SISTEM *DOWNLOADER* DATA MEMORI MIKROKONTROLER ATMEL-AVR MELALUI *RADIO FREQUENCY* MENGGUNAKAN SISTEM MULTI TARGET : SISI TARGET

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Teknik

Disusun oleh:
ANANG FAKHRUDIN
NIM. 0410630010

KEMENTRIAN PENDIDIKAN NASIONAL
UNIVERSITAS BRAWIJAYA
FAKULTAS TEKNIK
JURUSAN ELEKTRO
MALANG
2010

SISTEM *DOWNLOADER* DATA MEMORI MIKROKONTROLER ATMEL-AVR MELALUI *RADIO FREQUENCY* MENGGUNAKAN SISTEM MULTI TARGET : SISI TARGET

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Teknik

Disusun oleh :
ANANG FAKHRUDIN
NIM. 0410630010

DOSEN PEMBIMBING

Pembimbing I	Pembimbing II
Ir. Nanang Sulistiyanto	**Adharul Muttaqin, ST, MT**
NIP. 19700113 199403 1 002	**NIP. 19760121 200501 1 001**

LEMBAR PENGESAHAN

SISTEM *DOWNLOADER* DATA MEMORI MIKROKONTROLER ATMEL-AVR MELALUI *RADIO FREQUENCY* MENGGUNAKAN SISTEM MULTI TARGET : SISI TARGET

SKRIPSI
JURUSAN TEKNIK ELEKTRO

Diajukan Untuk Memenuhi Sebagian Persyaratan
Memperoleh Gelar Sarjana Teknik

Disusun oleh:
ANANG FAKHRUDIN
NIM. 0410630010-63

Skripsi ini telah diuji dan dinyatakan lulus pada
tanggal 23 Desember 2010

DOSEN PENGUJI

M. Rif'an, ST., MT.	**Panca Mudjirahardjo, ST, MT.**
NIP. 19710301 200012 1 001	**19700329 200012 1 001**

Dr. Agung Darmawansyah, ST., MT
NIP. 19721218 199903 1 002

Mengetahui
Ketua Jurusan Teknik Elektro

Rudy Yuwono, ST., M.Sc
NIP. 19710615 199802 1 003

PENGANTAR

Segala pujian hanya milik Allah, Rabb semesta alam. Saya bersaksi tiada Illah yang berhak diibadahi kecuali hanya Allah semata, tidak ada sekutu bagi-Nya dan saya bersaksi bahwa Muhammad adalah hamba dan utusan-Nya. Hanya karena pertolongan Allah sematalah penulis mampu menyelesaikan penyusunan skripsi ini. Yaa Rabb, Engkaulah Dzat yang membolak-balikkan hati manusia, tetapkanlah hati hamba-Mu ini di atas agama-Mu, Amien. Semoga shalawat dan salam senantiasa tercurahkan kepada Nabi Muhammad SAW, keluarga beliau, para sahabat dan pengikut beliau sampai hari kiamat nanti.

Skripsi berjudul "Sistem *Downloader* Data Memori Mikrokontroler Atmel-AVR Melalui *Radio Frequency*: Sisi Target" ini disusun untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Teknik di Jurusan Teknik Elektro Universitas Brawijaya.

Penulis menyadari bahwa penyusunan skripsi ini tidak terlepas dari bantuan berbagai pihak. Oleh karena itu, dengan ketulusan dan kerendahan hati penulis menyampaikan terima kasih kepada:

- Ibu dan Bapak atas segala kasih sayang, perhatian serta kesabarannya dalam mendidik dan membesarkan penulis, yang telah mendoakan, membimbing dan mendukung penulis hingga terselesaikannya skripsi ini,
- Rudy Yuwono, ST., MSc selaku Ketua Jurusan Teknik Elektro Universitas Brawijaya,
- Muhammad Aziz Muslim, ST., MT., PhD. selaku Sekretaris Jurusan Teknik Elektro Universitas Brawijaya,
- Ir. M. Julius St, MS selaku Ketua Kelompok Dosen Keahlian Elektronika Jurusan Teknik Elektro Universitas Brawijaya,
- Ir. Nanang Sulistiyanto selaku Dosen Pembimbing I atas segala bimbingan, pengarahan, gagasan, ide, saran serta motivasi yang telah diberikan,
- Adharul Muttaqin, ST., MT selaku Dosen Pembimbing II atas segala bimbingan, saran dan masukan yang telah diberikan.

- Kun ubaidan fahmi selaku partner penelitian yang selalu menemani dalam pengerjaan penelitian dan penyelesaian alat maupun dalam penyusunan laporan.
- Segala pihak yang yang memberikan support untuk pengerjaan penelitian baik itu secara langsung ataupun tidak langsung yang tidak dapat disebut satu persatu.

Pada akhirnya, penulis menyadari bahwa skripsi ini masih belum sempurna. Oleh karena itu, penulis sangat mengharapkan kritik dan saran yang membangun. Penulis berharap semoga skripsi ini dapat bermanfaat bagi pengembangan ilmu pengetahuan dan teknologi.

Malang, 1 Desember 2010

Penulis

ABSTRAK

Anang Fakhrudin, Jurusan Teknik Elektro Fakultas Teknik Universitas Brawijaya, November 2010, *Sistem Downloader Data Memori Mikrokontroler Atmel-AVR Melalui Radio Frequency Menggunakan Sistem Multitarget : Sisi Target*, Dosen Pembimbing: Ir. Nanang Sulistiyanto dan Adharul Muttaqin, ST., MT.

Penggunaan mikrokontroler keluaran Atmel-AVR mencakup berbagai aspek dalam perancangan sistem elektronika, hal ini diarenakan penggunaannya yang praktis, ekonomis dan cukup handal. Penggunaan Atmel-AVR oleh tim robot dari Teknik Elektro Universitas Brawijaya (TEUB) dalam ajang utuk berkompetisi dalam KRI (Kontes Robot Indonesia) ataupun KRCI (Kontes Robot Cerdas Indonesia) menggunakan banyak mikrokontroler untuk tiap proses kerja robot, mulai dari sebagai pengolah hasil sensor dan tranduser, pengolah logika pemrogram, pengatur pergerakan dan posisi robot.

Dalam mencari performa robot secara maksimal dibutuhkan adanya proses *trial* dan *error* berulang-ulang. Dalam pengujiannya dibutuhkan pemindahan robot dari lapangan uji ke komputer untuk proses pemrograman memori mikrokontroler, hal ini menyita banyak waktu dan tenaga karena bobot rata-rata robot sekitar sepuluh kilogram Penggunaan media kabel panjang sebagai solusi terhadap permasalahan ini dianggap tidak praktis.

Alternatif solusi dari permasalahan tersebut adalah mengunakan *downloader* dengan media nirkabel yang terdiri atas sisi pemrogram dan sisi target. Sistem yang terdapat pada sisi target bertugas untuk menangani pemrograman memori mikrokontroler yang berbasis Atmel-AVR. Komunikasi sisi target dan sisi pemrogram menggunakan protokol yang dibangun diatas protokol data serial USART dengan menggunakan media transmisi *radio frequency*.

Operasi pemrograman memori Atmel-AVR meliputi hapus memori, baca tulis memori Flash, baca tulis memori EEPROM, baca tulis Fuse bit serta operasi baca *device signature* mikrokontroler target menggunakan metode ISP (*in-system progamming*) dengan antar muka SPI (*serial peripheral interface*). Sisi target juga dapat berkomunikasi dengan baik secara *half-duplex* dengan sisi pemrogram. Sistem terdiri atas Pusat kendali Atmega162, modul *radio frequency* dan SRAM eksternal 32*Kbyte* sehingga memori maksimal yang dapat ditangani adalah 32 K*byte*. *Baud rate* transmisi data serial dengan format 8N1melalui *radio frequency* yaitu 9600 bps dan jarak jangkau sampai 30 meter atau lebih. Hasil ujicoba lama operasi memori dengan frekuensi SCK 1382,4 kHz untuk operasi tulis flash 42.5 detik dan baca 9.5 detik untuk file ukuran 8*Kbyte*, tulis EEPROM 1.5 detik dan baca 1.5 detik untuk file ukuran 512 *byte*, tulis fusebits 1 detik dan baca 1.5 detik untuk ukuran file 3 *byte*.

Kata Kunci: ***downloader*, mikrokontroler AVR, *Radio frequency*, operasi memori, sistem multitarget, sisi target**.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

Mikrokontroler merupakan piranti elektronik yang memiliki banyak fitur selain dari segi penggunaan yang *user friendly* juga daya beli bagi masyarakat umum lebih ekonomis. Penggunaan mikrokontroler mencakup berbagai aspek dalam perancangan sistem elektronika, hal ini dikarenakan fasilitas yang tertanam pada *chip* mikrokontroler mencakup hampir keseluruhan yang dibutuhkan dalam suatu perancangan dengan pertimbangan penggunaannya yang praktis, ekonomis, dan cukup handal.

Mikrokontroler Atmel-AVR dengan arsitektur AVR (*advanced RISC*) merupakan mikrokontroler yang sering digunakan. Salah satu penggunaan dari mikrokontroler Atmel-AVR saat ini sebagai pengendali utama sistem elektronik tim robot Teknik Elektro Universitas Brawijaya (TEUB) untuk dipergunakan dalam lomba KRI (Kontes Robot Indonesia) ataupun KRCI (Kontes Robor Cerdas Indonesia).

KRI/KRCI merupakan sebuah *event* lomba tahunan dalam bidang robotika yang diselenggarakan oleh Direktorat Jendral Pendidikan Tinggi (DIKTI) melalui Direktorat Pembinaan Penelitian dan Pengabdian Kepada Masyarakat. Universitas Brawijaya merupakan salah satu instansi yang berperan aktif dalam kegiatan KRI/KRCI dengan mengirimkan salah satu tim tiap tahunnya.

Penggunaan mikrokontroler dalam sistem elektronik tim robot TEUB mencakup banyak hal dan terus berkembang. Perkembangan sampai saat ini yaitu penggunaan banyak mikrokontroler dalam sistem keseluruhan yang nantinya menunjang pergerakan robot atau disebut dengan multi-mikrokontroler. Mulai dari sebagai pengolah hasil sensor dan tranduser, Pengolah logika program, pengatur pergerakan dan posisi robot. Hal ini bertujuan untuk mempercepat proses eksekusi dan menghasilkan pergerakan robot yang sigap dan tangkas sehingga mampu berkompetisi dalam perlombaan.

Teknologi untuk proses baca tulis dan hapus memori dari mikrokontroler Atmel-AVR yaitu *downloader* memori Atmel-AVR dengan menggunakan port

parallel ataupun port serial PC (*personal computer*) dan teknologi yang terbaru sudah terintegrasi dengan port USB. Kebanyakan dari kesemua sistem itu menggunakan kabel sebagai media komunikasi PC dengan mikrokontroler, hal ini menjadi faktor kurang praktis dan efisien dalam proses baca tulis dan hapus memori.

Proses *trial* dan *error* untuk menemukan sistem pergerakan yang optimal merupakan faktor penting untuk mempersiapkan robot kondisi siap lomba. Dalam proses tersebut sering terdapat perubahan kode program untuk kondisi tertentu dalam pencarian performa pergerakan robot, Sehingga diperlukan adanya proses baca tulis dan hapus memori dari mikrontroler Atmel-AVR secara berulang-ulang. Sedangkan pada kondisi *real* ketika tersebut robot yang menjadi media mikrokontroler berulang-ulang dipindahkan dari lapangan simulasi lomba ke PC yang dipergunakan untuk penulisan kode program. Hal ini cukup menguras waktu dan tenaga karena untuk menyelesaikan 1 parameter logika program dibutuhkan pemindahan secara berulang-ulang, padahal parameter yang akan diuji jumlahnya tidak sedikit.

Penelitian ini merupakan pengembangan dari penelitian oleh Arief Mardhi Basuki, dkk (2008). Dengan pengembangan dari segi peningkatan kecepatan transfer data dan kompatibel dengan sistem multi-mikrokontroler.

1.2 Rumusan Masalah

Berdasarkan permasalahan yang telah dijelaskan pada bagian latar belakang, maka rumusan masalah yang ditetapkan dalam penyusunan skripsi ini adalah sebagai berikut:

1. Bagaimana merancang protokol komunikasi data serial serta rangkaian sistem pemancar dan penerima data dengan memanfaatkan modul frekuensi radio?
2. Bagaimana merancang perangkat lunak untuk transmisi data melalui USART?
3. Bagaimana merancang sistem untuk proses pemilihan mikrokontroler target?
4. Bagaimana merancang perangkat lunak untuk menangani proses baca tulis dan hapus memori pada mikrokontroler target?
5. Bagaimana merancang sistem pada sisi target sehingga mampu menggantikan *downloader* Atmel-AVR yang melalui media kabel dengan media frekuensi radio?

1.3 Tujuan

Tujuan skripsi ini adalah merancang dan membuat sistem penerima dan pengolah data. Sistem penerima dan pengolah data ini merupakan salah satu bagian dari "Sistem *Downloader* Data Memori Mikrokontroler Atmel-AVR Melalui *Radio Frequency* Menggunakan Sistem Multi Target".

1.4 Ruang Lingkup

Mengacu pada permasalahan yang ada maka pembatasan pokok bahasan pada penyusunan skripsi ini adalah sebagai berikut:

1. Pemancar dan penerima yang digunakan adalah YS-1020. Jenis komponen ini merupakan produk dari ShenZhen Yishi *Electronic technology*.
2. Jarak jangkau komunikasi perancangan diasumsikan dapat mencapai 20 meter kondisi udara terbuka berdasarkan jarak terjauh antara pemrogram dengan robot pada masa uji coba. Hal ini akan menjadi dasar dalam pengujian transfer data unit RF modul.
3. Sistem dirancang untuk menangani mikrokontroler dengan kapasitas memori program maksimum 32 kbyte.
4. Jumlah mikrokontroler yang terpasang dalam sistem yaitu empat buah, hal ini dikondisikan dengan pemakaian rata-rata untuk tiap robot di lomba KRI/KRCI.

1.5 Sistematika Penulisan

Sistematika yang digunakan dalam penyusunan skripsi ini adalah sebagai berikut:

BAB I Pendahuluan

Memuat latar belakang, rumusan masalah, tujuan, ruang lingkup dan sistematika penulisan.

BAB II Teori Penunjang

Membahas dasar teori yang mendukung dalam perencanaan dan pembuatan Sistem *Downloader* Data Memori Mikrokontroler Atmel-AVR Melalui *Radio Frequensi*: Sisi Target.

BAB III Metodologi Penelitian

Menjelaskan tahapan serta metode penelitian di dalam perancanaan alat serta serta menjelaskan metode pengujian alat.

BAB IV Perencanaan dan Pembuatan Alat

Menjelaskan spesifikasi, diagram blok, prinsip kerja serta perancangan Sistem *Downloader* Data Memori Mikrokontroler Atmel-AVR Melalui *Radio Frequensi*: Sisi Target.

BAB V Pengujian Alat

Memuat hasil pengujian terhadap alat yang telah direalisasikan.

BAB VI Kesimpulan dan Saran

Memuat kesimpulan dan saran terhadap hasil yang diperoleh dari penelitian.

BAB II
TINJAUAN PUSTAKA

Pengetahuan dan pemahaman mengenai teori dasar yang mendukung perencanaan dan realisasi alat meliputi :

1) Mikrokontroler ATmega162
2) Antarmuka SPI (*Serial Peripheral Interface*)
3) Metode ISP (*In-System Programming*)
4) Transmisi Data.
5) Unit RF (*Radio Frequency*)
6) Protokol Komunikasi
7) Metode pendeteksi kesalahan transmisi data

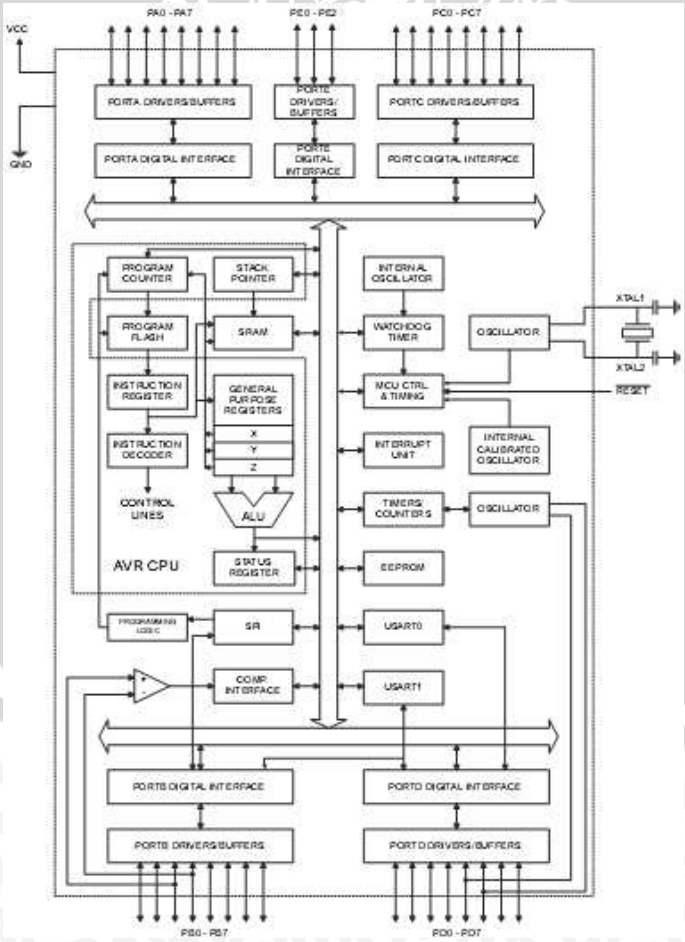
2.1 Mikrokontroler ATmega162

Mikrokontroler ATmega162 merupakan salah satu mikrokomputer 8 bit buatan Atmel yang terintegrasi dalam satu buah keping IC (*single chip microcomputer*) dan termasuk dalam keluarga AVR. AVR merupakan mikrokontroler produksi Atmel yang menggunakan arsitektur RISC (*Reduced Instruction Set Computing*). AVR pertama kali diperkenalkan pada tahun 1996. AVR mengkombinasikan arsitektur RISC, memori *flash* internal dan jumlah *register* yang banyak (32 buah) untuk memperoleh ukuran kode program, kinerja, dan konsumsi daya yang optimal.

Berdasarkan literatur (Gadre, 2001: 5), sebagian besar instruksi AVR dieksekusi dalam satu siklus *clock*, berkemampuan 12 kali lebih cepat daripada arsitektur CISC (*Complex Instruction Set Computing*) konvensional yang ada. Kelebihan lainnya, arsitektur AVR dirancang untuk bekerja secara efisien menggunakan bahasa tingkat tinggi seperti contohnya bahasa C. Mikrokontroler ini terdiri atas CPU, *on chip clock*, *timer*, paralel dan serial I/O, PEROM, RAM, EEPROM. Blok diagram dari mikrokontroler AVR ATmega162 dapat dilihat dalam Gambar 2.1.

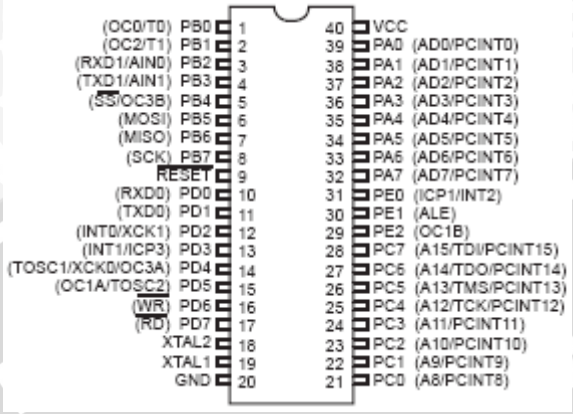
Mikrokontroler ATmega162 memiliki sejumlah kelengkapan antara lain sebagai berikut :

- Memiliki 16K *byte In-system Self-programmable* memori program *Flash*;
- Memiliki 512 *byte* EEPROM;
- Memiliki 1K *byte* SRAM internal;
- Mendukung akses memori eksternal sampai 64K *Byte*;
- Memiliki 2 buah *Timer/Counter* 8 bit dengan *prescaler* terpisah serta dapat beroperasi pada mode pembanding (*compare mode*);
- Memiliki 2 buah *Timer/Counter* 16 bit dengan *prescaler* terpisah, dapat beroperasi pada mode pembanding (*compare mode*) maupun *capture mode*;
- Memiliki 2 buah *programmable* serial USART;
- Antarmuka serial SPI *Master*/*Slave*;
- Memiliki 35 jalur I/O;
- Tegangan operasi antara 2,7 volt sampai 5,5 volt; dan
- Frekuensi kerja 0 sampai 16 MHz.



Gambar 2.1 Diagram blok ATmega162
Sumber: Atmel, 2007: 3

Konfigurasi pin mikrokontroler ATmega162 dengan kemasan 40-pin DIP (*dual in-line package*) dapat dilihat dalam Gambar 2.2.



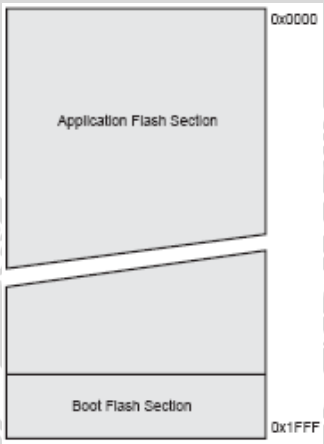
Gambar 2.2 Konfigurasi pin mikrokontroler ATmega162
Sumber: Atmel, 2007: 2

Dari Gambar 2.2 tersebut dapat dijelaskan secara fungsional konfigurasi pin mikrokontroler Atmel AVR ATmega162 sebagai berikut:

1. VCC merupakan pin yang berfungsi sebagai pin masukan catu daya.
2. GND adalah pin *ground*.
3. PortA (PA0...PA7) merupakan pin I/O dua arah dengan *pull-up* internal serta pin fungsi khusus, yaitu untuk pengalamatan *byte* rendah sekaligus saluran data untuk mengakses SRAM eksternal.
4. PortB (PB0...PB7) merupakan pin I/O dua arah dengan *pull-up* internal serta pin fungsi khusus, yaitu *Timer/Counter*0, *Timer/Counter*2, komparator analog, dan SPI.
5. PortC (PC0...PC7) merupakan pin I/O dua arah dengan *pull-up* internal serta pin fungsi khusus, yaitu untuk pengalamatan *byte* tinggi untuk mengakses SRAM eksternal, antarmuka JTAG.
6. PortD (PD0...PD7) merupakan pin I/O dua arah dengan *pull-up* internal serta pin fungsi khusus, yaitu interupsi eksternal, *Timer/Counter1*, *Timer/Counter3*, komunikasi serial USART serta sinyal-sinyal *strobe* untuk mengakses SRAM eksternal.
7. PortE (PE2..PE0) merupakan pin I/O dua arah dengan *pull-up* internal serta fungsi khusus yang diantaranya adalah ALE (*Address Latch Enable*).

8. RESET merupakan pin yang digunakan untuk mereset mikrokontroler.
9. XTAL1 dan XTAL2 merupakan pin masukan untuk *clock* eksternal.

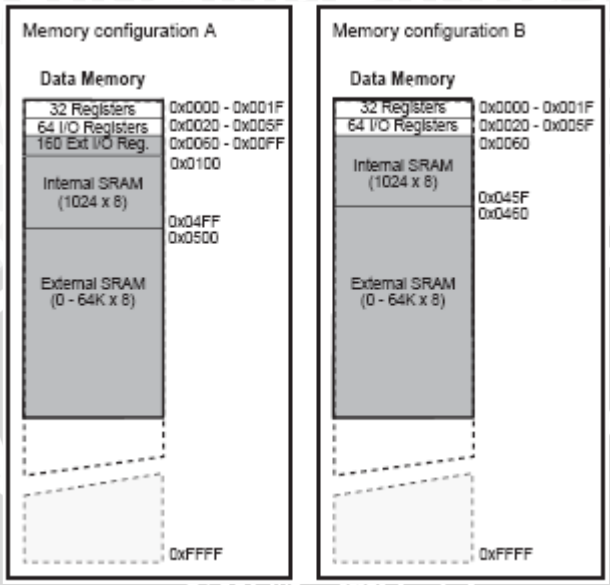
Mikrokontroler ATmega162 mempunyai memori yang terdiri atas memori Flash, memori SRAM, dan memori EEPROM. Mikrokontroler ATmega162 mempunyai 16K *Byte* memori *On-chip In-System Reprogrammable Flash* untuk menyimpan program. Instruksi pemrograman AVR hampir seluruhnya memiliki lebar 16 atau 32 bit sehingga organisasi memori Flash yang digunakan adalah 8K x 16. Untuk memenuhi keperluan keamanan, maka ruang memori Flash dibagi menjadi dua bagian yaitu bagian *Boot Program* dan bagian *Application Program*. Pembagian peta memori program ditunjukkan dalam Gambar 2.3



Gambar 2.3 Peta memori program mikrokontroler ATmega162
Sumber: Atmel, 2007: 17

Mikrokontroler ATmega162 mempunyai organisasi memori SRAM seperti yang tampak dalam Gambar 2.4. Organisasi memori dengan konfigurasi A tidak kompatibel dengan mikrokontroler ATmega161 sedangkan organisasi memori dengan konfigurasi B kompatibel dengan mikrokontroler ATmega161. Lokasi SRAM dengan 1280 alamat pertama terdiri atas file register, memori I/O, memori I/O tambahan, serta memori data SRAM internal. Lokasi SRAM dengan 32 alamat pertama adalah register file, kemudian 64 alamat selanjutnya adalah memori I/O standart, kemudian 160 alamat berikunya adalah memori I/O tambahan dan 1024 alamat selanjutnya adalah lokasi SRAM internal. Dalam mode kompatibel, 1120 alamat terendah memori data adalah lokasi register file, memori I/O, serta memori data SRAM internal. Lokasi SRAM dengan 96 alamat pertama

adalah register file dan memori I/O, kemudian 1024 alamat selanjutnya adalah lokasi SRAM internal.



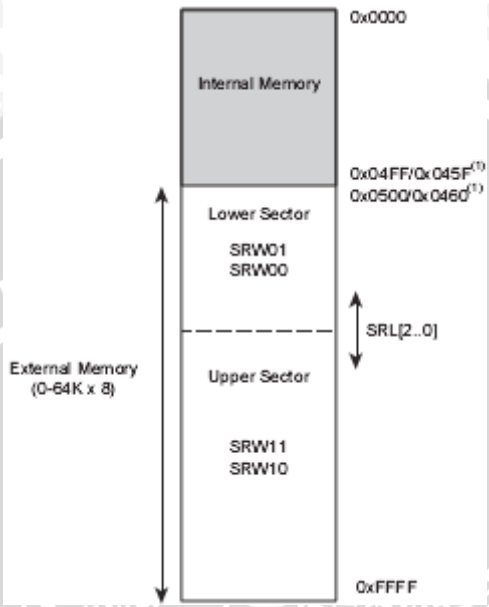
Gambar 2.4 Peta memori data mikrokontroler ATmega162
Sumber: Atmel, 2007: 19

Memori SRAM eksternal dapat digunakan secara bersama-sama dengan SRAM pada mikrokontroler ATmega162. Pengalamatan pada SRAM eksternal terletak pada 64k alamat lokasi memori setelah alamat SRAM internal. Pada mode normal, alamat SRAM paling rendah yang digunakan untuk mengalamati register file, I/O, *Extended I/O*, dan SRAM internal adalah 1280 *byte*. Sedangkan pada mode kompatibel dengan ATmega161, alamat SRAM paling rendah adalah 1120 *byte* karena tidak menggunakan *Extended I/O*, sehingga jika menggunakan 64K *byte* (65,536 *byte*) memori eksternal maka alamat yang tersedia untuk pengalamatan memori eksternal pada mode normal adalah 64,256 *byte* sedangkan pada mode kompatibel hanya dapat mengalamati 64,416 *byte*.

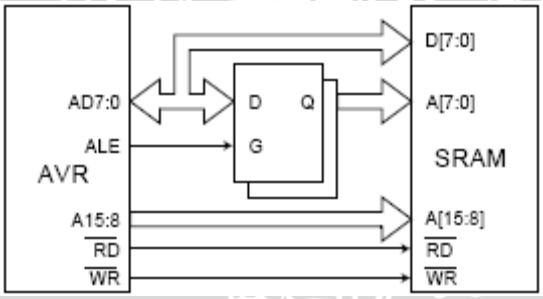
Ketika memori eksternal digunakan, pengalamatan memori di luar SRAM internal dilakukan dengan cara menggunakan pin memori eksternal. Konfigurasi memori dan pemilihan lokasi untuk pengalamatan memori eksternal ditunjukkan dalam Gambar 2.5.

Antarmuka memori SRAM eksternal dengan mikrokontroler yang terdiri atas beberapa jalur yaitu pin AD 7:0 sebagai pemilihan bus alamat *byte* rendah dengan bus data, A 15:8 sebagai bus alamat *byte* tinggi, ALE: (*Address Latch*

Enable) sebagai penahan alamat, RD: *Read strobe* sebagai sinyal kontrol untuk membaca memori, dan WR: *Write strobe* sebagai sinyal kontrol untuk menulis memori. Antarmuka mikrokontroler dengan SRAM eksternal seperti ditunjukkan dalam Gambar 2.6.



Gambar 2.5 Memori eksternal dengan pemilih alamat memori
Sumber: Atmel, 2007: 26



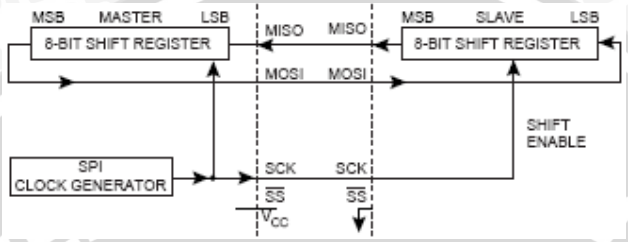
Gambar 2.6 Memori eksternal dengan pemilih alamat memori
Sumber: Atmel, 2007: 27

2.2 Antarmuka SPI (Serial Peripheral Interface)

Serial Peripheral Interface memungkinkan komunikasi data secara sinkron berkecepatan tinggi antar devais Atmel AVR atau antara devais Atmel AVR dengan *peripheral* lain yang mendukung SPI. Fitur SPI pada Atmel AVR adalah sebagai berikut:

- *Full Duplex*, data transfer sinkron menggunakan 3 kabel;
- Beroperasi sebagai *Master* atau *Slave*;

- Data transfer awal LSB atau MSB;
- Tujuh *bit rate* yang dapat diprogram;
- *Interrupt Flag* (Flag Interupsi) pada akhir transmisi data;
- *Colletion Flag Protection* (Flag Proteksi) untuk kegagalan penulisan;
- *Wake-up from idle mode*;
- *Double Speed (CK/2) Master SPI Mode*.



Gambar 2.7 Hubungan antara SPI *Master-Slave*
Sumber: Atmel, 2007: 158

Interkoneksi antara CPU *Master* dengan CPU *Slave* menggunakan SPI ditunjukkan dalam Gambar 2.8. Sistem terdiri atas dua register geser dan sebuah *Master Clock Generator*. *Master* SPI mengatur siklus komunikasi dengan SPI *Slave* ketika pin *Slave Select* ($\overline{SS}$) pin diberi logika rendah. *Master* dan *Slave* menyiapkan data yang akan dikirim ke dalam masing-masing *Shift Registers* (register geser), *Master* juga membangkitkan pulsa *clock* pada saluran SCK yang diperlukan untuk proses pertukaran data. Data selalu digeser dari *Master* menuju ke *Slave* pada saluran *Master Out - Slave In* (MOSI), dan pergeseran data dari *Slave* menuju ke *Master* pada saluran *Master In - Slave Out* (MISO).

Secara umum, konfigurasi arah data pada pin MISO, MOSI, SCK serta $\overline{SS}$ pada *Master* dan *Slave* SPI sesuai dengan data dalam Tabel 2.2.

Tabel 2.1 Konfigurasi pin pada mode *Master* dan *Slave* SPI

Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User Defined	Input
MISO	Input	User Defined
SCK	User Defined	Input
$\overline{SS}$	User Defined	Input

Sumber: Atmel, 2007: 158

Ketika dikonfigurasikan sebagai *Master* SPI, antarmuka SPI tidak secara otomatis dikendalikan dari saluran $\overline{SS}$. Hal ini harus dikendalikan melalui program sebelum proses komunikasi data dimulai. Setelah proses inisialisasi

Master SPI selesai, penulisan data satu *byte* pada *SPI Data Register* (SPDR) akan memulai *SPI Clock Generator* kemudian hardware akan menggeser delapan bit data tersebut menuju *Slave*. Setelah penggeseran data satu *byte* selesai maka *SPI Clock Generator* akan berhenti, pada akhirnya akan mengeset *Transmission Flag* (SPIF). Jika bit *SPI Interrupt Enable* (SPIE) pada *SPI Control Register* (SPCR) dalam kondisi set, maka akan terjadi interupsi. *Master* SPI dapat melanjutkan menggeser *byte* selanjutnya dengan menuliskannya pada SPDR atau meng-akhirinya dengan memberikan logika tinggi pada saluran *Slave Select* ($\overline{SS}$). *Byte* data terakhir akan disimpan di dalam *buffer* sampai proses selanjutnya.

Ketika dikonfigurasikan sebagai *Slave* SPI, antarmuka SPI akan menyebabkan *Slave* menjadi tidak aktif dengan MISO tri-sate selama $\overline{SS}$ diberi logika tinggi. Dalam keadaan ini, *software* dapat meng-*update* isi SPDR tetapi data tidak digeser ketika pulsa *clock* masuk pada pin SCK sampai pin $\overline{SS}$ diberi logika rendah. Setelah satu *byte* data selesai digeser, maka pada akhirnya akan mengeset *Transmission Flag* (SPIF). Jika bit SPIE pada SPCR dalam kondisi set, maka akan terjadi interupsi. *Slave* SPI selanjutnya dapat menempatkan data baru pada SPDR sebelum membaca data yang masuk. Data terakhir yang masuk akan disimpan pada register *buffer* sampai proses selanjutnya.

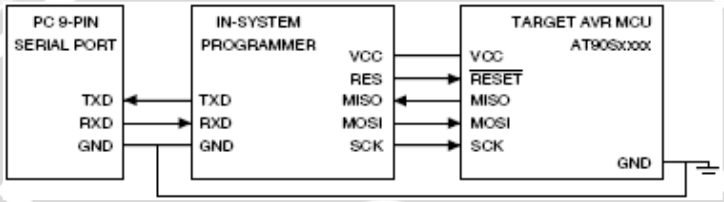
Sistem di-*buffer* (disangga) satu kali pada sisi pemancar dan di-*buffer* (disangga) dua kali pada sisi penerima. Hal ini mempunyai maksud bahwa *byte* data yang dikirimkan tidak dapat ditulis pada SPDR sebelum seluruh siklus penggeseran data lengkap. Ketika menerima data, data yang diterima harus dibaca dari SPDR sebelum seluruh pergeseran masuk data baru lengkap. Jika tidak, maka *byte* pertama akan hilang.

