Universität Bremen

HW/SW Codesign

Concepts

Prof. Dr. Rolf Drechsler
Dr. Muhammad Hassan
M.Sc. Jan Zielasko
M.Sc. Milan Funck

Announcements

- Exam schedule will be shared by Tutors

Sr.no	Von	Bis	03.02.2025	04.02.2025	06.02.2025	10.02.2025	11.02.2025	17.02.2025	18.02.2025	24.02.2025	26.02.2025
1	09:30	10:00									
2	10:00	10:30									
3	10:30	11:00									
4	11:00	11:30									
5	12:00	12:30									
6	12:30	13:00									
7	13:00	13:30									
8	13:30	14:00									

Universität Bremen

So far ….

- External architecture (aspects of a computer system that are visible to a programmer)
  - View of the programmer
    - perspective that software developers have of the computing system
    - set of instructions, addressing modes, registers, data types, and the memory architecture
  - Instruction Set Architecture (ISA)
    - native data types, instructions, registers, addressing modes, memory architecture, interrupt and exception handling, and external I/O

Universität Bremen

So far ….

- Internal architecture (organization of a computer's hardware)
  - Structure of a computer
    - physical components of a computer, such as the CPU, memory (RAM, ROM, cache), input/output devices, and storage devices.
  - Microarchitecture
    - data paths, data processing and storage elements
    - Pipeline

- The external architecture and the programmer's view are more concerned with the interface and usability
- The internal architecture, microarchitecture, and pipeline are focused on the operational efficiency and the physical implementation of the system's capabilities.

So far ….

How do I design a computer or a system for a given task?

Here: Holistic view

System = Hardware + Software

Hardware/Software Codesign

An integrated approach to designing **embedded systems** where the hardware and software components are developed concurrently and in a coordinated manner

- application specific processing system embedded in bigger technical context
- designed to perform a specific set of functions, e.g., automobiles, appliances, or mobile phones
- consists of cooperating optimized HW/SW components

This approach helps optimize
- Performance
- Cost
- power efficiency

This approach reduce
- development time
- complexity

Universität Bremen

Today's Agenda

- Why is HW/SW Codesign important?
- Design Methodology
- Specification and Modeling
  - Graph Modeling
    - Control Flow Graph
    - Data Flow Graph
    - Control Data Flow Graph
- Model Characteristics
  - Concurrency
  - Hierarchy
  - Communication
    - Shared Memory
    - Message Passing
  - Synchronization


László Gaál // testing Veo2

Nvidia Jetson Orin Nano vs HGX

Specifications

Jetson Orin Nano Super Developer Kit

AI Performance	67 INT8 TOPS
GPU	NVIDIA Ampere architecture with 1024 CUDA cores and 32 tensor cores
CPU	6-core Arm® Cortex®-A78AE v8.2 64-bit CPU 1.5MB L2 + 4MB L3
Memory	8GB 128-bit LPDDR5 102 GB/s
Storage	Supports SD card slot and external NVMe
Power	7W–25W

	HGX H100	
	4-GPU	**8-GPU**
Form Factor	4x NVIDIA H100 SXM	8x NVIDIA H100 SXM
FP8 Tensor Core*	16 PFLOPS	32 PFLOPS
INT8 Tensor Core*	16 POPS	32 POPS
FP16/BF16 Tensor Core*	8 PFLOPS	16 PFLOPS
TF32 Tensor Core*	4 PFLOPS	8 PFLOPS

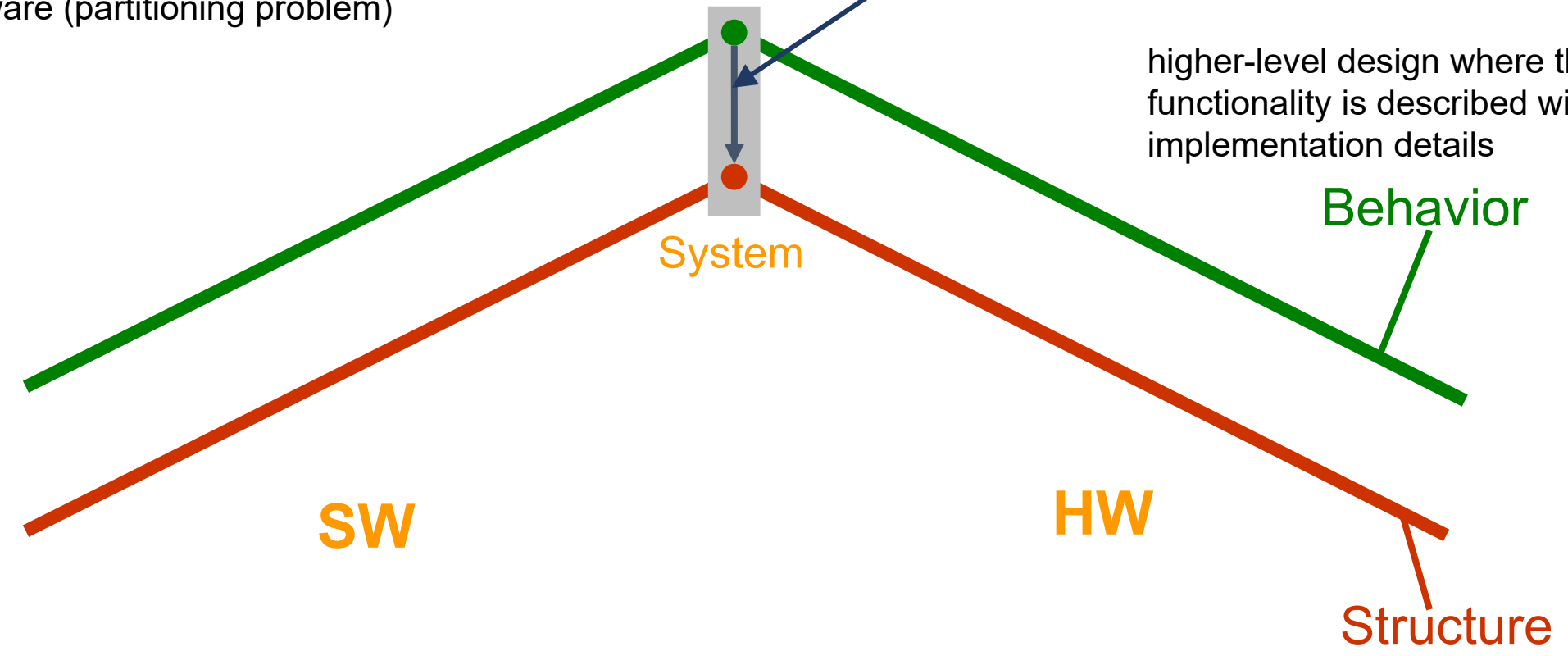
TOPS: Trillion Operations per Second

https://nvdam.widen.net/s/zkfqjmtds2/jetson-orin-datasheet-nano-developer-kit-3575392-r2
https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/nano-super-developer-kit/

Abstraction Levels

the overall system's behavior is defined and decisions are made about which components should be implemented in hardware and which in software (partitioning problem)

ensures that both software and hardware are integrated and optimized together.

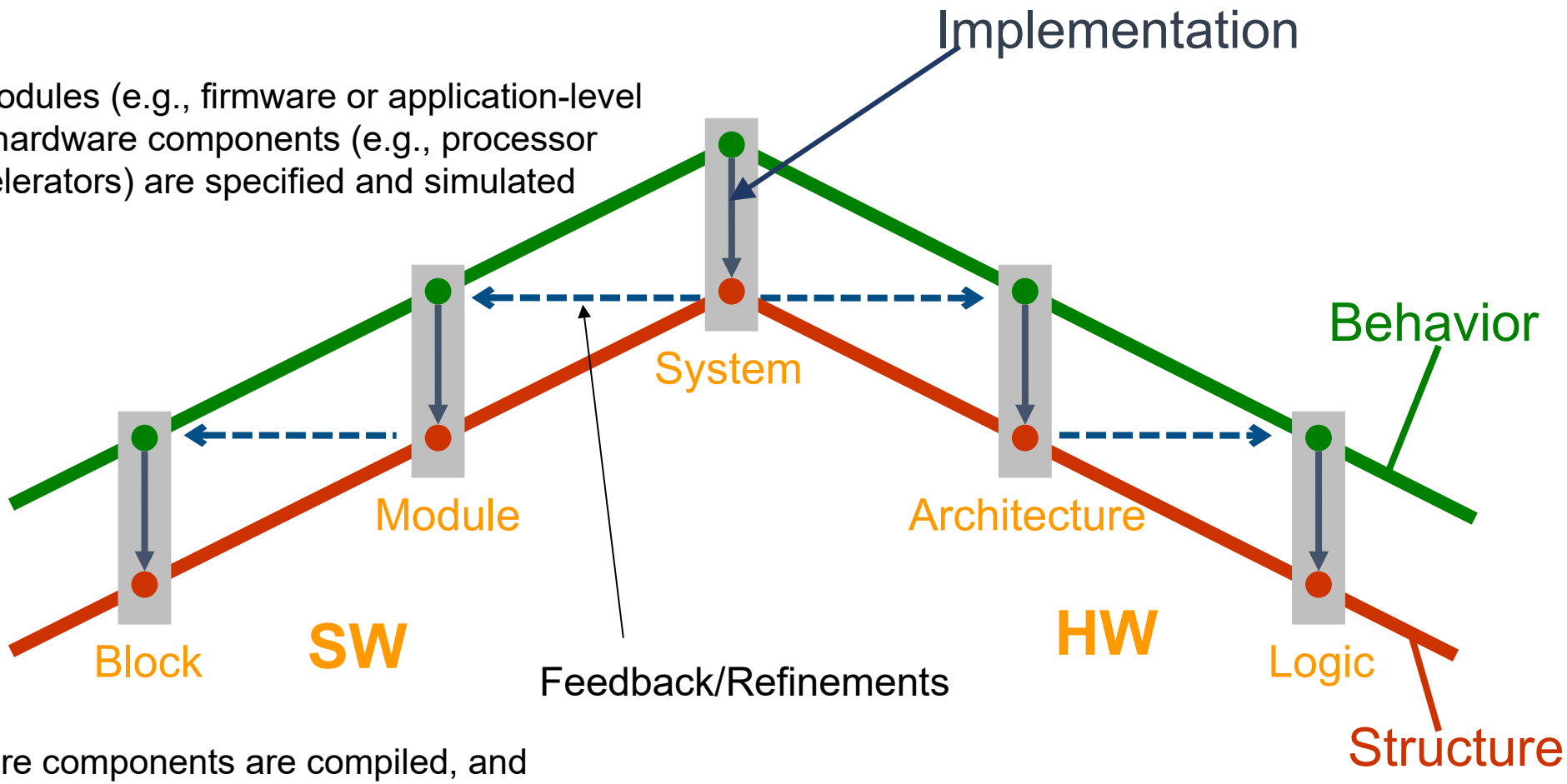


higher-level design where the system's functionality is described without going into implementation details

