Implementation Strategies

ROM-based Design

Example: BCD to Excess 3 Serial Converter

BCD	Excess 3 Code
0000	0011
0001	0100
0010	0101
0011	0110
0100	0111
0101	1000
0110	1001
0111	1010
1000	1011
1001	1100

Conversion Process

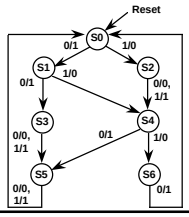
Bits are presented in bit serial fashion starting with the least significant bit

Single input X, single output Z

Implementation Strategies

Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
S0	S1	S2	1	0
S1	S3	S4	1	0
S2	S4	S4	0	1
S3	S5	S5	0	1
S4	S5	S6	1	0
S5	S0	S0	0	1
S6	S0	--	1	--

State Transition Table



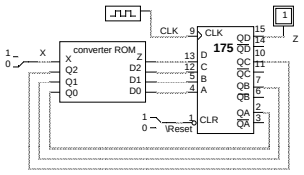
Derived State Diagram

Implementation Strategies

ROM-based Implementation

ROM Address X	Q2	Q1	Q0	ROM Outputs Z	D2	D1	D0
0	0	0	0	1	0	0	1
0	0	0	1	1	0	1	1
0	0	1	0	0	1	0	0
0	0	1	1	0	1	0	1
0	1	0	0	1	1	0	1
0	1	0	1	0	0	0	0
0	1	1	0	1	0	0	0
0	1	1	1	X	X	X	X
1	0	0	0	0	0	1	0
1	0	0	1	0	1	0	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	0	1	1	0
1	1	0	1	1	0	0	0
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Truth Table/ROM I/Os



Circuit Level Realization

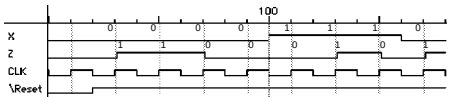
74175 = 4 x positive edge triggered D FFs

In ROM-based designs, no need to consider state assignment

Implementation Strategies

LSB MSB

Timing Behavior for input strings 0 0 0 0 (0) and 1 1 1 0 (7)



0 0 0 0 → 1 1 0 0 LSB

1 1 1 0 → 0 1 0 1 LSB

Implementation Strategies

PLA-based Design

State Assignment with NOVA

```
0 S0 S1 1
1 S0 S2 0
0 S1 S3 1
1 S1 S4 0
0 S2 S4 0
1 S2 S4 1
0 S3 S5 0
1 S3 S5 1
0 S4 S5 1
1 S4 S6 0
0 S5 S0 0
1 S5 S0 1
0 S6 S0 1
```

NOVA input file

```
S0 = 000
S1 = 001
S2 = 011
S3 = 110
S4 = 100
S5 = 111
S6 = 101
```

NOVA derived state assignment

9 product term implementation

Implementation Strategies

Espresso Inputs

```
.i 4
.o 4
.ilb x q2 q1 q0
.ob d2 d1 d0 z
.p 16
0 000 001 1
1 000 011 0
0 001 110 1
1 001 100 0
0 011 100 0
1 011 100 1
0 110 111 0
1 110 111 1
0 100 111 1
1 100 101 0
0 111 000 0
1 111 000 1
0 101 000 1
1 101 --- -
0 010 --- -
1 010 --- -
.e
```

Espresso Outputs

```
.i 4
.o 4
.ilb x q2 q1 q0
.ob d2 d1 d0 z
.p 9
0001 0100
10-0 0100
01-0 0100
1-1- 0001
-0-1 1000
0-0- 0001
-1-0 1000
--10 0100
---0 0010
.e
```

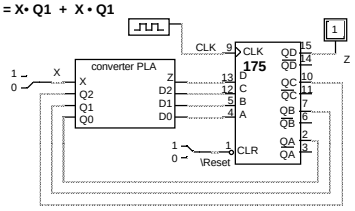
Implementation Strategies

$D2 = \overline{Q2} \cdot Q0 + Q2 \cdot \overline{Q0}$

$D1 = \overline{X} \cdot \overline{Q2} \cdot \overline{Q1} \cdot Q0 + X \cdot \overline{Q2} \cdot \overline{Q0} + \overline{X} \cdot Q2 \cdot \overline{Q0} + Q1 \cdot \overline{Q0}$

