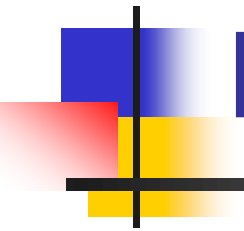


EL 6463 – Lecture notes #3

Simple Function (incl. first 2 lines)

- Round keys are fixed and given
  - S[0], S[1], S[2],…, S[25]
- 64-bit data input, 64-bit output

```
A = A + S[0];
B = B + S[1];
for i=1 to 12 do
        A = ((A XOR B) <<< B) + S[2*i];
        B = ((B XOR A) <<< A) + S[2*1+1];
```

Entity Definition

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY simple IS
    PORT
    (
            clr         : IN      STD_LOGIC;
            clk         : IN      STD_LOGIC;
            din         : IN      STD_LOGIC_VECTOR(63 DOWNTO 0);
            di_vld      : IN      STD_LOGIC;  -- handshake signal input is valid
            dout        : OUT     STD_LOGIC_VECTOR(63 DOWNTO 0);
            do_rdy      : OUT     STD_LOGIC   -- handshake signal output is ready
     );
END simple;
```

Internal Signals and Round Keys

```
ARCHITECTURE rtl OF simple IS
    SIGNAL i_cnt    : STD_LOGIC_VECTOR(3 DOWNTO 0); -- round counter

    SIGNAL ab_xor  : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL a_rot    : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL a        : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL a_pre    : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL a_reg    : STD_LOGIC_VECTOR(31 DOWNTO 0); -- register A

    SIGNAL ba_xor: STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL b_rot   : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL b        : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL b_pre   : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL b_reg   : STD_LOGIC_VECTOR(31 DOWNTO 0); -- register B
```

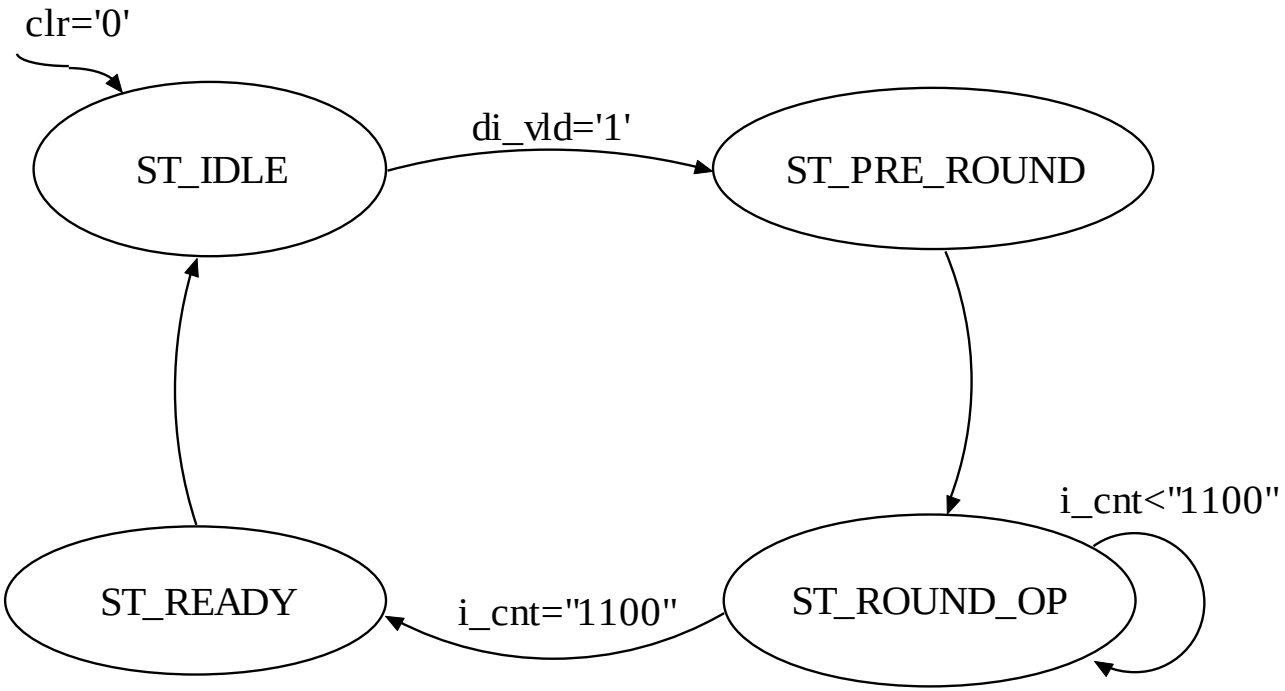
Simple function: Roundkey ROM

