ECE 545
Lecture 11

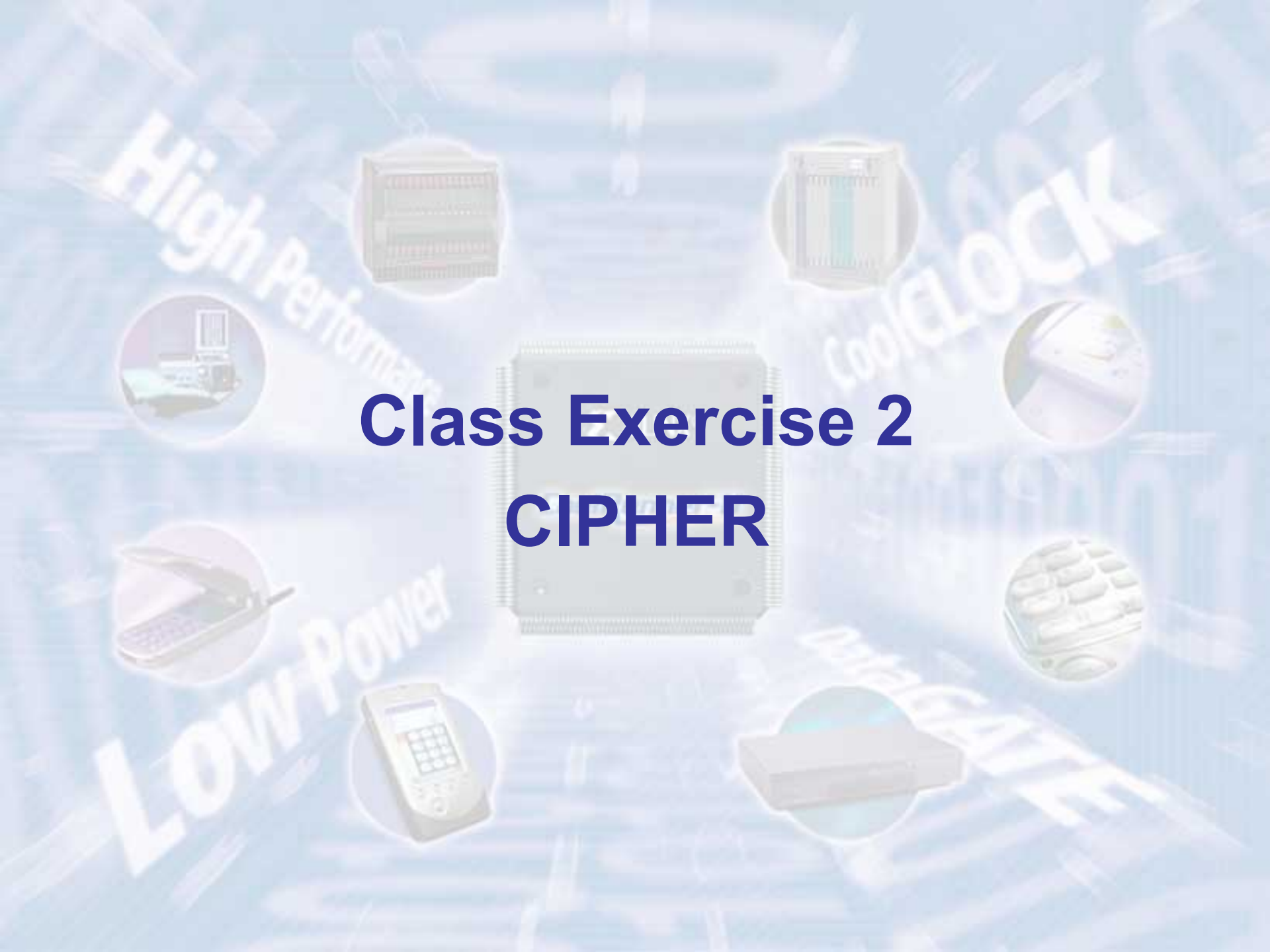
Design of Controllers

Part 2

Required reading

- P. Chu, *RTL Hardware Design using VHDL*

Chapter 10, Finite State Machine: Principle & Practice



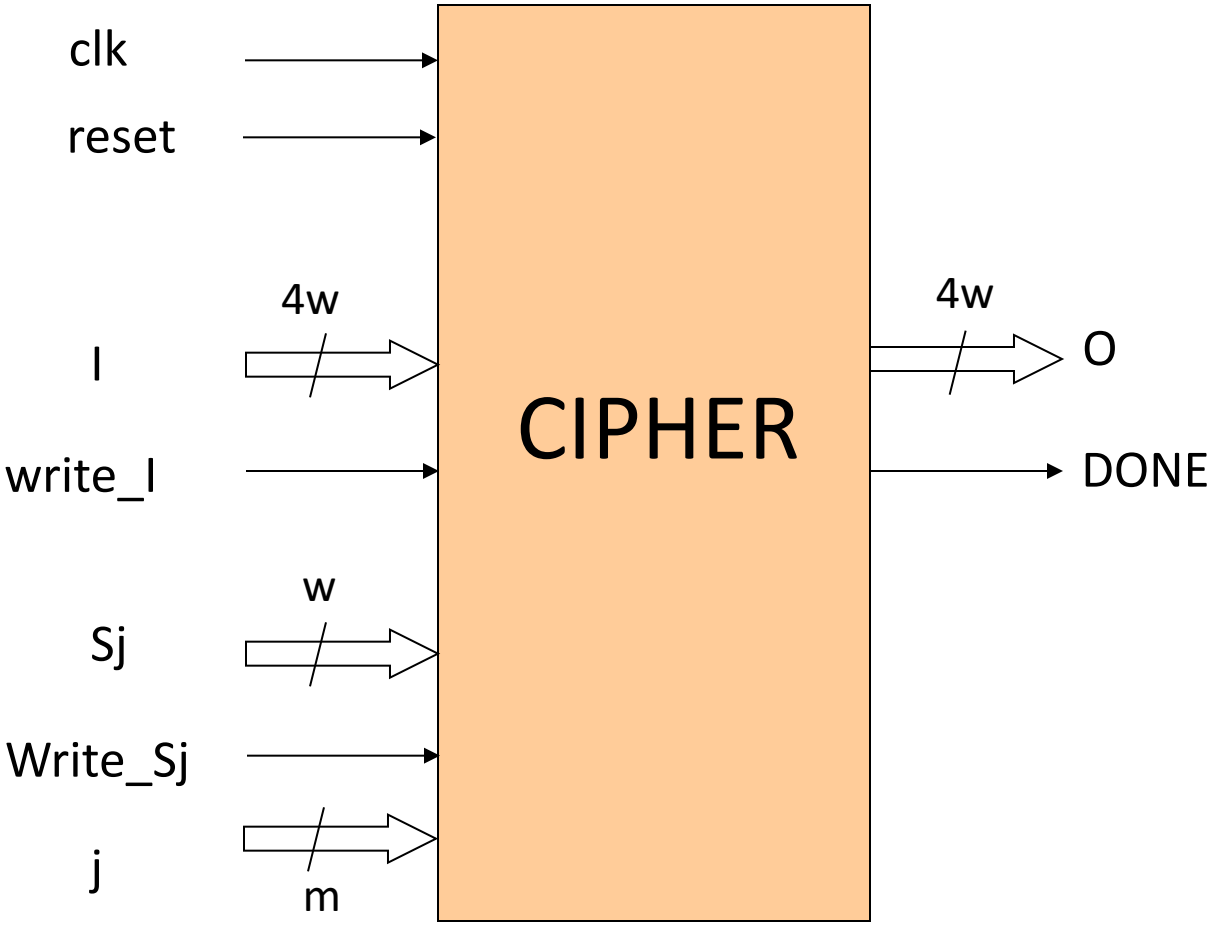
Class Exercise 2
CIPHER

Pseudocode

Split input I into four words, I3, I2, I1, I0, of the size of w bits each

```
A = I3; B = I2; C = I1; D=I0
B = B + S[0]
D = D + S[1]
        for i = 1 to r do
        {
                T = (B*(2B + 1)) <<< k
                U = (D*(2D + 1)) <<< k
                A' = ((A ⊕ T) <<< U) + S[2i]
                C' = ((C ⊕ U) <<< T) + S[2i + 1]
                (A, B, C, D)  =  (B, C', D, A')
        }
A = A + S[2r + 2]
C = C + S[2r + 3]
O = (A, B, C, D)
```

Circuit Interface



Interface Table

Port	Width	Meaning
clk	1	System clock.
reset	1	System reset – clears internal registers.
I	4w	Input block.
write_I	1	Synchronous write control signal for the input block I. After the block I is written to the CIPHER unit, the encryption of I starts automatically.
Sj	w	Round key S[j] loaded to one of the two internal memories.
write_Sj	1	Synchronous write control signal for the round key S[j].
j	m	Index of the round key S[j] loaded using input Sj.
O	4w	Output block.
DONE	1	Asserted for one clock cycle when the output O is ready.

Note:
m is a size of index j. It is a minimum integer, such that $2^m - 1 \geq 2r+3$.

Protocol (1)

An external circuit first loads all round keys
S[0], S[1], S[2], …, S[2r+2], [2r+3]
to the two internal memories of the CIPHER unit.

The first memory stores values of S[j=2i], i.e., only round keys with even indices. The second memory stores values of S[j=2i+1], i.e. only round keys with odd indices.

Loading round keys is performed using inputs: Sj, j, write_Sj, clk.

Then, the external circuits, loads an input block I to the CIPHER unit, using inputs: I, write_I, clk.

After the input block I is loaded to the CIPHER unit, the encryption starts automatically.

Protocol (2)

When the encryption is completed, signal DONE becomes active, and the output O changes to the new value of the ciphertext.

The output O keeps the last value of the ciphertext at the output, until the next encryption is completed. Before the first encryption is completed, this output should be equal to zero.

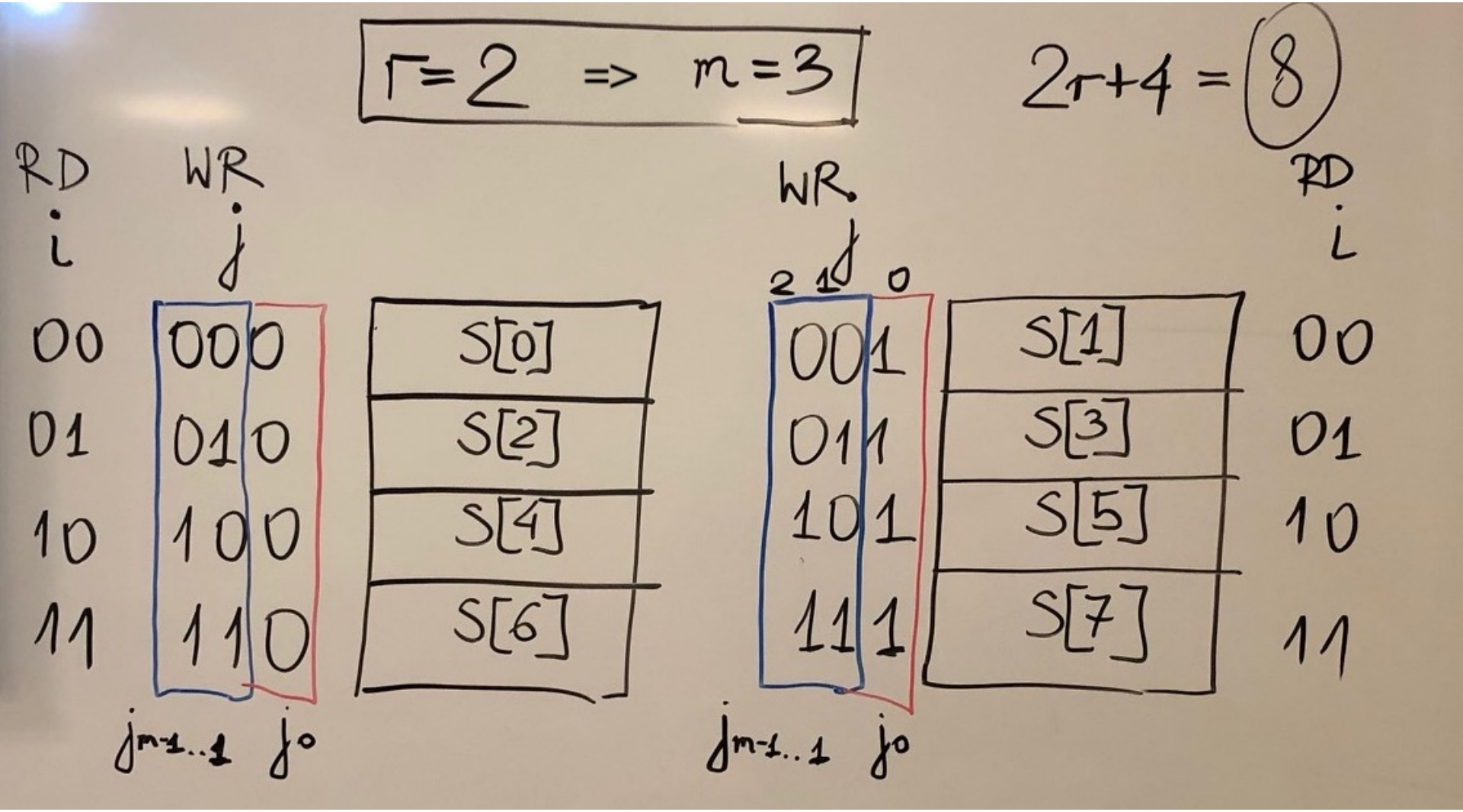
Assumptions

- 2r+4 clock cycles are used to load round keys to internal RAMs
- one round of the main for loop of the pseudocode executes in one clock cycle
- you can access only one position of each internal memory of round keys per clock cycle

