ARC labs handbook

Release 2018.09

Synopsys

2018

CONTENTS:

CHAPTER

ONE

OVERVIEW

This is a handbook for ARC labs which is a part of ARC university cour. It's written to to help students who attened the ARC university courses and anyone who is interested in ARC to get started in ARC processor development. It describes all the basic elements of ARC labs and how to finish the labs with step by step approach.

This book can be used as a Lab teaching material for ARC university courses at undergraduate or graduate level with majors in Commuter Science, Computer Engineering, Electrical Engineering; or for professional engineers.

This handbook includes 12 labs currently (more labs will be added in the future), which can be classified into 3 levels:

- Level1: ARC basic

The labs in this level cover the basic topics about ARC, e.g., the installation of tools, hello world, interrupts.

- Level2: ARC advance

The labs in this level cover the advanced topics about ARC, e.g., RTOS, customized linkage.

- Level3: ARC exploaration

The labs in this level will cover some complex applications.

Most of labs are based on the embARC Open Software Platform (OSP) is an open software platform to facilitate the development of embedded systems based on DesignWare® ARC® processors.

It is designed to provide a unified platform for DesignWare® ARC® processors users by defining consistent and simple software interfaces to the processor and peripherals, together with ports of several well known FOSS embedded software stacks to DesignWare® ARC® processors.

For more details about embARC OSP, please refere its online docs

1.1 Supported Hardware Platform

The following hardare platforms are supported in this handbook:

- ARC EM Starter Kit
- ARC IoT Development Kit

1.2 Reference

Here is reference for this hand book.

Item	Name
1	ARC EM Databook
2	MetaWare docs
3	ARC EM Starter Kit User Guide
4	ARC GNU docs

CHAPTER

TWO

GETTING STARTED

Use this guide to get started with your ARC labs development.

2.1 Software Requirement

- **ARC Development Tools** Choose **MetaWare Toolkit** and/or **ARC GNU Toolchain** from the following list according to your requirement.
  - MetaWare Toolkit
    - Premium MetaWare Development Toolkit (2017.09) The DesignWare ARC MetaWare Development Toolkit builds upon a 25-year legacy of industry-leading compiler and debugger products. It is a complete solution that contains all the components needed to support the development, debugging and tuning of embedded applications for the DesignWare ARC processors.
    - DesignWare ARC MetaWare Toolkit Lite (2017.09) A demonstration/evaluation version of the MetaWare Development Toolkit is available for free from the Synopsys website. MetaWare Lite is a functioning demonstration of the MetaWare Development Toolkit, but has a number of restrictions, including a code-size limit of 32 Kilobytes and no runtime library sources. It is available for Microsoft Windows only.
  - ARC GNU Toolchain
    - Open Source ARC GNU IDE (2017.09) The ARC GNU Toolchain offers all of the benefits of open source tools, including complete source code and a large install base. The ARC GNU IDE Installer consists of Eclipse IDE with ARC GNU plugin for Eclipse, ARC GNU prebuilt toolchain and OpenOCD for ARC
- Digilent Adept Software for Digilent JTAG-USB cable driver
- Tera Term or PuTTY for serial terminal connection, 115200 baud, 8 bits data, 1 stop bit and no parity (115200-8-N-1) by default.

Note: If using embARC with GNU toolchain on Windows, install Zadig to replace FTDI driver with WinUSB driver. See How to Use OpenOCD on Windows for more information.

Check the following items and set development environment.

- Make sure the paths of the above required tools for the MetaWare toolkit and ARC GNU toolchain are added to the system variable **PATH** in your environment variables.
- We recommend users to install ARC GNU IDE to default location. Otherwise you need to make additional changes as below.
  - If running and debugging embARC applications using **arc-elf32-gdb** and OpenOCD for ARC, make sure 1) the path of **OpenOCD** is added to the **PATH** in your environment variables, and 2) modify **OPENOCD_SCRIPT_ROOT variable** in *<embARC>/options/toolchain/toolchain_gnu.mk* according to your **OpenOCD** root path.

  - If running GNU program with using the GNU toolchain on Linux, modify the **OpenOCD** configuration file as Linux format with LF line terminators. **dos2unix** can be used to convert it.

