Multi-channel Software-Oriented Pulse Width Modulation (SPWM)

Marwan Sleiman
Department of Computer Science & Engineering
University of Connecticut
371 Fairfield Road Unit 2155, Storrs, CT 06269
Tel: 1-860-486-0005 Fax: 1-860-486-4817
Email: marwan@engr.uconn.edu

I. Abstract

In this paper, we propose a software-oriented method for generating multiple pulse width modulation channels from the parallel port of a computer without using any additional interface. We have developed an algorithm to modulate different signals that may come from different sources and generate the corresponding modulated signals on the parallel port of a computer. Our approach is tested in real-world applications, and the successful results open the door for relying more and more on software for the different types of modulation and various telecommunication systems like software radio.

Keywords: Software Modulation, Pulse Width Modulation

II. Introduction

Using software to perform the classical tasks of analog circuits in communication systems has got attention for about 10 years [13]. The quick advance of embedded systems makes the use of software-driven modulation very promising and cheap so we decided to integrate the Pulse Width Modulation (PWM) in the embedded systems rather than using external analog circuits.

Pulse width modulation (PWM) [1], also known as pulse duration modulation (PDM) and pulse length modulation (PLM), has a wide range of applications: it can be used in embedded control systems in the industry, into vehicles, or at home, to control the mechanical equipments that work on servomotors and in DC motor drive to control the speed of the motor. It can be used in digital-to-analog conversion in bit stream DACs for audio CD's, and it is also a powerful technique for controlling analog circuits instead of using a rheostat or voltage regulator. There are several classical methods to generate a PWM modulated signal. The earlier methods used analog electronic circuits. With the advances in hardware design, many hardware controllers, microcontrollers [2], and PLD’s/FPGA’s technologies were adopted to generate PWM signals [3]. In many cases, the source of a PWM signal comes from a digital computer—whether it is a desktop or a PC/104 [4] in an embedded control system. For example PC/104 is being widely used in industry as PID [5] (Proportional Integral and Derivative) servo controller and is replacing Programmable Logic Controllers (PLC). Considering that the classical methods face the problems of noise vulnerability, extra hardware space and extra power supply, in this paper, we propose a new technique to generate the PWM signals directly from the parallel port of a computer or other computing device without using any additional hardware interface. We call this approach Software-oriented Pulse Width Modulation, SPWM, since it is based on a software algorithm. The algorithm is able to generate eight simultaneous PWM channels that appear on the parallel port of the computer. We also test and validate SPWM, and obtain very promising results.

The rest of this paper is organized as follows. In Section II, we give a short overview of PWM and the classical generation methods. In Section III, we then introduce our new technique and describe our modulation algorithm. Finally, we validate our work and discuss the pros and cons of our method in Section IV and V, followed by a brief summary and conclusion in Section VI.

II. Background

Pulse width modulation (PWM) is a method of modulation for digitally encoding analog signal levels. As shown in Fig.1, the period of a PWM signal, T, consists of two parts: duty cycle, or on-time, \( T_H \); and off-time, \( T_L \). \( T_H \) corresponds to a constant full DC supply of +5 (or sometimes +12) Volts, while \( T_L \) corresponds to 0 Volts. The information (denoted by A) to be modulated and transmitted is carried by the width of the pulse of the modulated signal (\( T_H \)), i.e., \( A = f(T_H) \). Given a sufficient bandwidth (where the sampling frequency \( F_s \) is greater than or equal to twice the highest frequency component of the analog signal) [1], any analog value can be encoded with PWM.

\[
T = T_L + T_H
\]

Fig.1. Sample PWM signal
In the literature, there are two classical approaches to generate PWM signals:

1. Analog Approach: One of the most popular analog methods uses a triangular carrier wave modulated by a sine wave and the points of intersection determine the output signal. The triangular wave can be a saw-tooth generator or a ramp carrier. This modulation is implemented by using a comparator that switches on when one input is greater than the other as shown in Fig. 2.

   ![Analog Sine PWM](image)

   Fig. 2: Analog Sine PWM

   This method is simple and easy, but the equipment takes space in embedded control systems, where more than one signal is usually required. Moreover, an analog circuit may not always be the optimum solution because it tends to drift over time and can, therefore, be very difficult to tune. Precision analog circuits, which solve that problem, can be very large, heavy, and expensive. Analog circuits can also get very hot; the power dissipated is proportional to the voltage across the active elements multiplied by the current through them. In addition, analog circuitry is usually sensitive to noise. Because of its infinite resolution, any perturbation or noise on an analog signal necessarily changes the current value.

2. Digital Approach: In this type of method, by using hardware controllers to control analog circuits digitally, system costs and power consumption are drastically reduced. Furthermore, many micro-controllers and DSP’s already include on-chip PWM controllers, making implementation easy. For example, Microchip’s PIC16C67 includes two PWM controllers [6], [10], each of which has a selectable on-time (T_on) and period (T). To start PWM operation, the code of the microcontroller sets the period of an on-up counter that provides the modulating square wave and sets the on-time in a PWM control register.

