SOC Lab Final Project

What we do

- Accelerators for Qsort, FIR, and Matmult tasks.
- DMA controller with concurrent read-write ability.
- Arbiter for maintaining both DMA and CPU access.
- Modified Stream protocol with lower cost.
- Modified UART Hardware with better performance.
- Run two workloads simultaneously.
- Verification by software.

Team - 07

312611096	黃鉦淳
312605003	王語
511506022	何佳玲

Partner Team – 16

311511082	郭銘宸
311512065	郭紘碩

Outline

Result

- QoR of Qsort, FIR, Matmult ,and UART.

Design

- Architecture overview
- Data flow and Transmission Protocol
- DMA controller design
- Memory system design and trade-off

Insight

- Design review

Performance Compare

	Qsort	FIR	MatMul	workload	UART
Original	26,145 cycles	70,406 cycles	32,988 cycles	131,721 cycles	38 cycles
Our project	114 cycles	216 cycles	157 cycles	936 cycles	24 cycles
Compare	230%	325%	210%	140%	31.57%

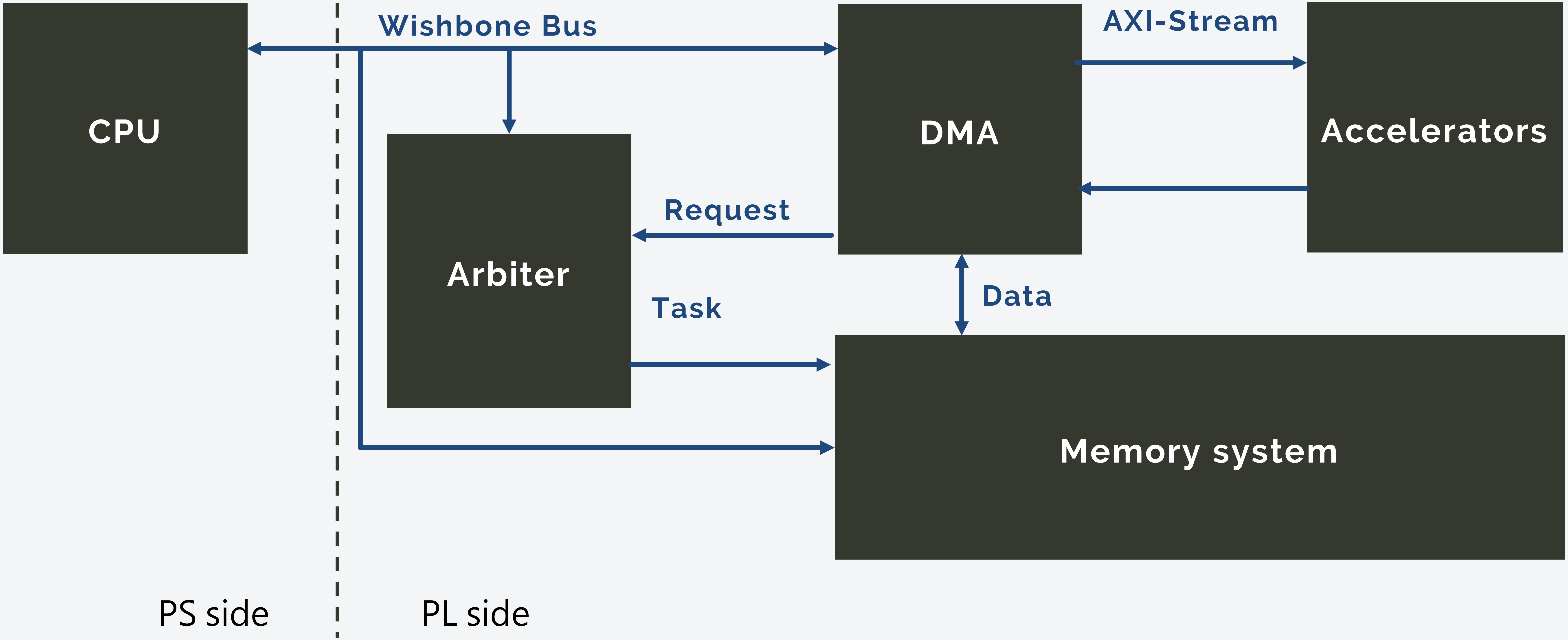
Workflow

- CPU initialize
- CPU moves instructions and data from the flask to BRAM (exmem).
- CPU reads instructions and programs DMA.
- DMA moves data from BRAM to the accelerator.
- DMA moves data from the accelerator to the BRAM.
- CPU reads DMA information.
- CPU reads data and verifies it by software.

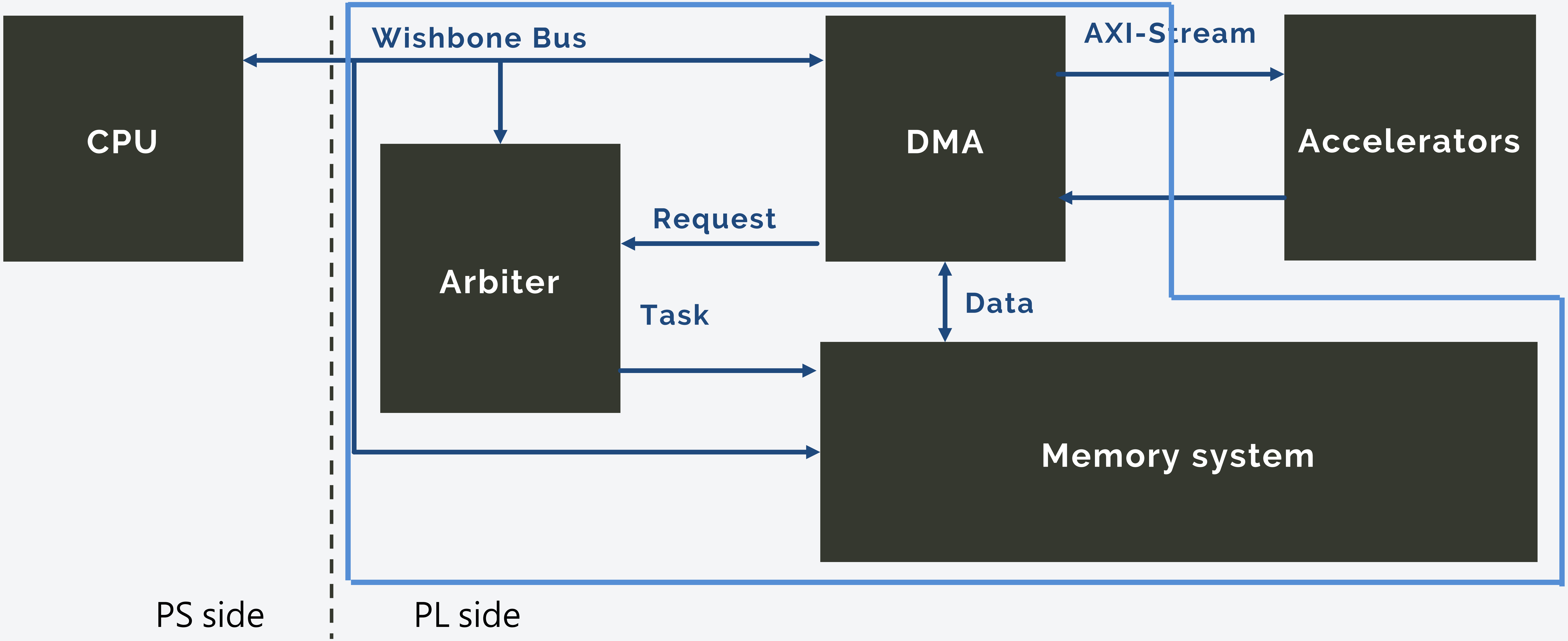
Workflow

```
void main() {
    // mprj_init
    // la_init
    // uart_interrupt_init
    // apply_config
    for(int i = 0; i < TIMES_RERUN; i++)
        {
                // Workload
                fir(); matmul(); qsort();
                // Workload_check
                fir_check(); matmul_check(); qsort_check();
        }
}
```

