Lecture 5: Memory and Peripheral Busses
September 16, 2014
Announcements

- Homework #2
- Where was I last week?
  - VLCS’14
  - MobiCom’14
  - HotWireless’14
Emerging Retail Environment: A Walled Garden

- Often have line-of-sight to lighting
  - Groceries
  - Drugstores
  - Megastores
  - Hardware stores
  - Enterprise settings
- Lots of overhead lighting in retail
- Retailers deploying LED lighting
- Customers using phones in stores
  - Surf, Scan, Share
- Customers installing retailer apps
  - Maps, Barcodes, Deals, Shopping
Visible Light Communications and Positioning

LED Luminaire

Smart Phone

Illuminate

Idle

TX <66>

TX packet

Captured using a rolling shutter

Image processing extracts beacon locations and frequencies

Compute

Minimize

\[
\begin{align*}
    d_{i,j}^2 &= (u_0 - u_i)^2 + (v_0 - v_j)^2 + (w_0 - w_j)^2 \\
    &= (K_0 a_0 - K_1 a_j)^2 + (K_0 b_0 - K_1 b_j)^2 + Z_j(K_0 - K_1)^2 \\
    &= K_0^2 |\bar{O}_{i0}|^2 + K_1^2 |\bar{O}_{j0}|^2 - 2K_0 K_1 (\bar{O}_{i0} \cdot \bar{O}_{j0}) \\
    &= (x_0 - x_i)^2 + (y_0 - y_j)^2 + (z_0 - z_j)^2,
\end{align*}
\]

\[
\sum_{i=1}^{N} \{ (x_i - x_i)^2 + (y_i - y_i)^2 + (z_i - z_i)^2 \}.
\]
Harmonia Tag

Beamforming & Localization

Mobile Tag

Carrier

UWB Mask

Harmonia

Tag

5.4e+09
5.6e+09
5.8e+09
6e+09
6.2e+09

Frequency (Hz)

Amplitude (dBm)
Outline

• Announcements

• Review

• ARM AHB-Lite
What happens after a power-on-reset (POR)?

- On the ARM Cortex-M3
- SP and PC are loaded from the code (.text) segment
- **Initial stack pointer**
  - LOC: 0x00000000
  - POR: SP ← mem(0x00000000)
- **Interrupt vector table**
  - *Initial* base: 0x00000004
  - Vector table is relocatable
  - Entries: 32-bit values
  - Each entry is an address
  - Entry #1: reset vector
    - LOC: 0x00000004
    - POR: PC ← mem(0x00000004)
- **Execution begins**

```assembly
.equ STACK_TOP, 0x20000800
.text
.syntax unified
.thumb
.global _start
.type start, %function

_start:
.word STACK_TOP, start

start:
  movs r0, #10
  ...
```
### System Memory Map

