Standard Cell Library

PersonalCompute.Net

About

This document is part of the **"Computation Structures"** course, available at https://PersonalCompute.net/resources/computation-structures.

The objective of this course is to provide a solid foundation on the inner workings of computers, and how to use them efficiently. Practically, it tries to answer the question "*Why is my computer working like this?*" (where "like this" can mean "slow", "fast", "efficient" or "intermittently freezing").

Its intended audience is first and second-year university students, so its prerequisites are high-school levels of understanding for math and physics, and a beginner-level understanding of programming. It is also very useful to anyone whose job involves programming, but hasn't taken a formal course in Computer Architectures - a topic that is often overlooked in software or math-oriented degrees.

The **Course Contents** chapters use the materials from the original course (the MIT OpenCourseWare release), with very small changes (mostly cosmetic in nature).

Where existing, the **Real World Implications** chapters provide some additional context and explanations, not present in the MIT OpenCourseWare edition.

If you wish to download the "source code" for the course, go to https://github.com/PersonalCompute-net/computation-structures/.

Credits

Computation Structures (6.004), Spring 2017 - Original course content, from MIT OpenCourseWare.
Course led by Chris Terman, at MIT.
Originally published at https://ocw.mit.edu/6-004S17 and https://github.com/computation-structures/course/.
Licensed under Creative Commons BY-NC-SA 4.0 - https://ocw.mit.edu/terms.

Eisvogel - LaTeX template and cover artwork.
Created by Pascal Wagler - https://github.com/Wandmalfarbe/.
Originally published at https://github.com/Wandmalfarbe/pandoc-latex-template/.
Licensed under BSD 3-clause license.

Licensing

Standard Cell Library

The Standard Cell Library defines a set of logic gates, latches and registers to be used when doing gate-level simulation. These gates are simulated using Jade's built-in logic primitives, rather than as pullup and pulldown networks of FETs. The resulting increase in simulation speed makes it possible to simulate very large digital designs.

Each library element includes information about its timing specifications and size, used by the gate-level simulator to determine the performance and total size of your design. This information is based on a 180nm CMOS process, which is now out of date! Current state-of-the-art processes have sub-20nm features.

Module	Function	t_{CD} (ns)	t_{PD} (ns)	t_R (ns/pf)	t_F (ns/pf)	load (pf)	size (μ^2)
/gates/inverter	$z = \overline{a}$	0.005	0.02	2.30	1.20	0.007	10
/gates/buffer	$z = a$	0.020	0.08	2.20	1.20	0.003	13
/gates/buffer_h		0.020	0.07	1.10	0.60	0.005	17
/gates/tristate	$z = \begin{cases} a & e = 1 \\ \text{not driven} & e = 0 \end{cases}$	0.030	0.15	2.30	1.30	0.004	23
/gates/and2	$z = a \cdot b$	0.030	0.12	4.50	2.30	0.002	13
/gates/and3	$z = a \cdot b \cdot c$	0.030	0.16	2.50	2.50	0.002	20
/gates/nand2	$z = \overline{a \cdot b}$	0.010	0.03	4.50	2.80	0.004	10
/gates/nand3	$z = \overline{a \cdot b \cdot c}$	0.010	0.05	4.20	3.00	0.005	13
/gates/nand4	$z = \overline{a \cdot b \cdot c \cdot d}$	0.010	0.07	4.40	3.50	0.005	17
/gates/or2	$z = a + b$	0.030	0.15	4.50	2.50	0.002	13
/gates/or3	$z = a + b + c$	0.040	0.21	4.50	2.50	0.003	17
/gates/or4	$z = a + b + c + d$	0.060	0.29	4.50	2.60	0.003	20
/gates/nor2	$z = \overline{a + b}$	0.010	0.05	6.70	2.40	0.004	10
/gates/nor3	$z = \overline{a + b + c}$	0.020	0.08	8.50	2.40	0.005	13
/gates/nor4	$z = \overline{a + b + c + d}$	0.020	0.12	9.50	2.40	0.005	20
/gates/xor2	$z = a \oplus b$	0.030	0.14	4.50	2.50	0.006	27
/gates/xnor2	$z = \overline{a \oplus b}$	0.030	0.14	4.50	2.50	0.006	27
/gates/mux2	$z = \begin{cases} d_0 & s = 0 \\ d_1 & s = 1 \end{cases}$	0.020	0.12	4.50	2.50	0.005	27

