BAB II

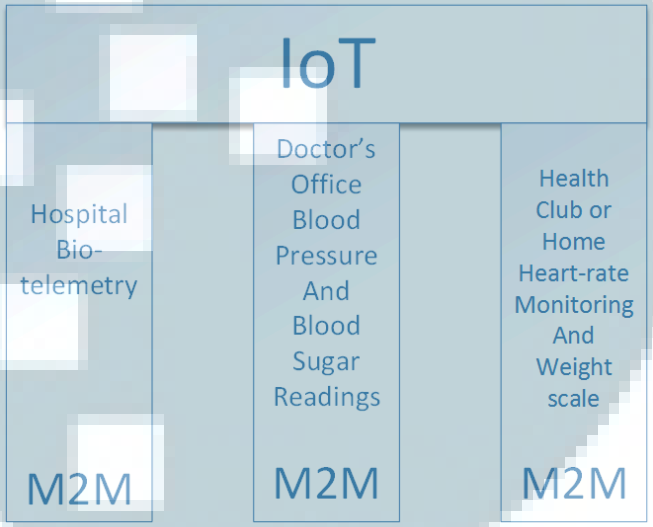
LANDASAN TEORI

2.1. *Internet of Things*

Konsep *Internet of Things (IoT)* secara sederhana adalah menghubungkan *device* apapun yang memiliki saklar *on/off* untuk terhubung ke Internet dan atau ke *device* lain [**6**]. *Device* yang dimaksud dapat berbentuk macam-macam, mulai dari telepon genggam, mesin kopi, mesin cuci, *headphones*, lampu, *wearable devices*, dan *device* lain yang dapat terpikirkan. Selain itu, *IoT* juga berlaku di komponen mesin seperti mesin jet dalam pesawat atau mesin pengeboran minyak. *IoT* merupakan jaringan besar yang terdiri dari "*things*" (benda-benda dan termasuk juga orang) yang saling terhubung.

Machine-to-machine communications (M2M) sering disamakan dengan *IoT*. Akan tetapi, kedua hal tersebut memang serupa namun tak sama. Ada berbagai macam pendapat mengenai perbedaan keduanya. Menurut Chantal Polsenetti pada artikel di *automationworld.com*, *M2M* dan *IoT* memiliki kesamaan di *remote access* [**7**]. Akan tetapi, cara keduanya mencapai *remote access* berbeda. *M2M* menggunakan komunikasi *point-to-point* dengan modul *embedded hardware* dan komunikasi selular atau kabel. Di lain sisi, *IoT* menggunakan jaringan berbasis IP untuk menghubungkan *device* ke *cloud* atau platform *middleware*.

Berbeda lagi pendapat dari Landon Cox pada artikelnya di *pubnub.com* [**8**]. *M2M* dan *IoT* hanya berbeda di skalanya. *IoT* adalah *M2M* dengan konteks yang lebih besar. Cara menggambarkannya adalah *IoT* merupakan jembatan penghubung *M2M*. Contoh seperti pada Gambar 2.1, IoT menggabungkan *health sensor data* dari tiga sumber yang berbeda.



Gambar 2.1 Hubungan antara IoT dan M2M [8]

Smart devices pada IoT menggunakan teknologi internet seperti Wi-Fi untuk berkomunikasi dengan device lainnya, laptop, dan terkadang langsung ke cloud [**9**]. Ada juga yang terhubung ke central point dari berbagai macam device. Idealnya, pemilik dapat mengakses central point itu dari smartphone dan tablet, baik dari rumah maupun dari luar.

Contoh yang paling umum adalah kulkas pintar yang dapat membaca RFID tags pada makanan yang dimasukkan ke dalam kulkas kemudian datanya dikirim melalui Internet untuk mengidentifikasi susu,

telur, mentega, dan makanan lainnya yang baru saja dibeli [**9**]. Kulkas dapat mengamati stok dan dapat memperingatkan pemilik jika ada makanan yang hampir habis. Kulkas juga dapat terhubung dengan kalender untuk mengecek apakah akan ada orang lain yang datang sehingga dapat memperingatkan pemilik jika stok makanan kurang. Bahkan, kulkas dapat memesan makanan sendiri. Kulkas pintar juga dapat memperingatkan mengenai makanan yang mendekati/melewati tanggal kadaluarsanya.

IoT juga tidak dapat dipisahkan dengan *Sensor Network* atau lebih spesifik lagi *Wireless Sensor Network (WSN)*. Saat ini, IoT sinonim dengan WSN [**10**].

Sensor network sendiri adalah suatu infrastruktur yang terdiri dari pengukuran, komputasi, dan elemen komunikasi yang memberikan administrator kemampuan untuk melengkapi, mengobservasi, dan memberikan reaksi terhadap suatu event dan fenomena di suatu lingkungan [**11**]. Terdapat empat komponen dasar dalam *sensor network* yaitu sensor lokal atau sensor yang terdistribusi, jaringan yang saling terkoneksi (biasanya, tetapi tidak selalu, berbasis *wireless*), titik pusat yang mengumpulkan informasi, dan sumber daya untuk komputasi yang berada di titik pusat tersebut atau di tempat lain.

Teknologi sensor terdiri dari sensor elektrik dan medan magnetik; sensor frekuensi gelombang radio; sensor optikal, elektrooptikal, dan infra merah; radar; laser; sensor lokasi/navigasi; sensor seismik dan tekanan

gelombang; sensor parameter lingkungan; dan sensor biokimia untuk keamanan [**11**]. Pada *WSN,* sensor terhubung satu sama lain dalam jaringan *wireless*.

Karakteristik khas *WSN* adalah keterbatasan sumber daya dan daya tahan baterai dalam *wireless network*, penerimaan data yang redundan, *duty cycle* yang rendah, dan hubungan banyak-ke-satu [**11**]. Maka dari itu, metodologi-metodologi baru dibutuhkan, tetapi tidak terbatas, pada bidang transportasi informasi, manajemen jaringan dan operasional, kerahasiaan (*confidentiality*), integritas (*integrity*), ketersediaan (*availability*), dan *in-network/local processing* [**12**].

Efisiensi daya pada *WSN* secara umum dapat dicapai dengan tiga cara, yaitu [**11**]:

1. operasi dengan *duty cycle* yang rendah;
2. *local/in-network processing* untuk mengurangi volume data (dan juga waktu transmisi);
3. jaringan *multihop* untuk mengurangi transmisi jarak jauh karena *signal path loss* berbanding terbalik dengan jarak. Setiap *node* dalam *sensor network* dapat berperan sebagai *repeater* sehingga mengurangi *link range coverage* dan juga daya transmisi.

Pada awal tahun 2000-an, pemasok peralatan sensor melakukan riset mengenai standarisasi *wireless network* (*WN)* [**11**]. *WN* biasanya mengirimkan sedikit data yang sederhana. Untuk aplikasi dalam gedung, perancang mengesampingkan protokol *Wi-Fi* (*Wireless Fidelity, IEEE*

80.11b) untuk sensor karena terlalu kompleks dan menyediakan *bandwidth* lebih banyak daripada yang dibutuhkan. Inframerah membutuhkan *line of sight* yang tidak selalu bisa diperoleh; *Bluetooth* (*IEEE 802.15.1*) awalnya dipertimbangkan, tetapi kemudian dianggap terlalu kompleks dan mahal. Hal ini membuka pintu bagi standar baru yaitu *IEEE 802.15.4* yaitu *ZigBee*. *Zigbee* didesain untuk melengkapi teknologi *Bluetooth, Wi-Fi,* dan *Ultra Wide Band (UWB)*, dan ditargetkan untuk aplikasi sensor *point-to-point* dimana koneksi kabel tidak memungkinkan dan dimana daya yang sangat rendah dan biaya yang rendah dibutuhkan. Tabel 2.1 dan Tabel 2.2 menunjukkan perbandingan antara 802.11 (*Wi-Fi*), 802.15.1 (*Bluetooth*), dan 802.15.4 (*ZigBee*) [**13**] [**14**].

