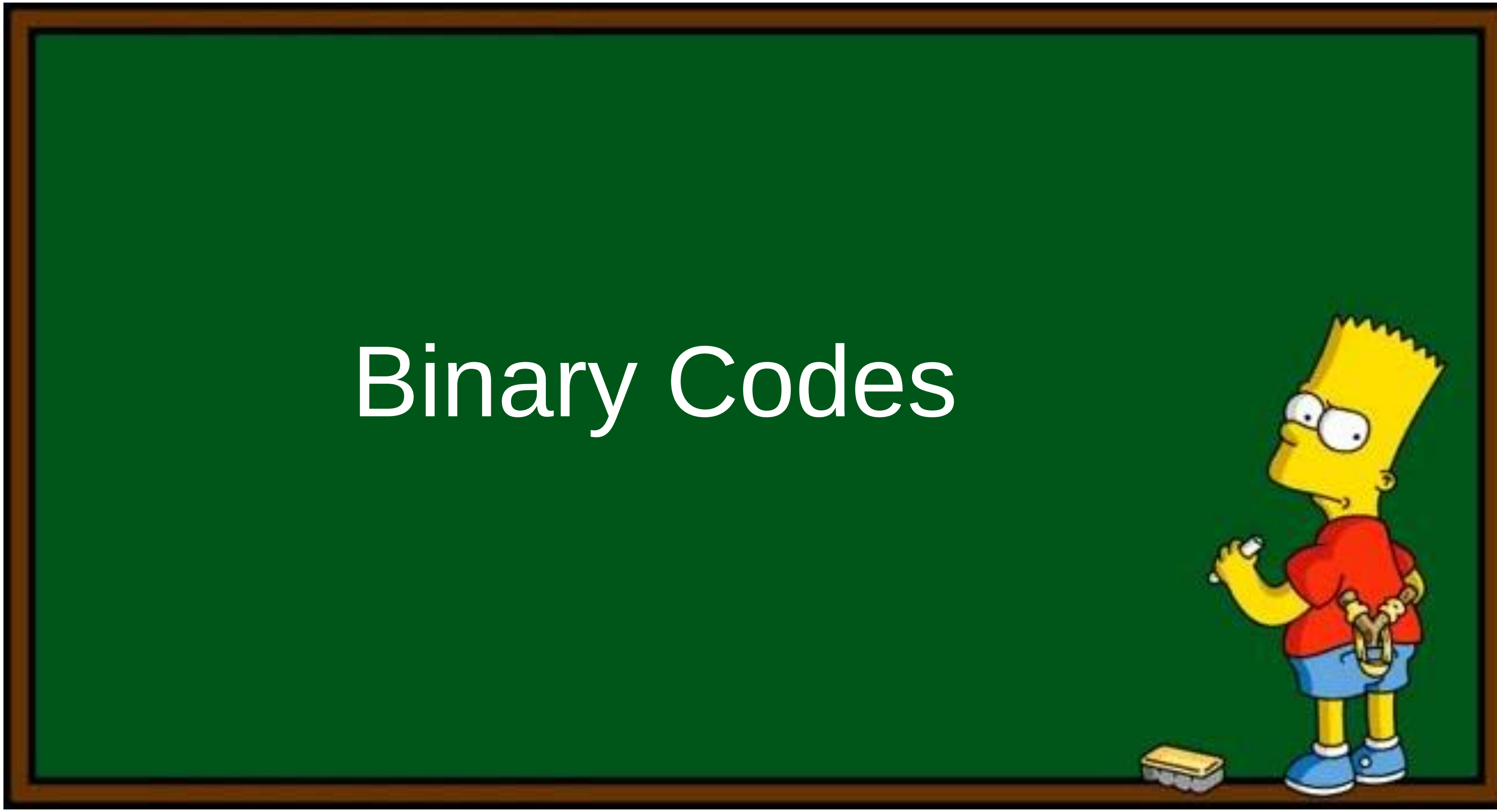
EGEC 281: Designing with VHDL

Fall 2024

Lecture 1: Binary Codes

Rakesh Mahto, Ph.D.
Office: *E 314, California State University, Fullerton*
Office Hours: *Monday and Wednesday 2:00 - 3:30 pm*
Or by appointment
Office Hour Zoom Meeting ID: *894 4126 5483*
Email: *ramahto@fullerton.edu*
Phone No: *657-278-7274*

