Implementing and profiling collaborative CPU-FPGA projects with real-time requirements

Alexander Wirthmüller
aw@mpsitechnologies.com

Introduction

About me

- Based in Munich
- Diploma in Electrical Engineering

- Founder and Director at MPSI Technologies
- MPSI Technologies: make Embedded Software development more pleasant – by replacing repetitive tasks with model-based source code generation

- Senior Staff Engineer with Symeo / indie Semiconductor (industrial radar)

Introduction

Scope

- CPU-based configuration (e.g. set FPGA IP core parameters once on start-up) & monitoring (e.g. accumulate throughput statistics)
  - Few updates per second / timing not critical
  - FPGA subsystem performs functionality even when left alone by the CPU

  → Not the topic of this talk

- Here: focus on functionality which requires continuous CPU-FPGA interaction
  - Sometimes with Realtime (RT) requirements

CPU-FPGA Collaboration

Why do it | How to do it

- Vast availability of specialized third-party libraries for CPU
- Significantly less effort for sequential C/C++ code as compared to RTL: coding, debugging, maintenance
- On-system information availability / distribution: e.g. in industrial context by default via Ethernet and CPU-side middleware
- FPGA's are slow (typ. 200 MHz with tuned sections > 400 MHz)
- Few [sequential] algorithms are *really* suitable for high-level synthesis (HLS) or should be tackled by HLS

CPU-FPGA Collaboration

Why do it | How to do it

- Two options in FPGA-SoC's
  - FPGA subsystem as memory-mapped peripheral CPU address space
  - Shared section of DDR memory (with of without formal DMA functionality)

- By extension: standalone FPGA system as PCIe peripheral of CPU host

Brief #1: Network packet scheduling

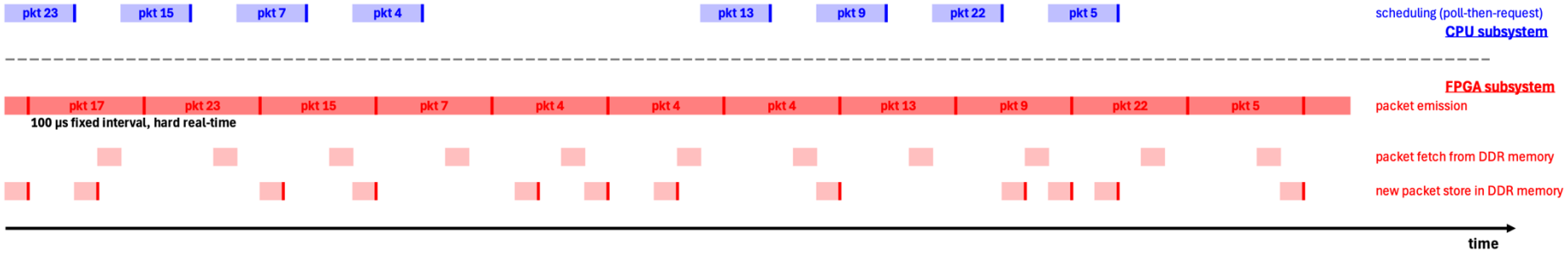
Application | Algorithm (simplified)

- FPGA accepts Ethernet frames, stores many frames in few-kB packets in DDR memory (1000+ slots)
- FPGA notifies CPU of stored packet (slot + ID)
- FPGA sends one packet (+ forward error correction) every 100 µs [hard realtime] via Gigabit transceiver and optical fiber
- CPU should ideally make one “send” decision per 100 µs and communicate it to FPGA
- Without decision, FPGA re-sends previous packet

- Implementation: Linux host on MPSoC Cortex-A53 polling updates via AXI lite; DDR memory access (reserved region) exclusive from FPGA

Brief #1: Network packet scheduling

Application | Algorithm (simplified)



Brief #2: Radar cube processing

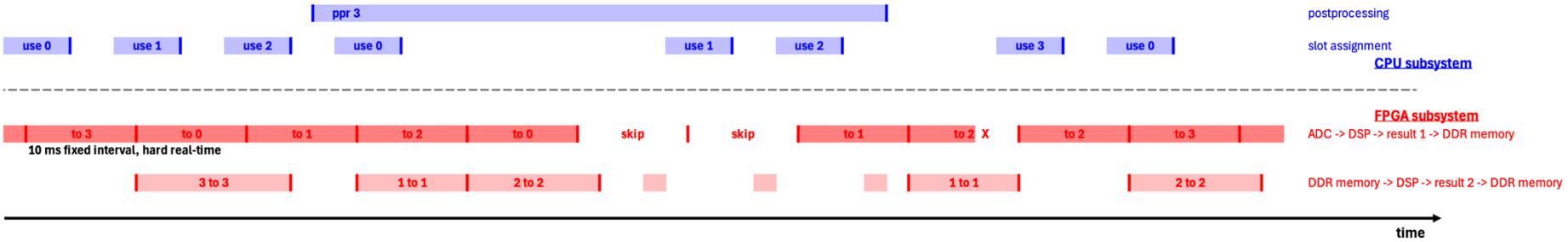
Application | Algorithm (simplified)

