EE447 Pre-Liminary Lab #4

Question 1:

a) In which mode is TIMER0A used?

Ans. TIMER0A is used in periodic countdown mode.

In this part we were tasked to create a pulse train of 20 KHz using the *Pulse.h* file which create a 1 MHz clock frequency, in that code the values of HIGH and LOW were defined which were to be altered theoretically using the following calculations to obtain a 40% duty cycle.

$$20\ KHz \rightarrow 1\ \mu s\ we\ need\ a\ multiple\ of\ 50$$

$$For\ 40\%\ high\ 50 \times 40\% = 20\ (high\ new\ value)\ , For\ 60\%\ low\ 50 \times 60\% = 30\ (low\ new\ value)$$

By writing the code for TIMER0A Handler interrupt function as in our code blocks, we can observe the following waveform for the above mentioned high and low values:

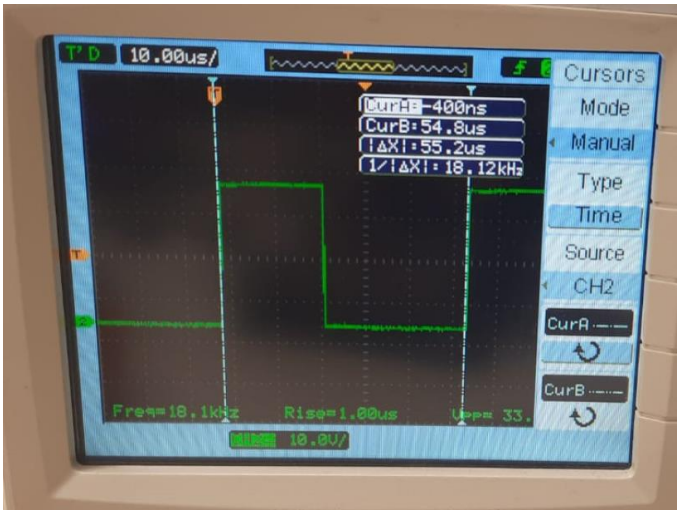

Fig 1.1 Theoretically calculated pulse train

However as you can see that the actual frequency measured on the oscilloscope is around 18.1 KHz and the period of the waveform is 55μs instead of the 50μs, this may occur due to the code statements being executed in the TIMER0A Handler interrupt function which may cause a 5.0μs additional time, therefore to get an approximate 20KHz frequency pulse train we had to modify the theoretical values to the value shown in the code and as observed in the figure below:

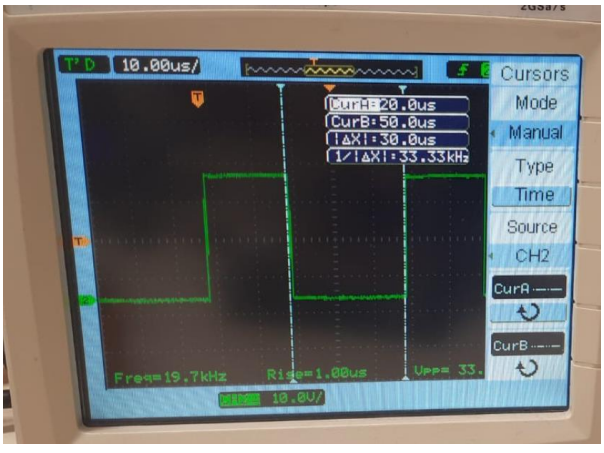

Fig 1.2 LOW On time for pulse train (Adjusted High and Low)

Fig 1.3 HIGH On time for pulse train (Adjusted High and Low)

The low on time can be seen on the right side of the which is roughly 30μs whereas on the left side we can observe that the high on time is 20μs which gives us the perfect 40% duty cycle with a clock frequency of approximately 20 KHz.

Question 2)

In this question we were tasked to find the pulse width, period and duty cycle for the signal that was generated in part 1, using the formulas:

$$pulse\ width = t_{rise_i} - t_{fall_i}\,, \qquad period = t_{rise_{i+1}} - t_{rise_i}, \qquad duty\ cycle = \frac{pulse\ width * 100}{period}$$

The figure below shows the period and the pulse width in which the results are displayed in the Termite.

