# EMBEEDDED SISTEM

Indrazno Siradjuddin

# TABLE OF CONTENTS

# ACKNOWLEDGMENTS

The authors would like to thank Christina Jarron for her invaluable contribution to proofreading this book and for her useful suggestions. Special thanks also to Rob Ash for helping us make the cover of the book distinctive with his original artwork. A massive thank you goes to Keith Knowles for his time and effort in reviewing and editing the final draft of this book. Finally, the authors would like to thank all the people who have provided feedback and suggestions.

# PURPOSE OF THIS BOOK

The purpose of this book is to provide students and young engineers with a guide to help them develop the skills necessary to be able to use VHDL for introductory and intermediate level digital design. These skills will also give you the ability and the confidence to continue on with VHDL-based digital design. In this way, you will also take steps toward developing the skills required to implement more advanced digital design systems. Although there are many books and on-line tutorials dealing with VHDL, these sources are often troublesome for several reasons. Firstly, much of the information regarding VHDL is either needlessly confusing or poorly written. Material with these characteristics seems to be written from the standpoint of someone who is either painfully intelligent or has forgotten that their audience may be seeing the material for the first time. Secondly, the common approach for most VHDL manuals is to introduce too many topics and a lot of extraneous information too early. Most of this material would best appear later in the presentation. Material presented in this manner has a tendency to be confusing, is easily forgotten if misunderstood or simply is never applied. The approach taken by this book is to provide only what you need to know to quickly get up and running in VHDL. As with all learning, once you have obtained and applied some useful information, it is much easier to build on what you know as opposed to continually adding information that is not directly applicable to the subjects at hand.

The intent of this book is to present topics to someone familiar with digital logic design and with some skills in algorithmic programming languages such as Java or C. The information presented here is focused on giving a solid knowledge of the approach and function of VHDL. With a logical and intelligent introduction to basic VHDL concepts, you should be able to quickly and efficiently create useful VHDL code. In this way, you will see VHDL as a valuable design, simulation and test tool rather than another batch of throw-away technical knowledge encountered in some forgotten class or lab.

Lastly, VHDL is an extremely powerful tool. The more you understand as you study and work with VHDL, the more it will enhance your learning experience independently of your particular area of interest. It is well worth noting that VHDL and other similar hardware design languages are used to create most of the digital integrated circuits found in the various electronic gizmos that overwhelm our modern lives. The concept of using software to design hardware that is controlled by software will surely provide you with endless hours of contemplation. VHDL is a very exciting language and mastering it will allow you to implement systems capable of handling and processing in parallel ns-level logic events in a comfortable software environment.

This book was written with the intention of being freely available to everybody. The formatted electronic version of this book is available from the Internet. Any part of this book can be copied, distributed and modified in accordance with the conditions of its license.

**DISCLAIMER:** This book quickly takes you down the path toward understanding VHDL and writing solid VHDL code. The ideas presented herein represent the core knowledge you will need to get up and running with VHDL. This book in no way presents a complete description of the VHDL language. In an effort to expedite the learning process, some of the finer details of VHDL have been omitted from this book. Anyone who has the time and inclination should feel free to further explore the true depth of the VHDL language. There are many on-line VHDL reference books and free tutorials. If you find yourself becoming curious about what this book is not telling you about VHDL, take a look at some of these references.

# BAB 1

# INTRODUCTION TO VHDL

## 1.1 VHDL

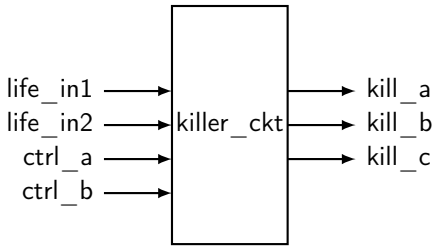
The VHDL entity construct provides a method to abstract the functionality of a circuit description to a higher level. It provides a simple wrapper for the lower-level circuitry. This wrapper effectively describes how the black box interfaces with the outside world. Since VHDL describes digital circuits, the entity simply lists the various inputs and outputs of the underlying circuitry. In VHDL terms, the black box is described by an entity declaration. The syntax of the entity declaration is shown in Listing **??**.

**Program 1.1: The entity declaration in VHDL.**

```
entity my_entity is
port(
    port_name_1 : in    std_logic ;
    port_name_2 : out   std_logic;
    port_name_3 : inout std_logic ); --do not forget the semicolon
end my_entity; -- do not forget this semicolon either
```

`my_entity` defines the name of the entity. The next section is nothing more than the list of signals from the underlying circuit that are available to the outside world, which is why it is often referred to as an interface specification. The `port_name_x` is an identifier used to differentiate the various signals. The next keyword (the keyword `in`) specifies the direction of the signal relative to the entity where signals can either enter, exit or do both. These input and output signals are associated with the keywords **in**, **out** and **inout**[1] respectively. The next keyword (the keyword `std_logic`) refers to the type of data that the port will handle. There are several data types available in VHDL but we will primarily deal with the `std_logic` type and derived versions. More information regarding the various VHDL data types will be discussed later.

