Lecture 12: Memory accesses in C
Memory and Peripheral Busses
PCB design/terminology

Slides developed in part by Prof. Dutta
Outline

• C and memory (review/exam question)

• Memory review (very short)

• AHB bus

• Start on PCB material (time allowing)

(no student talks today, but three on Thursday)
• Write a C function “clear_and_enable” which takes a single external interrupt source as an argument. It clears the pending status for that register and enables interrupts from that source. It should return a 1 if the source was already enabled, otherwise it should return a 0. You need not check to see if x is a reasonable value.
So, what do we want to do?

• Idea (in English):

• Pseudo code:
Let’s do it wrong.

```c
int clear_and_enable(int x)
{
    int *s_en     =0xE000E100;
    int *c_pend   =0xE000E280;
    int index     =x>>5;
    int offset    =x & 5'b11111;
    int mask      =1<<offset;
    int estatus   =*(s_en+index) & mask;

    *(c_pend+index)&=mask; // clear pending
    if(estatus) // (would estatus==1 work here?)
        return(1);
    *(s_en+index)|=mask; // set enable
    return(0);
}
```

(errors are described in the pptx notes)
Now try to do it right (without looking!)
Outline

- C and memory (review/exam question)
- Memory review (very short)
- AHB bus
- Start on PCB material (time allowing)
Memory basics

• Read only

• Volatile
  - Dynamic

• Wearout

• Latencies
Memory array types

Memory Arrays

Random Access Memory
- Read/Write Memory (RAM) (Volatile)
  - Static RAM (SRAM)
  - Dynamic RAM (DRAM)
- Read Only Memory (ROM) (Nonvolatile)
  - Serial In Parallel Out (SIPO)
  - Parallel In Serial Out (PISO)

Serial Access Memory
- Shift Registers
- Queues
  - First In First Out (FIFO)
  - Last In First Out (LIFO)

Content Addressable Memory (CAM)
- Erasable Programmable ROM (EPROM)
- Electrically Erasable Programmable ROM (EEPROM)
- Flash ROM
- Ferroelectric RAM
- Phase change memory
- Magnetoresistive RAM
- Programmable metallization cell
• C and memory (review/exam question)

• Memory review (very short)

• AHB-Lite bus

• Start on PCB material (time allowing)
Modern embedded systems have multiple busses
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 like APB
- Configurable bus width

Most figures in this section from AMBA 3 AHB-Lite Protocol v1.0
You have to register with ARM (trivial) for this, we can’t post it.
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]**: burst transfer size/order (1, 4, 8, 16 beats or undefined)
    - **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 (As APB)

- **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

Pipelined Address & Data 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)**
  - Master inserts idle cycles in a burst (wait states)
  - 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
## Table 3-1 Transfer type encoding

<table>
<thead>
<tr>
<th>HTRANS[1:0]</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b00</td>
<td>IDLE</td>
<td>Indicates that no data transfer is required. A master uses an IDLE transfer when it does not want to perform a data transfer. It is recommended that the master terminates a locked transfer with an IDLE transfer. Slaves must always provide a zero wait state OKAY response to IDLE transfers and the transfer must be ignored by the slave.</td>
</tr>
<tr>
<td>b01</td>
<td>BUSY</td>
<td>The BUSY transfer type enables masters to insert idle cycles in the middle of a burst. This transfer type indicates that the master is continuing with a burst but the next transfer cannot take place immediately. When a master uses the BUSY transfer type the address and control signals must reflect the next transfer in the burst. Only undefined length bursts can have a BUSY transfer as the last cycle of a burst. See <em>Burst termination after a BUSY transfer</em> on page 3-10 for more information. Slaves must always provide a zero wait state OKAY response to BUSY transfers and the transfer must be ignored by the slave.</td>
</tr>
<tr>
<td>b10</td>
<td>NONSEQ</td>
<td>Indicates a single transfer or the first transfer of a burst. The address and control signals are unrelated to the previous transfer. Single transfers on the bus are treated as bursts of length one and therefore the transfer type is NONSEQUENTIAL.</td>
</tr>
<tr>
<td>b11</td>
<td>SEQ</td>
<td>The remaining transfers in a burst are SEQUENTIAL and the address is related to the previous transfer. The control information is identical to the previous transfer. The address is equal to the address of the previous transfer plus the transfer size, in bytes, with the transfer size being signaled by the HSIZE[2:0] signals. In the case of a wrapping burst the address of the transfer wraps at the address boundary.</td>
</tr>
</tbody>
</table>

Figure 3-6 on page 3-6 shows the use of the NONSEQ, BUSY, and SEQ transfer types.
A four beat burst with master busy and slave wait

- Master busy indicated by $\text{HTRANS}[1:0]$
- One wait state added by slave by asserting $\text{HREADY}$ 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 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 number of **beats**

- 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
AHB-Lite takeaways

- AHB-Lite is a pipelined bus
  - Address of one transaction at the same time as data of another transaction

- AHB-Lite deals with real-world problems
  - Critical-word first (for caches)
  - Longer data phases (burst) to reduce overhead.
  - Locks

- Still has flow control
Outline

- C and memory (review/exam question)
- Memory review (very short)
- AHB bus
- Start on PCB material (time allowing)
So you want to make a Printed Circuit Board...

