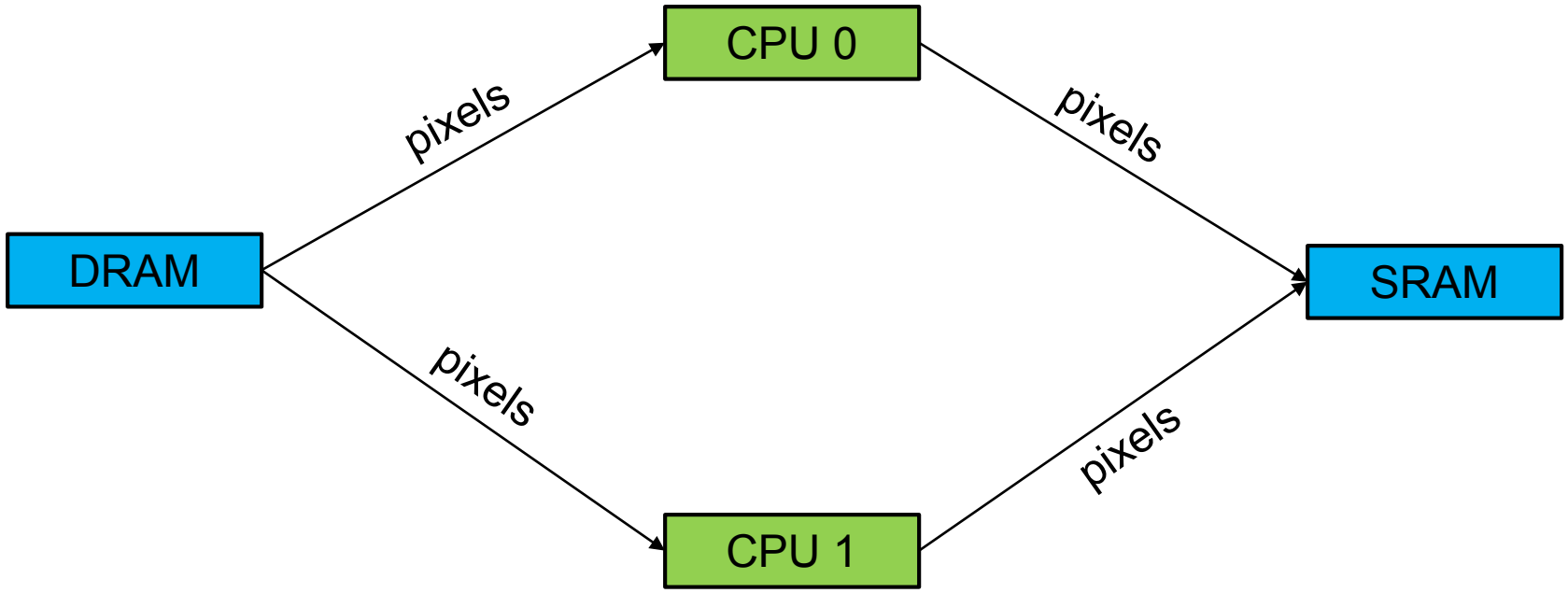
CSCE 313: Embedded Systems

Multiprocessor Systems

Instructor: Jason D. Bakos

UNIVERSITY OF SOUTH CAROLINA

Scaling to Multiple Processors



- DRAM:
  - Peak DRAM bandwidth = 32 bits / 4 bytes per cycle (200 MB/s)
  - Maximum fps = 200 MB/s / 225 KB/frame = 910 fps
- SRAM:
  - Width = 16 bits, so roughly half