[1]The `inout` data mode will be discussed later on in the book.




**Gambar 1.1:** Diagram Sederhana Killer Circuit

## 1.2 Tikz Picture

## 1.3 Arduino Code

**Program 1.2: Program OLED Minimalis**

```
#include <Wire.h>
#include <SSD1306Wire.h>

// Inisialisasi OLED: alamat I2C, SDA, SCL
SSD1306Wire display(0x3c, 18, 17);

void setup() {
    display.init();
    display.flipScreenVertically(); // T3-S3 memiliki orientasi terbalik
    display.setFont(ArialMT_Plain_10);
    display.clear();
    display.drawString(0, 0, "Halo Dunia!");
    display.drawString(0, 12, "LILYGO T3-S3");
    display.display(); // Penting: tanpa ini, layar tetap kosong
}

void loop() {
  // Tidak ada pembaruan dinamis
}
```

## 1.4 PHP Code

**Program 1.3: Endpoint PHP Minimalis**

```
<?php
echo "Halo dari Server!";
?>
```

## 1.5 Javascript

**Program 1.4: Javascript Minimalis**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0    ">
    <title>Document</title>
</head>
<body>
    <h1 id="greeting">Hello, World!</h1>
    <button onclick="changeGreeting()">Click Me</button>

    <script>
        function changeGreeting() {
            document.getElementById("greeting").innerText = "Hello, JavaScript!";
        }
    </script>
</body>
</html>
```

## 1.6 PYTHON

**Program 1.5: Python Minimalis**

```
        print("Halo, Dunia!")
```

## 1.7 ESP32

**Program 1.6: Blink example for ESP32**

```
#include <Arduino.h>

// put function declarations here:
int myFunction(int, int);

void setup() {
  // put your setup code here, to run once:
  int result = myFunction(2, 3);
}

void loop() {
  // put your main code here, to run repeatedly:
}

// put function definitions here:
int myFunction(int x, int y) {
  return x + y;
}
```

## 1.8 Fade

**Program 1.7: Fade example for ESP32**

```
// use 12 bit precision for LEDC timer
#define LEDC_TIMER_12_BIT 12

// use 5000 Hz as a LEDC base frequency
#define LEDC_BASE_FREQ 5000

// fade LED PIN (replace with LED_BUILTIN constant for built-in LED)
#define LED_PIN 4

// define starting duty, target duty and maximum fade time
#define LEDC_START_DUTY  (0)
#define LEDC_TARGET_DUTY (4095)
#define LEDC_FADE_TIME   (3000)

bool fade_ended = false;  // status of LED fade
bool fade_in = true;

void ARDUINO_ISR_ATTR LED_FADE_ISR() {
  fade_ended = true;
}

void setup() {
  // Initialize serial communication at 115200 bits per second:
  Serial.begin(115200);

  // Setup timer with given frequency, resolution and attach it to a led pin with auto-selected channel
  ledcAttach(LED_PIN, LEDC_BASE_FREQ, LEDC_TIMER_12_BIT);

  // Setup and start fade on led (duty from 0 to 4095)
  ledcFade(LED_PIN, LEDC_START_DUTY, LEDC_TARGET_DUTY, LEDC_FADE_TIME);
  Serial.println("LED Fade on started.");

  // Wait for fade to end
  delay(LEDC_FADE_TIME);

  // Setup and start fade off led and use ISR (duty from 4095 to 0)
  ledcFadeWithInterrupt(LED_PIN, LEDC_TARGET_DUTY, LEDC_START_DUTY, LEDC_FADE_TIME, LED_FADE_ISR);
  Serial.println("LED Fade off started.");
}

void loop() {
  // Check if fade_ended flag was set to true in ISR
  if (fade_ended) {
    Serial.println("LED fade ended");
    fade_ended = false;

    // Check what fade should be started next
    if (fade_in) {
      ledcFadeWithInterrupt(LED_PIN, LEDC_START_DUTY, LEDC_TARGET_DUTY, LEDC_FADE_TIME, LED_FADE_ISR);
      Serial.println("LED Fade in started.");
      fade_in = false;
    } else {
      ledcFadeWithInterrupt(LED_PIN, LEDC_TARGET_DUTY, LEDC_START_DUTY, LEDC_FADE_TIME, LED_FADE_ISR);
      Serial.println("LED Fade out started.");
      fade_in = true;
    }
  }
}
```

## 1.9 Penjelasan Program Fade LED Menggunakan LEDC pada ESP32

Program ini mengimplementasikan efek *fade in* dan *fade out* pada LED yang terhubung ke pin GPIO ESP32 menggunakan fitur LEDC (LED PWM Controller). Berikut penjelasan

bagian per bagian:

```
// use 12 bit precision for LEDC timer
#define LEDC_TIMER_12_BIT 12
```

1. Mendefinisikan resolusi timer LEDC sebesar 12 bit. Artinya, nilai *duty cycle* dapat berkisar antara 0 hingga $2^{12} - 1 = 4095$.

