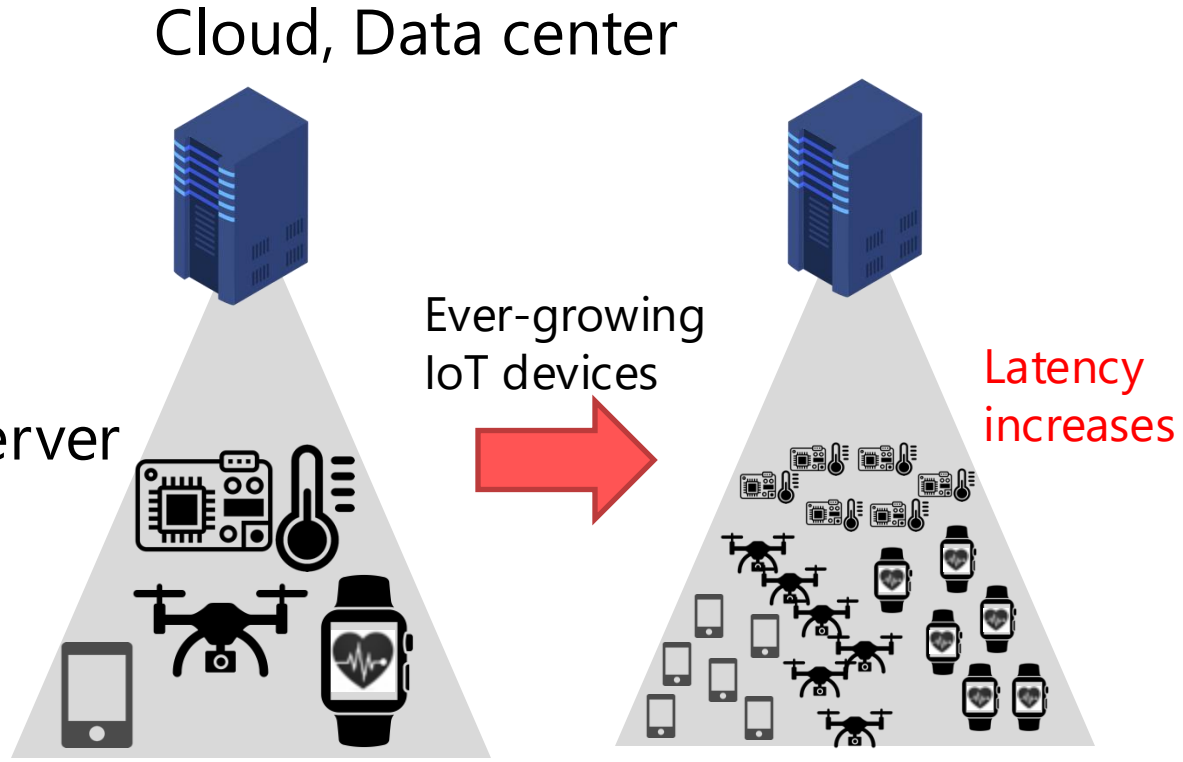
SLMLET: A RISC-V Processor SoC with Tightly-Coupled Area-Efficient eFPGA Blocks

Takuya Kojima[1], Yosuke Yanai[2] , Hayate Okuhara[3]
Hideharu Amano[2], Morihiro Kuga[4] , Masahiro Iida[4]

[1]The University of Tokyo, [2]Keio University,
[3]National University of Singapore, [4]Kumamoto University

