

Implementing combinational and sequential logic

We've used combinational logic, registers, and memory as building blocks in our digital circuits

Specified the behavior of combinational logic as truth table and in Verilog

- now we'll implement arbitrary combinational logic blocks with transistors

Specified the behavior of registers and memory

- now we'll implement a register with combinational logic and feedback

Overview of implementing combinational logic

- implement simple combinational logic functions (AND, OR, NOT)
- combine these to implement arbitrary combinational logic blocks

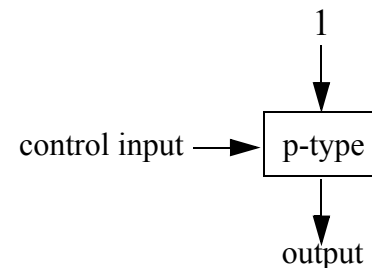
Transistors

Transistor is a controlled switch

- switch can be connected (source data flows to output data) or disconnected (source data does not flow to output data)
- state of the switch depends on a control input

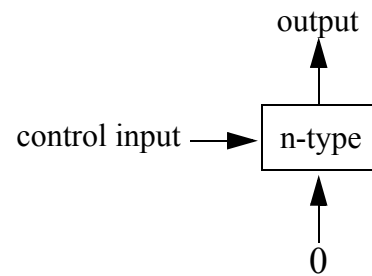
p-type transistor

- if control input = 0, then output is connected to source (1)
- if control input = 1, then output is disconnected



n-type transistor

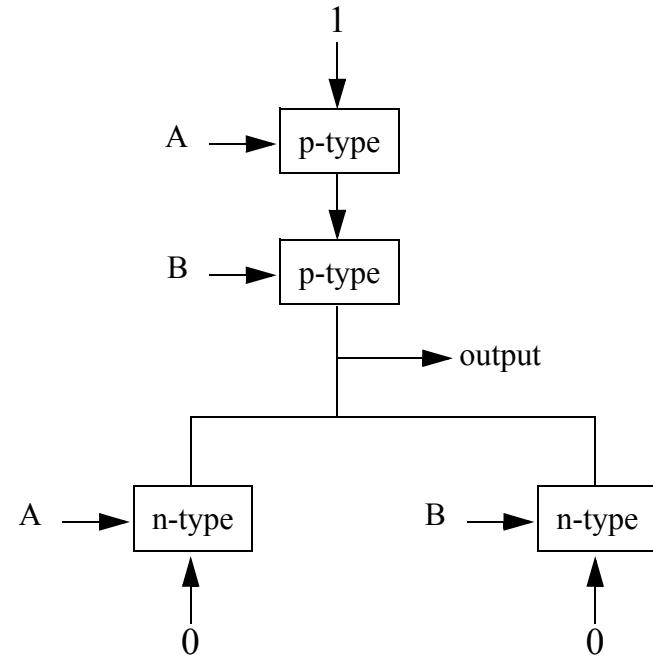
- if control input = 0, then output is disconnected
- if control input = 1, then output is connected to source (0)



What do these circuits do?



A	out
0	
1	



A	B	output
0	0	
0	1	
1	0	
1	1	

Implementing OR with transistors

How to implement OR of 3 inputs?

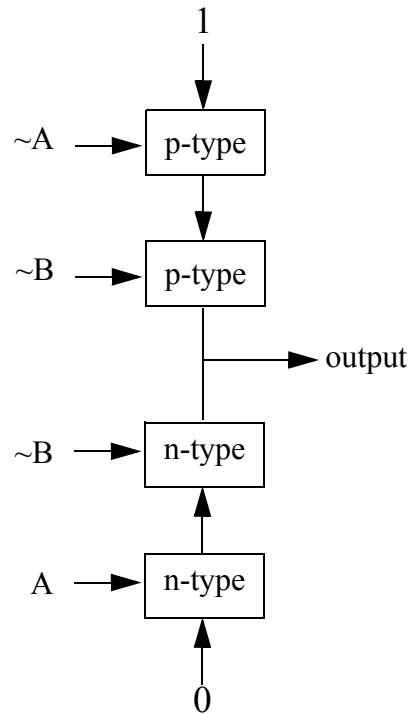
A	B	C	output
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Implementing NAND with transistors

A	B	output
0	0	1
0	1	1
1	0	1
1	1	0

Implementing AND with transistors

What does this circuit do?



