

Dec 7, 2013
CARL2013@Davis, CA



PyCoRAM

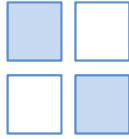
Yet Another Implementation of CoRAM Memory
Architecture for Modern FPGA-based Computing

Shinya Takamaeda-Yamazaki^{†‡}, Kenji Kise[†], James C. Hoe^{*}

[†]Tokyo Institute of Technology

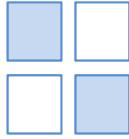
[‡]JSPS Research Fellow

^{*}Carnegie Mellon University

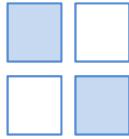


Agenda

- Background
- PyCoRAM Overview
- PyCoRAM Microarchitecture
- Evaluation
- Conclusion



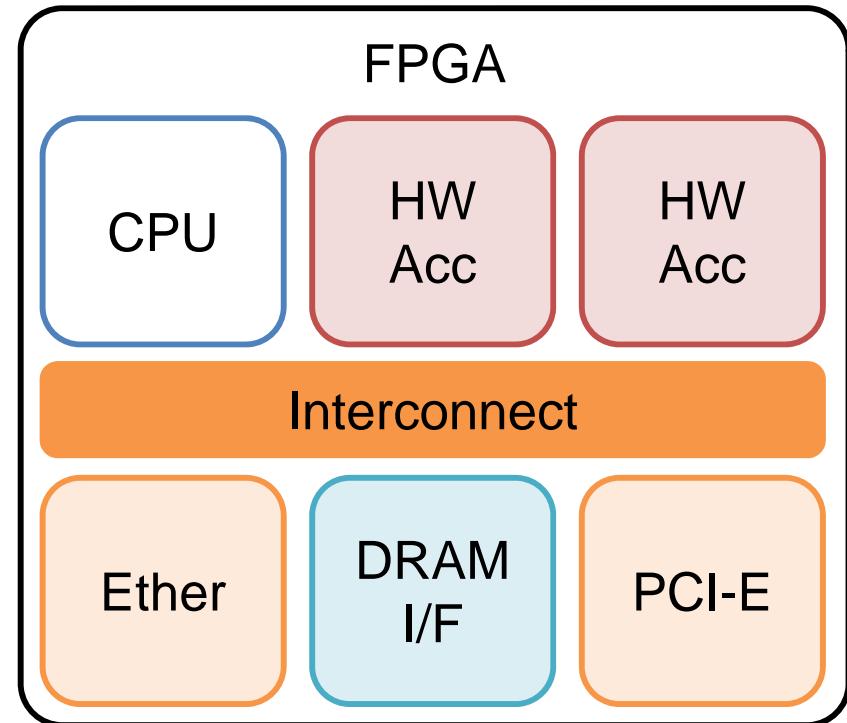
Background

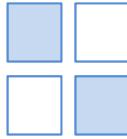


FPGA as SoC

■ Put together various components on a single FPGA

- CPU core
 - Microblaze (Soft-macro)
 - Cortex-A9 (Hard-macro)
- Hardware accelerator logic
 - Modeled in traditional RTL
 - Verilog HDL, VHDL
 - Modeled in new modeling tool
 - Bluespec, AutoESL, Chisel, ...
- DDRx DRAM interface
- PCI-express
- Ethernet, ...

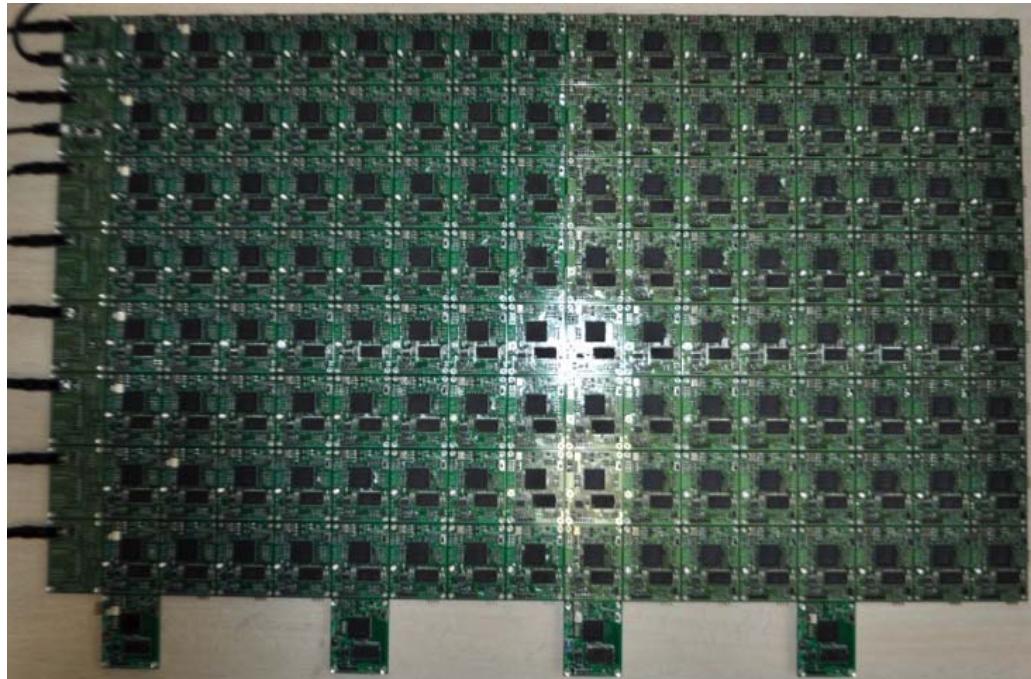




Portability Issue of Application Design

■ How to support various FPGA platforms?

- Different logic size,
memory interface,
peripherals and I/O property ☹



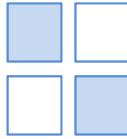
ScalableCore System (our FPGA system)
(Xilinx Spartan-6 LX16 × 128-node)



Digilent Atlys
(Xilinx Spartan-6 LX45)

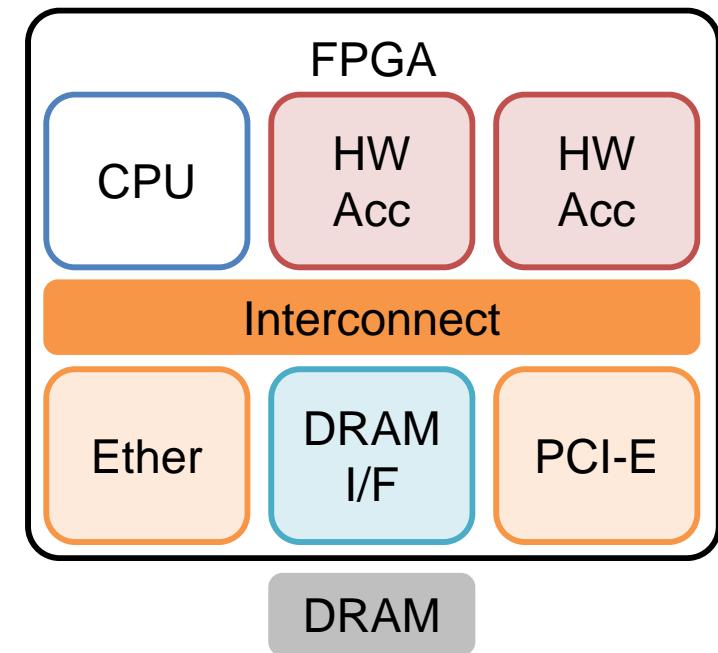
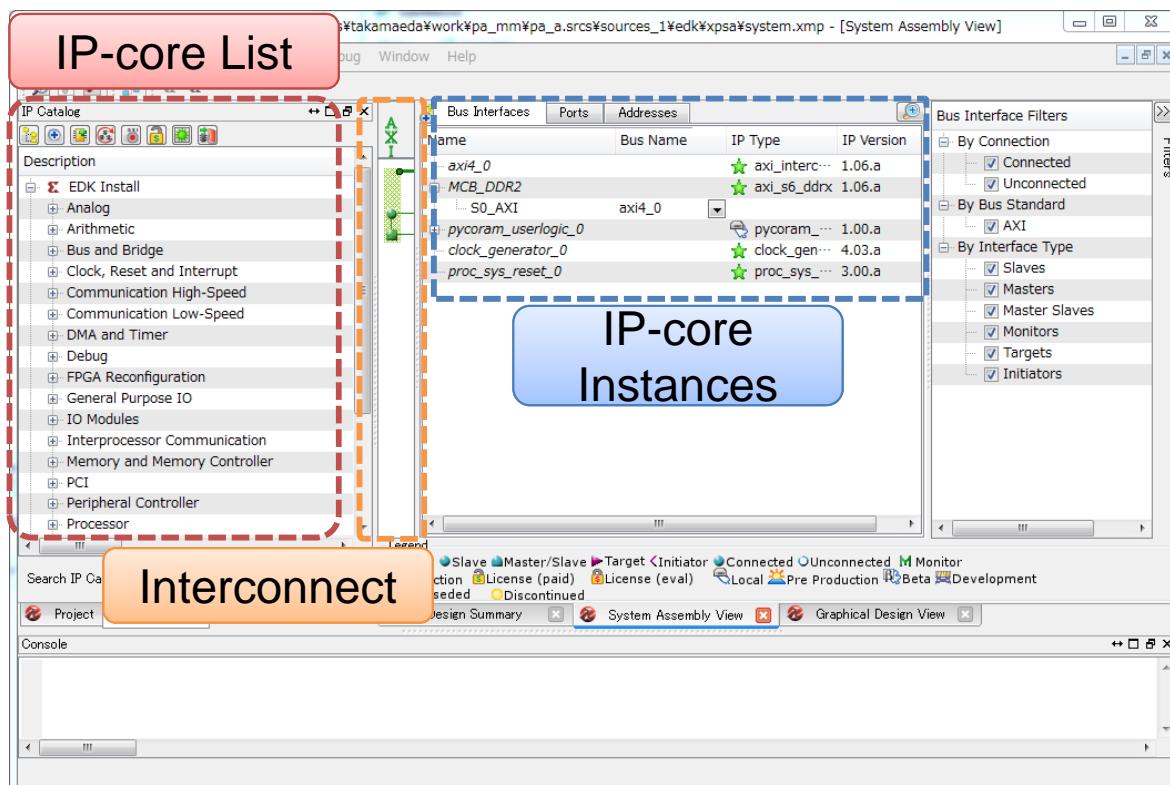


