**MODUL 1 Input Output** **Analog**

**Rifki Afriadi (13223049)**

Asisten (132220)

Tanggal Percobaan: 17/10/2025

EL3112-Praktikum Sistem Mikroprosesor

Laboratorium Dasar Teknik Elektro - Sekolah Teknik Elektro dan Informatika ITB

**Abstrak**

*Dalam praktikum ini*

Kata kunci: Fungsi Transfer, DAC, ADC, atenuasi

1. **Pendahuluan**

Pengolahan informasi pada mikroprosesor berbasis data digital. Namun, di alam nyata, seringkali data dipresentasikan secara analog. Karena itu, diperlukan sistem untuk melakukan konversi data dan sinyal dari analog ke digital dan sebaliknya. Salah satu perangkat yang mampu mengoprasikan kedua sistem tersebut adalah ESP32. Pada praktikum ini, akan dilakukan percobaan untuk mengamati proses dan kecepatan konversi analog-digital, mulai dari fungsi transfer hingga kecepatan konversi. Praktikum kali ini bertujuan untuk

Mengukur fungsi transfer ADC

Mengukur kecepatan ADC

Mengukur fungsi transfer DAC

Mengukur kecepatan DAC

Mengukur terjadinya aliasing **Studi Pustaka**

2.1 Mikrokontroler

Mikrokontroler adalah sebuah komputer kecil yang dikemas dalam bentuk chip berupa IC (Integrated Circuit) dan dirancang untuk melakukan tugas atau operasi tertentu seperti menerima sinyal input, mengolahnya, kemudian memberikan sinyal output sesuai dengan program yang telah diisikan ke mikrokontroler tersebut. Mikrokontroler sering digunakan pada aplikasi sistem dan perangkat yang tidak terlalu kompleks dan tidak memerlukan komputasi tinggi.[1]

2.2 ESP32

ESP32 adalah salah satu mikrokontroler paling populer. ESP32 memiliki sekitar 39 pin GPIO serta dilengkapi wifi dan bluetooth, membuatnya sering digunakan dalam Internet Of Things (IOT). Terdapat dua CPU yang masing-masing memiliki tingkat kecepatan 80 MHz dan 160 MHz. Selain itu, ESP32 juga memiliki banyak perangkat tambahan seperti ADC, DAC, I2C, I2S, SPI, dan UART untuk berbagai macam aplikasi.Modul ini dapat diprogram dengan bahasa pemrograman C atau C++.[2]

2.3 Interupsi Eksternal

Interupsi adalah proses dalam sistem mikrokontroler yang menghentikan aliran program utama akibat terjadinya (event) trigger (pemicu) tertentu dari suatu sumber (vector) interupsi dan memaksa sistem mikrokontroler untuk mengeksekusi sub-rutin/fungsi/blok program layanan interupsi (interrupt service routine, ISR) hingga selesai (complete).[3]

2.4 Relay

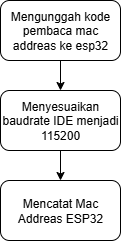
Relay merupakan sakelar elektronik yang menggunakan prinsip elektromagnetik atau teknologi solid-state untuk mencapai peralihan dan kontrol otomatis antar sirkuit. Dalam desain umum, diagram pengkabelan relai 4-pin atau diagram pengkabelan relai 5-pin digunakan untuk menghubungkan kontak-kontak seperti terminal normal terbuka (NO), terminal normal tertutup (NC), dan terminal umum (COM).[4]

2.5 Akses Register

Pada ESP32 terdapat pin GPIO (General Purpose Input Output) yang dapat dibaca oleh perangkat lunak. Membaca GPIO dapat dilakukan dengan library Arduino dengan fungsi digitalRead(), dan dapat juga dengan langsung mengakses register. Pembacaaan melalui register lebih cepat, namun sedikit lebih rumit dibandingkan menggunakan library Arduino. Berikut cara membaca Nilai GPIO pada ESP32 langsung melalui register.[5]

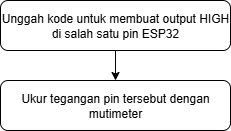
1. **Metodologi**

3.1 Identitas Modul ESP32



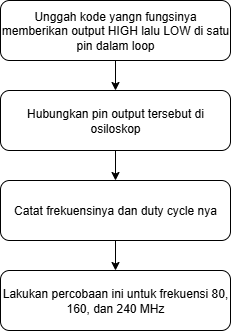
Gambar 3.1 diagram alir percobaan pembaca Mac Addreas ESP32

3.2 Membaca tegangan esp32



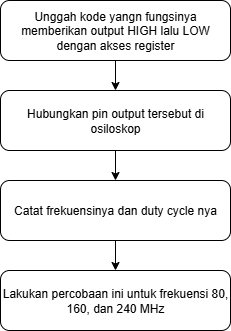
Gambar 3.2 Diagram alir percobaan membaca tegangan ESP32

3.3 Mengukur Kecepatan Output Digital Dengan Library Arduino



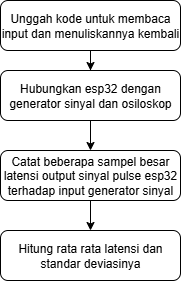
Gamba 3.3 diagram alir mengukur kecepatan output digital dengan library arduino

3.4 Mengukur Kecepatan Output Digital Dengan Akses Register



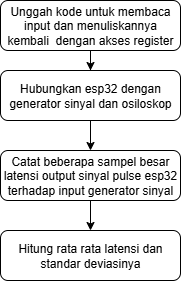
Gambar 3.4 Diagram alir mengukur output digital dengan akses register

3.5 Latency Input Output Digital Dengan Library Arduino



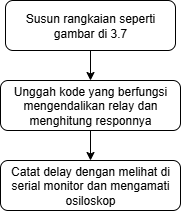
Gambar 3.5 Diagram alir Latenci Input Output Digital Dengan Library Arduino