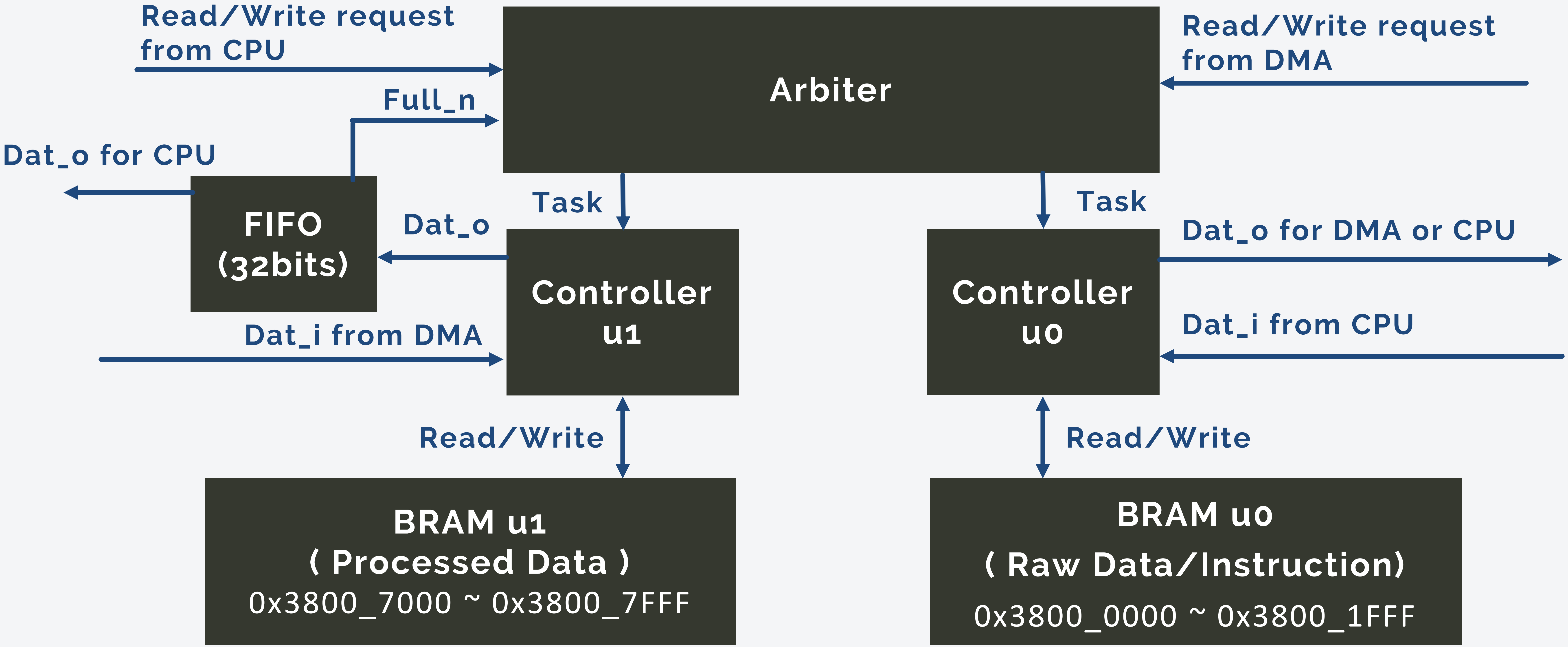
Architecture



Architecture



Memory System



Read/Write request from CPU

Full_n

Dat_o for CPU

FIFO (32bits)

Dat_o

Dat_i from DMA

Arbiter

Read/Write request from DMA

Task

Task

Controller u1

Controller u0

Dat_o for DMA or CPU

Dat_i from CPU

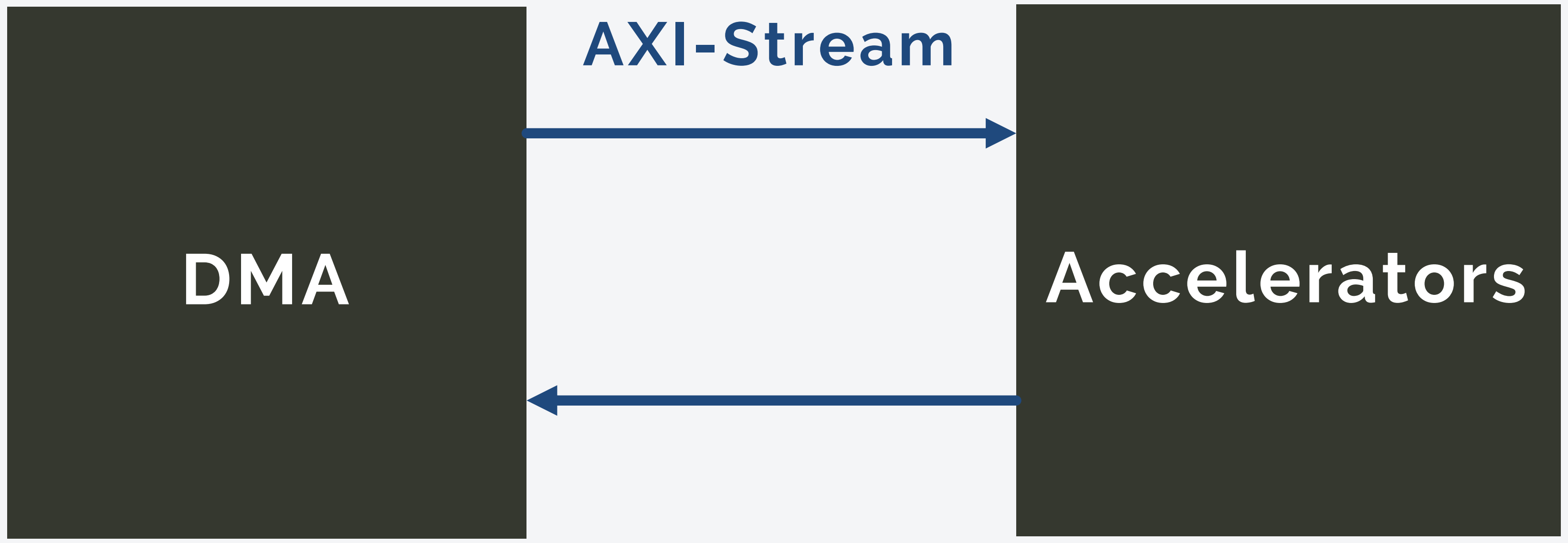
Read/Write

Read/Write

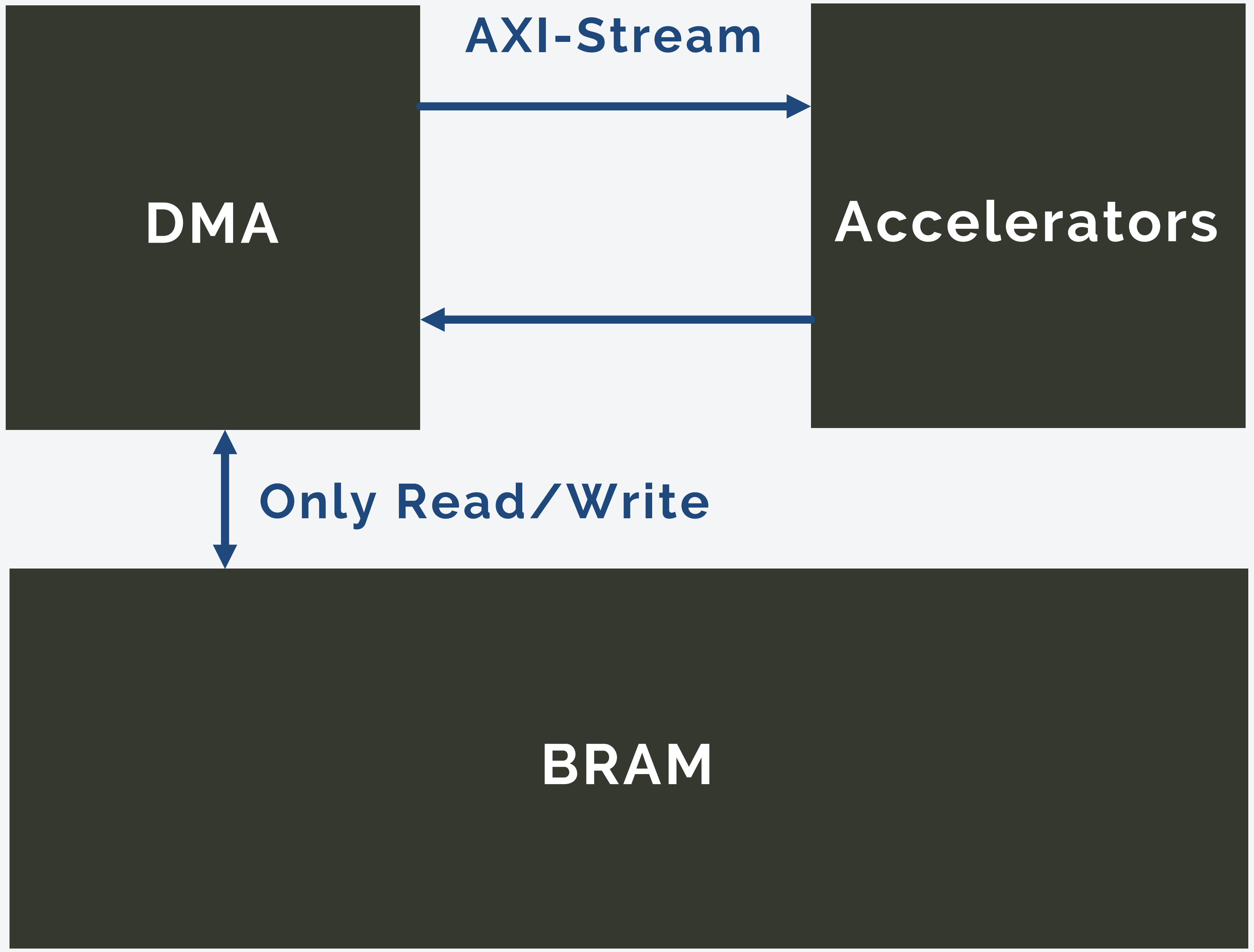
BRAM u1
(Processed Data)
0x3800_7000 ~ 0x3800_7FFF

