JITET (Jurnal Informatika dan Teknik Elektro Terapan)
Vol. 14 No. 3, pISSN: 2303-0577 eISSN: 2830-7062 http://dx.doi.org/10.23960/jitet.v14i3.9563

ANALISIS TRADE-OFFS PERFORMA ANTARA HARDWARE DAN SOFTWARE I²C PADA DRIVER MPU6050

Muhamad Rizqi Fajri[1*], Carudin, M.Kom.[2]

[1,2]Universitas Singaperbangsa Karawang; Telukjambe Timur, Karawang, Jawa Barat; (0267) 641177

Keywords:
Benchmark;
Design Pattern;
Embedded System;
ESP-IDF;
I²C

Corespondent Email:
2210631170081@student.unsika.ac.id

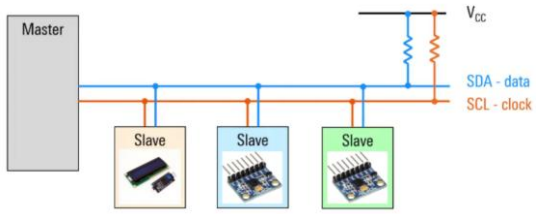
Abstrak. *Perbedaan implementasi I²C hardware (HW) dan software (SW) pada sistem tertanam sering menyebabkan tantangan kompatibilitas dalam pengembangan driver sensor, yang berujung pada duplikasi kode dan peningkatan beban CPU. Penelitian ini mengusulkan solusi berbasis Adapter Design Pattern untuk menyatukan antarmuka HW dan SW I²C, menggunakan sensor MPU6050 sebagai studi kasus. Kedua adapter (HardI2C dan SoftI2C) mengimplementasikan antarmuka I2CInterface yang sama, memungkinkan driver MPU6050 beroperasi secara identik pada kedua jenis I²C. Hasil eksperimen menunjukkan bahwa pada kecepatan clock 300kHz, implementasi HW I²C menghasilkan beban CPU <10%, sementara SW I²C mengonsumsi >28% CPU akibat overhead bit-banging. Temuan ini membuktikan bahwa solusi berbasis adapter tidak hanya meningkatkan kompatibilitas, tetapi juga memungkinkan pengembang memilih implementasi I²C secara dinamis berdasarkan pertimbangan kinerja atau ketersediaan periferal, tanpa modifikasi signifikan pada kode driver.*

Abstract. *The differing implementations of hardware (HW) and software (SW) I²C in embedded systems often lead to compatibility challenges in sensor driver development, resulting in code duplication and increased CPU load. This study proposes an Adapter Design Pattern-based solution to unify HW and SW I²C interfaces, using the MPU6050 sensor as a case study. Both adapters (HardI2C and SoftI2C) implement the same I2CInterface, allowing the MPU6050 driver to operate identically on both I²C types. Experimental results show that at a 300kHz clock speed, HW I²C maintains a CPU load of <10%, while SW I²C consumes >28% due to bit-banging overhead. These findings demonstrate that the adapter-based approach not only enhances compatibility but also enables developers to dynamically select I²C implementations based on performance or peripheral constraints—without significant driver code modifications.*

1. PENDAHULUAN

Protokol I2C merupakan standar komunikasi yang banyak digunakan dalam sistem tertanam untuk pertukaran data jarak dekat. Keunggulan utamanya terletak pada kesederhanaannya yang hanya membutuhkan dua jalur sinyal (SDA untuk data dan SCL untuk clock), serta kemampuannya menghubungkan hingga 1024 perangkat dengan alamat berbeda dalam satu bus yang sama [1]. Implementasi I2C banyak diaplikasikan pada berbagai modul sensor dan aktuator, salah satunya adalah modul LCD dengan converter I²C seperti PCF8574 yang mampu mengurangi

kebutuhan koneksi pin dari 14 pin parallel menjadi hanya 2 pin I²C, sehingga sangat mengoptimalkan penggunaan pin pada mikrokontroler yang terbatas.[7]



Gambar 1 Jalur / Bus I²C dengan 3 perangkat lain (slave)