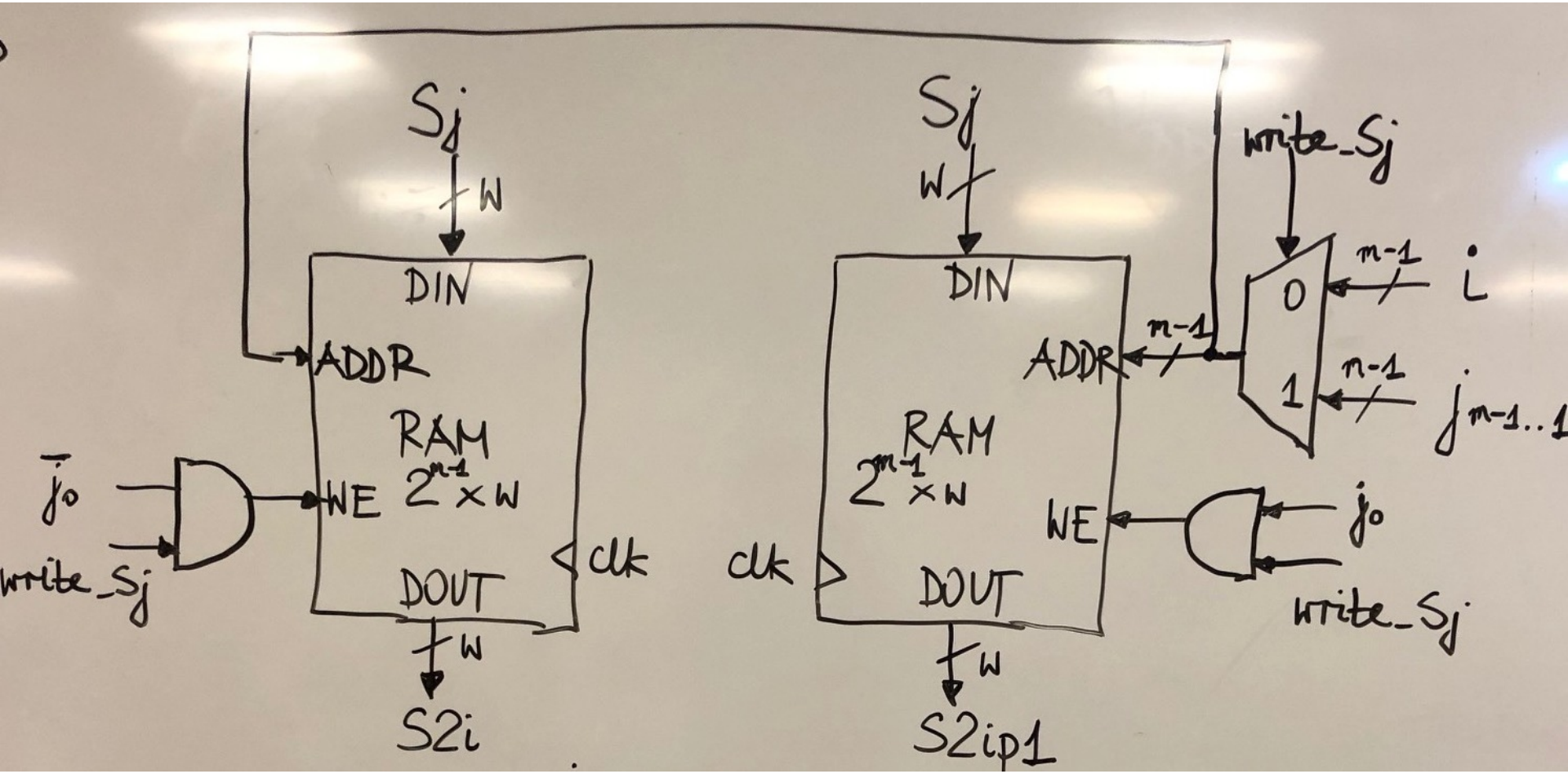
As a result, the encryption of a single input block I

should last r+2 clock cycles.

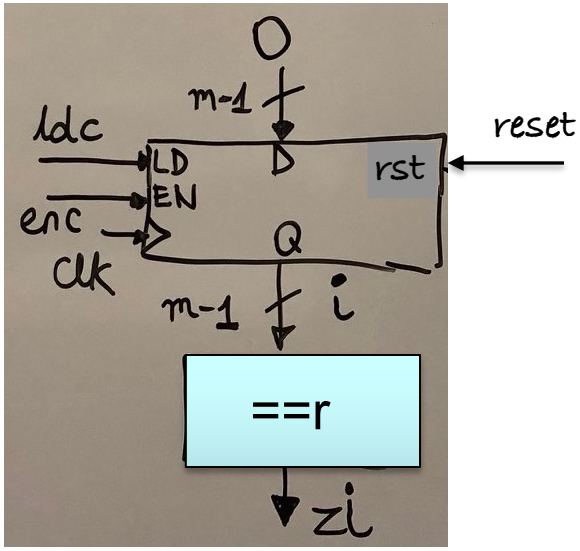
Memories of Round Keys – Contents



Datapath – Memories of Round Keys



Datapath – Loop Counter

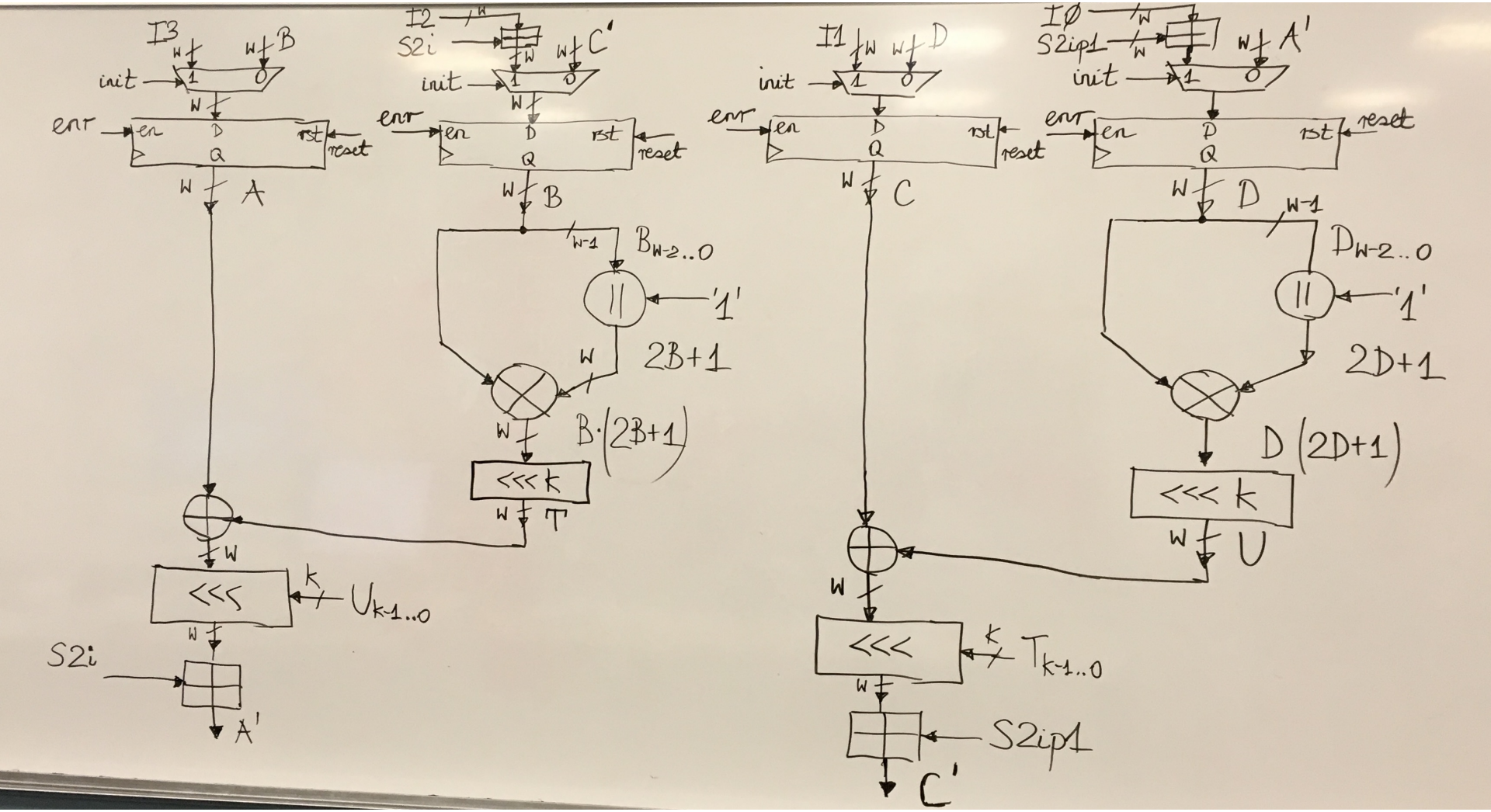


Pseudocode

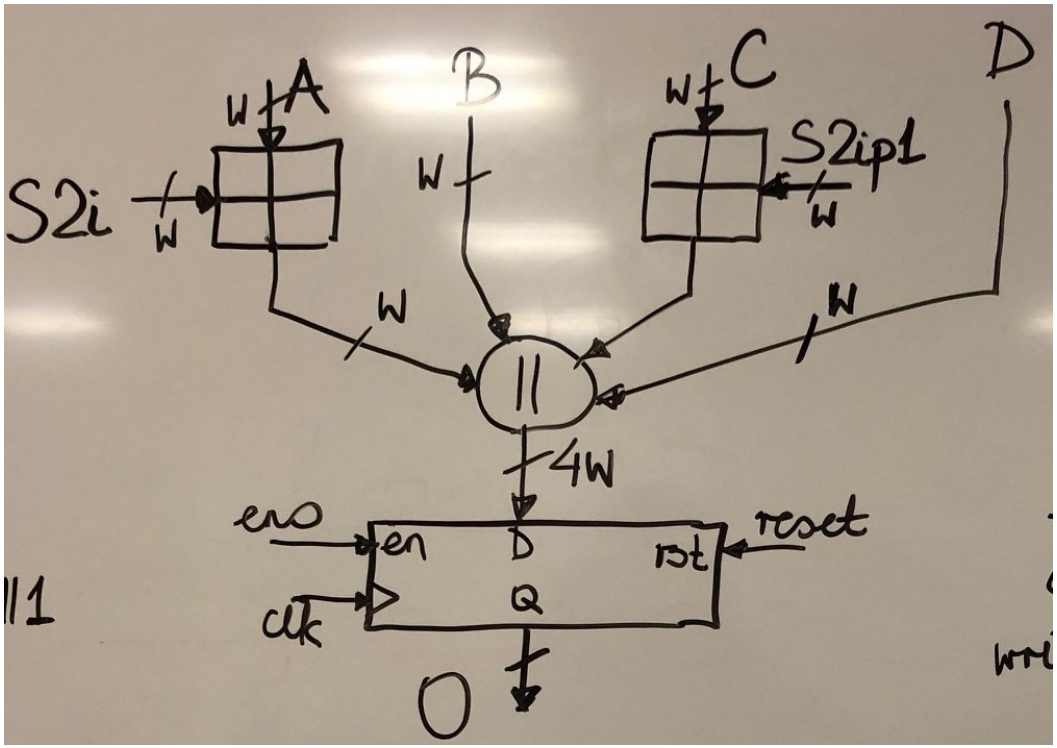
Split input I into four words, I3, I2, I1, I0, of the size of w bits each

```
A = I3; B = I2; C = I1; D=I0
B = B + S[0]
D = D + S[1]
        for i = 1 to r do
        {
                T = (B*(2B + 1)) <<< k
                U = (D*(2D + 1)) <<< k
                A' = ((A ⊕ T) <<< U) + S[2i]
                C' = ((C ⊕ U) <<< T) + S[2i + 1]
                (A, B, C, D)  =  (B, C', D, A')
        }
A = A + S[2r + 2]
C = C + S[2r + 3]
O = (A, B, C, D)
```

