Laporan Tugas 1

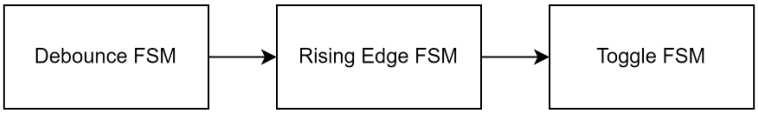
Nama : Elkhan Julian Brillianshah

NIM : 13219059

1 Lampu Toggle

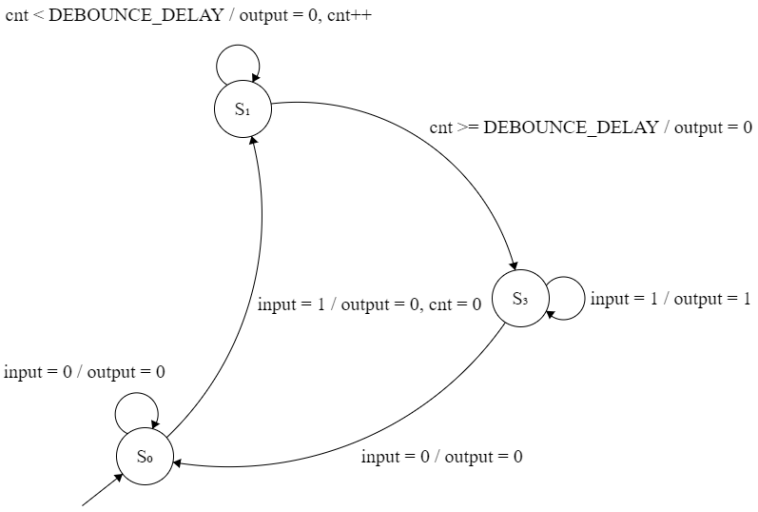
1.1 Desain FSM

FSM yang dapat digunakan untuk membuat lampu *toggle* adalah *cascaded FSM* yang terdiri dari tiga FSM yang saling terhubung. Ketiga FSM tersebut adalah *debounce* FSM untuk melakukan *debouncing*, *rising edge* FSM untuk mendeteksi *rising edge*, dan *toggle* FSM untuk melakukan *toggle*. *Rising Edge* FSM dan *Toggle* FSM merupakan FSM biasa sedangkan *Debounce* FSM adalah *extended* FSM yang menggunakan variabel cnt. Diagram blok *cascaded* FSM adalah seperti pada gambar berikut.



Gambar 1.1 Diagram Blok *Cascaded* FSM

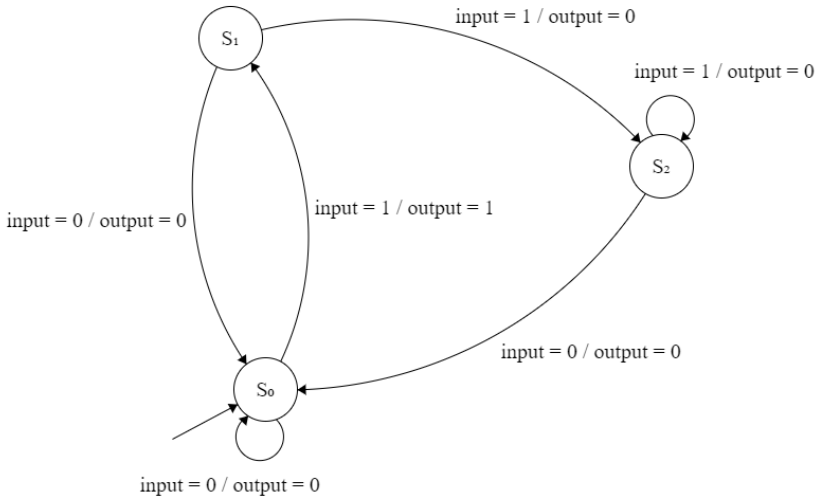
Diagram dan penjelasan dari *debounce* FSM, *rising edge* FSM, dan *toggle* FSM adalah seperti pada gambar-gambar dan paragraf-paragraf berikut.



Gambar 1.2 Diagram *Debounce* FSM

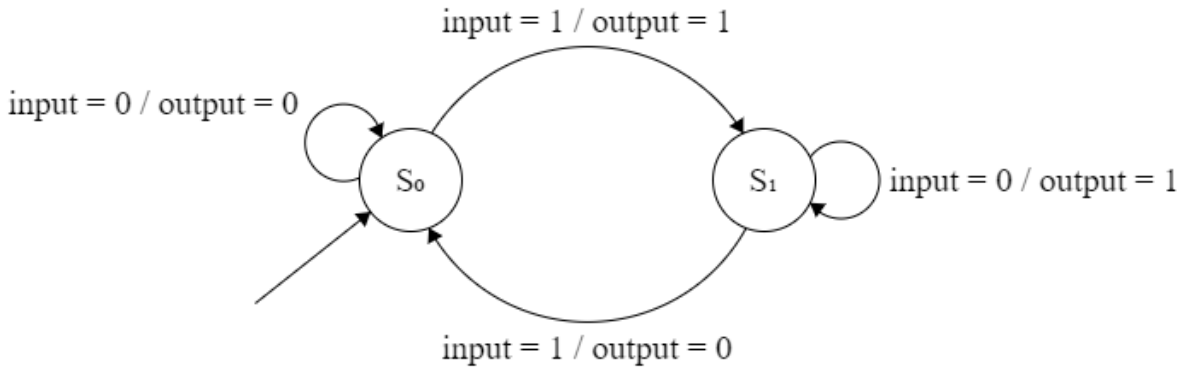
Debounce FSM memiliki 3 state yaitu S_0, S_1, dan S_2 yang secara berurutan memiliki alias IDLE_STATE, WAITING_STATE, dan READING_STATE. IDLE_STATE adalah *state* yang merepresentasikan keadaan awal dan keadaan ketika tidak ada tekanan tombol dari pengguna. WAITING_STATE adalah *state* yang merepresentasikan keadaan ketika tombol baru saja ditekan dan menghasilkan sinyal *bouncing*. READING_STATE adalah *state* yang merepresentasikan keadaan setelah sinyal *bouncing* sudah tidak ada dan pembacaan keadaan tombol dapat dilakukan. Selain itu, FSM ini memiliki satu masukkan yaitu input, satu keluaran yaitu output, dan satu variabel yaitu cnt. Masukkan input memodelkan sinyal dari tombol

