Implementasi System Digital dengan VHDL

bagian 1 – review, kombinasional

Arif Sasongko

Tujuan

- Review Sistem Digital dan FPGA
- Memberi referensi/acuan untuk melakukan perancangan dan implementasi sistem digital berbasis FPGA
- Mempersiapkan untuk tugas besar kelompok ke-2

Bukan Tujuan

- Mempejari FPGA dan sistem digital →dianggap sudah ada dasar
- Mempalajari detail VHDL → tidak cukup 1 pertemuan
- Mempelajari fasilitas FPGA tertentu

Bahan

- Review Sistem Digital dan FPGA
- Review VHDL
- Contoh Design
- Eksperimen (dan tool)
- Review/Diskusi

Review Sistem Digital

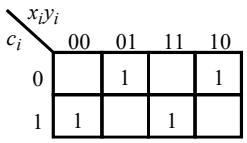
- Based on Boolean Algebra (0/1, AND, OR, ...)
- Transistor (technology dependent) → Gate → RTL→ (higher level)
- Contoh sistem digital:
  - rangkaian kombinasional → adder √
  - rangkaian sekuensial → FSM √
  - rangkaian kompleks (gabungan sekuensial dan kombinasional) → peripheral, co-processor √
  - processor →8051, ARM

Review Sistem Digital: Kombinasional

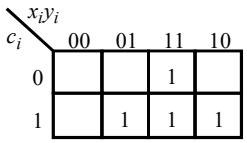
ADDER

c_i	x_i	y_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(a) Truth table

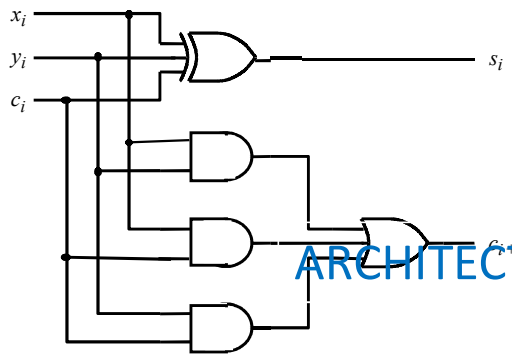


$s_i = x_i \oplus y_i \oplus c_i$



$c_{i+1} = x_iy_i + x_ic_i + y_ic_i$

(b) Karnaugh maps



(c) Circuit

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

LIBRARY

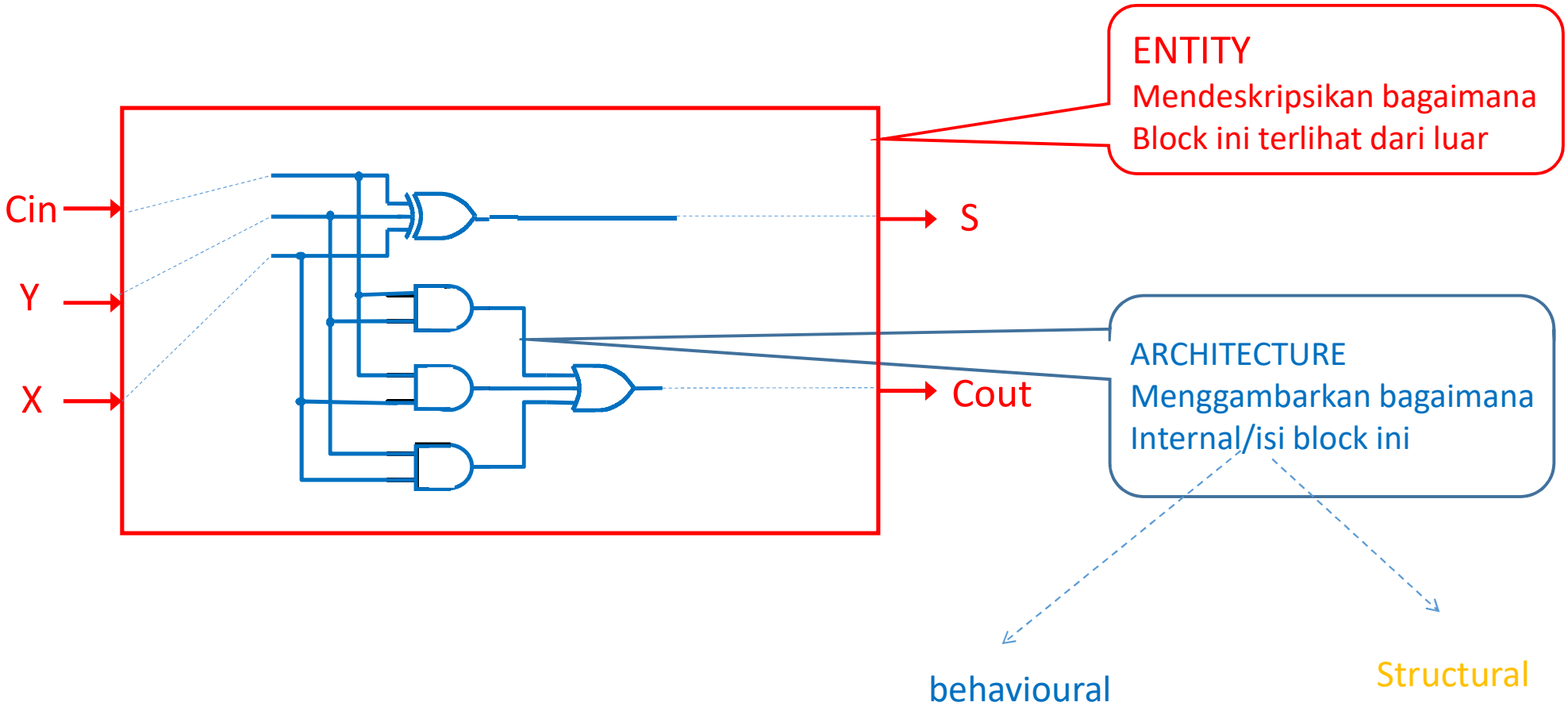
```
entity fulladder is
  port(
    X,Y,Cin : in std_logic;
    S,Cout  : out std_logic
    );
end fulladder;
```

ENTITY

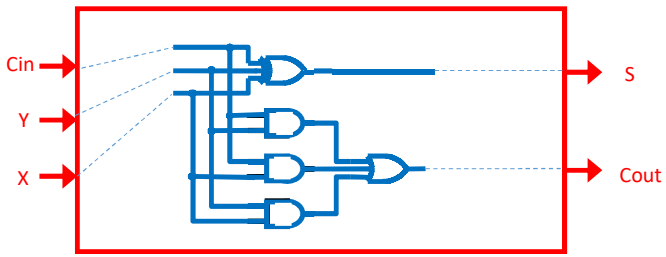
ARCHITECTURE

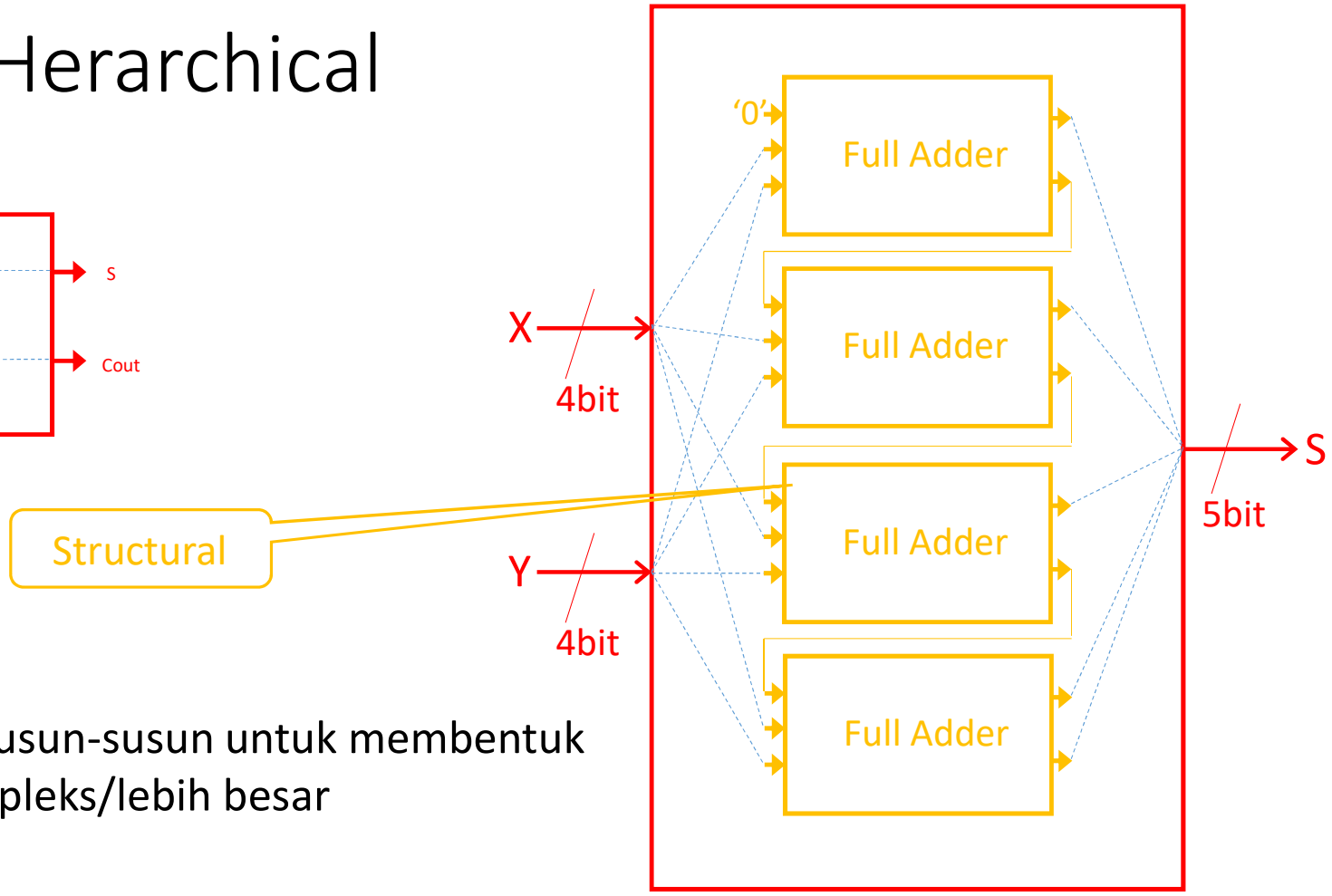
```
architecture Gate_level of fulladder is
begin
  S <= X xor Y xor Cin;
  Cout <= (X and Cin) or (X and Y) or
(X and Cin);
end Gate_level;
```

Review Sistem Digital: Kombinasional



Structural/Herarchical





Rangkaian Digital bisa disusun-susun untuk membentuk rangkaian yang lebih kompleks/lebih besar

Hati-hati dengan delay !!!

Structural/Herarchical (Contoh VHDL)

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

entity Adder_4bit is
  port(
   Input1: in std_logic_vector(3 downto 0);
   Input2: in std_logic_vector(3 downto 0);
   Result: out std_logic_vector(4 downto 0)
    );
end Adder_4bit;

architecture RTL of Adder_4bit is
Component fulladder is
  port(
    A,B,Cin : in std_logic;
    S,Cout  : out std_logic
    );
end component;
```

```
Signal Carry : std_logic_vector(3 downto
0);

begin
  FA1: fulladder port map
(Input1(0),Input2(0),'0',Result(0),Carry(0)
);
  FA2: fulladder port map
(Input1(1),Input2(1),Carry(0),Result(1),Car
ry(1));
  FA3: fulladder port map
(Input1(2),Input2(2),Carry(1),Result(2),Car
ry(2));
  FA4: fulladder port map
(Input1(3),Input2(3),Carry(2),Result(3),Car
ry(3));

  Result(4) <= Carry(3);
end RTL;
```

Template File VHDL

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

LIBRARY

```
ENTITY zzzz is
Port( A: in std_logic_vector 5 downto 0;
      B: out std_logic );
```

ENTITY

```
ARCITECTURE xxx of entity zzzz
    Component declaration
    Signal declaration
Begin
    Behaviour/Structure
End architecture xxx
```

ARCHITECTURE

Library

- WAJIB/SELALU digunakan (dalam RTL design):

Library ieee;

USE ieee.std_logic_1164.all;

- Mengunakan library sendiri

use work.my_package.all; (misal)

- Mengunakan library vendor

-Komponen FPGA

-Komponen untuk simulasi (VITAL)

- Bisa digunakan/Optional:

```
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use IEEE.std_logic_arith.all;
use IEEE.numeric_bit.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
use IEEE.math_complex.all;
```

Library -- contoh

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

ENTITY adderLPM IS
    PORT ( Cin    : IN    STD_LOGIC ;
           X, Y   : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           S      : OUT   STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           Cout   : OUT   STD_LOGIC ) ;
END adderLPM ;

ARCHITECTURE Structure OF adderLPM IS
BEGIN
    instance: lpm_add_sub
        GENERIC MAP (LPM_WIDTH => 4)
        PORT MAP (
            dataa => X, datab => Y, Cin => Cin, result => S, Cout => Cout ) ;
END Structure ;
```

Entity Declaration

- PORT
  - Direction (in-out-inout)
  - Type: scalar vs vector
  - STD_LOGIC vs BIT , 9 value logic (1,0,Z,-,X,h,l,W,U)
- GENERIC
  - Contoh: GENERIC(n: INTEGER := 4);

