Rutgers University
School of Engineering

Fall 2022

332:231 – Digital Logic Design

Sophocles J. Orfanidis
ECE Department
orfanidi@rutgers.edu

Unit 1 – Introduction to DLD

How to Succeed in DLD

"I hear and I forget,
I see and I remember,
I do and I understand"

Confucius

"The purpose of computing is insight, not numbers"

Richard Hamming

The keys to success in DLD are the above two quotes from Confucius and Hamming, that is, practice by doing a lot of problems on your own, including their computer implementations, without looking at solutions.

Passive reading of the textbook (or lecture notes) usually conveys a false sense of understanding and does not result in a good grasp of the material.

DLD Course & Lab Organization

see course syllabus on Canvas Files for details on:

- course topics (additional web resources on Canvas)
- textbook options (additional references on Canvas)
- course prerequisites – DLD lab is a corequisite
- recitations (beginning in the week of September 12, 2022)
- course requirements
- exam dates (exams are administered online through Canvas Quizzes)
- course grading (exam weights & letter-grade thresholds)
- homework assignments (assigned but not graded)
- instructor & TA contact information
- academic integrity office
- office of disability services

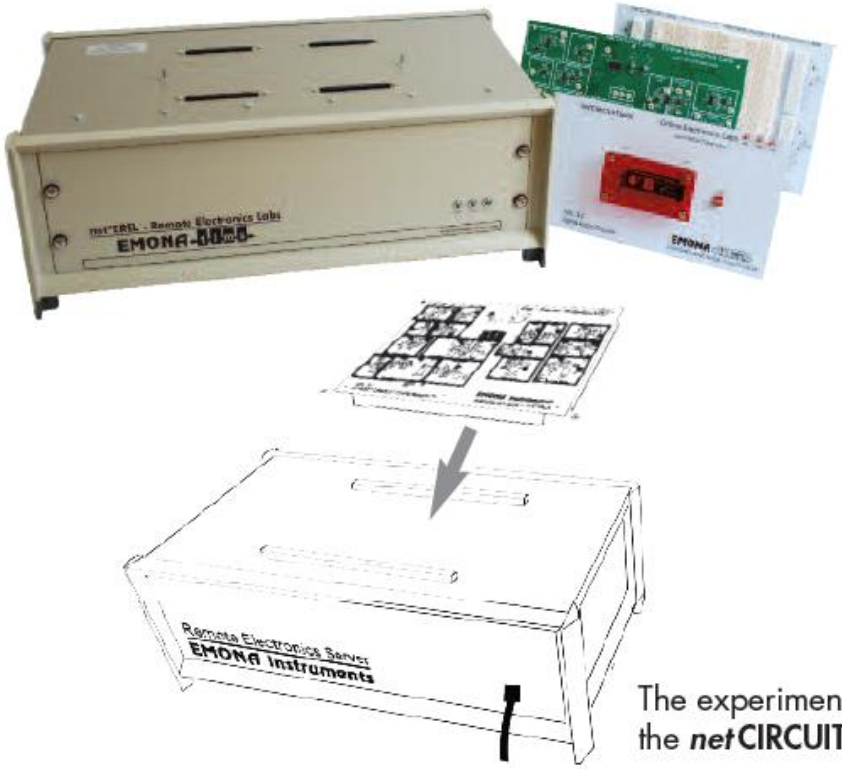
see DLD lab syllabus on Canvas Files for details on login info and:

- Emona FPGA board (manual & training videos on Canvas Files)
- lab sections (labs begin in the week of September 12, 2022)
- lab procedures (lab reports, declaration of authorship, screenshots)

DLD Lab

Emona netCIRCUITlabs FPGA board (residing in ECE-207)

netCIRCUITlabs CONTROL UNIT with MULTIPLE PLUG-IN BOARDS

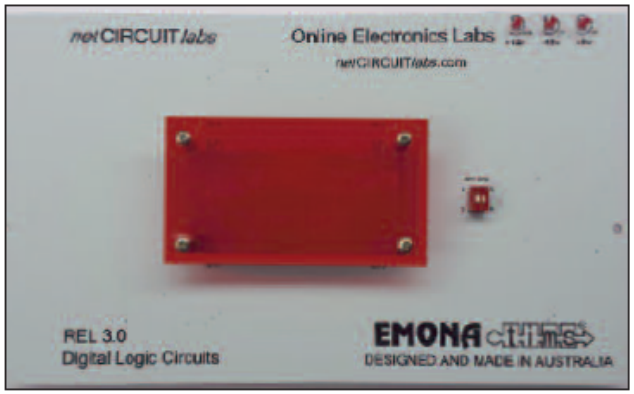


The experiments boards plugs into the ***net*CIRCUIT*labs*** Control Unit

- The netCIRCUITlabs Control Unit, located in your lab or office, and will accept any netCIRCUITlabs Lab Experiment board.
- Fast and easy implementation. No software to install and no setting up required.
- Secure access for professor to all ADMIN functions including student records and tracking.

DLD Lab

REL 3.0 DIGITAL LOGIC board - student wired experiments



All the logic functions and connections are implemented in an FPGA.

REL3.0 FUNCTIONALITY & EXPERIMENT CAPABILITIES

SIGNAL SOURCES:

- *HI/LO Logic Switches x 8*
- *8 bit Binary Counter*
- *4 bit Gray Counter*
- *4 bit Johnson Counter*

OVER 60 GATES & FLIP-FLOPS:

2, 3 & 4-input OR gates
X-OR gates
2, 3 & 4-input AND gates
Inverters
S/R, D & J/K Flip-Flops,
Inverters
Finite State Machines

STUDY:

- *Boolean logic and algebra*
- *Combinatorial circuits*
- *Truth tables*
- *Karnaugh Maps*
- *Quine-McCluskey method*
- *Designing Synch & Asynch sequential circuits*
- *Flip flops*
- *State diagrams*
- *Design of FSM*
- *Registers, Counters, Multiplexers, Encoders etc*
- *Introduction to HDL (Verilog)*

student login: **http://ece-emona1.engr.rutgers.edu/**

Course Topics

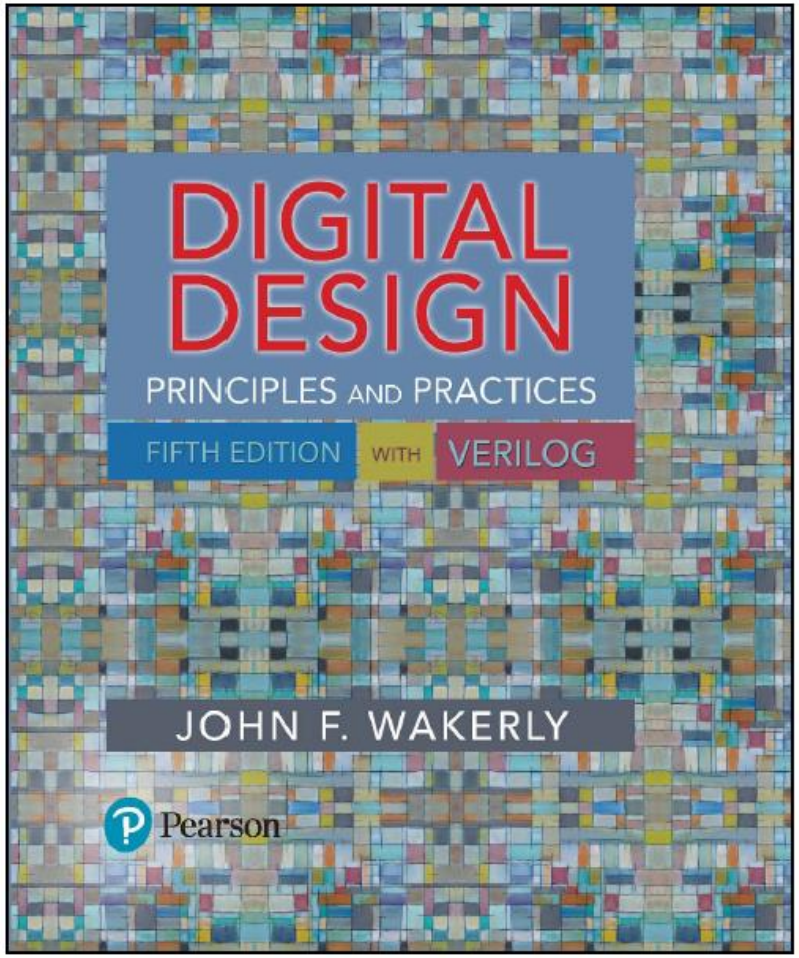
units →

1. Introduction to DLD, Verilog HDL, MATLAB/Simulink
2. Number systems
3. Analysis and synthesis of combinational circuits
4. Decoders/encoders, multiplexers/demultiplexers
5. Arithmetic systems, comparators, adders, multipliers
6. Sequential circuits, latches, flip-flops
7. Registers, shift registers, counters, LFSRs
8. Finite state machines, analysis and synthesis

Text: J. F. Wakerly, *Digital Design Principles and Practices*, 5/e, Pearson, 2018
additional references on Canvas Files > References

Digital Design: Principles and Practices

Fifth Edition With Verilog



see syllabus for ordering options

4th edition OK, but the material has been rearranged in the 5th edition, and you would need to figure out the correct mapping of the sections and problems between the two editions

Pearson

Copyright © 2018, 2006, 2000 Pearson Education, Inc. All Rights Reserved

References

main text:

J. F. Wakerly, *Digital Design Principles and Practices*, 5/e, Pearson, 2018.

supplementary texts:

S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with Verilog Design*, 3/e, McGraw-Hill, 2014.

D. M. Harris and S. L. Harris, *Digital Design and Computer Architecture*, 2/e, Elsevier, 2013.

M. Mano, C. R. Kime, and T. Martin, *Logic and Computer Design Fundamentals*, 5/e, Pearson, 2016.

E. O. Hwang, *Digital Logic and Microprocessor Design with Interfacing*, 2/e, Cengage, 2018.

Additional References

A. F. Kana, *Digital Logic Design*, [on Canvas].

B. J. Mealy & J. T. Mealy, *Digital McLogic Design*, 2012 [on Canvas].

E. Peasley, *An Introduction to Using Simulink*, 2018 [on Canvas].

H. Moore, *Ch.16 – Simulink – A Brief Introduction*, from *MATLAB for Engineers*, 3/e, Pearson, 2011.

S. A. Edwards, *Verilog Language*, 2001 [on Canvas].

B. Izadi, *Verilog Tutorial*, 2016 [on Canvas].

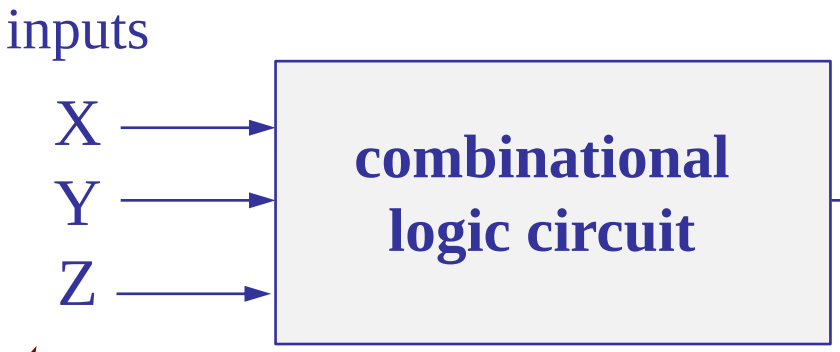
C. Maxfield, *Bebop to the Boolean Boogie*, 2/e, Newnes, 2009.

Minecraft-Logic-Gates.pdf [on Canvas], see also, Redstone Logic Gates

Unit-1 Contents: (current reading: Wakerly Chapter 1)

1. Functional and truth-table representations of logic circuits
2. Basic logic gates, AND, OR, NOT, NAND, NOR
3. Analysis & synthesis of combinational circuits – design example
4. Integrated circuits: SSI, MSI, VLSI, CPLD, FPGA, 74x family
5. CMOS realizations of logic gates
6. Design levels: (a) functional definition level
   (b) transistor level
   (c) truth-table level, FPGA look-up tables
   (d) logic-gate level
   (e) Verilog HDL, structural or behavioral models
   (f) MATLAB and Simulink implementations
7. Design example – multiplexer function
8. Using Simulink for logic circuits
9. Moore's law

Example - representation of a 3-input / 1-output logic circuit



Boolean variable

Boolean variables

0 = ‘false’

1 = ‘true’

logic function

F = f(X,Y,Z)

classes of logic circuits

combinational circuits

sequential circuits

implemented with flip-flops, to be discussed later

note: with *n* input Boolean variables, there are 2^n possible input *n*-tuplets, and thus, 2^n possible Boolean outputs F

next up, elementary building blocks of logic circuits: AND, OR, NOT gates →

Basic Logic Gates: AND, OR, NOT
elementary building blocks of logic circuits

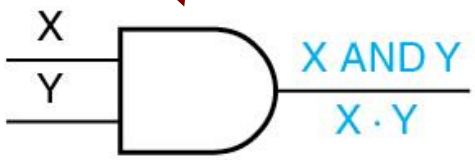
standardized IEEE symbols

Boolean variables

0 = 'false'
1 = 'true'

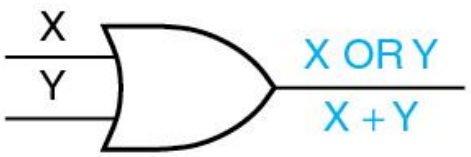
'bubble'

(a)



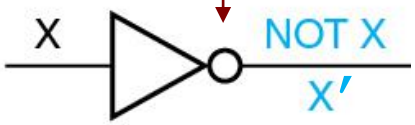
X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

(b)



X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

(c)



X	NOT X
0	1
1	0

AND, OR, NOT operations in MATLAB and Verilog notation:

& **|** **~** e.g., X & Y, X | Y, ~X

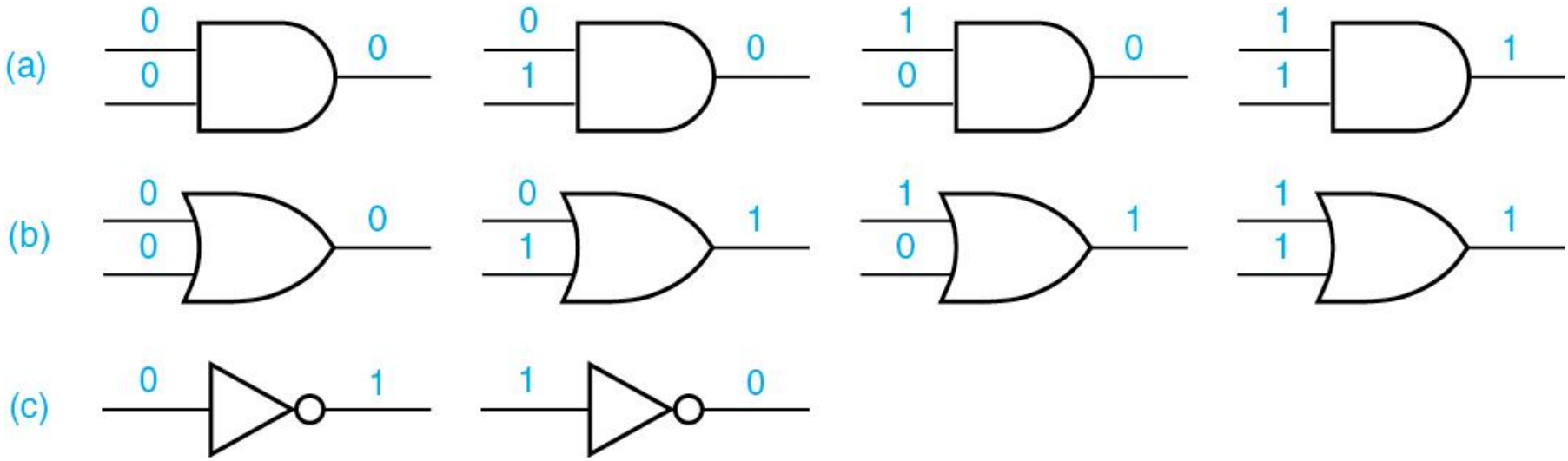
(X and Y) (X or Y) (not X)

Logic gates with all possible input values and outputs

(a) AND

(b) OR

(c) NOT or Inverter



Inverting gates

(a) NAND

(b) NOR

are discussed next

Inverting Gates
(a) NAND (b) NOR

bubble transformations

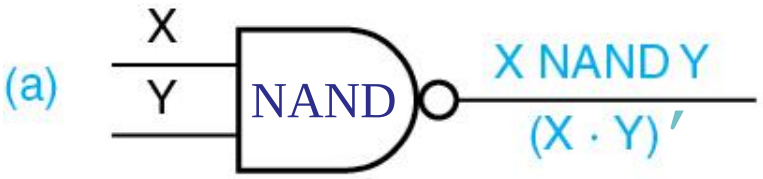
$X' + Y' = (X.Y)'$

NAND equivalent

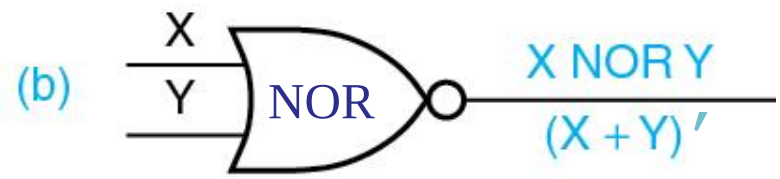
equivalent forms based on De Morgan's theorem – see unit-3

$X' . Y' = (X+Y)'$

NOR equivalent



X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0



X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0

Truth table, analysis & synthesis
combinational circuit example
Wakerly - Table 1.2

Objectives:

(a) start with a truth-table specification

(b) construct the logic function F=f(X,Y,Z)

(c) construct gate-level realizations

(d) implement them with MATLAB

(e) realize them in Simulink, and on

(f) the Emona FPGA board

these steps are reversible – one could start with any of them and derive the rest

	X	Y	Z	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

logic function

F = f(X,Y,Z)

with 3 input variables, there are 2^3=8 possible input triplets, and 2^3 outputs F

Example - constructing a logic function from a truth table (Table 1.2)

logic function
F = f(X,Y,Z)

X	Y	Z	F	
0	0	0	0	
0	0	1	1	→ X′.Y′.Z
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	1	→ X.Y.Z′
1	1	1	1	→ X.Y.Z

main technique:
focus on the 1's at the output

→ X.Y

there are at least 7 other mathematically equivalent formulas for realizing this logic function (see p.24)

→ F = (X.Y) + (X′.Y′.Z)

Example - constructing a logic function from a truth table (Table 1.2)

logic function
F = f(X,Y,Z)

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

X′.Y′.Z	X.Y.Z′	X.Y.Z	X.Y
0	0	0	0
1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	1	0	1
0	0	1	1

F = (X.Y) + (X′.Y′.Z)

note: X.Y.Z + X.Y.Z′ = X.Y
Z + Z′ = 1
A.Z + A.Z′ = A

Truth table, analysis & synthesis
combinational circuit example
Wakerly - Table 1.2

Objectives:

(a) start with a truth-table specification

(b) construct the logic function F=f(X,Y,Z)

(c) construct gate-level realizations

(d) implement them with MATLAB

(e) realize them in Simulink, and on

(f) the Emona FPGA board

these steps are reversible – one could start with any of them and derive the rest

	X	Y	Z	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

logic function

F = f(X,Y,Z)

with 3 input variables, there are 2^3=8 possible input triplets, and 2^3 outputs F

Example - constructing a logic function from a truth table (Table 1.2)

logic function
F = f(X,Y,Z)

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$X' . Y' . Z$

$X . Y . Z'$

$X . Y . Z$

$X . Y$

$F = (X . Y) + (X' . Y' . Z)$

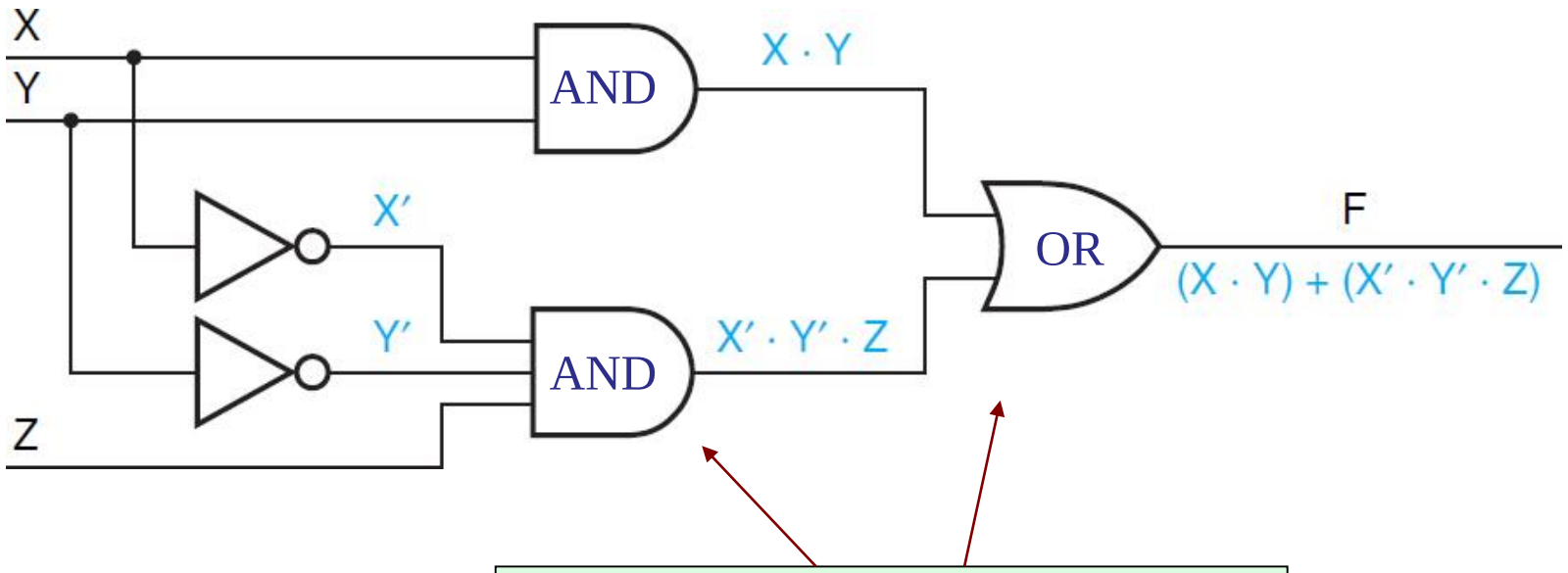
simplified form
harder to guess

full form
easy to guess

$F = (X' . Y' . Z) + (X . Y . Z') + (X . Y . Z)$

Gate-level realization
of the logic circuit of Table 1.2
(Wakerly / Fig.1-5)