yang menjadi masukkan FSM sedangkan keluaran output adalah sinyal yang merepresentasikan hasil *debouncing* dari sinyal input. Variabel cnt digunakan untuk merepresentasikan *state-state* lain yang digunakan untuk membuat jeda yang dibutuhkan untuk menunggu *bouncing* tombol.



Gambar 1.3 Diagram *Rising Edge* FSM

Rising Edge FSM memiliki 3 state yaitu S_0, S_1, dan S_2 yang secara berurutan memiliki alias LOW_STATE, EDGE_STATE, dan HIGH_STATE. LOW_STATE adalah *state* yang merepresentasikan keadaan awal dan keadaan ketika masukkan bernilai 0. EDGE_STATE adalah *state* yang merepresentasikan keadaan ketika masukkan baru saja berubah nilai dari 0 menjadi 1. HIGH_STATE adalah *state* yang merepresentasikan keadaan ketika nilai masukkan terus bernilai 1 setelah EDGE_STATE. FSM ini memiliki satu masukkan yaitu input dan satu keluaran yaitu output. Masukkan input memodelkan sinyal dari *Debounce* FSM yang menjadi masukkan FSM ini sedangkan keluaran output adalah sinyal yang merepresentasikan hasil deteksi *rising edge* dari sinyal input.



Gambar 1.4 Diagram *Toggle* FSM

Toggle FSM memiliki 2 state yaitu S_0 dan S yang secara berurutan memiliki alias OFF_STATE dan ON_STATE. OFF_STATE adalah *state* yang merepresentasikan keadaan awal dan keadaan ketika lampu

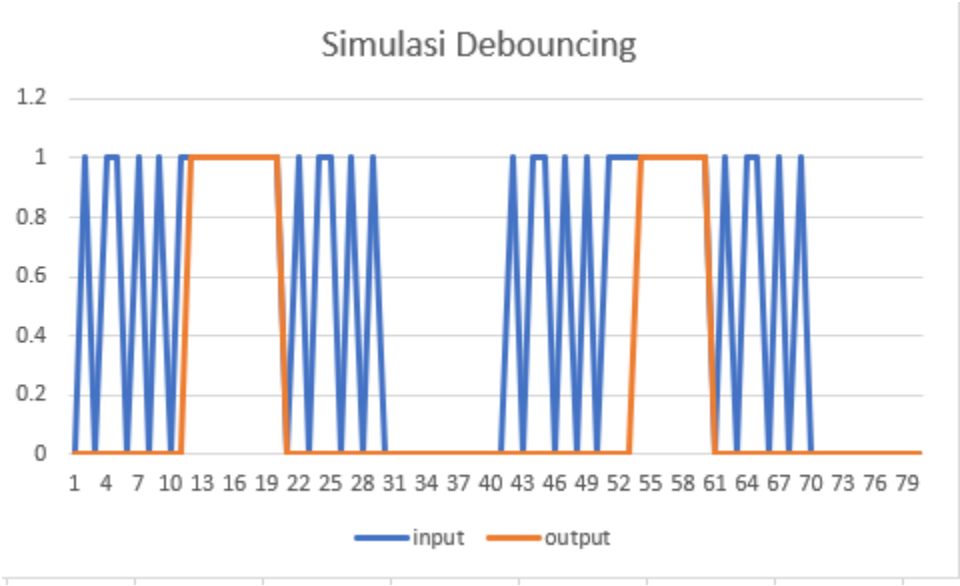
LED seharusnya mati. ON_STATE adalah kebalikan dari OFF_STATE dan merepresentasikan keadaan ketika lampu LED seharusnya menyala. FSM ini memiliki satu masukkan yaitu input dan satu keluaran yaitu output. Masukkan input memodelkan sinyal dari *Rising Edge* FSM yang menjadi masukkan FSM ini sedangkan keluaran output adalah sinyal yang merepresentasikan keadaan LED dengan nilai 1 merepresentasikan LED menyala dan nilai 0 merepresentasikan LED mati.

1.2 Implementasi FSM

Ketiga FSM yang dirancang diimplementasikan menggunakan bahasa C dalam bentuk prosedur-prosedur yang dapat digunakan secara terpisah. *Unit test* dan simulasi kemudian dilakukan untuk memastikan bahwa FSM sudah berfungsi dengan benar sebelum diimplementasikan di ESP32. Source code lengkap untuk implementasi FSM-FSM yang digunakan pada tugas ini dapat dilihat di GitHub.

