Grundlagen der Prozessorarchitektur

- **Logistics**
- **Introduction to MIPSfpga**
  - – Background
  - – Core and System
  - – Interfaces
    - System Interface
    - AHB-Lite Bus
    - EJTAG

Logistics

- Nexys4-DDR Board pick-up
  - Bring 50 Euro deposit (cash) - refundable after board return
  - Frau Reimund, Do. und Fr., 28.04. und 29.04.
  - Piloty E103, Do. 12:30-15:30, Fr. 09:30-12:30
- Lab 3: Start **early**!
  - For example, you should have a completely written pipelined processor by next week: ~04.05.
  - Allow at least a week for debugging
  - Average time to complete: ~15 hours (or more)
- **No lecture next week** (05.05.) – Feiertag

History of MIPS Architecture

- Developed by **John Hennessy** and his colleagues at Stanford in the 1980's
- One of the first commercial **Reduced Instruction Set Computer** (RISC) Architecture
- Hennessy co-founded MIPS Computer Systems – later called MIPS Technologies
- Used in many commercial systems, including Silicon Graphics workstations, Nintendo machines, and Cisco servers
- Studied by a majority of universities
- Over 5 billion MIPS microprocessors sold

History of MIPS Architecture

- Imagination Technologies purchased MIPS Technologies in February 2013
  - British-based company
  - Other products include: PowerVR mobile graphics processor, consumer electronics and audio equipment

MIPS Cores

- **MIPS R3000, R4000, R10000**
  - 1980’s and 1990’s
  - For example, found in Silicon Graphics workstations
- **Embedded: M4K, M14K**
  - Examples:
    - Microchip’s popular **PIC32** line of microcontrollers is based on M4K core
    - Samsung’s **Artik-1** IoT-focused chip is a MIPS core

MIPS Cores

- **microAptiv**
  – highly-efficient, compact, embedded core
  – based on M14K architecture
- **interAptiv, proAptiv**
  – higher-performance
  – multi-processor, superscalar, multi-threading
- **Warrior**
  – Newest line of Imagination MIPS cores
  – Range of high-performance to embedded cores

MIPS Cores

- **microAptiv: MIPSfpga is a microAptiv core**
  - highly-efficient, compact, embedded core
  - based on M14K architecture
- **interAptiv, proAptiv**
  - higher-performance
  - multi-processor, superscalar, multi-threading
- **Warrior**
  - Newest line of Imagination MIPS cores
  - Range of high-performance to embedded cores

MIPSfpga Background

What is MIPSfpga?

- A soft-core commercial MIPS processor implemented on an FPGA
- Made available to Universities by Imagination Technologies

MIPSfpga Terms

- Available for **Academic Use** only
- **Do not distribute**

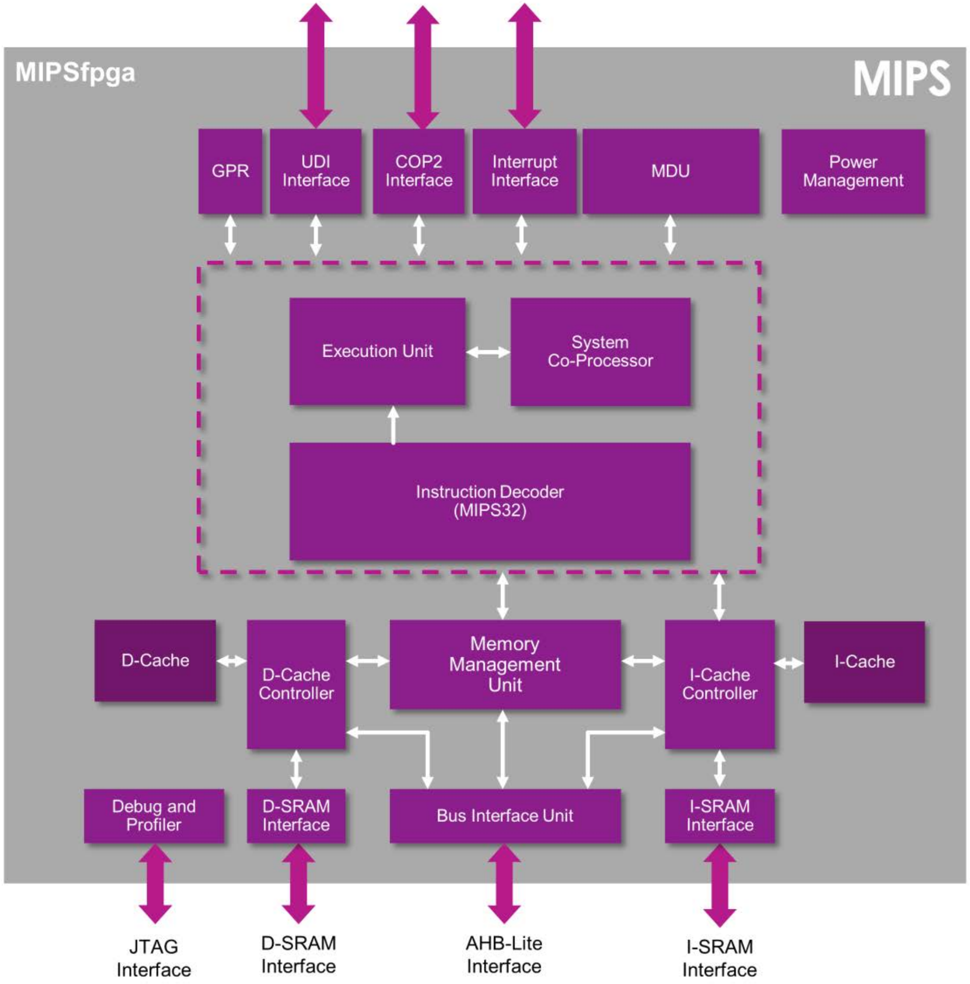
MIPSfpga: microAptiv Core

Commercial microAptiv core

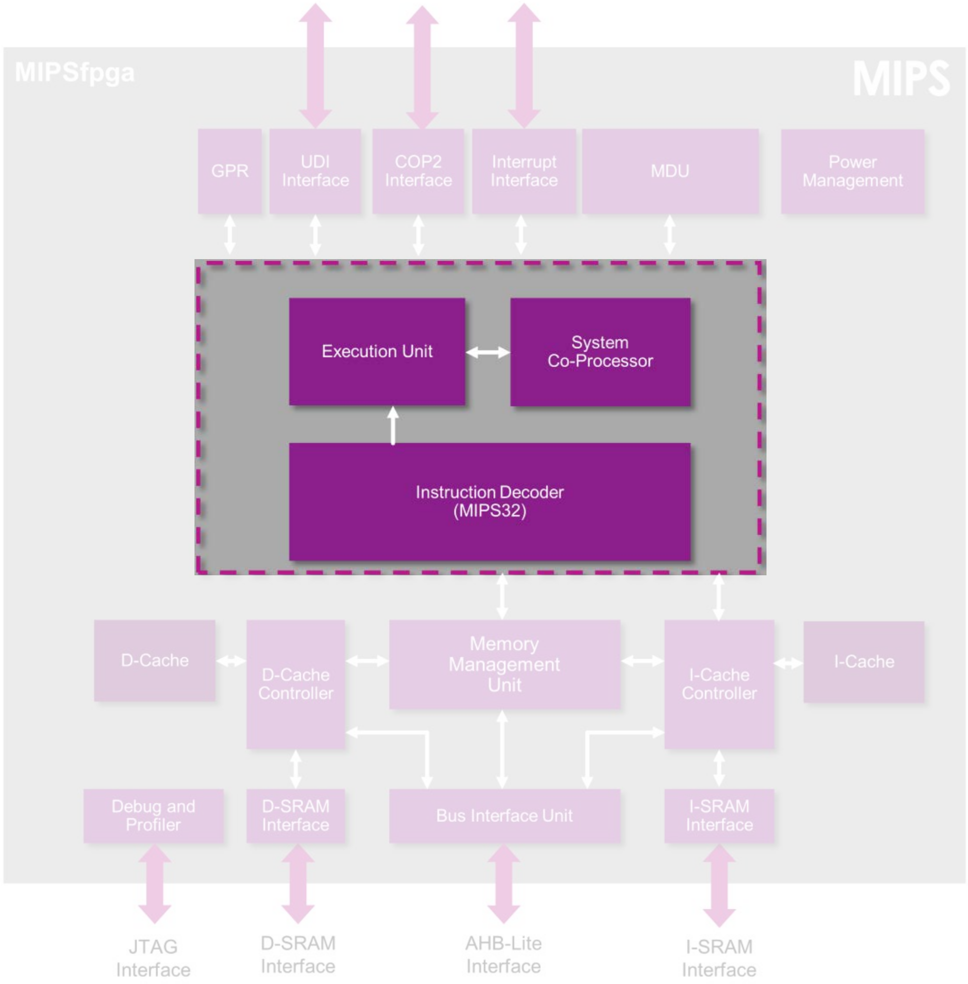
- 5-stage pipeline
- 1.5 Dhrystone MIPS/MHz
- 2-way associative I & D caches, 2 KB each
- MMU (memory management unit) with 16-entry TLB

MIPS

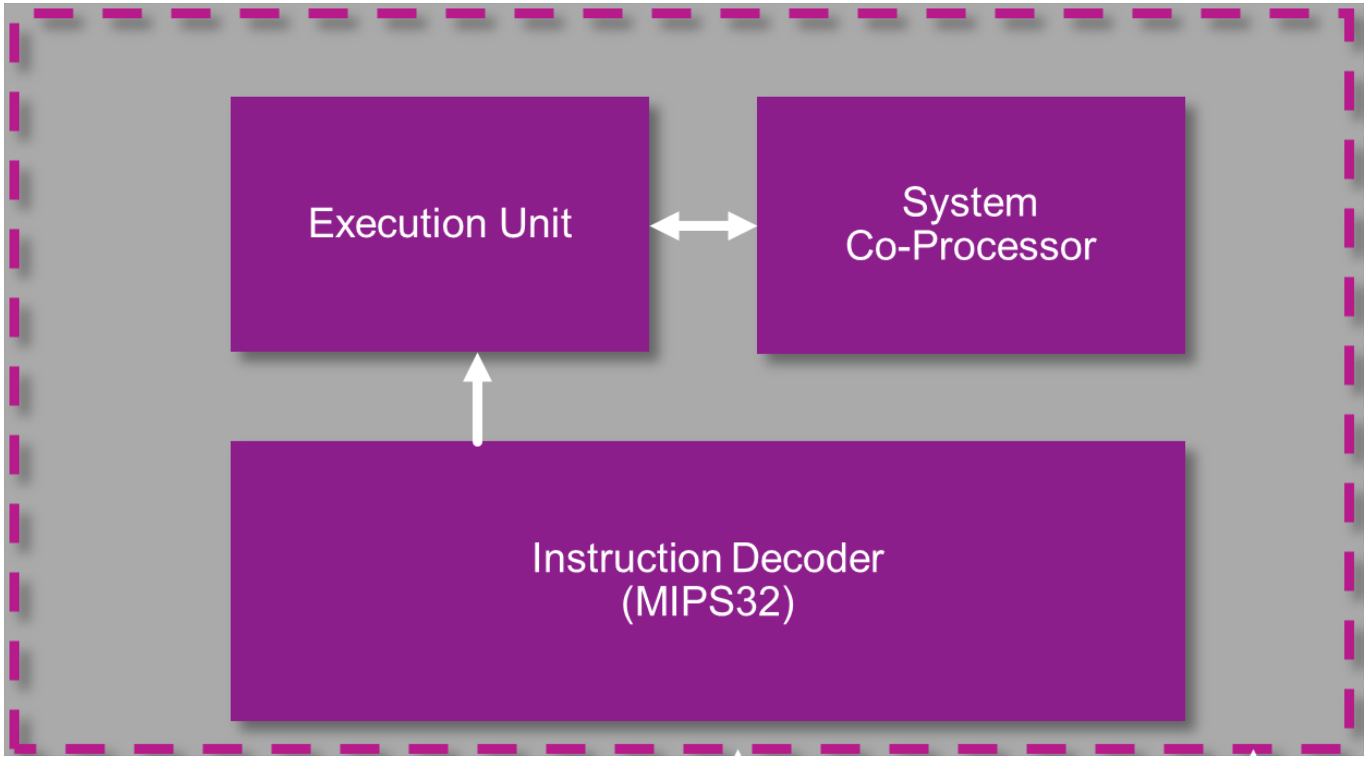
MIPSfpga Core



MIPSfpga Core

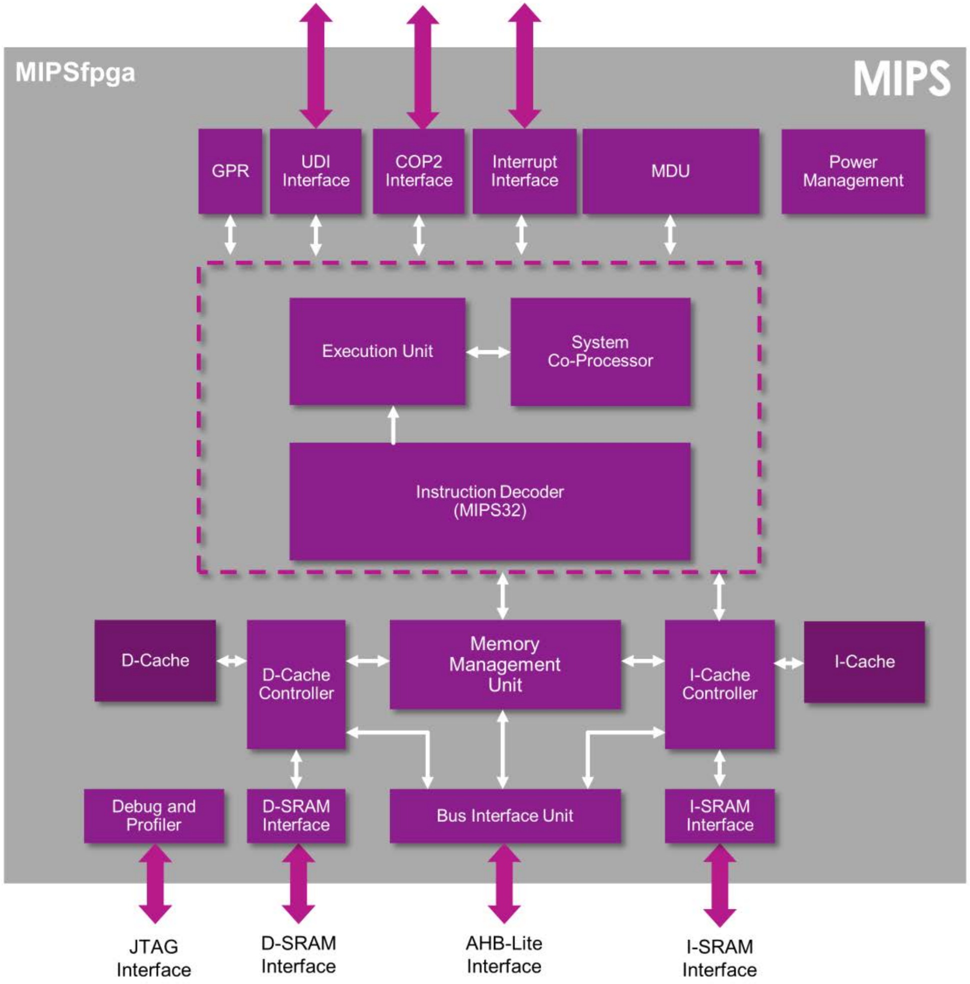


MIPSfpga Core



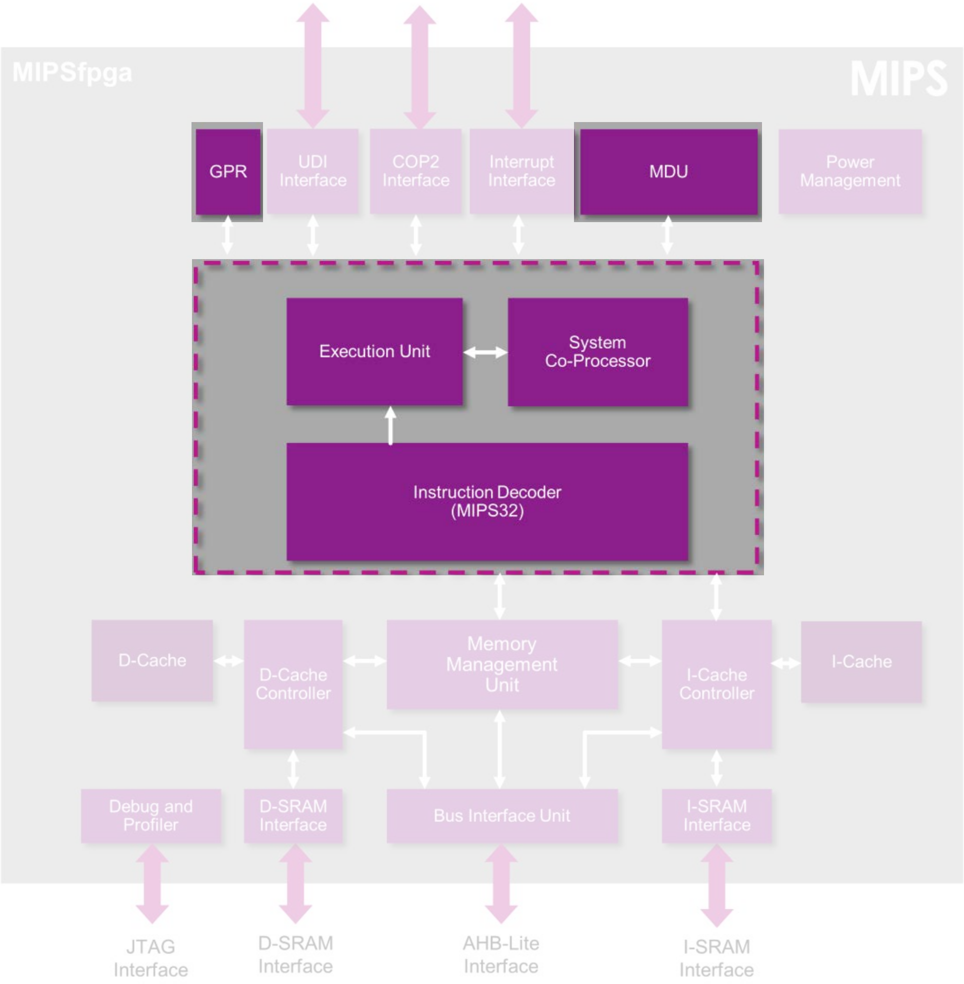
- **Processes instructions**
- **Co-processor: system registers, handles reset**