   PWM can also be generated from a programmable logic device by using a register and a down counter. The pulse width is readily scaled up or down simply by changing the width of the register and counter [3].

III. SPWM

In this section, we will discuss our new approach, SPWM (Software-oriented Pulse Width Modulation). We will describe the basic system architecture (including hardware and software components), and then elaborate our signal generation algorithms.

A. System Architecture

In SPWM, the PWM signal modulation and generation algorithms are on the same embedded control machine that controls the control channel as shown in Fig. 3.

![Embedded SPWM Control System](image)

Fig. 3 Embedded SPWM Control System

A.1 Hardware

We use a PC that can be the driver for a control system or other applications; this PC can also be an embedded control system such as a PC/104. To generate the desired PWM signal, we use the I/O peripherals of the PC that can be the Serial Port, the Parallel Port; a PCI, ISA, or VESA bus; or the universal serial bus, etc.

Note that, in our prototype design, we use the parallel port because it is the least expensive interface that enables us to take an output from a PC and because it provides us with the output speed we need. In effect the IEEE Std.1284 [11] specified the speed of the parallel port to 2MB/s. We can take from it up to 12 simultaneous outputs, which are, pin 1 (logically inverted), pin 2 to 9 (D0 to D7), pin 14 (logically inverted), pin 16, and pin 17 (logically inverted). The parallel port is shown in Fig. 4. More details about interfacing the parallel port can be found in [7].

```
  \[
  \begin{array}{cccccccccccc}
    1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\
    14 & 15 & 16 & 17 & 18 & \ldots & 25 \\
  \end{array}
  \]
```

Fig. 4. Parallel port pin assignment

A.2 Software

For each channel, the PWM pulse width is generated in real time and stored in an array. The pulse width, T_on is a linear function, g, of the amplitude (information), A, of the modulated signal: T_on= g(A). To obtain the desired outputs from the parallel port, we first need certain commands that depend on the available software. For example in an assembly language, we can use the following code to output a word on the parallel port:

```
mov dx, 0x378
mov ax, num
out dx, ax
```
In this code, 0x378 refers to the physical address of the parallel port of the computer and “num” is the word to be put on the parallel port. In a higher-level language, we can use `outport()` in C/C++, and `portw()` in Pascal for instance.

The kernel of our approach is the signal generation algorithms, which are illustrated next.

**B. Signal Generation Algorithms**

**B.1 Single Channel Signal Generation Algorithm**

The generation of a single pulse consists of the following steps:
1. Put a logical one (+5 V) at the port (or pin) where the signal is expected.
2. Delay for a period that equals the pulse width $T_{pl}$.
3. Put a logical zero at the same port and delay for a period $T_L$ that equals the period minus the pulse width. Then go back to step 1.

In the case that the programming package contains a function. One candidate method is following:

1. Calculate the clock period, $T_c$, of the PC.
2. Use a “for” loop that waits (does nothing) for a period time that equals the desired delay $T_d$: $T_d = n*T_c$. Where $n$ is the argument of the “for” loop.

**B.2 Multiple Channel Signal Generation Algorithm**

If we are using a peripheral that has more than one output pin, we can generate more than one PWM signal at the same time. This is the case of the parallel port where we can generate up to eight PWM signals at the same time from it. However, the procedure is a little bit cumbersome and more difficult than the case of a single channel because the control is done in parallel, i.e. all eight pulses appear simultaneously. This major improvement is essential because in the real world, we may need several simultaneous channels in order to perform different applications at the same time.

Compared with single channel signal modulation, generating multiple channel signals will confront many challenges. One problem is that the eight pulses are not necessarily equal, each corresponding to a different pulse width and, consequently a different time delay. We propose the following solution: at the beginning of each period we assert all eight lines (i.e. pins, or channels) with logical one, or +5 V, which corresponds to the beginning of the pulse, then consecutively clear the lines according to the pulse width. To clear the line, we set a zero on the corresponding pin. For each period, we proceed as follows: first, we sort the pulses in increasing order (based on pulse width) and keep track of the position of each one on the parallel port. We mean by position the corresponding pin assignment for each channel. At the beginning of each period, we output the hexadecimal number FF (11111111 in binary), which asserts +5 Volts to all eight lines simultaneously. Next, we look at which channel has the next smallest pulse width and clear its line (pin) accordingly after a delay corresponding to the duration of this pulse. The clearing is done by performing a logical “and” operation of the output of the parallel port with the corresponding word. This word, that we call control word, is chosen to be a hexadecimal number which is the equivalent of an 8-bit binary number. The 8-bit binary number consists of a series of ones except a zero in the position of the desired channel to be cleared. For example, if we want to clear channel 2, then we must “and” the output of the parallel port with the hexadecimal number FD (11111101 in binary), whereas for channel 4 we use F7 (11110111 in binary) and so on. The control words are shown in table 1.

