I/O Devices & Debugging

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
I/O Devices
Jasmine Block Diagram
Timers and Counters

- A **timer** is incremented by a periodic signal
- A **counter** is incremented by an asynchronous, occasional signal

- Rollover causes interrupt
Jasmine Timers (1)

- Timer base address at 0x82000000
- Four 32-bit countdown timers
- The clock speed of timer is \( \text{CLOCK\_SPEED}/2 \) (87.5MHz by default)
- Timer 4 reserved for SATA retry timer
- In periodic mode, the timer generates an interrupt when the counter reaches zero, and then reloads the initial value.
Jasmine Timers (2)

- Timer registers

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>TM_1_LOAD</td>
<td>TIMER_BASE + 0x00</td>
<td>Initial value of the timer. Reloaded in periodic mode.</td>
</tr>
<tr>
<td>TM_1_VALUE</td>
<td>TIMER_BASE + 0x04</td>
<td>The current value of the timer</td>
</tr>
<tr>
<td>TM_1_CONTROL</td>
<td>TIMER_BASE + 0x08</td>
<td>Enable/disable the timer, Set clock prescaling and free-running mode vs. periodic mode</td>
</tr>
<tr>
<td>TM_1_INT_CLR</td>
<td>TIMER_BASE + 0x0c</td>
<td>Clear an interrupt generated by the timer</td>
</tr>
</tbody>
</table>
### Jasmine Timers (3)

- **Predefined resolution:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Input clock divided by</th>
<th>Resolution</th>
<th>Ticks/sec</th>
<th>Time to rollover</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_PRESCALE_0</td>
<td>1</td>
<td>11.43 ns</td>
<td>87,489,064</td>
<td>49 s</td>
</tr>
<tr>
<td>TIMER_PRESCALE_1</td>
<td>16</td>
<td>182 ns</td>
<td>5,494,506</td>
<td>781 s</td>
</tr>
<tr>
<td>TIMER_PRESCALE_2</td>
<td>256</td>
<td>2.9 us</td>
<td>344,828</td>
<td>12,455 s</td>
</tr>
</tbody>
</table>

- **Starting and reading a timer:**

```c
start_interval_measurement (TIMER_CH1, TIMER_PRESCALE_0);
...
UINT32 current = GET_TIMER_VALUE (TIMER_CH1);
```
Jasmine Timers (4)

- Programming a timer

```c
start_interval_measurement (TIMER_CH1, TIMER_PRESCALE_0);

#define SETREG(ADDR, VAL)  
  *(volatile UINT32*)(ADDR) = (UINT32) (VAL)
#define GETREG(ADDR)        (*(volatile UINT32*)(ADDR))

...  
SETREG (TM_1_CONTROL, 0);
SETREG (TM_1_INT_CLR, 0x01);
SETREG (TM_1_LOAD, 0xffffffff);
SETREG (TM_1_CONTROL, TM_ENABLE | TM_BIT_32
       | TM_MODE_PRD | TIMER_PRESCALE_0);
```
Jasmine Timers (5)

- Measuring the elapsed time

```c
void ptimer_start (void) {
    start_interval_measurement (TIMER_CH1, TIMER_PRESCALE_0);
}

UINT32 ptimer_stop (void) {
    return (0xffffffff - GET_TIMER_VALUE (TIMER_CH1));
}

void f (void) {
    UINT32 ticks;
    ...
    ptimer_start ();
    do_something ();
    ticks = ptimer_stop (); // OK if within 49 sec
}
```
Watchdog Timer

- Watchdog timer is periodically reset by system timer
- If watchdog is not reset, it generates an interrupt to reset the host
Jasmine Watchdog Timer

- Watchdog timer at 0x84000000
- 32-bit down counter with a programmable timeout interval
- Interrupt output generation on timeout
- Reset signal generation on timeout if the interrupt from the previous timeout remains unserviced by software
- Currently not used by Jasmine firmware
LED

- Must use resistor to limit current
7-Segment LCD Display

- May use parallel or multiplexed input
GPIO

- General Purpose Input/Output (GPIO)
- A generic pin on a chip whose behavior can be controlled through software
- Each pin can be configured to be input or output
- Save the hassle of having to arrange additional circuitry to provide additional control lines
Jasmine GPIOs (1)

- GPIO base address at 0x83000000
- 7 GPIO pins (GPIO_0 ~ GPIO_6)
- 4 GPIO pins (GPIO_2 ~ GPIO_5) can be used to probe signals by a logic analyzer for debugging purpose
- GPIO_2 ~ GPIO_5 are also used for UART
- GPIO_0 used for factory mode jumper (J2)
- GPIO_6 is connected to LED (D4)
Jasmine GPIOs (2)

Controller GPIO Section

- `U1E`
  - `C18`: nTRST
  - `C17`: TCK
  - `C16`: ARM_TDO
  - `B18`: ARM_RTCK
  - `D16`: TMS
  - `E15`: TDI
  - `A16`: OSCCLK
  - `B17`: nRESET
  - `E16`: TEST
  - `D17`: External_Interrupt
  - `E18`: FACTORYSET
  - `F17`: Boot_Rom_Mode_SEL
  - `G16`: RXD/SCLK
  - `G15`: TXD/SSB
  - `D18`: nRTS/MOSI
  - `F16`: nCTS/MISO
  - `E17`: LED