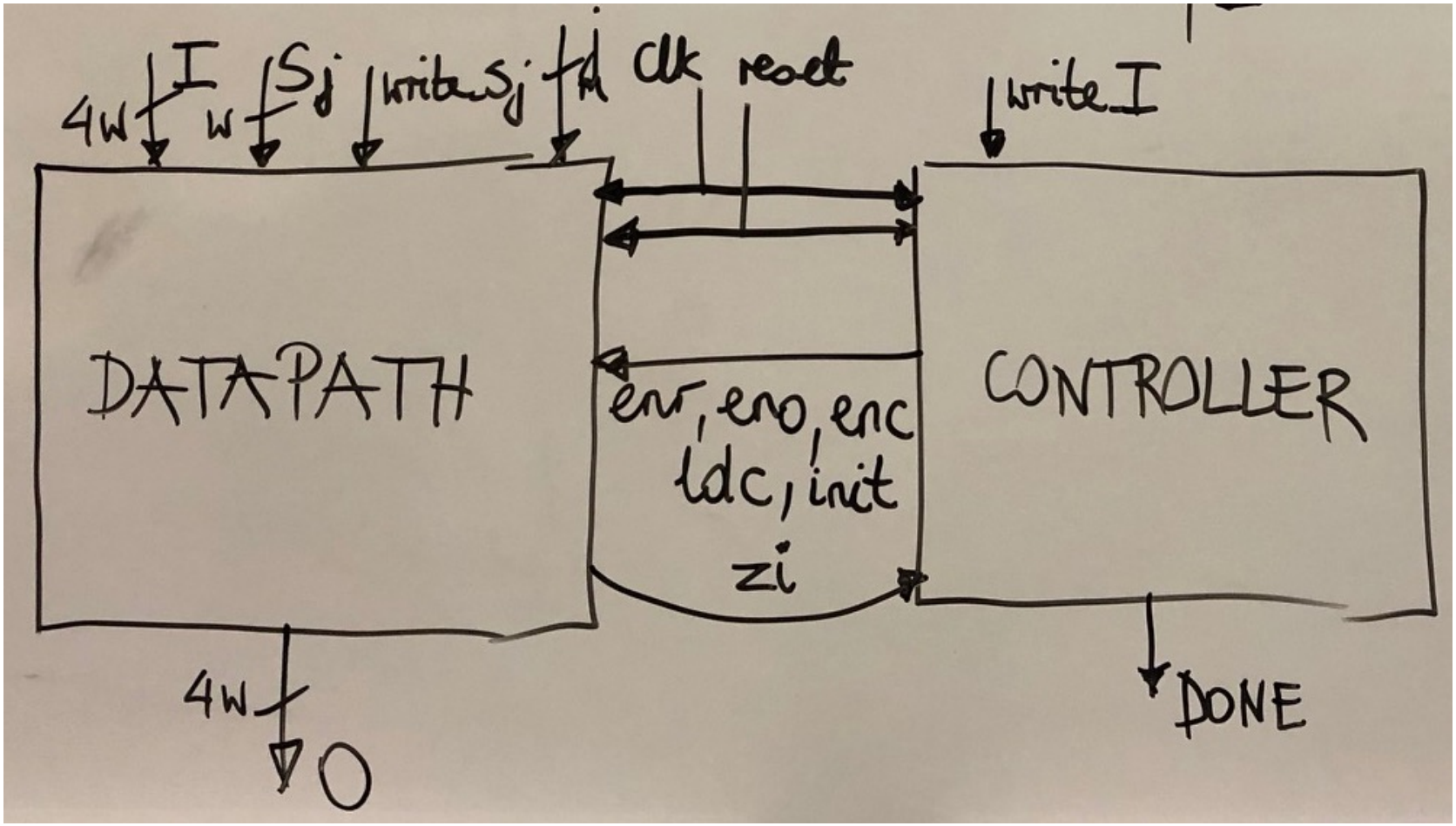
Datapath – Initialization & Main Loop



Datapath – Finalization



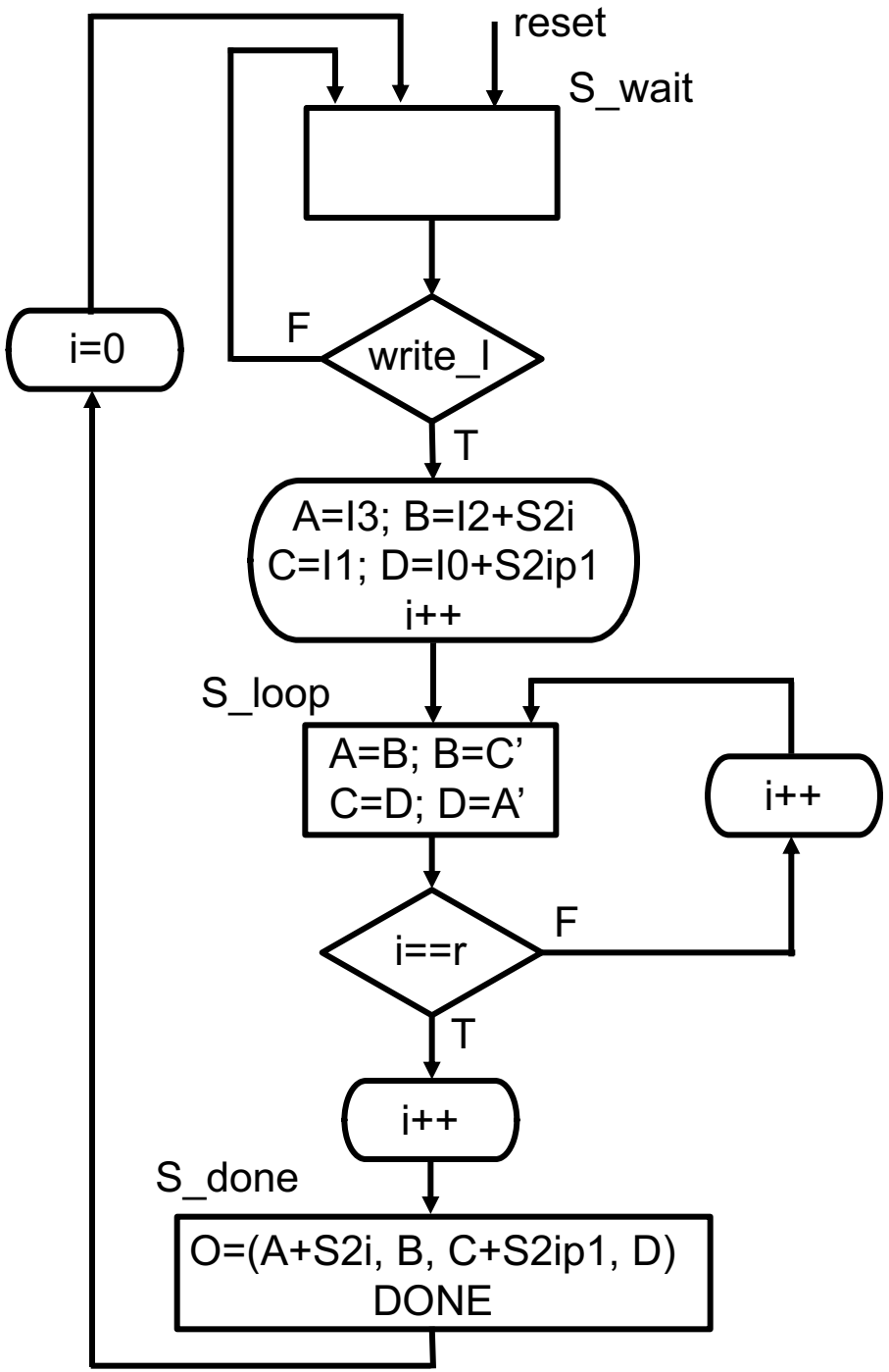
Interface with the Division into the Datapath and Controller