2.3 Metode ISP (In-System Programming)

ISP (*In-System Programming*) adalah salah satu metode untuk memprogram semua *non-volatile* memori yang terdapat pada *chip* mikrokontroler AVR. *In-System Programming* berkomunikasi secara serial menggunakan antarmuka tiga jalur SPI (*Serial Peripheral Interface*).

Antarmuka SPI yang terdiri atas tiga jalur komunikasi yaitu Serial *Clock* (SCK), *Master In - Slave Out* (MISO) dan *Master Out - Slave In* (MOSI). Pada

saat memprogram mikrokontroler Atmel AVR, *In-System Programmer* selalu dioperasikan sebagai *Master* sedangkan sistem target sebagai *Slave*. Selain ketiga jalur antarmuka SPI tersebut, ISP membutuhkan satu jalur tambahan untuk keperluan reset pada sistem target. Bagian *Master* dan bagian *Slave* harus dioperasikan dengan menggunakan tegangan referensi yang sama, yaitu dengan menghubungkan Vcc dan Ground kedua bagian. Antarmuka SPI ditunjukkan dalam Gambar 2.9 sedangkan fungsi keenam jalur antarmuka SPI terdapat dalam Tabel 2.3.



Gambar 2.8 Blok diagram antarmuka pada ISP
Sumber: Atmel, 2002b: 2

Memori Flash dan EEPROM dapat diprogram secara serial menggunakan bus SPI selama $\overline{\text{RESET}}$ diberi logika rendah. Antarmuka serial terdiri dari pin SCK, MOSI (input), dan MOSI (output). Setelah $\overline{\text{RESET}}$ diberi logika rendah, maka yang pertama kali diperlukan untuk dieksekusi adalah instruksi untuk mengaktifkan pemrograman. Setelah itu operasi untuk pemrograman maupun menghapus semua memori program dapat dieksekusi. Pemetaan pin untuk keperluan pemrograman serial terdapat pada Tabel 2.4 sedangkan antarmuka untuk keperluan proses verifikasi dan pemrograman serial SPI terdapat dalam Gambar 2.10.

Tabel 2.2 Fungsi jalur antarmuka SPI

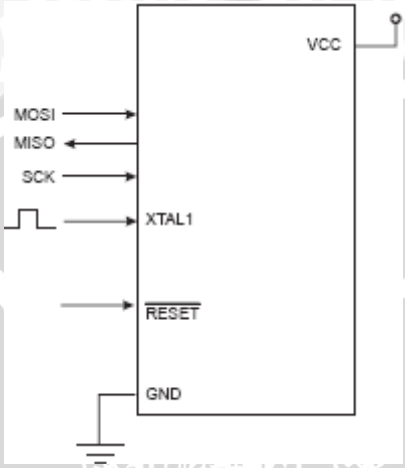
Pin	Name	Comment
SCK	Serial Clock	Programming clock, generated by the In-System Programmer (Master)
MOSI	Master Out – Slave In	Communication line from In-System Programmer (Master) to target AVR being programmed (Slave)
MISO	Master In – Slave Out	Communication line from target AVR (Slave) to In-System Programmer (Master)
GND	Common Ground	The two systems must share the same common ground
RESET	Target AVR MCU Reset	To enable In-System Programming, the target AVR Reset must be kept active. To simplify this, the In-System Programmer should control the target AVR Reset
V_{CC}	Target Power	To allow simple programming of targets operating at any voltage, the In-System Programmer can draw power from the target. Alternatively, the target can have power supplied through the In-System Programming connector for the duration of the programming cycle

Sumber: Atmel, 2002b: 3

Tabel 2.3 Pin Mapping SPI Serial Programming

Symbol	Pins	I/O	Description
MOSI	PB5	I	Serial Data in
MISO	PB6	O	Serial Data out
SCK	PB7	I	Serial Clock

Sumber: Atmel, 2007: 245



Gambar 2.9 Verifikasi dan pemrograman serial SPI
Sumber: Atmel, 2007: 245

2.4 Transmisi Data

Transmisi data merupakan proses pengiriman data dari satu sumber ke penerima data. Ada tiga komponen utama dalam proses transmisi data, yaitu: sumber data (*source*), media transmisi (*transmission media*), dan penerima (*receiver*). Media transmisi yang dapat digunakan sebagai jalur transmisi atau carrier dari data yang dikirimkan dapat berupa kabel maupun gelombang elektromagnetik. Bila sumber dan penerima data jaraknya cukup jauh, media transmisinya dapat digunakan gelombang elektromagnetik yang dipancarkan melalui udara terbuka yang dapat berupa sistem laser, gelombang radio atau sistem satelit.

Pengiriman data serial melalui media udara menggunakan gelombang radio sebagai pembawa data. Secara sederhana proses pengiriman data menggunakan gelombang radio adalah, sinyal informasi atau data yang akan dikirimkan ditumpangkan terlebih dahulu ke sinyal pembawa.

Proses menumpangkan sinyal informasi ini disebut dengan modulasi. Gabungan antara kedua sinyal tersebut kemudian akan dipancarkan oleh transmiiter. Pada receiver, gelombang pembawa yang membawa sinyal informasi

tersebut diterima, kemudian dipisahkan antara gelombang pembawa dan sinyal informasi, sehingga diperoleh kembali sinyal informasi. proses ini disebut demodulasi.

2.5 Unit RF (*Radio Frequency*)

Modul RF yang digunakan dalam perancangan ini adalah YS-1020. Digunakannya YS-1020 sebagai modul RF (*Radio Frequency*) pada perancangan ini. Input data adalah serial dengan level TTL (*Transistor – Transistor Logic*). Modul Low power RF seri YS-1020 didesain untuk system transmisi data tanpa kabel. Ys-1020 mengadopsi IC Chipeon CC1020RF, bekerja pada aliran pita frekuensi, penerima dan pemancar yang sudah terintegrasi pada modul. Alat ini langsung dapat dihubungkan dengan prosesor monolithic, pc, perangkat RS485, dan komponen UART dengan RS-232, RS-485 dan UART/TTL port level interface.

2.5.1 Fitur Utama Produk

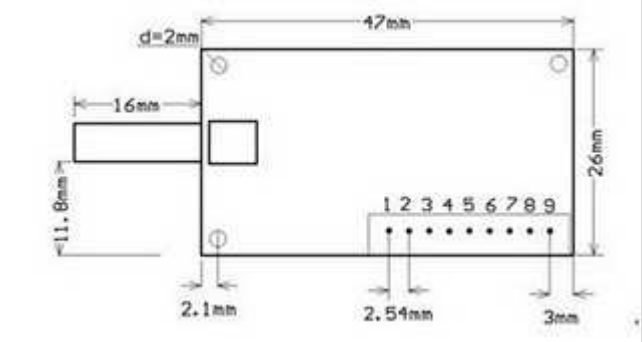
a. Menggunakan Frekuensi pembawa: 433/450/868MHz atau pilihan lainnya
b. Perangkat penghubung: RS-232/RS-485/TTL
c. Mendukung banyak saluran komunikasi: 8 saluran, dapat diperluas untuk 16/32 saluran
d. Baud-rate di udara: 1200/2400/4800/9600/19200/38400bps, Pengiriman data secara transparan: apa yang telah diterima persis dengan apa yang dikirim, cocok dengan standarisasi atau bukan standarisasi protokol pengguna
e. Format penghubung: ketetapan untuk pengguna 8N1/8E1/801, atau disesuaikan dengan format penghubung lainnya
f. Pemodulasian: GFSK. Berdasarkan pada modulasi GFSK, tahan terhadap gangguan yang besar dan Bit Error Rate yang rendah
g. Half duplex: penggabungan untuk penerima dan pengirim yang menjadi satu didalam satu alat, kemampuan merubah dengan sendirinya untuk menerima dan mengirim selama 10ms
h. Penggunaan daya yang rendah dan fungsi *stand by*

i. Batas temperatur : -35 derajat celsius sampai 75 derajat celsius (-31 ~ 167 F)

j. Batas kelembaban: 10% ~ 90% kelembaban relatif tanpa pengembunan

Gambar 2.10 Ys-1020 (*radio frequency*)
Sumber: ShenZhen Yishi, 2009

Sedangkan untuk konfigurasi pin YS-1020 ditunjukkan dalam Gambar 2.3. Dan untuk penjelasan dari tiap-tiap pin ditunjukkan dalam Tabel 2.1.



Gambar 2.11 Konfigurasi Pin YS-1020
Sumber: ShenZhen Yishi, 2009

Tabel 2.4 Konfigurasi YS-1020

No. pin	Nama pin	Penjelasan	Level	Koneksi dengan terminal
1	GND		Ground	
2	Vcc	+3,3 □ 5,5 V		
3	RXD/TTL	TTL	TxD	
4	TXD/TTL	TTL	RxD	
5	DGND			
6	A dari RS-485 atau TXD dari RS-232		A(RxD)	
7	B dari RS-485 atau RXD dari RS-232		B(TxD)	
8	Sleep control (input)	TTL	Sleep signal	Low level sleep
9	Ex-factory testing			

Sumber: **ShenZhen Yishi, 2009**

2.5.2 Spesifikasi

a. Daya untuk RF: 10mW/10dBm

b. Arus yang diterima: 25mA

c. Arus mengirim: 40mA
d. Arus sleep mode: 20µA
e. Suplai daya: Tegangan DC 5V atau 3.3V Sensitivitas dalam menerima : -115 dBm (@9600bps) -120 dBm (@1200bps)
f. Ukuran: 47mm x 26mm x 10mm (tanpa antenna)
g. Jangkauan : 0.5m (BER=10-3@9600bps, saat antena berada 2 meter diatas permukaan tanah pada area terbuka) 0.8m (BER=10-3@1200bps, saat antena berada 2 meter diatas permukaan tanah pada area terbuka)

2.6 Protokol Komunikasi

Protokol adalah sebuah aturan atau standar yang mengatur atau mengijinkan terjadinya hubungan, komunikasi, dan perpindahan data antara dua atau lebih perangkat komputer. Protokol dapat diterapkan pada perangkat keras, perangkat lunak atau kombinasi dari keduanya. Pada tingkatan yang terendah, protokol mendefinisikan koneksi perangkat keras.

Protokol perlu diutamakan pada penggunaan standar teknis, untuk menspesifikasi bagaimana membangun komputer atau menghubungkan peralatan perangkat keras. Protokol secara umum digunakan pada komunikasi *real-time* dimana standar digunakan untuk mengatur struktur dari informasi untuk penyimpanan jangka panjang.

Kebanyakan protokol memiliki salah satu atau beberapa dari hal berikut:

- Melakukan deteksi adanya koneksi fisik atau ada tidaknya komputer atau perangkat lainnya.
- Melakukan metode "jabat-tangan" (*handshaking*).
- Bagaimana mengawali dan mengakhiri suatu pesan.
- Bagaimana format pesan yang digunakan.
- Yang harus dilakukan saat terjadi kerusakan pesan atau pesan yang tidak sempurna.
- Mengakhiri suatu koneksi.

Dalam membuat protokol ada tiga hal yang harus dipertimbangkan, yaitu efektivitas, kehandalan, dan kemampuan dalam kondisi gagal. Agar protokol

dapat dipakai untuk komunikasi diberbagai pembuat perangkat maka dibutuhkan standarisasi protokol.

2.6.1 Komunikasi Data Serial USART

USART (*Universal Synchronous and Asynchronous serial Receiver and Transmitter*) adalah seperangkat komunikasi serial yang memiliki tingkat fleksibilitas yang tinggi. Beberapa fitur yang dimiliki oleh USART yaitu:

- Komunikasi *full-duplex* dengan register serial untuk penerima dan pengirim data;
- Beroperasi pada mode sinkron dan asinkron;
- Pada mode operasi sinkron *clock* berasal dari *Master* atau *Slave*;
- Memiliki pembangkit *baud rate* beresolusi tinggi;
- Pengiriman data serial dengan format *frame* 5, 6, 7, 8, dan 9 bit dan 1 atau 2 bit stop;
- Dukungan perangkat keras terhadap pengecekan paritas genap atau paritas ganjil;
- Pendeteksi kesalahan pada format *frame* data yang dikirim;
- Pendeteksi pengiriman kelebihan data;
- Memiliki 3 layanan interupsi yaitu *TX Complete*, *TX Data Register Empty*, dan *RX Complete*;
- Mode komunikasi multi prosesor;
- Dua kali kecepatan transfer pada komunikasi mode asinkron.

USART *baud rate* Register (UBRR) untuk menentukan kecepatan transfer data pada komunikasi serial dapat dihitung menggunakan persamaan yang terdapat dalam Tabel 2.1.

Tabel 2.5.Persamaan untuk menghitung pengaturan register *baud rate*

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{osc}}{16(UBRR+1)}$	$UBRR = \frac{f_{osc}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{osc}}{8(UBRR+1)}$	$UBRR = \frac{f_{osc}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{osc}}{2(UBRR+1)}$	$UBRR = \frac{f_{osc}}{2BAUD} - 1$

Sumber: Atmel, 2007: 169

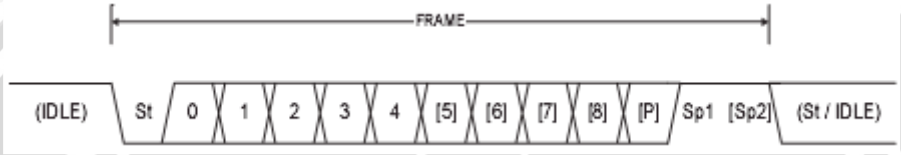
Keterangan :

BAUD : *Baud rate* dalam bit per secon (bps).

f_{OSC} : Frekuensi *clock* dari sistem osilator.

UBRR : Register *baud rate* yang terdiri atas Register UBRRL dan Register UBRRH.

Format *frame* data komunikasi serial didefinisikan sebagai sebuah karakter dari bit-bit data yang memiliki bit-bit sinkronisasi yaitu start bit dan stop bit, serta pilihan berupa bit paritas untuk mendeteksi terjadinya kesalahan pengiriman data. Format *frame* pengiriman data serial USART ditunjukkan dalam Gambar 2.7.



Gambar 2.12 Format *frame* pengiriman data serial USART
Sumber: Atmel, 2007: 171

Keterangan :

St : Bit start selalu berlogika rendah.

(n) : Bit data (0-8).

P : Bit paritas genap atau paritas ganjil.

Sp : Bit stop selalu berlogika tinggi (bit stop bisa berjumlah 1 atau 2).

IDLE : Tidak ada data yang ditransfer pada RX dan TX, IDLE selalu berlogika tinggi

2.7 Metode Pendeteksi Kesalahan Transmisi Data Menggunakan *Checksum*

Tujuan metode deteksi kesalahan adalah untuk membuat penerima mampu untuk memeriksa apakah data yang ditransmisikan melalui sebuah media yang rawan terhadap *noise* benar atau tidak. Untuk melakukan hal ini, pemancar memilih suatu nilai (disebut sebagai *checksum*) yang kemudian mengkombinasikannya dengan data yang dikirim menggunakan algoritma tertentu. Dengan algoritma yang sama pula, maka penerima melakukan perhitungan terhadap data yang diterima untuk dibandingkan dengan *checksum* dan pada akhirnya dapat diketahui apakah data yang diterima benar atau tidak.

Sebagai contoh deteksi kesalahan (*checksum*) sederhana adalah sebagai berikut:

Data: 6 23 4

Data dengan checksum: 6 23 4 33

Data setelah transmisi: 6 27 4 33

Keterangan: lebar 1 *byte* data adalah 0-256

Dari contoh di atas, *byte* kedua dari data yang diterima rusak/salah karena nilainya berubah menjadi 27. Akan tetapi penerima dapat mendeteksinya dengan membandingkan *checksum* (33) yang diterima dengan *checksum* perhitungan yakni 37 (6+27+4). Namun ketika *checksum* itu sendiri mengalami kerusakan, akan memungkinkan data yang seharusnya benar dianggap sebagai sebuah kesalahan. Hal yang paling fatal adalah bila data dan *checksum* kesemuanya mengalami kerusakan/kesalahan. Hal ini menyebabkan proses transmisi data menjadi tidak konsisten. Seluruh kemungkinan di atas memang tidak dapat dihindari, cara terbaik yang dapat dilakukan adalah memperkecil kemungkinan dengan menambah jumlah data pada *checksum* (sebagai contoh, memperlebar data *checksum* dari 1 *byte* menjadi 2 *byte*).

Dari contoh perhitungan *checksum* di atas dengan lebar data 1 *byte*, terdapat 1/256 kemungkinan *error* tidak dapat terdeteksi. Ketika lebar data kita ganti menjadi 2 *byte* (0-65536) maka kemungkinan terjadi *error* akan lebih kecil menjadi 1/65535. Namun tetap saja hal ini tidaklah cukup karena metode *checksum* di atas masih terlalu sederhana dan tidak cukup acak. *Checksum* tidak dapat mendeteksi error yang terjadi pada banyak paket (*byte* data) dan tidak dapat mendeteksi error pada *checksum* yang diberikan itu sendiri.

BAB III
METODE PENELITIAN

Sistem *downloader* mikrokontroler AVR melalui media r*adio frequency* ini terdiri atas dua bagian yaitu sisi pemrogram dan sisi target. Sistem yang dibahas pada laporan skripsi adalah sistem pada sisi target. Metode penelitian yang digunakan dalam penyusunan skripsi ini adalah sebagai berikut:

1) Perancangan dan Pembuatan alat
2) Pengujian alat

3.1 Perancangan dan Pembuatan Alat

Tahap perancangan alat meliputi penentuan spesifikasi sistem yang dirancang, penyusunan blok diagram sistem serta pembuatan skema rangkaian. Pemilihan komponen perangkat keras berdasarkan komponen yang mudah diperoleh di pasaran lokal. Perancangan perangkat lunak dilakukan dengan membuat diagram alir untuk program utama dan sub program.

Selanjutnya pada tahap pembuatan meliputi pembuatan desain *layout* PCB dan pembuatan PCB, pengujian komponen, pengeboran PCB, perakitan, penyolderan komponen pada PCB, pengujian awal serta pembuatan perangkat lunak. Tahap pembuatan perangkat lunak meliputi penulisan kode, pengujian (*debugging*), dan kompilasi program menggunakan s*oftware CodeVision AVR* versi 1.25.7 Profesional.

3.2 Pengujian Alat

Pengujian dilakukan dalam tiga tahap, yaitu pengujian perangkat keras, pengujian perangkat lunak serta pengujian sistem secara keseluruhan. Beberapa aspek pengujian yang dilakuan adalah sebagai berikut:

1) Pengujian perangkat keras.

   Pengujian perangkat keras meliputi pengujian rangkaian pemancar dan penerima modul sensor r*adio frequency*, transmisi data serial menggunakan *radio frequency*, SRAM eksternal serta pengujian antarmuka SPI (*Serial Peripheral Interface*).

2) Pengujian perangkat lunak.

   Pengujian perangkat lunak meliputi pengujian protokol transmisi data serial melalui r*adio frequency* dengan sisi pemrogram, akses data pada SRAM eksternal, pemrograman ISP (*In-System Programming*) menggunakan antarmuka SPI.

3) Pengujian sistem secara keseluruhan.

   Pengujian sistem secara keseluruhan bertujuan untuk mengetahui kemampuan sistem untuk melakukan proses *serial downloading* mikrokontroler target melalui media r*adio frequency*. Selain itu pengujian terhadap sistem secara keseluruhan juga bertujuan untuk mengetahui kesesuaian sistem dengan spesifikasi sistem yang diharapkan.

BAB IV
PERANCANGAN DAN PEMBUATAN ALAT

Perancangan dan pembuatan alat ini terdiri atas dua bagian yaitu perancangan dan pembuatan perangkat keras serta perancangan dan pembuatan perangkat lunak. Beberapa aspek lain yang perlu dijelaskan dalam bab perancangan dan pembuatan alat ini adalah penentuan spesifikasi sistem yang dirancang, blok diagram sistem serta diagram alir algoritma perangkat lunak yang digunakan.

4.1 Spesifikasi Alat

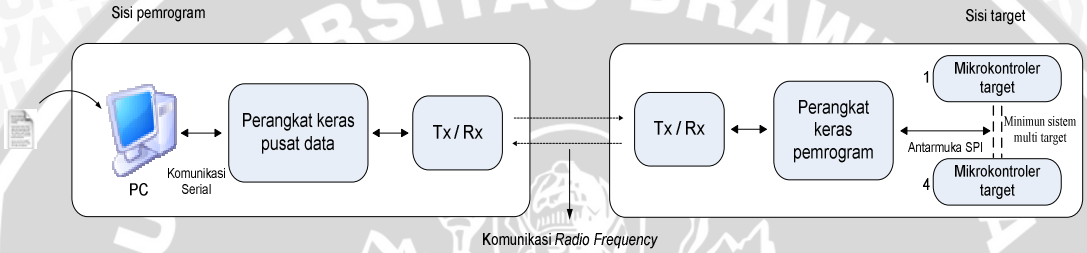
Spesifikasi alat yang terdapat pada sisi target dirancang sebagai berikut :

a. Pusat kendali proses komunikasi data serial dan proses *serial downloading* menggunakan mikrokontroler ATmega162.
b. Perangkat keras pemancar dan penerima menggunakan modul *radio frequency* YS-1020.
c. Komunikasi data serial menggunakan protokol USART dengan *baud rate* 9600 bps.
d. Alat dirancang menggunakan SRAM eksternal dengan kapasitas 32K *byte* seri UT62256CPC untuk menyimpan data sementara.
e. Antarmuka mikrokontroler dengan SRAM eksternal menggunakan octal D-Latch 74HC373.
f. Antarmuka mikrokontroler dengan mikrokontroler target dengan menggunakan rangkaian minimum sistem multi target.
g. Mikrokontroler target adalah mikrokontroler dari keluarga Atmel AVR yang menggunakan metode pemrograman ISP (*In-System Programming*) dengan antarmuka SPI (*Serial Peripheral Interface*).
h. Mampu berkomunikasi dengan empat mikrokontroler target secara bergantian.
i. Menggunakan catu daya DC dengan tegangan sebesar 5 volt.

4.2 Perancangan Sistem

4.2.1 Perancangan Sistem Secara Keseluruhan

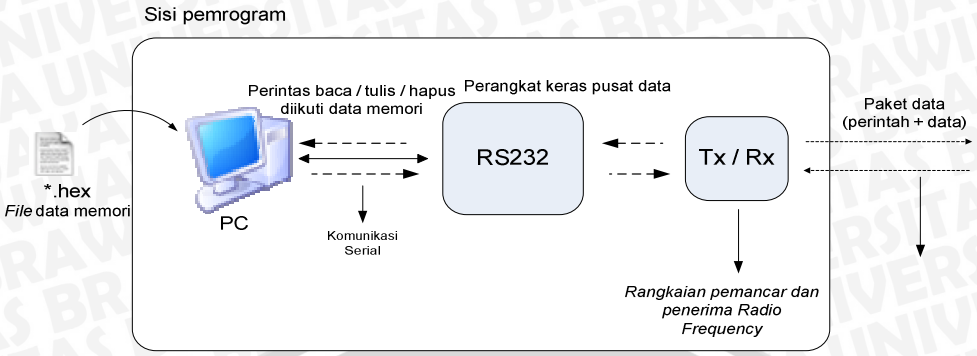
Sistem keseluruhan terdiri atas dua bagian. Bagian pertama adalah sisi pemrogram yang mencakup perangkat lunak PC (*Personal Computer*) dan perangkat keras sisi pemrogram dengan antarmuka serial. Bagian kedua adalah sisi target yang mencakup perangkat lunak dan perangkat keras sistem target dengan menggunakan antarmuka SPI (*Serial Peripheral Interface*) untuk proses *serial downloading* mikrokontroler target. Gambar 4.1 menunjukkan ilustrasi blok diagram sistem secara keseluruhan.



Gambar 4.1 Blok diagram sistem secara keseluruhan

Penjelasan prinsip kerja sistem secara keseluruhan adalah sebagai berikut: Perancangan sistem pada bagian sisi pemrogram terdiri atas:

a. Perangkat lunak pada PC, bertindak sebagai antarmuka dengan pengguna untuk mengatur seluruh proses operasi memori (baca, tulis atau hapus memori) yang ditujukan ke mikrokontroler target.
b. Perangkat lunak pada PC, bertindak untuk memilih mikrokontroler target mana yang akan dituju untuk melakukan proses operasi memori tersebut.
c. Pada perangkat keras sisi pemrogram terdapat perangkat lunak yang berfungsi untuk proses transfer data-data (data perintah maupun data memori) dari/ke PC untuk diteruskan/diterima melalui komunikasi r*adio frequency* dari/ke sisi target. Aliran data pada sisi pemrogram ditunjukkan dalam Gambar 4.2.
d. Perangkat keras r*adio frequency* terdiri atas dua bagian, pemancar dan penerima. Komunikasi data berlangsung dua arah secara serial pada bagian sisi pemrogram.



Gambar 4.2 Blok diagram aliran data pada sisi pemrogram

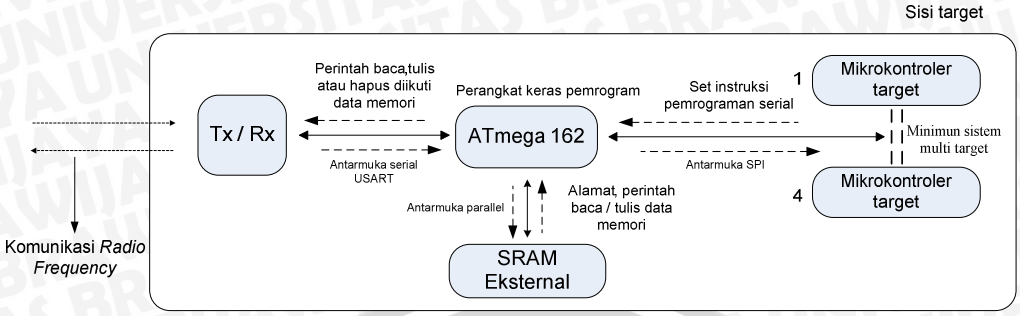
Perancangan sistem pada sisi target terdiri atas:

a. Di dalam mikrokontroler pada perangkat keras sisi target terdapat perangkat lunak yang berfungsi untuk proses transfer data (data perintah dan data memori) dari/ke sisi pemrogram melalui komunikasi r*adio frequency*, menampung sementara data transmisi ke SRAM eksternal serta mengolah data tersebut untuk proses *serial downloading* pada mikrokontroler target. Aliran data pada sisi target dapat dilihat dalam Gambar 4.3

b. Perangkat keras r*adio frequency* terdiri atas dua bagian, yaitu bagian pemancar dan bagian penerima r*adio frequency*. Perangkat keras r*adio frequency* berfungsi sebagai media pendukung komunikasi dengan menggunakan r*adio frequency*. Komunikasi data serial berlangsung dua arah secara *half-duplex* dengan memanfaatkan fasilitas serial USART pada mikrokontroler sisi target.

c. SRAM eksternal digunakan sebagai media penyimpan data sementara untuk proses *serial downloading*. Data yang disimpan di dalam SRAM eksternal merupakan data program yang merupakan kode heksa untuk mikrokontroler keluarga Atmel AVR.

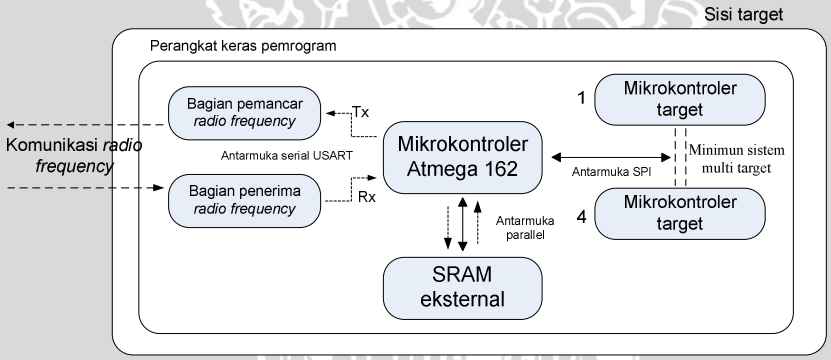
d. Mikrokontroler target yang digunakan untuk proses *serial downloading* adalah jenis mikrokontroler dari keluarga Atmel AVR. Bagian ini bukan merupakan bagian yang utama dari sistem akan tetapi turut ditampilkan sebagai sarana pendukung untuk menjelaskan sistem secara utuh.



Gambar 4.3 Blok diagram aliran data pada sisi target

4.2.2 Perancangan Sistem Sisi Target

Penelitian ini dibagi menjadi dua bagian, sisi pemrogram dan sisi target. Topik yang dibahas pada skripsi ini adalah sisi target. Sistem yang terdapat pada sisi target terdiri atas perangkat keras pemancar dan perangkat keras penerima *radio frequency*, mikrokontroler Atmel AVR ATmega162 sebagai pusat kendali, SRAM eksternal untuk media penyimpan data sementara, serta mikrokontroler target seperti terlihat dalam Gambar 4.4



Gambar 4.4 Blok diagram pada sisi target

Pada sistem sisi target, mikrokontroler ATmega162 berfungsi sebagai pusat kendali bertugas untuk menangani proses komunikasi data antara sisi pemrogram dengan sisi target, mengakses SRAM eksternal untuk menyimpan data sementara, serta mengendalikan proses pemrograman serial pada mikrokontroler target.

Penanganan terhadap proses komunikasi data antara sisi pemrogram dengan sisi target menggunakan media *radio frequency* dikendalikan oleh mikrokontroler ATmega162. Mikrokontroler menyediakan fasilitas komunikasi data serial USART beserta fasilitas pendukung USART. USART merupakan protokol komunikasi data serial yang mengirimkan/menerima data dalam bentuk paket-paket data. Setiap paket data serial dimulai dengan satu bit *start*, lalu diikuti

sejumlah n bit data (digunakan 8 bit data dan diakhiri dengan dua bit *stop*. Dalam perangkat lunak yang terdapat pada mikrokontroler dilengkapi dengan algoritma pendeteksi kesalahan menggunakan *checksum*.

Data yang dikirimkan oleh mikrokontroler pada sisi pemancar dipancarkan melalui unit pemancar *radio frequency*. Pada unit penerima *radio frequency*, sinyal yang diterima dari unit pemancar akan dibaca oleh modul YS-1020 yang nantinya diterima dan diproses oleh mikrokontroler untuk memperoleh data yang sesungguhnya

4.3 Perancangan dan Pembuatan Perangkat Keras

Perangkat keras terdiri atas sistem mikrokontroler dan rangkaian transmisi serial *radio frequency*. Rangkaian transmisi serial r*adio frequency* terdiri atas rangkaian pemancar dan penerima *radio frequency*.

4.3.1 Sistem Perangkat Keras Mikrokontroler ATmega162

Perancangan perangkat keras sistem mikrokontroler ATmega162 adalah sebagai berikut:

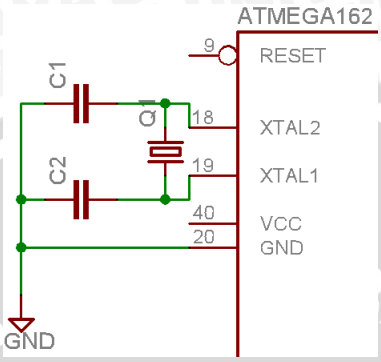
a. Sistem Minimum Mikrokontroler ATmega162 Sisi Target

Mikrokontroler ATmega162 sebagai pusat kendali berfungsi untuk menangani transmisi data serial melalui media r*adio frequency* serta pengendali proses *serial downloading* pada mikrokontroler target. Fungsional dan jalur I/O mikrokontroler yang digunakan dapat dilihat dalam Tabel 4.1.

Tabel 4.1 Penggunaan pin-pin I/O mikrokontroler ATmega162

No	Pin	Nama	Fungsi
1	32-39	PA0-PA7	Bus data (D0-D7) dan alamat (A0-A7) untuk akses SRAM eksternal.
2	5-8	PB4-[B7	Jalur antarmuka SPI (*serial peripheral inter face*).
3	21-28	PC0-PC7	Bus alamat (A8-A15) Untuk akses SRAM eksternal.
4	10	PD0	Pin RX0 terhubung dengan rangkaian penerima RF.
5	11	PD1	Pin TX0 terhubung dengan rangkaian pemancar RF.
6	12	PD2	Terhubung dengan ping *output enable* pada octal D-latch 74HC373
7	13	PD3	Driver LED indikator.
8	14	PD6	Terhubung dengan pin *write enable* pada SRAM eksternal untuk menulis data.
9	15	PD7	Terhubung dengan pin *output enable* pada SRAM eksternal untuk membaca data
10	31	PE0	Driver LED indikator
11	30	PE1	Terhubung dengan ping *latch enable* pada octal D-latch 74HC373

Mikrokontroler membutuhkan rangkaian tambahan sebagai sumber *clock*. Alat yang dirancang ini akan menggunakan osilator eksternal. Sumber *clock* dari kristal 11,0592 MHz dihubungkan ke XTAL1 dan XTAL2 mikrokontroler ATmega162. Rangkaian osilator ditunjukkan dalam Gambar 4.5.



Gambar 4.5 Rangkaian osilator pada mikrokontroler ATmega162

Nilai kapasitansi kapasitor C_1 dan C_2 yang digunakan adalah sebesar 16 pF. Nilai tersebut sesuai dengan nilai kapasitansi yang direkomendasikan dalam *datasheet* mikrokontroler ATmega162 untuk frekuensi kristal 11,0592 MHz, yang berkisar antara 12-22 pF.

a. USART Mikrokontroler ATmega162

Pada perancangan komunikasi data serial USART, fasilitas Tx dan Rx yang terdapat pada USART diaktifkan untuk mendukung komunikasi data serial secara *half-duplex*. USART dioperasikan pada mode asinkron normal.

Frame data serial yang digunakan terdiri atas 8 bit data dan 1 bit stop. Hal yang penting untuk diperhatikan adalah *baud rate* USART menyeseuaikan dengan *baud rate* perangkat keras r*adio frequency* yaitu 9600 bps. Pengaturan *baud rate* dilakukan dengan mengatur register UBRR yang nilainya dapat dihitung menggunakan rumus berikut:

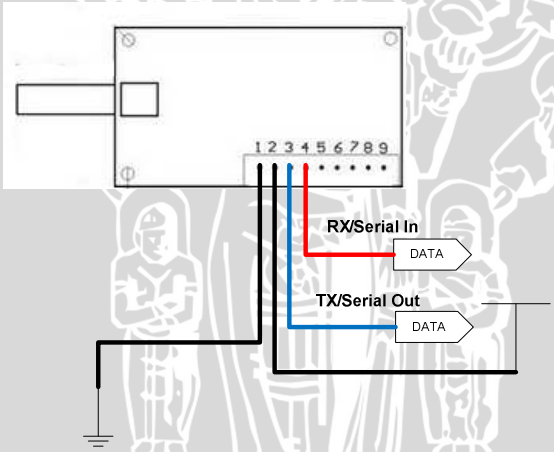
$$UBRR = \frac{f_{OSC}}{16BAUD} - 1$$

Jika frekuensi sistem *cloc*k yang digunakan adalah 11,0592 MHz dan *baud rate* yang dikehendaki adalah 9600 bps maka nilai UBRR adalah 0x0047.

