PERANGKAT KOMUNIKASI MULTI-EXTERNAL HARDWARE MELALUI LAN DENGAN MENGGUNAKAN MICROCONTROLLER

Marojahan M.T. Sigiro[1)]
[1)]*Teknik Komputer, Politeknik Informatika Del*
Jl. Sisingamangaraja, Sitoluama, Laguboti, Tobasa, 22381
email: marojahan@del.ac.id[1)]

Abstrak

Perangkat komunikasi yang dibuat merupakan sebuah perangkat elektronik dengan menggunakan microcontroller yang dapat digunakan sebagai interface antara external-hardware dan LAN sehingga external-hardware tersebut bisa dikontrol atau dimonitor secara remote. External-hardware yang dimaksud berupa perangkat elektronik yang memiliki serial port sebagai interface komunikasi misalnya printer, barcode-reader, switch, hub, dan berbagai perangkat lainnya. Agar external-hardware dapat dimonitor dan dikontrol secara remote melalui LAN maka perangkat yang dibuat ini mengimplementasikan protokol TCP/IP. Selain itu dibuat juga device driver agar perangkat tersebut bisa berkomunikasi dengan komputer yang mengaksesnya sesuai dengan protokol yang telah dispesifikasikan. Perangkat yang dibuat bisa digunakan sebagai interface ke beberapa external-hardware secara bersamaan dengan menyediakan beberapa port koneksi berupa serial port.

Keberadaan perangkat ini telah mengurangi kelemahan-kelemahan penggunaan PC sebagai interface komunikasi dengan external hardware. Penggunaan perangkat ini dapat memberi berbagai keuntungan seperti: menekan harga perangkat, mengurangi penggunaan sumber daya listrik, serta mengefektifkan penggunaan ruang.

Kata kunci:
Remote serial port, komunikasi SPI multi-slave, bidirectional SPI, linux pseudo TTY driver, implementasi TCP/IP pada microcontroller

1. Pendahuluan

A. Latar Belakang

Teknologi perangkat keras (*hardware*) saat ini berkembang dengan cepat. Perangkat keras yang digunakan saat ini dapat berupa perangkat *portable* maupun perangkat *non-portable*. Salah satu kecenderungan perangkat yang ada sekarang ini adalah perangkat yang dapat terhubung ke jaringan komputer, ke *internet* maupun saling terhubung ke perangkat lainnya dengan koneksi *peer-to-peer*.

Untuk menghubungkan suatu perangkat ke jaringan komputer atau *internet*, kebanyakan praktisi menggunakan PC sebagai *interface* antara perangkat yang digunakan (*external-hardware*) dengan jaringan komputer atau *internet*. Untuk keperluan tersebut maka harus disediakan PC sebagai *interface* ke *external-hardware* yang biasanya menggunakan *parallel port, serial port, USB, Blutooth* dan lain-lain.

Penggunaan PC sebagai *interface* akan memiliki beberapa kerugian, antara lain:

1. Harga PC yang relatif mahal
2. Ukuran PC yang relatif besar
3. PC membutuhkan daya listrik yang relatif besar
4. Banyak sumber daya PC yang tidak terpakai
5. Koneksi PC yang terbatas yang pada umumnya hanya bisa melakukan koneksi *one-to-one*

Dengan beberapa pertimbangan tersebut, maka didapat ide untuk menggantikan PC dengan perangkat yang menggunakan *microcontroller* yang nantinya diharapkan dapat memberi keuntungan berupa:

1. Harga jauh lebih murah
2. Ukuran yang kecil sehingga dapat diletakkan dimana saja
3. Konsumsi daya sedikit yang bisa di-*suplay* dengan menggunakan *battery*
4. Mampu menyediakan koneksi antara beberapa *external-harware* ke satu jaringan atau komputer *(many-to- one)*

dengan keuntungan yang dimiliki maka penggunaan perangkat tersebut akan jauh lebih menguntungkan.

B. Perumusan Masalah

Berdasarkan uraian yang telah dijelaskan pada latar belakang, maka masalah yang akan diselesaikan adalah bagaimana membuat suatu perangkat microcontroller yang berperan sebagai interface untuk menghubungkan beberapa external-hardware secara sekaligus dengan LAN. Penggunaan perangkat akan memungkinkan beberapa external-hardware bisa dikontrol dari satu buah komputer dalam LAN melalui jaringan TCP/IP. External-hardware berupa perangkat elektronik seperti switch, router, barcode reader, AC, dan lain-lain yang mendukung koneksi dengan menggunakan serial port atau RS232.

Untuk melakukan hal tersebut, diperlukan program yang dapat menjalankan fungsi protokol

TCP/IP, fungsi I/O untuk *external-hardware* dan *ethernet card* pada *microcontroller*, dan fungsi untuk mengatur komunikasi antara beberapa *external-hardware* dengan LAN pada *microcontroller*. Perangkat keras yang dibutuhkan antara lain *ethernet card* sebagai penghubung dengan LAN, sebuah *microcontroler* utama yang berperan sebagai pengatur komunikasi data antara *external-hardware* dengan *ethernet card*, beberapa *microcontroller* yang berperan sebagai penghubung *external-hardware* melalui *serial port* dengan *microcontroller* utama. Program yang dibuat akan di-*burn* kedalam ROM *microcontroller* sehingga keseluruhan perangkat dapat bekerja sebagai pengganti PC.

Untuk melakukan pengaksesan terhadap *external-hardware* melalui perangkat *microcontroller*, akan dilakukan dengan cara melakukan pengaksesan terhadap *virtual serial port* pada komputer dalam LAN yang berkoresponden dengan *serial port* fisik pada perangkat *microcontroller*. Untuk itu maka akan dilakukan pembuatan program berupa *device-driver* yang akan berperan sebagai *virtual serial port*. *Device-driver* tersebut akan di-*install* di komputer yang akan digunakan untuk mengakses *external-hardware* yang akan menyediakan beberapa *virtual serial port* yang sesuai dengan perangkat *microcontroller* yang digunakan.

