Makalah Seminar Tugas Akhir

MULTITASKING PADA MIKROKONTROLER ATMEGA16 MENGGUNAKAN *REAL TIME OPERATING SYSTEM* (RTOS) JENIS *COOPERATIVE*

FX Ryan Kurniawan[1], Iwan Setiawan, ST, MT[2], Sumardi, ST, MT[2]
Jurusan Teknik Elektro, Fakultas Teknik, Universitas Diponegoro
Jl. Prof. Sudharto, Tembalang, Semarang, Indonesia

ABSTRAK

*Perkembangan teknologi mikroprosesor yang menawarkan kemudahan-kemudahan dalam rancang bangun sistem teknologi mendorong penerapan teknologi ini dalam hampir semua bidang teknologi. Salah satunya adalah ketersediaan mikrokontroler, sebagai hasil dari perkembangan mikroprosesor, yang memudahkan dan mendorong penerapan teknologi prosesor yang bersifat khusus dalam sistem-sistem sederhana dan kecil. Penerapan yang menanamkan (*embed*) sistem mikrokontroler dalam peralatan sering disebut dengan embedded systems (sistem tertanam). Sistem tertanam biasanya digunakan sebagai komponen inti dari sebuah produk dan dirancang untuk tujuan khusus melakukan satu atau banyak tugas dalam komputasi yang real time. Agar dapat mewujudkan sistem yang dapat bekerja secara real time dibutuhkan sebuah sistem operasi yang khusus diperuntukkan untuk operasi real time yang dikenal dengan nama real time operating system (RTOS). Dengan sistem real time maka sebuah task (proses) dapat diselesaikan dalam kurun waktu tertentu yang bisa ditentukan pada saat proses perancangan. Selain itu terdapat scheduling (penjadwalan) yang memungkinkan pengerjaan beberapa task secara teratur sehingga kemungkinan untuk bertabrakannya beberapa task bisa dihindarkan.*

Tujuan tugas akhir ini untuk memperkenalkan prinsip kerja sebuah RTOS dan kemampuan serta fungsi apa saja yang terdapat di dalamnya. Kemudian mengetahui bagaimana performa jika sebuah RTOS ditanamkan pada mikrokontroler jenis AVR.

Dari hasil pengujian didapatkan hasil bahwa sebuah RTOS bisa ditanamkan pada mikrokontroler jenis AVR dengan tipe ATmega16. Kemudian dapat diamati juga kecepatan dan ketepatan scheduling dan pewaktuan dari RTOS yang digunakan. Tetapi yang paling penting dari sebuah RTOS adalah fungsi dari kernel yang ada pada RTOS itu sendiri.

Kata kunci: *RTOS,* task, scheduling, kernel.

1 PENDAHULUAN

Perkembangan teknologi mikroprosesor yang menawarkan kemudahan-kemudahan dalam rancang bangun sistem teknologi mendorong penerapan teknologi ini dalam hampir semua bidang teknologi. Salah satunya adalah ketersediaan mikrokontroler, sebagai hasil dari perkembangan mikroprosesor, yang memudahkan dan mendorong penerapan teknologi prosesor yang bersifat khusus dalam sistem-sistem sederhana dan kecil. Penerapan yang menanamkan (*embed*) sistem mikrokontroler dalam peralatan sering disebut dengan *embedded systems* (sistem tertanam). Sistem tertanam biasanya digunakan sebagai komponen inti dari sebuah produk dan dirancang untuk tujuan khusus melakukan satu atau banyak tugas dalam komputasi yang *real time*. Sistem waktu nyata atau lebih dikenal dengan *real time system* adalah sistem yang harus menghasilkan respon yang tepat dalam batas waktu yang telah ditentukan. Jika respon yang dihasilkan melewati batas waktu tersebut, maka akan terjadi degradasi performansi atau bahkan kegagalan sistem. Suatu sistem dikatakan *real time* jika memiliki *deadline* / jangka waktu penyelesaian tertentu, namun tetap mengutamakan ketepatan dan performa yang tinggi dalam prosesnya. Agar dapat mewujudkan sistem yang dapat bekerja secara *real time* dibutuhkan sebuah sistem operasi yang khusus diperuntukkan untuk operasi *real time* yang dikenal dengan nama *real time operating system* (RTOS).

Meski sudah diperkenalkan sejak lama namun penggunaan RTOS sendiri baru digunakan secara intensif kira-kira satu dekade ini. Banyak pengembang perangkat lunak yang mengembangkan RTOS, baik yang bersifat komersial atau non-komersial. RTOS yang disediakan oleh para pengembang perangkat lunak hampir bisa digunakan pada semua jenis *micro controller unit* (MCU) mulai dari MCU dengan arsitektur 8 *bit* hingga MCU dengan arsitektur 32 *bit*, seperti AVR, PIC, ARM, x86, ColdFire, Blackfin. Bisa dipastikan bahwa bidang RTOS akan berkembang lebih pesat lagi, oleh karena itu RTOS dipilih sebagai tema tugas akhir ini.

