SGS-THOMSON MICROELECTRONICS

APPLICATION NOTE

PWM Generation with ST62 Auto-reload Timer

J.NICOLAI

INTRODUCTION

This note presents how to use the ST62 Auto-reload Timer (ARTimer) for the generation of a PWM signal tunable in frequency and duty cycle. As example, the generation of a 30kHz PWM signal with duty cycle proportional to an analog input voltage is presented.

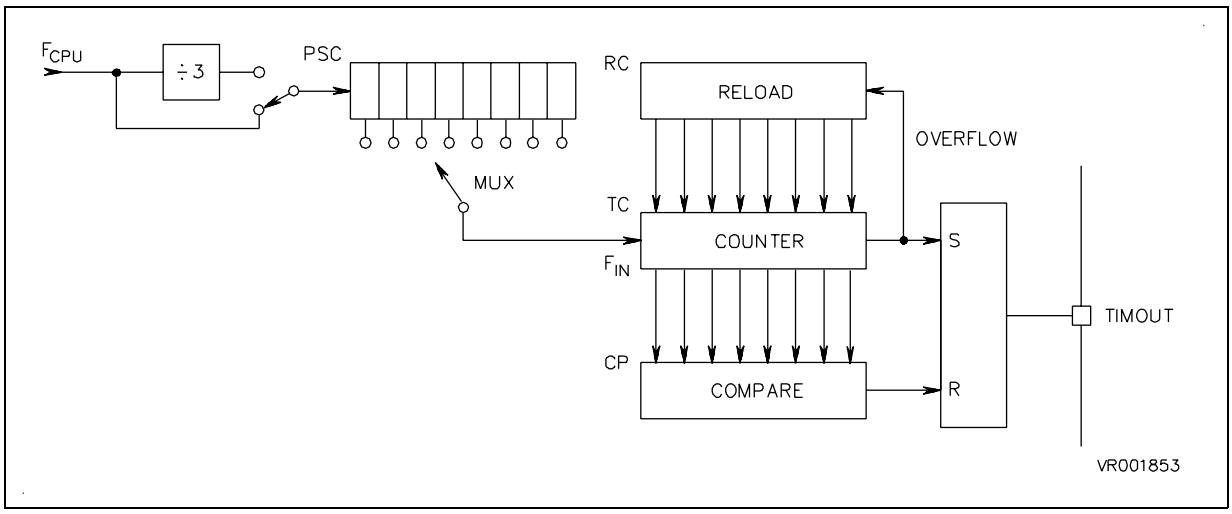
Auto-reload Timer description

This timer is an 8 bit timer/counter with prescaler. It includes auto-reload PWM, capture and compare capability with one input and one output pins. It is controlled by the following registers (8 bit):

- Mode Control Register (MC)
- Status registers (SC0, SC1)
- Load register (LR)
- Incremental counter register (TC)
- Compare register (CP)
- Reload/Capture register (RC)

It can also wake the MCU from wait mode and exit from stop mode if an external event is present on the input pin. The prescaler ratio can be programmed to choose the timer input frequency F_{IN} (see Table 1).

Figure 1. Auto-reload Timer Block Diagram



Pulse Width Modulation (PWM) Generation

Using the PWM generation capability of the AR-Timer, the CPU of the microcontroller has only to start/stop the timer and update the duty cycle. High speed PWM signals in the range of 100kHz can be generated. The timer clock input frequency F_{IN} can be selected by the oscillator clock f_{osc} and the prescaler ratio. The PWM signal period is controlled by the Reload register. The duty cycle is defined by the Compare register CP (see Figure 2).

The register TC is incremented from RC to 255d, then reloaded at RC to count again. The PWM output is set on the overflow of TC and reset when TC=CP.

CP can have any value between RC and 255d. So RC should be minimal and the prescaler ratio as small as possible to achieve a maximum resolution.

