通过 arduino 读取 PWM 的三种方法

作者：okhaoba
原文链接：https://ld246.com/article/1488766567430
来源网站：链滴

<h2 id="toc_h2_0">Three Ways To Read A PWM Signal With Arduino</h2>
PWM (Pulse-Width Modulation) is a modulation technique that controls the width of the pulse based on modulato signal information. PWM can be used to encode information for transmission or to co trol of the power supplied to electrical devices such as motors.
Generating a PWM signal with an Arduino is quite easy. There is significantly less documentation on how best to read a PWM signal. I needed to read the receiver signals for a remote controlled Quadcopter and aft r doing some research, I discovered three methods of reading a PWM signal with an Arduino. /p>
<h3 id="toc_h3_1">The Gist</h3>
PWM works by varying the width of the on signal (read Duty Cycle) within a fixed signal f equency or period of time. So what we are really looking for is the length of time the signal r mains high for each cycle. There are several ways to do this. The easiest is using the pulseIn f nction as shown below.
<h4 id="toc_h4_2">1. The pulseIn() Function</h4>
The pulseIn() waits for the pin to go HIGH, starts timing, then waits for the pin to go LOW and stops timing. Returns the length of the pulse in microseconds.

<table><caption>The pulseIn Function</caption>
<tbody>
<tr>
<td>

<code>byte PWM_PIN = 3;</code>

<code>int</code> <code>pwm_value;</code>

<code>void</code> <code>setup() {</code>
<code> </code><code>pinMode(PWM_PIN, INPUT);</code>
<code> </code><code>Serial.begin(115200);</code>
<code>}</code>

<code>void</code> <code>loop() {</code>
<code> </code><code>pwm_value = pulseIn(PWM_PIN, HIGH);</code

<code> </code><code>Serial.println(pwm_value);</code>
<code>}</code>

</td>
</tr>
</tbody>
</table>

<h4 id="toc_h4_3">2. External Interrupts</h4>
The pulseIn function works well and is really simple. However, the downside is that the p ocessor cannot be used while it is waiting for the pin to go low. This is not a very efficient use of our CPU cycles. We can improve this by using an event-driven interrupt system to handle t e measurement of the PWM signal. Arduino provides the <a href="https://ld246.com/f rward?goto=http%3A%2F%2Farduino.cc%2Fen%2FReference%2FattachInterrupt" target="_bl

nk" rel="nofollow ugc">attachInterrupt function to do just this.
Most Arduino boards have two external interrupts: numbers 0 (on digital pin 2) and 1 (on digital pin 3). These interrupts can be set to trigger on RISING or FALLING signal edges or on low level. Once attached, when an interrupt is triggered, the specified interrupt s rvice routine (ISR) will be called.
Note the use of volatile variables in this sketch. The Arduino docs state: A variable should be declared volatile whenever its alue can be changed by something beyond the control of the code section in which it appears such as a concurrently executing thread. In the Arduino, the only place that this is likely to oc ur is in sections of code associated with interrupts, called an interrupt service routine.

<table><caption>External Interrupts</caption>
<tbody>
<tr>
<td>

<code>volatile</code> <code>int</code> <code>pwm_value = 0;</code>
<code>volatile</code> <code>int</code> <code>prev_time = 0;</code>

<code>void</code> <code>setup() {</code>
<code> </code><code>Serial.begin(115200);</code>
<code> </code><code>// when pin D2 goes high, call the rising functio </code>
<code> </code><code>attachInterrupt(0, rising, RISING);</code></div

<code>}</code>

<code>void</code> <code>loop() { }</code>

<code>void</code> <code>rising() {</code>
<code> </code><code>attachInterrupt(0, falling, FALLING);</code></d v>
<code> </code><code>prev_time = micros();</code>
<code>}</code>

<code>void</code> <code>falling() {</code>
<code> </code><code>attachInterrupt(0, rising, RISING);</code></div

<code> </code><code>pwm_value = micros()-prev_time;</code></div

<code> </code><code>Serial.println(pwm_value);</code>
<code>}</code>

</td>
</tr>
</tbody>
</table>

<h4 id="toc_h4_4">3. Pin Change Interrupts</h4>
Using attachInterrupt allows for greater efficiency but now we are forced to use pins 2 a d 3 to read the PWM values and we are limited in the number of interrupts we can specify. If

e would like to trigger an interrupt on another pin, we need to use Pin Change Interrupts.</p

Pin Change Interrupts can be enabled on any of the Arduinos signal pins. The pin change nterrupts are grouped into 3 ports on the MCU. This means there are only 3 interrupt subrout nes for all 20 pins. This means the subroutine will need to be more complicated as it now nee s to determine which pin triggered the interrupt.
You can configure this manually but the PinChangeInt library makes it very quick and simple.

