TECHNISCHE UNIVERSITÄT DRESDEN

Elektrotechnik und Informationstechnik, Stiftungsprofessur hochparallele VLSI Systeme und Neuromikroelektronik

Circuit and System Design

Part 2: Digital Systems

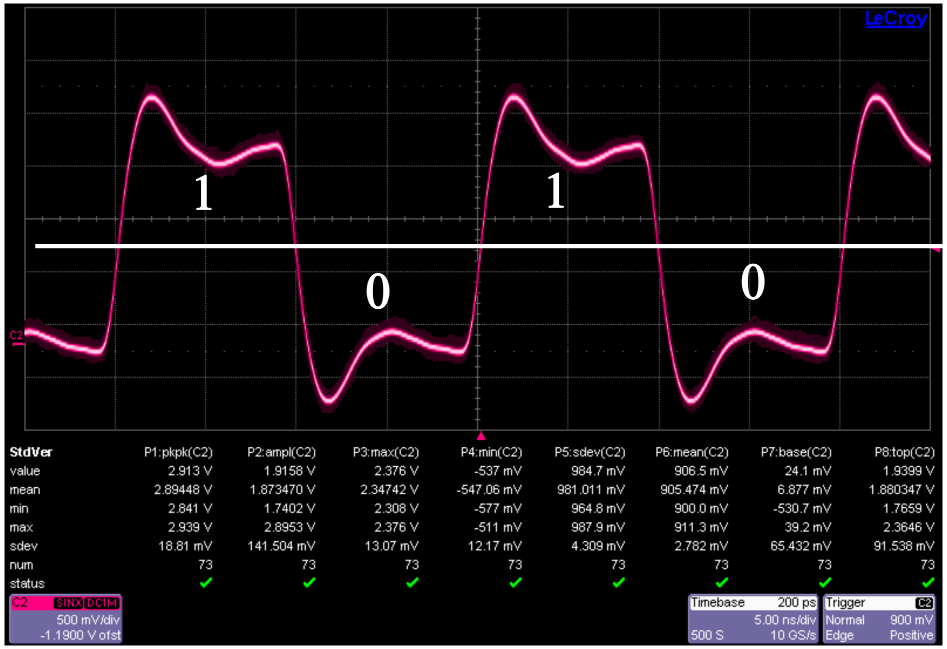
FAKULTÄT **ELEKTROTECHNIK** UND **INFORMATIONSTECHNIK**

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Logic Circuits

Binary Value Representation

- 2-valued representation:
  - 0, LOW, L, FALSE
  - 1, HIGH, H, TRUE
- Electrical representation as voltage signal:
  - $V_{sig} > V_{th} \rightarrow 1$
  - $V_{sig} < V_{th} \rightarrow 0$



StdVer	P1:pkpk(C2)	P2:ampl(C2)	P3:max(C2)	P4:min(C2)	P5:sdev(C2)	P6:mean(C2)	P7:base(C2)	P8:top(C2)
value	2.913 V	1.9158 V	2.376 V	-537 mV	984.7 mV	906.5 mV	24.1 mV	1.9399 V
mean	2.89448 V	1.873470 V	2.34742 V	-547.06 mV	981.011 mV	905.474 mV	6.877 mV	1.880347 V
min	2.841 V	1.7402 V	2.308 V	-577 mV	964.8 mV	900.0 mV	-530.7 mV	1.7659 V
max	2.939 V	2.8953 V	2.376 V	-511 mV	987.9 mV	911.3 mV	39.2 mV	2.3646 V
sdev	18.81 mV	141.504 mV	13.07 mV	12.17 mV	4.309 mV	2.782 mV	65.432 mV	91.538 mV
num	73	73	73	73	73	73	73	73
status	✓	✓	✓	✓	✓	✓	✓	✓

C2 SINX DC1M
500 mV/div
-1.1900 V ofst

Timebase 200 ps
5.00 ns/div
500 S 10 GS/s

Trigger C2
Normal 900 mV
Edge Positive

Logic Functions

- Connecting (relating) input variables to output variables

$$\boxed{\left(Z_1, Z_{2,} \cdots\right) = F\left(A_1, A_{2,} \cdots\right)}$$

- Forms of representation:
  - Logic equations
  - Truth table
  - Gate netlist

Boolean Operations

- Operators:
  - Inversion: $Z = \bar{A}$

A	Z
0	1
1	0

  - AND: $Z = A_1 \cdot A_2$

A_1	A_2	Z
0	0	0
0	1	0
1	0	0
1	1	1

  - OR: $Z = A_1 + A_2$

A_1	A_2	Z
0	0	0
0	1	1
1	0	1
1	1	1

Calculation Rules

- Calculation rules:

	Operation + (OR)	Operation · (AND)
1)	$(x+y)+z=x+(y+z)$	$(xy)z=x(yz)$
2)	$(x+y)=(y+x)$	$(xy)=(yx)$
3)	$x+xy=x$	$x(x+y)=x$
4)	$x+yz=(x+y)(x+z)$	$x(y+z)=xy+xz$
5)	$x+\bar{x}=1$	$x\bar{x}=0$
6)	$x+0=x$	$x1=x$
7)	$x+1=1$	$x0=0$
8)	$x+x=x$	$xx=x$
8)	$\overline{x+y}=\bar{x}\bar{y}$	$\overline{xy}=\bar{x}+\bar{y}$
9)	$\bar{\bar{x}}=x$	

Truth Table

- Representation of a logic function as a table

inputs			outputs	
A_1	A_2	A_3	Z_1	Z_2
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	0
1	1	1	0	0

TECHNISCHE UNIVERSITÄT DRESDEN

Don‘t care Statement

- Reduction of truth tables by Don‘t care (**X**)
- → Values which have no effect on the outputs

inputs			outputs	
A_1	A_2	A_3	Z_1	Z_2
X	0	0	0	0
X	0	1	1	1
X	1	0	1	0
X	1	1	0	0

Making up Logic Equations

- Relation between Min- or Max-terms of a logic function
- Minterm:
  - For exactly one output assignment of inputs = 1
  - OR function to connect minterms to get output value

inputs			outputs	
A_1	A_2	A_3	Z_1	Z_2
X	0	0	0	0
X	0	1	1	1
X	1	0	1	0
X	1	1	0	0

$\boxed{\text{Minterms}}$

$$Z_1 = \overline{A_2} \cdot A_3 + A_2 \cdot \overline{A_3}$$

$$Z_2 = \overline{A_2} \cdot A_3$$

Making up Logic Equations

- Maxterm:
  - For exactly one output assignment of inputs = 0
  - AND function to connect maxterms to get output value

inputs			outputs	
A_1	A_2	A_3	Z_1	Z_2
X	0	0	0	0
X	0	1	1	1
X	1	0	1	0
X	1	1	0	0

$\boxed{\text{Maxterms}}$

$$Z_1 = (A_2+A_3) \cdot (\overline{A_2} + \overline{A_3})$$

$$Z_2 = (A_2+A_3) \cdot (\overline{A_2} + A_3) \cdot (\overline{A_2} + \overline{A_3})$$

Karnaugh Map

- A method for simplifying logic functions
- Modified arrangement of the truth table values
- Karnaugh map with
  - 2 inputs:

Z	A_1	$\overline{A_1}$
A_2		
$\overline{A_2}$		

  - 3 inputs:

Z	A_1A_2	$\overline{A_1}A_2$	$\overline{A_1}\,\overline{A_2}$	$A_1\overline{A_2}$
A_3				
$\overline{A_3}$				

  - 4 inputs:

Z	A_1A_2	$\overline{A_1}A_2$	$\overline{A_1}\,\overline{A_2}$	$A_1\overline{A_2}$
A_3A_4				
$\overline{A_3}A_4$				
$\overline{A_3}\,\overline{A_4}$				
$A_3\overline{A_4}$				

Karnaugh Map: Minterms

- Enter the values in logic table into Karnaugh map
- Minterm method:
  - Bringing together the fields, which contain 1 (minerms)
  - Converting these fields into connected terms (AND functions)
  - Omitting variables in negated and not negated form
  - ORing these terms

Z	A_1A_2	$\overline{A_1}A_2$	$\overline{A_1A_2}$	$A_1\overline{A_2}$
A_3A_4	1	1	0	0
$\overline{A_3}A_4$	0	0	1	1
$\overline{A_3A_4}$	0	0	1	1
$A_3\overline{A_4}$	1	1	0	0

$$\boldsymbol{Z} = \boxed{\overline{\boldsymbol{A}_2} \cdot \overline{\boldsymbol{A}_3}} + \boxed{\boldsymbol{A}_2 \cdot \boldsymbol{A}_3}$$

Karnaugh Map: Maxterms

- Maxterm method:
  - Bringing together the fields, which contain 0 (maxterms)
  - Converting these fields into connected terms (OR functions), Invertion of variables
  - Omitting variables in negated and not negated form
  - ANDing these terms

Z	A_1A_2	$\overline{A_1}A_2$	$\overline{A_1A_2}$	$A_1\overline{A_2}$
A_3A_4	1	1	0	0
$\overline{A_3}A_4$	0	0	1	1
$\overline{A_3A_4}$	0	0	1	1
$A_3\overline{A_4}$	1	1	0	0

$$Z = \boxed{(\overline{A_2} + \overline{\overline{A_3}})} \cdot \boxed{(\overline{\overline{A_2}} + \overline{A_3})}$$

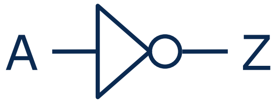
$$Z = (\overline{A_2} + A_3) \cdot (A_2 + \overline{A_3})$$

$$Z = \overline{A_2} \cdot \overline{A_3} + A_2 \cdot A_3$$

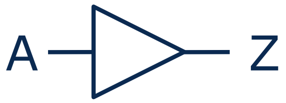
Combinational Logic Cells

- Implementing logic functions using logic cells (Gates)
- Examples:

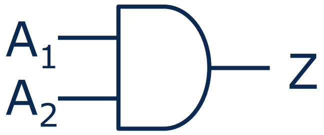
INV: $Z = \bar{A}$



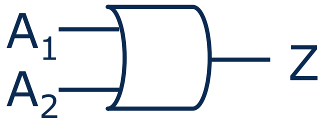
BUF: $Z = A$



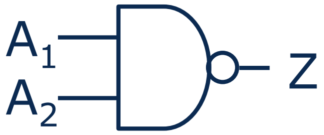
AND2: $Z = A_1 \cdot A_2$



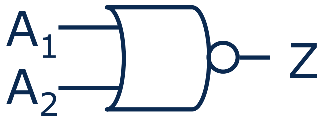
OR2: $Z = A_1 + A_2$



NAND2: $Z = \overline{A_1 \cdot A_2}$



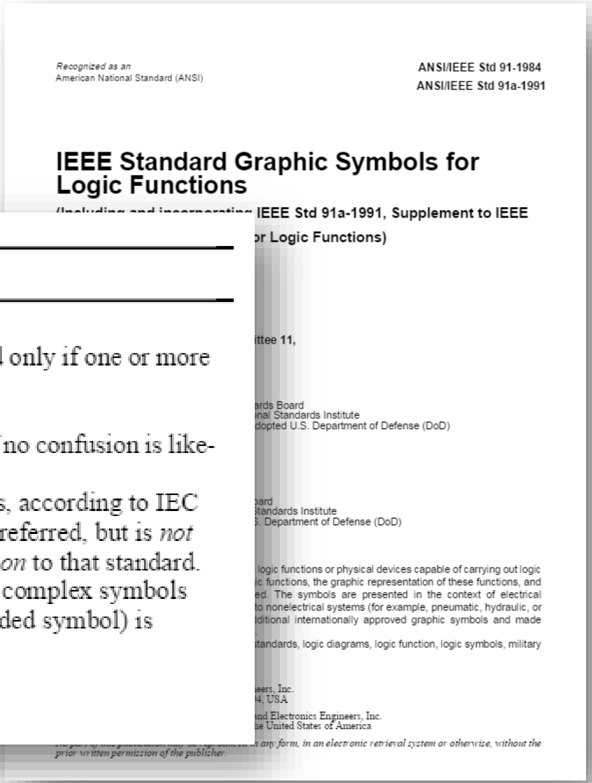
NOR2: $Z = \overline{A_1 + A_2}$



- Logic cell libraries additionally contain complex gates with >2 inputs
  - Adder, Multiplexer, AOI, OAI

Logic Symbols

- Standardized representation of logic symbols
  - IEC 60617-12 : 1997
  - ANSI/IEEE Std 91/91a-1991



Recognized as an
American National Standard (ANSI)

ANSI/IEEE Std 91-1984
ANSI/IEEE Std 91a-1991

IEEE Standard Graphic Symbols for Logic Functions

(Including and incorporating IEEE Std 91a-1991, Supplement to IEEE
or Logic Functions)

ittee 11,

ards Board
nal Standards Institute
dopted U.S. Department of Defense (DoD)

ard
tandards Institute
. Department of Defense (DoD)

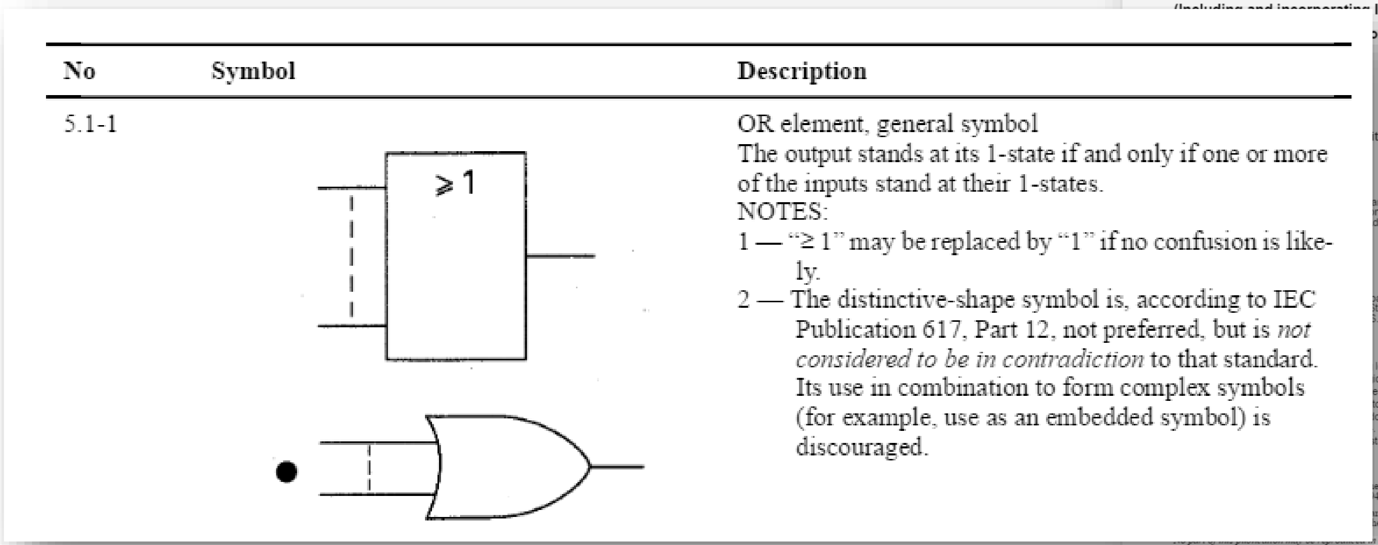
logic functions or physical devices capable of carrying out logic
ic functions, the graphic representation of these functions, and
ed. The symbols are presented in the context of electrical
to nonelectrical systems (for example, pneumatic, hydraulic, or
ditional internationally approved graphic symbols and made

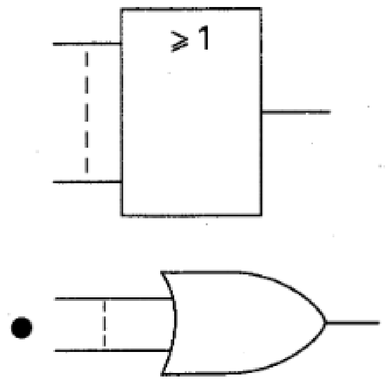
tandards, logic diagrams, logic function, logic symbols, military

eers, Inc.
4, USA

nd Electronics Engineers, Inc.
he United States of America

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

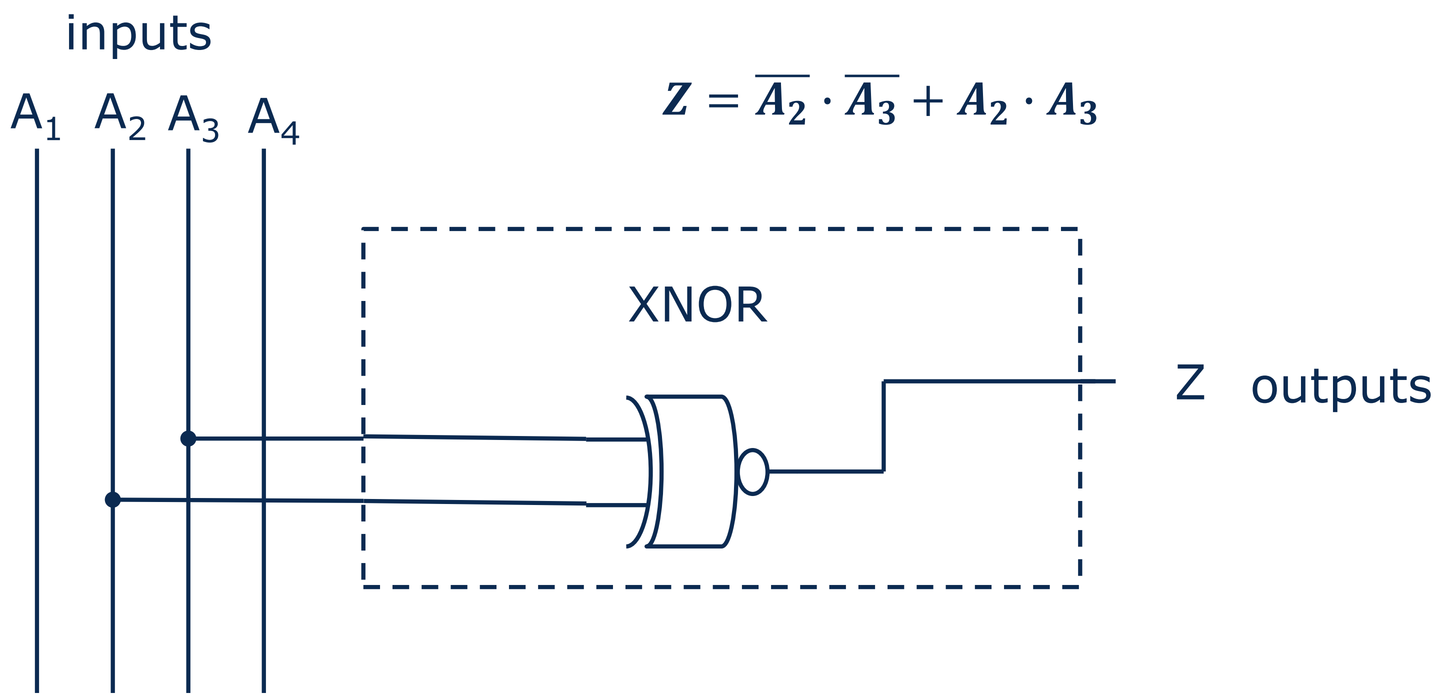


No	Symbol	Description
5.1-1		OR element, general symbol The output stands at its 1-state if and only if one or more of the inputs stand at their 1-states. NOTES: 1 — "≥ 1" may be replaced by "1" if no confusion is likely. 2 — The distinctive-shape symbol is, according to IEC Publication 617, Part 12, not preferred, but is *not considered to be in contradiction* to that standard. Its use in combination to form complex symbols (for example, use as an embedded symbol) is discouraged.

- Widespread use of "distinctive-shape symbols" in manuals, standard cell libraries, scientific publications
- "... in the English-speaking world the American symbols (middle column) were and are common. *The IEC symbols have met with limited international acceptance and are (almost) consistently ignored in American literature."* https://en.wikipedia.org/wiki/Logic_gate (23.10.2015)

Gate Netlist

- Representation of logic function using gate circuits

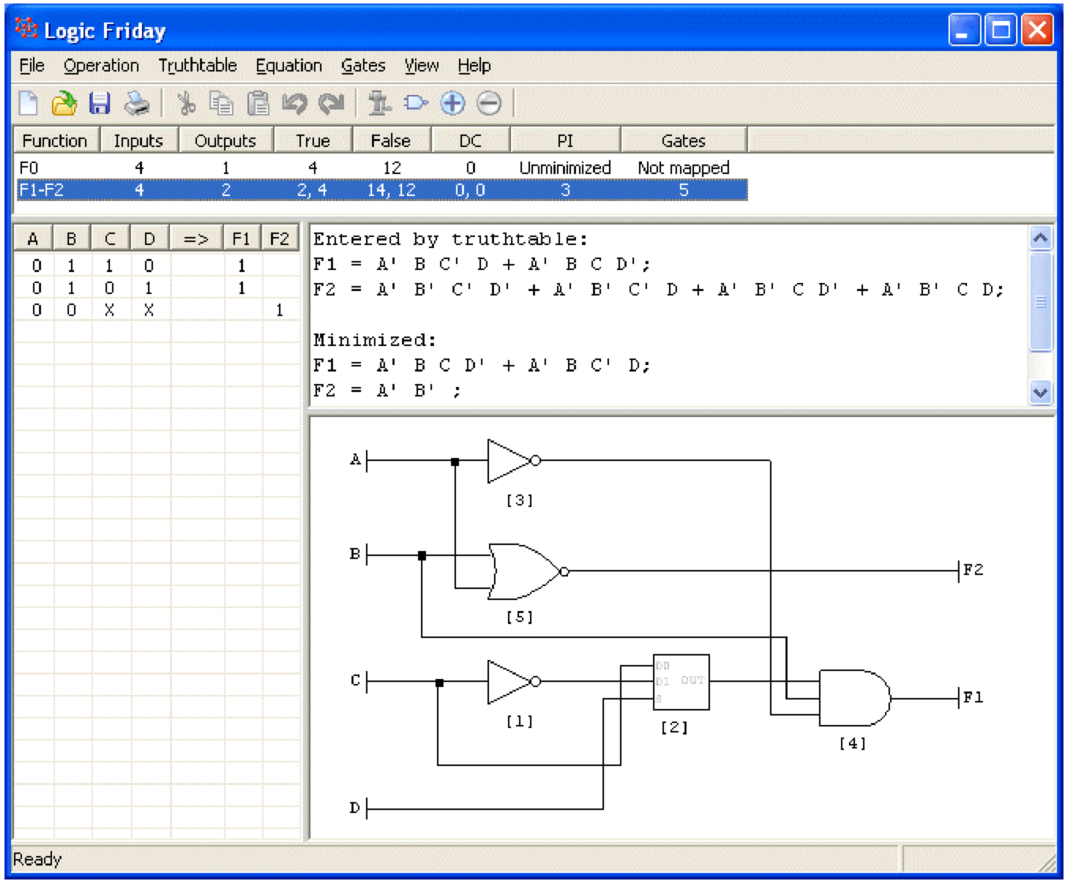


- Representation of the logic function depending on the available gate
- → Circuit Synthesis, Mapping

TECHNISCHE UNIVERSITÄT DRESDEN

Software: Logic Friday

- Freeware Logic Friday → **www.sontrak.com**
- Representation, simplification and optimization of logic functions (equation, table, gate netlist)



Function	Inputs	Outputs	True	False	DC	PI	Gates
F0	4	1	4	12	0	Unminimized	Not mapped
F1-F2	4	2	2, 4	14, 12	0, 0	3	5

A	B	C	D	=>	F1	F2
0	1	1	0		1	
0	1	0	1		1	
0	0	X	X			1

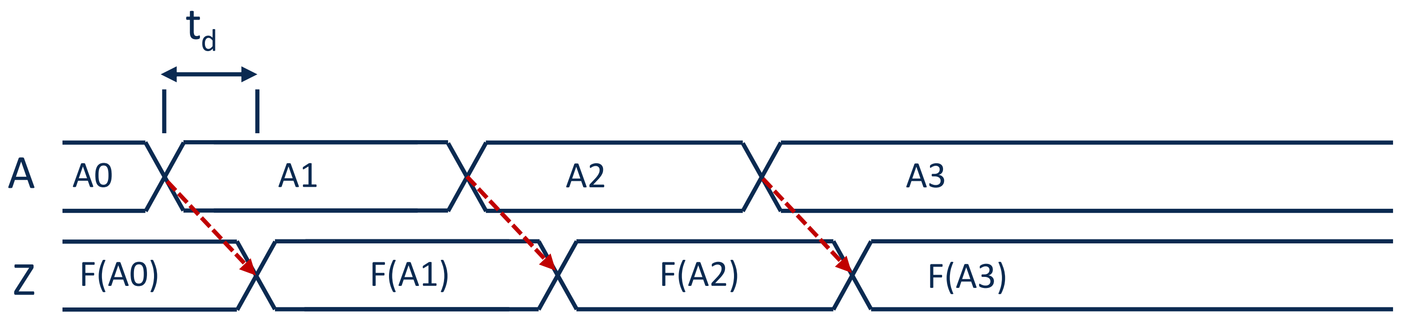
```
Entered by truthtable:
F1 = A' B C' D + A' B C D';
F2 = A' B' C' D' + A' B' C' D + A' B' C D' + A' B' C D;

Minimized:
F1 = A' B C D' + A' B C' D;
F2 = A' B' ;
```

Source: www.sontrak.com

TECHNISCHE UNIVERSITÄT DRESDEN

Time Behavior of Combinational Circuits

- The output value changes **when** the input values **change** after a delay time t_d (Delay)
- The ideal delay time is $t_d=0$
- Signal paths in circuits with multiple inputs and outputs may have different individual delays.

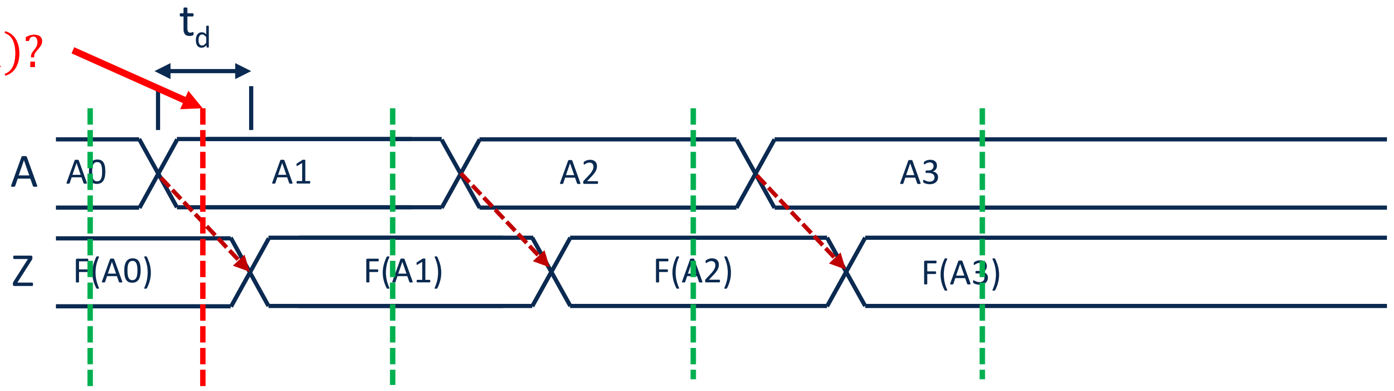


- Combinational circuits **do not include clocked memories**
- $Z = F(A)$

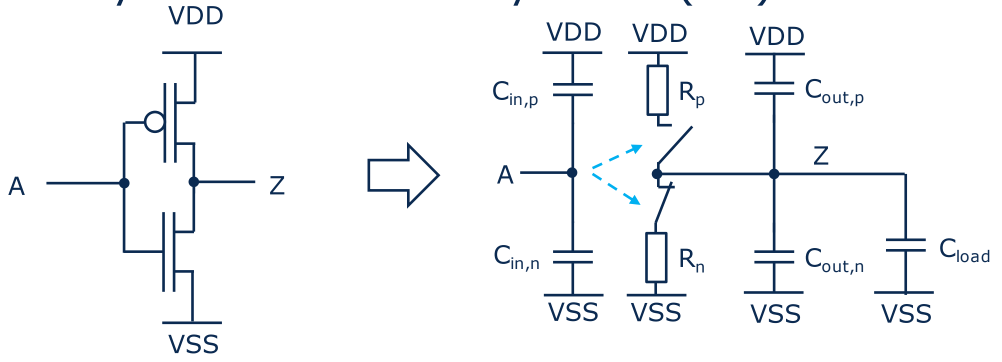
TECHNISCHE UNIVERSITÄT DRESDEN

Memory and Delay

- $Z = F(A)?$



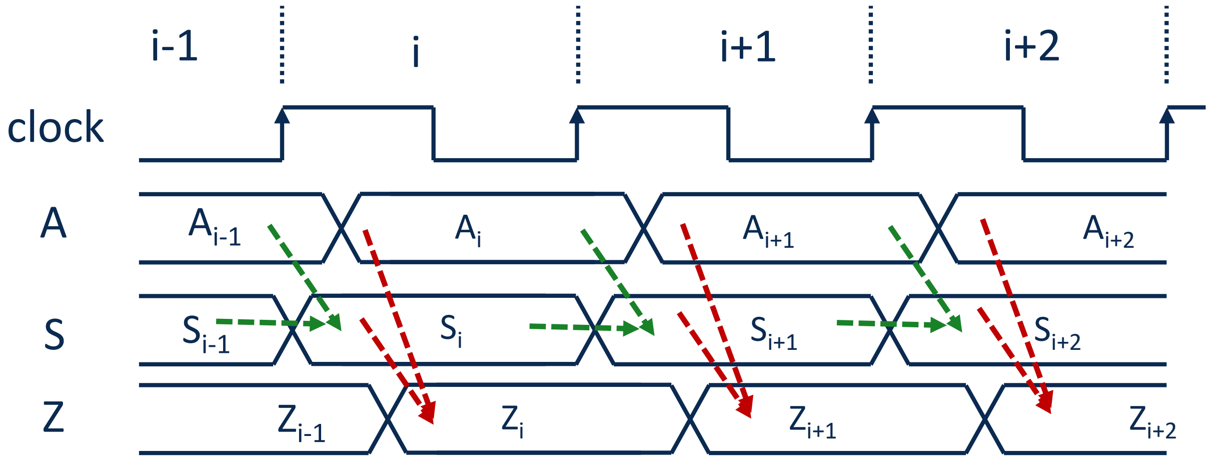
- Combinational circuits **do not include clocked memories**
- **BUT:** They have dynamic electrical systems (RC) with memory



- → Observing signals at **defined times** when $Z = F(A)$
- → Application of combinational logic theory **without memory.**

- Sequential logic circuits **include clocked memory**
- Memory state S
- Considering the system at times (i,i+1,i+2,…), e.g. realised by a clocked signal <u>(clock)</u>

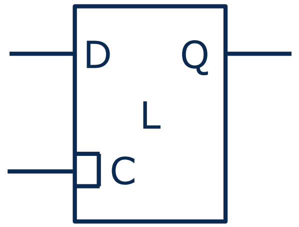
$$\boxed{\begin{aligned} S_{i+1} &= G(A_i, S_i) \\ Z_i &= F(A_i, S_i) \end{aligned}}$$

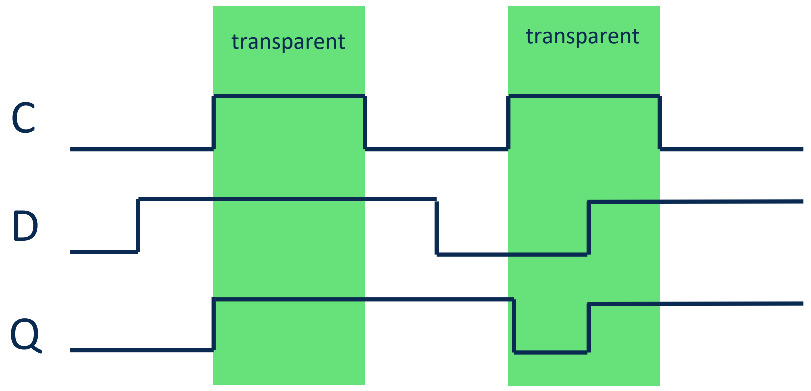


TECHNISCHE UNIVERSITÄT DRESDEN

Level-triggered Memory: D-Latch

- Level-triggered storage element
- Inputs:
  - D: Data input
  - C: Clock (alternatively also E: Enable)
    - C=1 turns the latch transparent (Memory is written)
- Output:
  - Q: Data Output

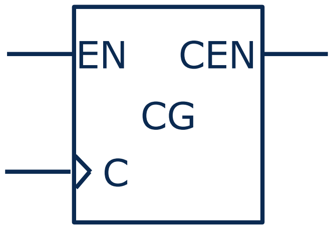


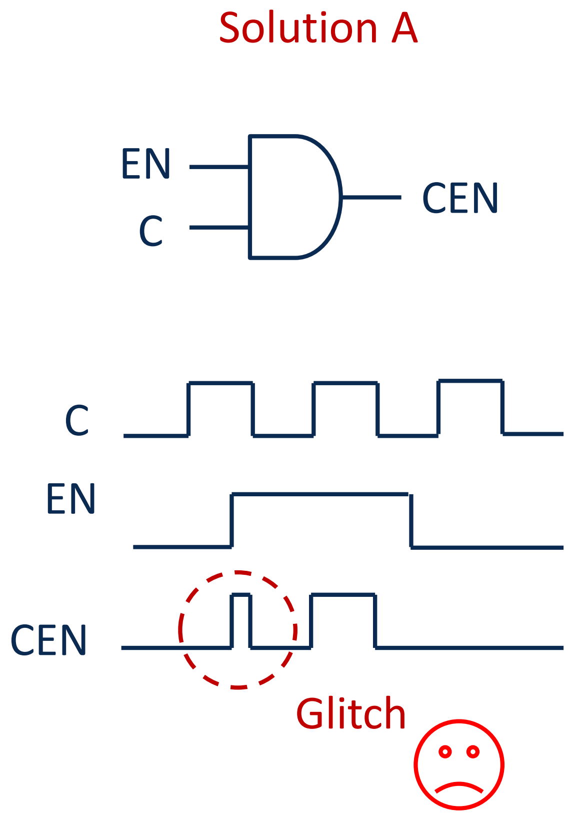


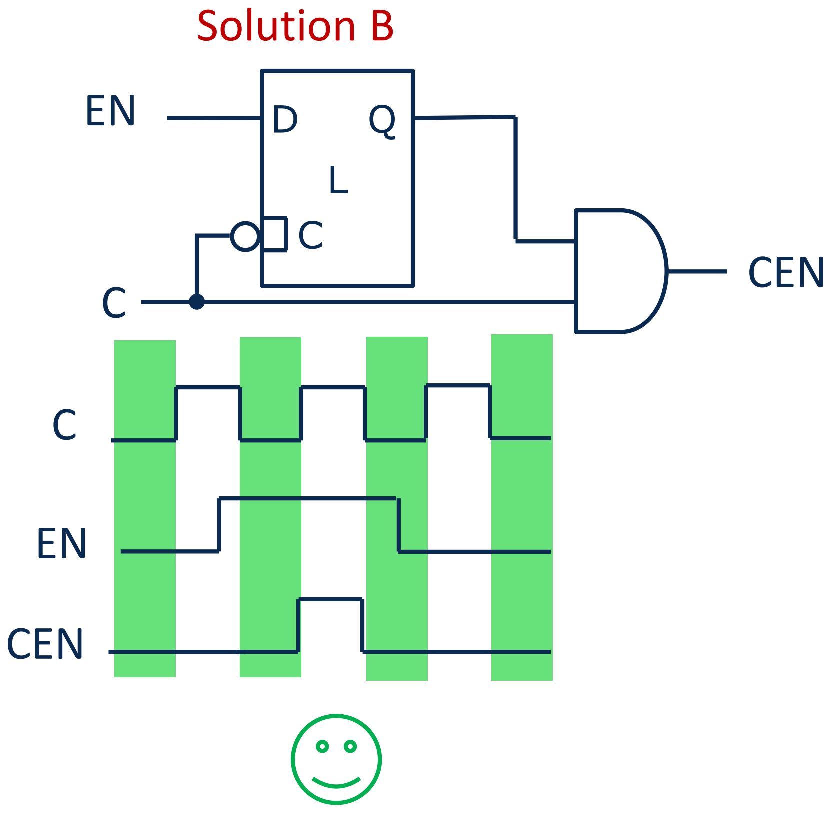
D	C	Q
0	1	0
1	1	1
X	0	Q_{i-1}

Example: Clock Gate

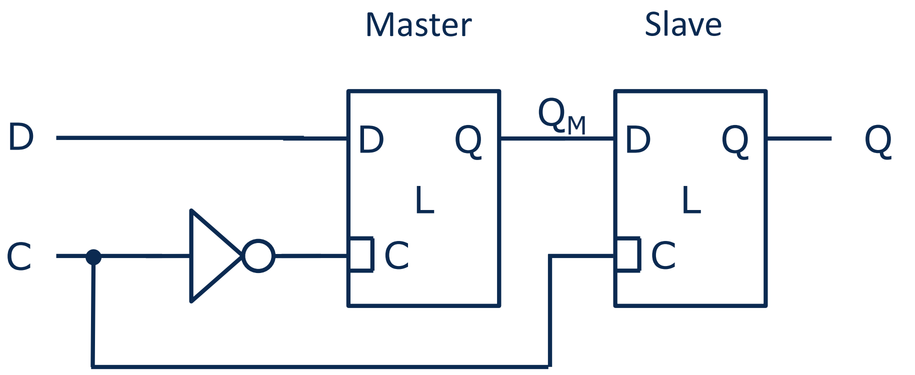
- Interrupting the clock signal by a control signal EN (enable)EN=0 → CEN=0
- Reduction of power consumption of temporarily unused circuit parts

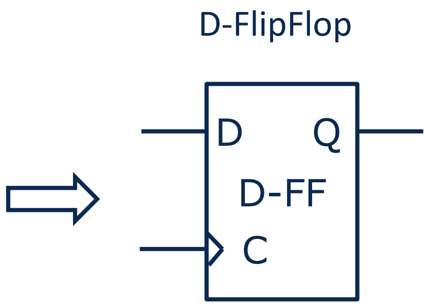


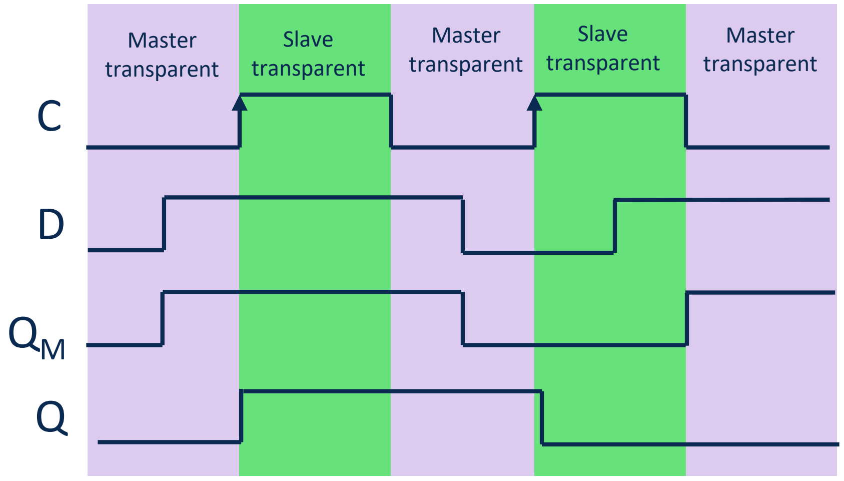




Edge-triggered Memory: Master-Slave FlipFlop





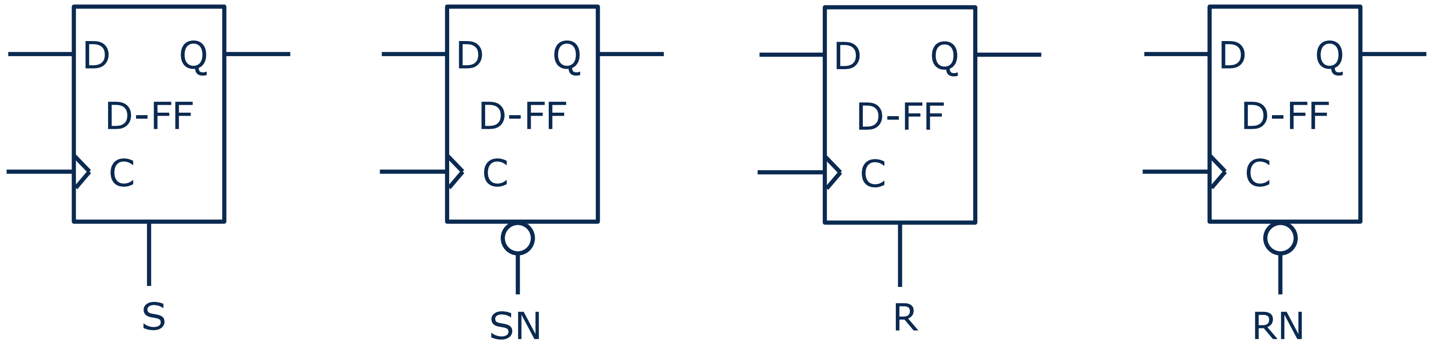


D	C	Q
0	↑	0
1	↑	1
X	↓	Q_{i-1}

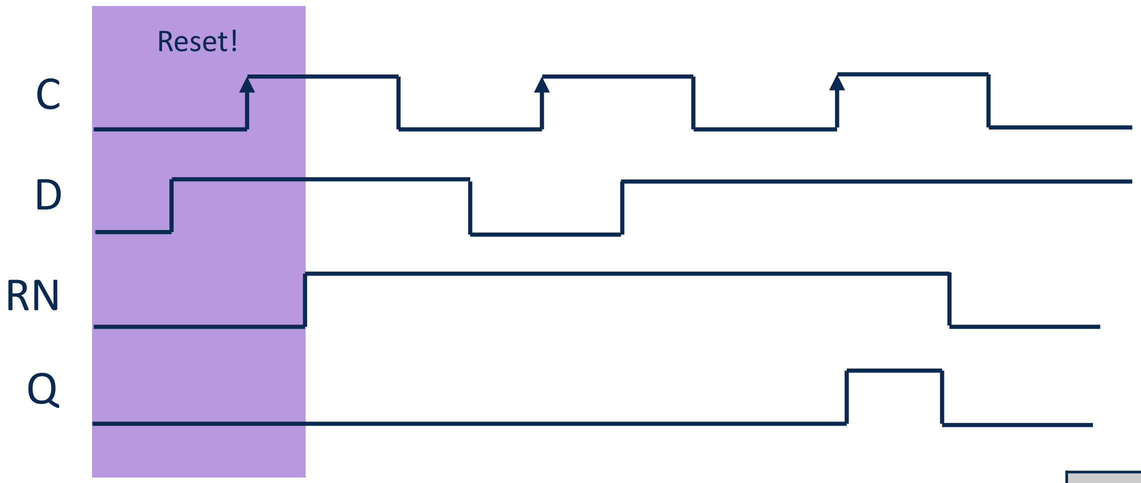
TECHNISCHE UNIVERSITÄT DRESDEN

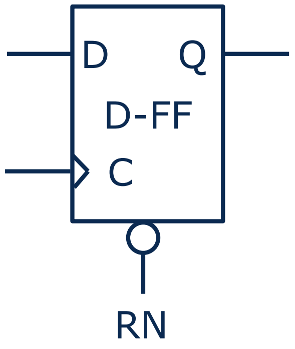
Asynchrones Set and Reset

- Additional asynchronous input for defining an initial state **independent of Clock C**
- Possible with latches and flipflops
- Only active to one logic level (active-high (1), active low (0))
  - Active-high Set S: active when S=1 → Q_0= 1
  - Active-low Set SN: active when SN=0 → Q_0= 1
  - Active-high Reset R: active when R=1 → Q_0= 0
  - Active-low Reset RN: active when RN=0 → Q_0= 0