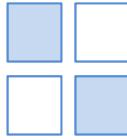
Xilinx ML605
(Xilinx Virtex-6 LX240T)



IP-core Based System Development

- To build a system, add IP-cores and connect them ☺
 - IP-cores are connected through a standard on-chip interconnect
 - EDK automatically generates an on-chip interconnection and (some) device-dependent interfaces
 - No (or few) annoying steps!

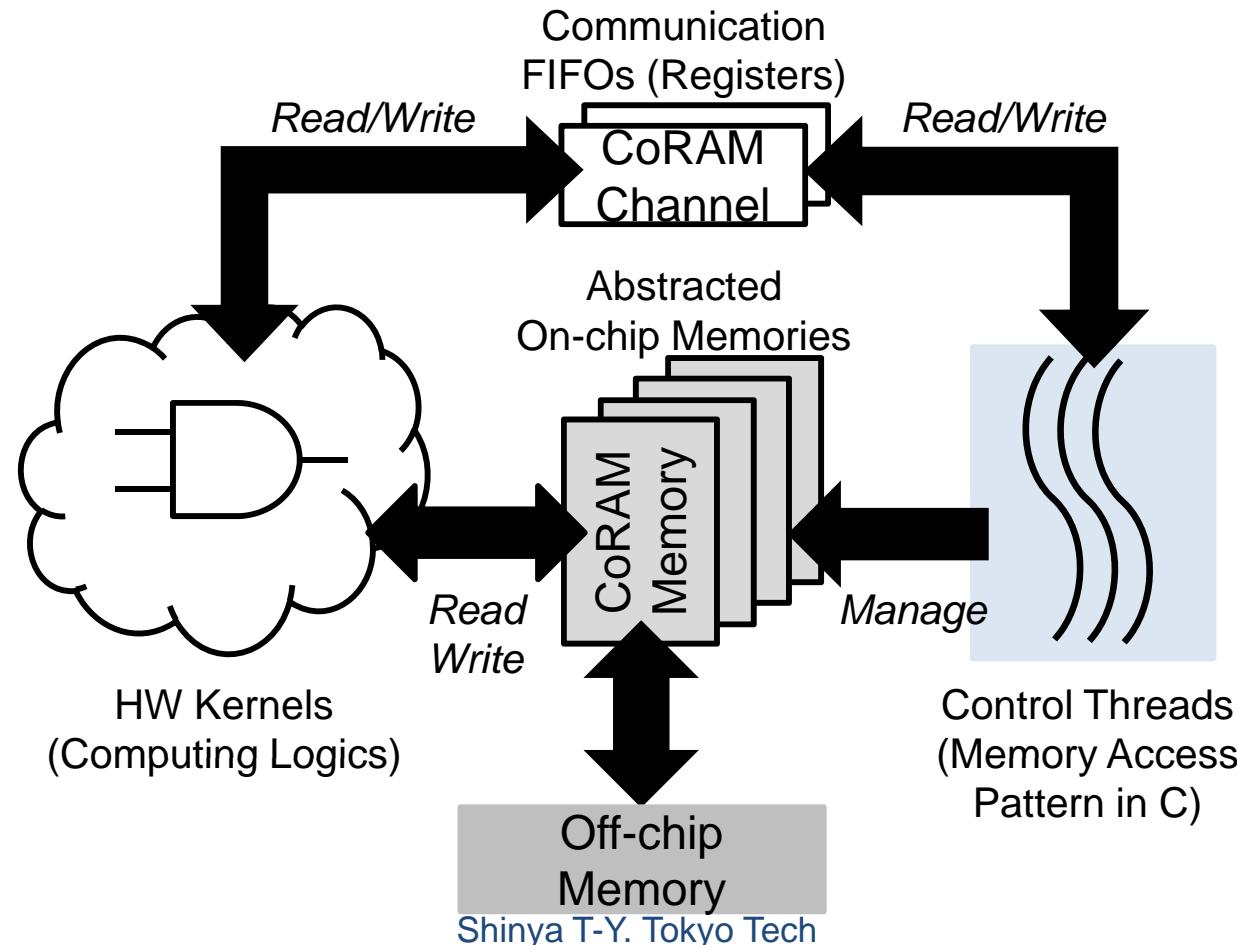




Abstract Memory System for FPGAs

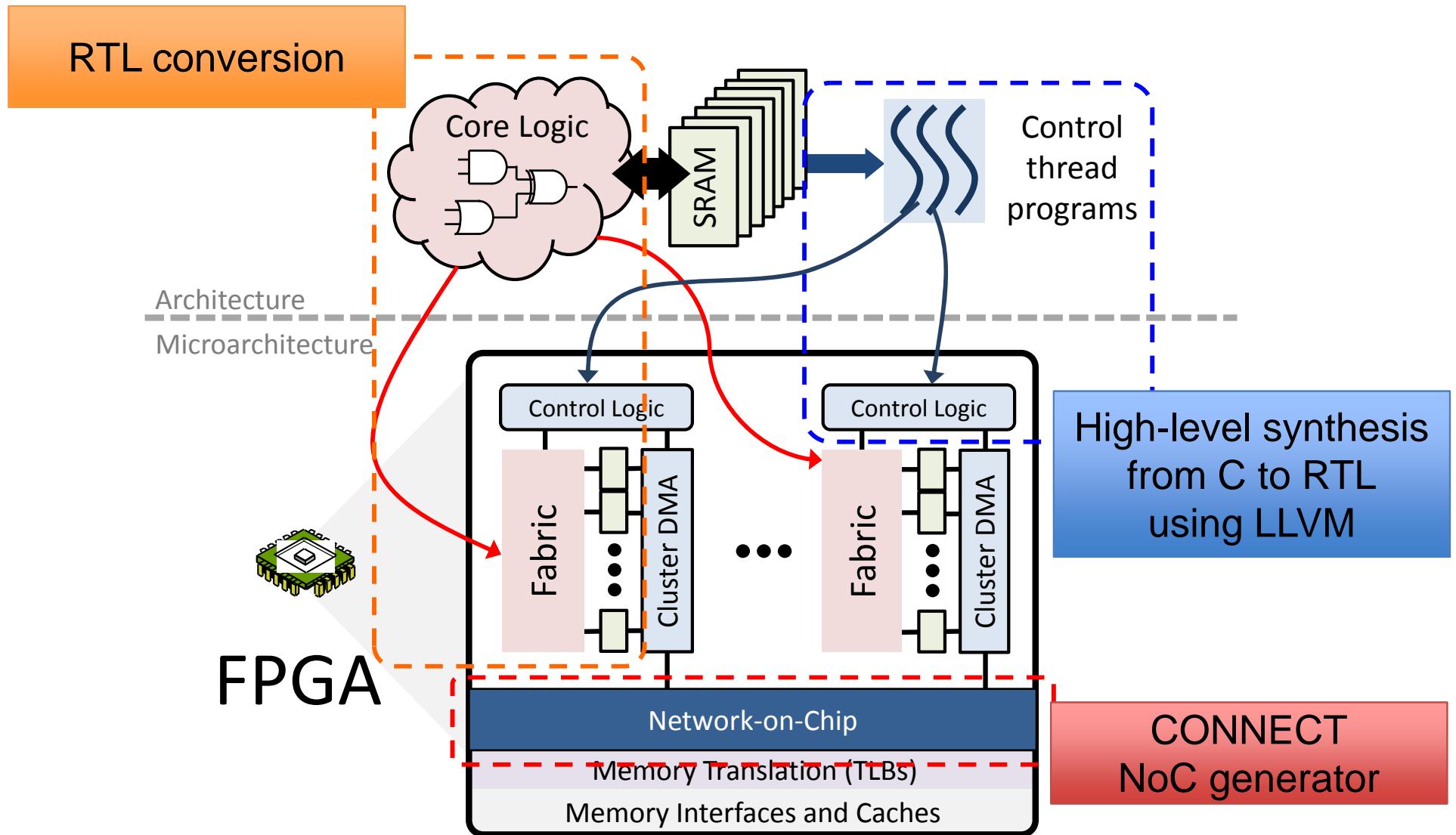
■ CoRAM (Connected RAM) [FPGA'11]

