University Of Diyala
College Of Engineering
Department of Computer Engineering

Digital System Design II

ASM Based Datapath and Control Design

Dr. Yasir Al-Zubaidi

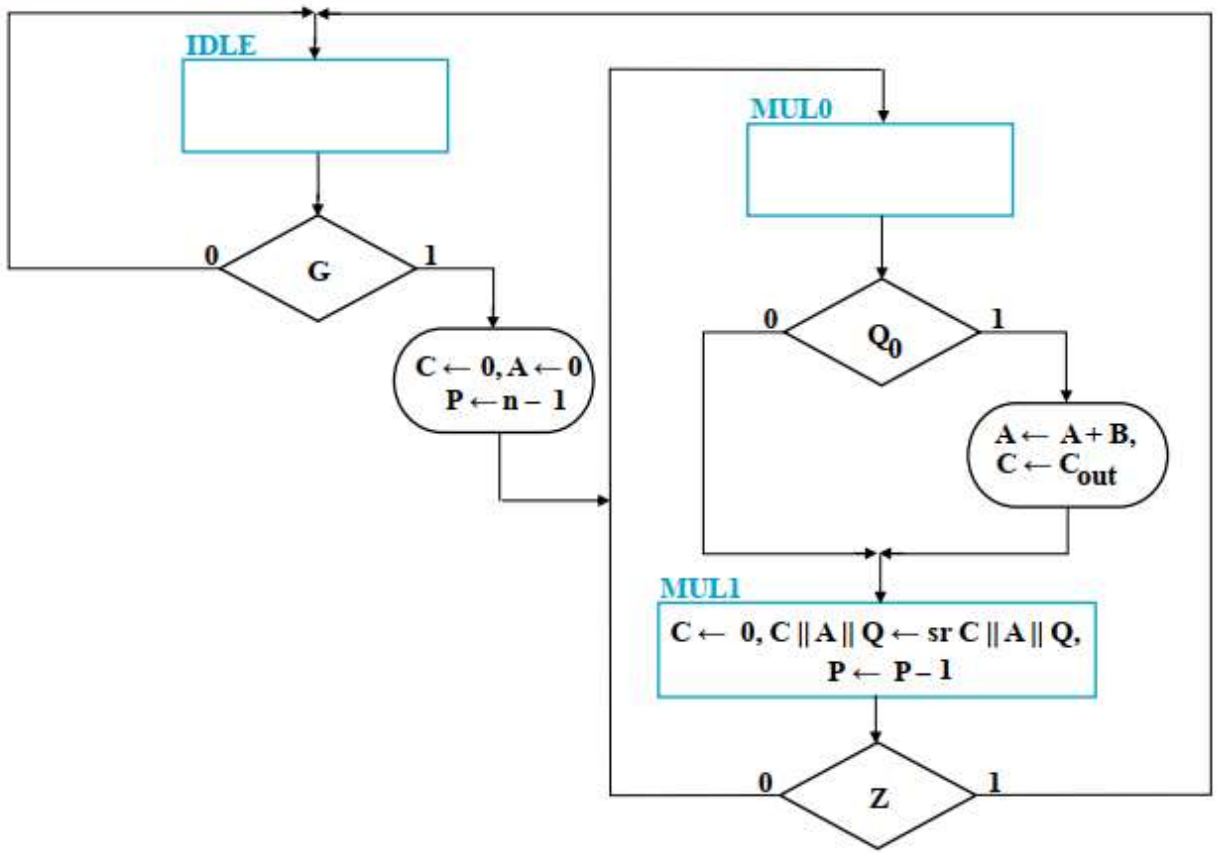
Third stage

2021

Overview

- Datapath and control
- Microoperations
- Sequencing and control
  - Algorithmic State Machines (ASM)
    - ASM chart
    - Timing considerations
    - ASM chart examples: Binary multiplier
  - Hardwired Control
    - Control design methods
    - Sequence register and decoder
    - One flip-flop per state
  - Microprogrammed control

Multiplier Example: ASM Chart



Multiplier Example: ASM Chart (Contd.)