2 *REAL TIME SYTEM* DAN *REAL TIME OPERATING SYSTEM*

Pada awalnya, istilah *real time* digunakan dalam simulasi. Istilah *real time* lazim dimengerti sebagai “cepat”, namun sebenarnya yang dimaksud adalah yang bisa menyamai proses

[1] Mahasiswa Teknik Elektro
[2] Dosen Teknik Elektro pembimbing tugas akhir

sebenarnya di dunia nyata. Sistem waktu nyata atau lebih dikenal dengan *real time system* adalah sistem yang harus menghasilkan respon yang tepat dalam batas waktu yang telah ditentukan. Jika respon yang dihasilkan melewati batas waktu tersebut, maka akan terjadi degradasi performansi atau bahkan kegagalan sistem. Dengan kata lain, *real time system* / sistem waktu nyata adalah sistem yang memiliki *deadline* / jangka waktu penyelesaian tertentu, namun tetap mengutamakan ketepatan dan performa yang tinggi dalam prosesnya. Ada dua model *real time system*, yaitu *hard real time system* dan *soft real time system.*

Hard real time system mewajibkan proses selesai dalam kurun waktu tertentu. Jika tidak selesai dalam kurun waktu yang telah ditetapkan, maka sistem dianggap gagal. Kegagalan pada model sistem ini menimbulkan efek berbahaya yang dapat merusak sistem secara keseluruhan. *Hard real time system* menjamin bahwa proses waktu nyata dapat diselesaikan dalam batas waktu yang telah ditentukan. Contoh penggunaan model sistem ini adalah pada *safety critical system*. *Hard real time system* biasanya berinteraksi pada tingkat rendah dengan *hardware* fisik.

Soft real time system tidak memberlakukan aturan waktu seketat *hard real time system*. Sistem ini menerapkan adanya prioritas dalam pelaksanaan tugas dan menjamin suatu proses terpenting mendapat prioritas tertinggi untuk diselesaikan diantara proses-proses lainnya. Kegagalan pada model sistem ini tidak menimbulkan efek yang berbahaya hanya saja akan menyebabkan degradasi performansi pada sistem. Aplikasi yang telah habis masa pengerjaan tugasnya akan dihentikan secara bertahap atau dengan kata lain masih diberikan toleransi waktu.

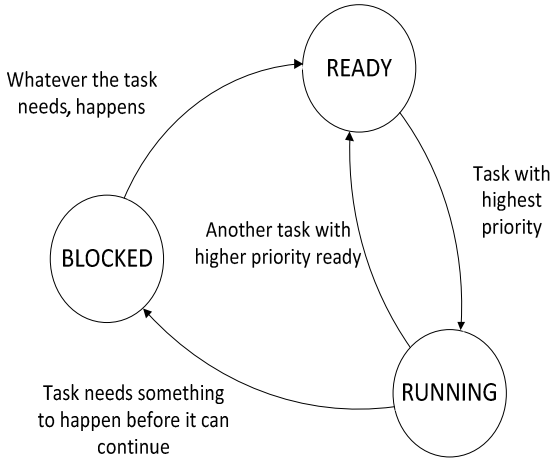
*Real time operating syste*m adalah suatu sistem operasi yang digunakan untuk memfasilitasi pembentukan sebuah *real time system*. Dalam mempelajari *real time operating system* ada beberapa hal yang perlu diketahui, seperti *task*, *kernel*, *mutual exclusion*, dan lainnya.

2.1 *Task*

Sebuah *task*, merupakan sekumpulan instruksi sederhana yang dapat diibaratkan memiliki CPU untuk *task* itu sendiri. Salah satu proses perancangan aplikasi *real time* menggunakan RTOS adalah membagi keseluruhan tugas pada aplikasi tersebut menjadi beberapa *task* dimana tiap-tiap *task* mempunyai tanggung jawab spesifik dalam penyusunan aplikasi.

Tiap *task* merupakan *loop* yang akan terus berulang. Dalam proses pengulangan tersebut, *task* akan mengalami tiga buah keadaan seperti pada Gambar 2.1 yaitu:

- *Running*, merupakan keadaan di mana sebuah *task* dengan prioritas tertinggi berjalan
- *Ready*, merupakan keadaan yang dialami sebuah *task* jika terdapat sebuah *task* lain sedang *running* dan *task* yang berada pada *ready* akan melanjutkan pengerjaan *task* yang sempat tertunda oleh *task* yang lebih tinggi prioritasnya.
- *Blocked*, merupakan keadaan di mana jika sebuah *task* membutuhkan *event* atau data maka akan masuk ke dalam *blocked* hingga *event* atau data yang dibutuhkan telah tersedia.



Gambar 2.1 Siklus *state* pada sebuah RTOS

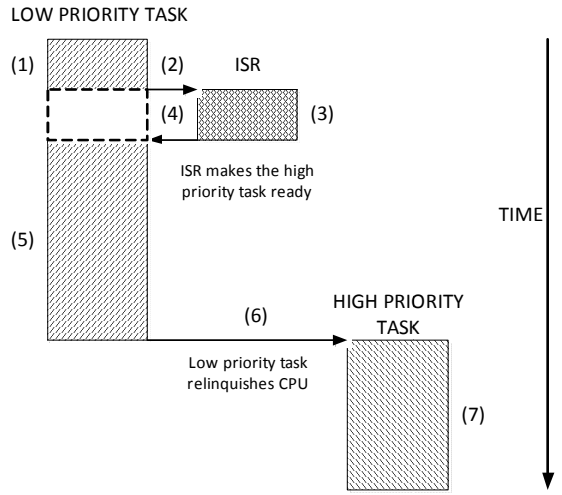
2.2 *Kernel*

Kernel merupakan salah satu bagian dari sistem *multitasking* yang mempunyai fungsi sebagai manajemen dari seluruh *task*, mengatur komunikasi antar *task* dan yang terpenting adalah mengatur pewaktuan untuk CPU sehingga tidak terjadi *crash*. Layanan standar yang disediakan oleh *kernel* adalah *context switching*. Penggunaan *kernel* akan menyederhanakan perancangan sistem dengan cara membagi aplikasi ke dalam beberapa *task* yang dikelola oleh *kernel*.