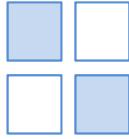
- High-level abstraction for memory management
 - Decoupling computing logics and memory access behaviors
 - Memory access patterns in software model (C language)



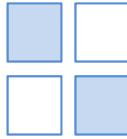
What “uns” CoRAM?

From
CoRAM Tutorial
@FPGA’13



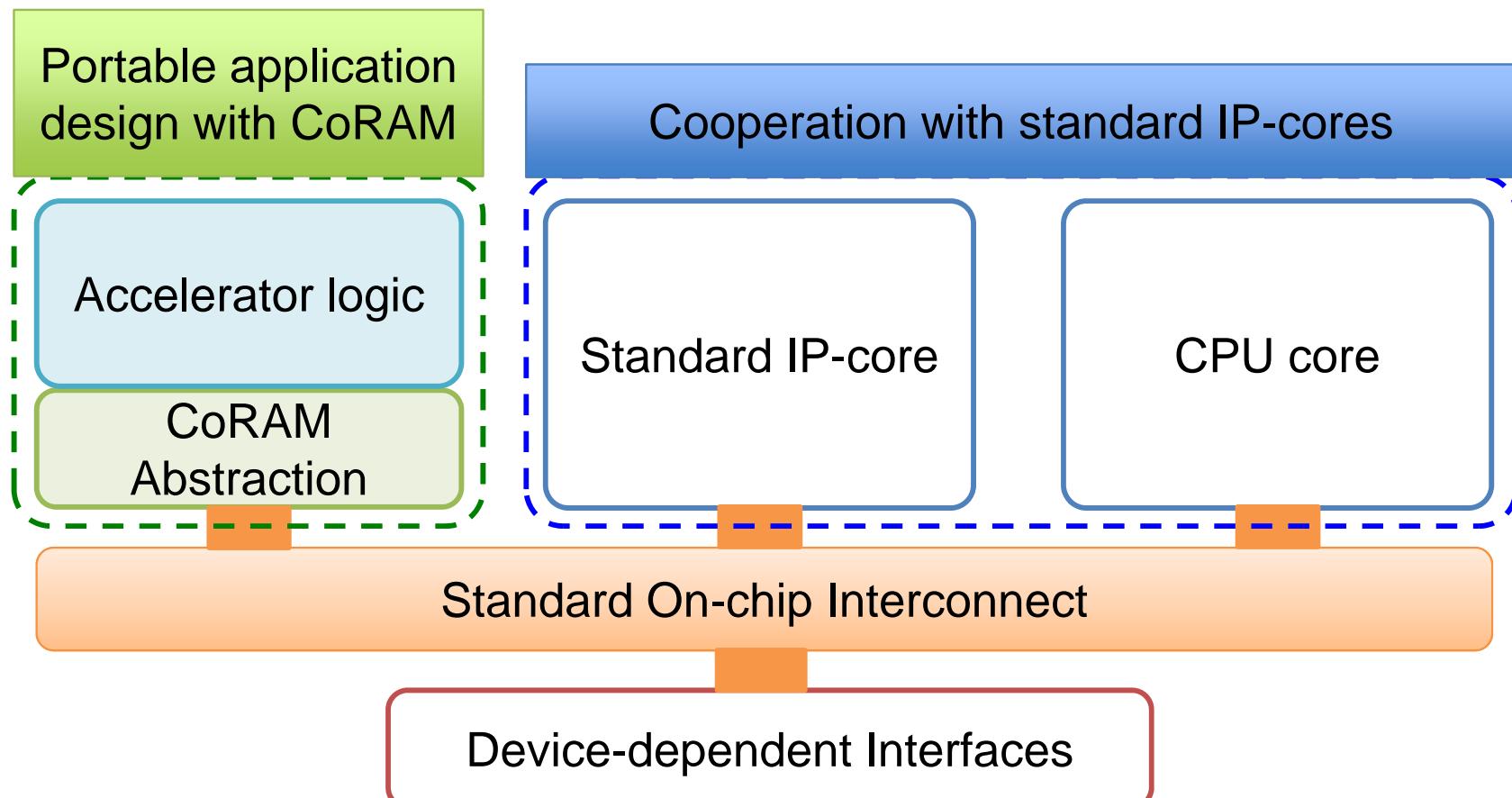


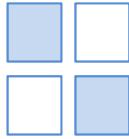
PyCoRAM Overview



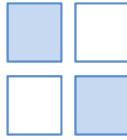
Motivation: CoRAM for EDK

- Integration of CoRAM memory architecture for modern EDK-based development flow with standard IP-cores





- Python-based implementation of CoRAM memory architecture for modern FPGA EDKs
 - CoRAM memory abstraction for EDK development flow
- Key features
 - Control Thread in Python
 - We developed Python-to-Verilog HLS Compiler from scratch
 - AMBA AXI4 Interconnect for on-chip interconnect
 - For IP-core based development on Xilinx Platform Studio (XPS)
 - Parameterized RTL Design Support for User-logic
 - Generate-statement and Parameter-statement analyzed by our original Verilog analysis tool-chain (Pyverilog)



Comparison with Original CoRAM

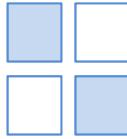
	CoRAM	PyCoRAM
Language for Control-Thread	C	Python
Supported Memory Operations	(Blocking/Non-Blocking) Read/Write	(Blocking/Non-Blocking) Read/Write
On-chip Interconnect	CONNECT NoC [FPGA'12]	AMBA AXI4
FSM Granularity in Control Thread	LLVM-IR	Python AST Node
Generate Statement Support for User logics	No	Yes
Supported FPGAs	Xilinx ML605 Altera Terasic DE-4	Any FPGAs supporting AXI Bus
# Lines of Code	11,682 lines (w/o CONNECT)	4,922 lines (w/o Pyverilog)