Meskipun memiliki banyak keunggulan, implementasi I2C pada platform mikrokontroler modern seperti ESP32-S3 menghadapi beberapa tantangan utama. Pertama, keterbatasan jumlah port I²C hardware (hanya tersedia 2 port), [2] menyebabkan ketidakmampuan menghubungkan beberapa perangkat dengan alamat I²C yang sama. Kedua, solusi software I²C (bit-banging) yang diimplementasikan melalui GPIO memang lebih fleksibel namun menimbulkan overhead komputasi yang signifikan (mencapai >28% utilisasi CPU pada kecepatan 300kHz) karena semua proses timing dan protokol harus dihandle oleh CPU. [4]

Berdasarkan permasalahan tersebut, penelitian ini bertujuan untuk: (1) menganalisis trade-off performa antara implementasi hardware dan software I²C secara komprehensif, (2) mengimplementasikan solusi berbasis adapter design pattern untuk menyatukan antarmuka keduanya [5] sehingga tidak diperlukan perubahan signifikan pada driver perangkat (seperti driver MPU6050), dan (3) memberikan rekomendasi implementasi berdasarkan karakteristik aplikasi. Berbeda dengan penelitian [6] yang menghubungkan beberapa MPU9250 menggunakan *multiplexer* (komponen external), untuk mengatasi keterbatasan port I²C controller, solusi dalam penelitian ini:

1. Memanfaatkan kombinasi dinamis antara port hardware I²C yang tersedia dan implementasi software I²C pada GPIO.
2. Menghindari kebutuhan komponen eksternal tambahan.
3. Memberikan fleksibilitas pada mikrokontroller dengan peripheral bawaan terbatas.

2. TINJAUAN PUSTAKA

2.1 Protokol I2C pada Sistem Tertanam

Inter-Integrated Circuit (I2C) merupakan protokol komunikasi serial sinkron yang широко digunakan dalam sistem tertanam untuk komunikasi antar perangkat dengan jarak pendek. Protokol ini hanya memanfaatkan dua jalur utama, yaitu Serial Data (SDA) dan Serial Clock (SCL), sehingga efisien dalam penggunaan pin mikrokontroler. Selain itu, I2C mendukung arsitektur multi-master dan multi-slave dengan pengalamatan unik untuk setiap perangkat dalam satu bus [1]. Dalam implementasinya, I2C banyak digunakan pada sensor dan modul eksternal seperti MPU6050, serta perangkat lain seperti ekspander I/O (misalnya PCF8574) yang memungkinkan efisiensi penggunaan pin pada mikrokontroler.[8]

Keterbatasan protokol I2C meliputi ruang pengalamatan yang terbatas sehingga terkadang memerlukan penggunaan multiplexer tambahan, serta bandwidth yang relatif rendah akibat kebutuhan resistor pull-up yang membatasi penggunaannya pada aplikasi berkecepatan rendah. Selain itu, tidak adanya fitur pengalamatan lanjutan seperti multicast dan broadcast mengurangi fleksibilitas komunikasi antar perangkat. Karakteristik open-drain pada bus I2C juga menyebabkan konsumsi daya yang tidak dapat diabaikan, terutama pada sistem dengan banyak perangkat yang terhubung.[9]

2.2 ESP32-S3 dan ESP-IDF

ESP32-S3 merupakan mikrokontroller keluarga ESP32 dari *Espressif System*, memiliki CPU clock hingga 240MHz, telah terintegrasi beberapa *peripheral* salah diantaranya adalah modul WiFi, *bluetooth,* dan 2x I2C *Controller.* ESP-IDF merupakan sebuah *framework* resmi buatan *Espressif System* untuk keluarga SoC ESP32 mereka, *framework* ini menyediakan kebebasan untuk mengkontrol *peripheral-peripheral* yang ada pada ESP32, menggunakan

bahasa C/C++ sebagai bahasa yang kompatible saat melakukan *programming.* [2]

Gambar 2 ESP32-S3 *dev board*

2.3 MPU6050

MPU6050 merup[akan sensor inersia yang didalam terintegrasi giroskop yang merepresentasikan kecepatan *angular* dan akselerometer yang merepresentasikan percepatan linear. Pertukaran data dilakukan melalui protokol I2C, dengan opsi alamat perangkat 0x68 dan 0x69, sehingga tanpa bantuan *hardware* external 1 bus I2C hanya dapat menampung 2 perangkat MPU6050 saja. [10]

