KU LEUVEN

FACULTY OF ENGINEERING TECHNOLOGY

CAMPUS GROUP T LEUVEN

Microcontrollers LAB 2

Interrupts, A/D convertor & PWM generation

Coordinator: L. Geurts

Cooperator: T. Stas

.

Academic Year 2019-2020

Interrupts

Interrupts on the PIC18F25k50

The PIC18F25k50 has multiple interrupt sources and an interrupt priority feature that allows each interrupt source to be assigned a high priority level or a low priority level. The high priority interrupt vector is at 000008h and the low priority interrupt vector is at 000018h. High priority interrupt events will interrupt any low priority interrupts that may be in progress.
Due to the bootloader that is installed on your PIC device however, the interrupt vectors are shifted to address 001008h and 001018 for high and low priority respectively.

There are few registers which are used to control interrupt operation:

Interrupt control registers	INTCON, INTCON2, INTCON3	Registers which contain various enable, priority and flag bits like eg. GIE, TMR0IE, TMR0IF, TMR0IP,… .
Peripheral interrupt request registers	PIR1, PIR2	Contain the individual flag bits for the peripheral interrupts.
Peripheral interrupt enable registers	PIE1, PIE2	Contain the individual enable bits for the peripheral interrupts.
Peripheral interrupt priority registers	IPR1, IPR2	Contain the individual priority bits for the peripheral interrupts.
Reset control register	RCON	Contains the IPEN bit which enables interrupt priorities.

Each interrupt source has three bits to control operation:

Flag bit	Indicate an interrupt event occurred.	Eg. TMR0IF
Enable bit	Allows program execution to branch to the interrupt vector address when the flag bit is set.	Eg. TMR0IE
Priority bit	Select high priority or low priority.	Eg. TMR0IP

Writing an Interrupt Service Routine

Observe the following guidelines when writing an ISR:

1. Use void as the return type and for the parameter specification.
2. Write each ISR prototype using the `interrupt` specifier.
3. If your device supports interrupt priorities, with each function use the `low_priority` or `high_priority` argument to `interrupt`.
4. Inside the ISR body, determine the source of the interrupt by checking the interrupt flag and the interrupt enable for each source that is to be processed, and make the relevant interrupt code conditional on those being set.

Example of an interrupt service routine:

```
/*****************************************************
      Interrupt Handler
*****************************************************/
void __interrupt (high_priority) high_ISR(void){
      if(INTCONbits.TMR0IF == 1){ // Determine source of interrupt
            counter = counter + 1;
            TMR0L = 0x00;
            INTCONbits.TMR0IF=0;  // Clear interrupt flag
      }
}
```

Assignment:

Have a look at the template program on toledo "timer0_ex.c". Look at the configuration registers that are needed to configure the TIMER0 module. Look at the configuration registers that are needed to enable the TIMER interrupt. Search in the datasheet from the PIC18F25k50 for more detailed information about each individual bit. Can you predict what the program is doing? Than download your program onto your board and test.

A/D convertor

The PIC18F25k50 has a built-in analog to digital converter that can perform 10-bit A/D conversions on 10 input channels. An analog signal can be sampled through one of the 12 channels and converted into digital values. The result will be stored in ADRESH (A/D Result High Register) and ADRESL (A/D Result Low Register). In this lab, A/D result format will be left justified, which means the higher 8 bits are the ADRESH register.

To use the module, the A/D hardware should first be initialized. There are three special function registers involved in the configuration of the A/D converter module:

A/D Control Register 0	ADCON0	Controls the operation of the A/D modules.
A/D Control Register 1	ADCON1	Configures the function of port pins and the voltage reference.
A/D Control Register 2	ADCON2	Configures acquisition time, conversion clock and justification.

The following steps should be followed to perform an A/D conversion:

1. Configure the A/D module:
   - Configure analog pins, voltage reference and digital I/O (ADCON1)
   - Select A/D input channel (ADCON0)
   - Select A/D acquisition time (ADCON2)
   - Select A/D conversion clock (ADCON2)
   - Turn on A/D module (ADCON0)
2. Configure A/D interrupt (if desired):
   - Clear ADIF bit
   - Set ADIE bit
   - Set GIE bit
3. Wait the required acquisition time (if required).
4. Start conversion:
   - Set GO/DONE bit (ADCON0 register)
5. Wait for A/D conversion to complete, by either:
   - Polling for the GO/DONE bit to be cleared

   OR
   - Waiting for the A/D interrupt
6. Read A/D Result registers (ADRESH:ADRESL); clear bit ADIF, if required.
7. For next conversion, go to step 1 or step 2, as required.

If you use the interrupt to read out the result of the A/D convertor, check the interrupt configuration registers to allow the interrupt logic to cause action. Do not forget to copy the result to another variable or memory location, otherwise it will be overwritten in the next conversion. After that in the interrupt routine you should restart A/D by setting GO/DONE bit.

PWM generation using CCP module

The PIC18F25k50 has a built-in Pulse Width Modulation (PWM) generator. The PWM is based on the TMR2 and one of the CCP modules (the user can normally select either CCP1 or CCP2). For this lab, please use CCP2, and set up the PIC so that pin RC1 is the PWM output.