4.3.2 Rangkaian Perangkat Keras *Radio Frequency*

Perancangan rangkaian pemancar dan penerima RF menggunakan modul RF YS-1020 yang dalam satu modul terdapat pemancar dan penerima data. Data yang dikirim oleh bagian pemancar yang menggunakan modul RF YS-1020 akan didemodulasi oleh modul YS-1020 kedua sehingga diperoleh sinyal data. Keluaran dan pembacaan data dari modul RF ini yaitu berupa data digital sehingga nantinya dapat dikomunikasikan dengan sistem komunikasi data serial.

Modul YS-1020 menggunakan modulasi didital GFSK (*Gaussian frequency Shift Keying*). Modul ini memiliki frekuensi kerja yang beragam yaitu 433,92 MHz, 450 MHz, 868 MHz. Menggunakan tegangan catu daya DC dengan range antara 3.3–5.0 volt. Mempunyai daya output RF sebesar ≤10mW. Kecepatan data serial *typical* yang ditransmisikan adalah sebesar 9,6 kbps sesuai yang tercantum dalam *datasheet* (Shenzen Yishi, 2009). Gambar 4.6 menunjukkan rangkaian penerima RF.



Gambar 4.6 Rangkaian Modul RF YS-1020
Sumber: ShenZhen Yishi, 2009

4.3.3 Rangkaian Antarmuka SRAM Eksternal

Mikrokontroler ATmega162 memiliki 3 jenis memori yang berbeda yaitu Memori Program (Flash), Memori Data (SRAM) dan EEPROM. Memori Flash dan EEPROM (*Electrically Eraseable Read Only Memory*) bersifat *nonvolatile* sedangkan SRAM (*Static Random Access Memory*) bersifat *volatile* yang berarti data yang tersimpan di dalam SRAM akan hilang ketika catu daya mikrokontroler ATmega162 dihilangkan.

Mikrokontroler ATmega162 memiliki SRAM internal dengan kapasitas 1 k*byte* serta memiliki fitur untuk mengakses SRAM eksternal dengan kapasitas sampai dengan 64K *byte*. Pada perancangan sistem ini, SRAM internal digunakan untuk menyimpan nilai variabel global, alokasi memori dinamis pada saat program dieksekusi serta dialokasikan untuk *data stack*. Sementara itu spesifikasi sistem yang dirancang untuk menganani data dalam ukuran besar menyebabkan sistem ini perlu dilengkapi dengan SRAM eksternal untuk menampung data tersebut.

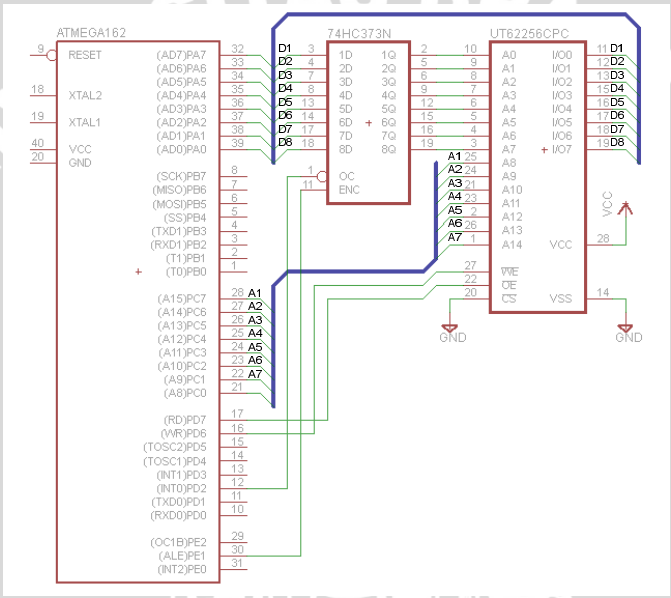
Data yang tersimpan di dalam SRAM eksternal adalah data transmisi antara sisi pemrogram dengan sisi target. Data tersebut merupakan kode heksa yang spesifik untuk/dari suatu devais mikrokontroler yang terpasang pada sistem *downloader* sebagai mikrokontroler target. Seluruh proses *serial downloading* yang melibatkan operasi baca/tulis memori flash maupun EEPROM mikrokontroler target selalu melalui tahap penyimpanan data pada SRAM eksternal.

Mikrokontroler ATmega162 memiliki bus data (D0-D7) dan bus alamat (A0-A7) yang berada dalam PortA sehingga untuk dapat mengakses SRAM eksternal perlu komponen Octal D-Latch agar PortA dapat berfungsi sebagai bus data dan bus alamat secara bergantian. Proses pergantian fungsi PortA sebagai bus data maupun bus alamat diatur oleh sinyal-sinyal kontrol yang dikendalikan oleh mikrokontroler ATmega162.

Sistem *downloader* ini dirancang menggunakan 32k x 8 bit CMOS SRAM seri UT62256CPC yang memiliki kecepatan akses tinggi serta tingkat konsumsi daya yang rendah dengan tegangan kerja berkisar antara 2 sampai dengan 5 V. Beberapa data yang merupakan spesifikasi SRAM yang tercantum di dalam *datasheet* (Utron, 2001) adalah kecepatan akses maksimal 70 ns, arus kerja 50 mA serta nilai beberapa parameter dalam kondisi pengujian (V_{CC} = 5 V, T = 25 ^{O}C) diantaranya adalah V_{OL} maksimal bernilai 0,4 V, V_{OH} minimal bernilai 2,4 V, I_{OL} bernilai 2,1 mA dan I_{OH} yang bernilai – 1,0 mA. Sedangkan Octal D-Latch menggunakan IC 74HC373 yang memiliki kecepatan akses tinggi serta memiliki nilai tipikal untuk beberapa parameter dalam kondisi pengujian pada *datasheet* (V_{CC} = 5 V, T = 25 ^{O}C) yaitu *propagation delay* antara sinyal data masukan (D_n)

dengan sinyal data keluaran (Q_n) adalah 12 ns dan *propagation delay* antara sinyal *Latch Enable* (LE) dengan sinyal data keluaran (Q_n) adalah 15 ns.

Konfigurasi pin yang terdapat di dalam SRAM seri UT62256CPC terdiri atas pin I/O1- I/O8, pin A0-A14, pin $\overline{WE}$, pin $\overline{OE}$, pin $\overline{CS}$, pin V_{CC} dan pin V_{SS}. Sedangkan konfigurasi pin Octal D-Latch 74HC373 terdiri atas pin D0-D7, pin Q0-Q7, pin $\overline{OE}$, pin LE, pin V_{CC} dan pin V_{SS}. Hubungan antara pin yang terdapat pada SRAM seri UT62256CPC, Octal D-Latch 74HC373 dengan pin yang terdapat pada mikrokontroler ATmega162 membentuk suatu rangkaian antarmuka SRAM eksternal seperti yang terlihat dalam Gambar 4.7.



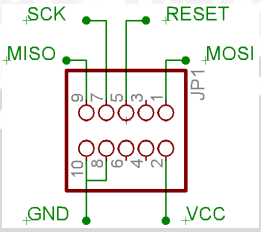
Gambar 4.7 Rangkaian antarmuka mikrokontroler ATmega162 dengan SRAM eksternal

4.3.4 Antarmuka Sistem Mikrokontroler

Sistem mikrokontroler yang terdapat pada sistem perangkat keras sisi target terdiri atas sistem mikrokontroler utama yang berfungsi sebagai pusat kendali seluruh proses yang terjadi serta sistem mikrokontroler target sebagai obyek untuk proses *serial downloading* yang dilakukan oleh mikrokontroler utama. Antarmuka yang digunakan pada kedua sistem mikrokontroler adalah antarmuka SPI (*Serial Peripheral Interface*).

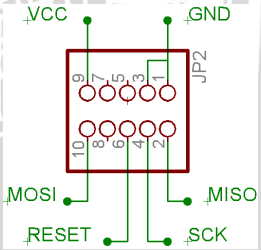
Interaksi antara mikrokontroler utama ATmega162 dengan mikrokontroler target pada sisi target menggunakan antarmuka SPI (*Serial Peripheral Interface*) untuk *serial downloading*. Antarmuka SPI pada mikrokontroler ATmega162 terdiri atas tiga jalur komunikasi yaitu Serial *Clock* (SCK), *Master In - Slave Out*

(MISO) dan *Master Out - Slave In* (MOSI). Jenis *programmer* yang dipakai adalah Kanda Sistem STK200/300 yang merupakan salah satu jenis *AVR Chip Programmer* standart Atmel AVR yang memiliki konfigurasi konektor seperti yang terlihat dalam Gambar 4.8



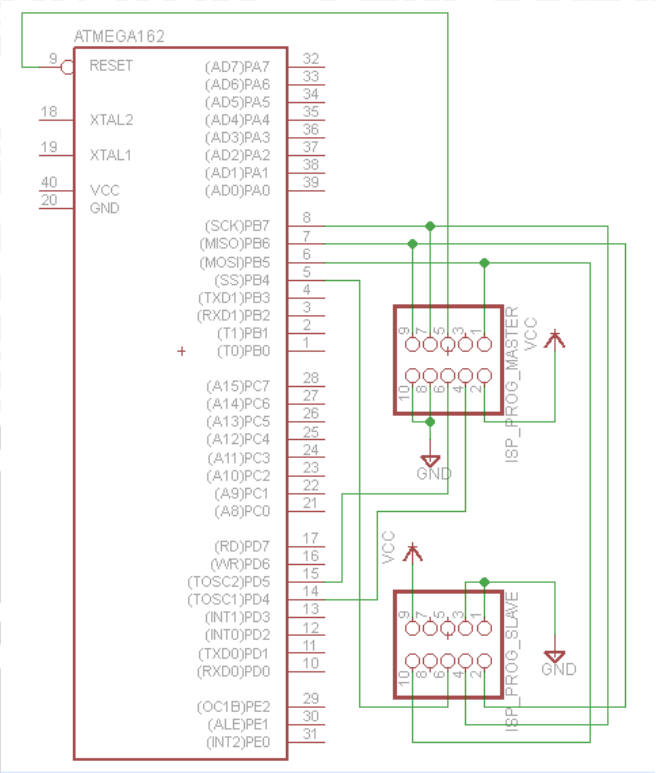
Gambar 4.8 Konektor standart Kanda Sistem STK200/300

Proses *serial downloading* yang dilakukan oleh mikrokontroler ATmega162 pada mikrokontroler target sebagai obyek menggunakan antarmuka SPI (*Serial Peripheral Interface*) dan tambahan jalur *reset* untuk mereset mikrokontroler target dengan menggunakan metode ISP (*In-System Programming*). Tipe konektor yang dipakai pada sistem mikrokontroler target adalah Kanda Sistem STK200/300 untuk memudahkan antarmuka sistem *downloader* dengan mikrokontroler target. Oleh karena itu, konektor yang terdapat pada sistem *downloader* sebagai penghubung dengan mikrokontroler target menggunakan konektor hasil pencerminan konektor standart Kanda Sistem STK200/300. Konfigurasi konektor yang merupakan hasil pencerminan konektor standart Kanda Sistem STK200/300 ditunjukkan dalam Gambar 4.9.



Gambar 4.9 Pencerminan konektor STK200/300

Konfigurasi pin konektor untuk mikrokontroler target terhubung dengan pin-pin SPI pada port B sedangkan pin reset terhubung dengan pin SS. Rangkaian antarmuka sistem target dengan mikrokontroler target ditunjukkan dalam Gambar 4.10.



Gambar 4.10 Rangkaian antarmuka mikrokontroler pada sistem *downloader*

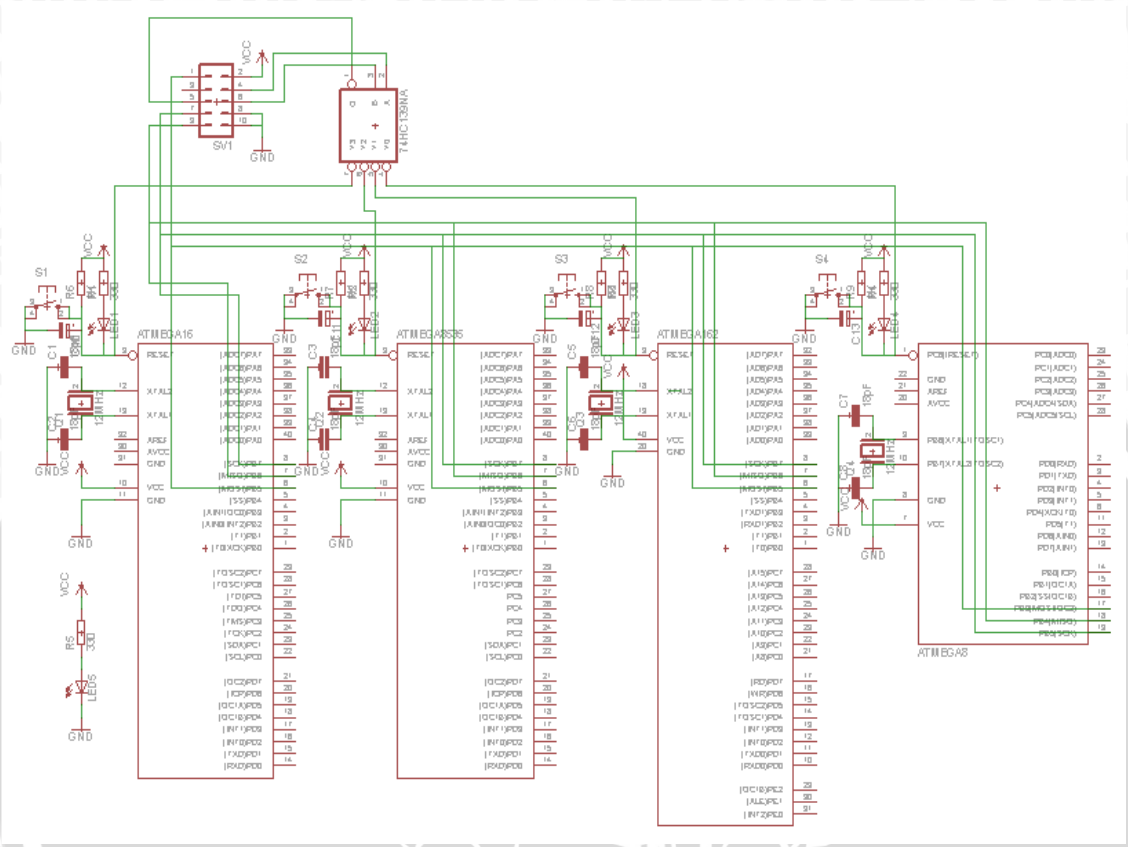
4.3.5 Antarmuka Sistem Downloader Dengan Sistem mikrokontroler target

Sistem perangkat keras mikrokontroler target sebagai obyek untuk proses *serial downloading* yang dilakukan oleh mikrokontroler utama terdapat mode pemilihan mikrokontroler, hal ini digunakan ketika beroperasi pemilihan mikrokontroler oleh sisi pemrogram. Komunikasi antara mikrokontroler utama dengan mikrokontroler target untuk proses *serial downloading* menggunakan metode ISP (*Serial Peripheral Interface*) dengan manggunakan antarmuka SPI (*Serial Peripheral Interface*) dan tambahan jalur *reset* untuk mereset mikrokontroler target.

Antarmuka sistem downloader dengan mikrokontroler target dirancang dengan menggunakan decoder 74HC139N, konfigurasi pin 74HC139N terdiri dari ping A0-A1, pin Y0-Y3, pin $\overline{E}$, pin V_{CC} dan pin Gnd. Tegangan kerja dari 74HC139N berkisar antara 2 sampai dengan 6 volt dengan arus kerja 50mA, nilai *input rise and fall time* maksimal 400nS sesuai dengan yang tercantum dalam *datasheet* (Fairchild semiconductor, 2003).

Mode pemilihan mikrokontroler tersebut memanfaatkan proses dari metode ISP (*Serial Peripheral Interface*) yaitu mikrokontoler terpilih direset yang

nantinya dapat dilakukan proses *serial downloading*. Antar muka sistem downloader dengan sistem mikrokontroler target ditunjukkan dalam gambar 4.11

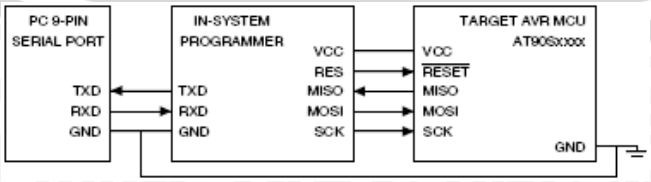


Gambar 4.11 Rangkaian antarmuka *downloader* dengan sistem mikrokontroler target

4.4 Perancangan Perangkat Lunak

4.4.1 Perancangan Perangkat Lunak *In-System Programming*

ISP (*In-System Programming*) adalah salah satu metode untuk memprogram semua *non-volatile* memori pada mikrokontroler AVR menggunakan antarmuka SPI. Antarmuka SPI ditunjukkan dalam Gambar 4.12 sedangkan fungsi keenam jalur antarmuka SPI terdapat dalam Tabel 4.2. Bagian *In-System Programmer* disebut sebagai *master* sedangkan mikrokontroler target disebut sebagai *slave*.



Gambar 4.12 Blok diagram antarmuka pada ISP
Sumber: Atmel, 2002b: 2

Tabel 4.2 Fungsi jalur antarmuka SPI

Pin	Name	Comment
SCK	Serial Clock	Programming clock, generated by the In-System Programmer (Master)
MOSI	Master Out – Slave In	Communication line from In-System Programmer (Master) to target AVR being programmed (Slave)
MISO	Master In – Slave Out	Communication line from target AVR (Slave) to In-System Programmer (Master)
GND	Common Ground	The two systems must share the same common ground
RESET	Target AVR MCU Reset	To enable In-System Programming, the target AVR Reset must be kept active. To simplify this, the In-System Programmer should control the target AVR Reset
V_{CC}	Target Power	To allow simple programming of targets operating at any voltage, the In-System Programmer can draw power from the target. Alternatively, the target can have power supplied through the In-System Programming connector for the duration of the programming cycle

Sumber: Atmel, 2002b: 2

Perangkat lunak dirancang untuk menangani operasi memori mikrokontroler target menggunakan instruksi pemrograman serial untuk semua mikrokontroler AVR. Algoritma pemrograman serial yang digunakan sesuai dengan algoritma pemrograman serial SPI di dalam *datasheet* mikrokontroler AVR pada bagian *memory programming* menggunakan metode *serial downloading*.

Algoritma pemrograman serial SPI adalah sebagai berikut:

1. Pin $\overline{\text{RESET}}$ mikrokontroler *slave* (selanjutnya disebut dengan *slave*) diberi logika tinggi minimal selama dua siklus *clock* setelah SCK diberi logika rendah.
2. Pin $\overline{\text{RESET}}$ *slave* diberi logika rendah dan tunggu waktu tunda minimal 20 ms kemudian mikrokontroler *master* (selanjutnya disebut dengan *master*) mengirimkan instruksi *Programming Enable* yang memungkinkan untuk melakukan pemrograman serial SPI pada *slave* melalui pin MOSI.
3. Setelah instruksi *Programming Enable* berhasil dikirimkan maka terjadi sinkronisasi antara *master* dan *slave*. Ketika terjadi sikronisasi, *slave* akan merespon *master* dengan mengirimkan *byte* data yang dikirimkan oleh *master* pada pengiriman *byte* data berikutnya. Apabila *master* dan *slave* kehilangan sinkronisasi, maka pin $\overline{\text{RESET}}$ *slave* diberi logika tinggi dan proses pemrograman serial dimulai kembali dari awal.
4. Pada pemrograman memori Flash, data ditulis pada halaman memori Flash secara bersamaan (*writing page mode*). Setiap satu *byte* data diisikan pada halaman memori Flash dengan cara *master* mengirimkan alamat dan data

secara bersamaan menggunakan instruksi *Load Program Memory Page* pada *slave*. Data *low byte* harus diisikan terlebih dahulu pada halaman memori Flash sebelum data *high byte* untuk memastikan kebenaran data yang ditulis pada halaman memori Flash. Waktu tunda minimal untuk proses penulisan halaman memori Flash sebelum melakukan penulisan halaman selanjutnya adalah t_{WD_FLASH} yang nilainya terdapat di dalam Tabel 4.3.

5. Penulisan EEPROM dilakukan pada setiap *byte* data (*writing byte mode*) dengan cara *master* mengirimkan alamat dan data secara bersamaan menggunakan instruksi *Write EEPROM* pada *slave*. Data pada alamat memori EEPROM secara otomatis akan dihapus sebelum ditulis dengan data yang baru. Waktu tunda minimal untuk proses penulisan satu *byte* memori EEPROM sebelum melakukan penulisan alamat memori selanjutnya adalah t_{WD_EEPROM} yang nilainya terdapat di dalam Tabel 4.3.
6. Waktu tunda pada operasi penulisan memori Flash maupun EEPROM dapat diminamalisir dengan cara menerapkan metode *polling* pada operasi tulis memori. Akan tetapi metode *polling* ini tidak dapat berfungsi pada penulisan data dengan nilai 0xFF sehingga harus tetap menunggu waktu tunda minimal sebelum melanjutkan ke proses penulisan selanjutnya. Selain waktu tunda minimal yang harus ditunggu untuk operasi penulisan memori Flash maupun memori EEPROM, di dalam Tabel 4.3 terdapat pula waktu tunda yang harus dipenuhi untuk operasi hapus memori chip maupun operasi tulis konfigurasi Fuse bit.

Tabel 4.3 Waktu tunda minimal untuk menulis lokasi memori

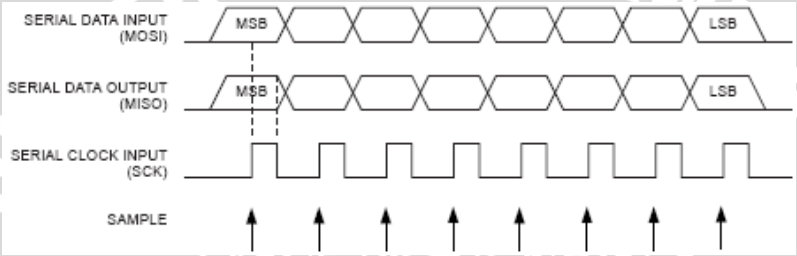
Symbol	Minimum Wait Delay
t_{WD_FLASH}	4.5 ms
t_{WD_EEPROM}	9.0 ms
t_{WD_ERASE}	9.0 ms
t_{WD_FUSE}	4.5 ms

Sumber: Atmel, 2007: 247

7. Seluruh data yang terdapat pada suatu lokasi memori dapat dibaca dengan menggunakan instruksi *Read* yang akan memberikan nilai pengembalian dari *slave* pada pin MISO.

8. Pada tahap akhir pemrograman, pin $\overline{\text{RESET}}$ *slave* diberi logika tinggi agar *slave* dapat beroperasi pada mode normal.

Beberapa persyaratan yang harus dipenuhi agar proses *serial downloading* dapat berfungsi dengan baik adalah perancangan perangkat lunak untuk operasi memori harus memenuhi diagram pewaktuan pemrograman serial SPI seperti yang terdapat dalam Gambar 4.13. Berdasarkan tuntutan spesifikasi ini maka pada perancangan komunikasi data SPI, MSB (*Most Significant Bit*) data SPI dikirimkan terlebih dahulu. Selain itu SCK juga diatur agar kondisi *idle* terjadi pada saat logika rendah serta pengambilan sampel data SPI dilakukan pada awal tepi naik sinyal SCK.



Gambar 4.13 Diagram pewaktuan pemrograman serial SPI
Sumber: Atmel, 2007: 247

Implementasi pemrograman serial SPI dapat dilakukan dengan mengirimkan instruksi sesuai dengan set intruksi pemrograman serial yang terdapat di dalam Tabel 4.4. Instruksi *Programming Enable* harus dieksekusi terlebih dahulu sampai terjadi sinkronisasi antara *master* dengan *slave* sehingga set intruksi pemrograman serial yang lain dapat dieksekusi dengan baik. Pada akhir proses, sistem harus keluar dari mode pemrograman agar *slave* dapat beroperasi pada mode normal.

Tabel 4.4 Set instruksi pemrograman serial SPI

Instruction	Instruction Format				Operation
	Byte 1	Byte 2	Byte 3	Byte4	
Programming Enable	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	Enable SPI Serial Programming after RESET goes low.
Chip Erase	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	Chip Erase EEPROM and Flash.
Read Program Memory	0010 **H**000	00**aa aaaa**	**bbbb bbbb**	**oooo oooo**	Read **H** (high or low) data **o** from Program memory at word address **a**:**b**.
Load Program Memory Page	0100 **H**000	00xx xxxx	xx**bb bbbb**	**iiii iiii**	Write **H** (high or low) data **i** to Program Memory page at word address **b**. Data low byte must be loaded before Data high byte is applied within the same address.
Write Program Memory Page	0100 1100	00**aa aaaa**	**bb**xx xxxx	xxxx xxxx	Write Program Memory Page at address **a**:**b**.
Read EEPROM Memory	1010 0000	00xx xx**aa**	**bbbb bbbb**	**oooo oooo**	Read data **o** from EEPROM memory at address **a**:**b**.
Write EEPROM Memory (byte access)	1100 0000	00xx xx**aa**	**bbbb bbbb**	**iiii iiii**	Write data **i** to EEPROM memory at address **a**:**b**.
Load EEPROM Memory Page (page access)	1100 0001	0000 0000	0000 00**bb**	**iiii iiii**	Load data **i** to EEPROM memory page buffer. After data is loaded, program EEPROM page.
Write EEPROM Memory Page (page access)	1100 0010	00xx xx**aa**	**bbbb bb**00	xxxx xxxx	Write EEPROM page at address **a**:**b**.
Read Lock Bits	0101 1000	0000 0000	xxxx xxxx	xx**oo oooo**	Read Lock bits. “0” = programmed, “1” = unprogrammed. See Table 96 on page 231 for details.
Write Lock Bits	1010 1100	111x xxxx	xxxx xxxx	11**ii iiii**	Write Lock bits. Set bits = “0” to program Lock bits. See Table 96 on page 231 for details.
Read Signature Byte	0011 0000	00xx xxxx	xxxx xx**bb**	**oooo oooo**	Read Signature Byte **o** at address **b**.
Write Fuse Bits	1010 1100	1010 0000	xxxx xxxx	**iiii iiii**	Set bits = “0” to program, “1” to unprogram. See Table 100 on page 233 for details.
Write Fuse High Bits	1010 1100	1010 1000	xxxx xxxx	**iiii iiii**	Set bits = “0” to program, “1” to unprogram. See Table 99 on page 233 for details.
Write Extended Fuse Bits	1010 1100	1010 0100	xxxx xxxx	**xxxx xxii**	Set bits = “0” to program, “1” to unprogram. See Table 98 on page 232 for details.
Read Fuse Bits	0101 0000	0000 0000	xxxx xxxx	**oooo oooo**	Read Fuse bits. “0” = programmed, “1” = unprogrammed. See Table 100 on page 233 for details.
Read Fuse High Bits	0101 1000	0000 1000	xxxx xxxx	**oooo oooo**	Read Fuse high bits. “0” = pro-grammed, “1” = unprogrammed. See Table 99 on page 233 for details.
Read Extended Fuse Bits	0101 0000	0000 1000	xxxx xxxx	**oooo oooo**	Read Extended Fuse bits. “0” = pro-grammed, “1” = unprogrammed. See Table 98 on page 232 for details.
Read Calibration Byte	0011 1000	00xx xxxx	0000 0000	**oooo oooo**	Read Calibration Byte
Poll RDY/$\overline{\text{BSY}}$	1111 0000	0000 0000	xxxx xxxx	xxxx xxx**o**	If **o** = “1”, a programming operation is still busy. Wait until this bit returns to “0” before applying another command.

Note: 1. **a** = address high bits, **b** = address low bits, **H** = 0 – Low byte, 1 – High Byte, **o** = data out, **i** = data in, x = don’t care

Sumber: Atmel, 2007: 248

Set instruksi pemrograman serial yang digunakan adalah set instruksi untuk semua devais AVR. Meskipun demikian, hanya devais AVR yang

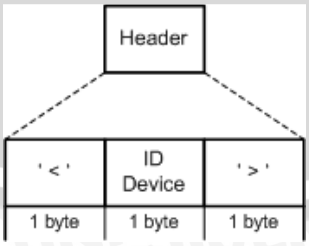
kompatibel saja yang akan merespon dengan memberikan suatu nilai pengembalian yang valid.

4.4.2 Perancangan Protokol Komunikasi Data *Radio Frequency*

Perancangan protokol komunikasi data antara sisi pemrogram dengan sisi target bertujuan untuk mengatur transmisi data secara serial melalui media *radio frequency* sehingga proses pertukaran data dapat berlangsung dengan baik. Protokol ini dibangun satu tingkat di atas protokol USART pada mikrokontroler.

Berdasarkan operasinya maka pada protokol ini terdapat tiga sesi transmisi, yaitu sesi pilih mikrokontroler target, sesi tulis target dan sesi baca target. Sesi pilih mikrokontroler target adalah saat protokol menangani proses transfer data dari sisi pemrogram ke sisi target untuk operasi pilih mikrokontroler pada minimum sistem multi target. Pada sesi tulis target, protokol menangani proses transfer data dari sisi pemrogram ke sisi target untuk operasi tulis memori. Sedangkan pada sesi baca target, protokol bertugas menangani proses transfer data dari sisi target ke sisi pemrogram untuk operasi baca memori.

Pada protokol komunikasi terdapat beberapa paket transmisi yang terdiri atas paket perintah pilih mikrokontroler, paket perintah operasi memori, paket data dan paket konfirmasi. Seluruh paket tersebut tersusun atas *header*, isi dan *checksum*, kecuali untuk paket konfirmasi. *Header* berisi data yang berfungsi sebagai awalan sebuah paket serta berisi ID devais. ID devais berfungsi sebagai identitas khusus bagi sistem sehingga hanya satu pasang perangkat keras sisi pemrogram dan sisi target saja yang akan saling mengenal ID dan dapat melakukan komunikasi data. *Frame header* dapat dilihat di dalam Gambar 4.16.



Gambar 4.14 Format *frame header* sebuah paket

Pembagian jenis paket transmisi data pada protokol komunikasi data melalui media r*adio frequency* adalah sebagai berikut:

a. Paket perintah pilih mikrokontroler.

Paket perintah pilih mikrokontroler berisi informasi mengenai kode mikrokontroler dari sisi pemrogram ke sisi target yang nantinya diproses oleh sisi target untuk memilih mikrokontroler mana yang akan dilakukan proses operasi memori pada salah satu mikrokontroler target. Format *frame* paket perintah ditunjukkan pada tabel 4.5 dan untuk ID mikrokontroler ditunjukkan pada tabel 4.6.

Tabel 4.5 Format *frame* paket perintah pilih mikrokontroler

3 *byte*	1 *byte*	1 *byte*	2 *byte*
Header	ID mikrokontroler	0x7C	*checksum*

Tabel 4.6 ID mikrokontroler dari sisi pemrogram

No.	**Nama Mikrokontroler**	**ID Mikrokontroler (HEX)**	**Panjang Data (*byte*)**
1	Mikrokontroler 1	0x01	1
2	Mikrokontroler 2	0x02	1
3	Mikrokontroler 3	0x03	1
4	Mikrokontroler 4	0x04	1

b. Paket perintah operasi memori.

Paket perintah operasi memori berisi informasi penting antara lain ID perintah, jumlah paket data yang akan ditransmisikan dan *device signature* mikrokontroler target. Pada sesi tulis target, sisi pemrogram akan mengirimkan paket perintah yang berisi data ID perintah, *device signature* dan jumlah paket data. Sedangkan pada sesi baca target, paket perintah berisi ID perintah dan *device signature*. Khusus pada sesi baca target, sisi target juga mengirimkan paket perintah untuk mengawali proses transmisi data pembacaan memori mikrokontroler target. Format *frame* paket perintah ditunjukkan dalam Tabel 4.7 Tabel 4.8, Tabel 4.9.

Tabel 4.7 Format *frame* paket perintah dari sisi pemrogram saat sesi tulis target

3 *byte*	1 *byte*	1 *byte*	2 *byte*	1 *byte*	3 *byte*	1 *byte*	2 *byte*
header	ID Perintah	0x7C	Jumlah paket	0x7C	Device signature	0x7C	*checksum*

Tabel 4.8 Format *frame* paket perintah dari sisi pemrogram saat sesi baca target

3 *byte*	1 *byte*	1 *byte*	2 *byte*	1 *byte*	3 *byte*	1 *byte*	2 *byte*
header	ID Perintah	0x7C	N/A	0x7C	Device signature	0x7C	*checksum*

Tabel 4.9 Format *frame* paket perintah dari sisi target saat sesi baca target

3 *byte*	1 *byte*	1 *byte*	2 *byte*	1 *byte*	1 *byte*	1 *byte*	2 *byte*
header	ID Perintah	0x7C	Jumlah paket	0x7C	ID Mikrokontroler	0x7C	*checksum*

c. Paket data.

Paket data terdiri atas dua jenis, yaitu paket data dari sisi pemrogram dan paket data dari sisi target. Paket data dari sisi pemrogram berisi data memori mikrokontroler yang dikirimkan secara langsung oleh sisi pemrogram, sehingga dalam pengirimannya akan mengirim secara keseluruhan *byte* data ke mikrokontroler target. Paket data dari sisi target berisi data memori mikrokontroler target yang dikirimkan secara langsung, sehingga dalam pengirimannya akan menyimpan secara keseluruhan *byte* data dari mikrokontroler target ke sisi pemrogram. Dalam paket data terdapat informasi ID perintah dan data memori mikrokontroler. Format *frame* paket data ditunjukkan dalam Tabel 4.10. Dan Tabel 4.11

Tabel 4.10 Format *frame* paket data sisi pemrogram

3 *byte*	1 *byte*	1 *byte*	n *byte*	2 *byte*
Header	ID Perintah	0x7C	DATA kiriman	*checksum*

Tabel 4.11 Format *frame* paket data sisi target

3 *byte*	n *byte*	2 *byte*
Header	DATA kiriman	*checksum*

d. Paket konfirmasi.

