EE 459/500 – HDL Based Digital Design with Programmable Logic

Lecture 13

Control and Sequencing: Hardwired and Microprogrammed Control

References:

Chapter s 4,5 from textbook

Chapter 7 of M.M. Mano and C.R. Kime, Logic and Computer Design Fundamentals, Pearson Prentice-Hall, 2008.

Overview

- Control and Sequencing
  - Algorithmic State Machine (ASM) Chart of Multiplier
  - Hardwired control
  - Microprogrammed control

Multiplier Example

- Example: (101 x 011)
- Partial products are:
  101 x 0, 101 x 1, and 101 x 1

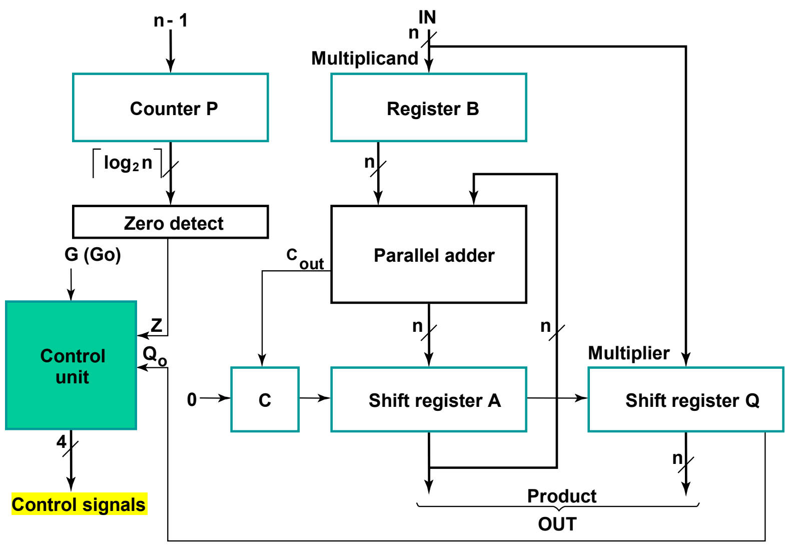
```
          1   0   1
      x   0   1   1
      -------------
          1   0   1
      1   0   1
  0   0   0
  -----------------
0 0   1   1   1   1
```

Example (1 0 1) x (0 1 1) again

- Reorganizing to follow hardware algorithm:

	1	0	1				**Multiplicand (B)**
x	0	1	1				**Multiplier (Q)**
0	0	0	0				**Clear C || A** (Carry and register A)
+	1	0	1				**Multiplier$_0$ = 1 => Add B**
0	1	0	1				**Addition**
0	0	1	0	1			**Shift Right (Zero-fill C)**
+	1	0	1				**Multiplier$_1$ = 1 => Add B**
0	1	1	1	1			**Addition**
0	0	1	1	1	1		**Shift Right**
0	0	0	1	1	1	1	**Multiplier$_2$ = 0 => No Add, Shift Right**

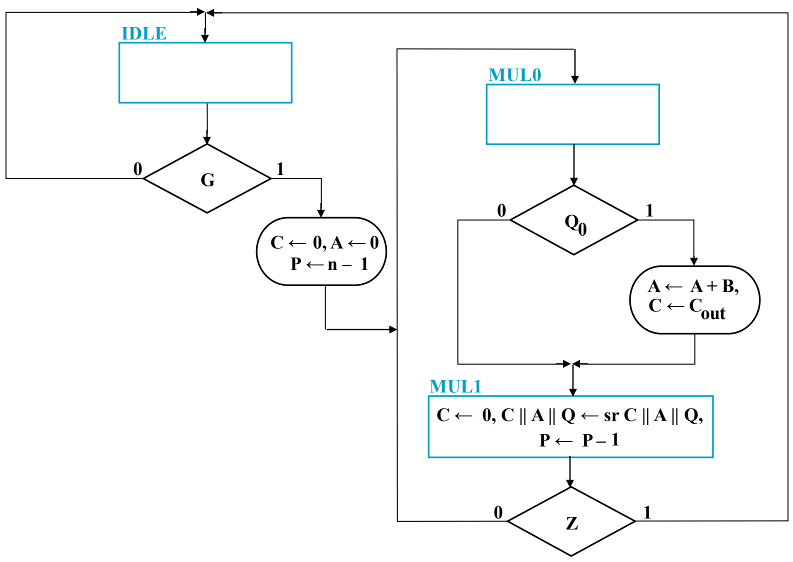
Multiplier Example: Block Diagram



Multiplier Example: Operation

1. The multiplicand is loaded into register B.
2. The multiplier is loaded into register Q.
3. When G becomes 1, register C|| A is initialized to 0.
4. Down Counter P is initialized to n – 1 (n = number of bits in multiplier)
5. The partial products are formed in register C||A||Q.
6. Each multiplier (Q) bit, beginning with the LSB, is processed (if bit is 1, B is added to partial product of A; if bit is 0, do nothing)
7. C||A||Q is shifted right using the shift register
   - Partial product bits fill vacant locations in Q as multiplier is shifted out
   - If overflow during addition, the outgoing carry is recovered from C during the right shift
8. Steps 6 and 7 are repeated until P = 0 as detected by Zero detect.

Multiplier Example: ASM Chart



7

Multiplier Example: ASM Chart (Contd.)