Multiprocessor Systems

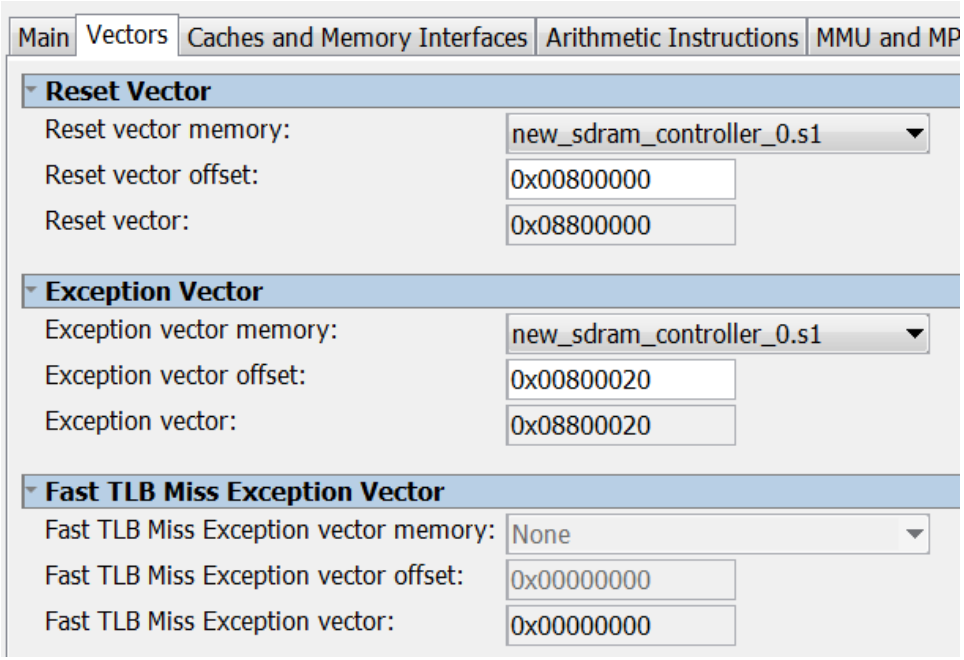
- Platform Designer allows you to add multiple CPUs to your design
- The CPUs can share memories and other system components
  - Connect instruction and data masters to SDRAM controller
  - Connect data master to pixel buffer DMA, SRAM, and performance counter
- Platform Designer also offers hardware components to allow multiple CPUs to synchronize and communicate
- Having multiple CPUs allows you to speed up the system by taking advantage of parallelism

Multiprocessor Systems

- Without an OS, the linker (encoded in the ELF file) determines the memory addresses used for:
  - Code
  - Global variables
  - Stack
  - Heap
- Must make sure code from different processors don't "step on" each other
- Can still use program controlled memory using constant memory addresses:
  - volatile int *shared_memory = 0x<hex address of memory>

Adding Processors

- The reset vector acts as an offset to all the address spans allocated by the linker
- Default:
  - reset vector: 0x0
  - exception vector: 0x20
- Add 4 MB separation between processors:
  - $8 \times 1024 \times 1024_{10} = 800000_{16}$

