Memory & I/O Devices

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Memory Architectures
Memory Components

- Several different types of memory
  - SRAM
  - DRAM
  - Flash

- Each type of memory comes in varying
  - Capacities
  - Widths
Random-Access Memory

- Dynamic RAM is dense, requires refresh
  - Synchronous DRAM is dominant type
  - SDRAM uses clock to improve performance, pipeline memory accesses

- Static RAM is faster, less dense, consumes more power
Read-Only Memory

- ROM may be programmed at factory
- Flash is dominant form of field-programmable ROM
  - Electrically erasable, must be block erased
  - Random access, but write/erase is much slower than read
  - NOR flash is more flexible
  - NAND flash is more dense
Requirements

**Code**

Mobile
Consumer Electronics
Networking

**Data**

Cards
MP3
USB Drives

<table>
<thead>
<tr>
<th>Read</th>
<th>Writes</th>
<th>Density</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast Random</td>
<td>Medium</td>
<td>Small – Medium</td>
<td>No bad bits</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Read</th>
<th>Writes</th>
<th>Density</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast Sequential</td>
<td>Fast</td>
<td>Large</td>
<td>Bad bits allowed</td>
</tr>
</tbody>
</table>

Source: “Non-Volatile Memories”, Intel Corp.
NOR XIP

**Pros**
- Simple, easy to design
- Execute-In-Place (XIP)
- Predictable read latency
- Code + Storage in NOR
- Firmware upgrades

**Cons**
- Slow read speed
- Much slower write speed
- The high cost of NOR
NOR Shadowing

- **Pros**
  - Faster read and write
  - Easy boot-up
  - Use a relatively pricey NOR only to boot up the system
  - Code can be compressed

- **Cons**
  - Larger DRAM needed
  - Require more design time
  - Not energy efficient
NAND Shadowing

**Pros**
- Faster read and write
- Cost effective
- NAND for both code and data storage

**Cons**
- Require a special boot mechanism
- Extensive ECC for NAND
- Larger DRAM needed
- Require more design time
- Not energy efficient
Hybrid NAND Shadowing

- **Pros**
  - Much faster read and write speed
  - ECC embedded
  - Cost effective
  - NAND for both code and data storage

- **Cons**
  - Larger DRAM needed
  - Not energy efficient
NAND Demand Paging

**Pros**
- Less DRAM required
- Low cost
- Energy efficient
- NAND for both code and data storage

**Cons**
- Require MMU-enabled CPU
- Unpredictable read latency
- Complex to design and test
The CPU Bus
The CPU Bus

- Bus allows CPU, memory, devices to communicate
  - Shared communication medium

- A bus is:
  - A set of wires
  - A communications protocol
Bus Protocols

- Bus protocol determines how devices communicate.
- Devices on the bus go through sequences of states.
  - Protocols are specified by state machines, one state machine per actor in the protocol
- May contain asynchronous logic behavior
Microprocessor Busses

- Clock provides synchronization
- R/W is true when reading
- Address is $a$-bit bundle of address lines
- Data is $n$-bit bundle of data lines
- Data ready signals when $n$-bit data is ready
Bus Multiplexing

- CPU
- Adrs enable
- device
  - data
  - adr
- Adrs enable
- data enable
DMA

- Direct memory access (DMA) performs data transfers without executing instructions
  - CPU sets up transfer
  - DMA engine fetches, writes
- DMA controller is a separate unit
Bus Mastership

- By default, CPU is bus master and initiates transfers.
- DMA must become bus master to perform its work
  - CPU can’t use bus while DMA operates
- Bus mastership protocol:
  - Bus request
  - Bus grant
DMA Operation

- CPU sets DMA registers for start address, and length
- DMA status register controls the unit
- Once DMA is bus master, it transfers automatically
  - May run continuously until complete
  - May use every $n^{th}$ bus cycle
System Bus Configurations

- Multiple busses allow parallelism
  - Slow devices on one bus
  - Fast devices on separate bus
- A bridge connects two busses
ARM AMBA
(Advanced Microcontroller Bus Architecture)
AMBA Specification

- Advanced System Bus (ASB)
- Advanced High-performance Bus (AHB)
  - High-performance system bus
- Advanced eXtensible Interface (AXI)
- Advanced Peripheral Bus (APB)
  - Lower speed, lower cost
  - All devices are slaves
- Advanced Trace Bus (ATB)
Typical Architecture

High-bandwidth Memory Interface

High-performance ARM processor

High-bandwidth on-chip RAM

DMA bus master

AHB

Bridge

APB

UART

Timer

Keypad

PIO
AHB

- High performance
- Pipelined operation
- Burst transfers
- Split transactions
- Multiple bus masters
- Single-cycle bus master handover
- Single-clock edge operation
Simple Transfer
APB

- Low power
- Latched address and control
- Simple interface
- Suitable for many peripherals
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

![Diagram showing interaction between host CPU and watchdog timer](image)
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
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

- TRST
- TCK
- TDO
- RTCK
- TMS
- TDI
- OSCCLK
- RESET
- TEST
- EINT
- GPIO_0
- GPIO_1
- GPIO_2
- GPIO_3
- GPIO_4
- GPIO_5
- GPIO_6
- IDLX

- C18
- C17
- C16
- C15
- B18
- D16
- E15
- E14
- A16
- B17
- E16
- D17
- E18
- F17
- G16
- G15
- D18
- F16
- E17

- nTRST
- TCK
- ARM_TDO
- ARM_RTCK
- TMS
- TDI
- OSCCLK
- nRESET
- TEST
- External_Interrupt
- FACTORYSET
- Boot_Rom_Mode_Sel
- RXD/SCLK
- TXD/SSB
- nRTS/MOSI
- nCTS/MISO
- LED

SW2
- RXD
- TXD
- RTS
- nRTS
- LED_G
- nCTS
- LED_Act
- ARM_TCK
- ARM_TDI
- ARM_TDO
- ARM_nTRST
- ARM_TMS

SW DIP-09

SW3

SW DIP-09

SW4

HEADER 4

J4 For probing
Jasmine GPIOs (3)

Factory Mode Jumper (J2)

Setting for UART

<table>
<thead>
<tr>
<th>SW2</th>
<th>pin 1, 2, 3, 4</th>
<th>ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW3</td>
<td>pin 1, 2, 3, 4</td>
<td>OFF</td>
</tr>
<tr>
<td>SW4</td>
<td>pin 1, 2, 3, 4</td>
<td>OFF</td>
</tr>
</tbody>
</table>

LED Indicator (D4)
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);
    }
}
```