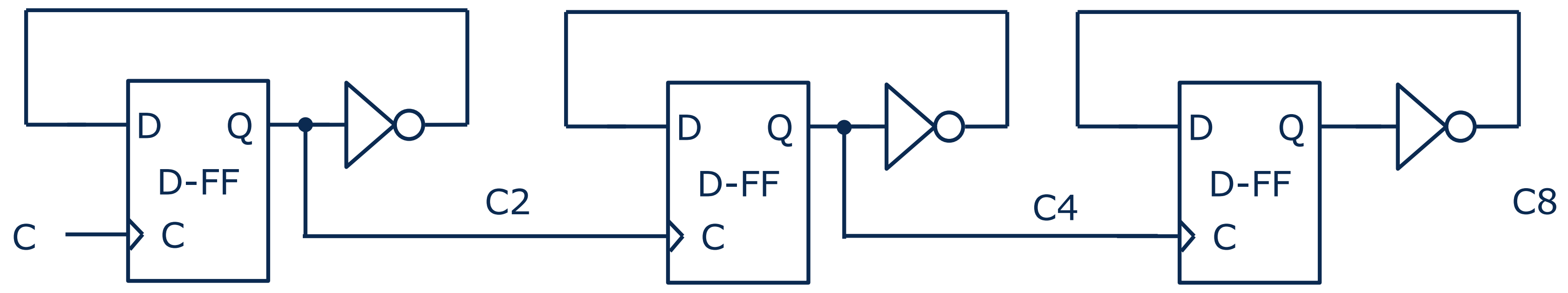
Example: D-FlipFlop with asynchronous active-low Reset

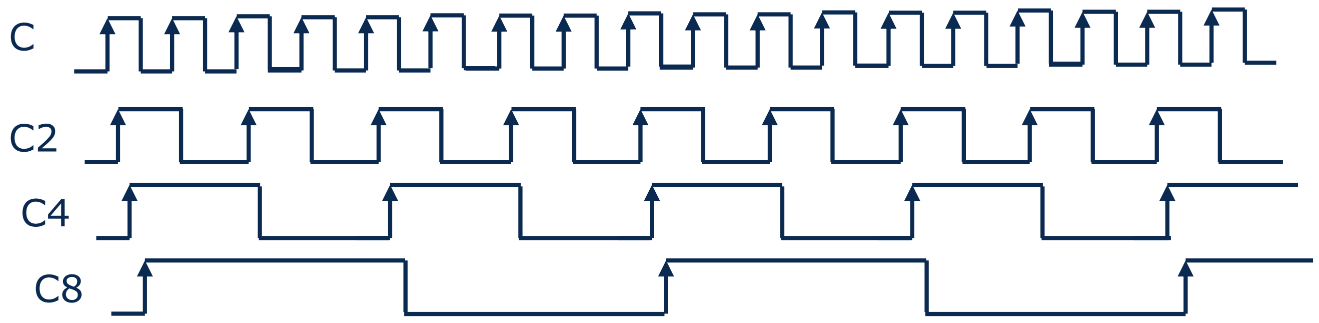




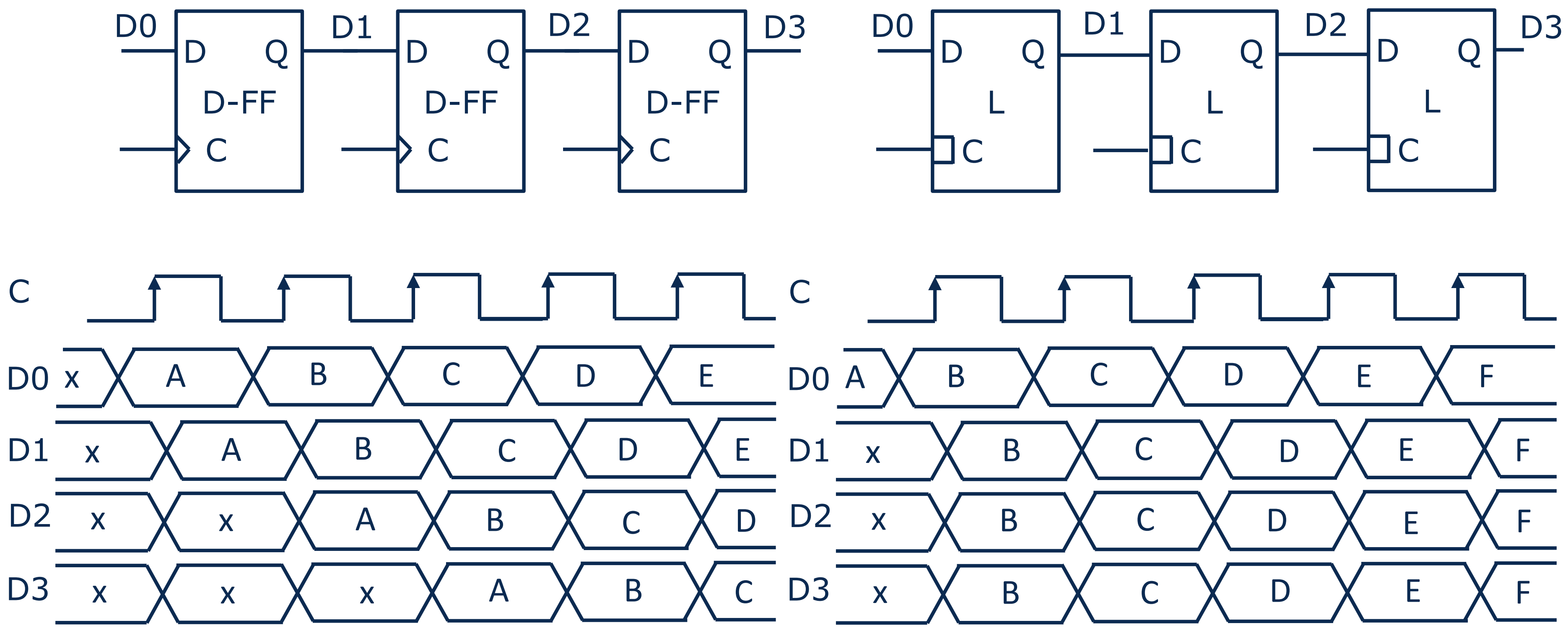
RN	D	C	Q
1	0	↑	0
1	1	↑	1
1	X	↓	Q_{i-1}
0	X	X	0

Example: Binary Frequency Divider



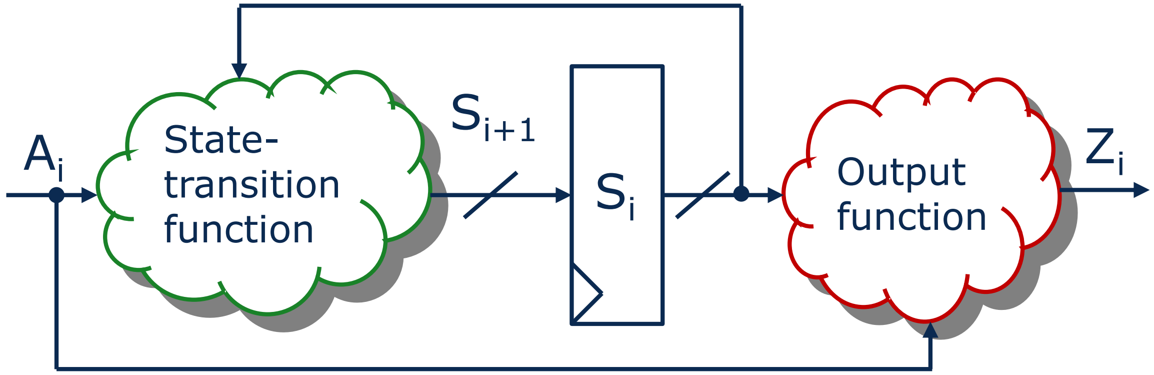


Example: Shift Register

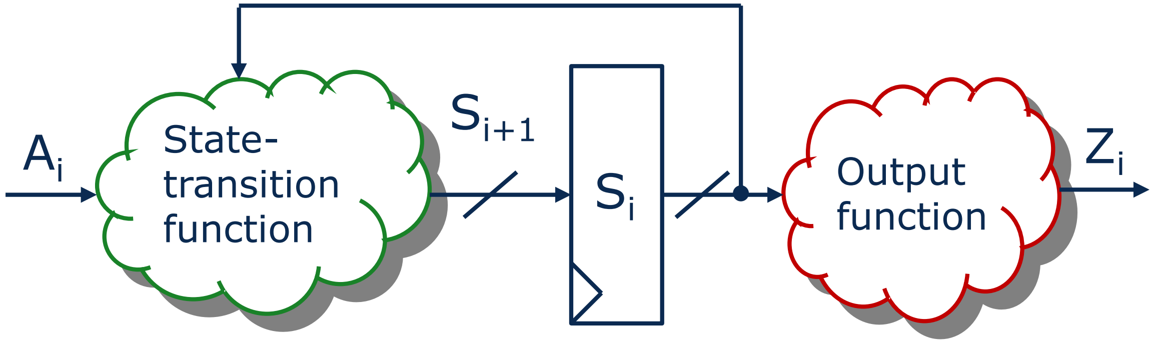


Finite State Machine (FSM)

- Finite-State Machines (FSM) are the basic components of digital control units
- FSMs are sequential digital systems
- **Mealy Machine**
  - $S_{i+1} = G(A_i, S_i)$
  - $Z_i = F(A_i, S_i)$

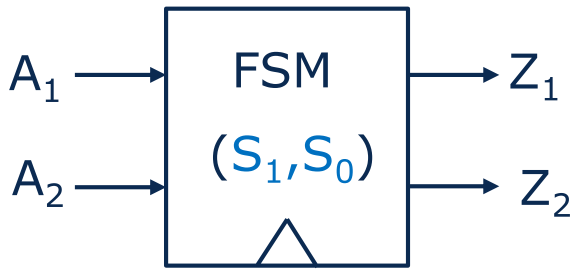


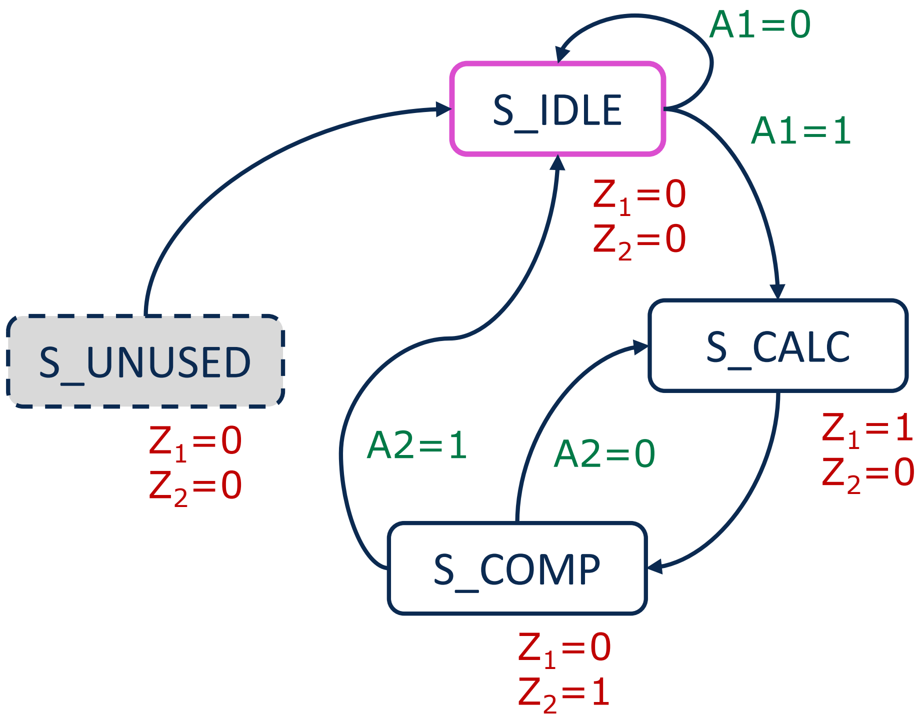
- **Moore Machine**
  - $S_{i+1} = G(A_i, S_i)$
  - $Z_i = F(S_i)$



State-transition Diagram

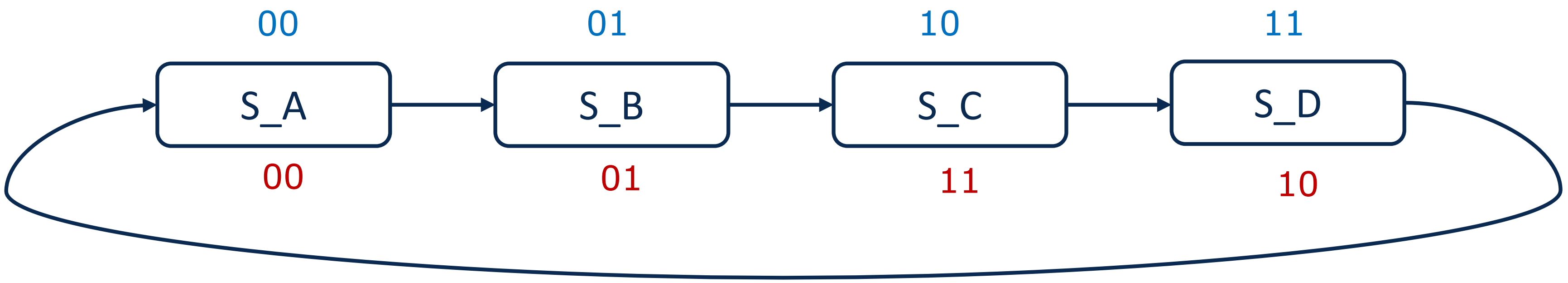
- Representation of states and their transitions
- Unique label for each state
- Marking of the Reset state



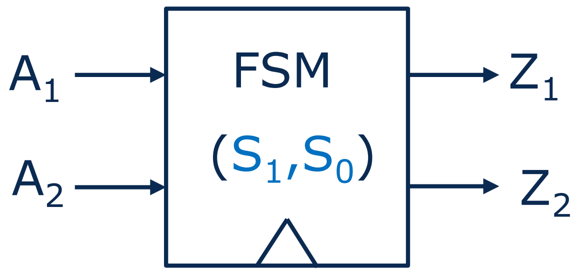


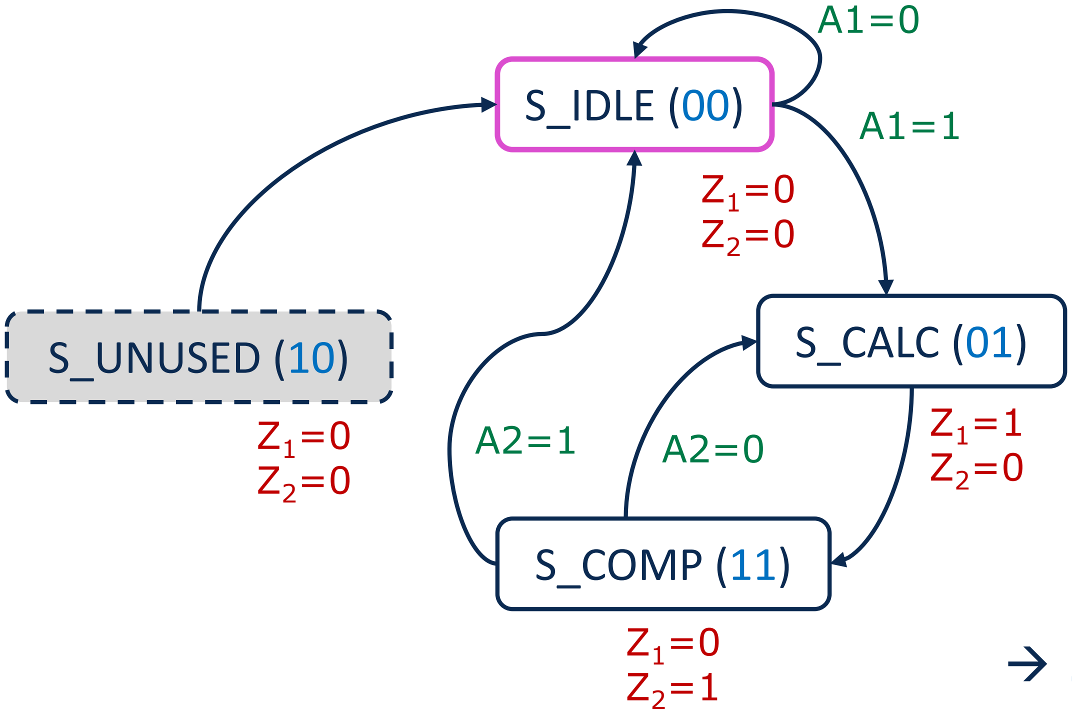
State Coding

- Assignment of a unique binary word for each state
  - With m states → N state bits with $2^N \geq m$
  - Coding of the 2^N-m unused states as well!
- State coding affects:
  - Power consumption
    - → Reduction of signal changes in the state word for the most frequently expected routes through the state graph
  - Complexity of state transition logic and output logic
- Example:
  - Binary-Code → 6 Toggles in **S** per flow
  - Gray-Code → 4 Toggles in **S** per flow



State-transition Coding





→ 2-Bit state variable

State-transition and Output Tables

$S_{1,i}$	$S_{0,i}$	A_2	A_1	$S_{1,i+1}$	$S_{0,i+1}$
0	0	X	0	0	0
0	0	X	1	0	1
0	1	X	X	1	1
1	0	X	X	0	0
1	1	0	X	0	1
1	1	1	X	0	0

$S_{1,i}$	$S_{0,i}$	$Z_{2,i}$	$Z_{1,i}$
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

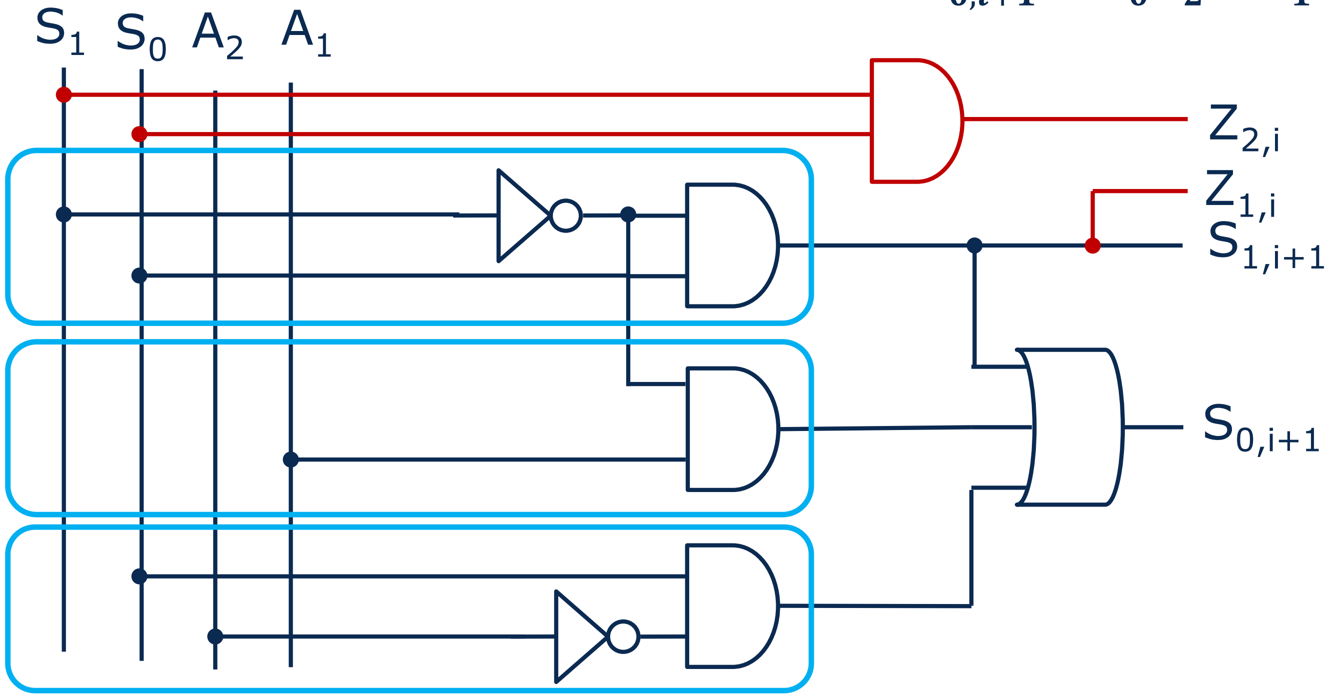
Simplification of Logic and Gate Netlist

$S_{1,i+1}$	S_1S_0	$\overline{S_1}S_0$	$\overline{S_1}\,\overline{S_0}$	$S_1\overline{S_0}$
A_1A_2	0	1	0	0
$\overline{A_1}A_2$	0	1	0	0
$\overline{A_1A_2}$	0	1	0	0
$A_1\overline{A_2}$	0	1	0	0

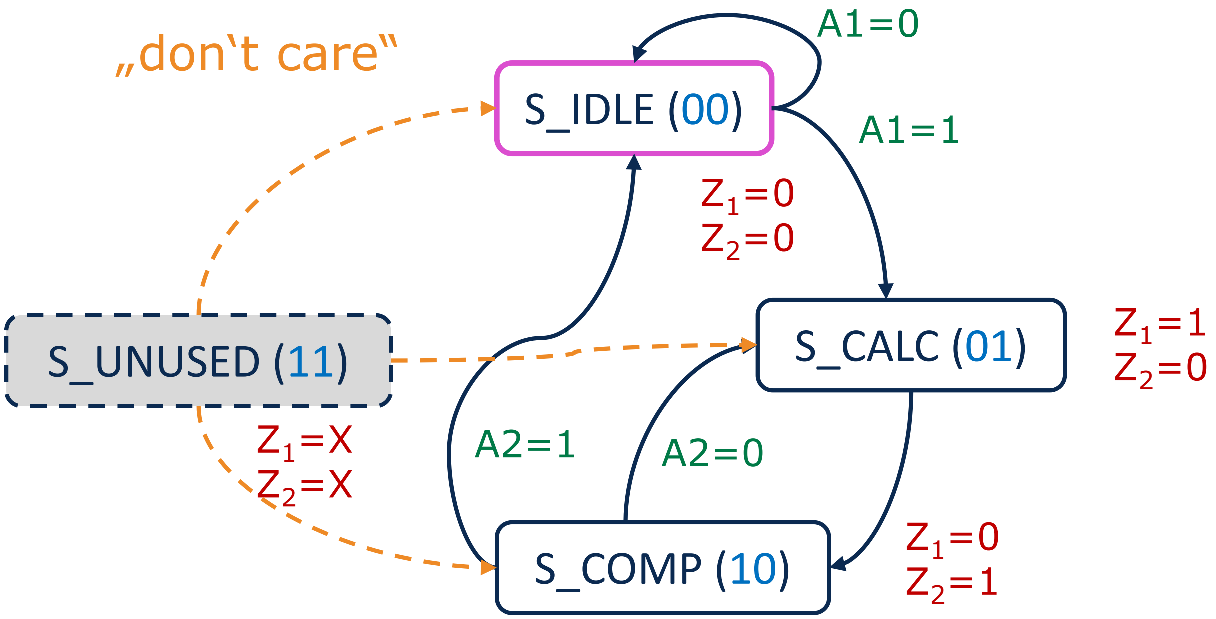
$S_{0,i+1}$	S_1S_0	$\overline{S_1}S_0$	$\overline{S_1}\,\overline{S_0}$	$S_1\overline{S_0}$
A_1A_2	0	1	1	0
$\overline{A_1}A_2$	0	1	0	0
$\overline{A_1A_2}$	1	1	0	0
$A_1\overline{A_2}$	1	1	1	0

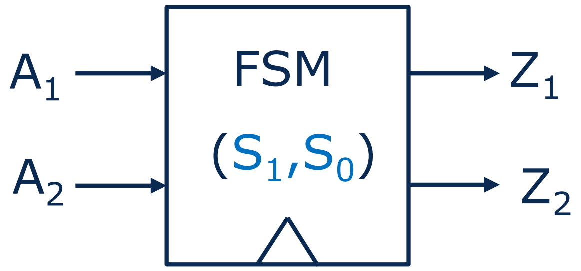
$$\boldsymbol{S_{1,i+1} = \overline{S_1}S_0}$$

$$\boldsymbol{S_{0,i+1} = S_0\overline{A_2} + \overline{S_1}S_0 + \overline{S_1}A_1}$$



Alternative State-transition Coding





$S_{1,i}$	$S_{0,i}$	A_2	A_1	$S_{1,i+1}$	$S_{0,i+1}$
0	0	X	0	0	0
0	0	X	1	0	1
0	1	X	X	1	0
1	0	0	X	0	1
1	0	1	X	0	0
1	1	X	X	X	X

$S_{1,i}$	$S_{0,i}$	$Z_{2,i}$	$Z_{1,i}$
0	0	0	0
0	1	0	1
1	0	1	0
1	1	X	X

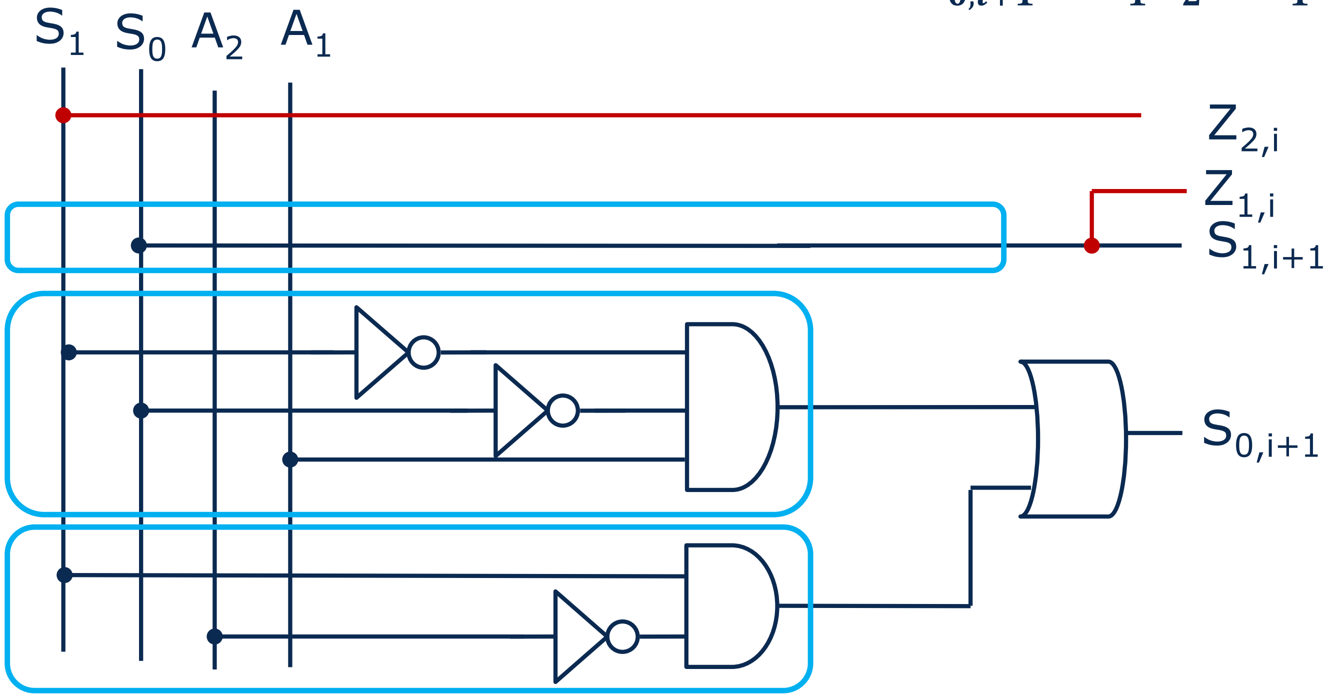
Alternative Simplification of Logic and Gate Netlist

$S_{1,i+1}$	S_1S_0	$\overline{S_1}S_0$	$\overline{S_1}\,\overline{S_0}$	$S_1\overline{S_0}$
A_1A_2	X	1	0	0
$\overline{A_1}A_2$	X	1	0	0
$\overline{A_1A_2}$	X	1	0	0
$A_1\overline{A_2}$	X	1	0	0

$S_{0,i+1}$	S_1S_0	$\overline{S_1}S_0$	$\overline{S_1}\,\overline{S_0}$	$S_1\overline{S_0}$
A_1A_2	X	0	1	0
$\overline{A_1}A_2$	X	0	0	0
$\overline{A_1A_2}$	X	0	0	1
$A_1\overline{A_2}$	X	0	1	1

$$S_{1,i+1} = S_0$$

$$S_{0,i+1} = S_1\overline{A_2} + \overline{S_1S_0}A_1$$



Summary: FSM Design

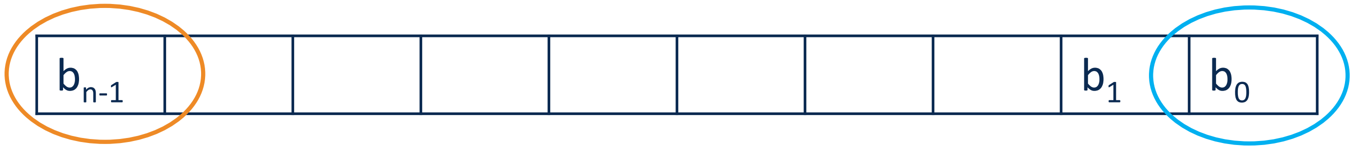
1. Set up state-transition diagram
2. Determine the number of state bits and state coding
3. State-transition table and output table
4. Simplification of logic (Karnaugh, equation)
5. Create a gate netlist

Arithmetic - Number Formats

Binary Numbers

- Representation of unsigned integers as n-bit vector
- $b \in (0;1)$
- Decimal value: $D = \sum_{i=0}^{n-1} b_i \cdot 2^i$

- Value range: $0 \leq D \leq 2^n - 1$



„most significant bit“ (MSB)

„least significant bit“ (LSB)

Representations

Decimal	Binary	Hexadecimal
0	0b0000	0x0
1	0b0001	0x1
2	0b0010	0x2
3	0b0011	0x3
4	0b0100	0x4
5	0b0101	0x5
6	0b0110	0x6
7	0b0111	0x7
8	0b1000	0x8
9	0b1001	0x9
10	0b1010	0xA
11	0b1011	0xB
12	0b1100	0xC
13	0b1101	0xD
14	0b1110	0xE
15	0b1111	0xF

TECHNISCHE UNIVERSITÄT DRESDEN

Addition of Binary Numbers

- 1-bit Addition a+b

a	b	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Example Addition:

		0	1	0	1	(5)
	+	1	1	0	1	(13)
Carry	1	1	0	1		
Sum		0	0	1	0	(2)

- Addition of n-bit numbers → n+1 bit output

Multiplication of Binary Numbers

- 1-bit Multiplication a·b

a	b	a ·b
0	0	0
0	1	0
1	0	0
1	1	1

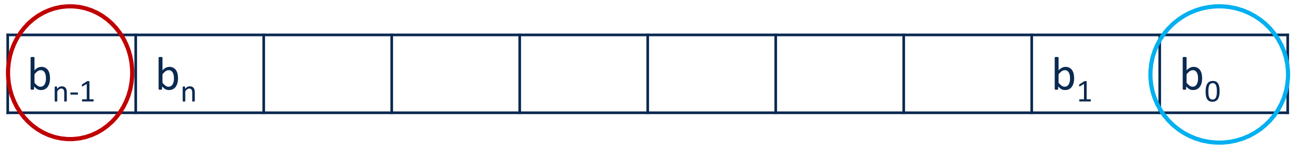
- Example: 0b1001 · 0b0101 (9·5)

					1	0	0	1	
					1	0	0	1	1
+				0	0	0	0		0
+			1	0	0	1			1
+		0	0	0	0				0
Product	0	0	1	0	1	1	0	1	(45)

- Multiplication of n-bit numbers → 2n-bit output

Signed Binary Numbers

- Representation of signed integer numbers as n-bit vector
- $b \in (0; 1)$
- 2‘s complement representation
- Decimal value:
  - positive number: when b_{n-1}=0: $D = \sum_{i=0}^{n-1} b_i \cdot 2^i$
  - negative number: when b_{n-1}=1: $D = -1 \cdot (\sum_{i=0}^{n-1} \overline{b_i} \cdot 2^i + 1)$
- Value range: $-2^{n-1} \leq D \leq 2^{n-1} - 1$



„sign bit“

„least significant bit“ (LSB)

Examples

Decimal value	Binary Signed (2‘s complement)
0	0b0000
1	0b0001
2	0b0010
3	0b0011
4	0b0100
5	0b0101
6	0b0110
7	0b0111
-8	0b1000
-7	0b1001
-6	0b1010
-5	0b1011
-4	0b1100
-3	0b1101
-2	0b1110
-1	0b1111

Signed Binary Numbers

- calculation of $-1 \cdot A$
  1. Inversion of all bits of A
  2. Addition of +1

- Example:
  - -1·3 =-1·(0b0011)→ 0b1100+0b0001=0b1101 =-3
  - -1·(-3) =-1·(0b1101)→ 0b0010+0b0001=0b0011 = 3

Addition/Subtraction of Signed Numbers

- Example 1:

		0	0	1	0	(2)
	+	1	1	1	0	(−2)
Carry	1	1	1	0		
Sum		0	0	0	0	(0)

OK

- Example 2:

		1	1	1	0	(−2)
	+	1	1	1	0	(−2)
Carry	1	1	1	0		
Sum		1	1	0	0	(−4)

OK

- Example 3:

		0	1	0	0	(4)
	+	0	1	1	0	(6)
Carry	0	1	0	0		
Sum		1	0	1	0	(−6)

Error
Addition of pos. numbers must give pos. result!

- Example 4:

		1	1	0	0	(−4)
	+	1	0	1	0	(−6)
Carry	1	0	0	0		
Sum		0	1	1	0	(6)

Error
Addition of neg. numbers must give neg. result!

Carry and Overflow

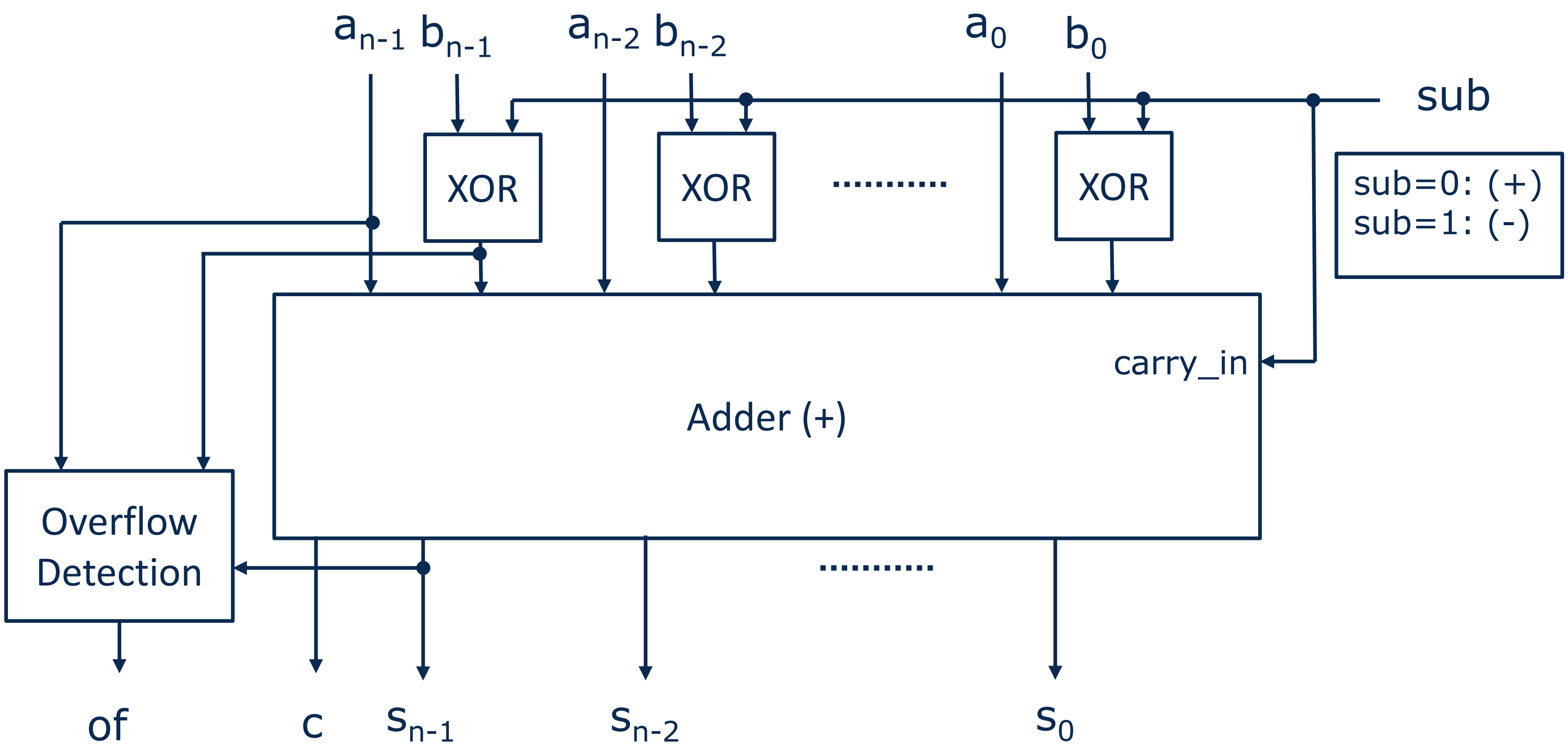
- Carry: Carry out the highest binary digit (MSB) in the **unsigned** range
- Overflow: Detection of the **signed** range overflows

Sign(A)	Sign(B)	Sign(A+B)	Overflow
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Decimal value	Unsigned representation	Signed represnetation
Carry Bit	**Error**	not relevant
Overflow Bit	not relevant	**Error**

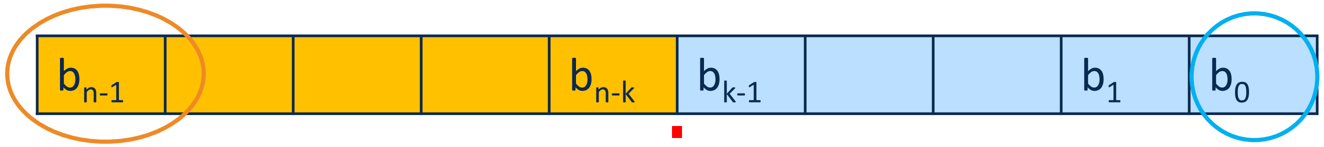
Addition/Subtraction of Signed Numbers

- Addition and Subtraction can be done with the same circuit



Fixed-Point Representation

- Representation of numbers with decimal point and decimal places as n-bit vector
- k decimal und n-k predecimal
- Values of LBS: 2^{-k}
- Decimal value: $$D = \sum_{i=0}^{n-1} b_i \cdot 2^{i-k}$$
- Value range: $0 \leq D \leq (2^n - 1) \cdot 2^{-k}$



„most significant bit“ (MSB) „Point“ „least significant bit“ (LSB)

Representation of signed fixed-point numbers analogous to integers (scaling with 2^{-k})

Fixed-Point Addition and Subtraction

- Addition/Subtraction of n-bit Fixed-Point numbers:
  - $A=a\cdot 2^{-k}$, $B=b\cdot 2^{-k} \rightarrow A\pm B=(a\pm b)\cdot 2^{-k}$
  - (n+1)-bit output

- Example 1:
  - n=4, k=2, 2^{-k}=0.25, unsigned

			0	1	1	0	(1.50)
	+		1	0	0	1	(2.25)
Carry		0	0	0	0		
Sum			1	1	1	1	(3.75)

- Example 2:
  - n=4, k=2, 2^{-k}=0.25, signed

			0	1	1	0	(1.50)
	+		1	0	1	1	(−1.25)
Carry		1	1	1	0		
Sum			0	0	0	1	(0.25)

Fixed-Point Multiplication

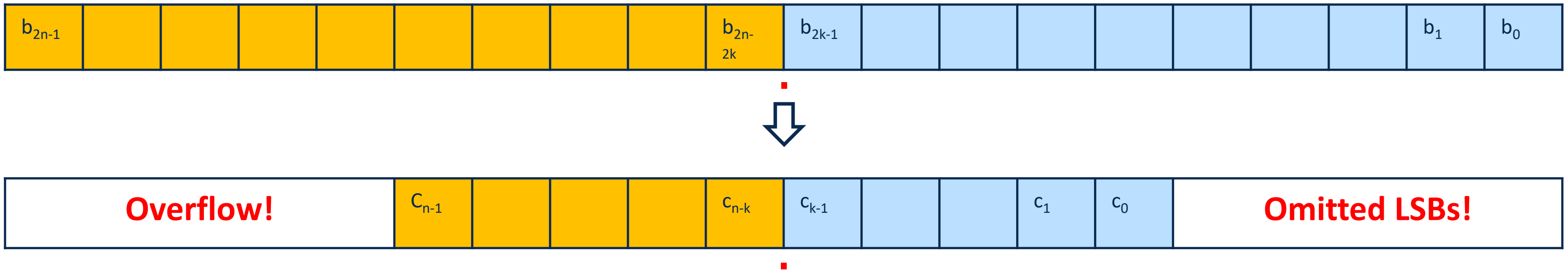
- Multiplication of n-bit Fixed-Point numbers:
  - $A=a\cdot2^{-k}$, $B=b\cdot2^{-k} \rightarrow A\cdot B=(a\cdot b)\cdot2^{-2k}$
  - $\rightarrow$ Doubling the number of decimal places
  - 2n-Bit output with 2^{-2k} LSB accuracy

- Example:
  - n=4, k=2, 2^{-k}=0.25, unsigned
  - 0b1001 · 0b0101 (2.25 · 1.25)

					1	**0**	**0**	**1**	
					1	0	0	1	**1**
+				0	0	0	0		**0**
+			1	0	0	1			**1**
+		0	0	0	0				**0**
Product	0	0	1	0	1	1	0	1	(2.8125)

Output Scaling

- Multiplication of n-bit values → 2n-bit output
- For n-Bit data buses, scaling is necessary c=Scale(b)



- In SSE lab excercise:
  - **SCALER64_to_32**: Determination of overflows (signed, unsigned)
  - Indication of the number k LSBs
  - Use only when overflow signals are needed
  - Otherwise: direct wiring of the bus signals

Fixed-Point Division

- Division of n-bit Fixed-Point numbers:
  - $A=a\cdot 2^{-k}$, $B=b\cdot 2^{-k} \rightarrow A/B=(a/b)\cdot 2^{-k+k}=a/b$
- For n-bit operands:
  - n-bit integer result + n-bit **Remainder (Modulo)**
    - Example: 0b1101 / 0b0011 (13/3) = 0b0100 Remainder: 0b0001
    - SSE lab : **DIV_FIXED32_signed** and **DIV_FIXED32_unsigned**
  - n-bit integer result + infinitely many decimal places
    - Example: 0b1101 / 0b0011 (13/3) =0b0100,010101… (4,33333…)
    - In SSE lab : **DIV_FIXED64_signed (**32 predecimal, and decimal, limited accuracy!)

