



Microcontroller Techniques

Winter Term 2016/2017

Prof. Dr. L. M. Reindl

Laboratory for Electrical Instrumentation
Department of Microsystems Engineering – IMTEK
University of Freiburg

UNI
FREIBURG

Your instructor

Prof. Dr. Leonhard Reindl

- 1985 Dipl.-Phys. at TU München
- 1985 cooperate research and technology (Siemens AG)
- 1997 PhD at TU Wien
- 1999 lecturer at TU Clausthal
- since 2003: professor and head of the Laboratory for Electrical Instrumentation at IMTEK, University of Freiburg



Contact Data

Prof. Dr. Leonhard Reindl

phone: 0761-203-7221

e-mail: reindl@imtek.de

office space: building 106, room 04-014

There are no set call times! Please make an appointment with the secretary (0761-203-7220) or write an e-mail.

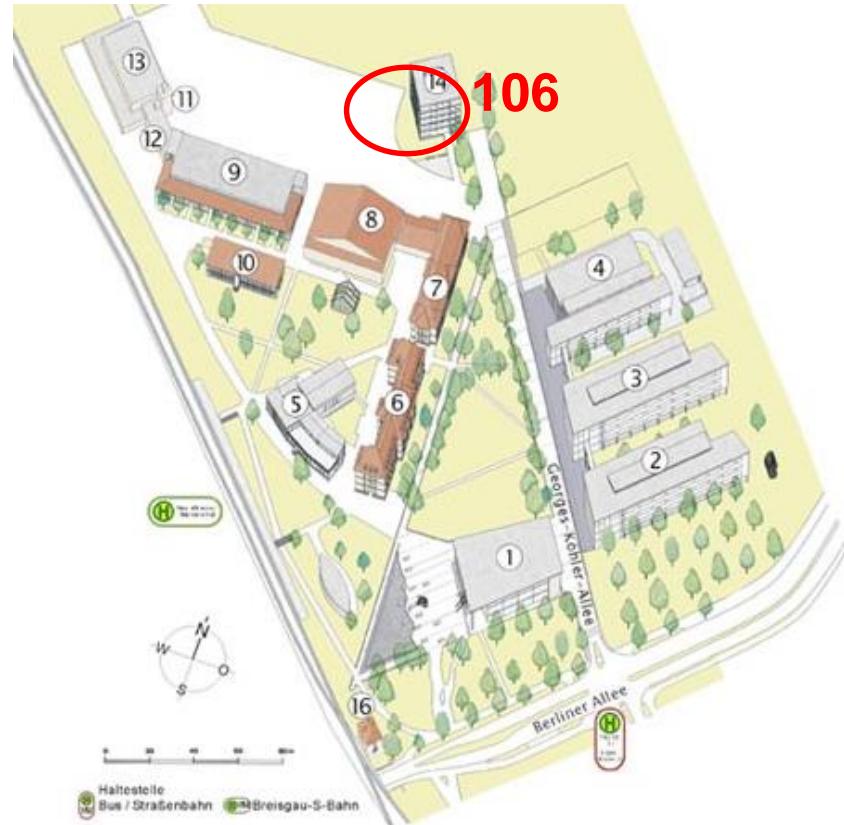
Sebastian Stöcklin

contact person for lab course & exam

phone: 0761-203-7157

e-mail: sebastian.stoecklin@imtek.uni-freiburg.de

office space: building 106, room 04-004



ORGANISATIONAL

Learning content and scope of the lecture

The lecture and the practical course are just a teaser content of the field of microcontroller technology! It is not possible to treat these contents in full depth in one semester. Therefore, it is strongly recommended to rework the given topics with a textbook to obtain a deeper understanding.

Overview of the course

■ Lecture

- contents: theoretical basis & material collection about microcomputer theory
- approx. 10 lectures (participation and rework recommended)
- exam tutorial session(date & room: will be announced)
- final exam - date & room: will be announced

■ Practical exercises

- contents: hands-on experience with microcontrollers (MSP430) & C programming
- exercises will be performed at home, hardware available from the library of the technical faculty
- 8 out of 9 exercise sheets must be passed with more than 50% of the points
- introduction course on 26th of October (see e-mail)

Objectives of the course

- Basic understanding of modern data processing systems
 - digital logic
 - binary arithmetics
- Structure of microcontrollers (hardware)
 - microprocessor
 - peripherals
- Programming of microcontrollers controller (software)
 - (assembler language)
 - C
- Evaluation and selection of microprocessors
- Practical skills in systems programming in the laboratory course

Permitted aids

- Non-programmable calculator
- C command summary for programming task (provided by us)

The nature and extent of the exam

- approx. 7 to 10 tasks, 120 minutes of time.
- both theoretical tasks (based on the knowledge imparted in the lecture) and practical tasks (based on the exercises)

What's relevant?

- Basically, all contents of the lecture and the practical exercise are relevant!

Forums, training portal, etc.

- ILIAS: ilias.uni-freiburg.de
Course „Microcontroller Techniques“
 - Password „Lab20162017“
- Slides of the lecture, materials for the practical course, submission of exercise sheets, etc.
- Registration until 31th of October, 2016

Additional literature

- John Davies, „MSP430 Microcontroller Basics“, Springer, 2008.
- Matthias Sturm, „Mikrocontrollertechnik: Am Beispiel der MSP430-Familie“, Hanser, 2011.
- Marian Walter, Stefan Tappertzhofen, „Das MSP430 Mikrocontroller Buch“, Elektor-Verlag Aachen, 2011.
- Th. Flik., „Mikroprozessortechnik (RISC, CISC, Systemaufbau, Assembler und C)“, Springer, 1998.
- Beierlein Th., Hagenbruch O.: „Taschenbuch Mikroprozessortechnik“, Fachbuchverlag Leipzig, 2004.

Introduction

OVERVIEW OF THE LECTURE

Structure of the lecture

- Introduction
- Chapter 0: basic knowledge
 - number systems, digital logic, ...
- Chapter 1: calculation and control systems
 - basic logic operations in CMOS, registers, memory, concepts, ...
- Chapter 2: Basic peripherals
 - GPIOs, ADCs, DACs, ...

Structure of the lecture

- Chapter 3: Advanced Peripherals
 - watchdog, programming interfaces, ...
- Chapter 4: Programming
 - assembler language, C, compilers, ...
- Chapter 5: Communication concepts
 - OSI model, topologies, UART, I2C, ...
- Chapter 6: Outlook (among others: safe systems)

Introduction

MOTIVATION

Why do we need microcomputers and -controllers?

These days, it is hardly conceivable to have equipment and processes without computer assistance. Microsystems technology provides sensors and actuators which are most useful with the help of computers, being integrated into the system to control and regulate a process.

Motivation

Forecasts characterized by terms such as

- Post-PC era
- Ubiquitous computing
- Pervasive computing
- Ambient intelligence
- Industry 4.0



© P. Marwedel, 2011

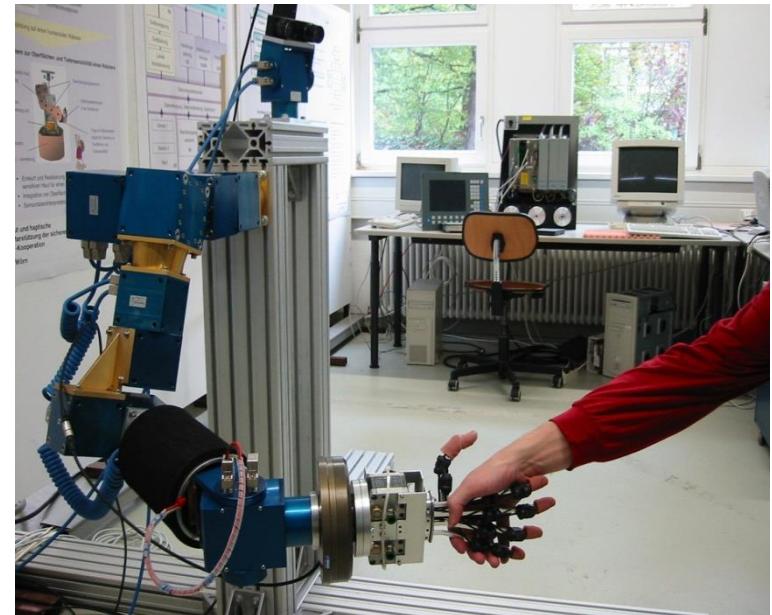
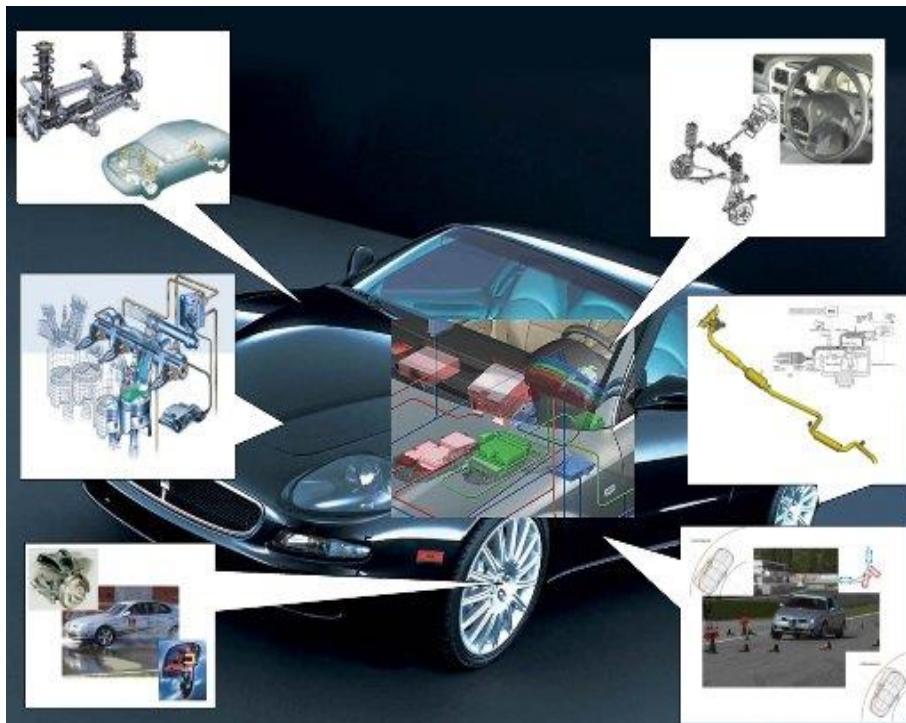
... are defined as information processing systems that

- are directly integrated into a surrounding product
- are controlling and monitoring this environment
- are adapted on the task, among other things in terms of
 - performance
 - power consumption
 - costs

Examples of embedded systems



Examples of embedded systems



Examples of embedded systems



Examples of embedded systems

Curiosity

- 256 MB DRAM
- 2 GB flash memory (to recognize errors and do trouble shooting)
- 256 KB EEPROM
- RAD750 CPU
 - Up to 200 MHz



Introduction

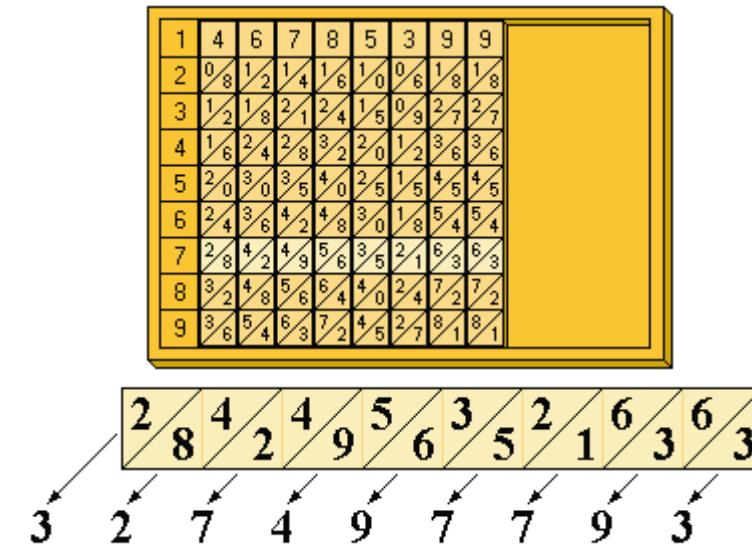
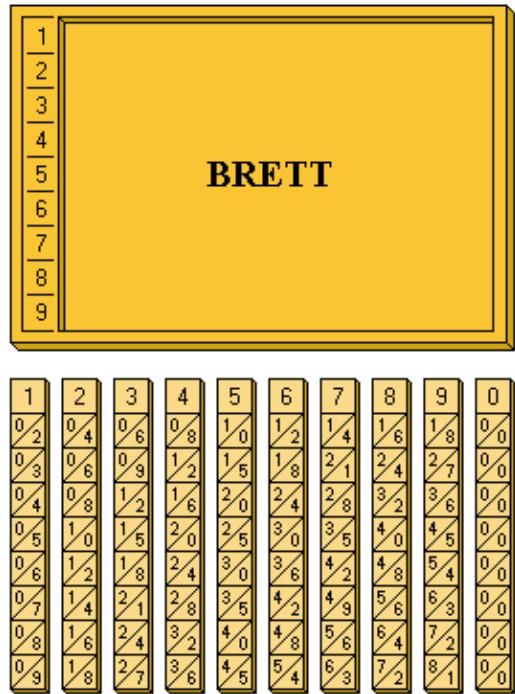
HISTORY



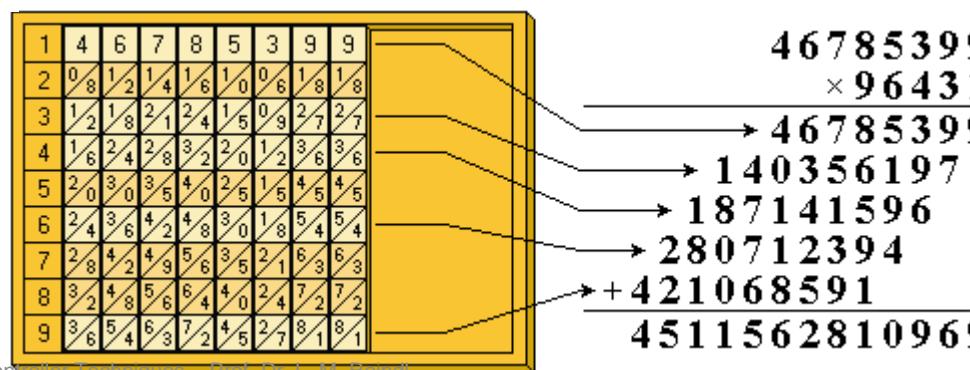
Adam Riese's "Auf den Linien", which is equivalent to the Abacus

Basics - Napier's bones (1617)

$$\begin{aligned}
 7 \times 1 &= 7 \\
 7 \times 2 &= 14 \\
 7 \times 3 &= 21 \\
 7 \times 4 &= 28 \\
 7 \times 5 &= 35 \\
 7 \times 6 &= 42 \\
 7 \times 7 &= 49 \\
 7 \times 8 &= 56 \\
 7 \times 9 &= 63
 \end{aligned}$$



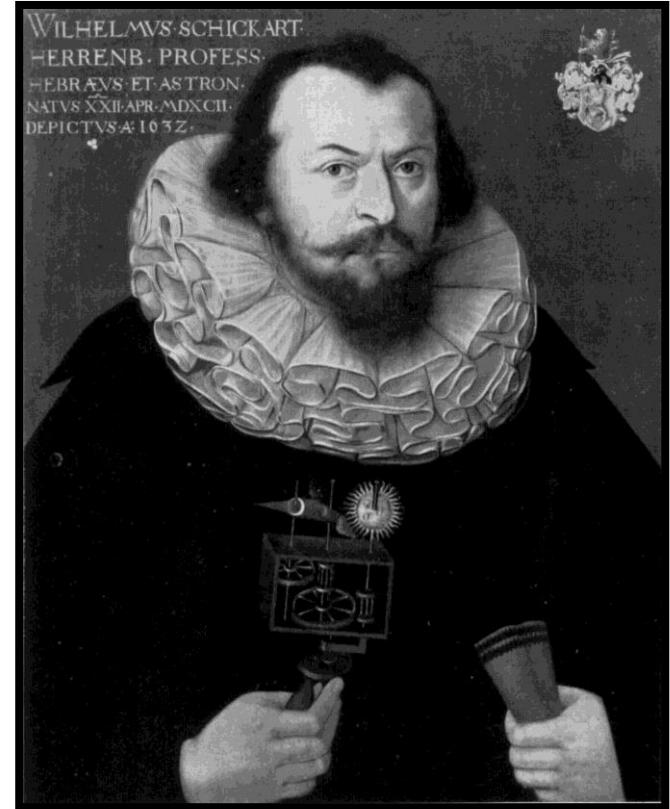
Computation: 7×46785399



Basics - calculating clock

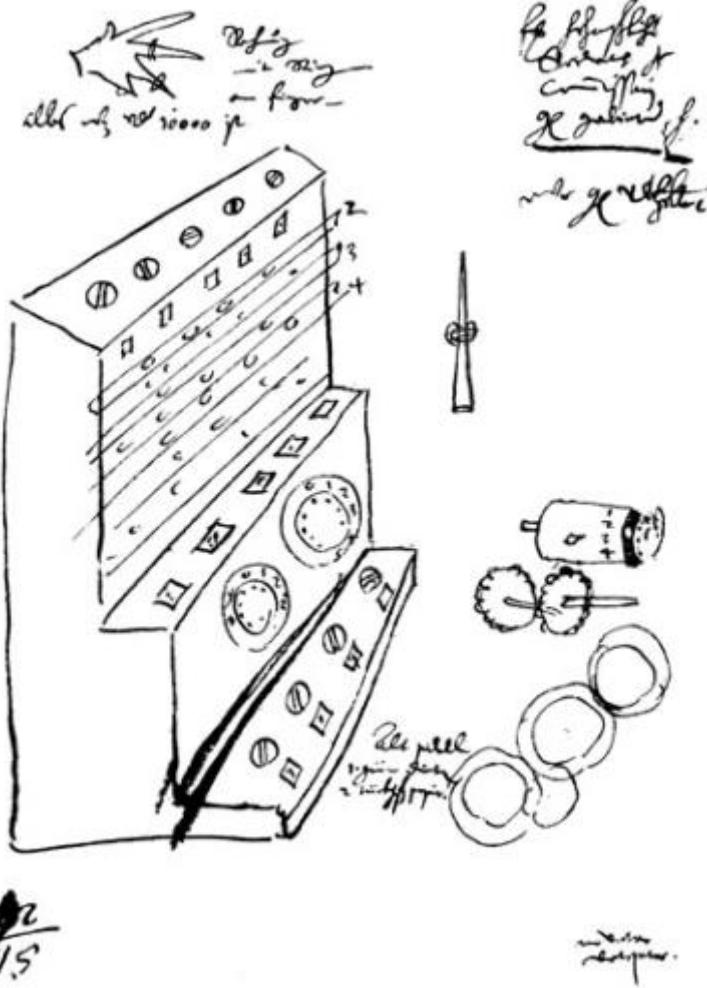


Replica of "calculating clock" of Schickard



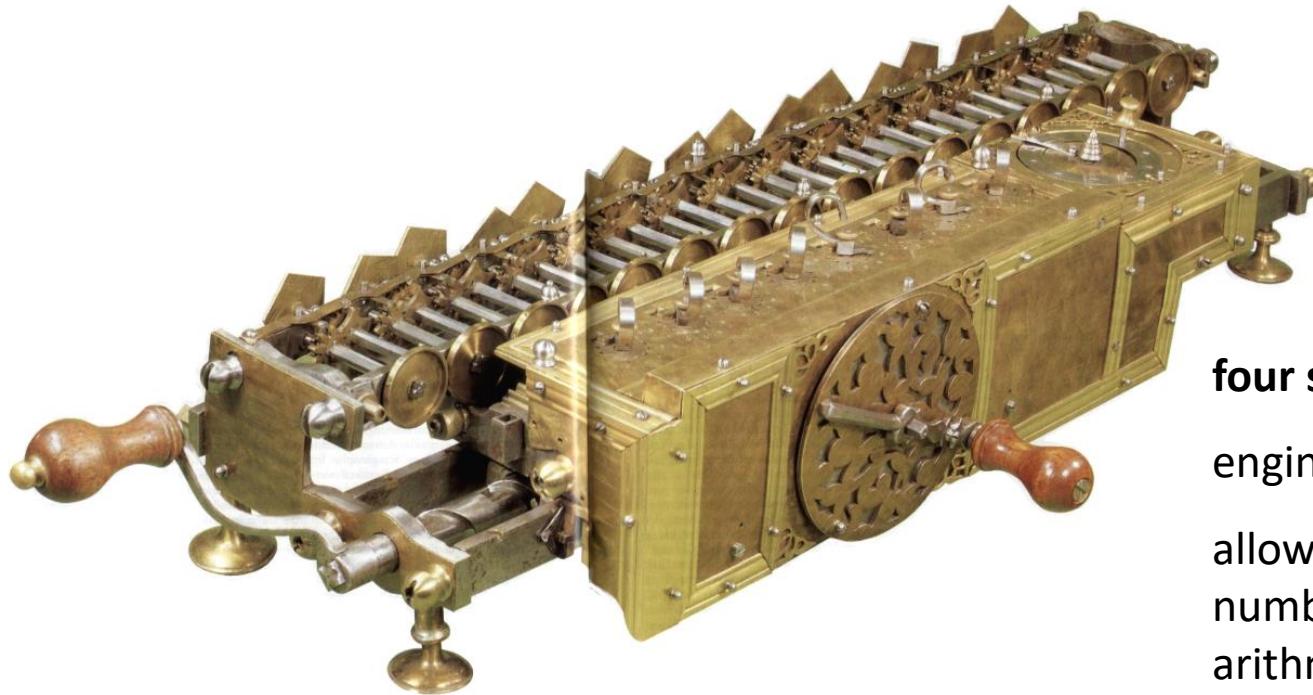
Wilhelm Schickard 1592 - 1635

Basics - calculating clock



Original drawing of "calculating clock" of Schickard

- Allows summation or subtraction of numbers with up to six digits
- "Buffer overflow" indicated by ringing of a bell
- Napier's Bones for complex calculations



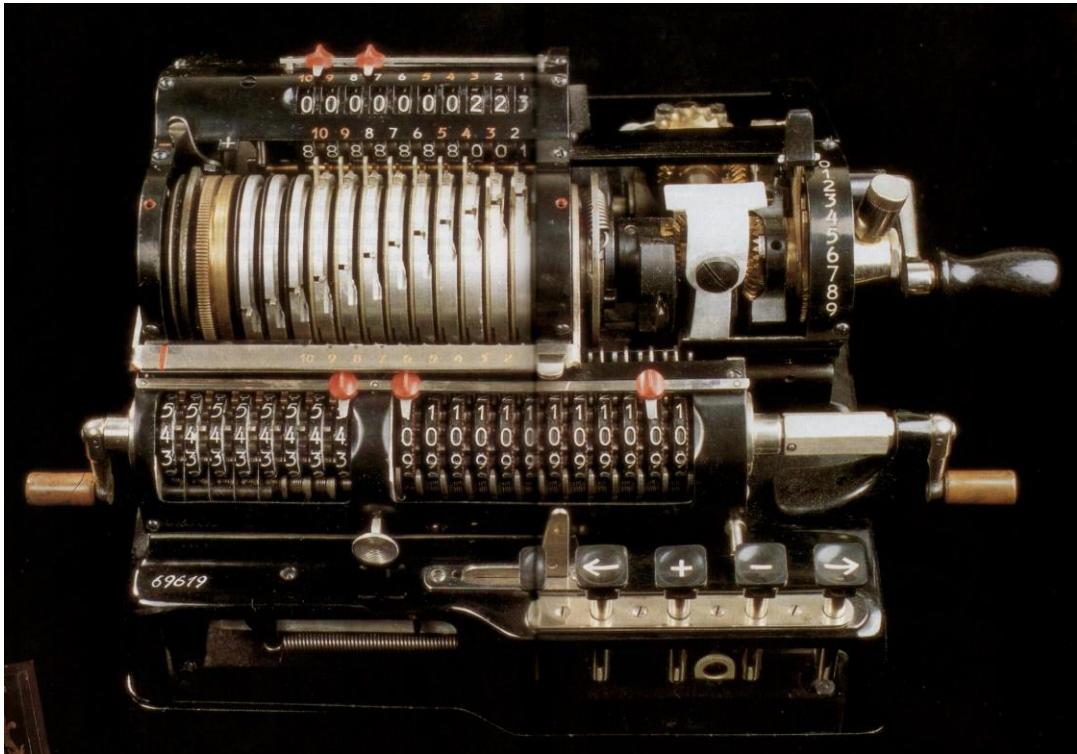
four species Abacus

engineered by Leibniz in 1673

allows the processing of 8-digit numbers using the four basic arithmetic operations addition, subtraction, multiplication and division

The machine failed during demonstrations in London because of manufacturing tolerances.

Basics - Brunsviga



Brunsviga Matador, 1905

Rechenmaschine „Brunsviga“
Modell 1905

rechnet immer fehlerlos

Additionen, Subtraktionen, Divisionen, Multiplikationen, Zinsrechnungen, Kalkulationen, Münz-, Mass- und Gewichtsrechnungen, Lohnberechnungen, Gleichungen, Quadratwurzel, Kubikwurzel etc.

Grösste Zeitersparnis.

Grimme, Natalis & Co.
C.-G. a. A.
BRAUNSCHWEIG.



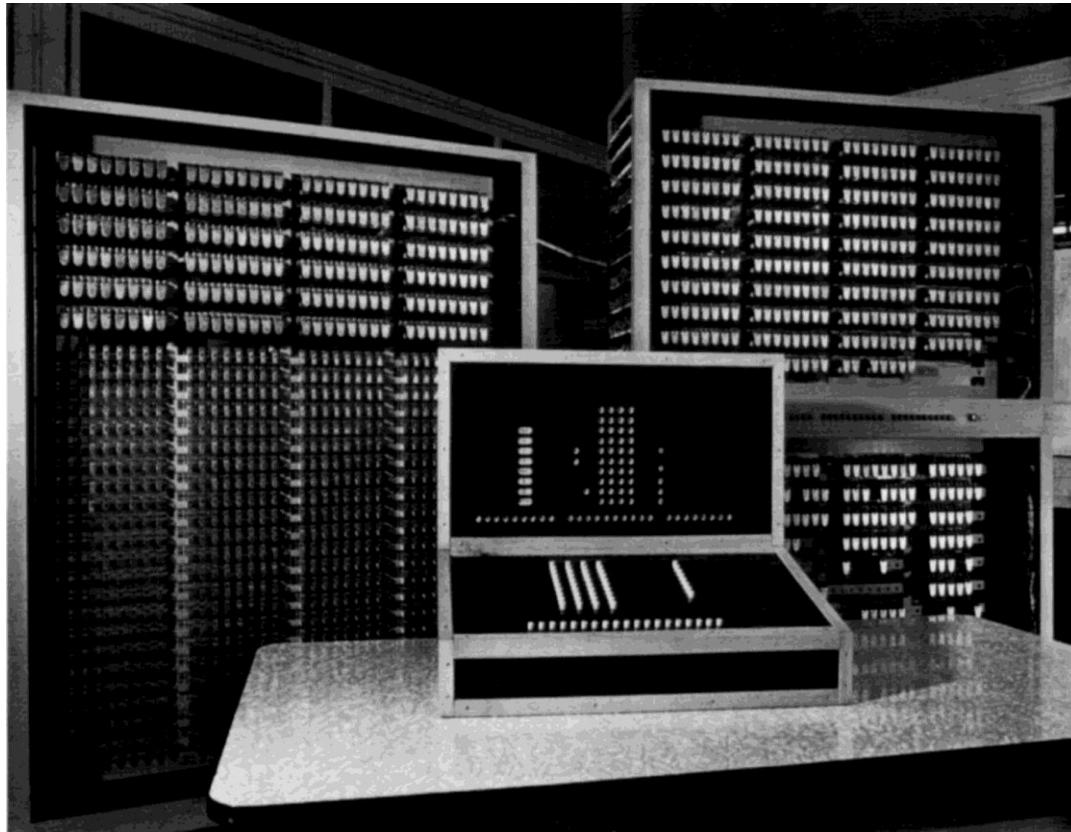
Konrad Zuse

* 22. June 1910 (Berlin)
† 18. December 1995

civil engineer

inventor of the (modern)
computer

Zuse became aware of the
works of Charles Babbage, Alan
Turing or Howard Aiken not
before the end of the Second
World War.



Zuse Z3, Year 1940

first operable digital
computer with
programmability

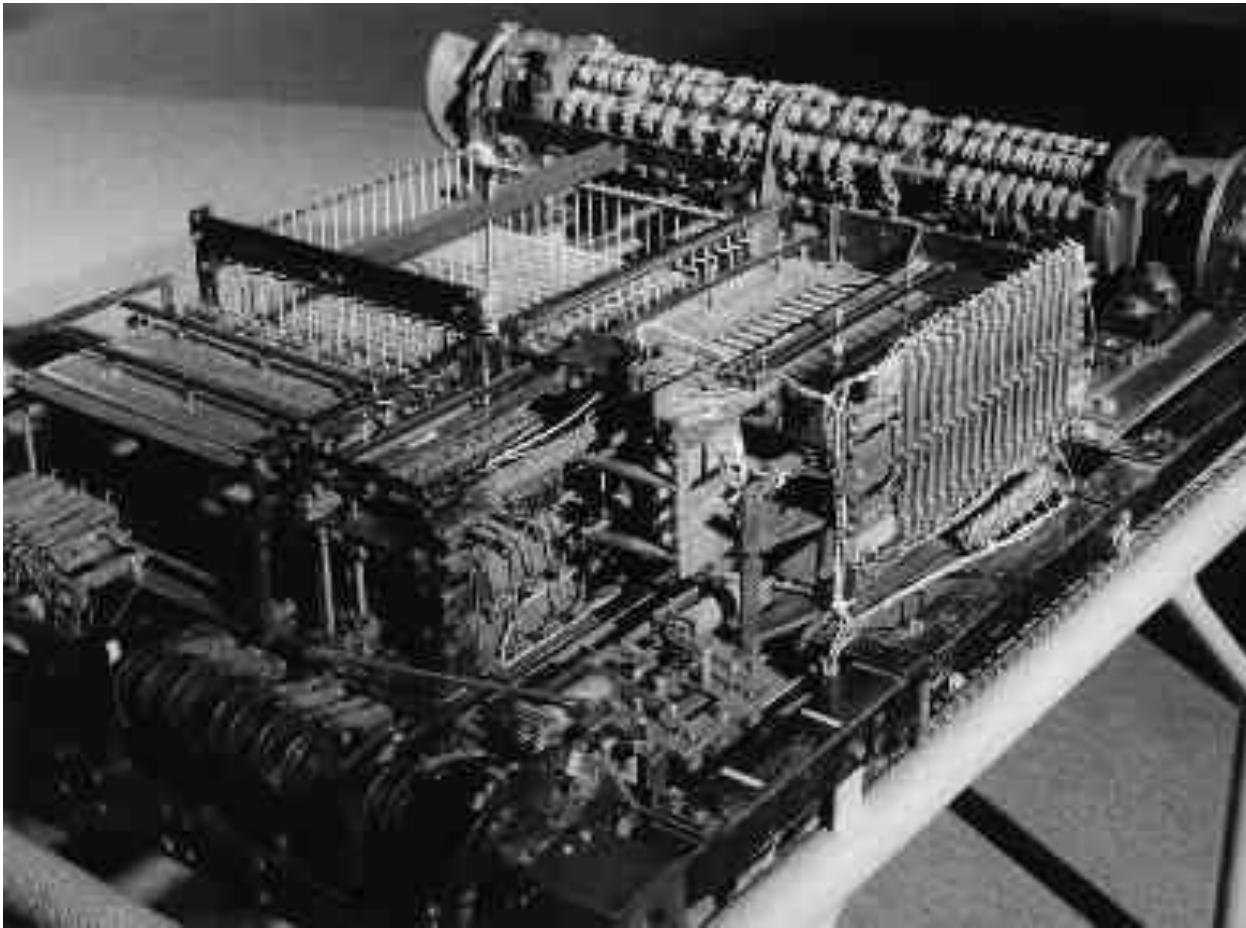
Calculator: 600 relays
Memory: 1600 relays
calculates in dual system
floating point arithmetic
square root calculation lasts
about 4 seconds



Zuse Z4, Year 1945

weight: 2,5 t
2200 relays total

4 kW of electric power

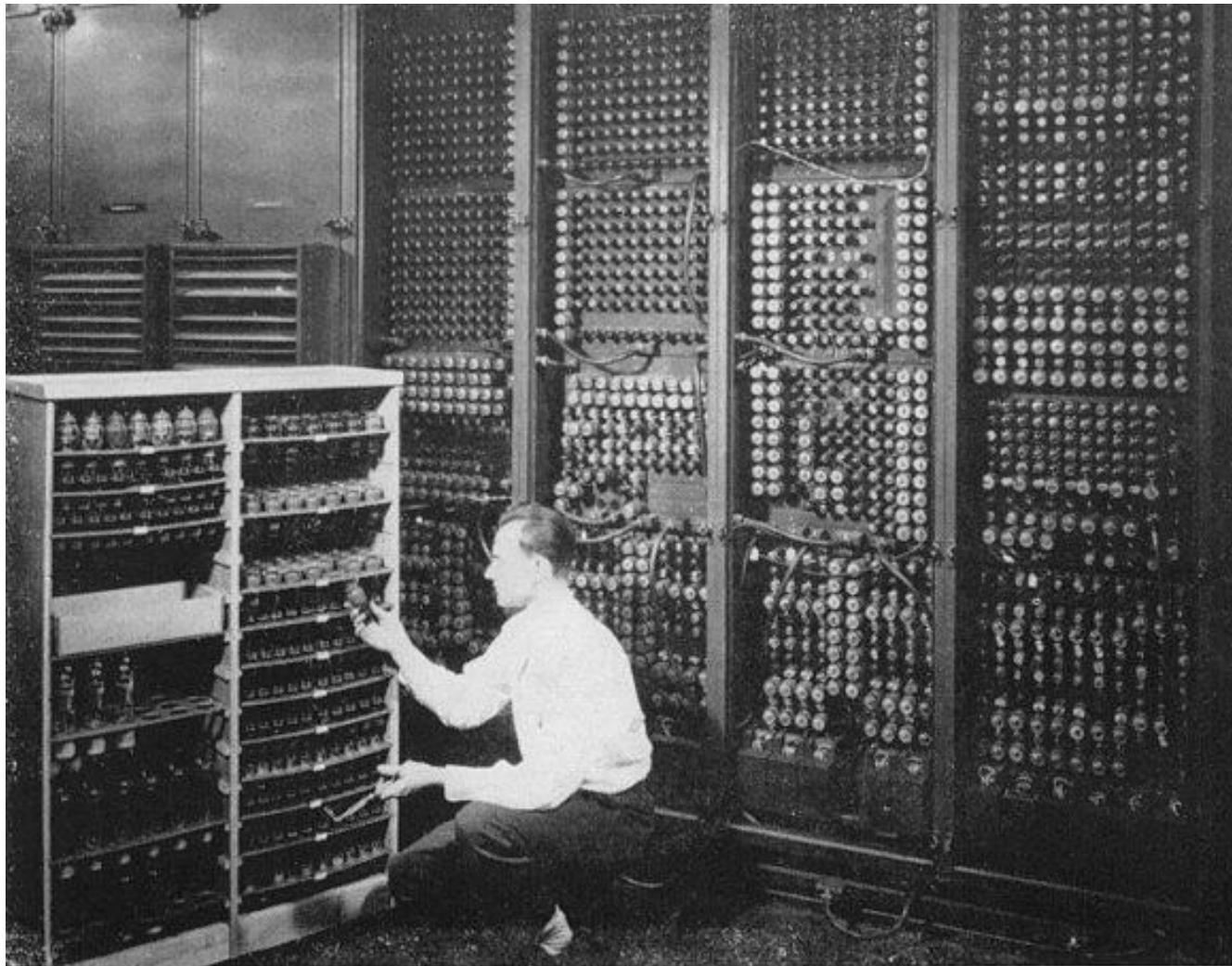


mechanical memory Z4
for 64 floating point words with 32 bit each



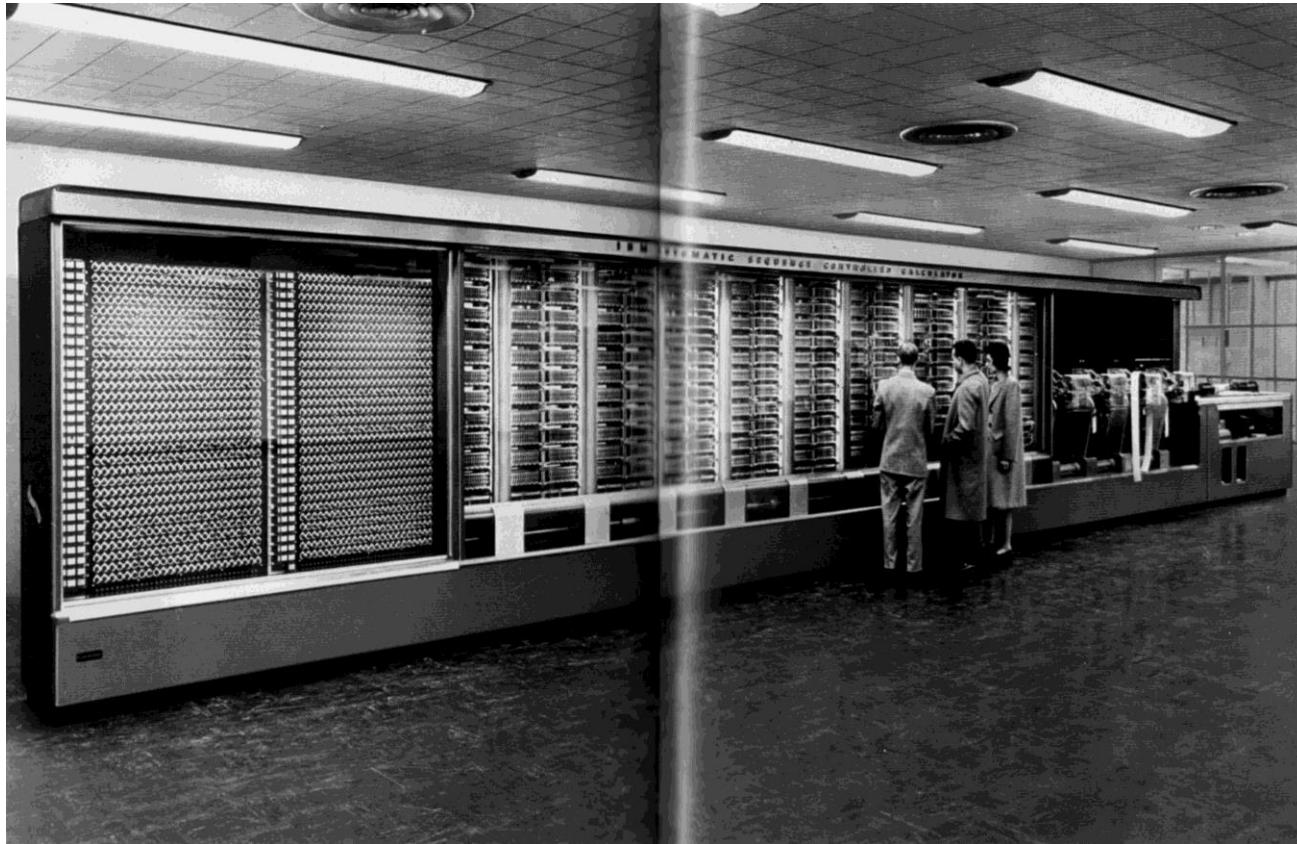
Z22, 1957, tube computer. addition 0,6 ms, multiplication 10 ms, still operable
(55 copies were sold; Image Museum of Technology Berlin)

- Electronic Numerical Integrator And Calculator
- In the early 40s (1944), the ENIAC was developed under the direction of J. Presper Eckert and John Mauchly at the University of Pennsylvania
- 17.468 vacuum tubes
- 30 tons
- 200 kW
- Performance equivalent to four species calculator (+, -, ·, :)
- clock speed of 100 kHz
- multiplication requires 3 ms
- Purpose of the development was the computer aided design of ballistics (e.g. calculating the trajectories of projectiles) for the U.S. Army in WW2.



Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

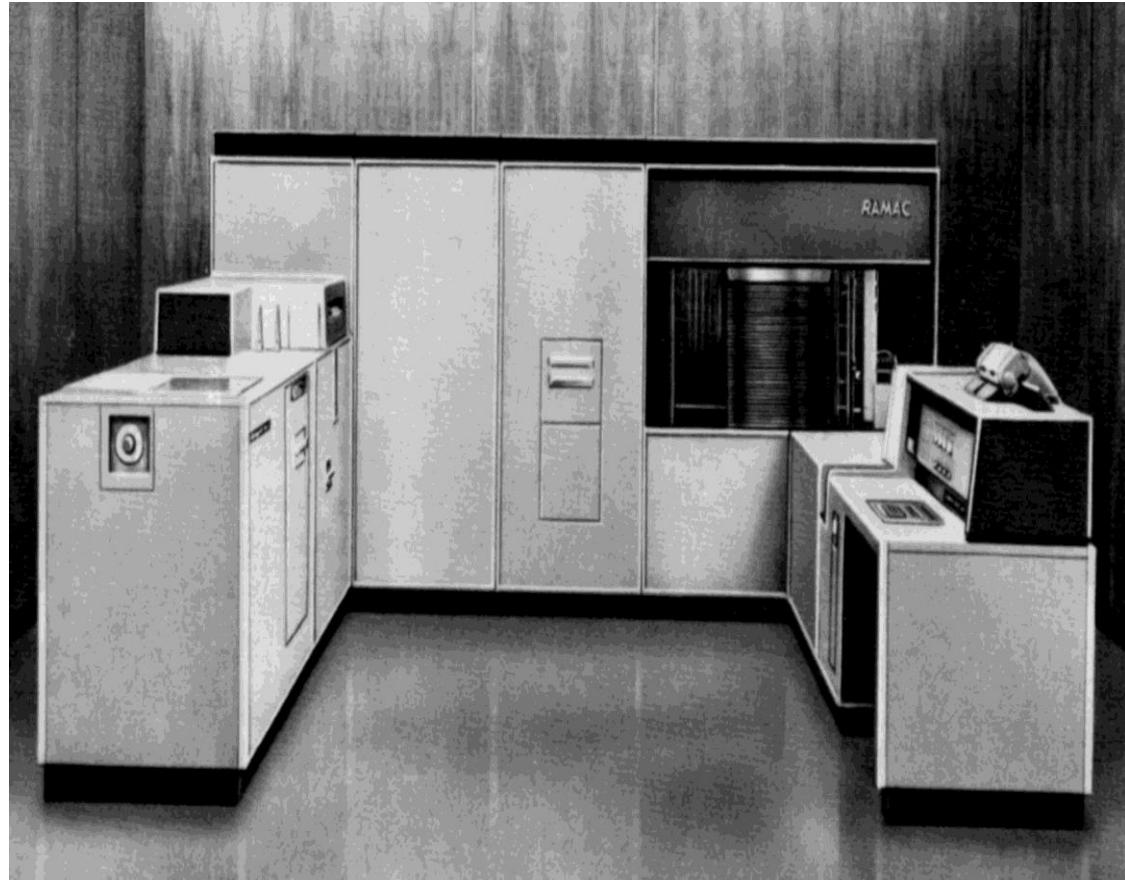
Microcontroller Techniques – Prof. Dr. L. M. Reindl



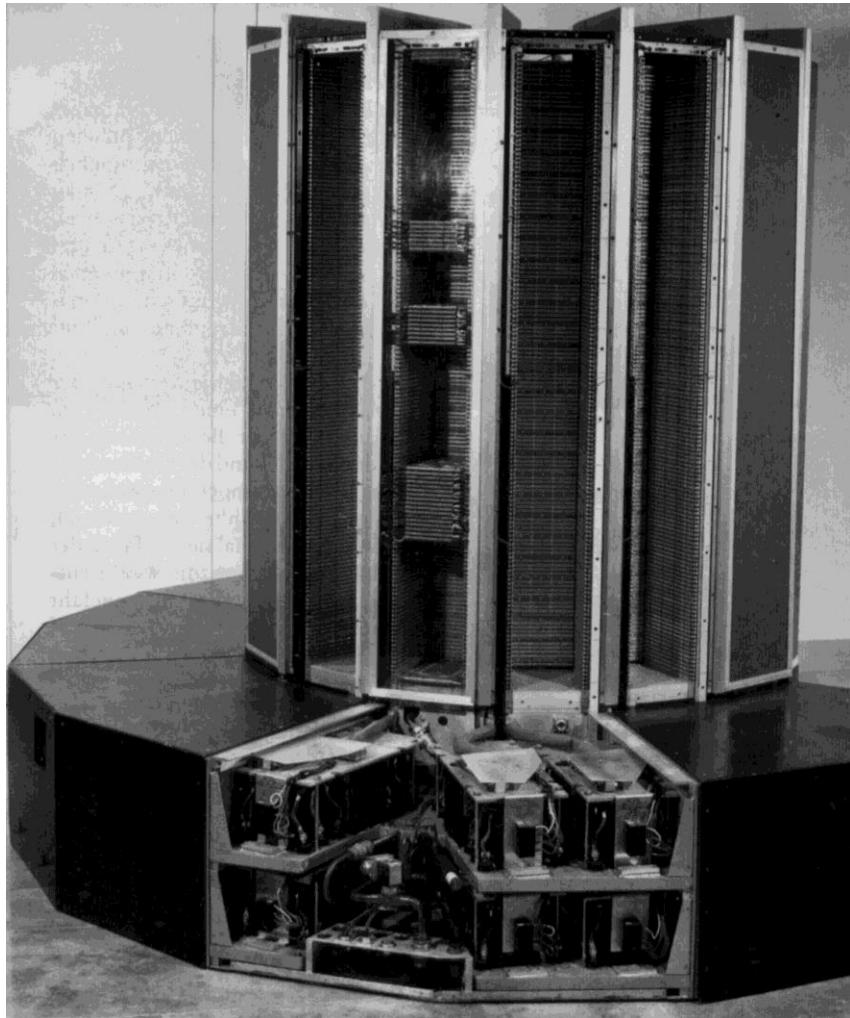
Harvard Mark I, the first stored-program computing machine of the United States, 1944, 23-digit decimal numbers, 16 m long, 2.5 m high



Harvard Mark IV, built by Aiken in 1952;
decimal computer with magnetic drum, 4000 cells, clock 16 kHz, 4000 tubes



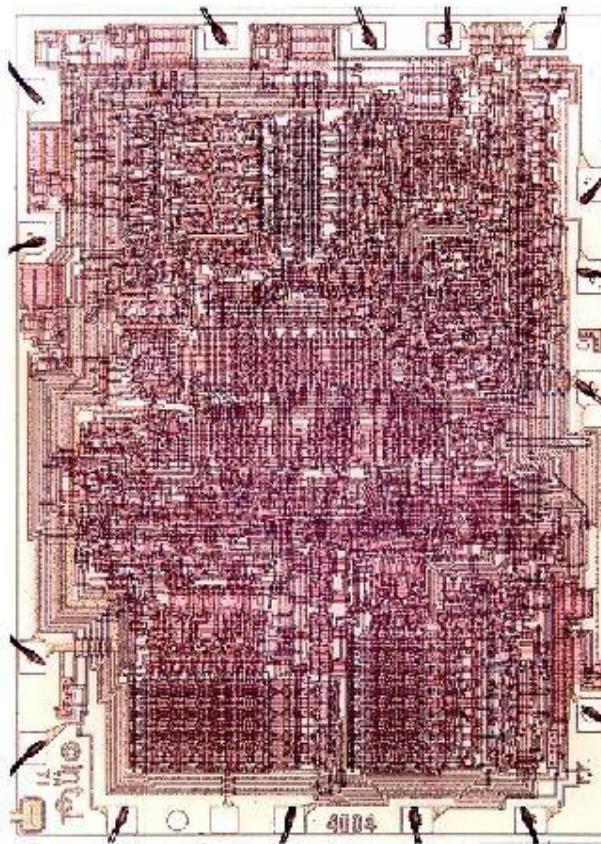
IBM 305 "RAMAC", in
use until 1963,
33 additions per
second, 3500 tubes,
1250 relays,
ferrite core memory
100 places



Cray-Supercomputer
first Version was launched in 1976

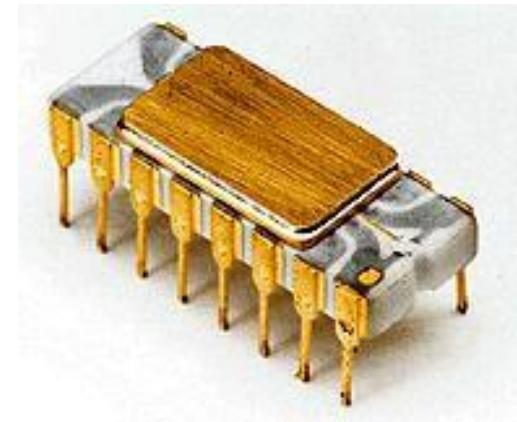
Intel 4004 (1971)

- 2300 Transistors
- word length of 4 bit
- up to 740 kHz



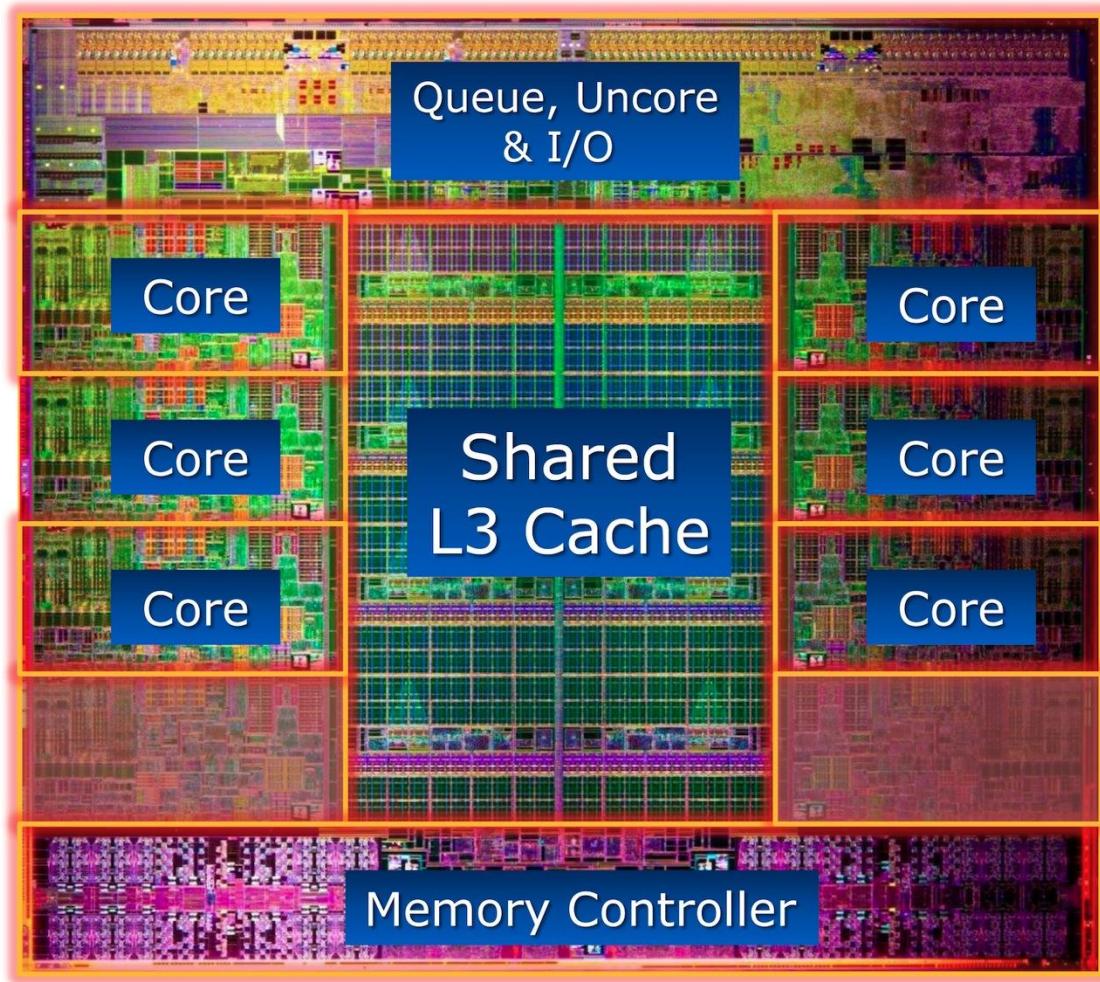
Intel Microprocessor Hall of Fame

www.intel.com/



hcs.ucla.edu/cluster1.htm

- 2270 million transistors
- 32 nm process
- 3300 to 3900 MHz
- 6 cores / 12 threads
- 435 mm²
- 2011 pins
- 130 watts



Technology development

Year	Name	Size (cu. ft.)	Power (watts)	Performance (adds/sec)	Memory (KB)	Price	Price- performance vs. UNIVAC	Adjusted price (2003 \$)	Adjusted price- performance vs. UNIVAC
1951	UNIVAC I	1,000	125,000	2,000	48	\$1,000,000	1	\$6,107,600	1
1964	IBM S/360 model 50	60	10,000	500,000	64	\$1,000,000	263	\$4,792,300	318
1965	PDP-8	8	500	330,000	4	\$16,000	10,855	\$75,390	13,135
1976	Cray-1	58	60,000	166,000,000	32,000	\$4,000,000	21,842	\$10,756,800	51,604
1981	IBM PC	1	150	240,000	256	\$3,000	42,105	\$5,461	154,673
1991	HP 9000/ model 750	2	500	50,000,000	16,384	\$7,400	3,556,188	\$9,401	16,122,356
1996	Intel PPro PC (200 MHz)	2	500	400,000,000	16,384	\$4,400	47,846,890	\$4,945	239,078,908
2003	Intel Pentium 4 PC (3.0 GHz)	2	500	6,000,000,000	262,144	\$1,600	1,875,000,000	\$1,600	11,452,000,000

Number of transistors

Processor	Year of introduction	Transistors
4004	1971	2,250
8008	1972	2,500
8080	1974	5,000
8086	1978	29,000
286	1982	120,000
386	1985	275,000
486 DX	1989	1,180,000
Pentium	1993	3,100,000
Pentium II	1997	7,500,000
Pentium III	1999	24,000,000
Pentium 4	2000	42,000,000

www.intel.com/research/silicon/mooreslaw.htm

Moore's Law

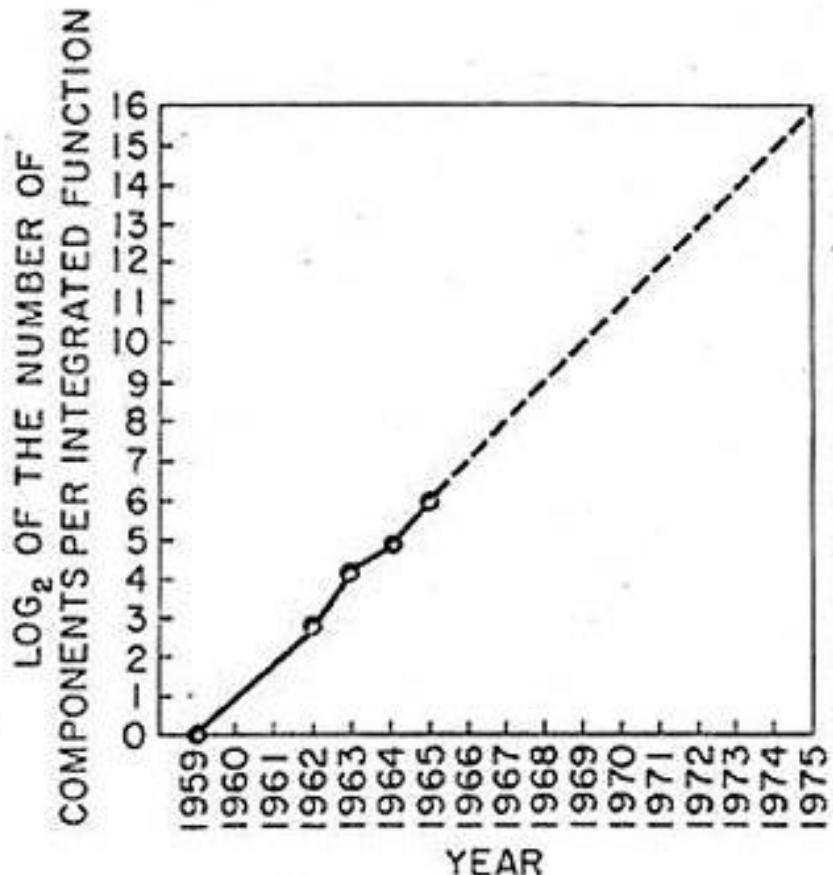


Fig. 2 Number of components per Integrated function for minimum cost per component extrapolated vs time.

Gordon E. Moore

* 03.01.1929, San Francisco

1954 Ph.D. in Chemistry & Physics
Univ. California, Berkeley

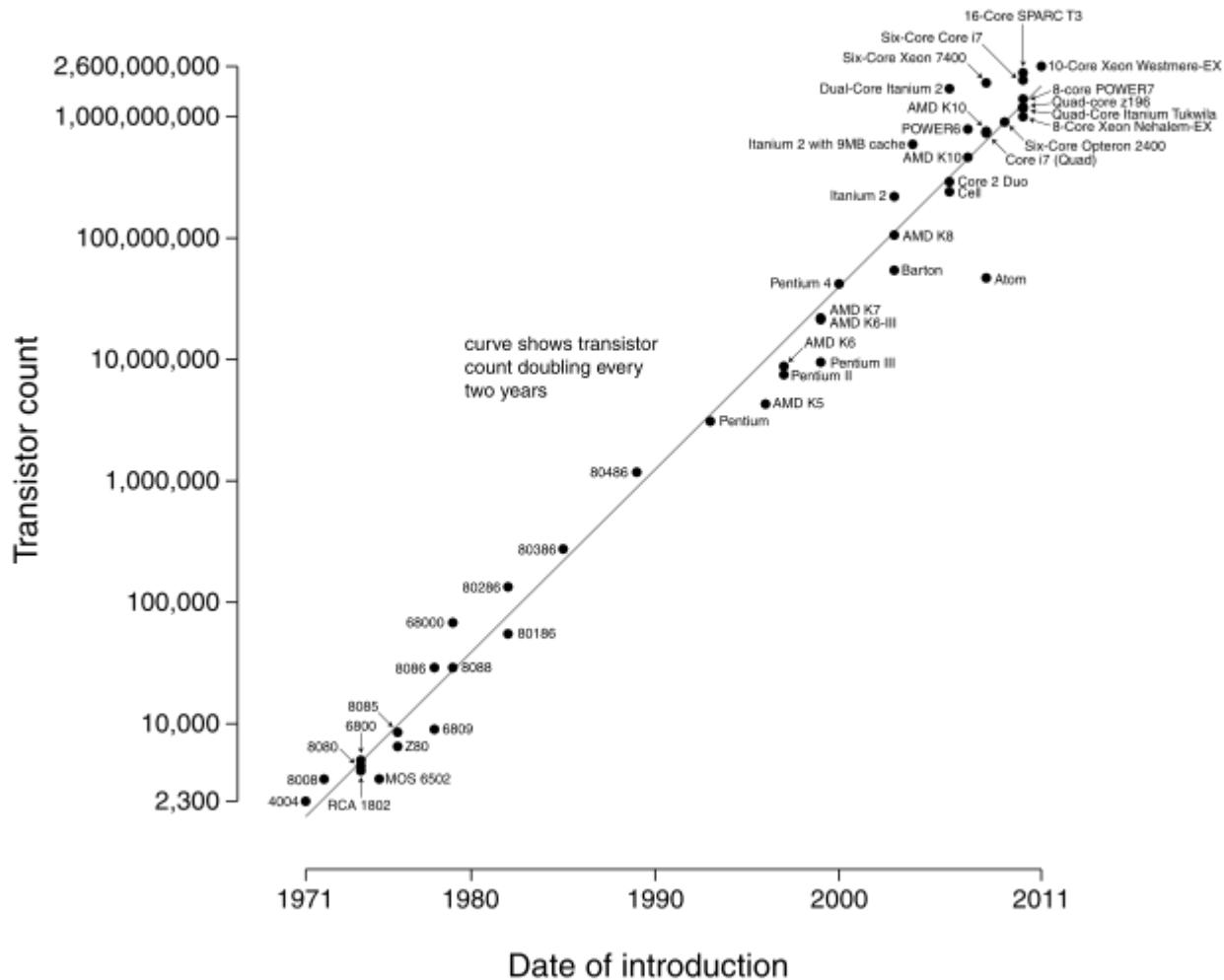
1968 founding member Intel Corp.
(Executive Vice President)

1987 retirement

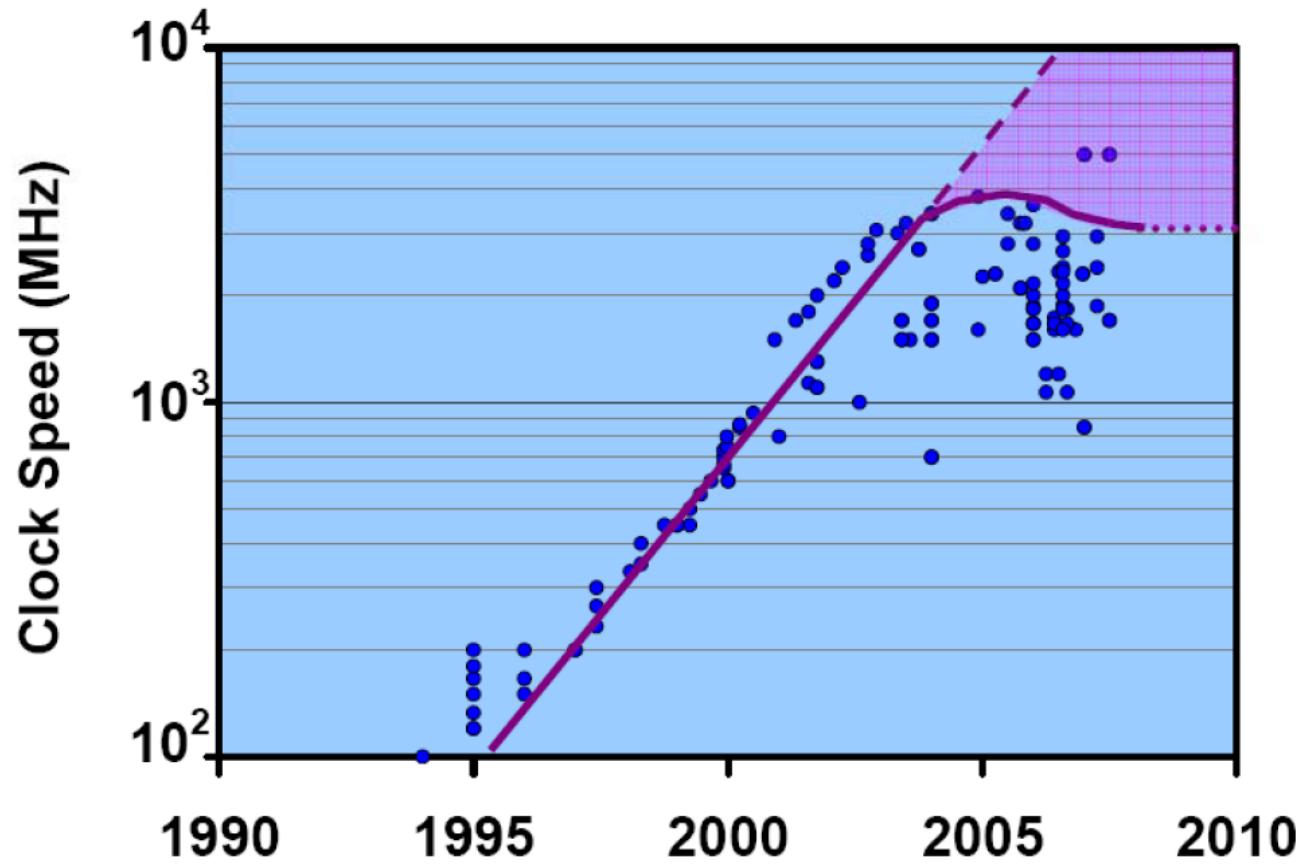
1997 Chairman Emeritus



Microprocessor Transistor Counts 1971-2011 & Moore's Law



Clock speed



Power consumption

When maintaining the computer architecture, it applies:

- computing power \sim clock rate

In CMOS technology, it holds:

- power \sim load capacity \cdot supply voltage² \cdot clock rate
 - clock rate should be increased in new generations
 - voltage is already small (5 V \rightarrow 1.5 V in the last years)
 - load capacity (number of transistors) is constant or increasing
 - Power ↑↑
 - Problem: heat dissipation

Change in strategy

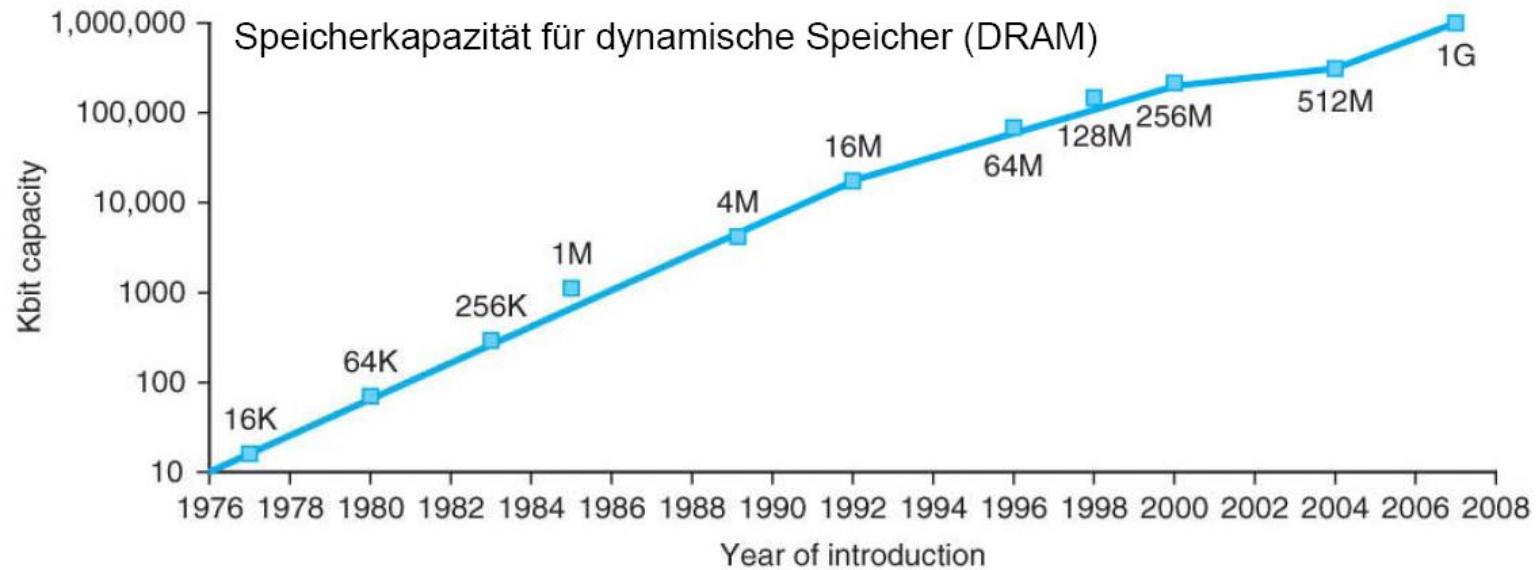
Change of the design strategy

- Lower clock speeds, but several processor cores
- According to Moore's Law, the required footprint of an IC is supposed to shrinking by half every 12 - 24 months
 - Aim: doubling the number of cores every 2 years

Product	AMD Opteron X4 (Barcelona)	Intel Nehalem	IBM Power 6	Sun Ultra SPARC T2 (Niagara 2)
Cores per chip	4	4	2	8
Clock rate	2.5 GHz	~ 2.5 GHz ?	4.7 GHz	1.4 GHz
Microprocessor power	120 W	~ 100 W ?	~ 100 W ?	94 W

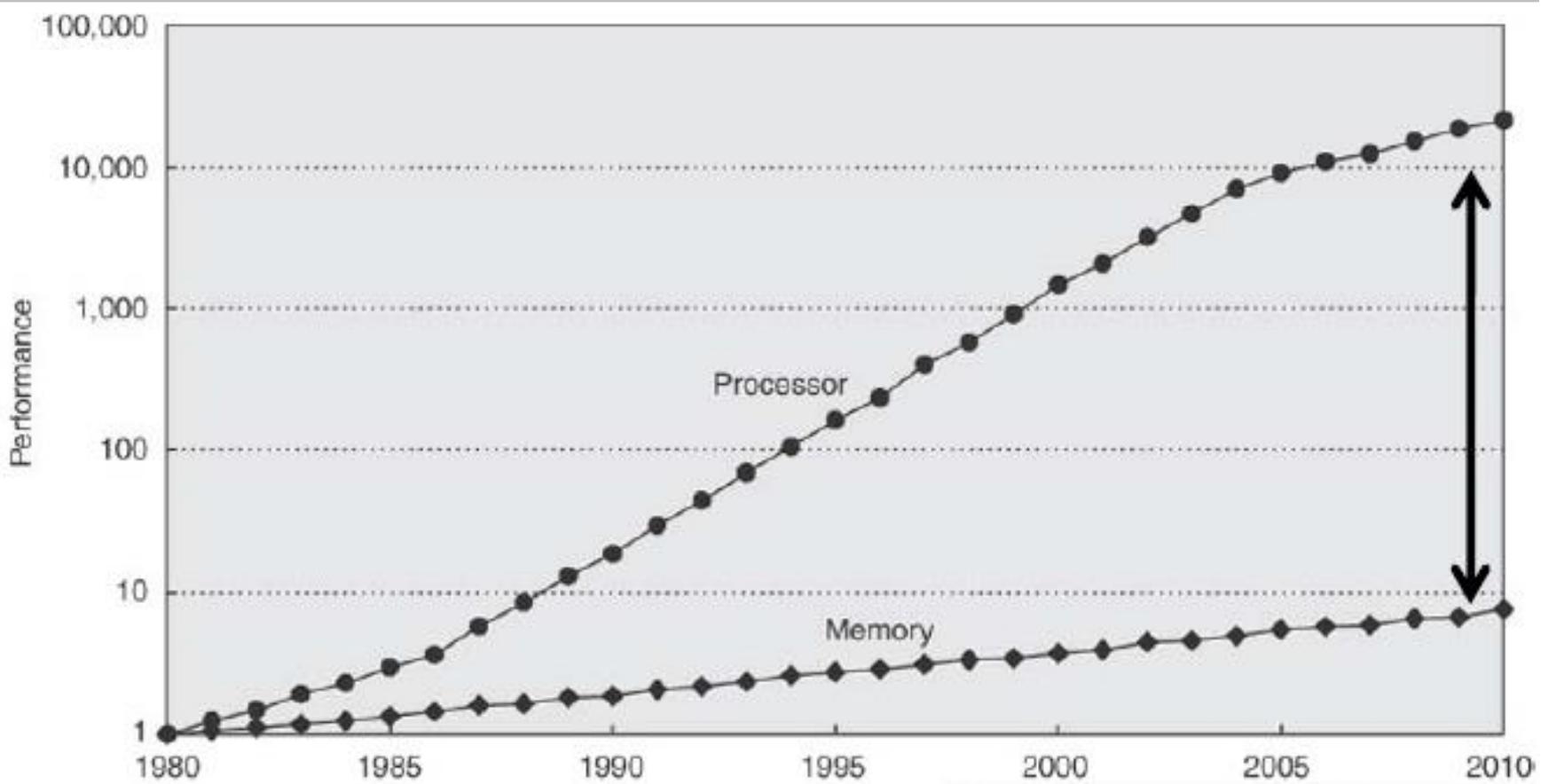
- Problems: Parallel programming, load distribution, communication between the cores, synchronization, ...

