CPE - 3150
PROJECT 2
DATE: 04/25/2024

Microcontrollers Project 2

CPE - 3150 - Spring 2024

Project Title: **AVR ASM PROGRAMMING APPLICATION**

Team Members: **Trenton Cathcart (tcgyq@mst.edu, EE)**

Jamie Madison (jrmn64@mst.edu, EE)

Instructor: **Dua, Rohit (rdua@mst.edu, MS&T EE)**

Presented to the
Electrical & Computer Engineering Faculty of
Missouri University of Science and Technology
In partial fulfillment of the requirements for
Microcontrollers (CPE 3150)
March 2024 (SPRING 2024)

Table of Contents

1. INTRODUCTION

1.1. EXECUTIVE SUMMARY

In this project, we were assigned the task of developing a Pulse Width Modulation (PWM) controller using the ATmega324B microcontroller, which is integrated into a pre-designed interfacing board. The primary objective of our project was to design and implement a system capable of generating PWM signals with variable duty cycles and frequencies, controlled through user inputs via switches on the board.

Our project revolved around the essential concept of PWM, which is a technique used in various applications to control the amount of power delivered to electronic devices. By adjusting the duty cycle of the PWM signal— the proportion of time the signal is high versus the total period of the waveform—we could effectively manage the power levels of connected devices. The ATmega324B microcontroller, known for its robust processing capabilities and flexibility in handling multiple I/O operations, was chosen to drive our application due to its suitability in embedded system applications involving precise control tasks.

We were tasked to enable the PWM generation at specific base frequencies assigned to our group. The system needed to allow the user to increase or decrease the base frequency and duty cycle through a set of switches. Each frequency adjustment was to alter the base by 5% per press, up to a maximum change of 25%, while ensuring that if the adjustments exceeded the allowed range, a buzzer would sound a warning tone.

Similarly, duty cycle adjustments were to be implemented with the same parameters. Moreover, the project required the inclusion of a reset functionality that would return the system to its initial frequency and duty cycle settings. Finally, we were tasked to make a creative portion of the project.

2. HARDWARE

In our project, the hardware setup was a critical component that integrated the ATmega324B microcontroller with a school-designed interfacing board. This board was specially tailored to facilitate a variety of input and output operations, which were essential for the realization of our project objectives.

The interfacing board was equipped with a diverse array of peripherals mapped directly to the ATmega324B processor. It featured 11 LEDs of various colors, providing visual feedback and status indication in response to system operations and user inputs. These LEDs were connected to specific port pins on the microcontroller, allowing us to control them directly through software to indicate different frequency and duty cycle settings, as well as error states.

Accompanying the LEDs were 9 pushbuttons, each mapped to different pins of the microcontroller. These buttons allowed users to interact with the system, specifically to adjust the PWM frequency and duty cycle, and to reset the system to its default settings. The tactile feedback from these buttons was essential for real-time user interaction, making the system more intuitive and responsive.

Additionally, the board contained a speaker or buzzer, which was used as an auditory indicator. The buzzer was crucial for alerting the user when the frequency or duty cycle settings exceeded their allowed ranges, providing immediate feedback to prevent erroneous operations.

The board also boasted numerous pin outputs, offering extensive connectivity options for further expansion and experimentation. These pins could be used to connect additional components like an oscilloscope to view the signal outputs created from the asm code.

Overall, the interfacing board served as a practical and effective platform for demonstrating the capabilities of the ATmega324B microcontroller in controlling PWM signals. It provided a robust framework for us to develop and showcase our project, combining a rich set of features with the flexibility needed for complex embedded system tasks.

3. SOFTWARE IMPLEMENTATION:

3.1. PRELIMINARY

In the preliminary phase of our software implementation, we were required to establish a foundational understanding of the MPLAB integrated development environment (IDE). This involved downloading and setting up MPLAB, which is crucial for developing applications for microcontrollers like the ATmega324B. Our initial experiments focused on writing and testing simple code snippets that would manipulate the port pins of the microcontroller to drive outputs to the LEDs and produce observable waveforms on an oscilloscope. These exercises were instrumental in familiarizing ourselves with the basic functionality of MPLAB, as well as with the input/output operations of the ATmega324B processor. This hands-on approach enabled us to gain practical insights into how different coding techniques affect the hardware behavior, setting a solid groundwork for the more complex PWM control tasks that would follow in the project.