<table>
<thead>
<tr>
<th>System Registers</th>
<th>0x0E043000 – 0xFFFF0FFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Memory Type 1</td>
<td>0xE0045000 – 0xE004FFFF</td>
</tr>
<tr>
<td>External Memory Type 0</td>
<td>0xE0000000 – 0xE001FFFF</td>
</tr>
<tr>
<td>eNVM Controller</td>
<td>eNVM Controller</td>
</tr>
<tr>
<td>eNVM Aux Block (spare pages)</td>
<td>eNVM Aux Block (spare pages)</td>
</tr>
<tr>
<td>eNVM Aux Block (array)</td>
<td>eNVM Aux Block (array)</td>
</tr>
<tr>
<td>eNVM Spare Pages</td>
<td>eNVM Spare Pages</td>
</tr>
<tr>
<td>eNVM Array</td>
<td>eNVM Array</td>
</tr>
<tr>
<td>Peripherals (BB view)</td>
<td></td>
</tr>
<tr>
<td>FPGA Fabric</td>
<td>FPGA Fabric</td>
</tr>
<tr>
<td>FPGA Fabric eSRAM Backdoor</td>
<td>FPGA Fabric eSRAM Backdoor</td>
</tr>
<tr>
<td>Analog Compute Engine</td>
<td>Analog Compute Engine</td>
</tr>
<tr>
<td>IAP Controller</td>
<td>IAP Controller</td>
</tr>
<tr>
<td>eROM</td>
<td>eROM</td>
</tr>
<tr>
<td>RTC</td>
<td>RTC</td>
</tr>
<tr>
<td>MSS GPIO</td>
<td>MSS GPIO</td>
</tr>
<tr>
<td>I2C_1</td>
<td>I2C_1</td>
</tr>
<tr>
<td>SPI_1</td>
<td>SPI_1</td>
</tr>
<tr>
<td>UART_1</td>
<td>UART_1</td>
</tr>
<tr>
<td>Fabric interface interrupt Controller</td>
<td>Fabric interface interrupt Controller</td>
</tr>
<tr>
<td>Watchdog</td>
<td>Watchdog</td>
</tr>
<tr>
<td>Timer</td>
<td>Timer</td>
</tr>
<tr>
<td>Peripheral DMA</td>
<td>Peripheral DMA</td>
</tr>
<tr>
<td>Ethernet MAC</td>
<td>Ethernet MAC</td>
</tr>
<tr>
<td>I2C_0</td>
<td>I2C_0</td>
</tr>
<tr>
<td>SPI_0</td>
<td>SPI_0</td>
</tr>
<tr>
<td>UART_0</td>
<td>UART_0</td>
</tr>
<tr>
<td>eSRAM_0/eSRAM_1 (BB view)</td>
<td></td>
</tr>
<tr>
<td>eSRAM_1</td>
<td>eSRAM_1</td>
</tr>
<tr>
<td>eSRAM_0</td>
<td>eSRAM_0</td>
</tr>
<tr>
<td>eNVM (Cortex-M3) Virtual View</td>
<td>eNVM (fabric) Virtual View</td>
</tr>
</tbody>
</table>

**Figure 2-4 • System Memory Map with 54 Kbytes of SRAM**
Accessing memory locations from C

- Memory has an address and value
- Can equate a pointer to desired address
- Can set/get de-referenced value to change memory

```c
#define SYSREG_SOFT_RST_CR 0xE0042030

uint32_t *reg = (uint32_t *)(SYSREG_SOFT_RST_CR);

main () {
    *reg |= 0x00004000; // Reset GPIO hardware
    *reg &= ~(0x00004000);
}
```
Some useful C keywords

• **const**
  - Makes variable value or pointer parameter unmodifiable
  - `const foo = 32;`

• **register**
  - Tells compiler to locate variables in a CPU register if possible
  - `register int x;`

• **static**
  - Preserve variable value after its scope ends
  - Does not go on the stack
  - `static int x;`

• **volatile**
  - Opposite of `const`
  - Can be changed in the background
  - `volatile int l;`
What happens when this “instruction” executes?

```c
#include <stdio.h>
#include <inttypes.h>

#define REG_FOO 0x40000140

main () {
    uint32_t *reg = (uint32_t *)(REG_FOO);
    *reg += 3;
    printf("0x%x\n", *reg); // Prints out new value
}
```
“*reg += 3” is turned into a ld, add, str sequence

• Load instruction
  - A bus read operation commences
  - The CPU drives the address “reg” onto the address bus
  - The CPU indicated a read operation is in process (e.g. R/W#)
  - Some “handshaking” occurs
  - The target drives the contents of “reg” onto the data lines
  - The contents of “reg” is loaded into a CPU register (e.g. r0)

• Add instruction
  - An immediate add (e.g. add r0, #3) adds three to this value

• Store instruction
  - A bus write operation commences
  - The CPU drives the address “reg” onto the address bus
  - The CPU indicated a write operation is in process (e.g. R/W#)
  - Some “handshaking” occurs
  - The CPU drives the contents of “r0” onto the data lines
  - The target stores the data value into address “reg”
Modern embedded systems have multiple busses.
Why have so many busses?