ARCHITECTURE

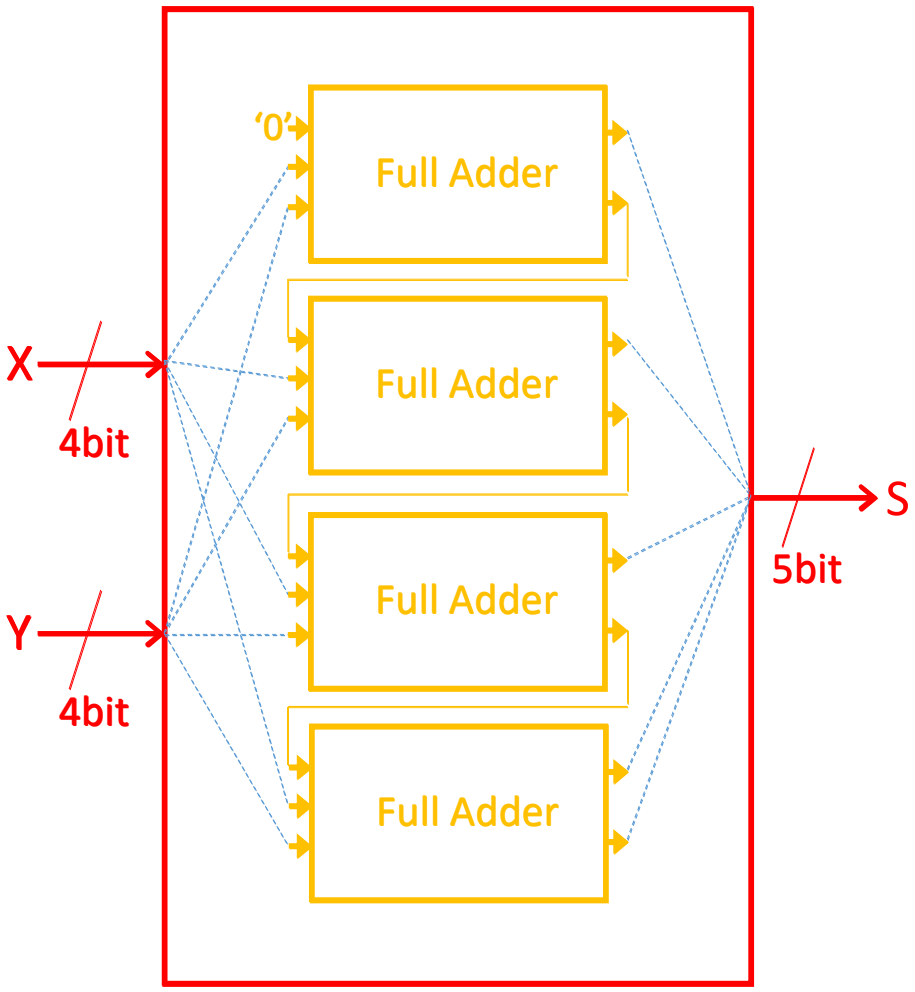
- COMPONENT mendeskripsikan rangkaian lain yang akan digunakan (semacam function prototype di C)
- **SIGNAL** mendeskripsikan signal-signal **internal** yang digunakan di dalam Architecture (tidak bisa diakses dari luar)
- Block-block/bagian yang ada didalam architecture pada dasarnya PARALEL (sehingga urutan tidak penting)
- Yang dilakukan pada waktu membuat code VHDL pada dasarnya adalah MENGGAMBAR RANGKAIAN dan BUKAN MENDESKRIPSIKAN ALGORITMA

CONTOH

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

entity Adder_N_Bit is
  Generic(N : integer := 8);
  Port(
      C_In : in std_logic;
      X        : in std_logic_vector(N-1 downto 0);
      Y        : in std_logic_vector(N-1 downto 0);
      Sum    : out std_logic_vector(N-1 downto 0);
      C_out: out std_logic
  );  end entity;

Architecture Structural of Adder_N_Bit is
component FULLADDER is
            port (
                        C_in        : in std_logic;
                        X                    : in std_logic;
                        Y                    : in std_logic;
                        Sum                  : out std_logic;
                        C_out       : out std_logic
            );
end component;
```

```
Signal C : std_logic_vector(N-1 downto 0);

begin

Adder_0 : FULLADDER
port map(
                        C_in => '0',
                        X  => X(0),
                        Y  => Y(0),
                        Sum => Sum(0),
                        C_out => C(0)
);

Adders : for i in  1 to N-1 generate
            Adder: FULLADDER
            port map(
                        C_in => C(i-1),
                        X  => X(i),
                        Y  => Y(i),
                        Sum => Sum(i),
                        C_out => C(i));
end generate;

C_out <= C(N-1);

end architecture structural;
```

CONTOH

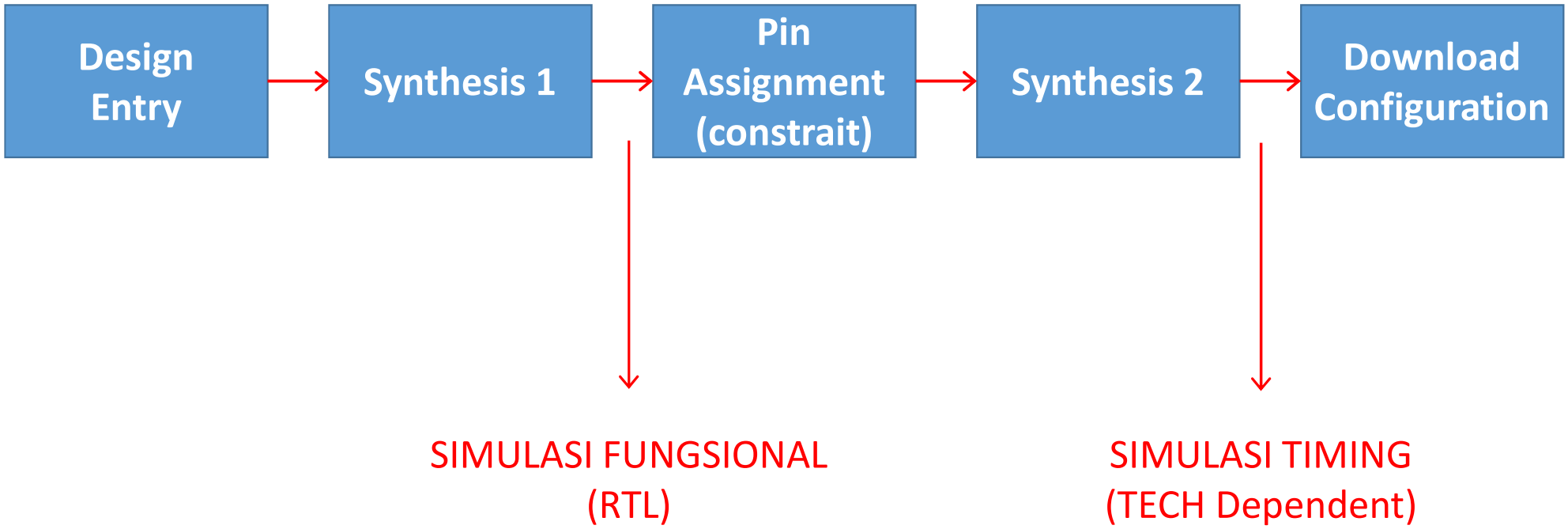



```
Signal C : std_logic_vector(N-1 downto 0);

begin

Adder_0 : FULLADDER
port map(
                        C_in => '0',
                        X  => X(0),
                        Y  => Y(0),
                        Sum => Sum(0),
                        C_out => C(0)
);

Adders : for i in  1 to N-1 generate
          Adder: FULLADDER
          port map(
                        C_in => C(i-1),
                        X  => X(i),
                        Y  => Y(i),
                        Sum => Sum(i),
                        C_out => C(i));
end generate;

C_out <= C(N-1);

end architecture structural;
```

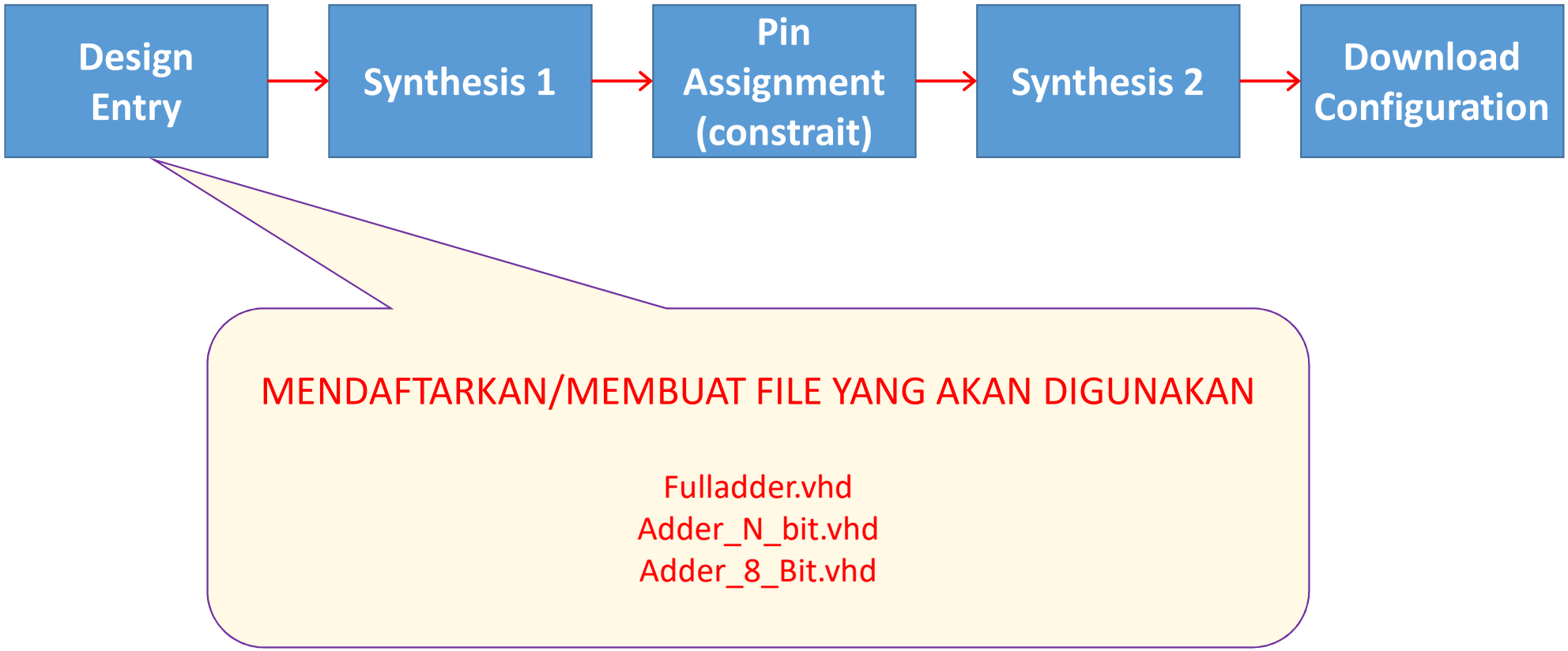
CONTOH PENGGUNAAN