Binary Codes



Binary Codes

- Digital systems represent and manipulate many other discrete elements of information (not only binary numbers)
- If information is continuous (analog signal) we must discretize it by sampling it at specified intervals (Nyquist Sampling Theorem) - ADC
- If information is discrete (digital signal) we must convert it to an analog signal if peripheral device requires it (DAC)
- Bit = binary digit

Powers of 2^n

Base	Power	Result	Denomination
2	1	2	
	2	4	
	3	8	
	4	16	
	5	32	
	6	64	
	7	128	
	8	256	
	9	512	
	10	1024	**KILO**
	11	2048	
	12	4096	
	13	8192	
	14	16384	
	15	32768	
	16	65536	
	17	131072	
	18	262144	
	19	524288	
	20	1048576	**MEGA**
	30	1073741824	**GIGA**
		1073741824	
		1.09951E+12	
	40	1.09951E+12	**TERA**

Logarithms

- In general: $\log_B A = x$ is equivalent to $A = B^x$
  - **$\log_2 8 = 3$ since $8 = 2^3$**
  - **$\log_5 1/5 = -1$ since $1/5 = 5^{-1}$**
  - **$\log_{81} 3 = 1/4$ since $3 = 81^{1/4}$**
- Properties:
  - **$\log_B (ab) = \log_B a + \log_B b$**
  - **$\log_B (a/b) = \log_B a - \log_B b$**
  - **$\log_B (a^p) = p \log_B a$**
  - **$\log_B (a^{1/r}) = 1/r \log_B a$**
  - **$\log_A (B) \log_B (C) = \log_A (C)$**
    - **$\ln B \log_B C = \ln C$**
    - **$\log_A B \log_B A = 1$**

Binary Codes

- How many distinct elements can we represent with n bits?
  - **$N = 2^n$**
  - **n-bit code is found by counting in binary from 0 to $(2^n - 1)$**
- We can let a set of bits represent distinct elements by a CODE (language)
- EX: wish to find a binary code to represent the ten decimal numbers (0,1,2,3,4,5,6,7,8,9)
  - **Have 10 distinct elements**
  - **Need $\log_2 N = n$ (number of binary digits) Ceiling Function**
  - **Need four bits minimum: $2^4 = 16 > 10$ elements; $n = 4$**
    - **10 symbols with meaning or CODE are used**
    - **6 symbols are extra; not used**

Binary Codes

- EX: wish to find a binary code to represent 1009 different objects
  - **Have 1009 distinct elements**
  - **Need $\log_2 N = n$ (number of binary digits) Ceiling Function**
  - **Need ten bits minimum: $2^{10} = 1024 > 1009$ elements; n = 10**
    - **1009 symbols with meaning or CODE are used**
    - **15 symbols are extra; not used**

Binary Codes

- EX: wish to find a binary code to represent 20,234,567 different objects. Only have $\log_{10}$ or ln functions
  - **Make C = 20,234,567; B =2; A =10**
  - **Using: $\ln(B)\log_B(C) = \ln(C)$**
    - **$\log_2(20{,}234{,}456) = \ln(20{,}234{,}567)/\ln(2) = 16.823/0.693$ = round up the integer 25**
  - **Using: $\log_A(B)\log_B(C) = \log_A(C)$**
    - **$\log_2(20{,}234{,}456) = \log_{10}(20{,}234{,}567)/\log_{10}(2) =$ 7.306/0.30 = integer 25**
  - **Need 25 bits minimum: 2^{25} = 33,554,432 > 20,234,567 elements; n = 25**
    - **20,234,567 symbols with meaning or CODE are used**
    - **13,319,865 symbols are extra; not used**

More on Binary Codes

Binary Codes

Although most large computers work internally with binary numbers, the input-output equipment generally uses decimal numbers. Because most logic circuits only accept two-valued signals, the decimal numbers must be coded in terms of binary signals. In the simplest form of binary code, each decimal digit is replaced by its binary equivalent. For example, 937.25 is represented by:

$$\begin{array}{cccccc} 9 & 3 & 7 & . & 2 & 5 \\ 1001 & 0011 & 0111 & . & 0010 & 0101 \end{array}$$

