Hi các bác em là newbie đang viết firmware cho VFD để làm đồ án. Em đang có thắc mắc về cách thay đổi tần số sóng sine đầu ra, qua tìm hiểu em tổng hợp được vài cách như sau.

* Cách 1: Tạo một bảng giá trị tham chiếu hình Sine cố định các phần tử ex: 1000 phần tử. Giữ nguyên tần số PWM, Dùng một timer riêng tạo thời gian ngắt để cập nhật giá trị so sánh. Ví dụ 20ns nhảy vào bảng lấy giá trị 1 lần khi lấy hết 1000 giá trị ta được Fsineout = 50hz; ... 200ns nhảy vào bảng lấy giá trị 1 lần ta được fsineout = 5hz
* Cách 2: GIữ nguyên tần số PWM. Thay đổi giá trị phần tử trong bảng tra hình Sine. Ví dụ ở tần số PWM 8000Hz. Ở Fsineout 5Hz cần 8000/ 5 = 1600 phần tử bảng Sine, Fsineout 20HZ cần 8000/20 = 400 phần tử bảng Sine.. Fsineout 50HZ cần 160 phần tử bảng Sine
* Cách 3 : Giữ nguyên số phần tử bảng thay đổi tần số PWM ở các mức Fsineout.

Theo các bác cách nào là tối ưu nhất trên vi điều khiển nhỉ

Anonymous participant sine thì dễ , quan trọng hồi tiếp về mới khó

Cả 3 cách này của e đều ko thấy có feedback, chạy openloop như vậy chỉ test được no Load vs Light Load thôi, medium load hay heavy load thì dễ oẳng lắm. Nếu làm đồ án chơi chơi thì ok, còn làm có ý nghĩa thì e tìm hiểu về FOC, SVPWM

Anonymous participant 3 cách của e đều đang hướng đến tạo ra sPWM. Đối với thằng sPwm này thì e nên dùng cách 1. em nên kiểm tra tín hiệu điều khiển out ra từ mcu để đánh giá về độ lệch pha cũng như duty đã chuẩn chưa. Chuẩn rồi thì cắm motor vào test

Khuyến khích làm cách 1 em nhé giữ cố định bảng sine table và up tần số bằng 1 time

SPWM 5000Hz thì có vẻ hơi thấp đẩy cao tầm 6-8Khz đi. Với anh từ 32-40 phần tử là ok rồi .anh đang làm inveter 1 phase có pid phản hồi với SPWM = 19500Hz số điểm sine là 32 phần tử Fout=50/60Hz sine đẹp long lanh khi qua LC rồi

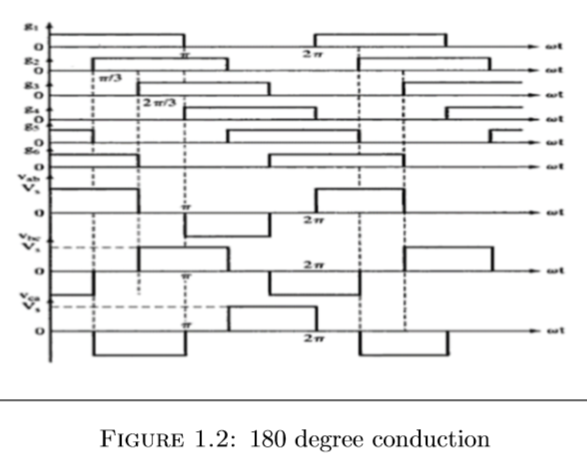
**Gate Driver Circuit for Three Phase Inverter**

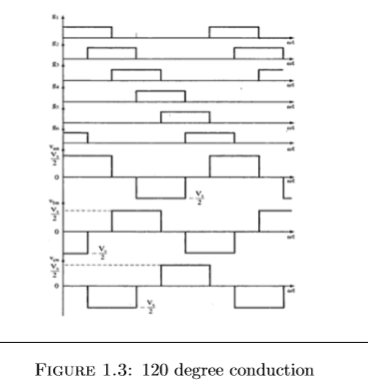
This project is basically a Driver Circuit For an Equipment called SemiTeach which we recently bought for our department. The image of the device is shown.

