EXPERIMENT 6 - PWM GENERATION USING MSP43x

LAB TITLE AND CODE: EMBEDDED COMPUTING LAB (MCCE283)
EXPERIMENT NUMBER: 6
DATE: 24/05/2022, TUESDAY

★ **AIM:**
Develop a program for edge and centre-aligned pulse width modulation (PWM) generation using an MSP43x microcontroller. Also generate PWM based on ADC input.

a) **EDGE ALIGNED PWM GENERATION:**

① Configure functionality of P2.7 as simple GPIO pin.
② Configure direction of P2.7 as output for timer.
③ Configure timer A0.4 as PWM pin. Set PWM period and duty cycle.
④ Set the following bits of TAxCCTLn Register as follows-
(i) Bit 8 - Capture mode to 0: Compare mode
(ii) Bit 7-5 - Output mode to 111: Reset/set
(iii) Bit 4 - Capture/compare interrupt enable to 0: Interrupt disabled
(iv) Bit 0 - capture/compare interrupt flag to 0: No interrupt pending
⑤ Set the following bits of TAxCTL (TimerA Control Register) as follows-
(i) Bit 0 - TimerA Interrupt Flag to 0: Timer did not overflow
(ii) Bit 1 - TimerA Interrupt Enable to 0: Disabled
(iii) Bit 2 - TimerA clear (Clear TAxR)
(iv) Bit 5-4 - Mode control to 01: Up mode: Timer counts up to TAxCCR0
(v) Bit 6-7 - Input divider to 00: divide by 1.
(vi) Bit 8-9 - TimerA clock source select to 10: SMCLK (internal clock)
⑥ Define an infinite while-loop since an embedded program never stops executing.

PTO

→ C PROGRAM CODE –

```
#include "msp.h"

// Main Function :
int main (void)
{
    // P2.7/PM_TA0.4 :-
    P2 → SEL0 |= 0x80; // configure functionality of P2.7 as simple GPIO pin
    P2 → SEL1 &= ~0x80;
    P2 → DIR |= 0x80; // configure direction of P2.7 as output for timer

    // Configure timer A0.4 as PWM pin :-
    TIMER_A0 → CCR[0] = 999-1; // PWM Period
    TIMER_A0 → CCR[4] = 250; // PWM Duty Cycle
    TIMER_A0 → CCTL[4] = 0xE0; // compare mode; Reset/set output mode; Interrupt disabled, No interrupt pending

    /* Timer did not overflow; Disable interrupt; clear TimerA; Up mode; Input divider - Divide by 1; clock source select - SMCLK (internal clock) */
    TIMER_A0 → CTL = 0x0214;

    // Infinite Loop (An embedded program does not stop):
    while (1) {}
}
```

B) Centre Aligned PWM Generation

① Configure functionality of P2.7 as simple GPIO pin.
② Configure direction of P2.7 as output for timer.
③ Configure timer A0.4 as PWM pin. Set PWM period and duty cycle.

a) Edge aligned PWM Generation :

