Grundlagen von VHDL

PHNÖ 3Z5B4SWJ22

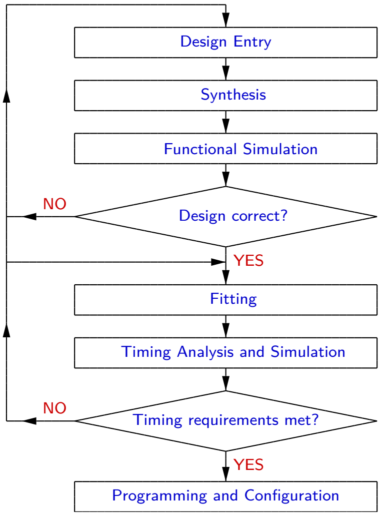
Karlheinz Oswald und Bernhard Wess

11. bis 13. März 2024

Teil I

Hardware-Beschreibung und -synthese

Typical FPGA CAD Flow



- Design Entry - The desired circuit is specified either by using a hardware description language, such as Verilog or VHDL, or by means of a schematic diagram.
- Synthesis - The CAD synthesis tool synthesizes the circuit into a netlist that gives the logic elements (LEs) needed to realize the circuit and the connections between the LEs.
- Functional Simulation - The synthesized circuit is tested to verify its functional correctness; the simulation does not take into account any timing issues.
- Fitting - The CAD fitter tool determines the placement of the LEs defined in the netlist into the LEs in an actual FPGA chip; it also chooses routing wires in the chip to make the required connections between specific LEs.
- Timing Analysis - Propagation delays along the various paths in the fitted circuit are analyzed to provide an indication of the expected performance of the circuit.
- Timing Simulation - The fitted circuit is tested to verify both its functional correctness and timing.
- Programming and Configuration - The designed circuit is implemented in a physical FPGA chip by programming the configuration switches that configure the LEs and establish the required wiring connections.

Teil II

Hardware-Beschreibungssprache VHDL

The VHDL Acronym

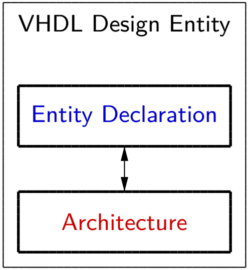
- VHDL - VHSIC Hardware Description Language
- VHSIC - Very High Speed Integrated Circuit

VHDL is a language used for

- describing the structure and behaviour of digital electronic hardware designs
- testing the functionality of a design

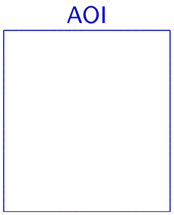
- VHDL is an international standard (1076), regulated by the IEEE.
- The definition of VHDL is non-proprietary.

VHDL allows separation of the internal and external view of a digital system.



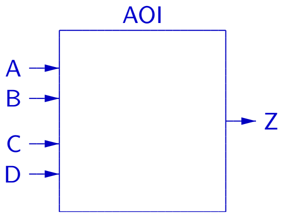
- An entity declaration describes the interface of the design to its external environment.
- An architecture body describes the functionality of a design.

```
-- entity declaration
entity AOI is

  -- place your entity
  -- declaration here

end AOI;
```



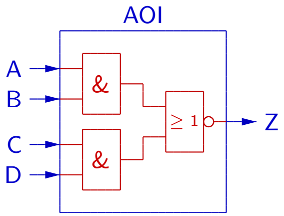
- The VHDL entity is used to give the design entity its name, in this case AOI.

```
-- entity declaration

entity AOI is

  port(A,B,C,D : in  std_logic;
       Z       : out std_logic);

end AOI;
```



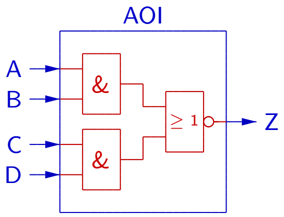
- List of ports, each port must also be given a type and an input / output mode.

```
-- entity declaration

entity AOI is
  port(A,B,C,D : in  std_logic;
       Z       : out std_logic);
end AOI;

-- architecture body

architecture AOI_a of AOI is
begin
  Z <= not((A and B) or (C and D));
end AOI_a;
```



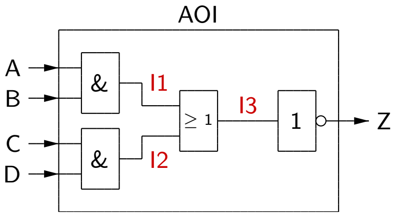
- The architecture is given a name which is different from the entity.
- It must also be associated with an entity declaration.

```
library IEEE;
use IEEE.std_logic_1164.all;

-- entity declaration
entity AOI is
  port(A,B,C,D : in  std_logic;
       Z       : out std_logic);
end AOI;

-- architecture body
architecture AOI_a of AOI is
begin
  Z <= not((A and B) or (C and D));
end AOI_a;
```



- Type std_logic comes from package std_logic_1164 of the IEEE library.
- A package is a VHDL construct in which new data types, functions, and procedures may be defined.
- std_logic_1164 is a package for representing digital signals.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity AOI is
  port(A,B,C,D : in  std_logic;
       Z       : out std_logic);
end AOI;

architecture AOI_a of AOI is
  signal I1,I2,I3 : std_logic;
begin
  I1 <= A and B;
  I2 <= C and D;
  I3 <= I1 or I2;
  Z  <= not I3;
end AOI_a;
```



- Internal signals are declared before the begin in the architecture.
- Each signal assignment represents a concurrent operation.

Library and Package

```
library ieee;
use ieee.std_logic_1164.all;
```

- adds additional types, operators, functions, etc. to VHDL

Entity Declaration

```
entity or_gate is
  port(
    a,b : in  std_logic;
    y   : out std_logic
  );
end or_gate;
```

- declares the name of the circuit
- specifies the I/O signals

Architecture Body

```
architecture dataflow of or_gate is
begin
  y <= a or b;
end dataflow;
```

- describes operation of the circuit

Signal Assignment

signal_name <= expression;

Example

```
Z <= A and B;
```

Signal Assignment

signal_name <= expression;

Conditional Signal Assignment

signal_name <=
expression **when** condition **else**
expression **when** condition **else**
expression;

Example

```
Z <= A and B;
```

Example

```
Z <= A when S="00" else
     B when S="11" else
     C;
```

Signal Assignment

signal_name <= expression;

Example

```
Z <= A and B;
```

Conditional Signal Assignment

signal_name <=
expression **when** condition **else**
expression **when** condition **else**
expression;

Example

```
Z <= A when S="00" else
     B when S="11" else
     C;
```

Selected Signal Assignment

with selection **select**
signal_name <=
expression **when** choices,
expression **when** choices,
expression **when others;**

Example

```
with S select
       Z <= A when "00",
            B when "11",
            C when others;
```

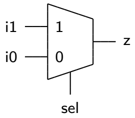
Example Statement

```
r <= a + b + c when m = n else
     a - b     when m > n else
     c + 1;
```

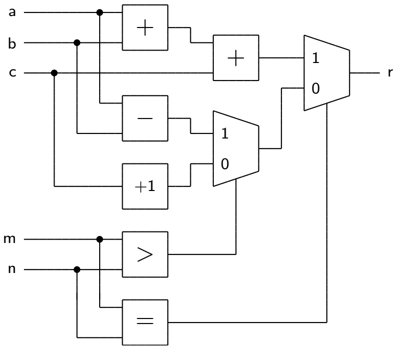
Example Statement