Storage capacity

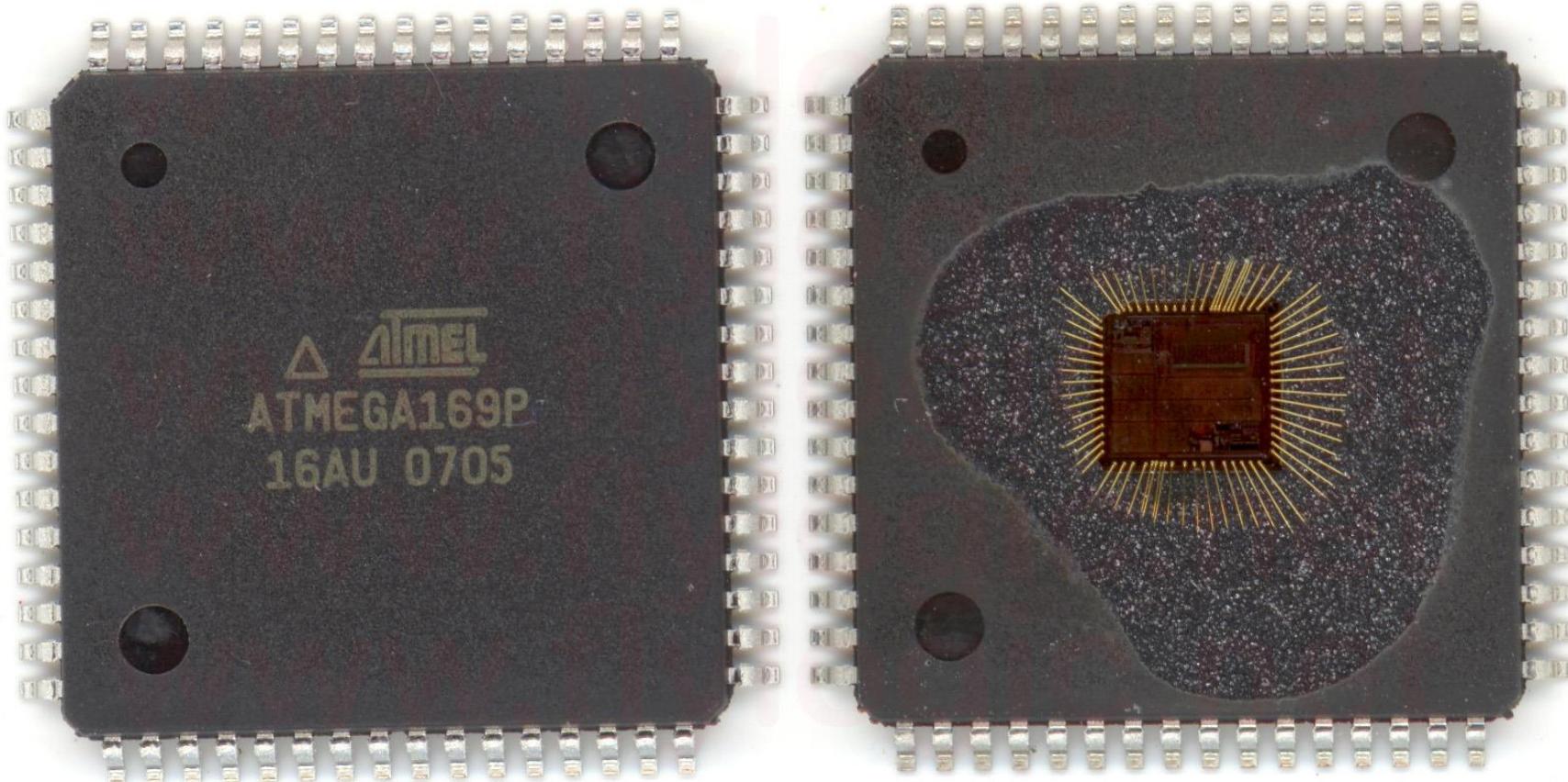


- Storage density increased by 60 % per year; only 30 % of increase in the last years
- access times decreases by approximately 30 % in 5 years

Processor-memory gap

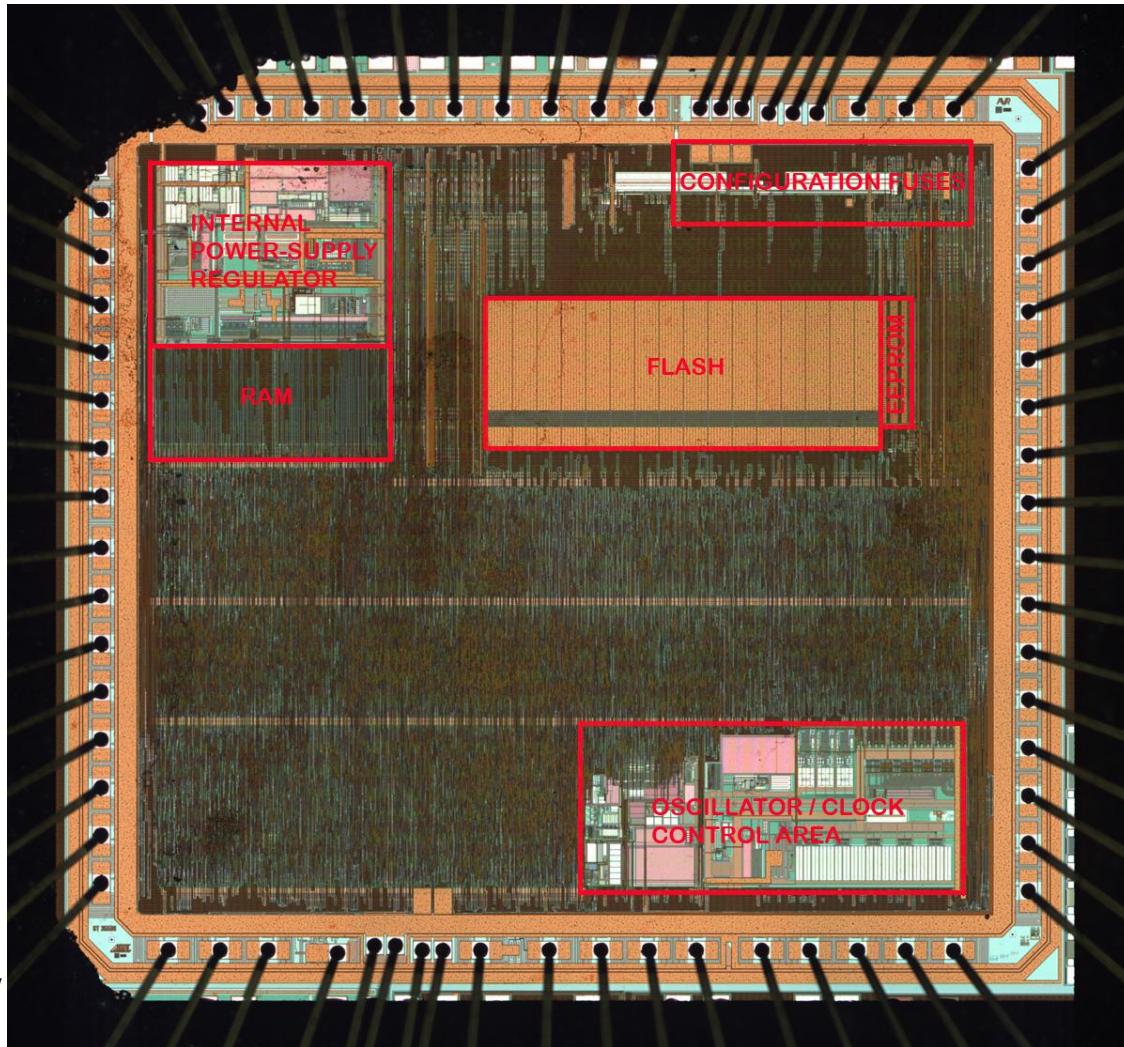


Here it is: The microcontroller!



<http://blog.ioactive.com/2007/11/atmega169p-quick-peek.html>

Here it is: The microcontroller!



<http://blog.ioactive.com/2007/11/atmega169p-quick-peek.html>

State of the lecture

✓ Introduction

- Chapter 0: basic knowledge
 - Number systems, digital logic, ...
- Chapter 1: calculation and control system
 - Basic operations in CMOS, registers, memory, concepts, ...

- Chapter 2: Basic peripherals
 - GPIO, ADC, ...

...

0. Basic knowledge

Digital signals, character and number representations, digital logic

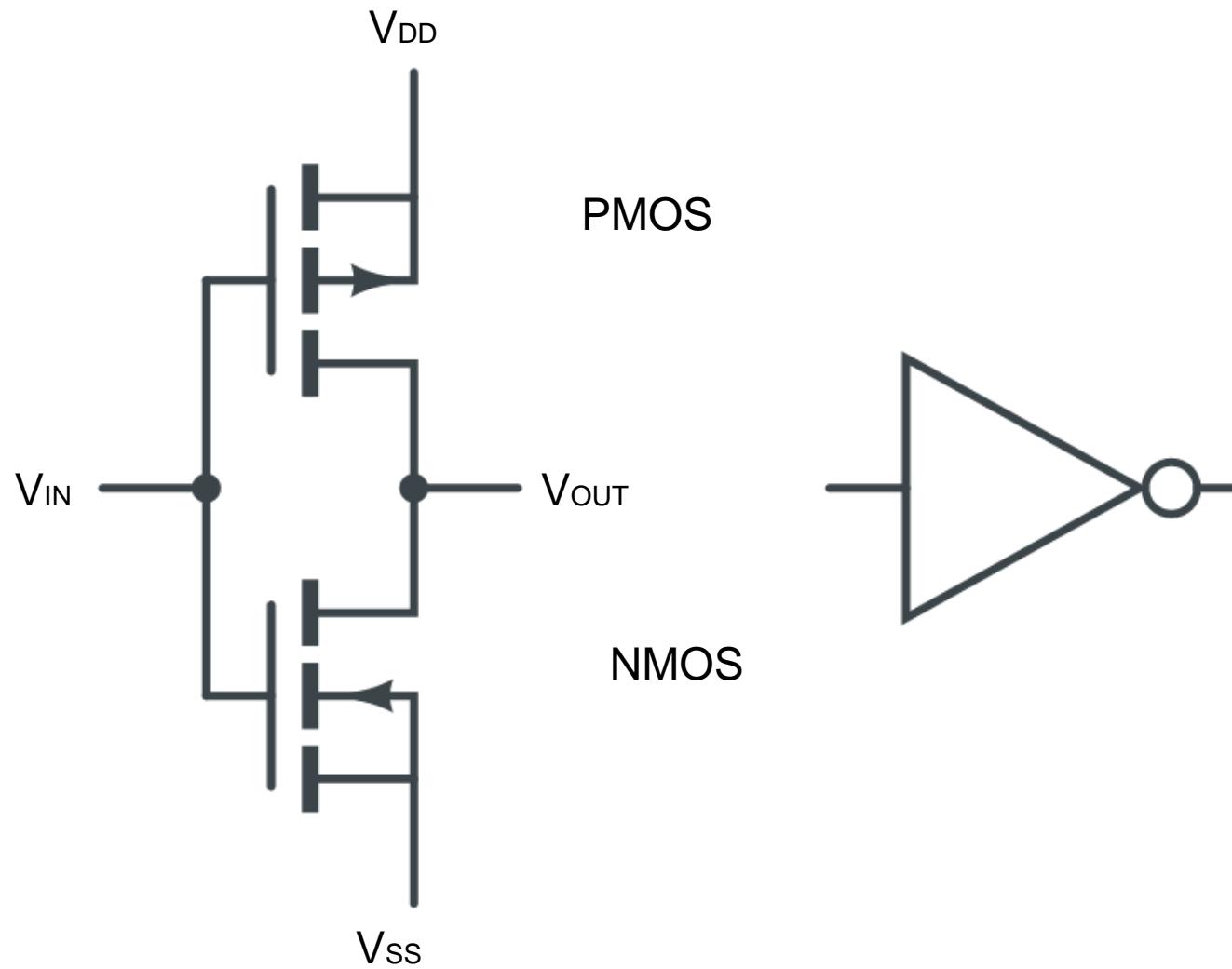
0. Basic knowledge

INTRODUCTION - DIGITAL SIGNALS

High, Low, Z, X

- High signal
 - connection to the (supply) voltage
 - 24 V, 12 V, 5 V, 3.3 V, 1.8 V, ... (generally referred as V_{DD} for CMOS)
 - logic 1
- Low signal
 - connection to ground
 - V_{SS} (for CMOS usually GND)
 - logic 0
- Z
 - high impedance
 - „floating wire“
 - neither 1 nor 0
- X
 - any signal

CMOS inverter in detail



0. Basic knowledge

CHARACTER- & NUMBER ENCODING

It's all about numbers

- Computers can't process any text
- Everything which is processed must be represented as a number
- Letters are translated to numbers
(e.g. ASCII = American Standard Code for Information Interchange)

Code	...0	...1	...2	...3	...4	...5	...6	...7	...8	...9	...A	...B	...C	...D	...E	...F
0...	<i>NUL</i>	<i>SOH</i>	<i>STX</i>	<i>ETX</i>	<i>EOT</i>	<i>ENQ</i>	<i>ACK</i>	<i>BEL</i>	<i>BS</i>	<i>HT</i>	<i>LF</i>	<i>VT</i>	<i>FF</i>	<i>CR</i>	<i>SO</i>	<i>SI</i>
1...	<i>DLE</i>	<i>DC1</i>	<i>DC2</i>	<i>DC3</i>	<i>DC4</i>	<i>NAK</i>	<i>SYN</i>	<i>ETB</i>	<i>CAN</i>	<i>EM</i>	<i>SUB</i>	<i>ESC</i>	<i>FS</i>	<i>GS</i>	<i>RS</i>	<i>US</i>
2...	<i>SP</i>	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3...	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4...	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5...	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6...	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7...	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Source: de.wikipedia.org - ASCII

Examples of character encoding

- Every character is represented by a number
- However, different (non-compatible) encoding schemes exist:
 - Hällü Wörld
(ANSI)
 - H„ll W”rld
(OEM 852)
 - HÄLLÄL' WÄ¶rld
(Windows 1250)
 - 01001000 11100100 01101100 01101100 11111100 00100000
01010111 11110110 01110010 01101100 01100100

Numeral systems

There are three most frequently encountered numeral systems in the field of computers:

- Binary
 - 0, 1, 10, 11, 100, ...
- Decimal
 - 0, 1, ..., 9, 10, 11, ...
- Hexadecimal
 - 0, 1, ..., 9, A, ... F, 10, 11, ...

Positional notation

General representation of integers

- Most number encodings are based on a ***positional notation***
- Integers are represented by:

$$A = a_{B-1}a_{B-2} \dots a_1a_0$$

with $a_i \in \{0, 1, \dots, r - 1\}$ and the position's value r^i

- The value of a number is defined by

$$A = \sum_{i=0}^{B-1} a_i r^i$$

- Usual bases are $r = 2$ (binary), $r = 10$ (decimal) and $r = 16$ (hexadecimal)

Positional notation

- 124
 - 0111 1100 (sometimes written as: 0111 1100_b or 0111 1100₂)
 - 0x7C [spoken „hex(decimal) seven C“; alternatively: 7Ch or 7C₁₆)
- 255
 - 1111 1111
 - 0xFF
- 256
 - 0001 0000 0000
 - 0x100 (spoken „hexadecimal one-zero-zero“)
 - If limited to 8 bits: 0000 0000 (0x00)

Basic arithmetic operations in binary system

Identical to basic arithmetic operations in the decimal system:

addition

$$\begin{array}{r} 0001 \\ + 0110 \\ \hline = 0111 \end{array}$$

multiplication

$$\begin{array}{r} 0101 \cdot 0011 \\ \quad \quad \quad 0000 \\ \quad \quad \quad 0000 \\ \quad \quad \quad 0101 \\ \quad \quad \quad 0101 \\ \hline = 0001111 \end{array}$$

subtraction

$$\begin{array}{r} 0111 \\ - 0110 \\ \hline = 0001 \end{array}$$

attention:

$$\begin{array}{r} 1111 \\ + 0001 \\ \hline = 10000 \end{array}$$

Identical with long numbers

$$\begin{array}{r} 010000000 \\ - 001111111 \\ \hline = 000000001 \end{array}$$

For a fixed word width of 4 bits
the result is a zero!

Negative numbers

- $5 = 0101$
 - $127 = 0111\ 111$

 - And -127?
 - computer don't know a negative sign on the bit level
- encoding is required

Number encoding: magnitude and sign

- The MSB (a_{B-1}) determines the sign:

$$A = \begin{cases} \sum_{i=0}^{B-2} a_i r^i & \text{for } a_{B-1} = 0 \\ -\sum_{i=0}^{B-2} a_i r^i & \text{for } a_{B-1} = 1 \end{cases}$$

- The range is $-(2^{B-1}-1) \dots 2^{B-1}-1$

Number encoding: magnitude and sign

- Most significant bit indicates sign, the magnitude is remaining

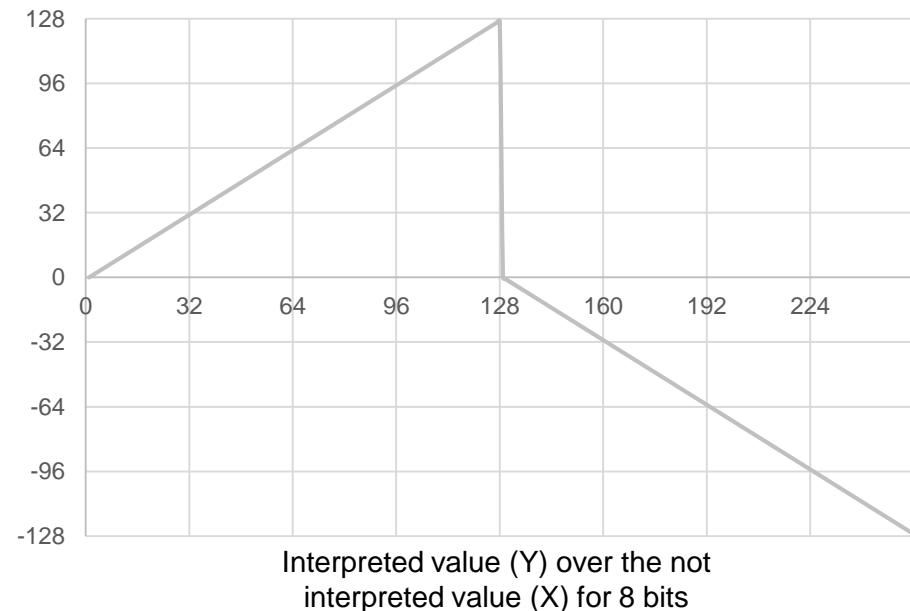
- Example

- 4 = 0000 0100
- -4 = 1000 0100
- 127 = 0111 1111
- -127 = 1111 1111

- $0000\ 0100 + 0000\ 0001 \text{ (} 4 + 1 \text{)}$
 $= 0000\ 0101 \text{ (} 5 \text{)}$

- Two issues:

- $0000\ 0100 + 1000\ 0100 = ?$
(1000 1000 would be -8, which is completely wrong ...)
- $1000\ 0000 \triangleq 0000\ 0000$



Number encoding: one's complement

- For negative numbers, the complement is formed (by inverting).
- The MSB (a_{B-1}) is determined to be the sign :

$$A = \begin{cases} \sum_{i=0}^{B-2} a_i r^i & \text{for } a_{B-1} = 0 \\ -2^{B-1} + 1 + \sum_{i=0}^{B-2} a_i r^i & \text{for } a_{B-1} = 1 \end{cases}$$

- The range is $-(2^{B-1} - 1) \dots 2^{B-1} - 1$

Number encoding: one's complement

- Most significant bit is the sign.
- If the number is negative, the magnitude is inverted
- Example

➤ 4 = 0000 0100

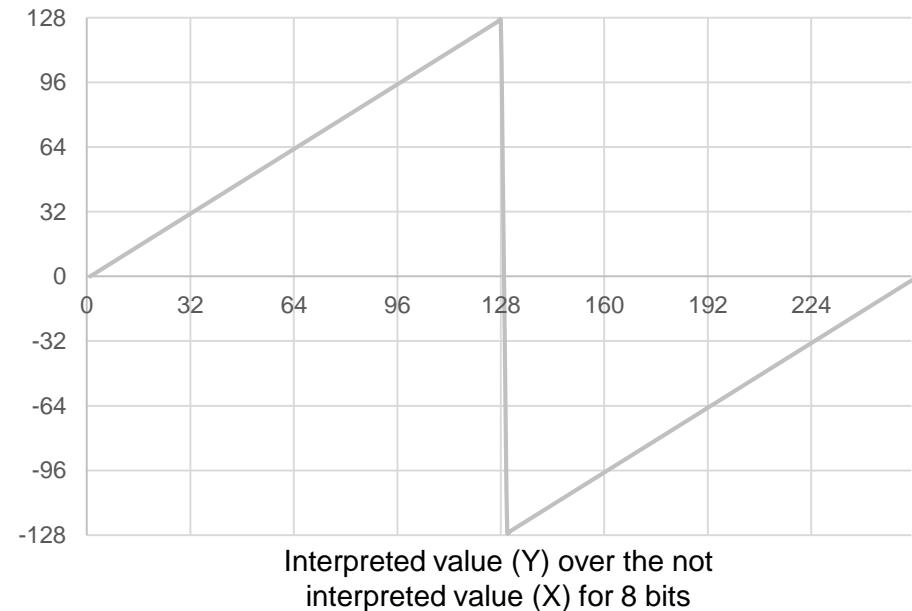
➤ -4 = 1111 1011

➤ 127 = 0111 1111

➤ -127 = 1000 0000

➤ $0000\ 0100 + 1111\ 1011$ ($4 + (-4)$)

= 1111 1111 (0)



- Again two problems
- $0000\ 0110 + 1111\ 1010$ ($6 + (-5)$) = ?
- $1111\ 1111 \triangleq 0000\ 0000$

Number encoding: two's complement

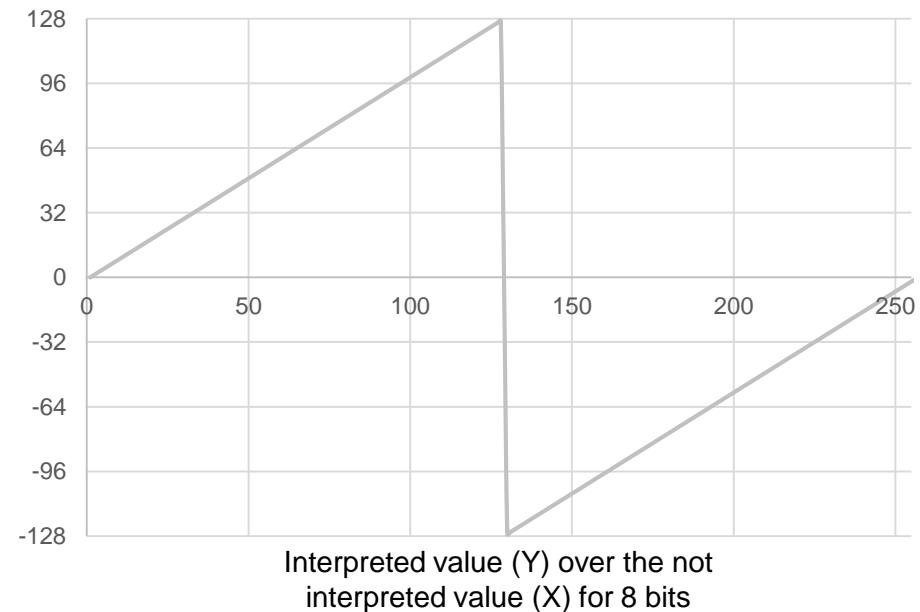
- For negative numbers, the 2's complement is formed by inverting and adding with one.
- The MSB (a_{B-1}) still determines the sign :

$$A = \begin{cases} \sum_{i=0}^{B-2} a_i r^i & \text{for } a_{B-1} = 0 \\ -2^{B-1} + \sum_{i=0}^{B-2} a_i r^i & \text{for } a_{B-1} = 1 \end{cases}$$

- The range is $-2^{B-1} \dots 2^{B-1} - 1$

Number Coding: two's complement

- Like one's complement - but negative numbers are incremented by one.
- Example
 - 4 = 0000 0100
 - -4 = 1111 1100
 - 127 = 0111 1111
 - -127 = 1000 0001
 - 0 = 0000 0000
 - -128 = 1000 0000
 - $0000\ 0110 + 1111\ 1011 (6 + (-5)) = 0000\ 0001$



Example 1: 16 + 25

$$16 = 00010000$$

$$25 = 00011001$$

One's complement

$$\begin{array}{r} 00010000 \\ + 00011001 \\ \hline \end{array}$$

$$00101001$$

Two's complement

$$\begin{array}{r} 00010000 \\ + 00011001 \\ \hline \end{array}$$

$$00101001$$

Example 2: 16 - 25

$$16 = 00010000$$

$$25 = 00011001$$

One's complement

$$-25 = 11100110$$

$$\begin{array}{r} 00010000 \\ + 11100110 \\ \hline \end{array}$$

$$\hline 11110110 = -9$$

Example 2: 16 - 25

$$16 = 00010000$$

$$25 = 00011001$$

One's complement

$$-25 = 11100110$$

$$\begin{array}{r} 00010000 \\ + 11100110 \\ \hline \end{array}$$

$$\hline \quad 11110110 = -9$$

Two's complement

$$-25 = 11100111$$

$$\begin{array}{r} 00010000 \\ + 11100111 \\ \hline \end{array}$$

$$\hline \quad 11110111 = -9$$

Binary arithmetic - Examples

Example 3: -16 + 25

$$16 = 00010000$$

$$25 = 00011001$$

One's complement

$$-16 = 11101111$$

$$\begin{array}{r} 11101111 \\ + 00011001 \\ \hline 11111111 \\ \hline 100001000 \end{array} = 8 ?$$

Binary arithmetic - Examples

Example 3: -16 + 25

$$16 = 00010000$$

$$25 = 00011001$$

One's complement

$$-16 = 11101111$$

$$\begin{array}{r} 11101111 \\ + 00011001 \\ \hline 11111111 \\ - \\ 100001000 \\ + 00000001 \\ \hline 00001001 \end{array} = 9$$

Example 3: -16 + 25

$$16 = 00010000$$

$$25 = 00011001$$

One's complement

$$-16 = 11101111$$

$$\begin{array}{r} 11101111 \\ + 00011001 \\ \hline 11111111 \\ - \\ 100001000 \\ + 00000001 \\ \hline 00001001 \end{array} = 9$$

Two's complement

$$-16 = 11110000$$

$$\begin{array}{r} 11110000 \\ + 00011001 \\ \hline 1111 \\ - \\ 100001001 \end{array} = 9$$

Binary arithmetic – More examples

M&S (magnitude & sign)	One's (one's complement)	Two's (two's complement)
0001 + 1000		
1010 + 0111		
0110 + 0111		
0110 - 0111		
1010 - 0111		
1111 - 1000		

Binary arithmetic – More examples

M&S	One's	Two's
0001 + 1000	0001 $(1 + (-0))$	1001 $(1 + (-7))$
1010 + 0111		
0110 + 0111		
0110 - 0111		
1010 - 0111		
1111 - 1000		

Binary arithmetic – More examples

M&S	One's	Two's
0001 + 1000	0001	(1 + (-0))
1010 + 0111	0101	(-2 + 7)
0110 + 0111		
0110 - 0111		
1010 - 0111		
1111 - 1000		

Binary arithmetic – More examples

M&S	One's		Two's	
0001 + 1000	0001	(1 + (-0))	1001	(1 + (-7))
1010 + 0111	0101	(-2 + 7)	0010	(-5 + 7)
0110 + 0111	01101	(6 + 7)	01101	(6 + 7)
0110 - 0111				
1010 - 0111				
1111 - 1000				

Binary arithmetic – More examples

M&S	One's		Two's	
0001 + 1000	0001	(1 + (-0))	1001	(1 + (-7))
1010 + 0111	0101	(-2 + 7)	0010	(-5 + 7)
0110 + 0111	01101	(6 + 7)	01101	(6 + 7)
0110 - 0111	1001	(6 - 7)	1110	(6 - 7)
1010 - 0111	11001	(-2 - 7)	10011	(-5 - 7)
1111 - 1000	1111	(-7 - 0)	0111	(0 - (-7))
			0111	(-1 - (-8))

0. Basic knowledge

INTRODUCTION - DIGITAL LOGIC

Digital Logic

- 0 / 1

- Four basic operations
 - NOT
 - AND
 - OR
 - XOR

- Three "advanced" operations
 - NAND
 - NOR
 - XNOR

Basic logic circuits

Combinational logic

- output signal only depends on the currently available input signals

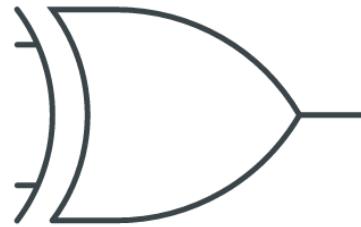
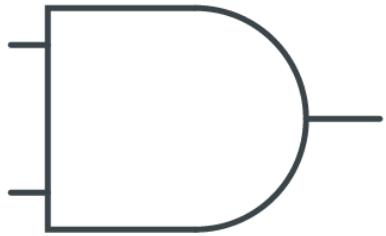
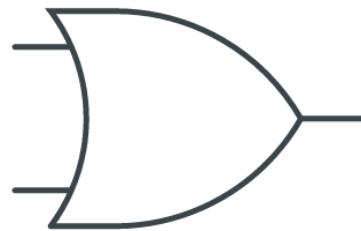
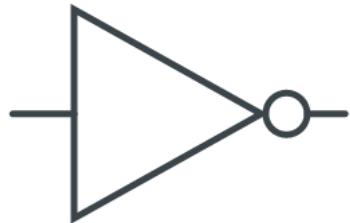
⇒ “simple” logic gates

Sequential logic

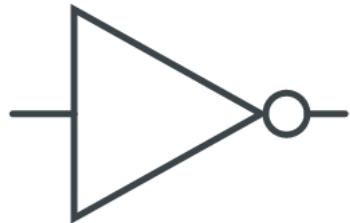
- output signal depending on currently existing input signals,
and
- from the preceding output signals

⇒ “latches” / “flip-flops”

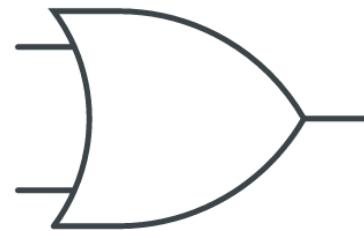
NOT, AND, OR, XOR



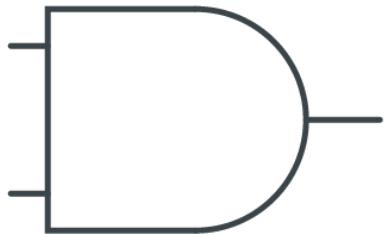
NOT, AND, OR, XOR



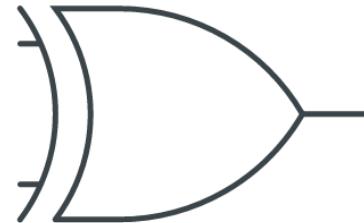
NOT



OR



AND



XOR

NOT

A	B	O
0	-	1
1	-	0
-	-	-
-	-	-

OR

A	B	O
0	0	0
0	1	1
1	0	1
1	1	1

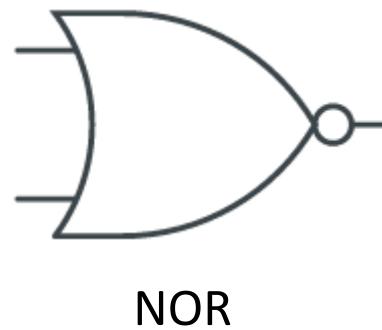
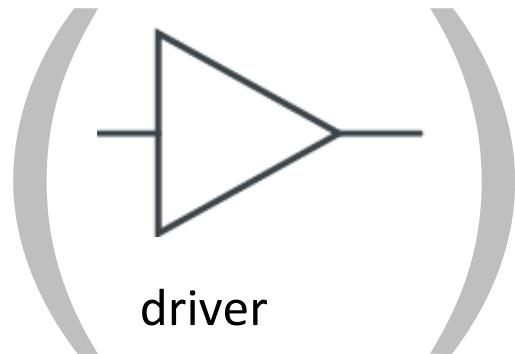
AND

A	B	O
0	0	0
0	1	0
1	0	0
1	1	1

XOR

A	B	O
0	0	0
0	1	1
1	0	1
1	1	0

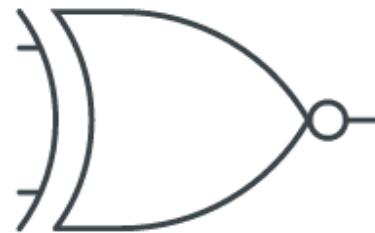
NAND, NOR, XNOR



NOR



NAND



XNOR

NAND, NOR, XNOR

(driver)

A	B	O
0	-	0
1	-	1
-	-	-
-	-	-

NOR

A	B	O
0	0	1
0	1	0
1	0	0
1	1	0

NAND

A	B	O
0	0	1
0	1	1
1	0	1
1	1	0

XNOR

A	B	O
0	0	1
0	1	0
1	0	0
1	1	1

Calculation examples

- NOT 110
- ~ 100
- 011 AND 111
- 011 & 100
- 111 OR 111
- 010 | 100
- 100 XOR 111
- 000 ^ 000
- 011 NAND 111
- $\sim (011 \& 100)$
- 111 NOR 111
- $\sim (010 | 100)$
- 100 XNOR 111
- $\sim (000 ^ 000)$

operations with 0 and 1

$$x_1 + 1 = 1$$

$$x_1 + 0 = x_1$$

$$x_1 \cdot 1 = x_1$$

$$x_1 \cdot 0 = 0$$

operations with negotiated variables

$$x_1 + \overline{x_1} = 1$$

$$x_1 \cdot \overline{x_1} = 0$$

tautology

$$x_1 + x_1 = x_1$$

$$x_1 \cdot x_1 = x_1$$

commutative property

$$x_1 + x_2 = x_2 + x_1$$

$$x_1 \cdot x_2 = x_2 \cdot x_1$$

distributive property

$$x_1 + x_2 \cdot x_3 = (x_1 + x_2) \cdot (x_1 + x_3)$$
$$x_1 \cdot (x_2 + x_3) = x_1 \cdot x_2 + x_1 \cdot x_3$$

associative property

$$x_1 + (x_2 + x_3) = (x_1 + x_2) + x_3$$
$$x_1 \cdot (x_2 \cdot x_3) = (x_1 \cdot x_2) \cdot x_3$$

De Morgan's law

$$\overline{x_1 + x_2} = \overline{x_1} \cdot \overline{x_2}$$

$$x_1 + x_2 = \overline{\overline{x_1} \cdot \overline{x_2}}$$

$$\overline{x_1 \cdot x_2} = \overline{x_1} + \overline{x_2}$$

$$x_1 \cdot x_2 = \overline{\overline{x_1} + \overline{x_2}}$$

absorption law

$$x_1 + x_1 \cdot x_2 = x_1$$

$$x_1 + \overline{x_1} \cdot x_2 = x_1 + x_2$$

$$x_1 \cdot x_2 + x_1 \cdot \overline{x_2} = x_1$$

$$x_1 \cdot (x_1 + x_2) = x_1$$

$$x_1 \cdot (\overline{x_1} + x_2) = x_1 \cdot x_2$$

$$(x_1 + x_2) \cdot (x_1 + \overline{x_2}) = x_1$$

NAND gates

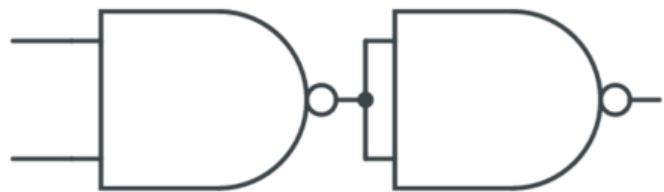


A	B	o
0	0	1
0	1	1
1	0	1
1	1	0

- all gates may be reduced to this form
- NOT



- AND

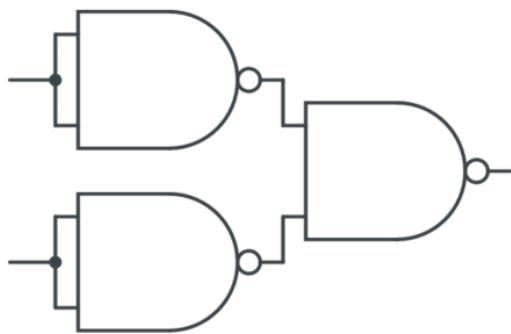


NAND gates

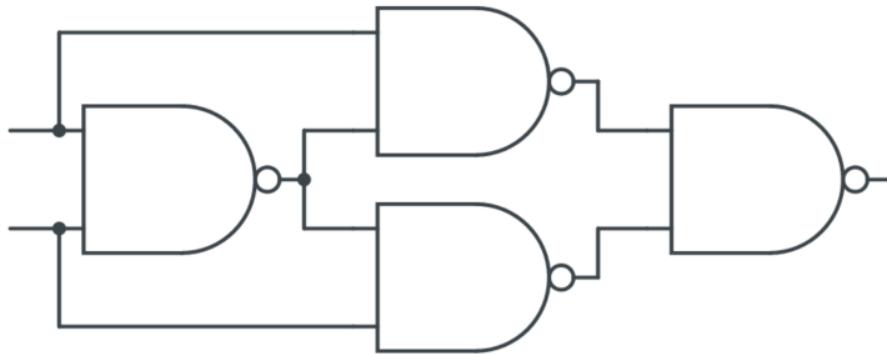


A	B	O
0	0	1
0	1	1
1	0	1
1	1	0

- OR



- XOR

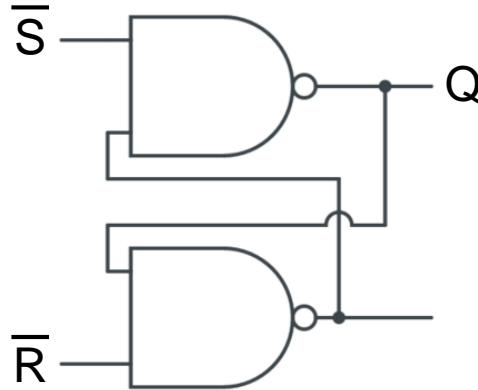


0. Basic knowledge

BASIC DIGITAL CIRCUITS

RS latch

- Wanted: hardware element to store state information

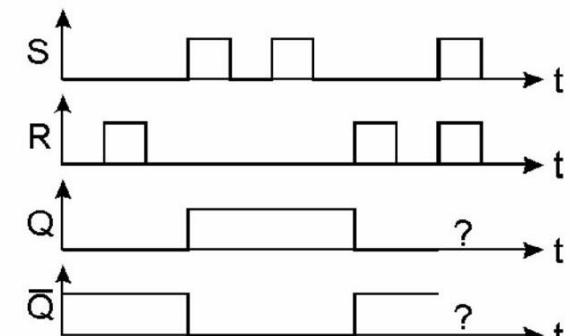


$/S$	$/R$	Q
1	1	No change
1	0	0
0	1	1
0	0	⚡

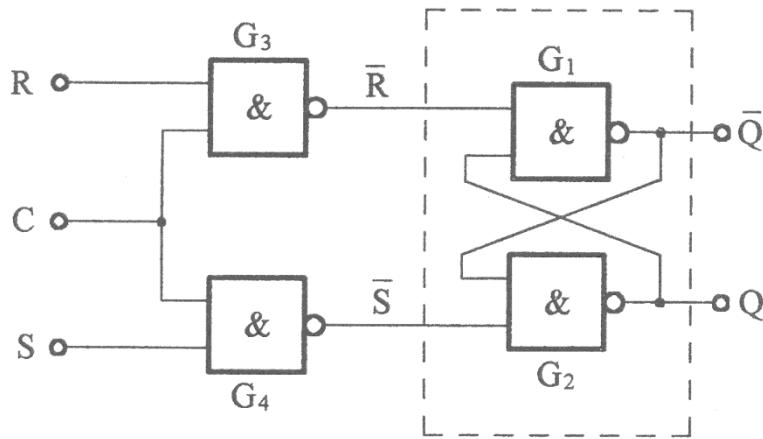
NAND:

A	B	O
0	0	1
0	1	1
1	0	1
1	1	0

- The RS memory element has the set input S, the reset input R and the two outputs Q and $/Q$.
- The latch is either set ($Q = 1, /Q = 0$) or reset ($Q = 0, /Q = 1$).
- It can be implemented as shown in the figure above by two NAND gates, where the output of a gate is applied to the input of the other. This feedback is responsible for the memory effect.



Clocked/ Gated RS latch

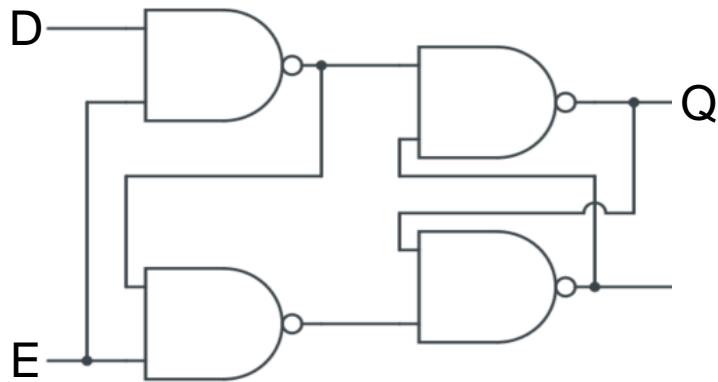


NAND:

A	B	O
0	0	1
0	1	1
1	0	1
1	1	0

- additional input for a clock signal to any kind of RS latch
- The signals S and R can only trigger a change if the clock signal C is '1'

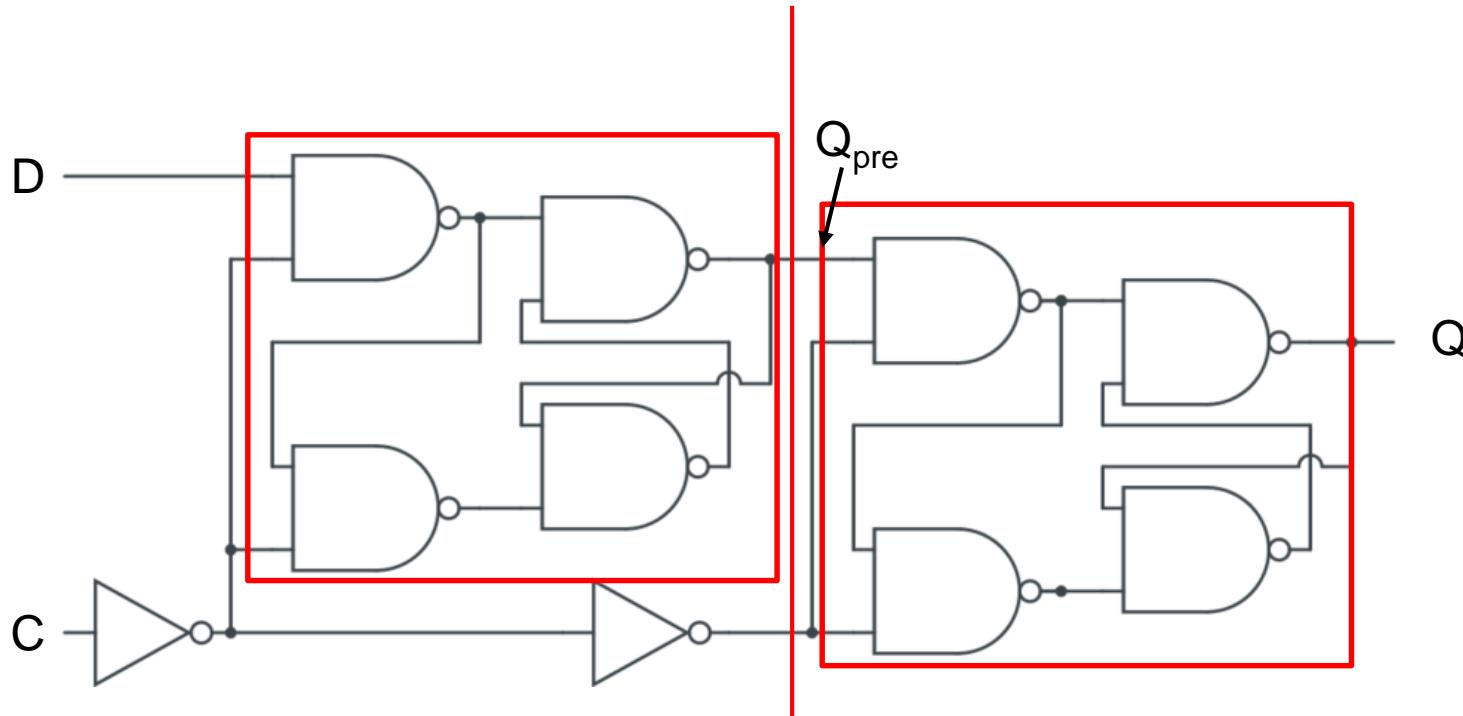
Gated D Latch



D	E	Q
X	0	no change
0	1	0
1	1	1

- Prevents the forbidded state $/S = 0$ and $/R = 0$ to occur simultaneously by the inversion $S = /R$.
- D must be actively selected by the clock E, i.e. the input D is transferred to the output during a HIGH phase of the clock.
- Disadvantage: clock-state-controlled (transparent during the whole phase of E being '1')

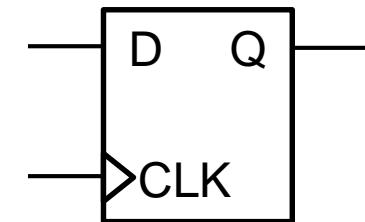
Clock-edge-controlled D flip-flop



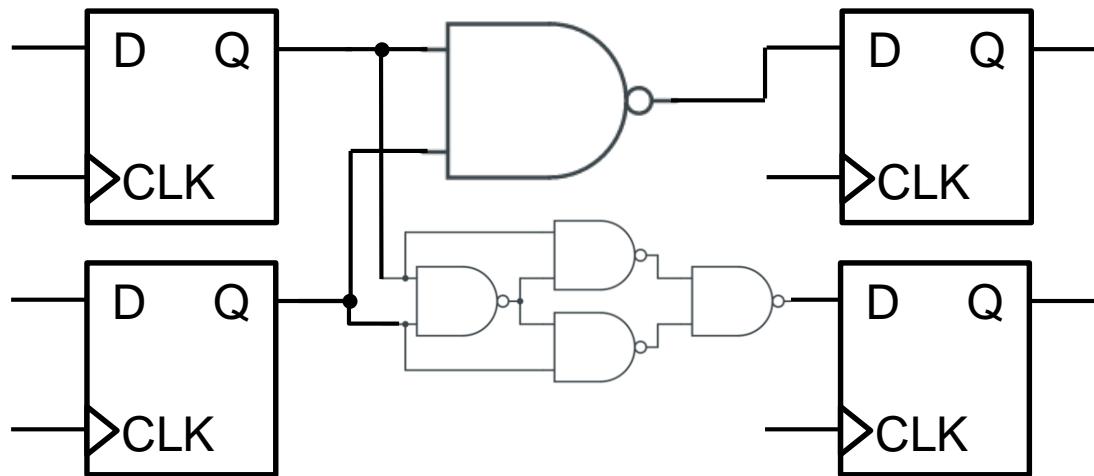
- Output is just captured at the edge of the clock signal; in case a latch is clock-edge-controlled, we call it ***flip-flop***.

D flip-flop

- always edge-controlled for the rest of the lecture

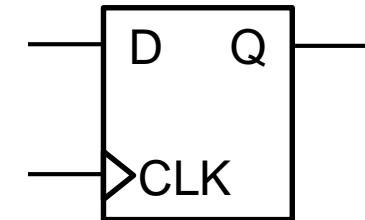


- exemplaric use case:

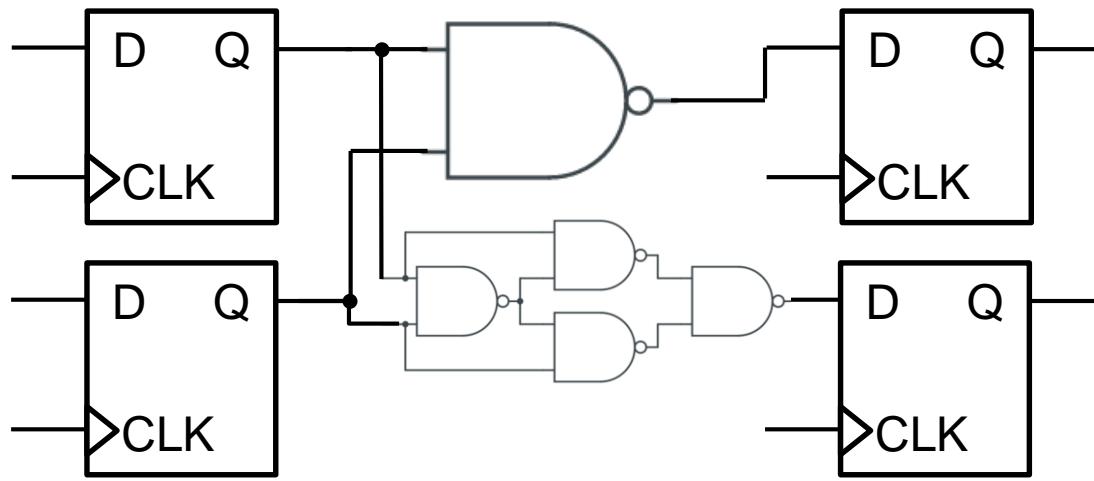


D flip-flop

- always edge-controlled for the rest of the lecture



- exemplaric use case:



NAND		
A	B	O
0	0	1
0	1	1
1	0	1
1	1	0

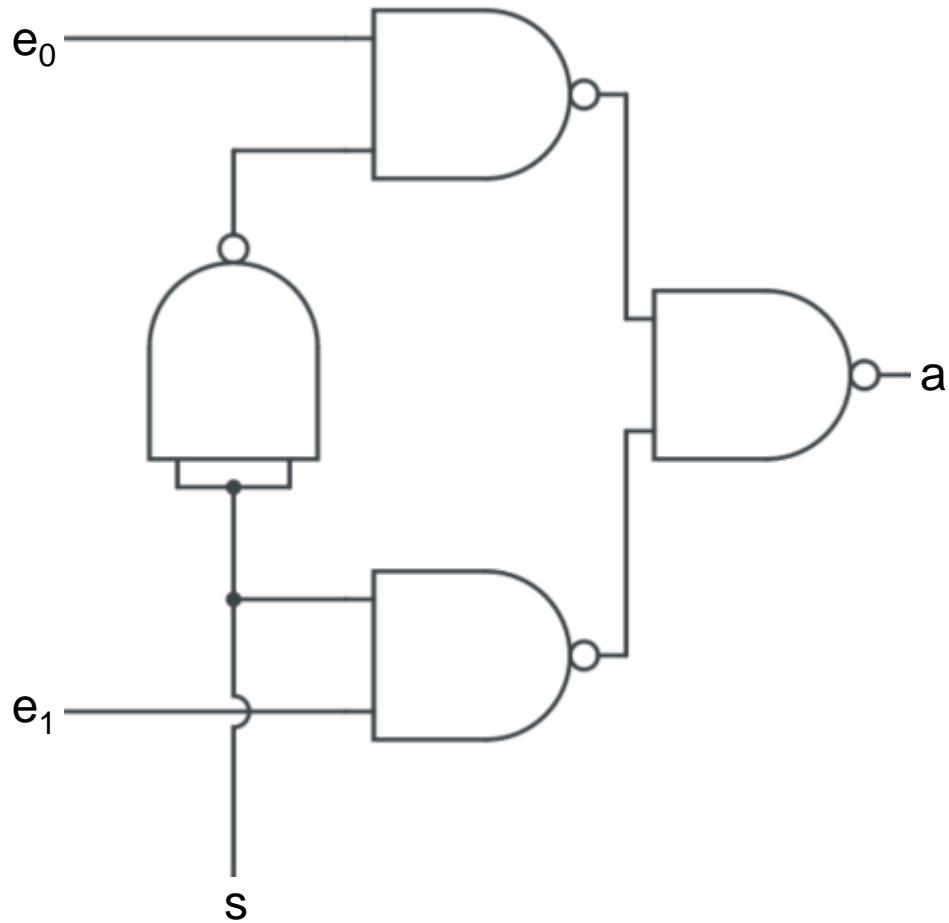
XOR		
A	B	O
0	0	0
0	1	1
1	0	1
1	1	0

Multiplexer (MUX)

- Digital switch
- Three inputs: e_0 , e_1 and s
- One output: a
- Hint: Four gates

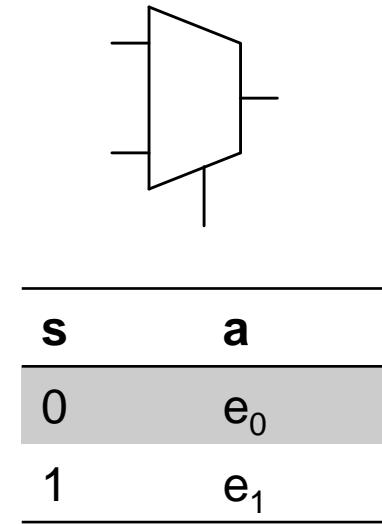
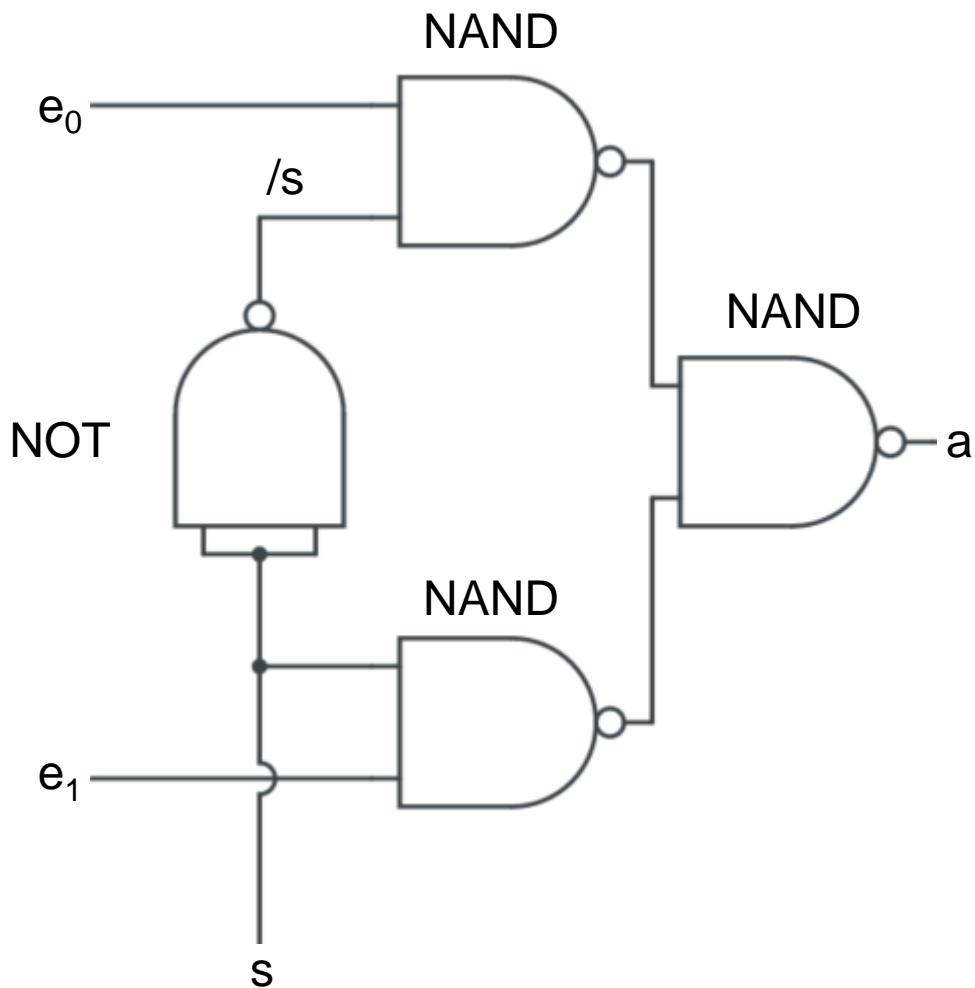
s	a
0	e_0
1	e_1

Multiplexer (MUX)



s	a
0	e_0
1	e_1

Multiplexer (MUX)



NAND	A	B	O
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

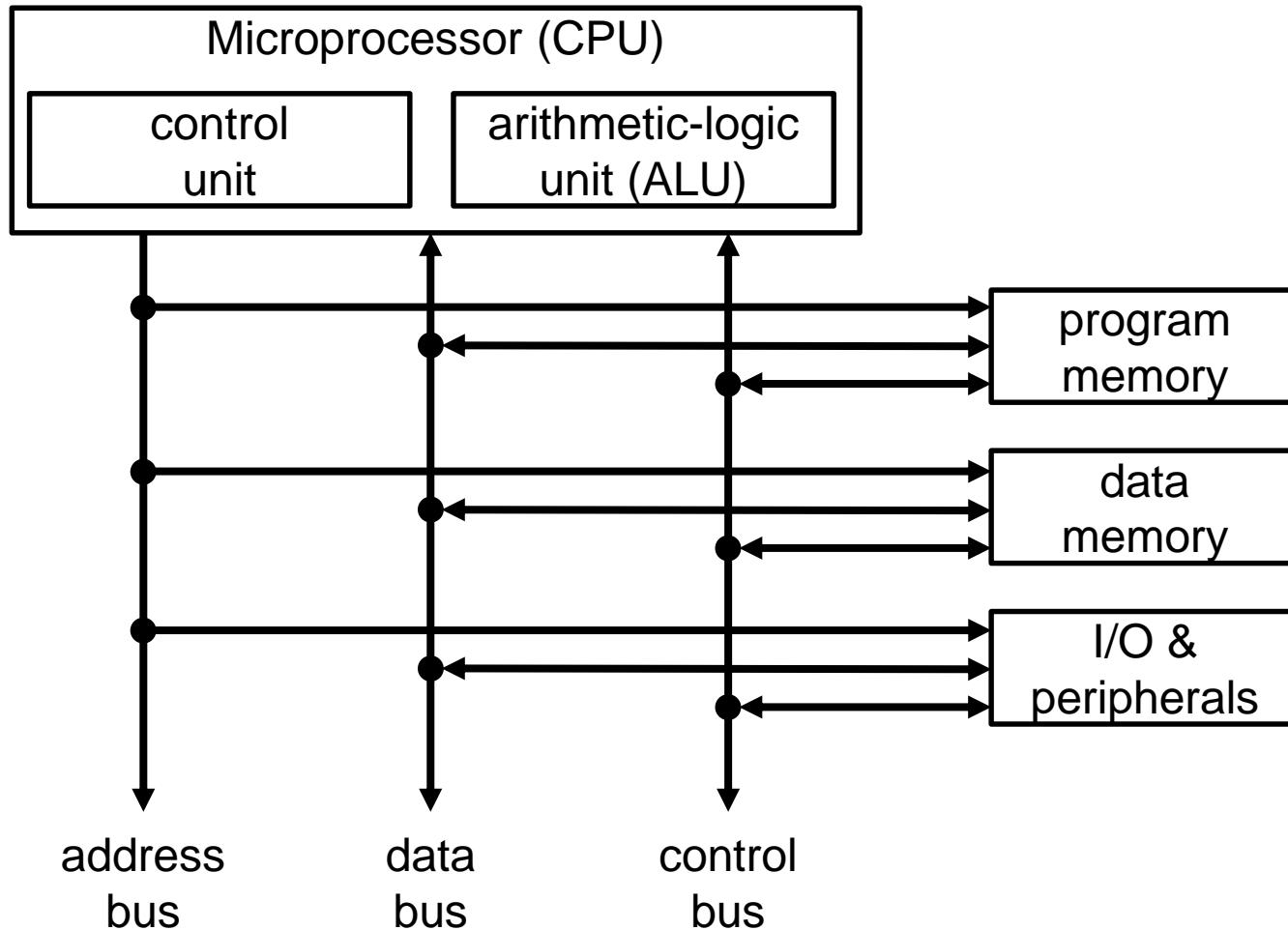
State of the Lecture

- ✓ Introduction
- ✓ Chapter 0: basic knowledge
 - ✓ Number systems, digital logic, ...
- Chapter 1: calculation and control system
 - Basic operations in CMOS, registers, memory, concepts, ...
- Chapter 2: Basic peripherals
 - GPIO, ADC, ...
- Chapter 3: Advanced Peripherals
 - Watchdog, programming interfaces, ...

1. Calculation and control system

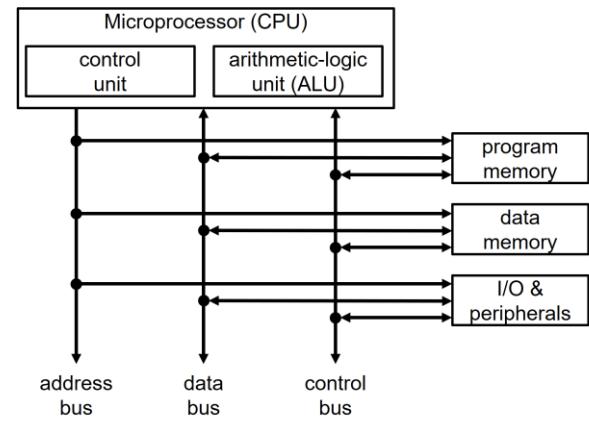
Basic operations, combination & control, registers, memory

Block diagram and components of a microcomputer



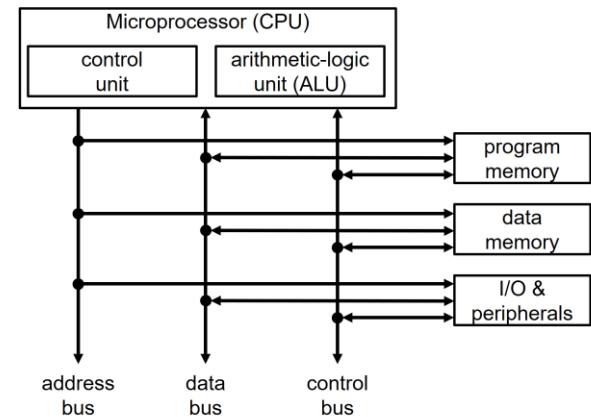
General structure of a microcomputer

- **Microprocessor:** It is the most important component and consists of control unit and arithmetic-logic unit. The microprocessor (MP) is therefore often called the Central Processing Unit (= CPU).
- **Arithmetic-logic unit (ALU):** Arithmetic (addition, subtraction, comparison) and logical operations (AND, OR, NOT, EXOR ...) are carried out inside of this unit. It might also include **registers** for storing (intermediate) results. **Flags** indicating additional information on the operation are computed. To speed up calculation, special functional units can be added, e.g., units for processing floating-point operands.



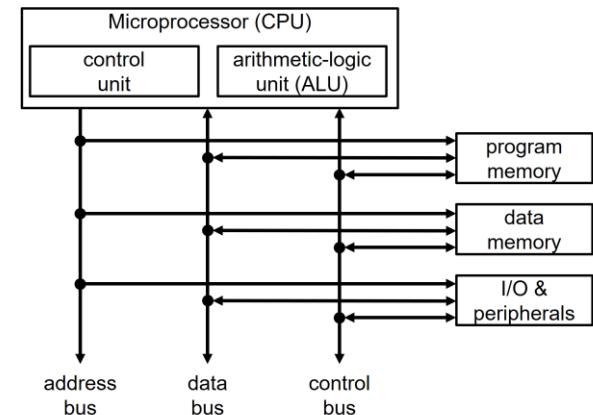
General structure of a microcomputer

- **Control Unit:** It organizes all the work of both the CPU as well as of the entire computer system.
Tasks of the control unit include:
 - Control of internal data flow, e.g., internally within the ALU, from and to the registers
 - Interpretation of the results of the ALU
 - Providing address information for reading and writing operations using the busses
 - control with external components
 - processing of interrupts and DMA
- An external clock generator (oscillator) is generating the basic clock. In modern processors, the oscillator frequencies go up to the GHz range.



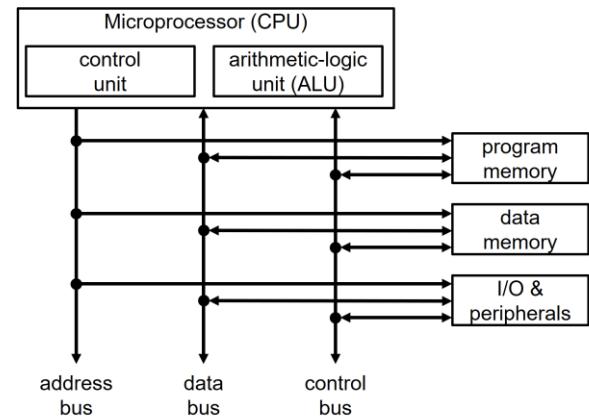
General structure of a microcomputer

- **Program memory:** Here, the program is available as a series of coded commands.
To run the program, the microprocessor must read the instructions one after another.
The command which is read from the program memory passes, for example, through the data bus to the microprocessor.
The selection of the subsequent instruction (addressing) is carried out by the microprocessor through the unidirectional address bus.
- **Data memory:** The data memory contains all variables. It can hold intermediate values for further processing in the microprocessor, or final results for the later output via the input/output unit.
The implementation is done by a read/write memory -> RAM -> Random Access Memory.
The data memory is addressed in the same way as the program memory.



General structure of a microcomputer

- **Buses:** Communication between the central processing unit and the other modules is realized by three components:
 - The **address bus**, which carries the desired memory address the CPU wants to write to and to read from. The address bus is generally unidirectional (CPU -> memory).
16 bit -> 2^{16} addresses, 32 bit -> 2^{32} addresses.
 - The **data bus** is characterized by the amount of data that can be transported in one clock cycle. There are the following gradations: 4/8/16/32/64 bit. Usually, it is a bi-directional interface.
 - The **Control Bus** transfers important command and control signals, for example, RD, WR, clock, interrupt.
 - **Input-/output- units or -circuits :** Allow for data input from a peripheral devices, e.g. keyboard, sensors. In addition, they allow for the data output to devices such as screens, LEDs, actuators, other computers.



Goal: development of minimalist CPU

- Central part: ALU
- Register / "small memory" within the CPU
- Connections to the storage
- Connections to peripherals

1. Calculation and control system

IMPLEMENTATION OF THE BASIC OPERATIONS

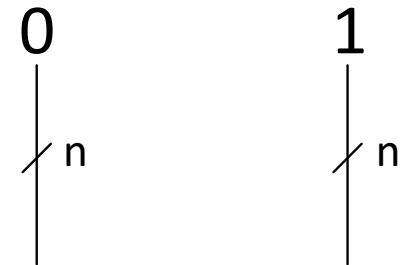
basic operations

- 0...0
- 1...1
- A AND B
- A OR B
- A XOR B

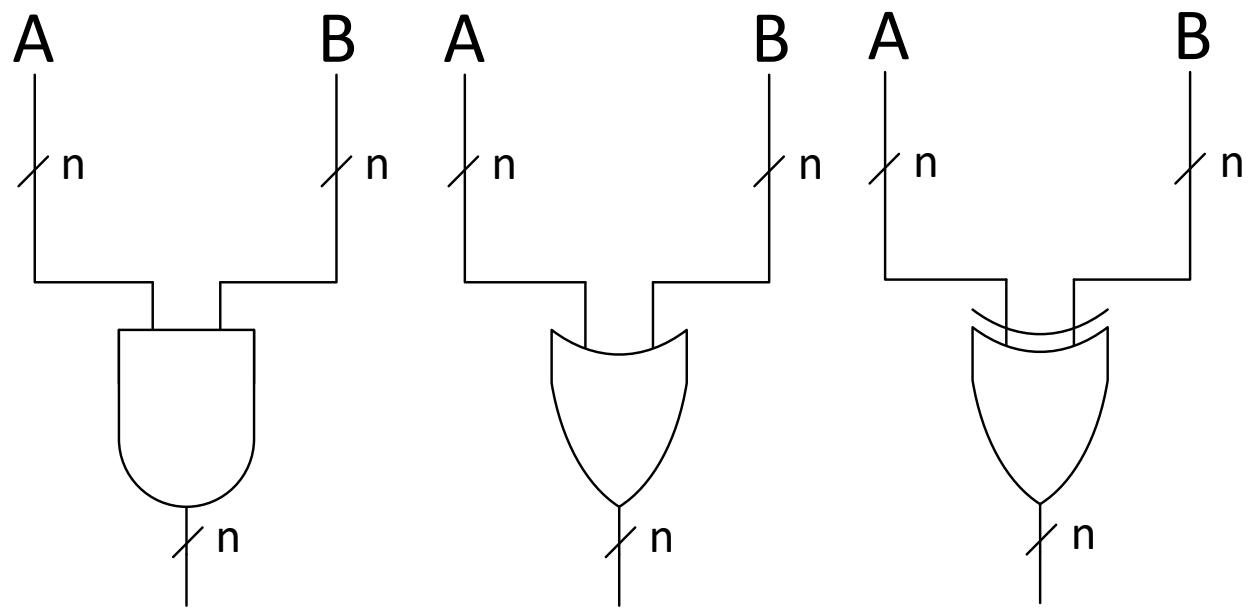
- A + B
- B - A
- A - B

0...0 and 1...1

- n lines either lead to GND (0...0)
- or even to Vcc (1...1)



- n AND side by side
- a_i and b_i will be performed on the gate AND_i
- OR & XOR: identical

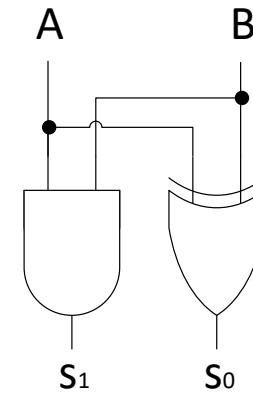


- Wanted: circuit for adding two bits
- Truth table:

a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

half adder

- Adds two bits
- Problem: $n > 1$?

