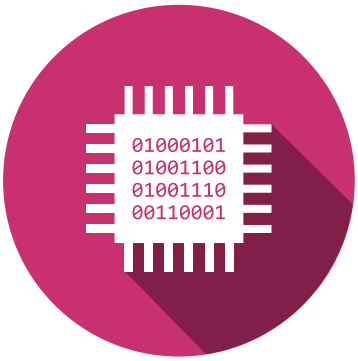


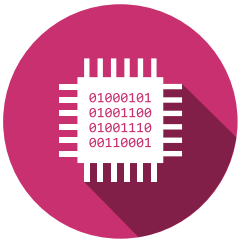
Cheatsheet (CHEAT)

Course Digital Design

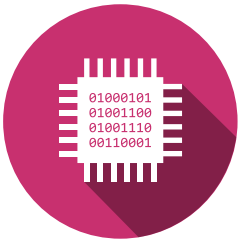
Hes·so VALAIS WALLIS

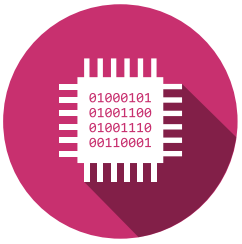
School of Engineering

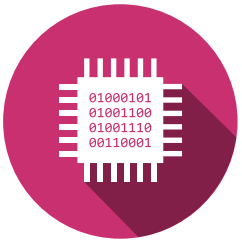
Orientation: Systems Engineering (SYND)
Course: Digital Design (DiD)
Author: Christophe Bianchi, François Corthay, Pierre Pompili, Silvan Zahno
Date: August 25, 2022
Version: v2.1



Contents







1 IND - Intro

1.1 Bender

1.1.1 Just giberish

$$0101100101_2 = 1 + 4 + 32 + 64 + 256 = 357_{10} \tag{1}$$

1.1.2 AAAAAAAAA

$$1010011010_2 = 2 + 8 + 16 + 128 + 512 = 666_{10} \tag{2}$$

1.2 Energy to change a bit

If a gate wants to send an impulse. For each state change $0 \Leftrightarrow 1$. A capacitor needs to be charged or discharged.

1 Bit change

$$Q = \frac{1}{2}CU^2 \tag{3}$$

One impuls

$$Q = CU^2 \tag{4}$$

The power P is the energy Q multiplied by the frequency f

$$P = fCU^2 \tag{5}$$

If the Power want's to be minimized we can to 3 things

1. Reduce the frequency f but this reduces also the computing power
2. Reduce the capacity by using smaller fabrication technology ($14nm \Longrightarrow 7nm$), but there is a physical limit
3. Reducing the voltage U most efficient because it is in square U^2. Reducing the voltage reduces the energy by a factor of two.

2 NUM - Numeric Representation and Codes

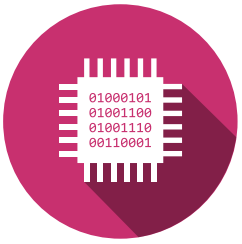
2.1 Number System

2.1.1 Representation Integer

$$\begin{aligned} A &= \sum_{i=0}^{n-1} a_i * p^i \\ 0 \leq a_i &\leq p-1 \\ 0 \leq A &\leq p^n \end{aligned} \tag{6}$$

2.1.2 Decimal System

$$245_{10} = 2 * 10^2 + 4 * 10^1 + 5 * 10^0 \tag{7}$$



2.1.3 Binary System

$$11110101_2 = 1 * 2^7 + 1 * 2^6 + 1 * 2^5 + 1 * 2^4 + 0 * 2^4 + 0 * 2^3 + 1 + 2^2 + 0 * 2^1 + 1 * 2^0 \tag{8}$$

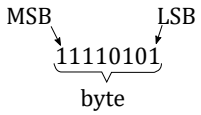


Figure 1: Binary Representation

2.1.4 Hexadecimal System

$$F5_{16} = F * 16^1 + 5 * 16^1 + 5 * 16^0 \tag{9}$$

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Table 1: Decimal vs Hexadecimal vs Binary

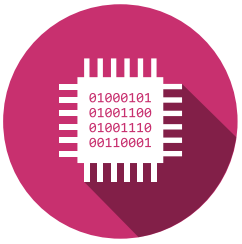
2.2 Transformations of Number Systems

2.2.1 Binary to Decimal

$$\begin{aligned} 11110101_2 &= 2^7 + 2^6 + 2^5 + 2^4 + 2^2 + 2^0 \\ &128 + 64 + 32 + 16 + 4 + 1 = 245_{10} \end{aligned} \tag{10}$$

2.2.2 Decimal to Binary

$$77_{10} = ...128 + \underline{64} + 32 + 16 + \underline{8} + \underline{4} + \underline{1} = 01001101_2 \tag{11}$$




```
245 / 2 = 122 + remainder 1
122 / 2 = 61  + remainder 0
61  / 2 = 30  + remainder 1
30  / 2 = 15  + remainder 1
15  / 2 = 7   + remainder 1
7   / 2 = 3   + remainder 1
3   / 2 = 1   + remainder 1
1   / 2 = 0   + remainder 1
                  245₁₀ = 11110101₂
```

Figure 2: Decimal to Binary

2.2.3 Hexadecimal to Binary

$$F5_{16} = 11110101_2 \tag{12}$$

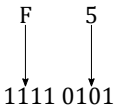


Figure 3: Hexadecimal to Binary

2.2.4 Binary to Hexadecimal

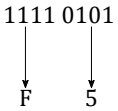


Figure 4: Binary to Hexadecimal

2.2.5 Hexadecimal to Decimal

$$\begin{aligned} F5_{16} &= 15 * 16^1 + 5 + 16^0 \\ &15 * 16 + 5 + 1 = 254_{10} \end{aligned} \tag{13}$$

2.2.6 Decimal to Hexadecimal

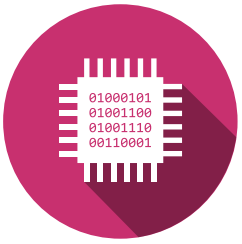
```
15  / 16 = 0  + remainder 15
1   / 2 = 0   + remainder 1
                  245₁₀ = F5₁₆
```

Figure 5: Decimal to Hexadecimal

2.3 Codes

2.3.1 BCD

Binary Coded Decimal. 4 binary bits correspond to one decimal number. Only 0.9 is used therefore inefficient coding. Calculations are more complicated.



Decimal	Hexadecimal	Binary	BCD
0	0	0000	0
1	1	0001	1
2	2	0010	2
3	3	0011	3
4	4	0100	4
5	5	0101	5
6	6	0110	6
7	7	0111	7
8	8	1000	8
9	9	1001	9
10	A	1010	-
11	B	1011	-
12	C	1100	-
13	D	1101	-
14	E	1110	-
15	F	1111	-

Table 2: BCD Coding

2.3.2 Gray-Code

Reflected code.

Binary	Gray Code
$b_4b_3b_2b_1$	$g_4g_3g_2g_1$
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Table 3: Gray Coding

Gray to Decimal

$$g_4 = b_4$$

$$g_3 = b_4 \oplus b_3$$

$$g_2 = b_3 \oplus b_2$$
$$g_1 = b_2 \oplus b_1$$

Gray to Decimal

$$b_4 = g_4$$
$$b_3 = g_4 \oplus g_3 = b_4 \oplus g_3$$
$$b_2 = g_4 \oplus g_3 \oplus g_2 = b_3 \oplus g_2$$
$$b_1 = g_4 \oplus g_3 \oplus g_2 \oplus g_1 = b_2 \oplus g_1$$

2.4 Operation on Integers

2.4.1 Binary Addition

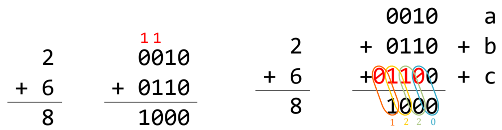


Figure 6: Binary Addition

2.4.2 Binary Substraction

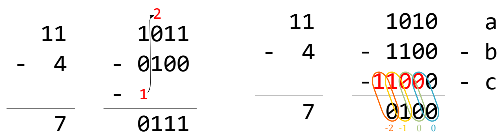


Figure 7: Binary Substraction

2.4.3 Binary Multiplication

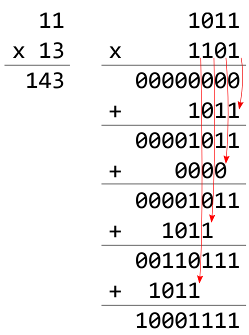
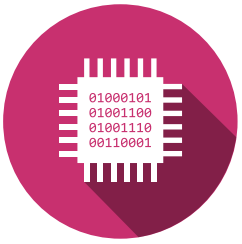


Figure 8: Binary Multiplication

2.5 Explanations

2.5.1 Addition and Substraction



$$\begin{array}{r} 3 \\ +(-5) \\ \hline -2 \end{array} \quad (14) \qquad \begin{array}{r} 0011 \\ +(1011) \\ \hline 1110 \end{array} \quad (15) \qquad \begin{array}{r} 3 \\ -5 \\ \hline -2 \end{array} \quad (16) \qquad \begin{array}{r} 0011 \\ -0101 \\ \hline 1110 \end{array} \quad (17)$$

2.5.2 Multiplication and Division by 2

A multiplication by the power of two is a shift left. A division by the power of two is a shift right. But be careful with the MSB depending on the binary format (2nd Complement etc.).

