Explanation for Homework IV

Advisor: Lih-Yih Chiou
Speaker: Yun-Ru Chen
Date: 2020/12/02

Outline

- New Instructions
- CSR
- Slaves
- Simulation Commands
- Verification

New Instructions

31 20	19 15	14 12	11 7	6 0		
imm[11:0]	**rs1**	**funct3**	**rd**	**opcode**	**Mnemonic**	**Description**

csr	rs1	001	rd	1110011	CSRRW	rd = csr, if #rd != 0
						csr = rs1

csr	rs1	010	rd	1110011	CSRRS	rd = csr, if #rd != 0
						csr = csr | rs1, if that csr bit is writable and #rs1 != 0

csr	rs1	011	rd	1110011	CSRRC	rd = csr, if #rd != 0
						csr = csr & (~rs1), if #rs1 != 0

csr	uimm	101	rd	1110011	CSRRWI	rd = csr, if #rd != 0
						csr = uimm(zero-extend)

csr	uimm	110	rd	1110011	CSRRSI	rd = csr, if #rd != 0
						csr = csr | uimm(zero-extend), if that csr bit is writable and uimm != 0

csr	uimm	111	rd	1110011	CSRRCI	rd = csr, if #rd != 0
						csr = csr & (~uimm(zero-extend)), if uimm != 0

0011000 00010	00000	000	00000	1110011	MRET	Return from traps in Machine Mode
0001000 00101	00000	000	00000	1110011	WFI	Wait for interrupt

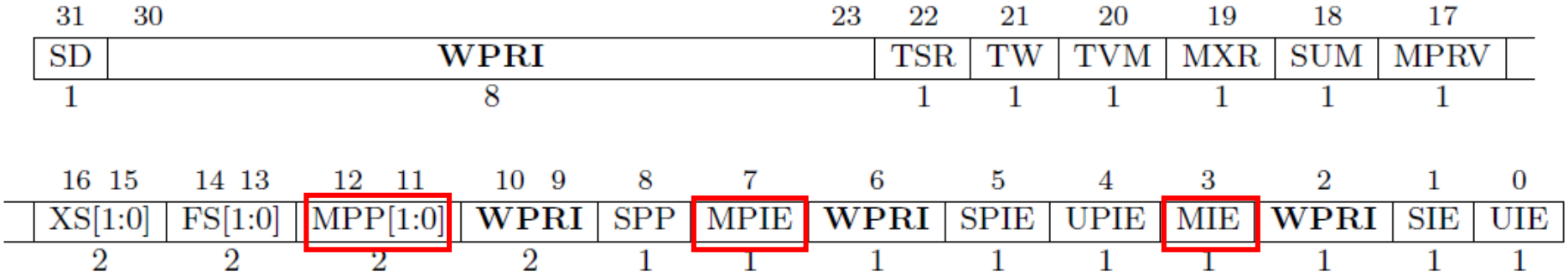
CSR

Table 1-4: Control Status Register (CSR)

Address	Privilege	Name	Requirement
0x300	M	mstatus	MIE, MPIE, MPP. Others hardwire to 0
0x304	M	mie	MEIE. Others hardwire to 0
0x305	M	mtvec	Hardwire to 32’h0001_0000
0x341	M	mepc	MEPC[31:2]. Others hardwire to 0
0x344	M	mip	MEIP. Others hardwire to 0
0xB00	M	mcycle	All
0xB02	M	minstret	All
0xB80	M	mcycleh	All
0xB82	M	minstreth	All

CSR - mstatus

Keeps track of and controls the current operating state.



- When interrupt is taken
  - MPIE <= MIE
  - MIE <= 0
  - MPP <= 2'b11(machine mode)

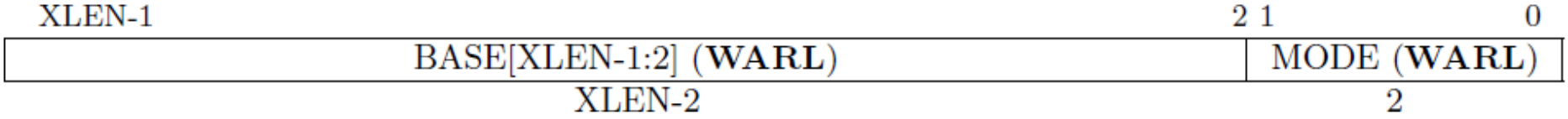
- When interrupt is return
  - MPIE <= 1
  - MIE <= MPIE
  - MPP <= 2'b11(machine mode)