Tabel 2.1 Tabel Perbandingan Protokol *Wireless* [13]

WIRELESS PROTOCOL COMPARISON			
PROPERTY	**802.11**	**802.15.1/BLUETOOTH**	**802.15.4/ZIGBEE**
Range	~100m	~10 to 100 m	~10m
Data throughput	~2 to 54 Mbits/sec	~1 Mbit/sec	~0.25 Mbits/sec
Power consumption	Medium	Low	Ultralow
Battery life measured in:	Minutes to Hours	Hours to Days	Days to Years
Size relationship	Large	Smaller	Smallest
Cost/complexity	>6	1	0.2

Tabel 2.2 Tabel Perbandingan Protokol *Wireless* 2 [14**]**

Feature(s)	IEEE 802.11b	Bluetooth	ZigBee
Power Profile	Hours	Days	Years
Complexity	Very Complex	Complex	Simple
Nodes/Master	32	7	64000
Latency	Enumeration up to 3 Seconds	Enumeration up to 10 seconds	Enumeration 30ms
Range	100 m	10m	10m-300m
Extendibility	Roaming Possible	No	YES
Data Rate	11Mb/s	1Mb/s	250kb/s
Stack size	100+ kbyte	100+ kbyte	8-60 kbyte
Topology	Star	Star	Star, cluster, mesh
Security	Authentication Service Set ID (SSID), WEP	64 bit, 128 bit	128 bit AES and Application Layer user defined

Aplikasi *sensor network* yang sudah ada dan potensial untuk dijalankan terdiri dari *military sensing, physical security, air traffic control, traffic surveillance, video surfeillance, industrial and manufacturing automation, distributed robotics, environment monitoring,* dan *building and structures monitoring* [**15**]. Daftar aplikasi *sensor network* adalah sebagai berikut [**11**].

1. Aplikasi di bidang militer
   a. Memonitor kekuatan musuh
   b. Memonitor kekuatan dan peralatan sendiri
   c. Pengamatan di medan perang
   d. *Targeting*
   e. Melihat kerusakan perang
   f. Deteksi serangan nuklir, biologis, dan kimia

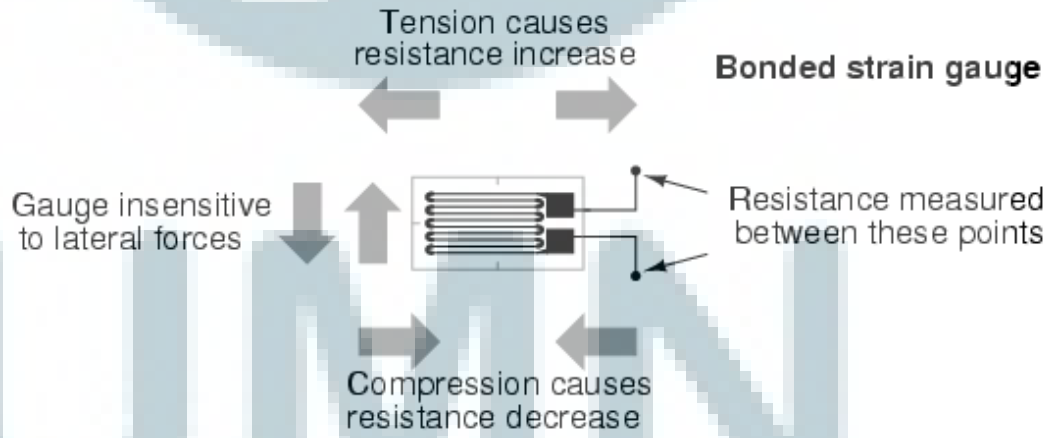
2. Aplikasi di bidang lingkungan
    a. *Microclimates*
    b. Deteksi kebakaran hutan
    c. Deteksi banjir
    d. *Precision agriculture*
3. Aplikasi di bidang kesehatan
    a. Memonitor data fisiologis dari jarak jauh
    b. Memonitor dokter dan pasien dalam rumah sakit
    c. Pemberian obat
    d. Membantu orang tua
4. Aplikasi di rumah
    a. *Home automation*
    b. *Instrumented environment*
    c. Pembacaan meter otomatis
5. Aplikasi di bidang komersial
    a. Kontrol bangunan kantor dan industri
    b. Kontrol stok barang
    c. Pelacakan kendaraan
    d. Pengamatan arus lalu lintas

2.2. *Strain Gauge*

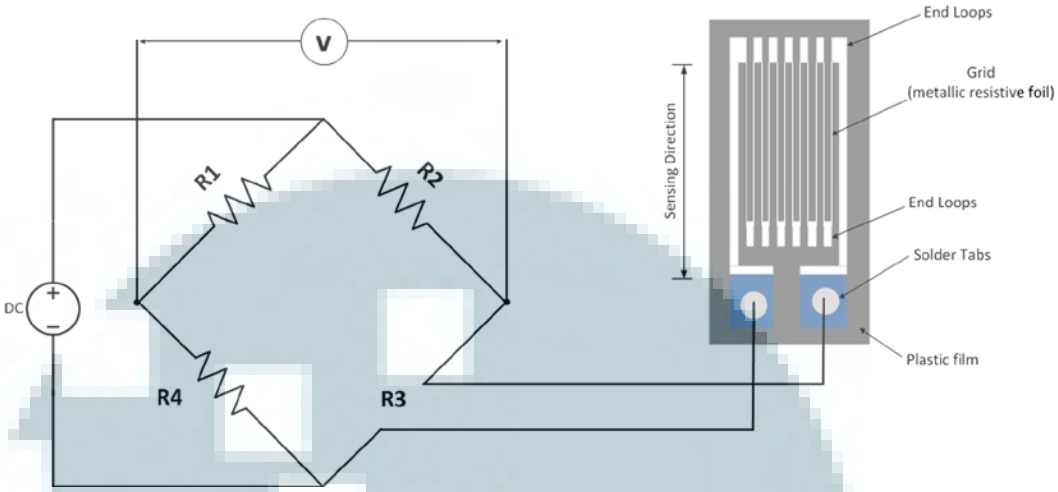
Sensor adalah alat untuk mengukur berbagai macam kondisi dengan mekanisme yang bervariasi [**16**]. Transducer mengonversi keluaran dari

sensor kemudian mengubahnya menjadi sinyal yang dapat diukur. *Strain gauge* merupakan salah satu sensor mekanik yang paling umum digunakan. *Strain gauge* dapat digunakan sebagai sensor tekanan, *load cell* [**17**], sensor tekanan, dan sensor torsi.

Pengukuran dilakukan dengan melihat perubahan resistansi karena material yang meregang (*strained*) atau memendek [**16**]. Ilustrasi dapat dilihat pada Gambar 2.2. *Strain gauge* umumnya memiliki nilai resistansi, mulai dari puluhan ohm hingga ribuan ohm. *Strain gauge* dipakai bersama dengan jembatan Wheatstone seperti pada Gambar 2.3. Saat *strain gauge* diberikan tekanan, maka akan mengubah resistansinya sehingga membuat jembatan Wheatstone tidak seimbang. Jika jembatan Wheatstone tidak seimbang, maka terdapat perbedaan potensial pada jembatan. Perbedaan potensial tersebut yang dibaca.