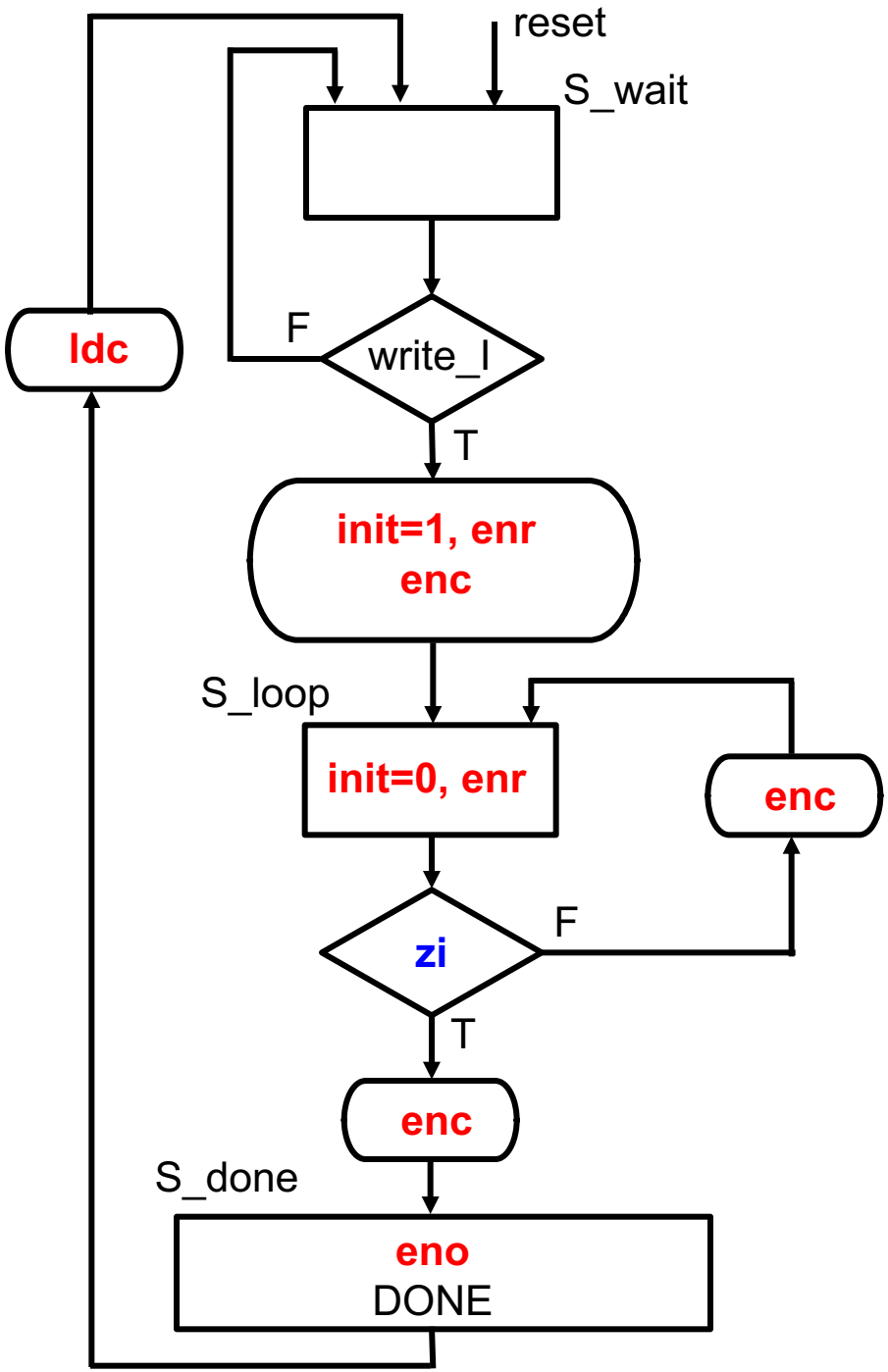


Pseudocode

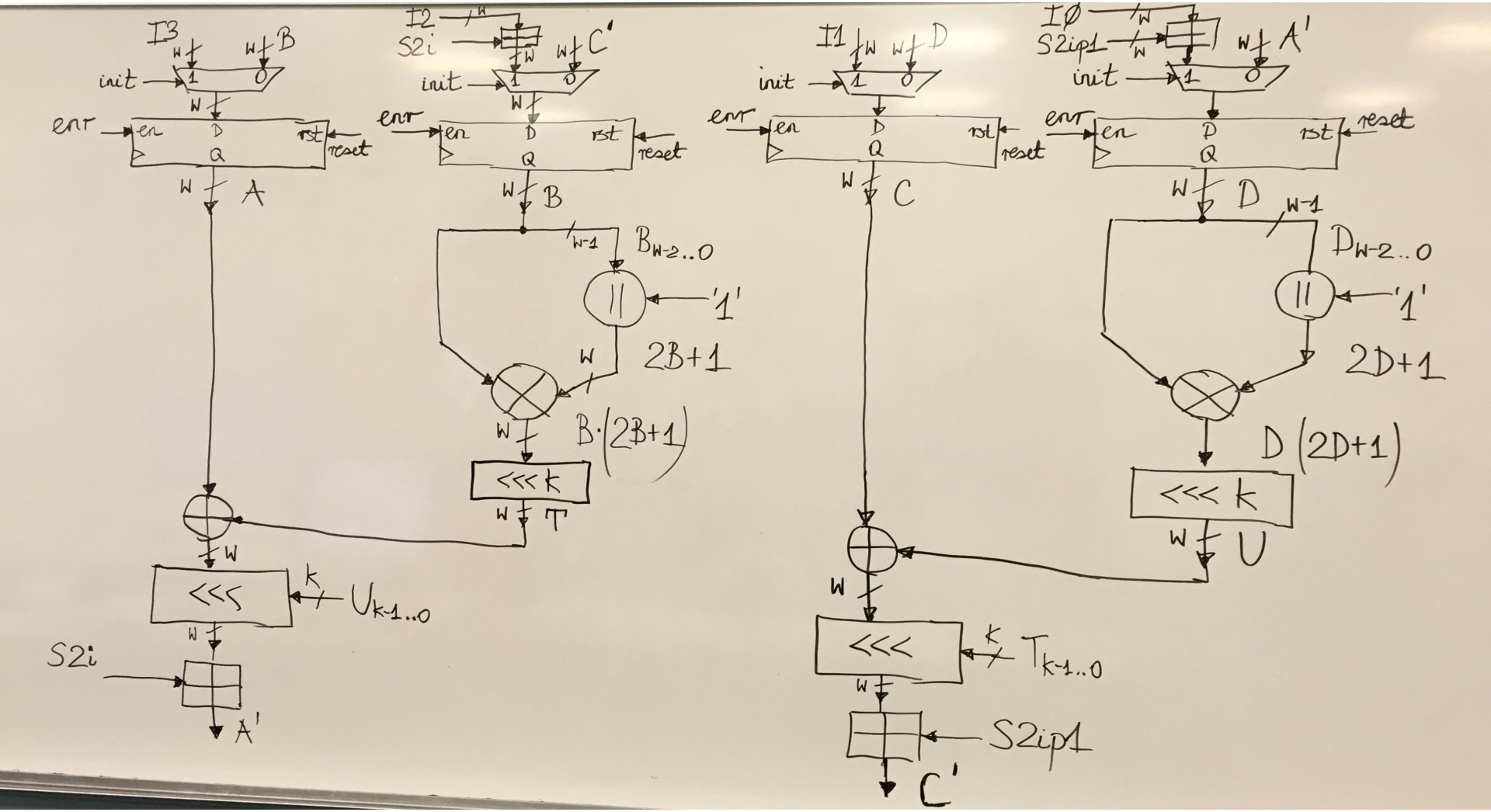
Split input I into four words, I3, I2, I1, I0, of the size of w bits each

```
A = I3; B = I2; C = I1; D=I0
B = B + S[0]
D = D + S[1]
      for i = 1 to r do
      {
            T = (B*(2B + 1)) <<< k
            U = (D*(2D + 1)) <<< k
            A' = ((A ⊕ T) <<< U) + S[2i]
            C' = ((C ⊕ U) <<< T) + S[2i + 1]
            (A, B, C, D)  =  (B, C', D, A')
      }
A = A + S[2r + 2]
C = C + S[2r + 3]
O = (A, B, C, D)
```

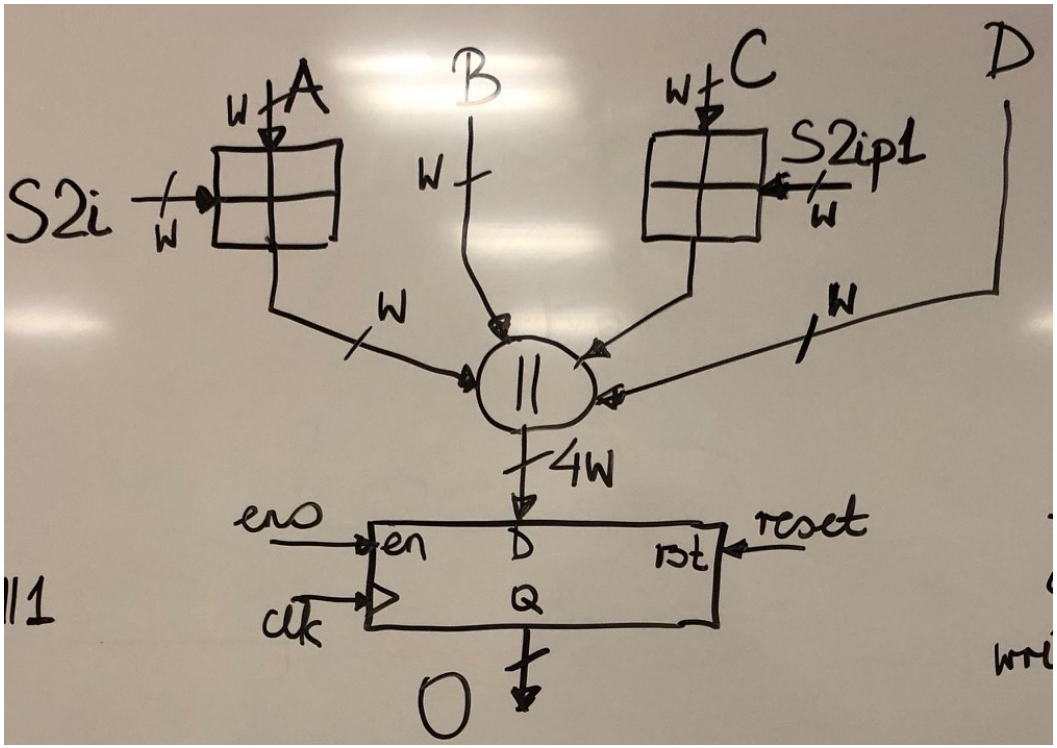

reset
S_wait
F
write_I
T
i=0
A=I3; B=I2+S2i
C=I1; D=I0+S2ip1
i++
S_loop
A=B; B=C'
C=D; D=A'
i++
i==r
F
T
i++
S_done
O=(A+S2i, B, C+S2ip1, D)
DONE


reset
S_wait
F
write_I
T
ldc
init=1, enr
enc
S_loop
init=0, enr
enc
zi
F
T
enc
S_done
eno
DONE

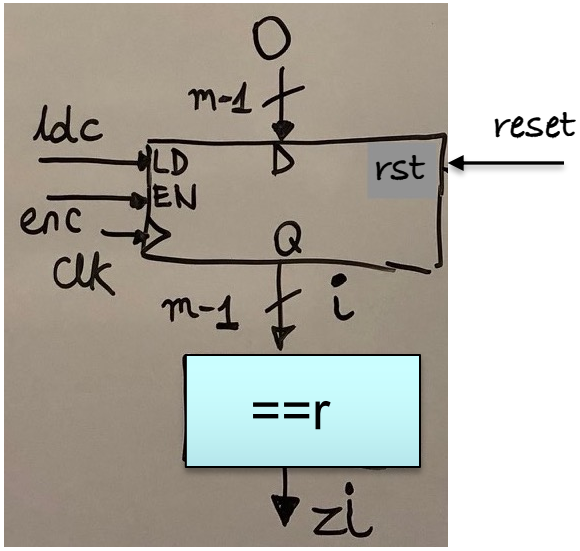
Datapath – Initialization & Main Loop



Datapath – Finalization



Datapath – Loop Counter



Timing Analysis – Key Setup & Encryption

Notation:

T_{CLK} – clock period in ns

N_M – number of encrypted message blocks I

$$T_{KeySetup}[cycles] = 2r+4$$

$$T_{KeySetup}[ns] = (2r+4) \cdot T_{CLK}$$

$$\mathbf{T_{Encryption}(N_M)[cycles] = (r+2) \cdot N_M}$$

$$T_{Encryption}(N_M)[ns] = (r+2) \cdot N_M \cdot T_{CLK}$$

Timing Analysis - Throughput

Notation:

T_{CLK} – clock period in ns

N_M – number of encrypted message blocks I

w – word size

$$Thr_{Encryption}(N_M)[Gbit/s] = \frac{4 \cdot w \cdot N_M}{(r+2) \cdot N_M \cdot T_{CLK}} = \frac{4 \cdot w}{(r+2) \cdot T_{CLK}}$$

r & w constants, determined by the security analysis of the cipher

For the security equivalent to AES-128, r=20 & w=32

T_{CLK} dependent on w, returned by the FPGA tools based on the static timing analysis of the critical path



Class Exercise 3

CIPHER – Lightweight Architecture

Pseudocode

Split input I into four words, I3, I2, I1, I0, of the size of w bits each

```
A = I3; B = I2; C = I1; D=I0
B = B + S[0]
D = D + S[1]
        for i = 1 to r do
        {
                T = (B*(2B + 1)) <<< k
                U = (D*(2D + 1)) <<< k
                A = ((A ⊕ T) <<< U) + S[2i]
                C = ((C ⊕ U) <<< T) + S[2i + 1]
                (A, B, C, D)  =  (B, C, D, A)
        }
A = A + S[2r + 2]
C = C + S[2r + 3]
O = (A, B, C, D)
```

Notation

w: word size, e.g., w=32 (constant)

k: $\log_2(w)$ (constant)