3.2. SOFTWARE DESIGN PHASE

3.2.1. MAIN LOOP AND I/O DECLARATIONS

In the design phase of our software implementation, we set the stage for our application's control flow and I/O interactions. Our main loop served as the central hub from which all peripheral operations were managed. It was here that we orchestrated the sequence of events for PWM control, ensuring a cohesive operation between input readings and output actions.

We began by initializing the I/O ports to configure the directionality of the pins connected to the LEDs and buttons. This was achieved through loading specific constants into the register R16 and writing them to the Data Direction Registers (DDRs) of ports A, D, and E. For example, setting all bits of DDRD to '1' using OUT DDRD, R16 declared all pins of Port D as outputs, readying them for sending signals to LEDs.

Following the DDR configurations, we set initial states for the ports. With the OUT PORTA, R16 instruction, we were able to drive the pins of Port A high, effectively setting a default state from which we could begin our control operations.

The main loop continuously checked for button presses using the sbis instruction, which skipped the following instruction if the designated button was not pressed. This method, combined with the subsequent RCALL to the debounce subroutine, provided a reliable means to read user inputs without false triggering due to switch bounce.

When a button press was detected, the corresponding subroutine was called to adjust the frequency or duty cycle of the PWM output. This dynamic interaction allowed real-time control of the system, as depicted in **Figure 1**, which shows the final code implementation for this section.

The output to the LEDs was handled by first disabling all LEDs on Port E to prevent any unintended illumination. This selective enabling ensured that our visual indicators were clear and accurate reflections of the current system state. The LED connected to Port D was then activated in an active-low configuration, symbolizing the PWM output or signaling a change in the system's state.

Each cycle of the loop included a call to a delay subroutine, referenced as RCALL ON, which timed the LED's on and off durations according to the current PWM settings. This subroutine was a critical component in creating the visible blinking pattern that correlated with our PWM's base frequency.

```
        LDI R16, 0xFF        ; Set R16 = 0xFF
        OUT DDRD, R16        ; Set all pins of Port D as outputs (DDRD controls direction of Port D)
        OUT PORTA, R16       ; Set all pins of Port A to high (if configured as outputs)
        OUT DDRE, R16        ; Set all pins of Port E as outputs

        ; Load the constant 0x01 into R16. This is used to set a specific pin as output.
        LDI R16, 0x01        ; Set R16 = 0x01
        OUT DDRA, R16        ; Configure Port A, Pin 0 as output, others as inputs

        ; Set various combinations of bits for PORTA, possibly for a sequential operation
        ; or device initialization. This shows configuring pins in steps for specific requirements.
        LDI R16, (1<<0)|(1<<1)|(1<<2)|(1<<3)|(1<<4)|(1<<5)|(1<<6)  ; Set R16 = 0x7F, preparing pins for a specific state
        OUT PORTA, R16       ; Apply configuration to PORTA
        LDI R16, (1<<0)|(1<<1)|(1<<2)|(1<<3)|(1<<4)|(1<<5)         ; Clear bit 6 in PORTA
        OUT PORTA, R16       ; Apply configuration to PORTA
        LDI R16, (1<<0)|(1<<1)|(1<<2)|(1<<3)|(1<<4)                ; Clear bit 5 in PORTA
        OUT PORTA, R16       ; Apply configuration to PORTA
        LDI R16, (1<<0)|(1<<1)|(1<<2)|(1<<3)                       ; Clear bit 4 in PORTA
        OUT PORTA, R16       ; Apply configuration to PORTA
        LDI R16, (1<<0)|(1<<1)|(1<<2)                              ; Clear bit 3 in PORTA
        OUT PORTA, R16       ; Apply configuration to PORTA

    loop:
        ; Check if button connected to PINA.7 is pressed to increase frequency
        sbis PINA, 7      ; Skip the next instruction if bit 7 of PINA is set (button is pressed)
        RCALL DEB         ; Call the debounce subroutine for button press, to increase frequency

        ; Check if button connected to PINA.6 is pressed to decrease frequency
        sbis PINA, 6      ; Skip the next instruction if bit 6 of PINA is set (button is pressed)
        RCALL DEBS        ; Call the debounce subroutine for button press, to decrease frequency

        SBIS PINA, 2
        RCALL CREATIVE

        ; Check if button connected to PINA.4 (not PINA.5 as comment suggests) is pressed to increase duty cycle
        sbis PINA, 4      ; Skip the next instruction if bit 4 of PINA is set (button is pressed)
        RCALL DEB5p       ; Call the debounce subroutine for button press, to increase duty cycle

        ; Check if button connected to PINA.3 is pressed to decrease duty cycle
        sbis PINA, 3      ; Skip the next instruction if bit 3 of PINA is set (button is pressed)
        RCALL DEB5m       ; Call the debounce subroutine for button press, to decrease duty cycle

        ; Disable (turn off) all LEDs on Port E to ensure only the target LED on Port D is manipulated
        LDI R16, 0xFF     ; Load R16 with 0xFF
        OUT PORTE, R16    ; Set all pins on PORTE to high, turning off LEDs assuming active-low configuration

        ; Turn on the target LED on Port D by setting all but the last bit, assuming an active-low configuration
        LDI R16, 0x7F     ; Load R16 with 0x7F, setting all bits high except for the last one
        OUT PORTD, R16    ; Output to PORTD, effectively turning on the LED connected to the last pin if active low

        ; Keep the LED on for a period based on the base frequency
        RCALL ON          ; Call a subroutine that likely implements a delay based on the current frequency

        ; Turn off the target LED on Port D by setting all bits high, assuming an active-low configuration
        LDI R16, 0xFF     ; Load R16 with 0xFF, setting all bits high
        OUT PORTD, R16    ; Output to PORTD, turning off the LED assuming active-low configuration

        ; Keep the LED off for a period based on the base frequency
        RCALL ON          ; Call the same subroutine for delay, maintaining the off period of the blinking cycle

        ; Loop indefinitely to continuously check for button presses and adjust LED behavior accordingly
        RJMP loop         ; Jump back to the start of the loop
```