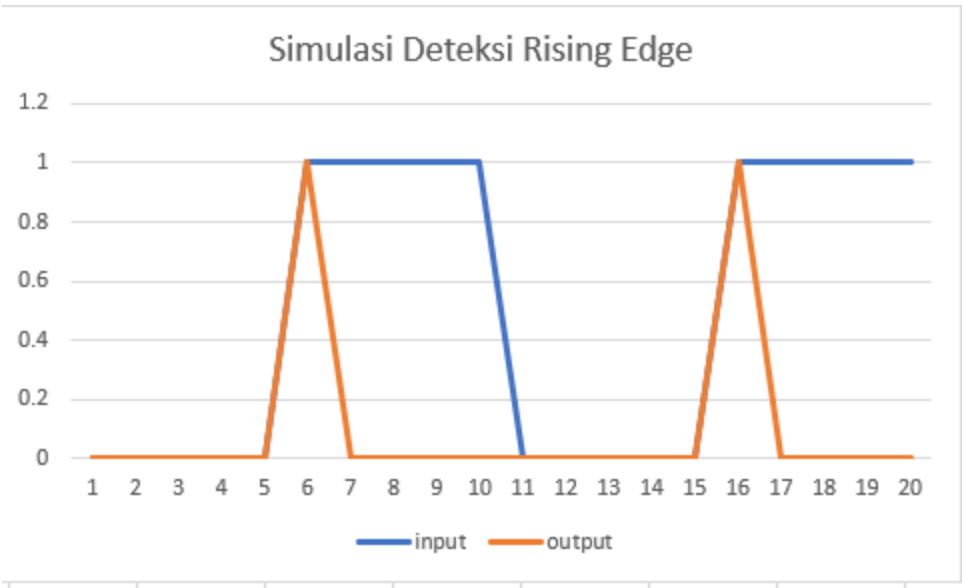
1.2.1 Simulasi Desktop

Untuk memastikan bahwa FSM yang dibuat bekerja sesuai dengan kebutuhan, dilakukan sebuah simulasi menggunakan bahasa C yang akan memberikan masukkan ke FSM dan membaca keluaran FSM dalam satu atau beberapa *loop*. Program simulasi akan mencetak nilai masukkan dan keluaran di terminal komputer. Hasil yang didapat kemudian akan di-*plot* menggunakan MS Excel. Hasil-hasil simulasi adalah seperti pada gambar-gambar berikut.



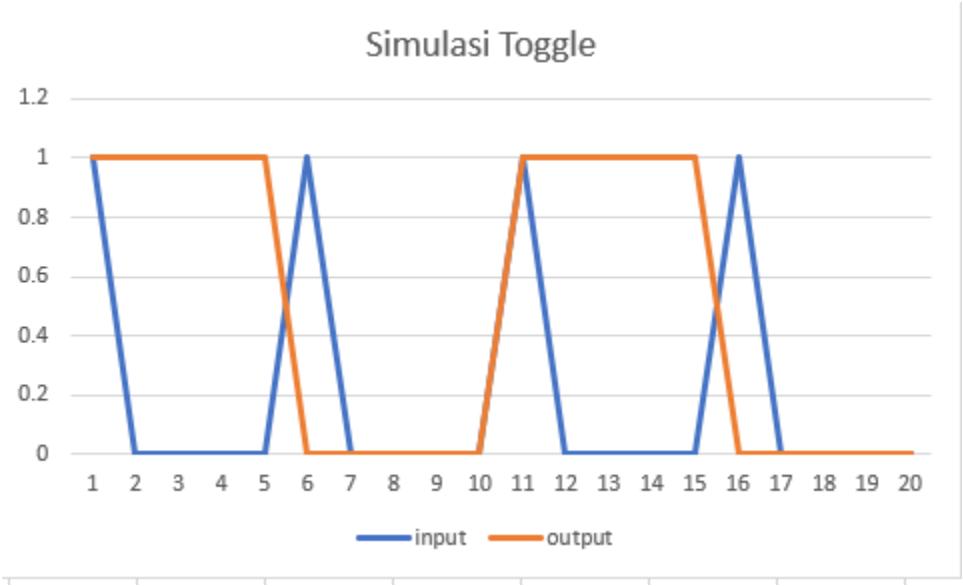
Gambar 1.5 Hasil Simulasi Debouncing

Pada simulasi ini, digunakan nilai DEBOUNCE_DELAY 10, namun nilai tersebut dapat diubah untuk kebutuhan yang berbeda. Dengan itu, didapatkan hasil pada gambar 1.5 yang menunjukkan bahwa *debounce* FSM mampu mengabaikan sinyal *bouncing* yang disimulasikan.