```
-- define a type for round keys
TYPE rom IS ARRAY (0 TO 25) OF STD_LOGIC_VECTOR(31 DOWNTO 0);
--instantiate round key rom with 26 round keys
CONSTANT skey : rom:=rom'(X"9BBBD8C8", X"1A37F7FB", X"46F8E8C5",
     X“460C6085”, X“70F83B8A”, X“284B8303”, X“513E1454”, X“F621ED22”,
      X"3125065D", X"11A83A5D", X"D427686B", X"713AD82D", X"4B792F99",
      X"2799A4DD", X"A7901C49", X"DEDE871A", X"36C03196", X"A7EFC249",
      X"61A78BB8", X"3B0A1D2B", X"4DBFCA76", X"AE162167", X"30D76B0A",
      X"43192304", X"F6CC1431", X"65046380");
```

Simple function state machine: states

```
-- Simple state machine
 TYPE  StateType IS (ST_IDLE, --
                     ST_PRE_ROUND, -- in this state simple pre-round op is performed
                     ST_ROUND_OP, -- in this state simple round op is performed. The state machine remains in this state for twelve clock cycles.
                     ST_READY --
                     );
 -- Simple state machine has four states: idle, pre_round, round and ready
 SIGNAL  state:   StateType;
```

Simple function state machine



Round Operation: Comb. Logic

```
BEGIN
 -- A=((A XOR B)<<<B) + S[2*i];
 ab_xor <= a_reg XOR b_reg;
 WITH b_reg(4 DOWNTO 0) SELECT
  a_rot<=ab_xor(30 DOWNTO 0) & ab_xor(31) WHEN "00001",
            ab_xor(29 DOWNTO 0) & ab_xor(31 DOWNTO 30) WHEN "00010",
             ab_xor(28 DOWNTO 0) & ab_xor(31 DOWNTO 29) WHEN "00011",
             ab_xor(27 DOWNTO 0) & ab_xor(31 DOWNTO 28) WHEN "00100",
             ab_xor(26 DOWNTO 0) & ab_xor(31 DOWNTO 27) WHEN "00101",
             ab_xor(25 DOWNTO 0) & ab_xor(31 DOWNTO 26) WHEN "00110",
             ab_xor(24 DOWNTO 0) & ab_xor(31 DOWNTO 25) WHEN "00111",
             ab_xor(23 DOWNTO 0) & ab_xor(31 DOWNTO 24) WHEN "01000",
             ab_xor(22 DOWNTO 0) & ab_xor(31 DOWNTO 23) WHEN "01001",
             ab_xor(21 DOWNTO 0) & ab_xor(31 DOWNTO 22) WHEN "01010",
             ab_xor(20 DOWNTO 0) & ab_xor(31 DOWNTO 21) WHEN "01011",
             ab_xor(19 DOWNTO 0) & ab_xor(31 DOWNTO 20) WHEN "01100",
             ab_xor(18 DOWNTO 0) & ab_xor(31 DOWNTO 19) WHEN "01101",
             ab_xor(17 DOWNTO 0) & ab_xor(31 DOWNTO 18) WHEN "01110",
```

Round Operation: Comb. Logic