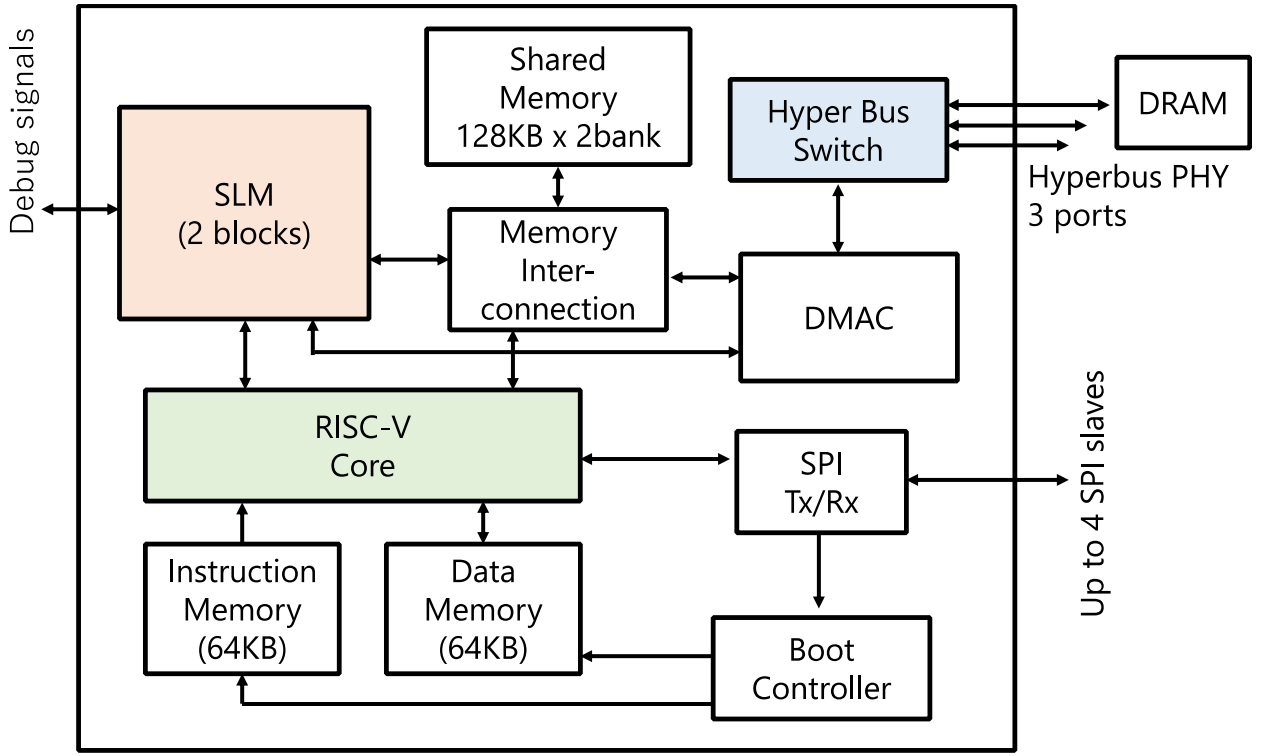
Paradigm Shift Towards *Edge Computing*

- Limited scalability of the conventional centralized cloud computing
  - Increases transferred data
  - Brings heavy computational loads
  - Increases processing latency
- *Edge Computing*
  - Offloads some processing to endpoint devices instead of Cloud server
- Emerging Demands for devices
  - High throughput
  - Low latency
  - Low energy Consumption



SLMLET: Our Proposed FPGA-CPU Hybrid SoC

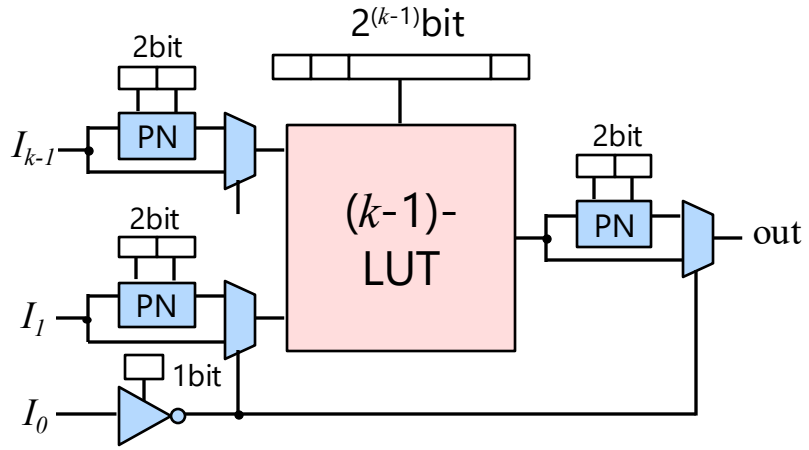
- CPU: RISC-V RV32I
  - Based on risc-v mini by UC Berkeley
  - Responsible for system control
  - Dedicated Inst./Data Mem. (Each 64KB)
- SLM (Scalable Logic Module)
  - Embedded FPGA Blocks
- HyperBus Interface
  - Compliant with JEDEC xSPI
- 2 banks of on-chip shared mem.
  - 128KB/bank
- SPI master interface
  - Boot loading & peripherals
  - Up to 4 channels