Major formulae for PWM generation are:

$F_{IN} = f_{osc}$ / (prescaler ratio)

$F_{PWM} = F_{IN}$ / (255 - RC)

Duty cycle: (CP -RC) / (255 - RC)

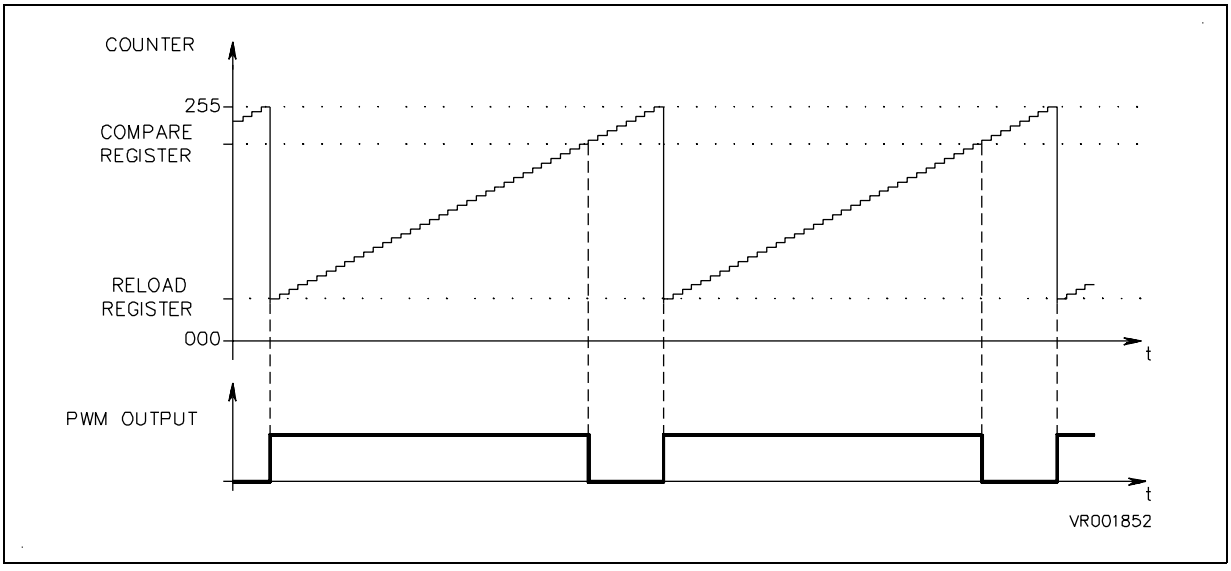
Resolution: 1 / (255 - RC)

With a 8MHz oscillator and a prescaler ratio of 1, the maximum resolution (1/255) leads to a PWM frequency F_{PWM} of 31.3kHz. A resolution of 1/64 allows to increase F_{PWM} to 125kHz.

Table 1. Prescaler Programming Ratio

Bit 0 Reg. SC1	PS2	PS1	PS0	PRESCALER Ratio
0	0	0	0	1
0	0	0	1	2
0	0	1	0	4
0	0	1	1	8
0	1	0	0	16
0	1	0	1	32
0	1	1	0	64
0	1	1	1	128
1	0	0	0	3
1	0	0	1	6
1	0	1	0	12
1	0	1	1	24
1	1	0	0	48
1	1	0	1	96
1	1	1	0	192
1	1	1	1	384

Figure 2. PWM Timer Operation



Example 1:

Target: Generate a 12kHz PWM signal with a duty cycle of 37%, using an oscillator frequency of 4MHz:

We want to generate a periodic signal at a frequency of 12 kHz and a duty cycle of 37% .

The CPU frequency (quartz frequency) is 4 MHz.

Let's try with a prescaler ratio of 3:

F_{IN} = 4 MHz / 3 = 1333.33 kHz

12 kHz = 1333.33 kHz / (255d - RC)

gives RC = 143.889, rounded to 144

Resolution = 1 /(255 - 144) = 1 / 111

Duty cycle desired: 37%

0.37 = (CP - 144) / (255 - 144) yields CP = 185.07, rounded to 185.

Summary: prescaler = 3, RC = 144d, CP = 185d: these numbers yield a PWM frequency of 12.012 kHz, a duty cycle of 36.94 % and a resolution of 0.9 %, which is very close to the initial goal.