FIGURE 1: MAIN LOOP AND I/O DECLARATIONS

3.2.2. DEBOUNCING

In our design, addressing the physical behavior of buttons was critical to ensure reliable operation. When a button is pressed, it doesn't simply close the circuit; it bounces between open and closed states in a very short time frame, which could lead to multiple high and low

voltage signals being registered by the microcontroller. This phenomenon, known as "switch bounce," could result in a single button press being read as multiple inputs, leading to erratic behavior.

To mitigate this, we implemented a debouncing algorithm in our code. The core concept behind our debouncing routine was to introduce a delay after the initial detection of a button press, and then recheck the state of the button after this period. If the button remained in the pressed state, we could confidently infer that an actual press occurred rather than a transient bounce.

Our code used nested loops to create this delay. By loading a counter value into register R27 and decrementing it within an outer loop, and then further decrementing another counter R28 within an inner loop, we achieved a significant delay. The NOP (No Operation) instructions provided additional, albeit minimal, delays within these loops.

After the delay, we used the SBIS and SBIC instructions to check the button's state. If the button was still pressed, indicated by a cleared bit in the PINA register, we proceeded with the corresponding action, such as increasing or decreasing the frequency or duty cycle. This routine ensured that only a stable, sustained button press would lead to a change in the system's operation.