Floating Point Representation

- Problems with Fixed-Point representation:
  - Limited value range
  - Fixed-Point: Compromise between value range and accuracy

- Number representation in the form: $\boxed{X = S \cdot M \cdot 2^{E}}$
  - Basis: 2
  - S: sign → 1-Bit
  - M: mantissa → unsigned fixed-point
  - E: Exponent (variable) → signed integer

- Very high precision with small numbers
- Very large value range for large numbers (with lower precision)
- Floating point representations are approximations!

Floating Point Example

- Example: Number 5 in floating point form
  - $5 = +1 \cdot 1.25 \cdot 2^{2}$
  - S=0
  - M=0b1,01_0000_0000_0000_0000_0000
  - E=0b00000010

- But also: $5 = +1 \cdot 0.625 \cdot 2^{3}$, $5 = +1 \cdot 0.3125 \cdot 2^{4}$, $5 = +1 \cdot 0.15625 \cdot 2^{5}$, …
- The floating-point representations are not uniquely determined

- → Normalization of the value range of Mantissa
- e.g. $1 \leq M < 2$

Bias

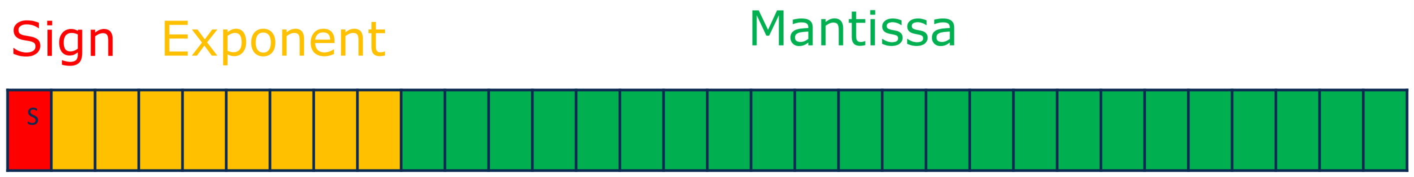
- The exponent of a floating point number is signed
- Hardware required for signed operations

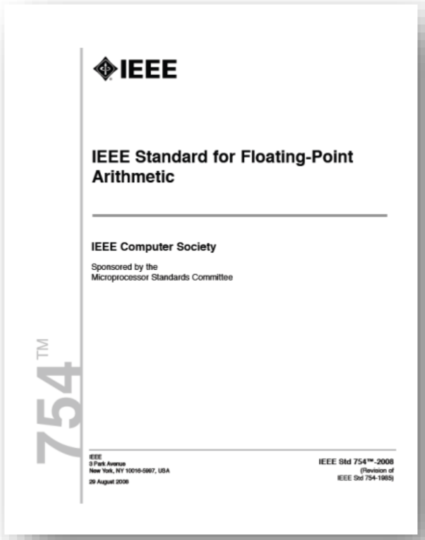
- Storing the exponent as E‵=E+B as an unsigned number
  - B: Bias
  - Example: 8-bit Exponent: B=127

- Enables use of unsigned arithmetic components for floating point units

IEEE 754 Floating Point Standard

- Defines floating point number formats and their representation in bits.




IEEE

IEEE Standard for Floating-Point Arithmetic

IEEE Computer Society

Sponsored by the
Microprocessor Standards Committee

754™

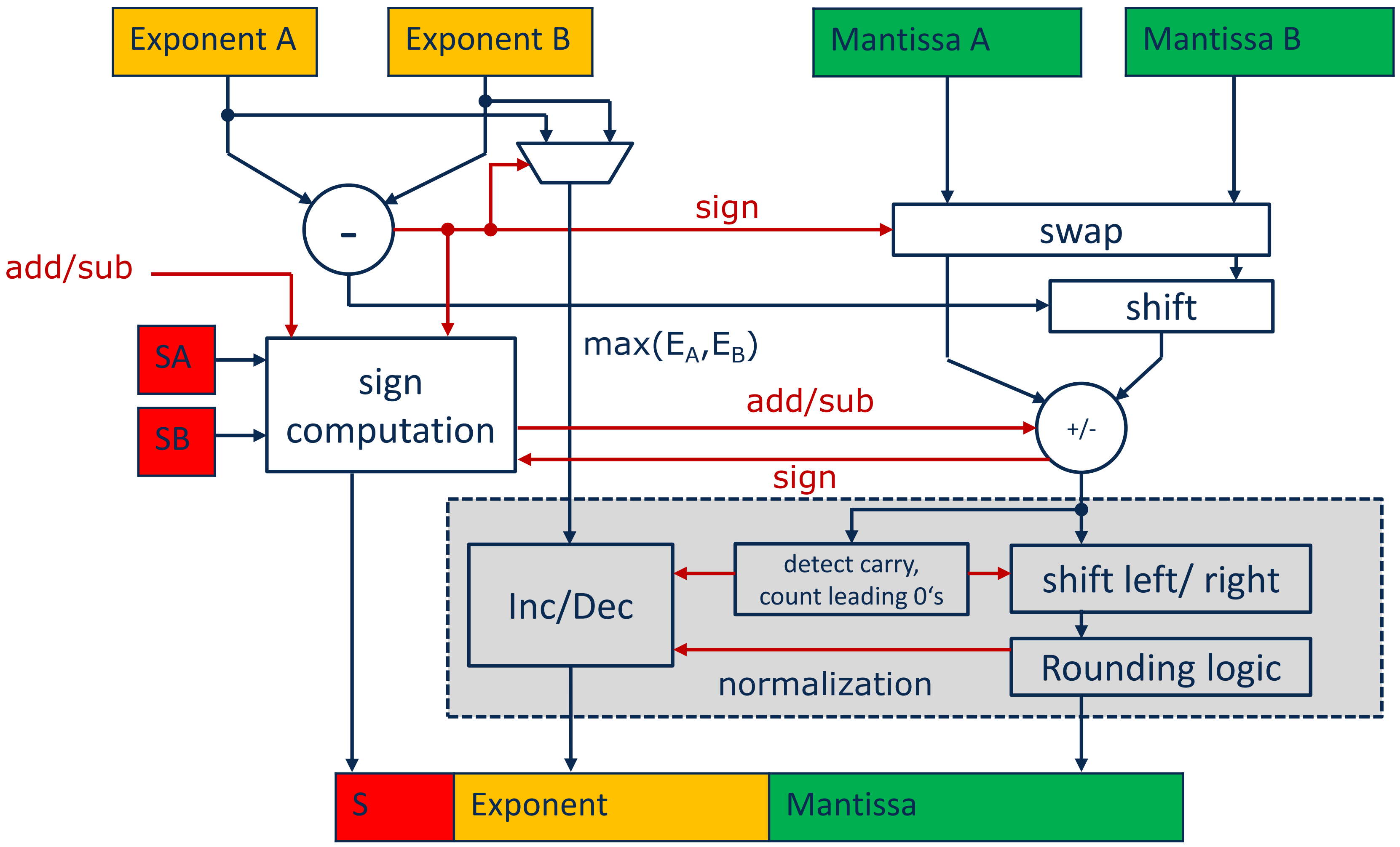
IEEE
3 Park Avenue
New York, NY 10016-5997, USA

29 August 2008

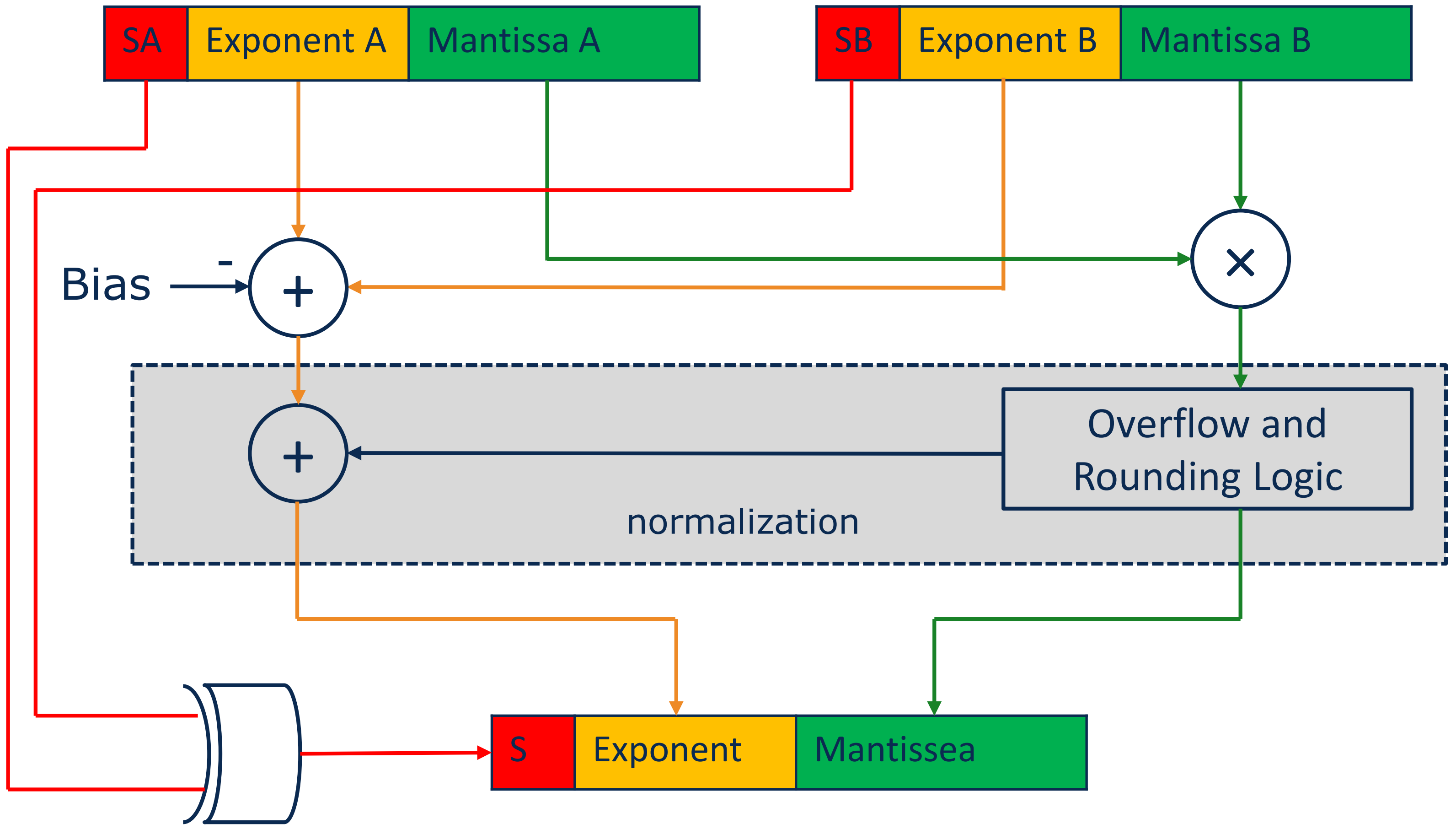
IEEE Std 754™-2008
(Revision of
IEEE Std 754-1985)

	Bits	Mantissa (Bit)	Exponent (Bit)	Bias	Value range	Accuracy (Decimal)
Single (float)	32	23	8	127	10^{-38} bis 10^{38}	≈7
Double	64	52	11	1023	10^{-308} bis 10^{308}	≈16

Floating Point Adder Block Diagram



Floating Point Multiplier Block Diagram



Summary

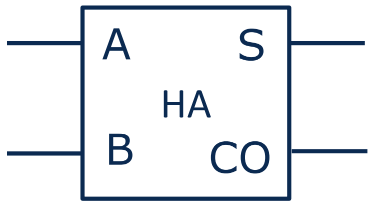
- Number formats fixed-point (singed, unsigned) and floating point
- Higher accuracy requires more effort regarding:
  - Higher storage requirements
  - Larger chip area for arithmetic
  - Longer logic runtimes → Frequently implementing complex arithmetic blocks in several clock cycles (pipelining)
  - Higher energy expenditure of the calculation

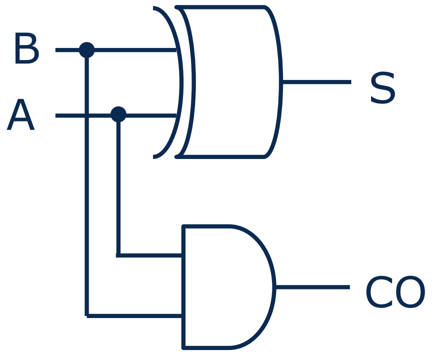
  - Choice of number format is critical to the efficiency of hardware implementation

Arithmetic - Circuits

Half Adder (HA)

- Addition of 2-bit
  - Inputs A, B
  - Output S (Sum), CO (Carry)





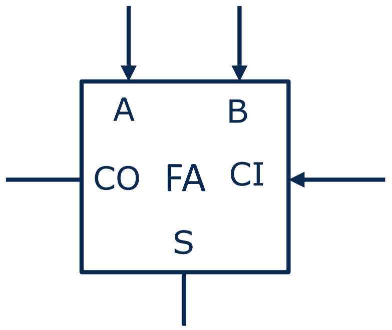
A	B	S	CO
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$\boldsymbol{S = A \oplus B}$$

$$\boldsymbol{CO = A \cdot B}$$

Full Adder (FA)

- Addition of 2 bit and Carry
  - Inputs A, B, CI (Carry In)
  - Outputs S (Sum), CO (Carry)



A	B	CI	S	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

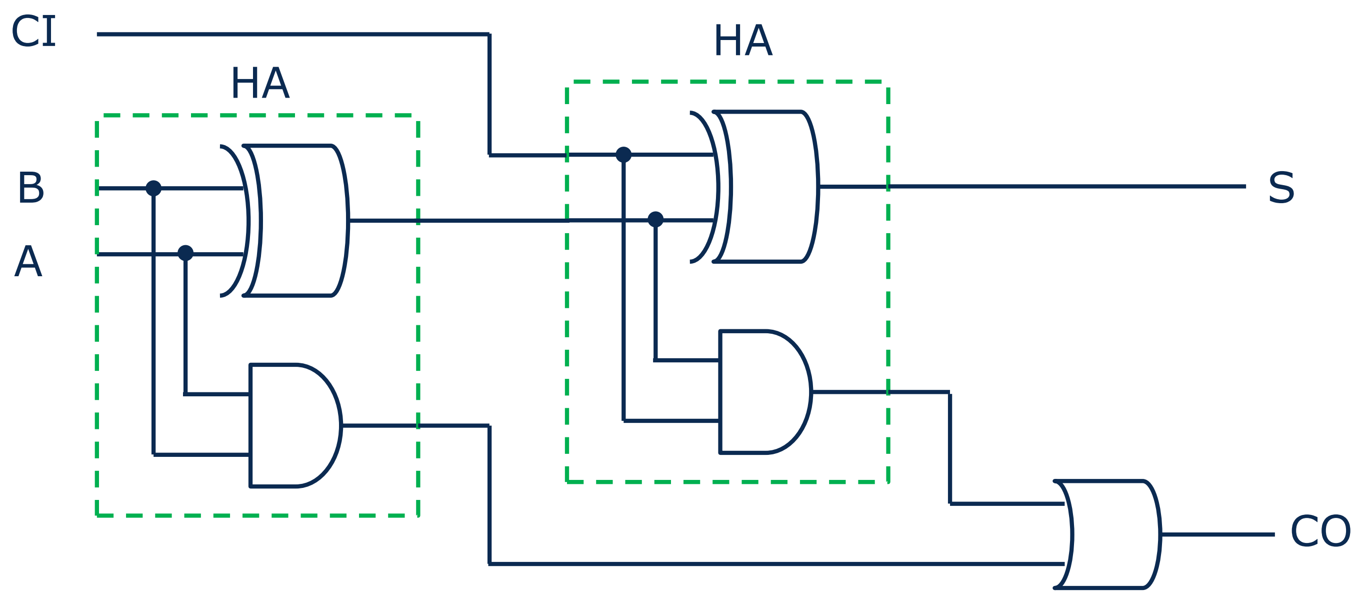
$$\boldsymbol{S = A \oplus B \oplus CI}$$

$$\boldsymbol{CO = A \cdot B + CI \cdot (A \oplus B)}$$

Full Adder Gate Circuit

$$S = A \oplus B \oplus CI$$

$$CO = A \cdot B + CI \cdot (A \oplus B)$$

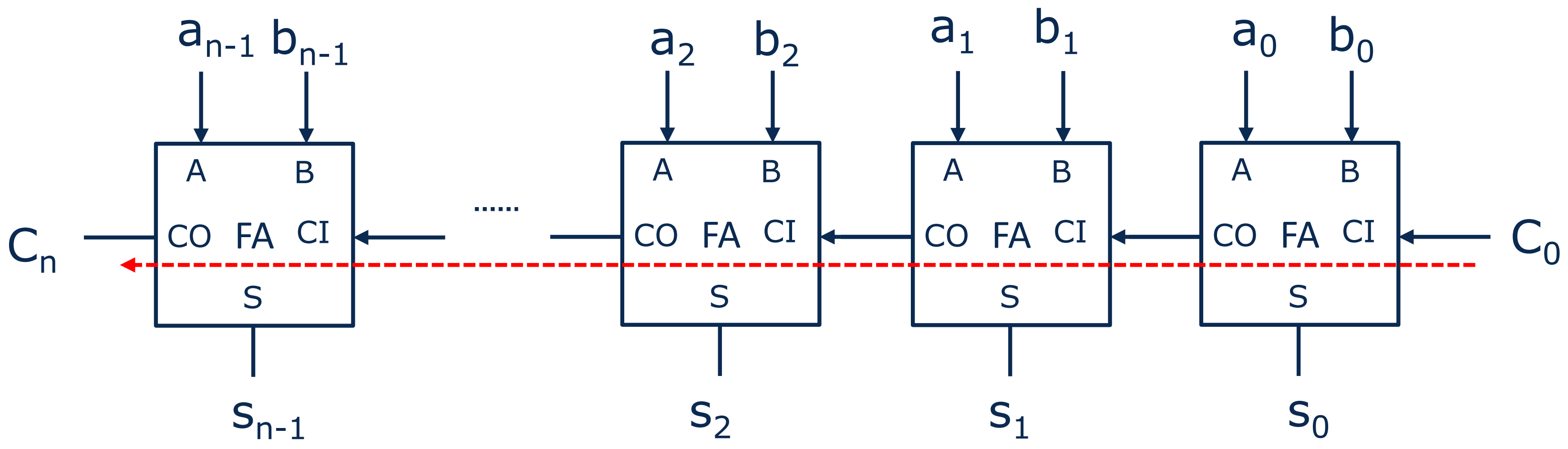


Carry path CI → CO: 2 gate stages

TECHNISCHE UNIVERSITÄT DRESDEN

Ripple Carry Adder

- Concatenation of full adders to add n-bit numbers
- If no CI is needed, first stage can be replaced with HA



Critical path: CI → CO: 2*n gate stages

- Example:
  - 32-Bit Adder → 64 gate delays
  - 28nm: $t_{gate} \approx 50ps$ → T=3,2ns → $f_{max} \approx 310MHz$
- State of the art: 32-bit processors at> 2GHz → How does it work?

Carry Generate and Propagate

- Carry paths are time critical!
- → Reduction of the logic levels in the carry path
- Introduce new values:
  - P: Propagate: Passing on the carry
  - G: Generate: Generating the carry

A	B	CI	S	CO	P	G
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	1	0	1	0
0	1	1	0	1	1	0
1	0	0	1	0	1	0
1	0	1	0	1	1	0
1	1	0	0	1	0	1
1	1	1	1	1	0	1

$$S = A \oplus B \oplus CI$$
$$CO = A \cdot B + CI \cdot (A \oplus B)$$

$$P = A \oplus B$$

$$G = A \cdot B$$

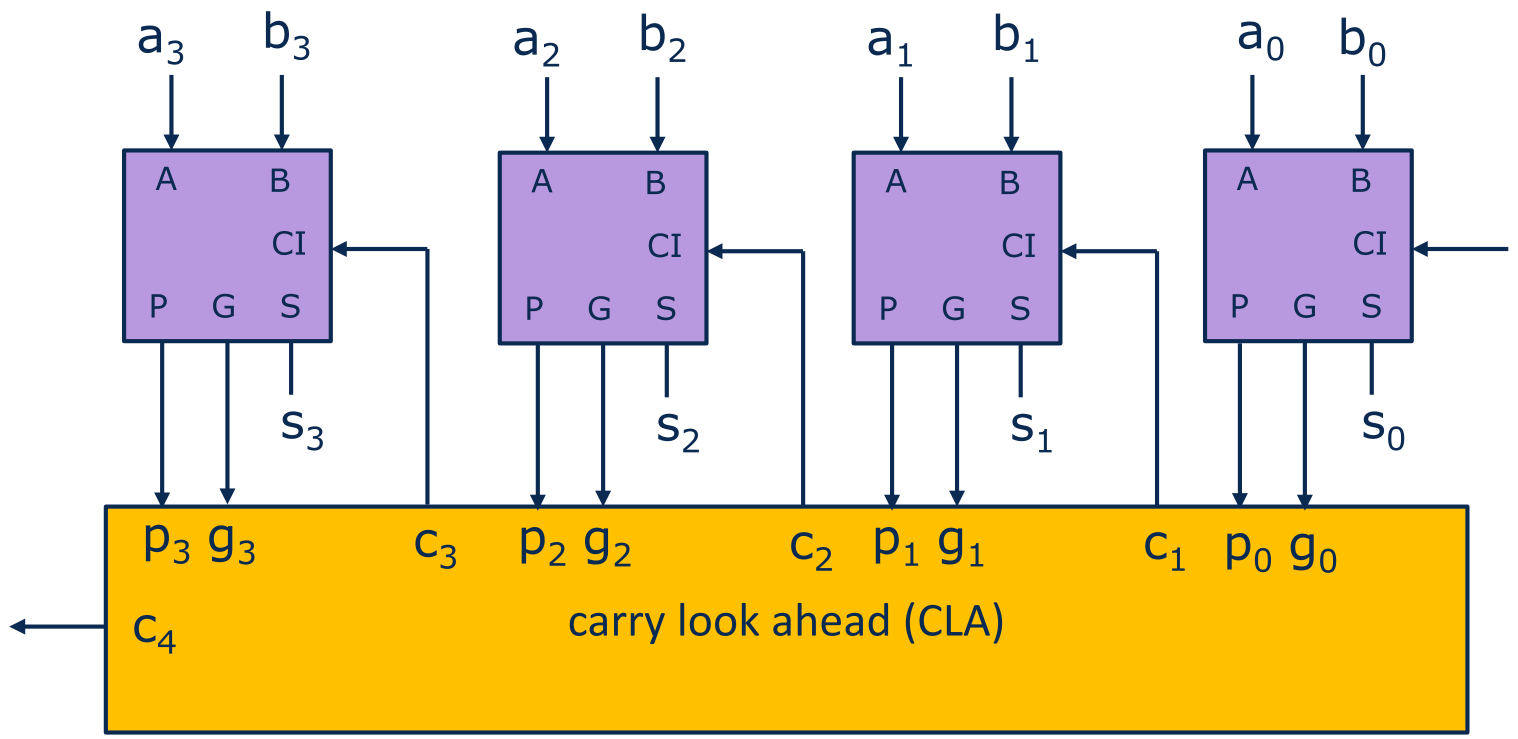
$$S = P \oplus CI$$
$$CO = G + CI \cdot P$$

Carry Look Ahead

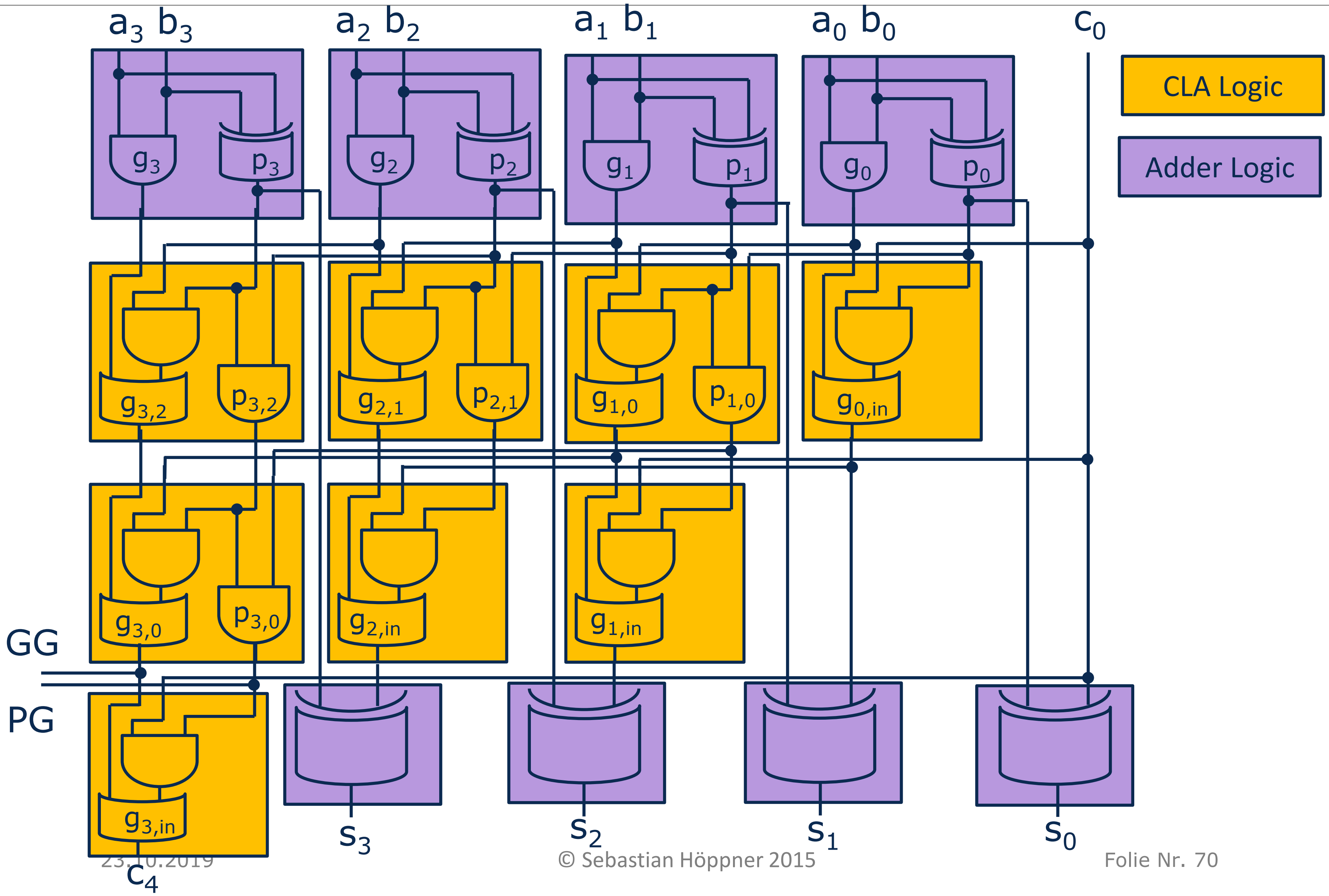
- Prediction of carry bits → Carry Look Ahead
- Example n=4:
  - $C_1=G_0+C_0P_0$
  - $C_2=G_1+C_1P_1=G_1+G_0P_1+C_0P_0P_1$
  - $C_3=G_2+C_2P_2=G_2+G_1P_2+G_0P_1P_2+C_0P_0P_1P_2$
  - $C_4=G_3+C_3P_3=G_3+G_2P_3+G_1P_2P_3+G_0P_1P_2P_3+C_0P_0P_1P_2P_3$

Requires only 5 logic stages!

Carry Look Ahead Adder

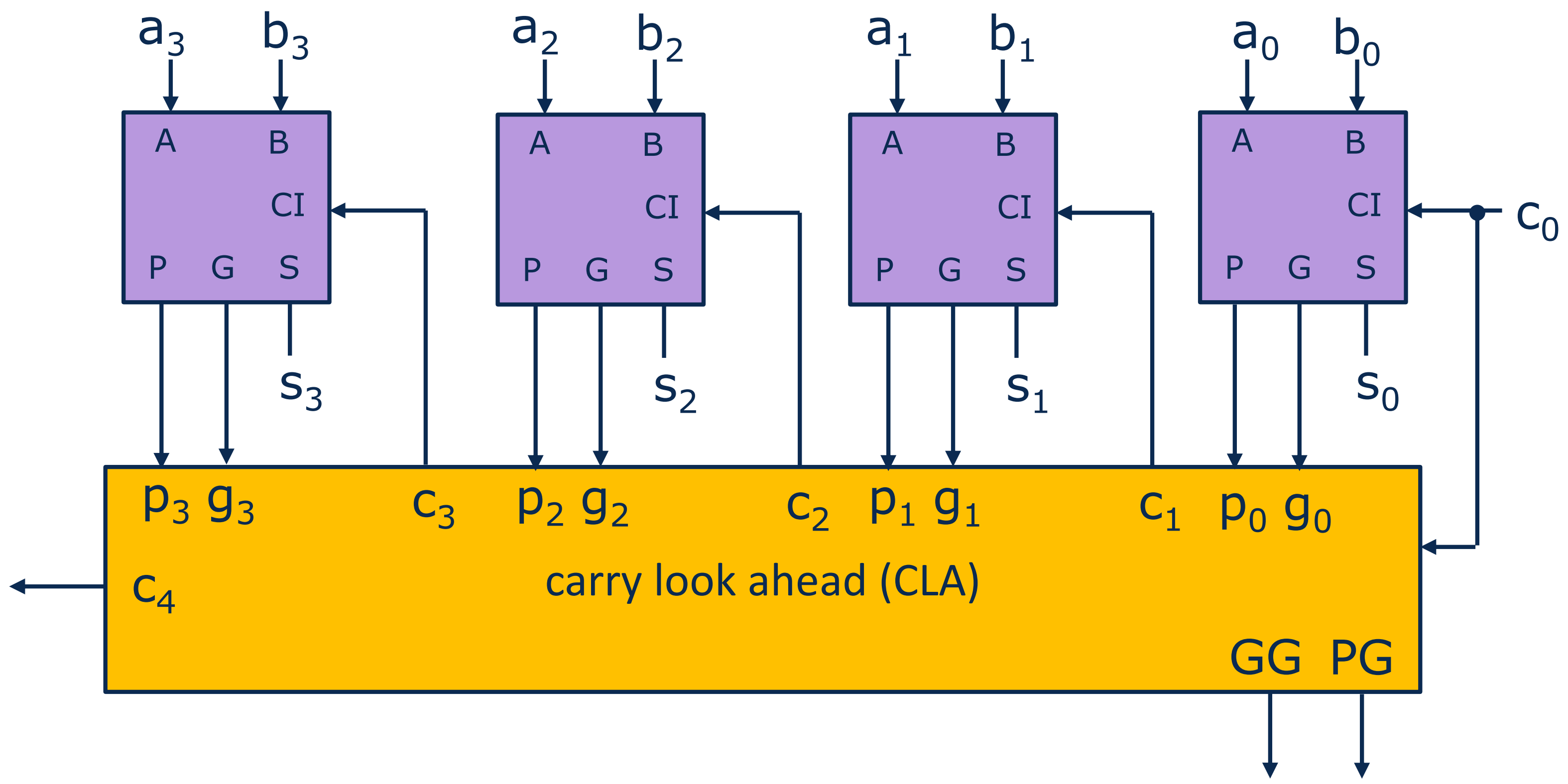

a_3 b_3 a_2 b_2 a_1 b_1 a_0 b_0
A B CI P G S
s_3 s_2 s_1 s_0
p_3 g_3 c_3 p_2 g_2 c_2 p_1 g_1 c_1 p_0 g_0
c_4
carry look ahead (CLA)

Carry Look Ahead Logic: „Kogge Stone Adder“



Group Propagate and Group Generate

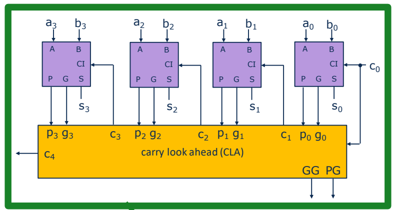
- Pre-calculation of Generate and Propagate for an n-bit group



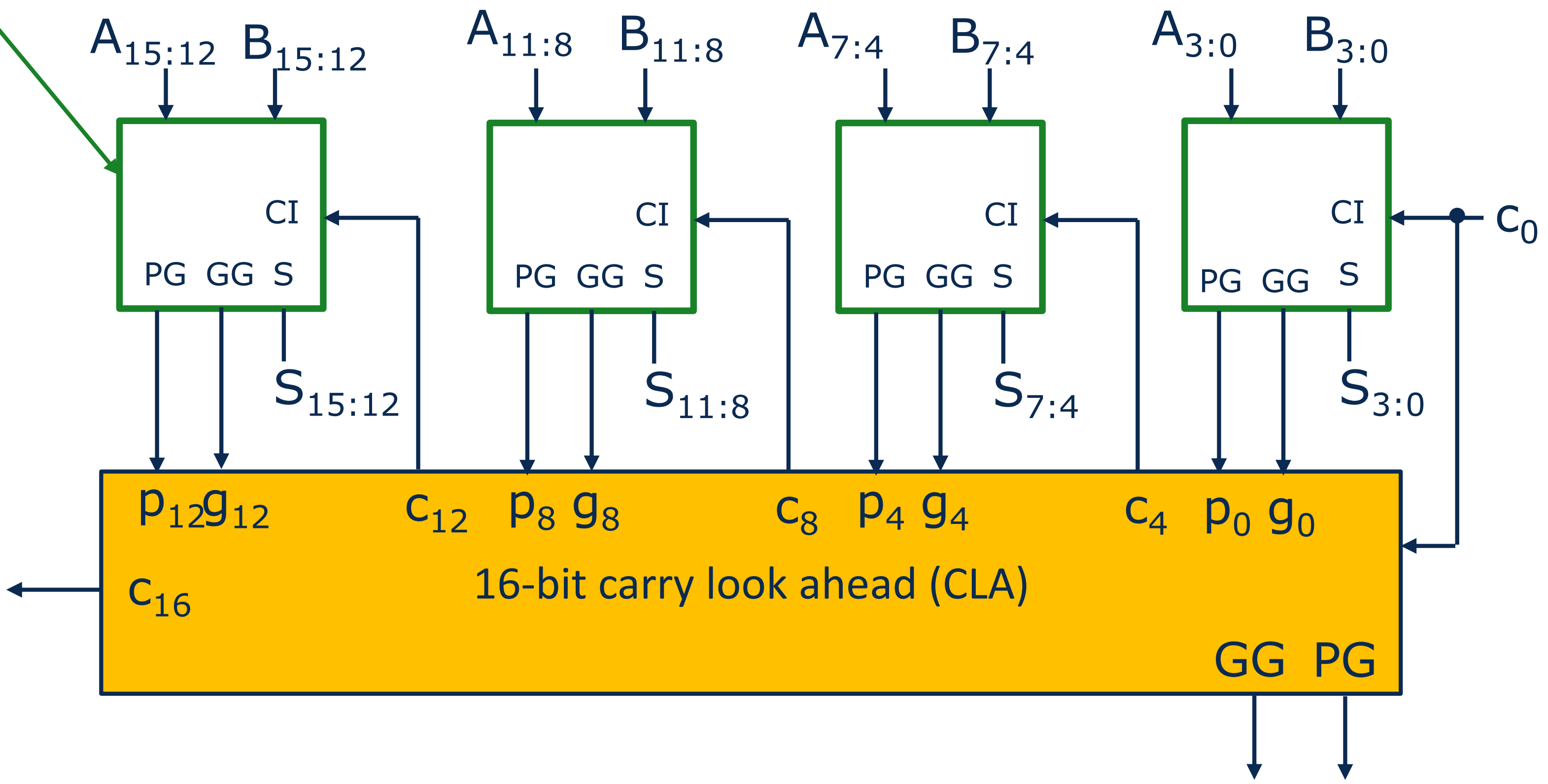
a_3 b_3 a_2 b_2 a_1 b_1 a_0 b_0
A B CI P G S
s_3 s_2 s_1 s_0
c_0
p_3 g_3 c_3 p_2 g_2 c_2 p_1 g_1 c_1 p_0 g_0
c_4
carry look ahead (CLA)
GG PG

- Group Propagate: $PG=P_3P_2P_1P_0$ → 2+1 gate stages
- Group Generate: $GG=G_3+G_2P_3+G_1P_3P_2+G_0P_3P_2P_1$ → 4+1 gate stages