BRAM u0
(Raw Data/Instruction)
0x3800_0000 ~ 0x3800_1FFF

Why use two BRAM



Why use two BRAM



DMA

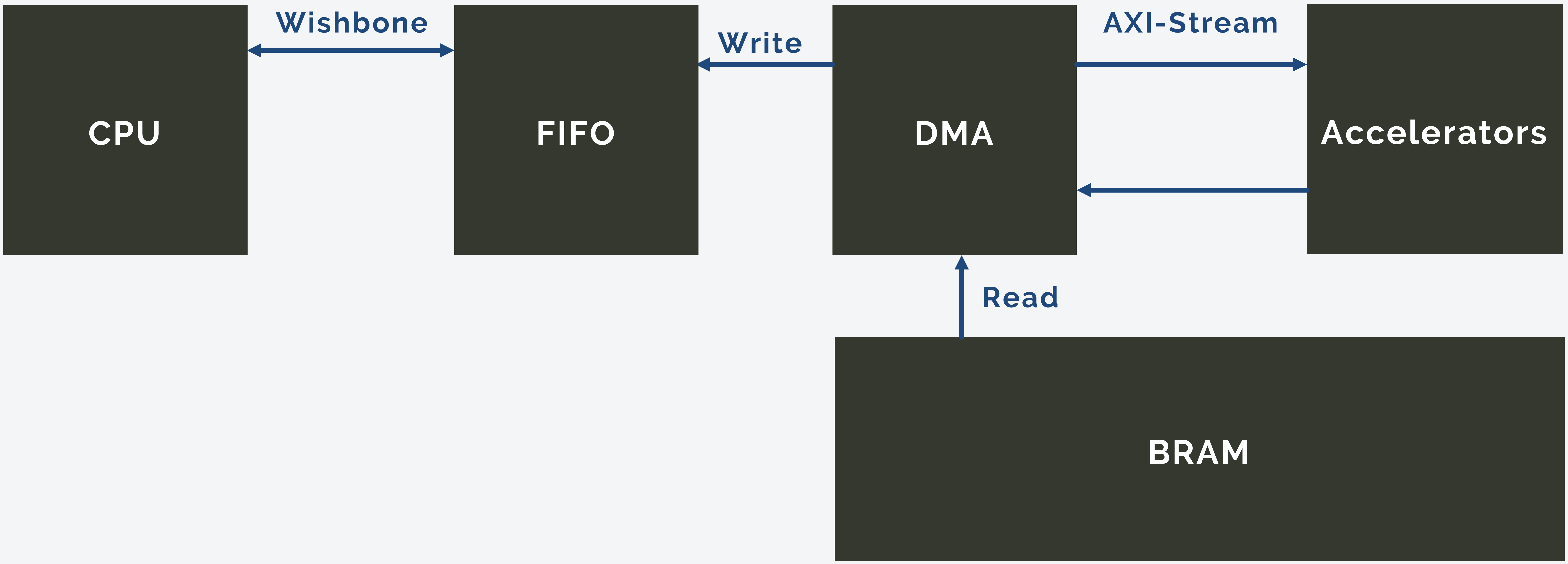
Accelerators

AXI-Stream

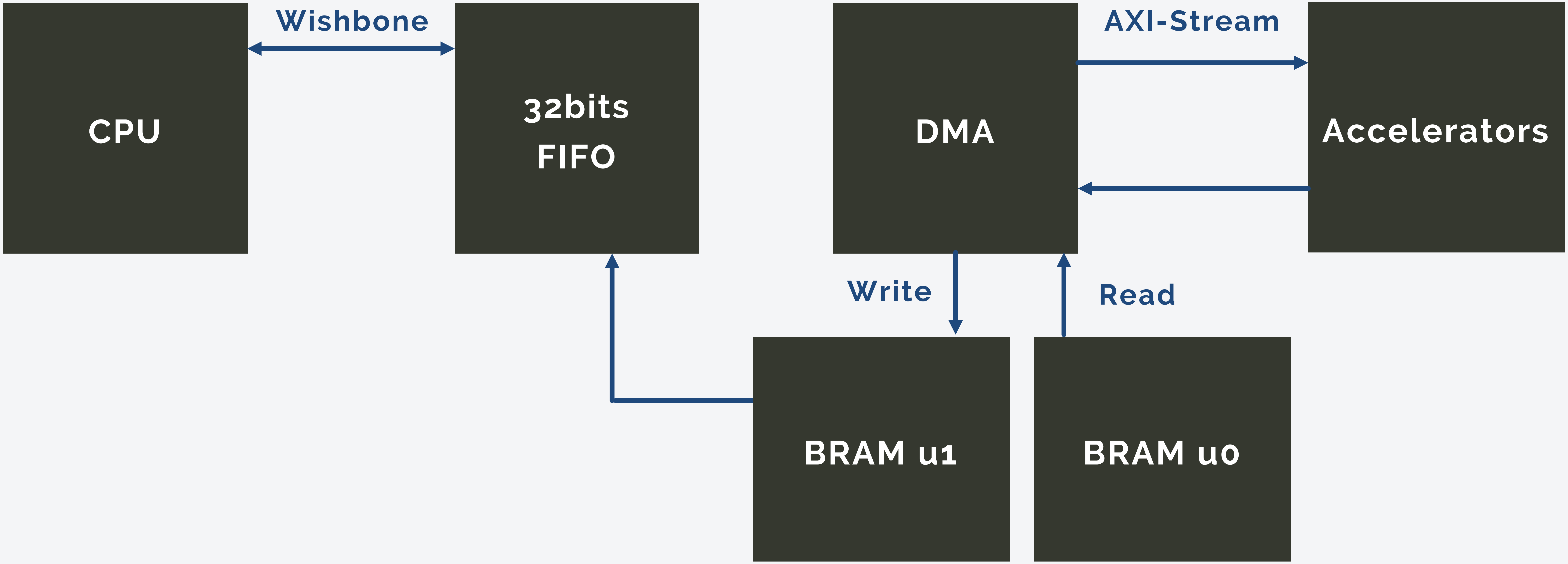
Only Read/Write

BRAM

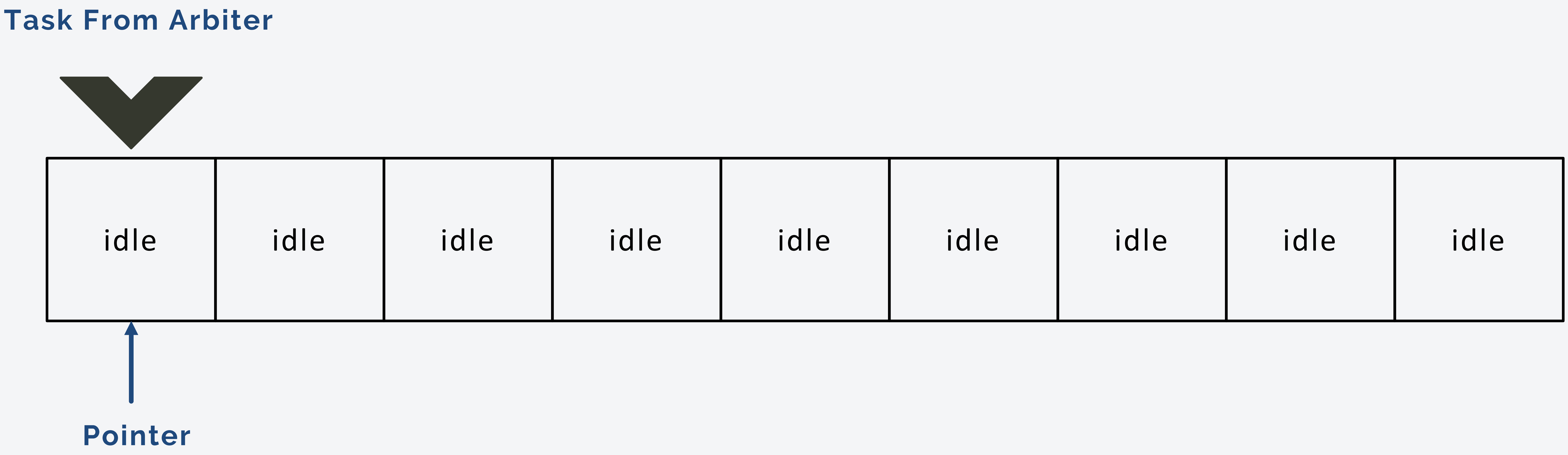
Why use two BRAM



Why use two BRAM



BRAM Controller



Task From Arbiter

idle	idle	idle	idle	idle	idle	idle	idle	idle

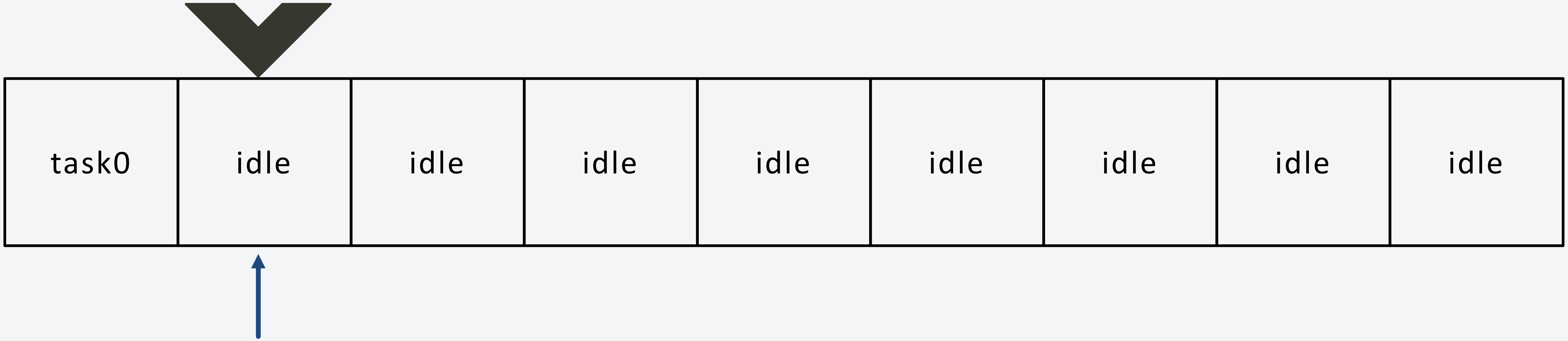
Pointer

BRAM Execution : IDLE