The complete implementation of our debouncing strategy, including the counter initialization, nested delay loops, and post-delay state checking, can be observed in **Figure 2**. This debouncing routine was a crucial part of our system's software, as it allowed us to accurately interpret user inputs and maintain robust control over the PWM output.

```
DEB:          ; Start of the debounce routine for button 7
    LDI R27,14     ; Load immediate value 14 into R27. This and the next steps create a delay
                   ; to allow the button's signal to stabilize.

DELAY:        ; Label for outer delay loop
    LDI R28,40     ; Load immediate value 40 into R28. This sets up the inner delay loop count.

DEBOUNCE:     ; Label for inner debounce delay loop
    NOP            ; No Operation - does nothing, effectively a small delay
    NOP            ; Another NOP for additional delay
    DEC R28        ; Decrement R28 by 1
    BRNE DEBOUNCE  ; Branch if Not Equal to zero, continue the inner loop if R28 is not 0
                   ; This inner loop runs 40 times for each iteration of the outer loop

    DEC R27        ; Decrement R27 by 1, counting down the outer loop
    BRNE DELAY     ; Branch if Not Equal to zero, if R27 is not 0, repeat the outer delay loop
                   ; This creates a nested loop for debouncing, ensuring enough time has passed

    ; Check if the button is still pressed after debouncing
UP: SBIS PINA,7    ; Skip the next instruction if bit 7 of PINA is set (button pressed)
    RJMP UP        ; If the button is not pressed (bounced up), jump back to UP and keep checking

    ; If the program reaches this point, the button has been pressed without bouncing
    sbic PINA, 7       ; Skip if Bit in I/O Register is Cleared - If bit 7 of PINA is not set,
                       ; this instruction is skipped. Seems redundant given the context and
                       ; could be a mistake or intended for a different logic structure.
    rjmp switch7_pressed1  ; Relative jump to `switch7_pressed1` label
                           ; It's intended to increase the base frequency by 5% upon a stable press.

    RET            ; Return from subroutine. Ends the debounce routine and returns to where
                   ; the routine was called from if the button press is not stable.

DEBS:         ; Start of the debounce routine for button 6
    LDI R27,14     ; Load immediate value 14 into R27. This prepares for the outer delay loop
                   ; as part of the debouncing process.

DELAYS:       ; Label for the outer delay loop
    LDI R28,40     ; Load immediate value 40 into R28. This sets up the inner delay loop count.

DEBOUNCES:    ; Label for the inner debounce delay loop
    NOP            ; No Operation - effectively a very short delay
    NOP            ; Another NOP for additional delay
    DEC R28        ; Decrement R28 by 1
    BRNE DEBOUNCES ; Branch if Not Equal to zero, continue inner loop if R28 is not 0
                   ; This inner loop iterates 40 times for each cycle of the outer loop

    DEC R27        ; Decrement R27 by 1 for the outer loop count
    BRNE DELAYS    ; Branch if Not Equal to zero, repeat outer loop if R27 is not 0
                   ; The nested loops create a significant delay for the button debouncing

    ; After the delay, check if the button is still pressed (debounced state)
UPS: SBIS PINA,6   ; Skip the next instruction if bit 6 of PINA is set (button pressed)
    RJMP UPS       ; If button is not pressed (bounce up), loop back to UPS and keep checking

    ; Proceed here if the button press is stable after debouncing
    sbic PINA, 6       ; Skip if Bit in I/O Register is Cleared - this line seems to be redundant
                       ; or possibly incorrectly copied, as it's not functional after SBIS.

    rjmp switch6_pressed1  ; Relative jump to `switch6_pressed1` label
                           ; Here, the intended action is to decrease the frequency by 5%.

    RET            ; Return from the subroutine. Ends the debounce process and returns to
                   ; the main program flow if the button press is not confirmed.


switch7_pressed1:   ; Entry point for the +5% base frequency adjustment
    sbis PINA,7        ; Skip the next instruction if button 7 is still pressed
    RCALL DEB1         ; Call the debounce routine for button 7. If pressed again, proceed to +10% base frequency

    sbis PINA,6        ; Check if button 6 is pressed to decrease frequency back to base
    RCALL DEBRETURN    ; Call the debounce return routine, likely resetting to base frequency if button 6 is pressed

    sbis PINA,4        ; Check if button 4 is pressed to increase the duty cycle
    RCALL DEB5p1       ; Call the debounce routine specific for button 4 press, adjusting the duty cycle accordingly

    sbis PINA,3        ; Check if button 3 is pressed to decrease the duty cycle
    RCALL DEB5m1       ; Call the debounce routine specific for button 3 press, adjusting the duty cycle accordingly

    sbis PINA, 5       ; Checks for button 5 press, which is possibly intended to reset the frequency
    RCALL DEBRETURN    ; Call the debounce return routine, likely resetting to base frequency if button 5 is pressed

    ; Control the LED state based on the new frequency setting (+5%)
    LDI R16,0x7F           ; Prepare to turn on the LED by setting PORTD appropriately
    OUT PORTD,R16          ; Output to PORTD, turning on the LED (assuming active-low LEDs)
    RCALL ON1              ; Call a routine to maintain the LED's ON state for the adjusted (+5%) frequency duration

    LDI R16,0xFF           ; Prepare to turn off the LED by setting all bits of PORTD high
    OUT PORTD,R16          ; Turn off the LED by setting PORTD high (for active-low configuration)
    RCALL ON1              ; Call the same ON1 routine, likely to keep the LED off for the adjusted duration

    RJMP switch7_pressed1 ; Loop back to the start to continuously check for button inputs and maintain LED control
```