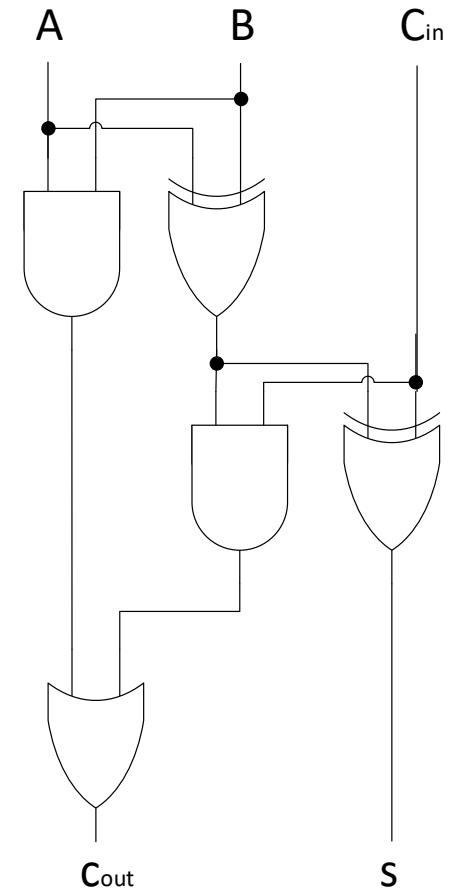


- Wanted: circuit for adding ~~two~~ three bits
- Truth table:

a	b	c _{in}	c _{out}	s
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

Full Adder

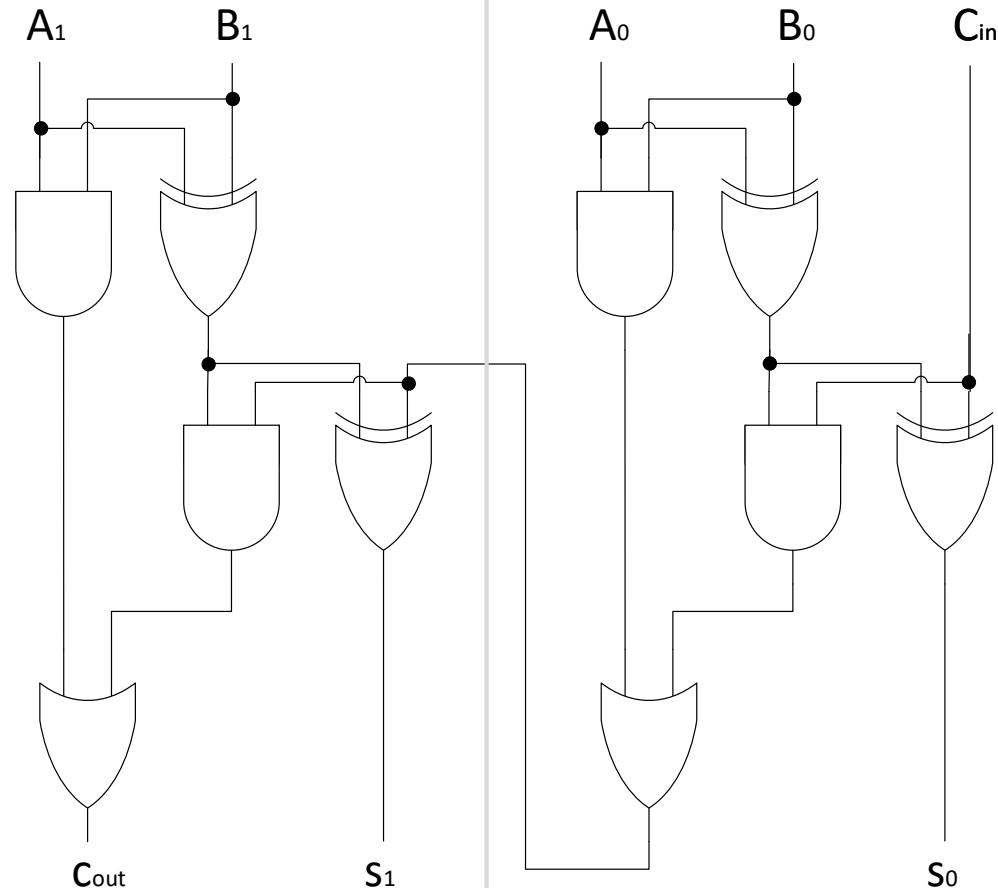
- Adds two bits (A, B) and an additional carry bit („Carry“) C_{in}
- Two Bit Output: sum and carry (for the next place)
- Result: cascadable



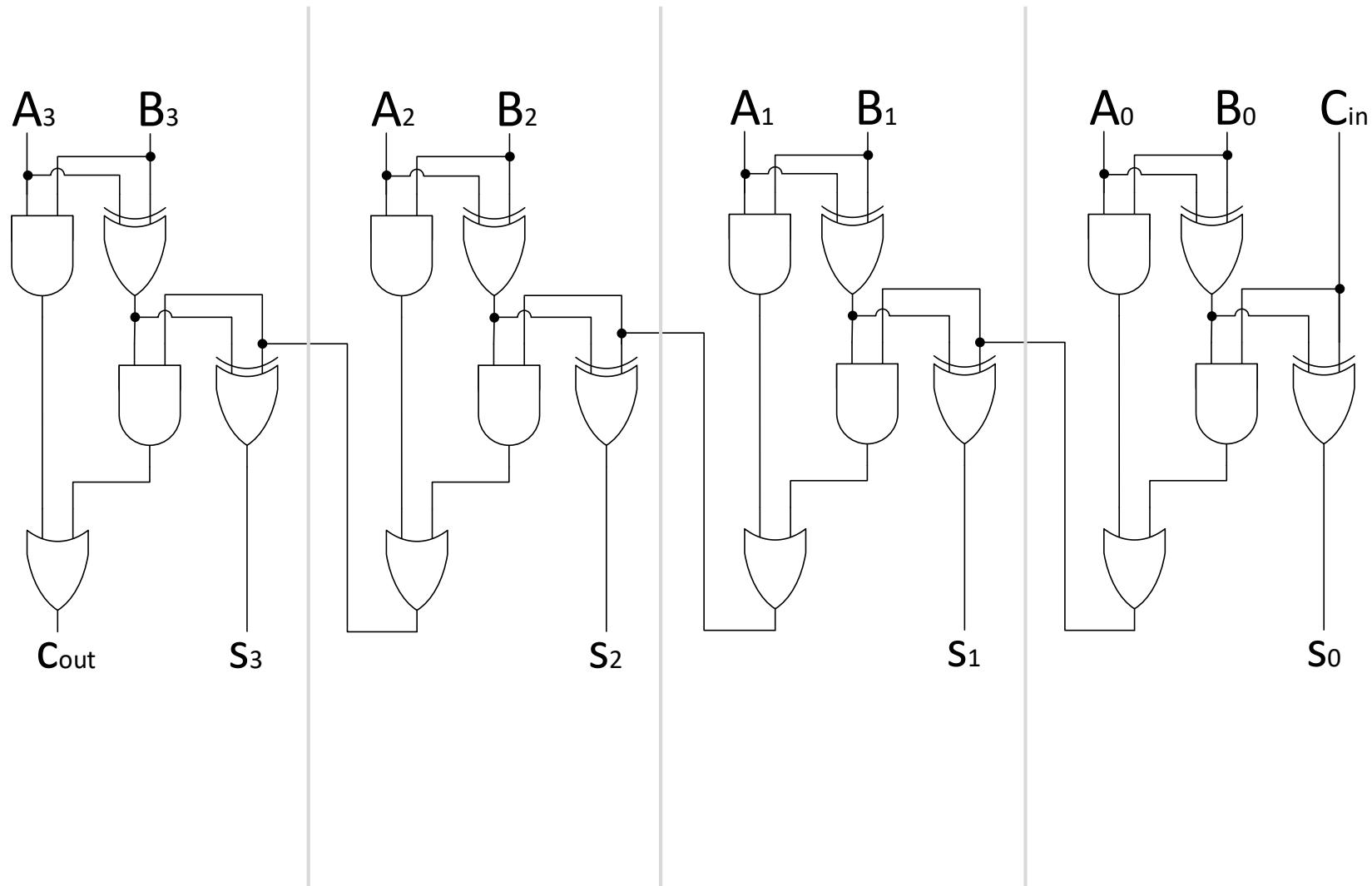
Carry Ripple Adder

- Numbers with several places possible through cascading
- Carry is constantly passed

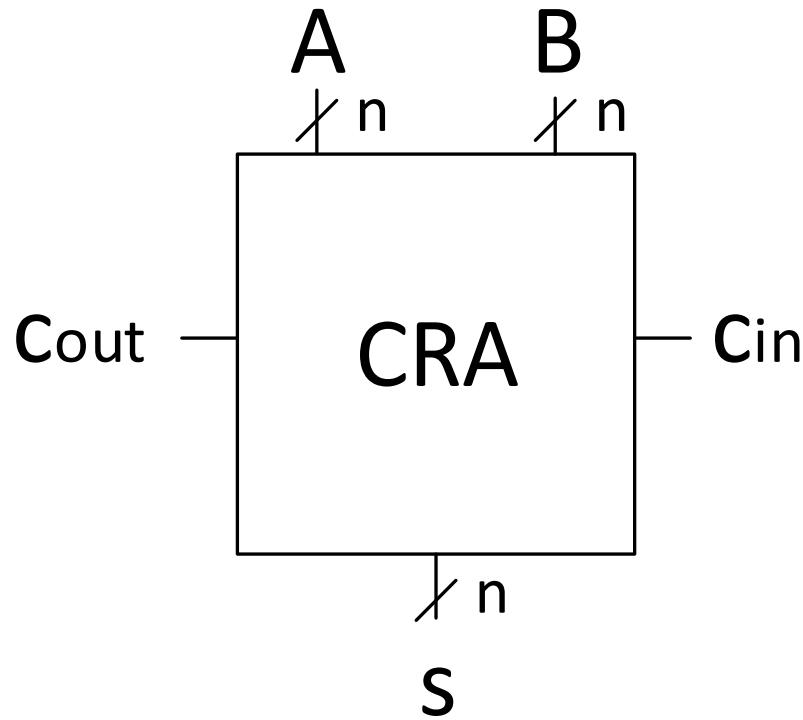
- Disadvantage: Linear Depth
- Better approaches:
 - Conditional Sum Adder
 - Carry Look Ahead Adder
 (see technical computer science and computer architecture)



Carry Ripple Adder



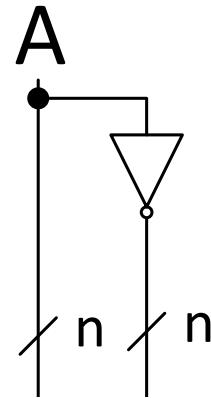
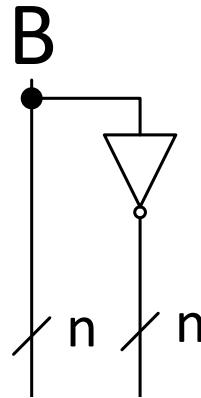
Carry Ripple Adder



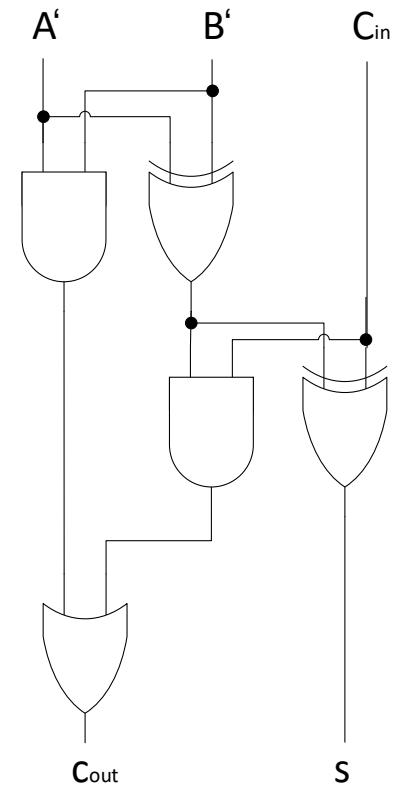
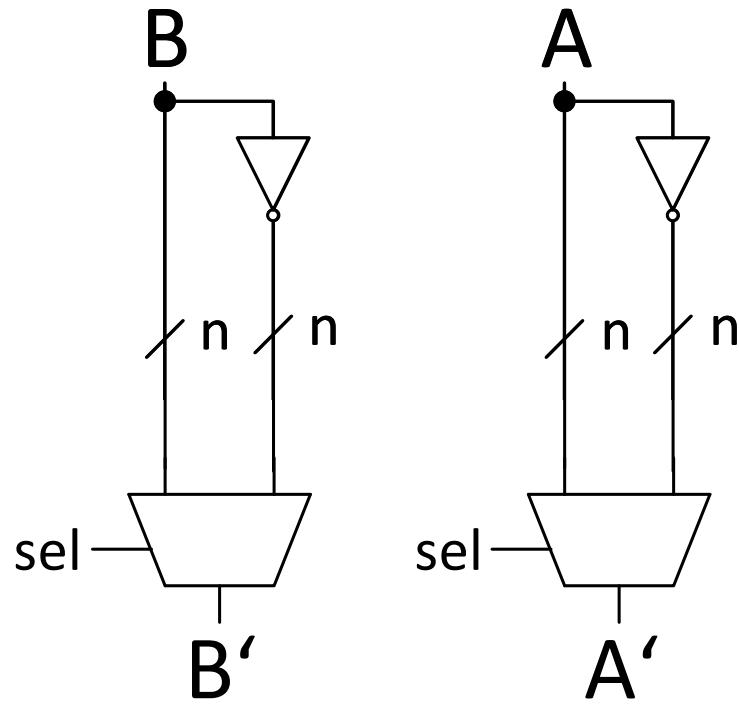
- Wanted: circuit for subtracting two numbers
- Reminder: two's complement
 - $-a = -a + 1$
- $a - b = s$
- $a + \neg b + 1 = s$
- Addition as known
 - $+ 1: C_{in}$
- Negation.
- $b - a$ analogously, because
 $b - a = -a + b = \neg a + b + 1$

SUB (Negation)

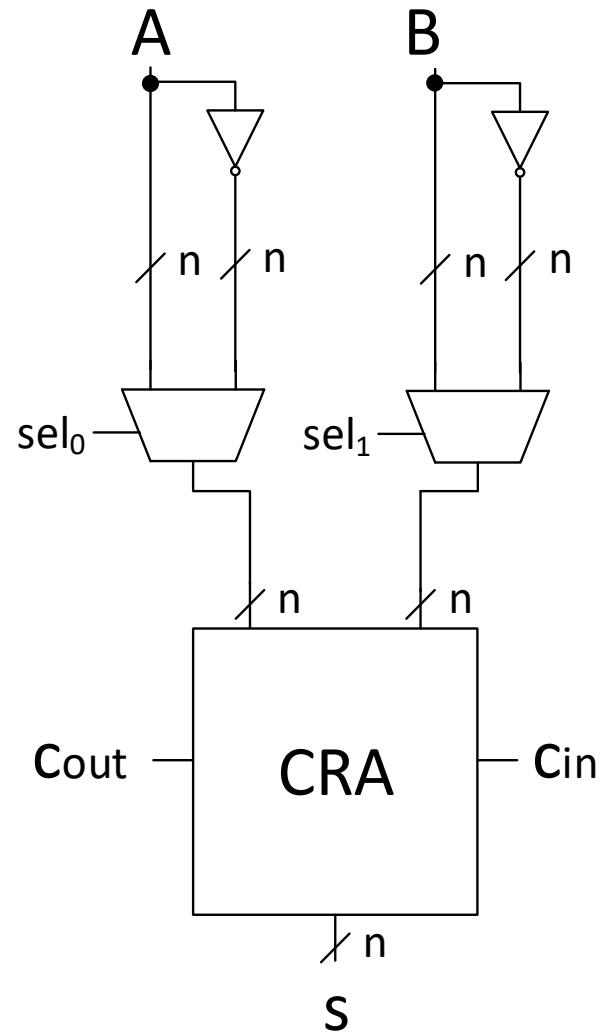
- Two possibilities
 - Additional compute cycles (and therein XOR 1)
 - Dedicated hardware



SUB vs. ADD



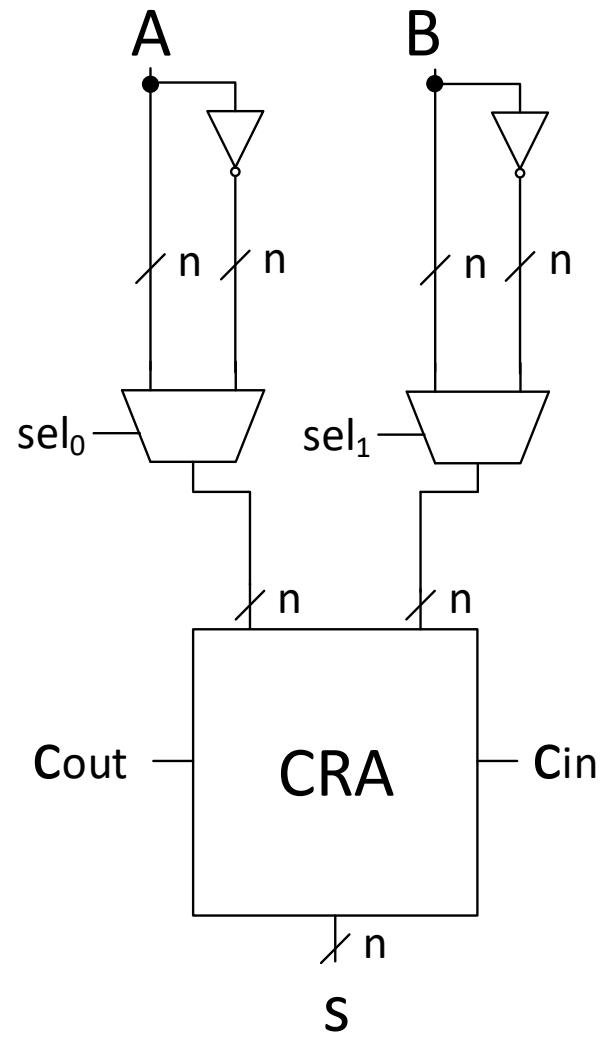
SUB vs. ADD



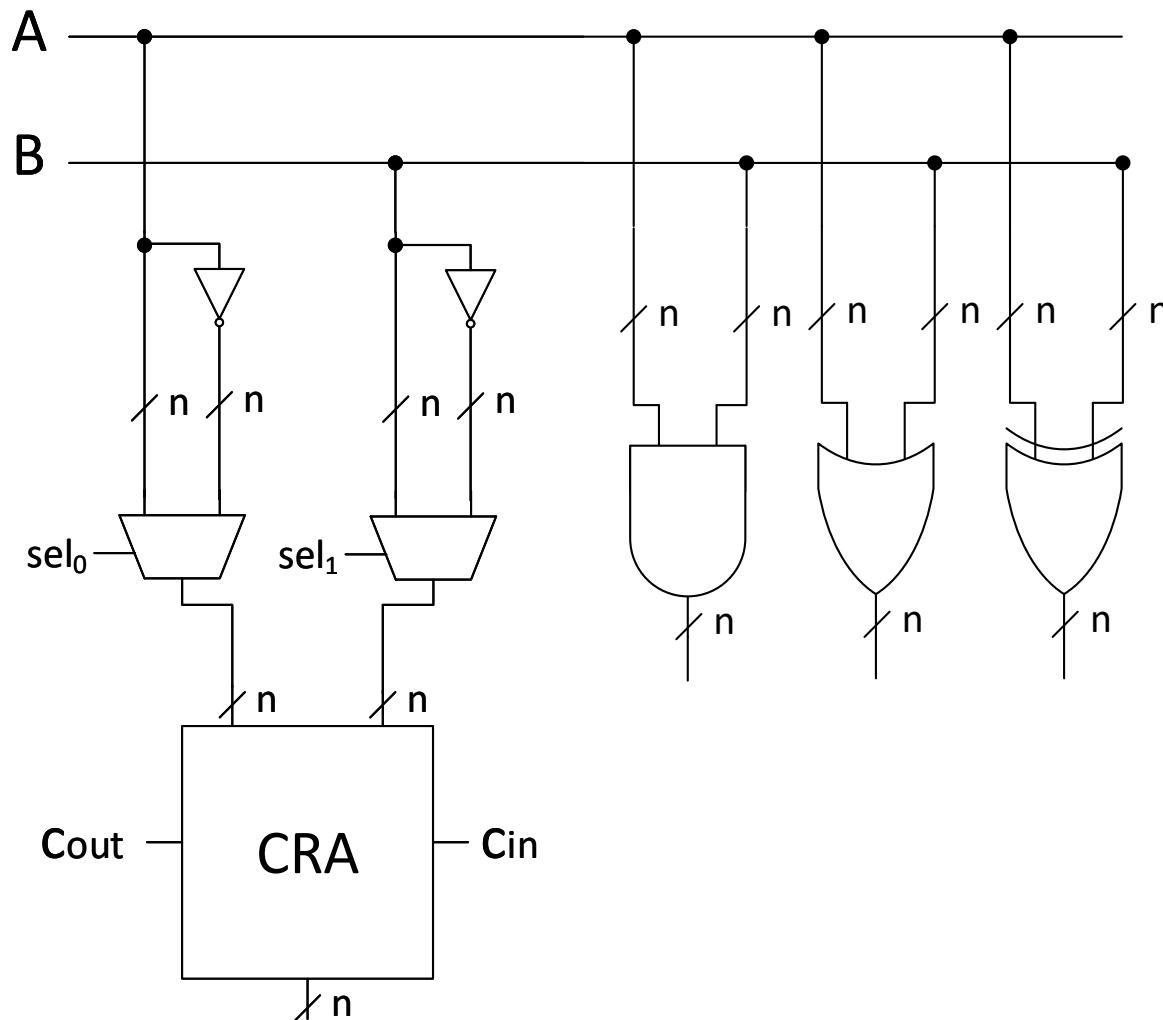
1. Calculation and control system

COMBINATION & CONTROL

ADD SUB AND OR XOR



ADD SUB AND OR XOR



Interconnection (Multiplexing)

- MUX should be usefully interconnected and controlled
- Input set:
 - 8 operations
 - Representable as three-digit opcode (000 ... 111)
 - 3 control lines ($o_2 o_1 o_0$)
- Predictive wiring needed

Interconnection (Multiplexing)

■ (Bad) Example

- Assumption:

Opcode for a-b is 100

Opcode for b-a is 011

Opcode for a+b is 010

- sel_0 and sel_1 require several gates for the generation of the corresponding control signal ($sel_1: o_2 \& \neg o_1 \& \neg o_0$ $sel_0: \neg o_2 \& o_1 \& o_0$)

■ (Better) Example

- Assumption:

Opcode for a-b is 000

Opcode for b-a is 001

Opcode for a+b is 010

- sel_0 and sel_1 together require only *two* gates for the generation of the corresponding control signal ($sel_1: \neg o_1 \& \neg o_0$ $sel_0: \neg o_1 \& o_0$)

Interconnection (Multiplexing), OPCodes

Opcode	function
0 0 0	
0 0 1	
0 1 0	
0 1 1	
1 0 0	
1 0 1	
1 1 0	
1 1 1	

Interconnection (Multiplexing), OPCodes

Opcode	function
0 0 0	$a - b$
0 0 1	$b - a$
0 1 0	$a + b$
0 1 1	
1 0 0	
1 0 1	
1 1 0	
1 1 1	

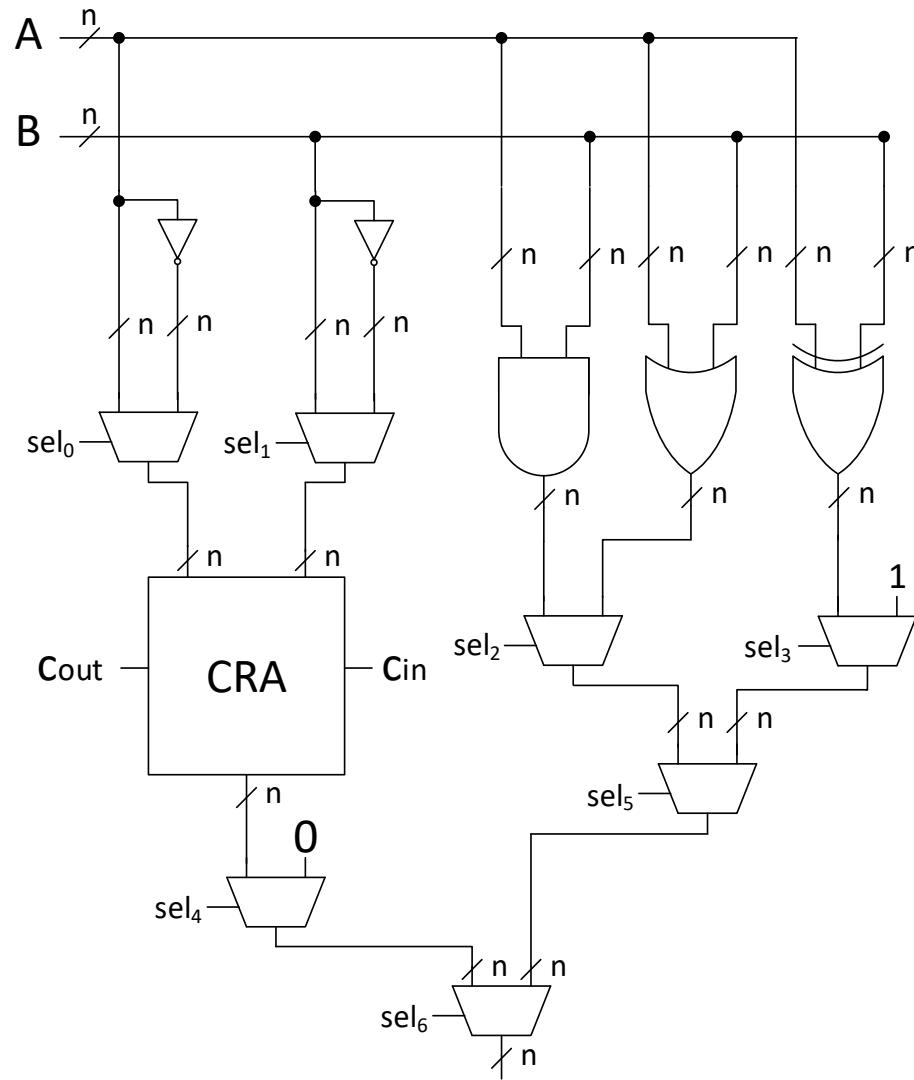
Interconnection (Multiplexing), OPCodes

Opcode	function
0 0 0	$a - b$
0 0 1	$b - a$
0 1 0	$a + b$
0 1 1	
1 0 0	$a \wedge b$
1 0 1	$a \vee b$
1 1 0	$a \oplus b$
1 1 1	

Interconnection (Multiplexing), OPCodes

Opcode	function
0 0 0	$a - b$
0 0 1	$b - a$
0 1 0	$a + b$
0 1 1	0
1 0 0	$a \wedge b$
1 0 1	$a \vee b$
1 1 0	$a \oplus b$
1 1 1	1

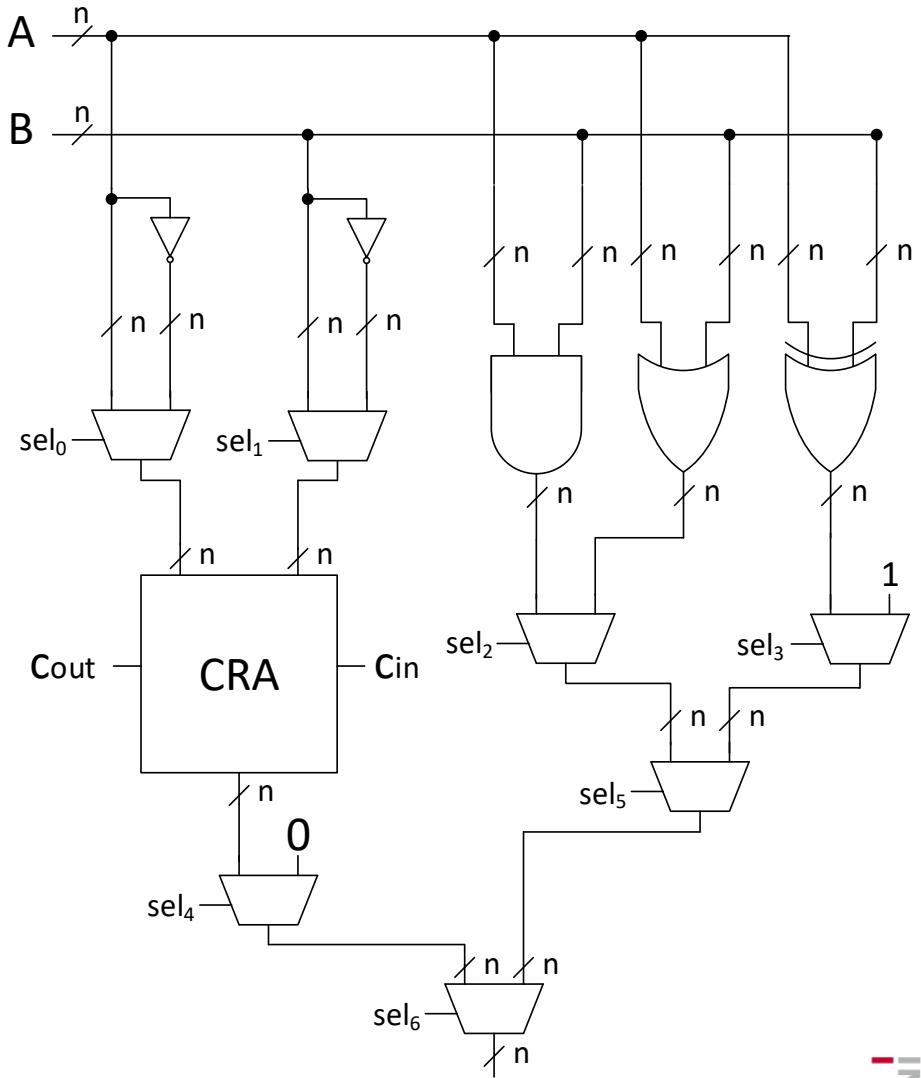
Interconnection (Multiplexing)



Interconnection (Multiplexing)

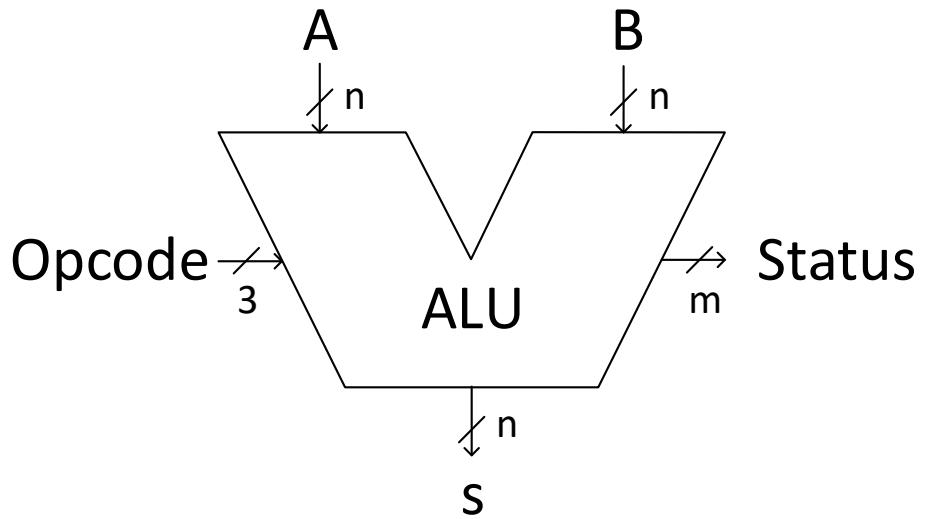
- sel_0
 - $\neg o_1 \wedge o_0$
- sel_1
 - $\neg o_1 \wedge \neg o_0$
- sel_2
 - o_0
- sel_3
 - o_0
 - $\neg o_1$
- sel_4
 - $o_1 \wedge o_0$
- sel_5
 - o_1
- sel_6
 - o_2
- c_{in}

Opcode	function
0 0 0	$a - b$
0 0 1	$b - a$
0 1 0	$a + b$
0 1 1	0
1 0 0	$a \wedge b$
1 0 1	$a \vee b$
1 1 0	$a \oplus b$
1 1 1	1



ALU – Arithmetic Logic Unit

- A, B: Input words, n Bit wide
- S: Output words, n Bit wide
- Opcode: 3 Bit wide
- Status: m Bit wide
U.a.:
 - Carry bit
 - Overflow bit
 - Zero bit

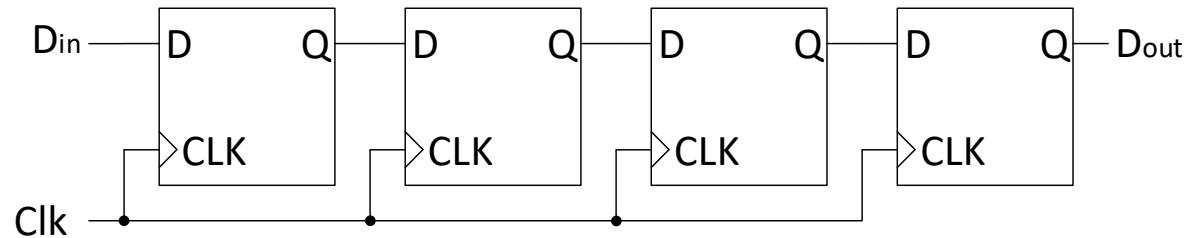
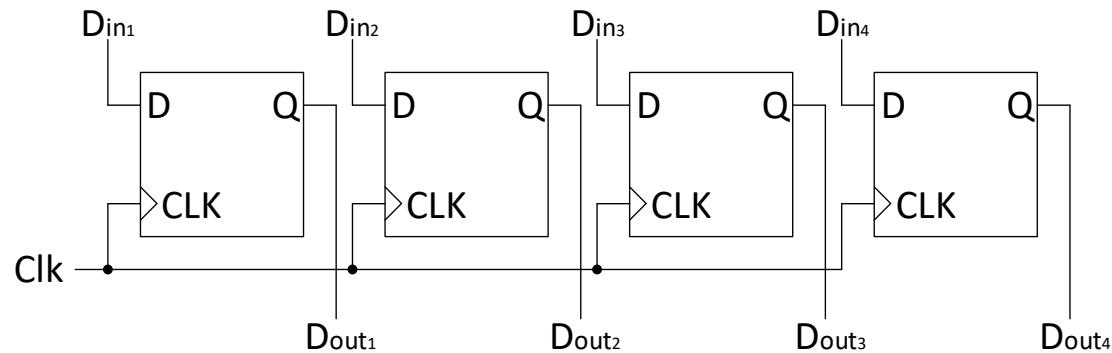


1. Calculation and control system

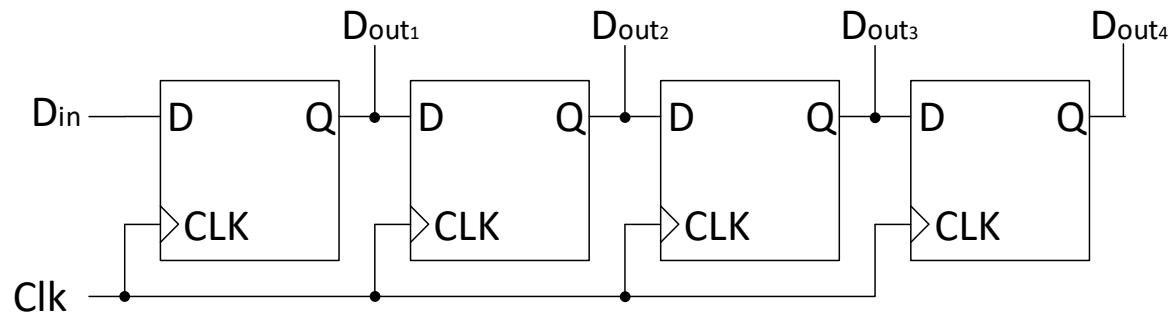
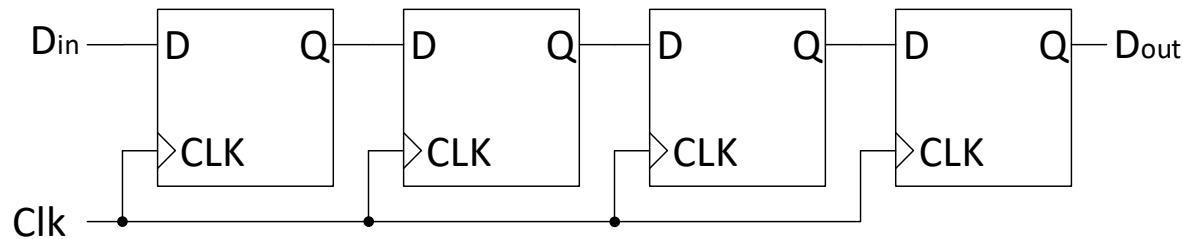
REGISTER

Register

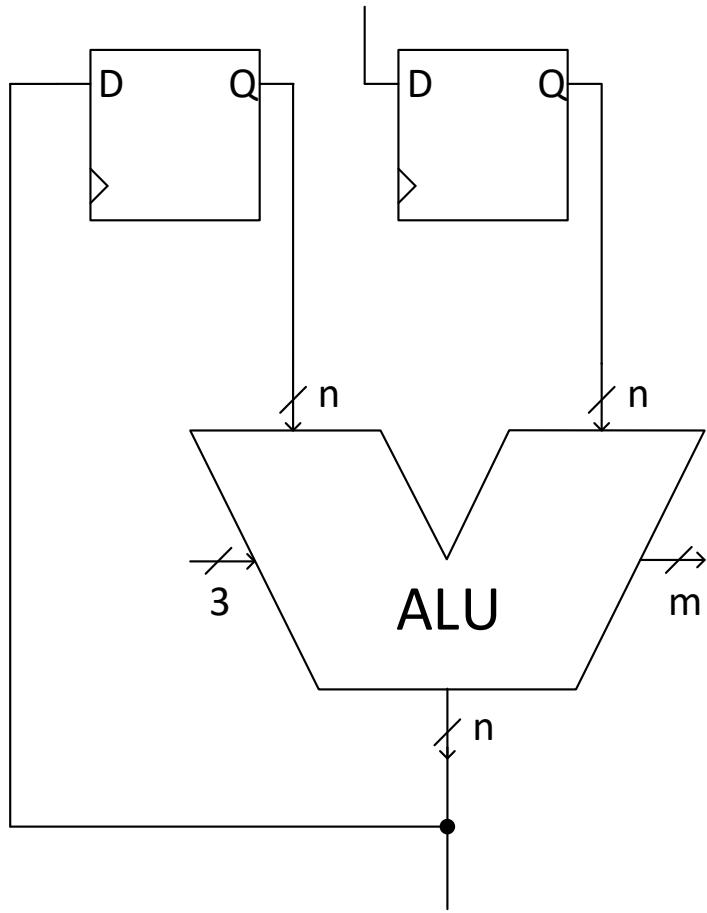
- Already known: D-Latch
- Register: more D-Latches adjacent (or consecutively)



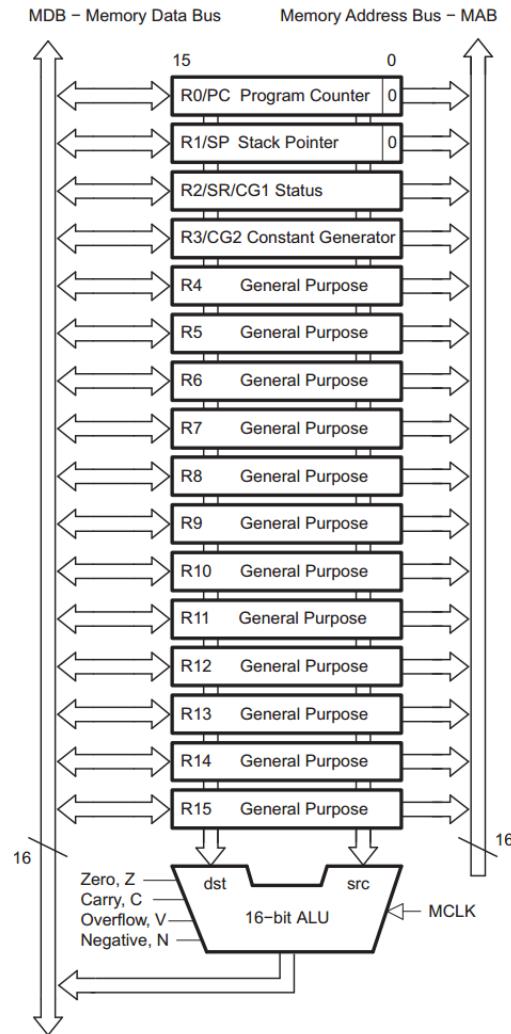
Register (brief digression: shift register)



Register + ALU



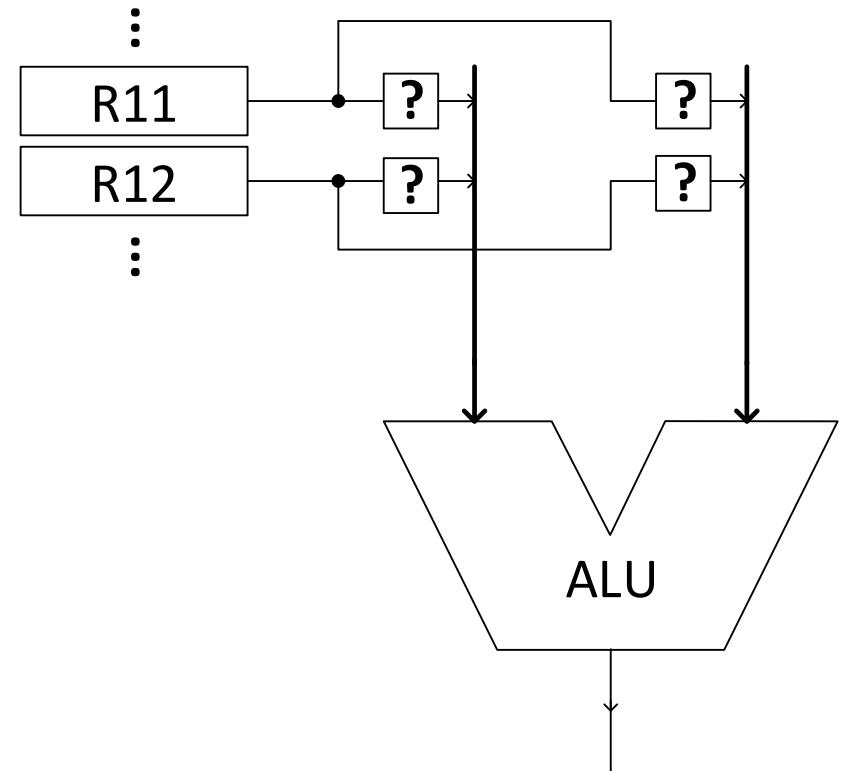
Register + ALU (MSP430)



Source:
MSP430x2xx Family – User's Guide

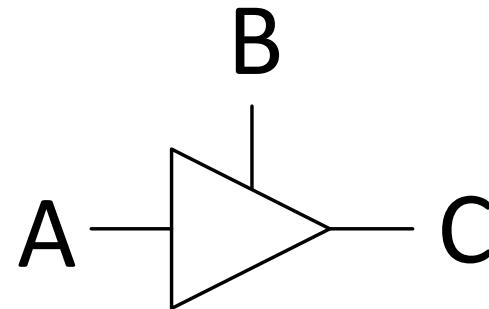
Register + ALU

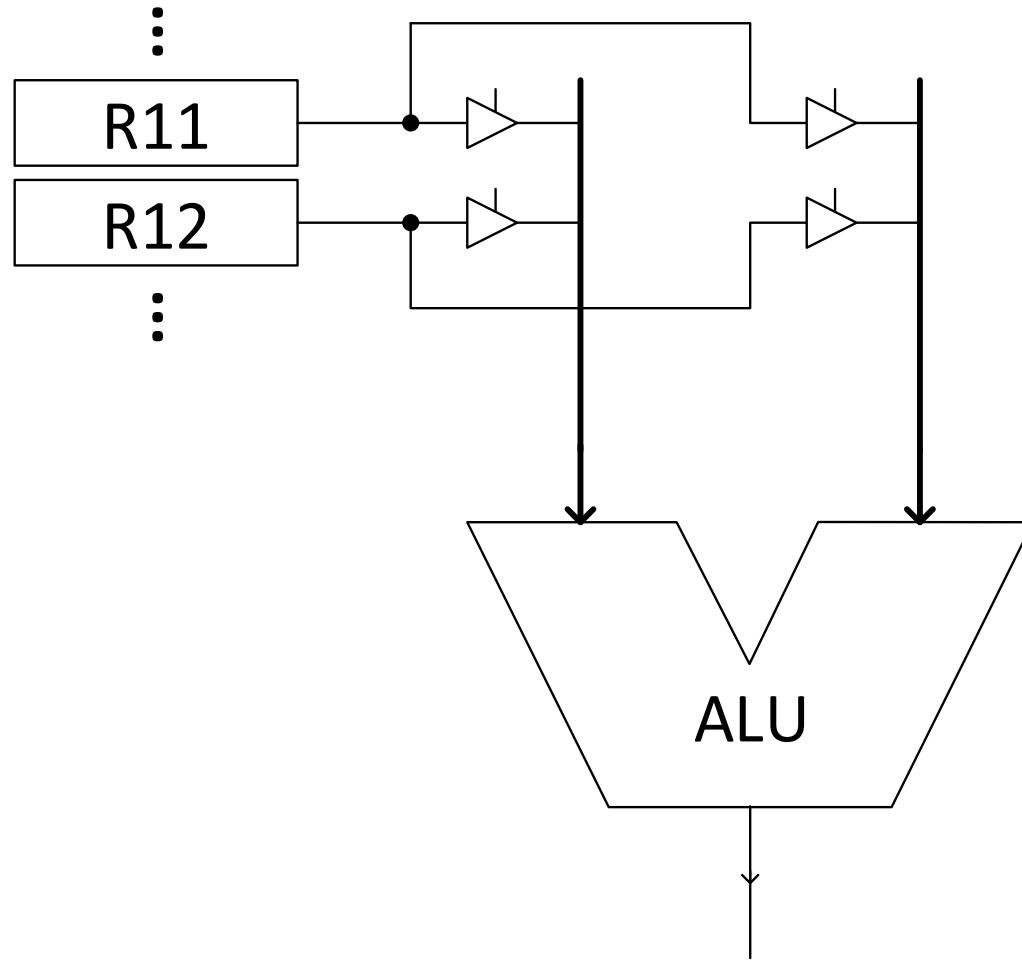
- Simultaneous writing causes problems
- How can exclusive write access be guaranteed?



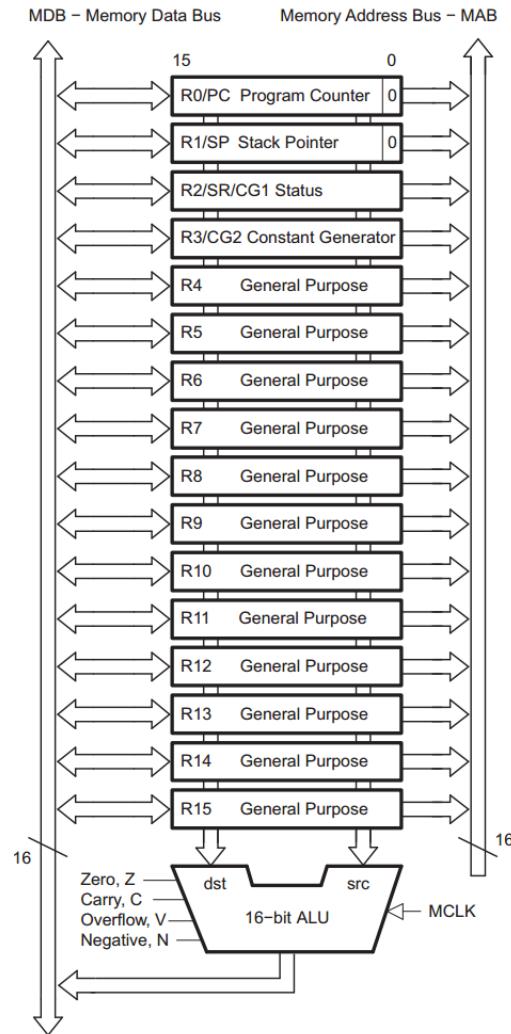
Tri-state driver

A	B	C
0	0	Z
1	0	Z
0	1	0
1	1	1



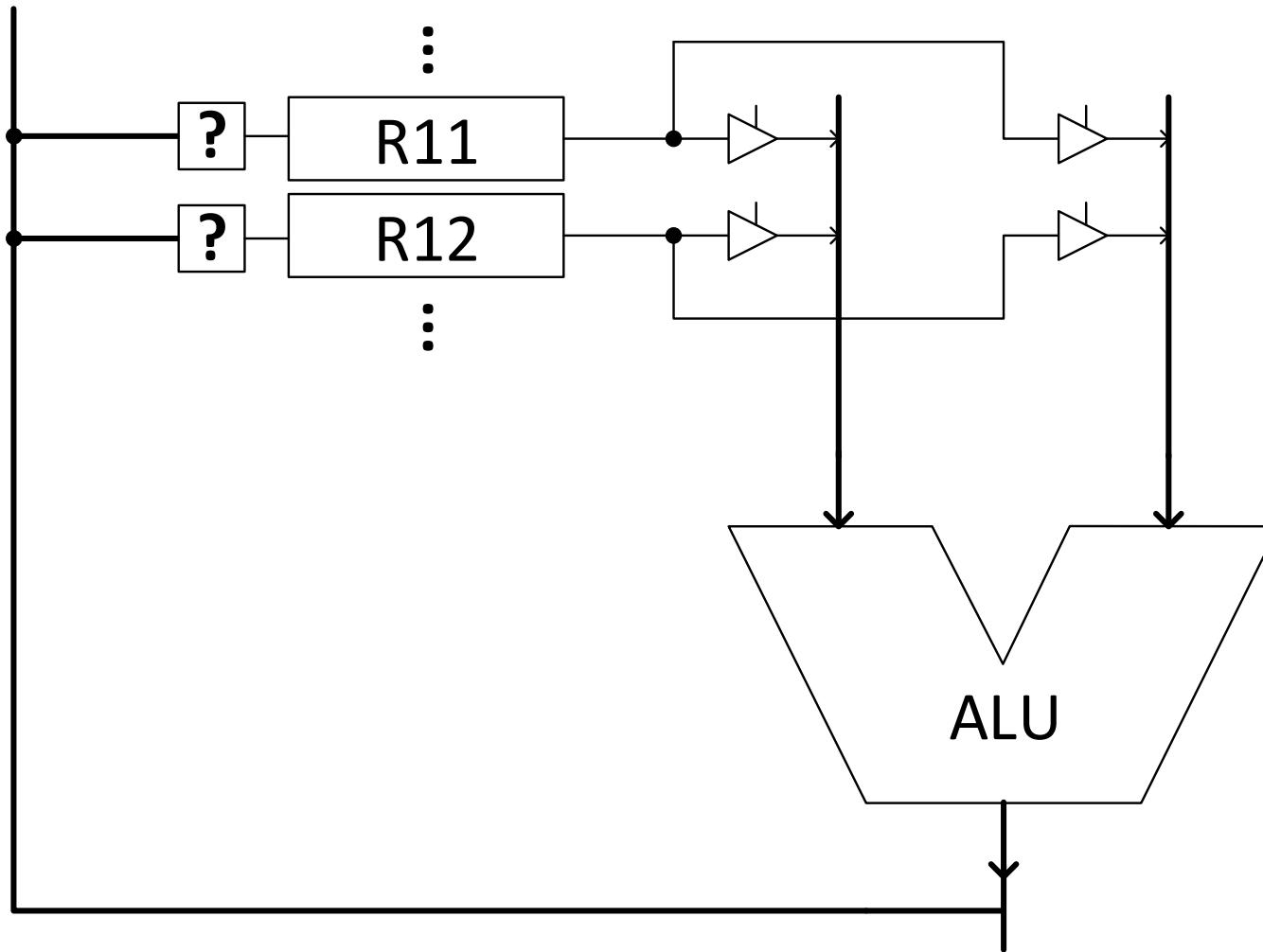


Register + ALU (MSP430)

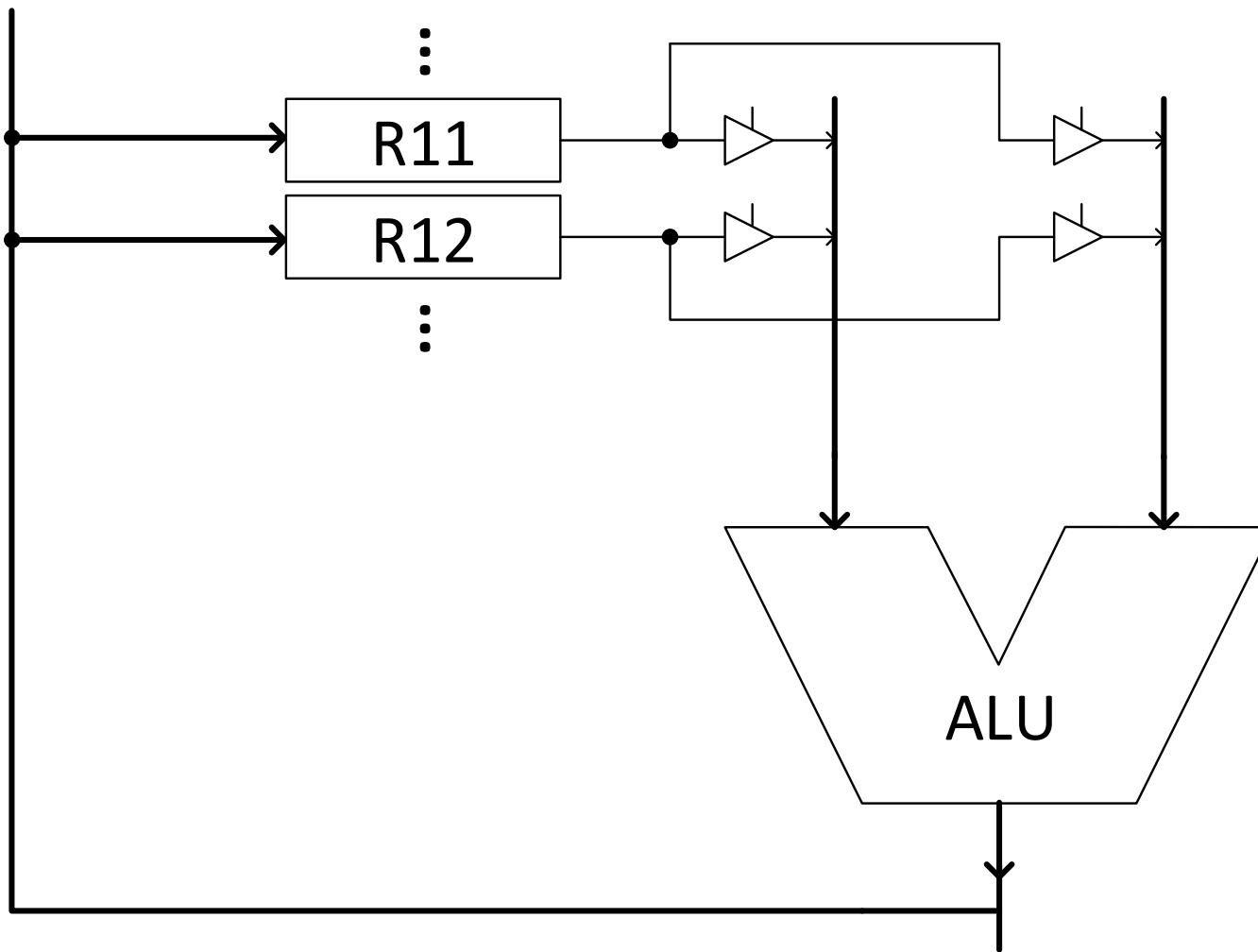


Source:
MSP430x2xx Family – User's Guide

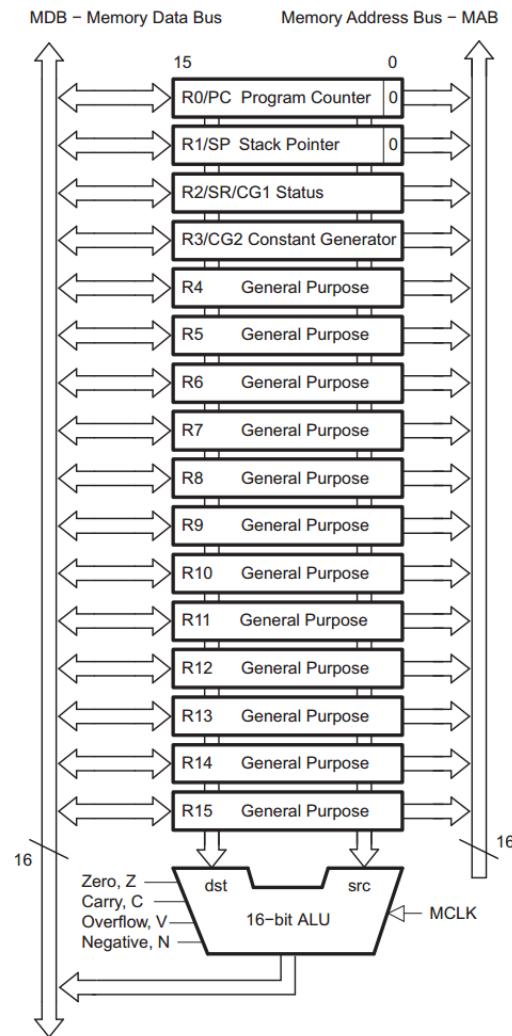
Register + ALU



Register + ALU



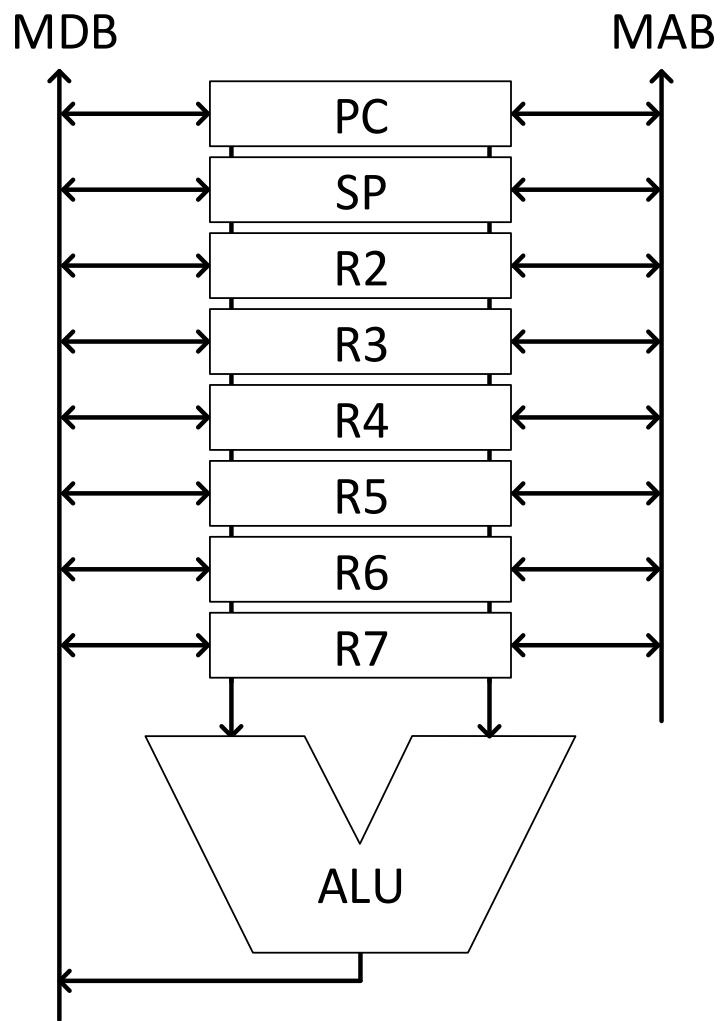
Register + ALU (MSP430)



Source:
MSP430x2xx Family – User's Guide

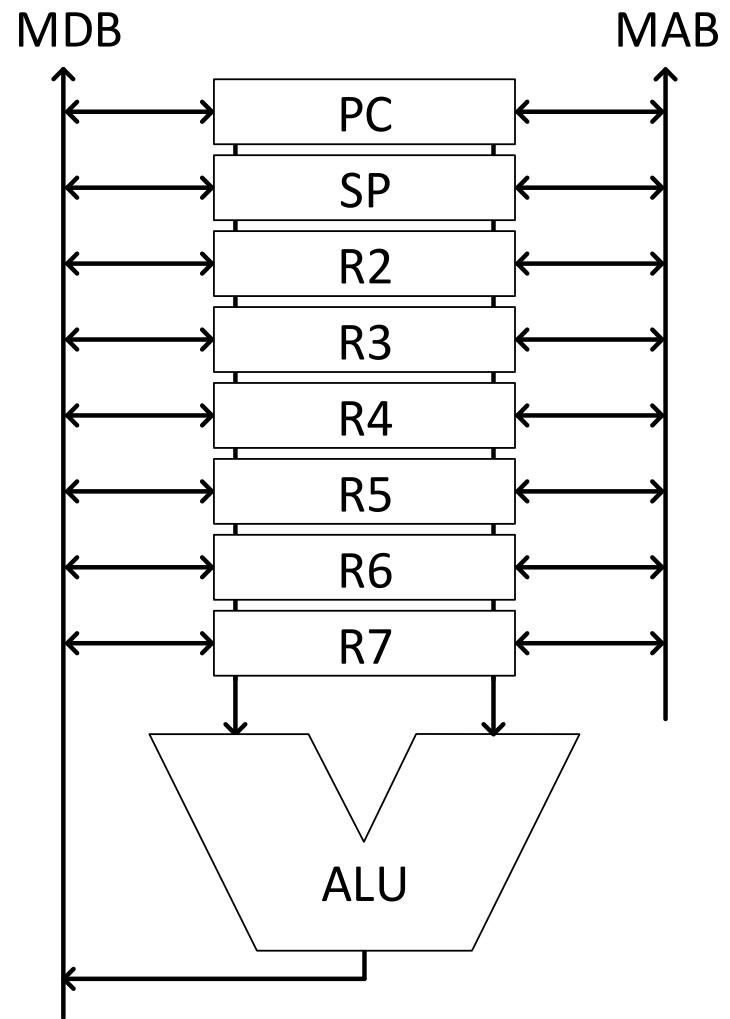
Register + ALU („Mini“-MSP430)

- Small version of the MSP430x2xx-CPU
- Same structure, only simplified
 - R8 ... R15 missing
 - „Constant Generator“ missing
 - Status Register hidden (for simplicity, we consider states to be stored within ALU)



Register + ALU („Mini“-MSP430)

- PC: Program Counter
 - Points to the next instruction to be executed. Initial value : 0
- SP: Stack Pointer
 - Points to the lastest address of the stack memory section (see chapter programming).
 - Initial value: the end of the RAM
- MAB: Memory Address Bus
 - Here, we apply the memory address to be read or written
- MDB: Memory Data Bus
 - Data to be read or written (applied by memory or processor)



1. Calculation and control system

MEMORY & RAM

General Layout of the Memory in the MSP430 family

- 0x????₂ – 0xFFFF: Flash
 - 0xFFE0 (or 0xFFC0) – 0xFFFF: interrupt vectors
 - Officially 0xFFC0, but the area is not used up to 0xFFE0 and, therefore, is in principle available for program memory
 - ? is still open, is set by the size of the memory (Flash, RAM).
- 0x200 – 0x????₁: RAM
- 0x100 – 0x1FF: 16-bit peripherals, e.g. ADC or TIMER
- 0x10 – 0xFF: 8-bit peripherals, e.g. GPIO
- 0x0 – 0xF: special function registers
- The data sheet says furthermore:
 - 512 Byte RAM
 - 16 Kilobyte Flash

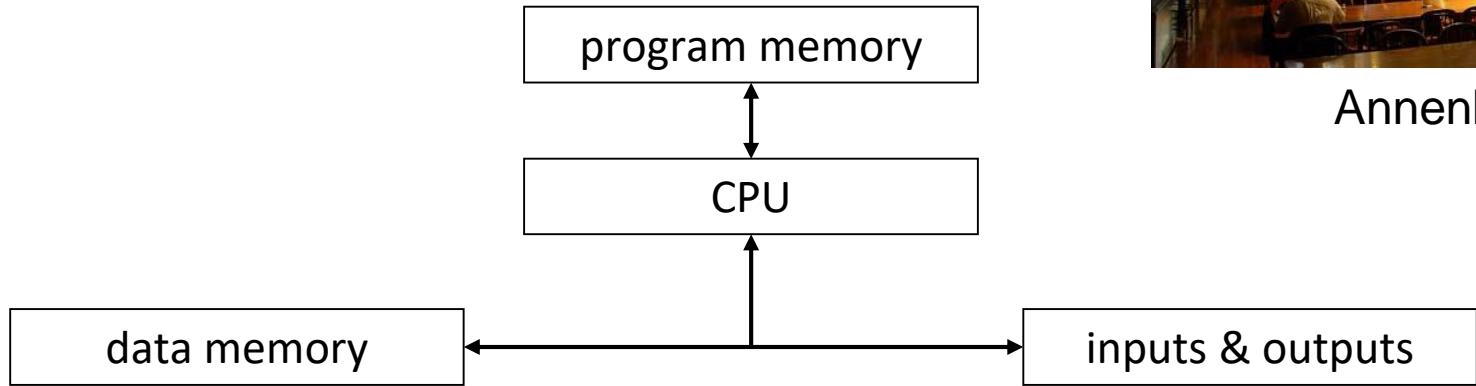
Memory layout in the MSP430, special as used in the course

- 0xC000 – 0xFFFF: Flash
 - 0xFFE0 (or 0xFFC0) – 0xFFFF: interrupt vectors
 - Officially 0xFFC0, but the area is not used from 0xFFE0 onwards and is therefore basically available for program memory
- 0x200 – 0x3FF: RAM
- 0x100 – 0x1FF: 16-bit peripherals, e.g. ADC or TIMER
- 0x10 – 0xFF: 8-bit peripherals, e.g. GPIO
- 0x0 – 0xF: special function registers
- Between 0x3FF and 0xC000: not connected
 - Same Layout for all MSP430-Processores
 - More Storage → less free space
 - If the flash memory is larger than the space between HIGH (RAM) and 0xFFFF, it will continue from 0x10000 to 0x1FFFF. This is outside of the 16 bit range and can be fixed by workarounds (e.g. constant number generator).

Memory connections



Annenberg Hall

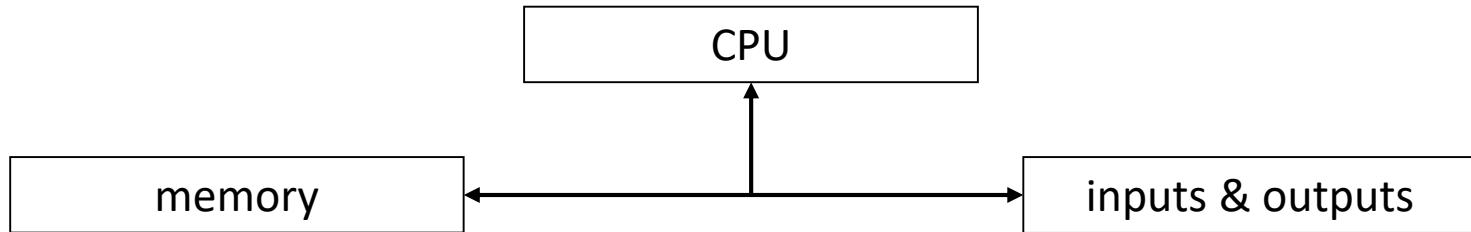


- Harvard architecture
- Data and program memory addressable by different buses

Memory connections



John von Neumann (1940)



- Von-Neumann architecture
- Data and program memory addressable by the same bus

Memory connections

Harvard (separate)

- ✓ Better capabilities for parallelization
- ✓ Easier to implement access rights

✗ Complex control

Von-Neumann (combined)

- ✓ Easier (cheaper) construction
- ✓ Reduced management requirements
- ✓ Memory can be utilized optimally

✗ Von Neumann bottleneck

Memory connections

- Von-Neumann dominates nowadays
- But: Harvard typical advantages are partially “upgraded” in the software domain
 - Memory access rights
 - Cache memories, which are connected by additional busses
- Von-Neumann bottleneck is softened by more internal registers:
 - This is exactly why the MSP430 CPU has 16 CPU internal registers

MSP430x2xx-Userguide (p.24):

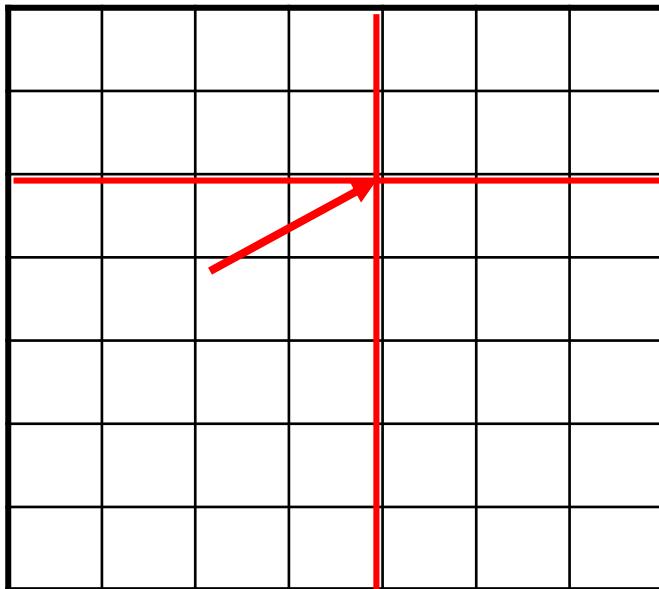
“Large register file eliminates working file bottleneck”

Types of memory

- Volatile memory
 - SRAM
 - DRAM
- Non-volatile memory
 - Magnetic memory
 - (PROM)
 - (EPROM)
 - EEPROM (special FLASH)

Matrix storage principle

- If we arrange memory cells in a matrix form so that an element is addressed if exactly two lines at the intersection are active (one vertical and one horizontal line), we require only \sqrt{k} lines to address k cells; e.g. 128 horizontal and 128 vertical lines to generate 16,384 crossing points (16 kilobytes of memory)



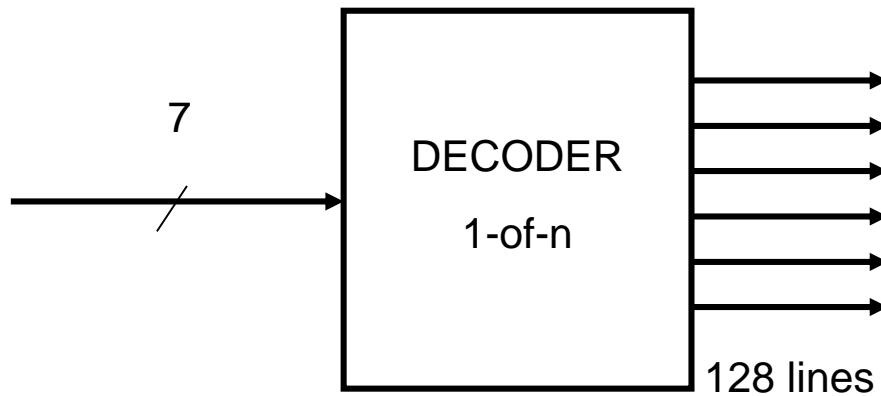
128 horizontal lines

128 vertical lines

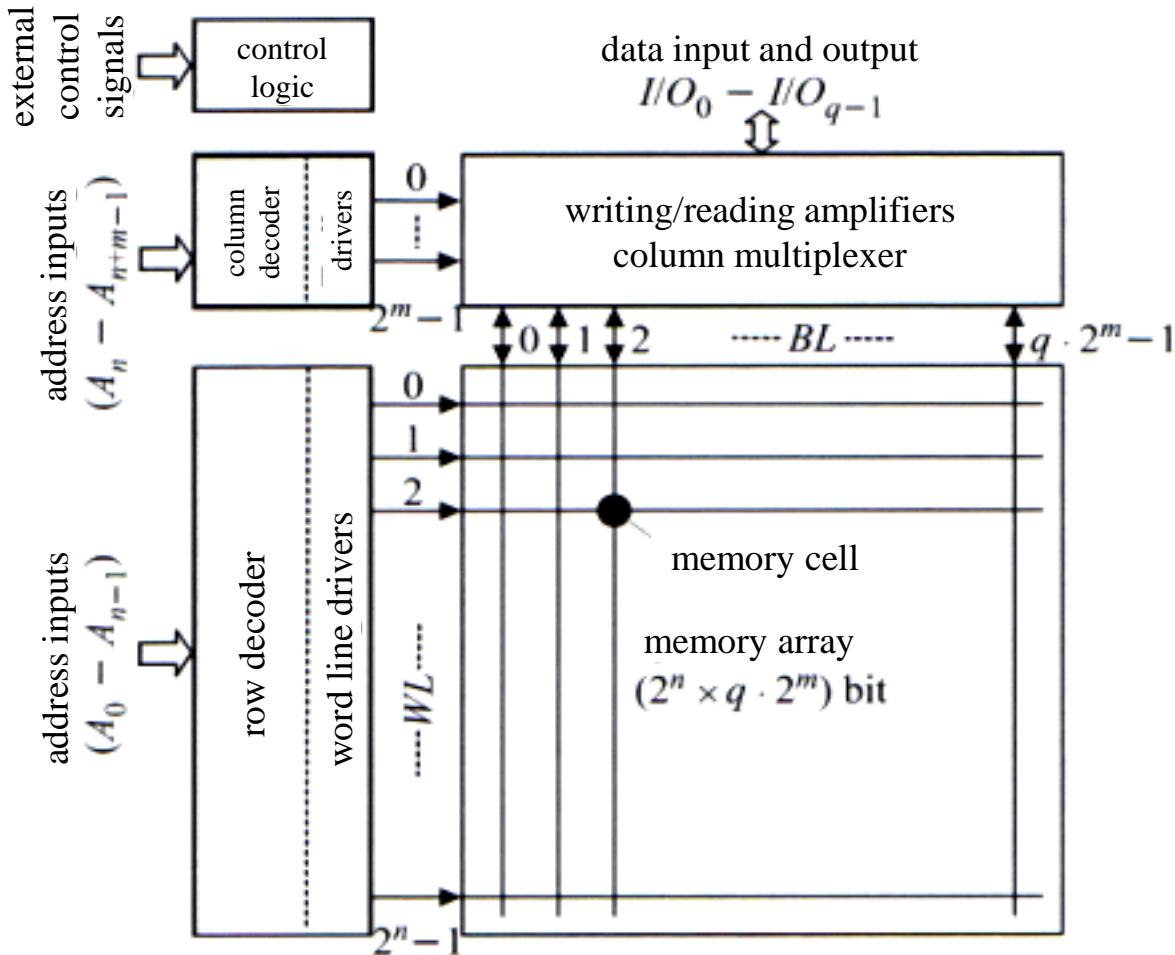
$2^{14} = 16k$ intersections

Arrow points to the selected cell

Decoder for row and column selection

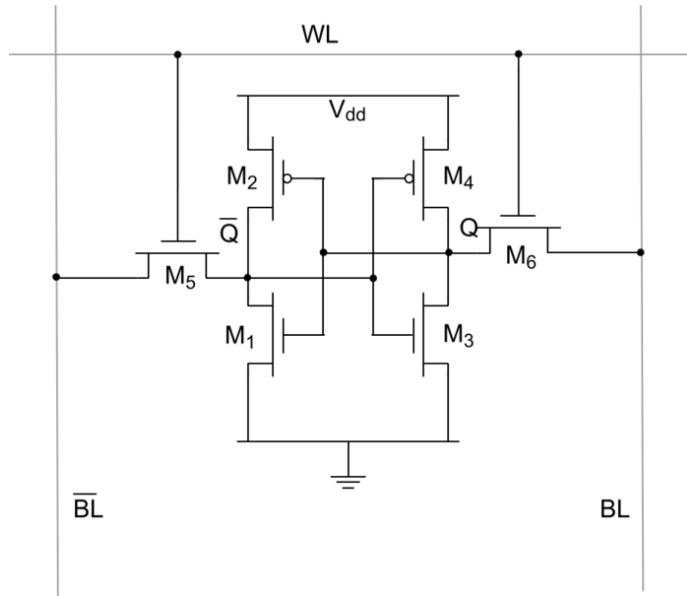


Block diagram of a memory array



Volatile memory: SRAM

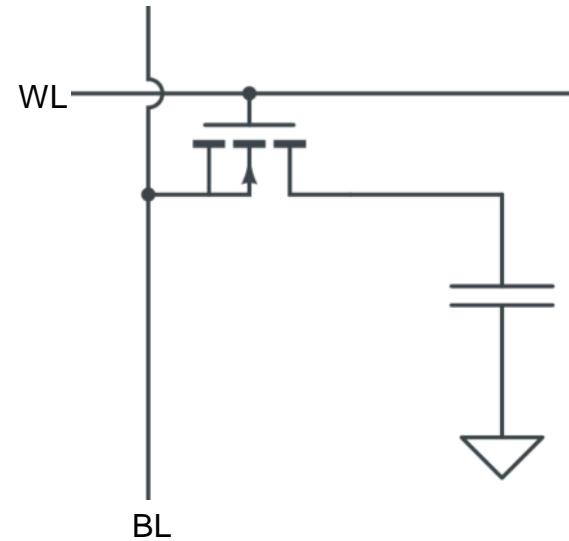
- Static Random Access Memory
- Storing the value in a state of a flip flop
- Fast
- Endless retention time (as long as the power supply is not interrupted)
- Low power consumption while retaining
- Large (6 transistors per cell)
- Expensive



Source: Abelsson / Wikimedia (CC BY-SA 3.0)
<http://commons.wikimedia.org/wiki/File:6t-SRAM-cell.png>

Volatile memory: DRAM

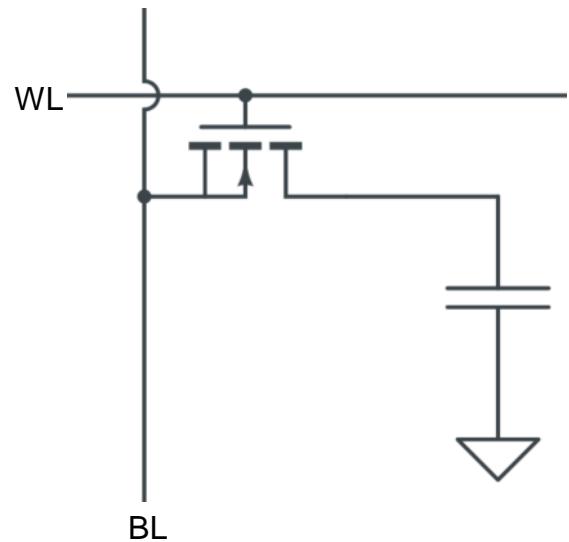
- Dynamic Random Access Memory
- Storing of the value in charge in a capacitor (capacitance in the range of several fF)
- Slower due to loading times
- Limited retention time (around 50 ms) due to leakage currents: refresh required
- Result: higher power consumption while retaining the memory content
- output amplification by read amplifiers required (e.g. by an arrangement similar to the SRAM cell without M5 and M6 (see previous slide))



Volatile memory: DRAM

- Dynamic Random Access Memory
- Storing of the value in a capacitor
(capacitance in the range of several fF)

- Small
- Cheap



Persistent memory: Magnetic memory

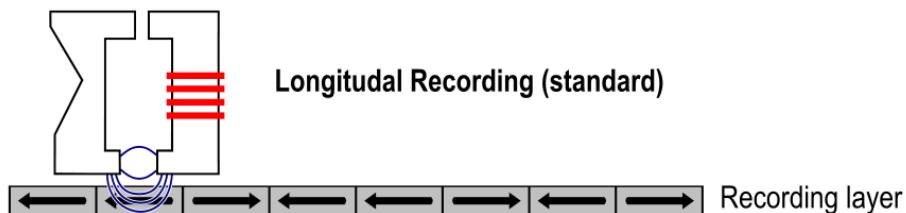
- Storage of data by magnetic polarization
- Bit and/or blockwise access (depending on implementation)

- Simple manufacturing of very high storage densities
- Cheap

- Example: Hard Drives

Persistent Storage: Magnetic memory

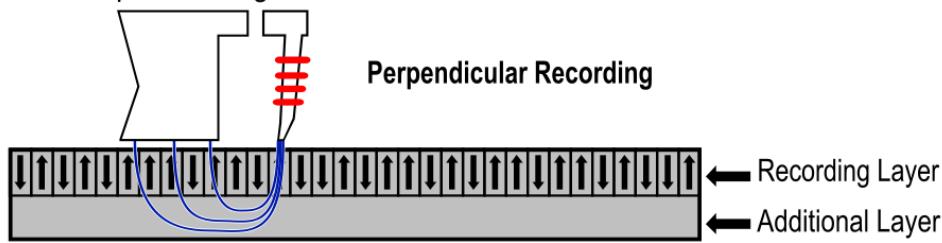
"Ring" writing element



Longitudinal Recording (standard)

Recording layer

"Monopole" writing element



Perpendicular Recording

Recording Layer
Additional Layer



Hard drives use magnetic memory to store giga- and terabytes of data in computers.