Decimal Codes

- Binary codes for decimal digits require a minimum of 4 binary digits or bits
  - **Have 10 distinct elements (0 – 9)**
  - **Need $\log_2 N = n$ (number of binary digits) Ceiling Function**
  - **Need four bits minimum: $2^4 = 16 > 10$ elements; n = 4**
    - **0000 0**
    - **0001 1**
    - **0010 2**
    - **0011 3**
    - **0100 4**
    - **0101 5**
    - **0110 6**
    - **0111 7**
    - **1000 8**
    - **1001 9**

BCD Code: Binary Coded Decimal

- **NOTE: 10 of 16 possible distinct elements are used; 6 are not used.**
- **One-to-one mapping to avoid ambiguity.**
- **This is not a conversion it is a CODE.**

Binary Codes for Decimal Digits

Decimal Digit	8-4-2-1 Code (BCD)	6-3-1-1 Code	Excess-3 Code	2-out-of-5 Code	Gray Code
0	0000	0000	0011	00011	0000
1	0001	0001	0100	00101	0001
2	0010	0011	0101	00110	0011
3	0011	0100	0110	01001	0010
4	0100	0101	0111	01010	0110
5	0101	0111	1000	01100	1110
6	0110	1000	1001	10001	1010
7	0111	1001	1010	10010	1011
8	1000	1011	1011	10100	1001
9	1001	1100	1100	11000	1000

Decimal numbers represented in various binary codes

Decimal	Binary code	8 4 2 1 or BCD code	Reflective Gray code	2-out-of-5 coded decimal code
0	0000	0000	0000	00011
1	0001	0001	0001	00101
2	0010	0010	0011	00110
3	0011	0011	0010	01001
4	0100	0100	0110	01010
5	0101	0101	0111	01100
6	0110	0110	0101	10001
7	0111	0111	0100	10010
8	1000	1000	1100	10100
9	1001	1001	1101	11000
10	1010	0001 0000	1111	00101 00011
25	11001	0010 0101	10101	00110 01100
37	100101	0011 0111	110111	01001 10010
98	1100010	1001 1000	1010011	11000 10100

Decimal to Reflective Gray Code conversion using the mirror method

Decimal	1-bit reflective Gray code
0	0
1	1

Decimal	2-bit reflective Gray code
0	00
1	01
2	11
3	10

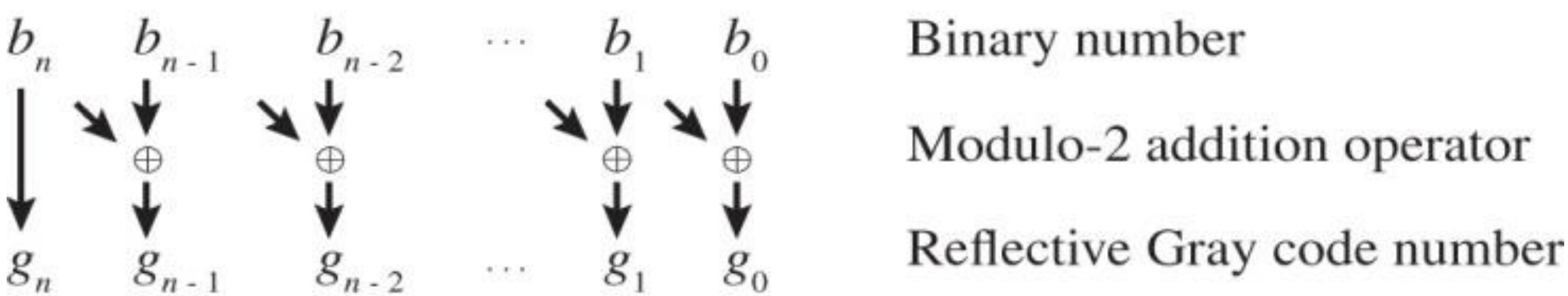
Mirror

Decimal	3-bit reflective Gray code
0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

Decimal	4-bit reflective Gray code
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

. . .

