Advanced Digital Circuit Design - Register Transfer Level & Design with ASM

Prof. Dr. Berna Örs Yalçın

Istanbul Technical University
Faculty of Electrical and Electronics Engineering
Department of Electronics and Communication Engineering
siddika.ors@itu.edu.tr

Overview

- Register transfer level (RTL) to model complex digital systems
- Goal: Algorithmic expression of a digital system
- Algorithmic State Machine (ASM) charts
  - to model complex digital systems
  - to map a complex design into hardware

Register Transfer Level

- Designing a complex digital system using state tables becomes difficult as the number of states increases
- Remedy
  - partition the system into modular subsystems
  - each module is designed separately
  - Modules are connected to each other

Register Transfer Level

- Register operations
  - move, copy, shift, count, clear, load, add, subtract
- A digital system is said to be represented at the register transfer level (RTL) when it is specified by the following three components

1. The set of registers
2. Operations performed on the data stored in the registers
3. The control supervises the sequence of operations in the system

The Control

- Control logic
  - Initiates the sequence of operations
  - generates timing (control) signals that sequence the operations in prescribed manner
  - Certain conditions that depend on the previous operations may determine the sequence of future operations
  - The outputs of control logic are binary variables that initiate the various operations in registers
- Transfer statement
  - $R_2 \leftarrow R_1$ ("replacement" operation)
- Conditional transfer statement
  - **if** $(T_1 = 1)$ **then** $R_2 \leftarrow R_1$
  - T_1 is a control signal

Register Transfer Operations

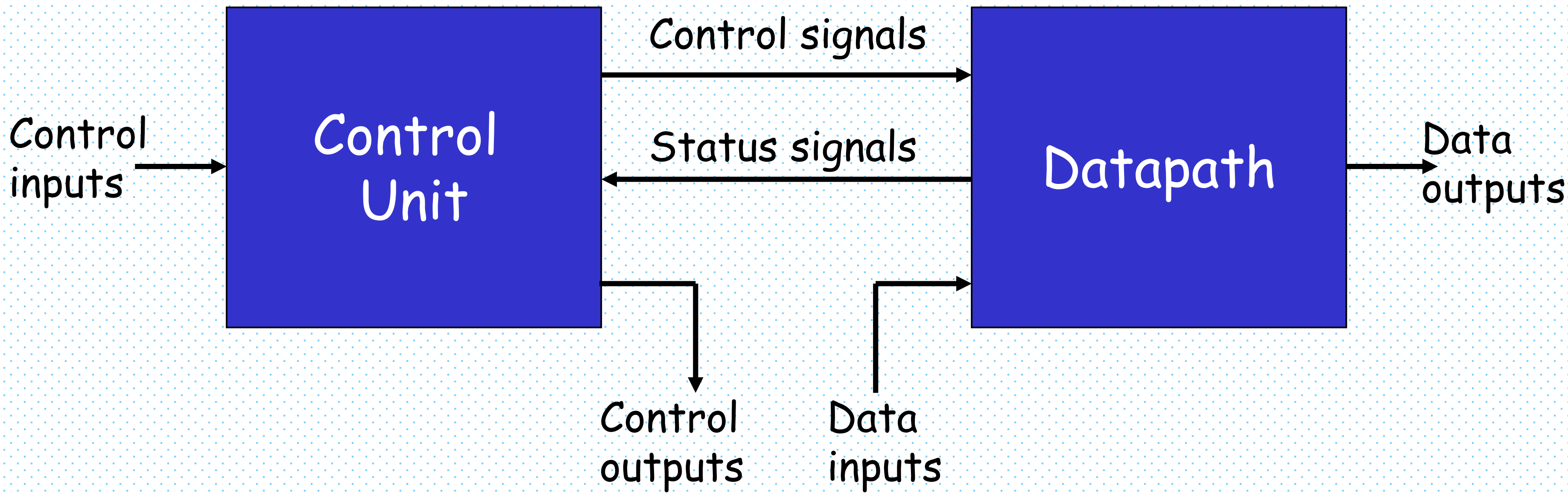
- In register transfer operations clock is not explicitly shown
  - but, transfer is assumed to happen at the clock edge.
  - previous example
    - **if** $(T_1 = 1)$ **then** $R_2 \leftarrow R_1$
    - T_1 may become 1 before the clock edge, but actual transfer happens exactly at the clock edge.
- Examples:
  - **if** $(T_3 = 1)$ **then** $(R_2 \leftarrow R_1,\ R_1 \leftarrow R_2)$
  - $R_1 \leftarrow R_1 + R_2$
  - $R_3 \leftarrow R_3 + 1$
  - $R_4 \leftarrow$ **shr** R_4
  - $R_5 \leftarrow 0$

Types of Register Operations

- Four categories

1. transfer operations
2. arithmetic operations
3. logic operations
4. shift operations

Datapath and Control



- One obvious distinction
  1. design of datapath (data processing path)
  2. design of control circuit

Hardware Algorithm & ASM Chart

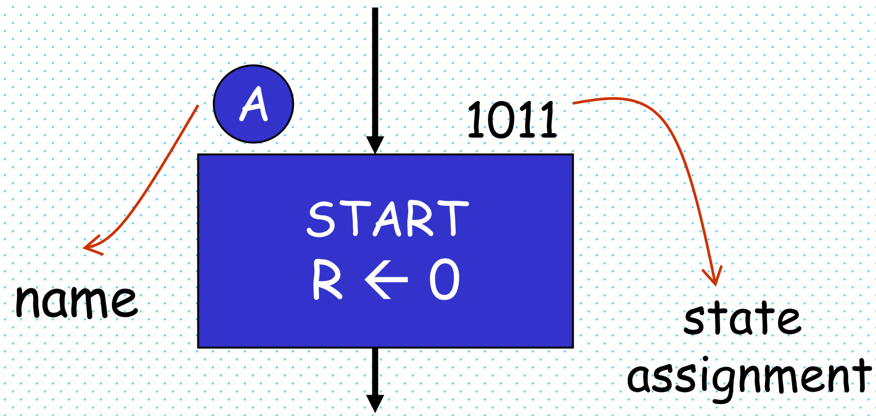
- Hardware algorithm
  - is a procedure for implementing the problem with a given piece of hardware equipments
  - specifies the control sequence and datapath tasks
- Algorithmic state machine (ASM) chart is a special type of flowchart used to define digital hardware algorithms.
  - state machine is another term for a sequential circuit
- ASM chart describes
  - the sequence of events,
  - events that occur at state transitions.

ASM Chart

- Three basic elements

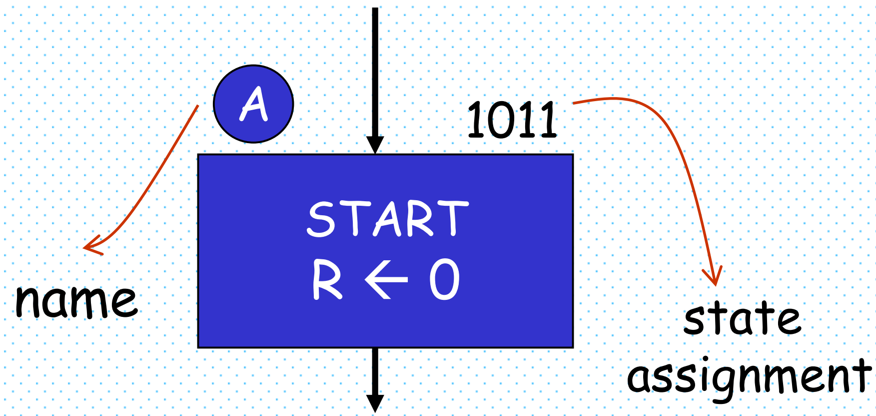
1. State box
2. Decision box
3. Conditional box

State Box

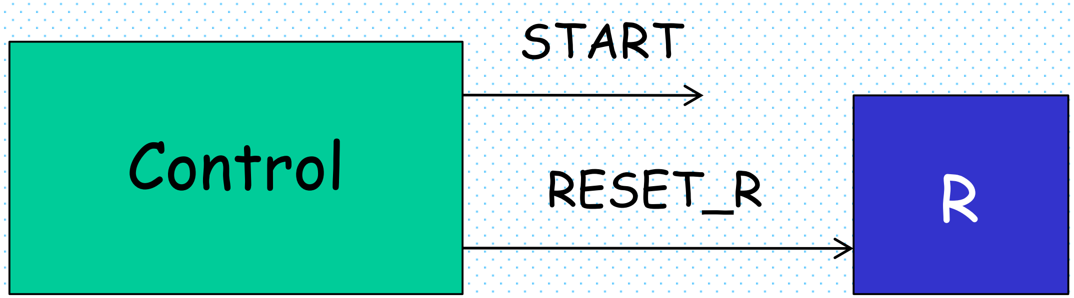


- The output signals (e.g. START) in the box take the specified values in the current state
  - if not specified in other states they are 0.
- The notation R ← 0 means that the register is cleared to 0 when the system transits from A to the next state

State Box

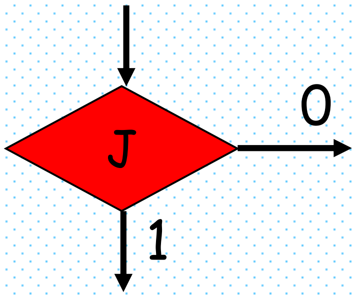


A
1011
START
R ← 0
name
state assignment



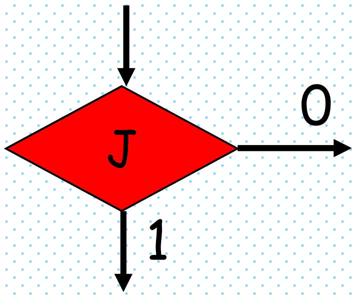
Control
START
RESET_R
R

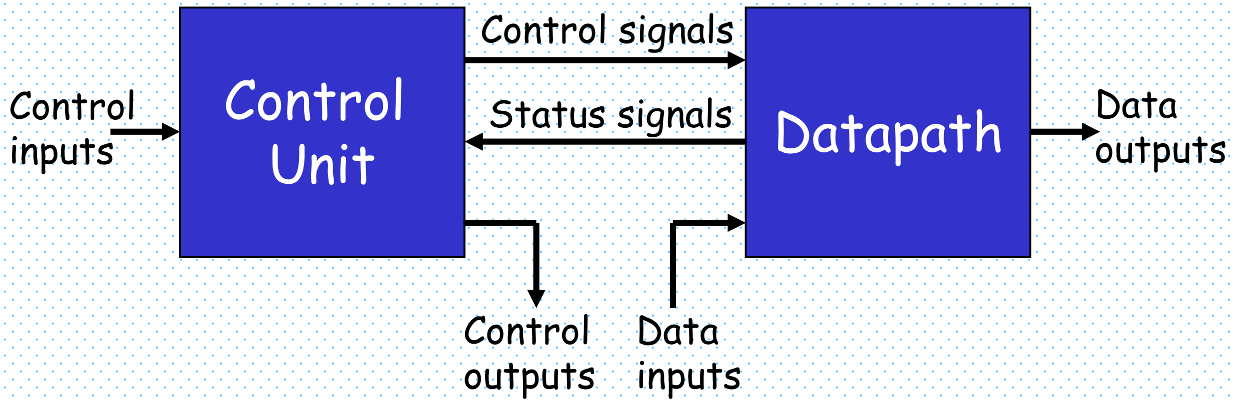
Decision Box

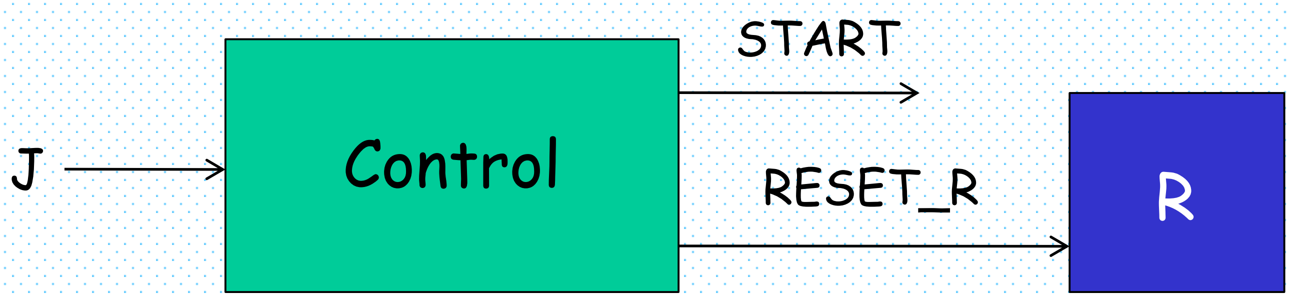


- Decision box has two or more branches going out.
- Decision is made based on the value of one or more input signals (e.g. signal J)
- Decision box must follow and be associated with a state box.
- Thus, the decision is made in the same clock cycle as the other actions of the state.

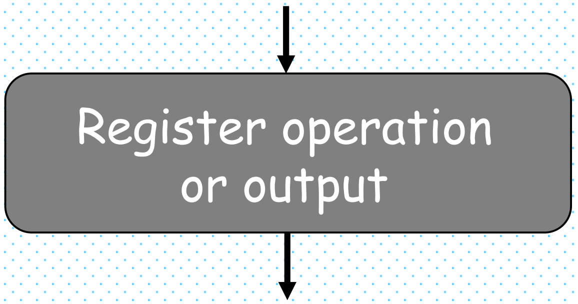
Decision Box





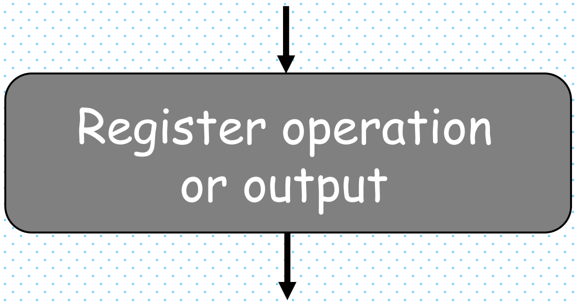


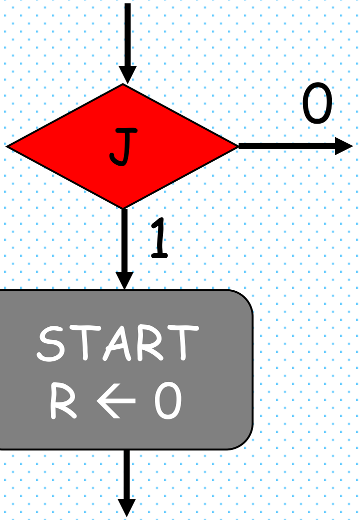
Conditional Box

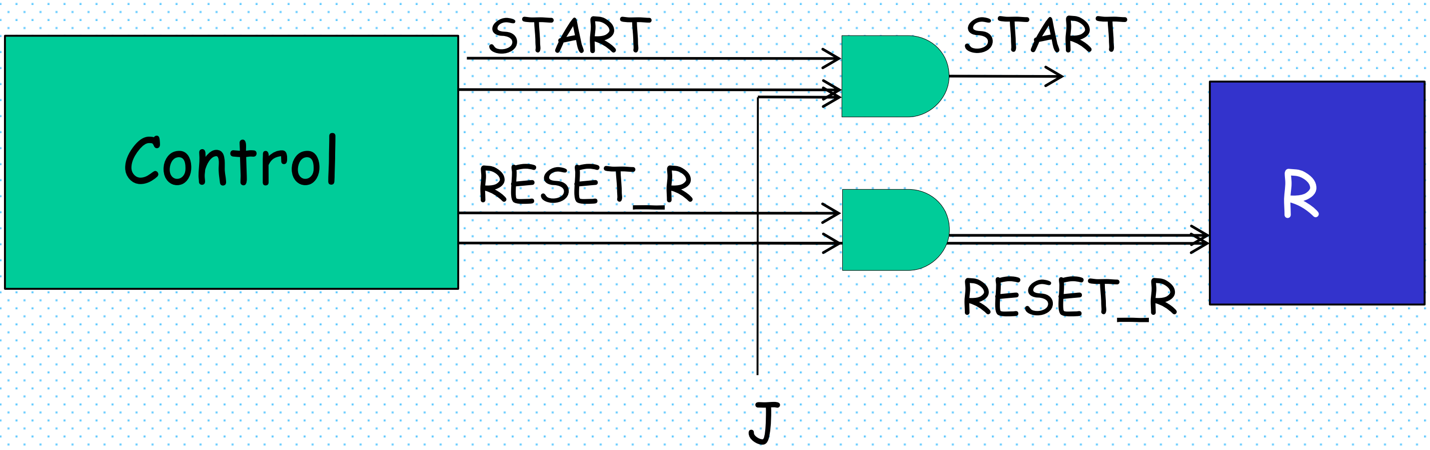


- A conditional box must follow a decision box.
- A conditional box is attached to a state box through one or more decision boxes.
- Therefore, the output signals in the conditional output box are asserted in the same clock cycle as those in the state box to which it is attached.
- The output signals can change during the current state as a result of changes on the inputs.
- The conditional output signals are sometimes referred as <u>Mealy outputs</u> since they depend on the input signals as well.

Conditional Box







Generate random $X_A < q$;

Calculate, $Y_A = q^{XA} \bmod p$

Calculate, $A_K = (Y_B)^{XA} \bmod p$

Y_A

Y_B

Generate random $X_B < q$;

Calculate, $Y_B = q^{XB} \bmod p$

Calculate, $B_K = (Y_A)^{XB} \bmod p$

Example: Modular Exponentiation (MExp)

Encryption in RSA cryptosytem: $C = M^E mod\ N$
Decryption in RSA cryptosytem: $M = C^D mod\ N$

Square and multiply algorithm

Input: $M = (m_{k-1}, m_{k-2}, \cdots, m_1, m_0)_2$
$E = (e_{k-1}, e_{k-2}, \cdots, e_1, e_0)_2$
$N = (n_{k-1}, n_{k-2}, \cdots, n_1, n_0)_2$
Output: $C = (c_{k-1}, c_{k-2}, \cdots, c_1, c_0)_2$

Step1: $C = 1$
Step2: for $i = k - 1 : 0$
Step3: $C = CxC\ mod\ N$
Step4: if $e_i = 1$
Step5: $C = CxM\ mod\ N$

$M = 2, E = 13, N = 15$
$E = (1101)_2$
$C = 1$
$e_3 = 1$
$C = CxM = 2$
$C = C^2 mod\ N = 2^2 mod\ 15 = 4$
$e_2 = 1$
$C = CxMmod\ N = 2^3 mod\ 15 = 8$
$C = C^2 mod\ N = 2^6 mod\ 15 = 4$
$e_1 = 0$
$C = C^2 mod\ N = 2^{12} mod\ 15$=1
$e_0 = 1$
$C = CxMmod\ N = 2^{13} mod\ 15 = 2$

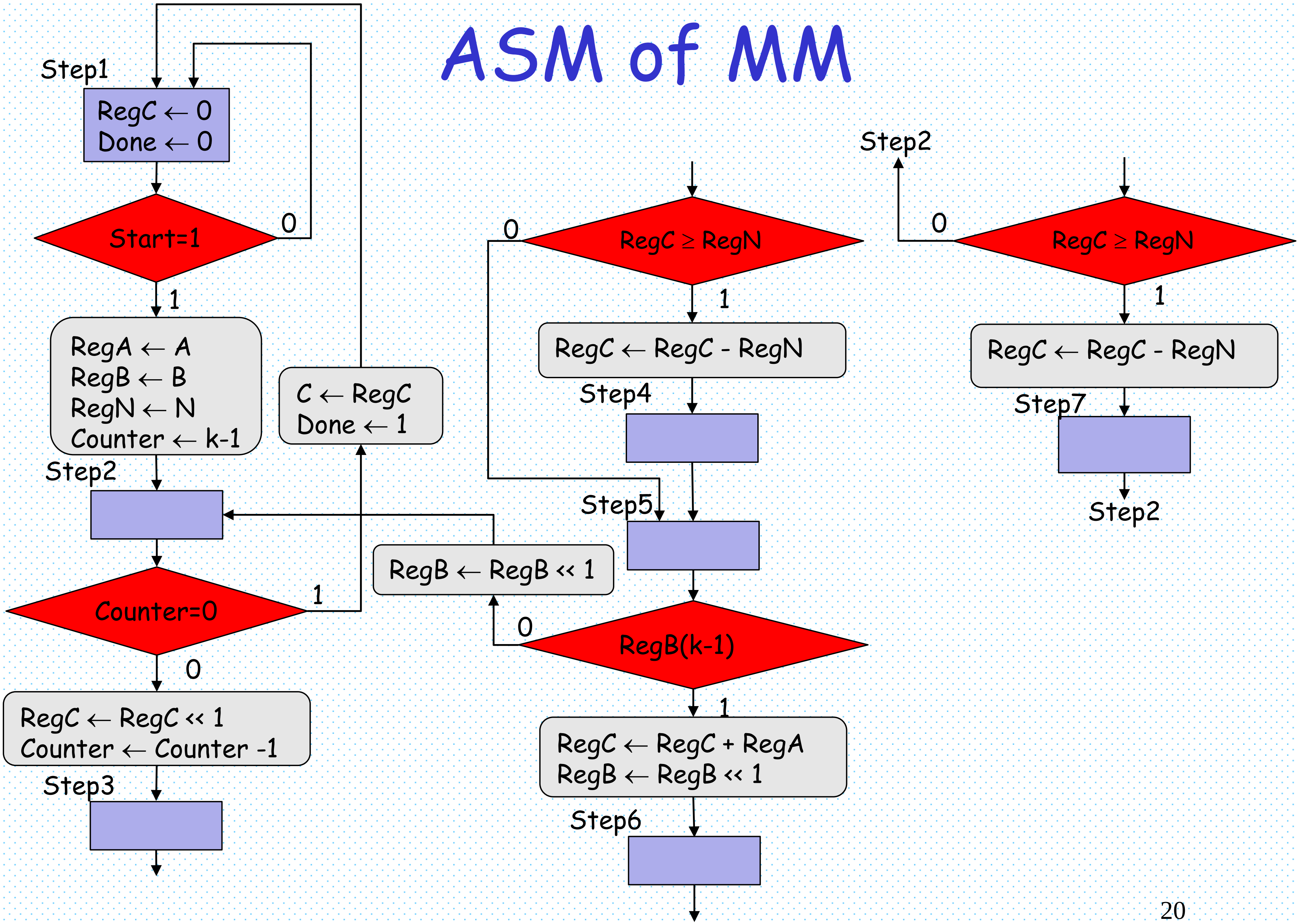
Example: Modular Multiplication (MM)

Left-to-right modular multipcation algorithm

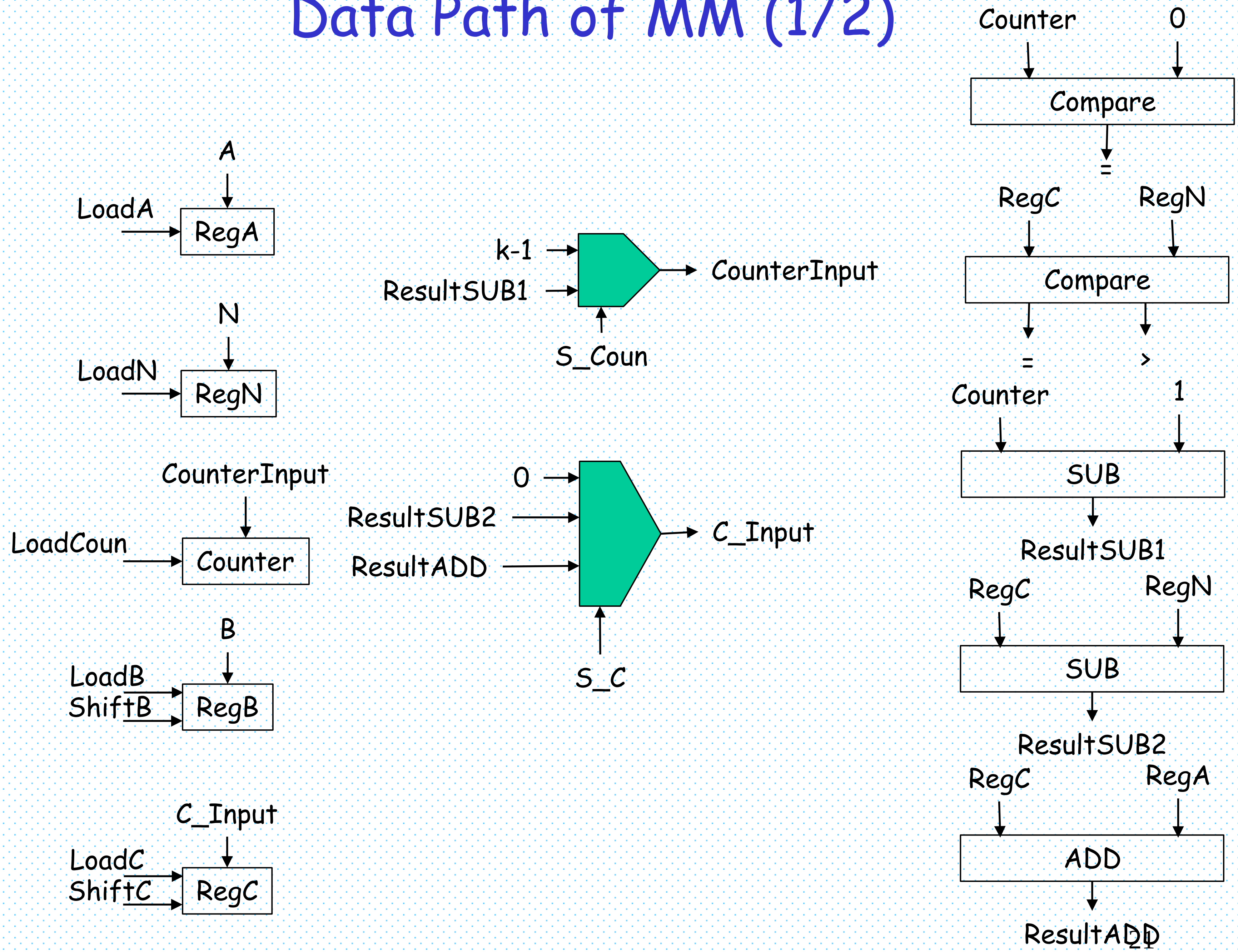
Input: $A = (a_{k-1}, a_{k-2}, \cdots, a_1, a_0)_2$
$B = (b_{k-1}, b_{k-2}, \cdots, b_1, b_0)_2$
$N = (n_{k-1}, n_{k-2}, \cdots, n_1, n_0)_2$
Output: $C = AxB \ mod \ N = (c_{k-1}, c_{k-2}, \cdots, c_1, c_0)_2$

Step1: $C = 0$
Step2: for $i = k - 1{:}\,0$
Step3: $C = 2xC$
if $C \geq N$
Step4: $C = C - N$
Step5: if $b_i = 1$
Step6: $C = C + A$
if $C \geq N$
Step7: $C = C - N$

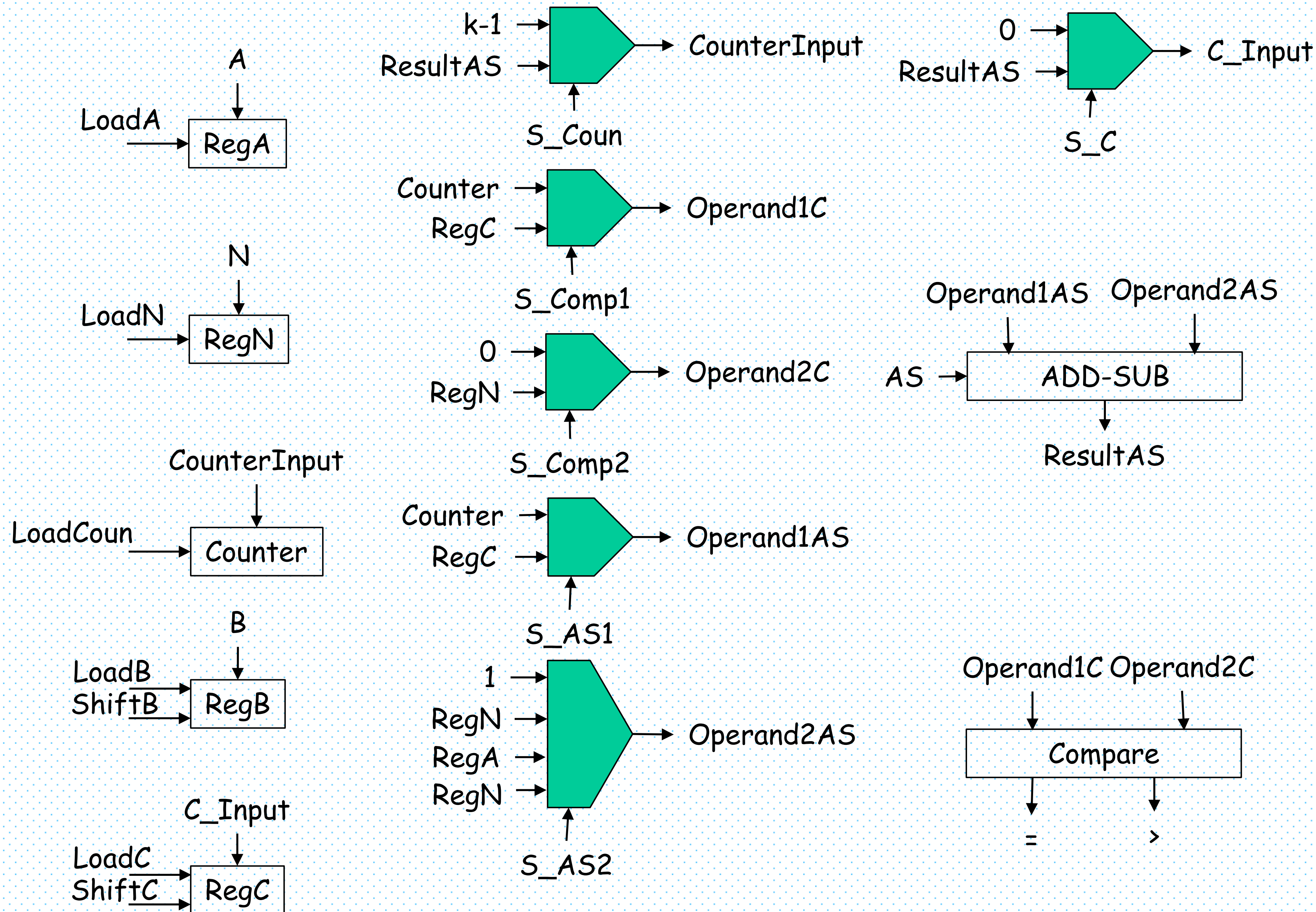
ASM of MM


Step1
RegC ← 0
Done ← 0
Start=1
0
1
RegA ← A
RegB ← B
RegN ← N
Counter ← k-1
C ← RegC
Done ← 1
Step2
Counter=0
1
0
RegC ← RegC << 1
Counter ← Counter -1
Step3
RegC ≥ RegN
0
1
RegC ← RegC - RegN
Step4
Step5
RegB ← RegB << 1
0
RegB(k-1)
1
RegC ← RegC + RegA
RegB ← RegB << 1
Step6
Step2
0
RegC ≥ RegN
1
RegC ← RegC - RegN
Step7
Step2