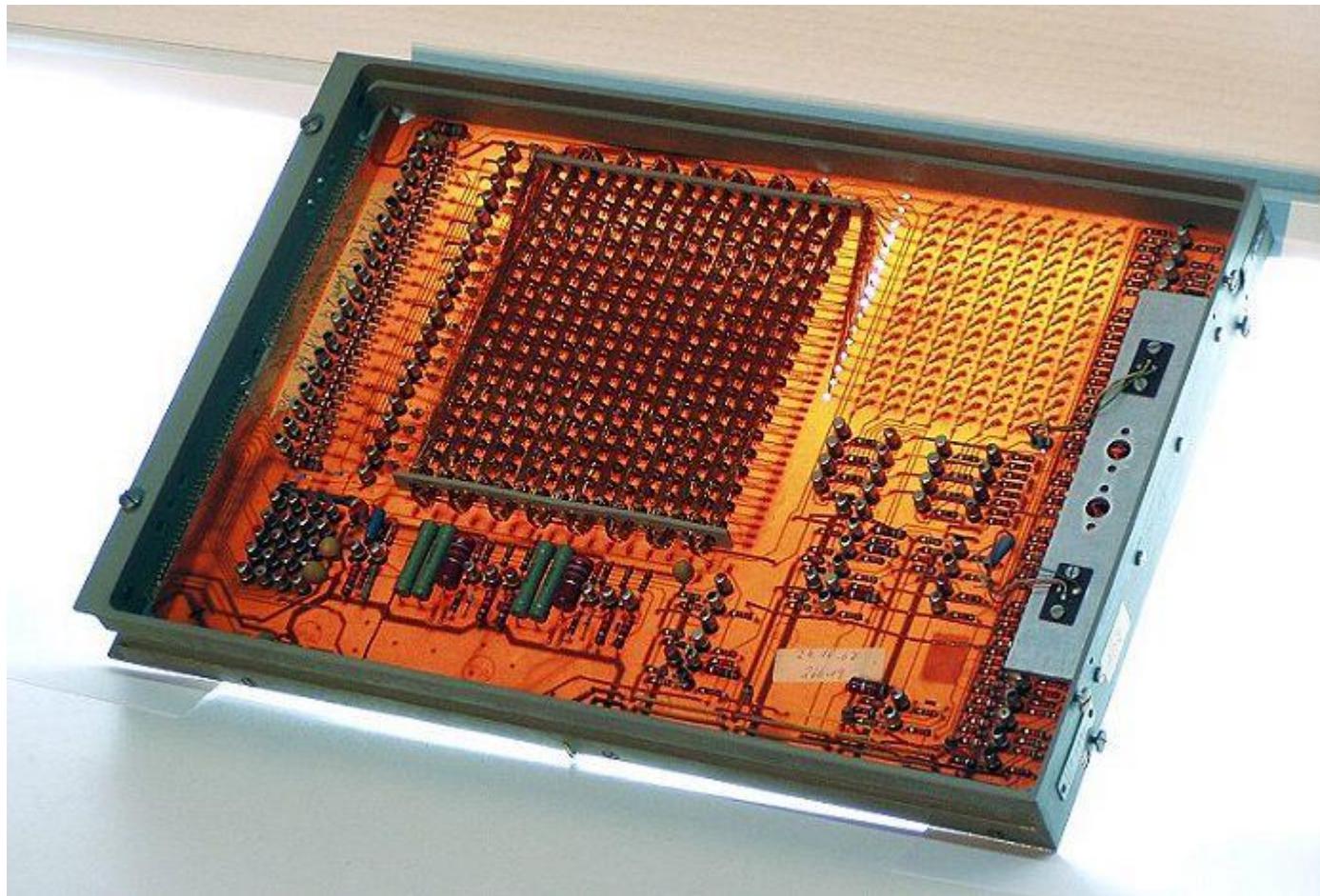
Source: TylzaeL - Wikimedia

<http://commons.wikimedia.org/w/index.php?title=File:Perpendicular-eng.jpg>

Persistent memory: ROM

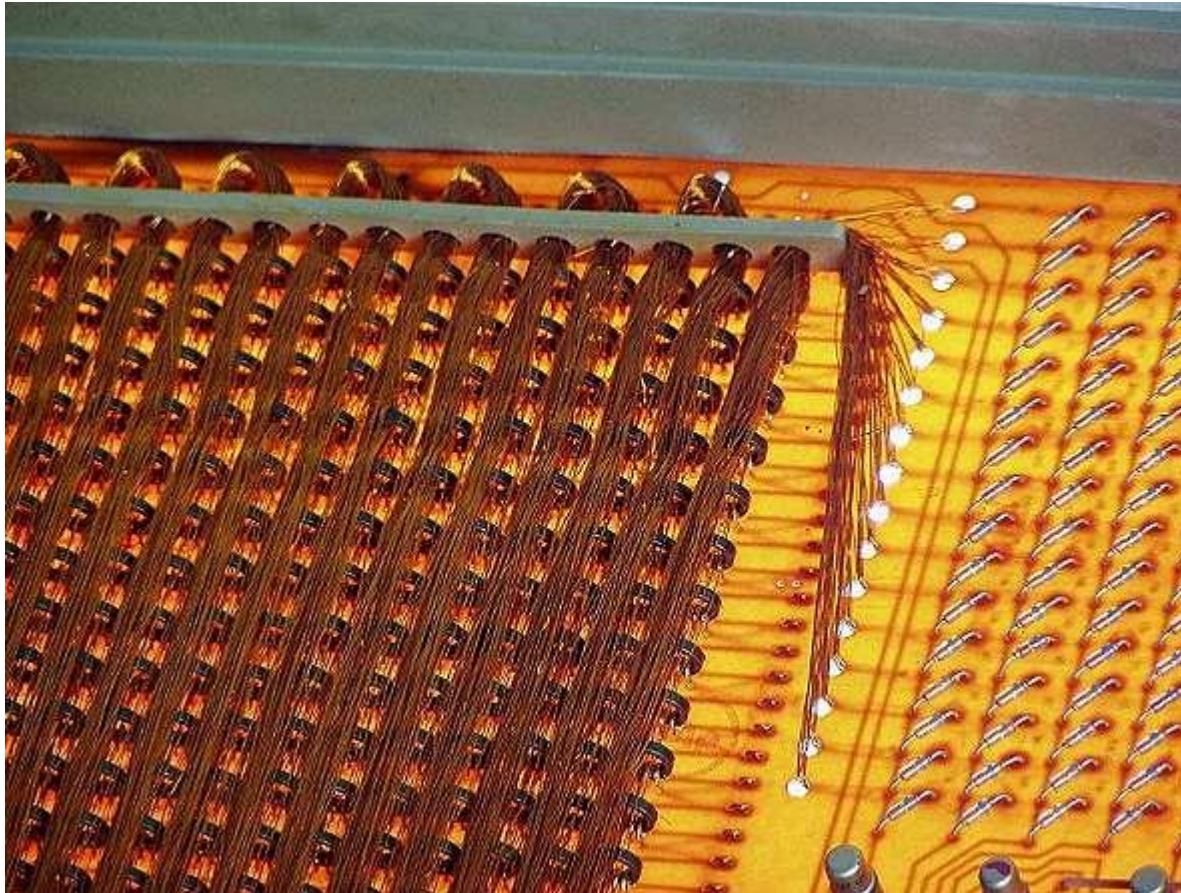
- Read-Only-Memory
- Data stored in the wiring / the physical construction
- Memory content is already determined in production
- Not changeable
- Example: threaded ROM of Nixdorf

Persistent memory: ROM



Source: technikum29.de

Persistent memory: ROM

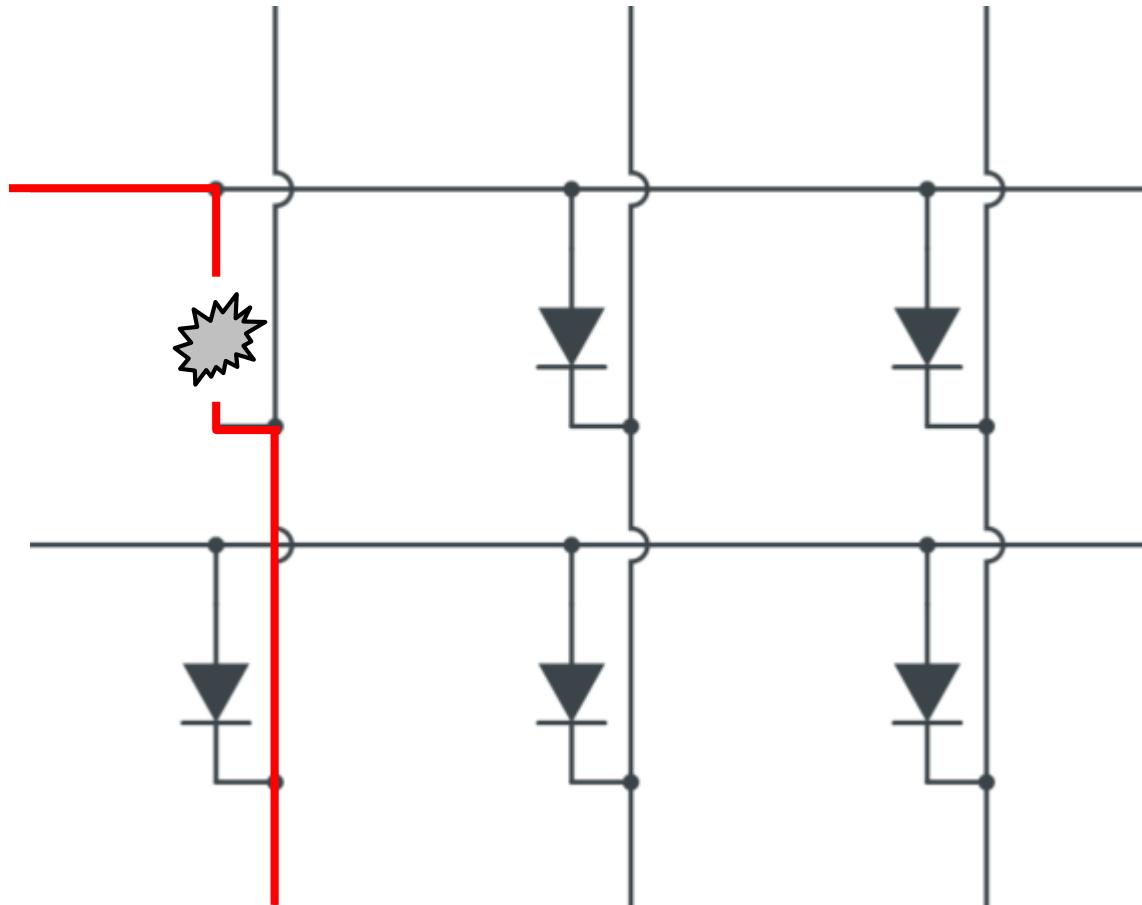


Source: technikum29.de

Persistent memory: PROM (archaic)

- Programmable Read-Only-Memory
- One-time programmable
- Not changeable afterwards
- "Burning in" of the program by means of a higher current than during normal operation
- Irreversible
- Example: BIOS in early computers

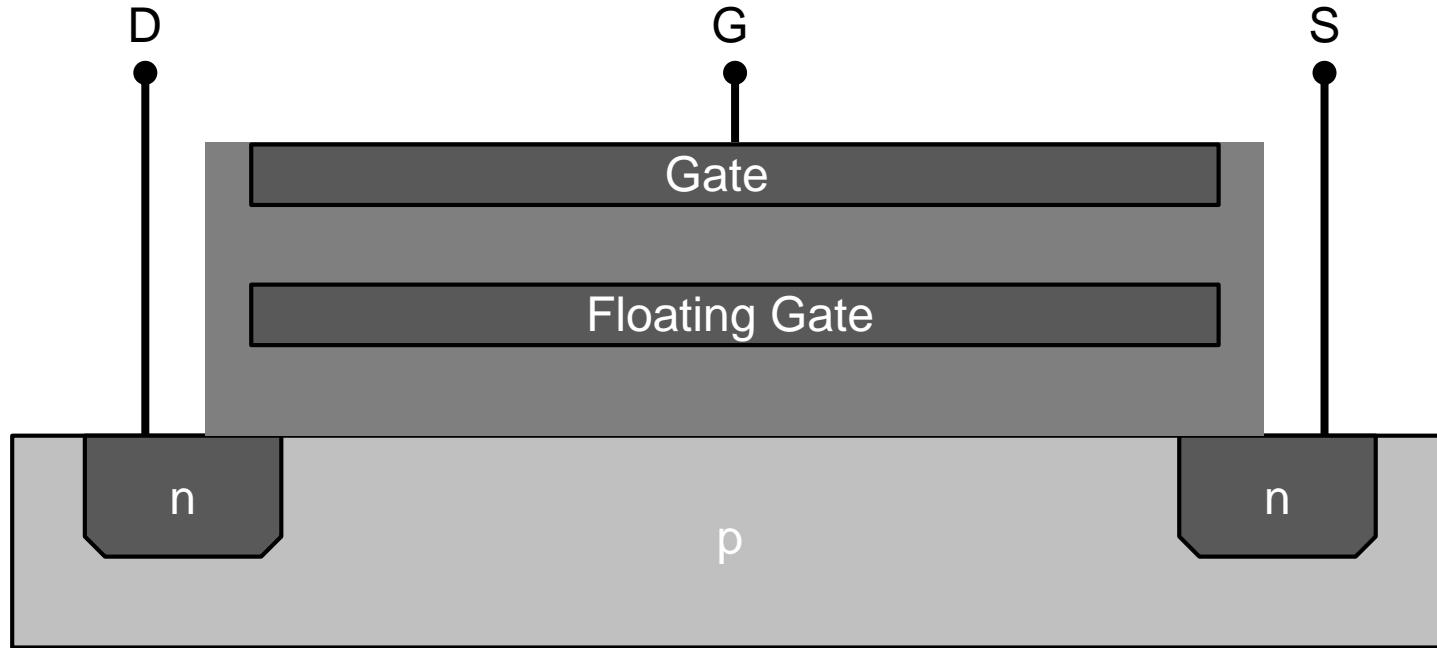
Persistent memory: PROM (archaic)



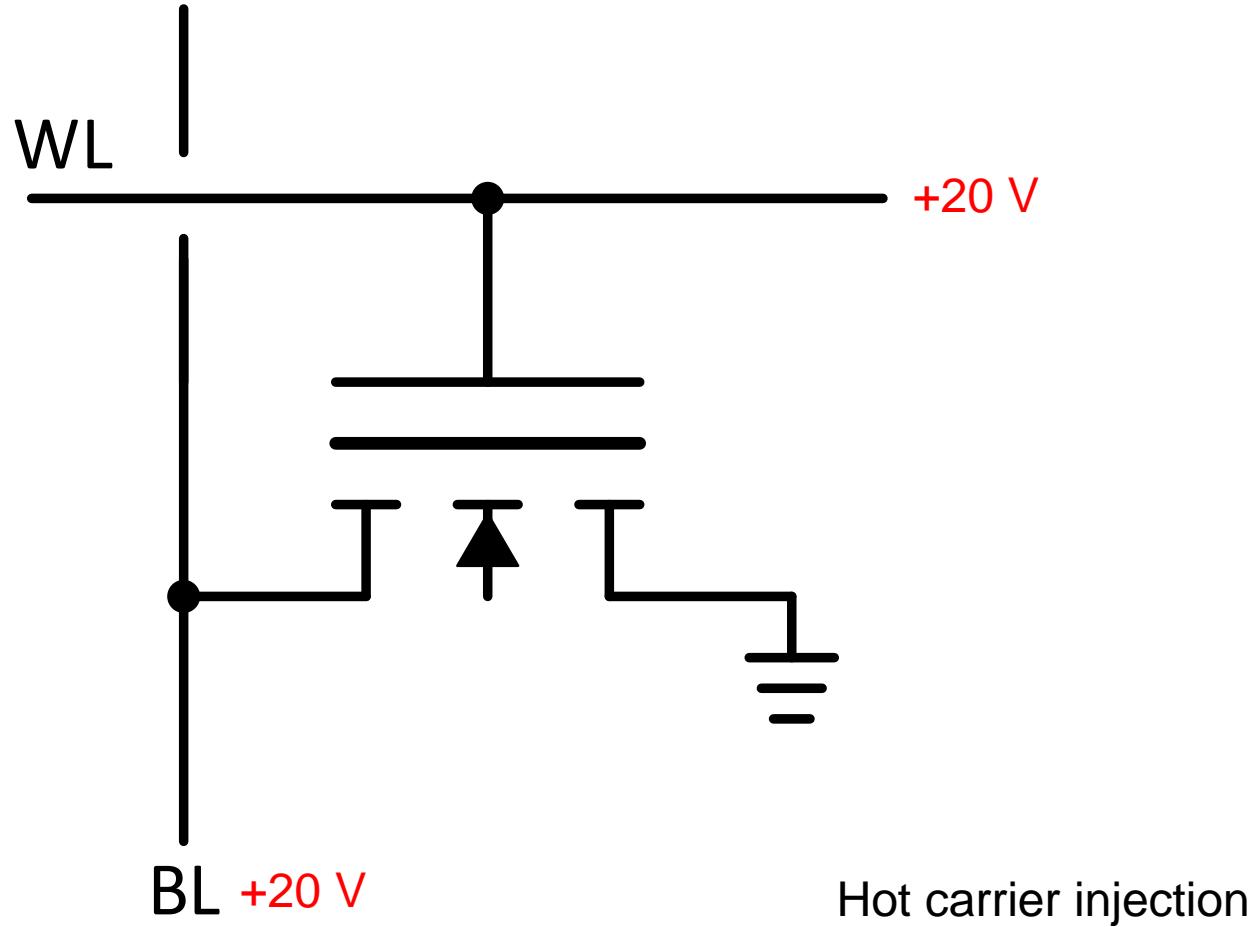
Persistent memory: EPROM (archaic)

- Erasable Programmable Read-Only Memory
- Retention time within several months to decades
- Programmable by floating gate technology
- But: much higher voltages are needed than for RAM
- Limited number of read / write cycles
- Erasable by UV light (ionization of the semiconductor, voltage flow)
- Example: Early microcontroller memory

Persistent memory: EPROM (archaic)



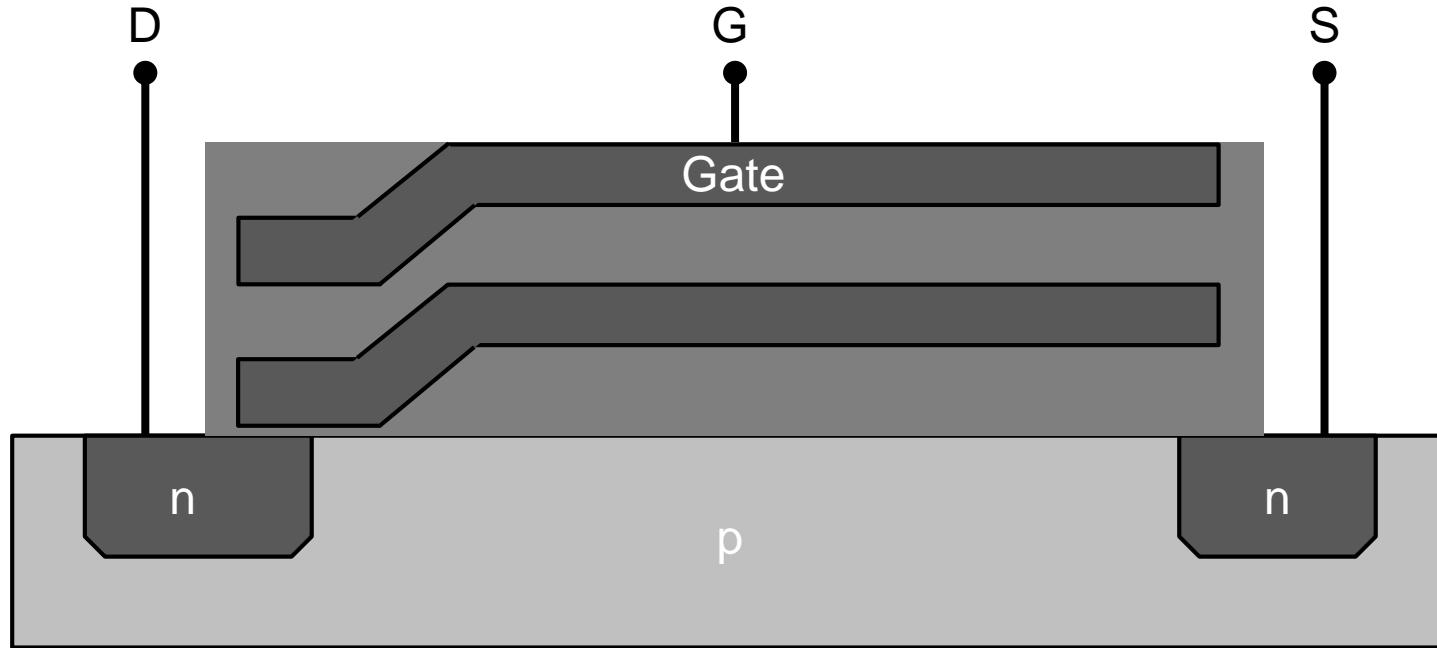
Persistent memory: EPROM (archaic)



Persistent memory: EEPROM

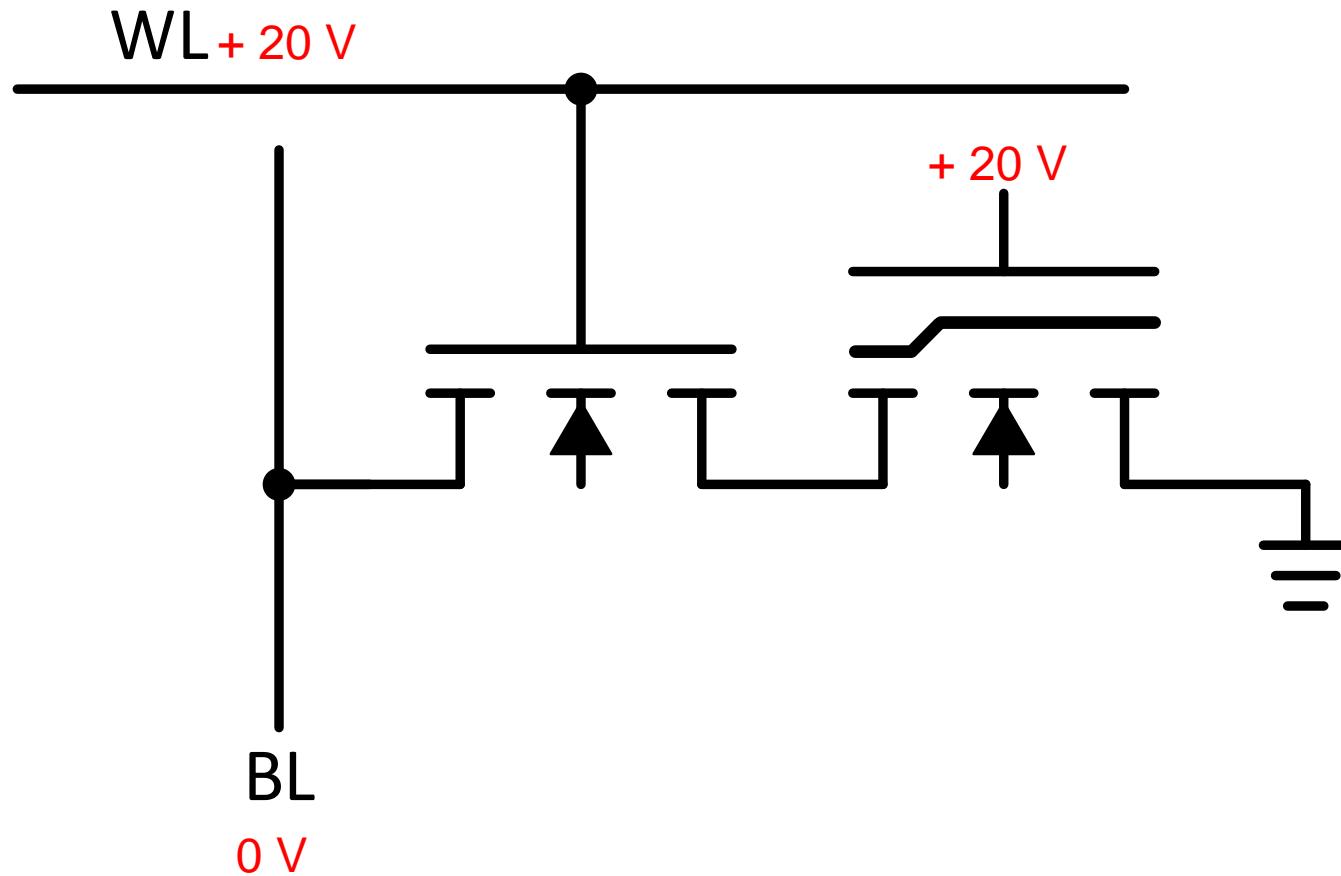
- Further evolution of the EPROM
- Modified design allows for electronic erasure
- Examples: Non-volatile data memory in microcontrollers

Persistent memory: EEPROM

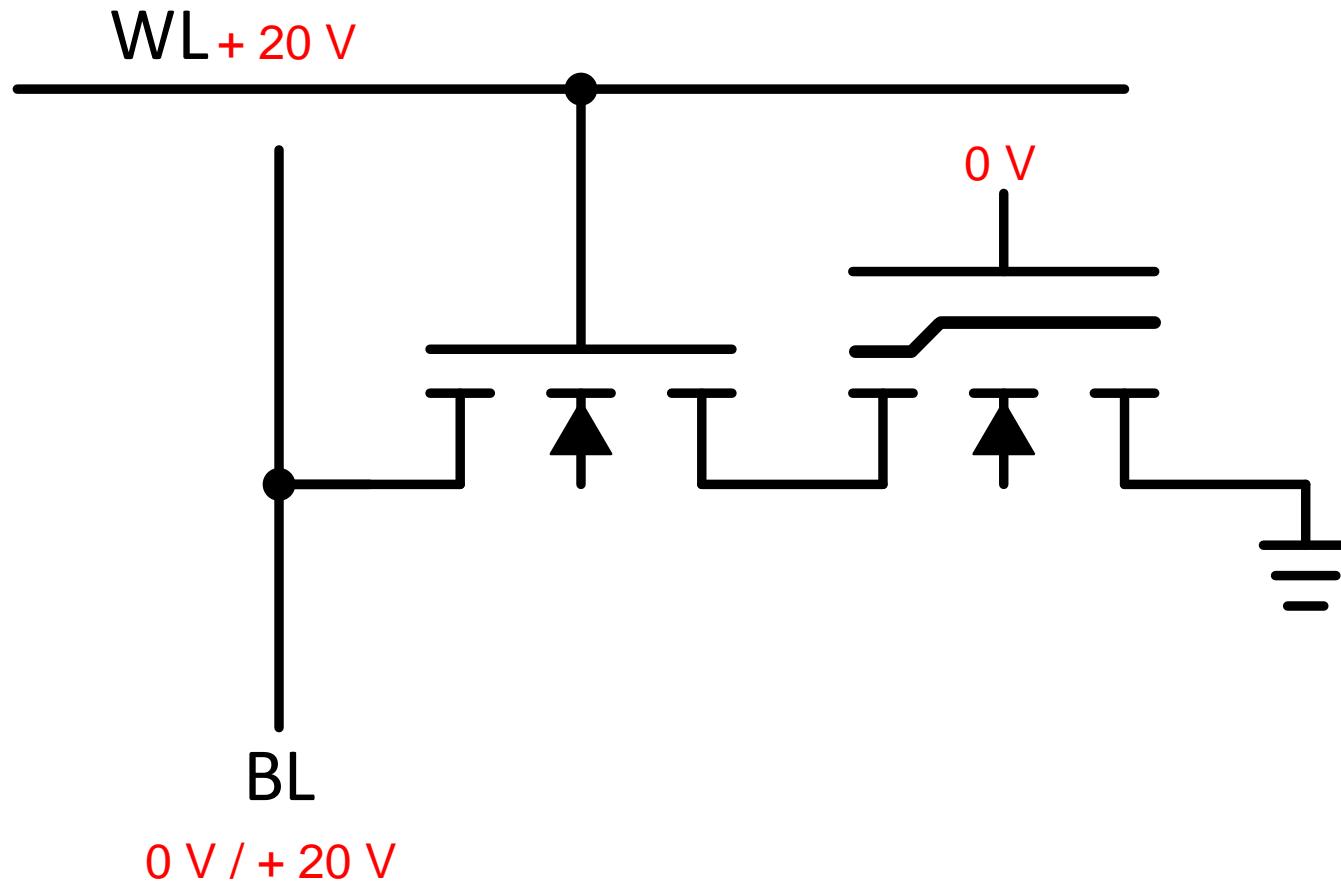


field electron emission
("Fowler–Nordheim tunneling")

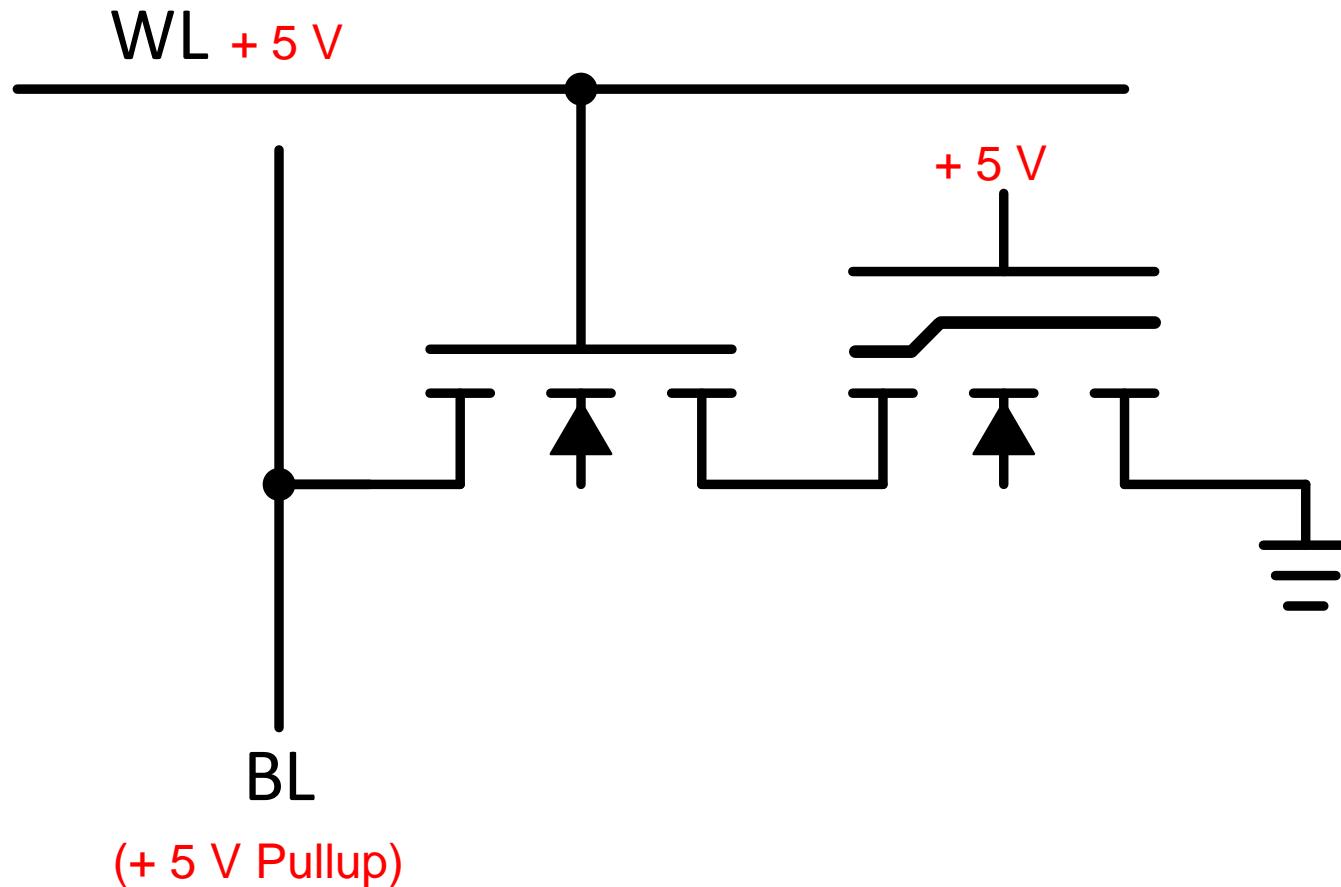
Persistent memory: EEPROM (delete)



Persistent memory: EEPROM (set)



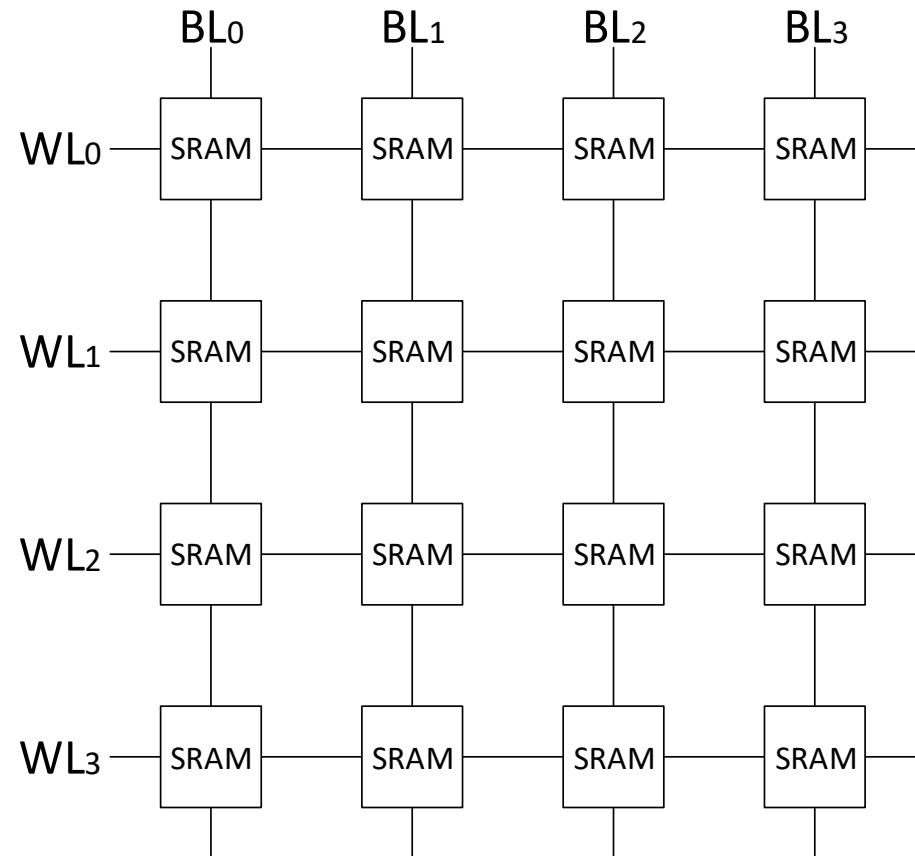
Persistent memory: EEPROM (read)



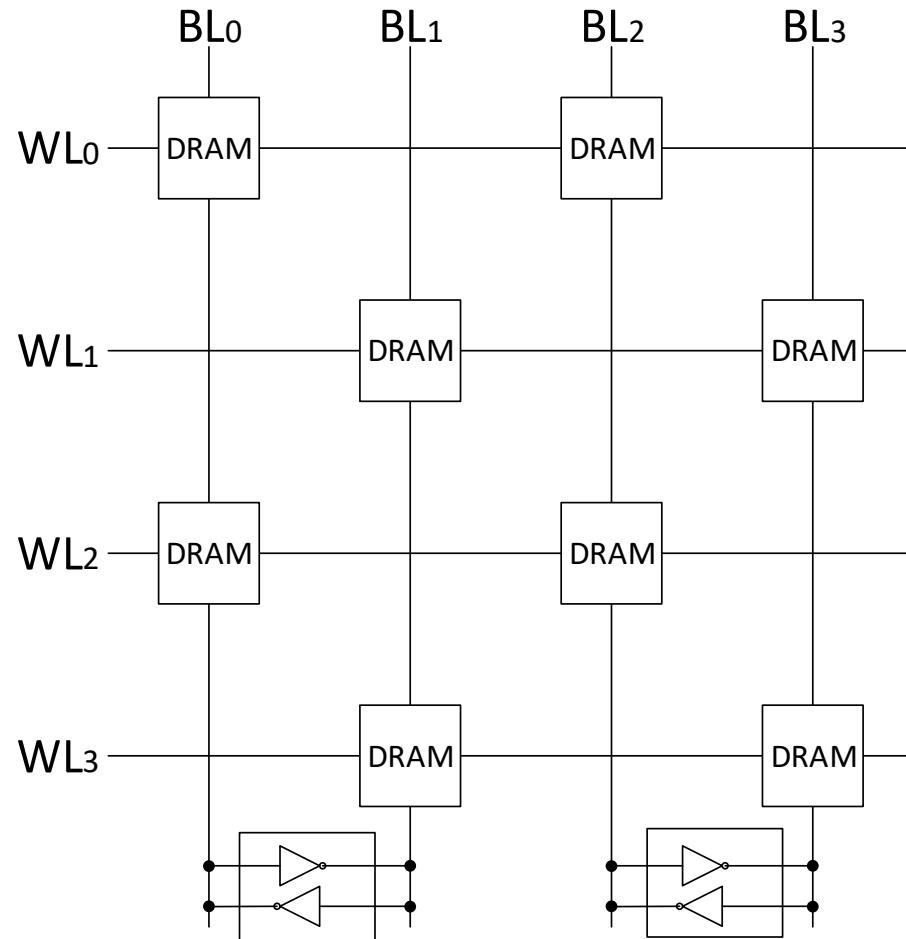
Persistent Storage: Flash Memory

- Special kind of EEPROM
- Block by block write access (compared to - typically - one byte at EEPROM)
- Smaller and cheaper than EEPROM.
- Examples: USB flash drives, SSDs, program memory in microcontrollers

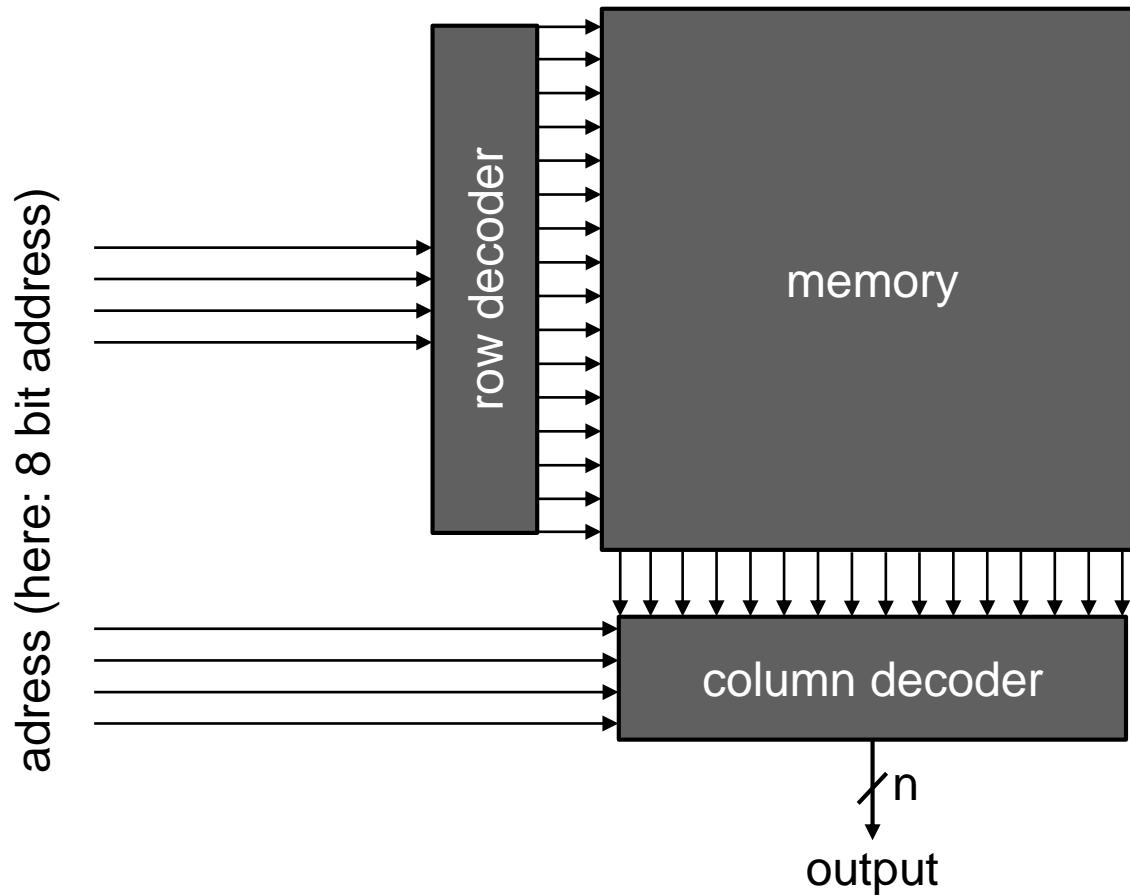
Matrix array of SRAM memory



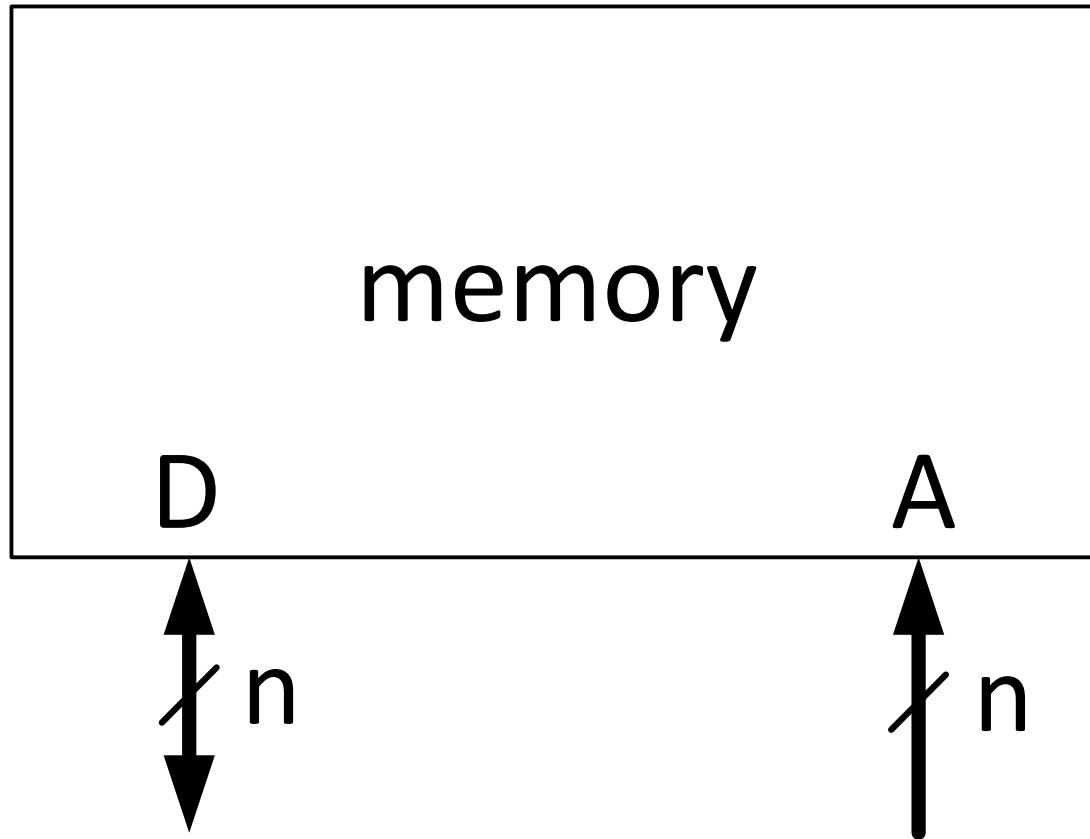
Matrix array of DRAM memory



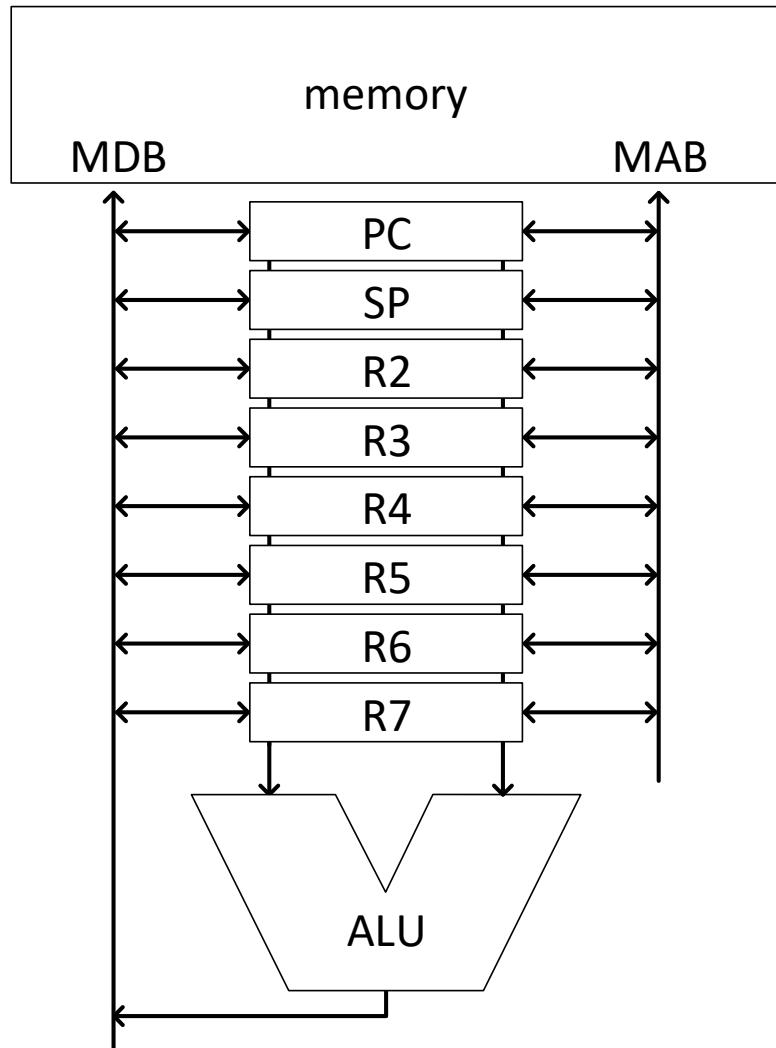
Matrix array of memory



Matrix array of memory

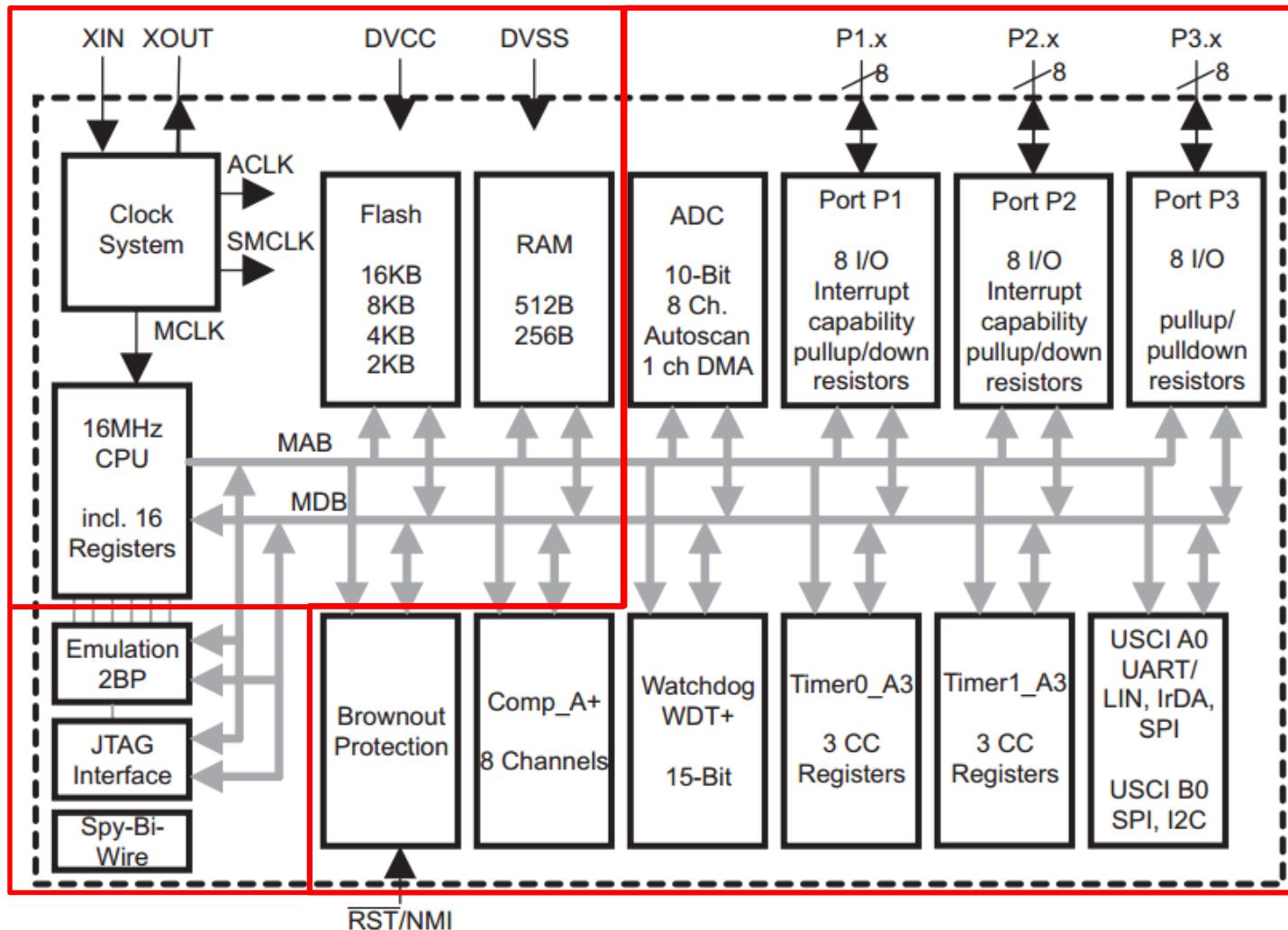


Connection to existing CPU



- LC oscillator
 - Low accuracy
 - High temperature dependence
- Crystal oscillator
 - High precision
 - Slight temperature dependence
- DCO / VCO
 - Medium to high precision
 - High temperature dependence, but partially compensable

Complete microcontroller system



Source: MSP430G2x53, MSP430G2x13 Mixed Signal Microcontroller (Rev. J)

State of the Lecture

- ✓ Introduction
- ✓ Chapter 0: basic knowledge
 - ✓ Number systems, digital logic, ...
- ✓ Chapter 1: calculation and control system
 - ✓ Basic operations in CMOS, registers, memory, concepts, ...
- Chapter 2: Basic peripherals
 - GPIO, ADC, ...
- Chapter 3: Advanced Peripherals
 - Watchdog, programming interfaces, ...

2. Basic peripherals

Connection to peripherals, GPIO, ADC, DAC, Timer

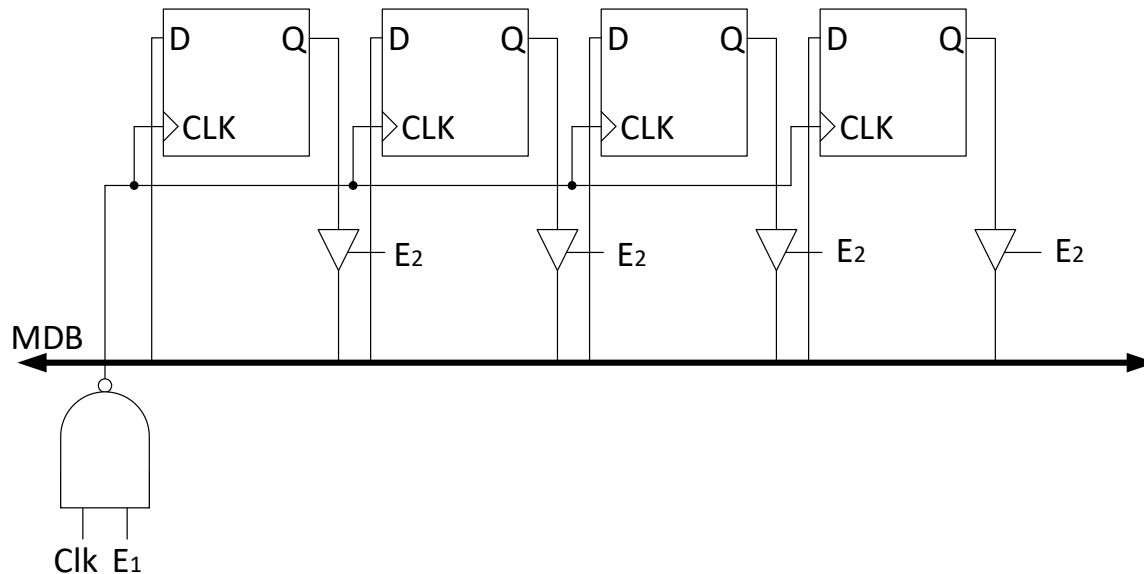
2. Basic peripherals

CONNECTION OF PERIPHERALS TO MAB AND MDB

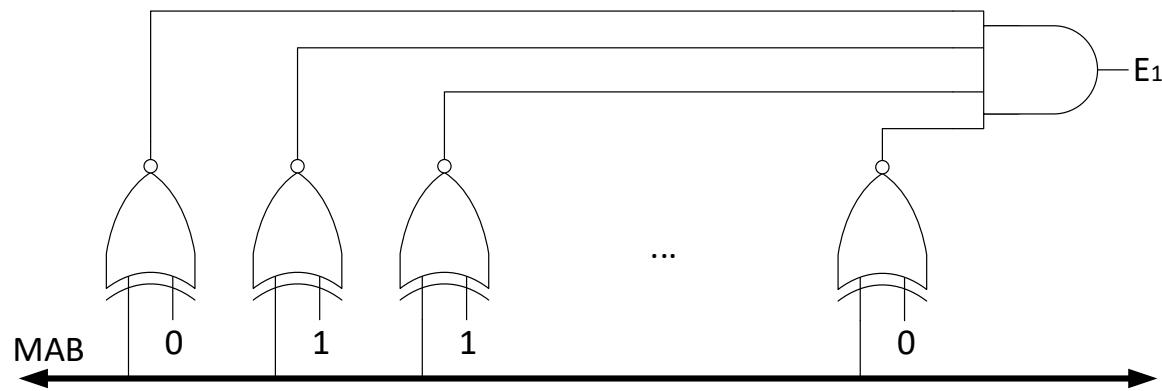
Connection of peripherals to MAB and MDB

- Any peripheral can be accessed by CPU registers
- These registers are connected to the bus system
- Problem: How to realize addressing in order to avoid multiple access?

Connection of peripherals to MAB and MDB

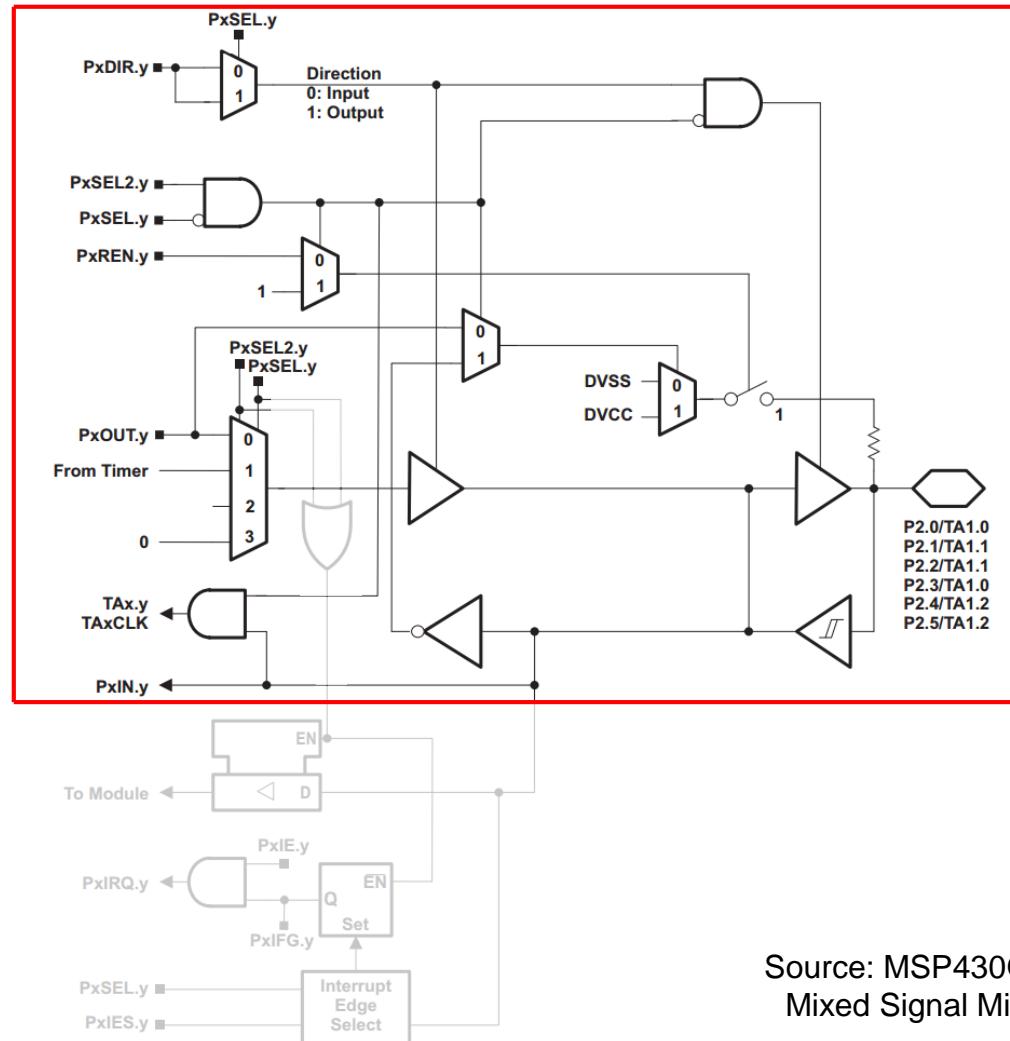


Connection of peripherals to MAB and MDB



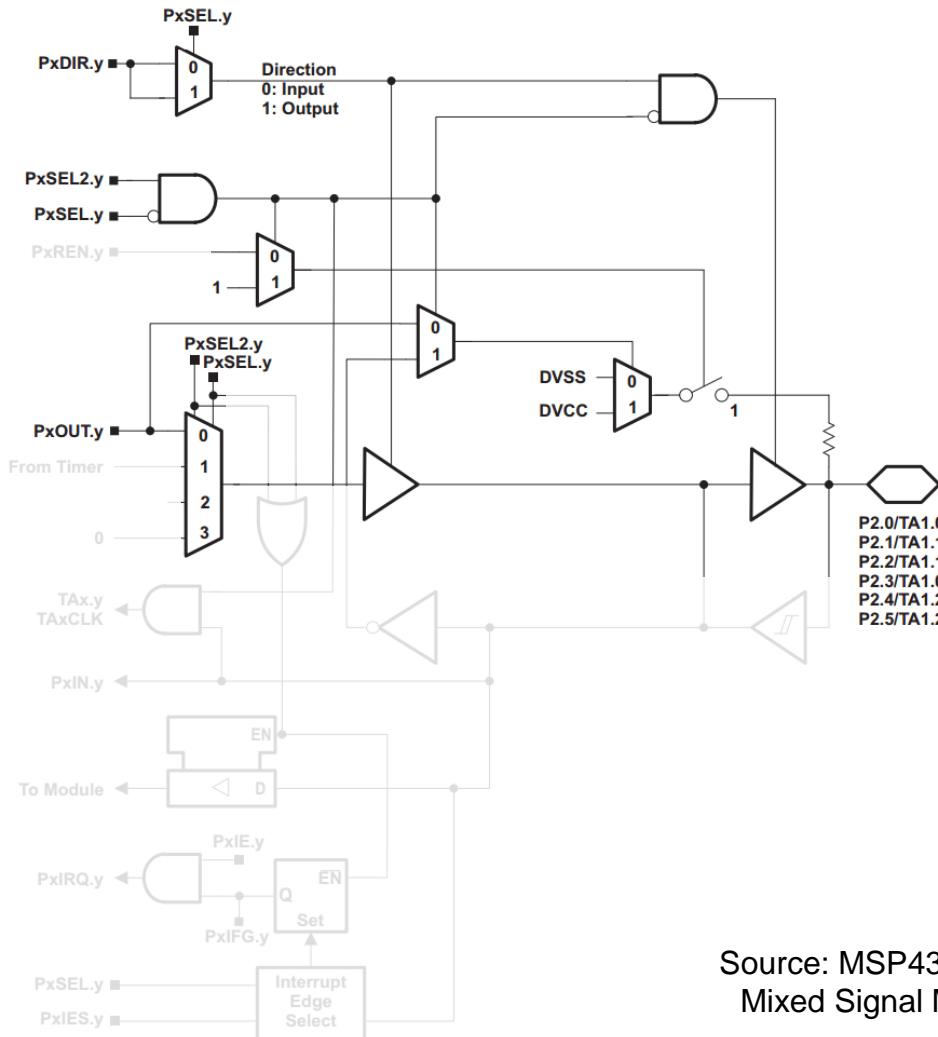
2. Basic peripherals

GPIOs



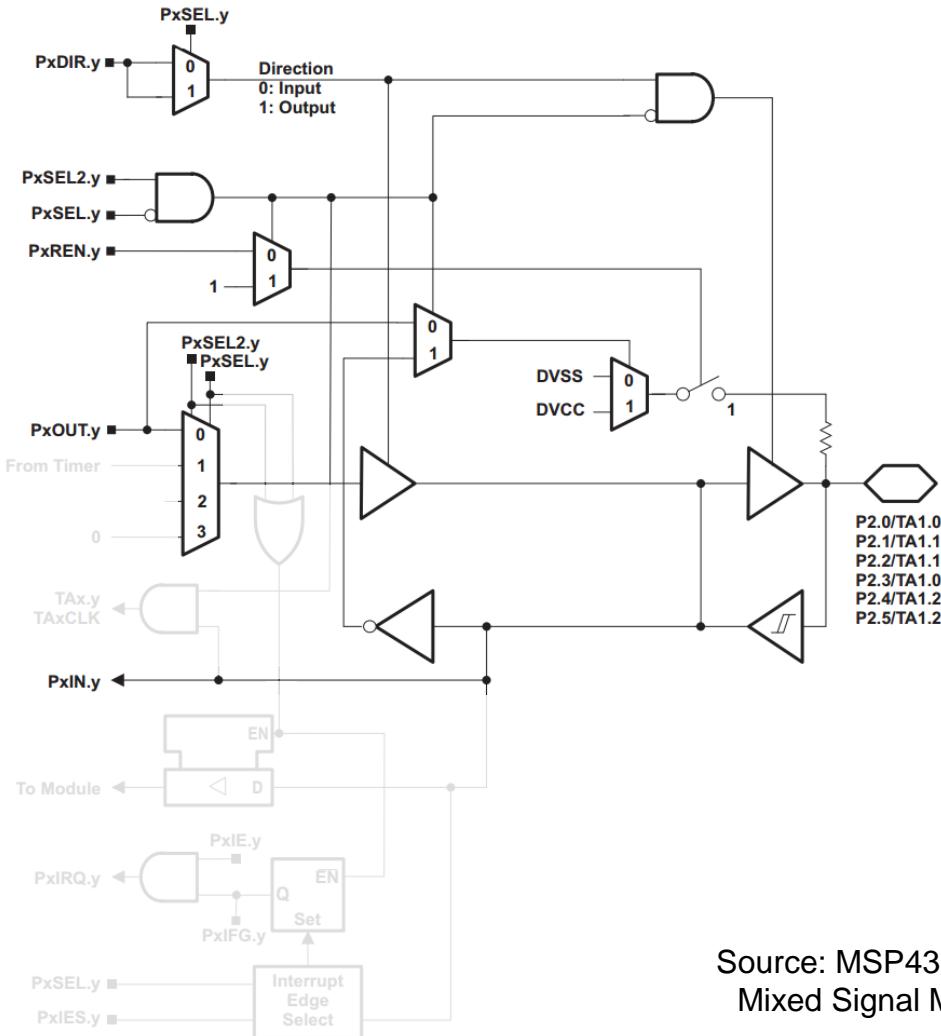
Source: MSP430G2x53, MSP430G2x13
Mixed Signal Microcontroller (Rev. J)

GPIO – Output



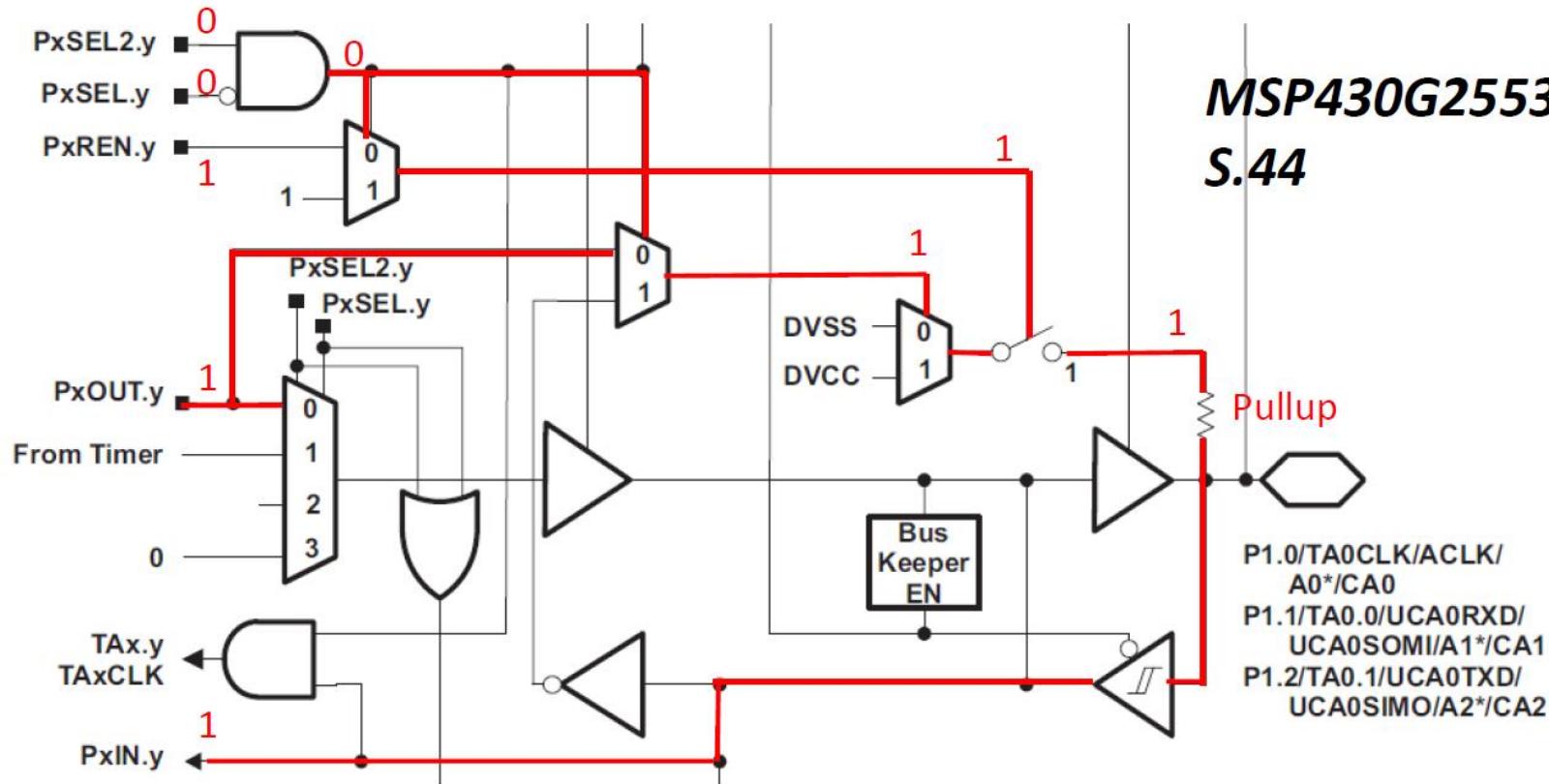
Source: MSP430G2x53, MSP430G2x13
Mixed Signal Microcontroller (Rev. J)

GPIO – Input

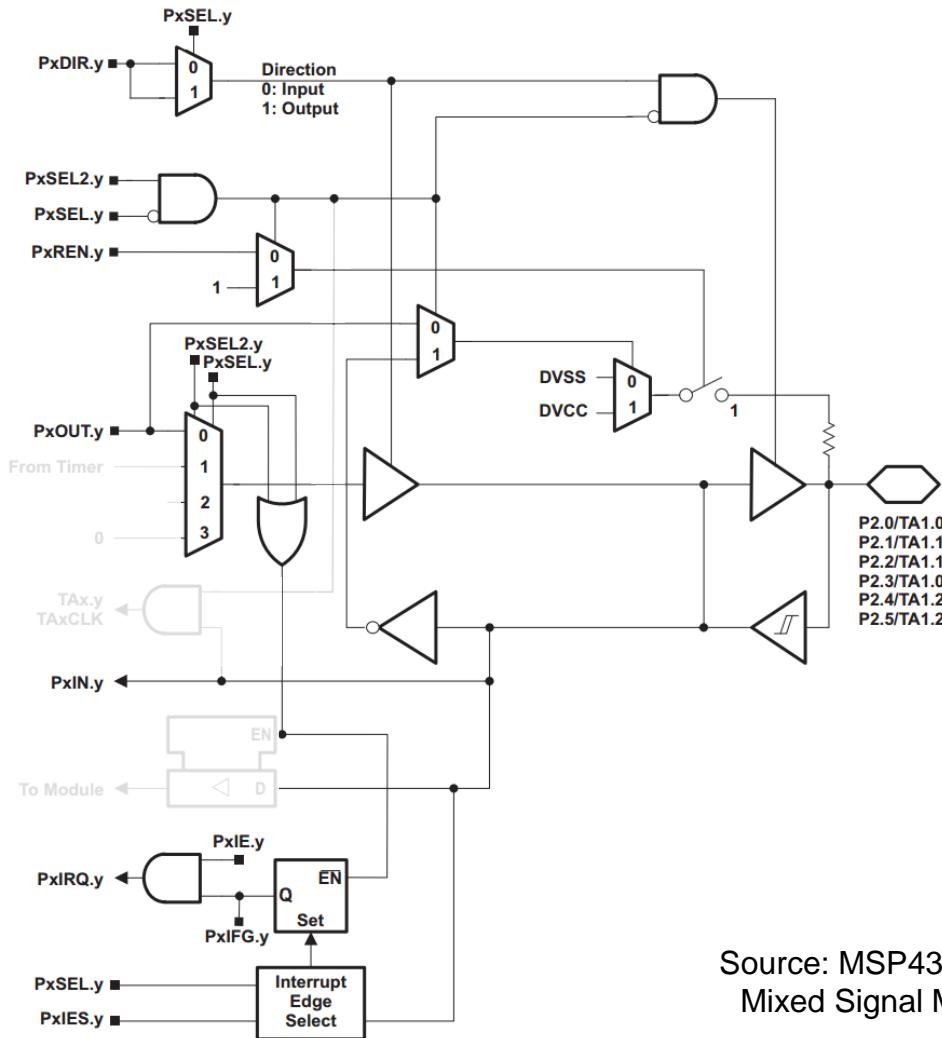


Source: MSP430G2x53, MSP430G2x13
Mixed Signal Microcontroller (Rev. J)

GPIO – I/O Ports Pull up / down resistors

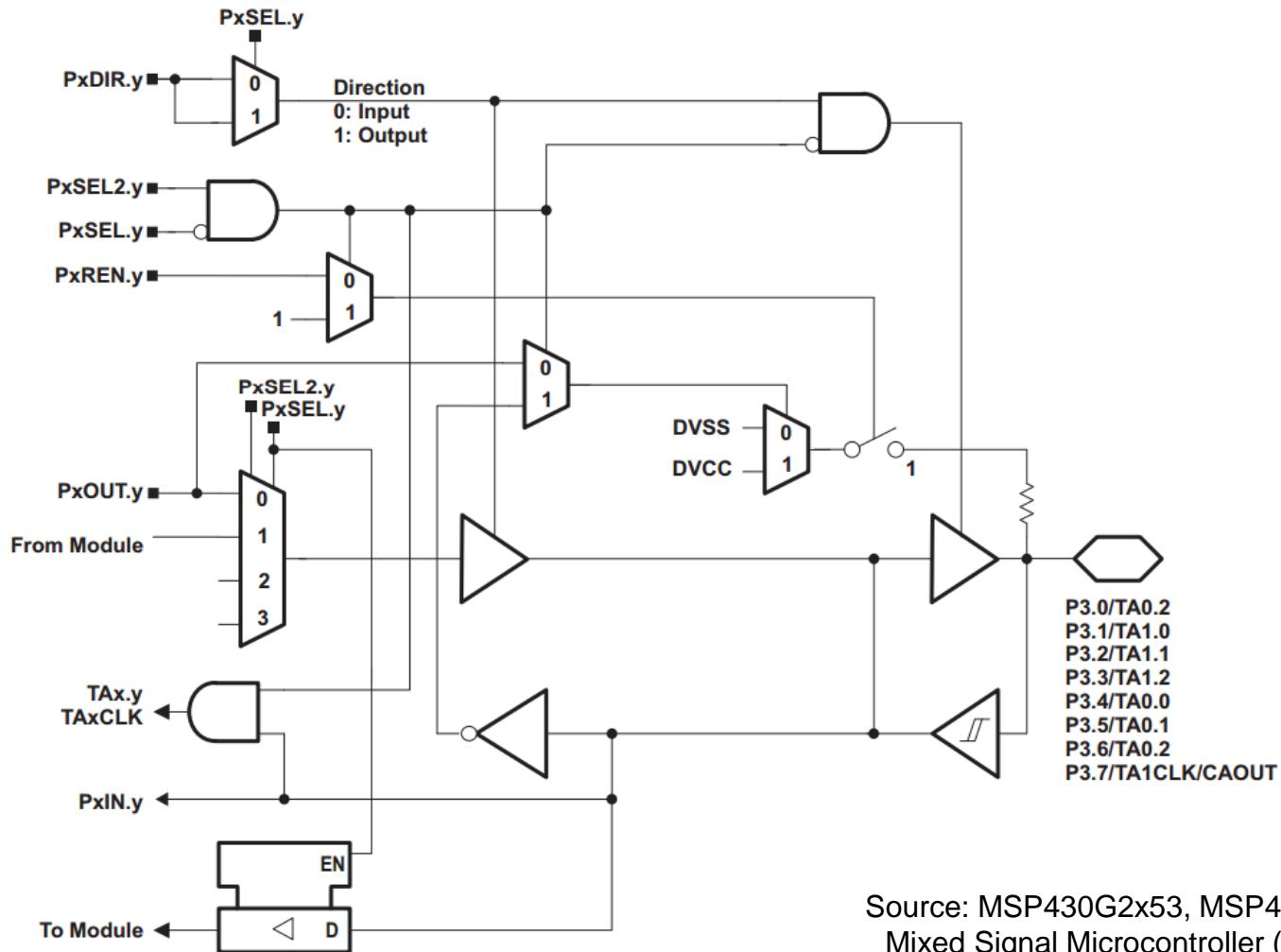


GPIO – Interrupt



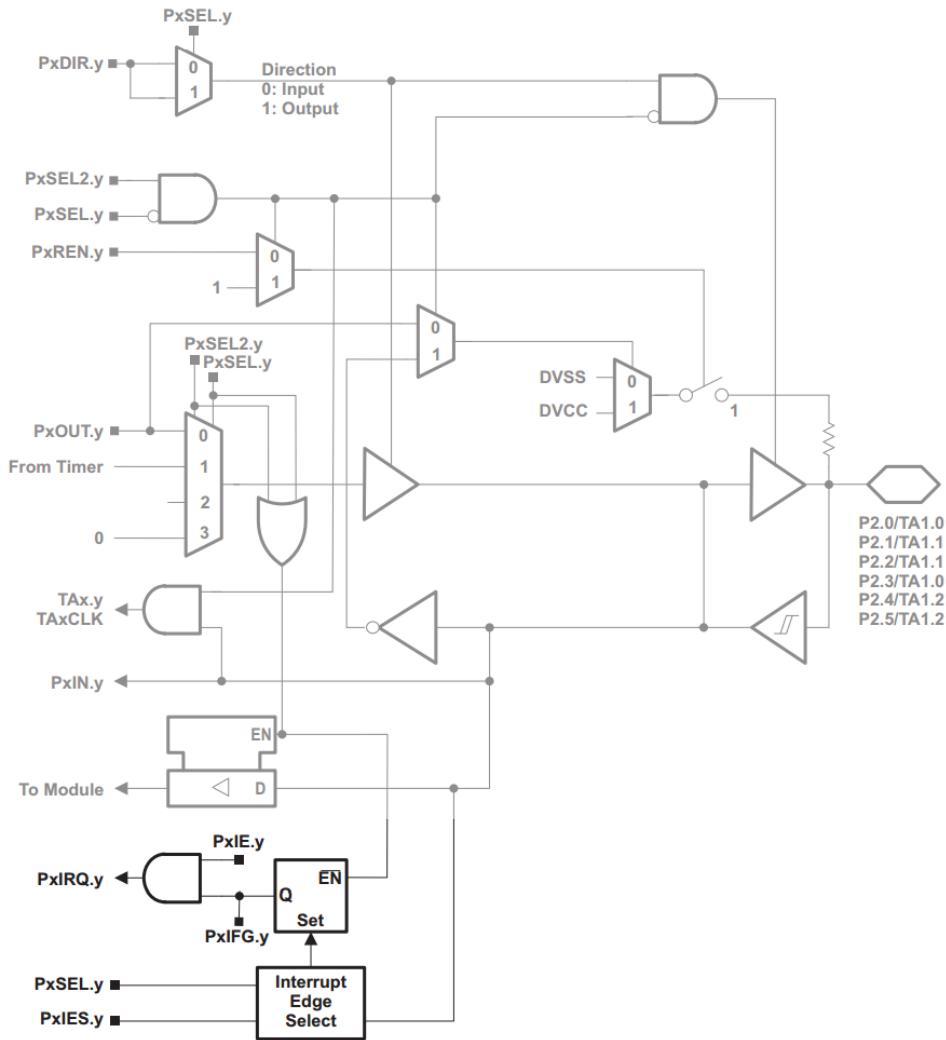
Source: MSP430G2x53, MSP430G2x13
Mixed Signal Microcontroller (Rev. J)

GPIO, here Port 3



Source: MSP430G2x53, MSP430G2x13
Mixed Signal Microcontroller (Rev. J)

Insertion: interrupts on the example of GPIO



Insertion: interrupts on the example of GPIO

- Interrupt is triggered (for example, by edge at the input line)
- Current program address is pushed to the *stack*.
- Jump to interrupt service routine (ISR)
- Execute the code from there
- Get the program address from the stack, return to the “regular” program

- Interrupts are partially maskable, i.e., the programmer can specify whether an interrupt will be triggered upon the occurrence of the associated event.