MIPSfpga: Registers, MDU



MIPSfpga: Registers, MDU

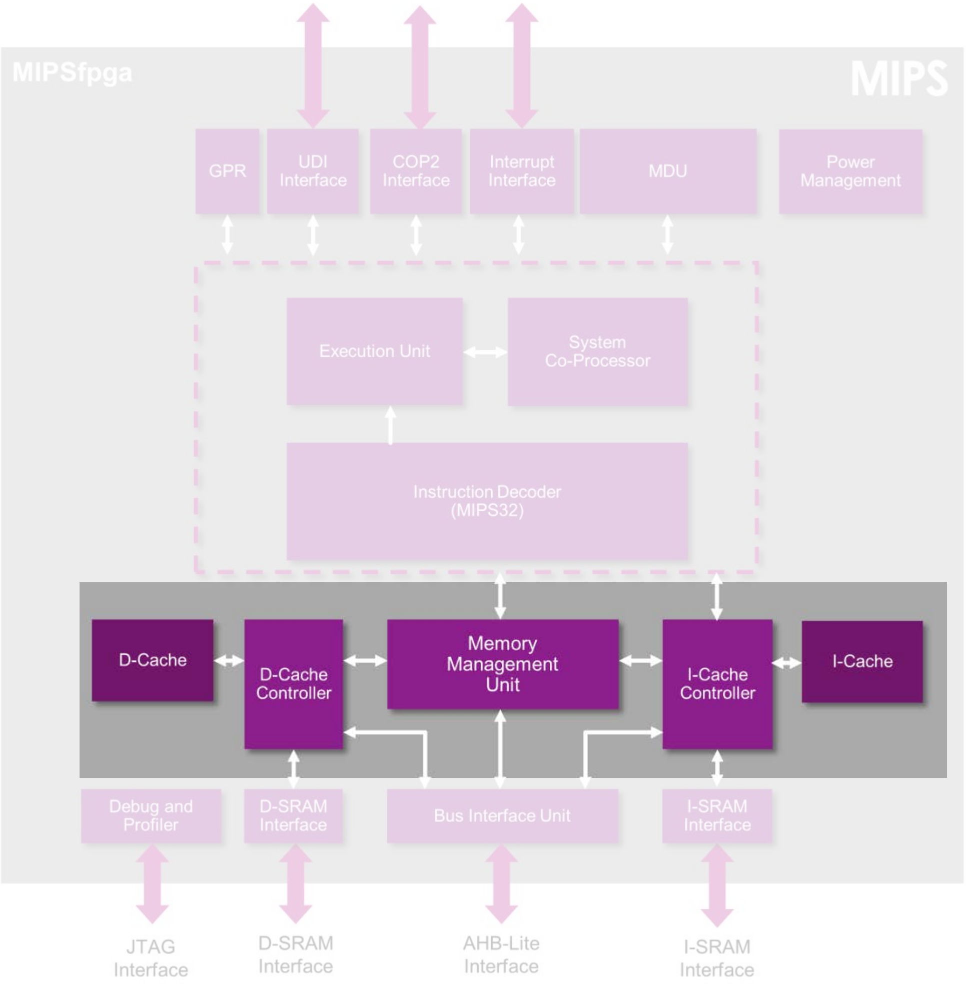
GPR:
General Purpose Registers



MDU:
Multiply/ Divide Unit

MIPSfpga: MMU, Caches

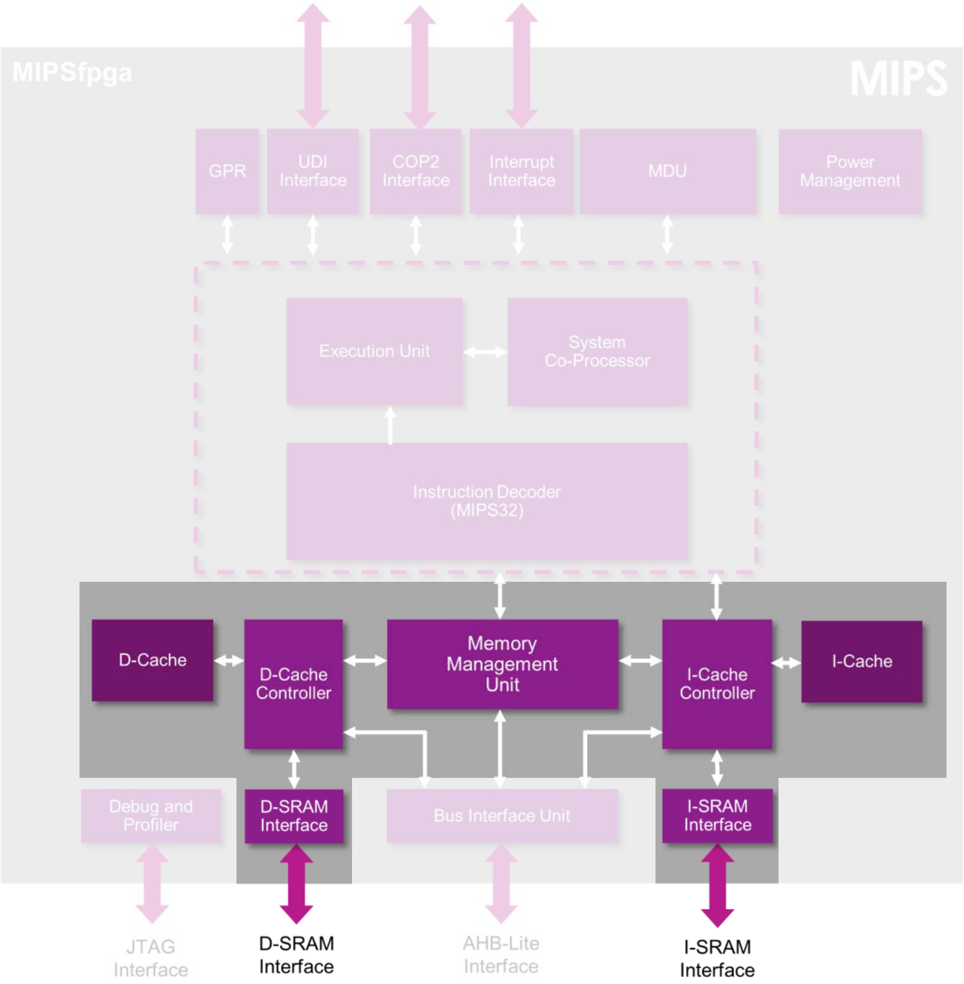
MMU:
Memory Management Unit



Caches:
Instruction & Data

Cache Controller:
Instruction & Data Caches

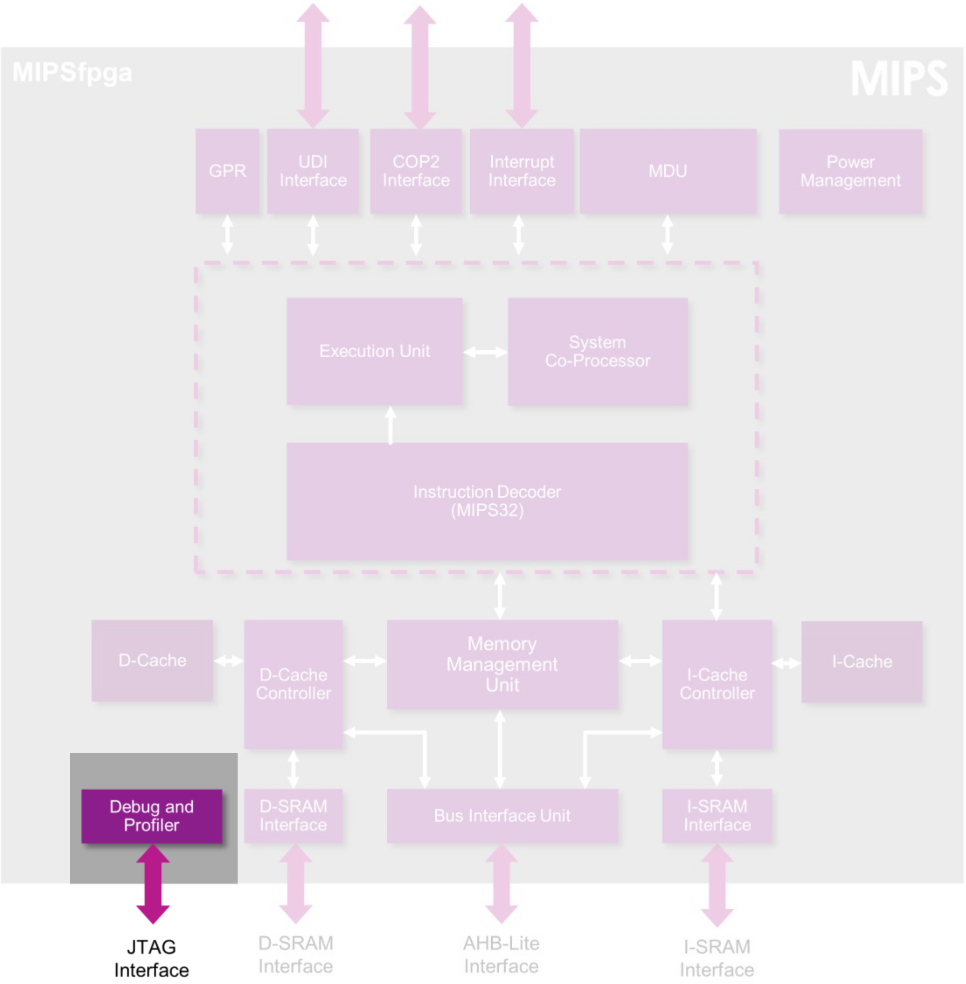
MIPSfpga: Cache Controller



Cache Controller: Interfaces with I & D Caches and external memory (called scratch RAM, SRAM)

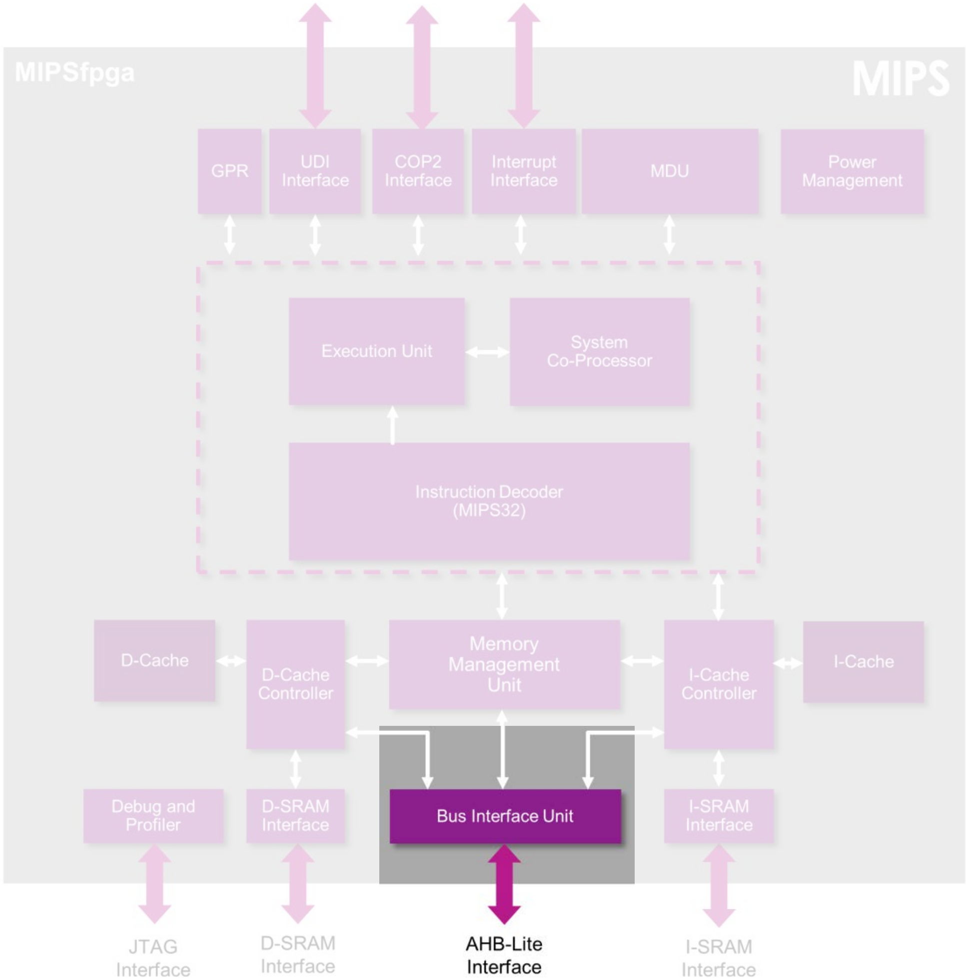
MIPSfpga (E)JTAG Interface

JTAG (also called **EJTAG**): Used for programming and real-time debugging of the core



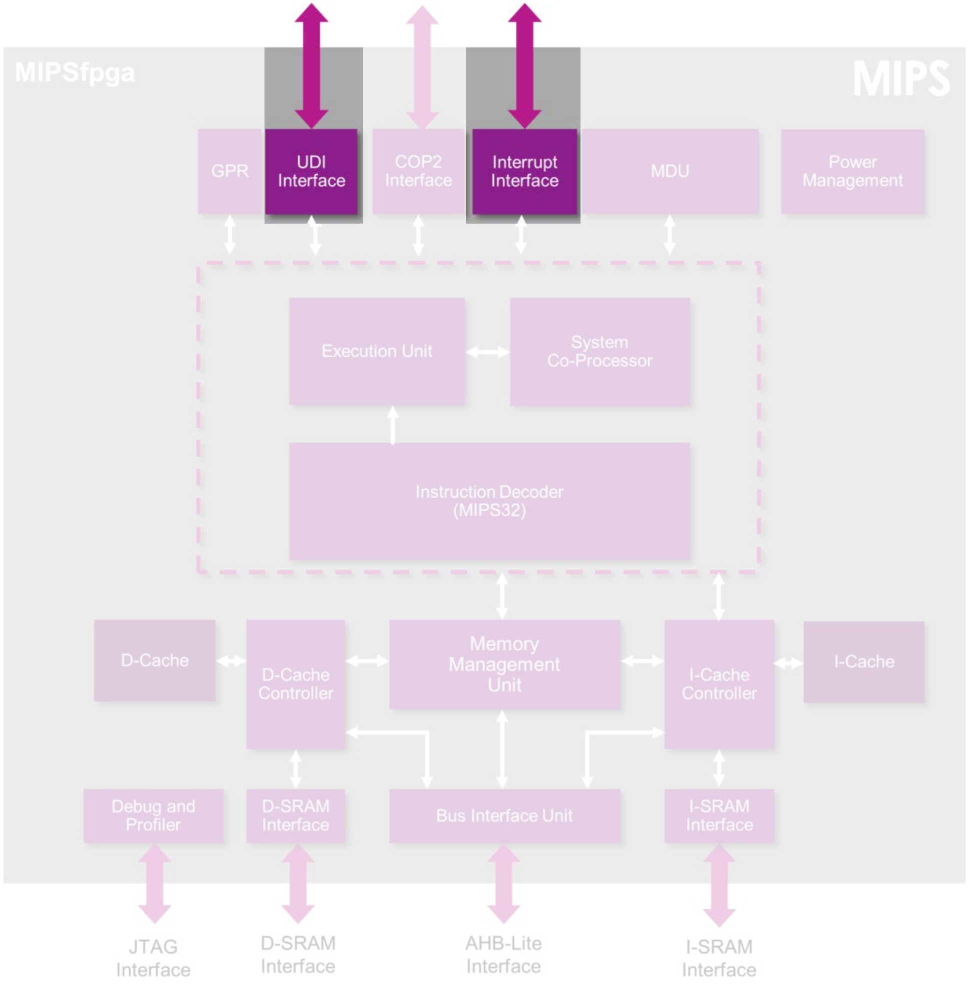
MIPSfpga AHB-Lite Bus

AHB-Lite Bus: Used for interacting with memory & peripherals



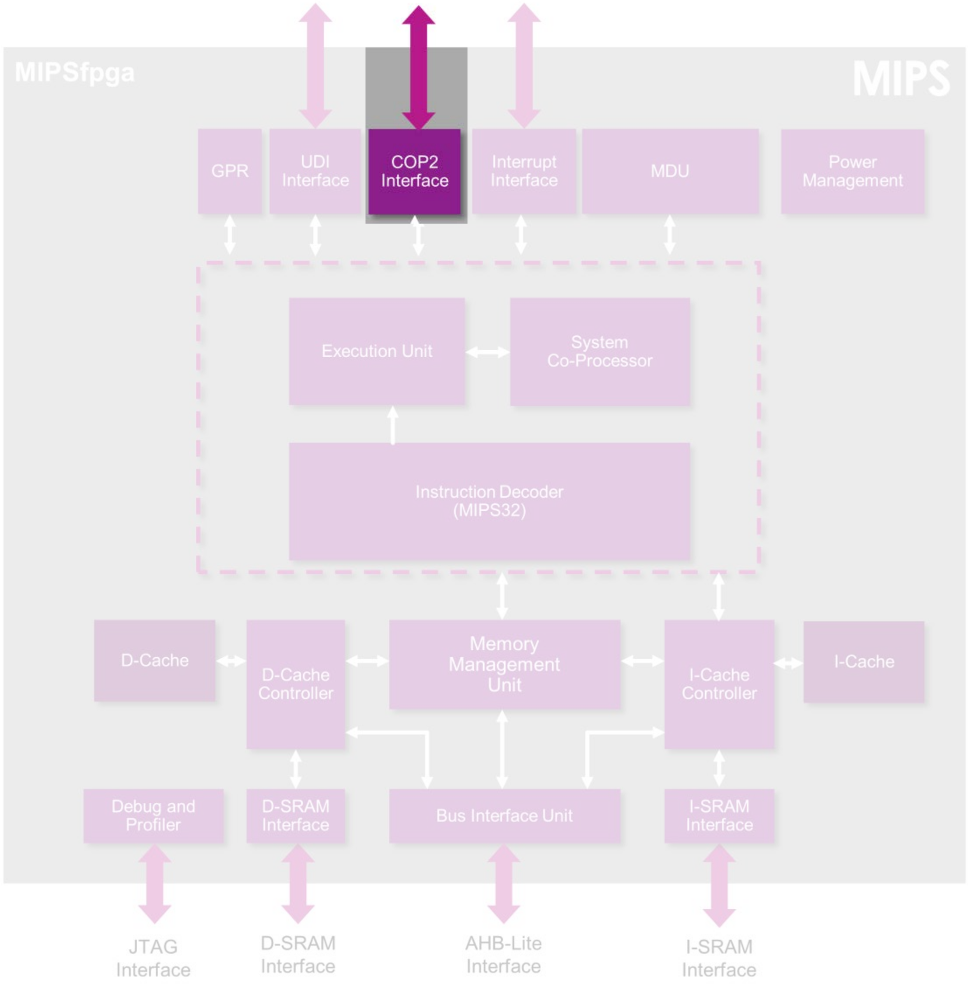
MIPSfpga UDI Interface, Interrupts

UDI (User-Defined Interface Unit): Enables user-defined instructions



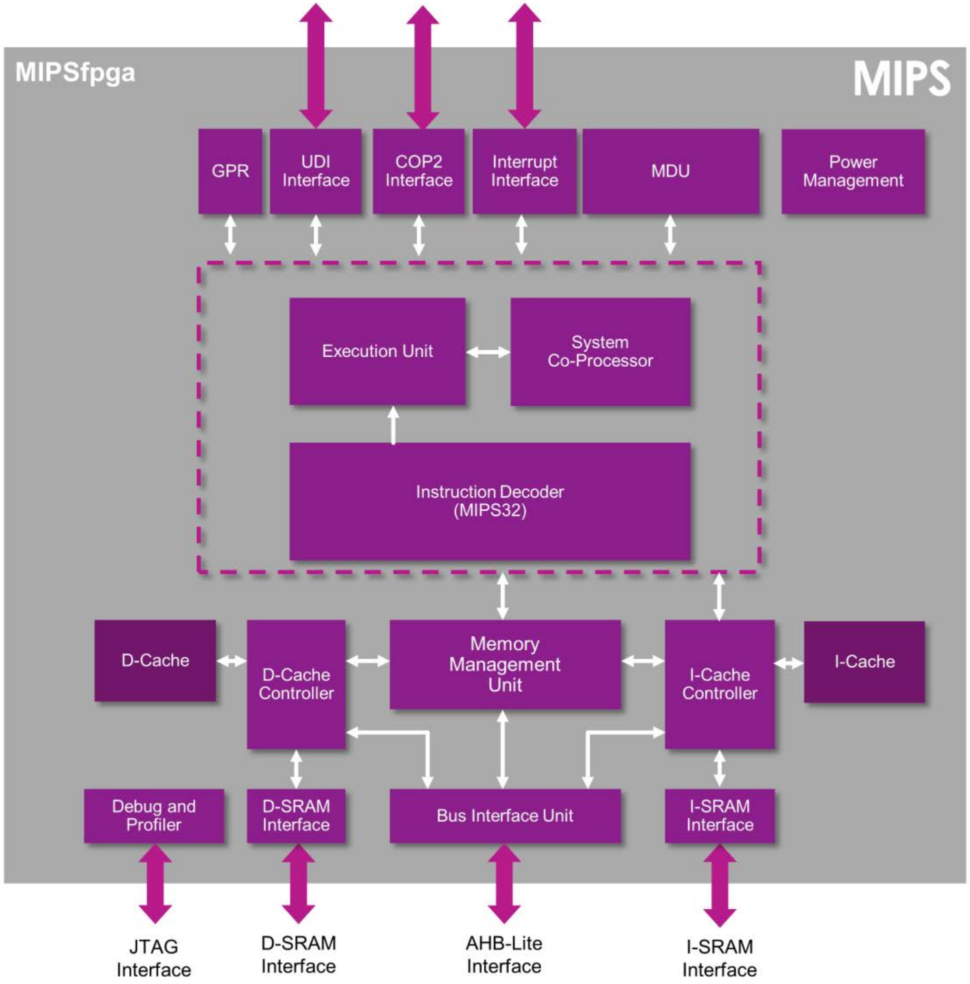
Interrupt Interface: used for hardware interrupts