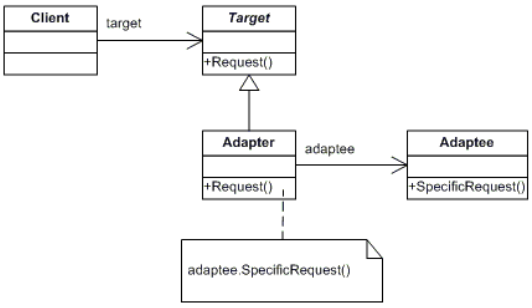
MPU6050 terbilang sangat akurat untuk merepresentasikan komponen *pitch* dan *roll* suatu orientasi, akan tetapi hal tersebut tidak berlaku untuk komponen *yaw* dikarenakan kurangnya sensor magnetometer. [3]

Gambar 3 Sensor MPU6050

2.4 Adapter Design Pattern

Abstraksi dalam pengembangan perangkat lunak merupakan pendekatan penting untuk mengurangi kompleksitas dan meningkatkan modularitas sistem. Dengan memisahkan antarmuka dari implementasi, pengembang dapat membangun komponen yang fleksibel dan mudah dipelihara tanpa bergantung pada detail teknis tertentu. Dalam konteks pengembangan driver pada sistem embedded, penggunaan abstraksi memungkinkan berbagai implementasi, seperti hardware dan software I2C, untuk diintegrasikan melalui satu antarmuka yang konsisten. Pendekatan ini sejalan dengan prinsip desain perangkat lunak yang menekankan reusability dan maintainability, serta memungkinkan perubahan atau penggantian implementasi tanpa mempengaruhi bagian sistem lainnya.[13]

Adapter Design Pattern adalah pola desain struktural yang mengubah antarmuka (interface) suatu kelas menjadi antarmuka lain yang diharapkan oleh klien, sehingga memungkinkan kelas-kelas dengan antarmuka yang tidak kompatibel untuk bekerja bersama; pola ini juga dikenal sebagai Wrapper. Pola ini sangat berguna untuk menggunakan kembali kelas yang sudah ada tanpa harus memodifikasi kode sumbernya, dengan cara bertindak sebagai penghubung antara dua antarmuka yang berbeda.[5]



Gambar 4 struktur UML dari *adapter design pattern*

2.5 FreeRTOS

FreeRTOS merupakan sistem operasi real-time (RTOS) ringan yang dirancang untuk sistem embedded dengan sumber daya terbatas. FreeRTOS menyediakan mekanisme multitasking berbasis prioritas, manajemen task, serta fitur sinkronisasi seperti semaphore, mutex, dan queue untuk mengatur komunikasi antar task. Dengan scheduler preemptive, FreeRTOS memungkinkan eksekusi task secara

efisien sesuai prioritasnya, sehingga cocok untuk aplikasi yang membutuhkan respons waktu nyata. Selain itu, FreeRTOS juga menyediakan API untuk monitoring sistem, seperti pengukuran runtime dan utilisasi CPU, yang mendukung analisis performa pada sistem embedded.[11]

2.6 Penelitian Terkait

Beberapa penelitian sebelumnya telah membahas penggunaan I2C dalam sistem embedded, termasuk penggunaan multiplexer untuk mengatasi keterbatasan alamat perangkat I2C yang sama [6]. Namun, pendekatan tersebut memerlukan komponen tambahan pada sisi hardware.

Berbeda dengan penelitian sebelumnya, pendekatan dalam penelitian ini berfokus pada pemanfaatan kombinasi hardware dan software I2C secara dinamis tanpa memerlukan komponen eksternal tambahan, serta menggunakan pendekatan desain perangkat lunak untuk meningkatkan fleksibilitas sistem. Serta melakukan perbandingan *trade-offs* antara implementasi *Hardware* dan *Software* dari I2C.