SLMLET SoCの構成

eFPGA Blocks based on SLM

- Scalable Logic Module(SLM)
  - Memory-saving logic cell Based on Shannon Expansion and NPN Equivalent Functions
  - K-input logic function realized with (K-1)-LUT & programmable NANDs (PNs)
- eFPGA IP Generator [1]
  - Allows customization of SLM input size, number of logic cells, etc.
  - Wilton-type Interconnection network



SLM Cell Structure for K-input logic

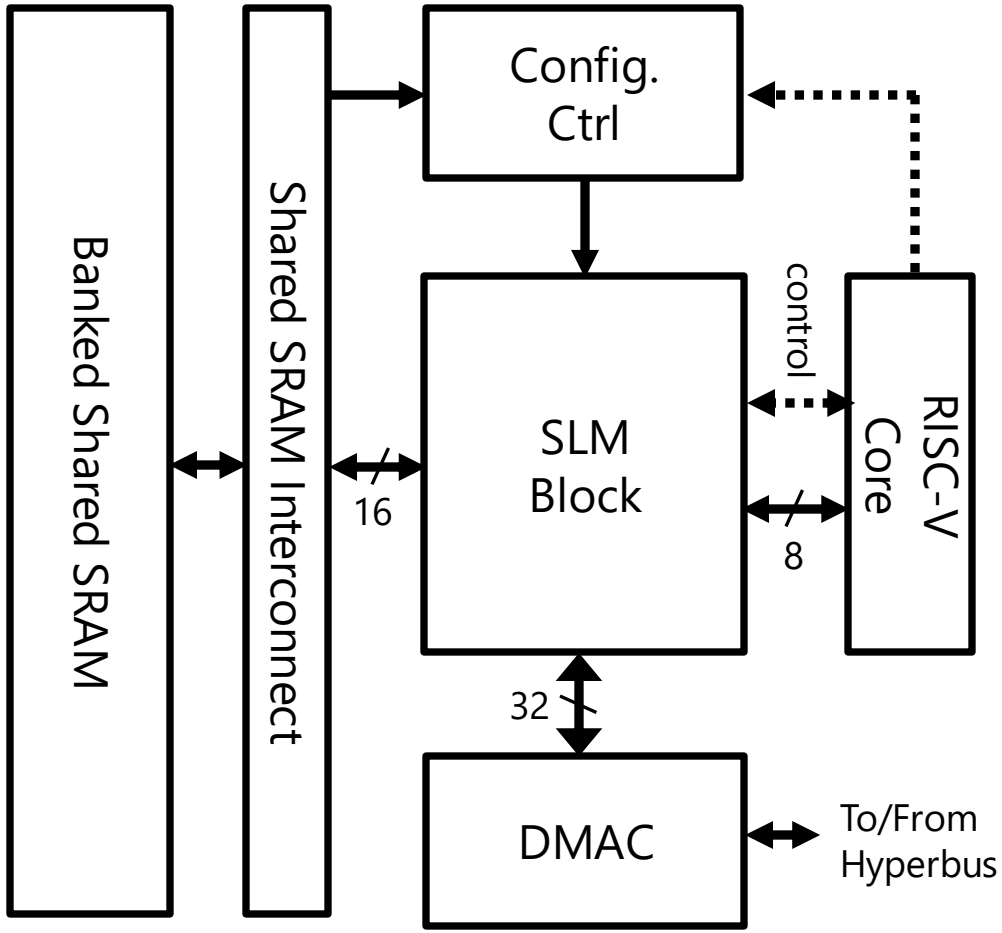
IP parameters employed for SLMLET

SLM Structure	SLM cells / LB	# of LB	Total SLMs	DSP block	Total FFs
5-SLM	4 BLE (SLM)	224	896	8	1024 bit

