EFOP-3.4.3-16-2016-00009

A felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása a Pannon Egyetemen

EMBEDDED SYSTEM DEVELOPMENT

(MISAM154R)

Created by Zsolt Voroshazi, PhD

voroshazi.zsolt@virt.uni-pannon.hu

Updated: 25. Sept. 2020.

SZÉCHENYI 2020

Magyarország Kormánya

Európai Unió
Európai Strukturális és Beruházási Alapok

BEFEKTETÉS A JÖVŐBE

EDUCATIONAL AIMS

- In the fall semester of 2010/11 „***Design Methods with Programmable Logic Devices (VHDL)***" was started, meeting the actual industry requirements in the course of Electrical Engineering BSc, which provides an introduction to VHDL-based design of FPGA-based digital networks.
- The subject called „***FPGA-based Embedded Systems***" was launched from the fall semester of 2011/12. It is a compulsory laboratory for Electrical Engineering BSc and from the fall semester of 2014/15, both courses started also in BSc in Computer Science. From fall semester of 2018/19 ***„Embedded System Development"*** was introduced also in Computer Science MSc course.
- During these lab exercises students must work together in small groups to solve the assigned tasks and implement by using *Digilent Zybo FPGA* platforms, thus encouraging them to meet real expectations: methodology for collaborative design, development and testing.
- Based on my 15 years of educational experience and student feedbacks, as well as the interest and needs of the industrial partners, a niche presentation has been now prepared based on internationally applied literature. Some parts of this presentation are based on Xilinx Vivado Embedded System Design Flow - Professor Workshop and Teaching Materials and Xilinx Vivado Embedded Linux on the ARM/MicroBlaze Processors.

Topics covered in this semester

1. **Introduction – Embedded Systems**
2. FPGAs, Digilent ZyBo development platform
3. Embedded System - Firmware development environment (Xilinx Vivado – „EDK” Embedded Development)
4. Embedded System - Software development environment (Xilinx VITIS – „SDK”)
5. Embedded Base System Build (and Board Bring-Up)
6. Adding Peripherals (from IP database) to BSB
7. Adding Custom (=own) Peripherals to BSB
8. Development, testing and debugging of software applications – Xilinx VITIS (SDK)
9. Design and Development of Complex IP cores and applications (e.g. camera/video/ audio controllers)

References

- Fodor Attila, Dr. Vörösházi Zsolt: Beágyazott rendszerek és programozható logikai alkatrészek (TÁMOP 4.1.2) tutorial (2011) – *in hungarian*

  http://www.tankonyvtar.hu/hu/tartalom/tamop425/0008_fodorvoroshazi/Fodor_Voroshazi_Beagy_0903.pdf

  (Recommended chapters : **1. Beágyazott rendszerek**, **2.9 Buszok, beágyazott processzorok**)

- Xilinx Teaching Materials

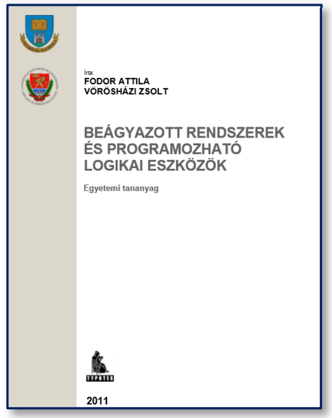
  https://www.xilinx.com/support/university/course-materials.html

- Digilent ZyBo FPGA board data sheets:
  - Digilent ZyBo HW platform:

    https://store.digilentinc.com/zybo-zynq-7000-arm-fpga-soc-trainer-board/

  - Digilent PMOD peripheral modules:

    https://store.digilentinc.com/pmod-modules-connectors/



Further references

- XILINX official website:
  - 🕮 http://www.xilinx.com
- EE Journal – Electronic Engineering:
  - 🕮 http://www.eejournal.com/design/embedded
- EE Times - News:
  - 🕮 http://www.eetimes.com/design/embedded

1. INTRODUCTION

Embedded Systems

SZÉCHENYI 2020

MAGYARORSZÁG KORMÁNYA

Európai Unió
Európai Strukturális és Beruházási Alapok

BEFEKTETÉS A JÖVŐBE

OVERVIEW

- Introduction – What does Embedded System mean?
- **FPGA/APSoC** – **F**ield **P**rogrammable **G**ate **A**rrays / **A**ll **P**rogrammable **S**ystem-**o**n-a-**C**hips?

What are we going to use?

- Hardware: **Digilent ZyBo** platform
- Software:
  - Xilinx VITIS - Unified Software Development Kit (2020.1)
    - Integrated Embedded + Software Dev. Kits
    - Xilinx Vivado Design Suite (version 2020.1) development tools

Embedded Systems

- An „**Embedded System”** is a combination of (computer) **hardware-** and **software** components that perform a given function, a specific (control) task, as opposed to general purpose computer systems.
- Embedded systems includes computer tools which can be integrated with application-oriented target devices (ASIC/ASSP, **FPGA/APSOC**, CPU/MCU, MPU, DSP, GPU, etc.) or complex application systems (even at OS level). Moreover they can operate **autonomously (without manual intervention)**.
  - *„Programmable”* (in our case „reconfigurable”) embedded systems have a programming interface that usually requires specific software/firmware development strategies and techniques.