Note: Check the version of your toolchain. The embARC software build system is purely makefile-based. make/gmake is provided in the MetaWare toolkit (gmake) and ARC GNU toolchain (make)

2.2 Install Software Tools

2.2.1 Install Metaware Toolkit

2.2.2 Install ARC GNU Toolchain

2.2.3 Install embARC OSP

2.2.4 Install USB-JTAG Drivers

CHAPTER

THREE

LABS

3.1 Overview

3.2 Labs

3.2.1 Level 1 Labs

How to use ARC IDE

Purpose

Equipment

Content

Principles

Steps

Exercises

How to use embARC OSP

Purpose

Equipment

Content

Principles

Steps

Exercises

ARC features: timer and auxiliary registers

Purpose

- To learn the timer resource of ARC EM processor

- To learn how to use the auxiliary registers to control the timer
- Read the count value of the timer, and implement a time clock by the timer

Equipment

PC, IoTDK, embARC OSP, example \Lab\timer in embARC OSP

Content

Read the auxiliary registers of ARC EM to get the version and other setting information of the timer resource. As all the EM processors have **Timer0**, we use the **Timer0** in this lab, and write the auxiliary registers to initialize, start and stop the timer. By reading the count value of the timer, we can calculate the execution time of a code block with the count value and the clock frequency.

Principles

Introduction of timer and auxiliary registers

The timers in ARC EM processor

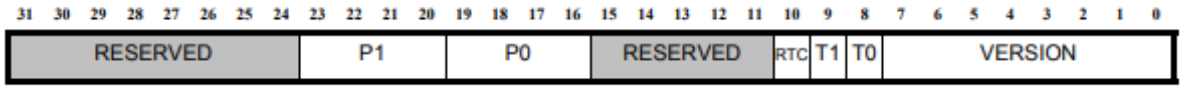
- Two 32-bits programmable timers **Timer0** and **Timer1**
- One 64-bits **RTC**(Real-Time Counter)

All the times are configurable, for example, there are four EM processor cores in **ARC EMSK1.1**, the configuration information are as follow.

Timer	EM4	EM4_16CR	EM4	EM4_16CR
HAS_TIMER0	1	1	1	1
HAS_TIMER1	1	0	1	0
RTC_OPTION	0	0	0	0

The auxiliary register Timer BCR storaged the timer resource information of an EM processor core, the register address of **TIMER_BUILD** is *0x75*.

