Makalah Seminar Tugas Akhir

PENERAPAN *REAL TIME OPERATING SYSTEMS* (RTOS) PADA MIKROKONTROLER AVR (STUDI KASUS CHIBIOS/RT)

Bayu Pujo Leksono[1], Iwan Setiawan, ST, MT[2], Budi Setiyono, ST, MT[2]
Jurusan Teknik Elektro, Fakultas Teknik, Universitas Diponegoro
Jl. Prof. Sudharto, Tembalang, Semarang, Indonesia

ABSTRAK

Penggunaan perangkat lunak dalam perancangan sistem kontrol bukan merupakan hal yang baru. Seiring dengan banyaknya perangkat keras yang menggunakan sistem kontrol maka akan semakin kompleks juga sistem pengontrolan yang akan dirancang. Untuk mempermudah perancangan sistem kontrol tersebut maka pada proses perancangan ikut dimasukkan sebuah perangkat lunak yang bernama real time operating systems *(RTOS). RTOS sendiri merupakan hasil pengembangan pada bidang IT yang kemudian bisa diadaptasikan untuk bidang otomatisasi, salah satunya digunakan untuk merancang sistem kontrol secara* real time*. Dengan sistem* real time *maka sebuah* task (*proses) dapat diselesaikan dalam waktu tertentu yang bisa* user *tentukan sendiri. Selain itu terdapat* scheduling *yang memungkinkan pengerjaan beberapa* task *secara teratur sehingga kemungkinan untuk bertabrakannya beberapa* task *bisa dihindarkan.*

Tujuan tugas akhir ini untuk mengenalkan prinsip kerja sebuah RTOS dan kemampuan serta fungsi-fungsi apa saja yang ada di dalamnya. Selain itu juga mengetahui apa saja kelebihan dan kekurangan menggunakan sebuah RTOS. Kemudian mengetahui bagaimana jika mikrokontroler jenis AVR ditanamkan sebuah RTOS.

Dari hasil pengujian didapatkan hasil bahwa mikrokontroler jenis AVR dengan tipe ATmega128L bisa ditanamkan sebuah RTOS. Kemudian kemampuan dari RTOS tersebut juga bermacam-macam, 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

Dewasa ini *embedded systems* (sistem tertanam) menjadi hal yang penting. Hal ini dapat dilihat dari perkembangannya yang sangat pesat dari aplikasi yang sederhana hingga aplikasi yang kompleks dan penggunaannya pada hampir semua peralatan elektronik dan kendaraan. Di sisi lain perkembangan perangkat lunak pada perancangan sistem tertanam juga sudah semakin banyak digunakan salah satunya adalah penggunaan *real time operating systems* (RTOS) pada beberapa perancangan sistem tertanam. Hal ini dikarenakan aplikasi sistem tertanam yang semakin ke depan semakin kompleks. Oleh karena itu dibutuhkan bantuan perangkat lunak untuk mengordinasikan aplikasi sistem tertanam agar sistem tersebut bukan hanya dapat berjalan dengan baik saja tetapi juga bisa tepat jadwal, deterministik dan efisien.

Perancangan untuk sistem tertanam sendiri terdiri dari 2 macam yaitu *non-real time* dan *real time*. Pada saat ini kebanyakan para pembuat/perancang sistem menggunakan sistem secara *real time* dengan memasukkan fitur RTOS pada sistem *real time*.

Meski sudah diperkenalkan sejak tahun 2003 namun penggunaan RTOS sendiri baru digunakan secara luas sekitar tahun 2007 – 2009. Dan ada saat ini sudah banyak pengembang perangkat lunak yang mengembangkan dan membuat RTOS, baik itu bersifat komersial atau non-komersial. RTOS yang disediakan oleh para pengembang perangkat lunak hampir bisa digunakan pada semua jenis mikrokontroler, sebut saja AVR, PIC, ARM, x86, ColdFire, Blackfin, dll. Bisa dipastikan bahwa bidang RTOS ini akan berkembang lebih pesat lagi, oleh karena itu penulis mengangkat tugas akhir dengan tema RTOS.