<table><caption>Pin Change Interrupts</caption>
<tbody>
<tr>
<td>

<code>#include <PinChangeInt.h></code>

<code>#define MY_PIN 5 // we could choose any pin</code>

<code>volatile</code> <code>int</code> <code>pwm_value = 0;</code>
<code>volatile</code> <code>int</code> <code>prev_time = 0;</code>
<code>uint8_t latest_interrupted_pin;</code>

<code>void</code> <code>rising()</code>
<code>{</code>
<code> </code><code>latest_interrupted_pin=PCintPort::arduinoPin;</ ode>
<code> </code><code>PCintPort::attachInterrupt(latest_interrupted_pin, &falling, FALLING);</code>
<code> </code><code>prev_time = micros();</code>
<code>}</code>

<code>void</code> <code>falling() {</code>
<code> </code><code>latest_interrupted_pin=PCintPort::arduinoPin;</ ode>
<code> </code><code>PCintPort::attachInterrupt(latest_interrupted_pin, &rising, RISING);</code>
<code> </code><code>pwm_value = micros()-prev_time;</code></div

<code> </code><code>Serial.println(state);</code>
<code>}</code>

<code>void</code> <code>setup() {</code>
<code> </code><code>pinMode(MY_PIN, INPUT); digitalWrite(MY_PIN, HIGH);</code>
<code> </code><code>Serial.begin(115200);</code>
<code> </code><code>PCintPort::attachInterrupt(MY_PIN, &rising, RISING);</code>
<code>}</code>

<code>void</code> <code>loop() { }</code>

</td>

</tr>
</tbody>
</table>

原文出处：http://www.benripley.com/diy/arduino/three-ways-to-read-a-pwm-signal-with arduino/

PWM (<a href="https://ld246.com/forward?goto=http%3A%2F%2Fen.wikipedia.org%2Fwki%2FPulse-width_modulation" target="_blank" rel="nofollow ugc">Pulse-Width Modulation /a>) is a modulation technique that controls the width of the pulse based on modulato signal information. PWM can be used to encode information for transmission or to co trol of the power supplied to electrical devices such as motors.
Generating a PWM signal with an Arduino is quite easy. There is significantly less documentation on how best to read a PWM signal. I needed to read the receiver signals for a remote controlled Quadcopter and aft r doing some research, I discovered three methods of reading a PWM signal with an Arduino. /p>
<h3 id="toc_h3_5">The Gist</h3>
PWM works by varying the width of the on signal (read Duty Cycle) within a fixed signal f equency or period of time. So what we are really looking for is the length of time the signal r mains high for each cycle. There are several ways to do this. The easiest is using the pulseIn f nction as shown below.
<h4 id="toc_h4_6">1. The pulseIn() Function</h4>
The pulseIn() waits for the pin to go HIGH, starts timing, then waits for the pin to go LOW and stops timing. Returns the length of the pulse in microseconds.

<table><caption>The pulseIn Function</caption>
<tbody>
<tr>
<td>

<code>byte PWM_PIN = 3;</code>

<code>int</code> <code>pwm_value;</code>

<code>void</code> <code>setup() {</code>
<code> </code><code>pinMode(PWM_PIN, INPUT);</code>
<code> </code><code>Serial.begin(115200);</code>
<code>}</code>

<code>void</code> <code>loop() {</code>
<code> </code><code>pwm_value = pulseIn(PWM_PIN, HIGH);</code

