CMPE-250 Assembly and Embedded Programming

Laboratory Exercise Ten

Timer Driver Input Timing

By submitting this report, I attest that its contents are wholly my individual writing about this exercise and that they reflect the submitted code. I further acknowledge that permitted collaboration for this exercise consists only of discussions of concepts with course staff and fellow students. Other than code provided by the instructor for this exercise, all code was developed by me.

Shubhang Mehrotra
Performed 8th April 2021.
Submitted 15th April 2021.

Lab Section 1
Instructor: Muhammad Shaaban
TA: Anthony Bacchetta
Aidan Trabuco
Sam Myers
Payton Burak

Lecture Section 1
Lecture Instructor: Dr. Roy Melton

Abstract

The purpose of this exercise was to develop a timer driver in a Cortex-M0+ assembly language program. The scope of the activity was to write assembly code that can utilize the timer driver to time the user input and echo it back to the user. The timer driver uses the Periodic Interrupt Timer (PIT) with channel zero. The program utilizes the ISR concepts developed previously to handle interrupt based serial connection to the KL05Z board. The activity was successful in implementing the timer driver accurate to 10 milliseconds.

Procedure

The PIT module provided on the KL05 board has two 32-bit down counters: timer 0 and timer 1. For the purposes of this activity, timer 0 was utilized. The timer 0 module counts down each clock cycle from the Timer Start value (TSV) to zero. The timer 0 in this implementation is setup to set interrupt flags once the down counter finishes operation. The interrupts are then counted to keep track of the interrupt period. Since the timer 0 uses the PIT Interrupt service routine to keep track of time, the PIT was initialized so that the timer 0 can be accurate to 0.01s.

The core clock of the KL05 is 48 MHz and the bus clock is 24 MHz.- Based on that, the value to be loaded to the TSV was calculated using the following formula:

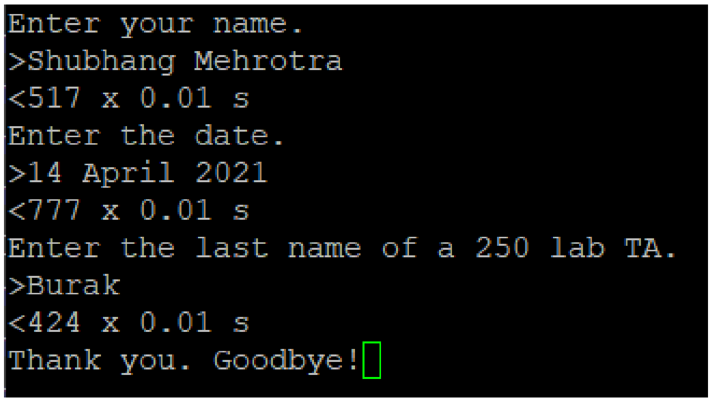
$$TSV = \frac{t_{TimerClockRate}}{T_{InterruptRate}} - 1$$

For an interrupt every 0.01s with a bus clock of 24MHz, the value of TSV was found to be 239,999, which was loaded to the LDVAL register of the PIT Module. To ensure the PIT timer is on the correct interrupt period, the bit controlling the Timer Control Register was cleared. Since PIT interrupts were essential for accurate measurements, their priority bit was set to 0 to denote highest priority in the NVIC. Before the module is ready to interrupt, it is also important to clear the MDIS field bit of the PIT Module Control Register. Finally, to enable the timer 0 to generate an interrupt, it is important to set bits 0 and 1 of the Timer Control Register.

The Interrupt service routine for the PIT module was written to keep track of the interrupts generated by the timer 0. The “RunStopWatch” variable oversaw when the ISR was to start keeping the count, and when to stop. So, the ISR only incremented the counter when RunStopWatch variable was set, otherwise it cleared the interrupt and returned. To make sure the PIT ISR was called as required, its Label was also replaced at the place of the Dummy_Handler in the KL05Z Vector table. The exact location for the change in this activity, was at line 38. The driver code for the program set and cleared the RunStopWatch variable between each user input. The user was prompted three different inputs which required different times to enter. The time taken between user prompt and input end was then calculated using the PIT ISR as described.

Results

The code written for the timer driver operations was translated to machine code and then downloaded to the Freedom Board to ensure desired operation. A PuTTY environment was used to obtain the timing output of different input prompts, as shown in Figure 1.



```
Enter your name.
>Shubhang Mehrotra
<517 x 0.01 s
Enter the date.
>14 April 2021
<777 x 0.01 s
Enter the last name of a 250 lab TA.
>Burak
<424 x 0.01 s
Thank you. Goodbye!
```

Figure 1. Output

The code first prompted the user for their name and started the counter as soon as it finished the prompt. When the user completed the input string, the timer counter stopped, and the time elapsed between the input prompt and finishing the string is output to the screen as shown in Figure 1. The time is accurate to within 0.01 s as shown by the time count unit.

The user was then prompted for the Date input and the last name of a CMPE 250 TA respectively. The code counted the time between each prompt and the end of user input and output it to the screen, as desired. Afterward, it sent a closing message and stopped functioning. The next two prompts and the “goodbye” message is also depicted in Figure 1.

The translation of the code generates a listing file for the assembly code, and building it generates the map file. The contents of the two files were analyzed to obtain the memory addresses of various objects in the code. For instance, the map file lists the execution addresses in the ROM for each entity in code. The execution address, along with the size of the entity give information about the exact address range used by a particular object. The data obtained for each object is listed in Table 1.

Table 1. Memory ranges

Object	Memory Address Start	Memory Address End (Start + Size - 1)	Size (bytes)
MyCode AREA	0x000000410	0x00000093F	1328
PIT ISR code	0x0000005BE	0x0000005D7	26
Constants in ROM	0x000001C4	0x0000022F	108
RAM Usage	0x1FFFFD00	0x1FFFFE3F	320

From Table 1, it can be seen that the instructions written for the Main program began at memory address 0x00000410 and went all the way up to address 0x0000093F, taking 1328 bytes in the memory. The PIT ISR Code occupied 26 bytes starting from address 0x0000005BE going to 0x0000005D7, while the constants defined in the program occupied 108 bytes starting from address 0x000001C4 and going up to address 0x0000022F.

The variables used by the program occupy space in a much more volatile system of the RAM. All the variables in the program occupied 320 bytes in RAM, as the code had variables present from previous lab exercises as well. The two Variables required for the timer driver code occupied a total of 5 bytes. Variable “RunStopWatch” was present at address 0x1FFFFD00 occupying 1 byte, and “Count” variable started at address 0x1FFFFD04, occupied 4 bytes, and went up to address 0x1FFFFD07. The input also utilized an “InputString” variable which occupied 79 bytes, starting from address 0x1FFFFD08, and going to address 0x1FFFFD57.

Conclusion

The activity was successful in designing and implementing a Timer Driver for a KL05Z board using assembly language. Many tasks in computing require accurate measurement of time between processes and inputs. This is enabled through such timer drivers which can keep track of the time elapsed by looking at the internal clock of the system and translating it to interface time. The code implemented in the activity was successfully keeping track of time with an accuracy of 10ms, which is more enough to calculate time between different user inputs but can be improved with better hardware and other implementations to time even shorter intervals.