★ Other bits can hardwire to 0

★WFI should be unaffected by MIE

CSR - mtvec

- Store the address where ISR start



- When interrupt is taken
  - PC <= { mtvec[31:2],2'b00 }

★ mtvec is hardwire to 0x0001_0000

CSR – mip & mie

mip: Contains information on pending interrupts

XLEN-1	12	11	10	9	8	7	6	5	4	3	2	1	0
	WIRI	MEIP	WIRI	SEIP	UEIP	MTIP	WIRI	STIP	UTIP	MSIP	WIRI	SSIP	USIP
XLEN-12		1	1	1	1	1	1	1	1	1	1	1	1

- MEIP: External interrupt is pending
  - MEIP = external interrupt signal

★ Other bits hardwire to 0

mie: Contains interrupt enable bits

XLEN-1	12	11	10	9	8	7	6	5	4	3	2	1	0
	WPRI	MEIE	WPRI	SEIE	UEIE	MTIE	WPRI	STIE	UTIE	MSIE	WPRI	SSIE	USIE
XLEN-12		1	1	1	1	1	1	1	1	1	1	1	1

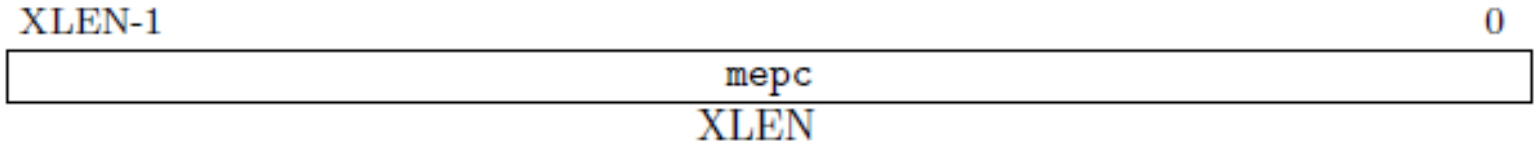
- MEIE: External interrupt enable
  - MEIE is set by CSR instructions

★ Other bits hardwire to 0

★ WFI shouldn't ignore the MEIE

CSR - mepc

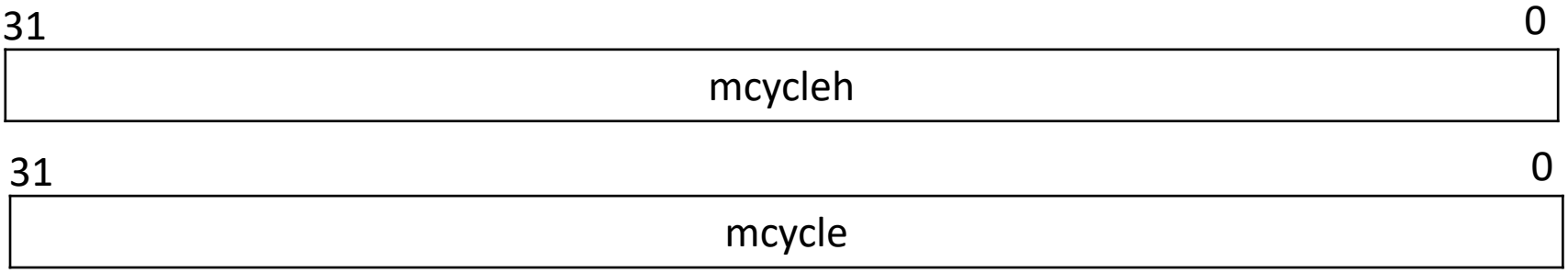
- Store the address of the instruction that encountered the interrupt



- When interrupt is taken
  - mepc <= pc+? (depends on your design)
- When interrupt return
  - pc <= mepc

CSR –mcycleh & mcycle

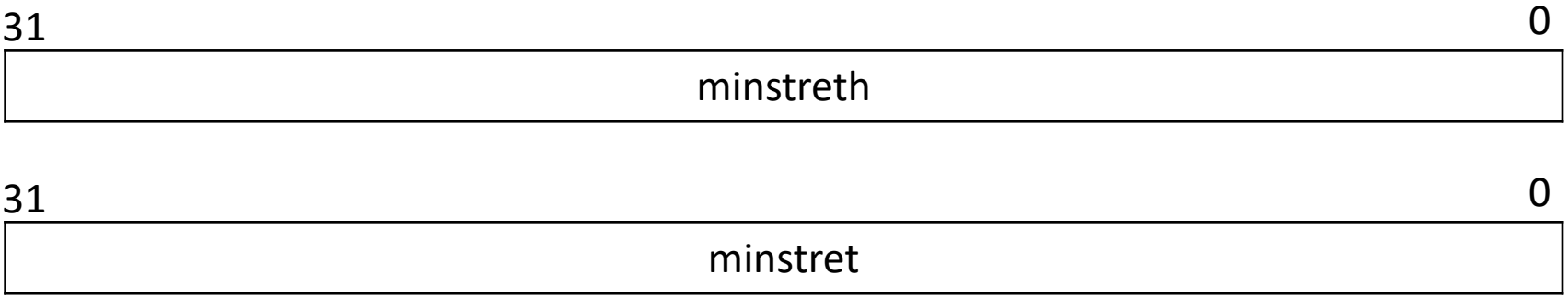
- Holds the number of cycles that hardware has executed
- Divide **64-bit mcycle** into mcycleh and mcycle