```
r <= a + b + c when m = n else
     a - b     when m > n else
     c + 1;
```

2-to-1 Multiplexer



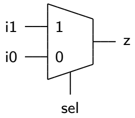
Block Diagram



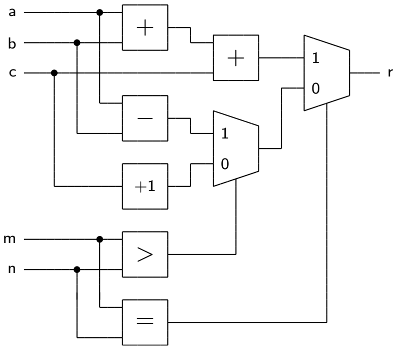
Example Statement

```
r <= a + b + c when m = n else
     a - b     when m > n else
     c + 1;
```

2-to-1 Multiplexer



Block Diagram



Note:

- Infers a priority routing structure.
- A large number of when clauses leads to a long cascading chain.

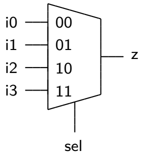
Example Statement

```
signal sel: std_logic_vector(1 downto 0);
 .
 .
 .
with sel select
  r <= a + b + c when "00",
       a - b     when "10",
       c + 1     when others;;
```

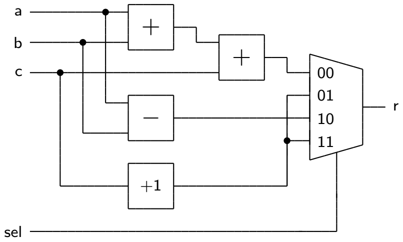
Example Statement

```
signal sel: std_logic_vector(1 downto 0);
 .
 .
 .
with sel select
  r <= a + b + c when "00",
       a - b     when "10",
       c + 1     when others;;
```

4-to-1 Multiplexer



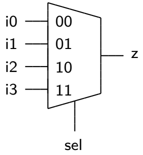
Block Diagram



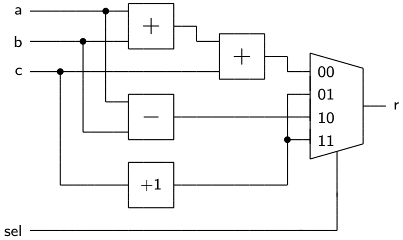
Example Statement

```
signal sel: std_logic_vector(1 downto 0);
 .
 .
 .
with sel select
  r <= a + b + c when "00",
       a - b     when "10",
       c + 1     when others;;
```

4-to-1 Multiplexer



Block Diagram



Note:

- Infers a multiplexing structure.
- A large number of choices leads to a wide multiplexer.

Format for Architecture Body

architecture architecture_name **of** entity_name **is**
    data type definitions
    internal signal declarations
    component declarations
    function and procedure declarations
begin
    component instantiations
    processes
    concurrent statements including
        signal assignment statements
        when-else statements
        with-select-when statements
end architecture architecture_name**;**

Note:

architecture in the end statement is optional.

Format for Process Statement

process_label: **process(**sensitivity_list**)**
variable variable_name: **type;**
⋮
variable variable_name: **type;**
begin
if-then-else statements
case-when statements
for-loop statements
while-loop statements
wait statements
end process process_label**;**

Note:

- process_label in the end statement is optional.
- An event on any signal of sensitivity_list cause the process to execute.
- Within the process each statement is executed sequentially and only sequential statements can be used in a process.

If-Then-Else

```
if (condition) then
    do stuff
elsif (condition) then
    do more stuff
else
    do other stuff
end if;
```

Note:

- **elsif** and **else** clauses are optional.
- Incompletely specified **if** statement (no **else**) implies a memory element.

If-Then-Else

if (condition) **then**
    do stuff
elsif (condition) **then**
    do more stuff
else
    do other stuff
end if;

Example

```
if (S="00") then
   Z <= A;
elsif (S="11") then
      Z <= B;
      else
      Z <= C;
end if;
```

Note:

- **elsif** and **else** clauses are optional.
- Incompletely specified **if** statement (no **else**) implies a memory element.

Case-When

```
case expression is
    when value =>
        do stuff
    when value =>
        do more stuff
    when others =>
        do other stuff
end case;
```

Case-When

```
case expression is
    when value =>
        do stuff
    when value =>
        do more stuff
    when others =>
        do other stuff
end case;
```

Example

```
case S is
       when "00" =>
             Z <= A;
       when "11" =>
             Z <= B;
       when others =>
             Z <= C;
end case;
```

Function Table

r	pcode
1---	100
01--	011
001-	010
0001	001
0000	000

Entity Declaration

```
entity prio_enc is
 port( r      : in  std_logic_vector(4 downto 1);
       pcode : out std_logic_vector(2 downto 0)
 );
end prio_enc;
```

Conditional Signal Assignment

```
architecture csa_behavior of prio_enc is
begin
 pcode <= "100" when (r(4)='1') else
          "011" when (r(3)='1') else
          "010" when (r(2)='1') else
          "001" when (r(1)='1') else
          "000";
end csa_behavior;
```

Selected Signal Assignment

```
architecture ssa_behavior of prio_enc is
begin
 with r select
  pcode <= "100" when "1000"|"1001"|"1010"|"1011"|
                      "1100"|"1101"|"1110"|"1111"|,
           "011" when "0100"|"0101"|"0110"|"0111"|,
           "010" when "0010"|"0011"|,
           "001" when "0001"|,
           "000" when others;
end ssa_behavior;
```

Function Table

r	pcode
1---	100
01--	011
001-	010
0001	001
0000	000

Entity Declaration

```
entity prio_enc is
 port( r      : in  std_logic_vector(4 downto 1);
       pcode : out std_logic_vector(2 downto 0)
 );
end prio_enc;
```

If Statement

```
architecture if_behavior of prio_enc is
begin
 process(r)
 begin
  if (r(4)='1') then
   pcode <= "100";
  elsif (r(3)='1') then
   pcode <= "011";
  elsif (r(2)='1') then
   pcode <= "010";
  elsif (r(1)='1') then
   pcode <= "001";
  else
   pcode <= "000";
  end if;
 end process;
end if_behavior;
```

Case Statement

```
architecture case_behavior of prio_enc is
begin
 process(r)
 begin
  case r is
   when "1000"|"1001"|"1010"|"1011"|
        "1100"|"1101"|"1110"|"1111"| =>
    pcode <= "100";
   when "0100"|"0101"|"0110"|"0111"| =>
    pcode <= "011";
   when "0010"|"0011"| =>
    pcode <= "010";
   when "0001"| =>
    pcode <= "001";
   when others =>
    pcode <= "000";
  end case;
 end process;
end case_behavior;
```

Unintended Memory

```
process(a)
begin
 if (a > b) then
  gt <= '1';
 elsif (a = b) then
  eq <= '1';
 end if;
end process;
```

Correct Code

```
process(a,b)
begin
 if (a > b) then
  gt <= '1';
  eq <= '0';
 elsif (a = b) then
  gt <= '0';
  eq <= '1';
 else
  gt <= '0';
  eq <= '0';
 end if;
end process;
```

Correct Code

```
process(a,b)
begin
  gt <= '0';
  eq <= '0';
 if (a > b) then
  gt <= '1';
 elsif (a = b) then
  eq <= '1';
 end if;
end process;
```

Rules to prevent unintended memory

- Include all input signals in the sensitivity list.
- Include the else branch in an if statement.
- Assign a value to every signal in every branch.

Predefined Data Types

- `integer`: minimal range is from $-(2^{31}-1)$ to $2^{31}-1$
  - subtypes: `natural` and `positive`
- `boolean`: defined as `(false, true)`
- `bit`: defined as `('0', '1')`
- `bit_vector`: defined als a one-dimensional array with elements of `bit` data type.

IEEE std_logic_1164 Package

- The `std_logic` data type consists of nine possible values:

  `('U', 'X', '0', '1', 'Z', 'W','L', 'H', '-')`
- The `std_logic_vector` data type is an array of elements with the `std_logic` data type.
- Include the necessary `library` and `use` statements before the `entity` declaration:

```
library ieee;
use ieee.std_logic_1164.all;
```

IEEE numeric_std Package

- New data types: `signed` and `unsigned`
  - Better control over the underlying hardware than with the `integer` data type.
- Required `library` statements:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

- Overloaded operators
  - `abs`, `*`, `/`, `mod`, `rem`, `+`, `-`
  - Operators with two operands:
    `unsigned` and `unsigned`, `unsigned` and `natural`, `signed` and `signed`, `signed` and `integer`
- Several new functions
  - `shift_left`, `shift_right`, `rotate_left`, `rotate_right`
  - Type conversion: `to_unsigned`, `to_signed`, `to_integer`

Operator	Description	Data type of a	Data type of b	Data type of result
`a ** b`	exponentiation	int	int	int
`abs a`	absolute value	int		int
`a * b`	multiplication	int	int	int
`a / b`	division	int	int	int
`a mod b`	modulo	int	int	int
`a rem b`	remainder	int	int	int
`+ a`	identity	int		int
`- a`	negation	int		int
`a + b`	addition	int	int	int
`a - b`	subtraction	int	int	int
`a & b`	concatenation	array, element	array, element	array
`a sll b`	shift left log.	bitv	int	bitv
`a srl b`	shift right log.	bitv	int	bitv
`a sla b`	shift left arith.	bitv	int	bitv
`a sra b`	shift right arith.	bitv	int	bitv
`a rol b`	rotate left	bitv	int	bitv
`a ror b`	rotate right	bitv	int	bitv

Operator	Description	Data type of a	Data type of b	Data type of result
`a = b`	equal to	any	same as a	boo
`a /= b`	not equal to	any	same as a	boo
`a < b`	less than	scalar, array	same as a	boo
`a <= b`	l. th. or eq. to	scalar, array	same as a	boo
`a > b`	greater than	scalar, array	same as a	boo
`a >= b`	g. th. or eq. to	scalar, array	same as a	boo
`not a`	negation	boo, bit, bitv		same as a
`a and b`	and	boo, bit, bitv	same as a	same as a
`a or b`	and	boo, bit, bitv	same as a	same as a
`a xor b`	and	boo, bit, bitv	same as a	same as a
`a nand b`	and	boo, bit, bitv	same as a	same as a
`a nor b`	and	boo, bit, bitv	same as a	same as a
`a xnor b`	and	boo, bit, bitv	same as a	same as a

Relational Operators

- The comparison procedure starts from the leftmost element and continues until a result can be established.
- For example, the following operations return `true`:
  `"011"="011"`, `"011">"010"`

Concatenation Operator

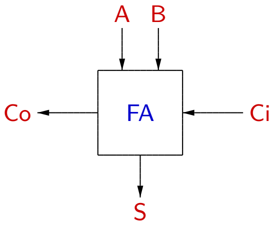
- We can combine elements and smaller arrays to form a larger array.
- Example: `"00" & a(7 downto 2)`

Array Aggregate

- Assignment: `a <= "10100000"`
- Positional Association: `a <= ('1','0','1','0','0','0','0','0')`
- Named Association:
  - `a <= (7=>'1',6=>'0',5=>'1',4=>'0',3=>'0',2=>'0',1=>'0',0=>'0')`
  - `a <= (7|5=>'1',6|4|3|2|1|0=>'0')`
  - `a <= (7|5=>'1',others=>'0')`

Teil III

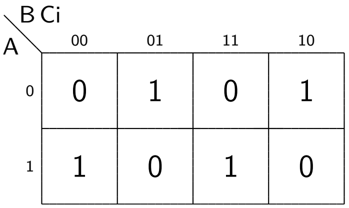
Hierarchisches VHDL-Design



Truth Table

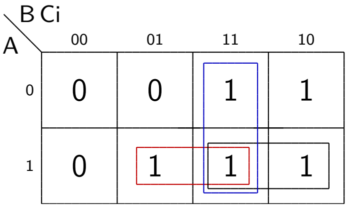
A	B	Ci	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sum



A \ B Ci	00	01	11	10
0	0	1	0	1
1	1	0	1	0

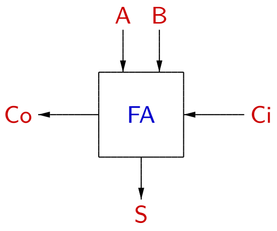
Carry Out



A \ B Ci	00	01	11	10
0	0	0	1	1
1	0	1	1	1

$$\begin{aligned} S &= A.\overline{B}.\overline{Ci} + \overline{A}.\overline{B}.Ci + A.B.Ci + \overline{A}.B.\overline{Ci} \\ &= \left(A.\overline{B} + \overline{A}.B\right).\overline{Ci} + \left(A.B + \overline{A}.\overline{B}\right).Ci \\ &= (A \oplus B).\overline{Ci} + \left(\overline{A \oplus B}\right).Ci \\ &= A \oplus B \oplus Ci \end{aligned}$$

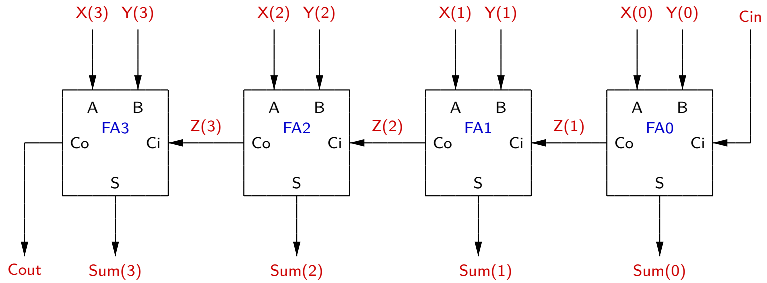
$$Co = A.B + \color{red}{A.Ci} + \color{blue}{B.Ci}$$

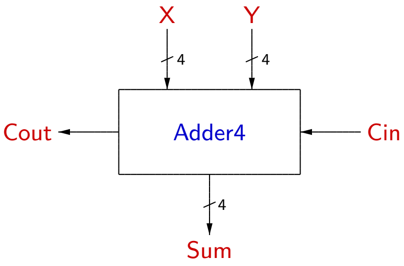


VHDL Code

