

EE-334

Digital System Design

Digital Systems

Andreas Burg

Digital Systems are Everywhere Today

- **Development** of digital systems is **complex and expensive**
- **Different markets** with different tradeoffs between volume and price
 - Different levels of complexity
 - Different volumes (from dozens to billions)
 - Different costs per device (cents to billions)

Economy of scale



Low volume
high price



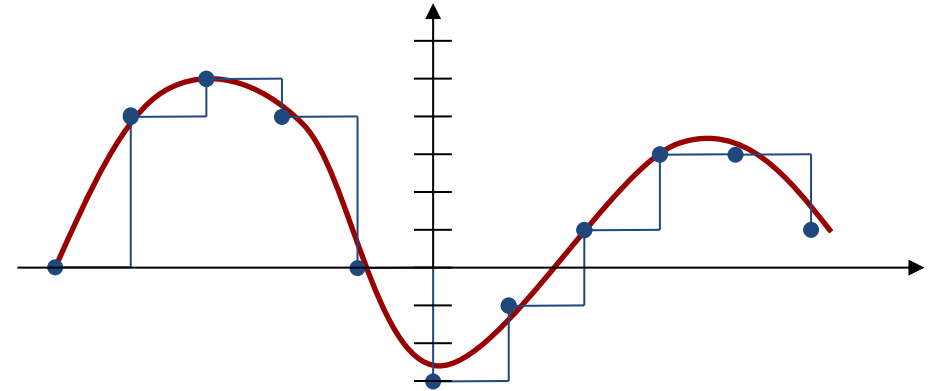
Medium volume
Medium price



High volume
Low price

What is a Digital System?

- As opposed to a **analog systems**, **digital systems** represent **everything** with **numbers** rather than physical quantities
- This automatically implies the need to
 - Discretize time
 - Discretize amplitude
- Discretization of amplitude: values expressed as integers
 - Numbers are typically represented in binary format
 - High immunity against noise
- Discretization of time: relate every action to a (periodic) time reference, often called a clock



0100, 1001, 1110



Why Binary (Discrete Time) Digital Systems?

- Operation of a digital system

Physical quantity
(e.g., light)

represented
as numbers...

...in binary
format with
each digit ...

represented as
physical quantity
(e.g., voltage)

used in
binary analog
computations
(DIGITAL LOGIC)

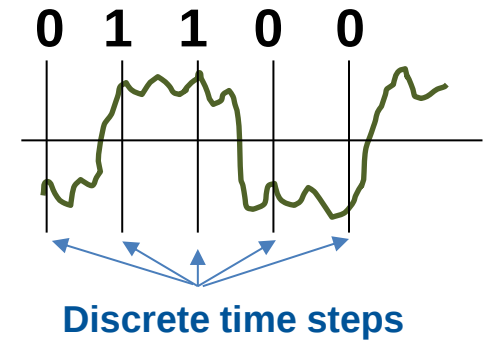
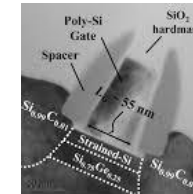
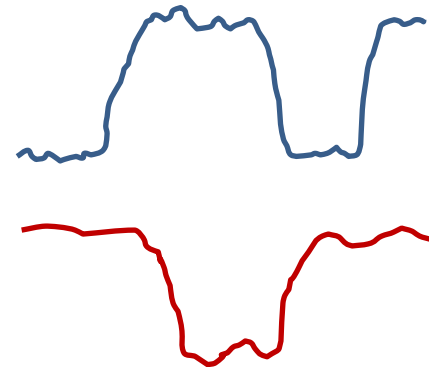
Leads to an analog
result that is
quantized back
to a binary number



000234523000
000213233200
000723832400
002450482840
000323775900
000036893000

00011

01000



- Computations are still performed with analog on physical quantities (voltages)
- **Binary representation** provides **robust against noise**
- **Discrete time** operation provides **robustness against uncertainties in time**

Ingredients for a Digital System

- Digital systems are often comprised of both **hardware** and **software**

Focus of this class



Physical components
that operate on or store
digital signals

Always required



Sequence(s) of **instructions**
that **direct the hardware** to
perform specific operations

Not always required

Key Differences Between Hardware and Software

Hardware

- Operates on physical quantities under the laws of physics
- Is a **composition of components** (abstracted devices)
 - Operates in **parallel**
- Fundamental components have **no inherent notion of time**
 - Ordered sequence of operations expressed by concatenating components
 - Time must be created artificially
- **Data shared through connections**
- Hardware and software design require **fundamentally different mindsets**