2. Tinjauan Pustaka

A. Microcontroller

Microcontroller adalah sebuah perangkat yang merupakan gabungan dari beberapa komponen *microprocessor* yang menghasilkan sebuah *microchip* [2]. *Microcontroller* dapat disebut sebagai *one-chip solution*. *Microcontroller* biasanya terdiri dari [1]:

1. CPU (*Central Processing Unit*)
2. RAM (*Random Access Memory*)
3. EPROM/PROM/ROM (*Erasable Programable Read Only Memory*)
4. I/O (*Input/Output*) berupa *serial* atau *parallel*
5. *Timers*
6. *Interrupt controller*

Biaya yang dibutuhkan dalam pembuatan *microcontroller* dapat ditekan dengan membuat *microcontroller* yang hanya memiliki komponen-komponen tertentu yang diinginkan.

B. Fitur I/O Port

Untuk berkomunikasi dengan perangkat lain, *microcontroller* memiliki port yang digunakan sebagai media I/O. *Port* I/O tersebut dibuat sesuai dengan bentuk komunikasi yang diinginkan. Beberapa jenis I/O port yang dimiliki oleh *microcontroller* antara lain [1]:

1. *Parallel Port*
2. UART (*Universal Asynchronous*
3. USART (*Universal Synchronous/Asynchronous Receiver Transceiver*)
4. *Synchronous Serial Port*
5. SPI (*Serial Peripheral Interface*)
6. SCI (*Serial Communication Interface*)
7. I2C *bus* (*Inter-Integrated Circuit*)
8. MICROWIRE / PLUS
9. CAN (*Controller Area Network*)
10. *Analog to Digital Conversion* (A/D)
11. D/A (*Digital to Analog*) *Converter*

C. Komunikasi data menggunakan serial port

Serial Port adalah *interface* yang digunakan sebagai media komunikasi untuk mentransmisikan data satu *bit* setiap waktu [3]. *Serial* port mengirim dan menerima data sebanyak 1 *bit* setiap satuan waktu melalui sebuah kabel, sehingga diperlukan 8 satuan waktu untuk mengirim satu *byte* data. Untuk mendukung pengiriman data *serial* secara *full-duplex* hanya membutuhkan 3 buah kabel secara terpisah yaitu satu kabel untuk mengirim, satu kabel untuk menerima dan satu kabel untuk *ground*. *Full-duplex* berarti pengiriman dan penerimaan data dapat dilakukan secara bersamaan. Untuk membedakan setiap satuan data maka setiap pengiriman satuan data diawali dengan pengiriman satu *start-bit* dan diakhiri dengan 2 *stop-bit*.

Satu satuan data dapat terdiri dari satu sampai 8 *bit* [4]. Seperti halnya pada setiap komunikasi data, pada komunikasi data *serial* juga terdapat adanya kesalahan pengiriman data. Untuk mengurangi kesalahan pengiriman data tersebut digunakan parity-*bit*. *Parity-bit* diperoleh dengan menghitung satuan data yang akan dikirim sehingga dengan menghitung kembali *parity-bit* pihak penerima data dapat memperkirakan data yang diterima rusak atau tidak. Beberapa macam *parity* yang ada dalam proses pengiriman data [4],

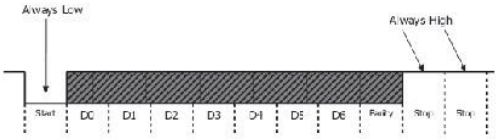
1. *Even parity*, bernilai 0 jika *bit* 1 yang dikirim berjumlah genap dan bernilai 1 jika *bit* 1 yang dikirim berjumlah ganjil.
2. *Odd parity*, bernilai 0 jika *bit* 1 yang dikirim berjumlah ganjil dan bernilai 1 jika *bit* 1 yang dikirim berjumlah genap.
3. *Mark parity*
4. *Space parity*

Ada dua jenis pengiriman data secara *serial* yaitu [4]:

1. *Synchronous*, sisi pengirim dan penerima data melakukan sinkronisasi dalam pengiriman dan penerimaan datanya. Dengan adanya sinkronisasi ini maka *start-bit* dan *stop-bit* tidak diperlukan dan dapat digunakan sebagai data sehingga secara keseluruhan pengiriman data dapat dilakukan lebih cepat. Meskipun pengiriman datanya bisa dilakukan lebih cepat, tetapi dibutuhkan mekanisme sinkronisasi antara sisi pengirim dan penerima

sehingga sistem tersebut relative lebih rumit dan dibutuhkan lebih dari 3 kabel yang digunakan dalam pengiriman sinyal.

2. *Asynchronous*, sisi pengirim dan penerima ridak perlu melakukan sinkronisasi, sehingga diperlukan mekanisme untuk membedakan seriap satuan data yang dikirim yaitu dengan menggunakan *start-bit* dan *stop-bit*. Karena tidak memerlukan mekanisme sinkronisasi maka pengiriman data secara *asynchronous* lebih sederhana. Sistem ini dikenal sebagai UART (*Universal Asynchronous Receiver Transmitter*). Untuk komunikasi secara *asynchronous*, *frame* data yang dipertukarkan dapat dilihat dalam Gambar 1.