2.2.1 *Non-preemptive Kernel*

Non-preemptive kernel biasa dikenal dengan nama lain *cooperative kernel*, di mana *task* bekerja sama satu sama lain untuk berbagi CPU. ISR bisa membuat sebuah *task* dengan prioritas tertinggi menjadi siap untuk dieksekusi, tetapi kemudian ISR akan kembali ke *task* yang sebelumnya mendapat interupsi. *Task* yang sudah siap tadi akan berjalan apabila *task* yang mendapat interupsi tadi sudah selesai berjalan atau dengan kata lain task yang sudah selesai berjalan akan menyerahkan CPU kepada *task* dengan prioritas tertinggi (seperti konsep pada lari estafet,

di mana pelari sebelumnya menyerahkan tongkat estafet kepada pelari selanjutnya).

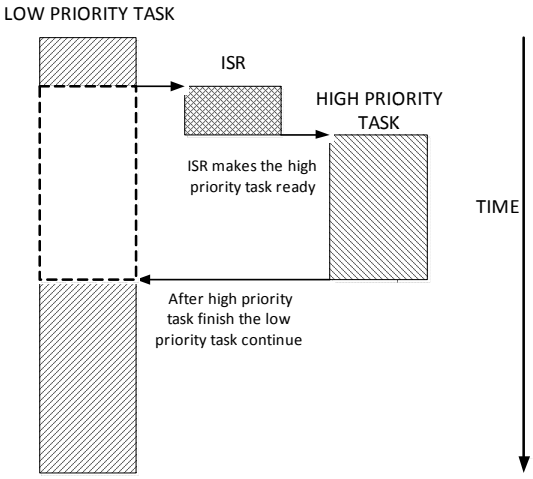


Gambar 2.2 Skema prinsip kerja *non-preemptive kernel*

Dari penjelasan di atas dapat diambil kesimpulan, bahwa *non-preemptive kernel* menjalankan *task* berurutan sehingga tidak akan terjadi tabrakan antar *task*. Hal ini dikarenakan untuk menjalankan tiap *task* dibutuhkan CPU dan CPU hanya bisa didapat apabila *task* sebelumnya sudah selesai melakukan tugasnya.

2.2.2 *Preemptive Kernel*

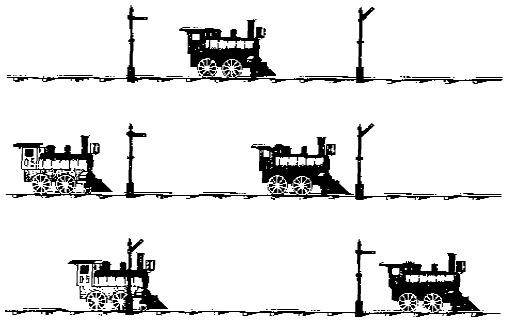
Gambar 2.3 menjelaskan prinsip kerja dari *preemptive kernel*. Di mana *task* dengan prioritas tertinggi yang sudah siap dieksekusi akan langsung berjalan. Jika pada saat itu sedang ada *task* dengan prioritas yang lebih rendah berjalan maka *task* dengan prioritas rendah tersebut akan ditunda. Jadi dapat disimpulkan bahwa *preemptive kernel* selalu mendahulukan *task* dengan prioritas tertinggi yang siap untuk dieksekusi.



Gambar 2.3 Skema prinsip kerja *preemptive kernel*

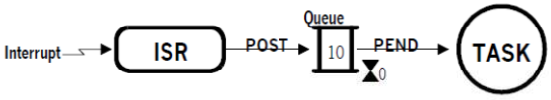
2.3 *Mutual Exclusion*

Mutual exclusion adalah suatu kondisi dimana setiap *resources* / sumber daya diberikan tepat pada suatu proses pada suatu waktu. Ada beberapa mekanisme yang dapat digunakan untuk mendukung *mutual exclusion* yaitu *semaphore* dan *message queue*.



Gambar 2.4 Visualisasi dari cara kerja *semaphore*

Semaphore berguna pada saat terdapat dua atau lebih *task* yang ingin menggunakan data atau *resource* yang sama.



Gambar 2.5 Skema penggunaan *message queue*

Message queue digunakan untuk mengirim satu atau lebih pesan kepada *task*. *Semaphore* dan *message queue* digunakan untuk mencegah terjadinya *error* karena *shared resource* atau *shared data*.

2.4 *Clock Tick*

Clock tick merupakan interupsi spesial yang muncul secara periodik. *Clock tick* bisa dianggap sebagai detak jantung dari sistem yang berfungsi sebagai dasar untuk menentukan *timer* pada sistem *real time* dengan RTOS. Waktu untuk tiap munculnya *clock tick* bisa ditentukan pada saat merancang sistem RTOS. Semakin cepat *clock tick*, semakin besar beban yang ditanggung oleh CPU.

Semua *kernel* mampu untuk menunda *task* sesuai dengan nilai dari *clock tick*. Tetapi sering juga terjadi kasus di mana penundaan tidak sesuai dengan waktu yang diinginkan.