```
library IEEE;
use IEEE.std_logic_1164.all;
entity Adder_16_Bit is
        Port(C_In : in std_logic;
             A      : in std_logic_vector(15 downto 0);
             B      : in std_logic_vector(15 downto 0);
             Sum  : out std_logic_vector(15 downto 0);
             C_out: out std_logic);
end entity;
architecture asep of Adder_16_bit is
component Adder_N_Bit is
        Generic(N : integer := 8);
        Port(       C_In : in std_logic;
                    X : in std_logic_vector(N-1 downto 0);
                    Y       : in std_logic_vector(N-1 downto 0);
                    Sum    : out std_logic_vector(N-1 downto 0);
                    C_out : out std_logic);
  end component;
Begin
  myadder : Adder_N_Bit
            Generic map(N:= 16)
            Port map(
                    C_In => C_in,
                    A => A,
                    B =>B,
                    Sum  =>Sum,
                    C_out -> C_out);
  end architecture asep;
```

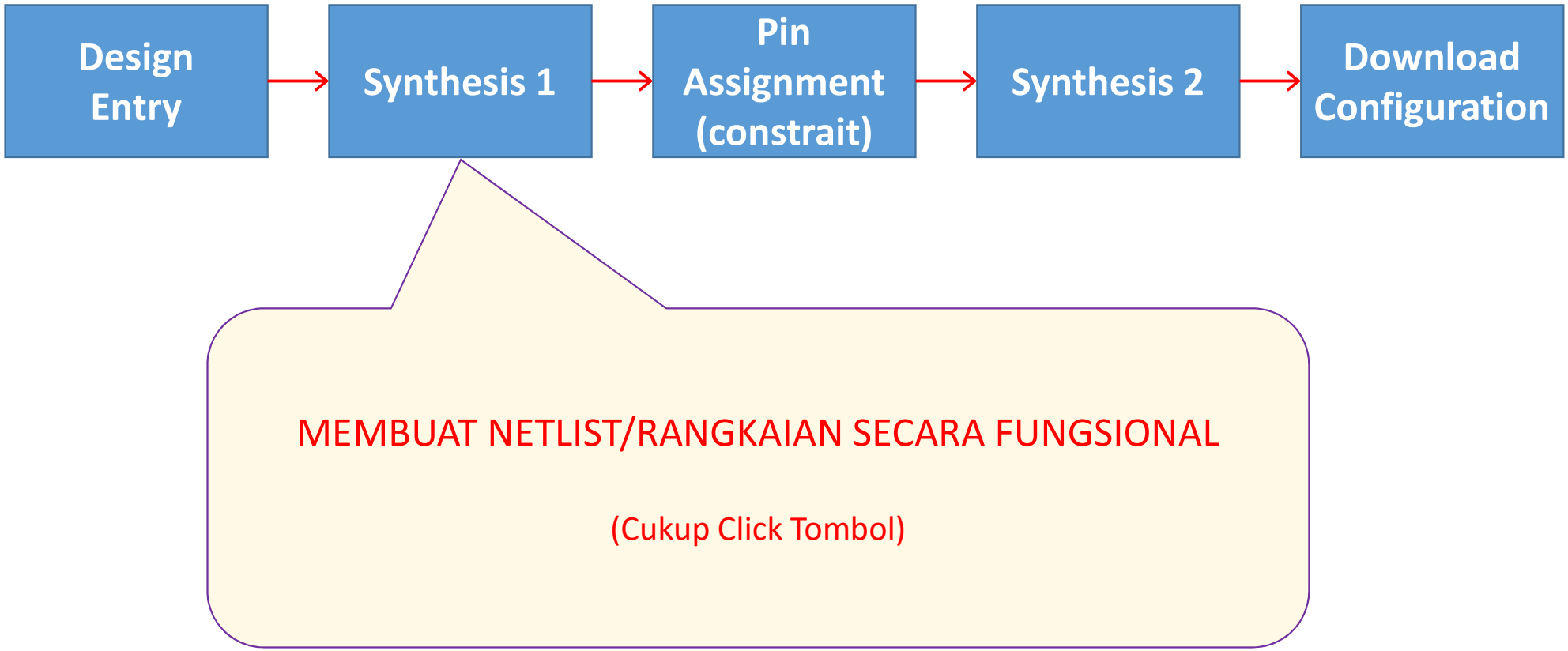
IMPLEMENTASI DENGAN FPGA



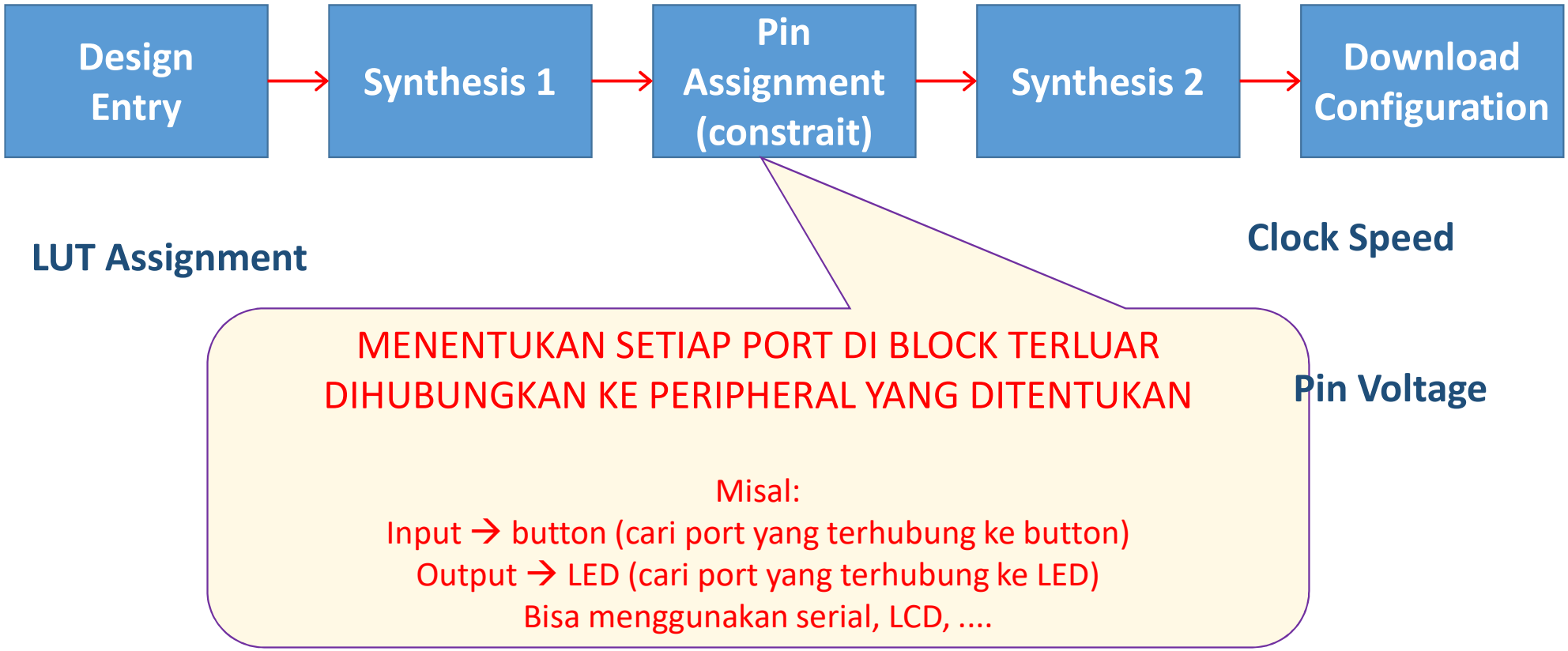
IMPLEMENTASI DENGAN FPGA



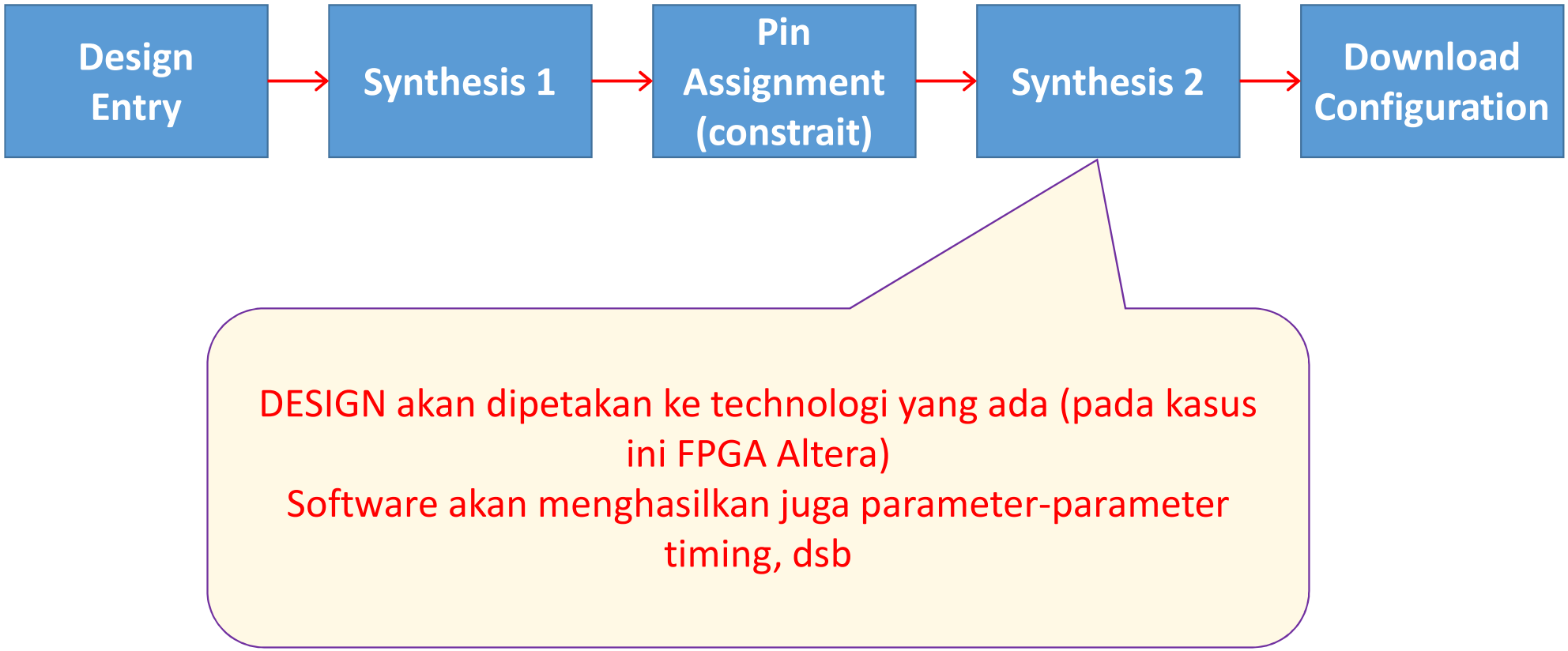
IMPLEMENTASI DENGAN FPGA



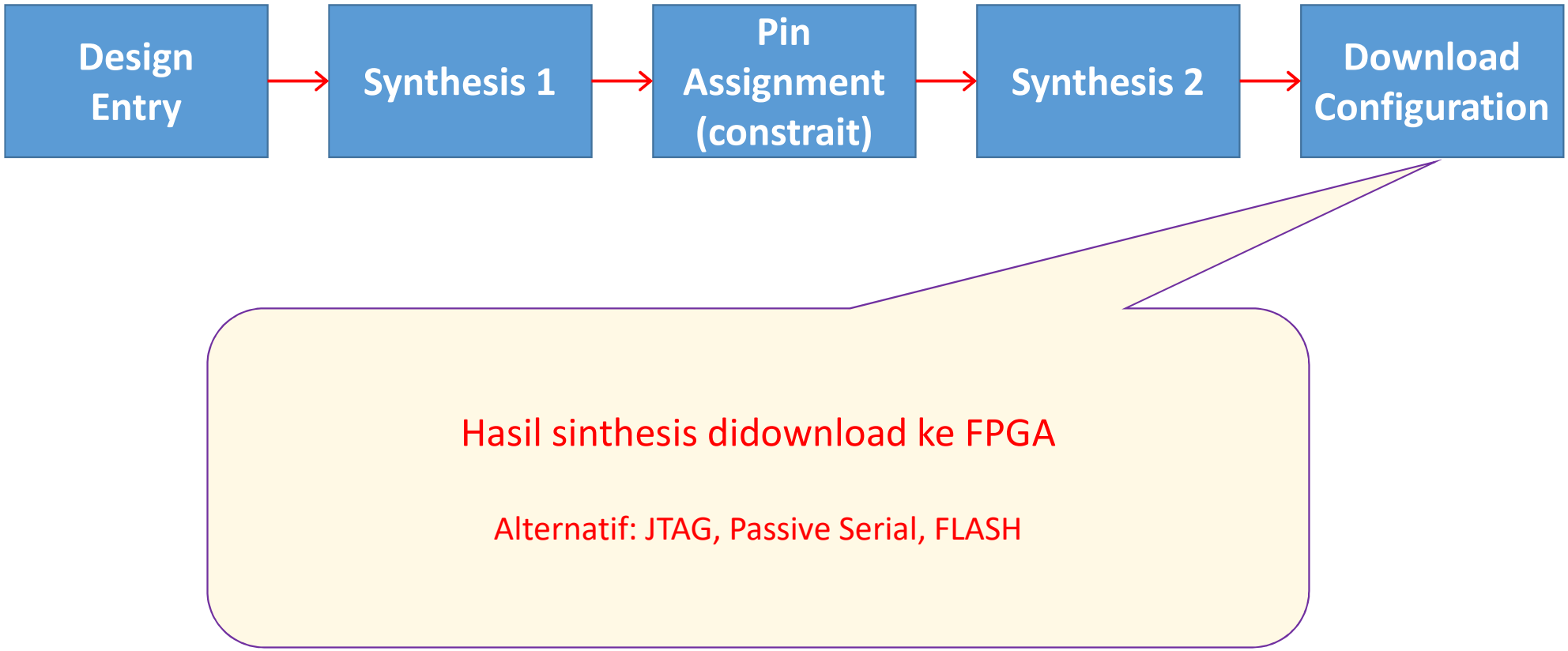
IMPLEMENTASI DENGAN FPGA



IMPLEMENTASI DENGAN FPGA



IMPLEMENTASI DENGAN FPGA



Implementasi System Digital dengan VHDL

bagian 2 – rangkaian sekuensial

Arif Sasongko

Bahan

- Review Rangkaian Sekuensial
- Flip-Flop
- Process
- FSM
- Contoh Rangkaian Sekuensial
- Contoh Rangkaian Sekuensial – Higher abstraction
- Contoh – Eksperimen !

Bahan

- **Review Rangkaian Sekuensial**
- Flip-Flop
- Process
- FSM
- Contoh Rangkaian Sekuensial
- Contoh Rangkaian Sekuensial – Higher abstraction

Review Rangkaian Sekuensial

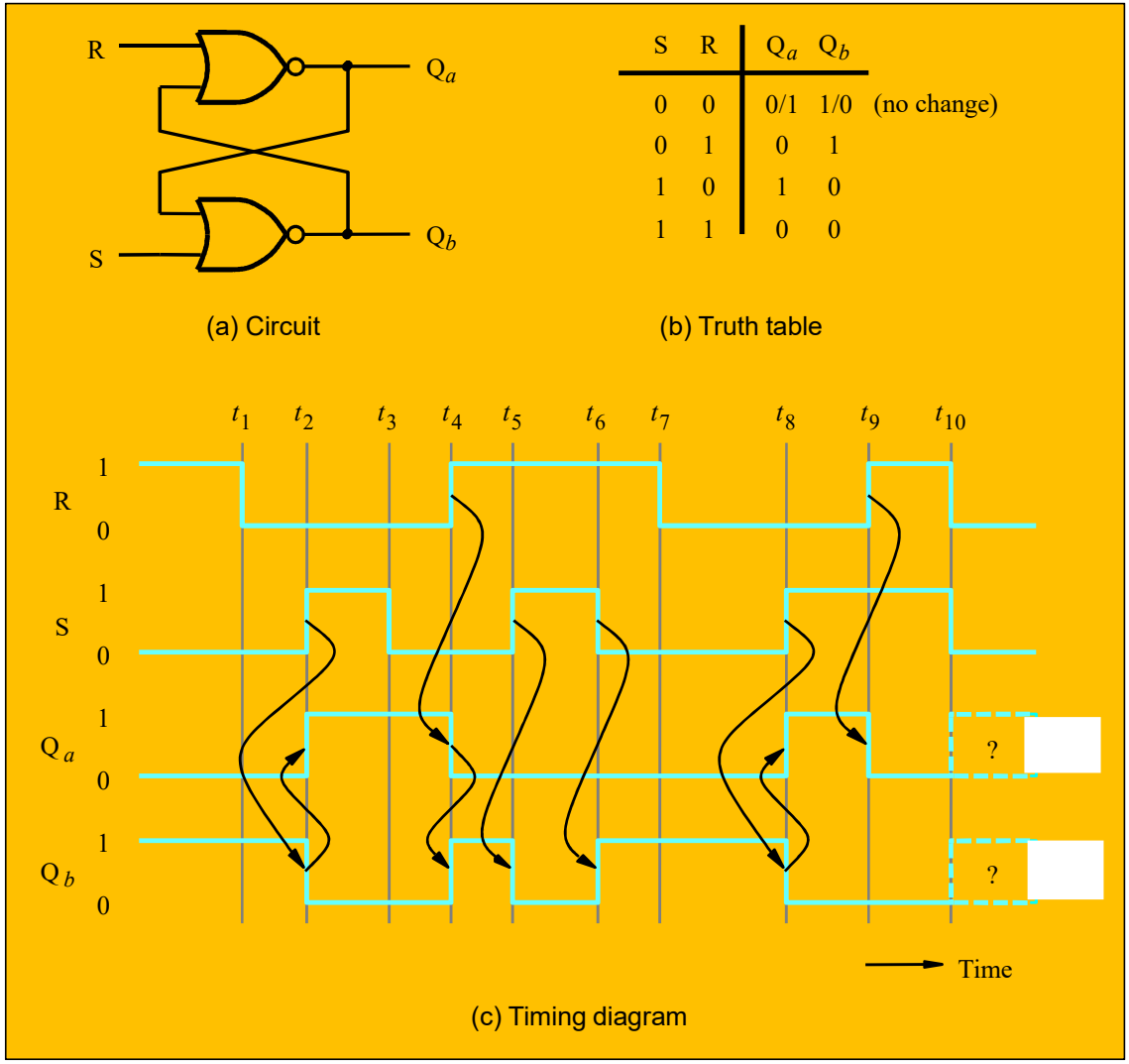
- Output Rangkaian Sekuensial tergantung input saat ini dan input sebelumnya
- Contoh:
  - Kalkulator, Vending machine
  - Processor
  - Hampir semua rangkaian digital mengandung bagian sekuensial
- Konten
  - Ada bagian rangkaian yang menyimpan informasi
  - Flip-flop, register, memory
  - FSM

Bahan

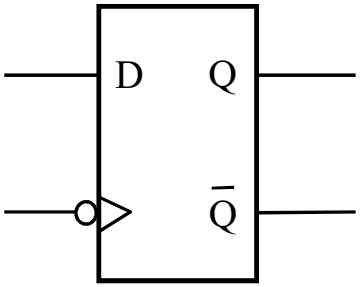
- Review Rangkaian Sekuensial
- **Flip-Flop**
- Process
- FSM
- Contoh Rangkaian Sekuensial
- Contoh Rangkaian Sekuensial – Higher abstraction

Flip-Flop

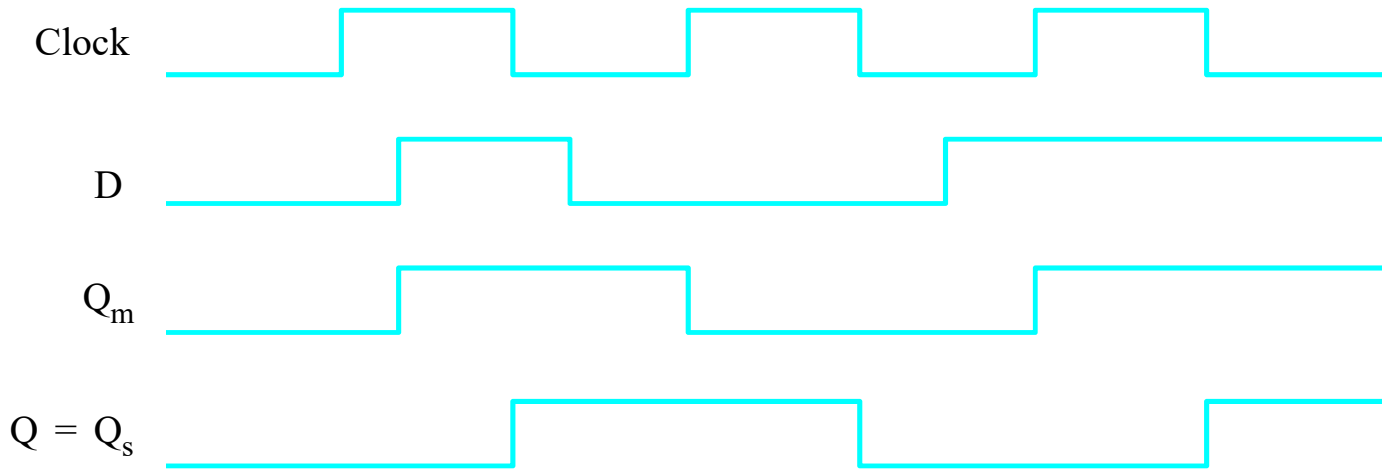
- Bentuk paling sederhana latch (tanpa clock)
- Ada/memanfaatkan feedback
- Macam/jenis:
  - D-Flip flop
  - T-Flip flop
  - SR-Flip flop
  - JK-Flip flop



Flip flop

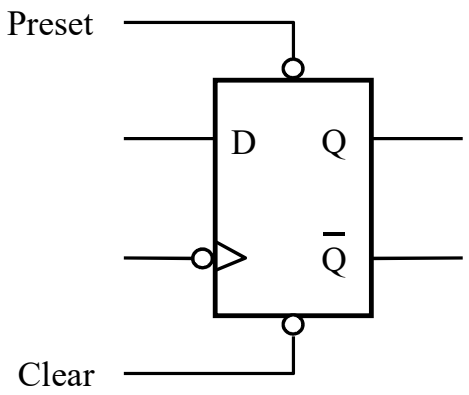


(a) Graphical symbol



(b) Timing diagram

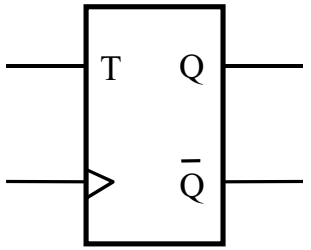
Flip-flop



(a) D Flip flop with Preset and Clear

T	Q(t + 1)
0	Q(t)
1	$\overline{Q}(t)$

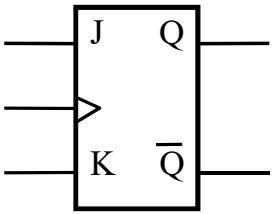
(b) T-FF Truth table



(c) T- FF Graphical symbol

J	K	Q (t + 1)
0	0	Q (t)
0	1	0
1	0	1
1	1	$\overline{Q}$ (t)

(d) JK-FF Truth table



(e) JK-FF Graphical symbol

Bahan

- Review Rangkaian Sekuensial
- Flip-Flop
- **Process**
- FSM
- Contoh Rangkaian Sekuensial
- Contoh Rangkaian Sekuensial – Higher abstraction

Process

- Rangkaian sekuensial membutuhkan process (Process ≠ R Sekuensial)
- Statement dalam process sekuensial
- Antar process sendiri block yang paralel
- Process akan dieksekusi setiap ada perubahan dalam sensitvity list !

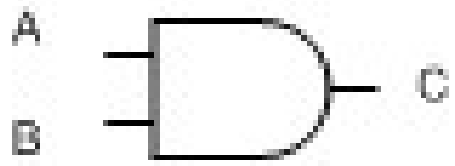
```
process( …. sensitivity list ….)
        declaration
begin
        Sequential Statements
        .
        .
        .
end process;
```

Process: Combinational Circuit (1)