Data Path of MM (1/2)



Data Path of MM (2/2)



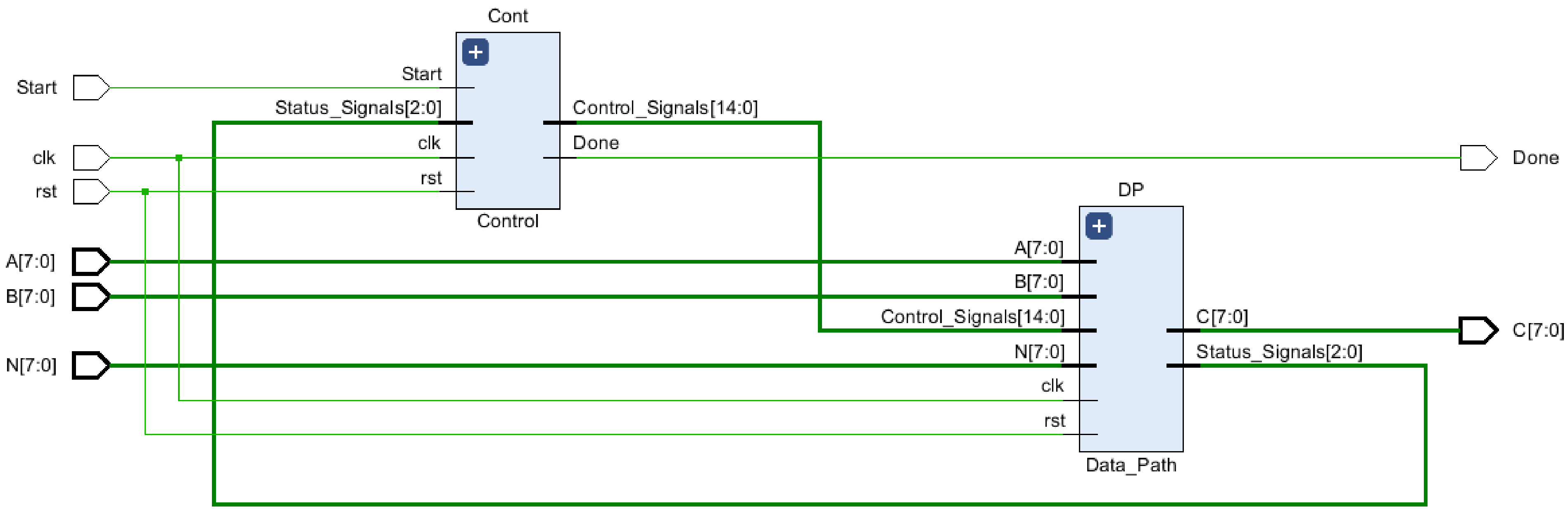
Control Signals of MM

		Status Signals			Control Signals													
State	Start	=	>	RegB (k-1)	Load A	Load N	Load Coun	Load B	Shift B	Load C	Shift C	S Coun	S Comp1	S Comp2	S AS1	S AS2	S C	AS
Step1	0	X	X	X	0	0	0	0	0	1	0	X	X	X	X	X	0	X
Step1	1	X	X	X	1	1	1	1	0	1	0	0	0	0	X	X	0	X
Step2	X	0	X	X	0	0	1	0	0	0	1	1	0	0	0	00	X	1
Step2	X	1	X	X	0	0	1	0	0	0	0	1	X	X	X	X	X	X
Step3	X	0	0	X	0	0	0	0	0	0	0	X	1	1	0	00	X	1
Step3	X	0	1	X	0	0	0	0	0	1	0	X	1	1	1	01	1	1
Step3	X	1	0	X	0	0	0	0	0	1	0	X	1	1	1	01	1	1
Step4	X	X	X	X	0	0	0	0	0	0	0	X	X	X	X	X	X	X
Step5	X	X	X	0	0	0	0	0	1	0	0	X	X	X	X	X	X	X
Step5	X	X	X	1	0	0	0	0	1	1	0	X	1	1	1	10	1	0
Step6	X	0	0	X	0	0	0	0	0	0	0	X	1	1	X	X	X	X
Step6	X	0	1	X	0	0	0	0	0	1	0	X	1	1	1	11	1	1
Step6	X	1	0	X	0	0	0	0	0	1	0	X	1	1	1	11	1	1
Step7	X	X	X	X	0	0	0	0	0	0	0	X	X	X	X	X	X	X

VHDL Code of MM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MM is
    generic (K : integer := 8);
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           Start :  in STD_LOGIC;
           A : in STD_LOGIC_VECTOR (K-1 downto 0);
           B : in STD_LOGIC_VECTOR (K-1 downto 0);
           N : in STD_LOGIC_VECTOR (K-1 downto 0);
           C : out STD_LOGIC_VECTOR (K-1 downto 0);
           Done : out STD_LOGIC);
end MM;

architecture Behavioral of MM is
component Data_Path is
    generic (K : integer := 8);
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           A : in STD_LOGIC_VECTOR (K-1 downto 0);
           B : in STD_LOGIC_VECTOR (K-1 downto 0);
           N : in STD_LOGIC_VECTOR (K-1 downto 0);
           Control_Signals : in STD_LOGIC_VECTOR (14
downto 0);
           Status_Signals : out STD_LOGIC_VECTOR (2
downto 0);
           C : out STD_LOGIC_VECTOR (K-1 downto 0));
end component;

component Control is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           Start :  in STD_LOGIC;
           Status_Signals : in STD_LOGIC_VECTOR (2 downto 0);
Control_Signals : out STD_LOGIC_VECTOR (14 downto 0);
Done : out STD_LOGIC);
end component;

signal Control_Signals : STD_LOGIC_VECTOR (14 downto 0);
signal Status_Signals : STD_LOGIC_VECTOR (2 downto 0);

begin

DP: Data_Path
    generic map(K)
    Port map(clk,rst,A,B,N,Control_Signals,Status_Signals,C);
Cont: Control
   Port map(clk,rst,Start,Status_Signals,Control_Signals,Done);

end Behavioral;
```

Schematic of MM



VHDL Code of Data Path (1/3)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity Data_Path is
    generic (K : integer := 8);
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           A : in STD_LOGIC_VECTOR (K-1 downto 0);
           B : in STD_LOGIC_VECTOR (K-1 downto 0);
           N : in STD_LOGIC_VECTOR (K-1 downto 0);
           Control_Signals : in STD_LOGIC_VECTOR (14 downto 0);
           Status_Signals : out STD_LOGIC_VECTOR (2 downto 0);
           C : out STD_LOGIC_VECTOR (K-1 downto 0));
end Data_Path;

architecture Behavioral of Data_Path is
component Load_Shift_Reg is
    generic (K : integer := 8);
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           Data_In : in STD_LOGIC_VECTOR (K-1 downto 0);
           Data_Out : out STD_LOGIC_VECTOR (K-1 downto 0);
           Load : in STD_LOGIC;
           Shift : in STD_LOGIC);
end component;
```

```
component MUX_2_to_1 is
    generic (K : integer := 8);
    Port ( S : in STD_LOGIC;
           I0 : in STD_LOGIC_VECTOR (K-1 downto 0);
           I1 : in STD_LOGIC_VECTOR (K-1 downto 0);
           Y : out STD_LOGIC_VECTOR (K-1 downto
0));
end component;
component MUX_4_to_1 is
    generic (K : integer := 8);
    Port ( S : in STD_LOGIC_VECTOR (1 downto 0);
           I0 : in STD_LOGIC_VECTOR (K-1 downto 0);
           I1 : in STD_LOGIC_VECTOR (K-1 downto 0);
           I2 : in STD_LOGIC_VECTOR (K-1 downto 0);
           I3 : in STD_LOGIC_VECTOR (K-1 downto 0);
           Y : out STD_LOGIC_VECTOR (K-1 downto
0));
end component;
component ADD_SUB is
    generic (K : integer := 8);
    Port ( A : in STD_LOGIC_VECTOR (K-1 downto
0);
           B : in STD_LOGIC_VECTOR (K-1 downto 0);
           Sum : out STD_LOGIC_VECTOR (K-1 downto
0);
           AS : in STD_LOGIC);
end component;
```

VHDL Code of Data Path (2/3)

```
component Compare is
    generic (K : integer := 8);
    Port ( A : in STD_LOGIC_VECTOR (K-1 downto 0);
           B : in STD_LOGIC_VECTOR (K-1 downto 0);
           Equal : out STD_LOGIC;
           Greater : out STD_LOGIC);
end component;
signal
LoadA,LoadN,LoadCoun,LoadB,ShiftB,LoadC,ShiftC,SCoun,SComp
1,SComp2,SAS1,SC,AS : std_logic;
signal SAS2 : STD_LOGIC_VECTOR (1 downto 0);
signal RegA, RegN, RegB, Counter_Input, Counter, C_Input,
RegC, temp_k, ResultAS, Operand1C, Operand2C, Operand1AS,
temp_1, Operand2AS : STD_LOGIC_VECTOR (K-1 downto 0);

begin
LoadA <= Control_Signals(14);
LoadN <= Control_Signals(13);
LoadCoun <= Control_Signals(12);
LoadB <= Control_Signals(11);
ShiftB <= Control_Signals(10);
LoadC <= Control_Signals(9);
ShiftC <= Control_Signals(8);
SCoun <= Control_Signals(7);
SComp1 <= Control_Signals(6);
SComp2 <= Control_Signals(5);
SAS1 <= Control_Signals(4);
```

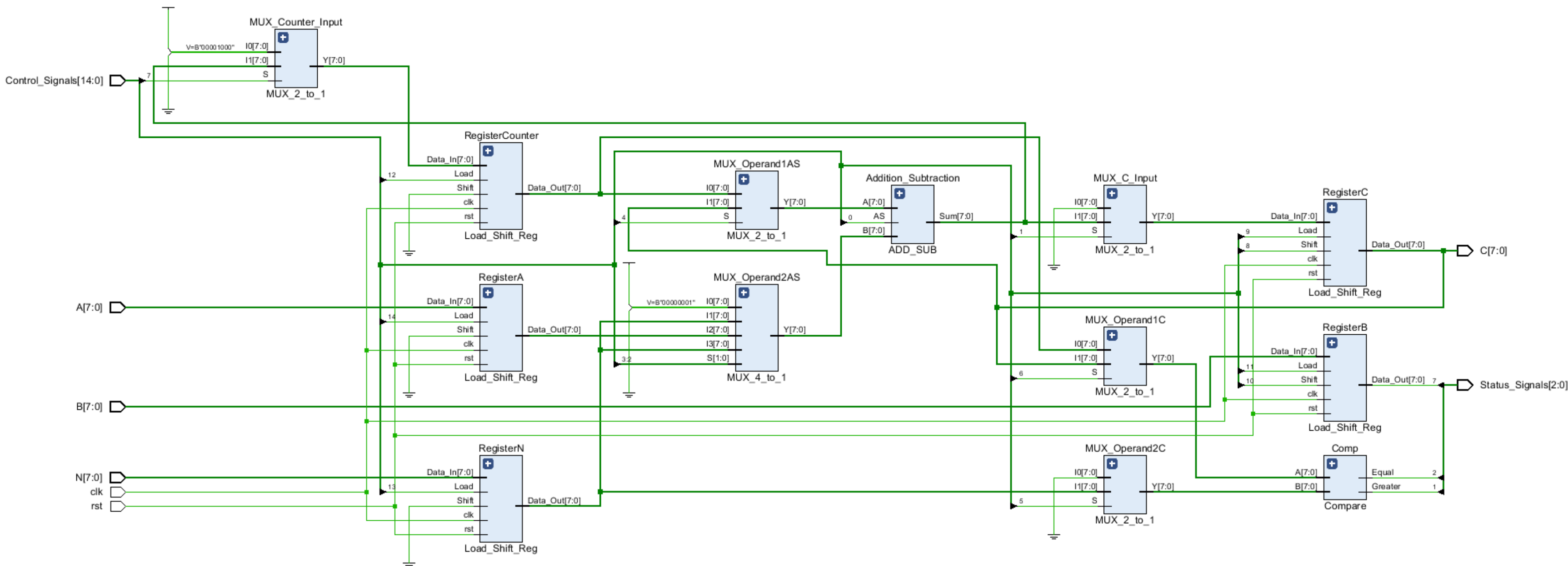
```
SAS2 <= Control_Signals(3 downto 2);
SC <= Control_Signals(1);
AS <= Control_Signals(0);

RegisterA: Load_Shift_Reg
        generic map(K)
        Port map(clk,rst,A,RegA,LoadA,'0');
RegisterN: Load_Shift_Reg
        generic map(K)
        Port map(clk,rst,N,RegN,LoadN,'0');
RegisterCounter: Load_Shift_Reg
        generic map(K)
        Port
map(clk,rst,Counter_Input,Counter,LoadCoun,'0');
RegisterB: Load_Shift_Reg
        generic map(K)
        Port map(clk,rst,B,RegB,LoadB,ShiftB);
RegisterC: Load_Shift_Reg
        generic map(K)
        Port
map(clk,rst,C_Input,RegC,LoadC,ShiftC);

temp_k <= conv_std_logic_vector(K, K);
```

VHDL Code of Data Path (3/3)

```
MUX_Counter_Input: MUX_2_to_1
            generic map(K)
            Port map(SCoun,temp_k,ResultAS,Counter_Input);
MUX_Operand1C: MUX_2_to_1
            generic map(K)
            Port map(SComp1,Counter,RegC,Operand1C);
MUX_Operand2C: MUX_2_to_1
            generic map(K)
            Port map(SComp2,(others => '0'),RegN,Operand2C);
MUX_Operand1AS: MUX_2_to_1
            generic map(K)
            Port map(SAS1,Counter,RegC,Operand1AS);
MUX_C_Input: MUX_2_to_1
            generic map(K)
            Port map(SC,(others => '0'),ResultAS,C_Input);
temp_1(0) <= '1';
temp_1(K-1 downto 1) <= (others => '0');

MUX_Operand2AS: MUX_4_to_1
            generic map(K)
            Port
map(SAS2,temp_1,RegN,RegA,RegN,Operand2AS);

Addition_Subtraction: ADD_SUB
              generic map(K)
              Port map(
Operand1AS,Operand2AS,ResultAS,AS);
```

```
Comp: Compare
    generic map(K)
    Port map(
Operand1C,Operand2C,Status_Signals(2),Status_Si
gnals(1));

Status_Signals(0) <= RegB(K-1);
C <= RegC;

end Behavioral;
```

Schematic of Data Path



VHDL Code of Control (1/3)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Control is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           Start :  in STD_LOGIC;
           Status_Signals : in STD_LOGIC_VECTOR (2 downto 0);
           Control_Signals : out STD_LOGIC_VECTOR (14 downto 0);
           Done : out STD_LOGIC);
end Control;
architecture Behavioral of Control is
  type state_type is
(Step1,Step2,Step3,Step4,Step5,Step6,Step7);
  signal current_state,next_state : state_type;
begin
  NS: process(current_state,Start,Status_Signals)
  begin
    case(current_state) is
      when Step1 =>
        if(Start='1') then
          next_state <= Step2;
        else
          next_state <= Step1;
        end if;
      when Step2 =>
        if(Status_Signals(2)='1') then
          next_state <= Step1;
        else
          next_state <= Step3;
        end if;
      when Step3 =>
        if(Status_Signals(2 downto 1)="00") then
          next_state <= Step5;
        else
          next_state <= Step4;
        end if;
      when Step4 =>
        next_state <= Step5;
      when Step5 =>
        if(Status_Signals(0)='1') then
          next_state <= Step6;
        else
          next_state <= Step2;
        end if;
```

VHDL Code of Control (2/3)

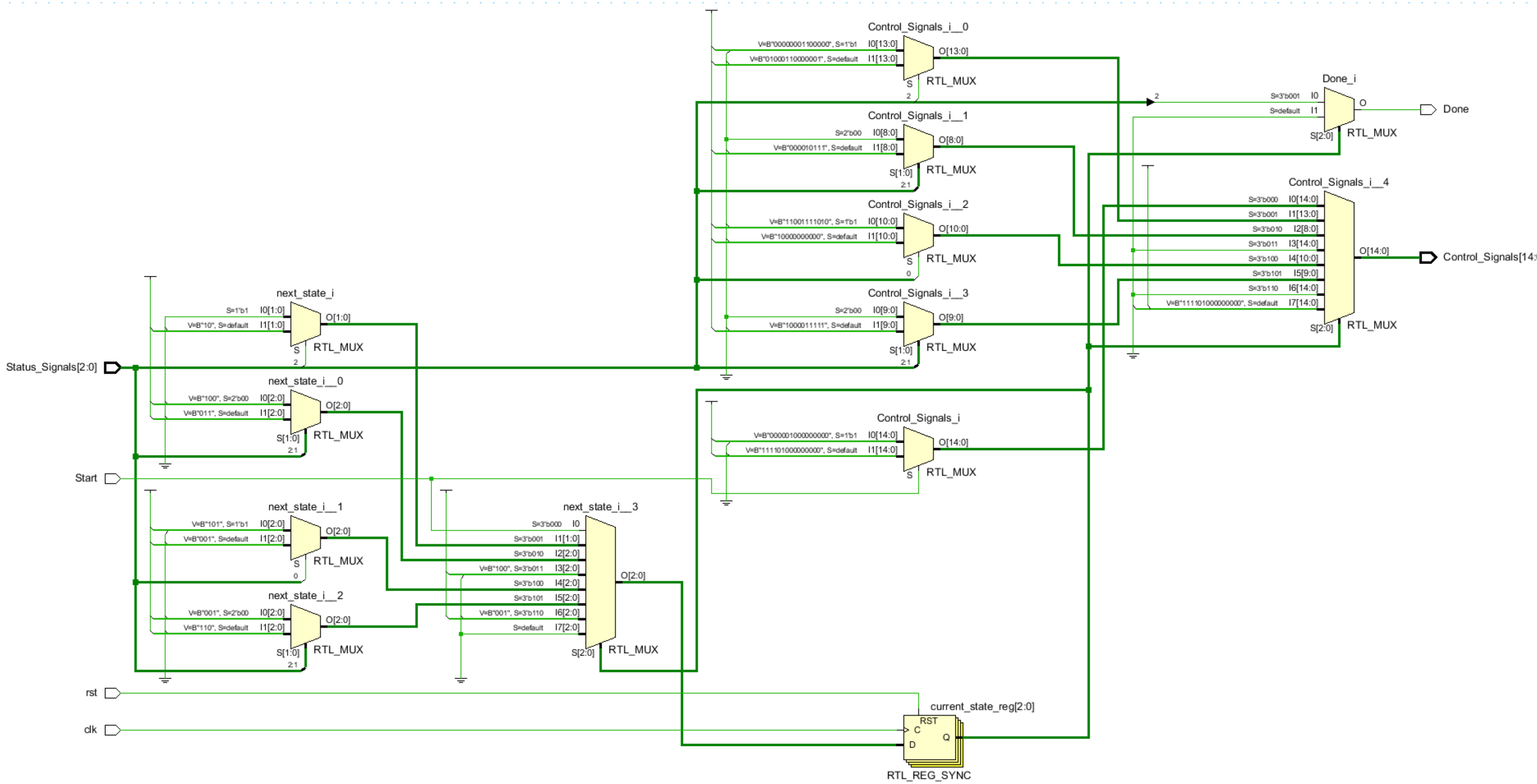
```
when Step6 =>
      if(Status_Signals(2 downto 1)="00") then
        next_state <= Step2;
      else
        next_state <= Step7;
      end if;
    when Step7 =>
      next_state <= Step2;
    when others =>
      next_state <= Step1;
  end case;
end process;

ST: process(clk)
begin
  if(clk'event and clk='1') then
    if(rst='1') then
      current_state <= Step1;
    else
      current_state <= next_state;
    end if;
  end if;
end process;
```

```
O_Done: process(current_state,Start,Status_Signals)
 begin
  case(current_state) is
    when Step2 =>
      if(Status_Signals(2)='1') then
        Done <= '1';
      else
        Done <= '0';
      end if;
    when others =>
      Done <= '0';
  end case;
 end process;

 O_Control_Signals: process(current_state,Start,Status_Signals)
 begin
  case(current_state) is
    when Step1 =>
      if(Start='1') then
        Control_Signals <= "1111010000XXX0X";
      else
        Control_Signals <= "00000100XXXXX0X";
      end if;
```

VHDL Code of Control (3/3)

```
when Step2 =>
  if(Status_Signals(2)='1') then
    Control_Signals <= "00100001XXXXXX1";
  else
    Control_Signals <= "0010001100000X1";
  end if;
when Step3 =>
  if(Status_Signals(2 downto 1)="00") then
    Control_Signals <= "0000000X11000X1";
  else
    Control_Signals <= "0000010X1110111";
  end if;
when Step4 =>
  Control_Signals <= "0000000XXXXXXXX";
when Step5 =>
  if(Status_Signals(0)='1') then
    Control_Signals <= "0000110X1111010";
  else
    Control_Signals <= "0000100XXXXXXXX";
  end if;
when Step6 =>
  if(Status_Signals(2 downto 1)="00") then
    Control_Signals <= "0000000X11XXXXX";
  else
    Control_Signals <= "0000010X1111111";
  end if;
when Step7 =>
        Control_Signals <= "0000000XXXXXXXX";
      when others =>
        Control_Signals <= "1111010000XXX0X";
    end case;
  end process;

end Behavioral;
```

Schematic of Control

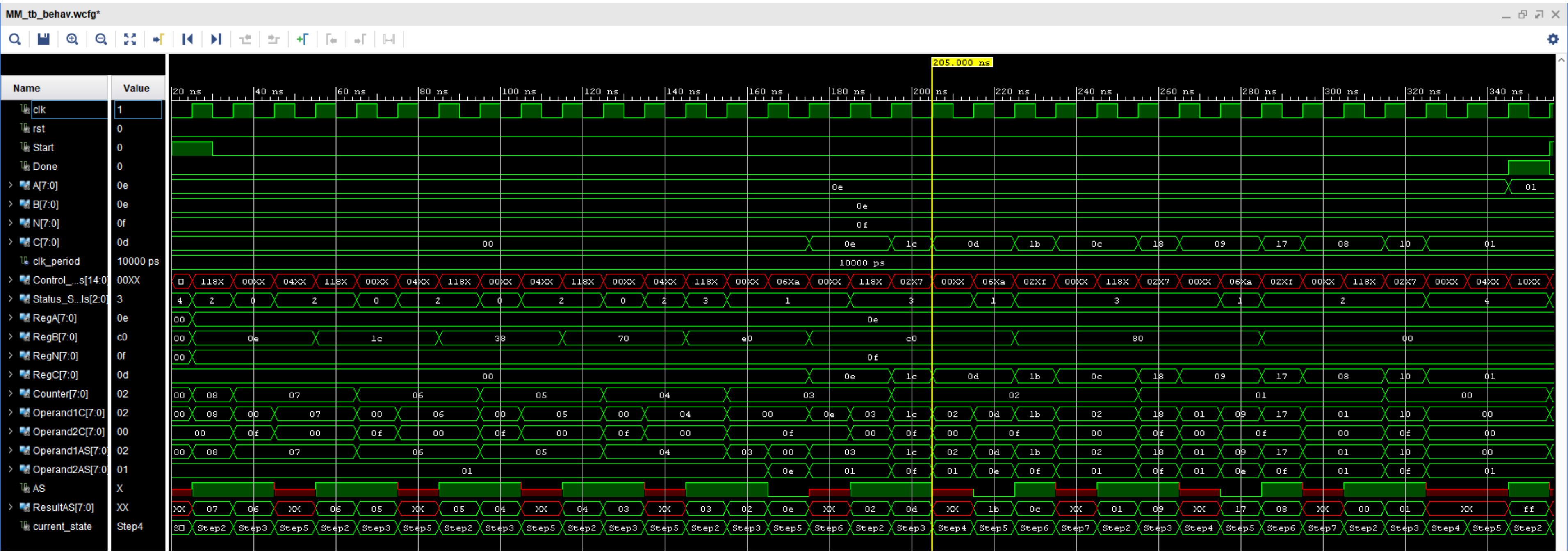


Testbench for MM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity MM_tb is
end MM_tb;
architecture Behavioral of MM_tb is
component MM is
    generic (K : integer := 8);
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           Start :  in STD_LOGIC;
           A : in STD_LOGIC_VECTOR (K-1 downto 0);
           B : in STD_LOGIC_VECTOR (K-1 downto 0);
           N : in STD_LOGIC_VECTOR (K-1 downto 0);
           C : out STD_LOGIC_VECTOR (K-1 downto
0);
           Done : out STD_LOGIC);
end component;
signal clk,rst,Start,Done : STD_LOGIC;
signal A,B,N,C : STD_LOGIC_VECTOR (7 downto
0);
constant clk_period : time := 10 ns;
begin
DUT: MM
     generic map(8)
     Port map(clk,rst,Start,A,B,N,C,Done);
```

```
clk_process: process
  begin
    clk <= '0';
    wait for clk_period/2;  --for 0.5 ns signal is '0'.
    clk <= '1';
    wait for clk_period/2;  --for next 0.5 ns signal is '1'.
  end process;
Input_Application: process
  begin
    A <= "00001110";    B <= "00001110";    N <= "00001111";
    rst <= '1';
    wait for 10 ns;    rst <= '0';
    wait for 10 ns;    Start <= '1';
    wait for 10 ns;    Start <= '0';
    wait until Done='1';
    for i in 1 to 12 loop
      A <= C;
      wait for 10 ns;      Start <= '1';
      wait for 10 ns;      Start <= '0';
      wait until Done='1';
    end loop;
    wait;
  end process;

end Behavioral;
```

Test Waveform of MM



Reminder: Modular Exponentiation (MExp)

$$C = A^B mod\ N$$

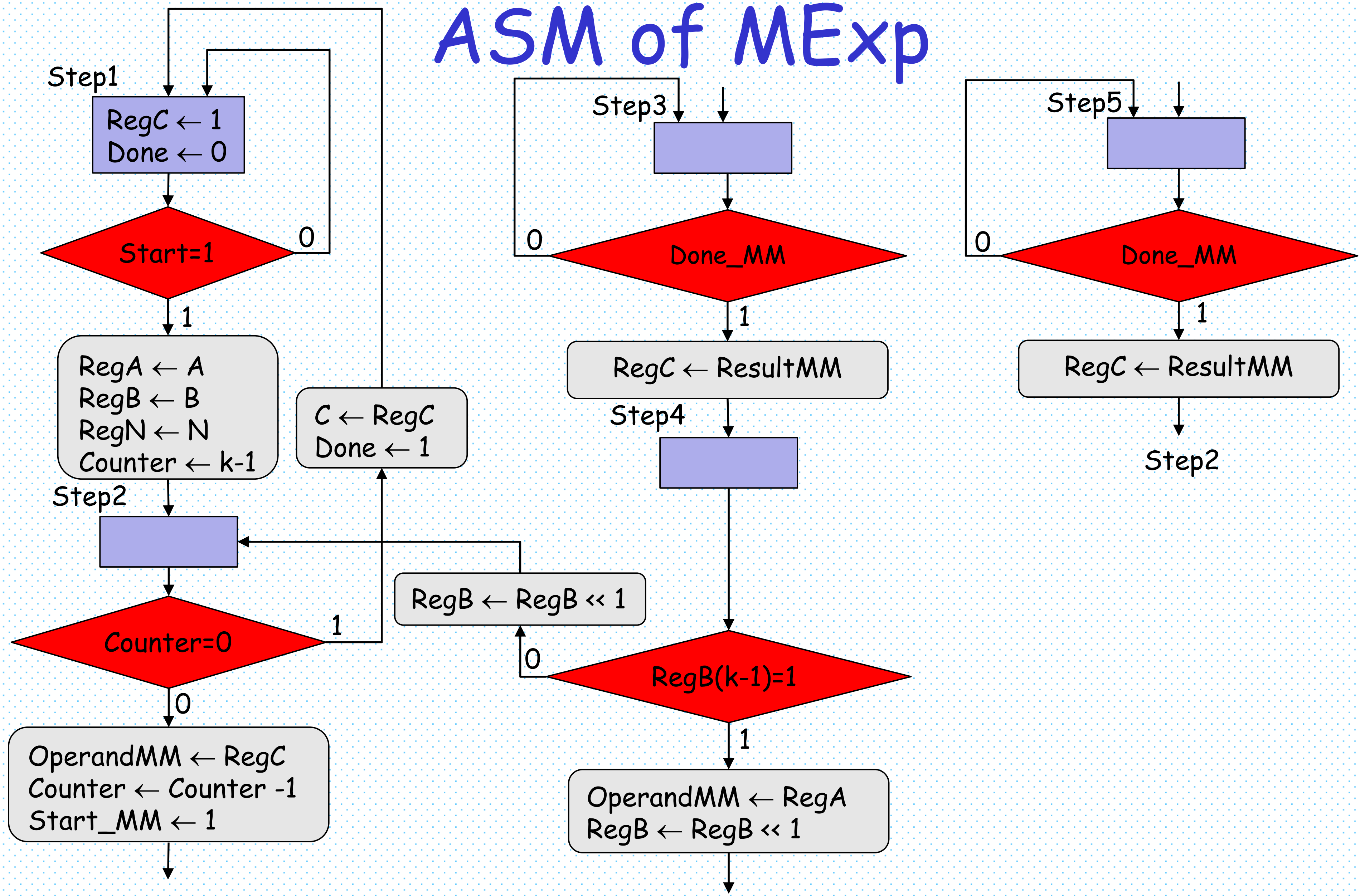
Square and multiply algorithm

Input: $A = (a_{k-1}, a_{k-2}, \cdots, a_1, a_0)_2, B = (b_{k-1}, b_{k-2}, \cdots, b_1, b_0)_2$
$N = (n_{k-1}, n_{k-2}, \cdots, n_1, n_0)_2$