A, B, C, D, U, T: w-bit variables

I3, I2, I1, I0: Four w-bit words of the input I

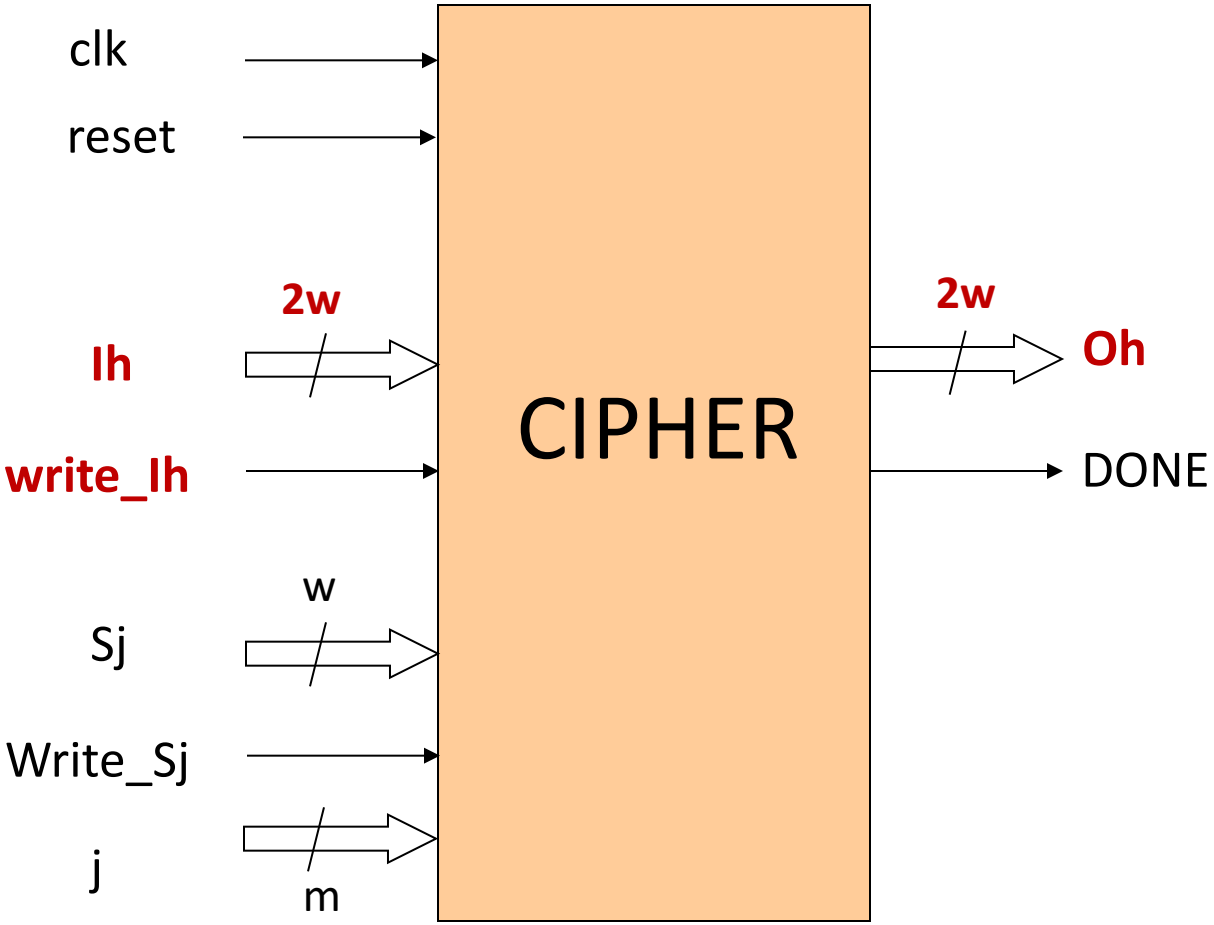
r: number of rounds (constant)

O: output of the size of 4w bits

S[j] : 2r+4 round keys stored in **one RAM**.

Each key is a w-bit word.

Circuit Interface



Interface Table

Port	Width	Meaning
clk	1	System clock.
reset	1	System reset – clears internal registers.
Ih	**2w**	**One half of an Input block. I3:I2 first, I1:I0 second.**
write_Ih	1	Synchronous write control signal for **a half of the input block Ih**. After **two halves of the input block** I are written to the CIPHER unit, the encryption of I starts automatically.
Sj	w	Round key S[j] loaded to one of the two internal memories.
write_Sj	1	Synchronous write control signal for the round key S[j].
j	m	Index of the round key S[j] loaded using input Sj.
Oh	**2w**	**One half of an Output block. O3:O2 first, O1:O0 second.**
DONE	1	Asserted for one clock cycle when the output O is ready.

Note:

m is a size of index j. It is a minimum integer, such that $2^m - 1 \geq 2r+3$.

Protocol (1)

An external circuit first loads all round keys
S[0], S[1], S[2], …, S[2r+2], [2r+3]
to **the internal memory** of the CIPHER unit.

Loading round keys is performed using inputs: Sj, j, write_Sj, clk.

Then, the external circuits, loads an input block I to the CIPHER unit, **in two clock cycles**, using inputs: **Ih**, **write_Ih**, clk.

After the input block I is loaded to the CIPHER unit, the encryption starts automatically.

Protocol (2)

When the encryption is completed, signal DONE becomes active, and the output O changes to the new value of the ciphertext.

The output O keeps the last value of the ciphertext at the output, until the next encryption is completed. Before the first encryption is completed, this output should be equal to zero.

Assumptions

- 2r+4 clock cycles are used to load round keys to internal RAMs
- one round of the main for loop of the pseudocode executes in **4 clock cycles**
- you can access only one position of **the internal memory** of round keys per clock cycle

As a result, the encryption of a single input block I should last **4r+4** clock cycles.

Pseudocode

i represents an index of a round key during encryption, not a round number

Split input I into four words, I3, I2, I1, I0, of the size of w bits each

```
A = I3; B = I2; C = I1; D=I0
B = B + S[i=0]
D = D + S[i=1]
      for n = 1 to r do
      {
            T = (B*(2B + 1)) <<< k
            U = (D*(2D + 1)) <<< k
            A' = ((A ⊕ T) <<< U) + S[i=2n]
            C' = ((C ⊕ U) <<< T) + S[i=2n + 1]
            (A, B, C, D)  =  (B, C', D, A')
      }
A = A + S[i=2r + 2]
C = C + S[i=2r + 3]
O = (A, B, C, D)
```

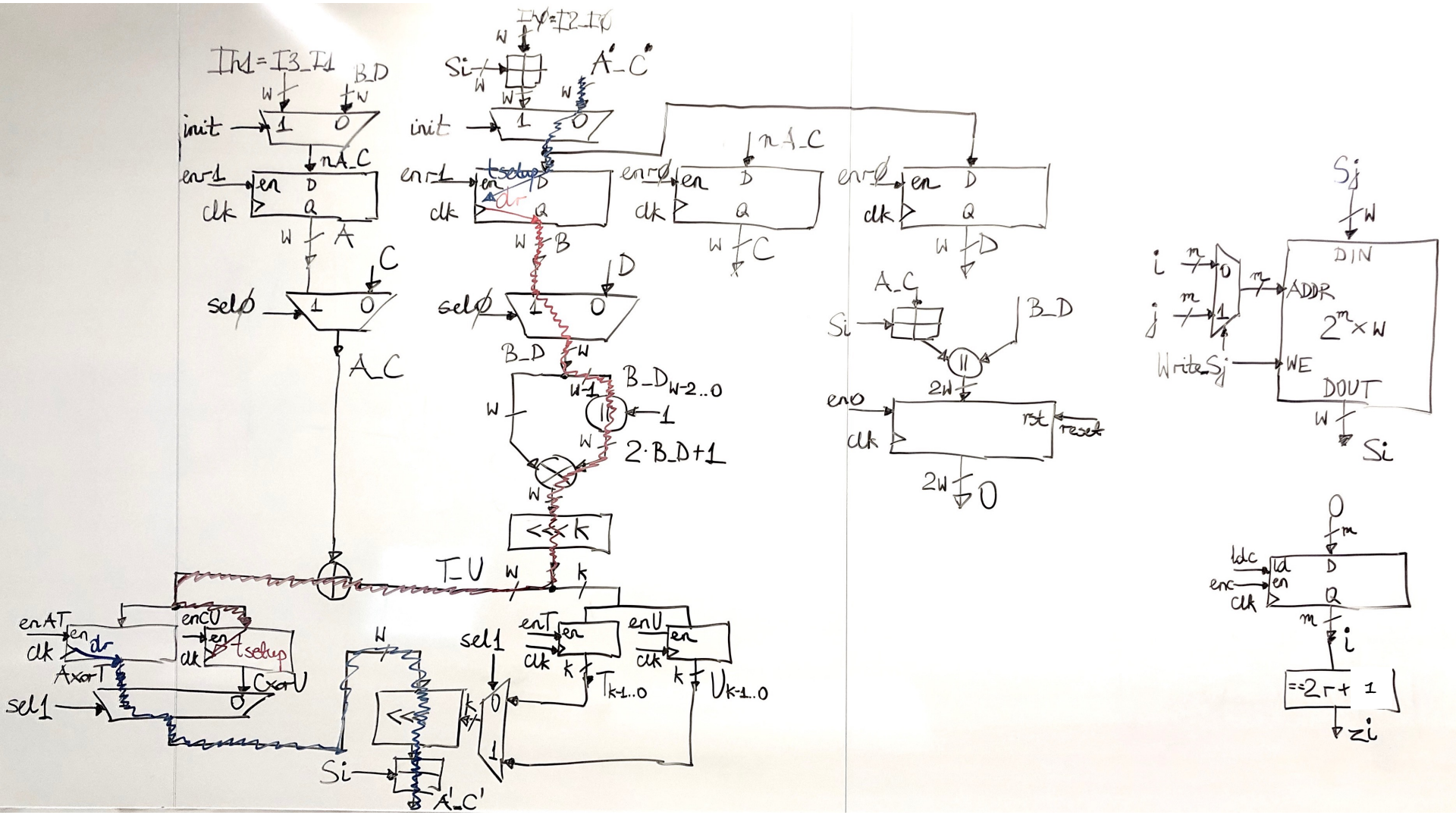
Pseudocode

i represents an index of a round key during encryption, not a round number

Split input I into four words, I3, I2, I1, I0, of the size of w bits each

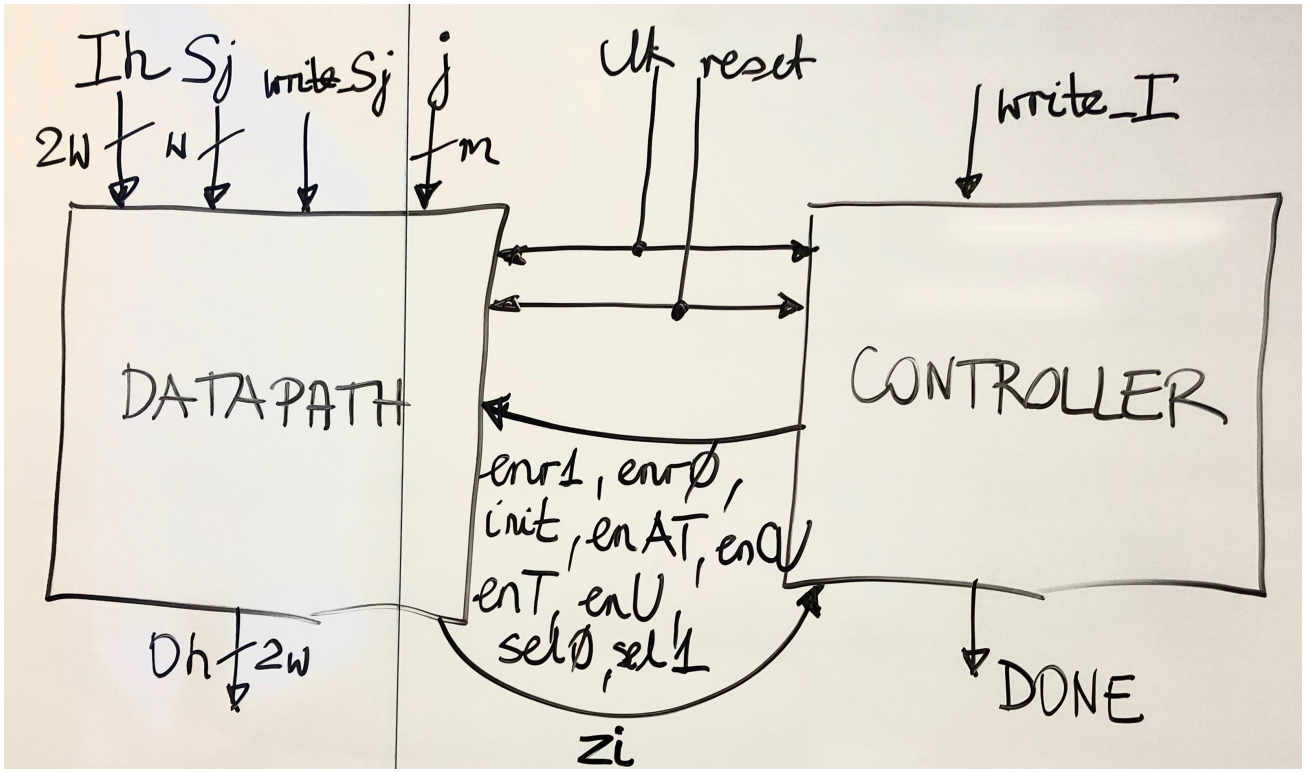
```
A = I3; B = I2; C = I1; D=I0
B = B + S[i=0]
D = D + S[i=1]
      for n = 1 to r do
      {
            T = (B*(2B + 1)) <<< k
            AxorT = A ⊕ T
            U = (D*(2D + 1)) <<< k
            CxorU = C ⊕ U
            A' = (AxorT <<< U) + S[i=2n]
           (C, D)  =  (D, A')
            C' = (CxorU <<< T) + S[i=2n + 1]
           (A, B)  =  (B, C')
      }
A = A + S[i=2r + 2]
C = C + S[i=2r + 3]
O = (A, B, C, D)
```