```
Process(A, B)
begin
  if A = '1' then
        C <= B;
  else
        C <=  0;
  end if;
end
```

Equivalent dengan:

```
C <= A and B
```



Process: Combinational Circuit (2)

```
Process(Current_State, input1)
Begin
   If Current_State = idle then
            Next_State <= Current_State;
           If input1 = '1' then
                    Next_State <= fetch;
           End if;
   elsif Current_State = fetch then
            Next_State <= execute;
   elsif Current_State = Execute then
            Next_State <= idle;
   else
            Next State <= idle
   end if;
end
```

HIGHER LEVEL ABSTRACTION !!!

Process: Combinational Circuit (3)

Semua input masuk ke sensitivity list

```
Process(A, B)
begin
  if A = '1' then
        C <= B;
  else
        C <=  0;
  end if;
end
```

```
Process(A)
begin
  if A = '1' then
        C <= B;
  else
        C <=  0;
  end if;
end
```

Process: Combinational Circuit (4)

Output untuk semua kombinasi input terdifinisi

```
Process(A, B)
begin
  if A = '1' then
        C <= B;
  else
        C <=  0;
  end if;
end
```

```
Process(A,B)
begin
  if A = '1' then
        C <= B;
  end if;
end
```

Process: FLIP-FLOP(1)

- Harus ada edge dalam statement
- Input yang ada diluar edge harus ada dalam sensitivity list

Process: FLIP-FLOP(2)

```
Entity DFF is
Port( D, Clk : in std_logic;
        Q : out);
End entity
Architecture behave of DFF is
Begin
  Process(Clk)
  Begin
    If clk = '1' and clk'event then
          Q <= D;
    end if;
  End Process;
End Architecture behave;
```

D Flip Flop !!

Clk Edge

Process: FLIP-FLOP(2)

D flip flop with reset

```
Process(Clk,reset)
Begin
  If reset = '1' then
        Q <= 0;
  else
      If clk = '1' and clk'event then
        Q <= D;
      end if;
    End if;
End Process
```

Process: FLIP-FLOP(3)

T flip flop

```
Process(Clk, T)
Begin
  If clk = ‘1’ and clk’event then
        if  T = ‘1’ then
              Q <= D;
        end if;
  end if;
End Process
```

Process: FLIP-FLOP(4)

Banyak flip-flop/Register (contoh 8bit)