- CPU assigns DDR memory slot to FPGA for the next frame
- FPGA accepts and time-stamps high-bandwidth multi-channel ADC data, does initial DSP, stores result (1) in DDR memory [hard realtime]
- Two failure points: a. no slot assigned => frame is skipped; b. buffer overflow writing to DDR memory => frame is skipped, CPU is notified
- FPGA continues to do DSP within assigned slot and processes result (1) into result (2)
- FPGA notifies CPU of completion, along with initially assigned time-stamp
- CPU post-processes result (2) with varying degree of time consumed

- Implementation: Linux host on MPSoC Cortex-A53 polling updates via AXI lite; shared DDR memory section

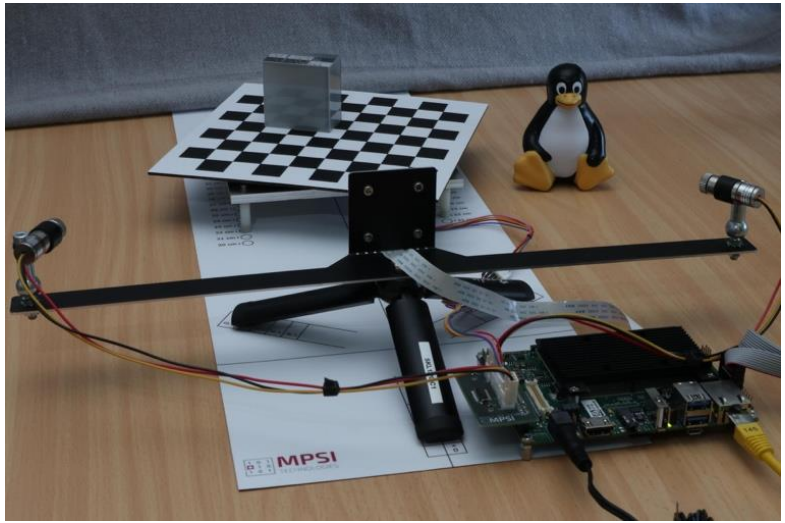
Brief #2: Radar cube processing

Application | Algorithm (simplified)



Example #3: Vision-to-rotary table PI controller

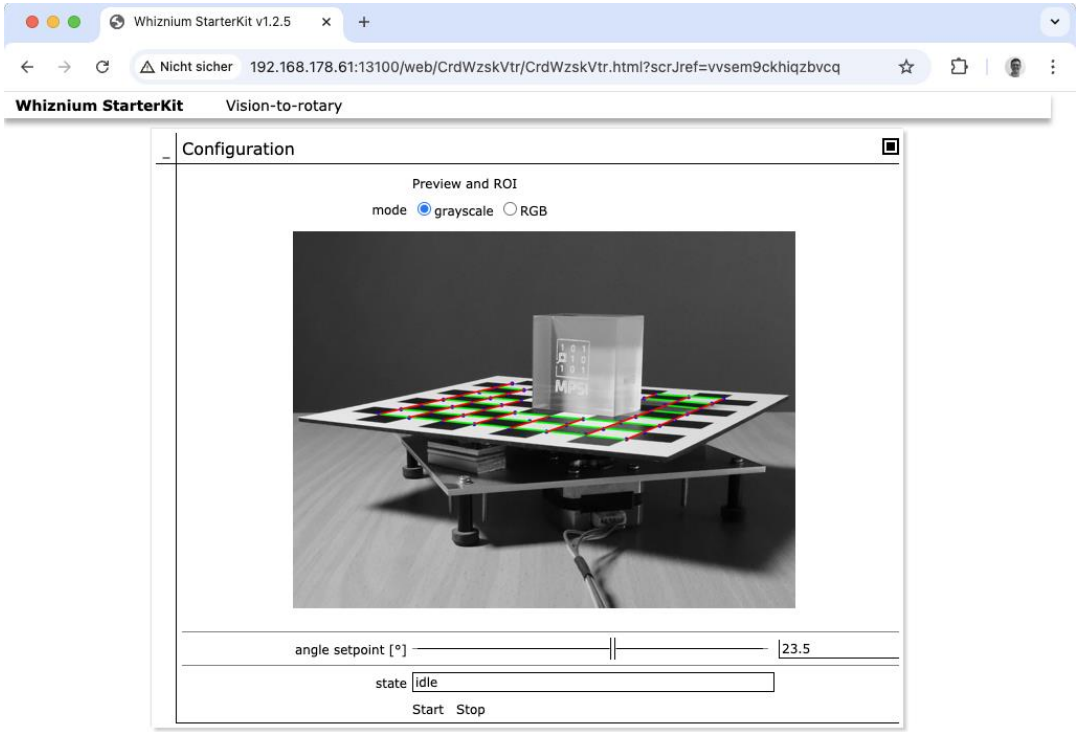
- Hardware: tabletop 3D laser scanner; ZUBoard (AMD MPSoC CG1), rotary table with stepper motor, IMX335 5MP MIPI camera, [line lasers]
- Objective: closed loop control of stepper motor with detection of rotary angle using FPGA-based computer vision