Implementing combinational logic with AND, OR, NOT

Two inputs, one output row is 1

A	B	output
0	0	0
0	1	1
1	0	0
1	1	0

$\sim A$	B	output
1	0	0
1	1	1
0	0	0
0	1	0

Multiple inputs, one output row is 1

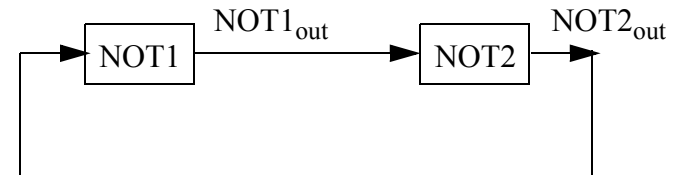
A	B	C	output
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Multiple inputs, multiple output rows are 1

A	B	C	output
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

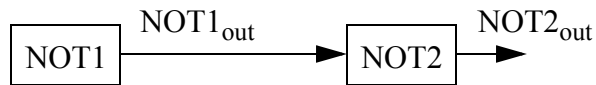
Implementing a 1-bit register

What does this circuit do?



Setting the stored value to 1

Remember that OR of anything with 1 produces 1



Setting SET=1 will

- change the output of the OR gate to 1, which will
- change NOT1_{out} to 0, which will
- change NOT2_{out} to 1
- after this, SET can go back to 0, and the value of NOT2_{out} will remain 1

Clearing the stored value to 0



Setting CLEAR=1 will

- change output of OR2 to 1, which will
- change NOT2_{out} to 0, which will
- change NOT1_{out} to 1
- after this, CLEAR can go back to 0, and the value of NOT2_{out} will remain 0

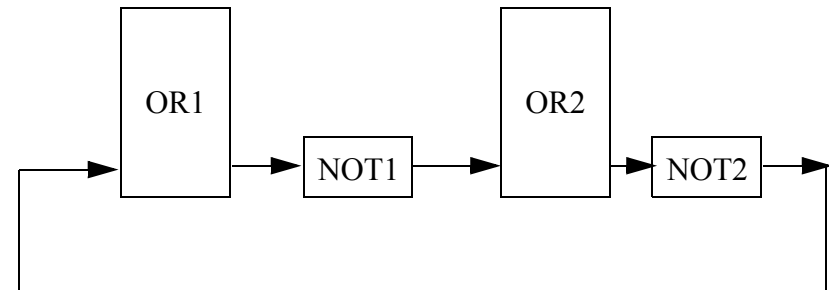
How to use this to store a value?

What if we tried to use this to store an input value In ?

- $SET = In$
- $CLEAR = \sim In$

Control when the circuit pays attention to SET and CLEAR

Remember that AND of anything with 0 produces 0



When $WRITE=0$

- SET and $CLEAR$ will both be 0, i.e. ignore IN and $\sim IN$

When $WRITE=1$

- $IN=1$ ($\sim IN=0$) will set $NOT2_{out}$ to 1
- $IN=0$ ($\sim IN=1$) will clear $NOT2_{out}$ to 0

Adding a clock

Edge-triggered register

The previous circuit is called a LATCH

- writes the new value whenever $CLOCK=1$ and $WRITE=1$

How to make an edge-triggered register (called a FLIP-FLOP) from a LATCH?

General-purpose computers

What would it take to turn the E100 on the DE2 into a standalone, general-purpose computer?

The E100 on the DE2 is quite capable

- execute general-purpose algorithms, by composing them from simpler operations, e.g. arithmetic average (with addition, division), square root (with successive approximation)
- input devices
 - keyboard
 - mouse
 - DPDT switches
 - microphone
 - SD card
 - serial port (receive)
- output devices
 - VGA
 - LCD module
 - speaker
 - LEDs
 - serial port (send)

What capabilities does a normal computer have that the E100 lacks?

What if we added these capabilities? Is this now a standalone, general-purpose computer?

Loading and running new programs

How could we load and run a new program, e.g. from disk?

- currently we load memory using Quartus' in-system memory content editor, but this won't work for a standalone DE2

When you look at a memory image file, how can you tell what is an instruction, and what is data (i.e. a variable)?

From Lab 5 exercise:

```
mem[12] = 1
mem[13] = 41
mem[14] = 28
mem[15] = 29
mem[16] = 0
mem[17] = 0
mem[18] = 0
mem[19] = 0
```