```
Process(Clk,reset)
Begin
  If reset = ‘1’ then
        Q <= “00000000”;
  else
      If clk = ‘1’ and clk’event then
        Q <= D;
      end if;
    End if;
End Process
```

Bahan

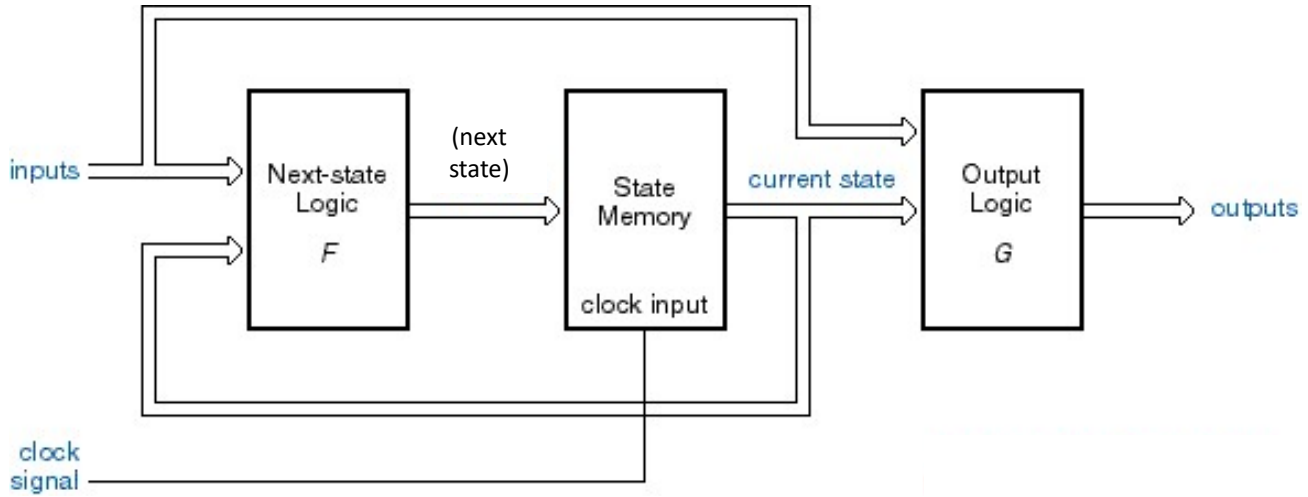
- Review Rangkaian Sekuensial
- Flip-Flop
- Process
- **FSM**
- Contoh Rangkaian Sekuensial
- Contoh Rangkaian Sekuensial – Higher abstraction
- Contoh – Eksperimen !

FSM:State Machine

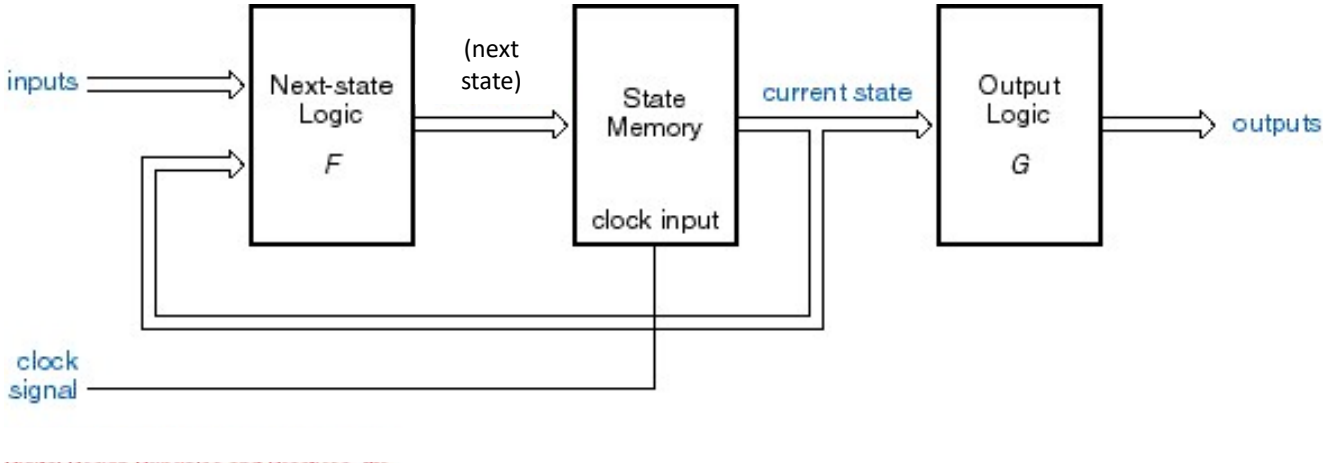
- Rangkaian digital sekuensial biasanya dimodelkan sebagai Finite State Machine (FSM)
- State mewakili (mengkodifikasi) seluruh input sebelumnya
- JADI output tergantung dari dua hal output dan state
- Ada dua model:
  - Meally model
  - Moore model

FSM: **Clocked synchronous state machines:**

Mealy:



Moore:



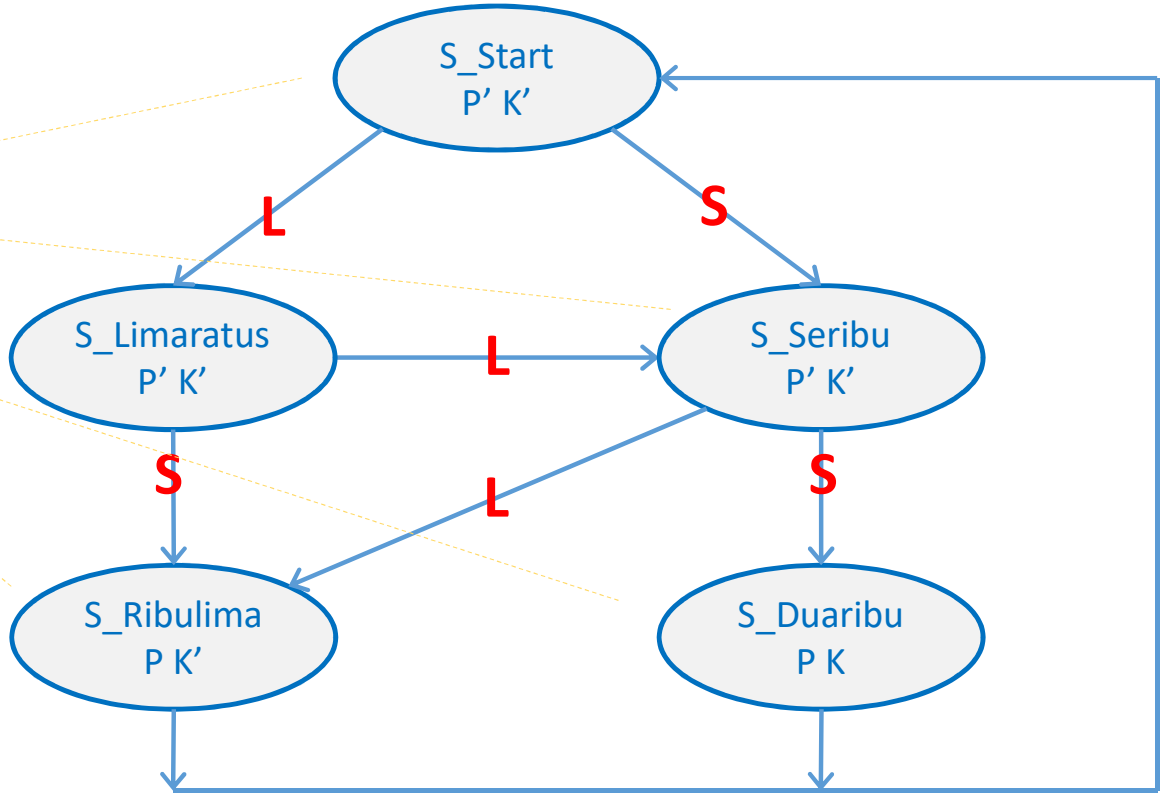
FSM:Contoh

- Sebuah vending machine (mesin penjual otomatis) yang menjual permen seharga 1500 rupiah
- Mesin ini menerima dua jenis input:
  - Uang logam lima ratus rupiah (diwakili variable/literal L)
  - Uang logam Seribu rupiah (diwakili variable/literal S)
- Mesin ini mengeluarkan dua jenis output:
  - Permen (diwakili variable/literal P)
  - Kembalian limaratus rupiah (diwakili variable/literal K)
- Mesin ini menerima input satu per satu (mekaniknya membatasi tidak bisa menerima uang lima ratus dan seribu bersamaan
- State dalam sistem ini akan mewakili kredit/jumlah uang yang sudah dimasukan

FSM:State diagram

ADA 5 STATE

PERLU 3 STATE VARIABLE !!



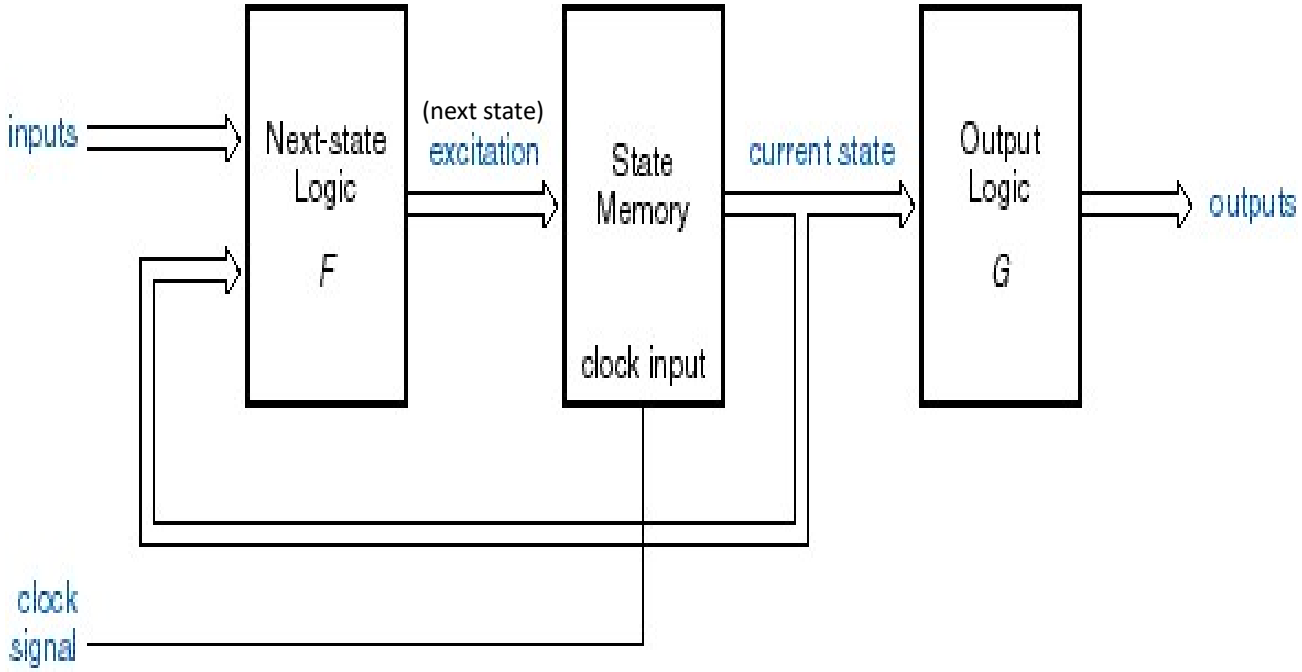
FSM:State Assigment

Misal:

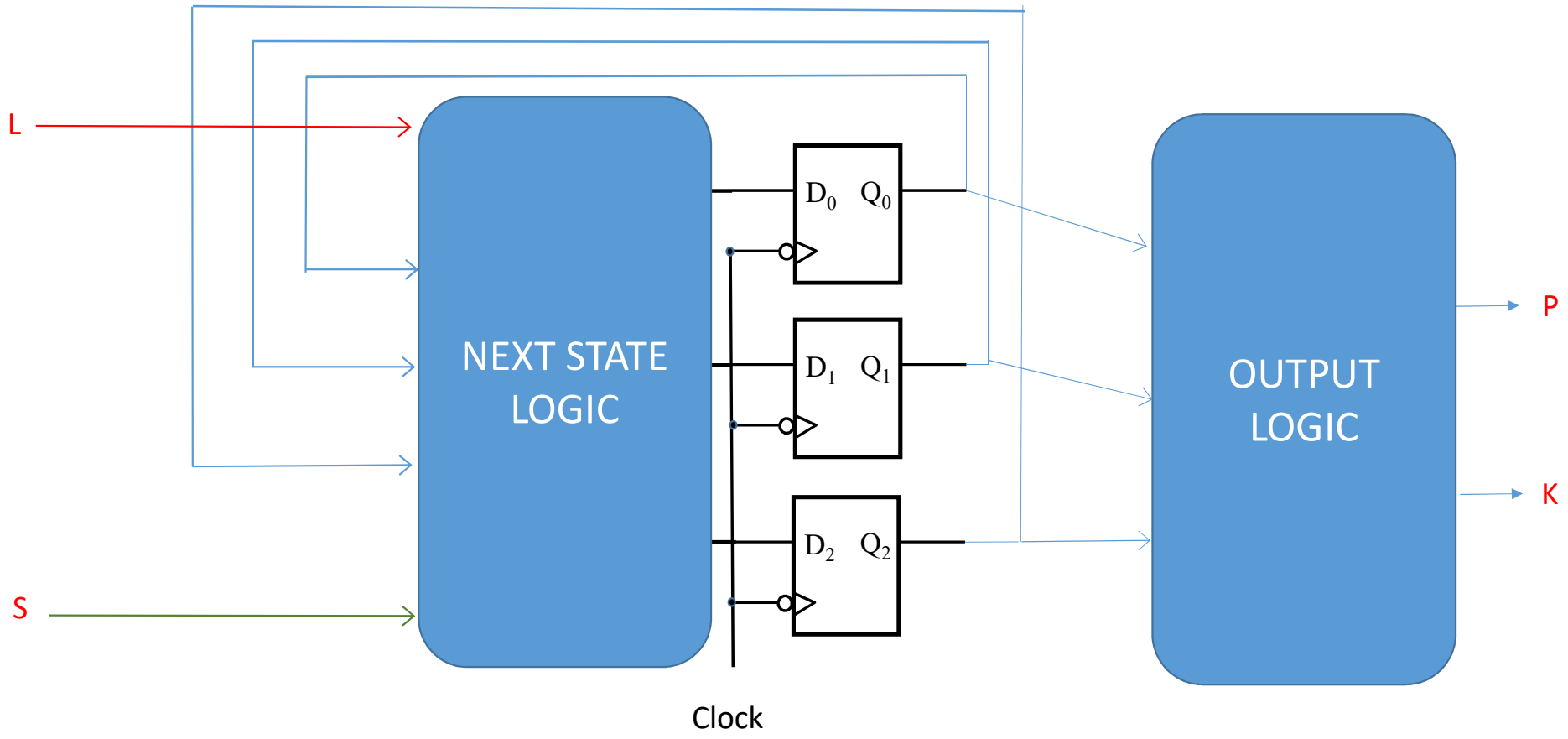
State	Q_2	Q_1	Q_0
S_Start	0	0	0
S_Limaratus	0	0	1
S_Seribu	0	1	0
S_Seribulima	0	1	1
S_Duaribu	1	0	0

Clocked synchronous state machines:

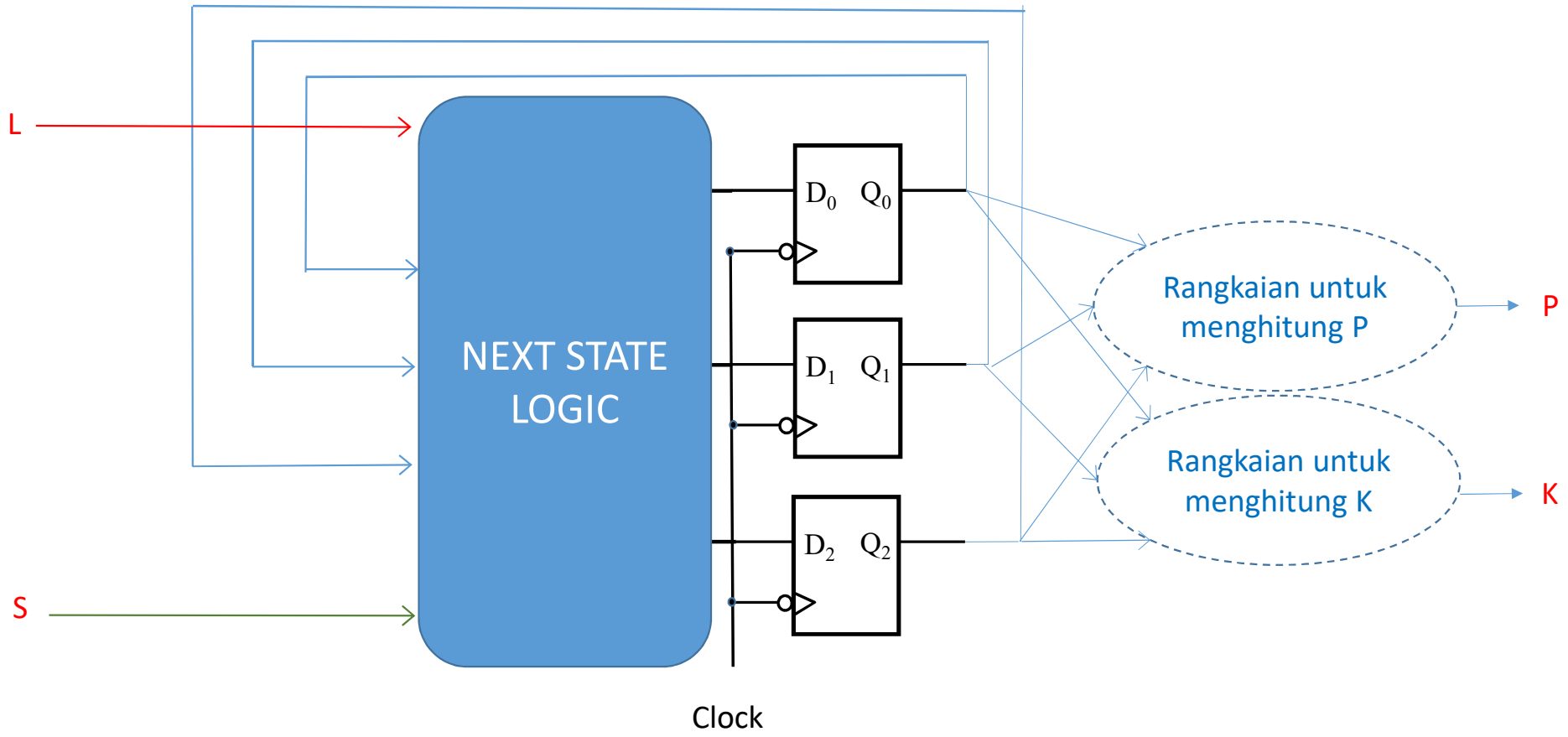
Moore:



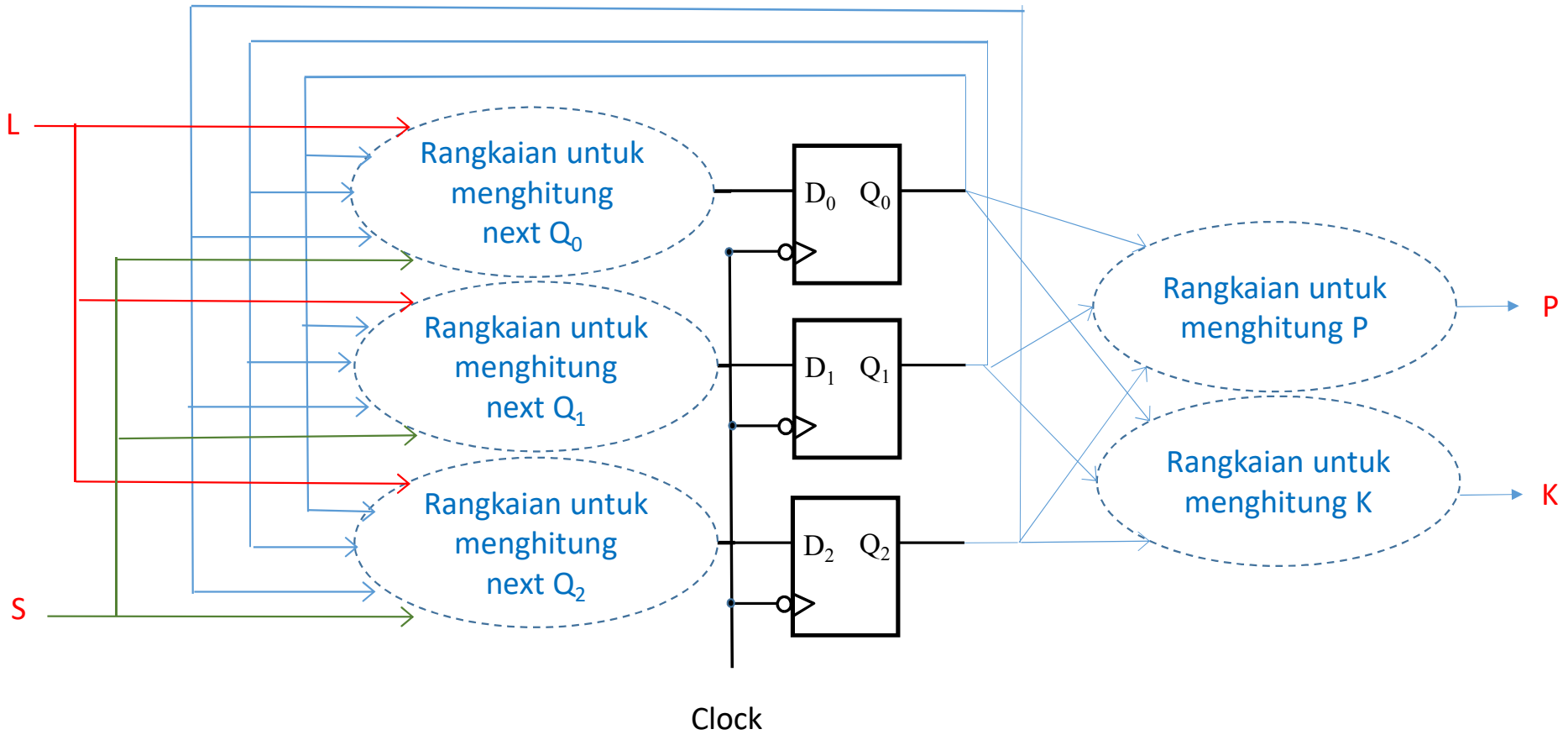
FSM:State Assigment



FSM:State Assigment



FSM:State Assigment



FSM:State Assignment

State	$Q_2Q_1Q_0$
S_Start	0 0 0
S_Limaratus	0 0 1
S_Seribu	0 1 0
S_Seribulima	0 1 1
S_Duaribu	1 0 0

Misal:

State	Next State				
$Q_2Q_1Q_0$	L'S'	L'S	LS'	LS	P K
0 0 0	0 0 0	0 1 0	0 0 1	x x x	0 0
0 0 1	0 0 1	0 1 1	0 1 0	x x x	0 0
0 1 0	0 1 0	1 0 0	0 0 1	x x x	0 0
0 1 1	0 0 0	0 0 0	0 0 0	x x x	1 0
1 0 0	0 0 0	0 0 0	0 0 0	x x x	1 1
1 0 1	x x x	x x x	x x x	x x x	x x
1 1 0	x x x	x x x	x x x	x x x	x x
1 1 1	x x x	x x x	x x x	x x x	x x

FSM:Next State and Output Logic

Misal:

State	Next State ($D_2D_1D_0$)				
$Q_2Q_1Q_0$	L'S'	L'S	LS'	LS	P K
0 0 0	0 0 0	0 1 0	0 0 1	x x x	0 0
0 0 1	0 0 1	0 1 1	0 1 0	x x x	0 0
0 1 0	0 1 0	1 0 0	0 0 1	x x x	0 0
0 1 1	0 0 0	0 0 0	0 0 0	x x x	1 0
1 0 0	0 0 0	0 0 0	0 0 0	x x x	1 1
1 0 1	x x x	x x x	x x x	x x x	x x
1 1 0	x x x	x x x	x x x	x x x	x x
1 1 1	x x x	x x x	x x x	x x x	x x

State	$Q_2Q_1Q_0$
S_Start	0 0 0
S_Limaratus	0 0 1
S_Seribu	0 1 0
S_Seribulima	0 1 1
S_Duaribu	1 0 0

FSM:Next State and Output Logic

Next State Logic

$D_0 = Q_1Q_0L' + Q_2LQ_0$

$D_1 = Q_2Q_1S + Q_1'Q_0L + Q_1Q_0'L'S'$

$D_2 = Q_2'Q_1Q_0'L'S$

Output Logic

$P = Q_2Q_1'Q_0' + Q_1Q_0$

$K = Q_2Q_1'Q_0'$

