« Real Time Embedded systems »
Multi Masters Systems

rene.beuchat@epfl.ch
LAP/ISIM/IC/EPFL
Chargé de cours
rene.beuchat@hesge.ch
LSN/hepia
Prof. HES
Multi Master on Chip

- On a System On Chip, Master can be:
  - A Processor
  - A DMA Master module
  - An accelerator composed of 1-2 DMA units

- Masters can communicate by:
  - Shared common memory
  - Network On Chip
  - Specialized interfaces with local memory as hardware mailbox
Simple Multi processors system

- Independents processors
- Each processor has its own local memory and programmable interface
- NO communication between processors
Multi Masters architecture

• Each processor has it's own local memory and/or cache memory and/or programmable interfaces
• A global memory is shared by all processor
• Mutual exclusion primitives are necessary for access to common resources
Multi Masters architecture

- Local bus
  - Processor 1
  - Local Memory 1
  - Local Interfaces 1
  - Processor 2
  - Local Memory 2
  - Local Interfaces 2

- Global bus
  - Global Memory
  - Global Interfaces

- Local Memory
  - Processor n
  - Local Memory n
  - Local Interfaces n

- Global Memory

- Global Interfaces
Multi Masters architecture

- Example with 2 processors:
  - Processor 1:
    - Memory 1
    - UART 1
    - Timer 1
  - Processor 2:
    - Memory 2
    - Timer 2
    - Shared Memory
Multiprocessors

- Symmetrical Multi Processors (SMP) are widely used in workstation and high end PC. They are quite complexes. Tasks are send to a free processor.

- On an embedded system, especially FPGA bases embedded systems, **asymmetric** multiprocessor systems are common now. The hardware development is not too difficult.
Multiprocessors

- Example of asymmetrical multiprocessor on Avalon system
Multi Masters, Avalon interconnection

The arbiter is at the Slave level on the Avalon Bus
Multi Master, Slave Arbiter view
Multi master round robin fairness arbitration

- With SOPC, the number of authorized consecutive cycles for one master can be specified,
- ex: Master 1: 3, Master 2: 4
- If a master don’t use all of its shares, it will wait until its next slot
Multi master round robin fairness arbitration

- SOPC: arbitration and bus selection
- For each slave, the number of shares between master is configurable
- View

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Description</th>
<th>Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu</td>
<td>Nios II Processor - Altera Design</td>
<td>clk</td>
</tr>
<tr>
<td>instruction_master</td>
<td>Master port</td>
<td></td>
</tr>
<tr>
<td>data_master</td>
<td>Master port</td>
<td></td>
</tr>
<tr>
<td>jtag_debug_module</td>
<td>Slave port</td>
<td></td>
</tr>
<tr>
<td>sys_clk_timer</td>
<td>Interval timer</td>
<td>clk</td>
</tr>
<tr>
<td>ext_ram_bus</td>
<td>Avalon Tri-State Bridge</td>
<td>clk</td>
</tr>
<tr>
<td>ext_flash</td>
<td>Flash Memory (Common)</td>
<td></td>
</tr>
<tr>
<td>ext_ram</td>
<td>IDT71V416 SRAM</td>
<td></td>
</tr>
<tr>
<td>epcs_controller</td>
<td>EPCS Serial Flash Controller</td>
<td>clk</td>
</tr>
<tr>
<td>lan91c111</td>
<td>LAN91c111 Interface (Common)</td>
<td></td>
</tr>
<tr>
<td>jtag_uart</td>
<td>JTAG UART</td>
<td>clk</td>
</tr>
</tbody>
</table>
Multiprocessors, resource sharing

- Resources sharing with mutual exclusion have to be realized
- A specific hardware interface **Mutex Core** is available on Altera Avalon based systems, the mutex core has the following basic behavior:
  - There are multiple processors accessing a single mutex core (it's a Global Interface)
  - Each processor has a unique identifier (ID)
Multiprocessors exclusion primitives

• To access a common resource mutual exclusion primitive is needed

• A indivisible Read-Modify-Write cycle is necessary

• Depending on the architecture this function can be implemented at:
  - Bus level with a lock bus primitive
  - Peripheral level with mutex programmable interface primitive
  - Message passing synchronization
  - …similar functionality
Multiprocessors exclusion primitives