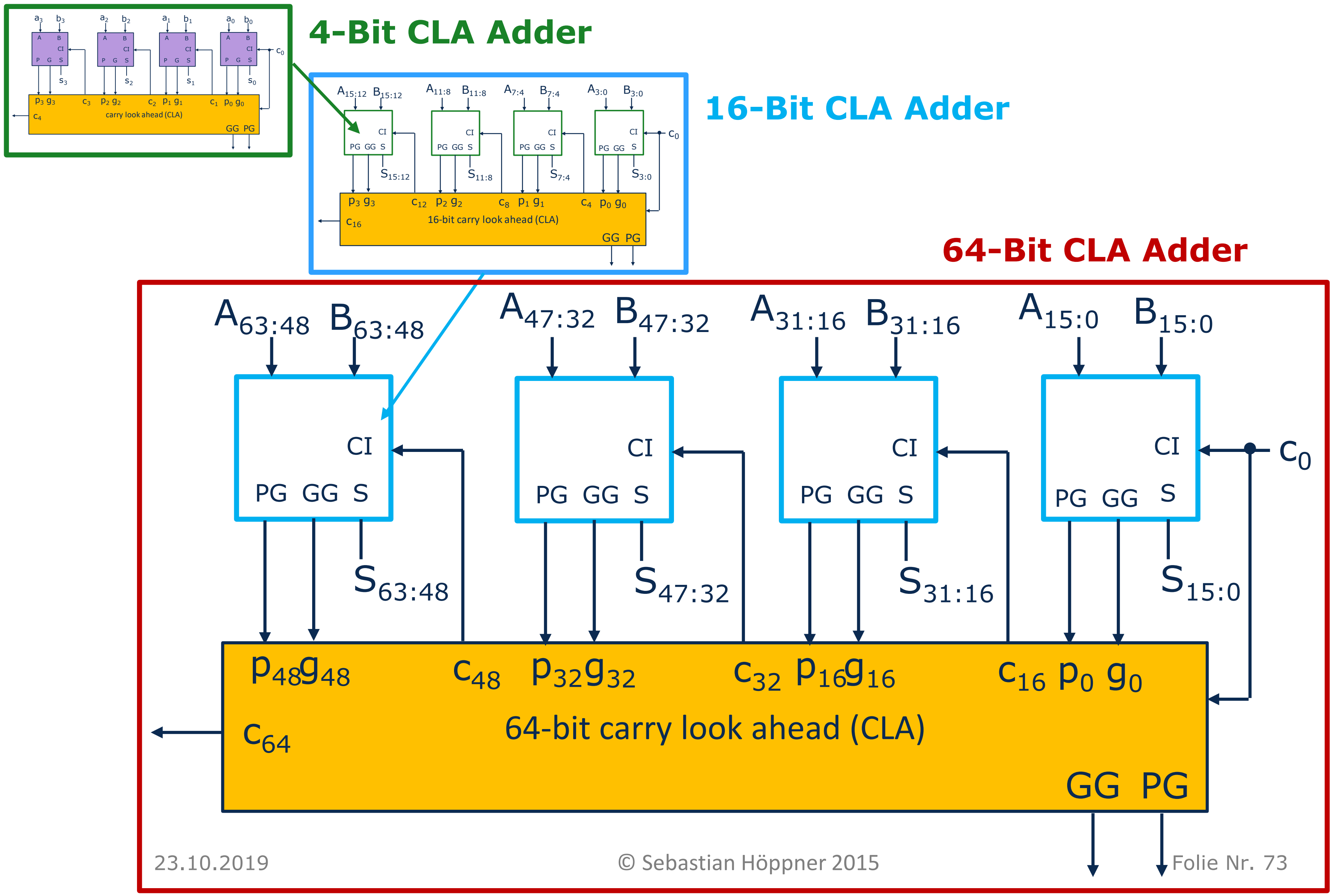
Partitioning of Fast Adders



4-Bit CLA Adder

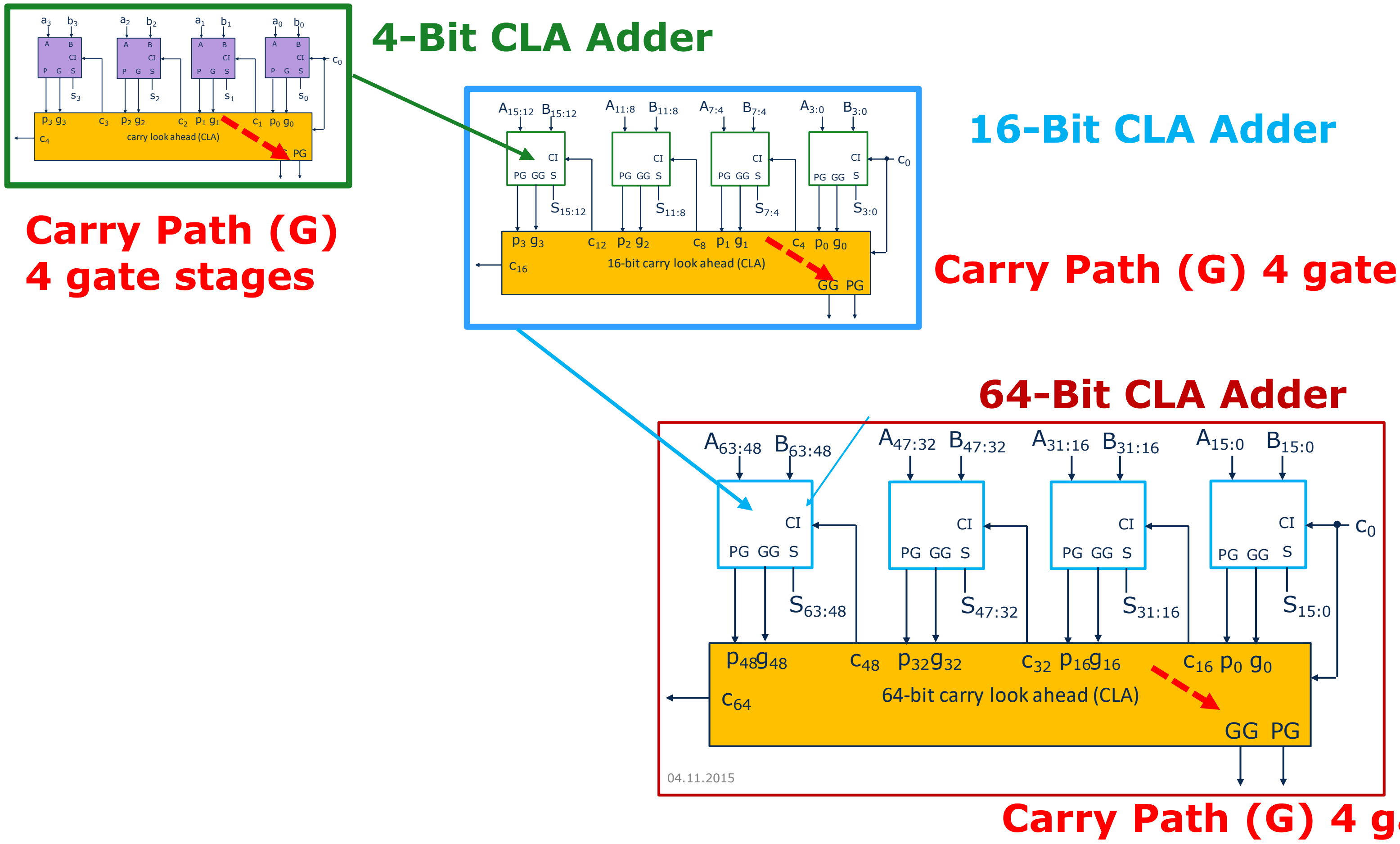


Partitioning of Fast Adders


4-Bit CLA Adder
a3 b3 a2 b2 a1 b1 a0 b0
A B CI P G S
s3 s2 s1 s0
c0
p3 g3 c3 p2 g2 c2 p1 g1 c1 p0 g0
c4
carry look ahead (CLA)
GG PG
16-Bit CLA Adder
A15:12 B15:12 A11:8 B11:8 A7:4 B7:4 A3:0 B3:0
CI PG GG S
S15:12 S11:8 S7:4 S3:0
c0
p3 g3 c12 p2 g2 c8 p1 g1 c4 p0 g0
c16
16-bit carry look ahead (CLA)
GG PG
64-Bit CLA Adder
A63:48 B63:48 A47:32 B47:32 A31:16 B31:16 A15:0 B15:0
CI
PG GG S
S63:48 S47:32 S31:16 S15:0
C0
p48 g48 c48 p32 g32 c32 p16 g16 c16 p0 g0
c64
64-bit carry look ahead (CLA)
GG PG

TECHNISCHE UNIVERSITÄT DRESDEN

Partitioning of Fast Adders



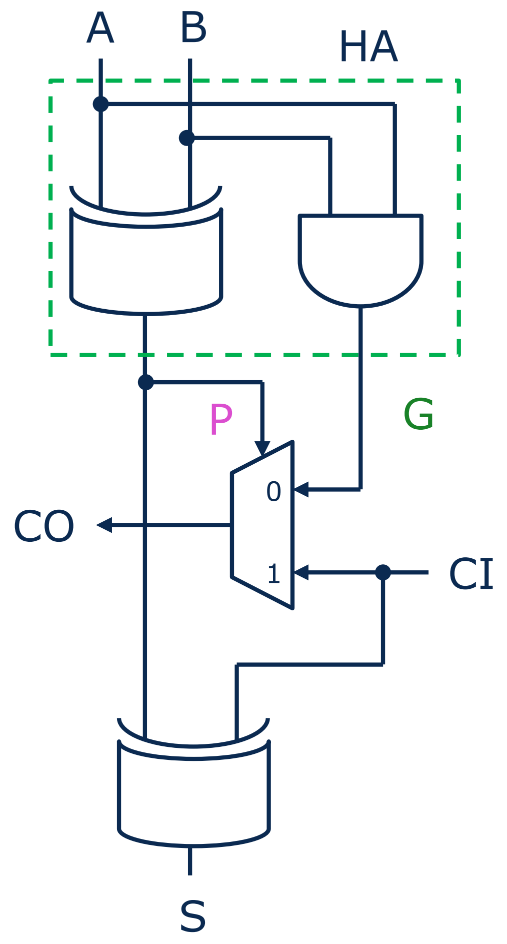
- → 1+3·4+2=15 gate stages for C64
- 64-Bit Ripple Carry Adder would have ≈128 gate stages

„Manchester Chain“ Full Adder

- Fast carry forwarding through multiplexer
- Homogeneous hardware implementation of adder chains

$$S = P \oplus CI$$

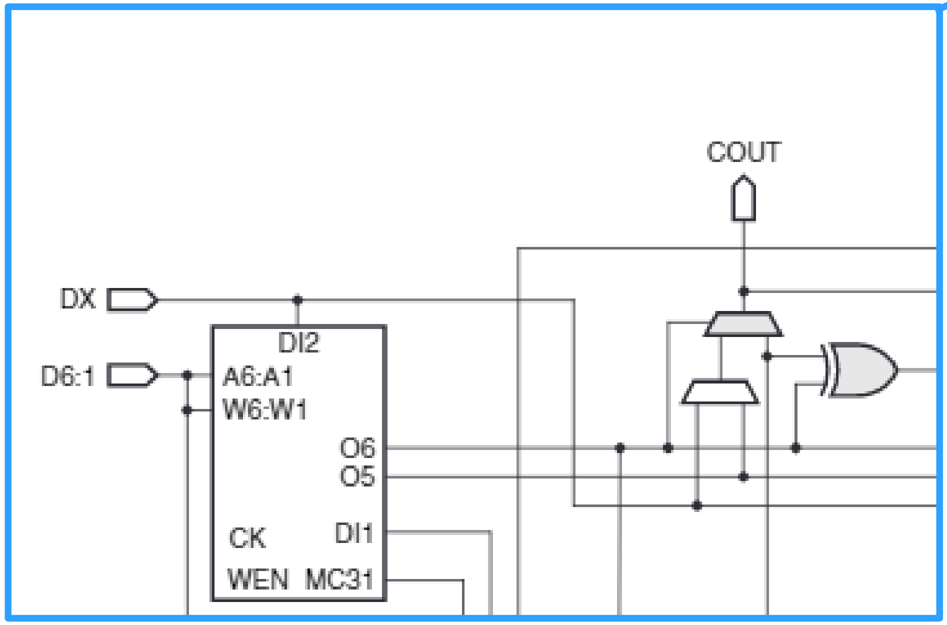
$$CO = G + CI \cdot P$$



Carry Path CI → CO: 1 Multiplexer

Example: Carry Chain in FPGA Logic Blocks

- Fast carry paths in Field-Programmable Gate Arrays (FPGAs)
- Implementing fast adders and subtractors with configurable word length



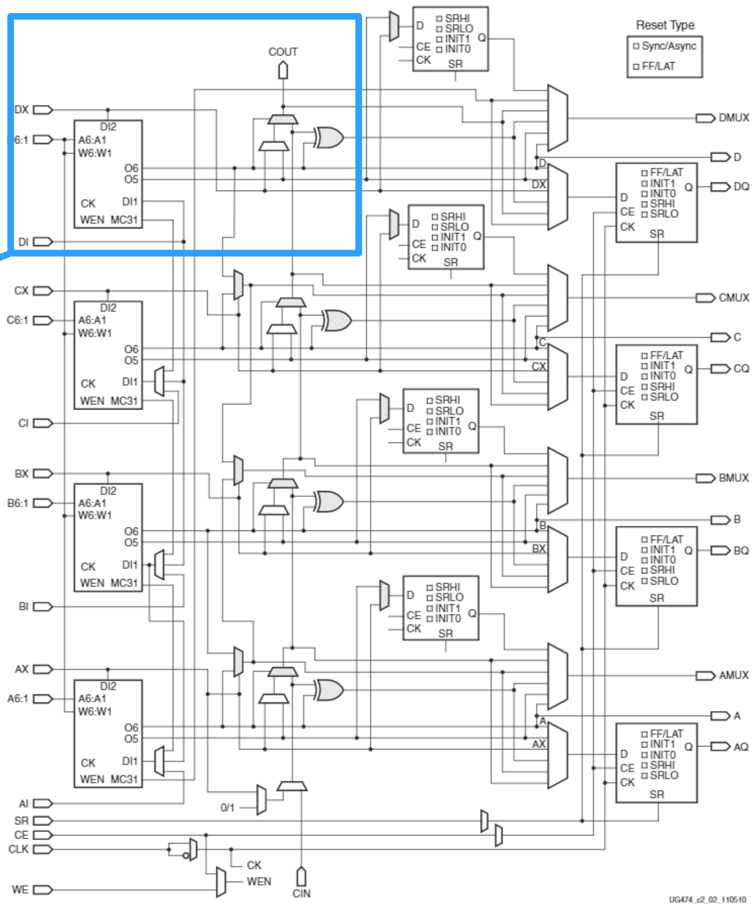


Figure 2-3: **Diagram of SLICEM**

- Source: Xilinx 7 Series FPGAs Configurable Logic Block, User GuideUG474 (v1.7) November 17, 2014

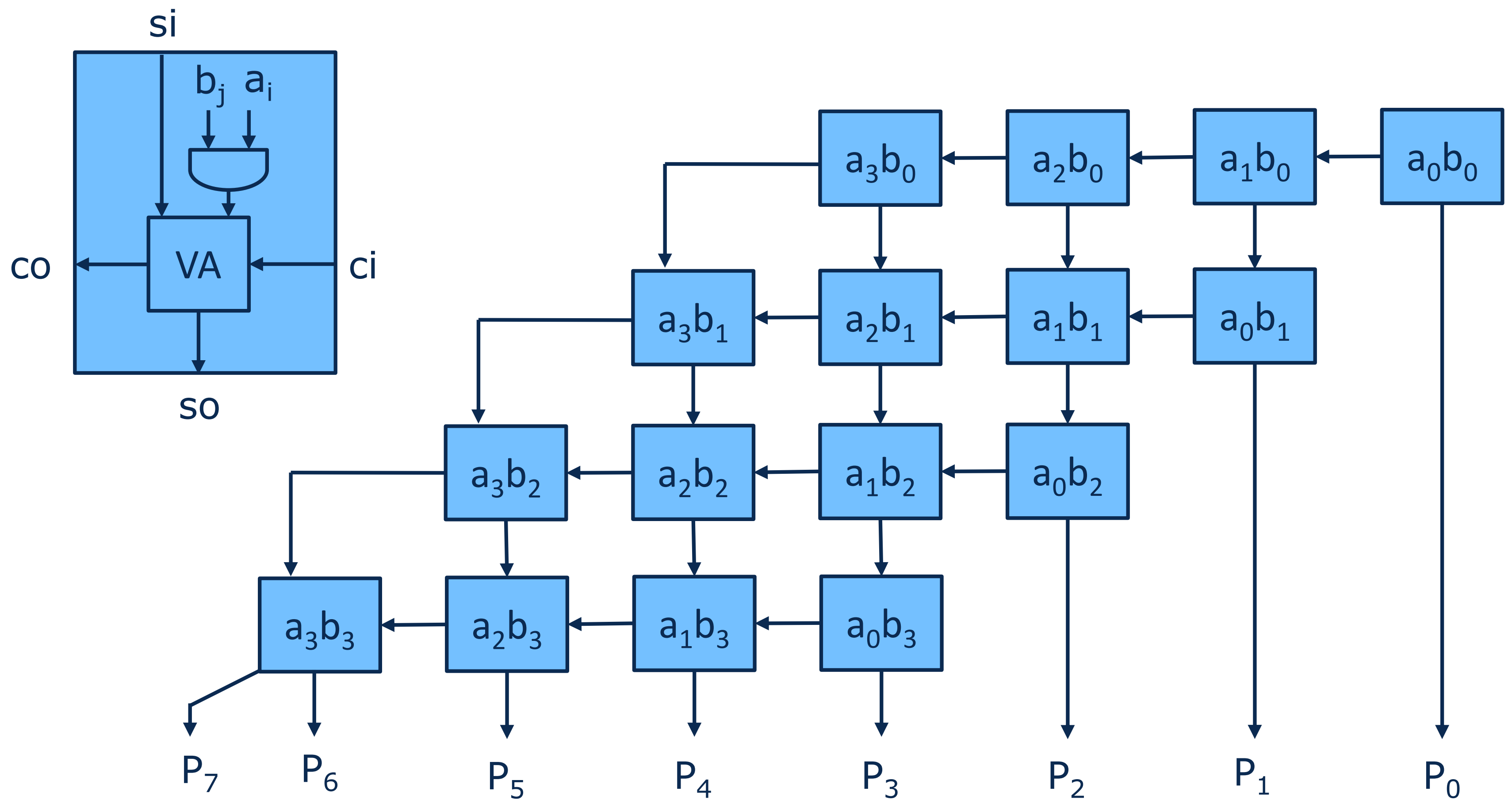
Multiplier Topologies

- Weighted addition of partial products $a_i \cdot b_j$

					A3	**A2**	**A1**	**A0**	
					A3B0	A2B0	A1B0	A0B0	B0
+				A3B1	A2B1	A1B1	A0B1		B1
+			A3B2	A2B2	A1B2	A0B2			B2
+		A3B3	A2B3	A1B3	A0B3				B3
	P7	P6	P5	P4	P3	P2	P1	P0	

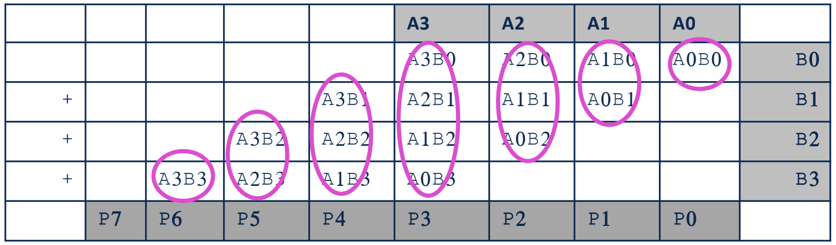
Multiplier Topologies

- Addition of partial products $a_i \cdot b_j$ by ripple carry chains

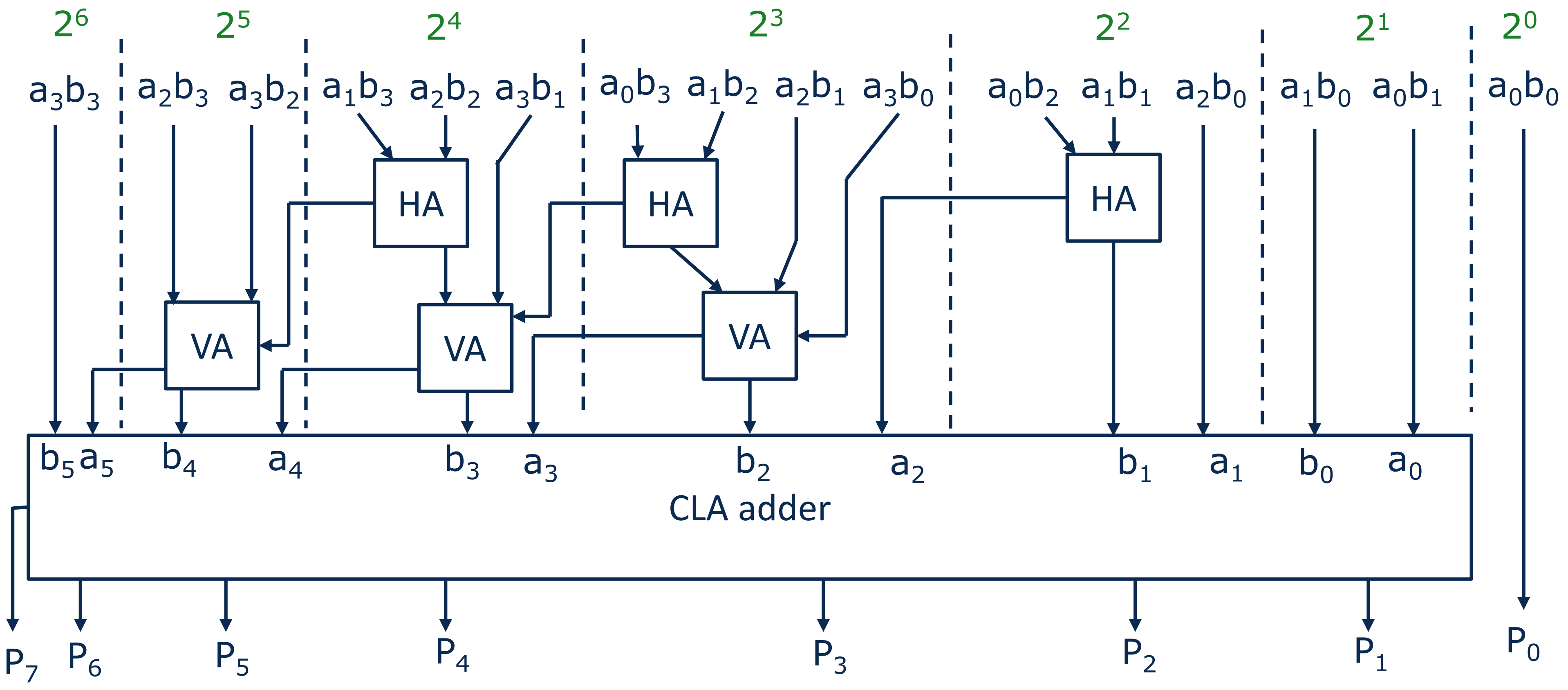


Wallace Tree Multiplier

- Addition of the **weighted** partial products in a tree structure Reduction of the logic levels especially for large adders (complexity O (log n))



					A3	A2	A1	A0	
					A3B0	A2B0	A1B0	A0B0	B0
+				A3B1	A2B1	A1B1	A0B1		B1
+			A3B2	A2B2	A1B2	A0B2			B2
+		A3B3	A2B3	A1B3	A0B3				B3
	P7	P6	P5	P4	P3	P2	P1	P0	



2^6
2^5
2^4
2^3
2^2
2^1
2^0
a_3b_3
a_2b_3 a_3b_2
a_1b_3 a_2b_2 a_3b_1
a_0b_3 a_1b_2 a_2b_1 a_3b_0
a_0b_2 a_1b_1 a_2b_0
a_1b_0 a_0b_1
a_0b_0
HA
HA
HA
VA
VA
VA
b_5 a_5 b_4 a_4 b_3 a_3 b_2 a_2 b_1 a_1 b_0 a_0
CLA adder
P_7 P_6 P_5 P_4 P_3 P_2 P_1 P_0

TECHNISCHE UNIVERSITÄT DRESDEN

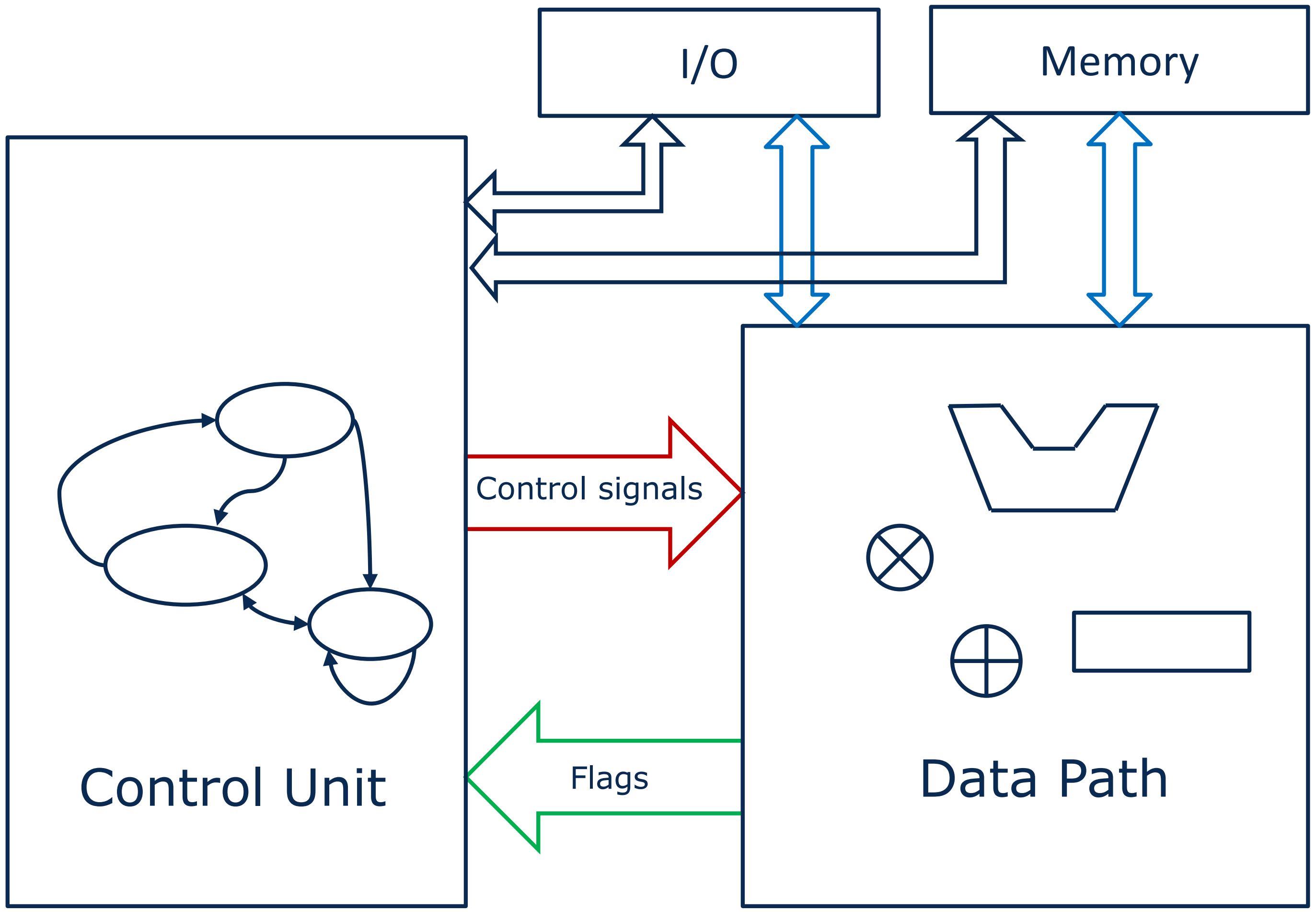
Summary of Arithmetic Circuits

- Presentation of circuits for binary addition and multiplication
- Strategies for optimizing hardware implementation regarding gate delays
- In the practical implementation further boundary conditions are to be considered, such as e.g. chip area, power consumption

Data Paths

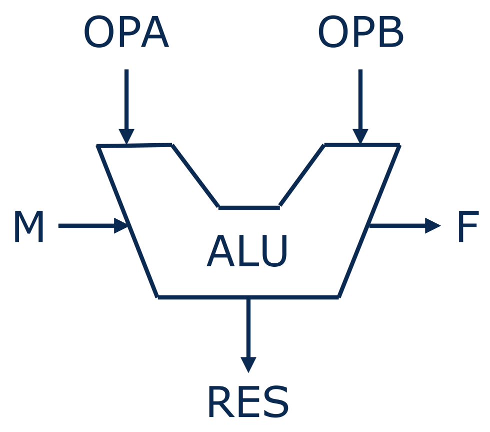
TECHNISCHE UNIVERSITÄT DRESDEN

Partitioning of Digital Systems



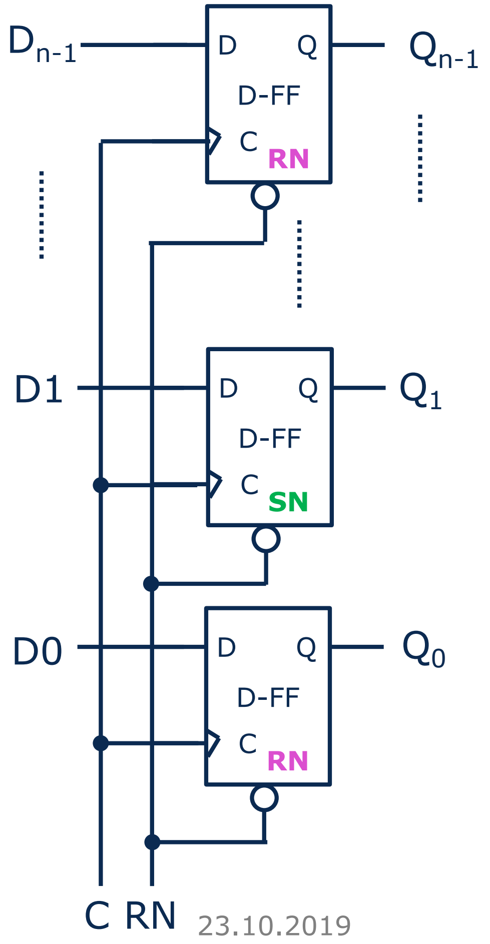
Arithmetic Building Blocks

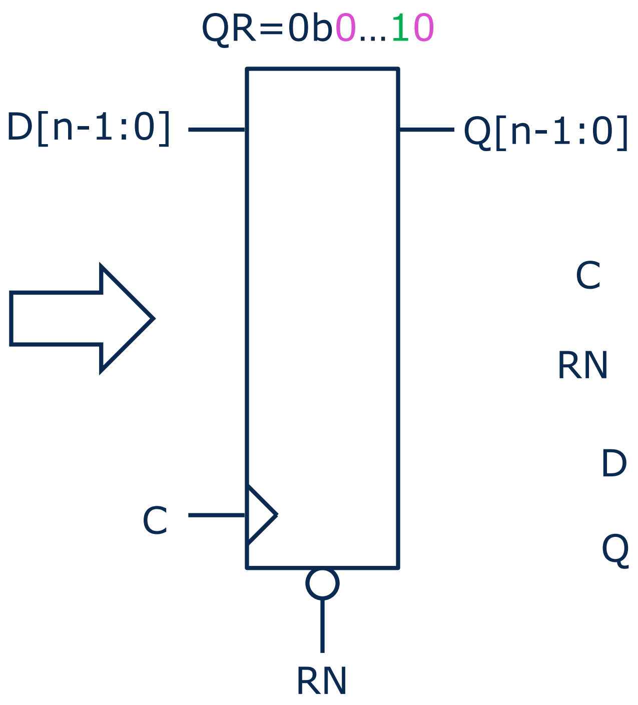
- Arithmetic Logic Unit (ALU) process numerical and logic data
  - Operands (OPA, OPB, …)
  - Mode (M): Choice of operation, e.g.
    - ADD, SUB, MUL, DIV, SHIFT, AND, OR, …)
  - Result (RES)
  - Status Flags (F): Additional information about the operation, e.g.
    - CARRY, OVERFLOW, SIGN, ZERO

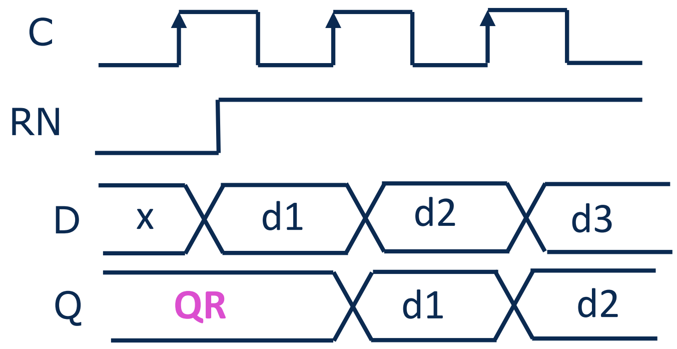


- ALU data path building blocks can be combinational or sequential
- In SSE Lab, dedicated building blocks are available for the arithmetic operations

- Ordering n-bit flip-flops for data storage
- **Reset-value** of register, set at implementation time by selecting the flip-flop type (**Set**, **Reset**)





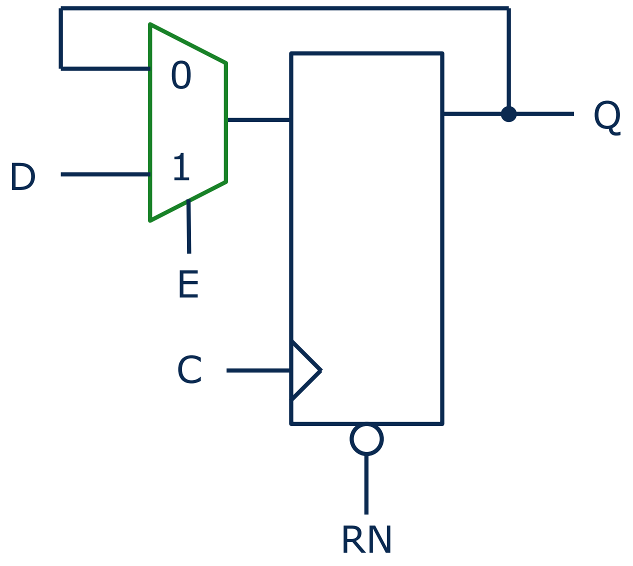


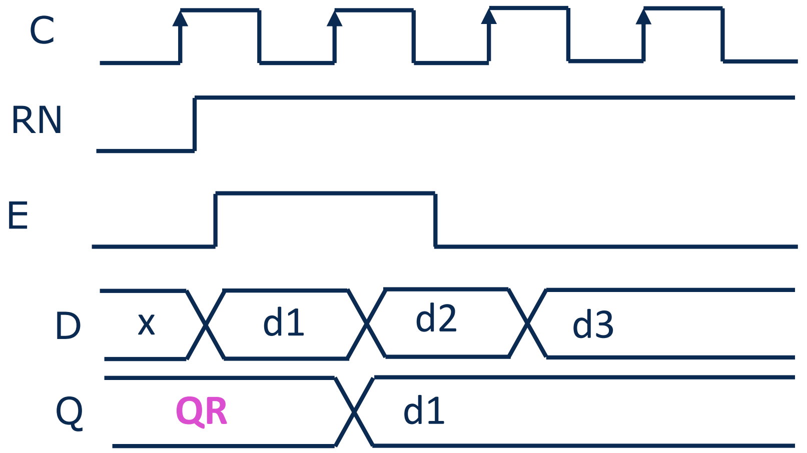
Very fast access (1 clock), very short delay CLK ↑ →Q

Register with Enable 1

- Conditional writing to flip-flop when E = 1
- $Q_{i+1}=D_i$, when E=1
  - $Q_{i+1}=Q_i$, otherwise

- Implementation using Multiplexer
  - Permenant data writing
  - Selection of data



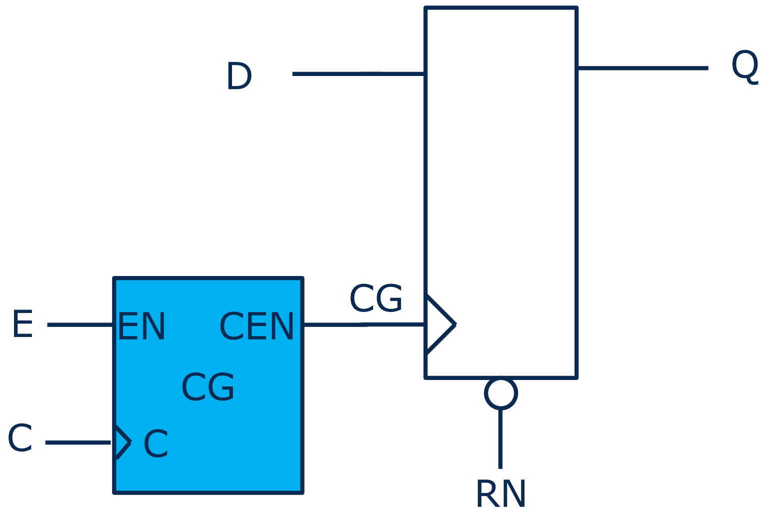


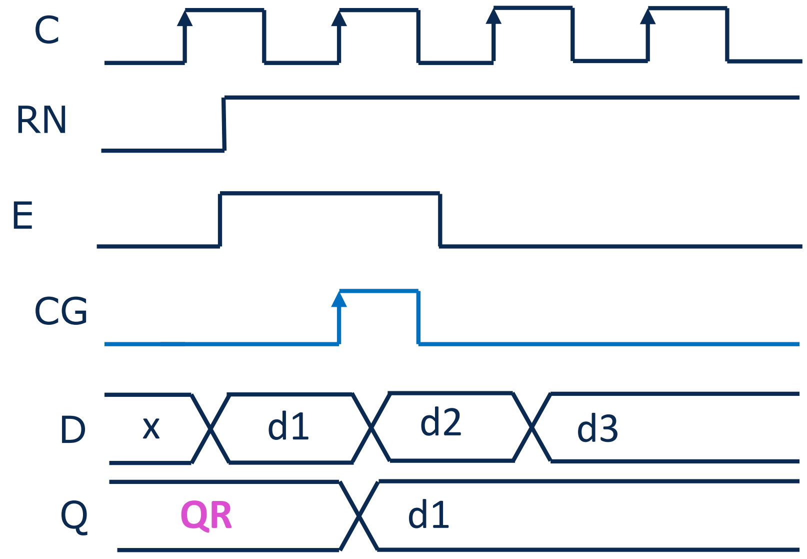
- Low circuit complexity
- Hardly any increase in power consumption with high write activity

TECHNISCHE UNIVERSITÄT DRESDEN

Register with Enable 2

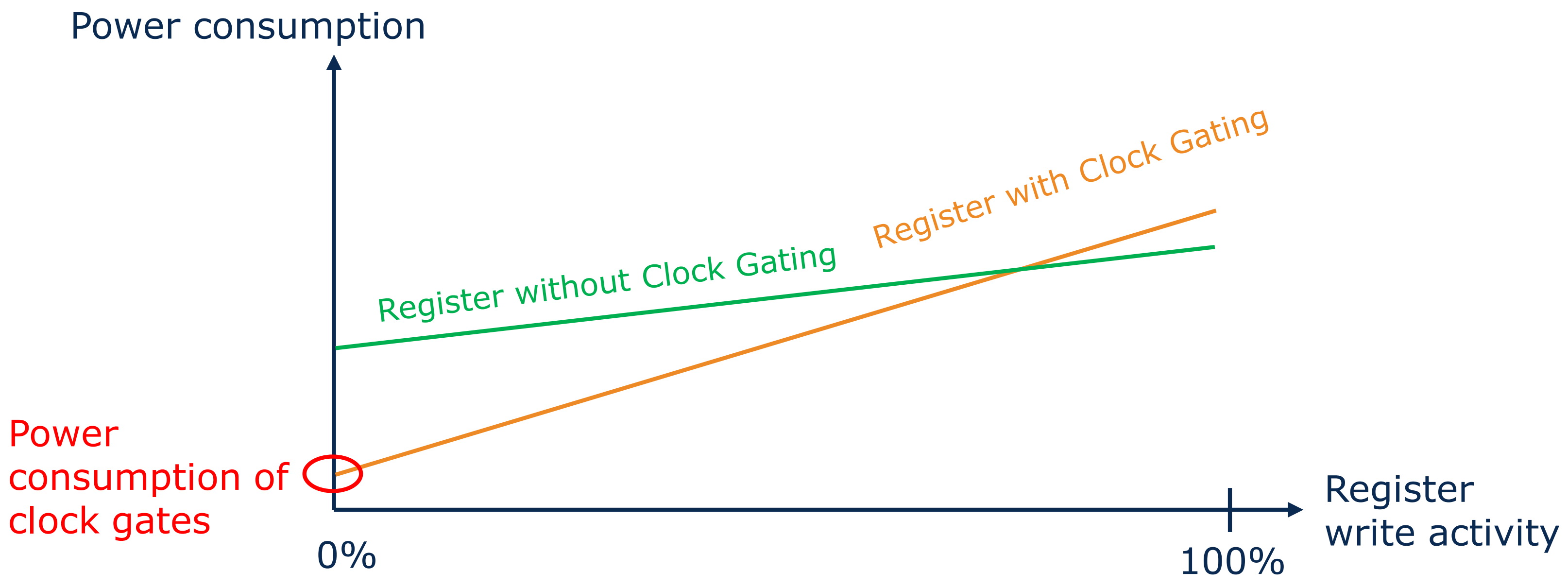
- Implementation using Clock Gate
  - Selective clocking for the register





- Higher circuit complexity (area)
- Higher power consumption with high write activity
- Lower power consumption with low write activity

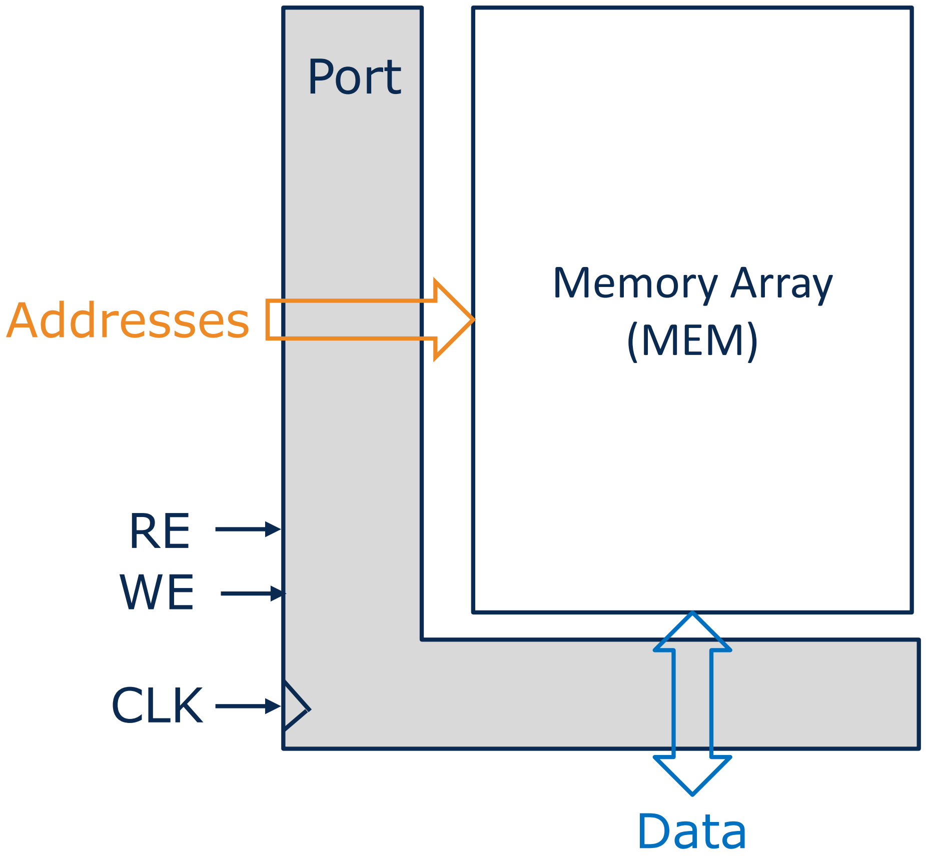
Power Loss Reduction through Clock-Gating



- → Clock Gating application depends on the operation scenario

Memory

- Addressable memory
- Memory Array
  - Data word length: n_D
  - Addresses: n_A
  - Capacity $n_A \cdot n_D$
- Access via ports
  - Write: MEM(addr)=D_{IN}
  - Read: Q=MEM(addr)
- Criteria:
  - Storage density [Bit/µm²]
  - Access time (number of clock cycles, delay CLK→Q)
  - Power consumption
  - Non-volatile memory (yes/no)



