Embedded Systems

Week 6: Combinational Circuits
Embedded Linux

Dr. Vecdi Emre Levent

Instructors

Assist. Prof. Dr. Vecdi Emre Levent

Email : emre@levent.tc

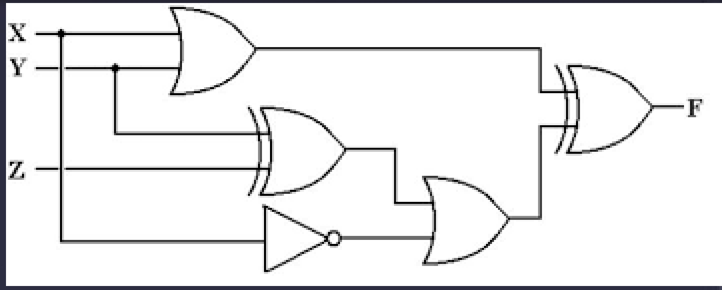
emre.levent@marmara.edu.tr

Web: www.levent.tc

Why Digital Systems?

- Computer Hardware
  - Softwares that require performance can be only written by who have a deep understanding of hardware.

- Almost all electronic devices are digital
  - Audio recorders , cameras , vehicles ph1s , medical devices…
  - Developing equipment needed in almost every industry
- It is an area that is highly needed both in our country and abroad. It could be a different career goal for you.



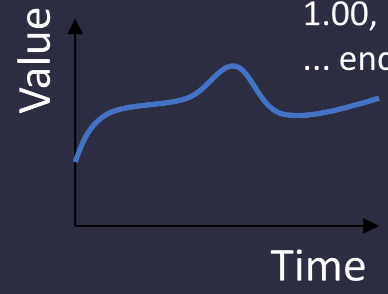
What Does Digital Mean?

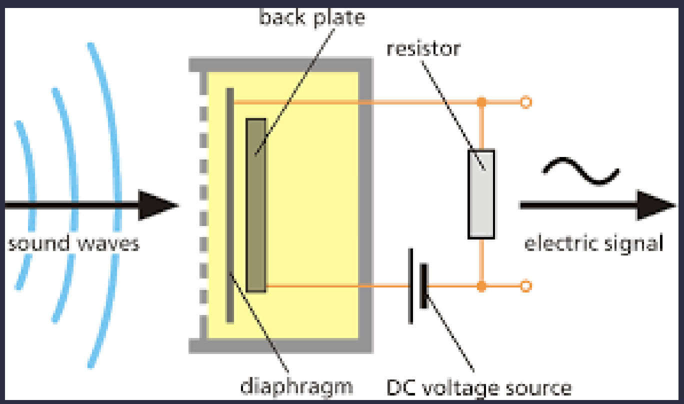
- Analog Signal
  - It has infinite possible value.
    - For example, the vibration created by a microph1 on the line.

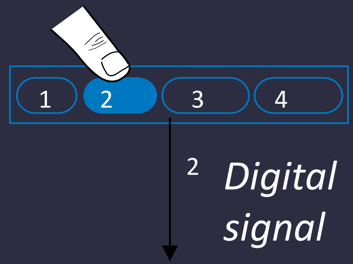
- Digital Signal
  – Finite possible values
    - For example : Pressing a button on a keypad

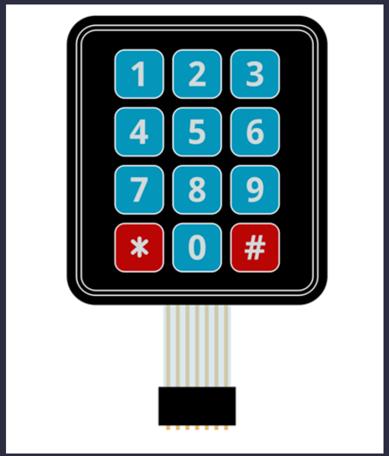
Analog Signal

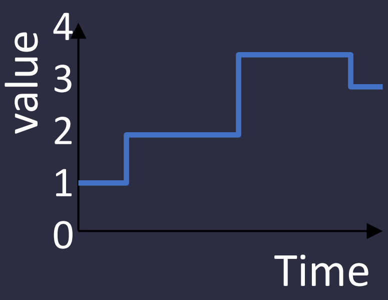
Likely Values
1.00, 1.01, 20000009,
... endless possibilities









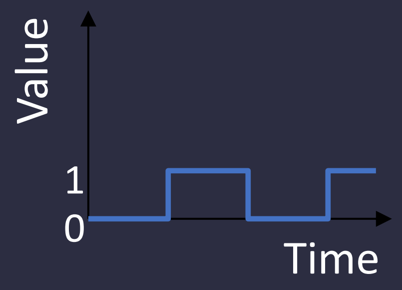


Possible values :
0, 1, 2, 3, or 4.

There are no other possible values

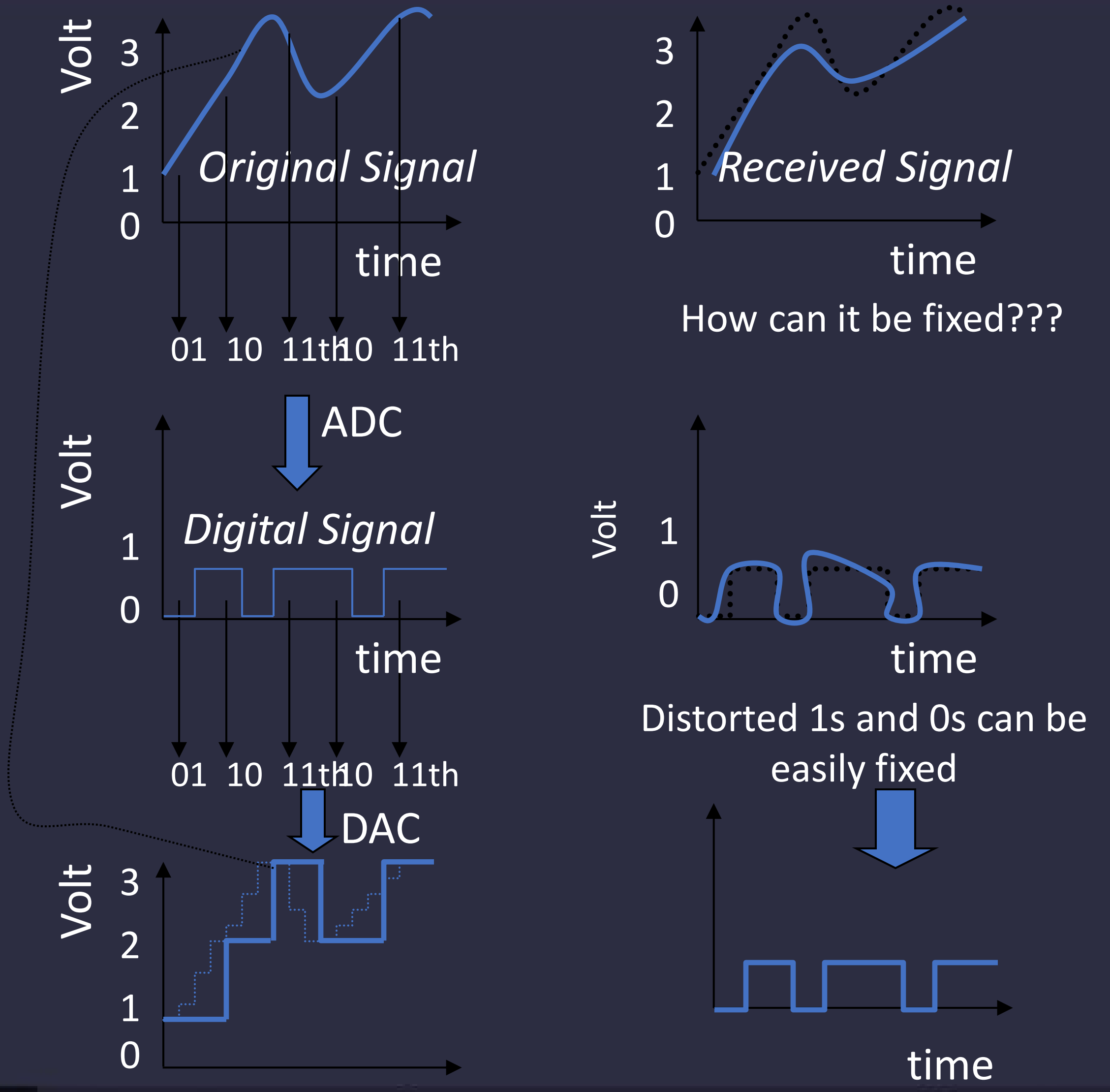
Digital Signals with Only Two Values: Binary

- ***Binary*** digital signals have only two possible values
  - These are shown as 0 and 1
  - A binary digit is expressed as a "bit".
  - Within the scope of the course, binary digital systems will be considered.
  - Binary is popular because:
    - Transistors , the most basic digital electrical comp1nt , operate at two voltage values (0 and 1)



Advantages of Digitization

- Analogue signal is very sensitive to noise
  - During transmission, voltage levels may change due to many factors.
- Digital signals are more resistant to degradation during transmission.
  - Voltage levels still may not transmit perfectly
  - However, some distorted 1s and 0s can be recovered.



Digitized Content, Compression Benefits

- Digitized Audios can be compressed
  - eg . MP3

- Compression can also be d1 on photos (jpeg) or videos (mpeg)

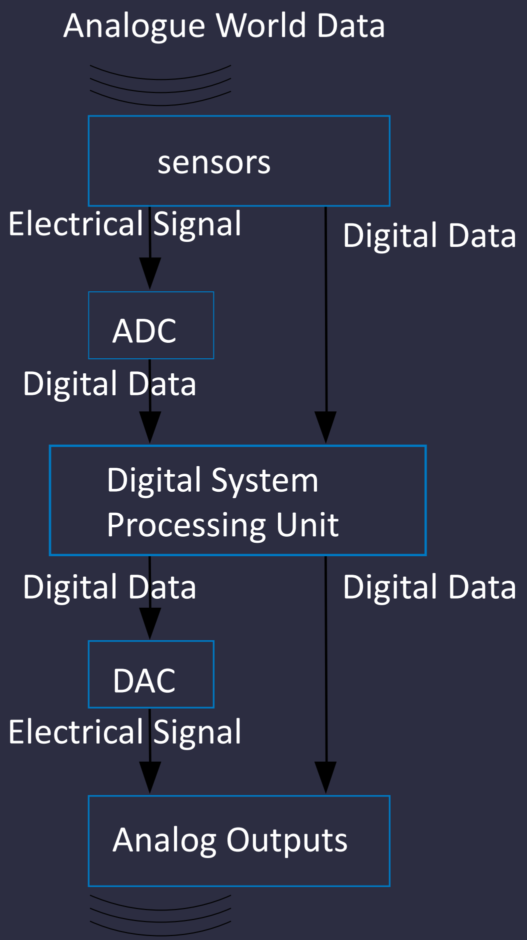
- Digitization has many different advantages.

Example Compression Table
00 --> 0000000000
01 --> 1111111111
1X --> X

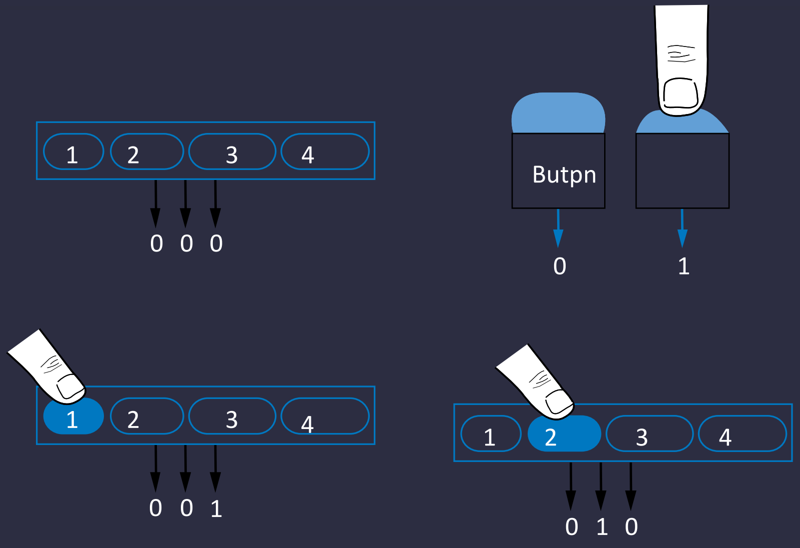
0000000000 0000000000 0000001111 1111111111

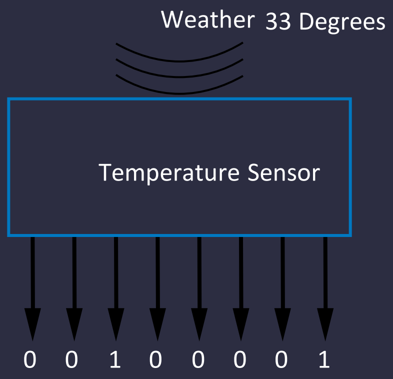
00 00 10000001111 01

Binary Data Encode



- If button is not pressed (0), if pressed (1)
- Multi-button : coding
  1st button =001, 2nd button =010, ...
- Some inputs are analog
  - Requires an analog-to-digital converter to switch to digital.
- ADC (Analog to Digital Converter): Converts analog signal to digital
- DAC (Digital to Analog Converter): Converts digital signal to analog





ASCII Encoding

- ASCII: 8 bits of each character and symbol. It is a table with the corresponding

Symbol	Mostding
R	1010010
S	1010011
T	1010100
L	1001100
N	1001110
TO	1000101
0	0110000
.	0101110
<tab>	0001001

Symbol	Mostding
r	1110010
s	1110011
t	1110100
l	1101100
n	1101110
to	1100101
9	0111001
!	0100001
<space>	0100000

1010010 1000101 1010011 1010100

REST

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Numbers Encoding

- Decimal base (***decimal***)
  - There are 10 symbols : 0, 1, 2, ..., 8, and 9
  - After 9 comes a new digit
    - So each digit is a power of 10.
    - Base of 10 is used as it is suitable for daily life operations.
- Binary Base (***binary***)
  - There are two symbols : 0 and 1
  - New power comes after 1
    - So each digit is a power of 2.

2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
512	256	128	64	32	16	8	4	2	1

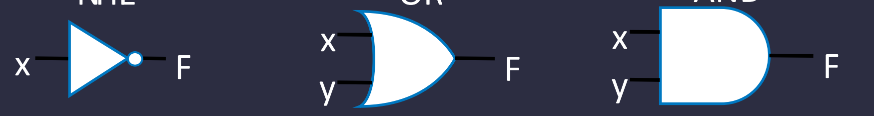
5	2	3
10^2	10^1	10^0

		1	0	1
2^4	2^3	2^2	2^1	2^0

Boolean Algebra

- Logic Gates are built with

Symbol



Truth Table

NHE

x	F
0	1
1	0

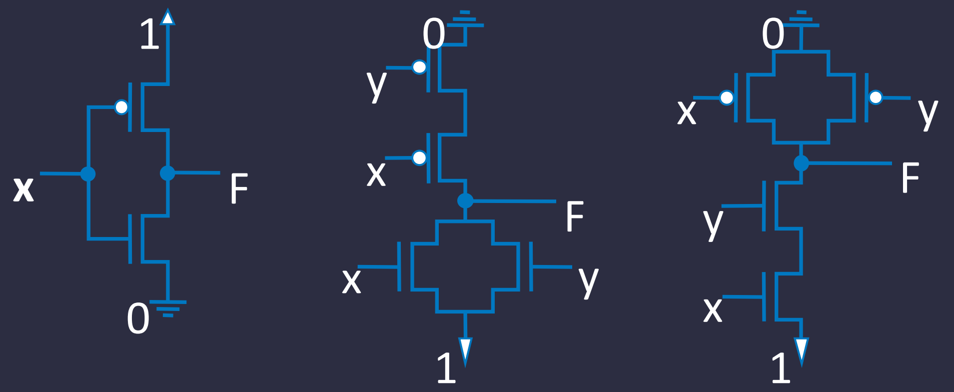
OR

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

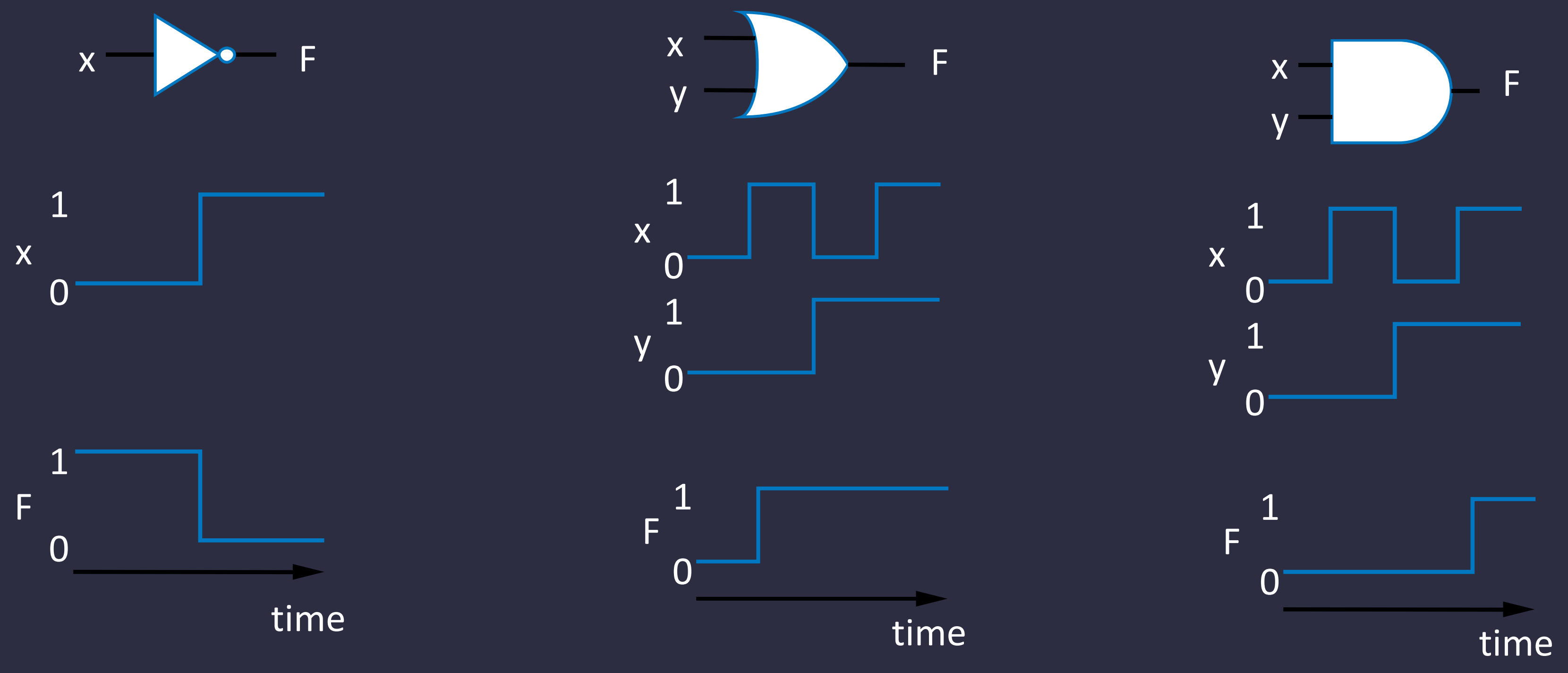
AND

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

Transistor circuit



NOT/OR/AND Logic Gates Time Diagram



Boolean Algebra Example

- a=1, b=1, c=1, d=0

F = (a **AND b) OR (c AND d)**

Answer : F = (1 AND 1) OR (1 AND 0)
= 1 OR 0 = 1.

a	b	AND
0	0	0
0	1	0
1	0	0
1	1	1

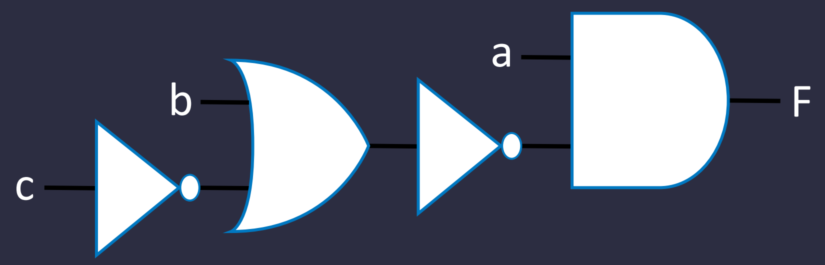
a	b	OR
0	0	0
0	1	1
1	0	1
1	1	1

a	NOT
0	1
1	0

Boolean Algebra Example

- boolean equation

  given below F = a AND NOT(b OR NOT(c))



Truth Tables

- F indicates output.

  - 2- Input : 4 lines
  - 3- Input : 8 lines
  - 4- Input : 16 lines

a	b	F
0	0	
0	1	
1	0	
1	1	

a	b	c	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

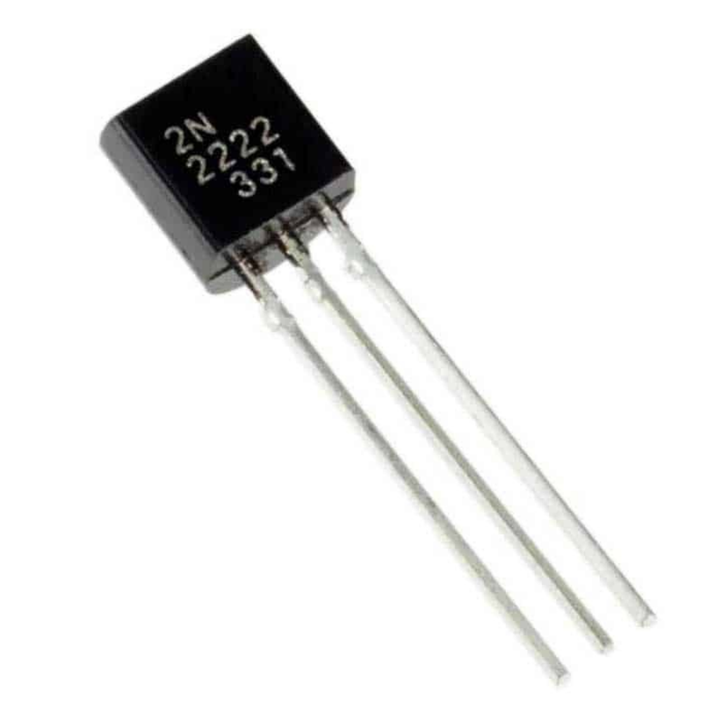
a	b	c	D	F
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	

How is Data Stored on a Computer?

- The computer is an electronic circuit.
  - It basically works by controlling the flow of electrons.

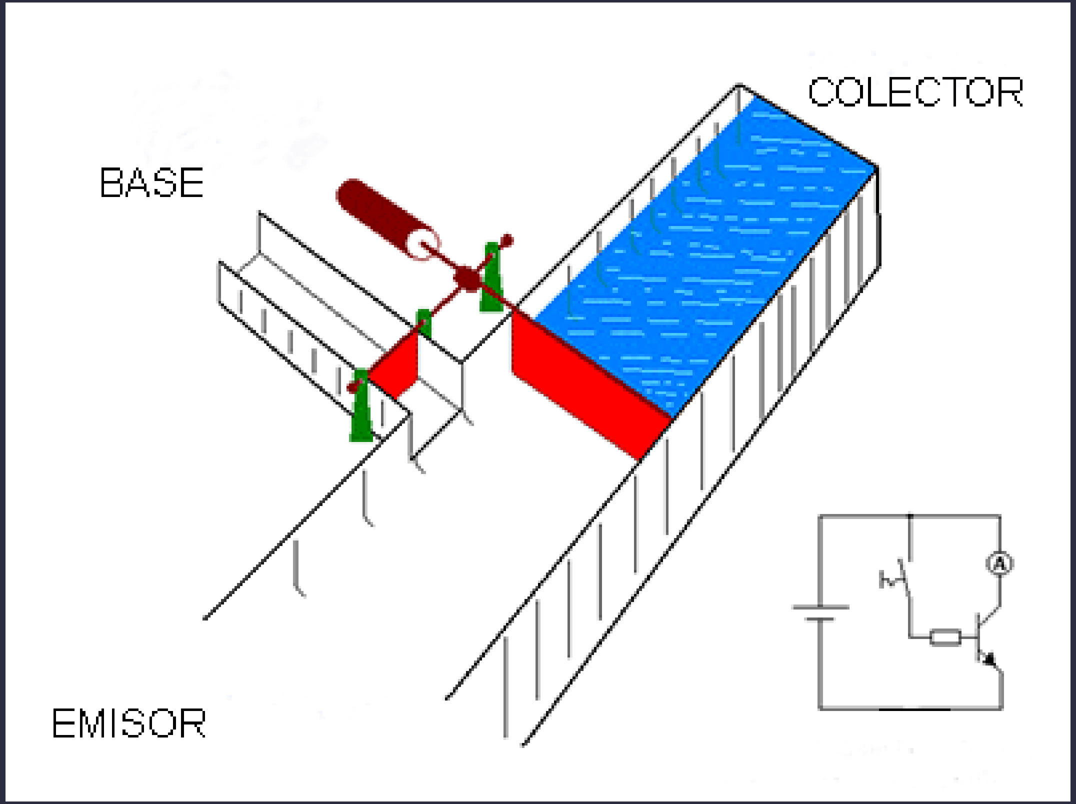
How is Data Stored on a Computer?

- Electrons are controlled by " Transistors " .
  - It basically works by controlling the flow of electrons.