Insertion: Interrupt priority

Table 5. Interrupt Sources, Flags, and Vectors

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-Up External Reset Watchdog Timer+ Flash key violation PC out-of-range ⁽¹⁾	PORIFG RSTIFG WDTIFG KEYV ⁽²⁾	Reset	0FFEh	31, highest
NMI Oscillator fault Flash memory access violation	NMIIFG OFIFG ACCVIFG ⁽²⁾⁽³⁾	(non)-maskable (non)-maskable (non)-maskable	0FFCCh	30
Timer1_A3	TA1CCR0 CCIFG ⁽⁴⁾	maskable	0FFAh	29
Timer1_A3	TA1CCR2 TA1CCR1 CCIFG, TAIFG ⁽²⁾⁽⁴⁾	maskable	0FFF8h	28
Comparator_A+	CAIFG ⁽⁴⁾	maskable	0FFF6h	27
Watchdog Timer+	WDTIFG	maskable	0FFF4h	26
Timer0_A3	TA0CCR0 CCIFG ⁽⁴⁾	maskable	0FFF2h	25
Timer0_A3	TA0CCR2 TA0CCR1 CCIFG, TAIFG ⁽⁵⁾⁽⁴⁾	maskable	0FFF0h	24
USCI_A0/USCI_B0 receive USCI_B0 I2C status	UCA0RXIFG, UCB0RXIFG ⁽²⁾⁽⁵⁾	maskable	0FFEEh	23
USCI_A0/USCI_B0 transmit USCI_B0 I2C receive/transmit	UCA0TXIFG, UCB0TXIFG ⁽²⁾⁽⁶⁾	maskable	0FFECh	22
ADC10 (MSP430G2x53 only)	ADC10IFG ⁽⁴⁾	maskable	0FFEAh	21
			0FFE8h	20
I/O Port P2 (up to eight flags)	P2IFG.0 to P2IFG.7 ⁽²⁾⁽⁴⁾	maskable	0FFE6h	19
I/O Port P1 (up to eight flags)	P1IFG.0 to P1IFG.7 ⁽²⁾⁽⁴⁾	maskable	0FFE4h	18

Source: TI: MSP430G2x53 MSP430G2x13 – Mixed Signal Microcontroller: Datasheet

2. Basic peripherals

ADC

Analog / digital signals

Analog signals

- The world is analog, i.e. the values are continuous
- Therefore, sensors capture analog data (at least in the first part of the measurement chain)

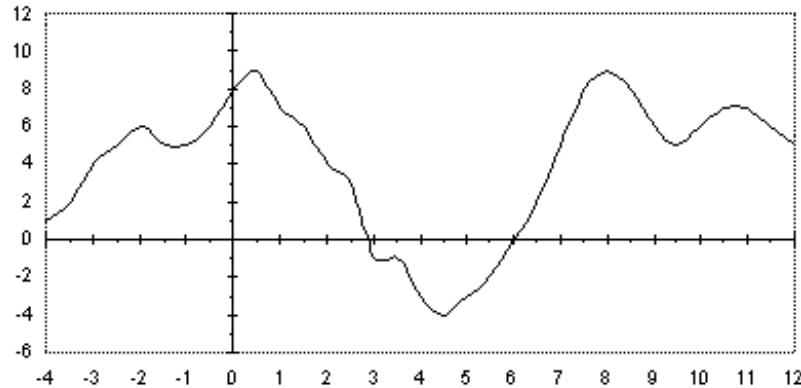
Digital signals

- Discrete values
- Robust to interference (e.g. EMF)
- Simple processing and easy storage

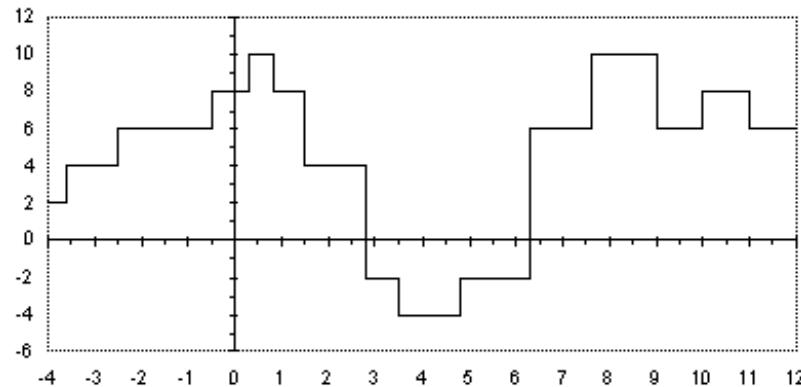
Signals

	time continuous	time discrete
value continuous	analog signal	sampling
value discrete	digitized values	digital signal

Time continuous signals

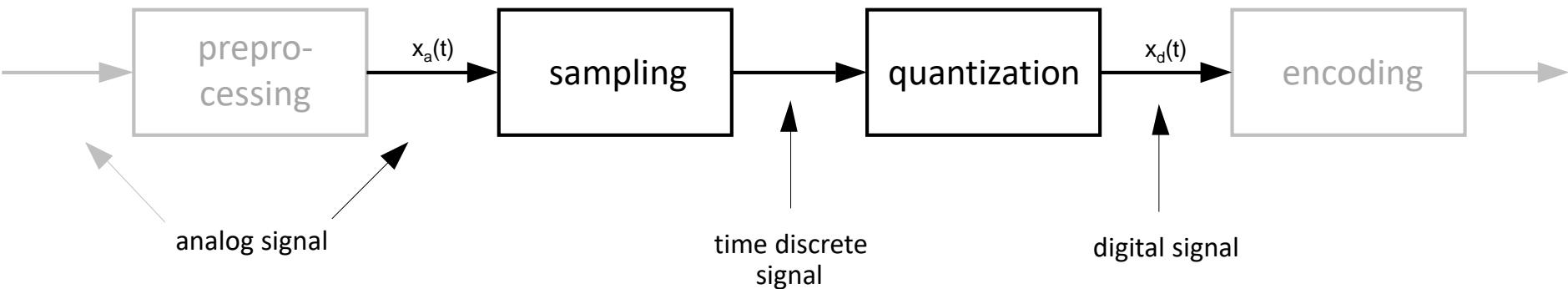


Continuous or analog signal: The information-bearing magnitude is **value- and time-continuous**.



Digitized signal: The information-bearing magnitude is **value discrete and time continuous**.

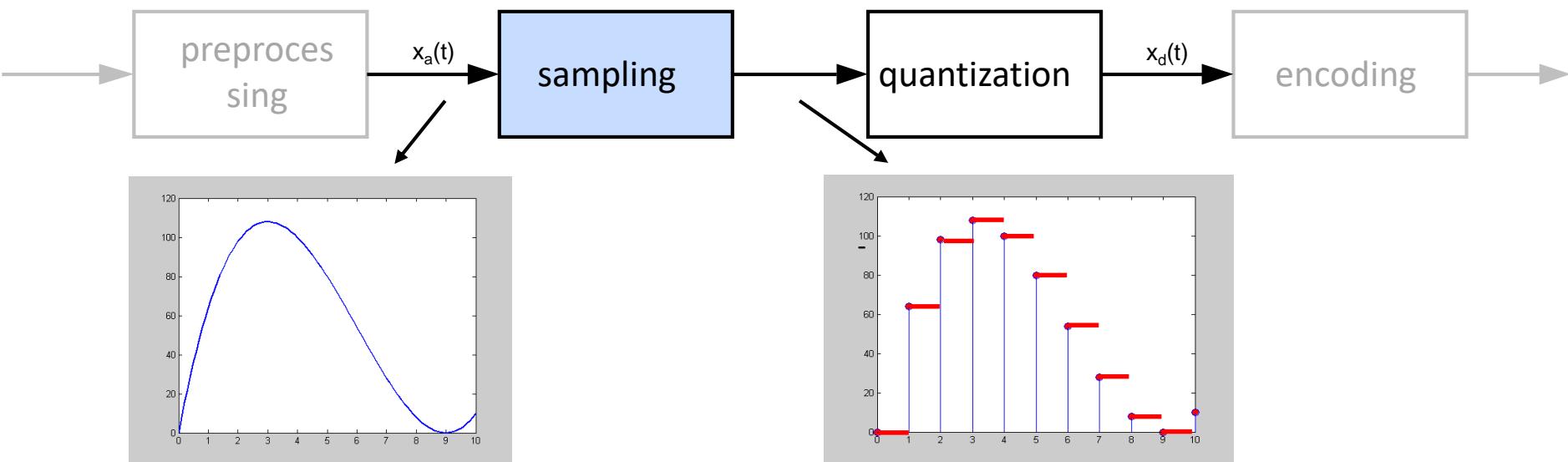
Analog-Digital-Converter(ADC)



4 Steps

1. preprocessing (e.g. anti-aliasing filter) → see lecture „Messtechnik“
2. sampling
3. quantization
4. encoding (Software)

ADC: Sampling (Sample & Hold) I



The sampling converts a time and value continuous signal into a time discrete and value continuous signal.

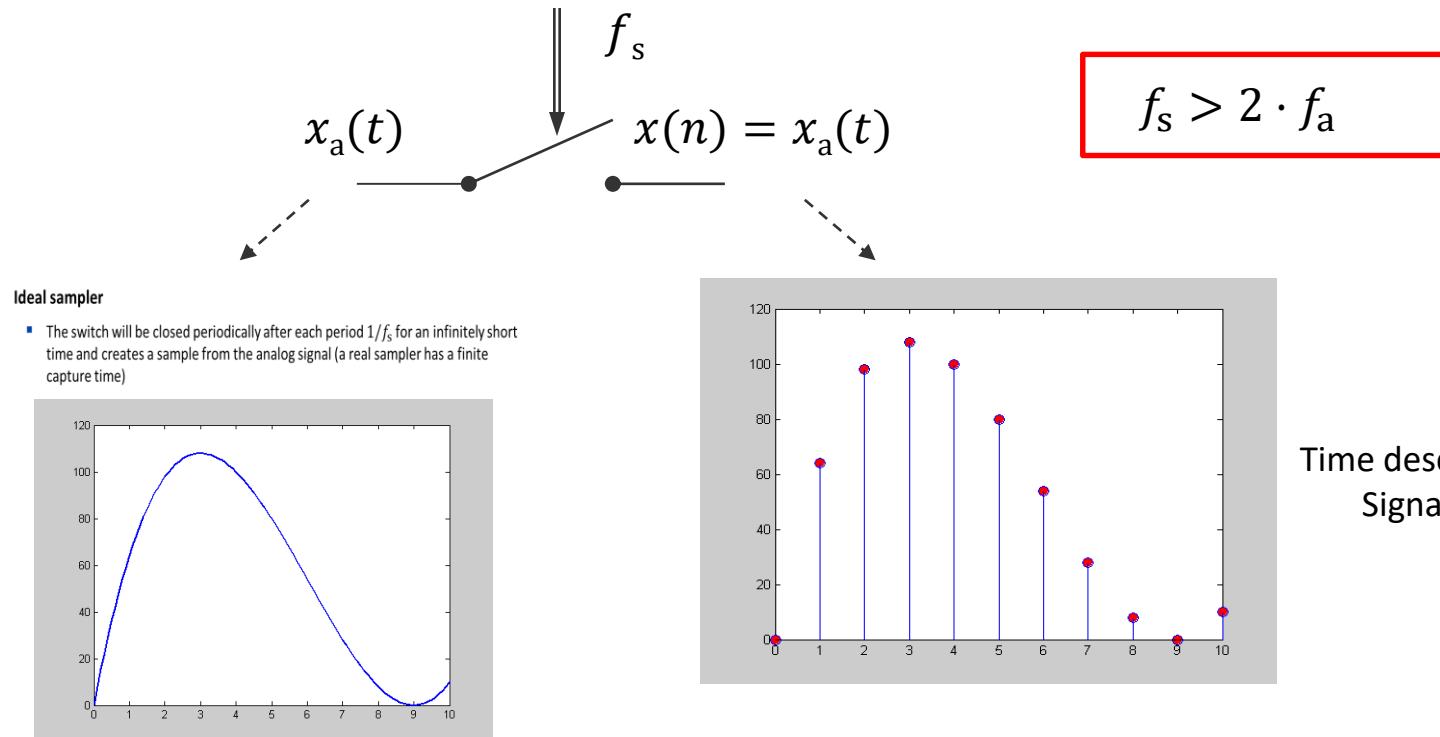
Reason for Sample & Hold:

- The analog-to-digital conversion takes time
- The signal amplitude is advantageously constant during the A/D conversion

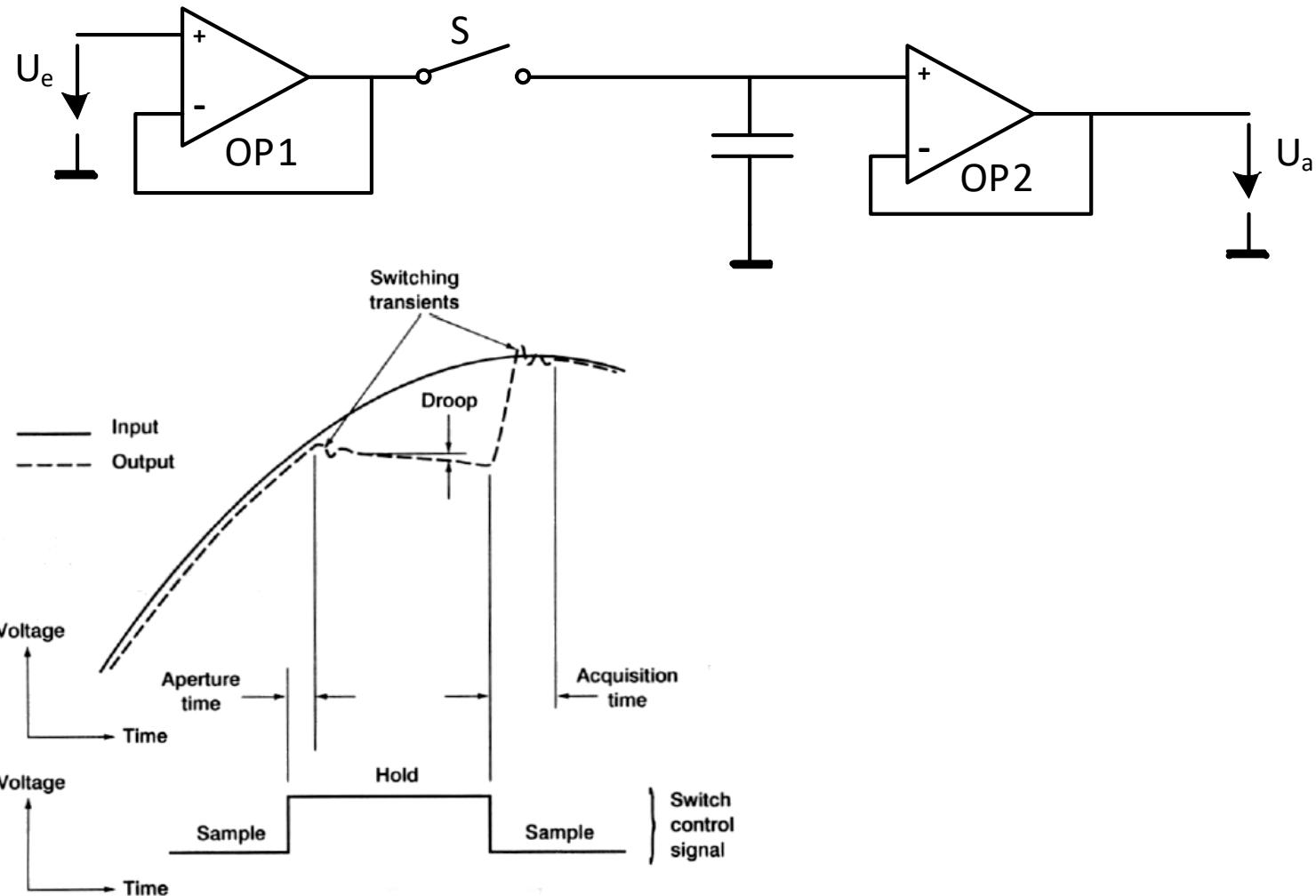
ADC: Sampling (Sample & Hold) II

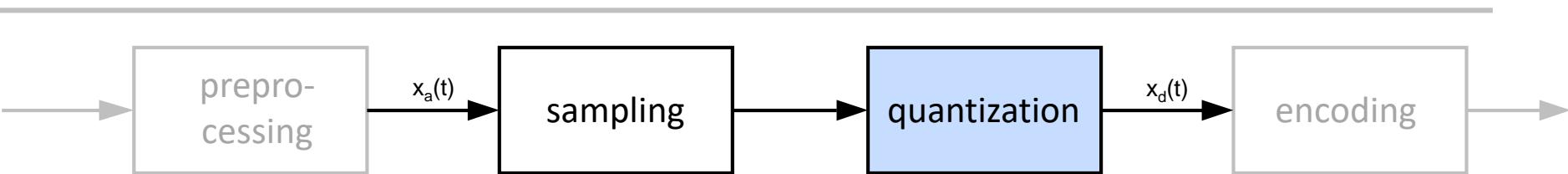
Ideal sampler

- The switch will be closed periodically after each period $1/f_s$ for an infinitely short time and creates a sample from the analog signal (a real sampler has a finite capture time)



ADC: Sampling (Sample & Hold) III

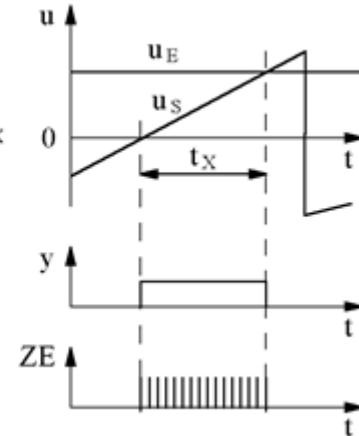
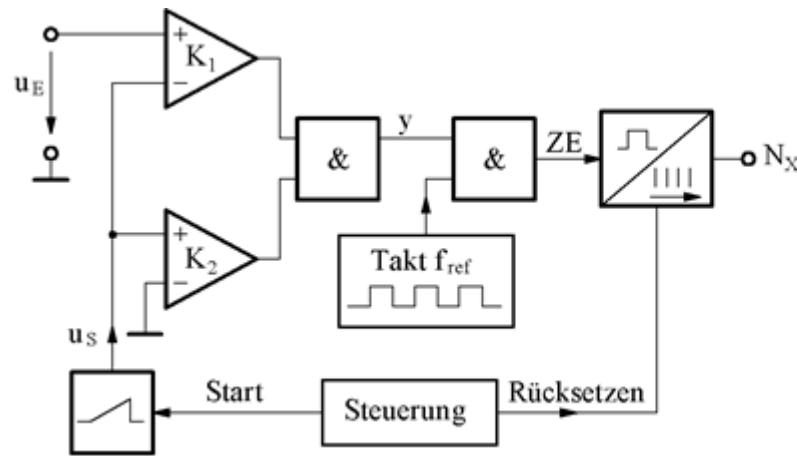




- Implementation of the time-discrete signal into a digital signal $x_d(n)$ (e.g. $x_d(n) = 01001011_2 = 75_{10}$)
- Digital signal can only have values of a fixed, limited range (e.g. with 10 bit resolution: 0 ... 1023)
- A measure of the resolution is the „least significant bit“ (LSB). The value range of U_{max} to U_{min} is divided in a n -bit quantization in $(2^n - 1)$ steps:

$$1 \text{ LSB} = (U_{max} - U_{min}) / (2^n - 1)$$

Single Slope ADC



principle of operation

- After reset, the counter starts counting up the ramp voltage till $U_S > U_E$
- The binary count N_x is proportional to U_e

properties

- relatively slow, sampling may take up to $2^n/f_{ref}$
- ramp must be very linear
(e.g. ramp produced with counter output at DAC)

$u(t)$ Analogsignal
 ↓
 After reset, the counter starts counting up the ramp voltage till $U_S > U_E$
 The binary count N_x is proportional to U_e

properties
 relatively slow, sampling
 ramp must be very linear
 (e.g. ramp produced with counter output at DAC)

time
DAC-value

Dual Slope ADC, functional principle

time t_1

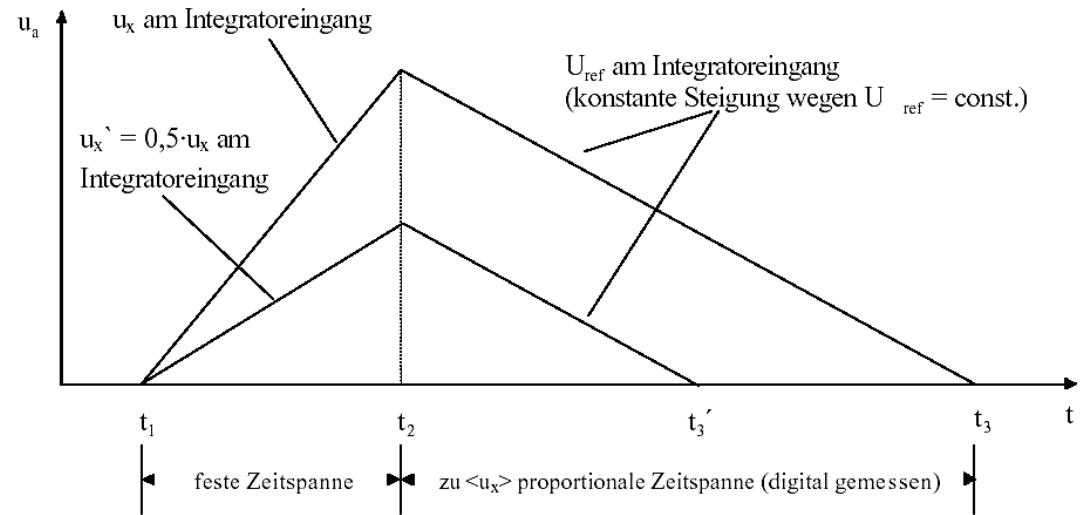
- All counters „reset“ and counter N_1 starts
- The integrator generates a ramp on U_a , whose slope is proportional to U_x

time t_2

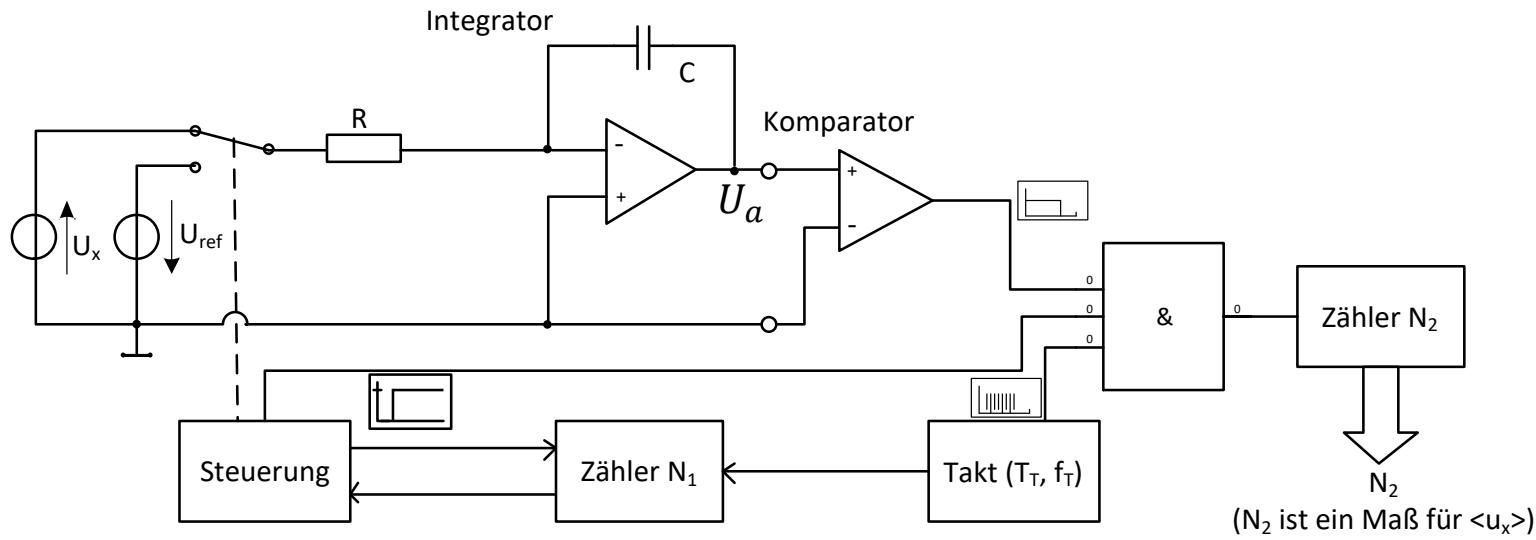
- After counter N_1 reached a defined value, a switch will apply the reference voltage U_{Ref} to the input (U_a is decreased with a defined ramp)
- counter N_2 is started

time t_3

- The counter N_2 counts until the comparator responds, that is $U_a < 0 \text{ V}$
- counter status N_2 is a measure of U_x



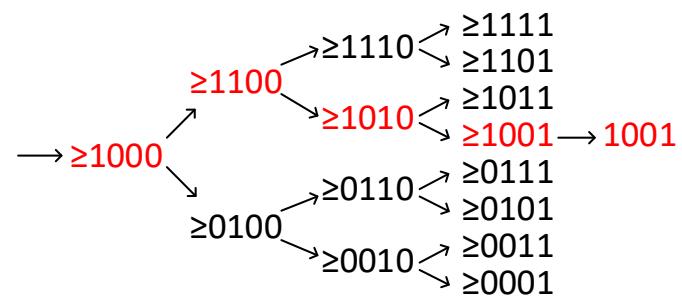
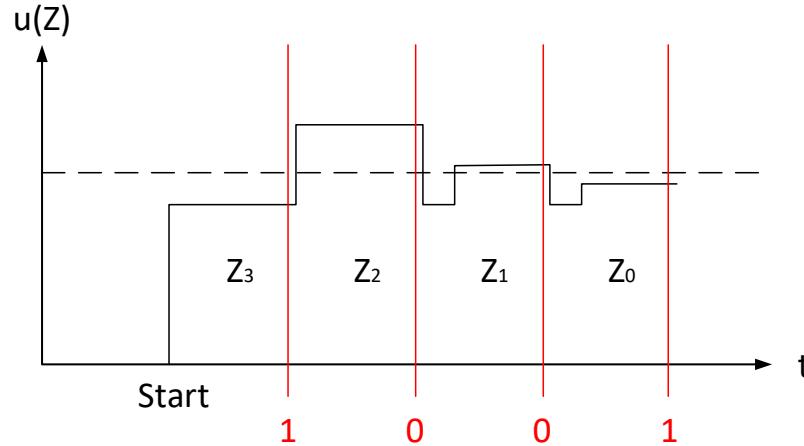
Dual Slope ADC properties



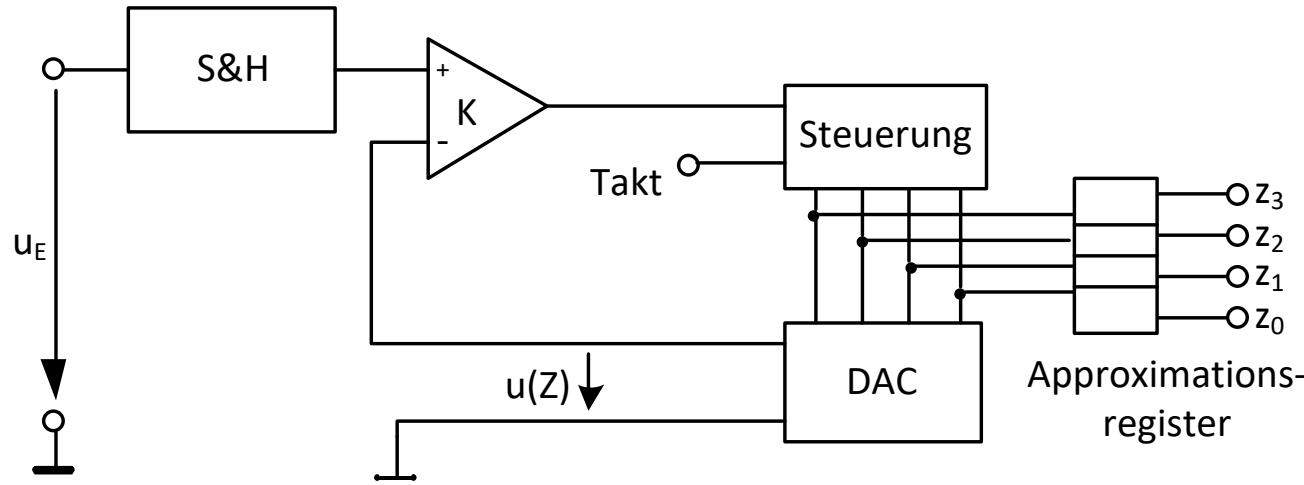
properties

- Very high resolution (used in multimeters)
- Slow (typ. f_s is some 10 Hz)
- Un sensitive to drift (clock, RC, etc.) and to noise (due to integration)

Successive Approximation (SAR), functional principle



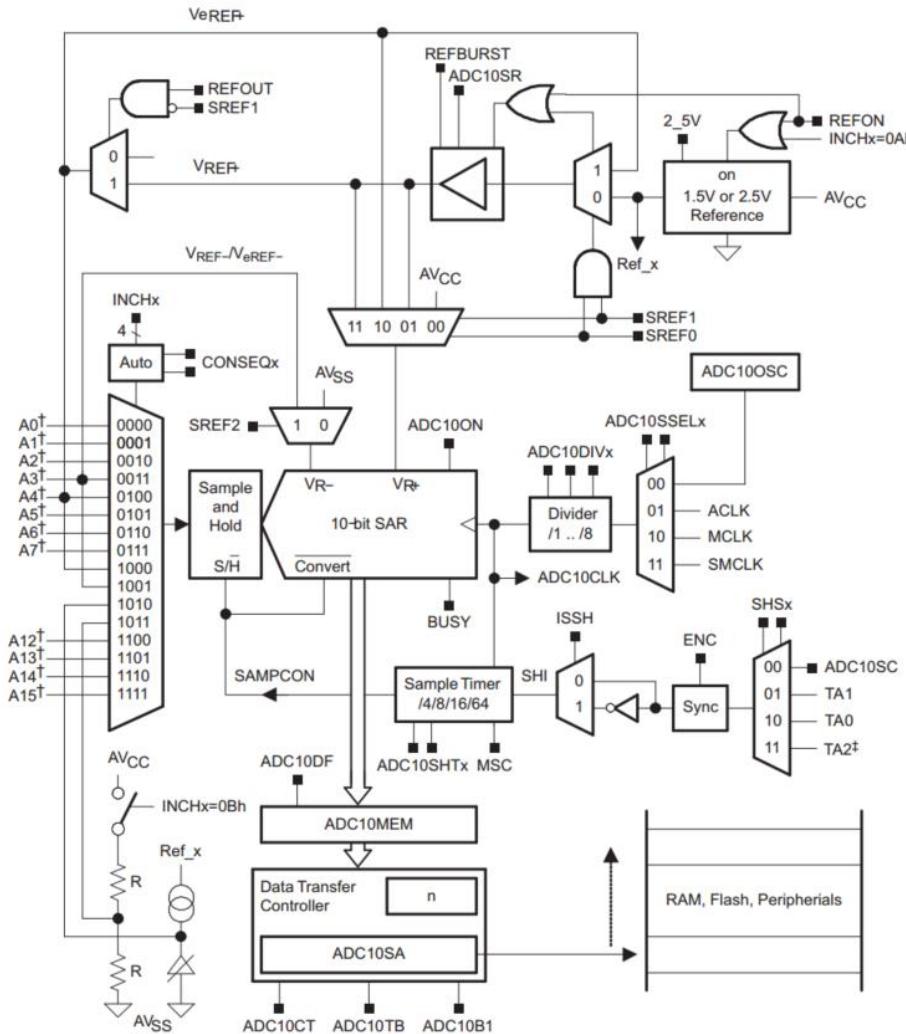
- At the beginning the approximation register provides a value corresponding to $\frac{1}{2} U_{E Max}$ (1000 that is just MSB = 1)
- If $u(Z) > u_E$ set MSB = 0, else MSB = 1
- Then repeat the same procedure bit by bit successively to the LSB
- Similar to a binary search



Properties

- fixed conversion time (n clocks at n -bit ADC), typically in the μs range
- Typically 8 to 16 bit of resolution
- Serial output easily realized because the individual bits are serially generated

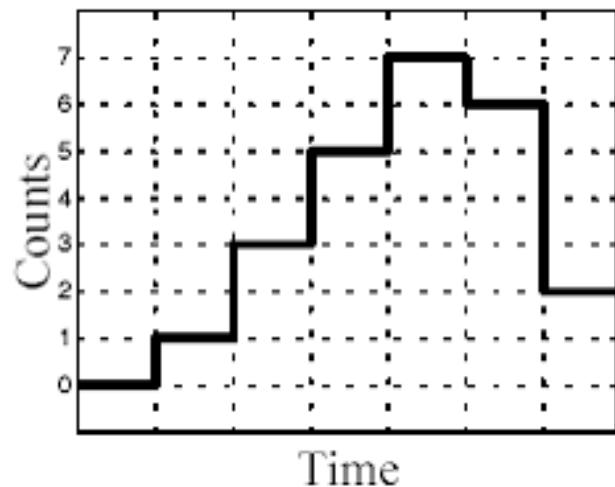
ADC in the MSP430G2553



Source: MSP430G2x53,
MSP430G2x13 Mixed
Signal Microcontroller
(Rev. J)

Encoding is the assignment of a (encoded) value for each discrete value

- at the same time: assignment of a corresponding unit



codes

0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

μg

0
5
10
15
20
25
30
35

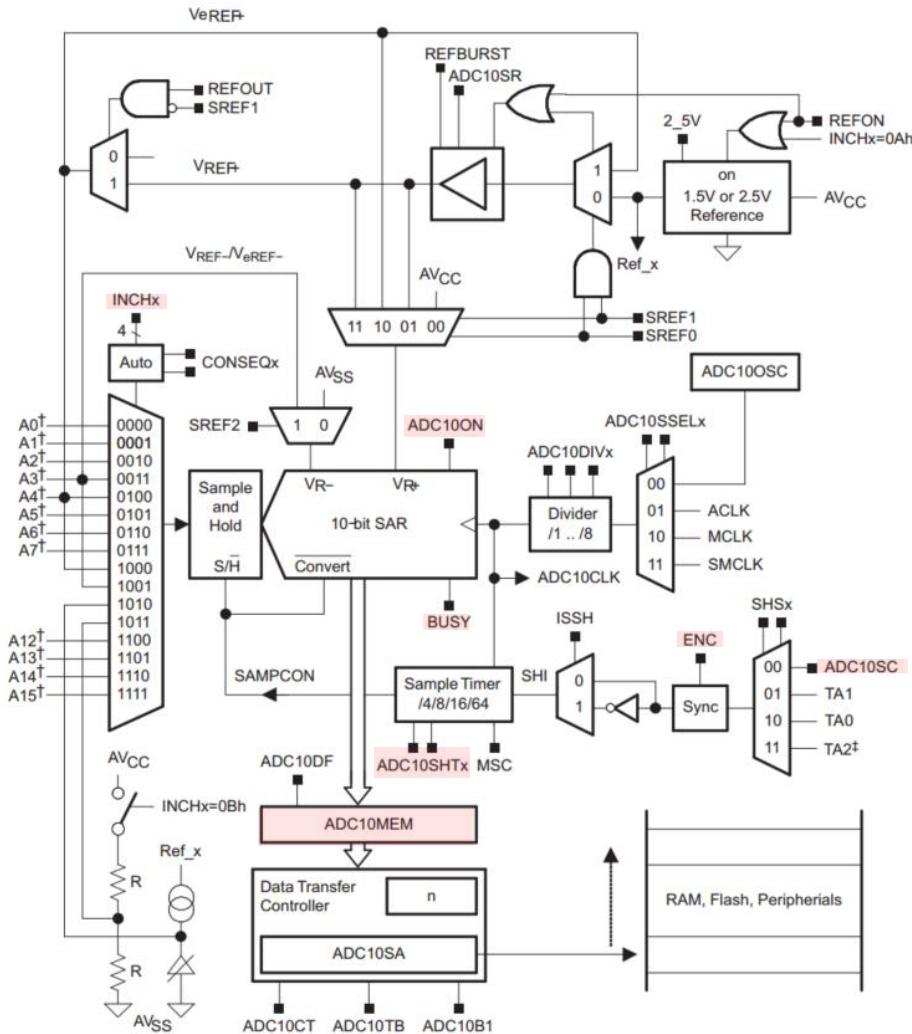
Comparators

-
- In µCs: ADC + register for storing a value which is set by the programmer
 - Action is triggered when reaching / exceeding / falling below the value
 - Usually: Interrupt is generated

ADC: MSP430, C-Example

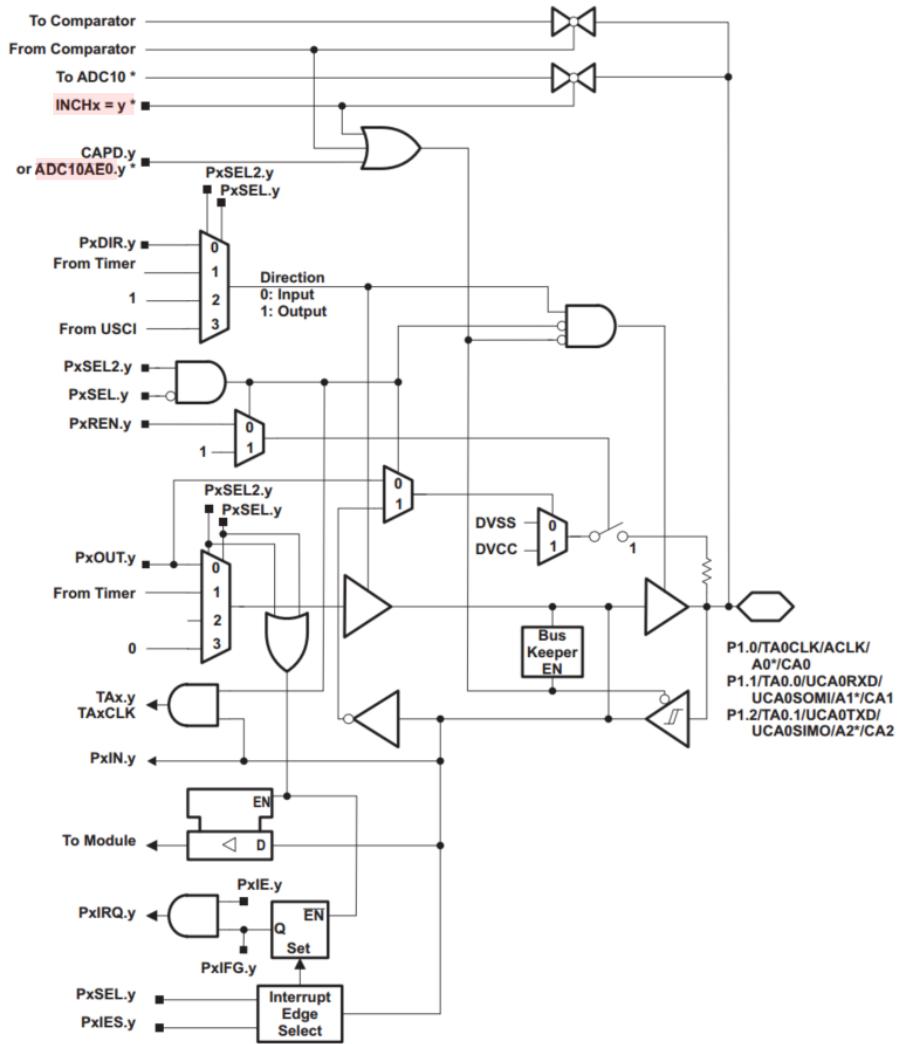
```
ADC10CTL0 = ADC10ON + ADC10SHT_2; // Power ADC on, use 16 clock  
// cycles as sample & hold time  
  
ADC10AE0 |= BIT7; // Enable P1.7 as AD-input  
  
ADC10CTL1 = INCH_7; // Select channel 7 as input  
  
ADC10CTL0 |= ENC + ADC10SC; // Start conversion  
  
while (ADC10CTL1 & ADC10BUSY); // Wait until result is ready  
  
int m1 = ADC10MEM; // When the result is ready, copy to m1
```

ADC in the MSP430G2553



Source: MSP430G2x53,
MSP430G2x13 Mixed
Signal Microcontroller
(Rev. J)

ADC in the MSP430G2553

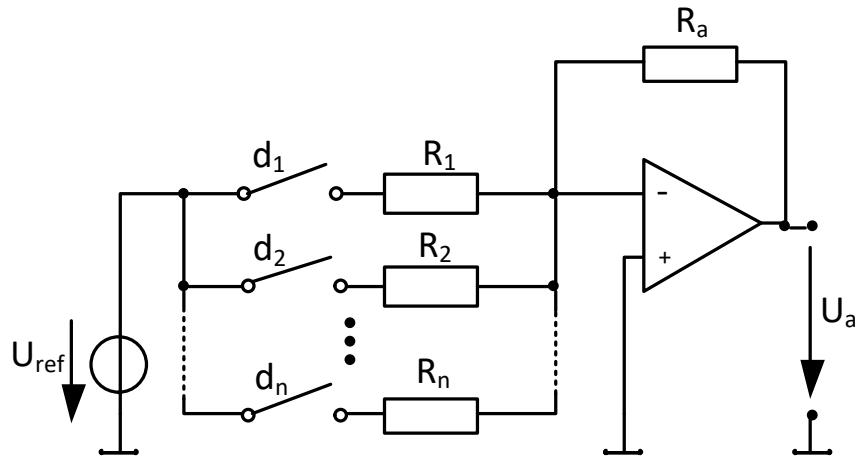


Source: MSP430G2x53,
MSP430G2x13 Mixed
Signal Microcontroller
(Rev. J)

2. Basic peripherals

DACs

Binary Weighted Ladder DAC



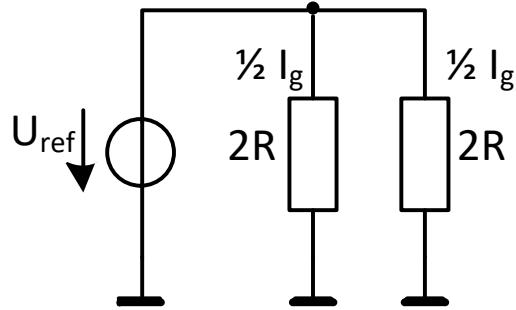
$$R_n = \frac{1}{2} R_{n-1} \quad R_a = R_1$$

$$U_a = -U_{ref} \cdot \sum_{i=1}^n \frac{2 \cdot d_i - 1}{2^i}$$

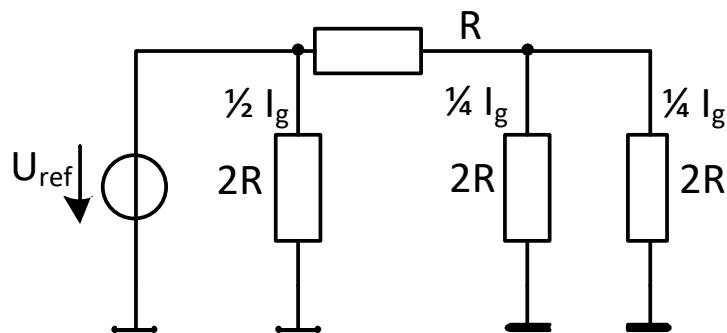
Challenges:

- The internal resistance of the network varies depending on the switch position
 - U_a deviates from the ideal value
- Resistors with very high precision and the same characteristics (for example, temperature coefficient) are required, ranging over several decades of resistance (difficult / impossible to implement in integrated circuits)

Operating principle of the R-2R DAC I

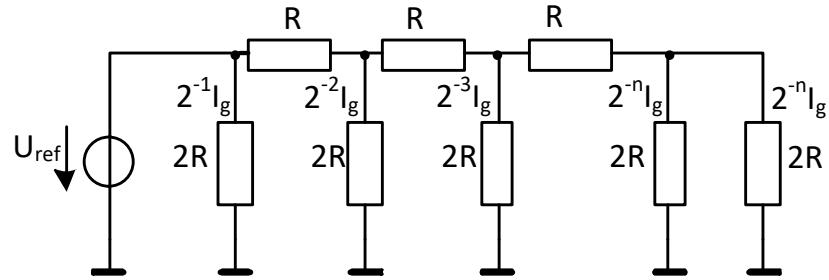


Simple current divider: In each of the circuits flows $I_g/2$

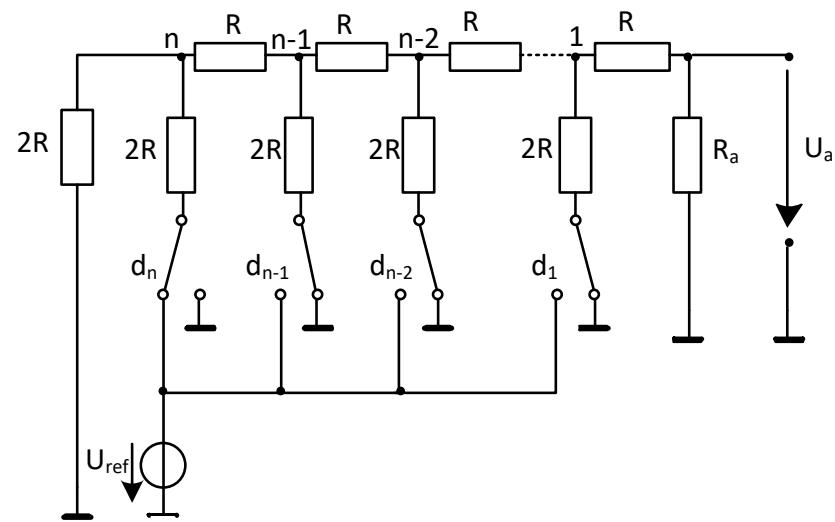


Next Mesh: no changes in first branch; second mainbranch has same overall resistance and features another division of the current to $I_g/4$

Operating principle of the R-2R DAC II

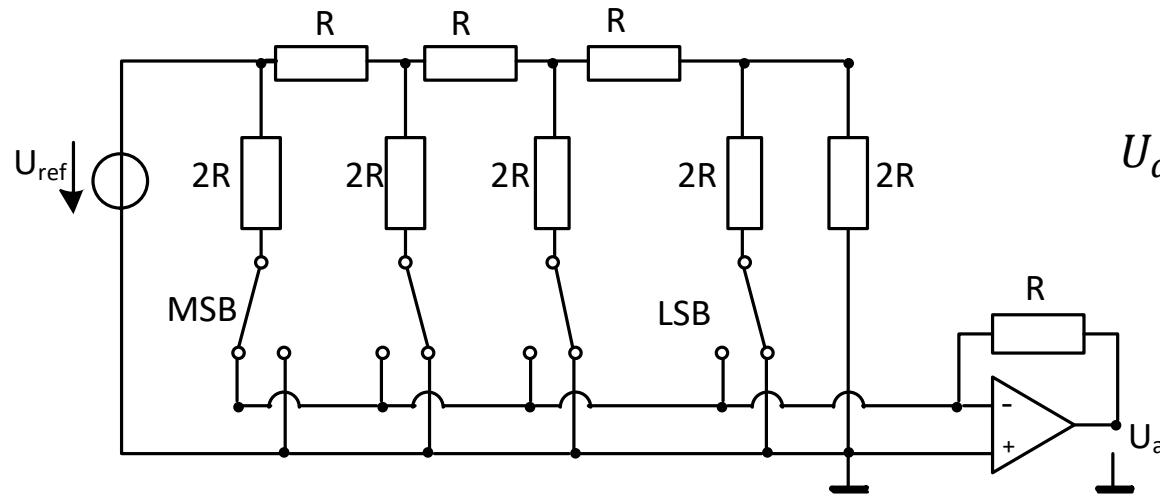


General R-2R current divider:
in the n th branch, there is a current flow of $2^{-n} \cdot I_g$



current-oriented architecture of the
complete R-2R DAC

R-2R Ladder DAC



$$U_a = U_{ref} \cdot \frac{R_a}{R_a + R} \cdot \sum_{i=1}^n \frac{d_i}{2^i}$$

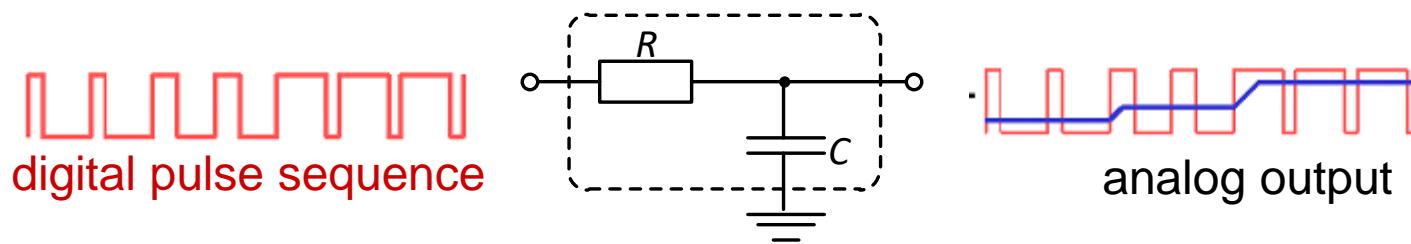
Advantages

- Only 2 types of resistors in the same range of values (feasible for integrated circuit implementation)
- Internal resistance of the network is independent from the switch position.
- Dependence of U_a on the load R_a can be removed by a buffer amplifier

DAC using PWM (pulse width modulation)

basic components

- Digital line with RC lowpass filter



operation principle

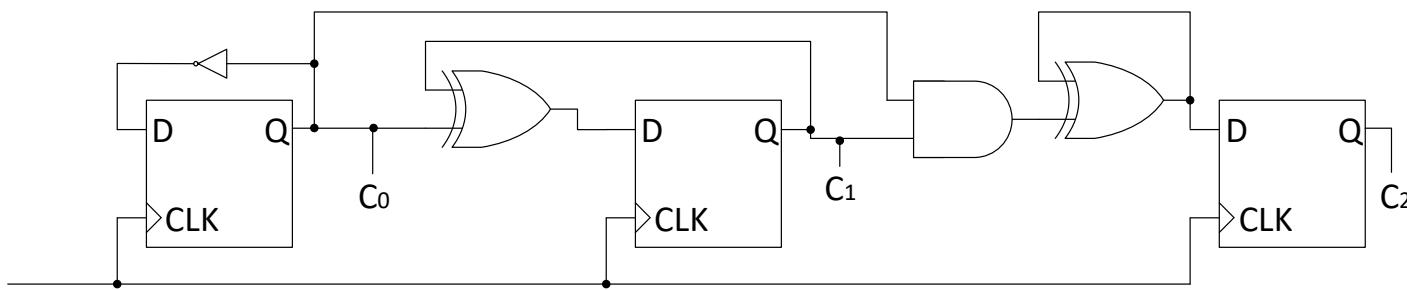
- On the digital line, a pulse sequence is generated at a fixed frequency
- The pulse width is selected to be proportional to the desired output voltage
- The generated digital pulse sequence is low pass filtered
- The analog output voltage of the low pass is proportional to the duty cycle

2. Basic peripherals

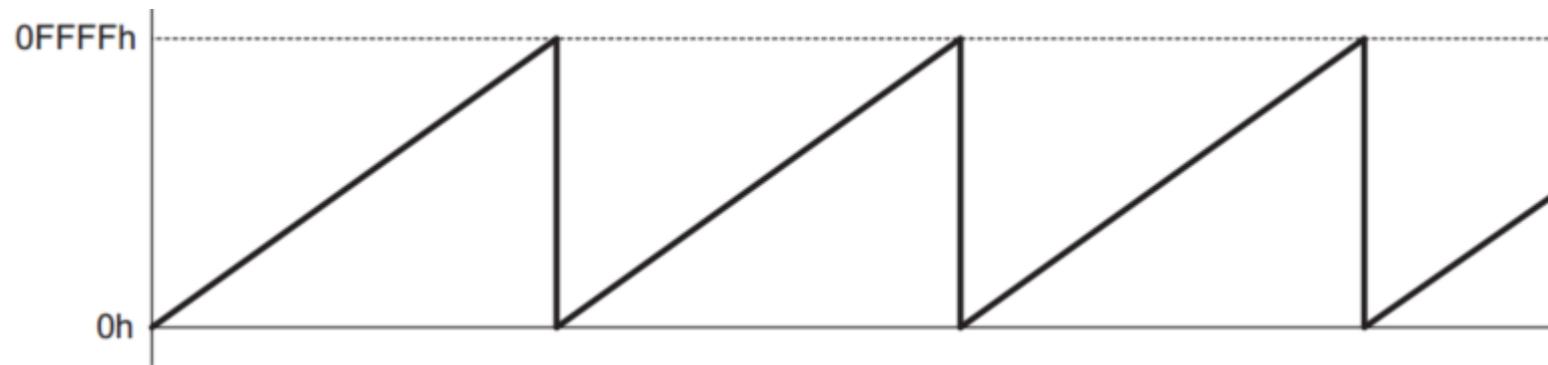
TIMER

- (Binary) counter
- Counts +1 per clock in the simplest case
- MSP430: 16 bit counters

$C_2 C_1 C_0$
0 0 0
0 0 1
0 1 0
...
1 1 0
1 1 1
0 0 0



Timer



Source: MSP430G2x53, MSP430G2x13 Mixed Signal Microcontroller (Rev. J)

Timer, CCR0/1/2

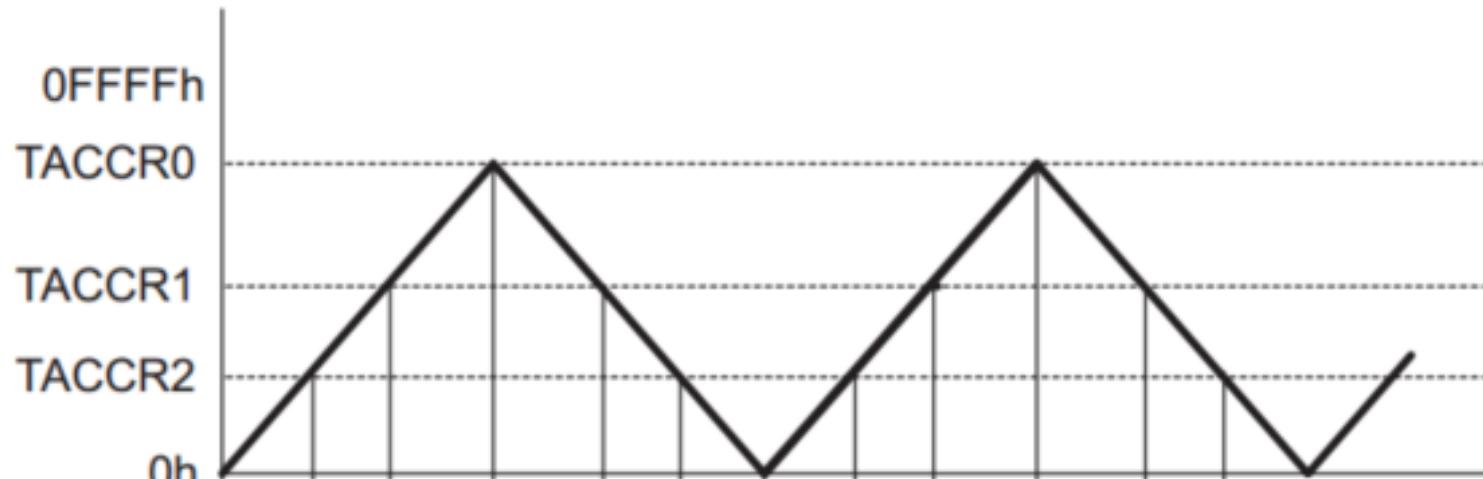
- By adding further registers, "target values" can be stored
- Upon reaching this value, actions (for example, IRQ) can be triggered

Timer, Up Mode



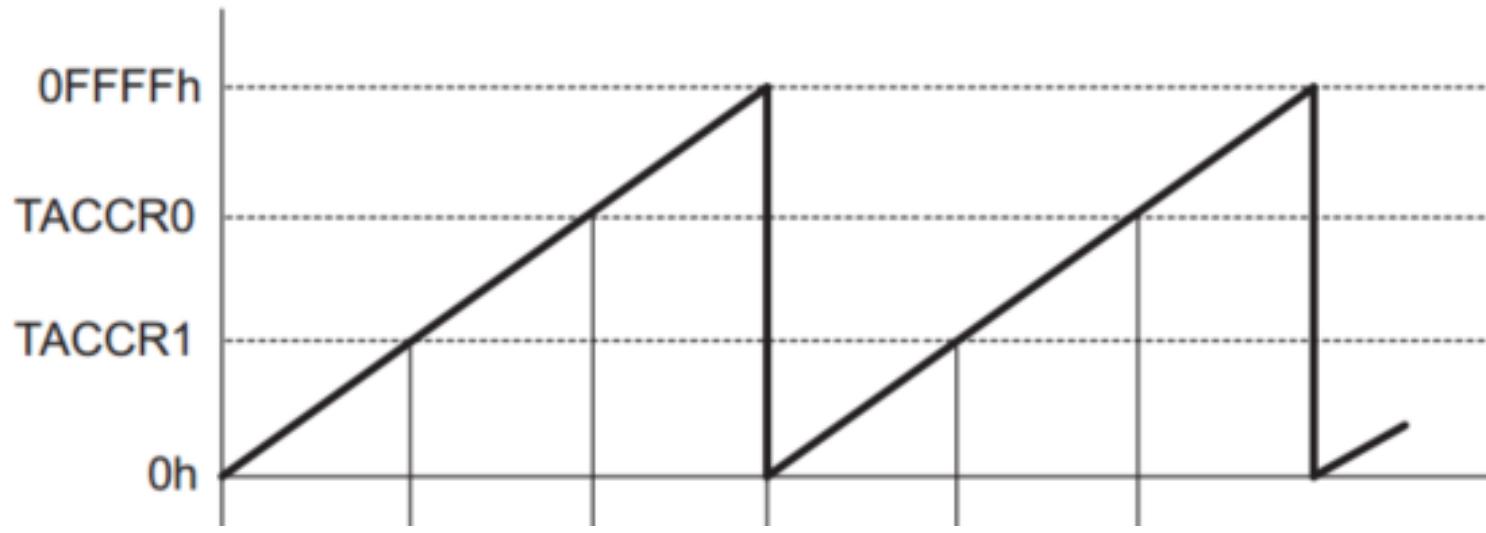
Source: MSP430G2x53, MSP430G2x13 Mixed Signal Microcontroller (Rev. J)

Timer, Up/Down Mode



Source: MSP430G2x53, MSP430G2x13 Mixed Signal Microcontroller (Rev. J)

Timer, Continuous Mode

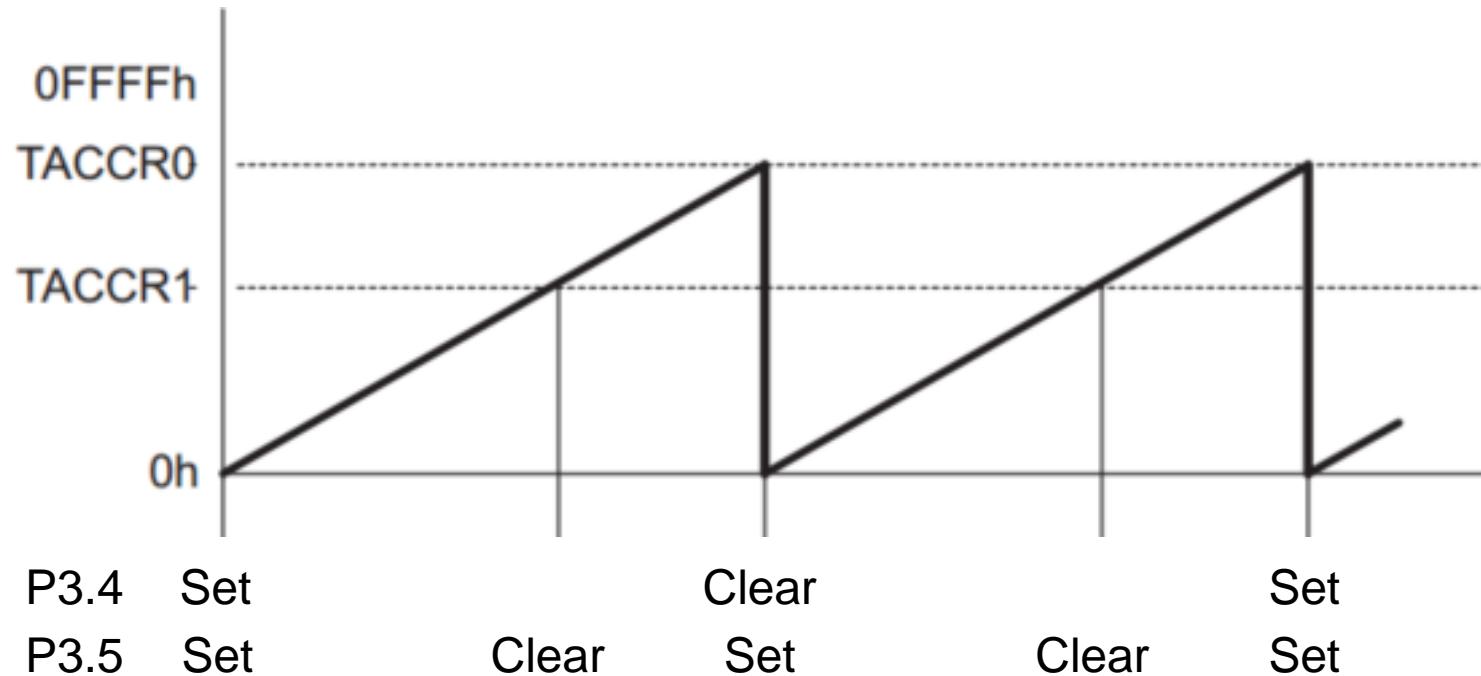


Source: MSP430G2x53, MSP430G2x13 Mixed Signal Microcontroller (Rev. J)

2. Basic peripherals

PWM (WITH TIMER)

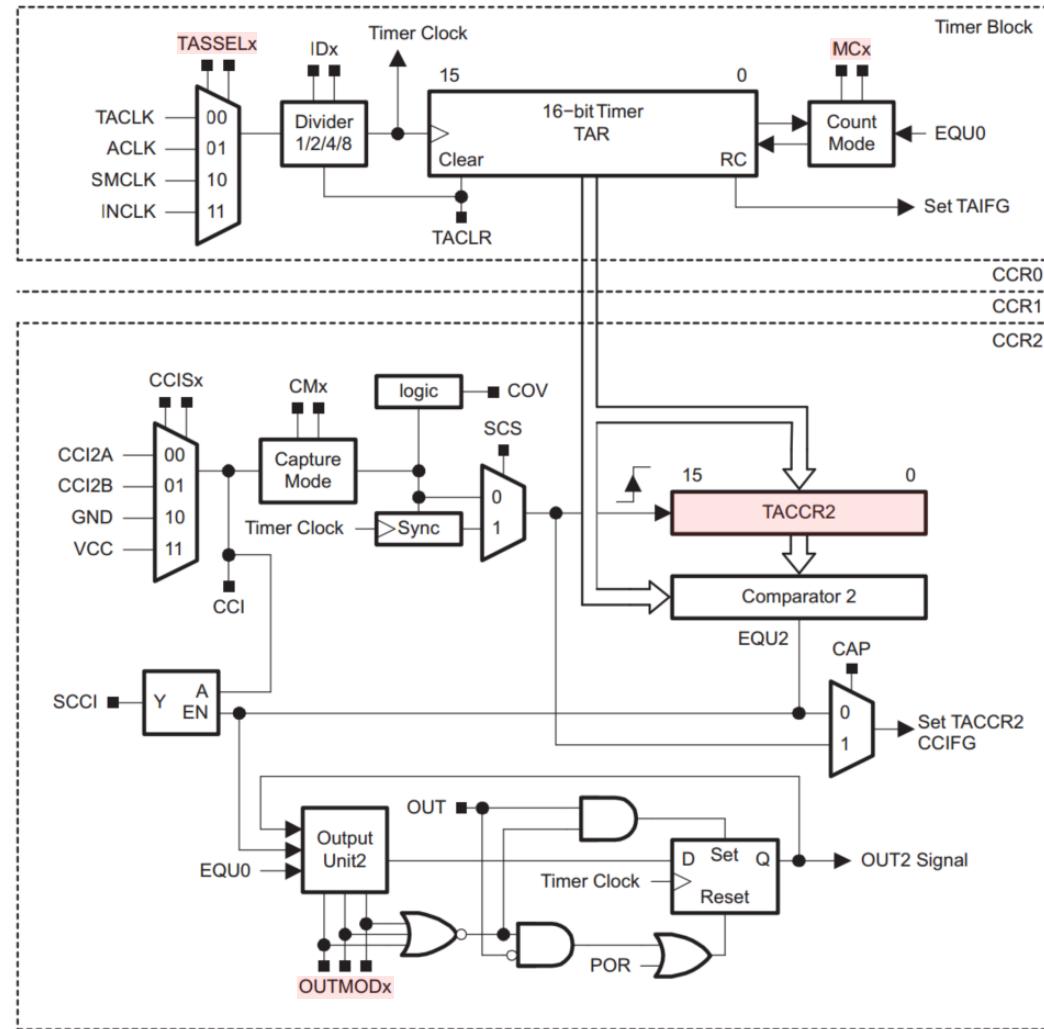
Timer, Up Mode



Source: MSP430G2x53, MSP430G2x13 Mixed Signal Microcontroller (Rev. J)

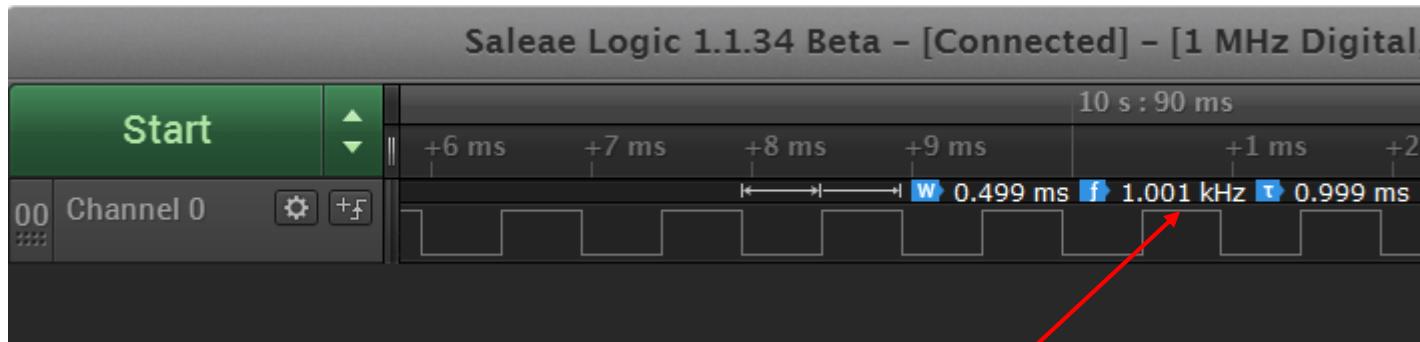
```
P3DIR |= BIT5;           // P3.5 output
P3SEL |= BIT5;           // P3.5 TA1 option
TA0CTL = TASSEL_2 + MC_1; // SMCLK; MC_1 (up mode)
TA0CCTL1 = OUTMOD_3;     // CCR1 set/reset
TA0CCR0 = 1000;          // Period: 1000 cycles ≈ 1000 us @ 1 MHz
TA0CCR1 = 500;           // CCR1: duty cycle (50 %)
```

PWM with timers: MSP430, C-Example



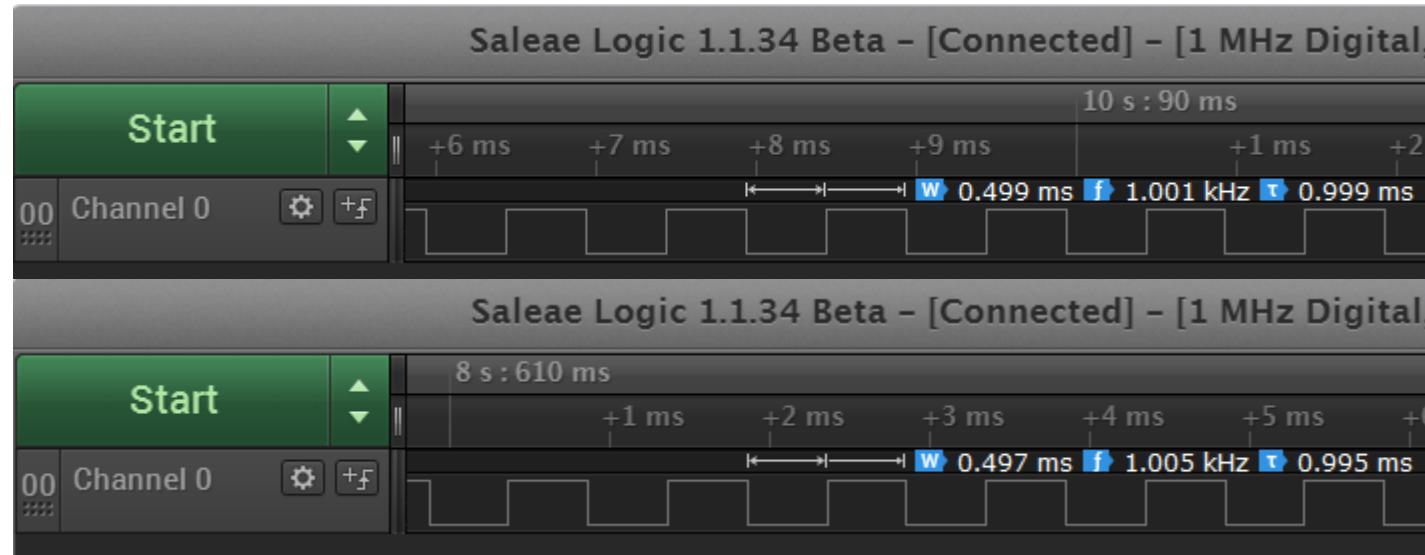
Source: TI:
MSP430G2x53
MSP430G2x13 –
Mixed Signal
Microcontroller:
Datasheet

PWM with timers: MSP430



(Light / temperature-dependent *jitter* due to DCO)

PWM with timers: MSP430



25 °C

-25 °C

PWM with timers: advantages

- No software loop required (no “active wait”)
- CPU can
 - sleep
 - perform other tasks
- Thus, it allows for
 - power savings
 - faster processing of other tasks
- Usually more accurate than software loops

State of the lecture

- ✓ Introduction

- ✓ Chapter 0: basic knowledge
 - ✓ Number systems, digital logic, ...

