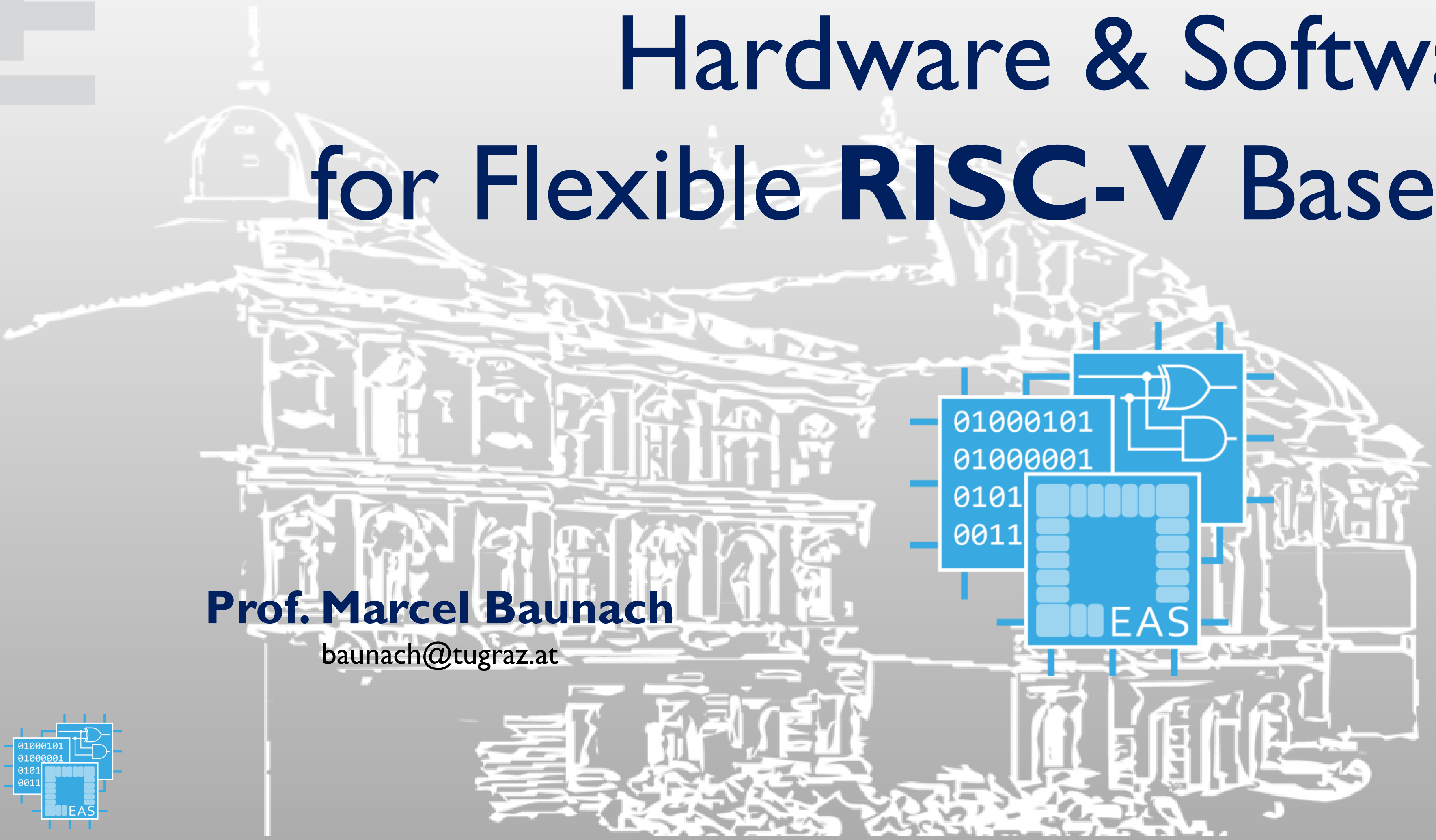


Hardware & Software for Flexible **RISC-V** Based Systems

Prof. Marcel Baunach
baunach@tugraz.at

Institute of Technical Informatics
Embedded Architectures & Systems Group
Graz University of Technology

Flexible Embedded Systems!?

Increasing Number of Systems

Bigger Infrastructures
More Services
Performance Demands
Industry / Consumer

50B devices in 2026

Add new devices!

When to stop?

Quickly changing Requirements

More Features
Improved Algorithms
New Legislation
Updates & Fixes

Obsolete Hardware

Change the software!

What if HW is the limit?

A ? B

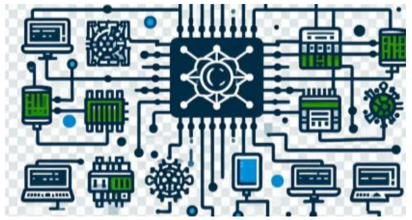
Replace existing Devices

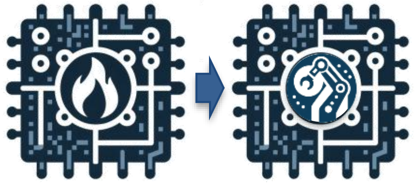
Sometimes difficult.
Time consuming.
Expensive.
Wasteful.

5-20kg CO2 per IC

Change and Reuse *flexible* HW & SW

Easier.
Faster.
Cheaper.
Sustainable.





Is it really "easier"? What are the challenges?
What is possible? Where are the limits?

?

Challenges in Flexible Embedded Systems Design…

Application Software

System Software

MCU / CPU

Electronics

Teaching / Education

- Dynamic Composition of Software Modules
- Verification & Portability for new and changing Architectures
- ASIC/FPGA Designs
- Partial Reconfiguration
- Electronics Generation
- Dynamic Power Analysis
- RTOS Design
- MCU Design

CoEVOLVE

CompEAS

Projects

EAS

Pro²Future

Elektrobit

Infineon

AUMOVIO

SmartOS
Sustainable modular adaptive real-time Operating System

HADES-V

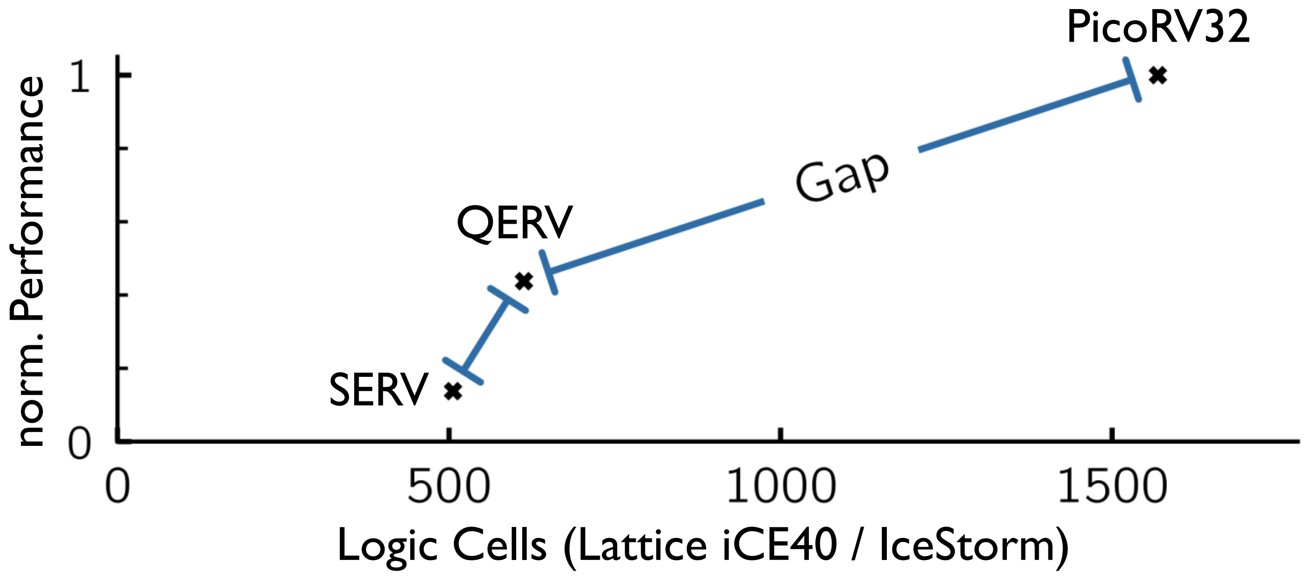
FazyRV: A scalable RISC-V core

Yet another RISC-V core?