Example #3: Vision-to-rotary table PI controller

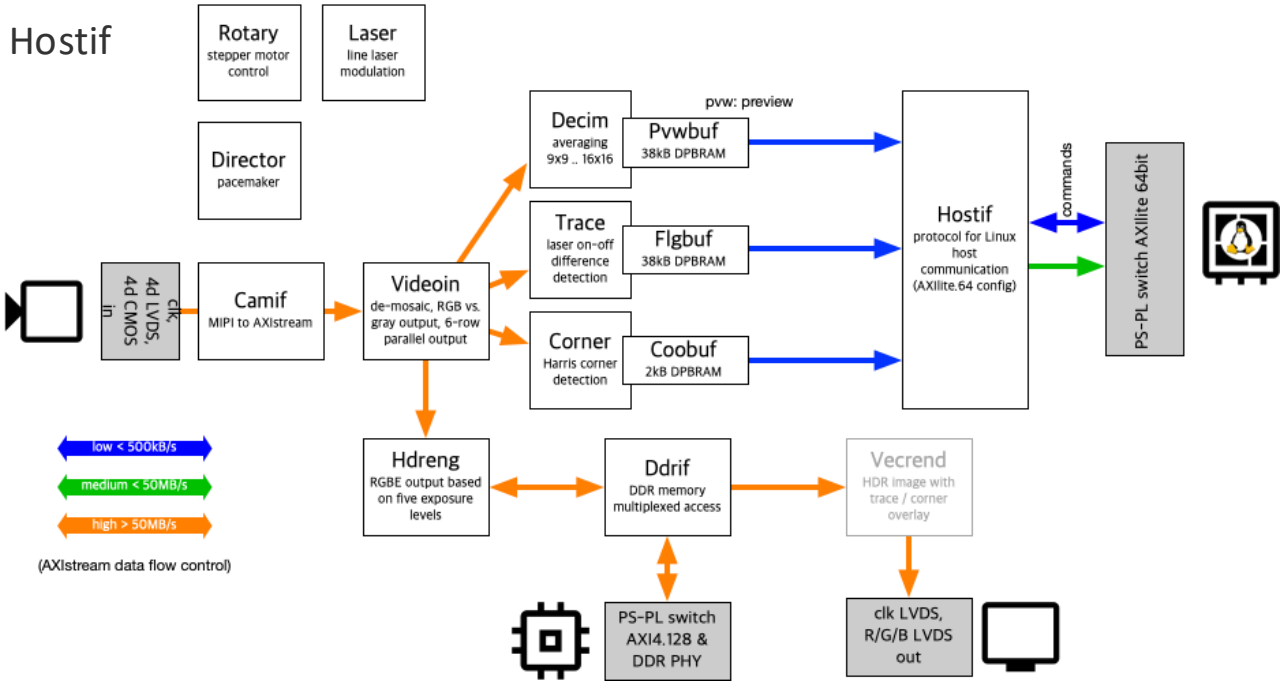
Introduction | FPGA design | Feedback loop | Results

- CPU-based angle determination used to be OpenCV based, now is custom



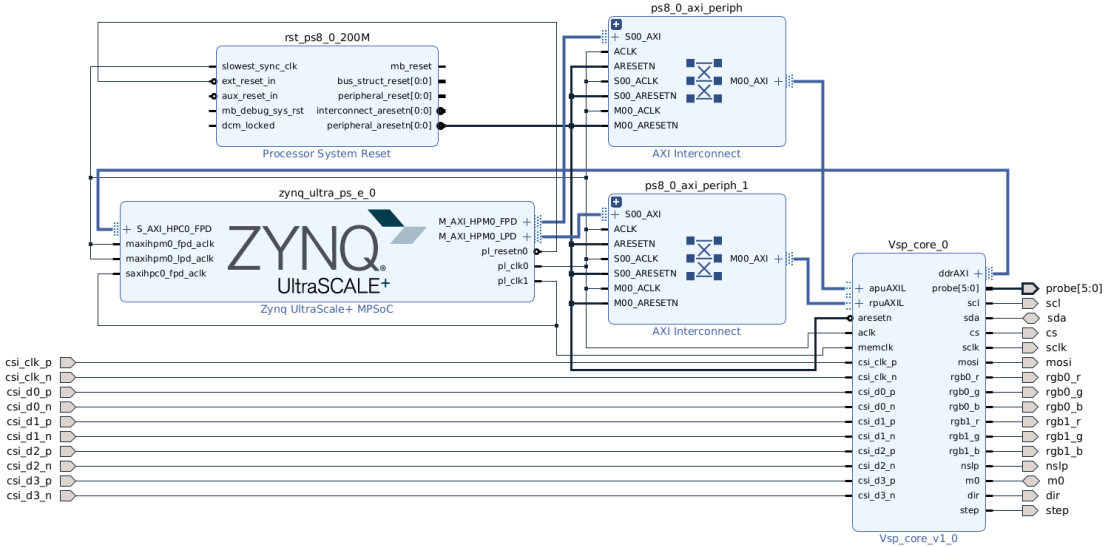
Example #3: Vision-to-rotary table PI controller