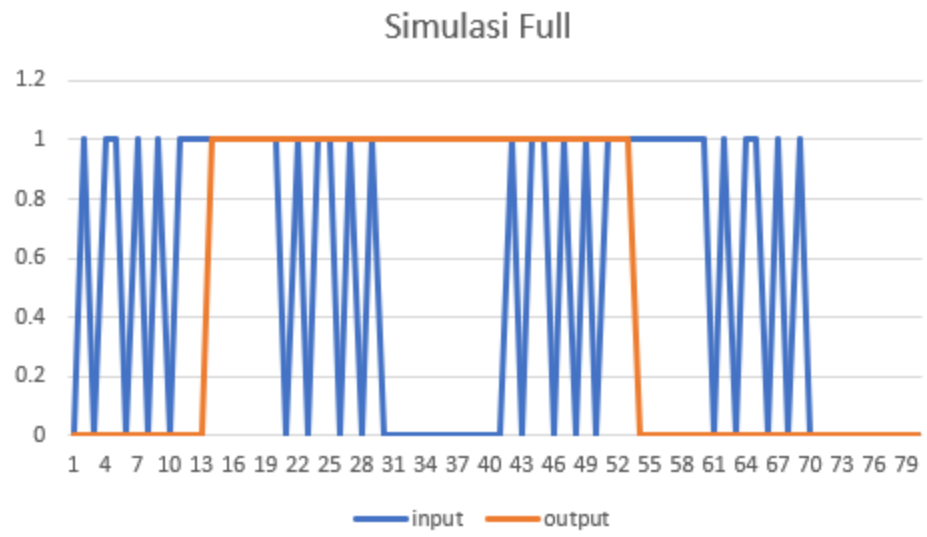
Gambar 1.6 Hasil Simulasi Deteksi Rising Edge

Simulasi pada gambar 1.6 dibuat dengan masukkan berupa sinyal yang dihasilkan oleh *debounce* FSM. Dengan itu, didapatkan bahwa *rising edge* FSM mampu melakukan deteksi *rising edge* dengan benar.



Gambar 1.7 Hasil Simulasi Logika Toggle

Simulasi pada gambar 1.7 dibuat dengan masukkan berupa sinyal yang dihasilkan oleh *rising edge* FSM. Dengan itu, didapatkan bahwa *toggle* FSM mampu melakukan deteksi *toggling* dengan benar jika diberi masukkan hasil deteksi *rising edge*.



Gambar 1.8 Hasil Simulasi FSM Gabungan

Gambar 1.8 merupakan hasil simulasi gabungan ketiga FSM sebelumnya. Masukkan simulasi merupakan simulasi sinyal yang dihasilkan oleh tombol dan keluaran dari simulasi merupakan keluaran FSM berdasarkan masukkan yang diberikan. Dengan itu, dapat diamati bahwa FSM mampu menerima masukkan berupa tekanan tombol yang mengalami *bouncing* dan memberikan keluaran yang sesuai dengan logika toggle yang diinginkan.

1.2.2 Unit Test di Desktop

Unit testing dilakukan untuk menguji kesesuaian nilai output serta *next state* yang diberikan oleh FSM dengan nilai masukkan, *state*, dan nilai variabel tertentu. Pengujian ini dilakukan dengan cara membuat sebuah program bahasa C. Pada program tersebut, terdapat sebuah prosedur yang akan menerima masukkan variabel check yang diperlakukan sebagai sebuah *boolean*. Variabel check akan diisi dengan condisi yang diharapkan dari FSM dengan nilai masukkan, *state,* dan variabel tertentu. Jika nilai check adalah *true,* maka prosedur tersebut akan mencetak “ok”. Hasil *unit test* untuk ketiga FSM adalah seperti pada potongan-potongan terminal berikut berikut.

```
gcc -o debounce-fsm-test ./test/debounce_fsm_test.c ./fsm/debounce_fsm.c
ok
ok
ok
ok
ok
ok
ok
```

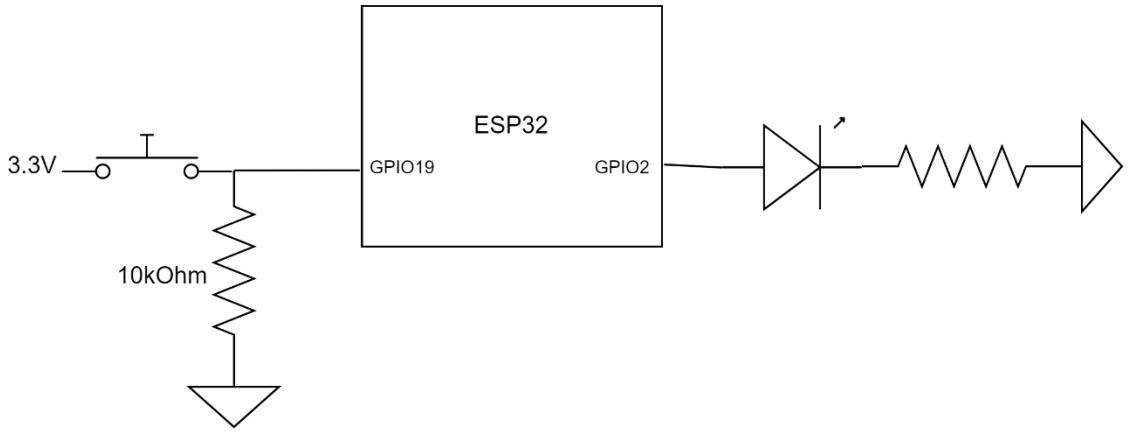
```
gcc -o rising-edge-fsm-test ./test/rising_edge_fsm_test.c ./fsm/rising_edge_fsm.c
ok
ok
ok
ok
ok
ok
```

```
gcc -o toggle-fsm-test ./test/toggle_fsm_test.c ./fsm/toggle_fsm.c
ok
ok
ok
ok
```

Dapat diamati bahwa semua *unit test* yang dilakukan memberikan hasil "ok" yang berarti transisi FSM memiliki perilaku yang sesuai dengan desain FSM.