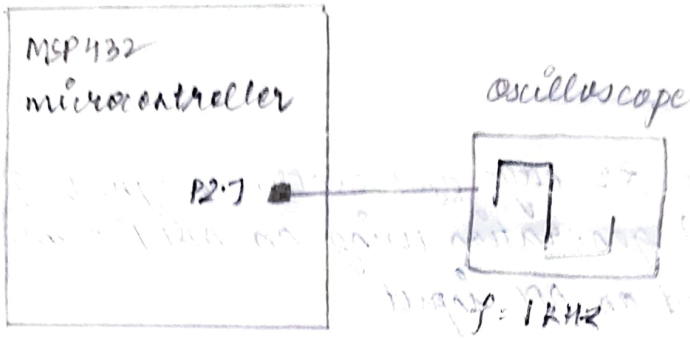

Figure 1- Circuit Diagram

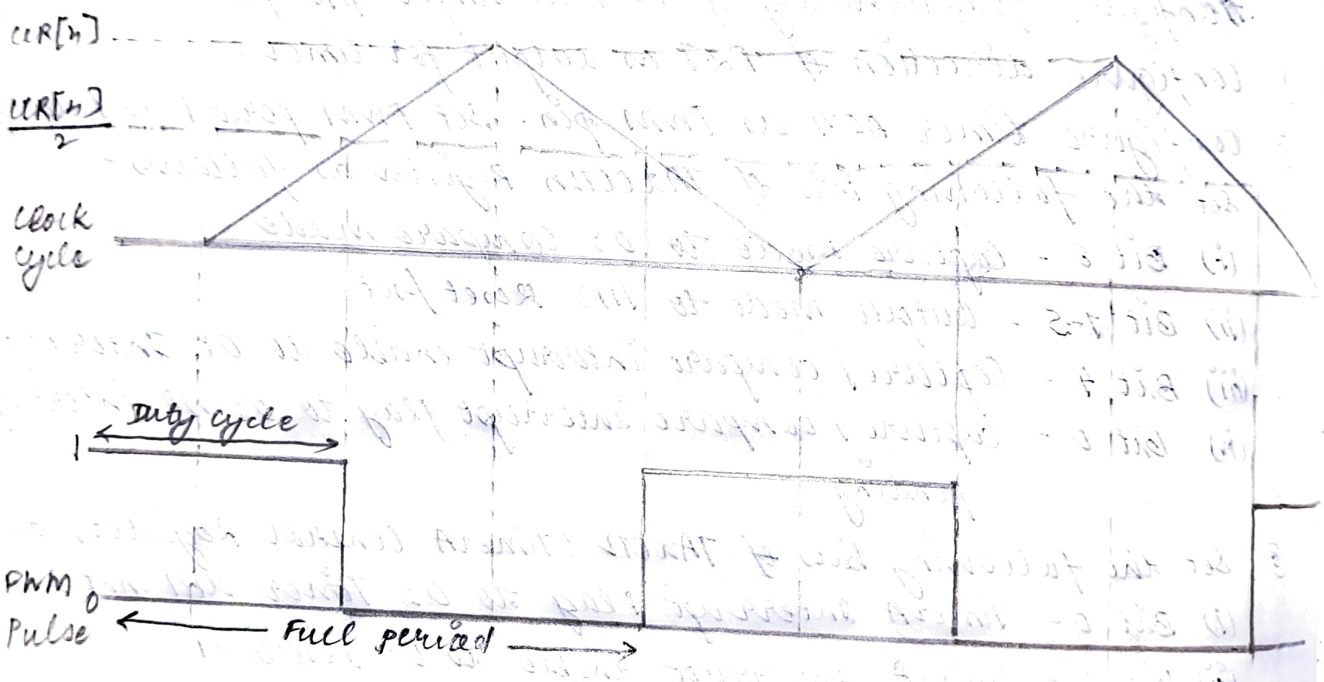

Figure 2- PWM Pulse and Timing Diagram

④ Set the following bits of TAxCCTLn Register as follows -
(i) Bit 8 - Capture mode to 0 : compare mode
(ii) Bit 7-5 - output mode to 010 : Toggle / reset
(iii) Bit 4 - Capture / compare interrupt enable to 0 : Interrupt disabled.
(iv) Bit 0 - Capture / compare interrupt flag to 0 : No interrupt pending

⑤ Set the following bits of TAxCTL (Timer A control Register) as follows -
(i) Bit 0 - TimerA Interrupt Flag to 0 : Timer did not overflow.
(ii) Bit 1 - TimerA Interrupt enable to 0 : Disabled
(iii) Bit 2 - TimerA clear (clear TAxR).
(iv) Bit 5-4 - Mode control to 11 : Up/down mode : Timer counts up to TAxCCR0 then down to 0.
(v) Bit 6-7 - Input divider to 00 : divide by 1.
(vi) Bit 8-9 - TimerA clock source select to 10 : SMCLK (internal clock)

⑥ Define an infinite while-loop since an embedded program never stops executing.

