Implementing an example algorithm

- Algorithm as a program
  - operations – data-part
  - sequence and conditions – control-part
- GCD (Greatest Common Divisor)

```
while ( x != y ) {
  if ( x < y ) y = y - x;
  else         x = x - y;
}
```

  - operations – comparisons and subtraction
  - implementing an operation?
    - A < B == A - B < 0 / A != B == A - B != 0 – only subtracter is needed?!
  - the same hardware for all operations or calculating in parallel?
    - speed or size? [and is it a problem at all?]

GCD (Greatest Common Divisor)

- Specification ~~ behavioral description
  - input/output timing fixed – control signals & clock

```
process -- gcd-bhv.vhdl
  variable x, y: unsigned(15 downto 0);
begin
  -- Wait for the new input data
  wait on clk until clk='1' and rst='0';
  x := xi;    y := yi;    rdy <= '0';
  wait on clk until clk='1';
  -- Calculate
  while  x /= y  loop
    if  x < y  then y := y - x;
    else            x := x - y;    end if;
  end loop;
  -- Ready
  xo <= x;   rdy <= '1';
  wait on clk until clk='1';
end process;
```

Problems
- inner loop not clocked
- complex wait statement
(- multiple wait statements)

What to look for?
- different synthesis tools
- minimizing resources
- maximizing performance

Target technologies – ASIC, FPGA

VHDL code & testbenches
`http://mini.pld.ttu.ee/~lrv/gcd/`

GCD – synthesizable code?

- Clocked behavioral style

```
process -- gcd-bhvc.vhdl
  variable x, y: unsigned(15 downto 0);
begin
  -- Wait for the new input data
  while  rst = '1'  loop
    wait on clk until clk='1';
  end loop;
  x := xi;    y := yi;    rdy <= '0';
  wait on clk until clk='1';
  -- Calculate
  while  x /= y  loop
    if  x < y  then y := y - x;
    else            x := x - y;    end if;
    wait on clk until clk='1';
  end loop;
  -- Ready
  xo <= x;   rdy <= '1';
  wait on clk until clk='1';
end process;
```

ASIC: synthesizable
961 e.g. / 20.0 ns
2 sub-s, 2 comp-s

FPGA: non-synthesizable
wait statements in loops :(
explicit FSM needed :(:(

Possible trade-offs
- functional unit sharing
- universal functional units
- out-of-order execution

GCD – behavioral FSM

```
process begin -- gcd-bfsm.vhdl
  wait on clk until clk='1';
  case  state  is
  -- Wait for the new input data
  when S_wait =>
   if  rst='0'  then
     x<=xi; y<=yi; rdy<='0'; state<=S_start;
   end if;
  -- Calculate
  when S_start =>
    if  x /= y  then
      if  x < y  then  y <= y - x;
      else             x <= x - y;    end if;
      state<=S_start;
    else
      xo<=x; rdy<='1'; state<=S_ready;
    end if;
  -- Ready
  when S_ready =>    state<=S_wait;
  end case;
end process;
```

ASIC: synthesizable
911 e.g. / 19.4 ns
2 sub-s, 2 comp-s

FPGA: synthesizable
108 SLC / 9.9 ns
2 sub-s, 2 comp-s

Can it be made better?

Again the possible trade-offs
- functional unit sharing
  one operation per clock step
- universal functional units
  A<B == A-B<0 / A/=B == A-B/=0
- out-of-order execution
  subtracting first then deciding

GCD example – universal functional units?

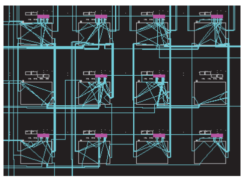
- A<B == A-B<0 / A/=B == A-B/=0

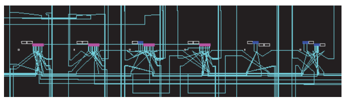
```
-- Three operations:
--   subtraction, and
--   comparisons not-equal &
--   less-than
xo <= xi - yi;

ne <= '1' when xi /= yi else '0';

lt <= '1' when xi < yi else '0';
```

