IP Core Design-Lab 4

ARM Hardware Development Environment

Outline

- About ARM Hardware Development Environment
- Examples:
  - Part A: Semihosting
  - Part B: Memory Usage
  - Part C:Timer/Interrupt
- Polling & Interrupt
- Lab Exercise
- Reference Topic & Related Documents

About ARM Hardware Development Environment

- ARM Integrator Application Platform (AP) Resources
- ARM Core Module (CM) Resources
- System Memory Map
- Running Images on ARM Integrator

ARM Integrator ASIC Platform/AP Resources

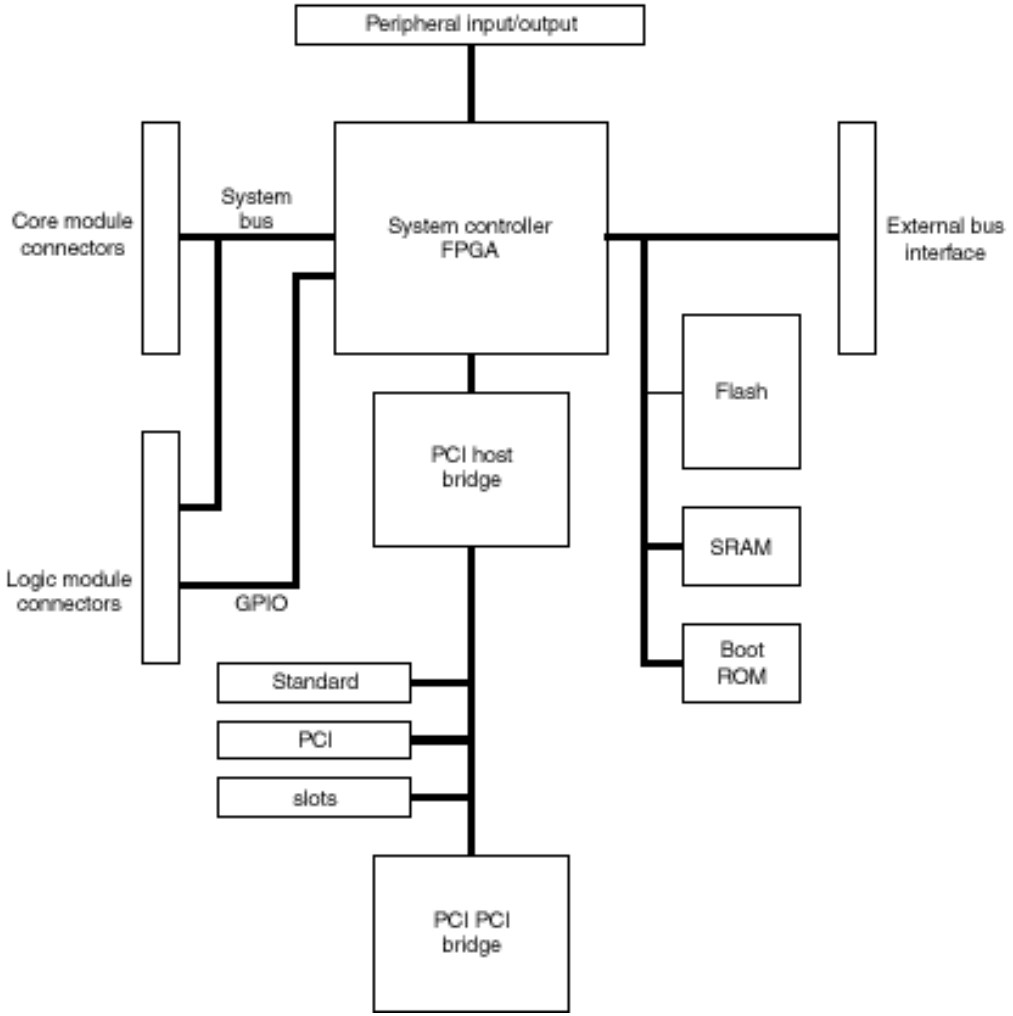
- About ARM Integrator AP
  - An ATX motherboard which can be used to support the development of applications and hardware with ARM processor.
  - Platform board provides the *AMBA* backbone and system infrastructure required.
  - Core Modules & Logic Modules could be attached to ASIC Platform.

ARM Integrator ASIC Platform/AP Resources

- ARM Integrator/AP Overview:
  - System controller FPGA.:
    - System bus to CMs and LMs
    - System bus arbiter
    - Interrupt controller
    - Peripheral I/O controller
    - 3 counter/timers
    - Reset controller
    - System status and control registers
  - Clock Generator
  - PCI bus interface supporting onboard expansion.
  - External Bus Interface (EBI) supporting external memory expansion.
  - Boot ROM
  - 32MB flash memory.
  - 512K SSRAM.

ARM Integrator ASIC Platform/AP Resources

- ARM Integrator/AP Architecture:



ARM Integrator ASIC Platform/AP Resources

- System Controller FPGA (1/3)
  - System Bus Interface
    - Supports transfers between system bus and the *Advanced Peripheral Bus* (APB).
    - Supports transfers between system bus and the PCI bus.
    - Supports transfers between system bus and the *External Bus Interface* (EBI).
  - System Bus Arbiter
    - Provides arbitration for a total of 6 bus masters.
      - Up to 5 masters on CMs or LMs.
      - PCI bus bridge.