- Sensor
  Camif -> Videoin -> Corner -> Hostif
- Actuation
  Hostif -> Rotary



Example #3: Vision-to-rotary table PI controller

- AXI lite/64 for Cortex-A53 CPU-triggered status polling, corner coordinate buffer read-out & actuator setpoint feedback
- Secondary AXI lite/32 co-host interface towards Cortex-R5 present but not used
- AXI full/128 interface to access shared DDR memory for other part of project (HDR imaging)

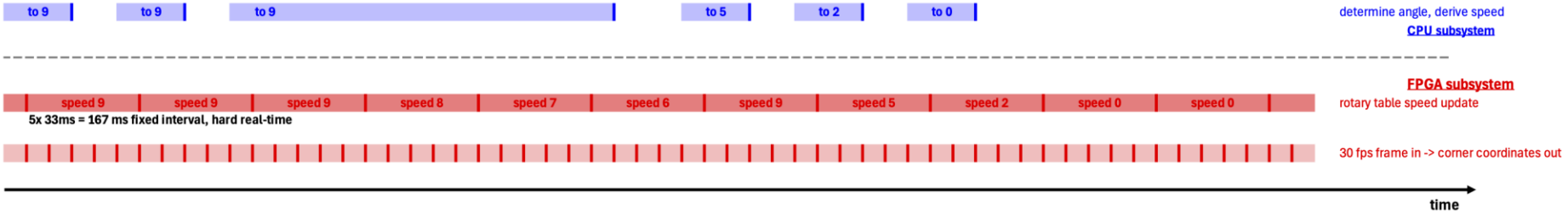


Example #3: Vision-to-rotary table PI controller

- MIPI sensor delivers image frames at 30 fps, FPGA-based Harris corner detection algorithm (fully pipelined) matches this pace with “zero latency”; no DDR memory is involved
- FPGA informs CPU of finalized coordinate buffer
- CPU locks coordinate buffer (inhibiting corner detection) and runs its portion of algorithm, then unlocks

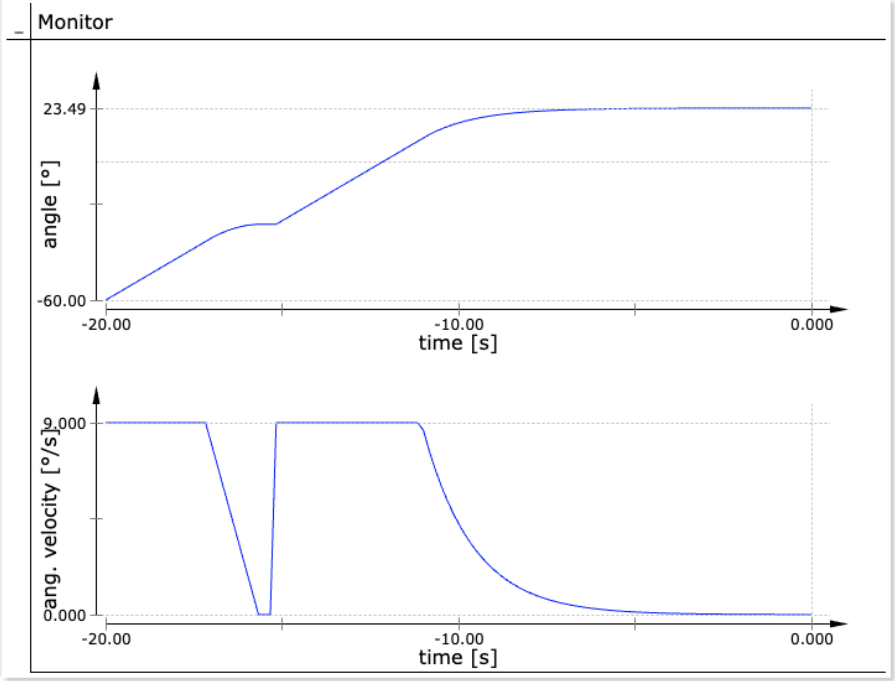
- In parallel, FPGA drives stepper motor and expects one variable update from CPU every 5th frame [hard realtime] (166 ms update interval; variable is angular velocity)
- If no input, FPGA throttles speed stepwise, down to zero