- Count every cycle

CSR – minstreth & minstret

- Holds the number of instructions that CPU has completed
- Divide **64-bit minstret** into minstreth and minstret



- Count when instruction retired

Slave Configuration

Table 1-5 : Slave configuration

NAME	Number	Start address	End address
ROM	Slave 0	0x0000_0000	0x0000_1FFF
IM	Slave 1	0x0001_0000	0x0001_FFFF
DM	Slave 2	0x0002_0000	0x0002_FFFF
sensor_ctrl	Slave 3	0x1000_0000	0x1000_03FF
DRAM	Slave 4	0x2000_0000	0x201F_FFFF

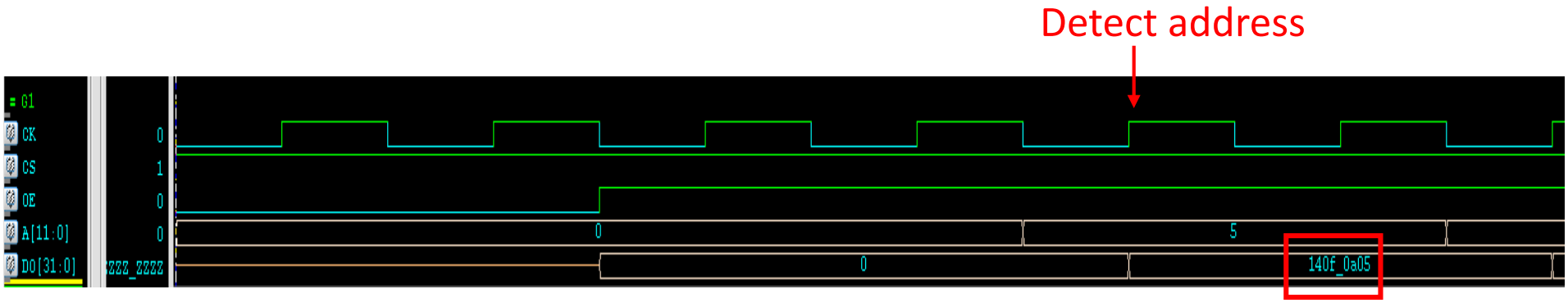
- You should design all slave wrappers !!

ROM (1/2)

ROM	System signals			
	CK	input	1	System clock
	Memory ports			
	DO	output	32	ROM data output
	OE	input	1	Output enable (active high)
	CS	input	1	Chip select (active high)
	A	input	12	ROM address input
	Memory Space			
	Memory_byte0	reg	8	Size: [0:4095]
	Memory_byte1	reg	8	Size: [0:4095]
	Memory_byte2	reg	8	Size: [0:4095]
	Memory_byte3	reg	8	Size: [0:4095]

ROM (2/2)

- ROM_tb.sv (sim/ROM)
  - Read



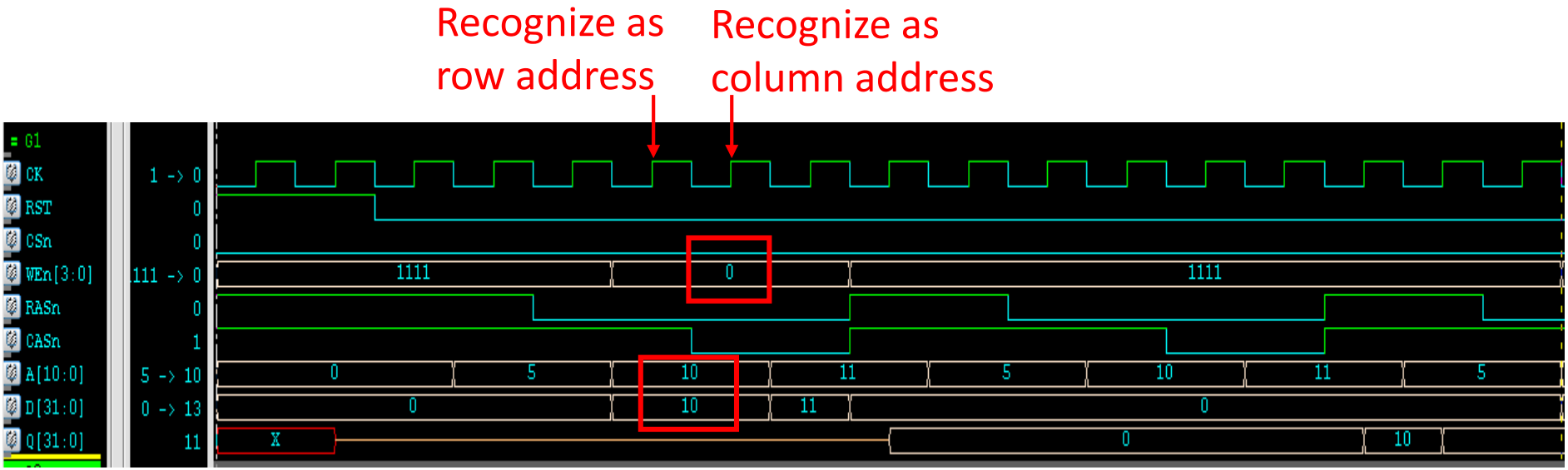
DRAM (1/3)