Output: $C = (c_{k-1}, c_{k-2}, \cdots, c_1, c_0)_2$

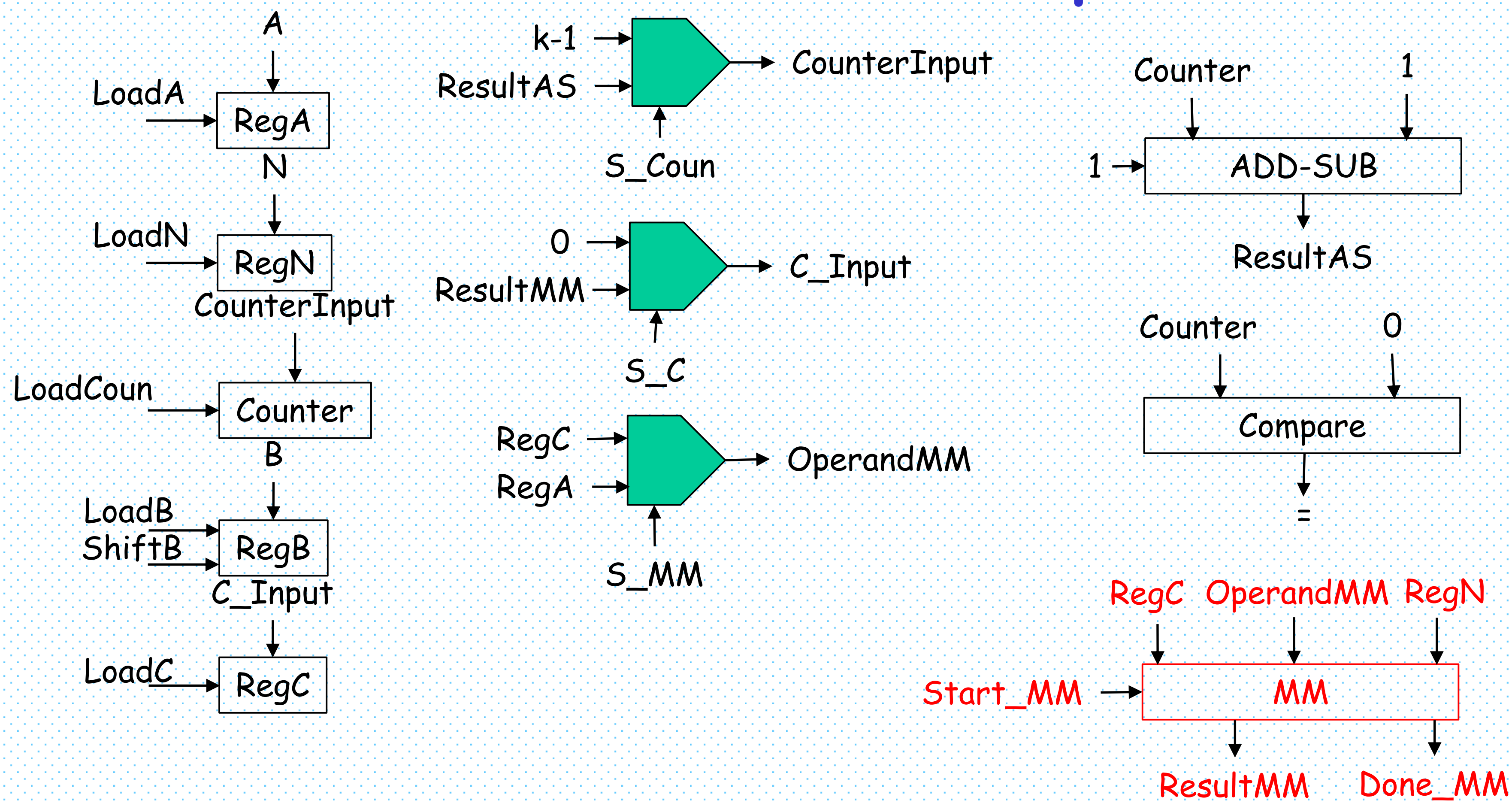
Step1: $C = 1$
Step2: for $i = k - 1{:}\,0$
Step3: $C = CxC\ mod\ N$
Step4: if $b_i = 1$
Step5: $C = CxA\ mod\ N$

ASM of MExp



Step1
RegC ← 1
Done ← 0
Start=1
0
1
RegA ← A
RegB ← B
RegN ← N
Counter ← k-1
C ← RegC
Done ← 1
Step2
Counter=0
1
0
OperandMM ← RegC
Counter ← Counter -1
Start_MM ← 1
RegB ← RegB << 1
Step3
0
Done_MM
1
RegC ← ResultMM
Step4
RegB(k-1)=1
0
1
OperandMM ← RegA
RegB ← RegB << 1
Step5
0
Done_MM
1
RegC ← ResultMM
Step2

Data Path of MExp



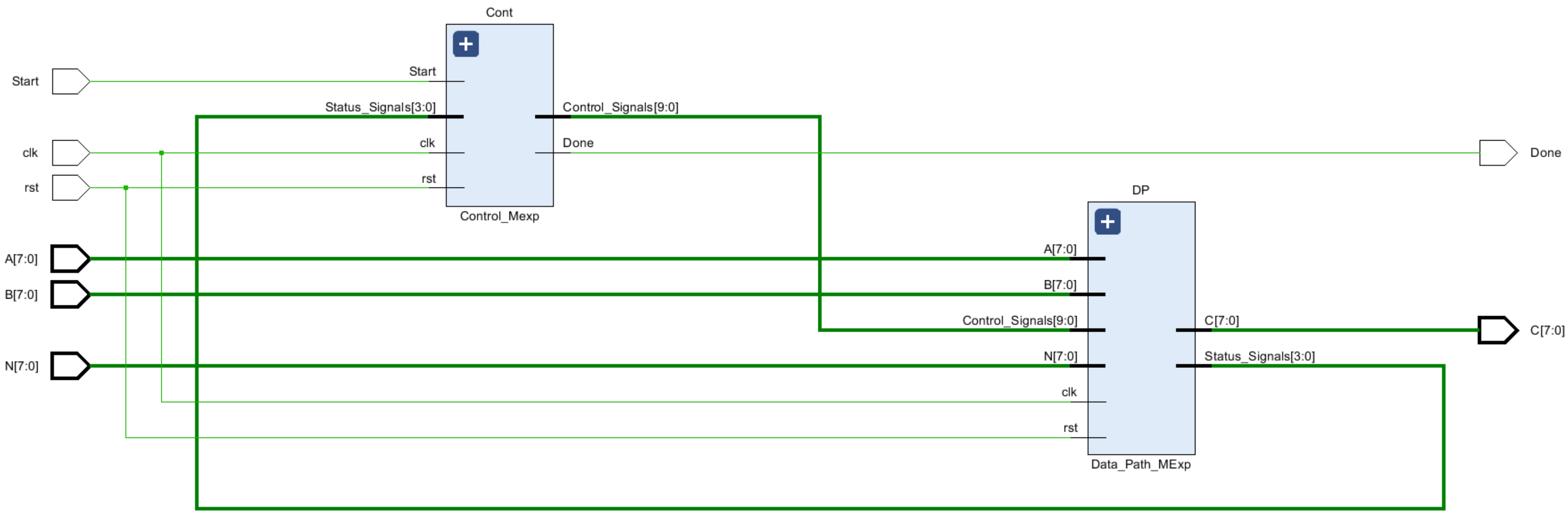
Control Signals of MExp

State	Start	=	RegB (k-1)	Done_ MM	Load A	Load N	Load Coun	Load B	Shift B	Load C	S Coun	S C	S MM	Start MM
Step1	0	X	X	X	0	0	0	0	0	1	0	0	X	0
Step1	1	X	X	X	1	1	1	1	0	1	0	0	0	0
Step2	X	0	X	X	0	0	1	0	0	0	1	0	0	1
Step2	X	1	X	X	0	0	1	0	0	0	0	0	X	X
Step3	X	X	X	0	0	0	0	0	0	0	0	1	0	0
Step3	X	X	X	1	0	0	0	0	0	1	0	1	0	X
Step4	X	X	0	X	0	0	0	0	1	0	X	X	X	0
Step4	X	X	1	X	0	0	0	0	1	0	X	X	1	1
Step5	X	X	X	0	0	0	0	0	0	1	0	1	1	0
Step5	X	X	X	1	0	0	0	0	0	1	0	1	1	X

VHDL Code of MExp

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MExp is
    generic (K : integer := 8);
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           Start :  in STD_LOGIC;
           A : in STD_LOGIC_VECTOR (7 downto 0);
           B : in STD_LOGIC_VECTOR (7 downto 0);
           N : in STD_LOGIC_VECTOR (7 downto 0);
           C : out STD_LOGIC_VECTOR (7 downto 0);
           Done : out STD_LOGIC);
end MExp;

architecture Behavioral of MExp is
component Data_Path_ MExp is
    generic (K : integer := 8);
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           A : in STD_LOGIC_VECTOR (7 downto 0);
           B : in STD_LOGIC_VECTOR (7 downto 0);
           N : in STD_LOGIC_VECTOR (7 downto 0);
           Control_Signals : in STD_LOGIC_VECTOR (9
downto 0);
           Status_Signals : out STD_LOGIC_VECTOR (2
downto 0);
           C : out STD_LOGIC_VECTOR (7 downto 0));
end component;

component Control_MExp is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           Start :  in STD_LOGIC;
           Status_Signals : in STD_LOGIC_VECTOR (2 downto 0);
Control_Signals : out STD_LOGIC_VECTOR (9 downto 0); Done
: out STD_LOGIC);
end component;

signal Control_Signals : STD_LOGIC_VECTOR (9 downto 0);
signal Status_Signals : STD_LOGIC_VECTOR (2 downto 0);

begin

DP: Data_Path_MExp
    generic map(K)
    Port map(clk,rst,A,B,N,Control_Signals,Status_Signals,C);
Cont: Control_MExp
   Port map(clk,rst,Start,Status_Signals,Control_Signals,Done);

end Behavioral;
```

Schematic of MExp



VHDL Code of Data Path of MExp (1/3)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity Data_Path_MExp is
    generic (K : integer := 8);
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           A : in STD_LOGIC_VECTOR (7 downto 0);
           B : in STD_LOGIC_VECTOR (7 downto 0);
           N : in STD_LOGIC_VECTOR (7 downto 0);
           Control_Signals : in STD_LOGIC_VECTOR (9 downto 0);
           Status_Signals : out STD_LOGIC_VECTOR (2 downto 0);
           C : out STD_LOGIC_VECTOR (7 downto 0));
end Data_Path_MExp;

architecture Behavioral of Data_Path_MExp is
component Load_Shift_Reg is
    generic (K : integer := 8);
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           Data_In : in STD_LOGIC_VECTOR (K-1 downto 0);
           Data_Out : out STD_LOGIC_VECTOR (K-1 downto 0);
           Load : in STD_LOGIC;
           Shift : in STD_LOGIC);
end component;
component MUX_2_to_1 is
    generic (K : integer := 8);
    Port ( S : in STD_LOGIC;
           I0 : in STD_LOGIC_VECTOR (K-1 downto 0);
           I1 : in STD_LOGIC_VECTOR (K-1 downto 0);
           Y : out STD_LOGIC_VECTOR (K-1 downto
0));
end component;
component ADD_SUB is
    generic (K : integer := 8);
    Port ( A : in STD_LOGIC_VECTOR (K-1 downto
0);
           B : in STD_LOGIC_VECTOR (K-1 downto 0);
           Sum : out STD_LOGIC_VECTOR (K-1 downto
0);
           AS : in STD_LOGIC);
end component;
component Compare_Eq is
    generic (K : integer := 8);
    Port ( A : in STD_LOGIC_VECTOR (K-1 downto
0);
           B : in STD_LOGIC_VECTOR (K-1 downto 0);
           Equal : out STD_LOGIC);
end component;
```

VHDL Code of Data Path of MExp (2/3)

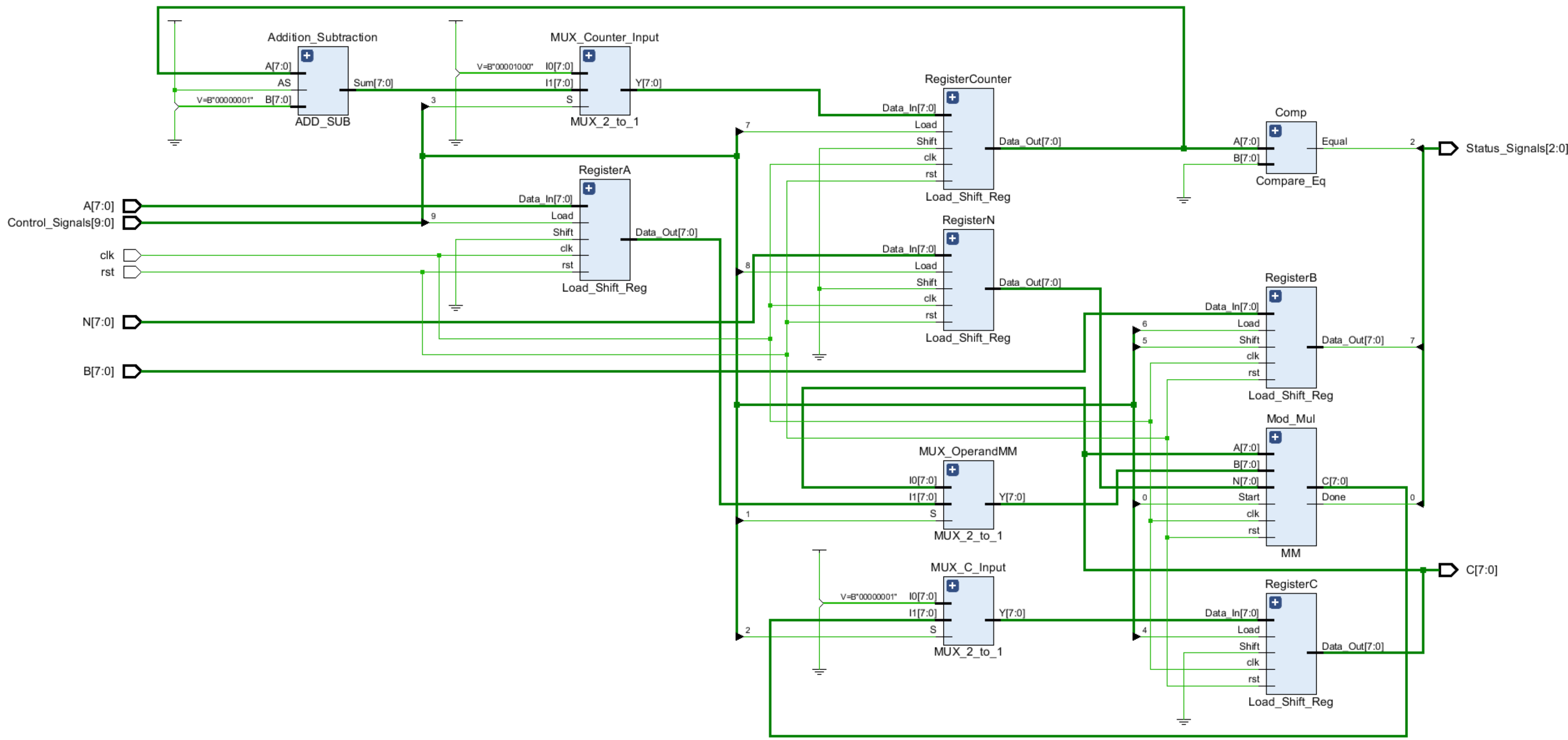
```
component MM is
    generic (K : integer := 8);
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           Start :  in STD_LOGIC;
           A : in STD_LOGIC_VECTOR (7 downto 0);
           B : in STD_LOGIC_VECTOR (7 downto 0);
           N : in STD_LOGIC_VECTOR (7 downto 0);
           C : out STD_LOGIC_VECTOR (7 downto 0);
           Done : out STD_LOGIC);
end component;
signal
LoadA,LoadN,LoadCoun,LoadB,ShiftB,LoadC,SCoun,SC,SMM,Star
t_MM,Done_MM : std_logic;
signal RegA, RegN, RegB, Counter_Input, Counter, C_Input,
RegC, temp_k, ResultAS, temp_1, OperandMM, ResultMM :
STD_LOGIC_VECTOR (K-1 downto 0);
begin
LoadA <= Control_Signals(9);
LoadN <= Control_Signals(8);
LoadCoun <= Control_Signals(7);
LoadB <= Control_Signals(6);
ShiftB <= Control_Signals(5);
LoadC <= Control_Signals(4);
SCoun <= Control_Signals(3);
SC <= Control_Signals(2);
SMM <= Control_Signals(1);
```

```
Start_MM <= Control_Signals(0);

RegisterA: Load_Shift_Reg
        generic map(K)
        Port map(clk,rst,A,RegA,LoadA,'0');
RegisterN: Load_Shift_Reg
        generic map(K)
        Port map(clk,rst,N,RegN,LoadN,'0');
RegisterCounter: Load_Shift_Reg
        generic map(K)
        Port
map(clk,rst,Counter_Input,Counter,LoadCoun,'0');
RegisterB: Load_Shift_Reg
        generic map(K)
        Port map(clk,rst,B,RegB,LoadB,ShiftB);
RegisterC: Load_Shift_Reg
        generic map(K)
        Port map(clk,rst,C_Input,RegC,LoadC,'0');

temp_k <= conv_std_logic_vector(K, 8);

MUX_Counter_Input: MUX_2_to_1
              generic map(K)
              Port
map(SCoun,temp_k,ResultAS,Counter_Input);
```

VHDL Code of Data Path of MExp (3/3)

```
MUX_OperandMM: MUX_2_to_1
                generic map(K)
                Port map(SMM,RegC,RegA,OperandMM);
temp_1(0) <= '1';
temp_1(K-1 downto 1) <= (others => '0');

MUX_C_Input: MUX_2_to_1
                generic map(K)
                Port map(SC,temp_1,ResultMM,C_Input);

Addition_Subtraction: ADD_SUB
                  generic map(K)
                  Port map( Counter,temp_1,ResultAS,'1');
Comp: Compare_Eq
     generic map(K)
     Port map( Counter,(others => '0'),Status_Signals(2));

Mod_Mul: MM
        generic map(K)
        Port map(
clk,rst,Start_MM,RegC,OperandMM,RegN,ResultMM,Done_MM);

Status_Signals(1) <= RegB(K-1);
Status_Signals(0) <= Done_MM;
C <= RegC;

end Behavioral;
```

Schematic of ModExp Data Path



VHDL Code of Control (1/3)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Control_Mexp is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           Start :  in STD_LOGIC;
           Status_Signals : in STD_LOGIC_VECTOR (2 downto 0);
           Control_Signals : out STD_LOGIC_VECTOR (9 downto 0);
           Done : out STD_LOGIC);
end Control_Mexp;
architecture Behavioral of Control_Mexp is
  type state_type is (Step1,Step2,Step3,Step4,Step5);
  signal current_state,next_state : state_type;
begin
  NS: process(current_state,Start,Status_Signals)
  begin
    case(current_state) is
      when Step1 =>
        if(Start='1') then
          next_state <= Step2;
        else
          next_state <= Step1;
        end if;
```

```
when Step2 =>
        if(Status_Signals(2)='1') then
          next_state <= Step1;
        else
          next_state <= Step3;
        end if;
      when Step3 =>
        if(Status_Signals(0)='1') then
          next_state <= Step4;
        else
          next_state <= Step3;
        end if;
      when Step4 =>
        if(Status_Signals(1)='1') then
          next_state <= Step5;
        else
          next_state <= Step2;
        end if;
      when Step5 =>
        if(Status_Signals(0)='1') then
          next_state <= Step2;
        else
          next_state <= Step5;
        end if;
```

VHDL Code of Control of MExp (2/3)

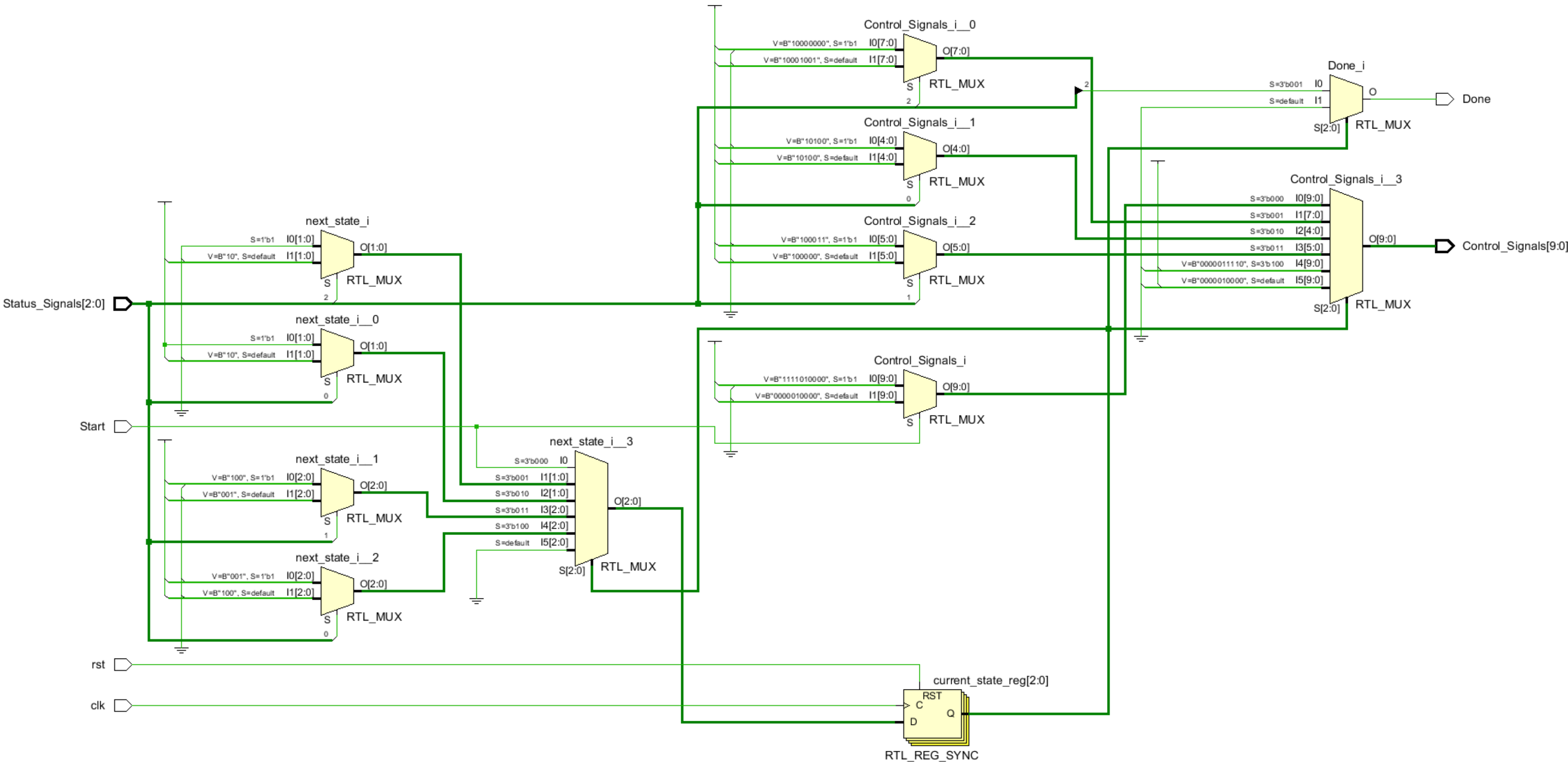
```
when others =>
        next_state <= Step1;
    end case;
  end process;
  ST: process(clk)
  begin
    if(clk'event and clk='1') then
      if(rst='1') then
        current_state <= Step1;
      else
        current_state <= next_state;
      end if;
    end if;
  end process;
  O_Done:
process(current_state,Start,Status_Signals)
  begin
    case(current_state) is
      when Step2 =>
        if(Status_Signals(2)='1') then
          Done <= '1';
        else
          Done <= '0';
        end if;
```

```
when others =>
        Done <= '0';
    end case;
  end process;

  O_Control_Signals: process(current_state,Start,Status_Signals)
  begin
    case(current_state) is
      when Step1 =>
        if(Start='1') then
          Control_Signals <= "11110100XX";
        else
          Control_Signals <= "00000100XX";
        end if;
      when Step2 =>
        if(Status_Signals(2)='1') then
          Control_Signals <= "00100000XX";
        else
          Control_Signals <= "0010001001";
        end if;
```

VHDL Code of Control of MExp (3/3)

```
    when Step3 =>
      if(Status_Signals(0)='1') then
        Control_Signals <= "000001010X";
      else
        Control_Signals <= "0000010100";
      end if;
    when Step4 =>
      if(Status_Signals(1)='1') then
        Control_Signals <= "000010XX11";
      else
        Control_Signals <= "000010XXX0";
      end if;
      when Step5 =>
        if(Status_Signals(0)='1') then
          Control_Signals <= "0000011110";
        else
          Control_Signals <= "0000011110";
        end if;
    when others =>
      Control_Signals <= "00000100XX";
  end case;
 end process;

end Behavioral;
```

Schematic of Mod Exp Control

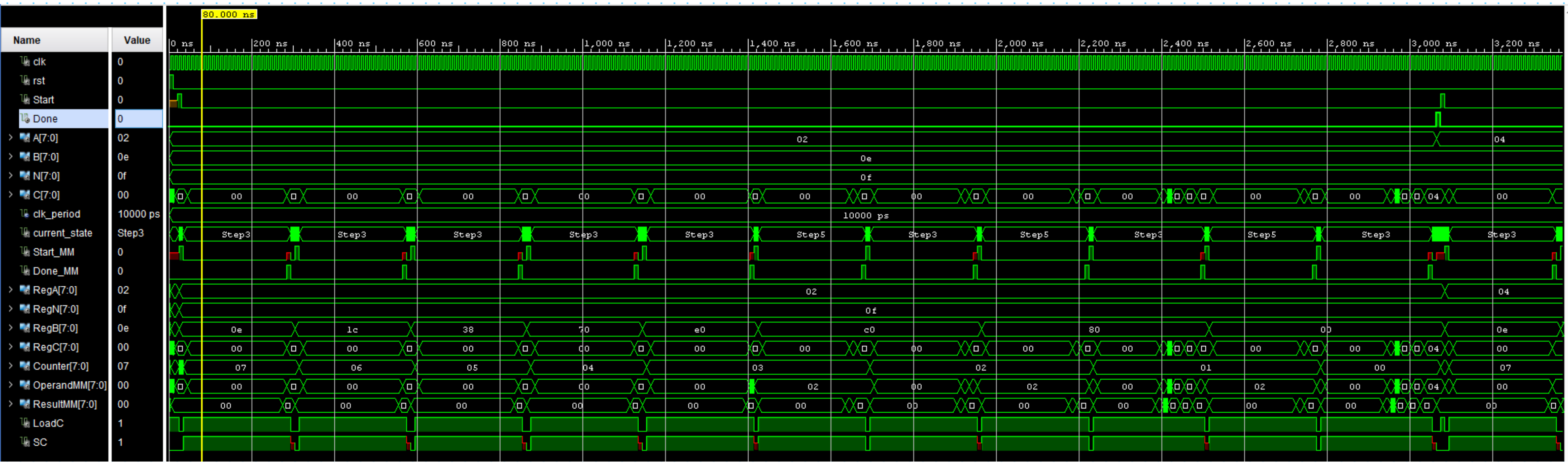


Testbench for MExp

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity MExp_tb is
end MExp_tb;
architecture Behavioral of MExp_tb is
component MExp is
    generic (K : integer := 8);
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           Start :  in STD_LOGIC;
           A : in STD_LOGIC_VECTOR (7 downto 0);
           B : in STD_LOGIC_VECTOR (7 downto 0);
           N : in STD_LOGIC_VECTOR (7 downto 0);
           C : out STD_LOGIC_VECTOR (7 downto 0);
           Done : out STD_LOGIC);
end component;
signal clk,rst,Start,Done : STD_LOGIC;
signal A,B,N,C : STD_LOGIC_VECTOR (7 downto 0);
constant clk_period : time := 10 ns;

begin
DUT: MExp
     generic map(8)
     Port map(clk,rst,Start,A,B,N,C,Done);
```

```
clk_process: process
  begin
    clk <= '0';
    wait for clk_period/2;  --for 0.5 ns signal is '0'.
    clk <= '1';
    wait for clk_period/2;  --for next 0.5 ns signal is '1'.
  end process;

Input_Application: process
  begin
    A <= "00000010";    B <= "00001110";
    N <= "00001111";    rst <= '1';
    wait for 10 ns;     rst <= '0';
    wait for 10 ns;     Start <= '1';
    wait for 10 ns;     Start <= '0';
    wait until Done='1';
    for i in 1 to 12 loop
      A <= C;
      wait for 10 ns;       Start <= '1';
      wait for 10 ns;       Start <= '0';
      wait until Done='1';
    end loop;
    wait;
  end process;
end Behavioral;
```