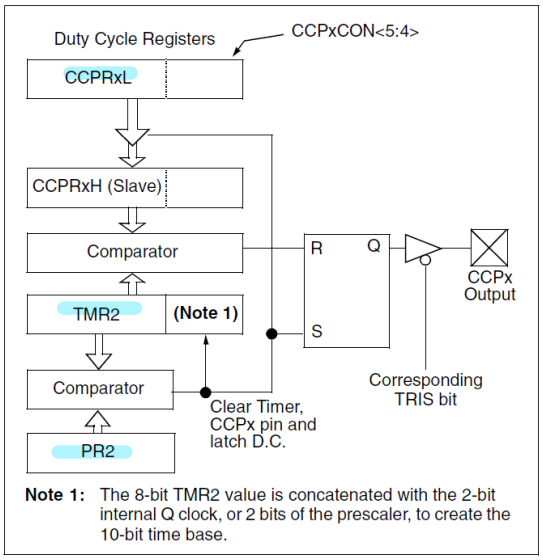


Figure 1: Simplified PWM block diagram

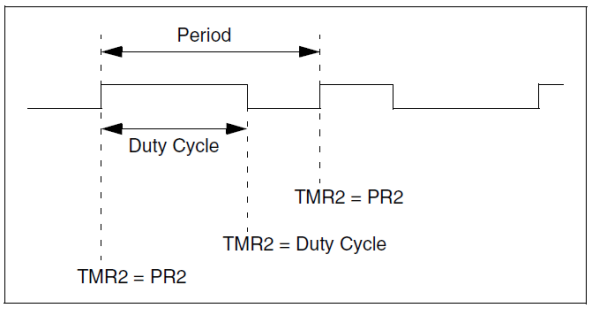


Figure 2: PWM output

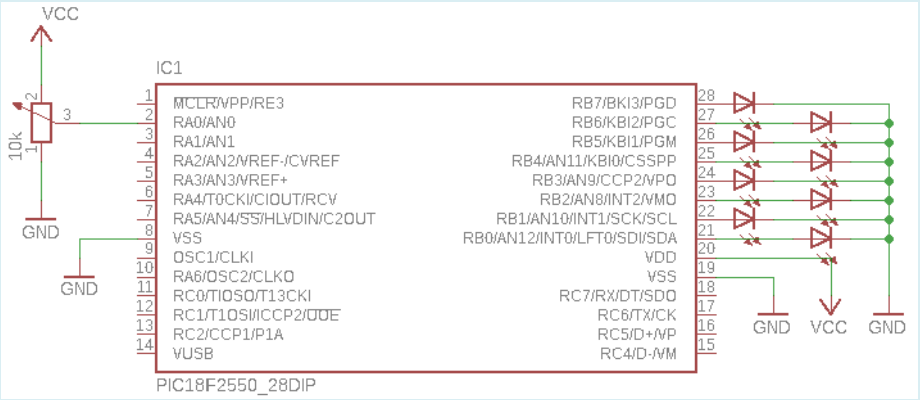
To configure CCP2 module, CCP2CON (Standard CCPx control Register) need to be initialized. T2CON is used to configure Timer 2. The PWM duty cycle is specified by writing to the CCPR2L register. When the CCPR2L matches TMR2, the CCP2 pin will be cleared. A PWM output has a period and a time that the output stays high (duty cycle). PR2 value represents the period of PWM. CCPR2L shows the duty cycle of the signal. Be careful CCPR2L < PR2!
The PWM period can be calculated using the following formula:

$$\text{PWM Period} = [(\text{PR2}) + 1] \cdot 4 \cdot \text{TOSC} \cdot (\text{TMR2 Prescale Value})$$

Assignment "AD conversion & Signal generation":

AD conversion

Read in analog data through RA0, show the digital (binary) result on PORTB. Connect 8 LED's to PORTB. Connect a potentiometer to RA0 to create an analog voltage between 0 and 5V.
Use an interrupt routine to take the value from the AD convertor. Use the interrupt of a timer to make a sampling frequency of 1kHz.



Signal generation

Use a 10KHz PWM signal to generate either a triangular waveform or a sawtooth waveform.
Implement a switch input to select a signal.
For the triangular waveform, the PWM duty cycle should change periodically and smoothly from 100% to 0% and back to 100%.
For the sawtooth waveform the PWM duty cycle should change automatically and smoothly from 100% to 0% then go abruptly back to 100%.
Implement a Low Pass Filter to make a smooth signal on the Oscilloscope.

Think about efficiency when you implement the assignment. Try to make use of the interrupt options of ADC, CCP and timer modules!

Assignment "Variable voltage input/output":

Combine both assignments. The microcontroller reads the value from the potentiometer and converts it into PWM duty cycle. Show the result on the oscilloscope, one channel measures the input voltage, the other the output voltage, both before or after the Low Pass Filter.

Assignment "Frequency regulation":

Use the value from the potentiometer to increase/decrease the frequency of the triangular/sawtooth waveform. Show the result on the oscilloscope.

References

MPLAB XC8 C Compiler User's Guide
(http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB_XC8_C_Compiler_User_Guide_for_PIC.pdf)
PIC 18F2250 Data Sheet
(https://ww1.microchip.com/downloads/en/devicedoc/39632c.pdf)