Datapath



i represents an index of a round key during encryption, not a round number

Interface with the Division into the Datapath and Controller

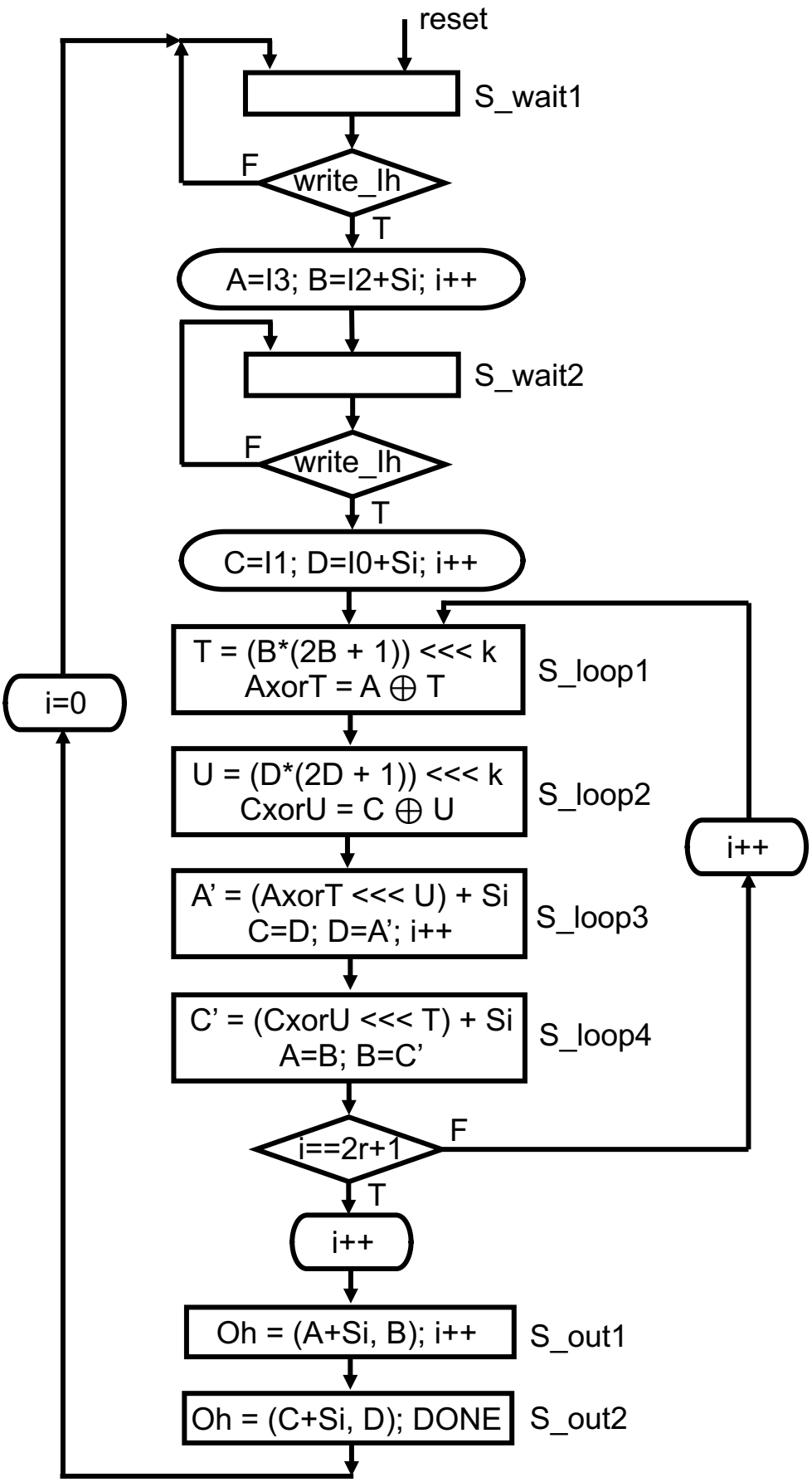


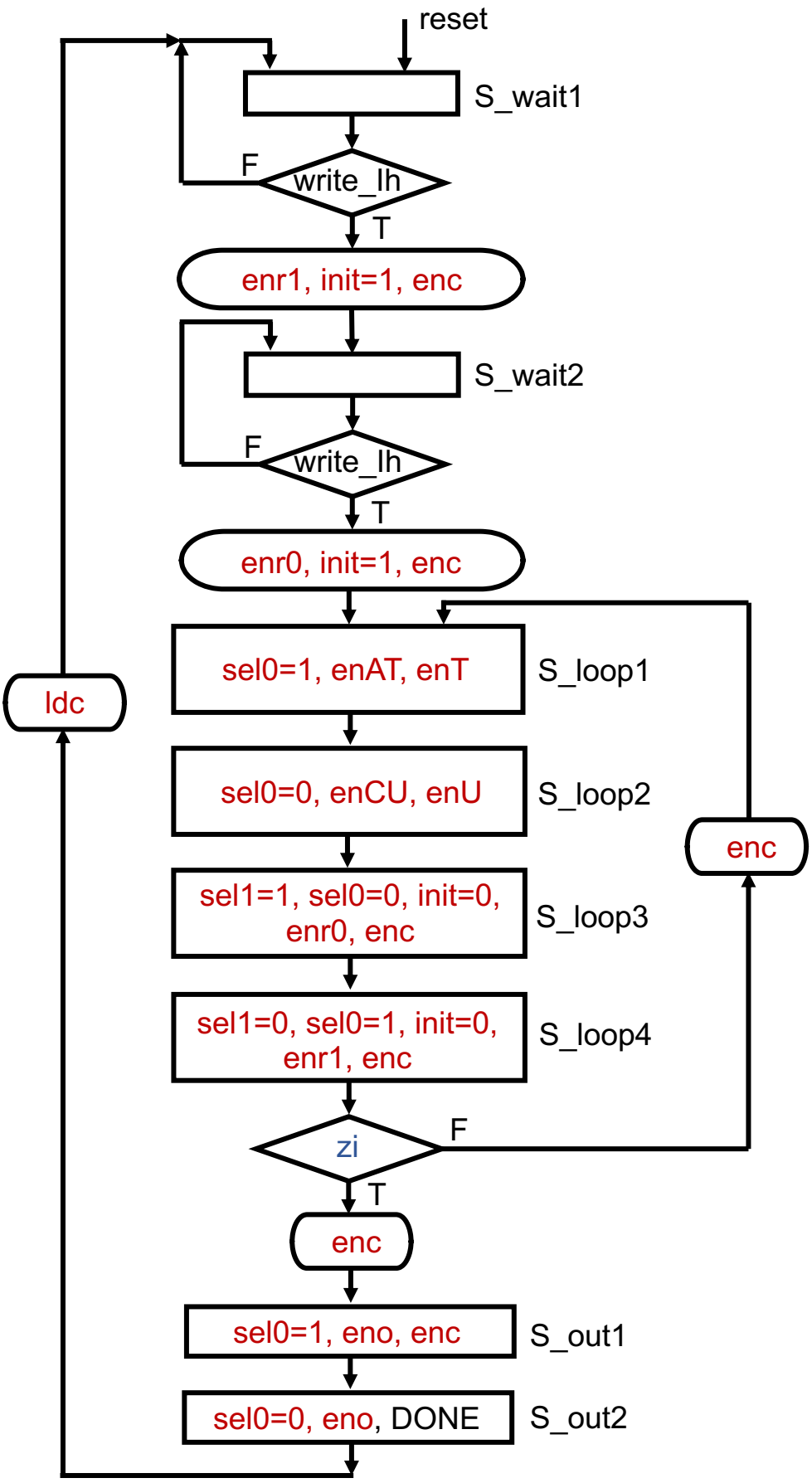
Pseudocode

i represents an index of a round key during encryption, not a round number

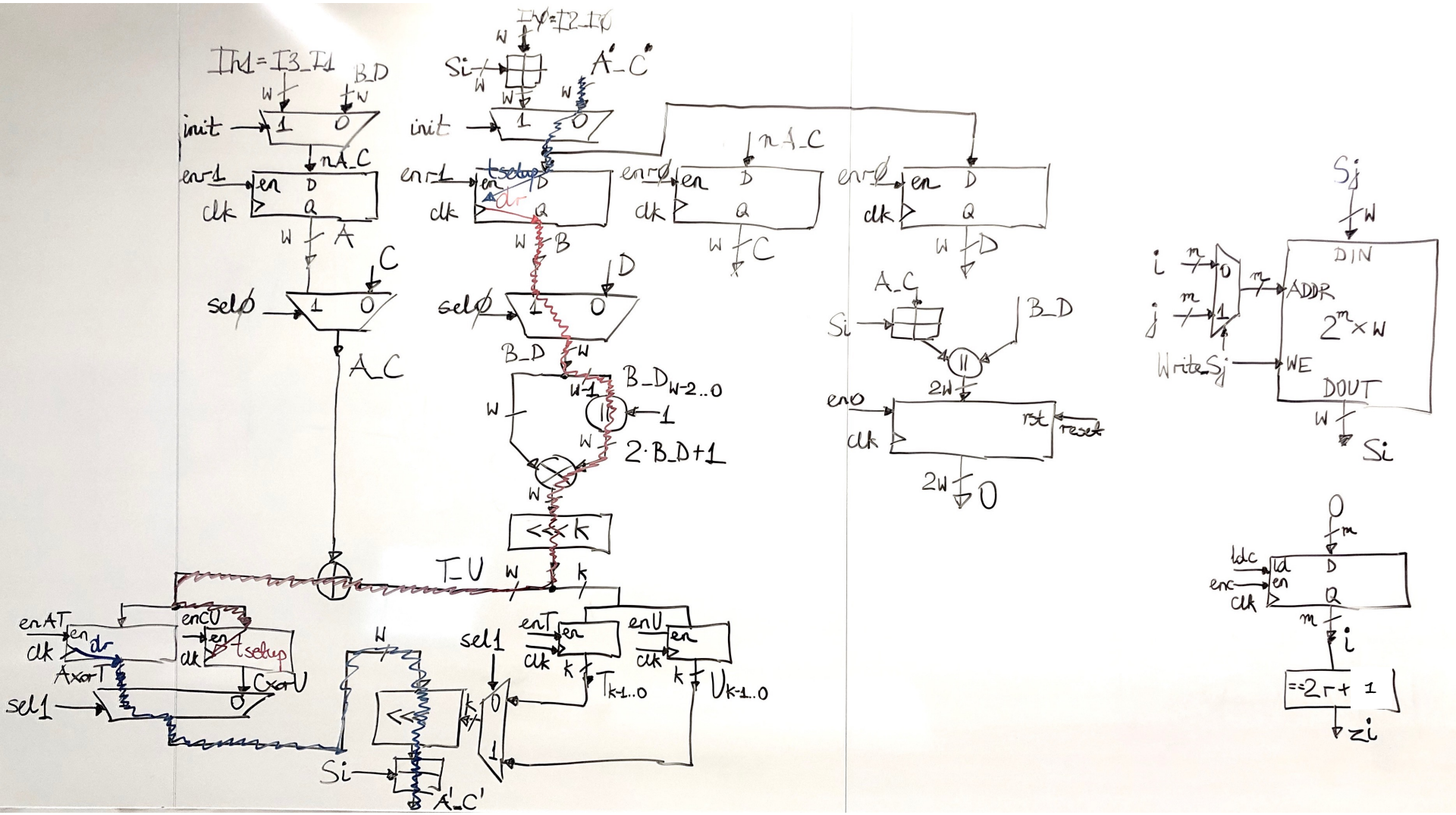
Split input I into four words, I3, I2, I1, I0, of the size of w bits each

```
A = I3; B = I2; C = I1; D=I0
B = B + S[i=0]
D = D + S[i=1]
      for n = 1 to r do
      {
            T = (B*(2B + 1)) <<< k
            AxorT = A ⊕ T
            U = (D*(2D + 1)) <<< k
            CxorU = C ⊕ U
            A' = (AxorT <<< U) + S[i=2n]
           (C, D)  =  (D, A')
            C' = (CxorU <<< T) + S[i=2n + 1]
           (A, B)  =  (B, C')
      }
A = A + S[i=2r + 2]
C = C + S[i=2r + 3]
O = (A, B, C, D)
```