- Many designs considerations
  - Master vs Slave
  - Internal vs External
  - Bridged vs Flat
  - Memory vs Peripheral
  - Synchronous vs Asynchronous
  - High-speed vs low-speed
  - Serial vs Parallel
  - Single master vs multi master
  - Single layer vs multi layer
  - Multiplexed A/D vs demultiplexed A/D

- Discussion: what are some of the tradeoffs?
APB

- **IDLE**
  - Default APB state

- **SETUP**
  - When transfer required
  - PSELx is asserted
  - Only one cycle

- **ACCESS**
  - PENABLE is asserted
  - Addr, write, select, and write data remain stable
  - Stay if PREADY = L
  - Goto IDLE if PREADY = H and no more data
  - Goto SETUP is PREADY = H and more data pending
APB signal definitions

- **PCLK**: the bus clock source (rising-edge triggered)
- **PRESETn**: the bus (and typically system) reset signal (active low)
- **PADDR**: the APB address bus (can be up to 32-bits wide)
- **PSELx**: the select line for each slave device
- **PENABLE**: indicates the 2\textsuperscript{nd} and subsequent cycles of an APB xfer
- **PWRITE**: indicates transfer direction (Write=H, Read=L)
- **PWDATA**: the write data bus (can be up to 32-bits wide)
- **PREADY**: used to extend a transfer
- **PRDATA**: the read data bus (can be up to 32-bits wide)
- **PSLVERR**: indicates a transfer error (OKAY=L, ERROR=H)
Let's say we want a device that provides data from a switch on a read to any address it is assigned. (so returns a 0 or 1)
Device provides data from switch A if address 0x00001000 is read from. B if address 0x00001004 is read from.

<table>
<thead>
<tr>
<th>PWRITE</th>
<th>PENABLE</th>
<th>PSEL</th>
<th>PADDR[7:0]</th>
<th>PCLK</th>
<th>PRDATA[32:0]</th>
<th>PREADY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mr. Switch

Mrs. Switch
All reads read from register, all writes write...

We are assuming APB only gets lowest 8 bits of address here…
Outline

- Announcements
- Review
- ARM AHB-Lite
Advanced Microcontroller Bus Architecture (AMBA)
- Advanced High-performance Bus (AHB)
- Advanced Peripheral Bus (APB)

**AHB**
- High performance
- Pipelined operation
- Burst transfers
- Multiple bus masters
- Split transactions

**APB**
- Low power
- Latched address/control
- Simple interface
- Suitable of many peripherals
Actel SmartFusion system/bus architecture
AHB-Lite supports single bus master and provides high-bandwidth operation

- Burst transfers
- Single clock-edge operation
- Non-tri-state implementation
- Configurable bus width
AHB-Lite bus master/slave interface

- Global signals
  - HCLK
  - HRESETn
- Master out/slave in
  - HADDR (address)
  - HWDATA (write data)
  - Control
    - HWRITE
    - HSIZE
    - HBURST
    - HPROT
    - HTRANS
    - HMASTLOCK
- Slave out/master in
  - HRDATA (read data)
  - HREADY
  - HRESP
AHB-Lite signal definitions

- **Global signals**
  - HCLK: the bus clock source (rising-edge triggered)
  - HRESETn: the bus (and system) reset signal (active low)

- **Master out/slave in**
  - HADDR[31:0]: the 32-bit system address bus
  - HWDATA[31:0]: the system write data bus
  - Control
    - HWRITE: indicates transfer direction (Write=1, Read=0)
    - HSIZE[2:0]: indicates size of transfer (byte, halfword, or word)
    - HBURST[2:0]: indicates single or burst transfer (1, 4, 8, 16 beats)
    - HPROT[3:0]: provides protection information (e.g. I or D; user or handler)
    - HTRANS: indicates current transfer type (e.g. idle, busy, nonseq, seq)
    - HMASTLOCK: indicates a locked (atomic) transfer sequence

- **Slave out/master in**
  - HRDATA[31:0]: the slave read data bus
  - HREADY: indicates previous transfer is complete
  - HRESP: the transfer response (OKAY=0, ERROR=1)
Key to timing diagram conventions