TECHNISCHE UNIVERSITÄT DRESDEN

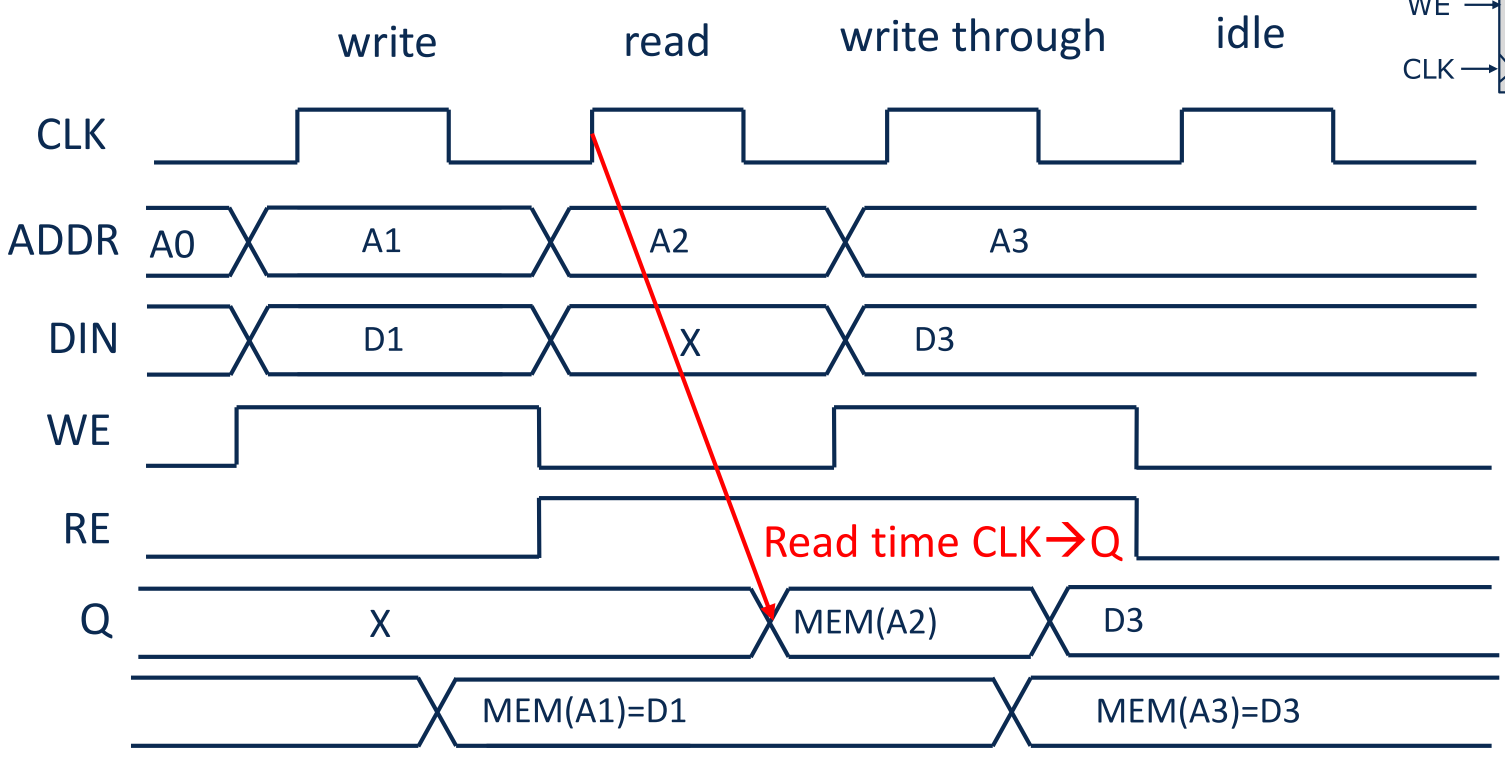
Types of on-Chip Memory

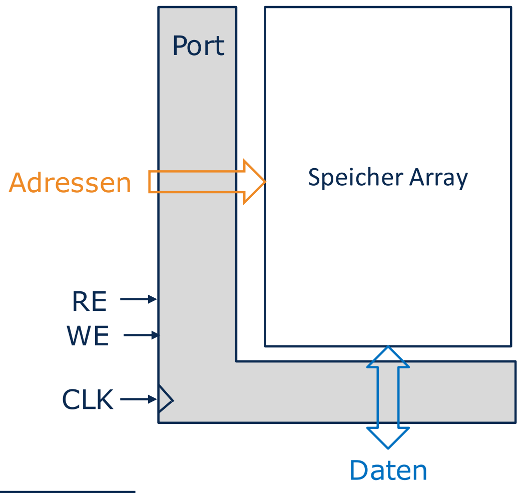
Type	NVM	Bits/µm²	Access time	Typical memory sizes	Comments
Latch	No	<1	some 10ps	some Bits	Control circuit needed bec. level sensitive!
FlipFlop	No	<0.4	some 10ps	some Bits	
SRAM	No	<7 (Cell) <5 (Macro)	some 100ps	some kBit to MBit	Complex control circuit (SRAM Macro)
(embedded) DRAM	No	<30	some 100ps	some kByte to Mbyte	Dynamic memory, refresh needed. Possibly extra process options (costs!)
(embedded) Flash	Yes	<30	some ns	some kByte to Mbyte	Extra process options (costs!)
ROM	Yes	<15	some 100ps	some kBit to MBit	„Mask programming“ during implementation
OTP	Yes	<10	some 100ps	some kBit to MBit	„one-time programming“ when testing the chip

Values estimated for 28nm technology node

Memory Access

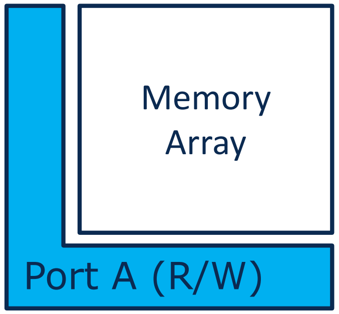
- Example:
  - Synchronous SRAM Port
  - 1 clock access time
  - Write-Through Function

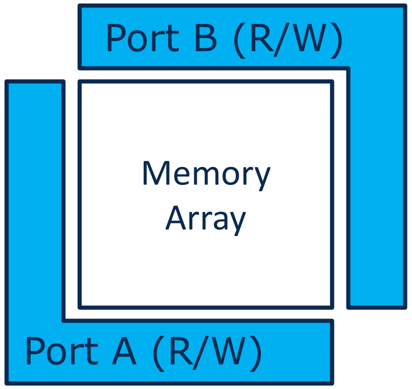


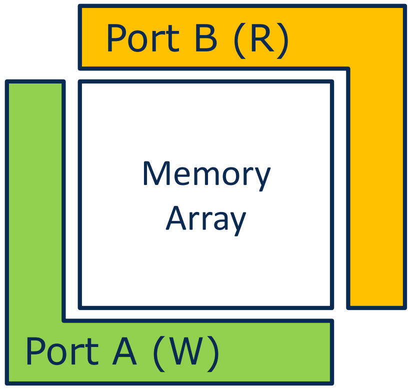


Classification of SRAM Macros

- Single Port (SP)
  - Access from **one** **Write/Read** Port
- Dual Port (DP)
  - Access from **2 independent** **Write/Read** Ports
  - Synchronous and asynchronous variants possible
  - Arbitration / prioritization of Write accesses to the same addresses necessary
- Two-Port (TP)
  - Access from **one** **Write Port** and **one seperated** **Read Port**

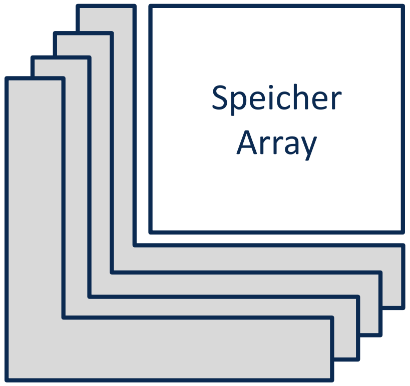


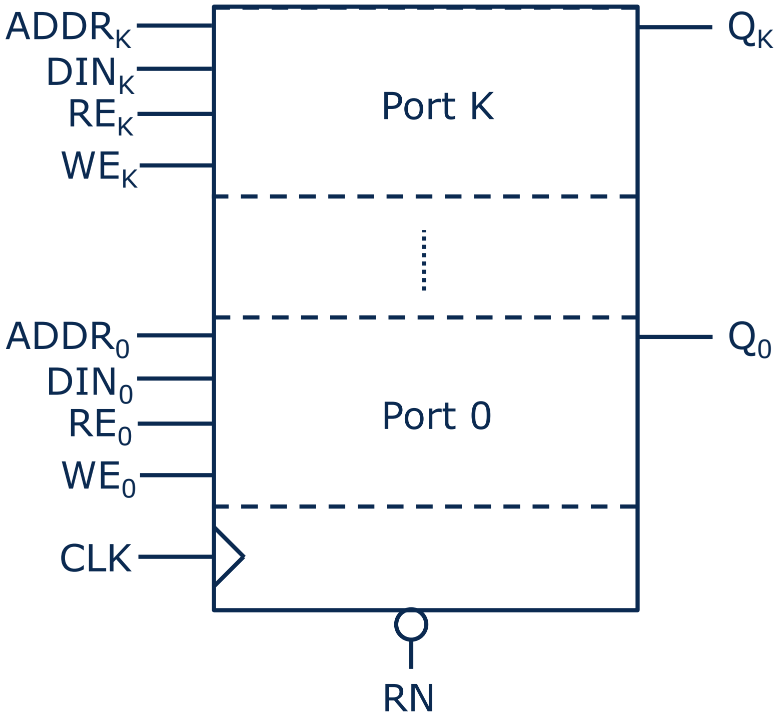




Register Files

- Arrays of registers with multiple write and read ports
- Addressable access
- Typically synchronous (same clock for all ports)
- Faster access (1 Clk), short delay CLK ↑ →Q



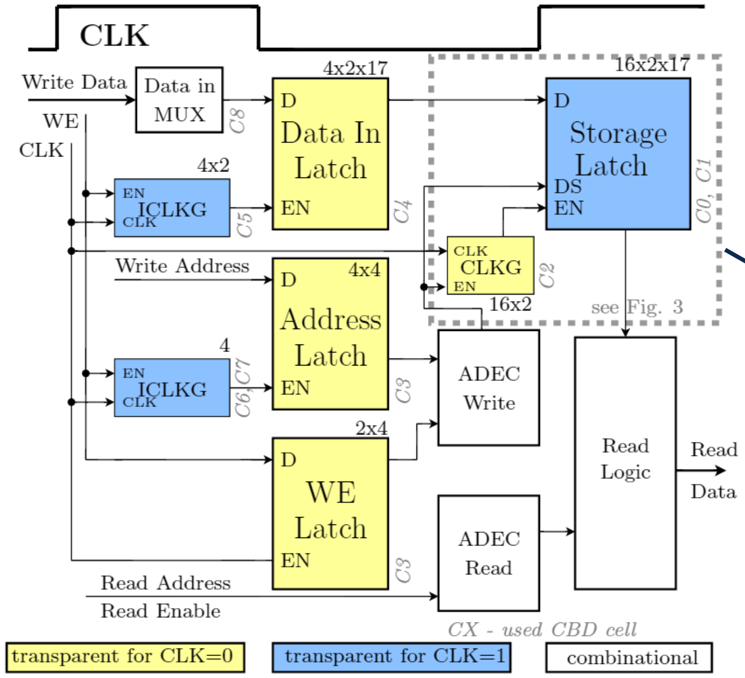


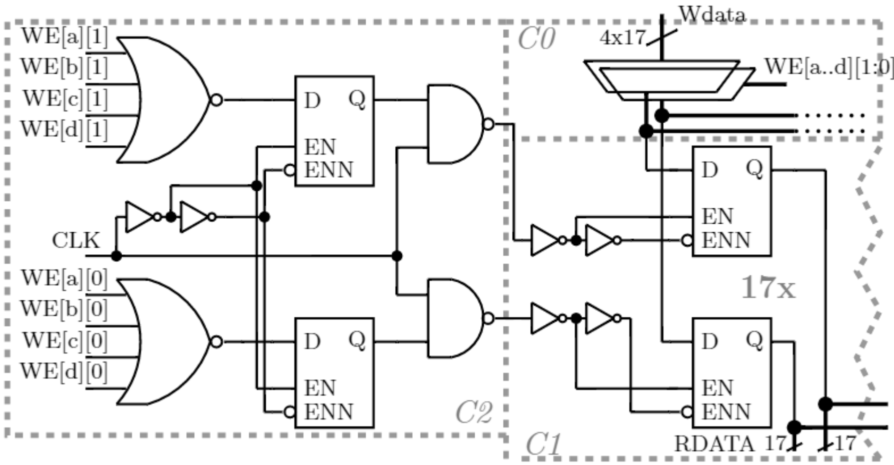
- Application in systems with multiple cores

Example: Register File for SDR Processor

- Register File for Infineon X-GOLD SDR^{TM} 20 Processor
- Developed by the HPSN chair (2008/2009)
- 16-words of 34 bits each, 4-Write Ports, 6 Read Ports
- Cell-Based Design → Optimization at Transistor level

Source: infineon.com

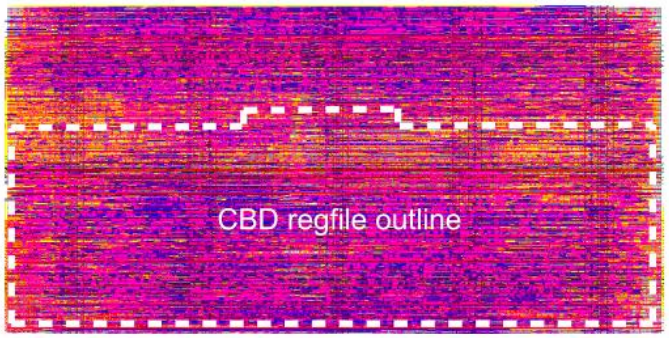


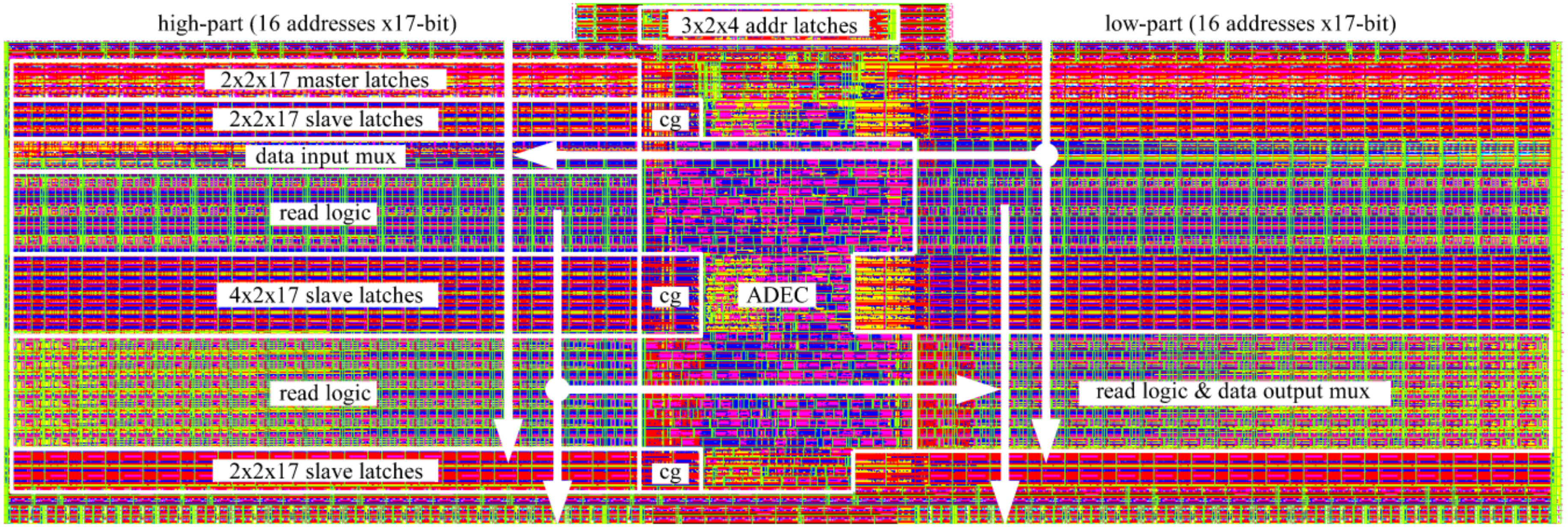


Johannes Uhlig, Sebastian Höppner, Georg Ellguth, and René Schüffny, A Low-Power Cell-Based-Design Multi-Port Register File in 65nm CMOS Technology, IEEE International Symposium on Circuits and Systems ISCAS 2010, 2010, p. 313-316,

Example: Register File for SDR Processor

- Implementation in 65nm CMOS Technology
- ≈0,026mm^2 chip area (0,02Bit/µm^2)
- 300MHz clock frequency
- Increased efficiency compared to synthesis and place & route implementation:
  - 44% of saved area and
  - 30% of power consumption reduced





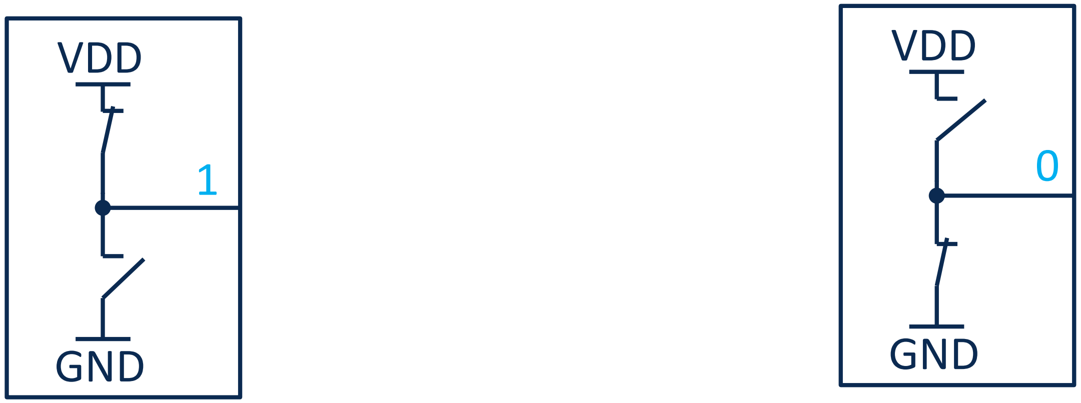
Johannes Uhlig, Sebastian Höppner, Georg Ellguth, and René Schüffny, A Low-Power Cell-Based-Design Multi-Port Register File in 65nm CMOS Technology, IEEE International Symposium on Circuits and Systems ISCAS 2010, 2010, p. 313-316,

Bus Systems

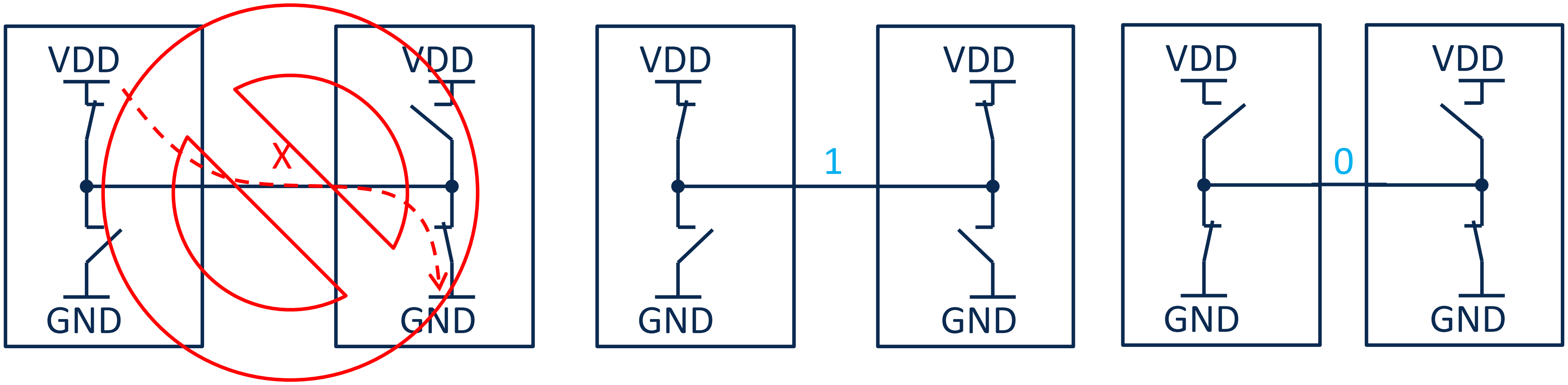
- The interconnects of data path blocks should be flexible and reconfigurable
- Bus systems serve the networking (connection) of data path elements
- 3 approaches will be presented:
  - Tri-state Bus
  - Multiplexer Bus
  - Complex bus systems and Network-on-Chip

Tri-State Driver

- Gate **outputs** generate 2 logic levels (0,1)



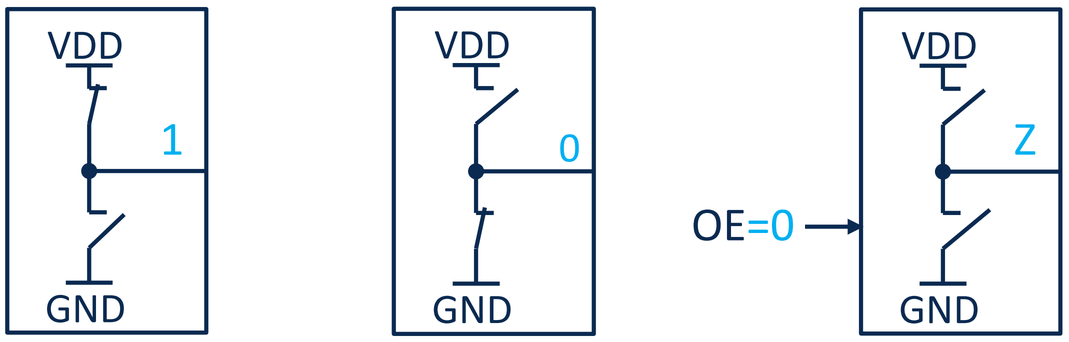
- Connecting several gate **outputs** is only possible if they drive the **same** level



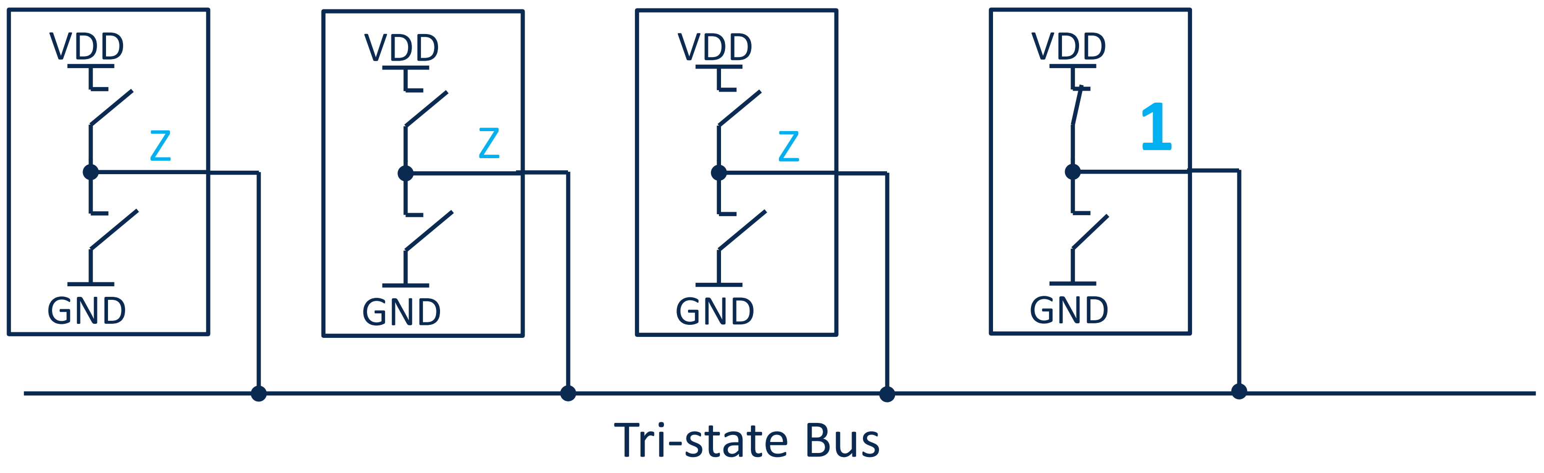
TECHNISCHE UNIVERSITÄT DRESDEN

Tri-State Driver

- Introducing a **third** initial state (Tri-State) **Z**
- Output driver is switched to high impedance, by separate control signal (OE)

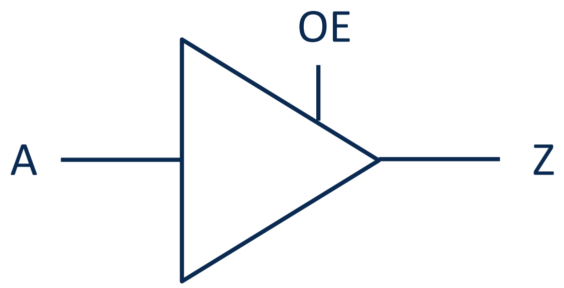


- Connecting several gate **outputs** in tri-state is possible.
- Only one driver may be active!

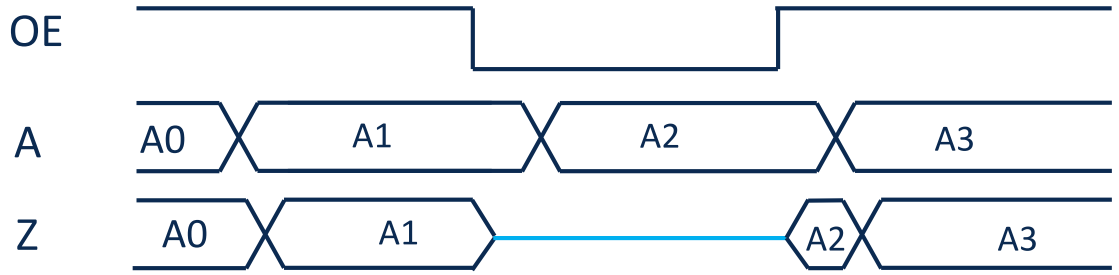


Tri-State Buffer

- Buffer with Tri-state output driver

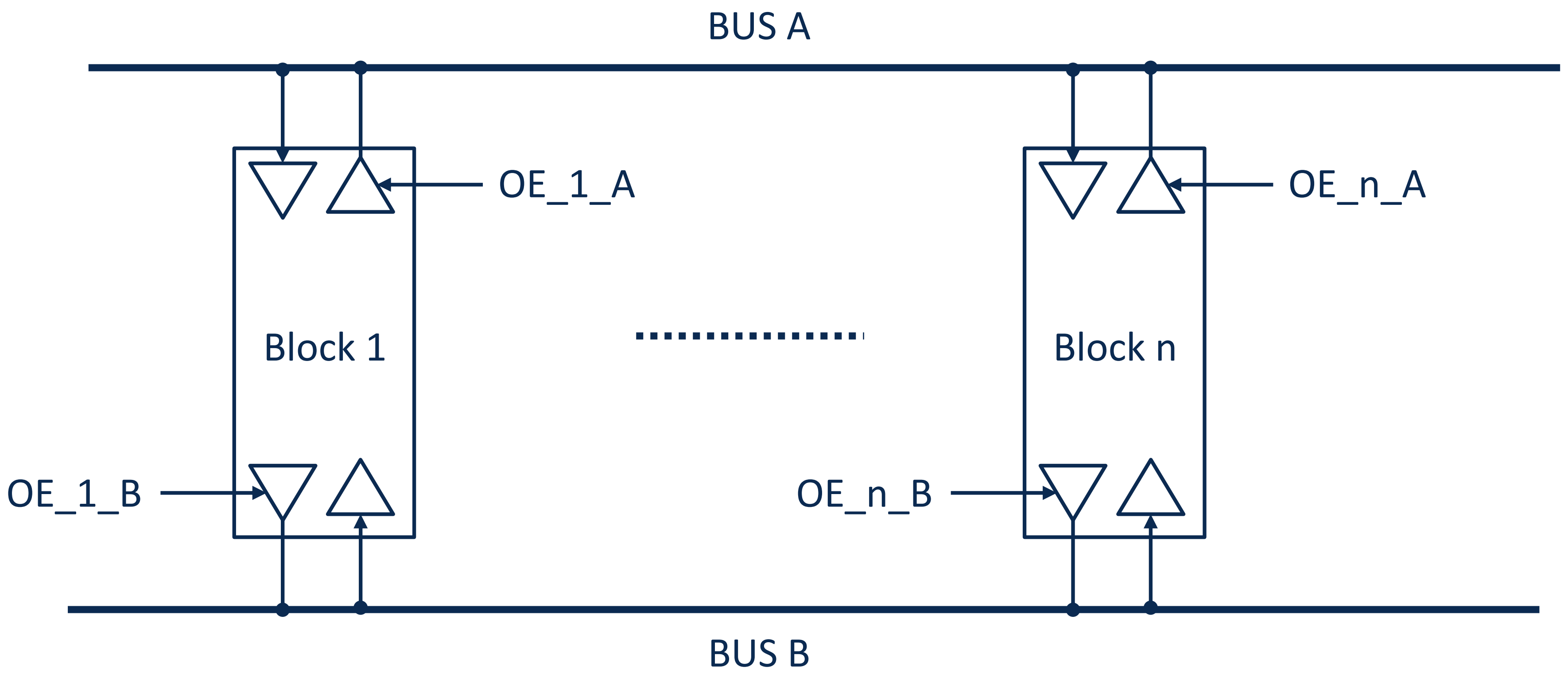


OE	A	Z
0	X	Z
1	0	0
1	1	1



- Combinational gates and sequential building blocks (e.g., registers) may be supplied with tri-state outputs.

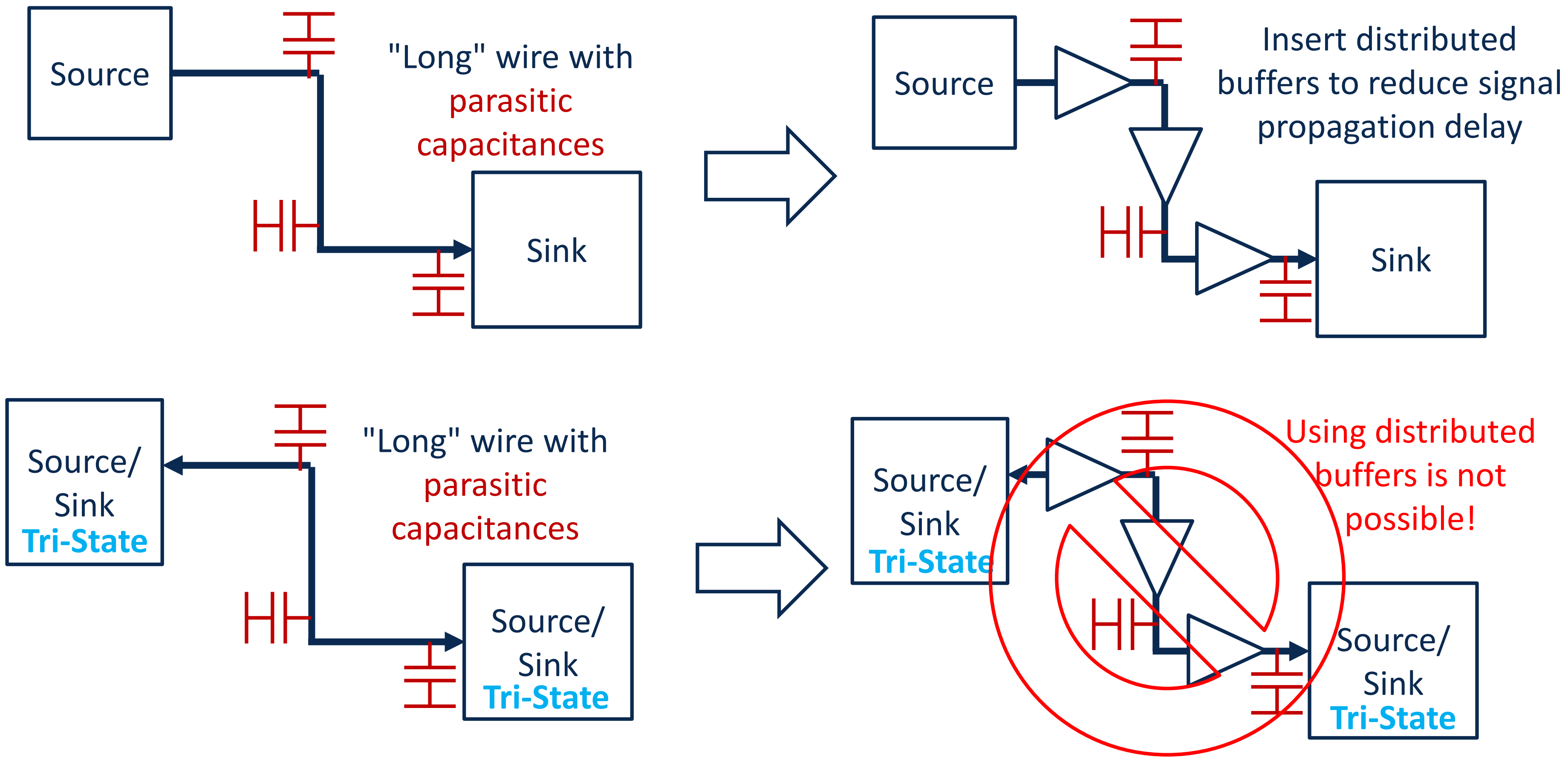
Data Paths with Tri-State Buses



- Several building blocks can be connected to **one** tri-state bus
- Only **one** driver may be active per tri-state bus → Control unit!
- Several buses possible for **parallel** data transfer between building blocks

TECHNISCHE UNIVERSITÄT DRESDEN

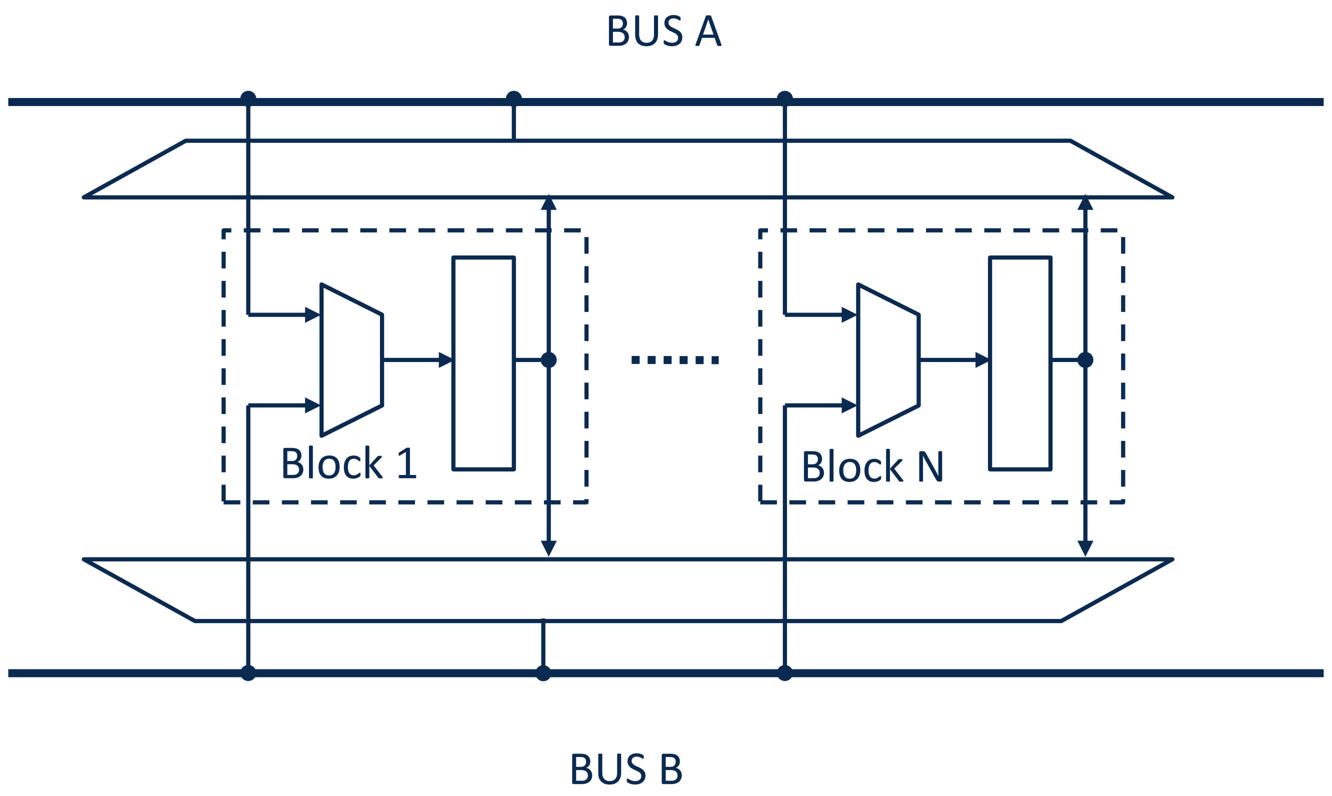
Disadvantage of Tri-State Buses



- In current designs with automated Place & Route (P&R), tri-state signals are avoided!