3.6 Latency Input Output Digital Dengan Akses Register

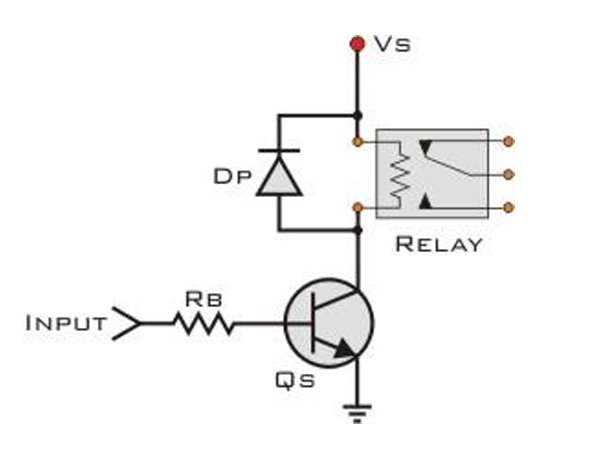


Gambar 3.6 diagram alir percobaan menghitung latensi dengan akses register

3.7 Pengendali Relay

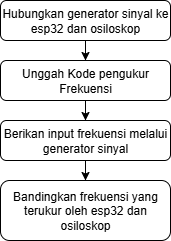


Gambar 3.8 diagram alir percobaan mengendalikan relay dan menghitung delaynya



Gambar 3.7 Rangkaian untuk percobaan pengendali relay

3.8 Pengukur Frekuensi



Gambar 3.9 Diagram alir percobaan pengukur frekuensi

1. **Hasil dan Analisis**

4.1 Identitas Modul ESP32

Kode percobaan ini diawali baris untuk meng-*import* pustaka wifi esp32. Lalu digunakan fungsi - readMacAddress() untuk membaca MAC Addreas dan ditampilkan di serial monitor. Dari percobaan diperoleh

MAC Addreas: D8:BC:38:F7:43:D4

Alamat ini berguna ketika ingin mengidentifikasi esp32 yang terhubung melalui wifi

4.2 Mengukur Tegangan VCC pada ESP32

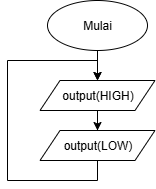
Pada praktikum ini, praktikan memberikan nilai HIGH pada pin GPIO26, lalu mengukur tegangannya dengan multimeter. Diperoleh tegangan sebesar

Tegangan Terukur : 3.27 V

Mengetahui tegangan tersebut sangat penting dlam merancang berbagai proyek IoT karena beberapa perangkat atau modul membutuhkan tegangan tertentu untuk dapat diaktifkan.

4.3 Mengukur Kecepatan Output Digital dengan Arduino Liibrary

Pada percobaan ini, digunakan kode berikut



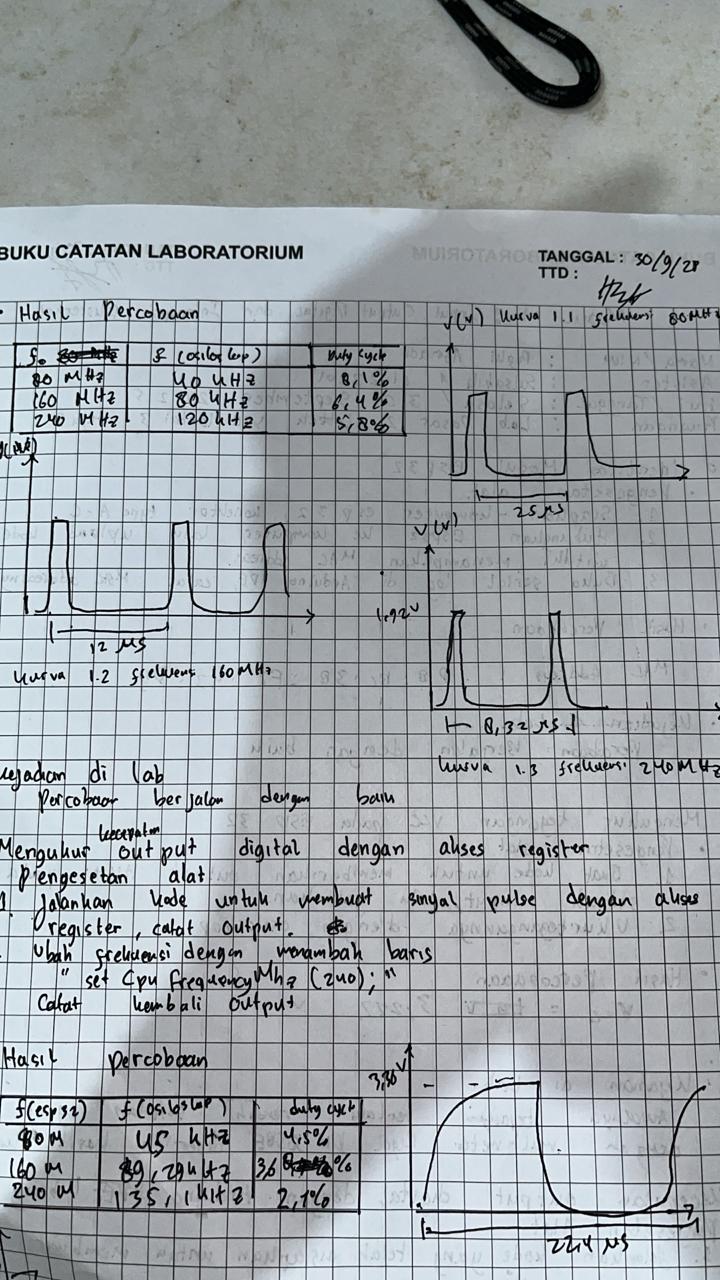
Gambar 4.1 diagram alir kode kecepatan output digital dengan library arduino

Kecepatan Beralihnya HIGH-LOW -HIGH bergantung pada krekuensi CPU arduino yang dapat diatur melalui kode *setCpuFrequensy()* ataupun melalui Arduino IDE. Dari percobaan tersebut diperoleh data berikut.