ARM Integrator ASIC Platform/AP Resources

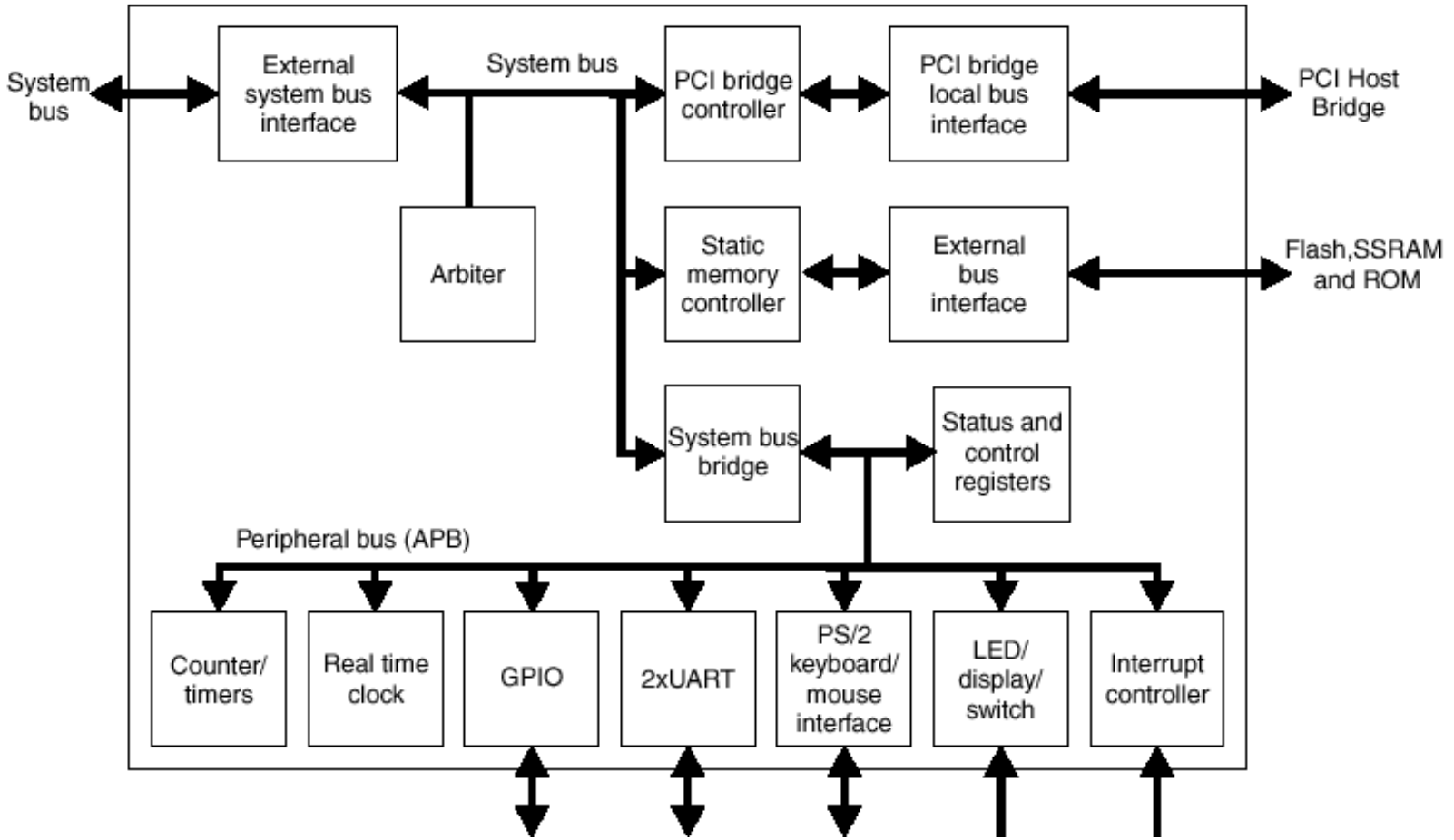
- System Controller FPGA (2/3)
  - Peripheral I/O Controllers
    - 2 ARM PrimeCell UARTs
    - ARM PrimeCell Keyboard & Mouse Interface (KMI)
    - ARM PrimeCell Real Time Clock (RTC)
    - 3 16-bit counter/timers
    - GPIO controller
    - Alphanumeric display and LED control, and switch reader
  - Reset Controller
    - Initializes the Integrator/AP when the system is reset
  - System Status & Control Register
    - Clock speeds
    - Software reset
    - Flash memory write protection

ARM Integrator ASIC Platform/AP Resources

- System Controller FPGA (3/3)
  - Interrupt Controller
    - Handles IRQs and FIQs for up to 4 ARM processors.
    - IRQs and FIQs originate form the peripheral controllers, OCI bus, and other devices on LMs.
    - Assigns IRQs and FIQs from any sources to any of the 4 ARM processors.
    - Interrupts are masked enabled, acknowledged, or cleared via registers in the interrupt controller.
    - Main sources of interrupts:
      - System controller's internal peripherals
      - LM's devices
      - PCI subsystem
      - software

ARM Integrator ASIC Platform/AP Resources

- System Controller FPGA block diagram



ARM Integrator Core Module/CM Resources

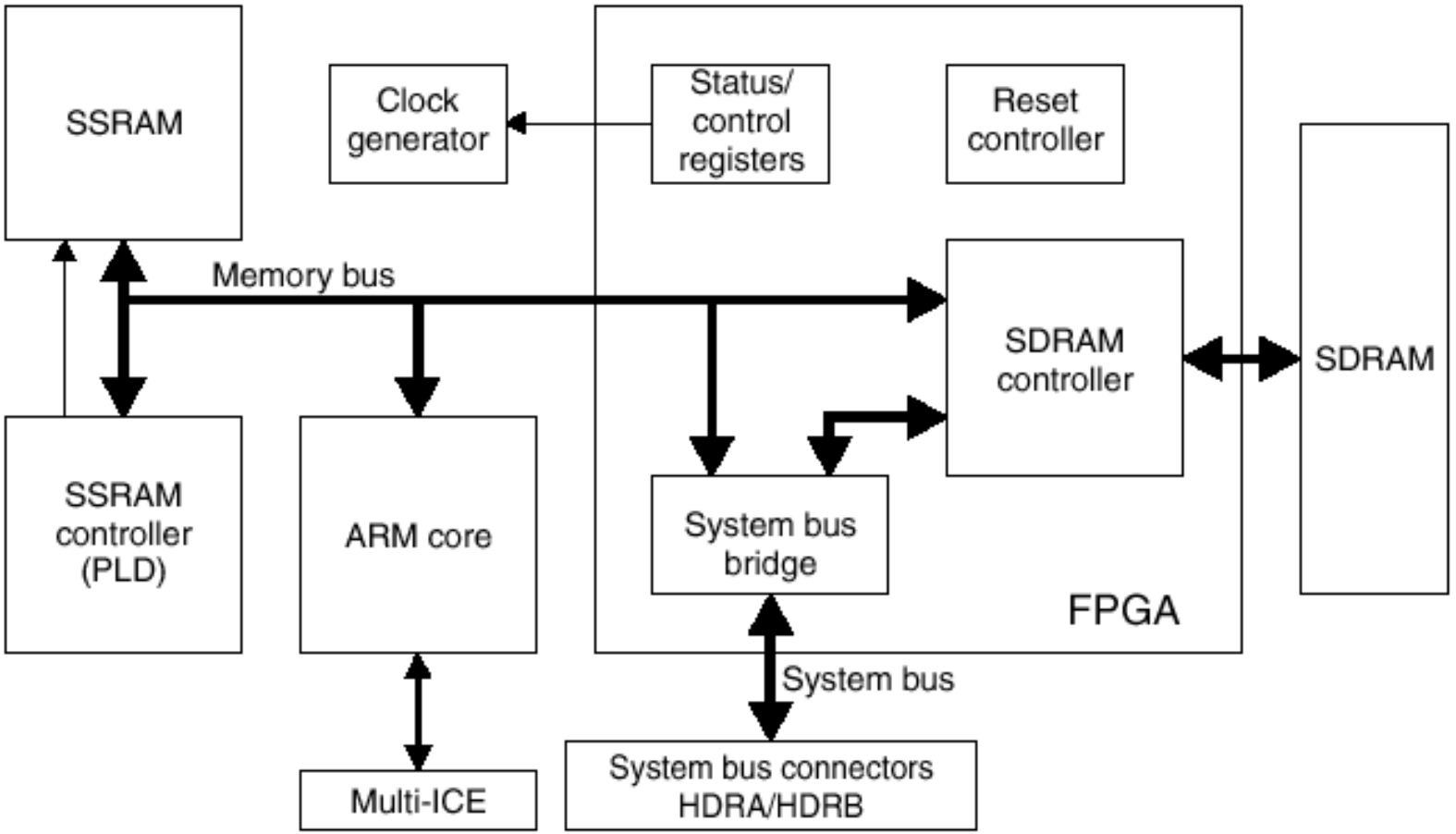
- ARM Integrator/Core Module (CM)
  - CM provides ARM core personality.
  - CM could be used as a standalone development system without AP.
  - CM could be mounted onto AP as a system core.
  - CM could be integrated into a 3rd-party development or ASIC prototyping system.