Closes the Gap between 32-Bit and Bit-Serial.

➔ Goal: Area vs. Performance trading

➔ ISA: RV32I user specification compliant

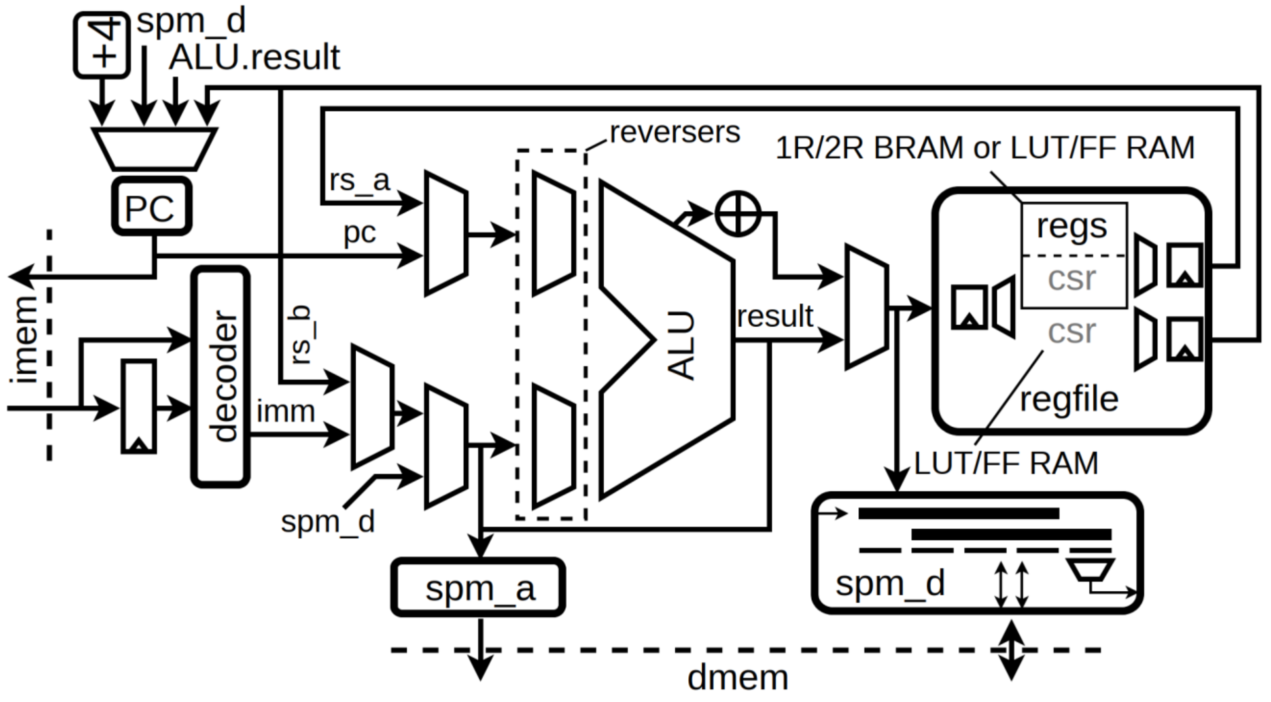


FazyRV: A scalable RISC-V core

Yet another RISC-V core?

Closes the Gap between 32-Bit and Bit-Serial.

➔ Goal:	Area vs. Performance trading
➔ ISA:	RV32I user specification compliant
➔ Scalability:	1, 2, 4, or 8-bit data path width
➔ Abstraction:	No hand optimization at gate level
➔ Variants:	“MIN” ⊂ “INT” ⊂ “CSR” single-port vs. dual-port RAM BRAM vs. LUT RAM