Test Waveform of MExp



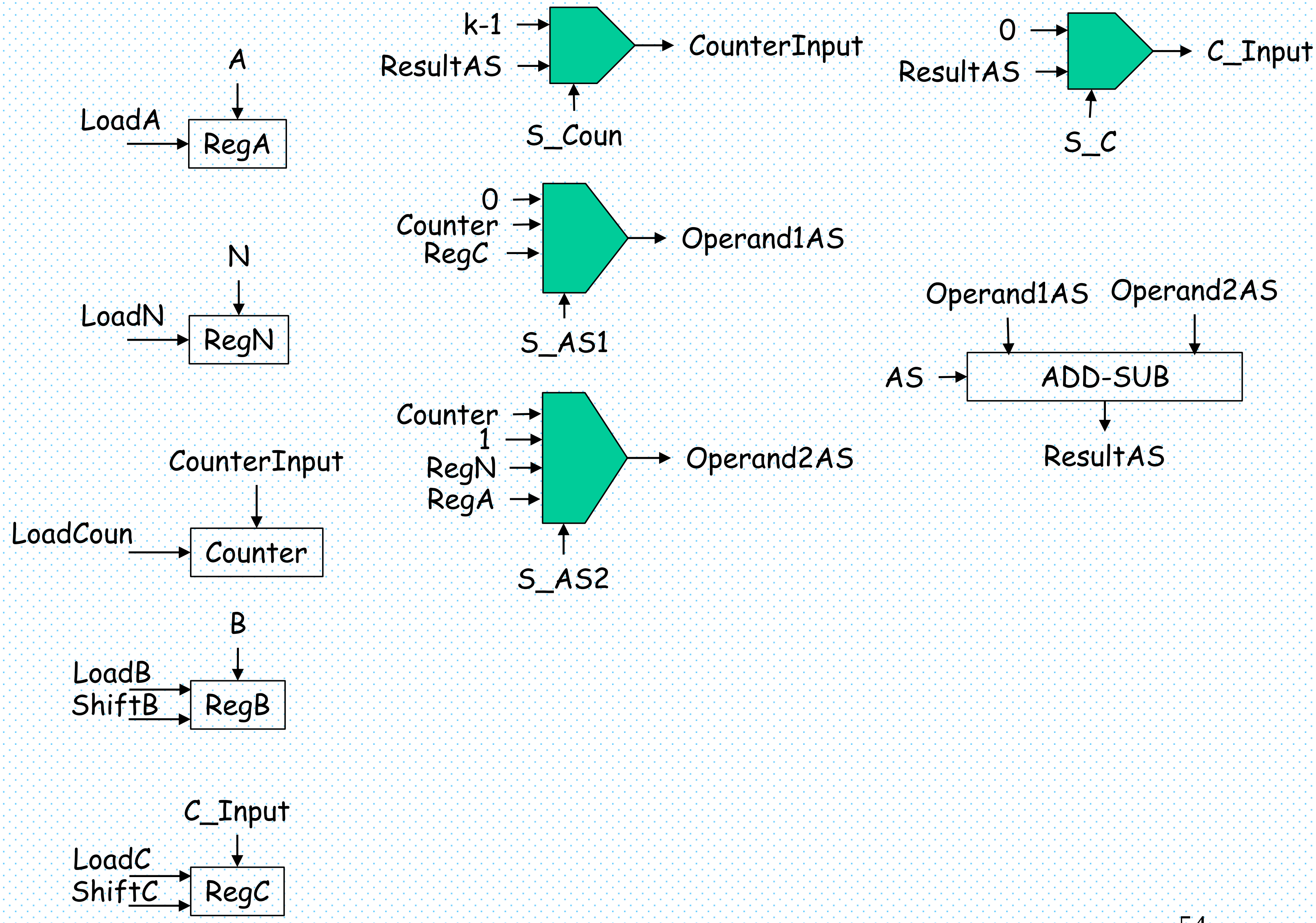
Implementation on Kintex7 xcku035-ffva1156-1-c Timing Results

- Min Clock Period 20 ns
- Max Clock Frequency 50 MHz
- Average number of clock cycles for one 128-bit ModMul 449
- Average number of clock cycles for one 128-bit ModExp 86401
- Throughput of 128-bit ModExp 578.7 bps
- One 128-bit RSA encryption 1.73 ms

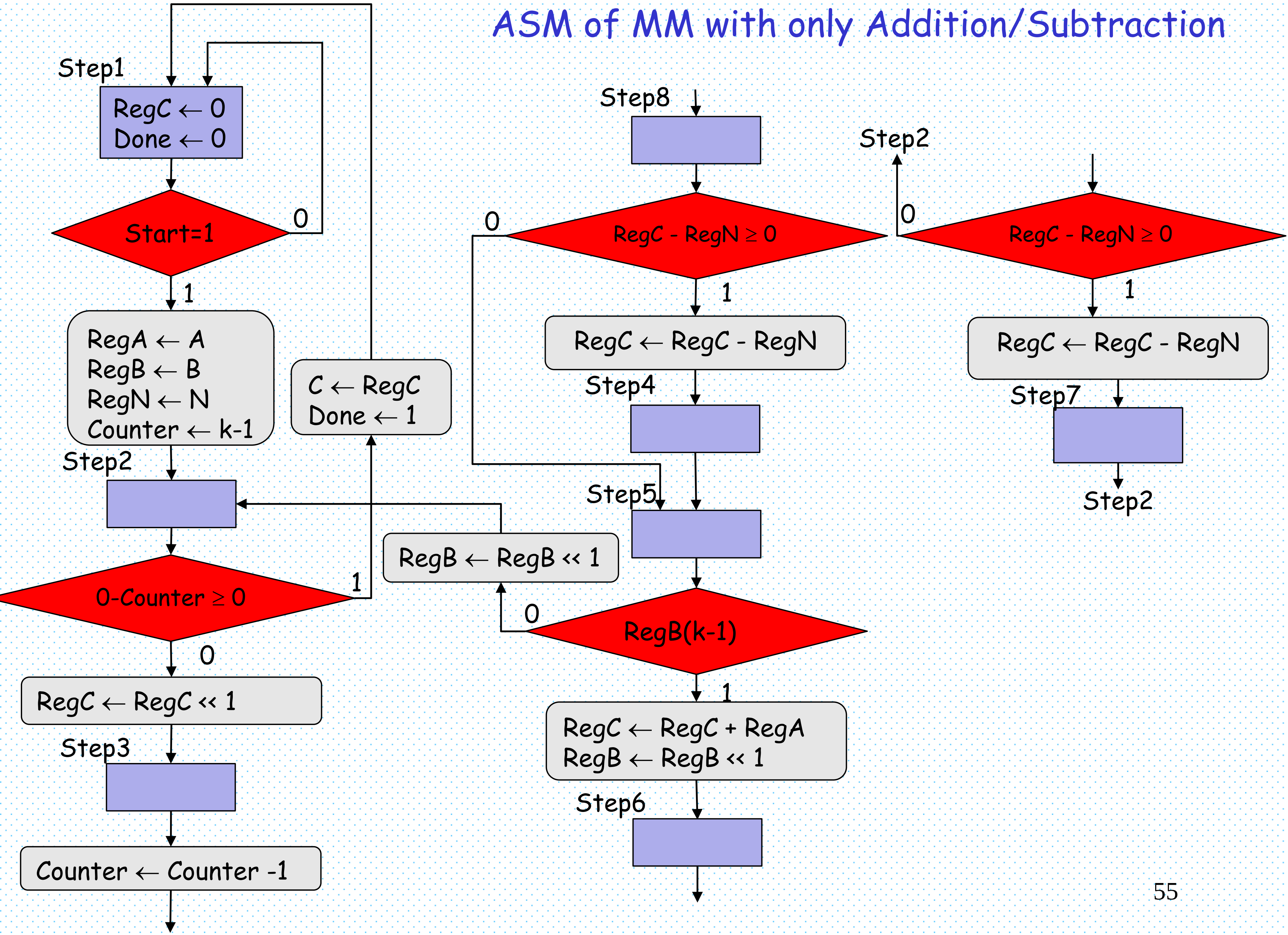
Implementation Area Results

Name	CLB LUTs (203128)	CLB Registers (406256)	CARRY 8 (30300)	CLB (30300)	LUT as Logic (203128)	LUT Flip Flop Pairs (203128)	Bonded IOB (520)	HPIOB (416)	HRIO (1...	GLOBAL CLOCK BUFFERs (480)
MExp	1682	1420	20	429	1682	662	516	412	104	1
Cont (Control_Mexp)	269	5	0	112	269	5	0	0	0	0
DP (Data_Path_MExp)	1413	1415	20	413	1413	506	0	0	0	0
Comp (Compare_Eq)	0	0	6	6	0	0	0	0	0	0
Mod_Mul (MM)	1184	647	14	244	1184	377	0	0	0	0
Cont (Control)	856	7	0	197	856	7	0	0	0	0
DP (Data_Path)	338	640	14	231	338	8	0	0	0	0
MUX_Operand1AS (MUX_2_to_1)	123	0	0	69	123	0	0	0	0	0
RegisterA (Load_Shift_Reg_4)	0	128	0	78	0	0	0	0	0	0
RegisterB (Load_Shift_Reg_5)	0	128	0	50	0	0	0	0	0	0
RegisterC (Load_Shift_Reg_6)	0	128	0	68	0	0	0	0	0	0
RegisterCounter (Load_Shift_Reg_7)	0	128	0	69	0	0	0	0	0	0
RegisterN (Load_Shift_Reg_8)	215	128	14	58	215	0	0	0	0	0
RegisterA (Load_Shift_Reg)	0	128	0	46	0	0	0	0	0	0
RegisterB (Load_Shift_Reg_0)	0	128	0	38	0	0	0	0	0	0
RegisterC (Load_Shift_Reg_1)	0	256	0	108	0	0	0	0	0	0
RegisterCounter (Load_Shift_Reg_2)	229	128	0	37	229	127	0	0	0	0
RegisterN (Load_Shift_Reg_3)	0	128	0	56	0	0	0	0	0	0

Data Path of MM with only Addition/Subtraction



ASM of MM with only Addition/Subtraction



Control Signals of MM with only Addition/Subtraction

		Status Signals		Control Signals											
State	**Start**	**ResultAS(k)**	**RegB (k-1)**	**Load A**	**Load N**	**Load Coun**	**Load B**	**Shift B**	**Load C**	**Shift C**	**S Coun**	**S AS1**	**S AS2**	**S C**	**AS**
Step1	0	X	X	0	0	0	0	0	1	0	0	X	X	0	X
Step1	1	X	X	1	1	1	1	0	1	0	0	X	X	0	X
Step2	X	0	X	0	0	1	0	0	0	1	1	0	00	X	1
Step2	X	1	X	0	0	1	0	0	0	0	1	X	X	X	X
Step3	X	0	X	0	0	0	0	0	0	0	X	0	00	X	1
Step3	X	0	X	0	0	0	0	0	1	0	X	1	01	1	1
Step3	X	1	X	0	0	0	0	0	1	0	X	1	01	1	1
Step4	X	X	X	0	0	0	0	0	0	0	X	X	X	X	X
Step5	X	X	0	0	0	0	0	1	0	0	X	X	X	X	X
Step5	X	X	1	0	0	0	0	1	1	0	X	1	10	1	0
Step6	X	0	X	0	0	0	0	0	0	0	X	X	X	X	X
Step6	X	0	X	0	0	0	0	0	1	0	X	1	11	1	1
Step6	X	1	X	0	0	0	0	0	1	0	X	1	11	1	1
Step7	X	X	X	0	0	0	0	0	0	0	X	X	X	X	X

VHDL Code of Data Path (1/3)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity Data_Path is
    generic (K : integer := 8);
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           A : in STD_LOGIC_VECTOR (K-1 downto 0);
           B : in STD_LOGIC_VECTOR (K-1 downto 0);
           N : in STD_LOGIC_VECTOR (K-1 downto 0);
           Control_Signals : in STD_LOGIC_VECTOR (13 downto 0);
           Status_Signals : out STD_LOGIC_VECTOR (1 downto 0);
           C : out STD_LOGIC_VECTOR (K-1 downto 0));
end Data_Path;

architecture Behavioral of Data_Path is
component Load_Shift_Reg is
    generic (K : integer := 8);
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           Data_In : in STD_LOGIC_VECTOR (K-1 downto 0);
           Data_Out : out STD_LOGIC_VECTOR (K-1 downto 0);
           Load : in STD_LOGIC;
           Shift : in STD_LOGIC);
end component;
component MUX_2_to_1 is
    generic (K : integer := 8);
    Port ( S : in STD_LOGIC;
           I0 : in STD_LOGIC_VECTOR (K-1 downto 0);
           I1 : in STD_LOGIC_VECTOR (K-1 downto 0);
           Y : out STD_LOGIC_VECTOR (K-1 downto
0));
end component;
component MUX_4_to_1 is
    generic (K : integer := 8);
    Port ( S : in STD_LOGIC_VECTOR (1 downto 0);
           I0 : in STD_LOGIC_VECTOR (K-1 downto 0);
           I1 : in STD_LOGIC_VECTOR (K-1 downto 0);
           I2 : in STD_LOGIC_VECTOR (K-1 downto 0);
           I3 : in STD_LOGIC_VECTOR (K-1 downto 0);
           Y : out STD_LOGIC_VECTOR (K-1 downto
0));
end component;
component ADD_SUB is
    generic (K : integer := 8);
    Port ( A : in STD_LOGIC_VECTOR (K-1 downto
0);
           B : in STD_LOGIC_VECTOR (K-1 downto 0);
           Sum : out STD_LOGIC_VECTOR (K downto
0);
           AS : in STD_LOGIC);
end component;
```

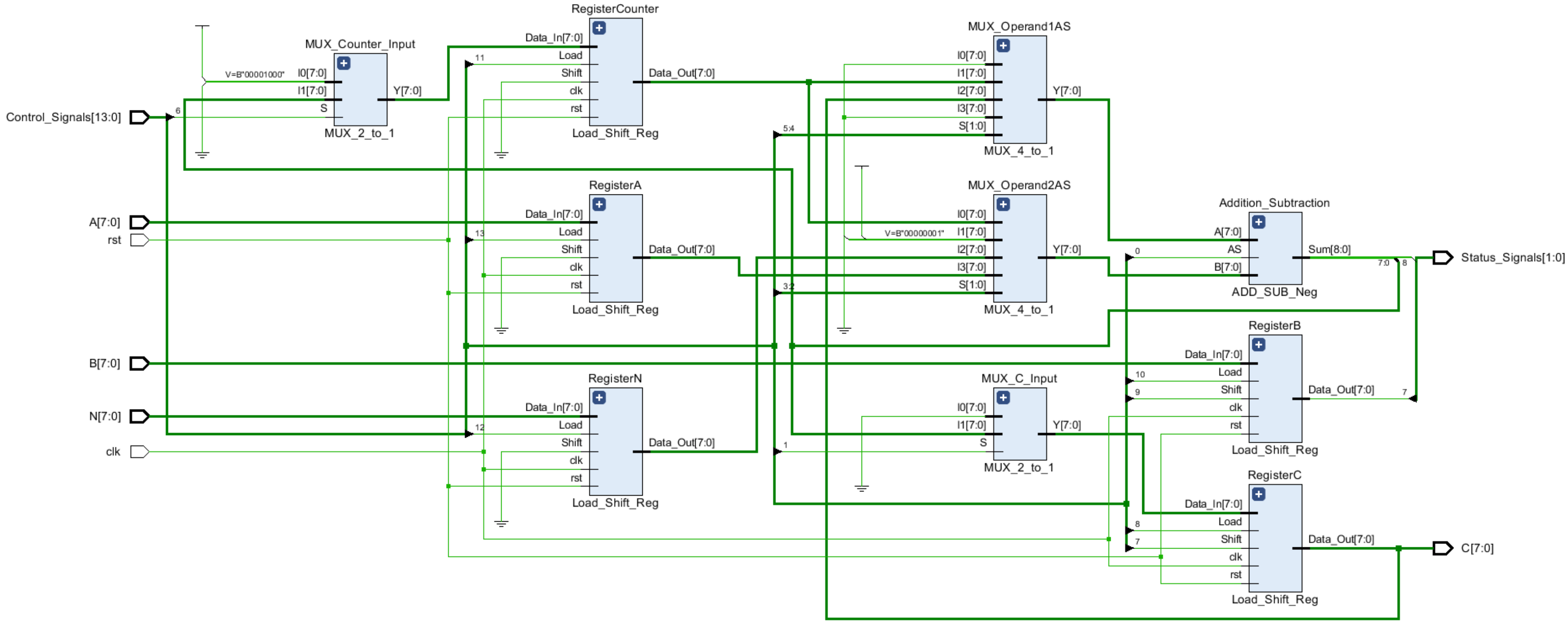
VHDL Code of Data Path (2/3)

```
signal LoadA,LoadN,LoadCoun,LoadB,ShiftB,LoadC,ShiftC,SCoun,
SC,AS : std_logic;
signal SAS1,SAS2 : STD_LOGIC_VECTOR (1 downto 0);
signal RegA, RegN, RegB, Counter_Input, Counter, C_Input,
RegC, temp_k, Operand1AS, temp_1, Operand2AS :
STD_LOGIC_VECTOR (K-1 downto 0);
signal ResultAS : STD_LOGIC_VECTOR (K downto 0);

begin
LoadA <= Control_Signals(13);
LoadN <= Control_Signals(12);
LoadCoun <= Control_Signals(11);
LoadB <= Control_Signals(10);
ShiftB <= Control_Signals(9);
LoadC <= Control_Signals(8);
ShiftC <= Control_Signals(7);
SCoun <= Control_Signals(6);
SAS1 <= Control_Signals(5 downto 4);
SAS2 <= Control_Signals(3 downto 2);
SC <= Control_Signals(1);
AS <= Control_Signals(0);

RegisterA: Load_Shift_Reg
        generic map(K)
        Port map(clk,rst,A,RegA,LoadA,'0');
RegisterN: Load_Shift_Reg
        generic map(K)
        Port map(clk,rst,N,RegN,LoadN,'0');
RegisterCounter: Load_Shift_Reg
        generic map(K)
        Port
map(clk,rst,Counter_Input,Counter,LoadCoun,'0');
RegisterB: Load_Shift_Reg
        generic map(K)
        Port map(clk,rst,B,RegB,LoadB,ShiftB);
RegisterC: Load_Shift_Reg
        generic map(K)
        Port
map(clk,rst,C_Input,RegC,LoadC,ShiftC);

temp_k <= conv_std_logic_vector(K, K);

MUX_Counter_Input: MUX_2_to_1
                generic map(K)
                Port map(SCoun,temp_k,ResultAS(K-1
downto 0),Counter_Input);
MUX_C_Input: MUX_2_to_1
                generic map(K)
                Port map(SC,(others => '0'),ResultAS(K-
1 downto 0),C_Input);
```

VHDL Code of Data Path (3/3)

```
temp_1(0) <= '1';
temp_1(K-1 downto 1) <= (others => '0');

MUX_Operand1AS: MUX_4_to_1
        generic map(K)
        Port map(SAS1,(others =>'0'),Counter,RegC, (others =>'0'),Operand1AS);
MUX_Operand2AS: MUX_4_to_1
        generic map(K)
        Port map(SAS2, Counter,temp_1,RegN,RegA,Operand2AS);

Addition_Subtraction: ADD_SUB
         generic map(K)
         Port map( Operand1AS,Operand2AS,ResultAS,AS);

Status_Signals(0) <= ResultAS(K);
Status_Signals(0) <= RegB(K-1);
C <= RegC;