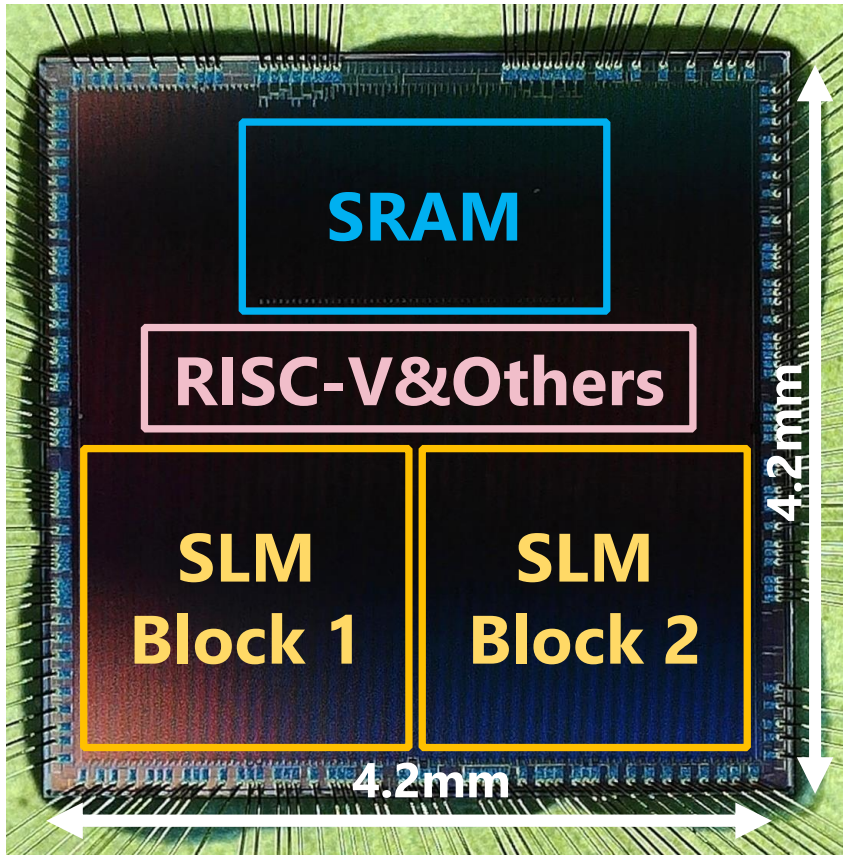
[1] Kuga, Morihiro, et al. "An eFPGA Generation Suite with Customizable Architecture and IDE." *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 106.3 (2023): 560-574.

SLM Block and Peripheral Modules

- Configuration Controller
  - Writing a Bitstream to the SLM Block
  - Parallel Reconfigurable available
- Interconnect between shared SRAM
  - Enables each module exclusive access
- Three difference data I/O of SLM Block
  1. Direct access by RISC-V core (CSR) 8bit width
  2. SRAM read/write 16bit data + 8bit address
  3. 32bit stream to/from DMAC
     - Enables the external DRAM via HyperBus I/F