reset
S_wait1
F
write_lh
T
A=I3; B=I2+Si; i++
S_wait2
F
write_lh
T
C=I1; D=I0+Si; i++
T = (B*(2B + 1)) <<< k
AxorT = A ⊕ T
S_loop1
U = (D*(2D + 1)) <<< k
CxorU = C ⊕ U
S_loop2
A' = (AxorT <<< U) + Si
C=D; D=A'; i++
S_loop3
C' = (CxorU <<< T) + Si
A=B; B=C'
S_loop4
i==2r+1
F
T
i++
i=0
Oh = (A+Si, B); i++
S_out1
Oh = (C+Si, D); DONE
S_out2


reset
S_wait1
F
write_lh
T
enr1, init=1, enc
S_wait2
F
write_lh
T
enr0, init=1, enc
ldc
sel0=1, enAT, enT
S_loop1
sel0=0, enCU, enU
S_loop2
sel1=1, sel0=0, init=0, enr0, enc
S_loop3
sel1=0, sel0=1, init=0, enr1, enc
S_loop4
zi
F
T
enc
enc
sel0=1, eno, enc
S_out1
sel0=0, eno, DONE
S_out2

Datapath



i represents an index of a round key during encryption, not a round number

Timing Analysis – Key Setup & Encryption

Notation:

T_{CLK} – clock period in ns

N_M – number of encrypted message blocks I

$$T'_{KeySetup}[cycles] = 2r+4$$

$$T'_{KeySetup}[ns] = (2r+4)\cdot \mathbf{T'_{CLK}}$$

$$T'_{Encryption}(N_M)[cycles] = (\mathbf{4r+4}) \cdot N_M$$

$$T'_{Encryption}(N_M)[ns] = \mathbf{4\cdot(r+1)} \cdot N_M \cdot \mathbf{T'_{CLK}}$$

Timing Analysis - Throughput

Notation:

T_{CLK} – clock period in ns

N_M – number of encrypted message blocks I

w – word size

$$Thr'_{Encryption}(N_M)[Gbit/s] = \frac{4 \cdot w \cdot N_M}{4 \cdot (r+1) \cdot N_M \cdot T'_{CLK}} = \frac{w}{(r+1) \cdot T'_{CLK}}$$

r & w constants, determined by the security analysis of the cipher

For the security equivalent to AES-128, r=20 & w=32

T'_{CLK} dependent on w, returned by the FPGA tools based on the static timing analysis of the critical path

Timing Analysis – Throughput Ratio

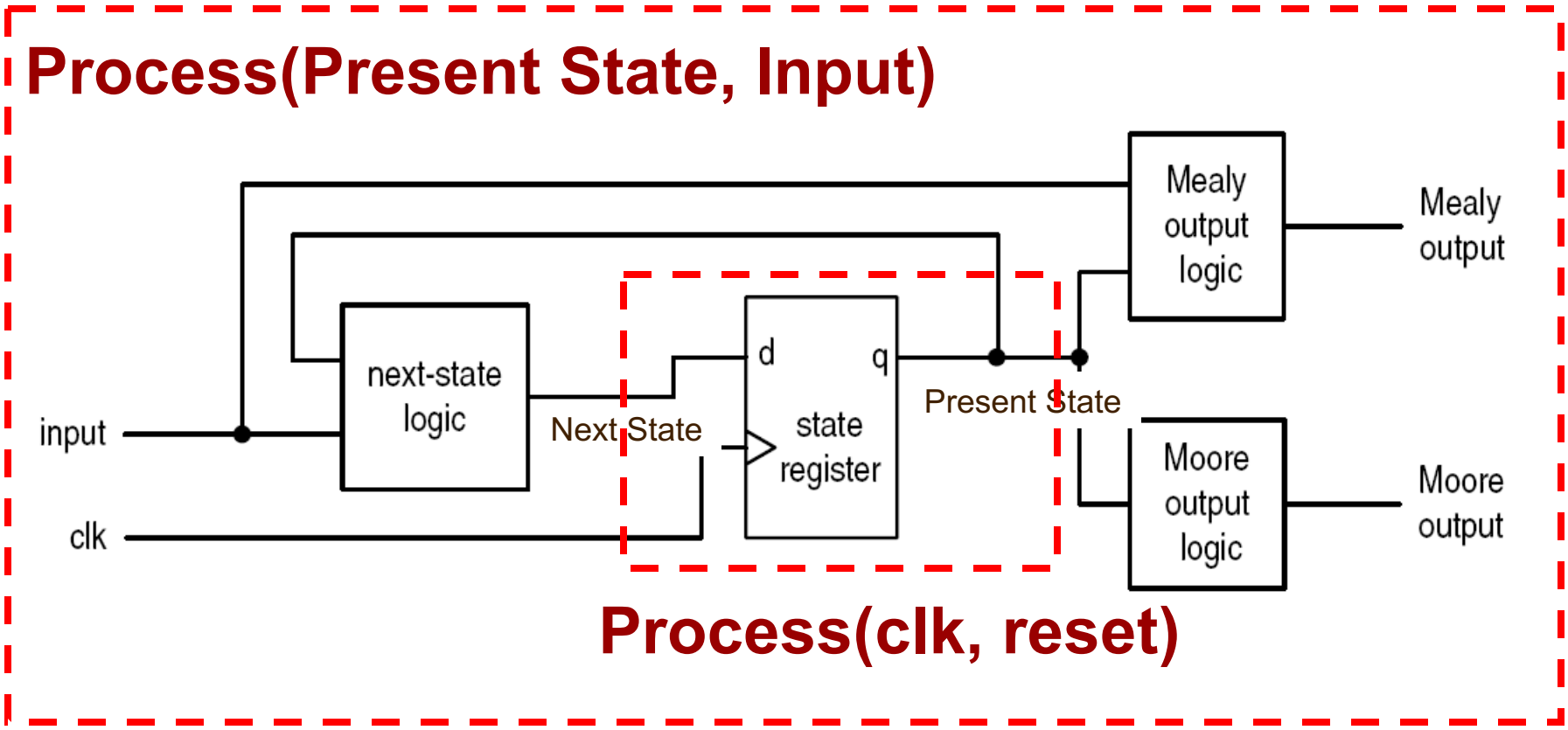
$$\frac{Thr}{Thr'} = \frac{\frac{4 \cdot W}{(r+2) \cdot T_{ak}}}{\frac{W}{(r+1) \cdot T_{ak}'}} = 4 \cdot \frac{r+1}{r+2} \cdot \frac{T_{ak}'}{T_{ak}} \approx 4 \cdot \frac{1}{2} = 2$$

Throughput decreases by approximately a factor of 2



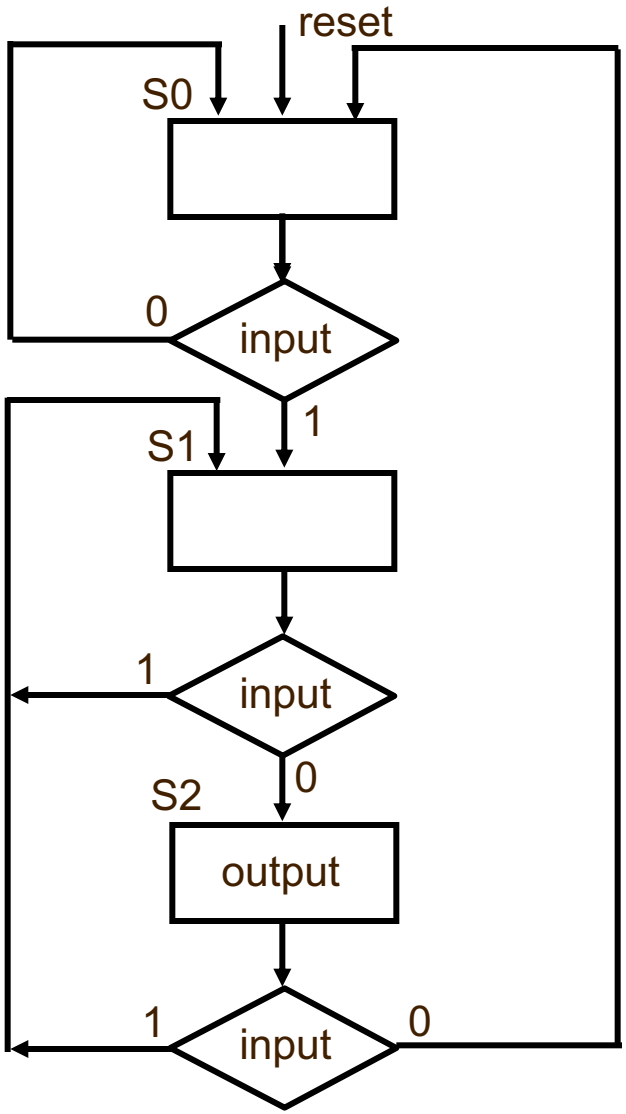
Finite State Machines in VHDL

Recommended FSM Coding Style



Based on RTL Hardware Design by P. Chu

ASM Chart of Moore Machine



Moore FSM in VHDL (1)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY FSM_Moore IS
      PORT ( clk          : IN     STD_LOGIC ;
             reset        : IN     STD_LOGIC ;
             input        : IN     STD_LOGIC ;
             output       : OUT    STD_LOGIC) ;
END FSM_Moore ;
```

Moore FSM in VHDL (1)

```
ARCHITECTURE behavioral of FSM_Moore IS
TYPE state IS (S0, S1, S2);
SIGNAL Present_State, Next_State: state;

