Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 3, No. 10, Oktober 2019, hlm. 10249-10257 http://j-ptiik.ub.ac.id

Implementasi *Data Mirroring* Pada Metode *Hot Standby Redundancy* Berbasis Protokol I^{2}C dan Arduino Uno

Arif Nur Agung Laksana[1], Wijaya Kurniawan[2], Agung Setia Budi[3]

Program Studi Teknik Komputer, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]agung.arifnur@gmail.com, [2]wjaykurnia@ub.ac.id, [3]agungsetiabudi@ub.ac.id

Abstrak

Sebuah sistem dapat dinyatakan sebagai sistem yang handal ketika sistem tersebut mampu bekerja sesuai dengan fungsinya walaupun sempat mengalami kegagalan. Salah satu cara untuk meningkatkan kehandalan sistem adalah dengan menerapkan *Hardware Redundancy* dimana dibuat sebuah duplikat dari komponen utama suatu sistem. Ada dua jenis metode dari *Hardware Redundancy* yaitu *Hot Standby Redundancy* dan *Cold Standby Redundancy* . Pada penelitian ini, selain menerapkan metode *Hot Standby Redundancy* tersebut, akan diterapkan juga mekanisme *Data Mirroring* pada sistem untuk memastikan komponen utama dan komponen cadangan menyimpan data yang sama. Sistem yang akan diduplikat adalah sistem pintu berbasis RFID dimana data yang akan disalin merupakan data tag dari RFID tersebut. Data tag RFID tersebut dapat disalin dari komponen utama ke komponen cadangan, maupun sebaliknya. Sistem juga dapat melakukan peralihan tugas dari komponen utama ke komponen cadangan ketika terjadi kegagalan pada komponen utama. Implementasi sistem dilakukan pada dua mikrokontroler Arduino UNO berbeda yang masing-masing terhubung dengan modul RFID, *relay 1 channel*, dan kunci pintu solenoid. Pengguna dapat mendaftarkan dan menghapus tag RFID yang nantinya akan digunakan untuk mengakses kunci pintu solenoid. Dari pengujian pengukuran waktu yang dibutuhkan oleh sistem ketika melakukan sinkronisasi, didapatkan waktu 950,4 mikrodetik dan 784,8 mikro detik ketika melakukan resinkronisasi.

Kata kunci: *sinkronisasi, mirroring, fault tolerant, Hardware Redundancy, Hot Standby Redundancy*

Abstract

A system can be declared as a reliable system when the system is able to work in accordance with its functions even though it has failed. One way to improve system reliability is to implement Hardware Redundancy where a main components of the system is duplicated. There are two types of Hardware Redundancy methods, namely Hot Standby Redundancy and Cold Standby Redundancy. In this research, besides applying the Hot Standby Redundancy method, Data Mirroring mechanism will be applied to the system to ensure the main components and backup components store the same data. The system to be duplicated is an RFID-based door system where the data to be copied is RFID tag data. The RFID tag data can be copied from the main component to the backup component, and vice versa. The system can also switch tasks from the main component to the backup component when the main component fails. The system is implementated on two different UNO Arduino microcontrollers, each is connected with an RFID module, 1 channel relay, and solenoid door lock. Users can register and delete RFID tags which will be used to access solenoid door locks. From the test measurement time required by the system when synchronizing obtained 950.4 microsecond and 784.8 micro seconds when resynchronizing.

Keywords: *synchronization, mirroring, fault tolerant, Hardware Redundancy, Hot Standby Redundancy*

1. PENDAHULUAN

Penggunaan sistem otomatis dalam perangkat *smart home* semakin marak digunakan, dimana jumlah konsumen perangkat-perangkat *smart home* mengalami peningkatan pada setiap tahun (Ablondi, 2014). Dengan semakin banyaknya pengguna, maka dalam pengembangan sistem perlu dipastikan sistem mampu bekerja secara handal. Sistem dinyatakan sebagai sistem yang handal apabila sistem tersebut mampu berjalan