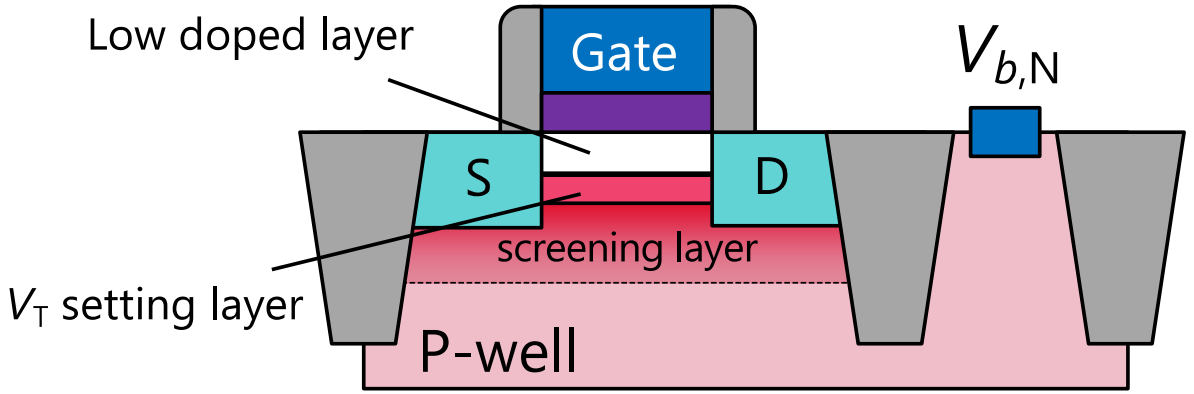


Prototype Implementation of SLMLET



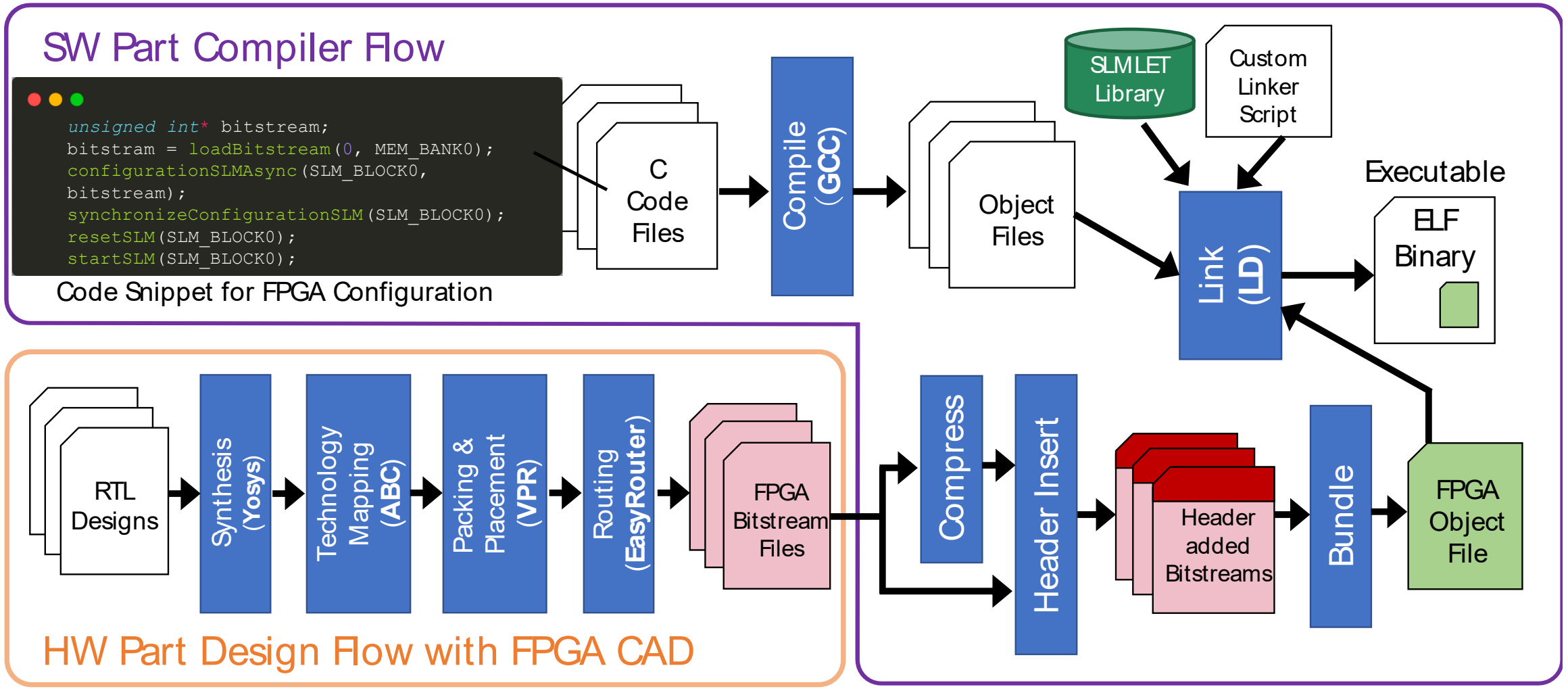
Chip photo of SLMLET

- Fabricated with USJC DDC 55nm
  - Small threshold voltage variation
  - 240mV/V of a high body effect coefficient
    - Exciting leakage reduction by body bias control
- Employed Library
  - C55DDCT07L60LVT



Transistor Structure of DDC Process

Our HW/SW development flow



Code example to use SLM Block

STEP 1: loading configuration data into shared memory

```
1  printf("load bitstream\n");
2  unsigned int* bitstream = loadBitstream(0, USE_SRAM_BANK);
3  if (!bitstream) {
4      printf("failed to load bitstream\n");
5      return 1;
6  }
```

STEP2 :Writing configuration data to SLM (Reconfiguration)

```
8  printf("start configuration\n");
9  configurationSLMAsync(SLM_BLOCK0, bitstream);
10 // do something
11 synchronizeConfigurationSLM(SLM_BLOCK0);
```