lower-level design where the physical realization of the system is described

Abstraction Levels

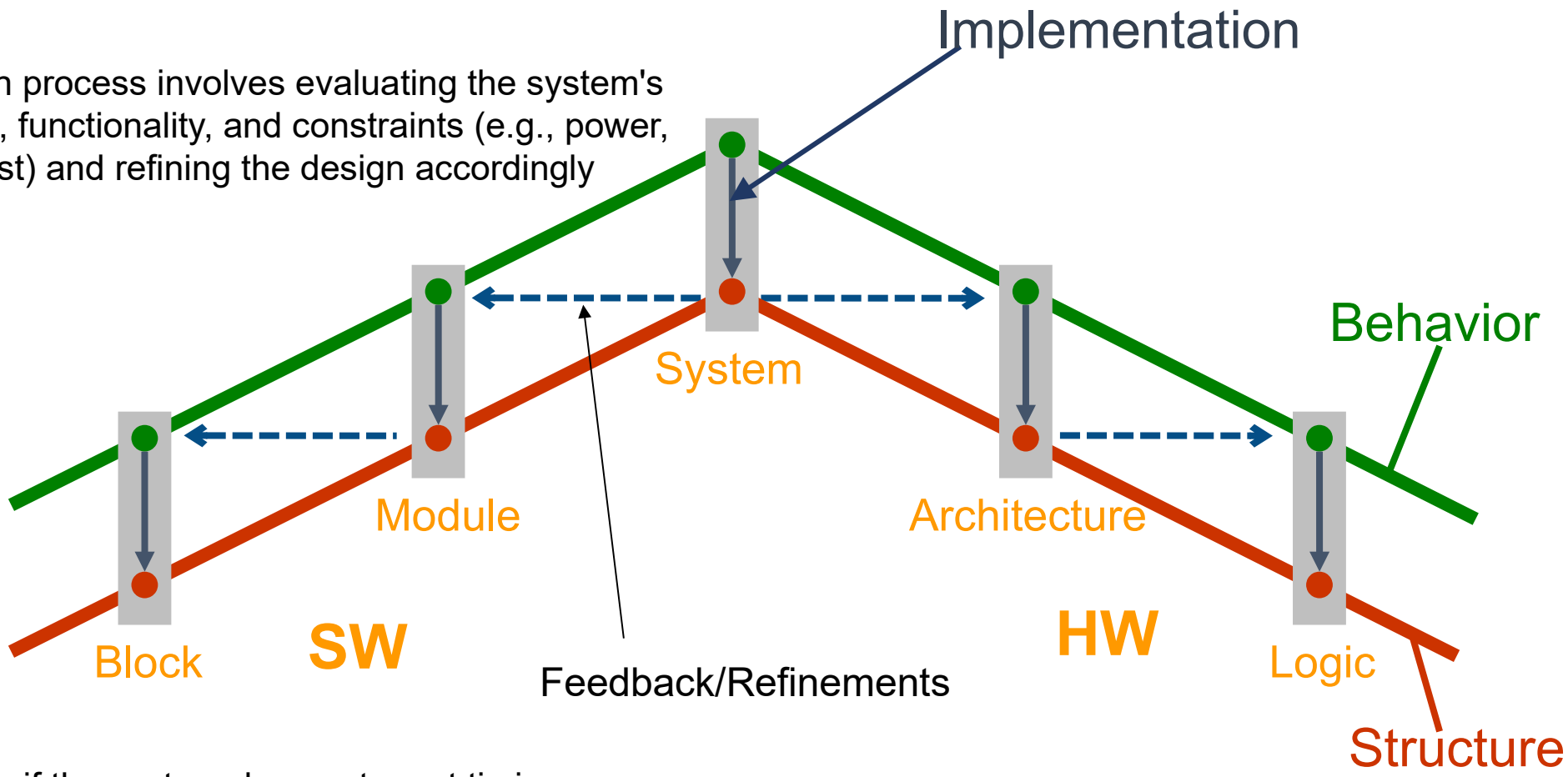
software modules (e.g., firmware or application-level code) and hardware components (e.g., processor cores, accelerators) are specified and simulated



Software components are compiled, and hardware components are synthesized (logic, gates, etc).

Abstraction Levels

the co-design process involves evaluating the system's performance, functionality, and constraints (e.g., power, area, and cost) and refining the design accordingly



For example, if the system does not meet timing constraints, adjustments may involve offloading more tasks to hardware or optimizing software algorithms.

Introduction HW/SW Codesign

- **Is about:**
  - specification & modelling of mixed HW/SW-Solutions at high abstraction levels
    - defining system functionality and behavior without getting into the details of implementation
  - Partitioning, scheduling & cost estimation
    - which parts of the system functionality are best implemented in hardware and which in software
    - determines the execution order of tasks, especially for those tasks that share the same hardware or software resources
  - with holistic HW/SW-component consideration
    - consider both hardware and software components as an integrated whole
  - design quality (cost reduction, time-to market)
    - reducing costs and shortening the time-to-market
  - optimized performance (low latency, high system throughput)

Requirements for HW/SW Systems

- **RAS** (Reliability, Availability, Serviceability):

  when R(t)=exp(-λt)

  $R(t) = MTTF(system) = \sum MTTF(subsystems) = \sum (1/failiure_{rate})$

  A(t) = MTTF/(MTTF + MTTR)

  $S(t) = MTTR$

- **Efficiency:** Cost, energy, execution time, area

Reliability : probability that a system will perform its required functions without failure over a specified period of time

Availability: probability that a system is operational and functional at a given time

Serviceability: reflecting the system's maintainability or how quickly a system can be restored to operating condition after a failure

In summary, RAS characteristics ensure that HW/SW systems are dependable and maintainable, while efficiency metrics focus on optimizing cost, energy consumption,

MTTF: Mean Time to Failure
MTTR: Mean Time To Repair
λ: The failure rate

Requirements for HW/SW Systems

- **Real-time capability:** system reacts to external stimuli from environment in defined time
  - Soft real-time: Information worthless after deadline (e.g. sound processor)
  - Hard real-time: Non-compliance may lead to system failure (e.g. rocket altitude control)
- **Flexibility** (freely programmable CPU resources)**:**
  Risk minimization, Time-to market, Post-shipment upgrades

Requirements …

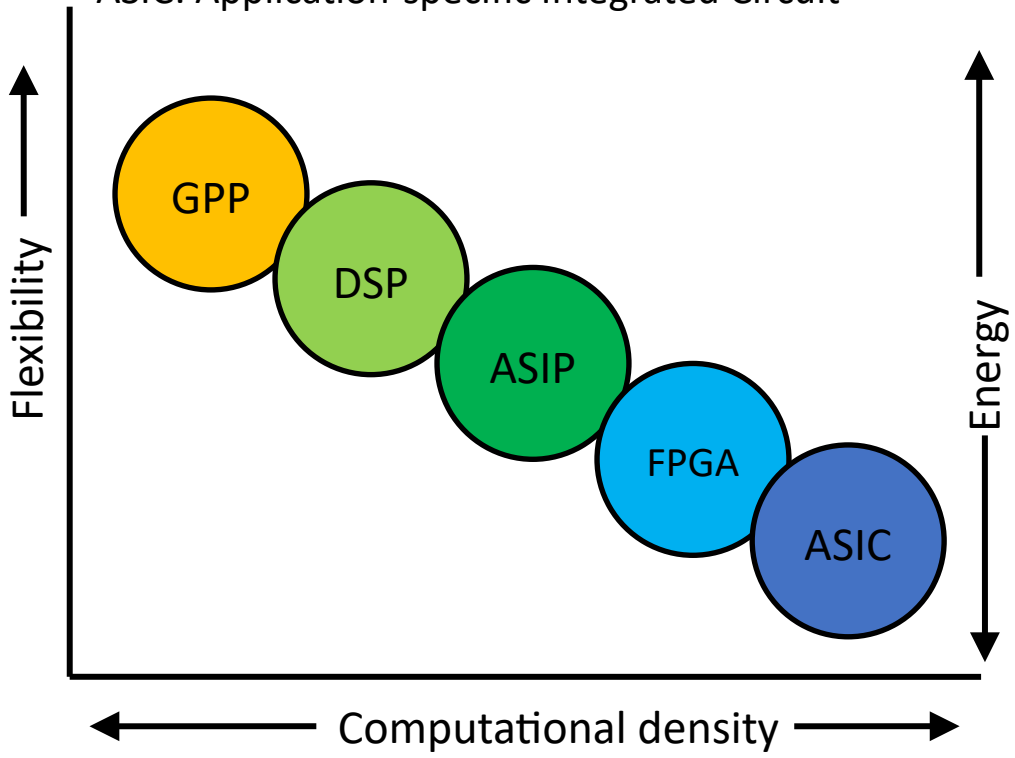
- **Computational density:** Compute operations per area and time
- **Functional diversity:** number of operations which can be changed instantaneously of compute entity

- **Moore's Law:** doubling of chip capacity every 2-3 years, how to deal with design gap?
  - the challenge that as the number of transistors on a chip increases, it becomes progressively more difficult for engineers to effectively utilize the increased complexity within the design cycle time

- **Design Productivity Improvements by raised levels of abstraction:**

  Polygons mask layout → Transistor circuity → Logic gates (standard cells) → RTL (Register Transfer Block, ALUs, Registers…) design → HW-description languages and behavioural synthesis

GPP: General Purpose Processor
DSP: Digital Signal Processor
ASIP: Application-specific Instruction Set Processor
FPGA: Field Programmable Gate Array
ASIC: Application-specific Integrated Circuit



Requirements …

- **Platform based SOC Design:** Conquer design complexity by reuse maximization
  - Shorter development cycles & higher chances for (first time) fault-free design.
  - Standard on-Chip buses/interfaces, CPU's, SW-development environments
- **Abstraction Levels**

Level	Hardware	Software
System	Network of communicating sub-systems/tasks/processes which model the desired application or system functionality	
Architecture/Module	Processors, ASIC, Memory, Buses etc	Interacting SW modules, processes
RTL/Block	Counter, Comparator, ALUs …	Iterative loops, program sequences
Logic/ Expression	Logic gates, Flip-flops …	Assignments, branches, arithmetic, logic operators ..
Device/Instruction	MOSFET transistors …	Machine code instructions

Design Methodology (1)- Terminologies

- **System design:**
  - process to implement a desired function with a given set of physical components;
  - determining the architecture, components, modules, interfaces, and data
- **Appropriate design process:**
  - Improves quality of the product,
  - Reduces cost and development time (time-to-market)

the system design process is foundational to the successful delivery of new products and systems

Design Methodology (2)- Terminologies

- **Design Flow:**
  - A sequence of steps that engineers follow when designing integrated circuits, software, or complex hardware systems
  - has proven practical value
  - identifies design faults during early phases of design (at high abstraction levels)
  - Avoid time consuming and costly iterations across multiple abstraction levels
  - Top-Down-Design

a top-down design methodology within a proven design flow framework is essential for creating complex systems efficiently and effectively.