```
            ab_xor(16 DOWNTO 0) & ab_xor(31 DOWNTO 17) WHEN "01111",
            ab_xor(15 DOWNTO 0) & ab_xor(31 DOWNTO 16) WHEN "10000",
            ab_xor(14 DOWNTO 0) & ab_xor(31 DOWNTO 15) WHEN "10001",
            ab_xor(13 DOWNTO 0) & ab_xor(31 DOWNTO 14) WHEN "10010",
            ab_xor(12 DOWNTO 0) & ab_xor(31 DOWNTO 13) WHEN "10011",
            ab_xor(11 DOWNTO 0) & ab_xor(31 DOWNTO 12) WHEN "10100",
            ab_xor(10 DOWNTO 0) & ab_xor(31 DOWNTO 11) WHEN "10101",
            ab_xor(9 DOWNTO 0) & ab_xor(31 DOWNTO 10) WHEN "10110",
            ab_xor(8 DOWNTO 0) & ab_xor(31 DOWNTO 9) WHEN "10111",
            ab_xor(7 DOWNTO 0) & ab_xor(31 DOWNTO 8) WHEN "11000",
            ab_xor(6 DOWNTO 0) & ab_xor(31 DOWNTO 7) WHEN "11001",
            ab_xor(5 DOWNTO 0) & ab_xor(31 DOWNTO 6) WHEN "11010",
            ab_xor(4 DOWNTO 0) & ab_xor(31 DOWNTO 5) WHEN "11011",
            ab_xor(3 DOWNTO 0) & ab_xor(31 DOWNTO 4) WHEN "11100",
            ab_xor(2 DOWNTO 0) & ab_xor(31 DOWNTO 3) WHEN "11101",
            ab_xor(1 DOWNTO 0) & ab_xor(31 DOWNTO 2) WHEN "11110",
            ab_xor(0) & ab_xor(31 DOWNTO 1) WHEN "11111",
            ab_xor WHEN OTHERS;
a_pre<=din(63 DOWNTO 32) + skey(0); -- A = A + S[0]
a<=a_rot + skey(CONV_INTEGER(i_cnt & '0'));  -- S[2*i]
```

Round Operation: Comb. Logic

```
-- B=((B XOR A) <<<A)     + S[2*i+1]
    ba_xor <= b_reg XOR a;
    WITH a(4 DOWNTO 0) SELECT
        b_rot<=ba_xor(30 DOWNTO 0) & ba_xor(31) WHEN "00001",
                  ......
                  ba_xor WHEN OTHERS;
    b_pre <= din(31 DOWNTO 0) + skey(1);  -- B = B + S[1]
    b<=b_rot + skey(CONV_INTEGER(i_cnt & '1'));  -- S[2*i+1]
```

Sequential Circuit: Register A

```
-- A register
PROCESS(clr, clk)  BEGIN
    IF(clr='0') THEN
       a_reg<=(OTHERS=>'0');
    ELSIF(clk'EVENT AND clk='1') THEN
        IF(state=ST_PRE_ROUND) THEN   a_reg<=a_pre;
       ELSIF(state=ST_ROUND_OP) THEN   a_reg<=a;   END IF;
    END IF;
END PROCESS;
```

Sequential Circuit: Register B

```
-- B register
    PROCESS(clr, clk)  BEGIN
        IF(clr='0') THEN
           b_reg<=(OTHERS=>'0');
        ELSIF(clk'EVENT AND clk='1') THEN
           IF(state=ST_PRE_ROUND) THEN   b_reg<=b_pre;
          ELSIF(state=ST_ROUND_OP) THEN   b_reg<=b;   END IF;
        END IF;
    END PROCESS;
```

Simple function: state machine