FSM: Finite State Machine

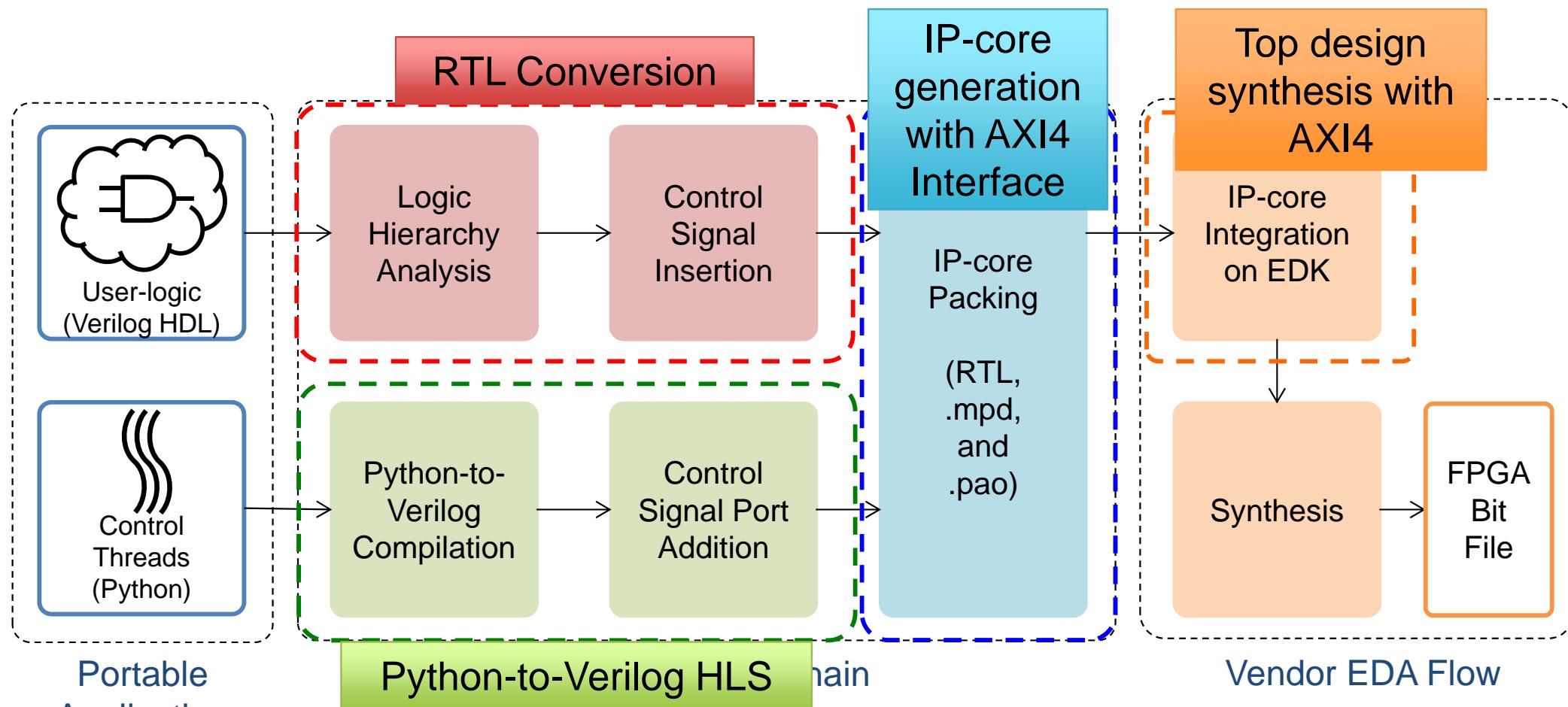
LLVM-IR: Low Level Virtual Machine Intermediate Representation

AST: Abstract Syntax Tree

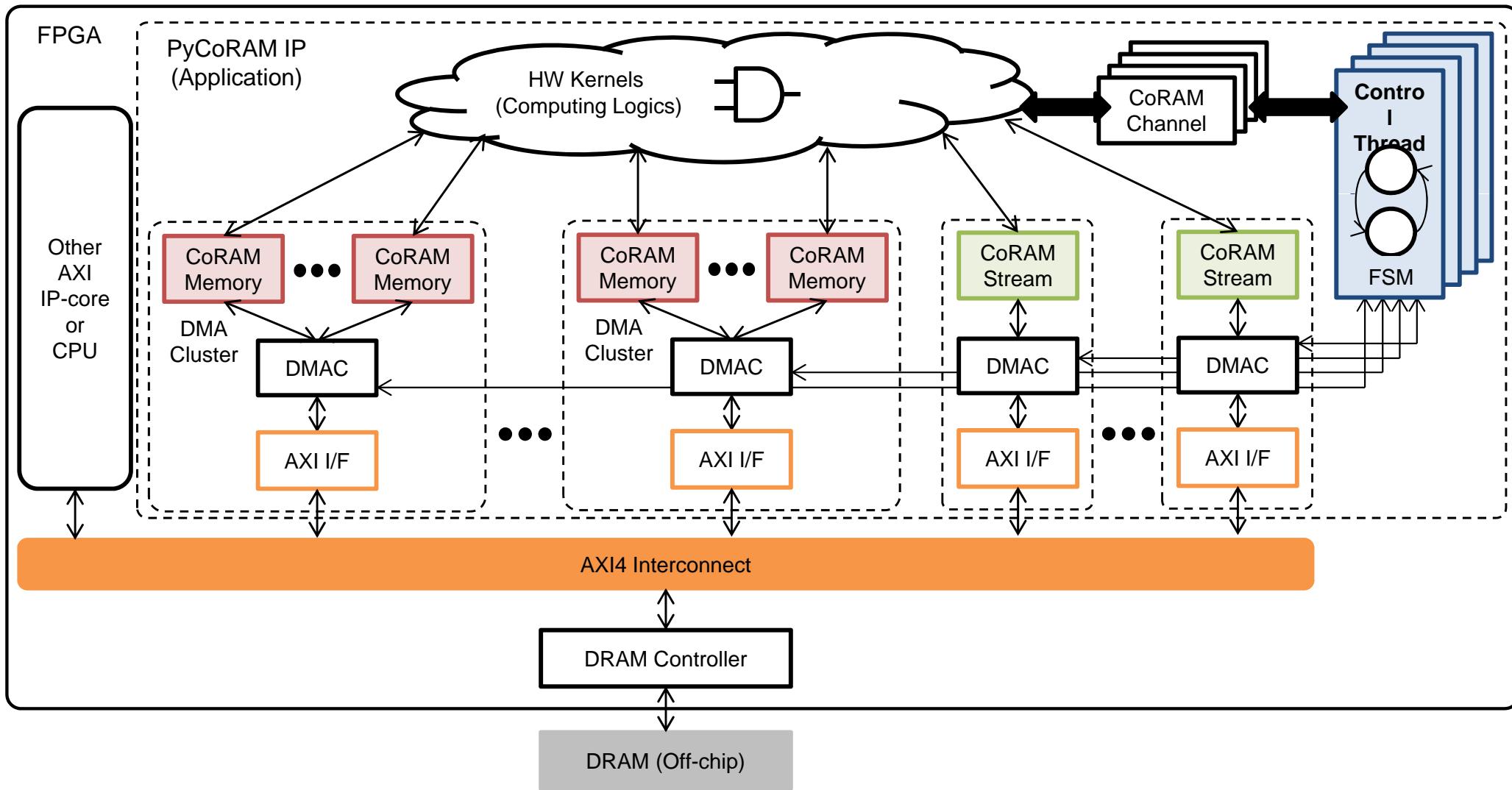
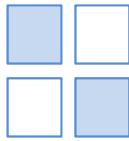
PyCoRAM Development Flow

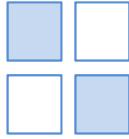


- PyCoRAM generates an IP-core package from user-logic RTLs and control thread scripts in Python
 - Each part can be replaced with the original CoRAM's component



FPGA Accelerator with PyCoRAM IP-core

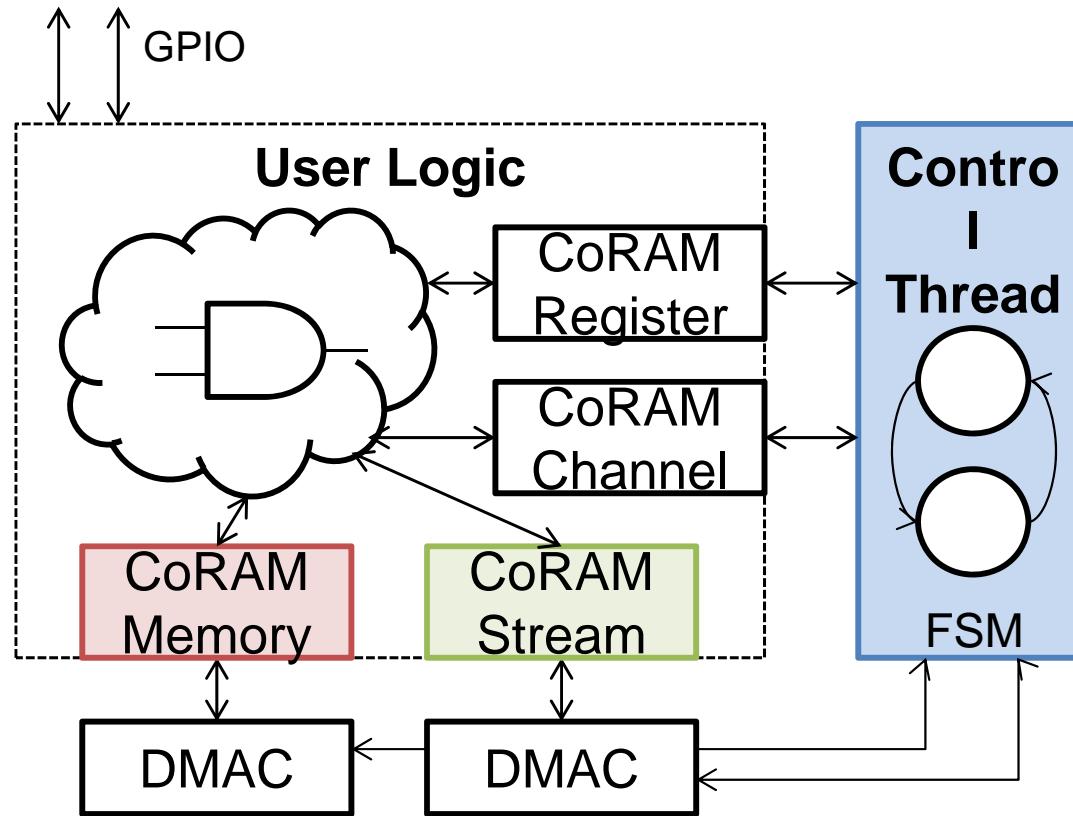
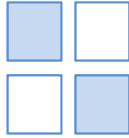




