Global Common Memory Allocation
Data Access to/from Global Common Memory

Hardware/Software Co-Design

Global Common Memory

Miaoqing Huang
University of Arkansas
Fall 2011
Outline

1. Global Common Memory Allocation
2. Data Access to/from Global Common Memory
Outline

1. Global Common Memory Allocation

2. Data Access to/from Global Common Memory
Global Common Memory

- Global Common Memory
  - 1 GB

Controller
- Altera Stratix II
  - EP2S130

Global Common Memory
- 1 GB

User Logic 1
- Altera Stratix II
  - EP2S180

User Logic 2
- Altera Stratix II
  - EP2S180

- 16 Banks of On-Board Memory (64 MB)

- 7.2 GB/s
- 7.2 GB/s

- 4.2 GB/s
- 4.2 GB/s

- 4.8 GB/s
- 256b

- 12.8 GB/s

- 19.2 GB/s

- 12.8 GB/s
Allocate Memory from GCM

```c
#include <libmap.h>
gcm_addr_t my_buffer;
my_buffer = gcm_allocate(256*sizeof(uint64_t));
```

- **Success**: return 64-bit address; Fail: return 0
- **size** must be a multiple of 8

```
#include <libmap.h>
gcm_addr_t my_in_buffer, my_out_buffer;
my_in_buffer = gcm_allocate_by_bank(256*sizeof(uint64_t), 1);
my_out_buffer = gcm_allocate_by_bank(256*sizeof(uint64_t), 2);
```

- Allocate memories from distinct banks to improve parallelism
**Explicit Segment Allocation**

```c
int gcm_allocate_seg (uint64_t size, gcm_seg_desc_t ** seg_desc); 

int gcm_allocate_seg_by_bank (uint64_t size, int vbank,  
                             gcm_seg_desc_t ** seg_desc); 

gcm_addr_t gcm_allocate_from_seg (uint64_t size,  
                                  gcm_seg_desc_t ** seg_desc); 

#include <libmap.h>
gcm_seg_desc_t *my_in_seg, *my_out_seg; 
int res; 
uint64_t my_seg_size = 10*1024*1024;  // 10 MB segment size 
gcm_addr_t my_in_buffer, my_out_buffer; 

// non-zero return: failure 
if (res = gcm_allocate_seg_by_bank (my_seg_size, 1, &my_in_seg)) {  
    printf("Allocate of GCM segment failed. Code=%d\n", res);  
}

my_in_buffer = gcm_allocate_from_seg(256*\sizeof(uint64_t),my_in_seg); 
my_out_buffer = gcm_allocate_from_seg(256*\sizeof(uint64_t),my_out_seg); 
```
Free GCM

```c
int gcm_free (gcm_addr_t buffer);

int gcm_free_seg (gcm_seg_desc_t *seg_desc);

GCM_Free_All ();
```
Outline

1. Global Common Memory Allocation
2. Data Access to/from Global Common Memory
From CPU Side

_SRC Hi-Bar Switch_

```
gcm_cp_to
```

- **SNAP**
- **Memory**
- **μP**

- **Common Memory Bank 1**
- **Common Memory Bank 2**
void gcm_cp_to (gcm_addr_t gcm_addr, void *from, uint64_t nbytes);

- nbytes must be a multiple of 8
- gcm_addr and from must be aligned on 8-byte boundaries

#include <libmap.h>
#define MB (1024*1024)
gcm_addr_t my_gcm_buffer;
uint64_t *my_micro_buffer;
int i;

my_gcm_buffer = gcm_allocate(MB);
my_micro_buffer = (uint64_t *) Cache_Aligned_Allocate (MB);
for (i=0; i<MB/sizeof(uint64_t); i++) {
    *my_micro_buffer = (uint64_t) i;
}
gcm_cp_to (my_gcm_buffer, my_micro_buffer, MB);
Global Common Memory Allocation

Data Access to/from Global Common Memory

**gcm_cp_from**

```c
void gcm_cp_from (gcm_addr_t gcm_addr, void *to, uint64_t nbytes);
```

- **nbytes** must be a multiple of 8
- **gcm_addr** and **to** must be aligned on 8-byte boundaries

```c
#include <libmap.h>
#define MB (1024*1024)
gcm_addr_t my_gcm_data_buffer, my_gcm_result_buffer;
uint64_t *my_data_buffer, *my_result_buffer;
int i, count;

my_gcm_data_buffer = gcm_allocate (MB);
my_gcm_result_buffer = gcm_allocate (MB);
my_data_buffer = (uint64_t *) Cache_Aligned_Allocate (MB);
my_result_buffer = (uint64_t *) Cache_Aligned_Allocate (MB);
count = MB/sizeof(uint64_t);
for (i=0; i<count; i++) {
    *my_data_buffer = (uint64_t) i;
}
gcm_cp_to (my_gcm_data_buffer, my_data_buffer, MB);
call my_calc (my_gcm_data_buffer, my_gcm_result_buffer, count, mapnum);
gcm_cp_from (my_gcm_result_buffer, my_result_buffer, MB);
```
Global Common Memory Allocation
Data Access to/from Global Common Memory

From MAP Processor Side

SRC Hi-Bar Switch

dma_GCM (GCM2OBM)

SNAP

Memory

\(\mu P\)

Common Memory Bank 1

On-Board Memory

MAP
Global Common Memory Allocation
Data Access to/from Global Common Memory

buffered_dma_gcm

```c
void buffered_dma_gcm (int direction, int dma_path,
    void *obmaddr, int64_t obmstride,
    gcm_addr_t gcm_addr, int64_t gcmstride, uint64_t nbytes);
```

- **direction**: OBM2GCM or GCM2OBM
- **dma_path**: PATH_0 or PATH_1
- **obmaddr**: OBM address of the start of the transfer
- **obmstride**: OBM stride/stripe constant
- **gcm_addr**: starting GCM address
- **gcmstride**: stride through GCM
- **nbytes**: number of bytes to be copied, has to be a multiple of 8

```c
buffered_dma_gcm (GCM2OBM, PATH_0, al, MAP_OBM_stripe(1, "A"),
               source, 1, nbytes);
```