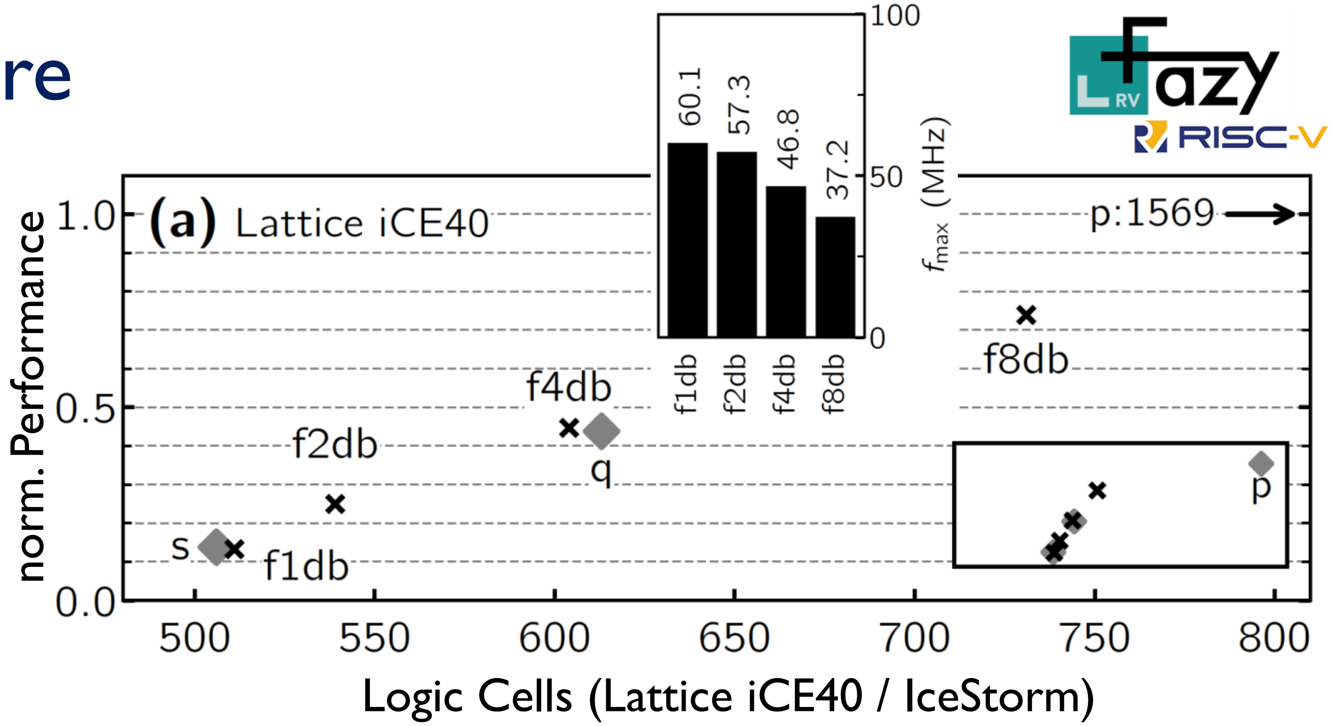


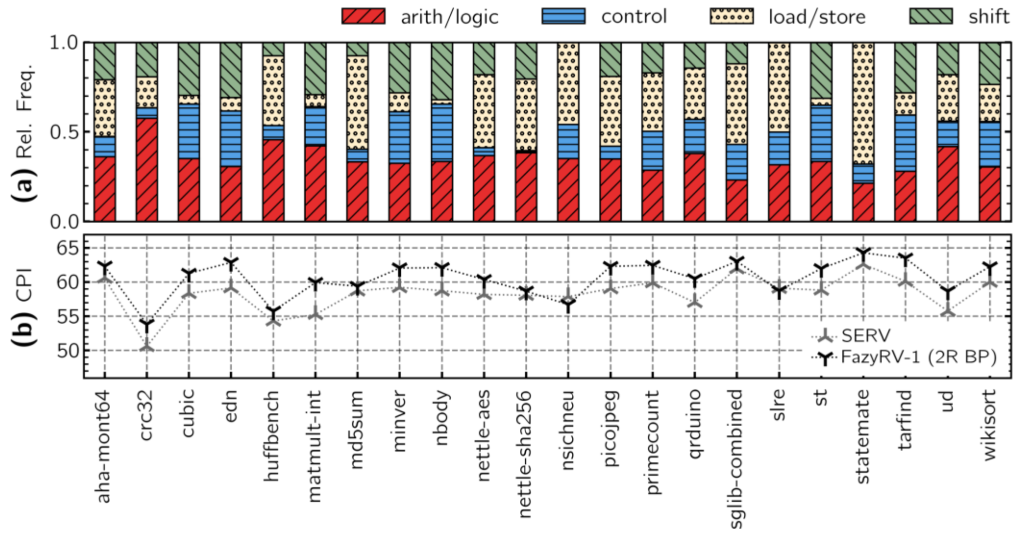
FazyRV: A scalable RISC-V core

Yet another RISC-V core?

Closes the Gap between 32-Bit and Bit-Serial.

➔ Goal:	Area vs. Performance trading
➔ ISA:	RV32I user specification compliant
➔ Scalability:	1, 2, 4, or 8-bit data path width
➔ Abstraction:	No hand optimization at gate level
➔ Variants:	"MIN" ⊂ "INT" ⊂ "CSR" single-port vs. dual-port RAM BRAM vs. LUT RAM





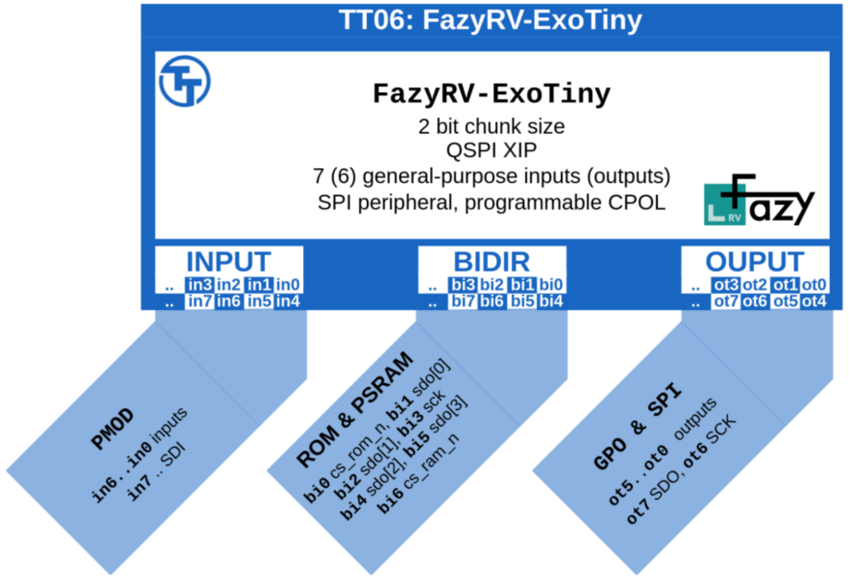
FazyRV: A scalable RISC-V core

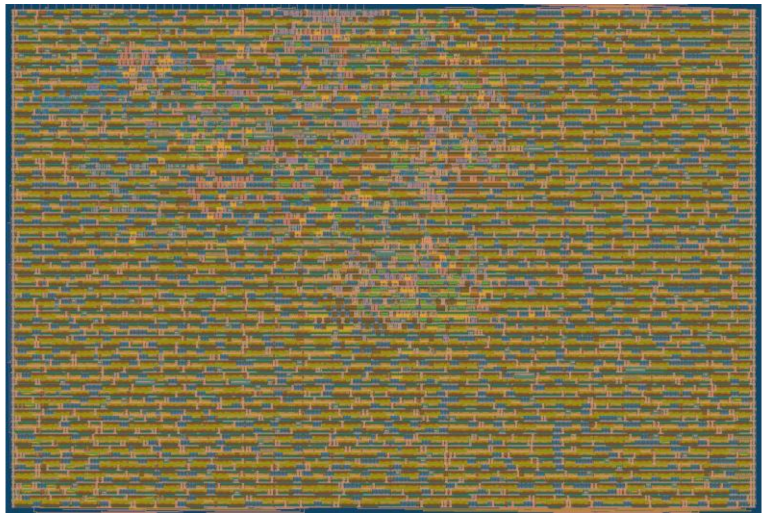
Yet another RISC-V core?

Closes the Gap between 32-Bit and Bit-Serial.

➔ Goal:	Area vs. Performance trading
➔ ISA:	RV32I user specification compliant
➔ Scalability:	1, 2, 4, or 8-bit data path width
➔ Abstraction:	No hand optimization at gate level
➔ Variants:	“MIN” ⊂ “INT” ⊂ “CSR” single-port vs. dual-port RAM BRAM vs. LUT RAM
➔ Tools:	Open Source Software only
➔ FazyRV:	Open Source License!