Program example:

- SC0 is not programmed, so it keeps its reset value 00h (all flags cleared)
- The PWM signal starts as soon as the last instruction (`ldi MC`) is executed

Program example

```
;**************** A-R Timer Register Set ******************
RC         .def 0D9h,0FFh,0FFh     ;reload/capture register
CP         .def 0DAh,0FFh,0FFh     ;compare register
MC         .def 0D5h,0FFh,0FFh     ;mode control register
SC0        .def 0D6h,0FFh,0FFh     ;status/control register 0
SC1        .def 0D7h,0FFh,0FFh     ;status/control register 1
LR         .def 0DBh,0FFh,0FFh     ;load register
;============================================================
           ldi CP, 185             ;compare register = 185d
           ldi RC, 144             ;reload register = 144d
           ldi SC1,001h            ;clock source= CPU clock divided by 3
                                   ;prescaler ratio = 1
           ldi MC, 11100000b       ;auto-reload mode,interrupts disabled
                                   ;PWMOUT enabled, start timer
```

Example 2:

Target: generate a PWM signal of frequency 31.3 kHz with a duty cycle proportional to an input analog voltage varying between 0V and V_{CC}: 0v corresponds to 0% duty cycle, V_{CC} corresponds to 100% duty cycle. The CPU clock is 8 MHz:

Referring to example 1, we find that the prescaler ratio must be 1, the value in RC must be 0 and the value in CP is taken directly from the A/D output (varying from 0 to 255d depending upon the analog input):

RC = 0
CP = 0...255d
prescaler ratio = 1
F_{IN} = 8 MHz
F_{PWM} = 8 MHZ / 255 = 31.37 kHz
Duty cycle = CP / 255d
Resolution = 1/255d = 0.39 %

We need to implement a software loop which repetitively converts the analog input to a digital value and copies this digital value into CP:

Program example

```
A          .def 0FFh,0FFh,0FFh     ;Accumulator
;************** Port Register Set ***************************
PA         .def 0C0h,0FFh,0FFh     ;Data Register Port A
PADIR      .def 0C4h,0FFh,0FFh     ;Data Direction Reg. Port A
PAOPT      .def 0CCh,0FFh,0FFh     ;Option Register Port A
;***************** Timer Register Set  ***********************
RC         .def 0D9h,0FFh,0FFh     ;reload capture register
CP         .def 0DAh,0FFh,0FFh     ;compare Register
MC         .def 0D5h,0FFh,0FFh     ;mode control register
SC0        .def 0D6h,0FFh,0FFh     ;status control register 0
SC1        .def 0D7h,0FFh,0FFh     ;status control register 1
LR         .def 0DBh,0FFh,0FFh     ;load register
;**************** ADC Register Set ***************************
ADCC       .def 0D1h,0FFh,0FFh     ;adc control register
ADC        .def 0D0h,0FFh,0FFh     ;adc result Register
;************* MAIN ***************************************
;port A must be an IOP3 type (port with analog input mode):
           ldi PADIR,000h
           ldi PAOPT,001h          ;PA0 is the analog input
           ldi PA,   001h          ;PA1..PA7 are input with pull-up
           ldi SC1,00000000b       ;PSC divides by 1
           ldi CP,07fh             ;compare register = 127d
                                   ;(initial duty = 50%)
           ldi RC,000h             ;reload register = 0
                                   ;(count up from 0 to 255)
           ldi MC,11100000b        ;auto-reload mode,interrupt disabled
                                   ;PWMOUT enabled, start timer with
                                   ;duty cycle 50% at start-up
adc_loop
           ldi ADCC,030h           ;start A/D conversion
wt_eoc
           jrr 6,ADCC,wt_eoc       ;wait for end-of-conversion
           ld A,ADC                ;save result into Accumulator
           ld CP,A                 ;copy it into CP register
           jp adc_loop             ;do it again
```

The length of the loop “adc_loop” is the interval at which the duty cycle of the PWM signal will be updated. In this example, this loop lasts around 76µs, so the PWM duty cycle is updated every 2 or 3 PWM cycles.

Calculation of the 80µs:

- Instructions `ldi`, `ld` and `jp` last 4 cycles of 13 clock periods -> 4 x 4 x 13 8MHz periods = 26µs

- Instruction `jrr` lasts as long as the A/D takes to convert, which is around 50µs at 8MHz -> total length of adc_loop at 8 MHz = approx. 76µs

This last example shows that, as requested, the CPU is not used to generate the PWM signal, but only to start/stop it and to update the duty cycle.

THE SOFTWARE INCLUDED IN THIS NOTE IS FOR GUIDANCE ONLY. SGS-THOMSON SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM USE OF THE SOFTWARE.