2.5 CocoOS

CocoOS adalah RTOS yang bersifat *open source*, tidak berbayar, dan merupakan *cooperative task scheduler* berdasarkan *coroutines* yang diperuntukkan bagi mikrokontroler tertanam seperti AVR dan MSP430. Konfigurasi cocoOS dilakukan dengan cara mendefinisikan 4 konstanta yang berada pada *header* `os_defines.h`. 4 konstanta itu adalah N_TASKS, N_QUEUES, N_SEMAPHORES, N_EVENTS. Rentang nilai yang diperbolehkan untuk masing-masing konstanta adalah 0 – 254. Konstanta-konstanta tersebut menyatakan nilai maksimum untuk

jumlah objek yang digunakan dengan tipe tertentu.

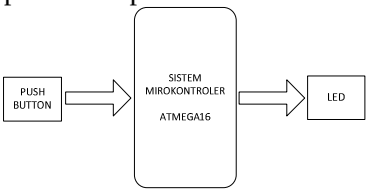
Ada beberapa hal yang harus diperhatikan dalam pengerjaan *application main function* pada cocoOS. Pada permulaan *main function* / fungsi utama selalu diawali dengan pengaturan sistem seperti *ports*, *clock*, dan lain-lain. Setelah itu dilanjutkan dengan pengaturan *kernel* dari cocoOS yang diaktifkan dengan perintah `os_init()`. Proses dilanjutkan dengan membuat semua *task*, *semaphore*, dan *event* yang akan digunakan. Dan langkah terakhir adalah menjalankan *clock* dan memanggil perintah `os_start()`. CocoOS mengatur pewaktuan dengan menghitung *ticks* yang dilakukan dengan cara memanggil perintah `os_tick()` secara berkala dari *clock tick* ISR.

Setiap aplikasi tersusun oleh beberapa *task*. Dan tiap *task* terhubung dengan sebuah prosedur (bisa saja sederhana) yang memiliki fungsi dan tujuan yang spesifik. Pengeksekusian *task* dikelola oleh *os kernel* dengan cara mempersilakan *task* dengan prioritas tertinggi dieksekusi terlebih dahulu. Setiap *task* setidaknya harus mempunyai satu perintah pemblokiran pada fungsi penjadwalan *kernel*. Ini dimaksudkan agar *task* dengan prioritas lebih rendah mempunyai kesempatan untuk dieksekusi. Ketika sebuah *task* telah selesai dieksekusi maka *task* tersebut akan memberikan kontrol CPU kepada *task* yang lain dengan cara memanggil salah satu perintah penjadwalan yang telah disediakan.

3 PERANCANGAN SISTEM

3.1 Perancangan Perangkat Keras

Perangkat keras pada tugas akhir ini digunakan untuk menguji secara nyata cara kerja dari sistem *real time* yang dirancang. Perancangan perangkat keras dalam tugas akhir ini meliputi modul sistem minimum ATmega16, modul LED dan *push button*. Secara umum perancangan sistem dapat dilihat pada Gambar 3.1.



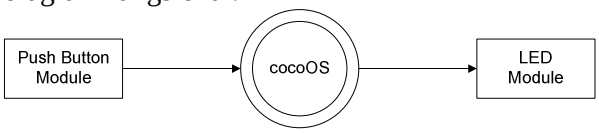
Gambar 3.1 Blok diagram sistem

Tiap-tiap bagian blok dari diagram blok tersebut dapat dijelaskan sebagai berikut:

1. Modul *push button* berfungsi sebagai masukkan untuk MCU.
2. Modul LED berfungsi sebagai keluaran di mana nantinya akan terlihat proses pergerakan *task* dari keseluruhan sistem.
3. Modul sistem minimum Atmega16 merupakan MCU di mana sistem *real time* akan ditanamkan dan semua proses RTOS terjadi di dalam modul ini.

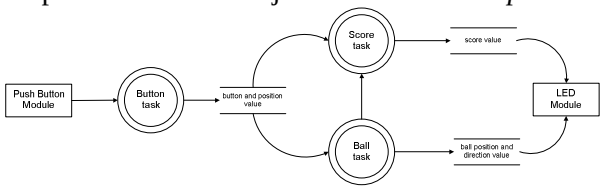
3.2 Perancangan Perangkat Lunak

Sistem yang dibuat pada tugas akhir ini adalah simulasi sederhana pingpong elektronik. Simulasi sederhana pingpong elektronik yang dirancang mempunyai beberapa penjelasan. Sistem tertanam sederhana ini adalah sebuah permainan untuk 2 orang dimana masing-masing pemain mempunyai *push button* sendiri yang merepresentasikan *paddle* dalam sebuah permainan pingpong. Salah satu pemain dapat memulai permainan dengan menekan *paddle* masing-masing. Bola pingpong, yang pergerakannya diwakili oleh 8 LED berjajar, akan berjalan menuju kepada lawan main. Lawan main harus dan hanya boleh menekan *paddle* saat bola pada posisi akhir dari LED. Permainan akan terus berjalan sampai salah satu pemain melanggar peraturan. Ketika itu terjadi maka pemain yang tidak melanggar peraturan akan mendapatkan skor yang akan ditunjukkan dengan menyalanya LED skor yang bersangkutan. Setelah LED skor menyala maka *out of play* LED akan menyala yang menunjukkan bahwa permainan berhenti dan bola tidak ada di lapangan. Untuk membuat sistem seperti yang dijelaskan sebelumnya, pada tugas akhir ini digunakan diagram pendekatan berupa diagram fungsional.