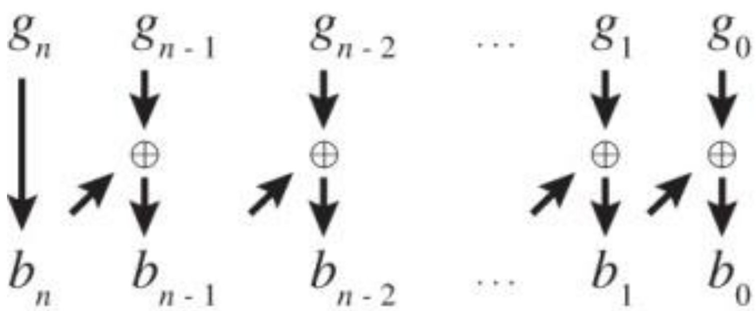
Binary to RGC conversion method



Modulo-2 addition table

$$\begin{array}{r} 0 \\ \oplus 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ \oplus 1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ \oplus 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ \oplus 1 \\ \hline 0 \end{array} \quad \begin{array}{l} \left.\begin{array}{l} \\ \\ \end{array}\right\} \text{Two bits to be added by modulo-2 addition} \\ \leftarrow \text{Sum bits, ignore carry} \end{array}$$

RGC to binary conversion method



Reflective Gray code number

Modulo-2 addition operator

Binary number

Alphanumeric Codes

- Permit representation of information other than just numbers
  - **We can represent**
    - **Alphabetic characters**
    - **Punctuation marks**
    - **Special symbols #, $, *, @, …**
    - **Numbers**
  - **Most common codes**

    ASCII: American Standard for Information Interchange
    - **7 bits + parity bit = 8 bit code**
    - **2^7 = 128 different symbols (elements)**

    EBCDIC: Extended BCD Interchange Code
    - **8 bits + parity bit = 9 bit code**
    - **2^8 = 256 different symbols (elements)**

    UNICODE: Universal Code
    - **16 bits**
    - **2^{16} = 65,536 different symbols (elements)**
    - **Universal characters; encode all world languages**

7-bit ASCII character et code (nonextended form)

(a)

	$b_6 b_5 b_4$							
$b_3 b_2 b_1 b_0$	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	NULL	DLE	Space	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYNC	&	6	F	V	f	v
0111	BELL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

(b)

Control	**Characters:**	**Control**	**Characters:**	**Graphic**	**Characters:**
NULL	Null	DLE	Data link escape	’	Apostrophe
SOH	Start of heading	DC1	Device control 1	-	Hyphen
STX	Start of text	DC2	Device control 2	/	Forward slant
ETX	End of text	DC3	Device control 3	<	Less than
EOT	End of transmission	DC4	Device control 4 (stop)	>	Greater than
ENQ	Enquiry	NAK	Negative knowledge	[	Opening bracket
ACK	Acknowledge	SYNC	Synchronous idle	\	Reverse slant
BELL	Bell (audible Signal)	ETB	End of transmission block	]	Closing bracket
BS	Backspace	CAN	Cancel	^	Circumflex
HT	Horizontal tabulation	EM	End of medium	_	Underline
LF	Line feed	SUB	Substitute	`	Grave accent
VT	Vertical tabulation	ESC	Escape	{	Opening brace
FF	Form feed	FS	File separator	|	Vertical line
CR	Carriage return	GS	Group separator	}	Closing brace
SO	Shift out	RS	Record separator	~	Overline (tilde)
SI	Shift in	US	Unit separator		
		DEL	Delete		

Character	ASCII Code A_6	A_5	A_4	A_3	A_2	A_1	A_0	Character	ASCII Code A_6	A_5	A_4	A_3	A_2	A_1	A_0
space	0	1	0	0	0	0	0	@	1	0	0	0	0	0	0
!	0	1	0	0	0	0	1	A	1	0	0	0	0	0	1
"	0	1	0	0	0	1	0	B	1	0	0	0	0	1	0
#	0	1	0	0	0	1	1	C	1	0	0	0	0	1	1
$	0	1	0	0	1	0	0	D	1	0	0	0	1	0	0
%	0	1	0	0	1	0	1	E	1	0	0	0	1	0	1
&	0	1	0	0	1	1	0	F	1	0	0	0	1	1	0
'	0	1	0	0	1	1	1	G	1	0	0	0	1	1	1
(	0	1	0	1	0	0	0	H	1	0	0	1	0	0	0
)	0	1	0	1	0	0	1	I	1	0	0	1	0	0	1
*	0	1	0	1	0	1	0	J	1	0	0	1	0	1	0
+	0	1	0	1	0	1	1	K	1	0	0	1	0	1	1
,	0	1	0	1	1	0	0	L	1	0	0	1	1	0	0
–	0	1	0	1	1	0	1	M	1	0	0	1	1	0	1
.	0	1	0	1	1	1	0	N	1	0	0	1	1	1	0
/	0	1	0	1	1	1	1	O	1	0	0	1	1	1	1
0	0	1	1	0	0	0	0	P	1	0	1	0	0	0	0
1	0	1	1	0	0	0	1	Q	1	0	1	0	0	0	1
2	0	1	1	0	0	1	0	R	1	0	1	0	0	1	0
3	0	1	1	0	0	1	1	S	1	0	1	0	0	1	1
4	0	1	1	0	1	0	0	T	1	0	1	0	1	0	0