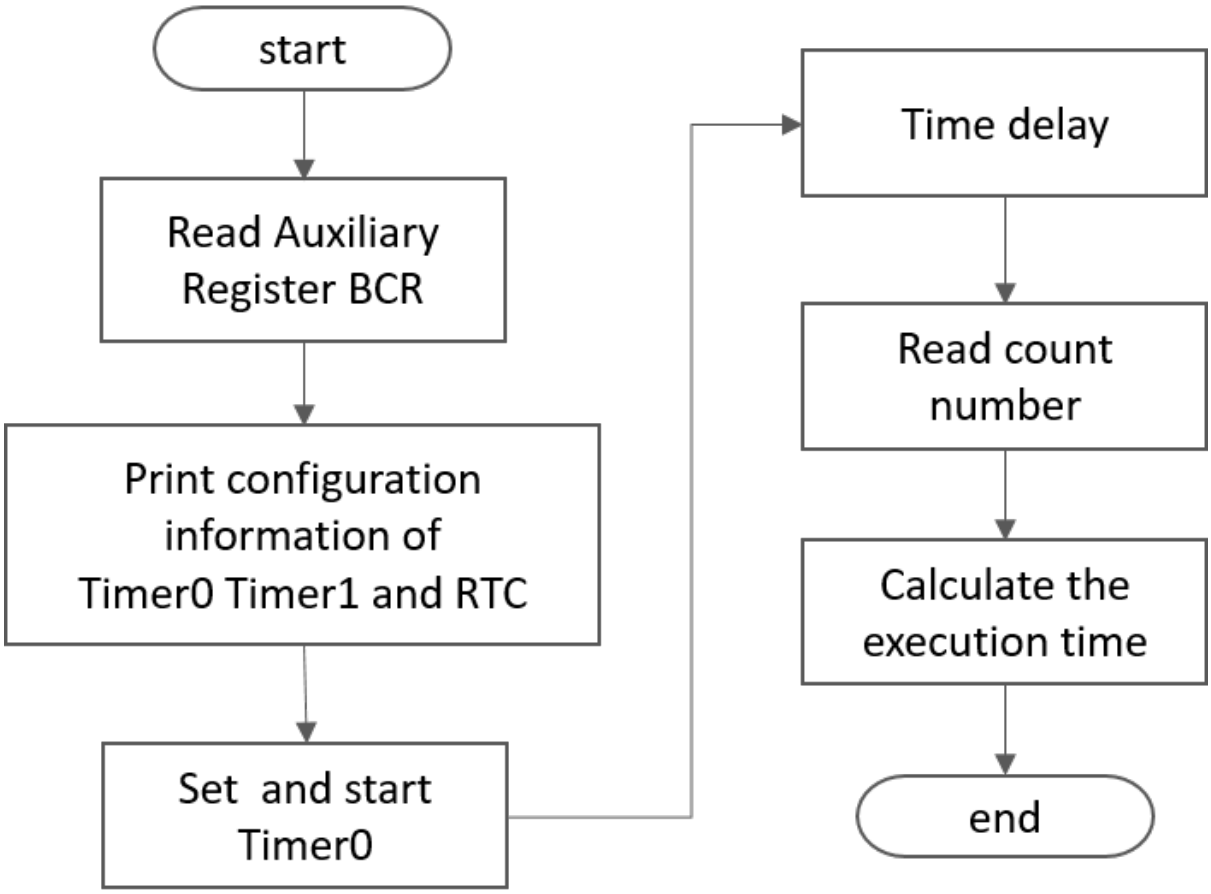
TIMER_BUILD



As we know the timer resources the EM processor configured, we can get the timer's configuration information and control the timers by writing and reading the auxiliary register of that timer. For example, there are the related auxiliary registers of the **Timer0**.

Auxiliary Register	Name	Permission	Description
0x21	COUNT0	RW	Processor timer 0 count value
0x22	CONTROL0	RW	Processor timer 0 control value
0x23	LIMIT0	RW	Processor timer 0 limit value

Program flow chart



Steps

Makefile configuration

There are two ways to do the configuration.

First, configured by compile command, for example:

```
make BOARD=emsk BD_VER=22 CUR_CORE=arcem9d -j4 TOOLCHAIN=gnu run
```

Second, configured by modifying the makefile. At here, the compile command will be very simple, for example:

```
make -j4 run
```

Open the folder *embarc_osp\example\Lab\timer*, and open the *makefile*, here is the default configuration.

```
# Application name
APPL ?= lab_3_Timer_Interrupts

##
# Current Board And Core
##
BOARD ?= nsim
BD_VER ?= 1506
CUR_CORE ?= arcemfull
```

(continues on next page)

(continued from previous page)

```
##
# Set toolchain
##
TOOLCHAIN ?= gnu

#
# root dir of embARC
#
EMBARC_ROOT = ../../..

MID_SEL = common

# application source dirs
APPL_CSRC_DIR = .
APPL_ASMSRC_DIR = .

# application include dirs
APPL_INC_DIR = .
```

- Reconfigure **BOARD** and **CUR_CORE**, in this lab, we use the launch board *iotdk*

```
##
# Current Board And Core
##
BOARD ?= iotdk
BD_VER ?= 10
CUR_CORE ?= arcem9d
```

- Reconfigure **TOOLCHAIN**, select the toolchain *gnu* or *metaware* you used

```
##
# Set toolchain
##
TOOLCHAIN ?= gnu
```

- Reconfigure **EMBARC_ROOT**, make sure the relative path between *embARC OSP* root folder and the *timer* folder is correct.

```
#
# root dir of embARC
#
EMBARC_ROOT = ../../..
```

Main code

Read auxiliary register BCR_BUILD

We can use the function `_arc_aux_read` to read the auxiliary register for the timer resource information.

Read auxiliary register **TIMER_BUILD**. In the register **TIMER_BUILD** The lower 8 bits indicate the core version information, the bit 9 indicate the **Timer0**, the bit 10 indicate the **Timer1**, the bit 11 indicate the **RTC**. Here is the code:

```
uint32_t bcr = _arc_aux_read(AUX_BCR_TIMERS);
int timer0_flag=(bcr >> 8) & 1;
int timer1_flag=(bcr >> 9) & 1;
int RTC_flag=(bcr >> 10) & 1;
```

Read timer related auxiliary registers, for example, the **Timer0**. Here is the code:

```
EMBARC_PRINTF("Does this timer0 exist?  YES\r\n");
/*Read auxiliary register configuration information*/
EMBARC_PRINTF("timer0's operating mode:0x%08x\r\n",_arc_aux_read(AUX_TIMER0_CTRL));
EMBARC_PRINTF("timer0's limit value :0x%08x\r\n",_arc_aux_read(AUX_TIMER0_LIMIT));
EMBARC_PRINTF("timer0's current cnt_number:0x%08x\r\n",_arc_aux_read(AUX_TIMER0_
↪CNT));
```

Stop-Set-Start the Timer0

We can use the function `_arc_aux_write` to write the auxiliary register.