Gambar 1 Format bit pengiriman byte data melalui serial port

D. Komunikasi data dengan fitur SPI

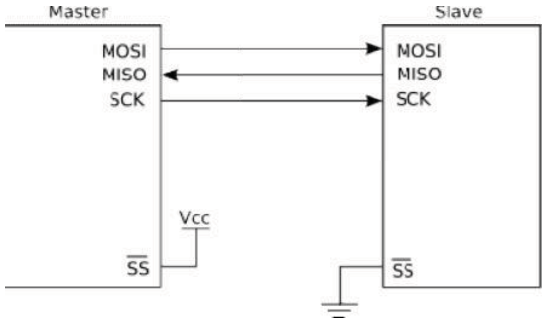
SPI (*Serial Peripheral Interface*) adalah sebuah *interface* komunikasi *serial* yang dapat digunakan untuk membangun sebuah jaringan *microcontroller* antara sebuah *master* dan satu atau lebih *slave*[5]. SPI bisa dibuat sebagai jalur komunikasi satu arah maupun dua arah. Selain sebagai media komunikasi antar *microcontroller*, SPI juga bisa digunakan sebagai media komunikasi dengan perangkat lain seperti sensor, memori, dan perangkat lainnya [5].

Signal yang digunakan oleh SPI dalam membentuk komunikasi dan kegunaannya dalam setiap node adalah [5]:

Tabel 1 Signal Komunikasi SPI

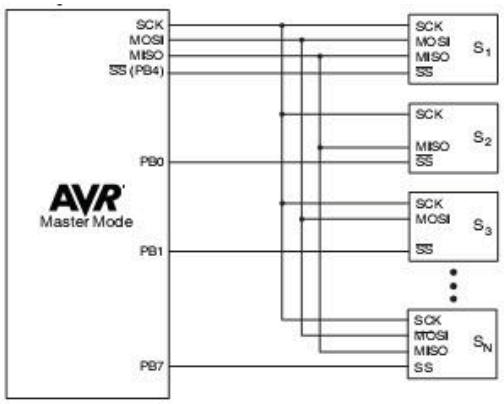
Signal	Kegunaan	Master	Slave
MOSI	*Data line, Master Out Slave In*	*Output*	*Input*
MISO	*Data line, Master Input Slave Output*	*Input*	*Output*
SCK	*Clock line, Serial Clock*	*Output*	*Input*
SS	*Control line, Slave Select*	*Output*	*Input*

Koneksi yang menghubungkan *master* dan *slave* bisa dibentuk menggunakan 3 pin yang saling terhubung untuk membentuk koneksi *single master-single slave* atau menggunakan 4 pin jika akan dibentuk koneksi single *master-multi slave*. *Signal/pin* SS harus di *de-asserted* atau bernilali *logical high* pada *master*. Pada *slave*, SS digunakan sebagai fungsi *chip select* jika pin tersebut di-*asserted* atau bernilai *logical 0* atau *low* yang berarti *slave* akan diaktifkan sebagai *slave* dan dapat bekomunikasi dengan *master*. Sebaliknya *slave* akan di-nonaktifkan jika SS bernilai *high* (*de-asserted*).



Gambar 2 *Wiring* koneksi *single master-single slave* [6]

Diagram diatas menggambarkan koneksi antara sebuah *master* dengan sebuah *slave*. SS pada *master* akan selalu bernilai *high* dan akan bernilai *low* pada *slave*. Untuk membentuk koneksi *single master-multi slave* maka SS pin harus dihubungkan antar *master* dan *slave* dan melakukan mekanisme *chip-select* pada *level software*.



Gambar 3 *Wiring* koneksi single *master*-multi *slave* [6]

Diagram pada gambar 3 menggambarkan koneksi antara satu *master* dan beberapa *slave* dengan menggunakan *microcontroller* AVR.

E. Device Driver pada system operasi Linux

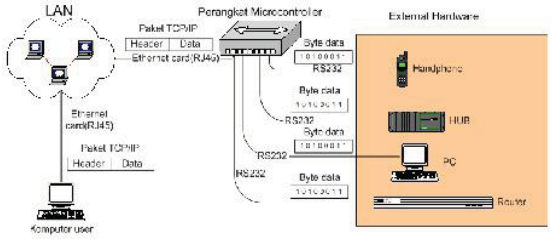
Device driver merupakan sebuah *interface* antara *user* dengan sebuah perangkat fisik maupun perangkat *virtual*[7]. *Device driver* memungkinkan *user* dapat melakukan pengaksesan terhadap perangkat melalui perantaraan *device driver*. *Device driver* menyederhanakan prosedur pengaksesan *device* maupun *resource* dalam *system* dengan memberikan antarmukan antara sistem yang kompleks dengan *user* [7]. *Device driver* menyediakan mekanisme pengaksesan *device* sehingga *user* tidak direpotkan dengan prosedur pengaksesan *device* secara fisik.

Dengan demikian maka *device driver* menentukan bagaimana perilaku dan kemampuan *device* yang sebenarnya. Dari penjelasan diatas perlu diperhatikan bahwa *device driver* menyediakan *mechanism*, bukan menyediakan *policy*. Hal ini berarti bahwa *device driver* menyediakan kemampuan-kemampuan yang dimiliki oleh *device* yang sesungguhnya (*mechanism*) dan bukan menyediakan cara pemanfaatan kemampuan-kemampuan *device* yang sesungguhnya (*policy*). Pemrograman *device driver* memiliki karaketristik yang berbeda jika dibandingkan dengan pembuatan aplikasi biasa. Hal itu disebabkan *device driver* berjalan pada level kernel (*kernel space*) dan aplikasi *user* berjalan pada level *user* (*user space*). Oleh karena itu *device driver* harus menggunakan semua resource yang ada pada level kernel.

