Timers in AT89C5131A
Lab 6

19 September 2019

1 Generating pulse-width modulated (PWM) signal using timers

1.1 Aim

1. Configure timers of AT89C5131A to generate a PWM signal.
2. Generate variable duty cycle based on user input.
3. Using PWM for simple digital-to-analog conversion.

2 Homework

You will need to read this whole document carefully to be able to perform the homework and labwork problems.

Note: Simulate your code in Keil and verify the signal frequency using the logic analyzer

1. Write assembly code to generate a 1kHz square wave on Pin P3.2 using polling.
2. Write assembly code to generate a 1kHz square wave on Pin P3.2 using timer interrupts.
3. Design a first order RC low pass filter with cut off frequency of 10Hz.

2.1 Background

Control of analog output voltage is not very easy for a micro-controller (though this can be done using a DAC, as we shall learn later). However, a micro-controller can change the duty cycle of the voltage.
If a voltage V is applied for time T_1 and 0 V is applied for time T_2, the average DC voltage is given by, $V_{avg} = \frac{T_1}{T_1+T_2}$. This average value can be controlled by varying T_1 and T_2, typically keeping $(T_1 + T_2)$ constant.

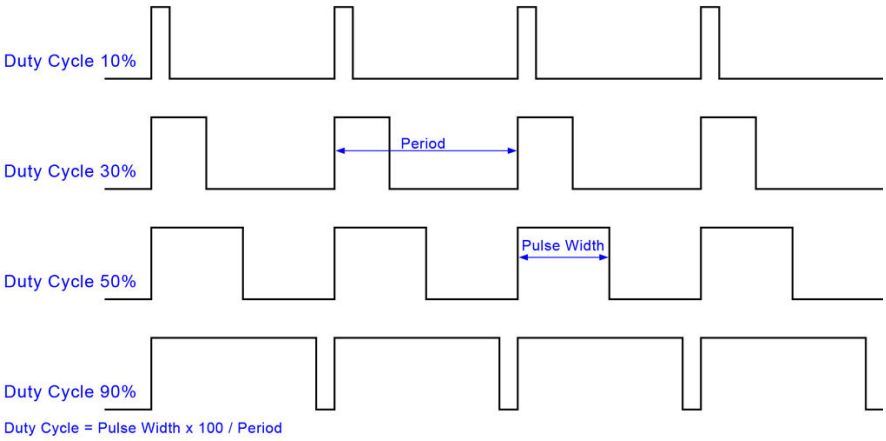


Figure 1: Pulse width modulation

2.2 Timer Basics

The 8051 has two timers T0 and T_1, which may be configured and used individually. The 8052 has an additional Timer T_2. All these counters count up on negative going edges at their inputs. These can be used as event counters (where they count the number of negative transitions on a pin connected to some external source), or as Timers, where they count up once every twelfth clock cycle.
Gating is accomplished by control bits in the special function registers, and using the external interrupt pins as gate inputs. There are no output pins associated with the timers. The processor can be interrupted by the timers and as a part of the interrupt service routine, the processor can perform any IO function through its ports. When used as baud rate generators, the output goes directly to the serial port hardware within the microcomputer.
8051 timers always count up. Each counter has a 16 bit count register in the SFR area. The low and high bytes can be accessed as separate bytes. When their count rolls over from the maximum count to 0000, they set the corresponding timer flag (TF1 or TF0) in TCON. The 8051 can be set up so that an interrupt occurs whenever TF1 or TF0 is set. When 8051 branches to the interrupt vector, it automatically clears the TF flag.
Please refer to " The 8051 Microcontroller and Embedded Systems Using Assembly and C ", Mazidi or "The 8051 Microcontroller ", Kenneth J Ayala to know more about timer interrupts.

2.2.1 Timer Functions

When used as timers, the 8051 timers count up every 12th clock cycle. This is selected by clearing the corresponding $C/\bar{T}$ flags in the TMOD special function register, placed at the address 89H.