Design Methodology (3)- Terminologies

- **Specification:**
  - Description of system behavior with formal executable models
- **Exploration:**
  - Comparison of alternative realizations with respect to cost, performance, robustness, …
- **Refinement:**
  - Synthesis of a structural system representation out of the functional specification

Design Methodology (4)- Terminologies

- **Design space exploration:**
  - roots on efficient estimation and simulation techniques which allow design characteristic evaluation prior to costly realization / implementation

- **Design at High Layers of Abstraction:**
  - Higher efficiency in design representation
    - few lines of HDL code represent 1000s of logic gates
  - Oversees a much bigger implementation space

```
#include <stdio.h>
int main()
{
    printf("Hello, %s\n“, "world");
    return 0;
}
```

.text
  .align 2
  .globl **main**
	Directive: Enter text section
	Directive: Align code to 2^2 bytes
	Directive: Declare global symbol **main**

main:
  addi sp,sp,-4
  sw ra,0(sp)
  la a0, str1
  la a1, str2
  call printf
  lw ra,0(sp)
  addi sp,sp,4
  li a0,0
  Ret
	Label for start of **main**
	Allocate stack frame
	save return address
	Pseudoinstructions
	load addresses of str1, str2
	Pseudoinstruction: call function printf
	Restore return address,
	deallocate stack frame
	Return with value 0
	---- 	

.section .rodata .balign 4	Directive: Enter read-only data section Directive: Align data section to 4 bytes

str1:
  .string "Hello, %s!\n"
str2:
  .string "world"
	Label for str1
	Directive: null-terminated string
	Label for str2
	Directive: null-terminated string

Design Verification

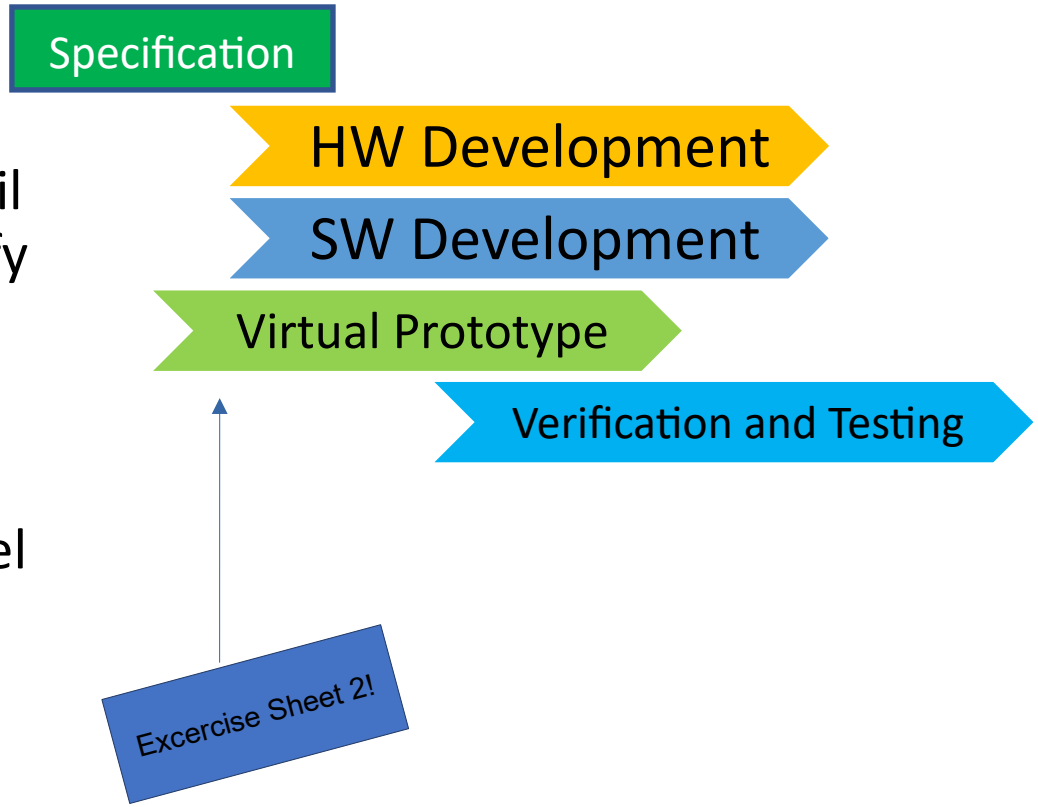
- **Design Verification by Simulation:** Simulation can't achieve exhaustive coverage of input combinations: 32-bit ALU has $2^{32} \times 2^{32} = 2^{64}$ input combinations, but is meaningful to reasonable subset of input combinations;
  - Typical input patterns obtain confidence in design but cannot prove correctness nor completeness
- Simulation Acceleration
  - Divide and Conquer: Parallel simulation of system blocks,
  - Mixed-Level Simulation: Simulate components at different level of detail
  - Reduction of simulated real-time:
    1s real time can simulate for eternity (seed config with saturated states);

Today's Agenda

- Why is HW/SW Codesign important?
- Design Methodology
- Specification and Modeling
  - Graph Modeling
    - Control Flow Graph
    - Data Flow Graph
    - Control Data Flow Graph
- Model Characteristics
  - Concurrency
  - Hierarchy
  - Communication
    - Shared Memory
    - Message Passing
  - Synchronization

Specification and Modeling

- **Specification:** defines supported functionality of system -> model is useful
- **Models:** describe how a system functions; Characteristics: Formal (complete/partial) description of a system, without unnecessary detail (abstraction), Understandable and simple to modify
- **Architecture:** describes how the system is implemented
- **Virtual Prototypes:** allow for the HW and SW components of a system to be developed in parallel (instead of sequential) by an ISA compatible HW-model



Universität Bremen

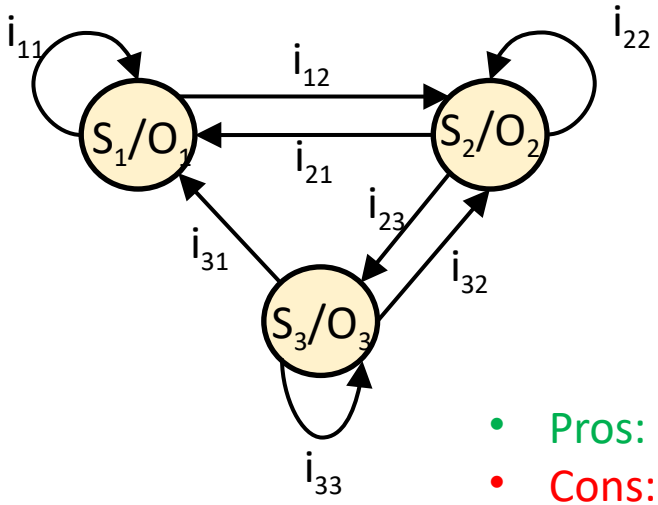
Model Classification

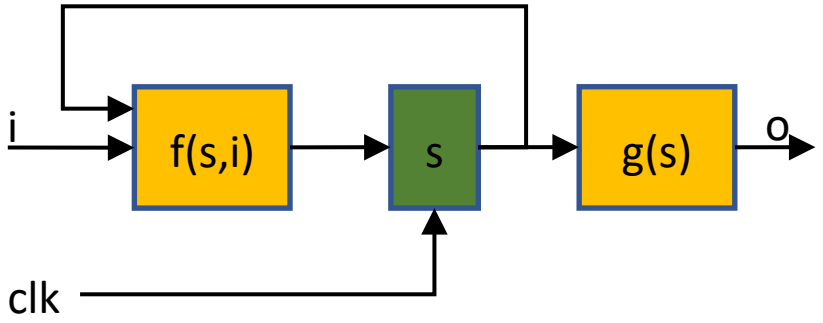
Classification	Model of Computation (MoC) Example
State oriented	Communicating FSMs
	Classical FSMs
Activity oriented	Asynchronous message passing (Kahn process networks)
	Synchronous message passing
Structure oriented	Component connection diagram
Time oriented	Discrete-time event model
	Continuous-time event model (differential equations)

Graph Models

- **State oriented** - states (vertices) connected by state transitions (edges), triggered by external events; **best suited for describing control units (real-time controllers, timing-latency important)**