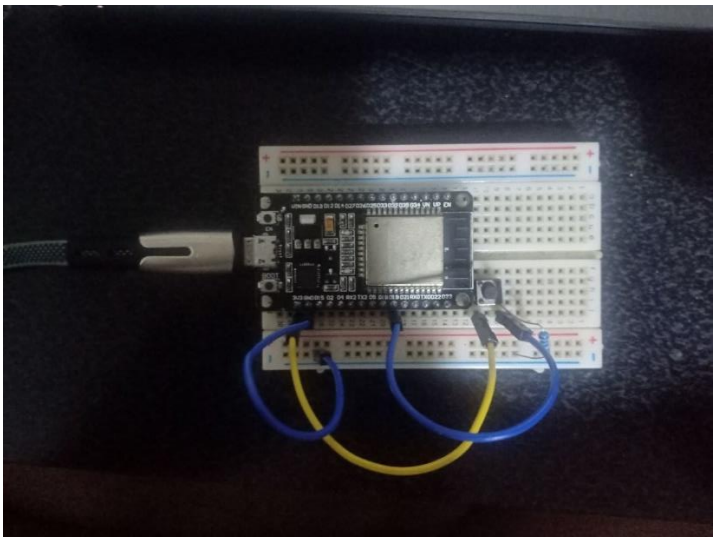
1.2.3 Implementasi di ESP32

Setelah melakukan *unit test* dan simulasi, FSM yang dibuat kemudian diimplementasikan di ESP32. Rangkaian yang digunakan untuk implementasi adalah seperti pada gambar berikut.



Gambar 1.9 Rangkaian Implementasi FSM di ESP32

Pada implementasi ini, digunakan sebuah *pushbutton* yang terhubung ke pin GPIO19 dengan konfigurasi *pull-down*. Lampu LED yang digunakan pada implementasi ini adalah LED *built in* yang terhubung ke GPIO2 di ESP32. Pemrograman ESP32 dilakukan dengan menggunakan kode-kode FSM yang telah dibuat sebelumnya, *library* FreeRTOS, dan *tools* ESP-IDF. Program RTOS bekerja dengan satu task yang dijadwalkan untuk dieksekusi setiap 10ms. Nilai DEBOUNCE_DELAY yang digunakan pada implementasi ini adalah 2 sehingga akan memberikan durasi 20ms untuk menunggu selesainya *bouncing pushbutton*. Bentuk fisik implementasi FSM di ESP32 adalah seperti pada gambar berikut.



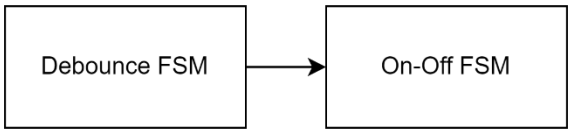
Gambar 1.10 Bentuk Fisik Implementasi FSM di ESP32

Video demonstasi implementasi FSM di ESP32 dapat dilihat di YouTube.

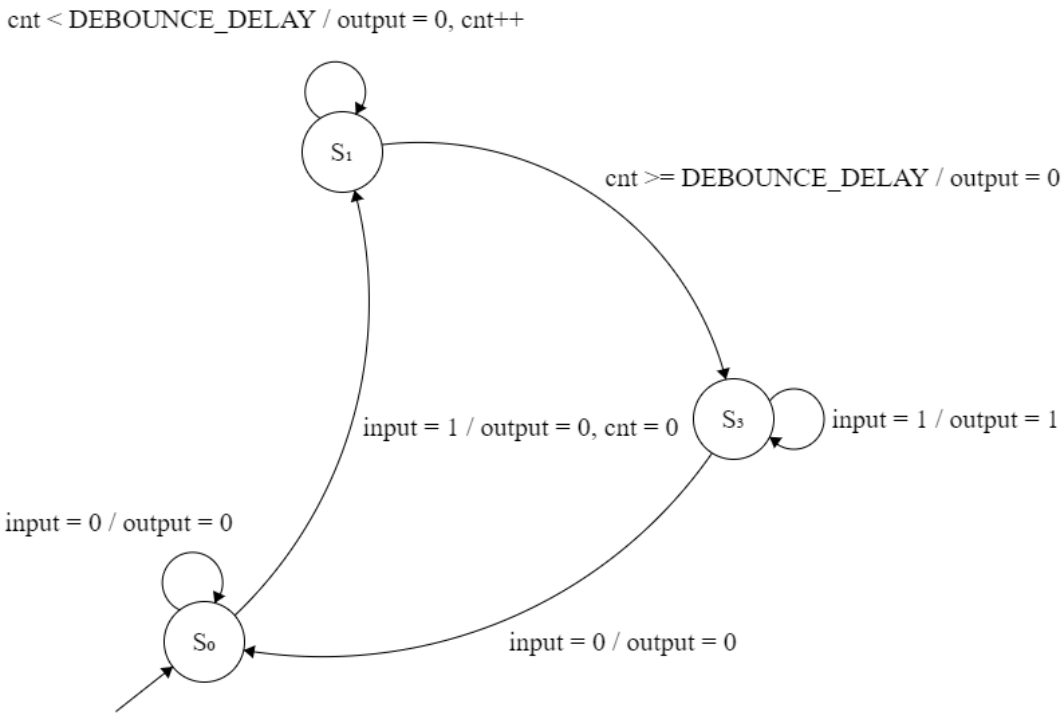
2 Sakelar On Off

2.1 Desain FSM

FSM yang dapat digunakan untuk membuat sakelar on-off adalah *cascaded FSM* yang terdiri dari dua FSM yang saling terhubung. Kedua FSM tersebut adalah *debounce* FSM untuk melakukan *debouncing* dan on-off FSM untuk melakukan logika on-off yang sesuai dengan spesifikasi. Logika on-off yang dimaksud adalah untuk membuat lampu LED menyala jika tombol ditekan secara cepat maupun lama dan membuat lampu LED mati jika tombol ditekan secara lama ketika LED sedang dalam keadaan menyaka. *Debounce* FSM dan on-off FSM adalah *extended* FSM yang menggunakan variabel cnt untuk mengetahui lama jeda diantara perubahan *state-state* tertentu. Diagram blok *cascaded* FSM adalah seperti pada gambar berikut.