sebagai mana mestinya meski terjadi *fault* pada sistem (Ramamoorthy, 1971). *Fault* itu sendiri dapat terjadi secara sementara maupun permanen (Sorin, 2009) sehingga diperlukan metode untuk menanggulangi *fault* tersebut. Salah satu cara yang dapat digunakan untuk menanggulangi *fault* yaitu metode redundansi. Secara umum redundansi dapat didefinisikan sebagai suatu duplikasi sistem baik dari segi perangkat keras (*hardware redundancy*) ataupun perangkat lunak (*software redundancy*) dengan tujuan untuk memastikan sistem tersebut dapat tetap berfungsi dengan normal walaupun terdapat satu atau lebih elemen yang tidak berfungsi (Dubrova, 2013). Salah satu jenis *hardware redundancy* adalah *standby redundancy*, yaitu unit spare bersifat *standby* dan akan aktif ketika komponen utama mengalami kegagalan fungsi. *Standby redundancy* sendiri memiliki dua metode yaitu *cold standby* dan *hot standby*. Pada *cold standby* unit *standby* tidak tercatu daya, sedangkan pada *hot standby* unit *standby* tercatu daya namun berada pada kondisi tidak aktif. *Hot standby redundancy* adalah sebuah mekanisme menambahkan duplikat dari suatu komponen penting dalam keadaan *sleep* sebagai cadangan yang hanya aktif ketika komponen penting tersebut mengalami kegagalan (Dubrova, 2013). *Hot standby* memiliki keunggulan dari waktu *downtime* yang lebih singkat dan kehandalan unit lebih terjaga karena unit *standby* tercatu daya. Namun pada mekanisme redundansi tersebut, terdapat kendala dimana ada kemungkinan komponen utama dan komponen cadangan menyimpan data yang berbeda. Hal tersebut dapat menyebabkan beberapa sistem mengalami malfungsi jika terjadi *fault*. Diperlukan pula proses sinkronisasi tersebut untuk memastikan komponen utama dan cadangan memiliki salinan data yang sama.

Beberapa penelitian telah dilakukan untuk mengembangkan metode redundansi tersebut. Seperti pada sistem akuisisi data sensor (Pangestu, 2018). Pada penelitian tersebut, mekanisme *hot standby redundancy* dilakukan dengan cara membuat unit *master* dalam kondisi hidup sedangkan unit *slave* berada pada kondisi *sleep*. Unit akan mengirimkan sinyal *interrupt* ke unit *slave* jika terjadi *error* pada unit *master*.. Namun dengan penggunaan sinyal *interrupt* untuk memicu aktivasi dari unit *slave*, jika terjadi kegagalan daya pada unit *master* maka unit *slave* tidak mendapat sinyal aktivasi yang mengakibatkan unit *slave* akan tetap berada dalam kondisi *sleep*. Penelitian lainnya adalah pengembangan *hardware redundancy* pada sistem *switching* pintu (Muzaki, 2018). Pada kedua penelitian tersebut, belum ada metode yang diterapkan untuk melakukan mekanisme *mirrorin data*.

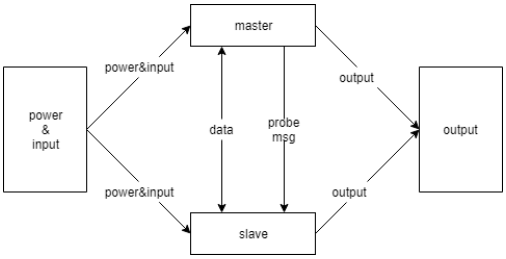
Penelitian ini mengusulkan protokol *Inter Integrated Circuit* (I^2C) sebagai jalur komunikasi data antara komponen *master* dengan *slave*. Sistem I^2C terdiri dari saluran SCL (*Serial Clock*) dan SDA (*Serial Data*) yang membawa informasi data antara I^2C dengan pengontrolnya (IEEE, 2009). Melalui protokol I^2C, data dari perangkat *master* akan dikirimkan secara berkala ke perangkat *slave* agar kedua perangkat memiliki data yang identik.

Implementasi protokol I^2C dilakukan pada dua mikrokontroler Arduino UNO berbeda yang masing-masing terhubung dengan modul RFID *reader*, aktuator *relay*, dan kunci pintu solenoid. Pengguna dapat mendaftarkan dan menghapus tag RFID yang nantinya akan digunakan untuk mengakses kunci pintu solenoid. Setelah data berhasil disimpan pada salah satu mikrokontroler, data tersebut nantinya akan disalin ke mikrokontroler yang lain untuk memastikan sinkronisasi data antara perangkat *master* dengan perangkat *slave*.

2. PERANCANGAN DAN IMPLEMENTASI

2.1 Gambaran Umum Sistem

Gambaran umum sistem dapat dilihat pada diagram blok Gambar 1.



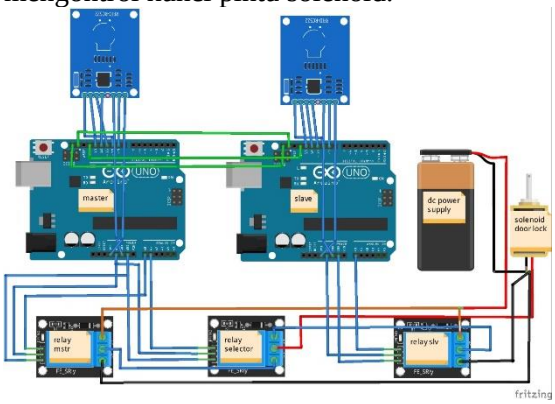
Gambar 1. Diagram blok sistem

Berdasarkan gambar 1, *master* dan *slave* saling terhubung melalui komunikasi serial I^2C dimana *slave* membaca *probe message* dari master sebagai penanda status *Keep-alive* ari *master*. Pengiriman data dilakukan dua arah baik dari *master* maupun *slave*. Data yang dikirimkan berupa data. tag RFID yang akan digunakan untuk mengontrol kunci pintu solenoid.