MIPSfpga Coprocessor 2 Interface



COP2 Interface

MIPSfpga Core



MIPSfpga Core Re-cap

- **Commercial microAptiv core**
  - 5-stage pipeline
  - 4 KB 2-way set-associative I & D caches
  - MMU (memory management unit) with 16-entry TLB
  - Performance counters, input synchronizers
  - No DSP, Coprocessor 2, or shadow registers
  - Interfaces:
    - AHB-Lite bus
    - EJTAG programmer/debugger
    - CorExtend for user-defined instructions

MIPSfpga 5-Stage Pipeline

#	Stage	Name	Description
1	I	Instruction	Fetch Instruction
2	E	Execution	Fetch operands from RF & perform ALU operation
3	M	Memory	Access Memory
4	A	Align	Align data to word boundary
5	W	Writeback	Write result to RF

MIPSfpga Operating Modes

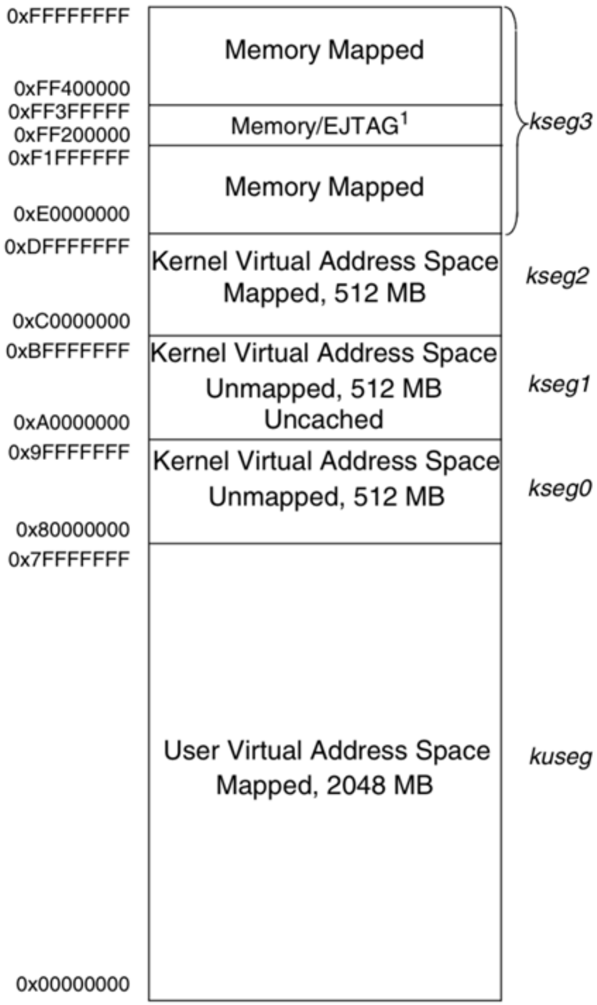
- Kernel
- User
- Debug

MIPSfpga Operating Modes

- Kernel
- User
- Debug

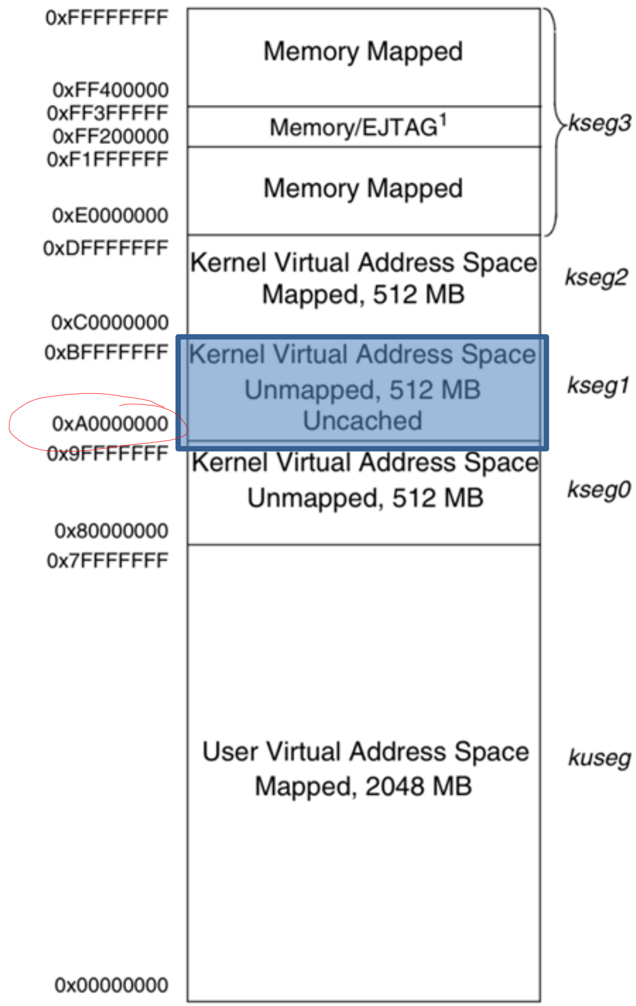
On reset, the processor begins in kernel mode and jumps to the reset vector at address 0xbfc00000.

MIPSfpga Memory Map



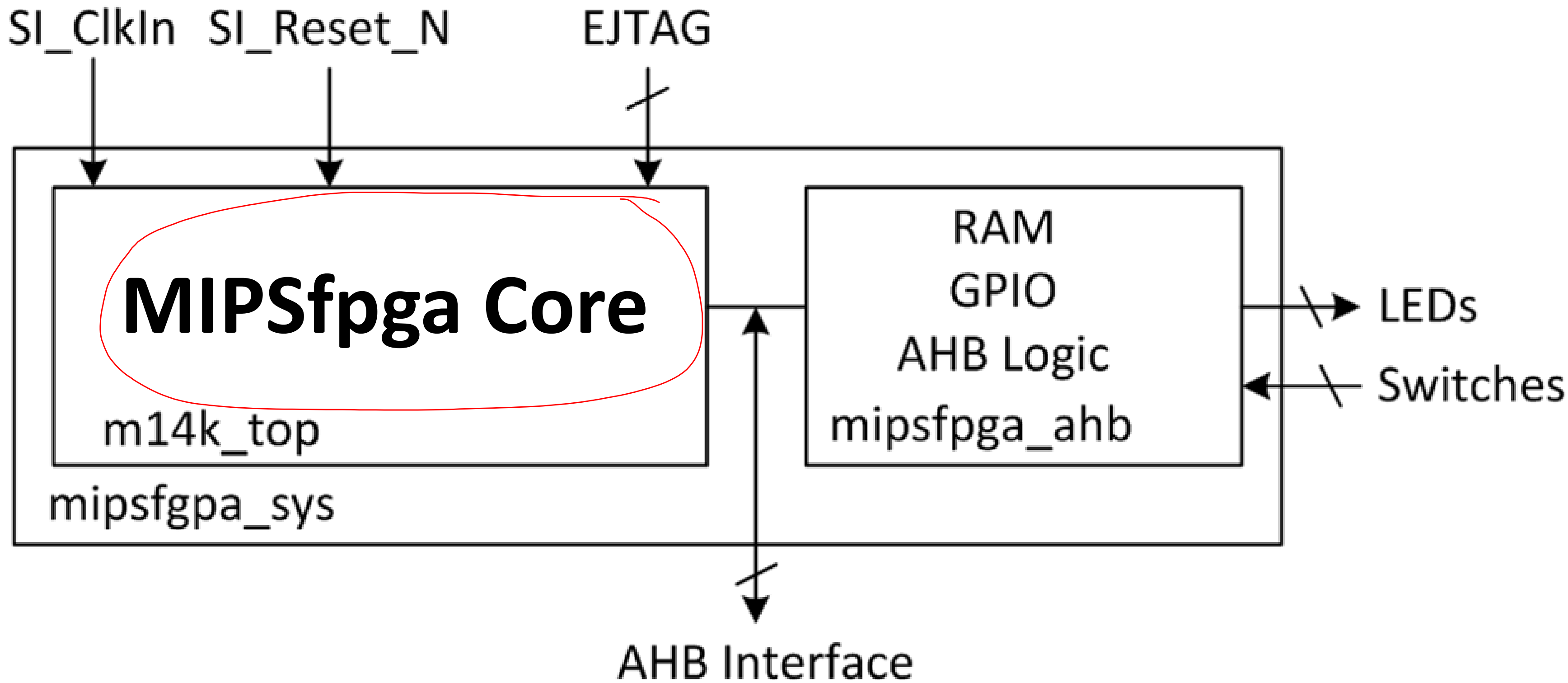
- **32-bit** virtual memory space (0x00000000 – 0xFFFFFFFF)
- Broken up into different segments
- **kseg0** and **kseg1** both map to physical addresses starting at 0x0, i.e.:
  - **0xA0000000** maps to physical address **0x00000000**
  - **0xBFC00000** => **0x1FC00000**
  - **0x80000000** => **0x00000000**

MIPSfpga Memory Map



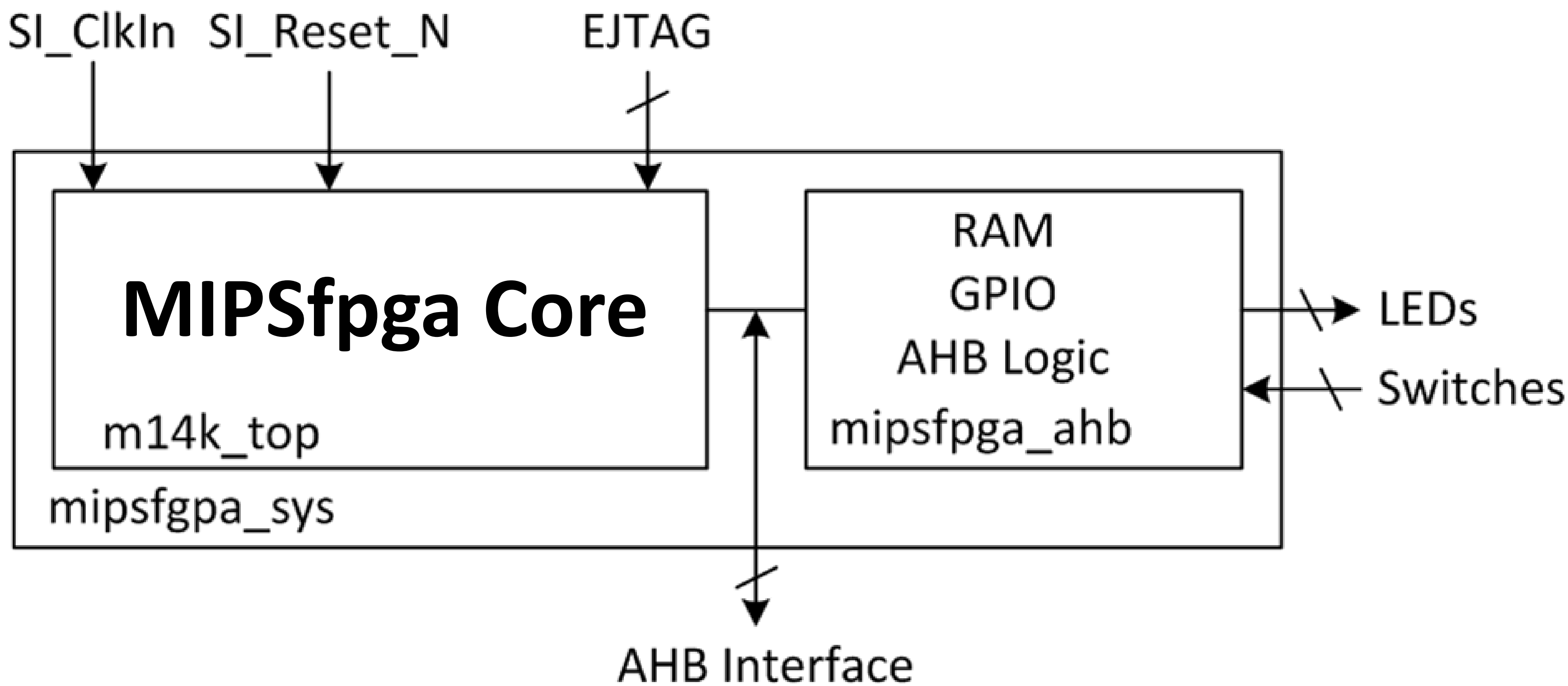
- On reset, processor begins in kernel mode and jumps to the reset vector at address **0xBFC00000**
- In **kseg1:** uncached and unmapped in TLB (nothing is initialized)
  - All instructions **fetched from external memory** (not cache)
  - **0xBFC00000** maps to physical address **0x1FC00000**

MIPSfpga System

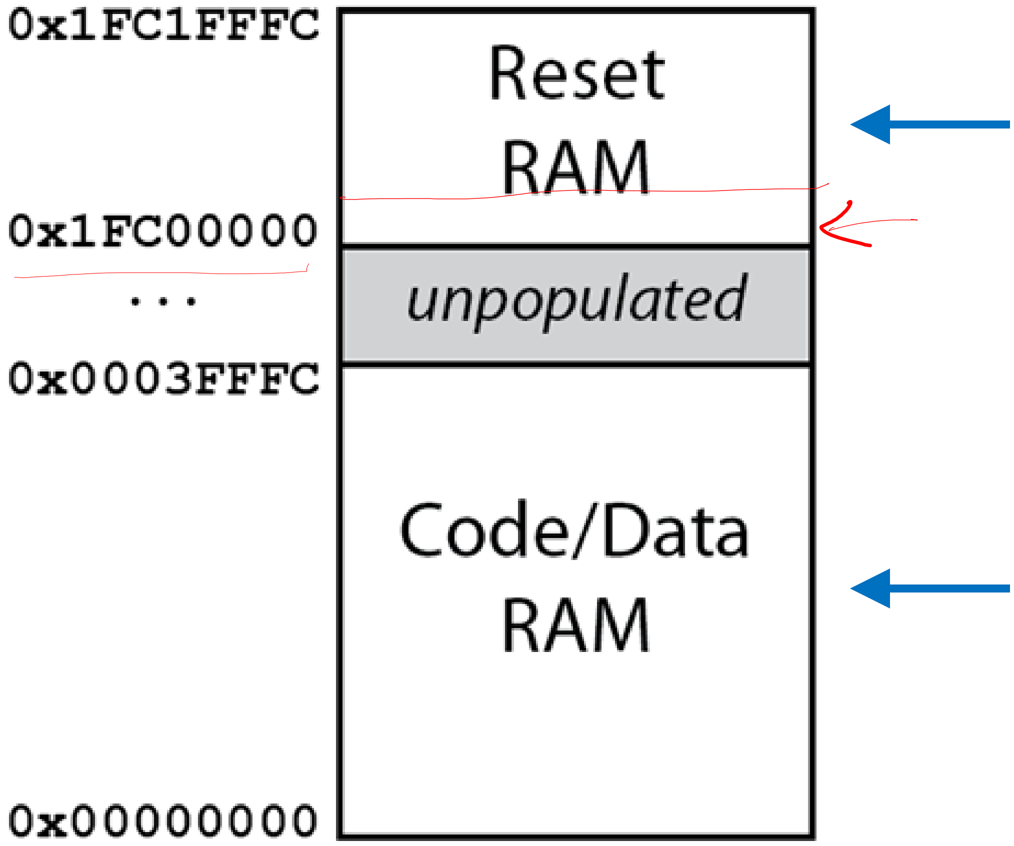


MIPSfpga System

RAM: 128 KB (boot code)
256 KB (user code)



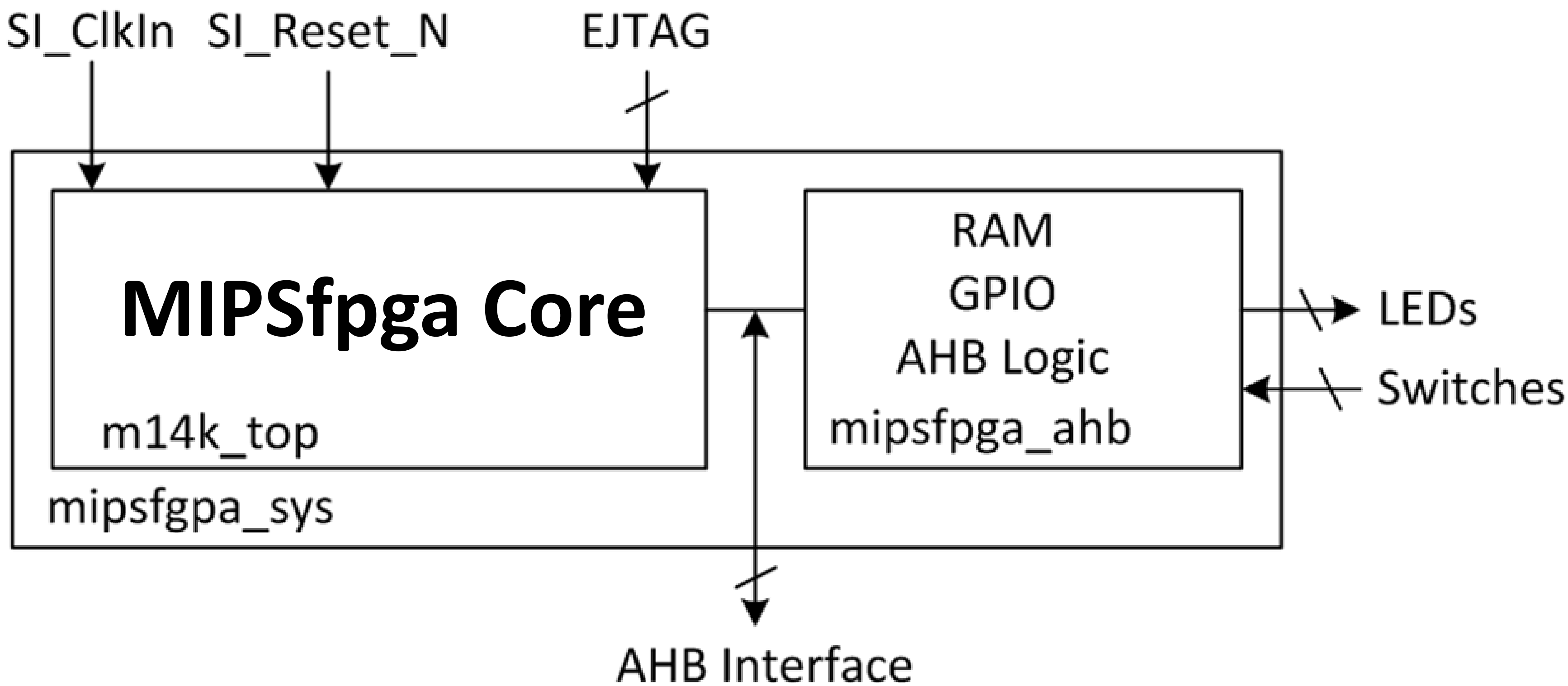
MIPSfpga System: Physical Memory



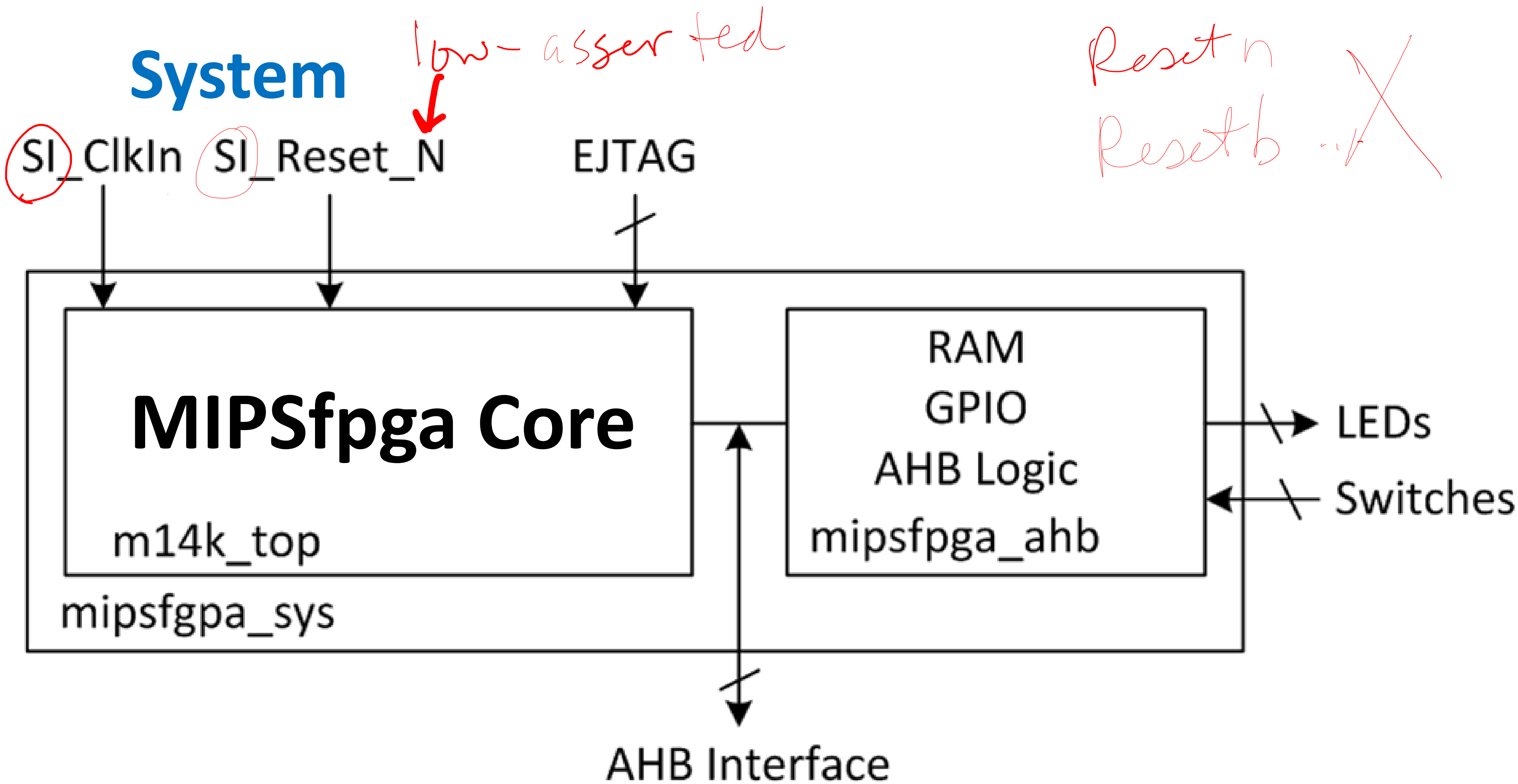
Boot Code:
Code executed at startup (128 KB)

