**NAME: LƯƠNG THÀNH VỸ (2151280)**

# OBJECTIVE:

* Understand and utilize external interrupts, timer interrupts.
* Scan 7-segment LED displays and LED matrices using timer interrupts.
* Understand how to control and measure motor speed

# References:

* Experiment guide, chapters 3, 4, 5, 7,12.
* Atmel-2505-Setup-and-Use-of-AVR-Timers\_ApplicationNote\_AVR130.pdf.

# EXPERIMENT 1:

1. Programming to generate a 1 kHz frequency signal on pin PC0 using Timer 1 overflow interrupt. When Timer 1 overflows, the interrupt routine will toggle the PC0 pin and reset the counter register value.
2. Connect PC0 to the oscilloscope to measure the waveform.

(Note: The clock frequency for the CPU on the experimental kit is 8 MHz.)

# EXPERIMENT 2:

1. Repeat exercise 1 using Timer 1 in CTC mode, utilizing the COMPARE\_MATCH interrupt, to generate a pulse with a frequency of 100 Hz on pin PC0.
2. Configure the timer to generate a COMPARE\_MATCH interrupt every 1 ms. Inside the interrupt, use a counter to count the number of interrupt occurrences and control pin PC0 to generate a pulse with a frequency of 100 Hz.

Instructions: Increment the counter by 1 each time the interrupt occurs. If the counter reaches 5, toggle PC0 and reset the counter to 0.

1. Compile the program and observe the oscilloscope to verify the functionality of the program.

# EXPERIMENT 3:

1. Connect the necessary signals to control the 7-segment LED display module.
2. Utilize the COMPARE\_MATCH interrupt of Timer 1, as in Exercise 2, to display the numbers 1-2-3-4 on four 7-segment LED displays with a scanning frequency of 50 Hz. To measure the scanning frequency, toggle pin PC0 each time it switches to the next LED and measure this pulse on the oscilloscope.

(Refer to Chapter 4 of the experiment guide for further details.)

# EXPERIMENT 4:

Requirements:

1. Write a program to control the speed of a DC motor using PWM with a frequency of 1 kHz, using Timer 0. Control the speed increase/decrease using two buttons, where each button press increases/decreases the duty cycle by 5%. Allow the motor to start/stop and control the motor direction (forward/reverse) using two switches on a dip switch.

* Connect the motor to the kit.
* Connect the signals from the two switches on the dip switch to two AVR ports.
* Connect the signals from the two buttons to two AVR ports.
* Connect the signal from pin OC0B to a test point channel for measurement.
* Connect the signals from two port pins to control the forward/reverse direction to a single LED for status checking.

1. Compile, execute, and test the program by measuring the waveforms on an oscilloscope and observing the LED status when changing the dip switch and pressing the speed increase/decrease buttons.
2. Connect the PWM signal to MOTOR\_ENABLE and the control signals for the motor direction to MOTOR\_CTRL1 and MOTOR\_CTRL2 on J76 of the DC\_MOTOR module.
3. Test the operation of the system.
4. Measure the waveforms from the two A-B signals of the encoder and compare them in the two cases of the motor rotating forward or backward.

# EXPERIMENT 1:

1. **Answer the questions:  
   a. In Normal mode, do we need to reset the count register when entering the Overflow interrupt?**

**ANSWER:**

In Normal mode, when an overflow occurs, the count register (TCNT) automatically resets to 0. However, we can manually resetting the TCNT register to a predefined value to shorten the time it takes for the timer to reach the next overflow, thereby increasing the frequency of overflow events.

While it is not necessary to manually reset the TCNT register in Normal mode, but by doing so can be useful for generating a waveform with a specific frequency. To achieve the desired frequency, we need to configure the prescaler to set the general frequency range and adjust the initial/reset value of the TCNT register to predefined value in order to decrease the frequency within that range.