- Remember to add either 1 or 0 to keep the sign in 2nd-Complement
- During a division the comma values are lost $\frac{5}{2} = 2$ or $\frac{-5}{2} = -3$

$$5 * 2 = 10 \quad (18) \qquad 5/2 = 2.5 = 2 \quad (19) \qquad -5 * 2 = -10 \quad (20) \qquad -5/2 = -2.5 = -3 \quad (21)$$
$$0101_2 * 2 = 01010_2 \qquad 0101_2/2 = 0010_2 \qquad 1011_2 * 2 = 10110_2 \qquad 1011_2/2 = 1101_2$$

3 COM - Kombinatorical Logicfunctions

aka Boolsche Algebra

- https://de.wikipedia.org/wiki/Boolesche_Algebra
- https://kogler.wordpress.com/2008/03/21/latex-use-of-math-symbols-formul

3.1 Operators

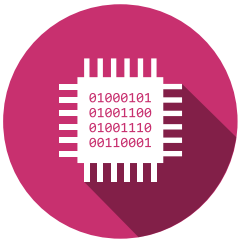
Operator	Latex Symbol	Latex Code
NEGATE	$\neg\ \overline{x}\ \bar{x}$	`\neg \overline {x} \bar {x}`
AND	$\bigwedge\ \wedge\ *\ \&$	`\bigwedge \wedge * \And`
OR	$\bigvee\ \vee\ +\ \mid$	`\bigvee \vee + \mid`
XOR	$\oplus$	`\oplus`

Table 4: Decimal vs Hexadecimal vs Binary

3.2 Algebra rules

3.2.1 Duality Rules

$$\begin{aligned} \overline{0} &= 1 \\ \overline{1} &= 0 \end{aligned} \quad (22)$$



3.2.2 Neurality Rules

$$\begin{aligned} a * 1 &= a \\ a + 0 &= a \end{aligned} \tag{23}$$

3.2.3 Extremity Rules

$$\begin{aligned} a * 0 &= 0 \\ a + 1 &= 1 \end{aligned} \tag{24}$$

3.2.4 Dual Negation Rule (Involution)

$$\overline{(\overline{a})} = a$$

3.2.5 Idempotence Rules

Idempotence is the property of certain operations in mathematics and computer science whereby they can be applied multiple times without changing the result beyond the initial application.

$$\begin{aligned} a * a &= a \\ a + a &= a \end{aligned} \tag{25}$$

3.2.6 Complementary Rules

$$\begin{aligned} a * \overline{a} &= 0 \\ a + \overline{a} &= 1 \end{aligned} \tag{26}$$

3.2.7 Commutativity Rules

Commutativity rule, is a mathematical rule. It applies when arguments of an operation can be interchanged.

$$\begin{aligned} a * b &= b * a \\ a + b &= b + a \end{aligned} \tag{27}$$

3.2.8 Associativity Rules

The law of Associativity is a rule of mathematics. A (two-digit) logic operation is associative if the order of its execution is irrelevant.

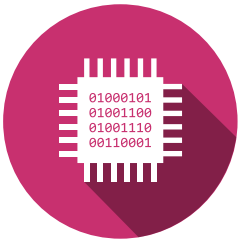
$$\begin{aligned} (a * b) * c = a * (b * c) \quad &= a * b * c \\ (a + b) + c = a + (b + c) \quad &= a + b + c \end{aligned} \tag{28}$$

3.2.9 Distributivity Rules

$$\begin{aligned} a * (b + c) &= (a * b) + (a * c) \\ a + (b * c) &= (a + b) * (a + c) \end{aligned} \tag{29}$$

3.2.10 De Morgansche Rules

$$\begin{aligned} \overline{(a * b)} &= \overline{a} + \overline{b} \\ \overline{(a + b)} &= \overline{a} * \overline{b} \end{aligned} \tag{30}$$



3.2.11 Absorption Rules

$$\begin{aligned} a + (a * b) &= a \\ a * (a + b) &= a \\ a + (\overline{a} * b) &= a + b \\ a * (\overline{a} + b) &= a * b \end{aligned} \tag{31}$$

3.3 Some Proofs

3.3.1 Absorption Rule 1

$$a * (a + b) = a \tag{32}$$

$$\begin{aligned} a * (a + b) &= a \\ (a * a) + (a * b) &= a \\ a + a * b &= a \\ a * 1 + a * b &= a \\ a * (1 + b) &= a \\ a * 1 &= a \\ a &= a \end{aligned} \tag{33}$$

3.3.2 Absorption Rule 2

$$a + (a * b) = a \tag{34}$$

$$\begin{aligned} a + (a * b) &= a \\ (a + a) * (a + b) &= a \\ a * a + b &= a \\ a + 0 * a + b &= a \\ a + (0 * b) &= a \\ a + 0 &= a \\ a &= a \end{aligned} \tag{35}$$

3.3.3 Absorption Rule 3

$$a + (\overline{a} * b) = a + b \tag{36}$$

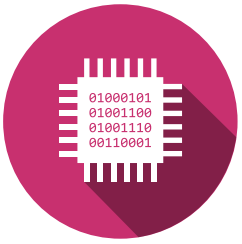
$$\begin{aligned} a + (\overline{a} * b) &= a + b \\ a + \overline{a} * a + b &= a + b \\ 1 * (a + b) &= a + b \\ a + b &= a + b \end{aligned} \tag{37}$$

3.3.4 Absorption Rule 4

$$a * (\overline{a} + b) = a * b \tag{38}$$

$$\begin{aligned} a * (\overline{a} + b) &= a * b \\ a * \overline{a} + a * b &= a * b \\ 0 + a * b &= a * b \\ a * b &= a * b \end{aligned} \tag{39}$$

3.3.5 Xor 4 Nand Solution 1



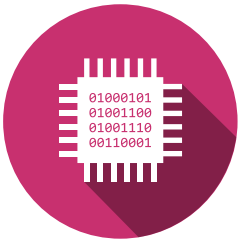
$$a \oplus b = a\overline{b} + \overline{a}b \tag{40}$$

$$\begin{aligned}
&a \oplus b \\
&a\overline{b} + \overline{a}b \\
&\overline{(\overline{a} + \overline{\overline{b}})} + \overline{(\overline{\overline{a}} + \overline{b})} \\
&\overline{(\overline{a} + b)} + \overline{(a + \overline{b})} \\
&\overline{\overline{(\overline{a} + b)} * \overline{(a + \overline{b})}} \\
&\overline{\overline{(\overline{a} + b) * (a + \overline{b})}} \\
&\overline{\overline{a}a + \overline{a}\overline{b} + ab + b\overline{b}} \\
&\overline{\overline{0 + \overline{a}\overline{b} + ab + 0}} \\
&\overline{\overline{a}\overline{b} + ab} \\
&\overline{\overline{a\overline{b}} * \overline{\overline{a}\overline{b}}} \\
&\overline{a\overline{b}} * \overline{\overline{a}\overline{b}} \\
&\overline{a\overline{b}} * \overline{\overline{\overline{a}} + \overline{b}} \\
&\overline{a\overline{b}} * (a + b) \\
&\overline{a\overline{b}}a + \overline{a\overline{b}}b \\
&\overline{\overline{a\overline{b}}a + \overline{a\overline{b}}b} \\
&\overline{\overline{a\overline{b}}a} * \overline{a\overline{b}}b \\
&\overline{\overline{a\overline{b}}a} * \overline{\overline{a\overline{b}}b}
\end{aligned} \tag{41}$$

3.3.6 Xor 4 Nand Solution 2

$$a \oplus b = a\overline{b} + \overline{a}b \tag{42}$$

$$\begin{aligned}
&a \oplus b \\
&a\overline{b} + \overline{a}b \\
&a\overline{a} + a\overline{b} + b\overline{b} + \overline{a}b \\
&a(\overline{a} + \overline{b}) + b(\overline{b} + \overline{a}) \\
&a\overline{ab} + b\overline{ab} \\
&\overline{\overline{aba} * \overline{abb}}
\end{aligned} \tag{43}$$



3.4 Summary

Topic	Rules
Misc	$\overline{0} = 1$ $\overline{1} = 0$ $\overline{(\overline{a})} = a$
AND	$a * 0 = 0$ $a * 1 = a$ $a * a = a$ $a * \overline{a} = 0$ $(a * b) * c = a * (b * c) = a * b * c$
OR	$a + 0 = a$ $a + 1 = 1$ $a + a = a$ $a + \overline{a} = 1$ $(a + b) + c = a + (b + c) = a + b + c$
XOR	$a \oplus b = a\overline{b} + \overline{a}b$
Commutativity	$a * b = b * a$ $a + b = b + a$
Associativity	$(a * b) * c = a * (b * c) = a * b * c$ $(a + b) + c = a + (b + c) = a + b + c$
Distributivity	$a * (b + c) = (a * b) + (a * c)$ $a + (b * c) = (a + b) * (a + c)$
De Morgan	$\overline{(a * b)} = \overline{a} + \overline{b}$ $\overline{(a + b)} = \overline{a} * \overline{b}$
Absorption	$a + (a * b) = a$ $a * (a + b) = a$ $a + (\overline{a} * b) = a + b$ $a * (\overline{a} + b) = a * b$