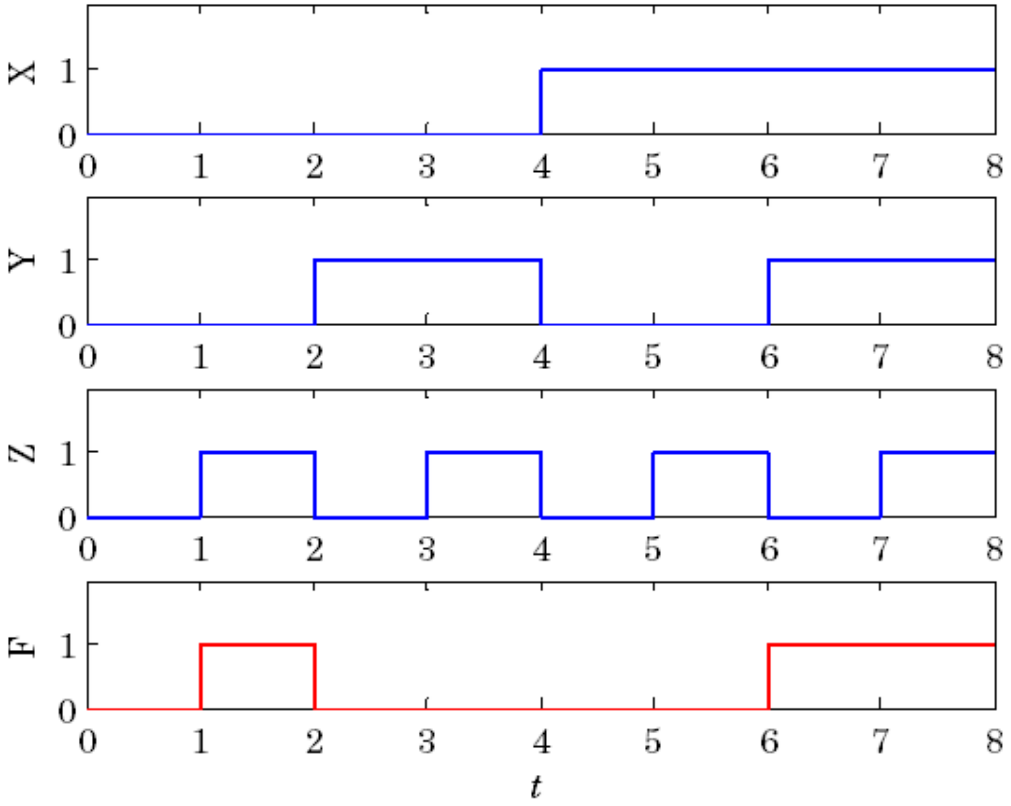
$$F = (X.Y) + (X'.Y'.Z)$$



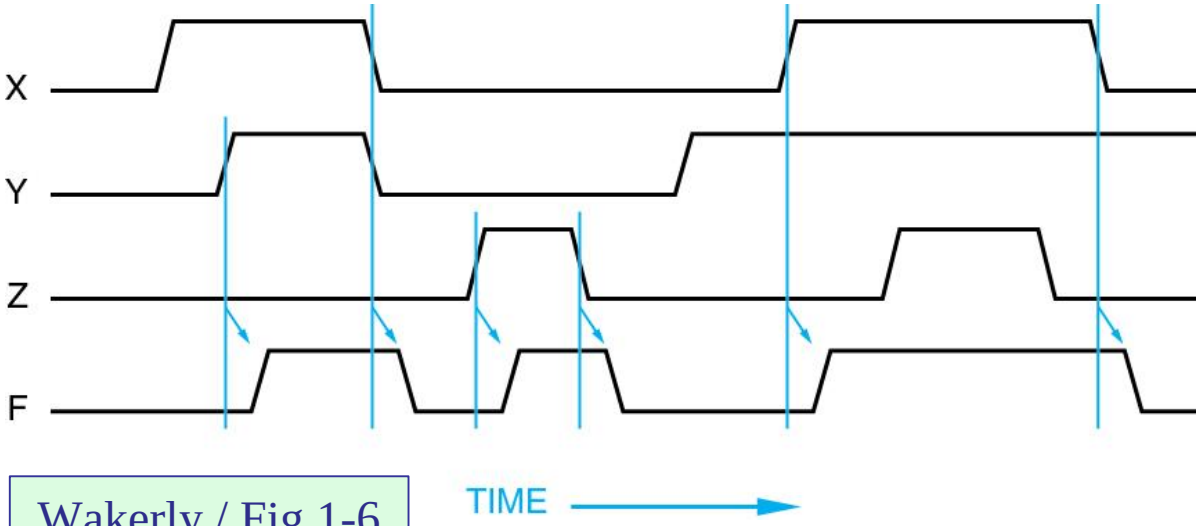
two-level, AND-OR, realization

Timing Diagrams

ideal timing diagram
with X,Y,Z derived
from 3-bit binary counter



realistic timing diagram
with arbitrary X,Y,Z
and small output delays



X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

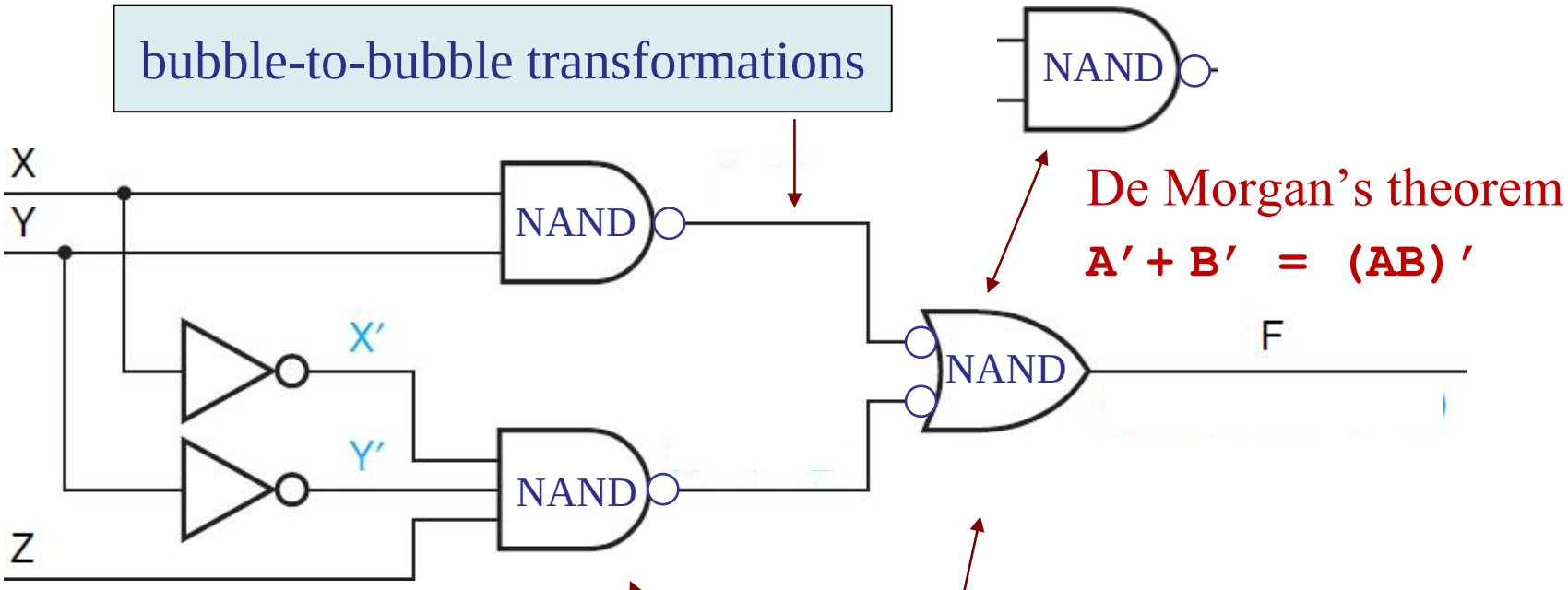
Wakerly / Fig.1-6

Alternative, gate-level realization of the logic circuit of Table 1.5 using NAND gates

$$F = XY + X'Y'Z$$
$$F = ((XY)'(X'Y'Z)')'$$

De Morgan's theorem

bubble-to-bubble transformations



De Morgan's theorem

$$A' + B' = (AB)'$$

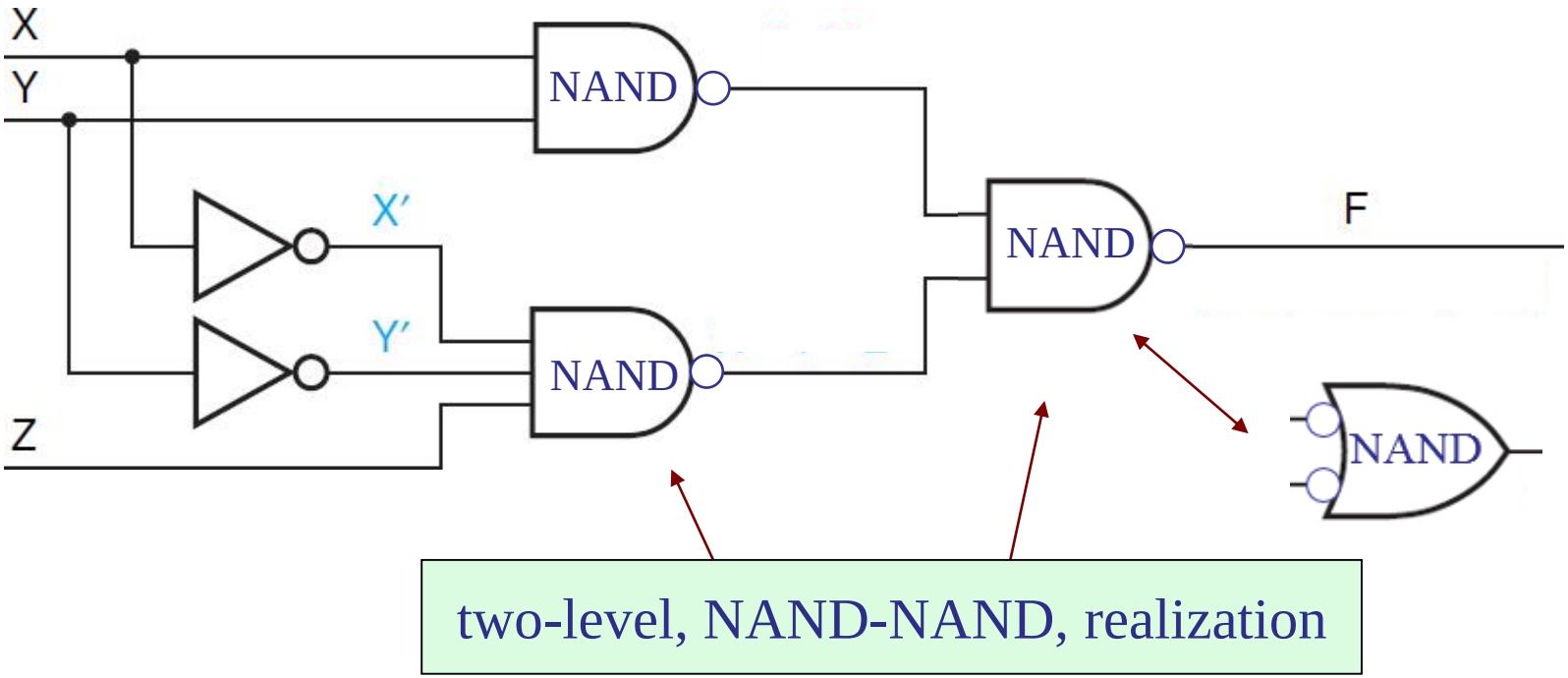
two-level, NAND-NAND, realization

Alternative, gate-level realization of the logic circuit of Table 1.5 using NAND gates

$$F = XY + X'Y'Z$$
$$F = ((XY)'(X'Y'Z)')'$$

equivalent expressions from De Morgan's theorem

$$A + B = (A'B')'$$



two-level, NAND-NAND, realization

other equivalent expressions
to be justified later in unit-3 and Ch.3

$F = XY + X'Y'Z$ = AND-OR

De Morgan

$= ((XY)'(X'Y'Z)')'$ = NAND-NAND

$= (X+Y')(X'+Y)(Y+Z)$ = OR-AND

$= (X+Y')(X'+Y)(X+Z)$ = OR-AND

De Morgan

$= ((X+Y')' + (X'+Y)' + (Y+Z)')'$ = NOR-NOR

$= ((X+Y')' + (X'+Y)' + (X+Z)')'$ = NOR-NOR

$= X'Y'Z + XYZ' + XYZ$

= canonical minterm sum-of-products (SOP) form

$= (X+Y+Z)(X+Y'+Z)(X+Y'+Z')(X'+Y+Z)(X'+Y+Z')$

= canonical maxterm product-of-sums (POS) form

MATLAB implementation - truth-table computation and timing diagram

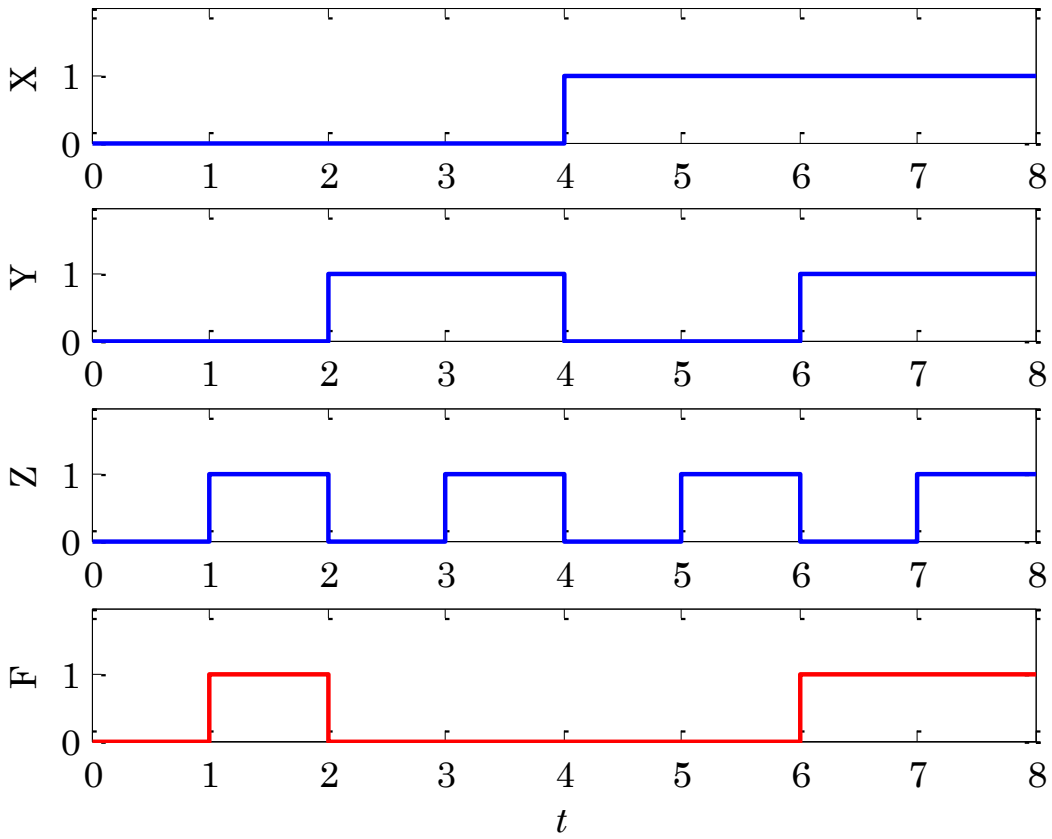
```
n = (0:7)'; % input integers
[X,Y,Z] = a2d(n,3);
F = (X & Y) | (~X & ~Y & Z);
```

$F = XY + X'Y'Z$

&, |, ~ are vectorized in MATLAB

```
[n, X,Y,Z,F]
 0 | 0 0 0 0
 1 | 0 0 1 1
 2 | 0 1 0 0
 3 | 0 1 1 0
 4 | 1 0 0 0
 5 | 1 0 1 0
 6 | 1 1 0 1
 7 | 1 1 1 1
```

```
% a2d in Canvas
% M-files folder
```



plotting the timing diagram

```
[X,Y,Z] = a2d(0:7,3);
F = (X & Y) | (~X & ~Y & Z);

t = (0:8);            % last bit goes from t=7 to t=8
x = [X; X(end)];      % extend duration of last bit to t=8
y = [Y; Y(end)];
z = [Z; Z(end)];
f = [F; F(end)];

figure;
subplot(4,1,1); stairs(t,x,'b-');
     yaxis(0,2,0:1); xaxis(0,8,0:8); ylabel('X');
subplot(4,1,2); stairs(t,y,'b-');
     yaxis(0,2,0:1); xaxis(0,8,0:8); ylabel('Y');
subplot(4,1,3); stairs(t,z,'b-');
     yaxis(0,2,0:1); xaxis(0,8,0:8); ylabel('Z');
subplot(4,1,4); stairs(t,f,'r-');
     yaxis(0,2,0:1); xaxis(0,8,0:8); ylabel('F');
xlabel('\itt');
```

a2d, xaxis, yaxis
are in Canvas M-files

verify truth-table in full minterm SOP form

$F = X'Y'Z + XYZ' + XYZ$

simplified form

$F = XY + X'Y'Z$

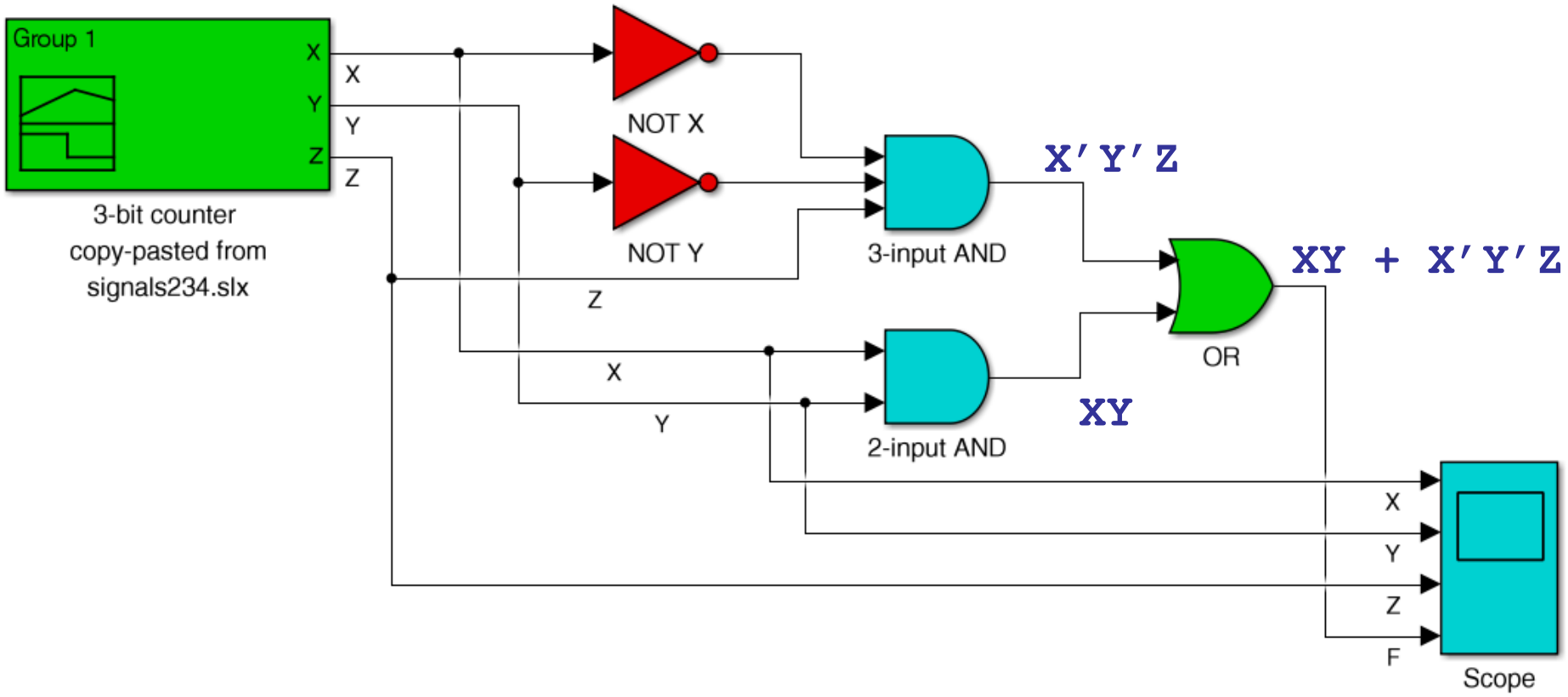
```
n = (0:7)';
[X,Y,Z] = a2d(n,3);

F = (~X & ~Y & Z) | (X & Y & ~Z) | (X & Y & Z);

[n,  X, Y, Z,  F,  ~X&~Y&Z, X&Y&~Z,  X&Y&Z]
```

n	X	Y	Z	F	~X&~Y&Z	X&Y&~Z	X&Y&Z
0	0	0	0	0	0	0	0
1	0	0	1	1	1	0	0
2	0	1	0	0	0	0	0
3	0	1	1	0	0	0	0
4	1	0	0	0	0	0	0
5	1	0	1	0	0	0	0
6	1	1	0	1	0	1	0
7	1	1	1	1	0	0	1

F = OR-ing of the three individual minterm columns

Simulink implementation - AND-OR version

$$F = XY + X'Y'Z$$



see file **`fig1_5.slx`** included in Canvas M-files folder

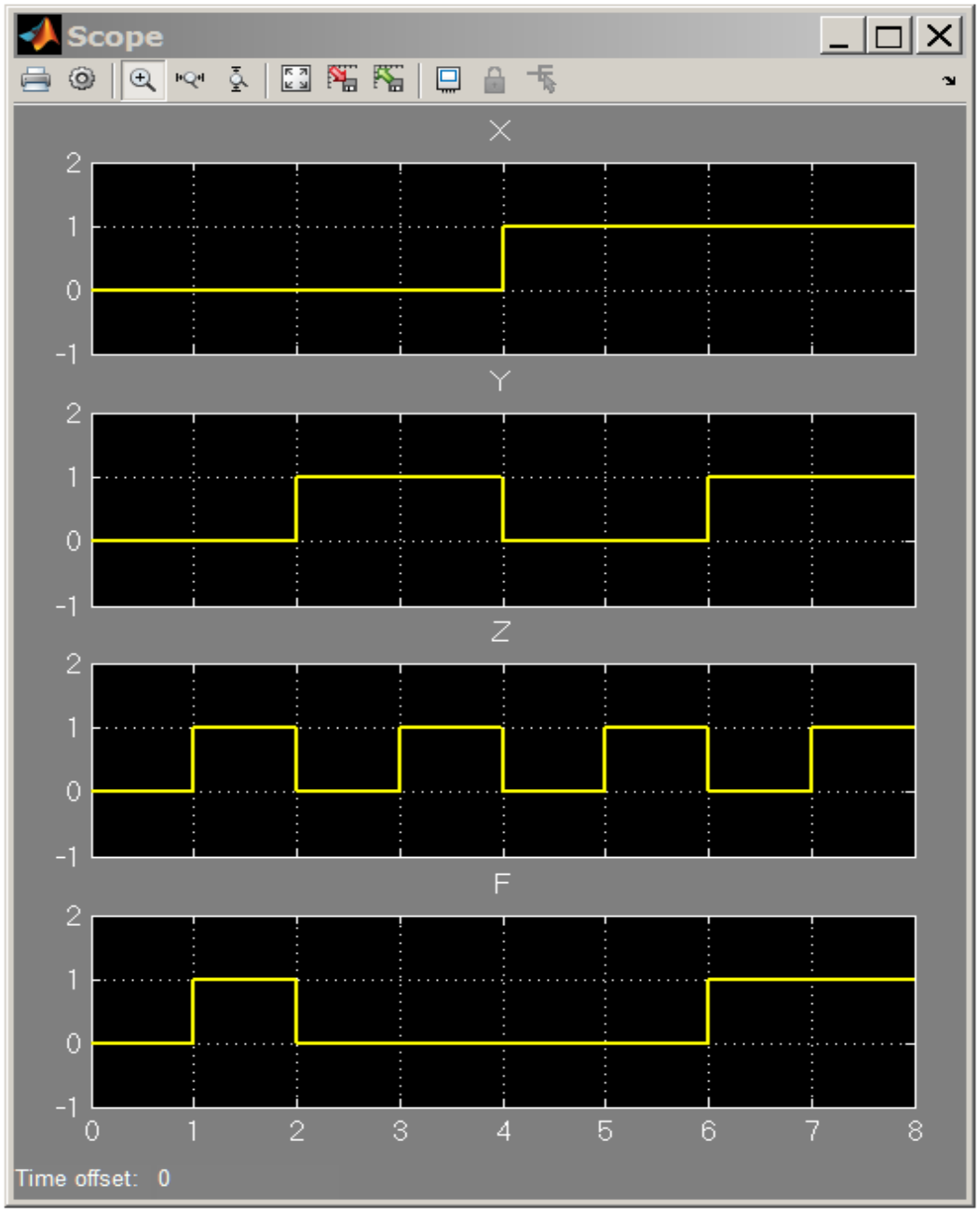
scope output

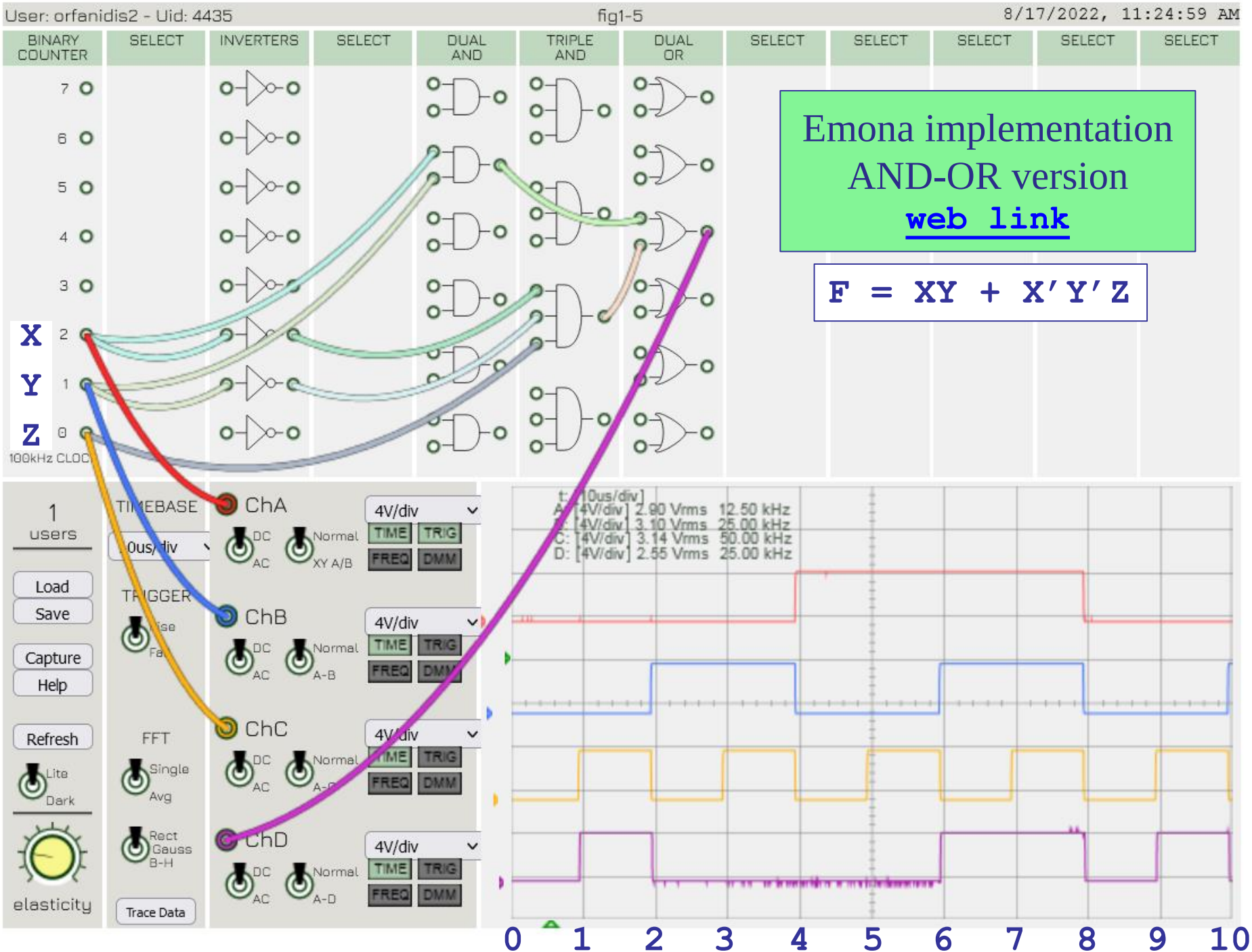
Simulink implementation

$$F = XY + X'Y'Z$$

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

scope output




User: orfanidis2 - Uid: 4435
fig1-5
8/17/2022, 11:24:59 AM
BINARY COUNTER
SELECT
INVERTERS
SELECT
DUAL AND
TRIPLE AND
DUAL OR
SELECT
SELECT
SELECT
SELECT
SELECT
X
Y
Z
100kHz CLOCK
Emona implementation
AND-OR version
web link
F = XY + X′Y′Z
1 users
Load
Save
Capture
Help
Refresh
Lite
Dark
elasticity
TIMEBASE
TRIGGER
FFT
Single
Avg
Rect
Gauss
B-H
Trace Data
ChA
ChB
ChC
ChD
DC
AC
Normal
XY A/B
A-B
A-D
4V/div
TIME
TRIG
FREQ
DMM
A: [4V/div] 2.90 Vrms 12.50 kHz
B: [4V/div] 3.10 Vrms 25.00 kHz
C: [4V/div] 3.14 Vrms 50.00 kHz
D: [4V/div] 2.55 Vrms 25.00 kHz
0 1 2 3 4 5 6 7 8 9 10