Table 1-3
ASCII code
(incomplete)

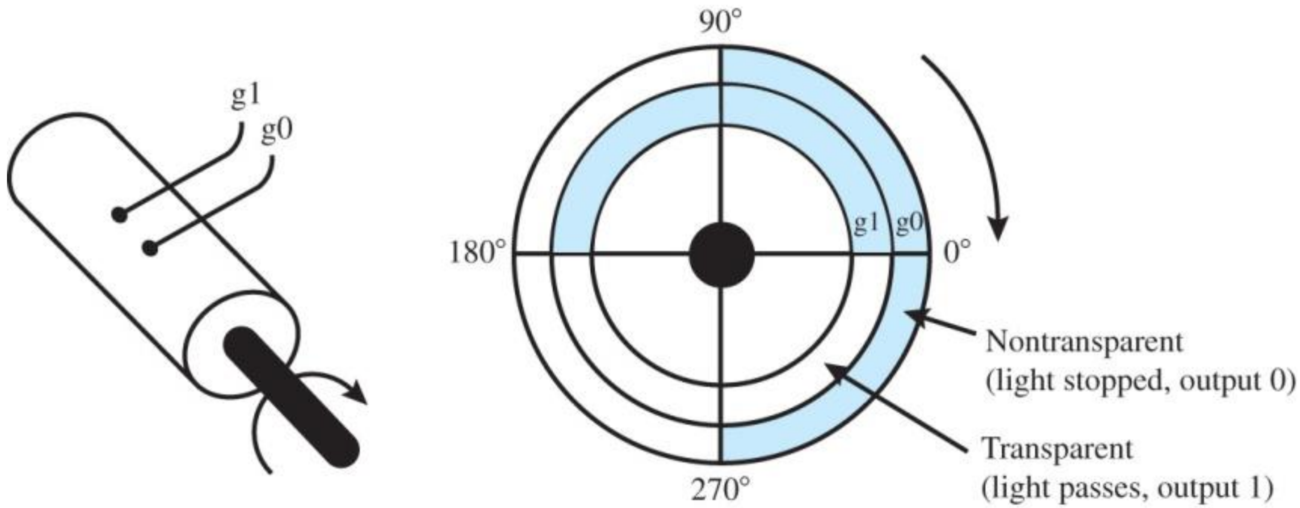
Error Detection Codes

- Most computers manipulate an 8-bit quantity as a single unit: 8-bits = byte
  - **Or multiple bytes, examples**
    - **16-bits = 2 bytes = 1 word**
    - **32-bits = 4 bytes = 1 longword**
- Some codes make use of a parity bit

  Parity bit = extra bit included with a message to make total number of 1's ODD or EVEN
  - **EX: ASCII letter A is 1000001 (7-bit code)**
  - **With ODD parity = 1 1000001 odd number of 1's in message "A"**
  - **With EVEN parity = 0 1000001 even number of 1's in message "A"**
  - **An error is detected if check parity (receiving end) is not equal to parity adopted. Detects only an ODD combination of errors. EVEN combination of errors is undetectable.**

7 Segment Code

7-segment Code

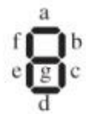


(a) (b)

Shaft position in degrees	Reflective Gray code g1 g0
0–90	0 0
90–180	0 1
180–270	1 1
270–360	1 0

(c)

7-segment code for a 7-segment display



7-segment code b_6 b_5 b_4 b_3 b_2 b_1 b_0	7-segment display				
0 1 1 1 1 1 1	0				
0 0 0 0 1 1 0	1				
1 0 1 1 0 1 1	2				
1 0 0 1 1 1 1	3				
1 1 0 0 1 1 0	4				
1 1 0 1 1 0 1	5				
1 1 1 1 1 0 0	6	or	6	With a flag	b_6 b_5 b_4 b_3 b_2 b_1 b_0 (1 1 1 1 1 0 1)
0 0 0 0 1 1 1	7				
1 1 1 1 1 1 1	8				
1 1 0 0 1 1 1	9	or	9	With a flag	b_6 b_5 b_4 b_3 b_2 b_1 b_0 (1 1 0 1 1 1 1)