ASIC: 209 e.g. / 19.9 ns
FPGA: 20 SLC / 12.1 ns
***three** adder chains!*

```
-- ALU - subtracting and then comparing
x_out <= xi - yi;    xo <= x_out;
process (x_out)
  variable or_tmp: unsigned(15 downto 0);
begin
  or_tmp(15) := x_out(15);
  for  i in 14 downto 0  loop
    or_tmp(i) := or_tmp(i+1) or x_out(i);
  end loop;
  ne <= to_bit(or_tmp(0));
end process;
lt <= to_bit(x_out(15));
```

ASIC: 148 e.g. / 21.8 ns / FPGA: 12 SLC / 14.6 ns





GCD example – design space exploration

- Different solutions – `http://mini.pld.ttu.ee/~lrv/gcd/gcd.html`

- gcd-bhv.vhdl – pure behavioral description, non-synthesizable
- gcd-bhvc.vhdl – fully clocked behavioral style, some synthesis tools can handle
- gcd-bfsm.vhdl – so called behavioral FSM (explicit FSM & behavioral data-path), synthesizable (but how efficient it is?)

- gcd-rtl1.vhdl – single ALU, 3 clock cycles per iteration – 1) “not equal?”, 2) “less than?”, 3) subtract

```
#1#    a=x-y;   a!=0 ? #2# : #ready#
#2#    a=x-y;   a<0 ? #3# : #4#
#3#    y=y-x    /    #4#    x=x-y
```

- gcd-rtl2.vhdl – single ALU, 2 clock cycles per iteration – 1) “not equal?” and “less than?”, 2) subtract

```
#1#    a=x-y;   a!=0 ? ( a<0 ? #2# : #3# ) : #ready#
#2#    y=y-x    /    #3#    x=x-y
```

GCD example – design space exploration (2)

- **Different solutions – `http://mini.pld.ttu.ee/~lrv/gcd/gcd.html`**

  - **gcd-rtl3.vhdl – comparator (less than) controls subtraction, 1 clock cycle per iteration – small but slow (sequential) data-path**

    ```
    #1#    x<y ? y=y-x : ( a=x-y;   a!=0 ? x=x-y : #ready# )
    ```

  - **gcd-rtl4.vhdl – out-of-order execution – both subtractions are calculated first then the decision is made (one subtracter compares for “less than”, another for “not equal”)**

    ```
    #1#    a1=x-y;   a2=y-x;   a2!=0 ? ( a1<0 ? y=a2 : x=a1 ) : #ready#
    ```

  - **gcd-rtl5.vhdl – out-of-order execution – both subtractions are calculated first then the decision is made (one subtracter compares for “less than” but separate “not equal”)**

    ```
    #1#    a1=x-y;   a2=y-x;   x!=y ? ( a1<0 ? y=a2 : x=a1 ) : #ready#
    ```

GCD example – single ALU, 2 clock cycles per iteration

gcd-rtl2.vhdl

```
-- Next state function of the FSM
process (state, rst, alu_ne, alu_lt) begin
  ena_x <= '0';     ena_y <= '0';      ena_r <= '0';
  set_rdy <= '0';   xi_yi_sel <= '0';  sub_y_x <= '0';
  next_state <= state;
  case state  is
  when S_wait =>     -- Wait for the new input data
    if  rst='0'  then
      xi_yi_sel <= '1';    ena_x <= '1';    ena_y <= '1';
      next_state <= S_start;
    end if;
  when S_start =>     -- Loop: ready?
    if  alu_ne='1'  then
      if  alu_lt='1'  then    next_state <= S_sub_y_x;
      else    next_state <= S_sub_x_y;    end if;
    else    next_state <= S_ready;
    end if;
  when S_sub_y_x =>     -- Loop: y-x
    ena_y <= '1';    sub_y_x <= '1';
    next_state <= S_start;
  when S_sub_x_y =>     -- Loop: x-y
    ena_x <= '1';    sub_y_x <= '0';
    next_state <= S_start;
  when S_ready =>     -- Ready
    ena_r <= '1';    set_rdy <= '1';
    next_state <= S_wait;
  end case;
end process;
```

```
-- ALU: subtract / less-than / not-equal
alu_o <= alu_1 - alu_2;
alu_lt <= to_bit(alu_o(15));
process (alu_o)
  variable or_tmp: unsigned(15 downto 0);
begin
  or_tmp(15)  := alu_o(15);
  for  i in 14 downto 0  loop
    or_tmp(i)  := or_tmp(i+1) or alu_o(i);
  end loop;
  alu_ne <= to_bit(or_tmp(0));
end process;

-- Multiplexers
x_i <= xi when xi_yi_sel='1' else alu_o;
y_i <= yi when xi_yi_sel='1' else alu_o;
alu_1 <= y when sub_y_x='1' else x;
alu_2 <= x when sub_y_x='1' else y;

-- Registers
process begin
  wait on clk until clk='1';
  state <= next_state;
  if  ena_x='1'  then    x <= x_i;    end if;
  if  ena_y='1'  then    y <= y_i;    end if;
  if  ena_r='1'  then    xo <= x;     end if;
  rdy <= set_rdy;
end process;
```