- Three states employed here:
  - IDLE state:
    - input G is used as the condition for starting the multiplication
    - C, A, and P are initialized
  - MUL0 state: conditional addition is performed based on the value of Q_0.
  - MUL1 state:
    - right shift is performed to capture the partial product and position the next bit of the multiplier in Q_0
    - Down counter P = P - 1
    - P=0 is used to sense completion or continuation of the multiplication.

Multiplier Example: Control Signal Table

Control Signals for Binary Multiplier

Block Diagram Module	Microoperation	Control Signal Name	Control Expression
Register A:	$A \leftarrow 0$	Initialize	IDLE $\cdot$ G
	$A \leftarrow A + B$	Load	MUL0 $\cdot$ Q_0
	$C \parallel A \parallel Q \leftarrow \text{sr}\ C \parallel A \parallel Q$	Shift_dec	MUL1
Register B:	$B \leftarrow IN$	*Load_B*	LOADB
Flip-Flop C:	$C \leftarrow 0$	Clear_C	IDLE $\cdot$ G + MUL1
	$C \leftarrow C_{out}$	Load	—
Register Q:	$Q \leftarrow IN$	*Load_Q*	LOADQ
	$C \parallel A \parallel Q \leftarrow \text{sr}\ C \parallel A \parallel Q$	Shift_dec	—
Counter P:	$P \leftarrow n - 1$	Initialize	—
	$P \leftarrow P - 1$	Shift_dec	—

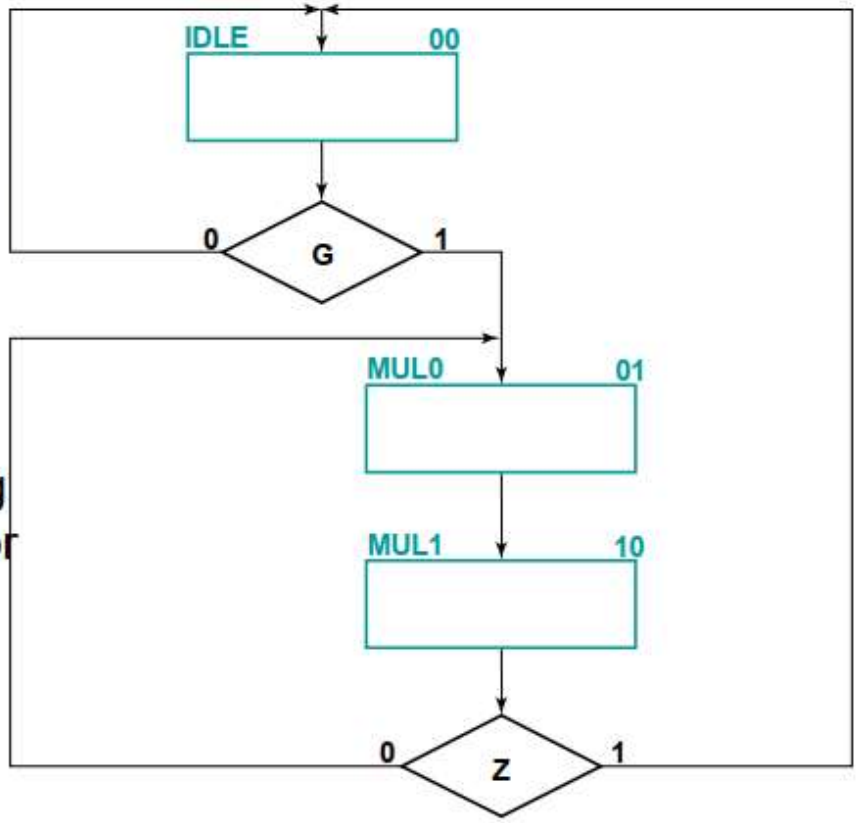
Multiplier Example: Control Signal Table (Contd.)