Gambar 3.2 *Context diagram* sistem

Context diagram sistem hanya menggambarkan aliran data dari modul *input* ke dalam MCU yang di dalamnya sudah terdapat cocoOS dan data output dari MCU dilanjutkan ke modul *output*.

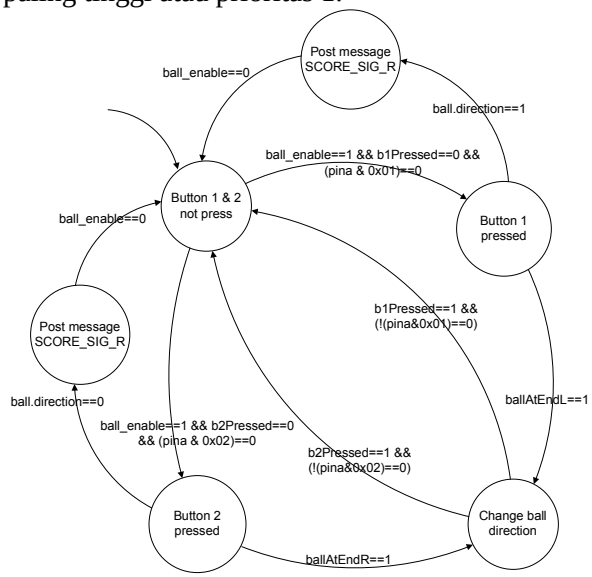


Gambar 3.3 Diagram alir data sistem level 1

Diagram alir data level 1 menunjukkan proses dalam cocoOS yang diperjelas menjadi 3 *multistate*.

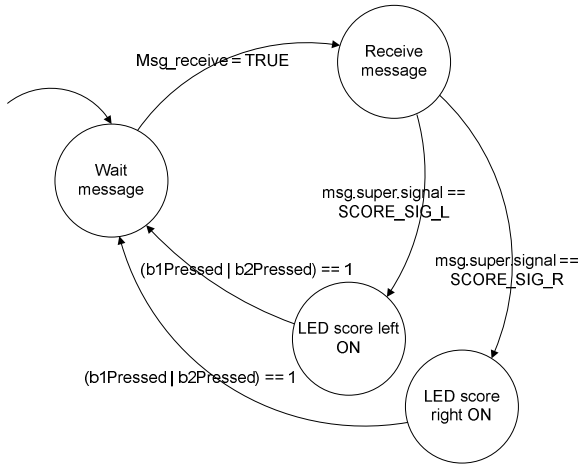
Task pertama yang dirancang adalah `button_task. button_task` adalah *task* yang digunakan untuk memonitor *push button* representasi dari *paddle* pingpong. *Task* ini memonitor *push button* 1 dan *push button* 2 setiap

50 milidetik sekali. *Task* 1 diberi label prioritas paling tinggi atau prioritas 1.



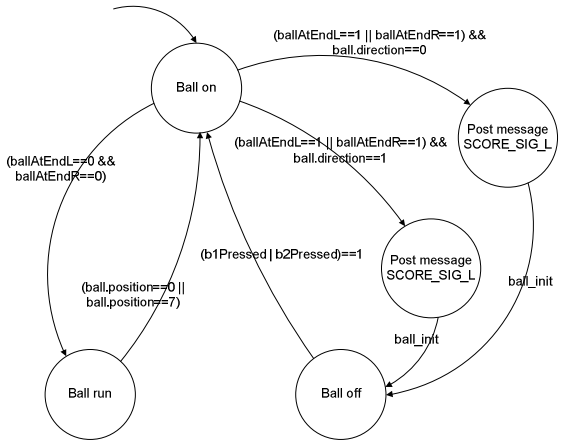
Gambar 3.4 Diagram keadaan *button_task*

Task selanjutnya adalah `score_task`. `score_task` mempunyai tugas mengatur kapan LED skor dan *out of play* menyala. *Task* ini aktif dengan menerima *messages* dari dua *task* lainnya. *Task* 2 akan menunggu dan menerima *messages* tiap 100 milidetik sekali. Jika *messages* telah diterima maka *messages* tersebut akan digunakan untuk memberikan arahan LED skor mana yang harus menyala. `score_task` diberi label prioritas menengah atau prioritas 2. Tingkah laku dari `score_task` dapat dimodelkan dengan diagram keadaan di bawah ini.



Gambar 3.5 Diagram keadaan *score_task*

Task terakhir yang digunakan dalam perancangan sistem ini adalah `ball_task`. *Task* ini berfungsi mengatur pergerakan (posisi dan arah) bola pingpong yang diwakili dengan LED setiap 500 milidetik. Pemberian periode 500 milidetik dilakukan agar pergerakan bola pingpong dapat dilihat oleh mata telanjang. *Task* 3 diberi label prioritas paling rendah atau prioritas urutan 3. Pada kondisi tertentu `ball_task` akan mengirim *messages* kepada `score_task` untuk memberi kesempatan pada `score_task` agar bisa berjalan. Tingkah laku dari `ball_task` dapat dimodelkan dengan diagram keadaan di bawah ini