```
library IEEE;
use IEEE.std_logic_1164.all;

entity FA is
  port(A,B,Ci : in  std_logic;
       S,Co   : out std_logic);
end FA;

architecture FA_a of FA is
begin
  S  <= A xor B xor Ci;
  Co <= (A and B) or (A and Ci) or (B and Ci);
end FA_a;
```

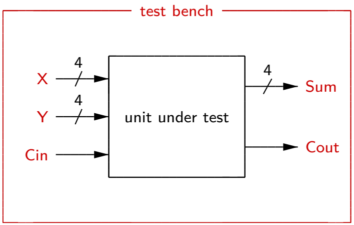
Schematic


X(3)
Y(3)
X(2)
Y(2)
X(1)
Y(1)
X(0)
Y(0)
Cin
A
B
FA3
Co
Ci
S
Z(3)
A
B
FA2
Co
Ci
S
Z(2)
A
B
FA1
Co
Ci
S
Z(1)
A
B
FA0
Co
Ci
S
Cout
Sum(3)
Sum(2)
Sum(1)
Sum(0)



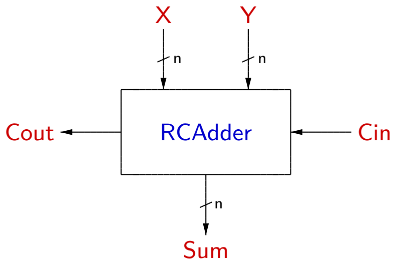
VHDL Code

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Adder4 is
  port(X, Y : in std_logic_vector(3 downto 0);
       Cin  : in std_logic;
       Sum  : out std_logic_vector(3 downto 0);
       Cout : out std_logic);
end Adder4;

architecture Adder4_a of Adder4 is

component FA
  port(A,B,Ci : in  std_logic;
         S,Co : out std_logic);
end component;

signal Z : std_logic_vector(3 downto 1);

begin
  FA0: FA port map (X(0), Y(0), Cin, Sum(0), Z(1));
  FA1: FA port map (X(1), Y(1), Z(1), Sum(1), Z(2));
  FA2: FA port map (X(2), Y(2), Z(2), Sum(2), Z(3));
  FA3: FA port map (X(3), Y(3), Z(3), Sum(3), Cout);
end Adder4_a;
```



Test Bench

```
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_adder4 is
end tb_adder4;

architecture tb_adder4_a of tb_adder4 is
  component Adder4
    port(X, Y : in std_logic_vector(3 downto 0);
         Cin  : in std_logic;
         Sum  : out std_logic_vector(3 downto 0);
         Cout : out std_logic);
  end component;
  signal X, Y : std_logic_vector(3 downto 0);
  signal Cin  : std_logic;
  signal Sum  : std_logic_vector(3 downto 0);
  signal Cout : std_logic;
begin
  UUT: Adder4 port map (X,Y,Cin,Sum,Cout);
  stimulus: process
    begin
      Cin <= '0';
      X <= "1011";
      Y <= "0100";
      wait for 100 ns;
      Cin <= '1';
      wait for 100 ns;
    end process;
end tb_adder4_a;
```



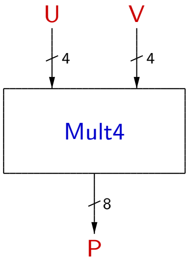
VHDL Code

```
library IEEE;
use IEEE.std_logic_1164.all;

entity RCAdder is
  generic(n : integer);
  port(X, Y : in std_logic_vector(n-1 downto 0);
       Cin  : in std_logic;
       Sum  : out std_logic_vector(n-1 downto 0);
       Cout : out std_logic);
end RCAdder;

architecture RCAdder_a of RCAdder is

component FA
  port(a,b,ci : in  std_logic;
       s,co   : out std_logic);
end component;

signal Z : std_logic_vector(n downto 0);

begin
  Z(0) <= Cin;
  Cout <= Z(n);
  AddGen: for i in 0 to n-1 generate
    FAn: FA port map (X(i), Y(i), Z(i), Sum(i), Z(i+1));
  end generate;
end RCAdder_a;
```

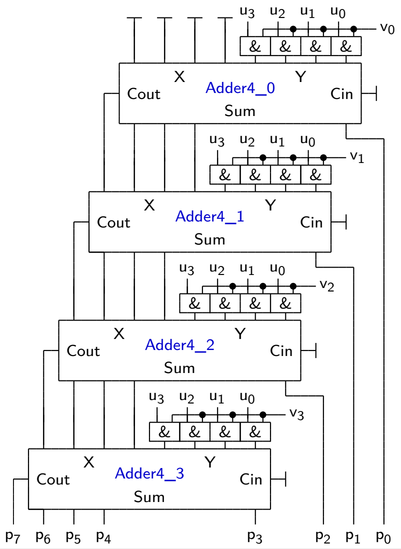
Test Bench

```
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_RCadder is
end tb_RCadder;

architecture tb_RCAdder_a of tb_RCAdder is

component RCAdder
    generic(n : integer);
    port(X, Y : in std_logic_vector(n downto 0);
         Cin  : in std_logic;
         Sum  : out std_logic_vector(n downto 0);
         Cout : out std_logic
        );
end component;

constant n : integer := 5;

signal X, Y : std_logic_vector(n-1 downto 0);
signal Cin  : std_logic;
signal Sum  : std_logic_vector(n-1 downto 0);
signal Cout : std_logic;

begin

UUT: RCAdder generic map (n) port map (X,Y,Cin,Sum,Cout);
```

```
stimulus: process
begin
  Cin <= '0';
  X <= "01010";
  Y <= "00100";
  wait for 100 ns;
  Cin <= '1';
  wait for 100 ns;
end process;

end tb_RCadder_a;
```


U
V
4
4
Mult4
8
P

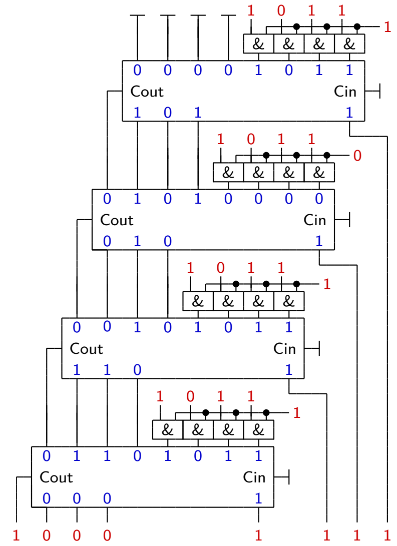
Schematic


u_3 u_2 u_1 u_0
v_0
&
Cout
X
Adder4_0
Y
Cin
Sum
v_1
Adder4_1
v_2
Adder4_2
v_3
Adder4_3
p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0

Example

```
        1 0 1 1 × 1 1 0 1
        -----------------
        1 0 1 1
      0 0 0 0
    1 0 1 1
  1 0 1 1
---------------
1 0 0 0 1 1 1 1
```

Schematic

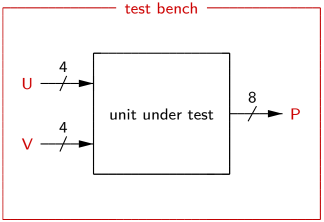


VHDL Code

VHDL Code

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mult4 is
  port(U, V : in std_logic_vector(3 downto 0);
       P    : out std_logic_vector(7 downto 0)
      );
end mult4;

architecture mult4_a of mult4 is

component and4
  port(A : in std_logic_vector(3 downto 0);
       x : in std_logic;
       Z : out std_logic_vector(3 downto 0)
      );
end component;

component Adder4
  port(X, Y : in std_logic_vector(3 downto 0);
       Cin  : in std_logic;
       Sum  : out std_logic_vector(3 downto 0);
       Cout : out std_logic
      );
end component;
```

```
signal X0, X1,
       X2, X3 : std_logic_vector(3 downto 0);
signal Y0, Y1,
       Y2, Y3 : std_logic_vector(3 downto 0);
signal S0, S1,
       S2, S3 : std_logic_vector(3 downto 0);
signal C0, C1,
       C2, C3 : std_logic;

begin
    And_0: and4 port map (U, V(0), Y0);
    And_1: and4 port map (U, V(1), Y1);
    And_2: and4 port map (U, V(2), Y2);
    And_3: and4 port map (U, V(3), Y3);
    Add_0: Adder4 port map (X0, Y0, '0', S0, C0);
    Add_1: Adder4 port map (X1, Y1, '0', S1, C1);
    Add_2: Adder4 port map (X2, Y2, '0', S2, C2);
    Add_3: Adder4 port map (X3, Y3, '0', S3, C3);
    X0 <= "0000";
    X1 <= C0&S0(3 downto 1);
    X2 <= C1&S1(3 downto 1);
    X3 <= C2&S2(3 downto 1);
    P <= C3&S3&S2(0)&S1(0)&S0(0);
end mult4_a;
```



Test Bench

