Embedded Systems - Complete Summary (WIP)

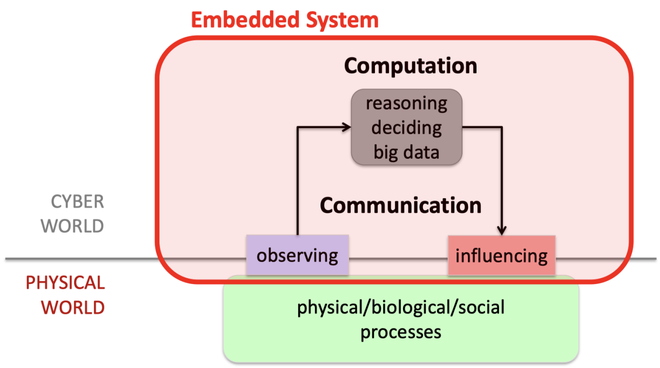
Ruben Schenk, ruben.schenk@inf.ethz.ch

January 13, 2022

Chapter 1: Introduction

1.1 Impact

Embedded systems are information processing systems embedded into a larger product. Usually they use feedback to influence the dynamics of the physical world by taking smart decisions in the cyber world.



1.2 Facts

Embedded systems are often *reactive:* reactive systems must react to stimuli from the system environment.

> "A reactive system is one which is in continual interaction with its environment and executes at a pace determined by that environment" - Bergé, 1995

ES often must meet *real-time constraints:* For hard real-time systems, right answers arriving too late are wrong. All other time-constraints are called soft. A *guaranteed system response* has to be explained without statistical arguments.

> "A real-time constraint is called hard, if not meeting that constraint could result in a catastrophe" - Kopetz, 1997

It is essential to *predict* how a cyber-physical system (CPS) is going to behave under any circumstances before it is deployed. CPS must *operate dependably,* safely, securely, efficiently and in real-time.

ES must be *efficient:*

- Energy efficient
- Code-size and data memory efficient
- Run-time efficient
- Weight efficient
- Cost efficient

ES are often *specialized* towards a certain application or application domain: Knowledge about the expected behavior and the system environment at design time is exploited to *minimize resource usage* and to *maximize predictability and reliability.*

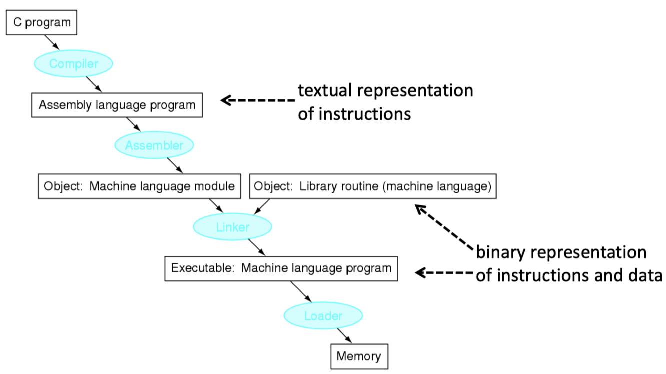
1.3 Trends

Some trends of embedded systems:

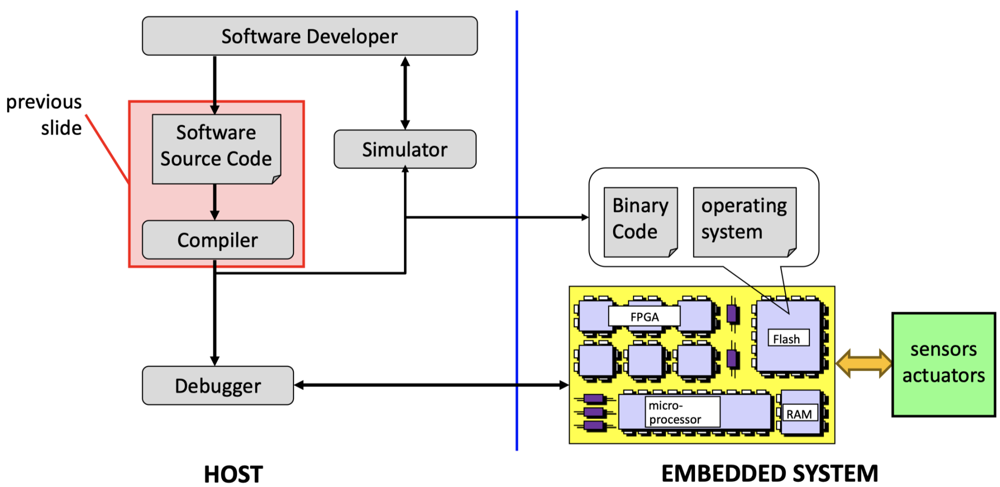
- ES communicating with each other, with servers or with the cloud. Communication is increasingly.
- Higher degree of integration on a single chip or integrated components:
  - Memory + processor + I/O units + communication
  - Use of networks-on-chip for communication between units
  - Use of homogeneous or heterogeneous multiprocessor system on a chip (MPSoC)
- Low power and energy constraints (especially for portable or unattended devices) are increasingly important, as well as temperature constraints
- There is increasing interest in energy harvesting to achieve long term autonomous operation