Embedded Systems = HW + FW + SW (OS)

Application fields

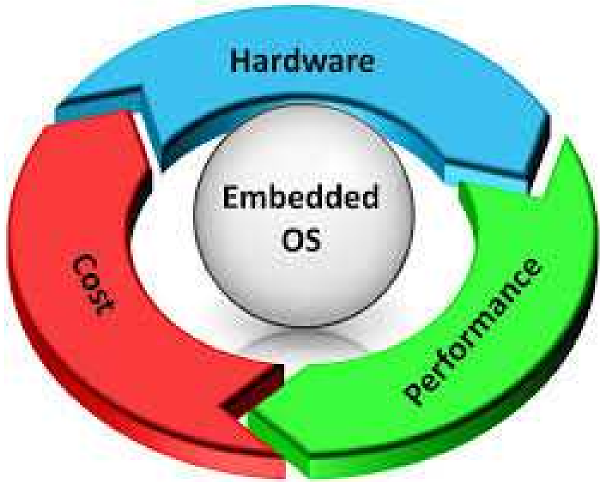
- Automotive Applications: Embedded Electronic Controllers
  - Safety-critical: Central Electronic Controller (ECU), Engine Control, Brake Assist, Transmission, Anti-lock Braking Control (ABS), Traction Control (ESP) Airbag, ADAS - Vehicle On-Board Devices
  - Passenger-oriented (comfort) systems: entertainment, seat / mirror control, etc.
- Aerospace and defense applications
  - Flight control systems (on-board navigation, GPS receiver), engine control, autopilot
  - Defense systems, radar systems, radio systems, missile control systems
- Medical equipment:
  - Medical Imaging, Signal Monitoring (PET, MRI, CT)
- Network / Telecommunication Systems (LTE/5G)
- IoT: Intelligent or smart systems
- WSN: Wireless Sensor Networks (motes)
- Robotics
- Household appliances, or. consumer electronics





General requirements

- Dedicated function
  - Support a well-defined (application specific) function(s)
- Strict requirements
  - Low **C**ost
  - **E**conomy – preferably made of minimal parts
  - **S**peed – preferably fast operation
  - **P**ower – Low dissipation



- Real-time operation and system response
  - continuous monitoring of the environment without manual intervention (if possible)
- Co-Design, Co-Simulation, Co-Verification of hardware and software components

Basic requirements

- **Time**: The embedded system should start processing a task within a specified time (after its occurrence).
- **Safety**: Control of a system that handles an event without causing any damage to human health or material (in case of malfunction).
- Along this philosophy, two classes of embedded systems can be defined:
  - **Real-time system (or time critical)**: where compliance with time requirements is the most important consideration,
  - **Safety-Critical System (no time critical):** where security features are more important than meeting time requirements.
  - *Note: In reality, it is not easy to group embedded systems into these two classes, because there may be real-time systems that have some of the features of security-critical systems (mixed). Standards and laws govern which applications require the use of a security critical system (eg. ADAS ISO 26262).*

Real-time embedded systems

Based on strict requirements, two types of real-time systems can be distinguished:

- **hard real-time system**: there are *strict* requirements and *critical* processes must be processed within a specified *time*,
- **soft real-time system**: requirements are *less strict*; critical processes are processed with only higher *priority*.

Scheduling (of processes)

- Process scheduling and managing resources optimally are also critical for (real-time) operating systems (**OS / RTOS**).
- Since every system communicates with its environment through some kind of peripherals, it is important to manage the peripherals in a way that meets the requirements of the real-time system: to comply with the response time, the sequence of instructions handling the *event* must be executed. Running the instruction sequence requires resources that must be provided to the OS in order to assign them to *time-critical processes*.
- The following levels of processor scheduling can be distinguished:
  - **Long-term** or work scheduling,
  - **Medium-term** scheduling, and
  - **Short-term** scheduling.

Levels/Terms of scheduling

The OS (kernel) contains a scheduler.

