Digital Circuits and Verilog

OffCourse Verilog Staff

21st February 2025

Preface

This document aims to introduce you to Verilog as well as digital logic. This document has been carefully compiled in our free time to provide you a supplement to official TU/e resources.

While this document does explain Verilog, it will take a few pages to get to it. This is for the simple reason that digital circuits deserve to be understood independent of what tool you use to work with them. Digital circuits and digital computers have existed since long before the creators of verilog were even born. Thus instead of tying your understanding of digital circuits directly to Verilog, there will be a degree of seperation.

Do let us know about your experience with this document, both positive and negative feedback is appreciated.

Preliminaries

To keep this document focused and not reinvent the wheel too much, not every concept will be fully explained. Here we will indicates what you are expected to understand. These are fairly basic concepts and we are confident you already know them or will quickly be able to understand them elsewhere.

0.1 Binary

Table 0.1: Counting in Decimal and Binary

Value	Decimal	Binary	Value	Decimal	Binary
zero	0	0	eleven	11	1011
one	1	1	twelve	12	1100
two	2	10	thirteen	13	1101
three	3	11	fourteen	14	1110
four	4	100	fifteen	15	1111
five	5	101	sixteen	16	10000
six	6	110	seventeen	17	10001
seven	7	111	eighteen	18	10010
eight	8	1000	nineteen	19	10011
nine	9	1001	twenty	20	10100
ten	10	1010	twenty-one	21	10101

You should be familair with the concept of different numeration systems, especially binary. Where decimal is a base-10 system with 10 symbols and a shift of a number results in a multiplication or division by 10. Binary is a base-2 system with 2 symbols where a shift results in a multiplication or division by 2.

Table 0.2: Shifting of Numbers

← left-shift	value	right-shift →	← left-shift	value	right-shift →
60	6	0.6	1100 (12)	110 (6)	11 (3)
100	10	1	10100 (20)	1010 (10)	101 (5)
times 10		divide 10	times 2		divide 2

The 0 and 1 in binary are called bits, from Bi(nary Digi)ts

If you have never seen binary before and are currently very confused. There are some excellent videos online from the usual suspects like *3Blue1Brown* and *Khan Academy*

We will ignore 1's and 2's compliment, this document only applies basic binary.

Chapter 1

Digital Circuits: The Basics

1.1 From Analogue to Digital

Analogue

The universe itself is analogue, everything is **continous**. The better your tools for observing the universe, the more information you can get from it. A tape measure will tell you your bathroom is 3.14 meters by 2.71 meters, measure the same room with a laser and now its 3.14159 meters by 2.71828 meters. Measure again and the least-significant digits might jump around a little. This creates uncertainty.

Analogue Circuits

Analogue ciruits are the circuits you are familiar with. Circuits built from both passive and active components: resistors, capacitors, operational amplifiers, etc. As analogue circuits are part of the continous universe, they can have many different voltage and current values. 9V, -12V, 6.94V are all valid voltages in an analogue circuit.

Problems

When building analogue circuits or cutting a pizza into slices you can run into some problems. Getting an exact voltage level from a voltage source and ensuring your pizza slices are all equally large is difficult. The uncertain nature of the analogue universe is problematic when you need to be exact.

Digital

Digital systems set themselves apart from analogue systems by replacing continuity with **discrete** values. This is exactly what you are doing when you measure your room. You turn the continous nature of your room into a discrete measurment that depends on the instrument you used to measure.

Digital Circuits

Digital circuits operate with *two* discrete values, based on the bits in binary, **0** and **1**. These two discrete values are obtained from the analogue universe, by approximating voltages into **logic levels**. As an example, the *PYNQ Board* uses 3.3V as its logic level. This means it approximates voltages around 3.3V into a **1** and voltages around 0V into a **0**. Any other voltage is not considered valid. For the *PYNQ* this means it could either be a 0 or a 1. Another popular logic level is 5V but many more exist.

0	False	off	*0V–0.8V*
1	True	on	*2V–3.3V*

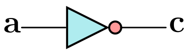
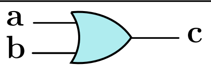
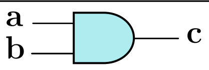
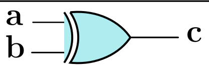
Digital circuits are discrete, either 0 or 1

1.2 The Building Blocks

In analogue circuits you have many different components that all have *continous* effects on the values in the circuit. Digital circuits have their own components that instead have *discrete* effects on the values in the circuit.