  Moore state machine



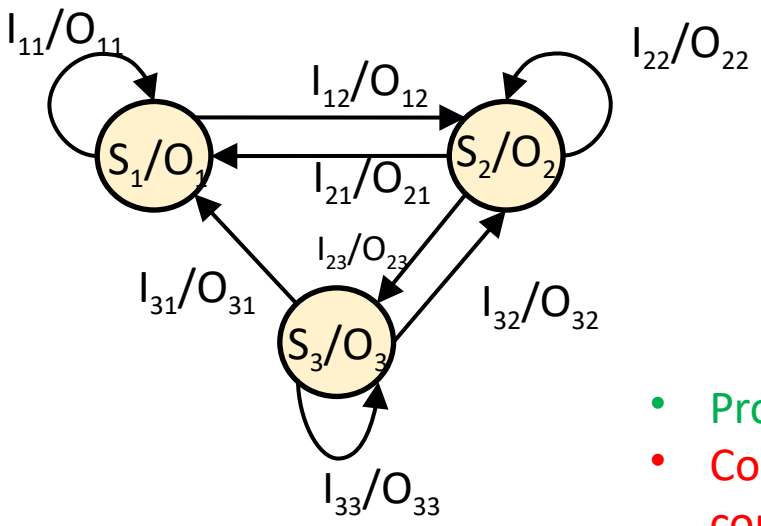


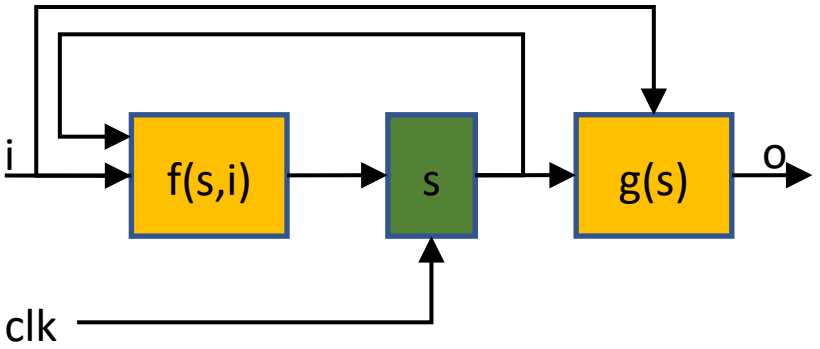
- Pros: No combinatorial path (limits logic depth),use identical design style
- Cons: Large number of states

Graph Models

- **State oriented** - states (vertices) connected by state transitions (edges), triggered by external events; **best suited for describing control units (real-time controllers, timing-latency important)**

  Mealy state machine



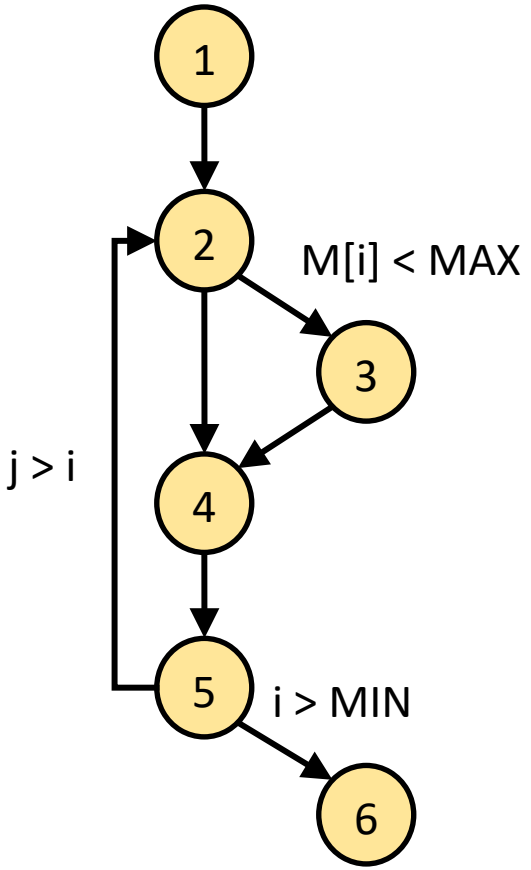


- Pros: Fewer states, clear layout; Most general FSM
- Cons: Long combinatorial paths when multiple FSMs are concatenated; output depend on current state and input
- Avoid whenever possible

Control Flow Graph (CFG)

- A directed, possibly cyclic graph
  - Vertices represent code without jumps
  - Edges represent jumps in the control flow
- Transitions in a CFG are triggered solely by the completion of the preceding block
- Only a single branch is taken to transition from one block to the next (unique!)

```
…
do {
  If (M[i] < MAX)
     ….
  …
  If (i > MIN)
     …
} while ( j> i)
```

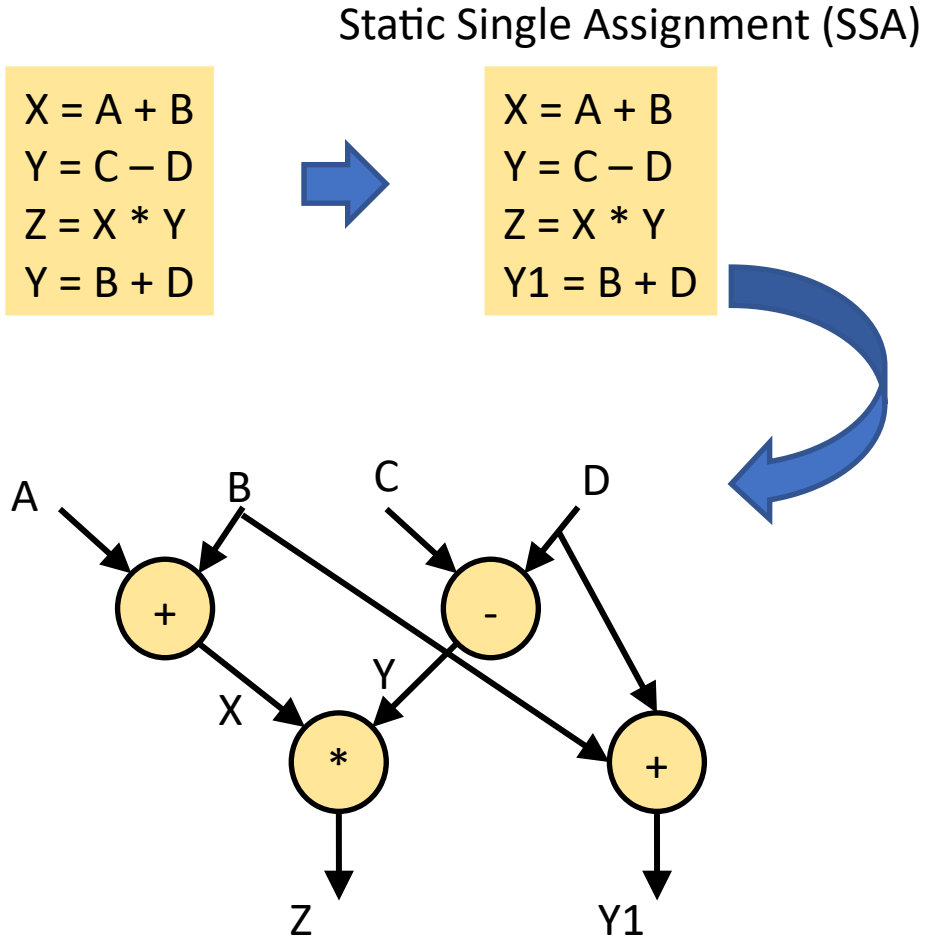


Graph Models

- Activity oriented
  - Describe a system as a set of actions which resolve dependencies.
  - Best suited for transformational systems
    - Digital signal processing - data passed through a transfer function at a fixed rate.

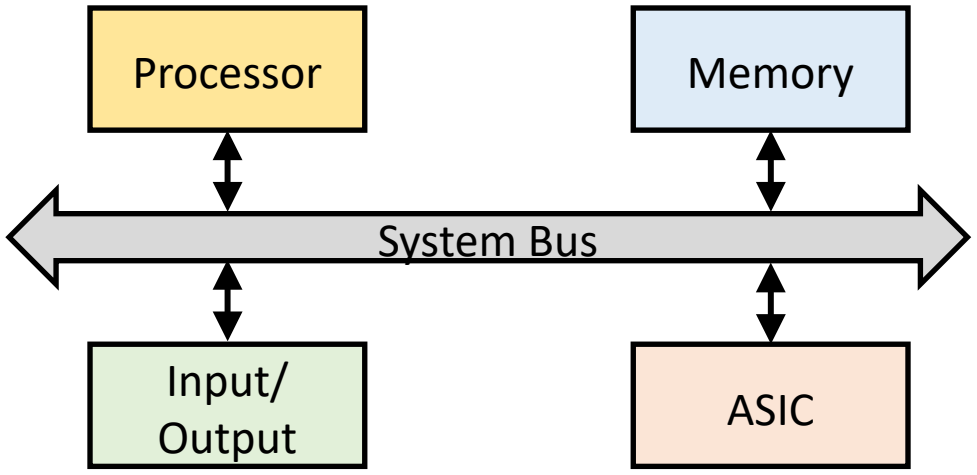
Data Flow Graph (DFG)

- Describe the data dependencies between a number of operations
  - A directed, acyclic graph
    - Vertices = operations
    - Edges = data flow
    - multiple-edges being traversed possible (unique)
- DFG's calculations are triggered by availability of data
- Cannot portray branches in code, but can depict parallelization

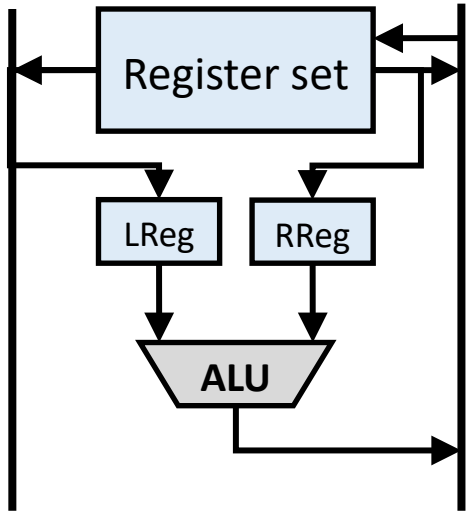


Structure and Data Oriented Graph Models

- **Structure Oriented Model:** describe a system as a set of physical components and their interconnects.
  - Used to depict the physical configuration of a system.



