Bulgarian Academy of Sciences. Space Research and Technology Institute.
Aerospace Research in Bulgaria. 28, 2016, Sofia

A RECTANGULAR UNIPOLAR PULSE WIDTH MEASUREMENT BY MEANS OF PIC18F2550 MCU

Konstantin Metodiev

Space Research and Technology Institute – Bulgarian Academy of Sciences
e-mail: komet@space.bas.bg

Abstract

The article examines an approach towards pulse width measurement by means of PIC18F2550 microcontroller unit (MCU). The proposed solution may come into use in process automation where the MCU decides in virtue of the measured quantity, for instance in case of a pulse with modulation. By way of illustration, it is possible to install the MCU on-board an unmanned aerial vehicle (UAV). In this case, the MCU reads a PWM input signal fed by the radio receiver and actuate a terminal mechanism afterwards depending upon the measured duty cycle value.

Special attention is given to the MCU software peculiarities. Additional computer simulation has also been made. The used software was MikroC Pro for PIC and Proteus VMS. The proposed solution has been shown to operate with sufficient precision. The source code is also included in the present article.

1. Introduction

Microcontrollers units (MCU) are vastly useful nowadays. Among many features supported, the ability of MCU to capture edges of a rectangular signal is applicable to many solutions. Should an edge happen to be registered, the MCU generates external interrupt and an interrupt service routine (ISR) is triggered afterwards. Because of this, the MCU no longer has to wait and check whether new impulse has arrived which is the major drawback of the so-called 'polling approach' [1]. Having that said, the possibility of counting time between consecutive rising and falling edges seems feasible.

The main purpose of the present article is to demonstrate the ability of such a simple solution to measure the pulse width automatically, see fig. 1, and then set in motion either a terminal mechanism or a circuit. Consider the emergency parachute on-board the LHK-3M unmanned aerial vehicle (UAV) as an example. By default, the chute is locked by a servo motor. The MCU might be connected parallel to the PWM wire as a sniffer. If a duty cycle value set in advance occurs, this means that the chute has been released. Then the MCU shuts the engine off by triggering a common emitter amplifier (bipolar transistor) and a relay.

2. Materials and methods

The electronic circuit consists of minimum required parts that make the MCU running according to Fig. 2, i.e. a high speed crystal of 20 MHz and two capacitors of 15 pF each. These are said to provide the MCU with stable instruction clock of 5 MHz [2].

The program algorithm is easy to understand. The MCU is initially set to expect a rising edge of the signal. Should this edge occurs, an external interrupt is triggered and the interrupt flag is switched on by the hardware. It is developer's responsibility to clear this flag each time when it is necessary otherwise it would be impossible for the MCU to trigger another interrupt. Having had the rising edge detected, the program code sets the interrupt flag off, starts an internal timer, and then adjusts the corresponding register so as to expect falling edge of the signal. In case of falling edge arrival, the interrupt flag is cleared again, the timer is stopped, and the program is set to expect the next rising edge. The timer value is converted into milliseconds and then it is cleared.

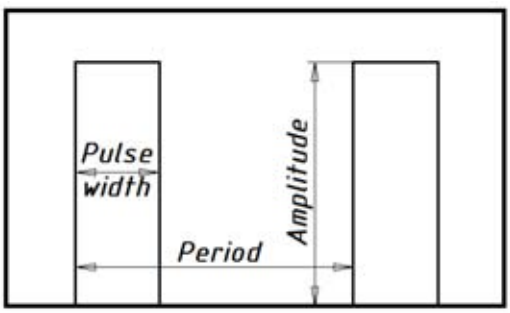

Fig. 1. Basic rectangular pulse definitions used in the article

On the other hand, the source code is a bit more complicated. It is published in the appendix section in the present paper. The code starts with setting port B as digital output and disabling the analog comparator. The former action is solely necessary if the application is to display results on a liquid crystal display (LCD). What follows is a function setting the capture ability of the MCU (see 'configureCapture' function in Appendix). For the present study, one out of two available Capture-Compare-PWM modules has been chosen, i.e. the CCP1. The external impulse is fed to RC2 pin which is why this pin is set as input.

The capture mode is initialized by setting the CCP1CON register. The most significant four bits remain unused in capture mode. The least significant four bits CCP1M<3:0> are set to 0b0101 so that the MCU expects the rising edge. Then, the Timer3 module is picked as a counter. It consists of two eight-bit registers TMR3H and TMR3L. Their values are eventually concatenated to obtain the pulse

width. As a matter of fact, in capture mode, two timers are available: Timer1 and Timer3. Neither of them should be chosen in preference to the other because both offer same input clock prescale values. These are set in corresponding control registers. In the present case study, the chosen prescale value for Timer3 is 1:8 which value is selected by setting the T3CON register accordingly [2]. What follows next is enabling the capture (CCP1IE_bit), peripheral (PEIE_bit), and global (GIE_bit) interrupts. Setting these bits is obligatory so that the ISR may happen. At the end of initialization function, Timer3 and CCP1 storage registers are cleared understandably. In addition, the Timer3 enabling bit (TMR3ON_bit) is set at zero for security reasons.