Table 5: Boolean Algebra Summary

4 KAR - Karnaugh Tables

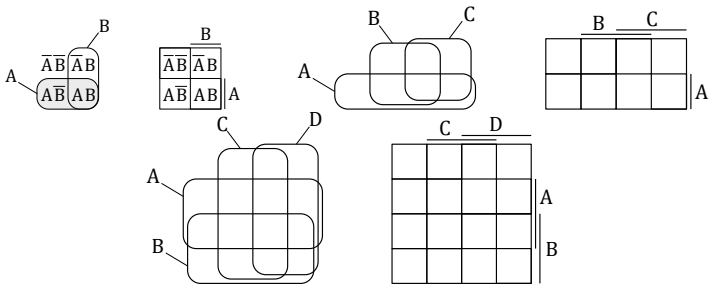


Figure 9: Karnaugh Tables 2, 3 and 4 variables

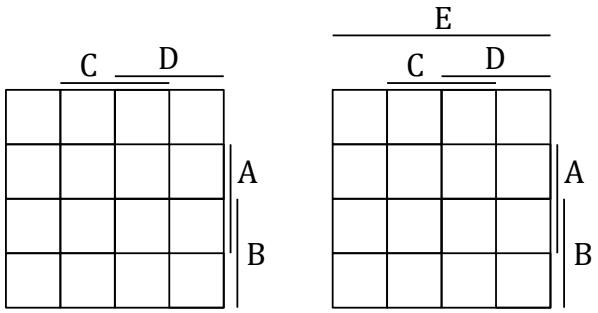


Figure 10: Karnaugh Tables 5 variables

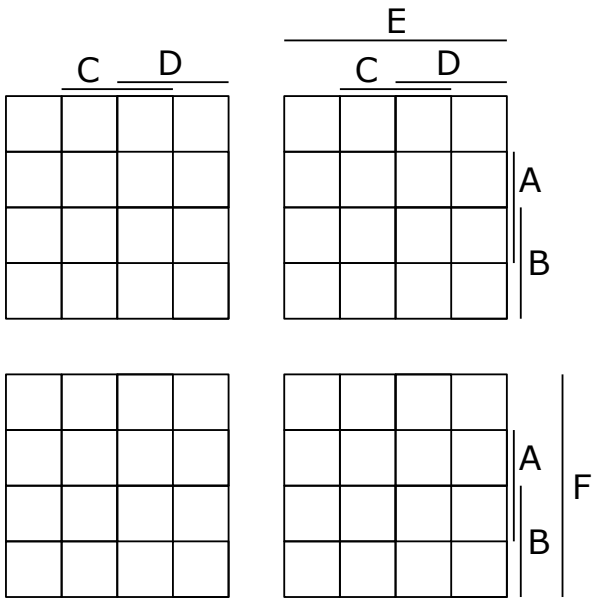


Figure 11: Karnaugh Tables 6 variables

4.1 Block Diagram

4.1.1 Block Tree Diagram

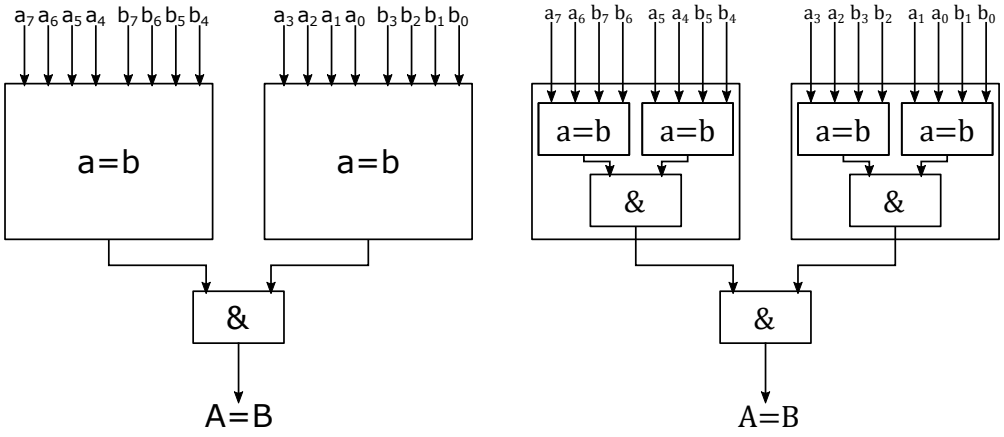


Figure 12: Compare Block Tree 1st Level and 2nd Level

4.1.2 Iterative Block Diagram

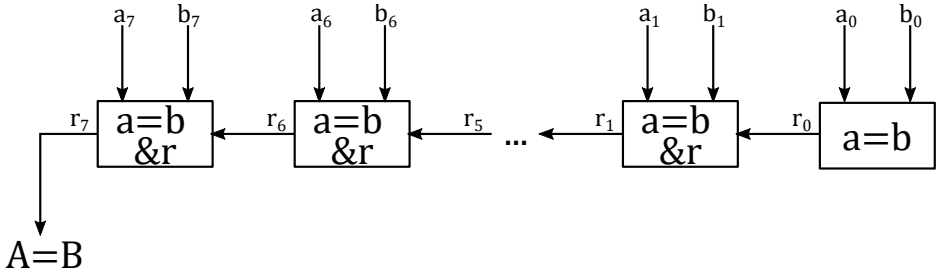


Figure 13: Compare Iterative Block Diagram

5 MUX - Multiplexer and Demultiplexer

5.1 Multiplexer

There are two methods to use multiplexer:

- as a switch between two signals
- to represent a truth table

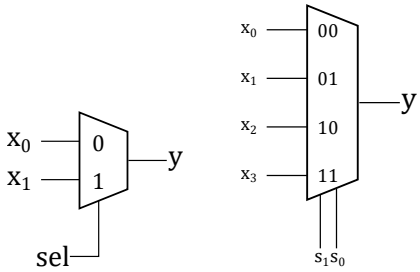


Figure 14: Multiplexer 2-1 / Multiplexer 4-1

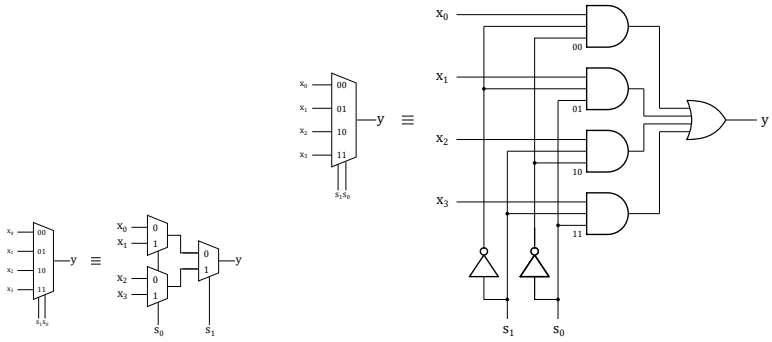


Figure 15: Multiplexer 4-1 Tree / Multiplexer 4-1 Gates

5.1.1 Simplifications

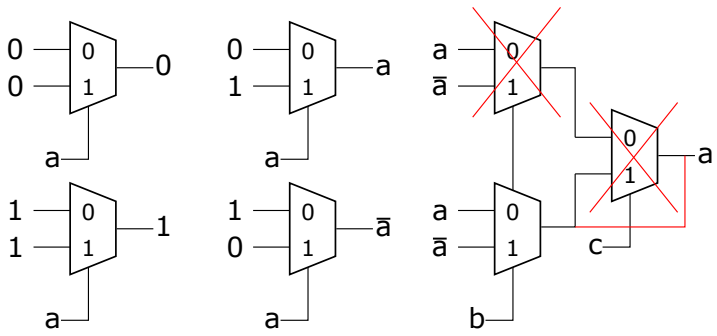


Figure 16: Multiplexer Simplification

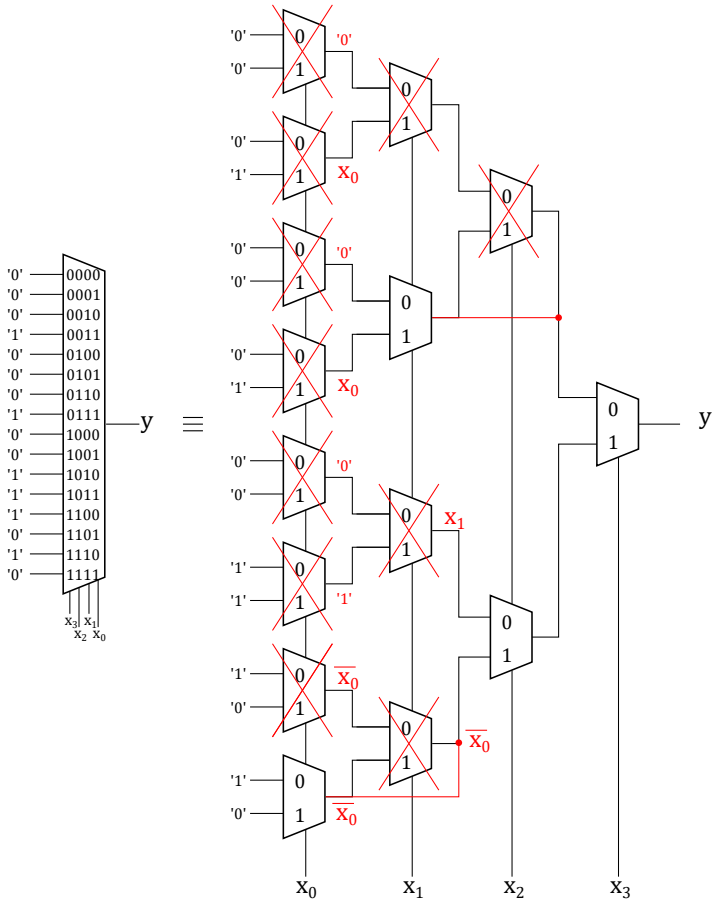


Figure 17: Multiplexer Simplification

5.2 Demultiplexer

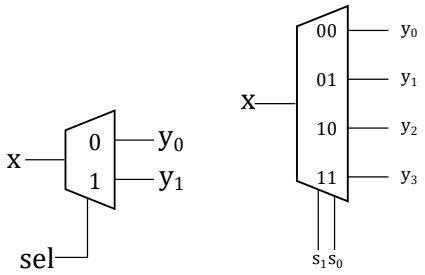


Figure 18: Demultiplexer 1-2 / Demultiplexer 1-2

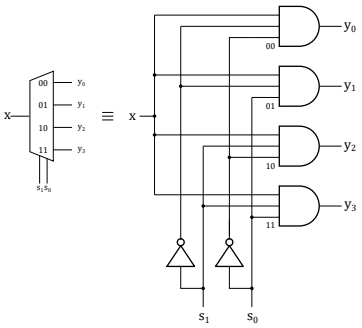


Figure 19: Demultiplexer 4-1 with Gates

6 LST - Logic States

6.1 Open-Drain

All Open-Drain Gates need to be active (set '0') for the output to be '0' ➪ Cabled-AND.

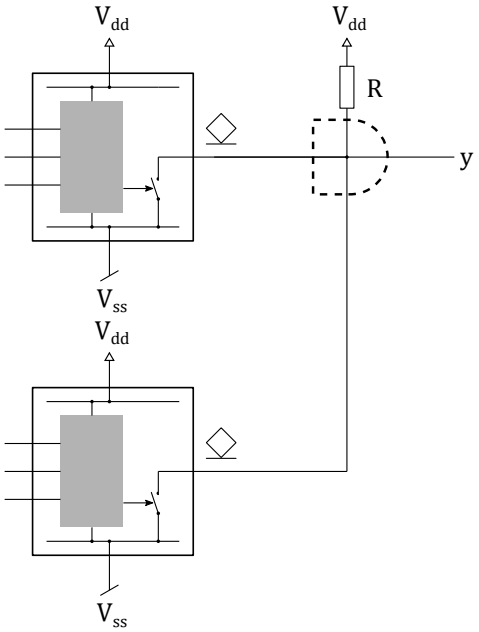


Figure 20: Open-Drain Schema

6.2 Open-Source

If one Open-Source Gate is active (set '1') the output is '1' ➪ Cabled-OR.