Connecting this driver circuit to 6 mosfets generates three 120 degree shifted Ac voltages. Range is 600 V for the SemiTeach device. Device also has in built error output terminals that give low state when detected error on any of the three phases

Inverters are commonly used in the Power Industry to convert the DC Voltage of many generation Sources into AC Voltages for eﬃcient transmission and distribution. Fur thermore, they are also used to extract energy from Uninterruptable Power Series (UPS). Inverters need a Gate Driver Circuit to drive the Power Electronics switches used in the circuit for the conversion. There are many types of Gate Signals that can be imple mented. The following report discusses the design and implementation of a Gate Driver Circuit for a Three Phase Inverter using 180 degree Conduction. This report focuses on the design of the Gate Driver Circuit in which the complete design details are written. Furthermore, this project also encapsulates the protection of the microcontroller and the circuit during the error conditions. The output of the circuit is 6 PWMs for 3 legs of the Three Phase Inverter.

## Step 1: Literature Review

[](https://content.instructables.com/FKX/T1B8/JHKTL9PX/FKXT1B8JHKTL9PX.png?auto=webp&frame=1&fit=bounds&md=b831edfbbe126ef2398dea4147ac1d2b)



Many applications in the Power Industry require the conversion of DC Voltage to AC voltage such as the connection of Solar Panels to the National Grid or to power AC devices. This conversion of DC to AC is achieved using Inverters. Based on the type of supply, there are two types of inverters: Single Phase Inverter and Three Phase Inverter. A Single Phase Inverter takes the DC voltage as input and converts it to Single Phase AC Voltage while a three Phase Inverter converter converts the DC Voltage into Three Phase AC Voltage.

Figure 1.1: Three Phase Inverter

A three phase inverter employs 6 transistor switches as shown above which are driven by PWM Signals using Gate Driver Circuits.

The Gating Signals of the inverter should have a phase diﬀerence of 120 degrees with respect to each other to acquire a three-phase balanced output. Two types of Control Signals can be applied to run this circuit

• 180 degree conduction

• 120 degree conduction

**180 degree Conduction Mode**

In this mode, each transistor is switched on for 180 degrees. And at any time, three transistors remain switched on, one transistor in each branch. In one cycle, there are six modes of operation and each mode operates for 60 degrees of the cycle. The gating signals are shifted from each other by a phase diﬀerence of 60 degrees to obtain threephase balanced supply.

Figure 1.2: 180 degree conductio

**120 degree Conduction Mode**

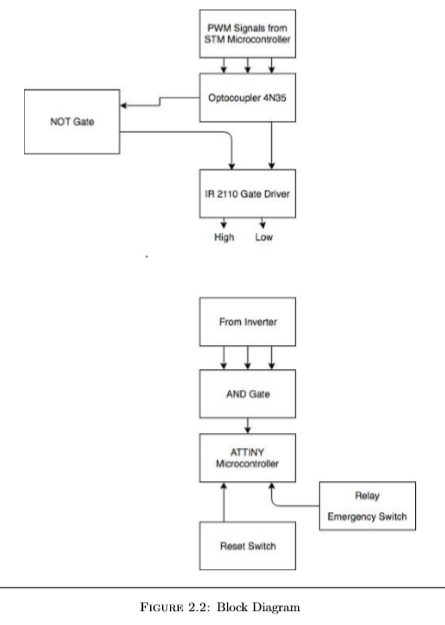
In this mode, each transistor is switched on for 120 degrees. And at any time, only two transistors conduct. It should be noted that at any time, in each branch, only one transistor should be on. There should be a phase diﬀerence of 60 degrees between the PWM Signals to get balanced three phase AC output.

Figure 1.3: 120 degree conduction

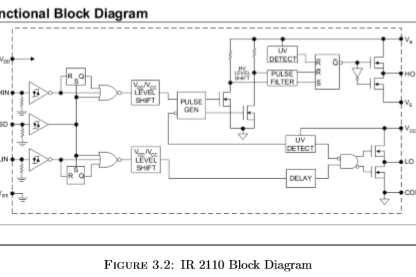
**Dead Time Control**

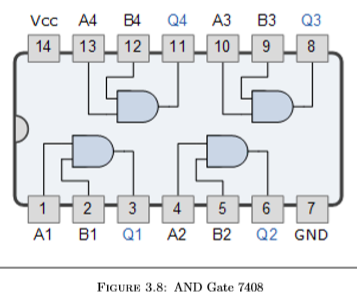
One very important precaution to be taken is that in one leg, both the transistors should not be on at the same time otherwise the DC Source will short circuit and the circuit is damaged. Therefore, it is very essential to add a very short time interval between the turning oﬀ of one transistor and the turning on of the other transistor.