Chapter 2: Software Development

Reminder: Compilation of a C program to machine language works as follows:



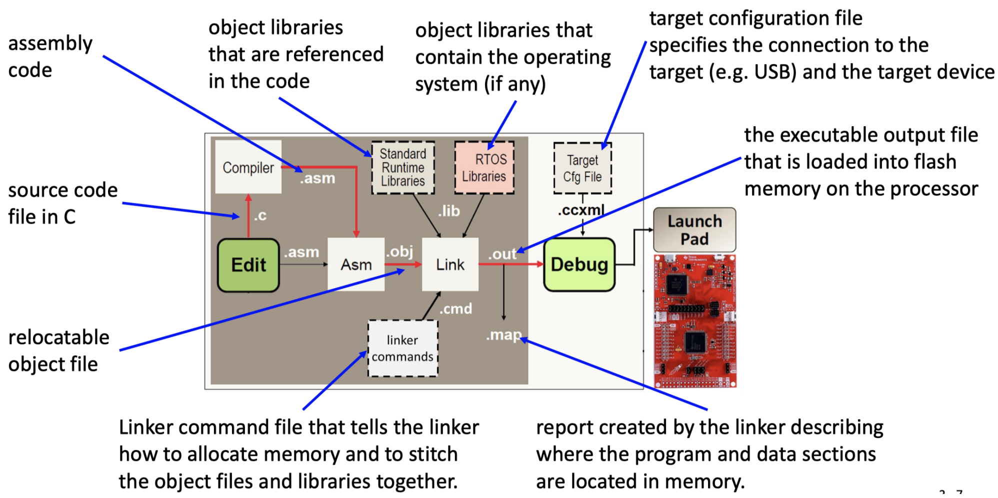
The main chain-of-events for **embedded software developments** is given by the following diagram:



Software development is nowadays usually done with the support of an IDE:

- Edit and build the code
- Debug and validate the code

A better overview on how this works with embedded systems is given below:



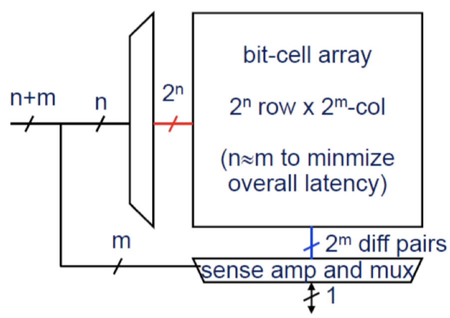
Chapter 3: Hardware Software Interface

3.1 Storage

3.1.1 SRAM / DRAM / Flash

In a **static random access memory (SRAM),** single bits are stored in bit-stable circuits. SRAM is used for:

- Caches
- Register files within the processor core
- Small but fast memories



If we want to *read* from SRAM:

1. Pre-charge all bit-lines to average voltage
2. Decode the address ($n + m$ bits)
3. Select the row of cells using n single-bit *word lines (WL)*
4. Selected bit-cells drive all *bit-lines (BL)* (2^m pairs)
5. Sense difference between bit-line pairs and read out

If we want to *write* to SRAM:

1. Select row and overwrite the bit-lines using strong signals

In **dynamic random access memory (DRAM),** single bits are stored as charges in capacitors:

- Bit cells lose their charge when they are read, and they drain over time
- Slower access than with SRAM due to small storage capacity in comparison to the capacity of bit-lines
- Higher density than SRAM (1 vs. 6 transistors per bit)

DRAMs require *periodic refresh* of charge:

- Performed by the memory controller
- Refresh interval is tens of ms

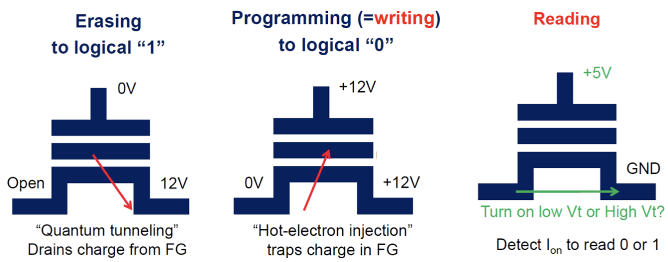
- DRAM is unavailable during refresh

A typical *access process* for DRAM is given by the following four steps:

1. Bus transmission from CPU to memory controller
2. Precharge and row access from memory controller to row decoder and then from memory array to the sense amps.
3. Column access from memory controller to column decoder and then from sense amps to the data in/out buffers.
4. Data transfer and bus transmission from data in/out buffers to memory controller and from there via the bus to the CPU.

Flash memory is electrically modifiable, non-volatile storage. It has the following principle of operation:

- Transistors with a second "floating" gate
- Floating gate can trap electrons
- This results in a detectable change in threshold voltage



3.1.2 Memory Map

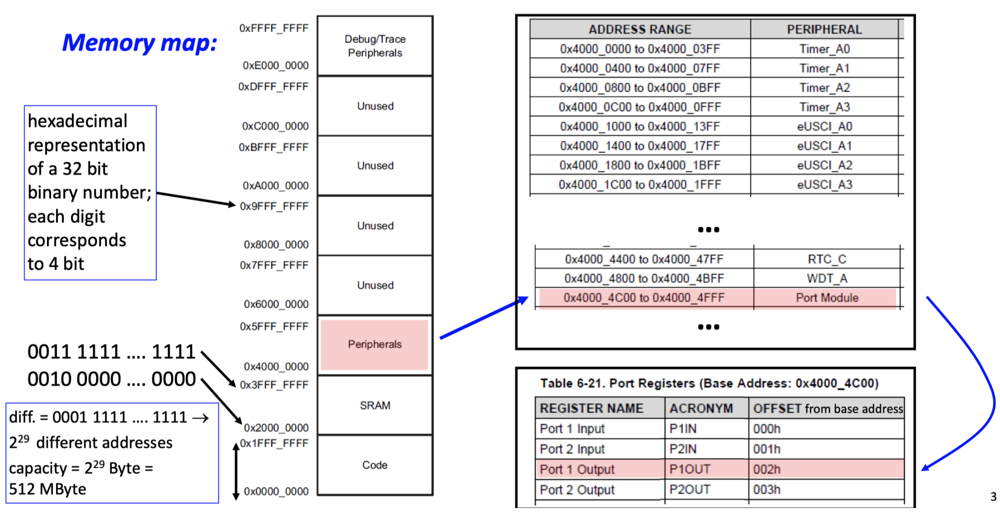
We look at the **memory map** by exploring the example of the MSP432. Its *available memory* is given by:

- 265 kB of built-in flash memory
- 64 kB SRAM
- 32 kB ROM (read-only memory)

The *address space* is built up as follows:

- The processor uses 32 bit addresses. Therefore, the addressable memory space is 4 GByte = 2^{32} Byte as each memory location corresponds to 1 Byte.
- The address space is used to address the memories (reading and writing), to address the peripheral units, and to have access to debug and trace information.
- The address space is partitioned into zones, each one with a dedicated use.