ARM Integrator Core Module/CM Resources

- ARM Integrator/CM Features (CM7TDMI):
  - ARM7TDMI microprocessor core.
  - core module controller FPGA :
    - SDRAM controller
    - System bus bridge
    - Reset controller
    - Interrupt controller
  - Supports 16MB~256MB pc66/pc100 168pin SDRAM.
  - Supports 256/512 KB SSRAM
  - Multi-ICE, logic analyzer, and optional trace connectors.

ARM Integrator Core Module/CM Resources

- ARM Integrator/CM Architecture:



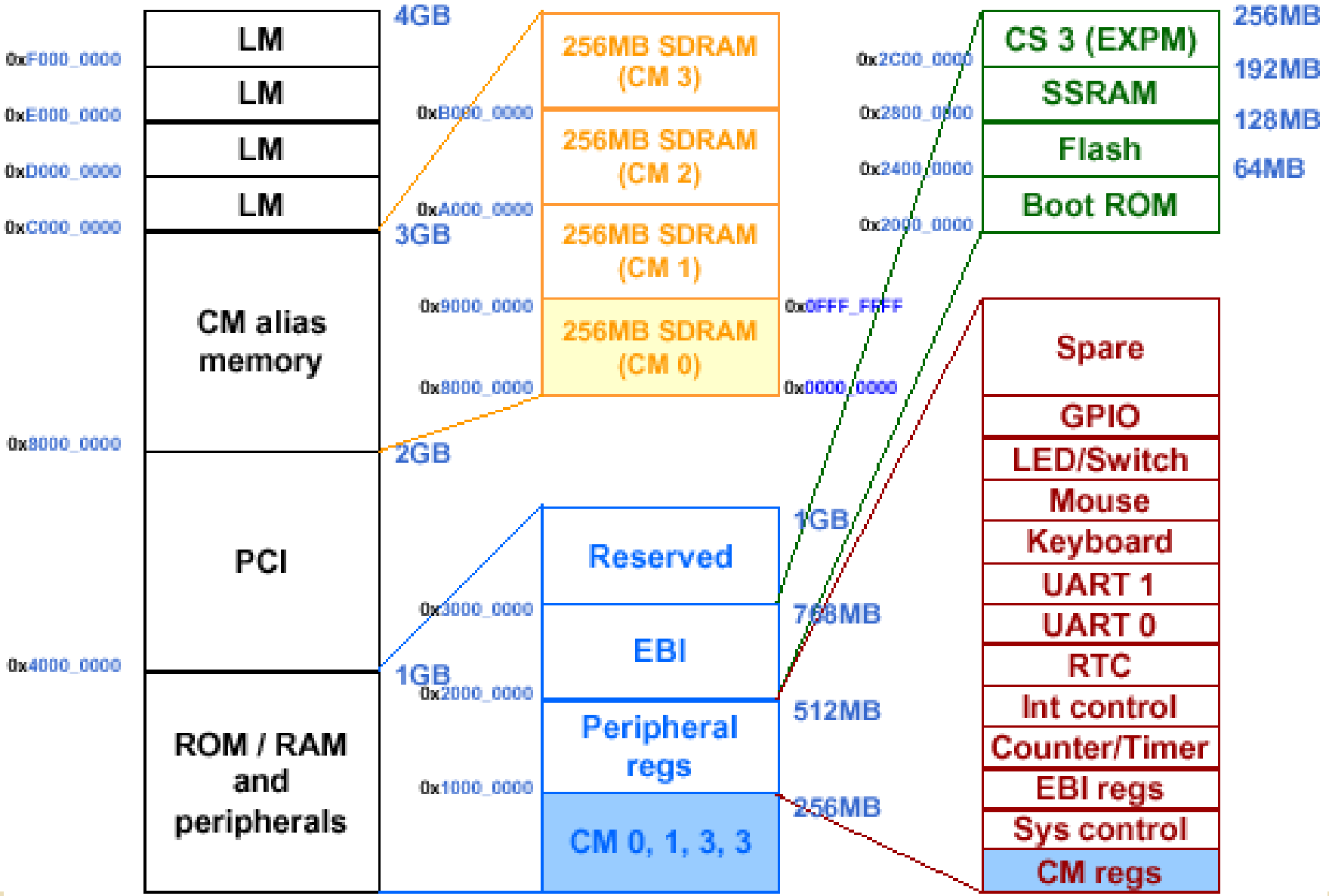
ARM Integrator Core Module/CM Resources

- Core Module FPGA
  - SDRAM controller
    - Supports for DIMMs from 16MB to 256MB.
  - Reset controller
    - Initializes the core.
    - Process resets from different sources.
  - Status and configuration space
    - Provides processor information.
    - CM oscillator setup.
    - Interrupt control for the processor debug communications channel.
  - System bus bridge
    - Provides Interface between the memory bus on the CM and the system bus on the AP.