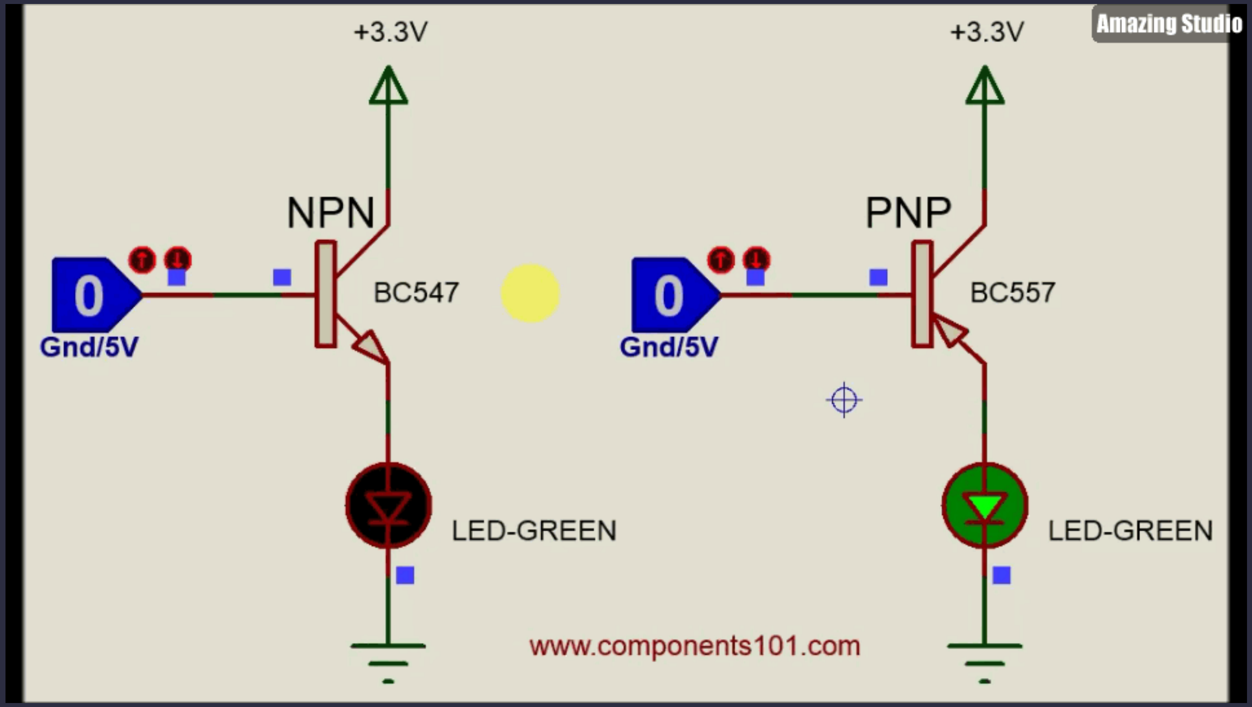
How is Data Stored on a Computer?

- Electrons are controlled by " Transistors " .
  - It basically works by controlling the flow of electrons.



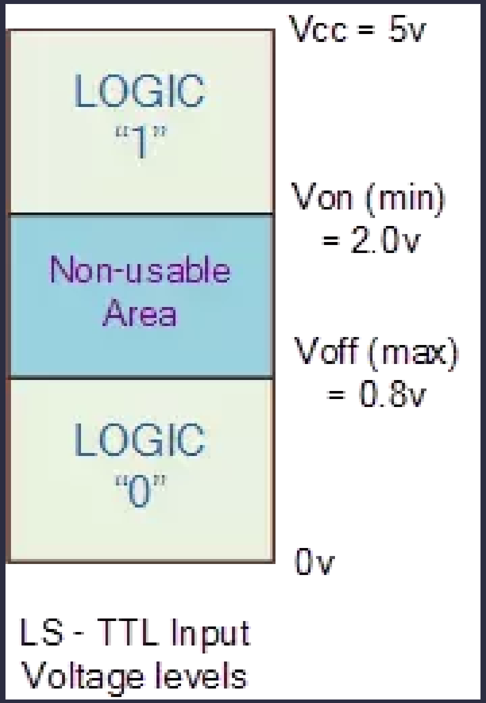
How is Data Stored on a Computer?

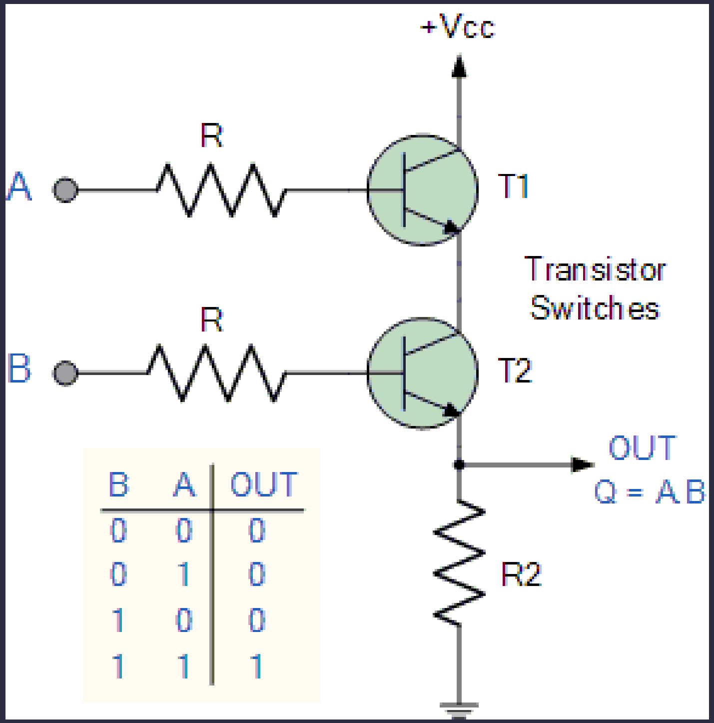
- Electrons are controlled by " Transistors " .
  - It basically works by controlling the flow of electrons.



How is Data Stored on a Computer?

- Data has two states :
  1. the voltage (Voltage) exists – This state is called "1".
  2. The state where the voltage disappears - This state is called "0".





How is Data Stored on a Computer?

- It is also possible to make a computer that works according to more than two voltage states.
  - But the control circuit of this computer will be much more complex.
  - For this reason, today's modern computers work with the concept of bit, which is the smallest unit, while expressing information.
  - It is the smallest data storage unit that can hold 0 or 1 on a bit.

The computers works with binary system.

Binary system :

- It has two states : 0 and 1

- Larger storage areas are obtained by combining multiple bits.
  - two bits , 4 different numbers can be expressed.

The computers works with binary system.

- Two bits , 4 different numbers can be expressed.

- 00 = 0 (in decimal)
- 01 = 1
- 10 = 2
- 11 = 3

The computers works with binary system.

- 3 bits 8 numbers can be expressed by combining them :

- 000 = 0
- 001 = 1
- 010 = 2
- 011 = 3
- 100 = 4
- 101 = 5
- 110 = 6
- 111 = 7

The computers works with binary system.

- In summary;

- 2^n with n bits different numbers can be expressed.

- ^{2}= 4 different numbers for 2 bits
- 2^3 = 8 different numbers for 3 bits
- 2^4 = 16 different numbers for 4 bits

…

can be expressed.

What types of data are expressed / stored in the computer?

- Numbers – signed (unsigned) , integers , decimal numbers (floating _ _ _ _ point), complex numbers (complex) , rational , irrational , ...
- Texts – Characters (characters) , texts (string) , ...
- Images – pixels , images , ...
- Sound
- Logic (logic) – true (true) , false (false)
- Operations (Instructions)
- ...

- Let's start with the numbers ...

Unsigned (Unsigned) Integers (Integers)

- Unsigned integers
  - They always store positive values

  - Ex :

329 (in base 10)

10^2 10^1 10^0

3x100 + 2x10 + 9x1 = 329

101 (in base 2)

2^2 2^1 2^0

1x4 + 0x2 + 1x1 = 5

Unsigned Integers

- An n - bit unsigned integer 2^n has a value : from 0 to 2^n-1 .

2^2	2^1	2^0	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Unsigned Binary Base Arithmetic

- Binary base addition (like base 10)
  - It is collected starting from the rightmost, and if it is available, it is transferred to the next total.

```
  10010     10010   1111
+  1001     +1011   +  1
  -----     -----   ----
  11011     11101  10000
```

Signed Integers (Integrs)

- With n bits , we can store 2^n different values .
  - 2^n different value;
  - Signed integers are obtained by assigning half to positive numbers and half to negative numbers.
  - Positive numbers 1 to 2^{n-1}
    Negative numbers $-(2^{n-1})$ to -1
  - For example , if we have a 3-bit storage;
  - Positive numbers are from 1 to 4 and negative numbers are from -4 to -1.

Signed Integers (Integrs)

- For example , if we have a 3-bit storage;
- Positive numbers are from 1 to 4 and negative numbers are from -4 to -1.

- If the number 0 is also used, a number from either positive or negative part is expressed as 0.

Signed Integers (Integrs)

- Positive integers
  - They are like unsigned integers.
    00101 = 5

- Negative integers
  - Sign Bit Representation – Always sign bit is first bit,
    Other bits are written as in unsigned representation.
    10101 = -5

  - 1 's complement – Each bit is inverted
    . 11010 = -5

  - In both representations, the largest bit represents the sign of the number :
    0= positive , 1= negative

Two’s complement

- Sign bit notation and 1 's complement problems
  - The number 0 has two representations (+0 and –0)

  - Sign Bit
    0 0 0 0 0 = +0
    1 0 0 0 0 = - 0

  - 1 's complement

00000 = +0
11111 = -0

Two’s complement

- Sign bit notation and 1 's complement problems
  - The necessary hardware circuits of arithmetic operations are very complex.

    - Problem with sign bit denoted addition

```
  1 0 1 1  ( -3 )
+ 0 0 1 0  ( 2 )
  1 1 0 1  ( -5 ) → Wrong
```

    - For the solution, before adding, it is necessary to check which one is larger, subtract the smaller from the larger, and place the sign of the larger number.

    - Therefore, it is necessary to have a circuit, subtractor and sign bit setter in the necessary hardware to perform the necessary addition process. That is, the hardware becomes complex and large.

Two’s Complement

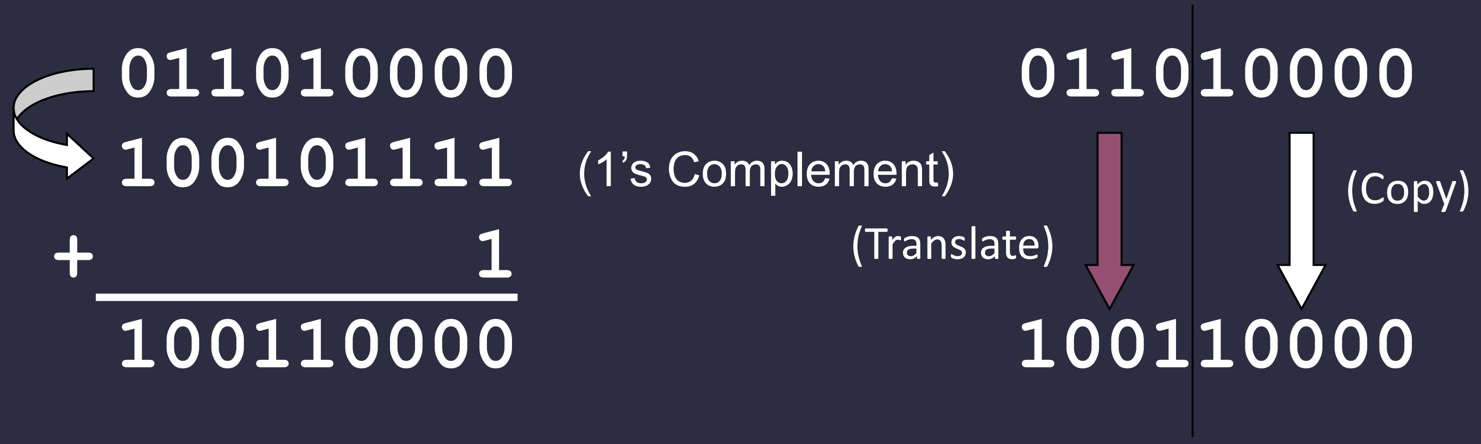
- If the value to be expressed is 0 or positive ,
  - They are written as unsigned integers, with the largest bits filled with 0.
- If the number is negative ,
  - written as a positive number
  - Each bit is inverted (1's complement)
  - 1 is added to the result.

```
  00101   (5)
  11010   (1 's complement )
+     1
  11011   (-5)
```

```
  01001   (9)
  1 0110  ( 1's complement )
+     1
  10111   (-9)
```

Two’s Complement

- Shortcut to find Two’s complement :
  - Copy bits of the number from right to left until you see the first "1"
  - Reverse remaining bits



Two’s complement

- Biggest bit sign bit and weight -2^{n-1} is .
- -2^{n-1} with n bits It can be expressed from 2^{n-1} to 1 .
  - The smallest negative number (-2^{n-1}) has no positive counterpart .

-2^3	2^2	2^1	2^0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

-2^3	2^2	2^1	2^0	
1	0	0	0	-8
1	0	0	1	-7
1	0	1	0	-6
1	0	1	1	-5
1	1	0	0	-4
1	1	0	1	-3
1	1	1	0	-2
1	1	1	1	-1

Convert binary complement to base 10

1. If the largest bit (leftmost) is 1, take the twos complement of the number and find its positive value.
2. Add the values by multiplying by powers of 2, starting with the rightmost bit.
3. If the number is negative when starting the process (i.e. its leftmost bit is 1), put a - sign on the base 10 number that appears.

$X = 01101000_{\text{binary}}$
$= 2^6 + 2^5 + 2^3 = 64+32+8$
$= 104_{\text{tens}}$

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Convert binary complement to base 10

$X = 00100111_{binary}$
$= 2^5 + 2^2 + 2^1 + 2^0 = 32+4+2+1$
$= 39_{tens}$

$X = 11100110_{binary}$
$-X = 00011010$
$= 2^4 + 2^3 + 2^1 = 16+8+2$
$= 26_{tens}$
$X = -26_{tens}$

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Decimal to Binary Complement Conversion

- Method 1 : *Division*

1. Get the absolute value of the decimal number . (It should always be positive .)
2. Divide by two – remainder is the smallest bit .
3. Keep dividing until you find 0, and write the remainder of the divisions from right to left.
4. Add zeros to the most right for completing width of the number. (in the example below the number is assumed to be 8 bits)
   If the decimal number is negative, take the binary complement of the resulting number.

$X = 104_{tens}$

104/2	=	52 k0	*bit 0*
52/2	=	26 k0	*bit 1*
26/2	=	13 k0	*bit 2*
13/2	=	6 k1	*bit 3*
6/2	=	3 k0	*bit 4*
3/2	=	1 k1	*bit 5*
1/2	=	0 k1	*bit 6*

$X = 01101000_{binary}$

Decimal to Binary Complement Conversion

- Second Method : *Subtracting powers of 2*

1. Get the absolute value of the decimal number.
2. Subtract the number less than or equal to the number from the powers of 2.
3. Place 1 in the relevant place .
4. Keep going until you get 0 .
5. Add zeros to the most right for completing width of the number. If the decimal number is negative, take the binary complement of the resulting binary number

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Decimal to Binary Complement Conversion

$X = 104_{tens}$

104 - 64	=	40	*bit 6*
40 - 32	=	8	*bit 5*
8 - 8	=	0	*bit 3*

$X = 01101000_{binary}$

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Addition

- Binary complement numbers is similar to the addition of unsigned numbers. No control mechanism is required.
    - The hand bit to be obtained from the largest bit is discarded.

```
   01101000  (104)
+  11110000  (-16)
   ---------
   01011000  (88)
```

Subtraction

- Find the negative form of the second number and add .
  - Binary complement of the second number and add it with the first number

```
   01101000  (104)
–  00010000  (16)
   --------
   01101000  (104)
+  11110000  (-16)
   --------
   01011000  (88)
```

Sign Extension

- When adding two numbers, both numbers must have the same bit width.
- If we just add 0 to the left of the two numbers to make them the same bit width;

4-bit		8-bit	
0100	(4)	00000100	(currently 4)
1100	(-4)	00001100	(12, not -4)

- For correct calculation, the sign bit of the number is placed where it will be expanded.

4-bit		8-bit	
0100	(4)	00000100	(currently 4)
1100	(-4)	11111100	(currently -4)

Overflow

- When numbers are very large , the sum may turn out to be too large to be expressed in n-bit numbers .

```
   01000  (8)          11000  (-8)
+  01001  (9)        + 10111  (-9)
   -----                -----
   10001  (-15)         01111  (+15)
```

- Overflow status :
  - It can happen in addition operations where both numbers have the same sign .

Logic Operations

- Are calculated as
  - There are two cases, True =1, False =0

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

A	Not A
0	1
1	0

Logic Operation Examples

- And
  - With 0 = result is 0
  - With 1 = result no change

```
     11000101
and  00001111
     --------
     00000101
```

- Or
  - With 0 or operation = no change
  - With 1 or operation = 1

```
     11000101
or   00001111
     --------
     11001111
```

- Not
  - It changes every bit.

```
not  11000101
     --------
     00111010
```

Hexadecimal Notation

- It is a 16 bit format that is frequently used on computers.
  - Each 4 bits of a binary number represents a hexadecimal representation.
  - It provides fewer mistakes than using long 0's and 1's.

Bin	Hex	Dec
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7

Bin	Hex	Dec
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Converting from Binary to Hexadecimal

- Each 4 bits equals 1
  - They are grouped starting from the right.

0111	1010	1000	1111	0100	1101	0111
3	A	8	F	4	D	7

Decimal Numbers : Fixed -Point Representation

- Decimals expression
  - A point is chosing for seperating integer and fraction parts
  - Addition and subtration operations are calculating as twos complement operations

$2^{-1} = 0.5$

$2^{-2} = 0.25$

$2^{-3} = 0.125$

```
   00101000.101 (40.625)
+  11111110.110 (-1.25)
   ------------
   00100111.011 (39.375)
```

Texts : ASCII Characters

- The ASCII table is an 8-bit table. Each number between 0-255 has a corresponding character or control signal.

00	nul	10	dle	20	sp	30	0	40	@	50	P	60	`	70	p
01	soh	11	dc1	21	!	31	1	41	A	51	Q	61	a	71	q
02	stx	12	dc2	22	"	32	2	42	B	52	R	62	b	72	r
03	etx	13	dc3	23	#	33	3	43	C	53	S	63	c	73	s
04	eot	14	dc4	24	$	34	4	44	D	54	T	64	d	74	t
05	enq	15	nak	25	%	35	5	45	E	55	U	65	e	75	u
06	ack	16	syn	26	&	36	6	46	F	56	V	66	f	76	v
07	bel	17	etb	27	'	37	7	47	G	57	W	67	g	77	w
08	bs	18	can	28	(	38	8	48	H	58	X	68	h	78	x
09	ht	19	em	29	)	39	9	49	I	59	Y	69	i	79	y
0a	nl	1a	sub	2a	*	3a	:	4a	J	5a	Z	6a	j	7a	z
0b	vt	1b	esc	2b	+	3b	;	4b	K	5b	[	6b	k	7b	{
0c	np	1c	fs	2c	,	3c	<	4c	L	5c	\	6c	l	7c	|
0d	cr	1d	gs	2d	-	3d	=	4d	M	5d	]	6d	m	7d	}
0e	so	1e	rs	2e	.	3e	>	4e	N	5e	^	6e	n	7e	~
0f	si	1f	us	2f	/	3f	?	4f	O	5f	_	6f	o	7f	del

Other Data Types

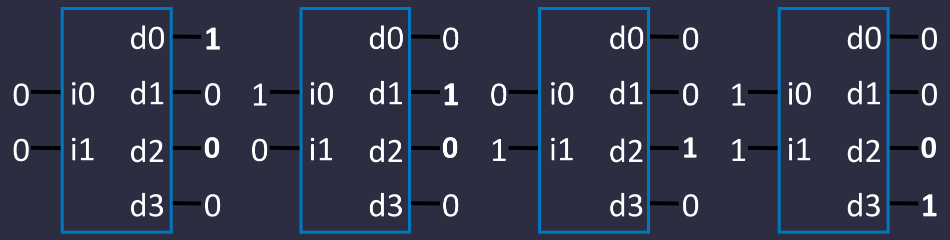
- Texts
  - Formed by sequential writing of characters
- Image
  - They are formed by the combination of pixels.
    - Black and White : 1 bit (1/0 = black / white)
    - Color : Red, Blue, Green (RGB) comp1nts are available. Each is stored as 8-bit numbers.
- Sound
  - It is usually represented as a sequential recording of fixed-point notation numbers.

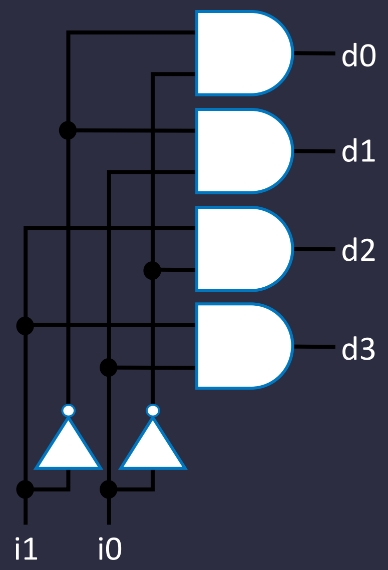
Design of Processor Blocks with

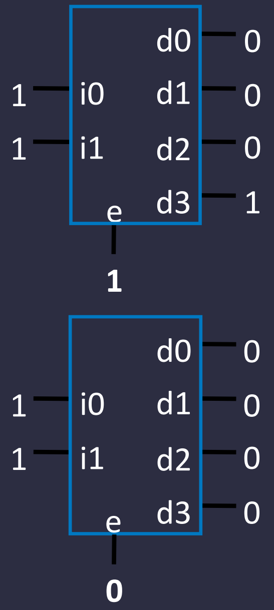
- Number of Transistors in Processors
  - Intel 4004 (1971): 2250
  - Intel 8088 (1979): 29k
  - AMD K6 (1997): 7.5 million
  - Intel Pentium 4 (2006): 184 Million
  - Intel I7 Haswell -E (2014): 2.6 Billion
  - AMD Epyc Rome (2019): 32 Billion

Decoder and MUX

- **Decoder** : Activates the pin corresponding to the number received in the input.
- 2 -input decoder: There are 4 possible inputs.
  - It has 4 outputs
- Enable with pin decoder
  - If e=0 , all outputs are 0
  - If e=1 , it behaves normally
- N input decoder: 2 ^{n} exit

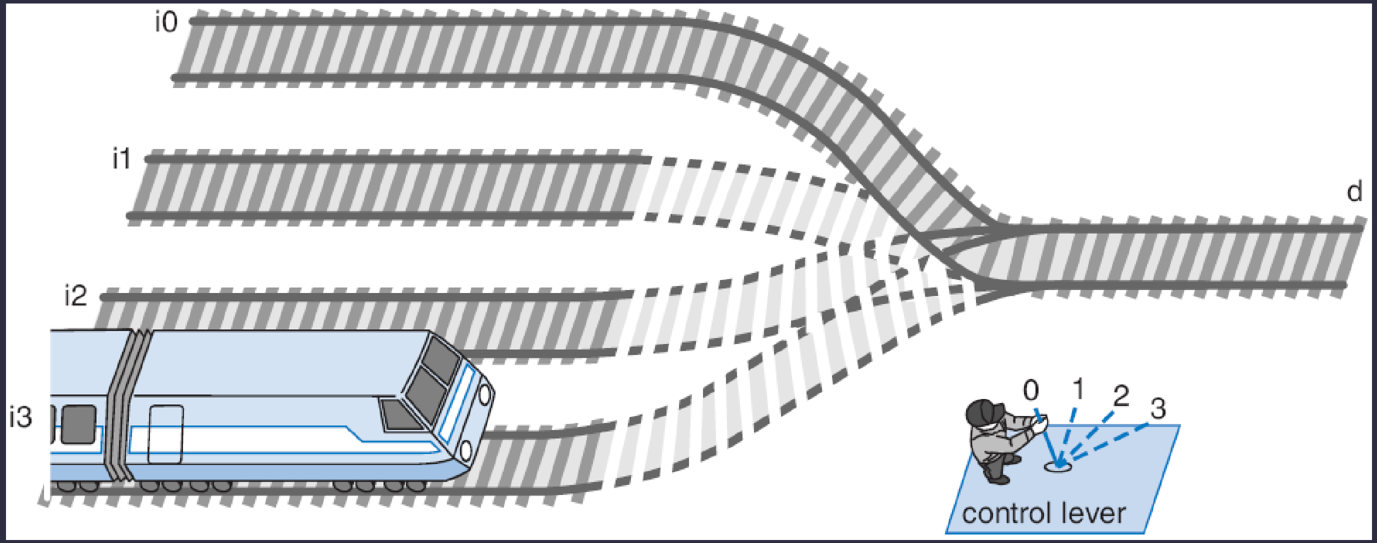




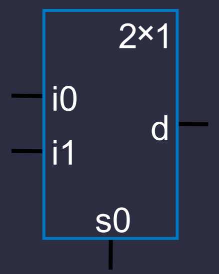


Multiplex e r (Mux)