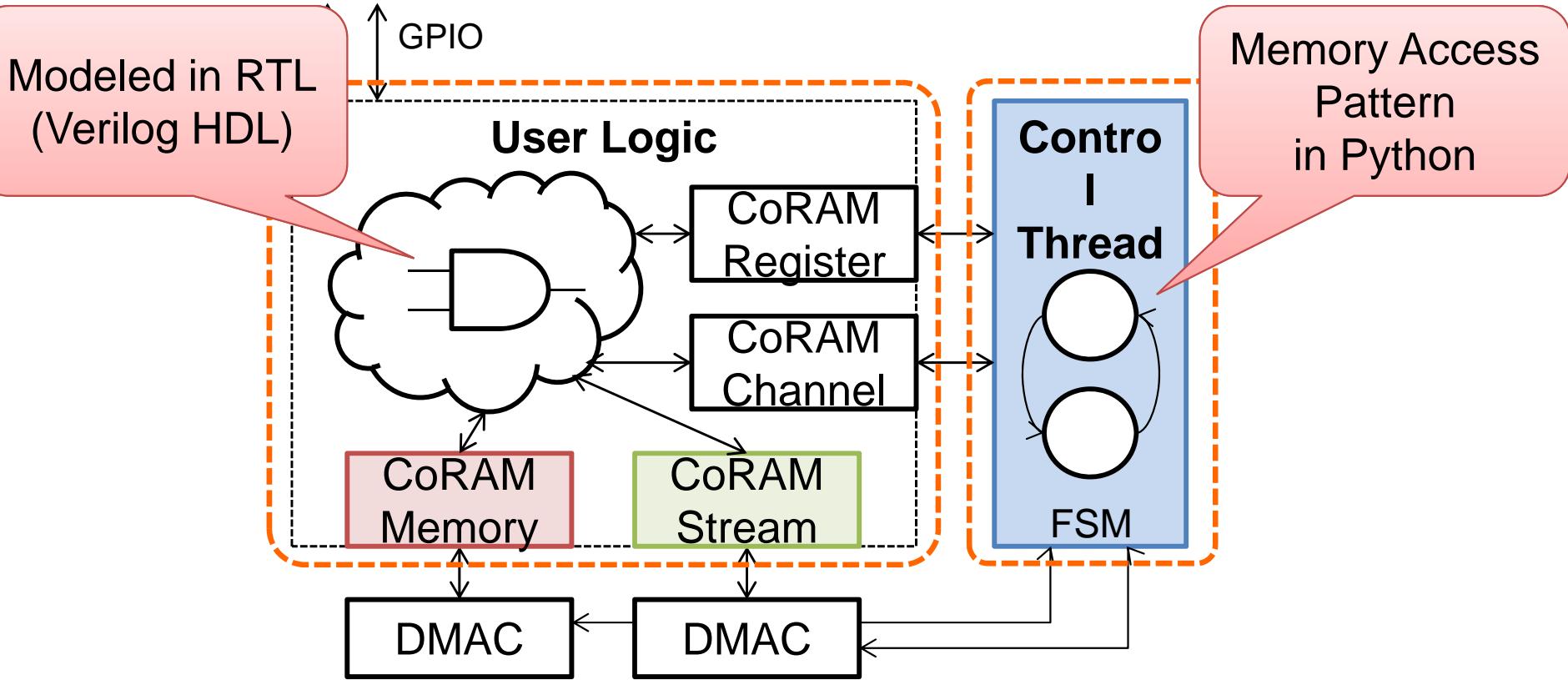
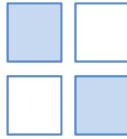
PyCoRAM

Microarchitecture

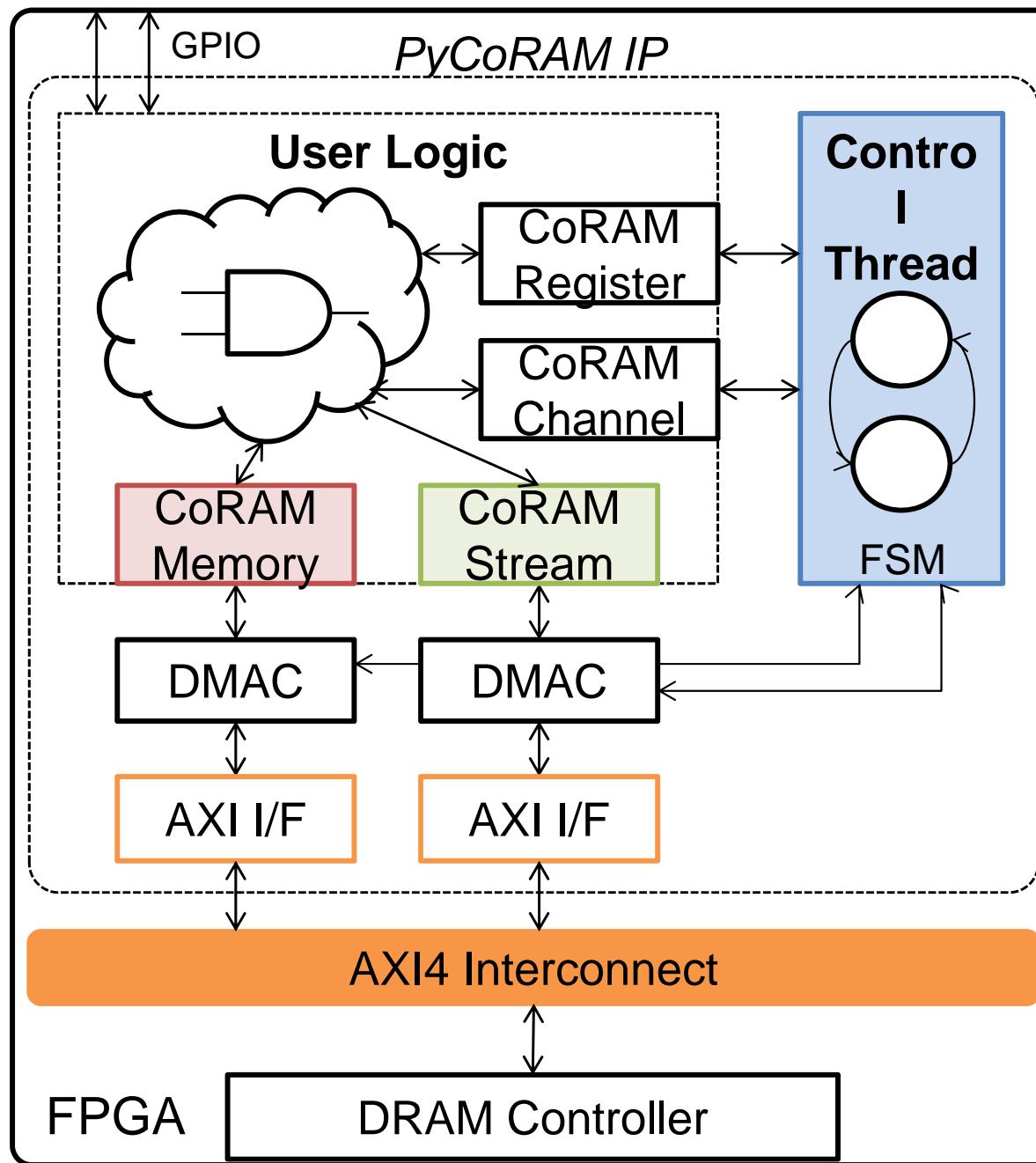
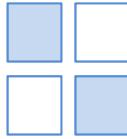
PyCoRAM Microarchitecture (Logical View)