User Code/Data
(256 KB)

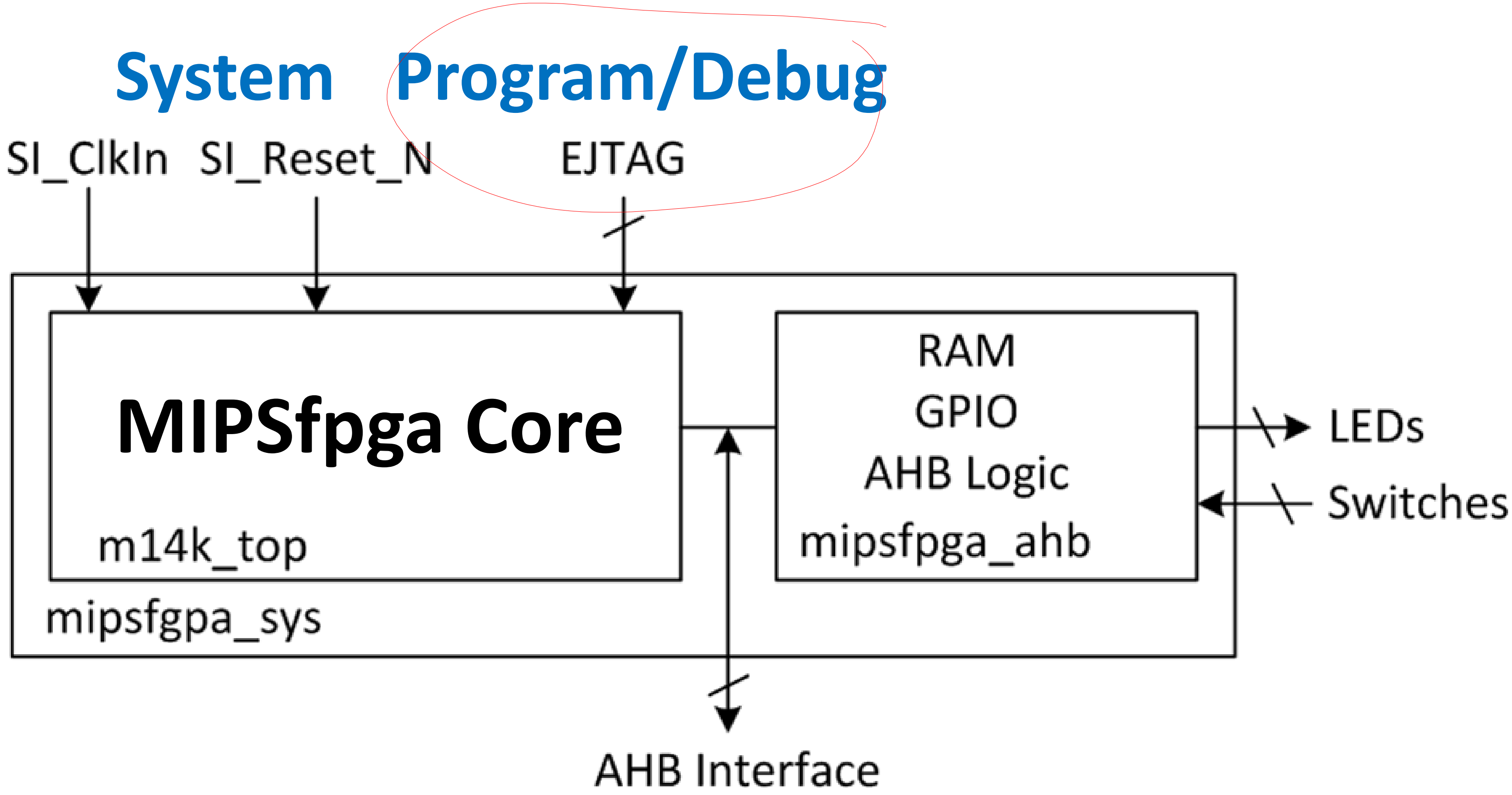
MIPSfpga System



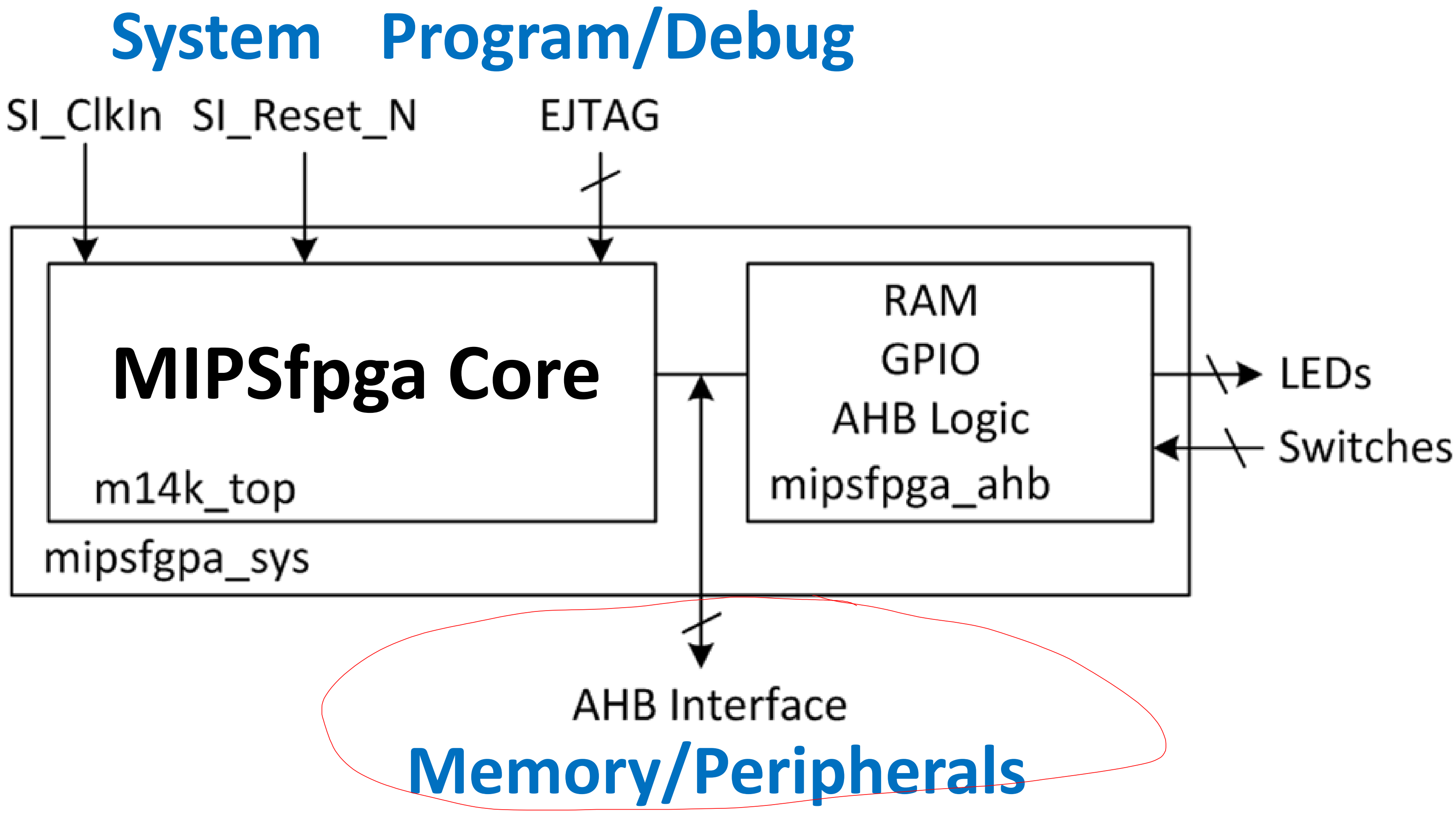
MIPSfpga System



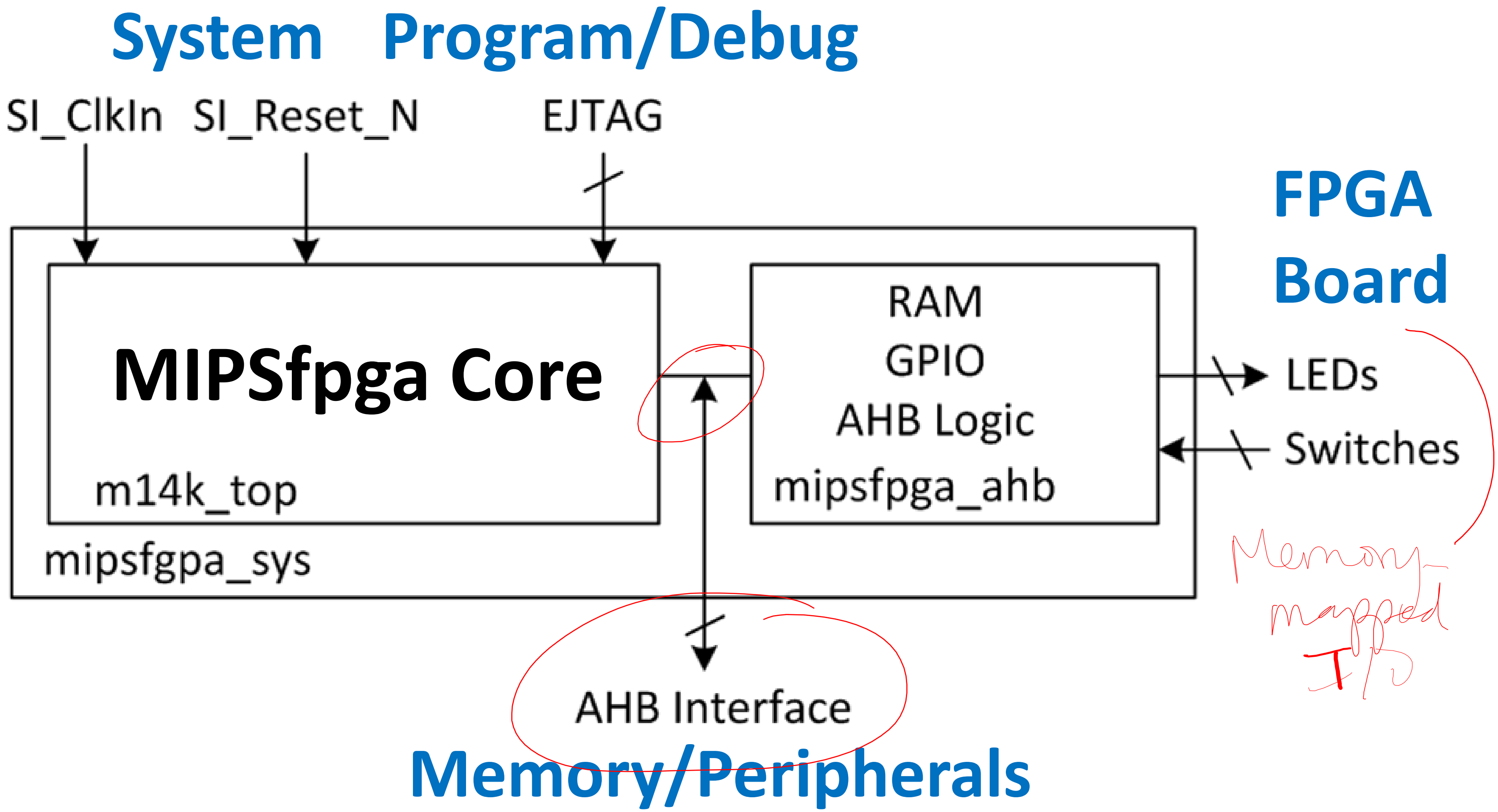
MIPSfpga System



MIPSfpga System



MIPSfpga System



MIPSfpga Interfaces: System

Signal Name	Description	Nexys4 DDR Board
SI_Reset_N	Resets the processor when 0	CPU Reset button
SI_ClkIn	System clock	50 MHz (derived from on-board 100 MHz clock)

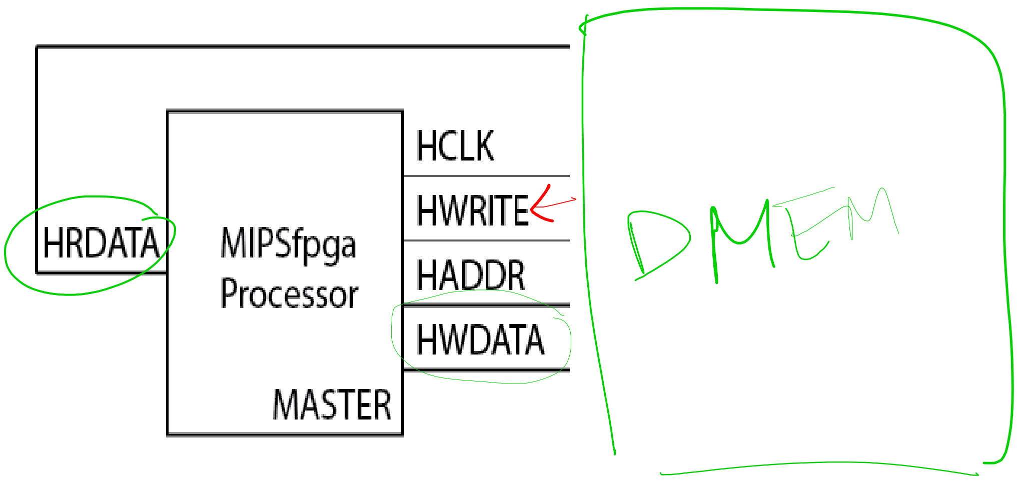
SI: prefix for system interface signals in Verilog files

MIPSfpga Interfaces: AHB-Lite

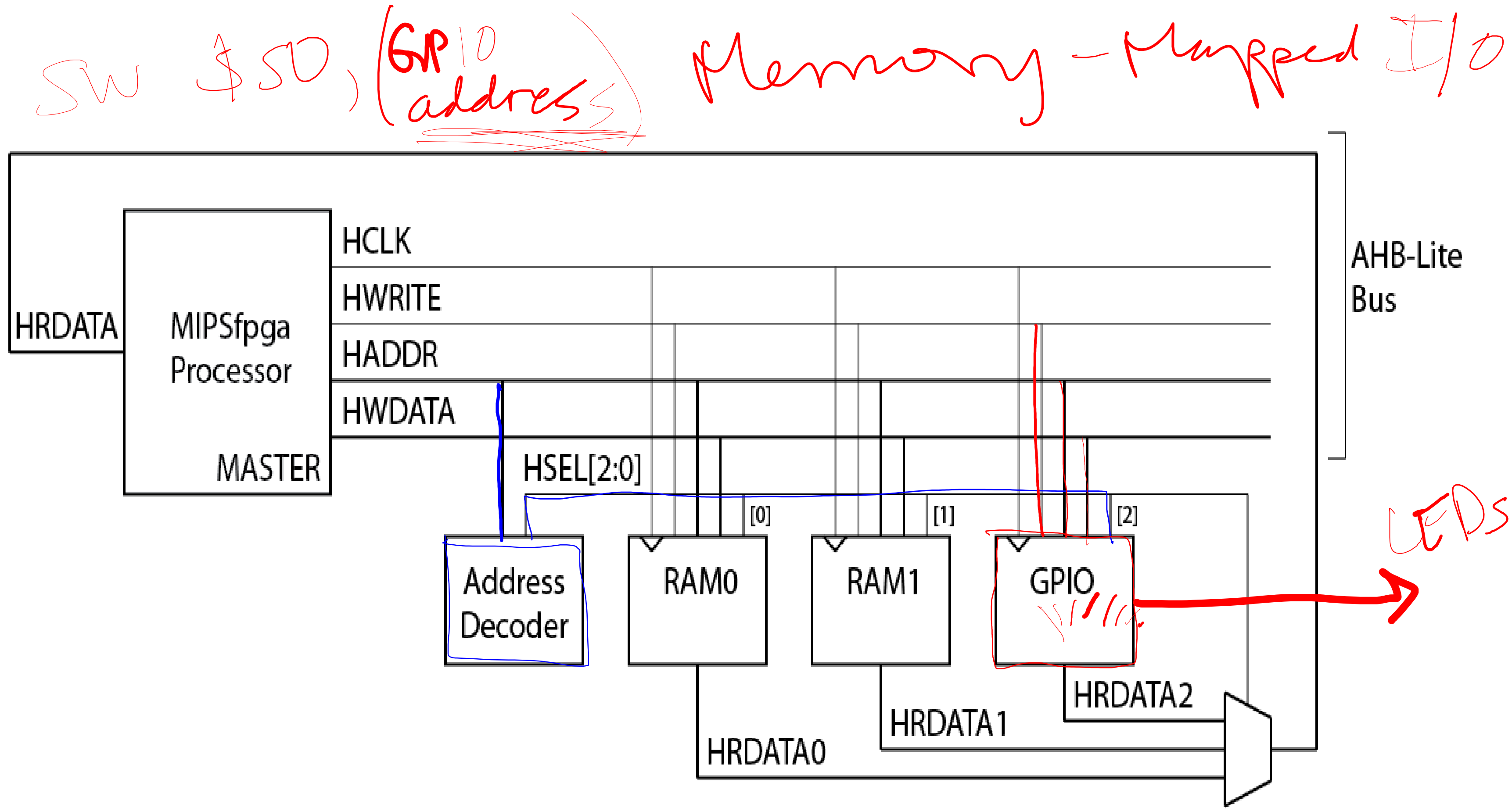
Signal Name	Description
HADDR[31:0]	Address bus
HRDATA[31:0]	Read data bus
HWDATA[31:0]	Write data bus
HWRITE	Write enable
HCLK	Clock – system clock