3. Metode Penelitian

A. Gambaran Umum

Perangkat yang dibangun merupakan sebuah perangkat jaringan yang berfungsi sebagai *interface* antara *external hardware* dengan LAN. Sebagai sebuah *interface*, sistem yang dibangun akan memungkinkan *user* bisa mengakses *external hardware* yang menggunakan *serial port* dari sebuah komputer yang terdapat dalam LAN dimana sistem berada. Keberadaan sistem dalam LAN akan dianggap sebagai sebuah data terminal. Sistem akan memiliki alamat IP sebagaimana perangkat lain yang berada dalam jaringan. Sistem memiliki dua jenis port koneksi yang memiliki fungsi dan karakteristik yang berbeda yaitu port serial yang berfungsi sebagai port koneksi bagi *external-hardware* dan port *ethernet* (RJ45) yang berfungsi sebagai port koneksi sistem ke LAN. Karena sistem akan diakses dengan menggunakan jalur TCP/IP maka data yang dikirim/diterima dari sistem merupakan paket TCP/IP. Selanjutnya data yang diterima dari LAN akan diteruskan oleh sistem ke *external hardware* dalam bentuk binary data. Secara umum, keterhubungan sistem dengan komputer dalam LAN dan *external-hardware* dapat dilihat dalam Gambar 4.



Gambar 4 Gambaran Umum Sistem

Metode pengaksesan sistem dari sebuah komputer dalam LAN akan dilakukan dengan menggunakan *virtual device driver*. *User* akan melakukan operasi baca/tulis ke *device driver* dalam komputer untuk mengirim dan menerima data dari *external hardware* dengan perantaraan sistem yang dibangun. Dari penjelasan diatas dapat dilihat bahwa sistem terdiri tiga komponen utama berupa *hardware*, *software* yang beroperasi dalam *hardware* dan *software* dalam komputer *User*. Selain ketiga komponen tersebut, sistem juga harus mengimplementasikan protokol TCP/IP sebagai protokol komunikasi dalam LAN.

B. Arsitektur Hardware

Hardware yang dibuat akan terhubung ke LAN dengan menggunakan sebuah *ethernet card*. Selain itu *hardware* juga akan terhubung ke beberapa *external hardware* dengan menggunakan serial port. Hal ini berarti sistem akan melakukan pemrosesan data yang diterima dari LAN untuk menentukan serial port yang merupakan tujuan dari data tersebut. Untuk melakukan hal tersebut maka sistem membutuhkan satu buah *microcontroller* untuk menangani setiap serial port dan satu buah untuk menangani *ethernet card* dan sebagai *microcontroller* utama.

Supaya data yang diproses dapat di teruskan ke *external-hardware* maka beberapa *microcontroller* akan berperan sebagai perantara. *Microcontroller-microcontroller* tersebut akan berperan untuk menerima data dari *microcontroller* utama dan mengirimkannya ke *external hardware* melalui serial port dan sebaliknya.

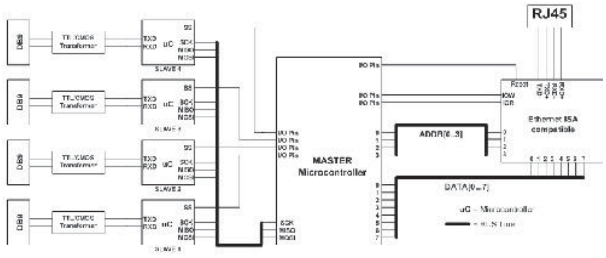
Hardware dari sistem terdiri 5 buah *microcontroller*, 4 buah serial port dan satu buah *ethernet card*. Sebuah *microcontroller* akan berperan sebagai *microcontroller* utama yang disebut dengan *master* dan 4 buah *microcontroller* lainnya sebagai *slave*. Komunikasi yang terjadi antar *microcontroller* yaitu komunikasi antara *master* dan *slave* dengan menggunakan perangkat komunikasi SPI. Jalur komunikasi ini akan digunakan untuk mengirimkan data antar *microcontroller master* dan *slave*.

Microcontroller master akan terhubung dengan *ethernet card* dengan menggunakan *slot* ISA. *Master* akan terhubung ke *ethernet card* dengan menggunakan 3 jenis koneksi yaitu koneksi data, koneksi *address*, koneksi *control*. *Ethernet* yang digunakan harus sesuai dengan ketersediaan *resource* sistem yaitu dapat bekerja dengan power supply sebesar 5 Volt, memiliki kecepatan yang dapat disesuaikan dengan kecepatan *microcontroller* dan dapat bekerja dalam mode 8 bit karena *microcontroller* yang digunakan bekerja dalam mode 8 bit. Untuk itu maka *ethernet* yang digunakan adalah *ethernet 10BaseT* yang bekerja dengan kecepatan 20Mhz, dapat beroperasi dengan voltase 5 Volt dan memiliki kecepatan transfer data sebesar 10Mbps.

Supaya sistem dapat terhubung dengan *external hardware* maka sistem menyediakan 4 buah port serial atau DB9. Masing-masing port serial akan terhubung ke sebuah *microcontroller slave*. *Microcontroller* yang digunakan sudah memiliki fitur built-in serial port yang disebut dengan UART sehingga *microcontroller* tidak memerlukan perangkat

komunikasi serial. Walaupun *slave* telah memiliki port serial tetapi *slave* tidak dapat digunakan sebagai *interface* serial port secara langsung karena *microcontroller* memiliki mode sinya TTL sedangkan mode sinyal serial port berupa CMOS. Pada CMOS logika 1 dilambangkan dengan voltase 12 Volt dan logika 0 dilambangkan dengan voltase -12 Volt sedangkan pada mode sinyal TTL logika 1 dilambangkan dengan voltase 5 Volt dan logika 0 dilambangkan dengan voltase -5 Volt. Oleh karena itu dibutuhkan *transformer* untuk mengubah sinyal TTL menjadi CMOS dan sebaliknya.

Secara keseluruhan maka arsitektur *hardware* dapat dilihat dalam Gambar 5.