→ C PROGRAM CODE -

```
#include "msp.h"

// Main Function :
int main (void)
{
    // P2.7 / PWM_TA0.4 :-
    P2 -> SEL0 |= 0x80; // configure functionality of P2.7 as simple GPIO pin
    P2 -> SEL1 &= ~0x80;
    P2 -> DIR |= 0x80; // Configure direction of P2.7 as output for timer

    // configure timer A0.4 as PWM pin :-
    TIMER_A0 -> CCR[0] = 400 - 1; // PWM period
    TIMER_A0 -> CCR[4] = 250; // PWM Duty Cycle
```

b) Center edge PWM generation :

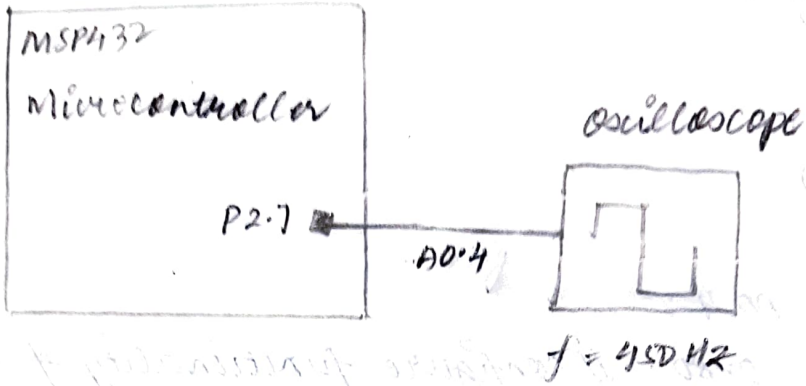

Figure 1- Circuit Diagram

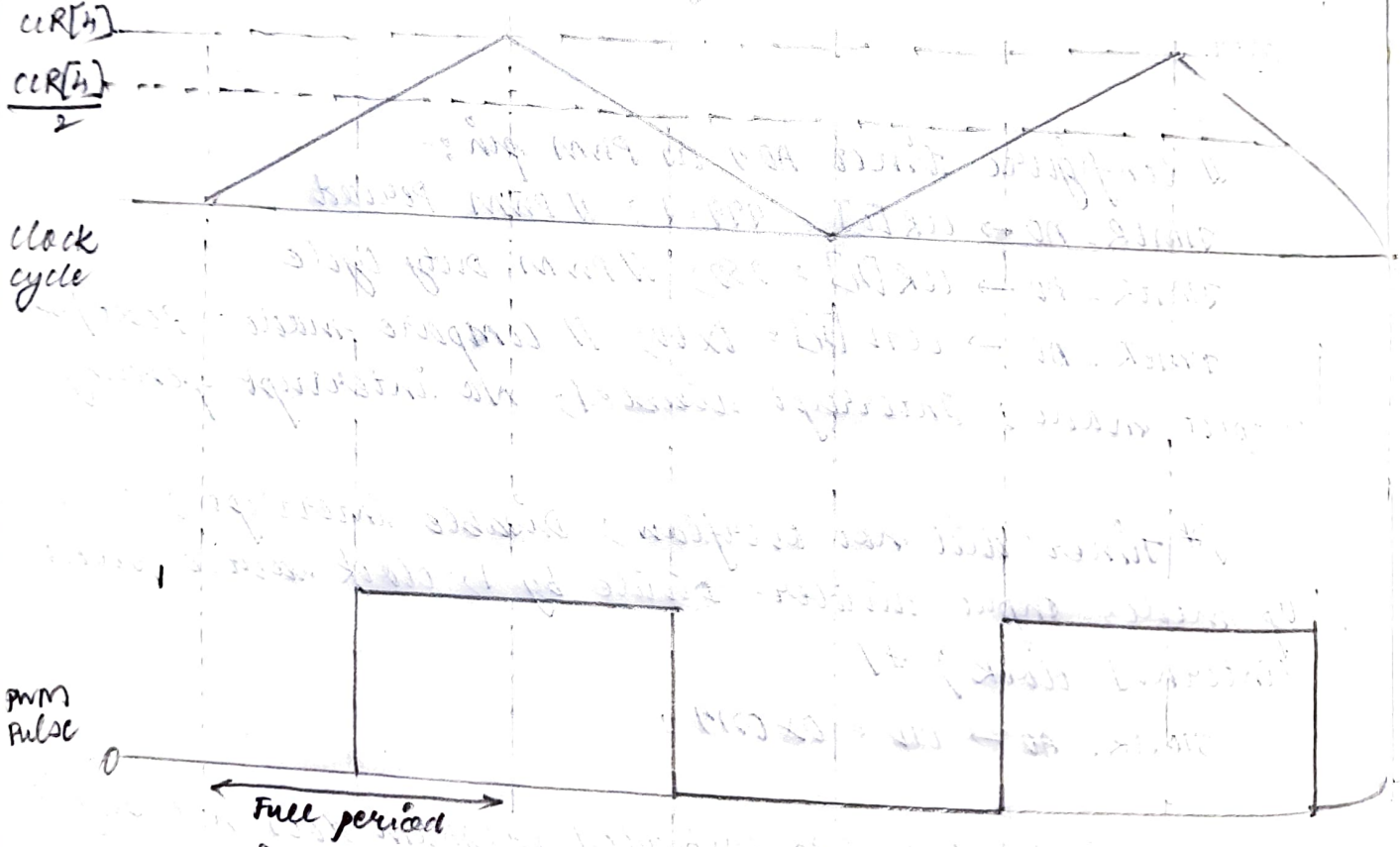

Figure 2 - Pulse and Timing Diagram

TIMER_A0 → CCTL[4] = 0x40; // compare mode; Toggle/reset output mode; Interrupt disabled; No interrupt pending

/* Timer did not overflow; Disable interrupt; clear TimerA; up/down mode; Input divider - Divide by 1; clock source select - SMCLK (internal clock) */

TIMER_A0 → CTL = 0x0234;

// Infinite Loop (An embedded program does not stop):
while (1) {}
}

c) PWM GENERATION BASED ON ADC INPUT:

1. Configure functionality of P2.5 as simple GPIO pin. Configure direction of P2.5 as output for timer.
2. Configure functionality of P4.7 as simple GPIO pin. Configure direction of P4.7 as input for ADC.
3. Power ON should be disabled during configuration. Sample-and-hold pulse-mode select, sysclk, 32 sample clocks and software trigger.
4. Set resolution as 12-bit (14 clock cycle conversion time).
5. To convert for memory register 5, repeat single channel (sequence sample is active) and enable ADC after conversion.