3. METODE PENELITIAN

3.1 Jenis Penelitian

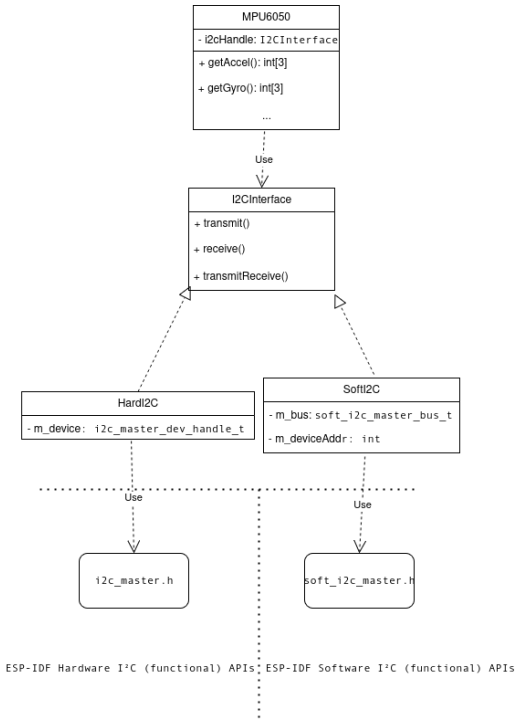
Penelitian ini merupakan penelitian eksperimental yang bertujuan untuk menganalisis perbedaan performa antara implementasi hardware I2C dan software I2C pada sistem embedded. Pendekatan yang digunakan adalah kuantitatif, dengan melakukan pengukuran terhadap parameter kinerja sistem seperti jumlah instruksi, siklus CPU, bubble cycle, runtime, dan utilisasi CPU.

3.2 Perancangan Sistem

Tahap ini merupakan implementasi dari tujuan penelitian yang telah didefinisikan menjadi sebuah desain yang dapat diimplementasikan.

3.2.1 Diagram Kelas Sistem

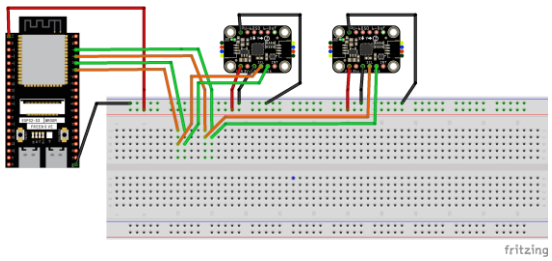
Gambar 5 menunjukkan Diagram kelas implementasi Adapter Design Pattern untuk unifikasi antarmuka HW/SW I²C menggunakan framework ESP-IDF.[5] HardI2C dan SoftI2C mengimplementasikan I2CInterface, memungkinkan driver MPU6050 bekerja pada kedua tipe I²C tanpa modifikasi lebih lanjut. Kelas MPU6050 diadaptasi dari implementasi i2cdevlib untuk ESP32. [14]



Gambar 5 Diagram Kelas Sistem

3.2.2 Diagram Skematik Hardware

Gambar 6 menunjukkan diagram skematis sistem secara keseluruhan, termasuk koneksi I²C antara ESP32-S3 (master) dan MPU6050 (slave). Jalur I²C pertama (via pin GPIO 1 dan GPIO 2) menggunakan HW I²C, sedangkan jalur I²C kedua (via pin GPIO 41 dan GPIO 42) menggunakan SW I²C.



Gambar 6 Diagram skematis *hardware*

3.3 Implementasi Sistem

Tahap ini merupakan proses realisasi dari desain skematik yang sebelumnya telah dibuat menjadi sebuah sistem yang fungsional. Tahap ini dimulai dari implementasi skematik *hardware* dan integrasi *firmware.*

3.4 Skenario Pengujian

Tahap ini dilakukan untuk mengumpulkan data-data aktual yang berkorespon terhadap tujuan penelitian daripada sistem.

3.4.1 Perangkat & Konfigurasi

Tabel 1 Perangkat & Konfigurasi Pengujian

	Master	***Slave***	***Clock Bus Frequency***	***Slave Vin***
HW I2C	ESP32-S3	MPU6050 (0x68)	300Khz	3.3V
SW I2C				

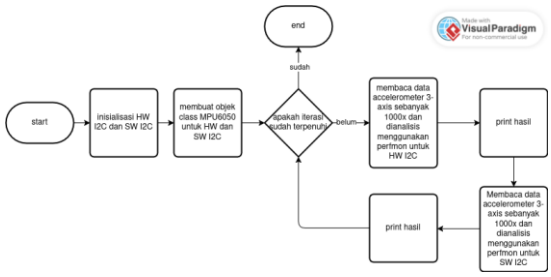
Tabel 1 memperlihatkan konfigurasi I2C pada proses pengujian. Kedua MPU6050 menggunakan *address* yang sama dengan jalur bus I2C yang berbeda.