end Behavioral;
```

Schematic of Data Path with only Addition/Subtraction



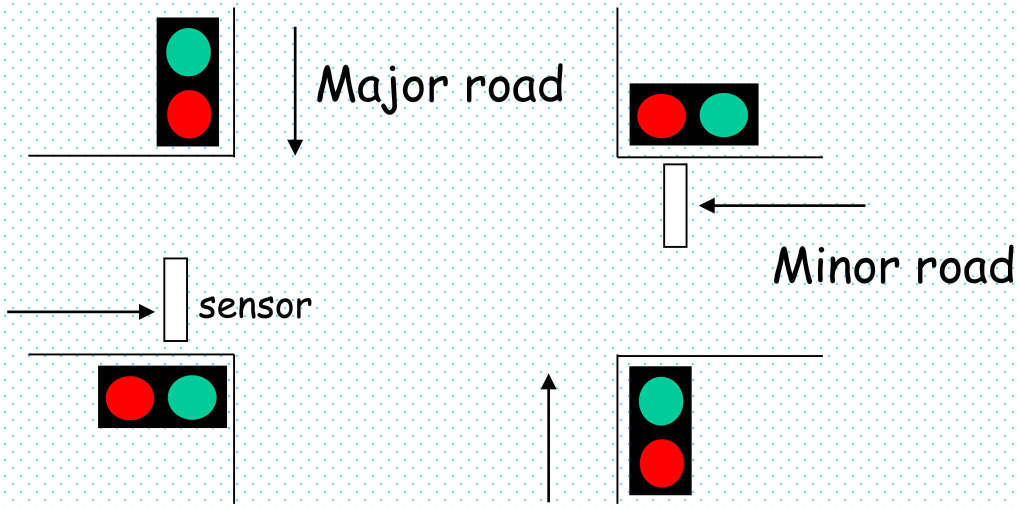
Implementation on Kintex7 xcku035-ffva1156-1-c Timing Results

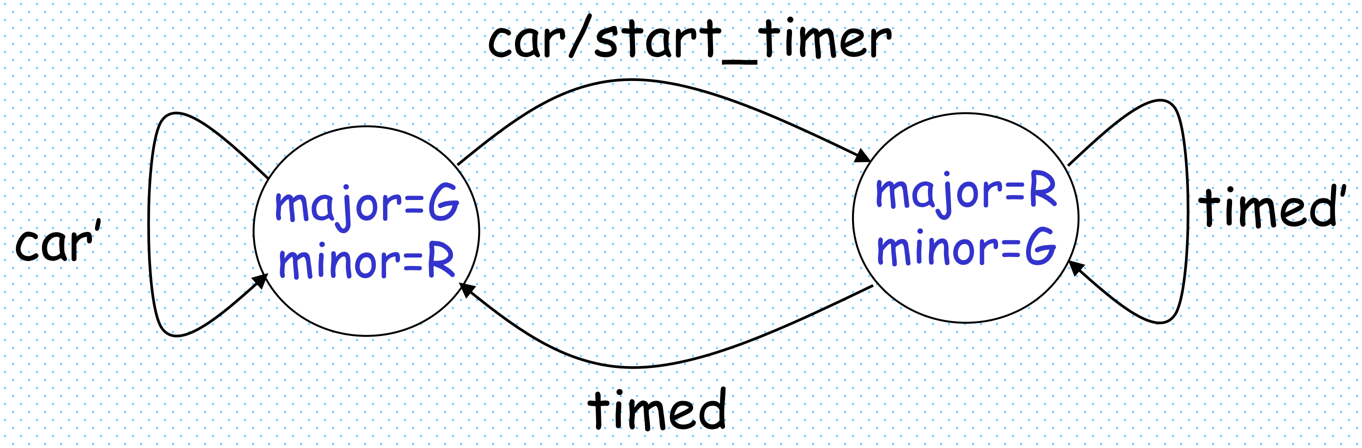
- Min Clock Period 18 ns
- Max Clock Frequency 55.5 MHz
- Average number of clock cycles for one 128-bit ModMul 449
- Average number of clock cycles for one 128-bit ModExp 86401
- Throughput of 128-bit ModExp 578.7 bps
- One 128-bit RSA encryption 1.73 ms

Implementation Area Results

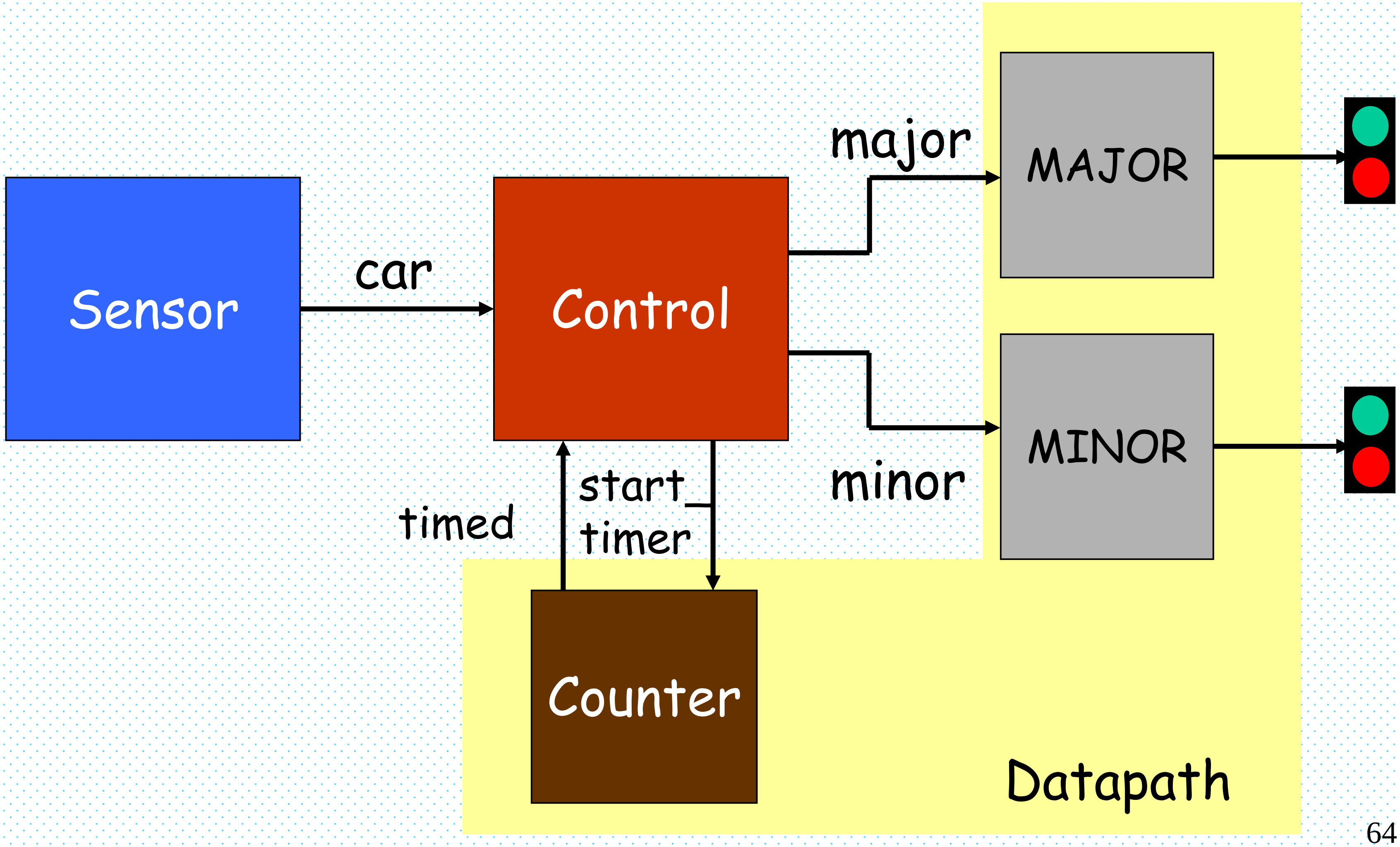
Name	CLB LUTs (203128)	CLB Registers (406256)	CARRY 8 (30300)	CLB (30300)	LUT as Logic (203128)	LUT Flip Flop Pairs (203128)	Bonded IOB (520)	HPIOB (416)	HRIO (1...	GLOBAL CLOCK BUFFERs (480)
MExp	1682	1420	20	429	1682	662	516	412	104	1
Cont (Control_Mexp)	269	5	0	112	269	5	0	0	0	0
DP (Data_Path_MExp)	1413	1415	20	413	1413	506	0	0	0	0
Comp (Compare_Eq)	0	0	6	6	0	0	0	0	0	0
Mod_Mul (MM)	1184	647	14	244	1184	377	0	0	0	0
Cont (Control)	856	7	0	197	856	7	0	0	0	0
DP (Data_Path)	338	640	14	231	338	8	0	0	0	0
MUX_Operand1AS (MUX_2_to_1)	123	0	0	69	123	0	0	0	0	0
RegisterA (Load_Shift_Reg_4)	0	128	0	78	0	0	0	0	0	0
RegisterB (Load_Shift_Reg_5)	0	128	0	50	0	0	0	0	0	0
RegisterC (Load_Shift_Reg_6)	0	128	0	68	0	0	0	0	0	0
RegisterCounter (Load_Shift_Reg_7)	0	128	0	69	0	0	0	0	0	0
RegisterN (Load_Shift_Reg_8)	215	128	14	58	215	0	0	0	0	0
RegisterA (Load_Shift_Reg)	0	128	0	46	0	0	0	0	0	0
RegisterB (Load_Shift_Reg_0)	0	128	0	38	0	0	0	0	0	0
RegisterC (Load_Shift_Reg_1)	0	256	0	108	0	0	0	0	0	0
RegisterCounter (Load_Shift_Reg_2)	229	128	0	37	229	127	0	0	0	0
RegisterN (Load_Shift_Reg_3)	0	128	0	56	0	0	0	0	0	0

Example: Traffic Control

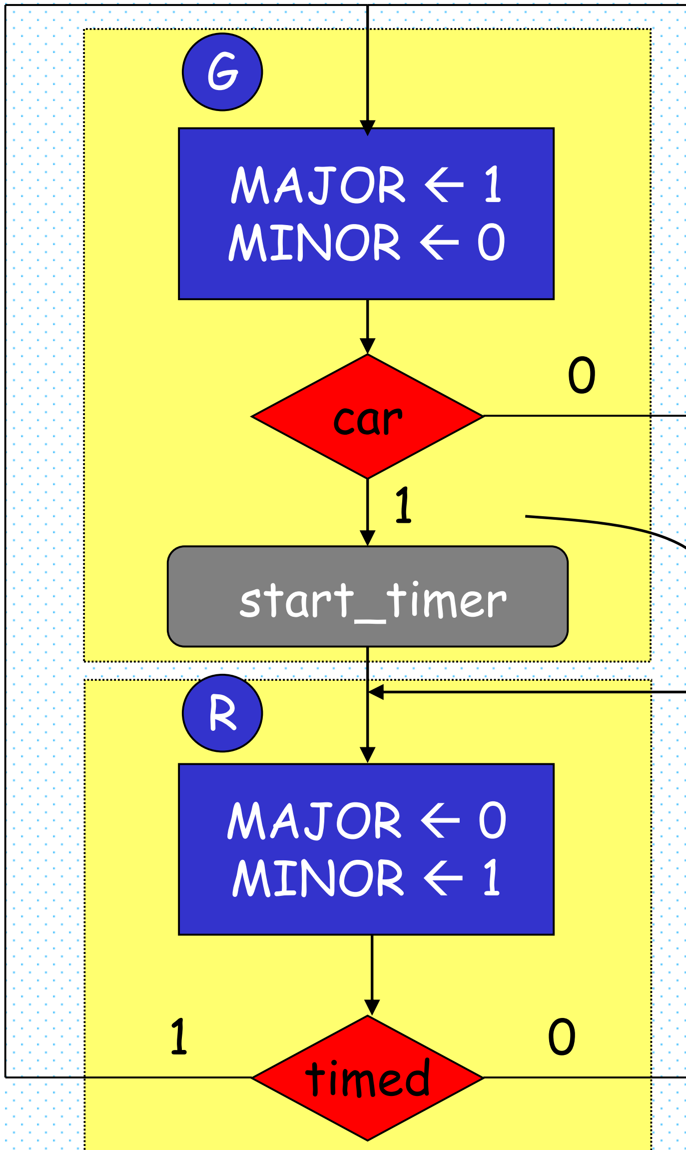




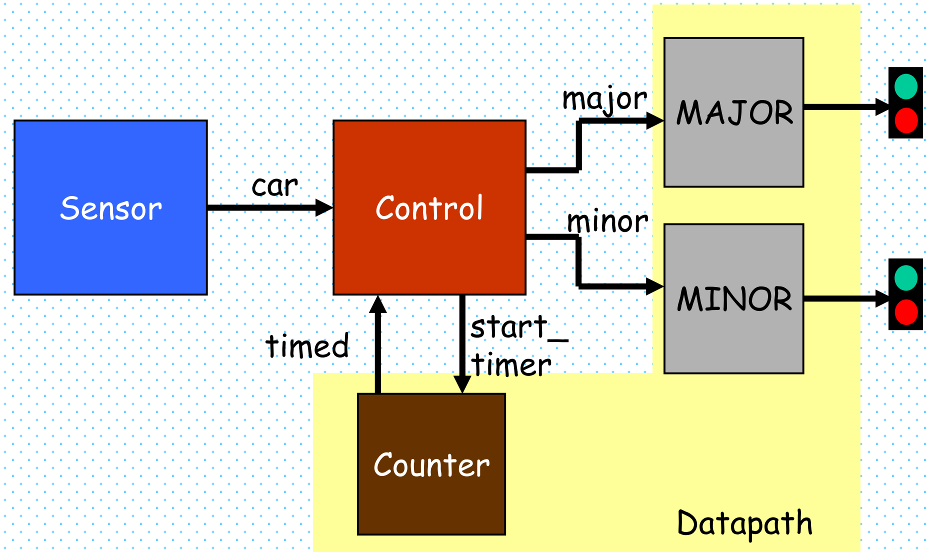
Datapath & Control



Example: ASM Chart



ASM Block



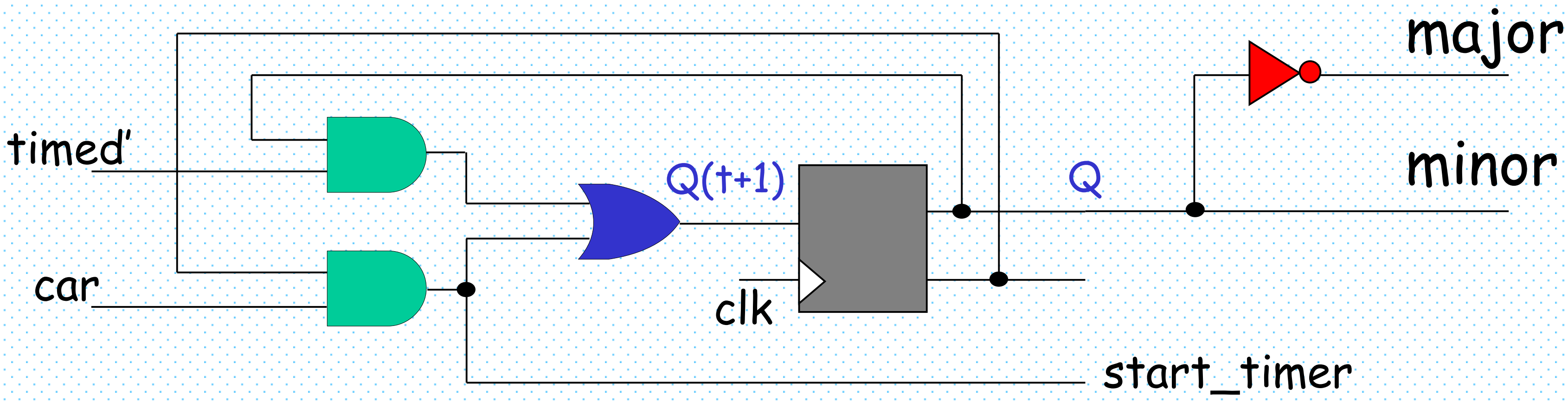
State Table

Current state	Input		Next state	Output
Q	car	timed	Q	start_timer
(G) 0	0	X	0	0
(G) 0	1	X	1	1
(R) 1	X	0	1	0
(R) 1	X	1	0	0

- Flip-flop input equation
  - $Q(t+1) = Q' \text{ car} + Q \text{ timed}'$
- Output equation
  - $\text{start\_timer} = Q' \text{ car}$

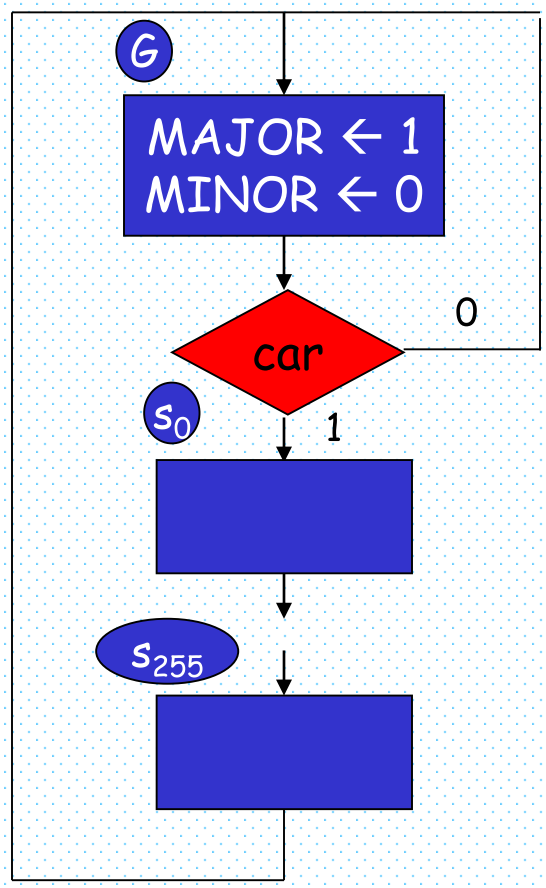
Circuit

- Flip-flop input equation
  - $Q(t+1) = Q' \, car + Q \, timed'$
- Output equation
  - $start\_timer = Q' \, car$



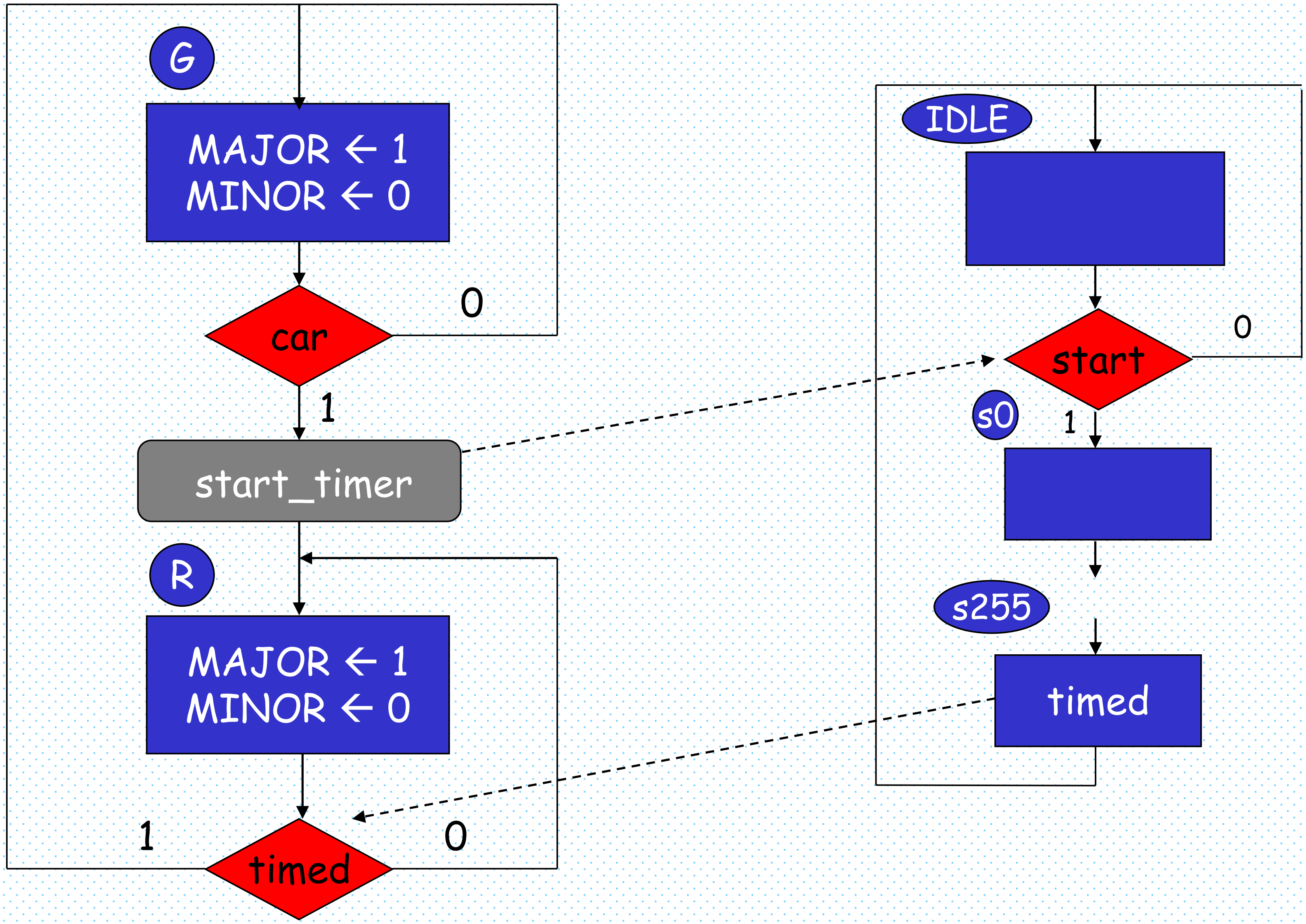
- When Q = 0 ➔ major = 1, minor = 0
- When Q = 1 ➔ major = 0, minor = 1

Traffic Controller with a Timer



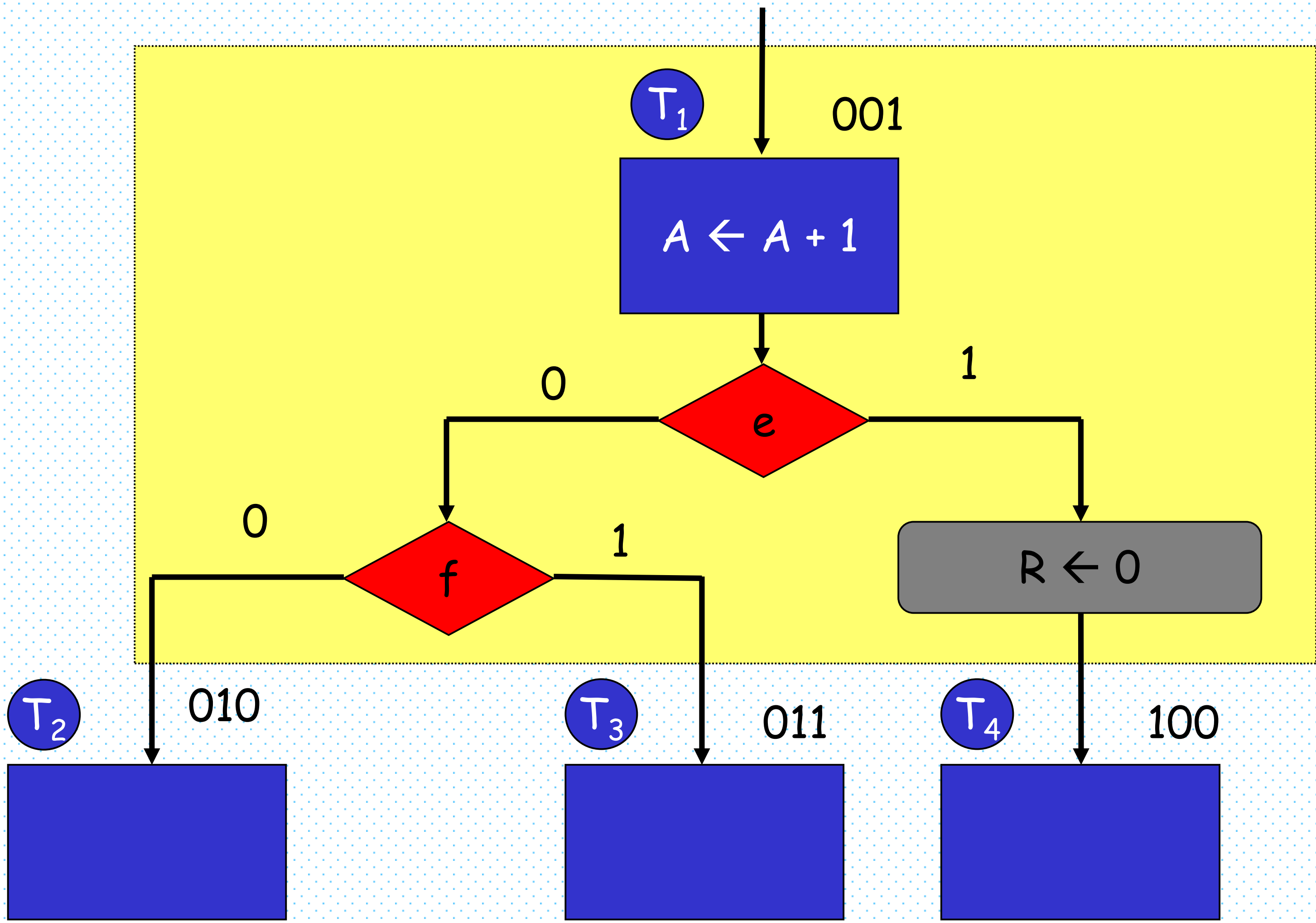
- There is an abundance of states.
- states from s_0 to s_{255} map to a simple counter
- we can just separate the traffic light controller from the timer.

Linked ASM Charts



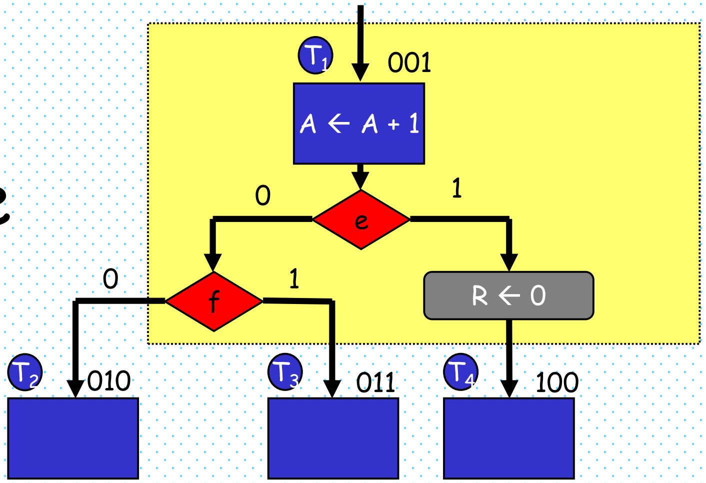
ASM Block

- A structure consisting of one state box and all the decision and conditional boxes associated with it.



ASM Block

