AVR 8-bit Microcontrollers

AVR136: Low-Jitter Multi-Channel Software PWM

APPLICATION NOTE

Introduction

Generation of stable and accurate pulse width modulated (PWM) signals on an AVR can easily be achieved using the waveform generator modes of the timer peripherals. However, only a limited number of channels can be implemented. If a large number of PWM channels are required then a software solution must be used, and the intention of this application note is to demonstrate a method of providing this whilst maintaining very low jitter on the PWM signals.

Software generation of PWM requires a certain amount of processing time to manage the signal level decision process, so it is only suitable for low base-frequency waveform generation. Signals of this type are suitable for DC control applications such as LED or lamp intensity control, brush-type DC motor speed control, analogue meter driving, or any situation requiring a low-dynamic DC control voltage.

A working implementation written in C is included with this application note. Full documentation of the source code and compilation information is found by opening the ‘readme.html’ file included with the source code.

Features

- Suitable for any Atmel® AVR® microcontroller with a free 8-bit timer
- Non-Cumulative jitter of ±0.015% of base frequency
- Up to 24 PWM channels using the code sample supplied
- For lamp intensity control, dc motor speed control, analogue meter drive, etc.
- Full accompanying source code written in C

Table of Contents

1. Principles

The software example presented here demonstrates the generation of ten PWM channels on an ATtiny2313, but is equally applicable to any other AVR with an 8-bit timer capable of generating an overflow interrupt. Low jitter is achieved by having the timer overflow as the only enabled interrupt, and by having the output signals updated during the first instructions in the interrupt service routine (ISR). This makes the execution tempo very predictable, the only jitter being variations in interrupt response time which depends on the instruction being executed at the exact moment the interrupt occurs, giving a typical jitter of ±1 clock cycle. With the full PWM cycle time being 65536 clock cycles, jitter is therefore ±0.0015% of the PWM base frequency and is non-cumulative over time.

The general principle of the software PWM is to mimic the operation of the hardware timers in PWM mode. An array of ‘compare’ values is established with elements set to the required PWM pulse widths, and a complementary ‘compbuff’ array is used to double-buffer any compare array update, ensuring consistent PWM operation. An 8-bit timer is initialized to count the main clock and generate an interrupt on overflow, so an interrupt occurs once every 256-clock cycles. This means the ISR must complete in less than 256 cycles to maintain the low jitter specification. An 8-bit soft counter is incremented during each ISR to act as a position indicator within the PWM cycle, giving a PWM resolution of 1/256 or ~0.4%, and an overall PWM base frequency of main clock / (256 * 256).

2. Interrupt Service Routine

A logical approach for the ISR would be to increment the soft counter, determine which PWM signals should change state at that position of the PWM cycle, then implement the changes. The problem with this is that the time taken between the start of the ISR and the pin state change will vary considerably depending on the results of the PWM position tests, causing significant jitter. To eliminate this the ISR performs the pin state update immediately, then carries out the increment and position tests to prepare the state values for the start of the next interrupt cycle. On overflow of the soft counter all pin conditions are prepared for being set high and the compare values are updated with any changes made to the compbuff values. If a compare value is zero the pin condition indicator will be returned to zero, so a PWM value of zero will give an output that is permanently low. The maximum PWM pulse width will be 255/256 of the base frequency period.

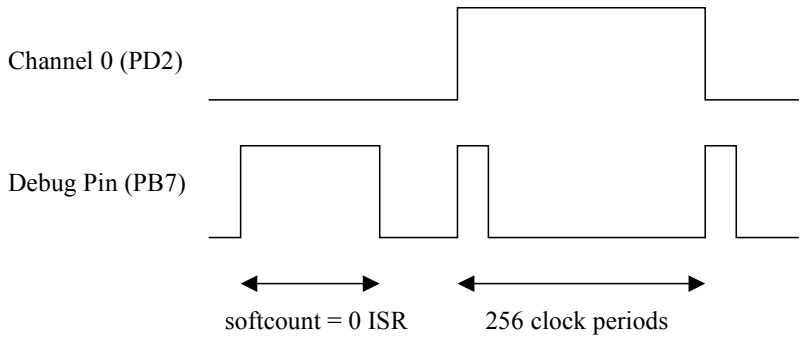
3. DEBUG Operation

In order to maintain the specified jitter performance, the ISR must complete within 256 clock cycles. The worst-case situation occurs in the softcount = 0 ISR when all but one channels have a compare value of zero, so the DEBUG option has been included to allow checking of the ISR timing when channel quantity has been modified.

Note: All channels being zero is theoretically worse, but as no channel is producing a PWM pulse any timing overflow will not be seen

Changing the DEBUG define to 1 allows the approximate time taken by the ISR to be measured by Timer 1, with the default channel settings changed to give the worst-case conditions, and the Timer 1 result is frequently displayed on an attached RS232 terminal. Ideally the ISR time displayed should be well below 0x00FF. A debug pin is also enabled allowing an oscilloscope to be used to view the ISR timing, with a typical waveform shown in the figure below.

Figure 3-1. Typical Waveform



The DEBUG option allows experimentation within the ISR to be monitored. The speed of the ISR is critical when trying to expand the number of channels available, so optimization or replacement of the code provided can be examined with DEBUG to determine if improvement has been made.

4. Control Interface

When not in DEBUG mode, a sample application is provided to allow manual control of the PWM channels via an RS232 terminal. Commands are sent to the ATtiny2313 in the ASCII format:

```
#nHH
```

where:

```
# is a synchronization character
n = 0-9 indicating the required channel number
HH = hex value corresponding to 0-255 for PWM high period
```

A successful command receives the response “OK” whereas an incorrect command will respond with “ERR”. This protocol could easily be adapted to allow the ATtiny2313 to be used as a slave PWM controller attached to another microcontroller and to allow more channels to be controlled.

5. Enhancements

The maximum number of channels that can be maintained is dependent on ISR efficiency and time slot allocated for the ISR. The ISR code efficiency will vary between compilers and will also be affected by the automatic optimization level applied at compile time. This gives great scope for experimentation to improve and optimize the ISR to increase the maximum channel limit, the sample code supplied has been written to be easy to understand rather than being the best possible solution.

ISR time slot is fixed at 256 clock cycles when using an 8-bit timer, however making use of a 16-bit timer in CTC mode could increase the time slot. For example, having a timer reset at 512 clock cycles and doubling the crystal frequency would allow at least twice the number of channels with the same PWM base frequency.

6. Revision History

Doc Rev.	Date	Comments
8020B	08/2016	New template and some minor changes
8020A	05/2006	Initial document release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**