H: prefix for A**H**B-Lite Interface signals in Verilog files

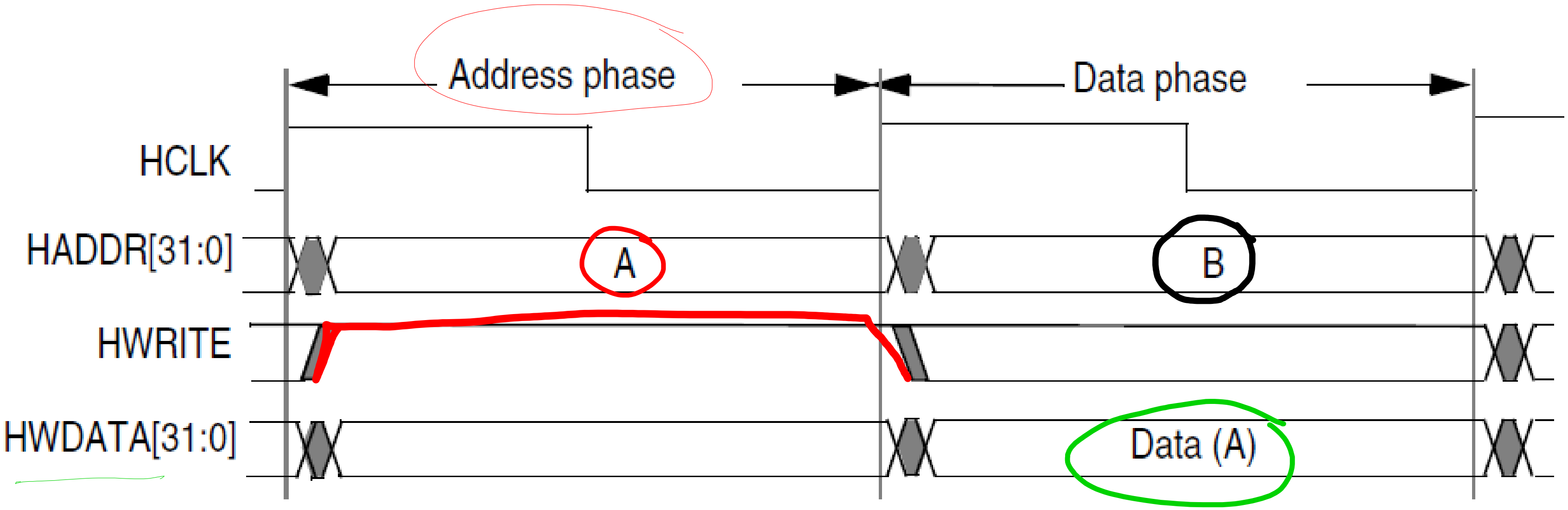
MIPSfpga Interfaces: AHB-Lite



AHB-Lite Memory / Peripherals



AHB-Lite Write Timing

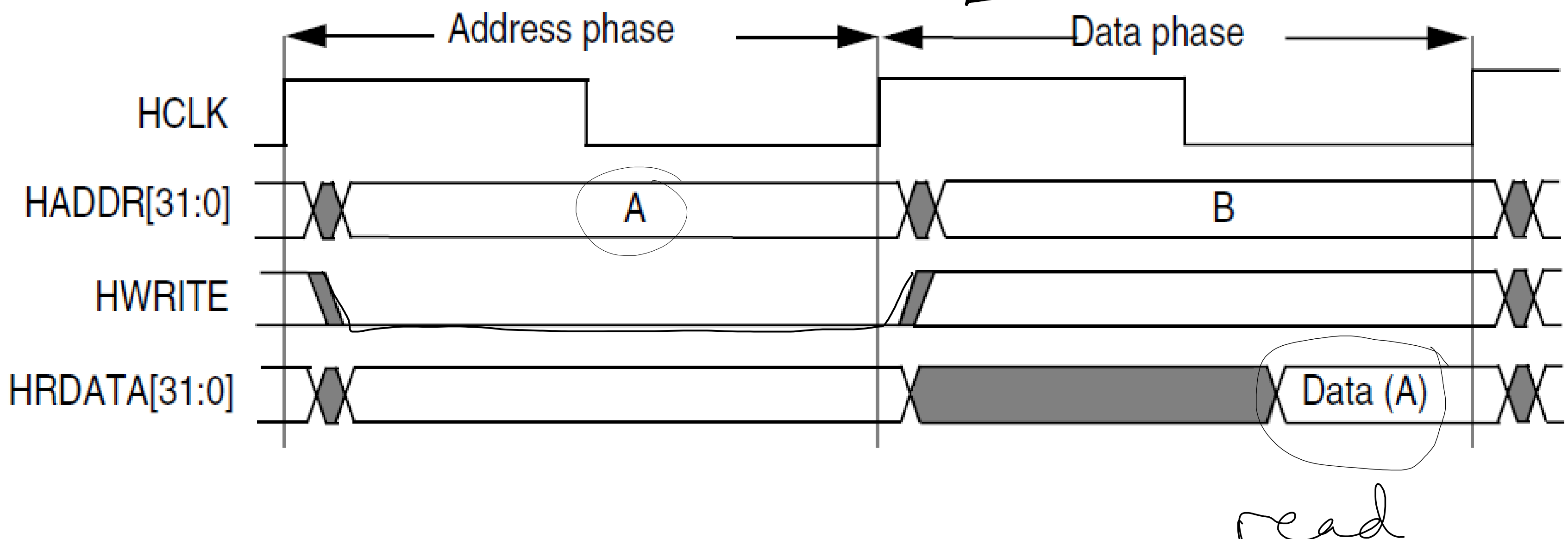


Cycle 1: Address and Write Enable (HADDR & HWRITE)

Cycle 2: Write Data (HWDATA)

AHB-Lite Read Timing

sw $t0, 4($s0)
add $t1, $t0, $t2



Cycle 1: **Address** (HADDR) (also, HWRITE = 0)

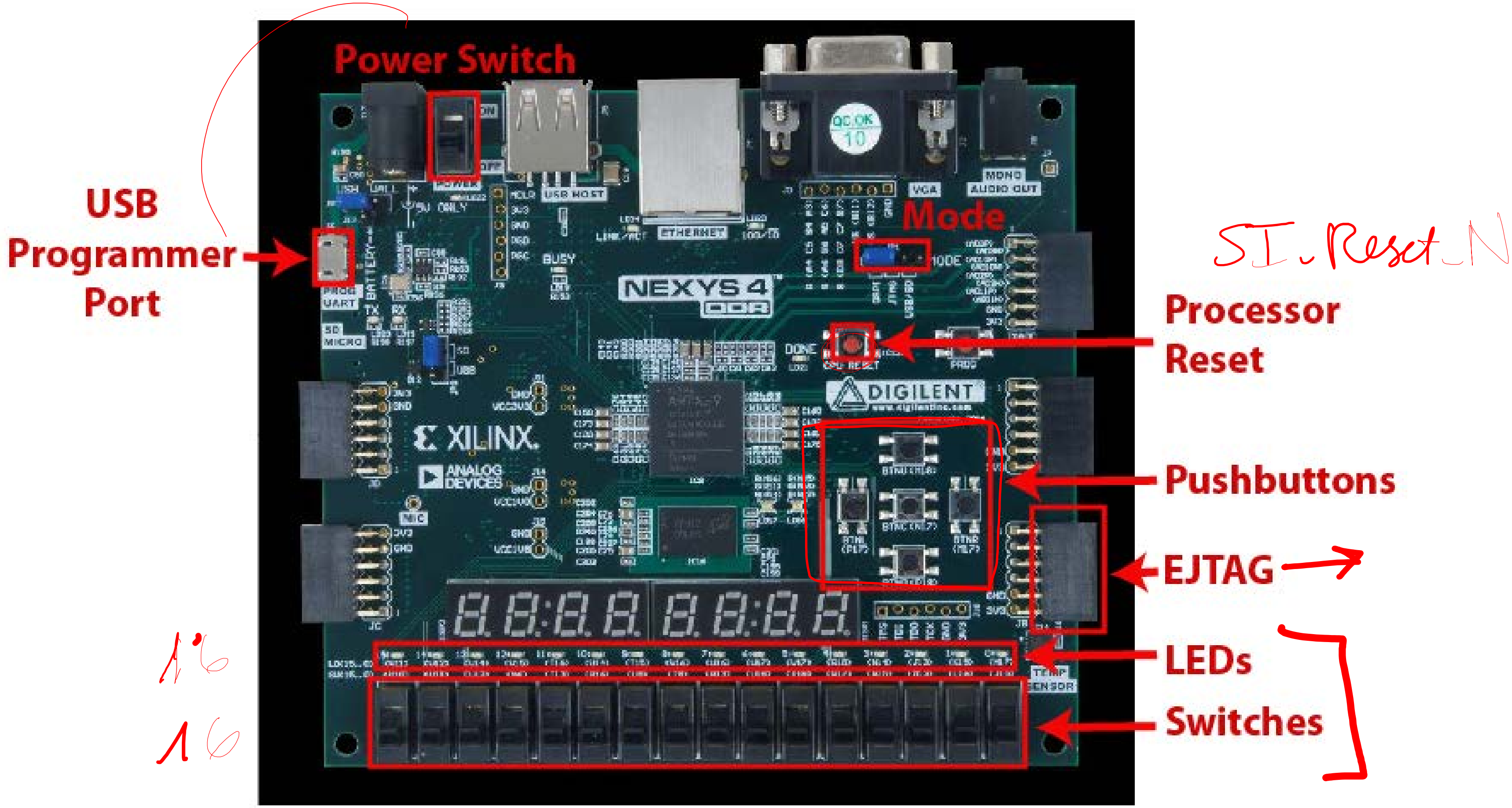
Cycle 2: **Read Data** (HRDATA)

MIPSfpga Interfaces: Nexys4 Board

Signal Name	Description
IO_LEDR[15:0]	LEDs
IO_Switch[15:0]	Switches
IO_PB[4:0]	Pushbuttons (BTNU, D, L, R, C)

IO: prefix for FPGA board I/O signals in Verilog files

Nexys4 DDR FPGA Board

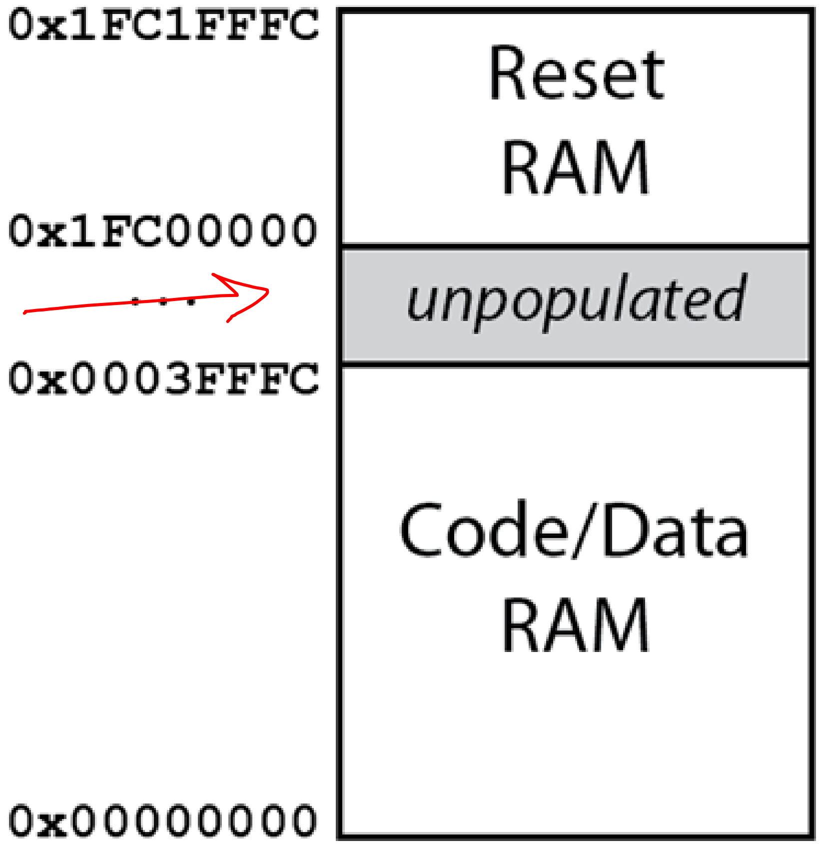


Memory-Mapped I/O

Signal Name	Virtual Address	Physical Address
IO_LEDR[15:0]	0xbf800000	0x1f800000
IO_Switch[15:0]	0xbf800008	0x1f800008
IO_PB[4:0]	0xbf80000c	0x1f80000c

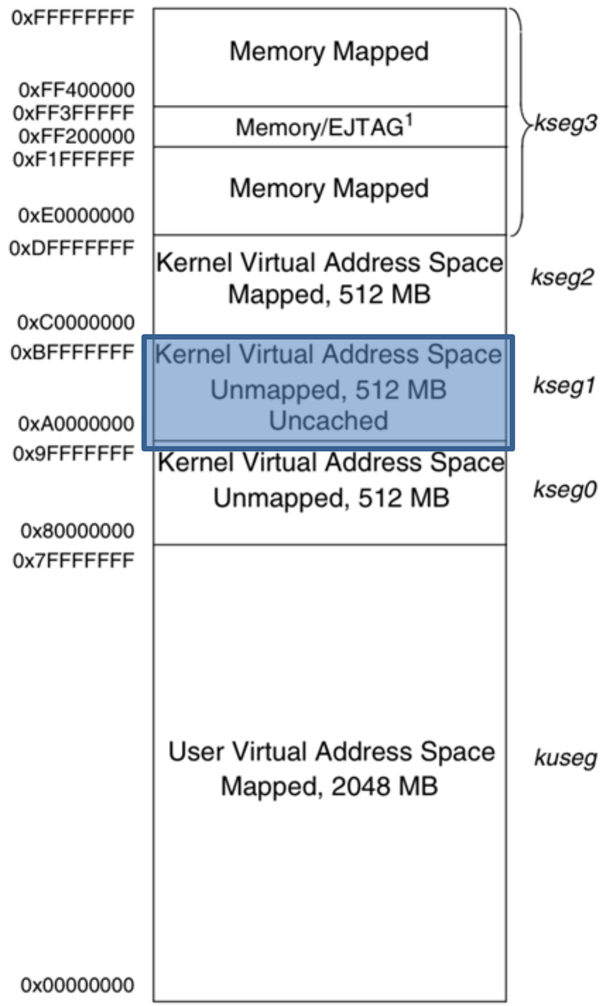
- Write to 0xbf800000 will **write to LEDs**
- Read from 0xbf800008 will **read value of switches**

MIPSfpga System: Physical Memory



No physical memory at memory-mapped addresses (0x1f800000+)

MIPSfpga Memory Map



- On reset, processor begins in kernel mode and jumps to the reset vector at address **0xBFC00000**
- In **kseg1:** uncached and unmapped in TLB (nothing is initialized)
  - All instructions **fetched from external memory** (not cache)
  - **0xBFC00000** maps to physical address **0x1FC00000**

Memory-Mapped I/O

Signal Name	Virtual Address	Physical Address
IO_LEDR[15:0]	0xbf800000	0x1f800000
IO_Switch[15:0]	0xbf800008	0x1f800008
IO_PB[4:0]	0xbf80000c	0x1f80000c

- Write to 0xbf800000 **writes to LEDs**

Memory-Mapped I/O

Signal Name	Virtual Address	Physical Address
IO_LEDR[15:0]	0xbf800000	0x1f800000
IO_Switch[15:0]	0xbf800008	0x1f800008
IO_PB[4:0]	0xbf80000c	0x1f80000c

- Write to 0xbf800000 **writes to LEDs**

```
// Write 0x543 to LEDs
addiu $7, $0, 0x543 # $7 = 0x543
lui   $5, 0xbf80    # $5 = 0xbf800000 (LED address)
sw    $7, 0($5)     # LEDs = 0x543
```

Memory-Mapped I/O

Signal Name	Virtual Address	Physical Address
IO_LEDR[15:0]	0xbf800000	0x1f800000
IO_Switch[15:0]	0xbf800008	0x1f800008
IO_PB[4:0]	0xbf80000c	0x1f80000c

- Write to 0xbf800000 **writes to LEDs**
- Read from 0xbf800008 **reads value of switches**

Memory-Mapped I/O

Signal Name	Virtual Address	Physical Address
IO_LEDR[15:0]	0xbf800000	0x1f800000
IO_Switch[15:0]	0xbf800008	0x1f800008
IO_PB[4:0]	0xbf80000c	0x1f80000c

- Write to 0xbf800000 **writes to LEDs**
- Read from 0xbf800008 **reads value of switches**

```
// Read value of switches into $10
lui    $5,   0xbf80    # $5 = 0xbf800000
lw     $10, 8($5)      # $10 = value of switches
```

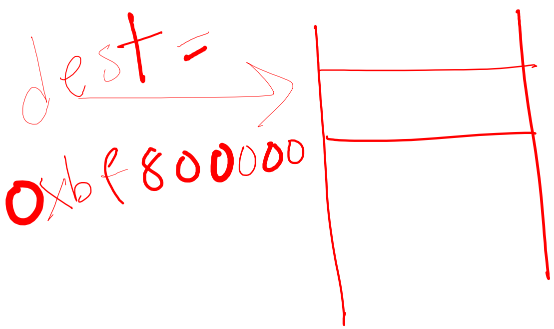
```
ori $5, $5, 0xC   # $5 = 0xbf80000c
lw $10, 0($5)
```

Example MIPS Program

C Code

```c
unsigned int val = 1;
volatile unsigned int* dest;
dest = 0xbf800000;

while (1) {
  *dest = val;   // LEDS = val
  val++;
}
```



Example MIPS Program

C Code

```
unsigned int val = 1;
volatile unsigned int* dest;
dest = 0xbf800000;

while (1) {
  *dest = val;
  val++;
}
```

MIPS Assembly Code

```
# $9=val, $8=0xbf800000
    addiu $9, $0, 1    # val=1
    lui   $8, 0xbf80   # address

L1: sw    $9, 0($8)    # write to addr
    addiu $9, $9, 1    # val++
    beqz  $0, L1       # loop
    nop                # branch delay
                       # slot
```

Example MIPS Program

C Code

```
unsigned int val = 1;
volatile unsigned int* dest;
dest = 0xbf800000;

while (1) {
  *dest = val;
  val++;
}
```

MIPS Assembly Code

```
# $9=val, $8=0xbf800000
    addiu $9, $0, 1    # val=1
    lui   $8, 0xbf80   # address

L1: sw    $9, 0($8)    # write to addr
    addiu $9, $9, 1    # val++
    beqz  $0, L1       # loop
    nop                # branch delay
                       # slot
```

Writes incremented values to memory address 0xbf800000 (LEDs)

How to Run Programs on MIPSfpga?

- **In Simulation**
- **In Hardware:**
  - **Load program into memory at synthesis**
  - **Load program into memory using EJTAG interface**

How to Run Programs on MIPSfpga?

- **In Simulation**
- **In Hardware:**
  - **Load program into memory at synthesis**
  - **Load program into memory using EJTAG interface**

MIPSfpga Overview

- History of MIPS architecture
- **MIPSfpga**
  - Background
  - Core and System
  - **Interfaces**
    - **System Interface**
    - **AHB-Lite Bus**
    - **EJTAG**

MIPSfpga Interfaces: EJTAG

- Used for programming and debugging the MIPSfpga core
- Borrows signal names and functionality from JTAG protocol

MIPSfpga Interfaces: EJTAG

Signal Name	Description
EJ_TDI	Test Data In
EJ_TDO	Test Data Out
EJ_TMS	Test Mode Select
EJ_TCK	Test Clock
EJ_DINT	Debug Interrupt Request
SI_ColdReset_N	Processor Reset
EJ_TRST_N_probe	Reset for EJTAG controller

EJ: prefix for EJTAG Interface signals in Verilog files

How to Run Programs on MIPSfpga?