2 KONSEP SISTEM *REAL TIME*

Karakter dasar dari RTOS adalah sebuah sistem yang mempunyai beberapa konsekuensi yang akan berpengaruh pada sistem apabila *deadline* (batas akhir waktu pelaksanaan *task*) tidak terpenuhi. RTOS sendiri terdiri dari 2 jenis yaitu, sistem *soft* RTOS dan sistem *hard* RTOS.

Soft RTOS bisa dideskripsikan sebagai sistem yang hampir selalu menyelesaikan *task* dengan waktu yang telah ditentukan. Pada *soft* RTOS kemungkinan penyelesaian *task* melewati batas waktu pelaksanaan *task* masih bisa terjadi. Dan pada sistem *soft* RTOS, apabila terjadi kegagalan mencapai *deadline* dalam waktu yang telah ditentukan maka sistem akan mengalami efek yang tidak begitu berbahaya bagi sistem. Contohnya seperti penurunan performa sistem.

Sedangkan *hard* RTOS merupakan sistem yang dipastikan selalu menyelesaikan *task* dalam waktu yang telah ditentukan. Dikatakan pasti selalu menyelesaikan *task* karena *hard* RTOS selalu menyelesaikan *task* sebelum *deadline* dan

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

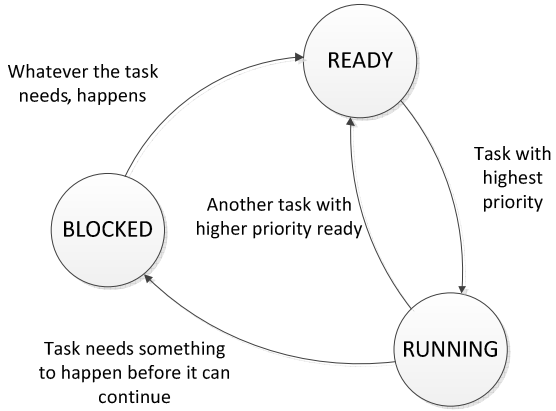
apabila terjadi kegagalan menyelesaikan *task* maka sistem akan mengalami efek berbahaya yang dapat merusak sistem secara keseluruhan.

2.1 *Task*

Sebuah *task*, merupakan sebuah objek/program yang dapat dieksekusi dan beranggapan mempunyai CPU untuk *task* itu sendiri. Salah satu proses perancangan aplikasi dengan RTOS yaitu membagi semua pekerjaan dalam aplikasi tersebut menjadi beberapa bagian *task*.

Tiap *task* merupakan *loop* yang akan terus berulang. Dalam proses pengulangan tersebut, *task* akan mengalami tiga buah keadaan (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

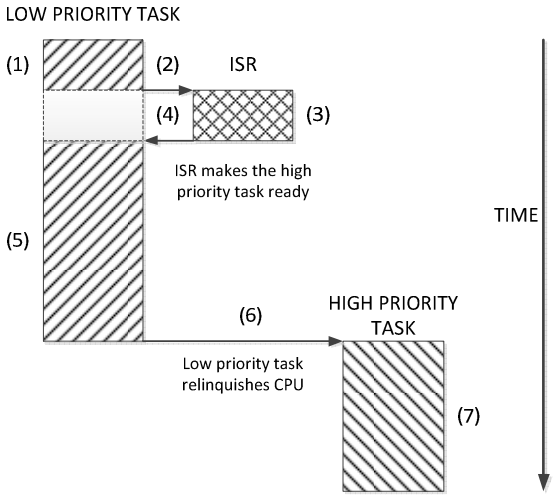
2.2 *Kernel*

Kernel merupakan salah satu bagian dari sistem *multitasking* yang mempunyai fungsi sebagai manajemen dari seluruh *task*, mengatur komunikasi tiap *task* dan yang terpenting adalah mengatur pewaktuan untuk CPU sehingga tidak terjadi *crash* pada CPU.