Example #3: Vision-to-rotary table PI controller



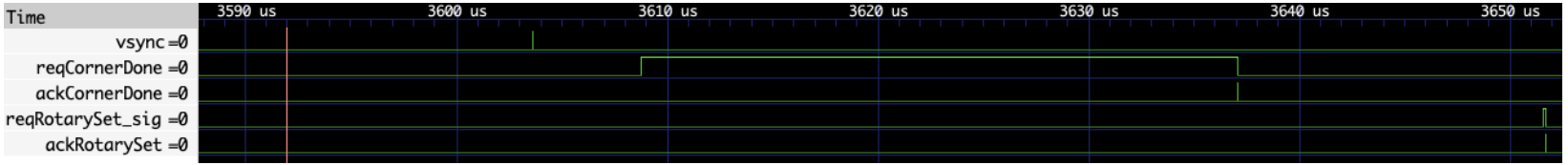
Example #3: Vision-to-rotary table PI controller

- Example going from -60° to +23.5° with 11 missed update intervals



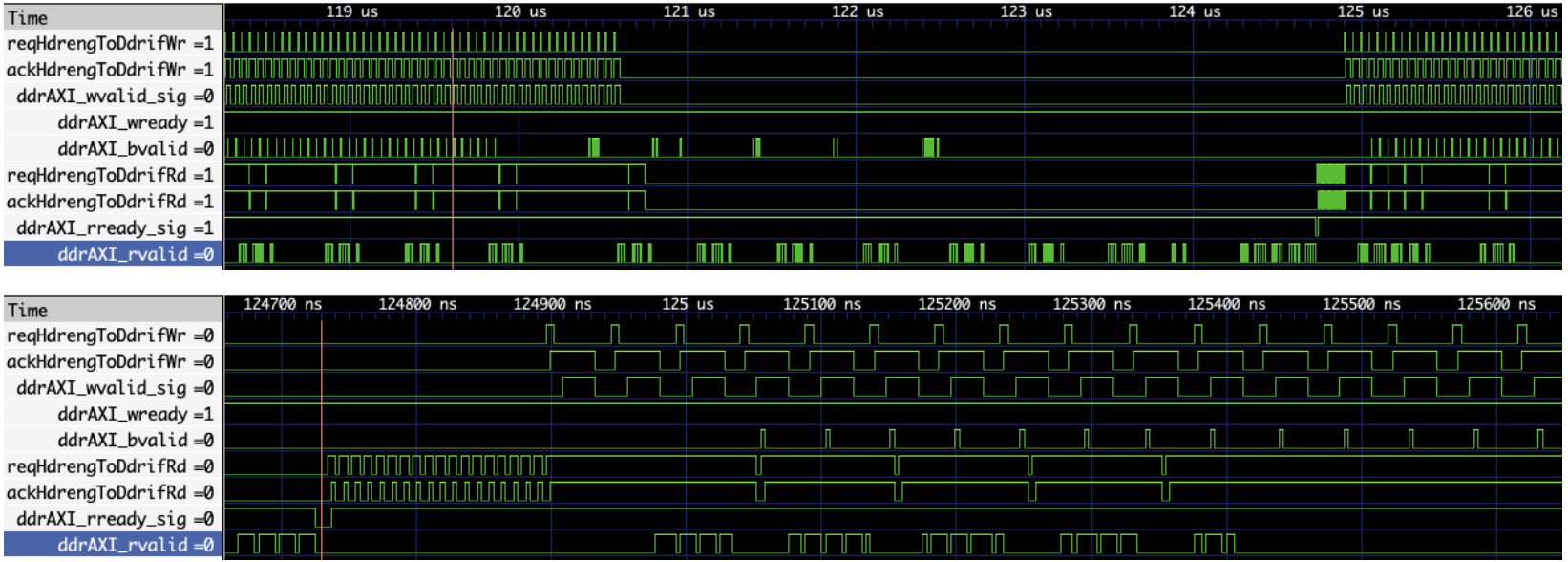
Example #3: Vision-to-rotary table PI controller

- FPGA-triggered handshake req/ackCornerDone (to be acknowledged by CPU) takes random time (here 27 µs) for ~10 AXIL io{read/write}64()'s from Linux … despite of 200 MHz PS-PL clock
- CPU-based processing and response reqRotarySet takes additional 14 µs
- Tuned version (minimizing io{read/write}64()'s) around 1-2 µs min. latency (on MPSoC!)