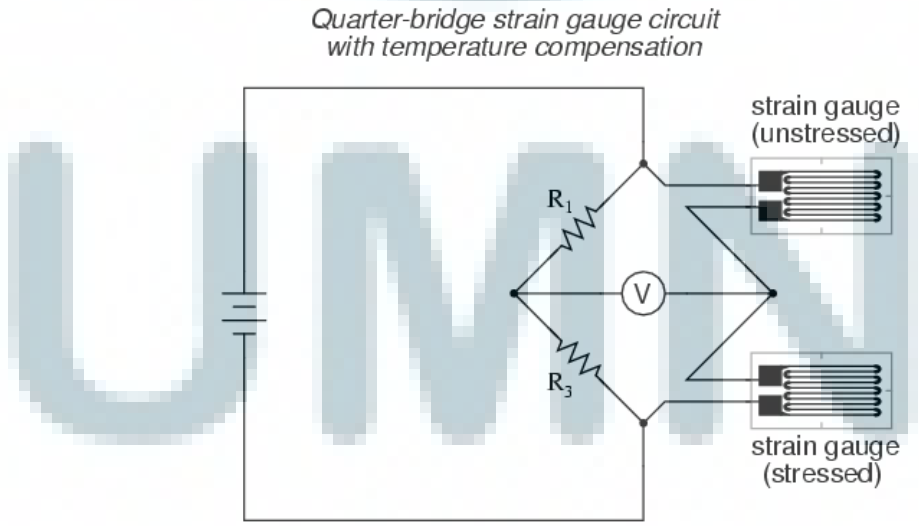


Gambar 2.2 Ilustrasi Perubahan Resistansi pada *Strain Gauge* [18]



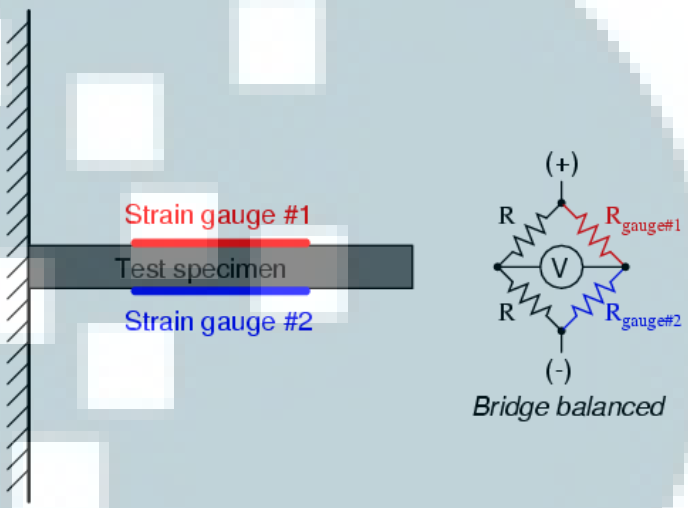
Gambar 2.3 Rangkaian Jembatan Wheatstone dengan *Strain Gauge* [16]

Akan tetapi, perubahan resistansi bisa juga terjadi karena temperatur yang dapat menyebabkan kesalahan pada pembacaan nilai voltase jembatan *Wheatstone* [**18**]. Masalah tersebut dapat diselesaikan dengan menambahkan *strain gauge* tambahan pada rangkaian seperti pada Gambar 2.4 sehingga jembatan tetap seimbang walaupun terjadi perubahan temperatur.

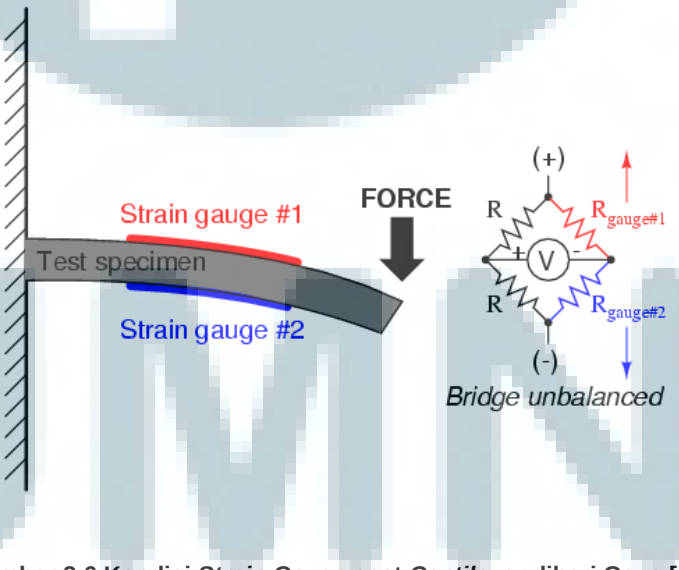


Gambar 2.4 Jembatan *Wheatstone* dengan 2 *Strain Gauges* [18]

Strain gauge dipasang pada *cantilever* seperti pada Gambar 2.5. Pada saat diberi beban seperti pada Gambar 2.6, *strain gauge #1* memanjang dan *strain gauge #2* memendek. Makin diberi gaya, makan makin besar perubahan panjang *strain gauge* dan juga resistansinya.



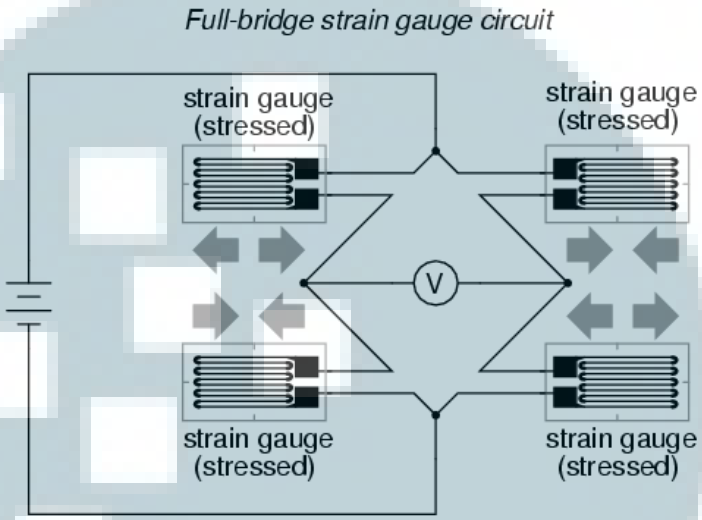
Gambar 2.5 Peletakan *Strain Gauge* pada *Cantilever* [18]



Gambar 2.6 Kondisi *Strain Gauge* saat *Cantilever* diberi Gaya [18]

Supaya lebih sensitif, rangkaian *full-bridge strain gauge* yang menggunakan empat *strain gauge* seperti pada Gambar 2.7 lebih

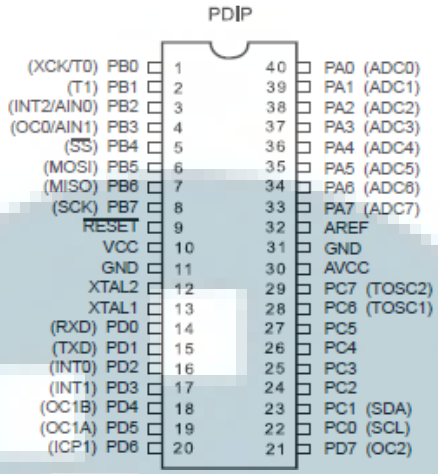
disarankan. Keluarannya proposional dengan gaya yang diberikan (dengan syarat perubahan resistansi keempat *strain gauge* sama). Rangkaian sebelumnya hanya kurang lebih proposional dengan gaya.



Gambar 2.7 Rangkaian *Full-Bridge Strain Gauge* [18]

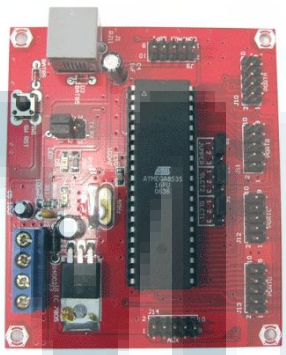
2.3. Mikrokontroler ATMega8535

ATmega8535 merupakan mikrokontroler *AVR* 8 bit dengan 8 *kilobytes* memori *flash*, 512 *bytes SRAM* (*Static Random Access Memory*), dan 512 bytes *EEPROM* (*Electrically Erasable Programmable Read-Only Memory*) [**19**]. ATmega8535 memiliki empat port I/O yang bisa dipakai juga untuk fungsi-fungsi khusus yang bisa dilihat pada Gambar 2.8.



Gambar 2.8 Konfigurasi Pin ATmega8535 [19]

DT-AVR Low Cost Micro System pada Gambar 2.9 merupakan *board* untuk *ATmega8535*, tetapi bisa juga dipakai untuk mikrokontroler *AVR* tipe lain dengan 40 pin [**20**]. Pin ke-1 dan 2 pada masing-masing *port* merupakan GND dan 5V. Pin ke-3 hingga 10 terhubung ke pin ke-0 hingga 7 pada setiap *port* di mikrokontroler. *Board* ini menggunakan dua macam tegangan *input* yaitu 9-12V atau 5V. Untuk menggunakan serial TTL, *jumper* pada J4 dan J5 harus menyambungkan pin 2 dengan pin 3.