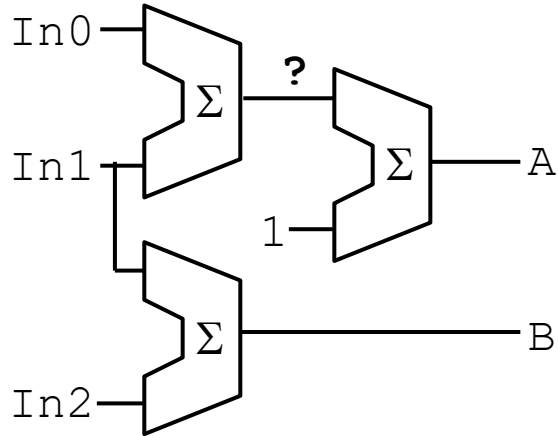
Software

- Operates on abstract variables and numbers
- Is an inherently ordered **sequence of instructions**
 - Operates in **sequentially**
- Sequential nature of a **program implies order**
 - Ordered sequence of operations by executing them one after the other
 - Parallelism must be created artificially
- **Data shared through variables**

Key Differences Between Hardware and Software

Hardware

- Best **described as block diagram** (i.e., components / parallelism)



More operations naturally
scale with resources

Software

- Best described as **sequence of instructions** (program)

```
int In0=1;  
int In1=2;  
int In2=3;  
int A, B;
```

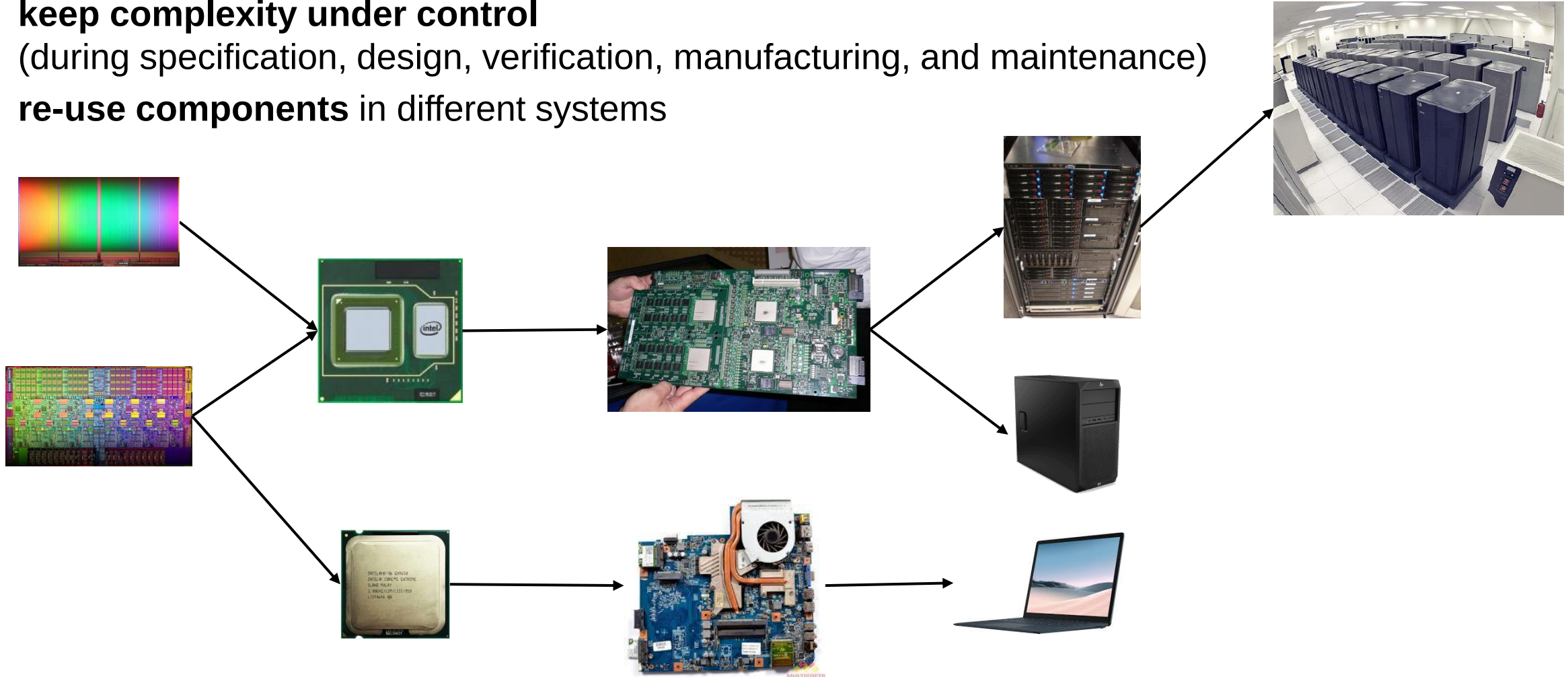
```
A = In0 + In1;  
A = A + 1;  
B = In1 + In2;
```

More operations naturally
scale required time

- Hardware and software design involves **fundamentally different tradeoffs**

Building Complex Digital Systems (Hardware)