The ISR (see 'checkExternalInterrupt' function in Appendix) is triggered by the impulse rising edge first in accordance with what is written in the initialization function. If interrupt occurs, hardware sets the CCP1IF bit to 1. As it was mentioned earlier, it is mandatory for this bit to be cleared in the ISR. The next part of the function relies on the developer's resourcefulness. An integer counter variable lets the MCU tell the upcoming edge apart, i.e. whether the edge is rising or falling. The counter solely gets two values, i.e. either 1 or 2. Each time the interrupt has been triggered, the counter is augmented by 1, yet it is cleared after the falling edge. If the counter equals to 1, it is the rising edge coming. This is the right moment to switch the Timer3 module on and set the CCP1M<3:0> bits to 0b0100 so that the MCU no longer looks for the rising edge but the falling. If the counter equals to 2, the falling edge is said to arrive, the CCP1M<3:0> bits are reverted to rising edge, and Timer3 is stopped. Also, a flag is set in order to indicate that Timer3 store registers contain the pulse width. When the capture mode is changed, a false capture interrupt may be generated [2]. The developer has to clear the CCP1IF bit once again. This is a special feature solely observable in the discussed MCU. The reader is referred to 'if condition' in 'checkExternalInterrupt' function, Appendix section.

The pulse width is further converted to seconds in 'get_timer3_capture' function. The function is indispensable because Timer3 module solely stores integers between 0 and 65535. Thus, the pulse width duration is obtained in accordance with following formula:

$$T_{pulse} = \left(CCPR1H << 8 + CCPR1L\right) * \frac{1}{\frac{F_{osc}}{4} Prescaler}, \quad [s]$$

where Prescaler = 1/8, Fosc/4 = 5E+06 is instruction clock and CCPR1 register stores a copy of TMR3 register value. In the end of the function both TMR3 and CCPR1 registers has to be cleared so that a new count may begin. The obtained result unit is seconds. It is wise to remind that both counter and flag variables must be cleared after each falling pulse edge.

3. Results

The proposed circuit has been put to the test as follows. A sequence of rectangular pulses with frequency 50 Hz and pulse duration 1.75 ms enters the CCP1 pin. The obtained results after simulation by means of Proteus VMS are visible in Fig. 2. On the upper left side of the figure is shown a dialog panel for adjustment the input signal parameters. On the bottom side, the result is written down onto a LCD. Both values (1.75 ms) coincide in practice.

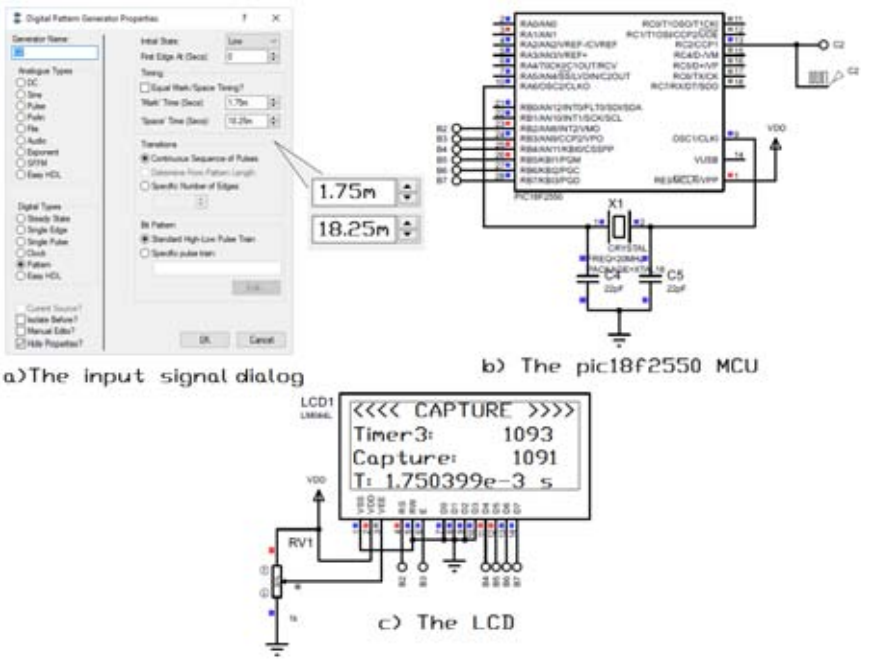

Fig. 2. Project simulation in Proteus VMS

4. Discussion

The proposed test case resembles a PWM signal which is widely used in remotely controlled vehicles for hobbyists. The MCU successfully measures the pulse width. The result is visible on the LCD in Fig. 2 and shows that the error is less than 1 %. This error may grow bigger however if the timer resolution get low. It is highly up to the program developer to decide. Nevertheless, the proposed solution might be used onboard an UAV as an actuator apart from standard articles sold off-the-shelf in hobby RC stores.