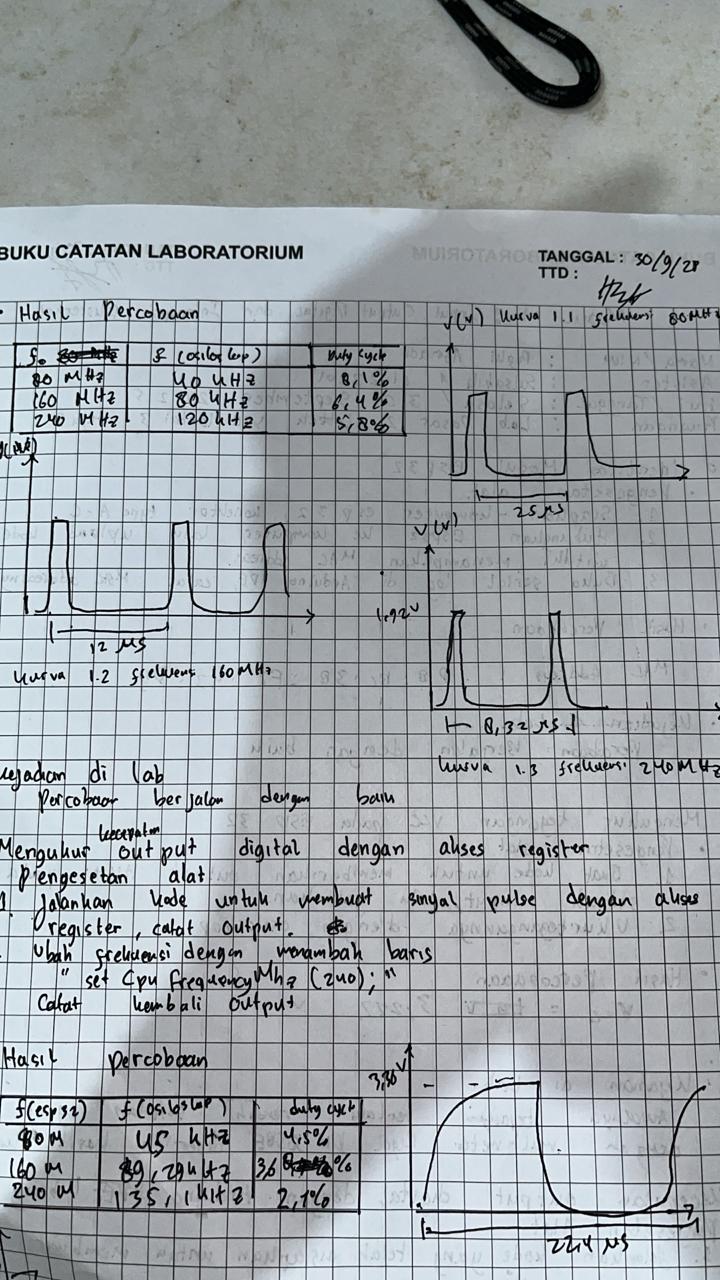
Tabel 4.1 Data frekuensi terukur di osiloskop untuk percobaan dengan library arduino

|  |  |  |
| --- | --- | --- |
| f esp32 (MHz) | f osiloskop (KHz) | Duty cycle (%) |
| 80 | 40 | 8.1 |
| 160 | 80 | 6.4 |
| 240 | 120 | 5.8 |

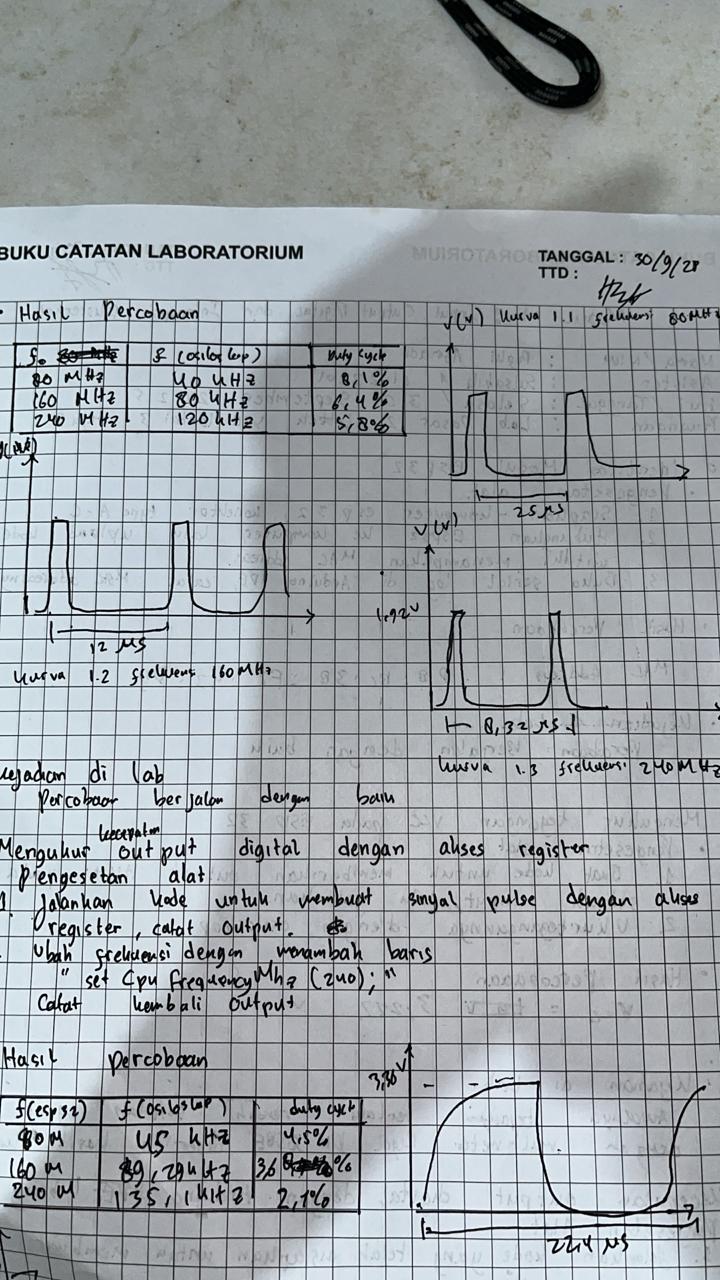
Data pengamatan sinyal dapat dilihat di kurva berikut



Kurva 4.1 frekuensi 80 MHz



Kurva 4.2 Frekuensi 160 MHz



Kurva 4.3 Frekuensi 240 MHz

Dari teabel tersebut terlihat bahwa terdapat perbedaan yang sangat jauh antara frekuensi cpu ESP32 dan frekuensi yang terekam di osiloskop. Hal ini karena fungsi digitalWrite memerlukan beberapa waktu agar bisa dieksekusi oleh esp32. Akibatnya, frekuensi yang dihasilkan menjadi sekitar 2000 kali lebih rendah.