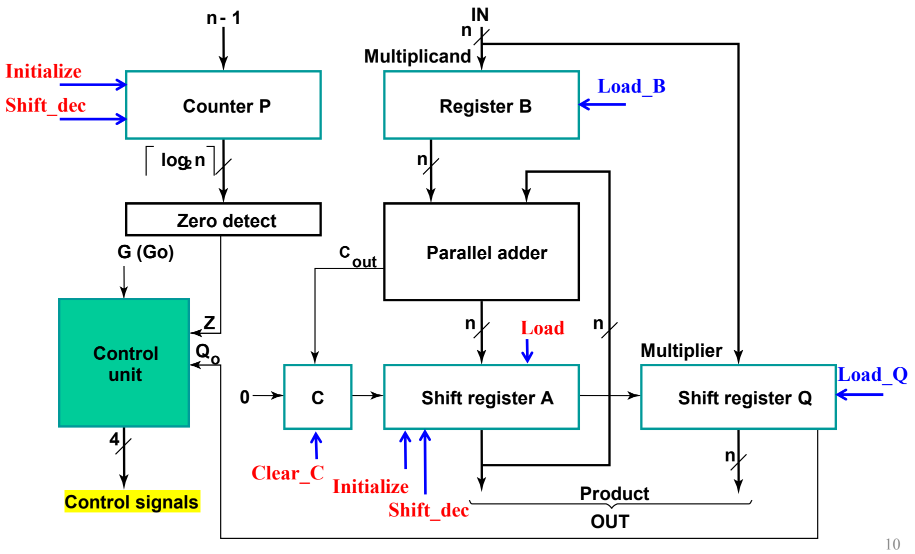
- Three states are necessary to implement multiplier
  - IDLE state:
    - Input G is used as the condition for starting the multiplication
    - C, A, and P are initialized
  - MUL0 state: conditional addition is performed based on the value of Q_0.
  - MUL1 state:
    - Right shift is performed to 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.

Control and sequencing

- The ASM chart provides information about
  - **Control** of the microoperations (control word)
  - **Sequencing** of these operations
- The design can be split up in two parts:
  - Control signals
  - Sequencing

9

Control signals for multiplier



10

Multiplier Example: Control Signal Table

Control Signals for Binary Multiplier

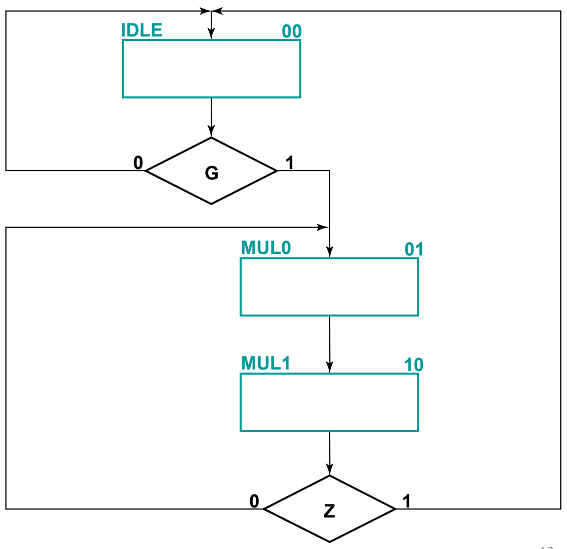
Block diagram module	Microoperation	Control signal name	Control signal expression
Register A:	$A \leftarrow 0$	Initialize	IDLE $\cdot$ G
	$A \leftarrow A + B$	Load	MUL0 $\cdot$ Q_0
	$C \parallel A \parallel Q \leftarrow$ sr $C \parallel A \parallel Q$	Shift_dec	MUL1
Register B:	$B \leftarrow IN$	*Load_B*	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
	$C \parallel A \parallel Q \leftarrow$ 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
- LOAD_Q and LOAD_B: external signals controlled from the system using the multiplier and will not be considered a part of this design
- Some control signals are "reused" for different registers.
  - Four 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 → Simplified ASM chart. Similar to a state diagram/graph but without outputs specified.



Overview

- Control and Sequencing
  - Algorithmic State Machine (ASM) Chart of Multiplier
  - Hardwired control
  - Microprogrammed control

Control

- Hardwired Control
  - Implemented using gates and flip-flops
  - Faster, less flexible, limited complexity
- Microprogram Control
  - Control Store
    - Memory storing control signals and next state info
    - Controller sequences through memory
  - Slower, more flexible, greater complexity

15

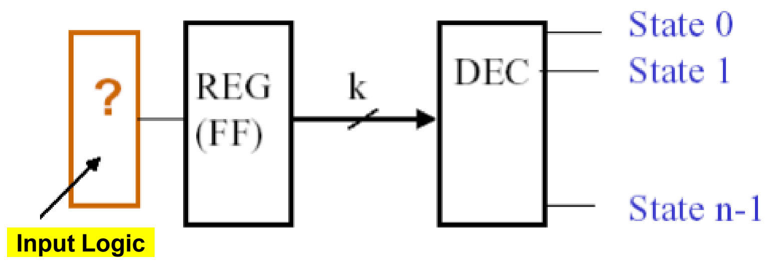
Hardwired control

- Control Design Methods
  - (1) Sequential circuit techniques; studied earlier in this course
  - Procedure specializations that use a single signal to represent each state
    - (2) 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.
    - (3) One Flip-Flop per State
      - Flip-flop outputs as "state" signals, e. g., 0001, 0010, 0100, 1000.