```
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_mult4 is
end tb_mult4;

architecture tb_mult4_a of tb_mult4 is
  component mult4
    port(U, V : in std_logic_vector(3 downto 0);
         P    : out std_logic_vector(7 downto 0));
  end component;
  signal U, V : std_logic_vector(3 downto 0);
  signal P    : std_logic_vector(7 downto 0);
begin
  UUT: mult4 port map (U,V,P);
  stimulus: process
    begin
      U <= "1011";
      V <= "0100";
      wait for 100 ns;
      U <= "1001";
      V <= "0101";
      wait for 100 ns;
    end process;
end tb_mult4_a;
```

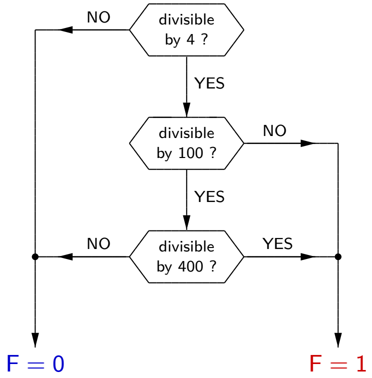
Teil IV

VHDL Design Example

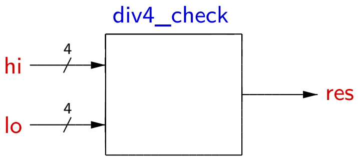
Problem

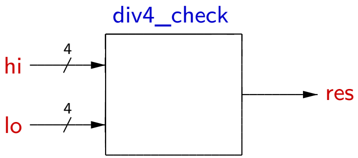
Construct a circuit that generates a leap year flag. Use binary coded decimal (BCD) notation for the year, four decimal digits of 4 bits each.

Leap Year Calculation



VHDL Code

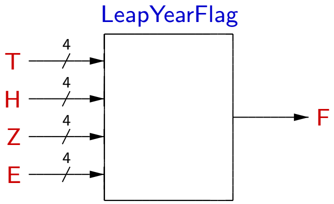


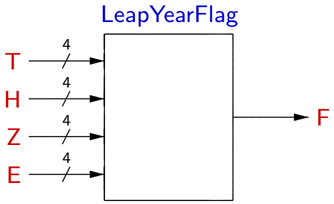


VHDL Code

```
library IEEE;
use IEEE.std_logic_1164.all;

entity div4_check is
  port(hi,lo : in  std_logic_vector(3 downto 0);
       res   : out std_logic);
end div4_check;

architecture div4_check_a of div4_check is
  signal n,v,a,z,s : std_logic;
  signal sel       : std_logic_vector(4 downto 0);
begin
  sel <= hi(0)&lo;
  with sel select
    n <= '1' when "00000",
         '0' when others;
  with sel select
    v <= '1' when "00100",
         '0' when others;
  with sel select
    a <= '1' when "01000",
         '0' when others;
  with sel select
    z <= '1' when "10010",
         '0' when others;
  with sel select
    s <= '1' when "10110",
         '0' when others;
  res <= n or v or a or z or s;
end div4_check_a;
```

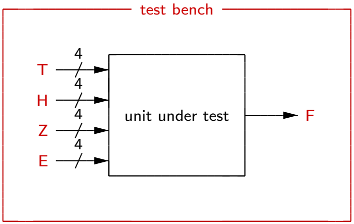
VHDL Code





VHDL Code

```
library IEEE;
use IEEE.std_logic_1164.all;

entity LeapYearFlag is
  port(T,H,Z,E : in std_logic_vector(3 downto 0);
       F       : out std_logic);
end LeapYearFlag;

architecture LeapYearFlag_a of LeapYearFlag is

component div4_check
  port(nm,nl : in  std_logic_vector(3 downto 0);
       res   : out std_logic);
end component;

signal div_by_4lo, div_by_100,
                   div_by_4hi : std_logic;
signal sel                    : std_logic_vector(7 downto 0);

begin
  Test4lo: div4_check port map (Z,E,div_by_4lo);
  sel  <= Z&E;
  with sel select
   div_by_100 <= '1' when "00000000",
                 '0' when others;
  Test4hi: div4_check port map (T,H,div_by_4hi);
  F  <= (div_by_4lo and
        (not div_by_100)) or (div_by_4hi and div_by_100);
end LeapYearFlag_a;
```



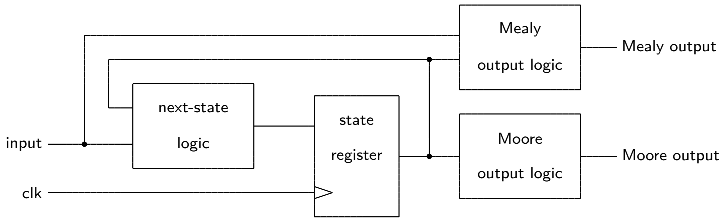
Test Bench

```
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_LeapYearFlag is
end tb_LeapYearFlag;