$D0 = \overline{Q0}$

$Z = X \cdot Q1 + \overline{X} \cdot \overline{Q1}$



Implementation Strategies

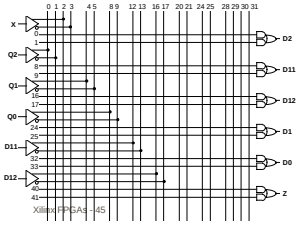
10H8 PAL: 10 inputs, 8 outputs, 2 product terms per OR gate

$D1 = D11 + D12$

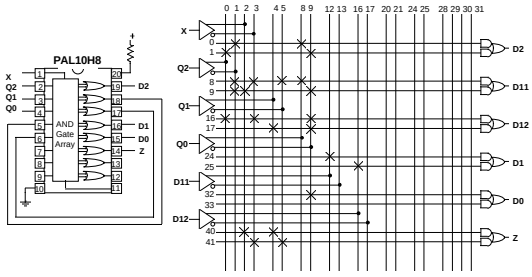
$D11 = \overline{X} \cdot \overline{Q2} \cdot \overline{Q1} \cdot Q0 + X \cdot \overline{Q2} \cdot \overline{Q0}$

$D12 = \overline{X} \cdot Q2 \cdot \overline{Q0} + Q1 \cdot \overline{Q0}$

0. $\overline{Q2} \cdot Q0$
1. $Q2 \cdot \overline{Q0}$
8. $\overline{X} \cdot \overline{Q2} \cdot \overline{Q1} \cdot Q0$
9. $X \cdot \overline{Q2} \cdot \overline{Q0}$
16. $\overline{X} \cdot Q2 \cdot \overline{Q0}$
17. $Q1 \cdot \overline{Q0}$
24. D11
25. D12
32. $\overline{Q0}$
33. not used
40. $X \cdot Q1$
41. $\overline{X} \cdot \overline{Q1}$



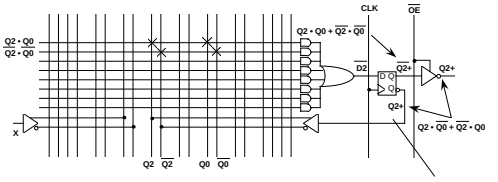
Implementation Strategies



Implementation Strategies

Registered PAL Architecture

Buffered Input or product term



$\overline{D2} = Q2 \cdot Q0 + \overline{Q2} \cdot \overline{Q0}$

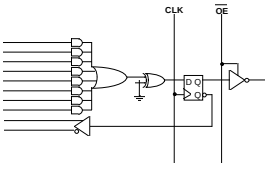
$\overline{D1} = \overline{X} \cdot \overline{Q2} \cdot \overline{Q1} \cdot \overline{Q0} + X \cdot Q2 + X \cdot Q0 + Q2 \cdot Q0 + Q1 \cdot Q0$

$\overline{D0} = Q0$

$\overline{Z} = X \cdot \overline{Q1} + \overline{X} \cdot Q1$

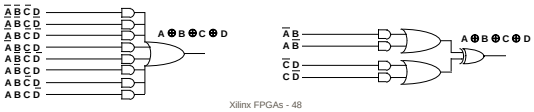
Implementation Strategies

Programmable Output Polarity/XOR PALs



Buried Registers: decouple FF from the output pin

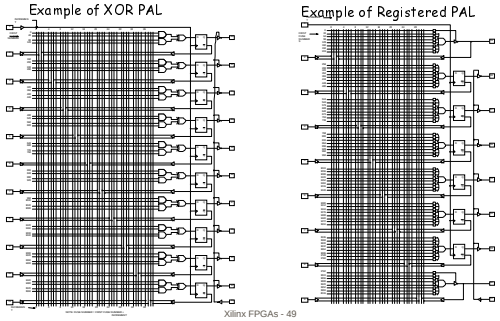
Advantage of XOR PALs: Parity and Arithmetic Operations



Implementation Strategies

Example of XOR PAL

Example of Registered PAL



Specifying PALs with ABEL

P10H8 PAL

```
module bcd2excess3
title 'BCD to Excess 3 Code Converter State Machine'
u1 device 'p10h8';

"Input Pins
 X,Q2,Q1,Q0,D11i,D12i  pin  1,2,3,4,5,6;

"Output Pins
 D2,D11o,D12o,D1,D0,Z  pin  19,18,17,16,15,14;

INSTATE = [Q2, Q1, Q0];
S0 = [0, 0, 0];
S1 = [0, 0, 1];
S2 = [0, 1, 1];
S3 = [1, 1, 0];
S4 = [1, 0, 0];
S5 = [1, 1, 1];
S6 = [1, 0, 1];

equations
 D2 = (!Q2 & Q0) # (Q2 & !Q0);
 D1 = D11i # D12i;
 D11o = (!X & !Q2 & !Q1 & Q0) # (X & !Q2 & !Q0);
 D12o = (!X & Q2 & !Q0) # (Q1 & !Q0);
 D0 = !Q0;
 Z = (X & Q1) # (!X & !Q1);

end bcd2excess3;
```