- **Timing diagrams**
  - Clock
  - Stable values
  - Transitions
  - High-impedance

- **Signal conventions**
  - Lower case ‘n’ denote active low (e.g. RESETn)
  - Prefix ‘H’ denotes AHB
  - Prefix ‘P’ denotes APB
Basic read and write transfers with no wait states

Figure 3-1 Read transfer

Figure 3-2 Write transfer
Read transfer with two wait states

Two wait states added by slave by asserting HREADY low

Valid data produced
Write transfer with one wait state

One wait state added by slave by asserting HREADY low

Valid data held stable
Wait states extend the address phase of next transfer

Address stage of the next transfer is also extended.

One wait state added by slave by asserting HREADY low.
Transfers can be of four types (HTRANS[1:0])

- **IDLE (b00)**
  - No data transfer is required
  - Slave must OKAY w/o waiting
  - Slave must ignore IDLE

- **BUSY (b01)**
  - Insert idle cycles in a burst
  - Burst will continue afterward
  - Address/control reflects next transfer in burst
  - Slave must OKAY w/o waiting
  - Slave must ignore BUSY

- **NONSEQ (b10)**
  - Indicates single transfer or first transfer of a burst
  - Address/control unrelated to prior transfers

- **SEQ (b11)**
  - Remaining transfers in a burst
  - Addr = prior addr + transfer size
A four beat burst with master busy and slave wait

Master busy indicated by HTRANS[1:0]

One wait state added by slave by asserting HREADY low
Controlling the size (width) of a transfer

- HSIZE[2:0] encodes the size
- The cannot exceed the data bus width (e.g. 32-bits)
- HSIZE + HBURST is determines wrapping boundary for wrapping bursts
- HSIZE must remain constant throughout a burst transfer

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>Byte</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>16</td>
<td>Halfword</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>32</td>
<td>Word</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>64</td>
<td>Doubleword</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>128</td>
<td>4-word line</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>256</td>
<td>8-word line</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>512</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1024</td>
<td></td>
</tr>
</tbody>
</table>
Controlling the burst beats (length) of a transfer

- Burst of 1, 4, 8, 16, and undef
- HBURST[2:0] encodes the type
- Incremental burst
- Wrapping bursts
  - 4 beats x 4-byte words wrapping
  - Wraps at 16 byte boundary
  - E.g. 0x34, 0x38, 0x3c, 0x30,…
- Bursts must not cross 1KB address boundaries

<table>
<thead>
<tr>
<th>HBURST[2:0]</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b000</td>
<td>SINGLE</td>
<td>Single burst</td>
</tr>
<tr>
<td>b001</td>
<td>INCR</td>
<td>Incrementing burst of undefined length</td>
</tr>
<tr>
<td>b010</td>
<td>WRAP4</td>
<td>4-beat wrapping burst</td>
</tr>
<tr>
<td>b011</td>
<td>INCR4</td>
<td>4-beat incrementing burst</td>
</tr>
<tr>
<td>b100</td>
<td>WRAP8</td>
<td>8-beat wrapping burst</td>
</tr>
<tr>
<td>b101</td>
<td>INCR8</td>
<td>8-beat incrementing burst</td>
</tr>
<tr>
<td>b110</td>
<td>WRAP16</td>
<td>16-beat wrapping burst</td>
</tr>
<tr>
<td>b111</td>
<td>INCR16</td>
<td>16-beat incrementing burst</td>
</tr>
</tbody>
</table>
A four beat wrapping burst (WRAP4)
A four beat incrementing burst (INCR4)
An eight beat wrapping burst (WRAP8)
An eight beat incrementing burst (INCR8) using half-word transfers
An undefined length incrementing burst (INCR)
Multi-master AHB-Lite requires a multi-layer interconnect

- AHB-Lite is single-master

- Multi-master operation
  - Must isolate masters
  - Each master assigned to layer
  - Interconnect arbitrates slave accesses

- Full crossbar switch often unneeded
  - Slaves 1, 2, 3 are shared
  - Slaves 4, 5 are local to Master 1
Questions?

Comments?

Discussion?