Paket konfirmasi merupakan paket yang berfungsi sebagai paket penanda berlangsungnya transmisi. Seluruh paket konfirmasi hanya terdiri dari data-data status saja tanpa tambahan *header* maupun *checksum*.

Paket konfirmasi ditransmisikan oleh sisi penerima data transmisi. Seluruh paket transmisi yang dikirimkan mulai dari paket perintah, paket data dan paket akhir diberikan balasan berupa sebuah paket konfirmasi. Paket konfirmasi terbagi menjadi dua jenis, yaitu paket ok dan paket gagal. Format *frame* paket konfirmasi ditunjukkan dalam Tabel 4.12.

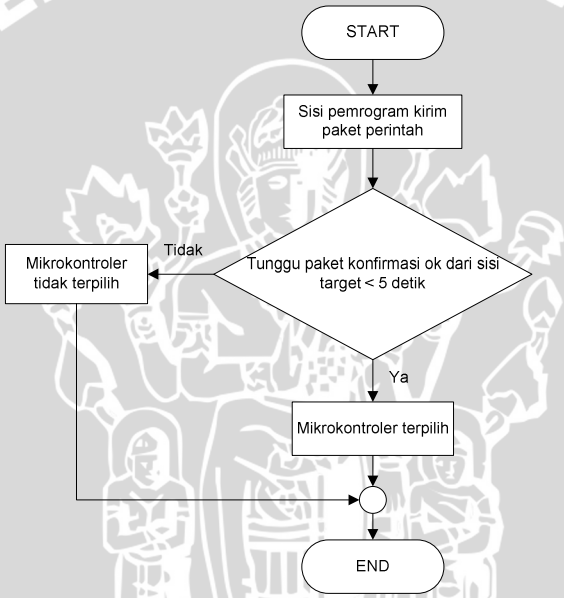
Tabel 4.12 Format *frame* paket konfirmasi

Paket OK	0x7C	0xAA	0x7C	0xAA
Paket Gagal	0x7C	0x55	0x7C	0x55

Untuk mempermudah analisis protokol komunikasi, maka perlu dibuat sebuah diagram skenario protokol komunikasi. Skenario protokol dibedakan berdasarkan sesi operasinya, yakni sesi pilih mikrokontroler target, sesi tulis target dan sesi baca target.

a. Skenario protokol untuk sesi pilih mikrokontroler target

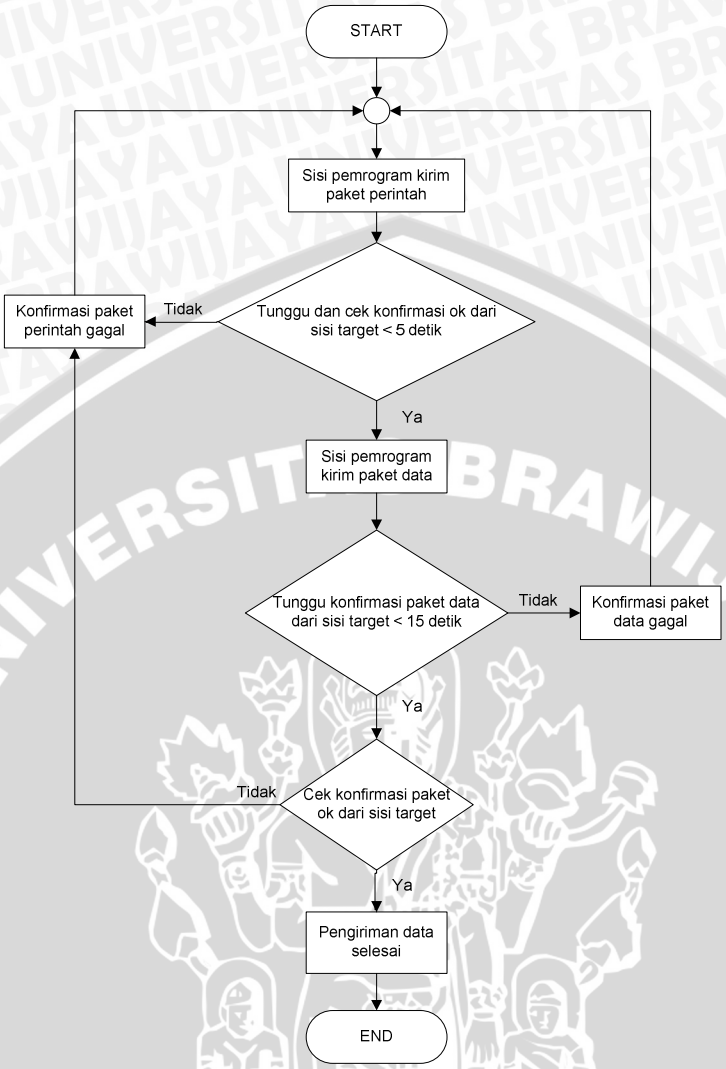
Sesi pilih mikrokontroler target diawali oleh pengiriman paket perintah pilih mikrokontroler oleh sisi pemrogram. Sisi target memberikan balasan berupa paket konfirmasi ok ketika paket perintah berhasil diterima dan dioalah maka mikrokontroler target telah terpilih. Skenario protokol untuk sesi pilih mikrokontroler target ditunjukkan dalam gambar 4.15.



Gambar 4.15 Diagram alir protokol untuk sesi pilih mikrokontroler target

b. Skenario protokol untuk sesi tulis memori target

Sesi tulis memori target diawali oleh pengiriman paket perintah tulis target oleh sisi pemrogram. Sisi target memberikan balasan berupa konfirmasi paket ok ketika paket perintah berhasil diterima dan diolah. Skenario protokol untuk sesi tulis memori target ditunjukkan dalam Gambar 4.16.



Gambar 4.16 Diagram alir protokol sesi tulis memori target

Pada proses selanjutnya, sisi pemrogram mengirimkan paket data sedangkan sisi target mengirimkan paket konfirmasi berdasarkan kondisi paket yang diterima yaitu berhasil atau gagal. Apabila paket data berhasil diterima dengan baik tanpa terdapat kesalahan data *checksum* maka sisi target akan mengirimkan konfirmasi paket ok ke sisi pemrogram. Sedangkan bila terjadi kesalahan dalam penerimaan maka sisi target akan mengirimkan konfirmasi paket gagal dan sisi pemrogram harus mengulang pengiriman paket data yang gagal tersebut. Proses di atas diulangi hingga seluruh data memori yang akan dituliskan ke mikrokontroler target telah diterima dengan baik oleh sisi target.

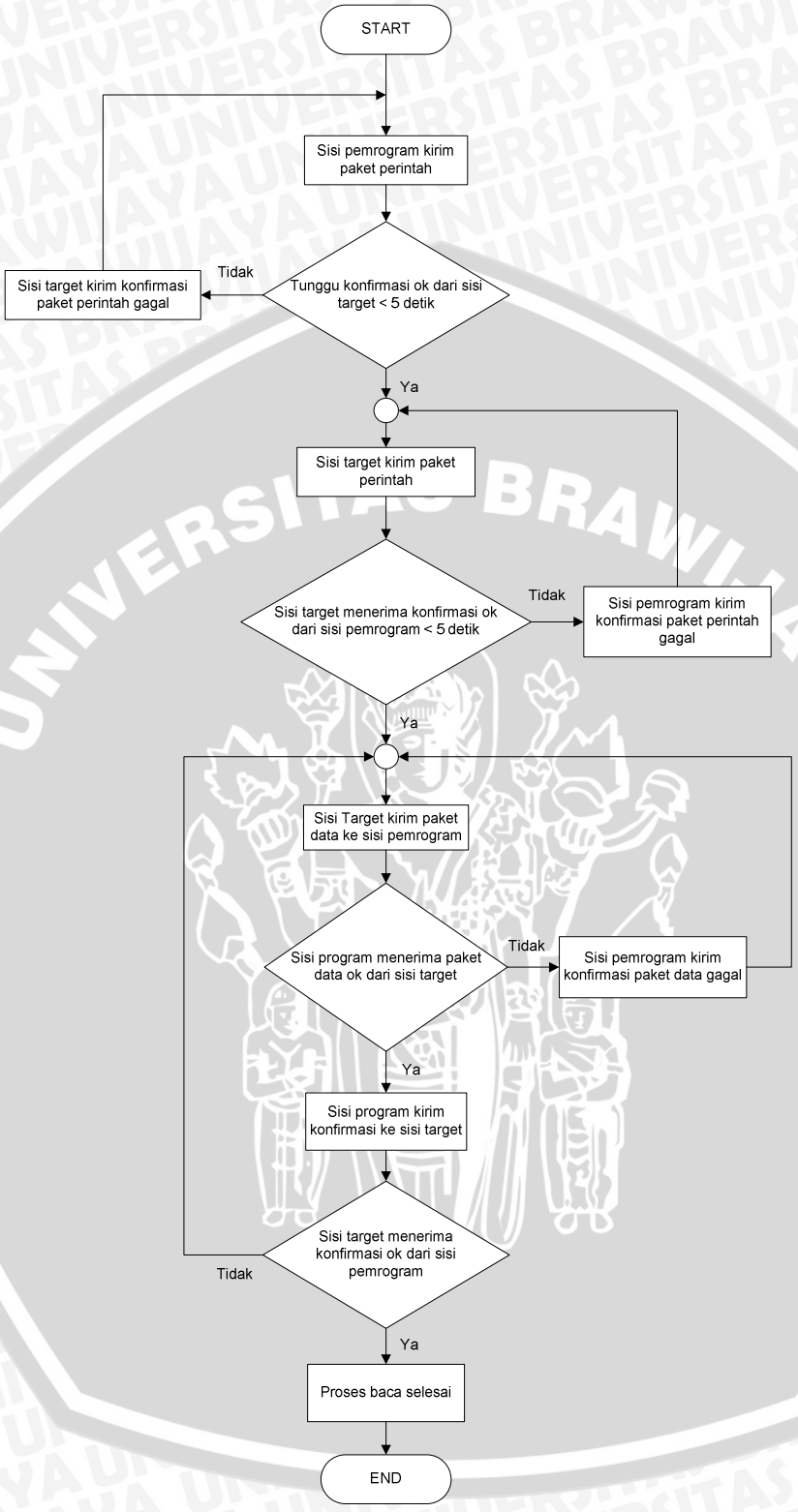
Apabila telah selesai maka sisi target mengirimkan balasan konfirmasi paket ok ke sisi pemrogram. Dengan diterimanya konfirmasi paket ok oleh sisi pemrogram maka sesi tulis memori target telah selesai.

c. Skenario protokol untuk sesi baca memori target

Sesi baca target diawali oleh pengiriman paket perintah baca target oleh sisi pemrogram. Sisi target memberikan balasan berupa konfirmasi paket ok ketika paket perintah berhasil diterima dan diolah. Setelah sisi pemrogram menerima balasan konfirmasi paket ok dari sisi target, sisi pemrogram akan menunggu pengiriman paket perintah dari sisi target. Gambar 4.17 menunjukkan skenario protokol untuk sesi baca memori target.

Sisi target melanjutkan proses berikutnya yaitu pengiriman paket perintah ke sisi pemrogram yang berisi jumlah paket yang akan dikirimkan. Sisi pemrogram akan memberikan balasan berupa konfirmasi paket ok. Sisi target akan melanjutkan dengan pengiriman paket data dan sisi pemrogram akan membalas dengan mengirimkan paket konfirmasi sesuai dengan kondisi paket yang diterimanya, berhasil ataukah gagal.

Apabila paket data berhasil diterima dengan baik tanpa terdapat kesalahan data *checksum* maka sisi pemrogram akan mengirimkan konfirmasi paket ok ke sisi target. Sedangkan bila terjadi kesalahan dalam penerimaan maka sisi pemrogram akan mengirimkan konfirmasi paket gagal dan sisi target harus mengulang pengiriman paket data kembali. Proses di atas diulang hingga seluruh data memori yang dibaca dari mikrokontroler target telah diterima dengan baik oleh sisi pemrogram. Sisi pemrogram akan menampilkan hasil pembacaan memori mikrokontroler target di perangkat lunak sisi pemrogram. Dengan diterimanya paket konfirmasi paket ok oleh sisi target maka sesi baca memori target telah selesai.



Gambar 4.17 Diagram alir protokol sesi baca memori target

Secara umum, beberapa data transmisi yang merupakan informasi tentang jenis instruksi (ID perintah) untuk operasi memori terdapat di dalam Tabel 4.13

sedangkan beberapa data informasi tentang ID pengirim data serta informasi tentang jumlah paket data transmisi maupun panjang satu paket data terdapat di dalam Tabel 4.14. Hasil yang diperoleh dari perancangan protokol komunikasi data menggunakan media *radio frequency* adalah sistem mampu untuk menstransmisikan data memori sampai dengan 32640 *byte* (32K *byte*). Nilai ini diperoleh dari hasil perkalian antara jumlah paket data maksimal dengan panjang satu paket data.

Tabel 4.13 ID perintah untuk operasi memori mikrokontroler target

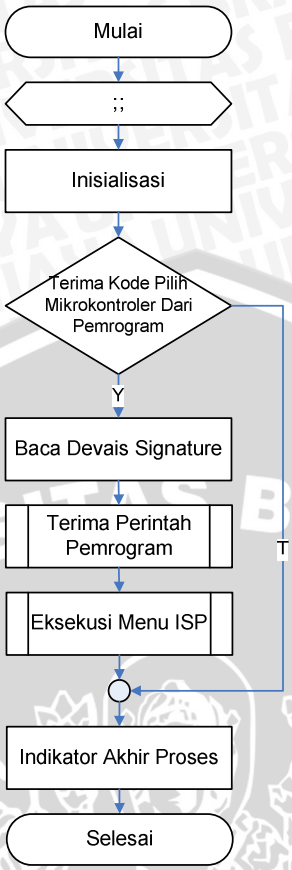
No	Jenis Instruksi	Kode Instruksi (desimal)	Kode Heksa
1	Tulis Memori Flash	16	0x10
2	Baca Memori Flash	32	0x20
3	Tulis Memori EEPROM	48	0x30
4	Baca Memori EEPROM	64	0x40
5	Tulis Fusebits	80	0x50
6	Baca Fusebits	96	0x60
7	Hapus Memori Chips	112	0x70

Tabel 4.14 Informasi ukuran data serta ID pengirim data transmisi

No	Informasi	Kode Instruksi (desimal)	Kode Heksa
1	ID sisi pemrogram	128	0x81
2	ID sisi target	129	0x82

4.4.3 Perancangan Perangkat Lunak Mikrokontroler Sisi Target

Perangkat lunak pada mikrokontroler secara umum menangani beberapa proses yaitu transmisi data serial melalui *radio frequency*, akses SRAM eksternal serta *serial downloading*. Diagram alir program utama pada sisi target ditunjukkan di dalam Gambar 4.18.



Gambar 4.18 Diagram alir program utama

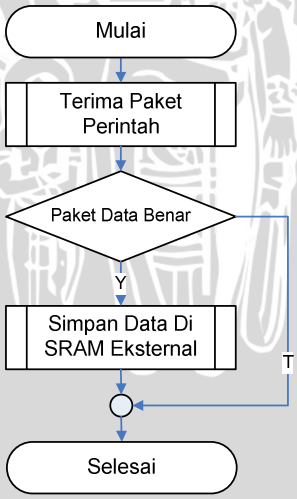
Pada proses insialisasi selain terdapat inisialisasi perangkat keras, inisialisasi variabel juga terdapat proses pembacaan eksistensi mikrokontroler target. Apabila mikrokontroler target tidak terpasang atau telah rusak maka proses selanjutnya tidak akan dilanjutkan. Pembacaan *device signature* dilakukan diawal setelah proses inisialisasi. Hasil pembacaan ini akan disimpan untuk kemudian dibandingkan dengan *device signature* yang dikirimkan oleh sisi pemrogram pada proses terima perintah dari sisi pemrogram.

Sementara itu, diagram alir transmisi data serial menggunakan *radio frequency* terdiri atas diagram alir proses terima perintah dari sisi pemrogram dan diagram alir proses pengiriman data ke sisi pemrogram. Proses terima perintah dari sisi pemrogram sesuai dengan protokol komunikasi data pada sesi tulis target. Sedangkan proses pengiriman data ke sisi pemrogram sesuai dengan protokol komunikasi data pada sesi baca target.

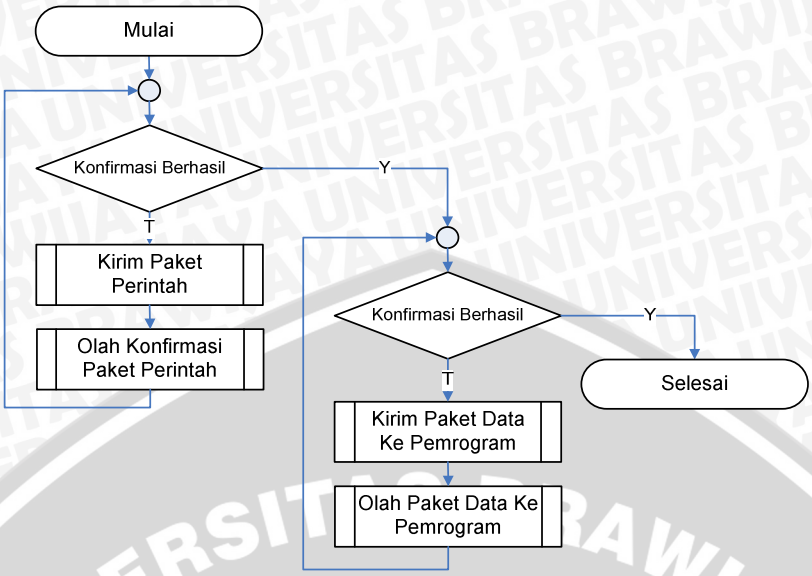
Diagram alir transmisi untuk proses terima perintah ditunjukkan dalam Gambar 4.19. Proses transmisi selalu diawali dengan pengiriman paket perintah

kemudian dilanjutkan dengan transmisi paket data. Perintah operasi memori dari sisi pemrogram disimpan dan digunakan pada proses eksekusi pemilihan operasi ISP yang terdiri atas baca tulis memori Flash, EEPROM, Fuse bit serta operasi hapus memori chip. Seluruh data yang akan ditulis ke memori mikrokontroler target disimpan terlebih dahulu ke dalam SRAM eksternal. Sebelum disimpan, setiap paket data akan diperiksa kebenarannya serta dideteksi ada tidaknya kesalahan data transmisi menggunakan algoritma pendeteksi kesalahan *checksum*. Setelah seluruh paket data terkirim, maka proses transmisi diakhiri dengan pengiriman paket akhir.

Diagram alir transmisi untuk proses pengiriman data ke sisi pemrogram ditunjukkan dalam Gambar 4.20. Proses transmisi diawali dengan pengiriman paket perintah oleh sisi target sesuai dengan instruksi sisi pemrogram kemudian dilanjutkan dengan pengiriman paket data. Paket data dikirimkan bersama dengan ID Paket data tersebut. Setelah sisi target memperoleh konfirmasi berhasil dari sisi pemrogram maka dikirimkan paket data selanjutnya. Proses transmisi diakhiri dengan pengiriman paket akhir oleh sisi target setelah seluruh paket data berhasil ditransmisikan oleh sisi target dan diterima oleh sisi pemrogram.

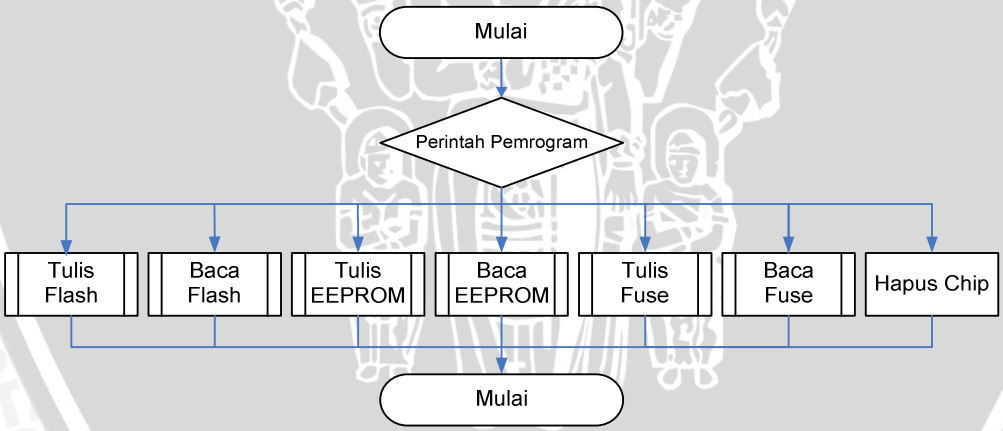


Gambar 4.19 Diagram alir transmisi untuk proses terima perintah sisi pemrogram



Gambar 4.20 Diagram alir transmisi untuk proses pengiriman data ke sisi pemrogram

Diagram alir pemilihan jenis operasi memori untuk dieksekusi oleh sisi target berdasarkan instruksi yang diterima dari sisi pemrogram ditunjukkan di dalam Gambar 4.21. Eksekusi operasi memori dilakukan pada saat sisi target menerima akhir paket sebagai penanda eksekusi instruksi dari sisi pemrogram.



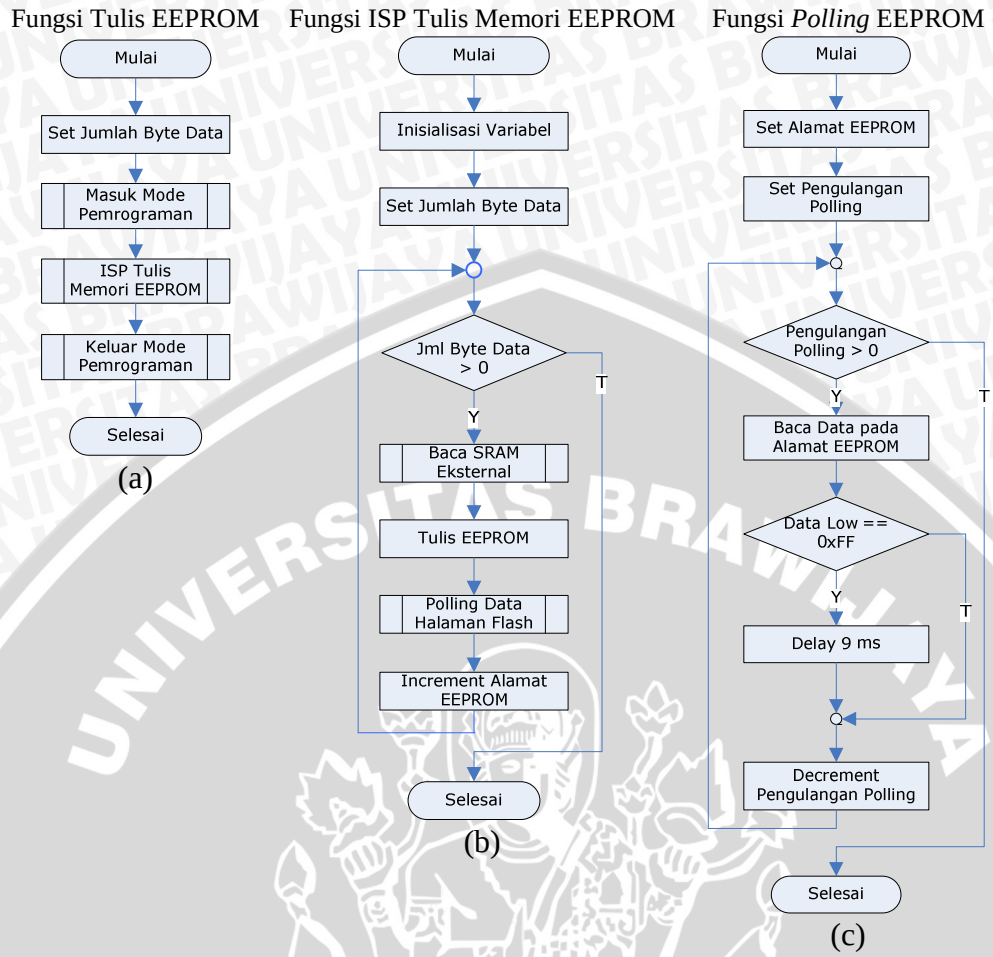
Gambar 4.21 Diagram alir pemilihan operasi memori

Ketika perintah pemrogram dieksekusi, maka proses selanjutnya adalah proses *serial downloading* mikrokontroler target. Pada operasi tulis memori, terdapat tiga tahap yaitu eksekusi terhadap fungsi tulis memori, fungsi ISP tulis memori serta fungsi *polling* memori pada operasi tulis Flash dan EEPROM. Fungsi tulis memori terdapat inisialisasi jumlah data yang akan ditulis sedangkan fungsi ISP tulis memori terdapat proses pembacaan data memori pada SRAM eksternal yang akan ditulis pada alamat memori mikrokontroler target. Sedangkan

fungsi *polling* berfungsi untuk melakukan *polling* pembacaan terhadap alamat memori target penulisan. Mekanisme ini dilakukan atas rekomendasi *datasheet* untuk mengurangi waktu tunda yang harus ditunggu pada penulisan data memori.

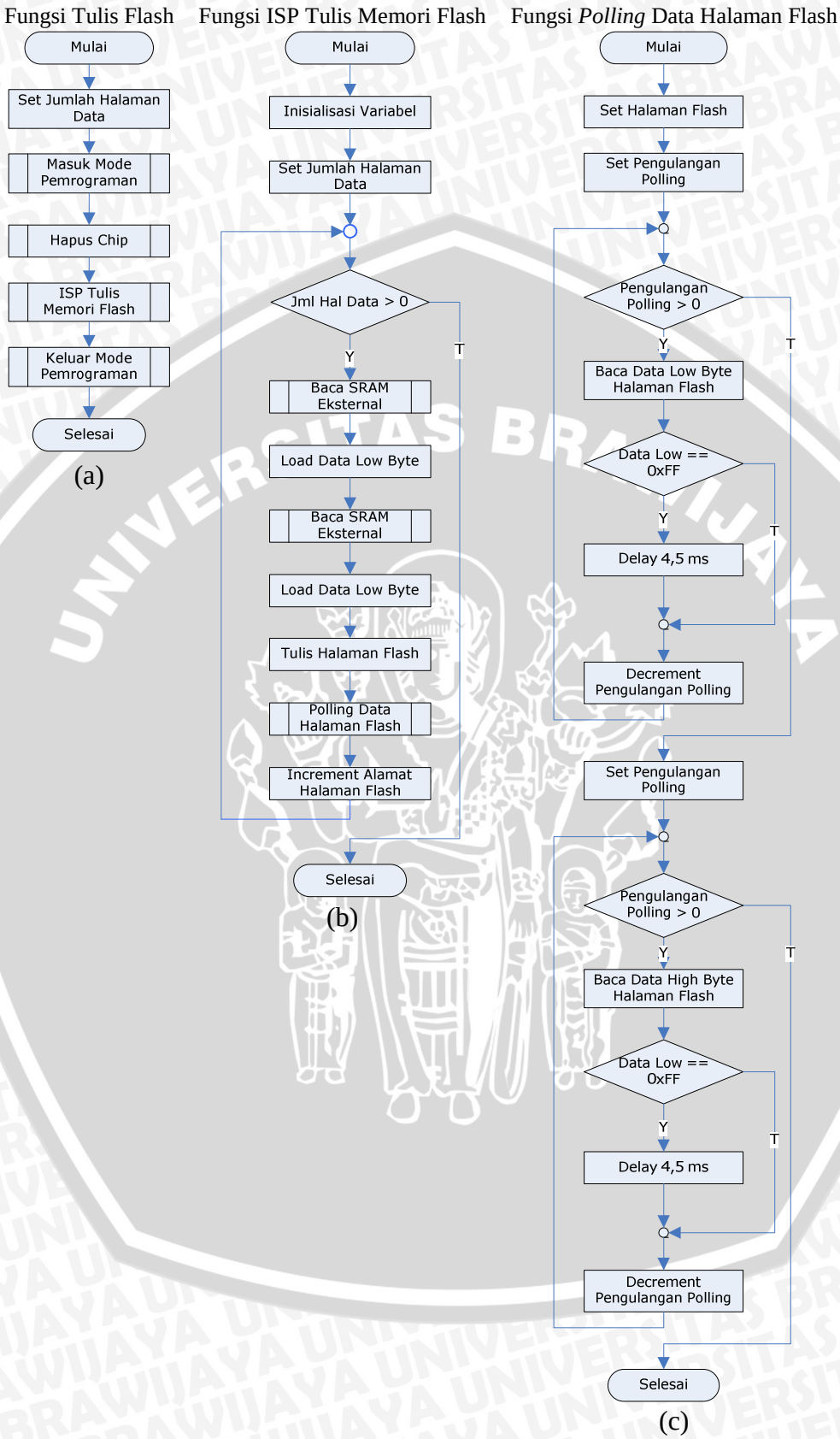
Sedangkan pada operasi baca memori, terdapat dua tahap yaitu eksekusi terhadap fungsi baca memori dan fungsi ISP baca memori target. Pada fungsi baca memori terdapat fungsi pembacaan ukuran memori Flash untuk menentukan jumlah paket data yang akan dikirimkan ke sisi pemrogram.

Algoritma penulisan memori EEPROM tunjukkan dalam Gambar 4.22 (a). Inisialisasi jumlah *byte* data yang akan ditulis pada memori EEPROM dilakukan pada awal proses. Hal ini dilakukan dengan mengambil nilai *counter* data yang tersimpan di dalam SRAM. Nilai ini digunakan sebagai acuan *counter* pada proses *auto increment* alamat memori EEPROM yang akan ditulis dengan data memori. Diagram alir fungsi ISP tulis memori EEPROM yang terdapat dalam Gambar 4.22 (b) merupakan fungsi yang menggunakan protokol *serial downloading* standart AVR untuk operasi tulis memori EEPROM. Data memori yang akan ditulis pada setiap alamat memori EEPROM berasal dari hasil pembacaan SRAM eksternal. Proses pembacaan SRAM eksternal dilanjutkan dengan proses tulis memori EEPROM yang dilakukan pada setiap alamat EEPROM secara berurutan. Fungsi *Polling* EEPROM yang ditunjukkan dalam Gambar 4.22 (c) dirancang untuk mengurangi waktu tunda yang harus ditunggu pada penulisan memori EEPROM.

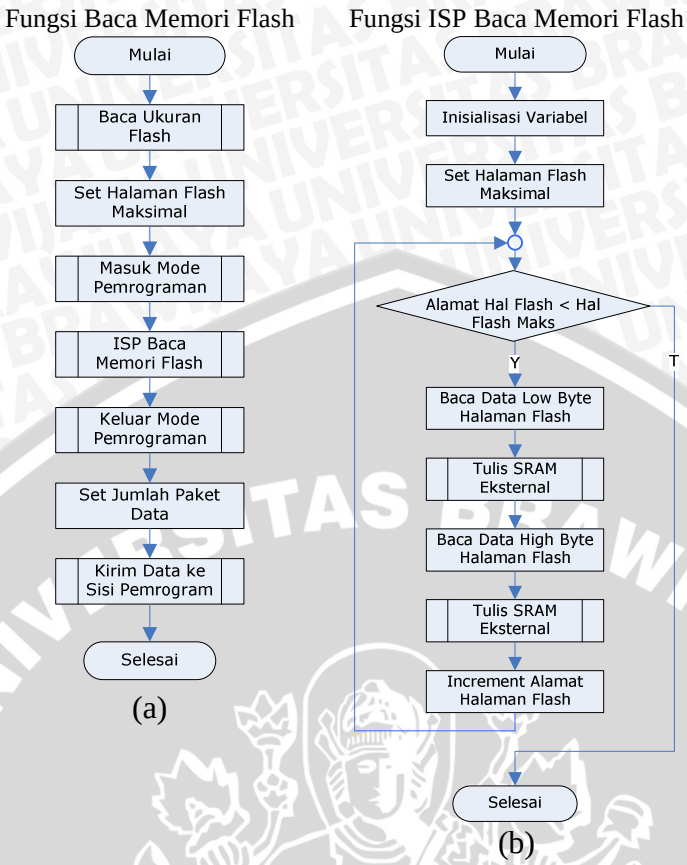


Gambar 4.22 Diagram alir operasi tulis memori EEPROM

Diagram alir penulisan memori Flash tunjukkan dalam Gambar 4.23 (a). Secara umum fungsi ini memiliki prinsip sama dengan prinsip pada fungsi tulis EEPROM. Memori Flash harus dihapus terlebih dahulu untuk menjamin bahwa data yang tertulis pada memori Flash adalah data yang baru. Diagram alir fungsi ISP tulis memori Flash yang terdapat dalam Gambar 4.23 (b) merupakan fungsi yang menggunakan protokol *serial downloading* standart AVR untuk operasi tulis memori Flash. Metode penulisan pada memori Flash berbeda dengan EEPROM. Metode yang digunakan adalah *writing page mode*, sehingga SRAM dibaca dua kali kemudian ditulis pada halaman memori Flash yang dimaksud. Fungsi *Polling* data halaman Flash yang ditunjukkan dalam Gambar 4.23 (c) dirancang untuk mengurangi waktu tunda yang harus ditunggu pada penulisan memori Flash.



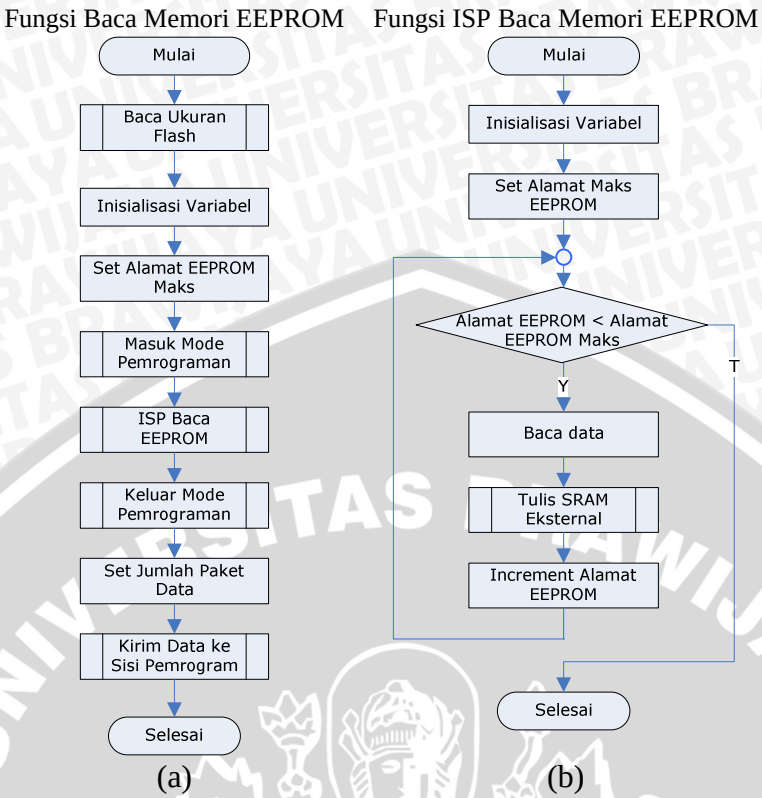
Gambar 4.23 Diagram alir operasi tulis memori Flash



Gambar 4.24 Diagram alir operasi baca memori Flash

Diagram alir pembacaan memori Flash tunjukkan dalam Gambar 4.24 (a). Hal yang pertama kali dilakukan pada proses pembacaan memori Flash adalah pembacaan ukuran memori Flash. Informasi tentang ukuran memori Flash dapat diperoleh dari *device signature*. Informasi ini digunakan untuk menentukan jumlah halaman memori Flash yang akan dibaca. Nilai yang diset pada jumlah halaman memori Flash digunakan sebagai acuan *counter* pada proses *auto increment* alamat memori Flash yang akan dibaca. Setelah proses pembacaan memori Flash berhasil, maka proses selanjutnya adalah menentukan jumlah paket data yang akan dikirimkan ke sisi pemrogram.