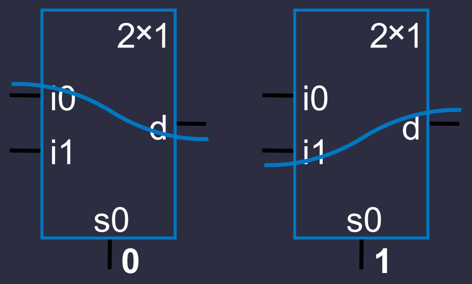
- Mux: It is a combinational circuit. Outputs incoming inputs according to the select bit.
  - 4 input mux → 2 select inputs
  - 8 input mux → 3 select input
  - N inputs → log $_2$ (N) select input

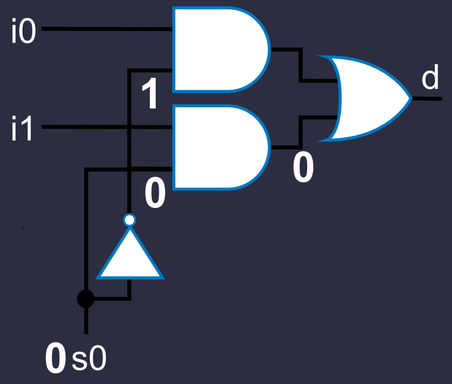


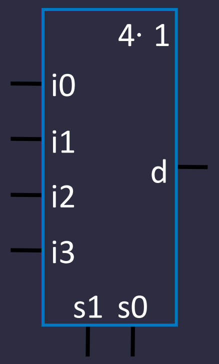
Mux Internal Structure



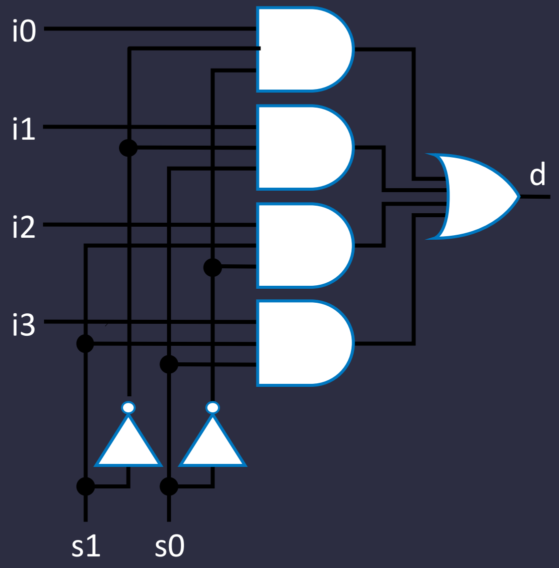
2x1 mux



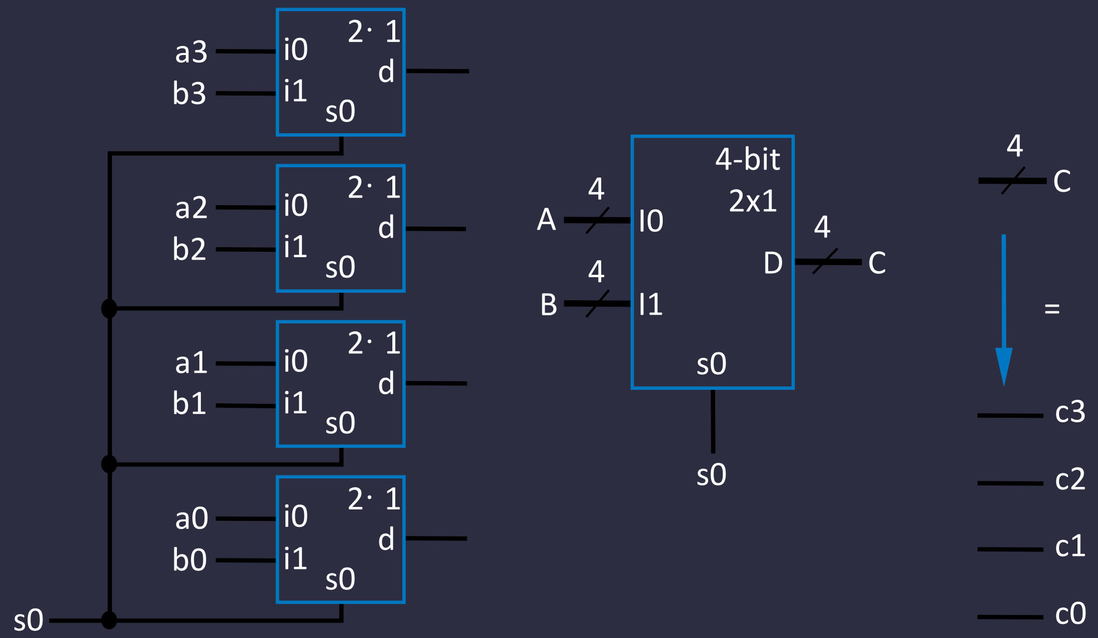




4x1 mux

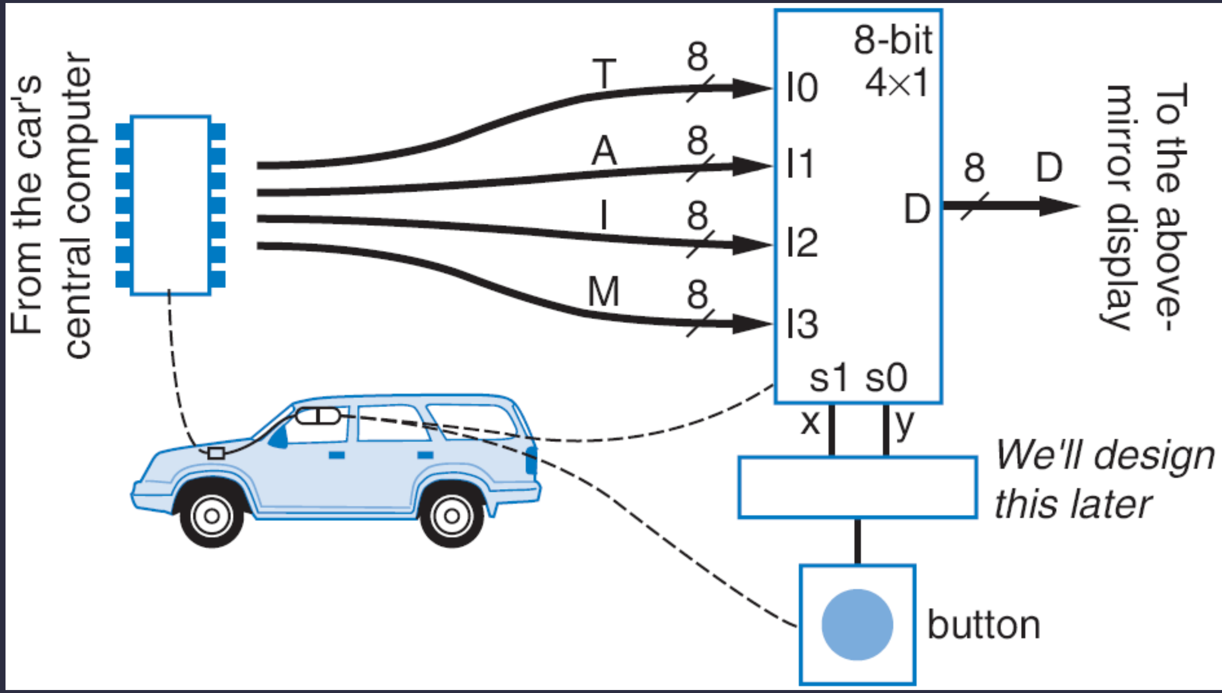


MUX Merge



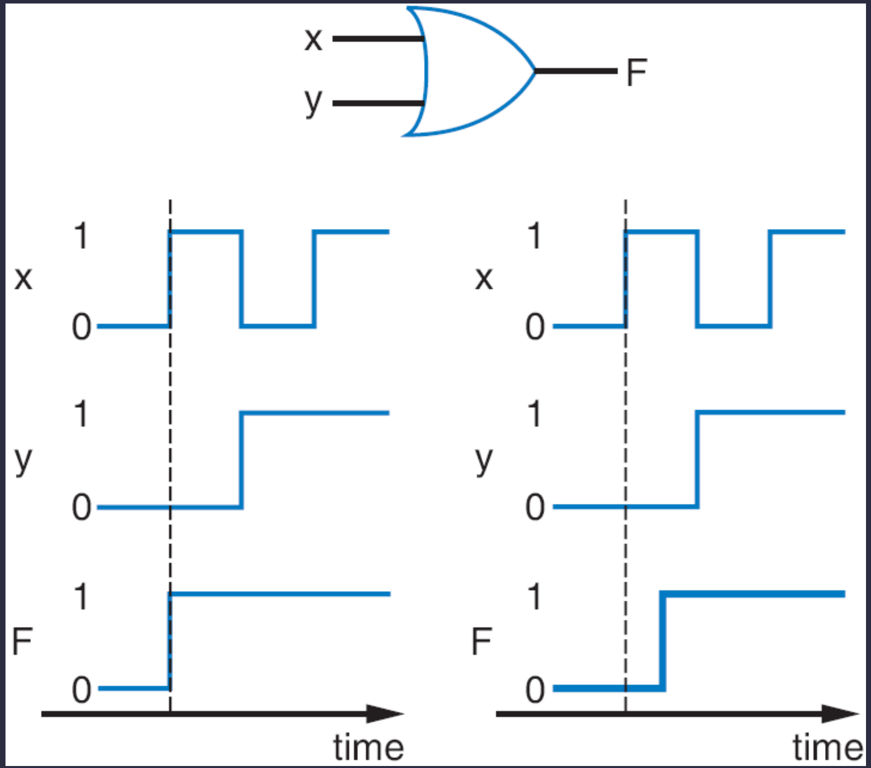
- Example : Two 4-bit inputs , A (a3 a2 a1 a0) and B (b3 b2 b1 b0)
  - 4-bit 2x1 MUX can be done using 4 1 bit, 2x1 MUX

N-bit MUX Example



- There are 4 possible texts to display
  - Temperature , Average Fuel Usage , Average Speed , KM Remaining - all
  - Which one will appear on the screen is selected with the x and y bits.
  - 8-bit 4x1 MUX can be used.

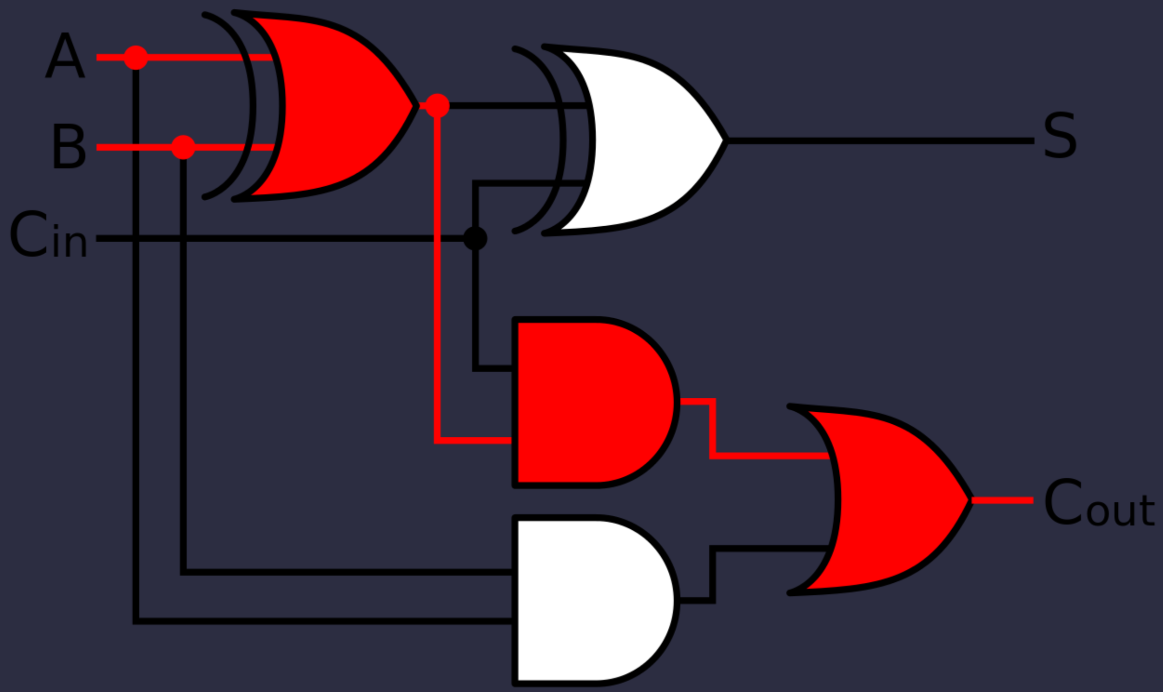
Gate Delays



- All circuits have a delay.
  - Outputs don't change instantly

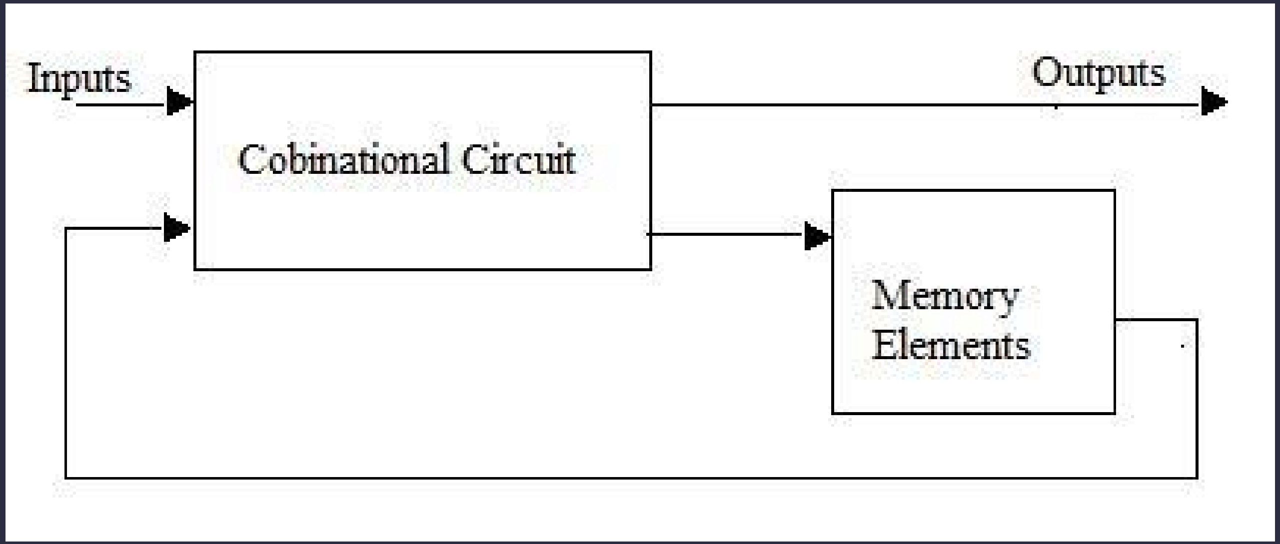
From Logic Gates to Control Units

- *Combinational Circuits*
  - The output of the circuit depends on the current input.
  - The output delay of the circuit depends on the longest path in the circuit.



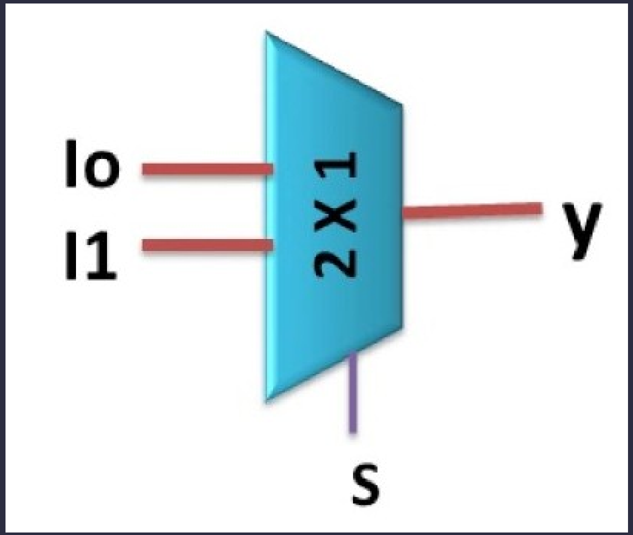
From Logic Gates to Control Units

- *Sequential Circuits*
  - The output depends on both the current input and the values in memory.
  - Some outputs of the circuit are stored in memory and reused.
  - We'll get into the details next week.

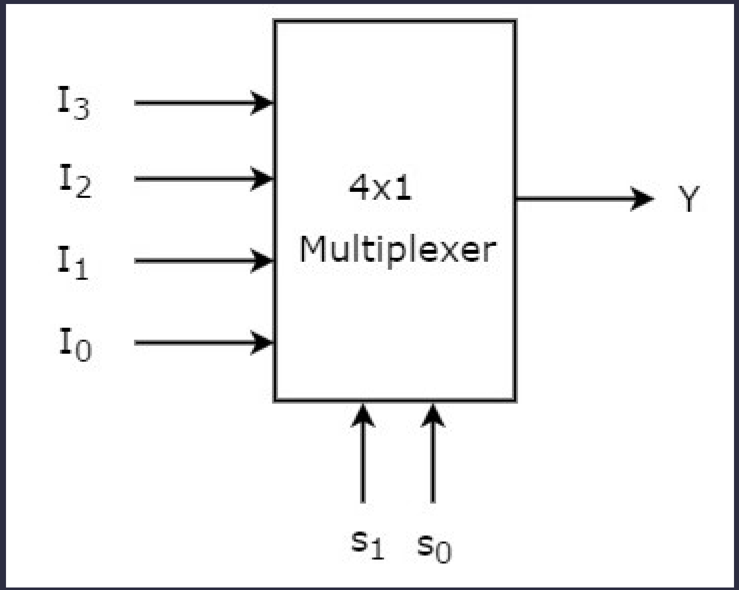


Multiplexer - MUX

- *n* - bit select, *2 ⁿ* input and It has only one output.
  - According to the select bit, the value from the input is transferred to the output.



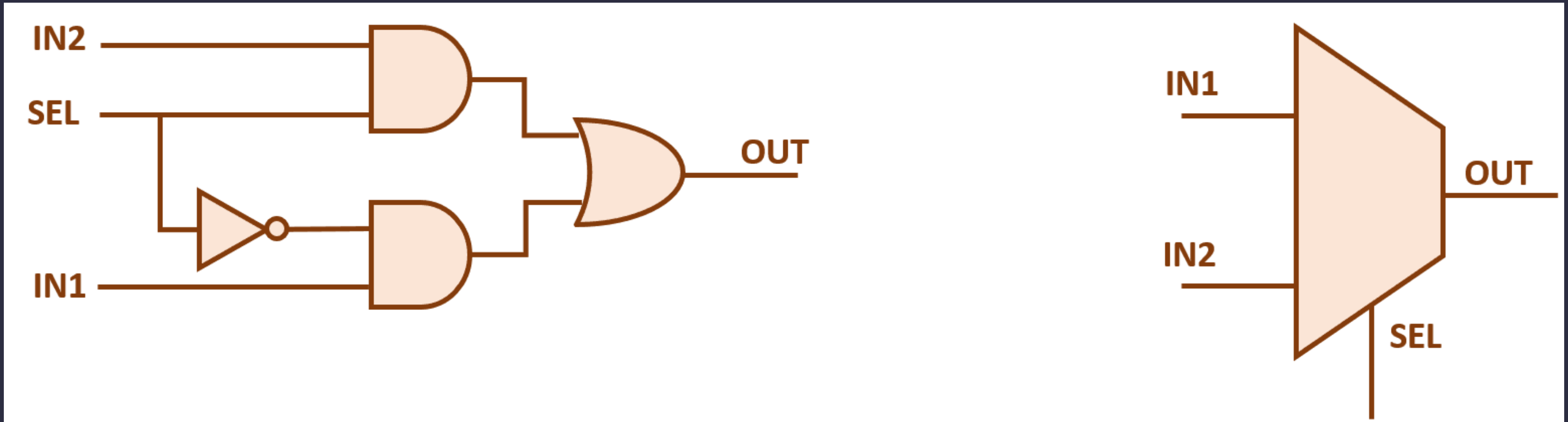
2 -1 MUX



4-1 MUX

Selector (Multiplexer - MUX)

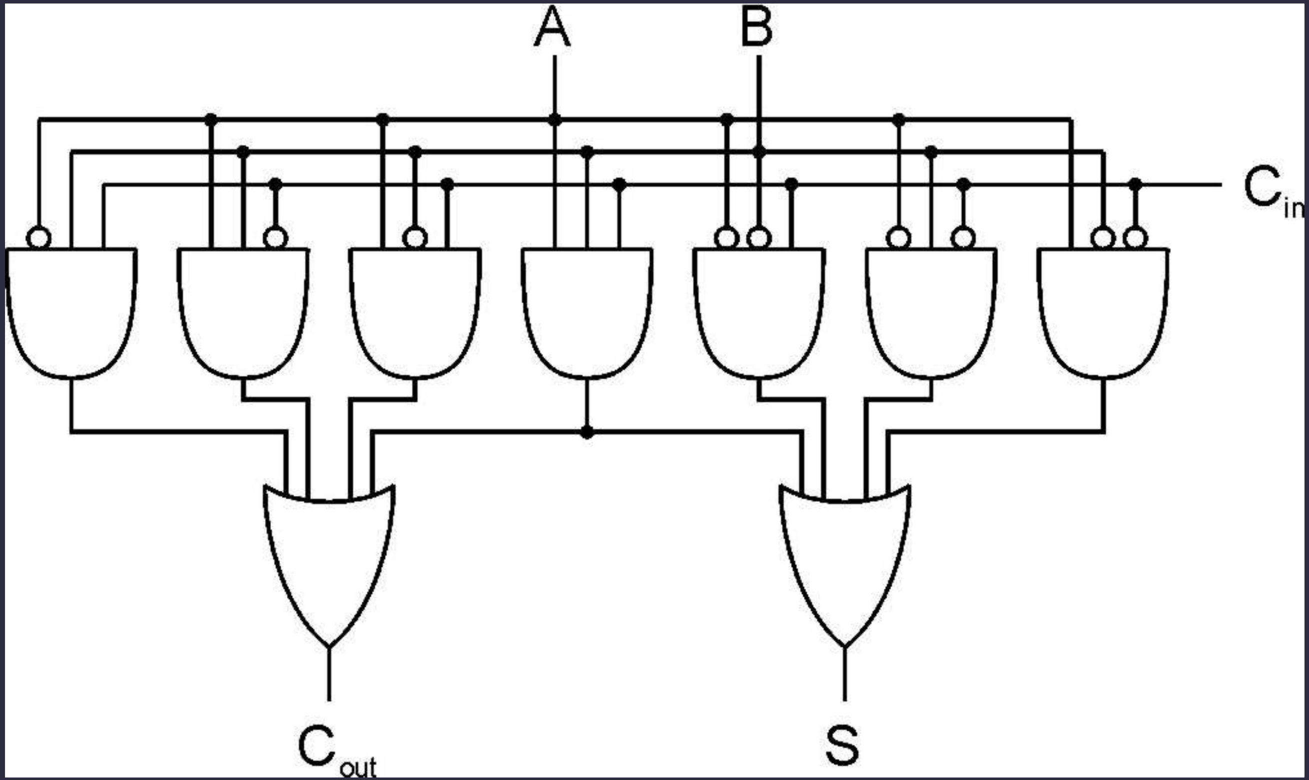
- n - bit select, 2^n input and It has only one output.
  - According to the select bit, the value from the input is transferred to the output.



2 -1 MUX

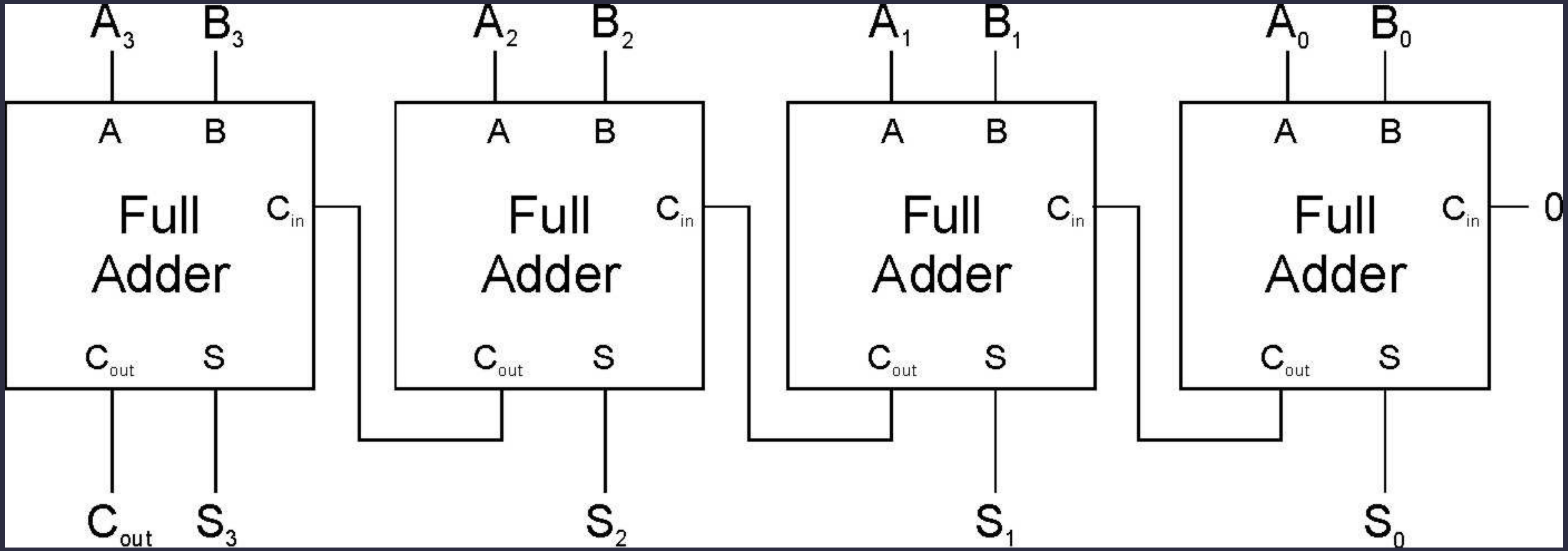
Full Adder

- Taking two bits (A and B) and a carry input (Cin), it produces a one-bit sum (S) and carry (Cout) .