3.4.2 Parameter dan Alat ukur

Tabel 2 Parameter yang diuji, alat ukur, serta tujuannya

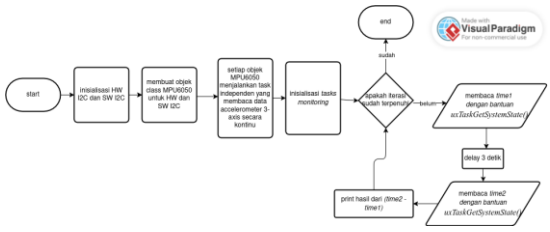
Parameter	**Alat Ukur**	**Tujuan**
Total Intruksi	***Perfmon*** (*library* dari ESP-IDF [15])	Mengukur kompleksitas intruksi antara HW dan SW I2C
Total *Cycles*		Membandingkan efesiensi *cycles* yang dibutuhkan untuk transaksi I2C yang sama
Total *Bubble Cycles*		Mengukur *cycle* yang tidak aktif digunakan oleh CPU
Total *Runtime*	***UxTaskGetSystemState()***, fungsi dari FreeRTOS, dengan mengukur selisih waktu transaksi dimulai dan selesai	Menghitung total *runtime* CPU saat "bekerja" pada selang waktu 3 detik

3.4.3 Alur Pengujian



Gambar 7 diagram alir pengumpulan data parameter pengujian (total intruksi, *cycles,* dan *buble-cycles*)

Gambar 7 menggambarkan diagram alir untuk mengumpulkan nilai aktual dari 3 parameter pengujian dengan bantuan *library perfmon*.



Gambar 8 diagram alir pengumpulan data parameter pengujian total *runtime*

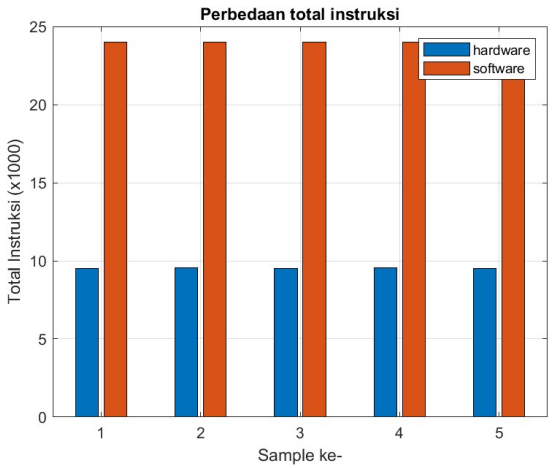
Gambar 8 merupakan diagram alir yang mengumpulkan nilai aktual dari parameter pengujian total *runtime* dengan bantuan fungsi ***uxTaskGetSystemState()***

3.5 Analisis Data

Tahap ini dilakukan untuk menganalisis data-data aktual yang telah dikumpulkan pada tahap sebelumnya. Metode analisis dilakukan secara komparatif antara Hardware dan Software I2C.

4. HASIL DAN PEMBAHASAN

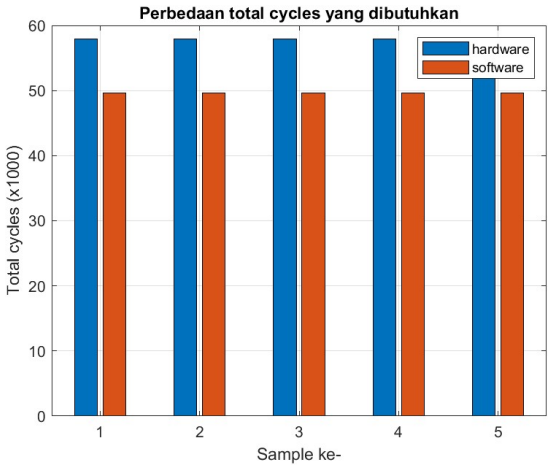
4.1 Perbandingan Beban Komputasi



Gambar 9 perbandingan total instruksi Setiap transaksi pengambilan data accelerometer

Berdasarkan **Gambar 9** SW I2C memiliki 2.6x lebih banyak instruksi (24.000) dibandingkan dengan HW I2C yang hanya memiliki sekitar 9.000 instruksi per transaksi. Penyebab hal ini adalah implementasi bit-banging pada SW I2C memiliki tambahan untuk manipulasi level pada GPIO, mengatur timing clock, dan juga penanganan error seperti apakah data tersebut ACK atau NACK, semuanya dibebankan ke CPU.