oT

BRAM Controller

Task From Arbiter



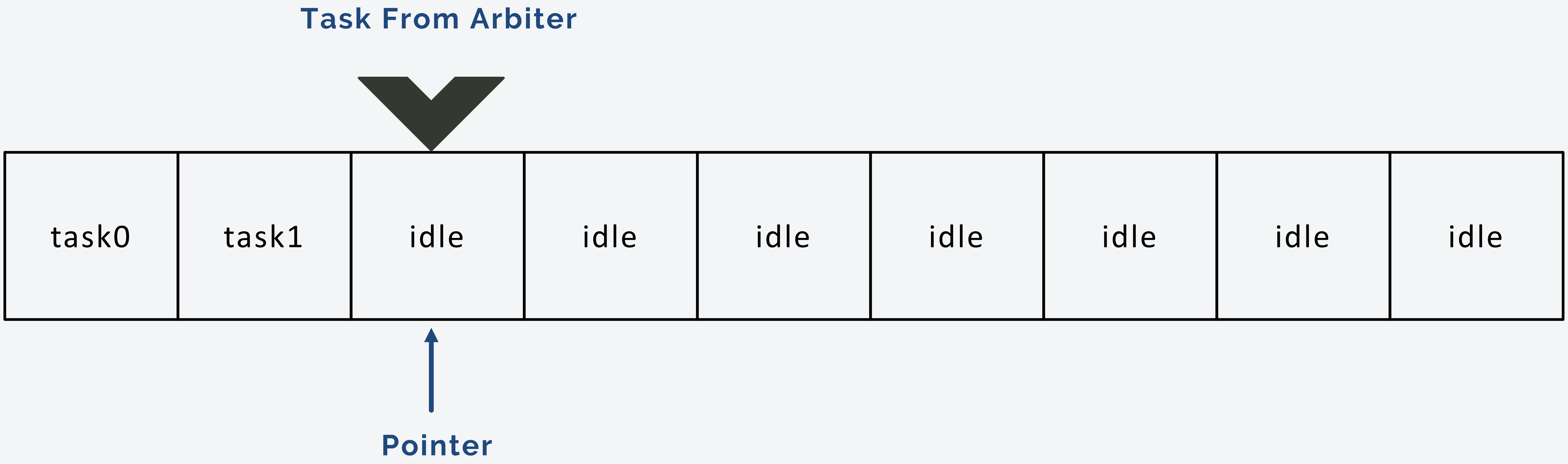
task0	idle	idle	idle	idle	idle	idle	idle	idle

Pointer

BRAM Execution : IDLE

1T

BRAM Controller



Task From Arbiter

task0	task1	idle	idle	idle	idle	idle	idle	idle

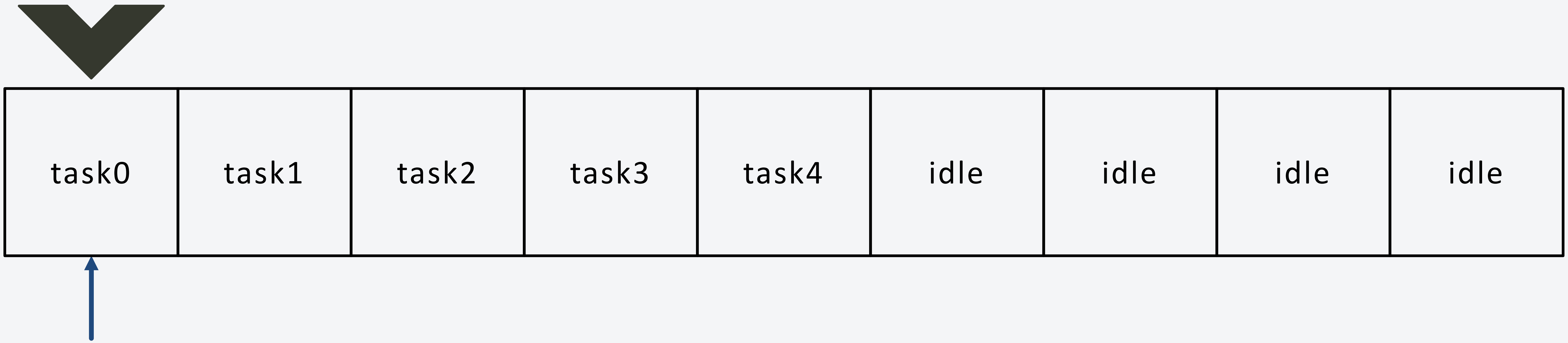
Pointer

BRAM Execution : IDLE

2T

BRAM Controller

Task From Arbiter



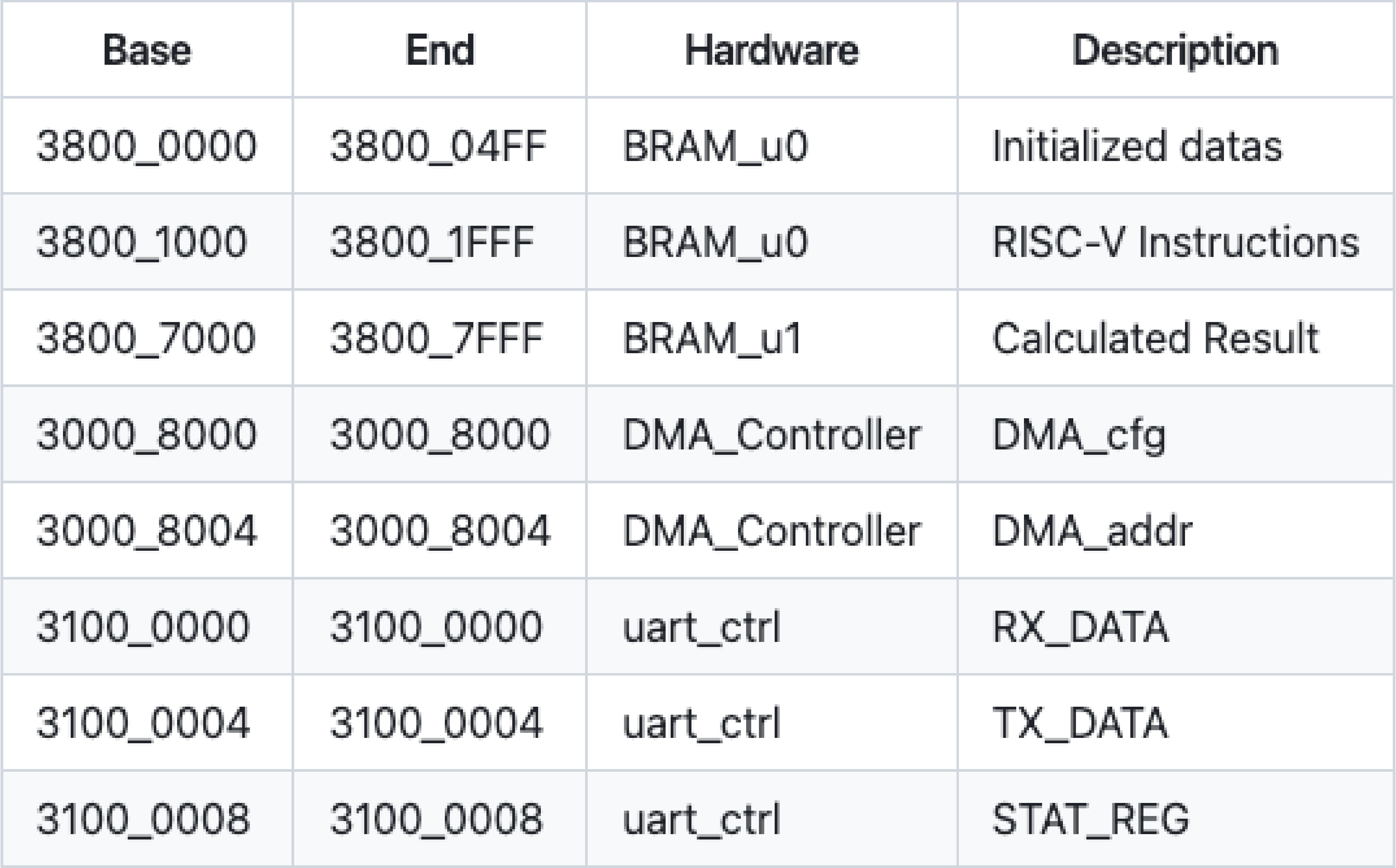
task0	task1	task2	task3	task4	idle	idle	idle	idle