FIGURE 2: DEBOUNCING FOR VERIFYING BUTTON PRESSES

3.2.3. DELAY LOOPS FOR FREQUENCIES

Fine-tuning the base frequency of our PWM signal was a meticulous process that necessitated precise control over the timing of our signal's high and low states. We accomplished this by carefully crafting delay loops within our assembly code, which directly affected the period of our PWM waveform. By manipulating the delay durations, we could effectively increase or decrease the frequency of our signal.

For our base frequency of 1100 Hz, precise delay loops were established by setting up counters in registers R18 and R19. The routine labeled ON1 was designed for a +5% increase from the base frequency. The process involved a series of decrement (DEC) and increment (INC) operations on the R18 register, flanked by NOP operations to fine-tune the timing further. Each cycle of decrementing and incrementing created a brief pause, and repeating this pattern provided the necessary delay corresponding to the intended frequency.

For larger frequency adjustments, such as a +10% increase, we developed a second routine labeled switch7_pressed2. Here, similar delay loops were used, but with adjusted counter values to produce a shorter overall delay, thus leading to a higher frequency output.

Two complete iterations of the delay loop (AGAIN1 and HERE1) were necessary to achieve the precise timing required for our target frequencies. By adjusting the initial values loaded into R19 and R18, we could extend or shorten the delay intervals, allowing us to fine-tune the frequency in increments or decrements of 5%.

The structure of the delay loops was pivotal for the control of our PWM signal. It provided us with the flexibility to create a series of frequency settings that users could cycle through using the designated buttons on our interfacing board. The final implementation of these delay loops, which form the cornerstone of our frequency control system, can be observed in **Figure 3**. This attention to detail in the software enabled us to offer fine precision in controlling the PWM signal, ensuring that the output met the project's stringent requirements for frequency adjustments.

```
ON1:     LDI R19,3     //Delay for +5% base frequency
AGAIN1:  LDI R18,255
HERE1:    dec R18             ; Decrement and increment pattern to adjust timing
          inc R18
          dec R18
          inc R18
          dec R18
          inc R18
          dec R18             ; End of decrement and increment pattern
         BRNE HERE1
         NOP
         NOP
         NOP
         NOP
         NOP
         NOP
         DEC R19
         BRNE AGAIN1
         RET

switch7_pressed2:   sbis PINA,7   //If button 7 was pressed 2 times output +10% base frequency
                    RCALL DEB2    //Check for button 7 pressed 3 times
                    sbis PINA,6   //Check for button 6 pressed when we are in +10 base frequnecy takes us to +5% base frequency
                    RCALL DEB1A
                    sbis PINA,4
                    RCALL DEB5p2
                    sbis PINA,3
                    RCALL DEB5m2mirror
                    sbis PINA, 5  //Check for button 5 pressed taking us to the base frequency and duty cycle
                    RCALL DEBRETURN
                    LDI R16,0x7F
                    OUT PORTD,R16
                    RCALL ON2
                    LDI R16,0xFF
                    OUT PORTD,R16
                    RCALL ON2
                    RJMP switch7_pressed2
```

FIGURE 3: FREQUENCY DELAY LOOPS; TYPICAL

3.2.4. DELAY LOOPS FOR DUTY CYCLES

Adjusting the duty cycle of our PWM signal required a strategic approach to manipulating the duration of the 'on' versus 'off' times within a single period while maintaining the same frequency. Our approach was to establish two separate delay loops: one controlling the 'on' time and the other managing the 'off' time. Modifying these delays enabled us to change the proportion of the signal's high time to its total period, effectively altering the duty cycle.

For each variation of the base frequency, we implemented a specific function that allowed us to fine-tune the duty cycle by increments of ±5%, up to a maximum range of ±25%. The assembly code provided robust control over these adjustments through subroutines tailored for each duty cycle setting. The DUTY5p label, for example, begins a loop for a +5% increase in duty cycle.

