CODES, CONVERSION & ARITHMETIC

The construction of a Truth Table is very often the first step in designing hardware to implement a Boolean function:-

$$F = f(A,B, \ldots ,C)$$

- Don't try to design functions of more than 4 input and 1 output variables.
- Synthesize from elements of this manageable size.
- A good design, based on inspiration and experience, will enjoy 50-75% reduction in complexity in design over a bad, though functionally correct, one.
- Various optimization techniques, applicable to logic design, may, at best, result in a 5-20% improvement.
- The key to designing multi-bit output functions is the identification of the iterative block.

Arithmetic operations associated with analytic number codes tend to produce simpler iterative architecture for multi-bit arithmetic.

Integers can be coded as binary numbers in various ways. Some of these ways result in analytic codes. An analytic coded integer can be expressed as:-

$$I = \sum_i K_i b_i + K_o$$

where:-

I ..... Integer to be coded

K_i ..... The i'th weighting constant

b_i ..... The i'th bit

K_o ..... The initial, constant offset

Two examples of analytic, self complementing codes:-

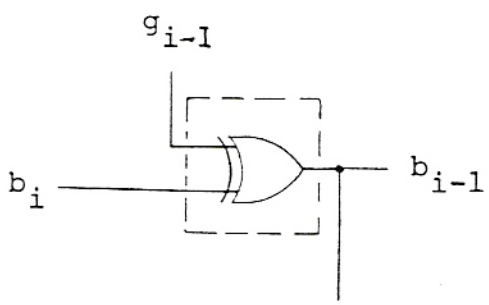
4-2-2*-1 BCD	I	b_4	b_3	b_2	b_1	b_4	b_3	b_2	b_1	Excess-3 BCD
	0	0	0	0	0	0	0	1	1	
$K_o = 0$	1	0	0	0	1	0	1	0	0	$K_o = -3$
	2	0	0	1	0	0	1	0	1	
$K_1 = 1$	3	0	1	0	1	0	1	1	0	$K_1 = 1$
	4	0	1	1	0	0	1	1	1	
$K_2 = 2$	5	1	0	0	1	1	0	0	0	$K_2 = 2$
	6	1	0	1	0	1	0	0	1	
$K_3 = 2$	7	1	1	0	1	1	0	1	0	$K_3 = 4$
	8	1	1	1	0	1	0	1	1	
K_4 4	9	1	1	1	1	1	1	0	0	$K_4 = 8$

Self complementing means that the 9's complement of any integer in the cycle can be formed by simply changing all 0's to 1's and all 1's to 0's.

GRAY CODE An example of a useful, nonanalytic code.

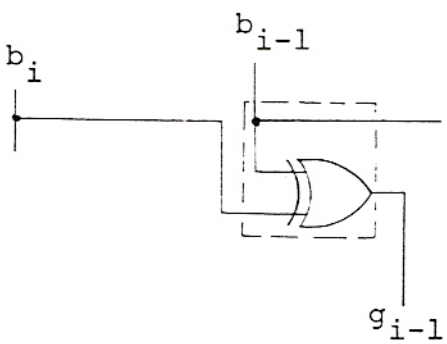
1. The iterative block to convert Gray to Straight Binary and to convert Straight Binary to Gray is very simple.
2 The Gray Code is such that adjacent integers differ by only one changed bit. This is particularly important in rotational and linear position encoding transducers as well as in ultrahigh speed ADC's (Analogue-to Digital-Convertors). In these instances a single quantum confusion can result in serious error if a binary, or any other noncyclic code is used.

I	g_4	g_3	g_2	g_1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	0	1	0
4	0	1	1	0
5	0	1	1	1
6	0	1	0	1
7	0	1	0	0
8	1	1	0	0
9	1	1	0	1
A	1	1	1	1
B	1	1	1	0
C	1	0	1	0
D	1	0	1	1
E	1	0	0	1
F	1	0	0	0



Iterative Block:-

Gray → Binary



Iterative Block:-

Binary → Gray

BINARY ARITHMETIC

In order to understand binary arithmetic well enough to design low level microprocessor interface hardware and software, we must practise both conversions and operations. By conversion we mean translation of numbers from the base 10 system with which we are familiar to the binary or base 2 system. By binary, we mean Straight Binary:-

00	0	0000
01	1	0001
02	2	0010
03	3	0011
04	4	0100
05	5	0101
06	6	0110
07	7	0111
10	8	1000
11	9	1001
12	A	1010
13	B	1011
14	C	1100
15	D	1101
16	E	1110
17	F	1111