Gambar 5 Arsitektur *Hardware*

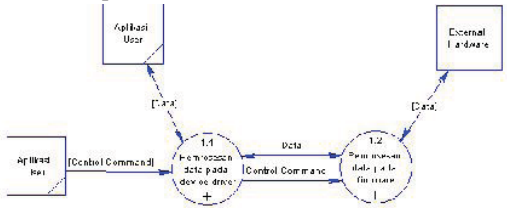
C. Arsitektur Software

Software dari sistem yang dikembangkan terdiri atas dua bagian besar yaitu *firmware* dan *device driver*. *Firmware* berjalan didalam *microcontroller* yang terdapat dalam *hardware* sistem. Sedangkan *device driver* berjalan pada komputer *user* yang berperan sebagai *interface* antara *user* dengan keseluruhan sistem. *Device driver* membuat semua mekanisme pengiriman data menjadi transparan bagi *user*.

Context diagram yang menggambarkan hubungan keseluruhan sistem dengan perangkat lainnya dapat dilihat dalam Gambar 6.

Gambar 6 Context Diagram Sistem (DFD Level 0)

Keseluruhan sistem terbagi atas dua proses utama yang berjalan secara terpisah. Proses-proses tersebut saling terhubung hanya dalam proses pengiriman dan penerimaan data. Pembagian sistem menjadi dua buah proses utama dapat dilihat dalam Gambar 7.



Gambar 7 Pembagian proses sistem (DFD Level 1)

i. Firmware

Firmware merupakan *software* yang di-*burn* kedalam *microcontroller* yang berfungsi untuk menjalankan keseluruhan sistem. *Firmware* dibutuhkan oleh *microcontroller* untuk menjalankan fungsi *logic* dari *microcontroller*. *Firmware* yang dibangun terdiri atas dua bagian besar yaitu *master* dan *slave*. *Firmware master* merupakan *firmware* yang berjalan dalam *microcontroller master* dan *Firmware slave* merupakan *firmware* yang berjalan pada *microcontroller slave*. Masing-masing *firmware* memiliki fungsi dan bagian-bagian yang berbeda yang berbeda.

Firmware master memiliki tugas dan tanggungjawab yang lebih besar karena *firmware master* akan berperan sebagai *communication manager* antara *Ethernet* dan *slave*. Sebagai *communication manager* maka fungsi-fungsi yang harus dimiliki oleh *firmware master* adalah:

1. Melakukan pengontrolan terhadap perangkat *Ethernet*.
2. Melakukan pemrosesan data yang diterima dari *ethernet* maupun dari *microcontroller slave*.
3. Bertanggung-jawab untuk menentukan port tujuan data yang diterima dari *Ethernet* maupun dari *microcontroller slave*.
4. Mengatur komunikasi antara *master* dan *slave*.
5. Bertanggungjawab untuk menjamin keberhasilan pengiriman data dari extenal *device* ke *ethernet* dan sebaliknya.

Untuk melanjutkan pengiriman data dari *firmware master* ke *external device* melalui *serial port*, *firmware slave* harus melakukan penerimaan data dari *master* dengan menggunakan perangkat SPI dan mengirimkannya ke *external device* melalui port serial dan sebaliknya. Dalam komunikasi dengan menggunakan SPI, *slave* tidak memiliki kemampuan untuk melakukan pengiriman data secara *independent*. *Slave* harus mengikuti prosedur pengiriman yang dilakukan oleh *master*. *Slave* dapat mengirim data ke *master* jika *master* melakukan pengiriman data ke *slave*. Untuk itu *slave* harus memiliki komponen berupa data manager yang berfungsi untuk menyediakan *buffer* penampung data yang datang dari *external device* selama *master* tidak menjalin komunikasi dengan *slave* dan melakukan pengiriman data ke *master*. Untuk itu diperlukan ukuran *buffer* yang sesuai sehingga tidak terjadi kesalahan berupa *buffer overflow* yang akan mempengaruhi validitas data.

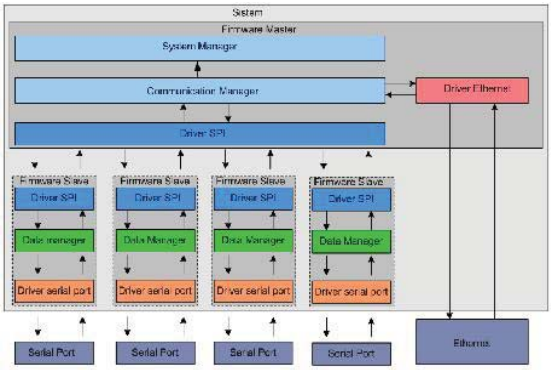
Secara garis besar maka *firmware* dari sistem memiliki beberapa bagian utama yang dibagi berdasarkan fungsinya yaitu:

1. *Communication manager*, berupa prosedur yang berfungsi untuk melakukan manajemen komunikasi dan pengolahan data dan terdapat dalam *firmware master*.
2. *Data manager*, berupa sebuah prosedur dalam *firmware slave* yang berfungsi berfungsi

untuk melanjutkan pengiriman data dari *master* ke *external device* melalui port serial.

3. *Driver SPI,* berupa kumpulan prosedur-prosedur yang terdapat dalam *firmware master* dan *slave* yang menyediakan operasi baca tulis terhadap perangkat SPI pada *microcontroller*.
4. *Driver Ethernet,* berupa kumpulan prosedur-prosedur yang terdapat dalam *firmware master* yang menyediakan operasi baca tulis dan pengontrolan perangkat *ethernet*.
5. *Driver serial port,* berupa kumpulan prosedur-prosedur yang terdapat dalam *firmware slave* yang menyediakan operasi baca tulis terhadap *hardware* serial port.
6. *Sistem Manager*, berupa sebuah posedur dalam *firmware master* yang berfungsi untuk melakukan pengkonfigurasian terhadap sistem berdasarkan perintah yang diterima dari *user*.

