Report: Exercise 1

Group 9: Øyvin Richardsen, Sandor Zeestraten, Stian Habbestad

Norwegian University of Science and Technology
TDT4258 Energy Efficient Computer Design
February 14, 2013

Abstract

In this exercise in we write assembly code for a AVR microcontroller on a development board. The goal is to be able to move an active LED left and right, on the 8 LED's available. Through this exercise we aim to become more familiar with AVR, assembly and energy efficiency related to the interrupt-based system. We build up our code step by step from simple LED activation, to moving it, and then introducing the interrupt design of the code. During the programming we also got well acquainted with GNU tools and the AVR toolchain for compiling and debugging.

Contents

1 Introduction

The objective of this exercise was to become familiar with GNU development tools, AVR32 assembly code programming and its architecture, AVR toolchain, the use of interrupts and general I/O programming. The task was to create a simple program in assembly code that enables the manipulation of LEDs with a pair of switches using interrupt based code. As the goal of this course is energy efficient computer design, the main focus is on creating interrupt-based systems.

2 Description and methodology

LEDs As advised in the compendium, we started by becoming familiar with the operation of the LED's. First off we started with trying to turn on all the LEDs. Thereafter we try to turn LEDs on and off, for example by only leaving one LED on. After a while we became comfortable manipulating the LED's and created a method for making the LED's either go left or right with the possibility of wrapping around, i.e. jumping from the far right to the far left and vice versa.

```
left:
/* Move LED to the left */
lsl r4, 1                /* Shift left to enable previous LED */
cp.w r4, r12             /* Check if out of bounds */
brle turn_on
mov r4, r11              /* Max wraparound */
rjmp turn_on
```

Above is an excerpt of the code that moves the LED and takes in account for the wrap around by checking if it is going out of bounds. Both the *left* and *right* methods jump to the *turnon* method in order to turn on the selected LED as seen below.

```
turn_on:
/* Turn on the selected LED */
st.w r1[AVR32_PIO_CODR], r8     /* Turn off all LED's */
st.w r1[AVR32_PIO_SODR], r4     /* Turn on the LED specified in r4 */
mov r9, 0xffff                  /* Countdown value used for debouncing */
rjmp intr_sleep
```

Switches After the LED's we looked at how to use the switches. We activate the switches as inputs and we also enable the pull-up resistors to make the input signals from the switches logically high. Now we assign two of the switches, *SW7* and *SW5*, for left and right movement respectively of the LEDs. If the LED was at the far left or far right, it wraps around as described in the previous section. The switches actually sends two signals for each complete press, one when pressed down and one when released. Below is the *interrupt routine* that reads the switch status, makes sure to avoid the double signals from a single press and calls the correct method for which button was pressed.

```
interrupt_routine:
/* Load which button was pressed */
ld.w r7, r0[AVR32_PIO_PDSR]     /* Read Pin Data Status Register */

/* Avoid double interrups on single button press */
cp.w r10, r7             /* Check if button was pressed on last interrupt */
breq return              /* Skip to avoid double interrupt */
mov r10, r7              /* Note which button was pressed for next time */
/* Check which button was pressed */
```

```
cp.w r5, r7              /* Check if left button was pressed */
breq left
cp.w r6, r7              /* Check if right button was pressed */
breq right
```

Bouncing is an issue with switches where there may be many registered contacts during a press which is often an unwanted feature. Below is the *intr sleep* method that gets called after a new LED is turned on to avoid the bouncing effect by waiting for a short duration.

```
intr_sleep:
/* Debouncing by counting down from 0xffff to 0 */
sub r9, 1
cp.w r9, 0
breq return
rjmp intr_sleep
```

Interrupts Now that we had a working program using a simple loop that checked for changes of the switch statuses, we saved this as a seperate file called *lights.s*. The next step is to implement interrupt, specifically for reading the switch status. Otherwise we want the program to go into a sleep mode to save energy. In the start of the program we set the *EVBA* offset to 0 as advised in the compendium so we could directly point to the part of the program we wanted to jump to in case of an interrupt.

```
_start:
...
/* Set up interrupt */
mov r3, 0b00000000          /* Set EVBA offset to 0 */
mtsr 4, r3
mov r3, interrupt_routine   /* Set autovector to interrupt_routine */
st.w r2[AVR32_INTC_IPR14], r3
csrf SR_GM                  /* Turn on interrupts */
rjmp loop
```

At the end of the whole process of waking up to an interrupt, detecting which switch was pressed, turning on the desired LED, we need to read the *Interrupt Status Register* to clear it so it can receive new interrupts. It then jumps back from the interrupt routine and to the original state before the interrupt as seen below.

```
return:
/* Read ISR and return to normal state after interrupt */
ld.w r3, r0[AVR32_PIO_ISR]        /* Read ISR to allow new interrupts */
rete
```

3 Results

The result of the exercise is that we have coded a simple program with assembly coding. It uses interrupt-based code where the microcontroller sleeps while waiting for input via I/O. In our case it checks for activity on the switches *SW7* and *SW5*. If they are pressed, they send an interrupt signal which wakes up the microcontroller and jumps to an interrupt routine that figures out if the left or right switch was pressed. It then moves a single active LED left or right depending on which switch was pressed. At the end it goes back to sleep waiting for further interrupts.

4 Tests

Description We've created a few test scenarios in order to test different aspects and corner cases of our code. The main prerequisite was that the STK1000 development board was setup with given jumper settings given in the compendium and the parallel I/O port B connected to the switches and port C to the LEDs. Both the STK1000 and JTAGICE MKII debugger were connected and powered on. The tests were conducted by a person interacting with the switches and another person logging the results.

Results Below is a table of the different tests we ran, the preconditions and the results.

Name	Preconditions	Description	Expected result	Test result
Steady-state test	Power is on, and the board is connected	Upload program to card, push reset switch	The board is powered and LED 7 should be on	Passed
Move LED right test	Program is active, only LED 7 is on	Push switch 6	LED 7 should be turned off and LED 6 should be turned on	Passed
Move LED left test	Program is active, only LED 6 is on	Push switch 7	LED 6 should be turned off and LED 7 should be turned on	Passed
Left LED wrap-around test	Program is active, only LED 7 is on	Push switch 67	LED 7 should be turned off and LED 0 should be turned on	Passed
Right LED wrap-around test	Program is active, only LED 0 is on	Push switch 6	LED 0 should be turned off and LED 7 should be turned on	Passed
Long push test	Program is active, only LED 7 is on	Push and hold switch 6 for a few seconds	LED 7 should be turned off and LED 6 should be turned on as soon as the switch is pushed	Passed

5 Evaluation of assignment

The compendium was of great help. Sometimes running in debug mode gave different results than a test run, which we found rather confusing. Overall the difficulty was just right although it was sometimes difficult to find the needed documentation.

6 Conclusion

After having completed this exercise we now have a better understanding of how a to program a microcontroller on a low level by coding in assembly. This will be helpful when we will start coding in C in the next exercises.

7 Appendix

References

[1] AVR32 Architecture Document `http://www.atmel.com/images/doc32000.pdf`

[2] AT32AP7000 Datasheet `http://www.atmel.com/Images/doc32003.pdf`

[3] TDT4258 Compendium `http://www.idi.ntnu.no/emner/tdt4258/_media/kompendium.pdf`

.