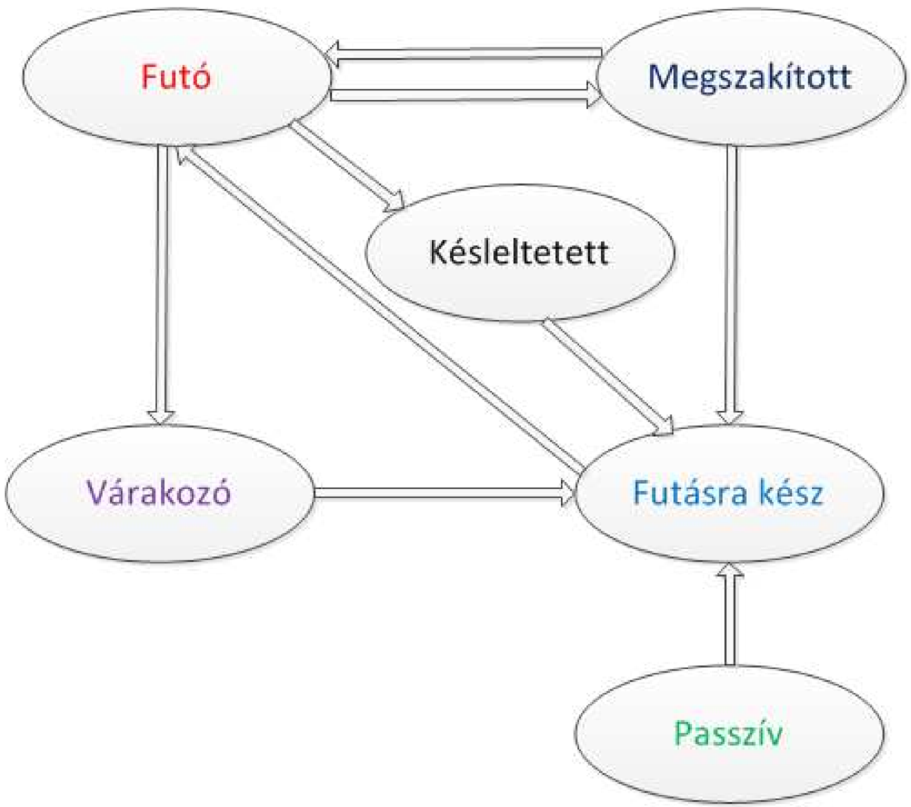
- The task of **long-term scheduling** is to determine which of the pending workloads to start running from a *mass storage*, and when the job is complete, and when select a new job to start. Therefore, the algorithm for long-term scheduling should <u>rarely run</u>.
- **Medium-term scheduling** has called for the elimination of *intermittent load fluctuations* to avoid timeouts for higher loads. The medium-term scheduling algorithm solves this by *suspending* certain (non-time-critical) processes and then *reactivating* as a function of system load. When a process is suspended, the process is stored in the *mass storage*, and the OS can take away resources from the process, which only returns to the suspended process when the process is reactivated.
- The task **of short-term scheduling** is to select which ready-to-run process will get the processor time. The algorithm that performs short-term scheduling runs <u>frequently and quickly</u>, so the OS always keeps the scheduler code in memory.

Scheduling – further definitions

The following basic concepts regarding scheduling and programs can be defined:

- **Task**: Individual subtask.
- **Job**: Small, regularly performed subtasks.
- **Process**: The smallest executable program unit, a particular scheduling entity that is treated by the OS as a standalone program. It has its own (protected) memory space, which is inaccessible by other processes. Tasks can be implemented by processes.
- **Thread**: A scheduling entity without its own memory space, *threads* belonging to the same parent process work on the same memory space.
- **Kernel**: An essential component of the OS that provides task management, scheduling, and communication between tasks. The kernel code is made up of hardware-dependent device (HAL – Hardware Abstraction Layers) and hardware-independent layers.

Task status changes



- **Passive (Dormant):** A passive (dormant) state that may represent a suspended state before-, or during the initialization.
- **Ready to run**: Indicates the Ready state. The priority level of the task is important, as well as the priority level of the currently running task, which is used by the scheduler to decide whether to launch the task.
- **Running**: The task is currently running.
- **Delayed**: This condition occurs when the task is forced to *wait for some time* interval. Usually occurs after a *timer* service call is made.
- **Waiting**: The task is waiting for a specific event. (This is usually some kind of I/O peripheral operation.)
  Interrupted: The task has been interrupted or the interrupt handling routine is interrupting the process (IRQ, INT).

Scheduling algorithms

There are two main types of scheduling algorithms:

- **Cooperative** (= non-preemptive): The operating principle and basic idea is that a given program or process *give up* the processor when it is already running or waiting for an I / O operation. This algorithm works well and efficiently as long as the software is running properly (not being in an infinite loop) and *giving up* the processor. However, if one of the programs / processes does not give up the processor or freezes, it can reduce the stability of the whole system. Therefore, the cooperative algorithm should never occur in real-time embedded OS.
- **Preemptive**: the scheduling algorithm that is part of the OS controls the running of programs / processes. In case of a preemptive multitask, the OS can *take away/stop* the running rights from the processes and give the *running rights* to other processes. Real-time OS schedulers are always preemptive algorithms, so stopping any program or process does not significantly affect system stability.

Communication between tasks

Because the tasks run parallel to each other during OS operation, it must ensure that the same I / O peripheral, resource, or memory area, is not used commonly by two or more tasks at the same time, as this may result in malfunctioning of the system.
The following known methods are available:

- **Mutex** (mutual exclusion) „Locking mechanism” (only the task that locked it can unlock it)
- **Semaphore**: "signaling" mechanism (one task signals the other to finish and take over the resource) ~ 1 bit information
- **Event flags**: they can exchange multiple bits of information.
- **Mailbox**: this can be used to transfer a more complex data structure.
- **Queue**: this is used to transfer content from multiple mailbox arrays.
- **Pipe (FIFO):** it allows direct, continuous (even streaming) communication between two tasks.