... not BCD or Gray coded binary which was discussed previously to illustrate the principles of

a) coding,
b) an analytic code and
c) a self complementing code

By binary we also mean the two most commonly used "chunked" or multiple binary numbers, i.e., base-8 or OCTAL and base-16 or HEXADECIMAL numbers

2 octal cycles — 1 hexadecimal cycle — 8 binary cycles

By operations we mean binary:-

a) addtion,
b) subtraction,
c) multiplication,
d) division and
e) square root extraction

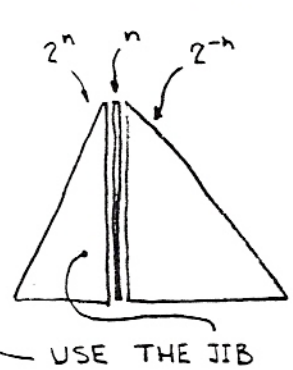
Central to the concept of designing logical iterative blocks to do arithmetic is the idea of code conversion. It was previously shown that in some instances, i.e. Gray ⇄ Binary, conversion is an easy logical operation. In other instances a combination of Table-Look-Up or TLU and arithmetic must be used to do conversions. This is the case, unfortunately, in entering numbers via a keyboard, having a computer perform calculations and then having the results of these printed. Consider the large part that code conversion plays in just doing binary arithmetic manually.

Addition

Add:- $562 + 921 = 1483$

(1) Convert 562:-

$562 - 512 = 50$ ... (9)
$50 - 32 = 18$ ... (5)
$18 - 16 = 2$ ... (4)
$2 - 2 = 0$ ... (1)



(2) Convert 921:-

$921 - 512 = 409$ ... (9)
$409 - 256 = 153$ ... (8)
$153 - 128 = 25$ ... (7)
$25 - 16 = 9$ ... (4)
$9 - 8 = 1$ ... (3)
$1 - 1 = 0$ ... (0)

The SAILBOAT is useful for conversion between binary and decimal

Table of Powers of 2

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

(3) Add 562+912:-

(10)	(9)	(8)	(7)	(6)	(5)	(4)	(3)	(2)	(1)	(0)
	1	0	0	0	1	1	0	0	1	0
+	1	1	1	0	0	1	1	0	0	1
1	0	1	1	1	0	0	1	0	1	1

(4) Convert result:-

(10)	...	1024
(8)	...	256
(7)	...	128
(6)	...	64
(3)	...	8
(1)	...	2
(0)	...	1
		1483

Subtraction

Subtract:- 921 − 562 = 359

(9)	(8)	(7)	(6)	(5)	(4)	(3)	(2)	(1)	(0)
1	1	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	0	1
									1
1	0	1	0	1	1	0	0	1	1

1's complement of 562, i.e. reverse all bits of 1000110010 and add 1 to form the 2's complement

x ignore carry

Convert result:-

(8)	...	256
(6)	...	64
(5)	...	32
(2)	...	4
(1)	...	2
(0)	...	1
		359

Multiplication

Multiply:- 21 × 62

	21
×	62
	42
126	
	1302

Convert:-

(4)	16		(5)	32	
(2)	4		(4)	16	
(0)	1		(3)	8	
	21		(2)	4	
			(1)	2	
				62	

(10)	(9)	(8)	(7)	(6)	(5)	(4)	(3)	(2)	(1)	(0)
						1	0	1	0	1
				x	1	1	1	1	1	0
						0	0	0	0	0
					1	0	1	0	1	
				1	0	1	0	1		
			1	0	1	0	1			
		1	0	1	0	1				
	1	0	1	0	1					
0	0	0	0	0						
1	0	1	0	0	0	1	0	1	1	0
(10)	(9)	(8)	(7)	(6)	(5)	(4)	(3)	(2)	(1)	(0)

Convert:-

(10)	1024	
(8)	256	
(4)	16	
(2)	4	
(1)	2	
	1302	

Division Divide:- 21 ⟌ 3781 = 180

```
    21 | 3781
       | 21
       | ---
       | 168
       | 168
       | ---
       |  01
```

Convert:- 3781 = 2048 + 1024 + 512 + 128 + 64 + 4 + 1

	3781
(11)	2048
	1733
(10)	1024
	709
(9)	512
	197
(7)	128
	69
(6)	64
	5
(2)	4
	1
(0)	1

