University of California, Riverside

Lab #4 Report

Timer and PWM Operations

EE 128 Section 21, Fall 2025

Name: Muneer Al Jufout

Student ID: 862410258

Lab Partner: Sameer Anjum

Lab Partner ID: 862422332

TA: Johnson Zhang

Abstract

In this lab, we worked with the FRDM-K64F board and an Arduino Uno to learn how to measure a PWM signal. The Arduino created a PWM signal, and the K64F read it using the input-capture feature of the FlexTimer. The K64F calculated the duty cycle of the signal and showed the percentage on two 7-segment displays. Overall, we were able to generate, measure, and display different PWM duty-cycle values successfully.

Experiment System Specification

For this lab, Port C was used to drive one 7-segment display (excluding pin 6), and Port D was used to drive the second 7-segment display. The Arduino was connected to the FRDM-K64F by tying Arduino GND to the K64F GND and routing the Arduino PWM output to the K64F PWM input-capture pin.

Flowchart

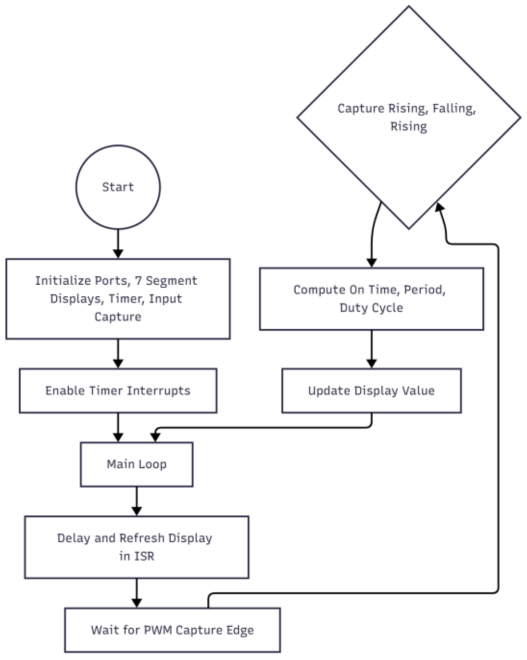
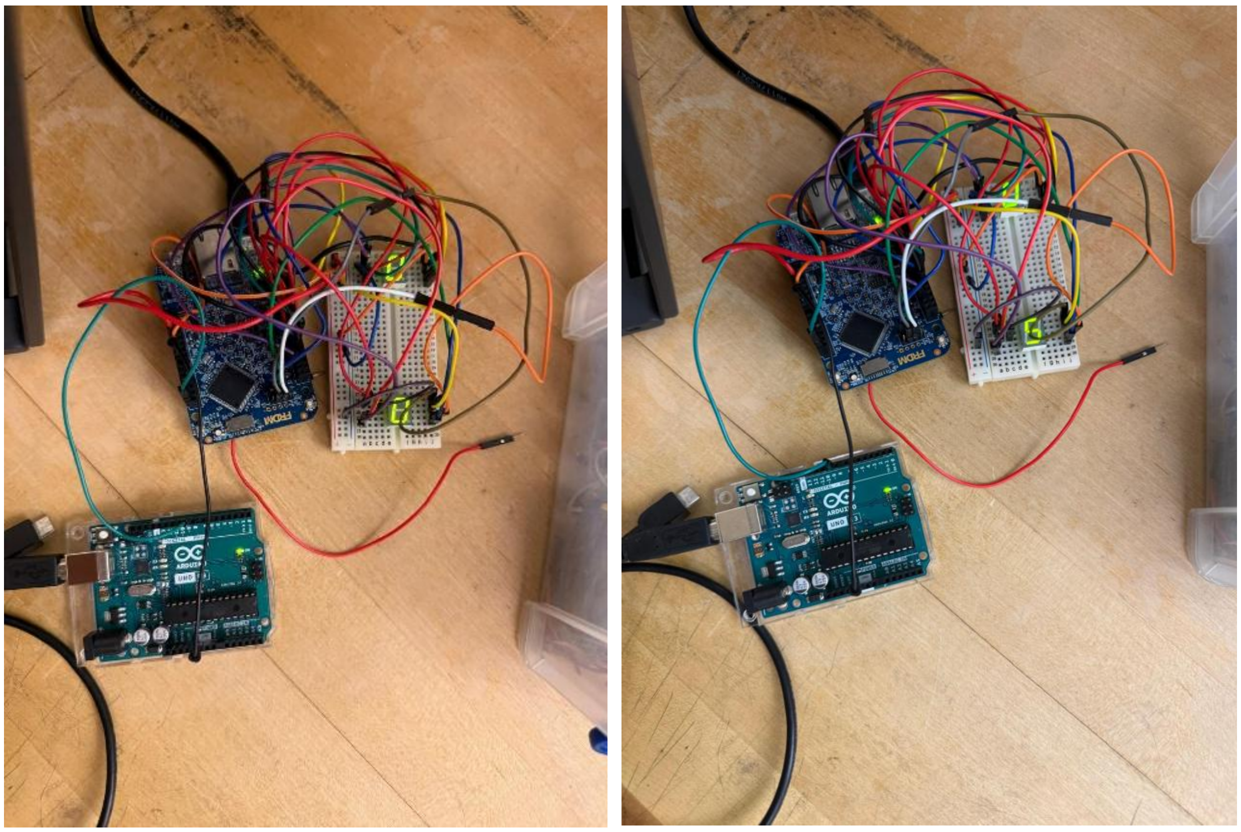