```
Architecture Gatelevel of VendingMachine is
        signal Q,D   : std_logic_vector(2 downto 0);
begin
-- State Register: advance to the next state
process (CLOCK, reset)
begin
        if reset = '1' then
                Q <= "000";
        elsif (rising_edge(CLOCK)) then
                Q <= D;
        end if;
end process;
  -- Next State Logic
    D(0) <= (Q(1) and Q(0) and not L) or (Q(2) and L and Q(0));
    D(1) <= (Q(2) and Q(1) and S) or (not Q(0)and Q(0) and L) or Q(1)and not Q(0) and not L and not S)
    D(2) <= not Q(2) and Q(1) and not Q(0)and not L and S;
  --Output LOGIC
    P <= (Q(2) and not Q(1) and not Q(0))+ (Q(1) and Q(0));
    K <= Q(2) not Q(1) and not Q(0);
end Behavioural;
```

```
Architecture Behavioural of Control_FSM is
type my_state_type is (S_Start,S_Limaratus,S_Seribu,
    S_Seribulima, S_Duaribu);
signal current_state,next_state   : my_state_type;

begin
-- State Register: advance to the next state
process (CLOCK, reset)
begin
        if reset = '1' then
                current_state <= S_Start;
        elsif (rising_edge(CLOCK)) then
                current_state <= next_state;
        end if;
end process;
-- Next State Logic
process (current_state, START)
begin
        next_state <= current_state;
                case current_state is
when S_Start =>
        if L = '1' then
                next_state <= S_Limaratus;
        end if;
        if S = '1' then
                next_state <= S_Seribu;
        end if;
when S_Limaratus =>
        if L = '1' then
                next_state <= S_Seribu;
        end if;
        if S = '1' then
                next_state <= S_Seribulima;
        end if;
when S_Seribu =>
        if L = '1' then
                next_state <= S_Seribulima;
        end if;
        if S = '1' then
                next_state <= S_Duaribu;
        end if;
```

```
            when s_Seribulima =>
                        next_state <= s_Start;
            when s_Duaribu =>
                        next_state <= s_Start;
            end case;
end process;

--OUTPUT LOGIC
K <= '1' when current_state = S_Duaribu ELSE
    '0';
P <= '1' when current_state = S_Duaribu ELSE
    '1' when current_state = S_Seribulima ELSE
    '0';

end Behavioural;
```

FSM:Cara alternatif: RTL level

Next State Logic

$D_0 = Q_1Q_0L' + Q_2LQ_0$

$D_1 = Q_2Q_1S + Q_1'Q_0L + Q_1Q_0'L'S'$

$D_2 = Q_2'Q_1Q_0'L'S$

Output Logic

$P = Q_2Q_1'Q_0 ' + Q_1Q_0$

$K = Q_2Q_1'Q_0 '$

FLIP-FLOP(5)

Register pada Mealy/Moore machine (State memory)

```
Process(Clk,reset)
Begin
  If reset = '1' then
        Current_State  <= idle;
  else
      If clk = '1' and clk'event then
        Current_State<= Next State;
      end if;
    End if;
End Process
```

NEXT STATE LOGIC

```
Process(current_state)
Begin
  case current_state is
        when idle =>
              if start_s = '1' then
                    next_state <= count;
              end if;
        when idle =>
              if time_up= '1' then
                    next_state <= waiting;
              end if;
        when idle =>
              if start_s = '0' then
                    next_state <= idle;
              end if;
End Process
```

State Memory

Register pada Mealy/Moore machine

```
Process(Clk,reset)
Begin
  If reset = '1' then
        current_state  <= idle;
  else
      If clk = '1' and clk'event then
        current_state<= next_state;
      end if;
    End if;
End Process
```

Output Logic