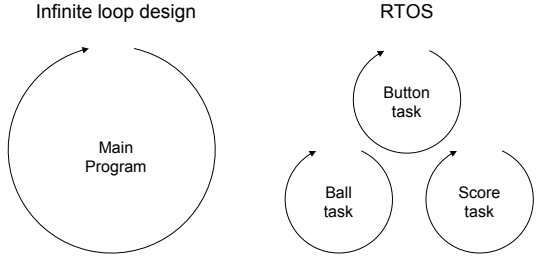


Gambar 3.6 Diagram keadaan *ball_task*

4 PENGUJIAN DAN ANALISA

4.1 Komparasi Sistem

Perancangan sistem tertanam bisa menggunakan *Real Time Operating System* atau *infinite loop design*. Jika kita membandingkan bagaimana sebuah perangkat lunak/program dikembangkan bagi sistem tertanam ukuran kecil dan menengah menggunakan RTOS dan *infinite loop design*, maka kita akan menemukan bahwa tidak terdapat banyak perbedaan. Perbedaan utama antara RTOS dan *infinite loop design* terletak pada arstitektur dan cara penulisan kode/program. RTOS menyediakan *template* tersendiri (termasuk di dalamnya *kernel* dan *scheduler*) yang mempermudah pengguna dalam mengatur penjadwalan. RTOS juga menyediakan beberapa fitur seperti *priority*, *semaphore*, *message queue*, dan lain-lain dalam perancangan sistem. Sedangkan metode *infinite loop design* mengharuskan pengguna untuk membuat dan mengatur sendiri penjadwalan yang akan digunakan.



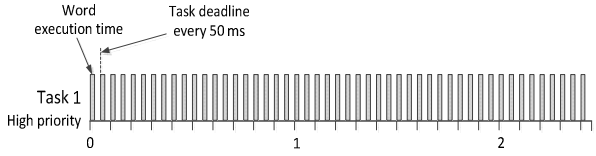
Gambar 4.1 Perbandingan antara sistem non RTOS (*infinite loop design*) dan RTOS

Gambar 4.1 menunjukkan perbedaan mendasar yang terjadi pada fungsi utama antara sistem yang tidak menggunakan RTOS (metode *infinite loop design*) dengan sistem yang menggunakan RTOS. Dapat dilihat bahwa sistem tanpa RTOS akan melakukan eksekusi fungsi utama secara berurutan dan hanya terdapat satu *loop* pemrosesan program. Sedangkan pada sistem yang menggunakan RTOS terdapat beberapa *loop* yang merupakan *task-task* yang akan dijadwal sesuai dengan prioritasnya.

Jika dilihat dari perfomansinya, kedua sistem sudah menunjukkan hasil yang sesuai dengan keinginan. Kekurangan yang terdapat pada metode *infinite loop design* terdapat pada *interrupt* penekanan tombol. Saat terjadi penekanan tombol masih dijumpai *bouncing* sehingga mengganggu kenyamanan dalam penekanannya. Pada sistem yang menggunakan RTOS, tidak dijumpai adanya *bouncing* dalam proses penekanan tombol. Perancangan sistem menggunakan RTOS memberikan beberapa keuntungan, seperti memberi kemudahan dalam mengimplementasikan penjadwalan, menjadikan sistem yang dibuat tahan uji, dan mempermudah dalam proses pemeliharaan ataupun pengembangan lebih lanjut.

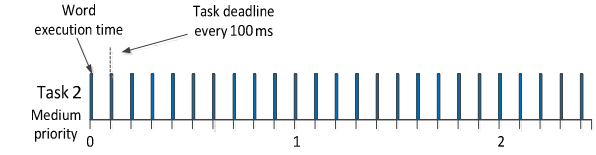
4.2 *Timing Diagram*

Pada bagian ini akan dianalisa mengenai *timing diagram* (diagram pewaktuan) dari sistem *real time* yang telah dibuat, dimulai dari *timing diagram* tiap *task* sampai *timing diagram* keseluruhan sistem. *Task* 1 atau *button task* memiliki prioritas paling tinggi atau *high priority* dengan *deadline* pengerjaan *task* hingga 50 milidetik.



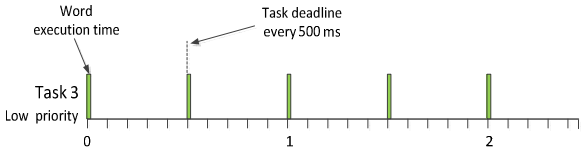
Gambar 4.2 *Timeline* dari *task* 1

Task 2 atau *score task* memiliki prioritas urutan kedua atau *medium priority* dengan *deadline* pengerjaan *task* hingga 100 milidetik.



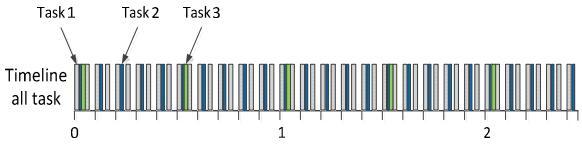
Gambar 4.3 *Timeline* dari *task* 2

Task 3 atau *ball task* memiliki prioritas paling rendah atau *low priority* dengan *deadline* pengerjaan *task* hingga 500 milidetik.



Gambar 4.4 *Timeline* dari *task* 3

Ketiga gambar di atas adalah gambar *timing diagram* untuk tiap *task*. Masing-masing *task* memiliki *deadline* pengerjaan tugas sendiri. Dapat dilihat bahwa waktu eksekusi tiap *task* tidak melebihi *deadline* yang telah ditentukan. Kemudian berdasarkan *timing diagram* tiap *task* dan sifat dari *kernel* dapat dilihat *timing diagram* sistem *real time electronic pingpong* yang mencakup keseluruhan *task*.