- Signals are defined on a register basis
- LOADQ and LOADB: external signals controlled from the system using the multiplier and will not be considered a part of this design
- Many control signals are “reused” for different registers.
  - These 4 control signals are the “outputs” of the control unit: initialize, load, shift_dec, clear_c

Multiplier Example - Sequencing Part of ASM

- With the outputs represented by the table, they can be removed from the ASM making the ASM to represent only the sequencing (next state) behavior

Similar to FSM



Hardwired Control

- Control Design Methods
  - Procedure specializations that use a single signal to represent each state
    - Sequence Register and Decoder
      - Sequence register with encoded states, e.g., 00, 01, 10, 11.
      - Decoder outputs produce “state” signals, e.g., 0001, 0010, 0100, 1000.
    - One Flip-flop per State
      - Flip-flop outputs as “state” signals, e. g., 0001, 0010, 0100, 1000.

Multiplier Example: Sequencer and Decoder Design - Specification

- Initially, use sequential circuit design techniques
- First, define:
  - States: IDLE, MUL0, MUL1
  - Input Signals: G, Z, Q_0 (Q_0 affects outputs, not next state)
  - Output Signals: Initialize, LOAD, Shift_Dec, Clear_C
  - State Transition Diagram (Use Sequencing ASM)
  - Output Function: Use Control Signal Table
- Second, find
  - State Assignments
  - Use two state bits to encode the three states IDLE, MUL0, and MUL1.

State	M1	M0
IDLE	0	0
MUL0	0	1
MUL1	1	0
Unused	1	1

Multiplier Example: Sequencer and Decoder Design - Formulation

- Assuming that state variables M1 and M0 are decoded into states, the next state part of the state table is:

Current State	Input G Z	Next State M1 M0
IDLE	0 0	0 0
IDLE	0 1	0 0
IDLE	1 0	0 1
IDLE	1 1	0 1
MUL0	0 0	1 0
MUL0	0 1	1 0
MUL0	1 0	1 0
MUL0	1 1	1 0

Current State M1 M0	Input G Z	Next State M1 M0
MUL1	0 0	0 1
MUL1	0 1	0 0
MUL1	1 0	0 1
MUL1	1 1	0 0
Unused	0 0	d d
Unused	0 1	d d
Unused	1 0	d d
Unused	1 1	d d

Multiplier Example: Sequencer and Decoder Design –Equations Derivation/Optimization

- Finding the equations for M1 and M0 using decoded states:

  $M1 = MUL0$

  $M0 = IDLE \cdot G + MUL1 \cdot \overline{Z}$

- The output equations using the decoded states:

  $Initialize = IDLE \cdot G$

  $Load = MUL0 \cdot Q_0$

  $Clear\_C = IDLE \cdot \overline{G} + MUL1$

  $Shift\_dec = MUL1$

- Doing multiple level optimization, extract $IDLE \cdot G$:

  $START = IDLE \cdot G$

  $M1 = MUL0$

  $M0 = START + MUL1 \cdot \overline{Z}$

  $Initialize = START$

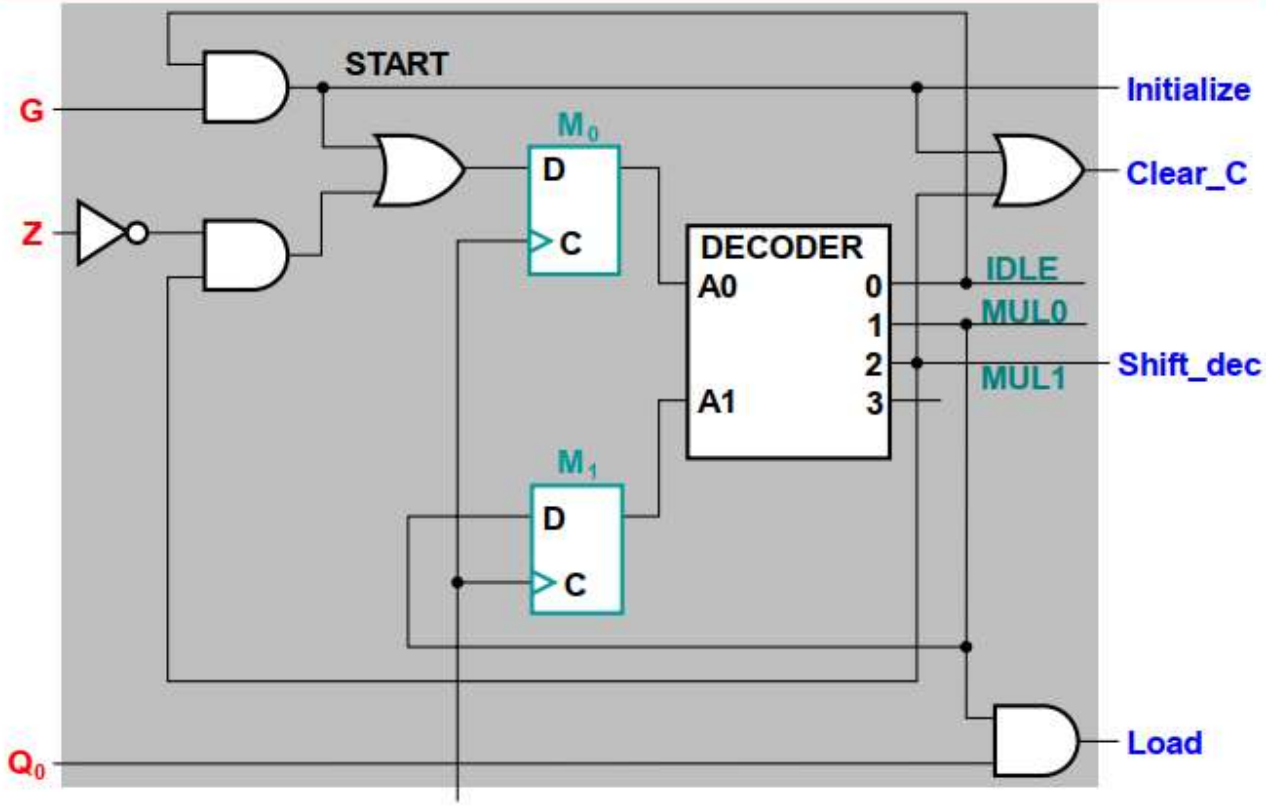
  $Load = MUL0 \cdot Q_0$

  $Clear\_C = \overline{START} + MUL1$

  $Shift\_dec = MUL1$

- The resulting circuit using flip-flops, a decoder, and the above equations is given on the next slide.

Multiplier Example: Sequencer and Decoder Design - Implementation



```
--Binary multiplier with n=4
library ieee;
use ieee.std_logic_unsigned.all;
entity binary_multiplier is
      port(CLK, RESET, G, LOADB, LOADQ: in std_logic;
         MULT_IN : in std_logic_vector (3 downto 0);
         MULT_OUT : out std_logic_vector (7 downto 0));
end bianry_multiplier;

architecture behavior_4 of binary_multiplier is
      type state_type is (IDLE, MUL0, MUL1);
      variable P:=3;
      signal state, next_state : state_type;
      signal A, B, Q:std_logic_vector(3 downto 0);
      signal C, Z:std_logic;
begin
      Z<= P(1) NOR P(0);
      MULT_OUT <= A & Q;

      state_register : process (CLK, RESET)
      begin
        if (RESET = '1') then
           state <= IDLE;
        elsif (CLK'event and CLK='1') then
           state <= next_state;
        endif;
      end process;

      next_state_func : process (G, Z, state)
      begin
        case state is
          when IDLE =>
                if G='1' then     next_state <= MUL0;
                else              next_state <= IDLE;
                end if;
          when MUL0 =>
                next_state <= MUL1;
          when MUL1 =>
                if Z='1' then     next_state <= IDLE;
                else              next_state <= MUL0;
                end if;
        end case;
      end process;
```

```
datapath_func : process (CLK)
variable CA: std_logic_vector (4 downto 0);
begin
  if (CLK'event and CLK='1') then
      if LOADB='1' then
        B <= MULT_IN;
      end if;
      if LOADQ = '1' then
        Q <= MULT_IN;
      end if;
      case state is
        when IDLE =>
           if G = '1' then
                C <= '0';
                A <= "0000";
                P <= "11";
           end if;
        when MUL0 =>
           if Q(0) ='1' then
                CA := ('0' & A) + ('0' & B);
           else
                CA := C & A;
           end if;
           C <= CA(4);
           A <= CA(3 downto 0);
        when MUL1 =>
           C <= '0';
           A <= C & A(3 downto 1);
           Q <= A(0) & Q(3 downto 1);
           P <= P - "01";
      end case;
  end if;
end process;

end behavior_4;
```