BEGIN

U_Moore: PROCESS (clk, reset)
BEGIN
      IF(reset = '1') THEN
         Present_State <= S0;
      ELSIF rising_edge(clk) THEN
         Present_State <= Next_State;
      END IF;
END PROCESS;
```

Moore FSM in VHDL (2)

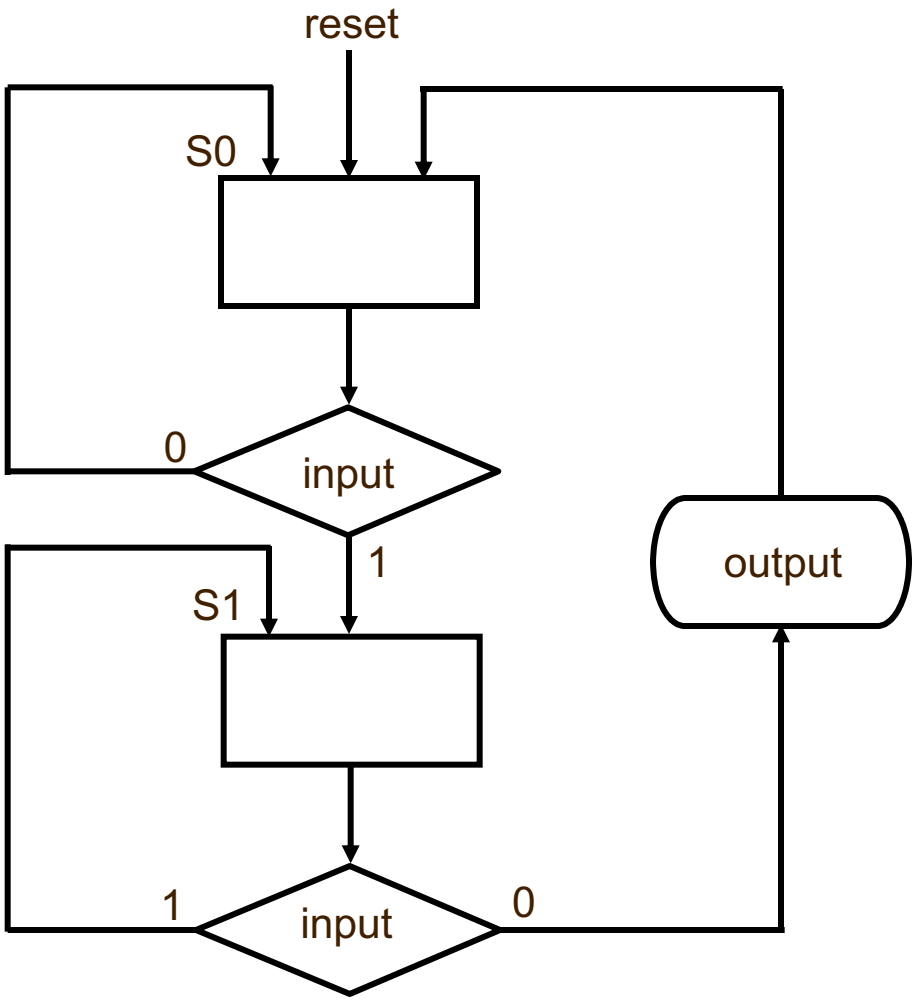
```
Next_State_Output:
PROCESS (Present_State, input)
BEGIN
      Next_State <= Present_State;
      output <= '0';
      CASE Present_State IS
            WHEN S0 =>
             IF input = '1' THEN
                Next_State <= S1;
             ELSE
                Next_State <= S0;
             END IF;
```

Moore FSM in VHDL (3)

```
            WHEN S1 =>
                IF input = '0' THEN
                    Next_State <= S2;
                ELSE
                    Next_State <= S1;
                END IF;
            WHEN S2 =>
                output <= '1' ;
                IF input = '1' THEN
                    Next_State <= S1;
                ELSE
                    Next_State <= S0;
                END IF;
        END CASE;
END PROCESS;

END behavioral;
```

ASM Chart of Mealy Machine



Mealy FSM in VHDL (1)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY FSM_Mealy IS
      PORT ( clk : IN        STD_LOGIC ;
             reset     : IN      STD_LOGIC ;
             input               : IN      STD_LOGIC ;
             output              : OUT     STD_LOGIC) ;
END FSM_Mealy ;
```

Mealy FSM in VHDL (1)

```
ARCHITECTURE behavioral of FSM_Mealy IS
TYPE state IS (S0, S1);
SIGNAL Present_State, Next_State: state;

BEGIN
U_Mealy: PROCESS(clk, reset)
BEGIN
      IF(reset = '1') THEN
         Present_State <= S0;
      ELSIF rising_edge(clk) THEN
         Present_State <= Next_State;
      END IF;
END PROCESS;
```

Mealy FSM in VHDL (2)

```
Next_State_Output:
PROCESS (Present_State, input)
BEGIN
      Next_State <= Present_State;
      output <= '0';
      CASE Present_State IS
            WHEN S0 =>
               IF input = '1' THEN
                    Next_State <= S1;
               ELSE
                    Next_State <= S0;
               END IF;
```

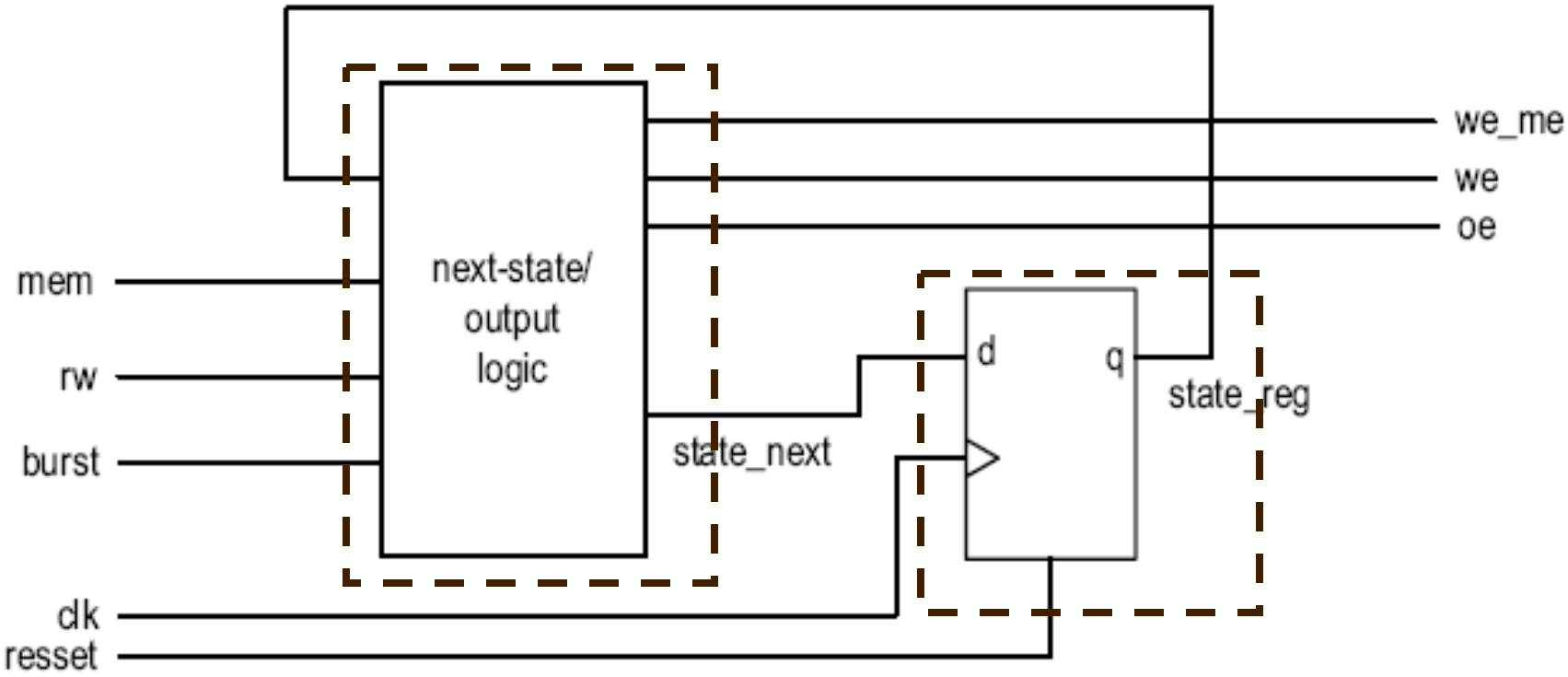
Mealy FSM in VHDL (3)

```
                WHEN S1 =>
                  IF input = '0' THEN
                        Next_State <= S0;
                        Output <= '1' ;
                  ELSE
                        Next_State <= S1;
                  END IF;
        END CASE;
END PROCESS;

END behavioral;
```

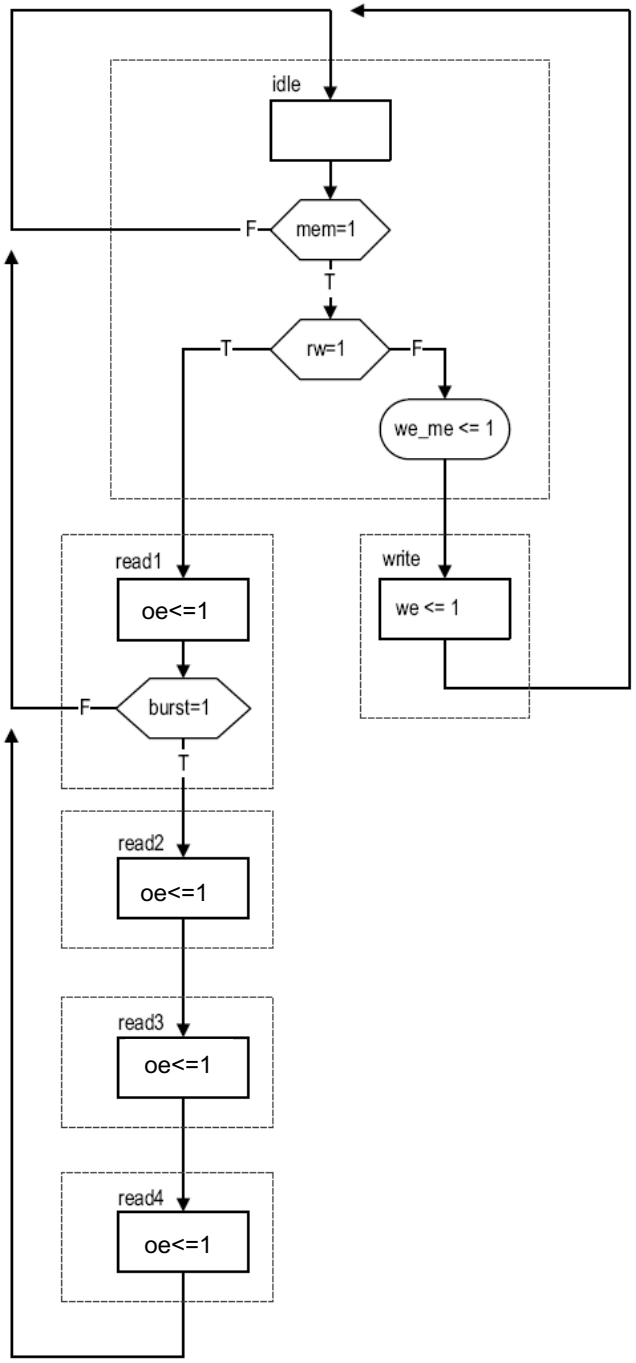
FSM Coding Style

Process(Present State,Inputs)



Process(clk, reset)

Memory Controller



```
library ieee;
use ieee.std_logic_1164.all;
entity mem_ctrl is
port (
   clk, reset: in std_logic;
   mem, rw, burst: in std_logic;
   oe, we, we_me: out std_logic);
end mem_ctrl ;

architecture mult_seg_arch of mem_ctrl is
   type mc_state_type is
      (idle, read1, read2, read3, read4, write);
   signal state_reg, state_next: mc_state_type;

begin
   -- state register
   process(clk, reset)
   begin
      if (reset='1') then
         state_reg <= idle;
      elsif (clk'event and clk='1') then
         state_reg <= state_next;
      end if;
   end process;
```

```
process(state_reg, mem, rw, burst)
begin
  oe <= '0';      -- default values
  we <= '0';
  we_me <= '0';
  case state_reg is
    when idle =>
      if mem='1' then
        if rw='1' then
          state_next <= read1;
        else
          state_next <= write;
          we_me <= '1';
        end if;
      else
        state_next <= idle;
      end if;
    when write =>
      state_next <= idle;
      we <= '1';
```

```
        when read1 =>
          if (burst='1') then
            state_next <= read2;
          else
            state_next <= idle;
          end if;
          oe <= '1';
        when read2 =>
          state_next <= read3;
          oe <= '1';
      when read3 =>
          state_next <= read4;
          oe <= '1';
        when read4 =>
          state_next <= idle;
          oe <= '1';
      end case;
end process;
```