**b. Explain the values written to the timer configuration registers and prescaler.**

**ANSWER:**

The program generates 1 KHz signal by toggling PC0, the program creates a square wave with a 1 kHz frequency (period of 1 ms). This is done by toggling the PC0 pin during each Timer1 overflow interrupt every 0.5 ms via ISR.

The AVR operates at 8 MHz, which is equivalent to 0.125 us per cycle. In 0.5 ms, there are clock cycles. The Timer1 is a 16-bit timer capable of counting up to 65,535 without a prescaler, the number of increments of TCNT1 matches the number of clock cycles.

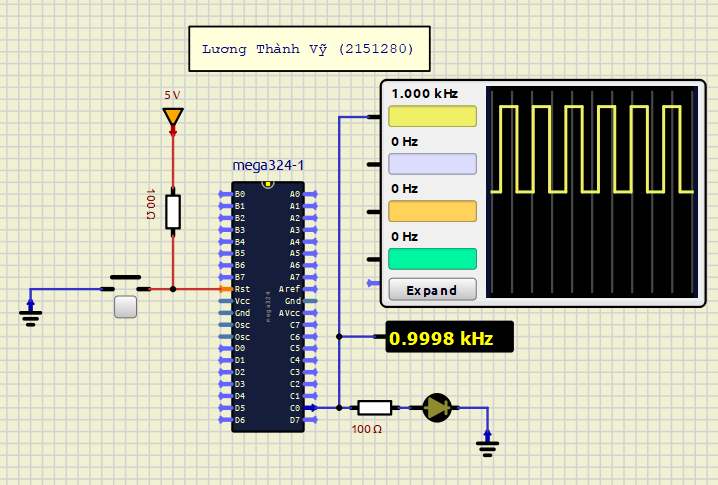
An overflow event occurs when the TCNT1 register increments past its maximum value of 65,535. To trigger the overflow every 4000 cycles, the TCNT1 register must be initialized to upon overflow.

1. Program source code with comments

|  |
| --- |
| ; Search in "m324Pdef.inc" for the correct Interrupt Vector  .ORG 0x0000 ; Reset vector  JMP MAIN  .ORG 0x001E ; Timer1 Overflow Interrupt vector  JMP ISR\_TIMER1\_OF  **MAIN:** LDI R16, HIGH(RAMEND) ; Initialize Stack, for Good practice  OUT SPH, R16  LDI R16, LOW(RAMEND)  OUT SPL, R16  SBI DDRC, 0 ; Set PC0 as Output  CBI PORTC, 0 ; Clear PC0  LDI R16, (1 << TOIE1) ; Enable Timer1 Overflow Interrupt  STS TIMSK1, R16  CALL TIMER1\_INIT ; Config and start Timer1  SEI ; Enable Global interrupt  **INF\_LOOP**:  RJMP INF\_LOOP ; Do "other tasks", let ISR generate 1 KHz wave    ; Internal clk = 8 MHz <==> 0.125 ns per cycle  ; f = 1 KHz ==> T = 1 ms  ; We need 0.5 ms delay ==> 4000 cycles  ; In Normal mode, TCNT will start (reset) at 65535 - 4000 = 61535  .EQU TCNT1\_INIT = 61552 ; Adjusted to match 1 KHz frequency on simulation  .EQU TCCR1A\_mode = 0b00000000 ; Normal mode  .EQU TCCR1B\_mode = 0b00000001 ; No Prescaler mode  **TIMER1\_INIT:**  LDI R16, HIGH(TCNT1\_INIT) ; Initialize Timer1 value  STS TCNT1H, R16  LDI R16, LOW (TCNT1\_INIT)  STS TCNT1L, R16  LDI R16, TCCR1A\_mode ; Enable OC1A and Mode  STS TCCR1A, R16  LDI R16, TCCR1B\_mode ; Start Timer  STS TCCR1B, R16  RET    **ISR\_TIMER1\_OF**:  PUSH R16 ; Reserved register values  PUSH R17    LDI R16, HIGH(TCNT1\_INIT) ; Reset Timer1 value  STS TCNT1H, R16  LDI R16, LOW (TCNT1\_INIT)  STS TCNT1L, R16  IN R16, PINC ; Read state of PC0  LDI R17, (1 << PC0) ; Create a mask for PC0  EOR R16, R17 ; Toggle PC0  OUT PORTC, R16  POP R17 ; Restore register values  POP R16  RETI |