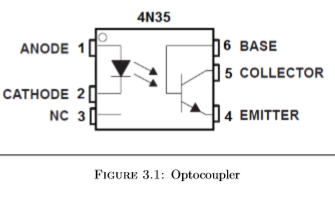
## Step 2: Block Diagram

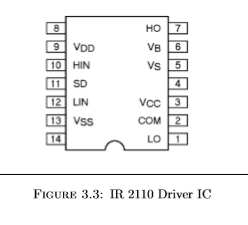
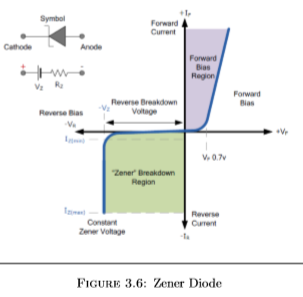
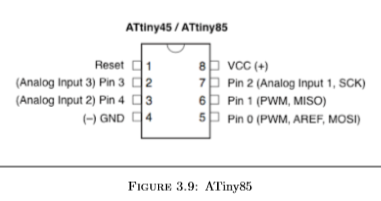
[](https://content.instructables.com/FRA/UD9X/JHKTL93M/FRAUD9XJHKTL93M.png?auto=webp&frame=1&fit=bounds&md=bf624a3c9ce5858639e4de372cf8dfb3)

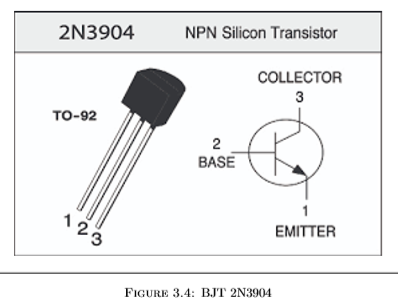
## Step 3: Components

[](https://content.instructables.com/F6Z/MXGP/JHKTL95W/F6ZMXGPJHKTL95W.png?auto=webp&frame=1&fit=bounds&md=14ed17849716e0942abe900d7498230b)







In this section details about design will be presented and will be analyzed.

**Component List**

• Optocoupler 4n35

• IR2110 driver IC

• Transistor 2N3904

• Diode (UF4007)

• Zener Diodes

• Relay 5V

•AND Gate 7408

• ATiny85

> **Optocoupler**

4n35 optocoupler has been used for optical isolation of the microcontroller from the rest of the circuit. The resistance selected is based on the formula:

Resistance = LedVoltage/CurrentRating

Resistance = 1.35V/13.5mA

Resistance = 100ohms

Output resistance acting as pull down resistance is 10k ohm for proper voltage development across it.

> **IR 2110**

It is a gate driving IC typically used for driving the MOSFETs. It is a 500 V High and Low Side Driver IC with typical 2.5 A source and 2.5 A sink currents in 14 Lead Packaging IC.

> **Bootstrap Capacitor**

The most important component of driver IC is the bootstrap capacitor. The bootstrap capacitor must be able to supply this charge, and retain its full voltage, otherwise there will be a signiﬁcant amount of ripple on the Vbs voltage, which could fall below the Vbsuv undervoltage lockout, and cause the HO output to stop functioning. Therefore the charge in the Cbs capacitor must be a minimum of twice the above value. The minimum capacitor value can be calculated from the equation below.

C = 2[(2Qg + Iqbs/f + Qls + Icbs(leak)/f) / (Vcc−Vf −Vls−Vmin)]

Where as

Vf= Forward voltage drop across the bootstrap diode

VLS= Voltage drop across the low side FET (or load for a high side driver)

VMin= Minimum voltage between VB and VS

Qg= Gate charge of high side FET

F= Frequency of operation

Icbs(leak) = Bootstrap capacitor leakage current

Qls = level shift charge required per cycle

We have selected a value of 47uF.

> **Transistor 2N3904**

The 2N3904 is a common NPN bipolar junction transistor used for general-purpose low-power amplifying or switching applications. It can handle 200 mA current(absolute maximum) and frequencies as high as 100 MHz when used as an ampliﬁer.

> **Diode (UF4007)**

A high resistivity I-type semiconductor is utilized to provide signiﬁcantly lower diode capacitance (Ct). As a result, PIN diodes act as a variable resistor with forward bias, and behave as a capacitor with reverse bias. High frequency characteristics (low capacitance ensures minimal eﬀect of signal lines) make them suitable for use as variable resistor elements in a wide variety of applications, including attenuators, high-frequency signal switching (i.e. mobile phones requiring an antenna), and AGC circuits.