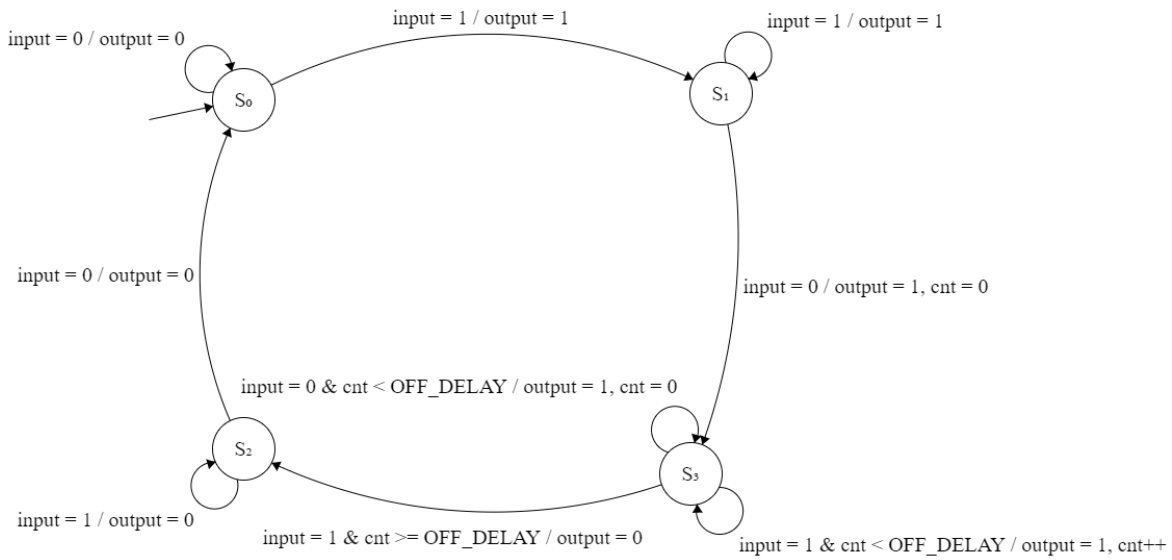


Gambar 2.1 Diagram Blok *Cascaded* FSM



Gambar 2.2 Diagram *Debounce* FSM

Sama seperti sebelumnya, *Debounce* FSM memiliki 3 state yaitu S_0, S_1, dan S_2 yang secara berurutan memiliki alias IDLE_STATE, WAITING_STATE, dan READING_STATE. IDLE_STATE adalah *state* yang merepresentasikan keadaan awal dan keadaan ketika tidak ada tekanan tombol dari pengguna. WAITING_STATE adalah *state* yang merepresentasikan keadaan ketika tombol baru saja ditekan dan menghasilkan sinyal *bouncing*. READING_STATE adalah *state* yang merepresentasikan keadaan setelah sinyal *bouncing* sudah tidak ada dan pembacaan keadaan tombol dapat dilakukan. Selain itu, FSM ini memiliki satu masukkan yaitu input, satu keluaran yaitu output, dan satu variabel yaitu cnt. Masukkan input memodelkan sinyal dari tombol yang menjadi masukkan FSM sedangkan keluaran output adalah sinyal yang merepresentasikan hasil *debouncing* dari sinyal input. Variabel cnt digunakan untuk merepresentasikan *state-state* lain yang digunakan untuk membuat jeda yang dibutuhkan untuk menunggu *bouncing* tombol.