Gambar 2.9 *DT-AVR Low Cost Micro System*

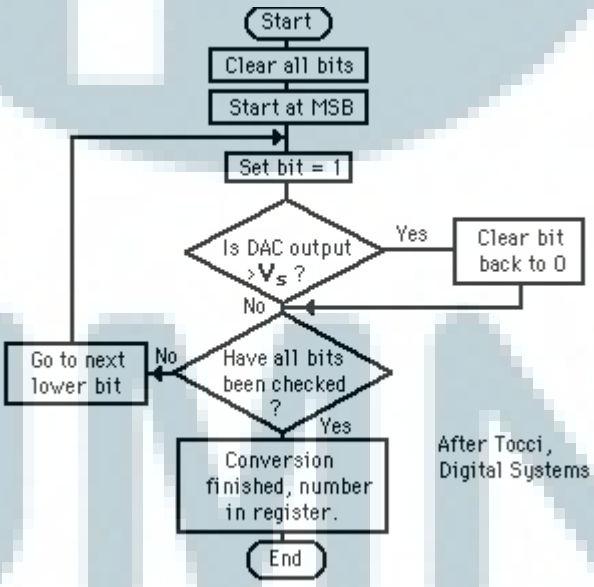
AVR menyediakan bebarapa macam *interrupt*. *Interrupt* dan *Reset Vector* yang terpisah masing-masing memiliki *Program Vector* terpisah di

dalam *program memory space* [**19**]. Semua *interrupt* masing-masing memiliki *enable bit* yang dapat ditentukan nilainya untuk mengaktifkan/menonaktifkan *interrupt*. Alamat terkecil dari *program memory space* secara *default* adalah *Reset* dan *Interrupt Vectors*. Daftar alamat *Reset* dan *Interrupt Vectors* dapat dilihat dari Tabel 2.3. Makin kecil alamatnya, prioritasnya makin tinggi. *Reset* memiliki prioritas yang paling tinggi kemudian diikuti oleh *External Interrupt* 0.

Tabel 2.3 *Reset* dan *Interrupt Vectors* [19]

Vector No.	Program Address(2)	Source	Interrupt Definition
1	0x000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	TIMER2 COMP	Timer/Counter2 Compare Match
5	0x004	TIMER2 OVF	Timer/Counter2 Overflow
6	0x005	TIMER1 CAPT	Timer/Counter1 Capture Event
7	0x006	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	0x007	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	0x008	TIMER1 OVF	Timer/Counter1 Overflow
10	0x009	TIMER0 OVF	Timer/Counter0 Overflow
11	0x00A	SPI, STC	Serial Transfer Complete
12	0x00B	USART, RXC	USART, Rx Complete
13	0x00C	USART, UDRE	USART Data Register Empty
14	0x00D	USART, TXC	USART, Tx Complete
15	0x00E	ADC	ADC Conversion Complete
16	0x00F	EE_RDY	EEPROM Ready
17	0x010	ANA_COMP	Analog Comparator
18	0x011	TWI	Two-wire Serial Interface
19	0x012	INT2	External Interrupt Request 2
20	0x013	TIMER0 COMP	Timer/Counter0 Compare Match
21	0x014	SPM_RDY	Store Program Memory Ready

ATMega8535 memiliki 10 bit *successive approximation ADC* [**19**]. Cara kerja dapat dilihat pada Gambar 2.10. *ADC* terkoneksi dengan 8-*channel Analog Multiplexer* yang menghasilkan delapan pin untuk voltase input yang dapat diakses di *port A*. Setiap pin input me-*refer* ke 0V (GND). Selain itu, ATMega8535 juga menyediakan kombinasi 16 *differential voltage input* pada pin ADC1-ADC0 dan ADC3-ADC2. Fitur ini dapat digunakan jika *input* tidak me-*refer* GND. Pada fitur ini, ATMega8535 menyediakan amplifikasi secara *software* sebesar 1x, 10x, dan 200x. Jika menggunakan amplifikasi 1x atau 10x, resolusi ADC menjadi 8 bit. Jika menggunakan aplifikasi 200x, resolusi ADC menjadi 7 bit. Pemilihan mode *input* diatur dari *Register MUX* bit 4 sampai bit 0 yang dapat dilihat pada Tabel 2.4.



Gambar 2.10 *Successive Approximation ADC* [21]

Tabel 2.4 Pemilihan *ADC Input Mode* [19]

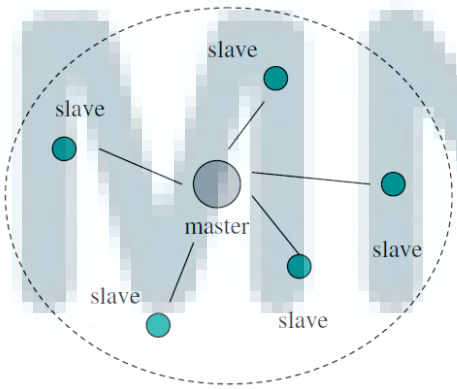
MUX4..0	Single Ended Input	Pos Differential Input	Neg Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x

2.4. *Bluetooth*

Ericsson Mobile Communication memulai penelitian pada 1994 untuk mengetahui kegunaan RF sebagai komunikasi antara telepon genggam,

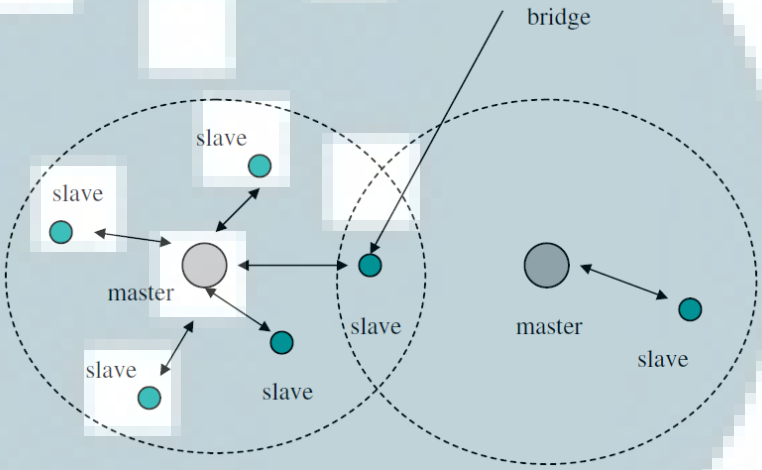
wireless device, dan PC [**22**]. Hasilnya dapat disimpulkan bahwa semua *device* tersebut dapat dihubungkan menggunakan gelombang RF (*Bluetooth*). Untuk membuat standar tersebut, Ericsson bergabung dengan IBM, Nokia, dan Toshiba untuk membentuk *Bluetooth Special Interest Group* (SIG) pada 1998. Pada 1999, Motorola, Microsoft, Lucent, dan 3Com bergabung pada grup tersebut. Secara umum, *Bluetooth* merupakan standar di mana *device* dapat terkoneksi secara *wireless* dengan jarak maksimum 10 meter.

Ada dua macam topologi yang ditawarkan oleh *Bluetooth*, yaitu *Piconet* dan *Scatternet* [**22**]. *Piconet* merupakan topologi standar *Bluetooth* yang terdiri dari satu *master* dan yang lainnya sebagai *slave* seperti yang dapat dilihat pada Gambar 2.11. *Master* dapat terkoneksi ke 7 *active slave* bersamaan atau 200 *inactive slave/parked slave*. Setiap *active slave* memiliki 3 bit alamat (dari 1-7) yang disebut *Active Member Address (AMA)*. Setiap *parked slave* memiliki 8 bit alamat yang disebut *Parked Member Address (PMA)*.