Capture Rising, Falling, Rising
Start
Initialize Ports, 7 Segment Displays, Timer, Input Capture
Compute On Time, Period, Duty Cycle
Enable Timer Interrupts
Update Display Value
Main Loop
Delay and Refresh Display in ISR
Wait for PWM Capture Edge

Fig. 1: The Program Flowchart

Hardware Design

Two 7-segment displays were connected to Port C and Port D of the K64 microcontroller. The Arduino ground pin was connected to the K64 ground pin to establish a common reference. The Arduino PWM pin 9 output pin was then connected to the K64 PTC10 PWM input-capture pin for duty-cycle measurement.



a) b)

Fig. 2: Photos of the Setup

Schematic Diagram

This schematic included:

- MK64FN1M0VLL12 microcontroller
- Two 7 Segment displays

- ATMEGA328P-AU microcontroller (Arduino)
- Ground (GND)

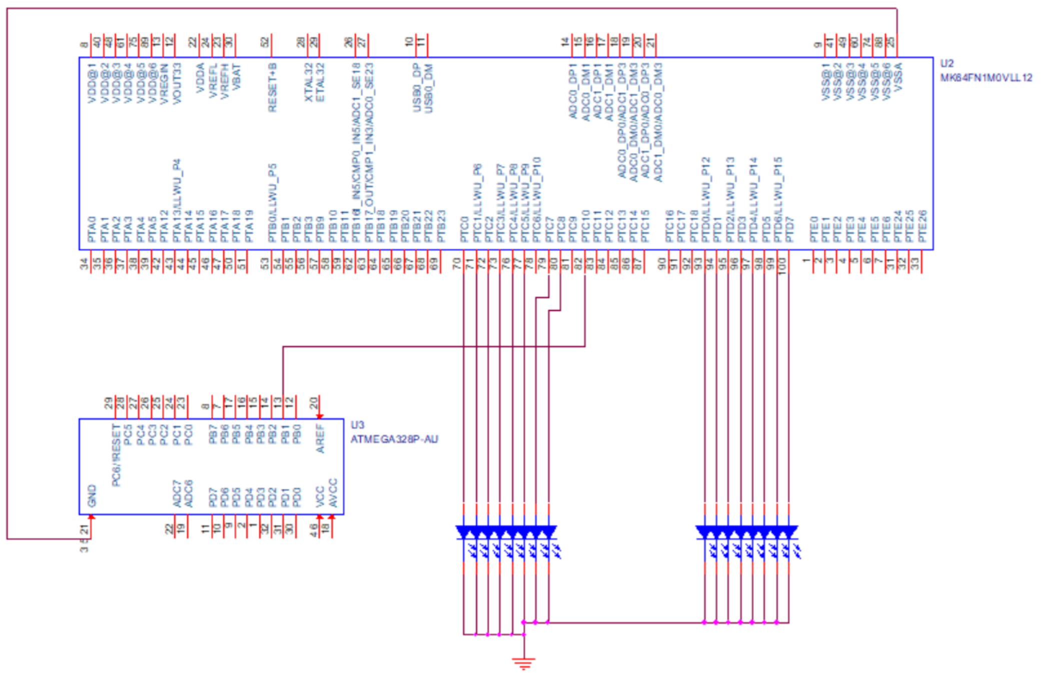


Fig. 3: Schematic Diagram

High-Level Description

The system uses two 7-segment displays, with one connected to Port C and the other connected to Port D. A PWM signal is generated externally using an Arduino board. The Arduino ground is tied to the K64F ground, and the Arduino PWM output pin is connected to the K64F input-capture pin. The K64F measures the duty cycle of the incoming PWM signal using the FlexTimer module and converts the captured timing information into a duty-cycle percentage.

Program Listing (Main Sections)

FRDM-K64F Code

```
#include "fsl_device_registers.h"

// ==========================================================================
// Global Variables
// ============================================================================
volatile int duty_display = 0;
volatile int activeDigit = 0;

unsigned char decoder[10] = {
    0x7E, 0x30, 0x6D, 0x79, 0x33,
    0x5B, 0x5F, 0x70, 0x7F, 0x7B
};

// ============================================================================
// Display Functions
// ============================================================================
void displayDigitPortD(int value) {
    GPIOD_PCOR = 0xFF;
    GPIOD_PSOR = decoder[value];
}

void displayDigitPortC(int value) {
    GPIOC_PCOR = 0x1BF;
    uint8_t pattern = decoder[value];
    GPIOC_PSOR = (pattern & 0x3F);
    if (pattern & 0x40) GPIOC_PSOR = (1 << 7);
}

// ============================================================================
// FTM0 Interrupt: Display Multiplexing
// ============================================================================
void FTM0_IRQHandler(void) {
    int tens = (duty_display / 10) % 10;