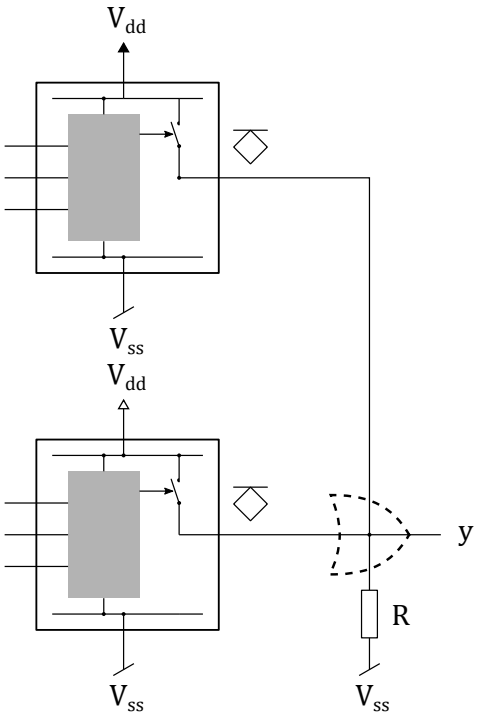


Figure 21: Open-Source Schema

6.3 Tristate

OE	*x*	*y*	Function
0	0	'Z'	Output inactive
0	1	'Z'	
1	0	'0'	Output active
1	1	'1'	

Table 6: SR-Latch Truth Table

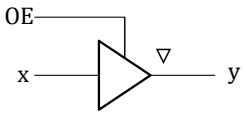


Figure 22: Tristate Element

6.4 Signal States

There are different state defined, they are especially important within the simulation. They help find architectural problems and errors.

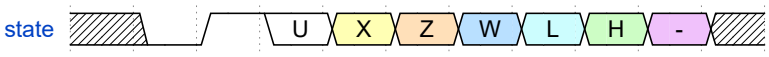
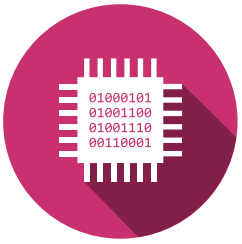


Figure 23: All Signal States



6.4.1 State $'U'$ Uninitialized

At the beginning the output of the gate cannot be evaluated therefore the state $'U'$ is defined to indicate an uninitialized state.

Figure 24: Uninitialized state

6.4.2 State $'X'$ Unknown

The state $'X'$ Unknown is mostly used when there is a problem in the design. It should normally never happen. For example a $'0'$ and a $'1'$ create a shortcircuit which is defined as state $'X'$.

In the example below $'1' and' X' = ' X'$ but $'0' and' X' = ' 0'$ because its an AND.

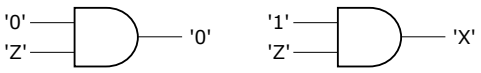


Figure 25: Unknown state

6.4.3 `std_logic` vs `std_ulogic`

- `std_logic` - The resolved type
  - Closer to the Hardware
  - For Tri-States
- `std_ulogic` - The unresolved type
  - For Simulation
  - Generates Metavalue warnings during simulation
  - Generates compilation errors

States	U	X	0	1	Z	W	L	H	$-$
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	X	0	0	0	0	X
1	U	X	X	1	1	1	1	1	X
Z	U	X	0	1	Z	W	W	W	X
W	U	X	0	1	W	W	W	W	X
L	U	X	0	1	L	W	L	W	X
H	U	X	0	1	H	W	W	H	X
$-$	U	X	X	X	X	X	X	X	X

Table 7: Signal state resolving table

The following image shows the difference if `std_ulogic` or `std_logic` is used.

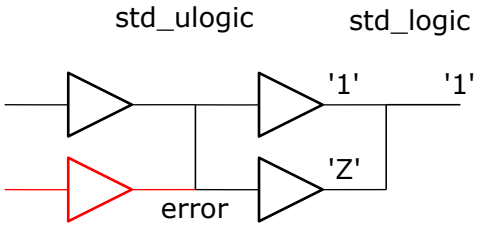


Figure 26: `std_logic` vs `std_ulogic`

7 LAT - Memory and FlipFlops

7.1 Clock Signal

The main clock signal is like the reset an external signal. It is generated with the help of a quartz. Inside is a circuit and a piezo crystal. The piezo can be aggitated and it will "vibrate" in a given frequency.

- is like a tuning fork (Stimmgabel) (Diapason)
- once agitated is will give a fixed frequency depeding on the physical properties
- but a tuning fork will fade.
- with an inverter the output is feed back into the crystal and agitates exactly when its needed to. (Like a swing (Schaukel)(Balançoire))
- a piezo crystals properties can be change with additional capacitors which define the resonance frequency
- normally you buy a Quartz which includes crystal and its setup condensators

Figure 27: Crystal Representation

7.1.1 Calculation

Serial resonance

$$f_s = \frac{1}{2\pi\sqrt{LC_1}}$$

Parallel resonance

$$f_p = \frac{1}{2\pi\sqrt{L\frac{C_0C_1}{C_0+C_1}}}$$

Quality Factor

$$Q = \frac{1}{2\pi F_s R C_1}$$

7.2 Latch

The simplest memory element consists of two inverters. The inverters are looped so that the saved signal is trapped.

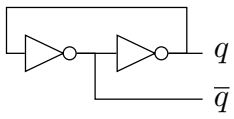


Figure 28: Latch Diagram

7.2.1 SR-Latch

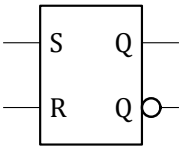


Figure 29: SR-Latch Diagram

$$q = s + \overline{r}q \tag{44}$$

s	r	q	$\overline{q}$	Function
0	0	unchanged	unchanged	Storage
0	1	0	1	Set to Zero
1	0	1	0	Set to One
1	1	0	0	Not allowed

Table 8: SR-Latch Truth Table

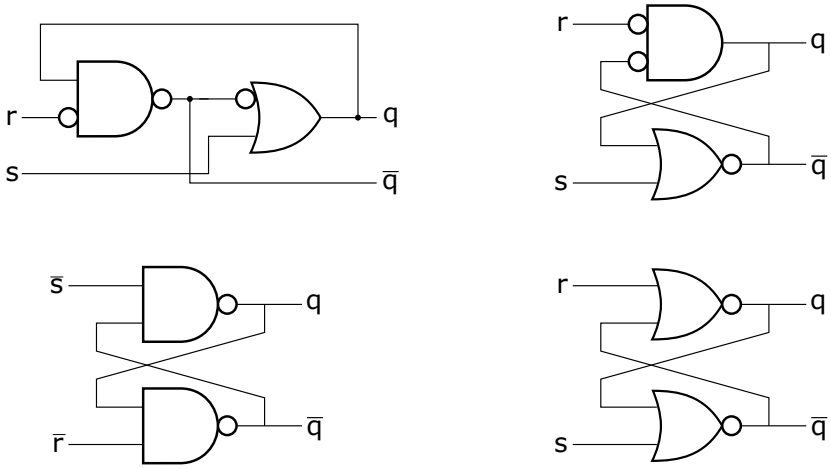


Figure 30: SR-Latch Diagrams

7.2.2 D-Latch

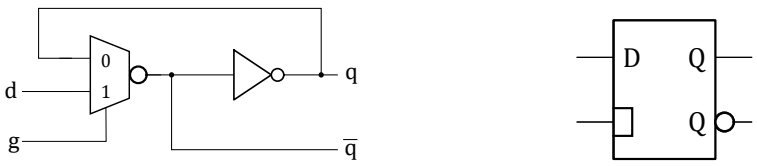


Figure 31: D-Latch Diagram

$$q = gd + \overline{g}q \tag{45}$$

g	d	q	Function
0	0	unchanged	Storage
0	1	0	
1	0	1	Load Value D
1	1	0	

Table 9: D-Latch Truth Table

7.3 FlipFlops

7.3.1 SR-FlipFlop

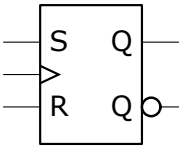


Figure 32: SR-FlipFlop Diagram

$$q^+ = s + \overline{r}q \tag{46}$$

7.3.2 D-FlipFlop

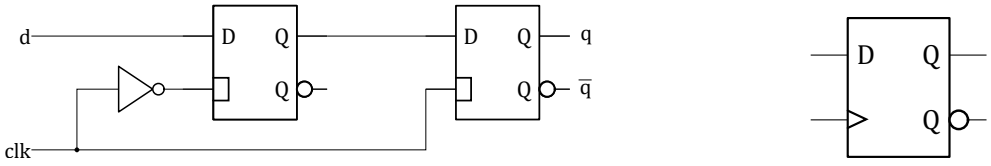


Figure 33: D-FlipFlop Diagram

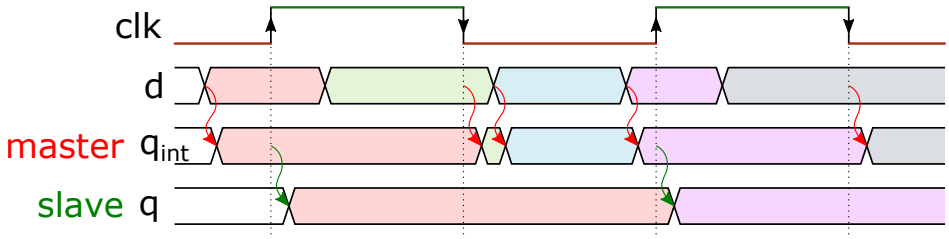
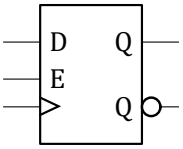


Figure 34: D-FlipFlop Timing

$$q^+ = gd + \overline{g}q \tag{47}$$

7.3.3 DE-FlipFlop



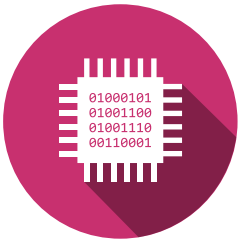


Figure 35: DE-FlipFlop Diagram

$$q^+ = ed + \overline{e}q \tag{48}$$

e	d	q	q^+	Function
0	0	0	0	Storage
0	0	1	1	
0	1	0	0	
0	1	1	1	
1	0	0	0	Load Value D
1	0	1	0	
1	1	0	1	
1	1	1	1	