At system/architecture level



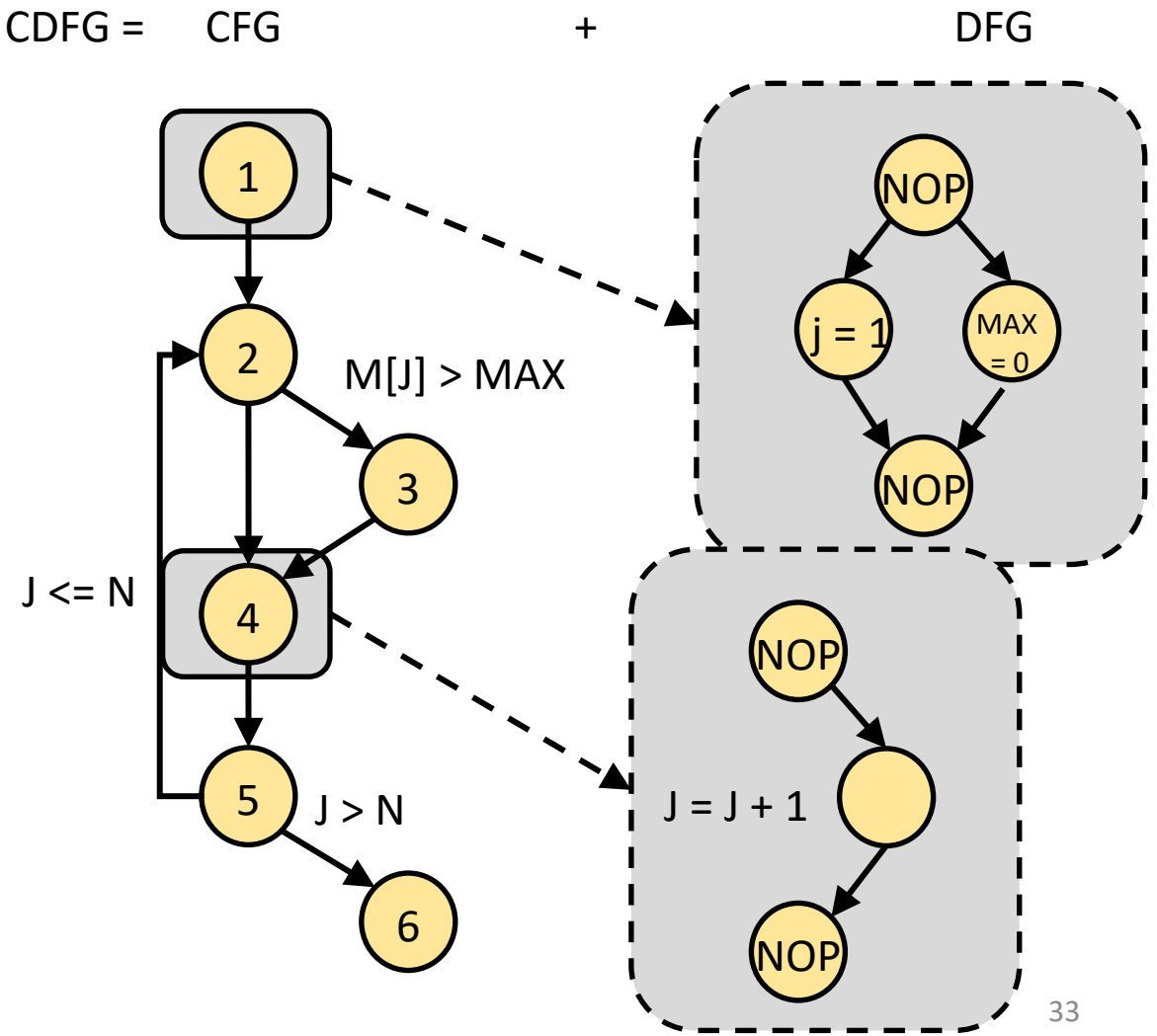
At RT-level

Structure and Data Oriented Graph Models

- **Data Oriented Model:** describe a system as a hierarchy of data structures.
  - The structure of data (e.g., relationships, dependencies, or flow between data elements) is the primary concern.
  - Best suited for describing systems in which the structural representation of data is more important than the system's functionality (e.g. network protocols)
- **Combined Models:** merges benefits of simpler models, allows complete description of a complex system.
  - Best for systems that span a large design domain, e.g. real-time systems or ASICs.

Control Data Flow Graph (CDFG)

- Simultaneous description of the control-structure (e.g. branches) and data dependencies
- CFG: State machine representing the sequential control flow
  - The operations contained within a block (vertex) are expanded in form of a DFG
- DFG: NOP operations provide a uniform entry and exit point for each block



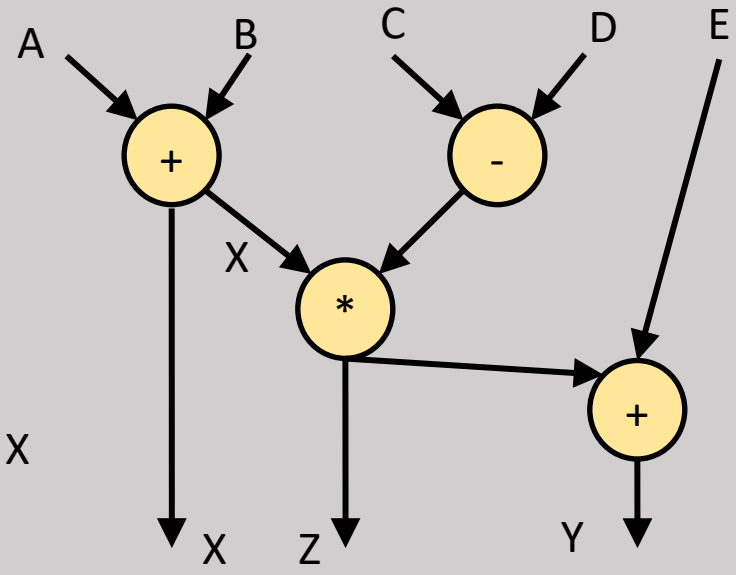
Today's Agenda

- Why is HW/SW Codesign important?
- Design Methodology
- Specification and Modeling
  - Graph Modeling
    - Control Flow Graph
    - Data Flow Graph
    - Control Data Flow Graph
- Model Characteristics
  - Concurrency
  - Hierarchy
  - Communication
    - Shared Memory
    - Message Passing
  - Synchronization

Model Characteristics

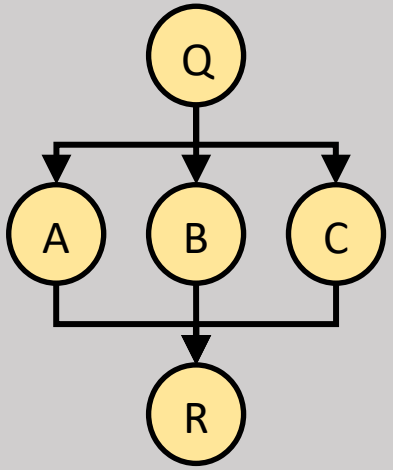
- **Concurrency:** often simpler to split system into concurrent sub-systems: e.g. 2 FSMs with 1 state is simpler than 1 FSM with 2 states.

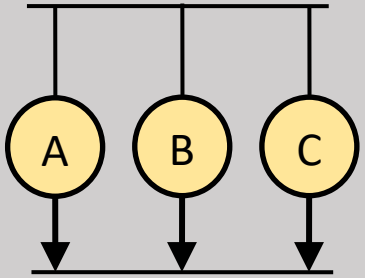
Data oriented concurrency:
No specifc order, single assignment rule



X = A + B
Y = E + Z
Z = (C - D)* X

Control oriented concurrency:
Explicit control instructions (fork-join concurrent behaviour) determine order of operations





Model Characteristics

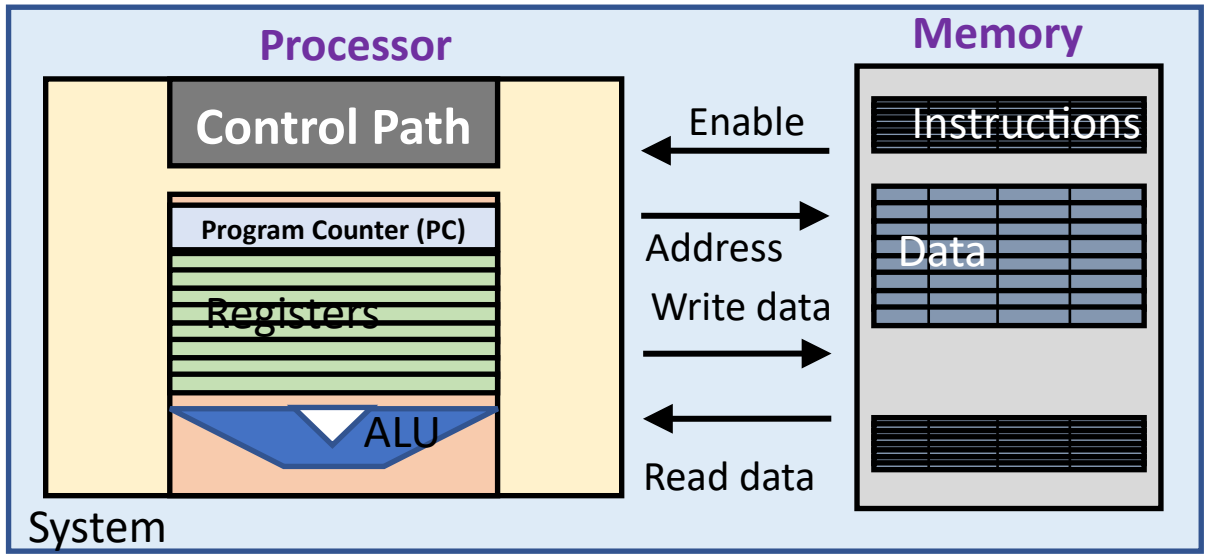
- State Transitions
  - Transitions depend on conditions/states
  - System with N states can have up to N^2 transitions
  - Control centric behavior

- Hierarchy
  - Real systems are too complex to be viewed in entirety
    - Hierarchy splits system into smaller subsystems so developers can focus on their subsystem
    - Allows reuse, not in depth understanding needed

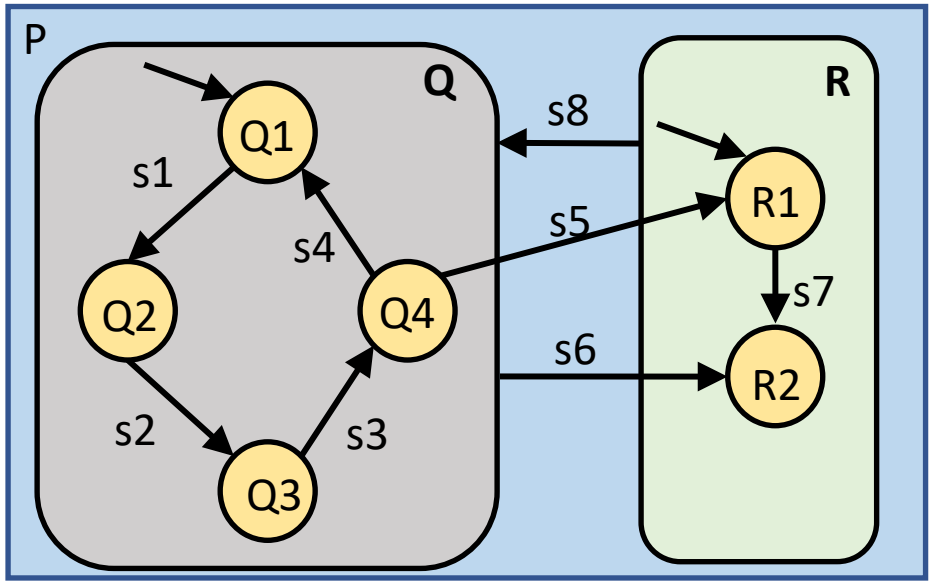
Model Characteristics

- **Hierarchy:** real systems are too complex to be viewed in entirety
  - hierarchy splits system into smaller subsystems so developers can focus on their subsystem (allows reuse, not in depth understanding needed)