Untuk *kernel* sendiri terdiri dari dua jenis yaitu, *non-preemptive* dan *preemptive.*

2.2.1 *Non-preemptive Kernel*

Non-preemptive scheduling biasa dikenal dengan nama lain *cooperative multitasking*, 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.

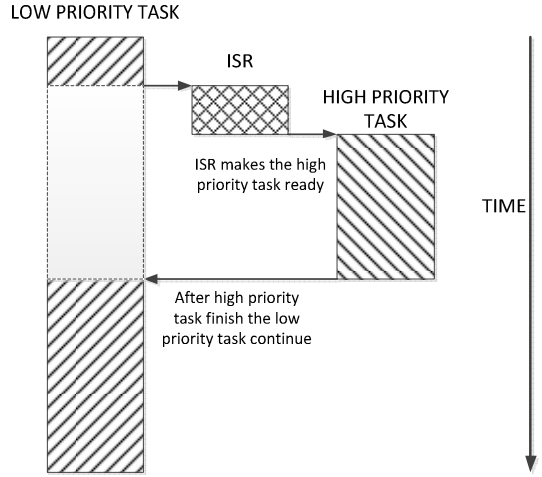
2.2.2 *Preemptive Kernel*

Preemptive kernel banyak digunakan untuk membuat aplikasi dengan RTOS. Hal ini karena *preemptive kernel* mempunyai respons yang lebih bagus daripada *non-preemptive kernel*.

Dari gambar 2.3 dapat dijelaskan prinsip kerja dari *preemptive kernel*. Di mana *task* dengan prioritas tertinggi yang sudah siap dieksekusi akan langsung berjalan. Dan 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. Dengan *preemptive kernel* respons sistem bisa mencapai optimal dan waktu untuk menjalankan *task* dengan prioritas tertinggi bisa ditentukan berbeda

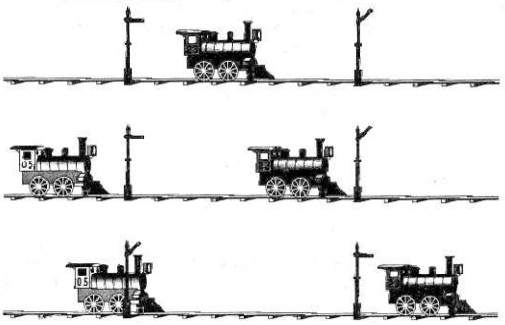
dengan *non-preemptive kernel* yang tidak bisa ditentukan.



Gambar 2.3 Skema prinsip kerja *preemptive kernel*

2.3 *Semaphore*

Pada gambar 2.4 bagian atas dapat dilihat bahwa terdapat sebuah lokomotif yang sedang berhenti karena *semaphore* sebelah kanan sedang turun. Dan ketika lokomotif tersebut berhenti maka *semaphore* yang berada pada sebelah kiri akan turun. Ketika lokomotif kedua datang dan melihat *semaphore* kiri yang turun maka lokomotif kedua akan berhenti dan menunggu *semaphore* kiri menjadi naik. Kemudian *semaphore* kanan akan naik dan lokomotif pertama akan berjalan dan kemudian *semaphore* yang kanan akan turun dan *semaphore* yang kiri akan naik. Jika lokomotif pertama telah berjalan maka sekarang berganti lokomotif kedua yang menunggu *semaphore* kanan untuk naik.



Gambar 2.4 Visualisasi dari cara kerja *semaphore*

Jadi jika dalam RTOS maka lokomotif tersebut adalah *task* yang ingin mengakses *shared data* atau *shared hardware*.

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 oleh 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*. Sering terjadi kasus di mana penundaan sebesar tidak sesuai dengan waktu yang diinginkan.