Dari pengamatan osiloskop, kita juga melihat bahwa semakin tinggi frekuensi CPU, periode saat high menjadi semakin kecil. Hal itu juga terlihat dari data duty cycle yang beriksar 8.1%-5.8%. Hal ini terjadi karena cara deklarasi kode berikut

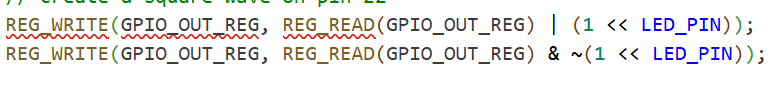


Gambar 4.4 Cuplikan kode pada output digital dengan library arduino

Terlihat bahwa setelah pin di-*set* HIGH, pin langsung di-*set* LOW. Akibatnya sinyal kotak mengalami HIGH dengan periode lebih singkat dibanding LOW yang setelahnya memperoleh delay loop.

4.4 Mengukur Kecepatan Output Digital dengan akses register

Metode dan kode yang digunakan pada percobaan ini serupa dengan percobaan sebelumnya, hanya saja kode di gambar 4.4 diganti kode berikut.



Gambar 4.5 Cuplikan kode untuk mengganti nilai register esp32

Dari percobaan yang dilakukan dengan 3 frekuensi CPU, diperoleh data berikut.

Tabel 4.2 Data frekuensi terukur di osiloskop untuk percobaan dengan library arduino

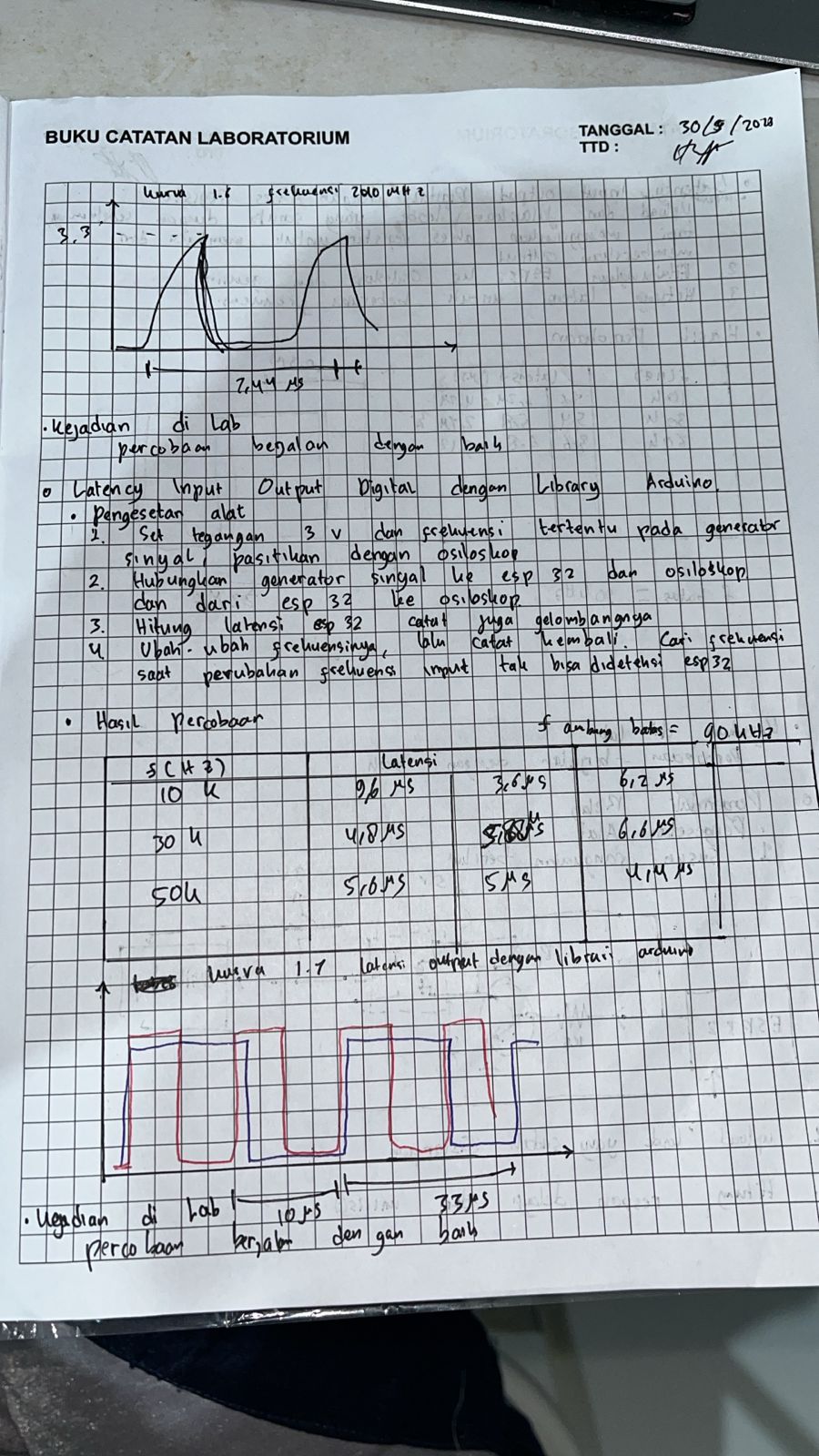
|  |  |  |
| --- | --- | --- |
| f esp32 (MHz) | f osiloskop (KHz) | Duty cycle (%) |
| 80 | 45 | 4.5 |
| 160 | 89,29 | 3.6 |
| 240 | 135,1 | 2.1 |