16

(2) Sequencer (sequence register) and Decoder

- Use a register to represent the states and a decoder to generate an output signal corresponding to each state
- Use the State Table to find the input logic



17

Multiplier Example: Sequencer and Decoder Design - Specification

- 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
  - Decide on type of flip-flops to use
- 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

18

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	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

State Table with Decoder Outputs

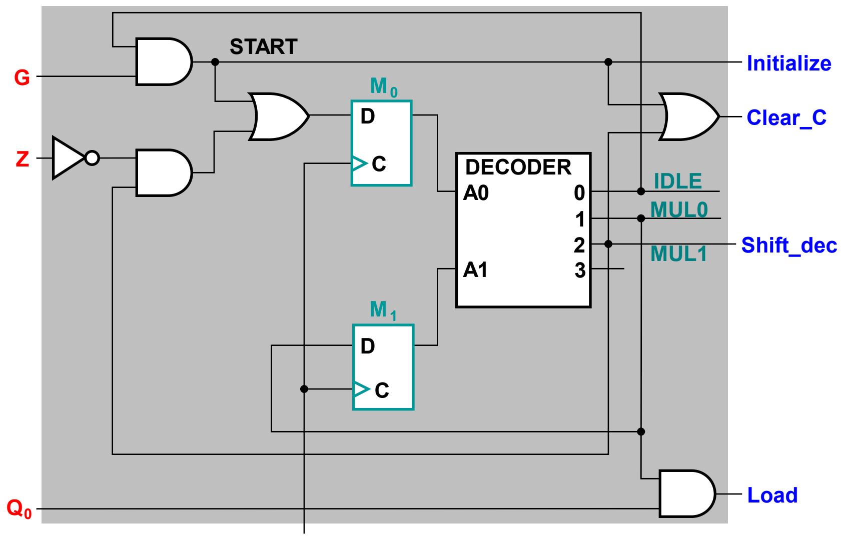
Present State			Inputs		Next State		Decoder Outputs		
Name	M1	M0	G	Z	M1	M0	Idle	MUL0	MUL1
Idle	0	0	0	X	0	0	1	0	0
	0	0	1	X	0	1	1	0	0
MUL0	0	1	X	X	1	0	0	1	0
MUL1	1	0	X	0	0	1	0	0	1
	1	0	X	1	0	0	0	0	1
-	1	1	X	X	X	X	X	X	X
					D1	D0			

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

- Finding the equations for M1 and M0 using decoded states:
  - M1 = MUL0
  - M0 = IDLE · G + MUL1 · $\overline{Z}$
- The output equations using the decoded states:
  - Initialize = IDLE · G
  - Load = MUL0 · Q_0
  - Clear_C = IDLE · G + MUL1
  - Shift_dec = MUL1
- Doing multiple level optimization, extract IDLE·G:
  - START = IDLE · G
  - M1 = MUL0
  - M0 = START + MUL1 · $\overline{Z}$
  - Initialize = START
  - Load = MUL0 · Q_0
  - Clear_C = START + MUL1
  - Shift_dec = MUL1
- The resulting circuit using flip-flops, a decoder, and the above equations is given on the next slide.

21

Multiplier Example: Sequencer and Decoder Design - Implementation



22

VHDL code: behavioral 3 processes

```vhdl
--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 binary_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;
```

VHDL code: structural

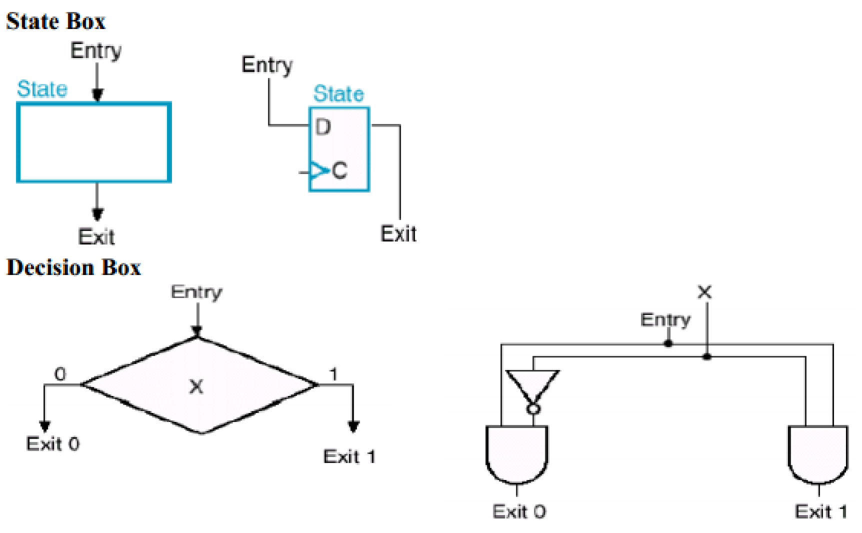
- Homework assignment #5:
  - Write VHDL description of structural architecture similar to the structural description of the architecture from Example 2, Implementation 2 of Lab #4.
  - Create a testbench and simulate in Aldec-HDL.
  - Report should contain: title, name, brief description, VHDL code (with nice indentation and useful comments throughout the code), simulation waveforms (black on white and horizontal).
  - Report should be a single PDF file named "hw5_YourFirstName_YourLastName.pdf"

(3) One Flip-Flop per State

- This method uses one flip-flop per state and a simple set of transformation rules to implement the circuit.
- The design starts with the ASM chart, and replaces
  1. State Boxes with flip-flops,
  2. Scalar Decision Boxes with a demultiplexer with 2 outputs,
  3. Vector Decision Boxes with a (partial) demultiplexer,
  4. Junctions with an OR gate, and
  5. Conditional Outputs with AND gates.
- More flip-flops needed than in previous method