A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

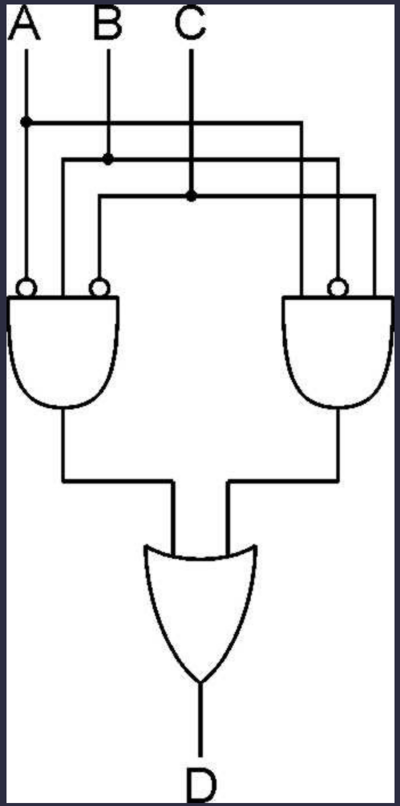
4-bit Adder



Other Circuits

- Any circuit can be expressed with And, Or and Not gates.

A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

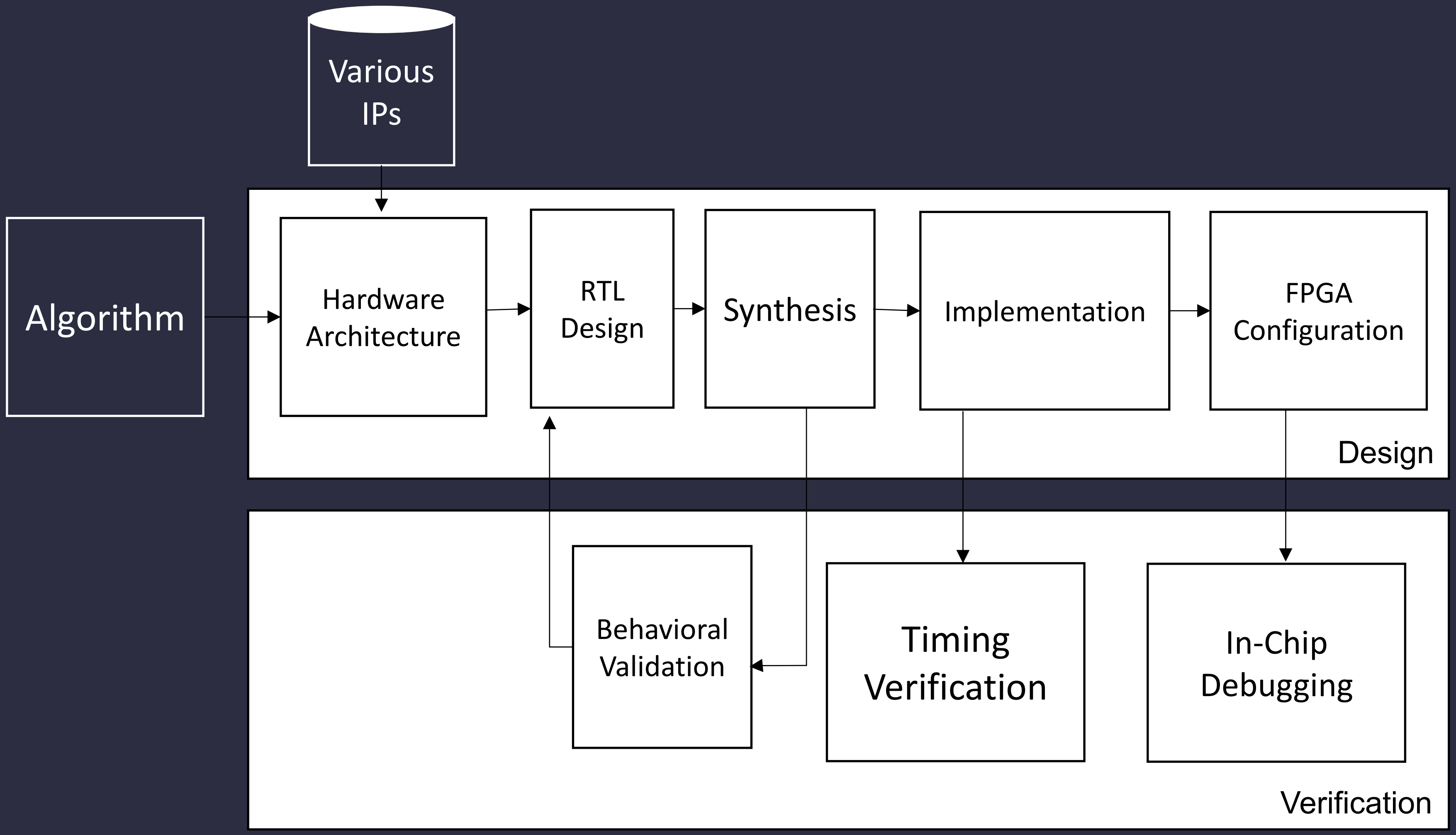


1. In the truth table, do and operation for 1 outputting rows

2. Combine these and gates with or gate

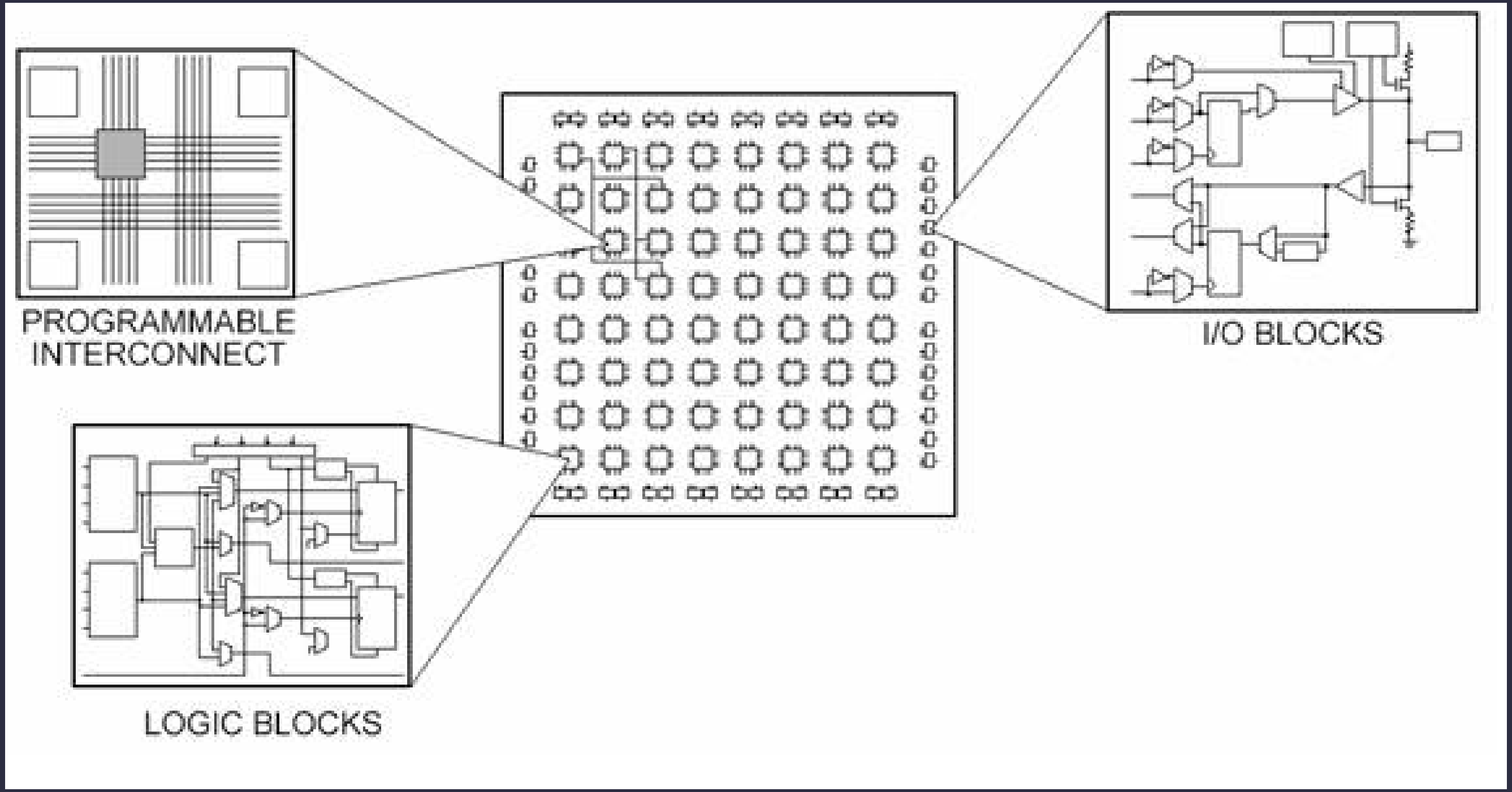
Vivado Design Tool

- *Design Flow*



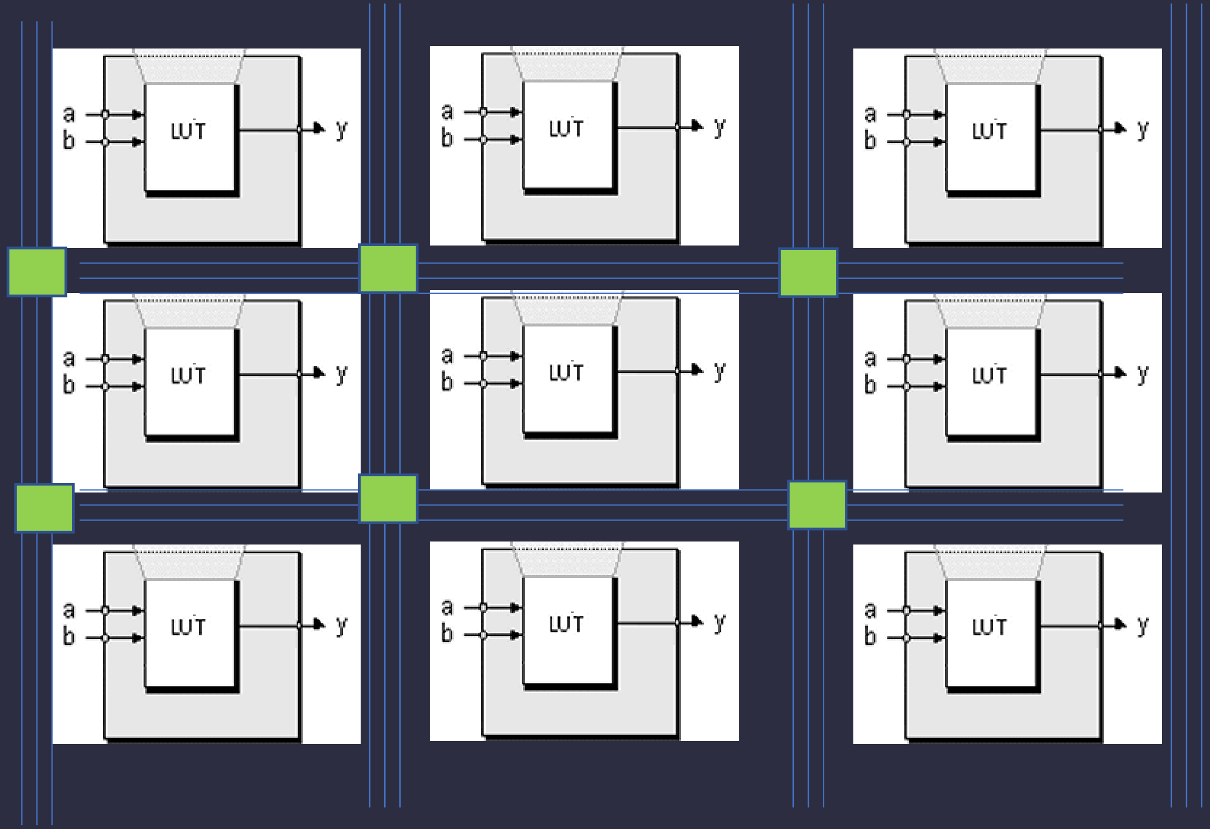
Verilog – Combinational Circuits

What is FPGA?



Verilog – Combinational Circuits

What is FPGA?



What is FPGA?

- FPGA (Field programmable Gate Arrays) is an integrated chip.
- It contains programmable blocks and configurable connections between these blocks.
- By programming these blocks and connections, the desired circuit can be implemented in the FPGA .

Example FPGA Chip

What is FPGA?

- Some FPGAs can only be programmed once.
- It is called one-time programmable (OTP).

What is FPGA?

- "Field Programmable" means that it can be programmed in the required application with unlimited count.

- This means that after the FPGA chips are produced in the factory, they can be used in the desired design later on.

Why is FPGA important?

Wide variety of integrated circuits (IC – Integrated circuits) are available.

- Memory
- Microprocessors
- Programmable logic devices (Programmable logic devices)
  - SPLD (Simple)
  - CPLD (Complex)
- ASIC – Application specific integrated circut
- And FPGAs ...

ASICs

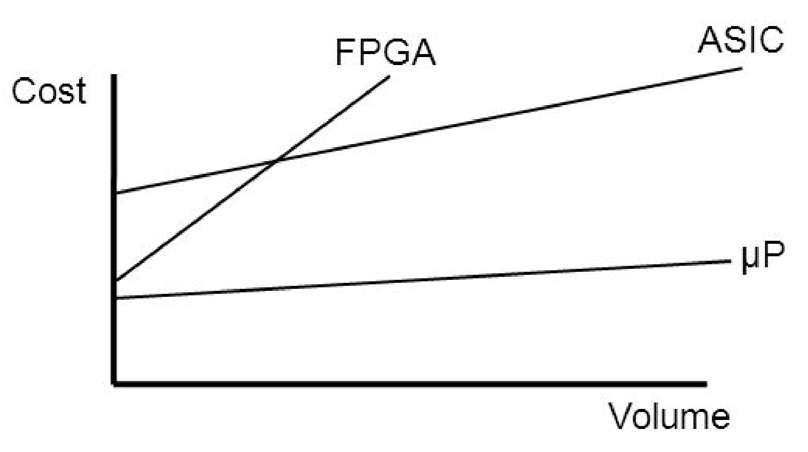
- They are useful in the implementation of huge and complex circuits.
- ASICs are integrated circuits built to perform a specific operation.
- ASICs provide very high performance (operation at high frequencies) , ASIC design complexity and design time and costs are quite high.
- And the produced chips cannot be modified . In case of a fault with the chip, all produced chips will be thrown away.

FPGAs

- FPGAs It is programmable.
- Unlike ASICs ; In the case of an error in the design, the design can be repeatedly modified and tested.

FPGAs

- FPGAs They are much cheaper than ASICs . (ASICs are only cheap when millions of units are produced)
- FPGAs is much easier than creating an ASIC design.
- Due to the shorter design time, once a product is produced, the time to market is shorter.
- NRE (Non-recurring engineering) is high when developing a ASIC Design



FPGA is mainly used

- Telecommunication
- Networking
- Automotive
- Medical
- Various industrial applications
- Prototypes of ASIC designs
- DSP (Digital signal processor) applications
- SoC (System on Chip), a single IC where all necessary electronics are gathered together

Reasons for FPGA Use

- Computing Power
- Controlling with nanosecond order

Reasons for FPGA Use

Processors

```
c[0]=a[0]+b[0];
c[1]=a[1]+b[1];
c[2]=a[2]+b[2];
c[3]=a[3]+b[3];
…
c[1000]=a[1000]+b[1000];
```

FPGA

```
c[0]<=a[0]+b[0];
c[1]<=a[1]+b[1];
c[2]<=a[2]+b[2];
c[3]<=a[3]+b[3];
…
c[1000]<=a[1000]+b[1000];
```

Reasons for FPGA Use

Processors

```
c[0]=a[0]+b[0];
c[1]=a[1]+b[1];
c[2]=a[2]+b[2];
c[3]=a[3]+b[3];
...
c[1000]=a[1000]+b[1000];
```

FPGA

```
c[0]<=a[0]+b[0];
c[1]<=a[1]+b[1];
c[2]<=a[2]+b[2];
c[3]<=a[3]+b[3];
...
c[1000]<=a[1000]+b[1000];
```

Reasons for FPGA Use

Processors

```
c[0]=a[0]+b[0];
c[1]=a[1]+b[1];
c[2]=a[2]+b[2];
c[3]=a[3]+b[3];
...
c[1000]=a[1000]+b[1000];
```

FPGA

```
c[0]<=a[0]+b[0];
c[1]<=a[1]+b[1];
c[2]<=a[2]+b[2];
c[3]<=a[3]+b[3];
...
c[1000]<=a[1000]+b[1000];
```

Reasons for FPGA Use

Processors

```
c[0]=a[0]+b[0];
c[1]=a[1]+b[1];
c[2]=a[2]+b[2];
c[3]=a[3]+b[3];
...
c[1000]=a[1000]+b[1000];
```

FPGA

```
c[0]<=a[0]+b[0];
c[1]<=a[1]+b[1];
c[2]<=a[2]+b[2];
c[3]<=a[3]+b[3];
...
c[1000]<=a[1000]+b[1000];
```

Reasons for FPGA Use

Processors

```
c[0]=a[0]+b[0];
c[1]=a[1]+b[1];
c[2]=a[2]+b[2];
c[3]=a[3]+b[3];
...
c[1000]=a[1000]+b[1000];
```

FPGA

```
c[0]<=a[0]+b[0];
c[1]<=a[1]+b[1];
c[2]<=a[2]+b[2];
c[3]<=a[3]+b[3];
...
c[1000]<=a[1000]+b[1000];
```

Reasons for FPGA Use

Processors

```
c[0]=a[0]+b[0];
c[1]=a[1]+b[1];
c[2]=a[2]+b[2];
c[3]=a[3]+b[3];
...
c[1000]=a[1000]+b[1000];
```

FPGA

```
c[0]<=a[0]+b[0];
c[1]<=a[1]+b[1];
c[2]<=a[2]+b[2];
c[3]<=a[3]+b[3];
...
c[1000]<=a[1000]+b[1000];
```

Reasons for FPGA Use

Processors

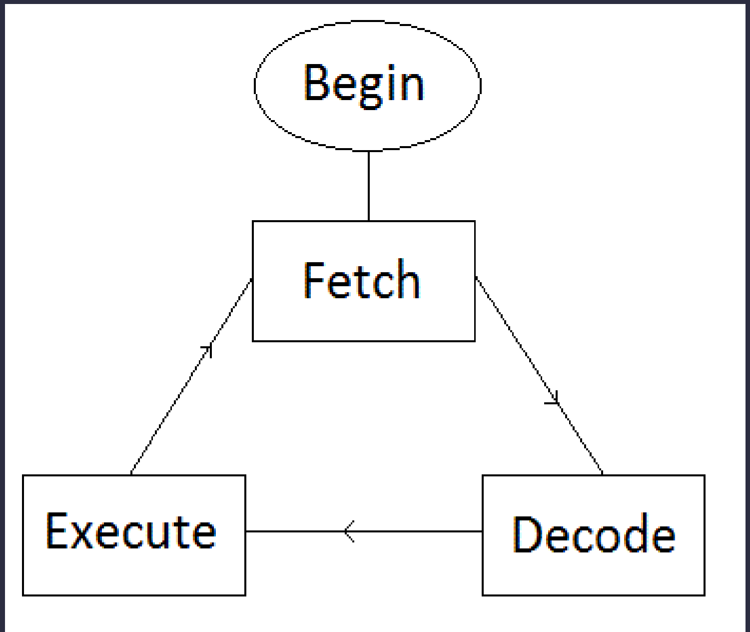
```
c[0]=a[0]+b[0];
c[1]=a[1]+b[1];
c[2]=a[2]+b[2];
c[3]=a[3]+b[3];
...
c[1000]=a[1000]+b[1000];
```

FPGA

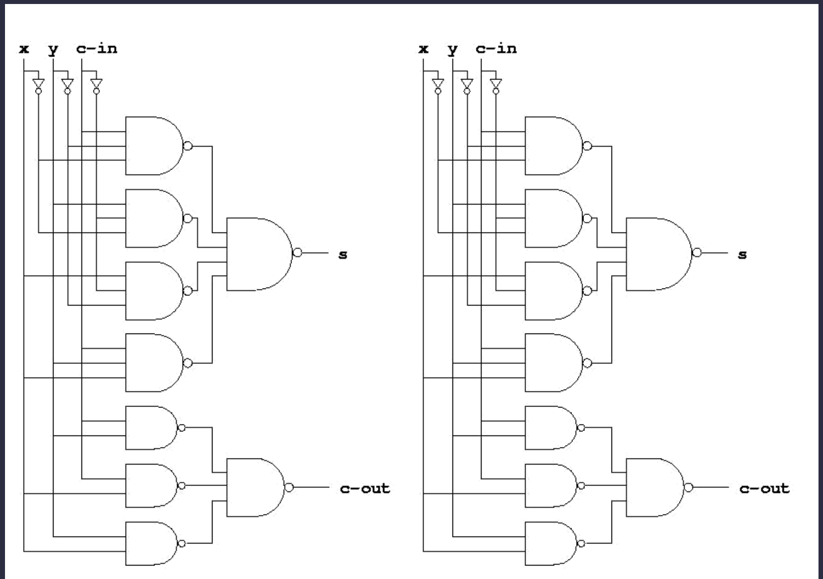
```
c[0]<=a[0]+b[0];
c[1]<=a[1]+b[1];
c[2]<=a[2]+b[2];
c[3]<=a[3]+b[3];
...
c[1000]<=a[1000]+b[1000];
```

Reasons for FPGA Use

Processors



FPGA



Reasons for FPGA Use

CPU

For 1000 transactions (assume each operation takes 1 cycle),

Average Processor frequency: 3 GHz,

Required Time = Number of operations X (1/Frequency)

= 1000 X 1/3 billion
= 333 nanoseconds

FPGA

For 1000 transactions (assume each operation takes 1 cycle),

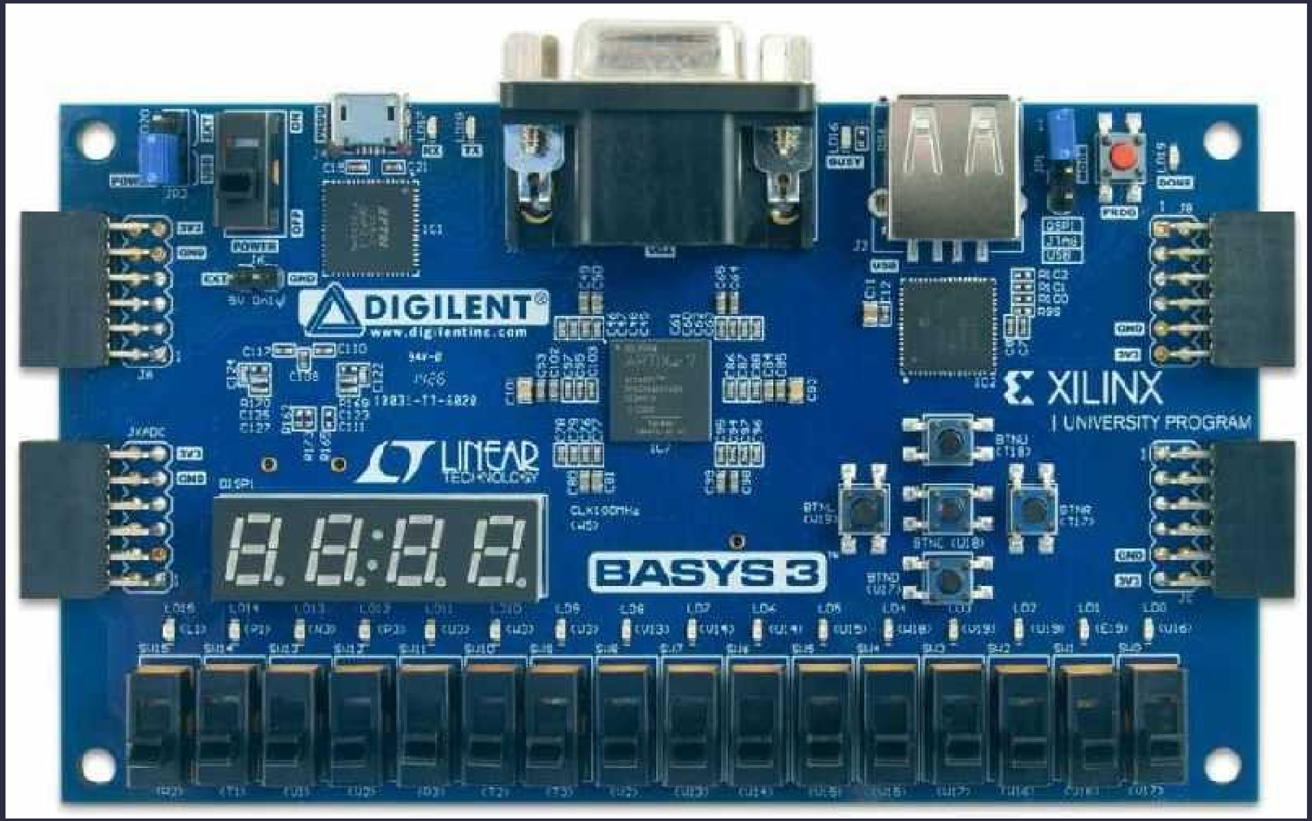
Average FPGA frequency: 100mhz

Time required = 1 x (1/Frequency)

= 1 X 1/100m
= 10 nanoseconds

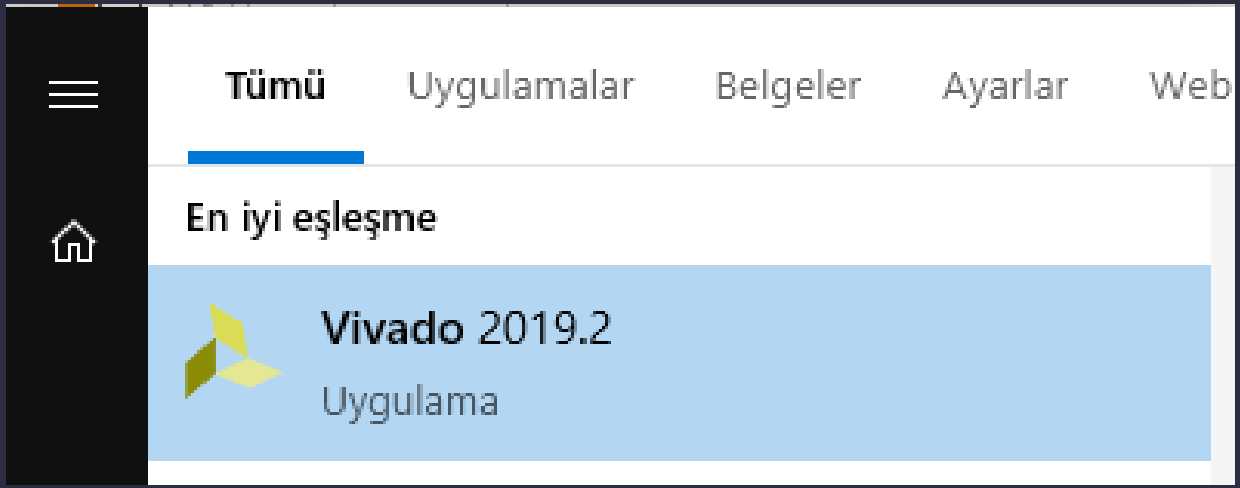
Verilog – Combinational Circuits

In LABs Xilinx Basys3 FPGAs with Artix 7 FPGAs will be used.