Structural hierarchy:
Every component is made up of a sub-structure to lower level of abstraction



Behavioral/ Functional hierarchy
Divides functions into sequential or concurrent sub-functions

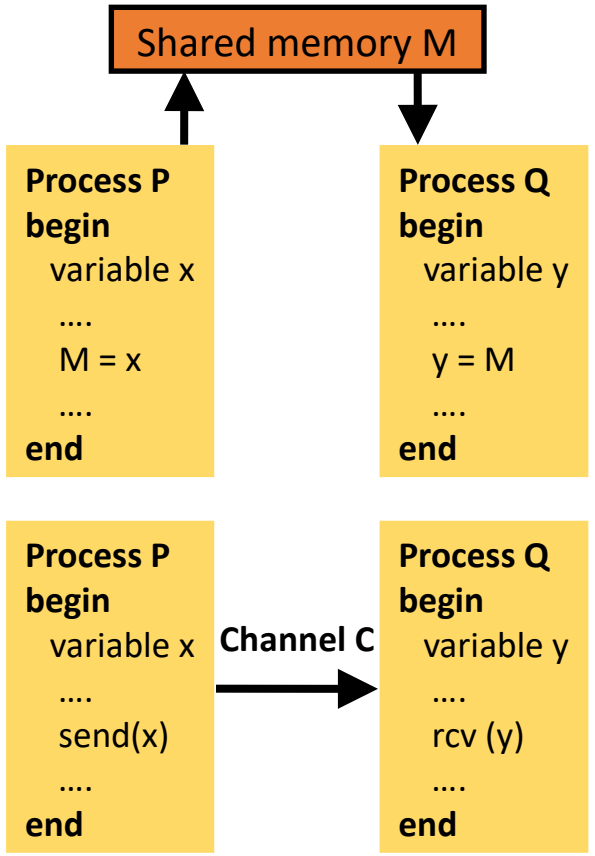


Model Characteristics …

- Program Structures/Constructs
  - Many functions can be described best by sequential algorithms including branches, iterations, subroutines, e.g., sorting algorithm
- Completion
  - Process ability to indicate it has stopped
  - All calculations are made or all variables got assigned their new value
  - Sequential processing
    - ensures that the next step in an algorithm does not begin until the current step is complete
    - coordinate processes by ensuring that dependencies are resolved
- Communication
  - Connect HW/SW subsystems

Model Characteristics …

- Communication – Connect HW/SW subsystems



Shared memory

Sending process writes global variable into shared resource; all receiving processes can now read var; sync must be done separate

Message parsing

- Data between processes is exchanged through communication channels (uni/bidirectional, point-to-point, shared bus)

- Channel can be blocking on non-blocking transfer
  - Blocking-trans: sending process waits until receiving process has accepted data
  - Non-blocking-trans: sending process writes data in queue and continues processing.
    - Receiver can read it at its leisure => standard today, additional memory for queue needed

Memory-Mapped Input/Output (I/O)

- peripheral devices are mapped into the same address space as the program memory.
- In memory-mapped I/O, reads and writes to specific memory addresses are translated into operations on the peripherals
- What operations does software need to perform on peripherals?
  - Get and set parameters
  - Receive and transmit data
  - Enable and disable functions
- How can we imagine providing this interface to software?
  - Specialized CPU instructions
  - Accessing devices like they are memory

Memory-Mapped Input/Output (I/O)

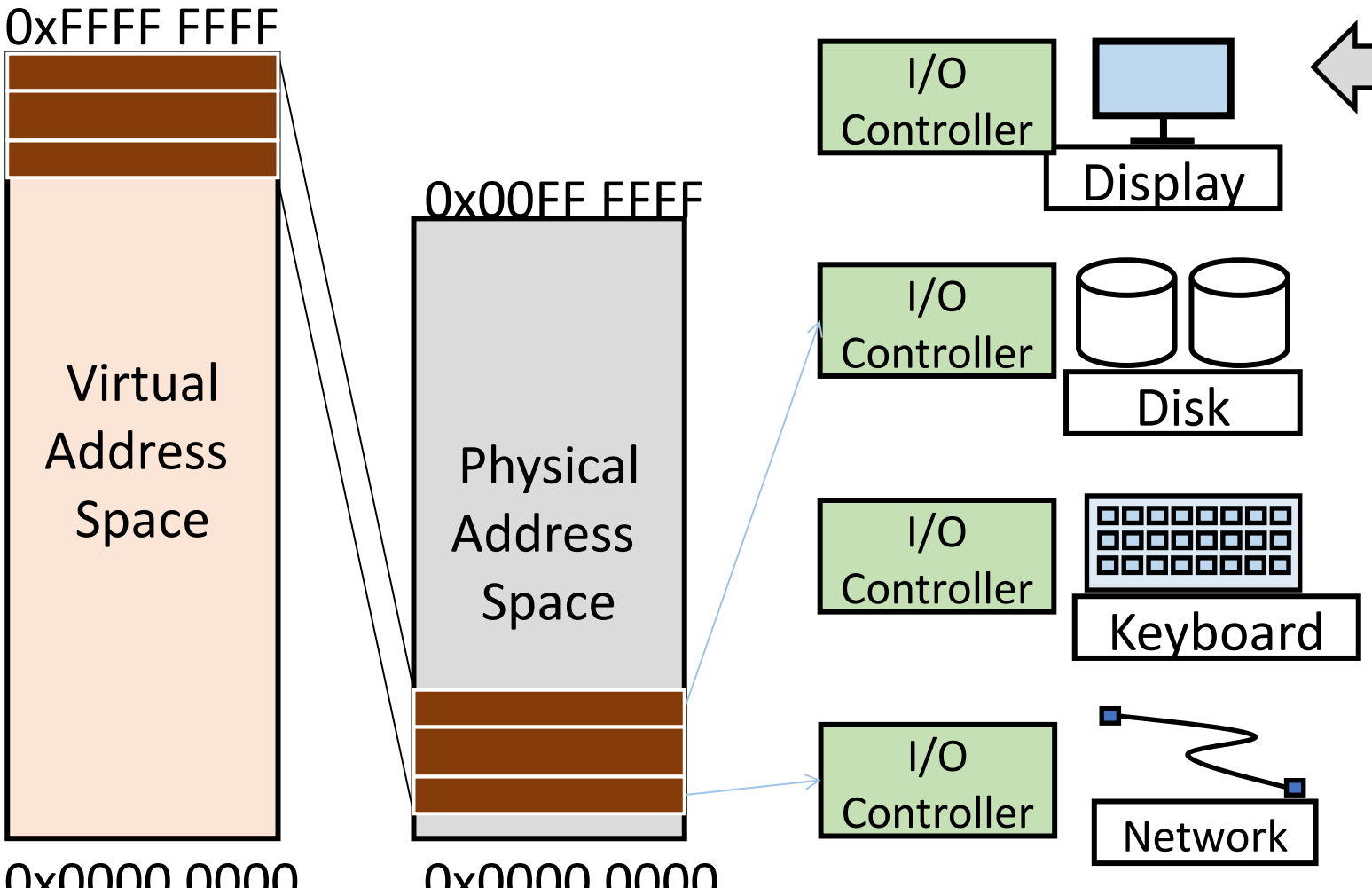
- Access I/O devices (like keyboards, monitors, printers) just like it accesses memory
  - Each I/O device assigned one or more address
  - When that address is detected, data is read from or written to I/O device instead of memory
  - A portion of the address space dedicated to I/O devices (for example, addresses 0xFFFF0000 to 0xFFFFFFFF in reserved segment of memory map)
- But we need additional hardware to help us
  - After all we will not really write to and read from memory

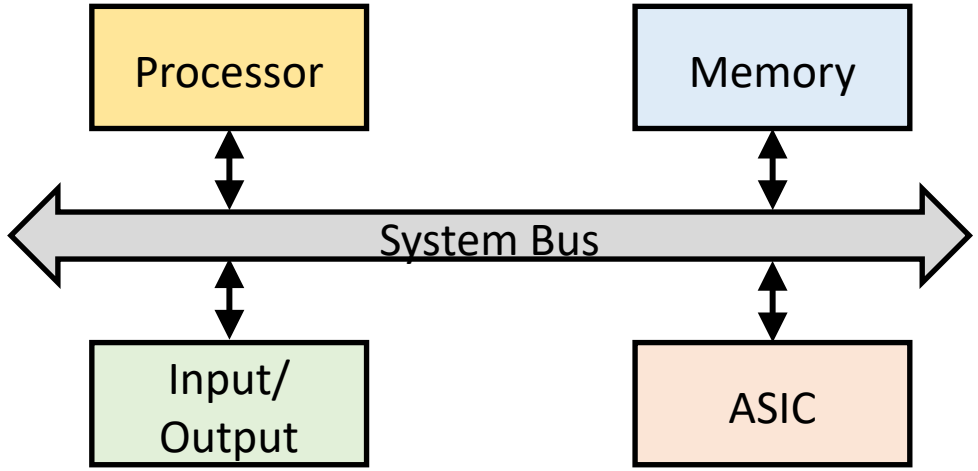
Universität Bremen

Memory-Mapped I/O Hardware

- Address Decoder:
  - Looks at address to determine which device/memory communicates with the processor
- I/O Registers:
  - Hold values written to the I/O devices
- ReadData Multiplexer:
  - Selects between memory and I/O devices as source of data sent to the processor