<code> </code><code>Serial.println(pwm_value);</code>
<code>}</code>

</td>
</tr>
</tbody>

</table>

<h4 id="toc_h4_7">2. External Interrupts</h4>
The pulseIn function works well and is really simple. However, the downside is that the p ocessor cannot be used while it is waiting for the pin to go low. This is not a very efficient use of our CPU cycles. We can improve this by using an event-driven interrupt system to handle t e measurement of the PWM signal. Arduino provides the attachInterrupt function to do just this.
Most Arduino boards have two external interrupts: numbers 0 (on digital pin 2) and 1 (on digital pin 3). These interrupts can be set to trigger on RISING or FALLING signal edges or on low level. Once attached, when an interrupt is triggered, the specified interrupt s rvice routine (ISR) will be called.
Note the use of volatile variables in this sketch. The Arduino docs state: A variable should be declared volatile whenever its alue can be changed by something beyond the control of the code section in which it appears such as a concurrently executing thread. In the Arduino, the only place that this is likely to oc ur is in sections of code associated with interrupts, called an interrupt service routine.

<table><caption>External Interrupts</caption>
<tbody>
<tr>
<td>

<code>volatile</code> <code>int</code> <code>pwm_value = 0;</code>
<code>volatile</code> <code>int</code> <code>prev_time = 0;</code>

<code>void</code> <code>setup() {</code>
<code> </code><code>Serial.begin(115200);</code>
<code> </code><code>// when pin D2 goes high, call the rising functio </code>
<code> </code><code>attachInterrupt(0, rising, RISING);</code></div

<code>}</code>

<code>void</code> <code>loop() { }</code>

<code>void</code> <code>rising() {</code>
<code> </code><code>attachInterrupt(0, falling, FALLING);</code></d v>
<code> </code><code>prev_time = micros();</code>
<code>}</code>

<code>void</code> <code>falling() {</code>
<code> </code><code>attachInterrupt(0, rising, RISING);</code></div

<code> </code><code>pwm_value = micros()-prev_time;</code></div

<code> </code><code>Serial.println(pwm_value);</code>
<code>}</code>

</td>
</tr>
</tbody>
</table>

<h4 id="toc_h4_8">3. Pin Change Interrupts</h4>
Using attachInterrupt allows for greater efficiency but now we are forced to use pins 2 a
d 3 to read the PWM values and we are limited in the number of interrupts we can specify. If
e would like to trigger an interrupt on another pin, we need to use Pin Change Interrupts.</p

Pin Change Interrupts can be enabled on any of the Arduinos signal pins. The pin change
nterrupts are grouped into 3 ports on the MCU. This means there are only 3 interrupt subrout
nes for all 20 pins. This means the subroutine will need to be more complicated as it now nee
s to determine which pin triggered the interrupt.
You can configure this manually but the <a href="https://ld246.com/forward?got
=http%3A%2F%2Fplayground.arduino.cc%2FMain%2FPinChangeInt" target="_blank" rel="no
ollow ugc">PinChangeInt library makes it very quick and simple.

<table><caption>Pin Change Interrupts</caption>
<tbody>
<tr>
<td>

<code>#include <PinChangeInt.h></code>

<code>#define MY_PIN 5 // we could choose any pin</code>

<code>volatile</code> <code>int</code> <code>pwm_value = 0;</code>
<code>volatile</code> <code>int</code> <code>prev_time = 0;</code>
<code>uint8_t latest_interrupted_pin;</code>

<code>void</code> <code>rising()</code>
<code>{</code>
<code> </code><code>latest_interrupted_pin=PCintPort::arduinoPin;</
ode>
<code> </code><code>PCintPort::attachInterrupt(latest_interrupted_pin,
&falling, FALLING);</code>
<code> </code><code>prev_time = micros();</code>
<code>}</code>

<code>void</code> <code>falling() {</code>
<code> </code><code>latest_interrupted_pin=PCintPort::arduinoPin;</
ode>
<code> </code><code>PCintPort::attachInterrupt(latest_interrupted_pin,
&rising, RISING);</code>
<code> </code><code>pwm_value = micros()-prev_time;</code></div

<code> </code><code>Serial.println(state);</code>
<code>}</code>

<code>void</code> <code>setup() {</code>
<code> </code><code>pinMode(MY_PIN, INPUT); digitalWrite(MY_PIN,

HIGH);</code>
<code> </code><code>Serial.begin(115200);</code>
<code> </code><code>PCintPort::attachInterrupt(MY_PIN, &rising, RISING);</code>
<code>}</code>

<code>void</code> <code>loop() { }</code>

</td>
</tr>
</tbody>
</table>