Explicit equations for partitioned output functions

Specifying PALs with ABEL

P12H6 PAL

```
module bcd2excess3
title 'BCD to Excess 3 Code Converter State Machine'
u1 device  'p12h6';

"Input Pins
 X, Q2, Q1, Q0  pin  1, 2, 3, 4;

"Output Pins
 D2, D1, D0, Z  pin  17, 18, 16, 15;

INSTATE = [Q2, Q1, Q0]; OUTSTATE = [D2, D1, D0];
S0in = [0, 0, 0];  S0out = [0, 0, 0];
S1in = [0, 0, 1];  S1out = [0, 0, 1];
S2in = [0, 1, 1];  S2out = [0, 1, 1];
S3in = [1, 1, 0];  S3out = [1, 1, 0];
S4in = [1, 0, 0];  S4out = [1, 0, 0];
S5in = [1, 1, 1];  S5out = [1, 1, 1];
S6in = [1, 0, 1];  S6out = [1, 0, 1];

equations
 D2 = (!Q2 & Q0) # (Q2 & !Q0);
 D1 = (!X & !Q2 & !Q1 & Q0) # (X & !Q2 & !Q0) #
      (!X & Q2 & !Q0) # (Q1 & !Q0);
 D0 = !Q0;
 Z = (X & Q1) # (!X & !Q1);

end bcd2excess3;
```

Simpler equations

Specifying PALs with ABEL

P16R4 PAL

```
module bcd2excess3
title 'BCD to Excess 3 Code Converter'
u1 device 'p16r4';

"Input Pins
 Clk, Reset, X, !OE     pin  1, 2, 3, 11;

"Output Pins
 D2, D1, D0, Z     pin 14, 15, 16, 13;

SREG = [D2, D1, D0];
S0 = [0, 0, 0];
S1 = [0, 0, 1];
S2 = [0, 1, 1];
S3 = [1, 1, 0];
S4 = [1, 0, 0];
S5 = [1, 1, 1];
S6 = [1, 0, 1];
```

```
state_diagram SREG
state S0: if Reset then S0
      else if X then S2 with Z = 0
      else S1 with Z = 1
state S1: if Reset then S0
      else if X then S4 with Z = 0
      else S3 with Z = 1
state S2: if Reset then S0
      else if X then S4 with Z = 1
      else S4 with Z = 0
state S3: if Reset then S0
      else if X then S5 with Z = 1
      else S5 with Z = 0
state S4: if Reset then S0
      else if X then S6 with Z = 0
      else S5 with Z = 1
state S5: if Reset then S0
      else if X then S0 with Z = 1
      else S0 with Z = 0
state S6: if Reset then S0
      else if !X then S0 with Z = 1

end bcd2excess3;
```


FSM Design with Counters

Synchronous Counters: CLR, LD, CNT

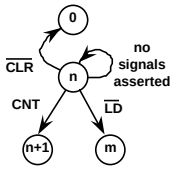
Four kinds of transitions for each state:

(1) to State 0 (CLR)

(2) to next state in sequence (CNT)

(3) to arbitrary next state (LD)