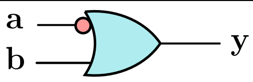
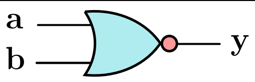
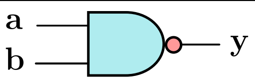
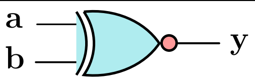
Logic Gates

Table 1.1: Basic Logic Gates

NOT	OR	AND	XOR
~a → c	a | b → c	a & b → c	a ^ b → c
	0 0 | 0	0 0 | 0	0 0 | 0
0 | 1	0 1 | 1	0 1 | 0	0 1 | 1
1 | 0	1 0 | 1	1 0 | 0	1 0 | 1
	1 1 | 1	1 1 | 1	1 1 | 0
			
Inverts bit	Any bit	All bits	Exclusively one bit

The basic building blocks, the **logic gates**, are shown in table 1.1. Per logic gate the relationship of inputs to output is represented as a **truth table**. These four logic gates are the basis from which all other digital circuit components are constructed.

Table 1.2: Inverted Logic Gates

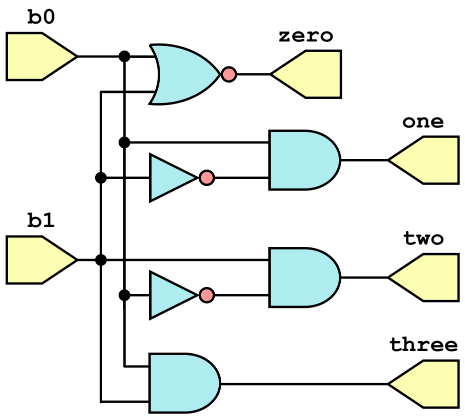
OR (inverted a)	NOR	NAND	XNOR
(~a) | b → y	~(a | b) → y	~(a & b) → y	~(a ^ b) → y
0 0 | 1	0 0 | 1	0 0 | 1	0 0 | 1
0 1 | 1	0 1 | 0	0 1 | 1	0 1 | 0
1 0 | 0	1 0 | 0	1 0 | 1	1 0 | 0
1 1 | 1	1 1 | 0	1 1 | 0	1 1 | 1
			
Everything but just a	No bits	Everything but all bits	All or no bits

Combining basic gates with inverters already defines new gates. Both the outputs and inputs of logic gates can be draw with an implied NOT gate.

1.3 Combinational Logic

The next step after individual logic gates is combining them into circuits. These will be **combinational** logic circuits, *combining a set of inputs into a set of outputs*. Comparable to a mathematical functions where every set of inputs maps to a set of outputs.

Combinational logic maps a set of inputs to a set of outputs



b1	b0	zero	one	two	three
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Figure 1.1: Two bit binary to decimal circuit

Figure 1.1 shows a simple combinational circuit, which turns a two bit binary number into the equivalent decimal number. The relation between the inputs and the outputs of the circuit are expressed with a **truth table**.

Constructing Truth Tables

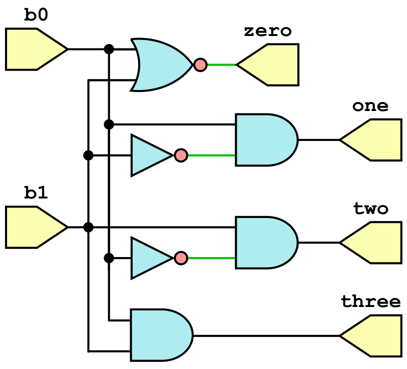
Constructing a truth table is as simple as trying every possible combination of inputs and observing the related outputs. On the next page, in figure 1.2, a visualisation of this process is provided.

Describing combinational circuits

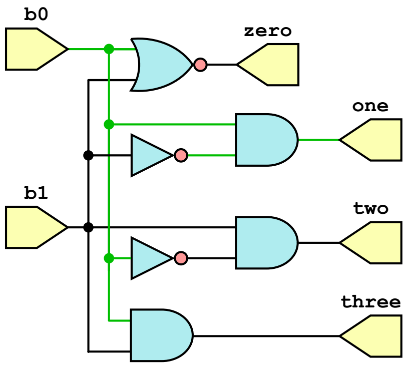
Another method of expressing a combinational circuit, besides circuit diagrams and truth tables, is by describing the outputs as the results of logical operations between the inputs.