Skema keterhubungan antara komponen-komponen *firmware* dari sistem dapat dilihat dalam Gambar 8.



Gambar 8 Skema komponen *firmware* sistem

ii. Device Driver

Device driver merupakan *software* yang berjalan dalam komputer *user*. *Device driver* berperan sebagai *interface* antara aplikasi *user* dan sistem fisik. Dengan adanya *device driver* maka akan memungkinkan *user* untuk mengakses sistem dengan lebih mudah tanpa melakukan pengaksesan secara langsung terhadap perangkat fisik. Dengan mengakses *device driver* maka *user* seolah-olah mengakses sistem yang berada pada komputer *user* secara lokal. Dalam pembuatan *device driver*, perlu diperhatikan bahwa program akan berjalan pada *kernel space*. Untuk itu kernel harus mengunakan semua resource yang ada dalam kernel. Untuk berhubungan dengan *user*, *device driver* akan menggunakan sebuah node yang disediakan oleh file sistem yang berupa *device* serial.

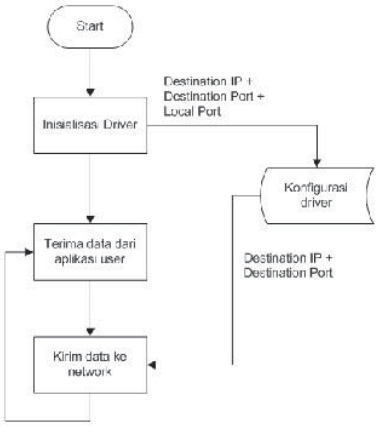
Secara garis besar, *device driver* yang akan bangun terbagi atas dua bagian yaitu:

1. Utilitas *Serial character*
   Bagian ini merupakan bagian yang tampak bagi *user*. Bagian ini akan melakukan penerimaan dan pengiriman data ke *user* melalui *interface* serial *device*. Bagian ini bisa diakses oleh *user* melalui sebuah node yang disediakan oleh *filesystem* dalam direktori */dev*. Data dari jaringan juga akan diterima oleh bagian *serial char* melalui utilitas *network*. Bagian *serial character* akan menangani semua *event* baca tulis terhadap *device* yang dilakukan oleh *user*.
2. Utilitas *network*
   Utilitas *network* berfungsi sebagai *interface* modul dari bagian serial ke jaringan. Utilitas *network* akan mengimplementasikan *system call* untuk berhubungan dengan jaringan dengan menggunakan protokol transport UDP.

Dari penjelasan diatas maka keseluruhan proses *device driver* dapat dibagi atas dua subproses yaitu penerimaan dan pemberian data ke aplikasi *user* dan penerimaan dan pengiriman paket data ke jaringan. *Device driver* akan melayani *event* utama yaitu pembacaan dan penulisan *device driver*.

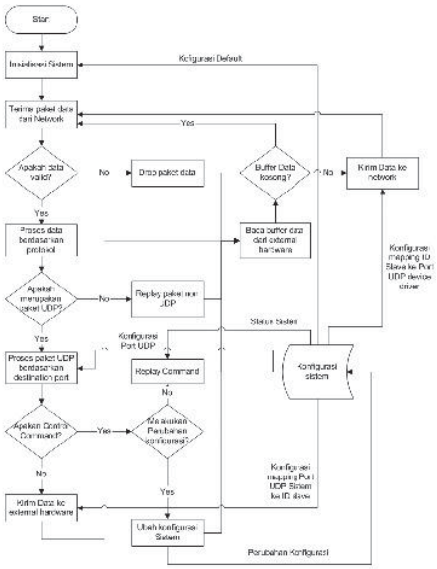
D. Mekanisme komunikasi device driver dan firmware

Komunikasi antara *device driver* dengan *firmware* hanya sebatas pengiriman dan penerimaan data. Dalam proses pengiriman data dalam jaringan komputer, yang menjadi perhatian utama adalah alamat pengiriman data. Oleh karena itu, untuk dapat berkomunikasi dengan *firmware*, *device driver* akan menyimpan informasi berupa alamat IP *remote device* (*hardware*) dan port-port UDP yang dibuka oleh *firmware*. Informasi ini akan digunakan oleh *device driver* sebagai tujuan dalam proses pengiriman paket. Selain itu *user* juga akan mendefinisikan nomot port yang akan dibuka oleh *device driver* untuk menerima data dari *firmware*. Informasi ini akan dikirimkan oleh *device driver* pada saat *device* tersebut dibuka oleh aplikasi *user*. Keseluruhan informasi ini didefinisikan oleh *user* sebagai argumen pada saat melakukan peng-*install*-an *device driver* menggunakan program *insmod*. Informasi ini akan didefinisikan untuk setiap *device driver* yang digunakan untuk berkomunikasi dengan sistem. Mekanisme ini memberikan manfaat untuk membuat *device driver* yang tidak statis yang bisa digunakan untuk berkomunikasi dengan beberapa sistem dengan *firmware* yang sama tetapi memiliki IP *address* yang berbeda.



Gambar 9 Flowchart Pengiriman Data pada *Device* Driver

Data yang dikirim terbagi atas dua bagian yaitu data yang merupakan data yang akan dikirimkan ke *external-hardware* dan data yang ditujukan ke perangkat sistem untuk melakukan konfigurasi terhadap sistem atau disebut dengan *control comand*. Untuk itu *firmware* akan membuka 4 buah port UDP untuk melakukan pengiriman dan penerimaan data dan 1 buah port UDP untuk jalur *control command*. *Firmware* akan melakukan pemeriksaan paket data yang akan diterima untuk melihat jenis data yang diterima dari *network* berdasarkan protokol data dan nomor port yang digunakan. Dari pemeriksaan ini maka data yang diterima dari *network* akan dibagi atas tiga bagian yaitu paket *network* (paket ICMP dan ARP), paket data yang ditujukan ke *external hardware*, dan *control command*.