https://iti.tugraz.at/FazyRV



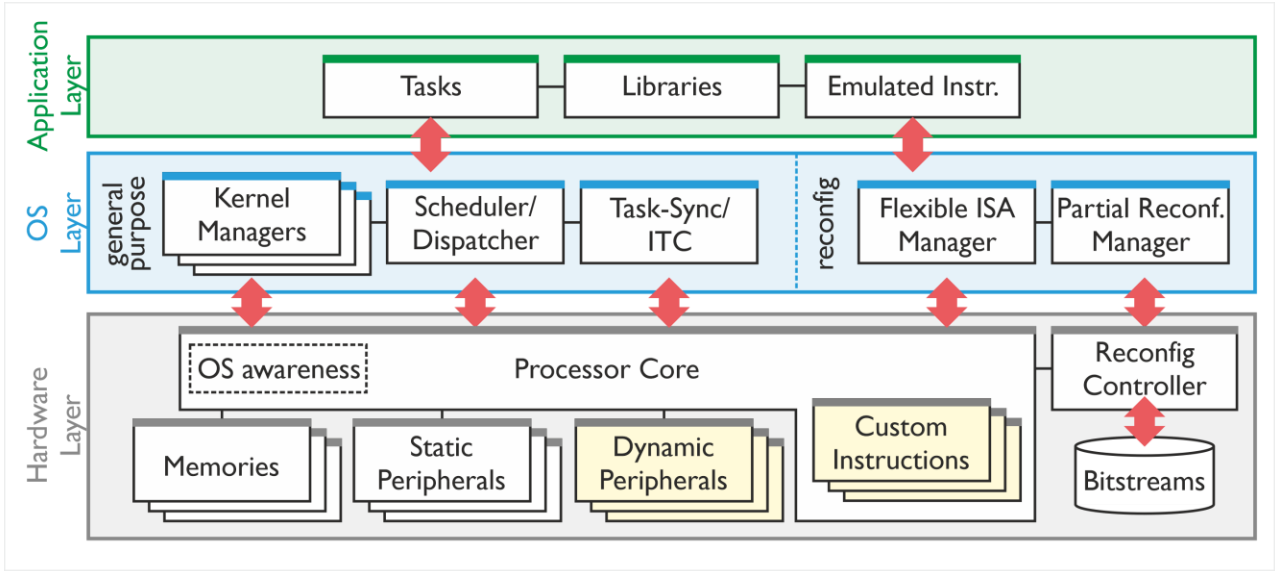


*more*MCU: Partial Reconfiguration at Runtime

Yet another RISC-V core?

Allows logic modifications at runtime.

→ Goal:	Application specific Logic
→ ISA:	Extensible (Base: RV32I)
→ Peripherals:	Extensible

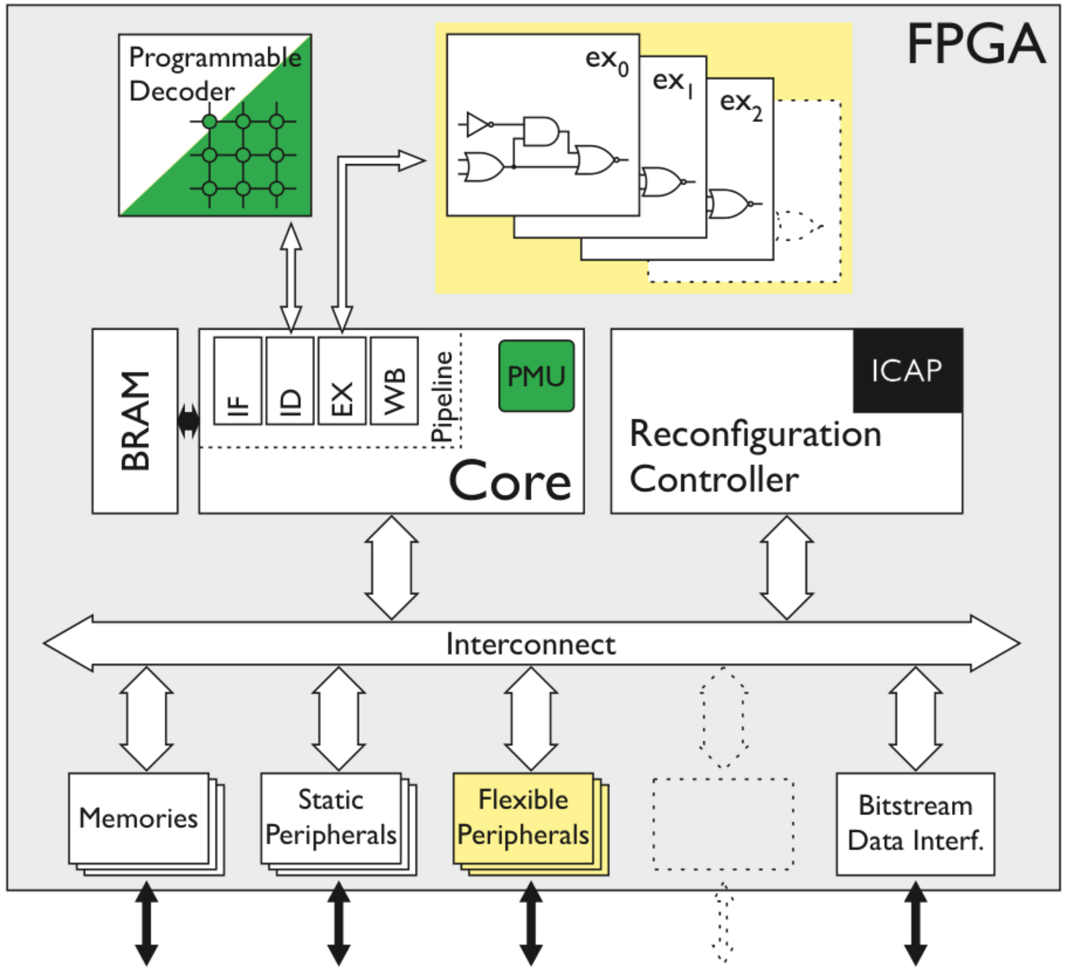


*more*MCU: Partial Reconfiguration at Runtime

Yet another RISC-V core?

Allows logic modifications at runtime.

- ➔ Goal: Application specific Logic
- ➔ ISA: Extensible (Base: RV32I)
- ➔ Peripherals: Extensible
- ➔ Reconfiguration:
  - Managed by OS (Profiling in HW/SW)
  - Bitstream loading in HW (in parallel to SW exec)



moreMCU: Partial Reconfiguration at Runtime

Yet another RISC-V core?

Allows logic modifications at runtime.

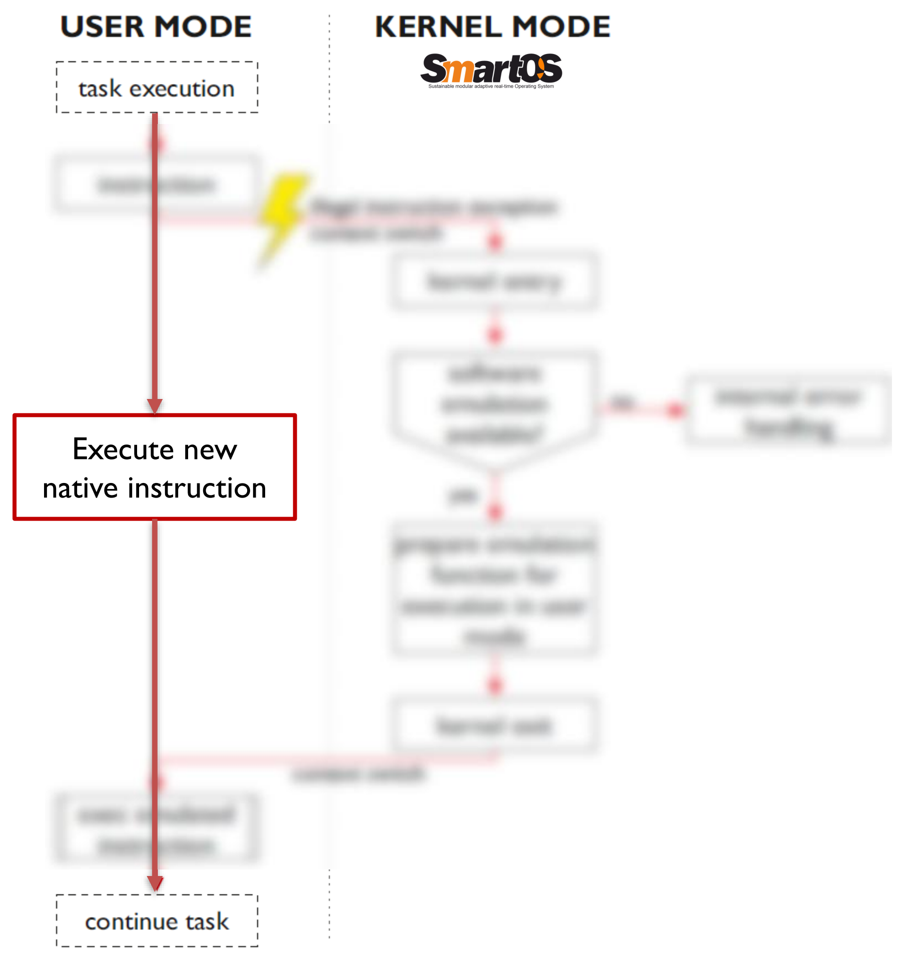
➔ Goal:	Application specific Logic
➔ ISA:	Extensible (Base: RV32I)
➔ Peripherals:	Extensible