GCD – comparator+subtracter, 1 clock cycle

gcd-rtl3.vhdl

```
-- Next state function of the FSM
process (state, rst, alu_ne) begin
  ena_xy <= '0';    ena_r <= '0';
  set_rdy <= '0';    xi_yi_sel <= '0';
  next_state <= state;
  case state is
  when S_wait =>    -- Wait for the new input data
    if  rst='0'  then
      xi_yi_sel <= '1';    ena_xy <= '1';
      next_state <= S_start;
    end if;
  when S_start =>    -- Calculate
    if  alu_ne='1'  then
      ena_xy <= '1';    next_state <= S_start;
    else
      ena_r <= '1';    set_rdy <= '1';
      next_state <= S_ready;
    end if;
  when S_ready =>    -- Ready
    ena_r <= '1';    set_rdy <= '1';
    next_state <= S_wait;
  end case;
end process;

-- Comparator (less-than)
sub_y_x <= '1' when x < y else '0';
```

```
-- Subtracter (+not-equal)
alu_o <= alu_1 - alu_2;
process (alu_o)
  variable or_tmp: unsigned(15 downto 0);
begin
  or_tmp(15) := alu_o(15);
  for  i in 14 downto 0  loop
    or_tmp(i) := or_tmp(i+1) or alu_o(i);
  end loop;
  alu_ne <= to_bit(or_tmp(0));
end process;
-- Multiplexers
x_i <= xi when xi_yi_sel='1' else alu_o;
y_i <= yi when xi_yi_sel='1' else alu_o;
alu_1 <= y when sub_y_x='1' else x;
alu_2 <= x when sub_y_x='1' else y;
ena_x <= '1' when (sub_y_x='0' and ena_xy='1') or
                  xi_yi_sel='1' else '0';
ena_y <= '1' when (sub_y_x='1' and ena_xy='1') or
                  xi_yi_sel='1' else '0';
-- Registers
process begin
  wait on clk until clk='1';
  state <= next_state;
  if  ena_x='1'  then    x <= x_i;    end if;
  if  ena_y='1'  then    y <= y_i;    end if;
  if  ena_r='1'  then    xo <= x;    end if;
  rdy <= set_rdy;
end process;
```

GCD example – out-of-order execution (2 sub-s)

gcd-rtl5.vhdl

```
-- Next state function of the FSM
process (state, rst, alu_ne) begin
  ena_xy <= '0';    ena_r <= '0';
  set_rdy <= '0';    xi_yi_sel <= '0';
  next_state <= state;
  case state is
  -- Wait for the new input data
  when S_wait =>
    if  rst='0'  then
      xi_yi_sel <= '1';    ena_xy <= '1';
      next_state <= S_start;
    end if;
  -- Calculate
  when S_start =>
    if  alu_ne='1'  then
      ena_xy <= '1';
      next_state <= S_start;
    else
      ena_r <= '1';    set_rdy <= '1';
      next_state <= S_ready;
    end if;
  -- Ready
  when S_ready =>
    ena_r <= '1';    set_rdy <= '1';
    next_state <= S_wait;
  end case;
end process;
```

```
-- Subtracter (x-y) / comparator (x<y)
alu_o1 <= x - y;
sub_y_x <= '1' when alu_o1(alu_o1'high)='1' else '0';

-- Subtracter (y-x)
alu_o2 <= y - x;

-- Comparator (y/=x)
alu_ne <= '1' when x /= y else '0';

-- Multiplexers
x_i <= xi when xi_yi_sel='1' else alu_o1;
y_i <= yi when xi_yi_sel='1' else alu_o2;
ena_x <= '1' when (sub_y_x='0' and ena_xy='1') or
         xi_yi_sel='1' else '0';
ena_y <= '1' when (sub_y_x='1' and ena_xy='1') or
         xi_yi_sel='1' else '0';

-- Registers
process begin
  wait on clk until clk='1';
  state <= next_state;
  if  ena_x='1'  then    x <= x_i;    end if;
  if  ena_y='1'  then    y <= y_i;    end if;
  if  ena_r='1'  then    xo <= x;    end if;
  rdy <= set_rdy;
end process;
```