- Digital Systems are built hierarchically to
 - **keep complexity under control**
(during specification, design, verification, manufacturing, and maintenance)
 - **re-use components** in different systems

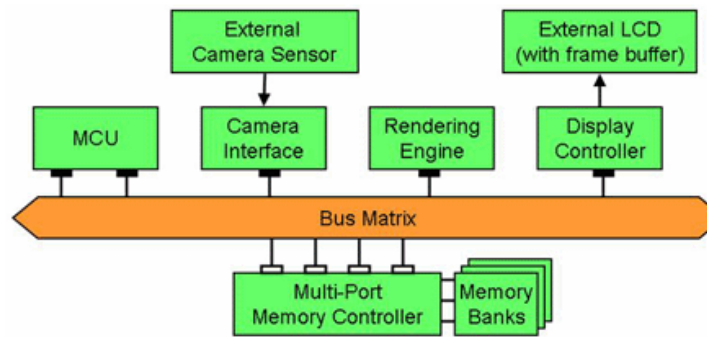


Designing with Abstraction

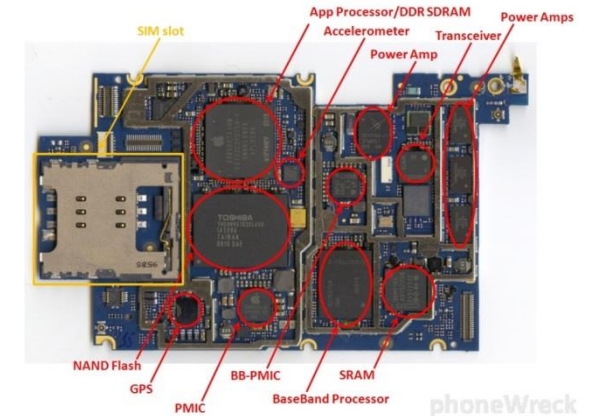
- Abstracting helps to **structure the design process**
- Different **levels of abstraction** are often called **VIEWS**
- **Three different views** are frequently used
 - **Functional view**: tasks (functions) that fulfill specific purposes
 - **Architectural view**: functional units (architectural components) needed to carry out the tasks
 - **Physical/technology view**: technologies to implement and connect architecture components



Functional view



Architectural view



Physical / technology view

Four Steps to Build a Digital System

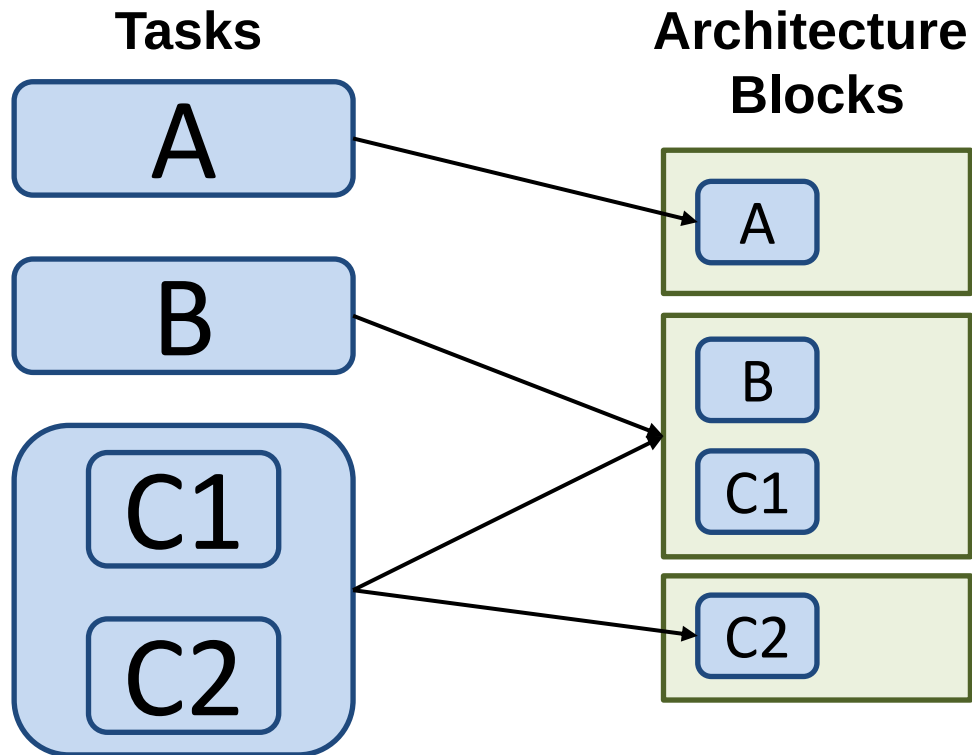
- **Specification:** describe intended behavior and requirements (what not how)
- **Functional design:** define and partition behavior into tasks based on
 - Interaction between tasks
 - Communication requirements between tasks
 - Nature of the tasks (e.g., datapath or control)
- **Architectural design:** define the architectural building blocks of the system, how they are connected and which tasks they execute
- **Implementation:** choose implementation option for building blocks based on
 - Overall constraints (e.g., area, power, cost, ...)
 - Requirements and nature of the task (e.g., computational load, memory needs, ...)
 - Communication requirements between subtasks