**SIMULATION RESULT**

The initial/reset value of TCNT1 was adjusted from the calculated 61,535 to 61,552 to achieve a precise 1 kHz signal during simulation. This adjustment accounts for an additional 17 cycles beyond the original calculation. These 17 extra cycles correspond to the instructions needed to reset or initialize TCNT1 after the overflow is detected. Since these cycles occur before the reset, they are not counted by the timer, so they must be included in the new TCNT1 reset value to maintain accurate timing.



# EXPERIMENT 2:

1. Answer the questions:
   1. **In CTC mode, do we need to reset the count register when entering the COMPARE\_MATCH interrupt?**

**ANSWER:**

CTC (Clear Timer on Compare Match) mode functions similarly to Normal mode, with the key difference being that the TCNT register automatically resets to 0 when a compare match occurs instead of Overflow. We also can manually reset the TCNT register to a predefined value to shorten the interval until the next compare match, thereby increasing the frequency of compare match events. However, this is typically not necessary.

Manually resetting TCNT can be beneficial when precise waveform generation is needed. To achieve the desired frequency, the prescaler must be set to determine the general frequency range. Additionally, both the initial/reset value of the TCNT register and the Output Compare registers (OCRA and OCRB) must be adjusted to fine-tune the frequency within that range.

* 1. **What are the advantages of this mode compared to the configuration in Exercise 1?**

**ANSWER:**

CTC mode and Normal mode can both count the same number of cycles. However, CTC mode offers more flexibility because it allows the adjustment of the TOP (the value at which the timer resets) and BOTTOM (the initial/reset value of the timer). This reduces the need to manually reset the TCNT register. For example, to execute a specific number of cycles, you do not need to subtract the desired value from the maximum count as in Normal mode; instead, we can simply set the OCRA register to the target value.

The TCNT register automatically resets only when a compare match occurs with OCRA. The OCRB can be used for compare match events that trigger specific actions, such as generating an interrupt or toggling an output pin, but it does not reset the TCNT register.

**c. Explain the values written to the timer configuration registers and prescaler.**

**ANSWER:**

The program generates a 100 Hz pulse signal, which means the PC0 pin toggles via an ISR every 10 ms. To achieve this, a compare match must occur every 10 ms, corresponding to 80,000 clock cycles with an 8 MHz clock. Since Timer1 is a 16-bit timer capable of counting up to 65,535, this count exceeds its range without a prescaler. To address this, a prescaler of 8 is used to divide the clock, reducing the target TOP value to 10,000 cycles. Therefore, the OCR1A register should be set to 10,000, and Timer1 should be started with a prescaler of clk/8 to generate the desired pulse frequency.

For question b, to achieve a compare match every 1 ms, 8000 cycles need to be counted. This count fits within the 16-bit range of TCNT1, so a prescaler is not needed in this case. The OCR1A register should be set to 8000, which corresponds to 1 ms with an 8 MHz clock.