To control the **Timer0** with the related auxiliary registers.

- **COUNT0**: write this register to set the initial value of the **Timer0**. It will increase from the set value at anytime you write this register.
- **CONTROL0**: write this register to update the control modes of the **Timer0**.
- **LIMIT0**: write this register to set the limit value of the **Timer0**, the limit value is the value after which an interrupt or a reset must be generated.

In this lab, we should stop timer before setting and starting it, the function `timer_stop` is already encapsulated in embARC OSP, you can use this function or directly write the register. And then set the timer work mode, enable interrupt or not and set the limit value. At last start the timer. Here is the code:

```
    /* Stop it first since it might be enabled before */
    _arc_aux_write(AUX_TIMER0_CTRL, 0);
    _arc_aux_write(AUX_TIMER0_LIMIT,0);
    _arc_aux_write(AUX_TIMER0_CNT, 0);
    /*This is a example about timer0's timer function.*/
    uint32_t mode = TIMER_CTRL_NH;/*Timing without triggering interruption.*/
    uint32_t val = MAX_COUNT;
    _arc_aux_write(AUX_TIMER0_CNT, 0);
    _arc_aux_write(AUX_TIMER0_LIMIT,val);
/*start the specific timer*/
    _arc_aux_write(AUX_TIMER0_CTRL,mode);
```

When the timer is running, we can read the count value of the timer,and calculate the execution time of a code block. Here is the code:

```
uint32_t start_cnt=_arc_aux_read(AUX_TIMER0_CNT);
/**
 * code block
 */
uint32_t end_cnt=_arc_aux_read(AUX_TIMER0_CNT);
uint32_t time=(end_cnt-start_cnt)/(BOARD_CPU_CLOCK/1000);
```

Compile and debug

- Compile and download

Open `cmd` under the folder *example\Lab\timer*, input the compile command as follow:

```
make -j4 run
```