(Embedded) Operation Systems - EOS

There are several types of classification possible:

- General purpose or **embedded OS**
- Real-time (time-critical) or non-time-critical OS,
- Open source or licensable OS, etc.

General Purpose Processor's Operating Systems (OS):

- MS-DOS, Linux, Windows, etc.

Real-time Operating Systems (RTOS) for Embedded Processors:

- Linux
- Android
- Micrium uC / OS
- QNX
- RTLinux
- **Windriver VxWorks** (RT)
- Windows Embedded, IoT, etc.

Classification of processors

- By integration:
  - uP / CPU: conventional microprocessors + physically separate memory + external I/O peripheral chips (chipsets)
  - uC / MCU: microcontrollers: integrated on a single chip processor, memory (usually flash) and some I/O peripherals
    - System-on-a-Chip (SoC): A single-chip system
    - Small size and cost, low dissipated power
- By instruction set:
  - RISC vs. non-RISC (CISC) ISA instruction set architectures
- By memory access of Instruction/Data:
  - Von Neumann (Common) vs. Harvard Architectures (Separated)
- Some architecture types include: Intel 8051, ARM, AVR, MicroChip (PIC), MIPS, IBM PowerPC, x86 (32/64), Sun SPARC, etc.

Clarification of definitions

- **FPGA**: **Field-Programmable Gate Arrays =** A programmable circuit consisting of general logic and dedicated resources
  - e.g. Xilinx Artix-7 series
- **SoC: System-on-a-Chip** = Entire system on a single chip
  - All functions (analog, digital, or RF) are integrated on a single chip, rather than using many different devices. This is also true for today's MCU, DSP, ASIC, or FPGA circuits.
- **Zynq** = „Zinc" as an alloying element. Tightly integrates traditional FPGA logic (PL) with processor system (PS) => PL + PS integration!
- **APSoC** (like a Xilinx Zynq): **All Programmable SoC**, which is programmable in all its components.
- A Zynq APSoC chip integrates the following two main parts:
  - Conventional FPGA Logic (**PL**) = Artix-7 FPGA Logic,
  - Processor system (**PS**): ARM-Cortex-A9 cores

Technologies and strategies

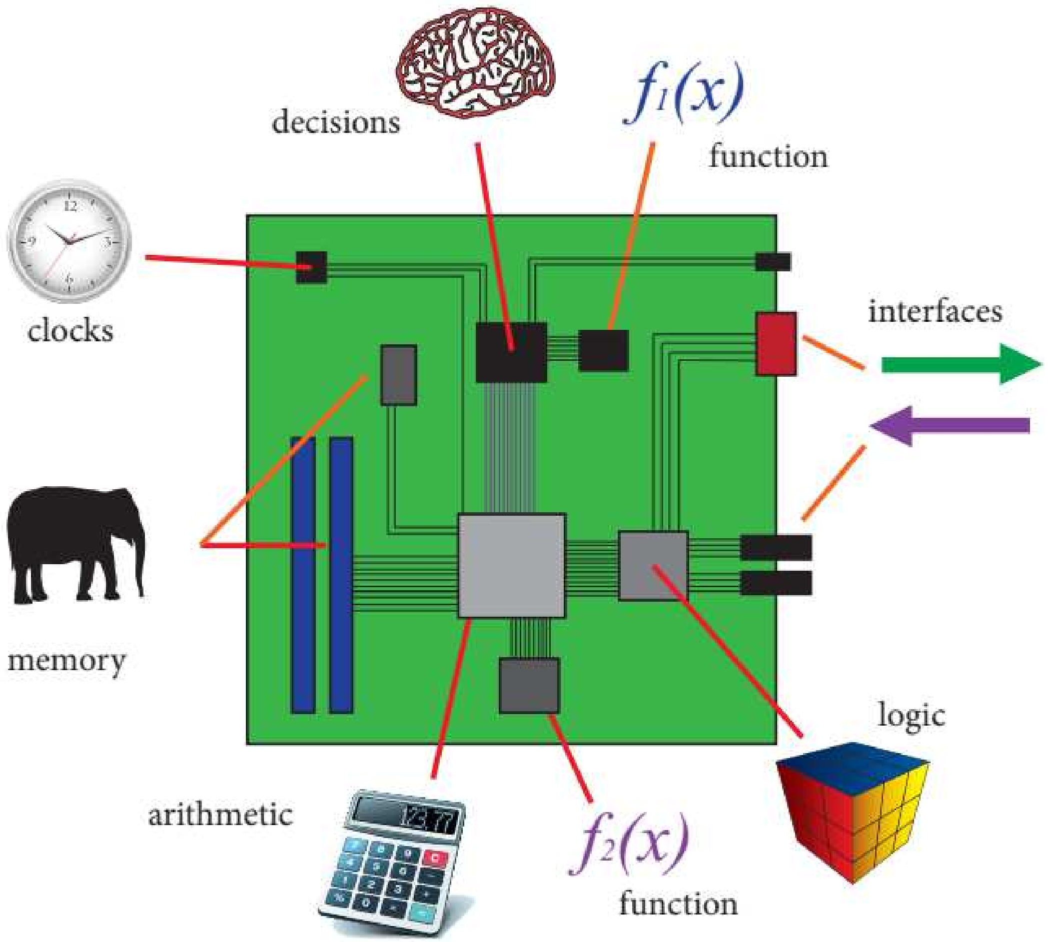
Leading *Technologies* for Design and Implementation of Embedded Systems – classification of processign units like:

- (DSP): Digital Signal Processor based systems
- (MCU): Microcontroller Unit based systems
- (ASIC / ASSP): Application Specific (equipment oriented) Integrated Circuit technology based systems
- (FPGA): Programmable Logic Gate Circuit Technology based systems
- (CPU / MPU / GPU): A microprocessor or graphics processor units
- SoC: System-on-a-Chip: a one-chip system that can integrate the above technologies together!

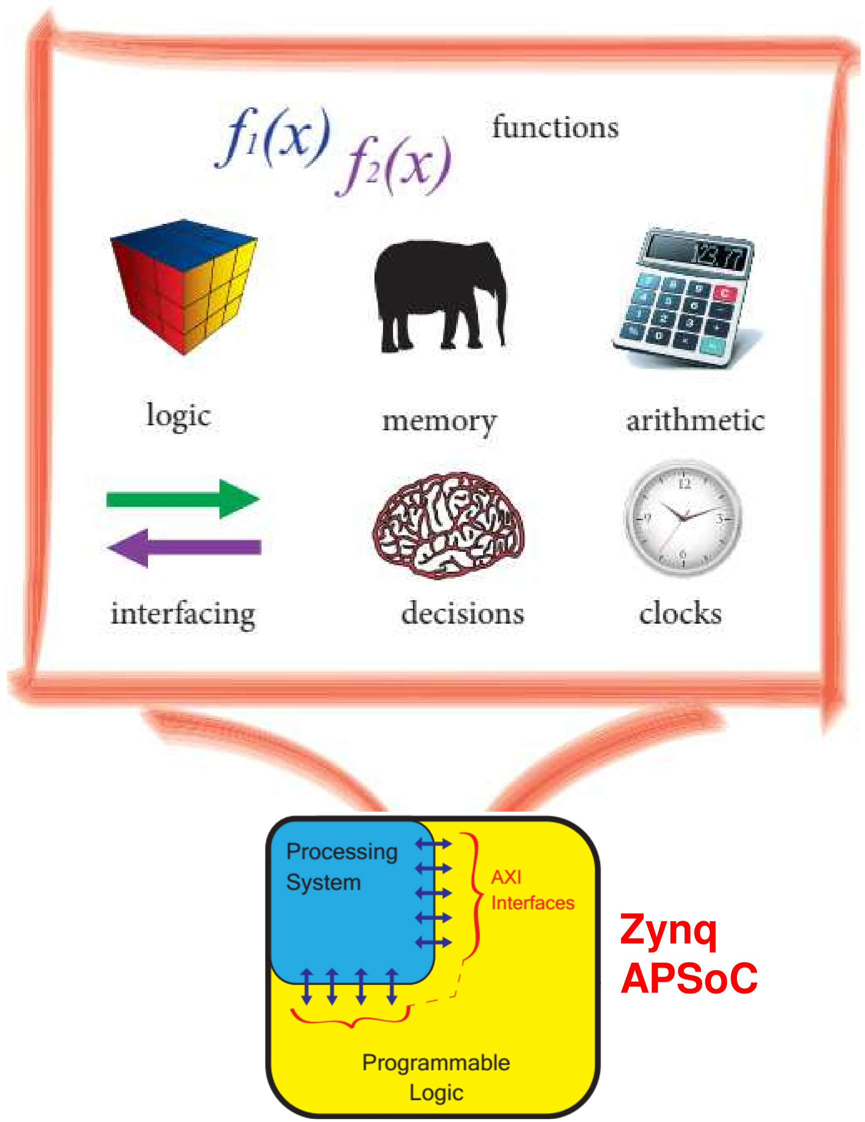
Development strategies:

- HW / SW co-design: collaborative design of HW / SW parts
- HW / SW co-verification: collabborative inspection and testing of HW / SW parts

System-On-a-Board vs. System-On-a-Chip



vs.



Reference: http://www.zynqbook.com/

Typical I/O peripherals

- Asynchronous serial communication interfaces: **RS-232**, RS-422, RS-485, etc.
- Synchronous serial communication interfaces: **I^2C, SPI,** etc.
- Universal Serial bus: **USB**
- Multimedia cards: **(SD) Smart Cards**, (CF) Compact Flash etc.
- Network: Ethernet (1GbE / 10 GbE / 100 GbE)
- Industrial Networks or „Field-bus” protokold: CAN, LIN, PROFIBUS, IO-LINK etc.
- Timer-schedulers: PLL(s), Timers, Counters, Watchdog timers (WDT)
- General Purpose I/Os - **GPIOs**: LEDs, push-buttons, dip switches, LCD displays, etc.
- Analog/Digital – Digital/Analog (ADC/DAC) converters
- Debug ports: **JTAG**, ISP, ICSP, BDM, DP9, etc.