6. In an infinite while-loop, perform the following operations-
   (i) Wait for ADC conversion to complete.
   (ii) Read ADC data register.
   (iii) Set PWM period and duty cycle.
   (iv) Set the following bits of TAxCCTLn register as follows -
      a. Bit 8 - Capture mode to 0: Compare mode
      b. Bit 7-5 - Output mode to 010: Toggle/reset
      c. Bit 4 - Capture/compare interrupt enable to 0: Interrupt disabled
      d. Bit 0 - Capture/compare interrupt flag to 0: No interrupt pending

P.T.O.

(V) Set the following bits of TAxCTL (TimerA control register) as follows

a. Bit 0 - TimerA Interrupt Flag to 0 : Timer did not overflow.
b. Bit 1 - TimerA Interrupt enable to 0 : Disabled
c. Bit 2 - TimerA clear (clear TAxR).
d. Bit 5-4 - Mode control to 11 : up/down mode : Timer counts up to TAxCCR0 then down to 0.
e. Bit 6-7 - Input divider to 00 : divide by 1.
f. Bit 8-9 - TimerA clock source select to 10 : SMCLK (internal clock)

→ C PROGRAMS CODE -

```
#include "msp.h"

// Main Function :-
int main (void)
{
  int result;

  P2->SEL0 |= 0x20; // Configure functionality of P2.5 as simple GPIO pin
  P2->SEL1 &= ~0x20;
  P2->DIR |= 0x20; // Configure direction of P2.5 as output for timer

  P4->SEL1 |= 0x80; // Configure functionality of P4.7 as simple GPIO pin
  P4->SEL0 |= 0x80; // Configure direction of P4.7 as input for ADC

  ADC14->CTL0 = 0x00000010; // Power ON should be disabled during configuration
  ADC14->CTL0 |= 0x04080300; // sample-and-hold pulse-mode select, sysclk, 32 sample clocks, software trigger
```