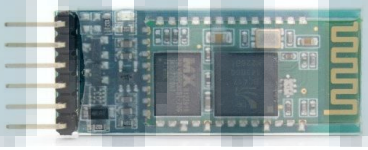
Gambar 2.11 Topologi *Piconet* [23]

Topologi *Scatternet* merupakan gabungan dari beberapa topologi *Piconet* [**22**]. Topologi ini memungkinkan komunikasi antar device yang berbeda piconet. Seperti yang dapat dilihat pada Gambar 2.12, antar piconet dapat terhubung jika ada satu slave yang berada di dua piconet sebagai bridge.



Gambar 2.12 Topologi *Scatternet* [23]

Gambar 2.13 merupakan modul HC-05 yang merupakan salah satu modul *Bluetooth serial interface* yang dipakai untuk keperluan non-industri [**24**]. Modul *Bluetooth serial* dipakai untuk mengonversikan *serial port* ke *Bluetooth*. Terdapat dua mode yaitu *master* dan *slave device*. Untuk HC-05, pengguna dapat mengganti kedua mode tersebut.



Gambar 2.13 Modul *Bluetooth* HC-05

Contoh penggunaan modul *Bluetooth serial* dalam menggantikan *serial port* adalah sebagai berikut [**24**].

a. Jika terdapat dua mikrokontroler yang ingin berkomunikasi satu sama lain, salah satu dihubungkan ke *Bluetooth master device* dan yang lainnya dihubungkan ke *slave*. Koneksi dapat terjadi setelah kedua *device* melakukan *pairing*. Koneksi tersebut sama saja dengan koneksi *serial port line* yang terdiri dari sinyal RXD dan TXD.
b. Saat mikrokontroler memiliki modul *Bluetooth slave*, ia dapat berkomunikasi dengan *bluetooth adapter* yang ada di komputer dan *smart phone*. Terdapat *virtual serial port line* antara mikrokontroler dan komputer atau *smart phone*.
c. *Bluetooth device* di pasar kebanyakan adalah *slave device*, seperti *Bluetooth printer, Bluetooth GPS*. Maka, modul *master* dapat digunakan untuk *pairing* dengan mereka dan berkomunikasi dengan mereka.

Komunikasi antara dua modul *Bluetooth* membutuhkan paling tidak dua kondisi sebagai berikut [**24**].

a. Komunikasi harus terjadi antara *master* dan *slave.*
b. *Password* harus benar.

Untuk melakukan pengaturan pada modul *Bluetooth*, pengguna harus masuk ke mode *AT Command*. Ada dua mode pada HC-05, yaitu **[24]**,

a. *AT Mode 1* : Setelah modul dinyalakan, ia dapat masuk ke *AT Mode* dengan memberikan PIN34 dengan *high level* (3.0V – 4.2V). *Baud rate* untuk *AT Command* sama dengan *baud rate* untuk komunikasi [**24**] [**25**].

b. *AT Mode 2* : Pertama memberikan PIN34 dengan *high level* lalu setelah itu menyalakan modulnya. Atau memberikan PIN34 dengan *high level* sambil menyalakan modul. *Baud rate* yang dipakai adalah 38400 [**24**] [**25**].

PIN34 harus tetap diset pada *high level* jika ingin menggunakan semua *AT Command* yang tersedia [**24**]. Jika tidak, hanya beberapa *AT Command* yang dapat digunakan setelah masuk ke *AT Mode*. Saat proses komunikasi, modul juga dapat masuk ke *AT Mode* dengan mengeset PIN34 dengan *high level* dan dapat kembali ke mode komunikasi saat PIN34 menjadi *low level.*

Pada HC-05, *AT Command* selalu diakhiri dengan "\r\n" dan *case-sensitive* [**25**]. Berikut adalah *AT Command* yang dipakai.

a. *Test*

Command	Respon	Parameter
AT	OK	-

Dipakai untuk mengetes apakah *device* sudah masuk ke *AT Mode* atau belum.

b. Mendapatkan *soft version*

Command	Respon	Parameter
AT+VERSION?	+VERSION:<Param> OK	Param: nomor versi

Untuk mengetahui nomor versi *firmware* yang dipakai.

c. Mengembalikan ke *setting* awal/*default*

Command	Respon	Parameter
AT+ORGL	OK	-

Setting awal terdiri dari:

- *Device type* : 0
- *Inquire code* : 0x009e8b33
- Mode modul : *slave*
- Mode koneksi : koneksi ke *Bluetooth device* yang ditentukan
- Parameter serial : *baud rate* 38400bps; *stop bit* 1 bit; *parity bit* tidak ada
- *Password* : 1234
- Nama *device* : H-C-2010-06-01

d. Mendapatkan alamat Bluetooth

Command	Respon	Parameter
AT+ADDR?	+ADDR: <Param> OK	Param: alamat *Bluetooth*

Misalnya alamat *Bluetooth* adalah 12: 34: 56: ab: cd: ef

Keluaran dari *AT Command* adalah:

+ADDR:1234:56:abcdef

OK

e. Mendapatkan/mengganti nama *device*

Command	Respon	Parameter
AT+NAME=<Param>	OK	Param: nama *device* *Bluetooth* *Default*: HC-05
AT+NAME?	1. Jika sukses, +NAME:<Param> OK 2. Jika gagal, FAIL	

f. Mendapatkan/mengganti mode *device* (*master/slave*)

Command	Respon	Parameter
AT+ROLE=<Param>	OK	Param: 0: *Slave* 1: *Master* 2: *Slave-Loop* *Default*: 0
AT+ROLE?	+ROLE:<Param> OK	

Slave : koneksi pasif

Slave-Loop : koneksi pasif, menerima data dari *master* kemudian mengirimkan kembali ke *master*

Master : mencari *Bluetooth slave* di sekitar, membangun koneksi dengannya, dan membuat transmisi data antara *master* dan *slave*.

g. Mendapatkan/mengganti *password*

Command	Respon	Parameter
AT+PSWD=<Param>	OK	Param: *password* *Default*: 1234
AT+PSWD?	+PSWD:<Param> OK	

h. Mendapatkan/mengganti parameter serial

Command	Respon	Parameter
AT+UART=<Param>,<Param2>,<Param3>	OK	Param: *baud rate* dalam desimal (4800, 9600, 19200, 38400, 57600, 115200, 23400, 460800, 921600, 1382400) Param2: *stop bit* 0: 1 bit 1: 2 bit Param3: *parity bit* 0: Tidak ada 1: *Parity* ganjil 2: *Parity* genap *Default*: 9600,0,0
AT+UART?	+UART:<Param>,<Param2>,<Param3> OK	

2.5. *Zigbee*

Perbedaan utama *Zigbee* dengan *Bluetooth* adalah *Bluetooth* baik digunakan di komunikasi *point-to-point* sedangkan *Zigbee* baik digunakan di kontrol *wireless* di mana dua hingga ribuan *nodes* terhubung satu sama lain di jaringan *multi-hop mesh* [**26**]. *Zigbee* memiliki karakteristik seperti konsumsi daya yang rendah sehingga *Zigbee end device* dapat berjalan dalam waktu berbulan-bulan atau bahkan dalam hitungan tahun tanpa mengganti baterai [**22**]. Selain itu, *Zigbee* itu *low cost,* memiliki *low data rate* (maksimum 250Kbps), mudah diimplementasikan, dapat menampung hingga 65.000 *nodes* dalam satu jaringan, dapat secara otomatis membentuk jaringannya, dan menggunakan paket berukuran kecil dibandingkan dengan *WiFi* dan *Bluetooth*.