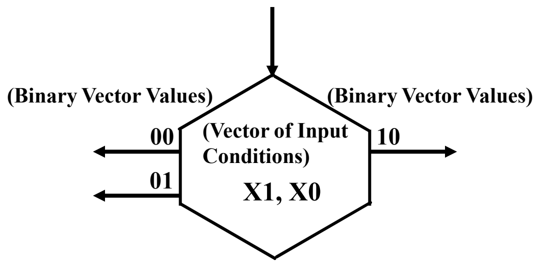
25

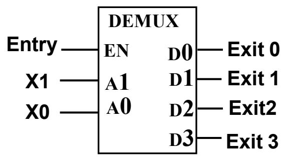
State box and Scalar decision box transformations



Vector decision box transformation

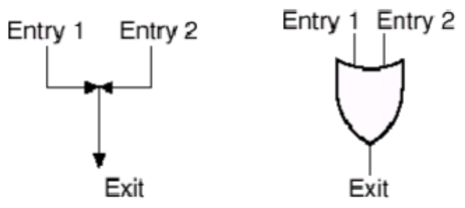
- Each Decision box transforms to a Demultiplexer
- Entry point is Enable inputs
- The Conditions are the Select inputs
- Demultiplexer Outputs are the Exit points



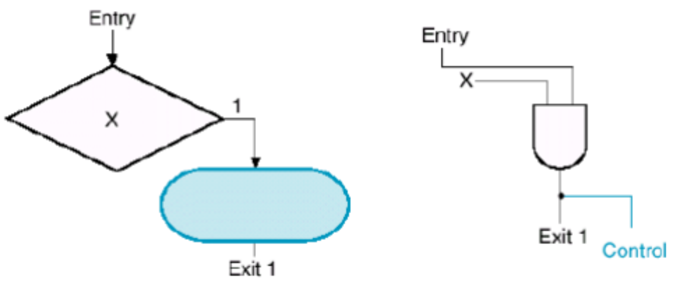


Junction transformation, Conditional output box transformation

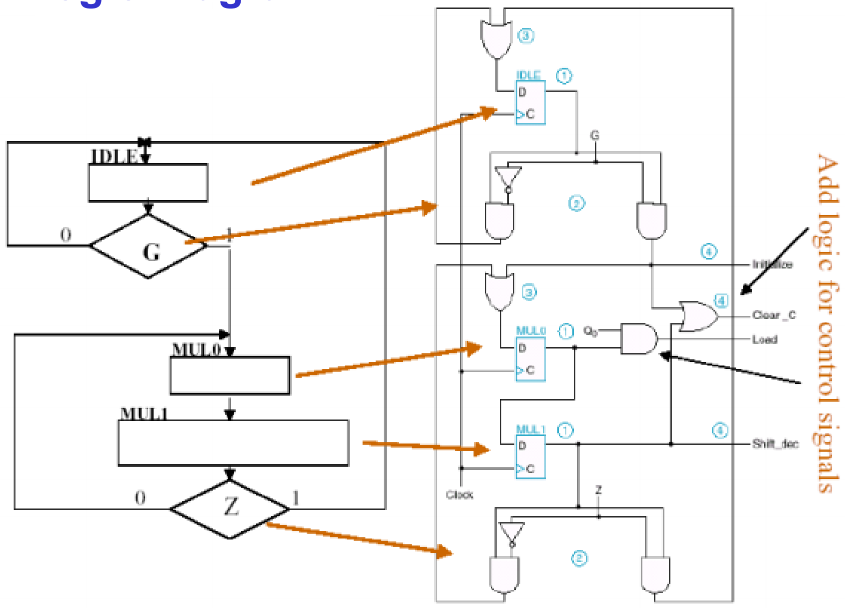
Junction



Conditional output box: use the output of the decision box as control signal



Logic Diagram



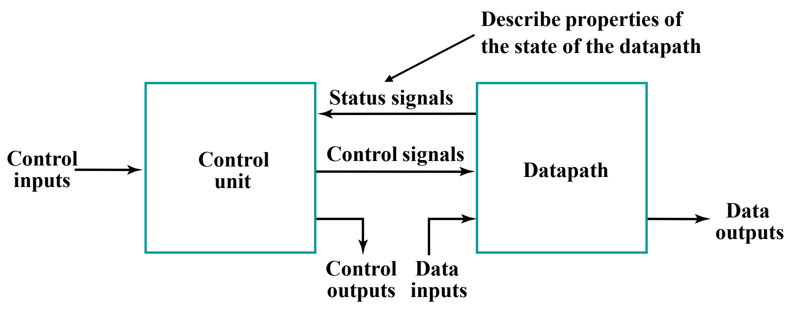
We assume that G is active for only one clock period.