2.5 ChibiOS/RT

ChibiOS/RT merupakan salah satu dari sekian banyaknya RTOS yang ada pada saat ini. Kata *chibi* yang ada pada ChibiOS/RT merupakan bahasa Jepang yang mempunyai arti kecil. ChibiOS/RT merupakan RTOS yang menggunakan bahasa pemrograman C dan C++. Untuk saat ini ChibiOS/RT sudah bisa di-*porting* ke dalam beberapa arsitektur mikrokontroler di antaranya jenis AVR, ARM7, ARM9, ARMMCx (ARMv6/7-M), STM, MSP430 dan PowerPC. Selain itu juga bisa digunakan sebagai *simulator* pada OS Linux dan Windows x86. Sedangkan terdapat beberapa *compiler* yang sudah *support* dengan ChibiOS/RT di antaranya GCC, IAR, Keil RVCT, Cosmic dan Raisonance (RC).

Gambar 2.14 Logo dari ChibiOS/RT

ChibiOS/RT sendiri merupakan perangkat lunak gratis dengan lisensi GNU *general public license* 3 atau GPL3. Di mana merupakan perangkat lunak ini bisa diperbanyak atau disebarluaskan kepada umum tetapi tidak boleh mengklaim tentang hak cipta dari ChibiOS/RT dan tidak disarankan mengubah konfigurasi dan fungsi yang sudah ada (kecuali konfigurasi untuk mikrokontroler yang digunakan) untuk lebih jelasnya dapat dilihat pada *homepage* ChibiOS/RT http://www.chibios.org. Berikut merupakan fitur-fitur pada ChibiOS/RT antara lain:

a. Perangkat lunak gratis dengan lisensi GPL3.
b. Dirancang untuk aplikasi RTOS.
c. *Portable*.
d. *Preemptive scheduling.*
e. 256 tingkat prioritas, di mana bisa terdapat dua atau lebih *task* dengan prioritas yang sama.
f. *Round robin scheduling* untuk *task* dengan prioritas yang sama.

g. Terdapat *task/thread, virtual timers, semaphores, mutexes, condvars, event flags, messages, mailboxes,* I/O *queues.*
h. Perancangan bisa dilakukan pada PC dengan Windows atau Linux.
i. Terdapat fungsi opsional *heap allocator subsystem* dan *memory pools allocator subsystem.*
j. *Blocking* dan *non-blocking* jalur I/O dengan kemampuan *timeout* dan pembuat *event*.
k. Hampir semua tertulis dalam bahasa C dengan sedikit bahasa *assembler* untuk *porting*.
l. Terdapat *hardware abstraction layer* (HAL) yang mendukung untuk berbagai macam peralatan seperti, *serial*, ADC, CAN I2C, MAC, MMC, PWM, SPI, UART, uIP, lwIP, dan FatFs.

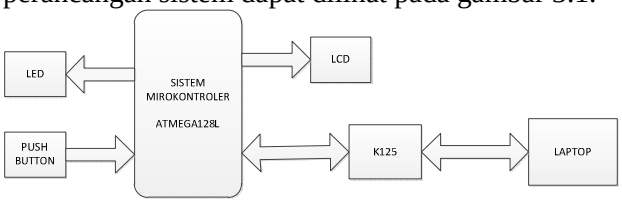
Sedangkan untuk menggunakan ChibiOS/RT diperlukan mikrokontroler yang memenuhi spesifikasi minimum sebagai berikut:

a. Arsitektur minimum CPU dengan 8-*bits*.
b. Mendukung untuk bahasa standar C89 dan C99.
c. Mendukung untuk *maskable interrupt sources*.
d. Memiliki RAM minimal sebesar 2 KB.
e. Memiliki memori untuk program sebesar 16 KB.

3 PERANCANGAN SISTEM

3.1 Perancangan Perangkat Keras

Perangkat keras pada tugas akhir ini hanya untuk mengetes secara nyata cara kerja dari sistem *real time* yang dirancang. Karena jika tanpa perangkat keras maka tidak relevan antara teori dan kenyataan. Dalam perancangan perangkat keras dalam tugas akhir ini meliputi modul sistem minimum ATmega128L, modul LED dan *push button* dan LCD. Secara umum perancangan sistem dapat dilihat pada gambar 3.1.