Gambar 2.3 Diagram On-Off FSM

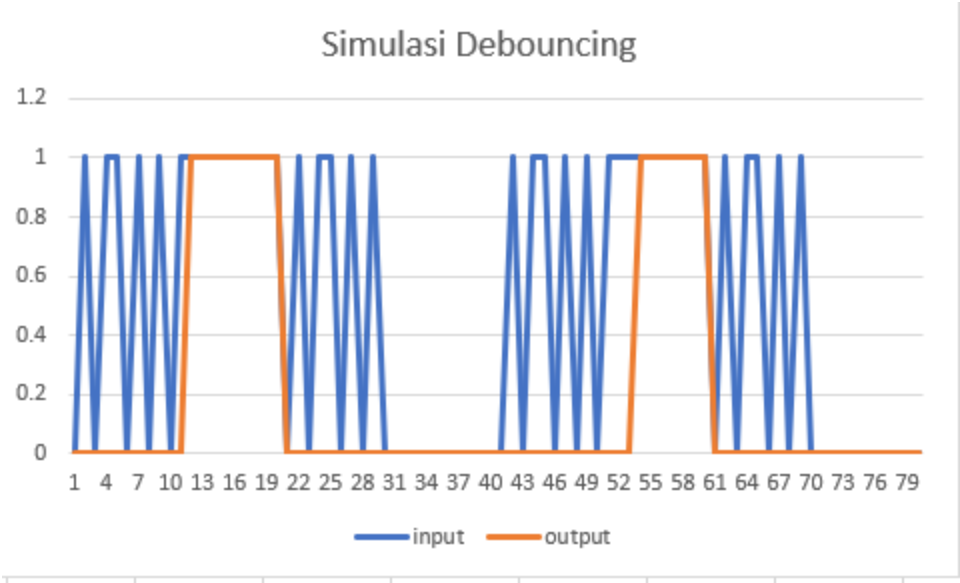
On-off FSM memiliki 4 state yaitu S_0, S_1, S_2, dan S_4 yang secara berurutan memiliki alias OFF_STATE, JUST_ON_STATE, ON_STATE, dan JUST_OFF_STATE. OFF_STATE adalah *state* yang merepresentasikan keadaan awal serta keadaan ketika tidak ada tekanan tombol dari pengguna dan lampu LED sedang dalam keadaan mati. JUST_ON_STATE adalah *state* yang merepresentasikan keadaan ketika tombol baru saja ditekan dan lampu LED baru saja menyala. ON_STATE adalah *state* yang merepresentasikan keadaan ketika lampu LED sudah menyala setelah JUST_ON_STATE. JUST_OFF_STATE adalah *state* yang merepresentasikan keadaan ketika lampu LED baru saja mati setelah tombol ditekan selama beberapa saat oleh pengguna. Selain itu, FSM ini memiliki satu masukkan yaitu input, satu keluaran yaitu output, dan satu variabel yaitu cnt. Masukkan input memodelkan sinyal dari *Debounce* FSM sedangkan keluaran output adalah sinyal yang merepresentasikan hasil keadaan lampu LED yang diinginkan dengan nilai 1 merepresentasikan LED menyala dan nilai 0 merepresentasikan LED mati. Variabel cnt digunakan untuk merepresentasikan *state-state* lain yang digunakan untuk membuat jeda yang dibutuhkan untuk mendeteksi tekanan tombol yang lama.

2.2 Implementasi FSM

Kedua FSM yang dirancang diimplementasikan menggunakan bahasa C dalam bentuk prosedur-prosedur yang dapat digunakan secara terpisah. *Unit test* dan simulasi kemudian dilakukan untuk memastikan bahwa FSM sudah berfungsi dengan benar sebelum diimplementasikan di ESP32. Source code lengkap untuk implementasi FSM-FSM yang digunakan pada tugas ini dapat dilihat di GitHub.

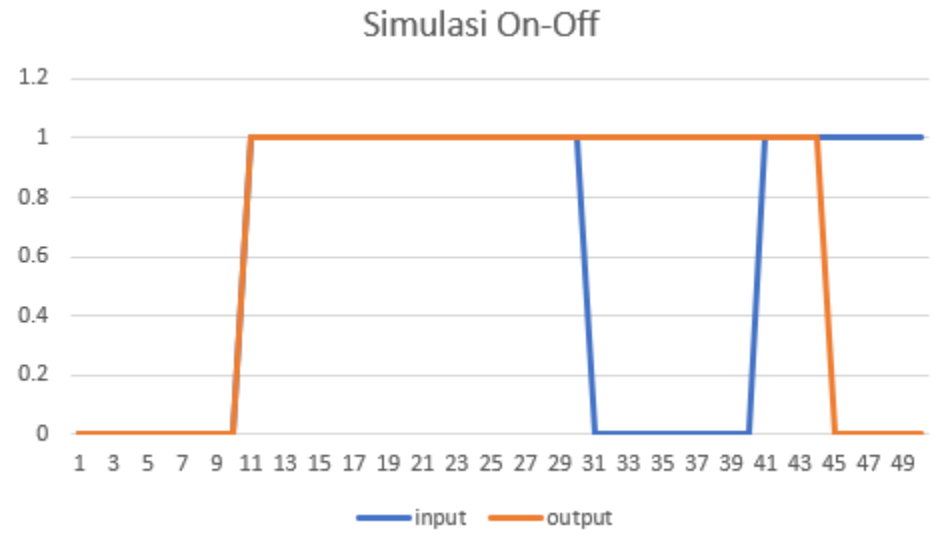
2.2.1 Simulasi Desktop

Sama seperti sebelumnya, dilakukan sebuah simulasi menggunakan bahasa C yang akan memberikan masukkan ke FSM dan membaca keluaran FSM dalam satu atau beberapa *loop*. Program simulasi akan mencetak nilai masukkan dan keluaran di terminal komputer. Hasil yang didapat kemudian akan di-*plot* menggunakan MS Excel. Hasil-hasil simulasi adalah seperti pada gambar-gambar berikut.



Gambar 2.4 Hasil Simulasi Debouncing

Pada simulasi ini, digunakan nilai DEBOUNCE_DELAY 10, namun nilai tersebut dapat diubah untuk kebutuhan yang berbeda. Dengan itu, didapatkan hasil pada gambar 2.4 yang menunjukkan bahwa *debounce* FSM mampu mengabaikan sinyal *bouncing* yang disimulasikan.



Gambar 2.5 Hasil Simulasi Logika On-Off

Pada simulasi ini, digunakan nilai OFF_DELAY 200, namun nilai tersebut dapat diubah untuk kebutuhan yang berbeda. Pada bagian simulasi untuk mematikan LED dari keadaan LED menyala, waktu simulasi per sampel dipercepat untuk memberikan hasil yang lebih mudah untuk diamati. Dengan itu, didapatkan hasil pada gambar 2.5 yang menunjukkan bahwa on-off FSM mampu mengimplementasikan logika on-off yang diinginkan.