- ✓ Chapter 1: calculation and control system
 - ✓ Basic operations in CMOS, registers, memory, concepts, ...

- ✓ Chapter 2: Basic peripherals
 - ✓ GPIO, ADC, ...

- Chapter 3: Advanced Peripherals
 - Watchdog, programming interfaces, ...

3. Advanced Peripherals / Advanced Hardware Features

clock management, sleep modes, watchdog, programming interfaces

3. Advanced Peripheral / Advanced Hardware Features

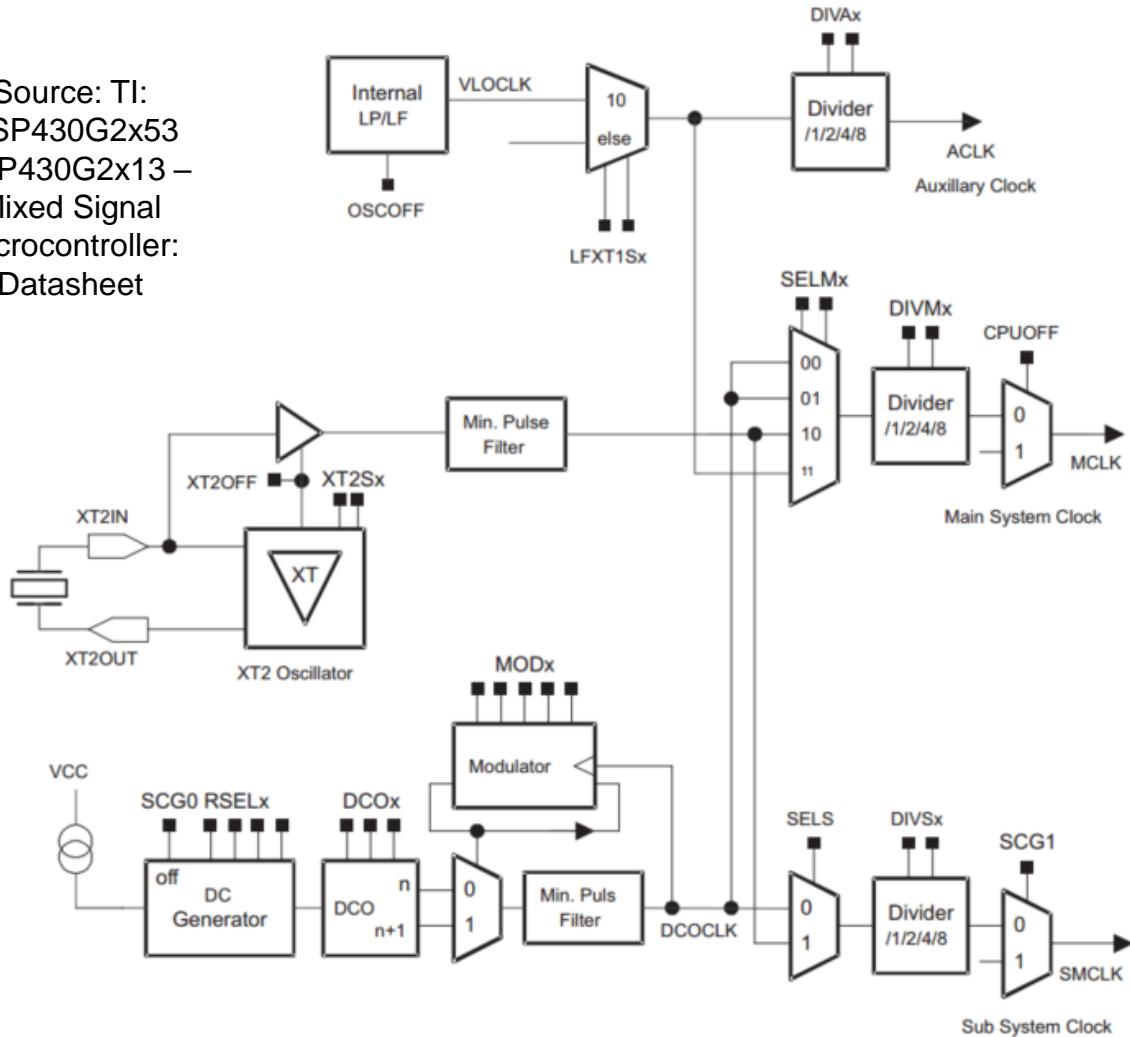
CLOCK MANAGEMENT

Why different clocks?

- Individual peripherals should run at different speeds
- Example:
 - ADC should sample as fast as possible because the analog value changes rapidly
 - CPU should operate as efficiently as possible
 - Difficult to implement with a single clock, since a fast clock will lead to higher power consumption in the CPU, but a slower clock will lead to insufficient sampling rates

Clock modul MSP430

Source: TI:
 MSP430G2x53
 MSP430G2x13 –
 Mixed Signal
 Microcontroller:
 Datasheet



Available Clocks, MSP430

- MCLK
 - Master Clock
- SMCLK
 - Sub-System Master Clock
- ACLK
 - Auxillary Clock

- LFXT1CLK
 - Low frequencies (32 kHz) as well as high frequencies (20 MHz)
 - External source
 - connection via XIN / XOUT (P2.6 / P2.7 → PxSEL ...)
- VLOCLK
 - Low, fixed adjusted frequency (approx. 12 kHz)
temperature dependent! 4 kHz to 20 kHz possible!
 - Internal oscillator
 - energy efficient
 - LFXT1 is automatically deactivated, if VLO is used
- DCOCLK
 - Frequency can be adjusted flexibly by software (0.06 MHz to 26.0 MHz)
 - Internal oscillator

■ MCLK

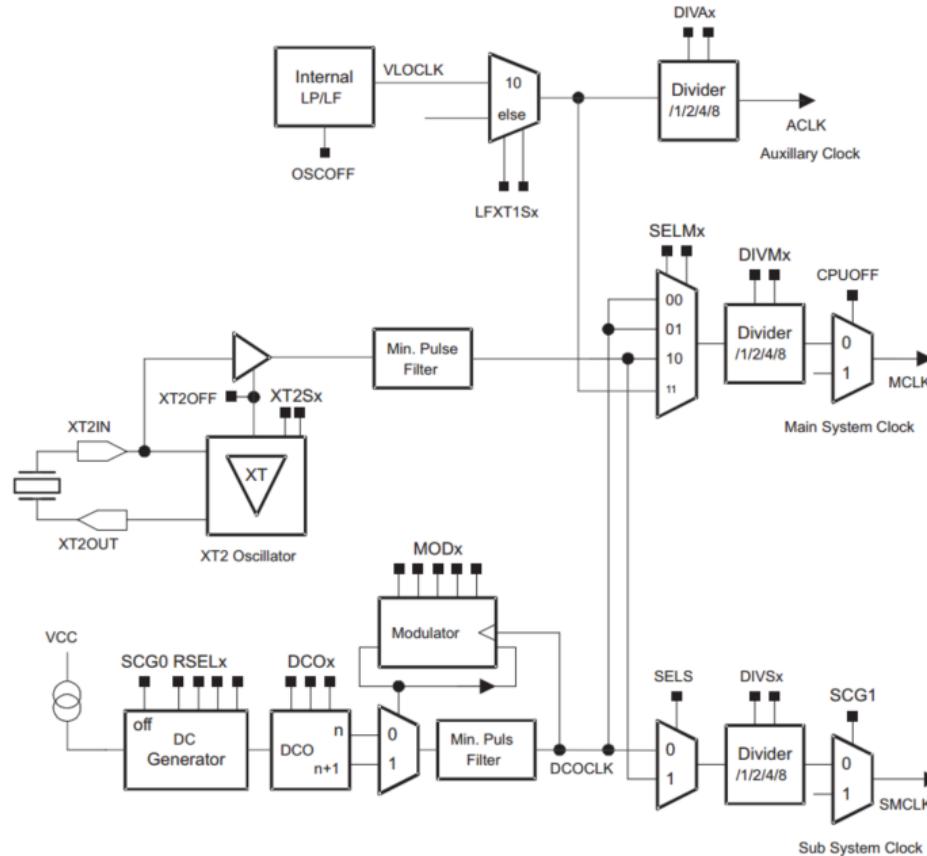
- DCO
- LFXT1
- VLOCLK

■ SMCLK

- DCO
- LFXT1

■ ACLK

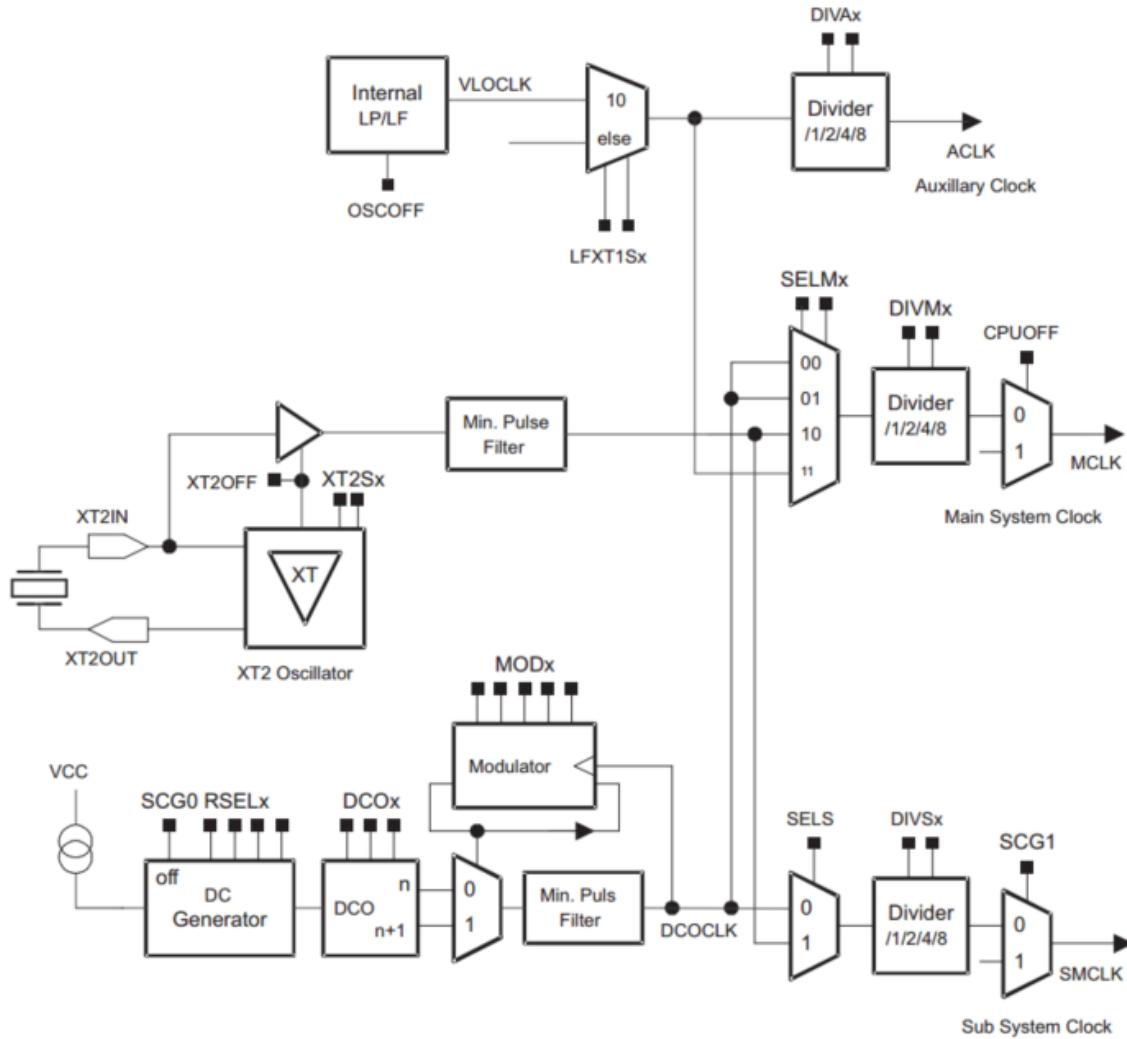
- LFXT1CLK
- VLOCLK



Clock-Prescaler

- sometimes „slow“ is still to fast:
frequencies below 12 kHz?
- Prescaler to divide the actual clock by a fixed factor
- In case of the MSP430:
 - 1 (if the frequency is not to be changed)
 - 2
 - 4
 - 8
 - applicable to each clock
- Prescaler are partially used for precise clocks (higher base frequency and corresponding prescaler):
→ Averaging, thereby reducing the jitter error)

Clock-Modul MSP430



3. Advanced Peripherals / Advanced Hardware Features

SLEEP MODES

Sleep modes

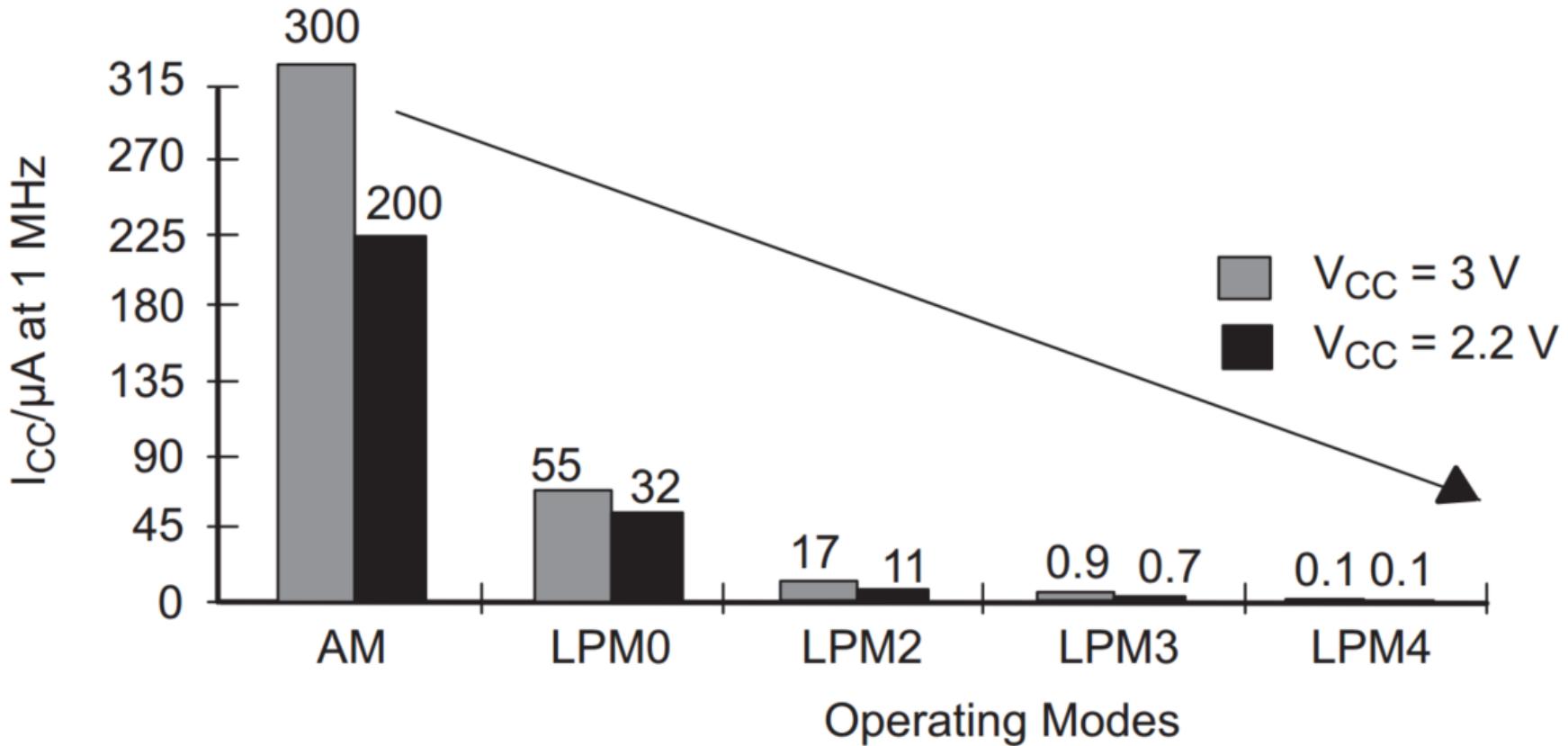
- objective: „low power“ – reduce energy consumption
- Possibilities:
 - Lower or multiple clock rates
See clock management
 - Turn off unused peripherals
E.g. ADCs, timers, ... - but also external hardware as radio modules
 - CPU should „go to sleep“

Sleep modes - LPM

- Active
 - CPU active, all clocks on.
- LPM0
 - CPU and MCLK off, SMCLK and ACLK on. DCO is on.
- LPM1
 - CPU and MCLK off. SMCLK and ACLK on. DCO is off, if SMCLK is not using it.
- LPM2
 - CPU, MCLK, SMCLK, DCO on, ACLK on.
- LPM3
 - CPU, MCLK, SMCLK, DCO off, ACLK on.
- LPM4
 - CPU off, all clocks off.

Sleep modes – MSP430

Source: MSP430x2xx Family, User's Guide (Rev. July 2013)



Sleep modes – below 100 nA

- Some MSP430:
Switching off the complete voltage regulation (LPM4.5)
 - Up to 90% lower consumption (however, some series can not use LPM4.5 ...)
- LPM5 / Deep sleep mode:
 - Voltage supply for the RAM is disabled
 - Contents of registers and RAM is therefore lost(!)
 - power consumption ... ?
No more numbers from TI, since the mode is called by TI itself as "dangerous".
- Deep sleep from the competition:
 - Microchip PIC24F: < 30 nA

Sleep modes – time sequence

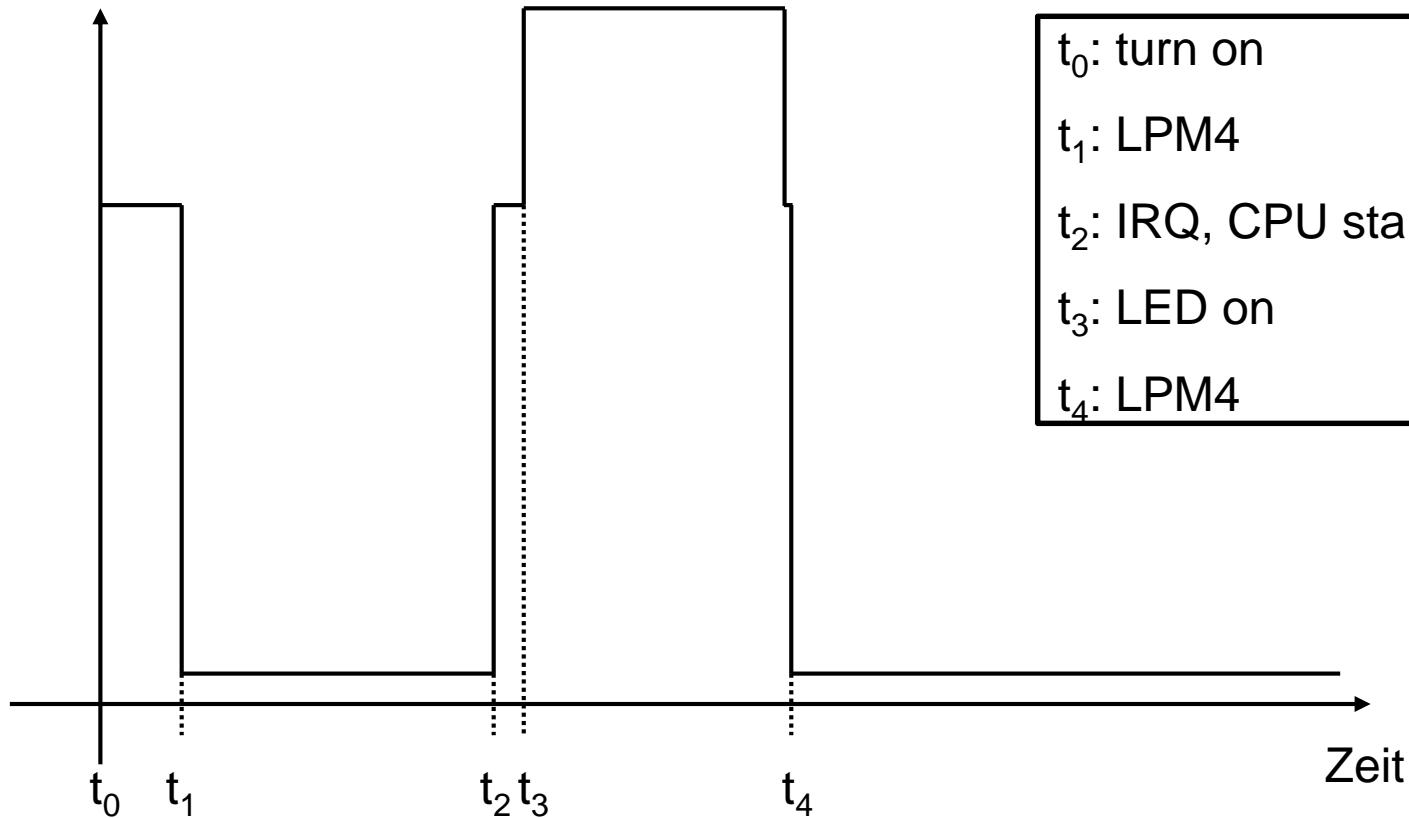
- CPU is actively entering a LPM (lower power mode).
- CPU is woken from sleep by interrupts
 - Timers (periodically)
 - Comparators (Above / below a limit for the analog signal)
 - Digital signals (button pressed, switch closed, etc.)
 - ...
- In such an interrupt, the CPU is usually brought back to active mode so that the pending task can quickly be performed

Sleep modes – C-Example, MSP430

```
...
while (1) {
    __bis_SR_register(LPM4_bits | GIE); //Enter low power mode
    P1OUT ^= BIT5; // program continues here if exit from LPM
    __delay_cycles(200000);
    P1OUT ^= BIT5;
}
...
#pragma vector=PORT1_VECTOR
_interrupt void Port_1 (void) {
    P1IFG &= ~BIT0;
    __bic_SR_register_on_exit(LPM4_bits); //Exit low power mode
}
```

Sleep modes – C-Example, MSP430

power consumption



Sleepmode – Warning

Deep sleep modes can be critical if they are not carefully programmed!

If a processor is send into deep sleep right after start-up, it may be that the programming / debugging tools no longer work; at worst, it may be that the chip has to be replaced, because you can not put it back to programming mode.

3. Advanced Peripheral / Advanced Hardware Features

WATCHDOG

Watchdog

- Problem: a program can hang because of...
 - Programming errors
 - Undefined boundary conditions
 - External interference such as electromagnetic waves
 - ...
 - Restart is required
- Expensive & time-intensive

- Idea:
 - Timer that just counts
 - Program regularly reports that it is still alive
 - If an overflow is reached, a PUC (Power-Up Clear) is triggered
 - System restarts, program starts again from the beginning
- In the MSP430:
 - Timer (as discussed in chapter 2)
 - 16 Bit (I.e. a maximum of 32768 cycles to the overflow)
 - Adjustable reset time: 32768, 8192, 512 and 64 clock cycles until reset is performed
 - Longer intervals can be achieved by using different clocks

- Simplest case:

```
WDTCTL = WDTPW | WDTHOLD; // Stop watchdog
```

- monitoring:

```
WDTCTL = WDTPW | WDTCNTCL; // Reset watchdog
```

Watchdog – alternative use cases

- Usage as a timer:
 - During overflow, a PUC is not triggered, but a software interrupt.
- Use as software reset:
 - If WDTPW is not specified, a PUC is triggered directly
 - Simple way to restart the chip from the software

3. Advanced Peripherals / Advanced Hardware Features

PROGRAMMING INTERFACES

Our microcontroller is almost ready, but the program is still missing ...

- We do not want to include a ROM
- A programming interface must be created ...

- ... and we want to debug the program during development

Programming interfaces

There are basically two variants:

- Programming methods that work only without a surrounding circuit
(= cannot be used when the chip is embedded into a system)
 - Atmel High-Voltage-Flashing, PIC Burning, ...
 - Programming usually with voltages outside the permissible operating voltage of the chip
 - This would often result in destruction of the surrounding hardware
 - Programming is usually independent from the currently loaded software
- Programming methods that work with / in spite of the surrounding circuit
 - Atmel ICSP, Atmel DebugWire, TI SpyByWire, JTAG, ...
 - Programming with (approximately) the supply voltage of the target system
 - Programming often depends on the loaded software

Programming interfaces

- ISP / ICSP with SPI
 - Atmel AVR, Microchip PICmicro, ...
 - Typically: only access to fuses and the flash memory
 - Four lines: Data In, Data Out, Clock, Reset
- JTAG
 - Standardized method (Daisy-chained JTAG - IEEE 1149.1)
 - Five lines: Data In, Data Out, Clock, Reset, Mode Select
- SBW (Spy-Bi-Wire)
 - Serialized JTAG, TI/MSP430-specific
 - Two lines: Data, Clock
 - Is used e.g. on the board for the practical exercise
- DebugWIRE
 - Serial protocol, Atmel-specific (→ no official documentation available)
 - one single line

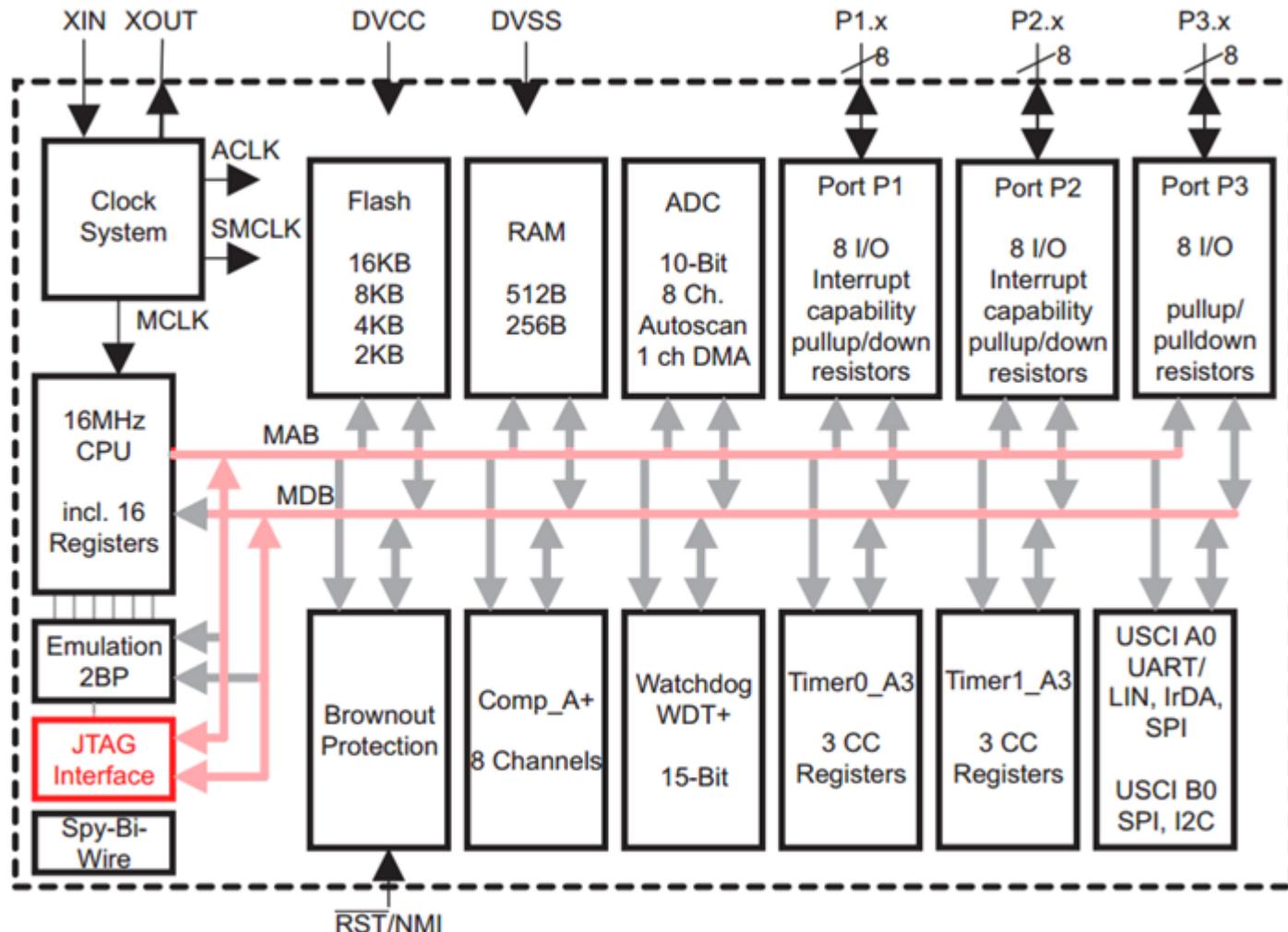
In Circuit/System Programming

- Supports only one target at a time
- Restricted access to the address and memory bus of the chip
- Direct write to / read from flash memory
- access to „fuses“
 - Settings that can not be changed by the actual program
 - For example: (De)activating the ISP access, read-flags, ...

Joint Test Action Group

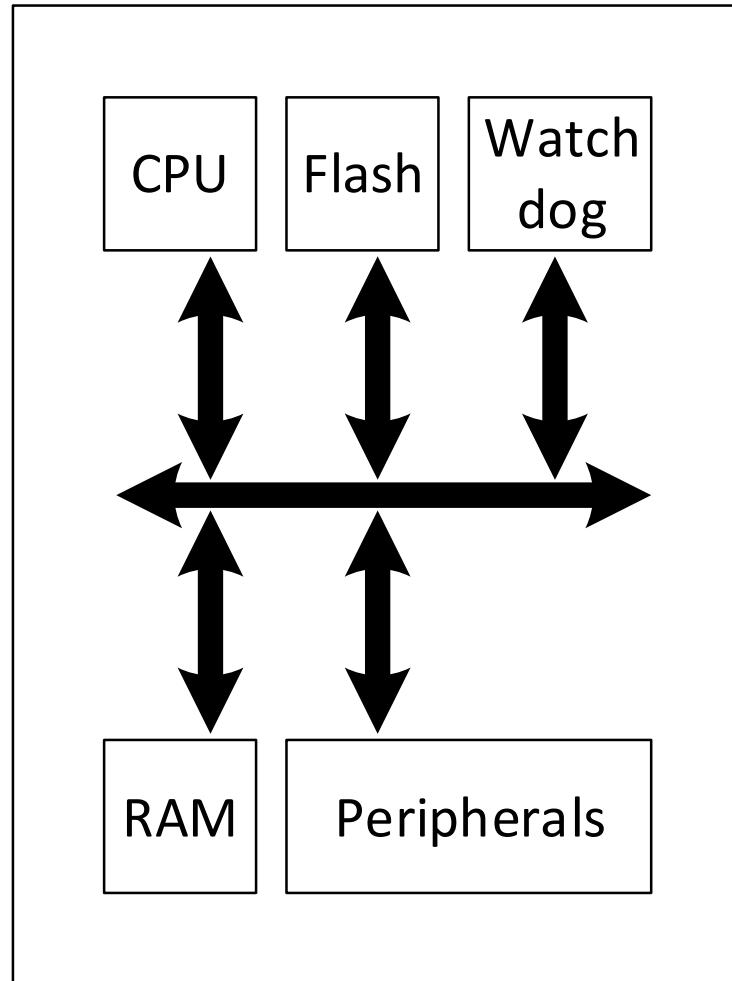
- Supports the communication with several devices simultaneously (so-called “JTAG Chain”)
- Requires special, dedicated hardware within the target
- Full read and write access to most registers within the target; access to memory (Flash, RAM) via (partly manufacturer-specific) commands within the protocol possible

Programming interfaces – JTAG

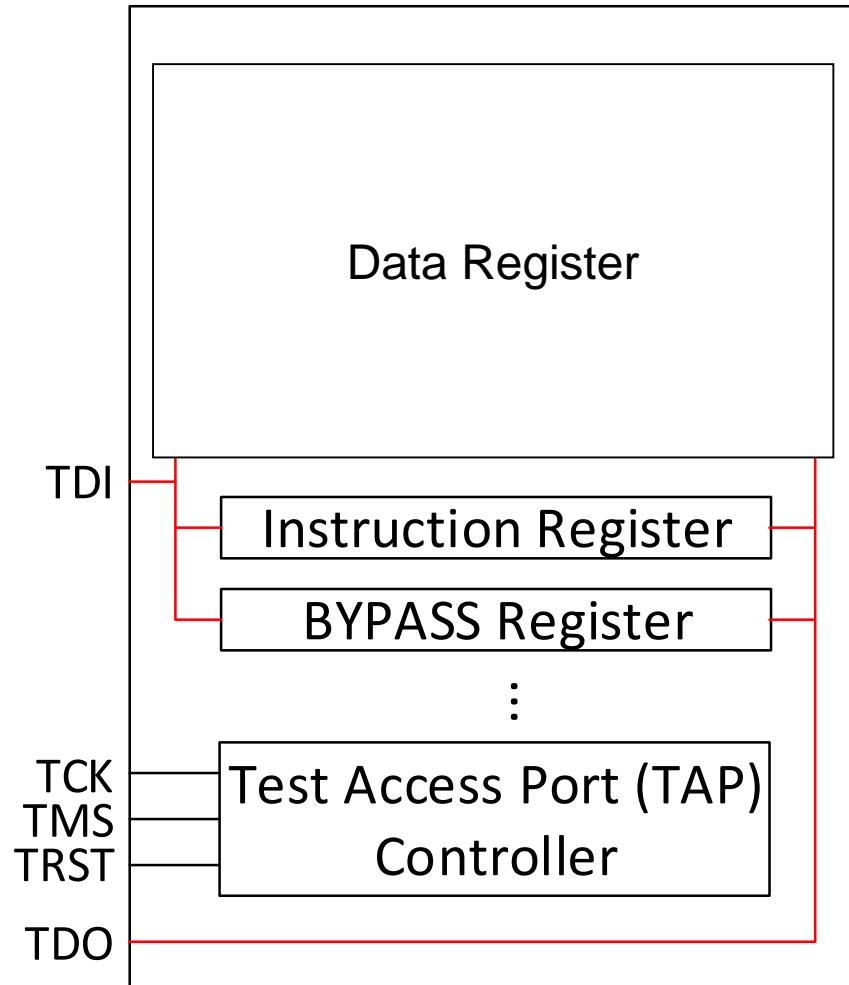


Source: MSP430G2x53, MSP430G2x13 Mixed Signal Microcontroller (Rev. J)

Programming interfaces – JTAG

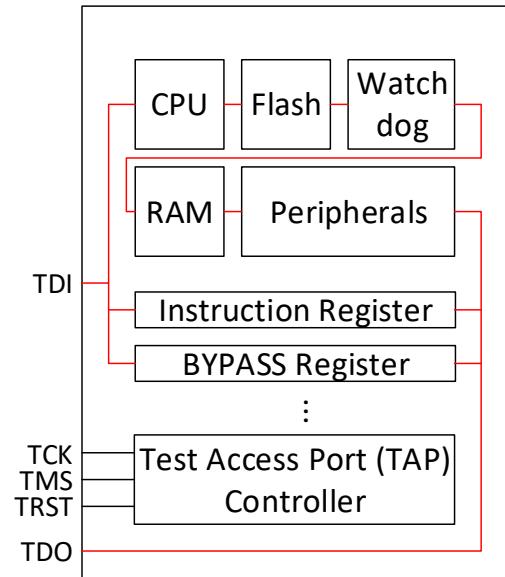


Programming interfaces – JTAG

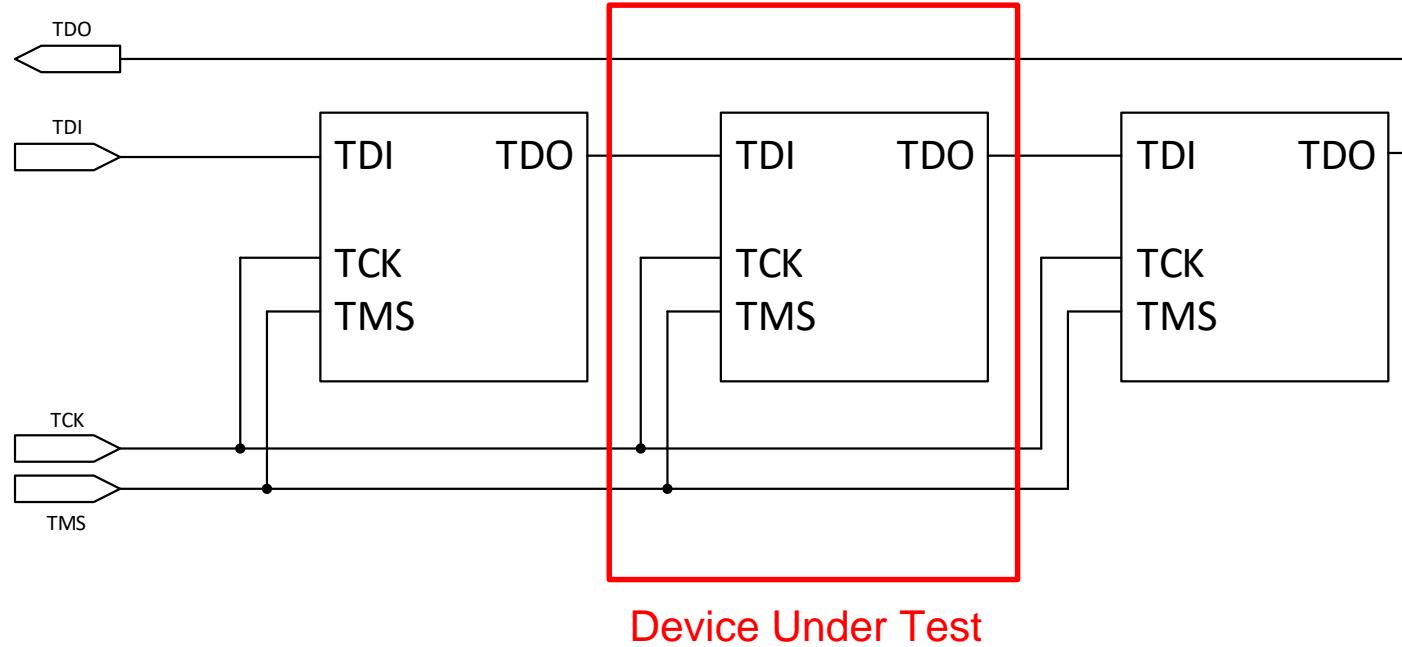


Programming interfaces – JTAG

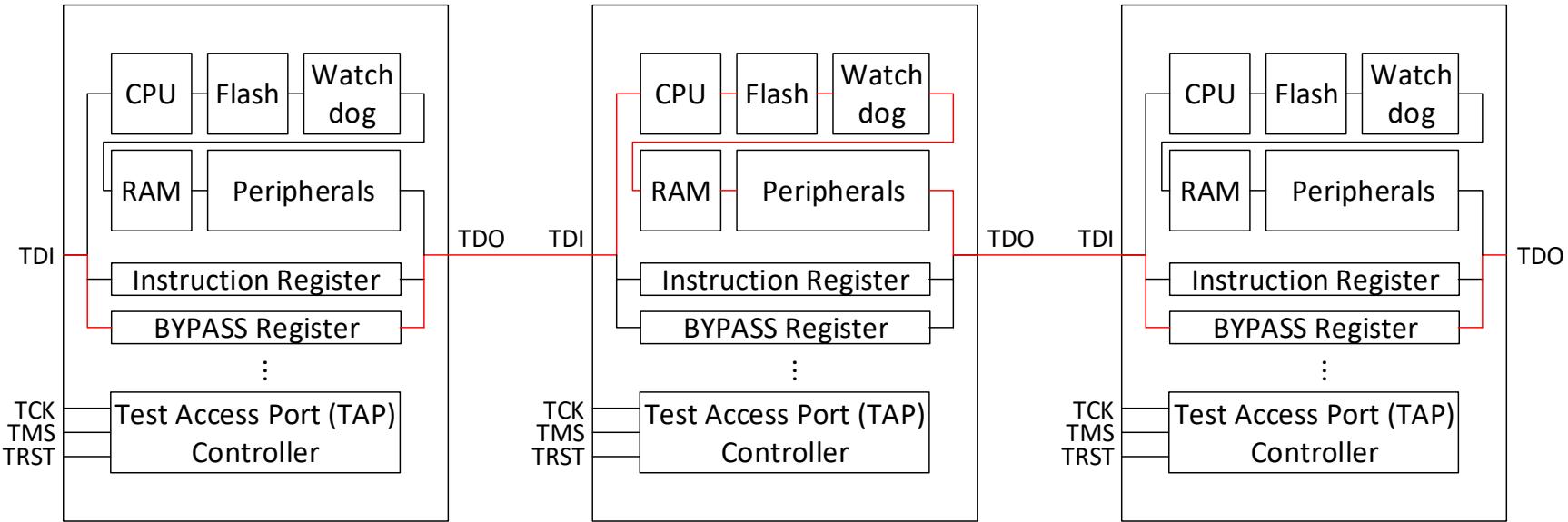
- TDI
 - Test Data In
 - Change of registers possible
- TDO
 - Test Data Out
 - Read all connected registers possible
- TCK
 - Test Clock
- TMS
 - Test Mode Select
- TRST
 - Test Reset (Optional, can also be triggered internally (i.e., by TMS))



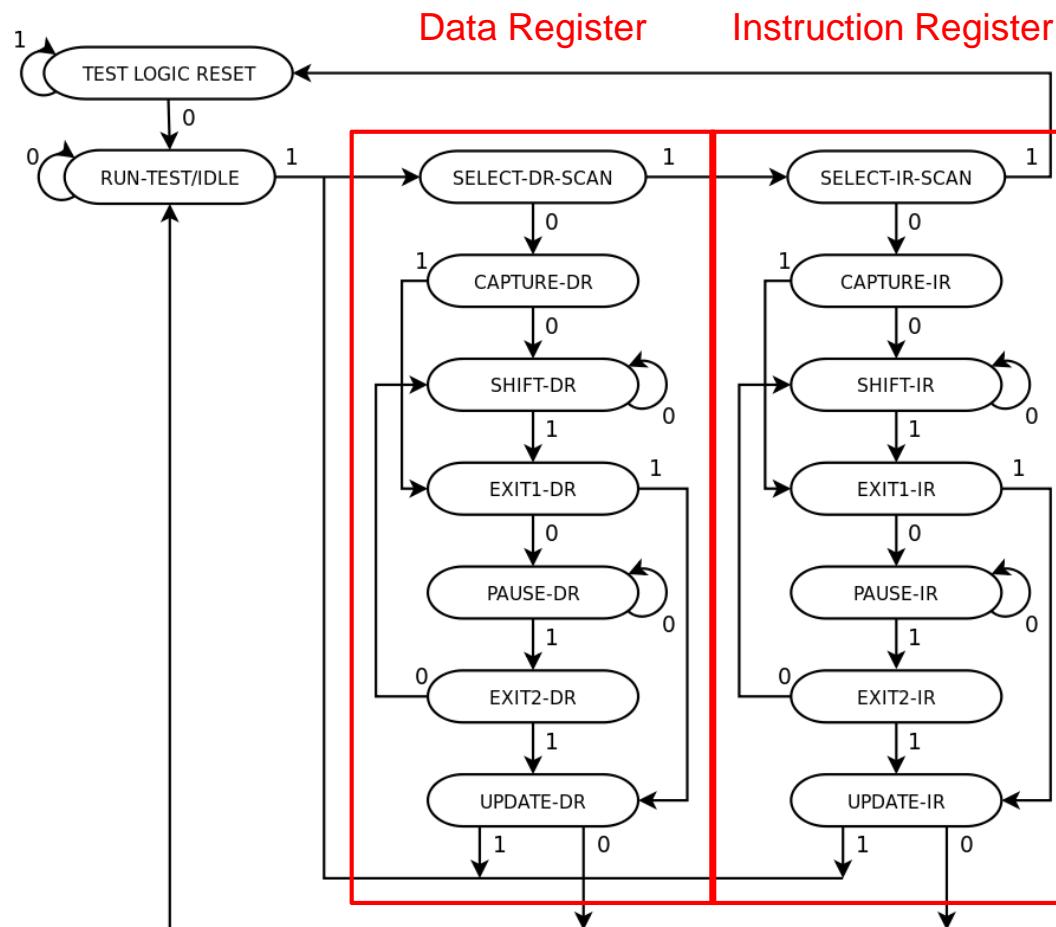
Programming interfaces – JTAG-Chain



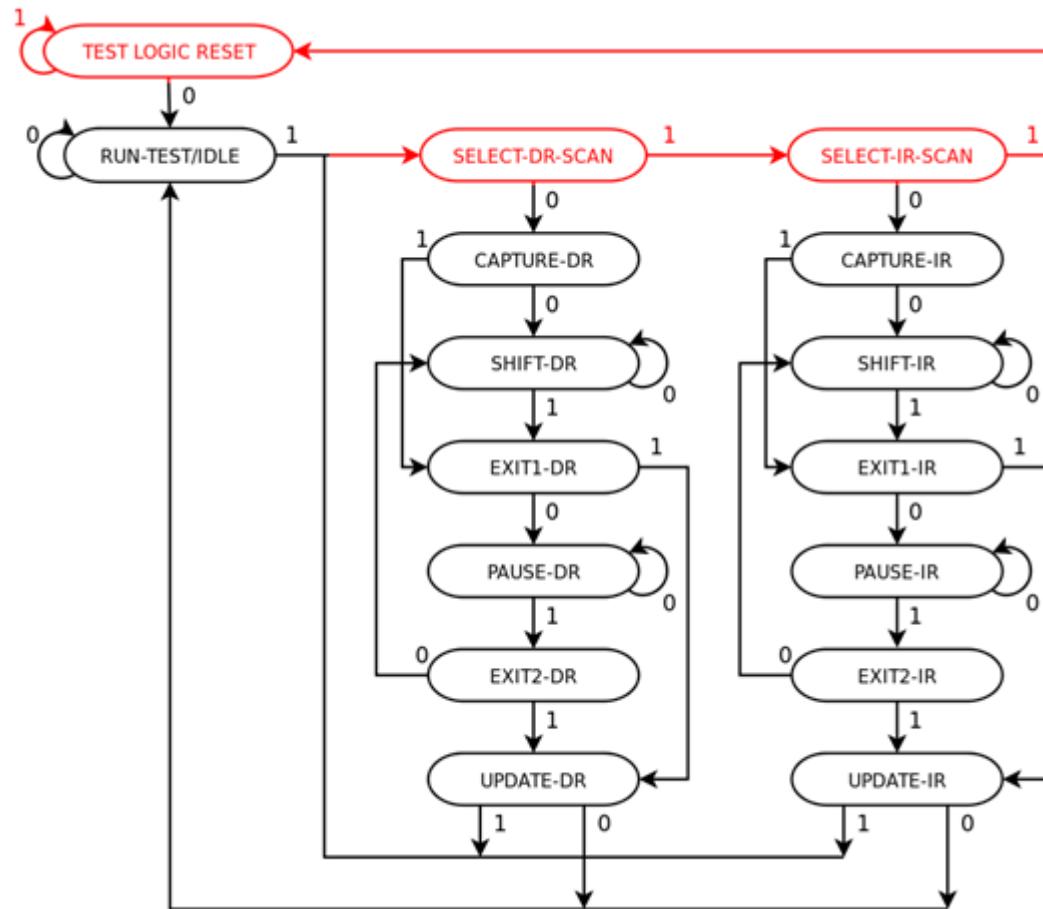
Programming interfaces – JTAG-Chain with DUT



- Length of the "Boundary Scan Chain" must be known (information from the manufacturer)
- Length of the JTAG chain must be known (fill BYPASS with zeros, then push a one and count)



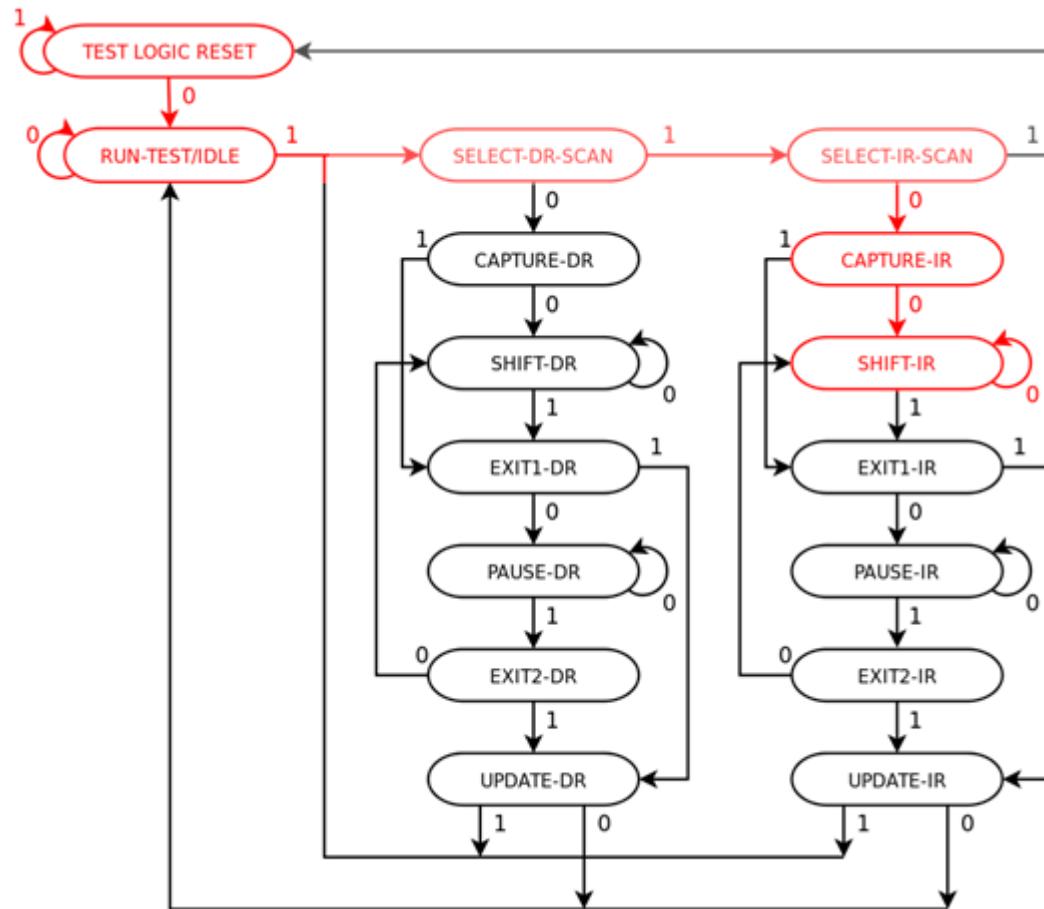
Source: Rudolph H / Wikipedia (Bild-frei)
http://de.wikipedia.org/wiki/Datei:JTAG_TAP_Controller_State_Diagram.svg



RESET:

TMS: 11111

Source: Rudolph H / Wikipedia (Bild-frei)
http://de.wikipedia.org/wiki/Datei:JTAG_TAP_Controller_State_Diagram.svg



BYPASS:
TMS: 01100 0...0
TIN: xxxx 1...1

Source: Rudolph H / Wikipedia (Bild-frei)
http://de.wikipedia.org/wiki/Datei:JTAG_TAP_Controller_State_Diagram.svg

State of the lecture

- ✓ Introduction

- ✓ Chapter 0: basic knowledge
 - ✓ Number systems, digital logic, ...

- ✓ Chapter 1: calculation and control system
 - ✓ Basic operations in CMOS, registers, memory, concepts, ...

- ✓ Chapter 2: Basic peripherals
 - ✓ GPIO, ADC, ...

- ✓ Chapter 3: Advanced Peripherals
 - ✓ Watchdog, programming interfaces, ...

State of the lecture

- Chapter 4: Programming
 - Assembler, C, Compiler, ...
- Chapter 5: Communication
 - OSI-Model, topologies, UART, I2C, ...
- Chapter 6: Outlook (among others: Safe Systems)

4. Programming

Instructions / machine code, assembler, C, graphical programming

4. Programming

INSTRUCTIONS / MACHINE CODE

Instructions / Machine Code - Reminder

Opcode	function
0 0 0	$a - b$
0 0 1	$b - a$
0 1 0	$a + b$
0 1 1	0
1 0 0	$a \wedge b$
1 0 1	$a \vee b$
1 1 0	$a \oplus b$
1 1 1	1

- 16 bit wide (With optional expansion to 32 bits for Immediate commands)
- MSP 430 has 28 implemented commands (32 possible)
- Opcode itself has variable length
 - Double-Operand („Format I“): 4 Bit
 - Single-Operand („Format II“): 6 Bit Prefix + 3 Bit Opcode = 9 Bit
 - Jump: 3 Bit + 3 Bit „condition“ (jump condition) = 6 Bit
- Machine code is architecture dependent!

Instructions / Machine Code - MSP430

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0xxx																
4xxx																
8xxx																
Cxxx																
1xxx	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI			
14xx																
18xx																
1Cx _x																
20xx							JNE/JNZ									
24xx							JEQ/JZ									
28xx							JNC									
2Cx _x							JC									
30xx							JN									
34xx							JGE									
38xx							JL									
3Cx _x							JMP									
4xxx							MOV, MOV.B									
5xxx							ADD, ADD.B									
6xxx							ADDC, ADDC.B									
7xxx							SUBC, SUBC.B									
8xxx							SUB, SUB.B									
9xxx							CMP, CMP.B									
Axxx							DADD, DADD.B									
Bxxx							BIT, BIT.B									
Cxxx							BIC, BIC.B									
Dxxx							BIS, BIS.B									
Exxx							XOR, XOR.B									
Fxxx							AND, AND.B									

Source:
MSP430x2xx Family
– User’s Guide

Instructions / Machine Code - MSP430

- Example Format I: ADD

Opcode					Source			Ad	B/W	As	Destination				
0	1	0	1	s3	s2	s1	s0	ad	bw	a1	a0	d3	d2	d1	d0

- Example Format II: CALL

Opcode					B/W		Ad/As		Dest./Source						
0	0	0	1	0	0	1	0	1	0	a1	a0	s3	s2	s1	s0

- Example Jump: JEQ / JZ (Jump if Equal / Jump if Zero)

Opcode		Condition			PC Offset										
0	0	1	0	0	1	o9	o8	o7	o6	o5	o4	o3	o2	o1	o0

- Ad (Addressing Mode, Destination) As (Addressing Mode, Source)
- B/W (Byte (1) or Word (0) Operation)

Instructions / machine code - Hollywood etc.

- Quote: "You can read machine language!?"
(from: WHO AM I, 2014)
- Machine code is not impossible to read - but very hard.
- BUT: Not because machine code itself is very complicated, but because the formatting is uncommon:
01000011100100010000000000
00000001000011101000010000
00000000001001000001000111
11000000000000001001010001
0010111101001111000000100
0000000000100
- Even difficult with clear formatting:
01000011 10010001
00000000 00000000
01000011 10100001
00000000 00000010
01000001 00011111
00000000 00000010
01010001 00101111
01001111 10000001
00000000 00000100
- Or in hexadecimal notation:
4391 0000
43A1 0002
411F 0002
512F
4F81 0004

Insertion: CISC / RISC

- The developed CPU is a RISC CPU
 - Reduced Instruction Set Computer
 - Smaller command set
 - Constant number of cycles per command
 - Many registers to handle larger programs without permanent main memory access
- In contrast to: CISC
 - Complex Instruction Set Computer
 - Significantly larger command set
 - Different number of cycles per command
 - Can often process several commands at the same time by more powerful pipelining
 - Fewer registers required because a lot is achieved via internal "intermediate registers"

4. Programming

ASSEMBLER

- The "first" language
- Each command in assembler is translated directly into a few lines (usually one to three lines) of machine code
- Assembler commands have “readable” names instead of binary prefixes:
 - 0101 becomes ADD
 - 000100101 becomes CALL
 - 001001 becomes JEQ or JZ
- The machine code from before:

4391 0000	MOV.W	#1,0x0000(SP)
43A1 0002	MOV.W	#2,0x0002(SP)
411F 0002	MOV.W	0x0002(SP),R15
512F	ADD.W	@SP,R15
4F81 0004	MOV.W	R15,0x0004(SP)

- Architecture dependency generally persists

What is that code doing?

- MOV.W #1,0x0000(SP)
 - Put a 1 in RAM at the point where SP (the stack pointer) is currently pointing
- MOV.W #2,0x0002(SP)
 - Place a 2 in the RAM at the location pointed to (SP + 2)
- MOV.W 0x0002(SP),R15
 - Copy the content of the cell with the address (SP + 2) into the register R15
- ADD.W @SP,R15
 - Add the contents of the cell at address SP to register R15
- MOV.W R15,0x0004(SP)
 - Copy the contents of R15 into the cell at address SP

What is that code doing?

- t_0
 - MOV.W #1,0x0000(SP)
 - t_1
 - MOV.W #2,0x0002(SP)
 - t_2
 - MOV.W 0x0002(SP),R15
 - t_3
 - ADD.W @SP,R15
 - t_4
 - MOV.W R15,0x0004(SP)
 - t_5
- SP is the address of the stack pointer and is set by the compiler

CPU	t_0	t_1	t_2	t_3	t_4	t_5
R13	X	X	X	X	X	X
R14	X	X	X	X	X	X
R15	X	X	X	2	3	3

RAM	t_0	t_1	t_2	t_3	t_4	t_5
SP	X	1	1	1	1	1
SP + 2	X	X	2	2	2	2
SP + 4	X	X	X	X	X	3

Instructions / Assembler - MSP430

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0xxx																
4xxx																
8xxx																
Cxxx																
1xxx	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI			
14xx																
18xx																
1Cx _x																
20xx							JNE/JNZ									
24xx							JEQ/JZ									
28xx							JNC									
2Cx _x							JC									
30xx							JN									
34xx							JGE									
38xx							JL									
3Cx _x							JMP									
4xxx							MOV, MOV.B									
5xxx							ADD, ADD.B									
6xxx							ADDC, ADDC.B									
7xxx							SUBC, SUBC.B									
8xxx							SUB, SUB.B									
9xxx							CMP, CMP.B									
Axxx							DADD, DADD.B									
Bxxx							BIT, BIT.B									
Cxxx							BIC, BIC.B									
Dxxx							BIS, BIS.B									
Exxx							XOR, XOR.B									
Fxxx							AND, AND.B									

Source:
MSP430x2xx Family
– User’s Guide

Assembler code must be converted to machine code before execution.

This process is taken over by the so-called compilers; software that analyzes the assembler code and translates it into equivalent machine code.

The resulting code merely reflects the functionality of the input code, but its length does not necessarily have to be proportional to that of the input code:

MOV.W 0x0002(SP), R15

becomes

411F 0002 (2 Words)

ADD.W @SP, R15

becomes

512F (1 word)

- Improvement of machine code
- But still:
 - Hard to read, especially with long code
 - Poorly portable because the exact command syntax differs from architecture to architecture
- Idea: To increase the level of abstraction
- Several languages followed (A-0, Fortran, Lisp, COBOL, BASIC, B, Pascal, ...) until the language C was introduced by Dennis Ritchie in 1972.