Table 10: D-Latch Truth Table

7.3.4 T-FlipFlop

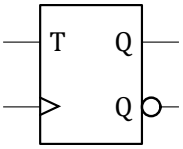


Figure 36: T-FlipFlop Diagram

$$q^+ = t \tag{49}$$

r	q	q^+	Function
0	0	0	Storage
0	1	1	
1	0	1	Invert
1	1	0	

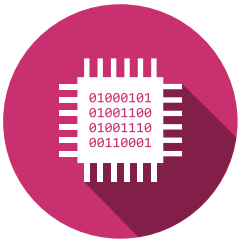
Table 11: D-Latch Truth Table

7.3.5 JK-FlipFlop

$$q^+ = j\overline{q} + \overline{k}q \tag{50}$$

j	k	q^+	Function
0	0	No Change	Storage
0	1	1	Set
1	0	0	Reset
1	1	Invert	Invert

Table 12: D-Latch Truth Table



7.4 Timing

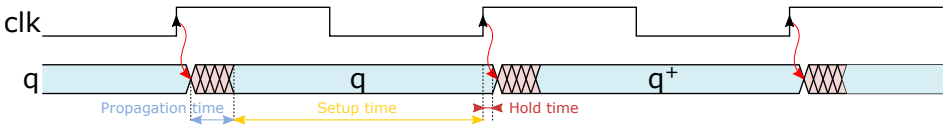


Figure 37: FlipFlop Timing

8 CNT - Synchronous Counters

8.1 Counter Architectures

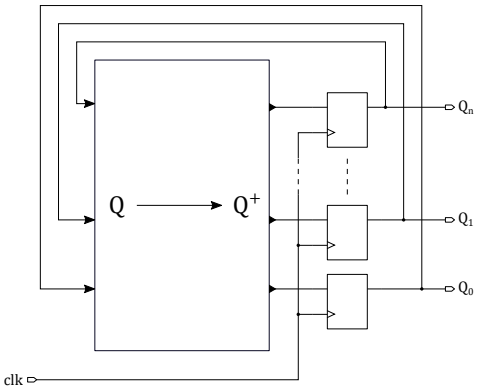


Figure 38: Counter Block Architecture

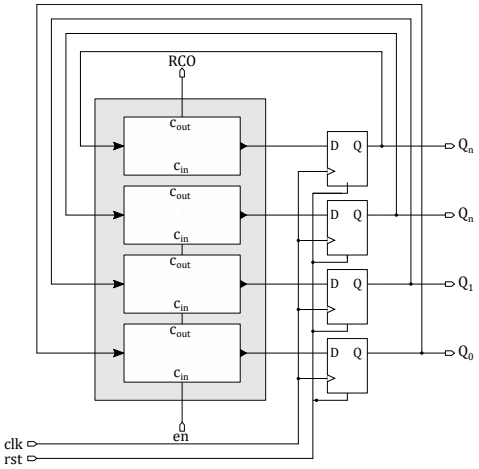


Figure 39: Iterative Counter Architecture

8.2 Up counter

0 ➪ 1 ➪ 2 ➪ 3 ... ➪ 14 ➪ 15

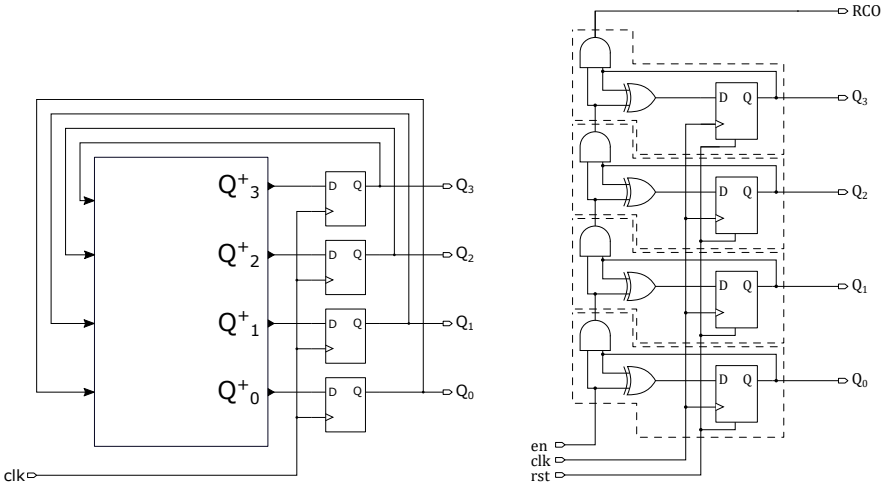


Figure 40: 4 Bit Counter

8.2.1 D-FlipFlops

$$D_0 = Q_0 \oplus 1 = \overline{Q_0}$$
$$D_1 = Q_1 \oplus Q_0$$
$$D_2 = Q_2 \oplus Q_1 Q_0$$
$$D_3 = Q_3 \oplus Q_2 Q_1 Q_0$$

8.2.2 T-FlipFlops

$$T_0 = 1$$
$$T_1 = Q_0$$
$$T_2 = Q_1 Q_0$$
$$T_3 = Q_2 Q_1 Q_0$$

8.3 Translation Table

Q	Q^+	D	T	E	D
0	0	0	0	0 1	- 0
0	1	1	1	1	1
1	0	0	1	1	0
1	1	1	0	0 1	- 1

Table 13: Translation Table D-FF and T-FF and DE-FF

8.4 Counter Example 0-99

In semester project chronometer we want to have a counter for 1 sec. Our Clock is 66MHz which means a counter counting to 66'000'000. Here a example of a iterative counter counting from 0 to 99 and back to 100.

First binary of 99 = ~~128~~ + 64 (rest 35) + 32 (rest 3) + ~~16~~ + ~~8~~ + ~~4~~ + 2 (rest 1) + 1 = 0110 0011

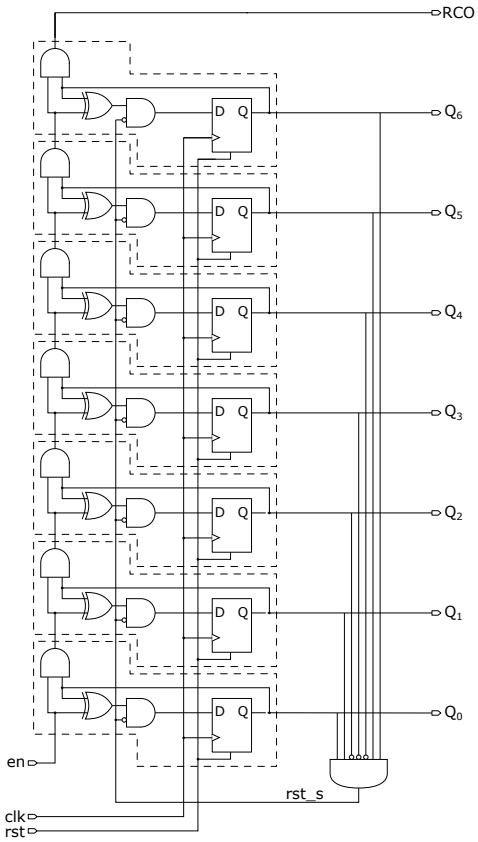


Figure 41: 0-99 Counter

8.5 Multifunctional Iterative Counter

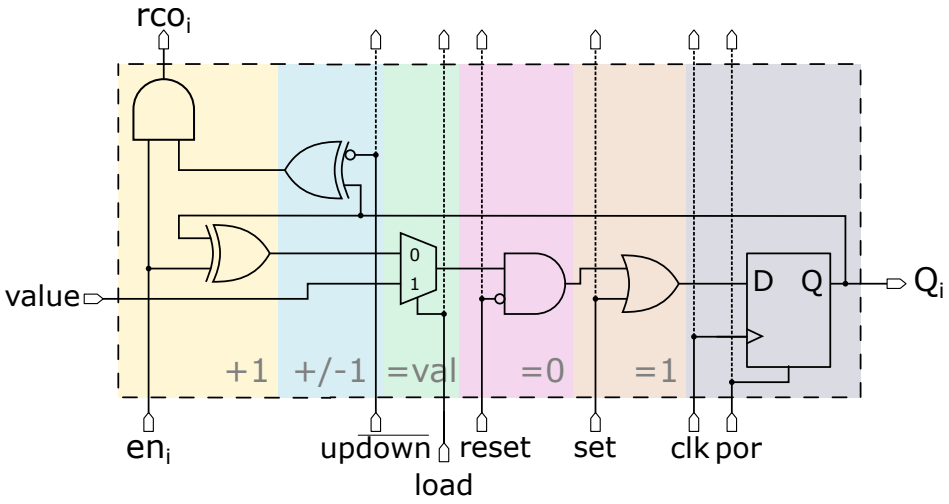
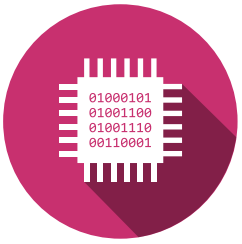