Example Summary:

(a) we started with a truth-table specification

(b) and constructed the logic function, F = f(X,Y,Z)

(c) then, obtained a gate-level realization (and variants)

(d) implemented it with MATLAB

(e) and realized it in Simulink and on the Emona board

more general procedures

Design Levels: **(a) functional definition level**

(b) transistor level

(c) truth-table level, FPGA look-up tables

(d) gate-level realizations

(e) Verilog HDL, structural or behavioral models

(f) MATLAB, Simulink, Emona implementations

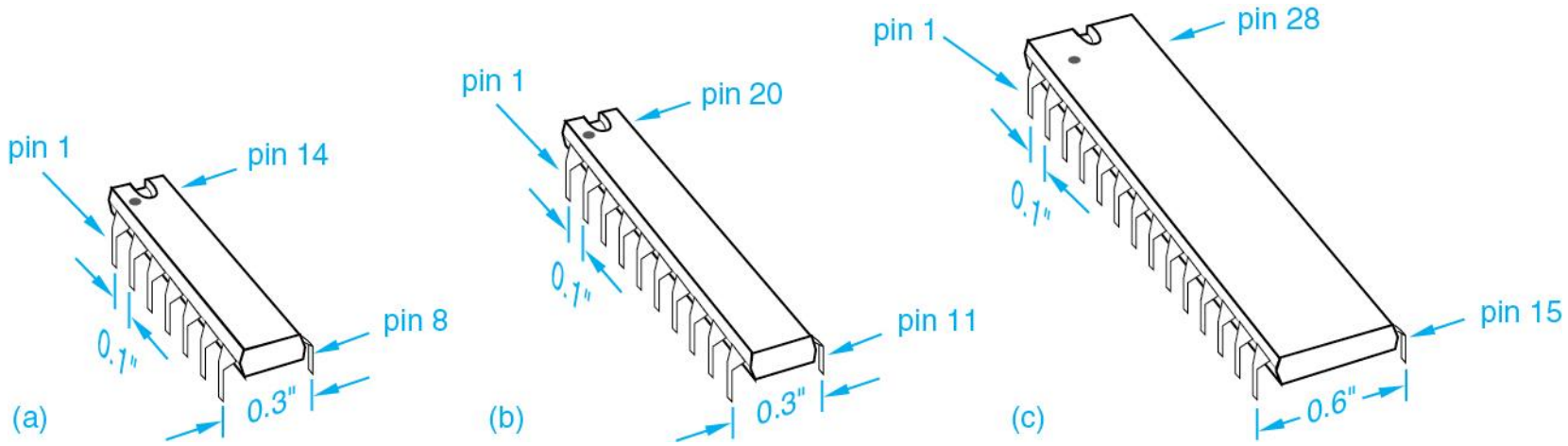
integrated circuit complexity

complexity	gates/chip	application examples
small scale integration (SSI)	10	logic gates, flip flops
medium scale integration (MSI)	10-100	adders, counters
large scale integration (LSI)	100-10,000	ROM, RAM, 8-bit processors
very large scale integration (VLSI)	10,000-100,000	16- and 32-bit processors

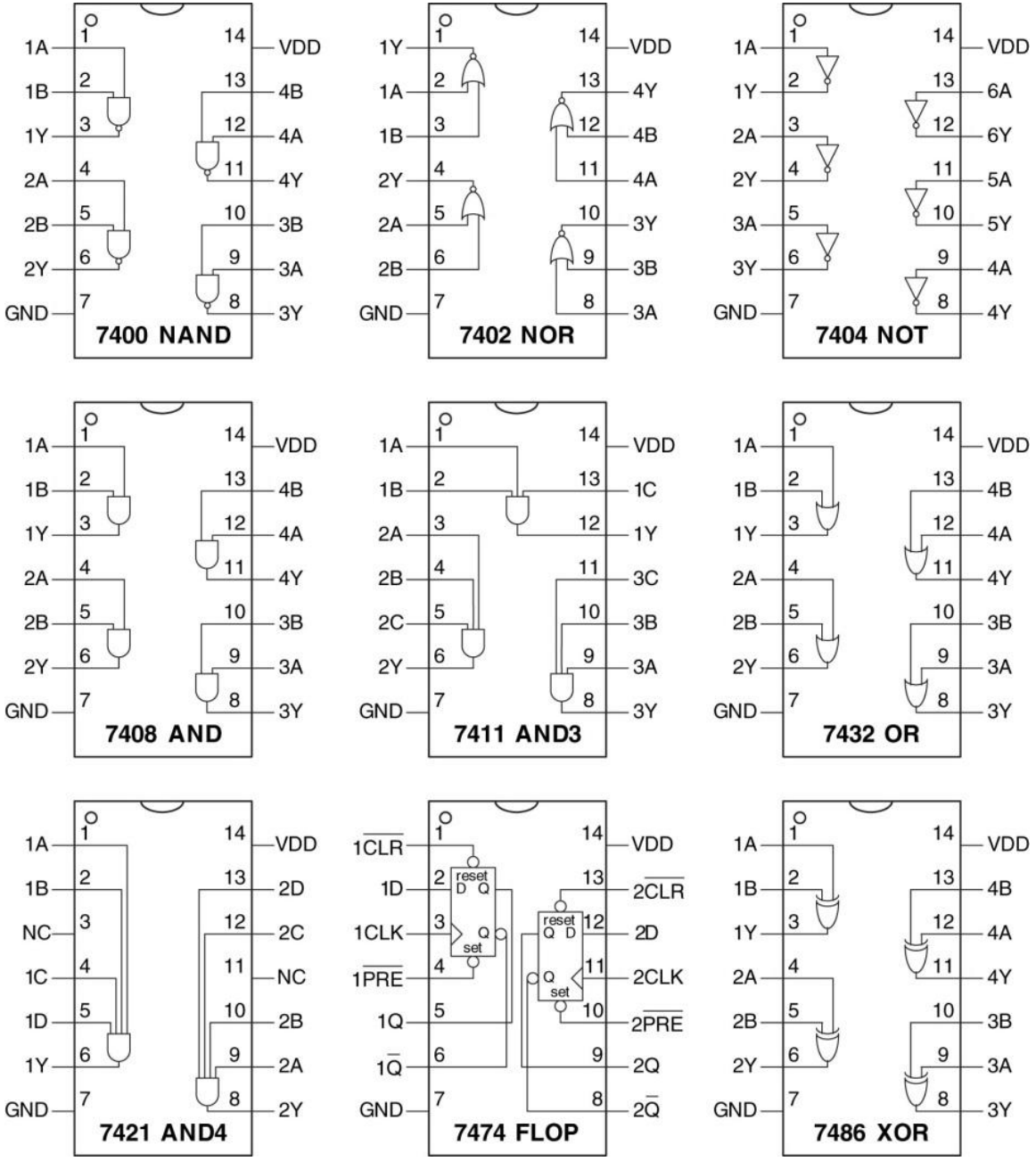
web link – Wikipedia – Integrated Circuits

Integrated Circuits (ICs)
LSI, MSI, VLSI
Moore's Law

Fig. 1-8 Dual Inline Pin (DIP) Packages:
(A) 14-Pin; (B) 20-Pin; (C) 28-Pin

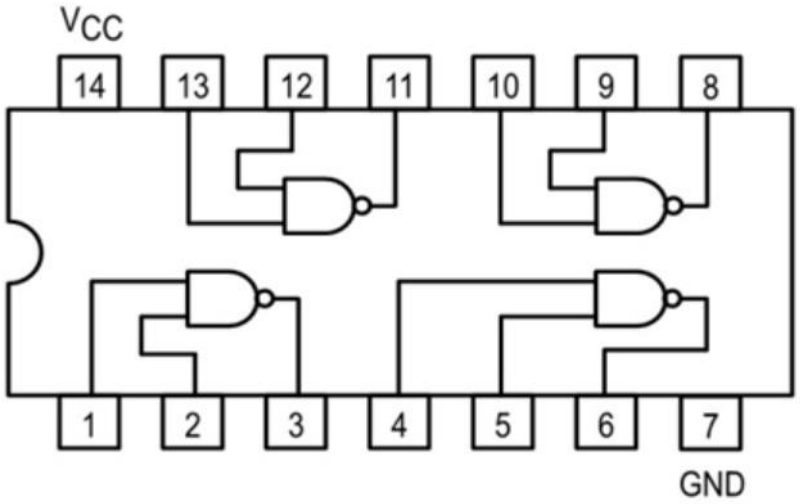


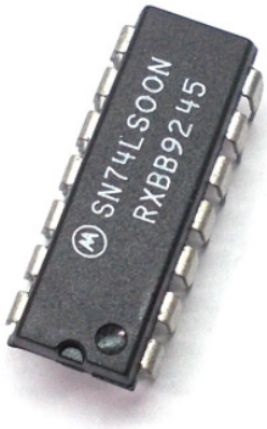
Examples of ICs

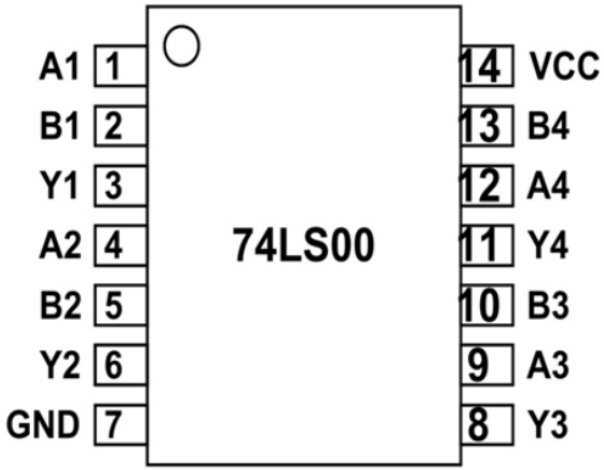


(c) D. M. Harris & S. L. Harris, *Digital Design and Computer Architecture*, 2/e, Elsevier, 2013

NAND gate

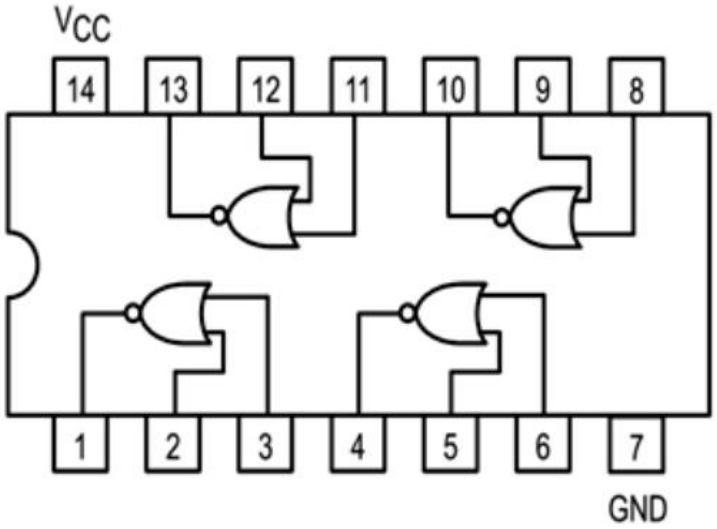


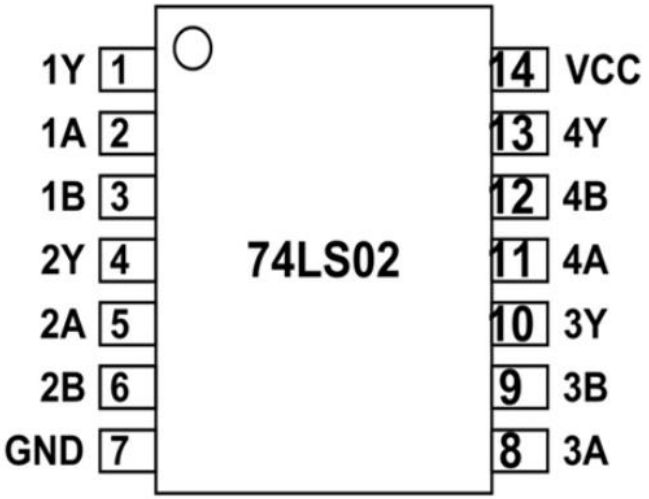




https://components101.com/ics/74ls00-quad-two-input-nand-gate

NOR gate





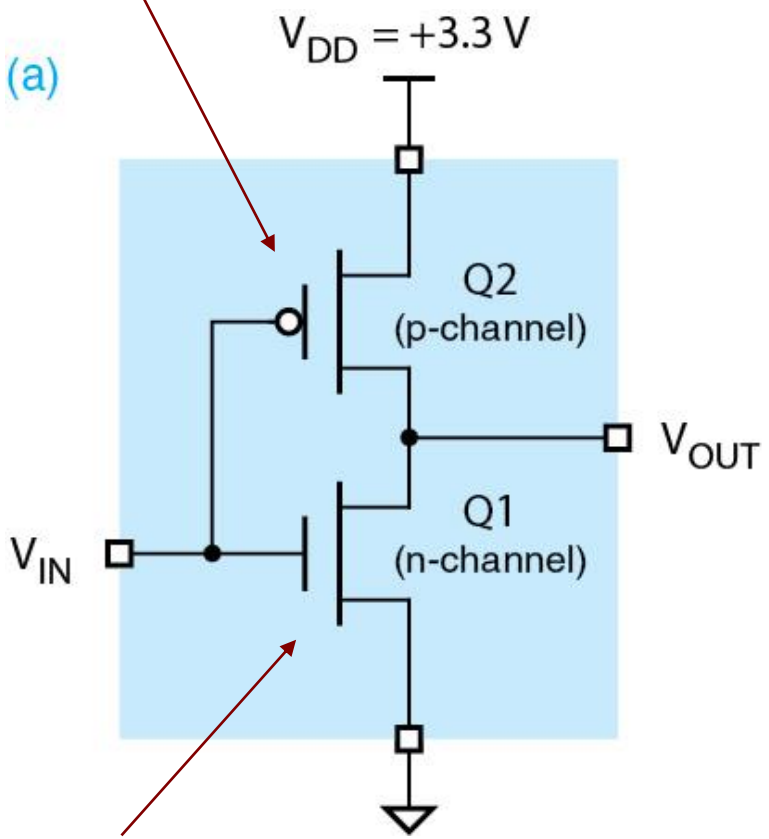
https://components101.com/ics/74ls02-nor-gate-ic

CMOS NOT gate

PMOS transistor

https://en.wikipedia.org/wiki/CMOS

(a)

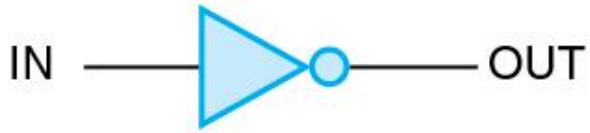


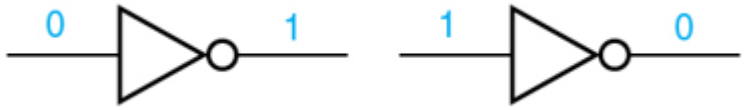
NMOS transistor

(b)

V_{IN}	Q1	Q2	V_{OUT}
0.0 (LOW)	off	on	3.3 (HIGH)
3.3 (HIGH)	on	off	0.0 (LOW)

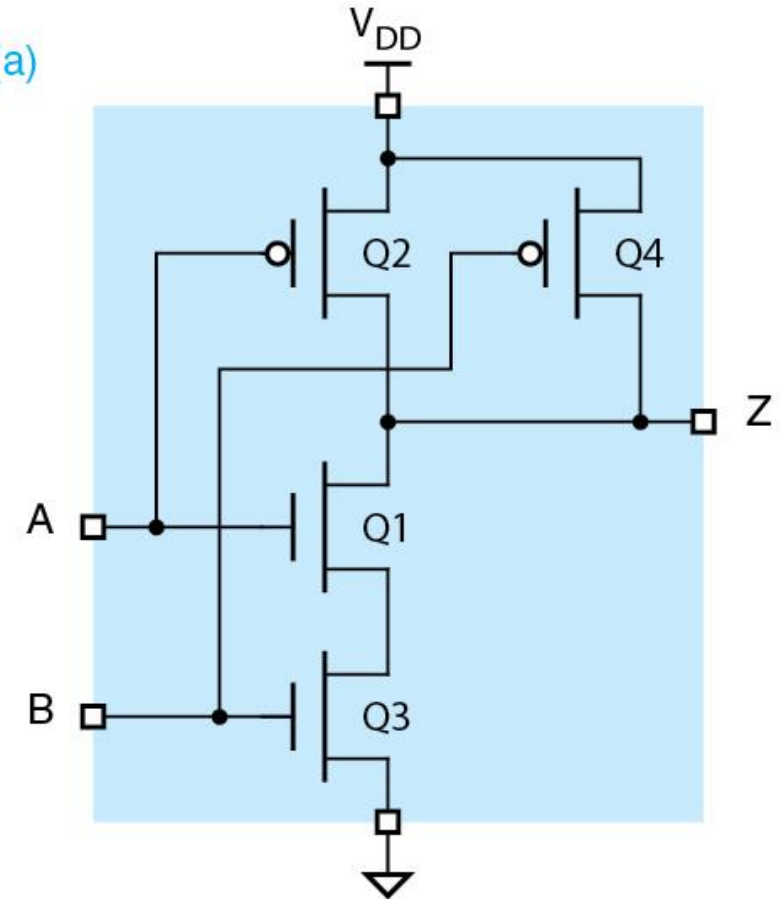
(c)





CMOS NAND gate

(a)



(b)

A	B	Q1	Q2	Q3	Q4	Z
LOW	LOW	off	on	off	on	HIGH
LOW	HIGH	off	on	on	off	HIGH
HIGH	LOW	on	off	off	on	HIGH
HIGH	HIGH	on	off	on	off	LOW

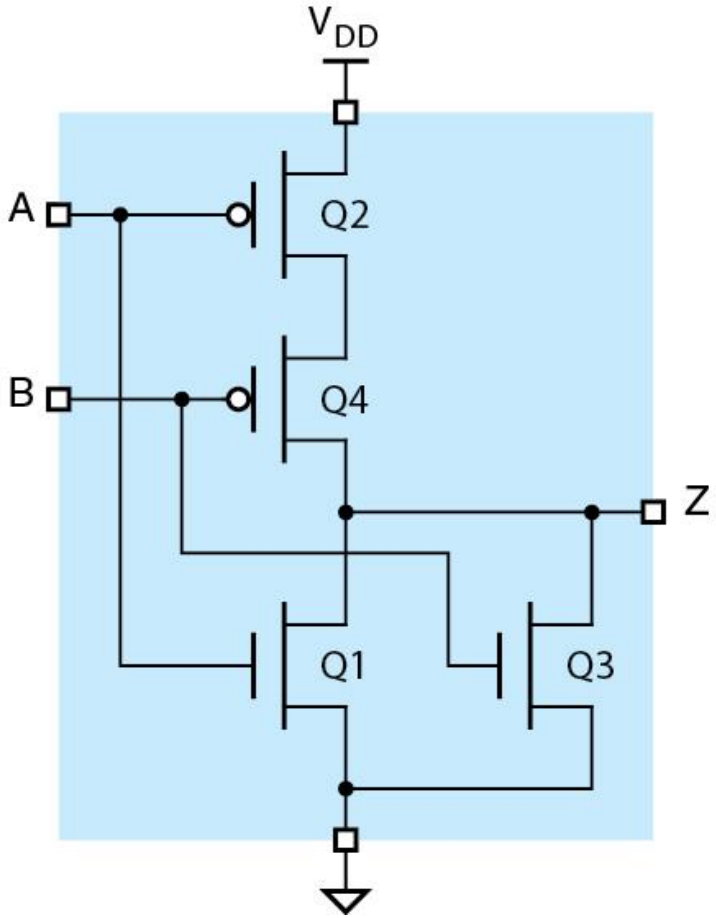
(c)

X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0

https://en.wikipedia.org/wiki/CMOS

CMOS NOR gate

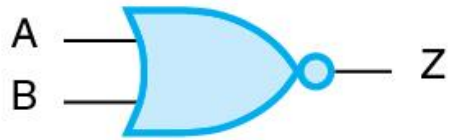
(a)



(b)

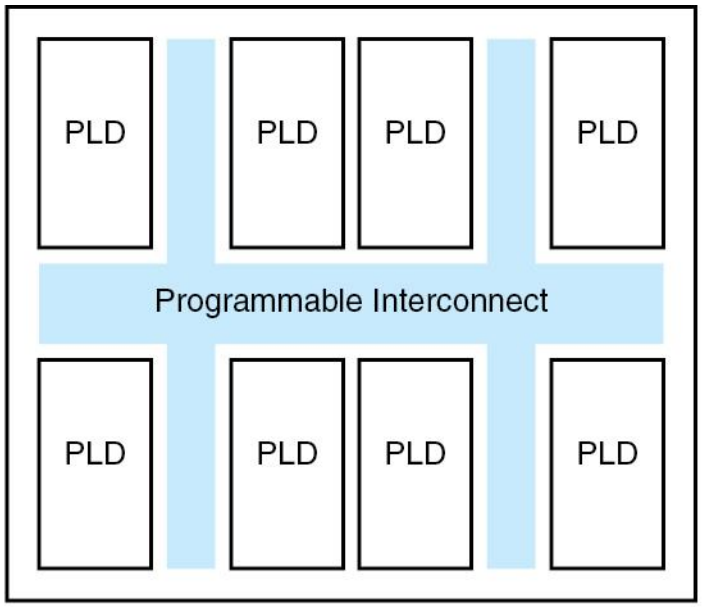
A	B	Q1	Q2	Q3	Q4	Z
LOW	LOW	off	on	off	on	HIGH
LOW	HIGH	off	on	on	off	LOW
HIGH	LOW	on	off	off	on	LOW
HIGH	HIGH	on	off	on	off	LOW

(c)

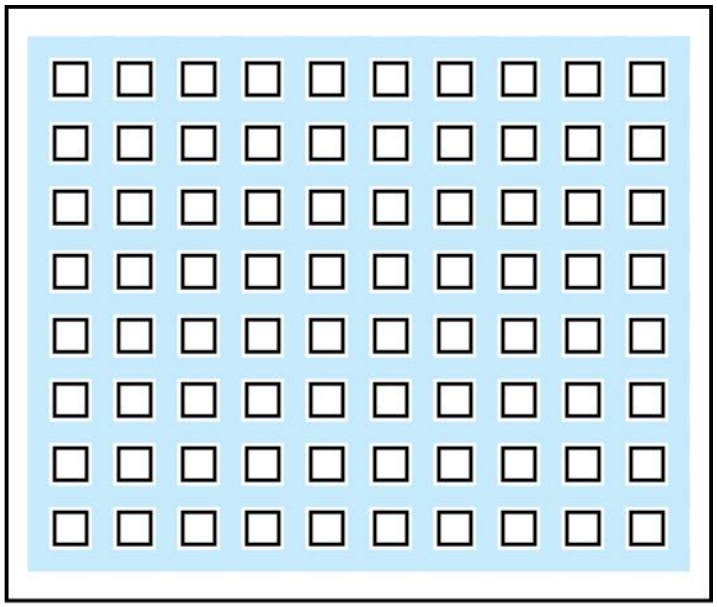


X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0

(A) Complex Programmable Logic Device (CPLD)
(B) Field-Programmable Gate Array (FPGA)



(a)



(b)