> **Zener Diode**

A Zener diode is a particular type of diode that, unlike a normal one, allows current to ﬂow not only from its anode to its cathode, but also in the reverse direction, when the Zener voltage is reached. It is used as a voltage regulator. Zener diodes have a highly doped p-n junction. Normal diodes will also break down with a reverse voltage but the voltage and sharpness of the knee are not as well deﬁned as for a Zener diode. Also normal diodes are not designed to operate in the breakdown region, but Zener diodes can reliably operate in this region.

> **Relay**

Relays are switches that open and close circuits electromechanically or electronically. Relays control one electrical circuit by opening and closing contacts in another circuit. When a relay contact is normally open (NO), there is an open contact when the relay is not energized. When a relay contact is Normally Closed (NC), there is a closed contact when the relay is not energized. In either case, applying electrical current to the contacts will change their state

> **AND GATE 7408**

A Logic AND Gate is a type of digital logic gate whose output goes HIGH to a logic level 1 when all of its inputs are HIGH

> **ATiny85**

It is a low-power Microchip 8-bit AVR RISC-based microcontroller combining 8KB ISP ﬂash memory, 512B EEPROM, 512-Byte SRAM, 6 general purpose I/O lines, 32 general purpose working registers, one 8-bit timer/counter with compare modes, one 8-bit high speed timer/counter, USI, internal and external Interrupts, 4-channel 10-bit A/D converter.

## Step 4: Working and Circuit Explained

In this section the working of the circuit will be explained in detail.

**PWM generation**

PWM has been generated from STM microcontroller. TIM3, TIM4 and TIM5 has been used to generate three PWMs of 50 percent duty cycle. The phase shift of 60 degrees was incorporated between three PWMs using time delay. For 50 Hz PWM signal, following method was used to calculate the delay

delay = TimePeriod∗60/360

delay = 20ms∗60/360

delay = 3.3ms

**Microcontroller Isolation using Optocoupler**

Isolation between microcontroller and the rest of the circuit has been done using optocoupler 4n35. The isolation voltage of 4n35 is about 5000 V. It is used for the protection of the microcontroller from the reverse currents. As a microcontroller can’t bear negative voltage,hence, for the protection of the microcontroller, optocoupler is used.

**Gate Driving Circuit**  
IR2110 driver IC has been used to provide switching PWMs to the MOSFETs. PWMs from the microcontroller have been provided at the input of the IC. As IR2110 don’t have the built in NOT Gate therefore BJT is used as an inverter to the pin Lin. It then gives the complementary PWMs to the MOSFETs which are to be driven

**Error Detection**

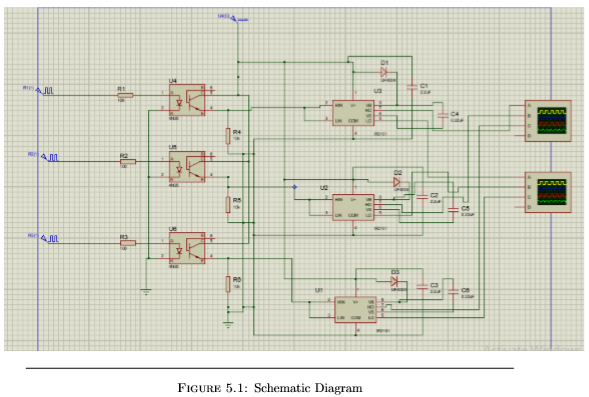
The SemiTeach module has 3 error pins which are normally HIGH at 15 V. Whenever there is any error in the circuit, one of the pins go to level LOW. For the protection of the components of the circuit, the circuit must be cut oﬀ during error conditions. This was accomplished using AND Gate, ATiny85 Microcontroller and a 5 V Relay. **Use of AND Gate**

The input to the AND Gate are 3 error pins which are at HIGH state in normal condition so the output of AND Gate is HIGH in normal conditions. As soon as there is an error, one the pins go to 0 V and hence the output of the AND Gate goes LOW. This can be used to check whether there is an error or not in the circuit. The Vcc to the AND Gate is provided through a Zener Diode.

**Cutting the Vcc through ATiny85**

The output of the AND Gate is fed to the ATiny85 Microcontroller which generates an interrupt as soon as there is any error. This further drives the Relay which cuts oﬀ the Vcc of all the components except ATiny85.