2.2.2 External Event Counting

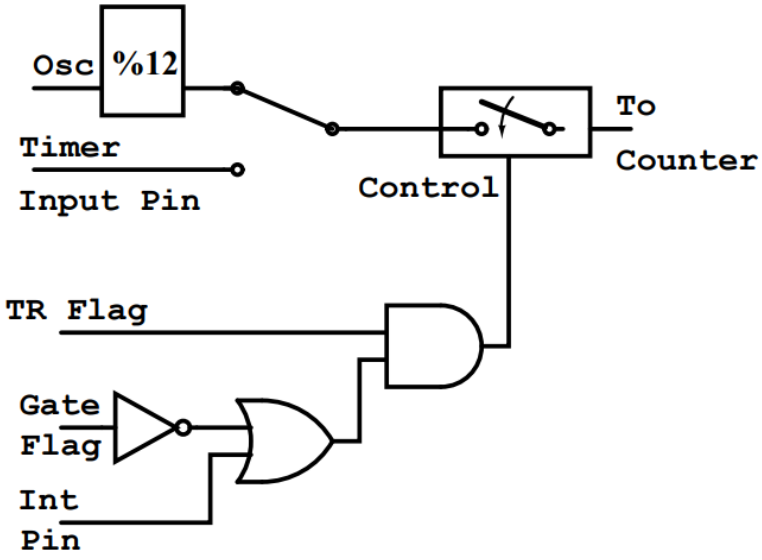
Port P3 of 8051 is a multi-function port. Different lines of this port carry out functions which are additional to data input-output on the port.
Lines P3.5 and P3.4 can be used as inputs to Timers T_1 and T0 respectively. If the $C/\bar{T}$ flag of a timer is set, the corresponding line is sampled once every machine cycle. The count is advanced when a negative step is noticed on the line: this involves sampling a high level in one cycle and a low one on the next. Since each machine cycle takes 12 clock cycles, the fastest event counting rate is clock frequency/24.

2.2.3 Special Function Registers

The functioning of these timers is controlled through several special function registers.

TCON register at BYTE address 88H								
Bit No.	7	6	5	4	3	2	1	0
Bit Name	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Bit Addr	8F	8E	8D	8C	8B	8A	89	88
TMOD register at BYTE address 89H								
Bit No.	7	6	5	4	3	2	1	0
Timer:	T1				T0			
Bit Name	G1	C/T1	T1M1	T1M0	G0	C/T0	T0M1	T0M0

The TR0 and TR1 flags in the TCON register (at address 88H) enable a timer to run, when set.
The $C/\bar{T}$ flags in the TMOD register (at address 89H) decide whether event counter operation (flag set) or timer operation (flag cleared) will be used. (TMOD is not bit addressable).
The Gate flags in TMOD decide whether counting will be gated by the corresponding external interrupt pin in P3. If the Gate Flag is cleared, the counter is enabled by the TR flag alone. If the Gate flag is set, counting also requires the corresponding external interrupt pin in P3 to be HIGH. This is useful for measuring pulse widths.



Various control registers for timers are placed in the SFR area as shown below:

Reg. Name	Address	Function
TCON	88H	Timer status
TMOD	89H	Timer modes and Config
TL0	8AH	T0 count register: Low
TL1	8BH	T1 count register: Low
TH0	8CH	T0 count register: High
TH1	8DH	T1 count register: High

2.2.4 Timer modes

1. **Mode 0** In this mode, the timers act as 13 bit counters. This mode is largely meant for providing compatibility with an older microcontroller from intel (8048). This mode is practically never used in fresh designs. Except for the counter size, this mode is identical to mode 1.

2. **Mode 1** In this mode, the timers are 16 bits in size. This is a commonly used mode. It is common to configure the timer to cause an interrupt when it overflows. The interrupt routine then reloads the timer.

3. **Mode 2** This mode provides an 8 bit counter with auto-reload. It uses the high byte of the count register to store the count value and the low byte as the actual counter. The counter is automatically re-loaded from TH when it overflows. Thus, there is no software overhead for re-loading the registers.This is convenient for generating baud rates etc.

   The timing resolution is much lower in this mode (only 8 bits). Therefore crystal frequencies have to be carefully chosen to generate accurate baud rates. Crystals of 11.059 MHz are often used rather than 12 MHz for this reason.

4. **Mode 3** In this mode Timers T0 and T_1 behave quite differently. T0 acts as two independent 8 bit counters. Count register TL0 uses the resources (such as the RUN flag, overflow flag) in TCON, TMOD etc. meant for T0. Similarly, TH0 uses the resources meant for T_1. Thus, TR1 will enable running the 8 bit counter made up of TH0. TF1 will be set whenever TH0 overflows.