2.2 Perancangan Sistem

2.2.1. Perangkat *Hardware*

Perancangan *hardware* pada sistem meliputi perancangan komponen *master* dan *slave* dimana keduanya terhubung satu sama lain melalui komunikasi serial I2C. Pengembangan *prototype* sistem memanfaatkan mikrokontroler Arduino UNO. Arduino UNO memiliki kapasitas *flash memory* sebesar 32 KB untuk menyimpan kode program, dan kapasitas EEPROM sebesar 1 KB untuk menyimpan tag RFID yang nantinya akan digunakan untuk mengontrol kunci pintu solenoid.



Gambar 2. Perancangan *prototype* sistem

Rangkaian menggunakan 3 buah relay 1 channel yang salah satunya berfungsi sebagai *selector* mikrokontroler mana yang dapat mengakses *solenoid door lock*. Kemudian 2 board Arduino Uno tersebut dihubungkan dengan komunikasi serial I^2C.Komunikasi serial I2C tersebut digunakan untuk melakukan pengecekan terhadap *Keep-alive* status dari *master*, sekaligus untuk melakukan sinkronisasi data antara *master* dengan *slave*. Modul RFID berfungsi sebagai masukan dari sistem. Modul tersebut berfungsi untuk mendaftarkan maupun menghapus tag RFID yang digunakan untuk mengontrol kunci pintu solenoid.*Master* dan *slave controller* terhubung secara langsung dengan RFID *reader* MFRC522 melalui pin digital pada mikrokontroler Arduino UNO. Kemudian *relay 1 channel* dihubungkan ke masing-masing mikrokontroler melalui pin analog. Pin A0 pada *master* akan digunakan untuk mengirim sinyal pengendali ke *relay selector* dimana sinyal tersebut akan menentukan kontroler mana yang akan dapat mengendalikan kunci pintu solenoid. Kemudian pin A1 pada *master* dan *slave* akan digunakan untuk mengirim sinyal pengendali ke kunci pintu solenoid melalui *relay selector* dimana kondisi dari *relay selector* tersebut akan ditentukan oleh *master controller.*

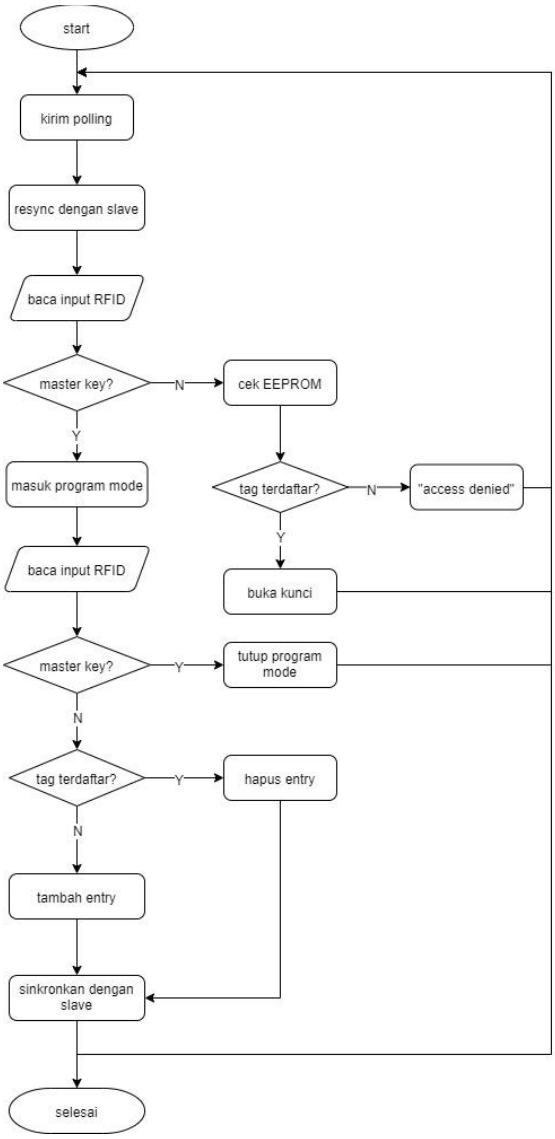
Tabel 1 memperlihatkan komponen *input* dan *output* pada setiap perangkat *smart home* yang dikembangkan.