4. Programming

C LANGUAGE

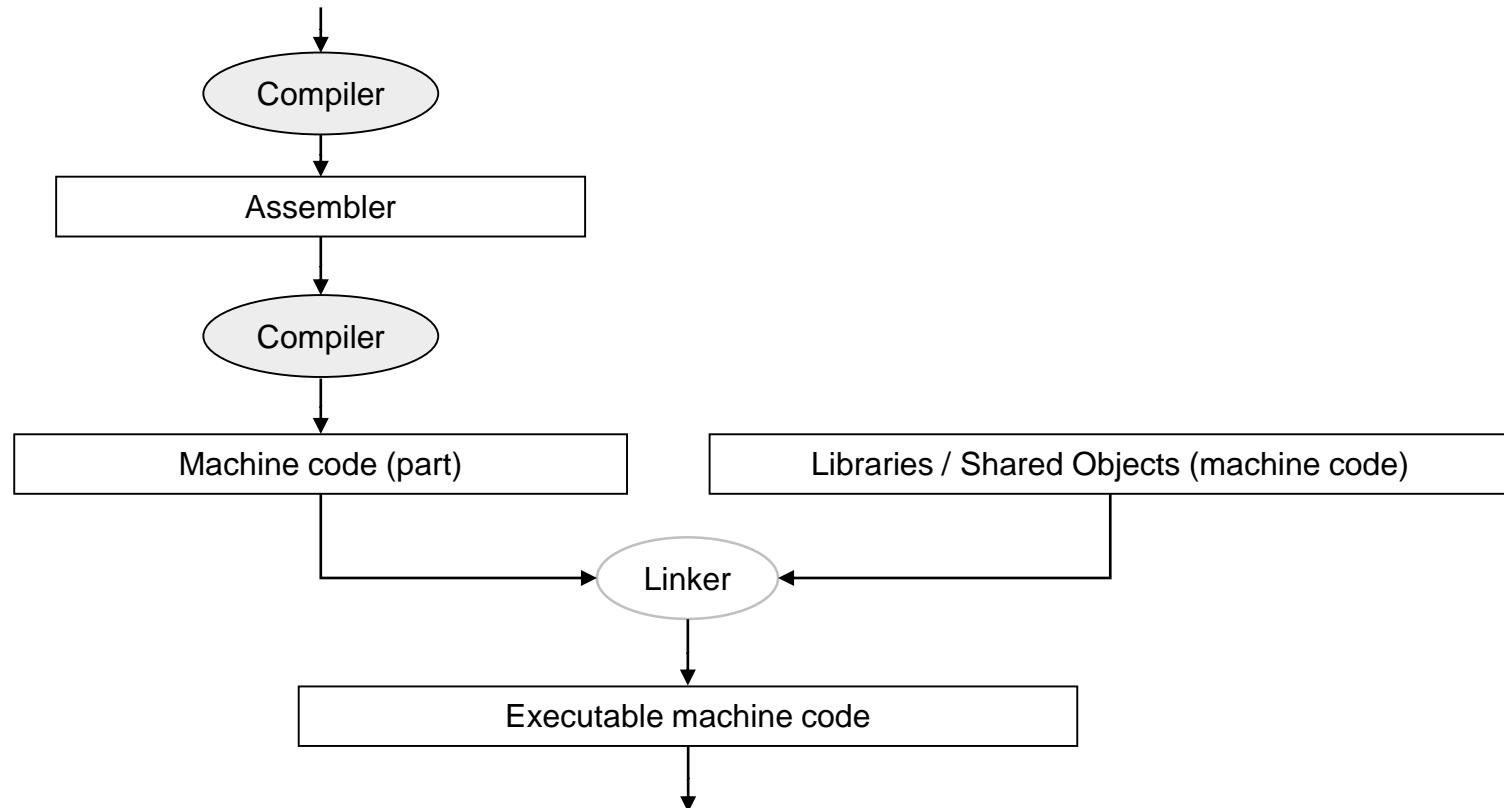
- C allows hardware-oriented programming, but offers various advantages over ASM:
 - The code can generally be kept independent from the architecture – the platform-specific implementation only happens during compilation
 - The code is easier to maintain and understand than ASM, especially by using multiple files
 - A project can be edited by several people at the same time
- However, by further moving away from the actual machine code in terms of abstraction, C code is usually slower than ASM code, and in most cases, it takes up more memory space.
- For the development on microcontrollers, C is still the language of choice - if individual sections of a program are particularly time critical, inline assembler code can be used.

```

6 #include <msp430.h>
7
8 void main(void) {
9     int a;
10    for (a = 0; a < 8; a++) {
11        P1DIR |= 1 << a;
12    }
13    while (1) {
14        P1OUT ^= 0xFF;
15    }
16 }

```

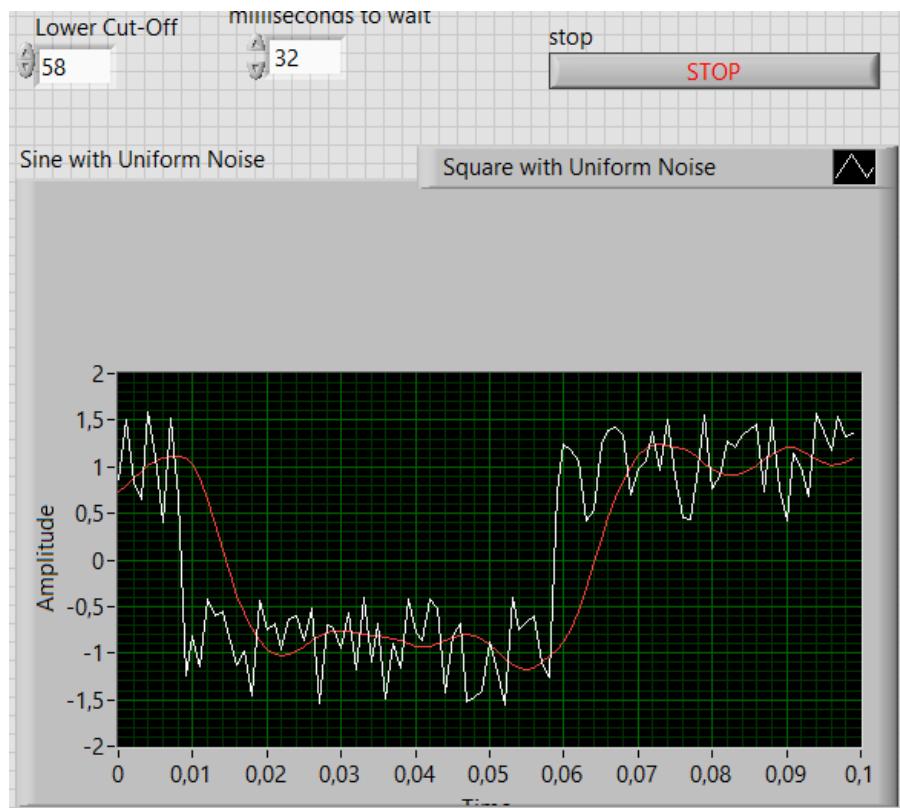
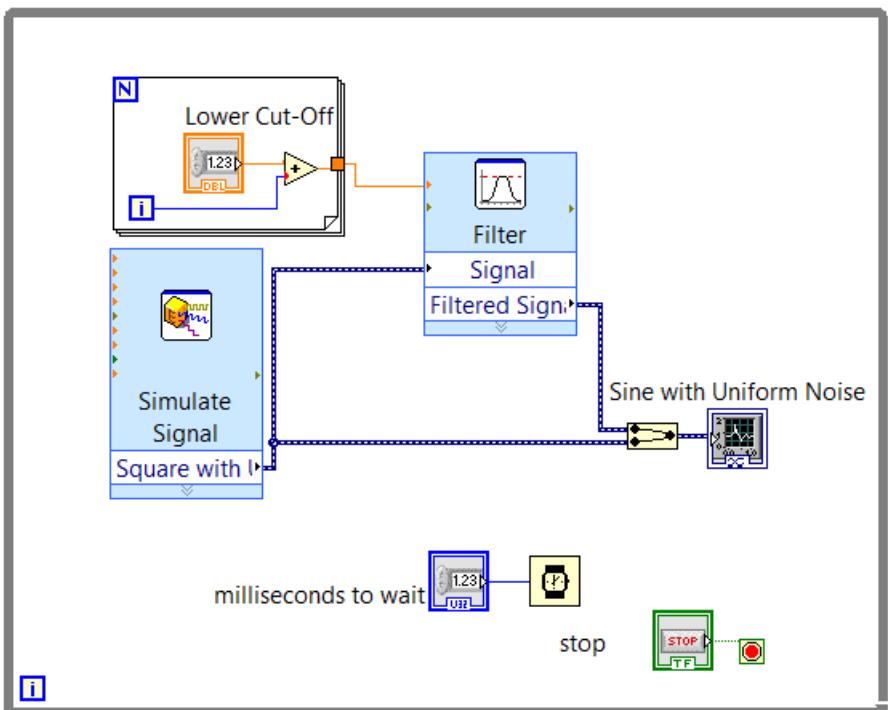
	main():	
c02c:	430F	CLR.W R15
c02e:	923F	CMP.W #8,R15
c030:	3409	JGE (\$C\$L2)
	\$C\$L1:	
c032:	431C	MOV.W #1,R12
c034:	4F0D	MOV.W R15,R13
c036:	12B0 0000	CALL # _mspabi_slli
c03a:	DCC2 0022	BIS.B R12,&Port_1_2_P1DIR
c03e:	531F	INC.W R15
c040:	923F	CMP.W #8,R15
c042:	3BF7	JL (\$C\$L1)
	\$C\$L2:	
c044:	E3F2 0021	INV.B &Port_1_2_P1OUT
c048:	3FFD	JMP (\$C\$L2)



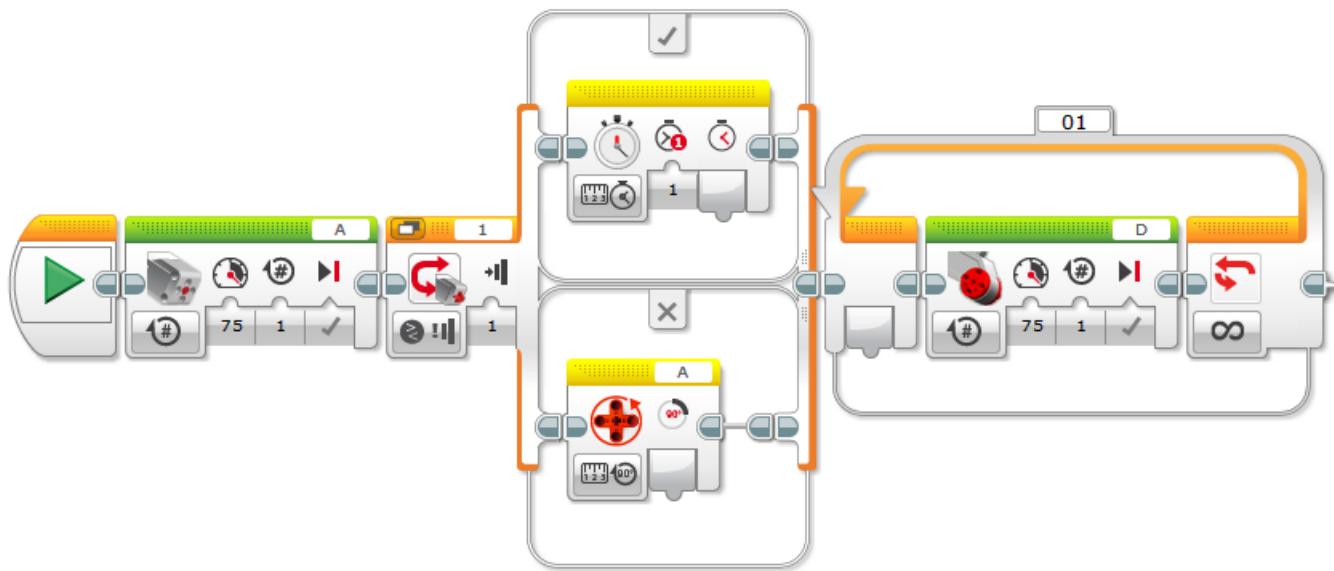
4. Programming

OUTLOOK GRAPHICAL PROGRAMMING

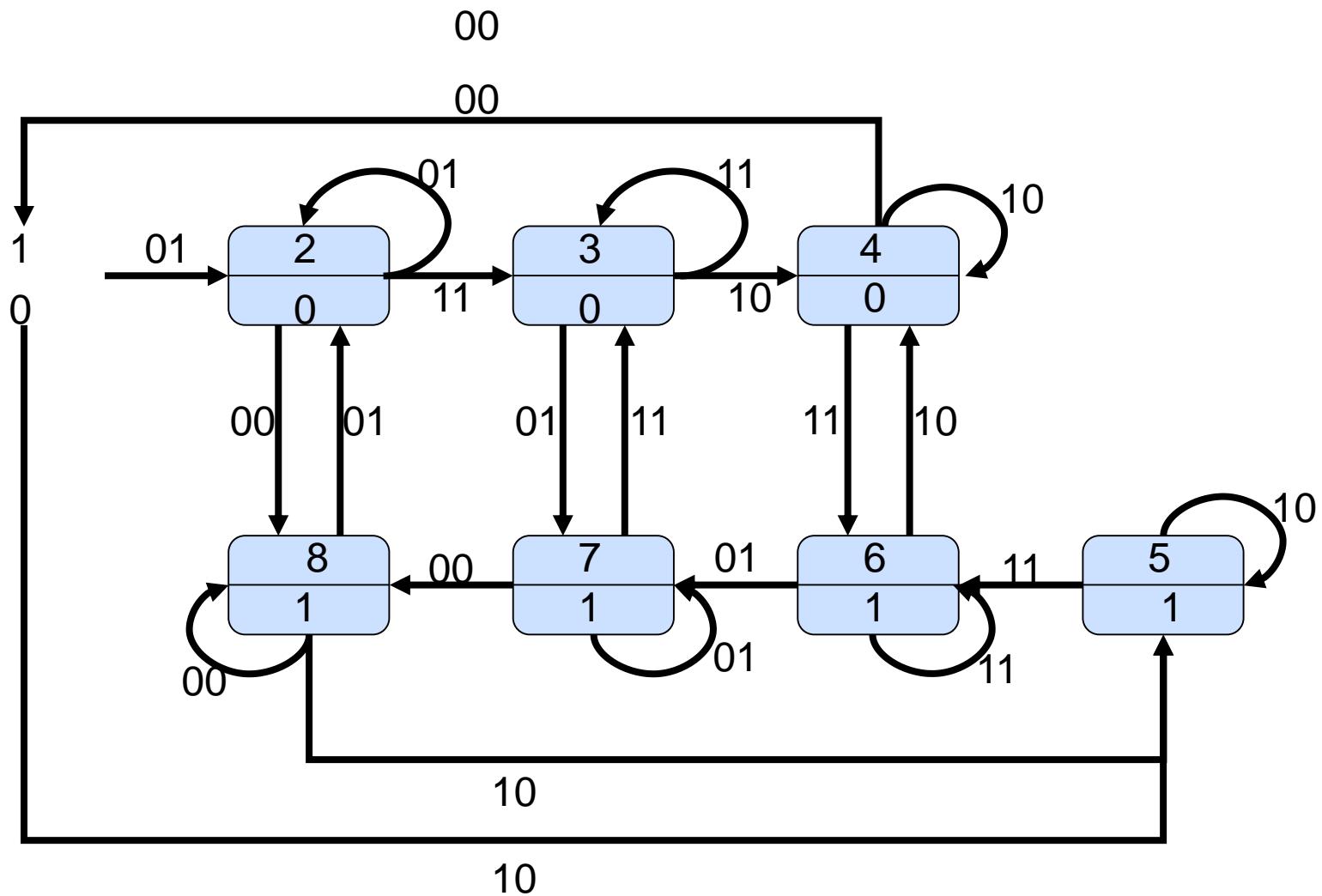
Graphical Programming - LabVIEW



Graphical Programming - LEGO Mindstorms



Graphical programming - Stateflow diagram



Graphical programming

- Advantage
 - Focus on solving the problem (and not on the way to it)
 - Tasks can be edited much faster
 - No training in the language / syntax required
 - No training in the target system necessary
 - Better understanding for laymen
 - Easy debugging possible

- Disadvantage
 - In most cases: no way to access the actual code
 - Only limited optimization possible
 - Often (due to a lack of insight into the complexity of the functions used) much slower and / or larger than manually written code

4. Programming

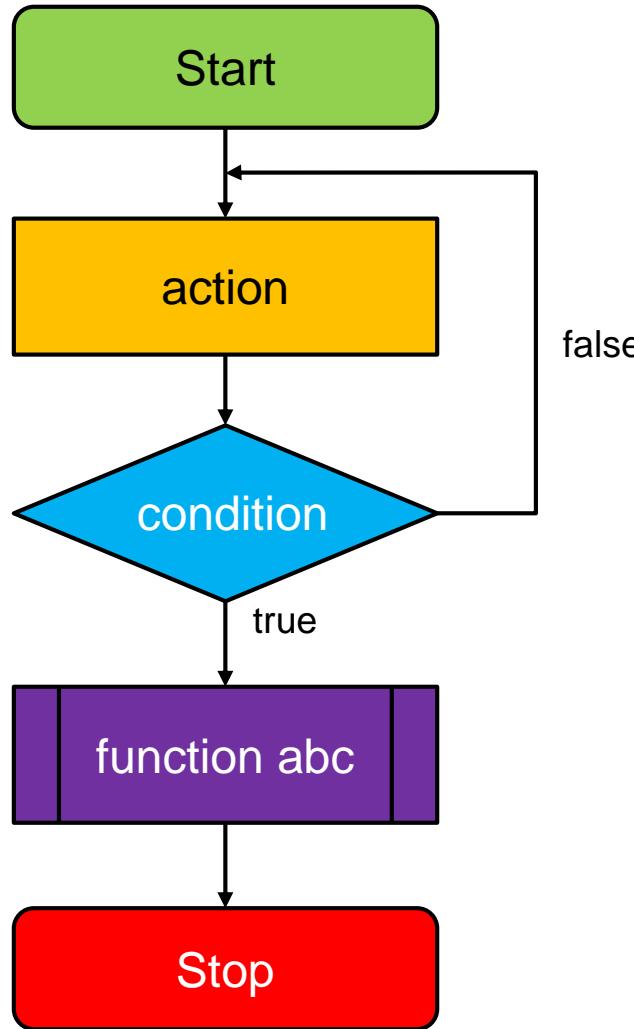
DOCUMENTATION: FLOWCHARTS

Flowcharts

- Documentation / description of the sequence of a program
- Basically arbitrarily exact or arbitrarily abstract possible

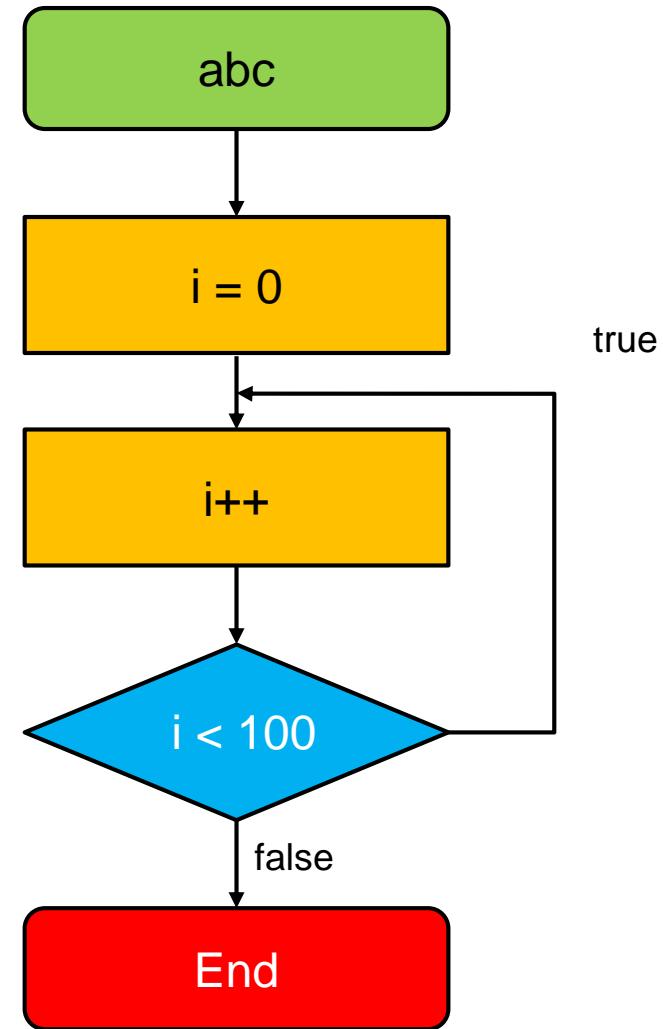
- Five main elements
 - start / stop
 - actions
 - conditions
 - subroutines
 - arrows (indicate the logical sequence)

Flowcharts



false

true

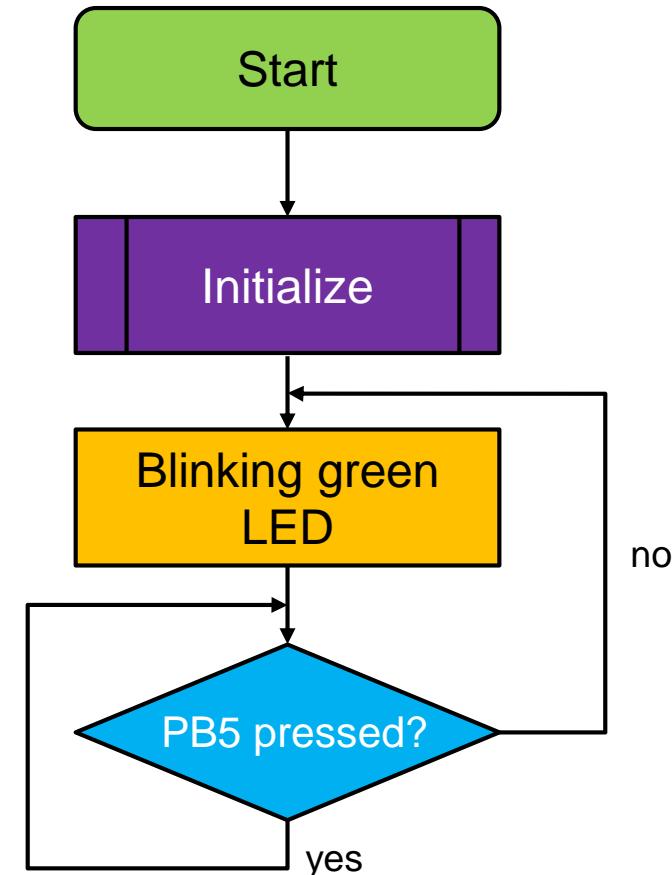


true

Flowcharts

Flowcharts should rather be abstract than detailed.

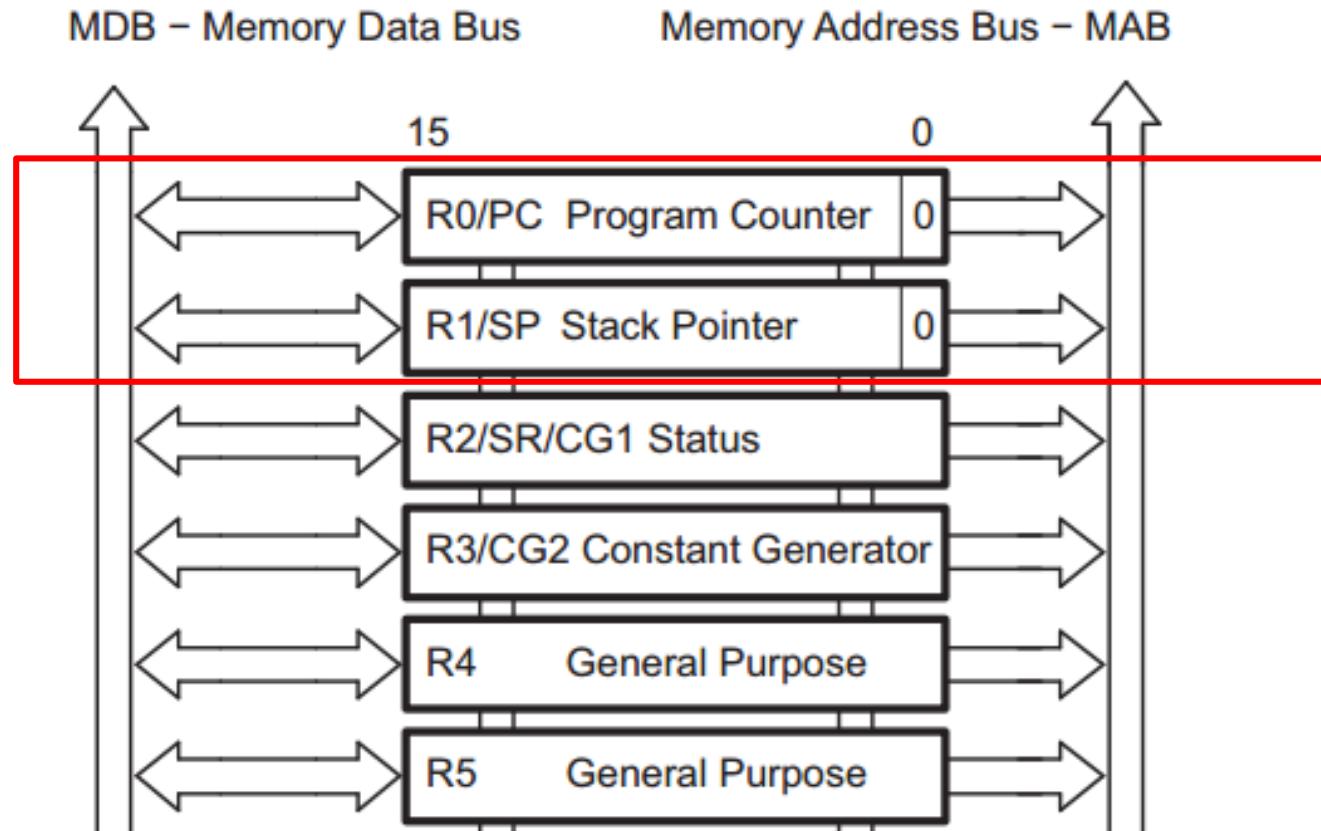
It does not help if each variable is listed individually in the flowchart if the task of the program remains unclear.



4.5. Program Counter, Stack

PROGRAM COUNTER (PC)

Reminder: Register in the MSP430



Source: MSP430x2xx Family – User's Guide

Program Counter (PC)

- Contains the address of the next command to be executed
- At the beginning: 0xC000
- Each command: automatically increased by 2
 - (Word ↔ Byte)
- Manual changes possible
 - Either relative (+4, -20, ... - especially with JMP commands)
 - Or absolute (= 0xCAFE - by means of MOV commands, for example))

4.5. Program Counter, Stack **STACK**

Stack

- „pushdown storage“, Last-In-First-Out (LIFO)
- start address: 0x3FE
- Dynamic memory
- PUSH / POP
 - PUSH: Set the variable to the stack, SP -2
 - POP: Move the variable from the stack, SP +2
- CALL
 - Place the current PC on the stack (SP -2)
 - PC = Address of the called function
- RET
 - PC = Current value on the stack
 - SP +2

Stack – Example

t_0	t_1	t_2	t_3
R12 = AFFE	R12 = AFFE	R12 = AFFE	R12 = AFFE
R13 = 7353	R13 = 7353	R13 = 7353	R13 = 7353
SP = 3FE	SP = 3FC	SP = 3FA	SP = 3F8
PC = C0FE	PC = C100	PC = C102	PC = F000
3FA xxxx	3FA xxxx	3FA xxxx	3FA C104
3FC xxxx	3FC xxxx	3FC 7353	3FC 7353
3FE xxxx	3FE AFFE	3FE AFFE	3FE AFFE
...
C0FE PUSH R12	C0FE PUSH R12	C0FE PUSH R12	C0FE PUSH R12
C100 PUSH R13	C100 PUSH R13	C100 PUSH R13	C100 PUSH R13
C102 CALL 0xF000	C102 CALL 0xF000	C102 CALL 0xF000	C102 CALL 0xF000
C104 POP R12	C104 POP R12	C104 POP R12	C104 POP R12
C106 POP R13	C106 POP R13	C106 POP R13	C106 POP R13
...
F000 RET	F000 RET	F000 RET	F000 RET
...

Stack – Example

t_3	t_4	t_5	t_6
R12 = AFFE	R12 = AFFE	R12 = 7353	R12 = 7353
R13 = 7353	R13 = 7353	R13 = 7353	R13 = AFFE
SP = 3F8	SP = 3FA	SP = 3FC	SP = 3FE
PC = F000	PC = C104	PC = C106	PC = C108
3FA C104	3FA C104	3FA C104	3FA C104
3FC 7353	3FC 7353	3FC 7353	3FC 7353
3FE AFFE	3FE AFFE	3FE AFFE	3FE AFFE
...
C0FE PUSH R12	C0FE PUSH R12	C0FE PUSH R12	C0FE PUSH R12
C100 PUSH R13	C100 PUSH R13	C100 PUSH R13	C100 PUSH R13
C102 CALL 0xF000	C102 CALL 0xF000	C102 CALL 0xF000	C102 CALL 0xF000
C104 POP R12	C104 POP R12	C104 POP R12	C104 POP R12
C106 POP R13	C106 POP R13	C106 POP R13	C106 POP R13
...
F000 RET	F000 RET	F000 RET	F000 RET
...

State of the lecture

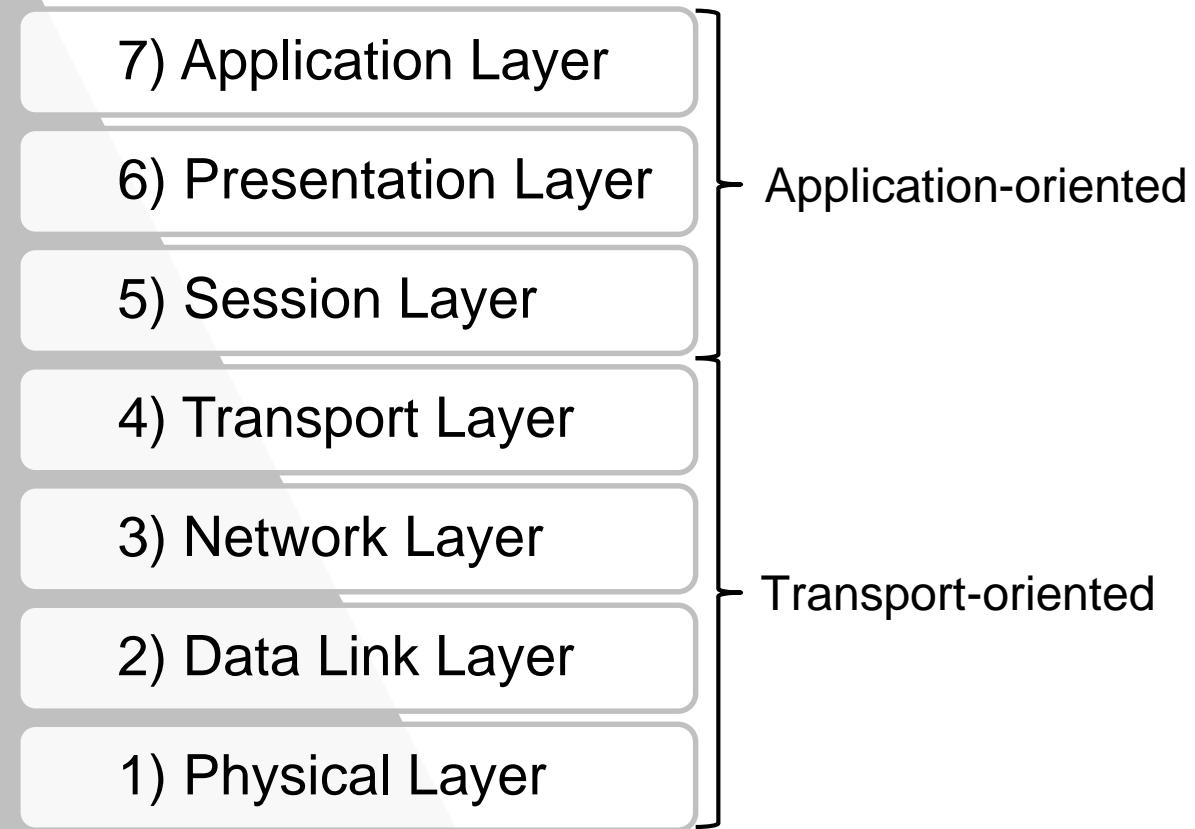
- ✓ Chapter 4: Programming
 - ✓ Assembler, C, Compiler, ...
- Chapter 5: Communication
 - OSI-Model, topologies, UART, I2C, ...
- Chapter 6: Outlook (among others: Safe Systems)

5. Communication

OSI model, topologies, UART, I2C, ...

5. Communication

OSI MODEL



1) Physical Layer

- Transmits the individual symbols (in the simplest case, only ones and zeros).
Example: RS-232, "normal" UART

2) Data Link Layer

- Securing communication (checksums). Example: 802.11

3) Network Layer

- Addressing the participants and switching the actual data (routing).
Example: IP

4) Transport Layer

- Segmentation of the data stream (division of a long message into individual packets), repetition on loss. Example: TCP

5) Session Layer

6) Presentation Layer

7) Application Layer

- application-related; often negligible in microcontroller applications

5. Communication

TOPOLOGIES

Topologies

Topologies:

- Peer to Peer, P2P
- Line
- Ring
- Star
- Bus
- Tree
- Meshed
- Fully meshed

Measured values:

- Maximum distance
(diameter) (smaller is better)
- Failure safety
(connectivity) (larger is better)
- Connection effort
(degree) (less is better)
- Data throughput
(Bisection bandwidth)
(Number of paths that need to be separated to split a network of N nodes into two networks with $N / 2$ nodes)
(Bigger is better)

Topologies: Peer to peer (P2P)

- Diameter: 1
- Connectivity: 1
- Degree: 1
- Bisection bandwidth: 1

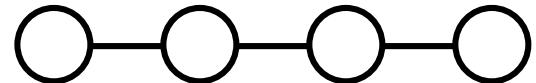
- Advantages
 - Direct connection with maximum data throughput possible.
 - Fault by other participants not possible (since nonexistent).
- Disadvantages
 - Only two participants



Topologies: line

- Diameter: $N-1$
- Connectivity: 1
- Degree: 2
- Bisection bandwidth: 1

- Advantages
 - Small number of connections
 - Easy Routing
- Disadvantages
 - Low throughput
 - No redundancy
 - All nodes must be active for communication to work

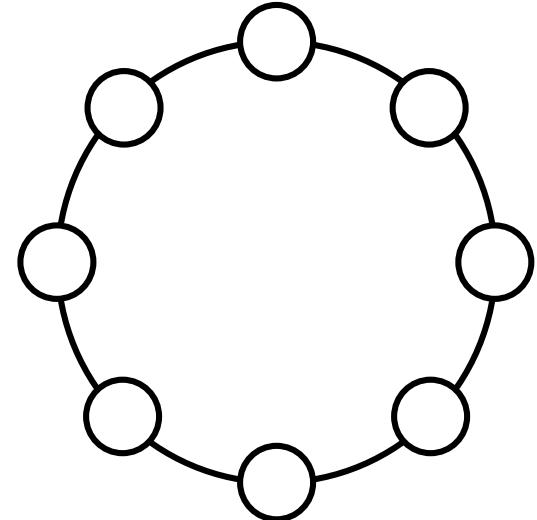


Topologies: Ring

- Diameter: $N/2$
- Connectivity: 2
- Degree: 2
- Bisection bandwidth: 2

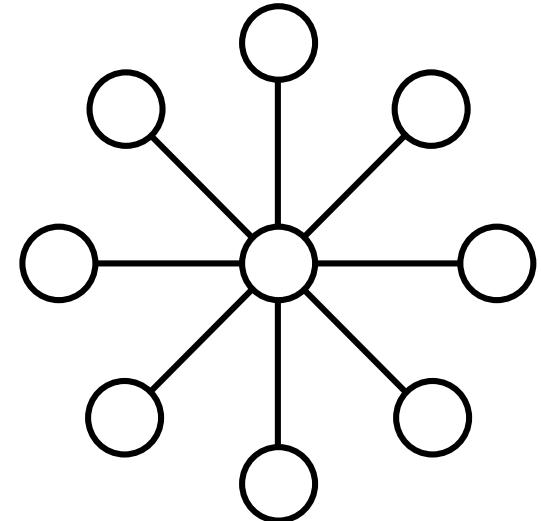
- Advantages
 - Redundancy (but only slightly)
 - Small number of connections
 - Easy Routing

- Disadvantages
 - Very low throughput
 - All nodes must be active for communication to work



Topologies: Star

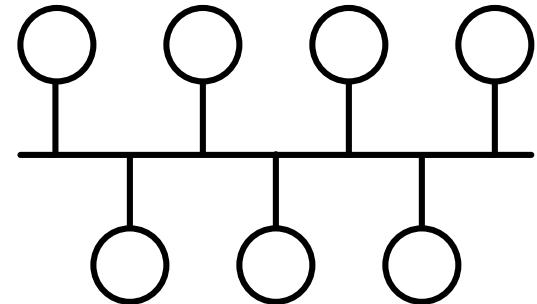
- Diameter: 2
- Connectivity: 1
- Degree: $N-1$
- Bisection bandwidth: $N/2$
- Advantages
 - Only one node must be permanently active
 - High data throughput (if the star center is sufficiently powerful)
 - Different protocols possible
- Disadvantages
 - Small number of connections
 - Failure of the center leads to total failure



Topologies: Bus

- Diameter: 1
- Connectivity: 1
- Degree: 1
- Bisection bandwidth: 1

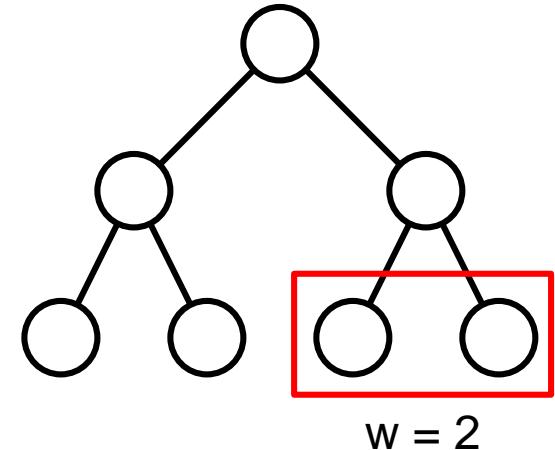
- Advantages
 - Low connection effort
 - No routing required
 - Nodes can lock in and out at any time
- Disadvantages
 - Protocol needed, who can write and when (e.g. CSMA / CD)
 - Misbehavior (or failure) can lead to total failure



Topologies: Tree

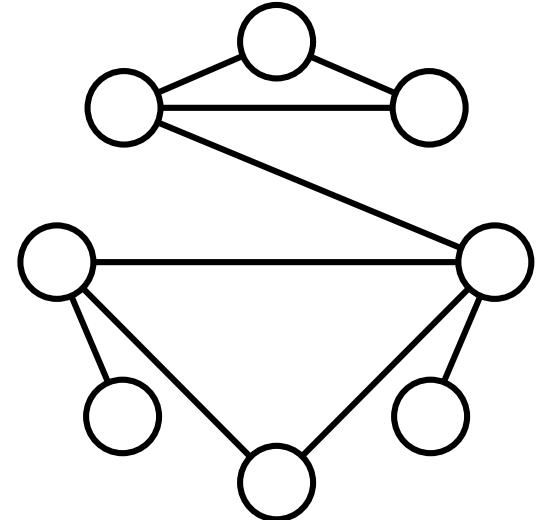
- Diameter: typically $2 \log_w(N - 1)$
(w = Number of Children)
- Connectivity: 1
- Degree: $w + 1$
- Bisection bandwidth: 1

- Advantages
 - In the case of particularly hierarchical networks: optimum ratio of cabling complexity to data throughput at great distances
 - Different protocols possible
- Disadvantages
 - If the network is not hierarchically structured, the data throughput is poor
 - No redundancy



Topologies: Meshed

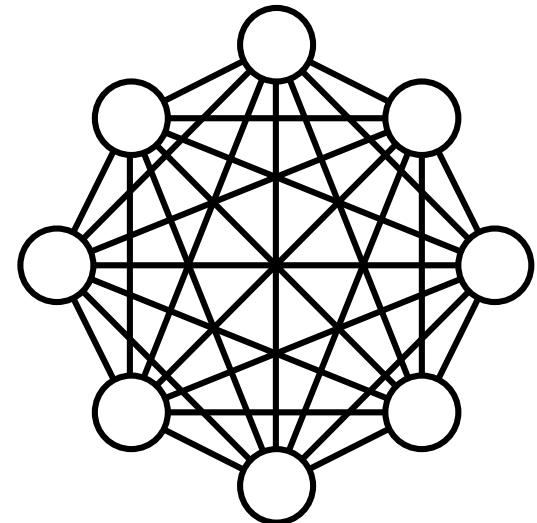
- Diameter: 1 .. N-1
- Connectivity: 1
- Degree: 1 .. N-1
- Bisection bandwidth: 1 .. N-1
- Advantages (with good planning)
 - Many redundancies
 - High data throughput
 - The ratio of failure safety to connection complexity can be chosen differently according to the situation



Topologies: fully meshed

- Diameter: 1
- Connectivity: $N-1$
- Degree: $N-1$
- Bisection bandwidth: $(N/2)^2$

- Advantages
 - Many redundancies
 - High data throughput
- Disadvantages
 - High connection effort



5. Communication

DEFINITIONS

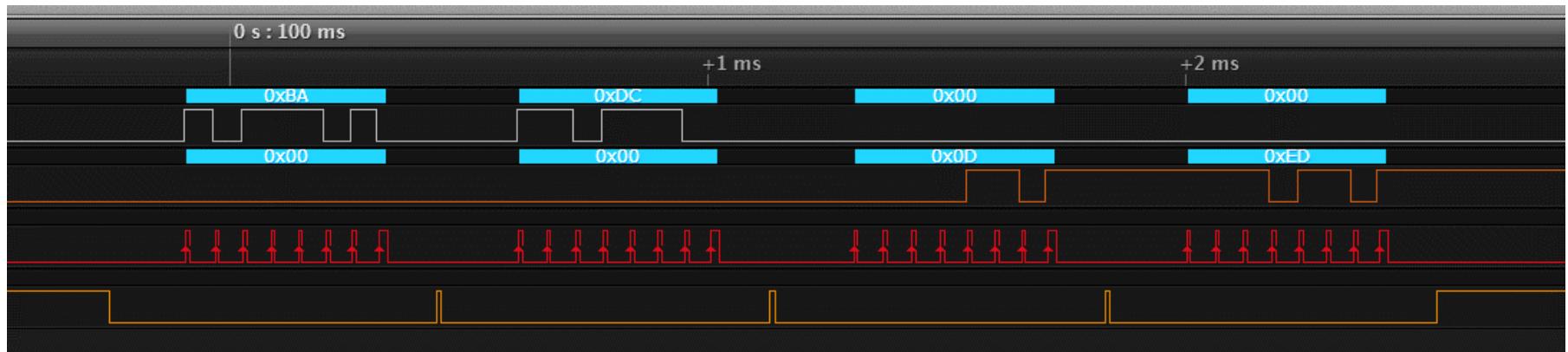
Definitions for (hardware) communication

- Asynchronous communication
 - Communication without a dedicated clock signal. Either external synchronization of the clock or using an embedded clock signal (e.g. Manchester code)
- Synchronous communication
 - Communication using a dedicated clock signal
- Parallel communication
 - Multiple lines, bits are transmitted simultaneously
- Serial communication
 - One line, bits are transmitted one after the other
- ! Not always clearly separable, mixed forms exist

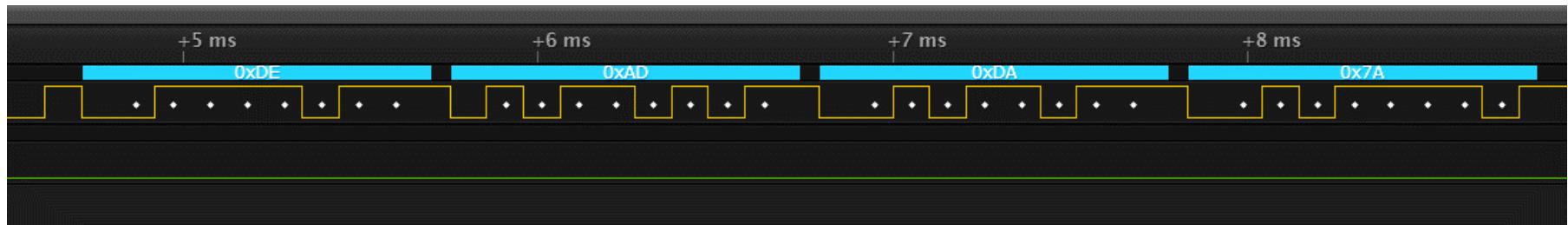
MSB first: Most Significant Bit

LSB first: Least Significant Bit

Example



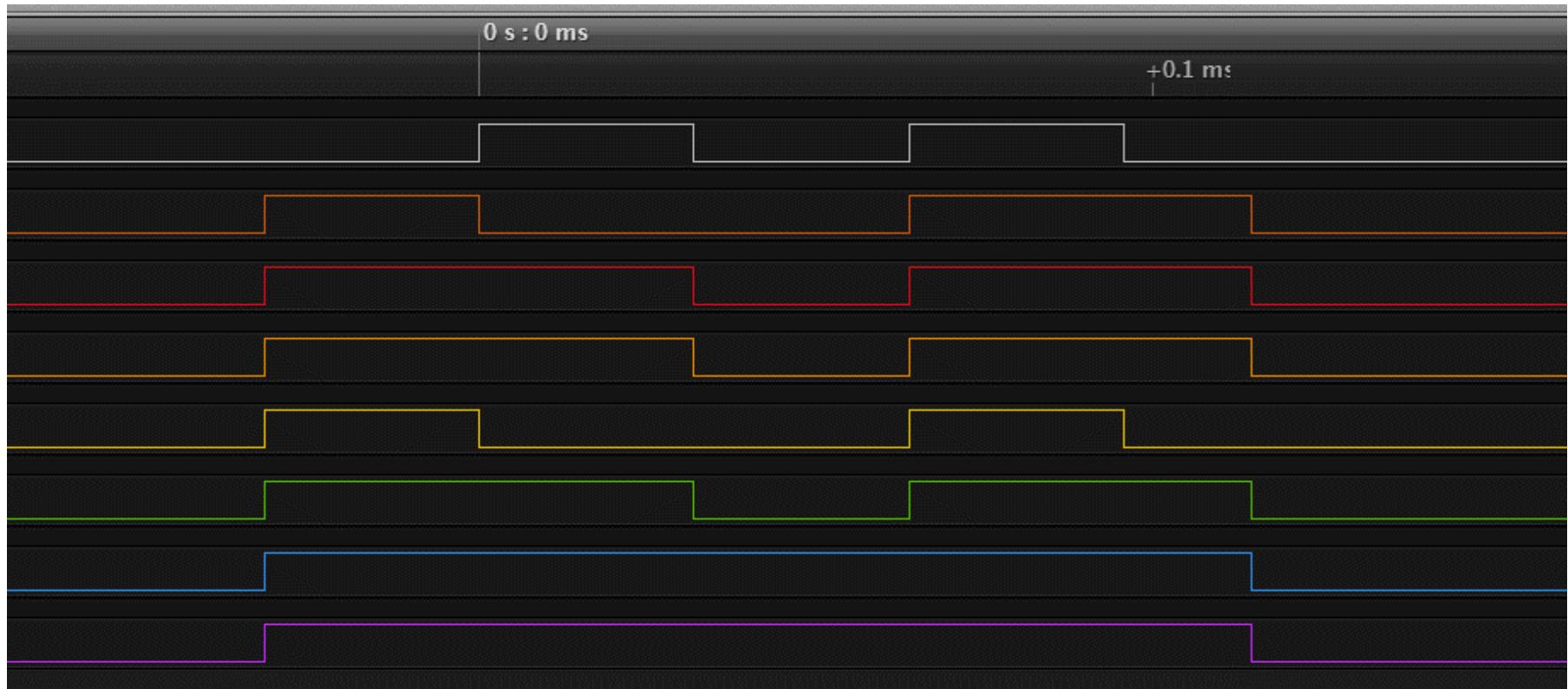
Example



Example



Examples

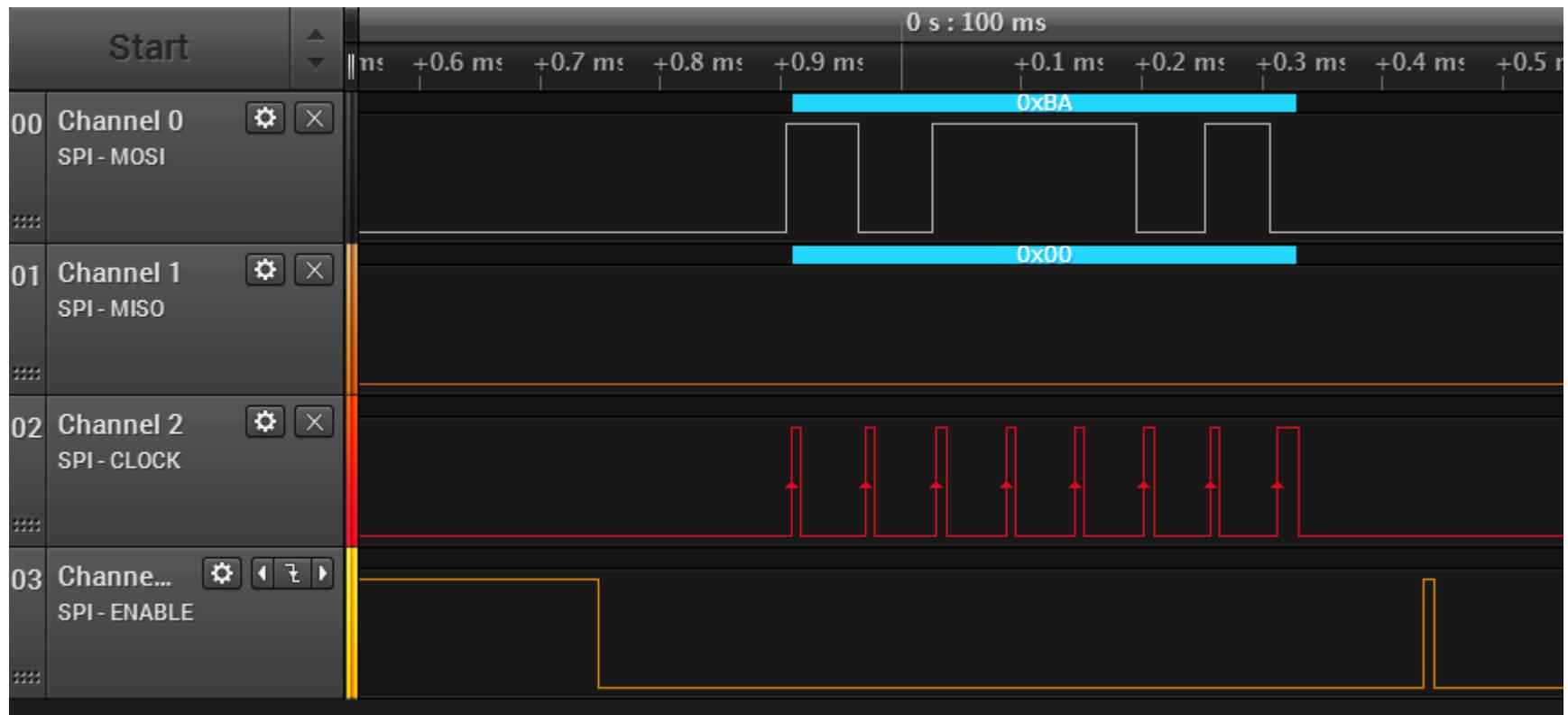


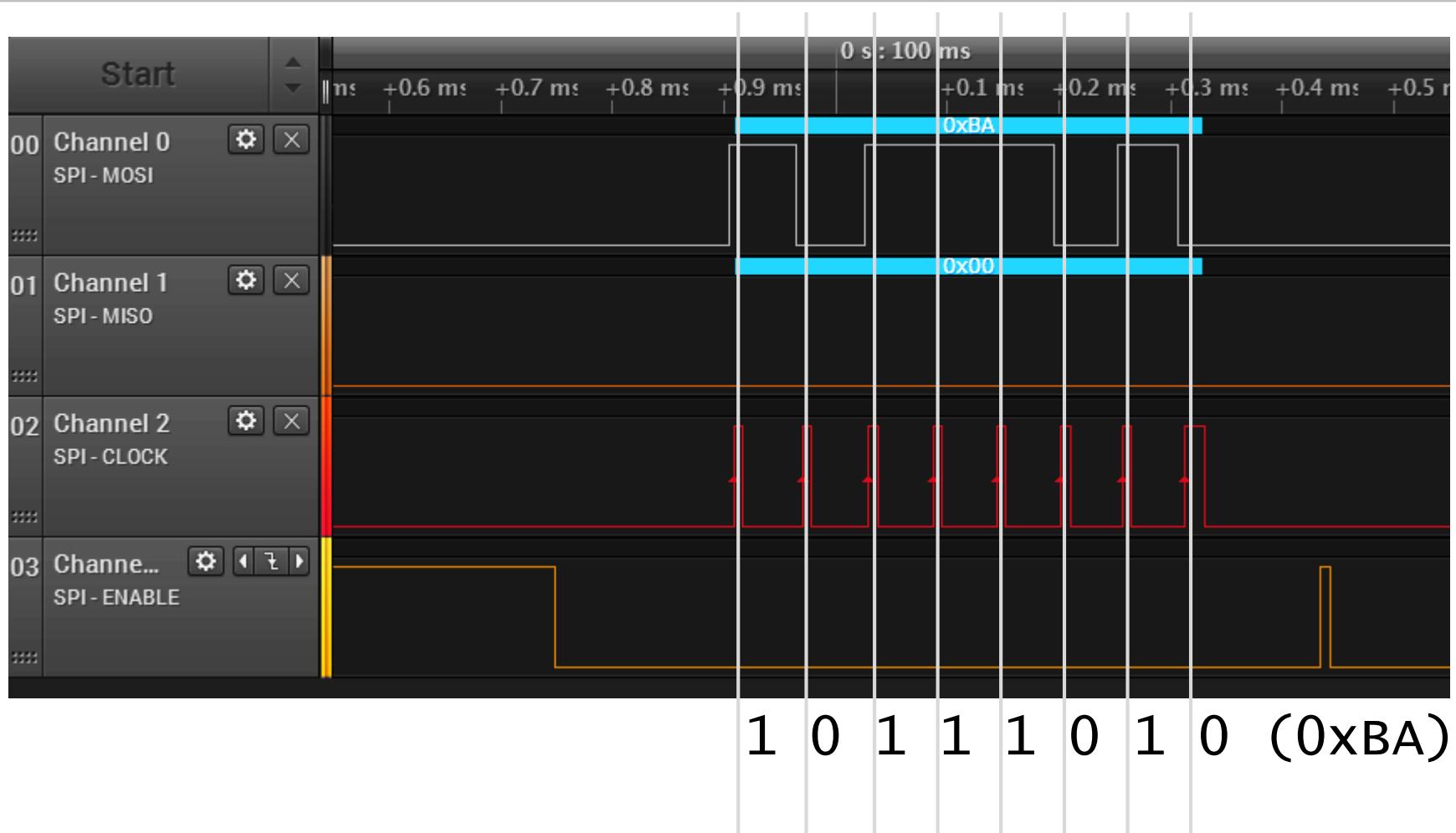
5. Communication

SPI

Serial Peripheral Interface

- Synchronous
- Serial (parallel elements can be added for higher speeds)
- Four Lines
 - MOSI (Master Out Slave In)
 - MISO (Master In Slave Out)
 - CLK (Clock)
 - CS (Chip select – One line per slave)
 - MISO and CS are partially saved (if unnecessary)
- Similar to UART, the actual physical rules are partly covered by the standard.
- Layer 1





5. Communication

UART

Universal Asynchronous Receiver Transmitter

- Asynchronous
- Serial
- Two Lines: TX (transmit), RX (receive)
- UART only defines the logical sequence.
- Physical signals are defined by the specific interface (e.g. RS232)
- Using an optional parity, the UART layer fulfills 2 tasks within the OSI model

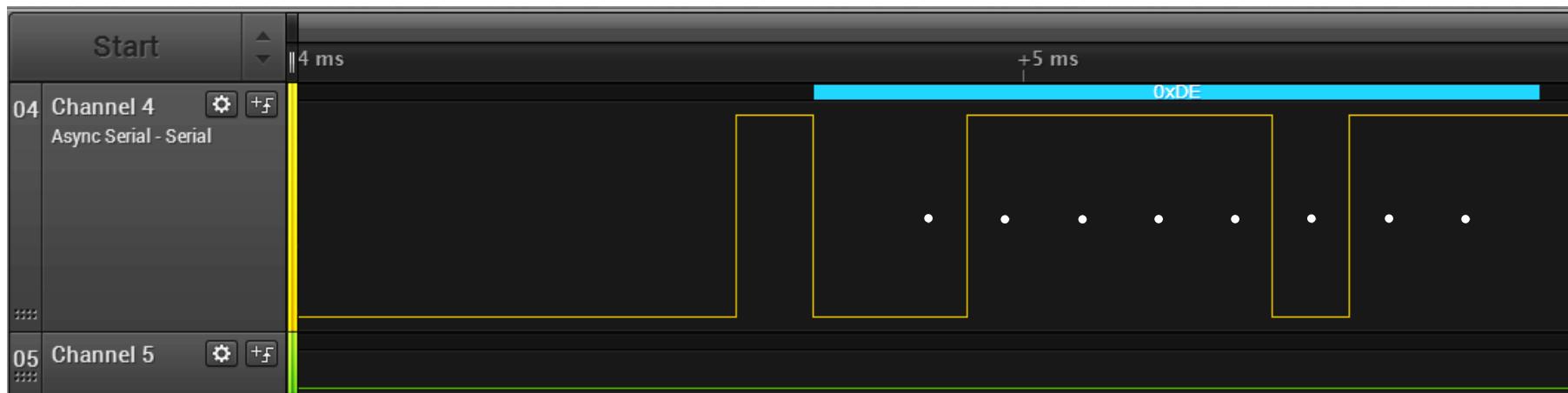
- Sequence
(generally)

- Idle signal on the line (High)
- Startbit (Low)
- LSB
- Bit 1
- Bit 2
- Bit 3
- Bit 4
- Bit 5
- Bit 6
- MSB
- Parity (Odd / Even) (optional)
- Stop Bit (High, At least one 1)

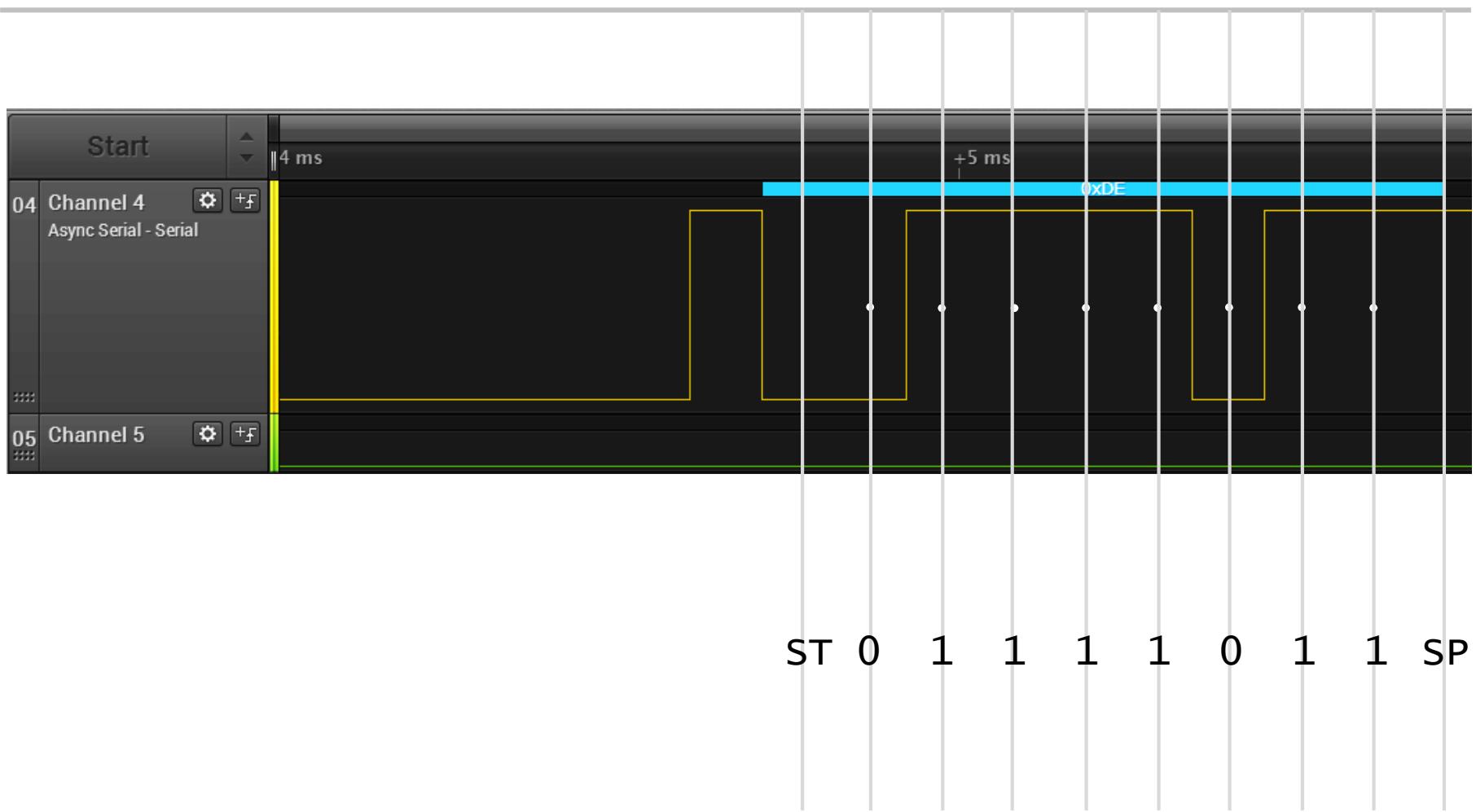
- Sequence 8-N-1
(8 Bits, No Parity, 1 Stop)

- Idle signal on the line (High)
- Startbit (Low)
- LSB
- Bit 1
- Bit 2
- Bit 3
- Bit 4
- Bit 5
- Bit 6
- MSB
- Stop Bit (High)

UART – Signal



UART – Signal



UART: Odd & Even

- Sum of all bits and the parity bit must be "odd" (odd) or "even" (even)

- Example:

0x`13` transmitted using 8-E-1:

111110`1`100`1`000`1`11111

- 0x`97` transmitted using 8-O-2:

111110`1`110`1`00`1`011111

UART: Baud rate

- symbol rate
 - How many different symbols can be transmitted per second
 - In the simplest case (ones and zeros) it is equivalent to the bit rate.
 - In more complex cases (for example, with QAM), still proportional.
-
- Side note:
QAM: quadrature amplitude modulation
 - Symbol selection by both changing the phase and the amplitude of a signal
 - Two-dimensional plane
 - Some radio standards support over 256 different symbols

Outlook: RS-232

- Uses UART
- Works (usually) with 12V-level
- Inverse voltage level
 - Low = 12 V
 - High = -12V

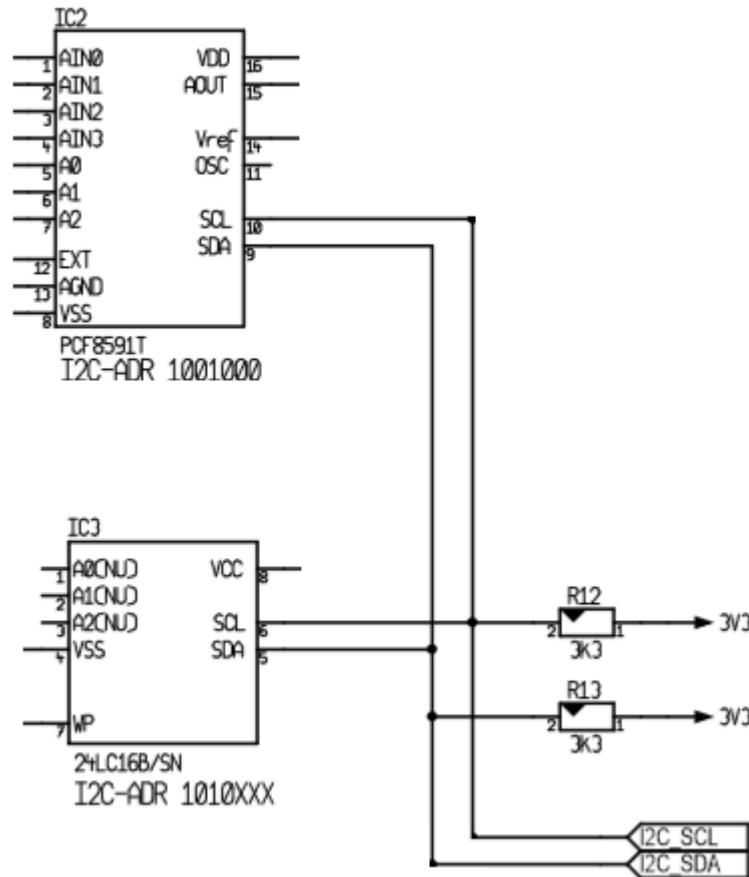
5. Communication

I²C

Inter-Integrated Circuit

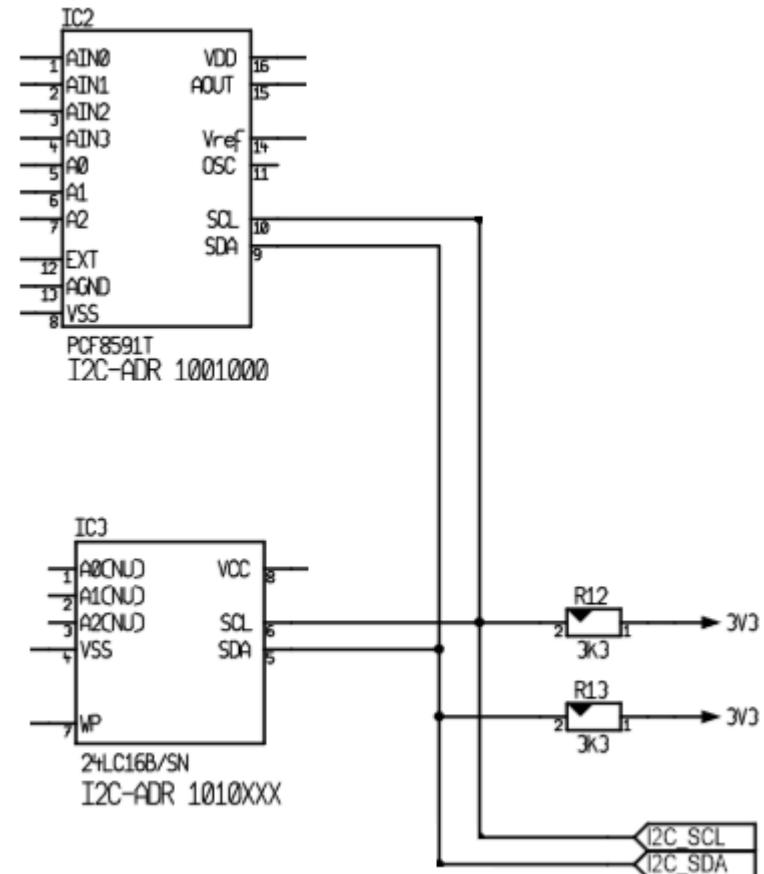
- Synchronous communication
- Serial
- Two lines
 - SCL (Clock)
 - SDA (Data)
- I²C defines the logical and partly the physical process
- Introduces addresses and therefore implements layer 1, 2 and 3
 - However, layer 2 is simply ignored (i.e., no parity)

I²C – bus structure

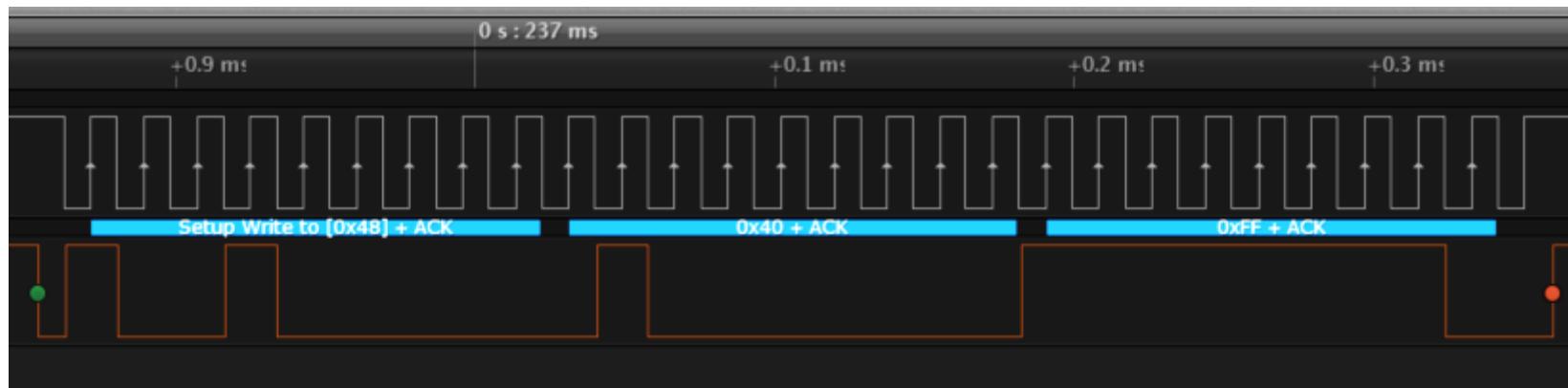


I²C – bus structure

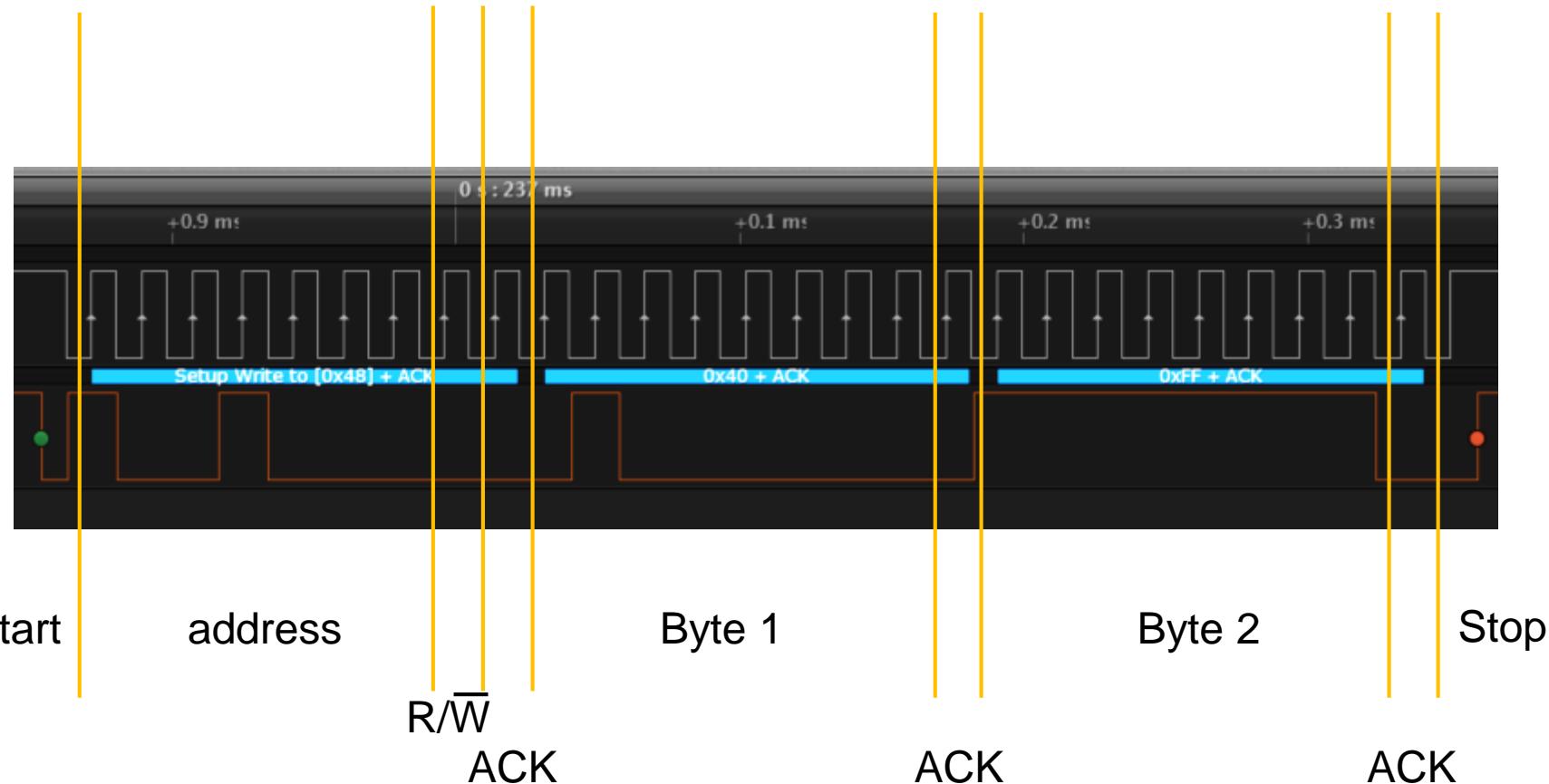
- 7 bit addresses
- pull-up resistors
 - Participants do not write an “1”
 - Only “0” or “High Z”
 - If “High Z” is set by all participants, then the pull-up resistors pull the line to VCC
 - Advantage: no shortcut possible, even when more than one participant is writing simultaneously
- Idle state: SDA high, SCL high
- standard mode: 100 kHz , using extensions up to 3.4 MHz



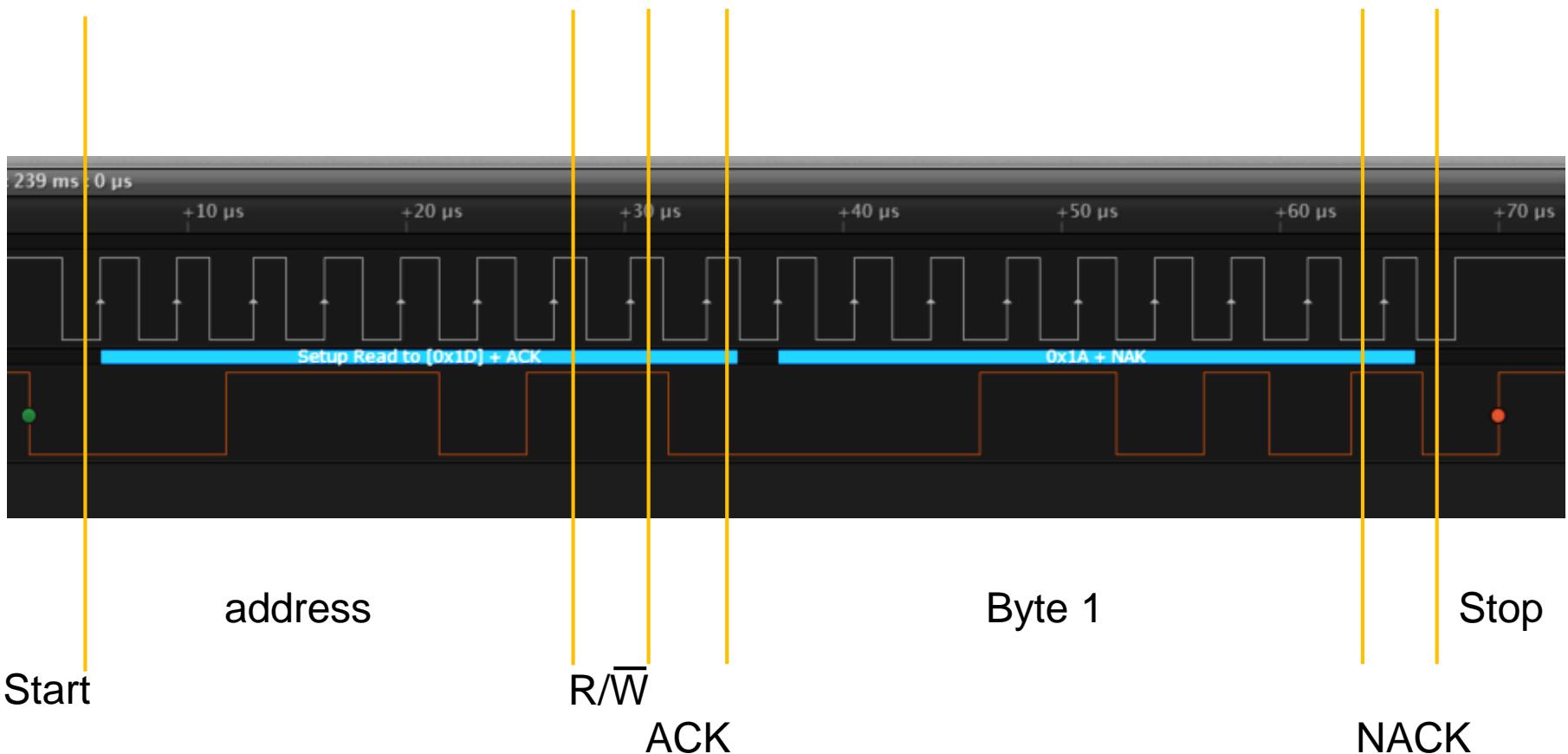
I²C – Signal



I²C – writing



I²C - reading



I²C – Start (ST) / Stop (SP)

- Communication is initiated by START
 - If several masters write simultaneously: bus arbitration
- 1st byte: address
 - Acknowledgment (ACK) or abort / error (NACK)
 - Abort usually results in complete termination of the transmission by sending STOP
- n data byte
 - Confirmation or abort (STOP)
- Finally, either STOP or "Repeated START"
 - STOP: line is enabled, other master can write
 - "Repeated START": Line is not enabled, current master can perform an additional requests (e.g., when reading a particular register: Master writes 1 byte (register address), Repeated START, then reads the contents of this register,
 - in the case of an interim release, another master could disrupt this procedure)

I²C – (N)ACK

- The reading subscriber (i.e., for a write operation the slave, for a read operation the master) confirms the (correct) reception of the data with an ACK
- If there was an error or if the transfer should be terminated, a NACK is sent
- If the slave needs more time to process a received byte, it pulls the SCL line LOW; as the master does not write the line actively, it just waits until the slave pulls the line to HIGH.