STEP 3: reset & enable signal to SLM Block

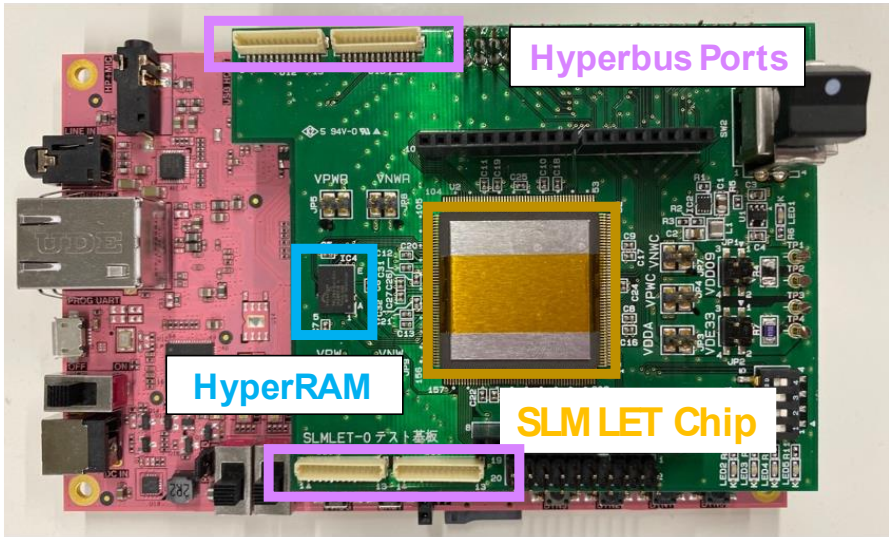
```
13 printf("enable SLM\n");
14 resetSLM(USE_SLM_BLOCK);
15 startSLM(USE_SLM_BLOCK);
```

STEP 4: Access to memory mapped registers on SLM

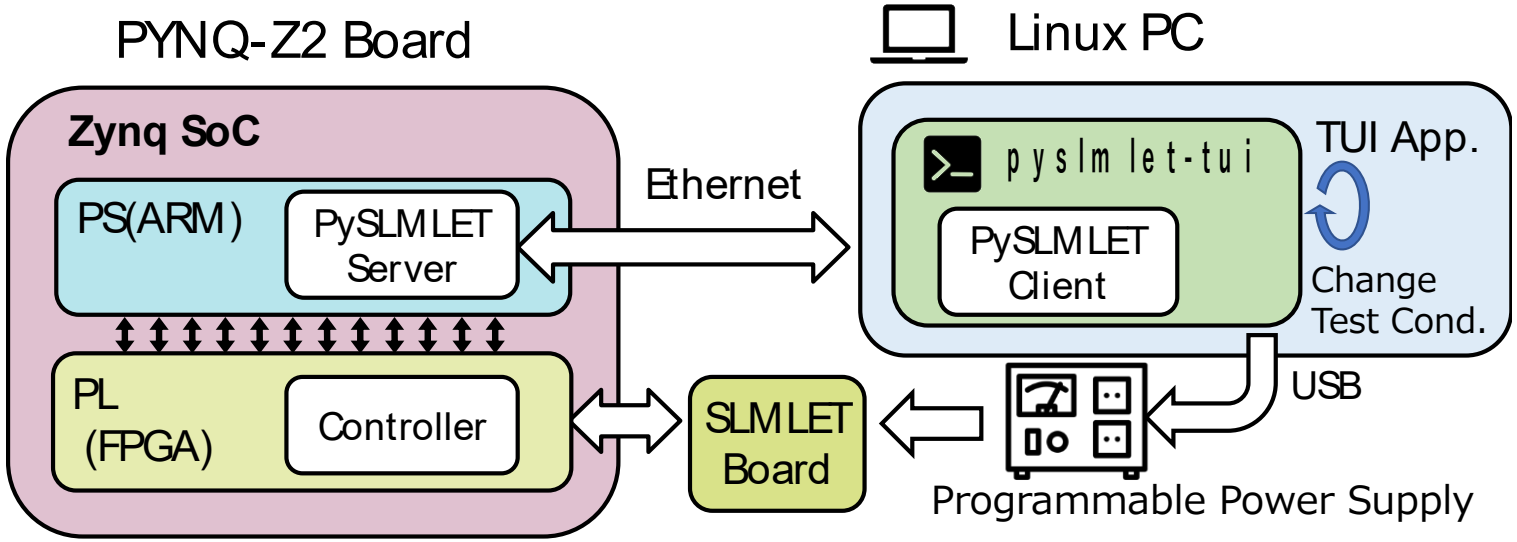
```
17 printf("write data to SLM\n");
18 for (i = 0; i < FPGA_REG_COUNT; i++) {
19     writeSLM(USE_SLM_BLOCK, (int*)(4 * i), &write_data[i]);
20 }
21
22 printf("read data from SLM:\n");
23 for (i = 0; i < FPGA_REG_COUNT; i++) {
24     printf("FPGA Reg %d: %08X\n", i, readSLM(USE_SLM_BLOCK, (int*)(4 * i)));
25 }
```