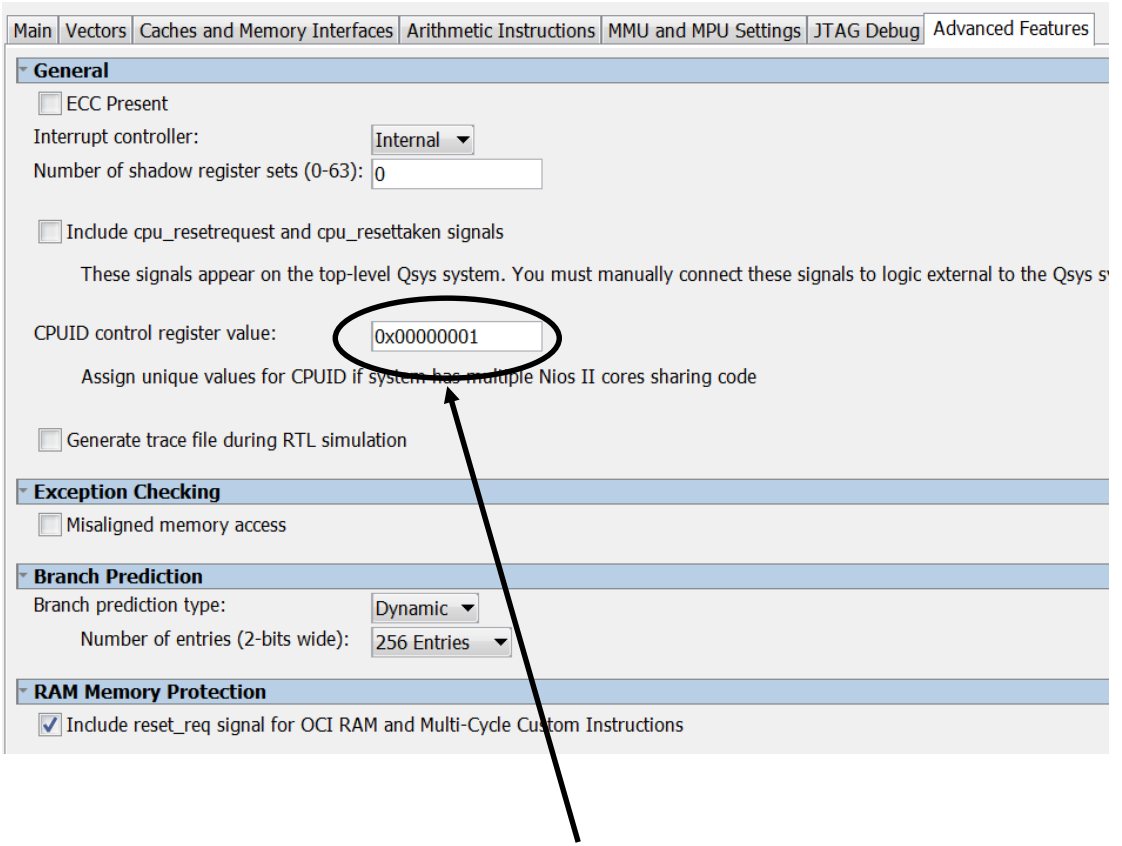


for cpu1->
reset vector 0x800000
exception vector 0x800020

CPUID

- Each processor must be able to self-identify to behave accordingly
- CPUID register provides this functionality:

cpuid=__builtin_rdctl(5);



for cpu1, cpuid=1

Adding Processors

- Create second app and bsp for second processor:

(in software directory)

nios2-swexample-create --name=lights --sopc-file=../nios_system.sopcinfo --type=hello_world --cpu-name=nios2_gen2_1 --app-dir=lights2 --bsp-dir=lights2_bsp

cd lights2

./create_this_app

- Link original hello_world.c (do for all source files):

rm hello_world.c

ln ../lights/hello_world.c

ln ../lights/myfile.h

ln ../lights/myfile.c

nios2-app-update-makefile --add-src-files myfile.c --app-dir .

Running Two Processors

- Start GDB on both:

nios2-gdb-server --tcpport 8000 --tcppersist --instance 0&
nios2-gdb-server --tcpport 8001 --tcppersist --instance 1&

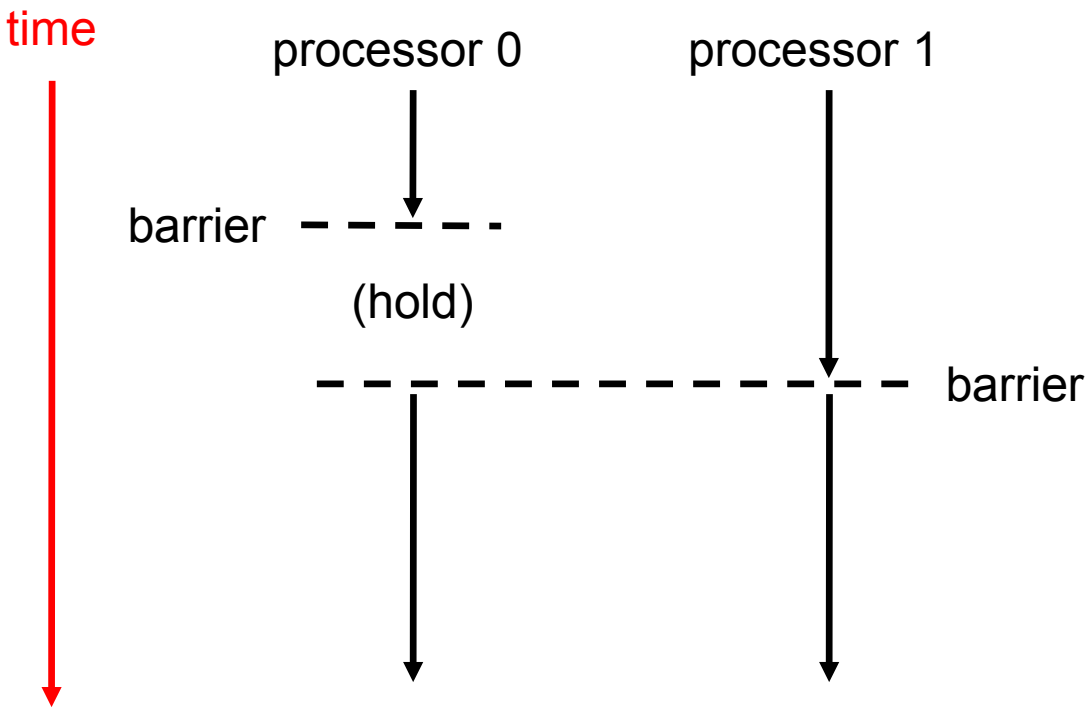
- Debug both (in two terminals):

```
nios2-elf-gdb lights/lights.elf
target remote localhost:8000
load
Loading section .entry, size 0x20 lma
0x8000000
Loading section .text, size 0x11a6c lma
0x8000020
Loading section .rodata, size 0x38798 lma
0x8011a8c
Loading section .rwdata, size 0xb58 lma
0x804ad7c
Start address 0x8000020, load size 306556
Transfer rate: 124 KB/sec, 398 bytes/write.
```

```
nios2-elf-gdb lights2/lights.elf
target remote localhost:8001
load
Loading section .entry, size 0x20 lma
0x8800000
Loading section .text, size 0xdc80 lma
0x8800020
Loading section .rodata, size 0x38708 lma
0x880dca0
Loading section .rwdata, size 0xadc lma
0x8846e84
Start address 0x8800020, load size 290436
Transfer rate: 129 KB/sec, 398 bytes/write.
```