Architectural Building Blocks

- **Characteristics of a task decide** which **type of architecture** is required
- **Resources can be**
 - generic or specific for a task
 - custom built or already available
- **Examples** for resources
 - **I/O Interfaces:** .connect components or a system to the outside world
 - **Custom digital logic:** tailored to a very specific task
 - **Dedicated digital logic:** standard function for a specific, but common task
 - **Programmable processors:** execute software
 - **Storage arrays:** keep large amounts of data in a “hardware-efficient” way
 - Volatile memory (SRAM, DRAM)
 - Non-volatile memory
 - **Interconnect / busses:** provides communication architectural building blocks

Mapping Tasks to Architecture Building Blocks

- **Assigning tasks to architecture blocks is NOT a 1:1 mapping**
 - A **single block** in an architecture can sometimes **handle or serve multiple tasks**
 - A **single task** (or different sub-tasks), carried out **on multiple (interacting) blocks**

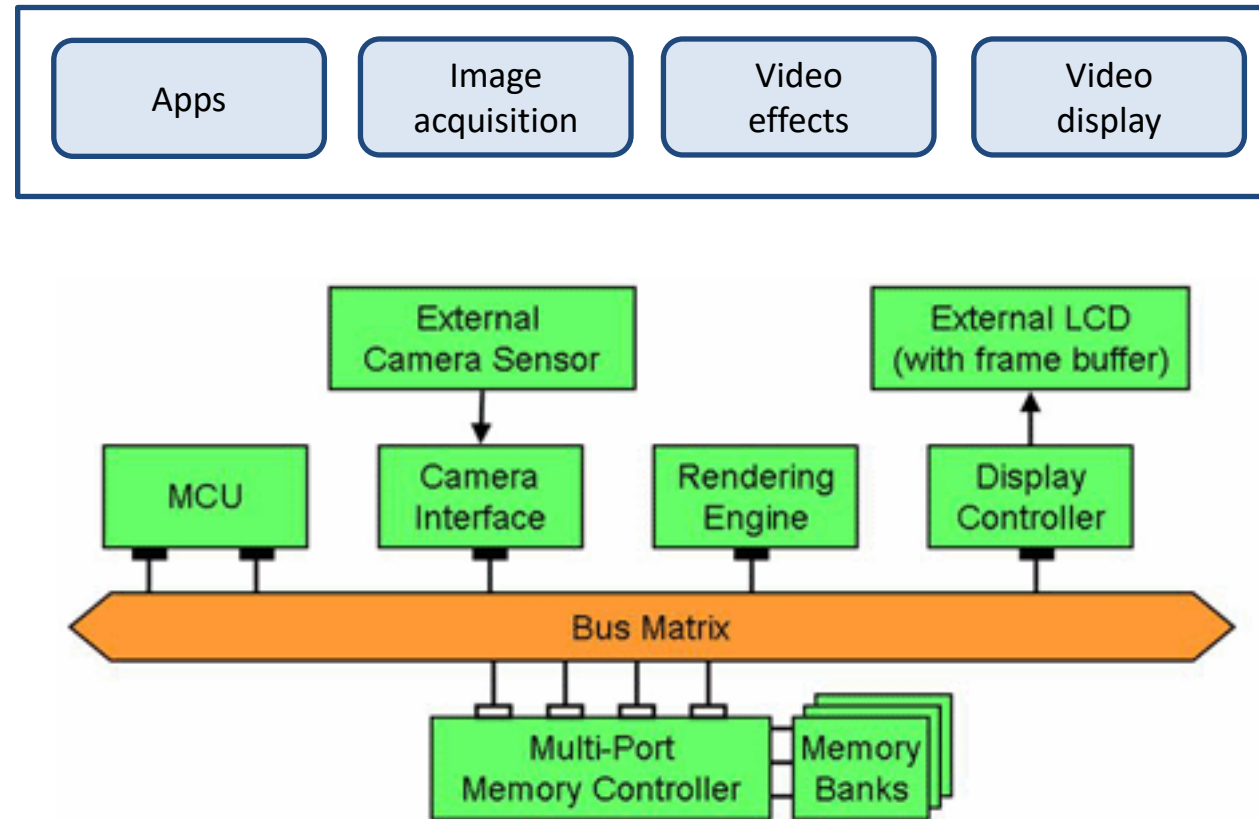


- **Consideration**

- Processing requirements and capabilities
- Real-time requirements (latency)
- Local memory requirements of a task
- Capability of an architecture to carry out multiple tasks (parallel processing)
- Flexibility of an architecture (ability to map tasks with different characteristics)
- Communication requirements between tasks and sub-tasks
- Communication interface protocols

