Standard IO

Speaker: Juin-Nan Liu

Adopted from National Taiwan University
SoC Design Laboratory
Goal of This Lab

- Familiarize with ARM I/O architecture
- Know what Semihosting is
- Semihosting exercise
Outline

- **ARM system input/output (I/O) functions**
- Semihosting [5]
- Lab - Semihosting
Memory-Mapped Peripherals

- The input/output (I/O) functions are implemented in an ARM system using a combination of memory-mapped addressable peripheral registers and the interrupt inputs.
- A peripheral device contains a number of registers. In a memory-mapped system, each of these registers appears like a memory location at a particular address.
### Peripheral Registers

<table>
<thead>
<tr>
<th>Base address</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1F000000</td>
<td>16MB</td>
<td>Spare</td>
</tr>
<tr>
<td>0x1E000000</td>
<td>16MB</td>
<td>Spare</td>
</tr>
<tr>
<td>0x1D000000</td>
<td>16MB</td>
<td>Spare</td>
</tr>
<tr>
<td>0x1C000000</td>
<td>16MB</td>
<td>Spare</td>
</tr>
<tr>
<td>0x1B000000</td>
<td>16MB</td>
<td>GPIO</td>
</tr>
<tr>
<td>0x1A000000</td>
<td>16MB</td>
<td>LED display and boot switch</td>
</tr>
<tr>
<td>0x19000000</td>
<td>16MB</td>
<td>Mouse</td>
</tr>
<tr>
<td>0x18000000</td>
<td>16MB</td>
<td>Keyboard</td>
</tr>
<tr>
<td>0x17000000</td>
<td>16MB</td>
<td>UART1</td>
</tr>
<tr>
<td>0x16000000</td>
<td>16MB</td>
<td>UART0</td>
</tr>
<tr>
<td>0x15000000</td>
<td>16MB</td>
<td>RTC</td>
</tr>
<tr>
<td>0x14000000</td>
<td>16MB</td>
<td>Interrupt controller</td>
</tr>
<tr>
<td>0x13000000</td>
<td>16MB</td>
<td>Counter/timers</td>
</tr>
<tr>
<td>0x12000000</td>
<td>16MB</td>
<td>EBI configuration registers</td>
</tr>
<tr>
<td>0x11000000</td>
<td>16MB</td>
<td>System controller registers</td>
</tr>
<tr>
<td>0x10000000</td>
<td>16MB</td>
<td>Core module registers (for core modules)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Spare (for logic modules)</td>
</tr>
</tbody>
</table>

*SOC Consortium Course Material*
The ARM fast interrupt (FIQ) architecture includes more **banked registers** than the other exception modes in order to minimize the register save and restore overhead associated with handling one of these interrupts. [1]
Input/Output

- In many ARM systems I/O locations are made inaccessible to user code, so the only way the devices can be accessed is through *supervisor calls* (SWIs) or through C library functions written to use those calls.

- The I/O area of memory is normally marked as *uncache-able*, and accesses bypass the cache.

- All the low-level detail of the I/O device registers and the handling of interrupts is the responsibility of the OS.
ARM µHAL

- µHAL is a *Hardware Abstraction Layer* that is designed to conceal hardware difference between different systems.
- ARM µHAL provides a standard layer of board-dependent functions to manage I/O, RAM, boot flash, and application flash.
µHAL Examples

- µHAL provides simple & extended functions that are linkable and code reusable to control the system hardware.

AFS: ARM Firmware Suite
Outline

- ARM system input/output (I/O) functions
- **Semihosting** [5]
- Lab - Semihosting
Semihosting

What is Semihosting?
- A mechanism whereby the target communicates I/O requests made in the application code to the host system, rather than attempting to support the I/O itself.

Semihosting overview
How Semihosting Work

- The application invokes the semihosting SWI (Software Interrupt).
- The debug agent then handles the SWI exception.
- The debug agent provides the necessary communication to the host system.
- Semihosting operations are requested using a semihosted SWI numbers:
  - 0x123456 in ARM state.
  - 0xAB in Thumb state.
SWI Interface

- A Software Interrupt (SWI) is requested with an SWI number.
  - Semihosting SWI numbers: 0x123456 (ARM), 0xAB (Thumb)
- Different operations in the SWI are identified using value of $r0$.
- Other parameters are passed in a block that is pointed by $r1$.
- The result is returned in $r0$. It could be an immediate value or a pointer.
Semihosting SWIs

- Semihosting operations used by C library functions such as `printf()`, `scanf()`.
- No need to implement semihosting operations for default standard I/O functions.

<table>
<thead>
<tr>
<th>SWI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SYS_OPEN (0x01)</code> on page 5-12</td>
<td>Open a file on the host</td>
</tr>
<tr>
<td><code>SYS_CLOSE (0x02)</code> on page 5-14</td>
<td>Close a file on the host</td>
</tr>
<tr>
<td><code>SYS_WRITEC (0x03)</code> on page 5-14</td>
<td>Write a character to the console</td>
</tr>
<tr>
<td><code>SYS_WRITEX (0x04)</code> on page 5-14</td>
<td>Write a null-terminated string to the console</td>
</tr>
<tr>
<td><code>SYS_WRITE (0x05)</code> on page 5-15</td>
<td>Write to a file on the host</td>
</tr>
<tr>
<td><code>SYS_READ (0x06)</code> on page 5-16</td>
<td>Read the contents of a file into a buffer</td>
</tr>
<tr>
<td><code>SYS_READC (0x07)</code> on page 5-17</td>
<td>Read a byte from the console</td>
</tr>
<tr>
<td><code>SYS_ISERROR (0x08)</code> on page 5-17</td>
<td>Determine if a return code is an error</td>
</tr>
</tbody>
</table>
Outline

- ARM system input/output (I/O) functions
- Semihosting
- Lab - Semihosting
Lab 7: Standard I/O

- **Goal**
  - Introduce students to control I/O and learn the principle of polling, interrupt, and semihosting through this Lab.

- **Principle**
  - How to access I/O via the existing library function call.

- **Guidance**
  - Micro Hardware Abstraction Layer
  - How CPU access input devices

- **Steps**
  - This program controls the Intergator board LED and print strings to the host using uHal API.

- **Requirements and Exercises**
  - Modify the LED example. When it counts, we press any key to stop counting and then press any key to continue counting numbers.

- **Discussion**
  - Explain the advantage and disadvantage of polling & interrupt.
  - A system can be divided into hardware, software, and firmware. Which one contains uHAL.
References