= logic block

Design Levels: **(a) functional definition level**

(b) transistor level

(c) truth-table level, FPGA look-up tables

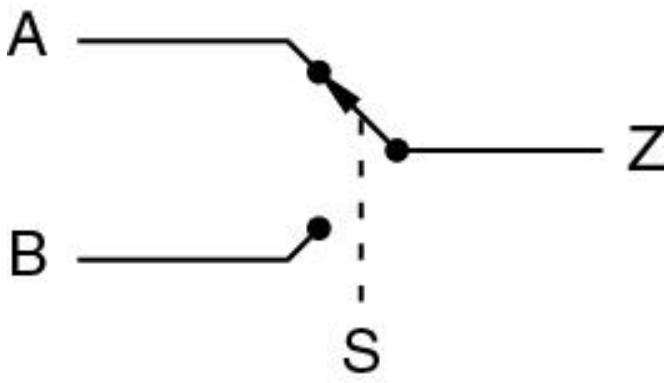
(d) gate level

(e) Verilog HDL, structural or behavioral models

(f) MATLAB and Simulink implementations

Example: switch model for a multiplexer function

Wakerly / Sect. 1.13

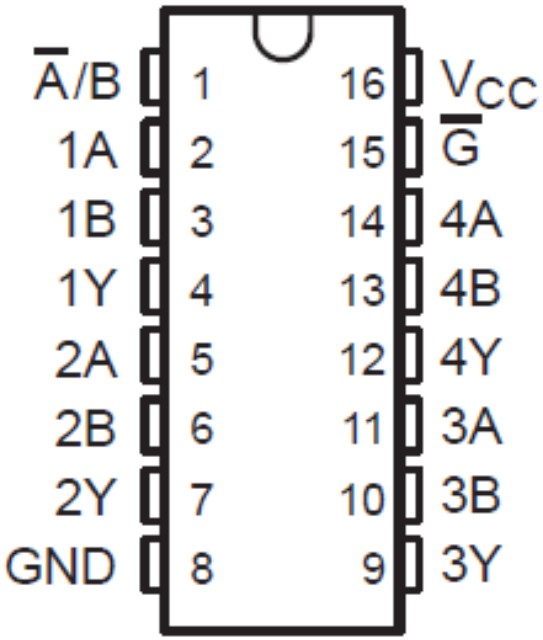


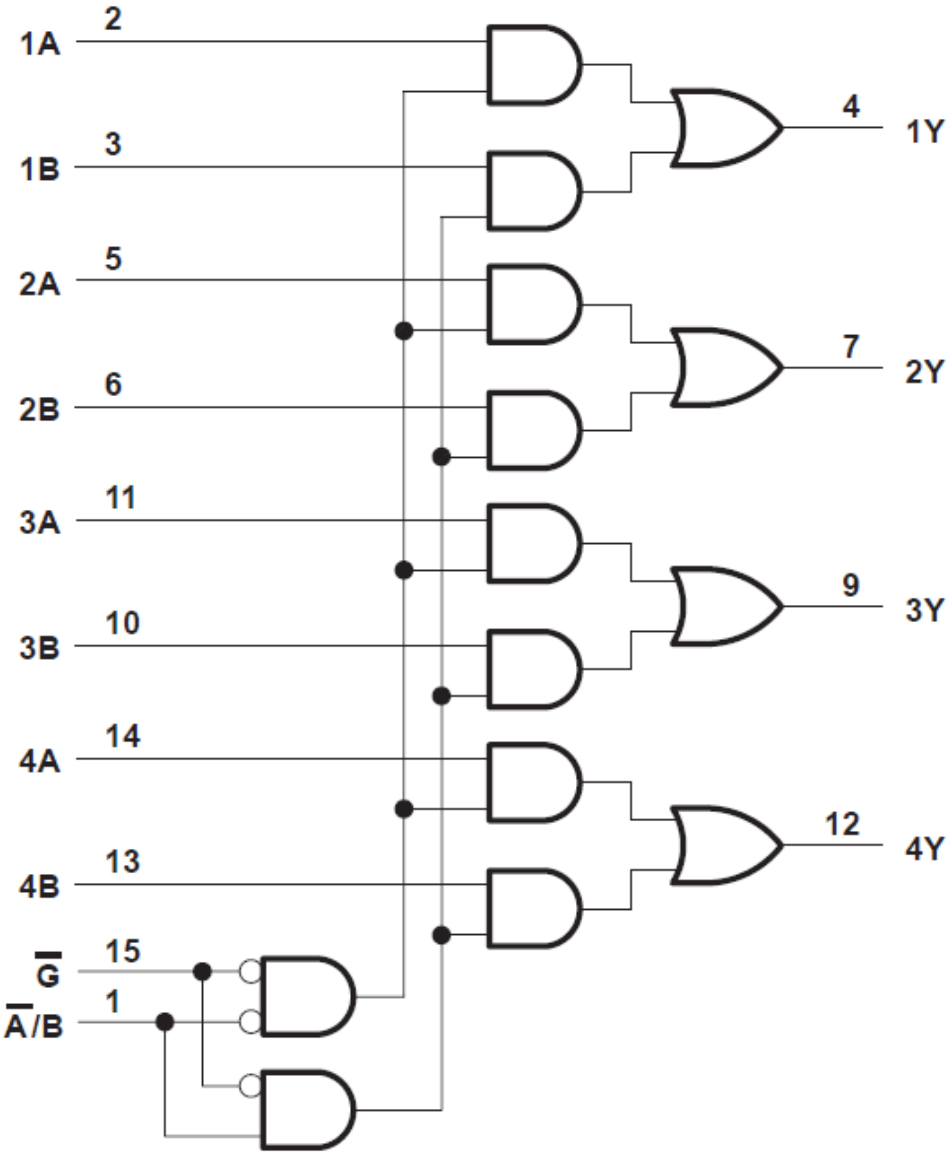
some multiplexer applications:

- network lines
- telephone lines
- communication systems
- memory controllers

MSI quad package for a multiplexer chip - TI SN74F157

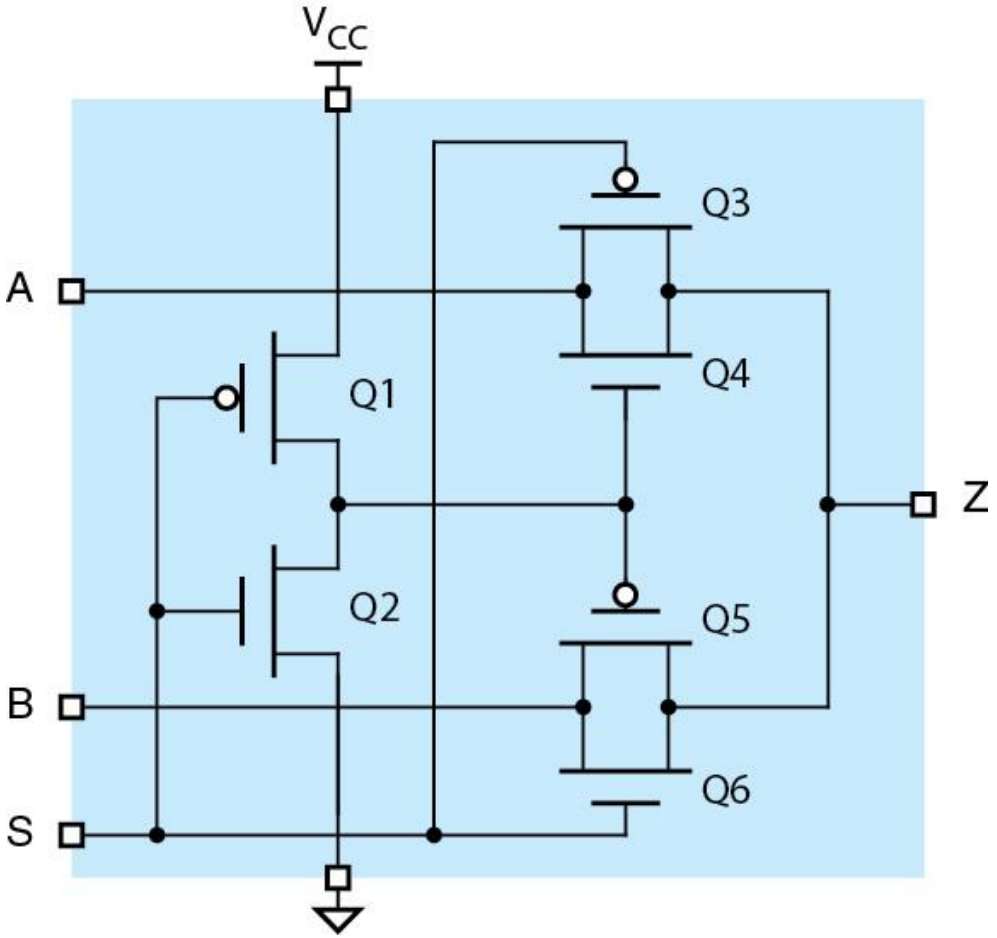
S
A
B
Z



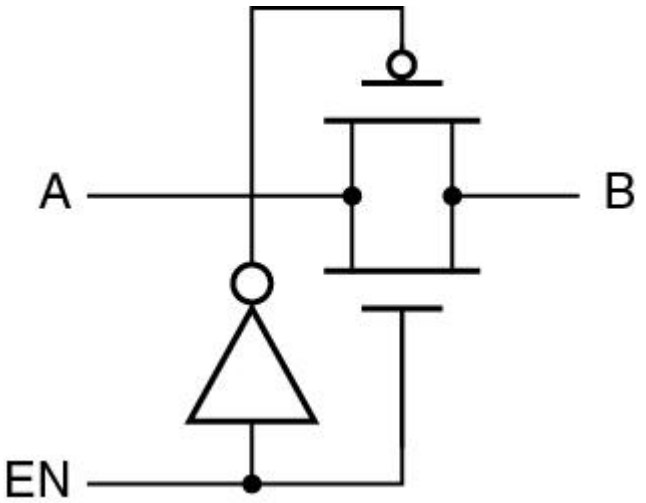


CMOS transistor model for multiplexer function

https://en.wikichip.org/wiki/multiplexer



CMOS transmission gate



truth table for multiplexer function

S	A	B	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$Z = S'AB' + S'AB + SA'B + SAB$$

full minterm sum-of-products (SOP) form

$$S'AB' + S'AB = S'A$$

$$SA'B + SAB = SB$$

simplified form

$$Z = S'A + SB$$

used properties

$$A' + A = 1$$
$$B' + B = 1$$

truth table for multiplexer function

S	A	B	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$Z = S'AB' + S'AB + SA'B + SAB$$

full minterm sum-of-products (SOP) form

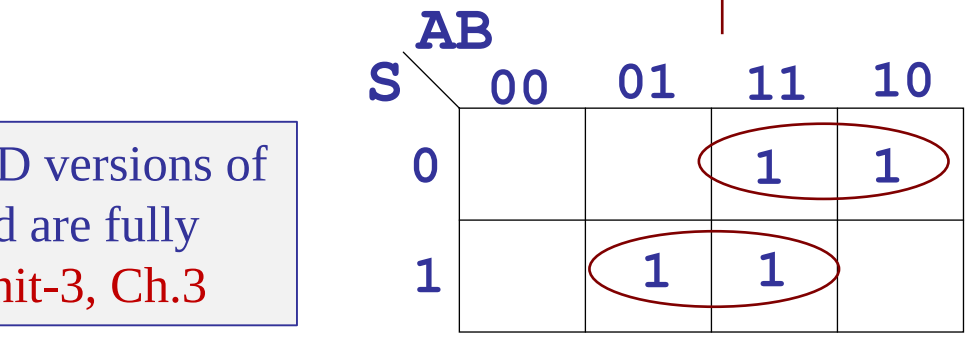
$$S'AB' + S'AB = S'A$$

$$SA'B + SAB = SB$$

simplified form

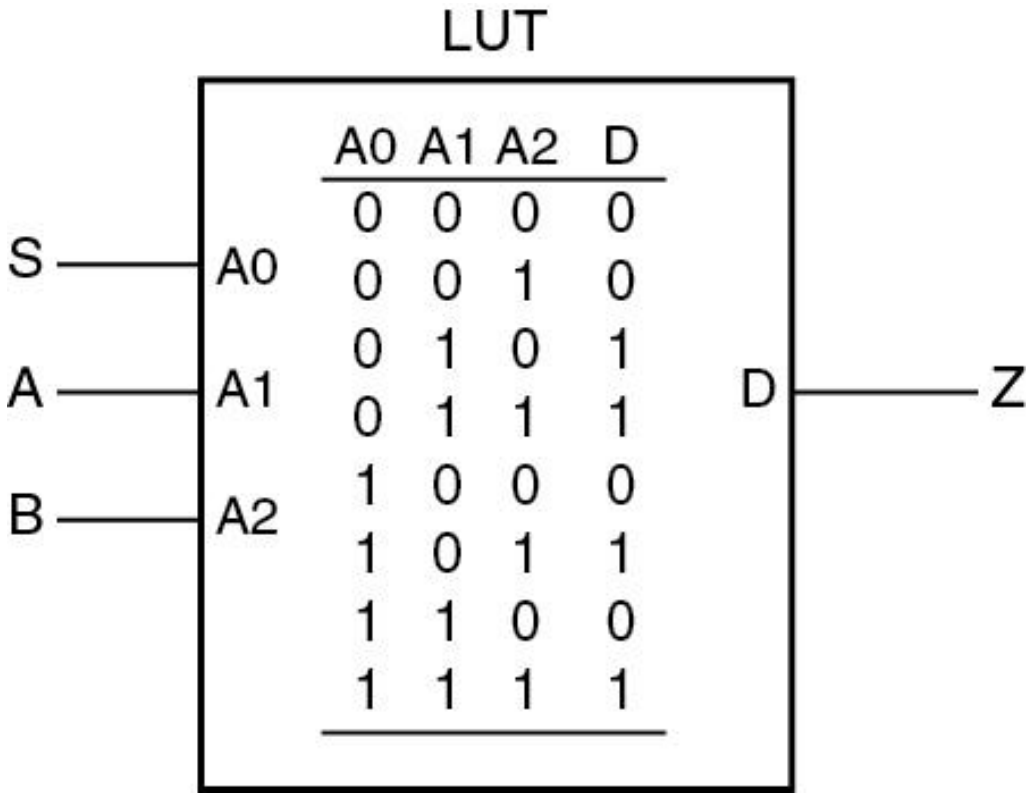
$$Z = S'A + SB$$

K-map derivation



K-maps are 2-D versions of truth tables and are fully discussed in unit-3, Ch.3

multiplexer function implemented on FPGA look-up table (LUT)

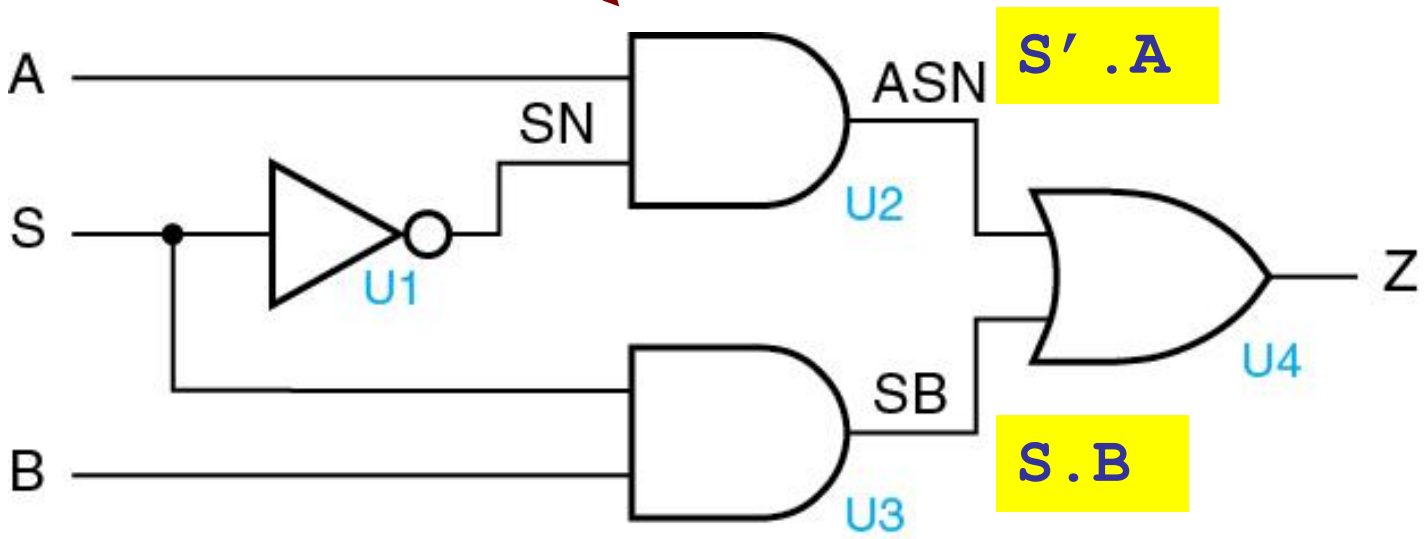


gate-level logic diagram for multiplexer function

$$Z = S'A + SB = ((S'A)'(SB)')'$$

AND-OR

NAND-NAND



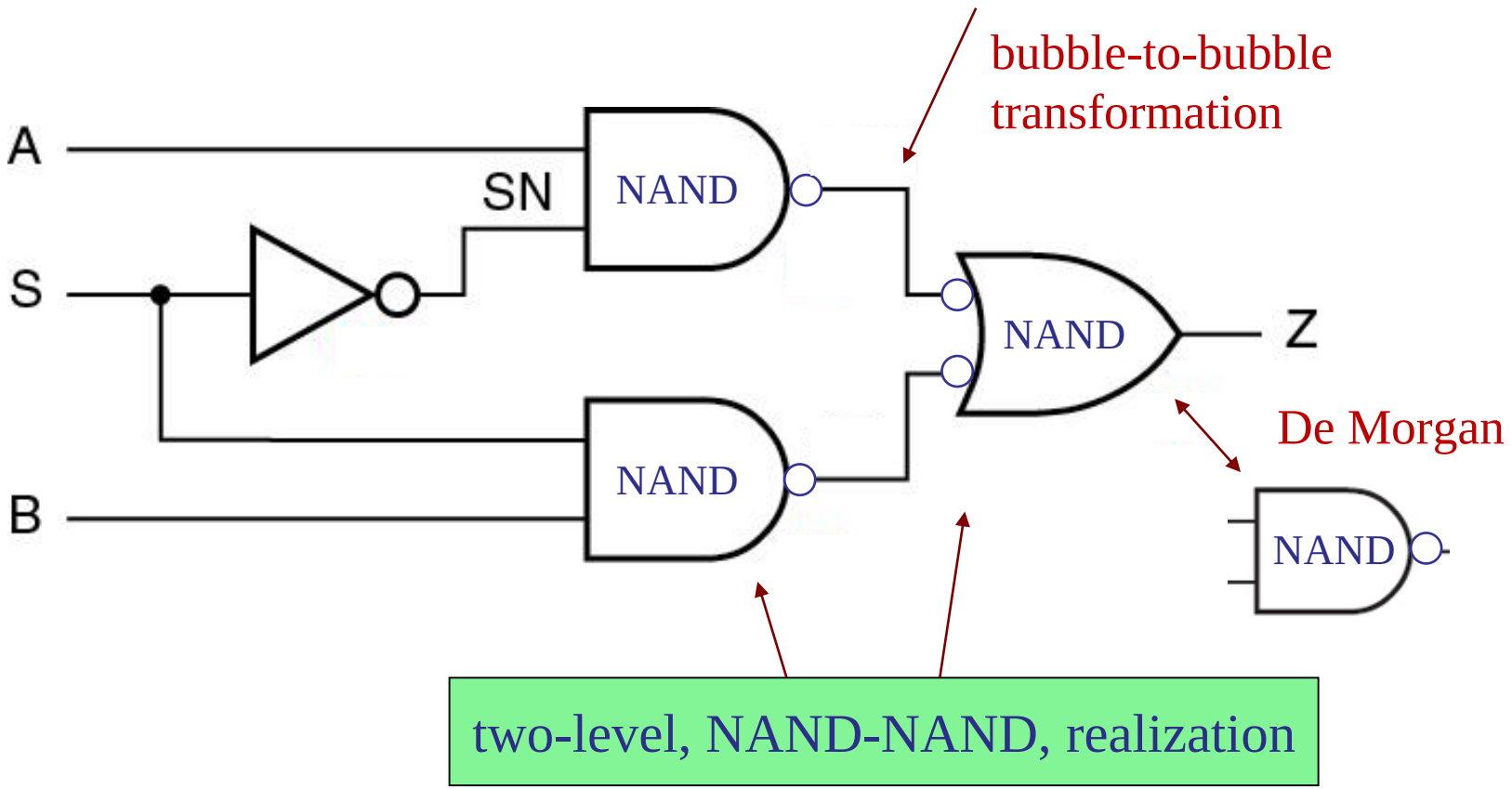
two-level, AND-OR, realization

gate-level logic diagram for multiplexer function

$$Z = S'A + SB = ((S'A)'(SB)')'$$

AND-OR

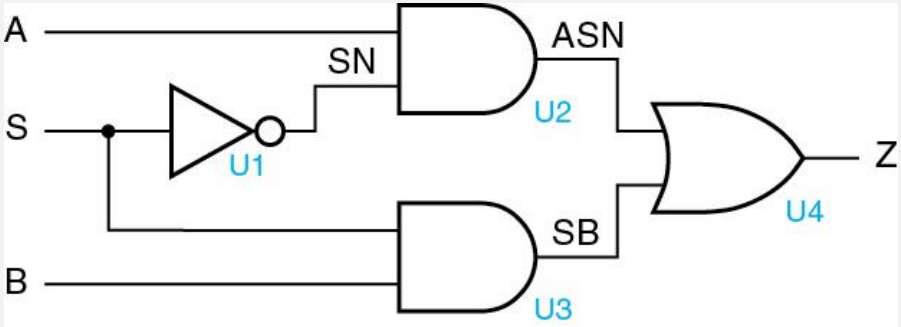
NAND-NAND



Verilog HDL – structural model

see also Wikipedia article on hardware description languages

```
// 2-input multiplexer

module Ch1mux_s(A, B, S, Z);

  input A, B, S;
  output Z;
  wire SN, ASN, SB;

  not U1 (SN, S);
  and U2 (ASN, A, SN);
  and U3 (SB, B, S);
  or  U4 (Z, ASN, SB);

endmodule
```

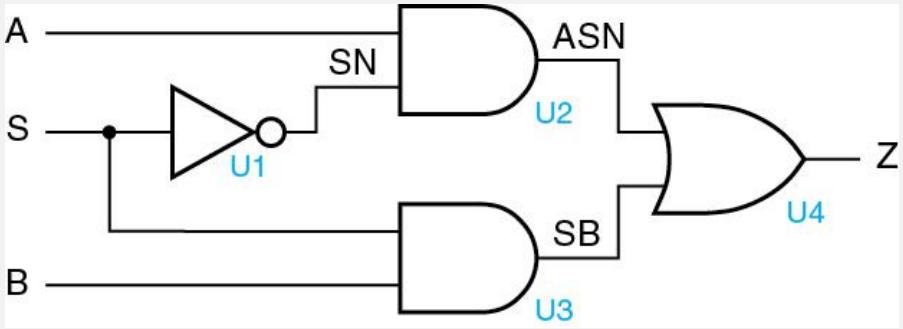


© 2018 Pearson Education, J. F. Wakerly, *Digital Design Principles and Practices*, 5/e

see also https://nandland.com/ for tutorials on HDLs

Verilog HDL – behavioral model

```
// 2-input multiplexer

module Ch1mux_b(A, B, S, Z);
  input A, B, S;
  output reg Z;

  always @ (A, B, S) if (S==1) Z = B; else Z = A;

endmodule
```



© 2018 Pearson Education, J. F. Wakerly, *Digital Design Principles and Practices*, 5/e

MATLAB implementation with M-file

$Z = S'A + SB$

```
% mux.m - 2-input multiplexer function

function Z = mux(S,A,B)
    Z = ((~S) & A) | (S & B);

% alternatively,
% Z = or(and(not(S),A), and(S,B));
```

or, and, not, are built-in functions

anonymous MATLAB function version

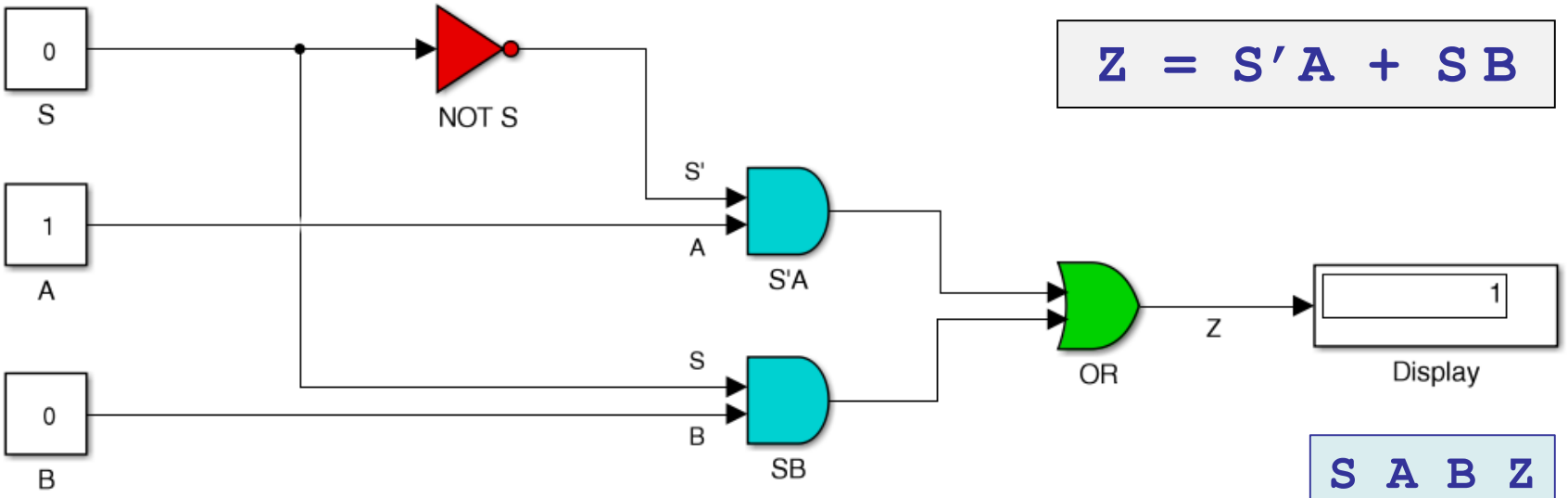
```
mux = @(S,A,B) ((~S) & A) | (S & B);

% Usage: Z = mux(S,A,B);
```