Gambar 3.1 Blok diagram sistem

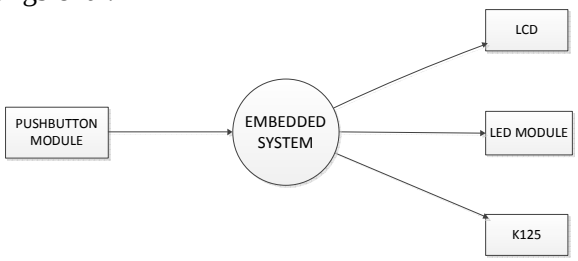
Tiap-tiap bagian blok dari diagram blok tersebut dapat dijelaskan sebagai berikut:

1. Modul *push button* berfungsi sebagai masukkan untuk MCU. Masukkan tersebut bisa berupa *interrupt* untuk sistem, *event* untuk sebuah *task*.
2. Modul LED berfungsi sebagai keluaran di mana nantinya akan dilihat *task* yang berjalan atau jika terdapat sebuah *event* yang terjadi.
3. Modul LCD berfungsi untuk melihat *task* berapa yang sedang berjalan dan tingkat prioritas dari *task* tersebut.
4. Modul sistem minimum ATmega128L merupakan MCU di mana sistem *real time* akan ditanamkan. Dan semua proses *kernel* terjadi di dalam modul ini.
5. Modul k125 merupakan modul 3 *in* 1 di mana selain dapat untuk memprogram MCU, modul ini juga dapat berfungsi sebagai catu daya dan penghubung koneksi *serial* antara MCU dengan laptop.
6. Laptop berfungsi untuk melihat hasil pengiriman data *serial* dari MCU.

3.2 Perancangan Perangkat Lunak

Sistem yang dibuat pada tugas akhir ini, hanya berupa pembuktian dan penjelasan dari beberapa fungsi dan yang terdapat pada ChibiOS. Sistem ini akan terdiri dari empat buah *task* dengan prioritas yang berbeda dan pada saat berjalan nantinya prioritas dari keempat *task* tersebut akan dapat berubah prioritas dan dari sini dapat dilihat respons sistem terhadap tingkat prioritas. Selain itu juga terdapat penggunaan *semaphore*, yang biasanya dipergunakan pada saat ada dua *task* atau lebih ingin mengakses sebuah *hardware*

Untuk membuat sistem seperti yang dijelaskan sebelumnya, pada tugas akhir ini digunakan diagram pendekatan berupa diagram fungsional.

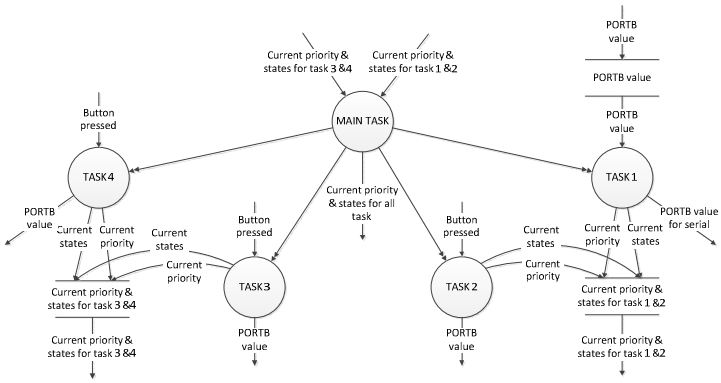


Gambar 3.2 *Context diagram* dari sistem

Main task sendiri merupakan perubahan dari fungsi *main* pada senarai program. Setelah proses inisialisasi pada fungsi *main* selesai maka fungsi *main* akan berubah menjadi *main task* dengan prioritas *normal* dengan keadaan *current running state*.