Kurva 4.4 Frekuensi 80 MHz



Kurva 4.5 Frekuensi 160 MHz

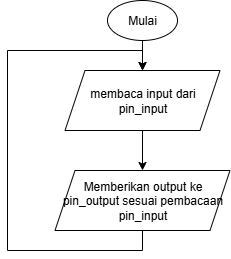


Kurva 4.6 Frekuensi 240 MHz

Dari tabel tersebut, terlihat bahwa frekuensi yang tercatat oleh osiloskop lebih tinggi dibanding sebelumnya. Namun ada konsekuensi hal tersebut, yaitu duty cycle yang lebih rendah. Sehingga, apabila sinyal diperbesar seperti yang bisa dilihat dikurva, sinyal bahkan tidak sampai berbentuk kotak.

4.5 Latensi Input Output Digital dengan library Arduino

Untuk melakukan percobaan ini, digunakan kode berikut



Gambar 4.6 Diagram Alir kode percobaan latensi dengan library Arduino

Lalu, dicatat latensi melalui pengamatan osiloskop dan diperoleh data berikut

Tabel 4.3 Data latensi dengan library Ardunio

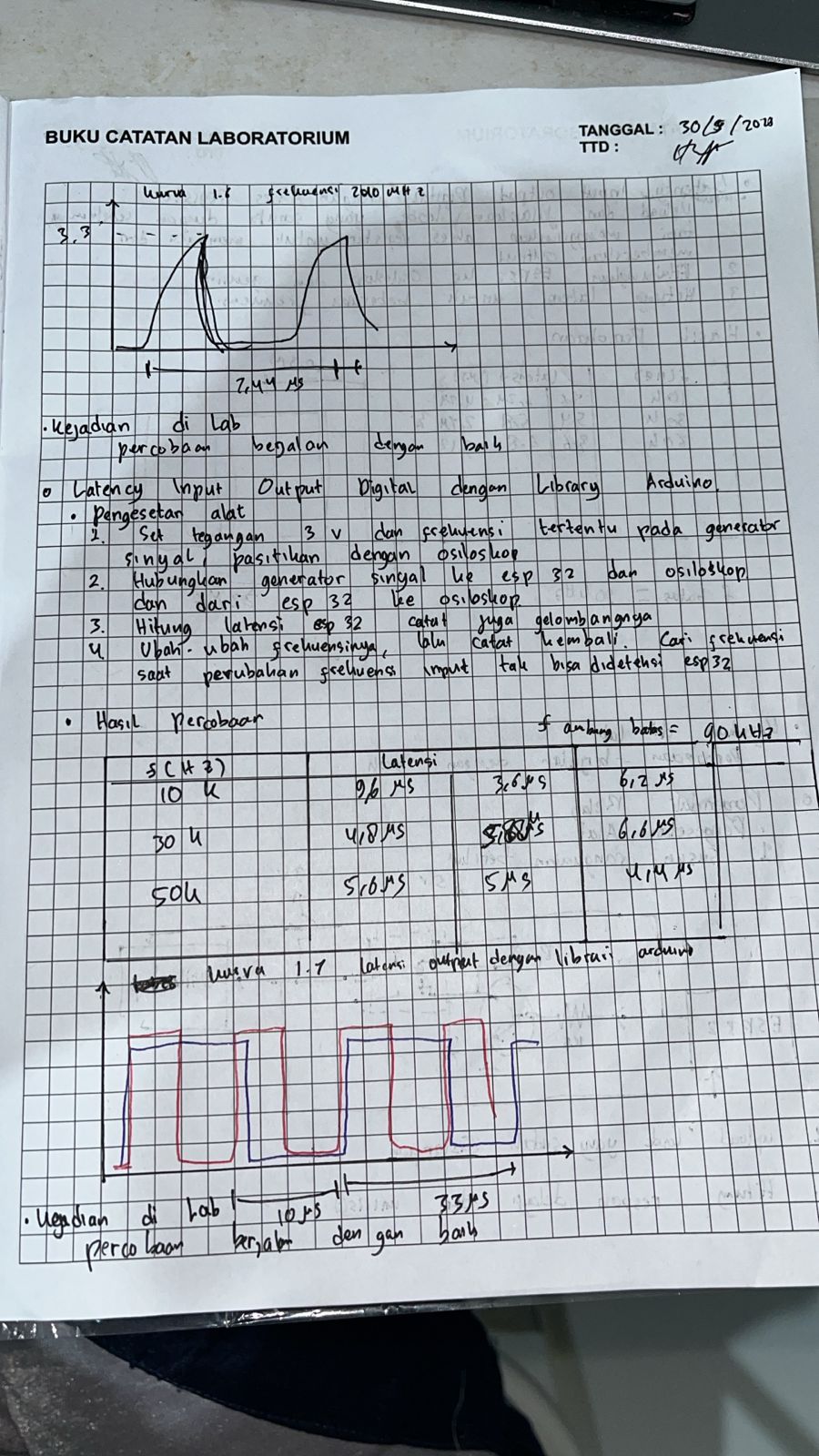
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| f (Hz) | Latensi (us) | | | Avg |
| 10k | 9.6 | 3.6 | 6.2 | 6.466666667 |
| 30k | 4.8 | 5.88 | 6.2 | 5.626666667 |
| 50k | 5.6 | 5.4 | 4.4 | 5.133333333 |
| avg | 5.742222222 | | |  |
| std | 1.685836423 | | |  |

Ketika frekuensi terus ditingkatkan, akan ada saat ketika frekuensi sinyal yang dihasilkan esp32 sudah tidak sesuai sinyal dari generator sinyal. Yaitu

f = 90 kHz

Dengan kurva yang dihasilkan sebagai berikut

Kurva 4.7 Plot sinyal input dan output ESP32

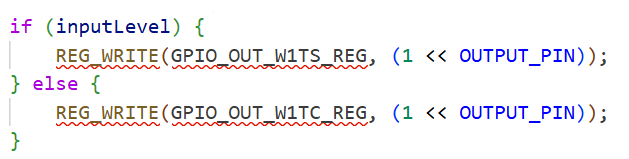


Ket :