Multiplexer Buses

- Selecting the data sources for bus using multiplexer
- Selecting bus for the input of data path element using multiplexer
- Multiplexer Select signals are set using control unit



Multiplexer Bus

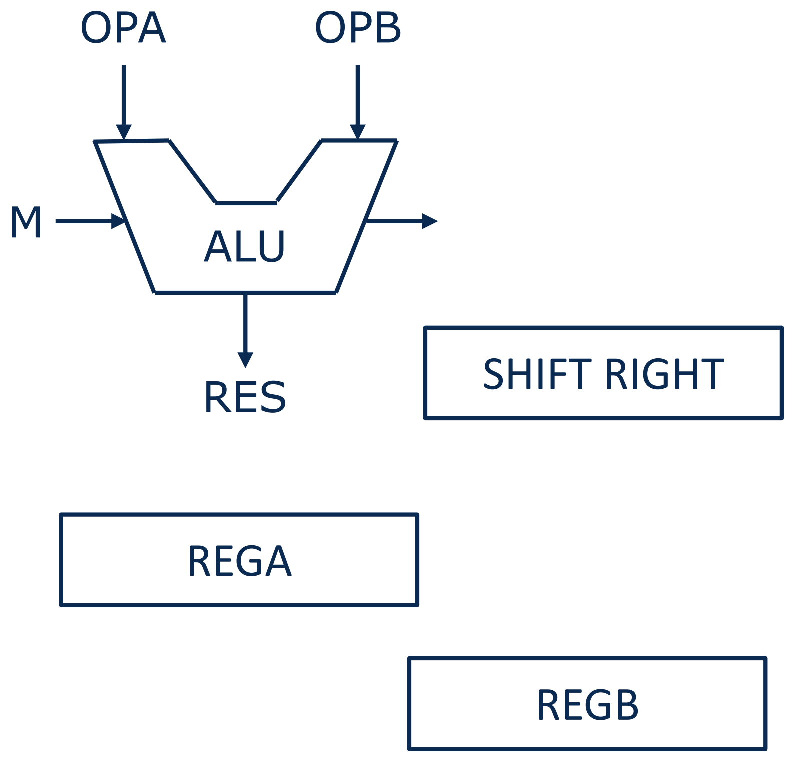
- Advantage: No Tri-State driver

- Number of multiplexer inputs depends on necessary data connections
- Goal: Minimization of the necessary multiplexer inputs
- Possibility of hierarchical implementation of multiplexers
- Bus Multiplexer: Selecting data source for the bus
- Input Multiplexer: Selecting data source for the input of data path element

Example: Bus Implementation

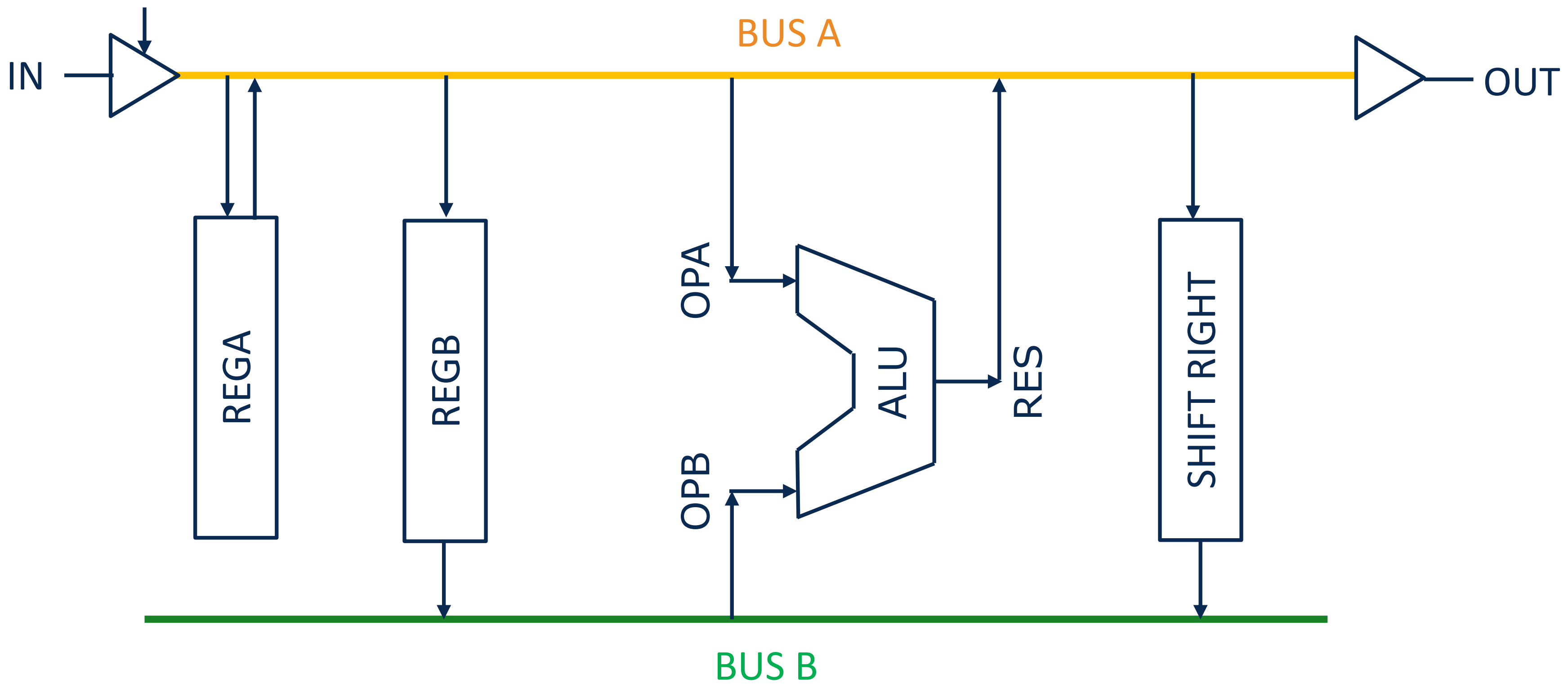
- Example: C=(A+B)+A/2

State	Data source	Number of buses
LOAD A	IN→ REGA	1
LOAD B	IN→ REGB	1
COMP1	REGA → ALU(OPA) REGB → ALU(OPB) REGA → SHIFT	2
COMP2	ALU(RES)→ ALU(OPA) SHIFT→ ALU(OPB)	2
STORE	ALU(RES) → OUT	1

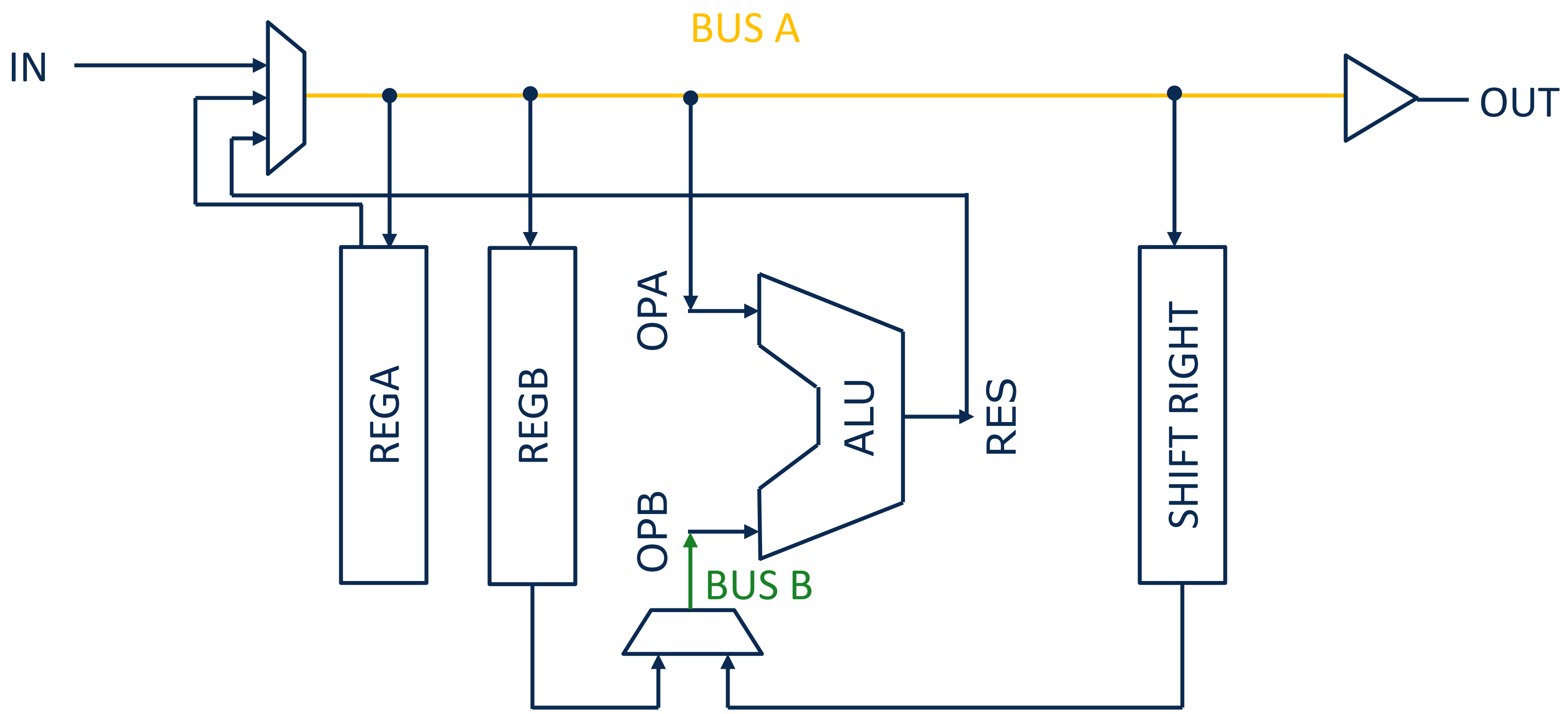


- → 2 Data buses needed (BUS A, BUS B)

Example: Bus Implementation (Tri-State Bus)



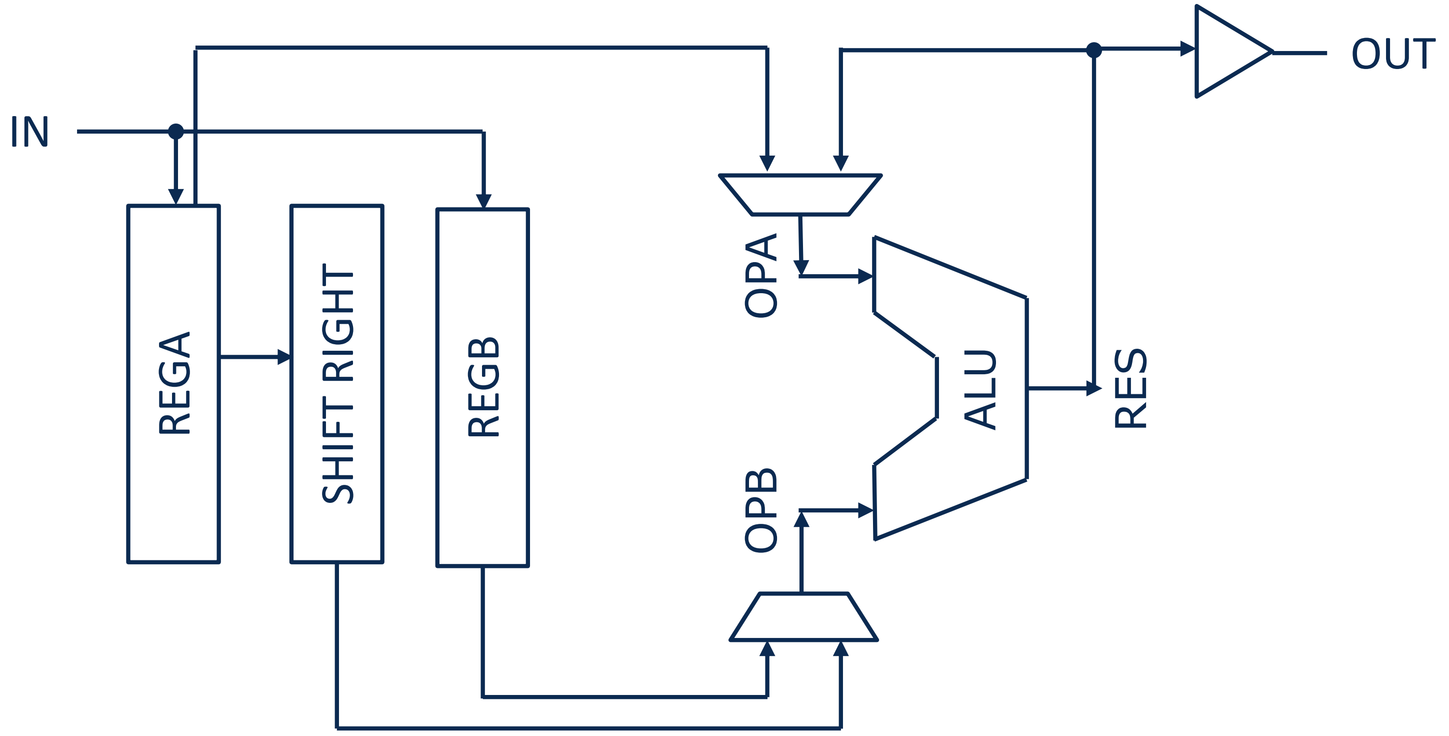
Example: Bus Implementation Multiplexer Bus Variant 1



TECHNISCHE UNIVERSITÄT DRESDEN

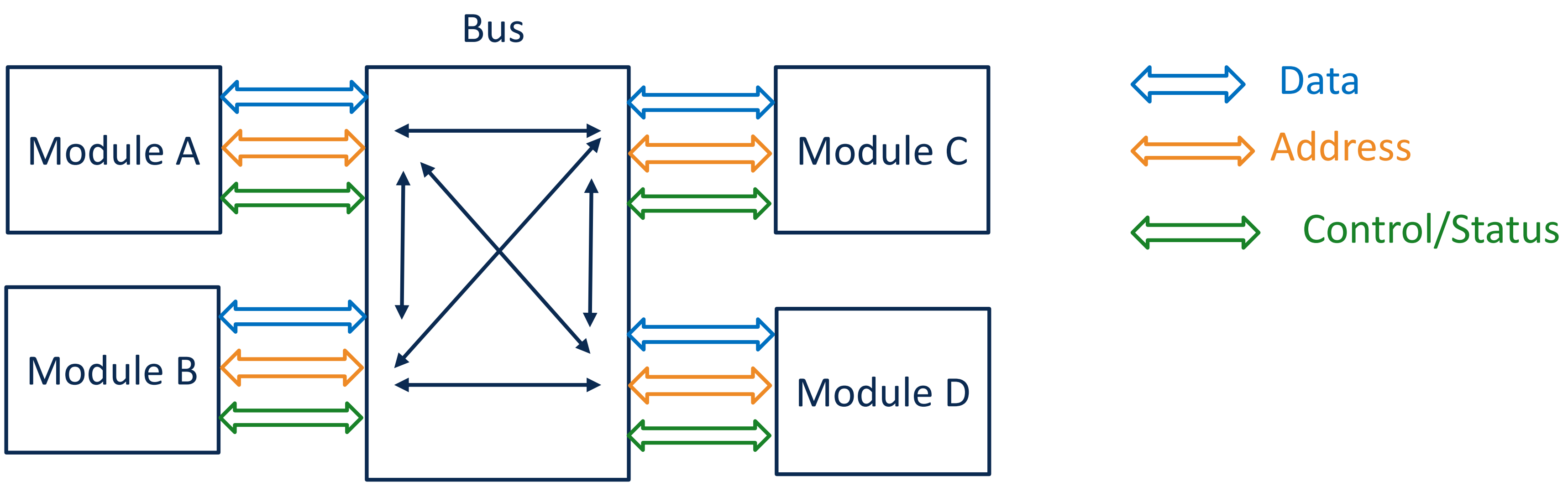
Example: Bus Implementation Multiplexer Bus Variant 2

- Reduction of Multiplexer



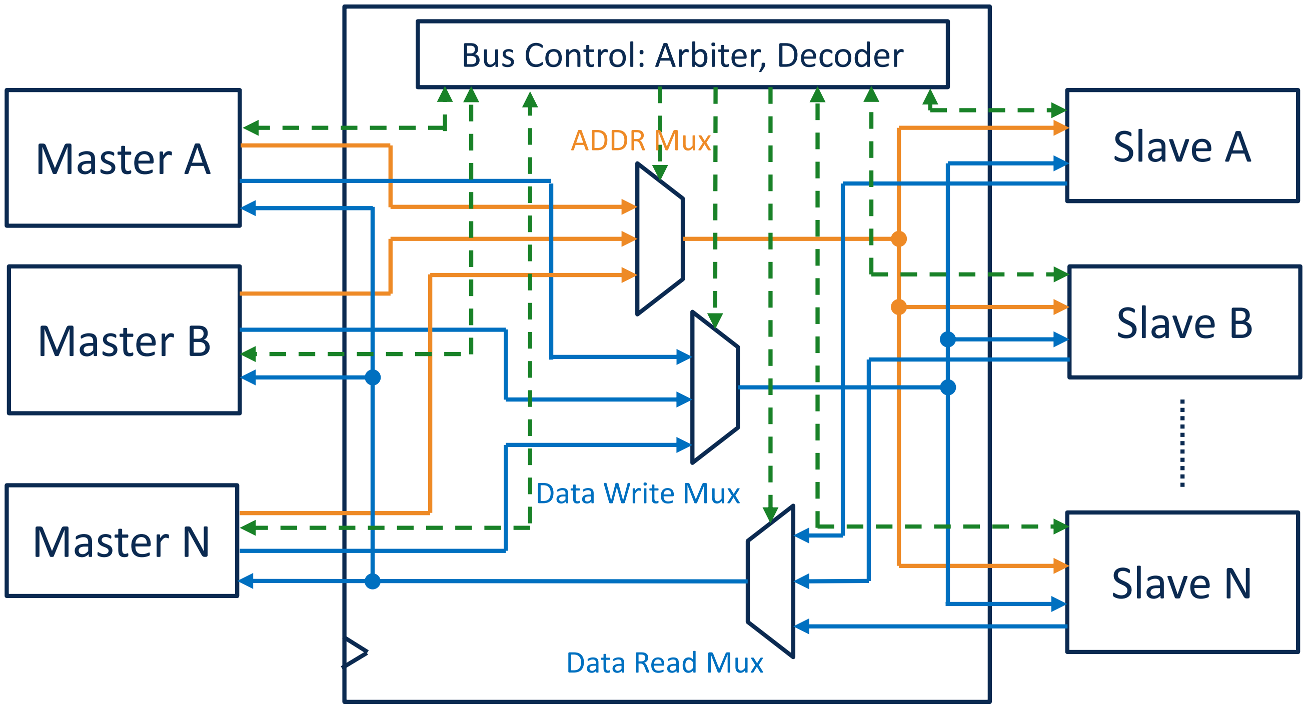
Complex Bus Systems

- The previous bus systems require **application-specific** control by a control unit
- They are not standardized → Integrating prefabricated building blocks (IP) is difficult.
- Complex digital systems require flexible, standardized bus systems
- Basic concept:
    - Standardized bus interface (Data, Address, Control and Status signals)
    - Addressing Bus users (ID)
    - Access via ports (similar to Memory)
    - Internal bus control logic is responsible for data transfer and monitoring the access



Complex Bus Systems– Crossbar Buses

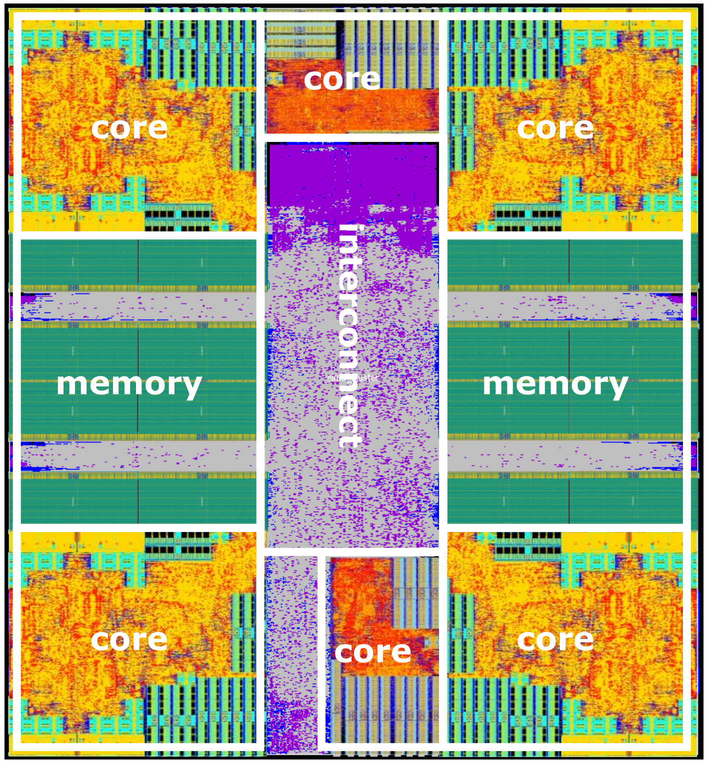
- „Circuit Switched Network“ → Multiplexer Bus
- Master and Slave components with standardized bus interface
- Arbitration and Prioritization of access by Bus Controller
- Synchronous (CLK)
- Sequential access (Address cycle, Data cycle, Wait cycles when Bus „busy“, Burst)
- Examples: ARM AHB, WishBone, …



TECHNISCHE UNIVERSITÄT DRESDEN

Example – Crossbar Buses

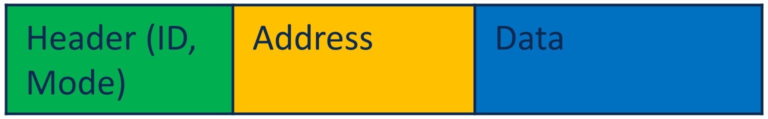
- Advantages:
  - Standardized bus interface
  - Low latency (few clock cycles)
  - Easy to implement (synchronous design)
- Disadvantages:
  - Local synchronous clocking for "global wiring" on the chip
  - Higher costs regarding chip area and power consumption
  - can limit the maximum clock frequency
- Example: Music 2 SIMD Cluster

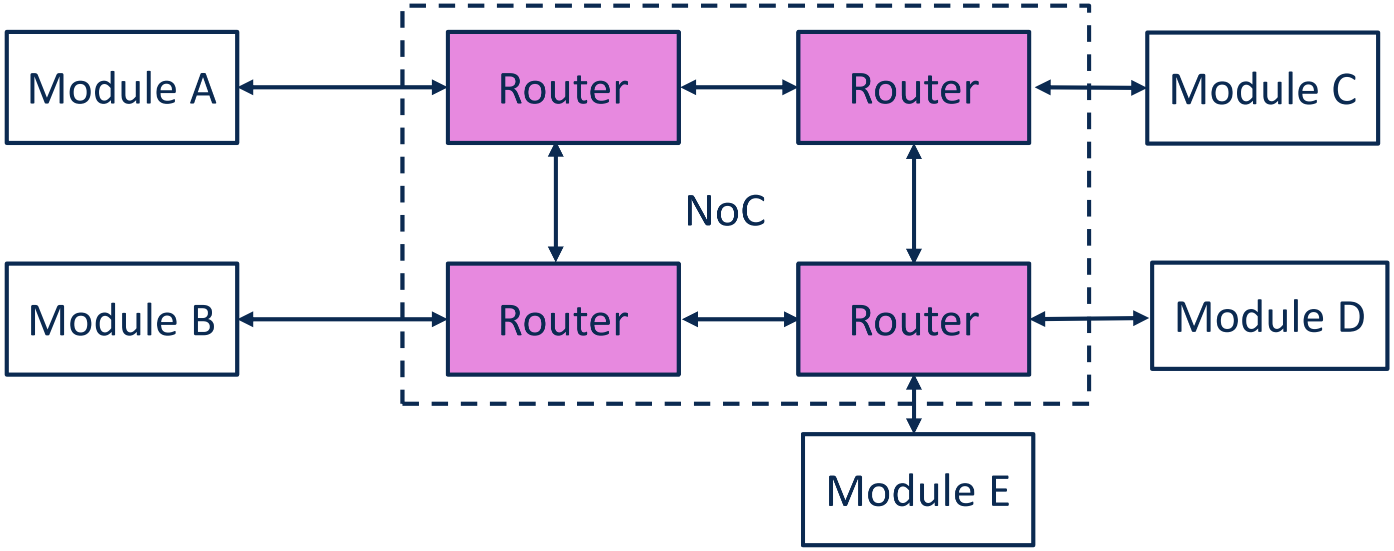


Infineon/IMC MuSiC2 SIMD Cluster (conv. interconnect)

Network on-Chip (NoC)

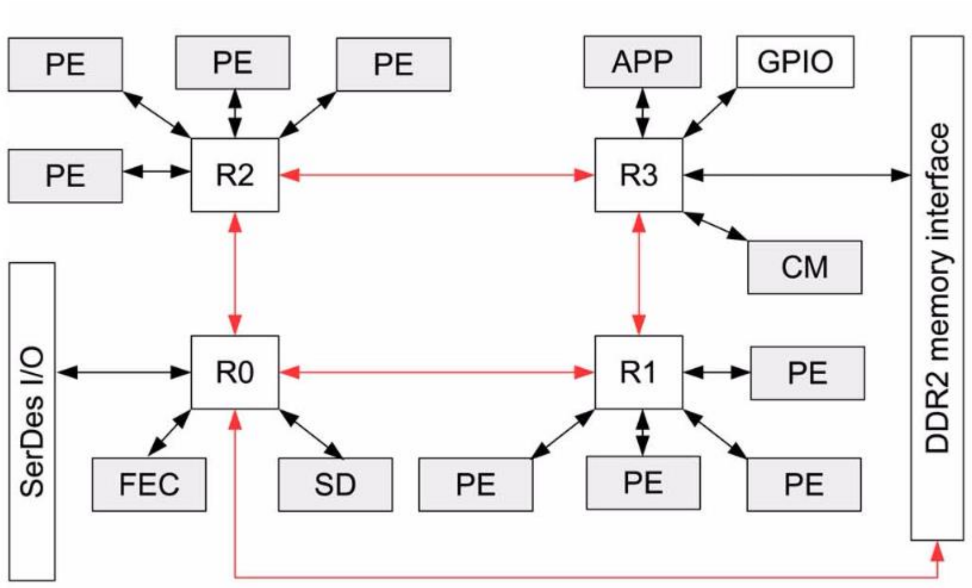
- „Packet Switched Network“ → Routing of Packets through the network (Router)
- Flexible topologies possible
- Standardized package format and components interface
- High performance (data throughput, latency) in complex systems through parallelism
- Global asynchronous realization possible, individual cycles of the individual components

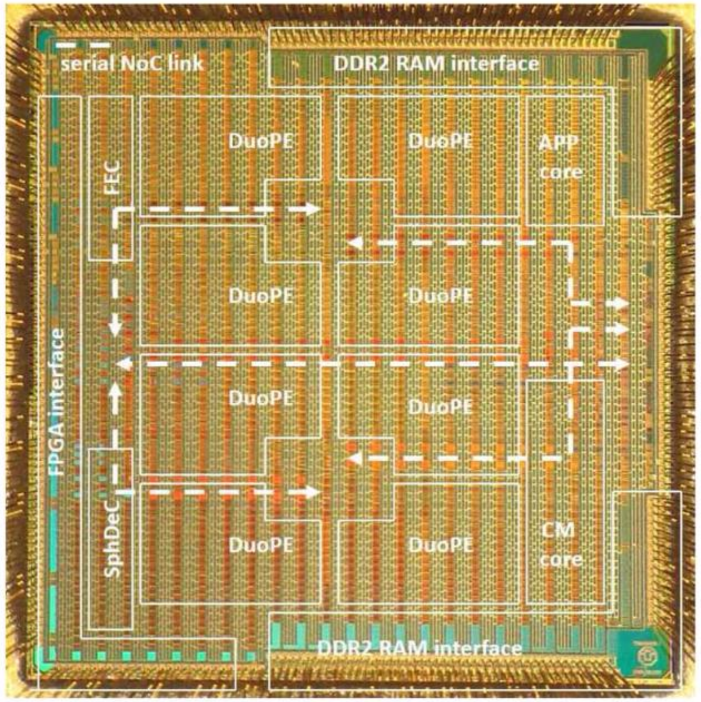




Example: Tomahawk 2 NoC

- Tomahawk2: Software-Defined Radio baseband processor, developed by TUD-MNS and TUD-HPSN
- Connecting system components in a NoC
- Point-to-point connections with fast serial links → compact layout



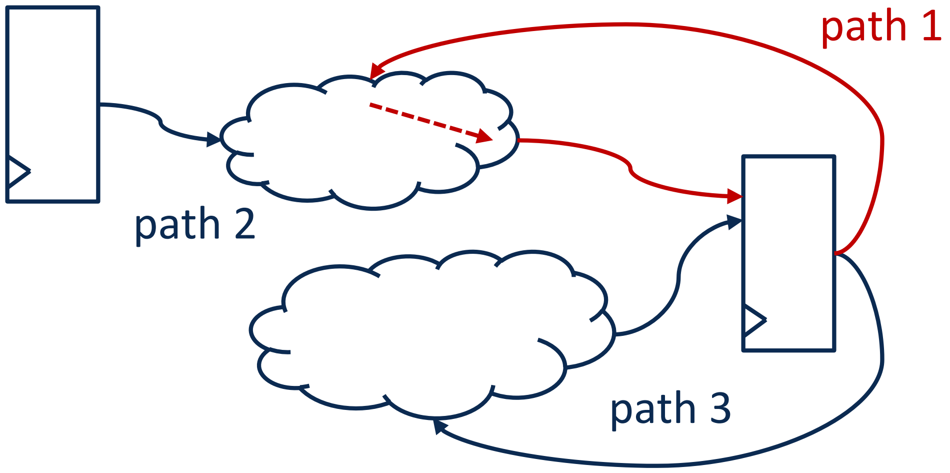


Höppner, Sebastian and Walter, Dennis and Hocker, Thomas and Henker, Stephan and Hänzsche, Stefan and Sausner, Daniel and Ellguth, Georg and Schlüssler, Jens-Uwe and Eisenreich, Holger and Schüffny, René, An Energy Efficient Multi-Gbit/s NoC Transceiver Architecture With Combined AC/DC Drivers and Stoppable Clocking in 65 nm and 28 nm CMOS, IEEE Journal of Solid State Circuits 50 (2015), no. 3, 749-762,

Requirements for Data Paths

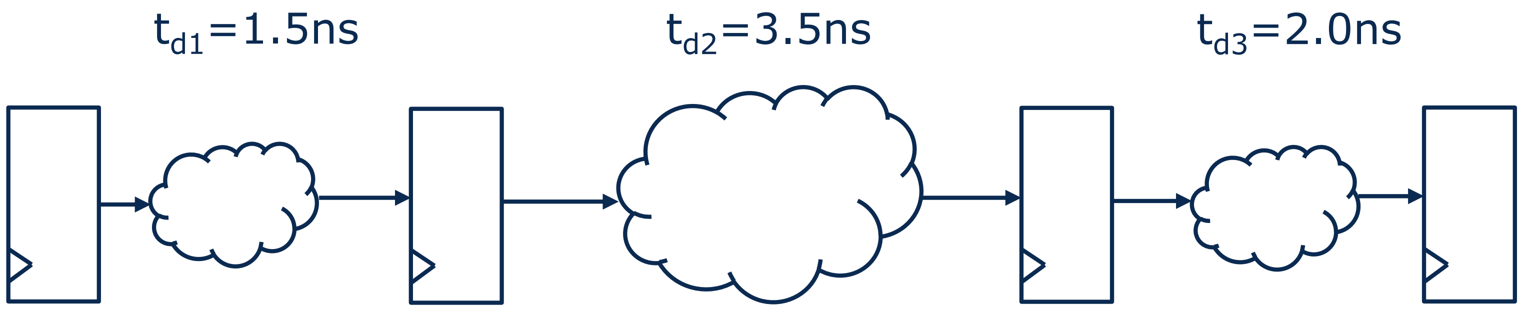
- Clock frequency and Data throughput, measured in
  - [GHz]
  - Operations per seconds [GOPS]
- Chip area, measured in
  - [μm^2]
  - NAND2 equivalents (technology-independent standardization)
- Energy per calculation step, measured in
  - [pJ]

Clock Frequency and Throughput



- Data paths have many timing paths with delay t_{delay}
  - Starting point: register output
  - End point: register input
  - Clock frequency is limited by the **critical path** ($f_{max} \approx 1/\max(t_{delay})$)
- Only few data path elements are critical to timing

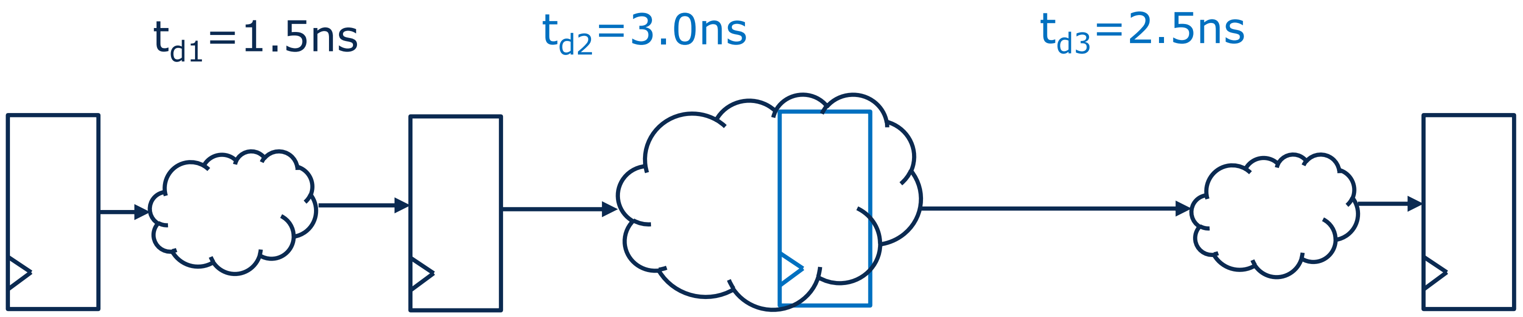
Clock Frequency and Throughput



- f_{max}=1/max(t_{delay}) → 285MHz
- Latency: 4 clock cycles
- Data throughput: 285MHz/4=**71 MOP/s** (without Pipelining)

Clock Frequency and Throughput Optimization

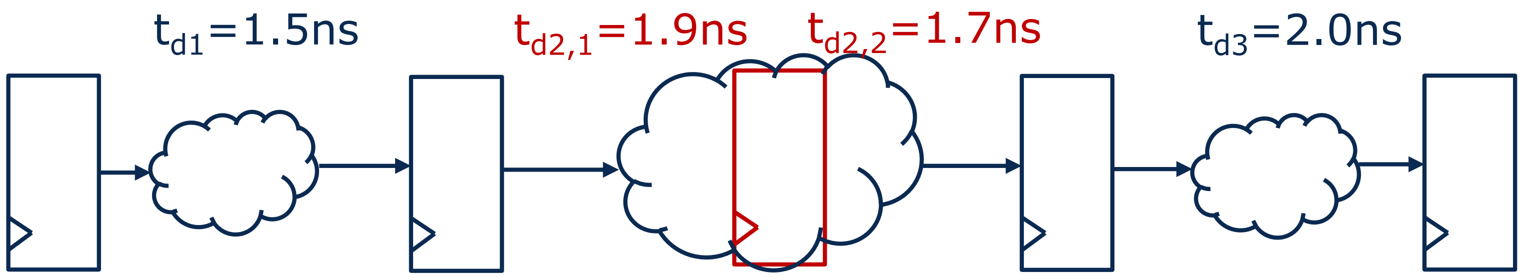
- Logic Re-Timing:



  - f_{max}=1/max(t_{delay}) → 333MHz
  - Latency: 4 clock cycles
  - Data throughput: 333MHz/4=**83 MOP/s** (without Pipelining)
- Can be done automatically by synthesis tools

Clock Frequency and Throughput Optimization

- Inserting register stages:

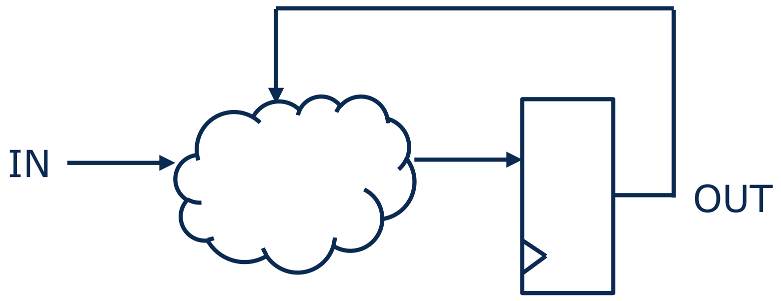


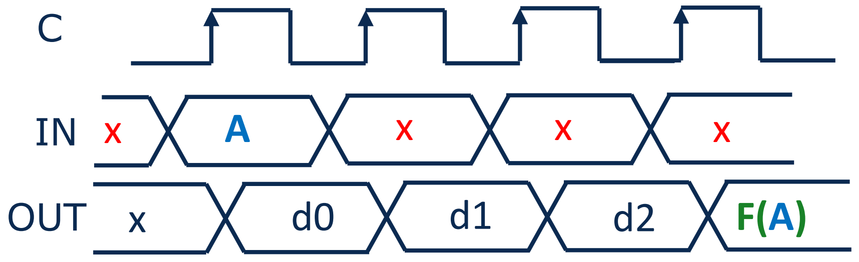
- f_{max}=1/max(t_{delay}) → 500MHz
- Latency: 5 clock cycles
- Data throughput: 500MHz/5=**100 MOP/s** (without Pipelining)

- Arithmetic Circuit IP (RTL) often configurable in terms of number of clock cycles needed (e.g., Synopsys Design Ware)

Sequential Data Path Blocks

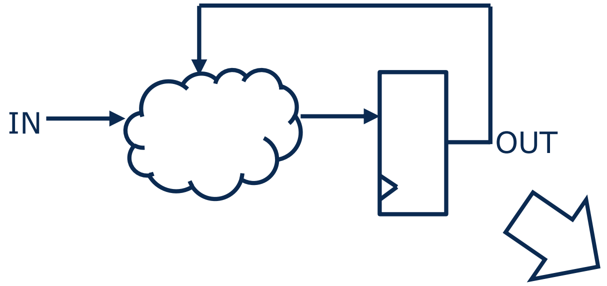
- Implementation of sequential processes
- Register values in time step i are the basis for calculation in step i+1
- Advantages:
  - Shorter logic delay time (calculation in several clock cycles)
  - Low hardware costs (reuse of blocks in different clock cycles)
- Disadvantages:
  - Longer calculation time



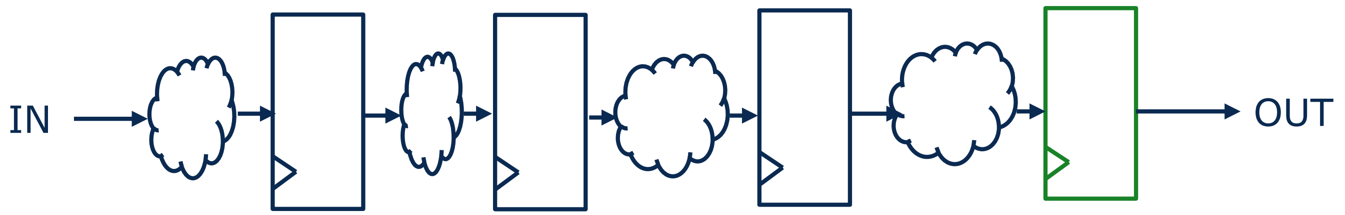


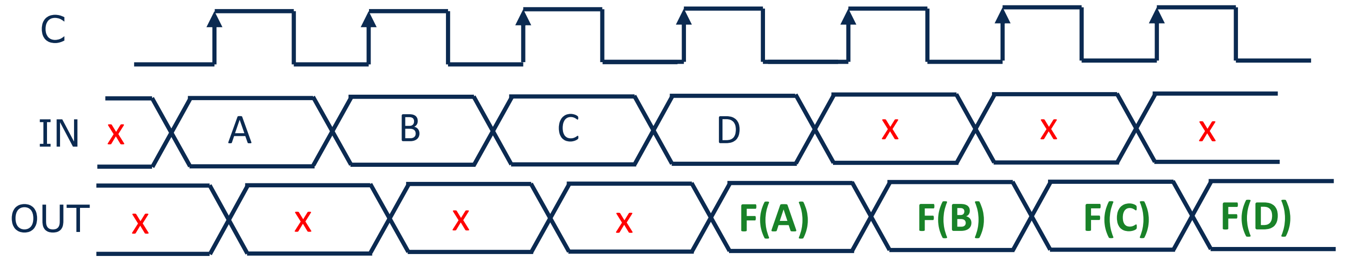
- Latency N clock cycles
- Data rate 1/N operations per cycle

Pipelining



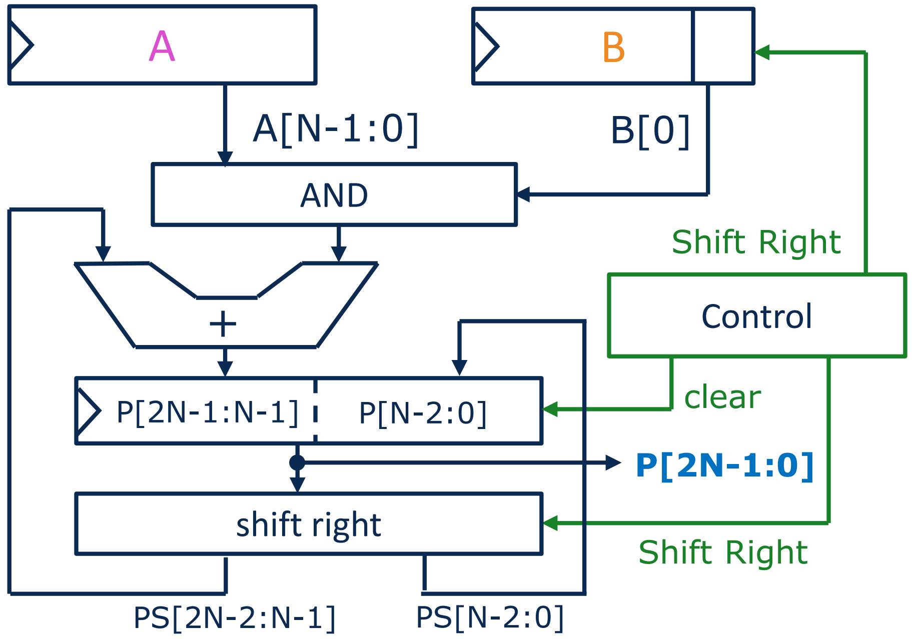
- **Parallel** processing of subtasks (pipelining), by:
  - Splitting the logic per computing step into **separate** circuits ("Loop Unrolling")
  - Or: inserting register stages into combinational arithmetic blocks



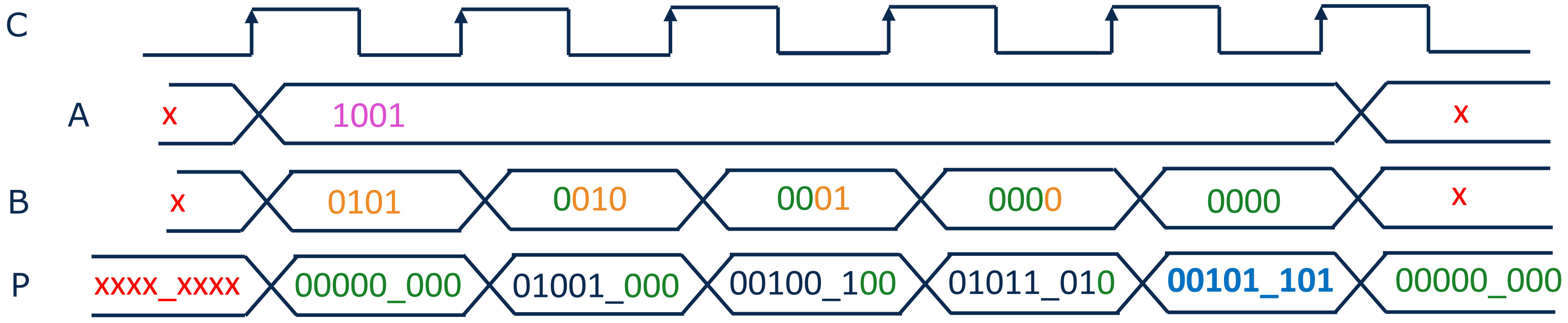


- Latency N clock cycles
- Data rate 1 operation per clock cycle

Example: Sequential Binary Multiplier



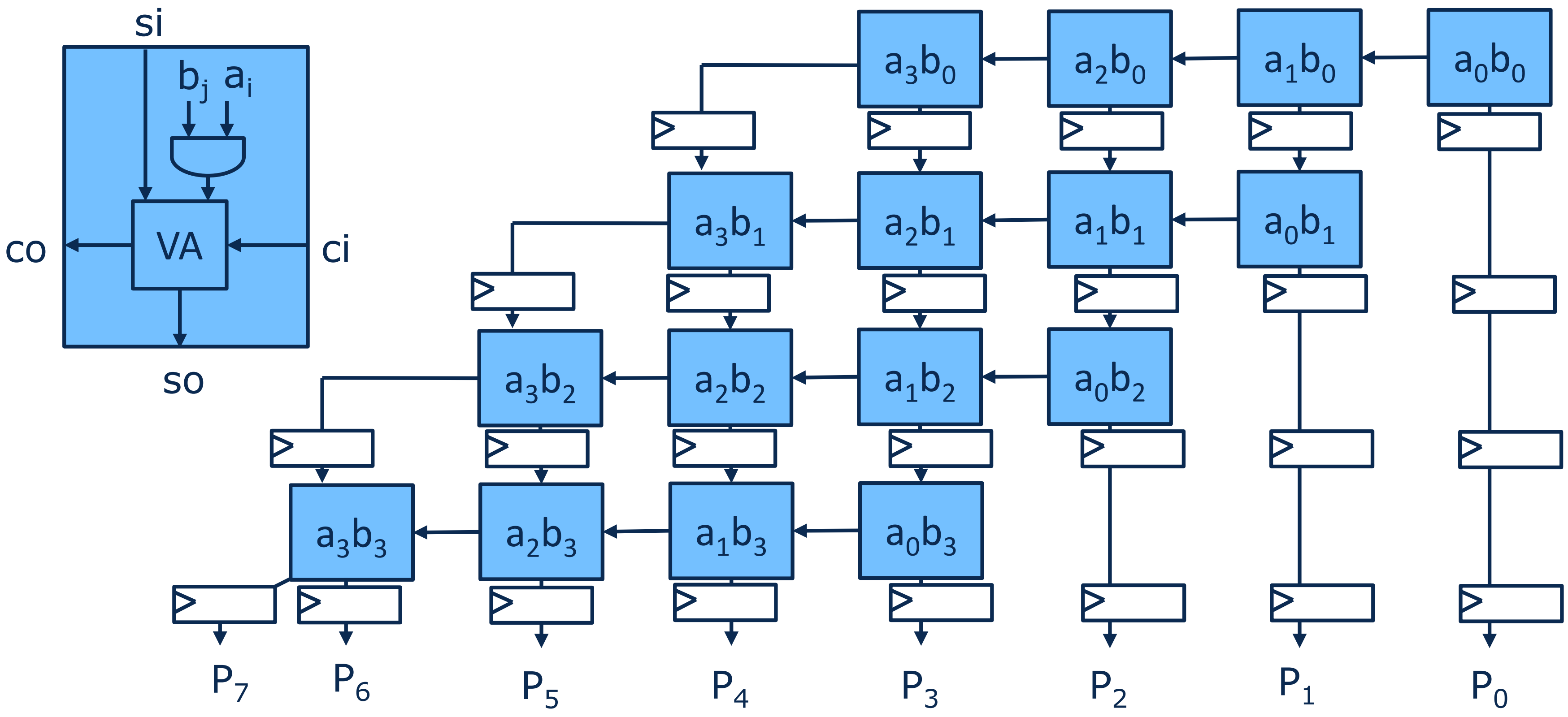
					1	0	0	1	
					1	0	0	1	1
+				0	0	0	0		0
+			1	0	0	1			1
+		0	0	0	0				0
P	0	0	1	0	1	1	0	1	