Speeding Up the Multiplier

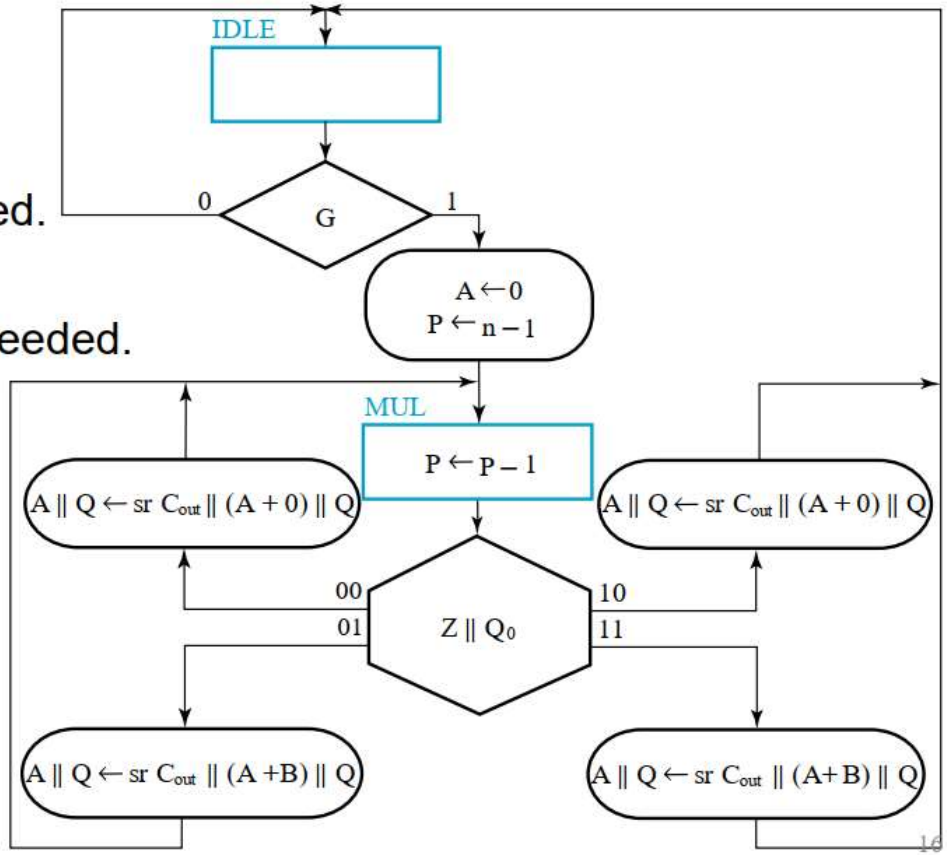
- In processing each bit of the multiplier, the circuit visits states MUL0 and MUL1 in sequence.
- By redesigning the multiplier, is it possible to visit only a single state per bit processed?

Speeding Up Multiply (Contd.)

- The operations in MUL0 and MUL1:
  - In MUL0, a conditional add of B
  - In MUL1, a right shift of C || A || Q in a shift register, the decrementing of P, and a test for P = 0 (on the old value of P)
- Any solution that uses one state must combine all of the operations listed into one state
  - The operations involving P are already done in a single state, so not a problem.
  - The right shift, however, depends on the result of the conditional addition. So these two operations must be combined!

Speeding Up Multiply (Contd.)

- By replacing the shift register with a combinational shifter and combining the adder and shifter, the states can be merged.
- The C-bit is no longer needed.
- In this case, Z and Q_0 have been made into a vector.



Microprogrammed Control

- *Microprogrammed Control* — a control unit with binary control values stored as words in memory.
- *Microinstructions* — words in the control memory.
- *Microprogram* — a sequence of microinstructions.
- *Control Memory* — RAM or ROM memory holding the microinstructions.
  - *Writeable Control Memory* — RAM Memory into which microinstructions may be written