Merah generator sinyal;biru esp32

Dari percobaan ini diperoleh rata-rata latensi sebesar 5.74 mikro sekon dengan standar deviasi yang cukup besar, yaitu 1.68 mikro sekon. Hal ini terjadi karena sinyal yang dihasilkan oleh esp32 memiliki rentang dan posisi yang tidak konstan, selalu berubah-ubah. Hal ini terjadi karena loop() esp32 tidak memiliki timing yang presisi yang menyebabkan sinyal dapat bergeser meskipun inputnya konstan. Akibatnya, ketika sinyal di osiloskop di-*pause*. Sinyal akan selalu berpindah pindah pada rentang tertentu. Bila frekuensi terus ditingkatkan, akan ada frekuensi ketika sinyal dari generator sinyal tidak dapat dibuat ulang oleh ESP32. Pada percobaan ini frekuensi tersebut adalah 90 KHz dan hasilnya, terlihat frekuensi yang dihasilkan oleh ESP32 menjadi setengah kalinya.

4.6 Latensi Input Output Digital dengan akses register

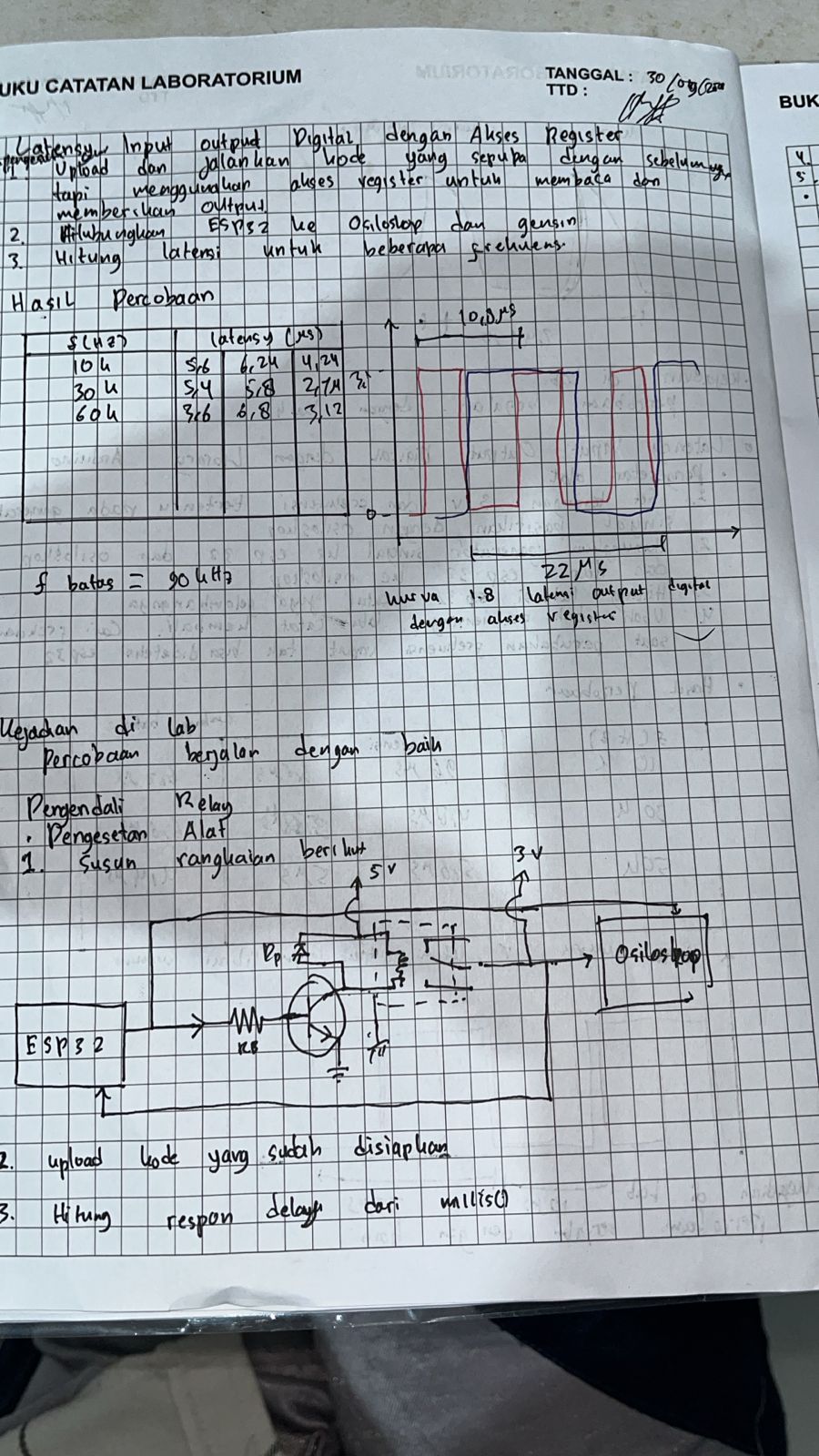
Percobaan ini dilakukan dengan kode yang hampir sama dengan percobaan sebelumnya. Hanya saja, kode perintah digitalWrite diganti dengan kode berikut  


Gambar 4.7 Cuplikan kode latensi dengan akses register

Tabel 4.4 Data latensi dengan akses register

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **f (Hz)** | **Latensi (us)** | | | **Avg** |
| 10k | 5.6 | 6.24 | 4.24 | 5.36 |
| 30k | 5.4 | 5.8 | 2.74 | 4.646666667 |
| 50k | 3.6 | 6.8 | 3.12 | 4.506666667 |
| avg | 4.837777778 | | |  |
| std | 1.452599203 | | |  |

Kurva 4.8 Plot sinyal input dan output ESP32 ketika menggunakan akses register.