DRAM	System signals			
	CK	input	1	System clock
	RST	input	1	System reset (active high)
	Memory ports			
	CSn	input	1	DRAM Chip Select (active low)
	WEn	input	4	DRAM Write Enable (active low)
	RASn	input	1	DRAM Row Access Strobe (active low)
	CASn	input	1	DRAM Column Access Strobe (active low)
	A	input	11	DRAM Address input
	D	input	32	DRAM data input
	Q	output	32	DRAM data output
	VALID	output	1	DRAM data output valid
	Memory space			
	Memory_byte0	reg	8	Size: [0:2097151]
	Memory_byte1	reg	8	Size: [0:2097151]
	Memory_byte2	reg	8	Size: [0:2097151]
	Memory_byte3	reg	8	Size: [0:2097151]

★ Row address is 11-bit and column address is 10-bit

DRAM (2/3)

- DRAM_tb.sv (sim/DRAM)
  - Write



Set RAS first, then set CAS

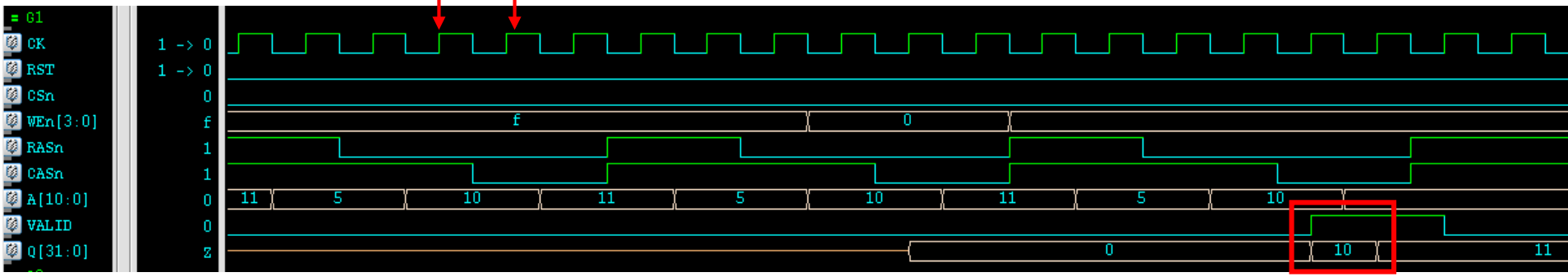
EX: Write data D=10 to DRAM row:10 col:10

DRAM (3/3)