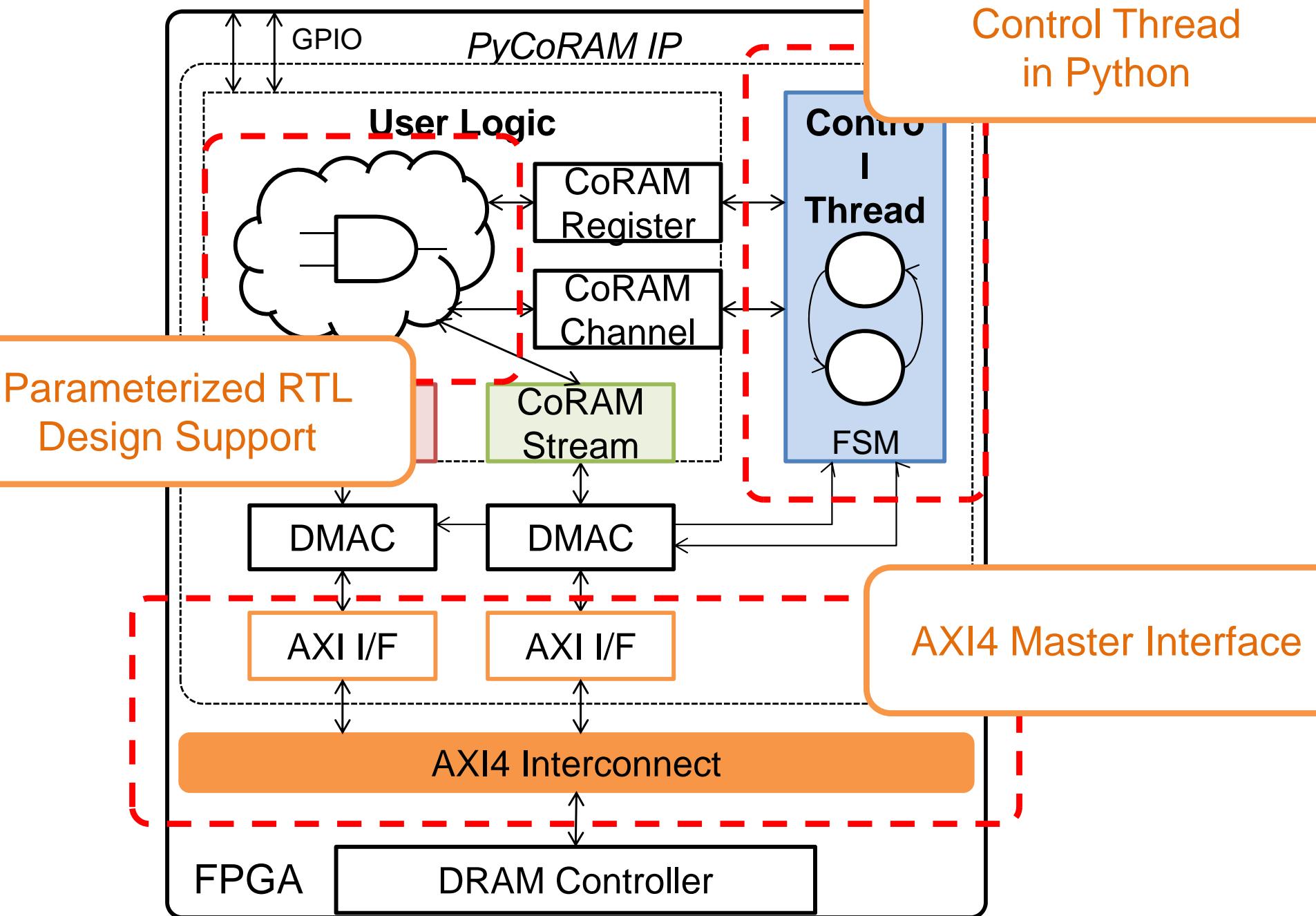
PyCoRAM Microarchitecture (Logical View)



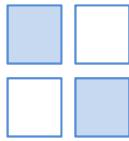
PyCoRAM Microarchitecture (Physical View)



PyCoRAM Microarchitecture (Physical View)

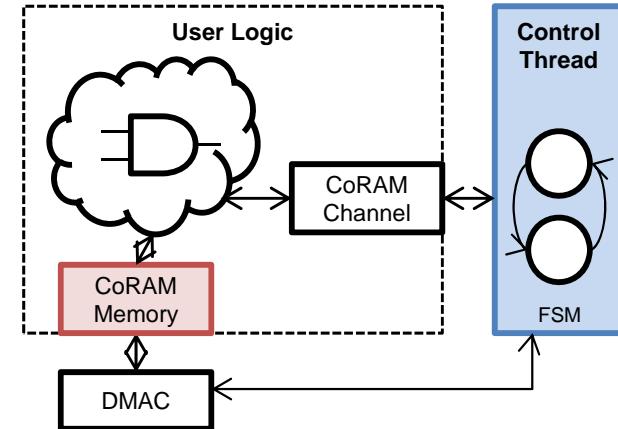


Control Thread in Python

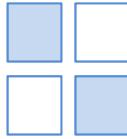


Operations for CoRAM objects

- To/from CoRAM Memory
 - Data movement pattern with DMA operations between on-chip CoRAM memory and DRAM
- To/from CoRAM Channel
 - Token communication action between user-logic and control thread



```
0 def calc_sum(times):  
1     ram = CoramMemory(idx=0, datawidth=32, size=1024)  
2     channel = CoramChannel(idx=0, datawidth=32)  
3     addr = 0  
4     sum = 0  
5     for i in range(times):  
6         ram.write(0, addr, 128)      # Transfer (off-chip DRAM to BRAM)  
7         channel.write(addr)       # Notification to User-logic  
8         sum += channel.read()    # Wait for Notification from User-logic  
9         addr += 128 * (32/8)  
10        print('sum=', sum)      # $display Verilog system task  
11    calc_sum(8)
```



CoRAM objects in User Logic

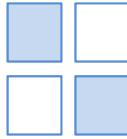
- CoRAM objects as standard BRAM or FIFO
 - Very similar interface to the standard memory components
 - User-logic can use their contents in them in the same way
- Essential parameters to define object characteristics
 - Thread name, ID, data width, address length, ...

```
CoramMemory1P
#(
    .CORAM_THREAD_NAME("thread_name"),
    .CORAM_ID(0),
    .CORAM_ADDR_LEN(ADDR_LEN),
    .CORAM_DATA_WIDTH(DATA_WIDTH)
)
inst_memory
(.CLK(CLK),
 .ADDR(mem_addr),
 .D(mem_d),
 .WE(mem_we),
 .Q(mem_q)
);
```

(a) CoRAM Memory

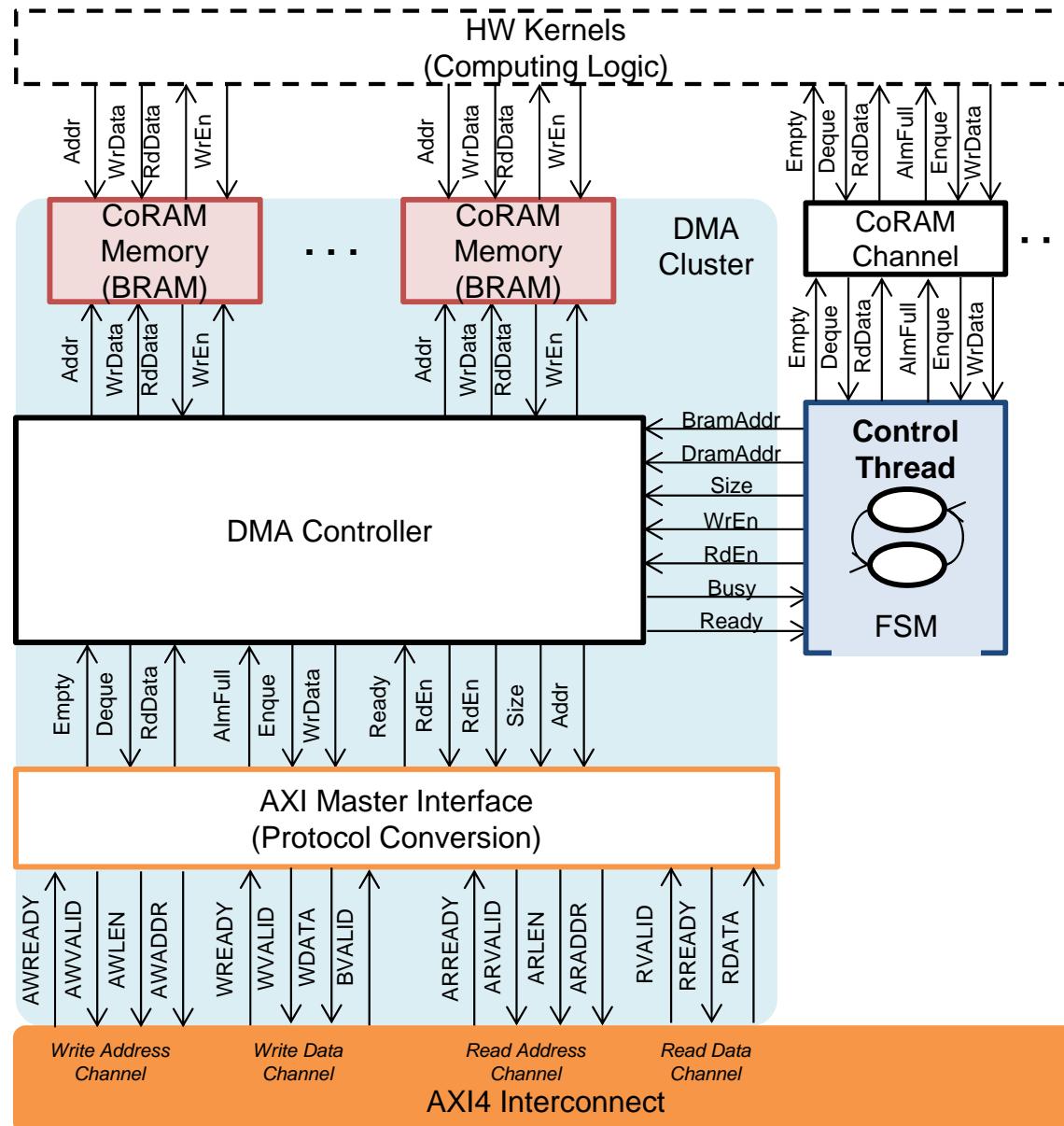
```
CoramChannel
#(
    .CORAM_THREAD_NAME("thread_name"),
    .CORAM_ID(0),
    .CORAM_ADDR_LEN(CHANNEL_ADDR_LEN),
    .CORAM_DATA_WIDTH(CHANNEL_DATA_WIDTH)
)
inst_channel
(.CLK(CLK),
 .RST(RST),
 .D(comm_d),
 .ENQ(comm_enq),
 .FULL(comm_full),
 .Q(comm_q),
 .DEQ(comm_deq),
 .EMPTY(comm_empty)
);
```