- At the end of the day a PCB is just a set of wires that connect components.
  - But there are some issues
    - The wires have restricted dimensionality
    - The wires are very thin
      - So high resistance (as conductors go)
    - The board needs to include holes (or pads) for the devices.
    - You can’t easily change things once you build it.

PCBs – basic terminology

http://www.linkwitzlab.com/Pluto/supplies-subw.htm,
Basic Terminology

• The wires you are laying out are called “traces” or “tracks”

• Inside of a given “layer” traces which cross are electrically connected.
  – If you have traces on both sides of the board, you are said to have two layers.

• Through-hole: Having holes in the PCB designed to have pins put through the hole
  – Contrast with surface mount where device goes on top.
Parts of a PCB

- Copper (pads & traces)
- Soldermask (green)
- Silkscreen (white)
- Drill holes
- Via
- Bottom side

PCBs – basic terminology
Vias

- Sometimes you need to connect two traces on two different layers.
  - To do this we use a via.
  - It is just a plated through hole
    - Generally smaller than a through hole for a part.
Clearances

• There will be space between the traces, plated holes and each other.
  – You need to meet the requirement of the manufacturer.
The layered construction of a PCB:
A six layer board
So, how do I design a PCB?

1. Create schematic
2. Place parts
3. Route interconnect
4. Generate files
Step 1: Create schematic

- The first thing you want is something that looks like a textbook circuit diagram. It just shows the devices and how they are connected.
  - Sometimes you will worry about pinouts here (say when working with a microprocessor maybe)
  - But usually you don’t

- **No notion of layout** belongs here!
Why a schematic?

• In general it is drawn to be *readable*.
  – This is probably what your sketch on paper would look like.
  – You can find and fix bugs more easily here than the PCB layout.
Step 2: Place parts

• You need to place the patterns on the board.
  – You need to not overlap them to that the components can actually fit on the board.
  – You want to leave room for the traces to connect everything.

• This is very much an art form.
  – In fact you will find people who rant about “sloppy” or “unprofessional” placements.

• Some tools will do this for you. No one seems to like them.
Patterns

• Once you know what it is you want to build, you need to figure out how to lay it out on the board.
  – You need to know how big each piece is, and where the holes need to be placed.

• Each device has a pattern which shows exactly that.
  – You will occasionally need to create a pattern.
Step 3: Route interconnect

- A route is a connection between devices.
  - It may consist of multiple traces
- There are design rules which include:
  - Minimum trace width
  - Minimum spacing between traces and holes
  - Minimum spacing between holes and holes.
- These rules will vary by manufacturer.
  - Even better, units will vary by manufacturer!
  - Time for a brief aside...
Issues of measure

- PCB land uses some interesting terminology.
  - A “thou” is a thousandth of an inch.
  - A “mm” is a millimeter.
  - A “mil” is a thousandth of an inch.
- Thou is generally preferred over mill to avoid confusion, but most tools/vendors use mill.
Trace width

• In general most PCB manufactures seem to have trace-width minimums of 6-10 thous.
  – Most are willing to go smaller for a price.
• A rule of thumb is to use a 50 thou minimum for power/ground and 25 for everything else.
  – This is to drop the resistance of the traces.
  – In general you are worried about heat dissipation
• There are lots of guidelines for width/power but in general you are looking at:
  – A 10cm trace needs to be 10 thou wide if it will carry 1 amp.
  – 5 amps at 10cm would require 110 thou.
Trace width continued

• The *problem* with wide traces is that they are hard to route.
  – In particular you might wish to go between pins of a device.

• One solution is to be wide normally and “neck down” when you have to.
  – This is more reasonable than you think.
    • Think resistors in series.
A rat’s nest shows the placement of the devices and the connections but not the routing

- Automatically generated for you.
- Sometimes before placement, sometimes after
  - Varies by tool.
Routing for real

• You can use an autorouter to route your traces
  – Some people hate these as the design will be “ugly”
  – Saves a lot of time.
  – Oddly, not as good as a person can do.
    • But much faster.
• Still generally need to do some (or all) of the routing by hand
  – Very tedious...

PCBs – design steps: routing
Routing quality

An example of GOOD power routing (Left) and BAD power routing (Right)
Step 4: Generate files

• Once the design is done, a set of files are generated.
  – Each file describes something different (e.g.)
    • Copper on a given layer
    • Silkscreen
    • Solder mask
  – Most files are in “Gerber” format
    • Human-readable (barely) ASCII format
    • Has commands like \textit{draw} and \textit{fill}.
  – Drill files are a different format called Excellon
    • Also human-readable (barely) ASCII with locations and diameters for the holes.

• Generally you zip all these files up and ship them as a single file to the PCB manufacturer.
  – Often a good idea to include the design file(s) too.
The schematic captures the logical circuit design.
Floorplanning captures the desired part locations
The auto-router places tracks on the board, saving time.
The layered construction of a PCB:
A six layer board

Figure from altium.com