Tabel 1. Skema *pinout* perangkat *prototype* sistem

RFID reader MFRC522	*Master*	*Slave*
SDA	10	10
SCK	13	13
MOSI	11	11
MISO	12	12
RQ	–	–
GND	GND	GND
RST	9	9
3.3v	3.3v	3.3v
–	SDA	SDA
–	SCL	SCL
–	GND	GND
–	–	A0
–	A1	A1

2.2.2. *Master Controller*

Master controller memiliki fungsi dasar untuk membaca *entry tag* dari RFID serta menyimpannya di EEPROM. Tag RFID tersebut digunakan mengontrol kunci pintu solenoid. *Master controller* juga dapat mengirimkan *probe message*sebagai penanda *keep-alive status master* kepada *slave* dalam interval waktu tertentu. Interval pengiriman tiap *polling* ditentukan selama 500ms. Pengiriman *command polling* tersebut dilakukan menggunakan komunikasi serial I^2C. Selain mengirimkan *command polling*, *master controller* juga akan melakukan sinkronisasi data secara bertahap dengan *slave controller* apabila terjadi perubahan data tag RFID pada *master controller* baik *entry* data maupun hapus data.Data tersebut juga dikirimkan melalui komunikasi serial I^2C Diagram alir *master controller* dapat dilihat pada Gambar 3 dibawah ini.

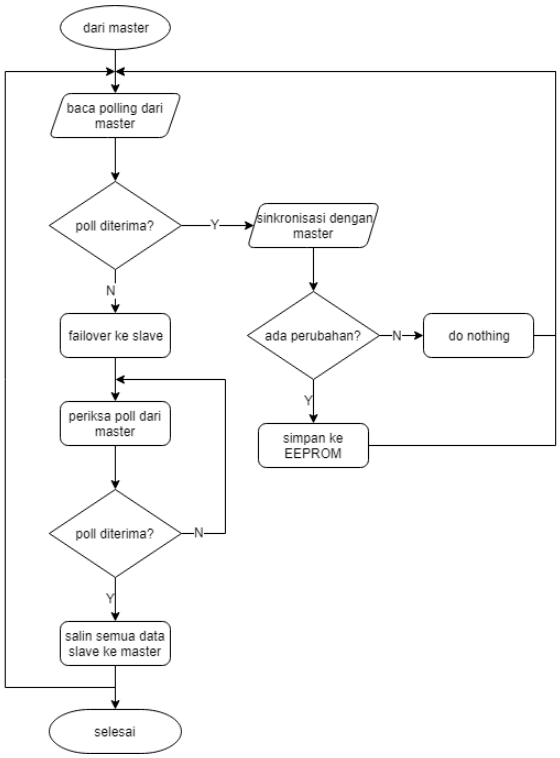


Gambar 3. Diagram alir *master controller*

Gambar 3 diatas memperlihatkan cara kerja *master controller* baik ketika bekerja secara normal maupun ketika terjadi kesalahan berupa *power outage*. *Master* akan selalu melakukan pengecekan sinkronisasi data setiap pertama kali dinyalakan. Hal tersebut guna memastikan data yang tersimpan pada *slave* sudah identik dengan data yang tersimpan pada *master*.

2.2.3 *Slave Controller*

Slave controller bertugas sebagai komponen cadangan menggantikan *master controller* ketika terjadi kesalahan pada *master controller*. *Slave* bekerja dengan membaca status dari *master controller* dimana status tersebut ditandai dengan dikirimkannya *probe message* oleh *master controller*. *Slave* hanya akan menjalankan fungsi utamanya sebagai pengndali kunci pintu solenoid jika mendeteksi status dari *master controller* berada pada keadaan offline. Cara kerja *slave* controller dapat dilihat pada Gambar 4 berikut.



Gambar 4. Diagram alir *slave controller*

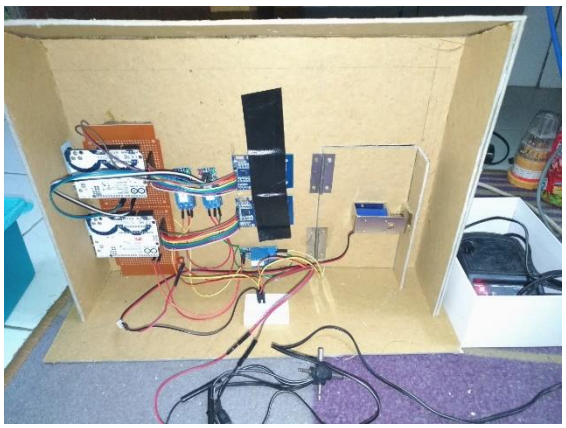
Gambar 4 diatas memperlihatkan alur kerja pada unit *slave controller*. Ketika *master* berada pada kondisi aktif, maka *slave* berada pada kondisi *idle* dan hanya akan membaca *probe message* dari master setiap 500ms. Namun ketika *master* gagal mengirim *probe message* dalam *timeout* waktu yang telah ditentukan (2500ms), maka *slave* akan berganti peran menjadi kontroller utama pada sistem. *Slave* akan mengirimkan semua data yang tersimpan dalam EEPROMnya ketika *slave* mendeteksi bahwa *master* telah kembali aktif, sekaligus memposisikan *slave* kembali menjadi kontroller cadangan bagi *master*.

