# Appendix A

# Setting Up ARMulator

The ARMulator can simulate various ARM cores and ARM development boards. However, the default settings of the ARMulator is different from that of an Integrator. Before you start doing the exercises, you have to make the ARMulator behave like the development board you are using. You have to modify the source code of high-level modules for the ARMulator and set a few processor core parameters. The following sections demonstrate how to set up an ARMulator as an Integrator/CM7TDMI mounting on an Integrator/AP. If you are using other development board or other core modules, refer to the user guides of these components for more detailed information.

## A.1 Configure Processor Parameters

1. Start AXD, and then select **Options** → **Configure Target** from menu. Choose **ARMUL** as target and then press **Configure** button on the left.

2. Configure ARMulator as below.

   - Processor: ARM7TDMI
   - Clock: Emulated 10MHz[1]
   - Floating Point Emulation: No
   - Memory Map File: *select a map file*

   The default setting for the ARMulator is to model a system with 4GB of zero wait state 32bit memory. However, real systems are unlikely to have such an ideal memory system. An alternative memory model can be defined by memory-map file. The map file contains one or more entries, each entry describes the type and speed of a memory block in the system. Check [1] for an in-depth view about memory-map.

   The syntax of memory mapping is shown in the entry below:

   ```
   start size name width access read-times write-times
   ```

[1] Do NOT forget to include the unit "MHz" when defining the clock rate. The default unit for clock rate is Hz, so a plain 10 means to operate at 10Hz only. Real Integrator/AP supports operating frequency ranging from 3MHz to 50MHz with a 0.5MHz step.

The meaning of each column is:

**start** memory start address in hexadecimal

**size** memory size in hexadecimal in byte

**name** descriptive name

**width** data bus width in byte

**access** access right
(**r** for read only, **w** for write only, **rw** for read-write, **-** for no access)

**read-times** sequential/non-sequential read access times in nano-second

**write-times** sequential/non-sequential write access times in nano-second

An example memory map with 32kB SRAM followed by 32kB ROM and 256MB DRAM is listed below. The DRAM has a redced size of 0xFFF0000 since the lower 64kB is overlapped by the SRAM and ROM.

```
00000000 8000    SRAM  4 RW 1/1     1/1
00008000 8000    ROM   2 R  100/100 100/100
00010000 FFF0000 DRAM  2 RW 150/100 150/100
```

## A.2 Modify High-level Modules

The high-level module is a module written in C language that can be plugged into the ARMulator to simulate the functionality of the hardware. Since the memory mapping of timer and interrupt controller is inconsistent between the ARMulator and Integrator, you have to make the following modifications correcting the differences. Check [2] for more details on the peripherals of Integrator/AP.

1. Re-define peripheral I/O mapping in high-level model.

   Edit **$ADS_ROOT/ARMulate/armulext/timer.c**. Find the following code segment near line 129. Originally, the base address is 0x0A000000 and the mamory mapping range is 0x40.

   ```
   err = ARMulif_RBusRange(&state->coredesc, state->hostif,
             ToolConf_FlatChild(config, (tag_t)"RANGE"),
             &state->my_bpar,
             0x0A000000,0x40,"");
   ```

   Change the timer's I/O base address and range to **0x13000000** and **0x300**, respectively.

   ```
   err = ARMulif_RBusRange(&state->coredesc, state->hostif,
             ToolConf_FlatChild(config, (tag_t)"RANGE"),
             &state->my_bpar,
             0x13000000,0x300,"");
   ```

2. Go to line near 640. Rewrite the code calculating the offset of an access event. The original code use bit masking to get the address offset indicating a specific event. This method requires that the masked bits of base address to be zero. For example, the base address 0x13000100 causes an error since 0x13000100 & 0x000003FF is an non-zero value 0x00000100. An access to 0x13000104 will get 0x00000104 instead of its real offset 0x00000004. In addition, you have to make extra modifications to the mask whenever the memory mapping range is changed.

```
unsigned long maskedAddress = addr & 0x000003FF;
```