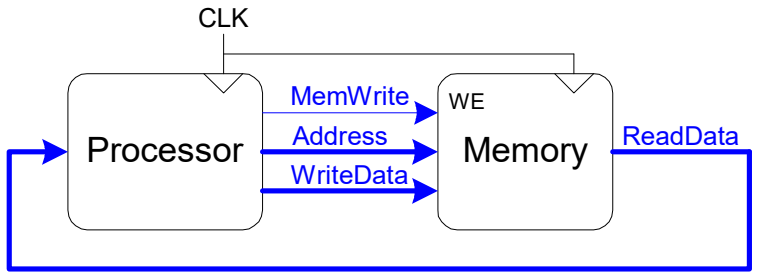
Memory-Mapped I/O

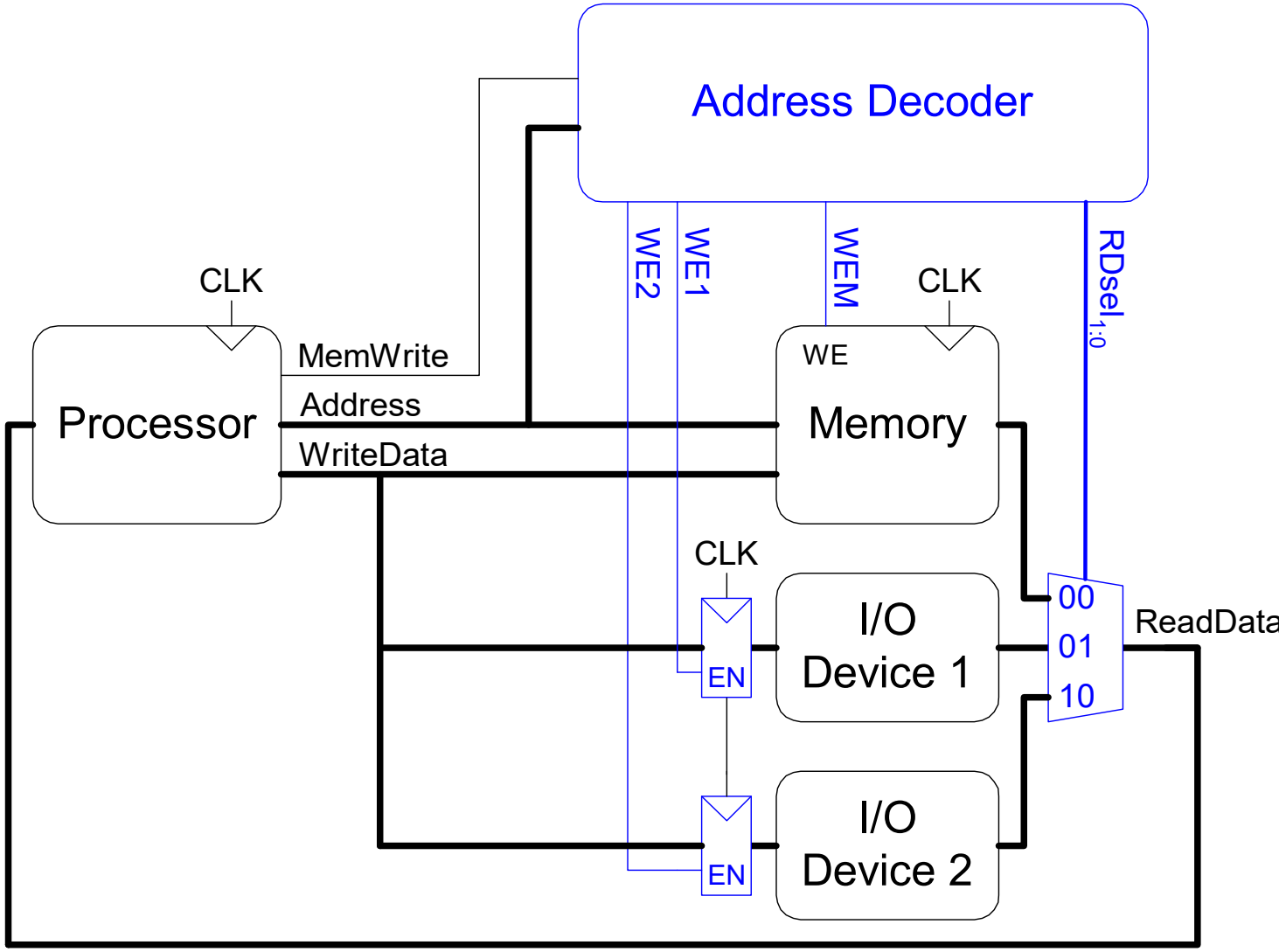




At system/architecture level

Memory-Mapped I/O Hardware





Universität Bremen

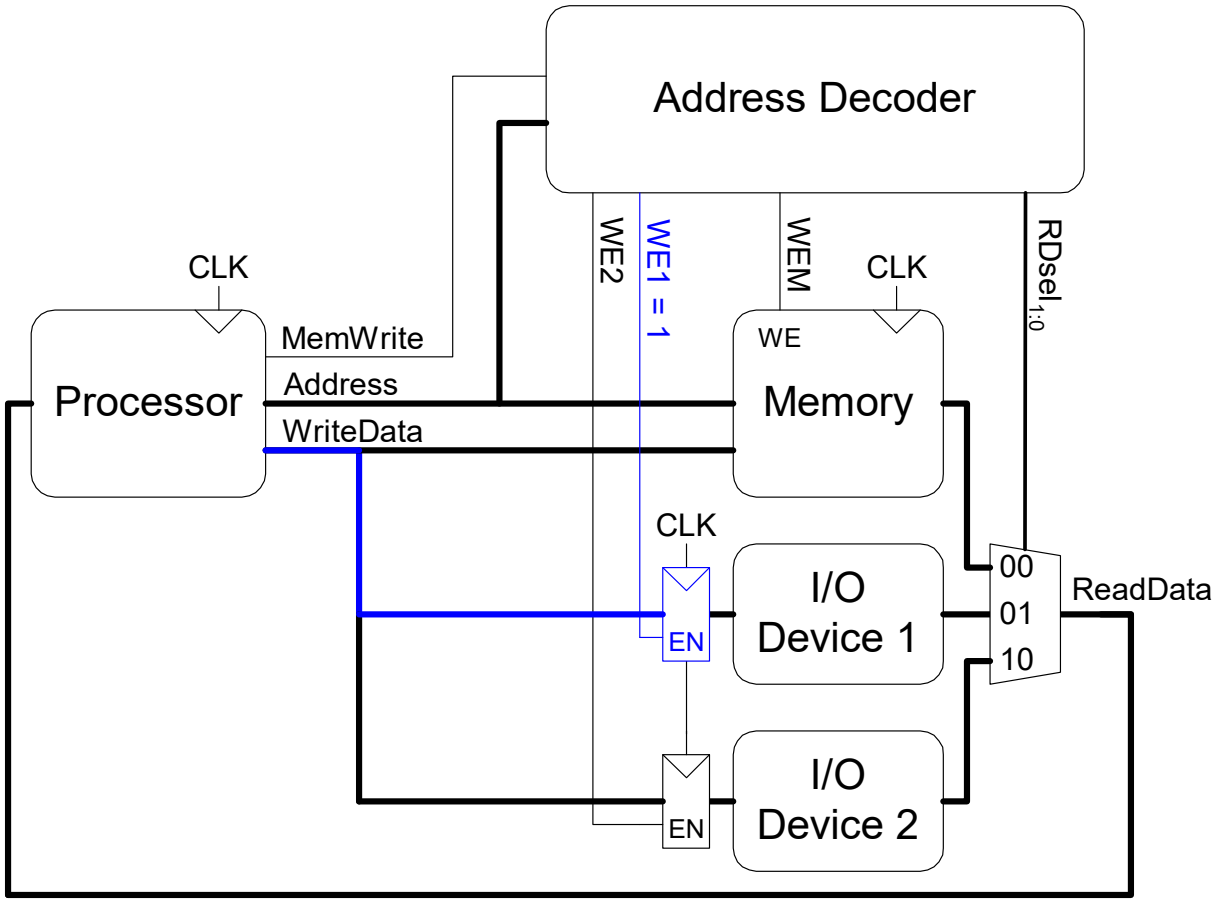
Memory-Mapped I/O Code

- Suppose I/O Device 1 is assigned the address 0xFFFFFFF4
  - Write the value 42 to I/O Device 1
  - Read the value from I/O Device 1 and place it in $t3

Memory-Mapped I/O Code: Write

Write 42 to I/O Device 1 (0xFFFFFFF4)

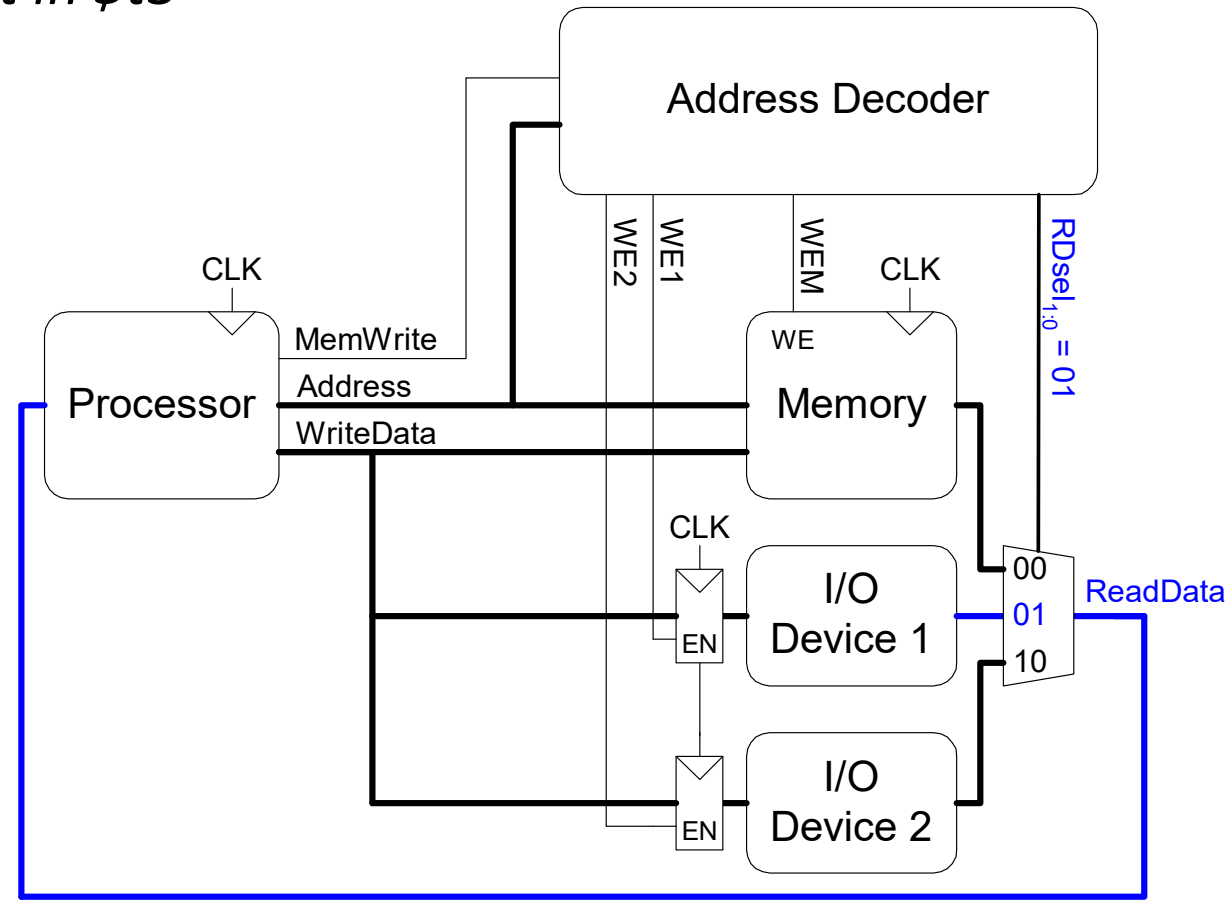
```
addi $t0, $0, 42
sw   $t0, 0xFFF4($0)

# Recall that the 16-bit immediate
# is sign-extended to 0xFFFFFFF4
```