- One input path, any number of exit paths
- Each ASM block describes the state of a system during one clock-pulse interval
  - The register operations within the state and conditional boxes in the example are executed with a common clock pulse when the system in this state (e.g. T_1 in the example)
  - The same clock pulse transfer the system controller to one of the next states (e.g. T_2, T_3, or T_4)

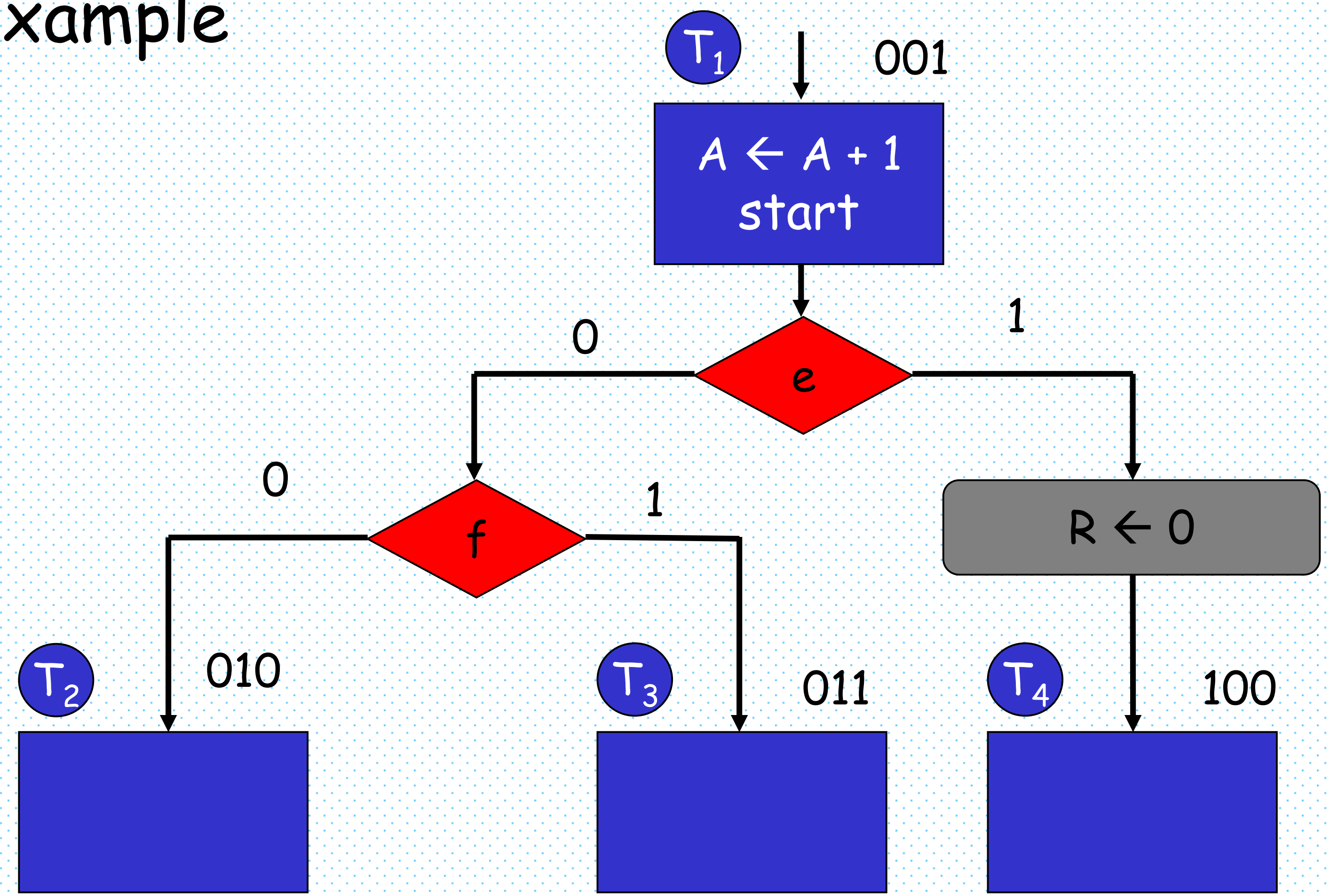


Timing Considerations 1/4

- The pulses of the common clock are applied to
  - registers in the datapath
  - all flip-flops in the control
- We can assume that inputs are also synchronized with the clock
  - Since they are the outputs of another circuit working with the same common clock.
  - Synchronous inputs

Timing Considerations 2/4

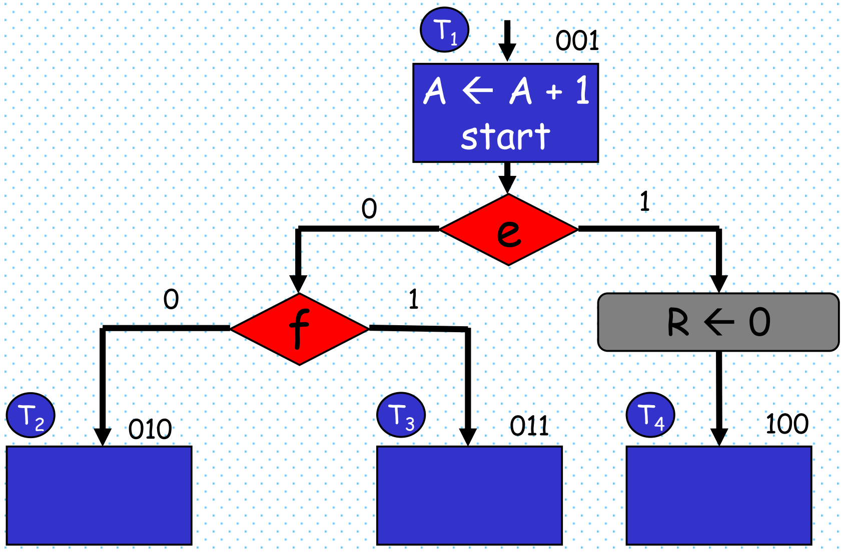
- Major difference between a conventional flow chart and ASM chart is in the time relations among the various operations
- Example



Timing Considerations 3/4

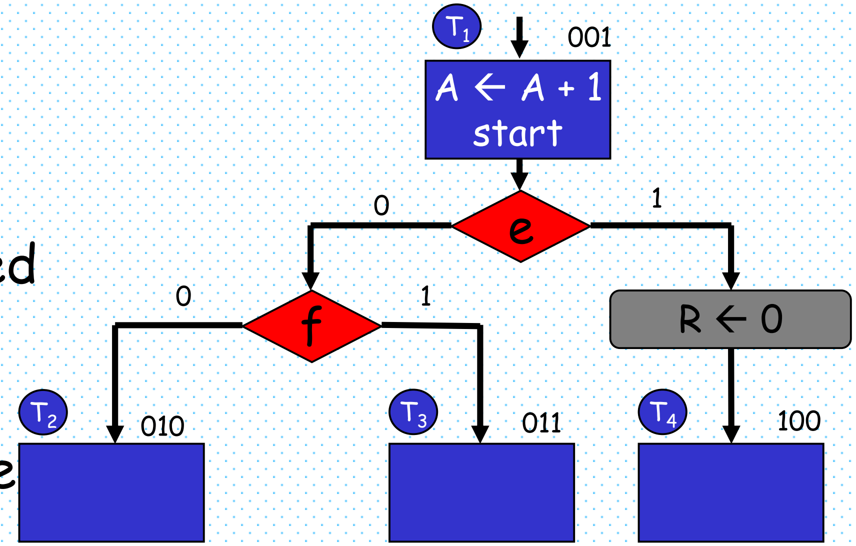
- If it were a conventional flowchart

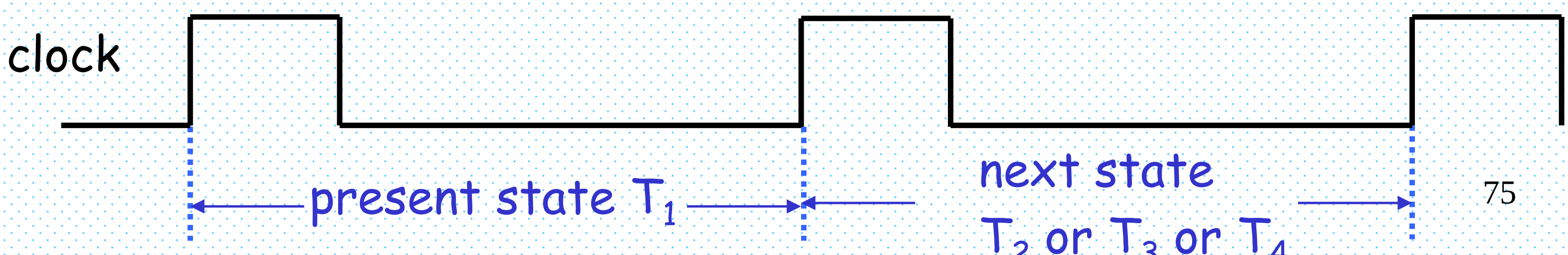
```
1. A ← A + 1
2. start = 1
3. if e = 1 then
     R ← 0
     next state is T4
4. else
     if f = 1 then
        next state is T3
     else
        next state is T2
```



Timing Considerations 4/4

- But, in ASM chart, interpretation is different
  - all operations in a block occur in synchronism with the clock
  - "start" is asserted in T_1
  - input signals "e" and "f" are checked in T_1
  - The following operations are executed simultaneously during the next positive edge of clock
    - $A \leftarrow A + 1$
    - $R \leftarrow 0$ (if $e = 1$)
    - control transfer to the next state



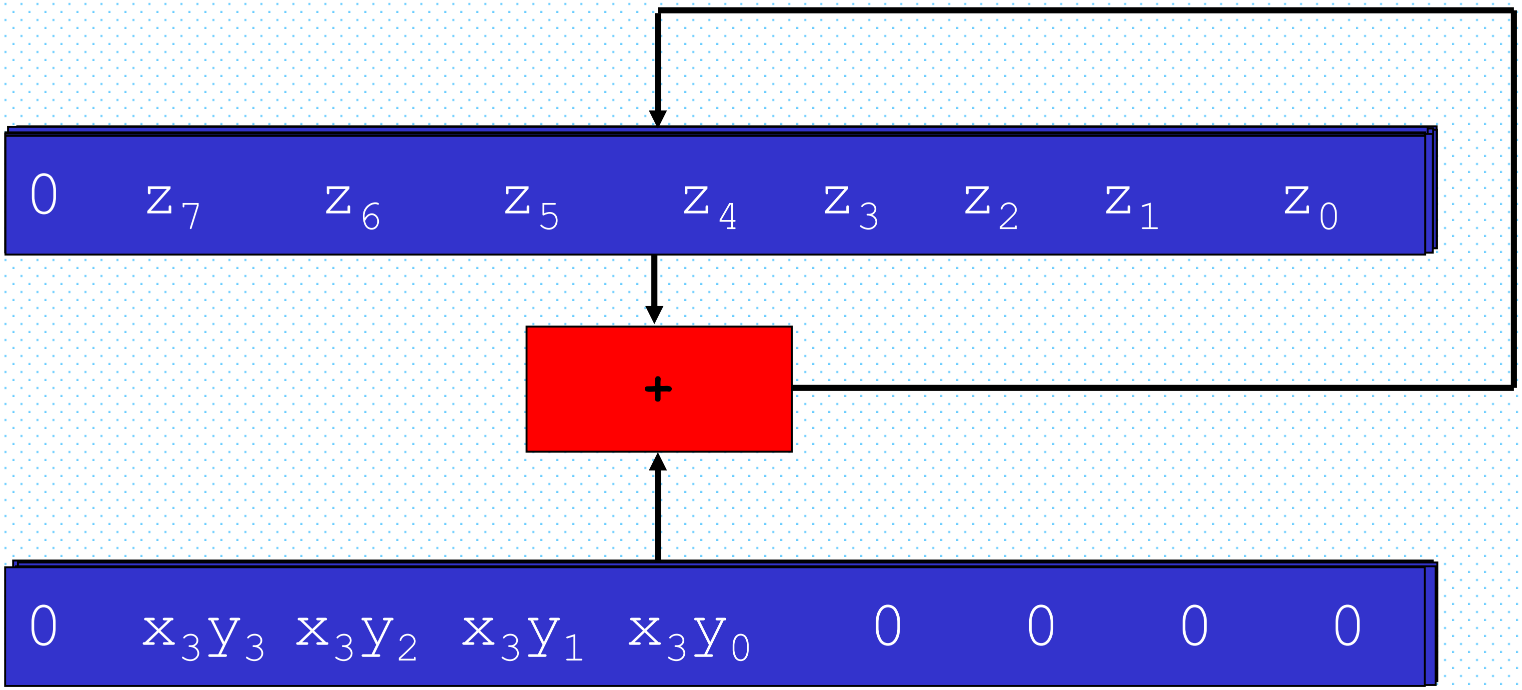


Example: Binary Multiplier

- Sequential multiplier
- Algorithm: successive additions and shifting

multiplicand					y_3	y_2	y_1	y_0
multiplier				$\times$	x_3	x_2	x_1	x_0
partial product					$x_0 y_3$	$x_0 y_2$	$x_0 y_1$	$x_0 y_0$
				$x_1 y_3$	$x_1 y_2$	$x_1 y_1$	$x_1 y_0$	
			$x_2 y_3$	$x_2 y_2$	$x_2 y_1$	$x_2 y_0$		
$+$		$x_2 y_3$	$x_3 y_2$	$x_3 y_1$	$x_3 y_0$			
	z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0
product								

Or



Numeric Example

C	A	Q	B	P
0	00000	00000	00000	000
0	00000	10011	10111	101
0	10111	10011	10111	101
0	01011	11001	10111	100
1	00010	11001	10111	100
0	10001	01100	10111	011
0	01000	10110	10111	010
0	00100	01011	10111	001
0	11011	01011	10111	001
0	01101	10101	10111	000

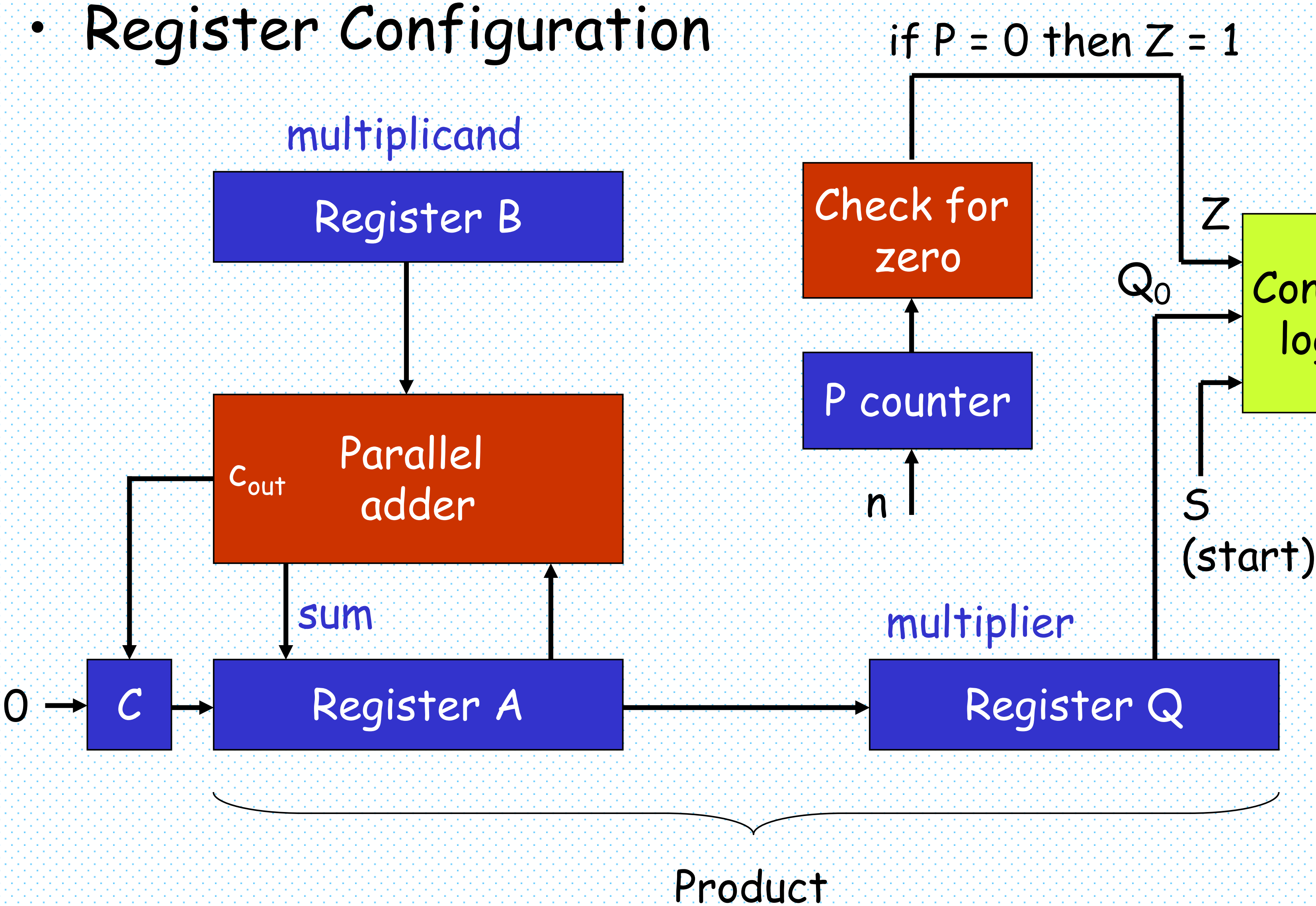
Y = 10111 = 23

X = 10011 = 19

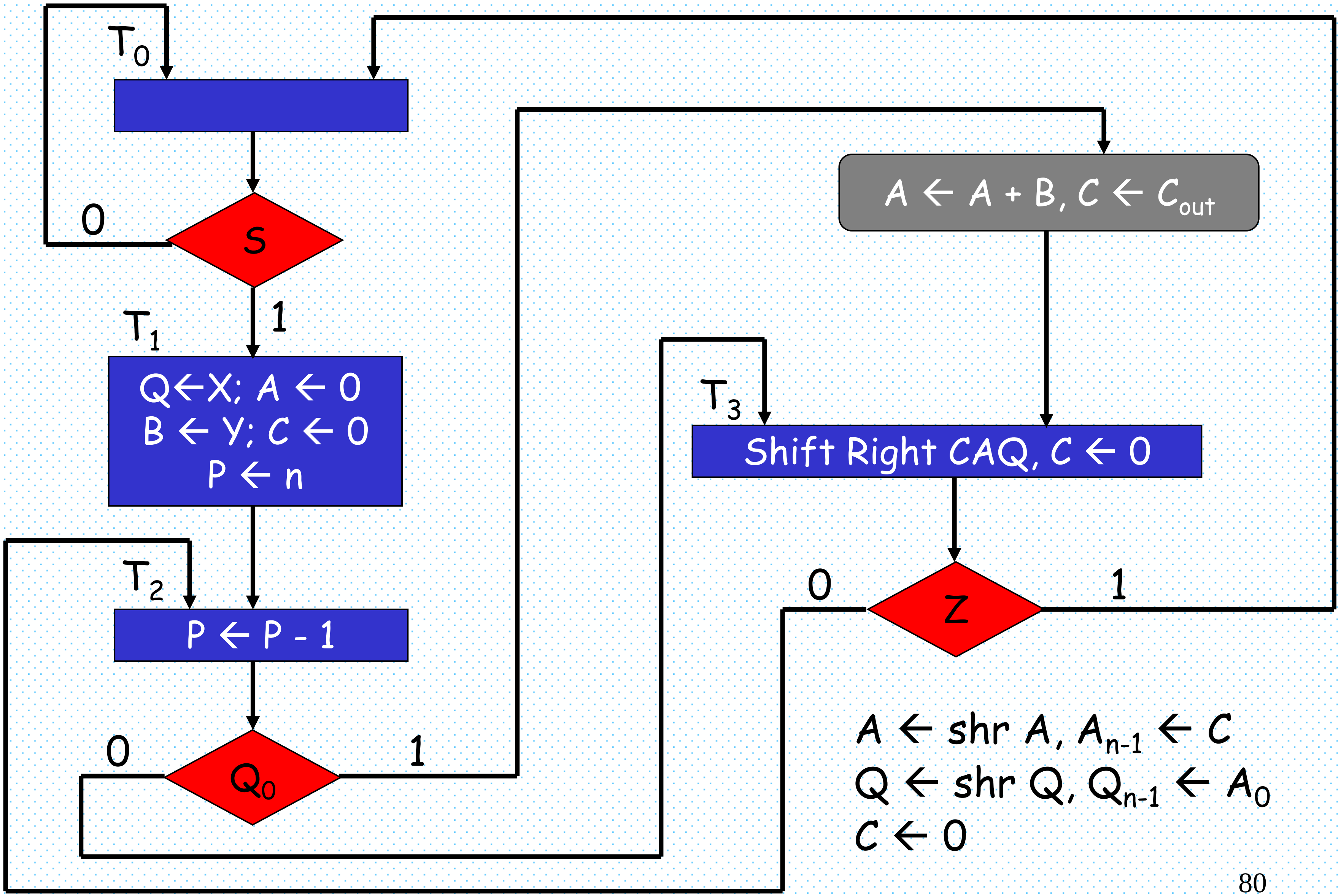
CAQ = 00110110101 = 437

Example: Binary Multiplier

- Register Configuration



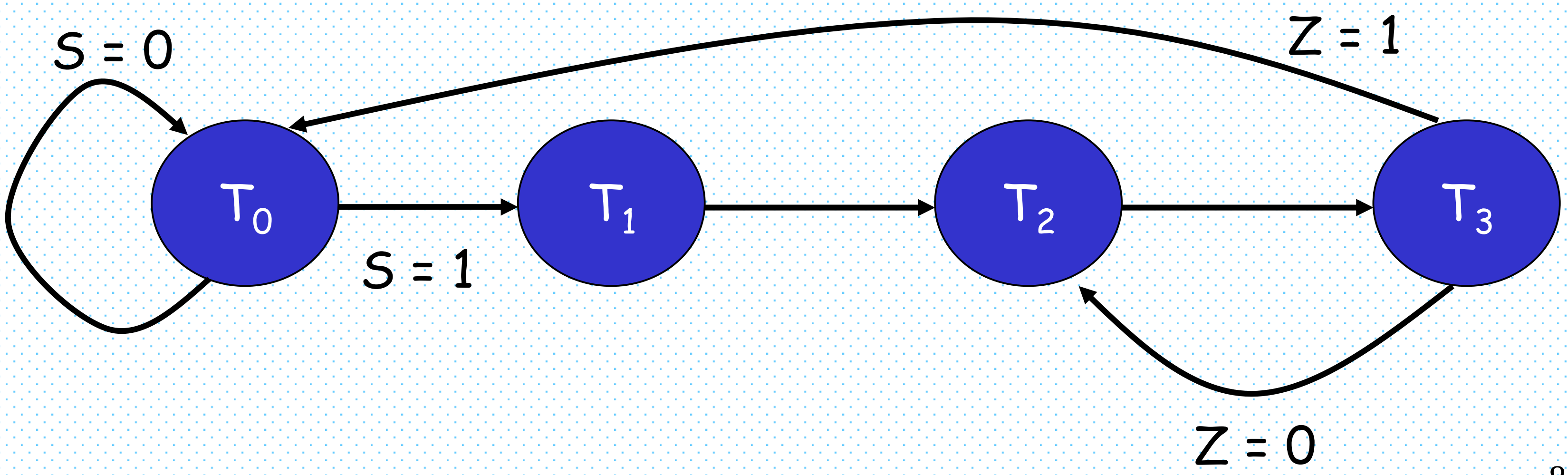
Example: ASM Chart



$A \leftarrow \text{shr } A,\ A_{n-1} \leftarrow C$
$Q \leftarrow \text{shr } Q,\ Q_{n-1} \leftarrow A_0$