The modified version use substraction to get the address offset. This method is more flexible since the limit on base address is removed.

```
unsigned long maskedAddress =
                  ((unsigned long) addr - (0x13000000));
```

3. Go to line near 650, where you will see a **switch(maskedAddress)** statement. Edit each case item so the offset values match that in Table A.1.

```
case 0x100: /* TIMER1 Load */
    *word = ts->timer1.TimerLoad & 0x0000FFFF;
    break;
case 0x104: /* TIMER1 Value */
    *word = ValueRegR(ts, &ts->timer1);
    break;
case 0x108: /* TIMER1 Control*/
    *word = ts->timer1.TimerControl;
    break;
...
```

There are three timers on Integrator/AP, but timer.c only defines two timers. You can selectively map the timers to any two of the three timers on Integrator. If all three timers are needed to complete the simulation, install an additional high-level module for the third timer. Throughout the lab exercises, we will map the two timers to TIMER1 at 0x13000100 and TIMER2 at 0x13000200 and leave TIMER0 unmapped. The reason to leave TIMER0 unmapped is that the $\mu$C/OS-II used in Lab RTOS requires TIMER1 and TIMER2 but not TIMER0.

4. Re-compile and install the high-level model for timer. In command line, go to **$ADS_ROOT/ARMulate/armulext/timer.b/intelrel/** and run **nmake**[2]. Copy the compiled timer.dll to directory **$ADS_ROOT/Bin**.

5. Edit corresponding section in **$ADS_ROOT/Bin/peripherals.ami** and **$ADS_ROOT/Bin/peripherals.dsc** for timer module. It is important to set the interrupt numbers **IntOne** for the first timer and **IntTwo** for the second timer. The interrupt numbers are listed in Table A.3.

   Re-define timer in peripherals.ami:

[2]You must have Visual C++ installed and environment variables correctly set.

```
{Default_Timer=Timer
  Waits=0
  Range:Base=0x13000000
  CLK=20000000
  IntOne=6
  IntTwo=7
}
```

Re-define timer in peripherals.dsc:

```
{Timer
  meta_sordi_dll=Timer
  META_GUI_INTERFACE=Timer
  Waits=0
  Range:Base=0x13000000
  CLK=20000000
  TicMCCfg=2
  IntOne=6
  IntTwo=7
}
{No_Timer=Nothing
  META_GUI_INTERFACE=Timer
}
```

6. Repeat step 1–5 for interrupt controller

   Edit **$ADS_ROOT/ARMulate/armulext/intc.c**, the source file of interrupt controller. The intc.c only defines one interrupt controller while Integrator/AP has four. Map the only interrupt controller to 0x14000000 for IRQ0 according to Table A.2. Unlike the timer, you must map the interrupt controller to IRQ0 since it is associated with the first Core Module mounted on the development board. IRQ1 to IRQ3 are associated with the second to fourth Core Modules on the development board.

7. Finally, restart AXD so the changes take effect.

| Address | Name | Type | Size | Function |
|---|---|---|---|---|
| 0x1300_0000 | TIMER0_LOAD | RW | 16 | Timer0 load |
| 0x1300_0004 | TIMER0_VALUE | R | 16 | Timer0 current value |
| 0x1300_0008 | TIMER0_CTRL | RW | 16 | Timer0 control |
| 0x1300_000C | TIMER0_CLR | W | 1 | Timer0 clear |
| 0x1300_0100 | TIMER1_LOAD | RW | 16 | Timer1 load |
| 0x1300_0104 | TIMER1_VALUE | R | 16 | Timer1 current value |
| 0x1300_0108 | TIMER1_CTRL | RW | 16 | Timer1 control |
| 0x1300_010C | TIMER1_CLR | W | 1 | Timer1 clear |
| 0x1300_0200 | TIMER2_LOAD | RW | 16 | Timer2 load |
| 0x1300_0204 | TIMER2_VALUE | R | 16 | Timer2 current value |
| 0x1300_0208 | TIMER2_CTRL | RW | 16 | Timer2 control |
| 0x1300_020C | TIMER2_CLR | W | 1 | Timer2 clear |