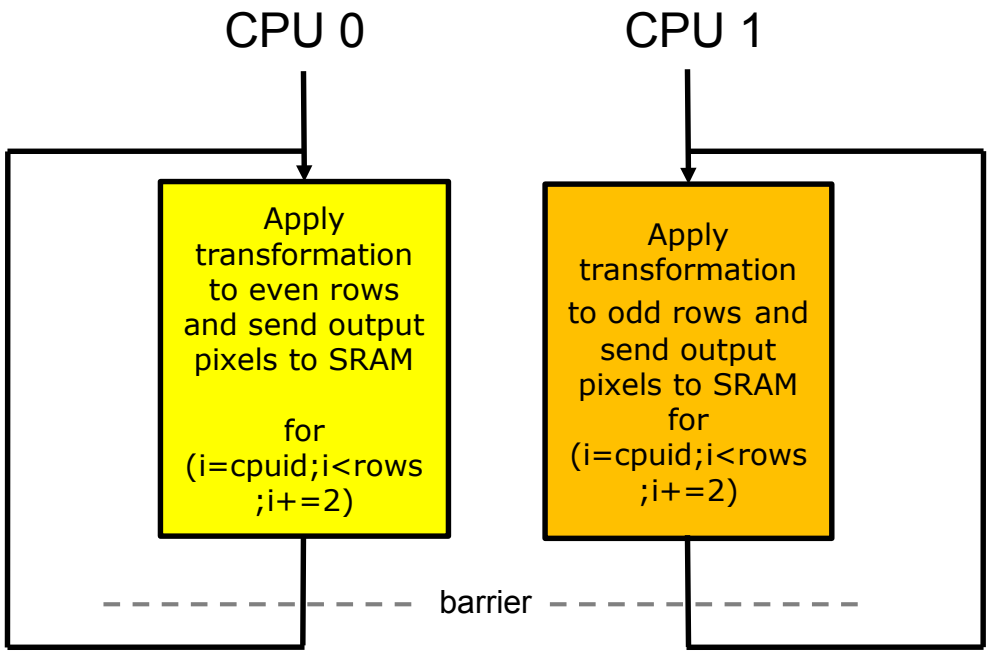
Processor Synchronization

- Make any processor reaching the barrier wait until all processors reach that point
  - Useful when parallelized computations occur in "stages"