architecture tb_LeapYearFlag_a of tb_LeapYearFlag is
  component LeapYearFlag is
  port(T, H, Z, E : in std_logic_vector(3 downto 0);
       F          : out std_logic);
  end component;
  signal T, H, Z, E : std_logic_vector(3 downto 0);
  signal F          : std_logic;
begin
  UUT: LeapYearFlag port map (T,H,Z,E,F);
  stimulus: process
    begin
      T <= "0010"; H <= "0000"; Z <= "0000"; E <= "1000";
      wait for 100 ns;
      T <= "0010"; H <= "0000"; Z <= "0000"; E <= "0011";
      wait for 100 ns;
      T <= "0010"; H <= "0000"; Z <= "0000"; E <= "0000";
      wait for 100 ns;
      T <= "0010"; H <= "0001"; Z <= "0000"; E <= "0000";
      wait for 100 ns;
    end process;
end tb_LeapYearFlag_a;
```

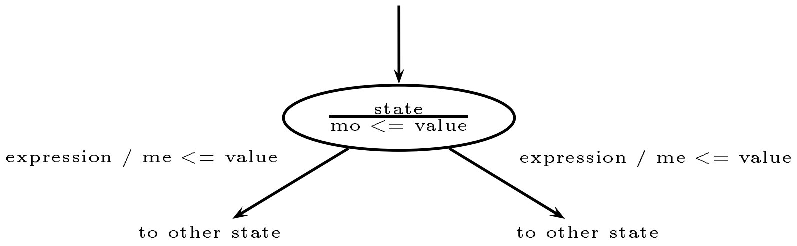
Teil V

Finite State Machines

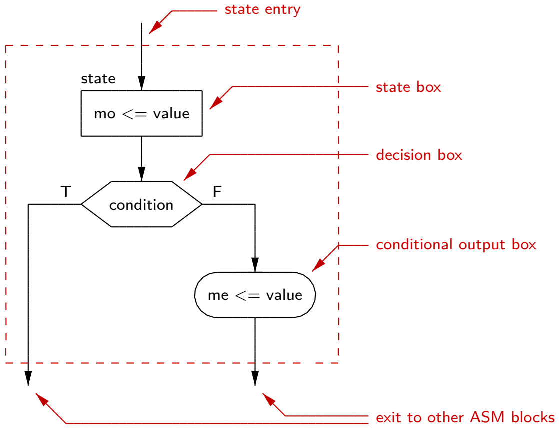
Block Diagram



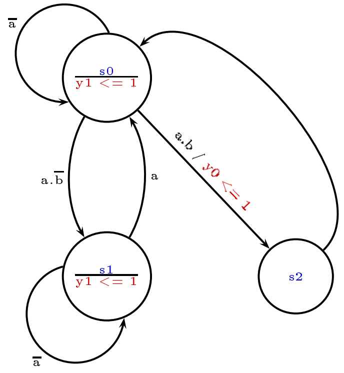
Node


state
mo <= value
expression / me <= value
expression / me <= value
to other state
to other state

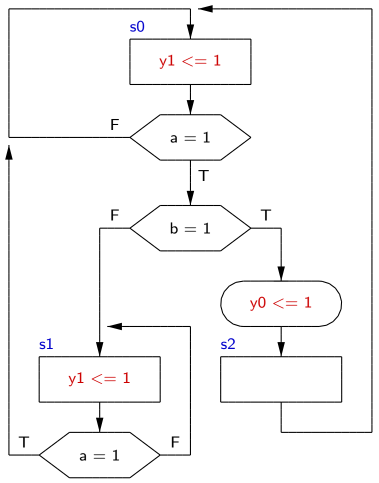
ASM Block


state entry
state
mo <= value
state box
decision box
T
condition
F
conditional output box
me <= value
exit to other ASM blocks

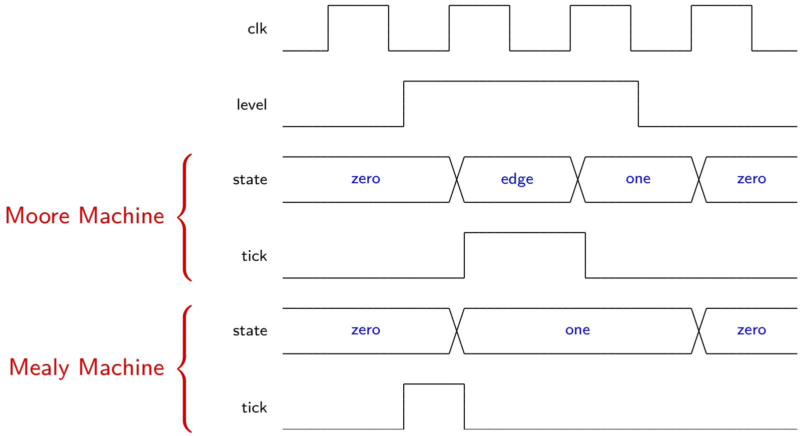
State Diagram



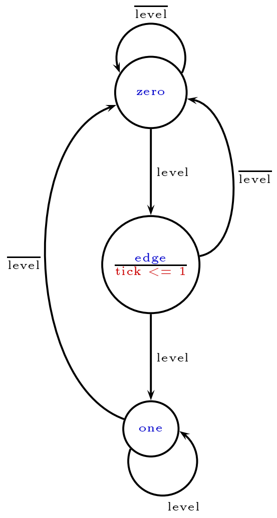
ASM Chart