Verilog – Combinational Circuits

Vivado IDE will be used to programming Xilinx Based FPGAs

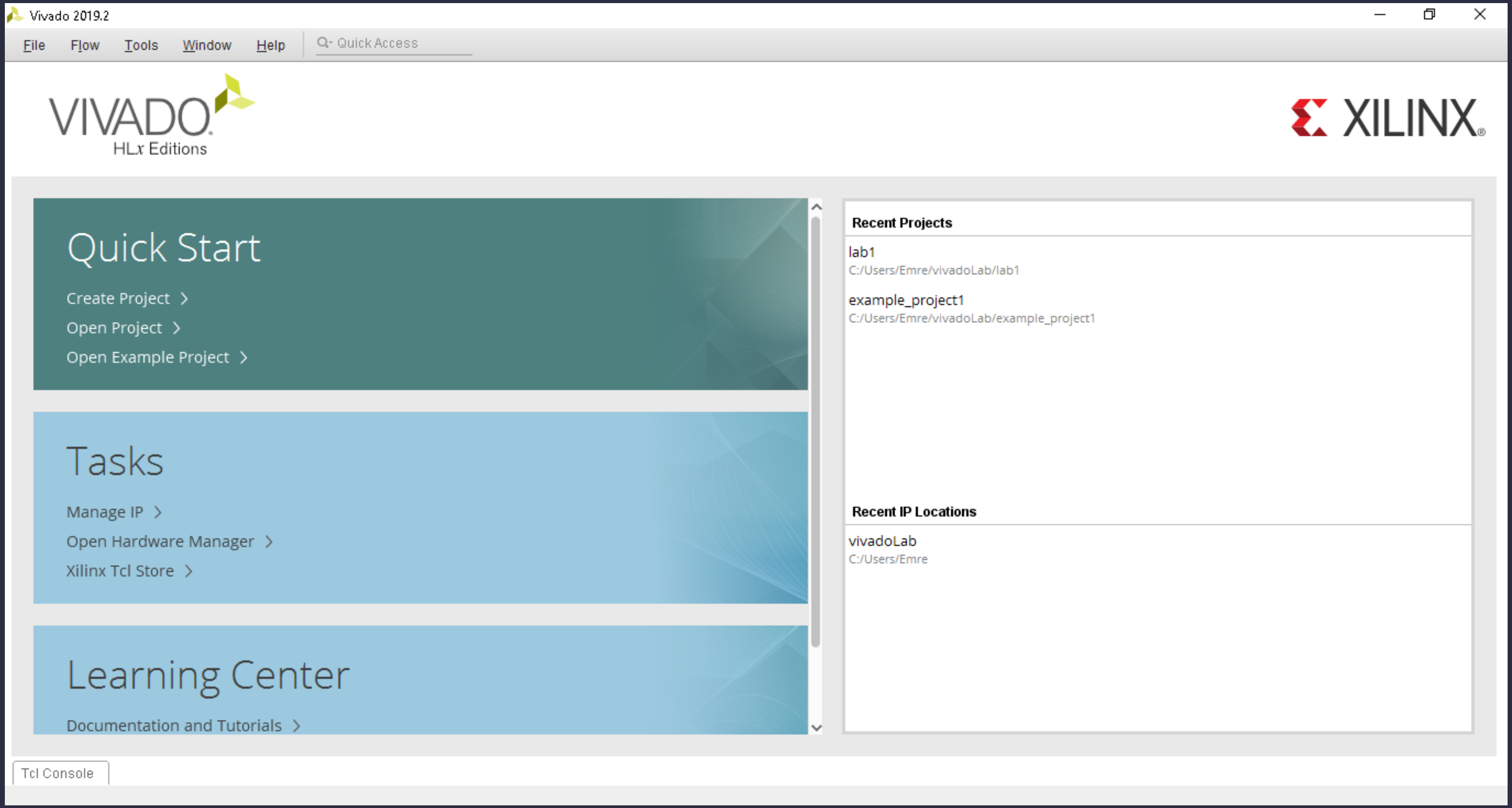


Vivado Design Tool

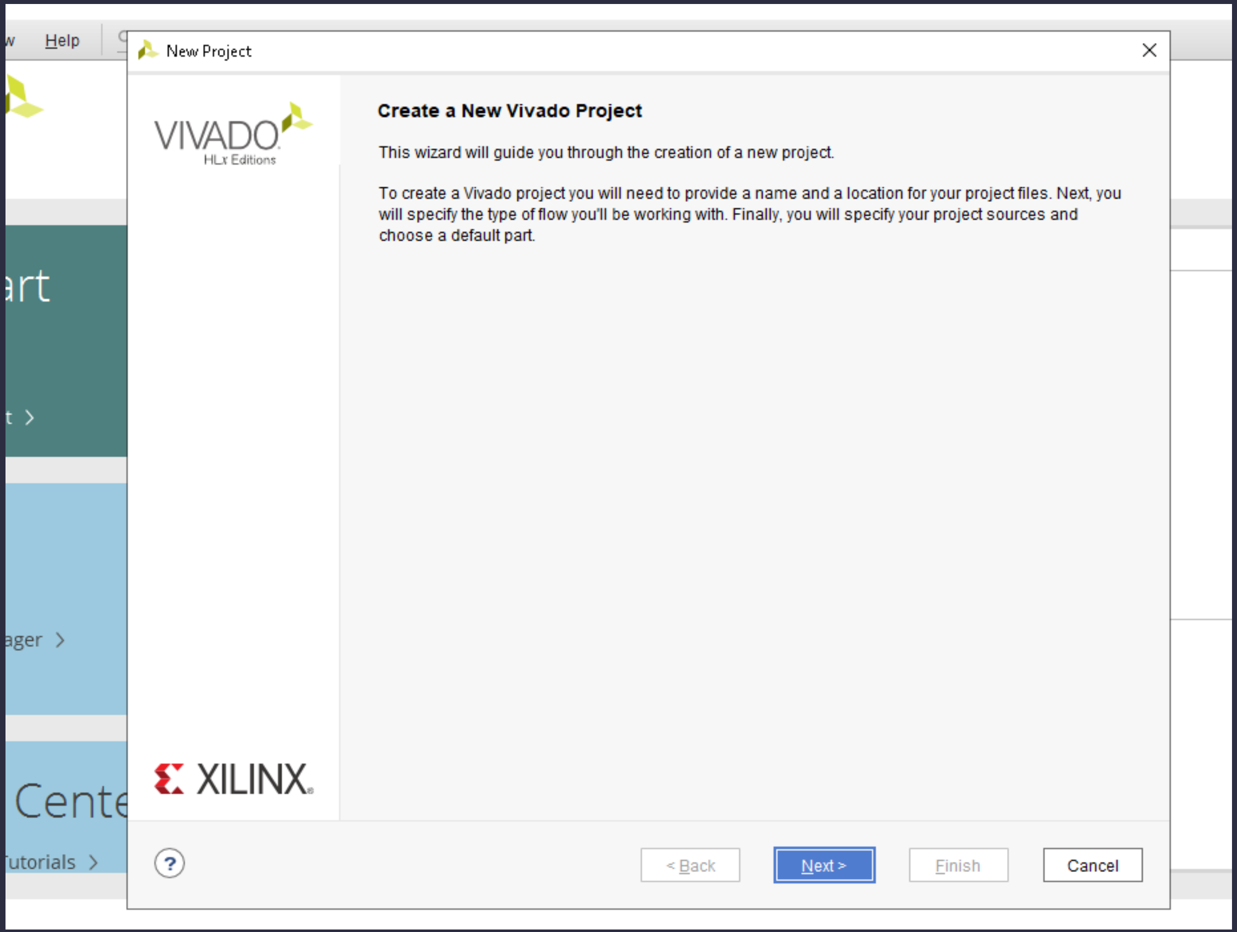
- Download Address: https://www.xilinx.com/support/download.html
- Installation and Licensing Video : https://www.youtube.com/watch?v=yW7t28XaVEs