➔ Reconfiguration:

Managed by OS (Profiling in HW/SW)
Bitstream loading in HW (in parallel to SW exec)

➔ Overhead for new instructions

Emulated:	~83 cycles
Native:	0 cycles

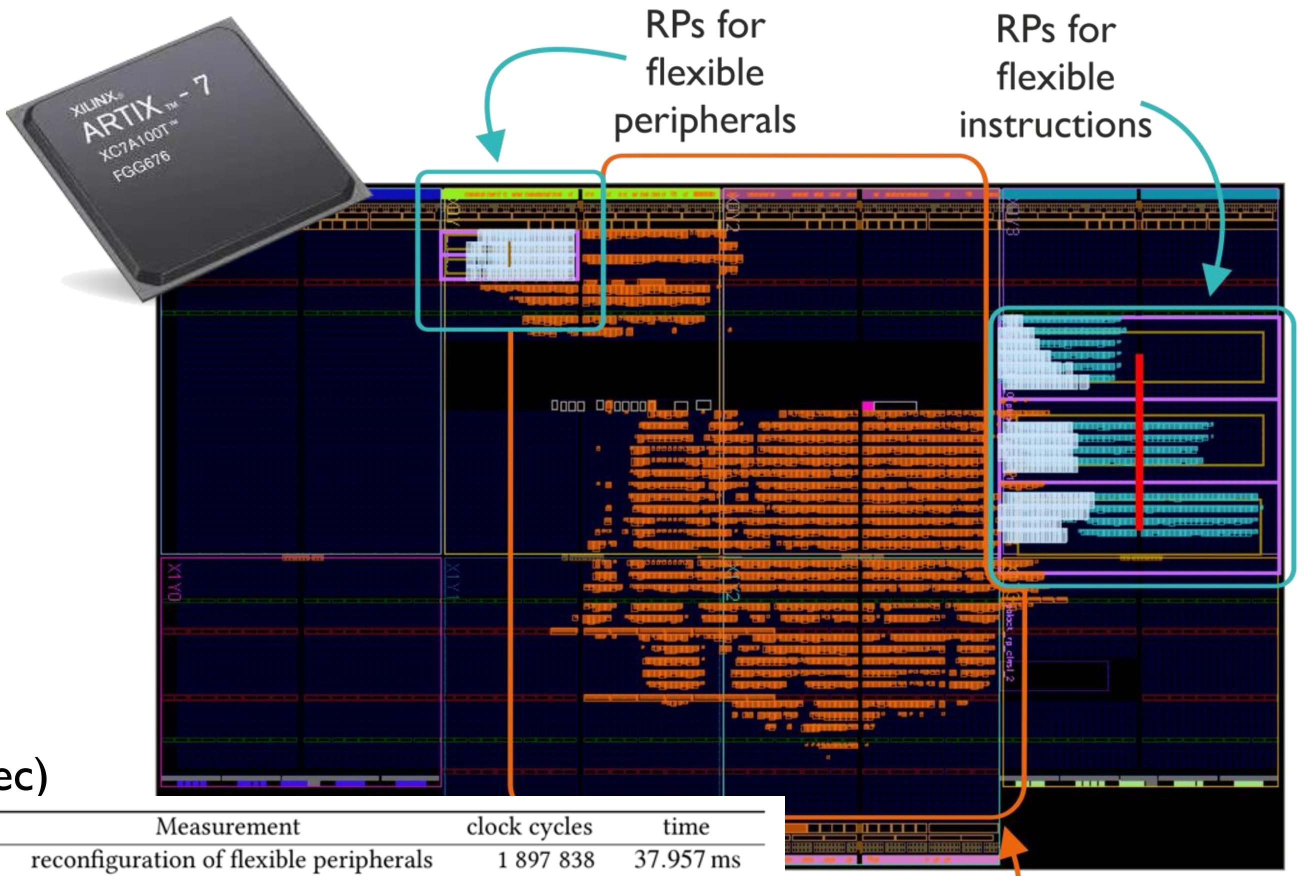


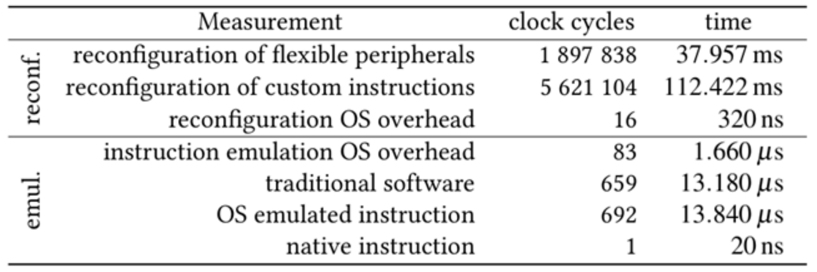
***more*MCU**: Partial Reconfiguration at Runtime

Yet another RISC-V core?

Allows logic modifications at runtime.