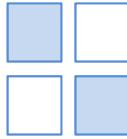
(b) CoRAM Channel



AXI4 Master Interface

- DMA controller works as AXI4 master IP-core interface





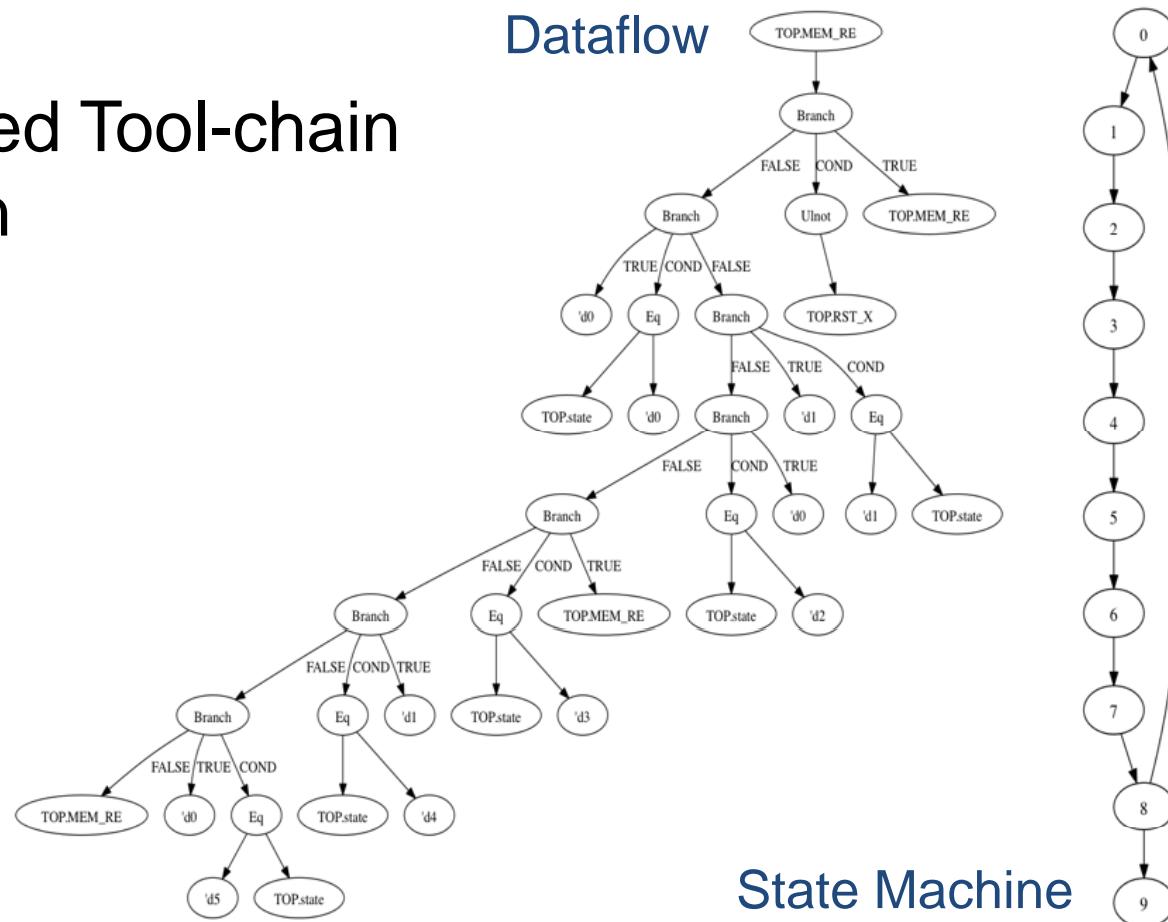
For Parameterized RTL design support

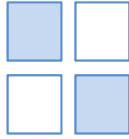
■ Generate-statement support by advanced RTL analyzer

- Not supported by the original CoRAM compiler

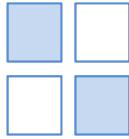
■ Pyverilog: Python-based Tool-chain for Verilog HDL Design

- Parser
- Dataflow Analysis
- Optimization
- RTL Code Generation
- Control flow Analysis
- Graphical Output





Evaluation



Evaluation

■ Point: Maximum memory bandwidth utilization

- PyCoRAM is a memory abstraction framework

■ Setup

- 2 FPGA boards

- Digilent Atlys
 - Spartan-6 LX45
 - DDR2-800 DRAM 128MB (1.2GB/s*)
*Due to 300MHz operation
 - AXI4 128-bit, 100MHz (1.6GB/s)

- Xilinx ML605
 - Virtex-6 LX240T
 - DDR3-800 DRAM 512MB (6.4GB/s)
 - AXI4 256-bit, 200MHz (6.4GB/s)

- EDK

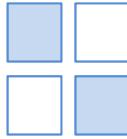
- Xilinx Platform Studio (14.6)



Digilent Atlys
(Xilinx Spartan-6 LX45)



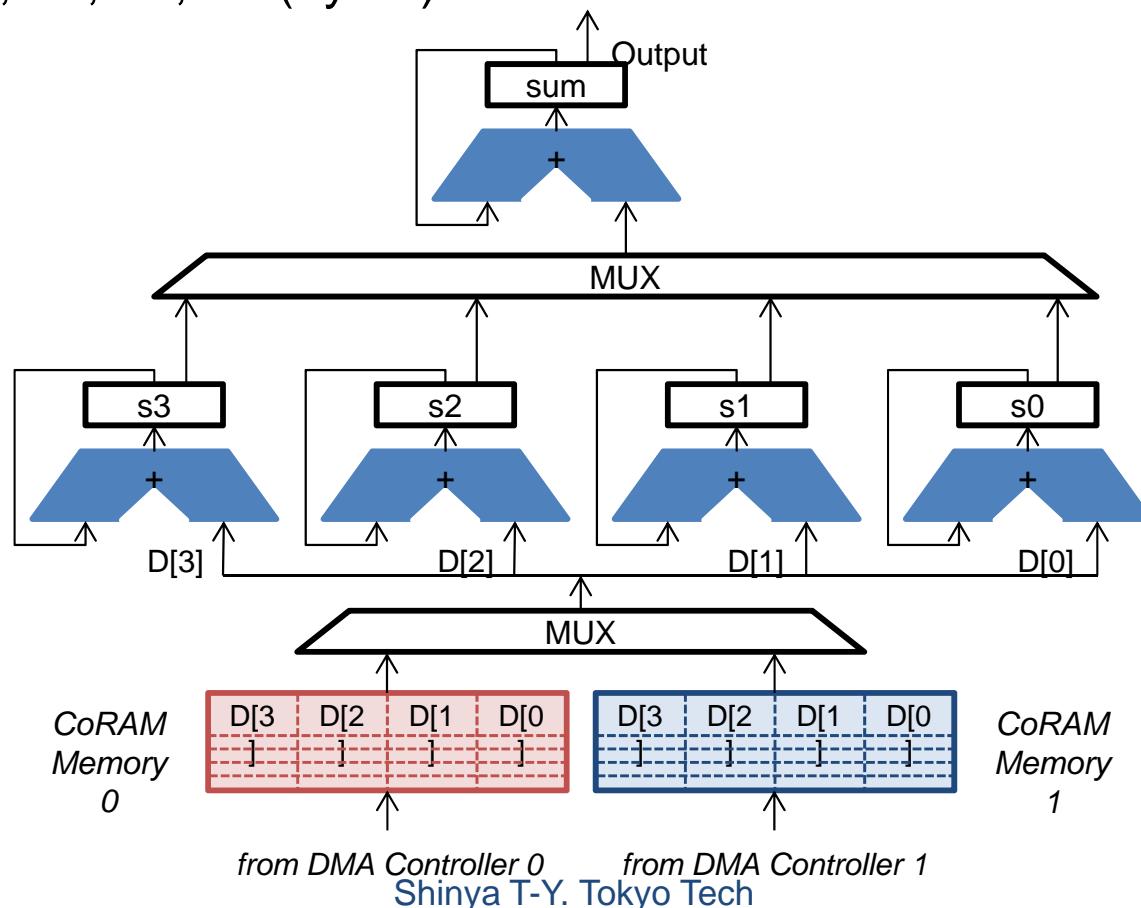
Xilinx ML605
(Xilinx Virtex-6 LX240T)