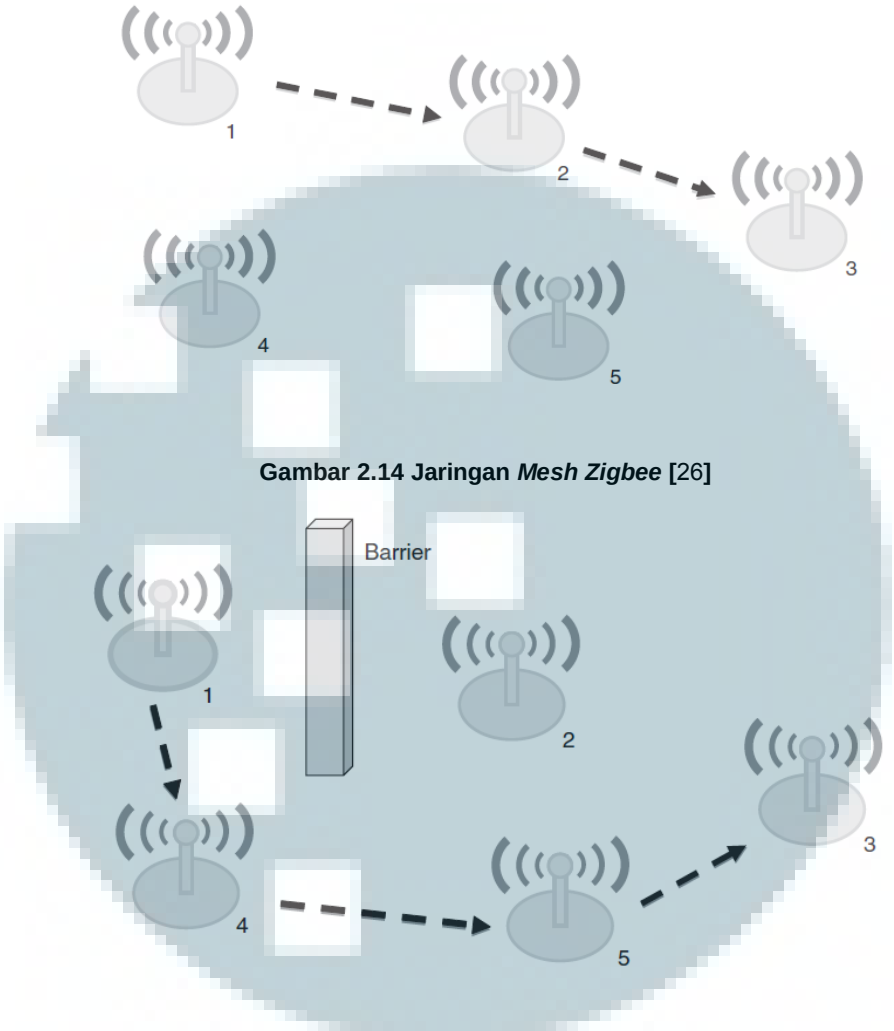
Salah satu hal yang membuat *Zigbee* sangat *reliable* adalah karena menggunakan *Offset-Quadrature Phase-Shift Keying* (*O-QPSK*) dan *Direct Sequence Spread Spectrum* (*DSSS*) [**26**]. Kombinasi keduanya membuat rasio *signal-to-noise* rendah.

Zigbee juga menggunakan *Carrier Sense Multiple Access Collision Avoidance* (*CSMA-CA*) untuk meningkatkan reabilitas [**26**]. Sebelum transmit data, *Zigbee* mendengarkan kanal terlebih dahulu. Jika tidak ada data yang ditransmisikan di kanal, *Zigbee* baru akan transmit data. Hal ini mencegah data *corrupt* saat beberapa data bersamaan ditransmisikan.

Untuk memastikan data yang ditransmisikan benar, *Zigbee* menggunakan 16-bit CRC di setiap paket yang disebut *Frame Checksum*

(*FCS*) [**26**]. Paket dapat ditransmisikan ulang hingga tiga kali (total empat kali transmisi). Jika sampai data keempat masih gagal mengirim data, *Zigbee* akan menginformasikannya ke *node* pengirim.

Cara lain *Zigbee* mencapai reabilitas adalah melalui jaringan *mesh* [**26**]. Jaringan *mesh* dapat meningkatkan tiga kapabilitas dari suatu jaringan *wireless* yaitu meningkatkan jangkauan melalui *multi-hop*, formasi jaringan secara *ad-hoc*, dan yang paling penting dapat menetukan rute secara otomatis dan *self-healing*. Data dari suatu *node* dapat menjangkau ke *node* lainnya di dalam jaringan yang sama di jarak berapapun selama ada *node* lain yang mencukupi untuk me-*relay* data. Seperti pada Gambar 2.14, *node 1* ingin mengirim data ke *node 3* yang berada di luar jangkauan gelombang radio *node 1*. *Zigbee* secara otomatis mencari rute terbaik sehingga data dioper dulu ke *node 2* yang kemudian dilanjutkan ke *node 3*. Jika suatu saat terjadi hal pada rute tersebut (misalnya *node 2* tidak berfungsi lagi/dihilangkan atau terdapat penghalang seperti tembok), *Zigbee* dapat secara otomatis mendeteksi hal tersebut dan membuat rute baru seperti pada Gambar 2.15.

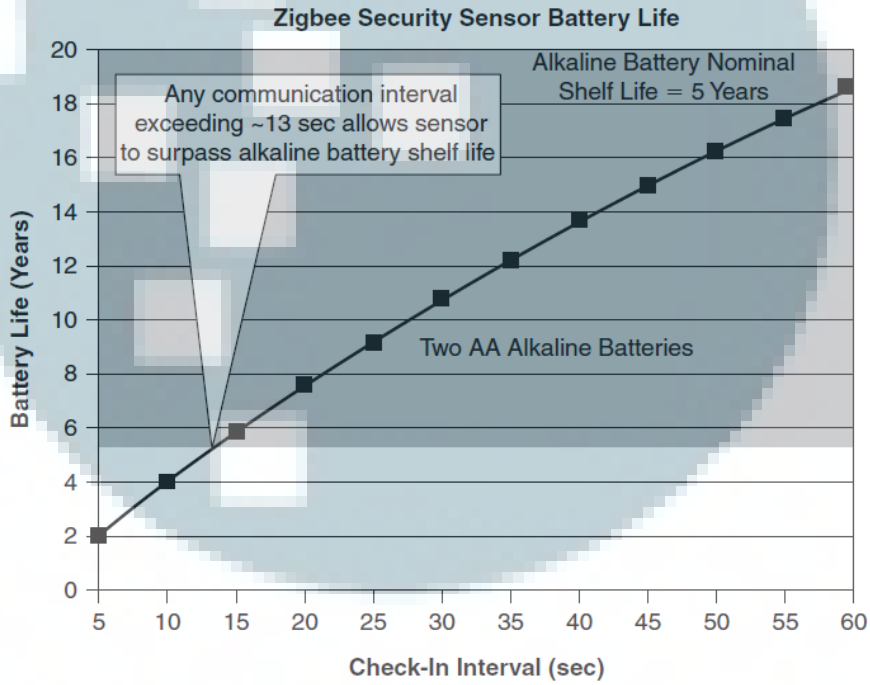


Gambar 2.14 Jaringan *Mesh Zigbee* [26]

Gambar 2.15 *Zigbee Rerouting* [26]

Zigbee device, jika di-*manage* dengan baik, akan bertahan lama [**26**]. Sepasang baterai AA dapat mengoperasikan *Zigbee* hingga lima tahun. Rahasia mengapa *Zigbee* mengonsumsi daya rendah, selain gelombang radio dan mikrokontroler yang dapat *sleep*, adalah karena *low duty cycle*. *Node* di *Zigbee* tidak perlu berkomunikasi secara konstan hanya untuk tetap berada di jaringan. Kenyataannya, jaringan *Zigbee* seringnya sangat diam/tidak aktif. Sebuah sensor temperatur biasanya hanya mentransmisikan data satu jam sekali. Sebuah saklar lampu dapat hanya

diganti statusnya 6 hingga 10 kali dalam sehari atau bahkan kurang. Hasil kalkulasi pada Gambar 2.16 menujukkan bahwa sebuah *Zigbee node* dapat bertahan lebih dari 5 tahun (waktu maksimum pemakaian baterai *Alkaline*) jika komunikasi dilakukan sekali setiap 13 detik atau lebih jarang lagi dengan asumsi radio dan mikrokontroler pada modul *Zigbee* dapat *sleep* dan modul *Zigbee* bekerja dalam *low duty cycle*.



Gambar 2.16 Umur Baterai dengan *Zigbee* [26]

Supaya aman, *Zigbee* menggunakan AES 128-bit untuk enkripsi dan autentikasi [**26**]. Enkripsi dilakukan supaya paket tidak dapat dimengerti oleh *node* yang tidak memiliki *key*. Autentikasi dilakukan supaya *malicious node* tidak dapat menginjeksi *false packet* ke jaringan.