$C \leftarrow 0$

Control Logic

- The design of a digital system can be divided into two parts:
  - design of the register transfer in the datapath
  - design of the control logic
    - sequential circuit design problem
    - state diagram approach can be used

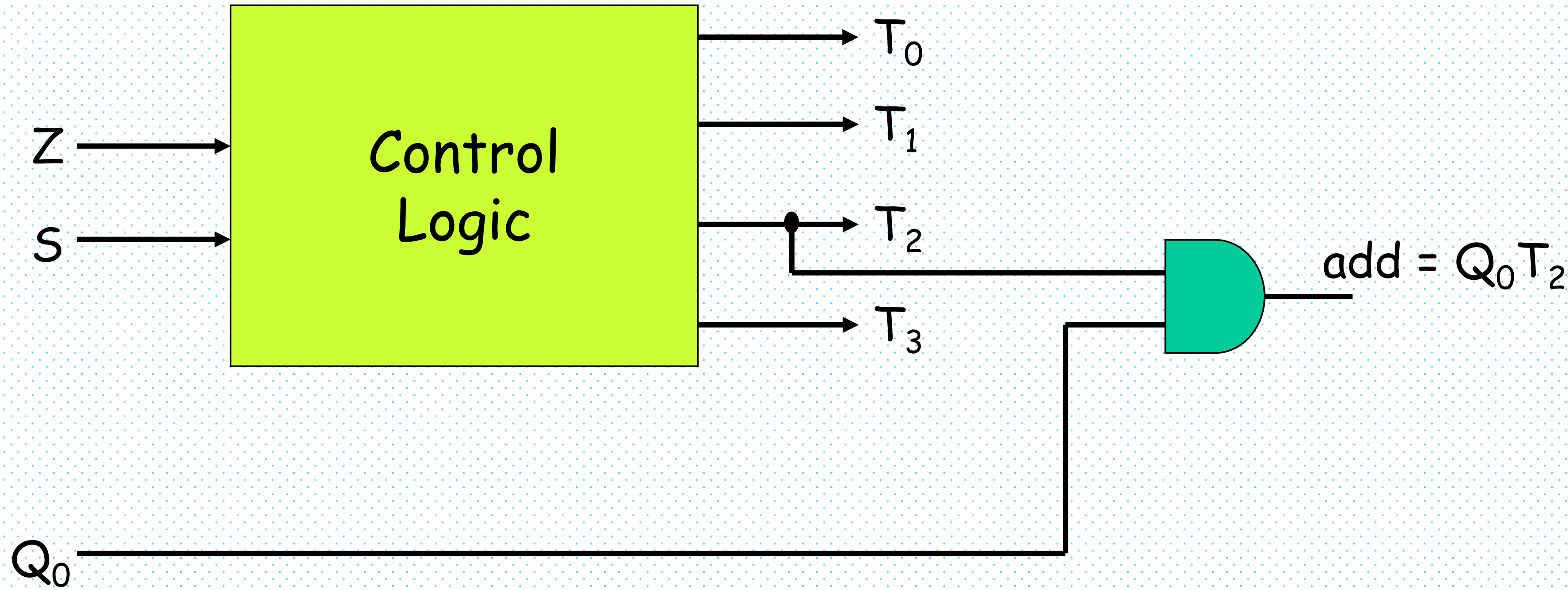


Designing Control Logic

- Two things we need to deal with
  1. Establish required sequence of states
     - specified in the state diagram
  2. provide signals to control the register operations
     - specified in the state and conditional boxes
     - These signals are
       - T_0: initial state
       - T_1: $A \leftarrow 0, C \leftarrow 0, P \leftarrow n, Q \leftarrow X, B \leftarrow Y$
       - T_2: $P \leftarrow P - 1$
         » if $(Q_0 = 1)$ then $(A \leftarrow A + B, C \leftarrow C_{out})$
       - T_3: shift right CAQ, $C \leftarrow 0$

Control Logic

- Block diagram



if (T_2 **AND** Q_0 = 1) **then** (A ← A + B, C ← C_{out})

Designing Control Logic

- State Table

Present state		Input		Next state		Outputs			
A_1	A_0	S	Z	A_1	A_0	T_0	T_1	T_2	T_3
0	0	0	X	0	0	1	0	0	0
0	0	1	X	0	1	1	0	0	0
0	1	X	X	1	0	0	1	0	0
1	0	X	X	1	1	0	0	1	0
1	1	X	0	1	0	0	0	0	1
1	1	X	1	0	0	0	0	0	1

$T_0 = A_1'A_0'$ $T_1 = A_1'A_0$ $T_2 = A_1A_0'$ $T_3 = A_1A_0$

Designing Control Logic

$T_0 = A_1'A_0'$ $T_1 = A_1'A_0$

$T_2 = A_1A_0'$ $T_3 = A_1A_0$

- With D flip-flops

A_1A_0 \ SZ	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	0	0	1
10	1	1	1	1

A_1A_0 \ SZ	00	01	11	10
00	0	0	1	1
01	0	0	0	0
11	0	0	0	0
10	1	1	1	1

$A_1(t+1) = A_0Z' + A_1A_0' + A_1'A_0$

$A_0(t+1) = SA_0' + A_1A_0'$

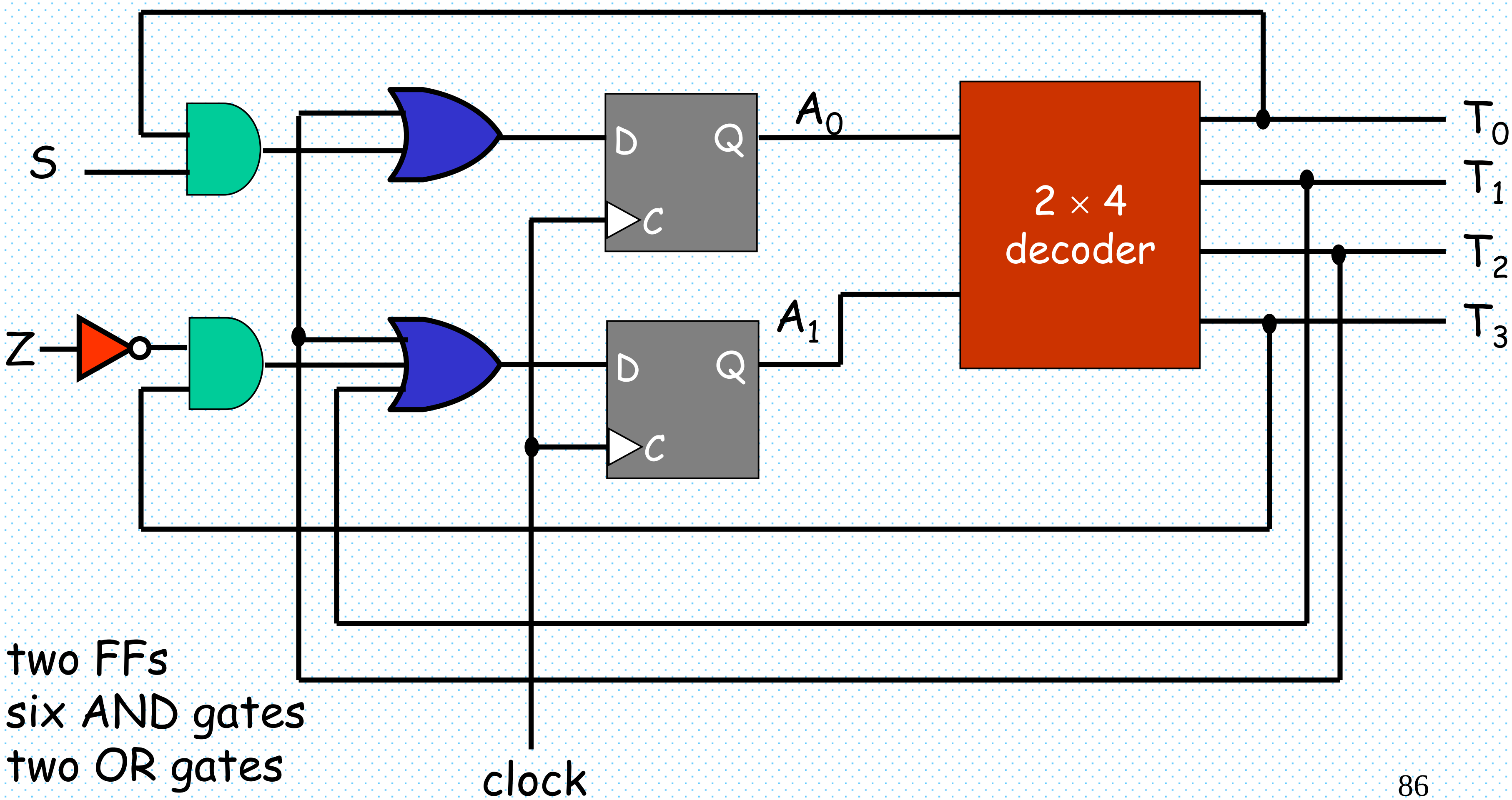
$A_1(t+1) = T_3Z' + T_2 + T_1$

$A_0(t+1) = ST_0 + T_2$

Control Logic

$A_1(t+1) = T_3Z' + T_2 + T_1$ $\qquad$ $A_0(t+1) = ST_0 + T_2$

$T_0 = A_1'A_0'$ $\quad$ $T_1 = A_1'A_0$ $\quad$ $T_2 = A_1A_0'$ $\quad$ $T_3 = A_1A_0$



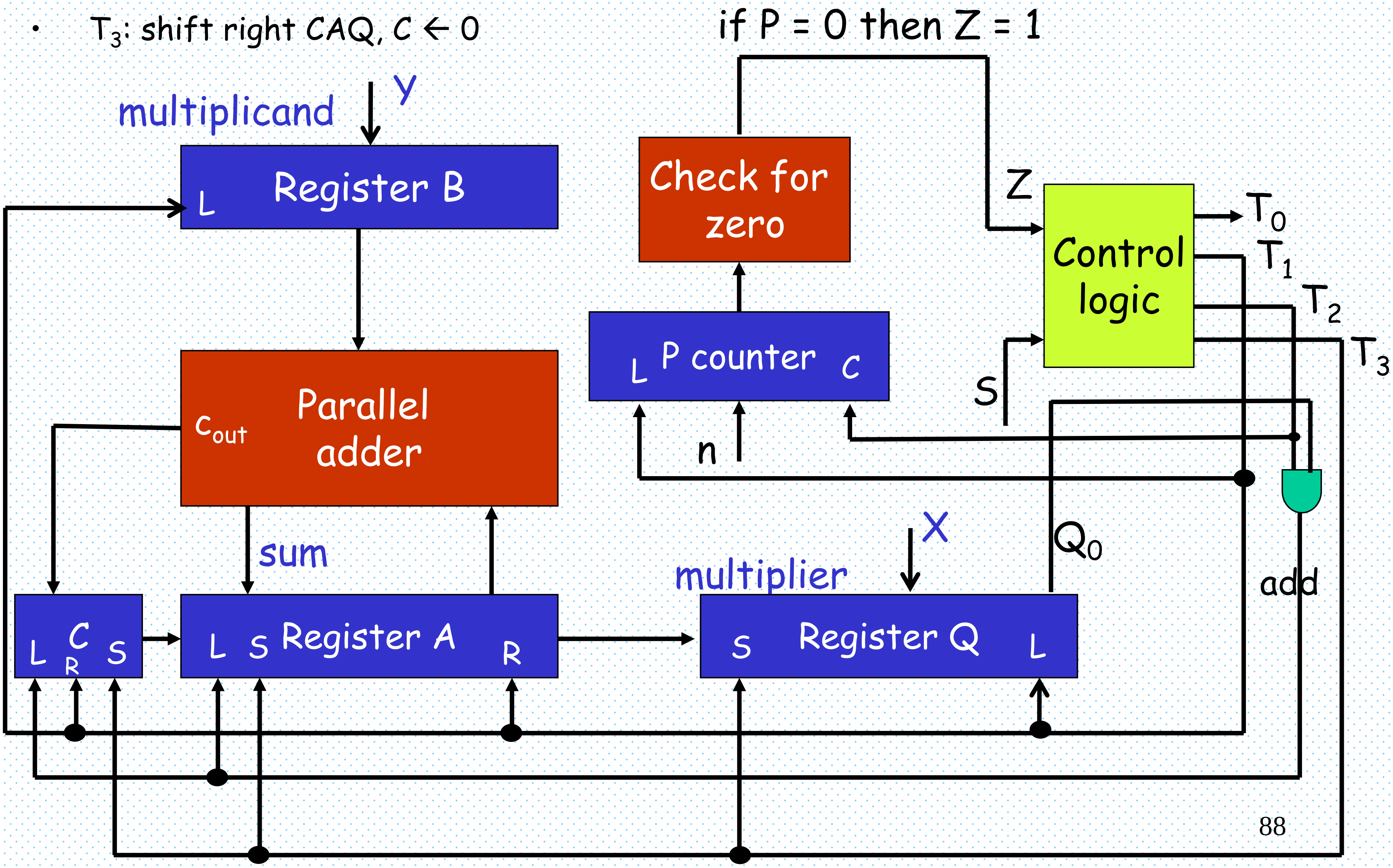
two FFs
six AND gates
two OR gates
one inverter

Reminder

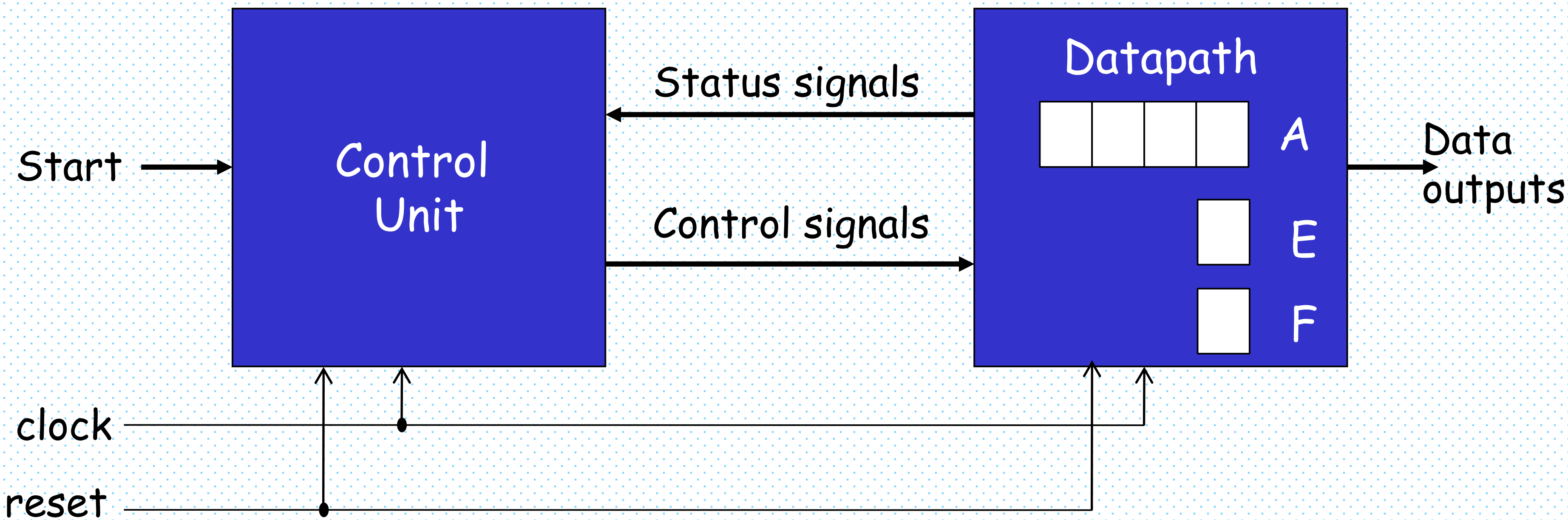
- T_0: initial state
- T_1: $A \leftarrow 0, C \leftarrow 0, P \leftarrow n$
- T_2: $P \leftarrow P - 1$
  - if $(Q_0 = 1)$ then $(A \leftarrow A + B, C \leftarrow C_{out})$
- T_3: shift right CAQ, $C \leftarrow 0$

Overall Circuit

- T_1: $A \leftarrow 0, C \leftarrow 0, P \leftarrow n$
- T_2: $P \leftarrow P - 1$
  - if (Q_0 = 1) then ($A \leftarrow A + B, C \leftarrow C_{out}$)
- T_3: shift right CAQ, $C \leftarrow 0$

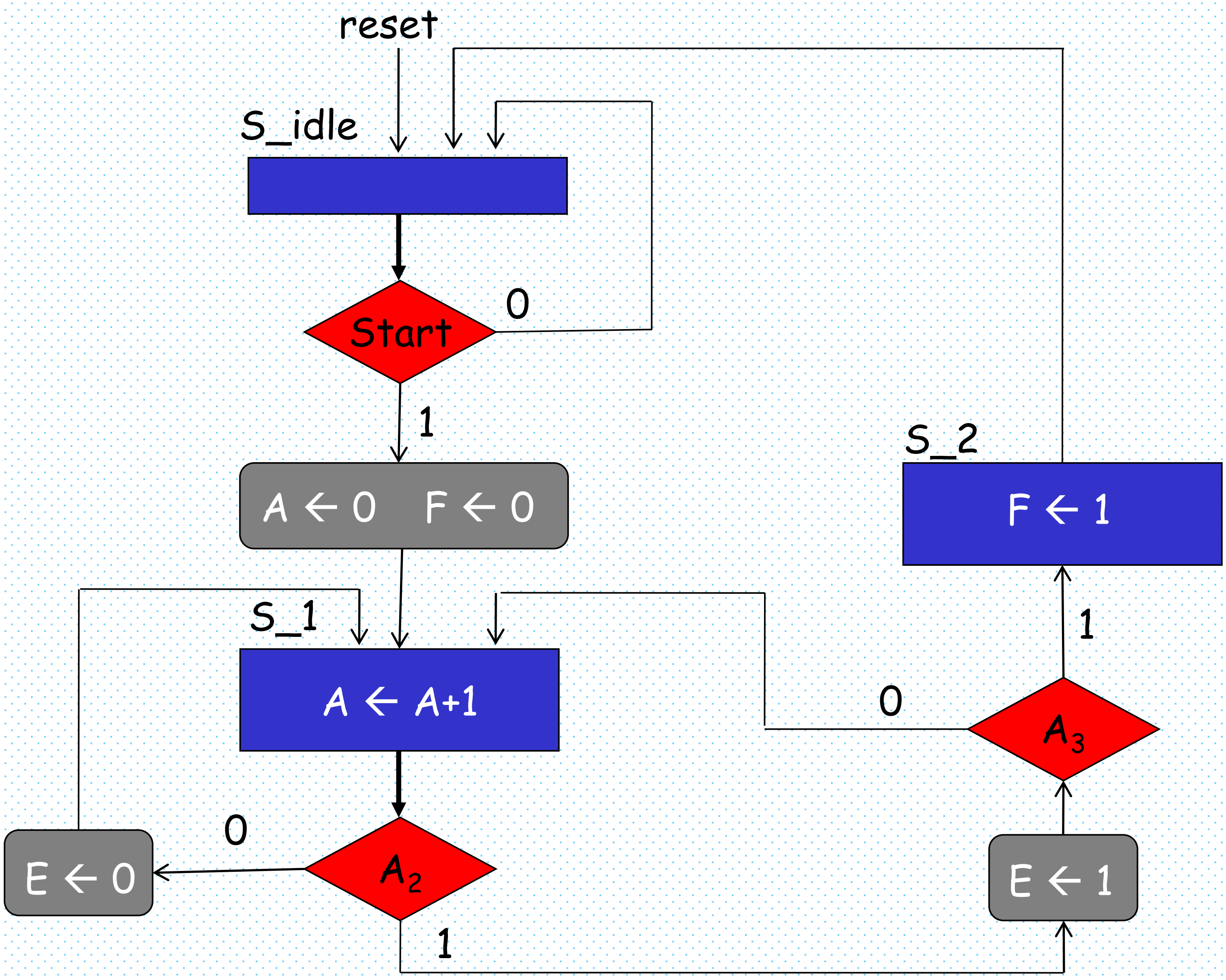


ASMD Charts

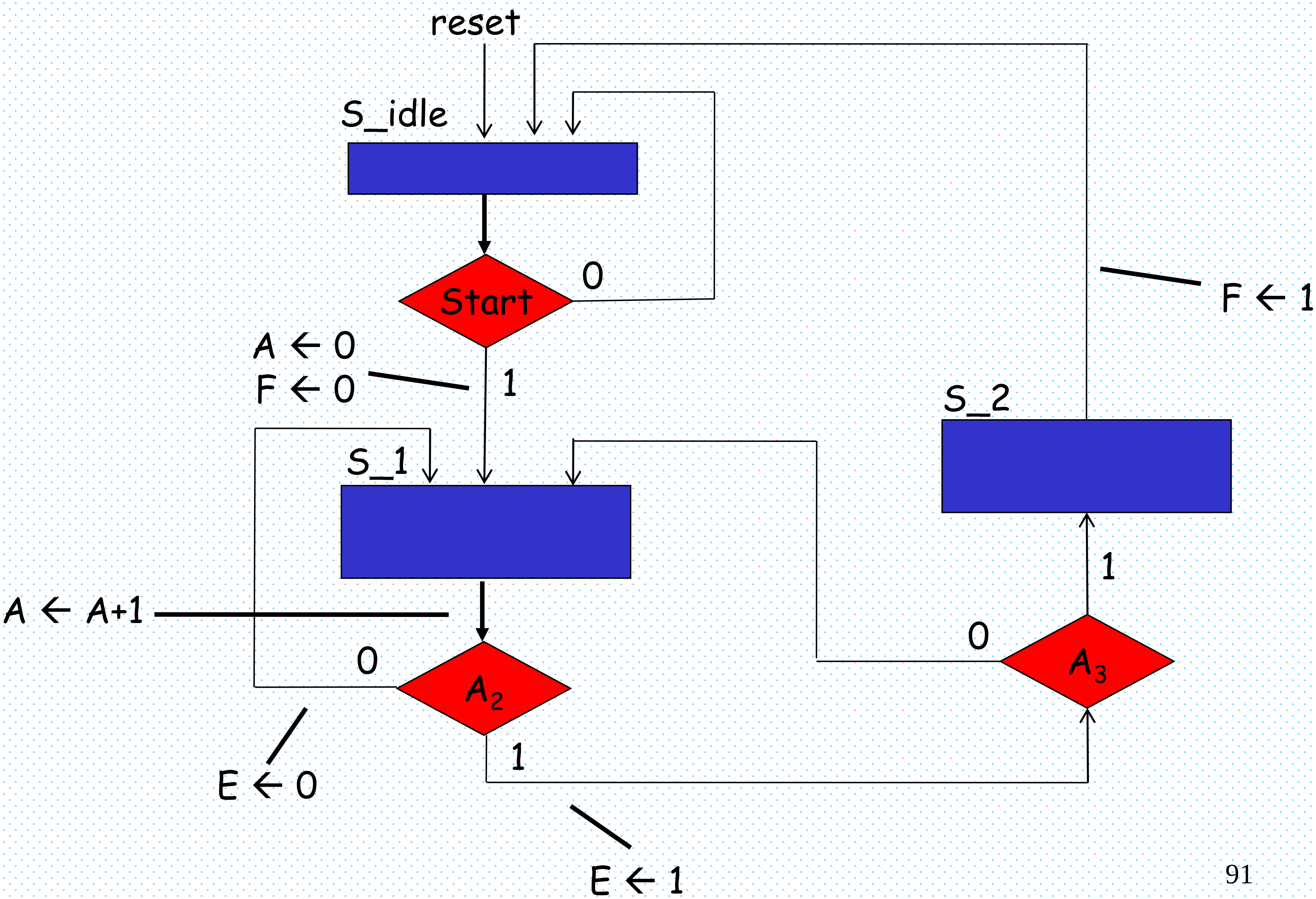


Start
Control
Unit
Status signals
Control signals
Datapath
A
E
F
Data
outputs
clock
reset

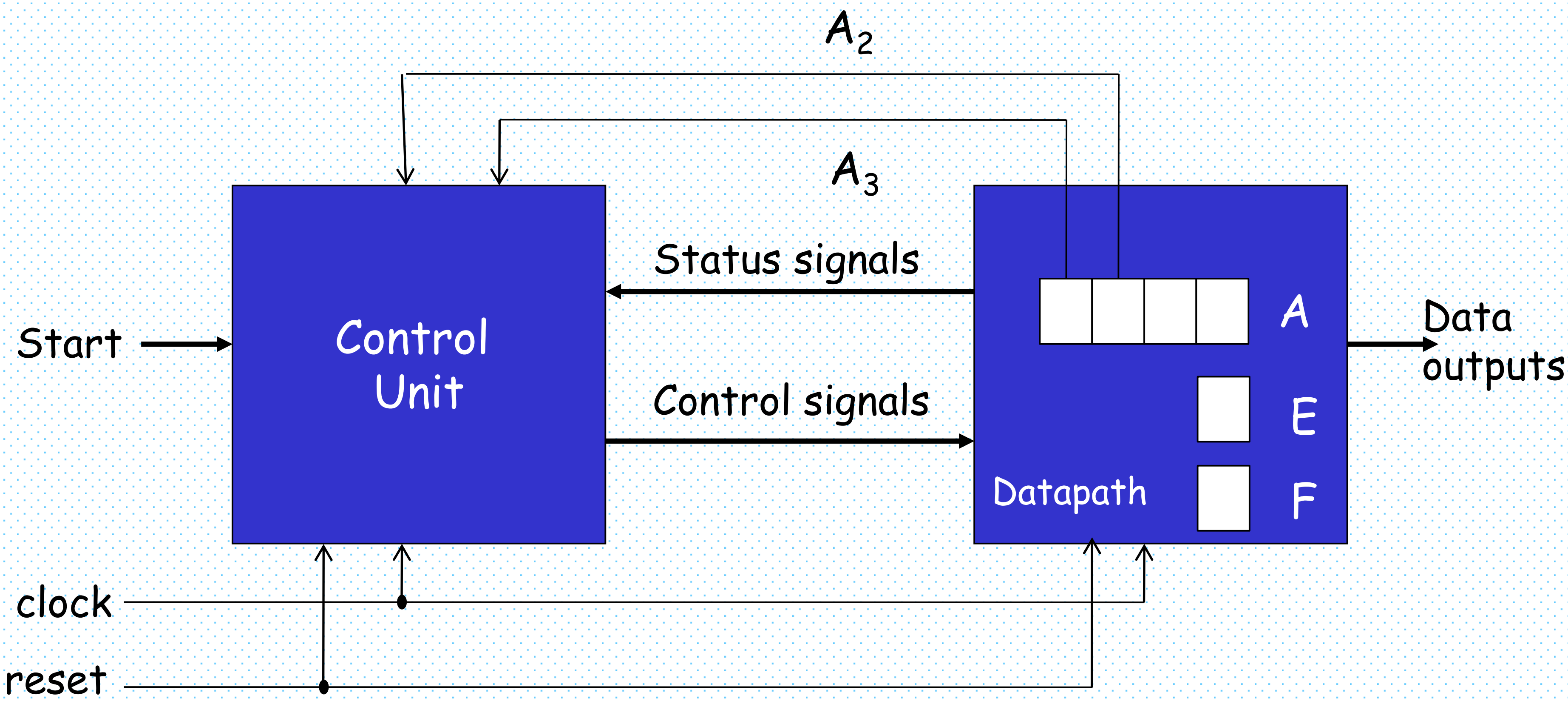
ASM Chart



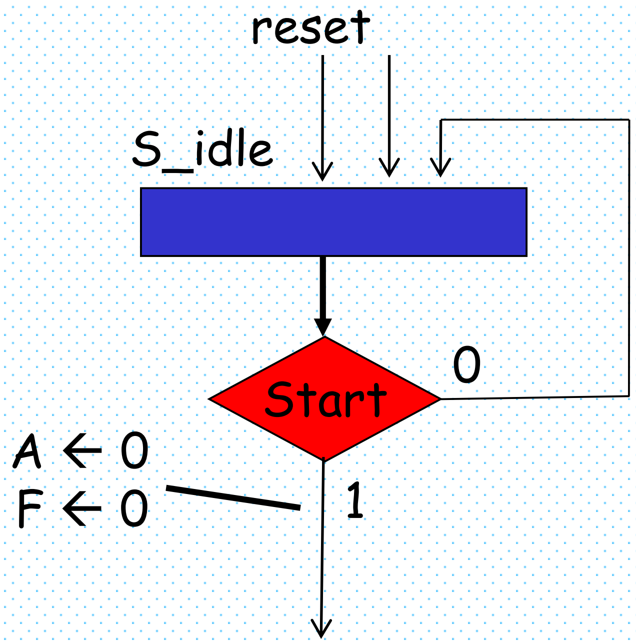
ASMD Chart – Partially Specified

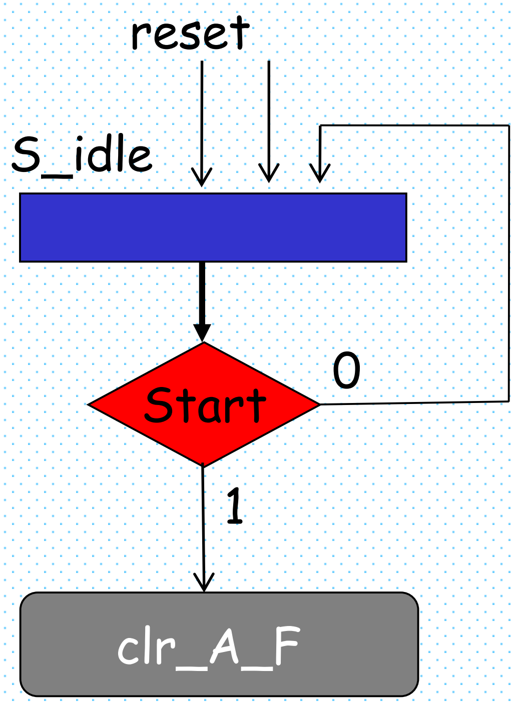


Status Signals

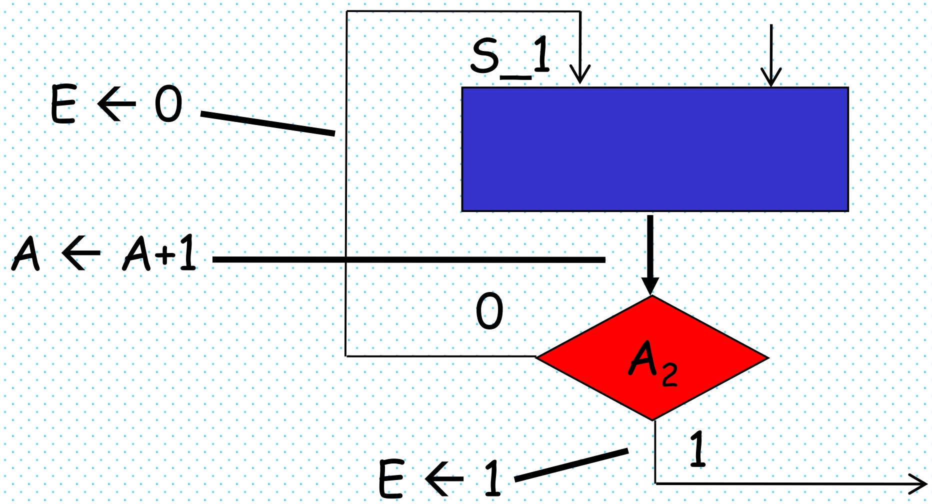