```
// use 5000 Hz as a LEDC base frequency
#define LEDC_BASE_FREQ 5000
```

1. Menetapkan frekuensi dasar PWM sebesar 5000 Hz. Frekuensi ini cukup tinggi sehingga manusia tidak dapat melihat kedipan LED, menghasilkan cahaya yang tampak mulus.

```
// fade LED PIN (replace with LED_BUILTIN constant for built-in LED)
#define LED_PIN 4
```

1. Menentukan pin GPIO tempat LED terhubung, dalam hal ini pin 4. Pengguna dapat menggantinya dengan `LED_BUILTIN` jika menggunakan LED bawaan papan.

```
// define starting duty, target duty and maximum fade time
#define LEDC_START_DUTY  (0)
#define LEDC_TARGET_DUTY (4095)
#define LEDC_FADE_TIME   (3000)
```

1. `LEDC_START_DUTY`: Nilai awal *duty cycle* (0 = mati).
2. `LEDC_TARGET_DUTY`: Nilai akhir *duty cycle* (4095 = nyala penuh).
3. `LEDC_FADE_TIME`: Durasi transisi *fade* dalam milidetik (3000 ms = 3 detik).

```
bool fade_ended = false;  // status of LED fade
bool fade_in = true;
```

1. `fade_ended`: Bendera untuk menandai apakah proses *fade* telah selesai, diatur oleh *interrupt service routine* (ISR).
2. `fade_in`: Bendera untuk menentukan arah *fade* berikutnya (`true` = *fade in*, `false` = *fade out*).

```
void ARDUINO_ISR_ATTR LED_FADE_ISR() {
  fade_ended = true;
}
```

1. Fungsi ISR yang dipanggil otomatis oleh perangkat keras ketika proses *fade* selesai.
2. Atribut `ARDUINO_ISR_ATTR` memastikan fungsi ini kompatibel dengan lingkungan ISR Arduino pada ESP32.
3. Fungsi ini hanya mengatur `fade_ended = true` untuk memberi tahu `loop()` bahwa transisi telah selesai.

```
void setup() {
  // Initialize serial communication at 115200 bits per second:
  Serial.begin(115200);

  // Setup timer with given frequency, resolution and attach it to a led pir
  ledcAttach(LED_PIN, LEDC_BASE_FREQ, LEDC_TIMER_12_BIT);

  // Setup and start fade on led (duty from 0 to 4095)
  ledcFade(LED_PIN, LEDC_START_DUTY, LEDC_TARGET_DUTY, LEDC_FADE_TIME);
  Serial.println("LED Fade on started.");

  // Wait for fade to end
  delay(LEDC_FADE_TIME);

  // Setup and start fade off led and use ISR (duty from 4095 to 0)
  ledcFadeWithInterrupt(LED_PIN, LEDC_TARGET_DUTY, LEDC_START_DUTY, LEDC_FAI
  Serial.println("LED Fade off started.");
}
```

1. `Serial.begin(115200)`: Menginisialisasi komunikasi serial untuk debugging.
2. `ledcAttach(...)` : Mengonfigurasi kanal LEDC secara otomatis pada `LED_PIN` dengan frekuensi dan resolusi yang ditentukan.
3. `ledcFade(...)` : Memulai transisi *fade in* tanpa ISR; program menunggu secara blok menggunakan `delay()` selama durasi *fade*.
4. Setelah *fade in* selesai, program memulai *fade out* menggunakan `ledcFadeWithInterrupt(...)`, yang memungkinkan ISR dipanggil saat selesai, sehingga eksekusi tidak perlu diblokir.

```
void loop() {
  // Check if fade_ended flag was set to true in ISR
  if (fade_ended) {
```

```
    Serial.println("LED fade ended");
    fade_ended = false;

    // Check what fade should be started next
    if (fade_in) {
      ledcFadeWithInterrupt(LED_PIN, LEDC_START_DUTY, LEDC_TARGET_DUTY, LEDC_FADE_TIME,
      LED_FADE_ISR);
      Serial.println("LED Fade in started.");
      fade_in = false;
    } else {
      ledcFadeWithInterrupt(LED_PIN, LEDC_TARGET_DUTY, LEDC_START_DUTY, LEDC_FADE_TIME, LED_FADE_ISR);
      Serial.println("LED Fade out started.");
      fade_in = true;
    }
  }
}
```

1. Di dalam `loop()`, program memeriksa apakah `fade_ended` bernilai `true`.
2. Jika ya, program mencetak pesan, mereset bendera, lalu memulai *fade* berikutnya sesuai arah yang ditentukan oleh `fade_in`.
3. Transisi selanjutnya selalu menggunakan `ledcFadeWithInterrupt`, sehingga sistem tetap responsif dan tidak terblokir oleh `delay()`.
4. Nilai `fade_in` diubah setiap kali transisi selesai, menciptakan siklus berulang *fade in* $\leftrightarrow$ *fade out*.

Secara keseluruhan, program ini menunjukkan penggunaan efisien fitur LEDC ESP32 untuk menghasilkan efek PWM halus dengan dukungan *hardware-accelerated fading* dan penanganan berbasis interupsi.