<table>
<thead>
<tr>
<th>Pin to be cleared</th>
<th>Parallel port pin assignment</th>
<th>Control word in Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11111110</td>
<td>FE</td>
</tr>
<tr>
<td>2</td>
<td>11111101</td>
<td>FD</td>
</tr>
<tr>
<td>3</td>
<td>11111011</td>
<td>FB</td>
</tr>
<tr>
<td>4</td>
<td>11101011</td>
<td>F7</td>
</tr>
<tr>
<td>5</td>
<td>11010111</td>
<td>EF</td>
</tr>
<tr>
<td>6</td>
<td>10101111</td>
<td>DF</td>
</tr>
<tr>
<td>7</td>
<td>01111111</td>
<td>BF</td>
</tr>
<tr>
<td>8</td>
<td>00111111</td>
<td>7F</td>
</tr>
</tbody>
</table>

After clearing all the pins, we have all zeros on the parallel port, so delay until the end of the PWM period and then we repeat the above steps for new periods.

The algorithm that summarizes the SPWM procedure is shown in Fig. 5.

**IV. Test and Validation**

We implemented the SPWM algorithm on a Dell Pentium IV PC using a C++ compiler. We also developed a graphical user interface that allows the user to change the pulse width using a Scroll Bar to the angle of control eight RC servomotors [8]. The output of the parallel port was observed on an oscilloscope and the results matched the desired PWM signal. We varied the PWM pulse width, $T_{pl}$, between 1.2 and 2.4 ms and the period, $T$, of the PWM signals was 14 ms. A sample SPWM channel output is shown in Fig. 6. The response time of the program, which is of the order of few nanoseconds, was not noticeable because it is of the order of few nanoseconds and is
negligible in the range used in embedded control systems. The shape of the observed channels didn’t show any deviation or delay from the desired signal. We also tested our techniques on eight Radio Control (RC) servo-motors without using any additional buffers, interfaces, or controllers [8] and we had the motors work precisely.

![Flow Chart of the SPWM algorithm](image)

**Fig. 5. Flow Chart of the SPWM algorithm**

### V. Discussions

Our technique has many advantages over the classical ones: SPWM allows the PC to directly control many applications at home and in industry at the same time without the overhead of the addition of extra hardware. The absence of additional hardware makes it more immune to noise. SPWM is flexible, as we can easily modify the signal being generated by software without the need of operating the hardware. Finally, SPWM is less power consuming because we do not have to install any additional electronic parts to generate it.

![Observed PWM channel](image)

**Fig. 6. Observed PWM channel**

Although you may think that our software algorithm may have some overhead delay because we have to sort the eight signal pulses in the beginning and at each signal change, this does not affect the shape of the generated signals because the sorting time is of the order of few clock cycles and it causes a negligible delay compared to the pulse width. In effect it has been shown [10] that the time complexity of a quick sort takes an execution time of the order of $8 \log(8) = 7.23$ instruction cycle, which is very small compared to a practical pulse width and to the speed time of a parallel port [11]. It was specified in [11] that the maximum bit bandwidth of the parallel port is less than 2MB/s, which corresponds to a minimum PWM limitation in our approach of few microseconds for the pulse width. Anyway, the delay of our program is much less than that resulting from the addition of an external Microcontroller because the typical minimum instruction cycle time of a microcontroller is 200ns [13].

The output given by the parallel ports of any PC is digital: a logical zero corresponds to 0 V DC, and a logical one corresponds to +5 V DC. The intensity of the current that can be taken from this port is 2.6 mA [12], so we may add some buffers or converters for some power consuming applications that are power consuming. In this case, PWM output should be buffered with a unity-gain op-amp circuit.

Please note that, in this paper, we assume that every channel has the same period as the remaining seven. In the case that all of the eight channels do not have the same periods, we need to expand our algorithm to support variable periods.

### VI. Summary and Conclusion

In this work, we presented SPWM: a new approach to generate multiple PWM channels from the parallel port of
We developed an algorithm to sort the pulse widths and control the output of the parallel port to give the desired results. With the drop in the cost of embedded systems, SPWM is now more economical, space saving, noise immune, and works well for embedded control systems. The success of alternative open the door for potential deployment of many Software modulation and demodulation techniques in embedded systems to replace the classical modulation types currently in use.

References


Biography

Marwan Sleiman received his BS and MS in computer engineering degrees University of Balamand, Lebanon, in 1997 and 1999 respectively. He worked as a computer and telecom engineer for several years before joining University of Connecticut to enroll in the PHD program in computer science and engineering in January 2004.