ADC Interrupt example `adc_interrupt`

---

1 Overview

In the `adc_basic` project, the processor had to stop and wait for the conversion to complete. However, the LPC824 ADC can also produce an INT at the end of conversion. In this example, an ISR to service that interrupt is described, as well as how the interrupt is configured. Since many parts of the program are identical to `adc_basic` project, only the significant changes will be described here.

The interrupt that is used in this example is external INT #16:
"`ADC_SEQA_IRQ ADC sequence A completion`" as shown in Sec. 4.3.1 "Interrupt sources", Table 5: "Connection of interrupt sources to the NVIC". Remember that INT #16 (or 17^{th} interrupt, since the table starts from '0') does not mean the 17^{th} entry in the **vector table**, because there are 16 more **internal interrupts** before the external interrupts. So the "`ADC_SEQA_IRQ ADC sequence A completion`" interrupt is the 33^{rd} entry in the vector table. Please take a moment to understand this.

As in the `adc_basic`, once the code is running, open a serial port terminal to see its output. When you press a key on your PC, Alakart starts conversion of the voltage applied to ADC0 Channel 1 which is labeled as P06. When it completes, the result value is written back on the serial terminal. Note that the serial terminal communication speed must be set to 115200 baud. You can apply a voltage to `PIO0_6` pin by using your potentiometer as explained in `adc_basic` project.

2 The Structure of the Program

The `adc_interrupt` project is very similar to the `adc_basic` project, so only the differences will be highlighted. 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 interrupts are enabled.
2. The ADC0 interrupt is enabled within the peripheral itself, and then globally within NVIC. This is the "ADC0 Sequence A Conversion Complete" interrupt. For ADC0 interrupt enable, see Sec. 21.6.9 "A/D Interrupt Enable Register" , and for how the interrupt is enabled in the chip level in NVIC, see Sec. 4.4.1 "Interrupt Set Enable Register 0 register". Bit 16 of `ISE_ADC_SEQA` register is used.

```
1 ADC_EnableInterrupts(ADC0, kADC_ConvSeqAInterruptEnable); // Within
   ADC0
2 NVIC_EnableIRQ(ADC0_SEQA_IRQn);
```

3. In the main loop (`while(1) {...`), the program waits for a character to arrive from the terminal:

```
GETCHAR();
```

After which, it sets the flag that signals the conversion is complete to '`false`' and then starts an ADC conversion:

```
    ADCConvCompleteFlag = false;
    ADC_DoSoftwareTriggerConvSeqA(ADC0); // Start conversion.
```

This part is important. The `ADCConvCompleteFlag` will be set to '`true`' **in the ISR when the ADC conversion has been completed**. Note that it is a global variable defined outside of `main()` as '`volatile`':

```
volatile bool ADCConvCompleteFlag;
```

The specification `volatile` means that it can be changed by an asynchronous event outside of main program flow, e.g., an ISR.
The main loop keeps checking *the flag* (and not the ADC0 registers) to test if the conversion is complete:

```
while (ADCConvCompleteFlag == false) { }
```

And when the conversion is complete it prints out the result.

3 The ISR

The next important part is the ISR. Note that the name of the ISR **must be the same as the name defined in** `startup_LPC824.S`. The name is actually a label for the address of the first instruction of the function, i.e., the interrupt vector defined in `startup_LPC824.S`:

```
    .long   ADC0_SEQA_IRQHandler          /* ADC0 sequence A completion. */
```

The ISR function declaration must have a prototype that is common to all ISRs:

```
void ADC0_SEQA_IRQHandler(void)
```

Within the ISR, the cause of the interrupt is checked

```
    if (kADC_ConvSeqAInterruptFlag ==
(kADC_ConvSeqAInterruptFlag & ADC_GetStatusFlags(ADC0)))
```

and the conversion result is stored in `ADCResultStruct` using the pointer (`ADCResultPtr`). This pointer was set to point at the actual structure at the begining of `main()` as follows:

```
ADCResultPtr = &ADCResultStruct;
```

Note that the pointer is used in the ISR and therefore must be global, whereas the structure itself need not be global, and is defined within the scope of `main()`

The ISR does two more things:

It clears the INT flag within ADC0. **Typically INT flags must be cleared by the ISR. If not, as soon as the ISR finishes and exits, the same INT will be issued again.** The flag can be cleared in different ways in different processors and peripherals. Please check how it is done here by reading the source code for the function `ADC_ClearStatusFlags(...)` in the drivers directory of PSL and also the Reference Manual.

```
1   ADC_GetChannelConversionResult(ADC0, ADC_CHANNEL, ADCResultPtr);
2   ADC_ClearStatusFlags(ADC0, kADC_ConvSeqAInterruptFlag);
```

Lastly, the ISR sets the global flag to 'true' to signal `main()` that an ISR has executed.

```
1   ADCConvCompleteFlag = true;
```

4 ADC0 Configuration

Note that the configuration and initialization of the ADC is exactly the same as in `adc_basic` project, and will not be mentioned further here.

5 Other Code Components of the Project

Similarly, the project structure is almost identical to the `adc_basic` project, except here the `main()` function resides in the file `adc_int.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