- The principle is the same as semaphore primitive in multi-thread on a mono processor system. In this case uninterruptible access to a semaphore (single bit or counter) is needed.
- In multi processor system locked read-modify-write of a semaphore is needed.
- Bus level with a lock bus primitive
  - Possible if the bus provides this primitive
  - The master needs to be able to use this primitive
Multiprocessors (mutex)

- Peripheral level with mutex primitive:
  - A special register has 2 fields:
    - Processor ID and Value
  - When value is 0, the mutex is unlocked (free)
  - Otherwise, the mutex is locked (busy)
  - The mutex register can always be read by any processor
  - When the mutex is locked, the Processor ID contain the value of the locking processor
Multiprocessors (mutex)

• A write operation changes the mutex register only if one or both of the following conditions is true:
  
  ➢ The VALUE field of the mutex register is zero.
  ➢ The OWNER field of the mutex register matches the OWNER field in the data to be written.
Multiprocessors (mutex)

- A processor attempts to acquire the mutex by writing its ID to the OWNER field, and writing a non-zero value to VALUE.
- The processor then checks if the acquisition succeeded by verifying the OWNER field.
Multiprocessors (mutex)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Name</th>
<th>R/W</th>
<th>Bit Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>mutex</td>
<td>RW</td>
<td>OWNER</td>
</tr>
<tr>
<td>1</td>
<td>reset</td>
<td>RW</td>
<td>VALUE</td>
</tr>
</tbody>
</table>

- Mutex register,
  - Owner: CPU ID
  - Value: can be modified by CPU owner or every processor if Owner = 0
- Reset = 1 at reset and must be cleared
- An Owner can be initialized at FPGA compile time
## Multiprocessors (mutex)

- **Avalon Mutex functions**

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>altera_avalon_mutex_open()</code></td>
<td>Claims a handle to a mutex, enabling all the other functions to access the mutex core.</td>
</tr>
<tr>
<td><code>altera_avalon_mutex_trylock()</code></td>
<td>Tries to lock the mutex. Returns immediately if it fails to lock the mutex.</td>
</tr>
<tr>
<td><code>altera_avalon_mutex_lock()</code></td>
<td>Locks the mutex. Will not return until it has successfully claimed the mutex.</td>
</tr>
<tr>
<td><code>altera_avalon_mutex_unlock()</code></td>
<td>Unlocks the mutex.</td>
</tr>
<tr>
<td><code>altera_avalon_mutex_is_mine()</code></td>
<td>Determines if this CPU owns the mutex.</td>
</tr>
<tr>
<td><code>altera_avalon_mutex_first_lock()</code></td>
<td>Tests whether the mutex has been released since reset.</td>
</tr>
</tbody>
</table>
Example: Opening and locking a mutex

```
#include <altera_avalon_mutex.h>

/* get the mutex device handle */
alt_mutex_dev* mutex = altera_avalon_mutex_open( "/dev/mutex" );

/* acquire the mutex, setting the value to one */
altera_avalon_mutex_lock( mutex, 1 );

/* Access a shared resource here. */

/* release the lock */
altera_avalon_mutex_unlock( mutex );
```
Multiprocessors (Mailbox)

• Multiprocessor environments can use the **mailbox core** with Avalon® interface (the mailbox core) to send messages between processors.

• The mailbox core contains mutexes to ensure that only one processor modifies the mailbox contents at a time. The mailbox core must be used in conjunction with a **separate shared memory** which is used for storing the actual messages.

• The mailbox core is designed for use in Avalon-based processor systems, such as a Nios® II processor system.