FPGA-based embedded systems

Main design steps for FPGA-base embedded systems:

- FPGA hardware (firmware) design
  - Selecting Embed**dable** / embed**ded** processor:
    - Licensable Soft-core: PicoBlaze / MicroBlaze™ / ARM-M0-3 (Xilinx); Nios II™ (Altera),
    - Licensable Hard-core: IBM PowerPC® (Xilinx), ARM® (Xilinx / Altera),…
    - Open source processor cores: e.g. www.opencores.org
  - Selecting Programmable Peripherals (see topics from *Development platforms* or *Embedded Peripherals*),
  - Generate device drivers and SW libraries
    - **BSP**: **B**oard **Su**pport **P**ackages = Domains
  - Application Development:
    - Software routines (API),
    - Interrupt handling routines,
    - Operating systems, real-time operating system

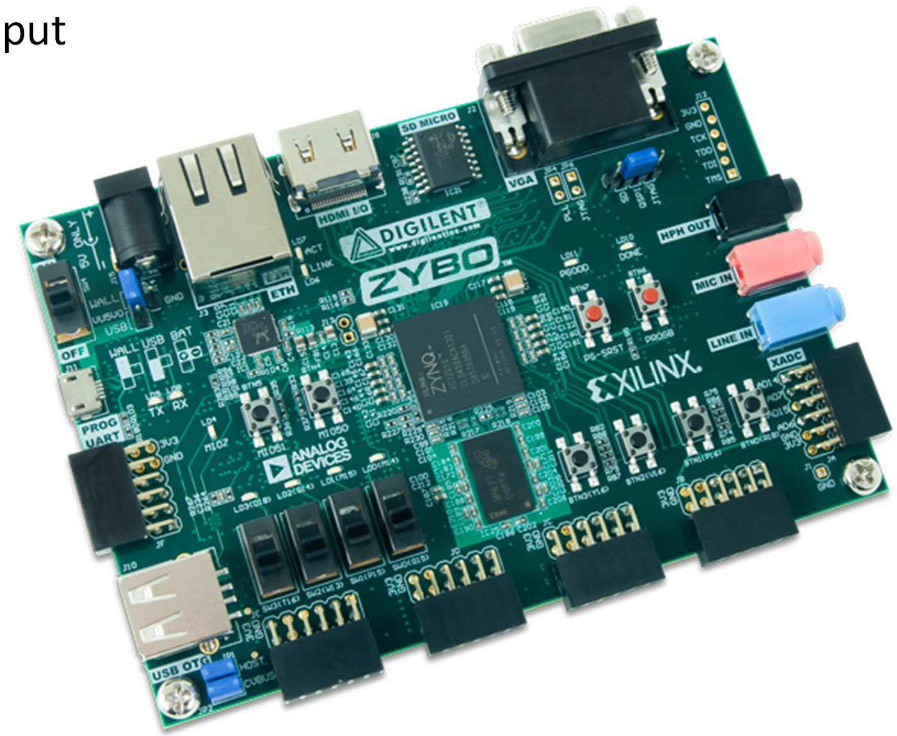
DIGILENT ZYBO + PMOD

Short introduction about development hardwares

Digilent ZyBo development platform

ZYBO™ Zynq FPGA/APSoC development kit

- *Xilinx Zynq-7000 (Z-7010)*
  - 650 MHz dual ARM Cortex-A9 cores (PS)
  - 8-channel DMA controller (PS)
  - 1G Ethernet, I2C, SPI, USB-OTG controller (PS)
  - Artix-7 FPGA logic (PL)
  - 28K logic cell, 240 Kbyte BRAM, 80 DSP multiplier (PL)
  - 12-bits, 1MSPS XADC (PL)
- 512 Mbyte DDR3 x32-bit (databus), 1050Mbps throughput
- Tri-mode 10/100/1000 Ethernet PHY
- HDMI port: Dual role (source/sink)
- VGA port: 16-bit
- uSD card: storing OS file system
- OTG USB 2.0 (host and device)
- Audio codec
- 128Mbit x Serial Flash/QSPI (storing configuration)
- JTAG-USB programmability, UART-USB controller
- GPIO: 4+1 LED, 4+2 push buttons, 4 dip switches
- 4+2 PMOD connector (+A/D)