```
Process(current_state,start_s,time_up)
Begin
  case current_state is
        when idle =>
              Timer <= '0';
              Count_Enable <= '0';
              …
              …
        when Count =>
              Timer <= '1';
                                    …
              …
        when idle =>
              …
        end case;
End Process
```

Simulasi menggunakan modelsim

- Buat project (tidak perlu kalau membuka langsung dari Quartus)
- Compile
- Pilih entity yang akan disimulasi
- Pilih signal yang akan diamati
- Beri masukan ke signal-signal input
- Running simulasi
- Amati signal

Simulasi modelsim: compile

- Ada dua tab: project dan library → pilih project untuk menambah source file dan mengcompile
- Untuk menambahkan source file: project → add to project → new file
- Untuk menngcompile: pilih file yang ingin dicompile click kanan pilih compile
- Sebaiknya compile dimulai dari file yang terkecil/terdalam

Simulasi modelsim: memilih entity untuk disimulasi

- Ada dua tab: project dan library → pilih library untuk menambah source file dan mengcompile
- Untuk memilih, buka folder work
- Pilih entity yang ingin disimulasikan (click kanan, pilih simulate)

Simulasi modelsim: memilih signal untuk ditampilkan

- Bisa dilakukan setelah ada entity yang disimulasi
- Ada dua cara:
  - Add → add to wave
  - Melalui jendela object, pilih object yang ingin ditampilkan di wave
- Semua input harus ditampilkan di jendela wave

Simulasi Modelsim: Masukan signal-signal input

- Lihat dijendela wave
- Pilih signal, lalu pilih forve (untuk mendrive suatu level tertentu), atau clock (untuk mendrive signal dengan clock/level yang berganti ganti)
- Masukan dulu semua input
- Setelah itu simulasi dapat di-*run* dan berbagai signal dapat diamati