Figure 42: Multifunctional Iterative Counter



9 MET - Methodology for Digital Circuit Development

9.1 Development Model - V Diagram

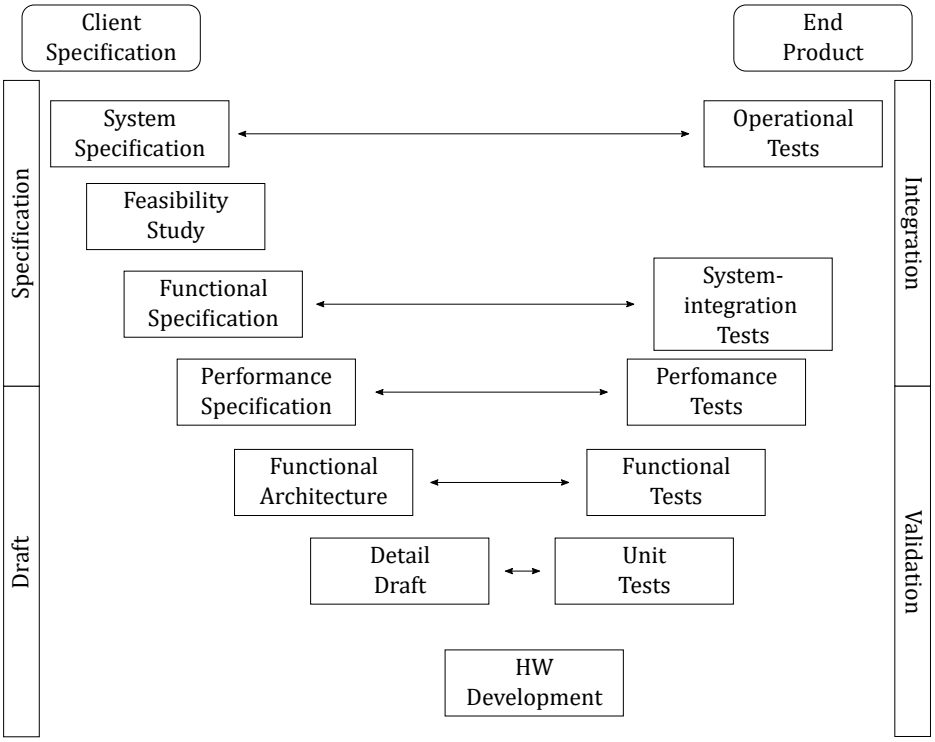


Figure 43: V-Diagram

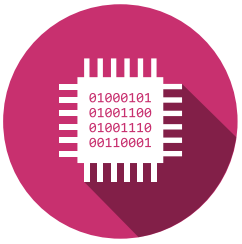
9.2 Scrumm Planning Poker

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

Weight Tasks only with specific values, mostly close to Fibbonacci numbers.

Value	Description
coffee	Omit from evaluation
0	No effort
$\frac{1}{2}$	-
1	-
2	-
3	-
5	-
8	-
13	-
20	-
40	-
100	-
?	unsure
∞	cannot be completed

Table 14: Scrum Poker Values



9.2.1 Specification Phase

9.2.1.1 System Specification

Solely created by Customer

9.2.1.2 Specification Documents

- General Specification
- Functional Specification
- Performance specification

9.2.2 Draft Phase

- Split per functions
- Architecture Document
- Small Description
- List In- Outputs and Pin definitions
- Split digital circuit into functional blocs
- Estamation of size of digital circuit
- Estimation consumation
- List critical functions
- Test strategy, Validation strategy, definition Test-I/O

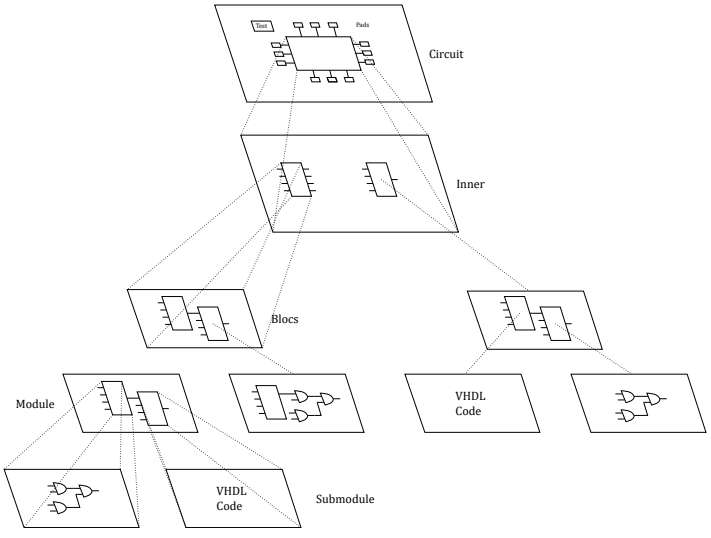


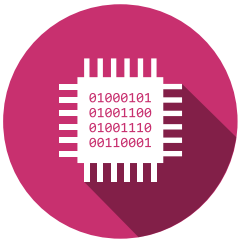
Figure 44: Bloc Diagram

9.3 Synchron vs Asynchrone Systems

9.3.1 Gate Delay Considerations

The gate delay time must be taken into account when designing the circuit.

Due to the cycle time, there is often a delay between a change at the input and the corresponding



adaptation at the output. This depends on the technology used and can vary greatly from one gate to another within the same technology. The delay also depends on the gate load.

Due to these delays, the output of a logic block may be in a different state than the input for a short moment. Such a time delay corresponds to the situation where two almost simultaneous transitions between two identical successive states occur at the block output. Such time disturbances are due to the delay times. The variation of an input signal can reach the output in two or more parallel paths, each with different delays.

9.3.2 Gate Delay Independence

Gate delay times are unreliable system parameters. It is therefore recommended that systems be designed so that they are not affected by gate delay times.

Synchronous logic is used for this purpose. A synchronous logic system is a system in which the clock signals of all toggle circuits are connected to the same command signal.

9.3.3 Sequential Logic and Gate Delay Rules

9.3.3.1 Rule 1 - Same clock and reset Any sequential logic must be synchronous. Therefore, if possible, use a single external clock generator that is only active on one edge. The most important elements of these sequential systems are the D or E flip-flop.

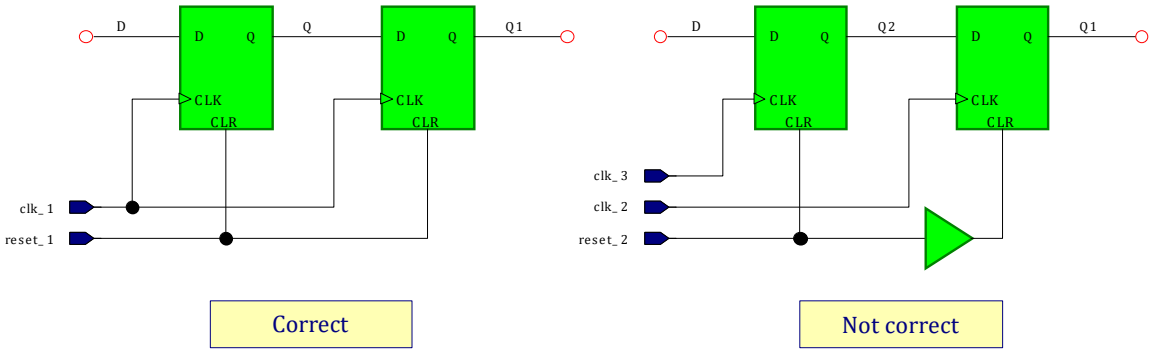


Figure 45: Gate Rule 1

9.3.3.2 Rule 2 - Don't use gate delay's It is forbidden to use the logic element cycle time to generate pulses (different cycle times of the original signal and the delay chain).

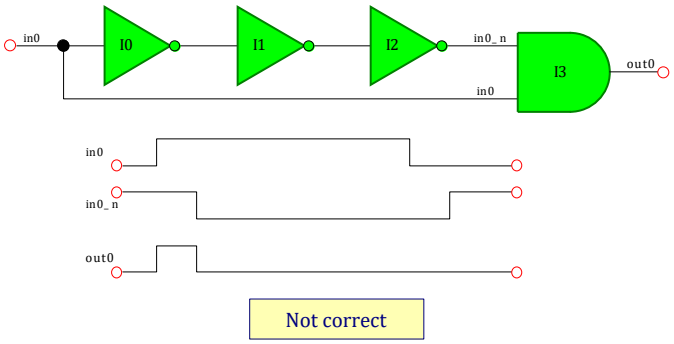
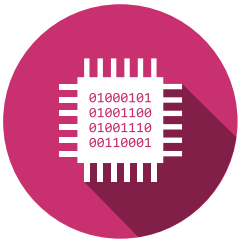


Figure 46: Gate Rule 2



9.3.3.3 Rule 3 - Clock connections It is preferable to route the clock signals directly to the flip-flop rather than controlling them with a logical element to avoid any time interference at the inputs. It is also necessary that the signals at the inputs synchronized by the clock keep the setup and hold times with respect to the clock.

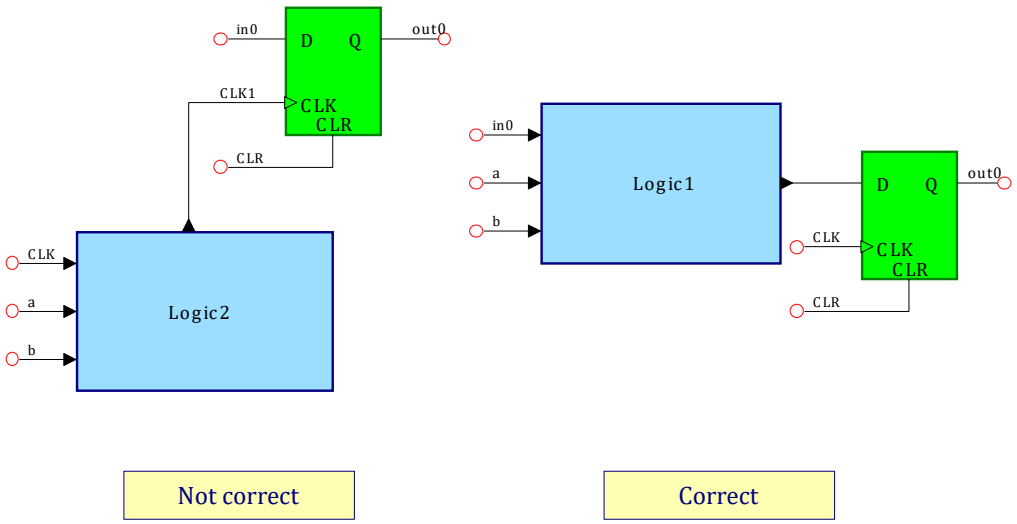


Figure 47: Gate Rule 3

Remarks: If the number of gates or consumption is to be kept as low as possible, compliance with the third rule may prove unfavourable. Asynchronous frequency dividers require much less elements than synchronous frequency dividers. If they are used together with logic elements that work at a higher frequency, the divider outputs must be re-synchronized with the higher frequency signal. Some input or output blocks require asynchronous operation to respond quickly to external conditions. In this case, the asynchronous part of the system must be kept to a minimum. It is important to intensively analyze the system's timing behavior for the asynchronous system parts and to design function blocks without gate delays.