Task 1 mempunyai fungsi untuk komunikasi serial. Nilai pada *port* B akan dimasukkan ke dalam *serial buffer* dan selanjutnya akan diproses untuk dilanjutkan

pengiriman menuju komputer. Sehingga nilai dan keadaan pada *port* B akan dapat dimonitor.



Gambar 3.3 Diagram alir data sistem level 1

Untuk *task* 2, *task* 3 dan *task* 4 akan menghasilkan keluaran untuk *port* B yang selanjutnya akan diteruskan ke dalam modul LED sehingga perubahan nilai pada *port* B bisa diamati. Pada *task* 2 keluaran yang dihasilkan akan melakukan penjumlahan bilangan *hexadecimal* secara bertahap dari nilai 0x00 hingga nilai 0xFF. Sehingga pada LED yang muncul adalah penjumlahan bilangan *biner* dari awal hingga akhir dan proses tersebut akan terus berulang hingga *task* mendapat *semaphore*, di *reset* atau ada *task* yang lebih tinggi prioritasnya.

Keluaran dari *task* 3 akan membuat LED menyala satu persatu dari bit rendah hingga bit tertinggi dan akhirnya menyalakan semua LED. Dan apabila semua LED telah menyala maka proses selanjutnya merupakan kebalikan dari menyalakan LED yaitu mematikan LED satu persatu dari bit terendah hingga bit tertinggi sampai semua LED mati.

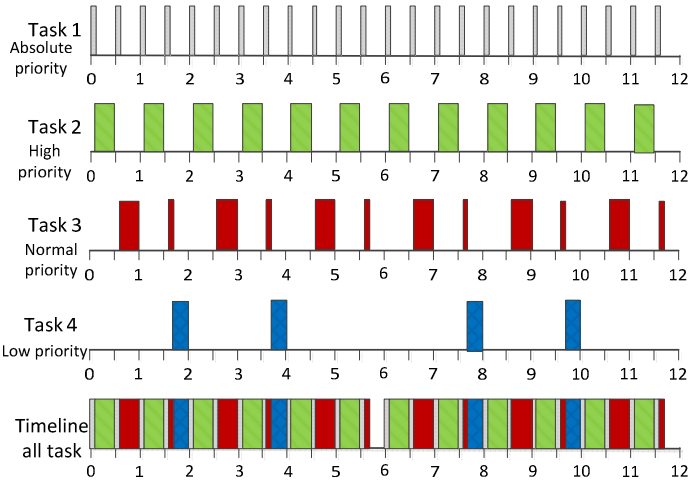
Sedangkan pada *task* 4, keluarannya berupa penjumlahan dan pengurangan nilai pada *port* LED. Sehingga pada LED akan memunculkan nyala LED secara acak secara terus menerus.

4 PENGUJIAN DAN ANALISA

4.1 *Preempt Task*

Pada pengujian *preempt task* akan dilihat sifat dari *kernel* yang digunakan yaitu *preemptive kernel*. Di mana *task* dengan prioritas tertinggi akan menunda *task* dengan prioritas yang lebih rendah.

Pada pengujian ini *task* 1 memiliki prioritas *absolute* dengan *deadline* pengerjaan *task* hingga 500 milidetik. Kemudian untuk *task* 2 memiliki prioritas *high* dengan *deadline* pengerjaan *task* hingga 1 detik. Sedangkan untuk *task* 3 memiliki prioritas *normal* dengan *deadline* pengerjaan *task* hingga 2 detik. Dan terakhir untuk *task* 4, *task* dengan prioritas *low* dan dengan *deadline* pengerjaan 3 detik. Kemudian untuk *timeline* pada saat semua *task* dijalankan secara bersamaan dapat dilihat pada gambar 4.1.