Throughput Zigbee hanya sebesar 25 kilobits per detik sehingga *Zigbee* memiliki *low data rate* [**26**]. Hal ini dilakukan karena *Zigbee*

memiliki spektrum 2.4GHz yang juga merupakan spektrum dari *WiFi, Bluetooth,* beberapa telepon tanpa kabel, dan bahkan *microwave oven*. Supaya dapat berada di antara teknologi-teknologi yang sudah ada, *Zigbee* membuat *low data rate* sehingga tidak mengganggu teknologi lain.

ETRX2 merupakan salah satu modul *Zigbee*. Tabel 2.5 merupakan tipe *device* yang disediakan oleh ETRX2. *Zigbee Coordinator* memiliki tugas membentuk jaringan, membuat kanal 802.15.4 di mana jaringan akan beroperasi, membuat *PAN ID*, menentukan *stack profile* yang akan digunakan, menjadi *Trust Center* untuk aplikasi dan jaringan yang aman, menjadi penengah hubungan *End Device*, menjadi *router* di jaringan *mesh*, dan menjadi puncak *tree* jika *tree routing* diaktifkan [**26**]. *Zigbee Router* bertanggung jawab dalam mencari dan bergabung ke jaringan yang benar, melanjutkan pesan *broadcast* di dalam jaringan, berpartisipasi di dalam *routing* (termasuk mencari dan me-*maintain* rute), mengijinkan *device* lain untuk bergabung ke jaringan, dan menyimpan *packet* untuk *sleeping children*. *Zigbee End Device* bertanggung jawab untuk mencari dan bergabung ke jaringan yang benar, melakukan *polling* ke *parent* untuk mengetahui apakah ada pesan yang ditujukan ke dirinya saat dirinya dalam keadaan *sleep*, mencari *parent* baru jika putus hubungan dengan *parent* lama, dan mengirit daya dengan cara *sleep* saat tidak digunakan oleh aplikasi. Akan tetapi, *parent* pada modul ETRX2 tidak menyimpan pesan untuk *children* di *buffer* melainkan langsung diteruskan ke *children* sehingga *Zigbee End Device* sebaiknya tidak masuk ke *sleep mode* [**27**].

Tabel 2.5 Tipe *Device ETRX2* [27]

Device Types		ZigBee Naming Convention
COO	Coordinator	ZigBee Coordinator (ZC)
FFD	Router	ZigBee Router (ZR)
ZED	End Device (non sleepy)	ZigBee End Device (ZED)
SED	Sleepy End Device	
MED	Mobile Sleepy end Device	

Ada dua alamat yang dipakai *coordinator* atau *router* untuk menamai *device*. Yang pertama, ada alamat *logical* sepanjang 16 bit [**22**]. Selain itu ada alamat *IEEE* sepanjang 64 bit yang unik dan tidak ada dua *device* yang memiliki alamat *IEEE* yang sama. Alamat 16 bit dapat dipakai di *device* yang berada di jaringan yang sama. Keuntungan menggunakan alamat 16 bit adalah memperpanjang umur baterai karena panjang *frame* lebih kecil sehingga waktu transmisi juga lebih singkat. Akan tetapi kerugian menggunakan alamat 16 bit adalah bisa ada dua *node* yang memiliki alamat yang sama. Alamat 16 bit (4 digit heksadesimal) di ETRX2 disebut NodeID [**27**]. Alamat 64 bit (16 digit heksadesimal) disebut EUI64.

Gambar 2.17 Konfigurasi Pin *ETRX2* [28]

Gambar 2.17 dan Tabel 2.6 merupakan konfigurasi pin pada modul *ETRX2*. Pin yang dipakai adalah pin GND, pin VCC, pin TXD sebagai

keluaran serial, dan pin RXD sebagai masukan serial. Modul dapat menerima VCC dari 2.1V sampai 3.6V [**28**].

Tabel 2.6 Konfigurasi Pin *ETRX2* [28]

ETRX2 Pad	Function	EM250 GPIO	ETRX2 Harwin Pin
1	GND	GND	
2	Antenna		
3	GND	GND	
4	I/O9	GPIO 0 (21)	1
5	Vreg {1}		2
6	GND	GND	3
7	Vcc		10
8	GND	GND	3
9	A/D1	GPIO 4 (26)	4
10	A/D2	GPIO 5 (27)	5
11	I/O7	GPIO 3 (25)	6
12	I/O6	GPIO 2 (24)	7
13	I/O5	GPIO 1 (22)	8
14	I/O4 or RTS {3}	GPIO 12 (20)	9
15	GND	GND	
16	SIF CLK	SIF CLK	
17	SIF MISO	SIF MISO	
18	SIF MOSI	SIF MOSI	
19	SIF LOADB	SIF LOADB	
20	GND	GND	
21	I/O8	GPIO 6 (29)	11
22	I/O2 or CTS {3}	GPIO 11 (19)	12
23	I/O3	GPIO 13 (43)	13
24	Reset	(13)	14
25	I/O1	GPIO 14 (42)	15
26	I/O0	GPIO 8 (31)	16
27	TXD {2}	GPIO 9 (32)	18
28	RXD {2}	GPIO 10 (33)	17
29	GND	GND	3
30	I/O10	GPIO 15 (41)	19
31	I/O11	GPIO 16 (40)	20
32	GND	GND	
33	RXTXSW	(11)	
34	GND	GND	
35	GND	GND	
36	GND	GND	
37	GND	GND	
38	N/C {4}	GPIO 7 (30)	

Untuk mengganti pengaturan dan mengirim pesan pada ETRX2, AT Command dibutuhkan. Setiap AT Command diakhiri dengan karakter <CR> atau '\r' [**27**]. Setiap baris respon dan *prompt* diawali dan diakhiri

dengan <CR><LF> atau '\r\n'. AT Command yang dipakai pada skripsi ini adalah sebagai berikut.

a. AT+EN : membentuk jaringan PAN baru (lalu menjadi *Coordinator*).

   Respon : 1. JPAN:<channel>,<PID>,<EPID>
   OK
   2. ERROR:<errorcode>

b. AT+JN : bergabung ke jaringan yang ada.

   Respon : 1. JPAN:<channel>,<PID>,<EPID>
   OK
   2. ERROR:<errorcode>

c. AT+N? : menampilkan informasi jaringan.

   Respon : 1. +N=<devicetype>,<channel>,<power>,<PID>,<EPID>
   OK
   2. +N=NoPAN
   OK

d. AT+IDREQ:<Address>[,XX] : mengetahui NodeID.

   <Address> : dapat merupakan EUI64. Diisi FF jika ingin mengetahui NodeID sendiri.

   XX : *optional.* Berisi nomor indeks

   Respon : 1. OK
   2. ERROR:<errorcode>

Setelah respon di atas kecuali errorcode 00, terdapat *prompt*: AddrResp:<errorcode>[,<NodeID>,<EUI64>] [nn. <NodeID>]

e. AT+EUIREQ:<Address>,<NodeID>[,XX] : mengetahui EUI64.

<Address> : dapat merupakan EUI64. Diisi FF jika ingin mengetahui NodeID sendiri.

XX : *optional.* Berisi nomor indeks

Respon : 1. SEQ:XX
OK

2. ERROR:<errorcode>

Setelah respon di atas kecuali errorcode 00, terdapat *prompt*: AddrResp:<errorcode>[,<NodeID>,<EUI64>] [nn. <NodeID>]

f. AT+ANNCE : *broadcast* ID dirinya ke semua *node* di jaringan.

Respon : 1. OK

2. ERROR:<errorcode>

Node lain akan mendapatkan *prompt* di bawah ini tergantung tipe *device* yang *broadcast:*

FFD:<EUI64>,<NodeID> [,syy,zz]

MED:<EUI64>,<NodeID> [,syy,zz]

SED:<EUI64>,<NodeID> [,syy,zz]

ZED:<EUI64>,<NodeID> [,syy,zz]

g. AT+BCAST:nn,<data> : *broadcast* pesan ke semua *node* di jaringan.

nn : jumlah maksimum *hop* untuk *broadcast*. Nilai dari 00 sampai 30. Jika bernilai 00, semua *node* akan mendapatkan pesan.