Gambar 10 Flowchart Pemrosesan Data Communication Manager

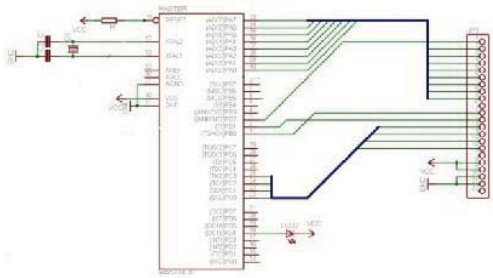
E. Mekanisme pengontrolan sistem

Perangkat fisik sistem yang akan dibuat tidak akan memiliki *keypad* sebagai perangkat masukan untuk melakukan konfigurasi. Padahal, sistem memiliki konfigurasi yang harus disesuaikan dengan konfigurasi jaringan dimana sistem berada. Untuk itu diperlukan mekanisme pengontrolan sistem yang meliputi pengkonfigurasian alamat IP yang digunakan oleh sistem dan pengkonfigurasian komunikasi SPI.

Pada saat ini, metode yang paling banyak digunakan dalam melakukan pengkonfigurasian alamat IP adalah dengan menggunakan DHCP, BOOTP, atau RARP. Dengan metode ini akan dibutuhkan sebuah *server* yang bertindak sebagai DHCP *server*, BOOTP *server*, atau RARP *server*. Selain itu sistem memiliki sumberdaya yang sangat kecil sehingga akan sulit untuk mengimplementasikan protokol DHCP, BOOTP, atau RARP. Untuk itu , pemberian alamat IP untuk pertama kalinya akan dilakukan dengan menggunakan paket *ethernet.* Paket *ethernet* yang berisi paket IP akan dikirim ke alamat *ethernet* perangkat *remote* (*MAC address*) dan perangkat *remote* akan memperoleh IP *address* dari *header destionation address* pada paket *ethernet* yang diterima. Hal ini juga akan dilakukan jika *user* ingin melakukan pengubahan *IP Address* perangkat *remote*.

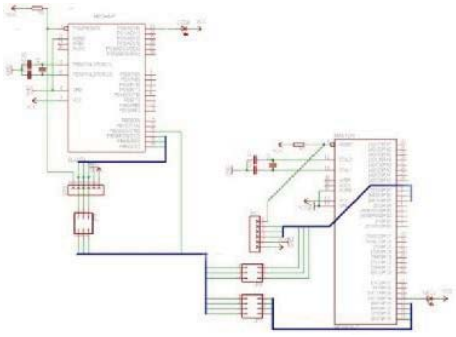
F. Implementasi

Sistem memiliki dua jenis perangkat I/O yaitu satu buah *ethernet* ISA dan 4 buah serial port. Selain itu sistem juga memiliki skema hubungan antar *microcontroller* dengan menggunakan perangkat SPI dan *power supply* yang terhubung ke seluruh komponen *hardware*. Skema koneksi I/O microncontroller dengan slot *ethernet* cs8900A dapat dilihat dalam Gambar 11.



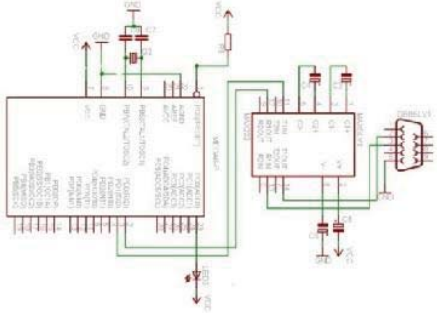
Gambar 11 Skema koneksi *microcontroller* ATmega16 dengan slot *Ethernet* cs8900A

Skema koneksi antar *microcontroller* dengan menggunakan perangkat SPI dapat digambarkan dalam Gambar 12.



Gambar 11 Skema koneksi *Microcontroller Master* dan *Slave* menggunakan SPI

Skema koneksi I/O *microcontroller* dengan TTL/CMOS *transformer* dan *serial port* dapat ditunjukkan dalam Gambar 13.



Gambar 13 Skema koneksi I/O ATMega8, Max232 dan Socket DB9

4. Pengujian dan Pembahasan

Pengujian dilakukan untuk mengetahui apakah sistem sudah memenuhi spesifikasi layanan yang telah dibuat sebelumnya. Selain itu, pengujian juga berguna untuk mengukur kehandalan sistem yang telah dibanguna. Fitur-fitur sistem yang diuji antara lain,

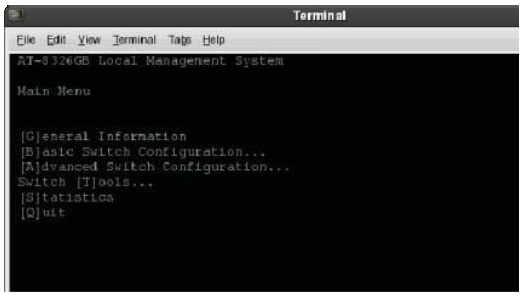
1. Kemampuan mengirim dan menerima data dari serial port
2. Melakukan pemberian atau pengubahan IP *Address* sistem
3. Pendeteksian keberadaan sistem pada jaringan melalui mekanisme *ping*.
4. Kemampuan menerima paket UDP dan meneruskannya ke serial port yang sesuai
5. Kemampuan mengirimkan data yang diterima dari serial port ke host menggunakan paket UDP
6. Kemampuan untuk mengganti IP *Address remote device* yang telah terdefenisi sebelumnya dalam *device driver*.
7. Kemapuan *device driver* untuk menerima data dari *user* dan mengirimkannya ke LAN menggunakan paket UDP
8. Kemampuan *device driver* untuk menerima data dari LAN dan meneruskan data tersebut ke *user* melalui jalur yang tepat

Dalam pengujian yang dilakukan, sistem dibagi atas dua bagian yaitu *device driver* yang terinstall dalam komputer *user* dan perangkat keras sistem yang akan dikenal dengan *remote device*.