- In Simulation
- In Hardware:
  - Load program into memory at synthesis
  - Load program into memory using EJTAG or USB interface

Example MIPS Program

C Code

```
unsigned int val = 1;
volatile unsigned int* dest;
dest = 0xbf800000;

while (1) {
  *dest = val;
  val++;
}
```

MIPS Assembly Code

```
# $9=val, $8=0xbf800000
    addiu $9, $0, 1    # val=1
    lui   $8, 0xbf80   # address

L1: sw    $9, 0($8)    # write to addr
    addiu $9, $9, 1    # val++
    beqz  $0, L1       # loop
    nop                # branch delay
                       # slot
```

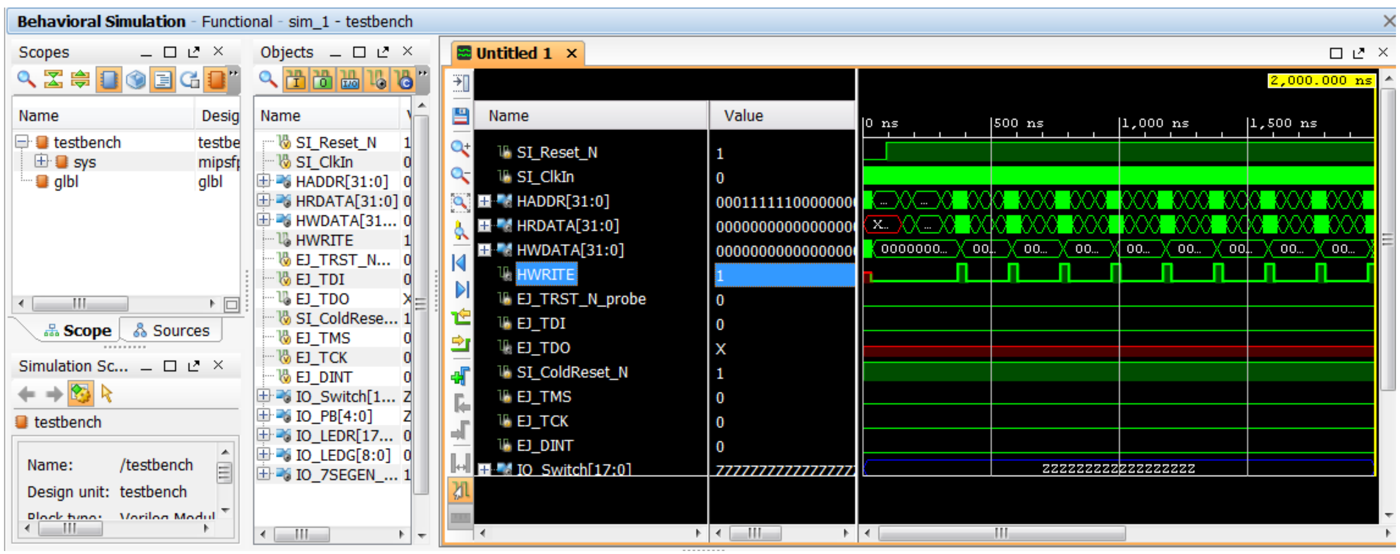
Writes incremented values to memory address 0xbf800000 (LEDs)

Machine Code

Machine Code	Instruction Address		MIPS Assembly Code
24090001	// bfc00000:		addiu $9, $0, 1
3c08bf80	// bfc00004:		lui $8, 0xbf80
ad090000	// bfc00008:	L1:	sw $9, 0($8)
25290001	// bfc0000c:		addiu $9, $9, 1
1000fffd	// bfc00010:		beqz $0, L1
00000000	// bfc00014:		nop

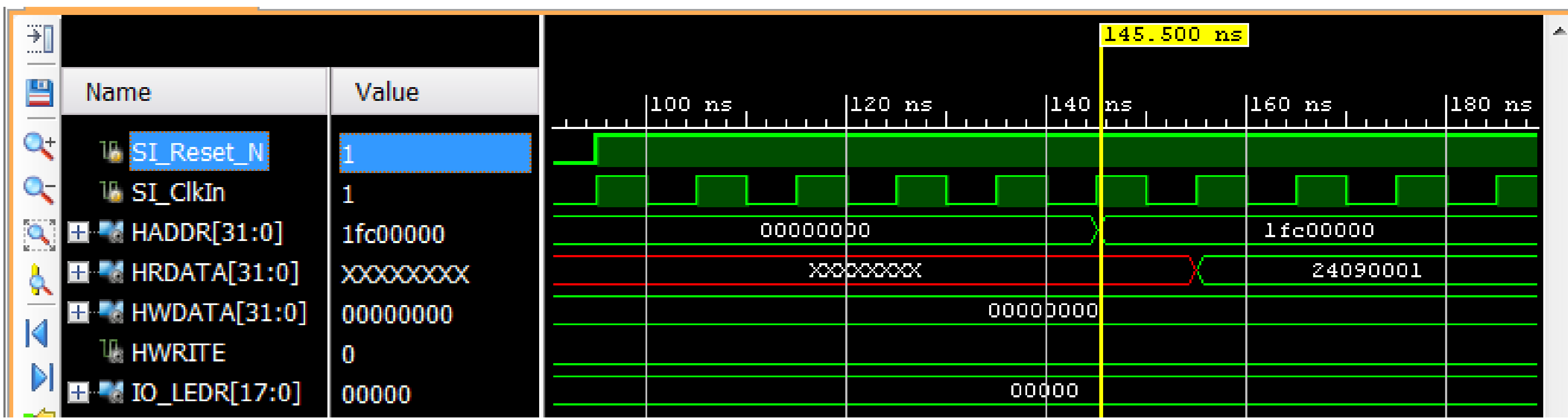
Simulating MIPSfpga

Machine Code	Instruction Address		MIPS Assembly Code
24090001	// bfc00000:		addiu $9, $0, 1
3c08bf80	// bfc00004:		lui $8, 0xbf80
ad090000	// bfc00008:	L1:	sw $9, 0($8)
25290001	// bfc0000c:		addiu $9, $9, 1
1000fffd	// bfc00010:		beqz $0, L1
00000000	// bfc00014:		nop



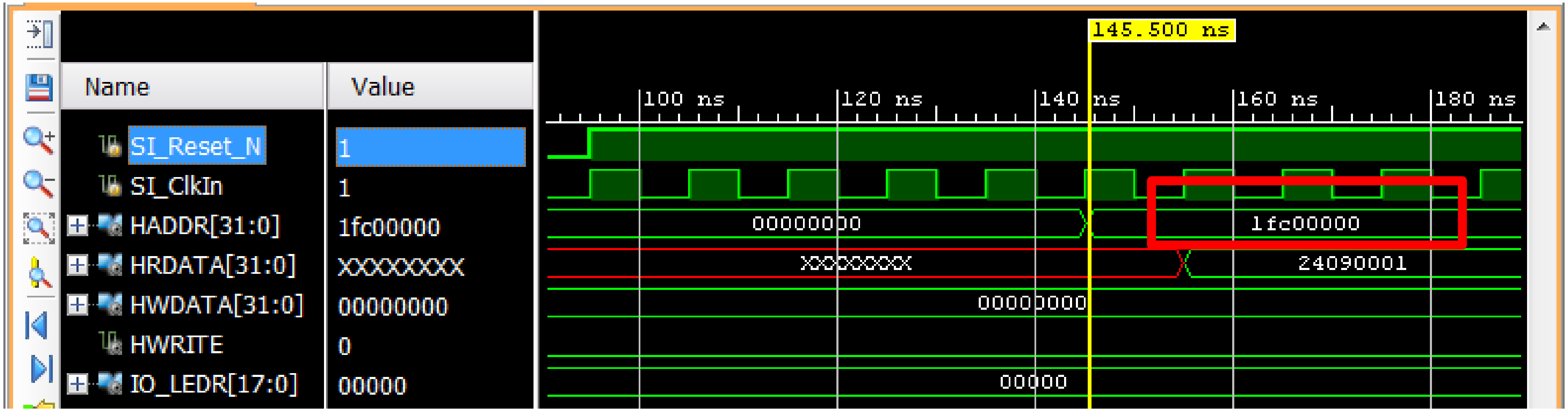
Simulating MIPSfpga

Machine Code	Instruction Address	MIPS Assembly Code
24090001	**// bfc00000:**	**addiu $9, $0, 1**
3c08bf80	// bfc00004:	lui $8, 0xbf80
ad090000	// bfc00008:	L1: sw $9, 0($8)
25290001	// bfc0000c:	addiu $9, $9, 1
1000fffd	// bfc00010:	beqz $0, L1
00000000	// bfc00014:	nop



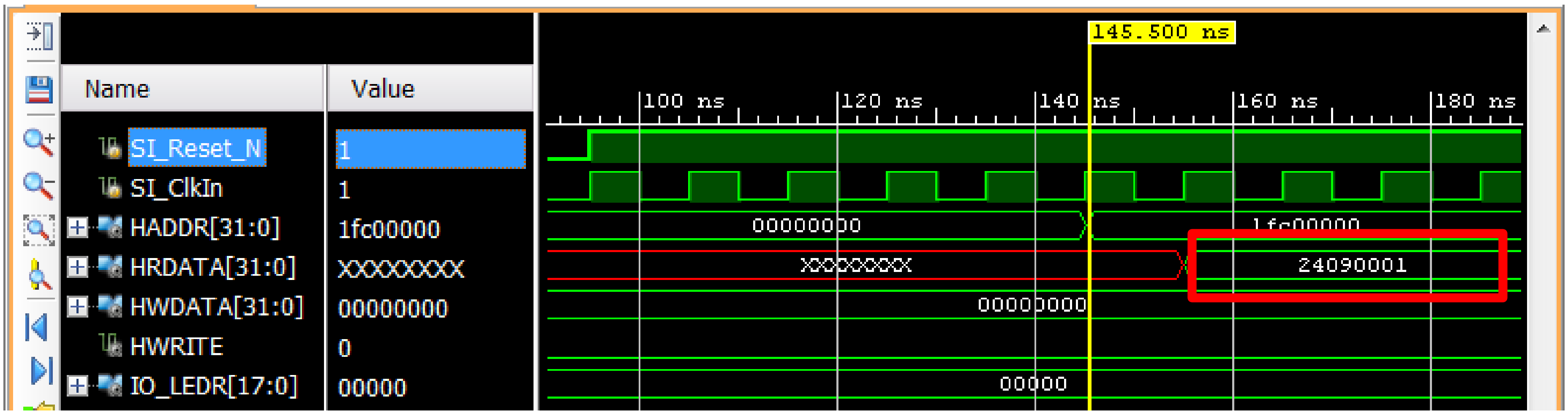
Simulating MIPSfpga

Machine Code	Instruction Address		MIPS Assembly Code
24090001	**// bfc00000:**		**addiu $9, $0, 1**
3c08bf80	// bfc00004:		lui $8, 0xbf80
ad090000	// bfc00008:	L1:	sw $9, 0($8)
25290001	// bfc0000c:		addiu $9, $9, 1
1000fffd	// bfc00010:		beqz $0, L1
00000000	// bfc00014:		nop



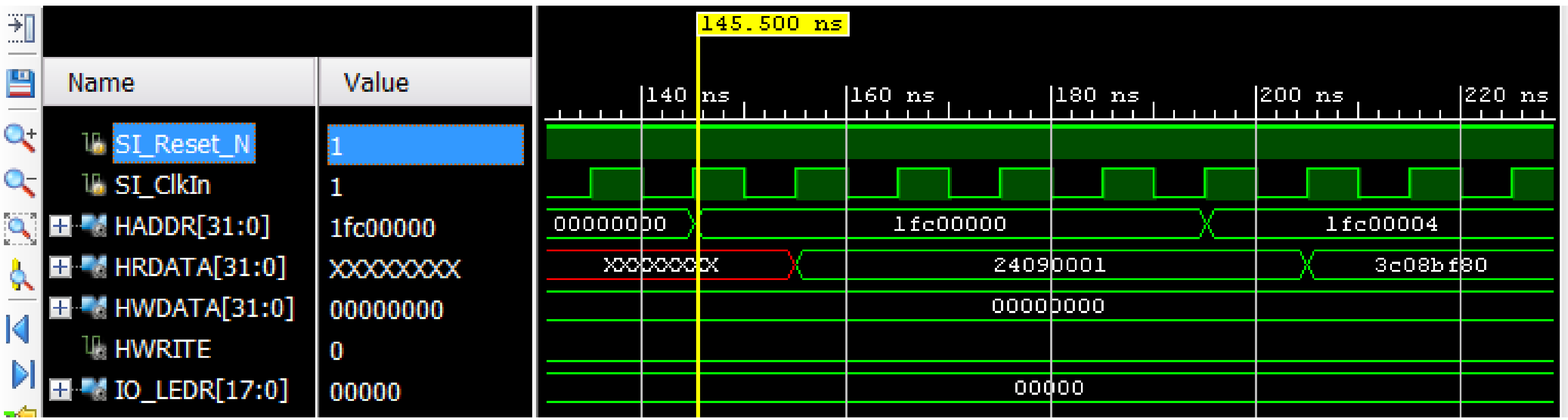
Simulating MIPSfpga

Machine Code	Instruction Address		MIPS Assembly Code
24090001	**// bfc00000:**		**addiu $9, $0, 1**
3c08bf80	// bfc00004:		lui $8, 0xbf80
ad090000	// bfc00008:	L1:	sw $9, 0($8)
25290001	// bfc0000c:		addiu $9, $9, 1
1000fffd	// bfc00010:		beqz $0, L1
00000000	// bfc00014:		nop



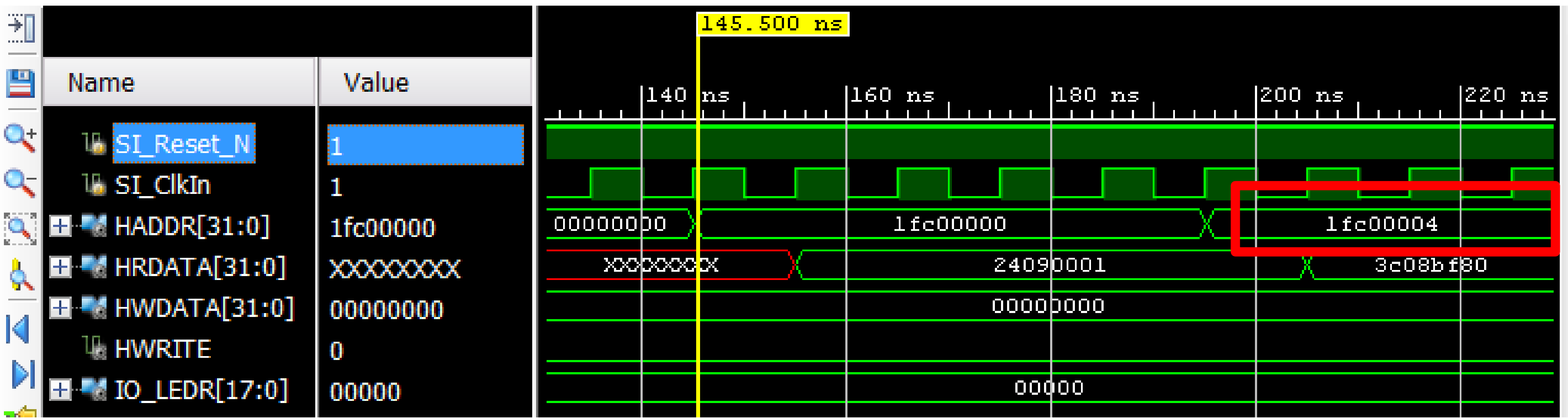
Simulating MIPSfpga

Machine Code	Instruction Address		MIPS Assembly Code
24090001	// bfc00000:		addiu $9, $0, 1
3c08bf80	**// bfc00004:**		**lui $8, 0xbf80**
ad090000	// bfc00008:	L1:	sw $9, 0($8)
25290001	// bfc0000c:		addiu $9, $9, 1
1000fffd	// bfc00010:		beqz $0, L1
00000000	// bfc00014:		nop



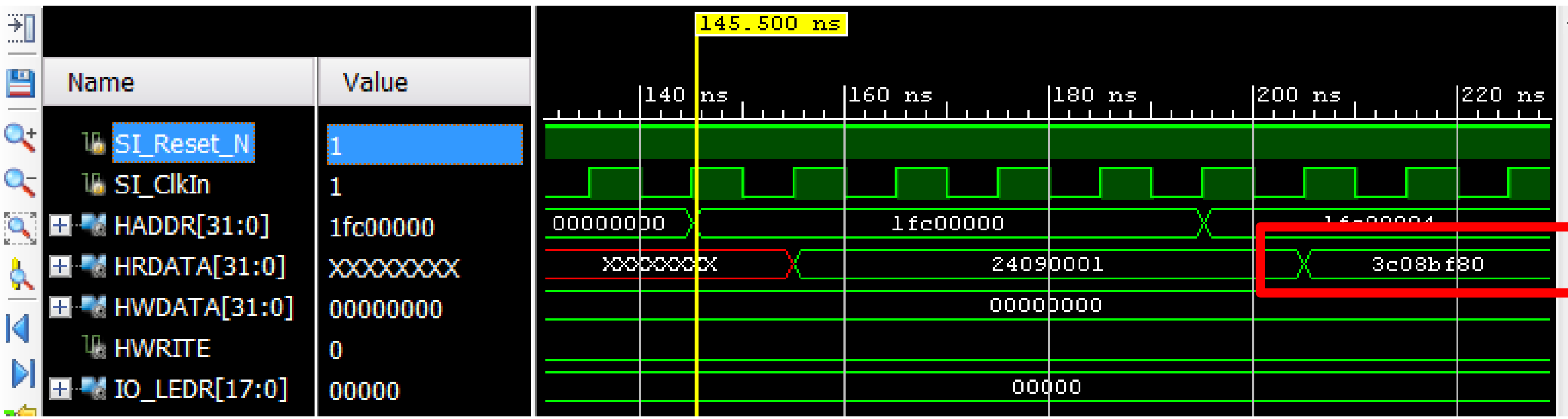
Simulating MIPSfpga

Machine Code	Instruction Address	MIPS Assembly Code
24090001	// bfc00000:	addiu $9, $0, 1
3c08bf80	**// bfc00004:**	**lui $8, 0xbf80**
ad090000	// bfc00008:	L1: sw $9, 0($8)
25290001	// bfc0000c:	addiu $9, $9, 1
1000fffd	// bfc00010:	beqz $0, L1
00000000	// bfc00014:	nop



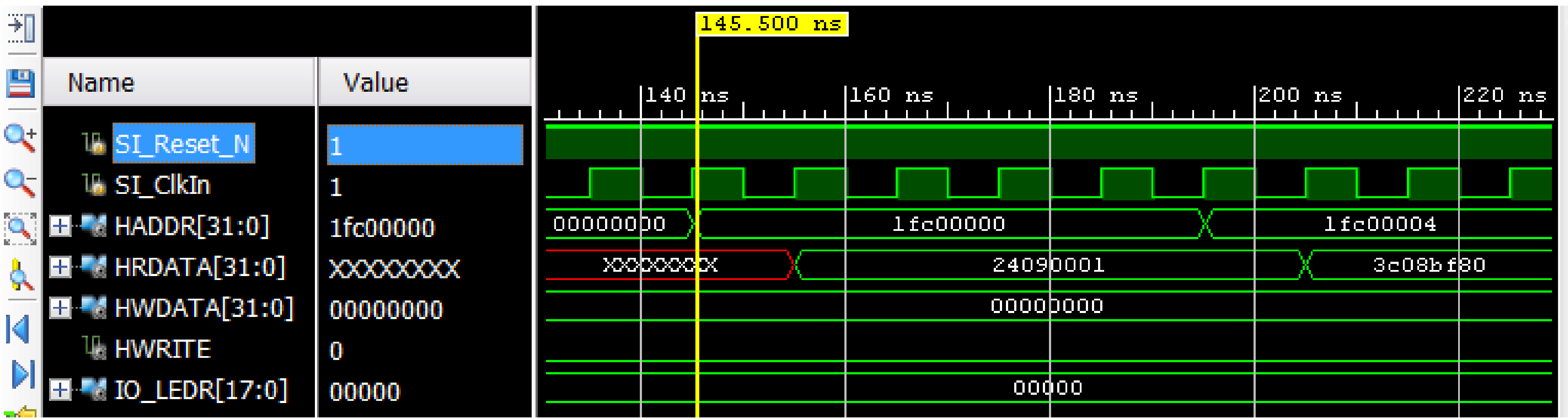
Simulating MIPSfpga

Machine Code	Instruction Address	MIPS Assembly Code
24090001	// bfc00000:	addiu $9, $0, 1
3c08bf80	**// bfc00004:**	**lui $8, 0xbf80**
ad090000	// bfc00008:	L1: sw $9, 0($8)
25290001	// bfc0000c:	addiu $9, $9, 1
1000fffd	// bfc00010:	beqz $0, L1
00000000	// bfc00014:	nop



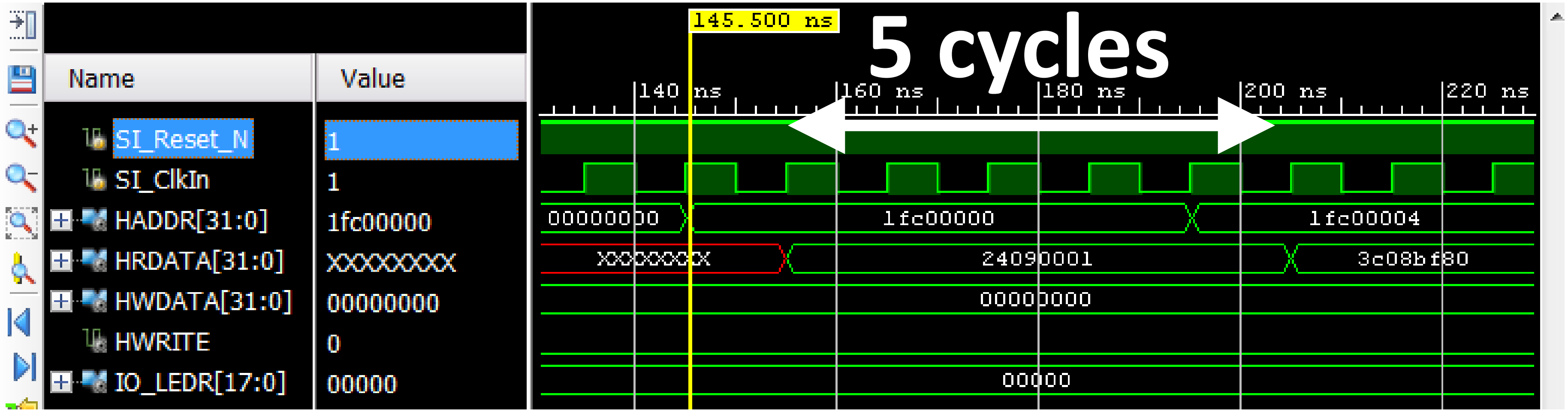
Simulating MIPSfpga

Each instruction takes 5 cycles instead of 1 before the caches are initialized.