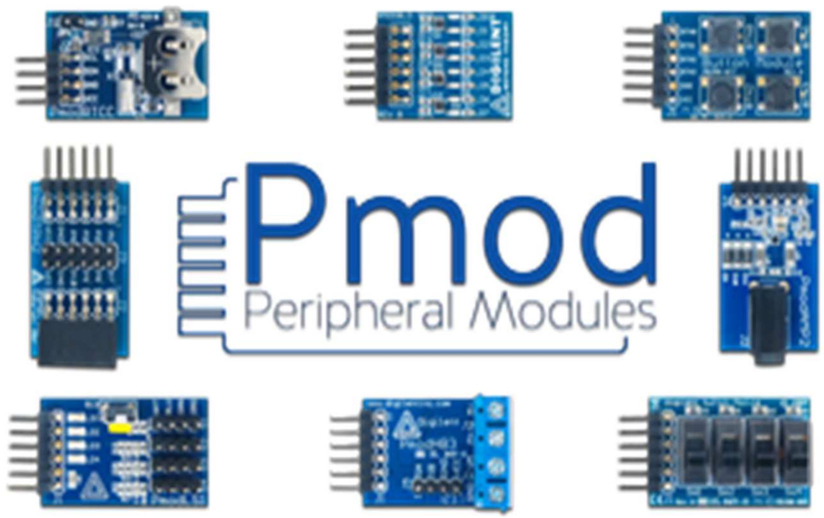
📖 Webpage: https://store.digilentinc.com/zybo-zynq-7000-arm-fpga-soc-trainer-board/

Expandability - PMODs

- Digilent Peripheral modules (PMOD), further expandability:
  - character LCD, OLED, 7segLEG
  - GPS, WiFi, Bluetooth,
  - Ethernet IF, USB-UART, RS232
  - Joystick, Rotary Enc., Switches,
  - SD Card, Serial Flash,
  - A/D, D/A converters, H-bridges
  - Accelerometer, Gyroscope,
  - Temperature sensor, ...stb.

  OR

  - „3rd party” solutions, development and adaptation of custom-designed expansion/ add-on cards.