Example #3: Vision-to-rotary table PI controller

- Observation of non-deterministic DDR memory read/write operations via AXI full
- Write is relatively stable but read cycles are scattered => strictly observe / utilize full AXI capabilities including posting many read addresses without waiting for first read data



Key takeaways (1/2)

- The FPGA subsystem is a formidable real-time processor
  - Clock-accurate repetition rate / control loop time constant
  - Multiple parallel processes that don't interfere with one another

- Thus, the CPU subsystem can afford relaxed real-time constraints ...
  - ... if the FPGA subsystem monitors glitches
  - ... and has mitigation strategies in place

Key takeaways (2/2)

- In preceding examples
  - #1 re-send previous data packet in (real-timed) transmission slot missed by CPU
  - #2a identify buffer overflows writing to DDR memory and void current ADC data's frame
  - #2b use multiple DDR memory slots so the CPU can lock one while the FPGA operates on others
  - #3 implement "dead man's switch" in PI controller if CPU is not responsive

The case for model-based CPU-FPGA software co-design

- Live-tracking (trigger in web UI, output to .vcd) of FPGA signals (vendor agnostic)
- Zero-effort preferences page

The case for model-based CPU-FPGA software co-design

- Live-tracking (trigger agnostic)
- Zero-effort preferen

The case for model-based CPU-FPGA software co-design

- Live-tracking (trigger in web UI, output to .vcd) of FPGA signals (vendor agnostic)
- Zero-effort preferences page
- Interactive terminal (web UI or command line)

The case for model-based CPU-FPGA software co-design

- Live-trac
- Zero-effc
- Interacti

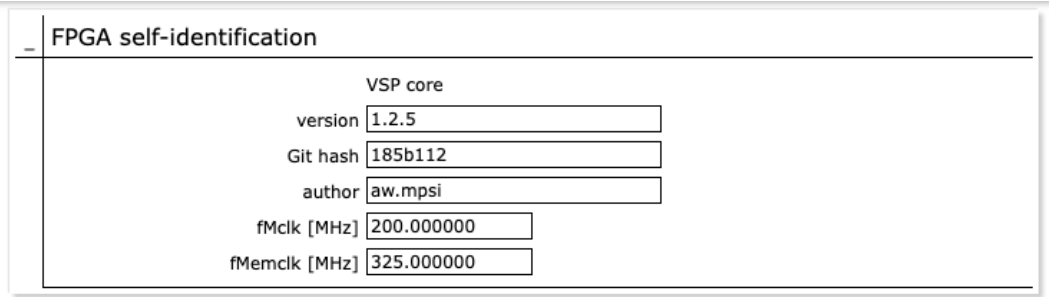
c)

The case for model-based CPU-FPGA software co-design

- Live-tracking (trigger in web UI, output to .vcd) of FPGA signals (vendor agnostic)
- Zero-effort preferences page
- Interactive terminal (web UI or command line)
- Single source of truth for CPU C++ library <- AXI Lite -> FPGA RTL decoder; command set and bulk data transfers per FPGA sub-module
- Enforced CPU-FPGA version compatibility check

The case for model-based CPU-FPGA software co-design

- Live-tracking (trigger in web UI, output to .vcd) of FPGA signals (vendor agnostic)
- Zero-effort preferences page
- Interactive termina
- Single source of tru r; command set and bulk data trans
- Enforced CPU-FPG