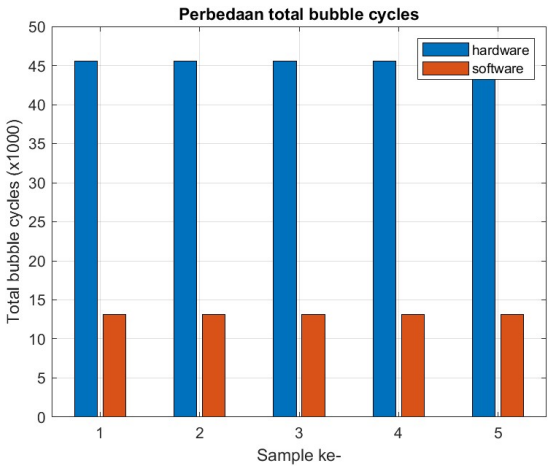
4.2 Perbandingan Total Cycle CPU



Gambar 10 perbandingan total cycle CPU setiap transaksi

Berdasarkan **Gambar 10** terlihat bahwa HW I^2C memiliki *total cycle* yang lebih dibandingkan dengan SW I^2C, akan tetapi *cycles* pada HW I^2C bukan karena CPU aktif bekerja atau menggunakan *cycle*-nya melainkan CPU sedang menunggu sesuatu dari *peripheral* lain, berbeda dengan SW I^2C yang diamana CPU secara langsung menggunakan *cycles* tersebut untuk eksekusi instruksi *bit-banging.*

4.3 Perbandingan Total Bubble-Cycle CPU



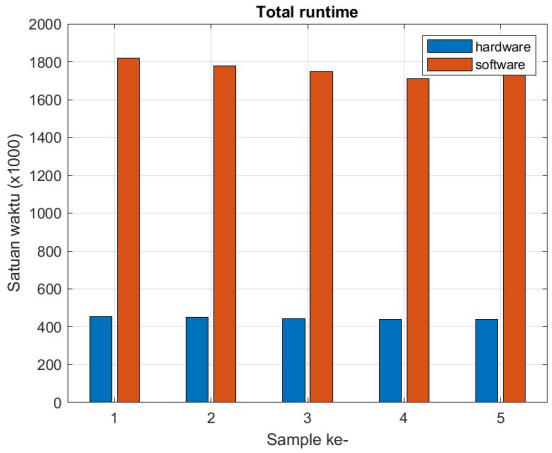
Gambar 11 perbandingan bubble-cycles setiap transaksi

Dari **Gambar 11** terlihat bahwa HW I2C memiliki sekitar 79% dari total cycles-nya adalah bubble-cycle, sedangkan untuk SW I2C memiliki sekitar 26%, sehingga,

$$67.089\% = \frac{(79-26)}{79} \cdot 100\% \quad (4.1)$$

Bubble-cycle pada SW I2C lebih rendah sebanyak 67% dibandingkan dengan HW I2C, hal ini dikarenakan segala operasi yang dibutuhkan pada proses transaksi I2C dibebankan semuanya ke CPU sehingga CPU lebih aktif menggunakan cycle-nya. Semakin besar bubble-cycle semakin jarang pula CPU aktif menggunakan cycle-nya.

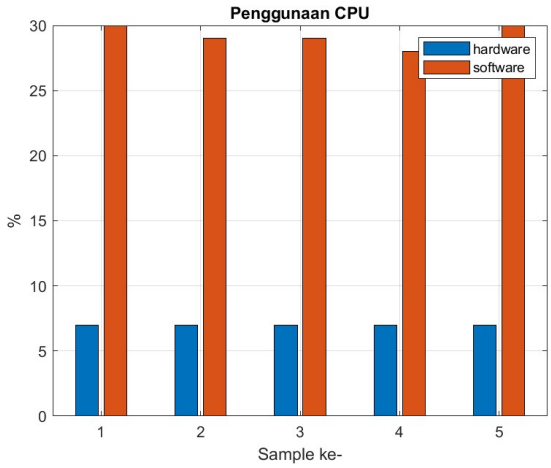
4.4 Perbandingan Total Runtime



Gambar 12 perbandingan total runtime (3 detik monitoring per sampel) CPU aktif menggunakan *cycle.*

Gambar 12 menggambarkan bahwa SW I^{2}C memiliki total runtime sekitar 328.57% lebih besar dari HW I^{2}C untuk hasil yang sama. Semakin besar *total runtime* untuk tiap transaksi maka semakin banyak pula energi yang dibutuhkan.