Overview

- Control and Sequencing
  - Algorithmic State Machine (ASM) Chart of Multiplier
  - Hardwired control
  - Microprogrammed control

Datapath + Control unit/path

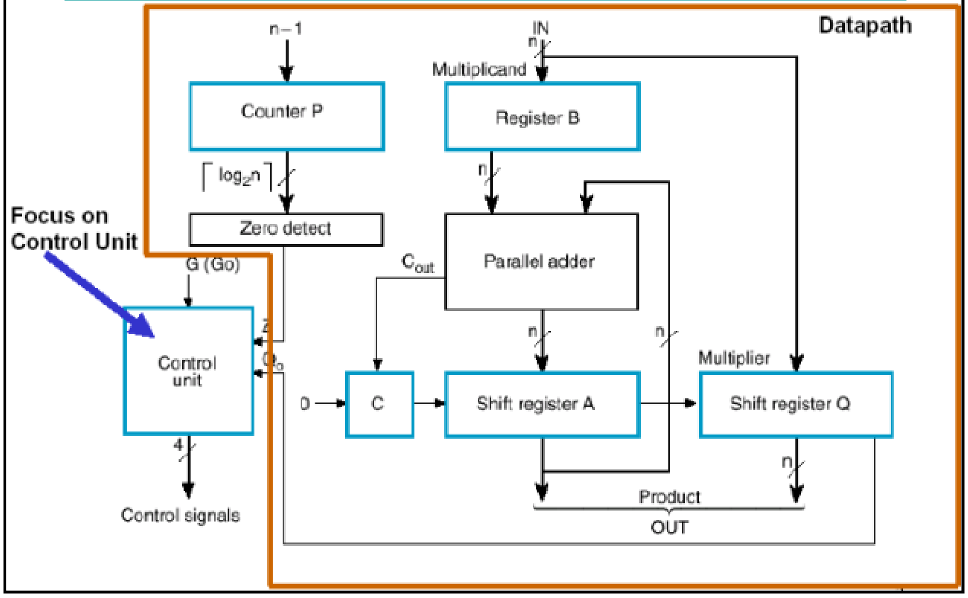
- Datapath - performs data transfer and processing operations
- Control unit/path - determines the enabling and sequencing of the operations



Datapath + Control unit/path

- Datapath:
  - Registers
  - MUXes, ALUs, Shifters, Combinational Circuits and Buses
  - Implements microoperations (under control of the control unit)
- Control unit:
  - Selects the microoperation
  - Determines the sequence (based on status and input signals)
  - Design:
    - State diagram or ASM
    - Microoperations
    - Sequence

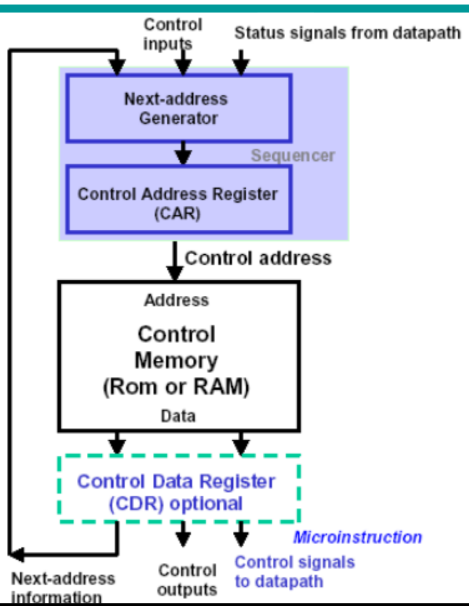
Multiplier Example



Microprogrammed Control

- **Is a control unit whose control words are stored in memory, called control memory.**
- A control word contains a **microinstruction** that specifies one or more **microoperations**.
- A sequence of microinstructions is called a **microprogram**.
- A microprogram is stored in ROM (thus fixed) or in RAM (called writable control memory).

Control Unit Organization


Control inputs
Status signals from datapath
Next-address Generator
Sequencer
Control Address Register (CAR)
Control address
Address
Control Memory (Rom or RAM)
Data
Control Data Register (CDR) optional
Microinstruction
Next-address information
Control outputs
Control signals to datapath

Microprogrammed Control

- The Control Data Register (CDR) is optional:
  - Allows higher clock frequencies (pipelining)
  - Makes the sequencing more complicated for decisions based on status bits
- If the CDR is omitted:
  - The only register in the control unit is the Control Address Register (CAR)
  - The memory and next-address generator are combinational
  - Thus the state of the control unit is given by the contents of the CAR
  - New control data will appear at the output of the memory as long as the address is applied

Microprogrammed Control

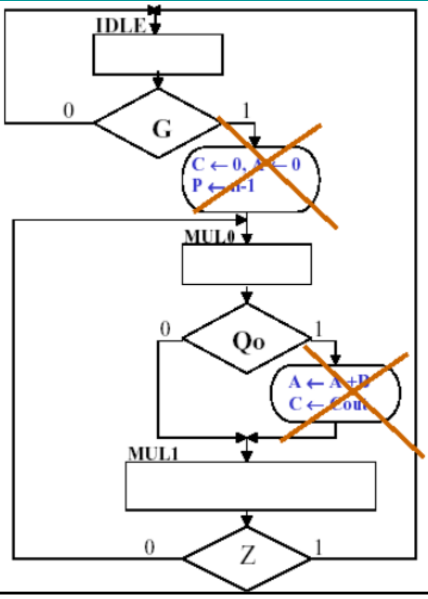
- The next address (determining the next instruction of the new state) is function of the next-address bits and the STATUS signals/bits.
- The way we designed the control unit, the status bits can only affect the next address (thus next state).

  [note: status bits do not control the datapath directly]
- Thus status bits cannot directly affect the output or cause a register transfer operation (except by affecting the address).
- This means that the sequential circuit of the control unit must be a Moore type.