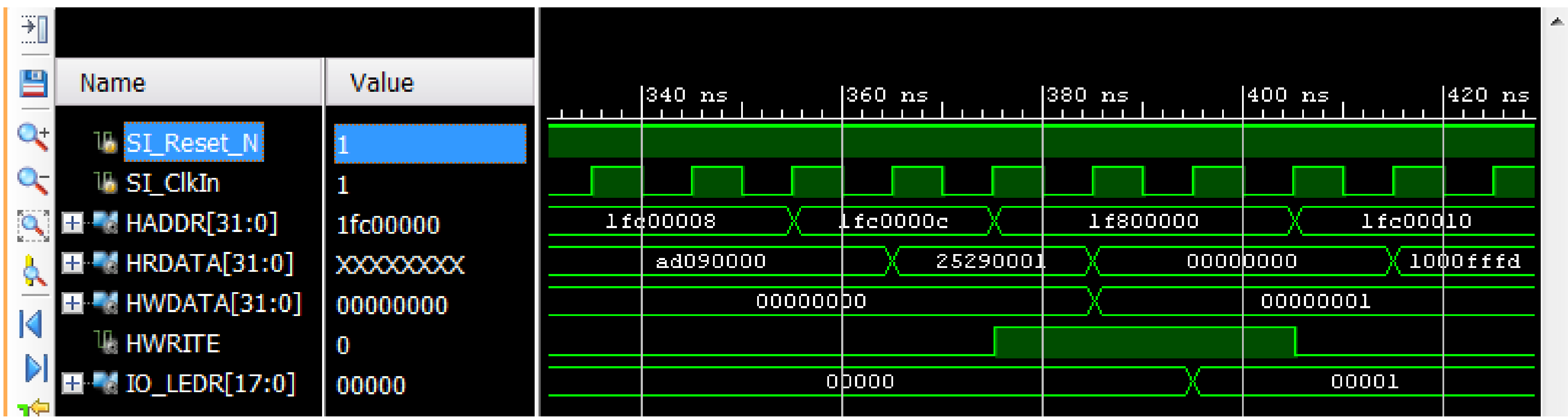
Simulating MIPSfpga

Each instruction takes 5 cycles instead of 1 before the caches are initialized.



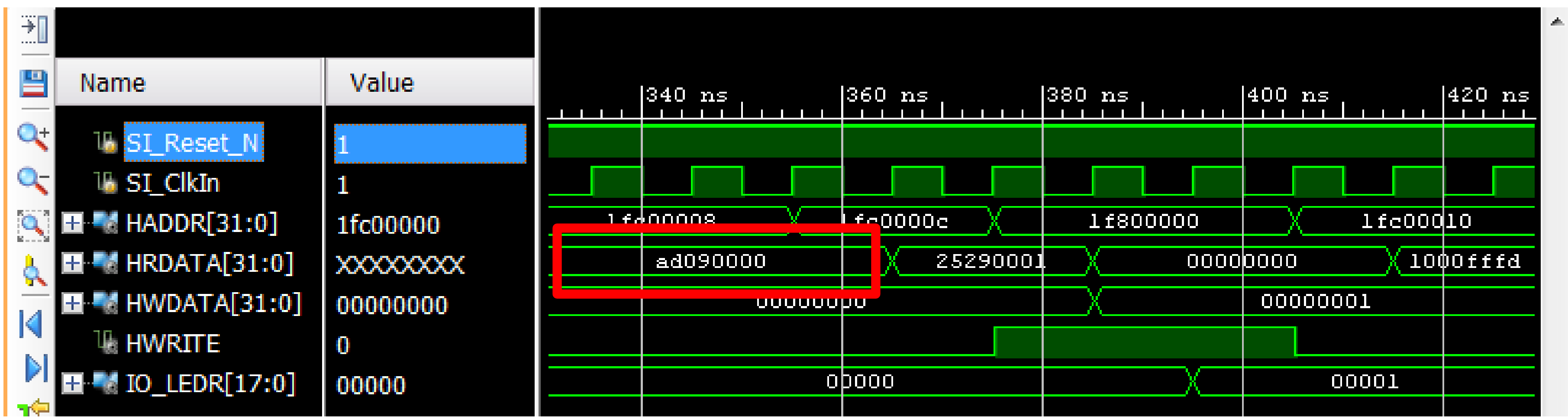
Simulating MIPSfpga

Machine Code	Instruction Address		MIPS Assembly Code
24090001	// bfc00000:		addiu $9, $0, 1
3c08bf80	// bfc00004:		lui $8, 0xbf80
ad090000	**// bfc00008:**	**L1:**	**sw $9, 0($8)**
25290001	// bfc0000c:		addiu $9, $9, 1
1000fffd	// bfc00010:		beqz $0, L1
00000000	// bfc00014:		nop



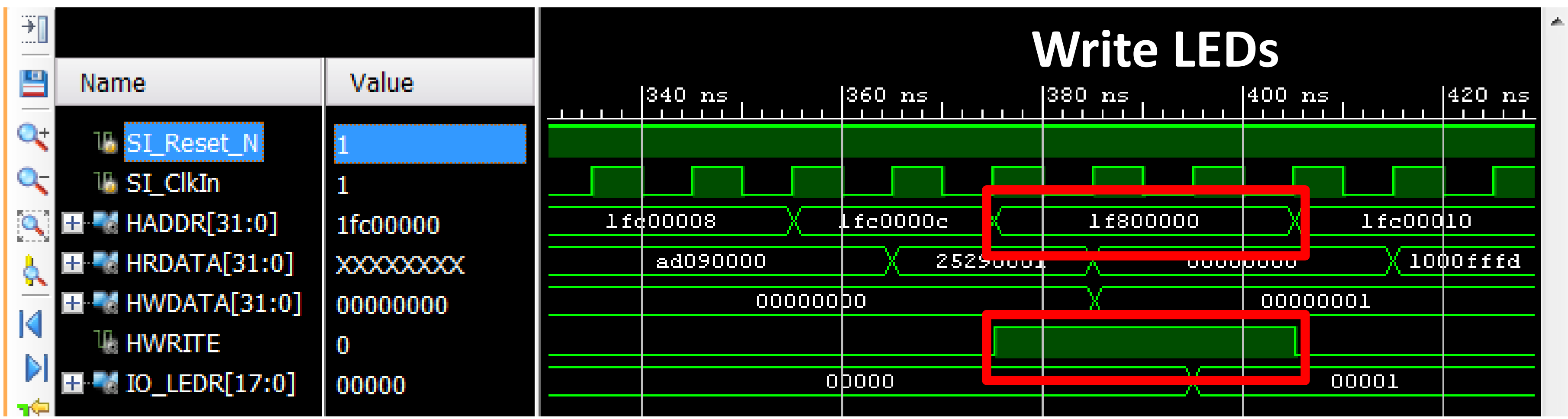
Simulating MIPSfpga

Machine Code	Instruction Address		MIPS Assembly Code
24090001	// bfc00000:		addiu $9, $0, 1
3c08bf80	// bfc00004:		lui $8, 0xbf80
ad090000	**// bfc00008:**	**L1:**	**sw $9, 0($8)**
25290001	// bfc0000c:		addiu $9, $9, 1
1000fffd	// bfc00010:		beqz $0, L1
00000000	// bfc00014:		nop



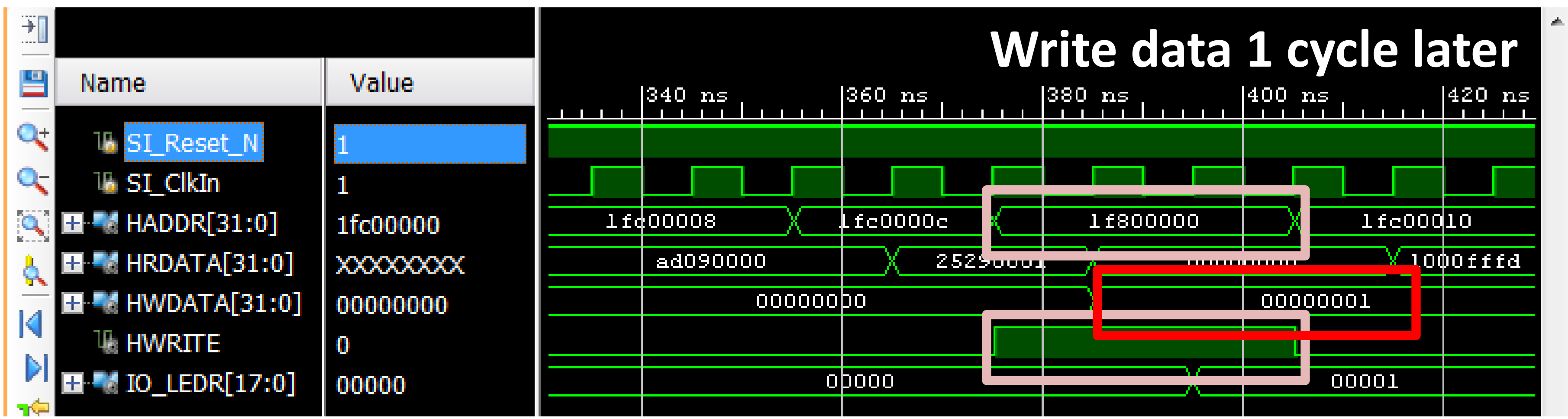
Simulating MIPSfpga

Machine Code	Instruction Address		MIPS Assembly Code
24090001	// bfc00000:		addiu $9, $0, 1
3c08bf80	// bfc00004:		lui $8, 0xbf80
ad090000	**// bfc00008:**	**L1:**	**sw $9, 0($8)**
25290001	// bfc0000c:		addiu $9, $9, 1
1000fffd	// bfc00010:		beqz $0, L1
00000000	// bfc00014:		nop



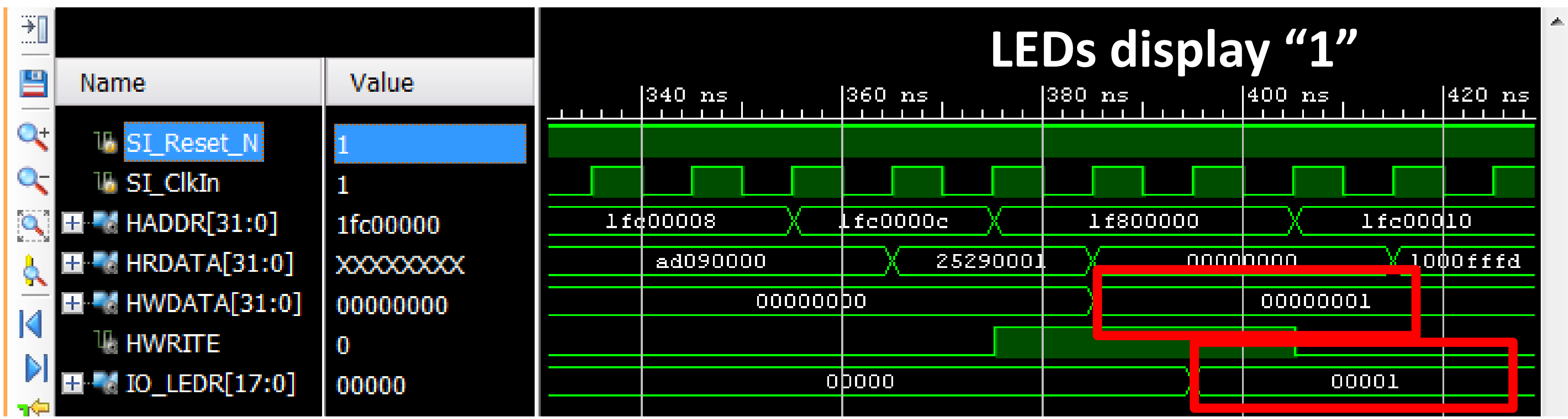
Simulating MIPSfpga

Machine Code	Instruction Address		MIPS Assembly Code
24090001	// bfc00000:		addiu $9, $0, 1
3c08bf80	// bfc00004:		lui $8, 0xbf80
ad090000	// bfc00008:	L1:	sw $9, 0($8)
25290001	// bfc0000c:		addiu $9, $9, 1
1000fffd	// bfc00010:		beqz $0, L1
00000000	// bfc00014:		nop



Simulating MIPSfpga

Machine Code	Instruction Address		MIPS Assembly Code
24090001	// bfc00000:		addiu $9, $0, 1
3c08bf80	// bfc00004:		lui $8, 0xbf80
ad090000	**// bfc00008:**	**L1:**	**sw $9, 0($8)**
25290001	// bfc0000c:		addiu $9, $9, 1
1000fffd	// bfc00010:		beqz $0, L1
00000000	// bfc00014:		nop



How to Run Programs on MIPSfpga?

- **In Simulation**
- **In Hardware:**
  - **Load program into memory at synthesis**
  - **Load program into memory using EJTAG or USB interface**

Program Loaded at Synthesis

Memory Module

```
module ram_reset_dual_port
  initial begin
    $readmemh("ram_reset_init.txt",
ram);
  end
```

0x1fc00000

Memory Initialization File

ram_reset_init.txt:

Machine Code	Instruction Address	MIPS Assembly Code
24090001	// bfc00000:	addiu $9, $0, 1
3c08bf80	// bfc00004:	lui $8, 0xbf80
ad090000	// bfc00008: L1:	sw $9, 0($8)
25290001	// bfc0000c:	addiu $9, $9, 1
3c050026	**// bfc00010: delay:**	**lui $5, 0x026**
34a525a0	**// bfc00014:**	**ori $5, $5, 0x25a0**
00003020	**// bfc00018:**	**add $6, $0, $0**
00a63822	**// bfc0001c: L2:**	**sub $7, $5, $6**
20c60001	**// bfc00020:**	**addi $6, $6, 1**
1ce0fffd	**// bfc00024:**	**bgtz $7, L2**
00000000	**// bfc00028:**	**nop**
1000fff6	**// bfc0002c:**	**beq $0, $0, L1**
00000000	**// bfc00030:**	**nop**

Add delay so eye can see LEDs change values

Labs 4-6 Overview

- **Lab 4:** Creating a Vivado project for MIPSfpga, Programming MIPSfpga in C & Assembly
- **Lab 5:** Adding peripherals: millisecond timer and buzzer
- **Lab 6:** Adding peripherals: SPI LCD

Adding Peripherals: Memory-Mapped I/O

Example: Add 7-Segment displays as memory-mapped I/O to MIPSfpga

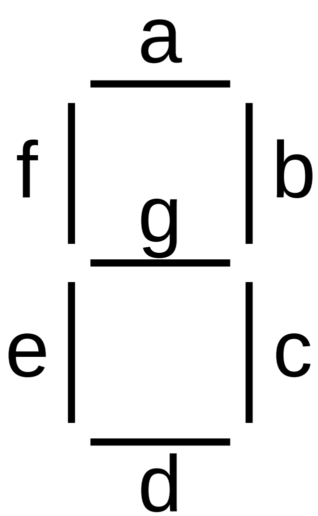
7-Segment Displays

Example: Add 7-Segment displays as memory-mapped I/O to MIPSfpga

Process:

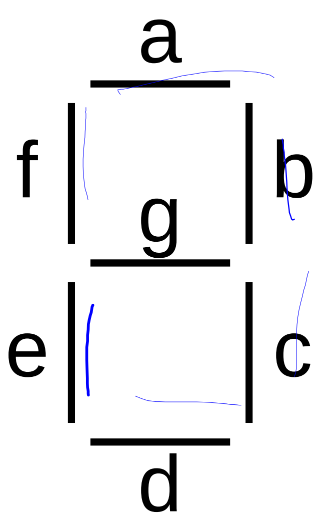
1. Add hardware to drive 7-segment displays
2. Memory-map digits and enables
3. Modify MIPSfpga interface to drive 7-segment and enable pins

7-Segment Displays

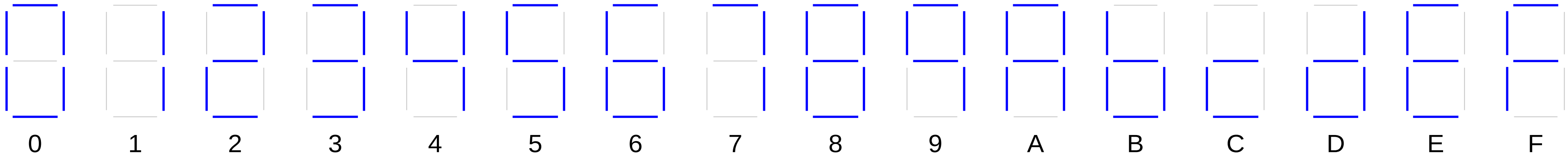


Display digits by selecting which segments to light up

7-Segment Displays



For example, 0 lights up: a,b,c,d,e,f
1 lights up: b,c
2 lights up: a,b,d,e,g
etc.



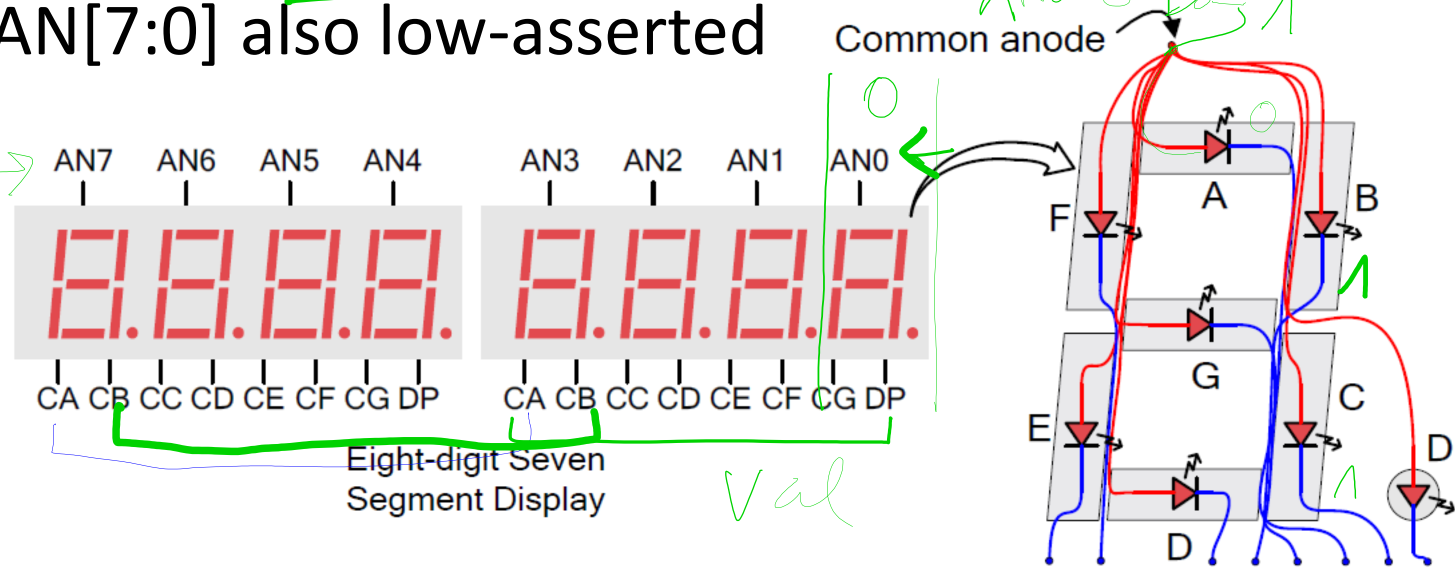
7-Segment Displays: low asserted

```
module mipsfpga_ahb_sevensegdec(input logic [3:0] data,
                                output reg logic [6:0] segments);
  always @(*) always_comb
    case(data)         // abc_defg
      4'h0: segments = 7'b000_0001;
      4'h1: segments = 7'b100_1111;
      4'h2: segments = 7'b001_0010;
      ...
      default:
            segments = 7'b111_1111;
    endcase
endmodule
```