   T_1 now has no control bits at all! It can only be used for services which require no control, no gating and no interrupts. Thus, T_1 can be used for Baud rate generation, while we still have two timers available (both with 8 bit resolution).

2.2.5 Enabling Timer Interrupts

SFR IE at byte address A8H contains flags which allow an interrupt to occur whenever a timer overflows.

Bit No.	7	6	5	4	3	2	1	0
Bit Name	EA	-	-	ES	ET1	EX1	ET0	EX0
Interrupt on	IE	U	U	SI	TF1	Ex1	TF0	Ex0

The most significant bit of the register is a global interrupt enable flag. This bit must be set in order to enable any interrupt. Bit 5 is used by 8052 for the third timer available in 8052. Bit 3 should be set to enable interrupts from Timer 1 overflow. Bit 1 enables interrupts from Timer 0 when it overflows.

2.2.6 Interrupt Vectors for Timers

When an interrupt occurs, the updated PC is pushed on the stack and is loaded with the vector address corresponding to the interrupt. The following table gives the vector addresses. The order of entries in the table is also the order in which the 8051 will poll these in case of multiple interrupts.

Interrupt Source	Vector address
External Interrupt 0	0003H
Timer 0 Overflow	000BH
External Interrupt 1	0013H
Timer 1 Overflow	001BH
Serial Interface	0023H

8051 starts executing from address 0000H at power-up or reset. The first 3 bytes are typically used for placing a long jump instruction to start of the code area. The interrupt vectors start from 0003 and are separated by 8 bytes from each other. Many simple interrupt handlers can be accommodated in this space. (For example, re-loading of counts after timer

overflow. Otherwise, jump.)
Thus, to enable interrupts from T0, we have to do

1. SetB EA ;(or SetB IE.7) to enable interrupts
2. SetB ET0 ;(or SetB IE.1) to enable interrupts from T0

After this, whenever T0 overflows, TF0 will be set (in SFR TCON), the currently running program will be interrupted, its PC value will be put on the stack (PC-L first, PC-H after because the stack grows upwards in 8051), and PC will be loaded with 000B H. The interrupt handler for T0 should be placed here, and it should end with the instruction: RETI .

2.2.7 Initializing Timer Counts

Timers count up and set their flags when they go from max. count to 0000. Therefore, it is more convenient to think of the initial count as negative numbers.
Suppose we want the timer to time out after 10 mS and we are using a 12 MHz crystal. Since the timer is incremented at $f_{osc}/12$ = 1MHz, each count cycle is 1 μs in duration. So, we need a count time of 10,000 cycles. We should therefore initialise the count to -10,000.
Since 10,000 = 39*256+16 = 2710H,
-10000 = 2s complement of 2710 = D8F0.
Another way to think about it is that the overflow count is actually 10000H. So we should subtract the required count from this and initialize the count to the difference. Thus, the initial count in the above example should be:

```
 10000
- 2710
------
= D8F0
------
```

Either way, we get the same result.