Our developed library offers useful APIs

Test & Evaluation Environment



SLMLET docked at PYNQ-Z2



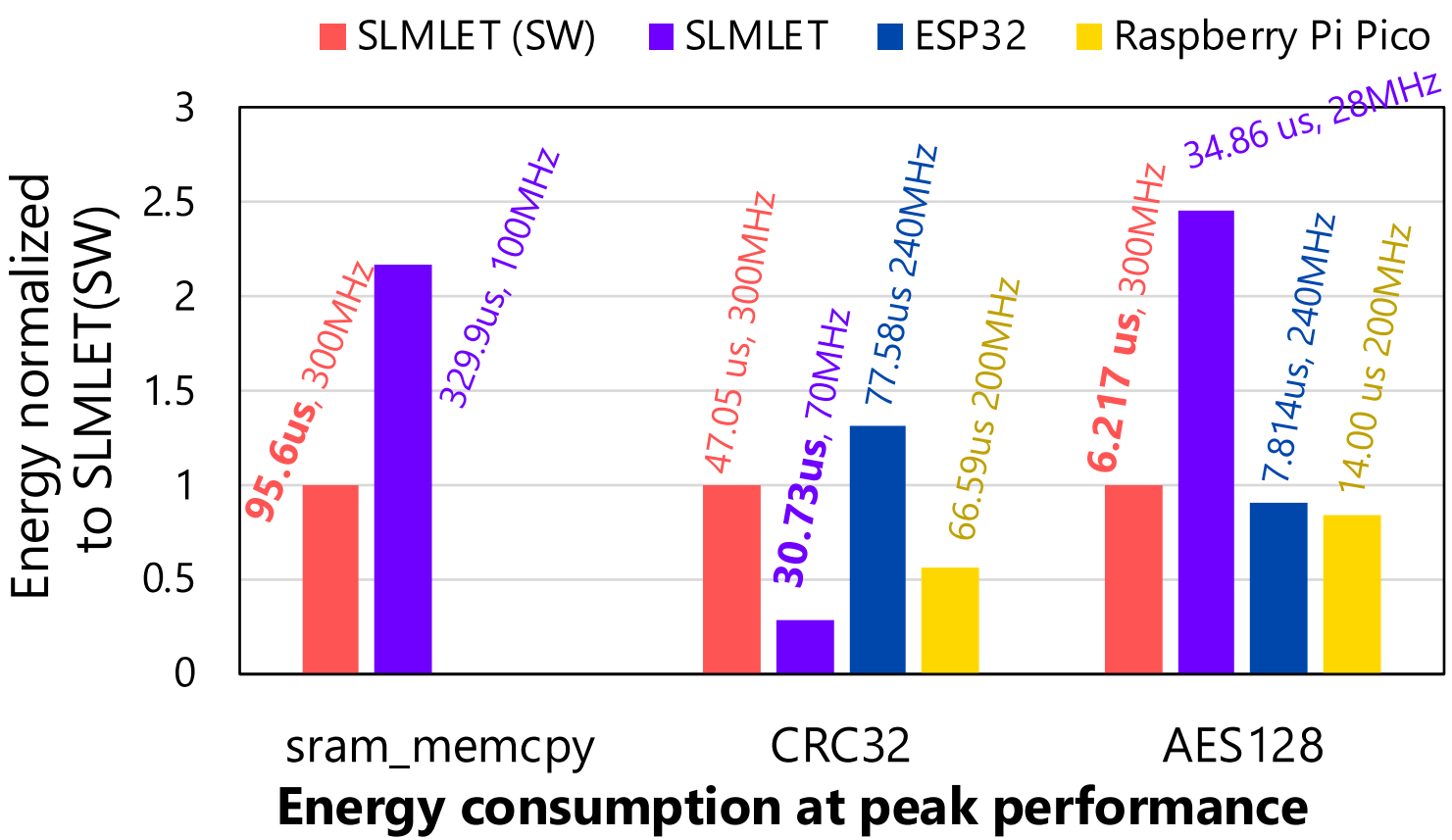
Fully automated test and evaluation

- PySLMLET
  - PYNQ-Z2 board works as SLMLET host
  - Developed Python driver to control HW on PYNQ-Z2 board
- PySLMLET-TUI for test automation
  - Text-based UI for testing with variable conditions (running software binaries, voltage, operational frequency, etc.)