Verilog – Combinational Circuits

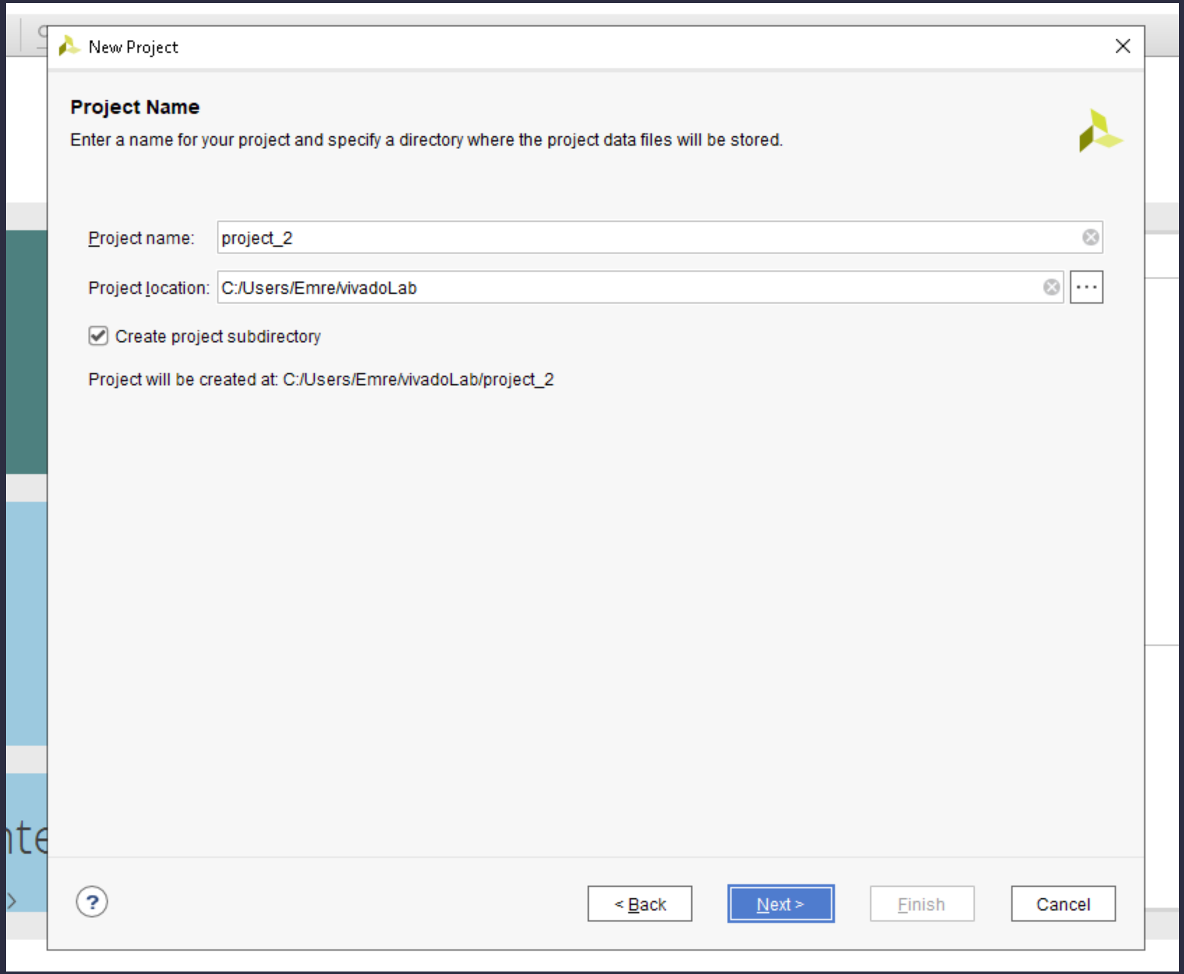
Vivado Design Tool



Vivado Design Tool

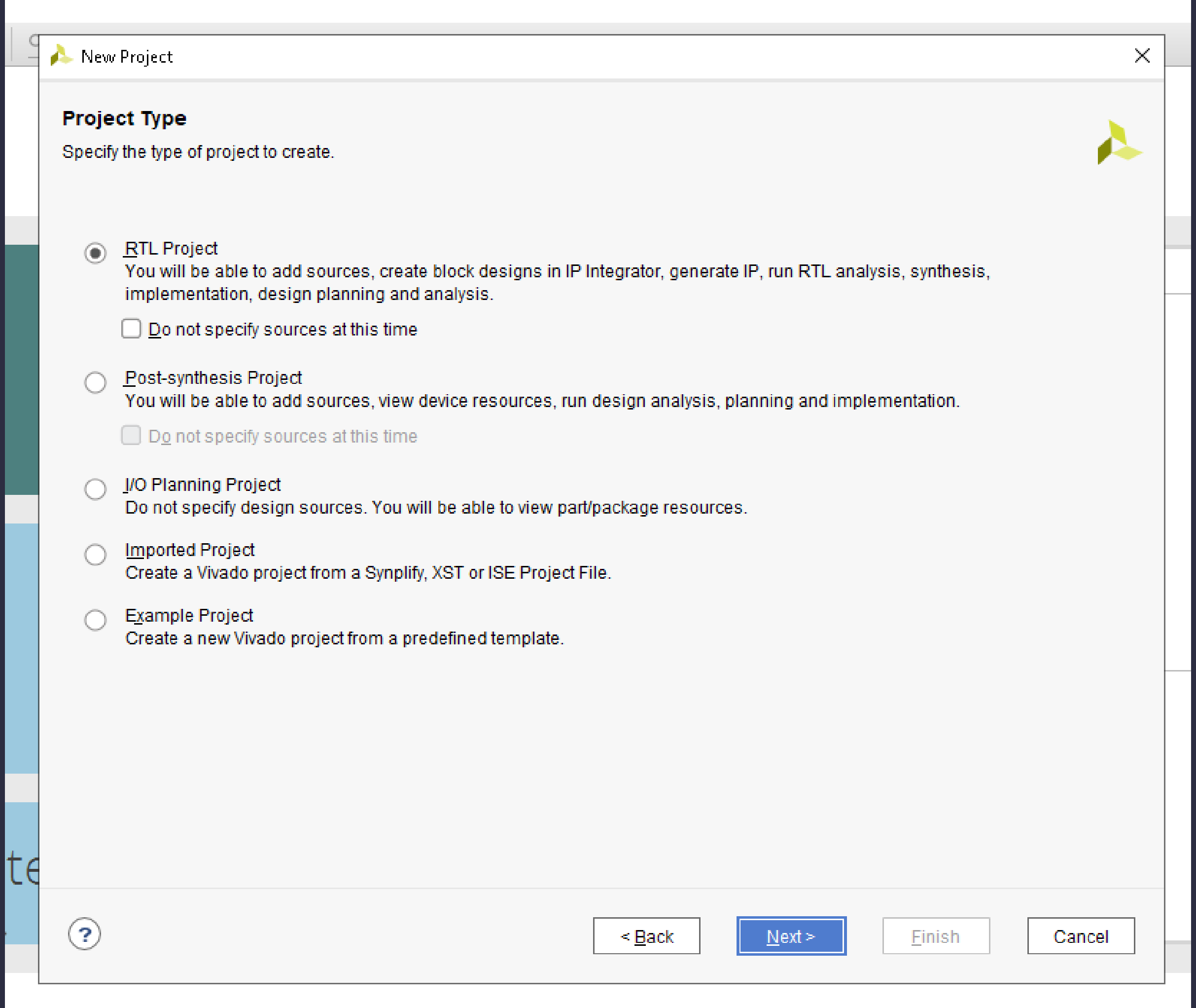


Vivado Design Tool



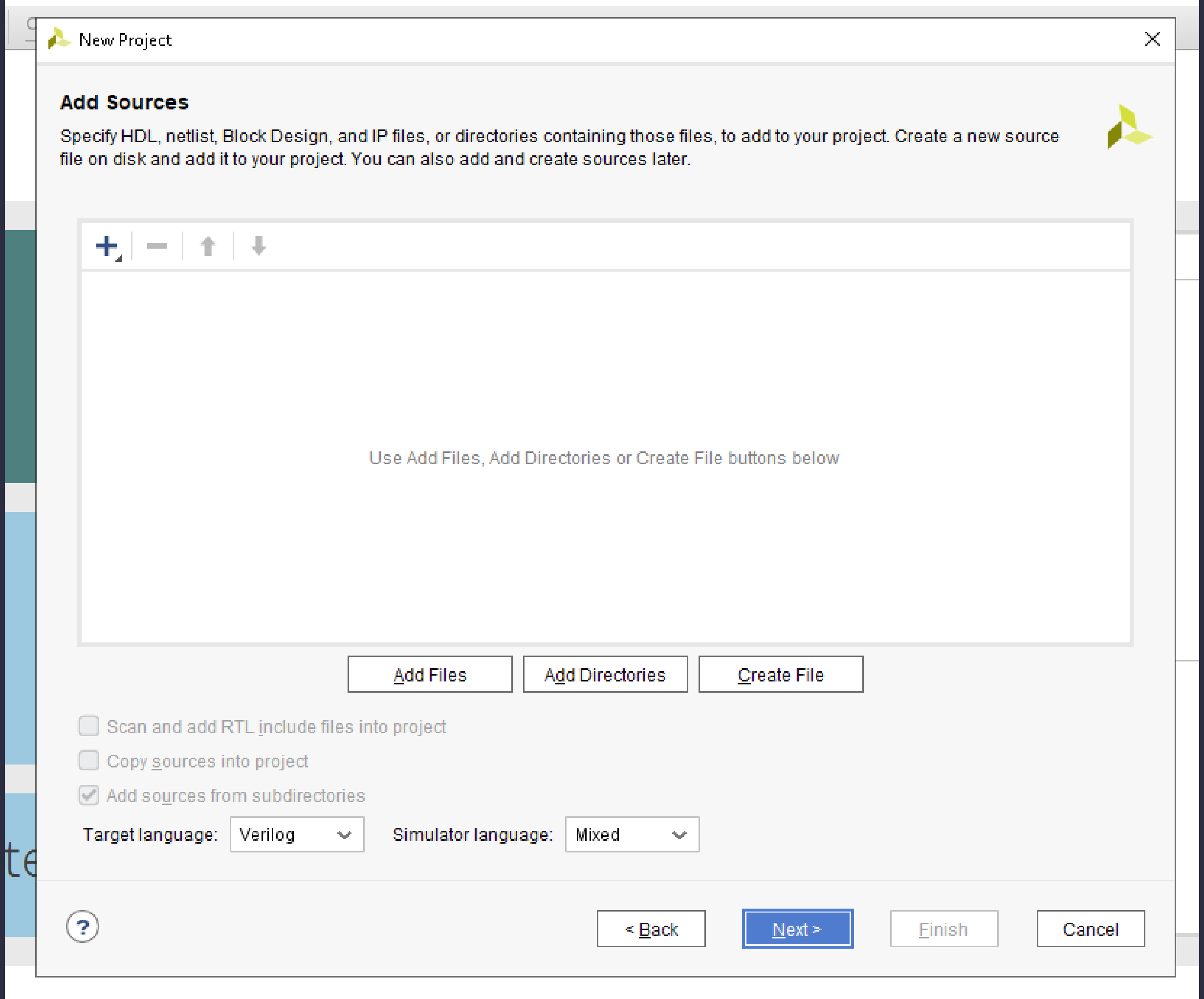
Verilog – Combinational Circuits

Vivado Design Tool

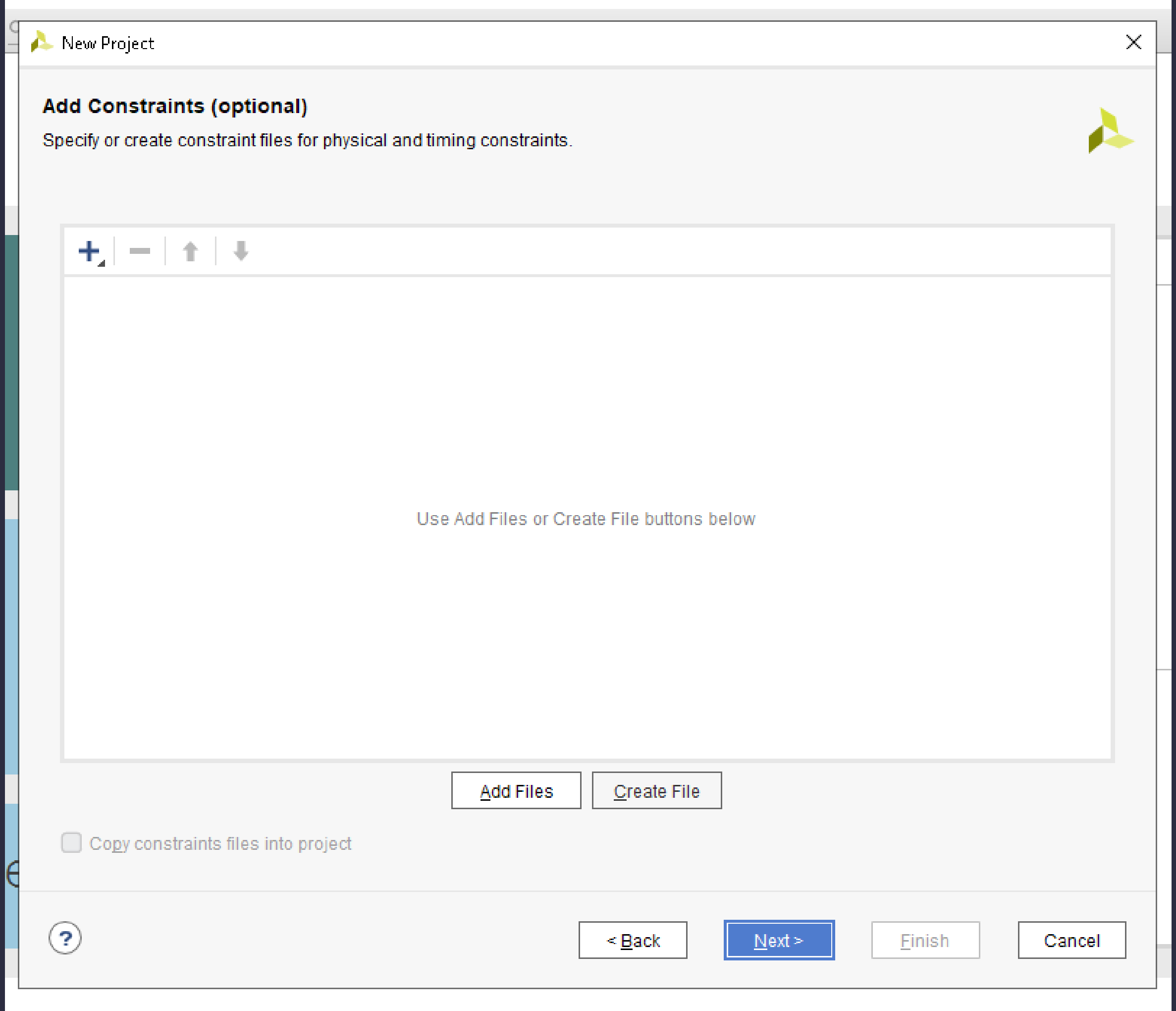


Verilog – Combinational Circuits

Vivado Design Tool



Vivado Design Tool



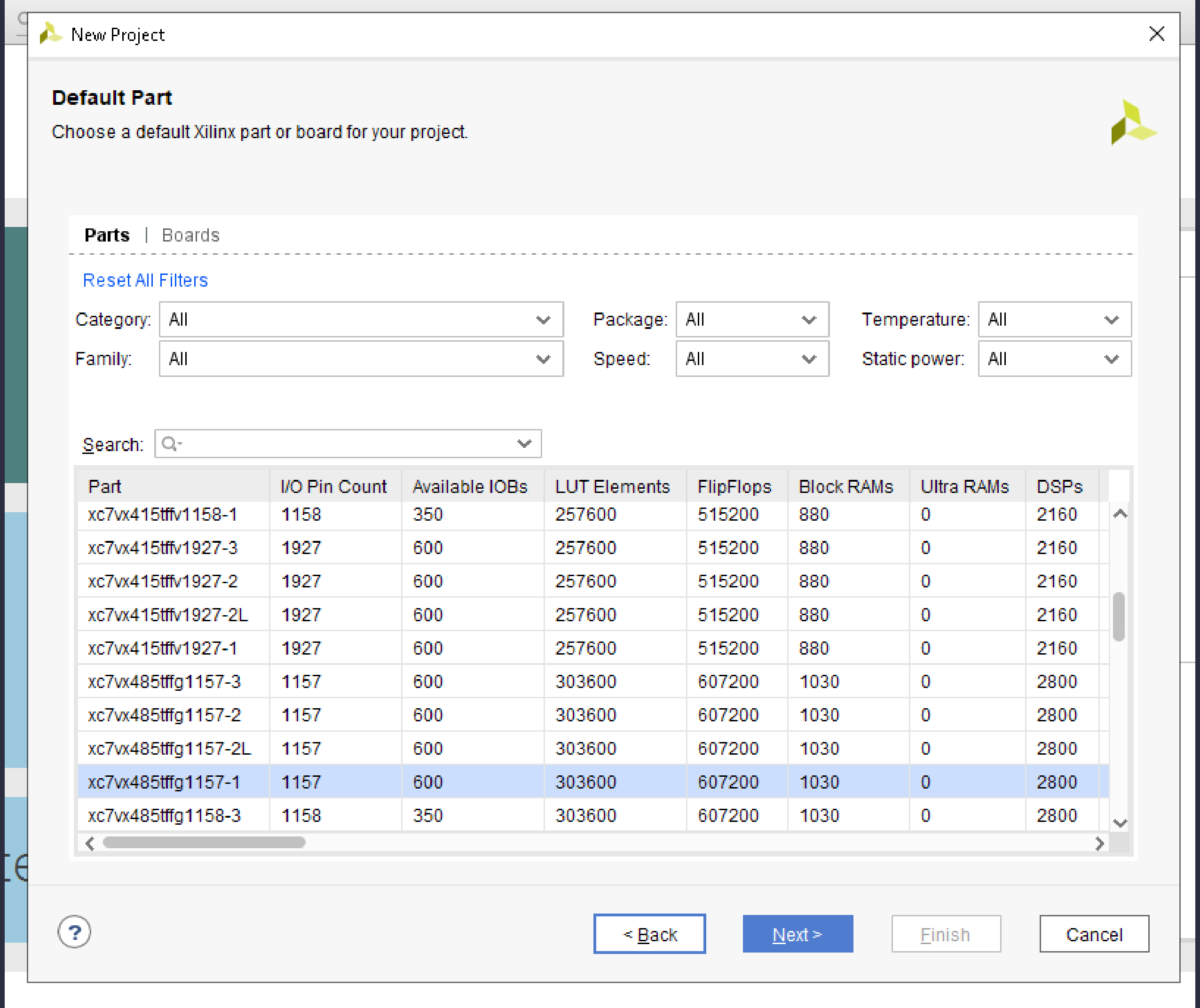
Verilog – Combinational Circuits

Vivado Design Tool

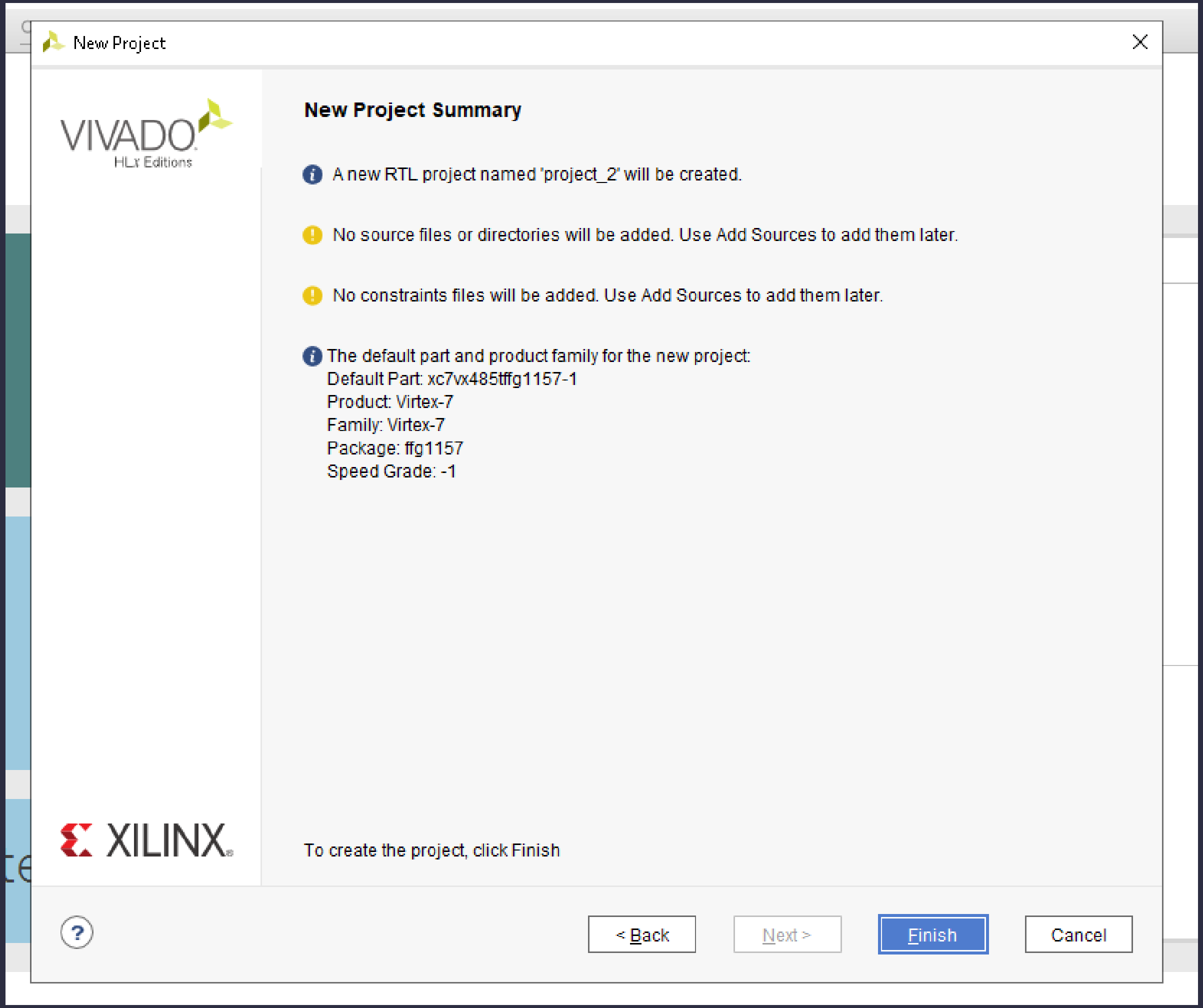
is the FPGA we will use in LABs

XC7A35Tcpg236-1

The model must be selected.