Within this loop, we employed conditional checks using the SBIS instruction to monitor button presses. If a user wished to increase the duty cycle further, we called the corresponding routine (DEB10p), while a decrease or reset to base duty cycle was handled by routines DEBRETURN and LOOP, respectively.

The subroutines ON5p and OFF5p were key to implementing the desired duty cycle changes. In ON5p, we set up a delay loop with a counter in R17 and a nested loop counter in R18. The NOP operations and DEC instructions together created a precise delay corresponding to the 'on' time for the LED, directly reflecting the +5% duty cycle. Conversely, OFF5p defined the 'off' time with a slightly longer delay, indicating a reduced duty cycle.

The correct balance between the ON5p and OFF5p delays ensured that while we adjusted the duty cycle, the base frequency of 1100 Hz remained constant. By keeping one parameter fixed and only varying the other, we could fine-tune the duty cycle without affecting the frequency, crucial for applications requiring consistent timing.

This intricate arrangement of delay loops for duty cycle control formed an extensive section of our code, one part of nearly 7000 lines dedicated to this function. Each potential combination of frequency and duty cycle adjustments had its corresponding set of delay loops. One section of this code can be observed within **Figure 4.**

```
987     ON5p:      LDI R17,10      //Delay for +5% duty cycle when the LED is on
988      AGAIN5p:  LDI R18,130
989      HERE5p:   NOP
990                NOP
991                DEC R18
992                BRNE HERE5p
993                DEC R17
994                BRNE AGAIN5p
995                RET
996
997     OFF5p:     LDI R17,10    //Delay for +5% duty cycle when the LED is off
998     AGAINF5p:  LDI R18,158
999     HEREF5p:   NOP
1000               NOP
1001               DEC R18
1002               BRNE HEREF5p
1003               NOP
1004               NOP
1005               NOP
1006               NOP
1007               NOP
1008               DEC R17
1009               BRNE AGAINF5p
1010               RET
```

FIGURE 4: FREQUENCY AND DUTY CYCLE DELAY LOOP; TYPICAL

3.2.5. CREATIVE PORTION

For the creative element of our project, we decided to enhance the user experience by incorporating an interactive audio-visual feature. By pressing a button connected to PINA.2, users could activate a sequence that made all LEDs blink in a pattern and simultaneously output a ticking noise from the speaker. This not only demonstrated the versatility of the ATmega324B microcontroller in handling multiple outputs synchronously but also added an engaging aspect to our technical project.

We employed direct bit manipulation techniques, using commands like SBI (Set Bit in I/O Register) and CBI (Clear Bit in I/O Register), to turn individual LEDs on and off in a precise sequence. By alternating the port values, specifically targeting the Port D and Port E where the LEDs and speaker were connected, we achieved a blinking effect across the array of LEDs.

The ticking noise on the speaker, controlled by toggling PE4, was synchronized with the visual display. We crafted a dedicated routine that would trigger a short pulse on the speaker to mimic a ticking sound, aligning with the LED flash pattern. This was timed using a series of RCALL instructions to delay subroutines, which controlled the duration of the on and off states for both the LEDs and the speaker output.

This interactive feature highlighted the responsiveness of our system to user inputs and provided a tangible demonstration of the microcontroller's capability to drive multiple peripherals in harmony. The creativity of our approach was showcased in the way the LEDs and speaker worked together to create an entertaining and functional display, all while ensuring that the base functionality of the PWM control was preserved.