1. **Program source code with comments**

**ANSWER:**

1. **Generating 100 Hz pulse on pin PC0:**

|  |
| --- |
| ; Search in "m324Pdef.inc" for the correct Interrupt Vector    .ORG 0x0000 ; Reset vector  JMP MAIN  .ORG 0x001A ; Timer1 Compare Match A Interrupt vector  JMP COMPARE\_MATCH  **MAIN**: LDI R16, HIGH(RAMEND) ; Initialize Stach, for good practice  OUT SPH, R16  LDI R16, LOW(RAMEND)  OUT SPL, R16  SBI DDRC, 0 ; Set PC0 as Output  CBI PORTC, 0 ; Clear PC0  LDI R16, (1 << OCIE1A) ; Enable Output Compare Match A for Timer1  STS TIMSK1, R16  CALL TIMER1\_INIT ; Select CTC mode and start Timer  SEI ; Enable Global interrupt  **INF\_LOOP**:  RJMP INF\_LOOP ; Do "other tasks", let ISR generate 100 Hz pulse    ; Internal clk = 8 MHz <==> 0.125 ns per cycle  ; f = 100 Hz ==> T = 10 ms ==> 80000  ; 80000 cycles ==> OCR1A = 10000, Prescaler = 8  .EQU TCNT1\_INIT = 0 ; Initialize Timer, start at 0  .EQU OCR1A\_value = 9999 ; Adjusted to match 100 Hz on simulation  .EQU TCCR1A\_mode = 0b00000000 ; Choose mode CTC  .EQU TCCR1B\_mode = 0b00001010 ; No Prescaler  **TIMER1\_INIT**:  LDI R16, HIGH(TCNT1\_INIT) ; Initialize Timer1 value, not necessary  STS TCNT1H, R16  LDI R16, LOW (TCNT1\_INIT)  STS TCNT1L, R16  LDI R16, HIGH(OCR1A\_value) ; Set OCR1A value  STS OCR1AH, R16  LDI R16, LOW (OCR1A\_value)  STS OCR1AL, R16  LDI R16, TCCR1A\_mode ; Select CTC mode  STS TCCR1A, R16  LDI R16, TCCR1B\_mode ; Start Timer  STS TCCR1B, R16  RET    COMPARE\_MATCH:  SBI PORTC, 0 ; Pulse PC0  ;NOP  CBI PORTC, 0  RETI |

**SIMULATION RESULT**

The initial/reset value of TCNT1 was adjusted from the calculated 10 000 to 9 999 to achieve an accurate 100 Hz pulse signal during simulation. This adjustment is necessary because TCNT1 needs to count 10 000 cycles in total, but since counting starts from 0, it only needs to reach 9999. The cycles used within the ISR are included by the timer itself, as there are no instructions that modify the TCNT1 value, unlike in Normal mode where manual adjustments may be required.

A computer screen shot of a computer

Description automatically generated

1. **Configure the timer to generate a COMPARE\_MATCH interrupt every 1 ms and use it to generate 100 Hz pulse**

To generate a 100 Hz pulse signal, the PC0 pin needs to be pulsed every 10 ms. If the interrupt occurs every 1 ms, a counter is required to count 10 interrupt occurrences, pulsing the PC0 pin when the counter reaches 9 (starting from 0).

The problem instruction mentions toggling PC0 after 5 interrupt occurrences, which would be correct for generating a 100 Hz square wave. However, in this program, PC0 will generate a 100 Hz pulse signal rather than a 100 Hz square wave.