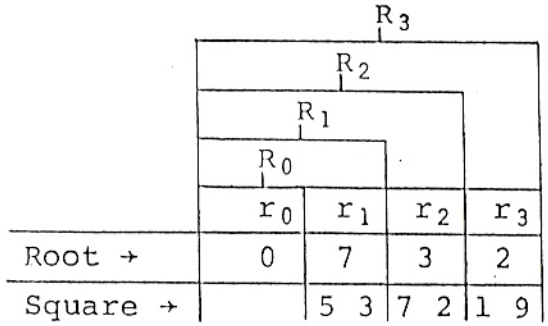
(7)	(6)	(5)	(4)	(3)	(2)	(1)	(0)
1	0	1	1	0	1	0	0

```
10101 | 1 1 1 0 1 1 0 0 0 1 0 1
        1 0 1 0 1
        ---------
          1 0 0 0 1
          0 0 0 0 0
          ---------
          1 0 0 0 1 0
            1 0 1 0 1
            ---------
              1 1 0 1 0
              1 0 1 0 1
              ---------
                1 0 1 0
            0 0 0 0 0
            ---------
              1 0 1 0 1
              1 0 1 0 1
              ---------
                      0
                    ---
                      1
                      0
                    ---
                      1
```

Convert:-

(7)	128
(5)	32
(4)	16
(2)	4
	180

Notice that integer division results in truncation or loss of any remainder.



s_0

$4\ 9$ $\quad q_1 = 2(bR_0r_1) + r_1^2 \leqq s_0 : r_1 \max(0,1,\ \ldots\ ,9)$

$4\ 7\ 2$ $\quad s_1 = s_0 - q_1$

$4\ 2\ 9$ $\quad q_2 = 2(bR_1r_2) + r_2^2 \leqq s_1 : r_1 \max(0,1,\ \ldots\ ,9)$

$4\ 3\ 1\ 9$ $\quad s_2 = s_1 - q_2$

$2\ 9\ 2\ 4$ $\quad q_3 = 2(bR_2r_3) + r_3^2 \leqq s_2 : r \max(0,1,\ \ldots\ ,9)$

s_0 ... Square whose root is to be taken

b ... Arithmetic number base = 10

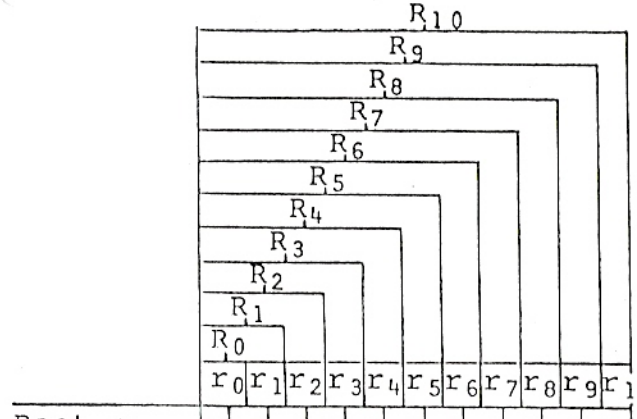
r_i ... $_i$th digit of the square root

R_i ... The square root, to i places

q_i ... Partial, trial square by which the residual square is reduced if $q_i \leqq s_{i-1}$

s_i ... Residual square at $_i$th step; $s_i = s_{i-1} - q_i$ if $q_i \leqq s_{i-1}$ otherwise $s_i = s_{i-1}$