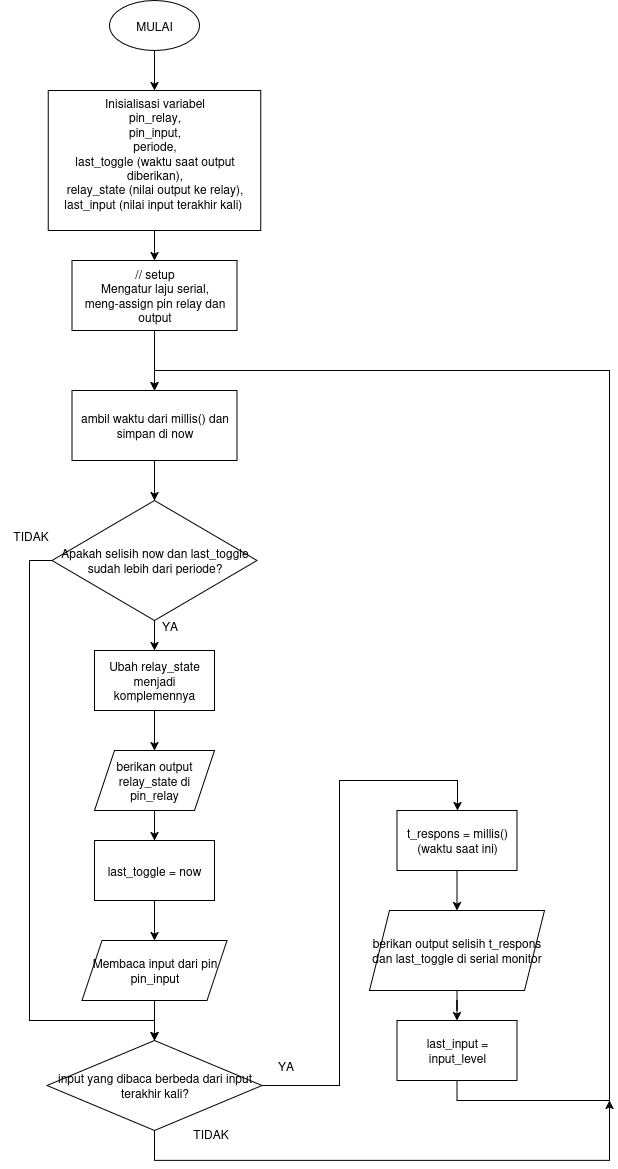


Frekuensi generator sinyal yang melewati ESP32 sama, yaitu 90 kHz.

Seperti yang disebutkan pada pernyataan…. Perintah dengan akses register akan memberikan perubahan nilai pin output yang lebih cepat sehingga latensinya lebih kecil sekitar 15%.

4.7 Pengendali Relay

Program yang digunakan untuk percobaan ini adalah sebagai berikut

****

Gambar 4.7 Diagram alir kode percobaan pengendali relay

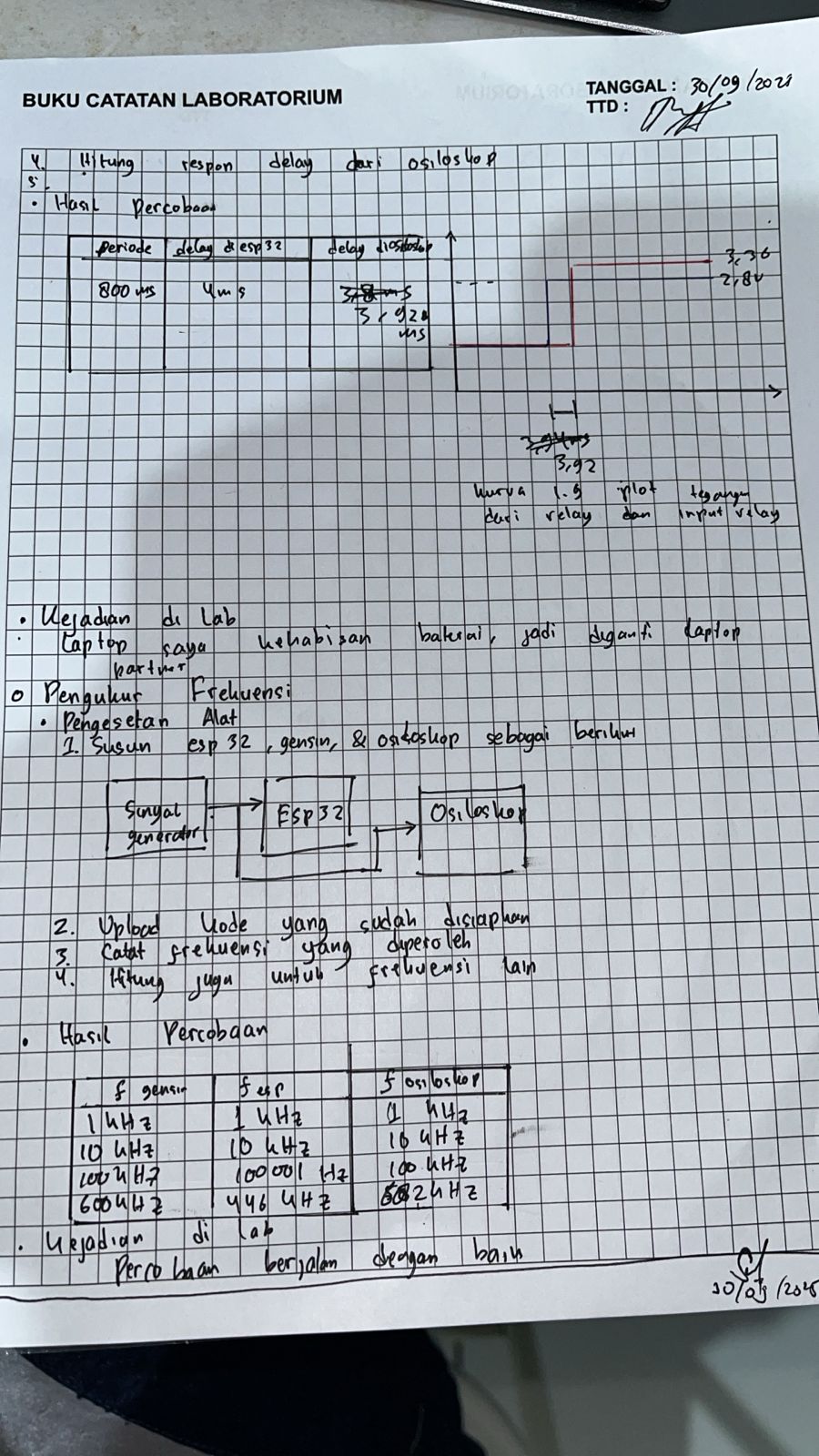
Dari percobaan tersebut diperoleh data berikut