📖 Webpage: https://store.digilentinc.com/pmod-modules-connectors/

DEVELOPMENT TOOLS

Xilinx Vivado / VITIS 2020.1

SZÉCHENYI 2020

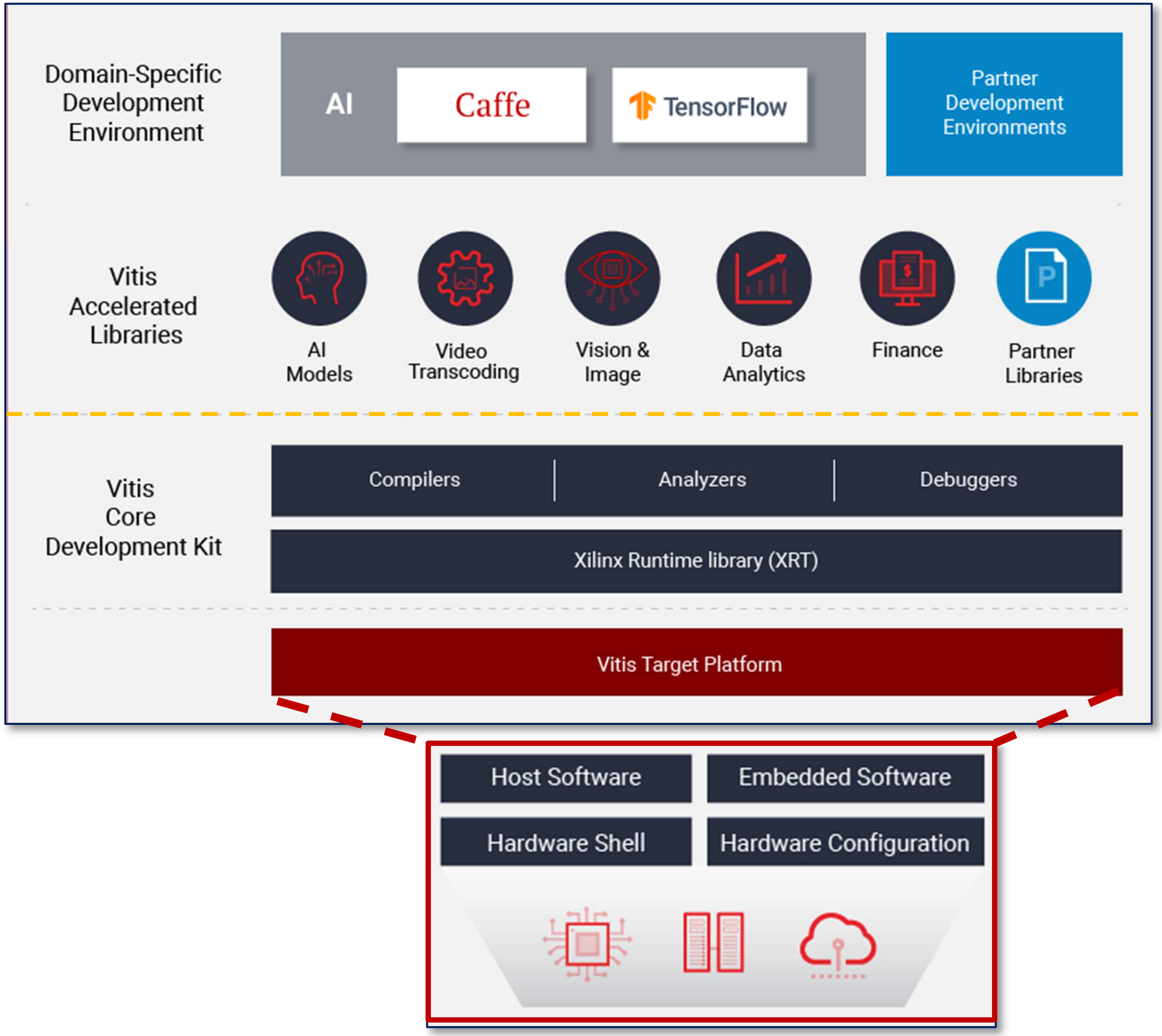
MAGYARORSZÁG KORMÁNYA

Európai Unió
Európai Strukturális és Beruházási Alapok

BEFEKTETÉS A JÖVŐBE

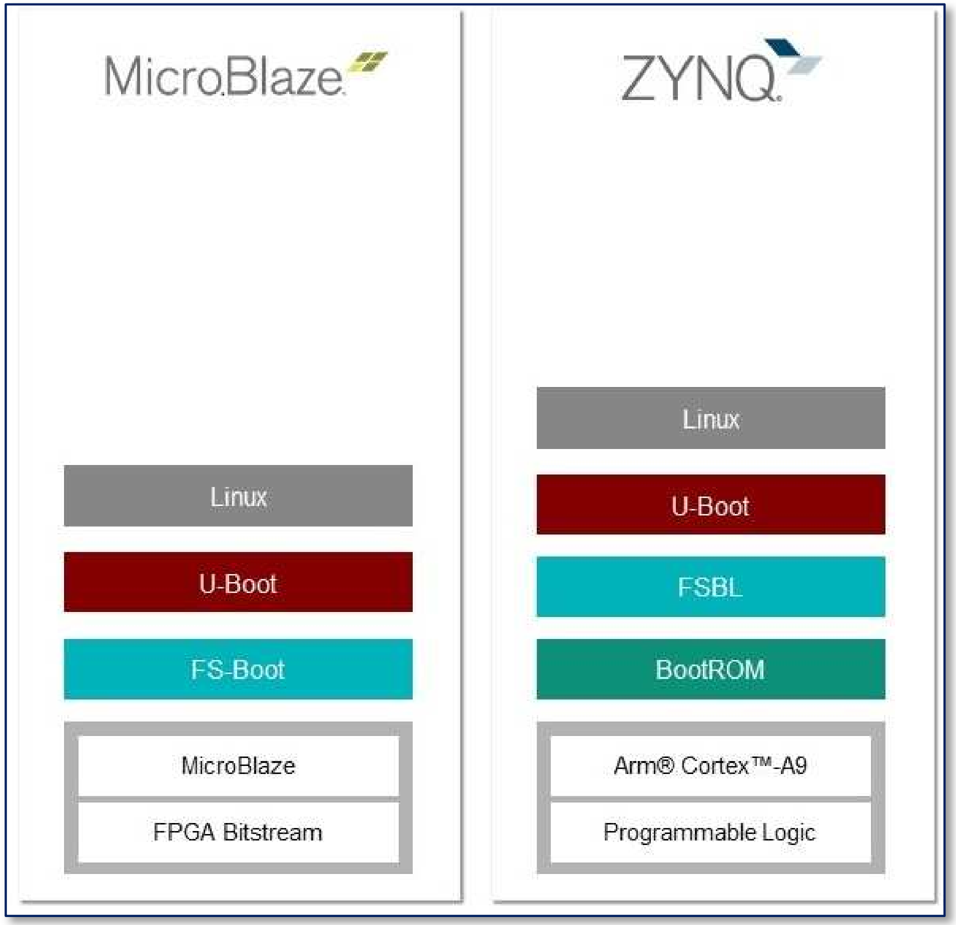
Development tools

- Xilinx VITIS – Unified software platform



- Comprehensive VITIS core development kit to seamlessly build accelerated/embedded applications
- Rich set of hardware-accelerated open-source libraries optimized for Xilinx hardware platforms
- Plug-in domain-specific development environments enabling development directly in familiar, higher-level frameworks
- A growing ecosystem of hardware-accelerated partner libraries and pre-built applications
- **VITIS Target Platform:**
  - Xilinx embedded devices, operating system, boot loader and drivers, root file system.
  - Predefined target platforms /or *own Xilinx based platforms (defined by Vivado Design Suite)*

Embedded Software Infrastructure



- Creation of embedded system using Xilinx Series-7, or Xilinx Zynq® SoC/MPSoC devices,
  - **ARM Cortex-A9**/A53/A57/A72 hard processor cores
  - MicroBlaze™ soft processor cores, and
  - ARM Cortex-M1/M3 micro controllers (soft cores)
- Includes open source operating systems and bare metal drivers,
- Multiple runtimes and Multi-OS environments,
- Compilers, debuggers, and profiling tools.

EFOP-3.4.3-16-2016-00009

A felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása a Pannon Egyetemen

THANK YOU FOR YOUR KIND ATTENTION!

SZÉCHENYI 2020

MAGYARORSZÁG KORMÁNYA

Európai Unió
Európai Szociális Alap

BEFEKTETÉS A JÖVŐBE