About ARM Hardware Development Environment

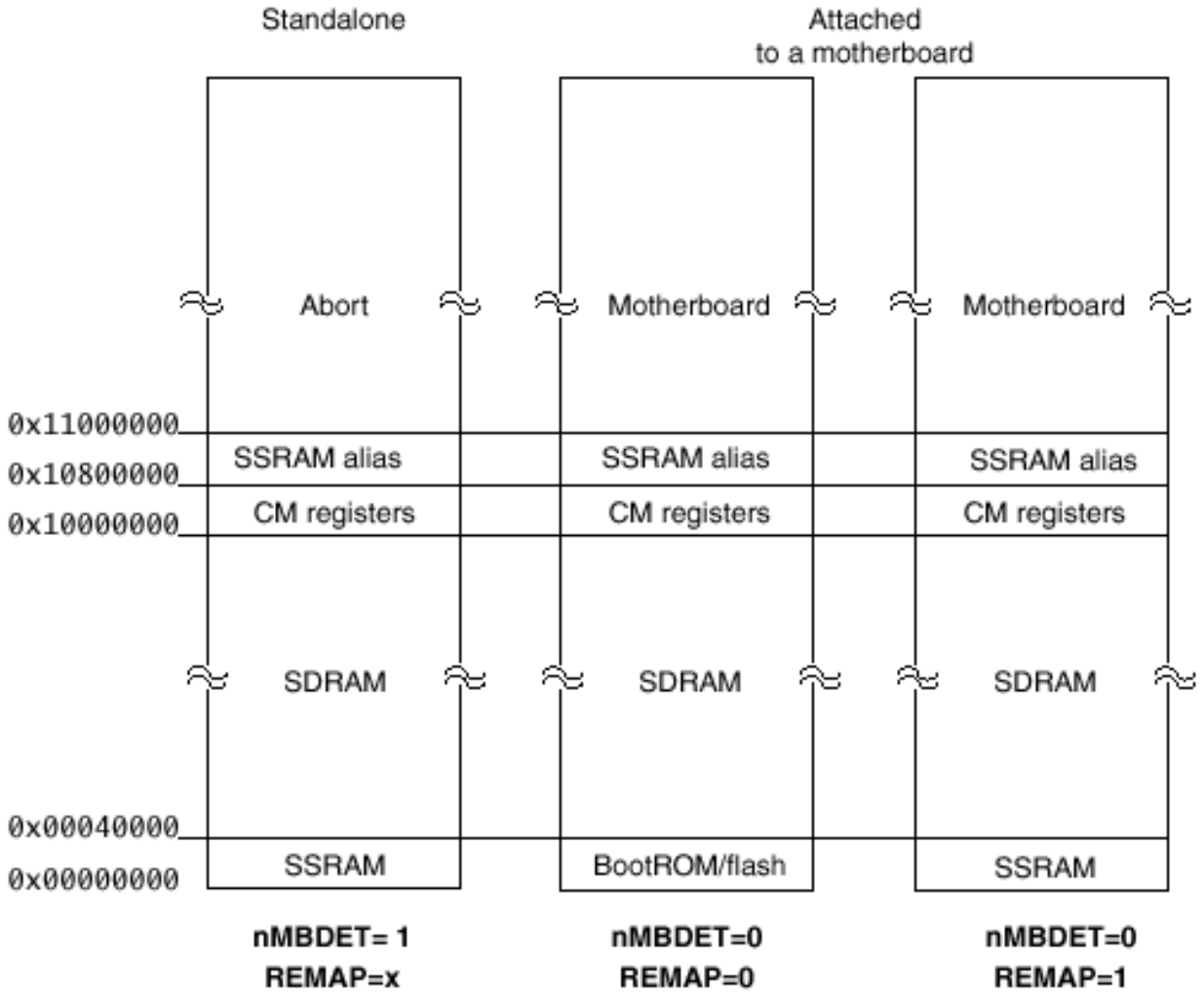
System Memory Map

System Memory Map



System Memory Map

- Core Module memory map



About ARM Hardware Development Environment

Running Images on the ARM Integrator

Running Images on the ARM Integrator

- Setting up the Integrator hardware:
  - **1.** Connect the Multi-ICE to the CM
  - **2.** Connect the null-modem cable to the host PC's serial port
  - **3.** Connect the Integrator/AP's power to a turned-off power plug.
    - **3.1** Turn on the the power plug after the system is properly connected.
  - **4.** Press the case's power switch to stand-by the Integrator.
    - **4.1** The power switch on the case is connected to the *stand-by* switch on the Integrator/AP board.
    - **4.2** The *stand-by LED* is red before pressing the power switch on the case.
    - **4.3** After the power switch has been pressed, the LEDs on the case's panel would be turned on.

Running Images on the ARM Integrator

- Setting up connection to the Integrator:
  - **5.** Start *Multi-ICE Server* from the Program Menu in the windows.
    - **5.1** Press the *Auto-Configure* button.
    - **5.2** *Multi-ICE Server* would detect the hardware after auto-configuration. The detected result would be shown in the program window.

Running Images on the Integrator

- Running the AXD:
  - **6.** Start *AXD*.
  - **7.** *Options>>Configure Target*
    - **7.1** Select the *MultiICE* and press *OK*.
  - **8.** Now AXD is operating using the real Integrator hardware system instead of using the ARMulator.

Running Images on the ARM Integrator

- Turning off the Integrator
  - **1.** Press the power switch on the case.
  - **2.** Turn off the power plug's power.
    - **2.1** The *stand-by LED* would remain red for several seconds after turning off the power plug's power.
    - **2.2** After the *stand-by LED* goes off, the Integrator is turned off.

Examples

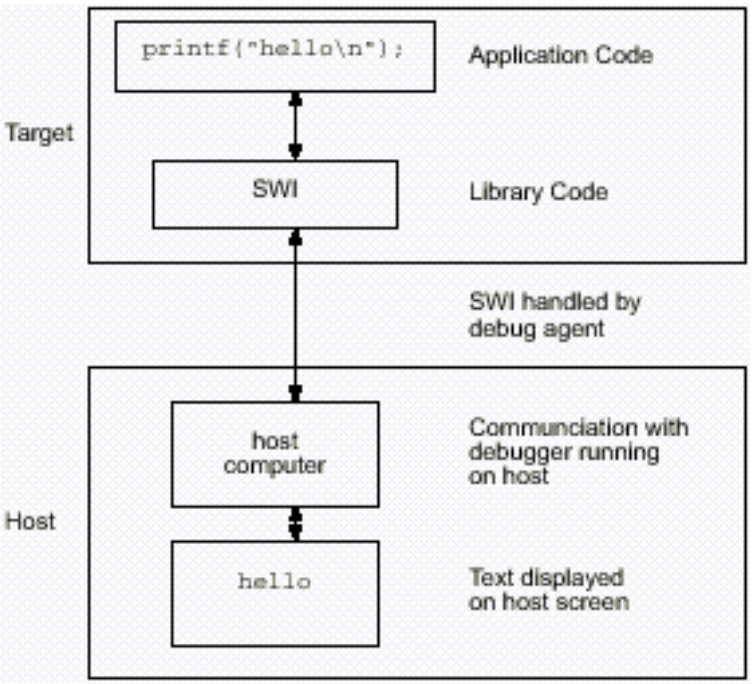
- Part A: Semihosting
- Part B: Memory Usage
- Part C: Timer/Interrupt

Examples

Part A: Semihosting

Part A: 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



Part A: Semihosting

- 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.

Part A: Semihosting

- 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.

Part A: Semihosting

- 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.

SWI	Description
SYS_OPEN (0x01) on page 5-12	Open a file on the host
SYS_CLOSE (0x02) on page 5-14	Close a file on the host
SYS_WRITEC (0x03) on page 5-14	Write a character to the console
SYS_WRITE0 (0x04) on page 5-14	Write a null-terminated string to the console
SYS_WRITE (0x05) on page 5-15	Write to a file on the host
SYS_READ (0x06) on page 5-16	Read the contents of a file into a buffer
SYS_READC (0x07) on page 5-17	Read a byte from the console
SYS_ISERROR (0x08) on page 5-17	Determine if a return code is an error

Example

Part B:
Memory Usage

Part B: Memory Usage

- This part contains only one example. It does the following tasks:
  - Backup the data in the SSRAM at locations 0x30000 to 0x38000 range to the SDRAM at locations 0x80000000 to 0x80008000.
  - Write values to the SSRAM at locations 0x30000 to 0x38000.
  - Verify the values in the SSRAM at locations 0x30000 to 0x38000.
  - Restore the backuped data back to their original locations.

Part B: Memory Usage

- Core Module Memory Map:
  - The boot ROM on the AP and local SSRAM on the CM share the same location within the system memory map.
  - The access to which memory at the boot ROM/SSRAM overlapped location is determined by the REMAP bit and the nMBDET bit.
    - nMBDET : 1st (lsb) bit of the CM_CTRL(0x1000000C) register.
      - If the CM is attached to the AP, nMBDET=0; else nMBDET=1.
    - REMAP : 2nd bit of the CM_CTRL(0x1000000C) register. Only has effect when nMBDET=0.
      - REMAP =0, 0x00000000 ~ 0x0003FFFF are directed to the boot ROM.
      - REMAP =1, 0x00000000 ~ 0x0003FFFF are mapped to the CM's local SSRAM.