→ Output in N+1 clock cycles

Example: Pipeline Multiplier

- Inserting register stages in a binary multiplier
- Latency: 4 clock cycles, Data throughput: 1 output/cycle



Summary of Data Paths

- We discussed:
  - Data path building blocks
  - Memory architectures
  - Bus systems
- Timing optimization for sequential data path blocks
  - Inserting register stages
  - Sequential Re-Timing
  - Pipelining

Hardware Implementation of Algorithms

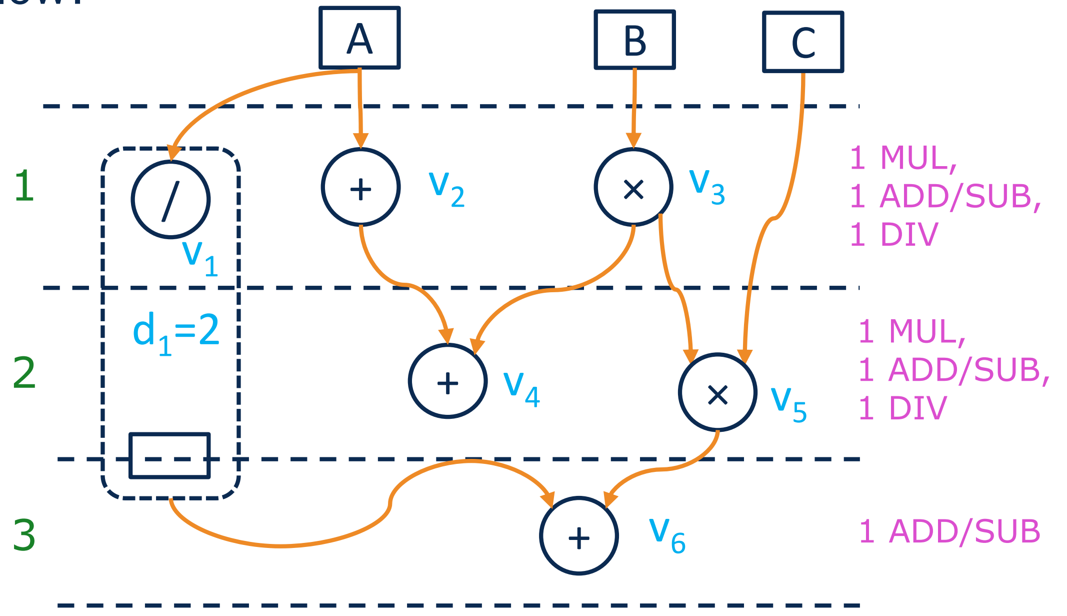
Hardware implementation of Algorithms

- Algorithms can be described as:
  - Text
  - Equations
  - (Pseudo-) Code
  - …

- Scenario 1:
  - Programming a given hardware **(Processor)**
    - Given data path, memory architecture, instruction set
  - **Mapping** the code to the hardware using a compiler
- Scenario 2:
  - Design of Algorithm Specific Hardware (ASIC)
  - **Design** of data path, memory architecture, control flow

Data Flow Graph (DFG)

- Formal representation of data flow:



- Operations $V=\{v_0,v_1,v_2,\ldots\}$
- Delay $D=\{d_0,d_1,d_2,\ldots\}$
- Set of edges $E=\{e_{ij},\ldots\}$
- $Pred_{vi}$ all predecessors of vi
- $Succ_{vi}$ all successors of vi

- The "runtime" of an operation is specified in clock cycles
- Presenting **start time** and **delay time**
- Specifying the necessary **hardware resources**

Scheduling

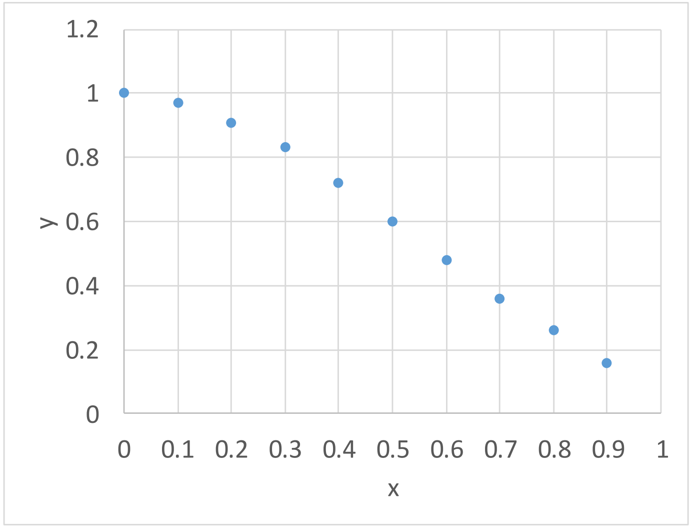
- Determines the schedule of operations
- Consideration of constraints
  - runtime (Timing Constraints)
  - Hardware complexity (Ressource Constraints)
- Scheduling is an optimization problem
  - Sets the start time of operations
  - Follows the given boundary conditions

- Result of scheduling:
  - Creating a DFG
  - Derived from this:
    - Data path
      - Which and how many data path blocks?
      - Which and how many registers?
      - How many buses?
    - Register transfer after designing the control unit

Example Algorithm

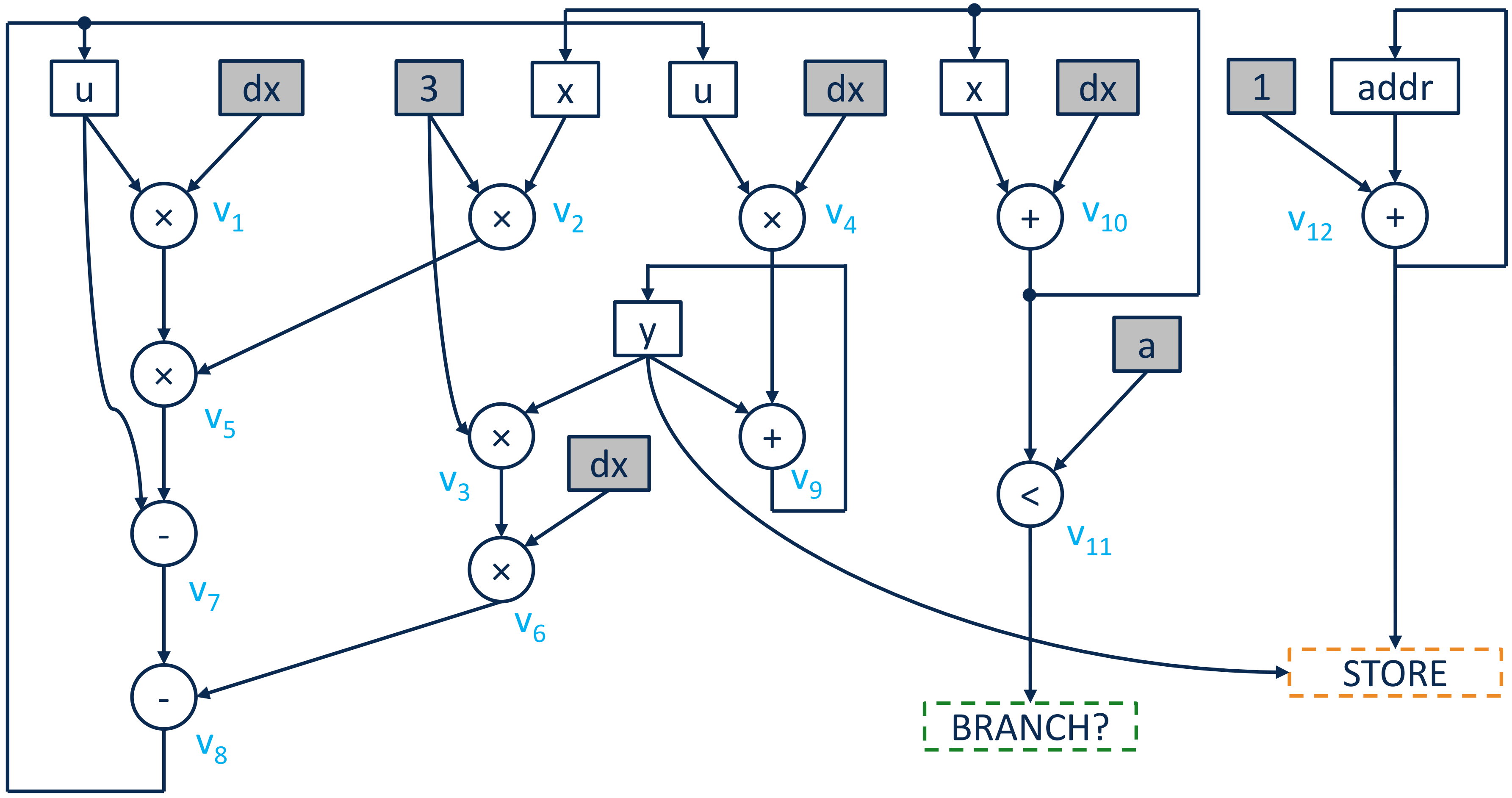
- Numerical DGL solution: $y''+3xy'+3y=0$
- Forward Euler method
- Saving output in RAM to ADDR

```
WHILE (x<a) DO
        x1:=x+dx;
        u1:=u-(3·x·u·dx)-(3·y·dx);
        y1:=y+(u·dx);
        x := x1; u := u1; y :=y1;
        addr := addr+1; STORE(y1,addr);
ENDWHILE
```



Source: Scheduling Algorithms for High-Level Synthesis, Zoltan Baruch

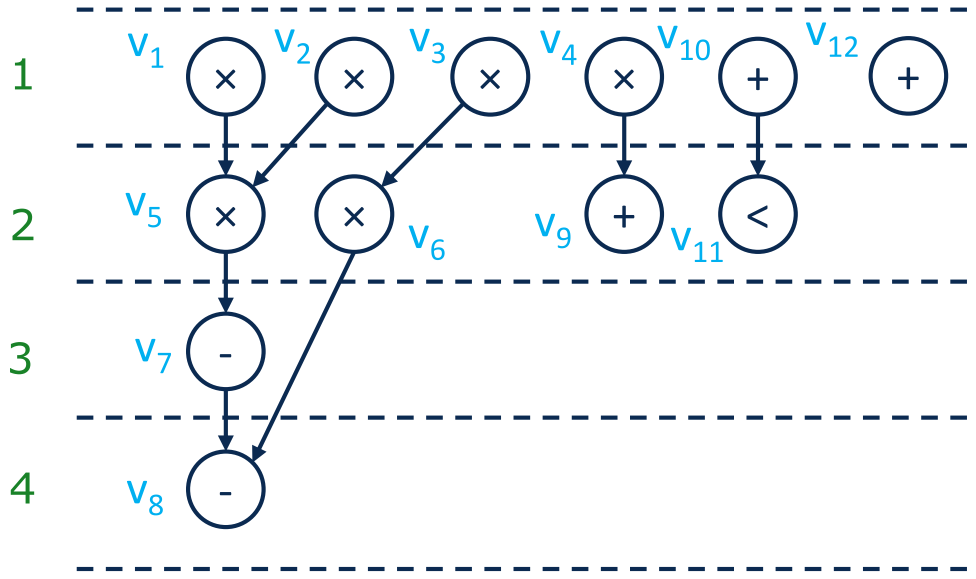
Example Algorithm Data Flow



Source: Scheduling Algorithms for High-Level Synthesis, Zoltan Baruch

As Soon As Possible (ASAP) Scheduling

- Scheduling Algorithm:
  - Repeat for all nodes v_i
  - Select a v_i whose **predecessors** are all assigned
    - Terminate v_i → **t_i=max (t_j +d_j)** for all j from Pred_{vi} (→ earliest time)
  - Until all v_i are assigned



4 MUL, 2 ADD/SUB

2 MUL, 2 ADD/SUB

1 ADD/SUB

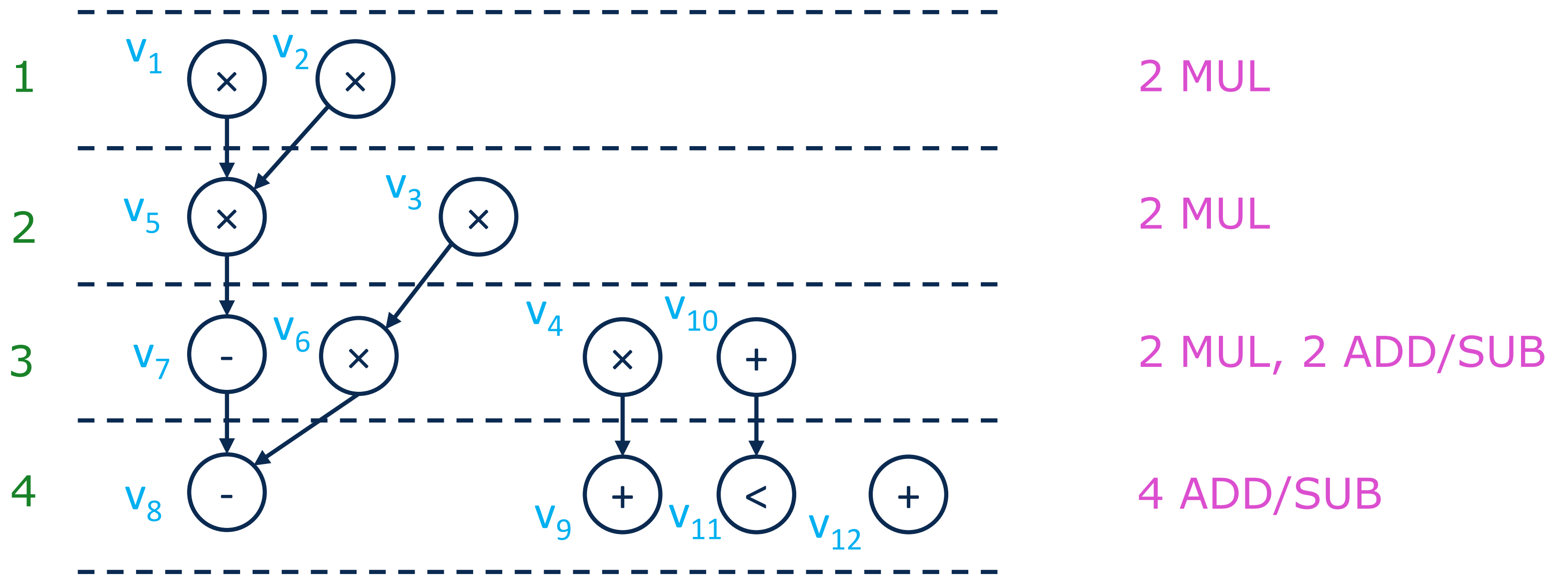
1 ADD/SUB

→ **4 MUL, 2 ADD/SUB**

TECHNISCHE UNIVERSITÄT DRESDEN

As Late As Possible (ALAP) Scheduling

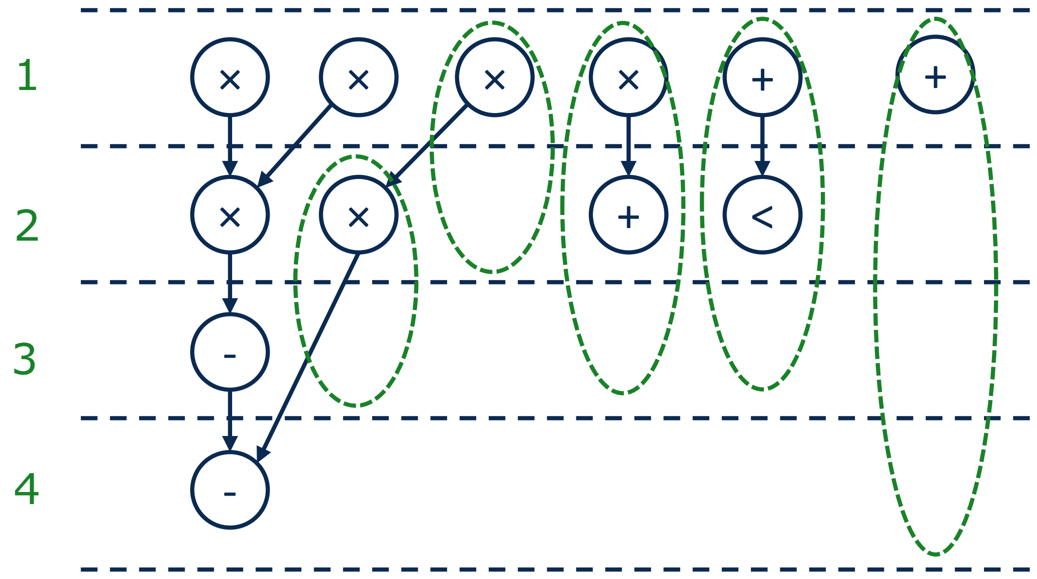
- Scheduling Algorithm :
  - Repeat for all nodes v_i
  - Select a v_i whose **successors** are all assigned
    - Terminate v_i → $\mathbf{t_i = min\ (t_j - d_i)}$ for all j from $Succ_{vi}$ (→ latest time)
  - Until all v_i are assigned



→ **2 MUL, 4 ADD/SUB**

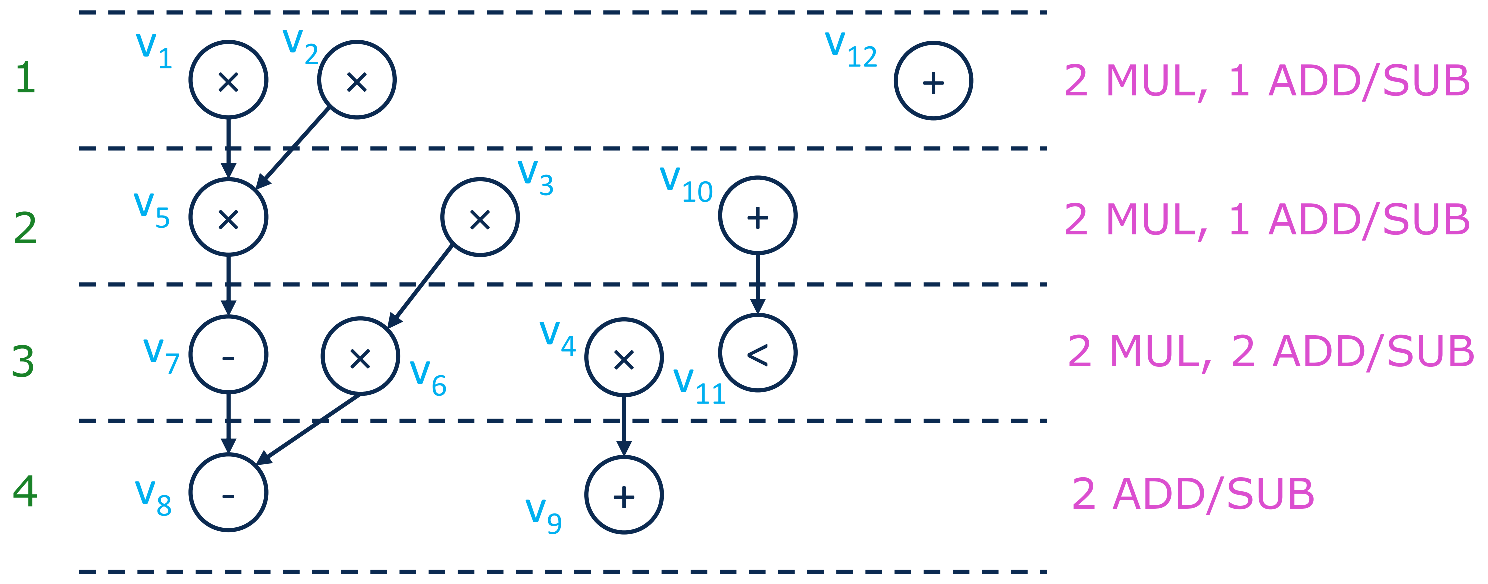
Mobility

- Degree of freedom of start time of operations without affecting the latency of DFG



Resource Constraints

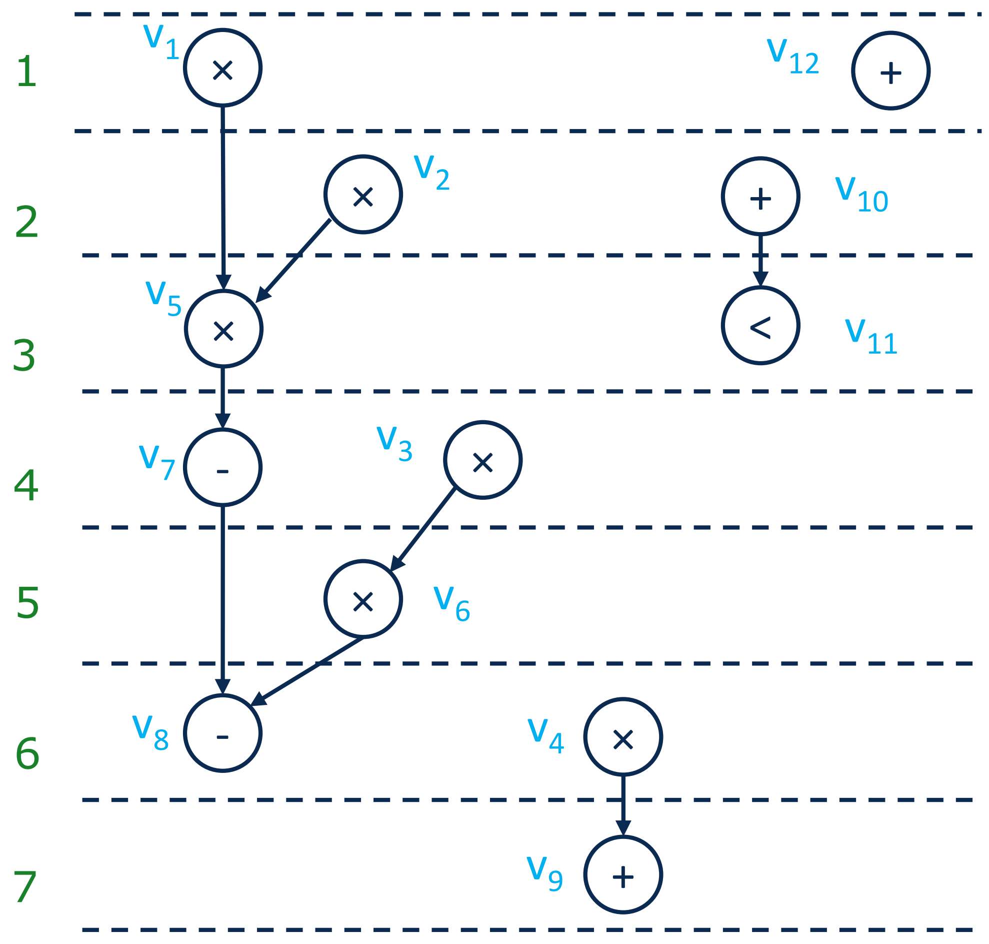
- Scheduling for
  - given hardware resources (chip area)
  - Extreme case: minimal hardware (modules, buses)
  - Use of mobility (no increase in latency):



→ 2 MUL, 2 ADD/SUB

Example: Scheduling for Minimal Hardware

- Inserting additional clock cycles → Increasing the latency



1 MUL, 1 ADD/SUB

1 MUL, 1 ADD/SUB

1 MUL, 1 ADD/SUB

1 MUL, 1 ADD/SUB

1 MUL

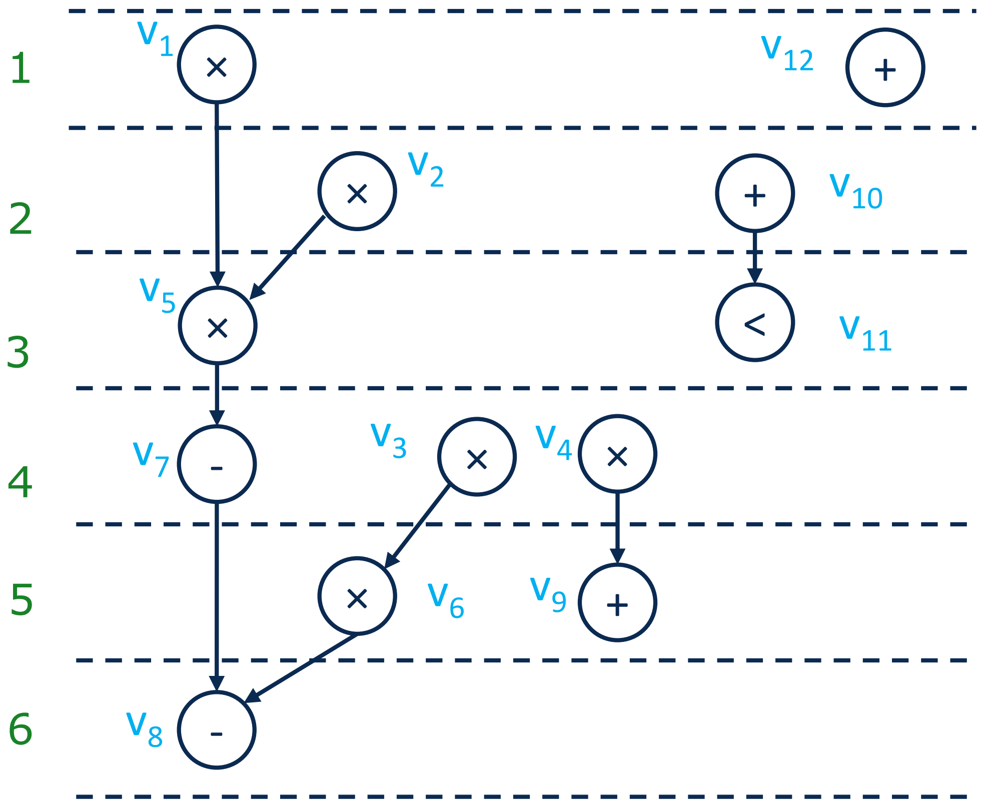
1 MUL, 1 ADD/SUB

1 ADD/SUB

→ 1 MUL, 1 ADD/SUB

Time Constraint Scheduling

- Scheduling for a given processing time
- Application e.g. digital signal processors (DSP) with real-time requirement → Output must be available according to the given number of cycles
- Example: Minimal Hardware with latency = 6 clock cycles



1 MUL, 1 ADD/SUB

1 MUL, 1 ADD/SUB

1 MUL, 1 ADD/SUB

2 MUL, 1 ADD/SUB

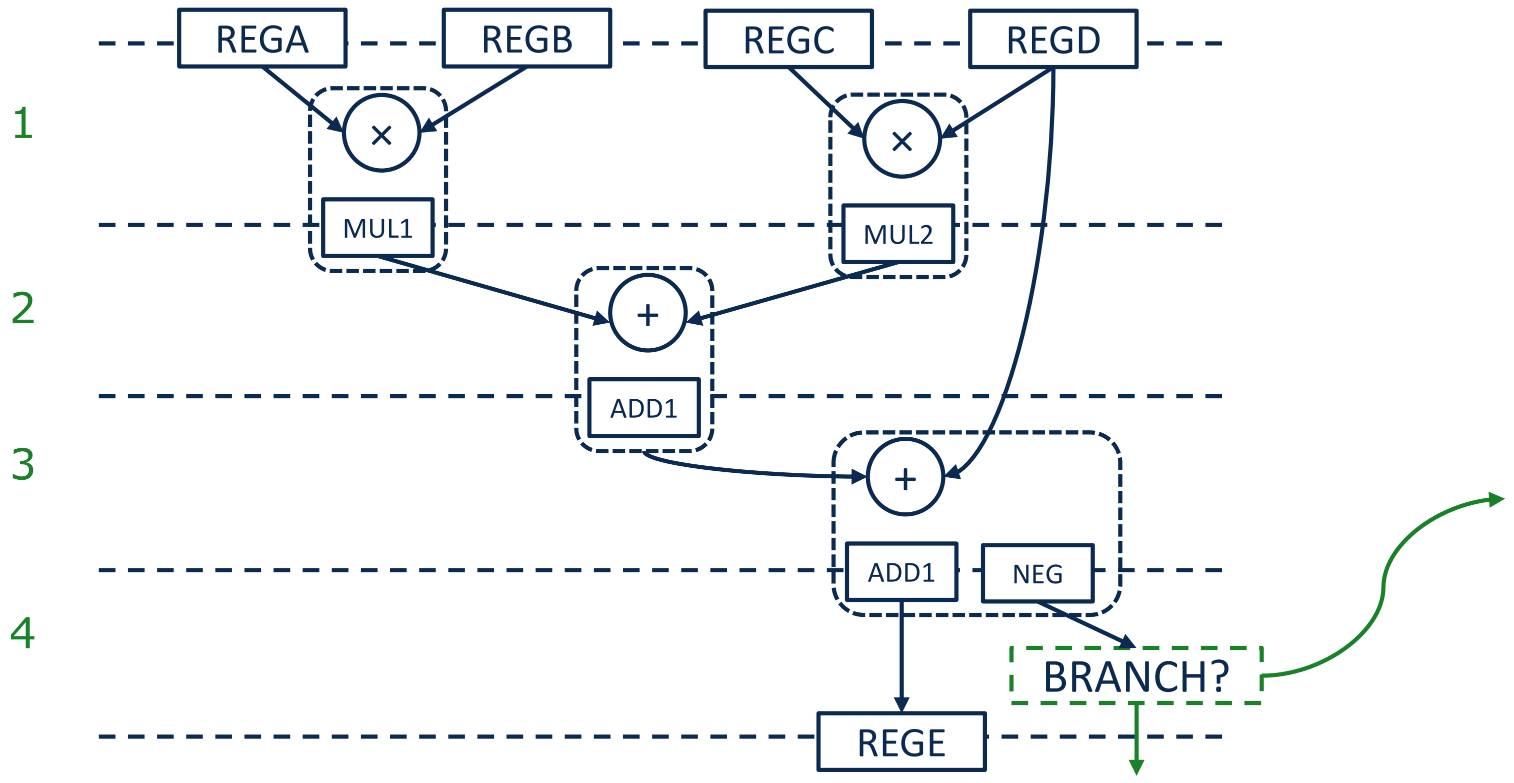
1 MUL, 1 ADD/SUB

1 ADD/SUB

→ 2 MUL, 1 ADD/SUB

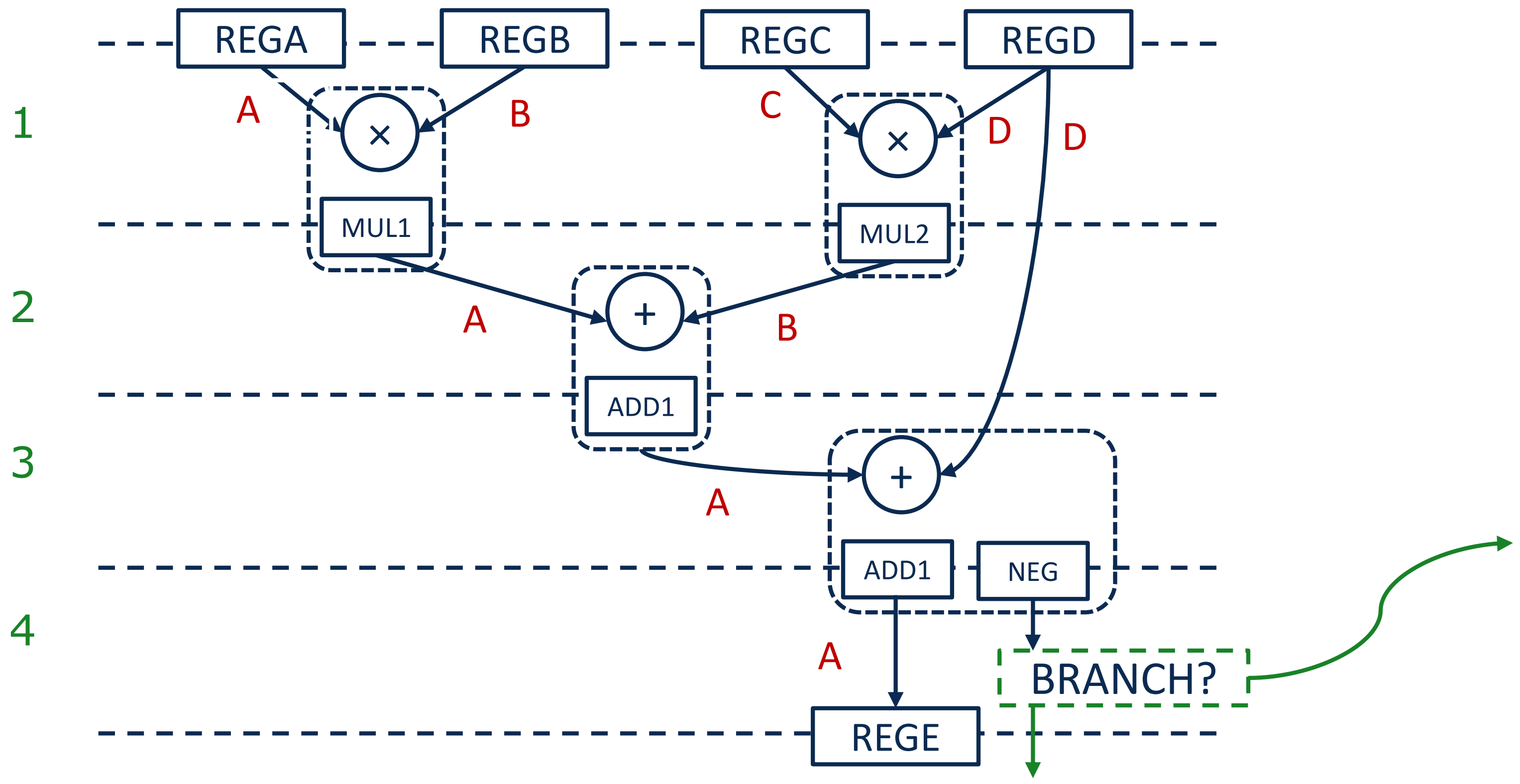
Registers and Branches in DFG

- Detailed DFG with explicit representation of registers
- Visualization of data storage → Hardware register
  - Basic for designing pipelines
  - Explicit representation of branches



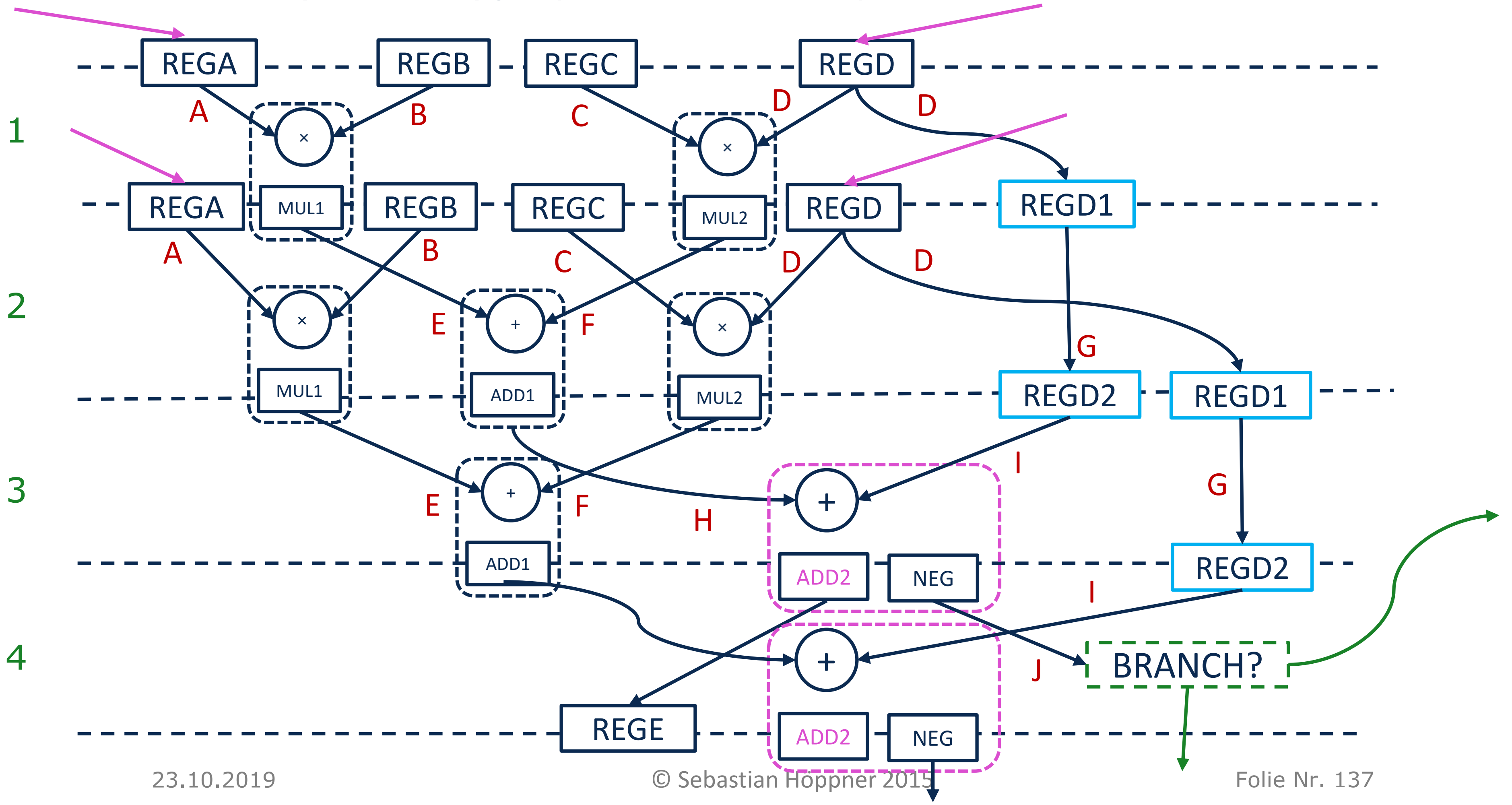
Data Buses in DFG

- Assigning data buses to edges of the DFG
- → Determines the number of necessary data buses



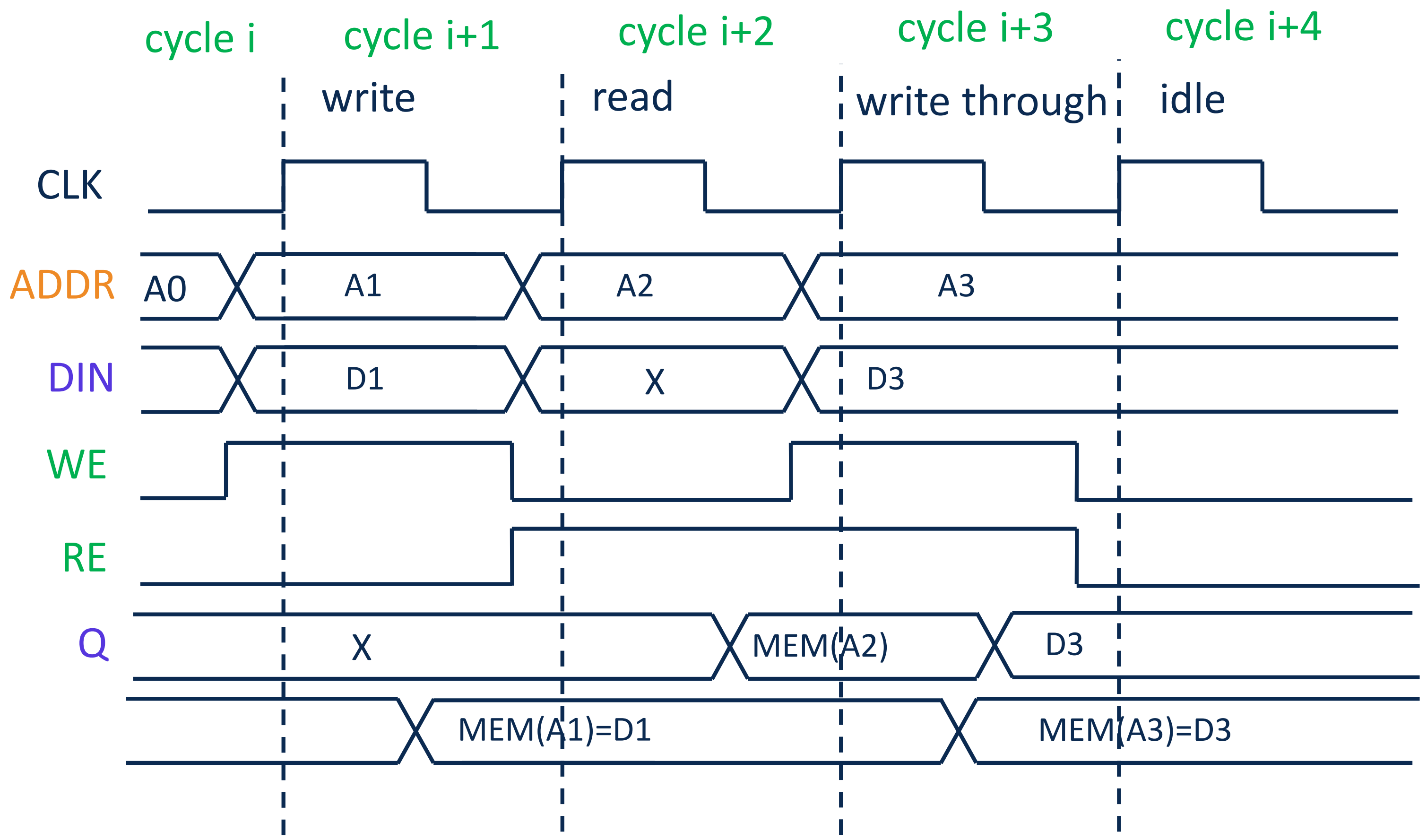
Pipelining

- Building pipelines by adding (if needed):
- Registers, Data path blocks, Buses
- Throughput of 1 output per clock cycle at N clock latency
- Be aware of (conditional) jumps! → → if necessary discard results