ASM of the Control Unit

- Moore type circuit:
  - No conditional output boxes allowed
  - Replace the conditional output boxes by states
  - Additional states are required for the same hardware algorithm
  - Also, only one decision box between states preferred (for simple next address generation)

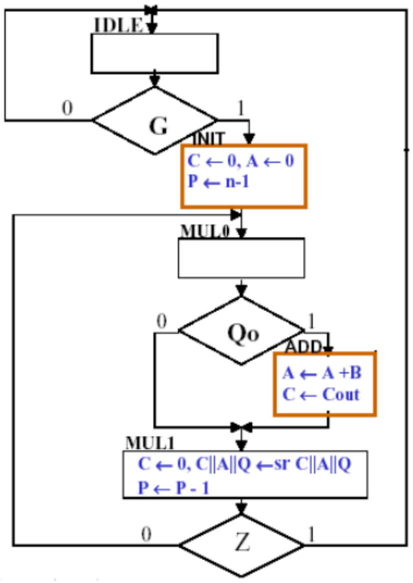
ASM of the Control Unit



39

ASM of the Control Unit

- Two additional states: INIT, ADD
- Total of 5 states needed

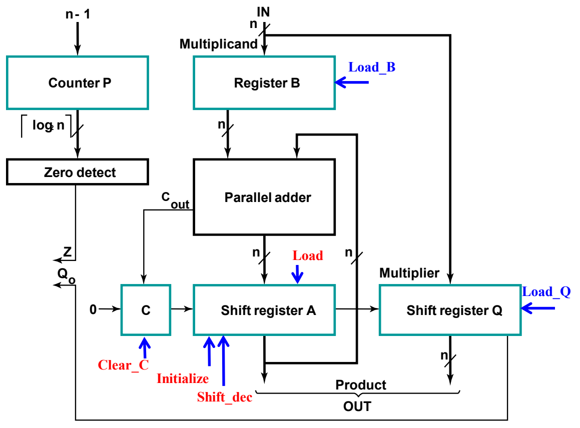


Design of the Control Unit

- To design the (micro)sequencer for the multiplier and the microprogram we need to determine:
  - The bits in the control word
  - The size of the Control memory (ROM)
  - The size of the Control Address Register (CAR)
  - Next-address generator structure
- Control word
- Sequencer

Control Signals and Datapath

- Use the same Datapath:



- We need four control signals:
  - Initialize
  - Load
  - Clear_C
  - Shift_Dec
- Status bits: Qo, Z

Control Signals and Register Transfers

- From the datapath and ASM one finds the register transfers initiated by the control signals.
- From the ASM one finds the states in which the signals are active

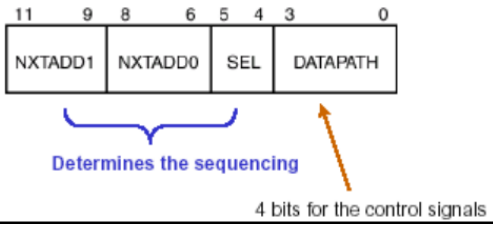
Control signal	Register transfers	States in which signal is active
Initialize	A←0, P←n-1	INIT
Load	A←A+ B, C←Cout	ADD
ClearC	C←0	INIT, MUL1
Shift/Dec	C||A||Q←srC||A||Q, P←P-1	MUL1

43

Control Signals, Control Word Format

- Four control signals needed.
- We can use these signals as is or encode them to reduce the number of bits needed in the control word.
- If we do not encode these: 4 bits needed
  - Initialize 0001
  - Load 0010
  - ClearC 0100
  - Shift/Dec 1000

Control Word Format



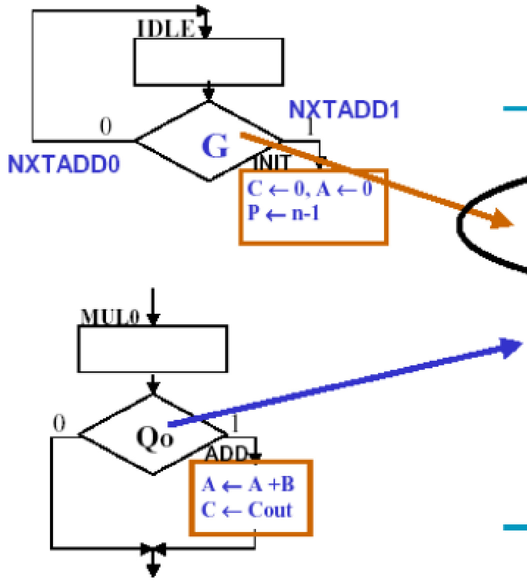
44

Sequencer

- The sequencing is determined by the ASM chart
- First, determine the sequencing requirements:
  - IDLE: next state function of G
  - MUL0: next state function of Qo
  - MUL1: next state function of Z
- We need a pair of addresses to direct to the next state depending on the values of the status or input signals
- SEL determines which next address to use

45

Control Sequences for the micro operations based on decision boxes in the ASM chart:



Sequencing Microoperations

$CAR \leftarrow NXTADD0$

$\overline{G}: CAR \leftarrow NXTADD0$

$G: CAR \leftarrow NXTADD1$

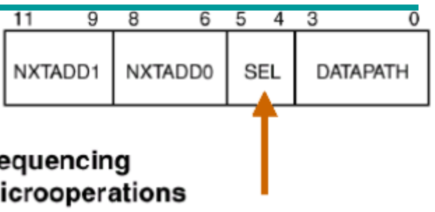
$\overline{Q_0}: CAR \leftarrow NXTADD0$

$Q_0: CAR \leftarrow NXTADD1$

$\overline{Z}: CAR \leftarrow NXTADD0$

$Z: CAR \leftarrow NXTADD1$

SEL Field definition and Code in the Control Word



SEL Field Definition:

SEL

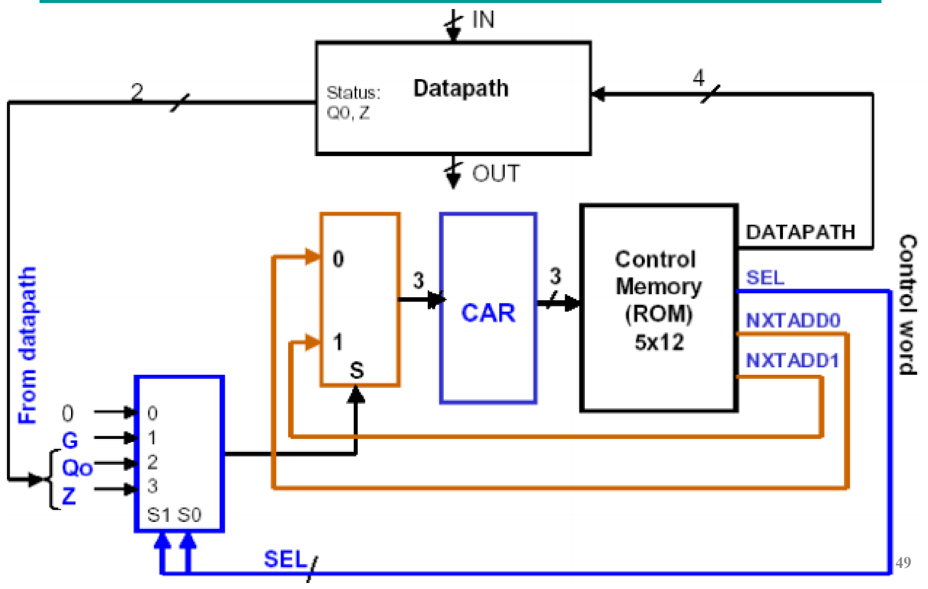
Symbolic notation	Binary Code	Sequencing Microoperations
NXT	00	$CAR \leftarrow NXTADD0$
DG	01	$\overline{G}: CAR \leftarrow NXTADD0$ $G: CAR \leftarrow NXTADD1$
DQ	10	$\overline{Q_0}: CAR \leftarrow NXTADD0$ $Q_0: CAR \leftarrow NXTADD1$
DZ	11	$\overline{Z}: CAR \leftarrow NXTADD0$ $Z: CAR \leftarrow NXTADD1$

17

Design of Control Unit

- ROM size:
  - Word length: 12 bits (control word)
  - Size: 5 storage location, one for each state
- Address bits:
  - 3 bits to address 5 locations
  - CAR is 3 bits wide
- The address loaded in the CAR:
  - Comes from the next-address info in the microinstruction: NXTADD0 or NXTADD1

Microprogrammed Control Unit



49

Register Transfer Description of the Microprogram

- Each memory location contains a microinstruction, to be executed in the corresponding state. The register transfer statements are:

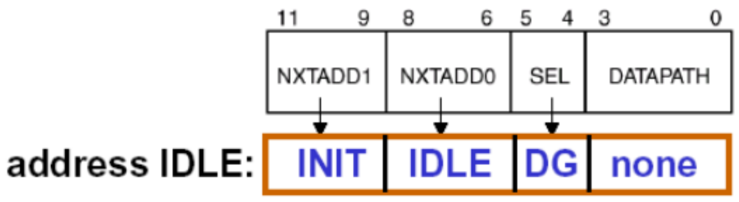
Address	Symbolic transfer statement
IDLE	G: $CAR \leftarrow \text{INIT}$, $\overline{G}$: $CAR \leftarrow \text{IDLE}$
INIT	$C \leftarrow 0, A \leftarrow 0, P \leftarrow n-1, CAR \leftarrow \text{MUL0}$
MUL0	Q_0: $CAR \leftarrow \text{ADD}, \overline{Q_0}$: $CAR \leftarrow \text{MUL1}$
ADD	$A \leftarrow A+B, C \leftarrow C_{out}, CAR \leftarrow \text{MUL1}$
MUL1	$C \leftarrow 0, C\|A\|Q \leftarrow \text{sr } C\|A\|Q$, Z: $CAR \leftarrow \text{IDLE}, \overline{Z}$: $CAR \leftarrow \text{MUL0}$, $P \leftarrow P-1$

Symbolic Microprogram

- The above Register Transfer Operation can be translated into a symbolic microprogram (control words):
- Example:

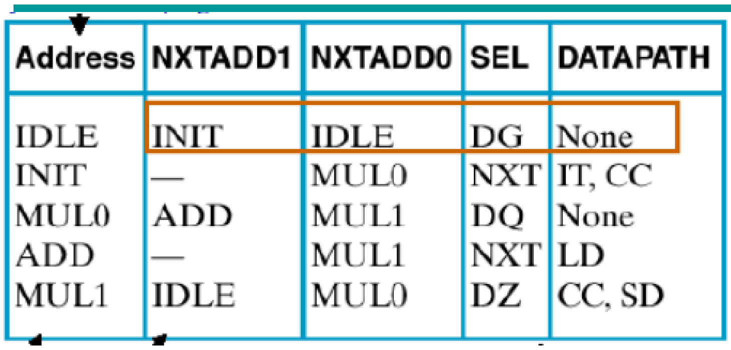
address IDLE: G: CAR ← INIT, G’: CAR ← IDLE