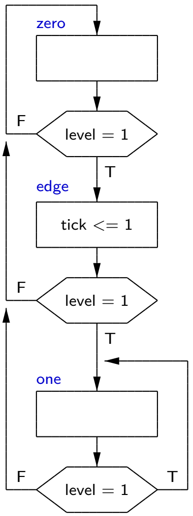
Timing Diagram



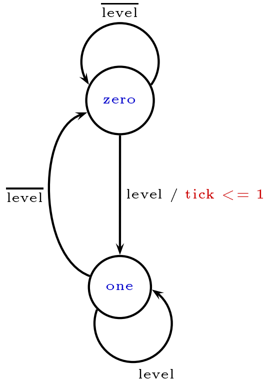
State Diagram


$\overline{\text{level}}$
zero
level
$\overline{\text{level}}$
$\overline{\text{level}}$
edge
tick <= 1
level
one
level

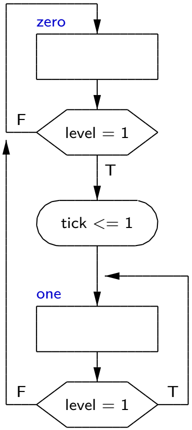
ASM Chart



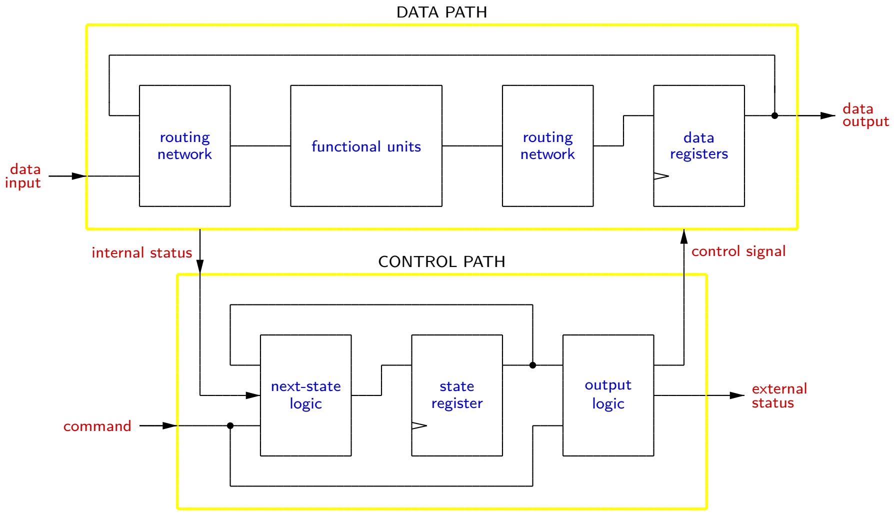
State Diagram



ASM Chart



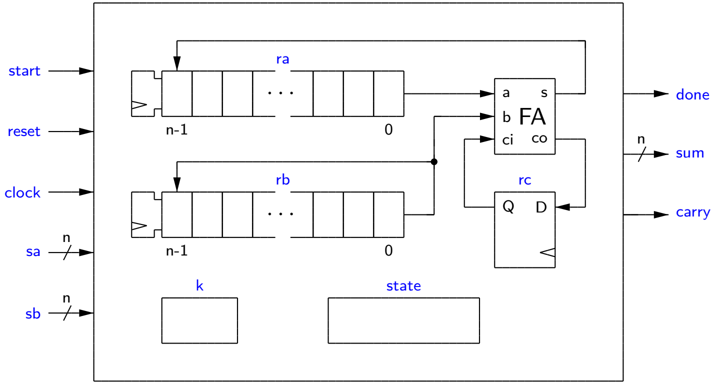
Basic Block Diagram



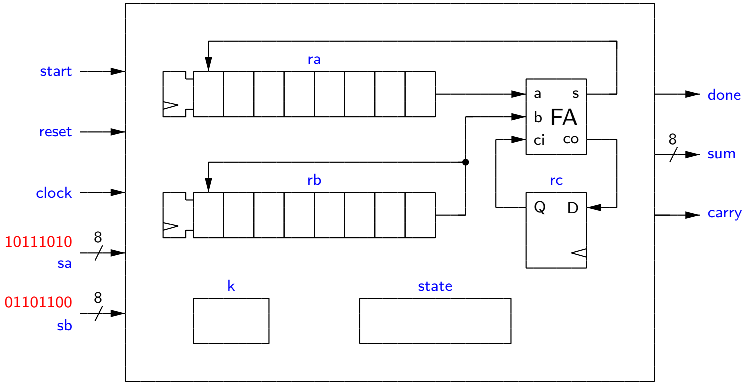
Teil VI

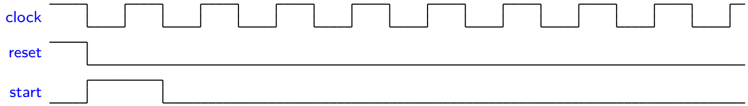
Top-Down Design of a Serial Adder

Block Diagram

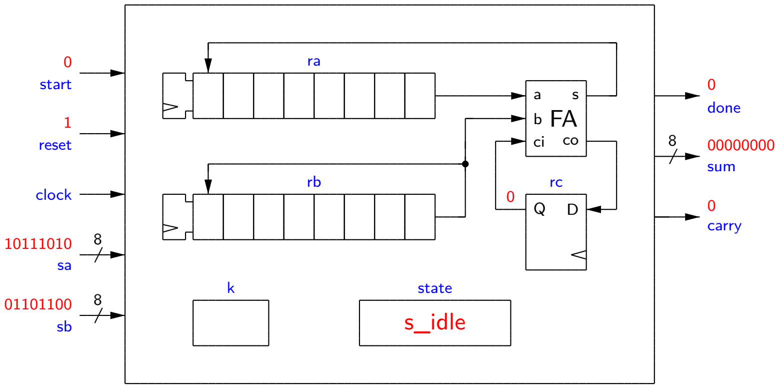


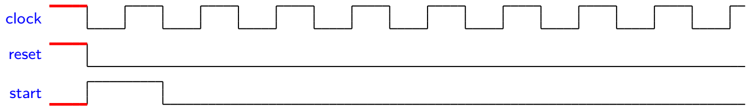
Example: 10111010 + 01101100



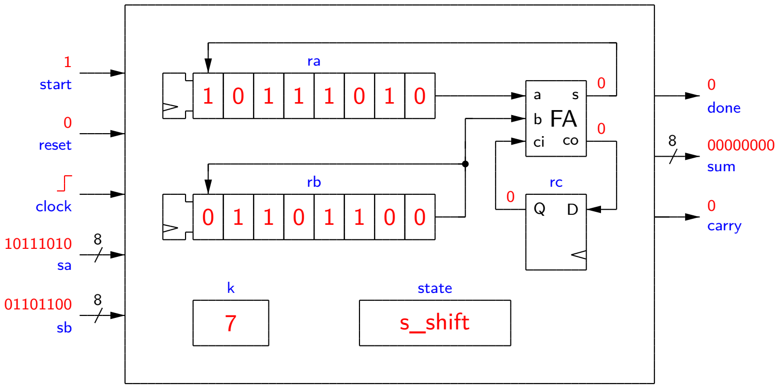


Reset

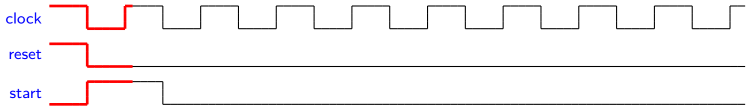

0
start
1
reset
clock
10111010
8
sa
01101100
8
sb
ra
rb
a
s
b
FA
ci
co
0
rc
Q
D
k
state
s_idle
0
done
8
00000000
sum
0
carry


clock
reset
start

Start

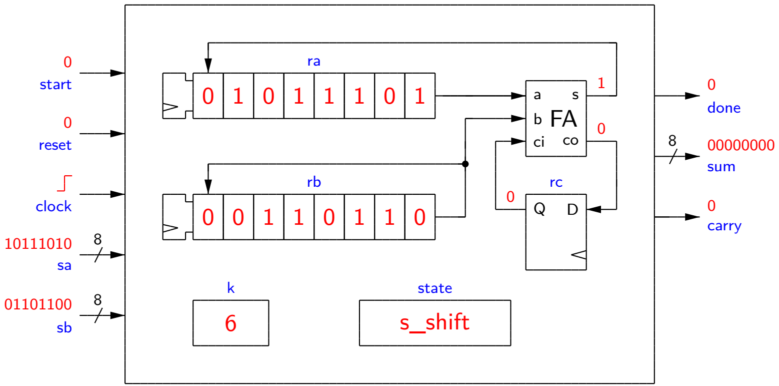


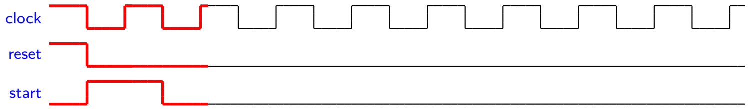
1
start
0
reset
clock
10111010
sa
8
01101100
sb
8
ra
1 0 1 1 1 0 1 0
rb
0 1 1 0 1 1 0 0
a
s
b
FA
ci
co
0
0
rc
0
Q
D
k
7
state
s_shift
0
done
8
00000000
sum
0
carry



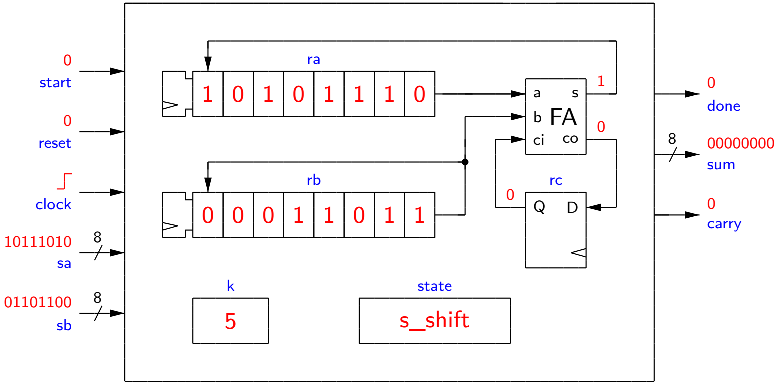
clock
reset
start

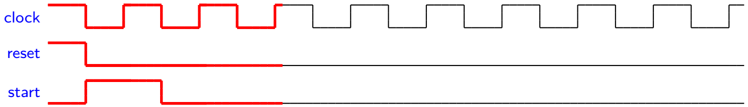
Add and Shift