Pointer

BRAM Execution : task0

9T

Memory Map



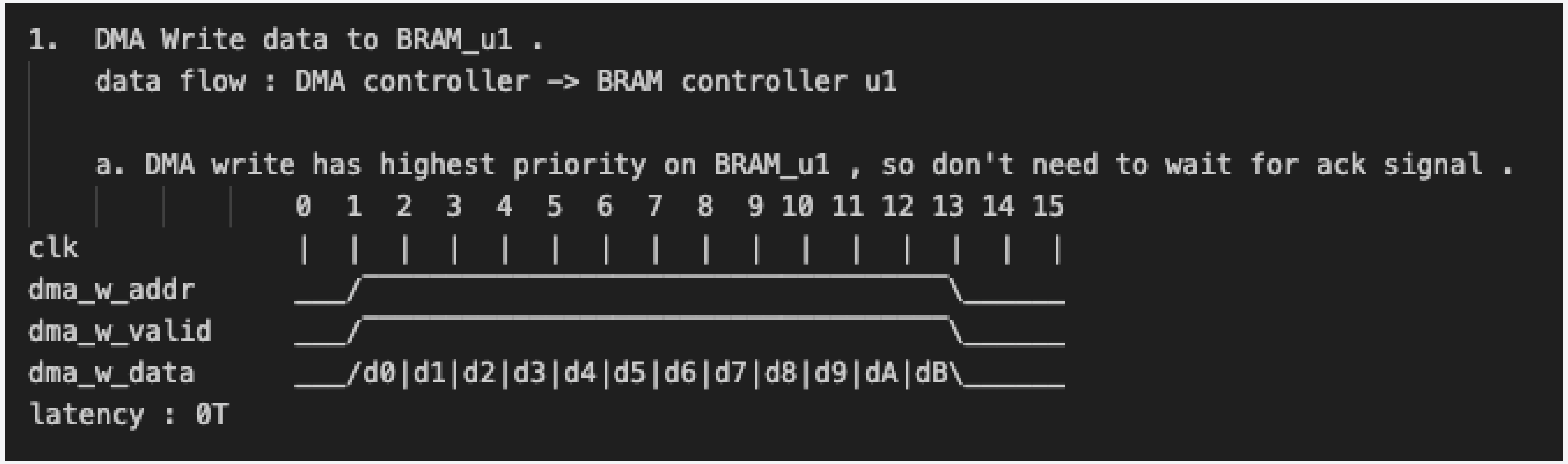
Base	End	Hardware	Description
3800_0000	3800_04FF	BRAM_u0	Initialized datas
3800_1000	3800_1FFF	BRAM_u0	RISC-V Instructions
3800_7000	3800_7FFF	BRAM_u1	Calculated Result
3000_8000	3000_8000	DMA_Controller	DMA_cfg
3000_8004	3000_8004	DMA_Controller	DMA_addr
3100_0000	3100_0000	uart_ctrl	RX_DATA
3100_0004	3100_0004	uart_ctrl	TX_DATA
3100_0008	3100_0008	uart_ctrl	STAT_REG

Access Priority

BRAM u1 (Processed Data) 0x3800_7000 ~ 0x3800_7FFF	BRAM u0 (Raw Data/Instruction) 0x3800_0000 ~ 0x3800_1FFF
DMA Write	CPU Write
CPU Read	CPU Prefetch
	DMA Read
	CPU Read

Modified Stream protocol

DMA Write

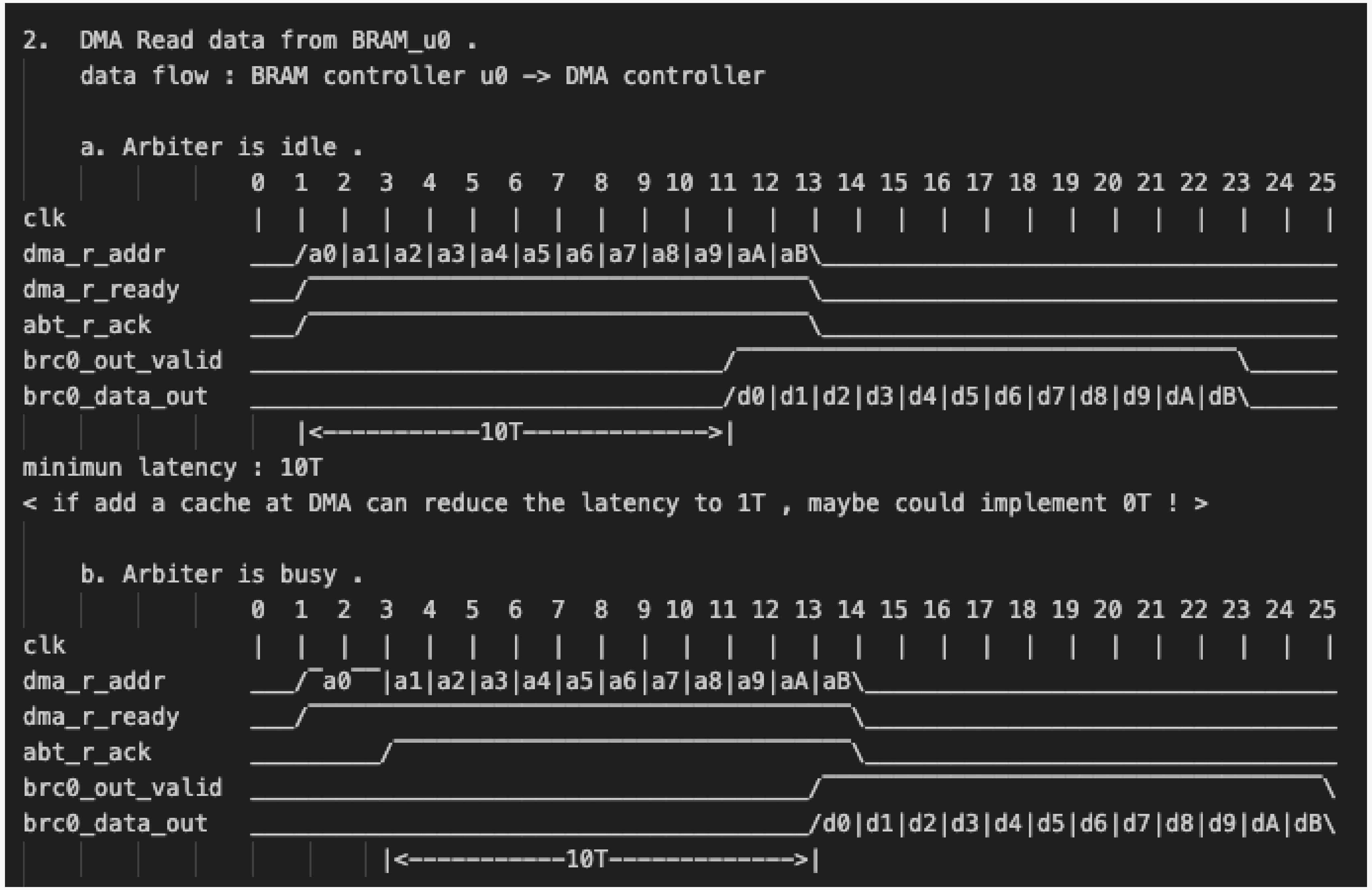


```
1.  DMA Write data to BRAM_u1 .
    data flow : DMA controller -> BRAM controller u1

    a. DMA write has highest priority on BRAM_u1 , so don't need to wait for ack signal .
                0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
clk             |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
dma_w_addr      ___/‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾\______
dma_w_valid     ___/‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾\______
dma_w_data      ___/d0|d1|d2|d3|d4|d5|d6|d7|d8|d9|dA|dB\______
latency : 0T
```