Part B: Memory Usage

- CM_CTRL (0x1000000C)

Bits	Name	Access	Function
31:6	Reserved		
5	BIGEND	R/W	0 = little-endian (default) 1 = big-endian
4	Reserved		
3	RESET	W	1 = Reset the CM
2	REMAP	R/W	0 = access Boot ROM 1 = access SSRAM
1	nMBDET	R	0 = mounted on motherboard 1 = stand alone
0	LED	R/W	0 = LED on 1 = LED off

Part B: Memory Usage

- *SSRAM.C*

```
#include <stdio.h>

int main(void){
        unsigned int        CM_CTRL_ADDR = 0x1000000C;
        unsigned int        SSRAM_ADDR = 0x00000000;    // CM's SSRAM ranges 0x0 ~
0x0003FFFF

        unsigned int        *CM_CTRL_PTR, *SSRAM_PTR, *SDRAM_PTR;

        unsigned int i;
        int SSRAM_test_error = 0;

        CM_CTRL_PTR = (unsigned int *) CM_CTRL_ADDR;
        SSRAM_PTR = (unsigned int *) SSRAM_ADDR;

        // Memory Remap to SSRAM
        *CM_CTRL_PTR =0x4;

        printf("SSRAM Write Test\n");
```

Part B: Memory Usage

- *SSRAM.C (continued)*

```
printf("Press any key to start SSRAM test!!\n");

getchar();
*CM_CTRL_PTR =0x5;    // turn  off the MISC LED on the CM

printf("Backup SSRAM data from 0x0000 to 0x8000 to SDRAM at 0x80000000\n");
for(i=0;i<0x8000;i+=4)
{
        SDRAM_PTR = (unsigned int *)(i+0x80000000);
        SSRAM_PTR = (unsigned int *)(i+0x30000);
        *SDRAM_PTR = *SSRAM_PTR;
}

printf("Writing...\n");
for(i=0;i<0x8000;i+=4)
{
        SSRAM_PTR = (unsigned int *)(i+0x30000);
        *SSRAM_PTR = i;
}
```

Part B: Memory Usage

- *SSRAM.C (continued)*

```
printf("Verifying...\n");
for(i=0x0;i<0x8000;i+=4)
{
        SSRAM_PTR = (unsigned int *) (i+0x30000);
        if(*SSRAM_PTR != i)
        {
                printf("SSRAM W/R test error!!\n");
                printf("Error address>> %x\n",i);
                SSRAM_test_error = 1;
                getchar();
        }
}

if(SSRAM_test_error != 1)
        printf("SSRAM test passed!!\n");

*CM_CTRL_PTR =0x4;
printf("SSRAM test finished.\n");
```

Part B: Memory Usage

- *SSRAM.C (continued)*

```
        printf("Restore original SSRAM data from 0x00000000 to 0x80008000 to SSRAM at
0x8000\n");
        for(i=0;i<0x8000;i+=4)
        {
                SDRAM_PTR = (unsigned int *)(i+0x80000000);
                SSRAM_PTR = (unsigned int *) (i+0x30000);
                *SSRAM_PTR = *SDRAM_PTR;
        }

        return 0;
}
```

Part B: Memory Usage

- Building *SSRAM.c*
  - **1.** Create a new *ARM Executable Image* project.
  - **2.** Add *SSRAM.c* to the project
  - **3.** Make the project
  - **4.** Run the project

Examples

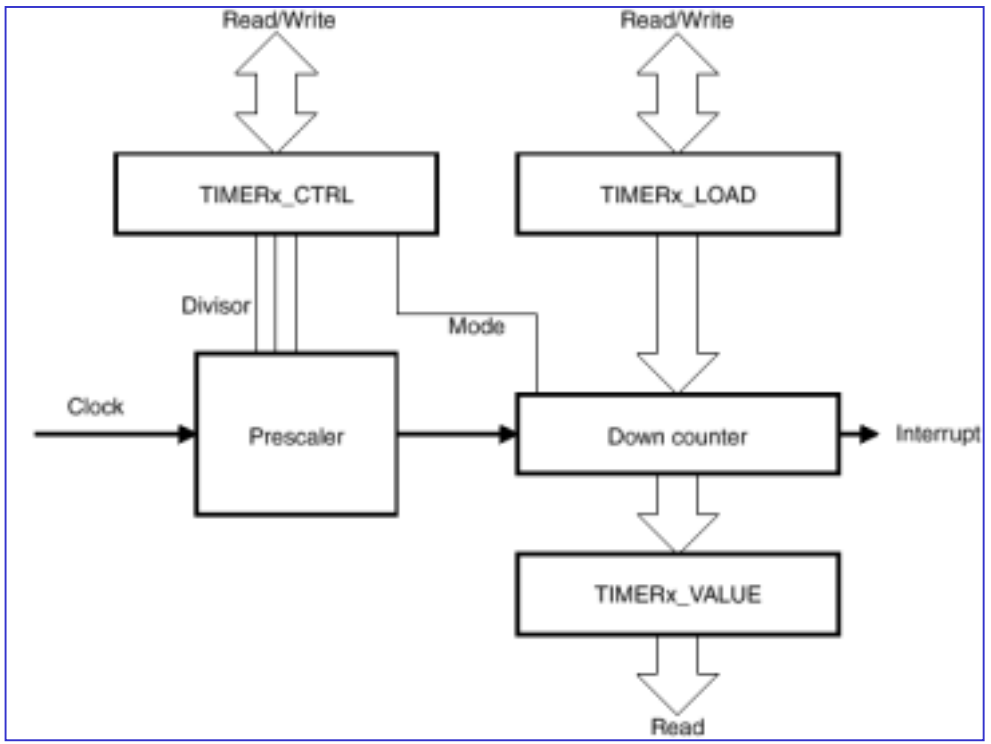
Part C: Timer/Interrupt

Part C: Timer/Interrupt

- This part contains 1 example:
  - Using Timer/Interrupts with pointers.(with no uHAL API)
    - This example installs a timer interrupt and it's handler to flash the LED.
- Observation key points
  - Check the Timer/Interrupt related registers values, see how the change.
  - Observe how interrupt is handled.

Part C: Timer/Interrupt

- About Counter/Timers
  - There are 3 counter/timers on an ARM Integrator AP.
  - Each counter/timer generates an IRQ when it reaches 0.
  - Each counter/timer has:
    - A 16-bit down counter with selectable prescale
    - A load register
    - A control register



Part C: Timer/Interrupt

- Counter/Timer Registers
  - These registers control the 3 counter/timers on the Integrator AP board.
  - Each timer has the following registers.
    - TIMERX_LOAD: a 16-bit R/W register which is the initial value in free running mode, or reloads each time the counter value reaches 0 in periodic mode.
    - TIMERX_VALUE: a 16-bit R register which contains the current value of the timer.
    - TIMERX_CTRL: an 8-bit R/W register that controls the associated counter/timer operations.
    - TIMERX_CLR: a write only location which clears the timer's interrupt.

