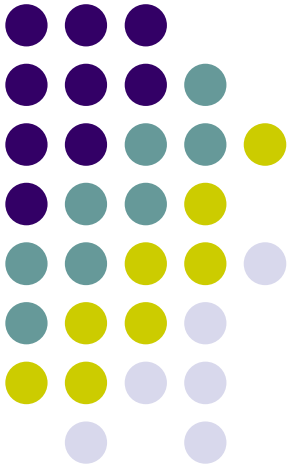
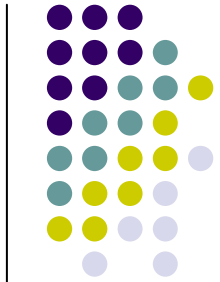
CSE 140 Discussion

Chengmo Yang
Mingjing Chen
06/01/09



Overflow



Prove that for two's complement number system arithmetic, the overfow of the addition is determined by the last two carry bits, i.e. *overflow flag* = $c_n \oplus c_{n-1}$.

- Adding two positive numbers → overflow iff $s_{n-1} = 1$

  $a_{n-1} = b_{n-1} = 0$ → c_n always = 0, $s_{n-1} = 1$ iff $c_{n-1} = 1$

  → overflow iff $c_n \oplus c_{n-1} = 1$

- Adding two negative numbers → overflow iff $s_{n-1} = 0$

  $a_{n-1} = b_{n-1} = 1$ → c_n always = 1, $s_{n-1} = 0$ iff $c_{n-1} = 0$

  → overflow iff $c_n \oplus c_{n-1} = 1$

- Adding two numbers with different signs → never overflow **Why?**

  $a_{n-1} \neq b_{n-1}$ → c_n always = c_{n-1} → $c_n \oplus c_{n-1} = 0$

Overflow cases:

3 + 5 = 8

0	1	1	1
	0	0	1	1
+	0	1	0	1
	1	0	0	0

-4 + -5 = -9

1	0	1	1
	1	1	0	0
+	1	0	1	1
	0	1	1	1

Carry Look Ahead (CLA) Adder

A carry look ahead adder inputs two-bit numbers (a1; a0) and (b1; b0), and a carry in c0. Use a minimal two-level NAND gate network to implement the carry out c2.

$c1 = a0b0 + (a0+b0)c0$

$c2 = a1b1 + (a1+b1)c1$
$= a1b1 + (a1+b1)a0b0 + (a1+b1)(a0+b0)c0$
$= a1b1 + a1a0b0 + b1a0b0 + a1a0c0 + b1a0c0 + a1b0c0 + b1b0c0$