The full implementation of our creative feature, from the initialization of the speaker and LED ports to the looping control structure that managed the audio-visual pattern, can be observed in **Figure 5**. This creative twist not only added flair to our project but also served as a testament to the power and flexibility of embedded system design.

```
;*************************************************************************************
.equ PE4 = 4                ; Manually define PE4 if not defined
.equ START_TUNE = 2         ; PINA.2 is used to start the tune
.equ STOP_TUNE = 5          ; PINA.5 is used to stop the tune and return to base features

CREATIVE:   SBIS PINA,5
            JMP LOOP
            LDI R16,0xFF
            OUT PORTD,R16
            OUT PORTE,R16
            CBI PORTD,0
            LDI R16,0x7F
            OUT PORTD,R16
            LDI R16,0xEF
            OUT PORTE,R16     // RCALL SOUND
            RCALL ON10
            LDI R16,0xFF
            OUT PORTD,R16
            OUT PORTE, R16
            RCALL ON10        //RCALL SOUND
            RCALL ONC
            CBI PORTD,1
            RCALL ON9
            RCALL ON8
            RCALL ON
            CBI PORTD,2
            RCALL ONC
            SBI PORTD,0
            CBI PORTD,1
            CBI PORTD,2
            CBI PORTD,4
            RCALL ONC
            SBI PORTD,1
            CBI PORTD,2
            CBI PORTD,4
            CBI PORTE,5
            RCALL ONC
            SBI PORTD,2
            CBI PORTD,4
            CBI PORTE,5
            CBI PORTD,3
            RCALL ONC
            SBI PORTD,4
            CBI PORTE,5
            CBI PORTD,3
            CBI PORTD,5
            RCALL ON7
            RCALL ON6
            RCALL ON5
            RCALL ON4
            RCALL ONC
            SBI PORTE,5
            CBI PORTD,3
            CBI PORTD,5
            CBI PORTD,6
            RCALL ONC
            SBI PORTD,3
            CBI PORTD,5
            CBI PORTD,6
            CBI PORTD,7
            RCALL ONC
            RJMP CREATIVE
```

FIGURE 5: CREATIVE PORTION LOOP

4. PRESENTING

When presenting our final project, we encountered a minor glitch concerning the return to the base frequency after cycling through the incremented and decremented frequencies and duty cycles. Despite thorough testing, this issue surfaced during the live demonstration, highlighting the unpredictable nature of complex systems and the importance of real-time debugging.

Dr. Dua meticulously engaged with our project, incrementing and decrementing each frequency and duty cycle to examine the responsiveness and precision of our PWM control system. Each press of the buttons showcased the effectiveness of our debouncing logic, with the system reliably reacting to the inputs without registering false presses.

The presentation also covered the creative portion of our project. Dr. Dua activated the interactive audio-visual sequence, noting the coordination between the blinking LEDs and the rhythmic ticking from the speaker. This part of the demonstration went smoothly, effectively displaying our ability to integrate a simple feature other than frequency manipulation.

5. WORK DISTRIBUTION

- TRENTON CATHCART:

Software Implementation: Trenton took on the challenging task of developing the initial software modules for generating the PWM signals. This involved writing and testing the assembly code for frequency and duty cycle adjustments, as well as implementing the debouncing logic to handle button inputs.

Testing and Debugging: Trenton applied his analytical skills to the testing phase, conducting rigorous tests to validate the software and hardware interactions. He was pivotal in debugging the system, identifying glitches, and working on resolving them to ensure a smooth demonstration of the project.

- JAMIE MADISON:

Research and Documentation: Jamie was responsible for researching the technical specifications of the ATmega324B microcontroller and understanding the requirements for PWM signal generation. He compiled this research into a comprehensive document that served as a reference throughout the project. Additionally, Jamie meticulously documented the project's progress, ensuring that every stage was clearly outlined and accessible.

Hardware Integration: Jamie was in charge of the hardware aspect of the project, which included setting up the microcontroller on the interfacing board, wiring the LEDs and buttons,

and ensuring that all connections were secure and functional. He also managed the integration of the buzzer for auditory feedback.

6. CONCLUSION

In conclusion, our project set out to harness and augment the capabilities of the ATmega324B microcontroller by developing a PWM signal control system. Through our combined efforts, we succeeded in creating a system that could adjust PWM signals across a spectrum of frequencies and duty cycles. Despite the initial challenges and the complexity of manual assembly programming, we delivered a solution that adhered closely to the project's specifications.

The glitches experienced during the presentation were a reminder that perfection in real-world applications is a journey rather than a destination. These issues provided a learning opportunity that no simulation could offer, emphasizing the importance of resilience and adaptability in engineering practices.

Reflecting on our project journey, we recognize the value of meticulous planning, continuous testing, and the willingness to iterate on our designs. These practices were critical in enabling us to meet our project goals and learn from the process.

Looking forward, we are motivated to refine our project further. By addressing the glitches and exploring additional features, we can enhance the system's reliability and user experience. This project has not only fortified our understanding of PWM signal control and embedded system design but has also prepared us to tackle more advanced and challenging projects in the future.