Part C: Timer/Interrupt

- Counter Timer Registers

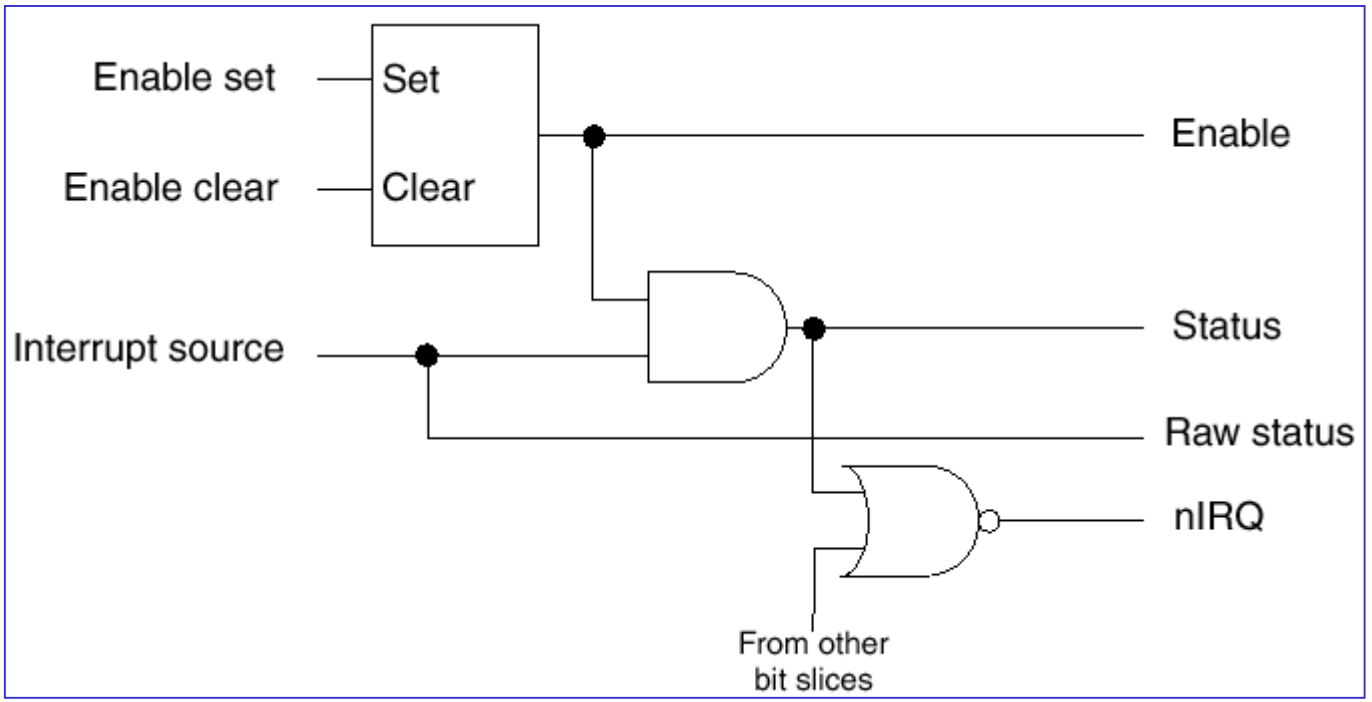
Address	Name	Type	Size	Function
0x13000000	TIMER0_LOAD	R/W	16	Timer0 load register
0x13000004	TIMER0_VALUE	R	16	Timer0 current value reg
0x13000008	TIMER0_CTRL	R/W	8	Timer0 control register
0x1300000C	TIMER0_CLR	W	1	Timer0 clear register

- Timer Control Register

Bits	Name	Function
7	ENABLE	Timer enable: 0=disable; 1=enable.
6	MODE	Timer mode: 0=free running; 1=periodic
5:4	unused	Unused, always 0
3:2	PRESCALE	Prescale divisor: 00=none; 01 = div by 16 10=div by 256; 11 = undefined
1:0	Unused	Unused,always 0

Part C: Timer/Interrupt

- About Interrupt Controller
  - Implemented in the system controller FPGA.
  - Provides interrupt handling for up to 4 processors.
  - There's a 22-bit IRQ and FIQ controller for each processor.



- IRQ Registers
  - The registers control each processor's interrupt handler on the Integrator AP board.
  - Each IRQ has following registers:
    - IRQX_STATUS: a 22-bit register representing the current masked IRQ status.
    - IRQX_RAWSTAT: a 22-bit register representing the raw IRQ status.
    - IRQX_ENABLESET: a 22-bit location used to set bits in the enable register.
    - IRQX_ENABLECLR: a 22-bit location used to clear bits in the enable register.

Part C: Timer/Interrupt

- IRQ Registers

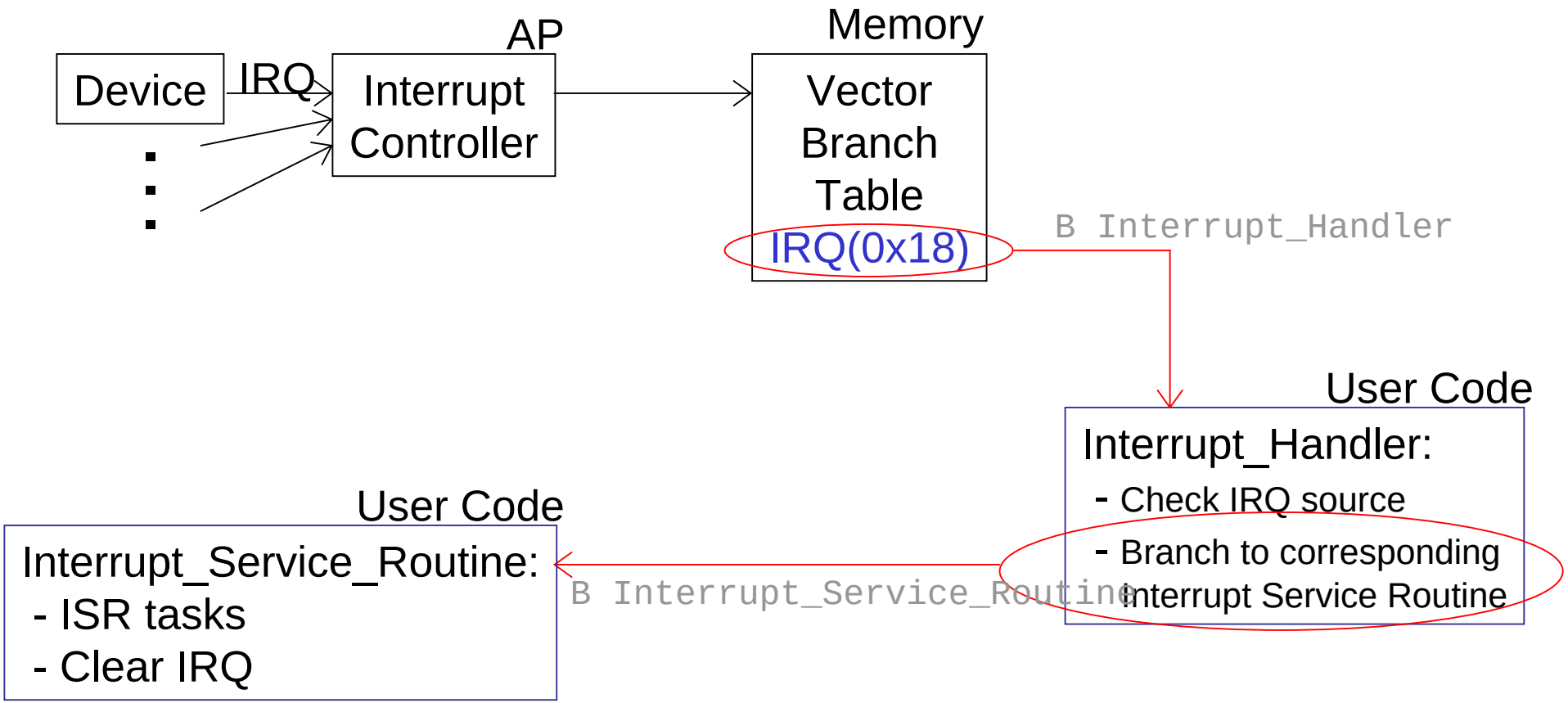
Address	Name	Type	Size	Function
0x14000000	IRQ0_STATUS	R	22	IRQ0 status
0x14000004	IRQ0_RAWSTAT	R	22	IRQ0 IRQ status
0x14000008	IRQ0_ENABLESET	R/W	22	IRQ0 enable set
0x1400000C	IRQ0_ENABLECLR	W	22	IRQ0 enable clear