- DRAM_tb.sv (sim/DRAM)
  - Read

Recognize as row address

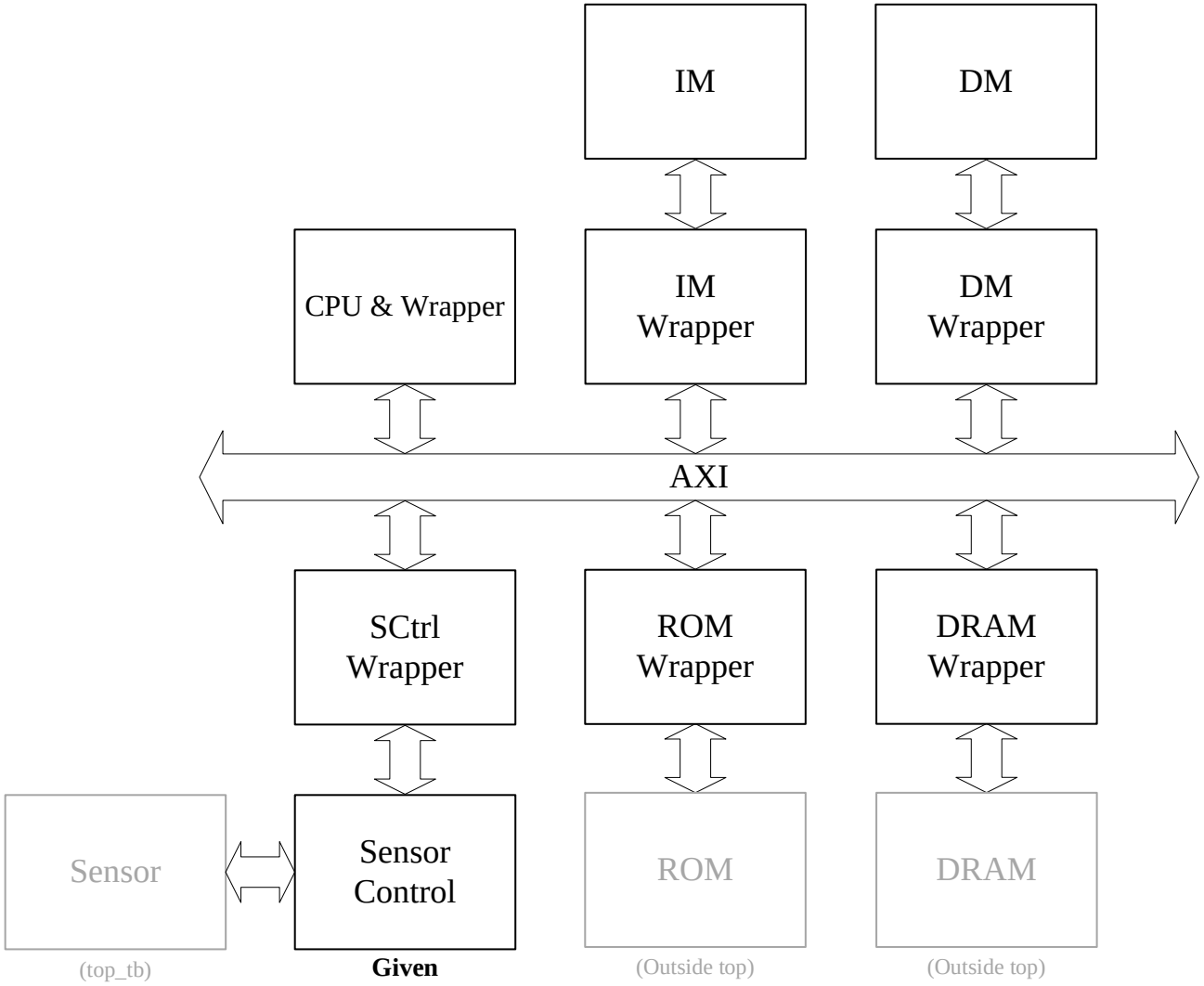
Recognize as column address



VALID keep 1 cycle

EX: Read data at DRAM row:10 col:10

System


IM
DM
CPU & Wrapper
IM Wrapper
DM Wrapper
AXI
SCtrl Wrapper
ROM Wrapper
DRAM Wrapper
Sensor
Sensor Control
ROM
DRAM
(top_tb)
Given
(Outside top)
(Outside top)

Sensor Controller (1/3)

sensor_ctrl	System signals			
	clk	input	1	System clock
	rst	input	1	System reset (active high)
	sctrl_en	input	1	Sensor controller enable (active high)
	sctrl_clear	input	1	Sensor controller clear (active high)
	sctrl_addr	input	6	Sensor controller address
	sctrl_interrupt	output	1	Sensor controller interrupt
	sctrl_out	output	32	Sensor controller data output
	sensor_ready	input	1	Sensor data ready
	sensor_out	input	32	Data from sensor
	sensor_en	output	1	Sensor enable (active high)
	Memory space			
	mem	logic	32	Size: [0:63]

Sensor Controller (2/3)

- Sensor generates a new data every 1024 cycles
- Sensor controller stores data to its local memory
- When local memory is full (64 data), sensor controller will stop requesting data (sensor_en = 0) and assert interrupt (sctrl_interrupt = 1)
- CPU load data from sensor controller and store it to DM
- Write non-zero value in 0x1000_0100 or 0x1000_0200 to enable stcrl_en or stcrl_clear

Address	Mapping
0x1000_0300 – 0x1000_03FF	mem[0] – mem[63]
0x1000_0100	stcrl_en
0x1000_0200	stcrl_clear

Sensor Controller (3/3)

- Any access from address 0x1000_0000 to 0x1000_03FF should be uncacheable, which means that D-cache should pass data between CPU and CPU wrapper without writing it into cache memory.
- Add following code in L1C_data.sv, and use this logic to decide whether to write valid bit, tag array, and data array in L1C_data when read miss.

```
logic cacheable;
always_comb cacheable =  (core_addr[31:16] != 16'h1000);
```