- `IDUX`

- `SW2`
  - `RYD`
  - `TXD`
  - `RTS`
  - `nCTS/MISO`
  - `nRTS/MOSI`
  - `LED_G`
  - `LED`
  - `TCK`
  - `TDI`
  - `nTRST`
  - `TMS`

- `SW 09`

- `SW 03`
  - `SCLK`
  - `SSB`
  - `MOSI`
  - `MISO`
  - `LED_Act`

- `SW 04`

- `J4` For probing
Jasmine GPIOs (3)

Factory Mode Jumper (J2)

LED Indicator (D4)

Setting for UART

<table>
<thead>
<tr>
<th>Switch</th>
<th>Pin</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW2</td>
<td>1, 2, 3, 4</td>
<td>ON</td>
</tr>
<tr>
<td>SW3</td>
<td>1, 2, 3, 4</td>
<td>OFF</td>
</tr>
<tr>
<td>SW4</td>
<td>1, 2, 3, 4</td>
<td>OFF</td>
</tr>
</tbody>
</table>
Jasmine GPIOs (4)

- Initializing GPIOs

```c
// <init_jasmine() @ target_spw/initialize.c>
#if OPTION_UART_DEBUG
SETREG (GPIO_DIR, BIT3 | BIT4 | BIT 6);
#else
SETREG (GPIO_DIR, BIT6);
#endif
UINT32   temp = GETREG (GPIO_REG);
SETREG (GPIO_REG, 0x3); // Set Pins 0 and 1 to HIGH
```

- Reading and writing GPIOs
Jasmine GPIOs (5)

- Controlling LED (D4) < target_spw/misc.c>

```c
void led (BOOL32 on)
{
    UINT32 temp;
    temp = GETREG (GPIO_REG);
    if (on)
        temp |= (1 << 6);
    else
        temp &= ~(1 << 6);
    SETREG (GPIO_REG, temp);
}

// NOTE: Infinite loop!!!
void led_blink (void)
{
    while (1)
    {
        led (1);
        delay (700000);
        led (0);
        delay (700000);
    }
}
```
Keyboard

- A switch must be debounced to eliminate multiple contacts caused by mechanical bouncing:
Encoded Keyboard

- An array of switches is read by an encoder
- Contain a microprocessor to preprocess button inputs
- Reads only one row of switches at a time
- N-key rollover remembers multiple key depressions
High-Resolution Display

- Liquid crystal display (LCD) is dominant form
- Plasma, OLED, etc.
- Frame buffer holds current display contents
  - Written by processor
  - Read by video
**Touchscreen**

- Includes input and output device
- Input device is a two-dimensional voltmeter (resistive touchscreen)
Development & Debugging
Debugging Challenges

- Target system may be hard to observe
- Target may be hard to control
- May be hard to generate realistic inputs
- Setup sequence may be complex
Host/Target Design

- Use a host system to prepare software for target system:

![Diagram showing host system connected to target system via a serial line.](image-url)
Host-based Tools

- Cross compiler
  - Compiles code on host for target system

- Cross debugger (or remote debugger)
  - Displays target state, allows target system to be controlled
Software Debuggers

- A monitor program residing on the target provides basic debugger functions
- Debugger should have a minimal footprint in memory
- User program must be careful not to destroy debugger program
- The debugger should be able to recover from some damage caused by user code
Breakpoints

- A breakpoint allows the user to stop execution, examine system state, and change state.

- Replace the breakpointed instruction with a subroutine call to the monitor program
ARM Breakpoints

0x400  MUL r4, r6, r6
0x404  ADD r2, r2, r4
0x408  ADD r0, r0, #1
0x40c  B loop

uninstrumented code

0x400  MUL r4, r6, r6
0x404  ADD r2, r2, r4
0x408  ADD r0, r0, #1
0x40c  BL bkpoint

code with breakpoint
Breakpoint Handler

- Save registers
- Allow user to examine machine
- Before returning, restore system state
  - Safest way to execute the instruction is to replace it and execute in place
  - Put another breakpoint after the replaced breakpoint to allow restoring the original breakpoint
Logic Analyzers

- A logic analyzer is an array of low-grade oscilloscopes
Boundary Scan

- Simplifies testing of multiple chips on a board
  - Registers on pins can be configured as a scan chain
  - Used for debuggers, in-circuit emulators
In-Circuit Emulators

- A microprocessor in-circuit emulator is a specially-instrumented microprocessor
- JTAG-based hardware debuggers use on-chip debugging hardware with standard production chips
- Allows you to stop execution, examine CPU state, modify registers
How to Exercise Code

- Run on host system
- Run on target system
- Run in instruction-level simulator
- Run on cycle-accurate simulator
- Run in hardware/software co-simulation environment