Module	Function	t_{CD} (ns)	t_{PD} (ns)	t_R (ns/pf)	t_F (ns/pf)	load (pf)	size (μ^2)
/gates/mux4	$z = \begin{cases} d_0 & s_1 s_0 = 0\text{b}00 \\ d_1 & s_1 s_0 = 0\text{b}01 \\ d_2 & s_1 s_0 = 0\text{b}10 \\ d_3 & s_1 s_0 = 0\text{b}11 \end{cases}$	0.020	0.12	4.50	2.50	0.005	27
/gates/dreg	$d \rightarrow q \text{ on } clk \uparrow$	0.030	0.19	4.30	2.50	0.002	56

The Jade Memory component

Jade has a built-in memory device that can be used to model memories with a specified width and number of locations, and with one to three independent ports. Each port has 3 control signals and the specified number of address and data wires. You can instantiate a memory device in your circuit by clicking on MEM in the toolbar and dragging it onto the schematic. Then double-click the component to specify the width of the address (naddr, between 1 and 20), the width of the data (ndata, between 1 and 128), the number of ports, and (optionally) the initial contents. All the ports of a memory access the same internal storage, but each port operates independently.

Each port has the following connections:

- **OE** is the output enable input for a read port. When 1, data is driven onto the data pins; when 0, the output pins are not driven by this memory port. If this port is only a write port, connect this terminal to ground (e.g., connect to the signal **0'1**). If the port is only a read port and should always be enabled, connect this terminal to the power supply node (e.g., connect to the signal **1'1**).
- **CLK** is the clock input for write ports. When **wen=1**, data from the data terminals is written into the memory on the rising edge of clk. If this port is only a read port, connect this terminal to ground.
- **WE** is the write enable input for write ports. See the description of CLK for details about the write operation. If this port is only a read port, connect this terminal to ground.
- **A[naddr-1:0]** are the address inputs, listed most significant bit first. The values of these terminals are used to compute the address of the memory location to be read or written. The number of locations in the memory is determined by width of the address: nlocations = 2^naddr.
- **D[ndata-1:0]** are the data inputs/tristate outputs, listed most significant bit first.

The contents property can be used to specify the initial contents of a memory (if not specified, the memory is initialized to all X's). The memory is initialized, location-by-location, from the data values

given in the list. The least significant bit (bit 0) of a value is used to initialize bit 0 of a memory location, bit 1 of a value is used to initialize bit 1 of a memory location, etc.

The contents property should be a list of numeric values separated by whitespace (spaces, tabs, newlines). You can use "**0b**" and "**0x**" notation to specify binary and hex values. The characters "**+**" and "**_**" are ignored and can be used to improve readability. The character "**?**" can be used in binary and hex values to represent "don't care" – Jade will replace "**?**" with "**0**". Comments can be included in the contents using the standard "**//**" and "**/* ... */**" comment syntax.

Initialized memories are useful for modeling ROMs (e.g., for control logic) or simply for loading programs into the main memory of a processor. One caveat: if the memory has a write port and sees a rising clock edge with its write enable not equal to 0 and with one or more of the address bits undefined (i.e., with a value of "X"), the entire contents of the memory will also become undefined. So you should **make sure that the write enable for a write port is set to 0 by your reset logic** before the first clock edge, or else your initialization will be for naught.

The following options are the default values for the electrical and timing parameters for the memory.

Parameter	Description
tcd= seconds	The contamination delay in seconds. Default value = 20ps.
tpd= seconds	The propagation delay in seconds. This is how long it takes for changes in the address or output enable terminals to be reflected in the values driven by the data terminals. Default value is determined from the number of locations:

Number of locations	t_{PD}	Inferred type
nlocations ≤ 128	$2ns$	Register file
$128 <$ nlocations ≤ 1024	$4ns$	Static RAM
nlocations > 1024	$40ns$	Dynamic RAM

tr= seconds_per_farad	The output rise time in seconds per farad of output load. Default value is 1000, i.e., 1 ns/pf.
tf= seconds_per_farad	The output fall time in seconds per farad of output load. Default value is 500, i.e., 0.5 ns/pf.
cin= farads	Input terminal capacitance in farads. Default value = 0.05pf.
cout= farads	Output terminal capacitance in farads. Default value = 0pf (additional t_{PD} due to output terminal loading is already included in default t_{PD}).

The size of a memory is determined by the sum of the sizes of the various memory building blocks shown in the following table:

Component	Size (μ^2)	Notes
Storage cells	nbits * cellsize * $1\mu^2$	nbits = nlocs * width cellsize = nports (for ROMs and DRAMs) cellsize = nprots + 5 (for SRAMs)
Address buffers	nports * naddr * $20\mu^2$	nports = total number of memory ports
Address decoders	nports * (naddr+3)/4 * $4\mu^2$	Assuming 4-input ANDs
Tristate drivers	nreads * width * $30\mu^2$	nreads = number of read ports
Write-data drivers	nreads * width * $20\mu^2$	nwrites = number of write ports