1st level: one 2-input NAND and six 3-input NAND
2nd level: one 7-input NAND

Simplification: use 3-level logic

1st level: $p1 = a1+b1, g1 = a1b1, p0 = a0+b0, g0 = a0b0,$

2st and 3rd levels: $c2 = g1 + p1g0 + p1p0c0$

Total gates: three 2-input AND, one 3-input AND
two 2-input OR, one 3-input OR

Carry Look Ahead (CLA) Adder

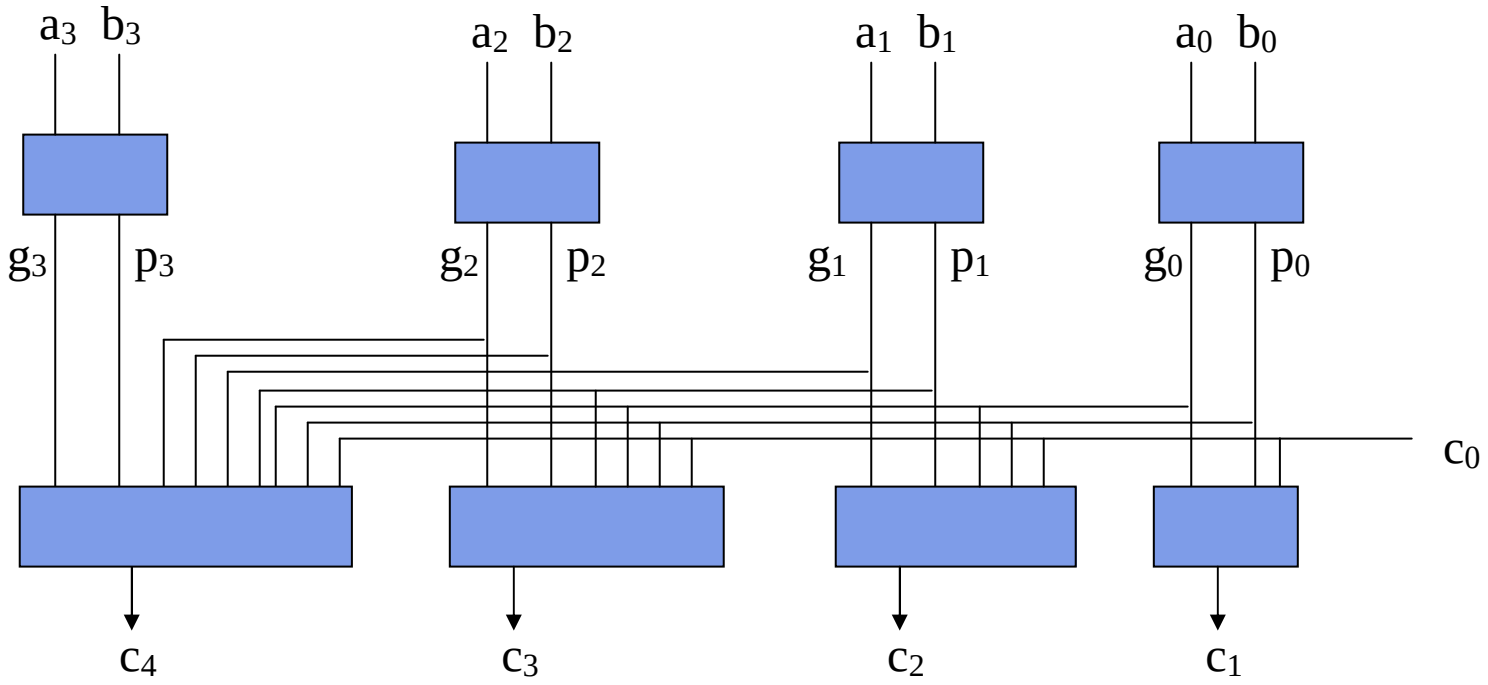
$c_1 = g_0 + p_0c_0$

$c_2 = g_1 + p_1c_1 = g_1 + p_1g_0 + p_1p_0c_0$

$c_3 = g_2 + p_2c_2 = g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0c_0$

$c_4 = g_3 + p_3c_3 = \underbrace{g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0}_{G_{30}} + \underbrace{p_3p_2p_1p_0}_{P_{30}}c_0$