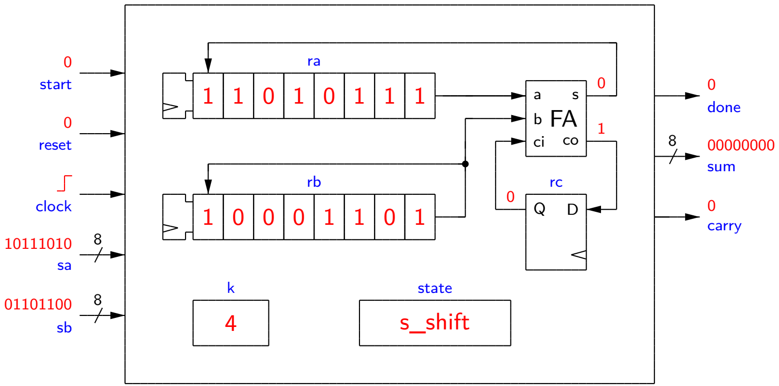


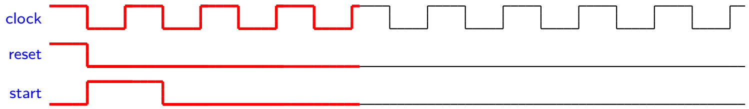
Add and Shift



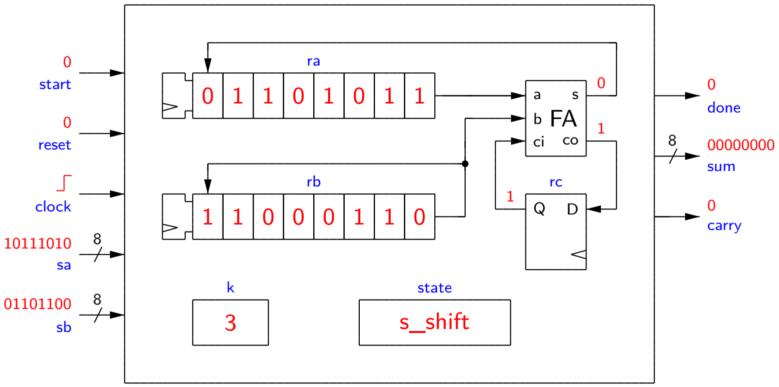


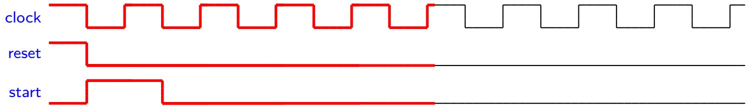
Add and Shift



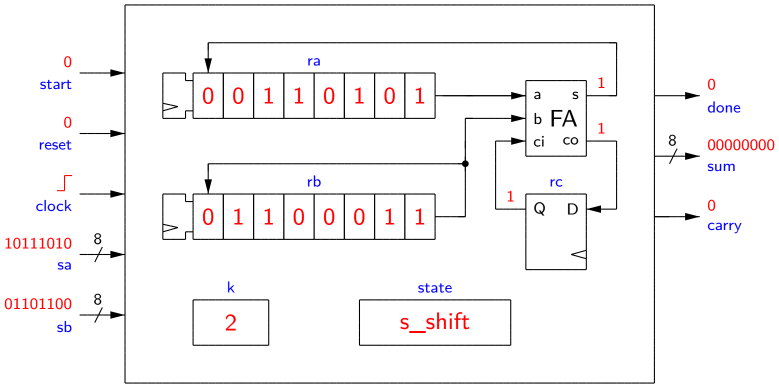


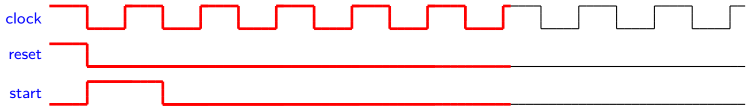
Add and Shift



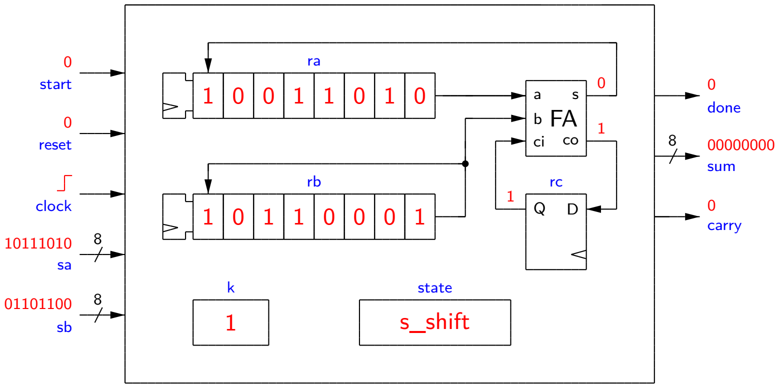


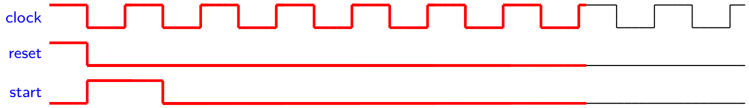
Add and Shift



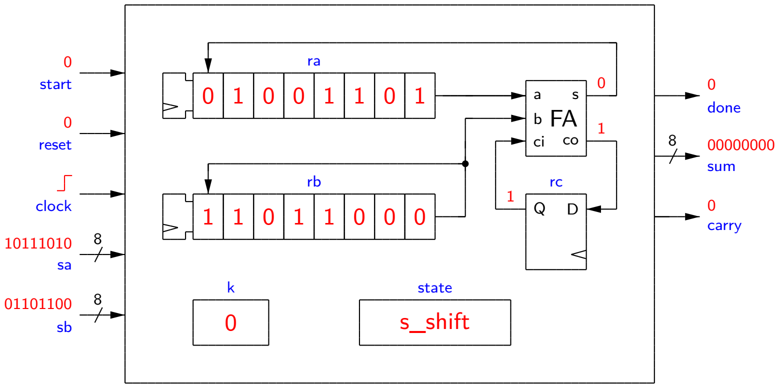


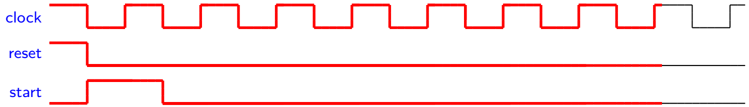
Add and Shift



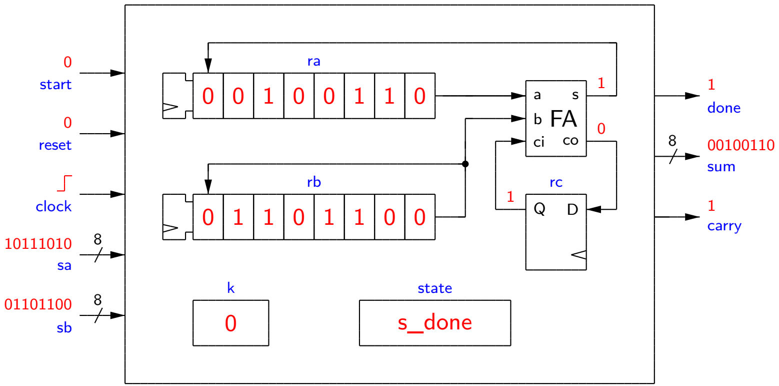


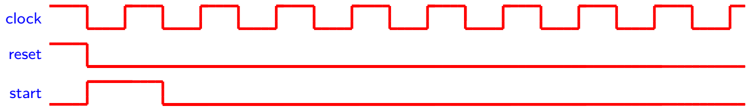
Add and Shift



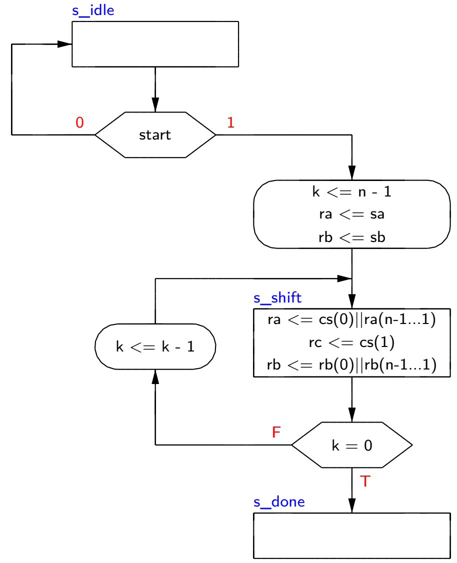


Add and Shift





ASM Chart



VHDL-Code

```
control_unit: process(clk,reset)
begin
  if reset='1' then state <= s_idle;
                   rc <= '0';
  elsif clk'event and clk='1' then
    case state is
      when s_idle =>
        if start='1' then state <= s_shift;
                          k <= n - 1;
                          ra <= sa;
                          rb <= sb;
        end if;
      when s_shift =>
        ra <= cs(0)&ra(n-1 downto 1);
        rc <= cs(1);
        rb <= rb(0)&rb(n-1 downto 1);
        if k=0 then state <= s_done;
               else k <= k - 1;
        end if;
      when s_done =>
    end case;
  end if;
end process;
```