Modified Stream protocol

DMA Read



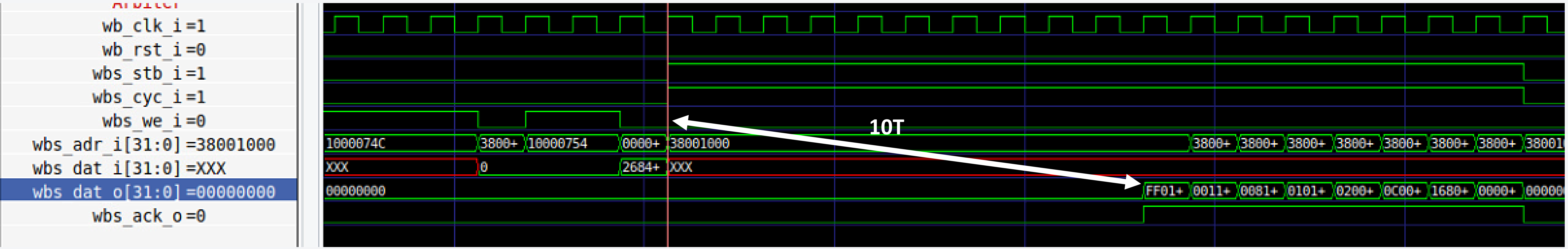
```
2.  DMA Read data from BRAM_u0 .
    data flow : BRAM controller u0 -> DMA controller

    a. Arbiter is idle .
                0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
clk             |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
dma_r_addr      ___/a0|a1|a2|a3|a4|a5|a6|a7|a8|a9|aA|aB\____________________________________
dma_r_ready     ___/‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾\____________________________________
abt_r_ack       ___/‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾\____________________________________
brc0_out_valid  _________________________________/‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾\______
brc0_data_out   _________________________________/d0|d1|d2|d3|d4|d5|d6|d7|d8|d9|dA|dB\______
                   |<-----------10T------------->|
minimun latency : 10T
< if add a cache at DMA can reduce the latency to 1T , maybe could implement 0T ! >

    b. Arbiter is busy .
                0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
clk             |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
dma_r_addr      ___/‾a0‾‾|a1|a2|a3|a4|a5|a6|a7|a8|a9|aA|aB\_________________________________
dma_r_ready     ___/‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾\_________________________________
abt_r_ack       _________/‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾\_________________________________
brc0_out_valid  _______________________________________/‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾\
brc0_data_out   _______________________________________/d0|d1|d2|d3|d4|d5|d6|d7|d8|d9|dA|dB\
                         |<-----------10T------------->|
```

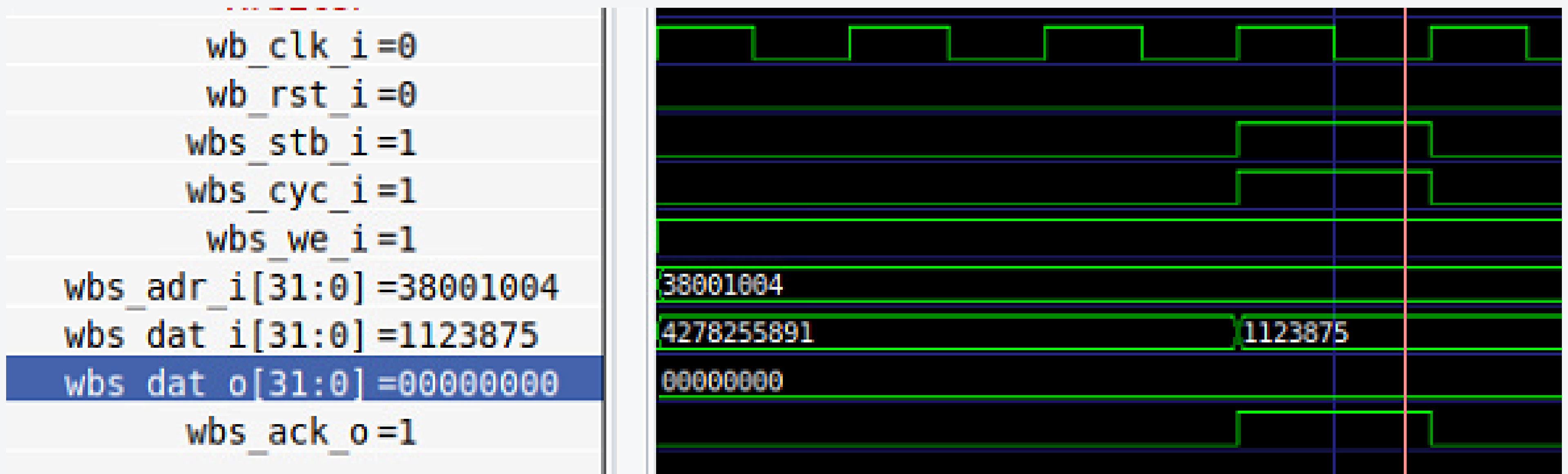
Data Flow

CPU Read



```
wb_clk_i=1
wb_rst_i=0
wbs_stb_i=1
wbs_cyc_i=1
wbs_we_i=0
wbs_adr_i[31:0]=38001000
wbs_dat_i[31:0]=XXX
wbs_dat_o[31:0]=00000000
wbs_ack_o=0
```

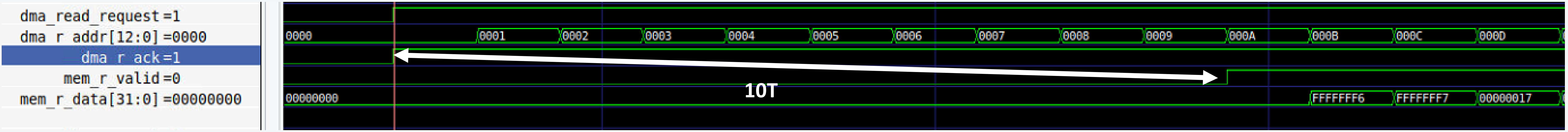
CPU Write



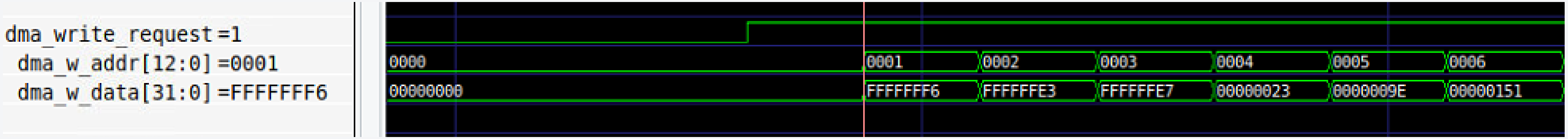
```
wb_clk_i=0
wb_rst_i=0
wbs_stb_i=1
wbs_cyc_i=1
wbs_we_i=1
wbs_adr_i[31:0]=38001004
wbs_dat_i[31:0]=1123875
wbs_dat_o[31:0]=00000000
wbs_ack_o=1
```

Data Flow

DMA Read



DMA Write



DMA (Direct Memory Access)

Process - Normal Way

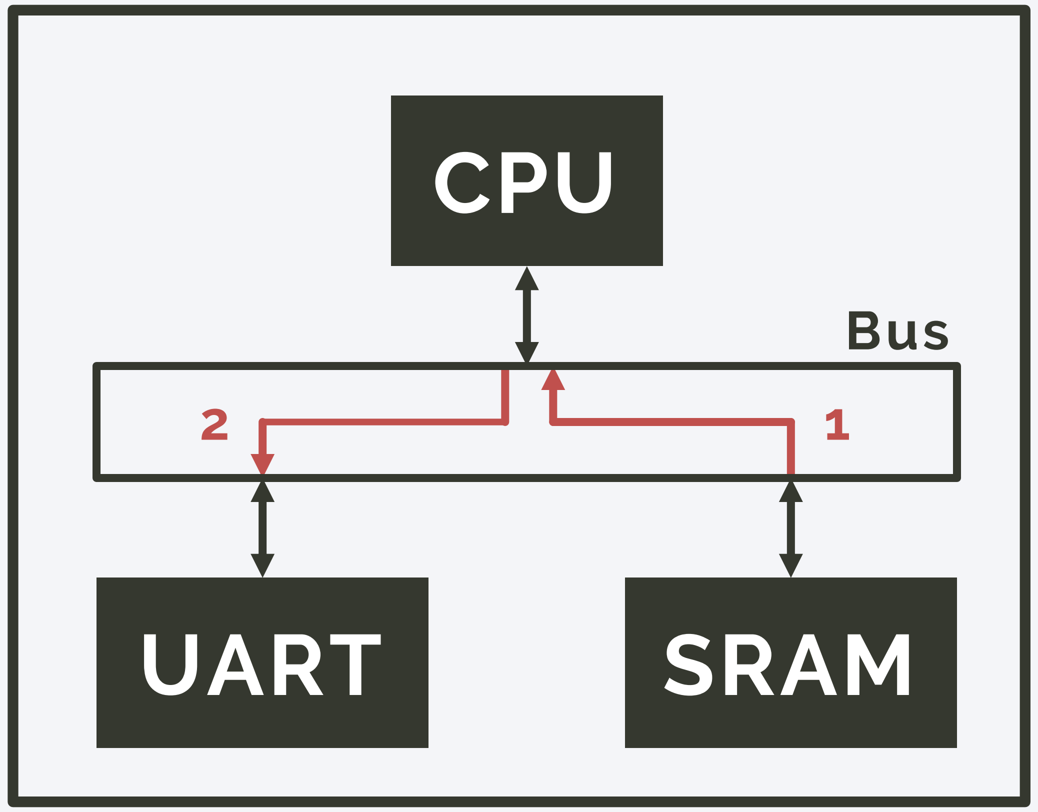
1. CPU Read data from RAM
2. CPU Write data to UART