$g_i = a_ib_i \qquad p_i = a_i + b_i$



Carry Look Ahead (CLA) Adder

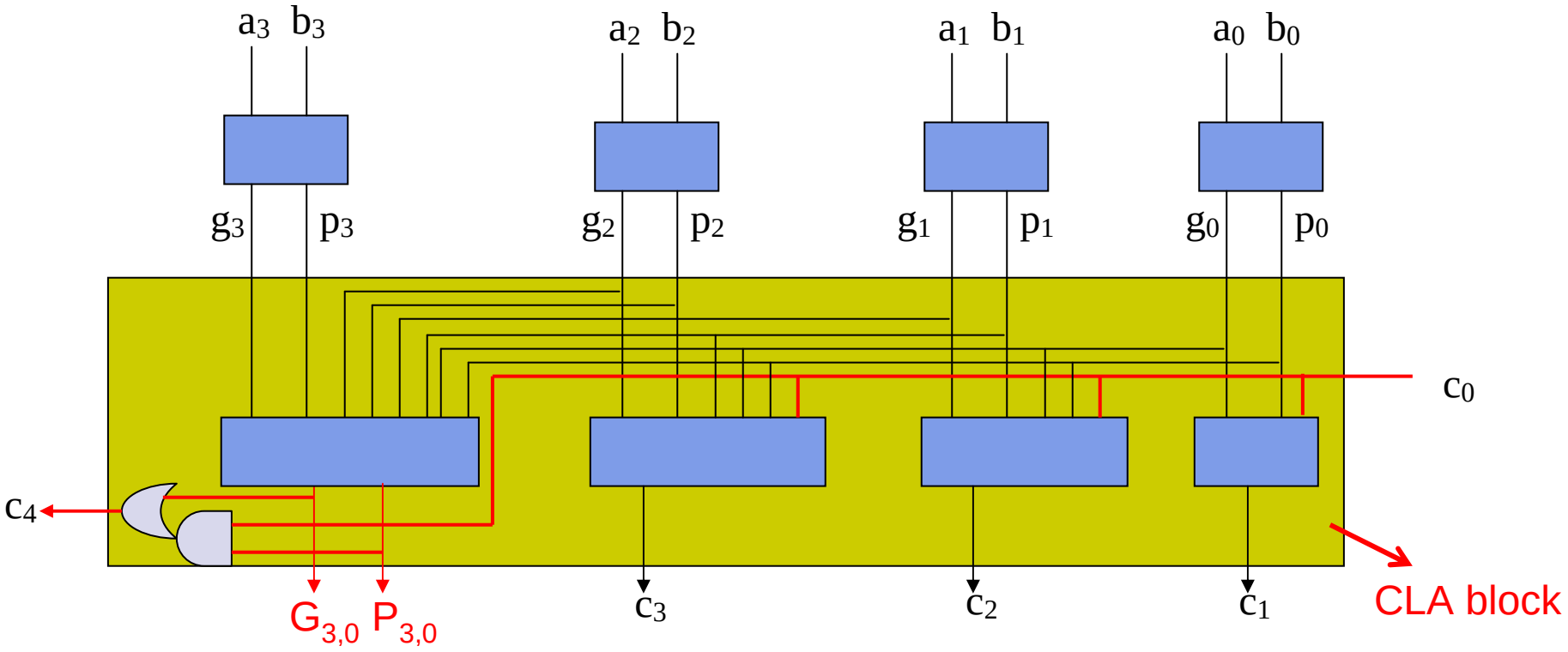
$$c_1 = g_0 + p_0c_0$$

$$c_2 = g_1 + p_1c_1 = g_1 + p_1g_0 + p_1p_0c_0$$

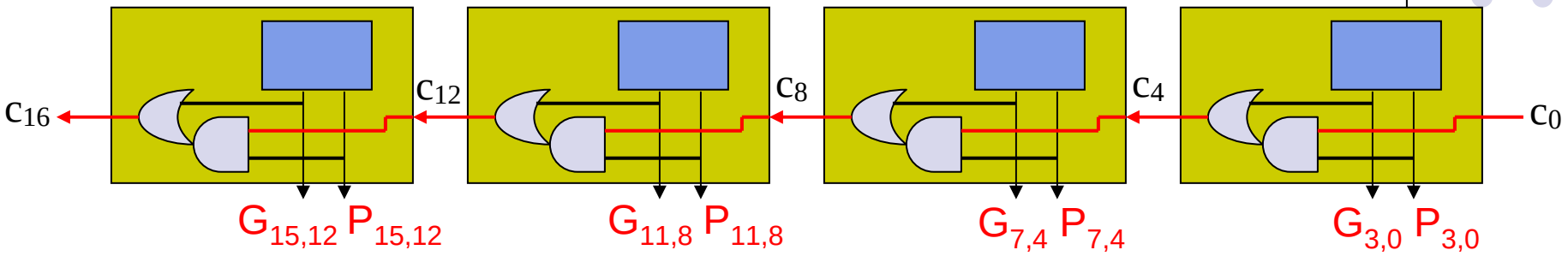
$$c_3 = g_2 + p_2c_2 = g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0c_0$$

$$c_4 = g_3 + p_3c_3 = \underbrace{g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0}_{G_{30}} + \underbrace{p_3p_2p_1p_0}_{P_{30}}c_0$$

$$g_i = a_ib_i \qquad p_i = a_i + b_i$$



CLA Adder Delay



- Step 1: compute g_i and p_i signals for each single bit in parallel
- Step 2: compute G and P for 4-bit blocks
- Step 3: C_0 propagates through each 4-bit CLA block
- Step 4: compute sum, $s_i = p_i \oplus c_i$
- **Longest path: a_0 to s_{15}**

CLA Adder Delay

- Delay of an N-bit carry-lookahead adder with k-bit blocks:

$$t_{CLA} = t_{pg} + t_{pg\_block} + (N/k - 1)t_{AND\_OR} + kt_{FA}$$

where

  - t_{pg} : delay of the column generate and propagate gates
  - t_{pg_block} : delay of the block generate and propagate gates
  - t_{AND_OR} : delay from C_{in} to C_{out} of the final AND/OR gate in the k-bit CLA block

- An N-bit carry-lookahead adder is generally much faster than a ripple-carry adder for $N > 16$

Subtracter

A subtracter inputs a two-bit number (x1; x0), a subtrahend (y1; y0) and a borrow-in bit b0, and outputs the dierence (d1; d0) and a borrow-out bit b2. Write the boolean expression of borrow-out bit b2 as a function of variables x1; x0; y1; y0; b0.

Full subtracter truth table

x_i	y_i	b_i	d_i	b_{i+1}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

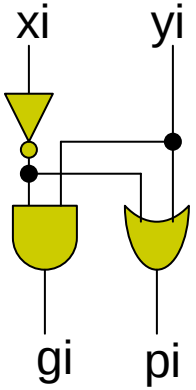
$$d_i = x_i \oplus y_i \oplus b_i$$
$$b_{i+1} = x_i'y_i + x_i'b_i + y_ib_i$$

Therefore we have:

$$b1 = x0'y0 + (x0'+y0)b0$$
$$b2 = x1'y1 + (x1'+y1)b1$$
$$= x1'y1 + (x1'+y1)x0'y0 + (x1'+y1)(x0'+y0)b0$$

Subtracter

Use two full adders and a minimal number of AND, OR, NOT gates to implement a look-ahead subtracter. Draw the schematic diagram.

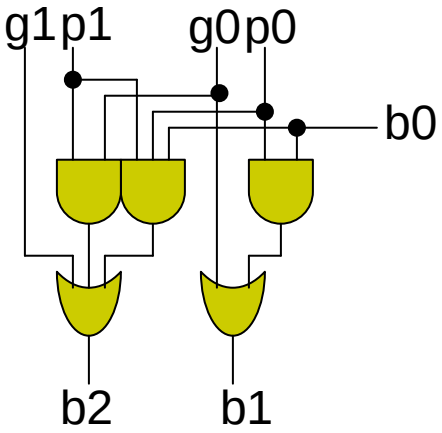


The whole circuit is composed of 3 parts:

Step1: generate g1, g0 and p1, p0 in parallel.

$g_i = x_i' y_i \qquad p_i = x_i' + y_i$

Step2: generate b1 and b2 in parallel.