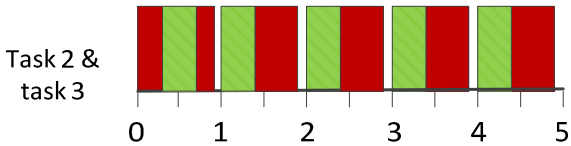


Gambar 4.1 *Timeline* jika semua *task* dijalankan bersamaan

Pada gambar 4.1 dapat dilihat bahwa *task* 1 selalu muncul tiap 500 milidetik kemudian diikuti oleh *task* 2 yang sebelumnya tertunda oleh *task* 1. Pada pengerjaan *task* 3 (setelah *task* 1 dan *task* 2 selesai) terlihat bahwa separuh dari *word execution time* belum selesai karena tertunda oleh *task* 1 dan *task* 2. Kemudian *word execution time* yang belum selesai tersebut akan dilanjutkan periode berikutnya setelah *task* 1 dan *task* 2 selesai.

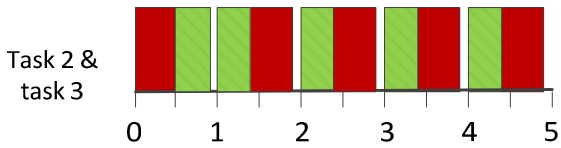
4.2 *Semaphore*

Pada pengujian mula-mula menjalankan *task* 3 yang di dalam senarainya sudah mengandung fungsi API `chBSemWait()` pada awal dan fungsi API `chBSemSignal()` pada akhir senarai. Ketika *task* 3 sedang berjalan kemudian dijalankan *task* 2.



Gambar 4.2 *Timeline* untuk *task* 2 dan *task* 3 jika tanpa *semaphore*

Karena *task* 2 lebih tinggi prioritasnya maka *task* yang sekarang berjalan, tetapi karena pada awal senarai *task* 2 terdapat fungsi API `chBSemWait()` dan *semaphore* sedang berada pada *task* 3 maka *task* 2 akan menunggu hingga *task* 3mencapai fungsi API `chBSemSignal()`. Jika *task* 3 sudah melepas *semaphore* maka *task* 2 dapat berjalan.



Gambar 4.3 *Timeline* untuk *task* 2 dan *task* 3 dengan menggunakan *semaphore*

4.3 *Clock Tick*

Clock tick bisa dikatakan sebagai detak jantung dari *kernel*. Dan pada tugas akhir ini nilai dari satu *clock tick* mendekati atau hampir sama dengan satu milidetik. Nilai tersebut didapat dari hasil perbandingan nilai pada *timer* 0. Dengan menggunakan persamaan 4.1 didapat nilai dari satu *clock tick* mendekati satu milidetik.

$$\frac{\frac{F_{CPU}}{64}}{CH_{Frequency}} - 1 \qquad (4.1)$$

Pada tugas akhir ini F_{CPU} bernilai 8 MHz dan $CH_{Frequency}$ bernilai 1000 Hz. Dan setelah dimasukkan ke dalam persamaan 4.1 didapatkan nilai 124. Kemudian nilai 124 di rubah ke dalam bentuk bilangan *hexadecimal* menjadi 0x7c. Nilai *hexadecimal* tersebut kemudian dimasukkan ke dalam OCR0 (*compare register* pada *timer* 0).

Maka *timer* 0 akan mulai menghitung dari nilai 0x00 hingga 0x7c. Interval dari 0x00 hingga 0x7c inilah yang menjadi satu buah *clock tick*.

Penggunaan utama dari *clock tick* untuk penentuan waktu. Pada tugas akhir ini *clock tick* lebih sering digunakan untuk menjadwal suatu *task*. Selain itu juga pada penggunaan *virtual timer*, di mana dengan *virtual timer* bisa digunakan penjadwalan dan penundaan dalam nilai mikrodetik, milidetik dan detik. Untuk mengubah nilai dari satuan pada *virtual timer* ke dalam *clock tick*, digunakan beberapa persamaan.