## Step 5: Simulation

[](https://content.instructables.com/FQK/B58B/JHKTL9RW/FQKB58BJHKTL9RW.png?auto=webp&frame=1&fit=bounds&md=64b48fe88ce6ff0ec9dae05fefebf908)

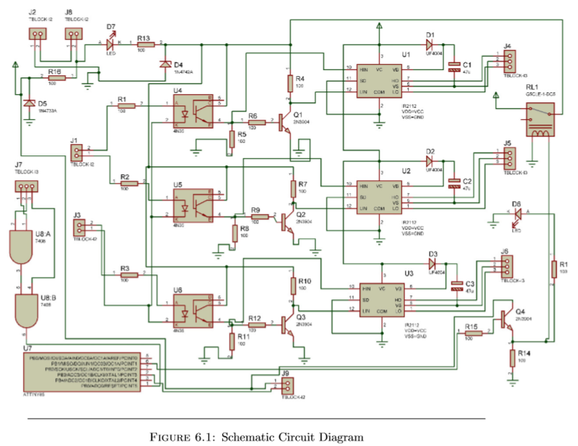
For the simulation, we have used the PWMs from the function generator in the Proteus rather than STMf401 model as it is not available on Proteus. We have used Opto-Coupler 4n35 for the isolation between micro-controller and the rest of the circuit. IR2103 is used in the simulations as a current ampliﬁer which gives us complementary PWMs.

**Schematic Diagram**  
The schematic diagram is given as follow:

**High Side Output**  
This output is between HO and Vs. Following ﬁgure shows the output of the three high side PWMs.

**Low Side Output**  
This output is between LO and COM. Following ﬁgure shows the output of the three high side PWMs.

## Step 6: Schematic and PCB Layout

[](https://content.instructables.com/FVS/4JWQ/JHKTL9TM/FVS4JWQJHKTL9TM.png?auto=webp&frame=1&fit=bounds&md=4667b14e81af770019c45e96120bf153)

**Complementary PWMs**

The following ﬁgure shows the output of one of the IR2110 which is complementary

**PWM of Phase A and B**

The phase A and B of are 60 degrees phase shifted. It is shown in the ﬁgure

**PWM of Phase A and C**

The phase A and C of are -60 degrees phase shifted. It is shown in the ﬁgure

## Step 8: Coding

Code was developed in Atollic TrueStudio. To install Atollic you can view my previous tutorials or download online.

The complete project has been added.

#include "stm32f4xx.h"

int i=0;

int c=0;

double freq = 50;

double PrescalerValue = 30;

double value = 0;

void timer4config(void);

void timer3config(void);

void timer5config(void);

int main(void)

{

// PC0

// PD12

// PA0

value = (uint16\_t) (SystemCoreClock/freq/(PrescalerValue+1))-1;;

timer3config();

for (i=0; i<10700; i++) {

c++;

}

timer4config();

c =0 ;

for (i=0; i<10700; i++) {

c++;

}

timer5config();

}

void timer3config(void)

{

// Structures

TIM\_TimeBaseInitTypeDef TIM\_TimeBaseStructure;

TIM\_OCInitTypeDef TIM\_OCInitStructure;

GPIO\_InitTypeDef GPIO\_InitStructure;

// Enable TIM3 Clock

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM3, ENABLE);

// Enable GPIOC Pin

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOC, ENABLE);

// Initialise pins 6 C

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_6;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AF;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_100MHz;

GPIO\_InitStructure.GPIO\_OType = GPIO\_OType\_PP;

GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_UP ;

GPIO\_Init(GPIOC, &GPIO\_InitStructure);

GPIO\_PinAFConfig(GPIOC, GPIO\_PinSource6, GPIO\_AF\_TIM3);

// Setup timer defaults

TIM\_TimeBaseStructure.TIM\_Period = value;

TIM\_TimeBaseStructure.TIM\_Prescaler = PrescalerValue;

TIM\_TimeBaseStructure.TIM\_ClockDivision = 0;

TIM\_TimeBaseStructure.TIM\_CounterMode = TIM\_CounterMode\_Up;

TIM\_TimeBaseInit(TIM3, &TIM\_TimeBaseStructure);

TIM\_OCInitStructure.TIM\_OCMode = TIM\_OCMode\_PWM1;