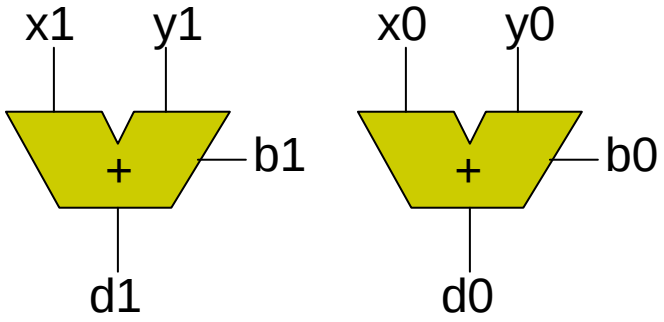
$b_1 = g_0 + p_0 b_0$

$b_2 = g_1 + p_1 g_0 + p_1 p_0 b_0$

Step3: generate d0 and d1 using full adders.

$d_0 = x_0 \oplus y_0 \oplus b_0$

$d_1 = x_1 \oplus y_1 \oplus b_1$



Sequential Adder

A sequential adder inputs a_i; b_i, the i^{th} bit of two binary numbers in each clock cycle for i = 0 to n-1 and outputs the sum s_i. Implement the adder with a JK flip-flop, and a minimal AND-OR-NOT network (if the network is needed). Draw the schematic diagram.

$c_{i+1} = a_ib_i + a_ic_i + b_ic_i$

$s_i = a_i'b_i'c_i + a_i'b_ic_i' + a_ib_i'c_i' + a_ib_ic_i$

Where is c_i?

Stored in the JK flip flop

Full adder truth table:

a_i	b_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

FSM state table:

a_i	b_i	CS	NS	s_i	J	K
0	0	0	0	0	0	X
0	0	1	0	1	X	1
0	1	0	0	1	0	X
0	1	1	1	0	X	0
1	0	0	0	1	0	X
1	0	1	1	0	X	0
1	1	0	1	0	1	X
1	1	1	1	1	X	0

$J = a_ib_i$

$K = a_i'b_i'$

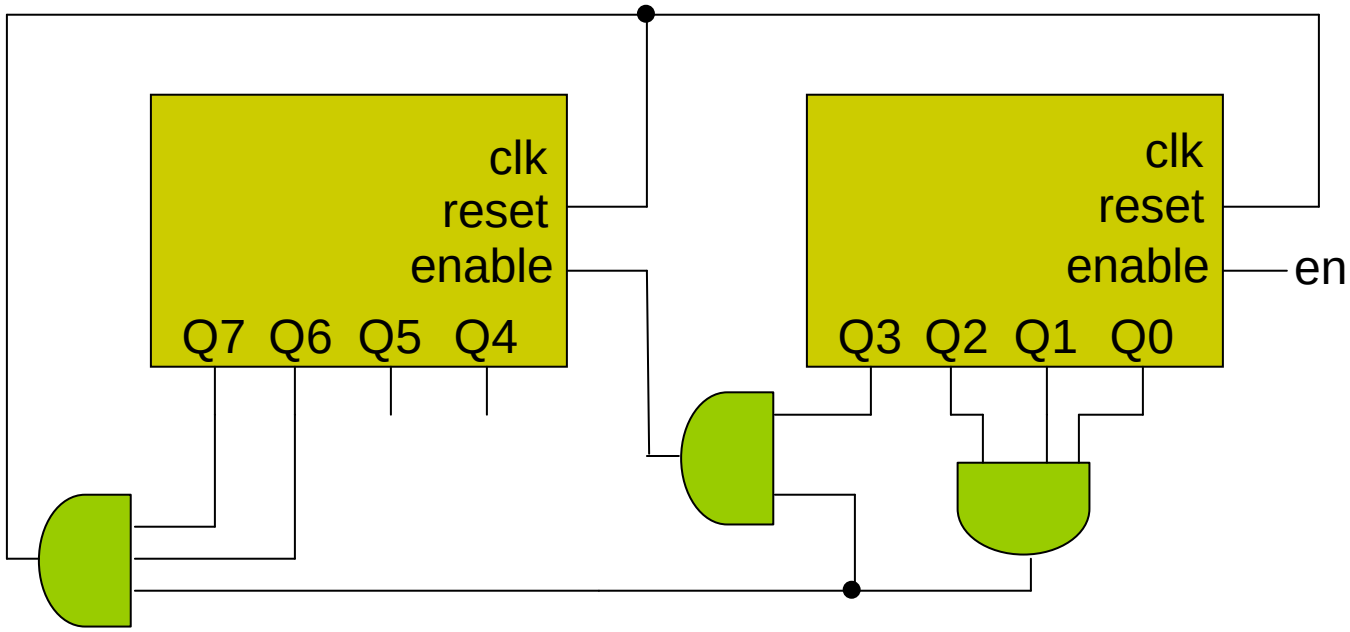
6.5. Counter

Counters: Given ***modulo-16 counters***, draw the logic diagram to show the following designs

