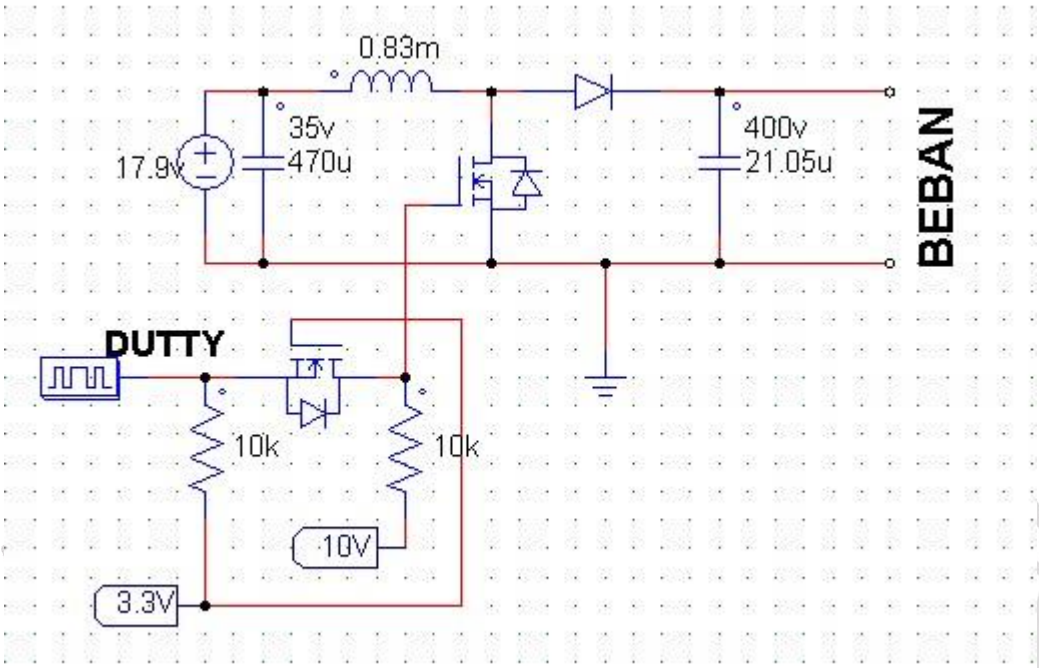
Lampiran 1 : Rangkaian Boost Converter



Gambar 4.2 Rangkaian Boost Converter

Lampiran 2 : Coding Program

```
#include "stm32f10x.h"

#include "stm32f10x_rcc.h"

#include "stm32f10x_gpio.h"

#include "stm32f10x_adc.h"

#include "delay.h"

#include "lcd16x2.h"

#include <stdio.h>

#include "stm32f10x_tim.h"
```

```
void ADC1_Init(void);
void PWM_Init(void);

uint16_t ADC1_Read(void);
void ADC2_Init(void);
uint16_t ADC2_Read(void);

float amper,volt,brignes,dutty,cycle,deltaI,deltaV,induktor,output,op;
char sAdcValue[5];
int z;
float nilai_sementara,nilai_sementara2,rata,rata1,J;

        int main(void)
{       DelayInit();
        lcd16x2_init(LCD16X2_DISPLAY_ON_CURSOR_OFF_BLINK_OFF);
        PWM_Init();
        // Initialize ADC
        ADC1_Init();
        ADC2_Init();
```

```
        while (1)
{
                DelayMs(1);
                z=0;
                nilai_sementara=0;
                nilai_sementara2=0;
                // Read ADC value
                while (z<151)
{               ADC1_Read();
                ADC2_Read();
                nilai_sementara = nilai_sementara+ADC1_Read();
                DelayMs(1);
                nilai_sementara2 = nilai_sementara2+ADC2_Read();
                z++;
        }
                rata= nilai_sementara/150;
                amper=((rata*(3/4095.000))-1.50)/0.055;
                        // Convert ADC value to string
                DelayMs(5);
                // Convert ADC value to string
                rata1= nilai_sementara2/150;
```

```
            volt=(rata1*(3/4095.000))/0.1;
            J=1.12;
            induktor=0.00083;
            output=(150/amper);
            deltaV=(150/amper)-volt;
            op=amper*volt;
            deltaI=(((1+((deltaV/volt)*J))*output)-volt)/induktor;
            cycle=(((volt+(induktor*deltaI))/output)-1);
            if(volt<18.5||amper<0.4||op>150){cycle=0;
    }       TIM2->CCR1 = (((7199 + 1) * cycle) / 100) - 1 ;
            //TIM2->CCR1 = (((7199 + 1) * cycle) / 100) - 1 ;
            sprintf(sAdcValue,"%.1f volt %.2f amp %.2f dutty",
            volt,amper,cycle);
            // Display ADC value to LCD
            lcd16x2_clrscr();
            lcd16x2_puts(sAdcValue);
            DelayMs(150);
    }
    }
            void ADC1_Init()
{
```

```
// Initialization struct
ADC_InitTypeDef ADC_InitStruct;
GPIO_InitTypeDef GPIO_InitStruct;

// Step 1: Initialize ADC1
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
ADC_InitStruct.ADC_ContinuousConvMode = DISABLE;
ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStruct.ADC_ExternalTrigConv = DISABLE;
ADC_InitStruct.ADC_ExternalTrigConv =
ADC_ExternalTrigConv_None;
ADC_InitStruct.ADC_Mode = ADC_Mode_Independent;
ADC_InitStruct.ADC_NbrOfChannel = 1;
ADC_InitStruct.ADC_ScanConvMode = DISABLE;
ADC_Init(ADC1, &ADC_InitStruct);
ADC_Cmd(ADC1, ENABLE);
// Select input channel for ADC1
// ADC1 channel 1 (PA1)
        ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1,
ADC_SampleTime_55Cycles5);
```

```
    // Step 2: Initialize GPIOA (PA1)
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_1;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOA, &GPIO_InitStruct);
}
    void ADC2_Init()
{
    // Initialization struct