    int ones = duty_display % 10;

    if (activeDigit == 0) {
        displayDigitPortD(tens);
        activeDigit = 1;
    } else {
        displayDigitPortC(ones);
        activeDigit = 0;
    }

    FTM0_SC &= ~(1 << 7); // clear interrupt flag
}

// ============================================================================
// Display Initialization
// ============================================================================
void initDisplay(void) {
    SIM_SCGC5 |= SIM_SCGC5_PORTC_MASK | SIM_SCGC5_PORTD_MASK;

    for (int i = 0; i <= 7; i++) PORTD->PCR[i] = PORT_PCR_MUX(1);
    GPIOD_PDDR = 0xFF;
    GPIOD_PCOR = 0xFF;

    for (int i = 0; i <= 5; i++) PORTC->PCR[i] = PORT_PCR_MUX(1);
    PORTC->PCR[7] = PORT_PCR_MUX(1);
    PORTC->PCR[8] = PORT_PCR_MUX(1);
    GPIOC_PDDR |= 0x1BF;
    GPIOC_PCOR = 0x1BF;
}

// =====================================================================
// FTM0 Initialization for Display Timer
// =====================================================================
void initFTM0_DisplayTimer(void) {
    SIM_SCGC6 |= SIM_SCGC6_FTM0_MASK;
    FTM0_MODE |= FTM_MODE_WPDIS_MASK;

    FTM0_SC = 0;
    FTM0_CNTIN = 0;
    FTM0_MOD = 6000;
    FTM0_CNT = 0;
    FTM0_SC = FTM_SC_CLKS(1) | FTM_SC_PS(7) | FTM_SC_TOIE_MASK;

    NVIC_EnableIRQ(FTM0_IRQn);
}

// =====================================================================
// Main program for duty cycle measurement
// =====================================================================
int main(void) {
    initDisplay();
    initFTM0_DisplayTimer();

    SIM_SCGC3 |= SIM_SCGC3_FTM3_MASK;
    PORTC_PCR10 = PORT_PCR_MUX(3);
    FTM3_MODE = 0x05;
    FTM3_MOD = 0xFFFF;
    FTM3_SC = 0x0E;

    unsigned int r1, f1, r2, on, per, duty;

    while (1) {
        FTM3_C6SC = 0x04;
        while (!(FTM3_C6SC & 0x80)); FTM3_C6SC &= ~(1<<7);
        r1 = FTM3_C6V;

        FTM3_C6SC = 0x08;
        while (!(FTM3_C6SC & 0x80)); FTM3_C6SC &= ~(1<<7);
        f1 = FTM3_C6V;

        FTM3_C6SC = 0x04;
        while (!(FTM3_C6SC & 0x80)); FTM3_C6SC &= ~(1<<7);
        r2 = FTM3_C6V;

        on  = (f1 >= r1) ? (f1 - r1) : (0x10000 - r1 + f1);
        per = (r2 >= r1) ? (r2 - r1) : (0x10000 - r1 + r2);

        duty = (per == 0) ? 0 : (on * 100U) / per;
        if (duty > 99) duty = 99;

        if      (duty < 8)  duty = 0;
        else if (duty < 38) duty = 25;
        else if (duty < 63) duty = 50;
        else if (duty < 88) duty = 75;
        else                duty = 99;

        duty_display = duty;
    }
}
```