- IRQ Registers bit assignments

Bit	**Name**	**Function**
0	SOFTINT	Software interrupt
5	TIMERINT0	Counter/Timer interrupt
6	TIMERINT1	Counter/Timer interrupt
7	TIMERINT2	Counter/Timer interrupt

Part C: Timer/Interrupt

- How Interrupt works:



Part C: Timer/Interrupt

- Timer/Interrupt example without uHAL:
- Several important functions are used in this example:
  - Install_Handler: This function install the IRQ handler at the branch vector table at 0x18.
  - myIRQHandler: This is the user's IRQ handler. It performs the timer ISR in this example.
  - enableIRQ: The IRQ enable bit in the CPSR is set to enable IRQ.
  - LoadTimer,WriteTimerCtrl,ReadTimer,ClearTimer: Timer related functions.

Part C: Timer/Interrupt

- *Timer_IRQ.c*

```
#include <stdio.h>

unsigned Install_Handler( unsigned routine, unsigned *vector )
{
        unsigned vec, oldvec;
        vec = ((routine - (unsigned)vector - 0x8) >> 2 );
                        /* --> routine is the pointer point to the IRQ handler. */
                        /* --> shift right 2 is for address word aligned. */
                        /* --> subtract 8 is due to the pipeline */
                        /* since PC will be fetching the 2nd instruction */
                        /* after the instruction currently being executed. */

        vec = 0xea000000 | vec;/* to implement the instruction B <address> */
                         /* 0xea is the Branch operation */
        oldvec = *vector;       /* the IRQ address or FIQ address */
        *vector = vec;          /* the contents of IRQ address is now the branch instruction
*/
        return (oldvec);
}
```

Part C: Timer/Interrupt

- *Timer_IRQ.c continued*

```
__irq void myIRQHandler (void)
{
        printf("\nFrom IRQ Handler>>HIHI!!\n");
        ClearTimer();        /* Clear the timer's IRQ */
}

// this function is used to set the I bit in CPSR
__inline void enable_IRQ(void)
{
  int tmp;
  __asm
  {
    MRS tmp, CPSR
    BIC tmp, tmp, #0x80
    MSR CPSR_c, tmp
  }
}
```

Part C: Timer/Interrupt

- *Timer_IRQ.c continued*

```
void d2b(int d_number, int array_len, int *b_number) {
 int len; /*array index*/                    /* This function transform data into binary digits */
 int temp=1;

 for (len=0;len<array_len;len++) {
    if (temp&d_number) b_number[len]=1;
    else b_number[len]=0;
    d_number=d_number>>1;
 }
}

void printB(int d_number, int array_len, int*b_number){
            int i;                                  /* This function prints the binary digits */
            for(i=(array_len-1);i>=0;i--){
                        printf("%d",b_number[i]);
                        if ( i%8==0 && i!=0)
            printf("_");
            }
            printf("\n");
}
```

Part C: Timer/Interrupt

- *Timer_IRQ.c continued*

```
void LoadTimer(int loadvalue){
        int TIMER0_LOAD_ADDR = 0x13000000;
        int *TIMER0_LOAD;

        TIMER0_LOAD = (int *)TIMER0_LOAD_ADDR;

        *TIMER0_LOAD = loadvalue;
        printf("Timer Message>>> Timer0 loaded!!\n");
}

int ReadTimer(void){
        int TIMER0_VALUE_ADDR = 0x13000004;
        int *TIMER0_VALUE;

        TIMER0_VALUE = (int *)TIMER0_VALUE_ADDR;

        printf("Timer Message>>> Timer0 value aquired!!\n");
        return *TIMER0_VALUE;
}
```

Part C: Timer/Interrupt

- *Timer_IRQ.c continued*

```
void WriteTimerCtrl(int writevalue){
        int TIMER0_CTRL_ADDR = 0x13000008;
        int *TIMER0_CTRL;

        TIMER0_CTRL = (int *)TIMER0_CTRL_ADDR;

        *TIMER0_CTRL = writevalue;
        printf("Timer Message>>> Timer0 control register changed!!\n");
}

void ClearTimer(void){
        int TIMER0_CLEAR_ADDR = 0x1300000C;
        int *TIMER0_CLEAR;

        TIMER0_CLEAR = (int *)TIMER0_CLEAR_ADDR;

        *TIMER0_CLEAR = 1;
        printf("Timer Message>>> Timer0 cleared!!\n");
}
```

Part C: Timer/Interrupt

- *Timer_IRQ.c continued*

```
int main(void) {
        int IRQ0_STATUS_ADDR = 0x14000000;
        int IRQ0_RAWSTAT_ADDR = 0x14000004;
        int IRQ0_ENABLESET_ADDR = 0x14000008;
        int IRQ0_ENABLECLR_ADDR = 0x1400000C;

        int *IRQ0_STATUS, *IRQ0_RAWSTAT, *IRQ0_ENABLESET, *IRQ0_ENABLECLR;

        int b_num[22];
        int i;
        unsigned *irqvec = (unsigned *)0x18;

        Install_Handler( (unsigned)myIRQHandler, irqvec ); /* Install user's IRQ Handler */

        enable_IRQ();                                      /* DR added - ENABLE IRQs */

        IRQ0_STATUS = (int *) IRQ0_STATUS_ADDR;
        IRQ0_RAWSTAT = (int *) IRQ0_RAWSTAT_ADDR;
        IRQ0_ENABLESET = (int *) IRQ0_ENABLESET_ADDR;
        IRQ0_ENABLECLR = (int *) IRQ0_ENABLECLR_ADDR;
```