5. Communication

BUS ARBITRATION

Bus Arbitration

- For multi-master protocols:
Ensure that no more than one master is active at the same time
- Procedure: First read what has just been written
- In the case of I²C:
 - Master monitors whether the line actually changes from LOW to HIGH while releasing the line
 - If not → It has lost the arbitration and must wait until line is clear again

General approach

- CSMA/CA (Collision Avoidance)
 - No active detection of a collision; the detection takes place only due to absence of the response (especially used for connections in which a simultaneous reading and writing is not feasible - for example in a radio communication)
- CSMA/CD – Carrier Sense Multiple Access / Collision Detection
 - Listen if someone is writing (CSMA)
 - If not, the system can write
 - If another system is writing at the same time, abort communication and try again after a random time interval
- CSMA/CR (Collision Resolution)
 - Similar to CD, however, not both systems abort communication, but only the one with a lower priority (prioritization by the address)

5. Communication

OUTLOOK: IMPROVING IMMUNITY TO INTERFERENCE / LVDS

Improving immunity to interference

Problem: Interferences on the line, for example due to EMF

Solutions (I):

- Decrease speed → low-pass filtering → faults can be detected easier
 - Problem: Speed ↓
- Increase the signal voltage → interfering signals have smaller effect
 - But: Lines themselves now produce a larger EMF, which causes higher interference on the lines
 - Large glitches are still a problem
 - Higher voltage jumps are often problematic due to line capacitances (speed ↓)

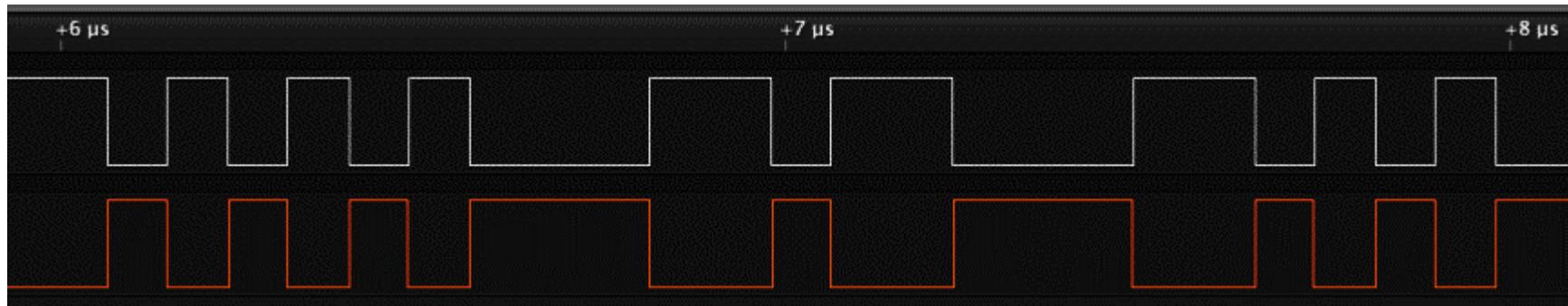
Improving immunity to interference

Solutions (II):

- Shielding the cable
 - "Faraday cage"
- Adapted / dynamic line termination
- Potential-free transmission
- Using differential signals
 - Nowadays especially important as "LVDS" (low voltage differential signal)

Idea:

- Using small voltage changes to keep the speed high
- Using differential signals in which the difference voltage between two lines determines the transmitted bit, but not the potential level of a individual line



Advantage:

- Interferences now affect both signals lines simultaneously
- As a result, the potential of both lines is shifted - but the difference remains the same



State of the lecture

- ✓ Chapter 4: Programming
 - ✓ Assembler, C, Compiler, ...
- ✓ Chapter 5: Communication
 - ✓ OSI-Model, topologies, UART, I2C, ...
- Chapter 6: Outlook (among others: Safe Systems)

6. Outlook

PERFORMANCE BENCHMARK FOR MICROCONTROLLERS

- Clock rate (limited), MIPS (Million Instructions Per Second)
- Benchmarking, application-dependent test (MFLOPS (Million Floating Operations Per Second) / Whetstone MIPS / Dhrystone MIPS)
- Command set, number of clock cycles per instruction, response time to interrupt requests
- Memory size, power consumption, physical size
- Cost
- Existing experience with the microcontroller family in the design group

Clock rate / MIPS

- „Misleading Information to Promote Sales“
- Million machine commands per second
- Integer operations

Problem:

- CISC / RISC
- Commands are diverse complex, therefore only limited significance

Benchmarks

- Standardized tests with the execution time as a performance rating
- Measures not only the CPU performance, but also other peripherals such as the bus and memory speed
- An option is to run the actual program for which the chip is intended
- "Famous" Examples:
 - Whetstone MIPS (floating point numbers)
 - Dhrystone MIPS (Integer operations)
- Special case MFLOPS
 - Specifies the number of floating point operations per second
 - However, on newer processors with a dedicated floating-point unit, it is specified like MIPS; on processors without such a unit, the measurement is a benchmark

Command set, number of clocks per instruction, response time to interrupt requests

- Especially important for embedded systems
- A good suited command set can solve tasks more code efficiently than a generic system
- An apparently slower system (i.e., lower clocked system) can nevertheless be faster than a higher clocked system depending on the number of clocks per instruction
- In the case of interrupt controlled systems, the response time to interrupt requests is very important because the computation time required here is "lost" (can not be used otherwise)

6. Outlook

REAL TIME SYSTEMS

- Classical systems offer no real-time mode, i.e.,
 - Logical correctness
 - But no temporal correctness (no processing in a defined maximum time interval)
- Real-time systems, on the other hand, meet both requirements - they are logically correct (i.e., they always deliver the correct result), but also temporally correct (they deliver the result within a defined, maximum period or within a maximum number of computation steps)
- WCET: Worst Case Execution Time
 - Must be proved (which is very difficult)
 - The entire processing chain must be considered!

- Often low-performance processors are deployed
 - reliability
 - computation time usually of secondary interest
- Complex processor features such as Dynamic Clock Management usually turned off → determinism
- Usually without interrupts (only using polling)

- Real-time systems are classified in:

- Hard real-time systems

Time constraints **must always** be ensured.

Example: ABS braking system, numerical control in production engineering.

- Fixed real-time systems (not recognized by some experts as real-time systems)

Time constraints do not always have to be respected, but the result is then worthless.

Example: Lambda probes in vehicles

- Soft real-time systems (also no "real" real-time)

Time constraints are more guidelines than fixed bounds, exceeding tolerable.

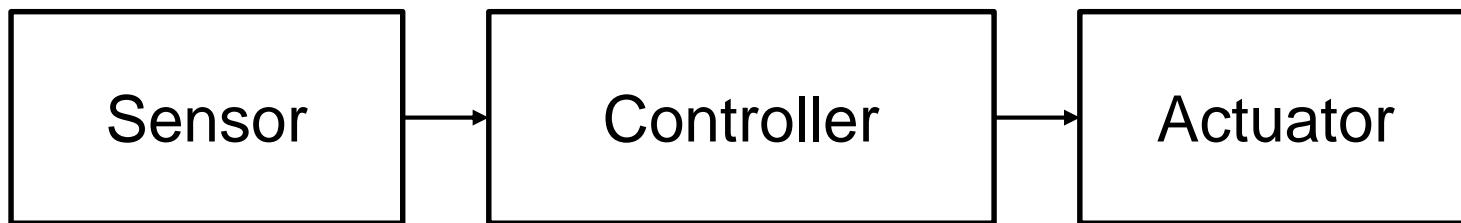
Example: video conferencing system, home thermostat (buildings)

6. Outlook

SAFE SYSTEMS

What is a safe system?

- Safe Systems run under all circumstances
- Safe systems must not be disturbed by uncontrolled marginal incidents
→ formal verification is often necessary



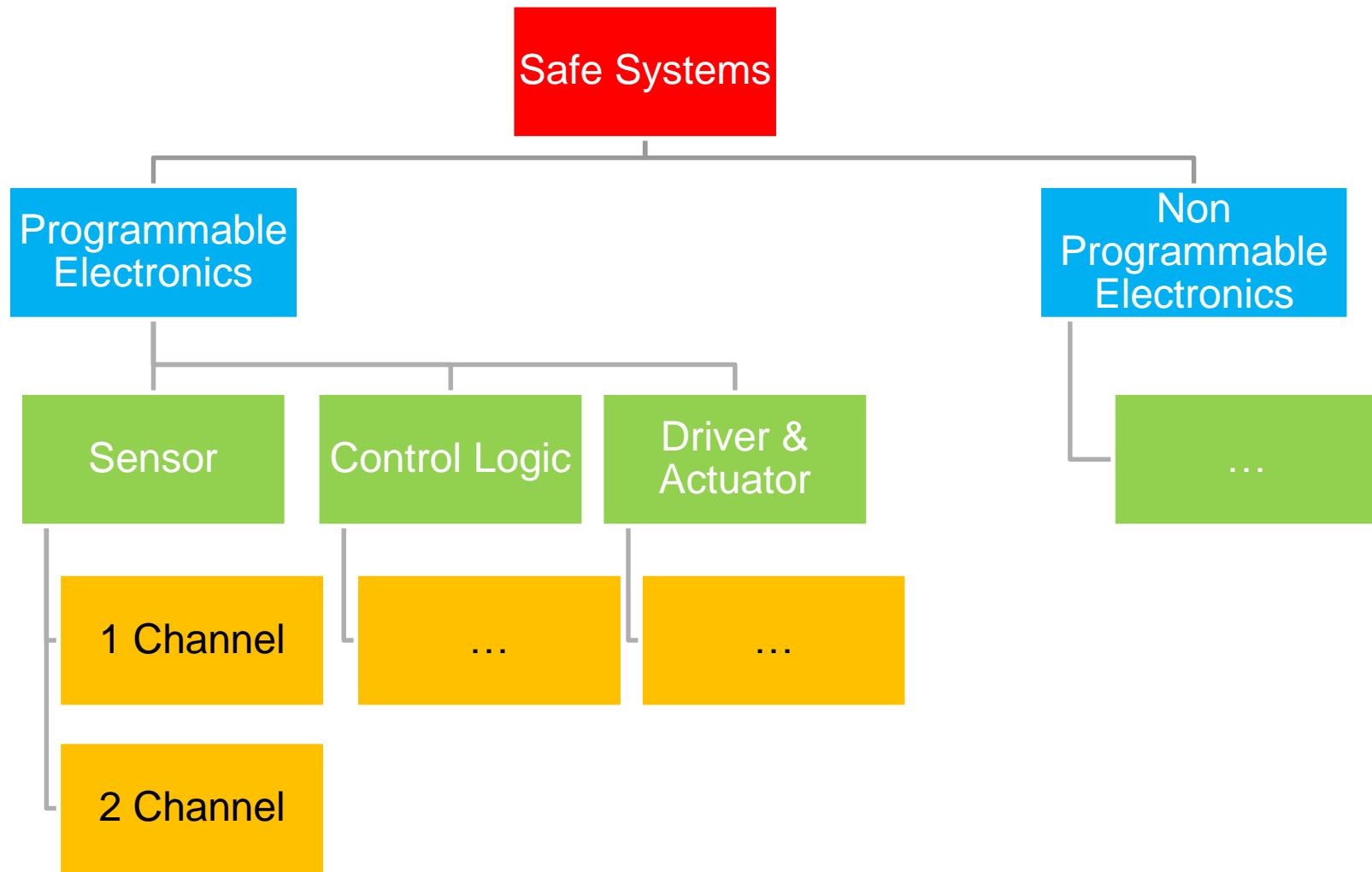
<http://www.scope-online.de/>



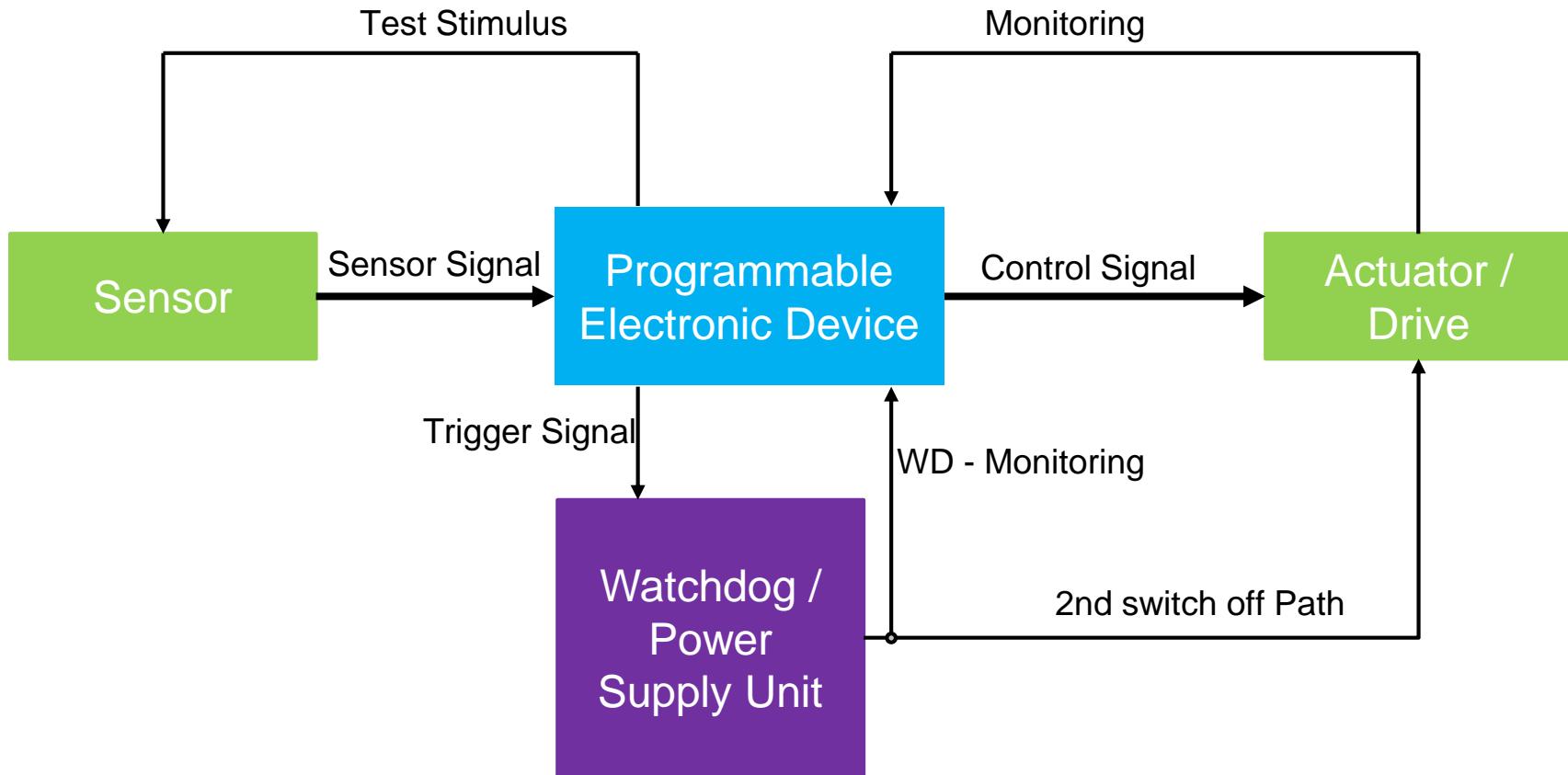
<http://bosch-mobility-solutions.de>



<http://www.pollin.de>



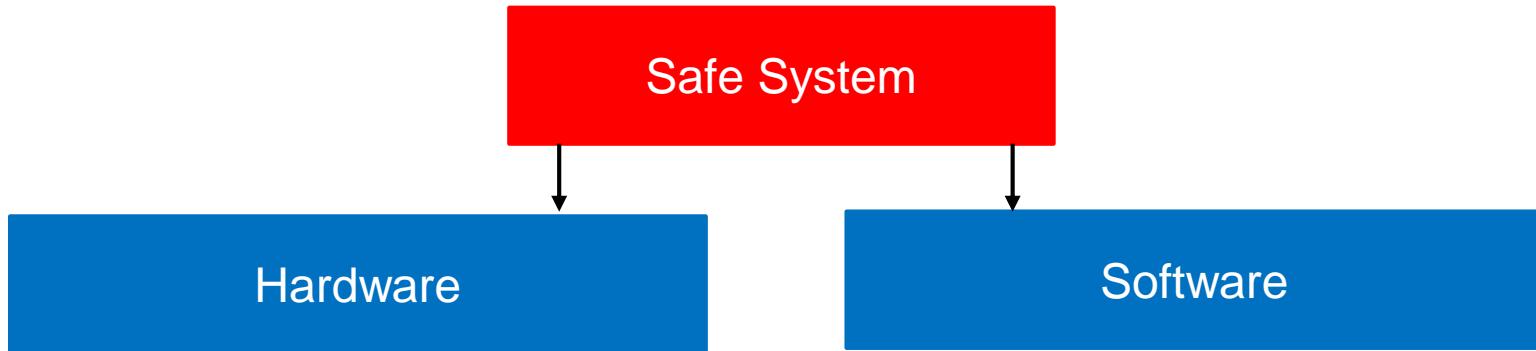
Example: Single Channel Safe System



Source: STSARCES, Annex 6

Safe System Applications

- Airplanes
 - Fly-by-Wire (Airplane balance & rudders control)
 - Airplane Control Unit
 - 3 up to 5 redundant control unit channels
- Trains
 - Train Control Unit
 - Rail Control Centre
- Automotive
 - Drive-by-Wire
 - Brake-by-Wire
- Industry
 - Cutter Machines
 - Hydraulic Presses



- The Hardware must not fail completely
- It must bring the system into safe state

- The Software must not stop or hang
- Common SW failures are not acceptable
(Out of memory, Stack over-/underflow, ...etc)
- It must bring the system into safe state

Failure Rate in Time (FIT)

- It is the amount of failed pieces of certain components in a given period of time
- Measurement unit: PPM (Part Per Million), FIT
- The FIT number and the failure modes are given by the manufacturer

Safe System Standards

- EN 954 (-1996) (Industrial)
- IEC 61508 (International Electrotechnical Commission - CEN)
(1998 - Active) (Industry)
- EN ISO 13849 (2009 - Active) (Industry)

- IEC 26262 (Automotive)
- IEC 61513 (Nuclear Power Plants)
- IEC 62279 (Rail Software)

PFD (Probability of Fail on Demand)

Safety integrity level	Low demand mode of operation (Average probability of failure to perform its design function on demand)
4	$\geq 10^{-5}$ to $< 10^{-4}$
3	$\geq 10^{-4}$ to $< 10^{-3}$
2	$\geq 10^{-3}$ to $< 10^{-2}$
1	$\geq 10^{-2}$ to $< 10^{-1}$

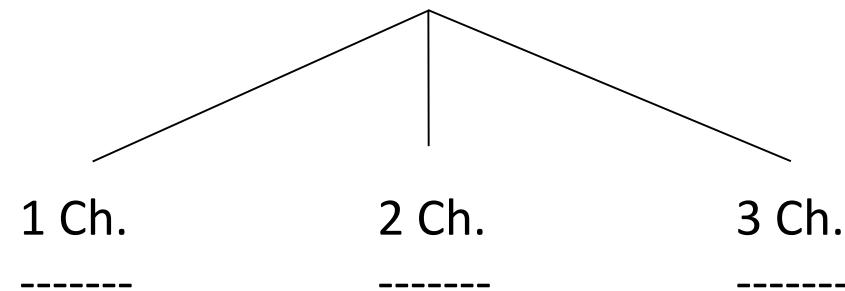
PFH (Probability of Fail per Hour)

Safety integrity level	High demand or continuous mode of operation (Probability of a dangerous failure per hour)
4	$\geq 10^{-9}$ to $< 10^{-8}$
3	$\geq 10^{-8}$ to $< 10^{-7}$
2	$\geq 10^{-7}$ to $< 10^{-6}$
1	$\geq 10^{-6}$ to $< 10^{-5}$

(Programmable Electronics) PE

	1 Ch.	2 Ch.	3 Ch.
-----	-----	-----	-----
Hardware Fault Tolerance (HFT)	0	1	2
Safe Failure Fraction (SFF)	$\geq 99\%$	90-99 %	60-90%
Diagnostic Coverage (DC)	$\geq 99\%$	90-99%	60-90%

(Programmable Electronics) PE

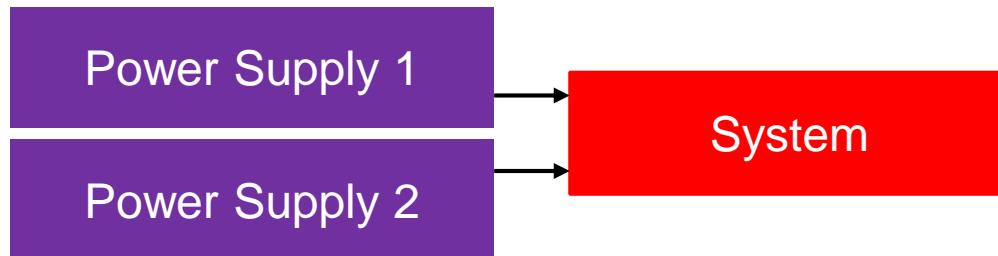


Hardware Fault Tolerance (HFT)	0	1	2
Safe Failure Fraction (SFF)	$\geq 99\%$	90-99 %	60-90%
Diagnostic Coverage (DC)	$\geq 99\%$	90-99%	60-90%

Planning & developing a Safe System

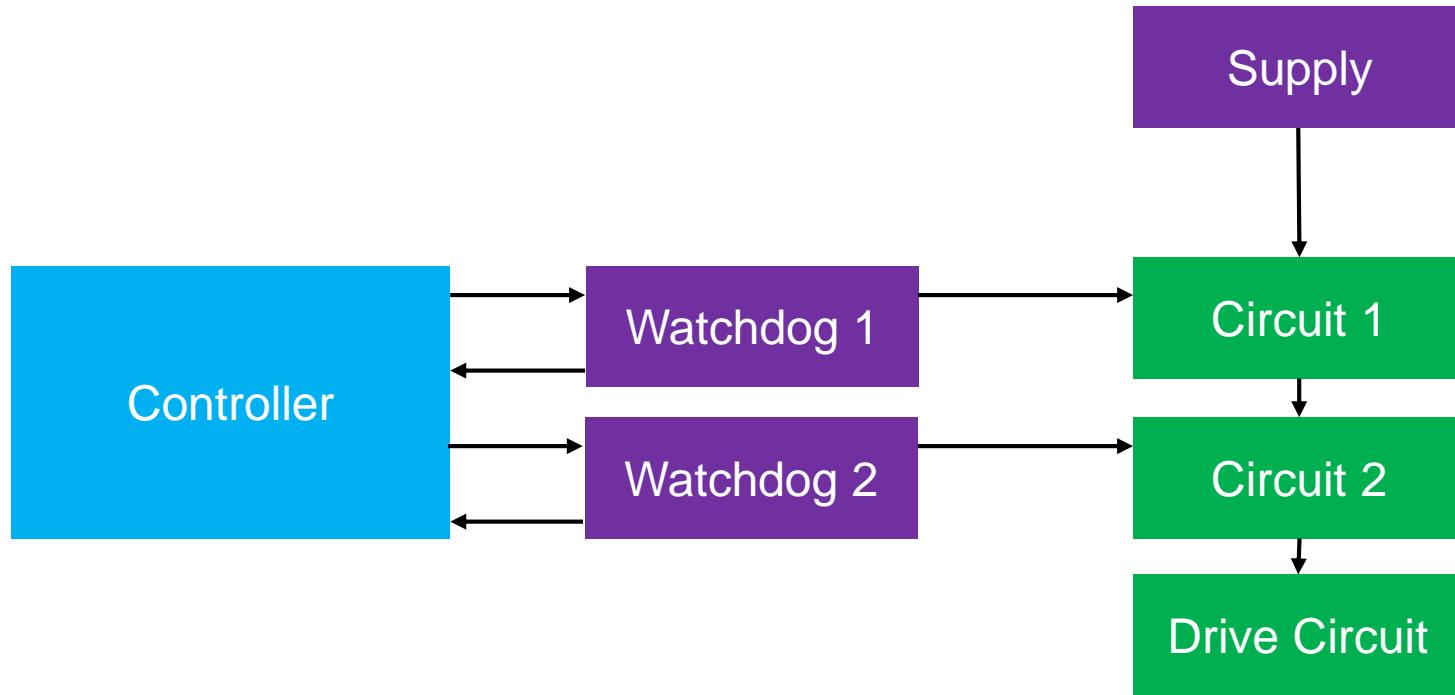
- Documentation
- Define the Safety Requirements
(Which parts in the system should be fail-safe)
- Define Hazards
(Type of hazards that could happen (Sensor / Actuator / Control unit fails))
- Hardware design according to the safety level
- Software design according to the safety level
- Define Failures (Failures could happen in a Unit / Component / Communication Buses / Software)
- Define Fail-Safe Action
(System shut-down, Turn into manual operation, Maintenance, etc.)
- Calculate the achieved safety
- Re-design if the required safety is not achieved

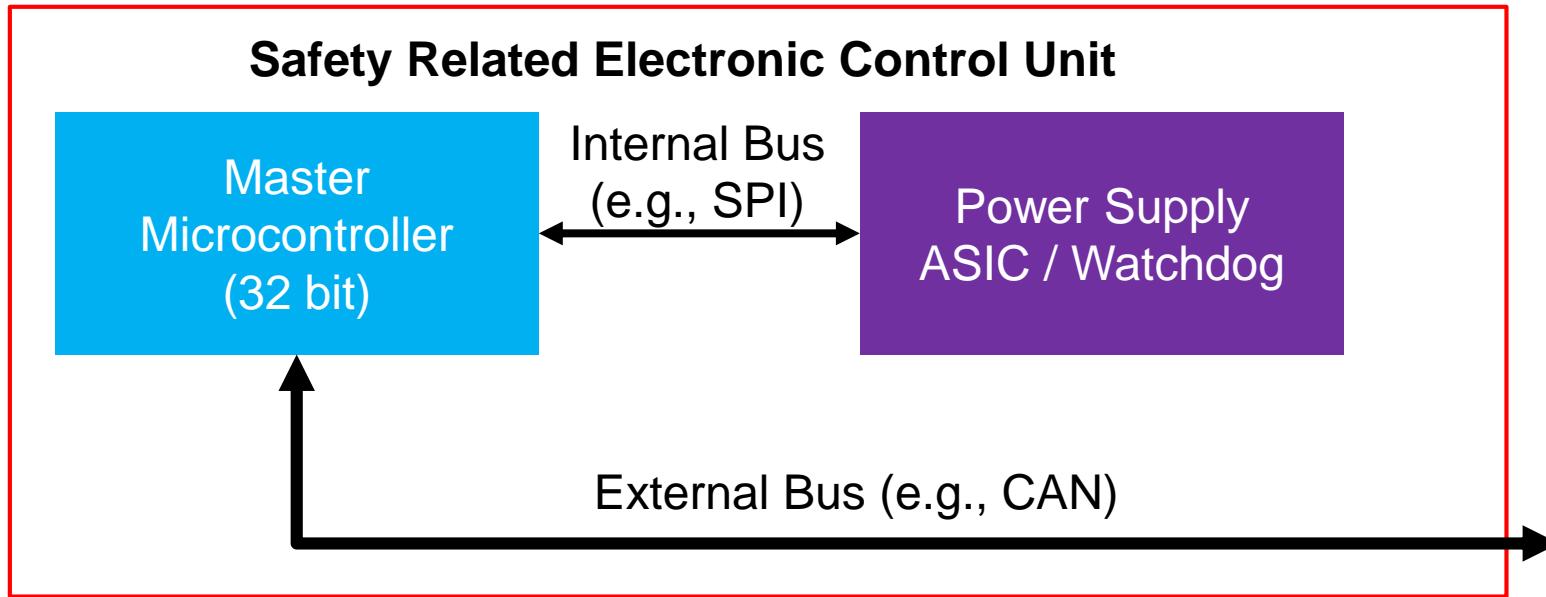
- Power supply
 - A redundant power supply (PS) units is required for some Safe Systems



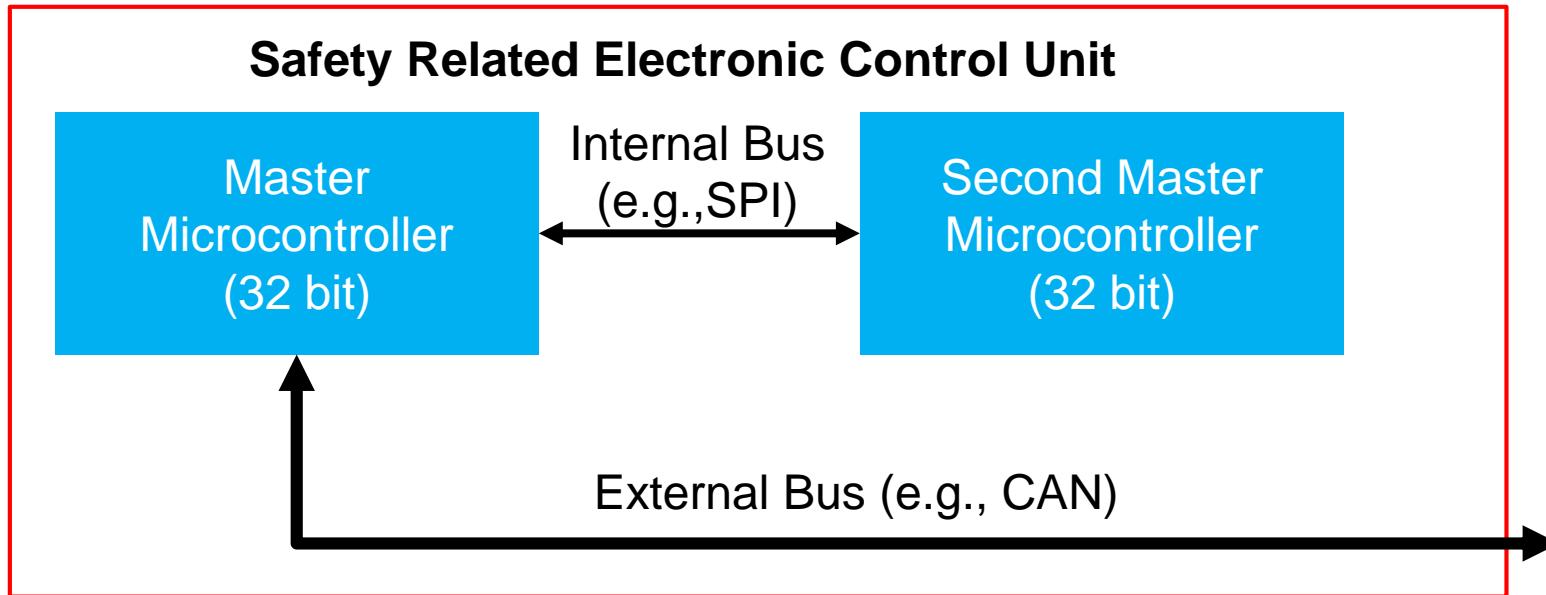
- Sensors & Actuators / Drivers
 - Increase Diagnostic Coverage (DC) to more than 99%, if possible

- Watchdog Timer
 - Enhance the work of the watchdog (WD), by working with single or double WD and the way to connect them with the external Hardware (HW).



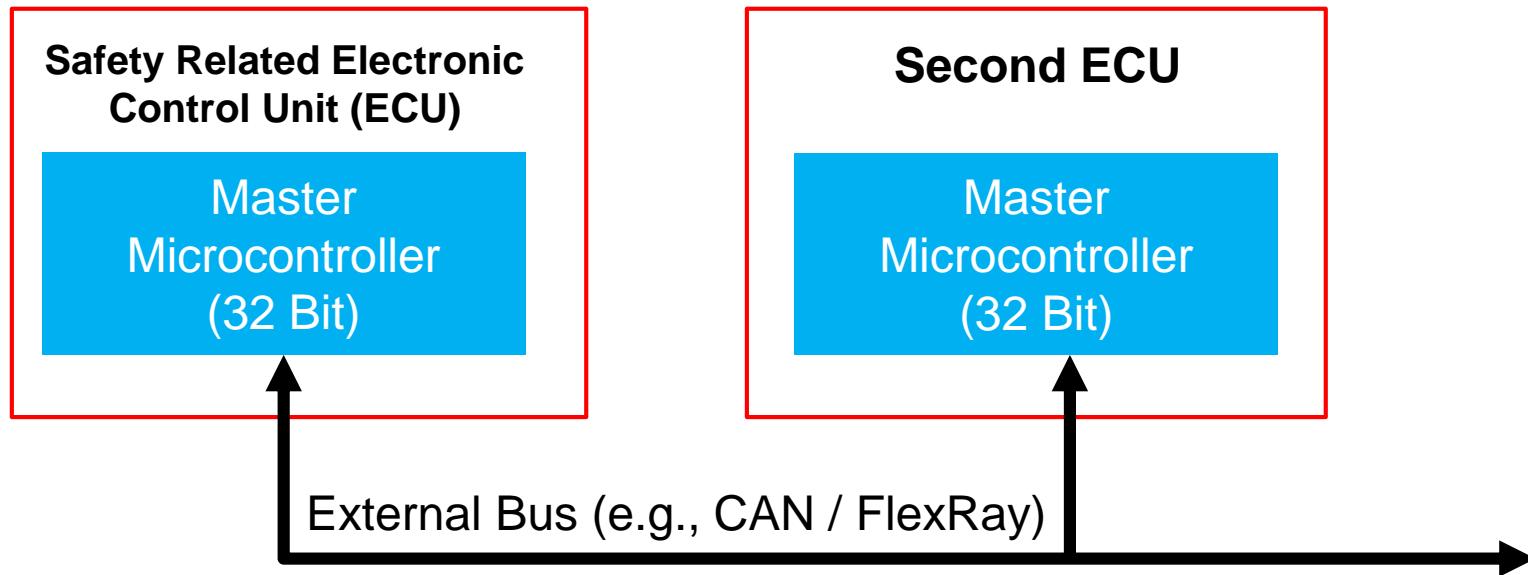


- Master controller plus external watchdog ASIC
- ASIC requires periodic service by Master
- ASIC has the ability to RESET the Master
- Master controller has to also run safety test code

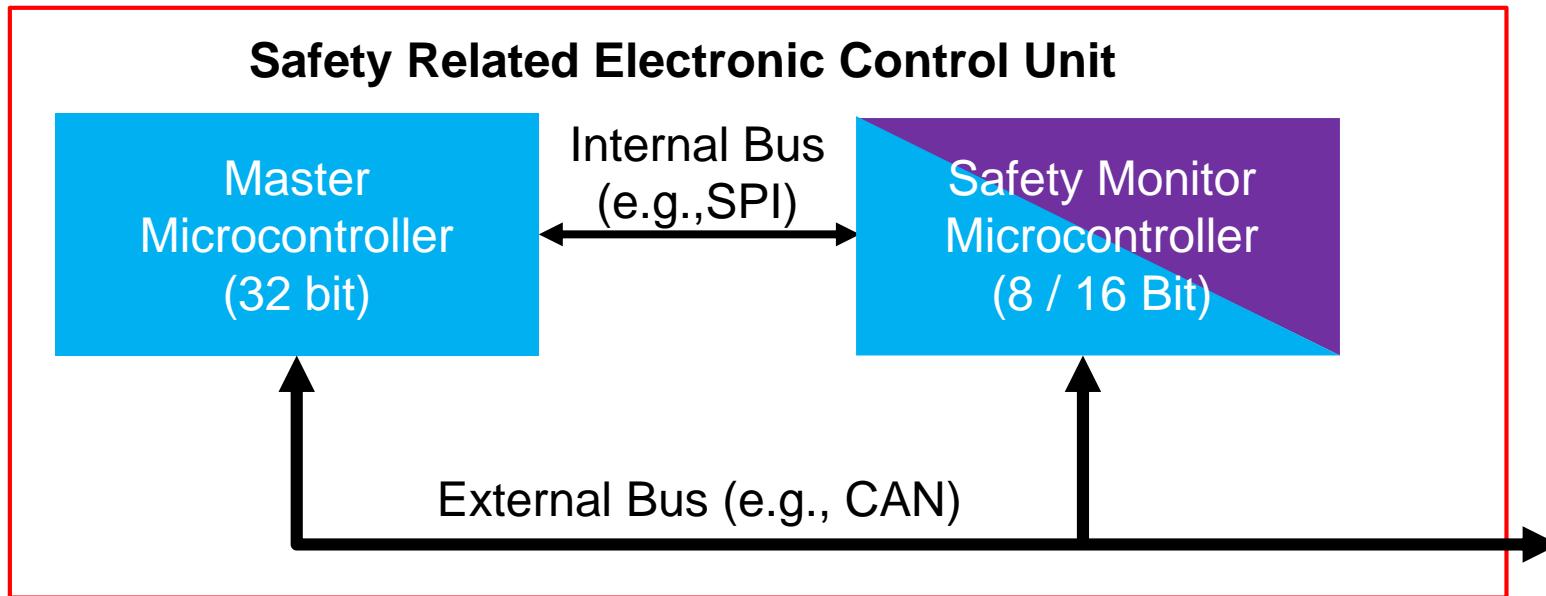


- Two identical controllers / technologies / vendors
- Both controller compute same algorithm / code
- Both controllers exchange results periodically
- Synchronization critical to ensure coherent data
- Both controllers have full size → expensive

Control Unit: Distributed Controller Architecture

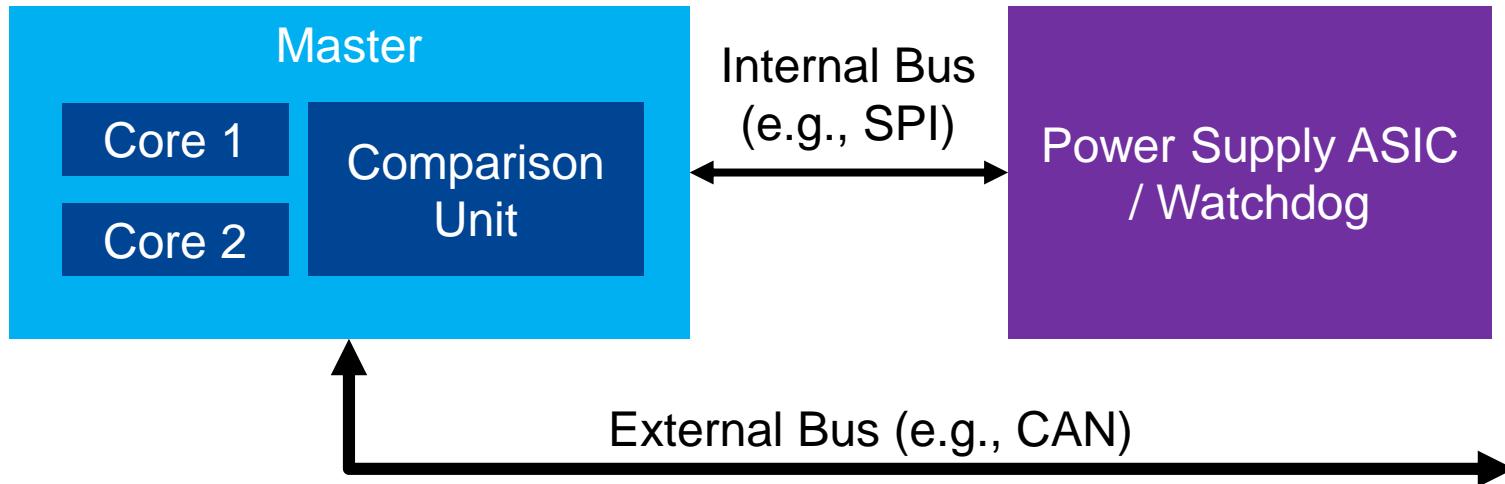


- Two separate ECUs connected by 'secure' bus
- Second ECU requests self test of Safety ECU
- Second ECU can request a Safety ECU shutdown
- External bus connection effects reliability / safety
- Cooperation of two ECUs is an engineering challenge



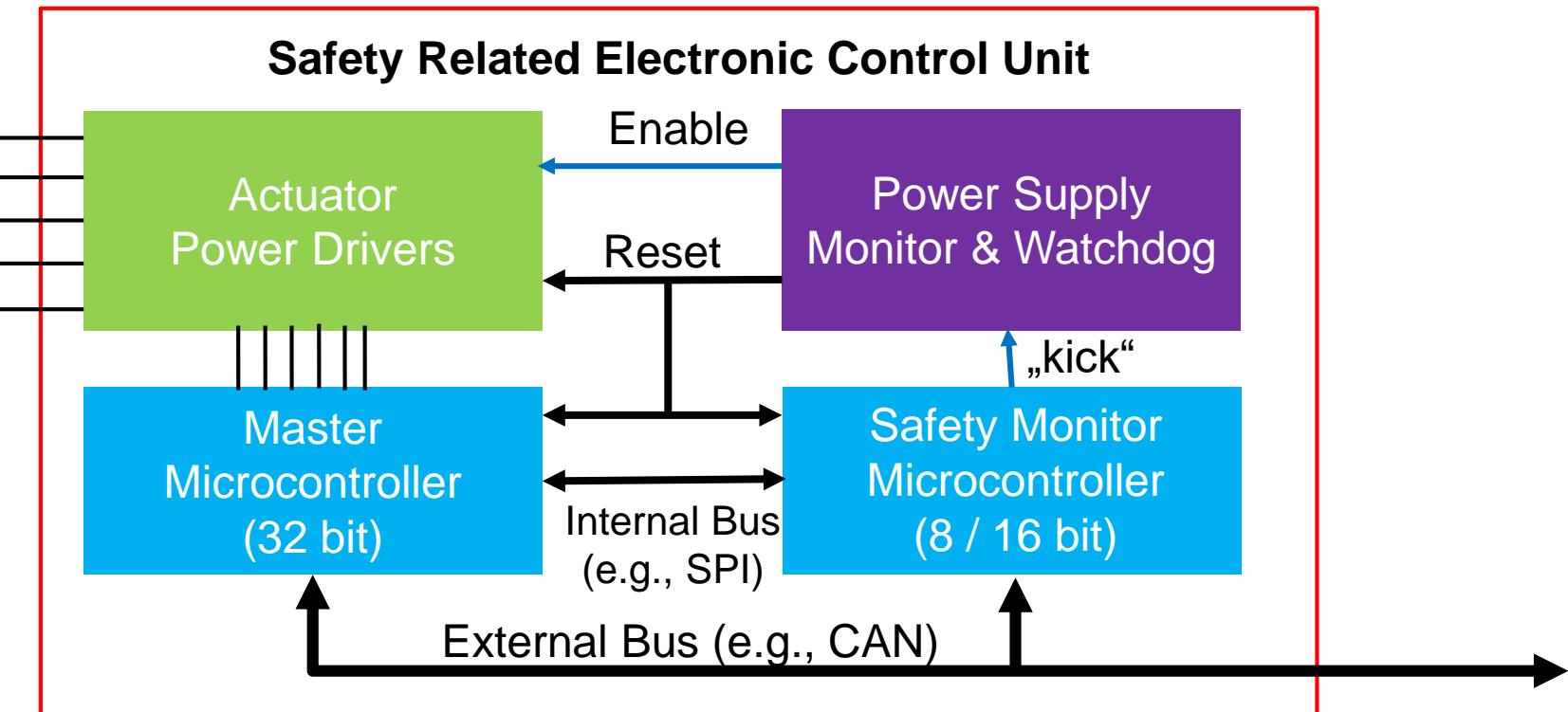
- Two different controllers / technologies / vendors
- Monitor request self checks by master
- Monitor has the ability to bring system to 'Safe State'
- Synchronization and bandwidth limits coverage
- Two independent toolsets make debugging tricky

Safety Related Electronic Control Unit

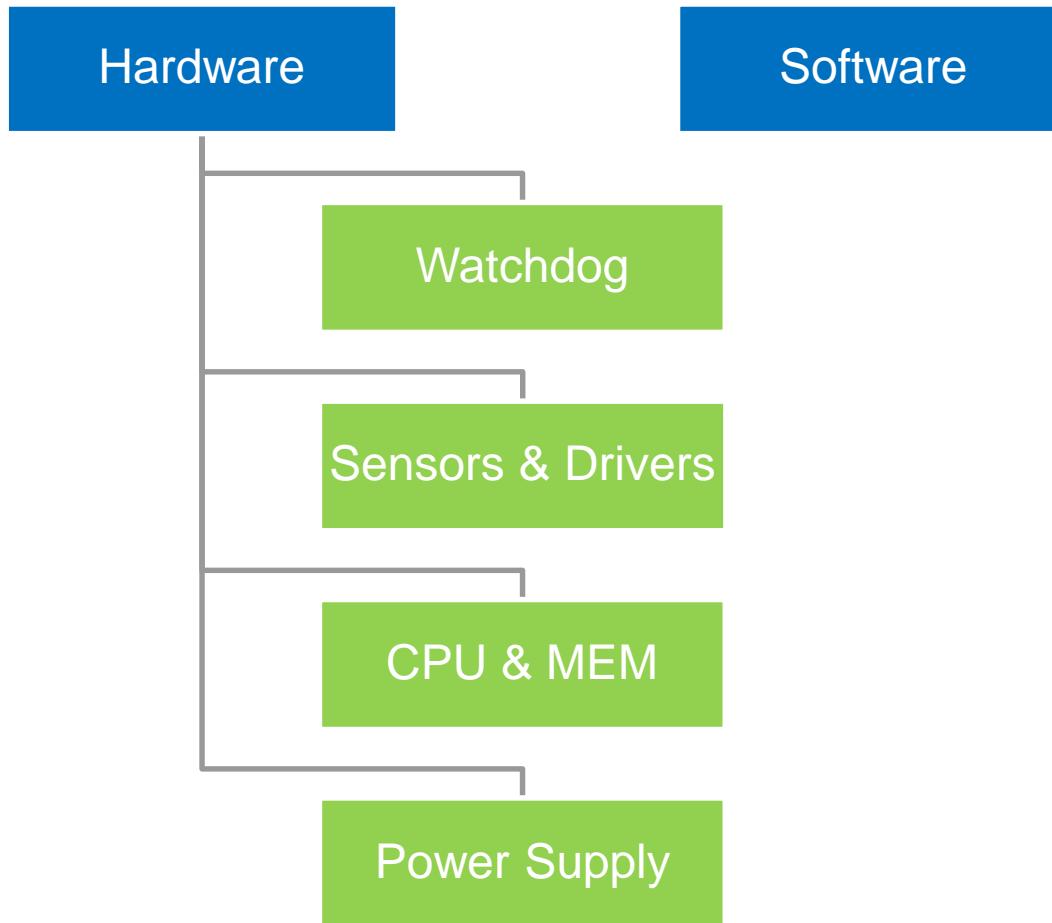


- Two instances of the same core
- Both cores run same software
- Comparison unit verifies coherent output
- Pay for two cores but get one core performance

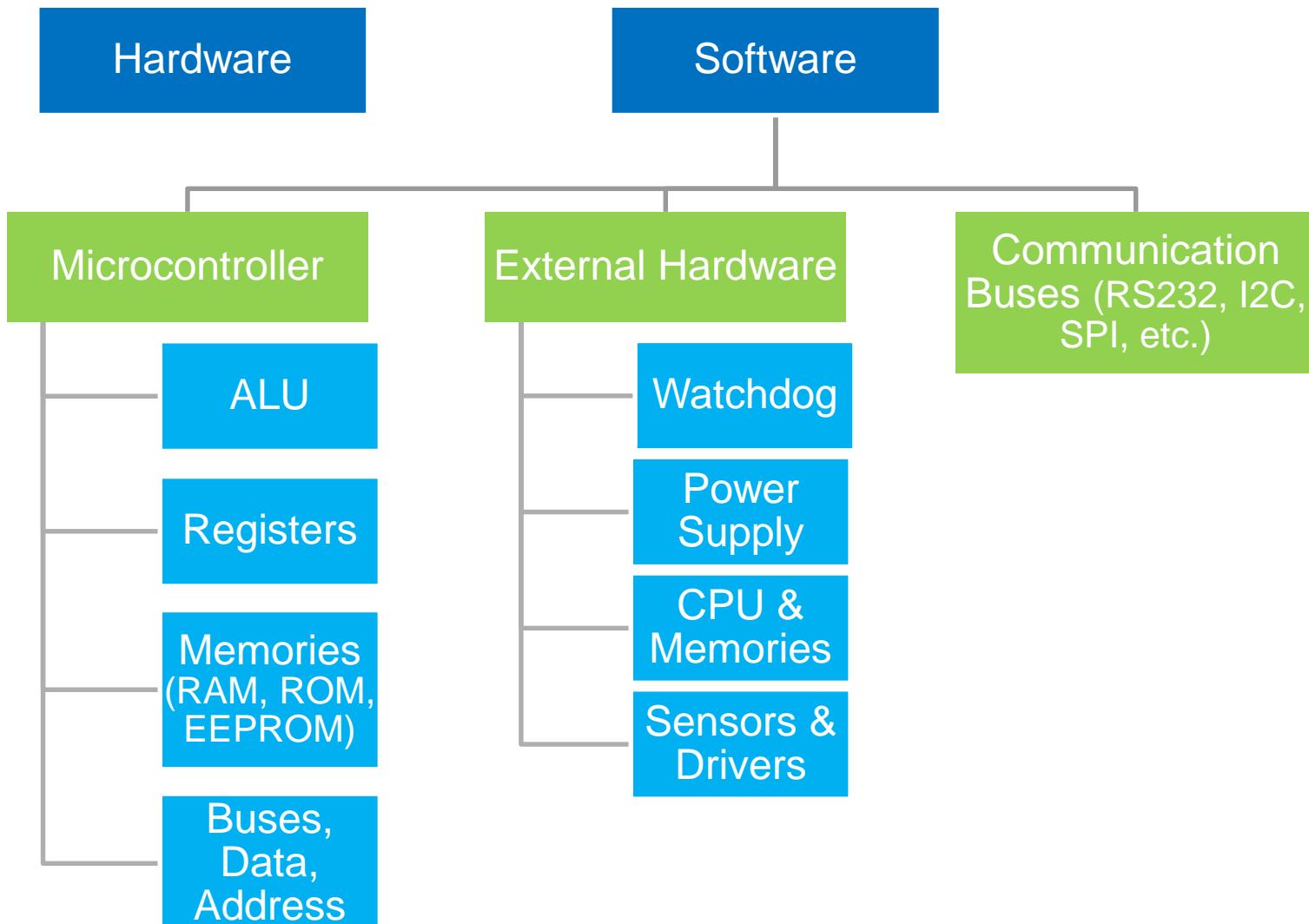
Control Unit: Detail of Asymmetric Controller Architecture



Safe System Software cyclic test routines



Safe System Software cyclic test routines



Reliability Block Diagram (RBD)

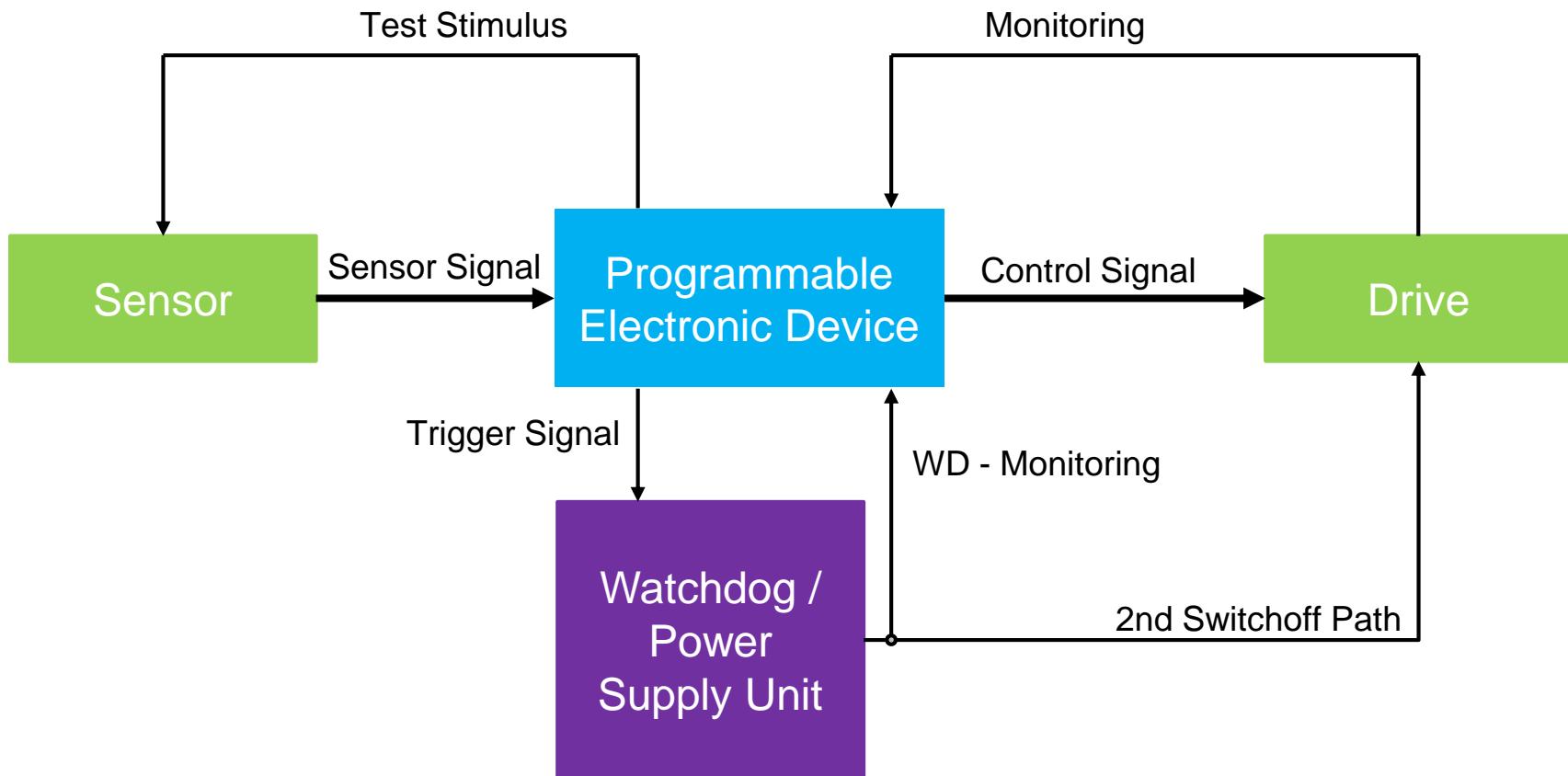
Block diagram of a single channel system without fault detection



Source: STSARCES, Annex 6

Reliability Block Diagram (RBD)

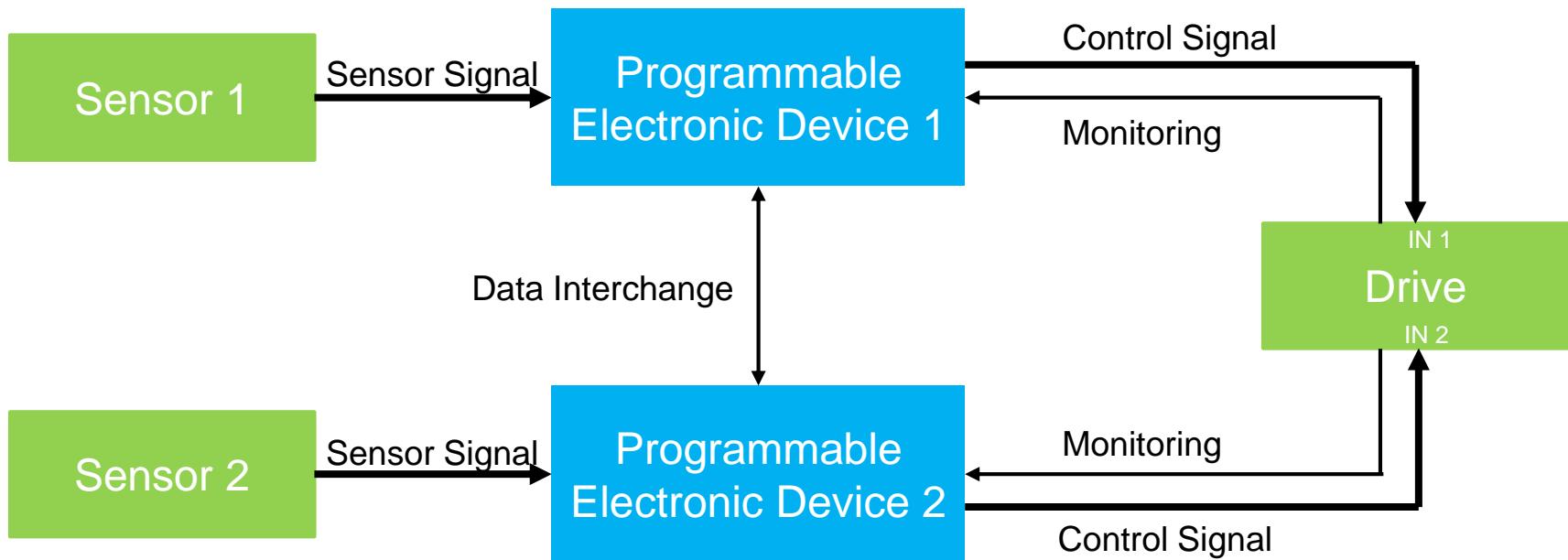
Block diagram for a Single Channel Safe System



Source: STSARCES, Annex 6

Reliability Block Diagram (RBD)

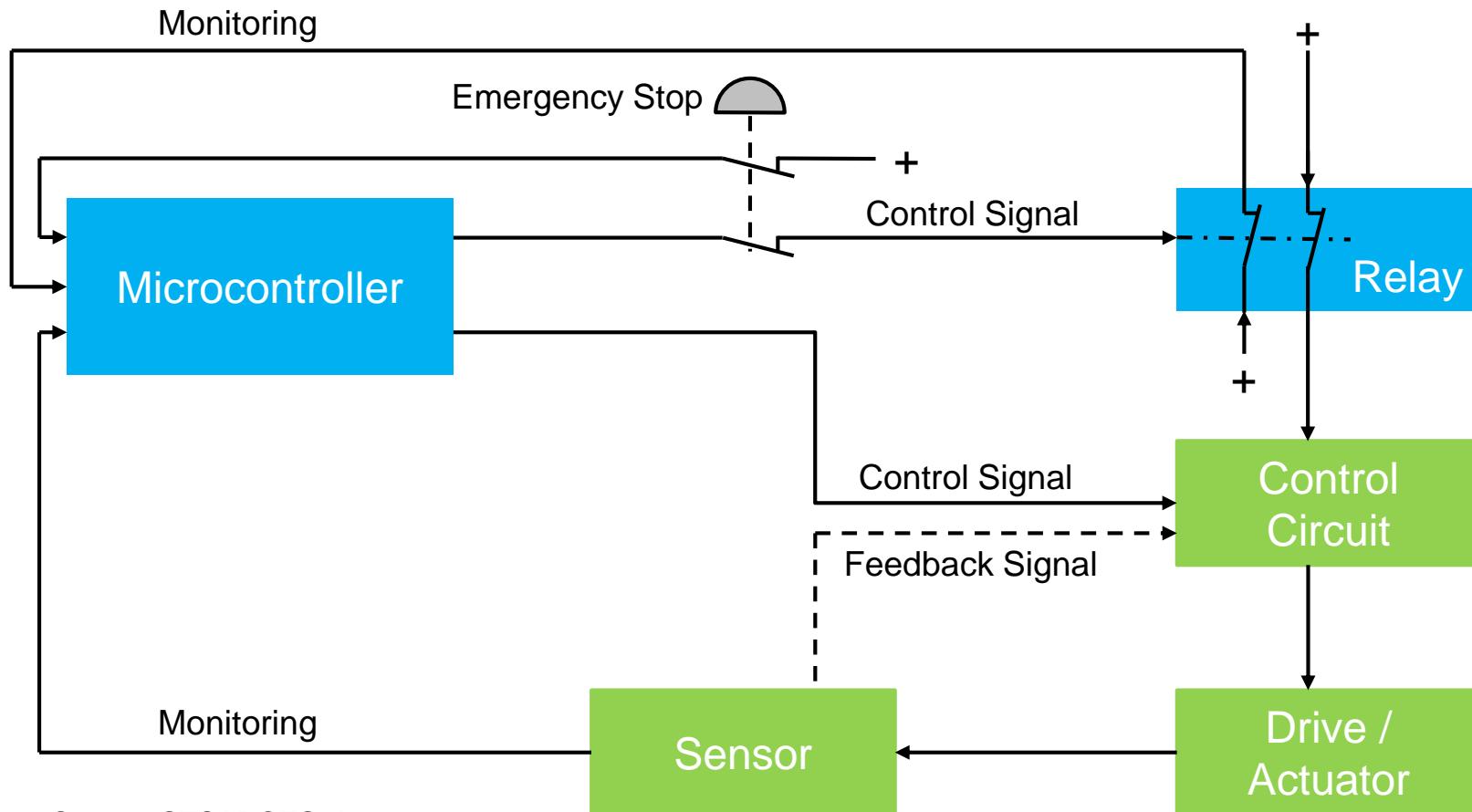
Block diagram for two channel system with Comparison



Source: STSARCES, Annex 6

Reliability Block Diagram (RBD)

Mixed technology Safe System



Source: STSARCES, Annex 6

Hardware Component List, their Failure & Effect

Name	Component	Qty	Function	Failure Mode	Effect	Lb	Pct_Lb	Distribution	Diagnostics	DC	Behavior
Infineon, TC1796	Microprocessor - BiCMOS (>100k-1M gates, >500k-5M transistors) --- Safe failures	1		Register, internal RAM		5,0E-08	50%	15%		99%	!
		1		Coding and execution (ALU) including flag register		5,0E-08	50%	60%		99%	!
		1		Address calculation		5,0E-08	50%	15%		99%	!
		1		Program counter, stack pointer		5,0E-08	50%	5%		99%	!
		1		Interrupt handling		5,0E-08	50%	5%		99%	!
		1		---		5,0E-08	50%	0%		0%	!
BOSCH, CY320	Watchdog - Voltage monitor CMOS (>300-3k transistors) --- Safe failures	1		Unable to trip		8,0E-09	50%	30%		99%	!
		1		False trip		8,0E-09	50%	50%		99%	!
		1		Short circuit between any two connections		8,0E-09	50%	20%		99%	!
		1		---		8,0E-09	50%	0%		0%	!
		1		---		8,0E-09	50%	0%		0%	!
		1		---		8,0E-09	50%	0%		0%	!
SMD Resistor	Resistor - Metal film --- No further functionality specified	1		Short circuit		2,0E-10	100%	10%		99%	!
		1		Open circuit		2,0E-10	100%	60%		99%	!
		1		Change in value (0.5x)		2,0E-10	100%	15%		99%	!
		1		Change in value (2x)		2,0E-10	100%	15%		99%	!
		1		---		2,0E-10	100%	0%		0%	!
		1		---		2,0E-10	100%	0%		0%	!

SIL Calculation - Excel Tool FMEDA (Failure Mode, Effect & Diagnostics Analysis)



Modify / Copy Components

Existing Type of Component:

Microprocessor - CMOS (< 1k gates, < 5k transistors) --- No further functionality specified

New Type of Component:

Microprocessor - CMOS (< 1k gates, < 5k transistors) --- No further functionality specified

CHANGE COMPONENT

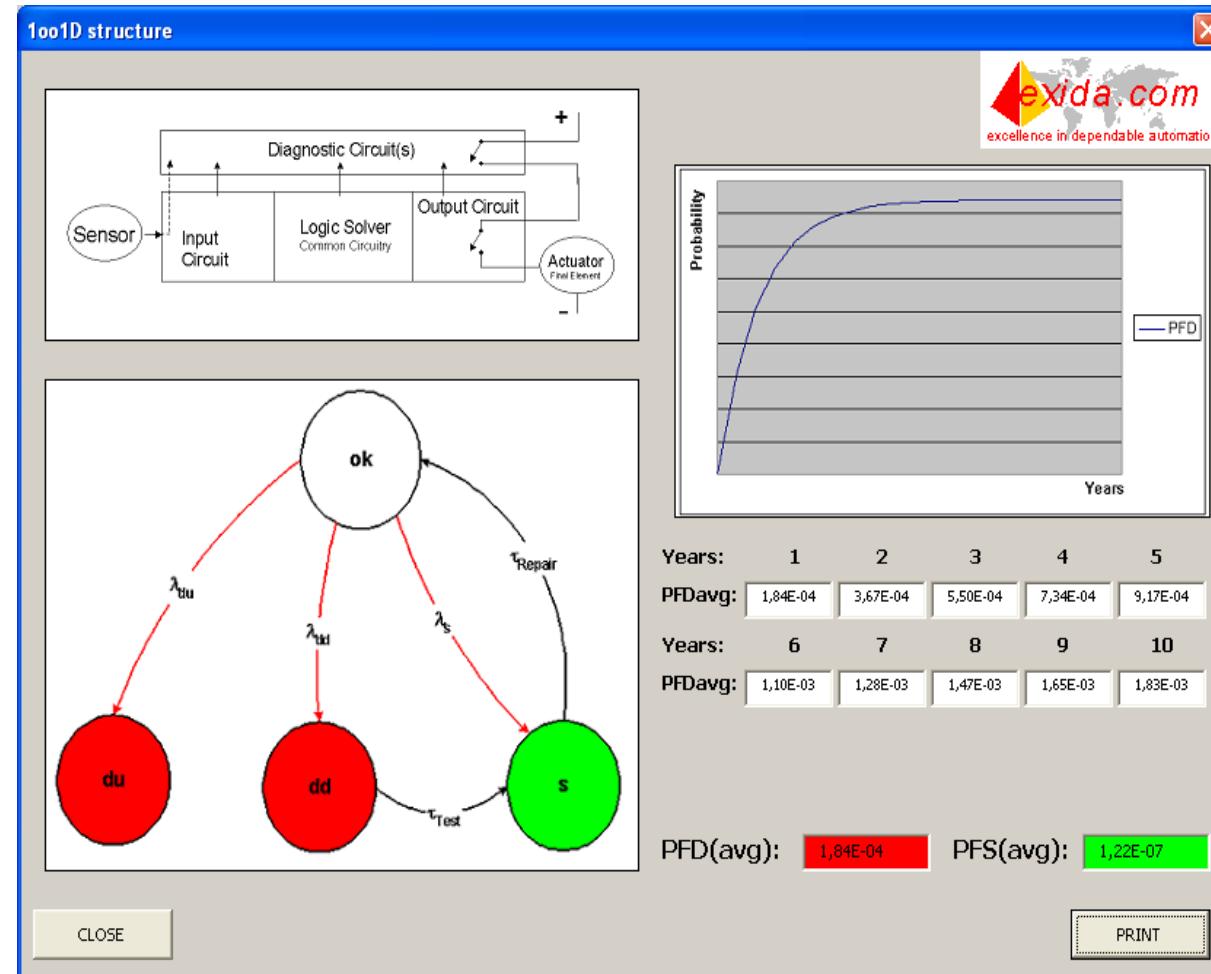
Failure rate L_b [1/h]: Percentage of L_b: Failure rate source:

4,0E-09 100% SN29500.xla

Existing Name(s):	New Name(s):	Quantity:	Function:	Reference:	Remarks:
IC221	IC221	1	XC161CS - µC with integrated ROM and SRAM	9115823 - sh.2	MPU A, µC A (fit rate supported by Infineon)
Failure Modes:		Distribution [%]:	Effects:	Diagnostics:	DC [%]: Behavior:
Register, internal RAM, internal ROM		15	Bad data in telegrams, bad program execution	RAM/ROM tests by FW, CRC checks, redundant circuit in other channel with FW-	99 D
Coding and execution		60	Bad data in telegrams, bad program execution	Code tests by FW, CRC checks, redundant circuit in other channel with FW-	99 D
Address calculation		10	Bad data in telegrams, bad program execution	Tests by FW, CRC checks, redundant circuit in other channel with FW-diagnostics	99 D
Program counter, stack pointer		5	Bad data in telegrams, bad program execution	Tests by FW, CRC checks, redundant circuit in other channel with FW-diagnostics	99 D
Interrupt handling		5	Bad data in telegrams, bad program execution	Tests by FW, CRC checks, redundant circuit in other channel with FW-diagnostics	99 D
Cross Com (SSC)		5		Tests by FW, CRC checks, redundant circuit in other channel with FW-diagnostics	99 D

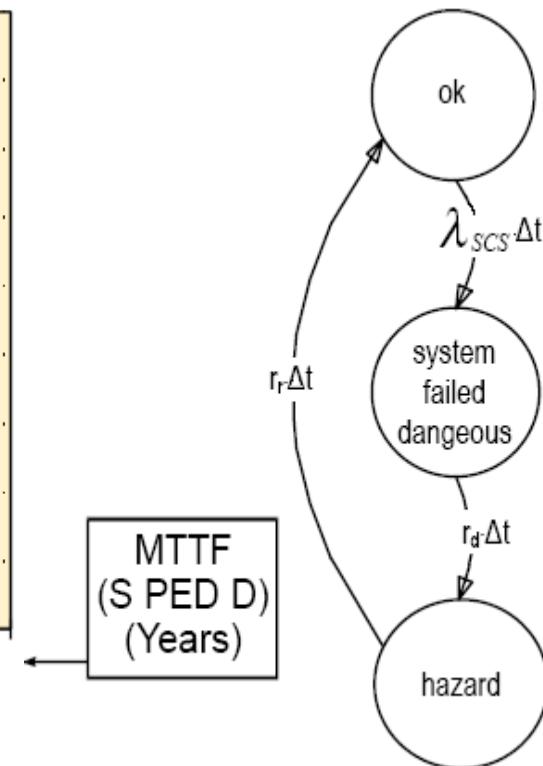
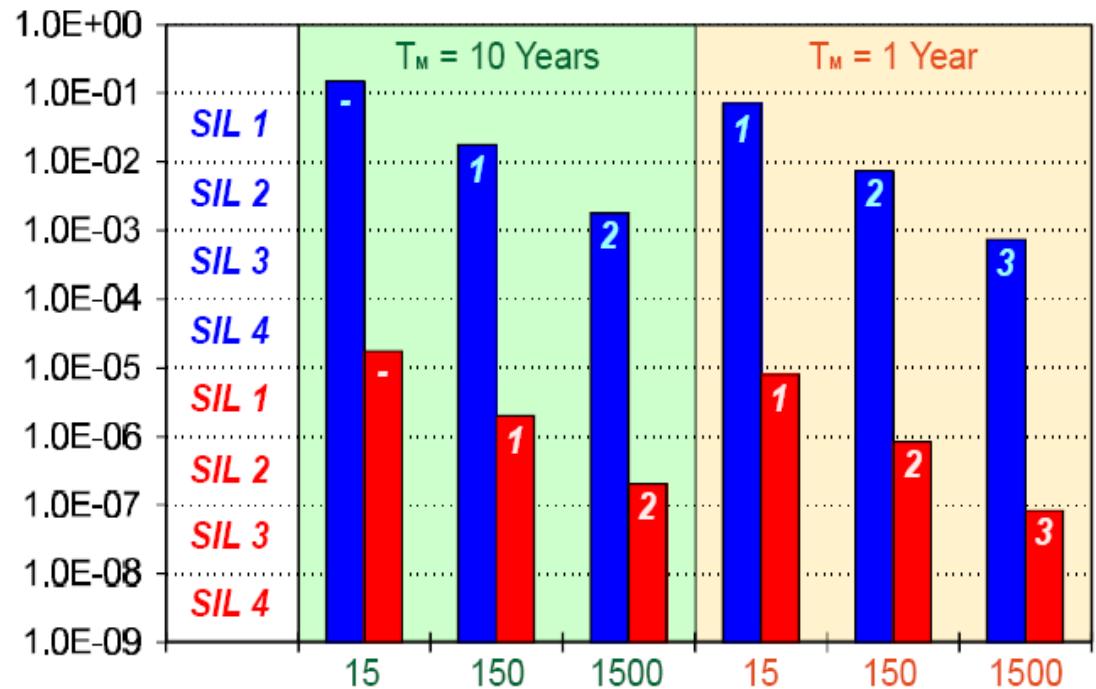
CANCEL SAVE HELP RESULTS ADD COPY MODIFY DELETE

SIL Calculation – FMEDA Result



SIL3: $10^{-4} < \text{PFD} < 10^{-3}$ as well $10^{-8} < \text{PFH} < 10^{-7}$

Markov Model & its Evaluation Result



MTTF
(S PED D)
(Years)

Demand rate = 1/year

Repair rate = 1/8 hours

MTTF = Mean Time To Fail

Certify Agency

- TÜV (Technischer Überwachungsverein)
- WSO (World Safety Organisation)

State of the lecture

- ✓ Chapter 4: Programming
 - ✓ Assembler, C, Compiler, ...
- ✓ Chapter 5: Communication
 - ✓ OSI-Model, topologies, UART, I2C, ...
- ✓ Chapter 6: Outlook (among others: Safe Systems)