TIM\_OCInitStructure.TIM\_OCPolarity = TIM\_OCPolarity\_High;

// Configure timer for PWM - channel 1

TIM\_OCInitStructure.TIM\_OutputState = TIM\_OutputState\_Enable;

TIM\_OCInitStructure.TIM\_Pulse = value/2;

TIM\_OC1Init(TIM3, &TIM\_OCInitStructure);

TIM\_OC1PreloadConfig(TIM3, TIM\_OCPreload\_Enable);

TIM\_ARRPreloadConfig(TIM3, ENABLE);

// Start timer

TIM\_Cmd(TIM3, ENABLE);

}

void timer4config(void)

{

// Structures

TIM\_TimeBaseInitTypeDef TIM\_TimeBaseStructure;

TIM\_OCInitTypeDef TIM\_OCInitStructure;

GPIO\_InitTypeDef GPIO\_InitStructure;

// Enable TIM4 Clock

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM4, ENABLE);

// Enable GPIOD Pin

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOD, ENABLE);

// Initialise pins 12D

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_12;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AF;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_100MHz;

GPIO\_InitStructure.GPIO\_OType = GPIO\_OType\_PP;

GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_UP ;

GPIO\_Init(GPIOD, &GPIO\_InitStructure);

GPIO\_PinAFConfig(GPIOD, GPIO\_PinSource12, GPIO\_AF\_TIM4);

// Setup timer defaults

TIM\_TimeBaseStructure.TIM\_Period = value;

TIM\_TimeBaseStructure.TIM\_Prescaler = PrescalerValue;

TIM\_TimeBaseStructure.TIM\_ClockDivision = 0;

TIM\_TimeBaseStructure.TIM\_CounterMode = TIM\_CounterMode\_Up;

TIM\_TimeBaseInit(TIM4, &TIM\_TimeBaseStructure);

TIM\_OCInitStructure.TIM\_OCMode = TIM\_OCMode\_PWM1;

TIM\_OCInitStructure.TIM\_OCPolarity = TIM\_OCPolarity\_High;

// Configure timer for PWM - channel 1

TIM\_OCInitStructure.TIM\_OutputState = TIM\_OutputState\_Enable;

TIM\_OCInitStructure.TIM\_Pulse = value/2;

TIM\_OC1Init(TIM4, &TIM\_OCInitStructure);

TIM\_OC1PreloadConfig(TIM4, TIM\_OCPreload\_Enable);

TIM\_ARRPreloadConfig(TIM4, ENABLE);

// Start timer

TIM\_Cmd(TIM4, ENABLE);

}

void timer5config(void)

{

// Structures

TIM\_TimeBaseInitTypeDef TIM\_TimeBaseStructure;

TIM\_OCInitTypeDef TIM\_OCInitStructure;

GPIO\_InitTypeDef GPIO\_InitStructure;

// Enable TIM4 Clock

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM5, ENABLE);

// Enable GPIOD Pin

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOA, ENABLE);

// Initialise pins 12D

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_0;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AF;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_100MHz;

GPIO\_InitStructure.GPIO\_OType = GPIO\_OType\_PP;

GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_UP ;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

GPIO\_PinAFConfig(GPIOA, GPIO\_PinSource0, GPIO\_AF\_TIM5);

// Setup timer defaults

TIM\_TimeBaseStructure.TIM\_Period = value;

TIM\_TimeBaseStructure.TIM\_Prescaler = PrescalerValue;

TIM\_TimeBaseStructure.TIM\_ClockDivision = 0;

TIM\_TimeBaseStructure.TIM\_CounterMode = TIM\_CounterMode\_Up;

TIM\_TimeBaseInit(TIM5, &TIM\_TimeBaseStructure);

TIM\_OCInitStructure.TIM\_OCMode = TIM\_OCMode\_PWM1;

TIM\_OCInitStructure.TIM\_OCPolarity = TIM\_OCPolarity\_High;

// Configure timer for PWM - channel 1

TIM\_OCInitStructure.TIM\_OutputState = TIM\_OutputState\_Enable;

TIM\_OCInitStructure.TIM\_Pulse = value/2;

TIM\_OC1Init(TIM5, &TIM\_OCInitStructure);

TIM\_OC1PreloadConfig(TIM5, TIM\_OCPreload\_Enable);

TIM\_ARRPreloadConfig(TIM5, ENABLE);

// Start timer

TIM\_Cmd(TIM5, ENABLE);

}