4.4 Perbandingan Total Penggunaan CPU



Gambar 13 perbandingan total penggunaan CPU (utilisasi)

Nilai pada **Gambar 13** didapat dari,

$$utilisasi = \frac{task_{runtime}}{total_{runtime} \cdot total_{cpucores}} \cdot 100\% \quad (4.2)$$

$task_{runtime}$ merupakan durasi CPU mengeksekusi suatu *task* (dalam hal ini adalah HW dan SW task, seperti pada **Gambar 8**), $total_{runtime}$ merupakan total durasi observasi pada tiap sampel, dan $total_{cpucores}$ merupakan total cpu core dari ESP32-S3 dalam hal ini adalah 2.

Pada **Gambar 13** terlihat bahwa SW I^{2}C memiliki utilisasi CPU jauh lebih besar dibandingkan dengan HW I^{2}C, dikarenakan pada HW I^{2}C, CPU terbantu dengan *peripheral* lain yakni I^{2}C *controller* yang tersedia dalam ESP32-S3.

4.5 Pembahasan Lanjutan

Tabel 3 perbandingan akhir dari HW dan SW I^{2}C

Aspek	HW I2C	SW I2C
Kecepatan	Cepat	Lambat
Fleksibilitas	Terbatas pada *peripheral* external	Tak terbatas pada *peripheral* external
Energi	Efisien	Boros

Pada **Tabel 3,**

1. hasil pada aspek kecepatan didapat berdasarkan total *runtime* yang telah didapat dari proses pengujian
2. hasil pada aspek fleksibilitas didapat dikarenakan HW I2C terbatas pada *peripheral* I2C *controller* yang tersedia pada mikrokontroller, setiap I2C *controller* hanya dapat menangani 1 jalur bus I2C saja, sedangkan SW I2C hanya terbatas pada pin GPIO yang tersedia pada mikrokontroller tersebut, dan juga *power* komputasi yang dimiliki oleh mikrokontroller tersebut.
3. Hasil pada aspek energi didapat dari data total intruksi, *cycles,* dan *bubble-cycles* yang sebelumnya didapatkan pada proses pengujian. Pada SW I2C semua proses transaksi dibebankan semuanya ke CPU sehingga butuh komputasi yang lebih dan juga sumber data yang lebih.

5. KESIMPULAN

Berdasarkan hasil pengujian dan pembahasan kinerja komputasi SW I^2C secara signifikan lebih berat dibandingkan dengan HW I^2C akibat *bit-banging* manual, dengan instruksi yang lebih banyak. Sedangkan HW I^2C mengalihkan beban komputasi ke I^2C *controller* sehingga memiliki beban komputasi yang lebih rendah untuk CPU, meskipun memiliki *bubble-cycle* yang tinggi dibandingkan dengan SW I^2C.

Dalam utilisasi CPU HW I^2C 328% lebih efisien dari SW I^2C, dengan CPU *load* kurang dari 10%. HW I^2C juga memiliki *total runtime* yang jauh lebih singkat dari SW I^2C, sehingga cocok untuk kasus yang membutuhkan penganan *real-time.* Meskipun SW I^2C memiliki utilisasi CPU dan *total runtime* yang jauh lebih tinggi daripada HW I^2C, SW I^2C jauh lebih fleksibel dan tidak tergantung terhadap ketersediaan I^2C *controller* pada mikrokontroller yang dipilih, sehingga cocok untuk sebuah kasus dimana dibutuhkan beberapa perangkat I^2C yang sama karena biasanya memiliki alamat yang sama.