Mapping Tasks to Architecture Building Blocks

- **Example:** An integrated camera system with live display and small apps



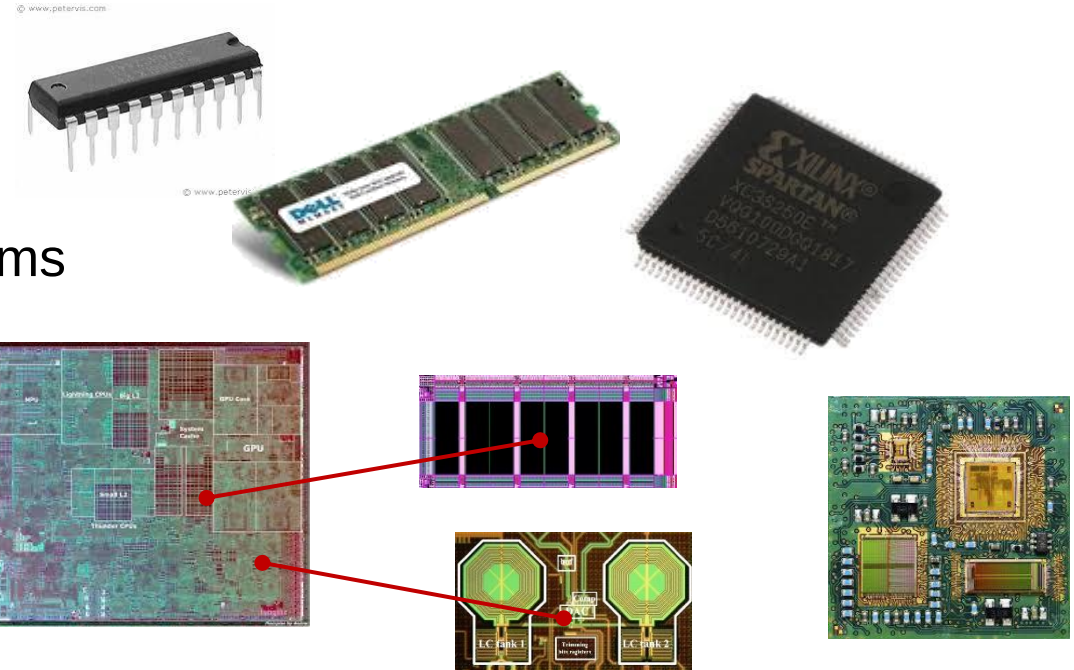
Implementation Options

- Often there are **many options** for realizing an **architectural building block**
- Implementation **options** often **differ significantly** in
 - Technical capabilities and performance
 - Ability to communicate with other components
 - Effort and cost during design and for the final product

Often many options are not valid since they can not meet the requirements of the architecture