MATLAB script

```
mux = @(S,A,B) ((~S) & A) | (S & B);

[S,A,B] = a2d(0:7,3);    % all possible S,A,B values
                         % a2d to be discussed in Unit-2
Z = mux(S,A,B);

[S,A,B,Z]                % print the truth table

%  S  A  B  Z            % S,A,B,Z are columns
% ----------
%  0  0  0  0
%  0  0  1  0
%  0  1  0  1
%  0  1  1  1
%  1  0  0  0
%  1  0  1  1
%  1  1  0  0
%  1  1  1  1
```

note:
the operations **&** and | are vectorized in MATLAB

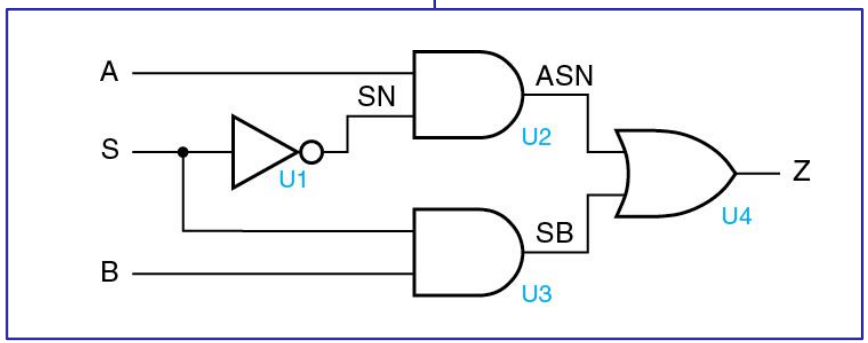
MATLAB / Simulink implementation

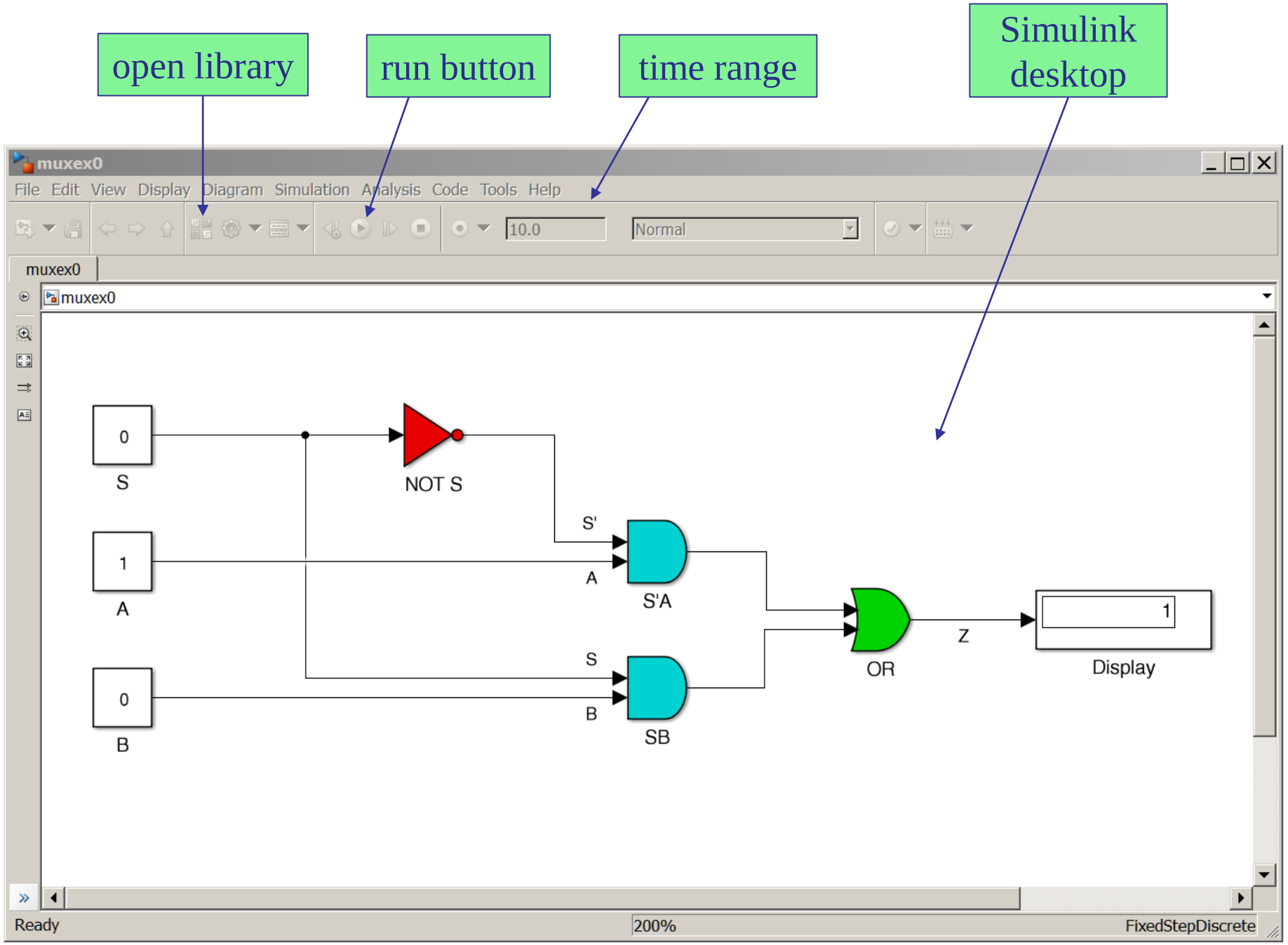


$$Z = S'A + SB$$

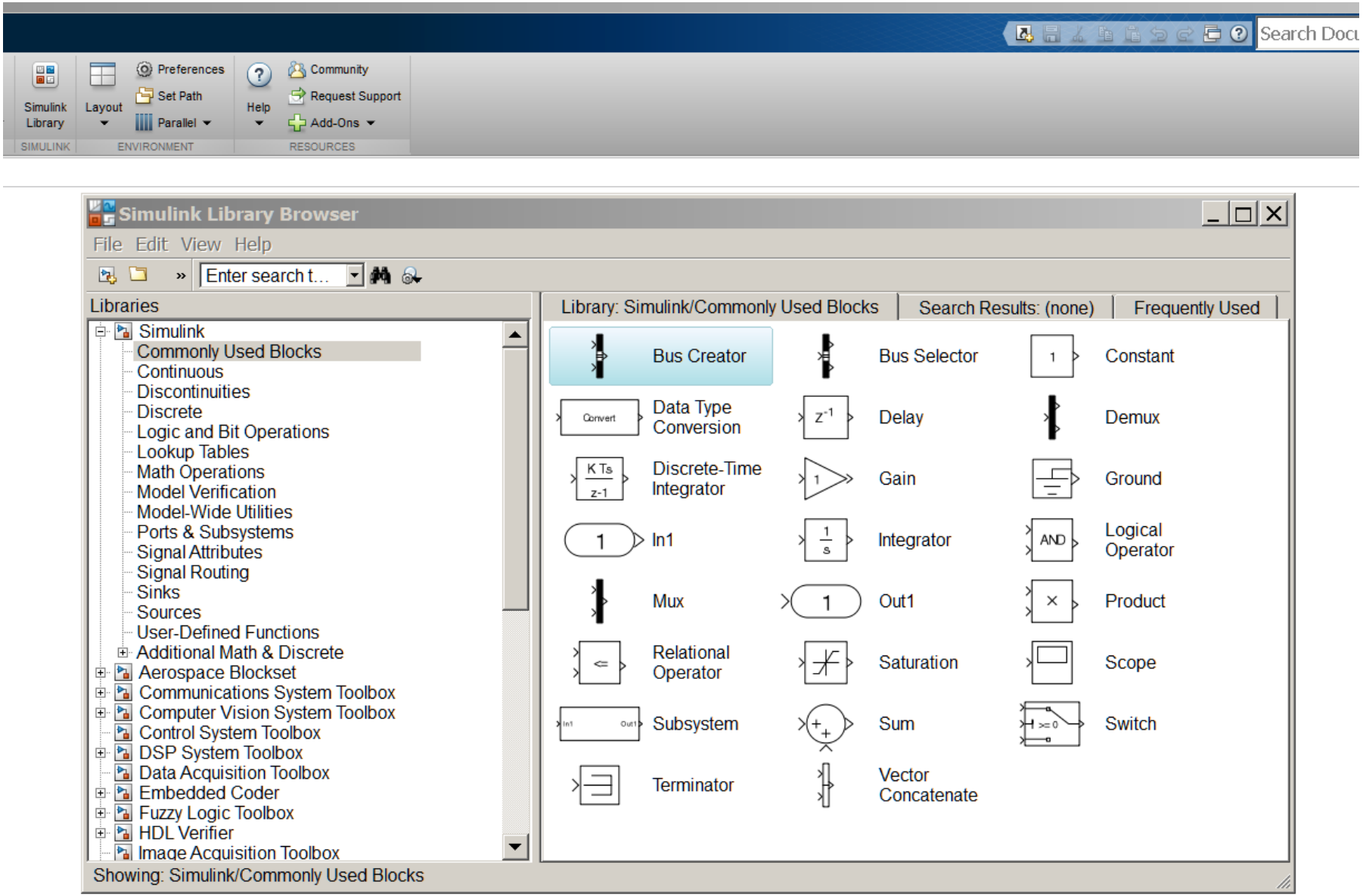
S	A	B	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

simulink implementation

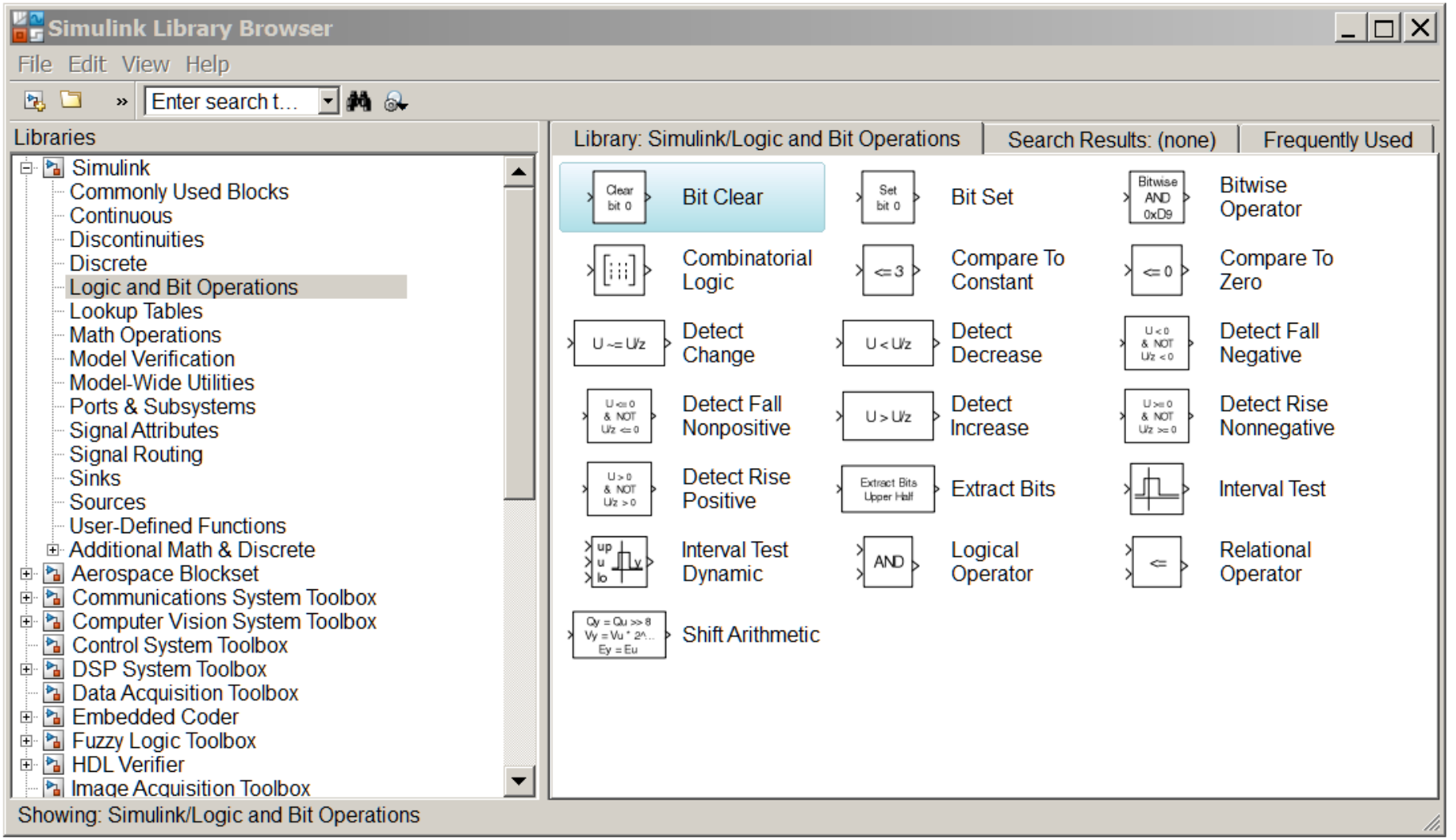



open library
run button
time range
Simulink desktop
muxex0
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
Normal
muxex0
muxex0
0
S
NOT S
1
A
S'
A
S'A
0
B
S
B
SB
OR
Z
1
Display
Ready
200%
FixedStepDiscrete

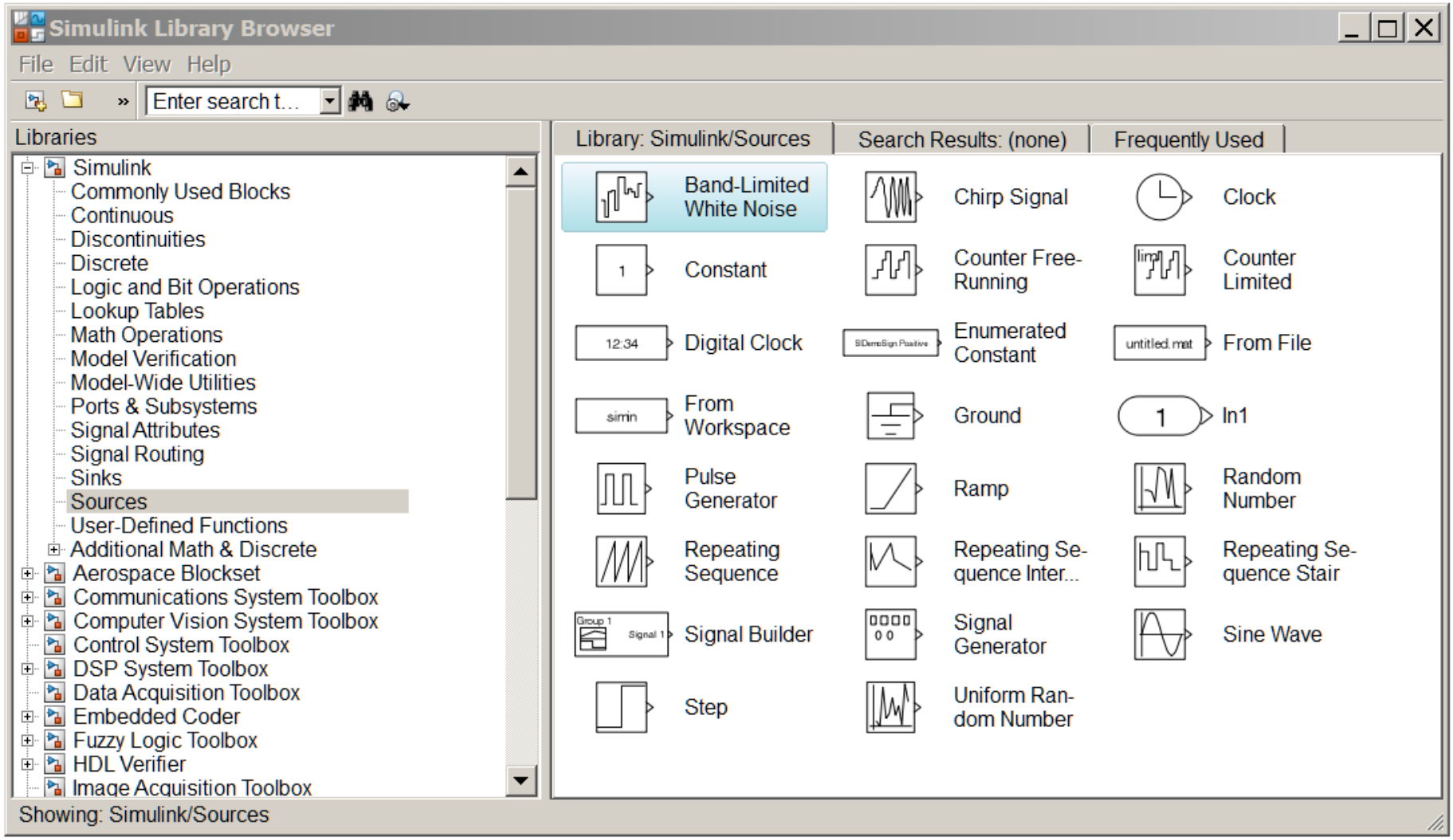
Simulink library – commonly used blocks



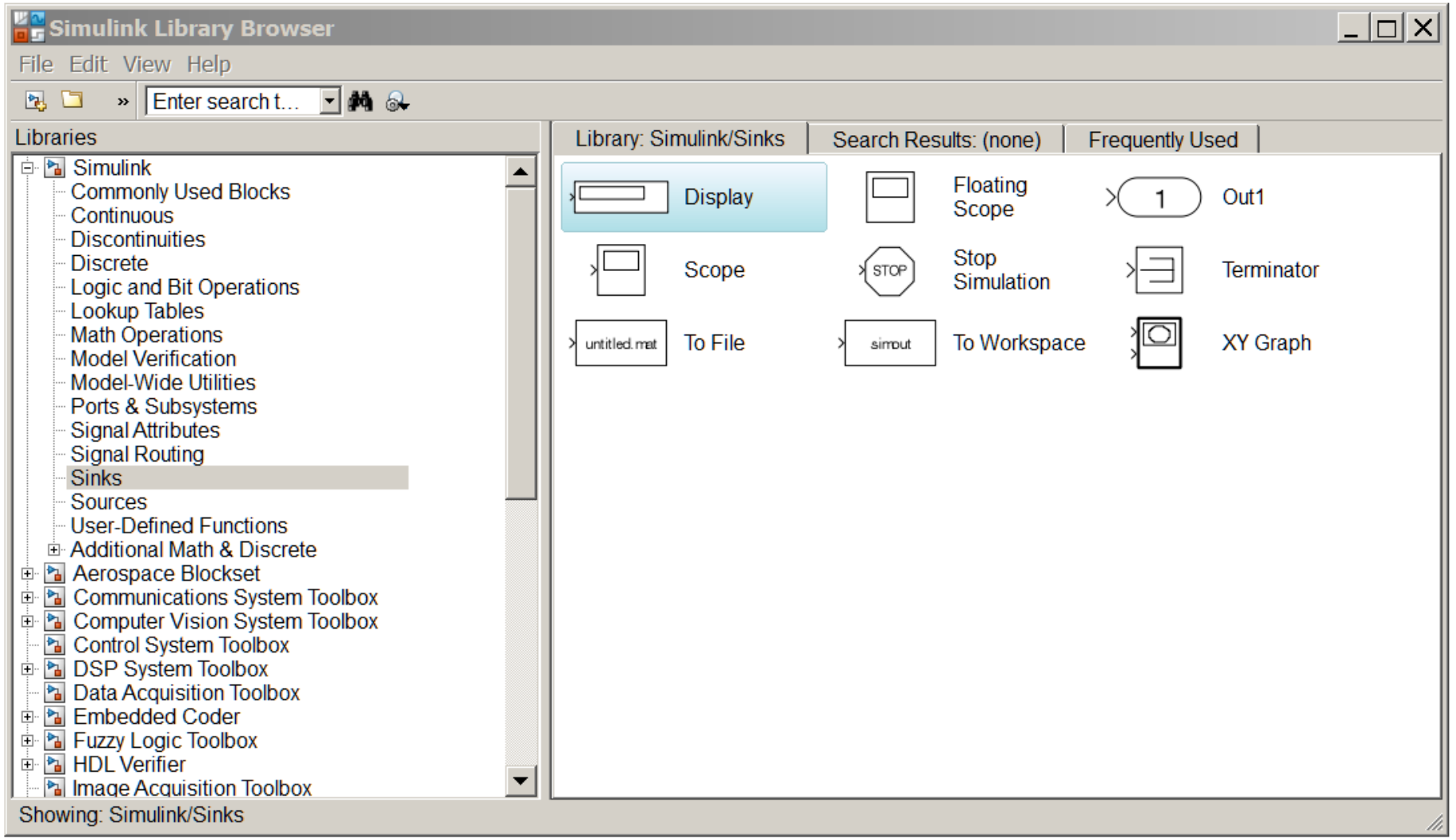
Simulink library – logic and bit operations



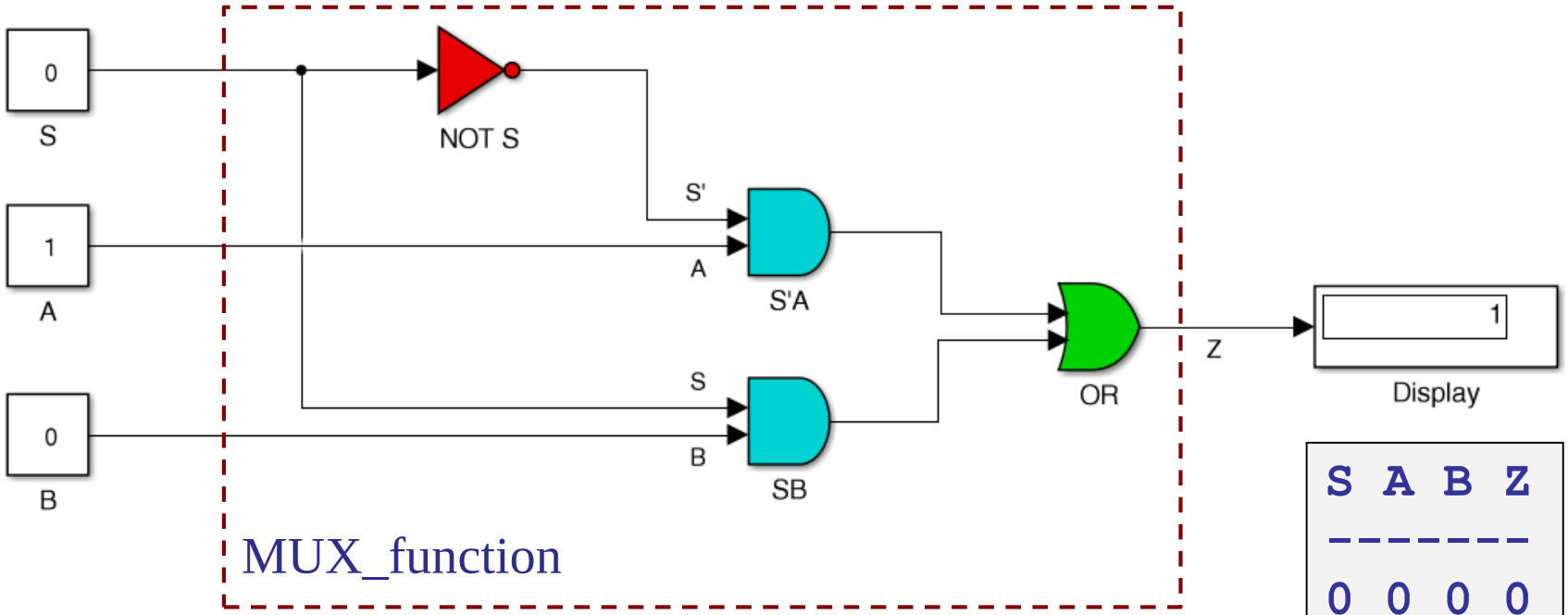
Simulink library – sources



Simulink library – sinks



MATLAB / Simulink implementation

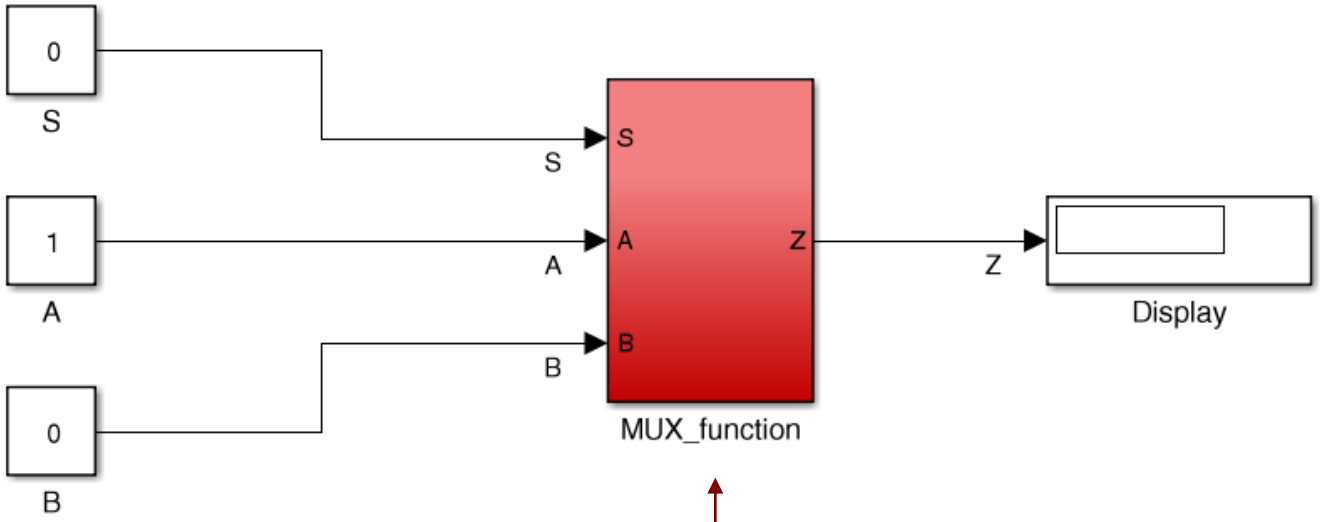


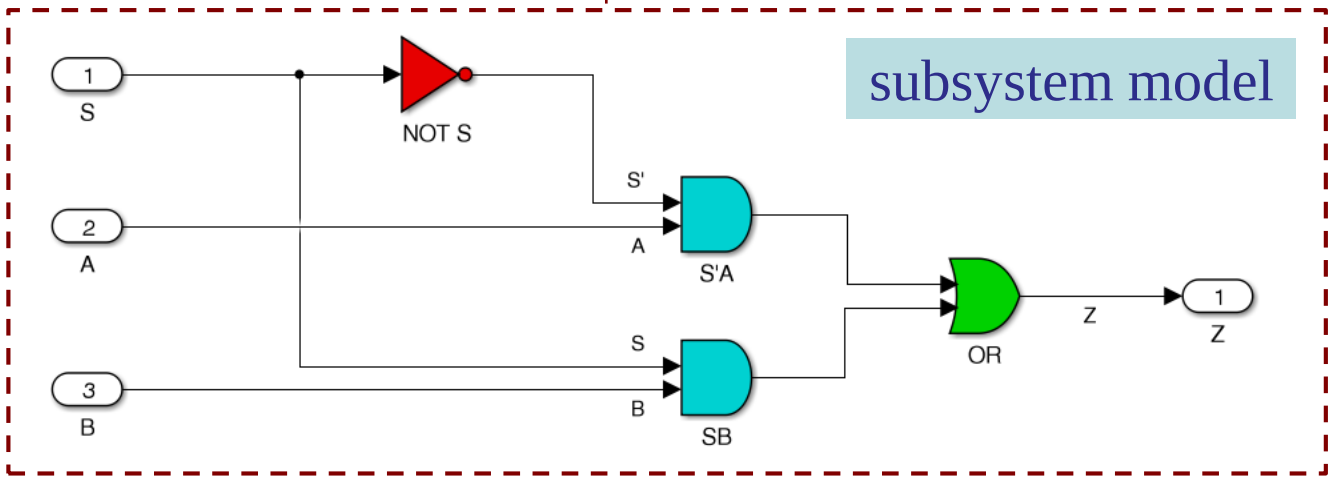
subsystem model can be exported into Verilog HDL, or VHDL, from Simulink