Dividing up the Work

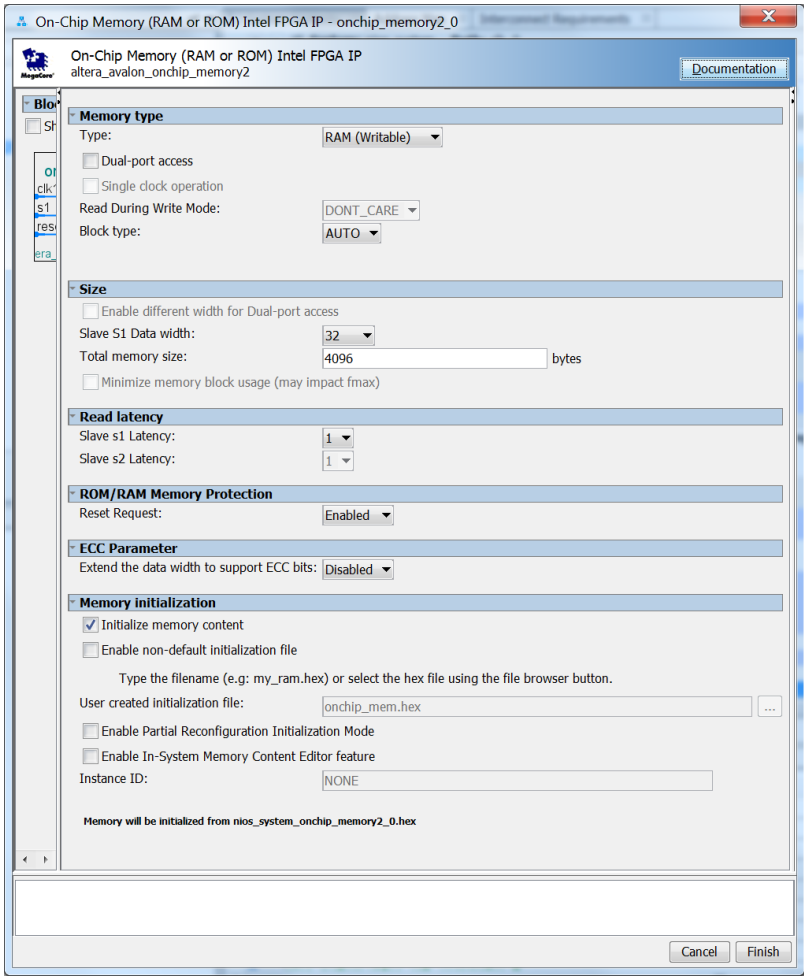
- How do you divide the work amongst multiple independent CPUs?
- In the context of lab 3...