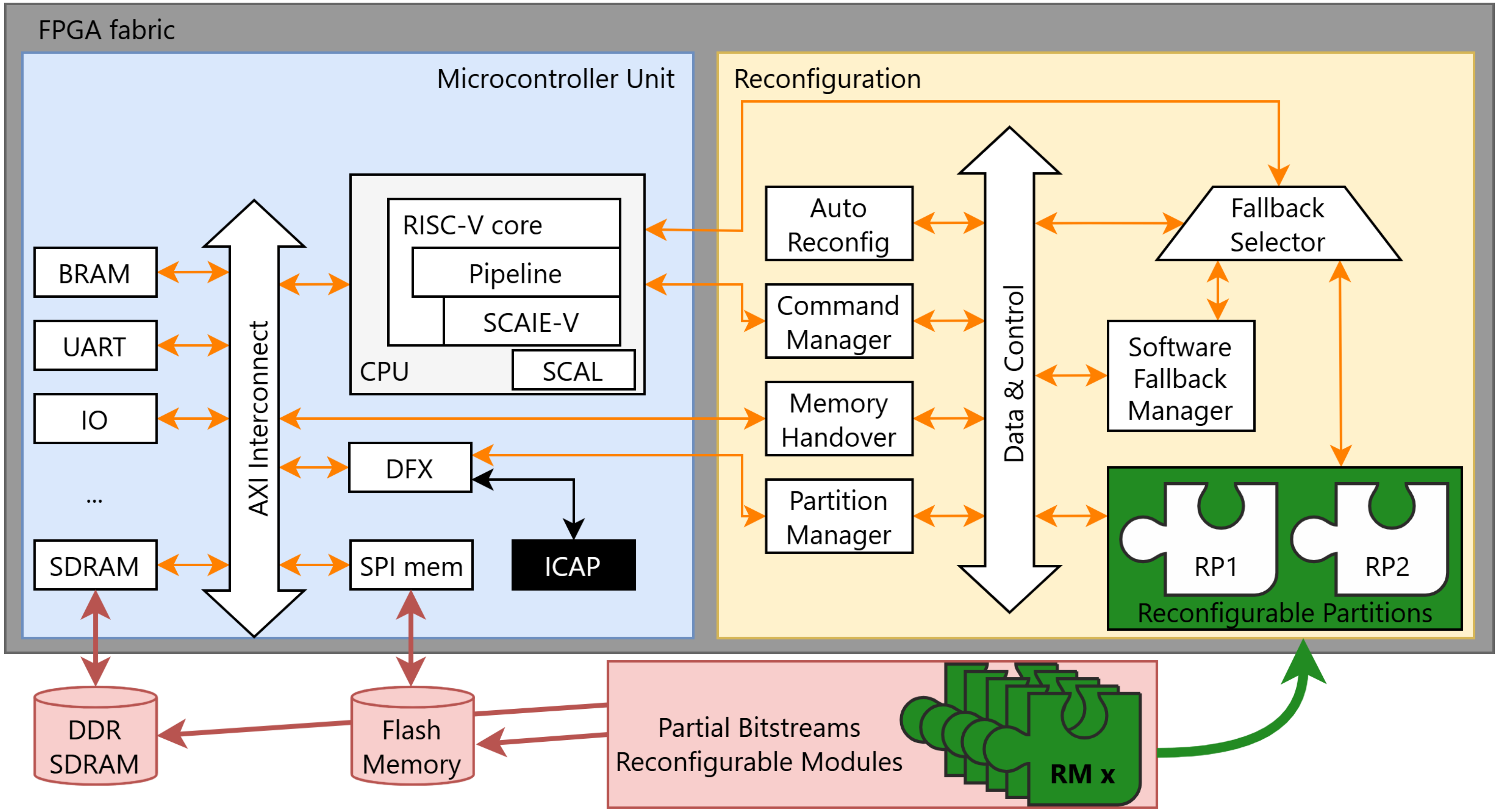
- ➔ Goal: Application specific Logic
- ➔ ISA: Extensible (Base: RV32I)
- ➔ Peripherals: Extensible
- ➔ Reconfiguration:
  - Managed by OS (Profiling in HW/SW)
  - Bitstream loading in HW (in parallel to SW exec)
- ➔ Overhead for new instructions
  - Emulated: ~83 cycles
  - Native: 0 cycles



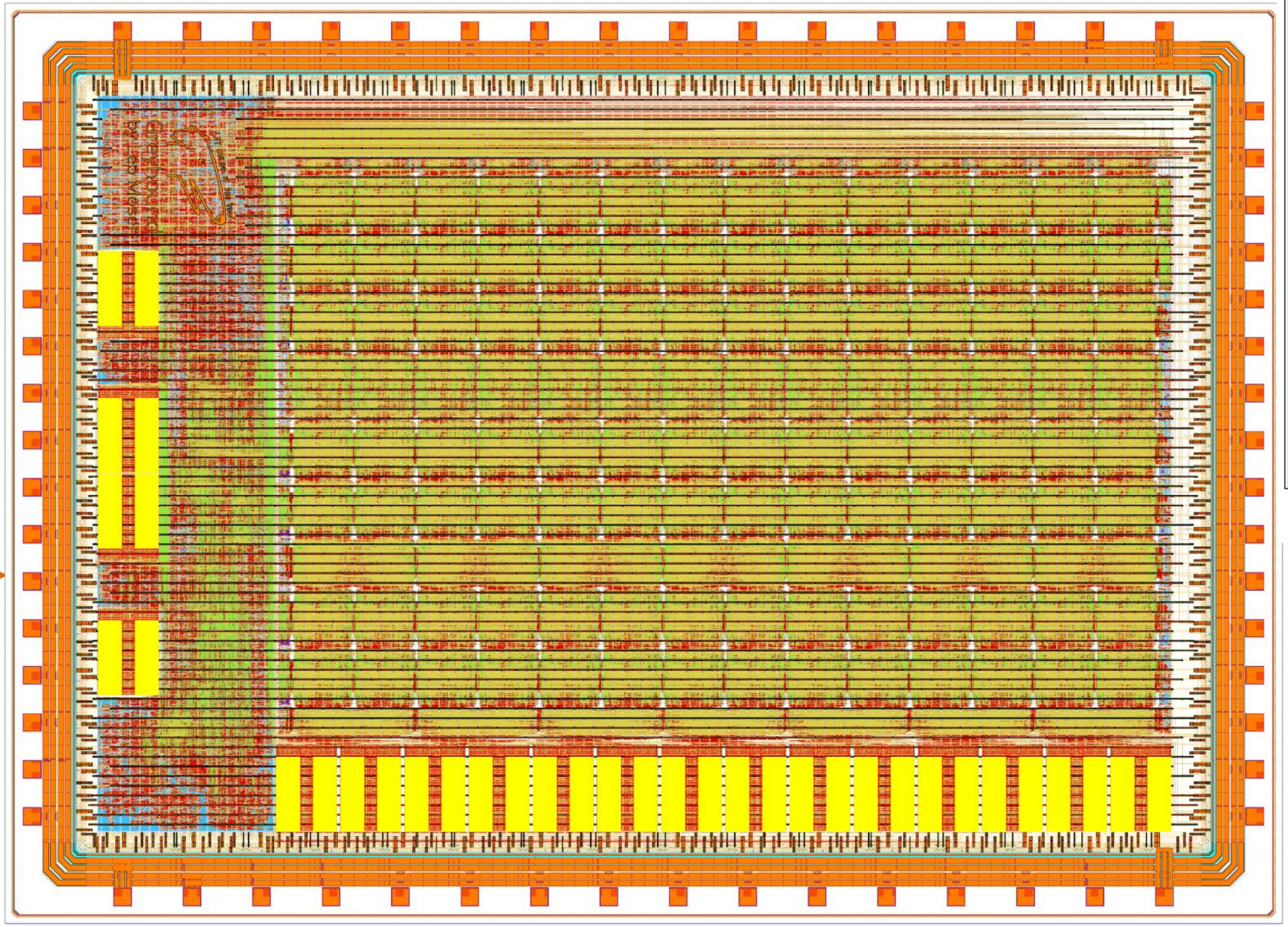


	Measurement	clock cycles	time
reconf.	reconfiguration of flexible peripherals	1 897 838	37.957 ms
	reconfiguration of custom instructions	5 621 104	112.422 ms
	reconfiguration OS overhead	16	320 ns
emul.	instruction emulation OS overhead	83	1.660 μs
	traditional software	659	13.180 μs
	OS emulated instruction	692	13.840 μs
	native instruction	1	20 ns