Decimal Square Root Procedure



s_0

$q_1 = 4R_0 + 1 \leq s_0 : r_1 = 1$

$s_1 = s_0 - q_1$

$q_2 = 4R_1 + 1 > s_1 : r_2 = 0$

$s_2 = s_1$

$q_3 = 4R_2 + 1 \leq s_2 : r_3 = 1$

$s_3 = s_2 - q_3$

$q_4 = 4R_3 + 1 \leq s_3 : r_4 = 1$

$s_4 = s_3 - q_4$

$q_5 = 4R_4 + 1 > s_4 : r_5 = 0$

$s_5 = s_4$

$q_6 = 4R_5 + 1 \leq s_5 : r_6 = 1$

$s_6 = s_5 - q_6$

$q_7 = 4R_6 + 1 \leq s_6 : r_7 = 1$

$s_7 = s_6 - q_7$

$q_8 = 4R_7 + 1 \leq s_7 : r_8 = 1$

$s_8 = s_7 - q_8$

$q_9 = 4R_8 + 1 > s_8 : r_9 = 0$

$s_9 = s_8$

$q_{10} = 4R_9 + 1 > s_9 : r_{10} = 0$

Binary Square Root Procedure

General Purpose 1-bit

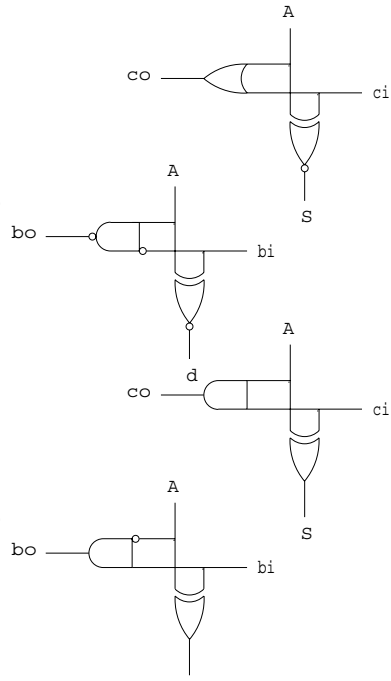
1. Incrementer
2. Decrementer
3. 1/2-Adder
4. 1/2-Subtracter