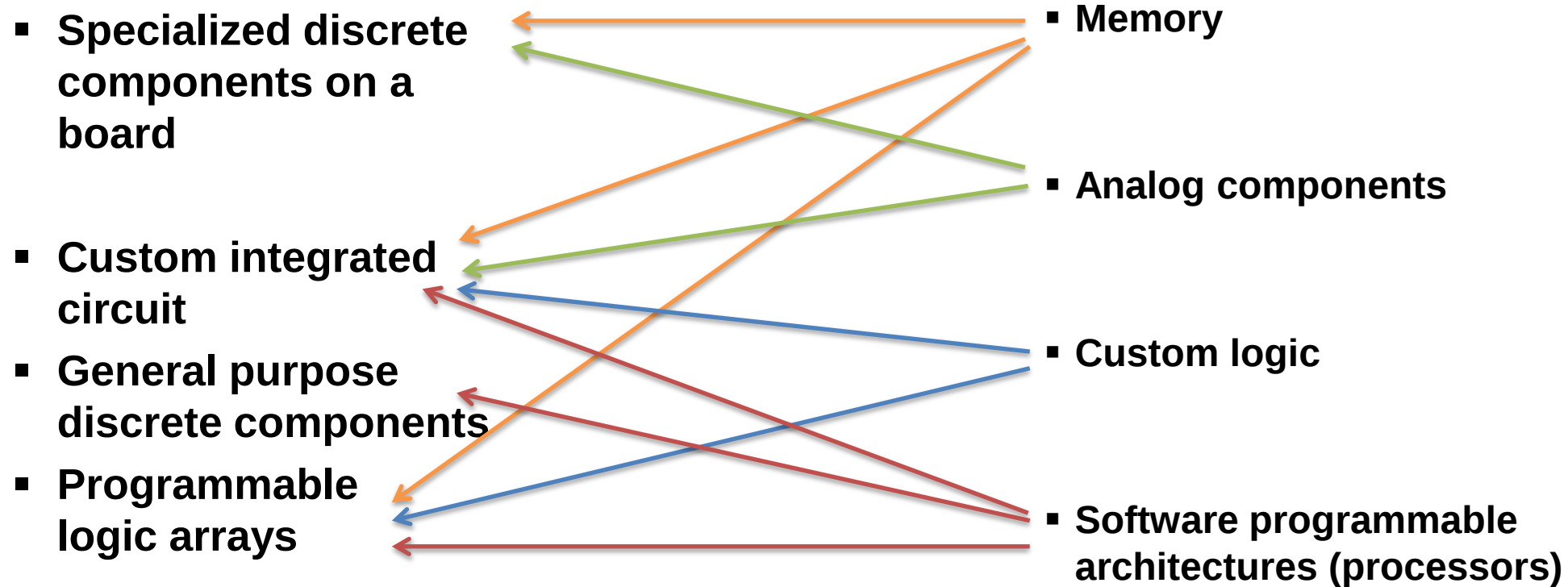
- **Examples** for resources

- Discrete logic elements or components
- Off-the-shelf complex components (ICs) or systems
- Programmable logic devices (FPGAs)
- Custom integrated circuits
 - Custom designed on-chip building blocks
 - Available IP components
 - Full custom analog blocks



Deciding on Implementation Options

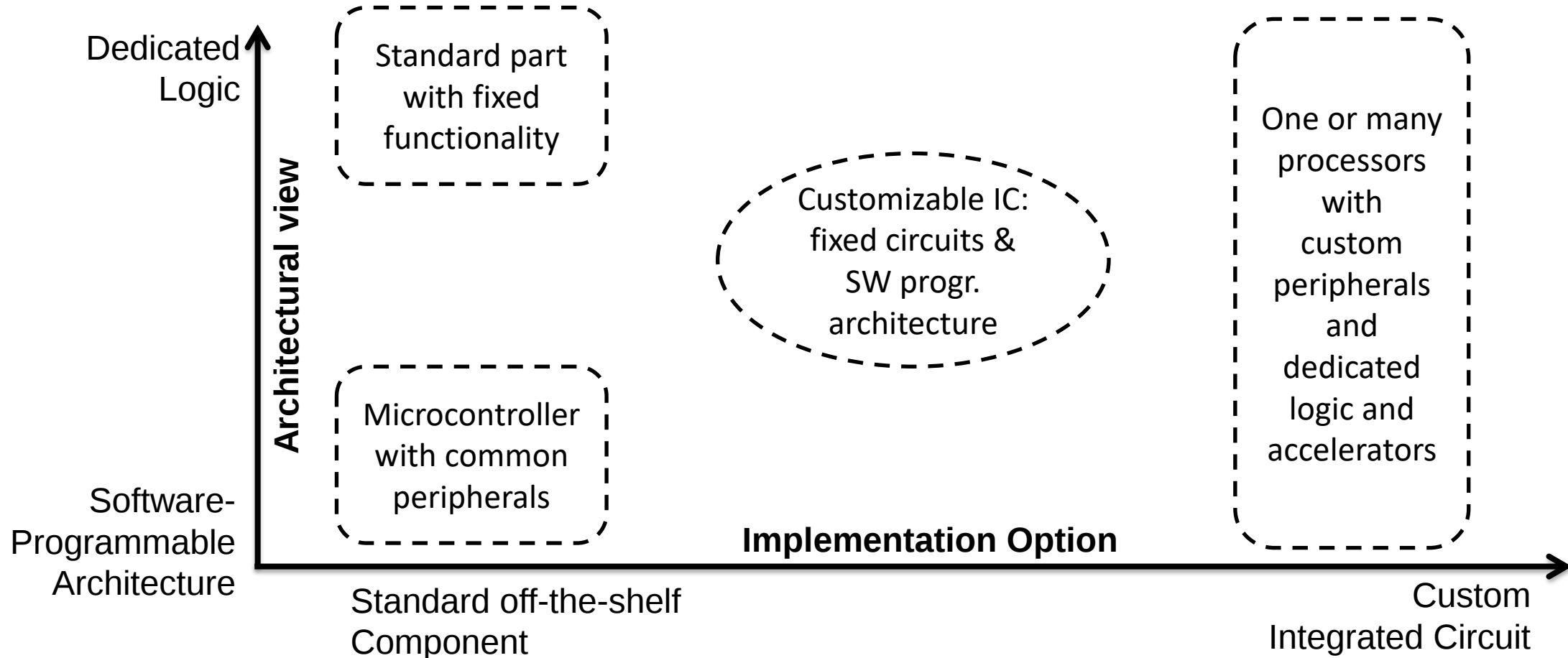
- Often **more than one option** exist to **implement** an architecture building block



- Implementation options often **impact** the **architecture** → need **iterations**