Verification (1/6)

- Prog0: booting + instructions verification
- Prog1: booting + interrupt verification
- You should write a boot program defined asboot.c in prog0 and prog1

Verification (2/6)

- Booting
  - The booting program is stored in ROM
  - Moves data from DRAM to IM and DM

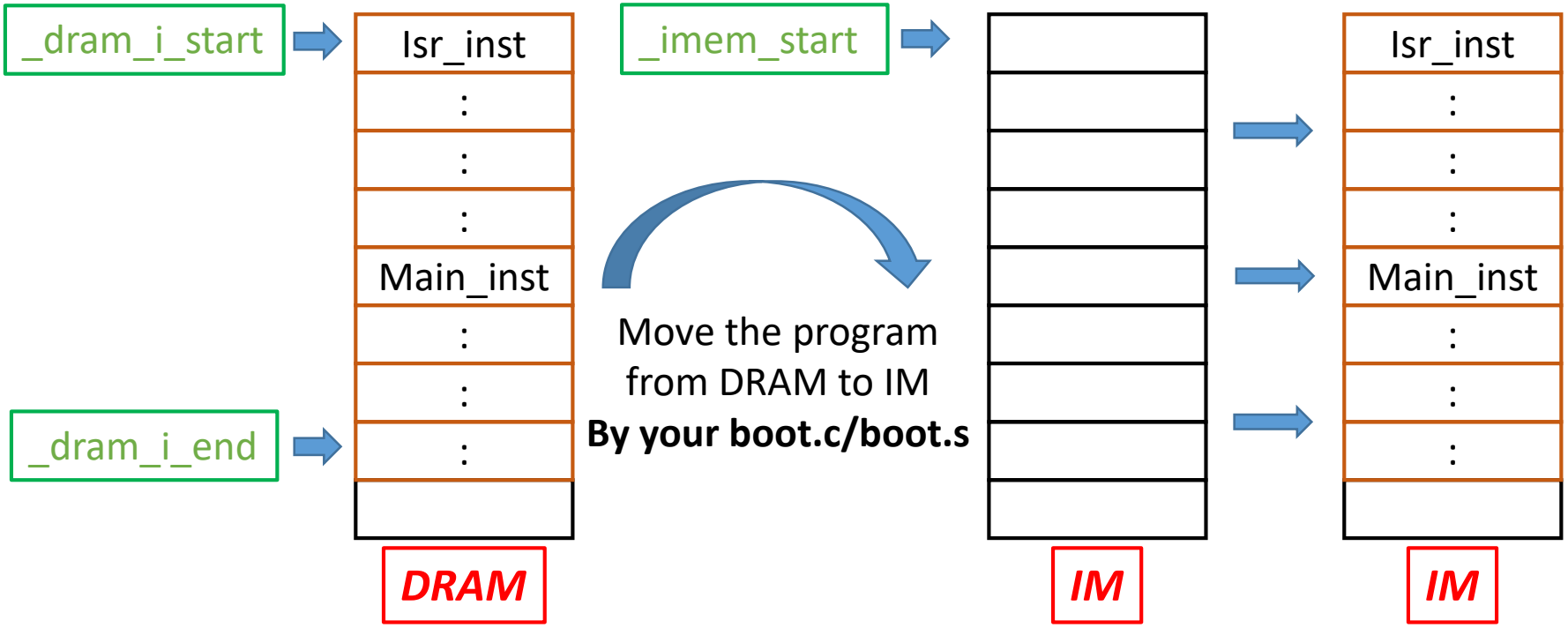
```
extern unsigned int _dram_i_start;
extern unsigned int _dram_i_end;
extern unsigned int _imem_start;

extern unsigned int __sdata_start;
extern unsigned int __sdata_end;
extern unsigned int __sdata_paddr_start;

extern unsigned int __data_start;
extern unsigned int __data_end;
extern unsigned int __data_paddr_start;
```

