NCTUser (/s/profile/0050X0000089RdIQAU) (Community Member) asked a question.
May 27, 2019 at 6:52 AM (/s/question/0D50X0000At02zOSQQ/stm32-general-purpose-timer-update-event-and-glitch-problem)

STM32 General Purpose Timer Update Event and glitch problem

I use STM32F102 and TIM2 timer in PWM1 mode to generate defined frequency and duty cycle
square wave. I have some questions and I hope can get some help from experienced colleagues.

1. I update the ARR/CCR1 register in an interrput which is defined by the communication
cycle and independent form timer frequency , which means I write the ARR/CCR1 asynchronusly
form Timer Update Event. I was afraid that the update event happens between the writing of
ACC and CCR1 and the content of two registers became inconsistent. But if I disable the
update event I got a missing pulse. How can I use the Timer to have always consistent ARR/CCR1 pairs and no missing pulses?

```
 TIM2->EGR &= ~TIM_EGR_UG;
/* ... */
TIM2->ARR = periodCh1; // If the UpdateEvent happens here the pulse is missing!
 TIM2->CCR1 = pulseCh1;
/* ... */
TIM2->EGR |= TIM_EGR_UG;
```

2. If the ARR and CCR1 registers are equal (which means for me the 100% duty cycle) I expected constant level on the timer output,
but I noticed there is a one timer clock long glitch on the output (one glitch per update event).
Is it the normal behavior? Should I manually force the output in constant state in this situation?

3. How can I force the output of the compare module in known state? The best way to change the Output compare 1 mode (OC1M)
to forced mode or should I configure the pin back to GPIO output?

Thank you.

STM32 MCUs (/s/topic/0TO0X000000BSqSWAW/) TIMER (/s/topic/0TO0X000000BYSRWA4/) STM32F1 (/s/topic/0TO0X000000BWT2WAO/)

Like Answer Share 5 answers 1.39K views

Top Rated Answers

waclawek.jan (/s/profile/0050X000007vqmpQAA) (Community Member)
Edited May 27, 2019 at 7:20 AM

1.
> TIM2->EGR &= ~TIM_EGR_UG;
This does nothing, and
> TIM2->EGR |= TIM_EGR_UG;
this forces an update event.

You can't stop the update event generation. TIM_EGR.UG *generates* an update event, even if the counter is not matching ARR. You could stop the counter itself (possibly while disabling higher priority interrupts), if a small change in period is not an issue.

2.
CCR1 = 0 means no pulse, CCR1 = 1 means pulse 1 cycle long, CCR1 = 2 means pulse 2 cycle long, ... , CCR1 = ARR means pulse ARR cycle long - but the period is ARR+1, so to achieve 100% you have to set CCR1 = ARR + 1 (or higher).

3.
Both methods work and it depends on context which one is "better". Changing pin mode to GPIO output preserves the timer's internal state - it may even continue to run and for example throw interrupts, if that's desired. On the other hand, setting output compare mode to forced may allow to have a defined width pulse if you switch it back to PWM in a random time.

JW

Selected as Best Like 1 like

All Answers

waclawek.jan (/s/profile/0050X000007vqmpQAA) (Community Member)
Edited May 27, 2019 at 7:20 AM

1.
> TIM2->EGR &= ~TIM_EGR_UG;
This does nothing, and
> TIM2->EGR |= TIM_EGR_UG;
this forces an update event.

You can't stop the update event generation. TIM_EGR.UG *generates* an update event, even if the counter is not matching ARR. You could stop the counter itself (possibly while disabling higher priority interrupts), if a small change in period is not an issue.

2.
CCR1 = 0 means no pulse, CCR1 = 1 means pulse 1 cycle long, CCR1 = 2 means pulse 2 cycle long, ... , CCR1 = ARR means pulse ARR cycle long - but the period is ARR+1, so to achieve 100% you have to set CCR1 = ARR + 1 (or higher).

3.
Both methods work and it depends on context which one is "better". Changing pin mode to GPIO output preserves the timer's internal state - it may even continue to run and for example throw interrupts, if that's desired. On the other hand, setting output compare mode to forced may allow to have a defined width pulse if you switch it back to PWM in a random time.

JW

Selected as Best Like Reply 1 like

NCTUser (/s/profile/0050X0000089RdIQAU) (Community Member)
a year ago

Thank you for your answer! Now I understand better what happened. But
what about the sequential writing of the ARR/CCR1 registers? Should I worry
about what will happen, when the update event occurs between the writing
of ARR and CCR1? Could happen the situation that the ARR register is updated
but the CCR1 not (and CCR1 holds previous value until the next update event)?

Like Reply

waclawek.jan (/s/profile/0050X000007vqmpQAA) (Community Member)
a year ago

As I've said, the easiest way is to stop the counter for the time you are updating ARR/CCR1.

JW

Like Reply

NCTUser (/s/profile/0050X0000089RdIQAU) (Community Member)
a year ago

I could bring the compare module's output to known (high) state
if immediately output switch needed:

```
  TIM2->ARR = 0;
  TIM2->CCR1 = 0xffff;
  TIM2->EGR |= TIM_EGR_UG;
```

It helps to avoid the GPIO or Output Mode reconfiguration, altough
both works well. I could prevent the ARR/CCR1 pair mismatch by
disabling the register reload:

```
  TIM2->CR1 |= TIM_CR1_UDIS;
  TIM2->ARR = periodCh1;
  TIM2->CCR1 = pulseCh1;
  TIM2->CR1 &= ~TIM_CR1_UDIS;
```

Like Reply

waclawek.jan (/s/profile/0050X000007vqmpQAA) (Community Member)
a year ago

Nice trick, using UDIS (if you don't need update eg. for periodic interrupt)!

JW

Like Reply

Login to answer this question