$$Z = S'A + SB$$

S	A	B	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

MATLAB / Simulink implementation

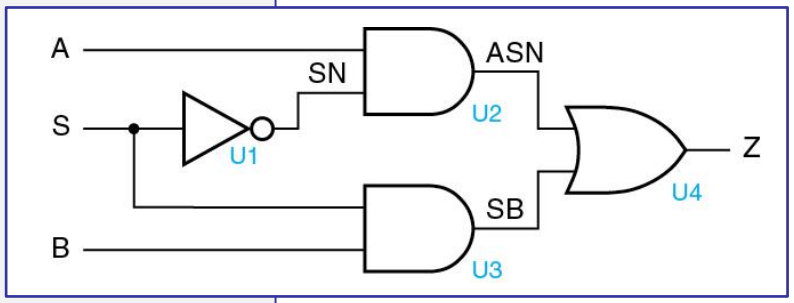




$$Z = S'A + SB$$

Verilog code generated by Simulink

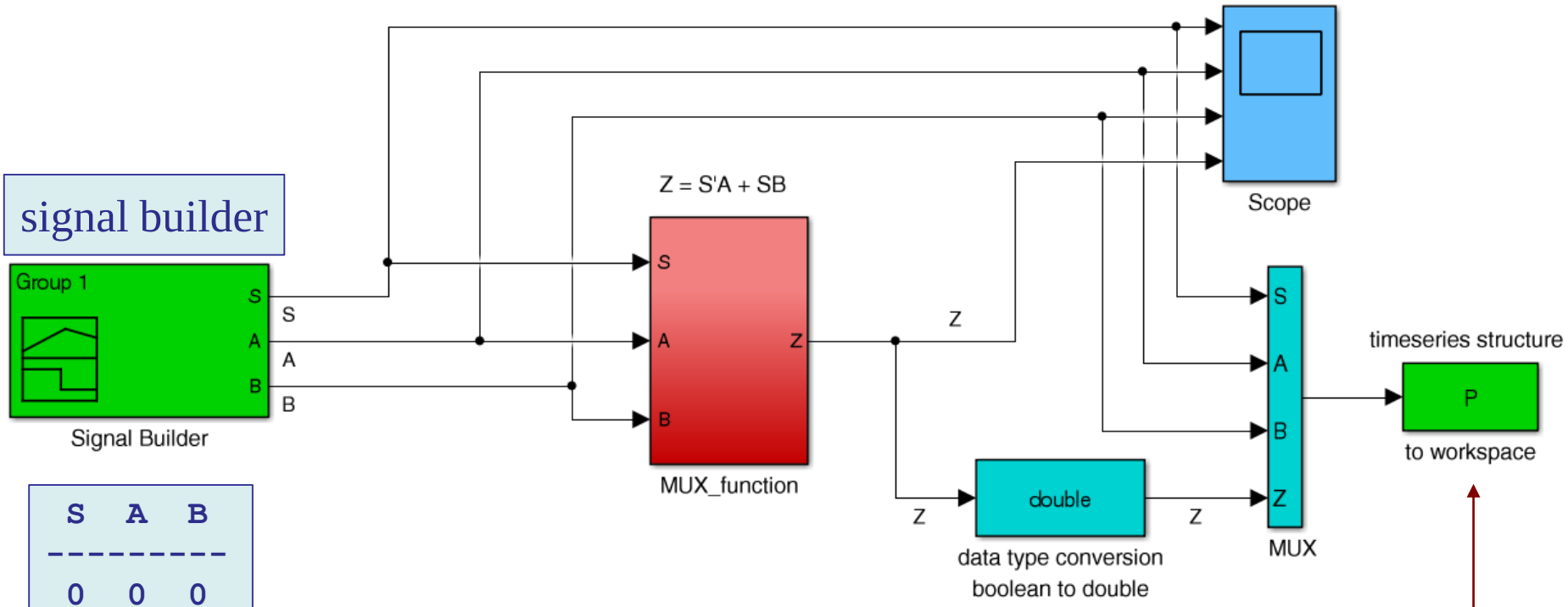
```
module MUX_function(S,A,B,Z);

  input   S, A, B;
  output  Z;

  wire S_1, S_2, S_3, S_4, S_A_out1;

  assign S_1 =  ~ S;
  assign S_2 = S_1;
  assign S_3 = S_2 & A;
  assign S_4 = S & B;
  assign S_A_out1 = S_3 | S_4;
  assign Z = S_A_out1;

endmodule
```



MATLAB / Simulink implementation

scope

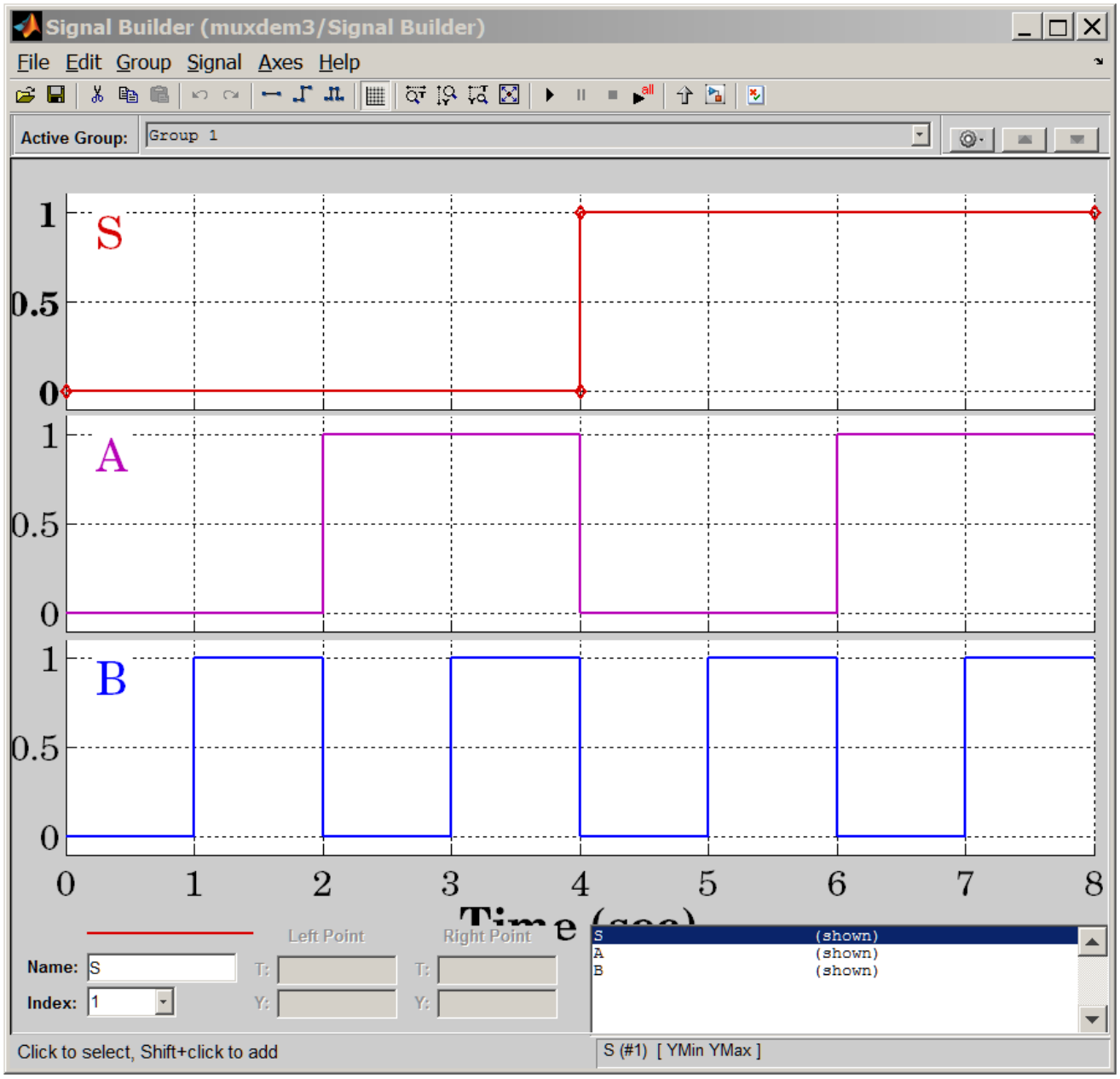
signal builder



S	A	B
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

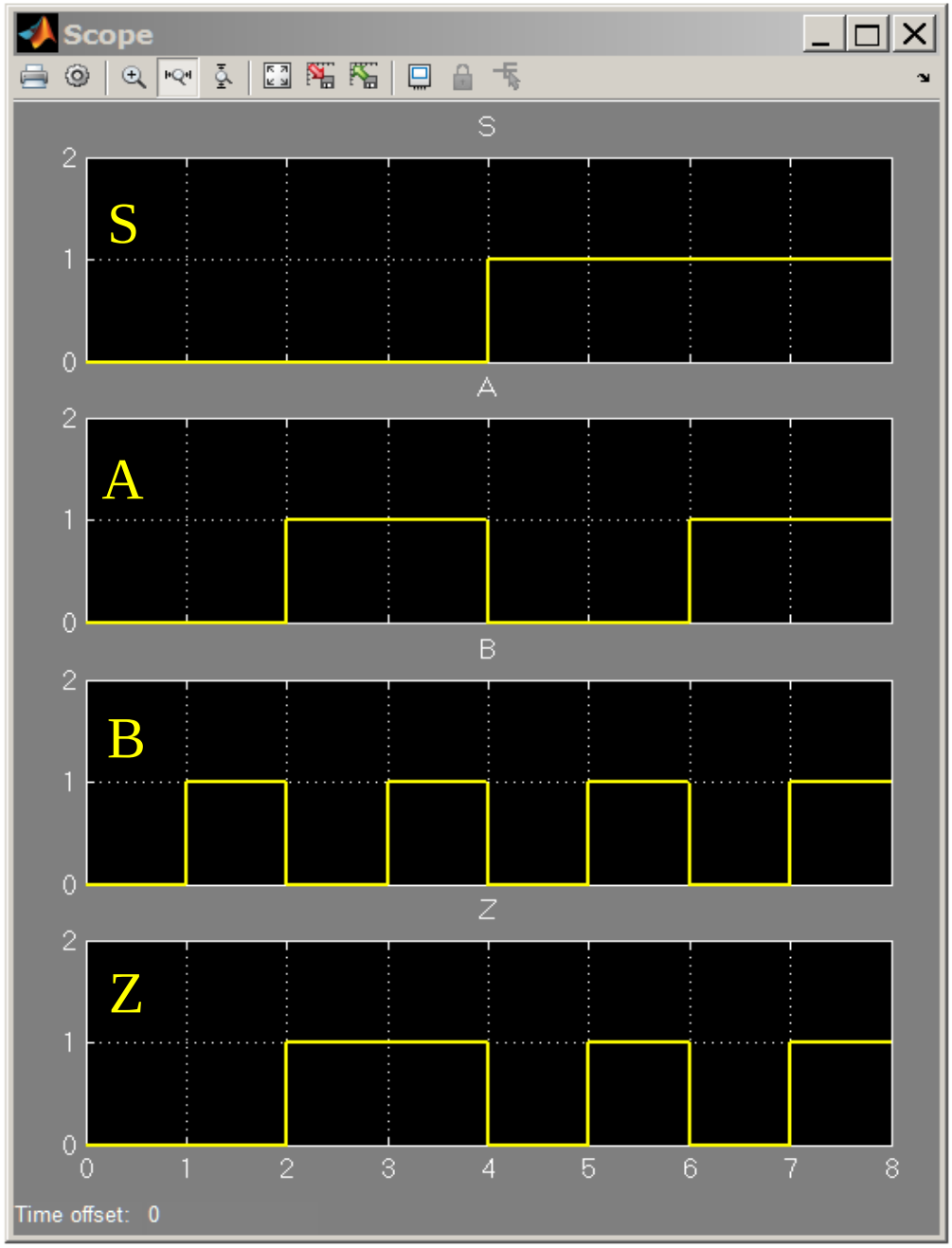
see **muxdem3.slx** on Canvas Files M-files folder

inputs and output saved as a timeseries structure into the workspace block, and brought into MATLAB for plotting



signal builder

S	A	B
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1



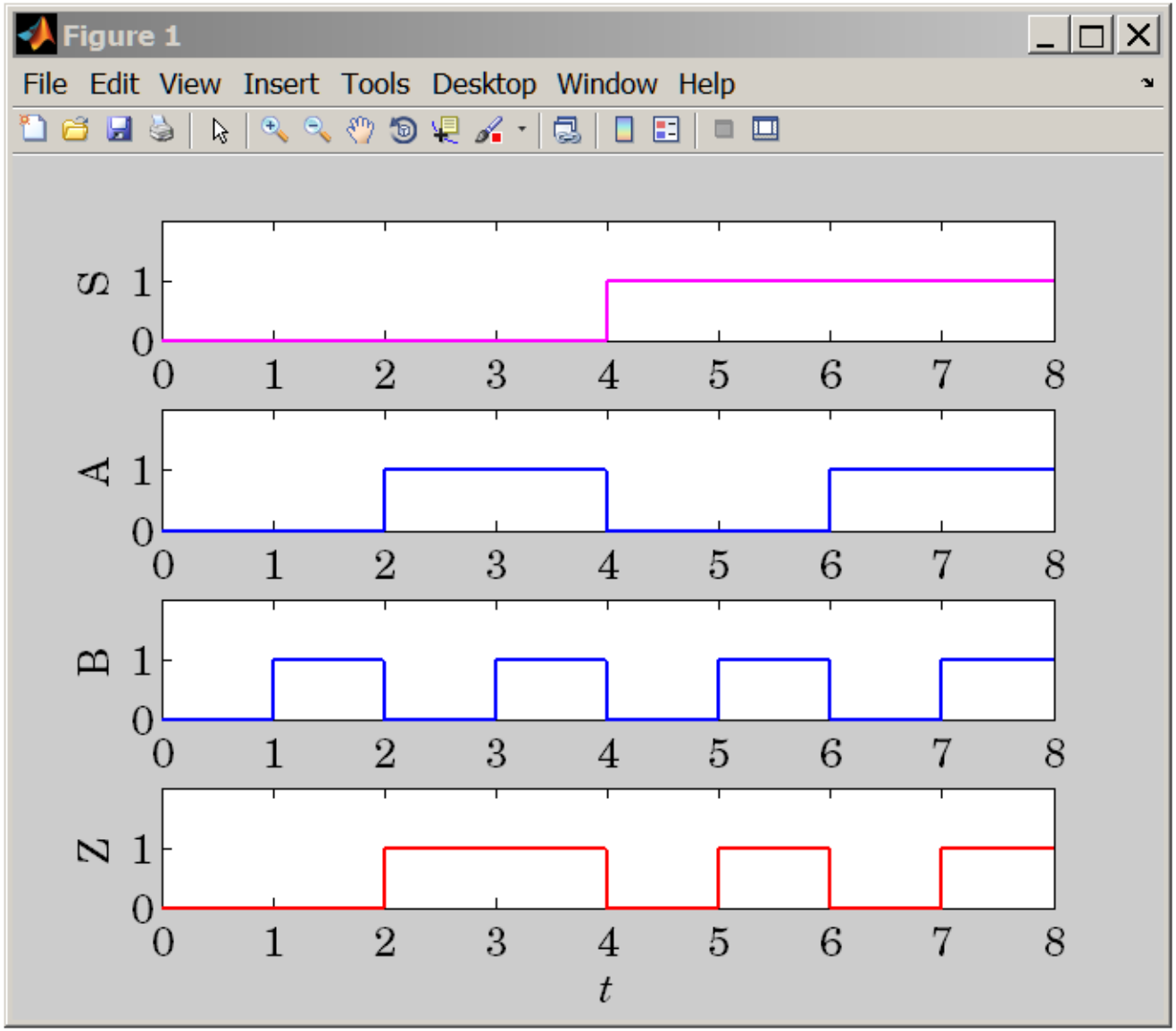
timing diagram from scope

S	A	B	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

```
% extract data from timeseries structure P

  t = P.time;          % equivalent to t = 0:0.01:8
  S = P.data(:,1);     % array lengths = 801 by default
  A = P.data(:,2);
  B = P.data(:,3);
  Z = P.data(:,4);

figure;
subplot(4,1,1); stairs(t,S,'m-');
ylabel('S'); xaxis(0,8,0:8); yaxis(0,2,0:2);

subplot(4,1,2); stairs(t,A,'b-');
ylabel('A'); xaxis(0,8,0:8); yaxis(0,2,0:2);

subplot(4,1,3); stairs(t,B,'b-');
ylabel('B'); xaxis(0,8,0:8); yaxis(0,2,0:2);

subplot(4,1,4); stairs(t,Z,'r-');
ylabel('Z'); xaxis(0,8,0:8); yaxis(0,2,0:2);
xlabel('\itt');

% xaxis() and yaxis() are on Canvas Resources;
```

staircase plot

timing diagram from timeseries structure P



S	A	B	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

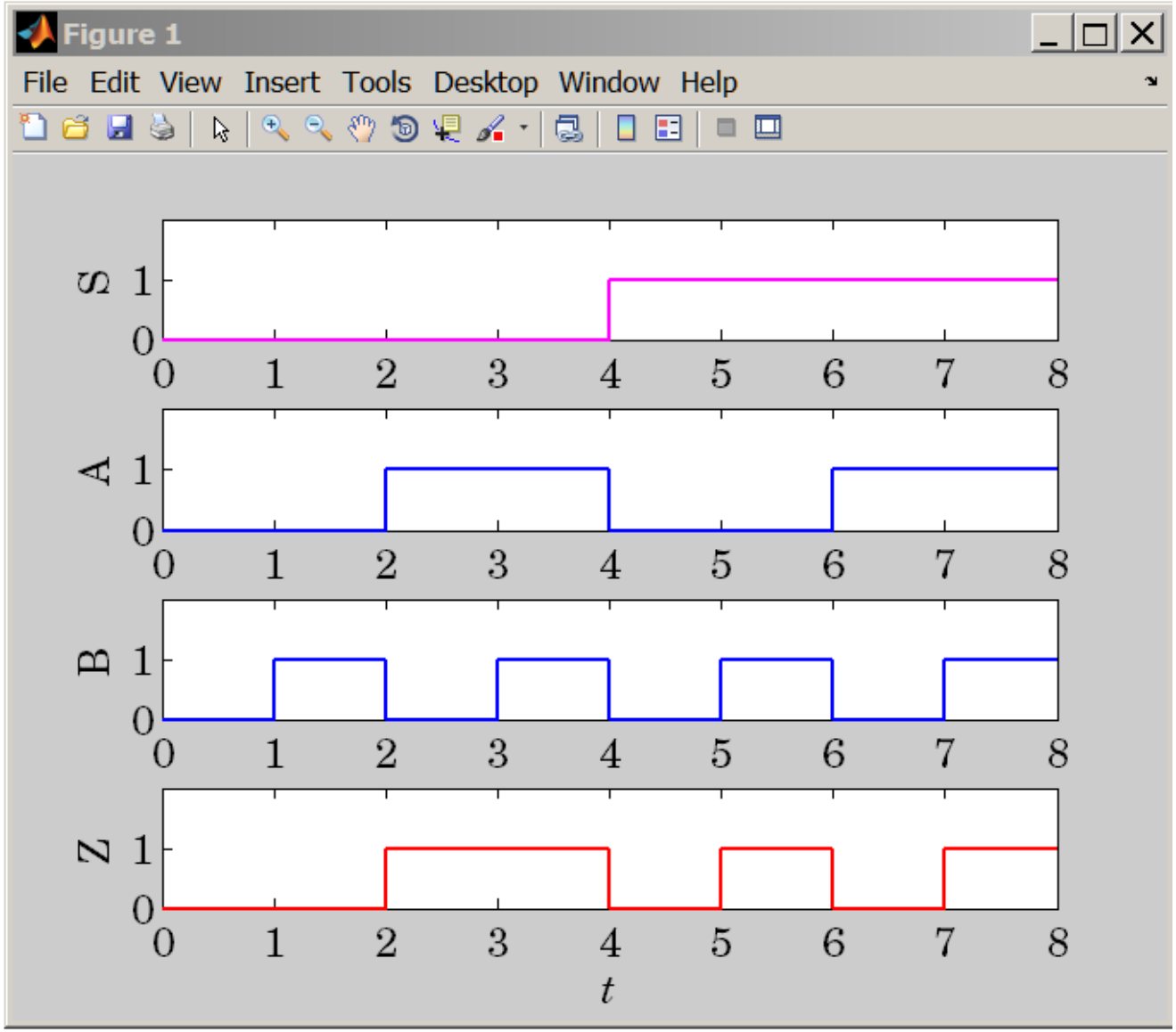
$$Z = S'A + SB$$

```
% calculate and plot timing diagram

t = (0:8);                  % last bit t=7 to t=8
[S,A,B] = a2d(0:7,3);       % 3-bit counter

S = [S; S(end,:)];          % extend last bit to t=8
A = [A; A(end,:)];          % extend last bit to t=8
B = [B; B(end,:)];          % extend last bit to t=8

Z = (~S & A) | (S & B);          % output

figure;
subplot(4,1,1); stairs(t,S,'m-');
ylabel('S'); xaxis(0,8,0:8); yaxis(0,2,0:1);

subplot(4,1,2); stairs(t,A,'b-');
ylabel('A'); xaxis(0,8,0:8); yaxis(0,2,0:1);

subplot(4,1,3); stairs(t,B,'b-');
ylabel('B'); xaxis(0,8,0:8); yaxis(0,2,0:1);

subplot(4,1,4); stairs(t,Z,'r-');
ylabel('Z'); xaxis(0,8,0:8); yaxis(0,2,0:1);
xlabel('\itt');
```