What if multiple data?

Firmware

```c
uint8_t data = 0xAB;
REG_TX_DATA = data;
```

System



DMA (Direct Memory Access)

Process

1. CPU Write DMA config
2. DMA move data

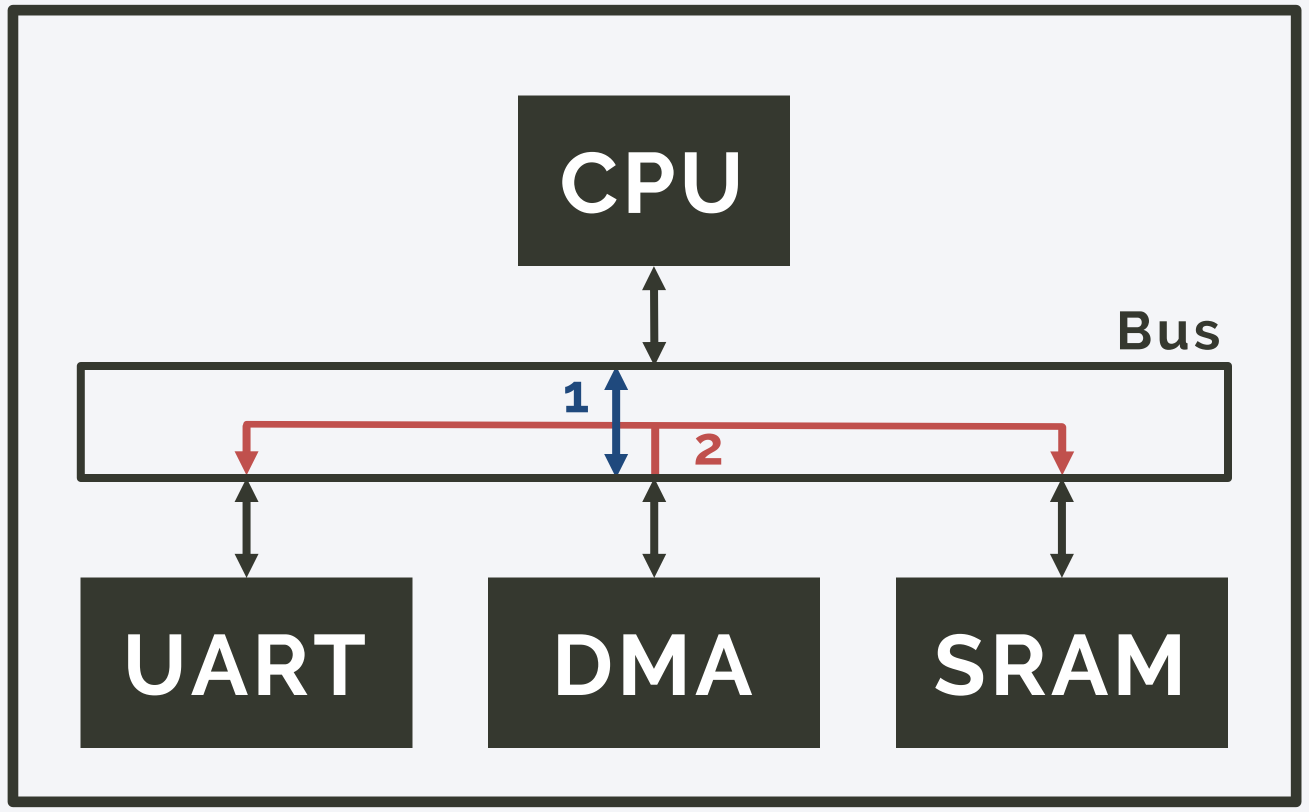
Config

1. Memory Address
2. Peripheral Address
3. Data Length
4. MEM/Peri. Direction
5. DMA status

Firmware

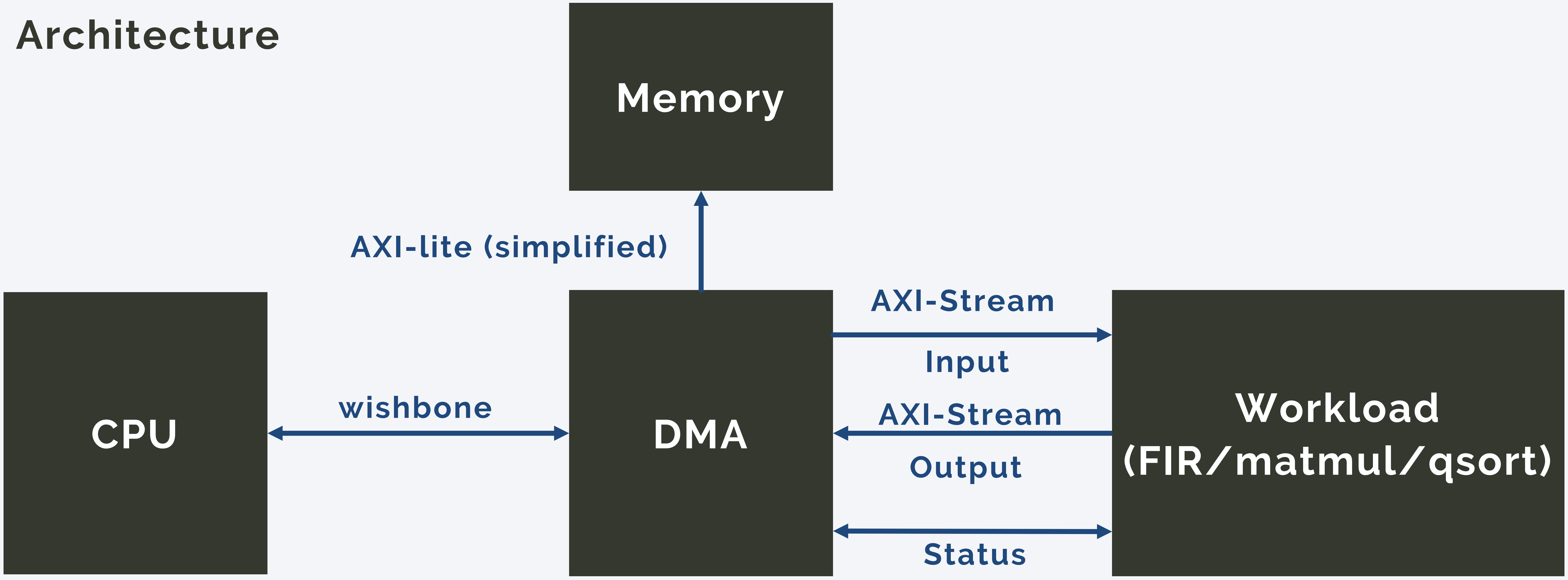
```
REG_DMA = DMA_cfg;
```

System



DMA (Direct Memory Access)