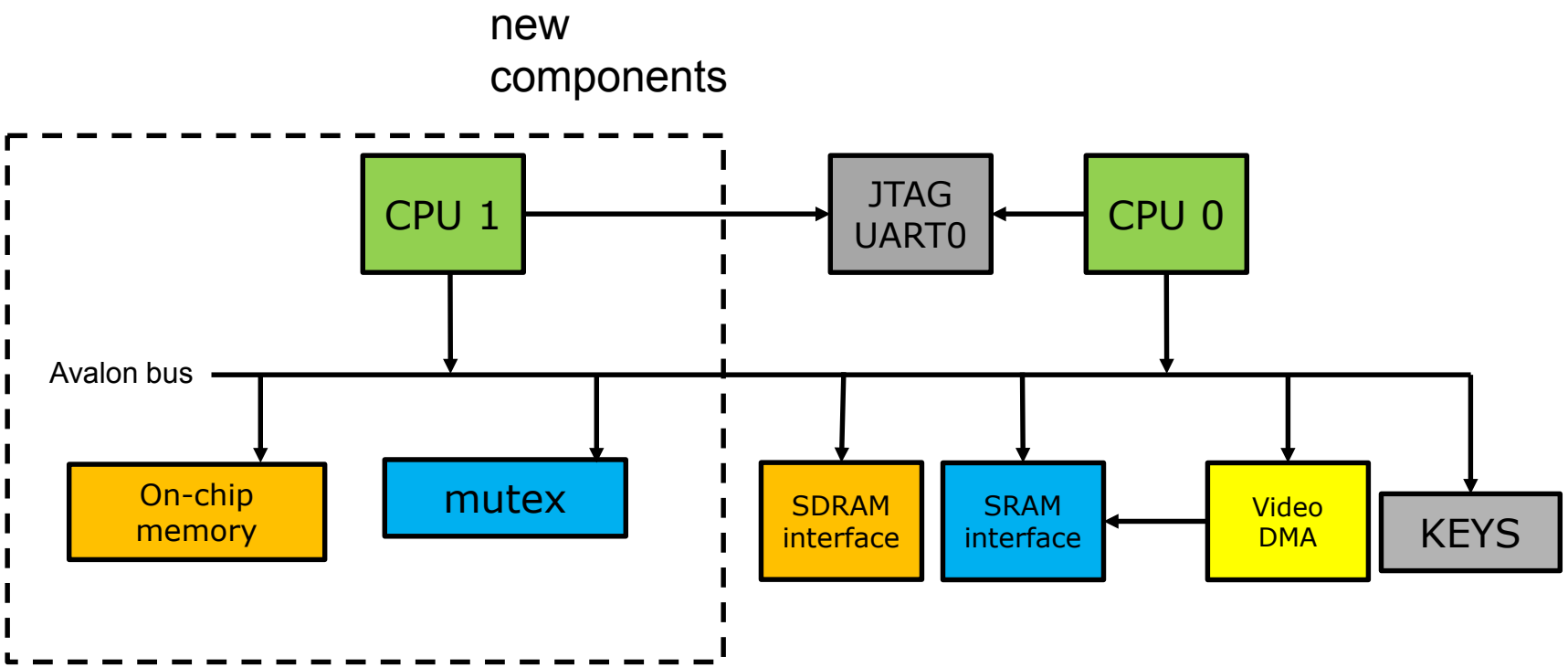
Implementing Barriers

- Instance an on-chip memory
- Connect memory to all CPUs
- Associate a counter for each CPU
- When processor A reaches the barrier, increment all counters except counter A
- Then, wait until counter A reaches count of N-1
- Afterward, reset counter A to 0