• The mailbox core is SOPC Builder-ready and integrates easily into any SOPC Builder-generated system.
Multiprocessors (Mailbox) deprecated

- Mailbox functionality using the mutexes and memory is implemented entirely in software.
- 2 Mutexes for the mailbox interface:
  - 1 mutex for write access to the software mailbox

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Name</th>
<th>R/W</th>
<th>Bit Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>mutex0</td>
<td>RW</td>
<td>OWNER</td>
</tr>
<tr>
<td>1</td>
<td>reset0</td>
<td>RW</td>
<td>–</td>
</tr>
<tr>
<td>2</td>
<td>mutex1</td>
<td>RW</td>
<td>OWNER</td>
</tr>
<tr>
<td>3</td>
<td>reset1</td>
<td>RW</td>
<td>–</td>
</tr>
</tbody>
</table>

*VALUE*
Multiprocessors (Mailbox) deprecated

• A complete mailbox for message passing need:
  - The mailbox interface
  - A shared memory

• Both of them need to be available for all of the processors in the systems who need to communicate together
Multiprocessors (Mailbox) **deprecated**

- Some part of function are available to manage and use the mailbox

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>altera_avalon_mailbox_close()</code></td>
<td>Closes the handle to a mailbox.</td>
</tr>
<tr>
<td><code>altera_avalon_mailbox_get()</code></td>
<td>Returns a message if one is present, but does not block waiting for a message.</td>
</tr>
<tr>
<td><code>altera_avalon_mailbox_open()</code></td>
<td>Claims a handle to a mailbox, enabling all the other functions to access the mailbox core.</td>
</tr>
<tr>
<td><code>altera_avalon_mailbox_pend()</code></td>
<td>Blocks waiting for a message to be in the mailbox.</td>
</tr>
<tr>
<td><code>altera_avalon_mailbox_post()</code></td>
<td>Posts a message to the mailbox.</td>
</tr>
</tbody>
</table>
Example: Writing to and reading from a mailbox

```c
#include <stdio.h>
#include "altera_avalon_mailbox.h"

int main()
{
    alt_u32 message = 0;
    alt_mailbox_dev* send_dev, recv_dev;
    /* Open the two mailboxes between this processor and another */
    send_dev = altera_avalon_mailbox_open("/dev/mailbox_0");
    recv_dev = altera_avalon_mailbox_open("/dev/mailbox_1");

    while(1)
    {
        /* Send a message to the other processor */
        altera_avalon_mailbox_post(send_dev, message);

        /* Wait for the other processor to send a message back */
        message = altera_avalon_mailbox_pend(recv_dev);
    }

    return 0;
}
```

Message could be a pointer to a shared memory
Multiprocessor cache memories

• In multiprocessor with data cache memories a big problem is cache coherency for shared common memories
• If a write access is done by a master, bus snooping by other caches need to invalidate them if necessary
• Or shared memory access could by-pass the cache memory (i.e. bit 31=1 on Avalon bus)
• Or flush of local data cache needs to be done before shared memory access
Multi Masters architecture

Local memory for each processor
Global memory for shared data/instr

Processor 1
Local Memory 1
Local Interfaces 1
Processor 2
Local Memory 2
Local Interfaces 2

Global bus

Processor n
Local Memory n
Local Interfaces n

Global shared Memory

Local bus
“Local memories” can be in separate area on the global physical memory
Processors could have local cache memories (instr/data)
Multiprocessor instruction memories

• Instruction cache memories need to be **flushed** after a new modification of the code (code loading).

• The problem is to be able to signal that fact to the corresponding processor! i.e. by interruption.
Multiprocessors on Avalon, memory mapping

• Each processor can have a reserved local access to the main common memory.
• NIOS IDE use the Exception address specified in SOPC from the NIOS description

| Local Mem1 | Local Mem2 | Local Mem3 | Shared Mem |
Memory Map on a typical processor (ex. NIOSII)

- Areas used by the linker:
  - `.stack`— where function-call parameters and other temporary data is stored
  - `.heap`— where dynamically allocated memory is located
  - `.rwdata`— where read-write variables and pointers are stored
  - `.rodata`— any read-only data used in the execution of the code
  - `.text`— the actual executable code
• The memory used for each area is specified in NIOS IDE through System Library Settings
Mapping for two processors

- Memory Map Specification for 2

[Diagram showing memory mapping for two processors with different sections for each processor.]
Memory configuration rules

- Be careful as NO barrier are available between the two memory maps
- Left 0x20 between Reset and Exception addresses: it corresponds to 1 instruction cache line
- The end of a processor area is the starting address of the next area
- For common memory, allocate it on one processor area and pass the pointer by message passing or use an separate internal SRAM available to all processors
Debugging with NIOS IDE

• With NIOS IDE, multiprocessors can be debugged as n single processors through processor ID specification for each NIOS program.