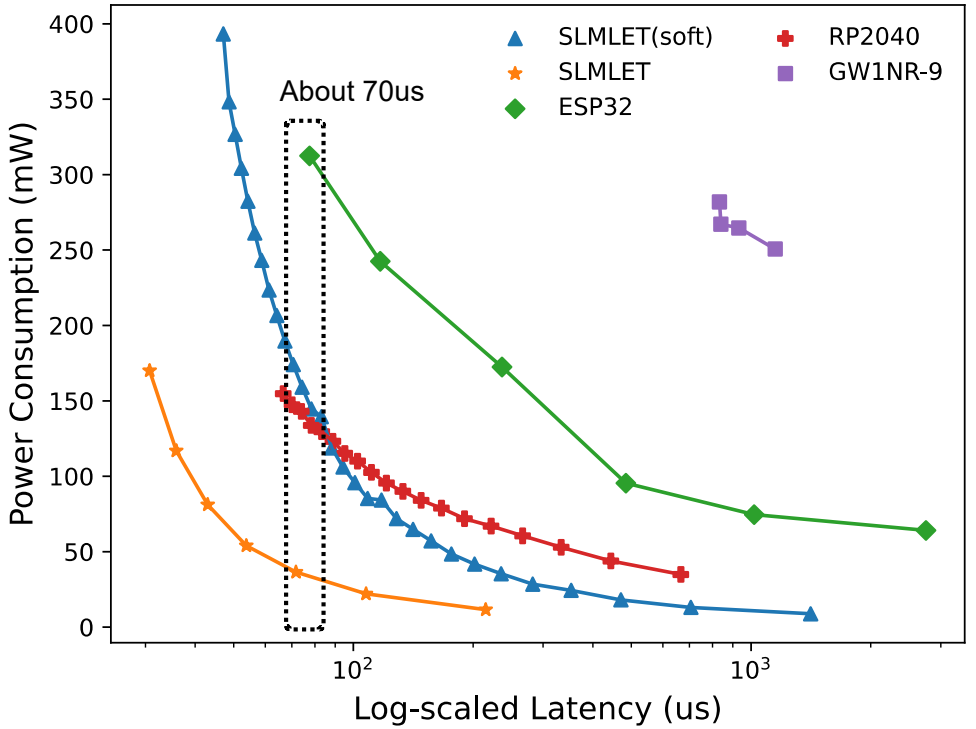
Evaluation benchmarks

- Used three applications
  1. *sram_memcpy*
     - Copies 16 KB data block on the shared SRAM
  2. *CRC32*
     - Computes CRC32 value for the given 1KB of binary data (polynomial 0x04C11DB7)
  3. *AES128*
     - Encrypts one block of 128-bit plain texts
- SLM block part is designed with RTL implementation
- Software-only cases uses MiBench source codes

Power and energy consumption comparison



Energy consumption at peak performance



Power consumption for each frequency

- HW-enabled design archives best energy consumption for CRC32
- However, it limits performance for sram_memcpy because of 16bit width memory I/F and AES128 because of SLM resource limitation and long critical paths.
- Software-only design for AES128 still outperforms the other two commercial micro-controllers (ESP32 and Raspberry Pi Pico)

Conclusion

- We proposed SLMLET, an SoC with a RISC-V core and eFPGA featuring SLM reconfigurable logic
- We also developed HW/SW co-design flow and software library for SLMLET
- Experimental results show
  - In benchmarks like CRC32, where the benefits of hardware implementation are significant, both latency reduction and energy savings are archived
  - RISC-V core outperforms commercially available micro-controllers
- Future works
  - Removing the performance bottlenecks and fully harnessing the capability of SLM blocks for more applications
  - Updating the design flow and library to support multi-SLMLET platform interconnected via HyperBus