- 6.5.1 Design a module-200 counter with a repeated output

Count through 0 to 199
$(199)_{10} = (11000111)_2$

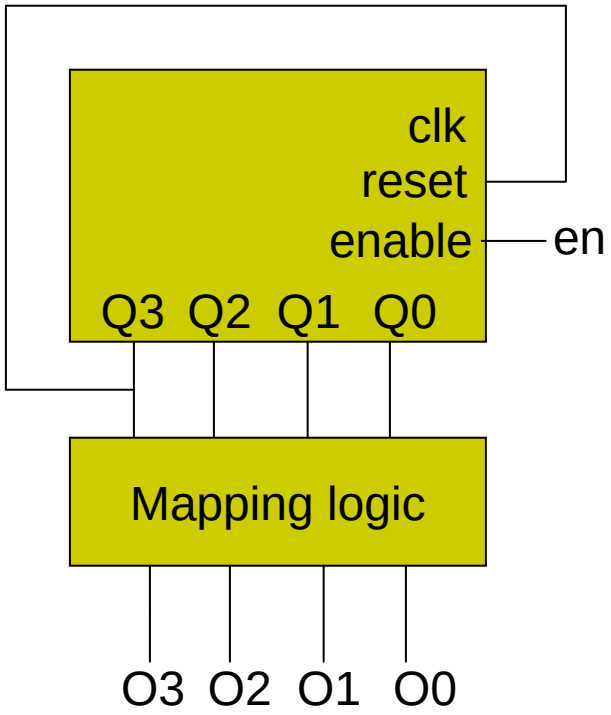
Reset the counter after it counts to 199



6.5. Counter

- 6.5.2 Design a counter with a repeated output sequence **15, 0, 1, 2, 8, 9, 10, 6, 7,** with a **modulo-16 counter** and a minimal combinational network

9-cycle sequence ➔ count from 0 to 8 and then reset
Need to map the counter output to the number sequence



Mapping logic

Q	O
0	0
1	1
2	2
3	8
4	9
5	10
6	6
7	7
8	15

6.5.2 Counter (Continued)

Mapping logic

Q	O
0	0
1	1
2	2
3	8
4	9
5	10
6	6
7	7
8	15

Truth table

Q3	Q2	Q1	Q0	O3	O2	O1	O0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	1	0	0	0
0	1	0	0	1	0	0	1
0	1	0	1	1	0	1	0
0	1	1	0	0	1	1	0
0	1	1	1	0	1	1	1
1	0	0	0	1	1	1	1

Kmap for O0

Q1Q0 \ Q3Q2	00	01	11	10
00	0	1	X	1
01	1	0	X	X
11	0	1	X	X
10	0	0	X	X

$O_0 = Q_2Q_1'Q_0' + Q_3 + Q_2'Q_1'Q_0 + Q_2Q_1Q_0$

Kmap for O3

Q1Q0 \ Q3Q2	00	01	11	10
00	0	1	X	1
01	0	1	X	X
11	1	0	X	X
10	0	0	X	X

$O_3 = Q_2Q_1' + Q_3 + Q_2'Q_1Q_0$

Kmap for O2

Q1Q0 \ Q3Q2	00	01	11	10
00	0	0	X	1
01	0	0	X	X
11	0	1	X	X
10	0	1	X	X

$O_2 = Q_2Q_1 + Q_3$

Kmap for O1

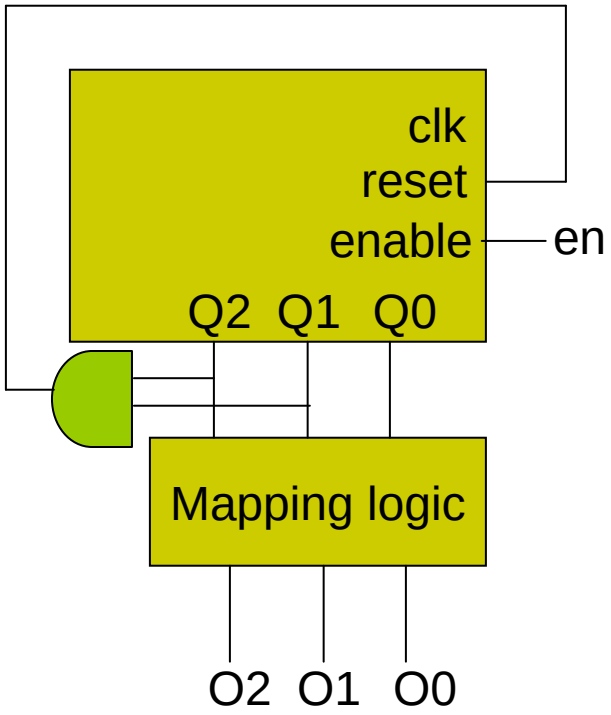
Q1Q0 \ Q3Q2	00	01	11	10
00	0	0	X	1
01	0	1	X	X
11	0	1	X	X
10	1	1	X	X

$O_1 = Q_2Q_0 + Q_3 + Q_1Q_0'$

6.6 Counter

- Design a counter with a repeated output sequence 0, 1, 2, 4, 5, 6, 3, with a modulo-8 counter and a minimal AND-OR-NOT network

