
31       to_unsigned(3, y'length)
32     );
33   end if;
34 end process;
35
36 end architecture;

```



From Circuits to Code and Back

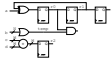
VHDL → Circuit

Example 2: Circuit - Combinational Process

```

19 process(all)
20   variable temp : std_ulogic;
21 begin
22   temp := '0';
23   for i in b'range loop
24     temp := temp or b(i);
25   end loop;
26
27   y <= r2;
28
29   if temp and r0 then
30     y <= std_ulogic_vector(
31       to_unsigned(3, y'length)
32     );
33   end if;
34 end process;
35
36 end architecture;

```



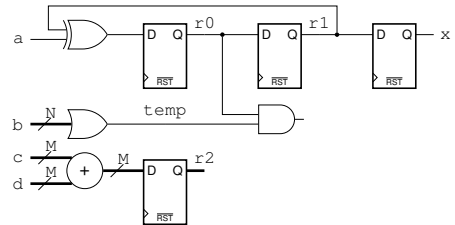
For that purpose we add an AND gate, that evaluates the conjunction used by the expression in the if-condition.

Example 2: Circuit - Combinational Process

```

19 process(all)
20   variable temp : std_ulogic;
21 begin
22   temp := '0';
23   for i in b'range loop
24     temp := temp or b(i);
25   end loop;
26
27   y <= r2;
28
29   if temp and r0 then
30     y <= std_ulogic_vector(
31       to_unsigned(3, y'length)
32     );
33   end if;
34 end process;
35
36 end architecture;

```



HWMod
WS25

C2C
Introduction
Circuit → VHDL
VHDL → Circuit
Entity
Architecture
Circuit

From Circuits to Code and Back

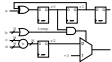
VHDL → Circuit

Example 2: Circuit - Combinational Process

```

19 process(all)
20   variable temp : std_ulogic;
21 begin
22   temp := '0';
23   for i in b'range loop
24     temp := temp or b(i);
25   end loop;
26
27   y <= r2;
28
29   if temp and r0 then
30     y <= std_ulogic_vector(
31       to_unsigned(3, y'length)
32     );
33   end if;
34 end process;
35
36 end architecture;

```



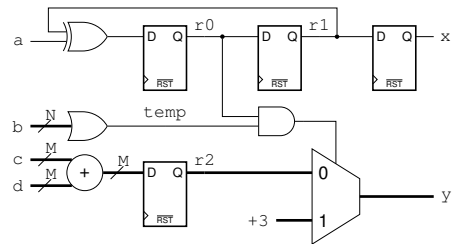
Finally we can use the output signal of this "AND" gate to control the multiplexer that produces the output y of our circuit. And with that our circuit diagram is complete.

Example 2: Circuit - Combinational Process

```

19 process(all)
20   variable temp : std_ulogic;
21 begin
22   temp := '0';
23   for i in b'range loop
24     temp := temp or b(i);
25   end loop;
26
27   y <= r2;
28
29   if temp and r0 then
30     y <= std_ulogic_vector(
31       to_unsigned(3, y'length)
32     );
33   end if;
34 end process;
35
36 end architecture;

```



HWMod
WS25

C2C
Introduction
Circuit → VHDL
VHDL → Circuit
Entity
Architecture
Circuit

Thank you for listening! We recommend you to immediately take the self-check test in TUWEL, to see if you understood the material presented in this lecture.

HWMMod
WS25

C2C
Introduction
Circuit → VHDL
VHDL → Circuit
Entity
Architecture
Circuit

Lecture Complete!