Berdasarkan hasil pengamatan terhadap hasil pengujian, jika sistem diletakkan pada lingkungan LAN yang banyak terdapat paket *broadcast*, sistem akan menjadi sibuk untuk menangani paket *broadcast* tersebut yang mengakibatkan pengiriman data dari host ke *external hardware* menjadi lebih lama atau akan mengakibatkan *buffer overflow* pada *buffer* ehernet. Hal ini dapat mengakibatkan hilangnya sebagian data yang dikirim. Selain itu hilang atau rusaknya data bisa terjadi karena kecepatan data yang dikirim ke sistem melalui paket UDP melebihi kecepatan transmisi data *serial port*. Sistem juga akan memiliki respon yang cenderung lebih lambat jika menggunakan 4 buah *serial port* sekaligus untuk berkomunikasi.

Putusnya koneksi komputer *user* ke LAN secara tiba-tiba dapat mengakibatkan gagalnya koneksi antara *device driver* dengan *firmware*. Hal ini diakibatkan tertutupnya port UDP yang dibuka oleh *device driver* karena utilitas jaringan komputer tersebut sudah mati. Selain itu, dari hasil pengujian dapat dilihat jika sistem melakukan pengiriman data pada jumlah yang besar dengan menggunakan seluruh port serial sekaligus dengan kecepatan full (9600 bps), masih didapati adanya kehilangan data. Hal ini mungkin disebabkan adanya *buffer overflow* pada *buffer* penampung data. Akan tetapi pengiriman data dalam jumlah yang banyak untuk satu port tertentu saja dapat berhasil dengan baik.

Walaupun sistem masih memiliki berbagai kekurangan, sistem sudah dapat digunakan sebagai interface antara LAN dan external hardware. Dari gambar 15 bisa dilihat bahwa sistem sudah dapat berfungsi dengan baik sebagai *interface* untuk melakukan pengkonfigurasian sebuah *external hardware* berupa *switch* AT-8326GB.



Gambar 15 sistem sebagai interface external hardware dan LAN

5. Kesimpulan dan Saran

Kesimpulan yang dapat diambil adalah sebagai berikut:

1. Perangkat komunikasi yang dibuat yang dapat digunakan sebagai *interface* antara beberapa *external-hardware* dengan LAN dan *device driver* agar perangkat tersebut bisa berkomunikasi dengan komputer yang mengaksesnya sesuai dengan protokol yang telah dispesifikasikan.
2. Perangkat akan dapat berjalan dengan baik jika kecepatan pengiriman data dari *device driver* ke *remote device* sesuai dengan kecepatan pengiriman data antar *microcontroller* dan *baudrate serial port.*
3. Proses *scheduling* yang dilakukan oleh *microcontroller master* untuk melayani semua *slave* sangat berperan penting untuk mengatasi kemungkinan terjadinya *buffer overflow* pada *microcontroller slave*. Sistem sudah memiliki

mekanisme *scheduling* yang bisa menangani 4 buah *slave* dengan baik, tetapi masih memiliki kekurangan untuk melakukan komunikasi data dengan kecepatan penuh (9600 bps) karena masih adanya terjadi kehilangan data.

4. Perangkat yang dihasilkan sebaiknya ditempatkan pada sebuah jaringan yang memiliki sedikit jumlah paket *broadcast* supaya sistem dapat bekerja dengan baik karena *ethernet driver* akan kewalahan untuk memproses paket *ethernet* yang sangat banyak yang dapat mengkibatkan terjadinya kehilangan data.
5. *Device driver* yang dibuat bukan sebuah *driver* yang menangani perangkat tertentu yang berada dalam komputer, melainkan sebuah perangkat yang terhubung ke komputer melalui LAN. Oleh karena itu, *device driver* yang dibuat dapat dipandang sebagai sebuah aplikasi yang berjalan dalam kernel yang berfungsi untuk berkomunikasi dengan sebuah perangkat yang berada dalam LAN.
6. Pembuatan program dalam *microcontoller* harus mengutamakan efisiensi penggunaan *resource* dan kesederhanaan program yang berkaitan dengan kecepatan pengeksekusian program. Hal ini dikarenakan *microcontroller* memiliki *resource* yang sangat terbatas yang harus digunakan secara efisien.

Daftar Pustaka

[1] www.faqs.org/faqs/microcontroller-faq/primer/, 23-08-2006
[2] mic.unn.ac.uk/miclearning/modules/micros/ch1/ micro01notes.html, 22-08-2006
[3] www.hyperdictionary.com/dictionary/serial+port,22-08-2006
[4] www.taltech.com/TALtech_web/resources/intro-sc.html#Synch, 23-08-2006
[5] www.thedotcommune.com/spi.html, 10-07-2006
[6] Atmel, 61: Setup and use of the SPI, Atmel, 2005
[7] Jonathan Corbet; Greg kroah-Hartman; Allesandro Rubini, Linux Device Driver, 3rd Edition, O'Reilly, 2005

Biodata Penulis

Marojahan M.T. Sigiro, memperoleh gelar Sarjana Teknik (S.T), Program Studi Teknik Informatika ITB, lulus tahun 2007. Tahun 2012 memperoleh gelar Master of Science (M.Sc) dari Program Studi Computer Science, Delft Institute of Technology. Saat ini sebagai Staf Pengajar program Teknik Informatika, Politeknik Informatika Del.