zero	=	b1 NOR b0	=	**~(b1 | b0)**
one	=	NOT b1 AND b0	=	**(~b1) & b0**
two	=	b1 AND NOT b0	=	**b1 & (~b0)**
three	=	b1 AND b0	=	**b1 & b0**

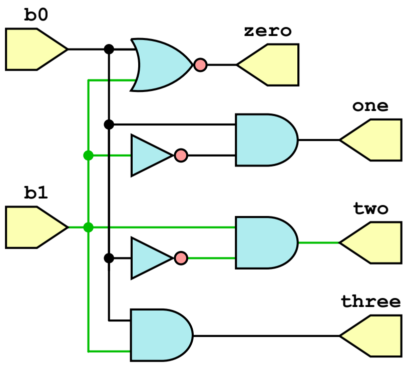
Figure 1.2: Step-by-step truth table



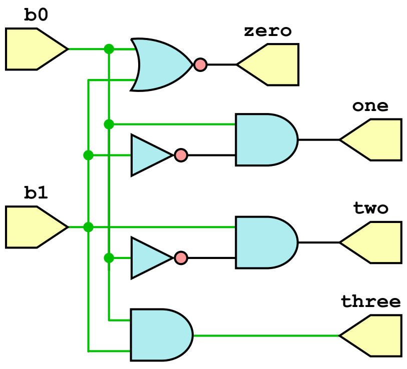
b1	b0	zero	one	two	three
0	0	1	0	0	0



b1	b0	zero	one	two	three
0	0	1	0	0	0
0	1	0	1	0	0



b1	b0	zero	one	two	three
0	0	1	0	0	0
0	1	0	1	0	0
1	0	1	0	1	0

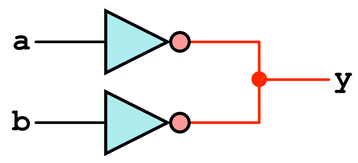
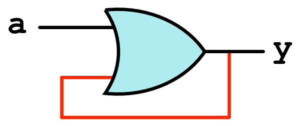


b1	b0	zero	one	two	three
0	0	1	0	0	0
0	1	0	1	0	0
1	0	1	0	1	0
1	1	0	0	0	1

1.4 Wires

Building these combinational circuits requires inter-connecting logic gates. These simple connections are refered to as **wires**. Wires carry a single bit from the output of a *single source*, refered to as the **driver**, to the inputs on one or multiple destinations.

Table 1.3: Wiring rules for combinational circuits

Multiple drivers: Illegal	Feedback: Illegal
	

Multiple drivers

Multiple drivers are illegal in digital circuits. Remember that the bits 0 and 1, represent *different voltage levels*. This means that if two drivers where to be connected to the same wires and both tried to drive the wire to different bits, the wire would become a *short* between two voltages. How these voltages interact is an analogue question and thus illegal in the digital world.

Multiple drivers are illegal, to prevent unforeseen consequences.

Feedback

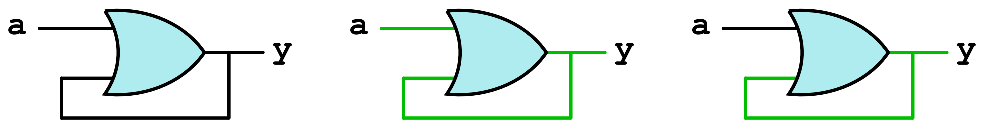


Figure 1.3: Memory Effect of feedback

Figure 1.3 shows the results of feedback in a circuit. After momentarily driving a **1** into *a* the OR gate *latches* the **1** into its output. This circuit has **memory** and unlike combinational logic doesn't have a static relation between inputs and outputs. Thus in order to preserve the combinational nature of combinational logic, memory and as a result feedback is illegal.

To prevent memory, feedback is illegal in combinational logic.

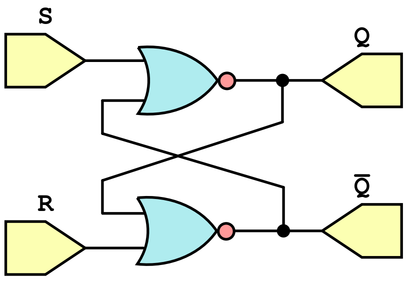
1.5 Sequential Logic

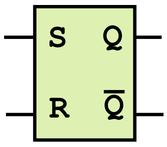
Combinational circuits are restricted to a static mapping of inputs to outputs, prohibiting the use of feedback. However, as figure 1.3 shows, feedback allows the creation of **memory**. Thus a new kind of circuit will be defined that does allow the use of feedback and by extension memory, the **sequential circuit**.