Implementasi System Digital dengan FPGA

bagian 3 – rangkaian komplex

Arif Sasongko

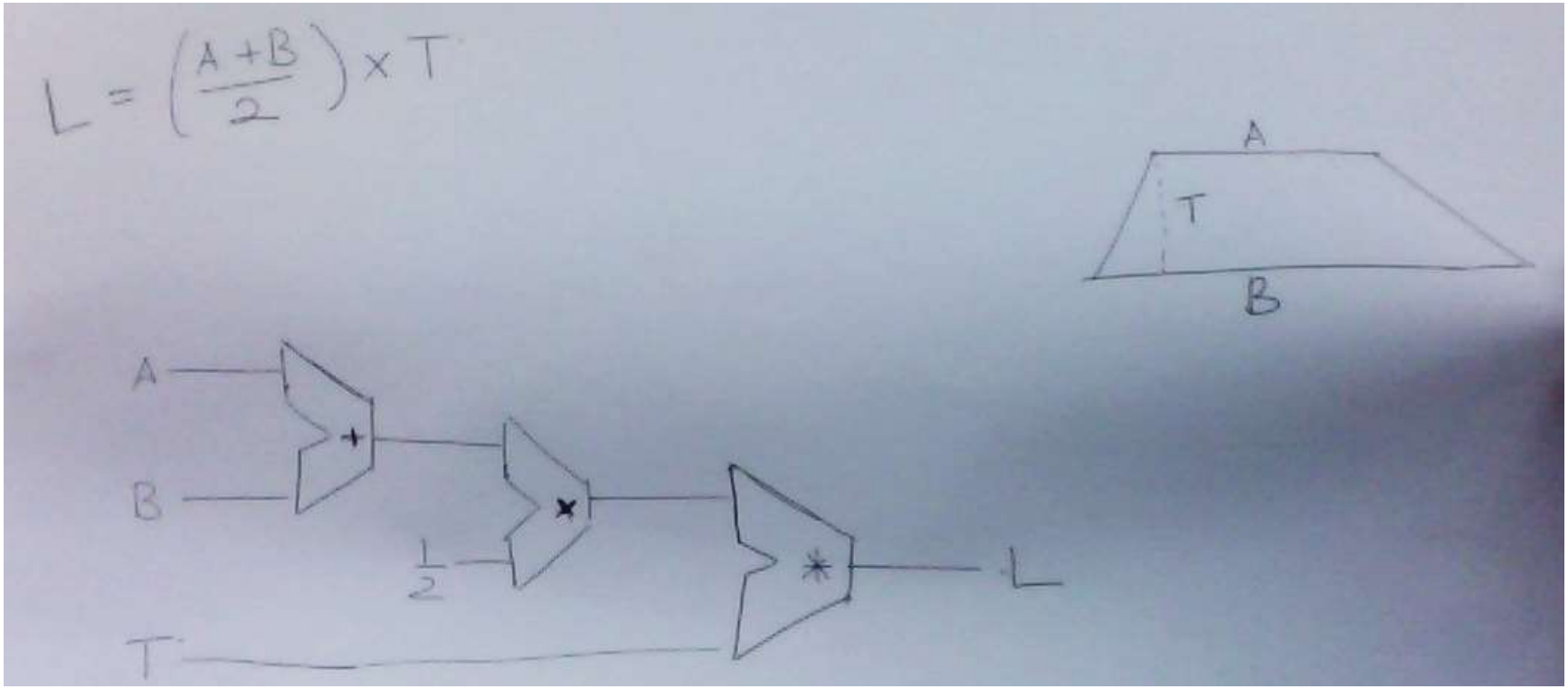
Bahan

- Rangkaian kompleks
- Contoh 1
- Dari Algoritma ke Rangkaian Digital

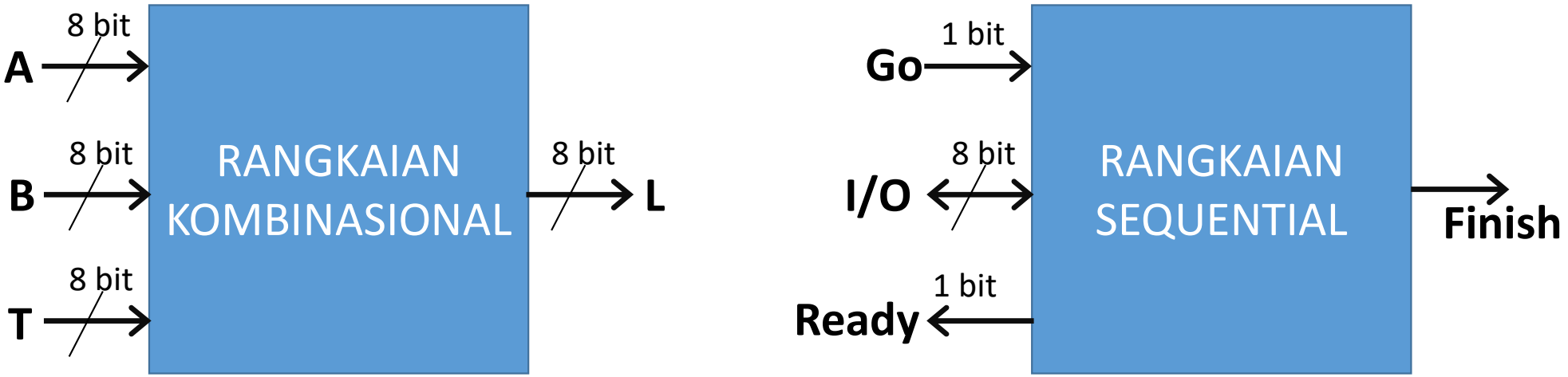
Rangkaian Kompleks

- Biasanya suatu sistem digital yang cukup kompleks diimplementasikan sebagai kombinasi
  - rangkaian sekuensial
  - rangkaian kombinasional
  - Register-register/memory
- Rangkaial kombinasional digunakan untuk blok-blok komputasi/logika
- Rangkaian sekuensial digunakan sebagai kontroler yang mengatur aliran data antara register dan rangkaian blok-blok komputasi/logika
- Register berfungsi sebagai penyimpan sementara data

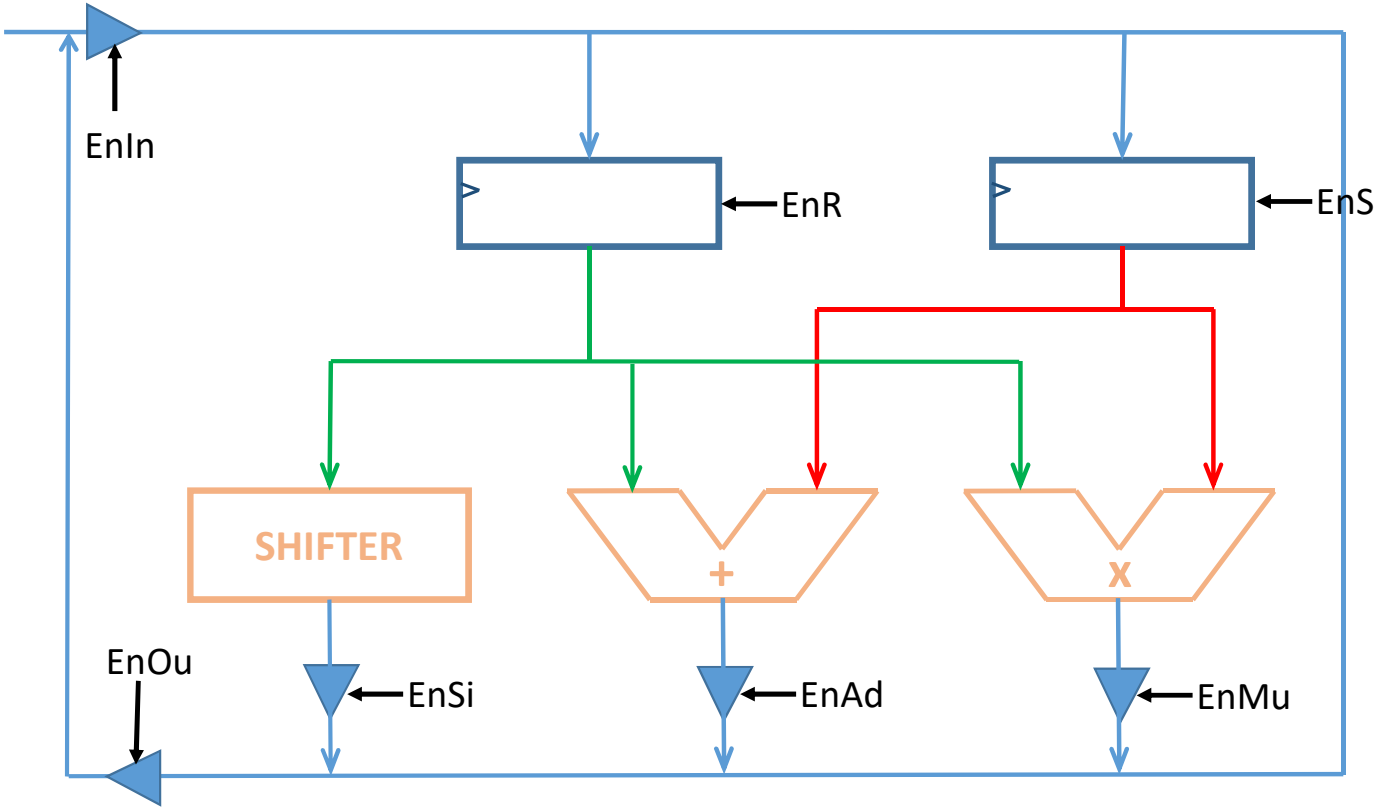
CONTOH: Luas Trapesium

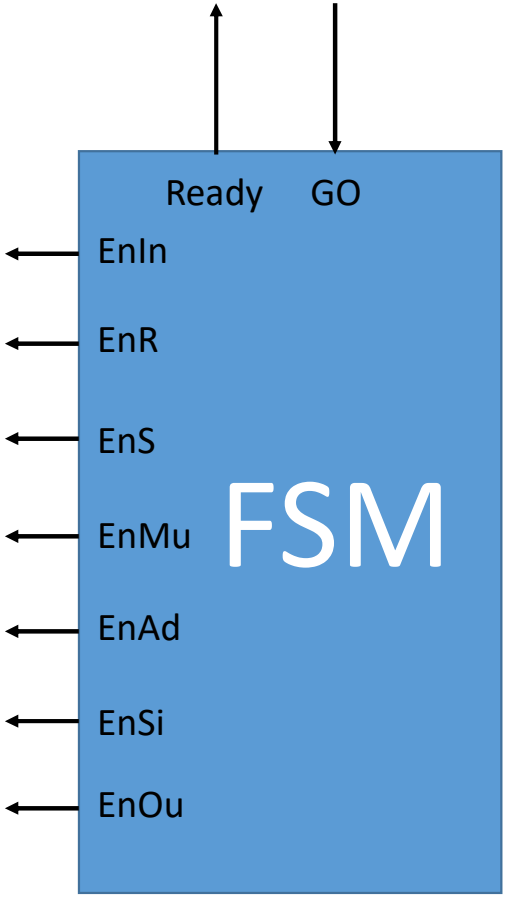


Sequential vs Combinational

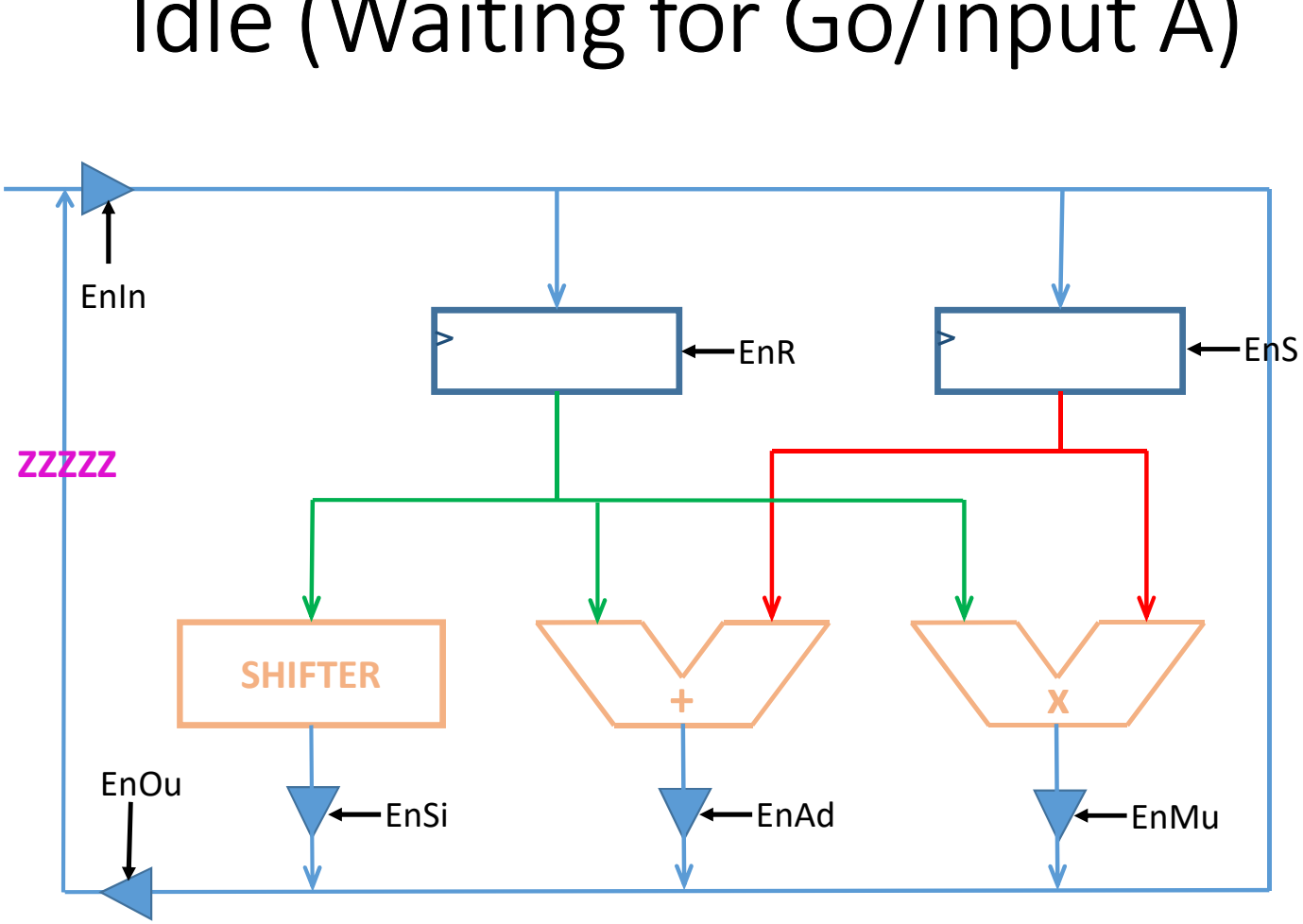


DATAPATH dan FSM

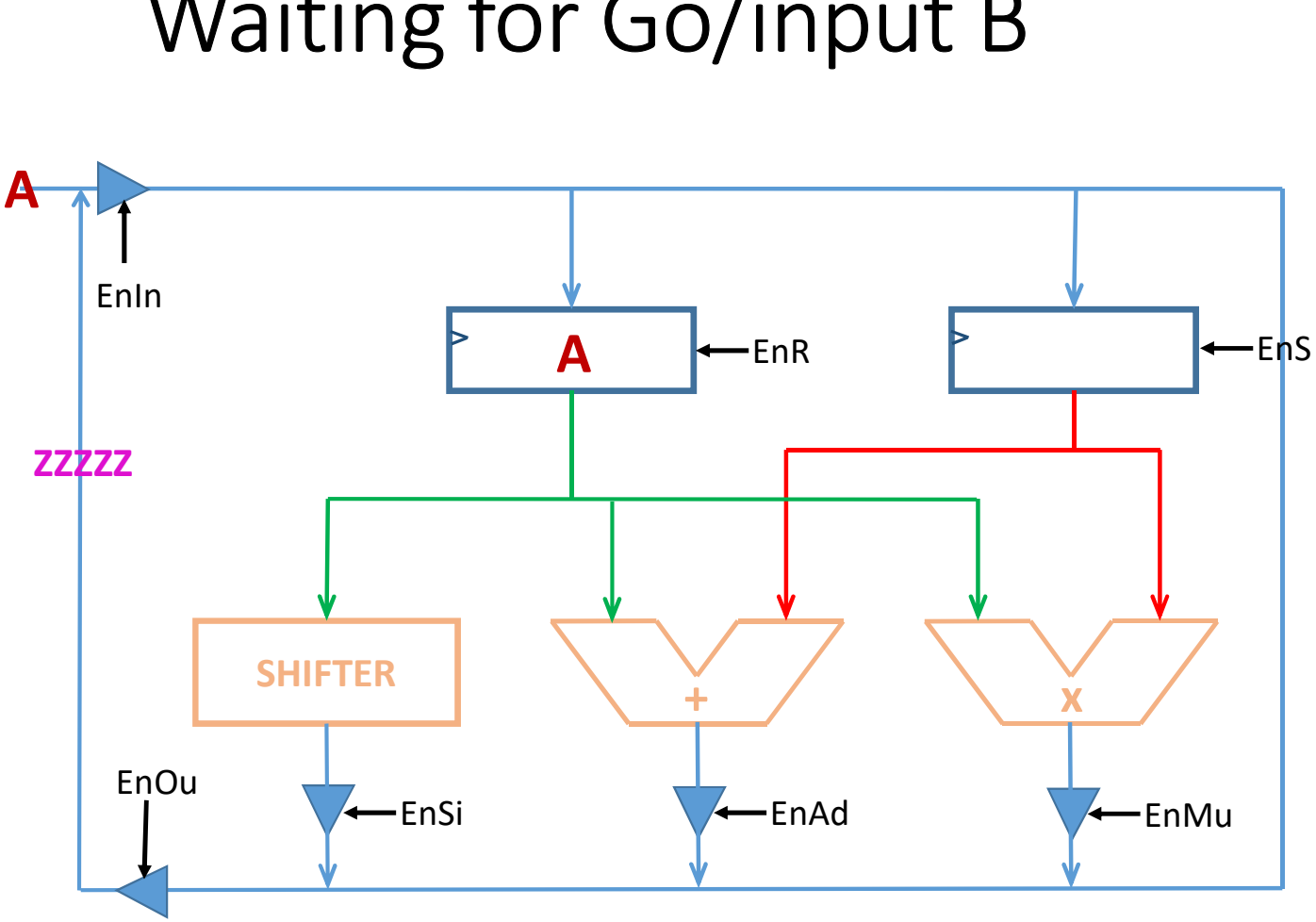




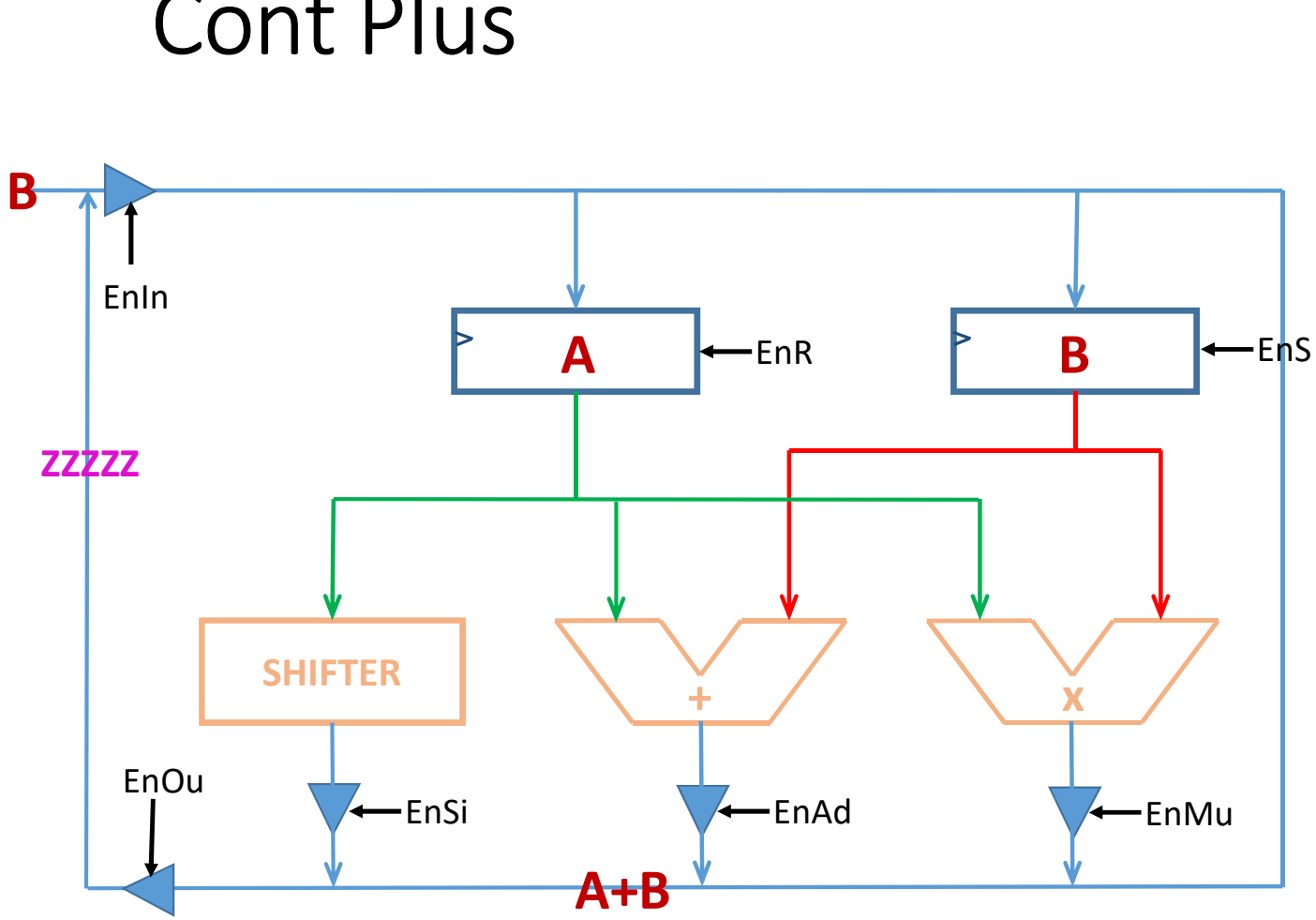
Idle (Waiting for Go/input A)



Waiting for Go/input B

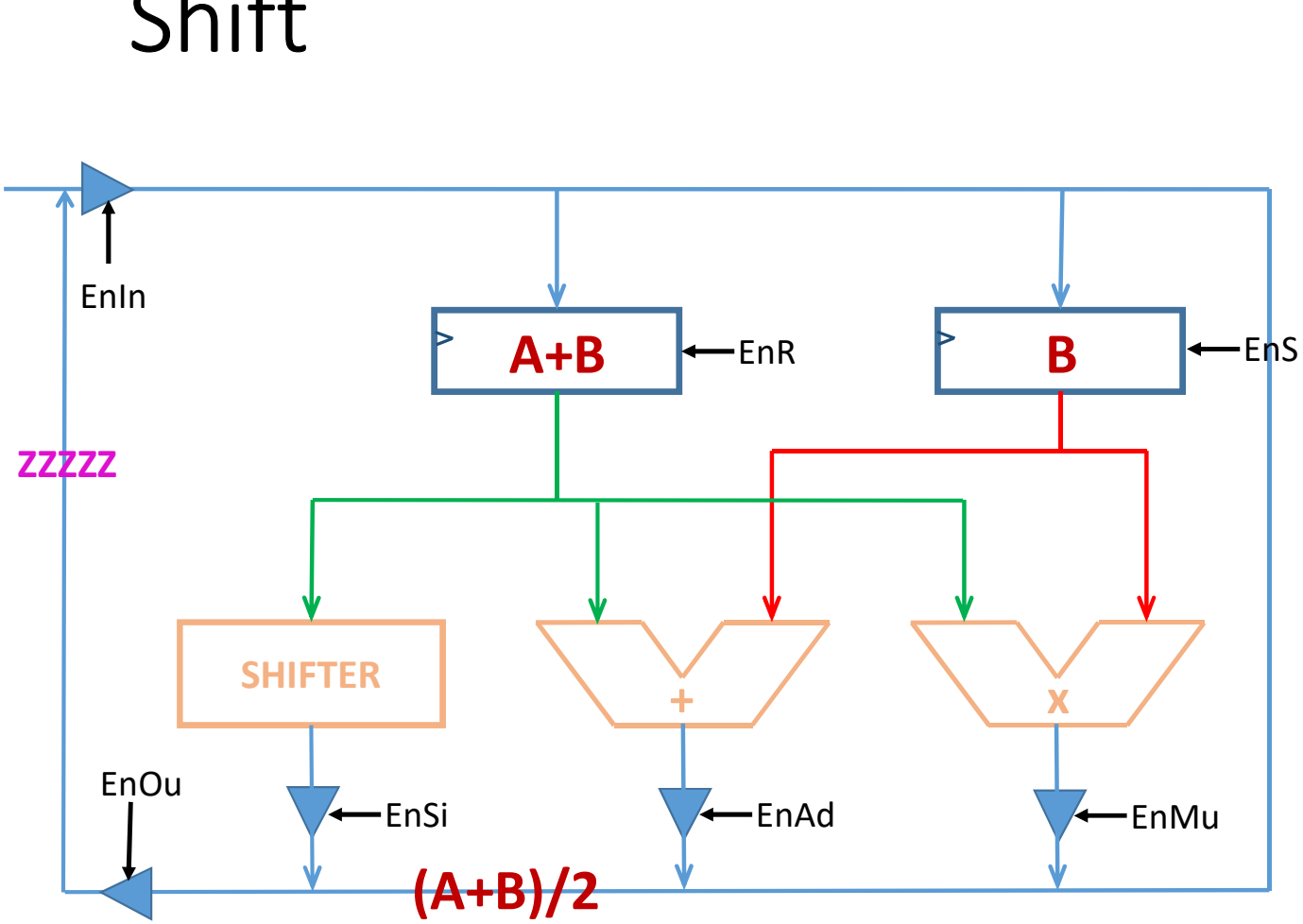


Cont Plus


B
EnIn
A
EnR
B
EnS
ZZZZZ
SHIFTER
+
x
EnOu
EnSi
EnAd
EnMu
A+B

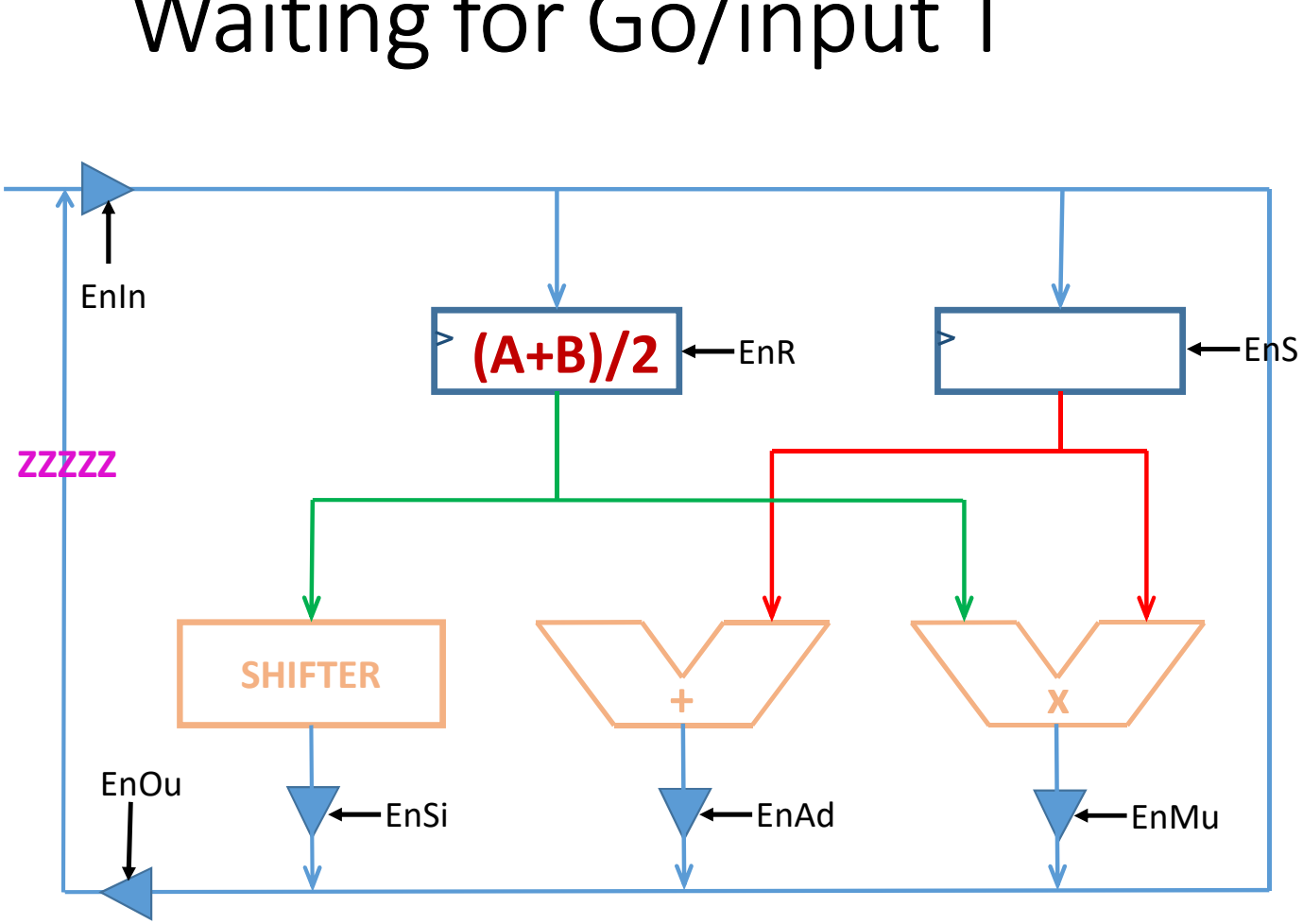

0
1
Ready
GO
1
EnIn
1
EnR
0
EnS
0
EnMu
FSM
1
EnAd
0
EnSi
0
EnOu

Shift

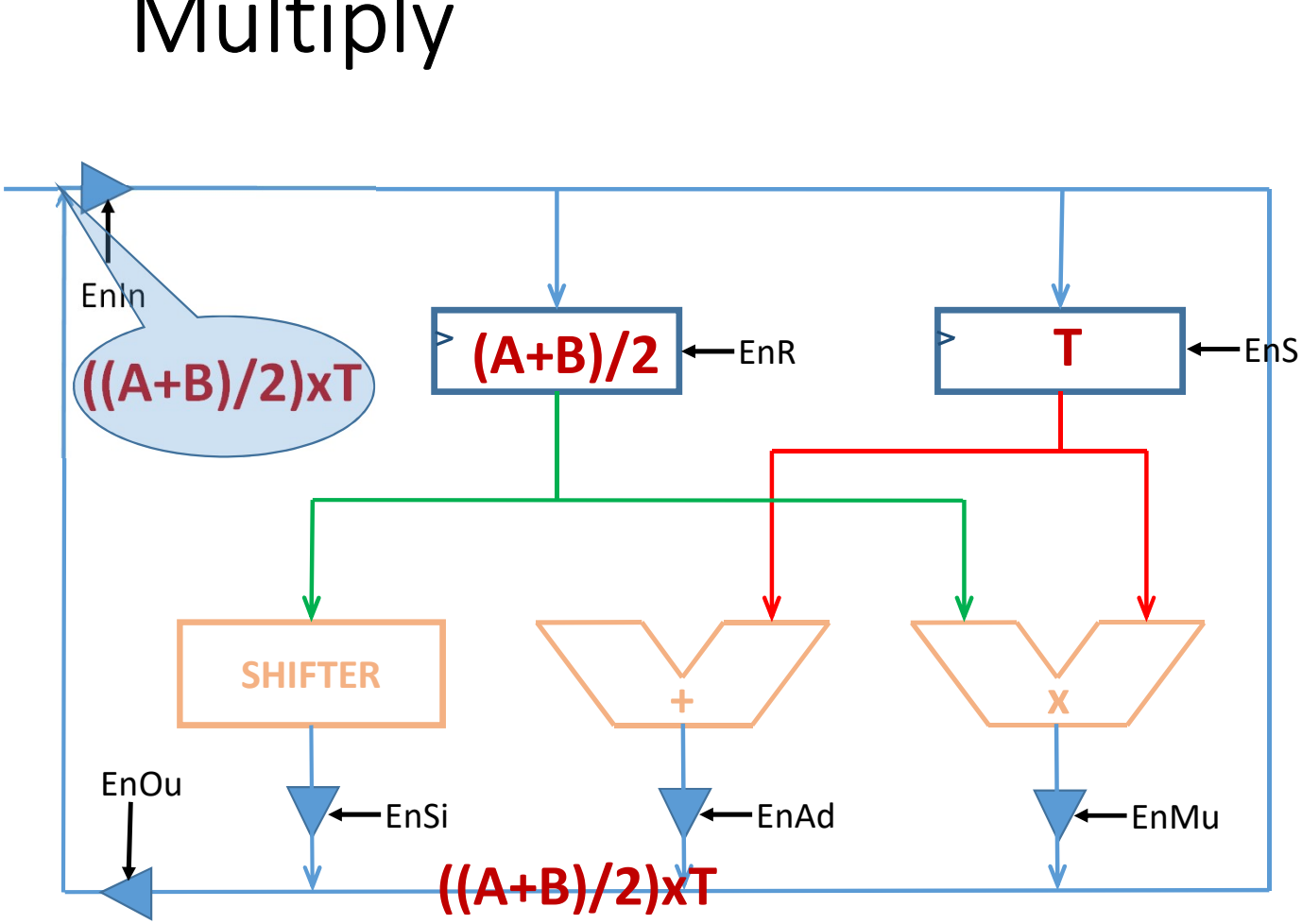

EnIn
A+B
EnR
B
EnS
ZZZZZ
SHIFTER
+
x
EnOu
EnSi
EnAd
EnMu
(A+B)/2


0
Ready
GO
1
EnIn
1
EnR
0
EnS
0
EnMu
FSM
0
EnAd
1
EnSi
0
EnOu

Waiting for Go/input T

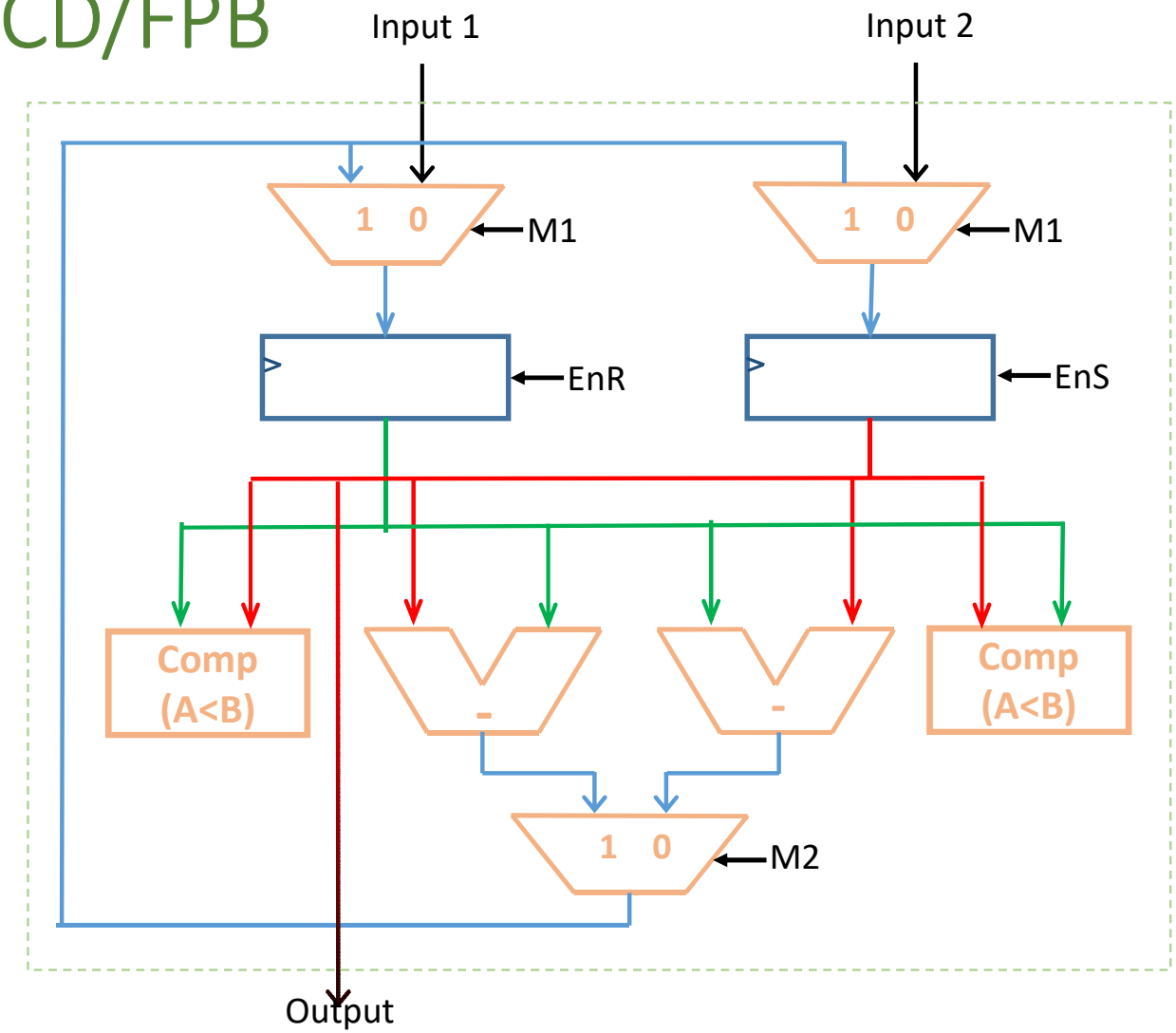


Multiply


EnIn
((A+B)/2)xT
(A+B)/2
EnR
T
EnS
SHIFTER
+
x
EnOu
EnSi
EnAd
EnMu
((A+B)/2)xT


1
1
Ready
GO
0
EnIn
0
EnR
0
EnS
1
EnMu
FSM
0
EnAd
0
EnSi
1
EnOu

GCD/FPB



CONTOH

Step	Register R	Register S	
1	15	21	R<S
2		3	R>S
3	12		R<S
4	9		R<S
5	6		R<S
6	3		