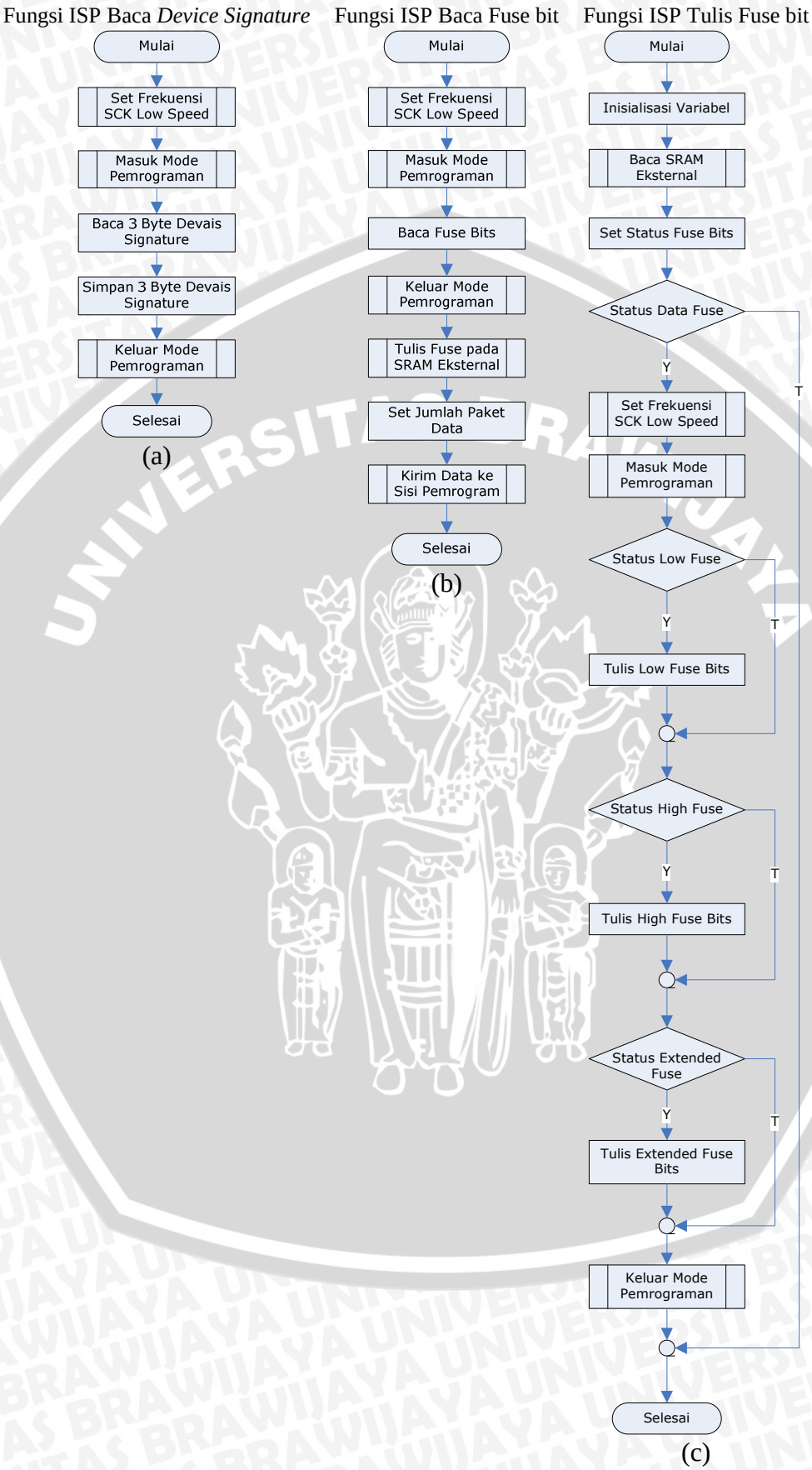
Diagram alir fungsi ISP baca memori Flash yang terdapat dalam Gambar 4.24 (b) merupakan fungsi yang menggunakan protokol *serial downloading* standart AVR untuk operasi baca memori Flash. Data memori yang telah terbaca pada setiap alamat halaman memori Flash disimpan di dalam SRAM eksternal. Data inilah yang nantinya akan dikirimkan ke sisi pemrogram apabila ada intruksi operasi baca memori Flash.



Gambar 4.25 Diagram alir operasi baca memori EEPROM

Secara umum prinsip pembacaan memori EEPROM sama dengan prinsip pembacaan memori Flash. Gambar 4.25 (a) menunjukkan diagram alir fungsi baca memori EEPROM. Alamat EEPROM maksimal diset dengan ukuran memori EEPROM yang ukurannya 1/16 dari ukuran memori Flash. Nilai yang diset pada alamat EEPROM maksimal digunakan sebagai acuan *counter* pada proses *auto increment* alamat memori EEPROM yang akan dibaca. Setelah proses pembacaan memori EEPROM berhasil, maka proses selanjutnya adalah menentukan jumlah paket data yang akan dikirimkan ke sisi pemrogram.

Diagram alir fungsi ISP baca memori EEPROM yang terdapat dalam Gambar 4.25 (b) merupakan fungsi yang menggunakan protokol *serial downloading* standart AVR untuk operasi baca memori EEPROM. Data memori yang telah terbaca pada setiap alamat memori EEPROM disimpan di dalam SRAM eksternal. Metode pembacaan memori EEPROM sedikit berbeda dengan memori Flash, yaitu pembacaan setiap alamat memori EEPROM membutuhkan 1 *byte* SRAM untuk menyimpan data. Data inilah yang nantinya akan dikirimkan ke sisi pemrogram apabila ada intruksi operasi baca memori EEPROM.



Gambar 4.26 Diagram alir baca *device signature* serta operasi baca tulis Fuse bit

Pembacaan *device signature* merupakan salah satu proses yang penting untuk mengetahui kode devais serta ukuran memori maksimal mikrokontroler target. Diagram alir fungsi ISP baca d*evice signature* yang terdapat dalam Gambar 4.26 (a) merupakan fungsi yang menggunakan protokol *serial downloading* standart AVR untuk operasi baca *device signature*. Data informasi tentang *device signature* terdiri atas 3 *byte* data. Alamat memori yang menyediakan data tentang *device signature* hanya dapat dibaca tanpa bisa ditulis.

Konfigurasi Fuse bit maksimal terdiri tiga buah Fuse yaitu *low fuse, high fuse* serta *extended fuse* tergantung pada jenis dan kompatibelitas devais yang dibaca. Data informasi Fuse bit terdapat pada tiga buah alamat yang berbeda sehingga pembacaannya harus dilakukan satu-persatu. Diagram alir fungsi ISP baca Fuse bit terdapat dalam Gambar 4.26 (b). Hasil pembacaannya disimpan di dalam SRAM kemudian dikirimkan ke sisi pemrogram dalam sebuah paket data.

Penulisan konfigurasi Fuse bit terdiri atas penulisan *low fuse, high fuse* serta *extended fuse*. Penulisan Fuse bit dilakukan berdasarkan instruksi sisi pemrogram yang mengirimkan status penulisan Fuse bit serta data Fuse bit yang akan ditulis. Penulisan Fuse bit rentan terhadap kesalahan sehingga penulisannya dilakukan pada SCK berfrekuensi rendah. Diagram alir fungsi ISP tulis Fuse bit terdapat dalam Gambar 4.28 (c).

BAB V
PENGUJIAN DAN ANALISIS

Tujuan pengujian dan analisis adalah untuk mengetahui tingkat kesesuaian hasil perancangan dan pembuatan alat dengan hasil yang diharapkan. Pengujian dan analisis Sistem *Downloader* Data Memori Mikrokontroler Atmel AVR Melalui *radio frequency* Multi Target Sisi Target terdiri atas dua bagian yaitu pengujian dan analisis perangkat keras serta pengujian dan analisis perangkat lunak. Pengujian dan analisis dilakukan dengan melakukan pengukuran pada tiap-tiap blok untuk memudahkan analisis kemudian dilanjutkan dengan pengujian alat secara keseluruhan dengan bantuan instrumen dan *software* elektronika serta peralatan sebagai berikut:

1. Minimum sistem AVR ATmega8535.
2. *Software CodeVision AVR* C Compiler.
3. *Software* MyWrite.
4. *Software* YS-1020 *transciever*.
5. *Serial RS232 to USB Converter*.
6. Rangkaian MAX 232.
7. Rangkaian Pemancar dan Penerima RF YS-1020.
8. PC (*personal computer*).

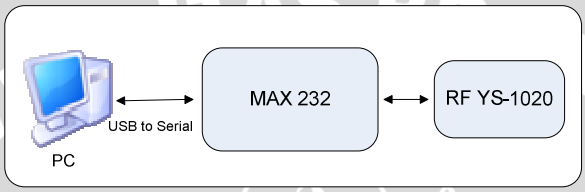
Pengujian dan analisis dilakukan pada tiap-tiap blok maupun alat secara keseluruhan yang meliputi:

1. Pengujian perangkat keras r*adio frequency*.
2. Pengujian antarmuka mikrokontroler ATmega162 dengan SRAM eksternal.
3. Pengujian perangkat lunak ISP (*In-System Programming*).
4. Pengujian sistem secara keseluruhan yang terdiri atas:
   a. Pengujian pengiriman instruksi.
   b. Pengujian pengiriman data.
   c. Pengujian terhadap jarak.
   d. Pengujian kecepatan proses *serial downloading*.

5.1 Pengujian Perangkat Keras *Radio Frequency*

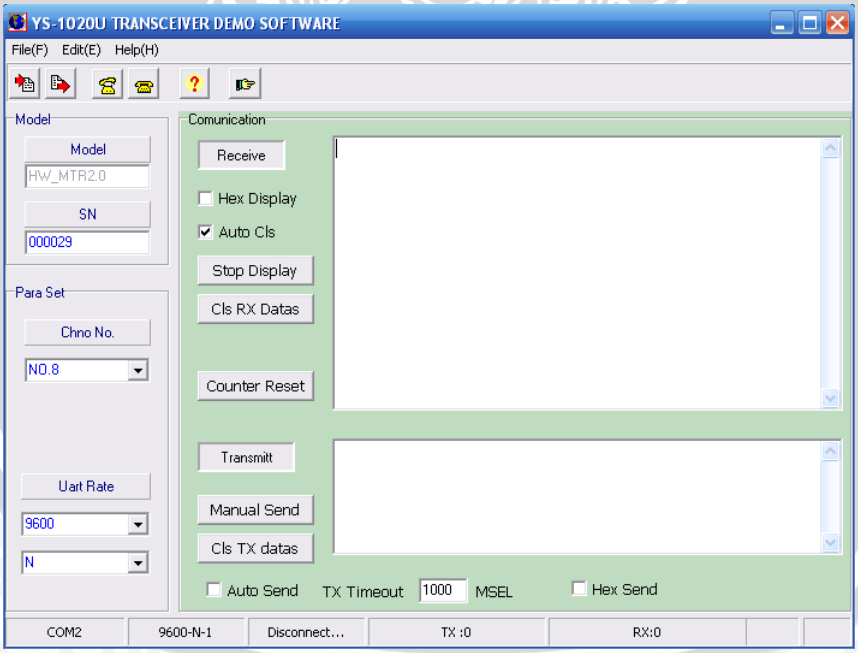
Pengujian ini bertujuan untuk mengetahui apakah rangkaian pemancar dan penerima RF dapat menerima dan mengirimkan data dengan baik. Dan bagaimana cara pengaturan *baudrate* dan *channel* modul YS-1020.

Peralatan yang digunakan dalam pengujian adalah rangkaian pemancar dan penerima YS-1020, *software* YS-1020 *transciever* yang bekerja pada frekwensi 433 MHz, PC dan rangkaian max 232. Prosedur pengujian pengaturan *baudrate* dan *channel* adalah dengan menyusun rangkaian seperti dalam Gambar 5.1 berikut ini :



Gambar 5.1 Diagram blok pengaturan *baudrate* dan *channel* pemancar dan penerima RF

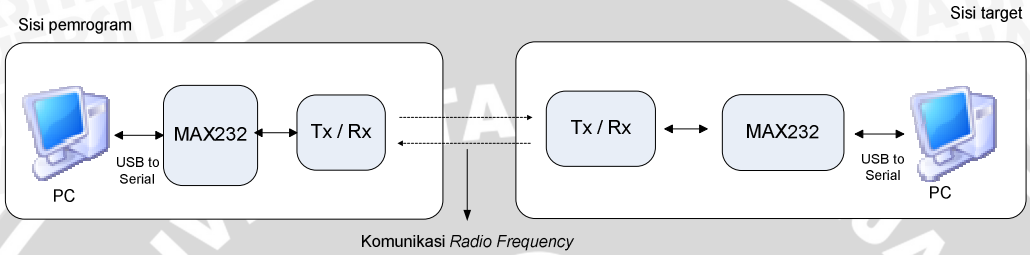
Untuk tampilan program pengaturan *baudrate* dan *channel* ditunjukkan dalam gambar 5.2 berikut ini :



Gambar 5.2 Tampilan program pengaturan *baudrate* dan *channel* pemancar dan penerima RF

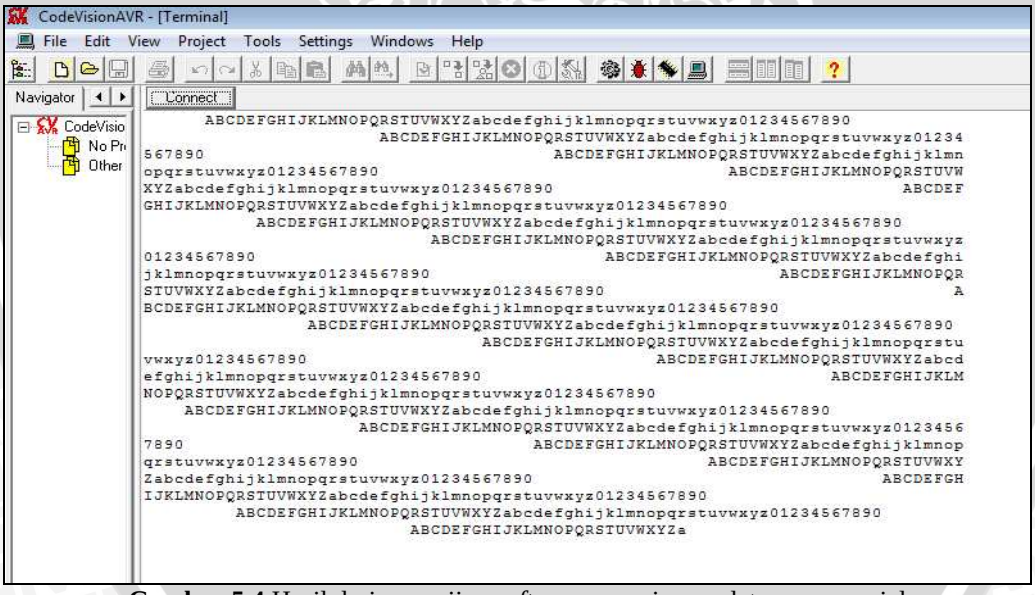
Dari tampilan program didapatkan keterangan atau setting RF yang digunakan yaitu: model hardware HW_MTR2.0, nomor serial hardware 000029, setting baudrate 9600 bps, data format serial 8N1 (8 bit data 1 bit stop), menggunakan chanel 8. Pengujian berikutnya dilakukan dengan mengirimkan data

secara serial menggunakan *hyperterminal* program *CodeVision AVR* pada PC sisi pemrogram dan menerima data-data tersebut menggunakan *hyperterminal* program *CodeVision AVR* pada PC sisi target. dalam proses pengiriman data berupa string yang terdiri atas karakter-karakter huruf dan angka. Kemudian *hyperterminal* sisi target akan menampilkan data yang dikirim oleh *hyperterminal* sisi pemrogram. Diagram blok pengujian penerimaan data secara serial ini ditunjukkan dalam Gambar 5.3.



Gambar 5.3 Diagram blok pengujian *software* penerimaan data secara serial

Hasil dari pengujian perangkat lunak penerimaan data secara serial ditujukkan dalam Gambar 5.4.



Gambar 5.4 Hasil dari pengujian *software* penerimaan data secara serial

Pada tampilan program *Hyperterminal CodeVision AVR* tampak bahwa pengujian komunikasi data dengan mengirimkan karakter dapat berhasil dengan baik dan tidak ada data yang hilang. Karakter yang dikirim oleh rangkaian pemancar RF dapat diterima dengan baik oleh rangkaian penerima RF.

Pengujian selanjutnya yaitu digunakan konfigurasi *baudrate* dan *channel* yang berbeda-beda kemudian dilakukan proses pengiriman data secara serial..

Hasil pengujian konfigurasi pengaturan *baudrate* dan *cannel* rangkaian pemancar dan penerima modul r*adio frequency* ini ditunjukkan dalam Tabel 5.1 berikut.

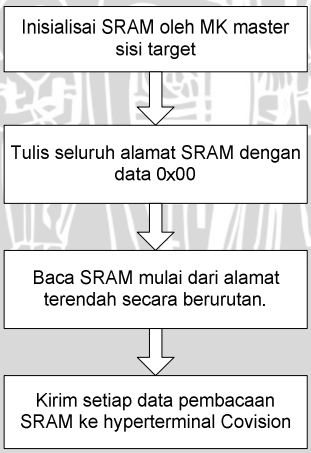
Tabel 5.1 Data hasil pengujian rangkaian RF YS-1020

Kondisi	Hasil pengujian
Baudrate dan *channel* sama	Pengiriman sukses
Baudrate sama dan *channel* beda	Pengiriman gagal
Baudrate beda dan *channel* sama	Pengiriman gagal
Baudrate beda dan *channel* beda	Pengiriman gagal

Berdasarkan hasil pengujian dalam Tabel 5.1 dapat dianalisis pengaturan bahwa *baudrate* dan *channel* dari kedua modul RF YS-1020 sisi pemrogram dan sisi target harus sama agar proses pengiriman data secara serial dapat berhasil

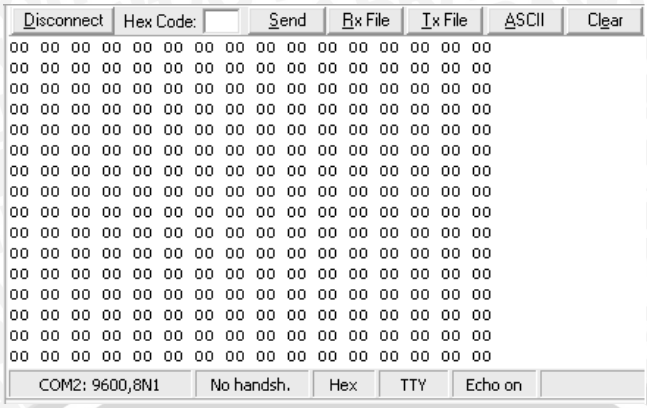
5.2 Pengujian Antarmuka Mikrokontroler ATmega162 Dengan SRAM Eksternal

Pengujian antarmuka mikrokontroler dengan SRAM eksternal bertujuan untuk mengetahui tingkat keberhasilan sistem untuk menyimpan data berukuran besar pada SRAM eksternal. Prosedur pengujian antarmuka mikrokontroler dengan SRAM eksternal diilustrasikan dalam Gambar 5.5. Antarmuka mikrokontroler dengan SRAM eksternal dikatakan berhasil apabila 32K *byte* memori SRAM eksternal seluruhnya berisi data 0x00.



Gambar 5.5 Prosedur pengujian antarmuka SRAM eksternal

Pembacaan memori SRAM eksternal pada mikrokontroler target diperoleh hasil 32K *byte* seluruhnya berisi data 0x00 diilustrasikan dalam Gambar 5.6, sehingga perancangan antarmuka mikrokontroler dengan SRAM berhasil dengan baik.



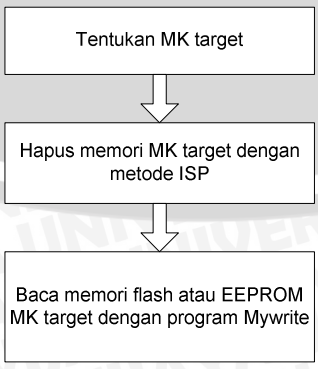
Gambar 5.6 Tampilan data SRAM eksternal

5.3 Pengujian Perangkat Lunak ISP (In-System Programming)

Pengujian ini bertujuan untuk mengetahui tingkat keberhasilan implementasi metode ISP (*In-System Programming*) dengan antarmuka SPI (*Serial Peripheral Interface*) untuk operasi pemrograman memori pada mikrokontroler target. Jenis operasi memori yang diuji meliputi operasi hapus *chip*, operasi baca *device signature*, operasi baca tulis memori Flash, operasi baca tulis EEPROM serta operasi baca tulis fuse bit. Penjelasan pada masing-masing pengujian operasi memori adalah sebagai berikut:

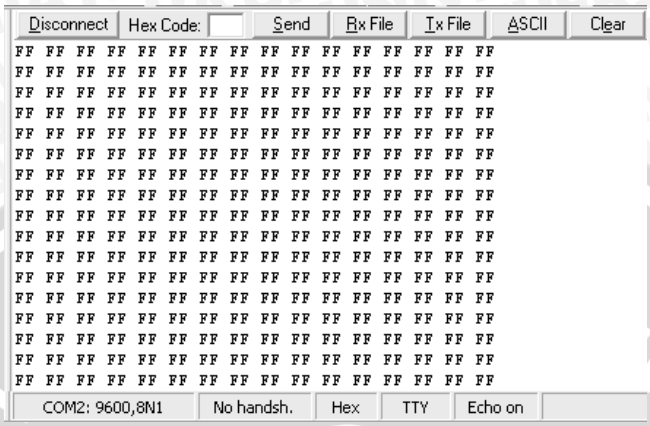
a. Pengujian operasi hapus memori *chip*.

Pengujian operasi hapus memori *chip* hanya dapat dilakukan pada mikrokontroler yang telah berisi data. Memori *chip* mikrokontroler yang dapat diprogram adalah memori Flash dan EEPROM. Operasi hapus memori *chip* dikatakan berhasil apabila hasil pembacaan terhadap memori Flash maupun EEPROM berisi nilai *default* yaitu 0xFF. Prosedur pengujian operasi hapus memori *chip* diilustrasikan dalam Gambar 5.7.



Gambar 5.7 Prosedur pengujian operasi hapus memori *chip*

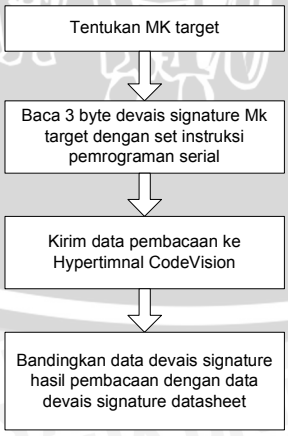
Hasil pembacaan terhadap isi memori Flash dan atau EEPROM memperlihatkan bahwa baik memori Flash maupun EEPROM seluruhnya berisi data 0xFF diilustrasikan dalam Gambar 5.8, sehingga dapat disimpulkan bahwa operasi hapus *chip* berhasil dengan baik.



Gambar 5.8 Tampilan data flash setelah proses hapus chip

b. Pengujian operasi baca *device signature*.

Pengujian operasi baca *device signature* (kode devais) dilakukan pada beberapa devais AVR yang berbeda yaitu pada mikrokontroler ATmega8, ATmega8535 dan ATmega32. Pengujian pembacaan *device signature* pada mikrokontroler target dikatakan berhasil apabila kode yang terbaca sama dengan *device signature* yang terdapat pada *datasheet* mikrokontroler target. Prosedur pengujian operasi baca *device signature* mikrokontroler target diilustrasikan dalam Gambar 5.9. Hasil pembacaan pada masing masing devais AVR terdapat dalam Tabel 5.2.



Gambar 5.9 Prosedur pengujian operasi baca *device signature*

Tabel 5.2 Data hasil pengujian operasi baca *device signature*

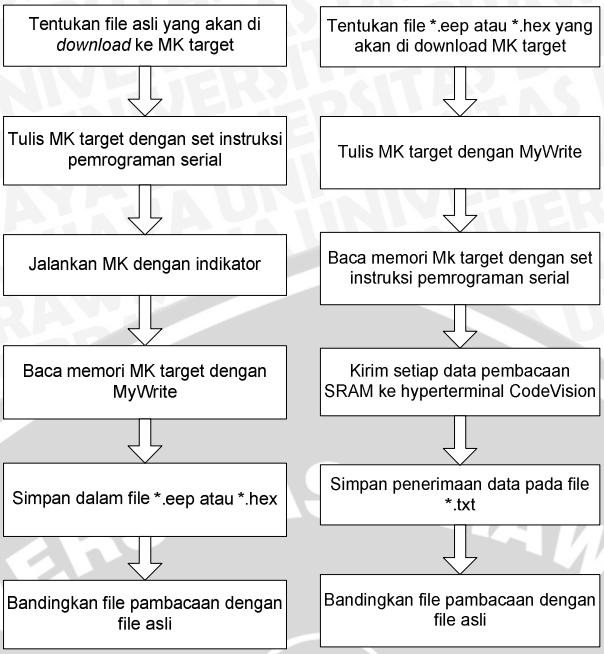
No	Nama devais	*Devais signature* hasil pembacaan	*Devais signature* pada *datasheet*
1	ATmega8	0x1E 0x93 0x07	0x1E 0x93 0x07
2	ATmega8535	0x1E 0x93 0x08	0x1E 0x93 0x08
3	ATmega32	0x1E 0x95 0x01	0x1E 0x95 0x01

Kesimpulan yang dapat diambil adalah operasi pembacaan *device signature* dikatakan berhasil karena Tabel 5.2 menunjukkan kesamaan antara hasil pembacaan *device signature* pada mikrokontroler target dengan *device signature* mikro-kontroler target yang tercantum pada *datasheet*.

c. Pengujian operasi tulis baca memori Flash atau EEPPOM.

Pengujian operasi tulis dan baca memori Flash atau EEPPOM memiliki prosedur yang sama. Pada operasi tulis memori, seluruh data memori yang tersimpan pada *chip* mikrokontroler target harus dihapus dengan operasi hapus memori *chip*. Sedangkan pada operasi baca, *chip* memori mikrokontroler target harus ditulis terlebih dahulu. Prosedur pengujian operasi tulis memori Flash maupun EEPROM diilustrasikan dalam Gambar 5.10 (a), sedangkan prosedur pengujian operasi baca memori Flash maupun EEPROM diilustrasikan dalam Gambar 5.10 (b).

Operasi tulis memori Flash maupun EEPROM dikatakan berhasil apabila *file* hasil pembacaan memori mikrokontroler target menggunakan *software* Mywrite sama dengan *file* asli yang ditulis pada memori mikrokontroler target. *File* yang ditulis harus melalui proses pemisahan data kode program dengan *header* dan *checksum*, kemudian disimpan pada *array* data mikrokontroler sisi target. Operasi baca memori mikrokontroler target dikatakan berhasil apabila hasil pembacaan sama dengan hasil pembacaan dengan menggunakan *software* Mywrite.



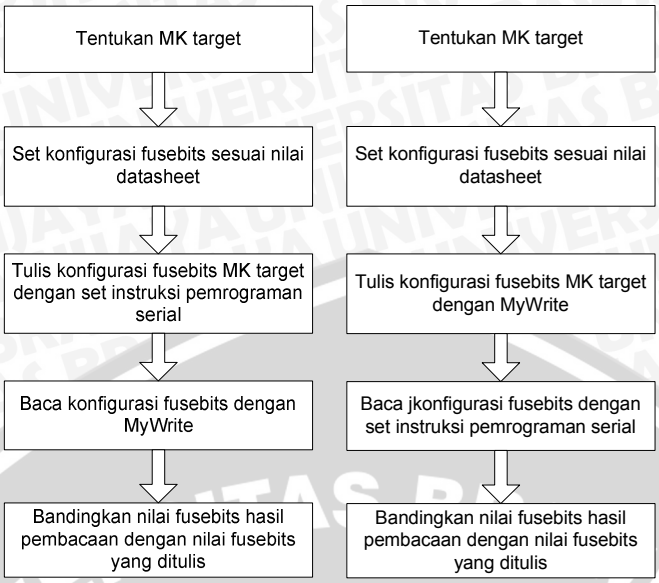
Gambar 5.10 Prosedur pengujian operasi tulis dan baca memori mikrokontroler target

Hasil pengujian terhadap operasi tulis maupun baca memori Flash dan atau EEPROM telah sesuai dengan *file* asli berektensi .hex atau .eep yang ditulis pada mikrokontroler target sehingga dapat disimpulkan bahwa operasi tulis maupun baca memori Flash dan atau EEPROM berhasil dengan baik.

d. Pengujian operasi tulis baca fuse bit.

Pengujian operasi pada fuse bit dilakukan dengan menggunakan bantuan Mywrite sebagai media pembanding. Hasil operasi tulis dan baca fuse bit oleh perangkat lunak pada sisi target dikatakan berhasil karena hasilnya sama dengan operasi baca tulis fuse bit menggunakan Mywrite.

Prosedur pengujian operasi tulis maupun baca data konfigurasi fuse bit secara berurutan diilustrasikan dalam Gambar 5.11 (a) dan Gambar 5.11 (b). Data hasil pengujian menunjukkan kesamaan antara hasil operasi fuse bit hasil perancangan dengan hasil operasi fuse bit menggunakan Mywrite.



Gambar 5.11 Prosedur pengujian operasi tulis dan baca fuse bit mikrokontroler target

Seluruh pengujian pada proses *serial downloading* berhasil dengan baik sehingga dapat disimpulkan bahwa perangkat lunak hasil perancangan yang bertugas untuk menangani proses ini sudah sesuai dengan yang diharapkan.

5.4 Pengujian Sistem Secara Keseluruhan

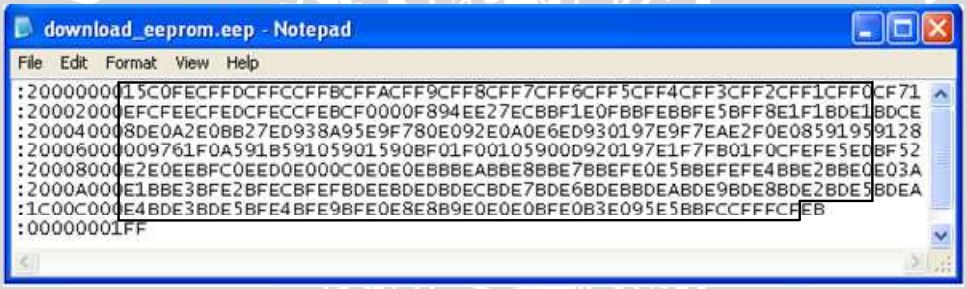
5.4.1 Pengujian Penampungan Data pada SRAM Eksternal

Pengujian penampungan data memori pada SRAM eksternal terdiri atas dua bagian yaitu pengujian penampungan data memori yang dikirim dari sisi pemrogram serta pengujian penampungan data memori dari hasil pembacaan mikrokontroler target. Pengujian yang pertama bertujuan untuk mengetahui kesesuaian antara data memori yang tersimpan pada SRAM eksternal dengan data memori yang dikirimkan oleh sisi pemrogram. Sedangkan pengujian yang kedua bertujuan untuk mengetahui kesesuaian antara data memori yang tersimpan di dalam SRAM dengan data hasil pembacaan mikrokontroler target setelah melalui operasi tulis memori.

Pengujian penampungan data memori yang dikirim dari sisi pemrogram menggunakan instrumen *serial RS232 to USB converter*. Operasi memori yang digunakan pada pengujian ini adalah operasi tulis memori EEPROM. Mikrokontroler target yang digunakan adalah ATmega8535 yang memiliki EEPROM berukuran 512 *byte*. Pengujian ini menggunakan *file* memori EEPROM

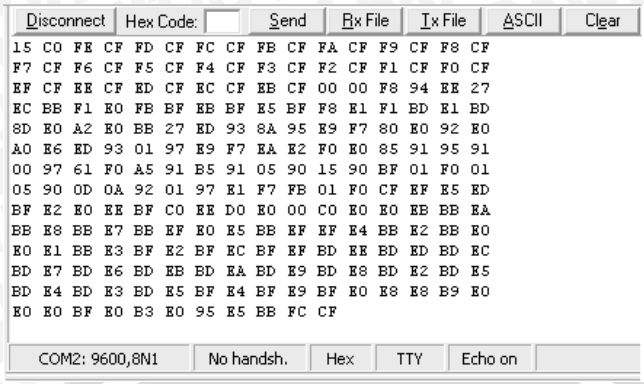
berekstensi eep berukuran 220 *byte*. Prosedur pengujian ini diilustrasikan dalam Gambar 5.12 sedangkan *file* berisi data memori EEPROM yang akan ditulis ditunjukkan dalam Gambar 5.13.

Gambar 5.12 Prosedur pengujian penampungan data dari sisi pemrogram pada SRAM



Gambar 5.13 *File* data memori EEPROM yang akan ditulis pada mikrokontroler target

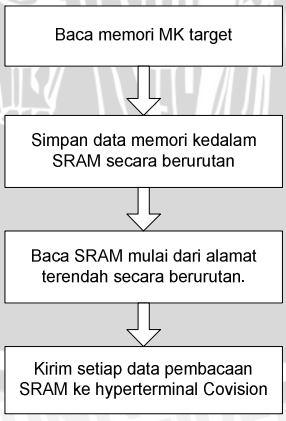
Data memori EEPROM yang dikirimkan oleh sisi pemrogram telah mengalami proses pemisahan antara *header*, data memori dan *checksum*. Pada bagian awal terdapat kode heksa yang merupakan *header* yang terdiri atas *start byte*, *byte count*, alamat serta *record type*, dan pada bagian akhir terdapat *checksum*. Data memori adalah kode heksa yang berada di dalam kotak hitam yang ditunjukkan dalam Gambar 5.13. Data inilah yang dikirimkan secara serial oleh sisi pemrogram mulai dari alamat terendah sampai dengan akhir kode heksa tersebut.