2.3. Implementasi Sistem

2.3.1. Prototipe Sistem Kunci Pintu Solenoid

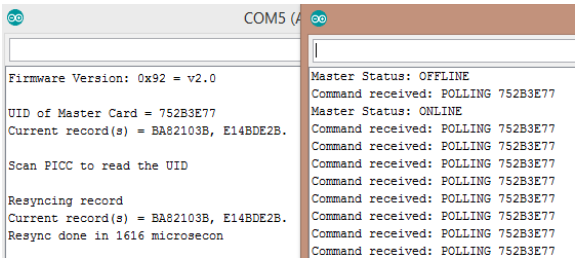
Seperti yang diperlihatkan pada Gambar 5, komponen perangkat keras yang berupa prototipe sistem kunci pintu solenoid menggunakan RFID, dirangkai pada suatu PCB berukuran 8 x 20 cm. Sumber tegangan mikrokontroler menggunakan daya sebesar 5v dan sumber tegangan yang digunakan kunci pintu solenoid sebesar 12v berasal dari power supply adapter 12v. Rangkaian alat dibuat berdasarkan perancangan yang didefinisikan pada sub bab sebelumnya.

2.3.2. Perangkat Lunak

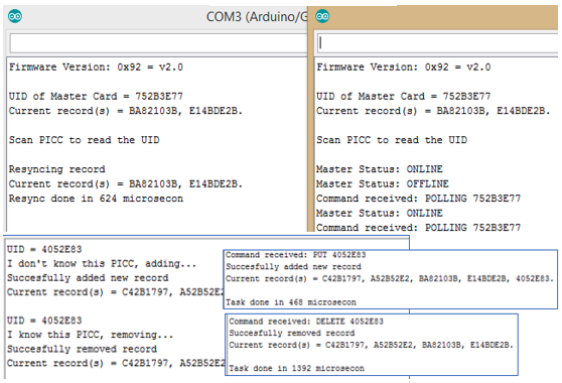


Gambar 5. Implementasi perangkat keras

Implementasi perangkat lunak dilakukan dengan menuliskan kode program pada mikrokontroler melalui Arduino IDE. Kemudian kode program diproses dengan cara mengunggah kode program tersebut pada mikrokontroler di tiap sub sistem *master* dan *slave*. Kode program yang dituliskan memiliki beberapa fungsi diantaranya fungsi monitoring, dan fungsi *mirroring*. Fungsi monitoring digunakan untuk memantau *keep-alive* status dari kontroler *master*. Fungsi *mirroring* digunakan untuk melakukan registrasi maupun penghapusan data pada EEPROM sekaligus melakukan sinkronisasi data pada tiap sub sistem. Berikut tampilan serial monitor dari Arduino IDE ketika menjalankan kode program dari sistem.



(a) *Monitoring*



(b) Sinkronisasi

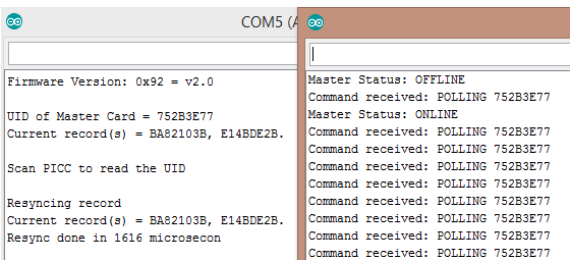
Gambar 6. Keluaran sistem

3. PENGUJIAN DAN ANALISIS

Dalam pengujian yang dilakukan, perangkat *master* dan *slave* terhubung pada serial monitor Arduino IDE untuk memantau *keep-alive* status kontroler *master* serta waktu eksekusi *task* yang dilakukan.

3.1. Pengujian Fungsi Monitoring

Pengujian ini bertujuan untuk memantau kondisi *keep-alive* status dari kontroler *master*. Pengujian dilakukan dengan memantau serial monitor pada *slave* kemudian menerapkan skenario *power-off* pada *master*. Gambar 7 memperlihatkan tampilan pada serial monitor pada kedua sub sistem.