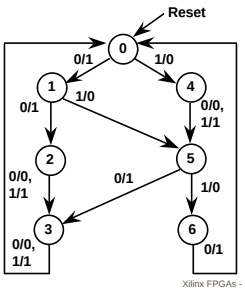
(4) loop in current state



Careful state assignment is needed to reflect basic sequencing of the counter

FSM Design with Counters

Excess 3 Converter Revisited



Note the sequential nature of the state assignments

FSM Design with Counters

Excess 3 Converter

Inputs/Current State				Next State			Outputs						
X	Q2	Q1	Q0	Q2+	Q1+	Q0+	Z	CLR	LD	EN	C	B	A
0	0	0	0	0	0	1	1	1	1	1	X	X	X
0	0	0	1	0	1	0	1	1	1	1	X	X	X
0	0	1	0	0	1	1	0	1	1	1	X	X	X
0	0	1	1	0	0	0	0	0	X	X	X	X	X
0	1	0	0	1	0	1	1	1	1	1	X	X	X
0	1	0	1	0	1	1	0	1	0	X	0	1	0
0	1	1	0	0	0	0	1	0	X	X	X	X	X
0	1	1	1	X	X	X	X	X	X	X	X	X	X
1	0	0	0	1	0	0	0	1	0	X	1	0	0
1	0	0	1	1	0	1	0	1	0	X	1	0	1
1	0	1	0	0	1	1	1	1	1	1	X	X	X
1	0	1	1	0	0	0	1	0	X	X	X	X	X
1	1	0	0	1	0	1	0	1	1	1	X	X	X
1	1	0	1	1	1	0	1	1	1	1	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X	X	X

CLR signal dominates LD which dominates Count

Implementing FSMs with Counters

Espresso Input File

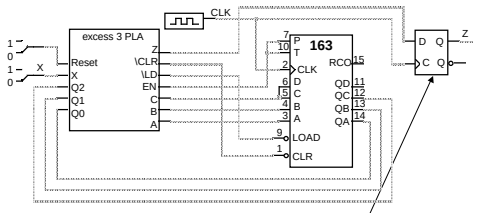
```
.i 5
.o 7
.ilb res x q2 q1 q0
.ob z clr ld en c b a
.p 17
1---- -0-----
00000 1111---
00001 1111---
00010 0111---
00011 00-----
00100 0111---
00101 110-011
00110 10-----
00111 -------
01000 010-100
01001 010-101
01010 1111---
01011 10-----
01100 1111---
01101 0111---
01110 -------
01111 -------
.e
```

Excess 3 Converter

```
.i 5
.o 7
.ilb res x q2 q1 q0
.ob z clr ld en c b a
.p 10
0-001 0101101
-0-01 1000000
-11-0 1000000
0-0-0 0101100
-000- 1010000
-0--0 0010000
0-10- 0101011
--11- 1000000
-11-- 0010000
-1-1- 1010000
.e
```

Espresso Output File

FSM Implementation with Counters



Excess 3 Converter Schematic

Synchronous Output Register

Implementation Strategies

Xilinx LCA Architecture

Implementing the BCD to Excess 3 FSM

$Q2+ = \overline{Q2} \cdot Q0 + Q2 \cdot \overline{Q0}$

$Q1+ = \overline{X} \cdot \overline{Q2} \cdot \overline{Q1} \cdot Q0 + X \cdot \overline{Q2} \cdot \overline{Q0} + \overline{X} \cdot Q2 \cdot \overline{Q0} + Q1 \cdot \overline{Q0}$

$Q0+ = \overline{Q0}$

$Z = Z \cdot Q1 + \overline{X} \cdot \overline{Q1}$

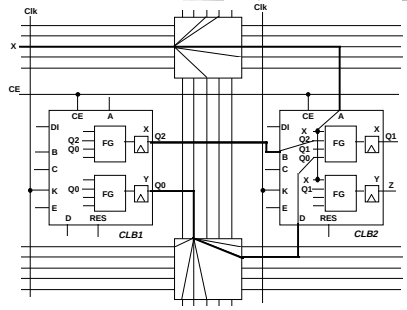
No function more complex than 4 variables
4 FFs implies 2 CLBs

Synchronous Mealy Machine

Global Reset to be used

Place Q2+, Q0+ in once CLB
Q1, Z in second CLB
maximize use of direct & general purpose interconnections

Implementing the BCD to Excess 3 FSM



Design Case Study

Traffic Light Controller

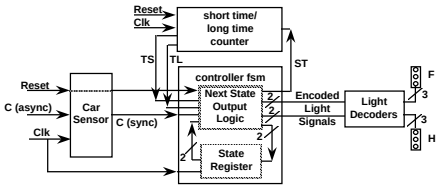
Decomposition into primitive subsystems

- Controller FSM
  next state/output functions
  state register
- Short time/long time interval counter
- Car Sensor
- Output Decoders and Traffic Lights