• A *collection* can be specified to group n processor and run their program automatically, !! They are NOT synchronized to start at the same time, it’s the user responsibility to do that.
Multi master DMA accelerator

• An accelerator is based on a DMA unit:
• Direct Memory Access unit capable of reading/Writing in memory as an Avalon Master (altera sopc system)
NIOS II Processor, hardware accelerator

- To assist the processor efficiency in some tasks, a specialized **accelerator** can be provide.
- It has the capacity of reading and writing in memory by itself.
- It can make (complex) operations between reading and writing,
- The processor has to initialize the source/destination addresses, the length and some parameters depending on the accelerator
NIOS II Processor, hardware accelerator
NIOS II Processor, hardware accelerator, examples (Stratix II, from Altera)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Speed Increase (vs. Nios II CPU)</th>
<th>System $f_{MAX}$ (MHz)</th>
<th>System Resource Increase (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autocorrelation</td>
<td>41.0x</td>
<td>115</td>
<td>124%</td>
</tr>
<tr>
<td>Bit Allocation</td>
<td>42.3x</td>
<td>110</td>
<td>152%</td>
</tr>
<tr>
<td>Convolution Encoder</td>
<td>13.3x</td>
<td>95</td>
<td>133%</td>
</tr>
<tr>
<td>Fast Fourier Transform (FFT)</td>
<td>15.0x</td>
<td>85</td>
<td>208%</td>
</tr>
<tr>
<td>High Pass Filter</td>
<td>42.9x</td>
<td>110</td>
<td>181%</td>
</tr>
<tr>
<td>Matrix Rotate</td>
<td>73.6x</td>
<td>95</td>
<td>106%</td>
</tr>
<tr>
<td>RGB to CMYK</td>
<td>41.5x</td>
<td>120</td>
<td>84%</td>
</tr>
<tr>
<td>RGB to YIQ</td>
<td>39.9x</td>
<td>110</td>
<td>158%</td>
</tr>
</tbody>
</table>
NIOS II Processor, hardware accelerator

- The accelerator can be design in HDL language as **VHDL** or **Verilog**
- It can be extracted directly from **C code** and translated in HDL language with initializing and API functions in C automatically
- It was the **C2H** compiler tool (**deprecated**)
NIOS II Processor, hardware accelerator

- More “universal” language OpenCL C code:
  - Kernel modules translated:
    - for multiprocessor
    - for GPU
    - for FPGA
NIOS II Processor, hardware accelerator design flow
Profiling

• To determine which part of code is useful to accelerate, code profiling is the best tool.

• Code with:
  - Loop
  - specific algorithm between source and destination
  - a large amount of data

• is a good candidate to accelerate
Minimal source code changes are required to take measurements for analysis with the GNU profiler. The only changes needed are as follows:

1. Add the profiler library via a checkbox in the Nios II IDE.
2. Change the main() function to call exit().
3. Rebuild the project.
The major advantage to measuring with the profiler is that it provides an **overview of the entire system**. Although there is some overhead, it is distributed evenly throughout the system.

The functions that are identified as consuming the most CPU time will still consume the most CPU time when run at full speed without profiler instrumentation.
Profiling, GNU Profiler Drawback

- Adding instructions to each function call for use with the profiler **affects the code’s behavior.**
- Each function is slightly larger. Each function calls another function to collect profiling information.
SOC + FPGA (ex. Cyclone V)
SOC + FPGA (ex.CycloneV)

SCU: Snoop Control Unit
ACP: Accelerator Coherency Port
SOC + FPGA (ex. Cyclone V)
SOC + FPGA (ex. Zynq Xilinx)
Ultra-Scale Psoc Architecture (ex: Zynq UltraScale+ EG)

- quad-core ARM® Cortex A53 up to 1.5GHz.
- dual-core Cortex-R5 real-time processors,
- Mali-400 MP2 graphics processing unit,
- and 16nm FinFET+ programmable logic

https://www.xilinx.com/content/dam/xilinx/docs/products/zynq/zynq-eg-block.png
Multiprocessor, ex. T4240 Freescale QorIQ serie