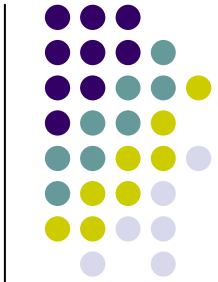
7-cycle sequence ➔ count from 0 to 6 and then reset
Need to map the counter output to the number sequence



Mapping logic

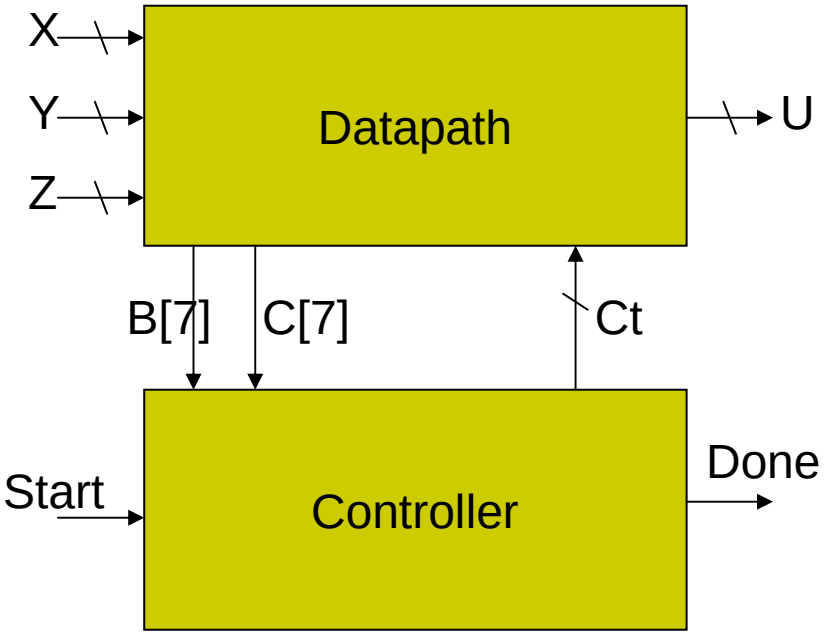
Q	O
0	0
1	1
2	2
3	4
4	5
5	6
6	3

6.7 System Design



- ***Alg(X, Y, Z, start, U, done);***
- ***Input X[7 : 0], Y [7 : 0], Z[7 : 0], start;***
- ***Output U[7 : 0], done;***
- Local-object A[7 : 0], B[7 : 0], C[7 : 0];
- S1: If start’ goto S1;
- S2: done <= 0 || A <= X || B <= Y || C <= Z;
- S3: A <= Add(A;B);
- S4: If B’[7] goto S3 || B <= Inc(B);
- S5: If C’[7] goto S3 || C <= Inc(C);
- S6: U <= A || done <= 1 || goto S1;
- End Alg

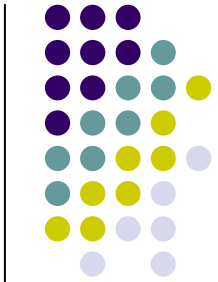
6.7 System Architecture



The controller

- Reads **B[7]** and **C[7]** to determine the destination of “goto”
- Sends control signals (**Ct**) to datapath to perform the appropriate operations