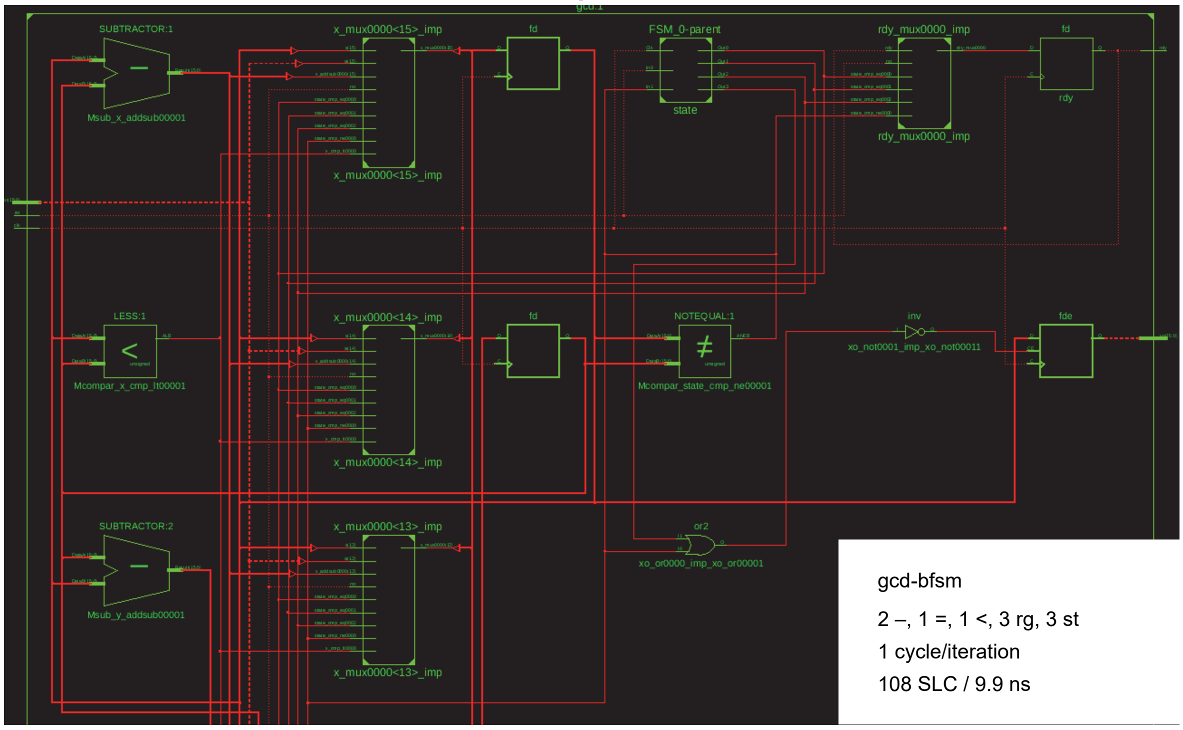
GCD example – synthesis results

Technology	FPGA				ASIC			
Constraint[1]	50 MHz		100 MHz		50 MHz		25 MHz	
	[SLC]	[ns]	[SLC]	[ns]	[e.g.]	[ns]	[e.g.]	[ns]
gcd-bhv[2]	*93*	*17.3*	-	-	*1141*	*20.0*	-	-
gcd-bhvc	-	-	-	-	961	20.0	977	31.1
gcd-bfsm	108	9.9	108	9.4	911	19.4	984	30.8
gcd-rtl1	50	10.8	50	9.7	986	19.8	883	32.4
gcd-rtl2	48	10.8	48	10.0	931	19.9	882	32.3
gcd-rtl3	58	17.0	58	14.6	1134	20.0	928	40.0
gcd-rtl4	78	12.6	78	9.0	976	19.9	928	29.0
gcd-rtl5	58	8.0	58	7.6	915	20.0	932	26.9

1) Clock period was the only constraint

2) gcd-bhv was synthesized using the help of prototype HLS tool xTractor

GCD example – why such differences?