Architecture

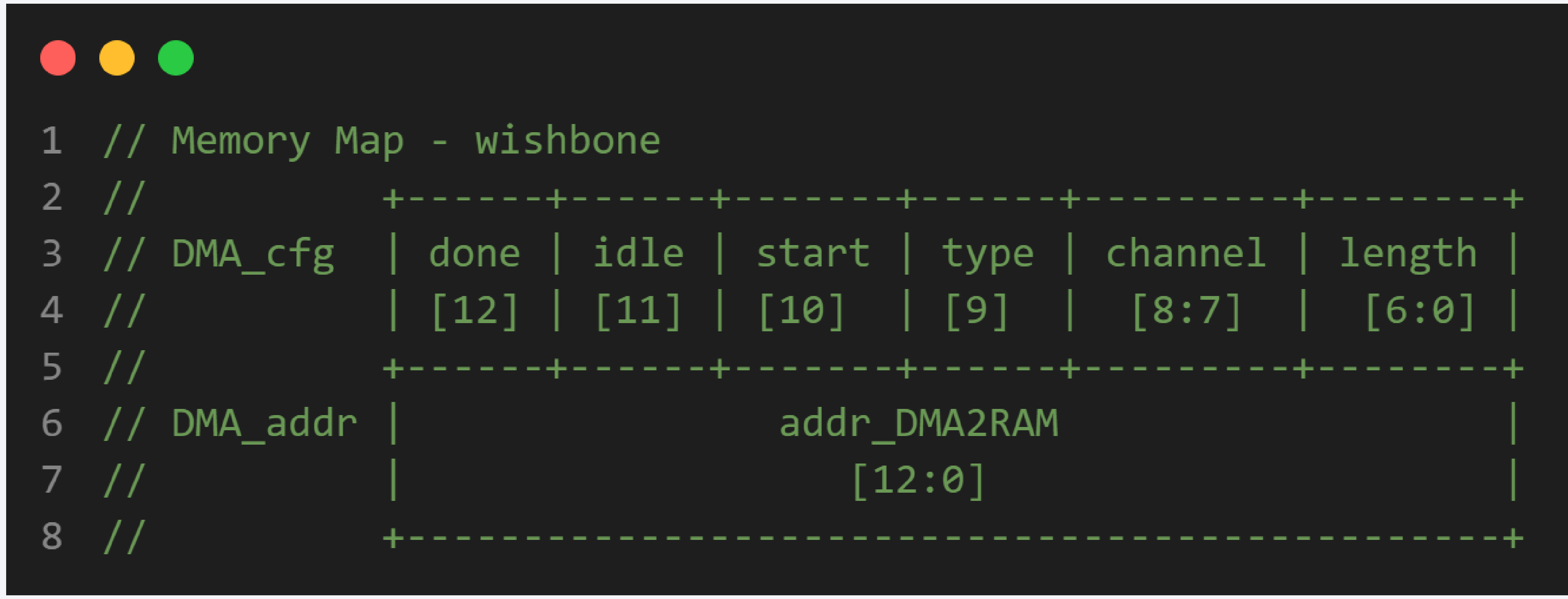


DMA (Direct Memory Access)

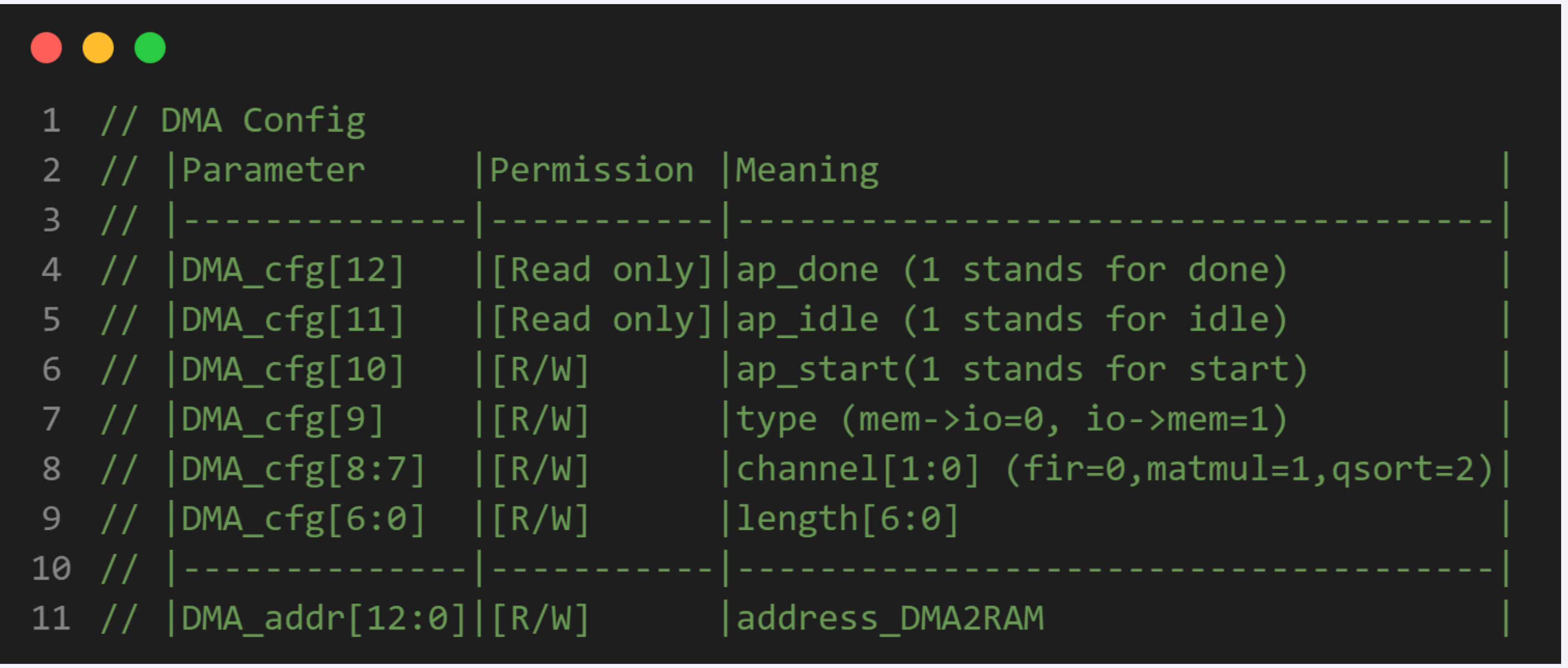
Config

MEM->Peri.

```
reg_DMA_addr=(qsort_input_base<<DMA_addr_base);    // Memory Address
reg_DMA_cfg =(1 << DMA_cfg_start) |                // DMA start
             (DMA_type_MEM2IO << DMA_cfg_type) |   // Direction
             (DMA_ch_qsort << DMA_cfg_channel) |   // Peri. Address
             (NUM_QSORT_INPUT<<DMA_cfg_length);    // Data Length
```



```
// Memory Map - wishbone
//           +------+------+-------+------+---------+---------+
// DMA_cfg   | done | idle | start | type | channel | length  |
//           | [12] | [11] | [10]  | [9]  |  [8:7]  |  [6:0]  |
//           +------+------+-------+------+---------+---------+
// DMA_addr  |                 addr_DMA2RAM                   |
//           |                    [12:0]                      |
//           +------------------------------------------------+
```



```
// DMA Config
// |Parameter     |Permission |Meaning                               |
// |--------------|-----------|--------------------------------------|
// |DMA_cfg[12]   |[Read only]|ap_done (1 stands for done)           |
// |DMA_cfg[11]   |[Read only]|ap_idle (1 stands for idle)           |
// |DMA_cfg[10]   |[R/W]      |ap_start(1 stands for start)          |
// |DMA_cfg[9]    |[R/W]      |type (mem->io=0, io->mem=1)           |
// |DMA_cfg[8:7]  |[R/W]      |channel[1:0] (fir=0,matmul=1,qsort=2)|
// |DMA_cfg[6:0]  |[R/W]      |length[6:0]                           |
// |--------------|-----------|--------------------------------------|
// |DMA_addr[12:0]|[R/W]      |address_DMA2RAM                       |
```

Insight

How can we improve further

- Reduce the number of CPU program DMAs by declaring variables with attributes and setting section.lds to fix the base of the data.DMA controller with concurrent read-write ability.

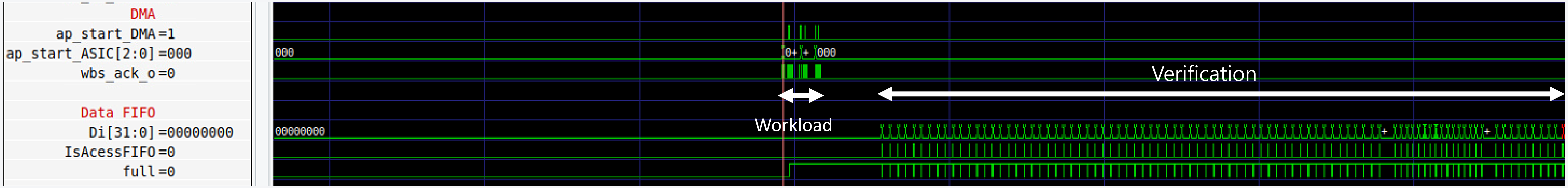
- Adding more FIFO channels to the gas pedal can further enhance parallel computing efficiency.

- Adding Instruction Cache & preload Mechanism.

Insight

How can we improve further

- Reschedule the workflow.



DMA

ap_start_DMA=1

ap_start_ASIC[2:0]=000

wbs_ack_o=0

Data FIFO

Di[31:0]=00000000

IsAcessFIFO=0

full=0

Workload

Verification

The End

Github Link

- https://github.com/pocper/112_SOC_final_project/tree/main