moreMCU: Partial Reconfiguration at Runtime

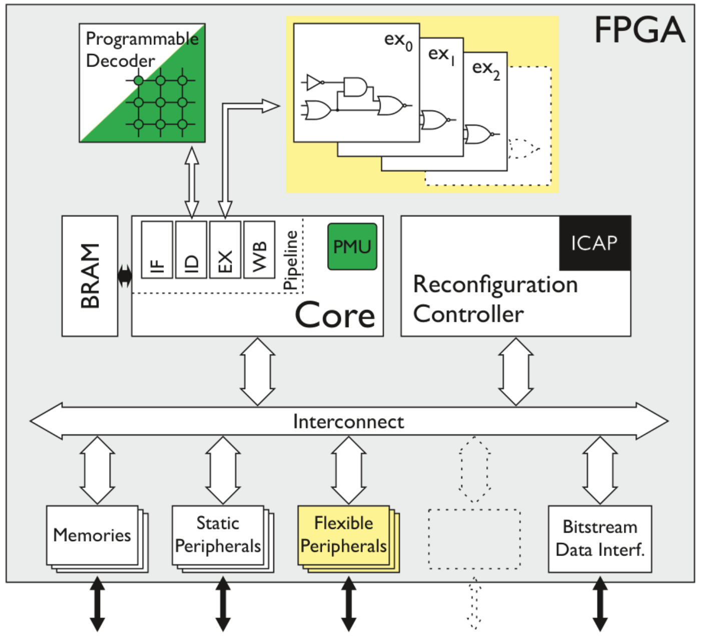


*more*MCU as ASIC: Greyhound



static core

embedded FPGA

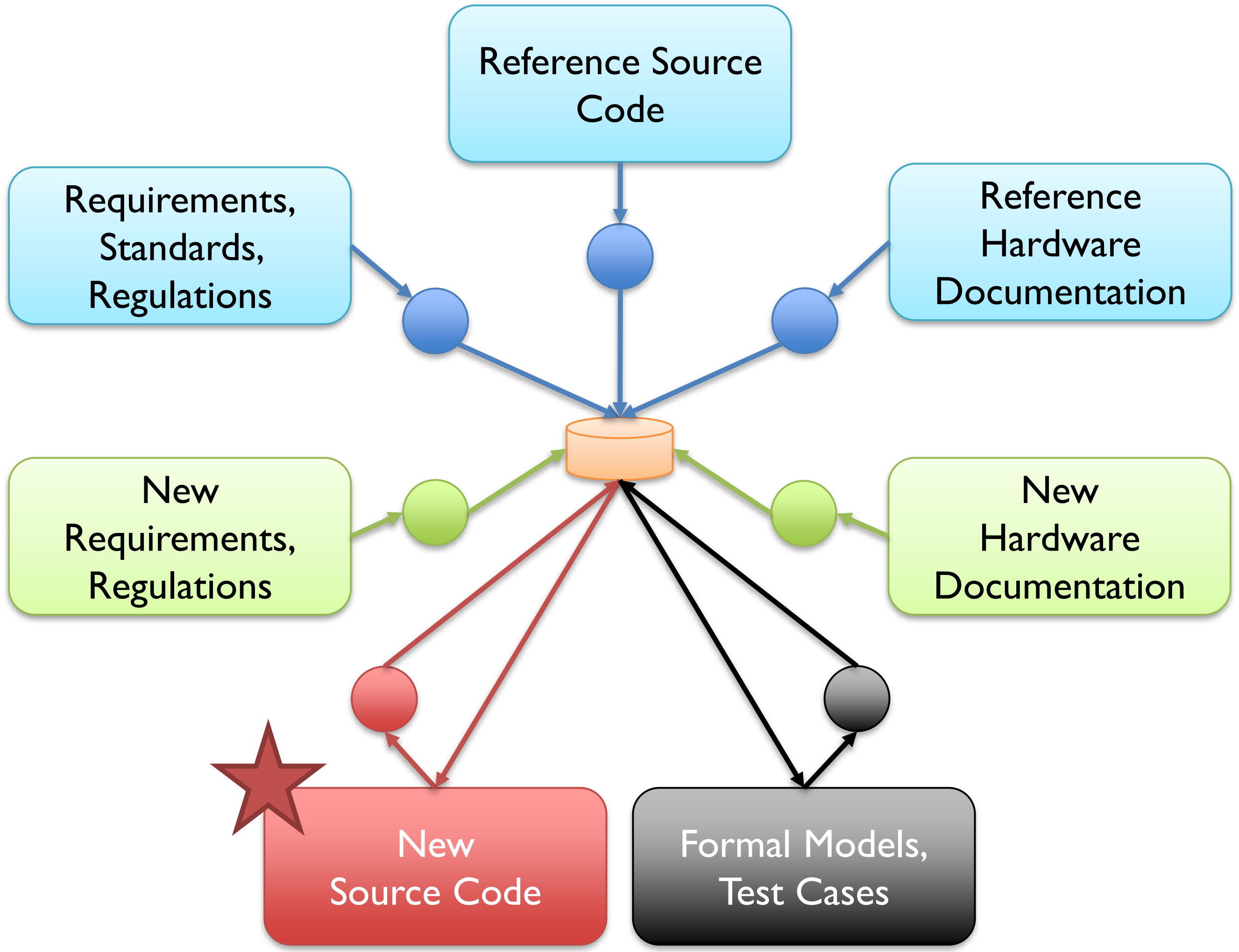


CoEvolve: Co-Evolution of HW & SW

Challenge:
Hardware changes, requirements change.
Software must adapt and stay
functional & correct.

Idea:
Implement an automated porting/adaptation
workflow that supports developers through
AI & FM.

1. Understand
2. Generate
3. Verify
4. Deliver


Reference Source Code
Requirements, Standards, Regulations
Reference Hardware Documentation
New Requirements, Regulations
New Hardware Documentation
New Source Code
Formal Models, Test Cases

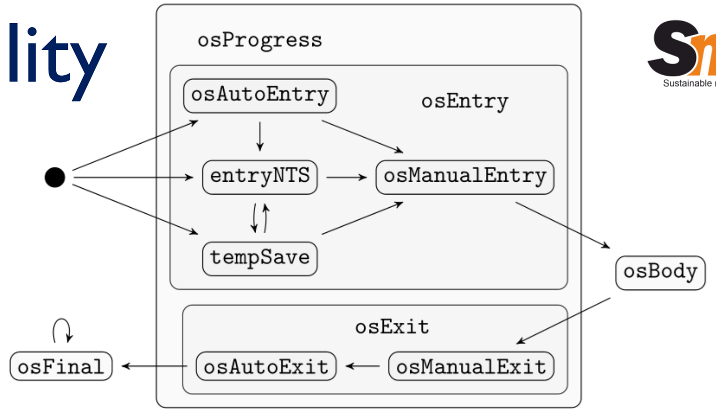
SmartOS: Kernel Verification and Portability

System Software for Flexible Hardware?

Formal Methods for Porting and Verification.

➔ Goal: Transit to new Architectures
Exploit new Features of
reconfigurable Architectures

➔ Formal Models (Event-B, Dafny, LEAN)
Verification
Code Generation



Event-B

```
Kernel_entry
savedCtx:  saveCtx(runTask)
             := saveCtx(runTask)
             ⩤ (f_sav2reg; currCtx)
kMode:  os_kernelMode := TRUE
currexec:  currExec := kernel_body
```

LLVM-IR

```
%R8 = call i16 @llvm.read_register.i16(r8)
%rTsk = load i16, i16* @rTask
// t_saved[rTask]
%gepRT = getelementptr i16, i16* @t_saved,
     i16 %rTsk
// t_saved[rTask][mR8], with mR8=5
%gepMR8 = getelementptr i16, i16* %gepRT,
    i16 5
store i16 %R8, i16* %gepMR8
```

```
Kernel_entry:
push r15
push r14
push r13
push r12
push r11
push r10
push r9
push r8
push r7
push r6
push r5
push r4
mov &os_runningTask,r4
mov sp,0*2(r4)
mov #1,&os_kernelMode
jmp kernel_body
```