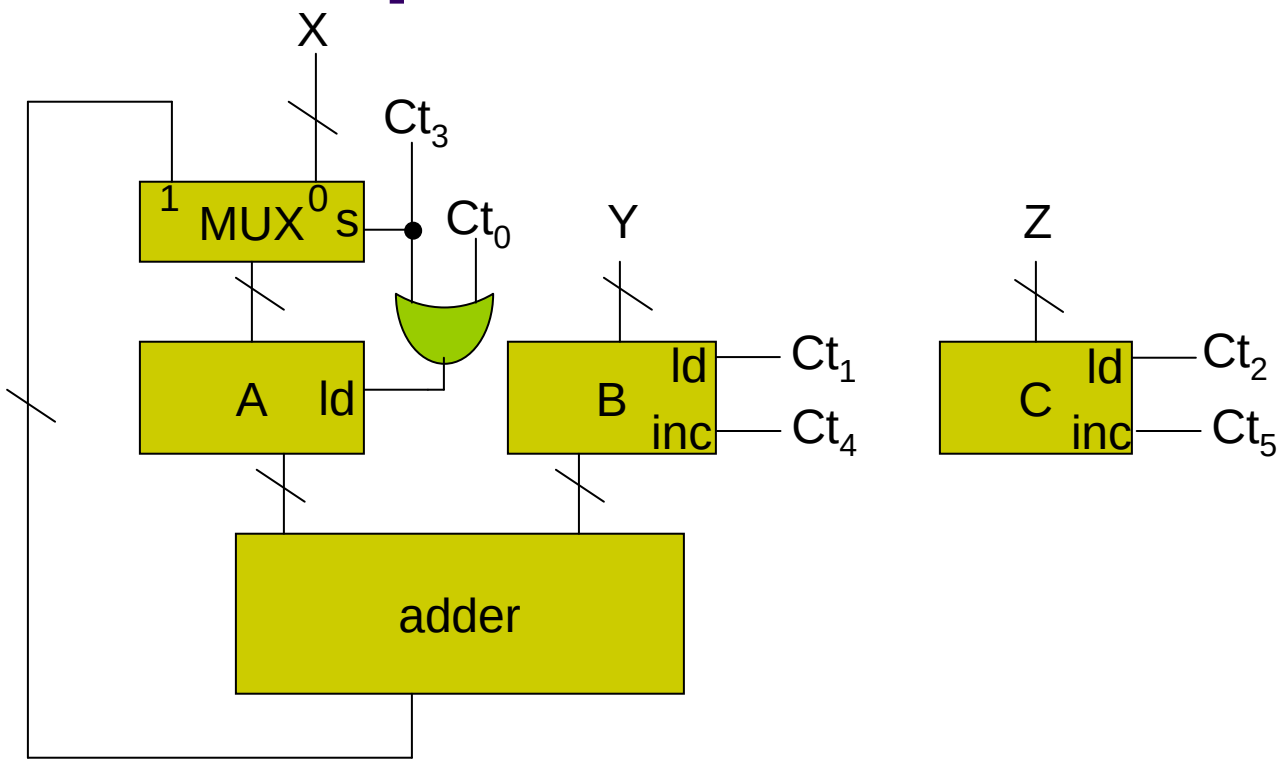
6.7 Operations and Its Control Signals



	operation	control
A <= X	A ← Load (X)	Ct_0
B <=Y	B ← Load (Y)	Ct_1
C <=Z	C ← Load (Z)	Ct_2
A<=A+B	A← Add(A,B)	Ct_3
B<=B+1	B ← INC(B)	Ct_4
C<=C+1	C ← INC(C)	Ct_5
U<=A	Wires	

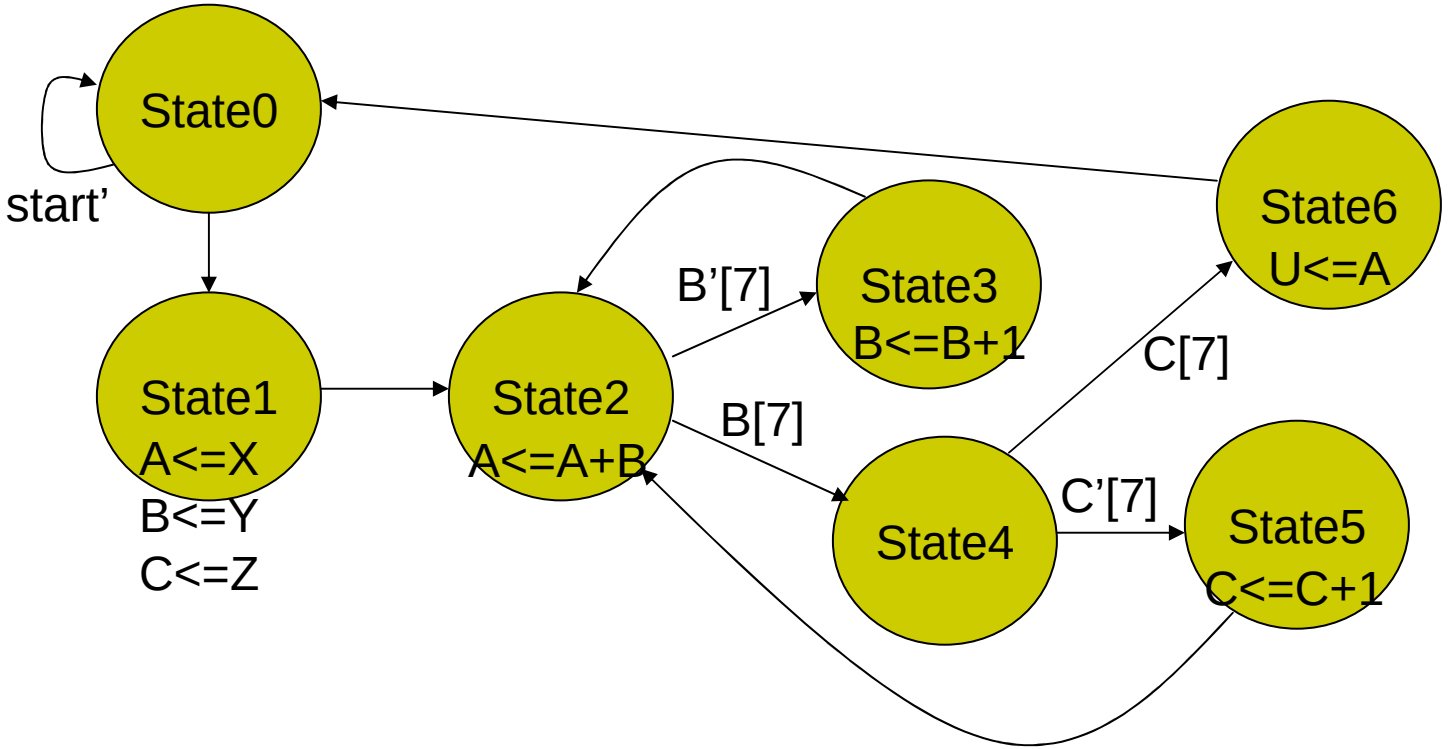
Hardware components needed:
One adder, two counters, a register