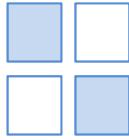


Evaluation: Application

■ Array-sum: calculate summation value of an array

- Two CoRAM memories as Double-buffered
- Varying SIMD width (=# simultaneous ops) to check the effect to the memory bandwidth utilization
 - 4, 8, 16, 32, 64 (bytes)





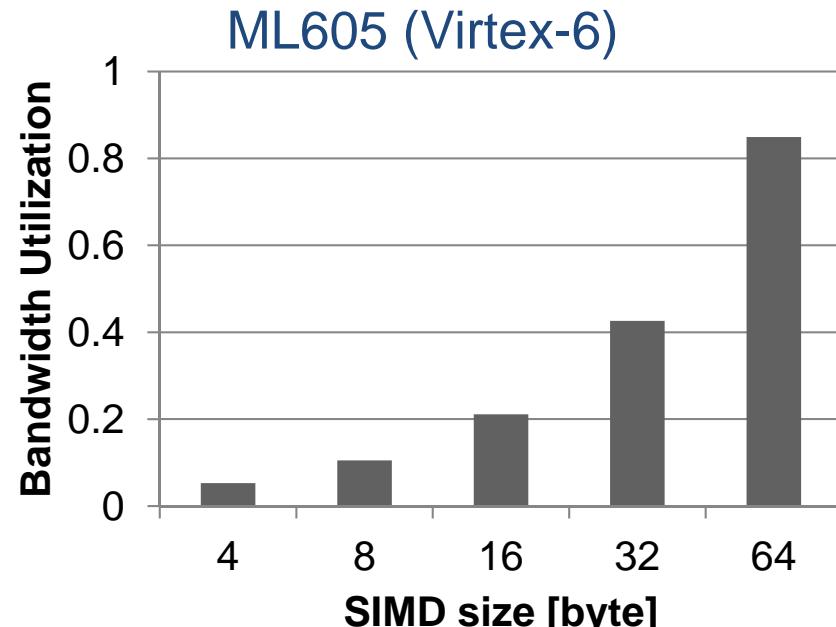
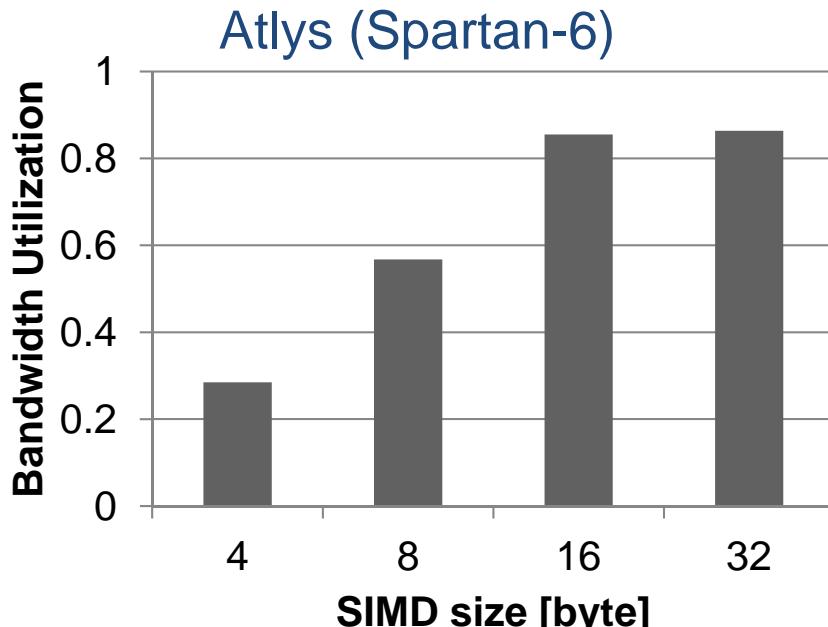
Memory Bandwidth Utilization

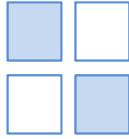
■ Good bandwidth utilization

- Atlys: 85.5% (at 16-byte)
- ML605: 84.9% (at 64-byte)

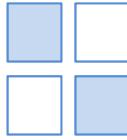
■ Degradation reasons

- Sequential (single) transaction for each DMA controller
 - Memory latency directly affects the performance adversely



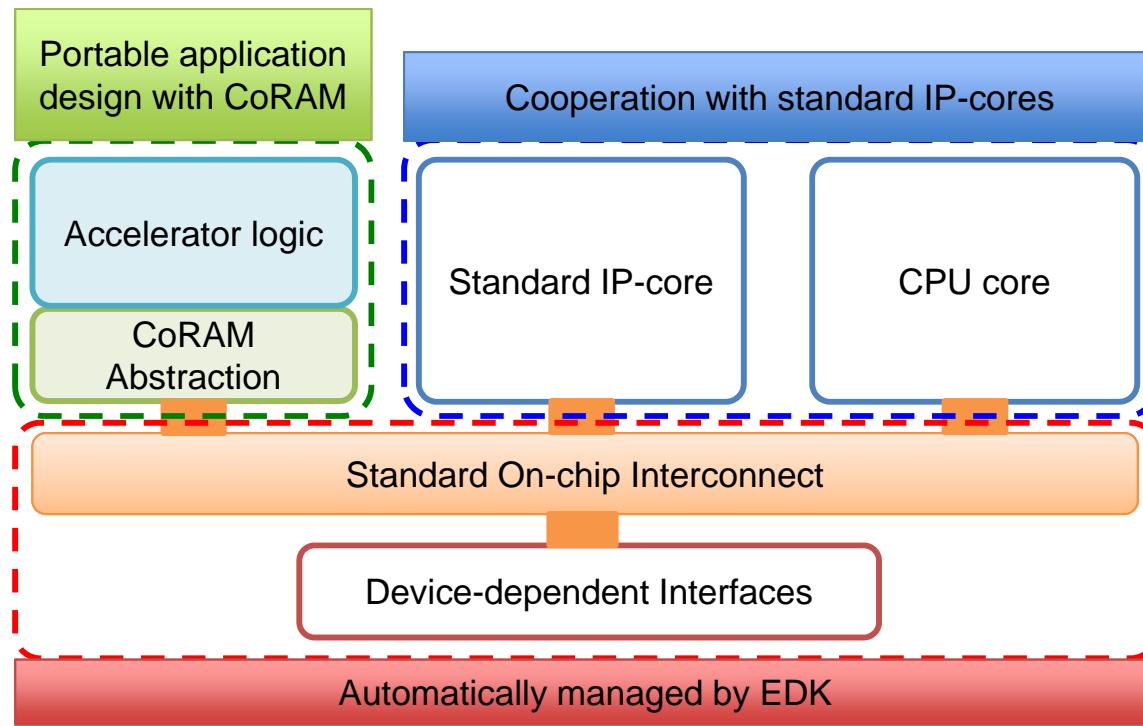


Conclusion and ...

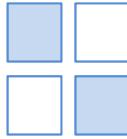


Conclusion

- **PyCoRAM**: Python-based implementation of CoRAM memory architecture for modern FPGA EDKs
- Future work
 - Further evaluation on more realistic applications
 - AXI4 slave feature for control thread
 - Tutorial slide😊



PyCoRAM and Pyverilog are ready for public!



■ PyCoRAM (0.7.0-public)

- <https://github.com/shtaxxx/PyCoRAM>

■ Pyverilog (0.6.0-public)

- <https://github.com/shtaxxx/Pyverilog>

Thanks!