&, | are vectorized operations

staircase plot

timing diagram from plain MATLAB

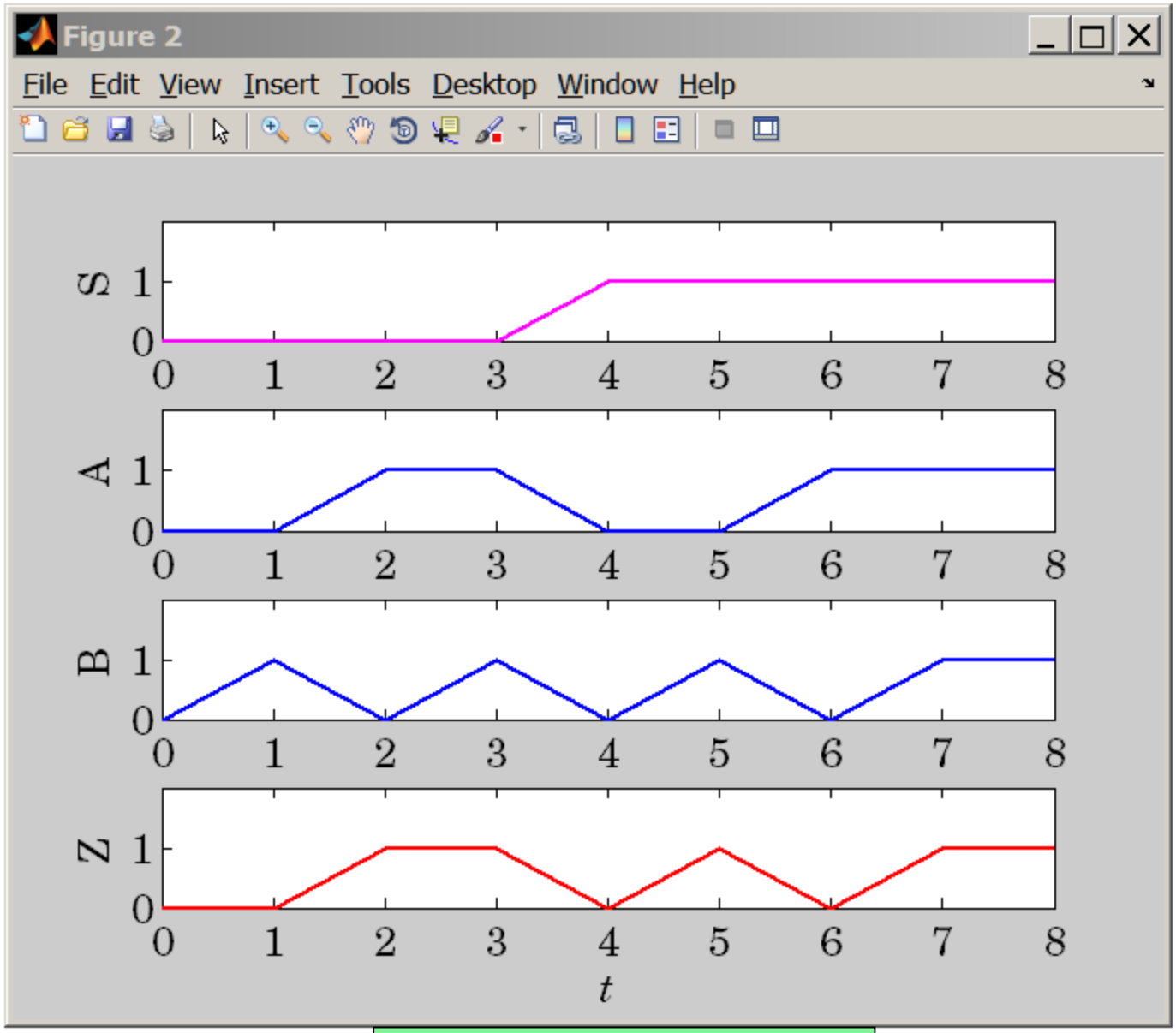


staircase plot

S	A	B	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$Z = S'A + SB$$

using plot( ) instead of stairs( )



note: here, array lengths = 9

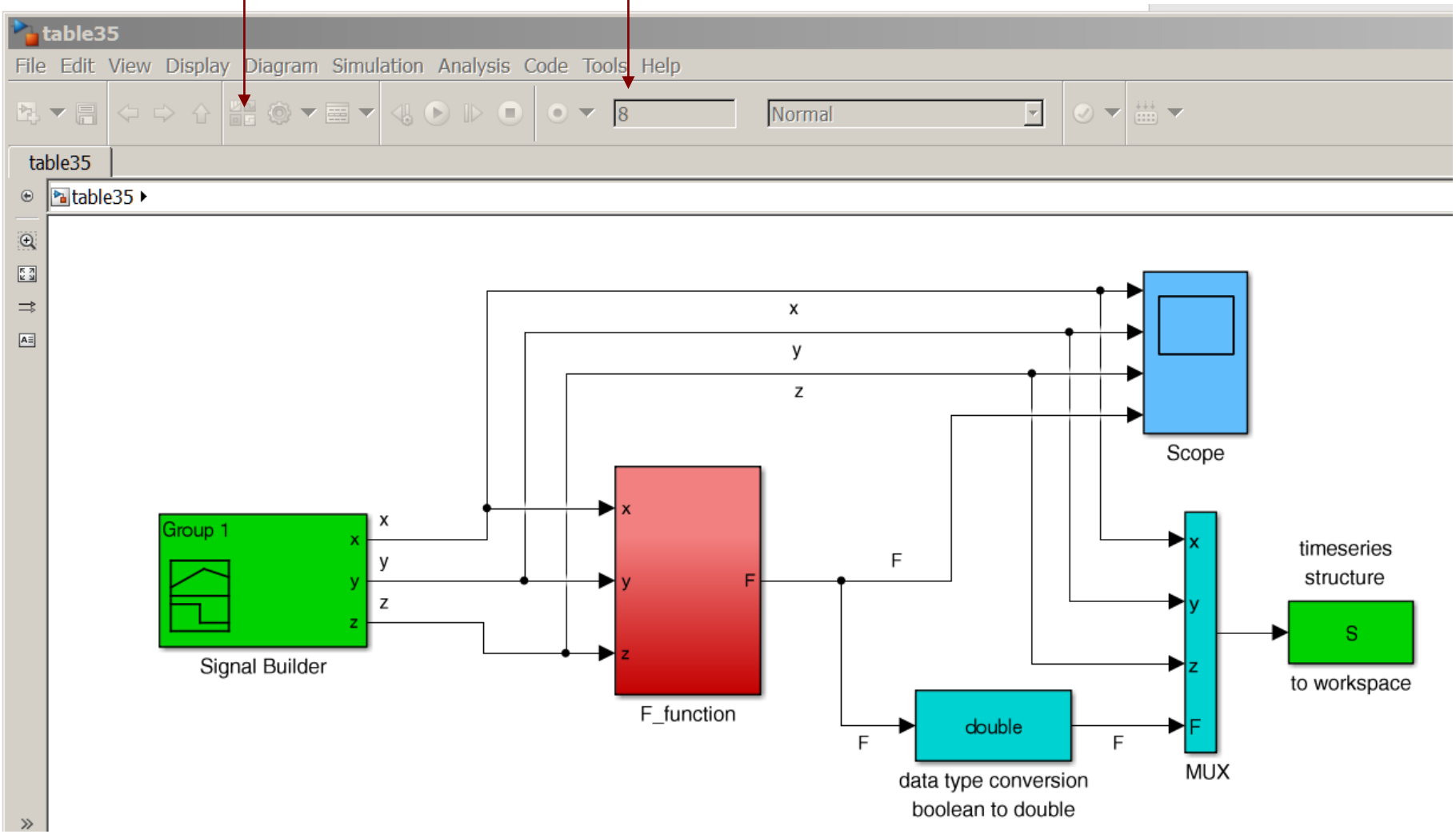
Z = S'A + S B

Additional notes on Simulink

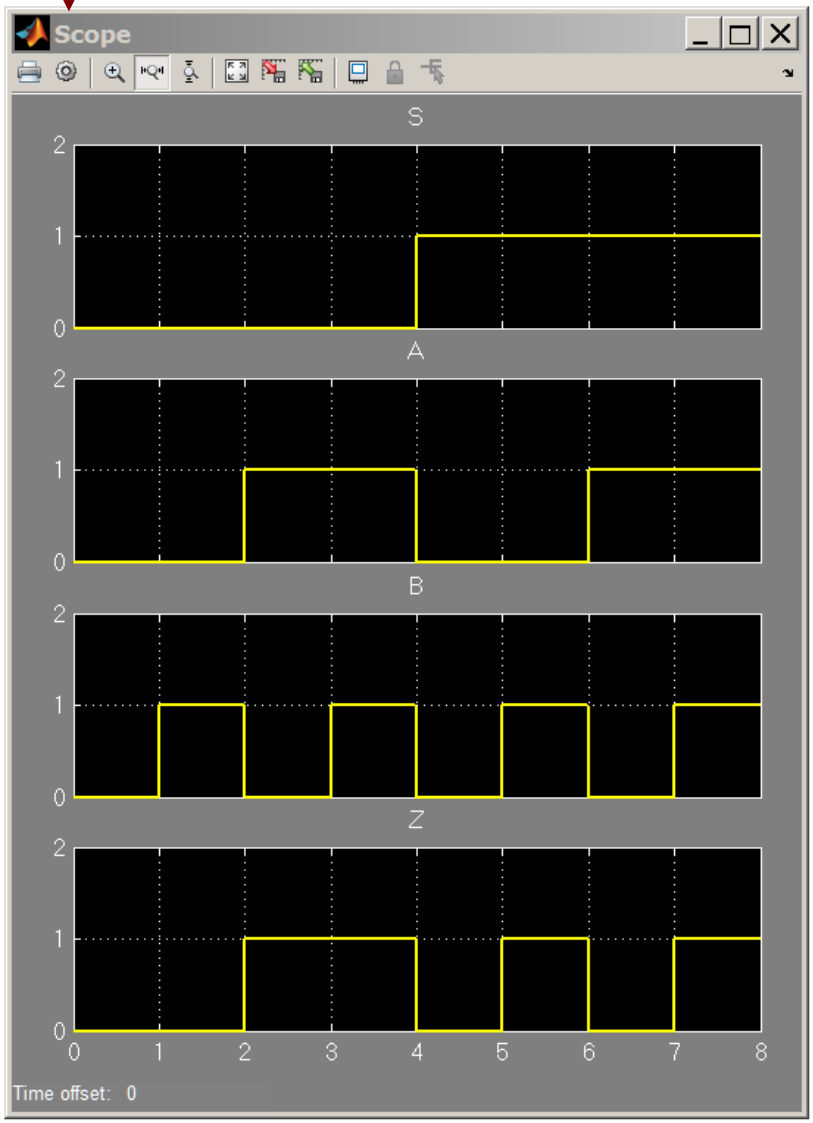
click here to open Simulink Library

set simulation duration to 8 time units

see **muxdem3.slx** on Canvas Files M-files folder

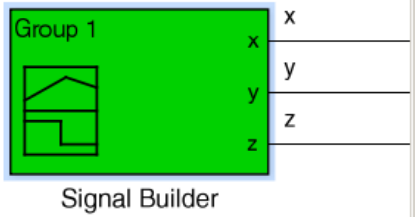


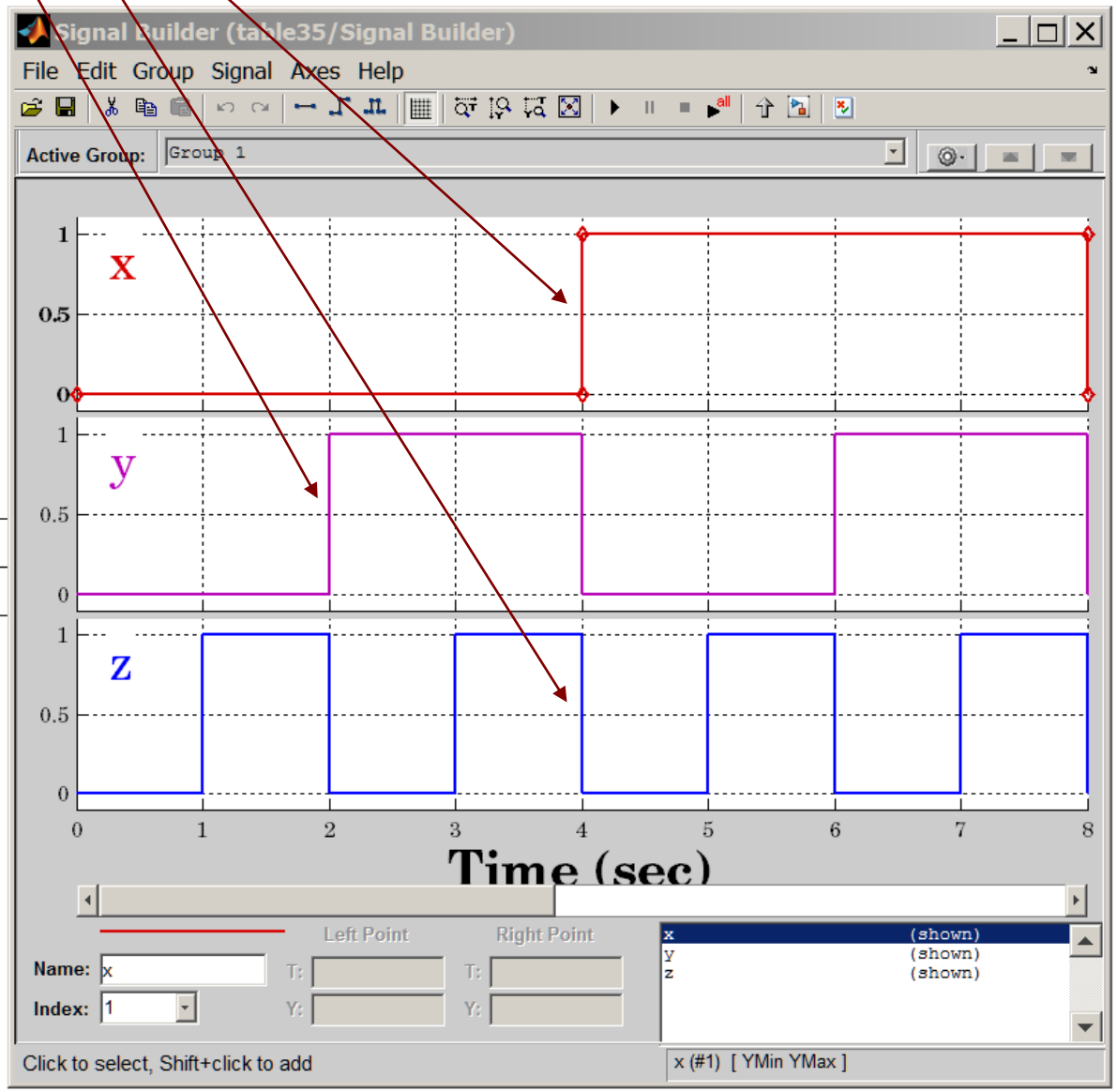
click here to open scope parameters and select 4 axes (for X,Y,X,F) and time range of 8 units



drag the signal edges to change the shape of the signals

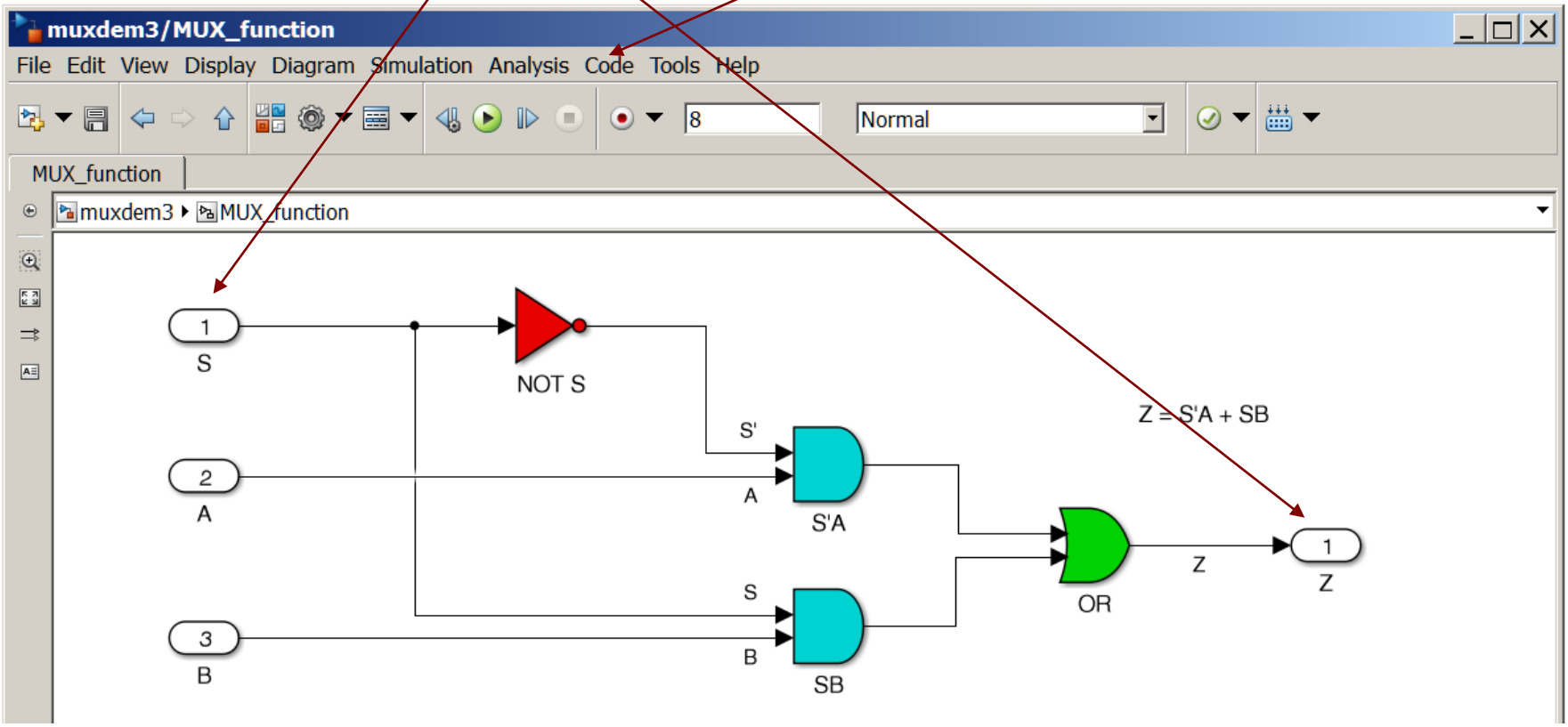
double-click on the Signal Builder block to view the signals X,Y,Z over the time range of 8 units





before exporting into Verilog, save the subfunction into a separate SLX Simulink file, then set the data types of the input/output ports to Boolean

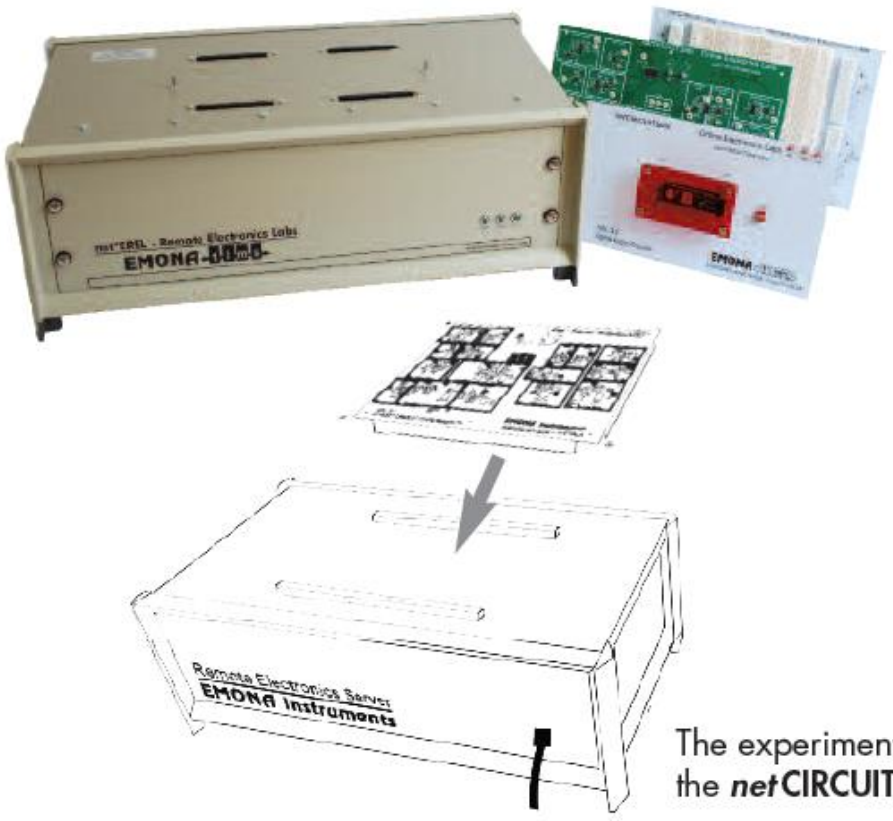
select Code, then HDL code, Options, and Verilog



to set the data types, double-click on each port, select signal attributes, and set data type to Boolean

Emona netCIRCUITlabs board

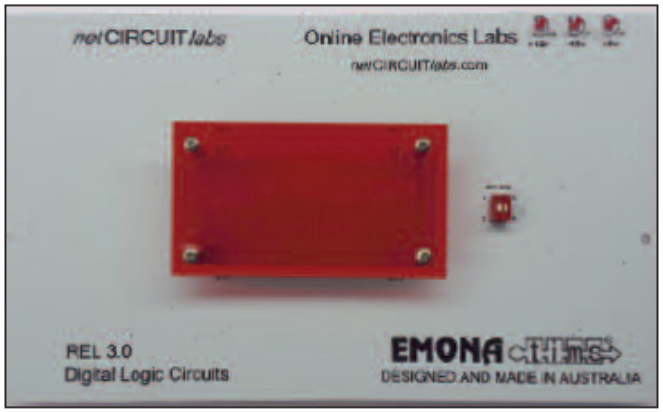
netCIRCUITlabs CONTROL UNIT with MULTIPLE PLUG-IN BOARDS



The experiments boards plugs into the ***net*CIRCUIT*labs*** Control Unit

- The netCIRCUITlabs Control Unit, located in your lab or office, and will accept any netCIRCUITlabs Lab Experiment board.
- Fast and easy implementation. No software to install and no setting up required.
- Secure access for professor to all ADMIN functions including student records and tracking.

REL 3.0 DIGITAL LOGIC board - student wired experiments



All the logic functions and connections are implemented in an FPGA.

REL3.0 FUNCTIONALITY & EXPERIMENT CAPABILITIES

SIGNAL SOURCES:

- *HI/LO Logic Switches x 8*
- *8 bit Binary Counter*
- *4 bit Gray Counter*
- *4 bit Johnson Counter*

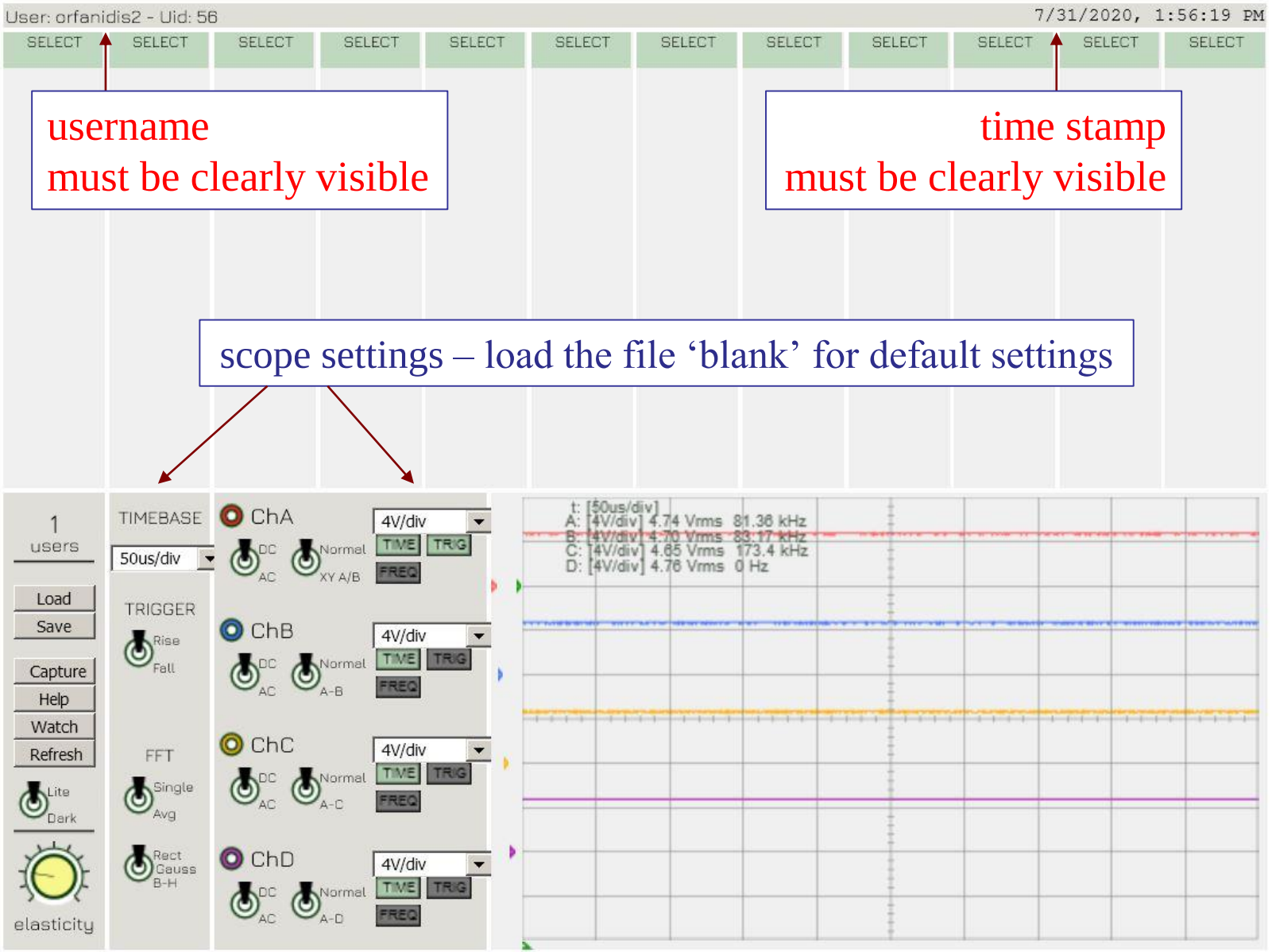
OVER 60 GATES & FLIP-FLOPS:

2, 3 & 4-input OR gates
X-OR gates
2, 3 & 4-input AND gates
Inverters
S/R, D & J/K Flip-Flops,
Inverters
Finite State Machines

STUDY:

- *Boolean logic and algebra*
- *Combinatorial circuits*
- *Truth tables*
- *Karnaugh Maps*
- *Quine-McCluskey method*
- *Designing Synch & Asynch sequential circuits*
- *Flip flops*
- *State diagrams*
- *Design of FSM*
- *Registers, Counters, Multiplexers, Encoders etc*
- *Introduction to HDL (Verilog)*

