UCD School of Electrical and Electronic Engineering

EEEN40280   
Digital and Embedded Systems

Microcontroller & Assembly Language Report

**Name:** Aidan O’Sullivan **Student Number:** 20209138

**Working with:** Joshua King

**Code submitted by:**

I certify that ALL of the following are true:

1. I have read the *UCD Plagiarism Policy* and the *College of Engineering and Architecture Plagiarism Protocol*. (These documents are available on Brightspace.)
2. I understand fully the definition of plagiarism and the consequences of plagiarism as discussed in the documents above.
3. I recognise that any work that has been plagiarised (in whole or in part) may be subject to penalties, as outlined in the documents above.
4. I have not already submitted this work, or any version of it, for assessment in any other module in this University, or any other institution.
5. The work described in this report is all the original work of my team, and this report is all my own work, except where otherwise acknowledged in the report.

**Signed:** A’OS **Date:** 7 February 2021

### Introduction

In this assignment, we were asked to generate a square wave output signal on a pin of the ADuC841 microcontroller, first using software delay, then using a hardware timer and interrupts.

### Software Delay

The algorithm is based on the blinking LED program provided as an example. The program changes the state of the output pin, waits for a suitable delay time, and repeats.

The frequency required is fixed at 3600 Hz, and we calculated that to be 3072 clock cycles at our clock frequency of 11.0592 MHz. So we wanted to change the state of the output pin every 1536 clock cycles. To achieve that in a software delay, we needed six 8-bit counters.

Time of Delay =

Total Cycles = (Clock Frequency)x(Time of Delay)=(11.0592 = 3072

Assuming a 50% duty cycle, the pin should be ON for half of the cycles and OFF for the other half of the cycles.

OFF/ON cycles =

#### Program Design

The main program has only 4 instructions, following the algorithm above exactly. It calls a delay subroutine to get the required time delay and changes the state of P3.6.

In the delay subroutine, we used register 5 as the inner counter, starting at the accumulator value of value of 252. This was decremented in a loop using just the DJNZ instruction, which uses 3 clock cycles. The outer loop used register 7, initialised at 2 and decremented with the DJNZ instruction.

The total instructions were calculated as follows (Table by Joshua):

|  |  |  |  |
| --- | --- | --- | --- |
| Instruction | Cycles/Instruction | # of occurrences | Total cycles |
| MOV A, #data | 2 | 1 | 2 |
| CPL | 2 | 1 | 2 |
| CALL | 3 | 1 | 3 |
| JMP | 3 | 1 | 3 |
| MOV Rn, #data | 2 | 1 | 2 |
| MOV Rn, A | 1 | 2 | 2 |
| MOV Rn, rel | 3 | 2(A+1) | 6A+6 |
| RET | 4 | 1 | 4 |
| NOP | 1 | 0 | 0 |
| Entire Routine |  |  | 24+6A=  24+6(252)=  1536 cycles |