ASM
RISC-V/MSP430/Aurix/…

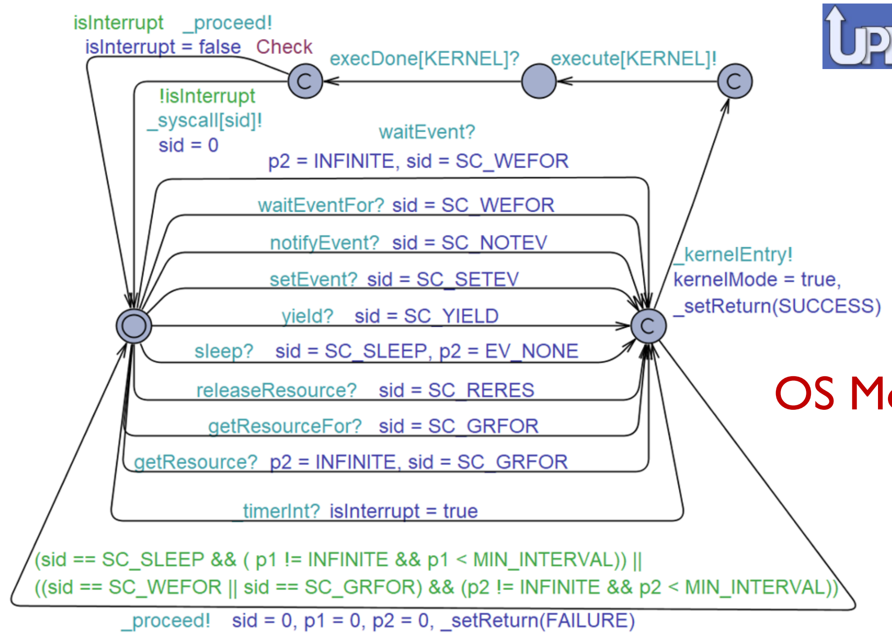
SmartOS: Compositional Application Software

Modular Application Software?

Formal Methods for Integration Checks.

➔ Goal: Partial Updates at Runtime.
Guarantee correct Operation.

➔ Independent Formal
OS & Application Model (Uppaal)


isInterrupt _proceed!
isInterrupt = false Check
execDone[KERNEL]?
execute[KERNEL]!
!isInterrupt
_syscall[sid]!
sid = 0
waitEvent?
p2 = INFINITE, sid = SC_WEFOR
waitEventFor? sid = SC_WEFOR
notifyEvent? sid = SC_NOTEV
setEvent? sid = SC_SETEV
yield? sid = SC_YIELD
sleep? sid = SC_SLEEP, p2 = EV_NONE
releaseResource? sid = SC_RERES
getResourceFor? sid = SC_GRFOR
getResource? p2 = INFINITE, sid = SC_GRFOR
timerInt? isInterrupt = true
_kernelEntry!
kernelMode = true,
_setReturn(SUCCESS)
(sid == SC_SLEEP && (p1 != INFINITE && p1 < MIN_INTERVAL)) ||
((sid == SC_WEFOR || sid == SC_GRFOR) && (p2 != INFINITE && p2 < MIN_INTERVAL))
_proceed! sid = 0, p1 = 0, p2 = 0, _setReturn(FAILURE)
C
C
C

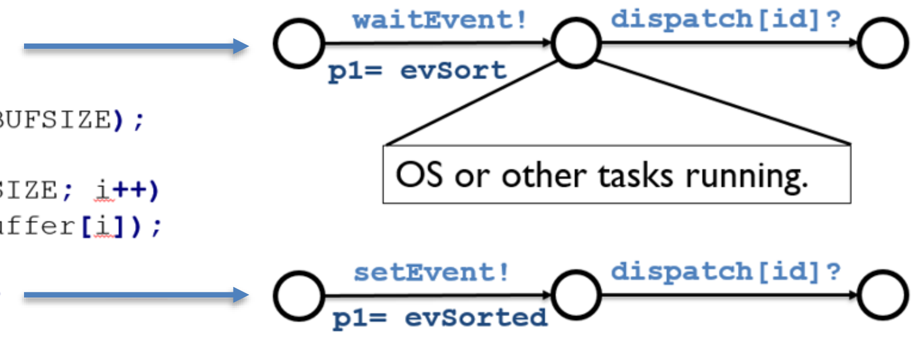
OS Model

```
OS_TASKENTRY (taskSort){
 while (1){
   waitEvent (evSort);

   quickSort (buffer, BUFSIZE);

   for (int i=0; i<BUFSIZE; i++)
     printf("\n\%u", buffer[i]);

   setEvent (evSorted);
 }
}
```



Application Code

Application Model

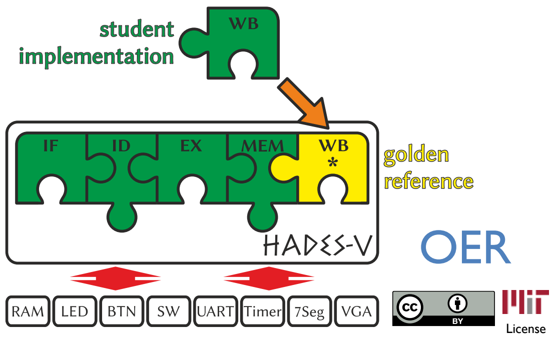
Teaching & Education

HADES-V

Yet another RISC-V core?

Teach students how to build their own CPU

- ➔ Language: SystemVerilog
- ➔ ISA: RV32I
- ➔ Method: Implement logic & synthesize for an FPGA

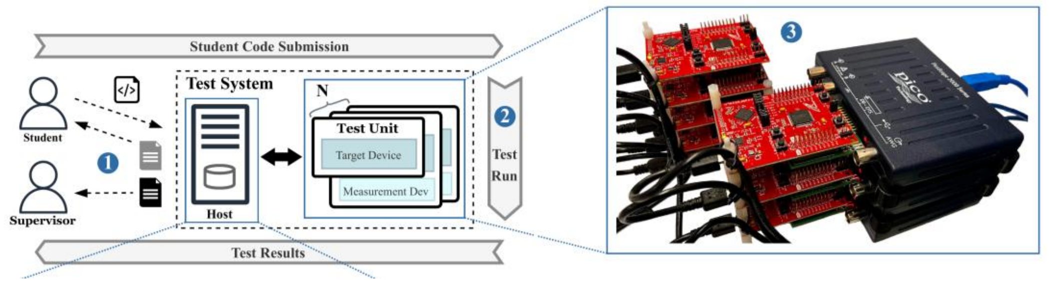


SmartOS
Sustainable modular adaptive real-time Operating System

Yet another RTOS?

Teach students how to build their own OS Kernel

- ➔ Language: C / ASM
- ➔ Target: MPS430 / RISC-V (from 2027)
- ➔ Method: Implement & analyze for correctness, timing, etc.



Flexible Embedded Systems!? A Summary…

Thank you!

It can be done!

- Requires HW/SW Co-Evolution (instead of Co-Design)
- New concepts required at all system levels: Electronics (engergy, flexibility), Logic (manufacturing, complexity), System Software (portability), Applications (composition)
- Reconfigurable hardware still resource demanding: Energy, space, cost, …
- Rigorous methods required for proving correctness of modifications
- Tooling & Education to support HW/SW engineering
- Experts & AI required!

Replace existing Devices

~~Sometimes difficult. Time consuming. Expensive. Wasteful.~~

Avoidable

Change and Reuse *flexible* HW & SW

Easier.
Faster.
(Cheaper?)
Sustainable.