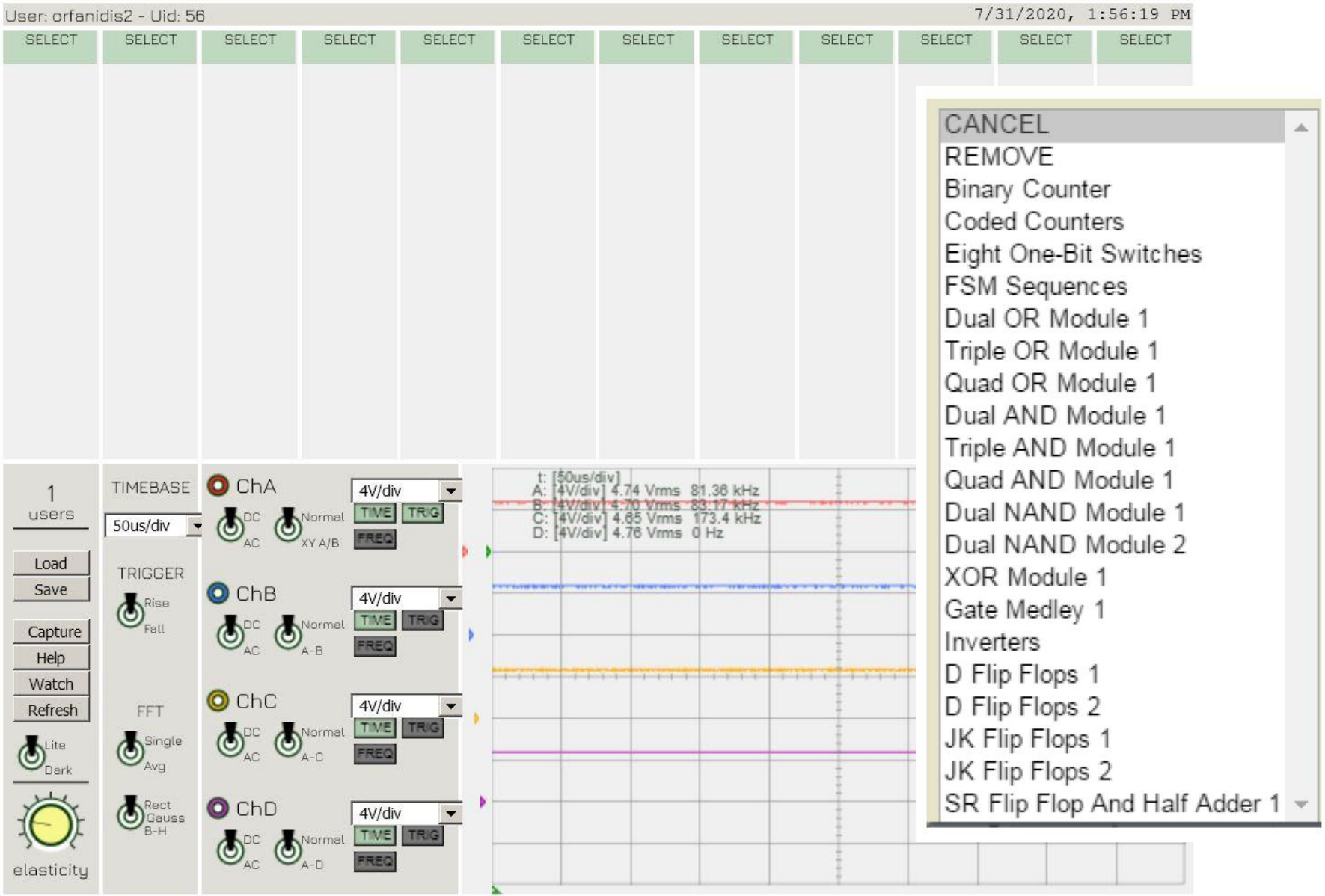
Multiplexer function implemented on the Emona board



user manual on Canvas

http://ece-emona1.engr.rutgers.edu/

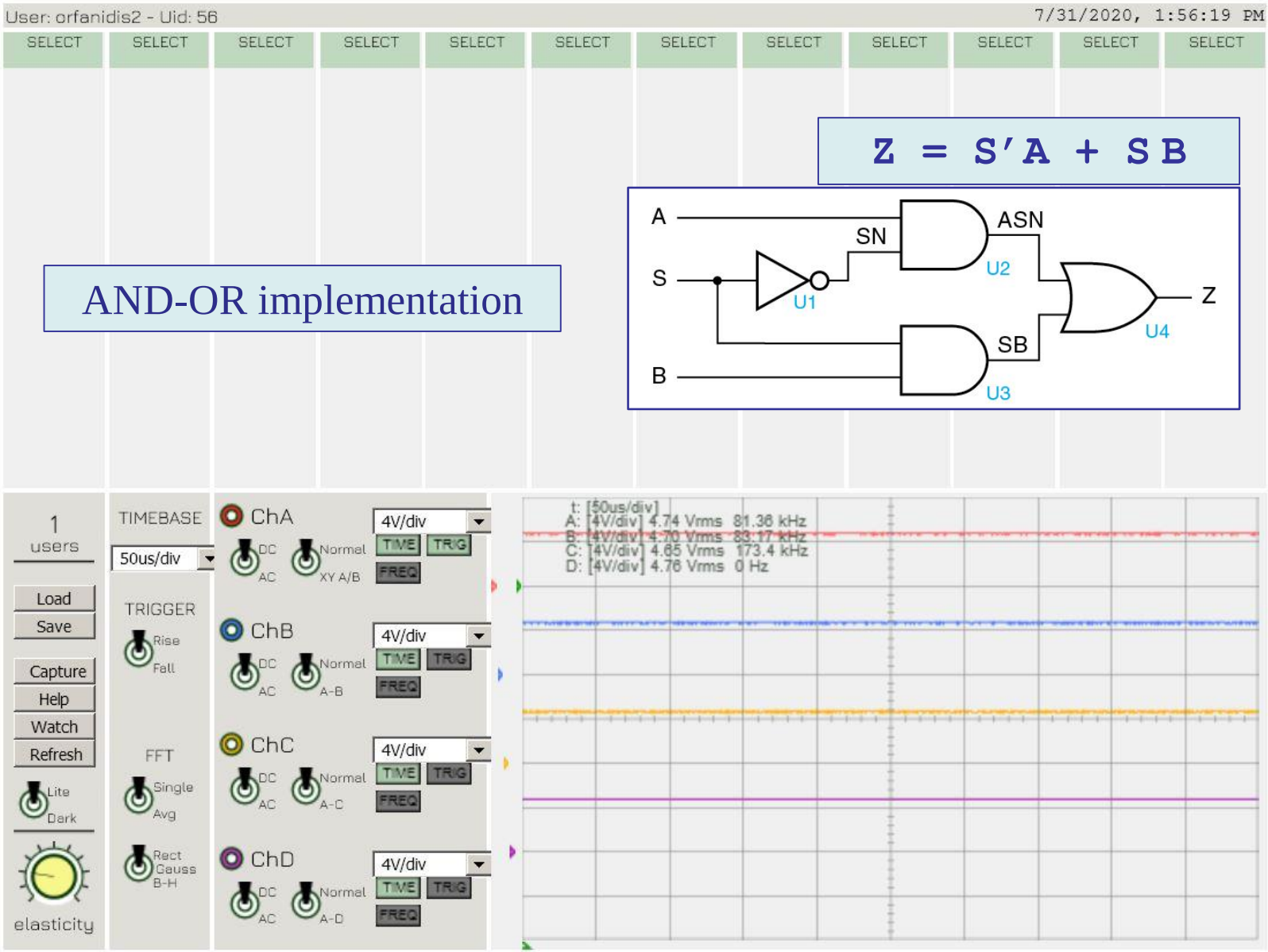
Multiplexer function implemented on the Emona board



user manual on Canvas

http://ece-emona1.engr.rutgers.edu/

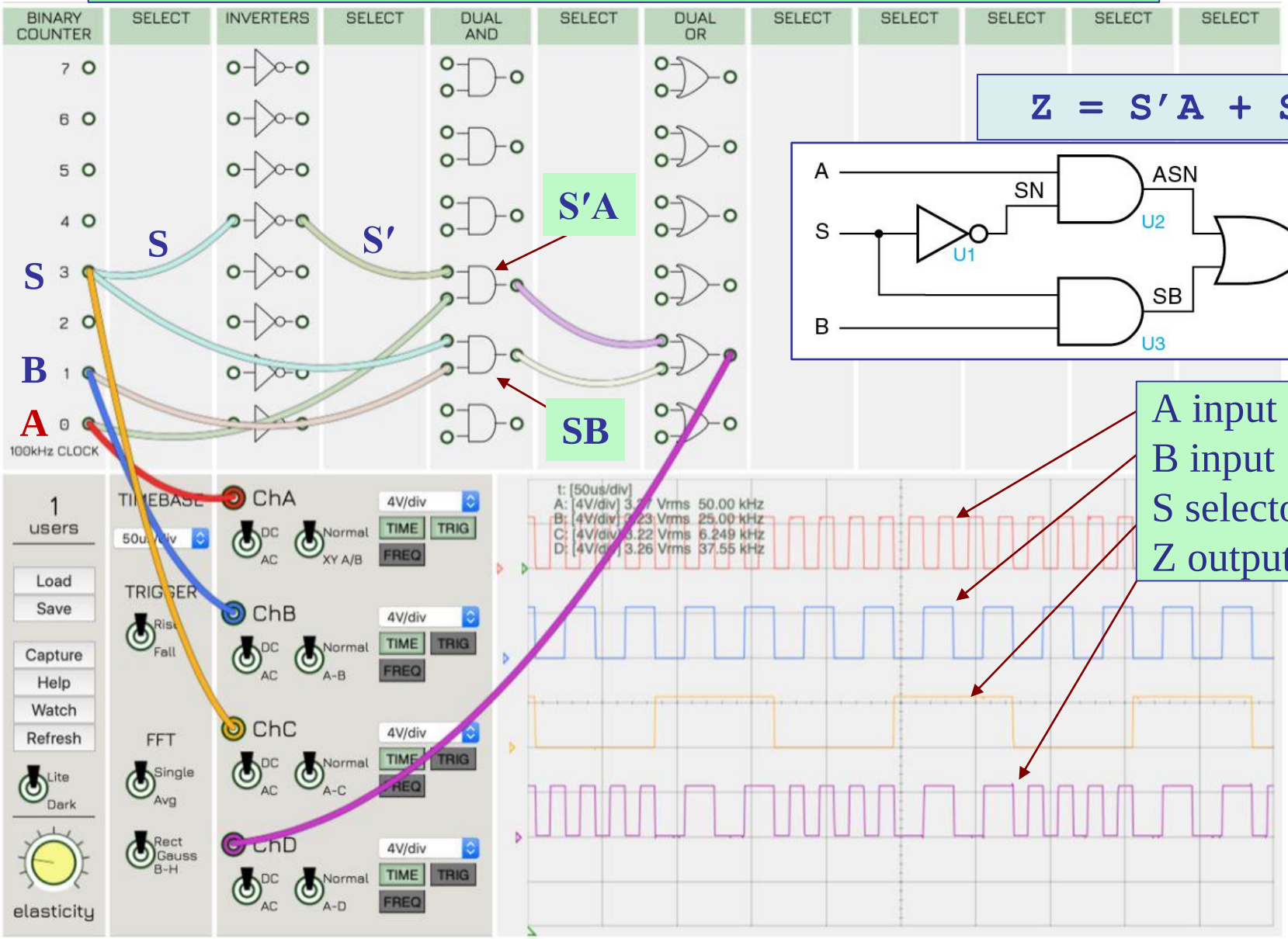
Multiplexer function implemented on the Emona board



user manual on Canvas

http://ece-emona1.engr.rutgers.edu/

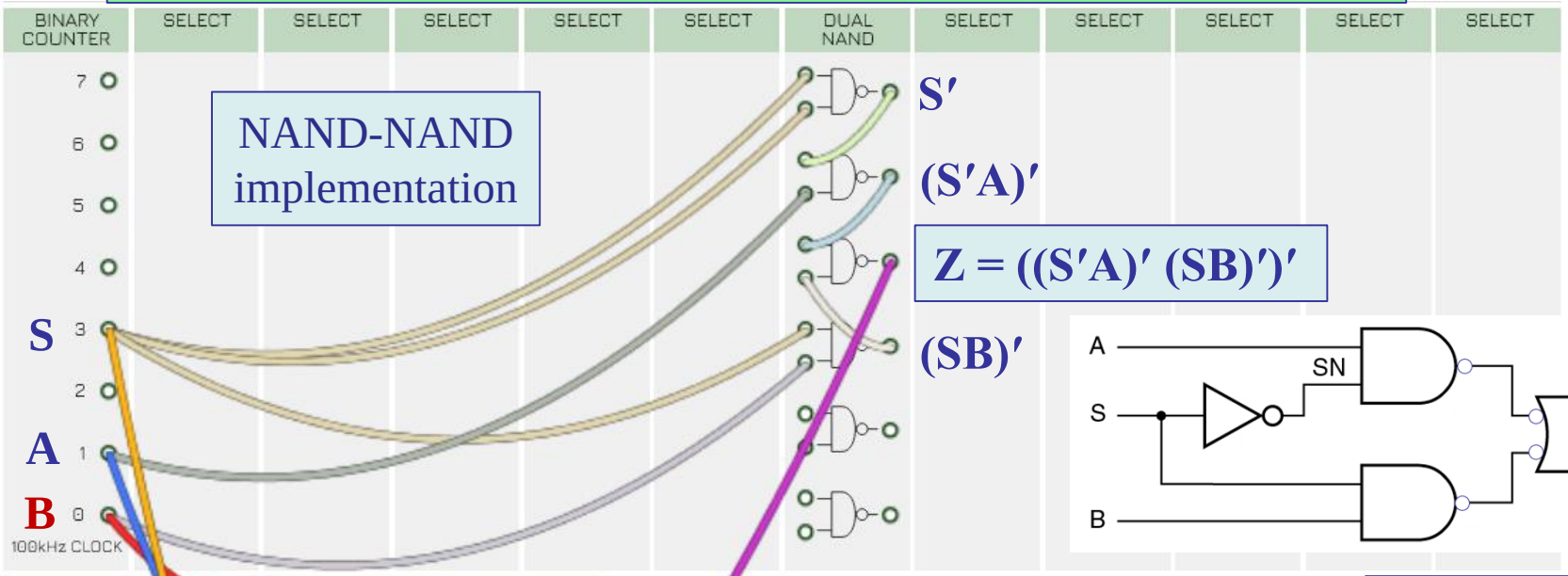
Multiplexer function implemented on the Emona board

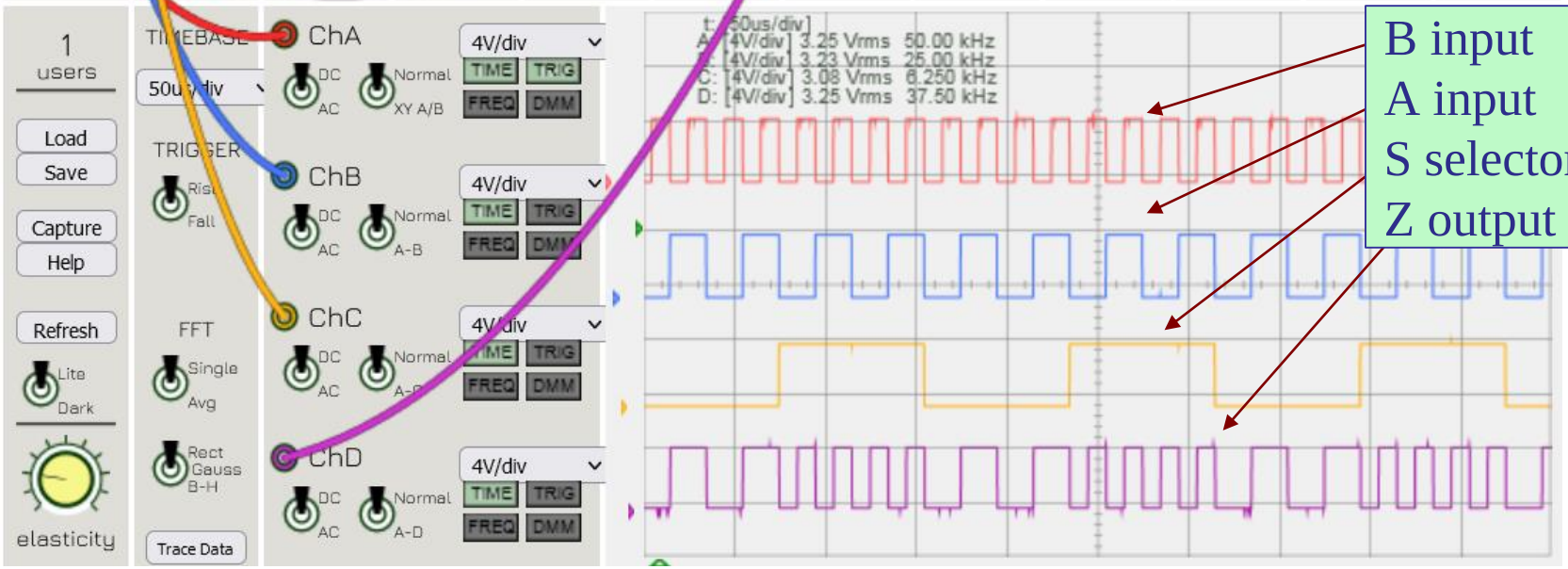


user manual on Canvas

http://ece-emona1.engr.rutgers.edu/

Multiplexer function implemented on the Emona board



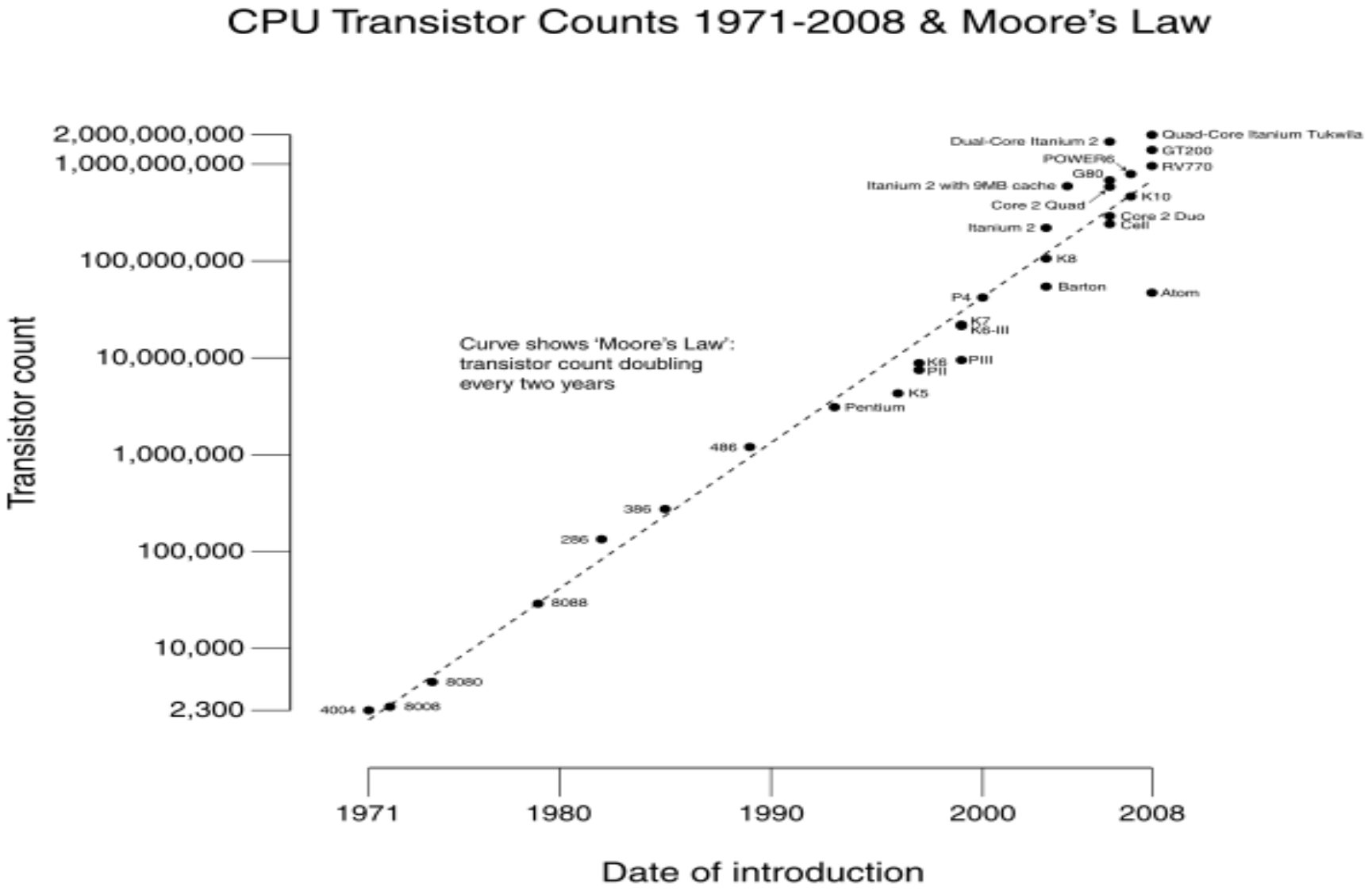


B input
A input
S selector
Z output

user manual on Canvas

http://ece-emona1.engr.rutgers.edu/

Moore's law – data fitting



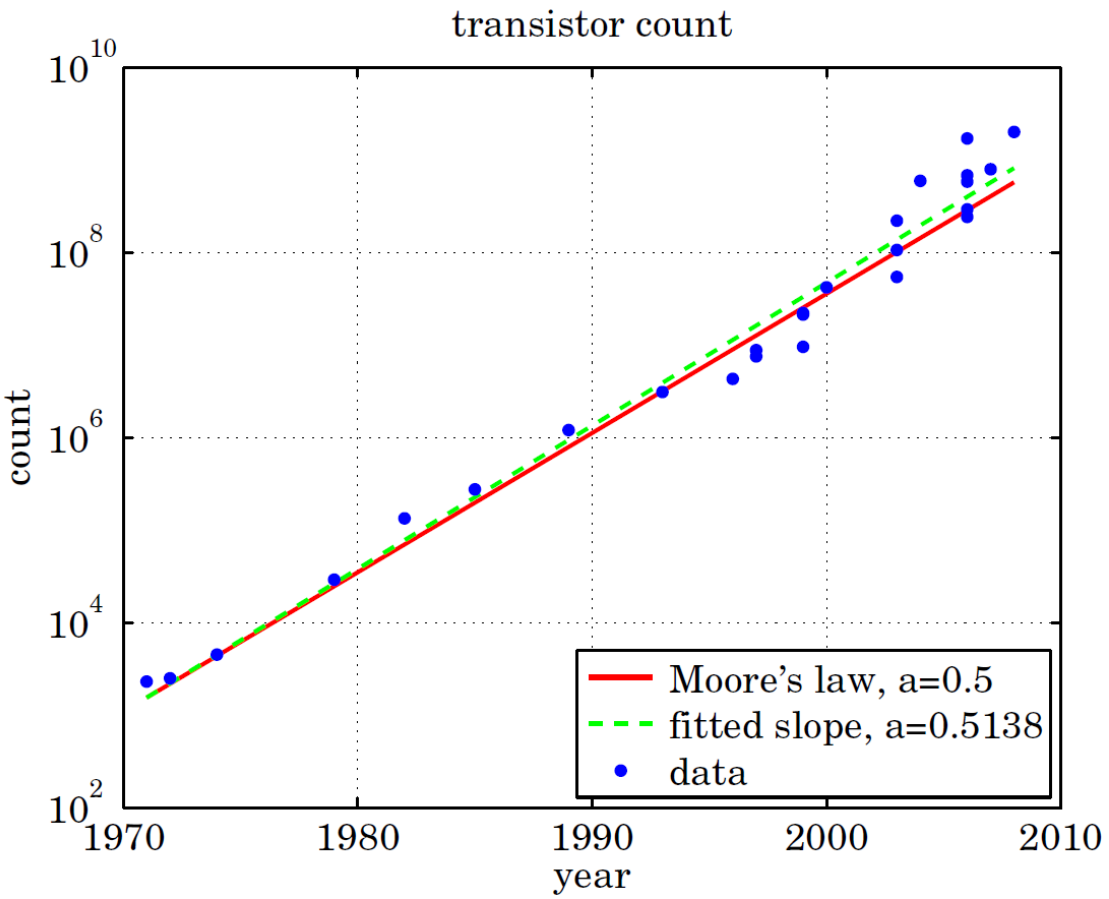
yi	ti
2.300e+003	1971
2.500e+003	1972
4.500e+003	1974
2.900e+004	1979
1.340e+005	1982
2.750e+005	1985
1.200e+006	1989
3.100e+006	1993
4.300e+006	1996
7.500e+006	1997
8.800e+006	1997
9.500e+006	1999
2.130e+007	1999
2.200e+007	1999
4.200e+007	2000
5.430e+007	2003
1.059e+008	2003
2.200e+008	2003
5.920e+008	2004
2.410e+008	2006
2.910e+008	2006
5.820e+008	2006
6.810e+008	2006
7.890e+008	2007
1.700e+009	2006
2.000e+009	2008

Moore's law

fitted model

$$f(t) = b\,2^{a(t-t_1)}$$

$$\log_2 f(t) = \log_2 b + a(t - t_1)$$



```
Y = load('transistor_count.dat');  % MATLAB textbook

y = Y(:,1);  t = Y(:,2);

t1 = t(1);   % t1 = 1971

p = polyfit(t-t1, log2(y), 1);

% p =
%    0.5138   10.5889   % b = 2^p(2) = 1.5402e+003

f = 2.^(polyval(p,t-t1));

semilogy(t,f,'r-', t,y,'b.', 'markersize',18)
```

should be 0.5 according to Moore's law

fitted model:

```
f(t) = b * 2.^(a*(t-t1)) = 2.^(a*(t-t1)+log2(b));

% a = p(1), log2(b) = p(2) --> b = 2^(p(2))

% a = 0.5139,  b = 1.5402e+03
```