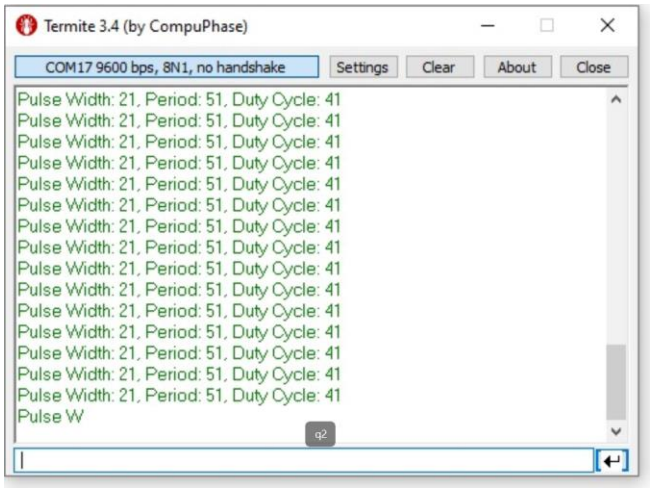

Fig 1.4 Termite display for signal in part a

We used another timer called TIMER3 to configure the rising and falling edges so that the timer modules don't overlap.

Code Explanation:

The main() function serves as the central execution point. It declares an extern function OutStr(char*), several variables are declared within the main() function:

- time_val[3] is an array designed to hold time values, initializing its third element to 123.
- count and countt are counters used within the program's control flow.
- tempchar[100] acts as a buffer to temporarily store formatted strings.
- pulse_width, period, and duty_cycle are variables designated to hold calculated values for pulse width, period, and duty cycle.

The code initiates the pulse-related configurations by calling pulse_init(), presumably setting up the microcontroller for tasks involving pulse handling. Following this, the program enters an infinite loop using while(1), indicating its continuous operation.

Within this loop, a conditional check (if (count > 2)) is made:

- It calculates pulse_width and period based on time differences stored in the time_val array.
- Duty_cycle is then computed as the ratio of pulse_width to period multiplied by 100.

The values are formatted into a string using sprintf() and stored in tempchar. The OutStr() function is called, presumably to output the contents of tempchar. Another conditional check (if (TIMER3->RIS & (1<<2))) appears to handle interrupts:

It captures the current timer value and stores it in time_val[count]. The interrupt flag is cleared (TIMER3->ICR = 5;). The count variable is incremented.

Question 3)

The ultrasonic sensor is used in this task to find the pulse width of the incoming signal and the result along with the distance is calculated and shown in the termite window:

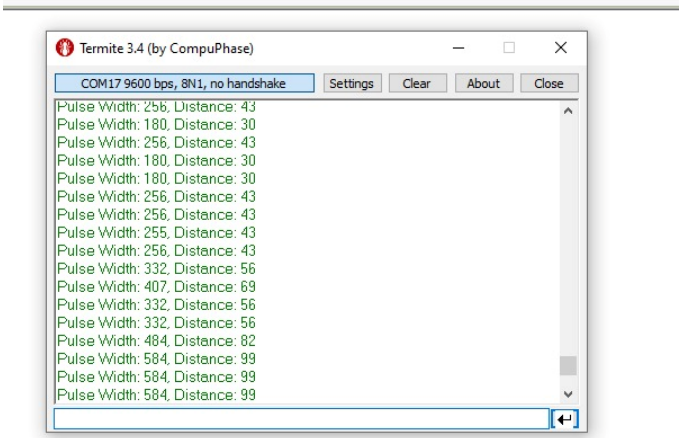

Fig 1.5 Termite display for the Ultrasonic Sensor

Code Explanation:

Global variables are declared to store essential values throughout the program. These include an array time_val[2] to hold time measurements, two counter variables count and countt, a character array tempchar[100] for temporarily storing formatted strings, and two integers pulse_width and distance used for pulse and distance calculations. The delay() function within the code seems to serve the purpose of introducing a delay in program execution. It utilizes a simple for-loop to create a delay in the program flow.

Moving on to the main() function, it initiates the pulse-related functionalities through pulse_init() and enters an infinite loop denoted by while(1). This loop suggests that the program continually executes the contained instructions without terminating. Within this loop, there's a segment responsible for generating pulses, measuring their widths, and subsequently calculating distances. This section involves toggling a specific pin on and off to generate a pulse, waiting for certain events related to Timer 3, measuring time values, and computing pulse width and distance based on these measurements.

After obtaining the pulse width and distance, the code formats this data into a string using sprintf() and stores it in the tempchar array for further processing or output. Finally, the formatted string tempchar is likely sent or processed through an external function OutStr(). This function is declared as an external function, meaning its definition lies outside the scope of this code snippet. This function presumably handles output operations, potentially displaying or transmitting the formatted data for monitoring, logging, or control purposes.