9.3.3.4 Rule 4 - Reset connections The asynchrone inputs of a flipflop set and reset can not be used to create a fonctionality of the circuit.

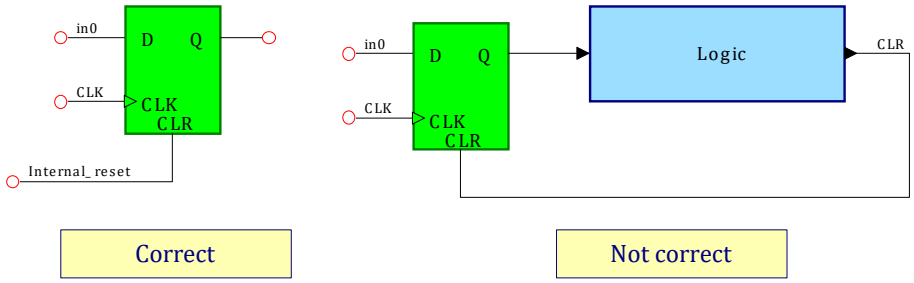
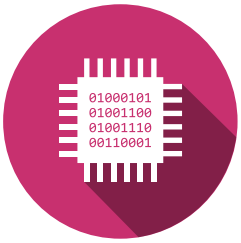


Figure 48: Gate Rule 4

9.3.3.5 Rule 5 - Initial state with set and reset's Each circuit must be set to a known state when it is fed into the circuit and at the start of simulation. The asynchronous inputs set and reset of the flipflops must be used. These inputs must not be used to operate the circuit, but only to ensure testability. Therefore, no signals with time interferences should be routed to the asynchronous inputs of the flip-flops. The set and reset signals must not originate from purely combinatorial decodings.



9.3.3.6 Rule 6 - Max clockspeed calculation The shortest period of a clock generator of a synchronous computer is calculated as follows:

$$T_{min} \leq T_{ClkQ_{max}} + TQD_{max} + T_{skew} - T_{setup_{max}} \tag{51}$$

- $TClkQ$ - is the delay time between the clock edge and the flip-flop output Q
- TQD_{max} - is the delay of the longest chain of gates between an output Q of a sequential logic and an input D of a sequential logic responsive to the same edge of the same clock
- $Tskew$ - is the clock shift with respect to the clock inputs of the sequential logic
- $Tsetup$ - is the setup time of the sequential logic

Remarks: In order to increase the functional speed of a circuit, synchronization registers can be inserted into the large gate chains.

9.3.3.7 Rule 7 - Synchronisazion of signals The input and output signals of a system must be synchronized with the aid of D-flip-flops. By means of the figure below, the time disturbances and metastable states that occur when a signal passes from an asynchronous to a synchronous system can be filtered.

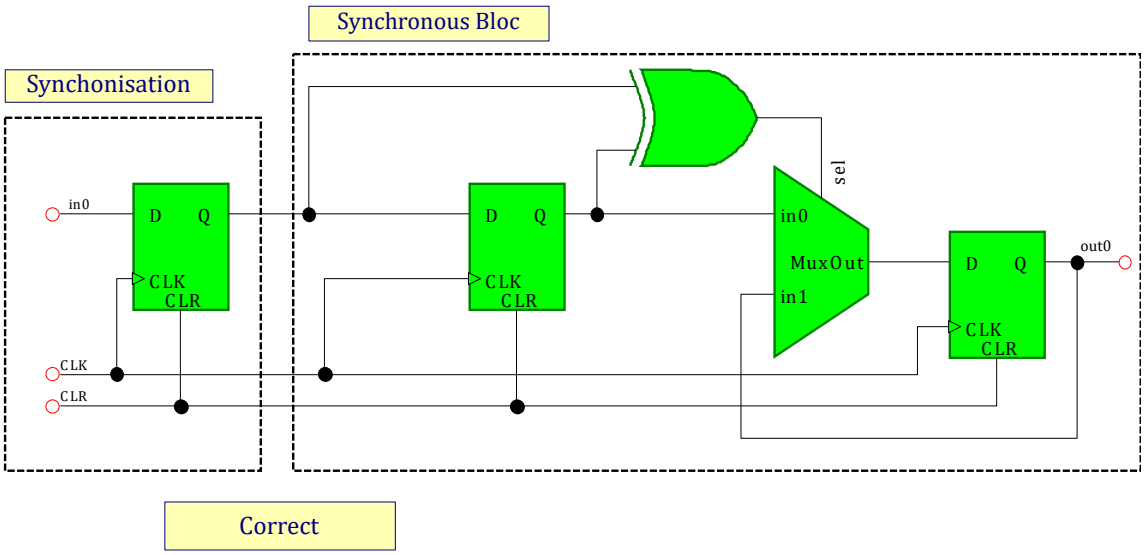
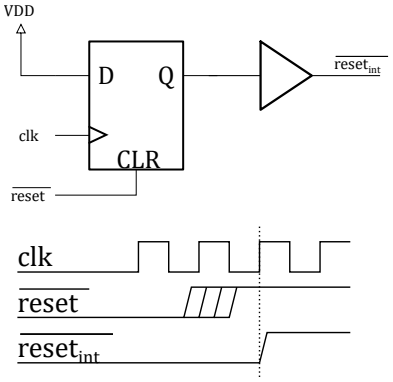


Figure 49: Gate Rule 7

9.3.3.8 Rule 8 - synchronize asynchron reset The disappearance of the internal initialization signal of the circuit must be synchronous to the clock, but its appearance is asynchronous.



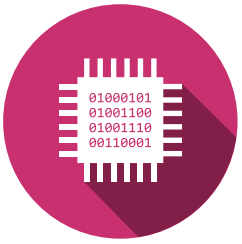


Figure 50: Gate Rule 8

9.3.3.9 Rule 9 - Gate fan-out Do not overload the gate outputs too much. For large signals, estimate the fan-out of the gate that generates it and the fan-in of the gate that must control it.

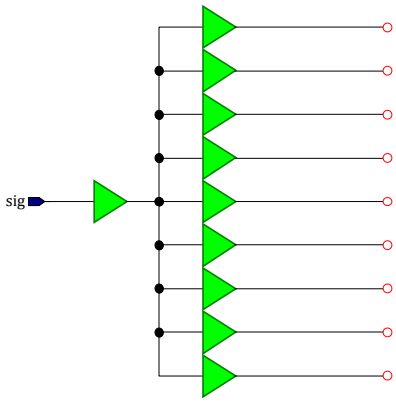


Figure 51: Gate Rule 8

9.4 Verification Phase

9.4.1 Validation

Do we create **THE** correct product?

9.4.2 Verification

Do we create a correct product?

9.4.3 Techniques

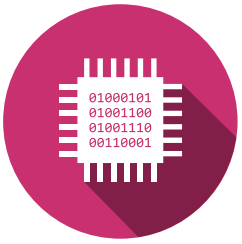
- Simulation of VHDL code
- Examination of the design documentation by competent auditors
- Mutual proofreading of the VHDL code by the various team members

9.5 Integration Phase

9.5.1 Results

The following elements needs to be part of the document:

- A detailed design report with a detailed description of the design activities, the coding and verification, the validations of the partial circuits as well as the integration and final validation. The focus must be on the conscientiousness of the validations performed and on the elements that will allow another team to continue the project at a later stage.
- User manual. This is the document that is delivered to the customer (end user). It must describe the installation, settings and use of the material. This document is actually written in the course of the project, as it adopts the model developed during the general design phase (which is based on the functional description of the product created during the specification phase).



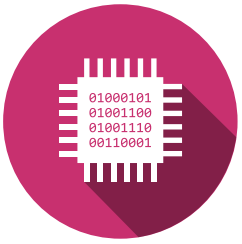
Identificationnumber of the Specification request	Description	Testbench and Simulation Setup	Time of validation	Validation Method	Validation Status	Attachment
DSxx-YYXXX DSxx Number in Specification document YY Chapter number XXX Number of Functional Specification	Keywords of Functional Specification to be validated	Identification of used Testbench and Simulation Steps	Time when the validation took place	Validation Method (Analyses, visual verification, Print, Log file, automatic validation	Validation Status (OK | NOK | measured Value)	Number of attachment if needed
(1)	(2)	(3)	(4)	(5)	(6)	(7)

Table 15: Verification Report

9.5.2 Execution Techniques

- Scheduling problems: The delays accumulated in the previous phases will inevitably become noticeable as the deadline approaches. Participants therefore often panic instead of focusing on quality and organisation. There is only one solution: PREVIEW, PREVIEW, PREVIEW. Whether the deadline is met is already decided during the specification phase. The developers and especially the project managers have to be very careful, especially in the initial phase (the two weeks that are missing at the end are often lost at the beginning). By conscientious project monitoring, catastrophes can also be avoided by redefining objectives in good time.
- Managing the work of the group: Many reflections of the previous phases can and/or must be carried out together in the group. However, this phase is characterised by a lot of individual work. The work can only be done if it is managed in parallel. But the simple addition of individual tasks does not automatically lead to their sum. This requires organization. Therefore, a structure must be set up for the distribution of tasks among team members and the time required for synchronization. The larger the group, the more precise the organization needs to be and the longer the synchronization takes. This time must be explicitly taken into account during planning. For a group of 5 to 6 people working full-time on a project, a daily briefing (coffee break) and a structured weekly meeting seem to be a realistic solution.