Arduino Code

```
// PWM Generator for Lab – ~245Hz on pin 9

const int pwmPin = 9;

int dutyValues[] = {1, 64, 127, 191, 250};
int numSteps = sizeof(dutyValues) / sizeof(dutyValues[0]);
int index = 0;

void setup() {
  pinMode(pwmPin, OUTPUT);

  // Change timer to ~245 Hz
  TCCR1B = (TCCR1B & 0b11111000) | 0x05;

  Serial.begin(9600);
  Serial.println("PWM generator running at ~245 Hz");
}

void loop() {
  int duty = dutyValues[index];
  analogWrite(pwmPin, duty);

  float dutyPercent = (duty / 255.0) * 100.0;
  Serial.print("Duty cycle ≈ ");
  Serial.print(dutyPercent, 1);
  Serial.println("%");

  index++;
  if (index >= numSteps) index = 0;

  delay(3000);
}
```

Technical Problems Encountered and Solutions

During the lab, the 7-segment display was initially showing incorrect numbers. After troubleshooting, the issue was traced to a loose jumper wire that was not fully inserted into the breadboard. Once the cable was properly connected, the display began showing the correct values.

Questions

1. When the PWM circuit modulates the LED, what is the minimum frequency you can use before you start to see the LED blink?

   The minimum frequency is anything below 60 Hz, so to have a PWM that does not blink, the PWM should be around or above 60Hz.

2. For the input capture circuit, what prescale values did you choose to use? When would you make the prescale value higher or lower?

   We used a prescale of 16. This worked well for the speed of the PWM signal coming from the Arduino. If the signal were slower, we would use a higher prescale so the timer doesn't overflow too fast. If the signal were faster, we would use a lower prescale to get better timing.

Conclusions

In the laboratory, we were able to gain experience using the FRDM-K64F microcontroller along with an external Arduino board. We correctly installed and configured the software tools, constructed the circuit connections for the 7-segment displays, and developed a program to read a PWM signal. We connected the Arduino PWM output to the K64F and measured the duty cycle using the FlexTimer input-capture feature. The measured duty-cycle values were then displayed on a two-digit 7-segment display. Through this lab, we observed the real-time behavior of the duty-cycle measurement system and verified correct operation as the Arduino changed the PWM duty cycle.

Team Contribution Summary

We worked together in most parts of the lab. Both of us helped set up the circuit and test the hardware. I (Muneer) worked on part of the K64F code, and Sameer worked on another part of the code. We also both helped with debugging and verifying the PWM signal and display output.