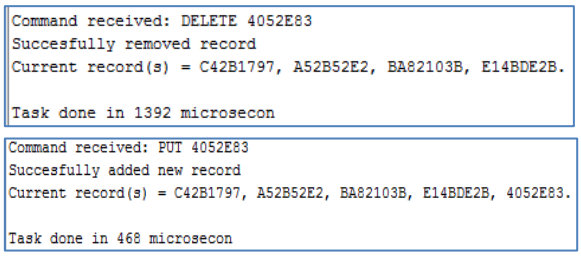


Gambar 7. Tampilan *keep-alive* status sistem

Interval pengiriman *keep-alive* status oleh *master* kepada *slave* adalah 500 ms, sedangkan waktu *timeout* yang diberlakukan agar *master* terdeteksi pada kondisi *offline* adalah 2500 ms. Serial monitor pada *slave* akan menampilkan kondisi dari *master* secara *realtime*. Berikut tabel hasil pengujian fungsi *monitoring*.

3.2. Pengujian Registrasi Data

Pengujian ini bertujuan untuk mengetahui apakah sistem dapat melakukan registrasi data berupa tag RFID atau tidak, sekaligus mengukur rata-rata waktu eksekusi sistem ketika melakukan registrasi data. Pengujian dilakukan pada komponen *master* maupun *slave*. Fungsi registrasi yang dimaksud yaitu sistem mampu membaca *tag* RFID, kemudian menyimpan ataupun menghapusnya pada EEPROM tiap kontroler. Tahapan pengujian ini dilakukan dengan menghidupkan sistem. Kemudian dilakukan entry data baru dan penghapusan data lama, masing – masing sebanyak 5 kali. Setelah tahapan – tahapan pengujian fungsi registrasi tersebut selesai dilakukan, hasil dari proses pengujian tersebut dipantau melalui serial monitor pada Arduino IDE. Gambar 8 memperlihatkan tampilan serial monitor ketika program menjalankan fungsi *entry* dan hapus data.



Gambar 8. Tampilan fungsi registrasi sistem

Waktu eksekusi *task* dihitung sejak pengiriman *command* sampai ketika task selesai dieksekusi. Hasil pengujian waktu eksekusi *task* ditampilkan dalam satuan waktu mikrodetik. Seperti yang tercantum pada Tabel 2, rata-rata waktu eksekusi *task* registrasi adalah 950,4 mikrodetik. Hasil pengujian menunjukkan bahwa sistem mampu melakukan registrasi data pada EEPROM dengan baik,

Tabel 2. Daftar hasil pengukuran waktu eksekusi *task* registrasi data (mikrodetik)

Uji ke-	Skenario	Master	Slave	Waktu eksekusi
1	*Entry* data	Berhasil	Berhasil	868
2	*Entry* data	Berhasil	Berhasil	1044
3	*Entry* data	Berhasil	Berhasil	188
4	*Entry* data	Berhasil	Berhasil	1404
5	*Entry* data	Berhasil	Berhasil	468
6	Hapus data	Berhasil	Berhasil	1392
7	Hapus data	Berhasil	Berhasil	1276
8	Hapus data	Berhasil	Berhasil	84
9	Hapus data	Berhasil	Berhasil	928
10	Hapus data	Berhasil	Berhasil	1852
Rata-rata				950,4

3.3. Pengujian fungsi *switching*

Pengujian ini bertujuan untuk mengetahui kemampuan sistem dalam melakukan *switching* ketika terjadi *fault* pada sistem. Proses *switching* pada program dilakukan ketika kontroler *slave* mendeteksi adanya *fault* pada kontroler *master* Fault pada master itu sendiri didefinisikan menjadi beberapa kondisi antara lain *error* pada RFID *reader master*, *error* pada mikrokontroler *master*, lalu *error* pada RFID *reader* sekaligus mikrokontroler *master*. Pada kasus pertama, pengujian dilakukan dengan memberikan *infinite loop* pada blok program kontroler *master* agar *master* tidak dapat mengirim *probe message* kepada *slave*, sehingga *slave* akan menganggap bahwa telah terjadi error pada *master*. Berikut hasil pengujian sistem dengan skenario *infinite loop*.

Tabel 3. Daftar hasil pengujian *switching* dengan skenario *infinite loop*

Uji ke-	Fail over	Fungsi Sistem
1	Berhasil	Normal
2	Berhasil	Normal
3	Berhasil	Normal
4	Berhasil	Normal
5	Berhasil	Normal
6	Berhasil	Normal
7	Berhasil	Normal
8	Berhasil	Normal
9	Berhasil	Normal
10	Berhasil	Normal

Tabel 3 memperlihatkan bahwa dari 10 kali pengujian, sistem mampu melakukan mekanisme *switching* berupa *failover* dari kontroler *master* ke kontroler *slave* ketika terjadi *fault* pada kontroler *master*.