1) PWM GENERATION USING ADC INPUT :

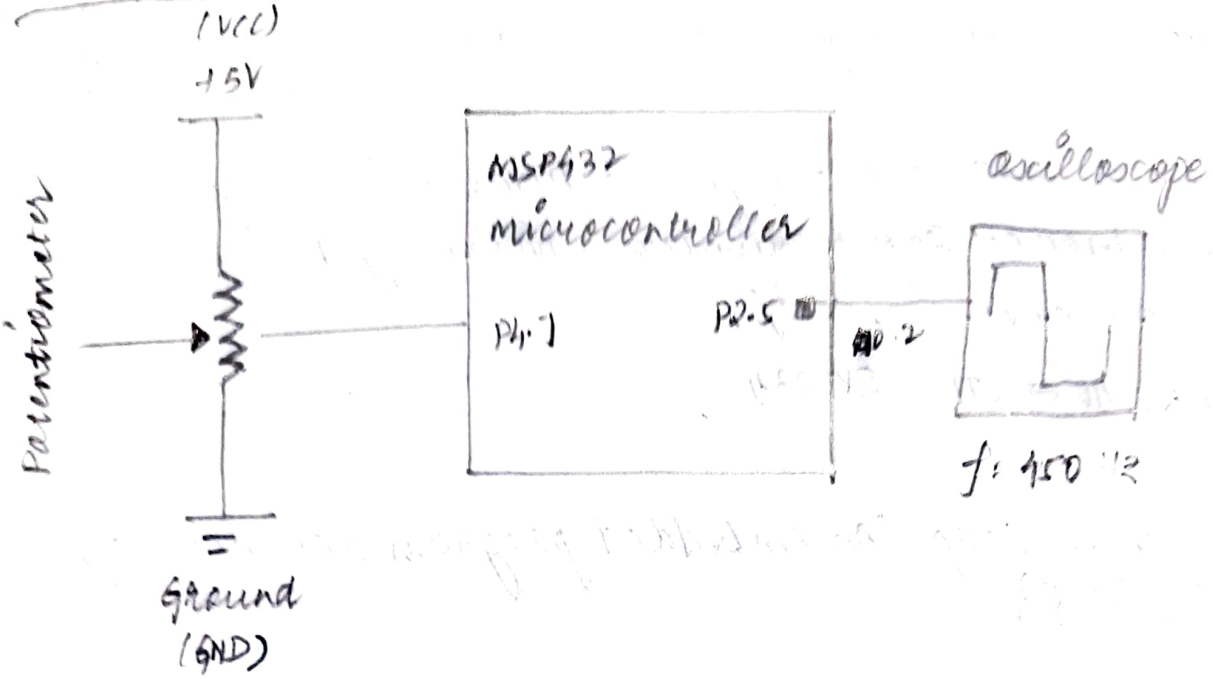

Figure - Circuit Diagram

```
ADC14 → CTL1 = 0x00000020; // Set resolution as 12-bit (14 clock cycle conversion time)

// Convert for memory register 10:-
ADC14 → CTL1 |= 0x000A0000; // Repeat single channel, sequence sample is active
ADC14 → CTL0 |= 2; // Enable ADC after configuration

// Infinite loop (An embedded program does not stop):-
while (1)
{
  while (!(ADC14 → IFGR0)) // Wait for ADC conversion to complete
  result = ADC14 → MEM[10]; // Read ADC data register

  TIMER_A0 → CCR[0] = 1000-1; // PWM period
  result = result >> 8;
  TIMER_A0 → CCR[2] = result; // PWM duty cycle
  TIMER_A0 → CCTL[2] = 0x40; // Compare mode; Toggle/reset output mode; Interrupt disabled; No interrupt pending

  /* Timer did not overflow, Disable interrupt; clear TimerA, Up/Down mode; Input divider - Divide by 1, clock source select- SMCLK (Internal clock) */

  TIMER_A1 → CTL = 0x0234;
}
}
```

d) SINUSOIDAL SIGNAL PWM GENERATION:

① Initialize a variable "flag" equal to zero.
② Configure functionality of P2.7 as simple GPIO pin.
③ Configure direction of P2.7 as output for timer.