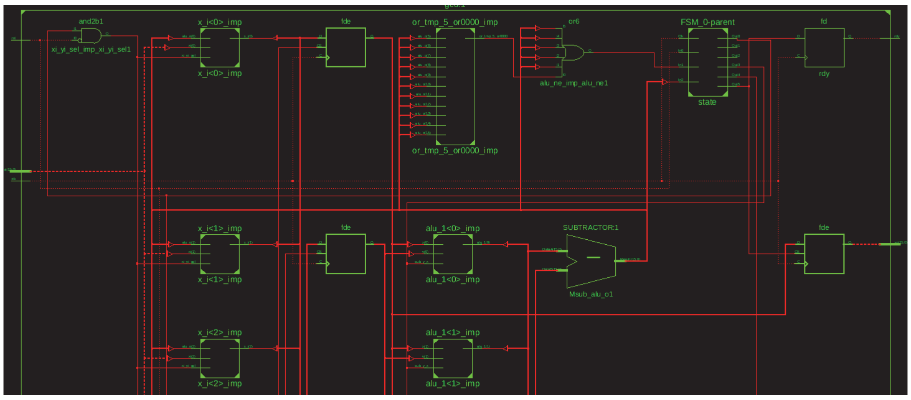
gcd-bfsm

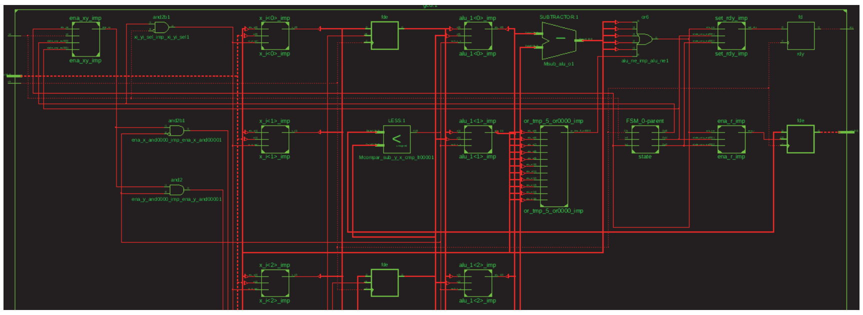
2 –, 1 =, 1 <, 3 rg, 3 st

1 cycle/iteration

108 SLC / 9.9 ns

GCD example – why such differences?