#### Result

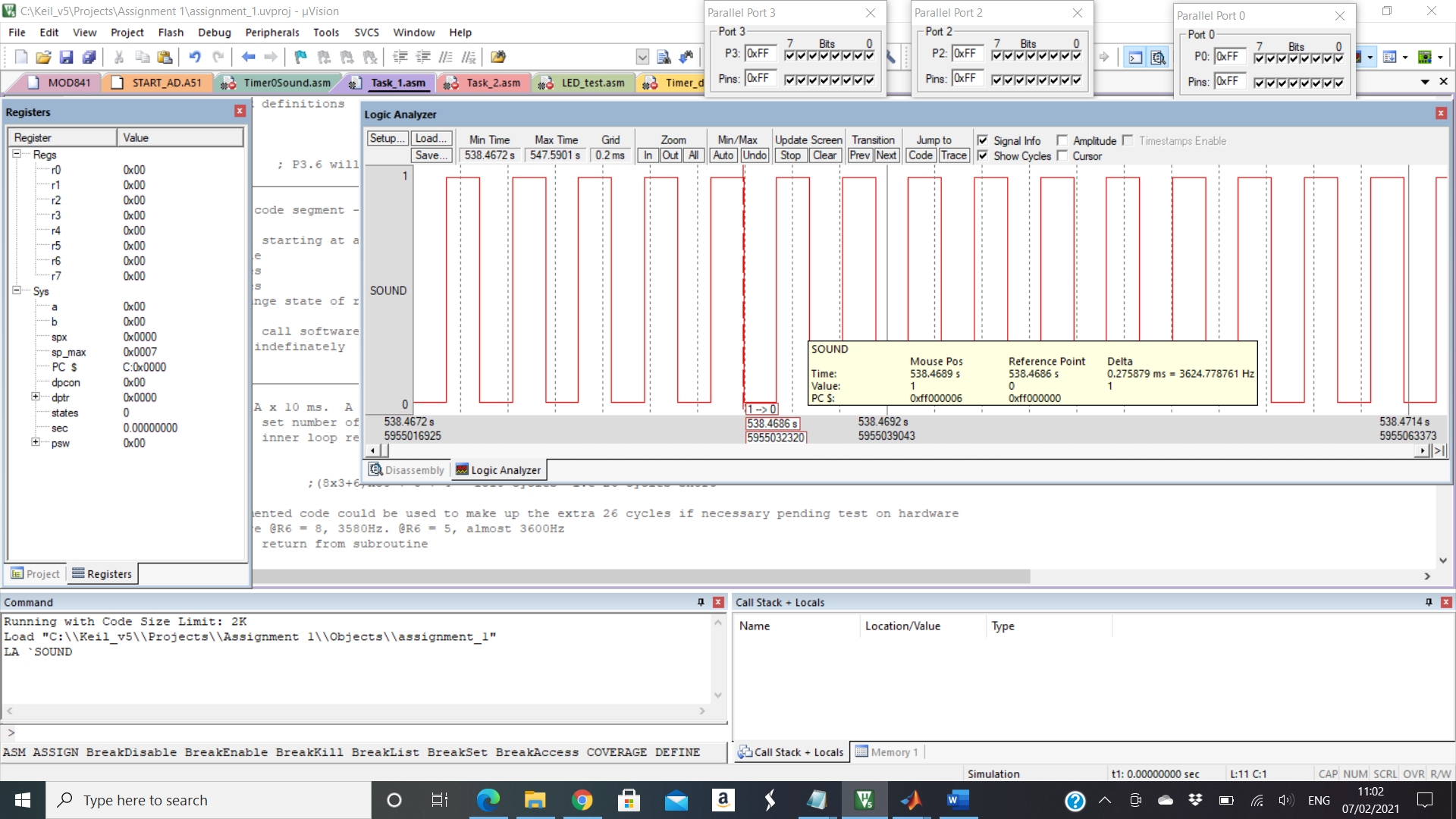


Figure 1 shows the output signal at P3.6, with a frequency of 3624 kHz

When ran on the hardware a frequency of 3600.27 Hz was measured. This is an error of 0.0075% from what we expected, and 0.0075% from the target value of 3600 Hz. We deemed this to be acceptable as the error was exceedingly small.

### Example Program

Timer 0 was set as a 13-bti timer in mode 0, we calculated the output frequency as:

Time of Delay =

Frequency =

When we tried to measure this frequency, we found that it was reasonably close to the expected value but there was a noticeable error of 8 Hz (1.18%). As it is quite a low frequency this could be interpreted as significant in some applications.