- Instance a mutex to ensure that only one CPU can update a counter at any time

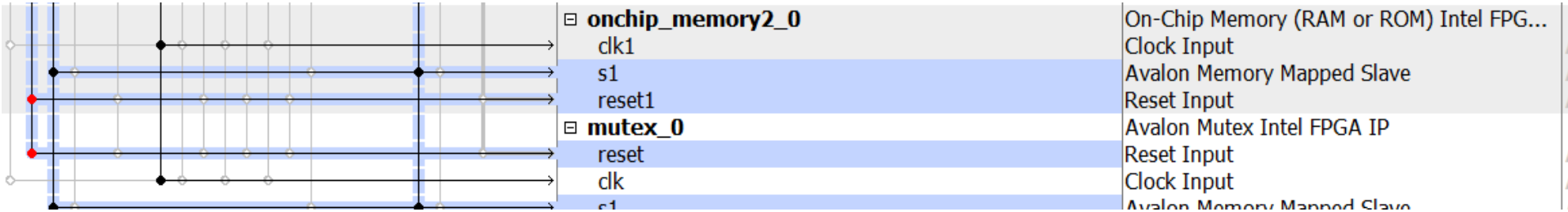


System Design



On-chip Memory and Mutex

- Make sure on-chip memory and mutex is connected to data masters of both CPUs



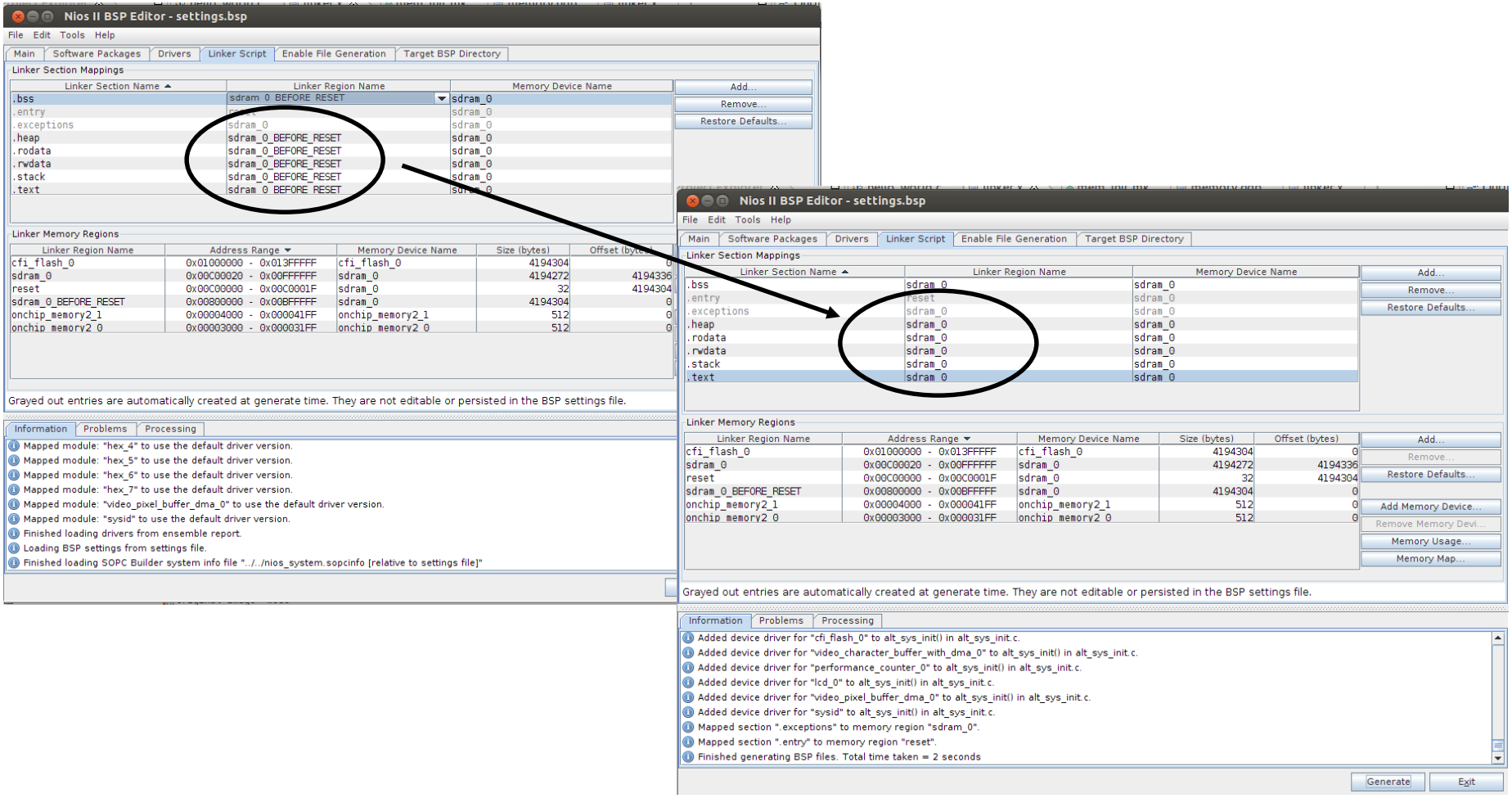
- Use intrinsics to bypass cache when loading and storing from shared memories:
  - IORD_32DIRECT(address,offset)
  - IOWR_32DIRECT(address,offset,data)

- Code modifications:
  - Include files:
    - #include <system.h>
    - #include <altera_avalon_mutex.h>
    - #include <io.h>

Barrier Code

```
void barrier () {
    volatile alt_u32 *counters = ONCHIP_MEMORY2_0_BASE;
    int i,count;
    alt_mutex_dev *my_mutex;
    alt_u32 cpuid;

    cpuid = __builtin_rdctl(5);

    my_mutex =
altera_avalon_mutex_open("/dev/mutex_0");
    if (!my_mutex) {
        perror("error opening mutex");
        return 0;
    }

    for (i=0;i<NUM_CPUS;i++) {
        if (i!=cpuid) {
                altera_avalon_mutex_lock(my_mutex, 1);
                count = IORD_32DIRECT(&counters[i],0);
                count++;
                IOWR_32DIRECT(&counters[i],0,count);
                altera_avalon_mutex_unlock(my_mutex);
        }
    }
    do {
        count = IORD_32DIRECT(&counters[cpuid],0);
    } while (count != NUM_CPUS-1);

    altera_avalon_mutex_lock(my_mutex, 1);
    IOWR_32DIRECT(&counters[cpuid],0,0);
    altera_avalon_mutex_unlock(my_mutex);

    altera_avalon_mutex_close(my_mutex);
}
```

BSP Editor Linker Script



Notes

- Only clear screen with one processor, e.g.

```
if (cpuid == 0)
alt_up_pixel_buffer_dma_clear_screen(my_pixel_buffer,0);
```

- Match cache sizes on processors
- Instance second floating-point instructions for second CPU
- Number of instructions per pixel should remain the same as Lab 3
  - CPI will change
  - Effective cycles per pixel will change