Can be written as:

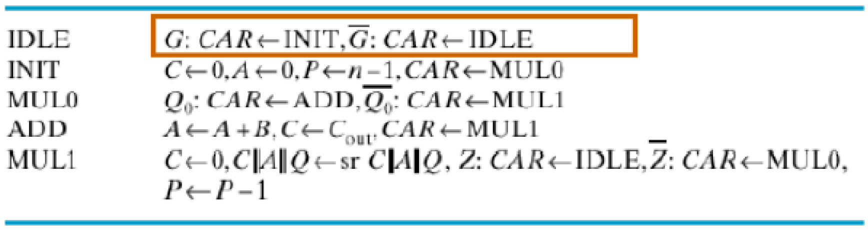


51

Symbolic Microprogram



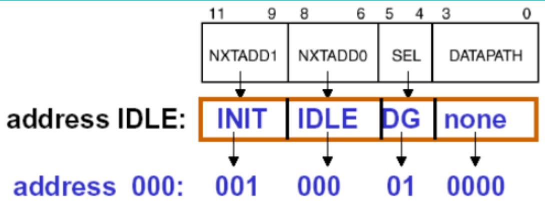
Address	NXTADD1	NXTADD0	SEL	DATAPATH
IDLE	INIT	IDLE	DG	None
INIT	—	MUL0	NXT	IT, CC
MUL0	ADD	MUL1	DQ	None
ADD	—	MUL1	NXT	LD
MUL1	IDLE	MUL0	DZ	CC, SD



IDLE	G: $CAR \leftarrow$ INIT, $\overline{G}$: $CAR \leftarrow$ IDLE
INIT	$C \leftarrow 0, A \leftarrow 0, P \leftarrow n-1, CAR \leftarrow$ MUL0
MUL0	Q_0: $CAR \leftarrow$ ADD, $\overline{Q_0}$: $CAR \leftarrow$ MUL1
ADD	$A \leftarrow A+B, C \leftarrow C_{out}, CAR \leftarrow$ MUL1
MUL1	$C \leftarrow 0, C\|A\|Q \leftarrow$ sr $C\|A\|Q$, Z: $CAR \leftarrow$ IDLE, $\overline{Z}$: $CAR \leftarrow$ MUL0, $P \leftarrow P-1$

52

Binary Microprogram



- Similar for the other instructions:

Address	NXTADD1	NXTADD0	SEL	DATAPATH	Address	NXTADD1	NXTADD0	SEL	DATAPATH
IDLE	INIT	IDLE	DG	None	000	001	000	01	0000
INIT	—	MUL0	NXT	IT, CC	001	000	010	00	0101
MUL0	ADD	MUL1	DQ	None	010	011	100	10	0000
ADD	—	MUL1	NXT	LD	011	000	100	00	0010
MUL1	IDLE	MUL0	DZ	CC, SD	100	000	010	11	1100

53

VHDL code

- Homework assignment #6:
  - Write VHDL structural description of this multiplier (slide #49) using the microprogrammed approach for the control unit.
  - Use a ROM as studied in Lab #5.
  - Create a testbench and simulate in Aldec-HDL.
  - Report should contain: title, name, brief description, VHDL code (with nice indentation and useful comments throughout the code), simulation waveforms (black on white and horizontal).
  - Report should be a single PDF file named "hw6_YourFirstName_YourLastName.pdf"

54

Summary

- Interaction between datapaths and control units
- Two types of control units:
  - Non-programmed
  - Programmed
- Two implementation approaches for Hardwired Control (non-programmed):
  - Sequence Register and Decoder
  - One Flip-Flop per state
- Use of ASM to specify control functions:
  - Microoperations
  - Sequence of operations
- Microprogrammed control is a more structured approach for complex systems

55

Appendix A: 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?

56

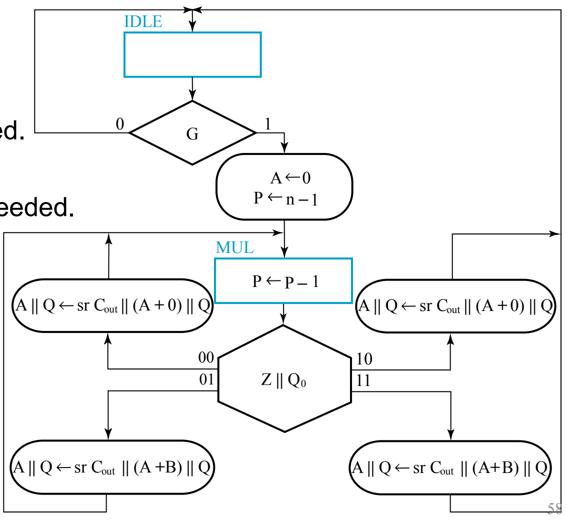
Speeding Up Multiply (Contd.)

- 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!

57

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.


IDLE
0
G
1
$A \leftarrow 0$
$P \leftarrow n-1$
MUL
$P \leftarrow P-1$
$A \,||\, Q \leftarrow \text{sr } C_{out} \,||\, (A+0) \,||\, Q$
$A \,||\, Q \leftarrow \text{sr } C_{out} \,||\, (A+0) \,||\, Q$
00
01
$Z \,||\, Q_0$
10
11
$A \,||\, Q \leftarrow \text{sr } C_{out} \,||\, (A+B) \,||\, Q$
$A \,||\, Q \leftarrow \text{sr } C_{out} \,||\, (A+B) \,||\, Q$

58