**CG1112 EPP 2**

**Week 6 – Tutorial 3 – Part 1**

Objectives:

* Explore the PWM block in the Atmel microcontroller.

**Q1.**

We saw that the AT328p has 3 Timers capable of generating 6 individual PWM signals. Consider the following scenarios and describe how you will be able to resolve the challenges.

1. Describe a SW approach to generate a PWM signal that is not dependent on the HW PWM peripheral block within the microcontroller. What is the drawback of generating the PWM using this approach?

Answer:

Pseudo-Code:

1. Configure Pin as output
2. Set as ‘1’
3. Delay for ‘T-on’
4. Set as ‘0’
5. Delay for ‘T-off’
6. Loop to Step 2.

With this approach, the microcontroller is continuously executing the loop to generate the required PWM. As such, it cant do anything else more productive. Furthermore, if there are interrupts in the system, the timing characteristics for the PWM will also be affected.

When the peripheral block is used to generate the PWM, only the initial setup code needs to be executed. Subsequently, it is driven by the HW with minimal processing time in the Interrupts.

1. The microcontroller that you have to use for a particular project (not the AT328p) doesn’t have any PWM module. How can you still generate a PWM signal WITHOUT relying on the software-based approach that you did Part A?

(Recall HW-based solutions from EPP1)

Answer:

One possible approach is to use the 555 Timer to generate the required PWM signal. The PWM and the Duty Cycle can be varied by adjusting the Ra and Rb value of the circuit.





\* You have learnt this in EPP1. You can refer to   
Week6 Studio1 from EPP1 to refresh your memory.

1. You feel that the approach in the earlier part is going to add additional cost to your project and you decide that stick with the AT328P. You require a very low-frequency PWM signal and even with the largest pre-scaler setting, the period is still too high. How will you be able to generate a PWM signal with the required period?

Propose at least **THREE** different approaches that can involve both HW and SW.

Answer:

Option A:

It is possible to clock the device using an external clock on T0 pin.

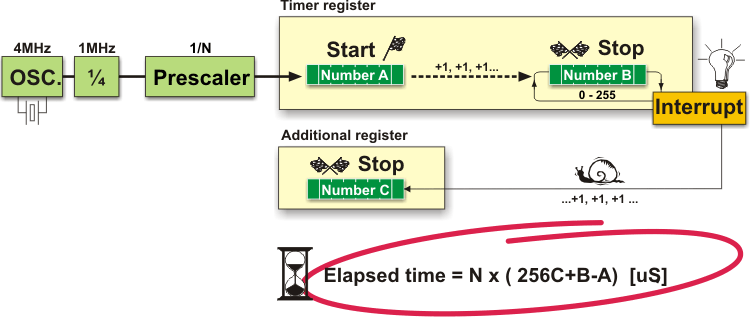




This option will require an external clock to be connected to the T0 pin.

Option B:

Another software approach would be to use the ISR to keep track of the number of Interrupts and then toggle the output pin accordingly. In this approach, we cannot allow the PWM block to control the pin directly. The ISR needs to handle it.



Option C:

Consider using a higher-bit timer. In the AT328P, we have Timer 1, which is a 16-bit timer/counter.

**Q2.**

In the studio, you were driving a single motor. If we were to drive both Motors concurrently, one possibility is to use both Timer 0 and 2. Assume that both Timer blocks are configured to use Interrupts and that the microcontroller was executing code in your loop() routine prior to the interrupts.

1. If both these interrupts were to be triggered at the same time, what would be the sequence of execution?

Answer:

Main()

Timer 2 ISR

Timer 0  
ISR

Main()

Context Switch

1. If Timer 0 triggered the interrupt just before Timer 2, what would be the sequence of execution?

Answer:

Main()

Timer 0 ISR

Timer 2  
ISR

Main()

Context Switch

By default, Nested Interrupts are disabled. When Timer 0 triggers the interrupt first, the controller will jump to Timer 0 ISR and start executing. When Timer 2 triggers, the event is captured by the flag, but the current ISR will complete execution and perform a context switch to the loop(). At this time, the flag will cause the jump to the Timer 2 ISR.

DISCUSSION POINT:

What if nested interrupts were enabled?

Main()

T0 ISR

Timer 2  
ISR

Main()

T0 ISR

Context Switch

T0 ISR pre-empted by T2

T0 ISR resumes