Example: The following is a simplified description to introduce the basic concepts:



3.2 Input and Output

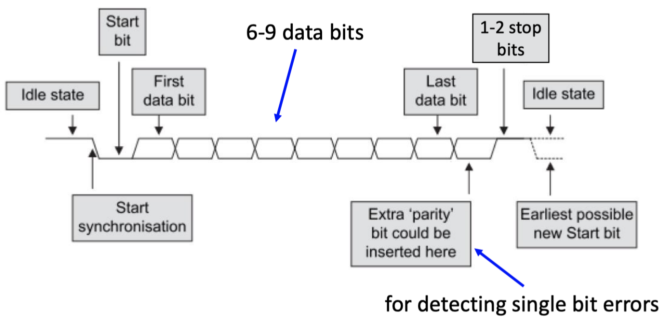
Very often, a processor needs to *exchange information with other processors* or devices. To satisfy the various needs, there exist many *communication protocols,* such as:

- Universal Asynchronous Receiver-Transmitter (UART)
- Serial Peripheral Interface Bus (SPI)
- Inter-Integrated Circuit (I2C)
- Universal Serial Bus (USB)

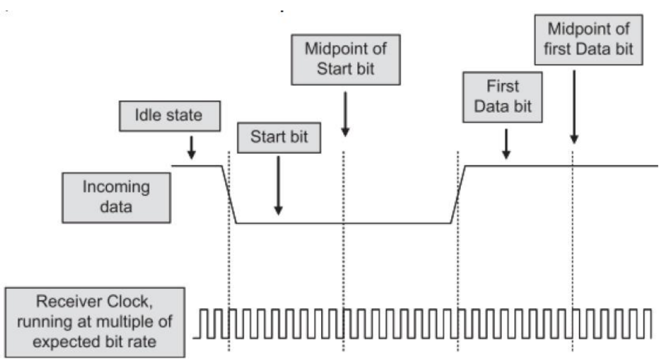
AS the principals are similar, we will just explain a representative of an asynchronous protocol (UART) and one of a synchronous protocol (SPI).

3.2.1 UART Protocol

The **Universal Asynchronous Receiver-Transmitter (UART)** protocol provides *serial communication* of bits via a single signal, i.e. UART provides parallel-to-serial and serial-to-parallel conversion. The sender and the receiver need to *agree on the transmission rate.* Transmission of a serial packet starts with a start bit, followed by data bits and is finalized by using a stop bit:



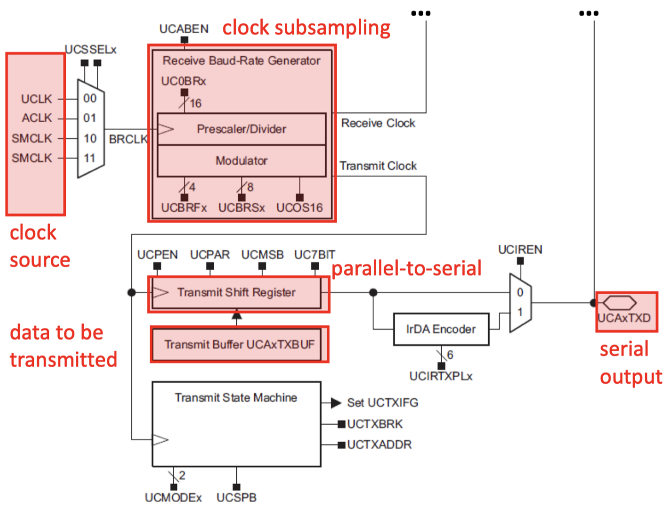
The receiver runs an *internal clock* whose frequency is an exact multiple of the expected bit rat. When a *start bit* is detected, a counter begins to count clock cycles, e.g. 8 cycles, until the midpoint of the anticipated start bit is reached. The clock counter counts a further 16 cycles, to the middle of the first *data bit,* and so on until the *stop bit:*



3.2.2 Memory Mapped Device Access

The configuration of the transmitter and the receiver must match, otherwise they cannot communicate. Examples of configurable parameters are:

- Transmission rate
- LSB or MSB first
- Number of bits per packet
- Parity bit
- Number of stop bits
- Interrupt-based communication
- Clock source



The *clock subsampling* block is complex, as one tries to match a large set of transmission rates with a fixed input frequency. The clock source is based on the quartz frequency (48 MHz), divided by 16 and then connected to SMCLK (in the labs, the SMCLK frequency is therefore 3 MHz).