Letter Display System

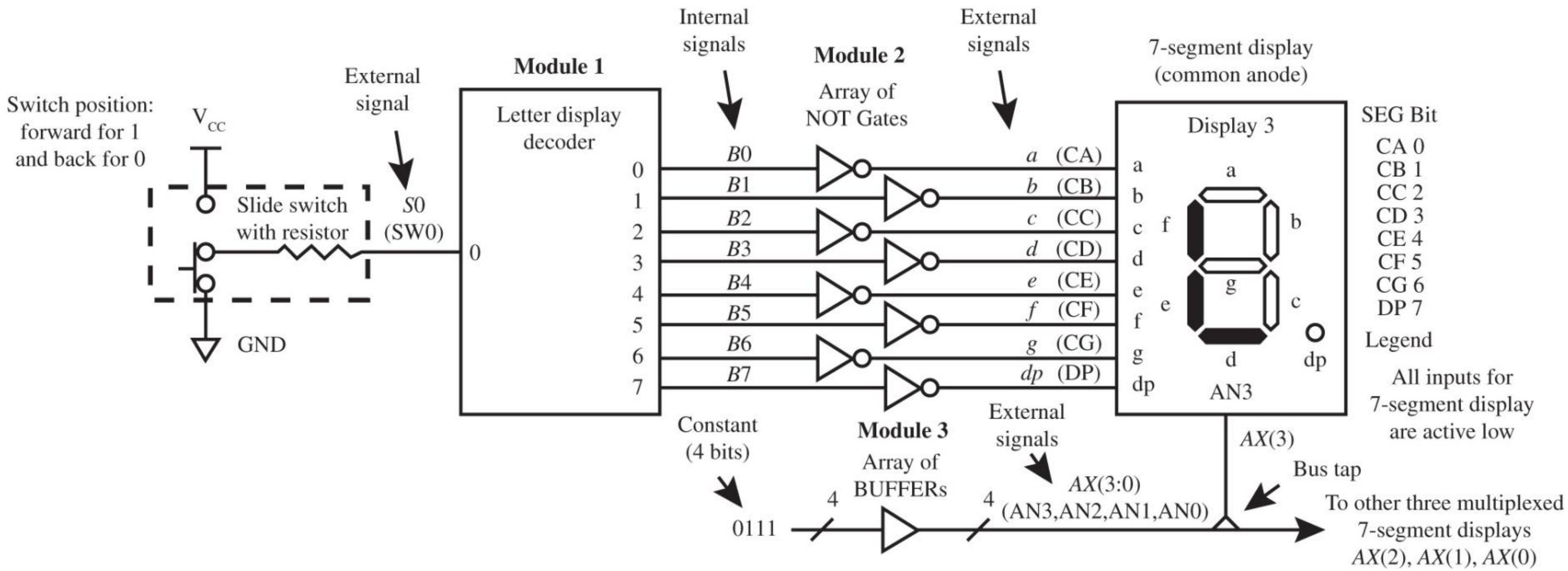
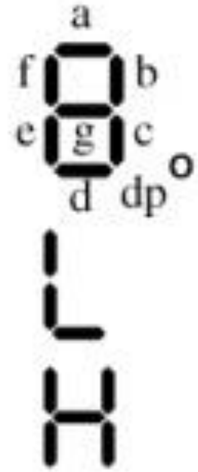


TABLE 2.5

Letter display decoder (active high outputs)