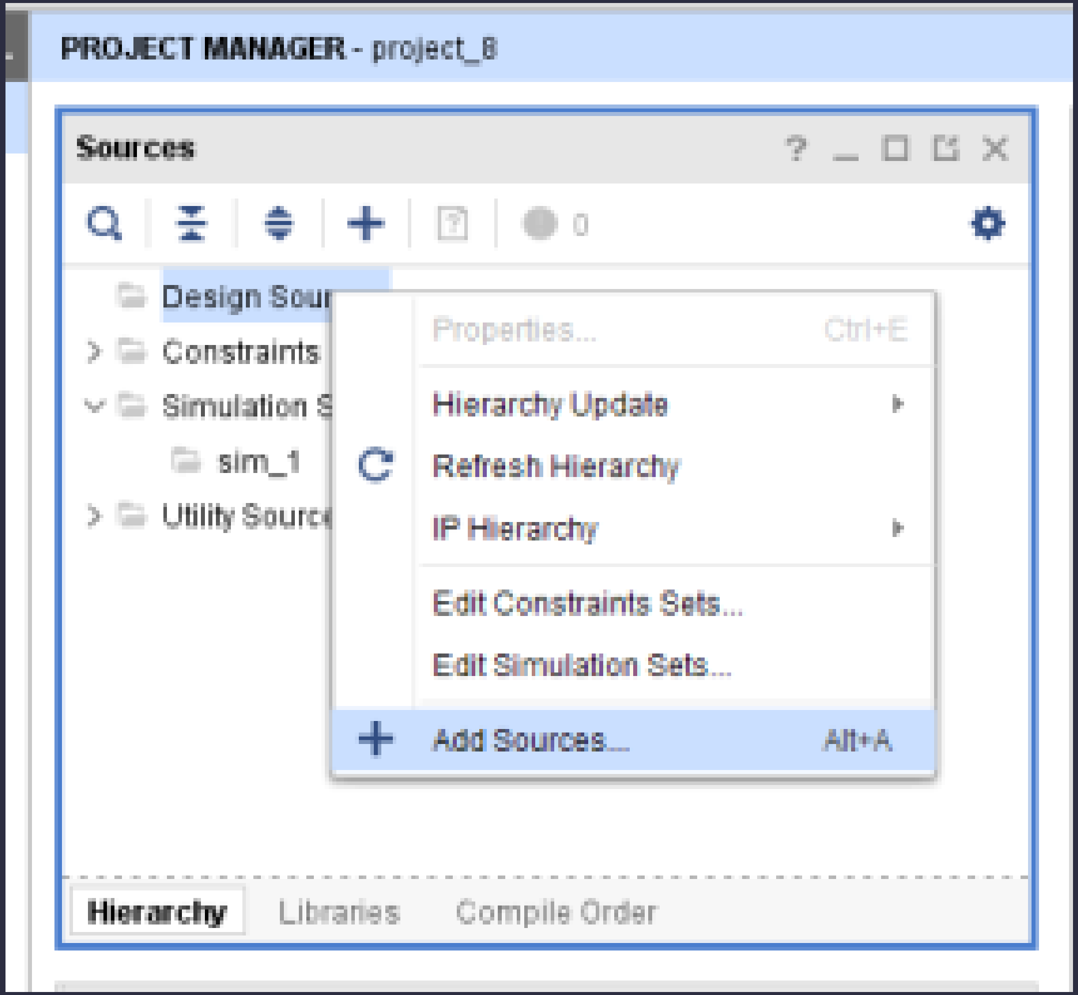


Vivado Design Tool



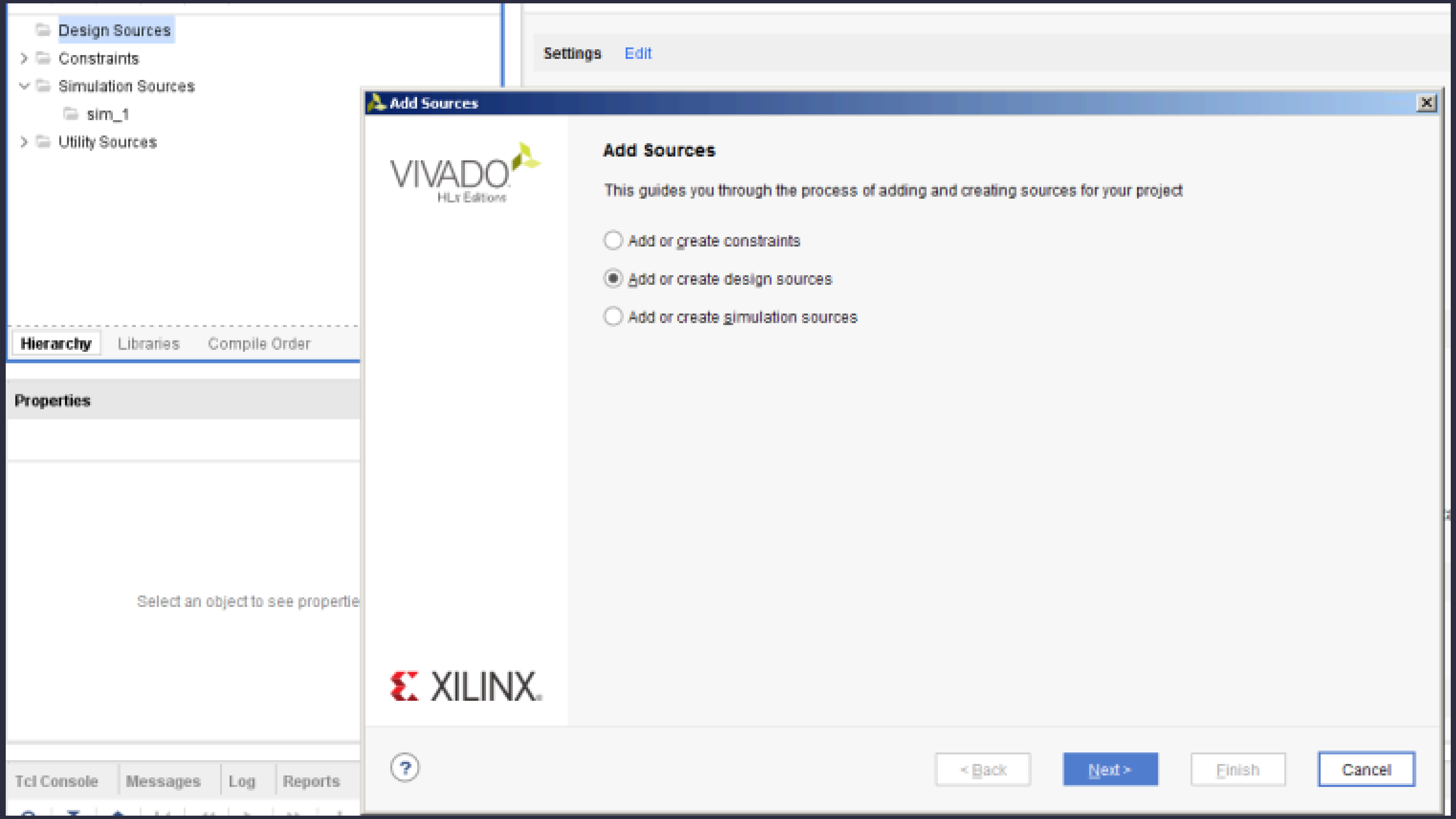
Vivado Design Tool

Adding a new design resource



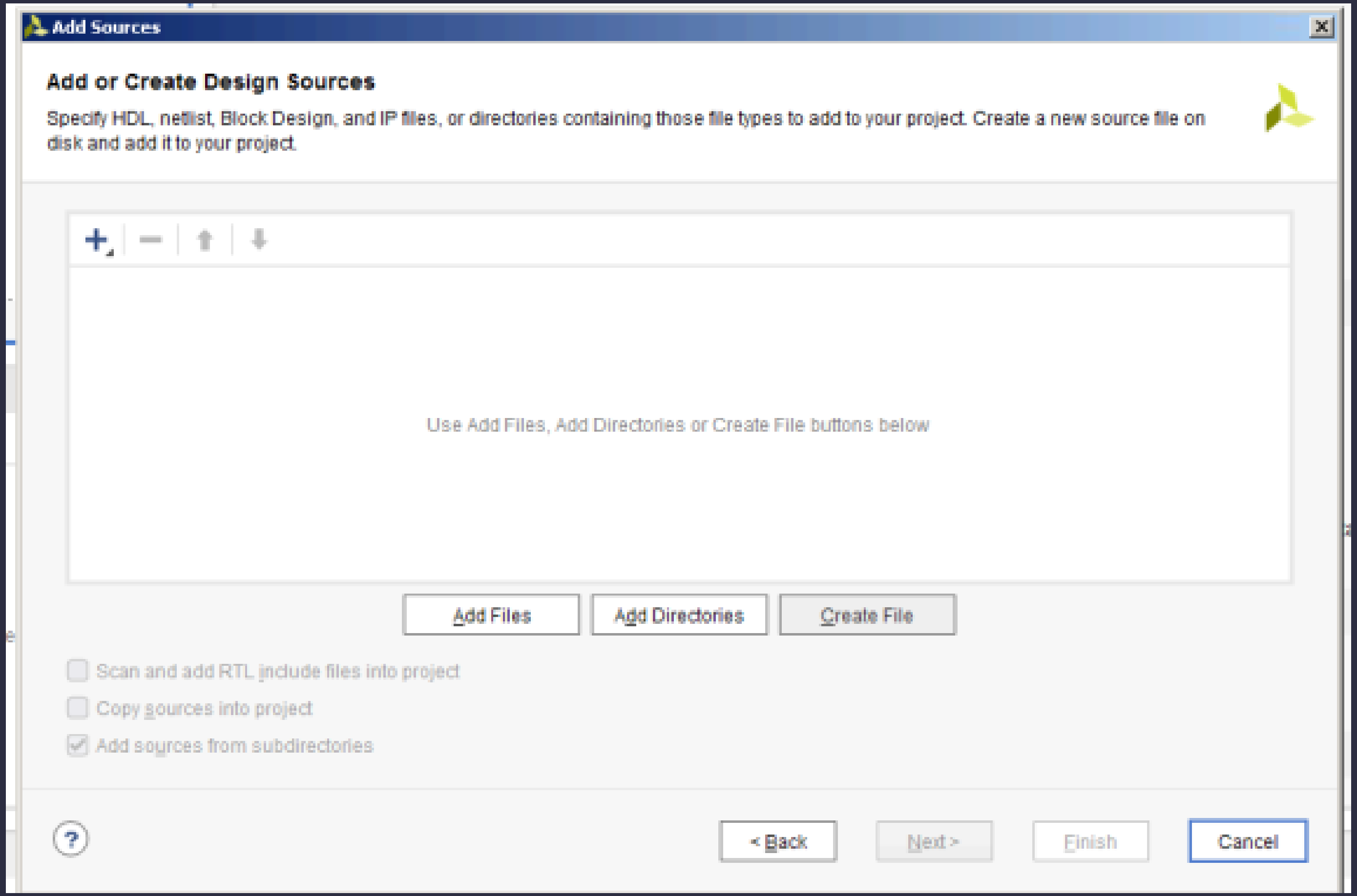
Verilog – Combinational Circuits

Vivado Design Tool



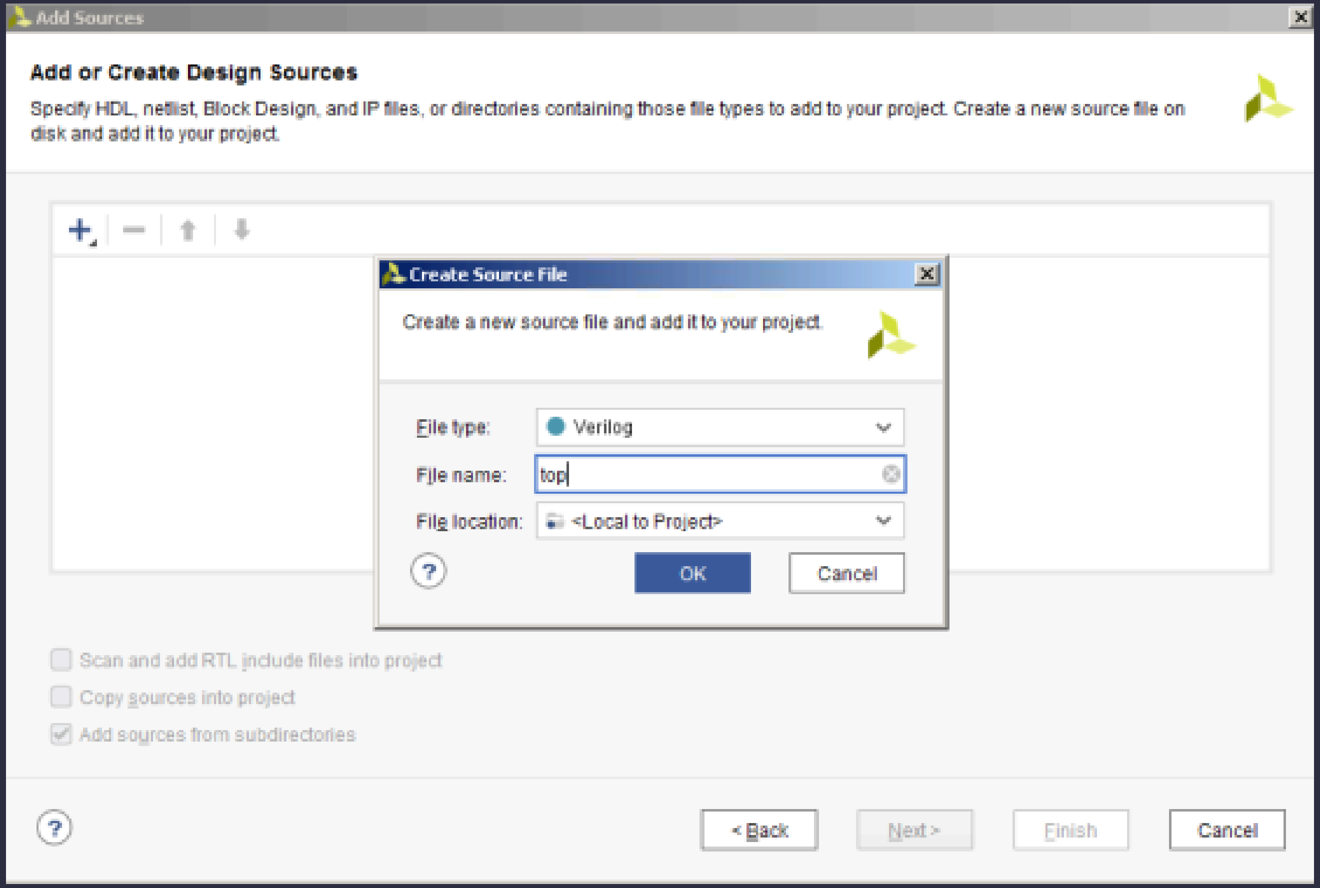
Verilog – Combinational Circuits

Vivado Design Tool

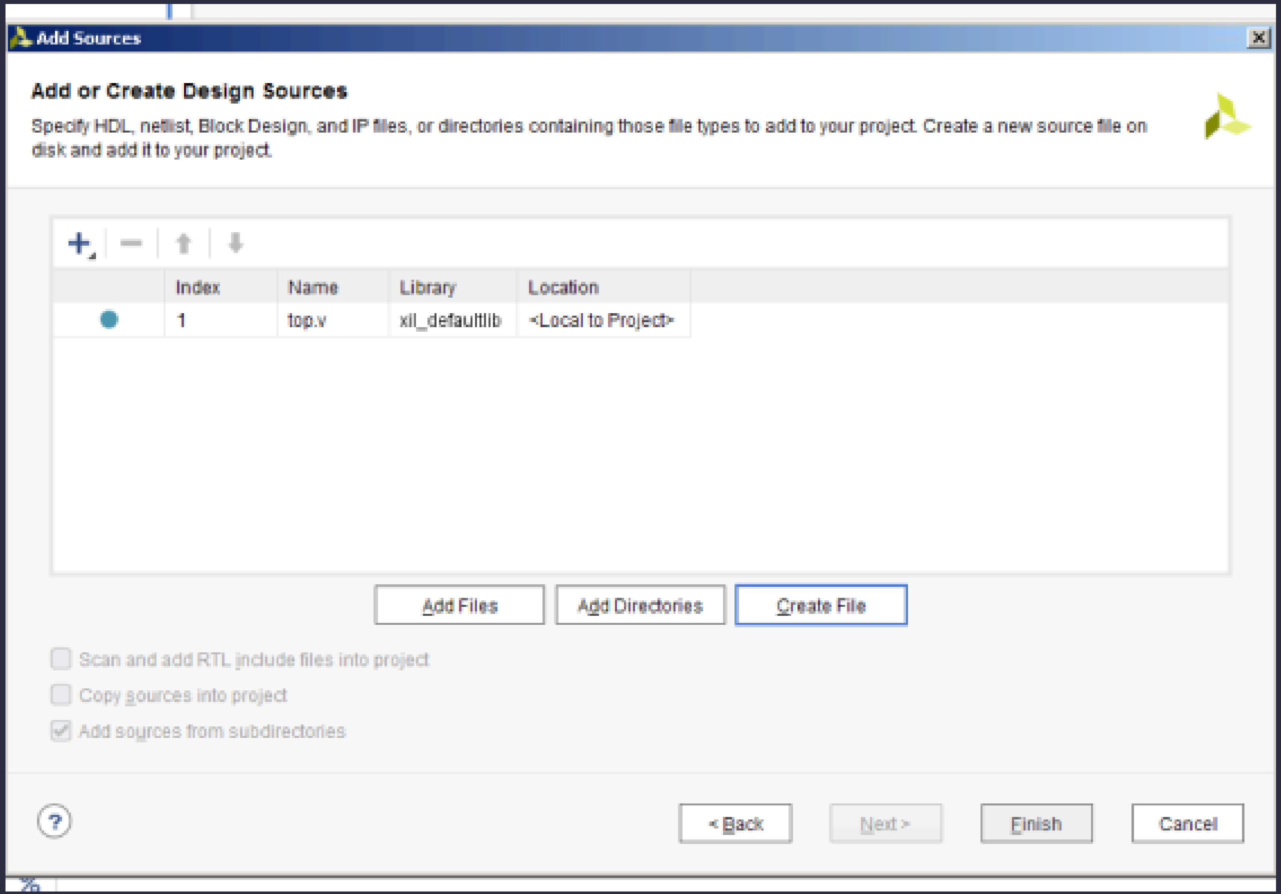


Verilog – Combinational Circuits

Vivado Design Tool



Vivado Design Tool

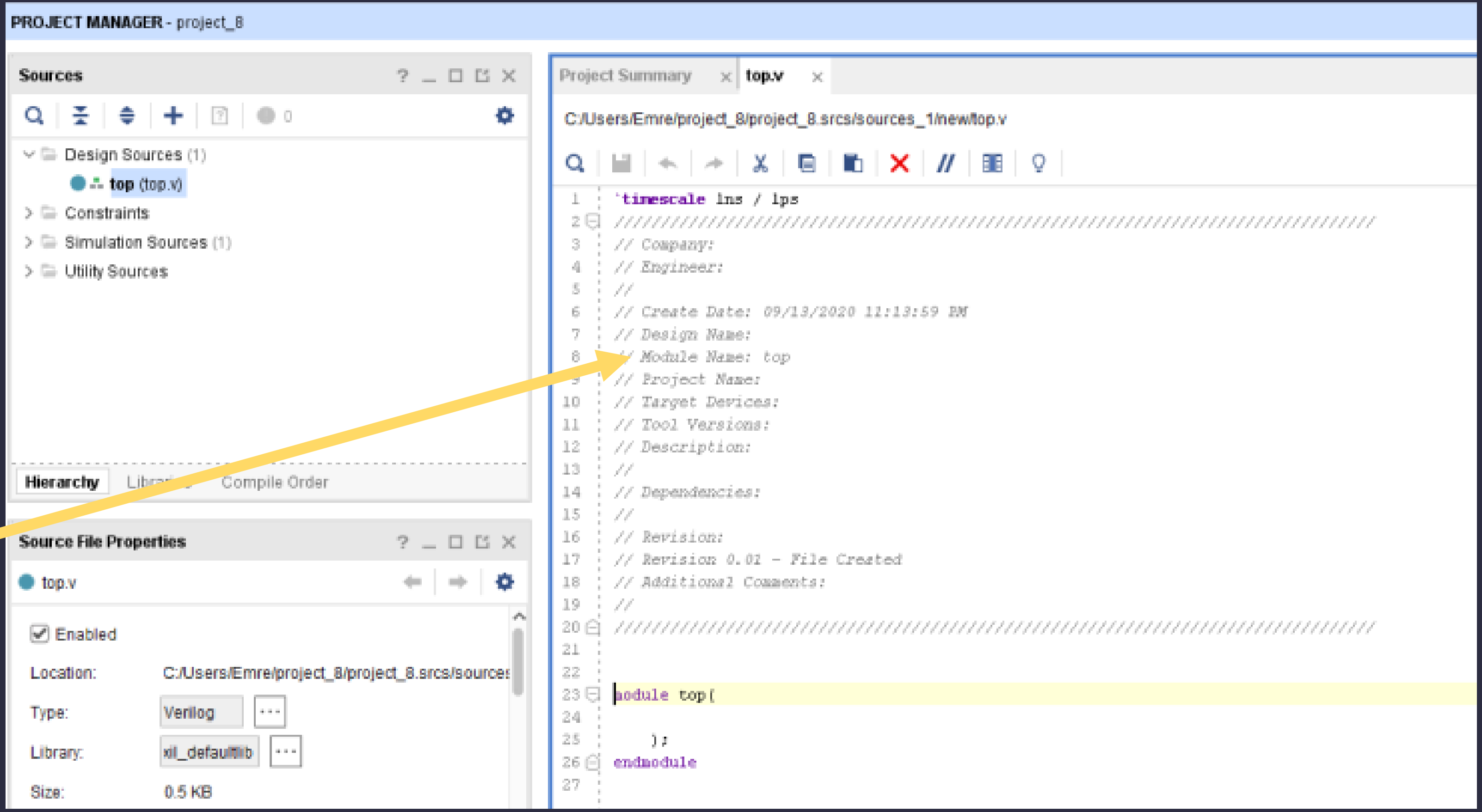


Verilog – Combinational Circuits

Vivado Design Tool

RTL Design

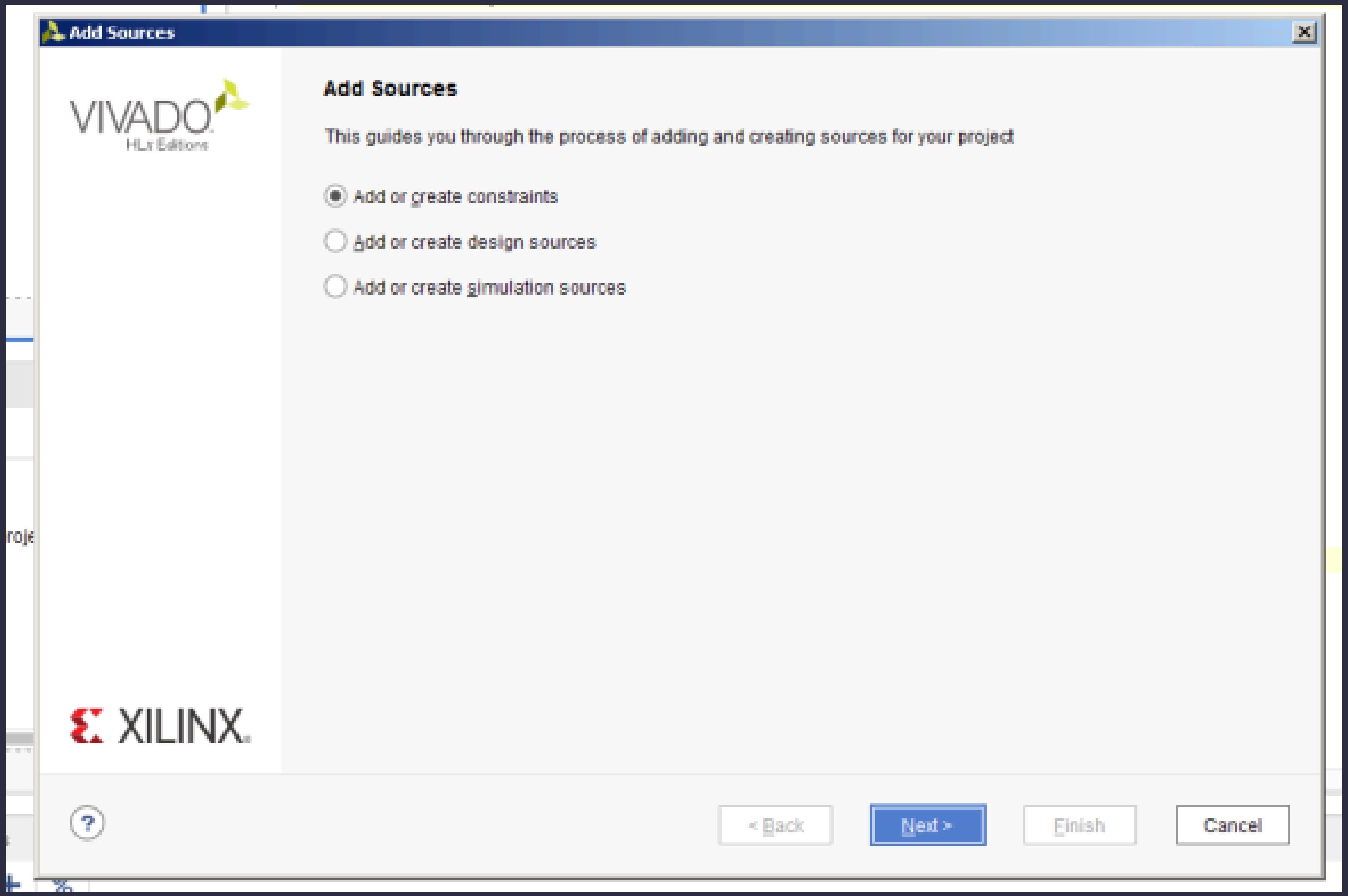
File To Do



Verilog – Combinational Circuits

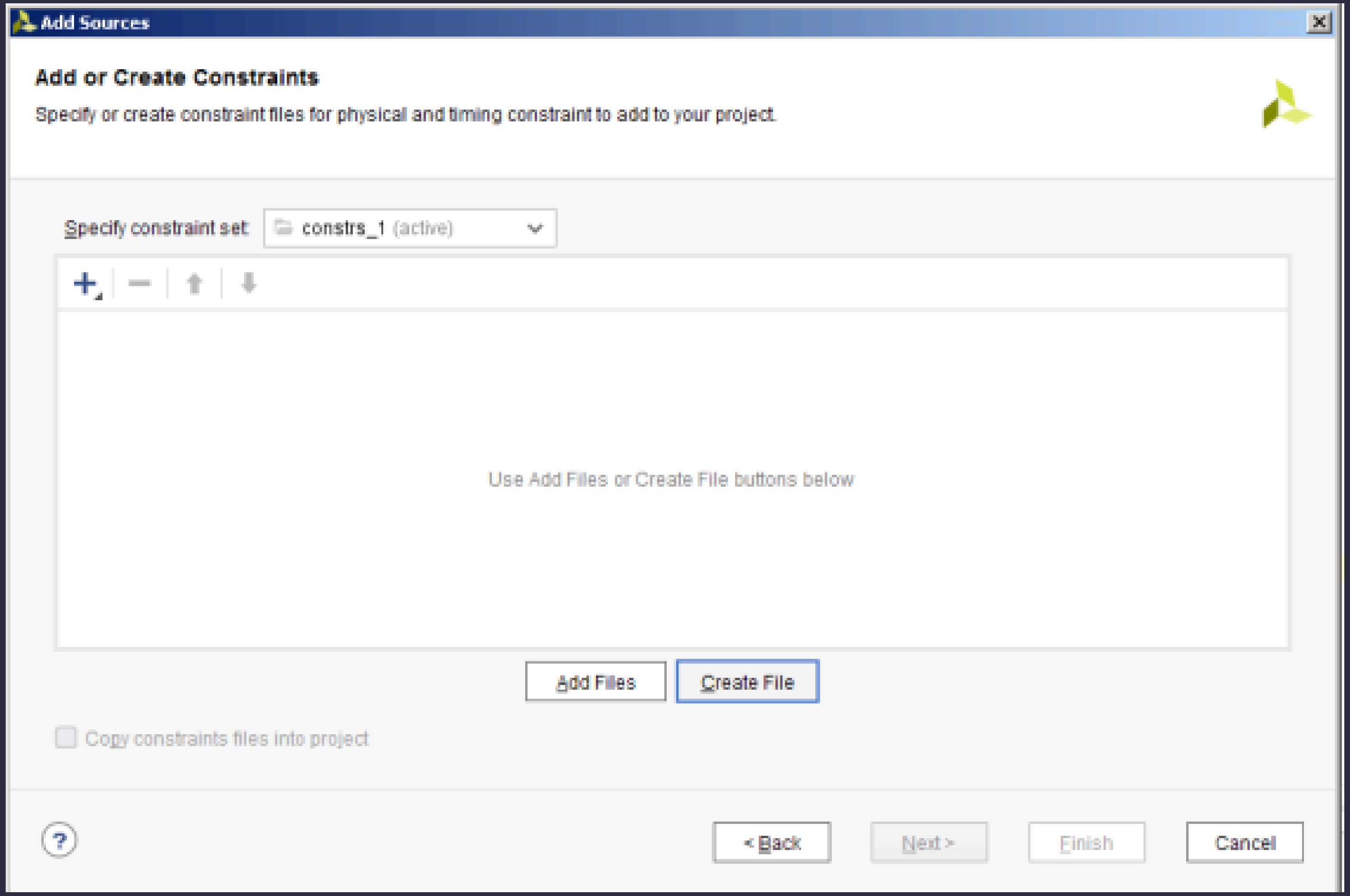
Vivado Design Tool

Adding a constraint



Vivado Design Tool

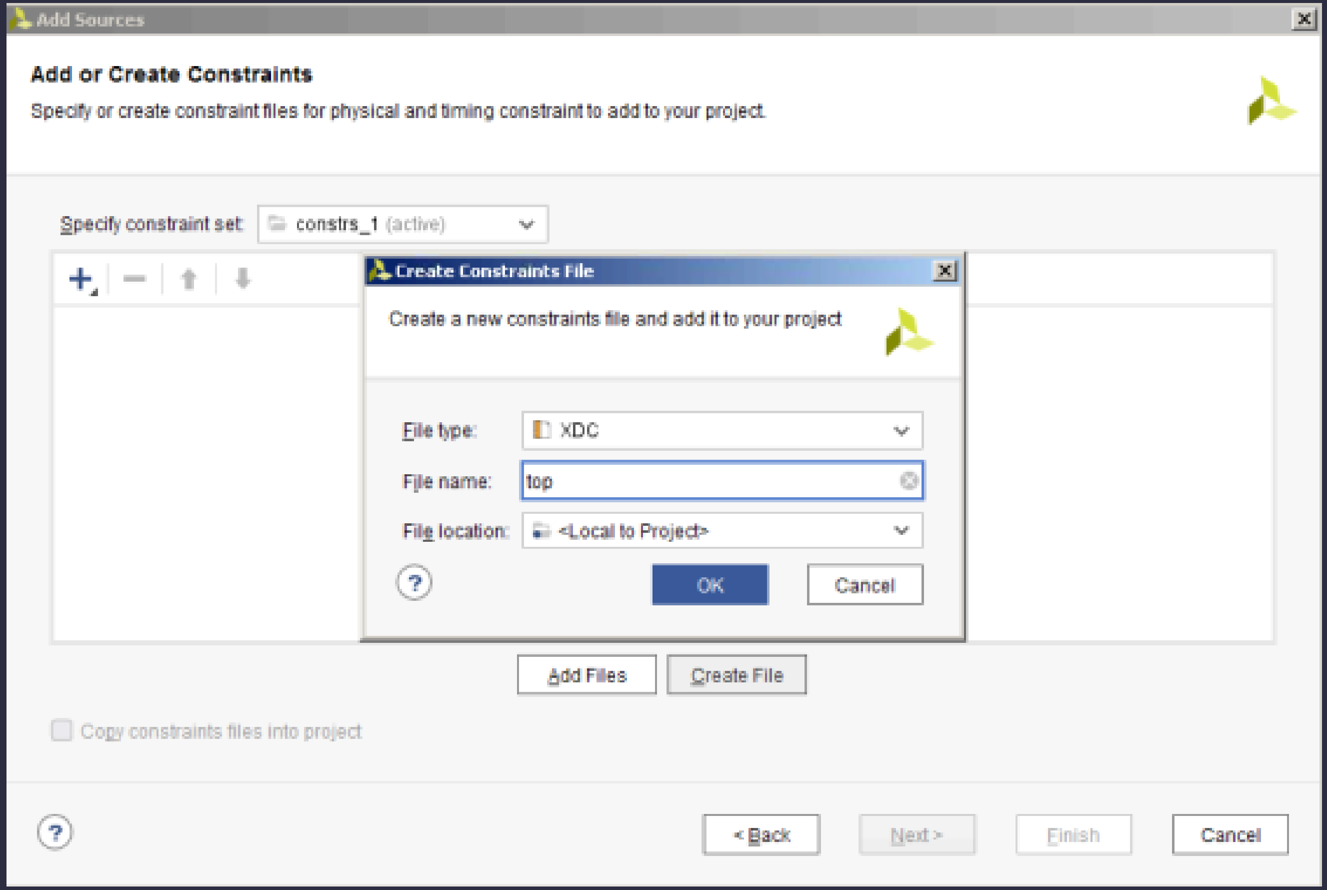
Adding a constraint



Verilog – Combinational Circuits

Vivado Design Tool

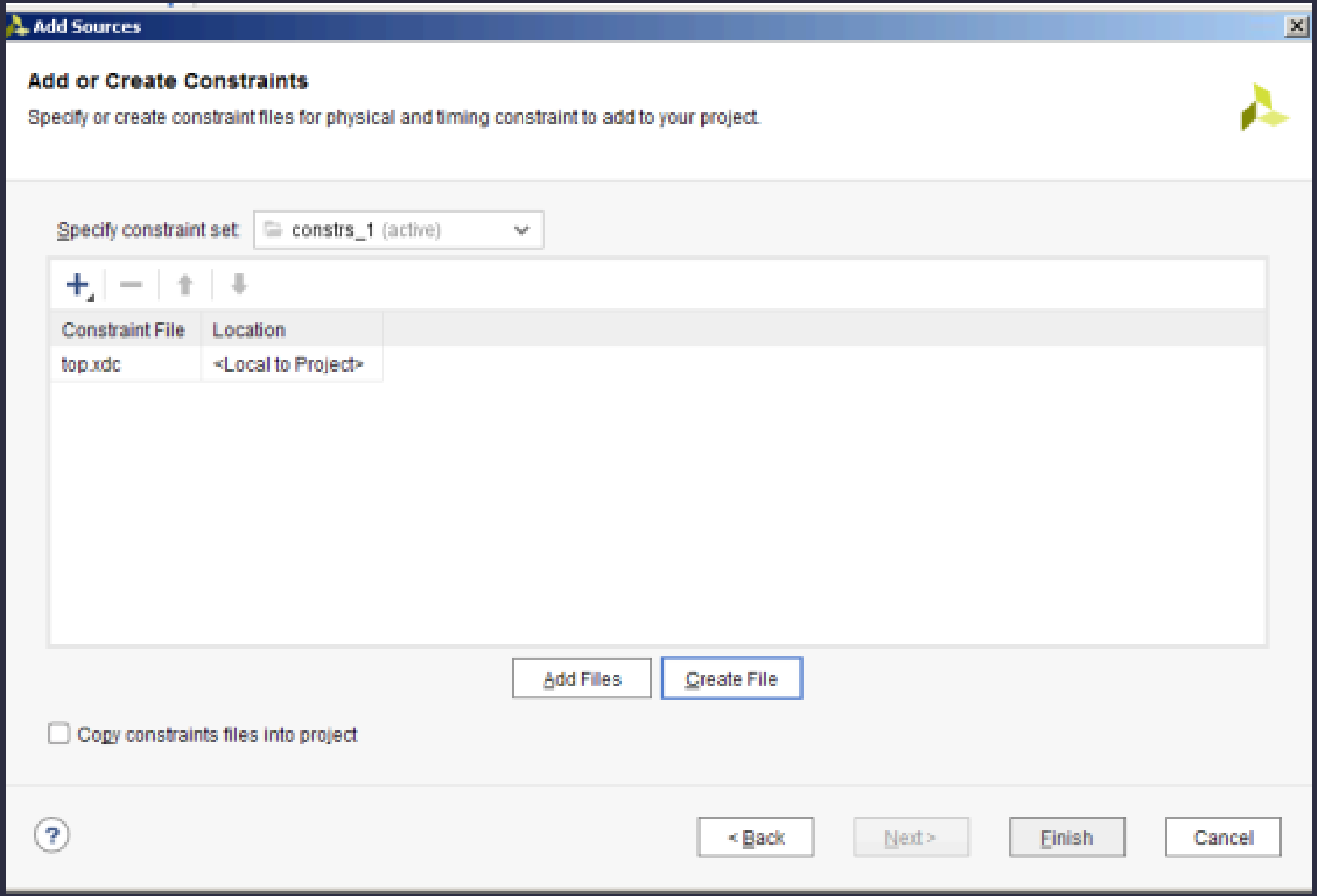
Adding a constraint



Verilog – Combinational Circuits

Vivado Design Tool

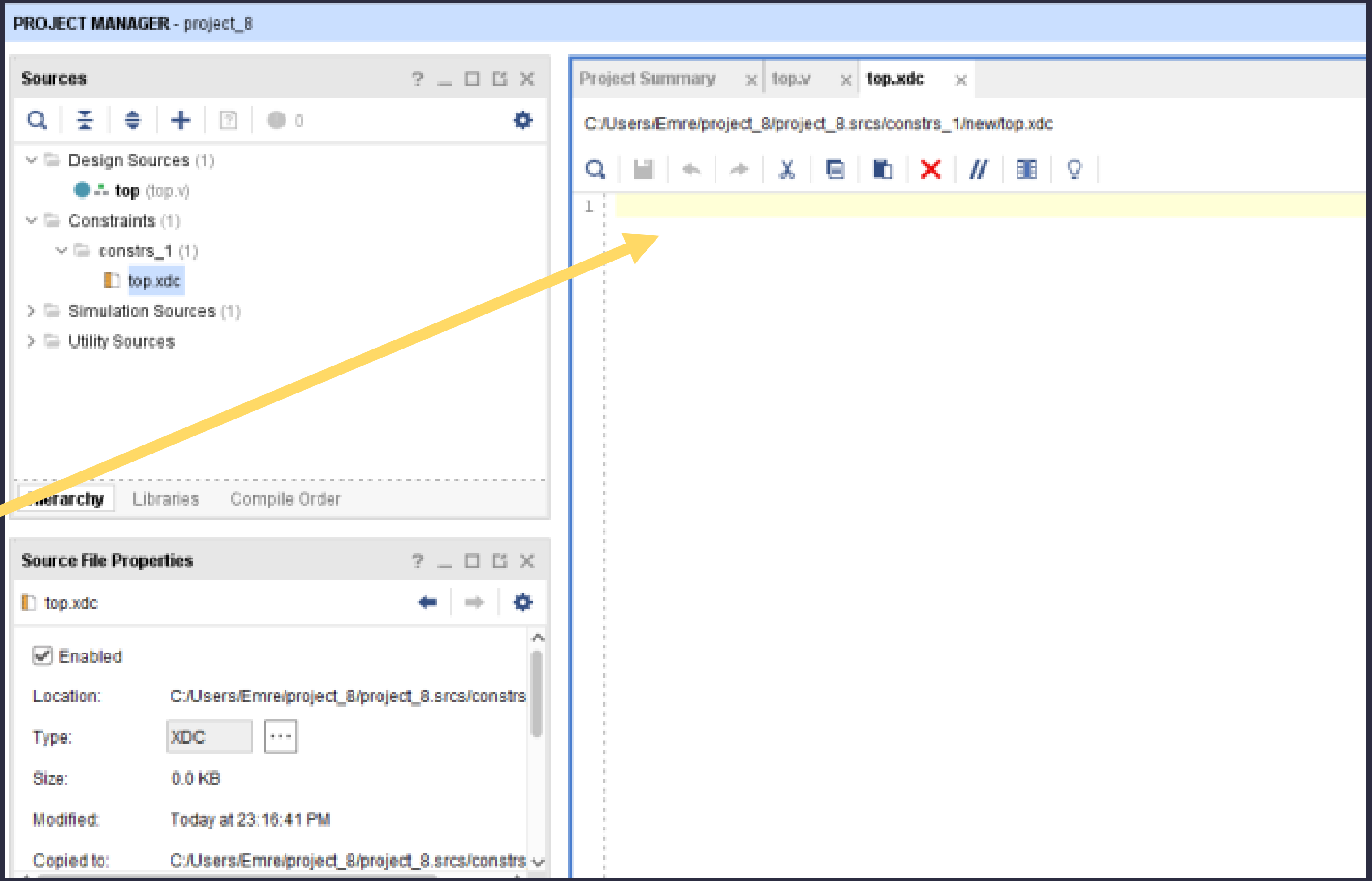
Adding a constraint



Verilog – Combinational Circuits

Vivado Design Tool

Adding a constraint

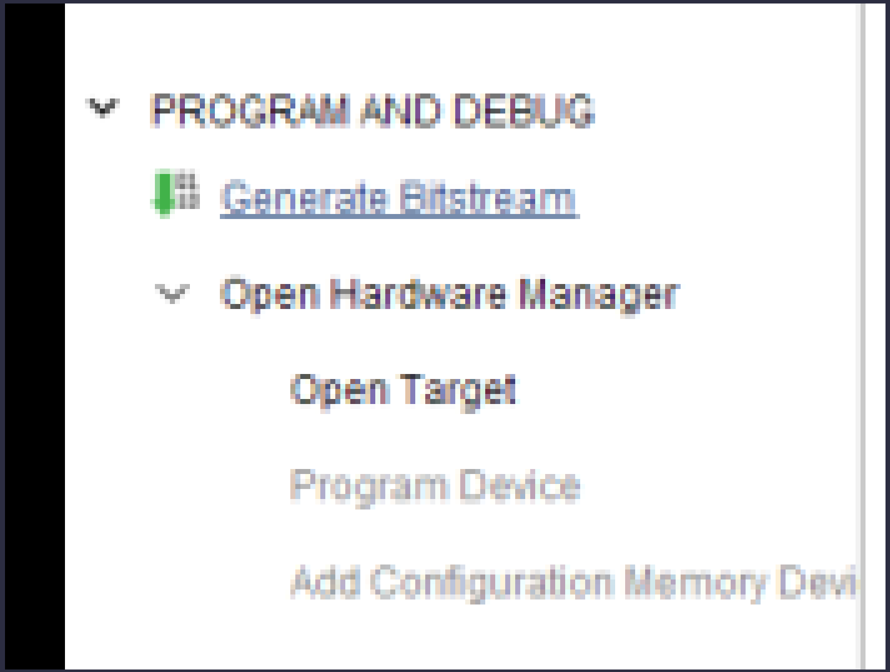


Write constraints

Verilog – Combinational Circuits

Vivado Design Tool

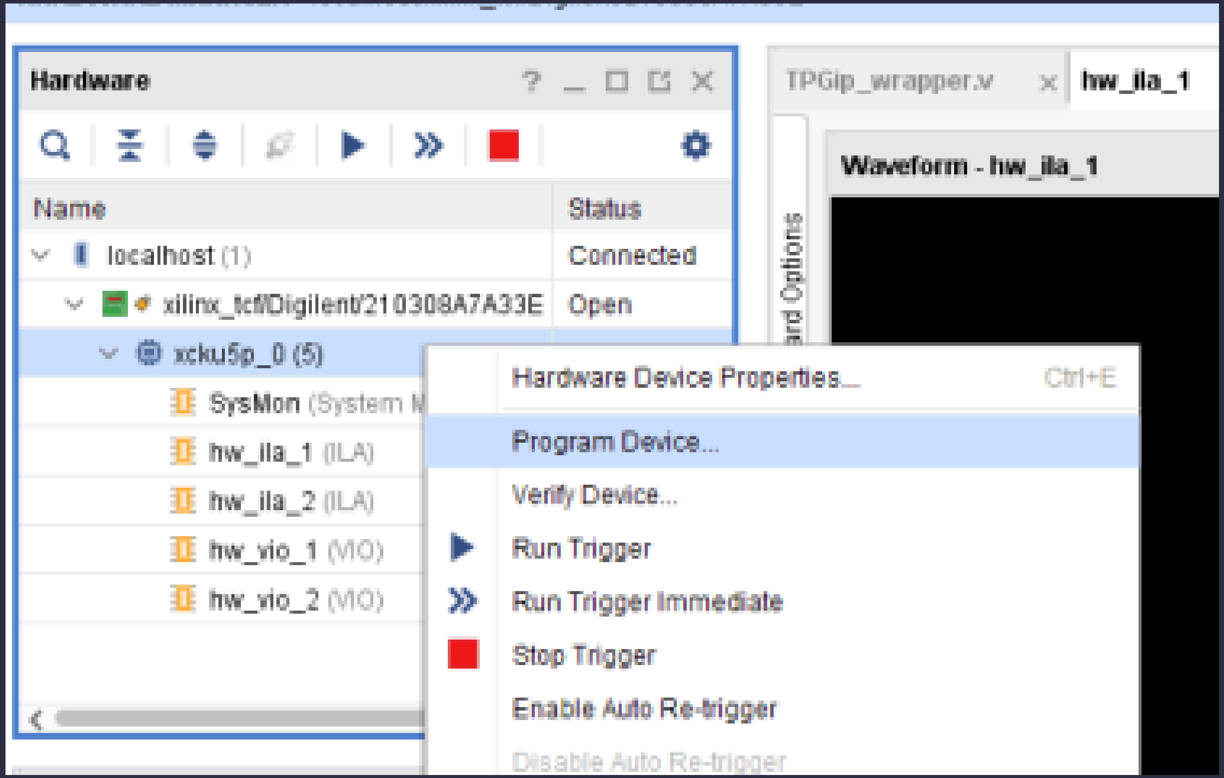
Bitstream generation



Verilog – Combinational Circuits

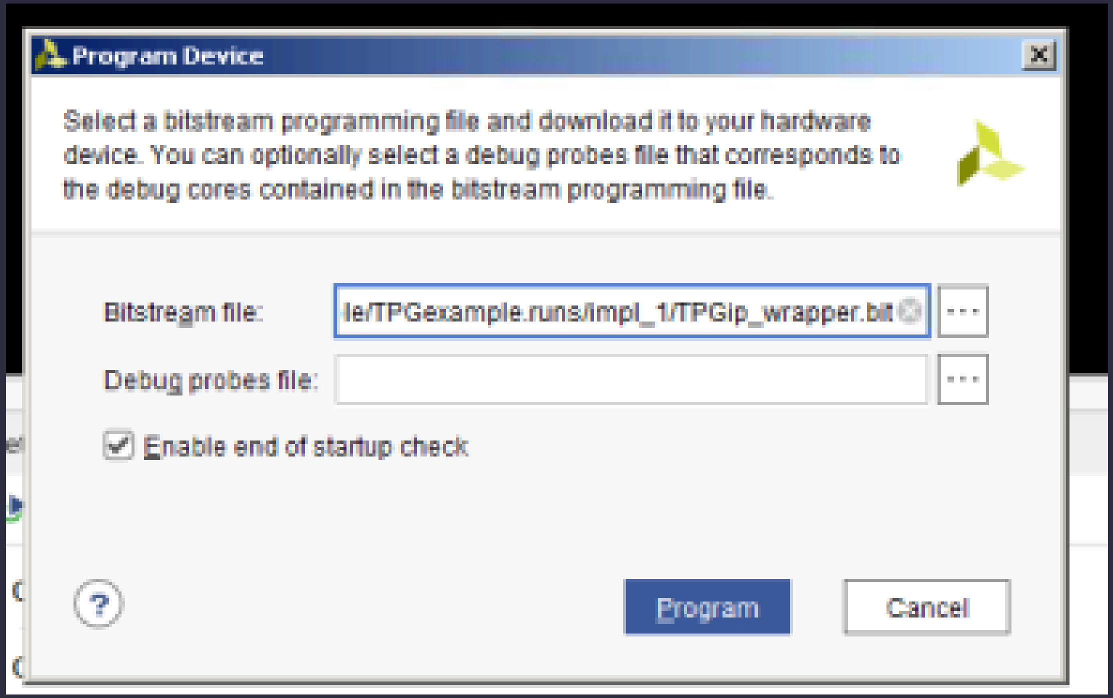
Vivado Design Tool

Bitstream generation



Vivado Design Tool

Bitstream generation

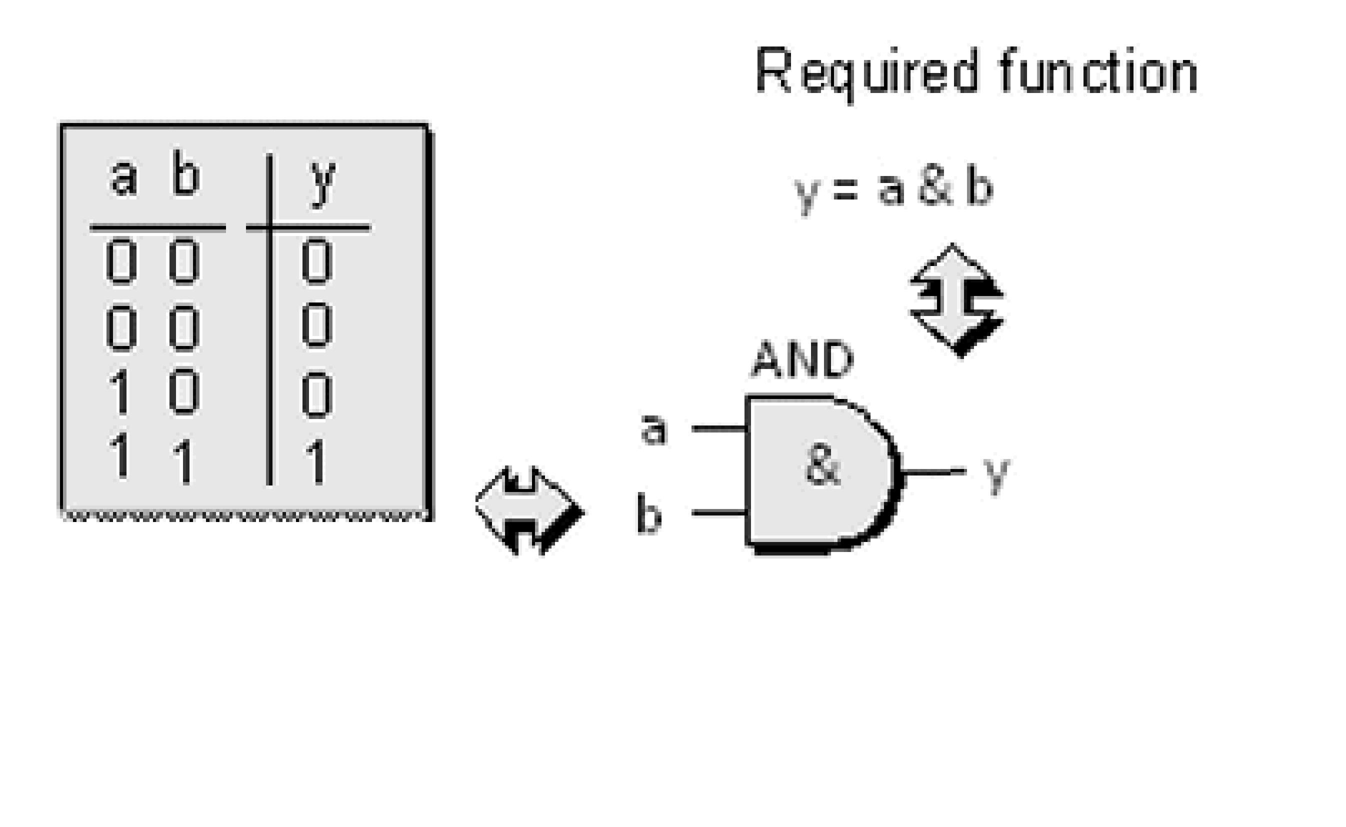