2.2.2 Unit Test di Desktop

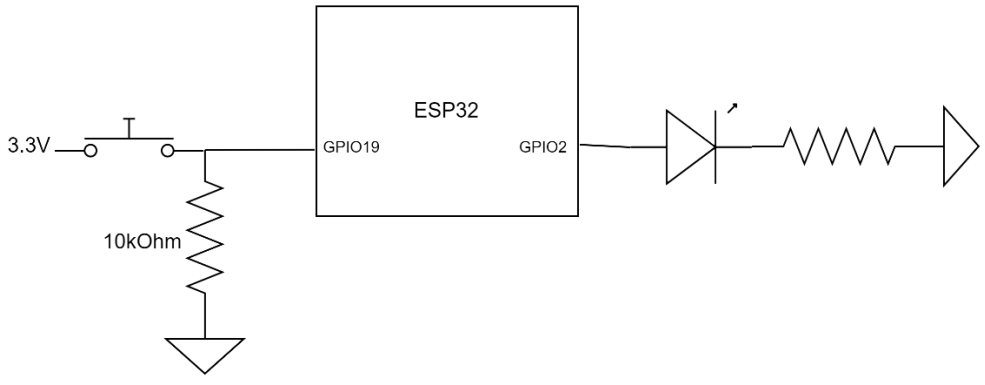
Unit testing dilakukan untuk menguji kesesuaian nilai output serta *next state* yang diberikan oleh FSM dengan nilai masukkan, *state*, dan nilai variabel tertentu. Pengujian ini dilakukan dengan cara membuat sebuah program bahasa C. Pada program tersebut, terdapat sebuah prosedur yang akan menerima masukkan variabel check yang diperlakukan sebagai sebuah *boolean*. Variabel check akan diisi dengan condisi yang diharapkan dari FSM dengan nilai masukkan, *state,* dan variabel tertentu. Jika nilai check adalah *true,* maka prosedur tersebut akan mencetak "ok". Hasil *unit test* untuk kedua FSM adalah seperti pada potongan-potongan terminal berikut berikut.

```
gcc -o debounce-fsm-test ./test/debounce_fsm_test.c ./fsm/debounce_fsm.c
ok
ok
ok
ok
ok
ok
ok
```

```
gcc -o on-off-fsm-test ./test/on_off_fsm_test.c ./fsm/on_off_fsm.c
ok
ok
ok
ok
ok
ok
ok
ok
ok
```

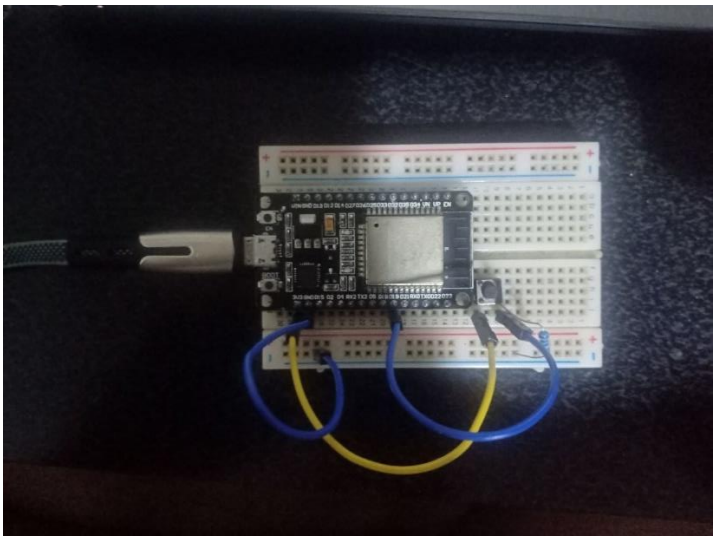
2.2.3 Implementasi di ESP32

Setelah melakukan *unit test* dan simulasi, FSM yang dibuat kemudian diimplementasikan di ESP32. Rangkaian yang digunakan untuk implementasi adalah seperti pada gambar berikut.



Gambar 2.6 Rangkaian Implementasi FSM di ESP32

Pada implementasi ini, digunakan sebuah *pushbutton* yang terhubung ke pin GPIO19 dengan konfigurasi *pull-down*. Lampu LED yang digunakan pada implementasi ini adalah LED *built in* yang terhubung ke GPIO2 di ESP32. Pemrograman ESP32 dilakukan dengan menggunakan kode-kode FSM yang telah dibuat sebelumnya, *library* FreeRTOS, dan *tools* ESP-IDF. Program RTOS bekerja dengan satu task yang dijadwalkan untuk dieksekusi setiap 10ms. Nilai DEBOUNCE_DELAY yang digunakan pada implementasi ini adalah 2 sehingga akan memberikan durasi 20ms untuk menunggu selesainya *bouncing pushbutton*. Nilai OFF_DELAY yang digunakan pada implementasi ini adalah 200 sehingga akan membuat lama tekanan tombol yang dibutuhkan untuk mematikan LED menjadi 2 detik. Bentuk fisik implementasi FSM di ESP32 adalah seperti pada gambar berikut.



Gambar 2.7 Bentuk Fisik Implementasi FSM di ESP32

Video demonstasi implementasi FSM di ESP32 dapat dilihat di YouTube.

3 Lampiran

Link Source Code : https://github.com/ubbeg2000/tugas1-el4121

Link Video Demonstrasi : https://www.youtube.com/watch?v=FajcCW099rw