|  |
| --- |
| .DSEG  INT\_COUNTER: .DB 1  .CSEG  .ORG 0x0000 ; Reset vector  JMP MAIN  .ORG 0x001A ; Timer1 Compare Match A vector  JMP COMPARE\_MATCH  **MAIN**: LDI R16, HIGH(RAMEND) ; Initialize Stach, for good practice  OUT SPH, R16  LDI R16, LOW(RAMEND)  OUT SPL, R16  SBI DDRC, 0 ; Set PC0 as Output  CBI PORTC, 0 ; Clear PC0    CLR R16  STS INT\_COUNTER, R16  LDI R16, (1 << OCIE1A) ; Enable Output Compare Match A for Timer1  STS TIMSK1, R16  CALL TIMER1\_INIT ; Select CTC mode and start Timer  SEI ; Enable Global interrupt  **INF\_LOOP**:  RJMP INF\_LOOP ; Do "other tasks", let ISR generate 100 Hz pulse    ; Internal clk = 8 MHz <==> 0.125 ns per cycle  ; T = 1 ms ==> 8000  ; 8000 cycles ==> OCR1A = 8000, Prescaler = 1  .EQU TCNT1\_INIT = 0 ; Initialize Timer, start at 0  .EQU OCR1A\_value = 7999 ; -1 because it start at 0  .EQU TCCR1A\_mode = 0b00000000 ; Choose mode CTC  .EQU TCCR1B\_mode = 0b00001001 ; No Prescaler  **TIMER1\_INIT**:  LDI R16, HIGH(TCNT1\_INIT) ; Initialize Timer1 value, not necessary  STS TCNT1H, R16  LDI R16, LOW (TCNT1\_INIT)  STS TCNT1L, R16  LDI R16, HIGH(OCR1A\_value) ; Set OCR1A value  STS OCR1AH, R16  LDI R16, LOW (OCR1A\_value)  STS OCR1AL, R16  LDI R16, TCCR1A\_mode ; Select CTC mode  STS TCCR1A, R16  LDI R16, TCCR1B\_mode ; Start Timer  STS TCCR1B, R16  RET    **COMPARE\_MATCH**:  PUSH R16    LDS R16, INT\_COUNTER  CPI R16, 9 ; Pulse after counting 10 Interrupt, including 0  BREQ PULSE\_PC0  INC R16 ; Increment Counter  STS INT\_COUNTER, R16  POP R16  RETI  **PULSE\_PC0**:  CLR R16 ; Clear Counter  STS INT\_COUNTER, R16  SBI PORTC, 0 ; Pulse PC0  CBI PORTC, 0    POP R16  RETI |

**SIMULATION RESULT**

A computer screen shot of a computer

Description automatically generated

# EXPERIMENT 3:

1. Answer the questions:

**a. To achieve a scanning frequency of 50Hz, how long will one LED remain lit?**

**ANSWER:**

To achieve a scanning frequency of 50 Hz, all four 7-segment LEDs must be illuminated within each 20 ms cycle. Since only one 7-segment LED can be lit at a time, as they need to display different numbers and share a common anode pin, the 20 ms cycle must be divided equally among the four LEDs. ***This results in each LED being illuminated for 5 ms before switching the anode to the next LED.***

**b. In that case, what will be the frequency of pin PC0 (toggled each time the LED switches)?**

**ANSWER:**

The LED switches every 5 ms, and if pin PC0 toggles each time an LED switches, it will create a square wave with a period of 10 ms. ***This corresponds to a frequency of 100 Hz.***

**c. How many interrupt occurrences are required to switch the LED?**

**ANSWER:**

Since the COMPARE\_MATCH interrupt is used to switch the LEDs, only one interrupt occurrence is required for each LED switch. Timer1 must be configured so that a COMPARE\_MATCH occurs every 5 ms. Four interrupt occurrences are needed to complete a full refresh or scanning cycle. Each time a COMPARE\_MATCH occurs, the LED index (ranging from 0 to 3 for the four 7-segment LEDs) is switched, and the predefined numbers (1 to 4) are displayed accordingly.