Verilog – Combinational Circuits

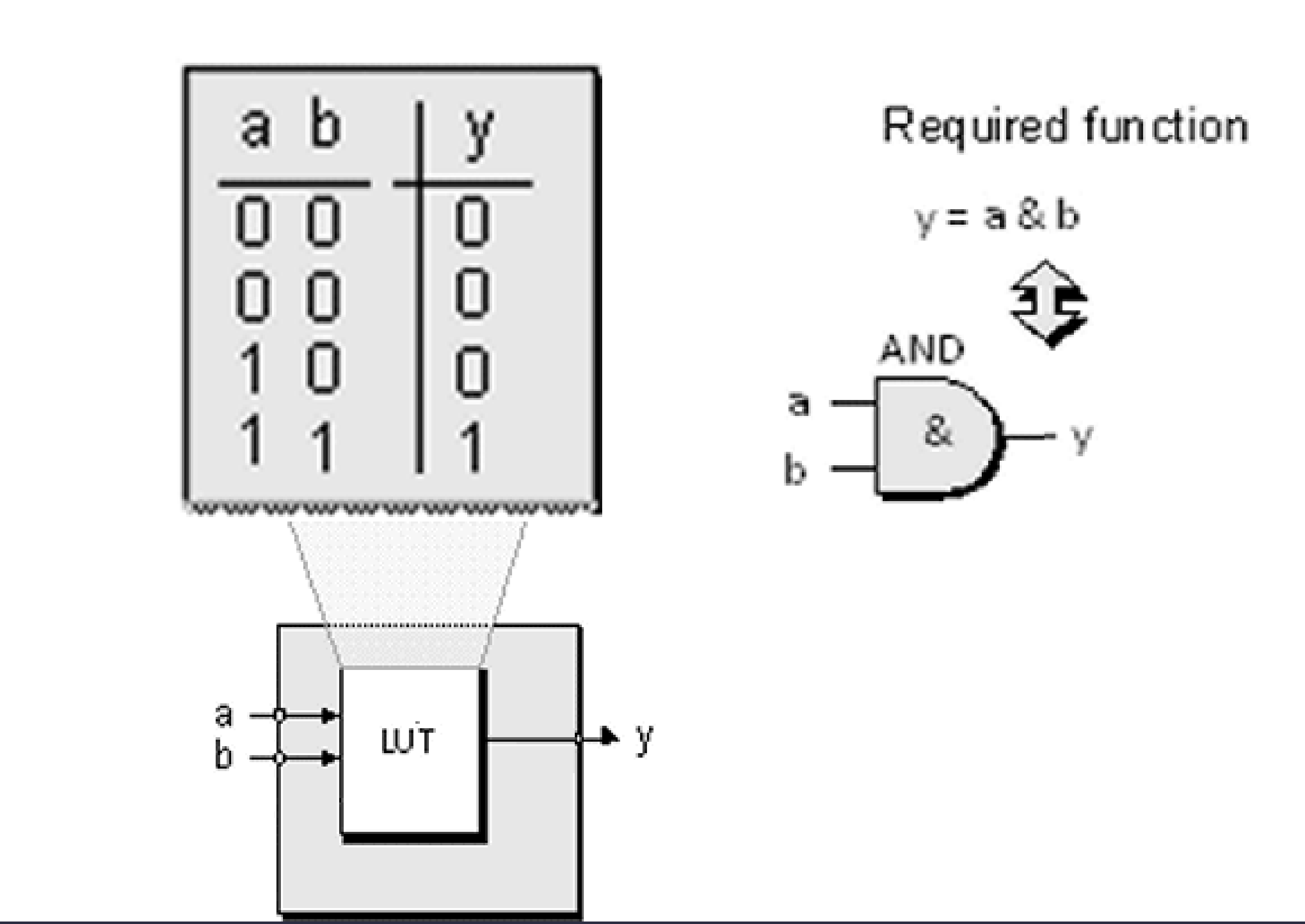
Most commonly using HDL (Hardware Description Language) Languages

- *Verilog*
- *System Verilog*
- *VHDL*

Verilog – Combinational Circuits



Verilog – Combinational Circuits



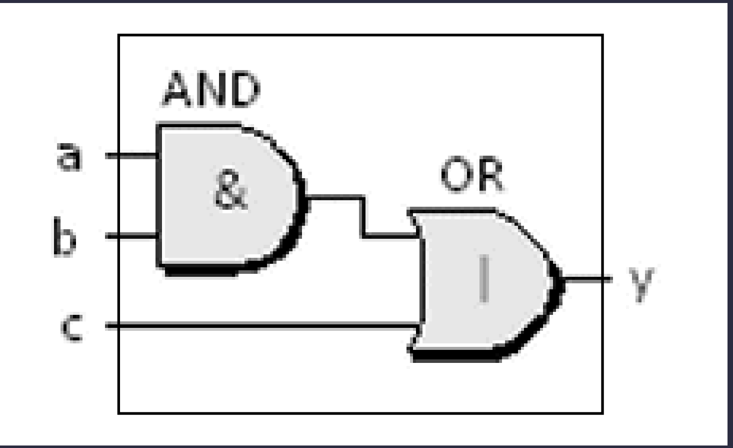
Verilog – Combinational Circuits

Vivado ,

- Verilog
- system Verilog
- VHDL

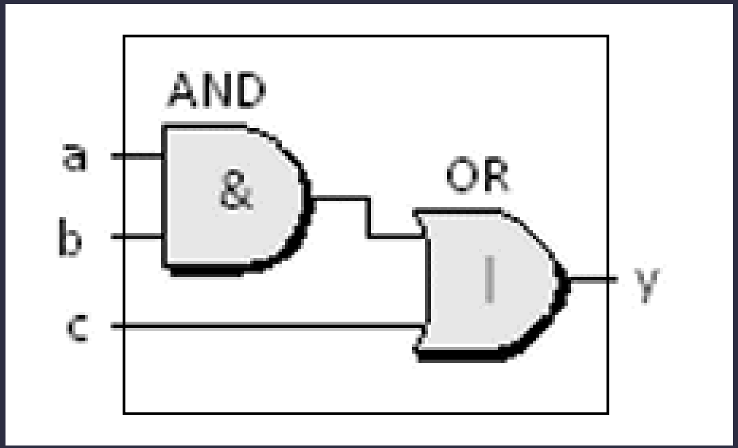
It supports languages. Within the scope of the course, designs will be made with Verilog language.

Verilog – Combinational Circuits



myModule

Verilog – Combinational Circuits



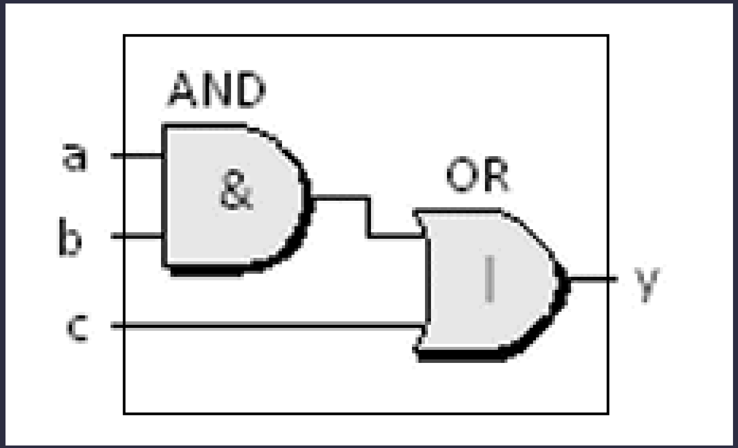
myModule

Verilog Design

```
module myModule(input  a, input b, input c,
output reg y);


endmodule
```

Verilog – Combinational Circuits



myModule

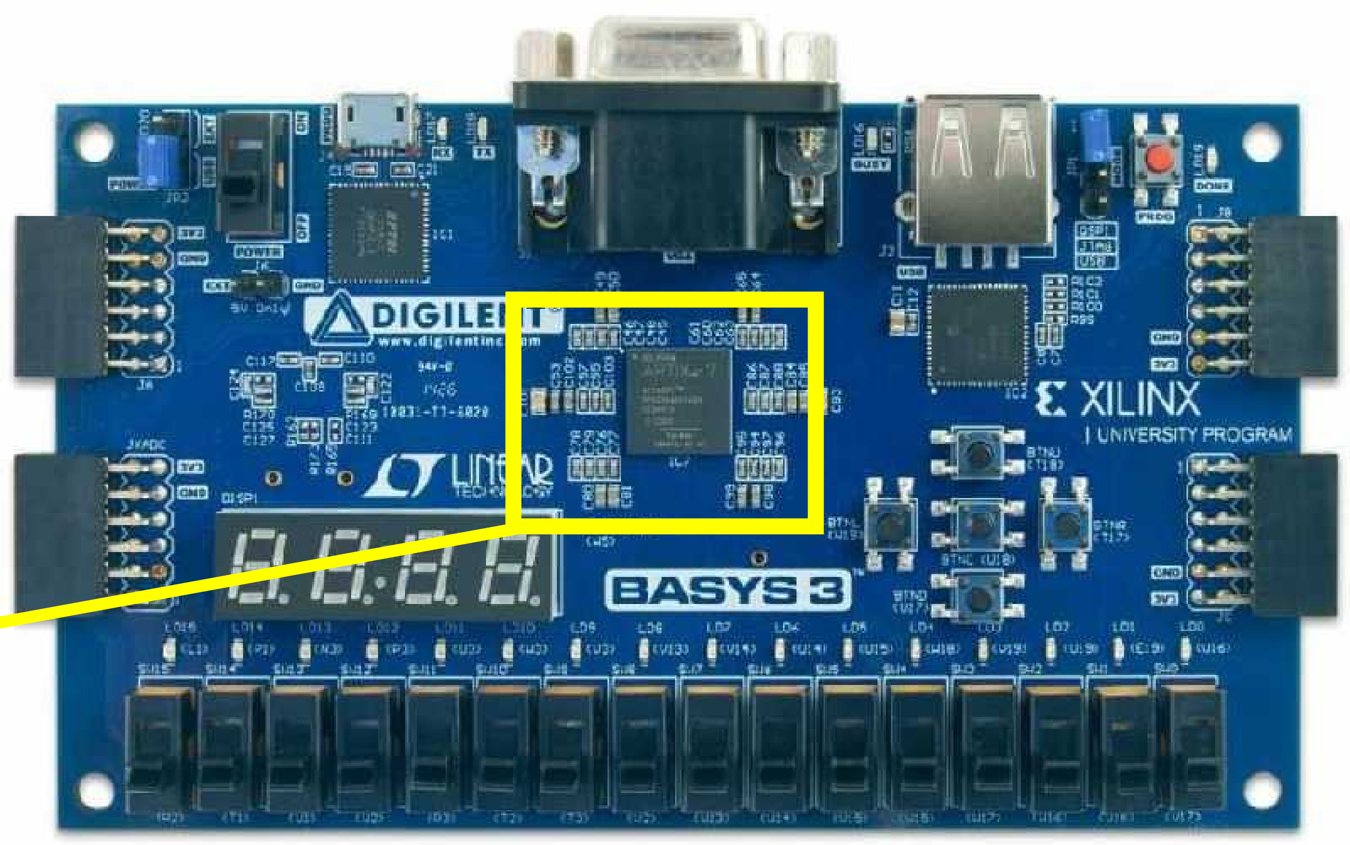
Verilog Design

```
module myModule(input  a, input b, input c, output reg y);

        reg tmp;
        always@(*) begin
                tmp = a & b;
                y = tmp | c;
        end

endmodule
```

Verilog – Combinational Circuits



FPGA

Verilog – Combinational Circuits

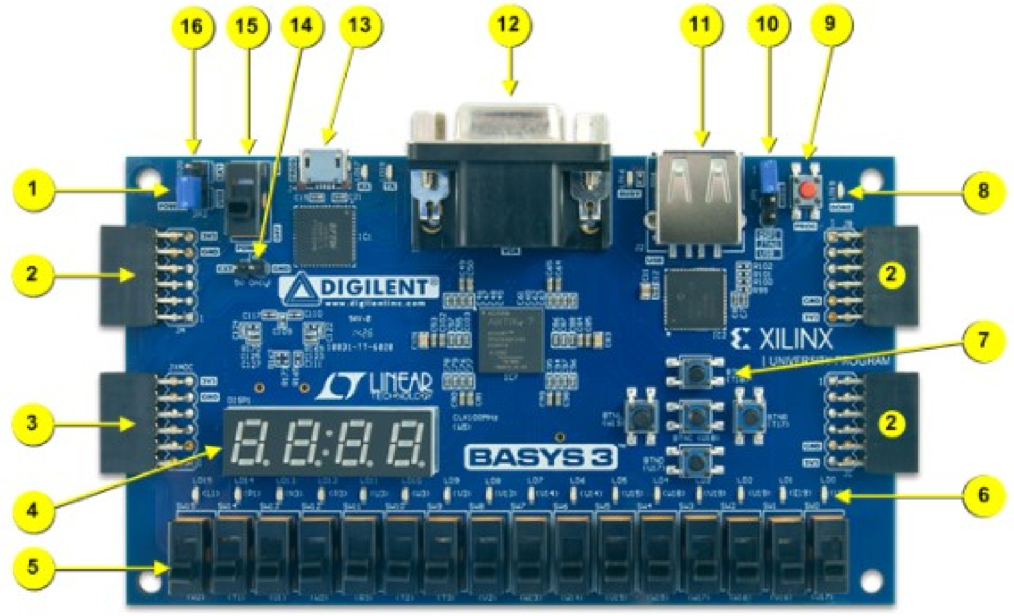
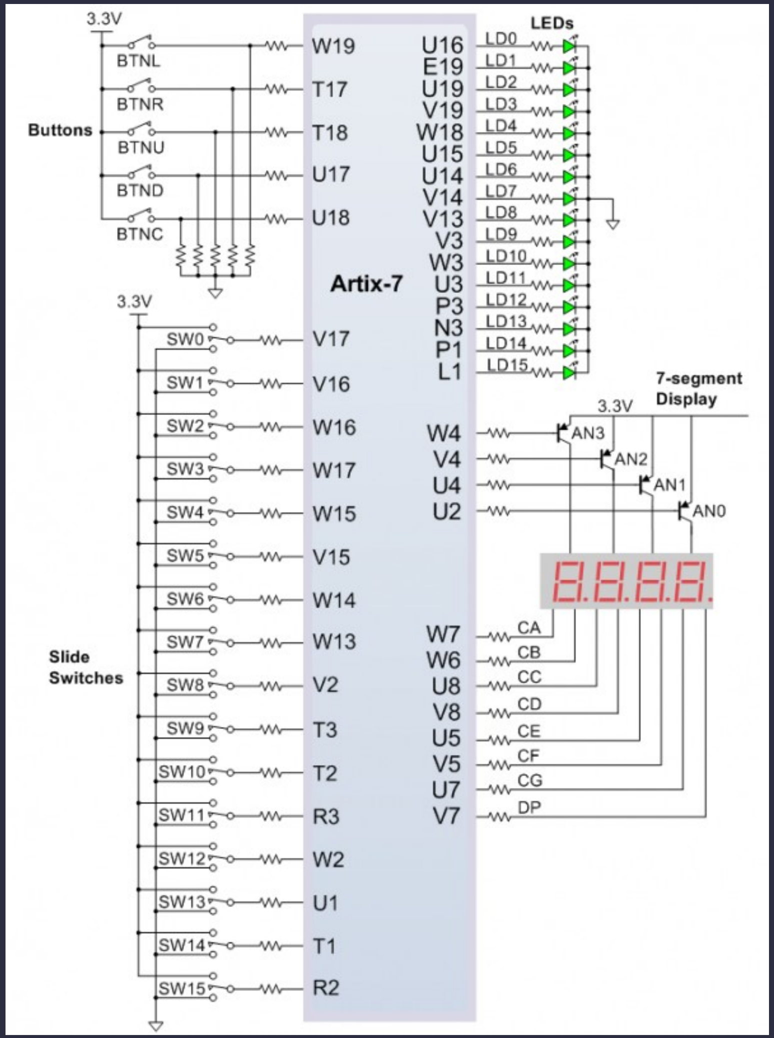


Figure 1. Basys3 board features

Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod connector(s)	10	Programming mode jumper
3	Analog signal Pmod connector (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

Verilog – Combinational Circuits



Constraint (XDC) File

http://levent.tc/files/courses/digital_design/labs/basys3.xdc