A	ci	S	co
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	1

A	bi	d	bo
0	0	1	1
0	1	0	1
1	0	0	0
1	1	1	1

A	ci	S	co
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

A	bi	d	bo
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



$p=0=\bar{t}$
$q=1$
$r=1=t$

t=1

$p=0=\bar{t}$
$q=0$
$r=1=t$

$p=1=\bar{t}$
$q=0$
$r=0=t$

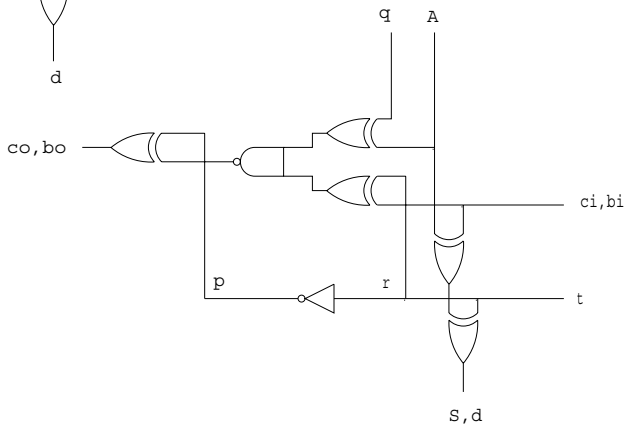
t=0

$p=1=\bar{t}$
$q=1$
$r=0=t$

Instruction

q	t
1	1
0	1
0	0
1	0

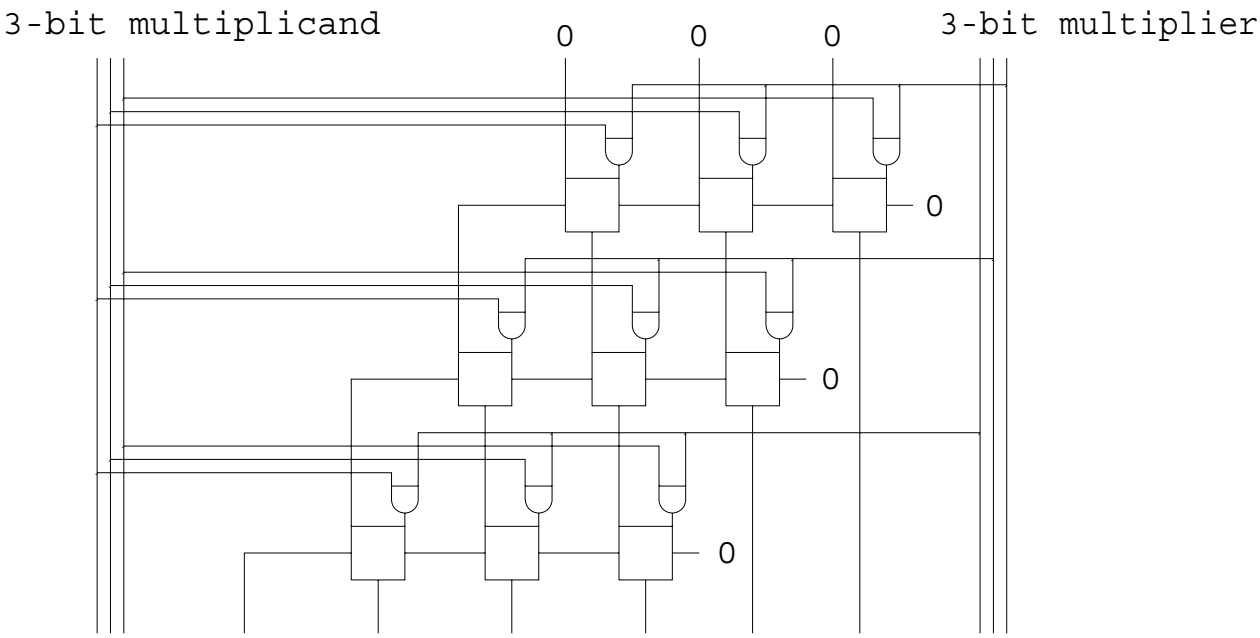