$S0$	$B7$	$B6$	$B5$	$B4$	$B3$	$B2$	$B1$	$B0$
0	0	0	1	1	1	0	0	0
1	0	1	1	1	0	1	1	0

7-segment display



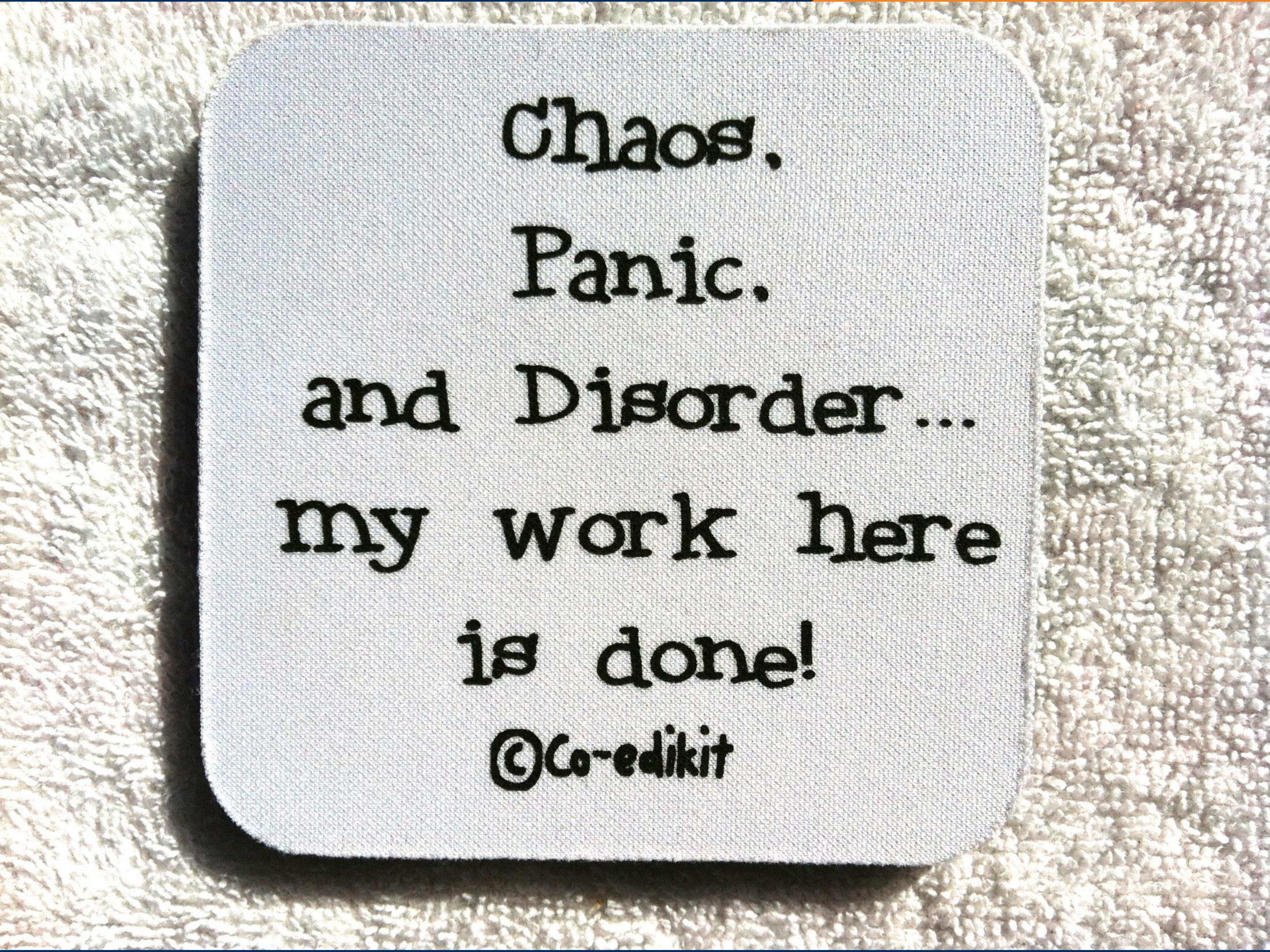
VHDL code for letter display system


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity LDS is port (
     s0 : in std_logic;
     a,b,c,d,e,f,g,dp : out std_logic;
     ax :out std_logic_vector (3 downto 0)
     );
end LDS;

architecture Boolean_functions of LDS is
--Internal signals
     signal b0,b1,b2,b3,b4,b5,b6,b7: std_logic;
begin

--Letter Display Decoder
     b0 <= '0'; b1 <= s0; b2 <= s0; b3 <= not s0;
     b4 <= '1'; b5 <= '1'; b6 <= s0; b7 <= '0';
--Array of NOT gates
     a <= not b0; b <= not b1; c <= not b2; d <= not b3;
     e <= not b4; f <= not b5; g <= not b6; dp <= not b7;
--Array of BUFFERs to enable 7-segment display
     ax <= "0111";
end Boolean _ functions;
```


Chaos.
Panic.
and Disorder...
my work here
is done!
©Co-edikit

Q&A