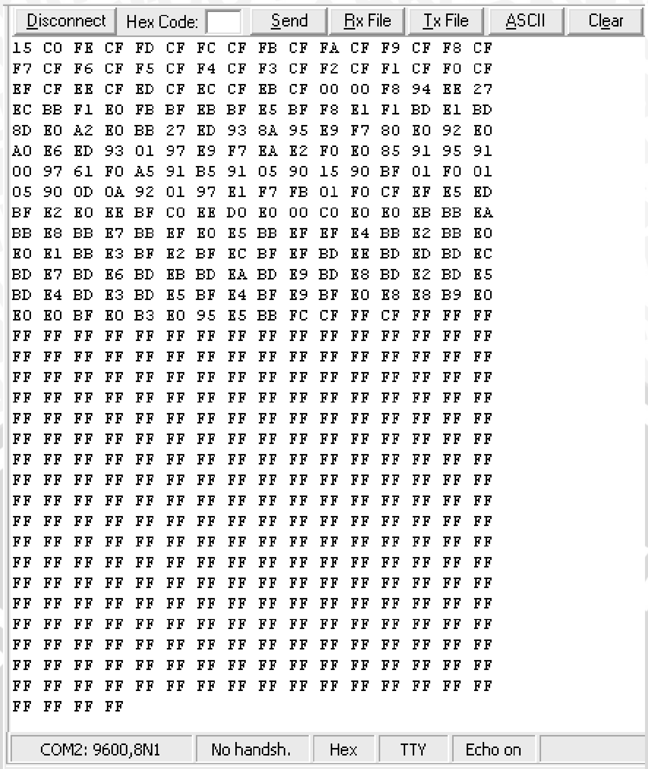
Gambar 5.14 Data memori dari sisi pemrogram yang tersimpan dalam SRAM eksternal

Paket data yang dikirimkan oleh sisi pemrogram berukuran 220 *byte*. Nilai tersebut adalah nilai *default* memori setelah dihapus tanpa operasi penulisan memori. Paket data yang dikirimkan kemudia diterima dan disimpan di dalam SRAM eksternal oleh sisi target. Rekaman data yang tersipan di dalam SRAM ditunjukkan dalam Gambar 5.14.

Pengujian penampungan data memori dari hasil pembacaan mikrokontroler target menggunakan instrumen *serial RS232 to USB converter*. Operasi memori yang digunakan pada pengujian ini adalah operasi baca memori EEPROM. Mikrokontroler target yang digunakan adalah ATmega8535 yang memiliki EEPROM berukuran 512 *byte*. Pembacaan terhadap memori EEPROM pada mikrokontroler target akan menghasilkan 512 *byte* data yang disimpan di dalam SRAM eksternal. Prosedur pengujian ini diilustrasikan dalam Gambar 5.15.



Gambar 5.15 Prosedur pengujian penampungan data dari mikrokontroler target pada SRAM



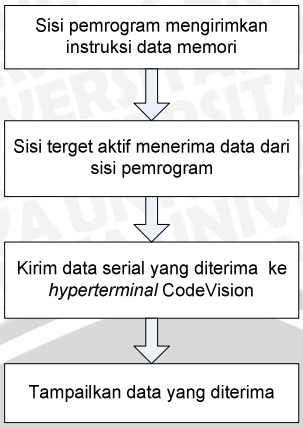
Gambar 5.16 Data memori mikrokontroler target yang tersimpan dalam SRAM eksternal

Gambar 5.16 menunjukkan rekaman data berukuran 512 *byte* yang tersimpan di dalam SRAM eksternal setelah sisi target melakukan operasi baca memori EEPROM. Dari hasil pengujian ini dapat dibandingkan dengan data pengujian dalam Gambar 5.15 dan Gambar 5.16 yang menunjukkan kesamaan data.

5.4.2 Pengujian Pengiriman Instruksi

5.4.2.1 Pengujian Pengiriman Instruksi dari Sisi Pemrogram ke Sisi Target

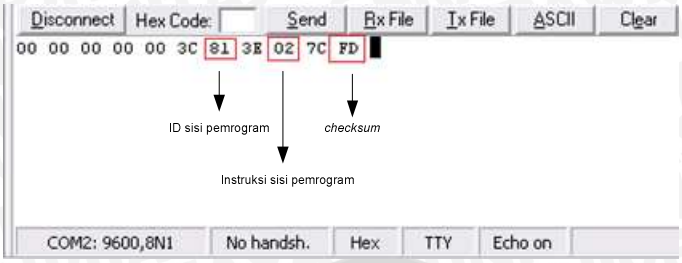
Pengujian penerimaan instruksi dan data memori dari sisi pemrogram menggunakan instrumen *serial RS232 to USB converter*. Seluruh data instruksi yang terdapat pada paket perintah, data memori pada paket data serta paket akhir yang diterima oleh sisi target di kirimkan ke *Hyperterminal CodeVision AVR* pada komputer. Hal ini bertujuan untuk merekam setiap data yang diterima sisi target kemudian disimpan dan dianalisis. Prosedur pengujian ini diilustrasikan dalam Gambar 5.17.



Gambar 5.17 Prosedur pengujian penerimaan perintah dan data dari sisi pemrogram

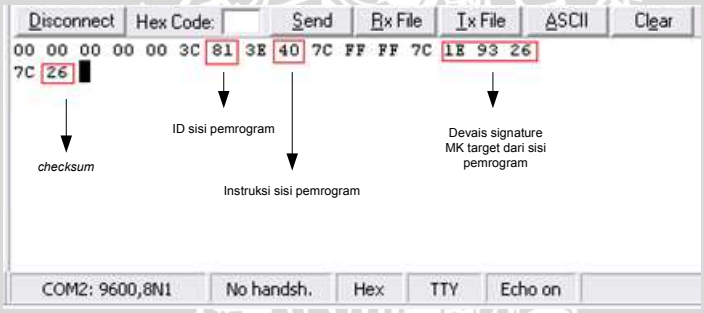
Operasi berdasarkan instruksi sisi pemrogram meliputi operasi pilih mikrokontroler dan seluruh operasi *serial downloading* untuk proses tulis baca memori Flash maupun EEPROM, operasi tulis baca fuse bit serta operasi hapus *chip* memori. Sedangkan operasi baca *device signature* tidak dilakukan karena operasi ini mengawali operasi *serial downloading* yang lain. Beberapa contoh hasil pengujian penerimaan perintah dan data dari sisi pemrogram yang ditampilkan pada layar monitor komputer adalah pengujian operasi pilih mikrokontroler, pengujian operasi baca memori Flash, pengujian operasi tulis memori Flash serta pengujian operasi hapus memori *chip*. Keempat hasil pengujian tersebut cukup mewakili dari seluruh operasi yang ada. Sedangkan untuk operasi memori yang lain dapat diwaikili oleh operasi memori baca tulis dan hapus flash karena operasi memori yang sejenis memiliki format data yang sama dan hanya dibedakan oleh instruksi operasi memori tersebut.

Hasil pengujian terhadap penerimaan instruksi berupa operasi pilih mikrokontroler ditunjukkan dalam Gambar 5.18. Format penerimaan data sesuai dengan protokol hasil perancangan. Mikrokontroler target yang digunakan pada pengujian ini adalah mikrokontroler kedua yaitu ATmega8535. Format penerimaan data terdiri atas ID sisi pemrogram yaitu 0x81, ID mikrokontroler 0x02 serta *checksum* untuk pendeteksi kesalahan penerimaan data.



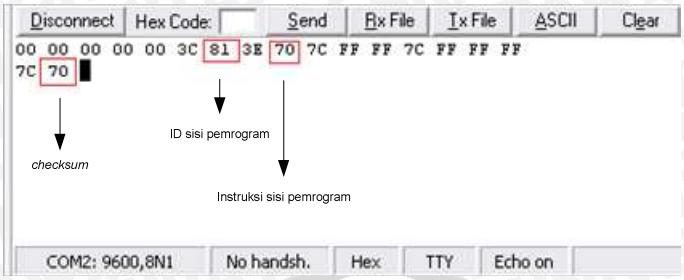
Gambar 5.18 Penerimaan instruksi operasi pilih mikrokontroler

Hasil pengujian terhadap penerimaan instruksi berupa operasi baca memori Flash ditunjukkan dalam Gambar 5.19. Operasi baca memori Flash mewakili operasi memori sejenisnya yaitu operasi baca EEPROM serta baca fuse bit. Format penerimaan data sesuai dengan protokol hasil perancangan. Mikrokontroler target yang digunakan pada pengujian ini adalah mikrokontroler ATmega8535. Format penerimaan data terdiri atas ID sisi pemrogram yaitu 0x81, instruksi sisi pemrogram untuk operasi baca memori Flash yaitu 0x20, 3 *byte device signature* ATmega8535 yaitu 0x1E 0x93 0x08, serta *checksum* untuk pendeteksi kesalahan penerimaan data.



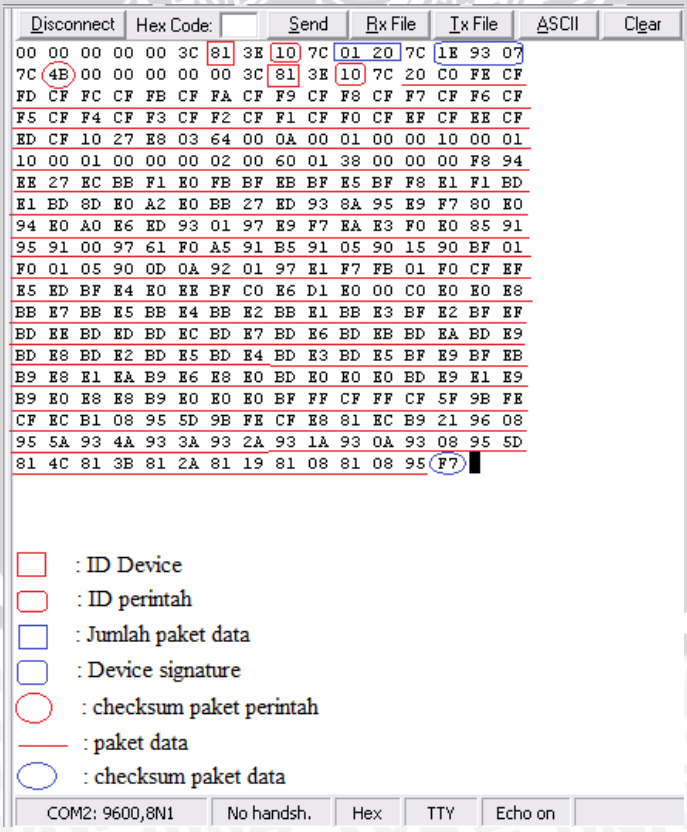
Gambar 5.19 Penerimaan instruksi operasi baca memori EEPROM

Hasil pengujian terhadap penerimaan instruksi berupa operasi hapus memori *chip* ditunjukkan dalam Gambar 5.20. Pada dasarnya format penerimaan data untuk operasi hapus memori *chip* sama dengan format penerimaan data pada operasi baca data memori. Perbedaannya terletak pada kode devais (*device signature*) yang tidak terdapat pada format operasi hapus memori *chip* karena informasi ini tidak diperlukan dalam operasi ini. Data yang seharusnya berisi informasi kode devais diisi dengan sembarang nilai data oleh sisi pemrogram dan diabaikan pada sisi target.



Gambar 5.20 Penerimaan instruksi operasi hapus *chip* memori dari sisi pemrogram

Format penerimaan data untuk operasi tulis memori mikrokontroler target terdiri atas paket perintah dan paket data. Hasil pengujian terhadap penerimaan instruksi berupa operasi tulis memori Flash ditunjukkan dalam Gambar 5.21. Operasi tulis memori Flash mewakili operasi memori sejenisnya yaitu operasi tulis EEPROM serta tulis fuse bit. Format penerimaan data pada paket perintah sama dengan format paket perintah pada operasi hapus memori *chip* yaitu 2 *byte* berisi jumlah paket data yang akan dikirim dan 3 *byte* yang berisi *device signature*. Pengujian ini menggunakan *file* memori Flash berekstensi heksa berukuran 288 *byte*.

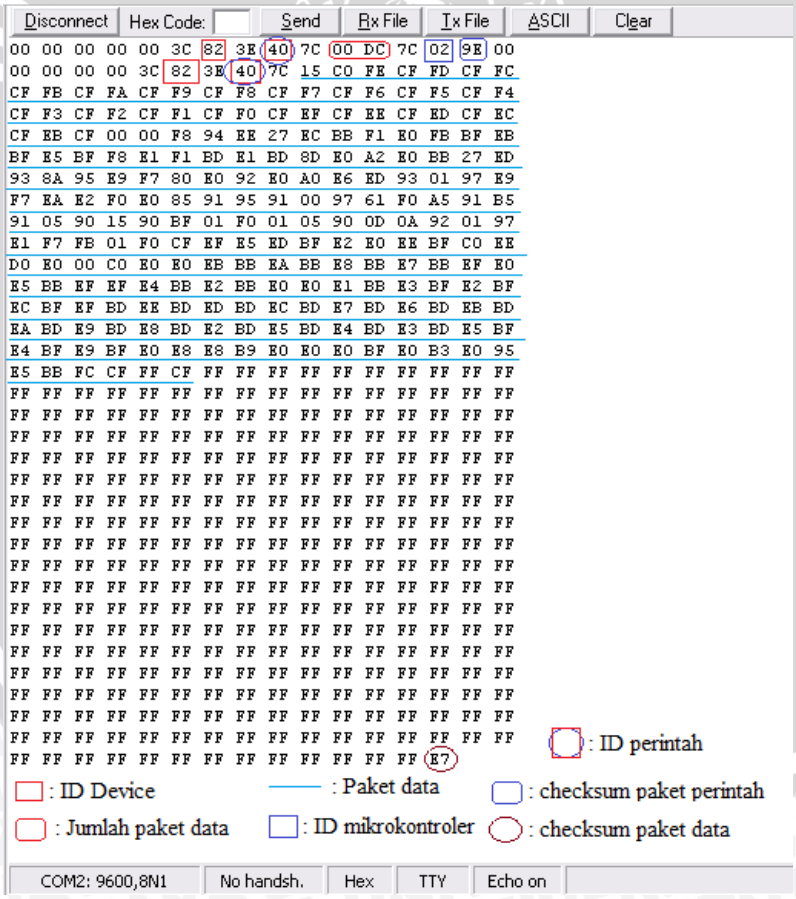


Gambar 5.21 Penerimaan instruksi operasi tulis memori Flash dari sisi pemrogram

Dari hasil pengujian, diketahui bahwa penerimaan instruksi dan data memori dari sisi pemrogram dapat diterima dengan baik oleh sisi target. Instruksi yang diterima meliputi operasi hapus memori *chip*, operasi baca dan tulis memori Flash, operasi baca dan tulis memori EEPROM, serta baca dan tulis konfigurasi fuse bit.

5.4.2.2 Pengujian Pengiriman Instruksi dari Sisi Target ke Sisi Pemrogram

Pengujian ini bertujuan untuk melihat respon sisi target terhadap permintaan data dari sisi pemrogram. Instrumen yang digunakan maupun prosedur pengujian ini sama dengan instrumen maupun prosedur pengujian penerimaan instruksi dan data memori dari sisi pemrogram. Sebagai contoh untuk pengujian ini adalah pengujian operasi baca memori EEPROM yang hasilnya ditunjukkan dalam Gambar 5.22.



Gambar 5.22 Data respon sisi target terhadap instruksi operasi baca memori EEPROM

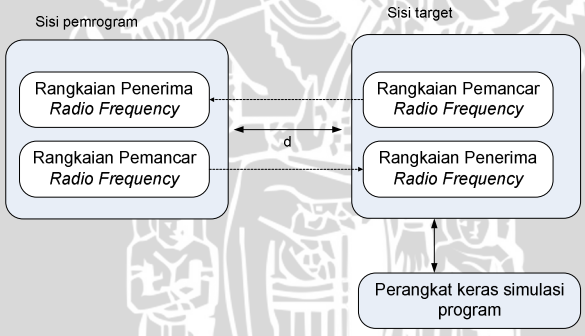
Dari hasil pengujian yang ditunjukkan dalam Gambar 5.22 dapat diamati bahwa ID sisi target adalah 0x82, kode instruksi untuk operasi baca memori

EEPROM adalah 0x40, satu *byte checksum* 0x07, dua *byte* jumlah paket data dan satu *byte* ID mikrokontroler. Hasil yang diperoleh ini sudah sesuai dengan perancangan. mikrokontroler target yang digunakan pada pengujian ini adalah ATmega8535 yang memiliki EEPROM berukuran 512 *byte*. Dari hasil pengujian, diketahui bahwa sisi target dapat merespon dengan baik permintaan data memori dari sisi pemrogram.

5.4.3 Pengujian Jarak Jangkau Maksimal Transmisi Data

Pengujian jarak jangkau maksimal transmisi data bertujuan untuk mengetahui jarak jangkauan maksimal penerimaan unit penerima r*adio frequency* saat menerima transmisi data dari unit pemancar r*adio frequency*. Dalam pengujian ini melibatkan pula sisi pemrogram sebagai pemancar transmisi data. Prosedur pengujian yang dilakukan:

- Menyusun instrumentasi pengujian seperti ditunjukkan dalam Gambar 5.23.



Gambar 5.23 Blok pengujian jarak jangkau maksimal transmisi data

- Transmisi data yang dilakukan adalah satu operasi tulis memori flash dengan data transmisi sebesar 288 *byte*. Terdiri dari paket perintah dan paket data.
- Perangkat keras pemancar r*adio frequency* yang digunakan adalah pemancar dengan pemancar r*adio frequency*. Selanjutnya secara bertahap mengubah jarak jangkau transmisi (d) setiap 1 meter kemudian mengamati proses transmisi. Setiap jarak jangkau dilakukan pengujian transmisi data sebanyak tiga kali. Hasil pengujian ditunjukkan dalam Tabel 5.3.

Tabel 5.3 Jarak jangkau transmisi data dengan r*adio frequency*

No.	Jarak Transmisi	Status Transmisi	Keterangan
1	1 meter	Berhasil	Data transmisi berhasil
2	2 Meter	Berhasil	Data transmisi berhasil
3	3 Meter	Berhasil	Data transmisi berhasil
4	4 Meter	Berhasil	Data transmisi berhasil
5	5 Meter	Berhasil	Data transmisi berhasil
6	6 Meter	Berhasil	Data transmisi berhasil
7	7 Meter	Berhasil	Data transmisi berhasil
8	8 Meter	Berhasil	Data transmisi berhasil
9	9 Meter	Berhasil	Data transmisi berhasil
10	10 Meter	Berhasil	Data transmisi berhasil
11	11 Meter	Berhasil	Data transmisi berhasil
12	12 Meter	Berhasil	Data transmisi berhasil
13	13 Meter	Berhasil	Data transmisi berhasil
14	14 Meter	Berhasil	Data transmisi berhasil
15	15 Meter	Berhasil	Data transmisi berhasil
16	16 Meter	Berhasil	Data transmisi berhasil
17	17 Meter	Berhasil	Data transmisi berhasil
18	18 Meter	Berhasil	Data transmisi berhasil
19	19 Meter	Berhasil	Data transmisi berhasil
20	20 Meter	Berhasil	Data transmisi berhasil
21	21 Meter	Berhasil	Data transmisi berhasil
22	22 Meter	Berhasil	Data transmisi berhasil
23	23 Meter	Berhasil	Data transmisi berhasil
24	24 Meter	Berhasil	Data transmisi berhasil
25	25 Meter	Berhasil	Data transmisi berhasil
26	26 Meter	Berhasil	Data transmisi berhasil
27	27 Meter	Berhasil	Data transmisi berhasil
28	28 Meter	Berhasil	Data transmisi berhasil
29	29 Meter	Berhasil	Data transmisi berhasil
30	30 Meter	Berhasil	Data transmisi berhasil

Dari hasil pengujian, maka jarak jangkau transmisi data diketahui bisa mencapai sejauh 30 meter dan ini sudah melebihi target dari perancangan semula yaitu 20 meter.

5.4.4 Pengujian Kecepatan Proses *Serial Downloading*

Pengujian ini bertujuan untuk mengetahui waktu yang diperlukan untuk operasi tulis maupun baca memori mikrokontroler target. Beberapa kondisi dalam pengujian ini adalah transmisi dianggap ideal tanpa gangguan dengan jarak sisi pemrogram dan sisi target sejauh 1 meter, serta mikrokontroler target yang digunakan adalah ATmega8535 dengan frekuensi *oscillator* kristal internal 8 MHz.

Variabel yang diubah-ubah pada pengujian ini adalah frekuensi SCK yang digunakan yaitu 86,4 kHz, 344 kHz, 691,2 kHz 1382,4 kHz serta 2764,8 kHz. Hasil pengujian terhadap terhadap waktu rata-rata yang diperlukan untuk operasi

baca memori menggunakan frekuensi SCK tersebut secara berturut-turut ditunjukkan dalam Tabel 5.4, Tabel 5.5, Tabel 5.6 Tabel 5.7 dan Tabel 5.8. Sedangkan hasil pengujian untuk operasi tulis memori secara berturut-turut ditunjukkan dalam Tabel 5.9, Tabel 5.10, Tabel 5.11, Tabel 5.12 dan Tabel 5.13.

Tabel 5.4 Lama operasi baca memori pada frekuensi SCK 86,4 kHz

No	Ukuran data (*byte*)	Memori target	Waktu yang dibutuhkan (detik)
1	8192	Flash	10
2	512	EEPROM	1.5
3	3	Fuse bit	1

Tabel 5.5 Lama operasi baca memori pada frekuensi SCK 344 kHz

No	Ukuran data (*byte*)	Memori target	Waktu yang dibutuhkan (detik)
1	8192	Flash	9.5
2	512	EEPROM	1.5
3	3	Fuse bit	1

Tabel 5.6 Lama operasi baca memori pada frekuensi SCK 691,2 kHz

No	Ukuran data (*byte*)	Memori target	Waktu yang dibutuhkan (detik)
1	8192	Flash	9.5
2	512	EEPROM	1.5
3	3	Fuse bit	1

Tabel 5.7 Lama operasi baca memori pada frekuensi SCK 1382,4 kHz

No	Ukuran data (*byte*)	Memori target	Waktu yang dibutuhkan (detik)
1	8192	Flash	9
2	512	EEPROM	1.5
3	3	Fuse bit	1

Tabel 5.8 Lama operasi baca memori pada frekuensi SCK 2764,8 kHz

No	Ukuran data (*byte*)	Memori target	Waktu yang dibutuhkan (detik)
1	8192	Flash	9
2	512	EEPROM	1.5
3	3	Fuse bit	1

Tabel 5.9 Lama operasi tulis memori pada frekuensi SCK 86,4 kHz

No	Ukuran data (*byte*)	Memori target	Waktu yang dibutuhkan (detik)
1	256	Flash	2
2	512	Flash	3.5
3	1024	Flash	6.5
4	2048	Flash	13
5	3072	Flash	19
6	4096	Flash	25
7	5120	Flash	31

8	6144	Flash	37
9	7168	Flash	43.5
10	8192	Flash	49
11	256	EEPROM	1
12	512	EEPROM	1.5
13	3	Fuse bit	1

Tabel 5.10 Lama operasi tulis memori pada frekuensi SCK 344 kHz

No	Ukuran data (*byte*)	Memori target	Waktu yang dibutuhkan (detik)
1	256	Flash	1.5
2	512	Flash	3.5
3	1024	Flash	6
4	2048	Flash	11.5
5	3072	Flash	17
6	4096	Flash	22.5
7	5120	Flash	27.5
8	6144	Flash	33
9	7168	Flash	38.5
10	8192	Flash	44.5
11	256	EEPROM	1
12	512	EEPROM	1.5
13	3	Fuse bit	1

Tabel 5.11 Lama operasi tulis memori pada frekuensi SCK 691,2 kHz

No	Ukuran data (*byte*)	Memori target	Waktu yang dibutuhkan (detik)
1	256	Flash	1.5
2	512	Flash	3.5
3	1024	Flash	6
4	2048	Flash	11.5
5	3072	Flash	17
6	4096	Flash	22
7	5120	Flash	27
8	6144	Flash	32.5
9	7168	Flash	38
10	8192	Flash	43.5
11	256	EEPROM	1
12	512	EEPROM	1.5
13	3	Fuse bit	1

Tabel 5.12 Lama operasi tulis memori pada frekuensi SCK 1382,4 kHz

No	Ukuran data (*byte*)	Memori target	Waktu yang dibutuhkan (detik)
1	256	Flash	1.5
2	512	Flash	3.5
3	1024	Flash	6
4	2048	Flash	11
5	3072	Flash	16.5

6	4096	Flash	22.5
7	5120	Flash	26.5
8	6144	Flash	32
9	7168	Flash	37.5
10	8192	Flash	42.5
11	256	EEPROM	1
12	512	EEPROM	1.5
13	3	Fuse bit	1

Tabel 5.13 Lama operasi tulis memori pada frekuensi SCK 2764,8 kHz

No	Ukuran data (*byte*)	Memori target	Waktu yang dibutuhkan (detik)
1	256	Flash	1.5
2	512	Flash	3.5
3	1024	Flash	6
4	2048	Flash	11
5	3072	Flash	16.5
6	4096	Flash	2.5
7	5120	Flash	26.5
8	6144	Flash	32
9	7168	Flash	37.5
10	8192	Flash	42.5
11	256	EEPROM	1
12	512	EEPROM	1.5
13	3	Fuse bit	1

Beberapa hal yang perlu diperhatikan dari data hasil pengujian tersebut adalah waktu yang diperlukan untuk proses tulis dipengaruhi oleh jenis memori target (Flash atau EEPROM) serta ukuran data yang ditulis. Secara umum kesimpulan yang dapat diambil dari hasil pengujian ini adalah proses *serial downloading* paling cepat terjadi pada saat frekuensi SCK sebesar 1382.4 kHz. Frekuensi SCK ini mengoptimalkan kecepatan operasi memori mikrokontroler target yang menggunakan *oscillator* dengan frekuensi 8 MHz. Berdasarkan hasil pengujian ini, maka nilai frekuensi SCK yang dipilih untuk digunakan pada sistem ini adalah 1382.4 kHz.

5.4.5 Pengujian Pengiriman Data

Pengujian ini bertujuan untuk mengetahui tingkat keberhasilan sistem dalam hal transmisi data memori sampai dengan 32K *byte*. Beberapa kondisi dalam pengujian ini adalah transmisi dianggap ideal tanpa gangguan serta mikrokontroler target yang digunakan adalah ATmega32 dengan frekuensi

oscillator kristal internal 8 MHz. Frekuensi SCK yang digunakan adalah 1382.4 kHz. Jenis operasi yang dipilih pada pengujian ini adalah operasi tulis memori Flash. Hasil pengujian ini terdapat dalam Tabel 5.14 yang sekaligus menunjukkan waktu rata-rata yang diperlukan untuk operasi tulis memori Flash.

Tabel 5.14 Lama operasi tulis data memori

No	Ukuran data (K*byte*)	Memori target	Waktu yang dibutuhkan (detik)
1	2	Flash	11
2	4	Flash	25
3	6	Flash	32
4	8	Flash	42.5
5	12	Flash	46
6	16	Flash	62.5
7	24	Flash	93
8	32	Flash	118

Kesimpulan yang dapat diambil dari hasil pengujian ini adalah sistem mampu melakukan transmisi sekaligus operasi data memori sampai dengan 32K *byte*. Jika hasil pengujian ini dipadukan dengan hasil pengujian operasi memori mikrokontroler ATmega8535, maka dapat disimpulkan bahwa sistem mampu melakukan operasi memori pada mikrokontroler keluarga Atmel AVR.

BAB VI
KESIMPULAN DAN SARAN

6.1 Kesimpulan

Dari hasil perancangan dan pengujian, diperoleh kesimpulan sebagai berikut:

1) Unit frekuensi radio yang terdiri dari satu modul pemancar dan penerima frekuensi radio YS-1020 dapat bekerja dengan baik, dengan *baudrate* transmisi 9600 bps dan jarak jangkau transmisi hingga 30 meter atau bahkan lebih.
2) Protokol komunikasi data melalui media frekuensi radio dibangun satu tingkat di atas protokol USART dengan format frame 1 bit start, 8 bit data serta 1 bit stop pada mikrokontroler. Paket transmisi yang terdiri atas paket perintah, paket data dan paket konfirmasi.
3) Perangkat lunak untuk operasi memori flash, EEPROM, dan fusebits dirancang menggunakan metode ISP (*In-System Programming*) dengan antarmuka SPI (*Serial Peripheral Interface*) serta jenis konektor yang digunakan adalah konektor standart Kanda Sistem STK200/300.
4) Sistem mampu menangani operasi memori empat buah mikrokontroler secara bargantian dengan adanya sesi pilih mikrokontroler target.
5) Sistem mampu memberi peringatan kepada pengguna apabila ada ketidakcocokan antara jenis devais yang dipilih pada program PC dengan devais yang terpasang pada sisi target.

6.2 Saran

Berdasarkan hasil pengujian terhadap kinerja sistem yang dirancang, maka beberapa hal yang dapat direkomendasikan untuk pengembangan dan penelitian lebih lanjut adalah sebagai berikut :

1) Sistem dikembangkan agar dapat menangani devais dengan kapasitas memori lebih dari 32 kbyte.
2) Sistem pada sisi target perlu dilengkapi dengan algoritma pendeteksian frekuensi *clock* mikro-kontroler target untuk keperluan kalibrasi frekuensi *clock* SCK yang digunakan.

3) Dimungkinkan sistem dapat dikembangkan untuk menangani operasi memori pada mikrokontroler selain dari keluarga Atmel AVR.

DAFTAR PUSTAKA

Atmel. 2002. AVR910*: In-System Programming. USA. Atmel Corporation.* http://atmel.com/dyn/resources/prod_documents/doc0943.pdf. Tanggal akses 7 Desember 2010.

Atmel. 2002a. *AVR236: CRC Check of Program Memory*. Atmel. http://atmel.com/dyn/resources/prod_documents/doc1143.pdf. Tanggal akses 7 Desember 2010.

Atmel. 2007. *8-bit AVR with 16K Bytes In-System Programmable Flash ATmega162, ATmega162L*. USA: Atmel Corporation. http://atmel.com/dyn/resources/prod_documents/doc2513.pdf. Tanggal akses 7 Desember 2010.

Arief Mardhi Basuki, Nanang Sulistyanto, Adharul Muttaqin.2008. *Sistem Downloader Data Memori Mikrokontroler Atmel-AVR Melalui Inframerah : Sisi Target*. Malang. Departemen Pendidikan Nasional Universitas Brawijaya Fakultas Teknik Jurusan Teknik Elektro

Gadre, Dhananjay V. 2001. *Programming And Customizing The AVR Microcontroller*. USA. McGraw-Hill

Shenzen Yishi. 2009. *YS-1020 RF Data Transciever*. Hongkong. Shenzen Yishi *Electronic Technology.* http://www.rfidglobal.org/uploadfiles/2008_2/2008021861669329.pdf. Tanggal akses 4 Desember 2010.

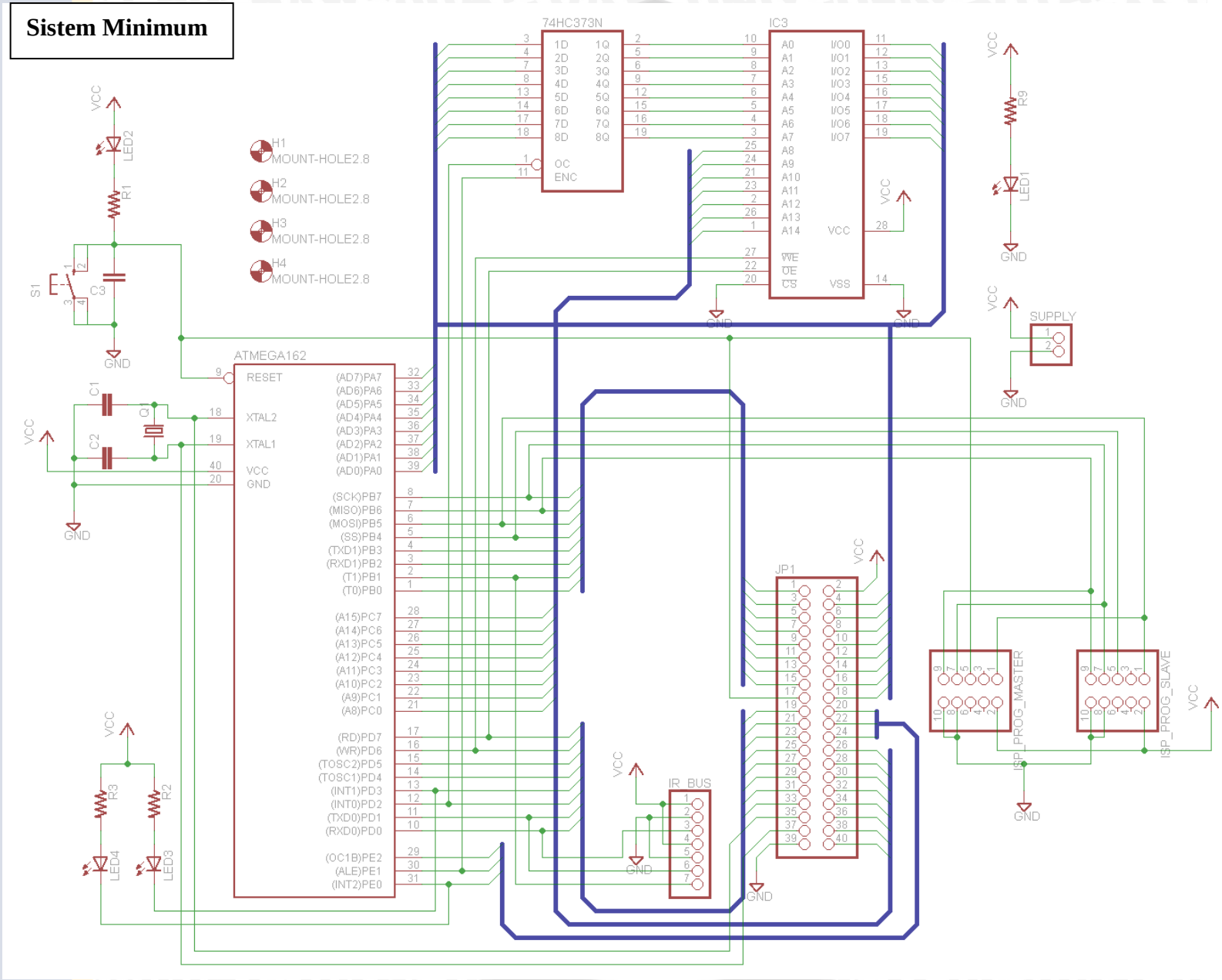
Texas. 2003. *CD54HC373, CD74HC373 Octal Transparent D-Type Latches with 3-State Outputs*. Texas Instrumen. http://pdf1.alldatasheet.com/datasheet-pdf/view/133389/TI/HC373/+03_49AVRh/1HEBzN+/datasheet.pdf. Tanggal akses 7 Desember 2010.