Selected Function



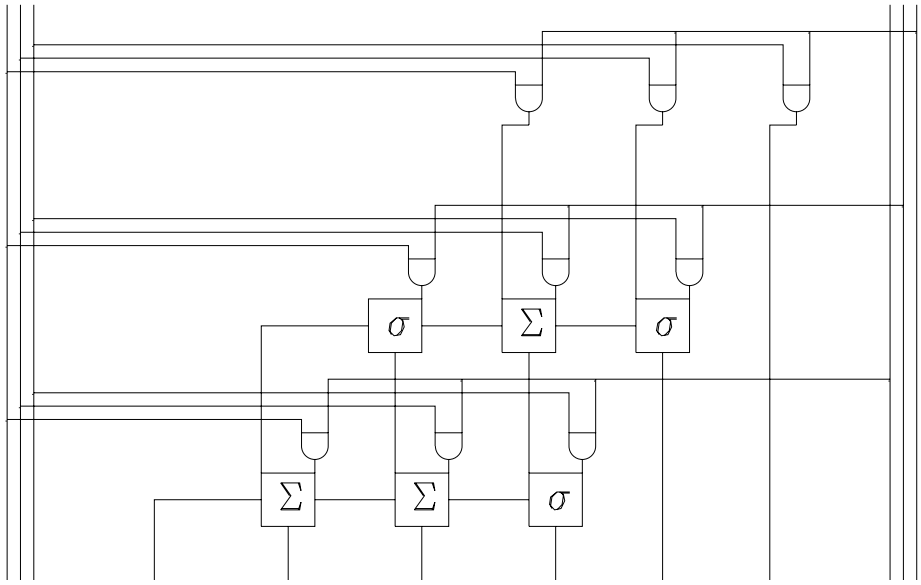
GP1BAU

3x3=6 bit PARALLEL MULTIPLIER

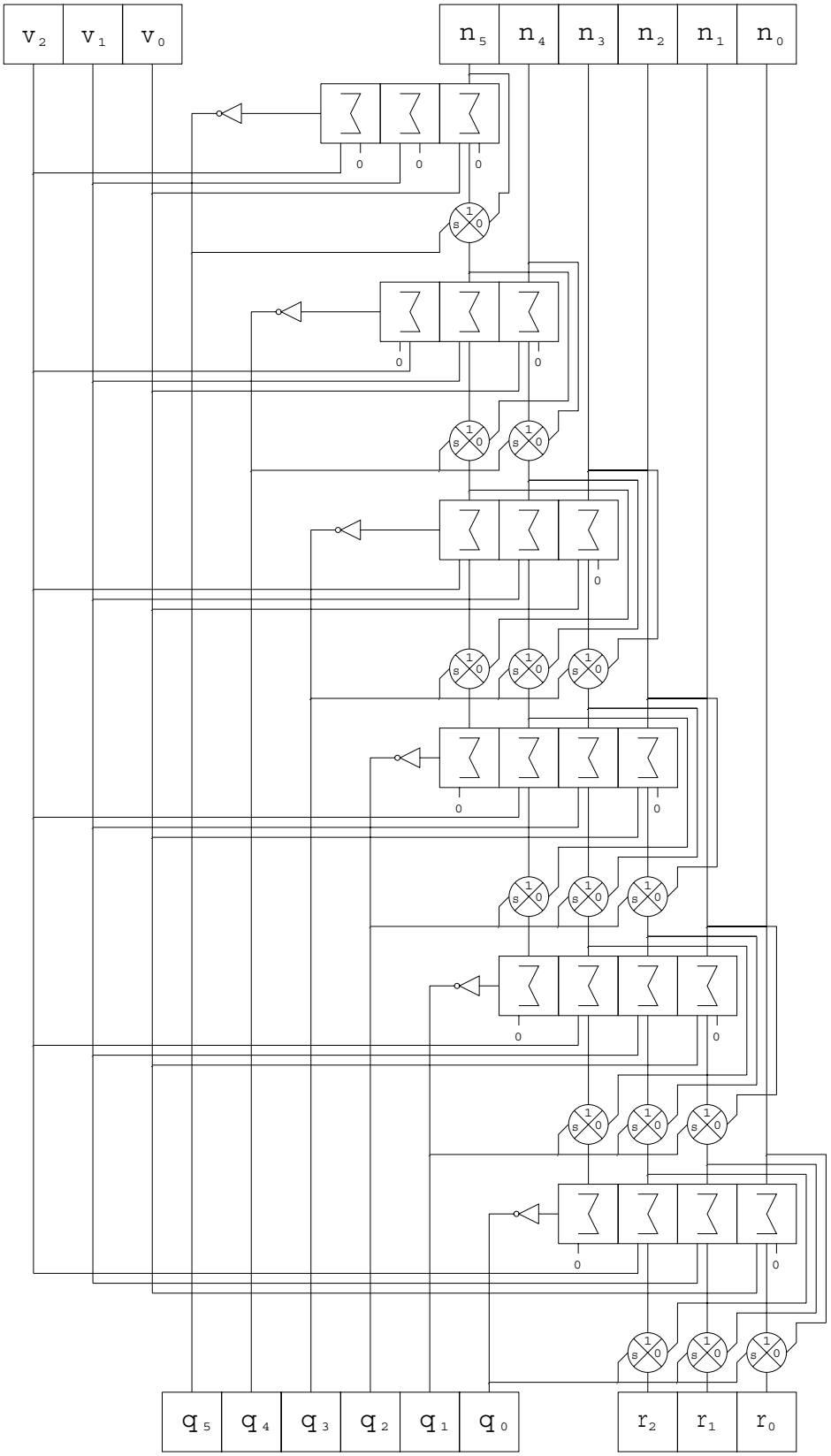
Regular Architecture



Minimal Architecture

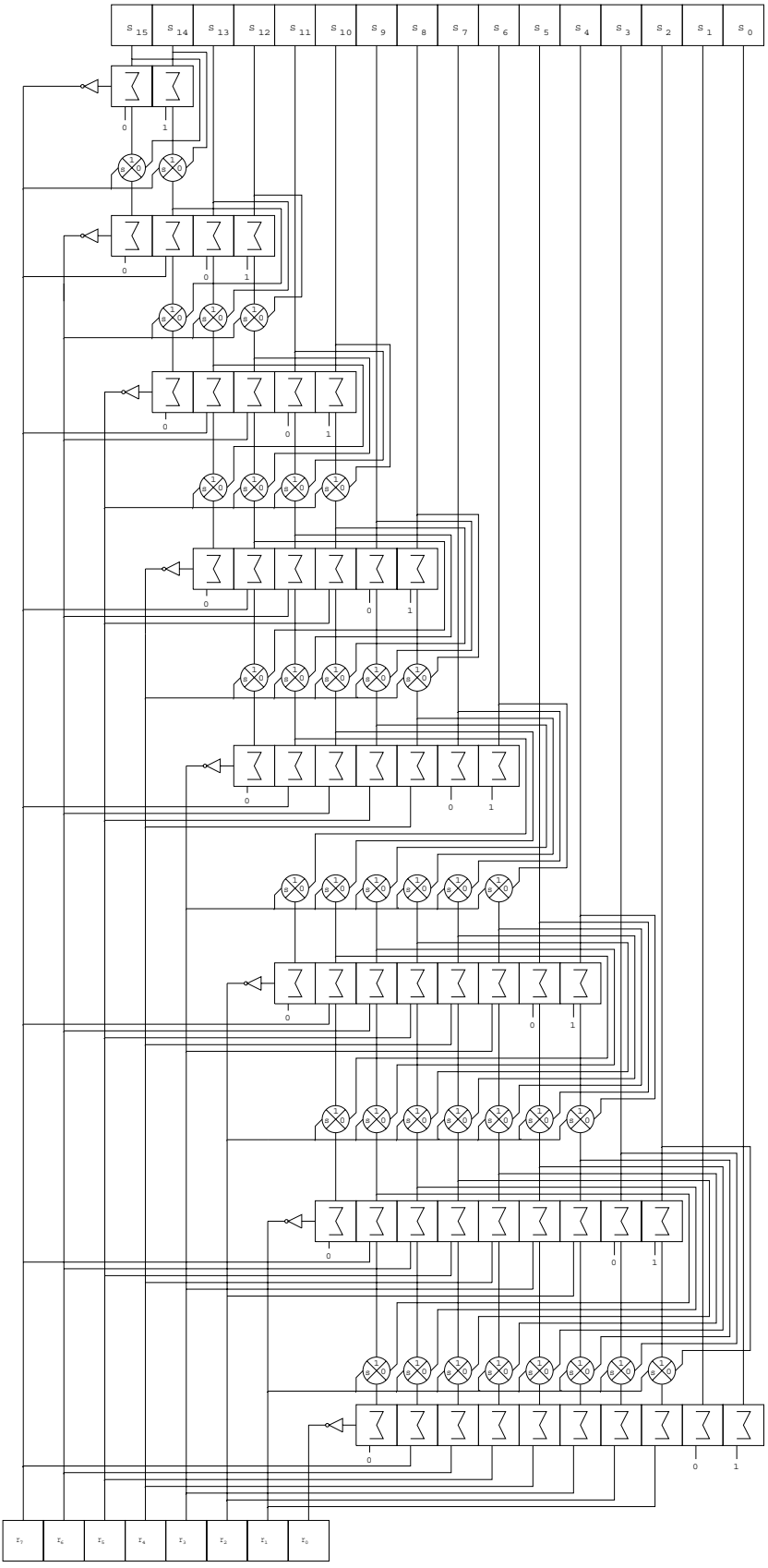


6-bit product



6-bit dividend, 3-bit divisor, 6-bit quotient divider

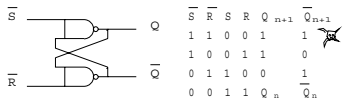
DD6DR3Q6



16-bit to 8-bit parallel square-rooter

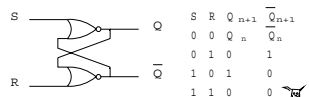
Elements of Sequential Circuits

NAND Flip-Flop



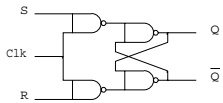
$\overline{S}$	$\overline{R}$	S	R	Q_{n+1}	$\overline{Q}_{n+1}$
1	1	0	0	1	1
1	0	0	1	1	0
0	1	1	0	0	1
0	0	1	1	Q_n	$\overline{Q}_n$

NOR Flip-Flop

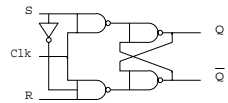


S	R	Q_{n+1}	$\overline{Q}_{n+1}$
0	0	Q_n	$\overline{Q}_n$
0	1	0	1
1	0	1	0
1	1	0	0

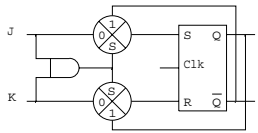
Gated Flip-Flop



SOC (Set overides Clear)

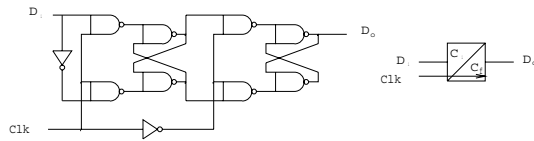


J-K Flip-Flop (Race problem)