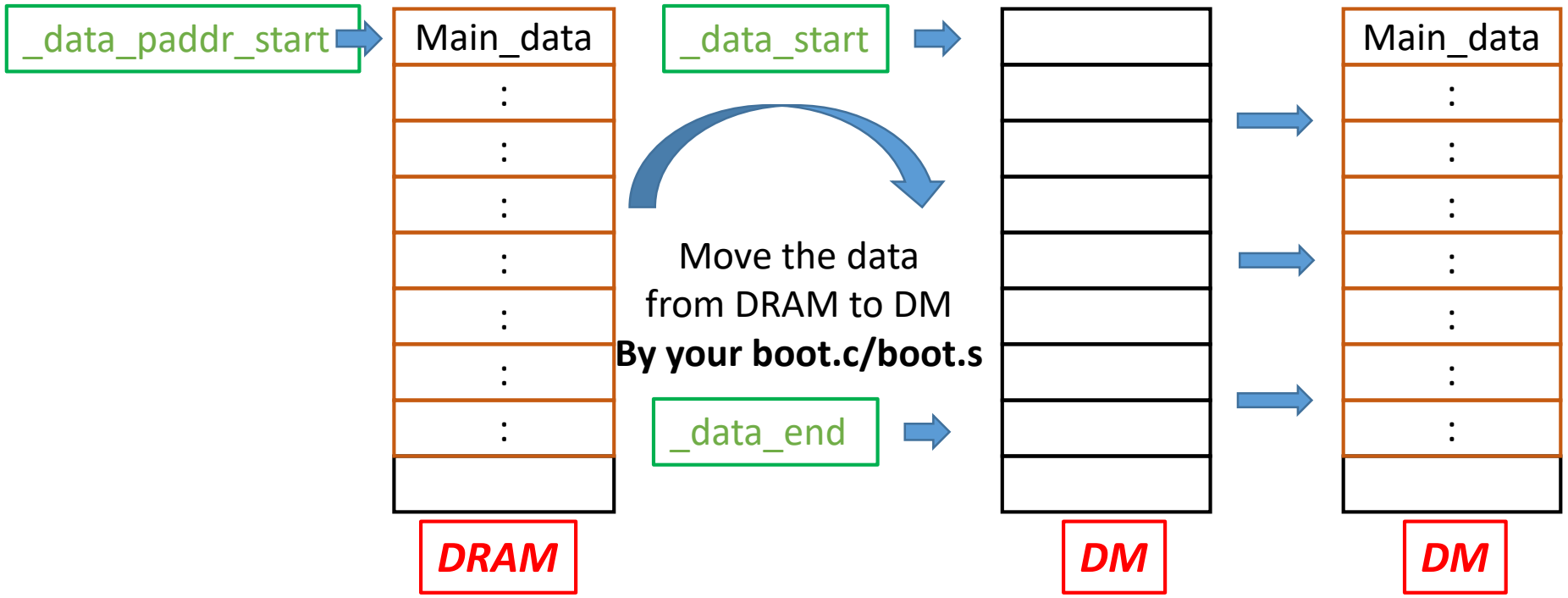
Verification (3/6)

- _dram_i_start = instruction start address in DRAM.
- _dram_i_end = instruction end address in DRAM.
- _imem_start = instruction start address in IM



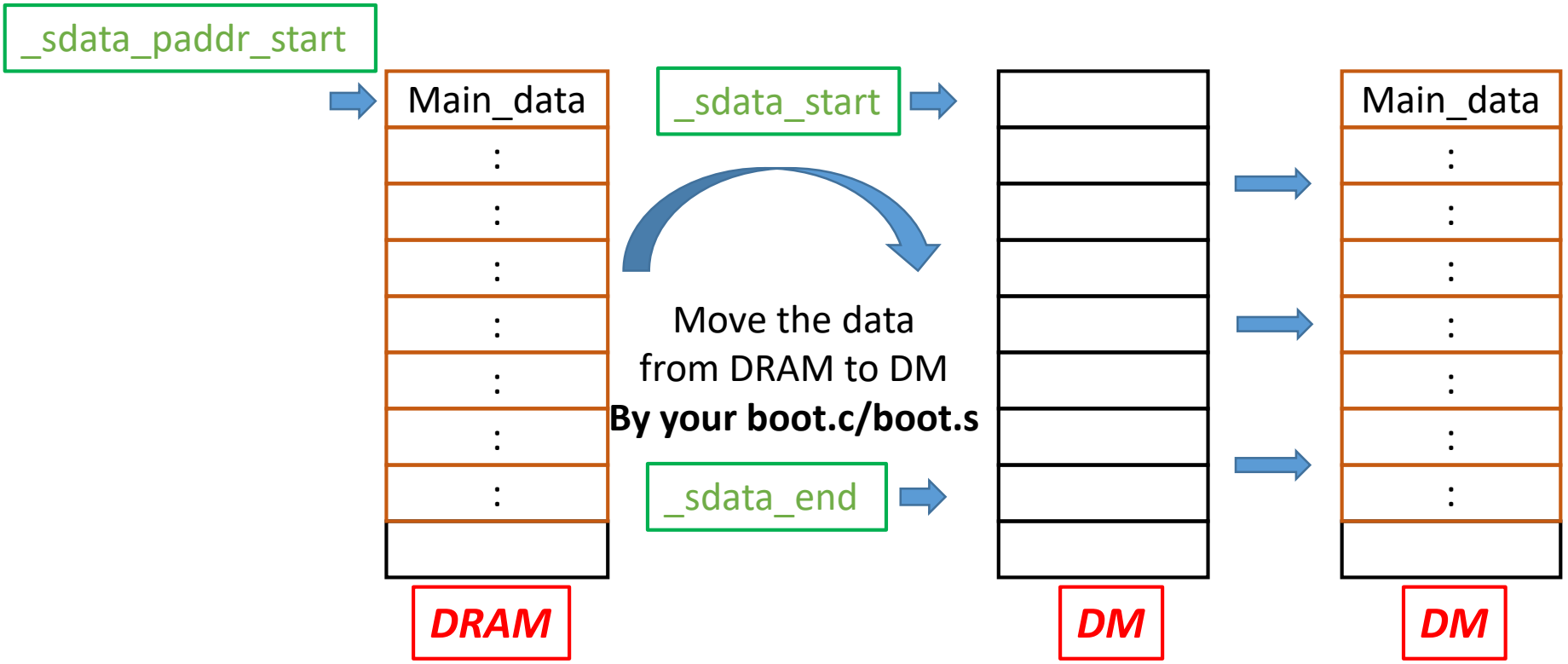
Verification (4/6)