④ Configure timer A0.4 as PWM pin. Set PWM period and duty cycle.

⑤ Set the following bits of TAxCCTLn Register as follows-
(i) Bit 8 - Capture mode to 0: compare mode
(ii) Bit 7-5 - Output mode to 111: Reset/set
(iii) Bit 4 - Capture/compare interrupt enable to 1: Interrupt enabled
(iv) Bit 0 - Capture/compare interrupt flag to 0: No interrupt pending

⑥ Set the following bits of TAxCTL (TimerA control Register) as follows-
(i) Bit 0 - TimerA Interrupt flag to 0: Timer did not overflow
(ii) Bit 1 - TimerA Interrupt Enable to 0: Disabled
(iii) Bit 2 - TimerA Clear (Clear TAxR)
(iv) Bit 5-4 - Mode control to 01: Up mode: Timer counts up to TAxCCR0
(v) Bit 6-7 - Input divider to 00: divide by 1.
(vi) Bit 8-9 - TimerA clock source select to 10: SMCLK (Internal clock)

⑦ In an infinite while-loop, perform the following operations -
(i) If interrupt flag is set, assign TAxCCTL[n] to 0xF0.
(ii) If TAxCCR register is set to full time period, decrement duty cycle by step size.
(iii) Else if TAxCCR register is set to 0, increment duty cycle by step size after.
(iv) Else, increment or decrement according to "flag".

① SINUSOIDAL SIGNAL - PWM GENERATION :