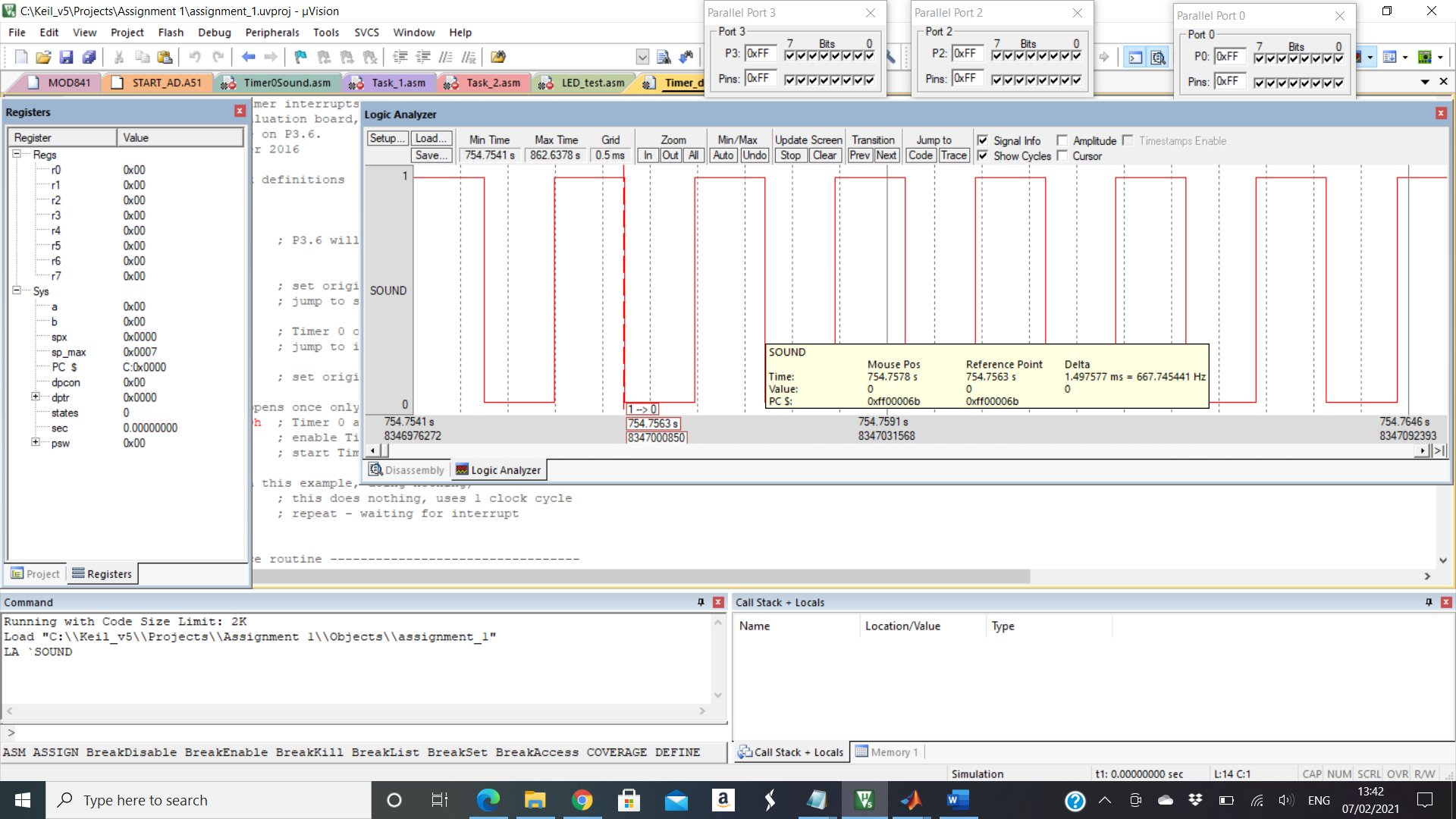


Figure 2 shows the simulated signal on P3.6 in the logic analyser

### Hardware Timing

We chose to use an algorithm based on suggestion 1 in the instructions, as we thought that it sounded more intuitive and straightforward. Using a purely hardware delay and not relying on software counts like the other options seemed more reliable, faster and gave less scope for error or miscalculation. We used Timer 2 to generate the interrupts, because we needed a 16 bit auto reload, which was supported by this timer. 8-bits would not offer enough counters to produce the required frequencies in a single overflow per ON/OFF state.

We chose a set of frequencies from 1-8 kHz in steps 0f 1 kHz. The values are shown in Table 1 below.

#### Program Design

The MAIN function only runs once, and it configures P2 as an input to allow the switch values to be read. Timer 2 is configured as a high priority 16 bit auto-reload timer, initiated and its interrupt enabled. External interrupt 0 is then enabled, with an interrupt occurring on a on a 1- to-0 transition. The carry bit C is also set, which allows the LED to be put in flashing mode in the repeating loop. External interrupt 0 is set as low priority so it will not interfere with the high priority signal generation of timer 2.