Dengan bantuan *adapter design pattern,* pengguna dapat memilih untuk menggunakan pendekatan HW I^2C ataupun SW I^2C untuk komponen externalnya menyesuaikan dengan kondisi yang dibutuhkan, tanpa harus mengubah *driver library* secasra signifikan.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan dukungan dalam pelaksanaan penelitian ini, khususnya kepada orang tua atas doa dan motivasi yang diberikan, serta kepada dosen pembimbing, Bapak Carudin, M.Kom., atas bimbingan, arahan, dan ilmu yang sangat berharga selama proses penelitian.

DAFTAR PUSTAKA

[1] Patel, Sagarkumar & Talati, Prachi & Gandhi, Saniya. (2019). Design of I2C Protocol.

[2] "ESP32 Wi-Fi & Bluetooth SOC," *Espressif Systems*. https://www.espressif.com/en/products/socs/esp32

[3] N. D. Imtinan, "Two-Wheel Self-Balancing Robot with PID-Fuzzy and Wall-Following," JITET, vol. 13, no. 3, Jul. 2025, doi: 10.23960/jitet.v13i3.6940.

[4] Halder, R. Dasgupta, J. Chepada, and N. Dash, 2011, Implementation of MCU Invariant I2C Slave Driver Using Bit Banging, *https://www.thinkmind.org/articles/icons_2011_1_10_20003.pdf*, diakses tgl 3 Agustus 2025.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, . 1994, Design patterns: elements of reusable Object-Oriented software*, https://www.grch.com.ar/docs/unlu.poo/Gama-DesignPatternsIntro.pdf,* diakses tgl 3 Agustus 2025.

[6] C. K. Mummadi, F. P. P. Leo, K. D. Verma, S. Kasireddy, P. M. Scholl, and K. Van Laerhoven, Sep. 2017, Real-time Embedded Recognition of Sign Language Alphabet Fingerspelling in an IMU-Based Glove, *ACM Digital Library*, No.11, 1–6, https://doi.org/10.1145/3134230.3134236.

[7] G. S. S. Varma, B. Naveen, P. R. Reddy, and N. Priyanka, "Design and Performance Evaluation of I2C and AMBA AXI Protocols for Embedded System Integration," 2025 8th International Conference on Computing Methodologies and Communication (ICCMC). IEEE, pp. 286–290, Jul. 23, 2025. doi: 10.1109/iccmc65190.2025.11140841.

[8] Zibayiwa, Morelife,. 2021. A Review on The Inter-Processor Communication: I2C, UART, and SPI interfacing techniques.

[9] P. Visconti, G. Giannotta, R. Brama, P. Primiceri, A. Malvasi, and A. Centuori,

"FEATURES, OPERATION PRINCIPLE AND LIMITS OF SPI AND I2C COMMUNICATION PROTOCOLS FOR SMART OBJECTS: A NOVEL SPI-BASED HYBRID PROTOCOL ESPECIALLY SUITABLE FOR IoT APPLICATIONS," International Journal on Smart Sensing and Intelligent Systems, vol. 10, no. 2, pp. 1–34, Jan. 2017, doi: 10.21307/ijssis-2017-211.

[10] TDK InvenSense, "MPU-6000 and MPU-6050 Product Specification," Rev. 3.4, PS-MPU-6000A-00, 2015. [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[11] FreeRTOS, "The FreeRTOS™ Reference Manual," Amazon.com, version 10.0.0 issue 1., 2017. Accessed: Apr. 26, 2026. [Online]. Available: https://www.freertos.org/media/2018/FreeRTOS_Reference_Manual_V10.0.0.pdf

[12] J. L. Hennessy and D. A. Patterson, *Computer Architecture: a Quantitative approach*. 2019. [Online]. Available: http://www.loc.gov/catdir/toc/els031/2001099789.html

[13] S. McConnell, *Code complete, second edition*. 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=1096143

[14] J. Rowberg, "MPU6050 library for ESP32 (ESP-IDF)," *i2cdevlib*, 2022. Accessed: Apr. 26, 2026. [Online]. Available: https://github.com/jrowberg/i2cdevlib/tree/master/ESP32_ESP-IDF/components/MPU6050.

[15] Espressif Systems, "Performance Monitor example (system/perfmon)," *ESP-IDF Examples*, 2026. Accessed: Apr. 26, 2026 [Online]. Available: https://github.com/espressif/esp-idf/tree/v6.0/examples/system/perfmon.