Kemudian pada skenario *fault* dimana terjadi *error* pada mikrokontroler *master*, dan *error* pada RFID *reader* sekaligus mikrokontroler *master* dilakukan pengujian fungsi switching dengan mematikan daya pada kontroler *master* ketika sistem sedang beroperasi untuk dilakukan pemeriksaan apakah sistem mampu melakukan *switching* ke kontroler *slave*. Kemudian kontroler *master* kembali dinyalakan sembari dilakukan pemeriksaan apakah sistem mampu melakukan *switchback* ke kontroler *master*. Berikut hasil pengujian dari fungsi *switching* sistem.

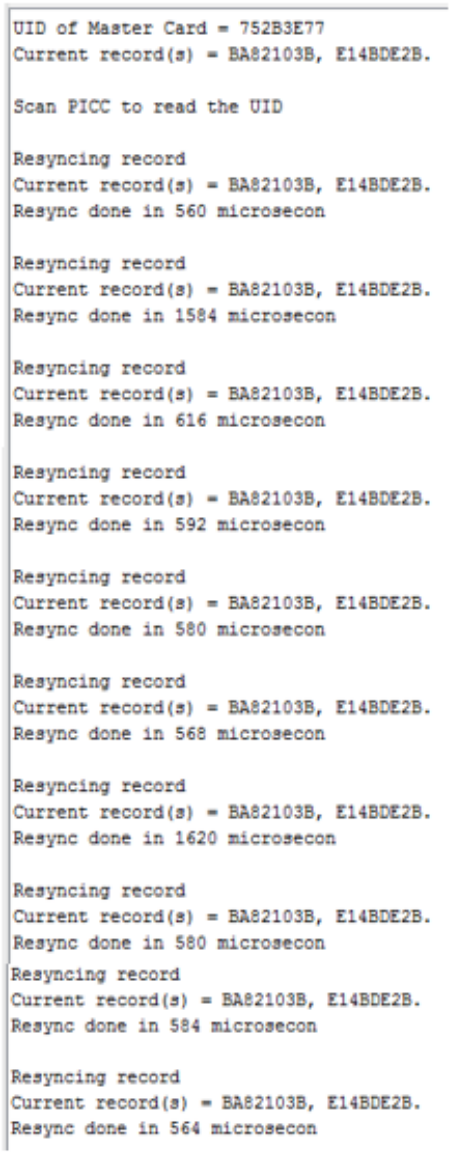
Tabel 4. Daftar hasil pengujian *switching* dengan skenario *power off*

Uji ke-	Fail over	Fail back	Fungsi Sistem
1	Berhasil	Berhasil	Normal
2	Berhasil	Berhasil	Normal
3	Berhasil	Berhasil	Normal
4	Berhasil	Berhasil	Normal
5	Berhasil	Berhasil	Normal
6	Berhasil	Berhasil	Normal
7	Berhasil	Berhasil	Normal
8	Berhasil	Berhasil	Normal
9	Berhasil	Berhasil	Normal
10	Berhasil	Berhasil	Normal

Tabel 4 memperlihatkan bahwa dari 10 kali pengujian, sistem mampu melakukan mekanisme *switching* berupa *failover* dan *failback* dari kontroler *master* ke kontroler *slave* ketika terjadi *fault* pada kontroler *master*..

3.4. Pengujian *Resync* data

Pengujian ini dilakukan untuk mengetahui seberapa lama waktu yang diperlukan bagi sistem untuk melakukan sinkronisasi ulang data dari *slave controller* ke *master controller*. Pengujian dilakukan dengan menyalakan *master controller* setelah sebelumnya berada pada posisi *off*. Gambar 9 berikut memperlihatkan tampilan serial monitor pada pengujian waktu *resync* data pada sistem.



Gambar 9. Tampilan serial monitor fungsi *resync* data

Gambar 9 diatas memperlihatkan *slave* mendeteksi *master* berada pada kondisi offline. Kemudian ketika *slave* mendeteksi *master* kembali berada pada posisi on, *slave* akan mengirim semua datanya ke *master*. Proses sinkronisasi ulang tersebut dilakukan sebanyak 10 kali. Rata-rata waktu yang diperlukan sistem untuk melakukan sinkronisasi ulang data yaitu 784,8 mikrodetik.