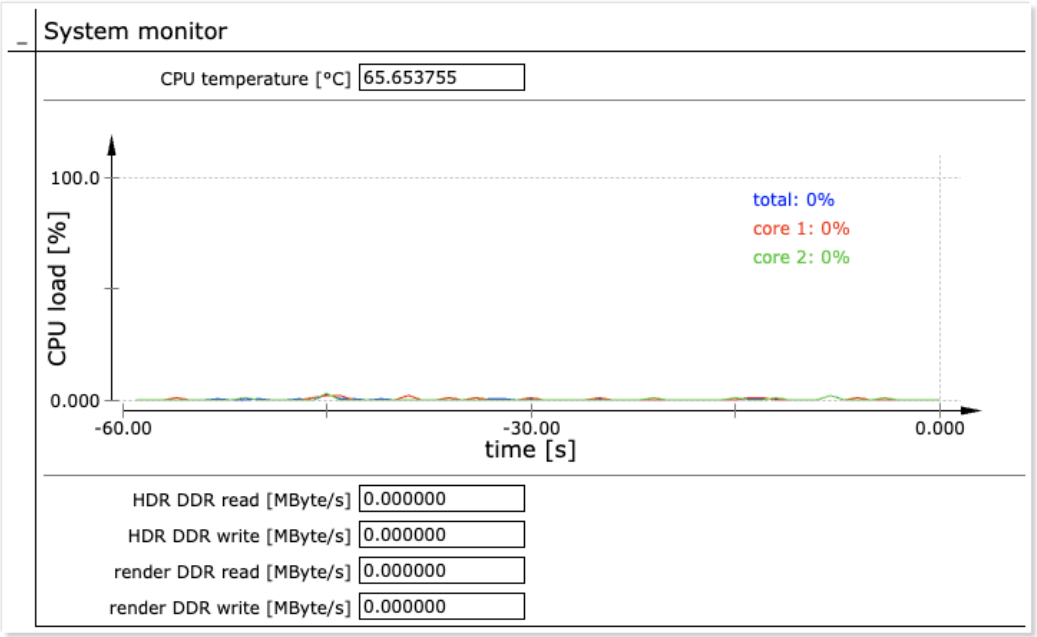


The case for model-based CPU-FPGA software co-design

- Live-tracking (trigger in web UI, output to .vcd) of FPGA signals (vendor agnostic)
- Zero-effort preferences page
- Interactive terminal (web UI or command line)
- Single source of truth for CPU C++ library <- AXI Lite -> FPGA RTL decoder; command set and bulk data transfers per FPGA sub-module
- Enforced CPU-FPGA version compatibility check
- Easy FPGA subsystem status probing (e.g. DDR memory read/write bandwidth) … with display in web UI

The case for model-based CPU-FPGA software co-design

- Live-tracking (trigg
- Zero-effort prefere
- Interactive termina
- Single source of tru
  and bulk data trans
- Enforced CPU-FPG
- Easy FPGA subsyst
  display in web UI



gnostic)

r; command set

width) … with

The Whiznium Universe & Developer Lifestyle

Concept | Key features | Workflow | Starter kit

WhizniumDBE (“Device Builder’s Edition”) is

- NOT high-level synthesis (HLS), not a compiler
- NOT your typical generator framework
- NOT a visual design tool

The Whiznium Universe & Developer Lifestyle

Concept | Key features | Workflow | Starter kit

WhizniumDBE (“Device Builder’s Edition”) is

- NOT
- NOT
- NOT

WhizniumDBE is

… a user-extensible framework written in C++, that for a given RTL design

… interprets its structure and features, specified in text-based model files

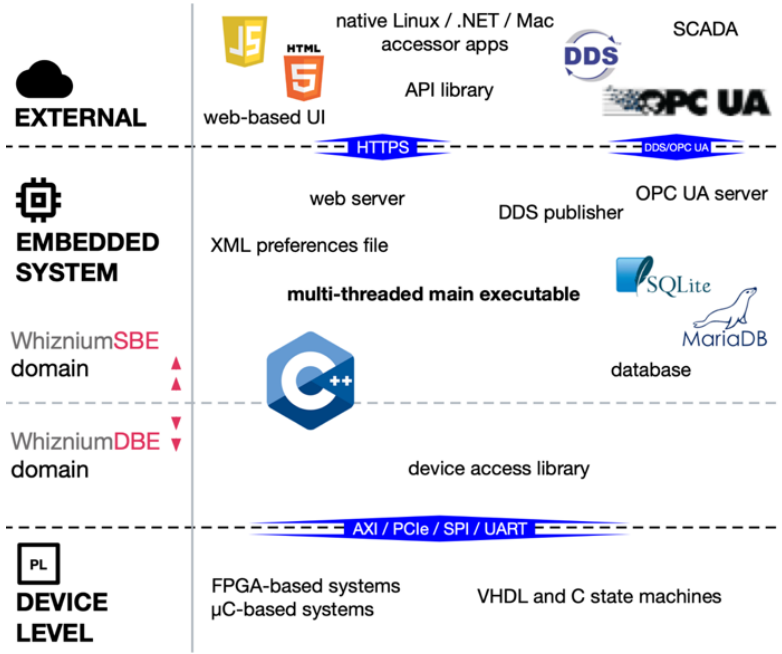
… composes and maintains a fine-grained RTL model *) in a SQL database

… then is able to write VHDL and C++ code based on it

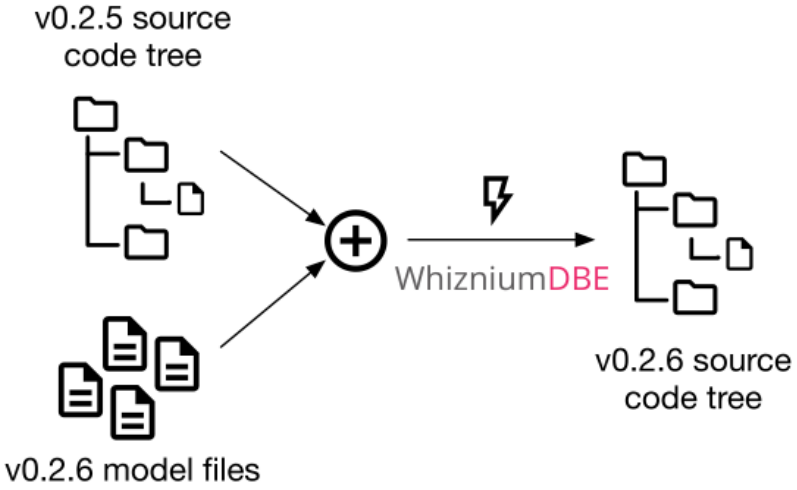
… taking into account manual code contributions of previous design versions

*) from hierarchical structure down to FSM’s incl. state transitions, CDC fabric, generics/ports/signals/variables