Memory Access in DFG

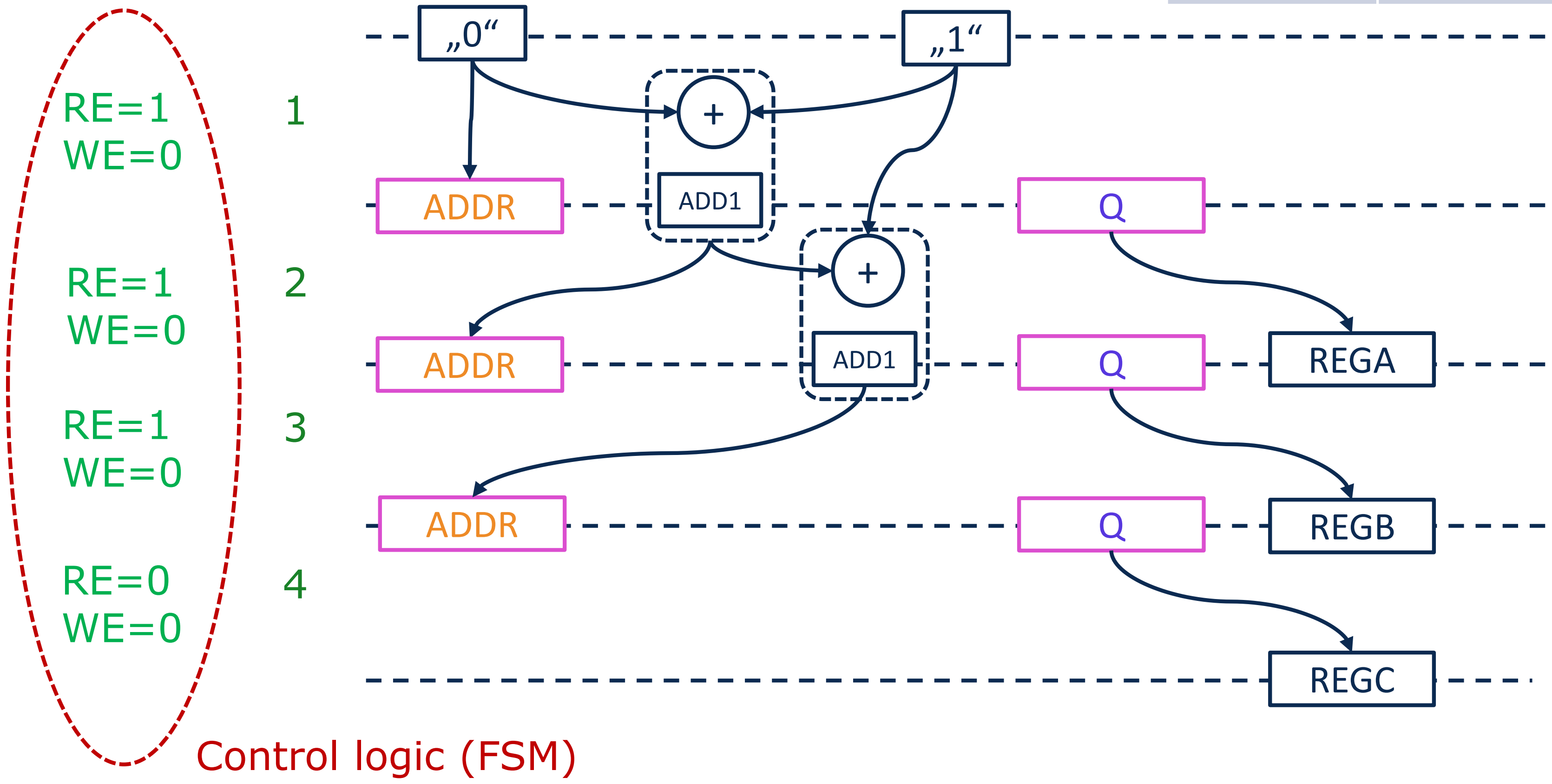
- Sequential SRAM access



Memory Access and Read from SRAM

- Calculate the address through Fixed-Point ALU
- Consider SRAM signals such as registers
- Example:

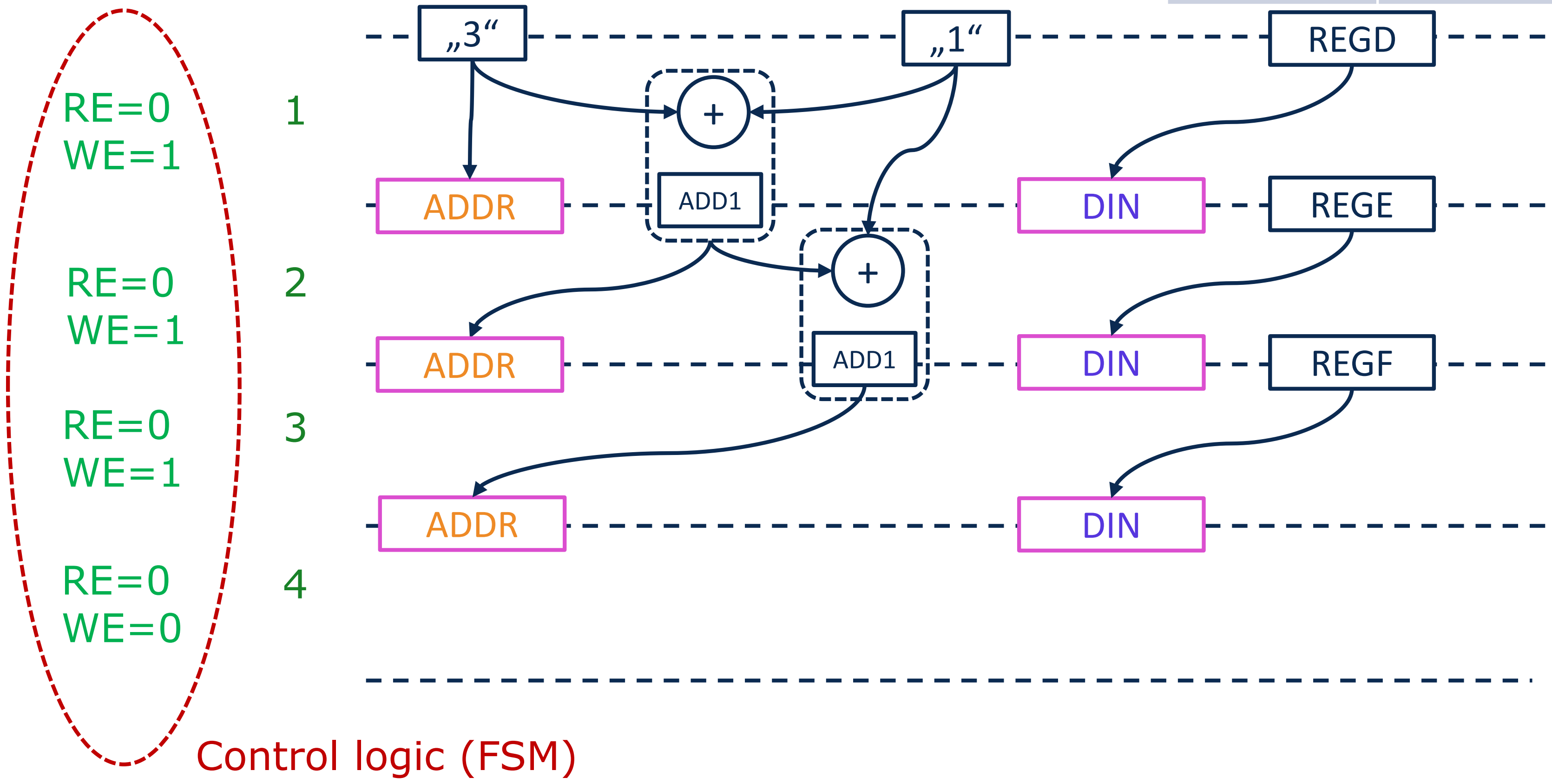
Address	Action
0	Read to REGA
1	Read to REGB
2	Read to REGC



Memory Access and Write to SRAM

- Calculate the address through Fixed-Point ALU
- Consider SRAM signals such as registers
- Example:

Address	Action
3	Write from REGD
4	Write from REGE
5	Write from REGF



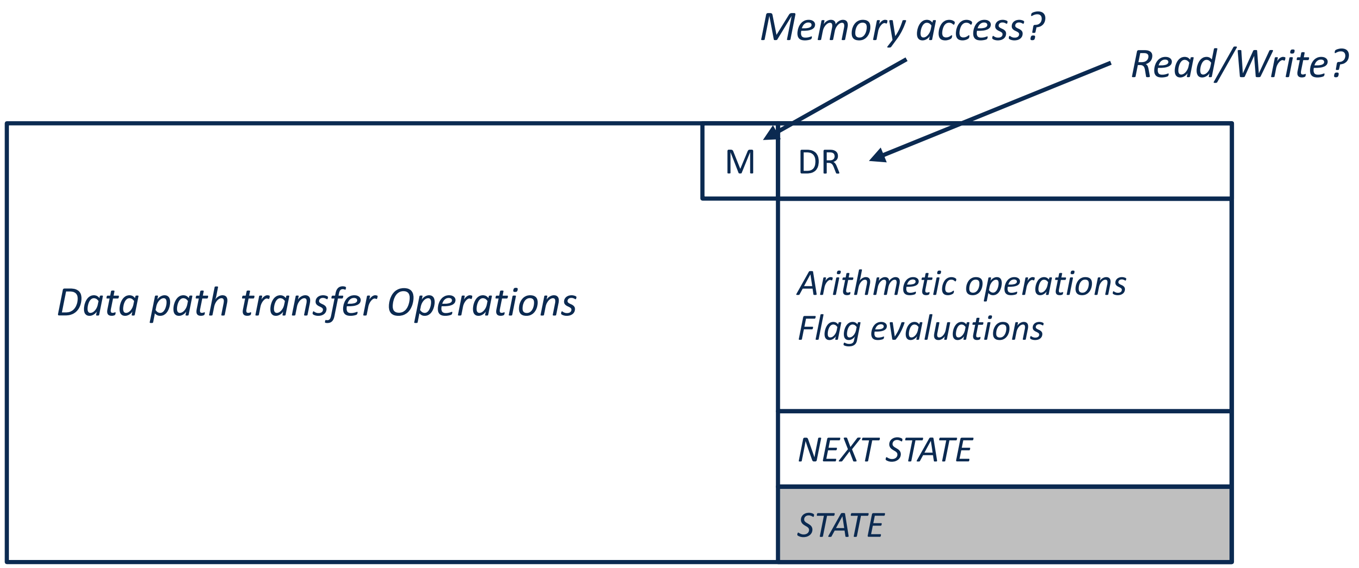
Data Path

- Constructing data path from a DFG
- Determine the number of
  - Modules
  - Registers
  - Buses/ data connections
- Design of data path with bus system (for example multiplexer) → see section Data paths / Buses

TECHNISCHE UNIVERSITÄT DRESDEN

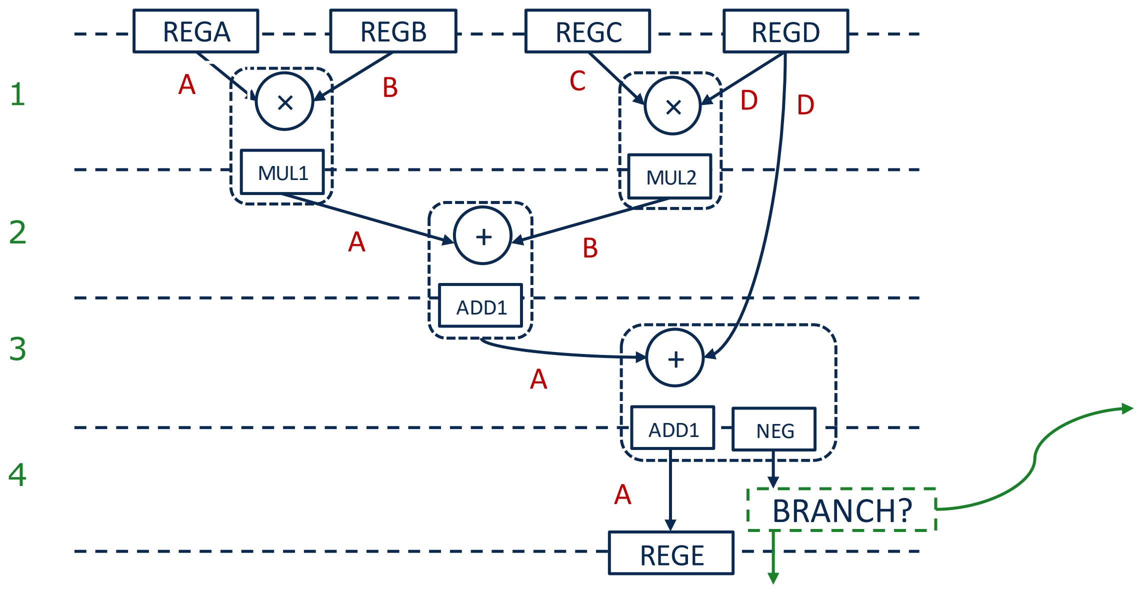
Register Transfer Representation

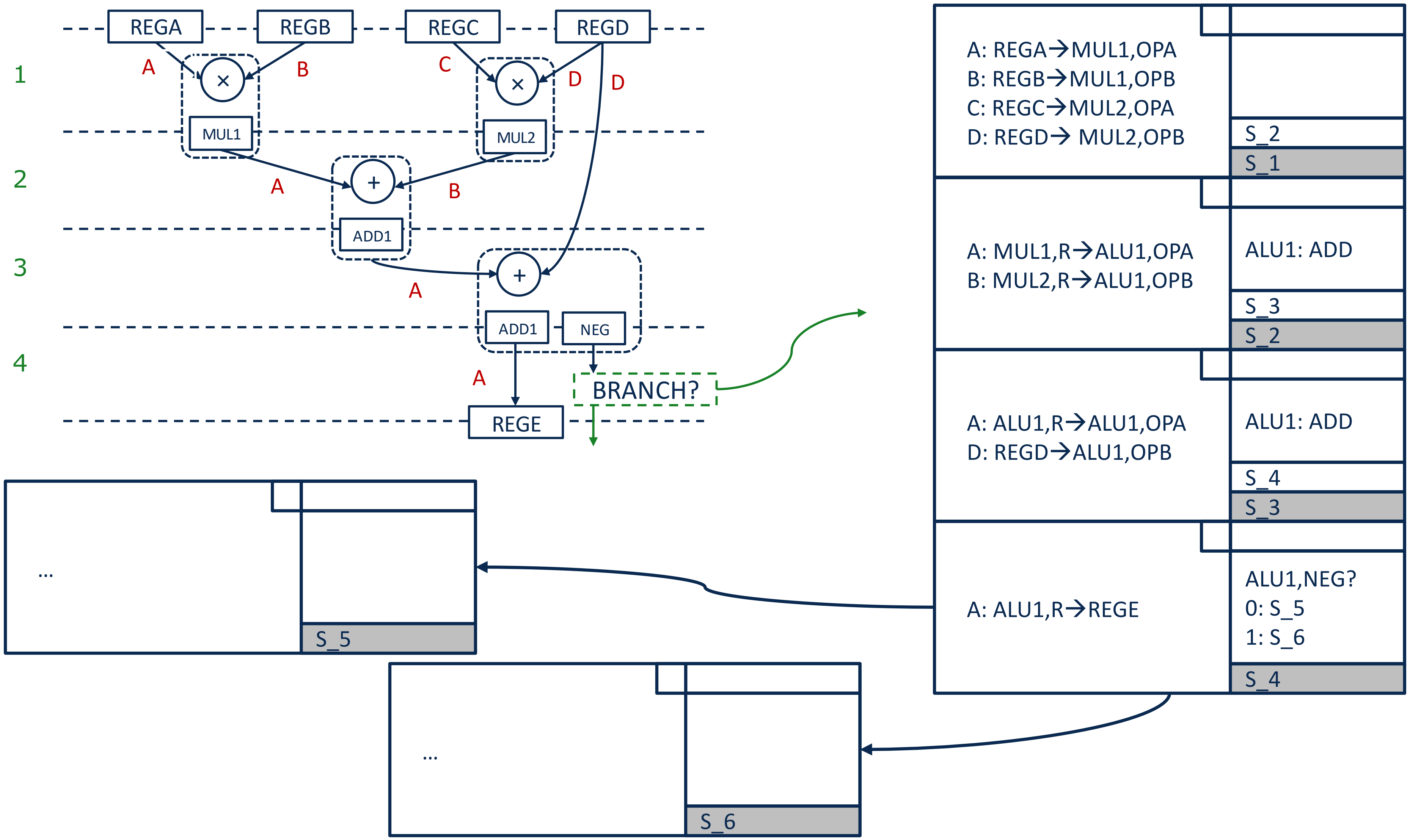
- Representation of processes in Register Transfer Notation



- A register transfer block describes **one** clock cycle

Example RT-Sequence





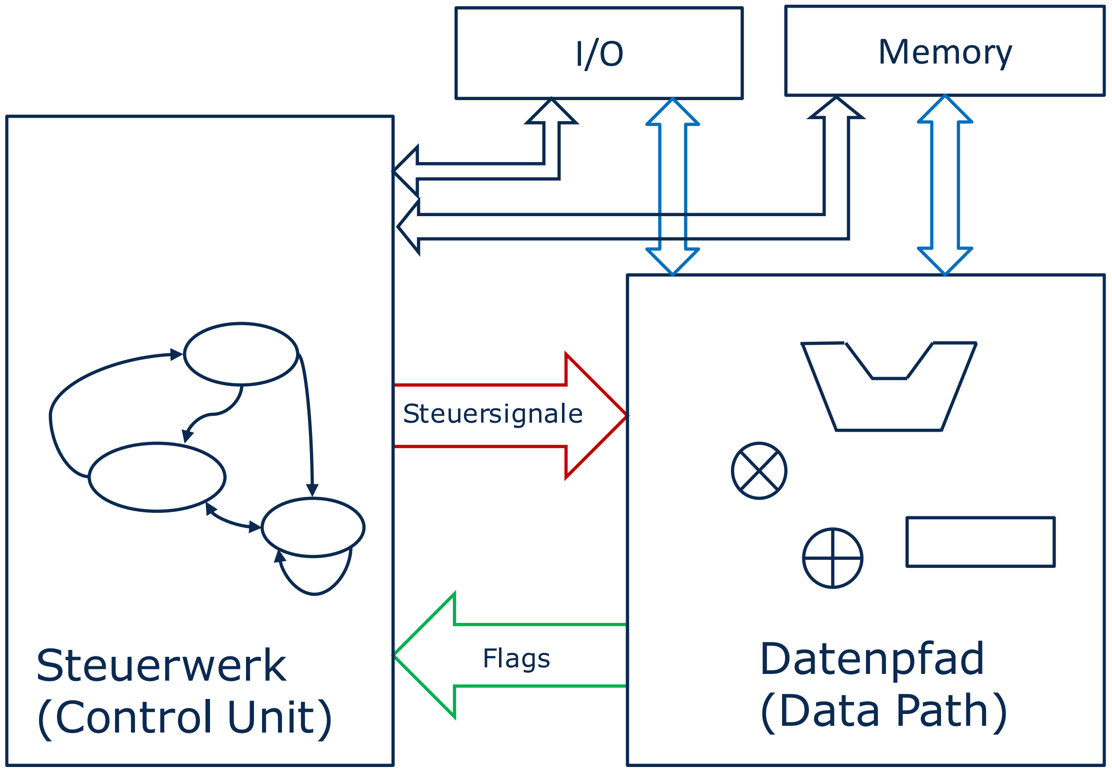
TECHNISCHE UNIVERSITÄT DRESDEN

State Machine and Control Logic

- RT representation defines the states and state transitions of FSM
- Conditions: Flags
- RT representation defines the signals → output logic of FSM
- Switching buses (tristate driver, multiplexer)
  - Configuring data path blocks (ADD / SUB)…
- → see section FSM

Summary of Algorithms

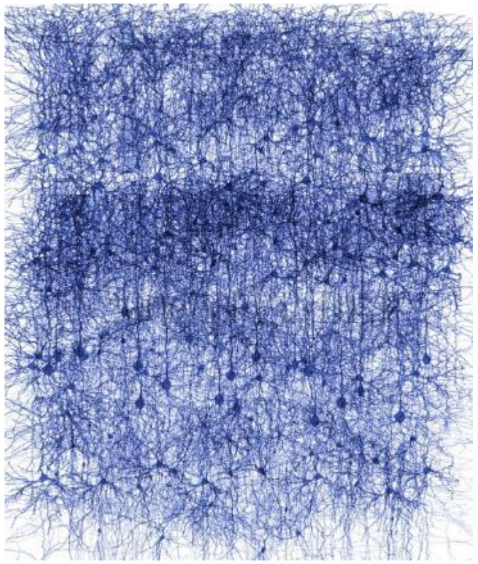
- Presentation of the data flow graph, scheduling and optimization
- ASAP, ALAP, Resource Constraints and Timing Constraints
  - Pipelining
- Register Transfer Sequence → Derive data path and FSM



Example: Exponential function for Neuromorphic Hardware

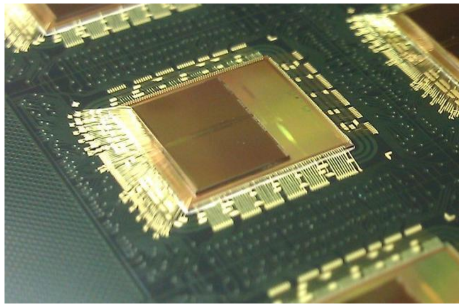
- Neuromorphic hardware for the Human Brain Project
  - Brain simulation
  - Technical applications, e.g. robotics
  - Challenges:
  - Simulation of a **large number** of neurons and synapses in **biological real time**
- Development of a many-core computer with> 4 million ARM processors (SpiNNaker 2)
  - Architecture Development → University of Manchester
  - Chip Design → TU Dresden

Human Brain Project



http://bluebrain.epfl.ch

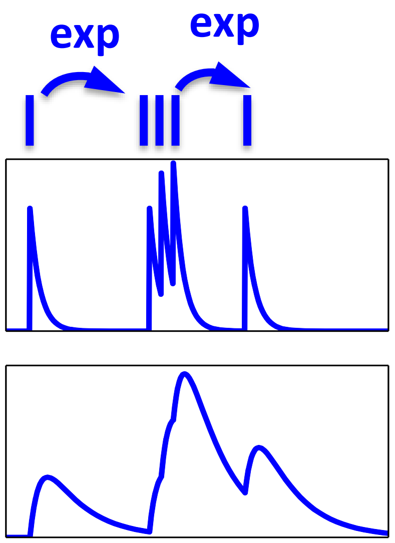
Source: Human Brain Project

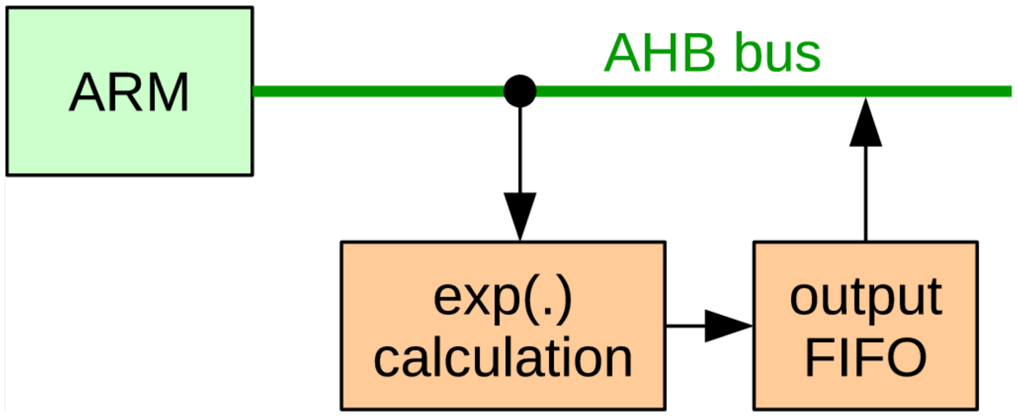


SpiNNaker 1 Chip

Example: Exponential function for Neuromorphic Hardware

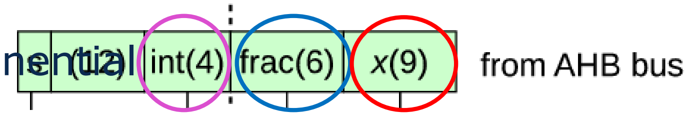
- Calculation of dynamics of neurons and synapses
- Often used: **e^x**
- Number format: 32-bit, s16.15 fixed-point
- Processor Core (ARM Cortex M4)
  - So far **exp()** realized in software (≈30 clocks)
  - → hardware accelerator for **exp (),**
  - Connection as AHB slave to processor

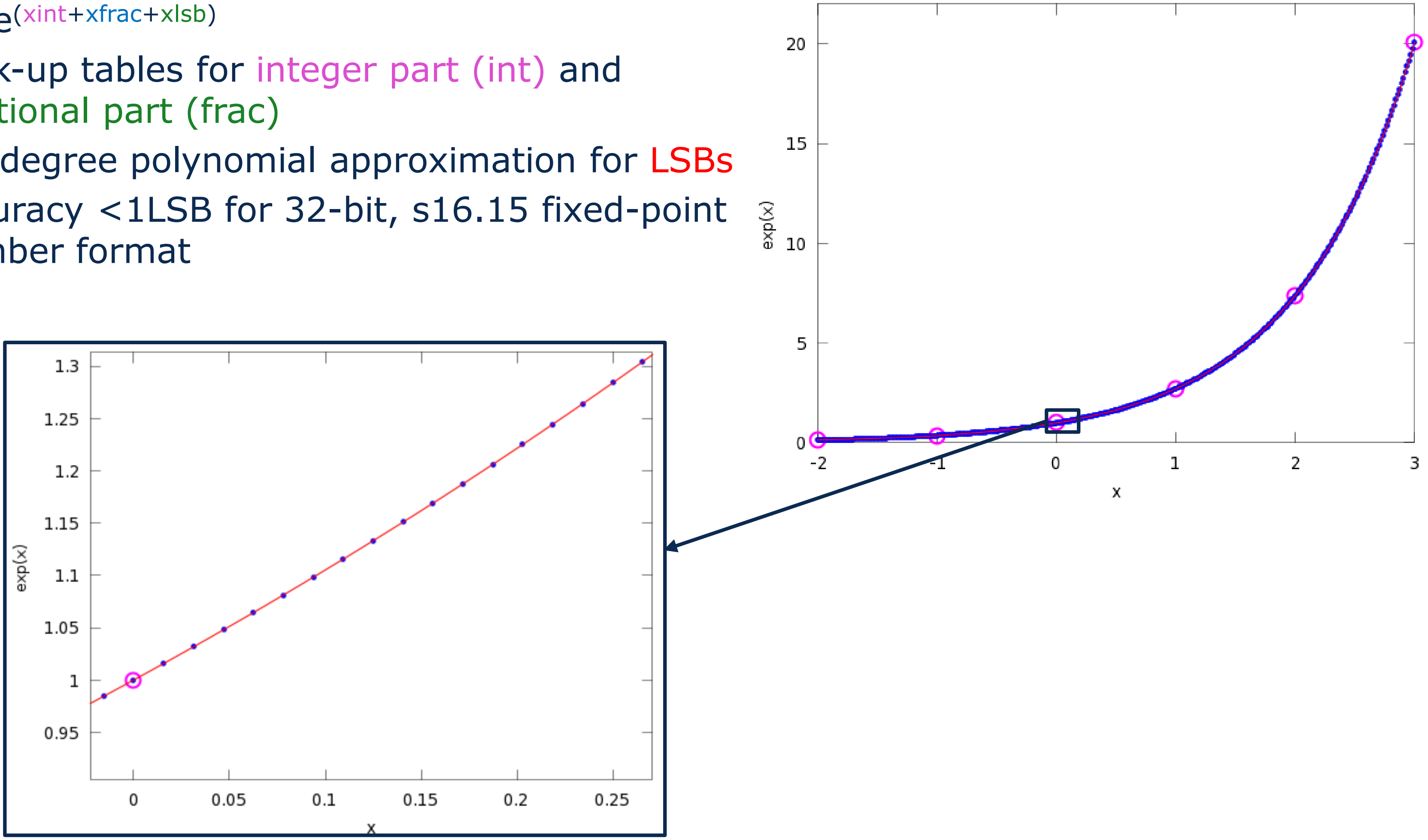




Example: Exponential function for Neuromorphic Hardware

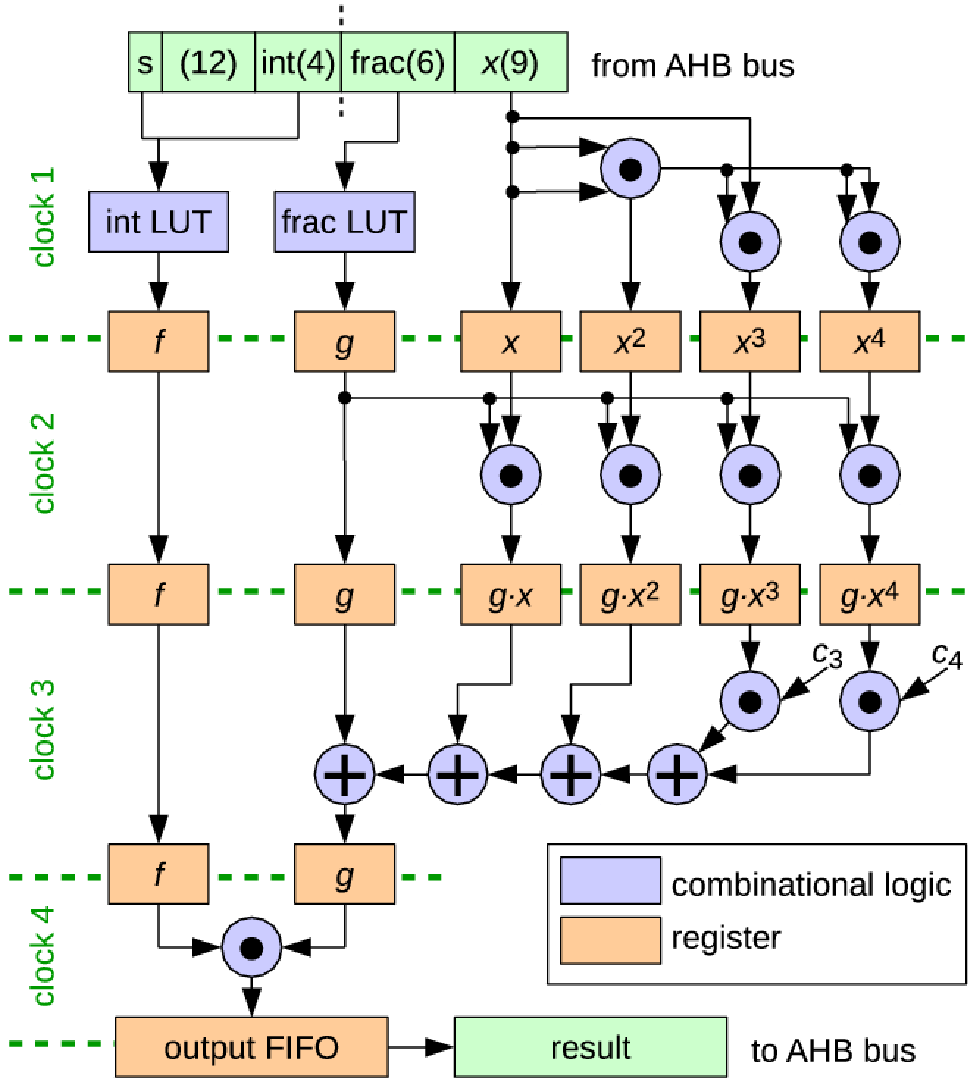
- Dividing the argument of the exponential function
- $e^x = e^{(xint + xfrac + xlsb)}$
- Look-up tables for integer part (int) and fractional part (frac)
- 4th degree polynomial approximation for LSBs
- Accuracy <1LSB for 32-bit, s16.15 fixed-point number format



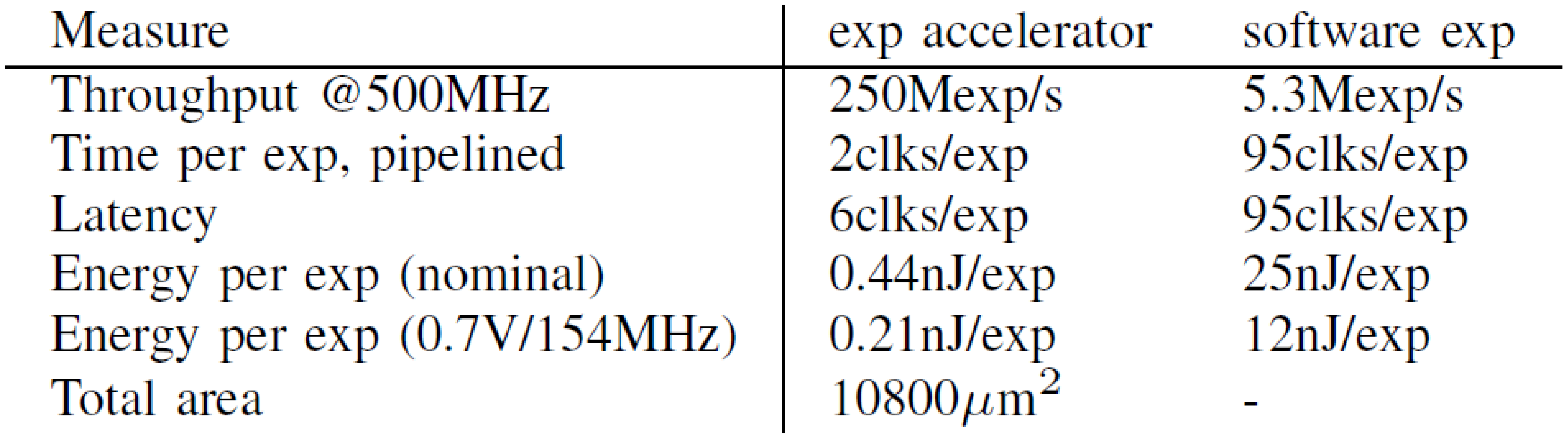


Example: Exponential function for Neuromorphic Hardware

- Pipeline with 4 stages
- FIFO at output (32 values)
- Implementation in 28nm CMOS
- >500MHz clock frequency possible



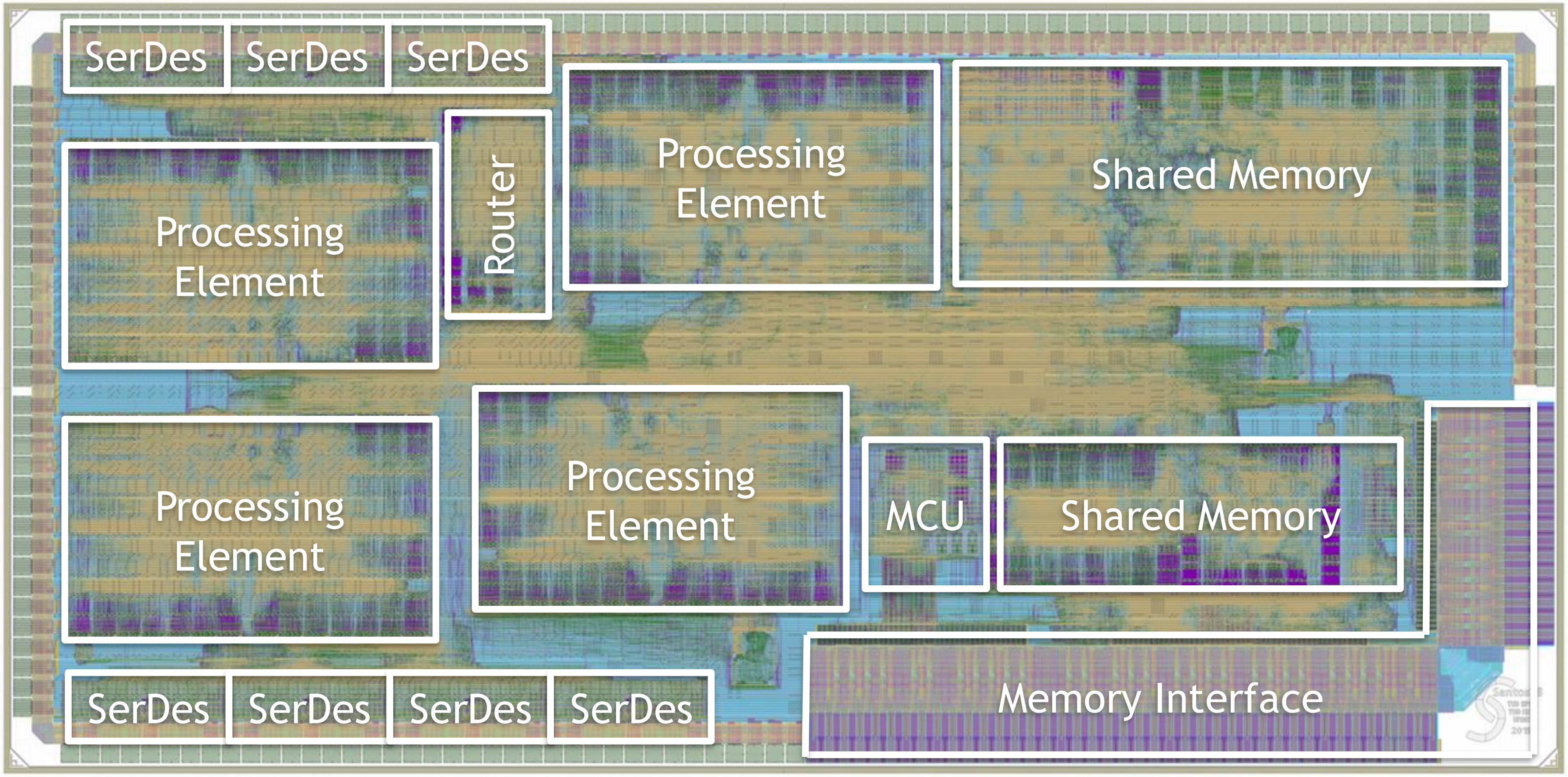
Example: Exponential function for Neuromorphic Hardware



Measure	exp accelerator	software exp
Throughput @500MHz	250Mexp/s	5.3Mexp/s
Time per exp, pipelined	2clks/exp	95clks/exp
Latency	6clks/exp	95clks/exp
Energy per exp (nominal)	0.44nJ/exp	25nJ/exp
Energy per exp (0.7V/154MHz)	0.21nJ/exp	12nJ/exp
Total area	$10800\mu m^2$	-

J. Partzsch et al., "A fixed point exponential function accelerator for a neuromorphic many-core system," 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, 2017, pp. 1-4. doi: 10.1109/ISCAS.2017.8050528

Example: Exponential function for Neuromorphic Hardware

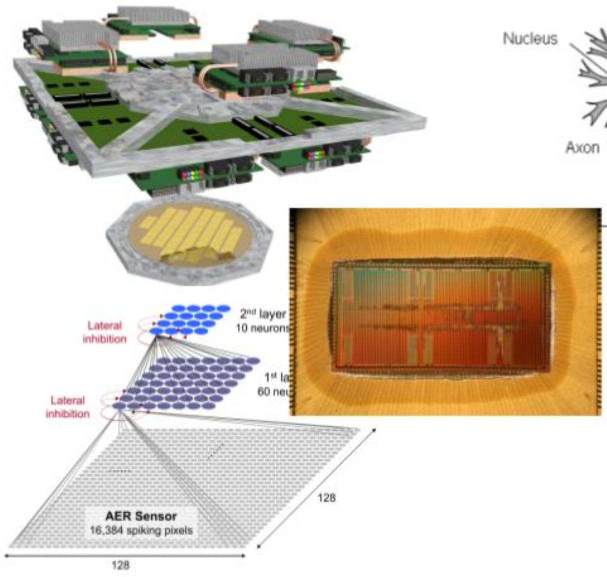


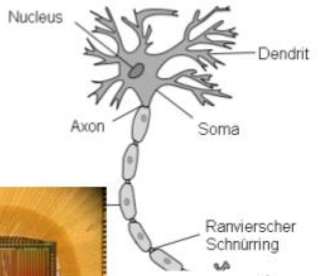
- Test chip: Santos
- 28nm SLP CMOS, 18mm²
- Component test chip for Neuromorphic Supercomputer
- 4 processing elements, memory interface, fast serial I/Os

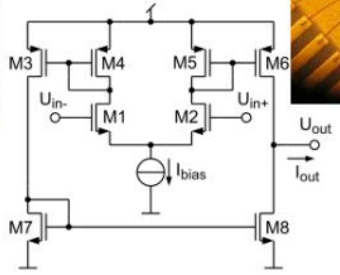
Modul Neuromorphic VLSI-Systems

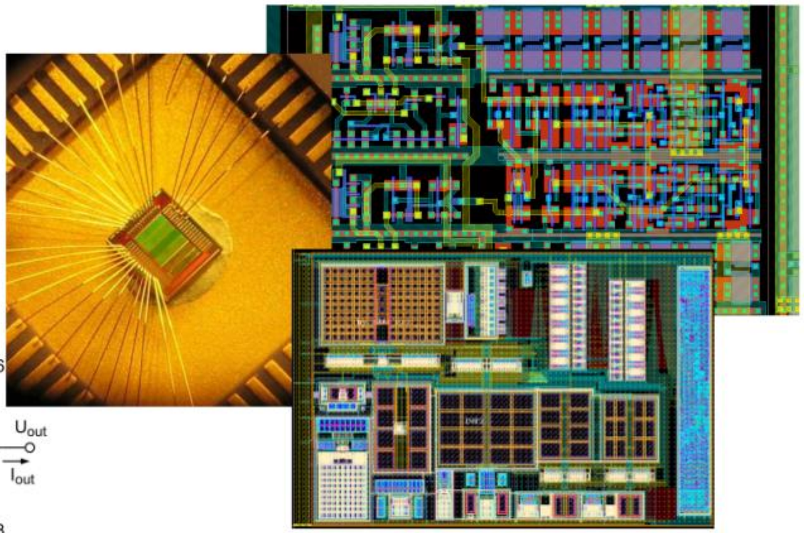
Neuromorphe Systeme

Analoger CMOS-Schaltungsentwurf









- Grundlagen zu neuronalen Netzen und ihrer technischen Realisierung
- Integrierte analoge Schaltungen: Entwurf, Simulation, Verifikation
- Praktischer Schaltungsentwurf und Layout mit Cadence

Human Brain Project