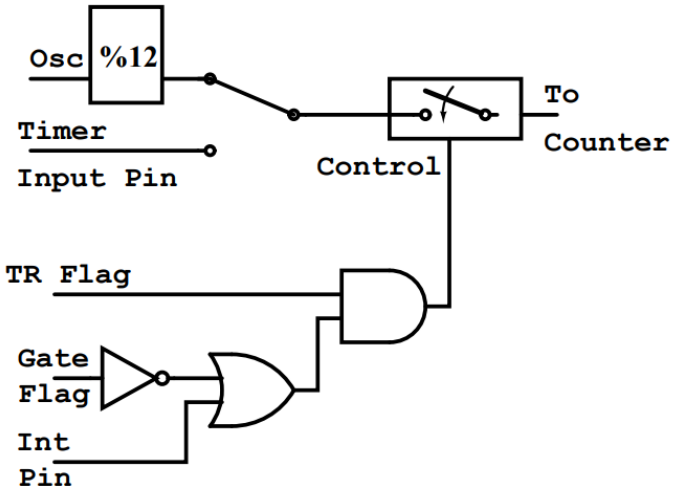
2.2.8 Reading Timer Counts

Since the current count is in two bytes, we have to read these sequentially. This presents a problem - because by the time we go to read the second byte, the first one may have changed. For example, suppose the count is 04FF. We do :
MOV R0, TL0
and get FF in R0. However, one machine cycle has passed, so the count reaches 0500. Therefore, if we do :
MOV R1, TH0
we shall get 05 in R1. Thus we can be misled into thinking that the count is actually 05FF. This problem had been tackled in 8253 and 8254 by the Latch Counter commands. This command would copy the count to a latch while the counter could continue operating. The latch could then be read safely. This command does not exist in an 8051. So how do we solve this problem?
One way is to read the high byte of the timer, then read the low byte, then read the high byte again. If the high byte read the second time is not the same as the high byte read the first time, one must repeat the cycle. In code, this would appear as:
READT0:
MOV A,TH0
MOV R0,TL0
CJNE A,TH0,READT0

Another way is to freeze the counter by clearing its RUN bit, read its value and then start it again by setting the RUN bit. Of course, you might miss a few counts this way - but at least the timer value wont be grossly wrong.

2.2.9 Use of gating

Gating is useful when we want an external source to start or stop the timer. If the gate bit in TMOD is set, the timer would be enabled only if the corresponding external interrupt pin is high. This can be used to measure pulse widths, by appling the pulse to the external interrupt pin. This can be understood by the circuit given next:



3 Comparing interrupt and polling methods

In this experiement, the timer flag can be monitored continuously (polled) and appropriate action can be taken when the flag is set. But this is not an efficient way to use of the microcontroller.

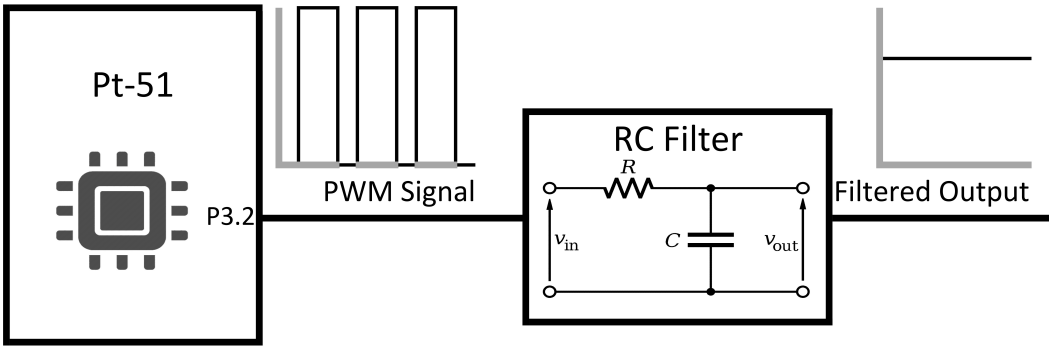
The microcontroller can serve many devices. Each device can get the attention of the microcontroller based on the priority assigned to it. Polling method can not assign priority because it checks all devices in a round-robin fashion.

In interrupt method, the microcontroller can also ignore a device's request for service, which is not possible in polling. Most importantly, polling method wastes much of the microcontroller's time by polling devices which do not need servicing. For all these reasons, interrupt method is preferred over polling method.

4 Building a filter

Once the PWM wave is generated we will pass it through a first order RC LPF of cutoff 10Hz to get the averaged DC output. Constructing a basic passive low pass filter is quite simple. A first order filter uses one capacitor and one resistor. Building a first order filter is as simple as choosing a capacitor value and then computing the resistor value using the following equation.

$$f_{3dB} = \frac{1}{2\pi R_f C_f}$$



5 Labwork

1. Read three on-board switches (P0.0-2) to accept the duty cycle from the user and write a assembly program to generate a PWM with the duty cycles corresponding to the switch positions as indicated in the following table. Note that the frequency of the PWM ($\frac{1}{T_1+T_2}$) is 1 kHz.

P1.0-2	Duty Cycle(%)
000	20%
001	30%
010	40%
011	50%
100	60%
101	70%
110	80%
111	90%

Table 1: Mapping of switch positions to duty cycle.

2. Observe the output waveform on the DSO and verify the duty cycle.

3. Connect the PWM generated to the RC low pass filter designed by you to get the average value of the generated PWM waveform. Now vary the input and observe the change in output voltage.