1. **Program source code with comments**

**ANSWER:**

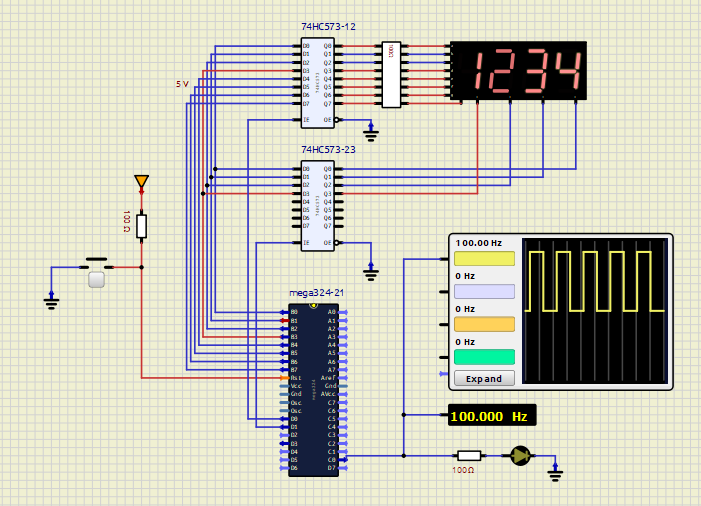
|  |
| --- |
| .ORG 0x0000 ; Reset vector  JMP MAIN  .ORG 0x001A ; Timer1 Compare Match A vector  JMP COMPARE\_MATCH  ; Lookup table for 7-segment codes  BCD\_TABLE: .DB 0xC0, 0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90,0x88,0x8  ; Lookup table for LED control  COMMON\_ANODE\_TABLE: .DB 0b00000001, 0b00000010, 0b00000100, 0b00001000  ; Store the BCD value to display  LED7\_NUM: .DB 0x01, 0x02, 0x03, 0x04  .EQU LED7SEG\_PORT = PORTB ; Two 74HC573 ICs (Data and Common Anode) use the same PORT  .EQU LED7SEG\_DDR = DDRB  .EQU nLEx\_PORT = PORTD ; Seperate PORT for controling LATCH pin  .EQU nLEx\_DDR = DDRD  .EQU nLE0\_PIN = 0  .EQU nLE1\_PIN = 1  .DEF LED7\_INDEX = R5    MAIN:  LDI R16, HIGH(RAMEND) ; Initialize Stach, for good practice  OUT SPH, R16  LDI R16, LOW(RAMEND)  OUT SPL, R16  SBI DDRC, 0 ; Set PC0 as Output  CBI PORTC, 0 ; Clear PC0    LDI R16, 3 ; Initialize LED7\_INDEX (R5)  MOV LED7\_INDEX, R16  RCALL LED\_7SEG\_PORT\_INIT  LDI R16, (1 << OCIE1A) ; Enable Timer1 Output Compare Match A Interrupt  STS TIMSK1, R16  CALL TIMER1\_INIT ; Select CTC mode, start Timer  SEI ; Enable Global interrupt  INF\_LOOP: JMP INF\_LOOP ; Do "other tasks", let to ISR display 7-segment LEDs  LED\_7SEG\_PORT\_INIT:  PUSH R20 ; Save R20 before using it  LDI R20, 0xFF ; Set LED7SEG port as output  OUT LED7SEG\_DDR, R20  ; Set as output for signal each common Anode of 7SEG LEDs  LDI R20, nLEx\_DDR  ORI R20, (1 << nLE0\_PIN) | (1 << nLE1\_PIN)  OUT nLEx\_DDR, R20  POP R20  RET  ; INPUT: R27 = Num  ; R26 = Index (from 0 -> 3, as for LED\_0 to LED\_3)  DISPLAY\_7SEG:  PUSH R16 ; Save R16 and R20 before using it  PUSH R20    CLR R16  LDI ZH, HIGH(BCD\_TABLE << 1) ; Z = base address of the look-up table  LDI ZL, LOW (BCD\_TABLE << 1)  ADD ZL, R27 ; Add R27 to Z (16-bit)  ADC ZH, R16 ; Add carry to ZH if needed    LPM R20, Z ; Load the code to the 7SEG pins  OUT LED7SEG\_PORT, R20  SBI nLEx\_PORT, nLE0\_PIN ; Pulse the latch to update  NOP  CBI nLEx\_PORT, nLE0\_PIN    ; Z = base address of the look-up control table  LDI ZH, HIGH(COMMON\_ANODE\_TABLE << 1)  LDI ZL, LOW (COMMON\_ANODE\_TABLE << 1)  ADD ZL, R26 ; Add R27 to Z (16-bit)  ADC ZH, R16 ; Add carry to ZH if needed  LPM R20, Z ; Load the code to the 7SEG pins  OUT LED7SEG\_PORT, R20  SBI nLEx\_PORT, nLE1\_PIN ; Pulse the latch to update  NOP  CBI nLEx\_PORT, nLE1\_PIN  POP R20 ; Restore the temporary register  POP R16  RET      ; Internal clk = 8 MHz <==> 0.125 ns per cycle  ; Refresh rate = 50 Hz ===> Refresh period = 20 ms  ; For four 7-segment LEDs, there is 5 ms delay before next LED is lit up  ; We need 5 ms delay ==> 40000 cycles ==> OCRA = 40000, Prescaler = 1  .EQU TCNT1\_INIT = 0 ; Initialize Timer  .EQU OCR1A\_value = 39999 ; Count 40000 cycles, -1 because it starts from 0  .EQU TCCR1A\_mode = 0b00000000 ; Select CTC mode  .EQU TCCR1B\_mode = 0b00001001 ; No Prescaler  TIMER1\_INIT:  PUSH R16  LDI R16, HIGH(TCNT1\_INIT) ; Initialize Timer1 value  STS TCNT1H, R16  LDI R16, LOW (TCNT1\_INIT)  STS TCNT1L, R16  LDI R16, HIGH(OCR1A\_value) ; Set Compare Register A value  STS OCR1AH, R16  LDI R16, LOW (OCR1A\_value)  STS OCR1AL, R16  LDI R16, TCCR1A\_mode ; Choose mode CTC  STS TCCR1A, R16  LDI R16, TCCR1B\_mode ; Start Timer  STS TCCR1B, R16  POP R16  RET    COMPARE\_MATCH:  PUSH R16  PUSH R26  PUSH R27  CLR R16 ; Clear R16, to add carry to ZH  MOV R26, LED7\_INDEX ; R26 = Index  LDI ZH, HIGH(LED7\_NUM << 1) ; Z = List of number to display  LDI ZL, LOW (LED7\_NUM << 1)  ADD ZL, R26 ; Z\* = Z\* + Index  ADC ZH, R16  LPM R27, Z  CALL DISPLAY\_7SEG ; Display Number    CPI R26, 0  BRNE TIMER1\_COMP\_ISR\_CONT ; If LED7\_INDEX = 0, Reset it to 3  LDI R26, 4 ; Else, decrease LED7\_INDEX  TIMER1\_COMP\_ISR\_CONT:  DEC R26  MOV LED7\_INDEX, R26 ; Save LED7\_INDEX  IN R26, PINC ; Get the state of PC0  LDI R27, (1 << PINC0) ; Set a mask to toggle PC0  EOR R26, R27 ; Apply Mask and toggle PC0  OUT PORTC, R26  POP R27  POP R26  POP R16  RETI |