Tabel 4. Daftar hasil pengujian waktu *resync* data (mikrodetik)

Uji ke-	Skenario	Slave	Waktu eksekusi
1	*Master reboot*	Berhasil	560
2	*Master reboot*	Berhasil	1584
3	*Master reboot*	Berhasil	616
4	*Master reboot*	Berhasil	592
5	*Master reboot*	Berhasil	580
6	*Master reboot*	Berhasil	568
7	*Master reboot*	Berhasil	1620
8	*Master reboot*	Berhasil	580
9	*Master reboot*	Berhasil	584
10	*Master reboot*	Berhasil	564
	Rata-rata		784,8

4. KESIMPULAN

Penelitian ini mengusulkan penerapan *mirroring* data pada metode *Hot Standby Redundancy* guna meningkatkan reliabilitas sistem. Penelitian ini menggunakan komunikasi serial I^2C sebagai jalur komunikasi data antara perangkat *master* dengan perangkat *slave* dalam sebuah sistem redundan. Fitur *data mirroring* yang diimplementasikan meliputi fitur registrasi data dan fitur sinkronisasi data antara perangkat *master* dengan perangkat *slave*. Kedua fitur tersebut dikombinasikan dengan fitur *realtime monitoring* yang terdapat pada metode redundansi *Hot Standby* agar menghasilkan sebuah sistem yang reliabel ketika terjadi *fault*. Metode *data mirroring* serta *Hot Standby Redundancy* tersebut diimplementasikan pada prototipe sistem kunci pintu otomatis menggunakan mikrokontroler Arduino UNO, RFID, dan kunci pintu solenoid. Data yang diproses pada sistem ini berupa data *tag* RFID yang berfungsi untuk membuka kunci pintu solenoid pada perangkat prototipe *embedded system*. Berdasarkan hasil pengujian, sistem berhasil melakukan registrasi data pada perangkat dengan rata-rata waktu 950,4 mikrodetik. Sistem juga berhasil melakukan sinkronisasi data dengan rata-rata waktu 784,8 mikrodetik.

Saran untuk pengembangan sistem yang dapat dilakukan pada penelitian selanjutnya adalah penambahan komponen cadangan pada sistem untuk semakin meningkatkan kehandalan sistem dan penerapan sistem yang lebih *user-friendly* untuk memudahkan pengguna dalam melakukan monitoring ataupun pengecekan sistem.

5. DAFTAR PUSTAKA

Ablondi, W. (2014, Mei 28). *2014 Smart Home Systems and Services Forecast Global Total.* Dipetik Agustus 27, 2018, dari https://www.strategyanalytics.com/access-services/devices/connected-home/smart-home/reports/report-detail/2014-smart-home-systems-and-services-forecast-global-total

Anwar, N. D., Yovita, L. V., & Mayasari, R. (2015). IMPLEMENTASI DAN ANALISA PERFORMANSI REDUNDANCY PADA JARINGAN. *e-Proceeding of Engineering, II*(3), 7159-7166.

Ariyus, D., & Andri, R. (2008). *Komunikasi Data* (1 ed.). Yogyakarta: Andi.

Dubrova, E. (2013). *Fault-Tolerant Design.* New York: Springer Science+Business Media.

Hemmanur, K. (2009). Inter-Integrated Circuit (I2C). *ECE 480 Journal*, 8.

Hord, M. (2014). *Inter Integrated Circuit (I2C)*. Diambil kembali dari https://learn.sparkfun.com/tutorials/i2c

IEEE. (2009). *An introduction to I2C and SPI protocols*. Dipetik 08 28, 2018, dari http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4762946&isnumber=4762936

Lubis, A. A. (2012). Perancangan Error Detection System And Error. *Jurnal USU*, 1-2.

Muzaki, M. R. (2018). Implementasi Hardware Redundancy Pada Switching Pintu Otomatis Dengan Metode Cold Standby Dan Watchdog.

Nedelkovski, D. (2017). *How RFID Works and How To Make an Arduino based RFID Door Lock*. Dipetik November 5, 2017, dari https://howtomechatronics.com/tutorials/arduino/rfid-works-make-arduino-based-rfid-door-lock/

Pangestu, A. P. (2018). 2.1.3 Implementasi Hardware Redundancy Pada Sistem

Akuisisi Data Sensor Dengan Menggunakan Metode Hot Standby Sparing. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, 2*.

Patnaikuni, D. R. (2017). A Comparative Study of Arduino, Raspberry Pi and ESP8266 as IoT Development. *International Journal of Advanced Research in Computer Science, VIII*(5), 2350-2352.

Ramamoorthy. (1971). Fault-Tolerant Computing: An Introduction and an Overview. *IEEE Transactions on Computers*, 1241 - 1244.

Rouse, M. (2017). *Radio Frequency Identification*. Diambil kembali dari http://internetofthingsagenda.techtarget.com/definition/RFID-radio-frequency-identification

Sorin, D. (2009). *Fault Tolerant Computer Architecture*. Morgan & Claypool.

Talal, B. K., & Rachid, M. (2013). Service Discovery - A Survey and Comparison. *International Journal of UbiComp (IJU), IV*(3), 23-39.