- Clean, ergonomic, source code structure (“tasteful naming conventions”, etc.)
- Parametrized templates for standard components (e.g. SPI, GPIO, CRC, Git-Ident; 35 and counting)
- Custom templates can interact with the model / module surroundings while a design is composed (not just simple files with placeholders)
- The applicable vendor(‘s primitives) can be an auto-derived template parameter
- Scope extends beyond the FPGA world with **WhizniumSBE** (Service Builder’s Edition)



- Regular RTL workflow (including use of vendor IDE's) augmented by "source code tree iteration"



The Whiznium Universe & Developer Lifestyle

Concept | Key features | Workflow | Starter kit RE-LAUNCH

- Whiznium onboarding vehicle but also teach [CPU+]FPGA best practices
- Re-launch (three platforms) with live demo at FPGA Horizons (London) in October

FPGA basics / topics covered

- ✓ clean modular project implementation
- ✓ vendor neutral where possible
- ✓ supervision by (Embedded) Linux
- ✓ FPGA-exclusive features: pipelined processing
- ✓ use of standard FPGA building blocks (DPRAM / ping-pong buffers, DSP, I/O)
- ✓ use of at least one advanced interface (MIPI, DDR, …)
- ✓ >1 clock domain and clock domain crossings

FPGA-based vision / topics covered

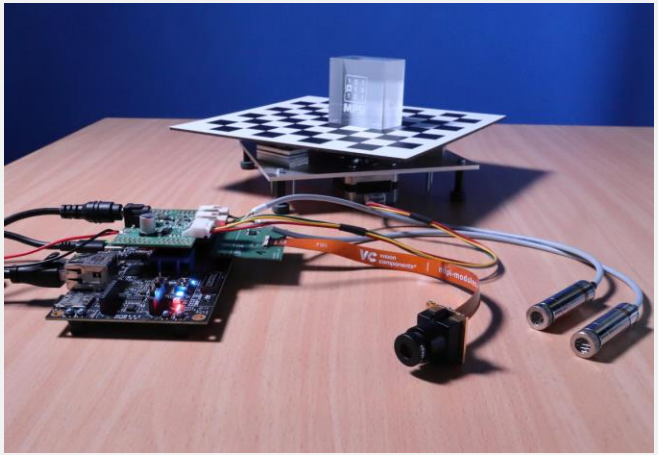
- ✓ de-mosaic
- ✓ pixel bit re-packaging
- ✓ Decimation / averaging
- ✓ HDR frame acquisition
- ✓ classical feature detection (Harris corner detector, laser on-off trace)
- ✓ TBD: machine learning feature

The Whiznium Universe & Developer Lifestyle

Concept | Key features | Workflow | Starter kit RE-LAUNCH

Avnet ZUBoard (AMD)

Zynq UltraScale+, Yocto on ARMv8 x2, 1GB DDR



PolarFire SoC discovery kit (Microchip)

Yocto on RISC-V x4, 1GB DDR

Titanium Ti180 dev kit (Efinix)

Buildroot on soft RISC-V x4, 256MB DDR

Conclusion & Outlook

- Collaborative CPU-FPGA projects
  - When partitioning, find good balance between CPU simplicity and FPGA performance
  - Make no performance assumptions: profile behavior of actual application at runtime
  - Leverage RT capabilities of FPGA to ease RT requirement of CPU

- Use model-based design provided by the OSS Whiznium tools
  - Have single source of truth for CPU and FPGA portions of your projects
  - Avoid vendor lock-in
  - Benefit from JTAG-free live probing in web UI

Resources

- Code for Example #3
  https://github.com/mpsitech/wzsk-Whiznium-StarterKit
  https://github.com/mpsitech/wskd-Whiznium-StarterKit-Device

- Both Whiznium tools are available free of charge on GitHub, including installation instructions
  https://github.com/mpsitech/The-Whiznium-Documentation
- WhizniumSBE reference (NEW)
  https://mpsitech.github.io/The-WhizniumSBE-Reference
- WhizniumDBE reference (NEW)
  https://mpsitech.github.io/The-WhizniumDBE-Reference

- Some more presentations on the topic
  https://www.mpsitech.com/documentation/presentations

Thank You!

Questions?

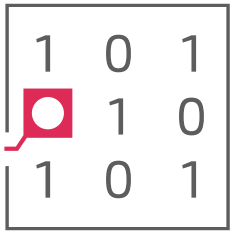
Also, feel free to connect.

- https://www.linkedin.com/in/wirthmua
- https://github.com/mpsitech

Alexander Wirthmüller
Founder & Director

Phone: +49 (89) 4524 3826
Mobile: +49 (175) 918 5480
E-Mail: aw@mpsitech.com

MPSI Technologies GmbH
Agnes-Pockels-Bogen 1
80992 Munich
Germany
www.mpsitech.com



MPSI TECHNOLOGIES