Untuk satuan mikrodetik digunakan persamaan 4.2 berikut.

$$\mu S2ST = \frac{(time - 1) \times CH_{Frequency}}{1000000} + 1 \qquad (4.2)$$

Untuk satuan milidetik digunakan persamaan 4.3 berikut.

$$mS2ST = \frac{(time - 1) \times CH_{Frequency}}{1000} + 1 \qquad (4.3)$$

Untuk satuan detik digunakan persamaan 4.4 berikut.

$$S2ST = time \times CH_{Frequency} \qquad (4.4)$$

Hasil yang didapat tidaklah seakurat atau sama persis dengan nilai waktu yang diinginkan karena terdapat pembulatan nilai (terutama pada satuan mikrodetik), tetapi nilai hasil perhitungan dengan menggunakan persamaan 4.2, 4.3 dan 4.4 mendekati dengan nilai waktu yang diinginkan.

5 PENUTUP

5.1 Kesimpulan

1. Tingkat prioritas berfungsi agar *task* yang mempunyai tugas lebih penting dari pada *task* lain didahulukan pengerjaannya.
2. Penentuan tingkat prioritas tertinggi bisa dilakukan dengan melihat periode dari seberapa seringnya muncul *task* tersebut. Semakin sering *task* tersebut muncul maka semakin tinggi prioritas dai *task* tersebut.
3. *Kernel* yang digunakan pada sistem *real time* pada tugas akhir ini bersifat *preemptive*. Di mana pada pengerjaan *task*, mendahulukan *task* dengan prioritas tertinggi dan menunda *task* dengan prioritas rendah.
4. Fungsi *semaphore* yang bekerja secara FIFO (*First In First Out*). Di mana *task* yang pertama kali memanggil *semaphore* maka akan menjalankan tugasnya terlebih dahulu meskipun prioritasnya lebih rendah dari pada *task* yang sudah berada pada *ready state*.
5. *Clock tick* yang merupakan detak jantung dari RTOS mempunyai fungsi yang banyak salah satunya untuk membuat periode *task* (*scheduling*) dan untuk periode kemunculan sebuah *event*.
6. Pengerjaan semua *task* tidak melebihi batas waktu (*deadline*) dari masing-masing *task*.

5.2 Saran

1. Untuk penelitian selanjutnya sebaiknya menggunakan mikrokontroler yang mempunyai memori (SRAM bukan *Flash*) lebih besar dari pada yang dimiliki ATmega128.
2. Jika menginginkan penggunaan interval waktu yang besar sebaiknya nilai *clock tick* dinaikkan menjadi lebih besar dari pada 1 milidetik.
3. Untuk penelitian selanjutnya sebaiknya digunakan mikrokontroler jenis ARM di mana nantinya bisa dilakukan pengujian untuk fitur PAL, MMC, I2C, SPI, dll.

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] Pont, Michael J., *Patterns fot Time-Triggered Embedded Systems*, TTE Systems Ltd., 2008.
[7] Simon, David E., *An Embedded Software Primer*, Pearson Education, Inc., India, 2005.
[8] -------, *Atmega128L Data Sheet*, http://www.atmel.com, Oktober 2009.
[9] -------, *ChibiOS/RT Documentation and Guides*, http://www.chibios.org/dokuwiki/doku.php?id=chibios:documents, Oktober 2010.
[10] -------, *ChibiOS/RT Forum*, http://forum.chibios.org/, Oktober 2010.
[11] -------, *ChibiOS/RT Help*, generate by doxygen, 2011.
[12] -------, *Control.com Forum*, http://www.control.com/, Februari 2011
[13] -------, *GNU Make – An Introduction to Makefiles*, http://www.nondot.org/sabre/Mirrored/GNUMake/make_toc.html, Januari 2011.
[14] -------, *Selected topics in Embedded Systems Design : Roadmaps for Research*, ARTIST project IST-2001-34820, http://www.artist-embedded.org/, 2004.
[15] -------, *Sonsivri Forum*, http://www.sonsivri.com/forum/index.php, Desember 2010.

Bayu Pujo Leksono
L2F006019

Lahir di Tangerang pada tanggal 14 Agustus 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	Budi Setiyono, ST, MT
NIP. 197309262000121001	NIP. 197005212000121001
Tanggal:_____	Tanggal:_____