```
PROCESS(clr, clk)
BEGIN
   IF(clr='0') THEN
      state<=ST_IDLE;
   ELSIF(clk'EVENT AND clk='1') THEN
      CASE state IS
         WHEN ST_IDLE=>  IF(di_vld=‘1’) THEN state<=ST_PRE_ROUND;  END IF;
         WHEN ST_PRE_ROUND=>    state<=ST_ROUND_OP;
         WHEN ST_ROUND_OP=>  IF(i_cnt=“1100”) THEN state<=ST_READY;  END IF;
         WHEN ST_READY=>   state<=ST_IDLE;
      END CASE;
   END IF;
END PROCESS;
```

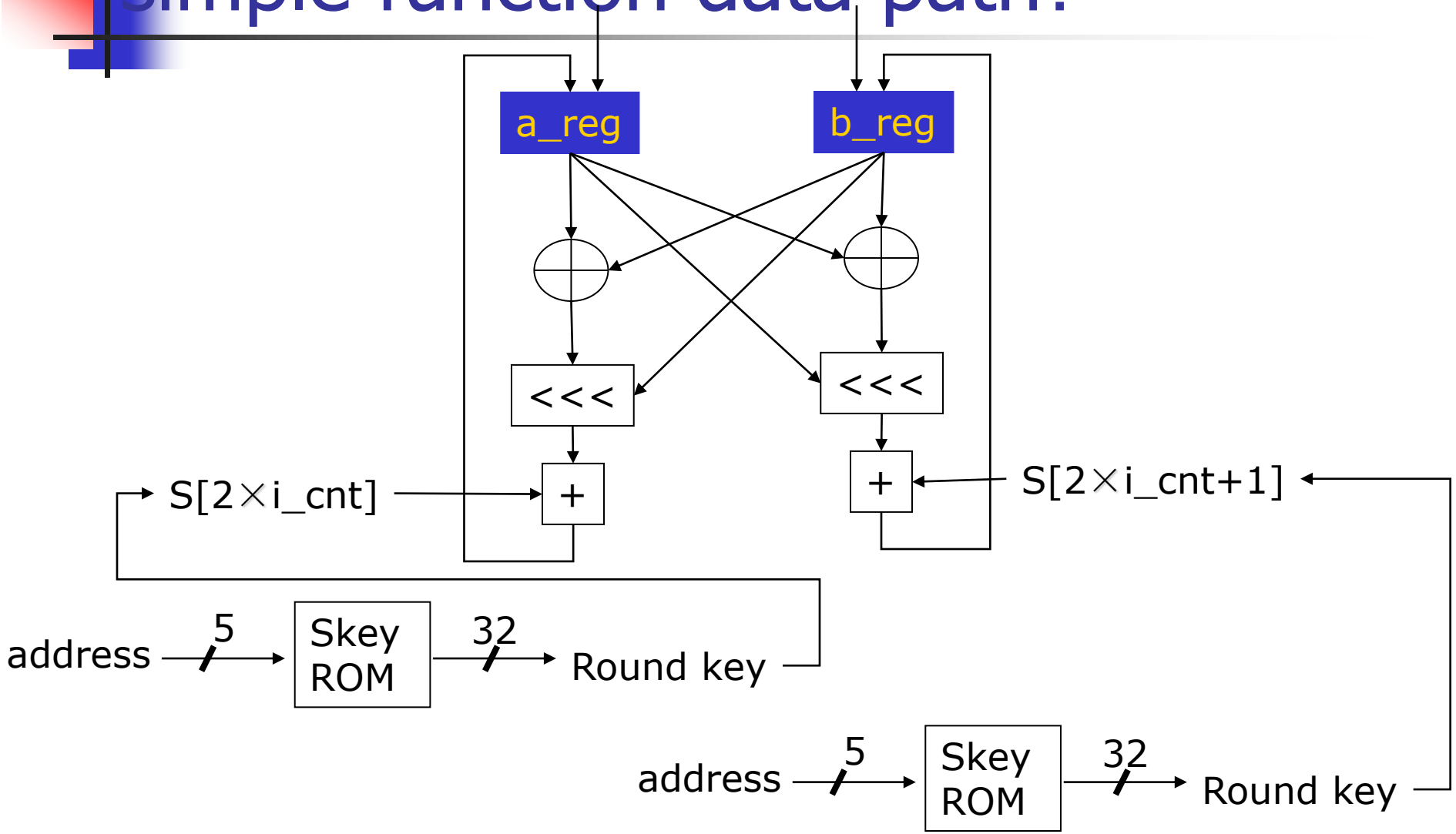
Sequential Circuit: round counter