Examples for Different Implementation Options

- Different **implementation options** often offer a large design space



Trend Toward Higher Integration

- **Reducing the number of physical components**
 - reduces unit cost
 - improves communication between components
 - reduces power consumption
 - improves reliability
- This integration comes with several difficulties
 - **Single technology can not always meet requirements of all architectural building blocks**
 - Non Volatile Memory (NVM) is not easily available in all CMOS technologies
 - Sensors or dense memories often require special technologies
 - FPGAs are not optimal for all typed of digital logic
 - ...
 - **Dense integration often** requires custom design: expensive and time consuming

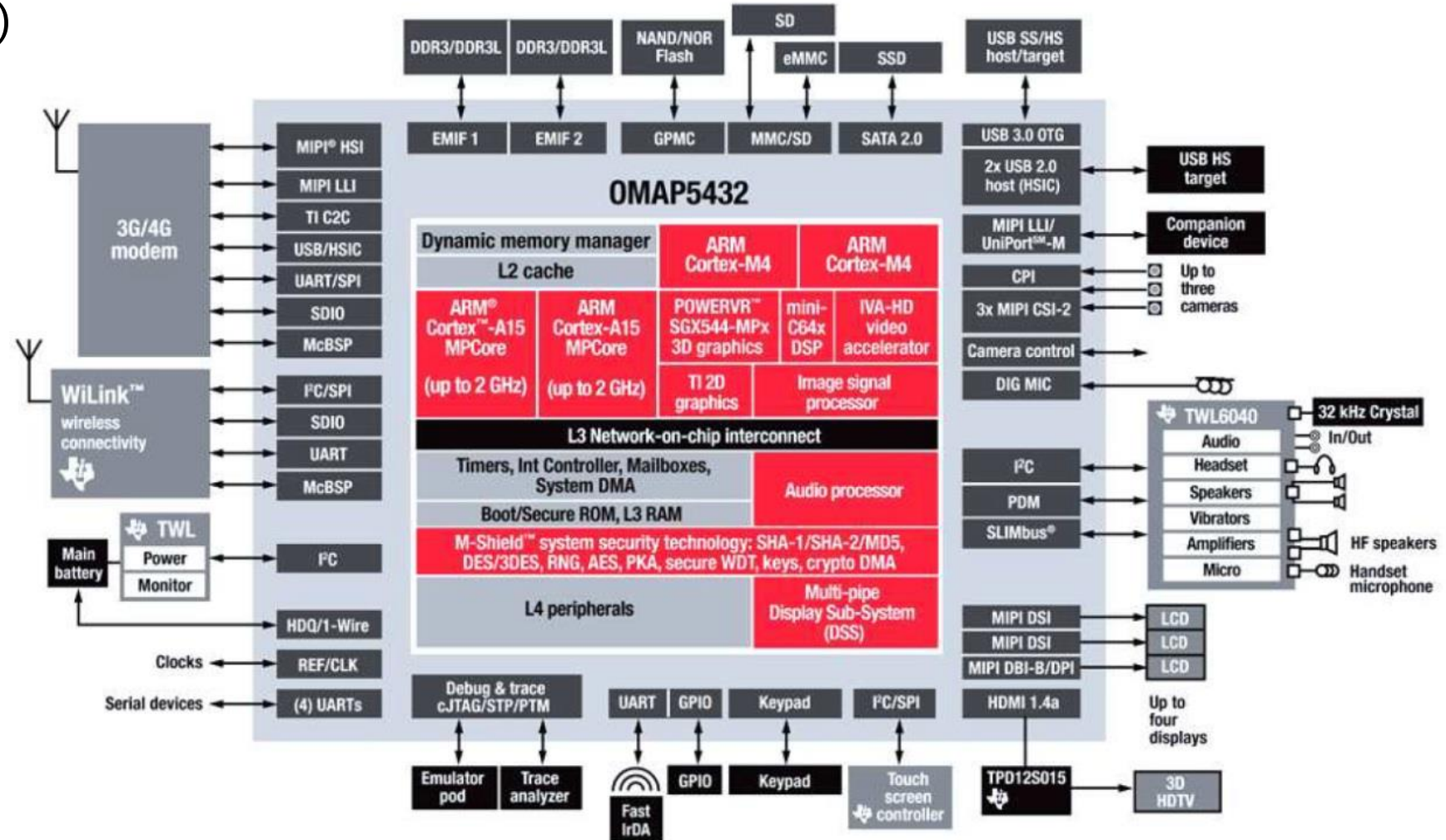


**System on Chip
(SoC)**

System Architecture Example: Mobile Phone

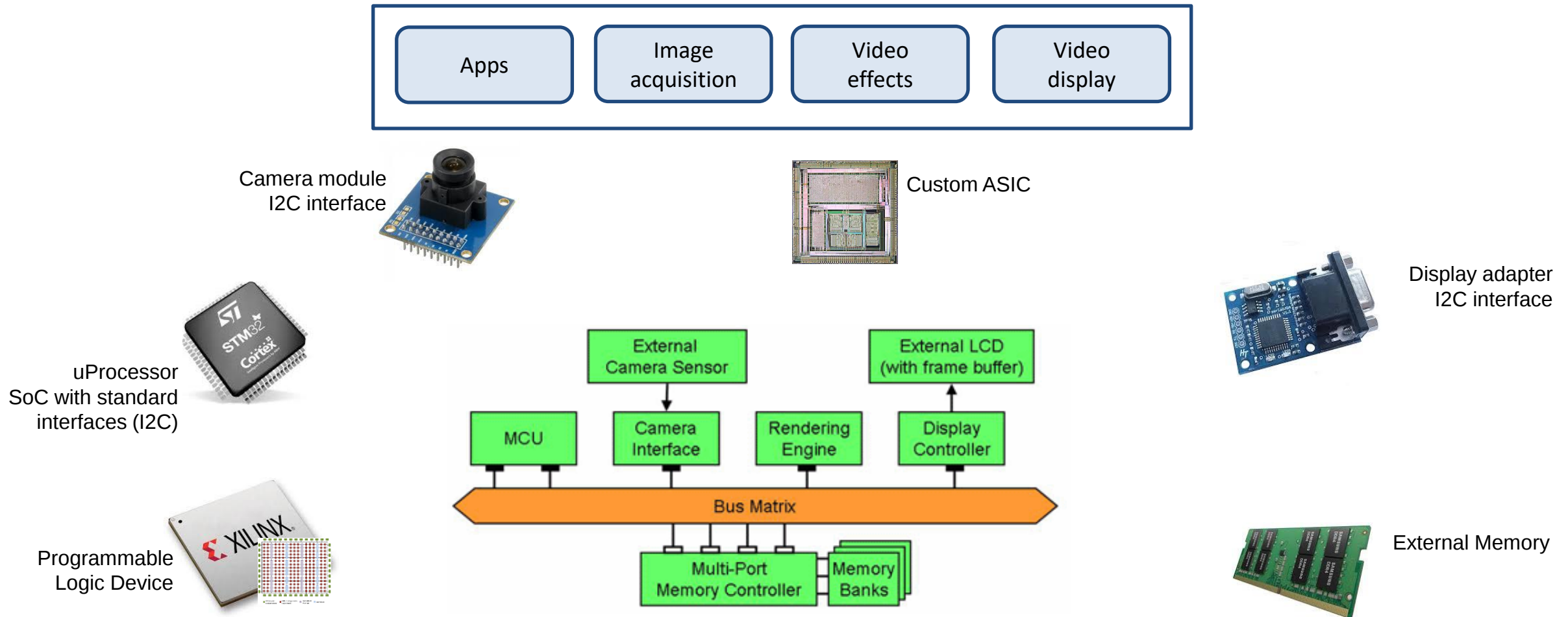
- **Architecture of 3rd generation phone** based on multiple integrated circuits with