Respon : 1. OK

2. ERROR:<errorcode>

Node lain mendapatkan *prompt* setelah menerima *broadcast*:

BCAST:[<EUI64>,]<length>=<data>

EUI64 adalah EUI64 pengirim.

h. AT+UCAST:<address>=<data> : mengirimkan *unicast* ke salah satu *node* di jaringan.

address : alamat tujuan. Dapat berupa EUI64 atau NodeID.

Respon : 1. SEQ:XX

OK

2. ERROR:<errorcode>

Pengirim akan mendapatkan *prompt* di bawah ini untuk mengetahui apakah pesan berhasil dikirim atau tidak.

1. ACK:XX
2. NACK:XX

Node penerima akan mendapatkan *prompt:*

UCAST:[<EUI64>,]<length>=<data>

i. ATSXX[x[x]]=<data>[,<password>] : mengganti nilai di *S-Register*.

Respon : 1. OK

2. ERROR:<errorcode>

S-Register yang diubah nilainya:

1. S12 : mengatur *UART*. Terdiri dari 4 digit heksadesimal.
2. S0E : menampilkan/menghilangkan *prompt* tertentu.

2.6. *XML-RPC*

XML-RPC merupakan suatu protokol yang menggunakan XML (eXtensible Markup Language) untuk menggambarkan *method call* dan *method result* [**29**]. *XML-RPC* dipakai supaya *software* yang berbeda bahasa dan berbeda sistem operasi dapat melakukan *method call* satu sama lain melalui Internet. *RPC* adalah *Remote Procedure Call*.

Pesan *XML-RPC* merupakan pesan *HTTP POST request* [**30**]. Bagian badan *request* berupa *XML*. Kemudian prosedur dijalankan di server dan mengembalikan *value* yang juga berupa *XML*. Parameter prosedur dapat berupa skalar, angka, *string*, tanggal, *struct*, dll.

Tabel 2.7 merupakan langkah-langkah komunikasi antara *client* dan *server* melalui *XML-RPC*.

Tabel 2.7 Prosedur Komunikasi *XML-RPC* [29].

Sisi *Client*	Sisi *Server*
1. *Client* membuat *XML element methodCall* yang berisi *method* yang akan dipanggil dan memberikan parameter untuk menjalankan *method.*	
2. *Client* mengirimkan pesan *POST request* yang berisi *XML element* yang baru saja dibuat.	
	3. *Server* menerima *request* dan menggunakan *HTTP header*

	Content-Length untuk mengetahui panjang *XML element*.
	4. *Server* mem-*parse XML element*, mengambil nama *method*, dan menerima parameter untuk menjalankan *method* dari *XML element*.
	5. *Server* mencari *method* yang diinginkan dan, jika ditemukan, menjalankannya dengan parameter yang diterima.
	6. Jika *method* berhasil dieksekusi, *server* membuat *return value* dalam *XML element methodResponse* dengan sebuah *params element* dan mengirimkannya ke *client*.
	7. Jika *method* tidak ditemukan atau tidak dapat dijalankan karena alasan tertentu, *server* membuat *XML element methodResponse* yang isi dari *value element* adalah *fault element* kemudian mengirimkannya ke *client*.
8. *Client* menerima respon, mem-*parse XML element* yang dikembalikan *server* yang dapat berupa *value* atau *fault* kemudian melaporkannya ke *client user*.	

Di bawah ini adalah contoh *XML-RPC request* yang dikirimkan *client* ke *server* [**30**]. Format *URI* pada baris pertama *request* tidak memiliki format khusus. *URI* dipakai untuk membantu mengarahkan *request* ke *code* yang meng-*handle XML-RPC request*. *User-Agent* dan *Host* harus diisi. *Content-Type* adalah *text/xml. Content-Length* harus diisi dan harus benar.

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><i4>41</i4></value>
      </param>
    </params>
  </methodCall>
```

Payload berbentuk format *XML*, dalam satu <*methodCall*> *stucture* yang harus berisi sebuah <*methodName*> *sub-item* yang berisi sebuah *string* yang merupakan nama *method* yang akan dipanggil [**30**]. *Server* dapat menentukan sendiri bagaimana menginterpretasikan *methodName*, misalnya *methodName* dapat berupa nama *file* atau nama *cell* di *database*. Jika ada parameter, <*methodCall*> harus berisi sebuah <*params*> *sub-item* yang berisi berapapun <*param*> yang masing-masing <*param*> berisi sebuah <*value*>.

Isi dari <*value*> dapat berupa skalar [**30**]. Tipe data ditentukan dari *tag* yang berada di dalam <*value*>. Tabel 2.8 menggambarkan tipe data dan *tag*-nya. Jika tipe data tidak disebutkan, dianggap tipe data *string*.

Tabel 2.8 Tipe Data dan *Tag* dalam <*value*> [30**]**

Tag	**Tipe Data**	**Contoh**
<i4> or <int>	four-byte signed integer	-12

<boolean>	0 (false) or 1 (true)	1
<string>	String	hello world
<double>	double-precision signed floating point number	-12.214
<dateTime.iso8601>	date/time	19980717T14:08:55
<base64>	base64-encoded binary	eW91IGNhbid0IHJlYWQgdGhpcyE=

Selain tipe data di atas, isi dari <*value*> juga dapat berupa *struct*. Tipe *struct* menggunakan *tag* <*struct*> yang berisi satu atau beberapa tag <*member*> yang masing-masing *tag* <*member*> berisi sebuah <*name*> dan <*value*> [**30**]. *Struct* dapat rekursif, yang artinya isi dari <*value*> dapat berisi <*struct*> lagi. Di bawah ini adalah contoh isi dari tipe data <*struct*> yang berisi dua elemen.

```
<struct>
  <member>
    <name>lowerBound</name>
    <value><i4>18</i4></value>
    </member>
  <member>
    <name>upperBound</name>
    <value><i4>139</i4></value>
    </member>
</struct>
```

Isi <*value*> juga dapat berupa *array.* Tipe data *array* memiliki *tag* <*array*> yang berisi sebuah <*data*> yang berisi satu atau beberapa <*value*> [**30**]. *Array* dapat rekursif, yang artinya isi dari <*value*> dalam *array* dapat berupa *array* lagi. Di bawah ini adalah contoh *array* yang berisi empat elemen.

```
<array>
  <data>
    <value><i4>12</i4></value>
    <value><string>Egypt</string></value>
    <value><boolean>0</boolean></value>
    <value><i4>-31</i4></value>
    </data>
  </array>
```

Kecuali ada *low-level error*, *server* selalu mengembalikan 200 OK [**30**]. Di bawah ini merupakan contoh respon dari *server* ke *client*. *Content-Type* adalah *text/xml*. *Content-Length* harus ada dan benar. Konten respon berisi satu *XML structure,* sebuah <*methodResponse*>*,* yang dapat berisi sebuah <*params*> yang berisi satu atau beberapa <*param*> yang masing-masing <*param*> berisi sebuah <*value*>*.*

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>South Dakota</string></value>
      </param>
    </params>
  </methodResponse>
```

Selain berisi <*params*>, <*methodResponse*> juga dapat berisi sebuah <*fault*> yang berisi sebuah <*value*> yang berisi sebuah <*struct*> [**30**]. <*struct*> tersebut terdiri dari dua element, yang satu bernama <*faultCode*> yang merupakan sebuah <*int*>, dan yang lainnya bernama <*faultString*> yang beerupakan sebuah <*string*>. Sebuah <*methodResponse*> tidak dapat memiliki <*fault*> dan <*params*> dalam waktu yang bersamaan. Di bawah ini merupakan contoh respon dengan <*fault*>.

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 426
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:02 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>4</int></value>
          </member>
        <member>
          <name>faultString</name>
          <value><string>Too many parameters.</string></value>
          </member>
        </struct>
      </value>
    </fault>
  </methodResponse>
```