Segments are low-asserted

Nexys4 DDR 7-Segment Displays

- 8 7-segment digits
- Each digit connects to same segment inputs (CA-CG)
- Enable signal (AN[7:0]) determines which digits are on – AN[7:0] also low-asserted

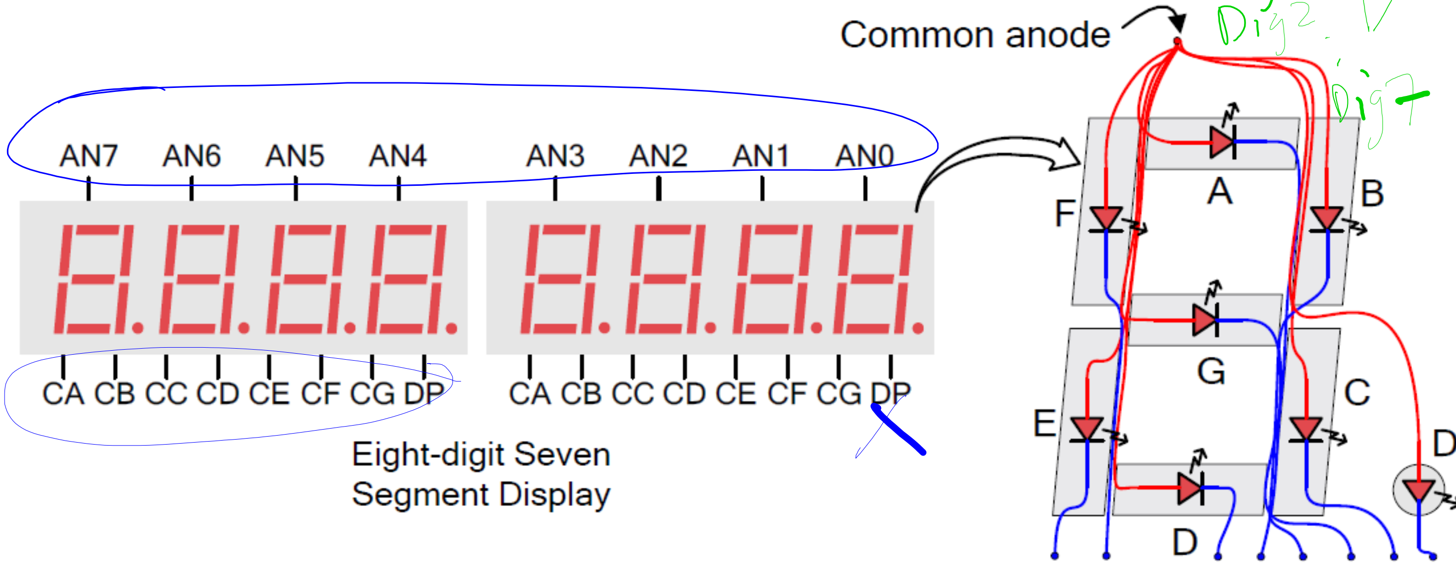


(Figure courtesy Nexys4 DDR Reference Manual)

Nexys4 DDR 7-Segment Displays

Example:

- AN[7:0] = 11111110_2 (only right-most digit is ON)
- Displayed value determined by CA-CG

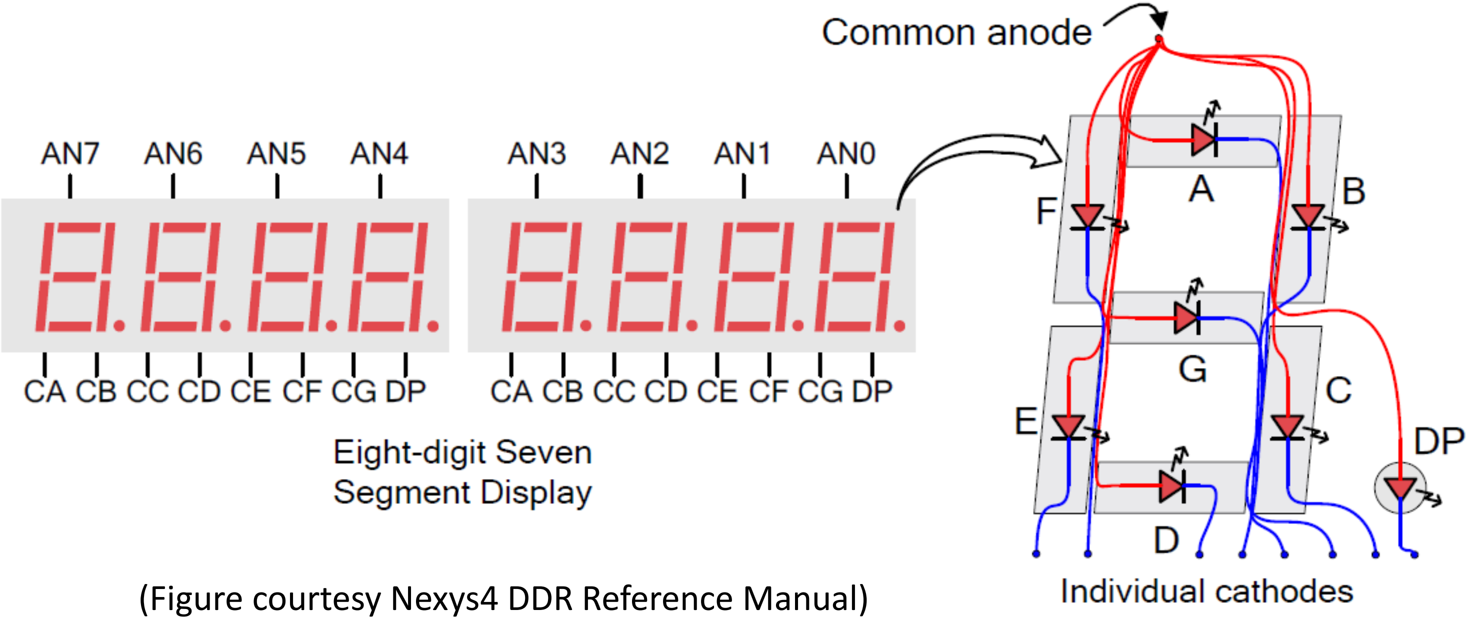


(Figure courtesy Nexys4 DDR Reference Manual)

Nexys4 DDR 7-Segment Displays

To drive multiple digits:

- Drive each digit one at a time
- But fast enough that can’t detect flicker



(Figure courtesy Nexys4 DDR Reference Manual)

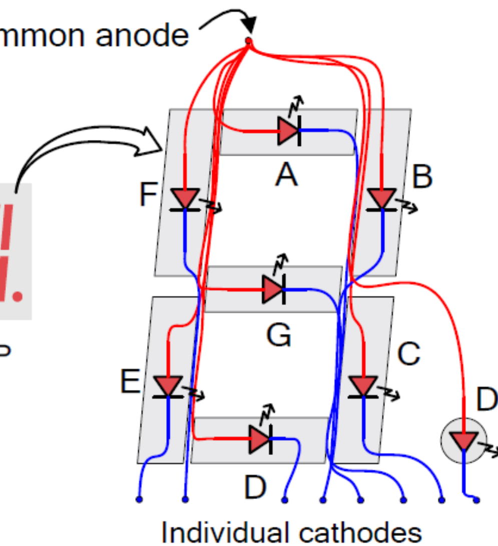
Nexys4 DDR 7-Segment Displays

0xF0

Every ~2 ms drive next display:

- At t=0, AN[7:0] = 11111110, CA-CG for Digit 0
- At t=2ms, AN[7:0] = 11111101, CA-CG for Digit 1
- At t=4ms, AN[7:0] = 11111011, CA-CG for Digit 2
- And so on…

Eight-digit Seven Segment Display

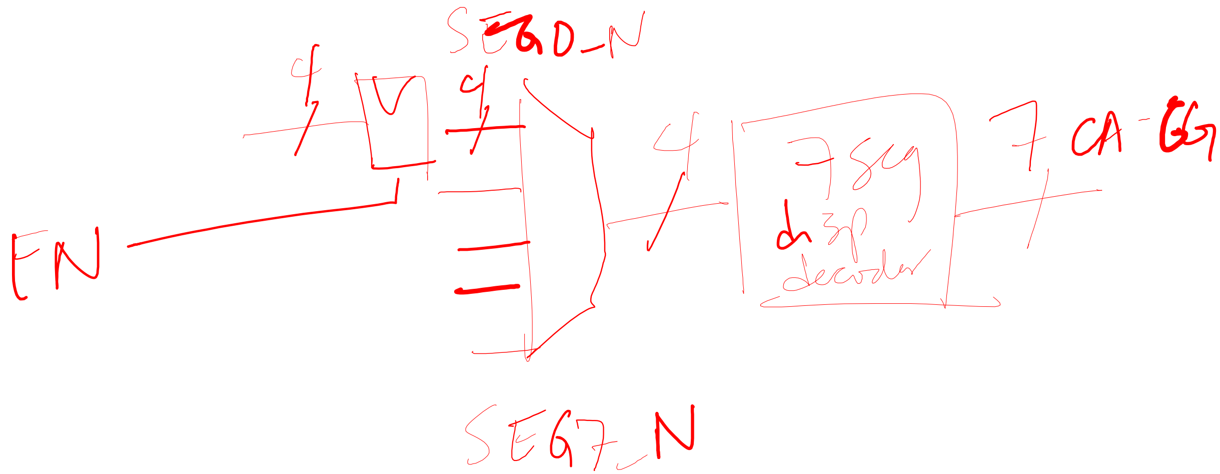


(Figure courtesy Nexys4 DDR Reference Manual)

Hardware for Nexys4 DDR 7-Segs

Need 9 memory-mapped registers:

- 1 to store which digits are enabled (SEGEN_N[7:0])
- 8 to store each digit value (SEG0_N[3:0], SEG1_N[3:0], … SEG7_N[3:0])



Hardware for Nexys4 DDR 7-Segs

Need 9 memory-mapped registers:

- 1 to store which digits are enabled (SEGEN_N[7:0])
- 8 to store each digit value (SEG0_N[3:0], SEG1_N[3:0], … SEG7_N[3:0])

A 3-bit counter (running at ~500 Hz, i.e., period = 2ms) selects each digit in sequence and displays it if it's enabled.

7-Segment Displays

Goal: Add 7-Segment displays as memory-mapped I/O to MIPSfpga

Process:

1. Add hardware to drive 7-segment displays
2. **Memory-map digits and enables**
3. Modify MIPSfpga interface to drive 7-segment and enable pins

Memory-Map Enables and Digits

Register Name	Description	Memory Address
SEGEN_N[7:0]	Enables	0xbf800010
SEG0_N[3:0]	Digit 0 value	0xbf800014
SEG1_N[3:0]	Digit 1 value	0xbf800018
SEG2_N[3:0]	Digit 2 value	0xbf80001c
SEG3_N[3:0]	Digit 3 value	0xbf800020
SEG4_N[3:0]	Digit 4 value	0xbf800024
SEG5_N[3:0]	Digit 5 value	0xbf800028
SEG6_N[3:0]	Digit 6 value	0xbf80002c
SEG7_N[3:0]	Digit 7 value	0xbf800030

Writing values to 7-Segment Displays

```
...
volatile int *IO_7SEGEN = (int*)0xbf800010;
volatile int *IO_7SEG0  = (int*)0xbf800014;
volatile int *IO_7SEG1  = (int*)0xbf800018;
...

*IO_7SEGEN = 0xF0;        // enable 4 right-most digits
*IO_7SEG0 = 5;            // write to 7SEG0
*IO_7SEG1 = 0xC;          // write to 7SEG1
...
```

Digit 0

Digit 1

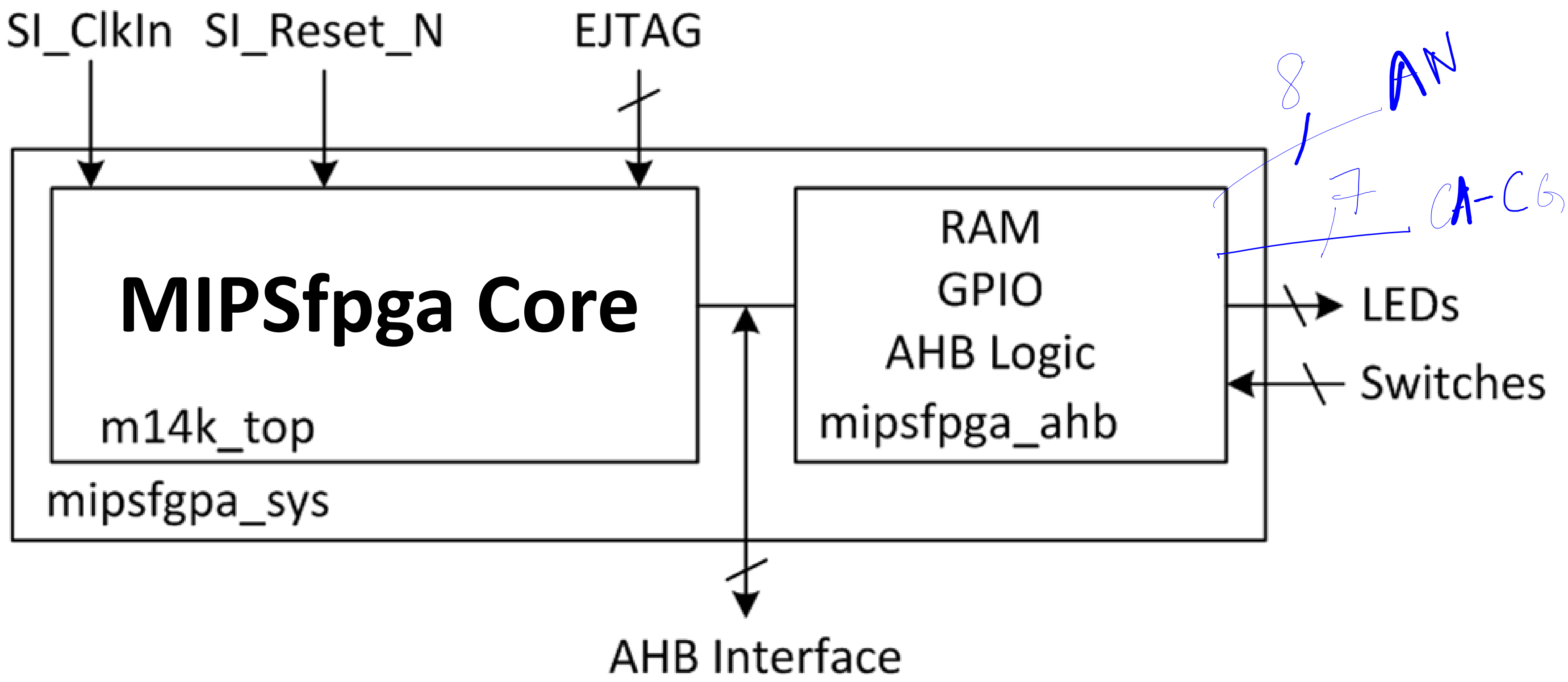
7-Segment Displays

Goal: Add 7-Segment displays as memory-mapped I/O to MIPSfpga

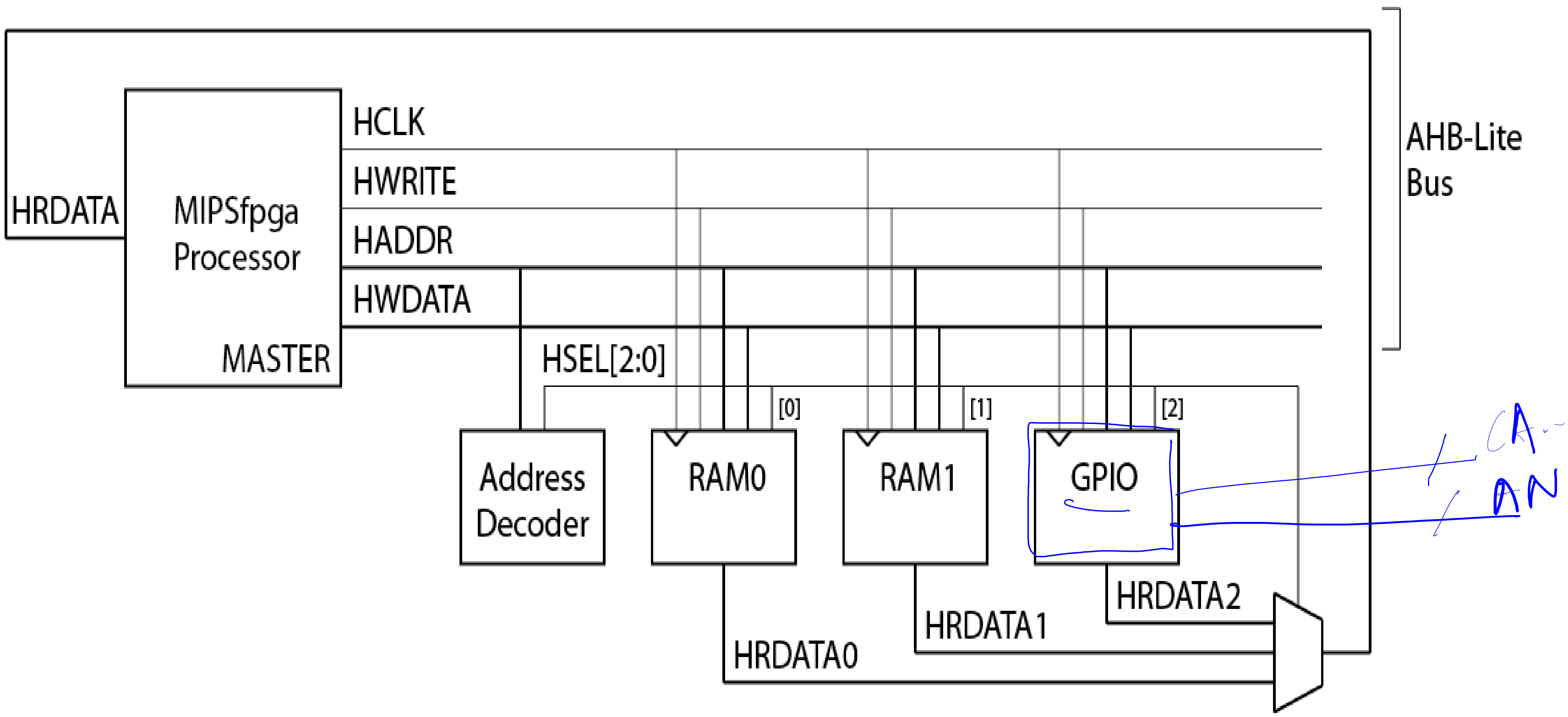
Process:

1. Add hardware to drive 7-segment displays
2. Memory-map digits and enables
3. **Modify MIPSfpga interface to drive 7-segment and enable pins**

MIPSfpga System



AHB-Lite Bus


HRDATA
MIPSfpga
Processor
MASTER
HCLK
HWRITE
HADDR
HWDATA
HSEL[2:0]
[0]
[1]
[2]
Address
Decoder
RAM0
RAM1
GPIO
HRDATA2
HRDATA1
HRDATA0
AHB-Lite
Bus
CA
AN

MIPSfpga Interface

GPIO Module & MIPSfpga System Outputs:

```
...
output       [  7: 0] IO_7SEGEN_N,
output       [  6: 0] IO_7SEG_N,
```

AN
CA – G

MIPSfpga Interface

GPIO Module & MIPSfpga System Outputs:

```
   ...
   output      [  7: 0] IO_7SEGEN_N,
   output      [  6: 0] IO_7SEG_N
```

Nexys4 DDR Wrapper Module:

```
module mipsfpga_nexys4_ddr( ...
      output [ 7:0] AN,
      output        CA, CB, CC, CD, CE, CF, CG);
  ...
  mipsfpga_sys mipsfpga_sys(
                  ...
                  .IO_7SEGEN_N(AN),
                  .IO_7SEG_N({CA,CB,CC,CD,CE,CF,CG})
```

MIPSfpga Interface: Nexys4 DDR Pins

MIPSfpga_Nexys4DDR.xdc:

```
set_property -dict { PACKAGE_PIN T10   IOSTANDARD
LVCMOS33 } [get_ports { CA }];
set_property -dict { PACKAGE_PIN R10   IOSTANDARD
LVCMOS33 } [get_ports { CB }];
...
set_property -dict { PACKAGE_PIN J17   IOSTANDARD
LVCMOS33 } [get_ports { AN[0] }];
set_property -dict { PACKAGE_PIN J18   IOSTANDARD
LVCMOS33 } [get_ports { AN[1] }];
...
```