gcd-rtl1

1 ALU, 3 rg, 6 st

3 cycles/iteration

50 SLC / 10.8 ns

gcd-rtl2

1 ALU, 3 rg, 5 st

2 cycles/iteration

48 SLC / 10.8 ns

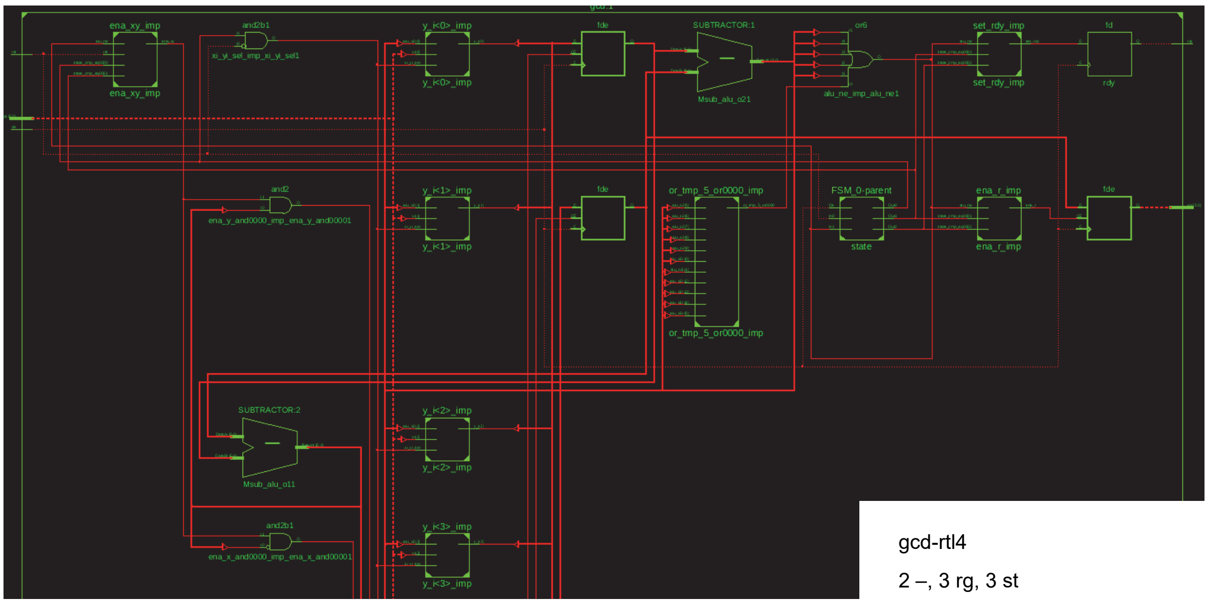
gcd-rtl3

1 –, 1 <, 3rg, 3st

1 cycle/iteration

58 SLC / 17.0 ns

GCD example – why such differences?



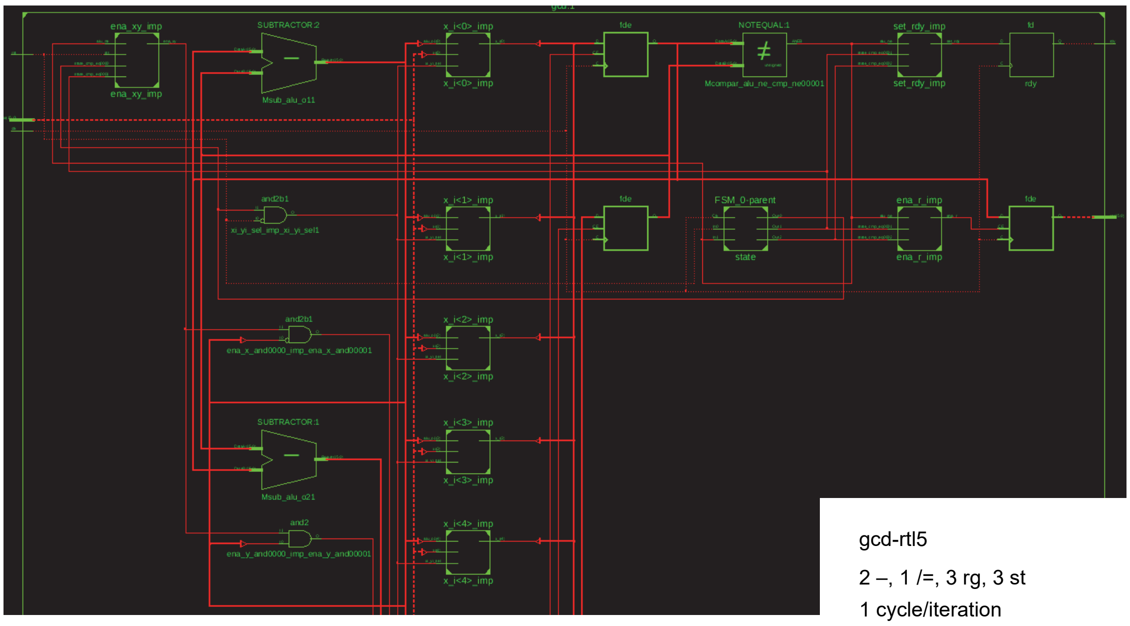
gcd-rtl4

2 –, 3 rg, 3 st

1 cycle/iteration

78 SLC / 12.6 ns

GCD example – why such differences?



gcd-rtl5

2 –, 1 /=, 3 rg, 3 st

1 cycle/iteration

58 SLC / 8.0 ns