```
-- round counter
    PROCESS(clr, clk)  BEGIN
        IF(clr='0') THEN
          i_cnt<="0001";
        ELSIF(clk'EVENT AND clk='1') THEN
          IF(state=ST_ROUND_OP) THEN
            IF(i_cnt="1100") THEN   i_cnt<="0001";
            ELSE    i_cnt<=i_cnt+'1';    END IF;
          END IF;
        END IF;
    END PROCESS;
```

Sequential Circuit: round counter

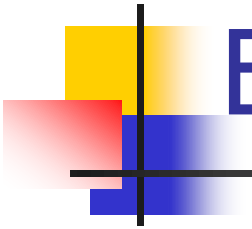
```
dout<=a_reg & b_reg;

    WITH state SELECT
        do_rdy<=    '1' WHEN ST_READY,     ---handshake signal
                    '0' WHEN OTHERS;
END rtl;
```

What if you want to implement this simple function data path?



Exercise #1

- Simulate simple function VHDL Model
  - Functional Simulation
    - Input/Plaintext (din): of 0xFEDCBA9876543210,
    - Output/Ciphertext (dout): 0x18C7C99A97F52AE0
- Synthesize Design (target the FPGA on your board)
  - Do Timing Simulation.
    - Pick the right clock period based on the timing report
  - Change clock period to 10ns and redo the timing simulation
    - what will happen? Explain
- Understand following concepts/modeling
  - Data path+controller
  - State machine: Define states with ENUM type; Define state transitions with a PROCESS



Exercise #2

- Implement inverse function
  - Simulate the Model
    - Show how ciphertext from Exercise 1 is decrypted to get plaintext.
    - Ciphertext (input/din) is 0x18C7C99A97F52AE0.
    - Plaintext (result/dout) is 0xFEDCBA9876543210.
  - Synthesize Design
    - Draw block diagram and explain it.
- Analyze the timing simulation results
  - What is the smallest clock period you can run your design?
  - What happens when you use 10 ns clock period? explain

Results

- Timing – 18ns delay! (from xilinx vivado)
- Area
  - Logic Utilization
    - Number of Slice Flip Flops: 97 out of 24,576 1%
    - Number of 4 input LUTs: 701 out of 24,576 2%
  - Logic Distribution
    - Number of occupied Slices: 381 out of 12,288 1%
    - Number of Slices containing only related logic: 381 out of 381 100%
    - Number of Slices containing unrelated logic: 0 out of 381 0%
  - Total Number 4 input LUTs: 725 out of 24,576 2%
    - Number used as logic: 701
    - Number used as a route-thru: 24
  - Number of bonded IOBs: 131 out of 404 23%
  - Total equivalent gate count for design: 6,089

The inverse function

```
for i=12 downto 1 do
    B = ((B – S[2*i + 1] >>> A) XOR A;
    A = ((A – S[2*i]) >>> B) XOR B;

B = B – S[1];
A = A – S[0];
```

The inverse function

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY inverse IS
    PORT
    (
            clr         : IN STD_LOGIC;  -- Asynchronous reset
            clk         : IN STD_LOGIC;  -- Clock signal
            di_vld      : IN STD_LOGIC;
            din         : IN STD_LOGIC_VECTOR(63 DOWNTO 0);  -- 64-bit input
            dout        : OUT STD_LOGIC_VECTOR(63 DOWNTO 0) -- 64-bit output
            do_rdy      : OUT STD_LOGIC
     );
END inverse;
```

The inverse function

```
ARCHITECTURE rtl OF inverse IS
    -- Declare all necessary signals and constants
    -- Refer to simple function model
BEGIN
    -- B=((B – S[2*i+1]) >>> A) XOR A, exercise 2 of Lab 1

    -- A=((A – S[2*i]) >>> B) XOR B;

    -- Update Register B, exercise 1 of Lab 2

    -- Update Register A;

    -- Round counter (counts from ? to 1, think about initialization)

    -- State machine
END rtl;
```