Part C: Timer/Interrupt

- *Timer_IRQ.c continued*

```
*IRQ0_ENABLESET = 0x0021;

d2b(*IRQ0_STATUS,22,b_num);
printf("IRQ0_STATUS: ");
printB(*IRQ0_STATUS,22,b_num);

d2b(*IRQ0_RAWSTAT,22,b_num);
printf("IRQ0_RAWSTAT: ");
printB(*IRQ0_RAWSTAT,22,b_num);

d2b(*IRQ0_ENABLESET,22,b_num);
printf("IRQ0_ENABLESET: ");
printB(*IRQ0_ENABLESET,22,b_num);

d2b(*IRQ0_ENABLECLR,22,b_num);
printf("IRQ0_ENABLECLR: ");
printB(*IRQ0_ENABLECLR,22,b_num);
```

Part C: Timer/Interrupt

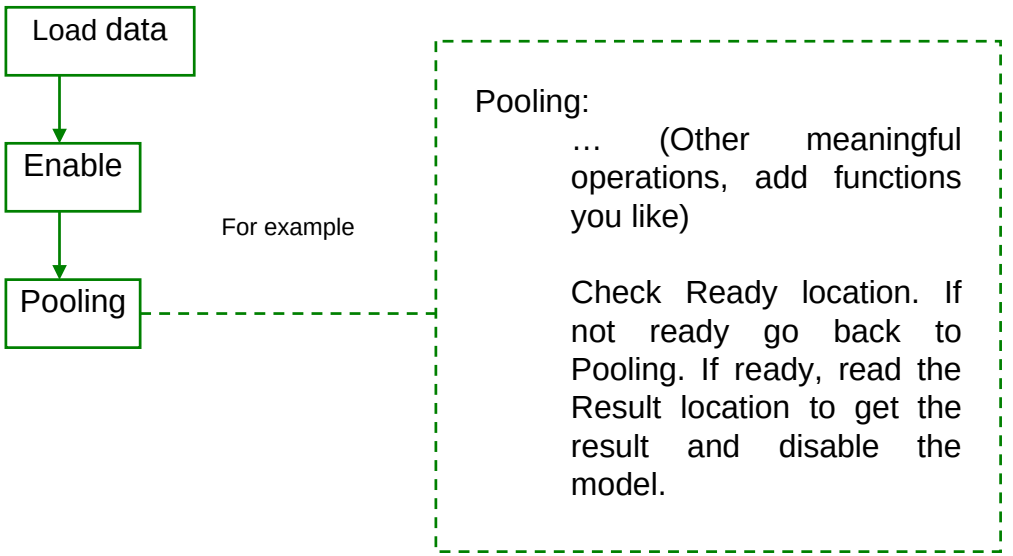
- *Timer_IRQ.c continued*

```
        LoadTimer(64);                  /* set Timer0 reload value to 64 */
        WriteTimerCtrl(0xC4);           /* Enable the timer */

        // wait for a while
        for(i=0;i<1000000000;i++)
        {
                ;
        }
        printf("\nEND\n");
        return 0;
}
```

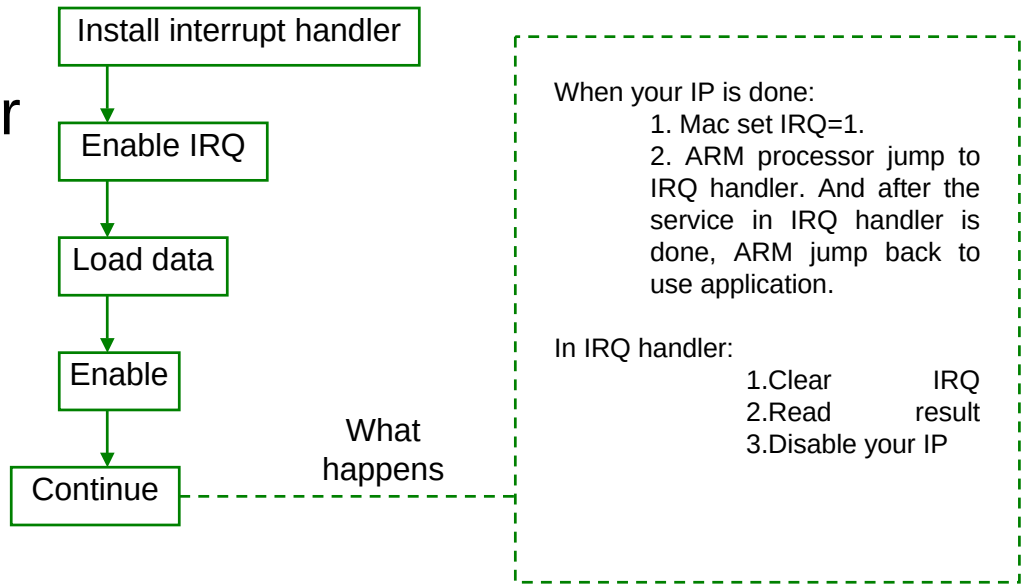
Polling & Interrupt(1/2)

1. Load data to register in your IP.
2. Write 1 to Enable register to enable your IP.
3. Pooling. Read the Ready location periodically. If ready = 1, read Result location to get the result.
4. Write 0 to Enable register to disable your IP.



Polling & Interrupt(2/2)

1. Install interrupt handler.
2. Enable IRQ.
3. Load data to data register in your IP.
4. Write 1 to Enable register to enable your IP.
5. Continue. Do any operations as you want.



Lab#2 Exercise

- Modify the memory usage example. Use timer to count the total access time of several data accessing to SSRAM and SDRAM. Compare the performance between using SSRAM and SDRAM.
- SSRAM is faster than SDRAM, therefore using SSRAM should have better performance over using SDRAM.

Reference Topic & Related Documents

Reference Topic & Related Documents

- Integrator ASIC Platform [DUI_0098B_AP_UG]
- System Memory Map [DUI_0098B_AP_UG 4.1]
- Counter/Timer [DUI_0098B_AP_UG 3.7, 4.6]
- Interrupt [DUI_0098B_AP_UG 3.6, 4.8]
- LEDs [DUI_0098B_AP_UG 4.5]
- Core Module [DUI_0126B_CM7TDMI]
- Core Module Registers[DUI_0126B_CM7TDMI 4.2]
- Core Module Memory Organization [DUI_0126B_CM7TDMI 4.1]
- SSRAM [DUI_0126B_CM7TDMI 3.2]
- SDRAM [DUI_0126B_CM7TDMI 3.4]
- SWI Interface [ADS_DebugTargetGuide 5.1.1]
- SWI Handling [ADS_DeveloperGuide 5.4]
- Semihosting [ADS_DebugTargetGuide 5]
- Building Semihosted application [ADS_CompilerLinkerUtil 4.2]
- Semihosting directly dependent functions [ADS_CompilerLinkerUtil Table4-1]
- Semihosting indirectly dependent functions [ADS_CompilerLinkerUtil Table4-2]
- I/O supported functions using semihosting SWI [ADS_CompilerLinkerUtil Table4-13]
- uHAL API [DUI_0102D_AFS_REF 2] [DUI_0136A_AFS_USER 2]