Utron. 2001. UT*62256 32k x 8bit low power CMOS SRAM*. Utron Electronics. http://www.100y.com.tw/pdf_file/UT62256.pdf. Tanggal akses 7 Desember 2010.

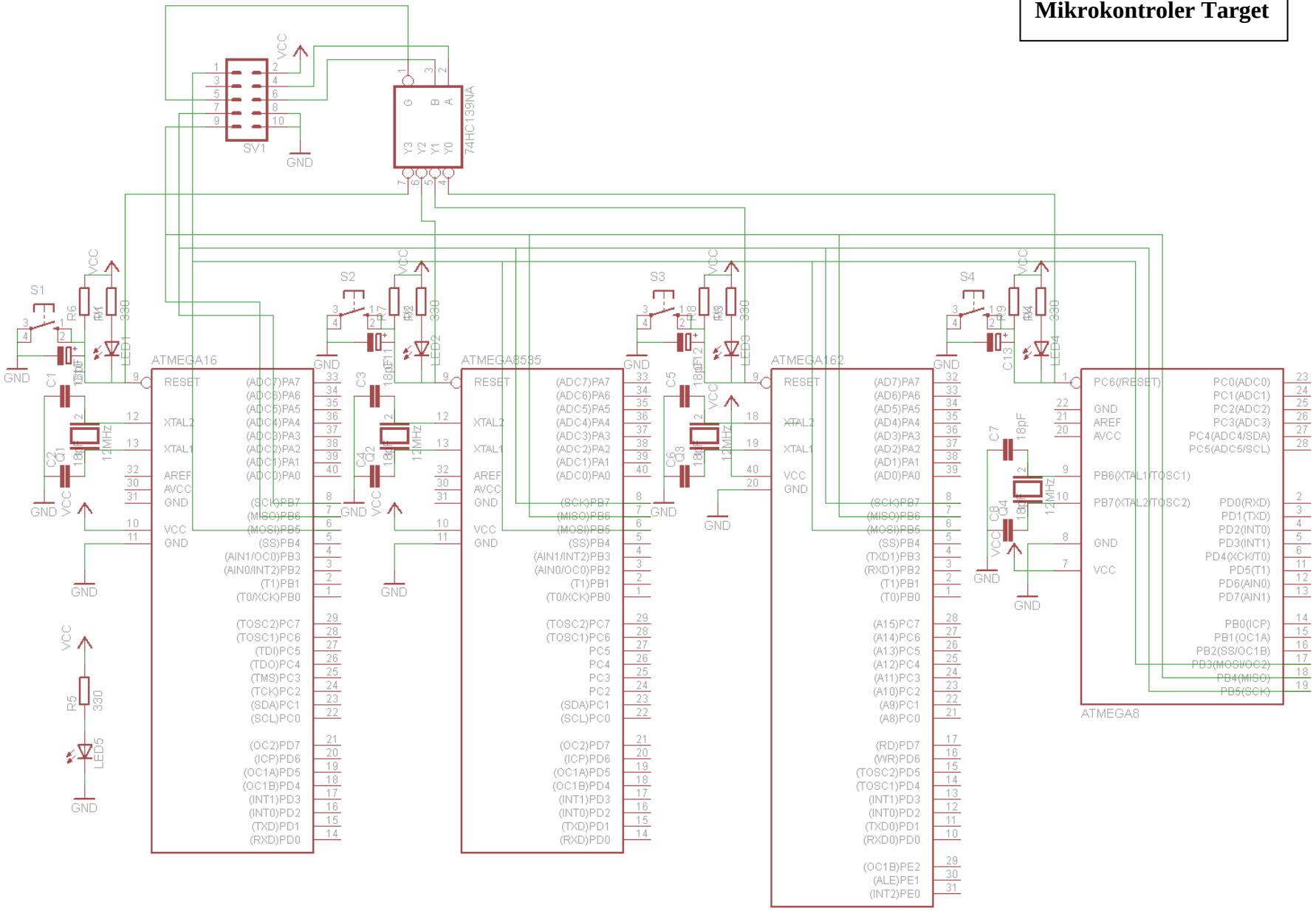
Lampiran I

Skema Rangkaian

Sistem Minimum

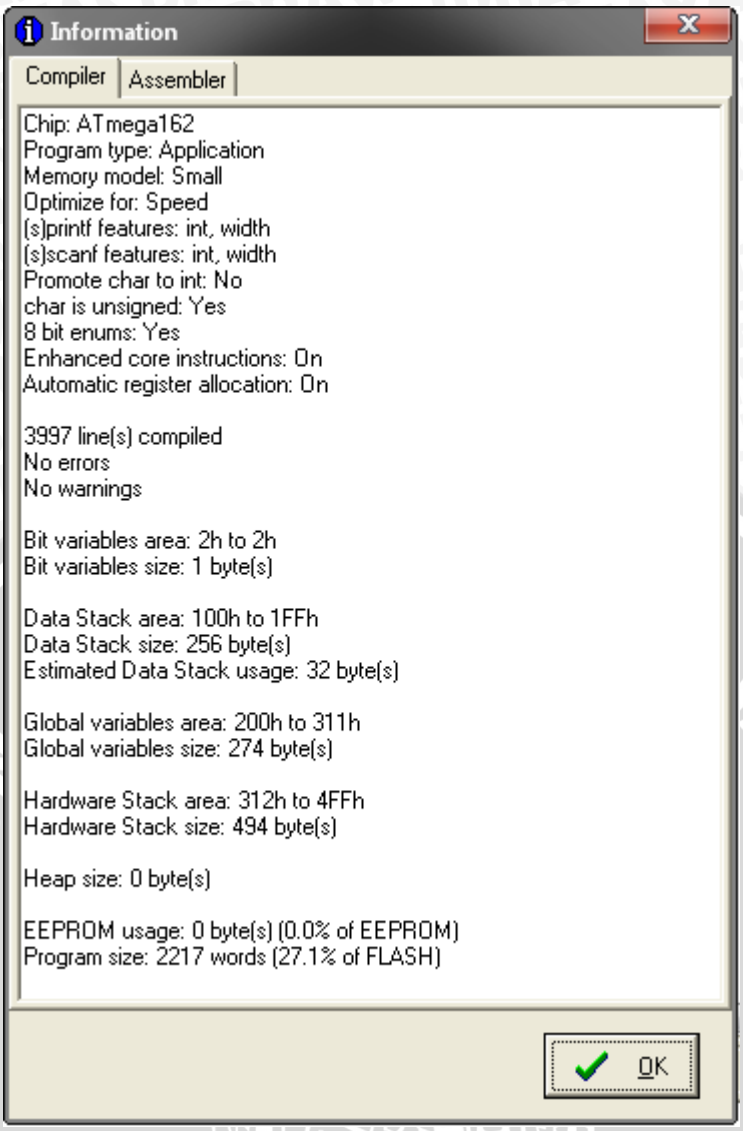


Mikrokontroler Target



Lampiran II

Listing Program



Gambar 1. Tampilan kapasitas software yang digunakan dalam sistem *downloader*

Keterangan :

1. Jenis mikrokontroler yang digunakan yaitu Atmega162.
2. Program yang dibutuhkan untuk menulis sistem yaitu 3997 baris.
3. Jumlah variabel yang digunakan yaitu 1 byte variabel bit, ukuran data stack 32 byte dan 274 byte variabel global.
4. Memori *flash* yang dibutuhkan untuk menulis program sistem *downloader* adalah 27,1% atau sekitar 4,336 Kbyte (16 Kbyte x 27.1%).

Perangkat Lunak Mikrokontroler

main.c

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.25.7 beta 5 Professional
Automatic Program Generator
© Copyright 1998-2007 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project : Program Skripsi
Version :
Date                    : 2010/09/15
Author                  : Anang Fakhrudin
Company                 : Hacker
Comments:


Chip type               : ATmega162
Program type            : Application
Clock frequency         : 11.059200 MHz
Memory model            : Small
External SRAM size      : 32768
Ext. SRAM wait state    : 0
Data Stack size         : 256
*****************************************************/

#include <mega162.h>
#include <delay.h>
#include "crc/crc.h"
#include "eeprom/eeprom.h"
#include "extsram/extsram.h"
#include "flash/flash.h"
#include "fuse/fuse.h"
#include "isp/isp.h"
#include "usart/usart.h"
#include "multitarget/multitarget.h"

#define TULIS_FLASH_MEMORY      0x10
#define BACA_FLASH_MEMORY       0x20
#define TULIS_EEPROM            0x30
#define BACA_EEPROM             0x40
#define TULIS_FUSE_BITS         0x50
#define BACA_FUSE_BITS          0x60
#define HAPUS_CHIP              0x70

#define mikrokontroler0         0
#define mikrokontroler1         0x01 // ATmega8
#define mikrokontroler2         0x02 // ATmega162
#define mikrokontroler3         0x03 // ATmega8535
#define mikrokontroler4         0x04 // ATmega16

#define SEI()           #asm("sei")
#define CLI()           #asm("cli")

//D-Latch
#define AKTIFKAN_LATCH()        PORTD &= ~(1<<2)
#define NON_AKTIFKAN_LATCH()    PORTD |= (1<<2)

#define PIN_TX_OFF()            PORTD &= ~(1<<1)
#define PIN_TX_ON()             PORTD |= (1<<1)

#define LED_BIRU_ON()           PORTD &= ~(1<<3)
#define LED_BIRU_OFF()          PORTD |= (1<<3)
#define LED_MERAH_ON()          PORTE &= ~(1<<0)
#define LED_MERAH_OFF()         PORTE |= (1<<0)

unsigned int idxBufExtSRAM;             //Indek untuk akses buffer data pada SRAM
unsigned int jml_byte_data;             //Jumlah byte data yang diterima -> deteksi paket sram

unsigned char devais_signature[3];
unsigned char fuse_bits[3];
unsigned char kode_devais[3];
unsigned char jml_paket0;
unsigned char jml_paket1;
```

```
unsigned char ID_perintah;
unsigned char ID_mikrokontroler;
unsigned char ID_mikro;
unsigned char data;
unsigned char instruksi_pemrogram;
unsigned char kepala[2];
unsigned char checksum;

bit status_paket_perintah;
bit status_paket_data;
bit status_konfirmasi;
bit tunggu_konfirmasi;
bit status_perintah_tulis;
bit eksistensi_devais;
bit status_kirim;
bit status_MK_pilih;

/*********************************************/
//Fungsi-Fungsi Interrupt Terima Data Serial RF
/*********************************************/
/*****************************************************************************************************
*****/
// USART0 Receiver buffer
#define RX_BUFFER_SIZE0 256
char rx_buffer0[RX_BUFFER_SIZE0];
unsigned int rx_wr_index0,rx_rd_index0,rx_counter0;
/*****************************************************************************************************
*****/
// USART0 Receiver interrupt service routine
interrupt [USART0_RXC] void usart0_rx_isr(void)
{
   unsigned char status,data;
   status=UCSR0A;
   data=UDR0;
   if ((status & ((FRAMING_ERROR)|(DATA_OVERRUN)))==0)
     {
        rx_buffer0[rx_wr_index0]= data;
        if (++rx_wr_index0 == RX_BUFFER_SIZE0) rx_wr_index0= 0;
        if (++rx_counter0 == RX_BUFFER_SIZE0) rx_counter0= 0;
     }
}
/*****************************************************************************************************
*****/
// Fungsi Pembacaan Data Serial pada Buffer Interrupt RXC
unsigned char bacaDataSerialRF(void)
{
   unsigned char data;

   while (rx_counter0==0);
   data= rx_buffer0[rx_rd_index0];
   if (++rx_rd_index0 == RX_BUFFER_SIZE0) rx_rd_index0= 0;
   CLI();
   --rx_counter0;
   SEI();
   return data;
}

/*****************************************************************************************************
*****/

void led_kedip (void)
{
     LED_BIRU_OFF();
     LED_MERAH_OFF();
     delay_ms(50);
     LED_BIRU_ON();
     LED_MERAH_ON();
     delay_ms(5);
     LED_BIRU_OFF();
     LED_MERAH_OFF();
     delay_ms(50);
     LED_BIRU_ON();
     LED_MERAH_ON();
     delay_ms(15);
     LED_BIRU_OFF();
```

```
    LED_MERAH_OFF();
    delay_ms(50);
    LED_BIRU_ON();
    LED_MERAH_ON();
    delay_ms(5);
    LED_BIRU_OFF();
    LED_MERAH_OFF();
    delay_ms(80);
    LED_BIRU_ON();
    LED_MERAH_ON();
    delay_ms(80);
    LED_BIRU_OFF();
    LED_MERAH_OFF();
}

/*****************************************************************************************************
*****/

void kirimPerintahKePemrogram(void)
{
unsigned char i;
unsigned char buffer_paket_data[10];

CLI();
buffer_paket_data[0]= '<';
buffer_paket_data[1]= 0x82;      // ID Sisi Target
buffer_paket_data[2]= '>';
buffer_paket_data[3]= ID_perintah;
buffer_paket_data[4]= '|';
buffer_paket_data[5]= jml_paket0;  // Jumlah Paket yang akan dikirimkan
buffer_paket_data[6]= jml_paket1;
buffer_paket_data[7]= '|';
buffer_paket_data[8]= ID_mikro;
buffer_paket_data[9]= '|';

for(i= 0; i <=9 ; i++)
   kirimDataSerialRF(buffer_paket_data[i]);

SEI();
}

void kirimDataKePemrogram(void)
{
unsigned int i;
unsigned char buffer_paket_data[3];

CLI();
buffer_paket_data[0]= '<';
buffer_paket_data[1]= 0x82;
buffer_paket_data[2]= '>';

for(i=0; i<=2; i++)
      {
      kirimDataSerialRF(buffer_paket_data[i]);
      };
for(i=0; i<jml_byte_data; i++)
      {
      data=bacaExtSRAM(i);
      kirimDataSerialRF(data);
      };
kirimDataSerialRF('O');
kirimDataSerialRF('K');

SEI();
}

void olahKonfirmasi(void)
{
        unsigned char data_konfirmasi;
        unsigned char buffer_konfirmasi[4];
        unsigned char indek_konfirmasi;

        while(tunggu_konfirmasi)
        {
                data_konfirmasi= bacaDataSerialRF();
```

```
                //kirimDataSerial(data_konfirmasi);
                buffer_konfirmasi[indek_konfirmasi++]= data_konfirmasi;

                if((indek_konfirmasi == 1 && data_konfirmasi != '|') || (indek_konfirmasi == 3 && data_konfirmasi != '|'))
              indek_konfirmasi= 0;
                else if(indek_konfirmasi == 4)
                {
                        if((buffer_konfirmasi[1] == 0xAA) && (data_konfirmasi == 0xAA))
                        {
                                status_konfirmasi= 1;
                                tunggu_konfirmasi= 0;
                                indek_konfirmasi= 0;
                        }

                    else
                    {
                          tunggu_konfirmasi= 0;
                      indek_konfirmasi= 0;
                    }
                }
        };
}

void kirimKonfirmasiOK(void)
{
        CLI();

//      kirimDataSerialRF('<');
//      kirimDataSerialRF(0x82);
//      kirimDataSerialRF('>');
        kirimDataSerialRF('|');
        kirimDataSerialRF(0xAA);
        kirimDataSerialRF('|');
        kirimDataSerialRF(0xAA);

        kirimDataSerial('|');  // debug log konfirmasi USB
        kirimDataSerial(0xAA);
        kirimDataSerial('|');
        kirimDataSerial(0xAA);

        SEI();
}


void kirimKonfirmasiGAGAL(void)
{
        CLI();

//      kirimDataSerialRF('<');
//      kirimDataSerialRF(0x82);
//      kirimDataSerialRF('>');
        kirimDataSerialRF('|');
        kirimDataSerialRF(0x55);
        kirimDataSerialRF('|');
        kirimDataSerialRF(0x55);

        kirimDataSerial('|');   // debug log konfirmasi USB
        kirimDataSerial(0x55);
        kirimDataSerial('|');
        kirimDataSerial(0x55);

        SEI();
}

void inisialisasi(void)
{
   CLI();
   kirimDataSerial(0xAA);

   //status eksistensi devais yang terpasang pada sisi target
   eksistensi_devais=0;

   //Jumlah paket data maksimal
   jml_paket0= 0;
   jml_paket1= 0;
```

```
    //Status Paket Data Transmisi
    status_paket_perintah= 1;
    status_paket_data= 0;

    //Status konfirmasi
    //OK= 1, GAGAL= 0
    status_konfirmasi= 0;
    tunggu_konfirmasi= 0;

    //status untuk terima data dari pemrogram
    status_kirim=0;

    //status untuk perintah tulis dari pemrogram
    status_perintah_tulis=0;

    //Indek untuk pengalamatan SRAM eksternal
    idxBufExtSRAM= 0;

    //Set jumlah data byte yang diterima
    jml_byte_data= 0;

    //Set LED indikator;
    LED_MERAH_OFF();
    LED_BIRU_OFF();
    RESET_SLAVE_NONAKTIF();

    //Pengkondisian awal transmisi
    AKTIFKAN_RECEIVER_RF();
    AKTIFKAN_RECEIVER_INTERRUPT_RF();

    //Aktifkan Global Interrupt Enable
    SEI();
}

void kirimPaketDataKePemrogram(void)
{

    //Status Paket Data Transmisi
    status_paket_perintah= 1;
    status_paket_data= 0;

    //Status konfirmasi
    //OK= 1, GAGAL= 0
    status_konfirmasi= 0;
    tunggu_konfirmasi= 0;

    while(status_paket_perintah==1)
    {
       kirimPerintahKePemrogram();
       tunggu_konfirmasi= 1;
       AKTIFKAN_RECEIVER_RF();
       AKTIFKAN_RECEIVER_INTERRUPT_RF();
       olahKonfirmasi();
       MATIKAN_RECEIVER_INTERRUPT_RF();
       MATIKAN_RECEIVER_RF();
       if (status_konfirmasi==1)
       {
       status_paket_perintah=0;
       status_paket_data=1;
       }

    };

    //Status konfirmasi
    //OK= 1, GAGAL= 0
    status_konfirmasi= 0;
    while(status_paket_data==1)
    {
       kirimDataKePemrogram();
       tunggu_konfirmasi= 1;
       AKTIFKAN_RECEIVER_RF();
       AKTIFKAN_RECEIVER_INTERRUPT_RF();
       olahKonfirmasi();
       MATIKAN_RECEIVER_INTERRUPT_RF();
```

```
        MATIKAN_RECEIVER_RF();
        if (status_konfirmasi==1)
        {
        status_paket_data=0;
        }
    };

    //Reset indek buffer interrupt RXC
    rx_wr_index0= 0;
    rx_rd_index0= 0;
    rx_counter0= 0;

}

unsigned int bacaUkuranFlash(void)
{
    // Ukuran memori flash berdasarkan devais signature
    // 0x90 -->  1024 byte
    // 0x91 -->  2048 byte
    // 0x92 -->  4096 byte
    // 0x93 -->  8192 byte
    // 0x94 --> 16384 byte
    // 0x95 --> 32768 byte
    // 0x96 --> 65536 byte

    unsigned char kode_devais;
    unsigned int ukuran_flash;

    kode_devais= devais_signature[1];
    CLI();
    switch(kode_devais)
    {
        case 0x90:
            ukuran_flash= 1024;
            break;
        case 0x91:
            ukuran_flash= 2048;
            break;
        case 0x92:
            ukuran_flash= 4096;
            break;
        case 0x93:
            ukuran_flash= 8192;
            break;
        case 0x94:
            ukuran_flash= 16384;
            break;
        case 0x95:
            ukuran_flash= 32768;
            break;
        case 0x96:
            ukuran_flash= 65536;
            break;
        default : ukuran_flash= 0;
    }
    return ukuran_flash;
}

void tulisMemoriFlash(void)
{
    unsigned int jml_page_data;
    jml_page_data= 0;

    CLI();
    jml_page_data= (unsigned int) (idxBufExtSRAM/2); //Benar bisa jalan OK
    LED_MERAH_ON();
    enterProgrammingMode();
    chipErase();
    ISPWriteFlashMemory(jml_page_data);
    leaveProgrammingMode();
    LED_MERAH_OFF();
    kirimKonfirmasiOK();
    SEI();
}
```

```
void tulisEEPROM(void)
{
   jml_byte_data= 0;

   CLI();
   jml_byte_data= idxBufExtSRAM;
   LED_MERAH_ON();
   enterProgrammingMode();
   ISPWriteEEPROM(jml_byte_data);
   leaveProgrammingMode();
   LED_MERAH_OFF();
   kirimKonfirmasiOK();
   SEI();
}

void bacaMemoriFlash(void)
{
   unsigned int flash_page_max;
   unsigned int ukuran_flash_max;

   CLI();
   //Baca flash memory kemudian data disimpan dalam SRAM Eksternal
   ukuran_flash_max= bacaUkuranFlash();
   flash_page_max= ukuran_flash_max/2; //OK sudah bisa

   LED_MERAH_ON();
   enterProgrammingMode();
   ISPReadFlashMemory(flash_page_max);
   leaveProgrammingMode();
   LED_MERAH_OFF();
   //Pengiriman data pembacaan flash ke sisi pemrogram
   jml_byte_data=ukuran_flash_max;
   jml_paket1 = (unsigned char) jml_byte_data;
   jml_paket0 = (unsigned char)(jml_byte_data>>8);
   kirimPaketDataKePemrogram();

   SEI()
}

void bacaEEPROM(void)
{
   unsigned int ukuran_flash_max;
   unsigned int EEPROM_size_max;

   CLI();
   //Baca EEPROM memory kemudian data disimpan dalam SRAM Eksternal
   ukuran_flash_max= bacaUkuranFlash();
   EEPROM_size_max= (unsigned int)(ukuran_flash_max/16); //OK sudah bisa
   LED_MERAH_ON();
   enterProgrammingMode();
   ISPReadEEPROM(EEPROM_size_max);
   leaveProgrammingMode();
   LED_MERAH_OFF();
   SEI();

   //Pengiriman data pembacaan EEPROM ke sisi pemrogram
   jml_byte_data=EEPROM_size_max;
   jml_paket1 = (unsigned char) jml_byte_data;
   jml_paket0 = (unsigned char)(jml_byte_data>>8);
   kirimPaketDataKePemrogram();
}

void bacaFuseBits(void)
{
   LED_MERAH_ON();
   /*** Set SCK dengan frekuensi rendah ***/
   inisialisasiSPILowSpeed();
   enterProgrammingMode();
   fuse_bits[0]= ISPReadLowFuseBits();
   fuse_bits[1]= ISPReadHighFuseBits();
   fuse_bits[2]= ISPReadExtendedFuseBits();
   leaveProgrammingMode();
   LED_MERAH_OFF();
   SEI();
```

```
    // Pengiriman data fuse bits sesuai dengan format protokol dengan pemrogram
    tulisExtSRAM(0, 0xFF);          //ID Fuse Bits (Diabaikan Pemrogram)
    tulisExtSRAM(1, '|');
    tulisExtSRAM(2, fuse_bits[0]);  //Low Fuse
    tulisExtSRAM(3, '|');
    tulisExtSRAM(4, fuse_bits[1]);  //High Fuse
    tulisExtSRAM(5, '|');
    tulisExtSRAM(6, fuse_bits[2]);  //Extended Fuse

    //Pengiriman data pembacaan Fusebits ke sisi pemrogram
    jml_byte_data=6;
    jml_paket1 = jml_byte_data;
    jml_paket0 = 0x00;
    kirimPaketDataKePemrogram();
}

void tulisFuseBits(void)
{
    unsigned char data_fuse[7];
    unsigned char informasi_fuse;
    unsigned char i;
    bit status_data_fuse;
    bit status_low_fuse;
    bit status_high_fuse;
    bit status_extended_fuse;

    CLI();
    status_data_fuse= 0;
    status_low_fuse= 0;
    status_high_fuse= 0;
    status_extended_fuse= 0;
    for(i=0;i<7;i++) data_fuse[i]= bacaExtSRAM(i);
    informasi_fuse= data_fuse[0];
    switch(informasi_fuse)
    {
        case 0x00:
            {
              status_low_fuse= 1;
              status_data_fuse= 1;
            }
            break;
        case 0x01:
            {
              status_low_fuse= 1;
              status_high_fuse= 1;
              status_data_fuse= 1;
            }
            break;
        case 0x02:
            {
              status_low_fuse= 1;
              status_high_fuse= 1;
              status_extended_fuse= 1;
              status_data_fuse= 1;
            }
            break;
        default : status_data_fuse= 0;
    }

    if(status_data_fuse)
    {
        LED_MERAH_ON();
        /*** Set SCK dengan frekuensi rendah ***/
        inisialisasiSPILowSpeed();
        enterProgrammingMode();
        if(status_low_fuse) ISPWriteLowFuseBits(data_fuse[2]);
        if(status_high_fuse) ISPWriteHighFuseBits(data_fuse[4]);
        if(status_extended_fuse) ISPWriteExtendedBits(data_fuse[6]);
        leaveProgrammingMode();
        LED_MERAH_OFF();
        kirimKonfirmasiOK();
    }
    SEI();
}
```

```
void hapusMemoriDevais(void)
{
   CLI();
   enterProgrammingMode();
   chipErase();
   leaveProgrammingMode();
   SEI();
}

void menuISP(unsigned char menu_pilihan)
{
        switch(menu_pilihan)
        {
                case BACA_FLASH_MEMORY:
                    bacaMemoriFlash();
                    break;
                case TULIS_FLASH_MEMORY:
                    tulisMemoriFlash();
                    break;
                case BACA_EEPROM:
                    bacaEEPROM();
                    break;
                case TULIS_EEPROM:
                    tulisEEPROM();
                    break;
                case BACA_FUSE_BITS:
                    bacaFuseBits();
                    break;
                case TULIS_FUSE_BITS:
                    tulisFuseBits();
                    break;
                case HAPUS_CHIP:
                    hapusMemoriDevais();
                    break;
        }
}

void menu_multitarget (unsigned char menu_mikrokontroler)
{
      switch(menu_mikrokontroler)
         {
            case mikrokontroler1:
                aktifkan_mikrokontroler_1();
                break;
            case mikrokontroler2:
                aktifkan_mikrokontroler_2();
                break;
            case mikrokontroler3:
                aktifkan_mikrokontroler_3();
                break;
            case mikrokontroler4:
                aktifkan_mikrokontroler_4();
                break;
            default : aktifkan_mikrokontroler_0();
         }
}

void olahMK (void)
{
   bit selesai_terima_pilihan;
   unsigned char data;
   unsigned char buffer_paket_data[5];  //[< 81 > ID_Mikrokontroler | CRC CRC]
   unsigned char indek_data;
   unsigned char i;

   while(!selesai_terima_pilihan)
   {
      data= bacaDataSerialRF();
      buffer_paket_data[indek_data++]= data;
      kirimDataSerial(data);   //Debug log serial USB

      if((indek_data == 1 && data != '<') || (indek_data == 2 && data != 0x81) || (indek_data == 3 && data != '>'))
         indek_data= 0;
      else if(indek_data == 5 && data != '|')
         indek_data= 0;
```

```
        else if (indek_data == 5)
        {
            for(i=0; i<indek_data; i++)
            checksum = checksum^buffer_paket_data[i];

            ID_mikrokontroler=buffer_paket_data[3];
            menu_multitarget(ID_mikrokontroler);

            //Baca Devais Signature untuk dibandingkan dengan kode devais dari pemrogram
            ISPReadDevaisSignature();

            //Cek eksistensi devais yang terpasang pada sisi target
            eksistensi_devais= cekEksistensiDevais();

            if (eksistensi_devais==1)
            {
                LED_BIRU_ON();
                RESET_SLAVE_AKTIF();
                selesai_terima_pilihan= 1;
                kirimKonfirmasiOK();
            }
            else if (eksistensi_devais==0)
            {
                LED_BIRU_OFF();
                RESET_SLAVE_NONAKTIF();
                selesai_terima_pilihan= 1;
                status_kirim=1;
                kirimKonfirmasiGAGAL();
            }

            indek_data=0;
        }
    };
}

void olahPaketPerintah(void)
{
    bit selesai_terima_perintah;
    unsigned char data;
    unsigned char buffer_paket_data[12];  //[< 81 > ID_perintah | jml_paket0 jml_paket1 | device_signature(3 byte)|CRC CRC]
    unsigned char indek_data;
    unsigned char i;

    while(!selesai_terima_perintah)
    {
        data= bacaDataSerialRF();
        buffer_paket_data[indek_data++]= data;
        kirimDataSerial(data);  //Debug log serial USB

        if((indek_data == 1 && data != '<') || (indek_data == 2 && data != 0x81) || (indek_data == 3 && data != '>'))
            indek_data= 0;
        else if(indek_data == 5 && data != '|')
            indek_data= 0;
        else if(indek_data == 8 && data != '|')
            indek_data= 0;
        else if(indek_data == 12 && data != '|')
            indek_data= 0;
        else if(indek_data == 12)  // Deteksi awal kiriman paket
        {
            for(i=0; i<indek_data; i++)
            checksum = checksum^buffer_paket_data[i];

            instruksi_pemrogram= buffer_paket_data[3];
            //Operasi Tulis dan Hapus
            if((instruksi_pemrogram==TULIS_FLASH_MEMORY) || (instruksi_pemrogram==TULIS_EEPROM) ||
(instruksi_pemrogram==TULIS_FUSE_BITS) || (instruksi_pemrogram==HAPUS_CHIP))
            {
                LED_BIRU_OFF();
                LED_MERAH_ON();

                if(instruksi_pemrogram!=HAPUS_CHIP)    // Bukan operasi hapus
                {
                  jml_paket0= buffer_paket_data[5];
                  jml_paket1= buffer_paket_data[6];
                  jml_byte_data = (unsigned int)jml_paket0;
```

```
                    jml_byte_data = (jml_byte_data<<8) + jml_paket1;
                        kirimKonfirmasiOK();  //status_perintah_tulis=1 (Mode tulis MK)  status_perintah_tulis=0 (Mode baca MK)
                    selesai_terima_perintah= 1;  // Status perlu simpan data yang akan ditulis
                    status_perintah_tulis= 1;
                   }
                   else // Operasi hapus tidak perlu data transmisi lagi
                   {

                    LED_BIRU_OFF();
                    LED_MERAH_ON();
                    chipErase();
                    kirimKonfirmasiOK();
                    LED_BIRU_ON();
                    LED_MERAH_OFF();
                    selesai_terima_perintah= 1;
                    status_kirim=0;
                   }
                }

                                // Operasi Baca
             else if ((instruksi_pemrogram==BACA_FLASH_MEMORY) || (instruksi_pemrogram==BACA_EEPROM) ||
(instruksi_pemrogram==BACA_FUSE_BITS))
                {
                    LED_BIRU_OFF();
                    LED_MERAH_ON();

                    kode_devais[0]= buffer_paket_data[8];
                    kode_devais[1]= buffer_paket_data[9];
                    kode_devais[2]= buffer_paket_data[10];
                    selesai_terima_perintah= 1;
                    status_perintah_tulis= 0;

                   if((kode_devais[0]==devais_signature[0]) && (kode_devais[1]==devais_signature[1]) &&
(kode_devais[2]==devais_signature[2]))
                    {
                     ID_perintah= instruksi_pemrogram;
                     ID_mikro = ID_mikrokontroler;
                     kirimKonfirmasiOK();
                     status_kirim=1;
                    }
                    else
                    {
                     kirimKonfirmasiGAGAL();
                     status_kirim=0;
                    }
                }
      indek_data=0;
      }
   };

}

void olahPaketData(void)
{
   bit selesai_terima_data;
   bit terima_data;
   bit status_cek_kirim;
   unsigned char data;
   unsigned char buffer_paket_data[5];   //[< 81 > ID_perintah | 'DATA' ]
   unsigned char indek_data;
   unsigned char c;

   while(!selesai_terima_data)
      {
         data= bacaDataSerialRF();
         buffer_paket_data[indek_data++]= data;
         //kirimDataSerial(indek_data); //Debug log indek_data serial USB

         if((indek_data == 1 && data != '<') || (indek_data == 2 && data != 0x81) || (indek_data == 3 && data != '>'))
            indek_data= 0;
         else if(indek_data == 5 && data != '|')
           indek_data= 0;
         else if(indek_data == 5)  //Penerimaan Data
                    {
                     idxBufExtSRAM=0;
```

```
                while (!terima_data)
                    {
                    data=bacaDataSerialRF();
                    checksum = checksum^data;
                    tulisExtSRAM(idxBufExtSRAM++,data);
                    kirimDataSerial(data);  //Debug log serial USB
                    if (jml_byte_data == idxBufExtSRAM)
                    terima_data= 1;
                    };

                while (!status_cek_kirim)
                    {
                    data=bacaDataSerialRF();
                    kepala[c++]=data;
                    //kirimDataSerial(data);  //Debug log serial USB
                    if (c==2 )
                    {
                    status_cek_kirim=1;
                    if((kepala[0]=='O')&&(kepala[1]=='K'))

                            {
                            kirimKonfirmasiOK();
                            selesai_terima_data=1;
                            status_kirim=1;
                            indek_data=0;
                            }

                    else
                            {
                            kirimKonfirmasiGAGAL();
                            selesai_terima_data=1;
                            status_kirim=0;
                            indek_data=0;
                            }
                    }
                    };
            indek_data=0;
            }
        };
}


void terimaDataDariPemrogram(void)
{
   olahMK();
   while(!status_kirim)             //status_kirim=0 (MK sudah terpilih)  status_kirim=1 (MK belum terpilih)
   {
      olahPaketPerintah();
      if(status_perintah_tulis == 1)            //status_perintah_tulis=1 (Mode tulis MK)  status_perintah_tulis=0 (Mode baca
MK)
      {
            olahPaketData();
      }
   };

}

void main(void)
{
// Declare your local variables here

// Crystal Oscillator division factor: 1
#pragma optsize-
CLKPR=0x80;
CLKPR=0x00;
#ifdef _OPTIMIZE_SIZE_
#pragma optsize+
#endif

// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x00;
```

```
// Port B initialization
// Func7=Out Func6=In Func5=Out Func4=Out Func3=In Func2=In Func1=In Func0=In
// State7=0 State6=T State5=0 State4=0 State3=T State2=T State1=T State0=T
PORTB=0b00000000;
DDRB=0b10110000;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=In
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=T
PORTD=0b00000000;
DDRD=0b11111110;

