Timer Triggered ADC Example `adc_timer_trigger`

1 Overview

In the `adc_interrupt` project, an interrupt was used to signal the processor the end of conversion. However, the conversion was still started from the terminal. In most cases, a precise timing is required to sample voltages in the embedded system.

In the `adc_timer_trigger` project, we will describe how the ADC conversion can be triggered **by the hardware and without intervention of any software**, using one timer. User manual Sec. 21.3.3 "ADC hardware trigger inputs" describes in Table 277, "ADC hardware trigger inputs", what hardware peripherals of the processor can trigger an ADC conversion. The possibilities are:

1. Software triggered (already described in the other projects)
2. `ADC_PINTRG0`: An external pin logic level change can trigger a conversion.
3. `ADC_PINTRG1`: Same as 1, for another sequence.
4. `SCT0_OUT3`: Timer SCT OUT3 can trigger an ADC conversion (this is what we will use).
5. `ACMP_O`: The analog comparator (essentially an Op-Amp) on the chip can be used. This is handy for building switch mode power supplies etc.
6. `ARM_TXEV` An ARM TXEV can trigger a conversion. It is not relevant at this preliminary level of discussion.

In this project we will configure the system so that the ADC conversion trigger source is the SCT timer, and conversions are performed at precise timing intervals. The end of conversion is still handled by the same interrupt as in the `adc_interrupt` project; INT #16: "`ADC_SEQA_IRQ ADC sequence A completion`". Therefore, the compelte voltage measurement from triggering to measurement and then notification of the result is completely done by the hardware, and you will see that the main loop of the program is completely empty.

The code is ran in a similar way to the `adc_basic` project: Connect a potentiometer to ADC0 Channel 1 as described elsewhere. Once the program is running on the processor, open a serial port terminal to see the measurement result. However in this project, there is no need to press any buttons; the ADC is periodically sampled using the SCT timer Output 3. The serial terminal communication speed must be set to 115200 baud as before.

2 The Structure of the Program

The `adc_timer_trigger` project builds up on to the `adc_interrupt` project, so only the differences will be highlighted here. The general flow is as follows:

1. In `main()`, the initialization code is almost the same and will be skipped. The main difference is where the SCT timer is also configured:

   ```
   CLOCK_EnableClock(kCLOCK_Sct);
   SCT_Configuration();
   ```

   The `SCT_Configuration()` function will be explained later.
2. The ADC0 interrupt is enabled in the same way as in `adc_interrupt` project.
3. The main loop is completely empty. All the work is performed in the background by hardware peripherals, interrupts and ISRs:

   ```
   while (1) {
   }
   ```

3 The ISR

The ISR is almost the same as in `adc_interrupt` project, so please refer to that for details. The only difference is that the result is printed out in the ISR itself, and not in the main loop. This was done to increase the dramatic effect that the main loop is completely empty.

Please note that typically it is bad practice to use functions with long or unpredictable execution times within an ISR because the ISR essentially causes all other code in the system to stop until it is done. In general an ISR must be written to complete and exit as quickly as possible. However, the PRINTF function may take a comparatively long time to execute because it has to re-format numbers to be able to print them as human readable characters in base 10 and also send them over a potentially slow communication channel. Therefore it is not advisable to use functions such as PRINTF in an ISR. PRINTF is used in an ISR in this example to emphasize the fact that the main loop is not doing anything.

4 ADC0 Configuration

Note that the configuration and initialization of the ADC is *almost* exactly the same as in `adc_interrupt` project with one difference; the source of the trigger source:

```
  adcConvSeqConfigStruct.triggerMask       = 3U;
```

The possible trigger sources have been described in Sec. 1 of this document. It can be seen there and in Table 277, "ADC hardware trigger inputs" of Sec. 21.3.3 "ADC hardware trigger inputs" of the Reference Manual, the trigger source of "ADC0 Sequence A" is set to '3' which corresponds to "SCT0 Output 3". In the other two projects, this was set to '0' which corresponds to "software trigger". The configuration struct is then assigned to Sequence A using the PSL function call:

```
ADC_SetConvSeqAConfig(ADC0, &adcConvSeqConfigStruct);
```

5 SCT0 Timer Configuration

The only remaining part is the configuration of the SCT0 timer to trigger the ADC. This is performed in the function `void SCT_Configuration(void)`. Actually, this is not a newly written function, and it was carried over from the example project `16bit_counter` which was covered in the timers discussion. It was slightly modified to fit this application. The modifications are:

1. The SCT is still configured as two separate “high” and “low” counters, but only low counter is used.
2. Previously, outputs 1 and 2 of the SCT timer were used and connected to package pins using the SWM function. In this project, only otput 3 is used and it is not connected to any physical pin (and therefore no configuration with the SWM is necessary).
3. A smaller match value is used so that more measurements are made per second.
4. The events to toggle the output when the counter counts down do not change.

SCT0 low side counter is configured to repetitively count and periodically toggle output 3 to trigger the ADC.

6 Other Code Components of the Project

The project structure is almost identical to the `adc_interrupt` project, except here the `main()` function resides in the file `adc_timer_trigger.c` and the name of the binary and hex files have been appropriately changed. No further explanation will be given in this section.

Ahmet Onat 2023