Sequential circuits are combinational circuits with memory.

The term sequential circuit is based on the fact that *the sequence of both past and present inputs define the outputs* of these circuits.

SR Latch





S	R	Q	$\overline{Q}$	Action
0	0	Q	$\overline{Q}$	No change
1	0	1	0	Set
0	1	0	1	Reset
1	1	X	X	Invalid

Figure 1.4: SR latch circuit and symbol

Figure 1.4 shows a sequential circuit, the S(et) R(eset) latch. With the set input the value of the latch is set to a **1** and the reset input resets the latch to a **0**. The outputs of the latch are Q and the inverse of Q, denoted as $\overline{Q}$. Q is considered the primary output. When both S and R are **0**, the SR latch keeps its previous Q value.

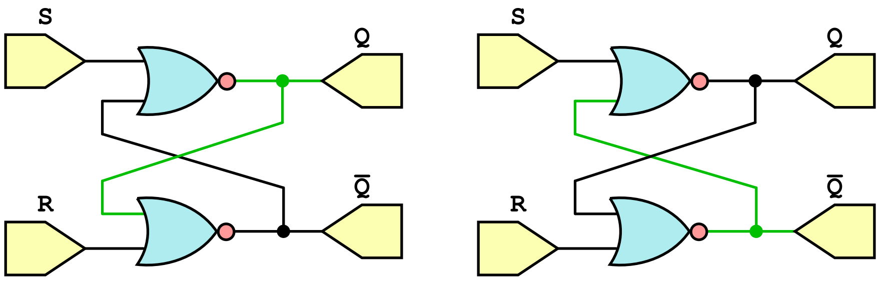


Figure 1.5: Initial conditions of a SR latch

The value a SR latch contains, upon powering on, is indeterminate. It could start with Q set to **0** or **1**, randomly. The value is based on analogue interactions that occur right after powering on the circuit. Thus before the SR latch can be used, it must always be **reset** to ensure the starting state is a known state.

Sequential circuits require a reset at startup to ensure a known state

State graphs

Figure 1.6: Comparing a stategraph to a truth table.

S	R	Q	$\overline{Q}$	Action
0	0	Q	$\overline{Q}$	No change
1	0	1	0	Set
0	1	0	1	Reset
1	1	X	X	Invalid

For combinational circuits a truth table was used to express the relation between inputs and outputs. With sequential circuits being dependent on both past and present inputs this becomes problematic as *a truth table does not show the history of the circuit.* Instead of truth tables, sequential circuits are expressed in time-dependent graphs that contains all inputs, outputs and values of the memory in a circuit. Such a graph is called a **waveform**.

Waveforms describe a circuit over time.

Figure 1.6 shows how a waveform can display indeterminate states upon startup aswell as inputs, outputs and the values of memory.

Constructing Waveforms

Unlike the truth tables

Chapter 2

Verilog: Single Module

2.1 Comparing C to Verilog

C

Programming, as you will have done it up until now, in languages such as C has been fairly straight forward. You write lines of code that the computer then executes in the order you have written. Besides some processing to turn the human text into machine readable instructions the structure of the program you write is maintained. The order of execution for variables, if-statements, function calls, etc is the same to you as it is to the computer.

C code becomes instructions the computer can execute.

Verilog

Verilog is not a programming language. It will resemble the C code you are familiar with, it will even have some of the same elements. If-statements and switch-statements will show up just like they do in C. *Verilog is however **not** a programming language* It is not used to create instructions for a computer to execute. Verilog instead exists to describe the design of digital circuits. Making verilog a **HDL** *hardware description language.*

Verilog describes digital circuits, not programs for a computer

First off explain the structure of verilog v C. aka the top module is the main function and submodules are subfunctions. BUT EVERYTHING IS MADE INTO A CIRCUIT SO SUBMODULES ARE RECREATED, NOT RECALLED.

THERE IS NO PROGRAM FLOW IN VERILOG, FOR THERE IS NO PROGRAM.

Second off do the three levels of verilog abstraction.

Gate level Modeling -> this is the shit one Data flow Modeling -> this is the fast one Behavrial Modeling -> this is the C-like one, this is the real stuff DO the same thing that guy in the video did with the MUX.

Everything till now has been combinational logic. Intro sequential logic and explain how its different. Touch on blocking and non-blocking assignment.

Do testbenches at the VERY FUCKING END. Because they kinda ruin the point about verilog.