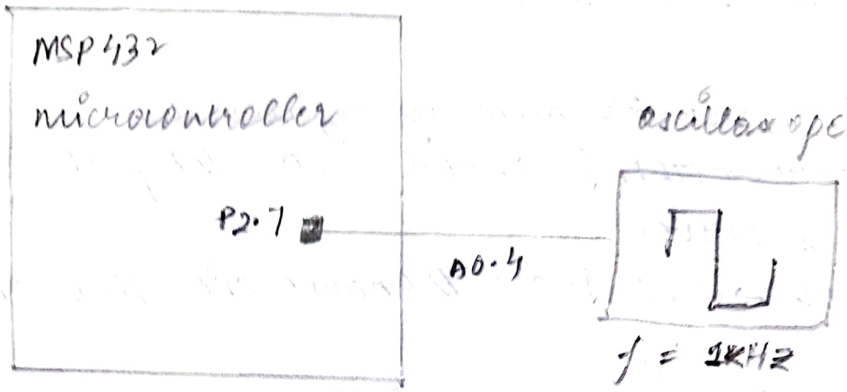

Figure 1 - Circuit Diagram

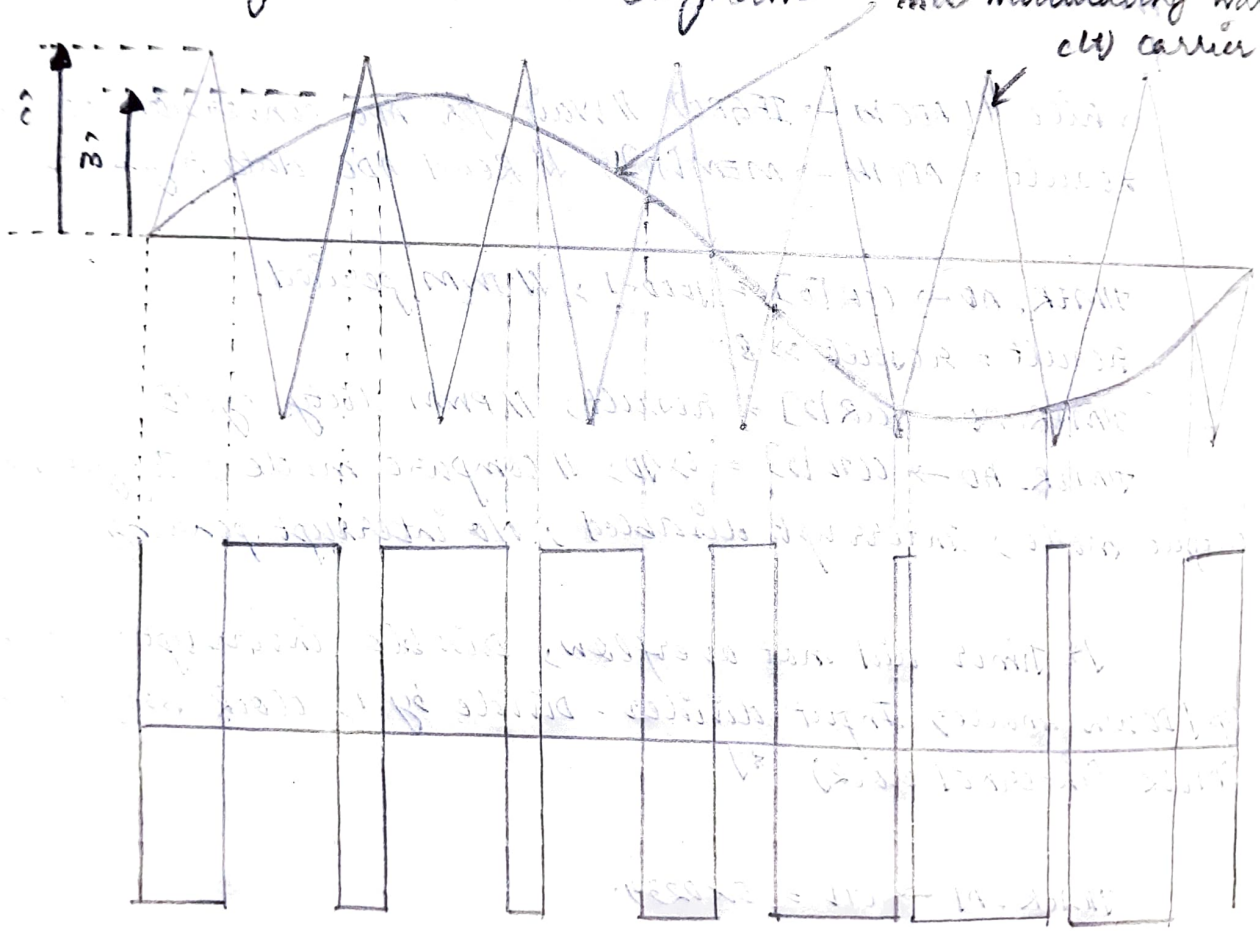

Figure 2 - Pulse and Timing Diagram

→ C PROGRAM CODE -

```
#include "msp.h"
int flag = 0;

// Main Function :
int main (void)
{
  P2 -> SEL1 |= 0x80; // Configure functionality of P2.7 as simple GPIO pin
  P2 -> SEL0 &= ~0x80;
  P2 -> DIR |= 0x80; // Configure direction of P2.7 as output for timer

  // Configure timer A0.4 as PWM pin :-
  TIMER_A0 -> CCR[0] = 999 - 1; // PWM Period
  TIMER_A0 -> CCR[4] = 250; // PWM Duty Cycle
  TIMER_A0 -> CCTL[4] = 0xF0; // Compare mode, Reset/set output mode; Interrupt enabled; No interrupt pending

  /* Timer did not overflow; Disable interrupt; Clear TimerA; Up mode; Input divider - Divide by 1; clock source select - SMCLK (internal clock) */

  // Infinite Loop (An embedded program does not stop):
  while (1)
  {
    while (TIMER_A0 -> CCTL[4] & 0xF1);
    TIMER_A0 -> CCTL[4] = 0xF0;

    if (TIMER_A0 -> CCR[4] == 1000)
    {
      flag = -1;
      TIMER_A0 -> CCR[4] += flag * 200;
    }
```

```
    else if (TIMER_A0->CCR[4] == 0)
    {
      flag = 1;
      TIMER_A0->CCR[4] += flag * 200;
    }

    else
      TIMER_A0->CCR[4] += flag * 200;
  }
}
```

* **RESULT:**

Thus, developed a program for edge and centre-aligned pulse width modulation (PWM) generation using an MSP43x microcontroller. Also, generated PWM based on ADC input. All simulation results were verified successfully.