Memory-Mapped I/O Code: Read

Read from I/O Device 1 and place it in $t3

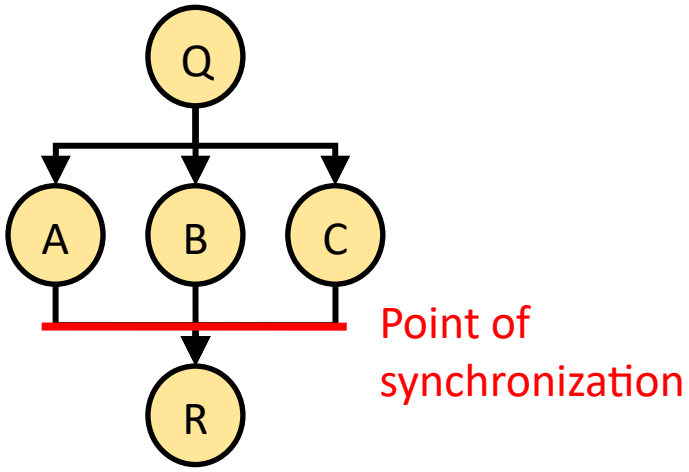
```
lw $t3, 0xFFF4($0)



# Recall that the 16-bit
immediate
# is sign-extended to 0xFFFFFFF4
```



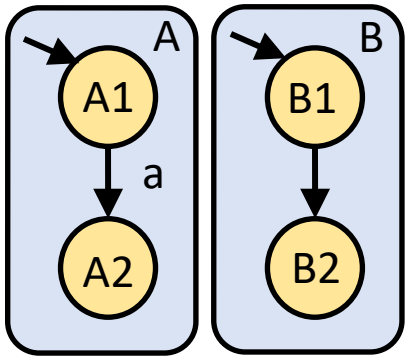
Model Characteristics …

- Synchronization
  - Concurrent processes are never fully independent from each other.
  - Sync to exchange data
  - Connect HW/SW subsystems

Control oriented sync
Control structure of functions determine sync



Data oriented sync
Sync by using inter-process communication
(shared memory, message passing)



Model Characteristics …

- Exception Handling
  - Events like a reset or interrupt can abruptly terminate a process.
  - If such event/exception occurs control is passed to a pre-defined exception handling routine.
- Non-Determinism
  - Allows specification of multiple options due to unclear best suiting operations for app
    - Put off final decision for later in design process

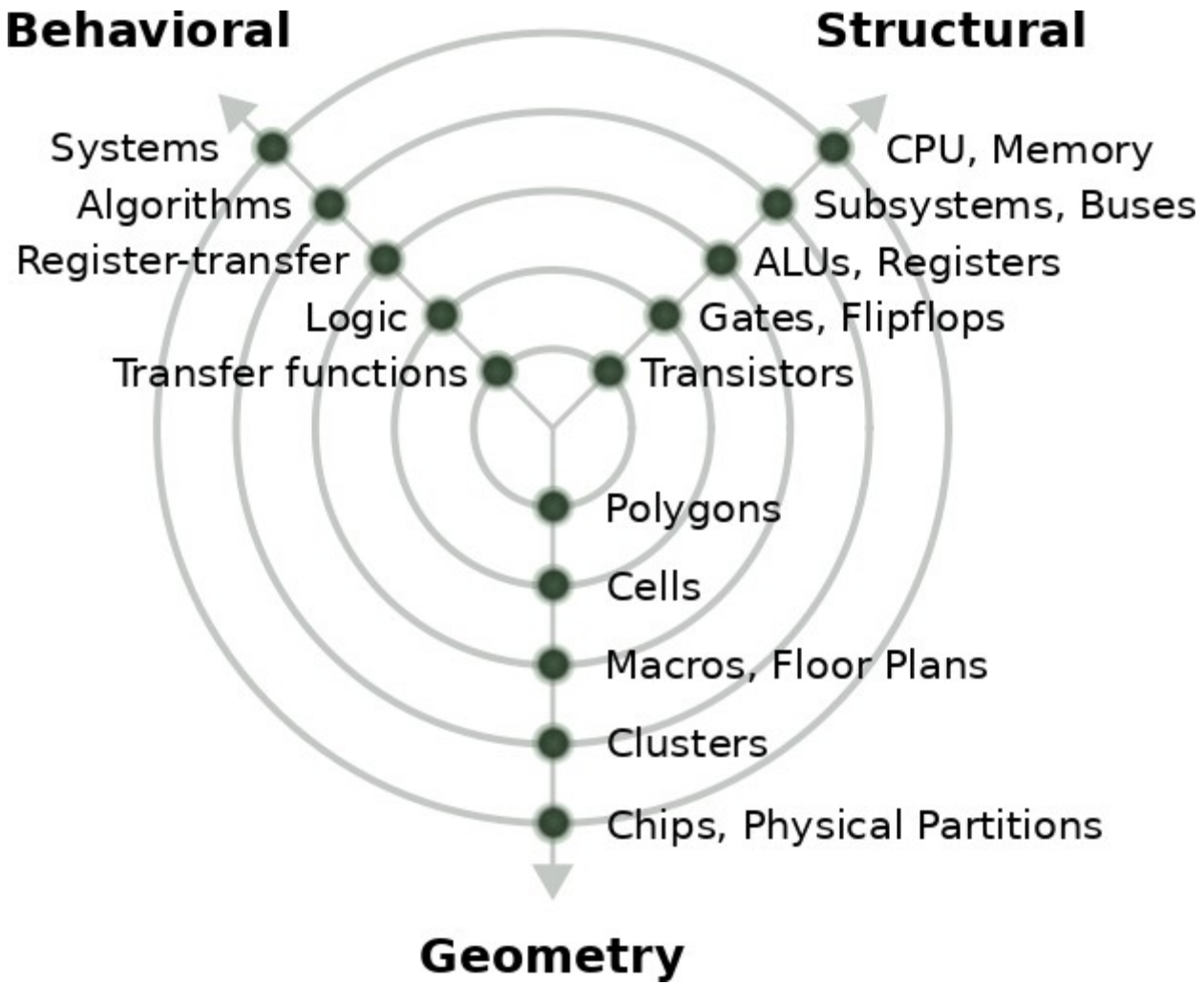
Universität Bremen

HW/SW Codesign

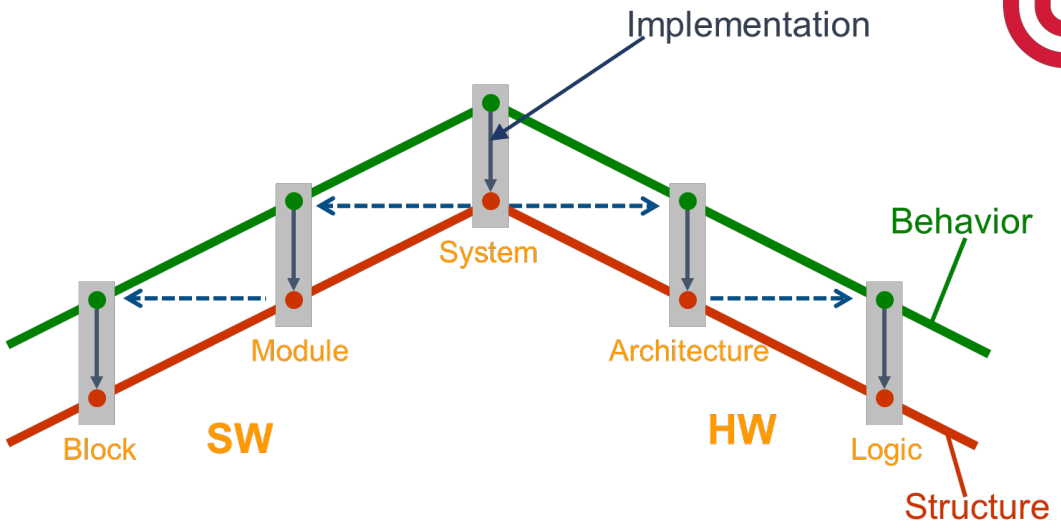
Concepts

Prof. Dr. Rolf Drechsler
Dr. Muhammad Hassan
M.Sc. Jan Zielasko
M.Sc. Milan Funck

- Backup


Behavioral
Structural
Systems
Algorithms
Register-transfer
Logic
Transfer functions
CPU, Memory
Subsystems, Buses
ALUs, Registers
Gates, Flipflops
Transistors
Polygons
Cells
Macros, Floor Plans
Clusters
Chips, Physical Partitions
Geometry

Design Views



Design Step	Starting Point	End Point
Implementation	Behavior/Structure	Structure
Analysis	Structure	Behavior
Optimization	Momentary iteration of a particular view and level	Improved iteration of same view and level
Refinement	Abstract design representation	More detailed design representation under same view
Synthesis	Behavior	More detailed and optimized structure
Abstraction	Detailed design representation	More abstract design representation under same view
Generation	Structure	Physical/geometric design
Extraction	Physical/Geometric design representation	Structure