Gambar 4.5 *Timeline* dari *task* 3

Pada Gambar 4.5 dapat dilihat bahwa *task* 1 selalu muncul tiap 50 milidetik kemudian dilanjutkan dengan *task* 2 dan *task* 3. *Task* 2 memiliki periode 100 milidetik dan *task* 3 memiliki periode 500 milidetik. Dapat dilihat bahwa semua *task* berjalan sesuai dengan urutannya, dimulai dari *task* 1, diikuti *task* 2 kemudian dilanjutkan *task* 3. Dari Gambar 4.5 dapat juga dilihat adanya *idle task* yang akan muncul tiap 100 milidetik sekali. Kemunculan *idle task* menunjukkan bahwa sistem sedang tidak mengerjakan *task* apapun karena semua *task* sudah berjalan sebelum *deadline* pengerjaan yang telah ditentukan.

4.3 *Clock Tick*

Pengaturannya dilakukan dengan cara memasukkan nilai satu *tick* yang diinginkan ke dalam fungsi API `clock_init(X)`. `X` adalah nilai satu *tick* yang diinginkan dalam satuan waktu milidetik. Pada tugas akhir ini diinginkan nilai satu *tick* sama dengan satu milidetik dan dari perhitungan didapatkan nilai satu *tick* sama dengan satu milidetik. Nilai tersebut didapat dari hasil pengolahan *prescaler* dan perhitungan nilai *timer* 0 pada *file* `clock.c`.

Senarai program *file* `clock.c` akan secara otomatis mengatur nilai *prescaler* dan nilai *timer* 0 berdasarkan dari masukan nilai satu *tick* yang diinginkan. Nilai *prescaler* digunakan untuk menentukan nilai $time\ resolution\ MCU$ dan nilai perhitungan pada *timer* 0 digunakan untuk menentukan nilai $nPulses$ dan $counterValue$.

Nilai $time\ resolution\ MCU$ dan nilai $nPulses$ digunakan untuk penentuan nilai satu *tick*.

$$nPulses = \frac{\left(CPU\_CLOCK \times \frac{tick\_ms}{1000}\right)}{(1 \ll prescaler[i])} \quad (4.1)$$

$$counterValue = 256 - nPulses \quad (4.2)$$

$$TCNT0 = counterValue \quad (4.3)$$

$$TCCR0 = \left((i+1) \ll CS00\right) \quad (4.4)$$

$$time\ resolution\ MCU = \frac{prescaler}{CPU\_CLOCK} \quad (4.5)$$

Kelima persamaan diatas adalah persamaan-persamaan yang digunakan untuk mencari nilai *clock tick*. Karena CPU_CLOCK yang digunakan adalah 8MHz dan nilai *prescaler* MCU yang didapat adalah 64 maka dapat dicari nilai $time\ resolution\ MCU$.

$$time\ resolution\ MCU = \frac{64}{8000000}$$

$$time\ resolution\ MCU = \frac{1}{125000}$$

$$time\ resolution\ MCU = 0{,}000008$$

Perhitungan $time\ resolution\ MCU$ mempunyai arti bahwa MCU akan menghasilkan satu buah sinyal *clock* tiap 8 µs. Jika nilai $nPulses$ dikalikan dengan nilai $time\ resolution\ MCU$ maka akan didapatkan hasil bahwa *timer* 0 akan menghasilkan nilai satu *tick* yang setara dengan satu milidetik.

$$125 \times 8\ \mu s = 1000\ \mu s$$

$$\frac{1000\ \mu s}{1000} = 1\ ms$$

$$\frac{1000\ \mu s}{1000} = 1\ ms$$

4.4 Waktu Tunda

Waktu tunda yang dimaksud adalah waktu tunda yang ada pada sebuah *task*. Penggunaan utama dari *clock tick* adalah untuk penentuan waktu. Pada tugas akhir ini *clock tick* lebih sering digunakan untuk menunda suatu *task*. Penundaan itu dilakukan dengan pemanggilan fungsi API `task_wait(X)`. `task_wait(X)` berfungsi untuk menunda *task* selama `X` *tick*. Lamanya waktu penundaan sangat bergantung pada besarnya nilai satu *tick* yang digunakan. Untuk mempermudah pengamatan dari besarnya waktu tunda pada *task* maka digunakan nilai satu *tick* sama dengan 10 milidetik dan dilakukan pengamatan serta pencatatan nilai waktu tunda menggunakan *stopwatch* pada saat *task* 3, yang memiliki waktu tunda *task* paling besar, sedang aktif.

Diinginkan nilai satu *tick* sama dengan 10 milidetik dengan CPU_CLOCK sebesar 8MHz. Dengan persamaan-persamaan pada pengujian *clock tick* dapat dicari nilai perhitungan satu *tick* sama dengan 10 milidetik.

$$time\ resolution\ MCU = \frac{1024}{8000000}$$

$$time\ resolution\ MCU = 0{,}000128$$

Jika nilai $nPulses$ dikalikan dengan nilai $time\ resolution\ MCU$ maka akan didapatkan hasil bahwa *timer* 0 akan menghasilkan nilai satu *tick* yang setara dengan sepuluh milidetik.

$$79 \times 128\ \mu s = 10112\ \mu s$$

$$\frac{10112\ \mu s}{1000} = 10{,}112\ ms$$

task 3 memiliki waktu tunda *task* sebesar 500 *tick*. Jika digunakan nilai satu *tick* sesuai yang diinginkan maka didapat waktu penundaan sebesar $500 \times 10\ ms = 5\ s$. Jika digunakan nilai satu *tick* hasil perhitungan maka didapat nilai penundaan sebesar $500 \times 10{,}112\ ms = 5{,}056\ s$.