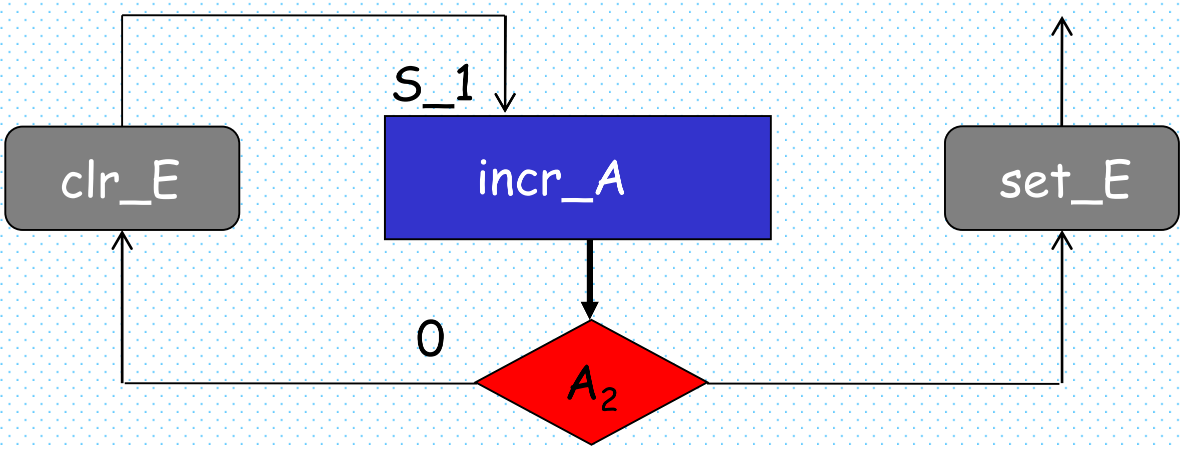
ASMD Chart - Fully Specified



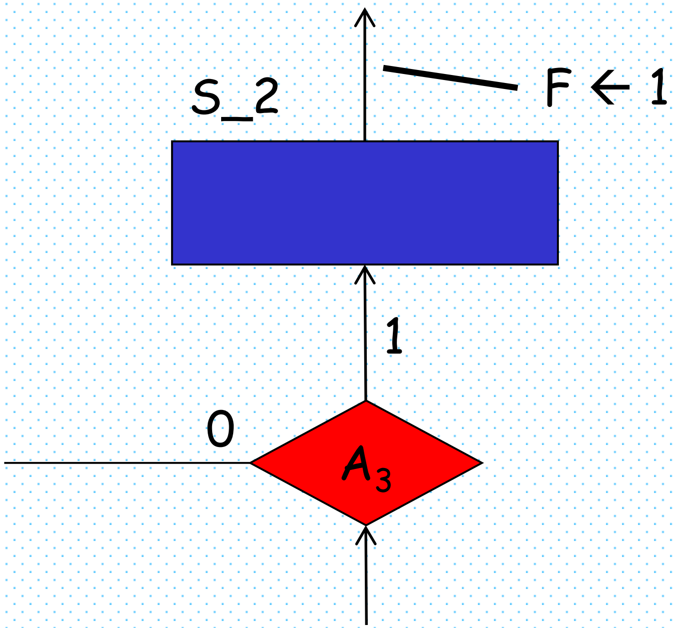


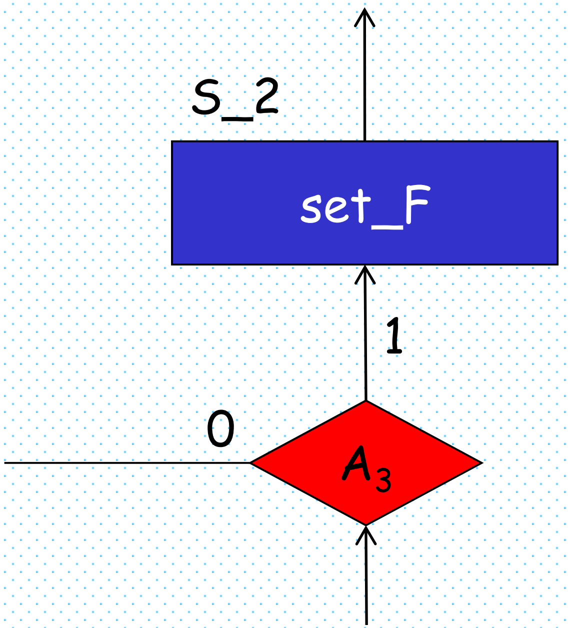
ASMD Chart - Fully Specified



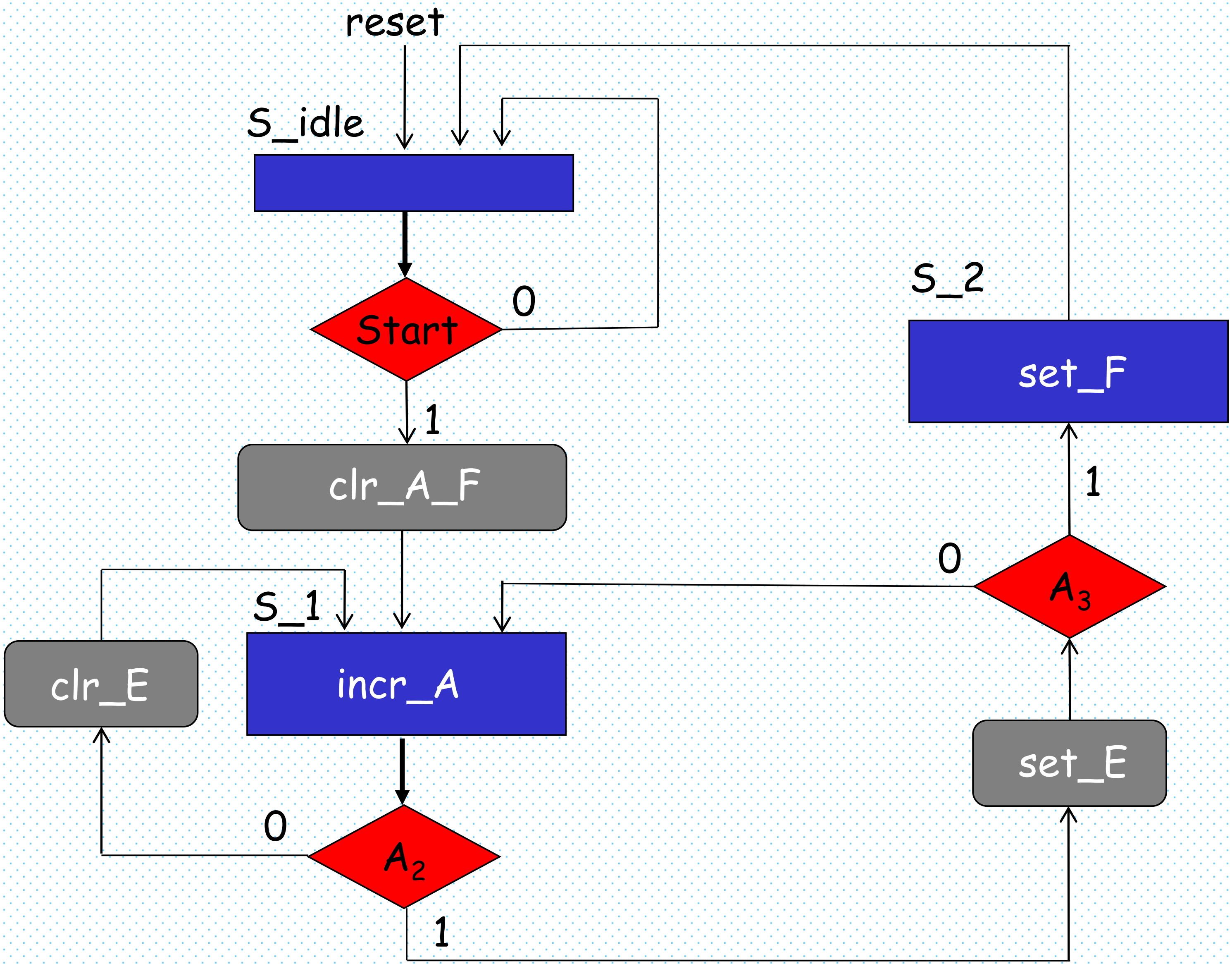


ASMD Chart - Fully Specified





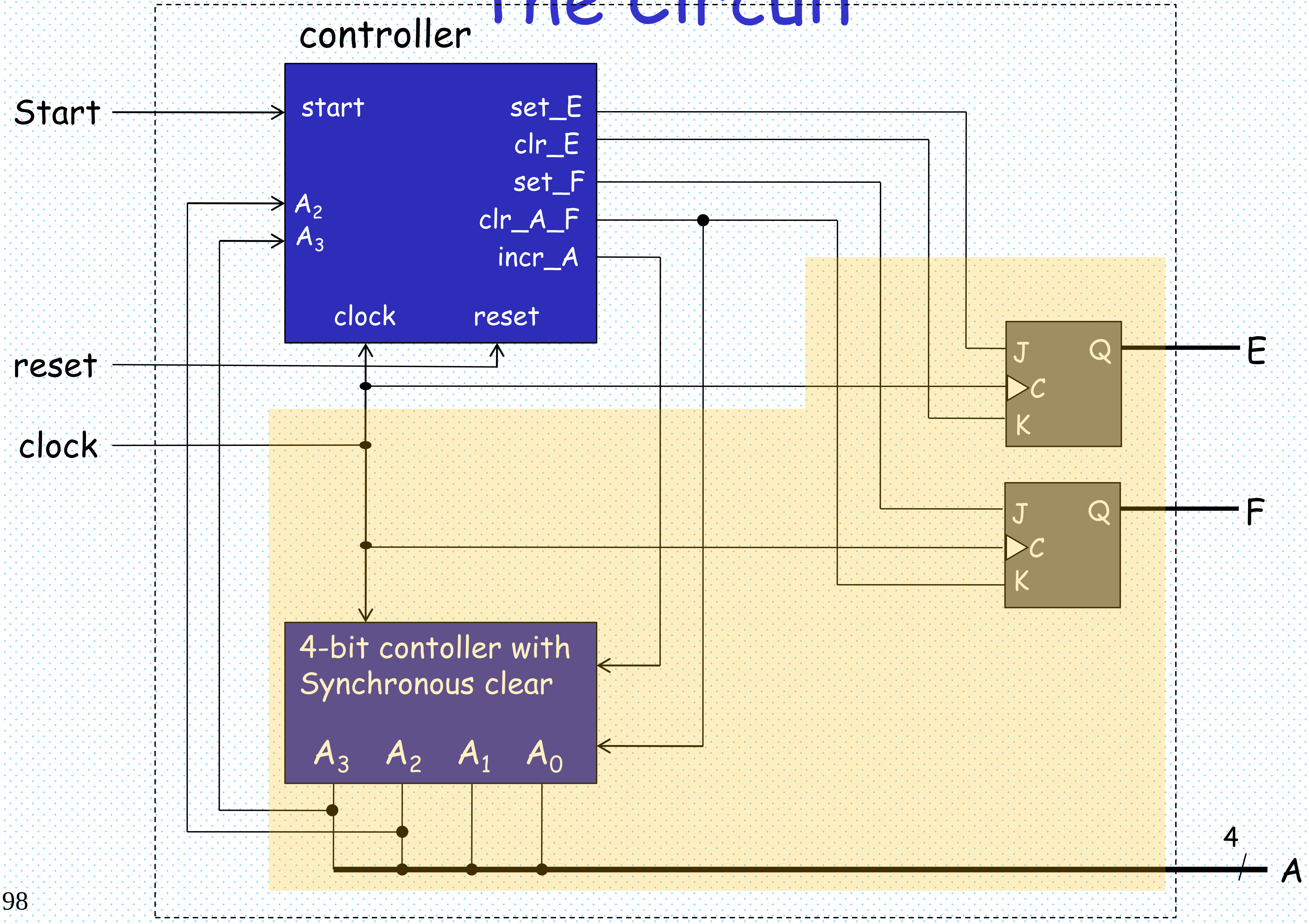
ASMD Chart - Fully Specified



Sequence of Operations

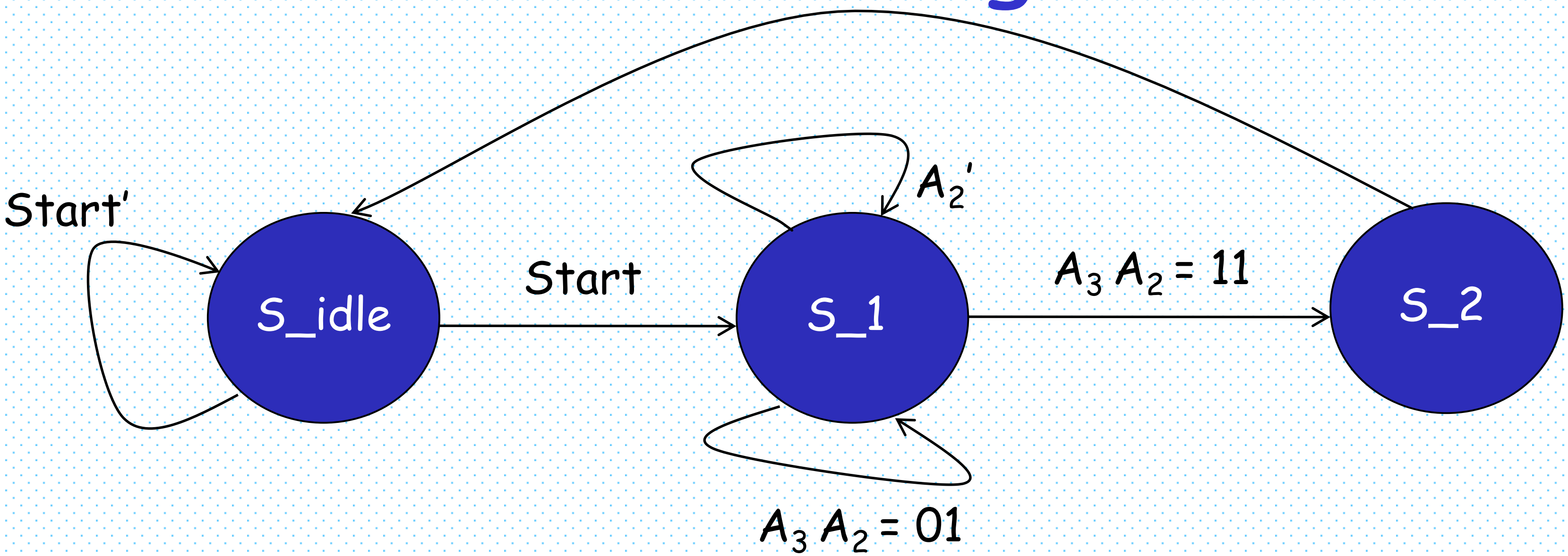
Counter				Flip-Flops		Conditions	State
A_3	A_2	A_1	A_0	E	F		
0	0	0	0	1	0	$A_2 = 0, A_3 = 0$	S_1
0	0	0	1	0	0		
0	0	1	0	0	0		
0	0	1	1	0	0		
0	1	0	0	0	0	$A_2 = 1, A_3 = 0$	S_1
0	1	0	1	1	0		
0	1	1	0	1	0		
0	1	1	1	1	0		
1	0	0	0	1	0	$A_2 = 0, A_3 = 1$	S_1
1	0	0	1	0	0		
1	0	1	0	0	0		
1	0	1	1	0	0		
1	1	0	0	0	0	$A_2 = 1, A_3 = 1$	
1	1	0	1	1	0		S_2
1	1	0	1	1	1		S_idle

The Circuit



controller
Start
start
set_E
clr_E
set_F
A_2
A_3
clr_A_F
incr_A
clock
reset
reset
clock
J
Q
C
K
E
J
Q
C
K
F
4-bit contoller with
Synchronous clear
A_3 A_2 A_1 A_0
4
A

State Diagram

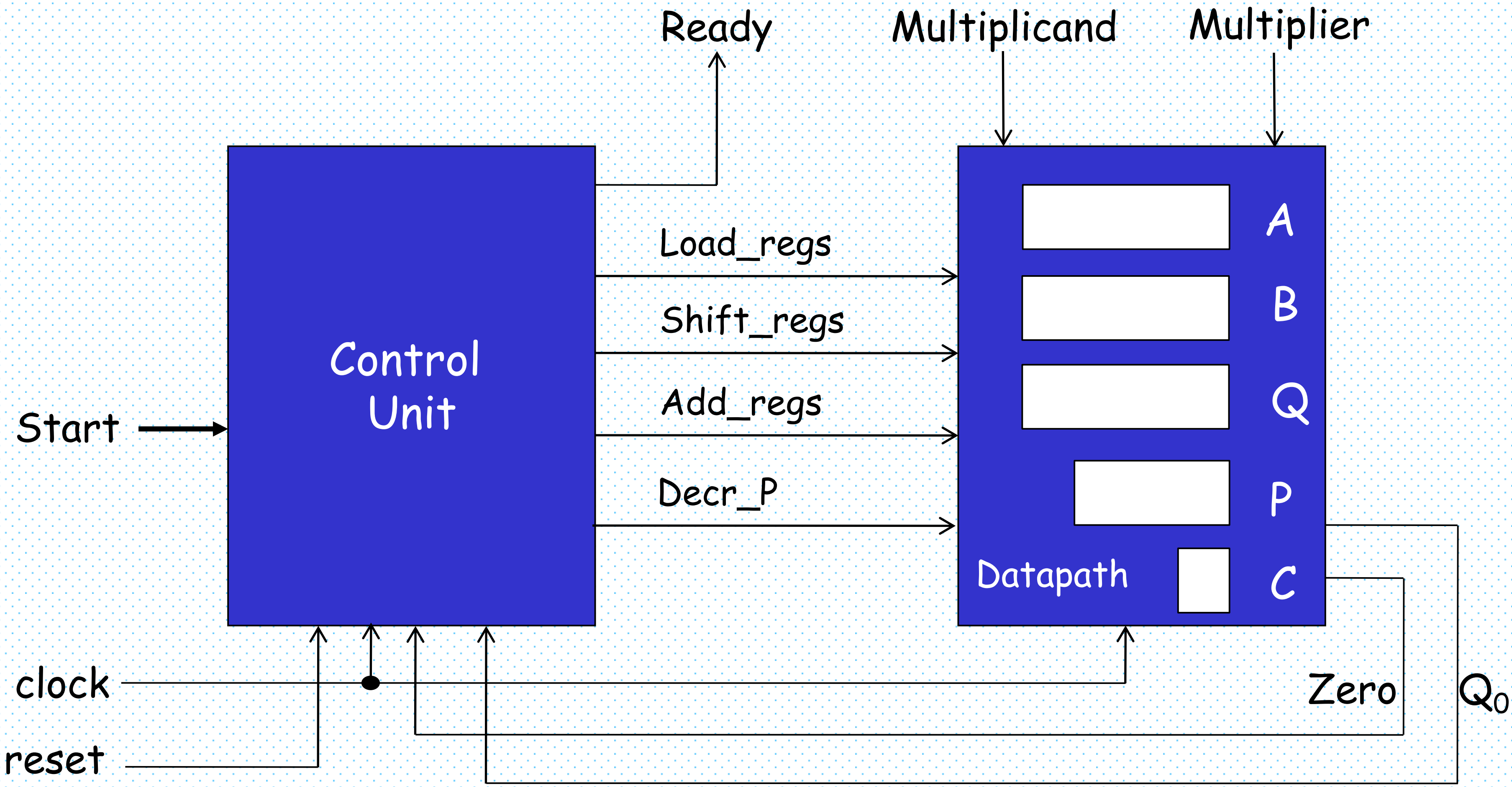


S_idle: if (Start = 1) clr_A_F = 1, next_state ← S_1, (A←0, F←0)
else next_state ← S_idle

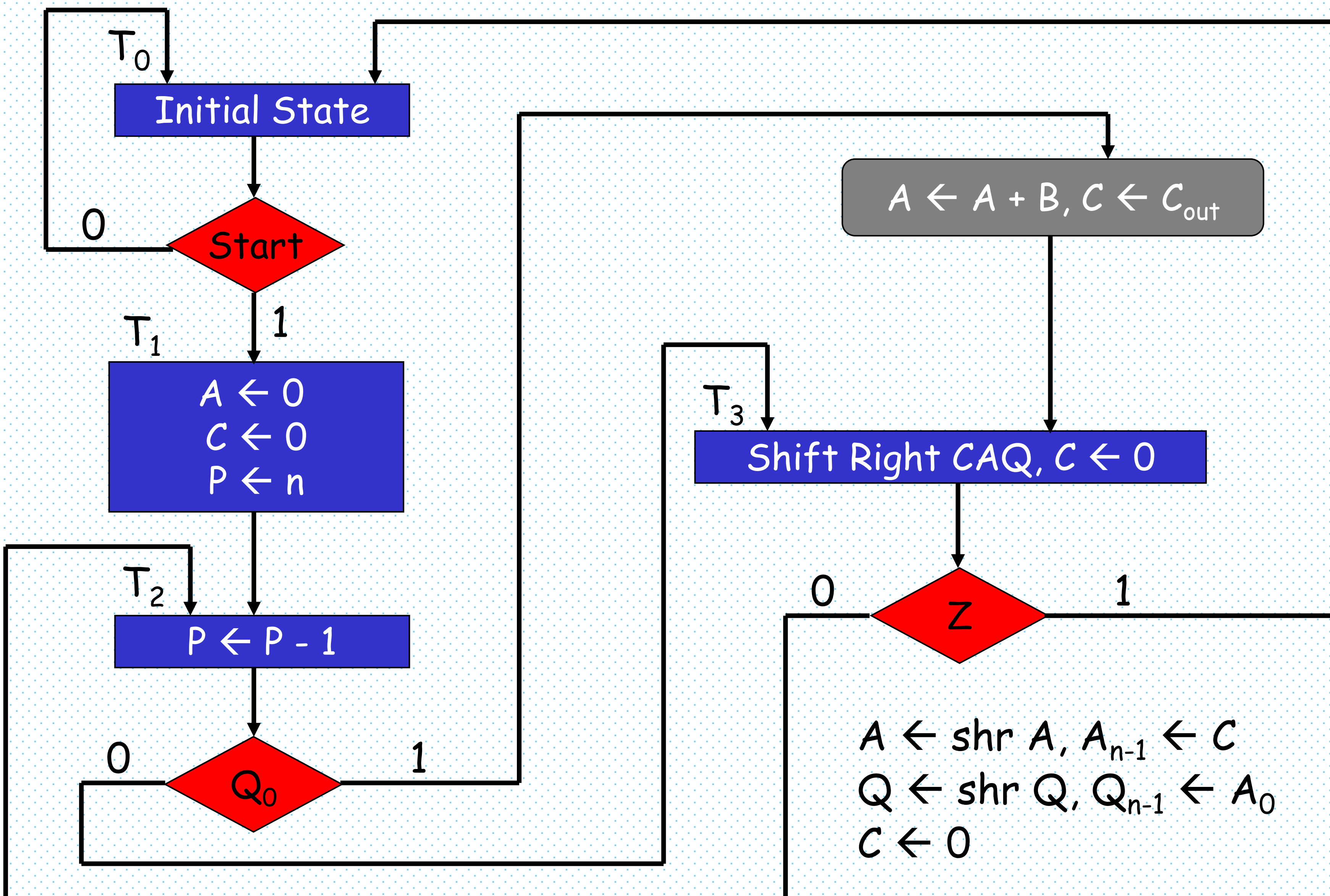
S_1: incr_A = 1
if (A_2 = 0) clr_E = 1, next_state ← S_1, (E←0)
else
if (A_3 = 0) set_E = 1, next_state ← S_1 (E←1)
else set_E = 1, next_state ← S_2

S_2: set_F = 1, next_state ← S_idle (F←1)

Binary Multiplier with ASMD



ASM Chart

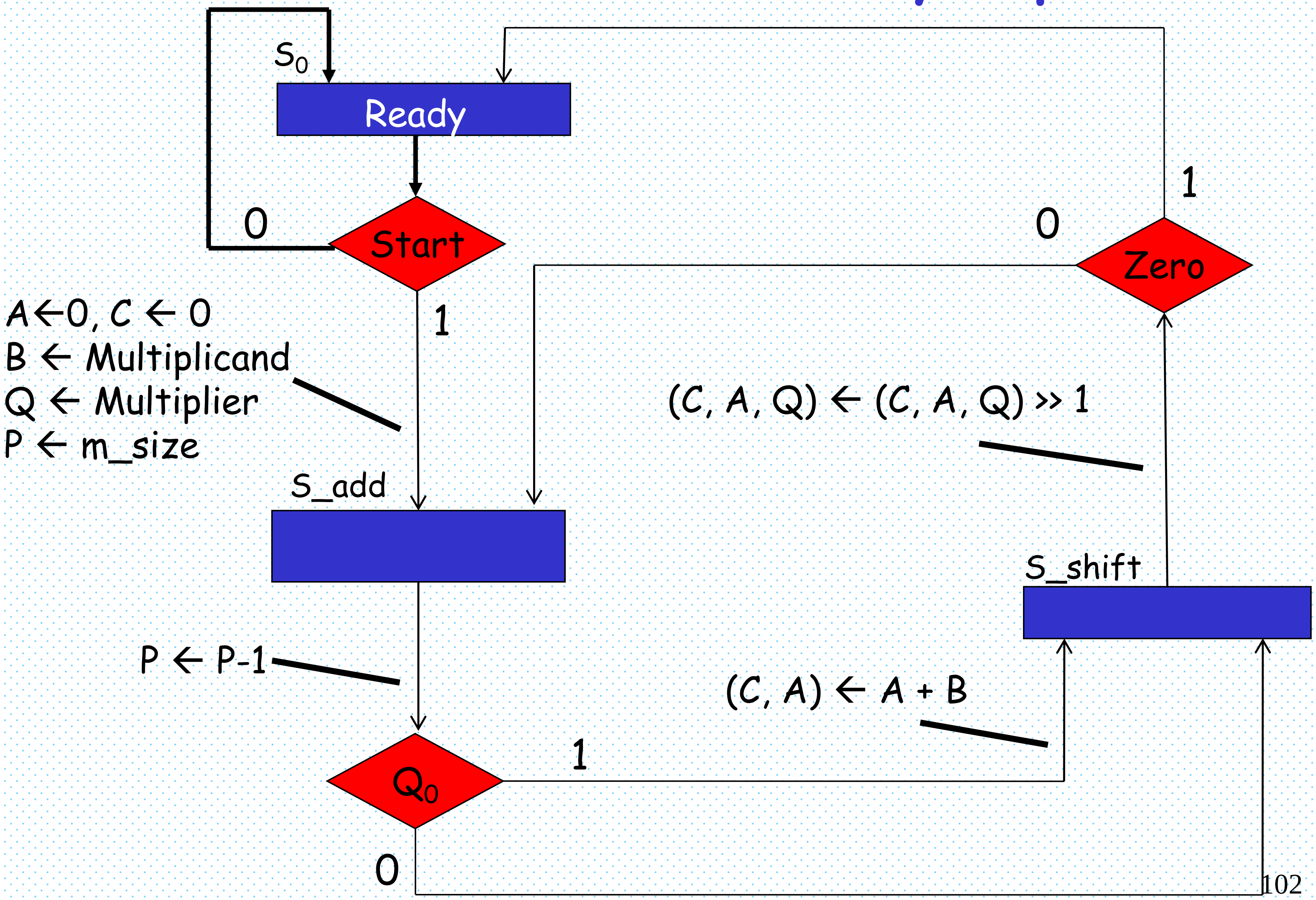


$A \leftarrow$ shr A, $A_{n-1} \leftarrow C$

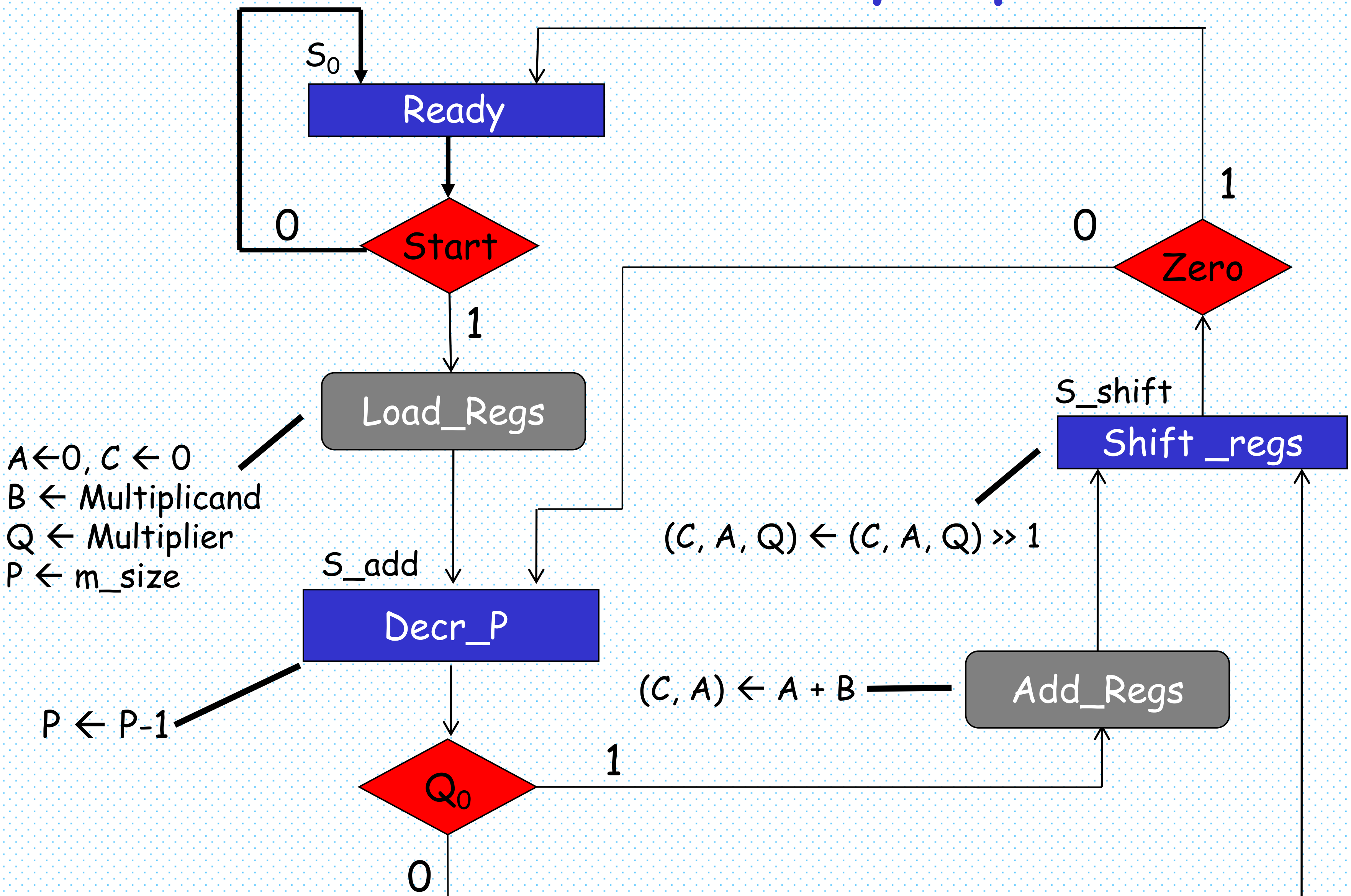
$Q \leftarrow$ shr Q, $Q_{n-1} \leftarrow A_0$

$C \leftarrow 0$

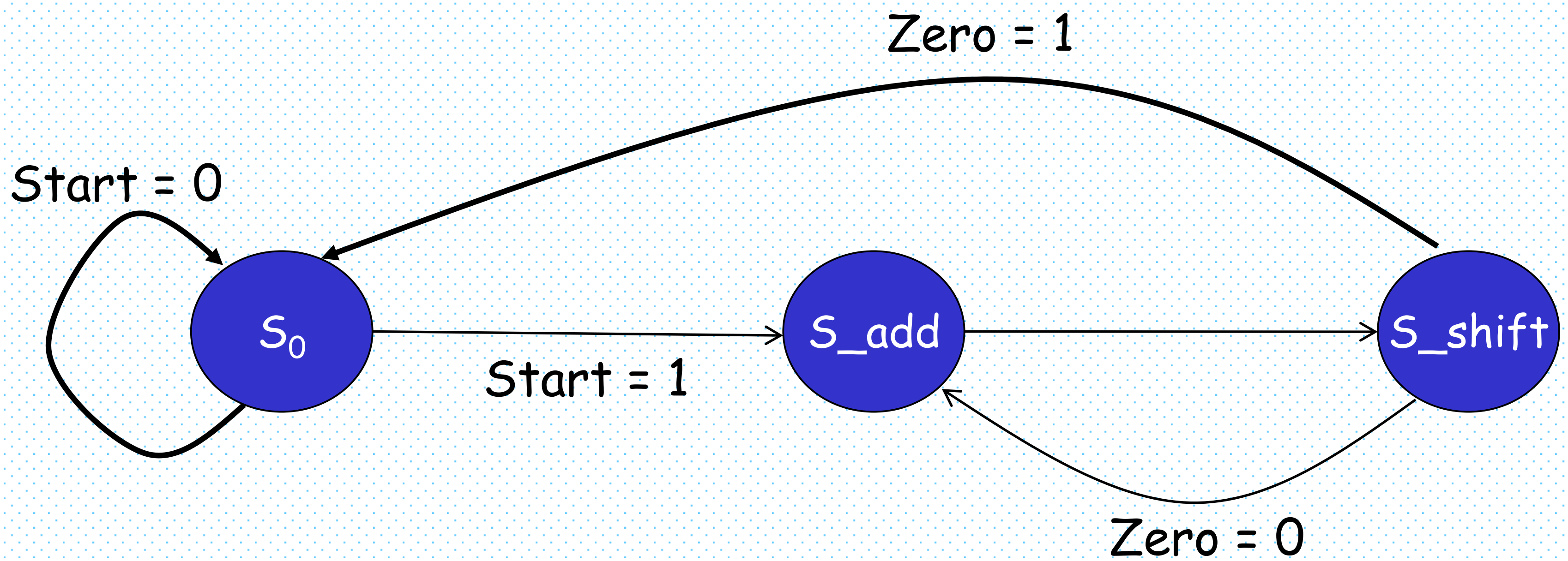
ASMD Chart – Partially Specified



ASMD Chart - Fully Specified



State Diagram

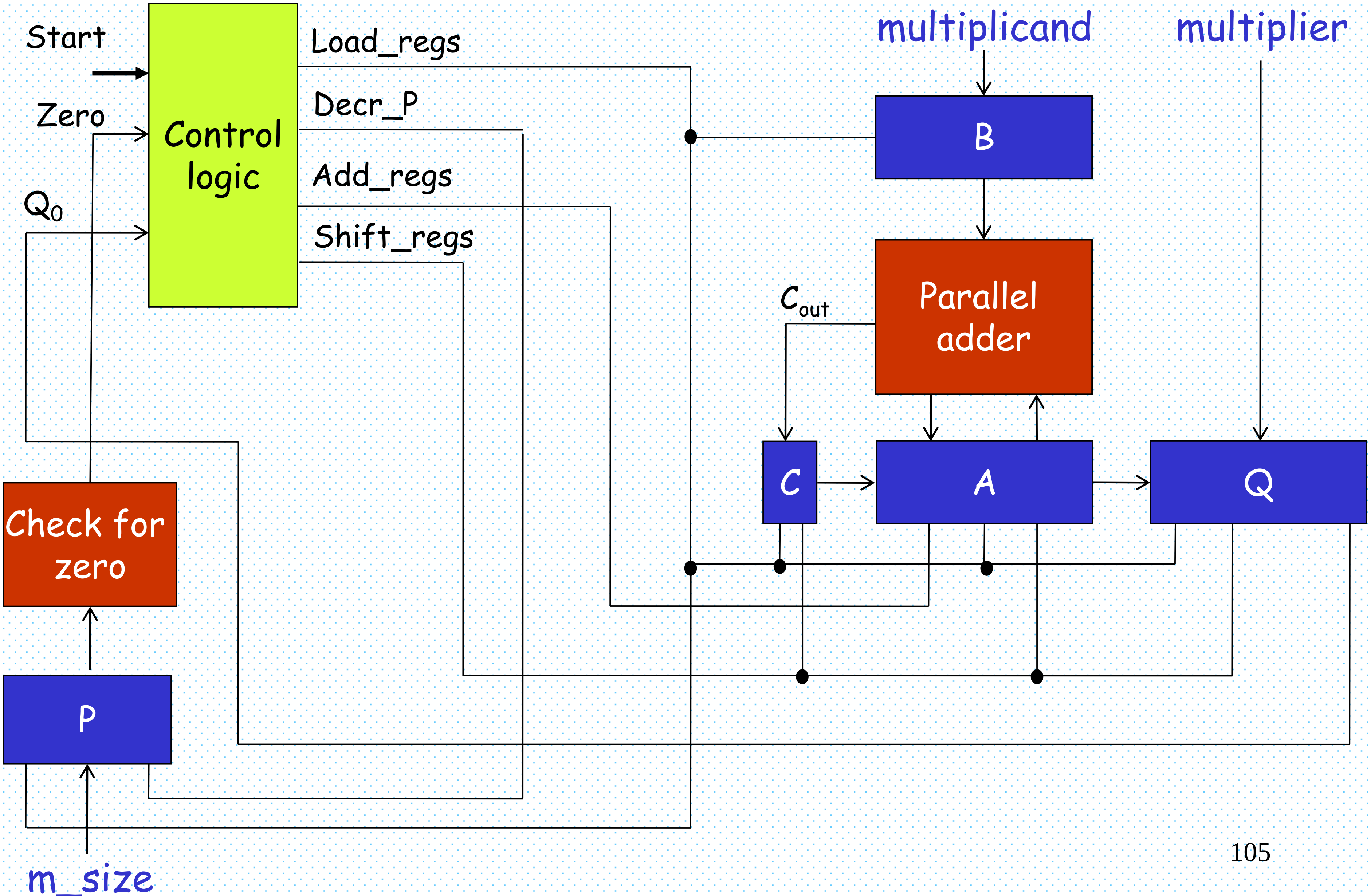


S_0: if (Start = 1) Load_regs = 1, next_state ← S_add
else next_state ← S_0

S_add: next_state ← S_shift
if (Q_0 = 1) Add_regs = 1

S_shift: Shift_regs = 1
if (Zero = 1) next_state ← S_0
else next_state ← S_add

Multiplier Circuit



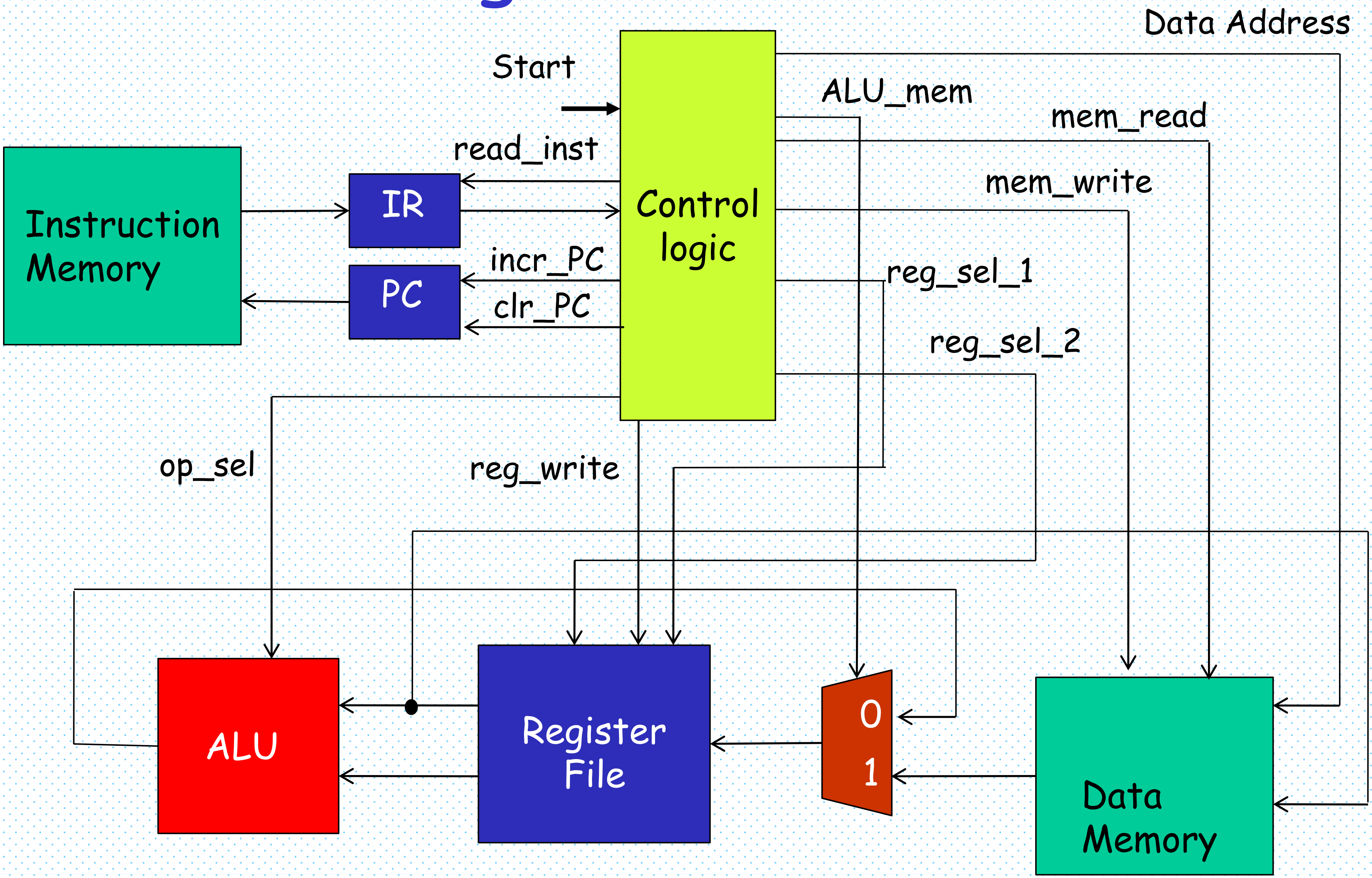
A Simple Microprocessor Design

- Operation
  - `a = b + c;`
- translated into
  - `LOAD R0, b`
  - `LOAD R1, c`
  - `ADD R0, R0, R1`
  - `STORE R0, a`
- Encoding
  - `LOAD: 00, ADD: 01, STORE: 10`
  - `R0: 00, R1: 01`
  - `Address: four bit value`

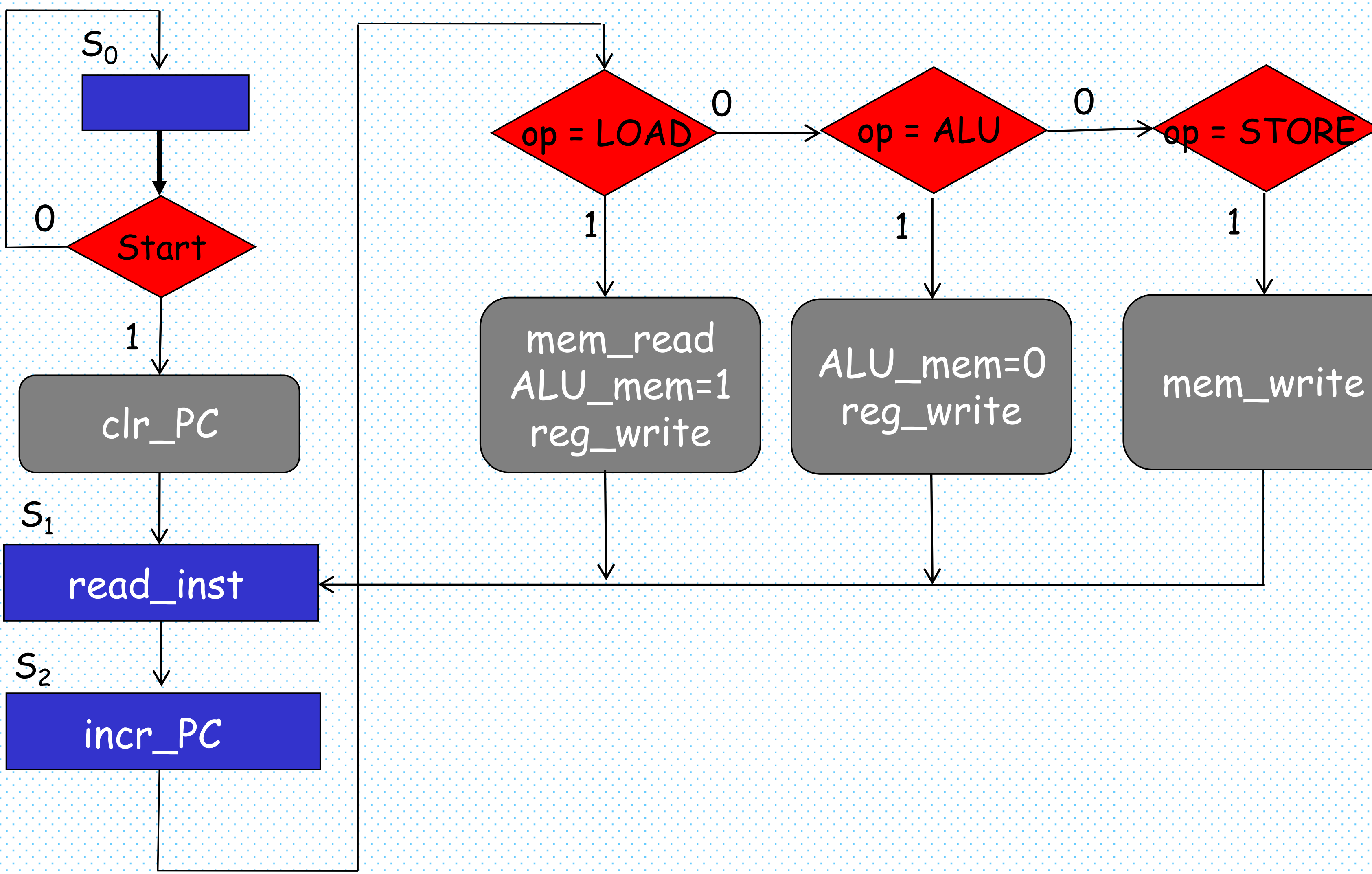
A Simple Microprocessor Design

- Encoding of Instructions
  - `LOAD R0, b` → `00 00 0000`
  - `LOAD R1, c` → `00 01 0001`
  - `ADD R0, R0, R1` → `01 00 00 01`
  - `STORE R0, a` → `10 00 0010`
- Addresses of Instructions
  - `LOAD R0, b` → `000`
  - `LOAD R1, c` → `001`
  - `ADD R0, R0, R1` → `010`
  - `STORE R0, a` → `011`

Block Diagram of the Processor



ASDM Chart



S_0
0
Start
1
clr_PC
S_1
read_inst
S_2
incr_PC
op = LOAD
0
op = ALU
0
op = STORE
1
1
1
mem_read
ALU_mem=1
reg_write
ALU_mem=0
reg_write
mem_write