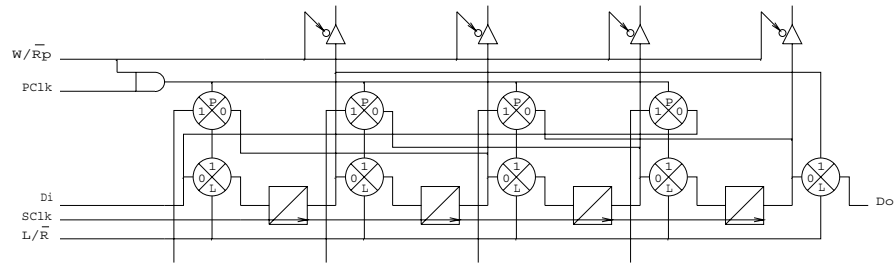


J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q}_n$

Master/Slave Data Flip-Flop

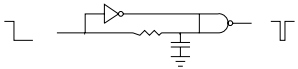


Parallel Outputs

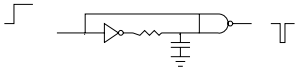


Parallel Inputs

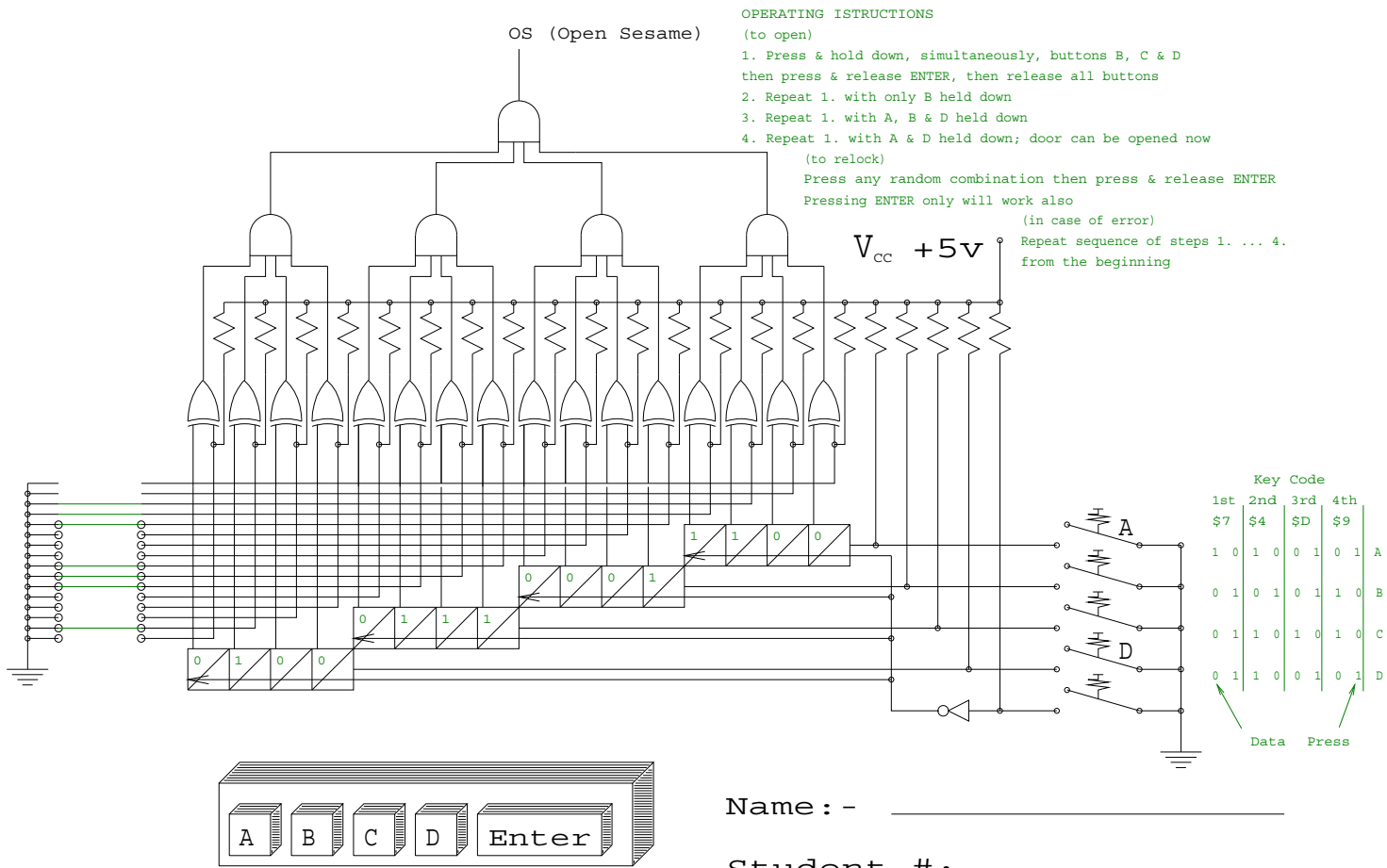
Low going falling edge trigger



Can it be done with NOR-gates?



Low going rising edge trigger



Name:- _____

Student #:- _____

DOORLOCK