In addition, this article might be found useful by developers who are less experienced in the interrupt technique.

References

1. http://www.electronics-base.com/useful-info/software-related/90-polling-vs-interrupt
2. PIC18F2455/2550/4455/4550 Data Sheet, Microchip Technology Inc., 2006.

Appendix: Source code, MikroC Pro for PIC v.6.6.3

```
unsigned int timer3_register;    // stores timer3 register value
unsigned int capture_register;   // stores capture register value
float period;                                  // stores the periodic
time
unsigned short counter = 0;          // edge counter (rising and
falling)
bit flag = 0; // if two edges are detected, frequency & period might
be calculated

void cofigureCapture(void) {
        TRISC.F2 = 1; // input pin
        // Capture mode, every rising edge; 1 – rising edge; 0 –
falling edge
        CCP1M3_bit = 0; CCP1M2_bit = 1; CCP1M1_bit = 0; CCP1M0_bit =
1;
        // Pick up Timer3 as a resource
        T3CCP2_bit = 1; T3CCP1_bit = 1;
        // prescaler 1:8
        T3CKPS1_bit = 1;
        T3CKPS0_bit = 1;
        CCP1IE_bit = 1; // enable capture interrupt
        // enable all interrupts
        PEIE_bit = 1;
        GIE_bit = 1;
        // reset high & low bytes of timer3 & capture registers
        TMR3H = 0; TMR3L = 0;
        CCPR1H = 0; CCPR1L = 0;
        TMR3ON_bit = 0;
        flag = 0; // might be omitted
        return;
}

void get_timer3_capture(void) {
       //get high & low bytes of timer3 & capture registers
       timer3_register = (TMR3H<<8) + TMR3L;
       capture_register = (CCPR1H<<8) + CCPR1L;
       //calculate period
       period = (float)(capture_register + 1);
       period = (float)period * (4);
       period = (float)period * 1/20000000;
       period = (float)period * 8;
       //reset timer3 & capture & period
       TMR3H = 0; TMR3L = 0;
       CCPR1H = 0; CCPR1L = 0;
```

```
       return;
}
void checkExternalInterrupt(void) {
        if (CCP1IF_bit == 1) {    // if capture interrupt occurred
(rising edge detected)
           CCP1IF_bit = 0;          // reset capture interrupt flag
           TMR3ON_bit = 0;       // stop timer1
           counter++;                  // counter increment by 1
           if (counter == 1) {       // if cnt = 1 then rising edge
has been detected
              TMR3ON_bit = 1;   // timer3 start counting
              // look for falling edge
//            CCP1M3_bit = 0; CCP1M2_bit = 1; CCP1M1_bit = 0;
              CCP1M0_bit = 0;
              CCP1IF_bit = 0;         // obligatory for pic18f2550
only
           } // if
           if (counter == 2) {       // if cnt = 2 then falling edge
has been detected
              TMR3ON_bit = 0;   // stop timer3
              flag = 1;         // set this flag to indicate that
pulse duration has been counted
              // look for rising edge
//            CCP1M3_bit = 0; CCP1M2_bit = 1; CCP1M1_bit = 0;
              CCP1M0_bit = 1;
              CCP1IF_bit = 0;      // obligatory for pic18f2550 only
           } // if
     } // if
     return;
}

void interrupt(void) { checkExternalInterrupt(); }

void capturePulseWidth(void) {
        if (flag == 1) {              // if pulse duration has been
counted
        get_timer3_capture();   // go and get results
        counter = 0;                     // then reset rising edge
counter
        flag = 0;                        // also reset this flag
        } // if
    return;
}

void main(void) {
       TRISB = 0;                        // all output
       CMCON = 0x07;               // disable comparators
       ADCON1 = 0b00001111; ADCON0 = 0b00101100; // all digital
       cofigureCapture();
       while(1) { capturePulseWidth(); }
       return; }
```

ИЗМЕРВАНЕ НА ШИРОЧИНАТА НА ПРАВОЪГЪЛЕН ЕДНОПОЛЯРЕН ИМПУЛС С ПОМОЩТА НА МИКРОКОНТРОЛЕР PIC18F2550

К. Методиев

Резюме

В настоящия доклад се разглежда подход за измерване на широчината на правоъгълен импулс чрез микрокотролер PIC18F2550. Предложеното решение може да се използва в автоматизиран процес, където микроконтролерът взема решение въз основа на измерената величина, например при широчинноимпулсна модулация. Възможно е устройството да се инсталира на борда на безпилотен летателен апарат. В случая контролерът чете широчинноимпулсно модулиран сигнал, подаван на входа от радиоприемника и въз основа на измерения коефициент на запълване задейства изпълнителен механизъм.

Специално внимание е отделено на особеностите на програмата за микроконтролера. Допълнително е направена компютърна симулация. Използваният софтуер е MikroC Pro for PIC и Proteus VMS. Показано е, че предложеното решение функционира със задоволителна точност. Кодът на програмата също е публикуван в настоящия доклад.