    ADC_InitTypeDef ADC_InitStruct;
    GPIO_InitTypeDef GPIO_InitStruct;

    // Step 1: Initialize ADC1
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC2, ENABLE);
    ADC_InitStruct.ADC_ContinuousConvMode = DISABLE;
    ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStruct.ADC_ExternalTrigConv = DISABLE;
    ADC_InitStruct.ADC_ExternalTrigConv =
    ADC_ExternalTrigConv_None;
    ADC_InitStruct.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStruct.ADC_NbrOfChannel = 1;
```

```
    ADC_InitStruct.ADC_ScanConvMode = DISABLE;
    ADC_Init(ADC2, &ADC_InitStruct);
    ADC_Cmd(ADC2, ENABLE);
    // Select input channel for ADC1
    // ADC1 channel 2 (PA0)
    ADC_RegularChannelConfig(ADC2, ADC_Channel_2, 1,
    ADC_SampleTime_55Cycles5);

    // Step 2: Initialize GPIOA (PA0)
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOA, &GPIO_InitStruct);
}
    void PWM_Init()
{
    // Initialization struct
    TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStruct;
    TIM_OCInitTypeDef TIM_OCInitStruct;
    GPIO_InitTypeDef GPIO_InitStruct;
```

```
// Step 1: Initialize TIM2
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
// Create 1kHz PWM
// TIM2 is connected to APB1 bus that have default clock 72MHz
// So, the frequency of TIM2 is 72MHz
// We use prescaler 10 here
// So, the frequency of TIM2 now is 72MHz
TIM_TimeBaseInitStruct.TIM_Prescaler = 10;
// TIM_Period determine the PWM frequency by this equation:
// PWM_frequency = timer_clock / (TIM_Period + 1)
// If we want 1kHz PWM we can calculate:
// TIM_Period = (timer_clock / PWM_frequency) - 1
// TIM_Period = (7.2MHz / 1kHz) - 1 = 7199
TIM_TimeBaseInitStruct.TIM_Period = 7199;
TIM_TimeBaseInitStruct.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBaseInitStruct.TIM_CounterMode =
TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseInitStruct);
// Start TIM2
TIM_Cmd(TIM2, ENABLE);
```

```
// Step 2: Initialize PWM
// Common PWM settings
TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_High;
// Duty cycle calculation equation:
// TIM_Pulse = (((TIM_Period + 1) * duty_cycle) / 100) - 1
// Ex. 25% duty cycle:
// TIM_Pulse = (((7199 + 1) * 25) / 100) - 1 = 1799
//TIM_Pulse = (((7199 + 1) * 75) / 100) - 1 = 5399
// We initialize PWM value with duty cycle of 0%
TIM_OCInitStruct.TIM_Pulse = 0;
TIM_OC1Init(TIM2, &TIM_OCInitStruct);
TIM_OC1PreloadConfig(TIM2, TIM_OCPreload_Enable);

// Step 3: Initialize GPIOA (PA0)
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
// Initialize PA0 as push-pull alternate function (PWM output) for
LED
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF_PP;
```

```
        GPIO_InitStruct.GPIO_Speed = GPIO_Speed_2MHz;
        GPIO_Init(GPIOA, &GPIO_InitStruct);
    }

        uint16_t ADC1_Read()
{
        // Start ADC conversion
        ADC_SoftwareStartConvCmd(ADC1, ENABLE);
        // Wait until ADC conversion finished
        while (!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC));
        return ADC_GetConversionValue(ADC1);
    }
        uint16_t ADC2_Read()
{
        // Start ADC conversion
        ADC_SoftwareStartConvCmd(ADC2, ENABLE);
        // Wait until ADC conversion finished
        while (!ADC_GetFlagStatus(ADC2, ADC_FLAG_EOC));
        return ADC_GetConversionValue(ADC2);
    }
```