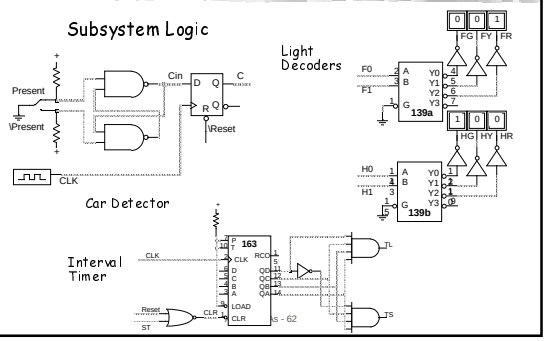
Design Case Study

Traffic Light Controller

Block Diagram



Design Case Study



Design Case Study

State Assignment: HG = 00, HY = 10, FG = 01, FY = 11

$P1 = C\ TL\ \overline{Q1} + \overline{TS}\ Q1\ \overline{Q0} + \overline{C}\ Q1\ Q0 + \overline{TS}\ Q1\ Q0$

$P0 = TS\ Q1\ \overline{Q0} + \overline{Q1}\ Q0 + \overline{TS}\ Q1\ Q0$

$ST = C\ TL\ \overline{Q1} + \overline{C}\ Q1\ Q0 + TS\ Q1\ \overline{Q0} + TS\ Q1\ Q0$

$HL[1] = TS\ Q1\ Q0 + \overline{Q1}\ Q0 + \overline{TS}\ Q1\ Q0$

$HL[0] = \overline{TS}\ Q1\ \overline{Q0} + TS\ Q1\ \overline{Q0}$

Next State Logic

$FL[1] = \overline{Q0}$

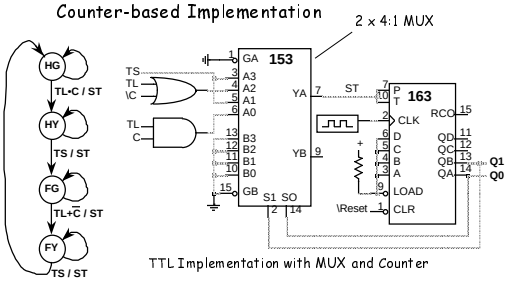
$FL[0] = TS\ Q1\ Q0 + \overline{TS}\ Q1\ Q0$

PAL/PLA Implementation:
- 5 inputs, 7 outputs, 8 product terms
- PAL 22V10 -- 11 inputs, 10 prog. IOs, 8 to 14 prod terms per OR

ROM Implementation:
- 32 word by 8-bit ROM (256 bits)
- Reset may double ROM size

Design Case Study

Counter-based Implementation



ST = Count

TTL Implementation with MUX and Counter

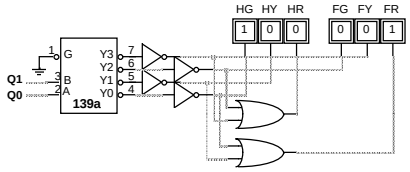
Can we reduce package count by using an 8:1 MUX?

Design Case Study

Counter-based Implementation

Dispense with direct output functions for the traffic lights

Why not simply decode from the current state?



ST is a Synchronous Mealy Output

Light Controllers are Moore Outputs

Design Case Study

LCA-Based Implementation

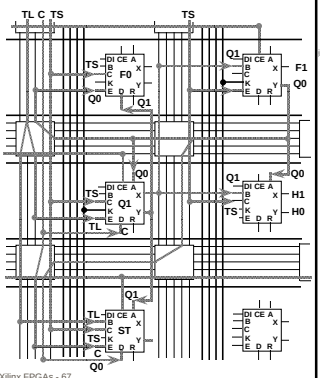
Discrete Gate Method:

- None of the functions exceed 5 variables
- P1, ST are 5 variable (1 CLB each)
- P0, HL1, HL0, FL0 are 3 variable (1/2 CLB each)
- FL1 is 1 variable (1/2 CLB)
- 4 1/2 CLBs total!

Design Case Study

LCA-Based Implementation

Placement of functions selected to maximize the use of direct connections



Design Case Study

LCA-Based Implementation

Counter/Multiplexer Method:

4:1 MUX, 2 Bit Upcounter

MUX: six variables (4 data, 2 control)
but this is the kind of 6 variable function that can be implemented in 1 CLB!

2nd CLB to implement TL • C and TL + C'

But note that ST/Cnt is really a function of TL, C, TS, Q1, Q0
1 CLB to implement this function of 5 variables!

2 Bit Counter: 2 functions of 3 variables (2 bit state + count)
Also implemented in one CLB

Traffic light decoders: functions of 2 variables (Q1, Q0)
2 per CLB = 3 CLB for the six lights

Total count = 5 CLBs