The LOOP function will repeat indefinitely. The JNC command checks the state of C. If it is set the state of the LED on P3.4 toggles between ON and OFF, with a software delay of around 0.5 seconds, achieved by calling the DELAY function. This allows for the flashing to be visible to the eye. If C was cleared in the INT0 ISR, the LED will remain off and inside LOOP it will no longer toggle the LED and instead skip from the JNC instruction to LOW\_func. In the functions, LOW\_func, HGH\_func and LED\_func the value of the switches at P2 are read and stored in the accumulator. This byte is ANDed with 07h so switches 4-8 are discounted and the accumulator will only equal between 0-7. This value is then used to reference the relevant low reload value, high reload value and LED value, that will be displayed in accordance with the frequency that the user has selected with switches 1-3. There are 3 loop-up tables and the value in the accumulator is used as an offset to reference the address of the relevant data to the chosen switch value.

In the timer 2 interrupt service routine the state of output pin P3.6 is toggled between ON and OFF to create a square wave, that causes the transducer to make a sound at the prescribed frequency. The timer 2 interrupt flag is also cleared by the software, as this is not done automatically by the hardware like in timer 1 and 0. The RETI instruction then returns the program to the LOOP.

In the External Interrupt 0 service routine the LED is turned off (state = 1) and the state of C is set to turn OFF the LED, when INT0 is pressed. If C had previously been set it will now be cleared and the LED will not be toggled when the program returns to the LOOP anymore due to the JNC instruction. If C was previously cleared and the LED was also off, pressing the button again will cause C to be set and when the program returns to the LOOP it will begin toggling the LED again.

Reload Register calculated as follow:

RCAP2L are the lower 8-bits of the reload value and RCAP2H are the upper 8-bits of the reload value.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Switch Value** | **Target Frequency (Hz)** | **Period (clock cycles)** | **Reload Value** | **Expected Frequency (Hz)** | **Measured Frequency (Hz)** |
| 0 | 1000 | 000 | TL2=66h  TH2=0EAh | 1000 | 1000 |
| 1 | 200 | 001 | TL2=33h  TH2=0F5h | 2000 | 2000 |
| 2 | 3000 | 010 | TL2=0CDh  TH2=0F8h | 3000 | 3000 |
| 3 | 4000 | 011 | TL2=9Ah  TH2=0Fah | 4000 | 4001.5 |
| 4 | 5000 | 100 | TL2=0AEh  TH2=0FBh | 5000 | 5000 |
| 5 | 6000 | 110 | TL2=66h  TH2=0FCh | 6000 | 5997.9 |
| 6 | 7000 | 110 | TL2=0EAh  TH2=0FCh | 7000 | 7000 |
| 7 | 8000 | 111 | TL2=4Dh  TH2=0FDh | 8000 | 8003 |

**Table 1** – Frequency calculations

#### Results

When implemented on the hardware the frequencies measured from the transducer were remarkably close to what was expected. The low frequency values were more accurate as the signals were ON and OFF for longer, so a small deviation from the expected time would give a smaller % error. The results backed up this assumption, as switches 0-2 gave the exact frequency values with no measured error. The larger frequencies were ON and OFF for far shorter times, so any small delay would have a far larger % error. This is shown in the slight deviation at high frequencies, with the max error at 8 kHz of only 3 kHz, amounting to only a 0.375% error. This is within an acceptable range for many applications.

One LED at a time lit at a time, each corresponding to their respective switch values from 0-7.

The LED on P3.4 flashed as instructed with a time delay of around 0.5 seconds between on and off. The external interrupt service routine functioned as a means for the user to turn off and back on the flashing LED by pressing INT0.

### Conclusion

Satisfactory results were achieved through the joint contribution of both members. The work was broken down in sections, which allowed for individual focus on specific tasks, while problems encountered were worked upon together in most cases to achieve solutions. The frequency values were exactly as expected in most cases or the error was low that it was within an acceptable range. While the initial program we wrote was effective but far longer, with an individual function for each switch value, look-up tables were identified as a means of streamlining the code. The size and efficiency of the software was far reduced. The Improvement section was also implemented quite effectively, reflecting the team’s incentive to make the program as advanced as possible and further their knowledge.