See also Scrum Development



10 FSM - Statemachines

10.1 General

AI synchronous systems need to comply with the following conditions:

- All Flipflops use the same clock signal source
- No Flipflop has asynchronous inputs

10.2 Moore Machines

- Flipflops save the internal state
- combinatorial logic bloc determines the future inner state depending to the current inner state and the current inputs
- combinatorial logic bloc determines the output signals depending on the current inner state

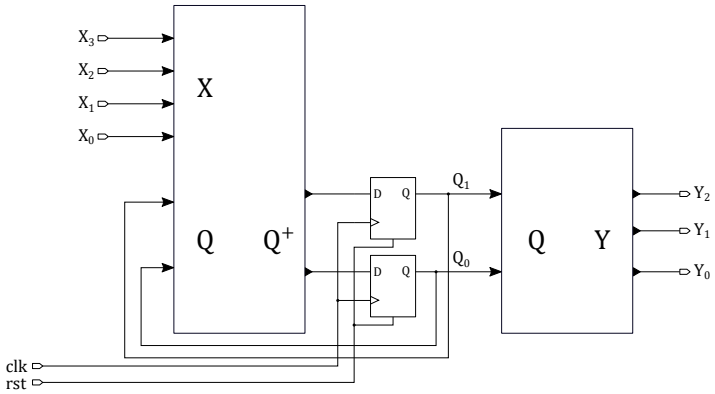


Figure 52: Moore State Machine

10.2.1 Example counter

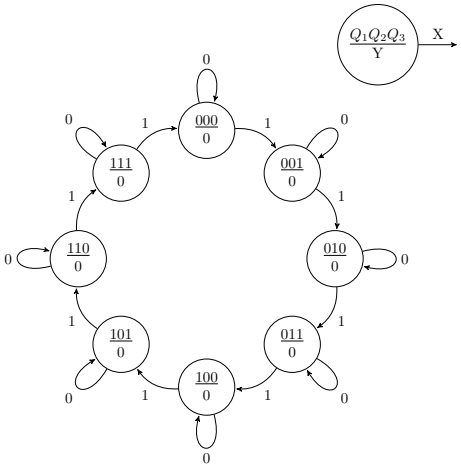


Figure 53: Moore Counter

10.3 Mealy Machines

In contrast to Moore machines, Mealy machines have outputs that can also depend on the inputs.

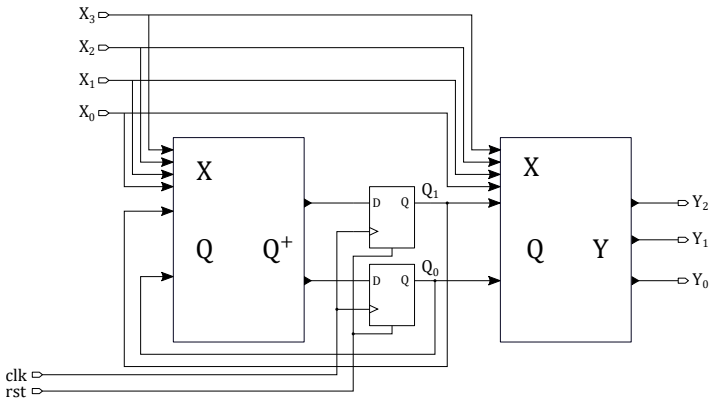


Figure 54: Mealy State Machine

10.3.1 Example Counter

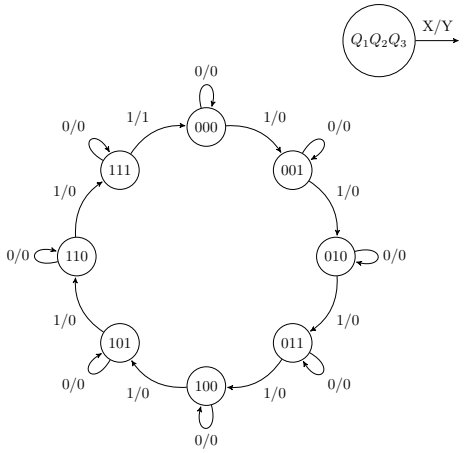


Figure 55: Mealy Counter

10.4 Timing Mealy vs Moore

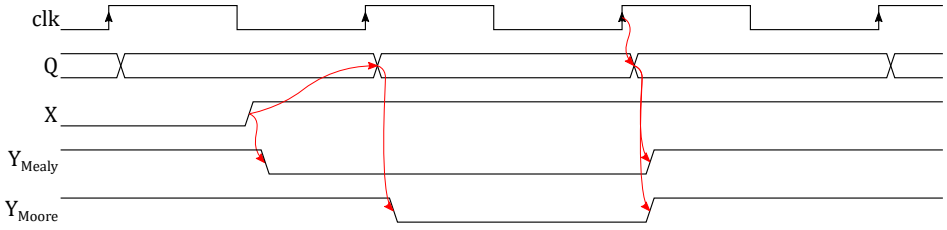
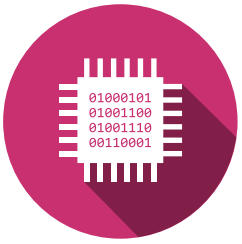


Figure 56: Mealy vs Moore Timing

10.5 Creation of a Finite State Machine

The state machines can be designed as follows:

- Expressing the specification by means of a state graph,
- Fill in the status table,
- Reduction of the number of states,



- Coding of the states,
- Realization of the logical circuit.

The most delicate thing is to develop a state graph based on the system's specifications. For this purpose 3 methods are listed below.

10.5.1 Development from a random state

One way to create a state graph is to proceed from state to state. To do this

1. a certain state of the machine can be defined.
2. For this state, all possible input conditions (2^n cases for n inputs) must be analyzed and the corresponding arrows drawn. Depending on the case, new states are created.
3. Gradually carry out all not yet treated states, analyze all possible input conditions (2^n cases for n inputs) and draw the corresponding arrows. Each time a new state is created, it must be checked whether it already exists in the developing graph. Repeat this step until the graph is complete.

Remark There may be situations in which certain input conditions cannot occur. Significantly, this is the case when several input signals cannot change simultaneously due to the structure. In this case less than 2^n arrows lead away from each state.

10.5.1.1 Example Debouncing Circuit

Figure 57: Debouncing FSM 1

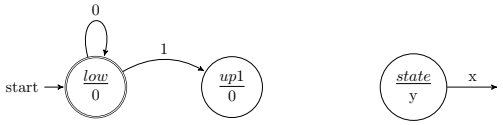


Figure 58: Debouncing FSM 2

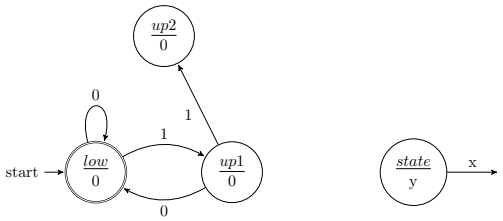


Figure 59: Debouncing FSM 3

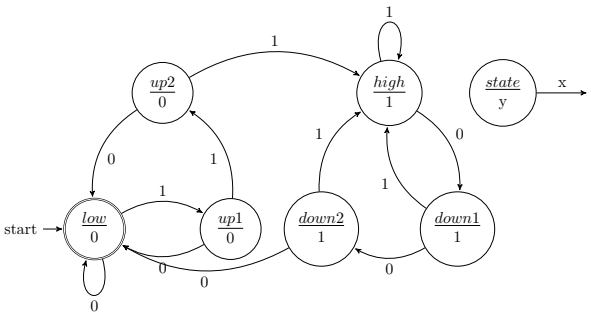


Figure 60: Debouncing FSM 4

10.5.2 Development from a scenario

The second way to develop a state graph is to define a scenario. To do this

- A scenario can be defined that covers a large part of the requirements specification.
- The scenario must be developed in the form of a graph, with the states corresponding to the stages of the scenario and the arrows corresponding to the actions.
- For each state defined in this way, all possible input conditions (2^n cases for n inputs) must be analysed and the arrows not yet present drawn. Sometimes new states arise, which in this case have to be analyzed completely.

10.5.2.1 Example Debouncing Circuit

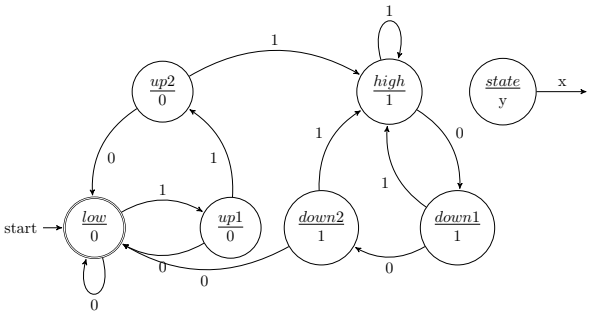


Figure 61: Debouncing FSM 4

10.5.3 Development from a State list

In order to develop a state graph in the third way, its states must be known in advance. For this

- the events or states to be stored are determined.
- A state is drawn for each event to be stored.
- For each defined state, all possible input conditions (2^n cases for n inputs) are analyzed and the corresponding arrows are drawn.

10.5.3.1 Example Debouncing Circuit

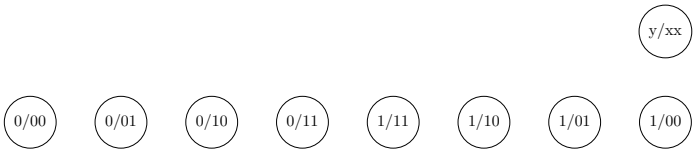


Figure 62: Debouncing FSM 5

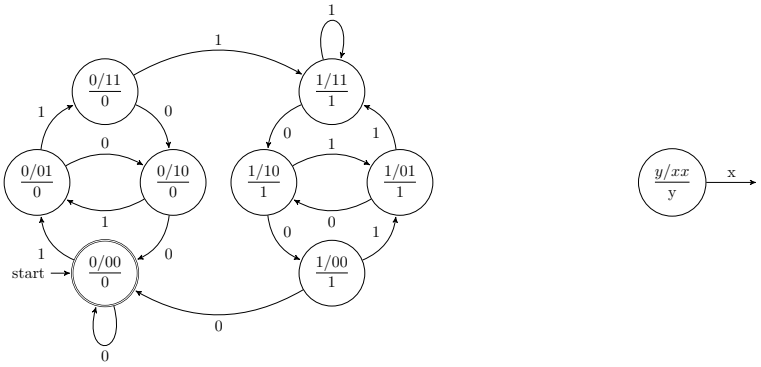
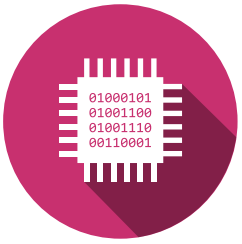


Figure 63: Debouncing FSM 6


01000101
01001100
01001110
00110001