**SIMULATION RESULT**

Since the order of numbers 1-2-3-4 is not specified in the problem, the LEDs in this simulation might display the numbers in reverse order compared to what is required. If needed, this can be adjusted easily by simply reverse the order of the predefined number array stored in the “LED7\_NUM” variable.

Note that modifying LED7\_INDEX to start from 0 and increment to 3, instead of counting down from 3 to 0 as in the current program, will not reverse the displayed number order. This is because both the index and the number to be displayed are determined by adding LED7\_INDEX to the base address of the number arrays LED7\_NUM and COMMON\_ANODE\_TABLE, which are loaded into register Z to select the number or index from the arrays.A computer screen shot of a circuit board

Description automatically generated

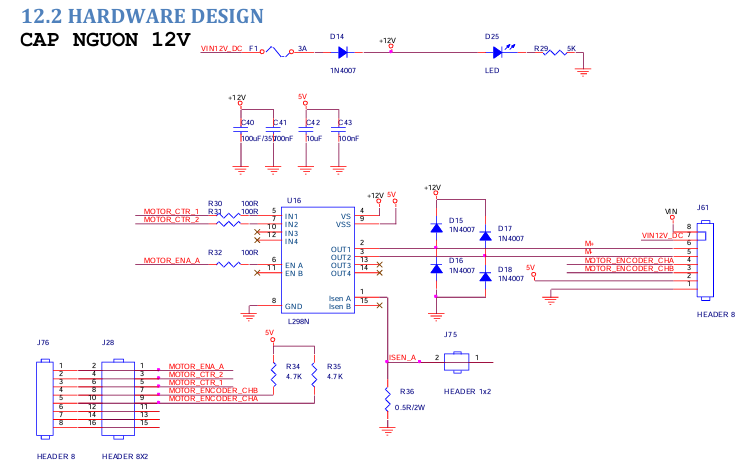


* PC0 is toggled 4 times in one scanning cycle 🡪 fPC0 = 2 x fscanning = 100 Hz
* Scanning frequency = 50 Hz

# EXPERIMENT 4:

1. **Answer the questions:**

**a. Describe the connections on the kit.**

**ANSWER:** ****

The L298N module can control two motors, with each motor requiring three control pins: two INx pins for direction control and an EN pin for speed control and motor activation.

On the experiment kit, only IN1, IN2, ENA (MOTOR\_CTR\_1, MOTOR\_CTR\_2, MOTOR\_EN\_A) are accessible externally for controlling one motor, while the other control pins are either unused or hardwired.

"The outputs on the experiment kit include MOTOR\_ENCODER\_CHA, MOTOR\_ENCODER\_CHB, and the ISENA pin, which are used to monitor the motor's direction and speed, as well as the current drawn by the motor.

The outputs of the L298N module, OUT1 and OUT2, are internally connected to two LEDs in both forward and reverse bias to indicate the motor's rotation direction. The L298N IC in the module is powered by a 5V supply, but the motor itself is driven by a 12V source, signified by an indicator LED0 - D25when powered on. This means that the output at the OUT1 and OUT2 pins is 12V, while the control signals at the IN1, IN2, and ENA pins are 5V.

The ISENA pin is internally connected to one side of the H-Bridge associated with Channel A (each channel controls one motor, and there are two channels). By placing a low-value current sense resistor between this pin and ground, the output analog signal across the resistor can be measured with ADC on AVR to monitor the current drawn by the motor. This feature is useful for applications involving motor load monitoring, stall detection, or current limiting. However, in this experiment, we will not use this output pin to measure the current drawn by the motor.

**b. Capture the waveform of the 2 encoder channels in both the forward and reverse rotation cases.**

**ANSWER:**

* **The Encoder pins are not found on my Simulation Tool**

To measure the motor's actual speed, an encoder is attached to the motor shaft. Encoders typically produce two phased signals, A and B, which help determine both the motor's direction and position. Some encoders also have a Z pulse that indicates when the motor reaches the zero position, useful for tracking complete revolutions.

In this setup, the motor includes a 2-channel encoder with signals A and B connected to header J28, although only channel A is used for speed measurement. The encoder outputs 11 pulses for each full revolution of the motor shaft. By counting these pulses over a 1-second interval, we can calculate the number of revolutions the motor completes per second.

1. **Program source code with comments**

**ANSWER**:

1. **How buttons and switches are read by AVR to control motor**

There are 2 switches and 2 buttons used for motor control, and we can use them to trigger an interrupt to handle motor operations through an ISR. In this program, all switches and buttons are configured to trigger a Pin Change Interrupt on the same PORT.

* The Pin Change Interrupt to be triggered both when pressing and releasing. When a Pin Change Interrupt is detected on the button pins, the program checks if the state is LOW (pressed). If it is LOW, the ISR jumps to the subroutine that controls the motor. If the state is HIGH (released), the ISR skips that button and checks the other button or switches.
* The switches have states that correspond to different control modes (changing direction, turning the motor on or off). When a Pin Change Interrupt is detected, the program checks the state of the switch pins and jumps to the appropriate subroutine in the ISR for the specific function associated with that state.

Since all 4 inputs share the same PORT for Pin Change Interrupts, the buttons are checked first as they respond only when pressed. The switches, on the other hand, are used to maintain continuous control of the motor.

1. How duty cycle of is modifies when an Interrupt is triggered