- _data_start = Main_data start address in DM.
- _data_end = Main_data end address in DM.
- _data_paddr_start = Main_data start address in DRAM



Verification (5/6)

- _sdata_start = Main_data start address in DM.
- _sdata_end = Main_data end address in DM.
- _sdata_paddr_start = Main_data start address in DRAM



Verification (6/6)

- Interrupt
  - When sensor controller is full, it will interrupt CPU
  - ISR will be activated and copy data to DM. Then, reset the counter of sensor controller
  - When copy is done, ISR return to main program and sort these data
  - After 4 groups of data are sorted, the simulation will complete

Simulation (1/2)

Table B-1: Simulation commands (Partial)

Simulation Level	Command
Problem1	
RTL	make rtl_all
Post-synthesis (optional)	make syn_all

Table B-2: Makefile macros (Partial)

Situation	Command	Example
RTL simulation for progX	make rtlX	make rtl0
Post-synthesis simulation for progX	make synX	make syn1
Dump waveform (no array)	make {rtlX,synX} FSDB=1	make rtl2 FSDB=1
Dump waveform (with array)	make {rtlX,synX} FSDB=2	make syn3 FSDB=2
Open nWave without file pollution	make nWave	
Open Superlint without file pollution	make superlint	
Open DesignVision without file pollution	make dv	
Synthesize your RTL code (You need write *synthesis.tcl* in *script* folder by yourself)	make synthesize	
Delete built files for simulation, synthesis or verification	make clean	
Check correctness of your file structure	make check	
Compress your homework to *tar* format	make tar	

Simulation (2/2)

Table B-3: Simulation commands

Situation	Command
Run JasperGold VIP on AXI bridge without file pollution (RTL only)	make vip_b

Report

- 請使用附在檔案內的Submission Cover
- 請勿將code貼在.docx內
  - 請將.sv包在壓縮檔內，不可截圖於.docx中
- 需要Summary及Lessons learned
- 不須寫出貢獻度，需標註每位組員的姓名和學號於封面

繳交檔案 (1/2)

- 依照檔案結構壓縮成 “.tar” 格式
  - 在Homework主資料夾(N260XXXXX)使用make tar產生的tar檔即可符合要求
- 檔案結構請依照作業說明
- 請勿附上檔案結構內未要求繳交的檔案
  - 在Homework主資料夾(N260XXXXX)使用make clean即可刪除不必要的檔案
- 請務必確認繳交檔案可以在SoC實驗室的工作站下compile，且功能正常
- 無法compile將直接以0分計算
- 請勿使用generator產生code再修改
- 禁止抄襲

繳交檔案 (2/2)

- 一組只需一個人上傳作業到Moodle
  - 兩人以上都上傳會斟酌扣分
- 壓縮檔、主資料夾名稱、Report名稱、StudentID檔案內的學號都要為上傳者的學號，其他人則在Submission Cover內寫上自己的學號。
  - Ex: A(N26071234)負責上傳，組員為B(N26075678)
  - N26071234.tar (壓縮檔)
    N26071234 (主資料夾)
    N26071234.docx (Report，Cover寫上兩者的學號)

繳交期限

- 2021/1/19 (二) 00:00前上傳
  - 不接受遲交，請務必注意時間
  - Moodle只會留存你最後一次上傳的檔案，檔名只要是「*N260XXXXX.tar*」即可，不需要加上版本號