Note: If your toolchain is meteware, you should use `gmake`

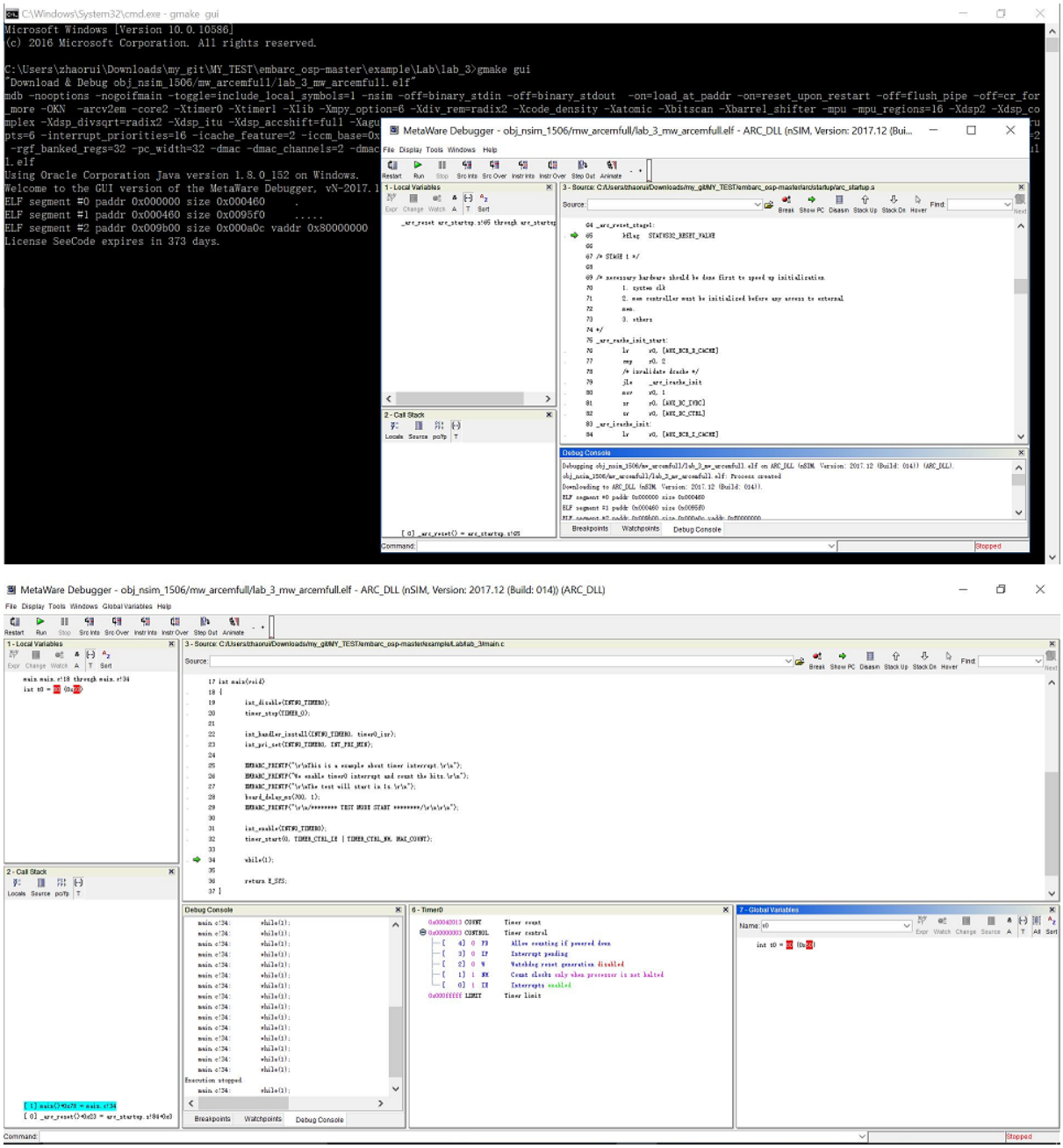
- Output

```
------------------------------------------------------------
      _____                                    _ _____
     |  _ \ ______      ______ _ __ ___  __| | __ ) _   _
     | |_) / _ \ \ /\ / / _ \ '__/ _ \/ _` |  _ \| | | |
     |  __/ (_) \ V  V /  __/ | |  __/ (_| | |_) | |_| |
     |_|   \___/ \_/\_/ \___|_|  \___|\__,_|____/ \__, |
                                                   |___/
                          _       _    _____   _____
           ___ _ __ ___  | |__   / \   |  _ \ / ___|
          / _ \ '_ ` _ \| '_ \ / _ \  | |_) | |
         |  __/ | | | | | |_) / ___ \|  _ <| |___
          \___|_| |_| |_|_.__/_/   \_\_| \_\\____|
     ------------------------------------------------------------

     embARC Build Time: Aug 22 2018, 15:32:54
     Compiler Version: Metaware, 4.2.1 Compatible Clang 4.0.1 (branches/release_40)
     Does this timer0 exist?  YES
     timer0's operating mode:0x00000003
     timer0's limit value :0x00023280
     timer0's current cnt_number:0x0000c236

     Does this timer1 exist?  YES
     timer1's operating mode:0x00000000
     timer1's limit value :0x00000000
     timer1's current cnt_number:0x00000000

     Does this RTC_timer exist?   NO

     The start_cnt number is:2
     /******** TEST MODE START ********/

     This is TEST CODE.

     This is TEST CODE.

     This is TEST CODE.

     /******** TEST MODE END ********/
     The end_cnt number is:16785931
     The board cpu clock is:144000000

     Total time of TEST CODE BLOCK operation:116
```

- Debug

Open `cmd` under the folder *example\Lab\timer*, input the command as follow:

```
make gui
```



The debug view will pop up automatically, we can watch the variables and registers.

Exercises

In the debug view, observe and understand the contents of the interrupt vector table.

Note: Click the Memory button in the debug view Debugger drop-down menu to see the contents of the memory in real time.

ARC features: interrupts

Purpose

Equipment

Content

Principles

Steps

Exercises

How to use ARC board

Purpose

Equipment

Content

Principles

Steps

Exercises

A simple bootloader

Purpose

Equipment

Content

Principles

Steps

Exercises

3.2.2 Level 2 Labs

A WiFi temperature monitor

Purpose

Equipment

Content

Principles

Steps

CHAPTER

FOUR

APPENDIX

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- search