- TI System-on-Chip (SoC) as central component
- External NVM
- External DRAM
- External wireless modem (RF/DSP)
- External power management



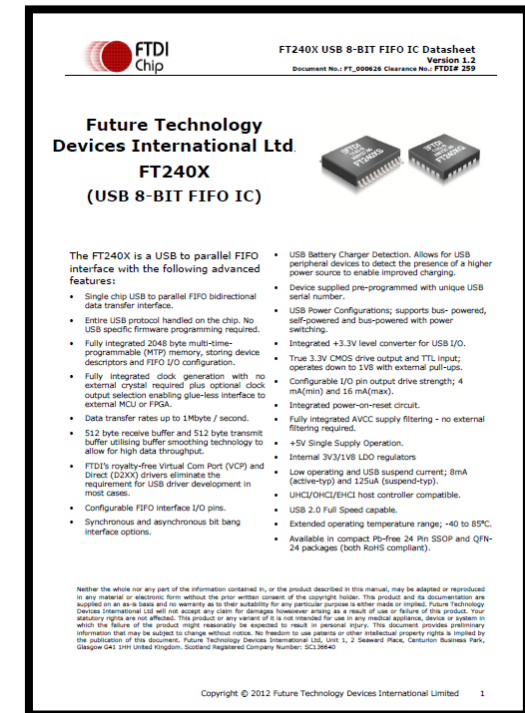
Mapping Architecture Blocks to Impl. Options

- **Example:** An integrated camera system with live display and small apps



How to Select Available Components?

- Almost all **systems include** also some **standard components**
- Component selection and interaction with them requires understanding
 - **Functionality** and performance characteristics
 - **Interfaces**: communication protocols
 - Electrical specifications: different IO standards
 - Power supply and thermal requirements
 - Mechanical and thermal properties and requirements
- This **information** is found in **datasheets**
 - Datasheets for most components are available online
 - Datasheets often contain also other useful information
 - a rough block diagram of a circuit
 - application notes and examples



Example: FTDI FTX240