Table A.1: Memory map table for timers on Integrator/AP

| Address | Name | Type | Size | Function |
|---|---|---|---|---|
| 0x1400_0000 | IRQ0_STATUS | R | 22 | IntCtrl0 status |
| 0x1400_0004 | IRQ0_RAWSTAT | R | 22 | IntCtrl0 raw status |
| 0x1400_0008 | IRQ0_ENABLESET | RW | 22 | IntCtrl0 enable set |
| 0x1400_000C | IRQ0_ENABLECLR | W | 22 | IntCtrl0 enable clear |
| 0x1400_0040 | IRQ1_STATUS | R | 22 | IntCtrl1 status |
| 0x1400_0044 | IRQ1_RAWSTAT | R | 22 | IntCtrl1 raw status |
| 0x1400_0048 | IRQ1_ENABLESET | RW | 22 | IntCtrl1 enable set |
| 0x1400_004C | IRQ1_ENABLECLR | W | 22 | IntCtrl1 enable clear |
| 0x1400_0080 | IRQ2_STATUS | R | 22 | IntCtrl2 status |
| 0x1400_0084 | IRQ2_RAWSTAT | R | 22 | IntCtrl2 raw status |
| 0x1400_0088 | IRQ2_ENABLESET | RW | 22 | IntCtrl2 enable set |
| 0x1400_008C | IRQ2_ENABLECLR | W | 22 | IntCtrl2 enable clear |
| 0x1400_00C0 | IRQ3_STATUS | R | 22 | IntCtrl3 status |
| 0x1400_00C4 | IRQ3_RAWSTAT | R | 22 | IntCtrl3 raw status |
| 0x1400_00C8 | IRQ3_ENABLESET | RW | 22 | IntCtrl3 enable set |
| 0x1400_00CC | IRQ3_ENABLECLR | W | 22 | IntCtrl3 enable clear |

Table A.2: Memory map table for interrupt controller on Integrator/AP

| Bit | Name | Function |
|---|---|---|
| 0 | SOFTINT | Software interrupt |
| 1 | UARTINT0 | UART 0 interrupt |
| 2 | UARTINT1 | UART 1 interrupt |
| 3 | KBDINT | Keyboard interrupt |
| 4 | MOUSEINT | Mouse interrupt |
| 5 | TIMERINT0 | Counter-timer 0 interrupt |
| 6 | TIMERINT1 | Counter-timer 1 interrupt |
| 7 | TIMERINT2 | Counter-timer 2 interrupt |
| 8 | RTCINT | Real time clock interrupt |
| 9 | EXPINT0 | Logic module 0 interrupt |
| 10 | EXPINT1 | Logic module 1 interrupt |
| 11 | EXPINT2 | Logic module 2 interrupt |
| 12 | EXPINT3 | Logic module 3 interrupt |
| 13 | PCIINT0 | PCI bus (INTA#) |
| 14 | PCIINT1 | PCI bus (INTB#) |
| 15 | PCIINT2 | PCI bus (INTC#) |
| 16 | PCIINT3 | PCI bus (INTD#) |
| 17 | LINT | V3 PCI brige interrupt |
| 18 | DEGINT | CompactPCI auxiliary interrupt (DEG#) |
| 19 | ENUMINT | CompactPCI auxiliary interrupt (ENUM#) |
| 20 | PCILBINT | PCI local bus fault interrupt |
| 21 | APCINT | External interrupt reserved for AutoPC (8 sources) |

Table A.3: IRQ register bit assignments

# Bibliography

[1] ARM Limited, editor. *ARM Development Suite Debug Target Guide*, chapter 4.13. ARM Limited, November 2000. ARM DUI 0058C.

[2] ARM Limited, editor. *ARM Integrator/AP User Guide*, chapter 4.4, 4.6. ARM Limited, September 1999. ARM DUI 0098A.