// Port E initialization
// Func2=In Func1=Out Func0=Out
// State2=T State1=0 State0=0
PORTE=0b00000000;
DDRE=0b00000011;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// Timer/Counter 3 initialization
// Clock value: Timer 3 Stopped
// Mode: Normal top=FFFFh
// Noise Canceler: Off
// Input Capture on Falling Edge
// OC3A output: Discon.
// OC3B output: Discon.
// Timer 3 Overflow Interrupt: Off
```

```
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR3A=0x00;
TCCR3B=0x00;
TCNT3H=0x00;
TCNT3L=0x00;
ICR3H=0x00;
ICR3L=0x00;
OCR3AH=0x00;
OCR3AL=0x00;
OCR3BH=0x00;
OCR3BL=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;
ETIMSK=0x00;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;

inisialisasiUSART();
inisialisasiSPILowSpeed();
inisialisasiSPIHighSpeed();
inisialisasiExtSRAM();
AKTIFKAN_LATCH();

for(;;)
    {
    inisialisasi();
    terimaDataDariPemrogram();
    menuISP(instruksi_pemrogram);
    led_kedip();
    delay_ms(500);
    };
```

isp.h

```
#ifndef __isp_h_included__
#define __isp_h_included__

#include <mega162.h>
#include <spi.h>

#define SPIF 7
#define SCK PORTB.7
#define RESET_SLAVE_AKTIF()     PORTB&=~(1<<4)
#define RESET_SLAVE_NONAKTIF()  PORTB|= (1<<4)

//Deklarasi Prototipe Fungsi Operasi-Operasi SPI-ISP
void inisialisasiSPILowSpeed(void);
void inisialisasiSPIHighSpeed(void);
unsigned char SPIMasterKirimData(unsigned char data);
void enterProgrammingMode(void);
void chipErase(void);
void leaveProgrammingMode(void);
unsigned char cekEksistensiDevais(void);

#endif
```

isp.c

```
#include "isp.h"

void inisialisasiSPILowSpeed(void)
{
      // SPI initialization
      // SPI Type: Master
      // SPI Clock Rate: 86,400 kHz
      // SPI Clock Phase: Cycle Half
      // SPI Clock Polarity: Low
      // SPI Data Order: MSB First
      SPCR=0x53;
```

```
    SPSR=0x00;
}

void inisialisasiSPIHighSpeed(void)
{
    // SPI initialization
    // SPI Type: Master
    // SPI Clock Rate: 2*691.200 kHz
    // SPI Clock Phase: Cycle Half
    // SPI Clock Polarity: Low
    // SPI Data Order: MSB First
    SPCR=0x51;
    SPSR=0x01;
}

unsigned char SPIMasterKirimData(unsigned char data)
{
  //Start Transmission
  SPDR = data;
  //Tunggu sampai SPI Interrupt Flag set
  while(!(SPSR & (1<<SPIF)));
  return SPDR;
}

void enterProgrammingMode(void)
{
  /*** Instruksi Programming Enable pada Serial Programming ***/
  //Reset harus diberi pulsa positif minimal selama 2 siklus clock CPU setelah pin SCK diset 0
  SCK=0;
  RESET_SLAVE_NONAKTIF();
  delay_ms(1);
  RESET_SLAVE_AKTIF();
  //Menunggu waktu tunda minimal selama 20 ms
  delay_ms(20);
  SPIMasterKirimData(0xAC);
  SPIMasterKirimData(0x53);
  SPIMasterKirimData(0x00);
  SPIMasterKirimData(0x00);
}

void chipErase(void)
{
  SPIMasterKirimData(0xAC);
  SPIMasterKirimData(0x80);
  SPIMasterKirimData(0x00);
  SPIMasterKirimData(0x00);
  //Menunggu waktu tunda minimal selama 10 ms
  delay_ms(10);
}

void leaveProgrammingMode(void)
{
  RESET_SLAVE_NONAKTIF();
  //Indikator Leave Programming Mode yaitu matinya LED reset pada MK Target
}
/*****************************************************************************************************
*****/
unsigned char cekEksistensiDevais(void)
{
 unsigned char status_eksistensi_devais;
 unsigned char respon_devais;

 do
 {
    enterProgrammingMode();
    SPIMasterKirimData(0xAC);
    SPIMasterKirimData(0x53);
    respon_devais= SPIMasterKirimData(0x00);
    SPIMasterKirimData(0x00);
    if(respon_devais==0x53)
    {
    status_eksistensi_devais= 1;
    status_MK_pilih=1;
    }
    else
```

```
    {
    status_eksistensi_devais= 1;
    status_MK_pilih=0;
    }
    leaveProgrammingMode();
  }
  while(!status_eksistensi_devais);
  return status_MK_pilih;
}
```

eeprom.h

```
#ifndef __eeprom_h_included__
#define __eeprom_h_included__

#define ULANG_POLLING_EEPROM_MAX    2

//Deklarasi Prototipe Fungsi Operasi-Operasi Memori EEPROM
void ISPWriteEEPROM(unsigned int jml_byte_data);
void ISPReadEEPROM(unsigned int addr_EEPROM_max);
void ISPPollingEEPROM(unsigned int addr_EEPROM_memory);

#endif
```

eeprom.c

```
#include "eeprom.h"
#include "extsram/extsram.h"
#include "isp/isp.h"

void ISPWriteEEPROM(unsigned int jml_byte_data)
{
   //Variabel Tulis Data EEPROM Memory Byte Mode
   unsigned char data;
   unsigned int i;
   unsigned int addr_EEPROM_memory;

   addr_EEPROM_memory= 0;
   i=0;

   while(jml_byte_data>0)
   {
      /** Instruksi Write EEPROM Memory pada Serial Programming **/
      data= bacaExtSRAM(i++);
      if(data!=0xFF)
      {
         SPIMasterKirimData(0xC0);
         SPIMasterKirimData(addr_EEPROM_memory>>8);
         SPIMasterKirimData(addr_EEPROM_memory);
         SPIMasterKirimData(data);

         /*** Polling Data EEPROM Memory ***/
         ISPPollingEEPROM(addr_EEPROM_memory);
      }

      /* Autoincrement alamat untuk writing EEPROM memory dan parameter polling EEPROM memory */
      addr_EEPROM_memory++;

      jml_byte_data--;
   }
}

void ISPReadEEPROM(unsigned int addr_EEPROM_memory_max)
{
   unsigned char data_EEPROM;
   unsigned int addr_EEPROM_memory;
   unsigned int i;

   addr_EEPROM_memory= 0;
   i= 0;

   while(addr_EEPROM_memory<addr_EEPROM_memory_max)
   {
      /*** Instruksi Read EEPROM Memory pada Serial Programming ***/
      //Read EEPROM Memory Low Byte
      SPIMasterKirimData(0xA0);
      SPIMasterKirimData(addr_EEPROM_memory>>8);
```

```
        SPIMasterKirimData(addr_EEPROM_memory);
        data_EEPROM=SPIMasterKirimData(0x00);
        tulisExtSRAM(i++,data_EEPROM);

        /* Autoincrement alamat EEPROM memory */
        addr_EEPROM_memory++;
    };
}

void ISPPollingEEPROM(unsigned int addr_EEPROM_memory)
{
    unsigned char ulang_polling_EEPROM;
    unsigned char data_EEPROM_memory;

    //Reload jumlah pengulangan polling data EEPROM memory
    ulang_polling_EEPROM= ULANG_POLLING_EEPROM_MAX;

    while(ulang_polling_EEPROM>0)
    {
        /*** Instruksi Read EEPROM pada Serial Programming ***/
        SPIMasterKirimData(0xA0);
        SPIMasterKirimData(addr_EEPROM_memory>>8);
        SPIMasterKirimData(addr_EEPROM_memory);
        data_EEPROM_memory=SPIMasterKirimData(0x00);
        if(data_EEPROM_memory==0xFF) delay_ms(9);
        ulang_polling_EEPROM--;
    }
}
```

flash.h

```
#ifndef __flash_h_included__
#define __flash_h_included__

#define ULANG_POLLING_FLASH_MAX     2

//Deklarasi Prototipe Fungsi Operasi-Operasi Memori Flash
void ISPWriteFlashMemory(unsigned int jml_page_data);
void ISPReadFlashMemory(unsigned int addr_flash_max);
void ISPPollingFlashMemory(unsigned int addr_rd_prog_memory);

#endif
```

flash.c

```
#include "flash.h"
#include "extsram/extsram.h"
#include "isp/isp.h"

/***  Operasi Penulisan Memori Flash Page Mode***/
void ISPWriteFlashMemory(unsigned int jml_page_data)
{
    //Variabel Tulis Data Flash Memory
    unsigned int addr_flash_memory_page;
    unsigned char data_low_byte, data_high_byte;
    unsigned int i;

    addr_flash_memory_page=0x00;
    i=0;

    while(jml_page_data>0)
    {
        //Baca data memori dari SRAM Eksternal
        data_low_byte=bacaExtSRAM(i++);
        data_high_byte=bacaExtSRAM(i++);

        /*** Instruksi Load Program Memory Page pada Serial Programming ***/
        //Load Program Memory Low Byte
        SPIMasterKirimData(0x40);
        SPIMasterKirimData((unsigned char)(addr_flash_memory_page>>8));
        SPIMasterKirimData((unsigned char) addr_flash_memory_page);
        SPIMasterKirimData(data_low_byte);  //data low byte

        /*** Instruksi Load Program Memory Page pada Serial Programming ***/
        //Load Program Memory High Byte
```

```
        SPIMasterKirimData(0x48);
        SPIMasterKirimData((unsigned char)(addr_flash_memory_page>>8));
        SPIMasterKirimData((unsigned char) addr_flash_memory_page);
        SPIMasterKirimData(data_high_byte);  //data high byte

        /*** Instruksi Write Program Memory Page pada Serial Programming ***/
        SPIMasterKirimData(0x4C);
        SPIMasterKirimData((unsigned char)(addr_flash_memory_page>>8));
        SPIMasterKirimData((unsigned char) addr_flash_memory_page);
        SPIMasterKirimData(0x00);

        /*** Polling Data Flash Memory ***/
        ISPPollingFlashMemory(addr_flash_memory_page);

        /* Autoincrement alamat untuk writing flash memory dan parameter polling flash memory */
        addr_flash_memory_page++;

        jml_page_data--;
    };
}


/***  Operasi Pembacaan Memori Flash Page Mode ***/
void ISPReadFlashMemory(unsigned int page_flash_max)
{
    unsigned char data_flash;
    unsigned int addr_flash_memory_page;
    unsigned int i;

    data_flash= 0;
    addr_flash_memory_page= 0;
    i= 0;

    while(addr_flash_memory_page<page_flash_max)
    {
        /*** Instruksi Read Program Memory pada Serial Programming ***/
        //Read Program Memory Low Byte
        SPIMasterKirimData(0x20);
        SPIMasterKirimData((unsigned char)(addr_flash_memory_page>>8));
        SPIMasterKirimData((unsigned char) addr_flash_memory_page);
        data_flash= SPIMasterKirimData(0x00);
        tulisExtSRAM(i++,data_flash);
//          data=bacaExtSRAM(i);
//          kirimDataSerial(data);

        //Read Program Memory High Byte
        SPIMasterKirimData(0x28);
        SPIMasterKirimData((unsigned char)(addr_flash_memory_page>>8));
        SPIMasterKirimData((unsigned char) addr_flash_memory_page);
        data_flash= SPIMasterKirimData(0x00);
        tulisExtSRAM(i++,data_flash);
//          data=bacaExtSRAM(i);
//          kirimDataSerial(data);

        /* Autoincrement alamat untuk writing flash memory dan parameter polling flash memory */
        addr_flash_memory_page++;
    };
}


/***  Operasi Polling Memori Flash Byte Mode untuk Menunggu Kesiapan Penulisan Byte Berikutnya***/
void ISPPollingFlashMemory(unsigned int addr_flash_memory_page)
{
   unsigned char ulang_polling_flash;
   unsigned char data_flash_memory;

   //Reload jumlah pengulangan polling data flash memory
   ulang_polling_flash=ULANG_POLLING_FLASH_MAX;

   //Polling Address Low Byte
   while(ulang_polling_flash>0)
   {
      /*** Instruksi Read Program Memory pada Serial Programming ***/
      SPIMasterKirimData(0x20);
      SPIMasterKirimData((unsigned char)(addr_flash_memory_page>>8));
```

```
        SPIMasterKirimData((unsigned char) addr_flash_memory_page);
        data_flash_memory=SPIMasterKirimData(0x00);
        ulang_polling_flash--;
        if(data_flash_memory==0xFF)
        {
            delay_ms(4.5);
            ulang_polling_flash= 0;
        }
    };

    //Reload jumlah pengulangan polling data flash memory
    ulang_polling_flash=ULANG_POLLING_FLASH_MAX;

    //Polling Address High Byte
    while(ulang_polling_flash>0)
    {
        /*** Instruksi Read Program Memory pada Serial Programming ***/
        SPIMasterKirimData(0x28);
        SPIMasterKirimData((unsigned char)(addr_flash_memory_page>>8));
        SPIMasterKirimData((unsigned char) addr_flash_memory_page);
        data_flash_memory=SPIMasterKirimData(0x00);
        ulang_polling_flash--;
        if(data_flash_memory==0xFF)
        {
            delay_ms(4.5);
            ulang_polling_flash= 0;
        }
    };
}
```

fuse.h

```
#ifndef __fuse_bits_h_included__
#define __fuse_bits_h_included__

//Deklarasi Prototipe Fungsi Operasi-Operasi Fuse Bits

/*** Read Signature Byte ***/
void ISPReadDevaisSignature(void);

/*** Fuse Low Byte Operation ***/
unsigned char ISPReadLowFuseBits(void);
void ISPWriteLowFuseBits(unsigned char fuse_low_byte);

/*** Fuse High Byte Operation ***/
unsigned char ISPReadHighFuseBits(void);
void ISPWriteHighFuseBits(unsigned char fuse_high_byte);

/*** Fuse Extended Byte Operation ***/
unsigned char ISPReadExtendedFuseBits(void);
void ISPWriteExtendedBits(unsigned char fuse_extended_byte);

#endif
```

fuse.c

```
#include "fuse.h"
#include "extsram/extsram.h"
#include "isp/isp.h"

/*** Operasi Baca Kode Devais ***/
void ISPReadDevaisSignature(void)
{
    unsigned char i;

    /*** Instruksi Read Signature Byte pada Serial Programming ***/
    //Devais Signature Byte ATmega8535
    //0x00 0x1E
    //0x01 0x93
    //0x02 0x08
    inisialisasiSPILowSpeed();
    enterProgrammingMode();
    for(i=0;i<3;i++)
    {
        SPIMasterKirimData(0x30);
```

```
        SPIMasterKirimData(0x00);
        SPIMasterKirimData(i);
        devais_signature[i]=SPIMasterKirimData(0x00);
    }
    leaveProgrammingMode();
    inisialisasiSPIHighSpeed();
}

/*****************************************************************************************/
//0xD9 -->> unprog clk (int clk), unprog EESAVE, prog BOOTSz1 & BOOTSz1, unprog BOOTRST
//0xC9 -->> prog clk (ext clk), unprog EESAVE, prog BOOTSz1 & BOOTSz1, unprog BOOTRST
//0xE4 -->> programmed clk 8MHz, programmed SUT0
//0xEF -->> unprogrammed clk, programmed SUT0
/*****************************************************************************************/

//************** Operasi Fuse Low Byte **************//
/*
Konfigurasi Fuse Low Byte
Bit 7 -->> BODLEVEL  |  Bit 3 -->> CKSEL3
Bit 6 -->> BODEN     |  Bit 2 -->> CKSEL2
Bit 5 -->> SUT1      |  Bit 1 -->> CKSEL1
Bit 4 -->> SUT0      |  Bit 0 -->> CKSEL1
"0" = programmed
"1" = unprogrammed
*/

/*** Operasi Baca Fuse Low Byte ***/
unsigned char ISPReadLowFuseBits(void)
{
    unsigned char fuse_low_byte;

    /*** Instruksi Read Fuse Low Bits pada Serial Programming ***/
    SPIMasterKirimData(0x50);
    SPIMasterKirimData(0x00);
    SPIMasterKirimData(0x00);
    fuse_low_byte=SPIMasterKirimData(0x00);
    return fuse_low_byte;
}

/*** Operasi Tulis Fuse Low Byte ***/
void ISPWriteLowFuseBits(unsigned char fuse_low_byte)
{
    /*** Instruksi Write Fuse Low Bits pada Serial Programming ***/
    SPIMasterKirimData(0xAC);
    SPIMasterKirimData(0xA0);
    SPIMasterKirimData(0x00);
    SPIMasterKirimData(fuse_low_byte);
    delay_ms(5);
}

//************** Operasi Fuse High Byte **************//
/*
Konfigurasi Fuse High Byte
Bit 7 -->> @@@@@     |  Bit 3 -->> EESAVE
Bit 6 -->> @@@@@     |  Bit 2 -->> BOOTSZ1
Bit 5 -->> SPIEN     |  Bit 1 -->> BOOTSZ0
Bit 4 -->> CKOPT     |  Bit 0 -->> BOOTRST
"0" = programmed
"1" = unprogrammed
Ket: @@@@@ Tergantung Devais
*/

/*** Operasi Baca Fuse High Byte ***/
unsigned char ISPReadHighFuseBits(void)
{
    unsigned char fuse_high_byte;

    /*** Instruksi Read Fuse High Bits pada Serial Programming ***/
    SPIMasterKirimData(0x58);
    SPIMasterKirimData(0x08);
    SPIMasterKirimData(0x00);
    fuse_high_byte=SPIMasterKirimData(0x00);
    return fuse_high_byte;
}
```

```
/*** Operasi Tulis Fuse High Byte ***/
void ISPWriteHighFuseBits(unsigned char fuse_high_byte)
{
   /*** Instruksi Write Fuse High Bits pada Serial Programming ***/
   SPIMasterKirimData(0xAC);
   SPIMasterKirimData(0xA8);
   SPIMasterKirimData(0xFF);
   SPIMasterKirimData(fuse_high_byte);
   delay_ms(5);
}

unsigned char ISPReadExtendedFuseBits(void)
{
   unsigned char extended_fuse_byte;

   /*** Instruksi Read Extended Fuse Byte pada Serial Programming ***/
   SPIMasterKirimData(0x50);
   SPIMasterKirimData(0x08);
   SPIMasterKirimData(0x00);
   extended_fuse_byte= SPIMasterKirimData(0x00);
   return extended_fuse_byte;
}

void ISPWriteExtendedBits(unsigned char fuse_extended_byte)
{
   /*** Instruksi Write Fuse High Bits pada Serial Programming ***/
   SPIMasterKirimData(0xAC);
   SPIMasterKirimData(0xA4);
   SPIMasterKirimData(0xFF);
   SPIMasterKirimData(fuse_extended_byte);
   delay_ms(5);
}
```

usart.h

```
#ifndef __usart_h_included__
#define __usart_h_included__

#include <mega162.h>

//USART Control and Status Register A – UCSRA
#define UPE   2
#define OVR   3
#define FE    4
#define UDRE  5
#define TXC   6
#define RXC   7

#define FRAMING_ERROR       (1<<FE)
#define PARITY_ERROR        (1<<UPE)
#define DATA_OVERRUN        (1<<OVR)
#define DATA_REGISTER_EMPTY  (1<<UDRE)
#define TX_COMPLETE         (1<<TXC)
#define RX_COMPLETE         (1<<RXC)

//USART Control and Status Register B – UCSRB
#define TXB8  0
#define RXB8  1
#define TXEN  3
#define RXEN  4
#define RXCIE 7

#define AKTIFKAN_TRANSMITTER_RF()           UCSR0B|= (1<<TXEN)
#define MATIKAN_TRANSMITTER_RF()        UCSR0B&= ~(1<<TXEN)

#define AKTIFKAN_RECEIVER_RF()                      UCSR0B|= (1<<RXEN)
#define MATIKAN_RECEIVER_RF()                       UCSR0B&= ~(1<<RXEN)

#define AKTIFKAN_RECEIVER_INTERRUPT_RF()   UCSR0B|= (1<<RXCIE)
#define MATIKAN_RECEIVER_INTERRUPT_RF()          UCSR0B&= ~(1<<RXCIE)

#define AKTIFKAN_TRANSMITTER()              UCSR1B|= (1<<TXEN)
#define MATIKAN_TRANSMITTER()                 UCSR1B&= ~(1<<TXEN)

#define AKTIFKAN_RECEIVER()                         UCSR1B|= (1<<RXEN)
```

```
#define MATIKAN_RECEIVER()                      UCSR1B&= ~(1<<RXEN)

#define AKTIFKAN_RECEIVER_INTERRUPT()           UCSR1B|= (1<<RXCIE)
#define MATIKAN_RECEIVER_INTERRUPT()            UCSR1B&= ~(1<<RXCIE)

void inisialisasiUSART(void);
//Pengiriman data serial menggunakan RF
void kirimDataSerialRF(unsigned char data);
//Pengiriman data serial untuk debugging program
void kirimDataSerial(unsigned char data);

#endif
```

usart.c

```
#include "usart.h"

void inisialisasiUSART()
{

      // USART0 initialization
      // Communication Parameters: 8 Data, 1 Stop, No Parity
      // USART0 Receiver: On
      // USART0 Transmitter: On
      // USART0 Mode: Asynchronous
      // USART0 Baud Rate: 9600
      UCSR0A=0x00;
      UCSR0B=0x98;
      UCSR0C=0x86;
      UBRR0H=0x00;
      UBRR0L=0x47;

      // USART1 initialization
      // Communication Parameters: 8 Data, 1 Stop, No Parity
      // USART1 Receiver: Off
      // USART1 Transmitter: On
      // USART1 Mode: Asynchronous
      // USART1 Baud Rate: 9600
      UCSR1A=0x00;
      UCSR1B=0x08;
      UCSR1C=0x86;
      UBRR1H=0x00;
      UBRR1L=0x47;
}

void kirimDataSerialRF(unsigned char data)
{
   //Wait for empty transmit buffer
   while (!(UCSR0A & (1<<UDRE)));
   UDR0=data;
}

//Pengiriman data serial untuk debugging program
void kirimDataSerial(unsigned char data)
{
   //Wait for empty transmit buffer
   while (!(UCSR1A & (1<<UDRE)));
   UDR1=data;
}
```

extsram.h

```
#ifndef __extsram_h_included__
#define __extsram_h_included__

#define OFFSET_ADDR_EXT_SRAM    0x0500    //Alamat awal buffer Ext SRAM
#define ADDR_EXT_SRAM_MAX       0x7FFF    //Ext SRAM 32 kbyte

void inisialisasiExtSRAM(void);
unsigned char bacaExtSRAM(unsigned int addr_read_ext_SRAM);
void tulisExtSRAM(unsigned int addr_write_ext_SRAM, unsigned char data);

#endif
```

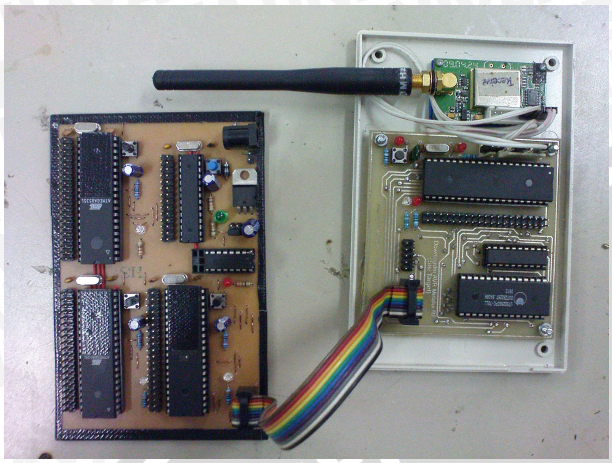
extsram.c

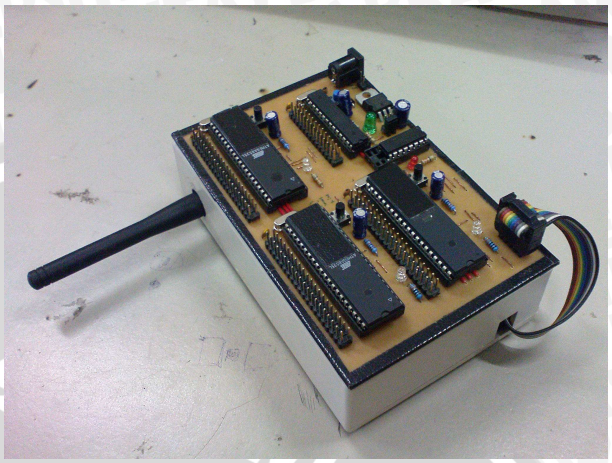
```
#include "extsram.h"

/*****************************************************************************/
// Fungsi Inisialisasi SRAM Eksternal
void inisialisasiExtSRAM(void)
{
   // External Interrupt(s) initialization
   // INT0: Off
   // INT1: Off
   // INT2: Off
   // Interrupt on any change on pins PCINT0-7: Off
   // Interrupt on any change on pins PCINT8-15: Off
   // External SRAM page configuration:
   //             -             / 0000h - 7FFFh
   // Lower page wait state(s): None
   // Upper page wait state(s): None
   MCUCR=0x80;
   EMCUCR=0x00;
}
/*****************************************************************************/
// Fungsi Pembacaan SRAM Eksternal
unsigned char bacaExtSRAM(unsigned int addr_read_ext_SRAM)
{
   unsigned char *ptr_data = (unsigned char *) (addr_read_ext_SRAM + OFFSET_ADDR_EXT_SRAM);
   unsigned char data;
   DDRC = 0xFF;
   PORTC = 0x00;
   data= *ptr_data;
   return data;
}
/*****************************************************************************/
// Fungsi Penulisan SRAM Eksternal
void tulisExtSRAM(unsigned int addr_write_ext_SRAM, unsigned char data)
{
   unsigned char *ptr_data = (unsigned char *) (addr_write_ext_SRAM +    OFFSET_ADDR_EXT_SRAM);
   DDRC = 0xFF;
   PORTC = 0x00;
   *ptr_data= data;
}
```

Lampiran III

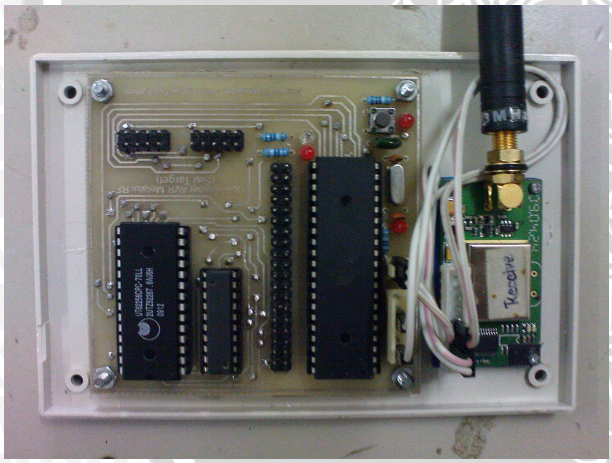
Foto Alat

Sistem Minimum Keseluruhan Sisi Target

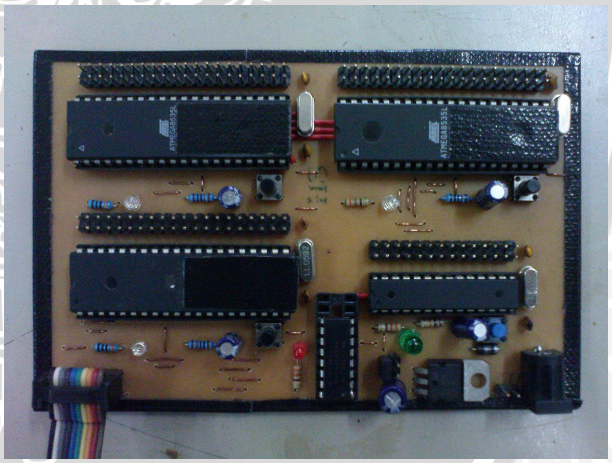




Sistem Minimum Sisi Target



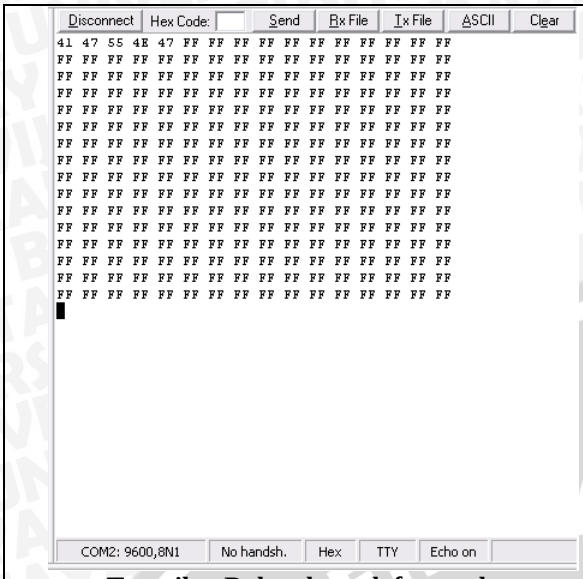
Sistem Mikrokontroler Target



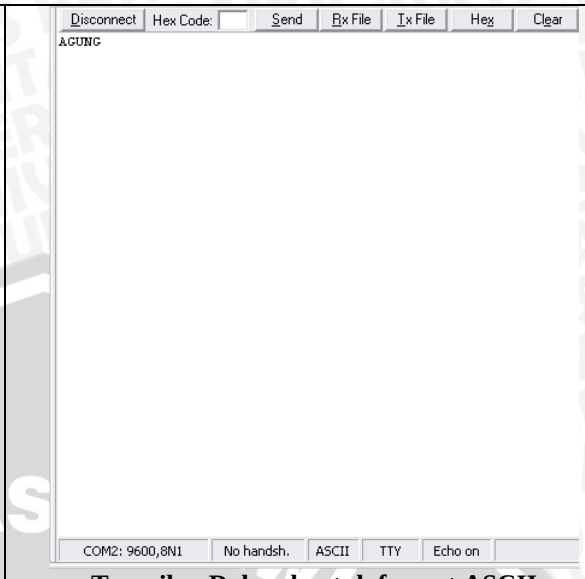
Lampiran IV

Uji Coba Jarak Jangkau

Tampilan Uji Coba jarak 1 meter

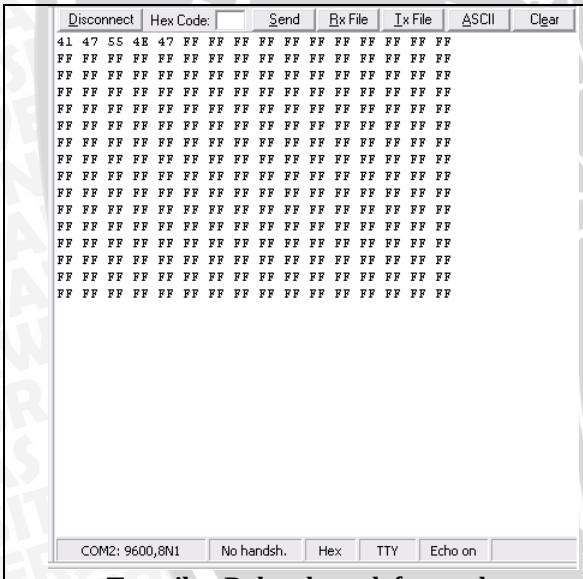


Tampilan Dalam bentuk format hex

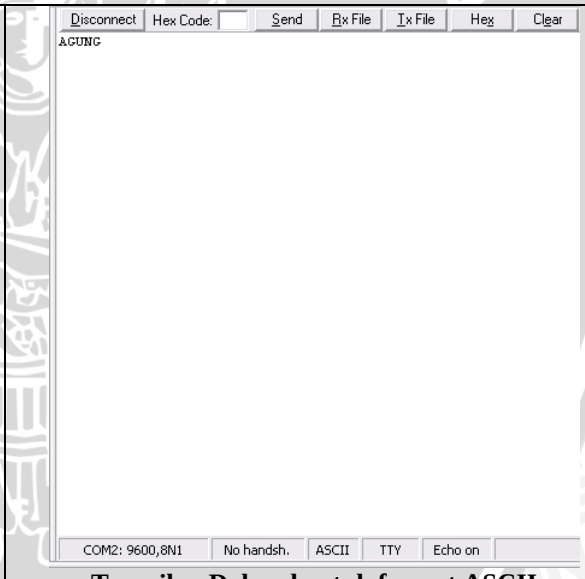


Tampilan Dalam bentuk format ASCII

Tampilan Uji Coba jarak 15 meter

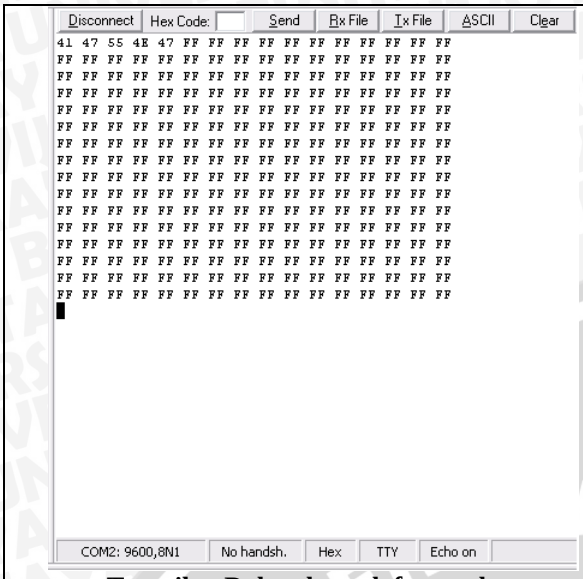


Tampilan Dalam bentuk format hex

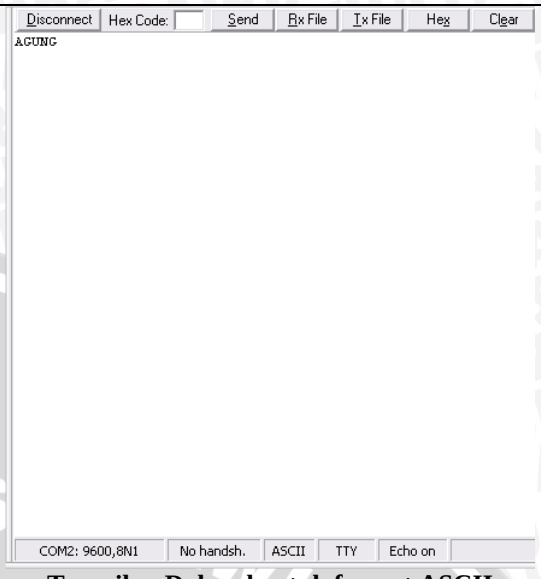


Tampilan Dalam bentuk format ASCII

Tampilan Uji Coba jarak 30 meter



Tampilan Dalam bentuk format hex



Tampilan Dalam bentuk format ASCII

Lampiran V

Datasheet