Tabel 4.5 data delay operasi relay

|  |  |
| --- | --- |
| **Delay di ESP32** | **Delay di Osiloskop** |
| 4 ms | 3.92 ms |

Plot tegangan dapat dilihat pada kurva berikut

Kurva 4.9 plot sinyal esp32 dan dari relay

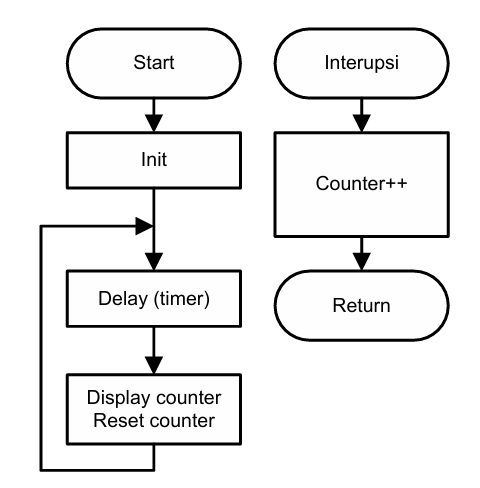


Dapat dilihat bahwa sejak ESP32 memberikan output high, perlu sekitar 4 ms bagi relay untuk menghubungkan dan memberikan respons ke ESP32, serta 3.92 ms untuk osiloskop. Terdapat sedikit perbedaan antara osiloskop dan ESP32 dikarenakan Mengetahui delay ini sangat penting dalam merancang proyek yang memerlukan relay sebagai sakelar untuk memastikan proyek berjalan dengan timing yang tepat. Hal ini terjadi karena penghitung waktu di esp32, yaitu millis() memiliki skala terkecil 1 ms sehingga hasil perhitungannya dibulatkan ke 4 ms.

4.8 Pengukur Frekuensi

Di percobaan ini, digunakan program sebagai berikut

Gambar 4.8 diagram alir percobaaan mengukur frekuensi



Dalam percobaan ini, digunakan fungsi interupsi eksternal untuk menghitung jumlah pulse sinyal. Interupsi memungkinkan mikrokontroler bereaksi instan saat ada perubahan pada pin input. Interupsi juga bekerja di luar loop() sehingga tidak akan terganggu oleh delay antar loop.

|  |  |  |
| --- | --- | --- |
| **f (kHz)** | **f terukur (Hz)** | |
| **f esp32** | **f osiloskop** |
| 1 | 1k | 1k |
| 10 | 10k | 10k |
| 100 | 100001 | 100k |
| 600 | 446k | 583k |

Dari percobaan ini, terlihat bahwa untuk frekuensi di bawah 100kHz, pencatatan frekuensi sangat akurat. Namun, semakin tinggi frekuensi perhitungannya menjadi lebih rendah dari frekuensi inputnya. Hal ini terjadi karena jumlah sinyal pulse per waktunya terlalu tinggi dan melampaui kemampuan ISR (Interrupt Service Routine)

1. **Kesimpulan**

Pin GPIO pada ESP32 dapat digunakan sebagai input dan output digital. Besar tegangan output untuk nilai HIGH adalah sekitar 3.2 V.

Pin digital memiliki kecepatan output tertentu yang bergantung pada frekuensi CPU, namun kecepatannya tidak sama persis, berdasarkan percobaan dengan menghasilkan sinyal pulse, diperoleh kecepatan output 2000 kali lebih rendah dari frekuensi CPU.

Pembacaan input dan penulisan output oleh ESP 32 memiliki latensi sekitar 5.7 ms jika menggunakan library arduino dan 4. 8 ms jika menggunakan akses register.

Apabila ingin mengoperasikan peralatan yang memiliki tegangan di atas 3 volt, perlu digunakan relay sebagai sakelar. Tapi penggunaan relay menyebabkan delay sebesar 4 ms.

Interupsi Eksternal dapat digunakan untuk menjalankan fungsi dengan input sinyal eksternal di luar loop. Namun terdapat jumlah maksimum interupsi yang dapat dilakukan. Di percobaan ini adalah sekitar 446000 per detik.

**Daftar Pustaka**

1. *Berkenalan dengan ESP32, Mikrokontroler untuk Proyek IoT - ITS News*. ITS News. <https://www.its.ac.id/news/2022/12/11/berkenalan-dengan-esp32-mikrokontroler-untuk-proyek-iot/>
2. Itsfa. (2022, December 11). Berkenalan dengan ESP32, Mikrokontroler untuk Proyek IoT - ITS News. ITS News. <https://www.its.ac.id/news/2022/12/11/berkenalan-dengan-esp32-mikrokontroler-untuk-proyek-iot/>
3. Suyadhi, T. D. (n.d.). Interupsi pada mikrokontroler AVR ATMEGA32. <https://www.robotics-university.com/2015/03/interupsi-pada-mikrokontroler-avr-atmega32.html>
4. Hty. (n.d.). Apa itu Relay dan Bagaimana Cara Kerjanya. <https://www.pcbasic.com/id/blog/how_does_a_relay_work.html>
5. Membaca GPIO pada ESP32 Dengan Register. (2025, August 24). Mikrokontroler ESP32. <https://esp32.iptek.web.id/docs/membaca-gpio-dengan-register/>
6. Petunjuk Praktikum Sistem Mikroprosesor Modul 1 : Input Output Digital dan Interupsi Eksternal