6.7 Datapath

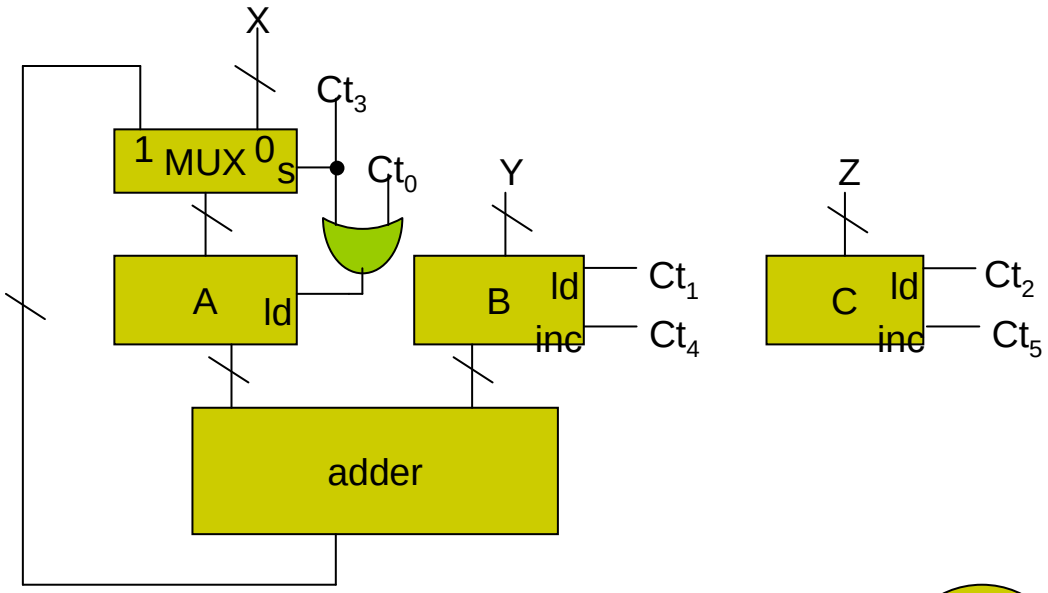


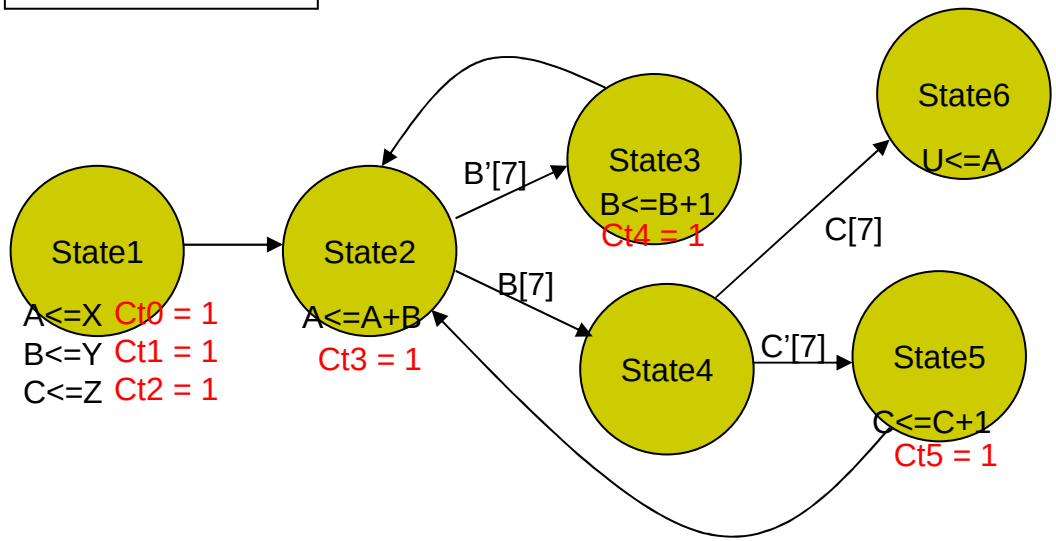
6.7 FSM

- S1: If start’ goto S1;
- S2: done <= 0 || A <= X || B <= Y || C <= Z;
- S3: A <= Add(A;B);
- S4: If B’[7] goto S3 || B <= Inc(B);
- S5: If C’[7] goto S3 || C <= Inc(C);
- S6: U <= A || done <= 1 || goto S1;



6.7 Controller Design





	Ct0	Ct1	Ct2	Ct3	Ct4	Ct5	done
state0	0	0	0	0	0	0	0
state1	1	1	1	0	0	0	0
state2	0	0	0	1	0	0	0
state3	0	0	0	0	1	0	0
state4	0	0	0	0	0	0	0
state5	0	0	0	0	0	1	0
state6	0	0	0	0	0	0	1