Tabel 4.1 Pengujian waktu tunda pada *task* 2 (dalam sekon)

pengamatan ke-	waktu penundaan berdasarkan		
	keinginan	perhitungan	pengamatan
1	5	5,056	5,12
2	5	5,056	5,02
3	5	5,056	5,03
4	5	5,056	5,05
5	5	5,056	5,10
6	5	5,056	5,06
7	5	5,056	5
8	5	5,056	5,13

Dari tabel 4.1 dapat dilihat bahwa terdapat perbedaan nilai dari masing-masing waktu penundaan berdasarkan keinginan, perhitungan, dan pengamatan. Nilai yang didapat dari hasil perhitungan memiliki selisih nilai dengan waktu penundaan berdasarkan yang diinginkan. Perbedaan nilai tersebut dikarenakan adanya pembulatan nilai saat melakukan perhitungan mencari nilai $nPulses$. Sedangkan perbedaan nilai yang terdapat pada hasil pengamatan dikarenakan karena keterbatasan dan ketidaktelitian dalam proses pencatatan waktu saat menggunakan *stopwatch*.

5 PENUTUP

5.1 Kesimpulan

1. Proses *porting* aplikasi *real time* menggunakan CocoOS ke dalam MCU AVR ATmega16 telah berhasil dilakukan.

2. Dari analisa komparasi sistem didapatkan bahwa sistem pingpong elektronik yang dirancang menggunakan RTOS mempunyai respon yang lebih baik daripada sistem yang dirancang menggunakan metode *infinite loop design*.
3. Dari analisa *timing diagram* dapat dilihat bahwa semua *task* berjalan sesuai dengan urutan.
4. Dari analisa *timing diagram* dapat dilihat bahwa semua *task* berjalan tidak melebihi *deadline* yang telah ditentukan.
5. Pada pengujian *clock tick*, bisa dipastikan bahwa nilai satu *tick* sama dengan satu milidetik.
6. Pada pengujian waktu tunda didapat nilai waktu tunda berdasarkan keinginan sebesar 5s, nilai waktu tunda berdasarkan perhitungan sebesar 5,056s, dan nilai waktu tunda berdasarkan pengamatan berkisar antara 5s hingga 5,13s.
7. Perbedaan nilai yang terjadi pada pengujian waktu tunda disebabkan oleh adanya pembulatan nilai pada perhitungan $nPulses$ dan ketidaktelitian dalam proses pencatatan waktu saat menggunakan *stopwatch*.

5.2 Saran

1 Untuk penelitian selanjutnya sebaiknya menggunakan MCU yang mempunyai memori data (SRAM) lebih besar daripada yang dimiliki Atmega16.

2 Untuk penelitian selanjutnya dapat dilakukan pengujian untuk fitur *semaphore* dipadukan dengan fitur-fitur yang lain seperti *messages* dan *event*.

DAFTAR PUSTAKA

[1] Ariyanto, Endo, *Sistem Operasi Waktu-Nyata*, Institut Teknologi Telkom, Bandung, 2010.

[2] Barry, Ricahrd, *Using The FreeRTOS Real Time Kernel*, http://www.FreeRTOS.org, 2009.

[3] Betz, Robert, *Introduction to Real Time Operating Systems*, Class note – ELEC371, University of Newcastle, Australia, 2001.

[4] Labrosse, Jean J., *µC/OS-II, The Real-Time Kernel*, R & D Publications, Kansas, 1998.

[5] Labrosse, Jean J, *The 10-Minute Guide to RTOS*, Application note AN-1004, Micriµm, Inc., 2001.

[6] Laplante, Phillip A., *Real-Time Systems Design and Analysis*, A John Wiley & Sons, Inc., Publication, 2004.

[7] Wilmshurst, Timothy, *Designing Embedded Systems with PIC Microcontrollers, Principles and Application*, Elsevier Ltd., 2007.

[8] Pont, Michael J., *Patterns fot Time-Triggered Embedded Systems*, TTE Systems Ltd., 2008.

[9] Simon, David E., *An Embedded Software Primer*, Pearson Education, Inc., India, 2005.

[10] -------, *ATmega16 Data Sheet*, http://www.atmel.com/literature, Oktober 2010.

[11] -------, *AVR Studio 4*, http://www.atmel.com/dyn/products/tools_apps.asp?category_id=163&family_id=607&subfamily_id=760&tool_id=2725, Oktober 2010.

[12] -------, *cocoOS*, http://www.cocoos.net/Files/cocoOS_2.2.0.zip, Januari 2011.

[13] -------, *cocoOS documentation 2.2.0*, http://www.cocoos.net/ cocoOS_2.2.0/Doc/html/main.html, Januari 2011.

[14] -------, *WinAVR*, http://sourceforge.net/projects/winavr/files/WinAVR/ 20100110/, Oktober 2010.

FX Ryan Kurniawan
L2F006041

Lahir di Semarang pada tanggal 14 Juni 1988. Merupakan calon Sarjana yang sedang berjuang untuk mendapatkan gelar sebagai Sarjana Teknik di Jurusan Teknik Elektro, Fakultas Teknik, Universitas Diponegoro Semarang dengan konsentrasi pada bidang kontrol dan otomatisasi.

Mengetahui dan mengesahkan,

Dosen Pembimbing I	Dosen Pembimbing II
Iwan Setiawan, ST, MT	Sumardi, ST, MT
NIP. 197309262000121001	NIP. 196811111994121001
Tanggal:_____	Tanggal:_____