BAB II

STUDI AWAL

2.1 Karya yang berkaitan dengan alat ukur otomatisasi piranti titrasi pada Lab. Tanah Pertanian UMY

2.1.1 Inovasi *titrasi*

Titrasi merupakan suatu metode yang bertujuan untuk menentukan banyaknya suatu larutan dengan konsenstrasi yang telah diketahui agar tepat habis bereaksi dengan sejumlah larutan yang dianalisis atau ingin diketahui kadarnya atau konsentrasinya. Prosesnya dilakukan di dalam Laboratorium yang tentunya semuanya menggunakan peralatan kimia. Peralatan tersebut dapat di bedakan menjadi 2 bagian yaitu perangkat keras dan perangkat cair.

a. Perangkat keras

   Adapun yang tergolong kedalam perangkat keras yaitu seperti *titrator*, dan tabung *erlenmeyer*.

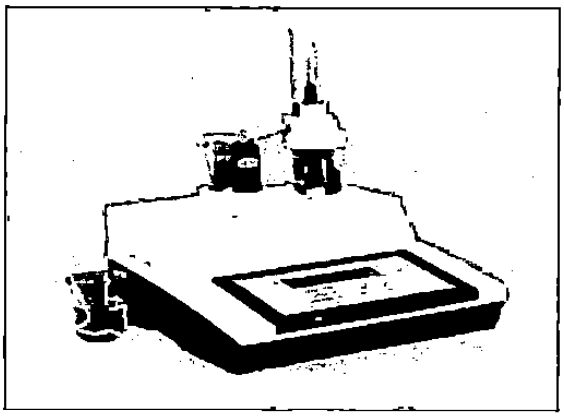
b. Perangkat cair

   Perangkat cair yaitu terdiri dari larutan atau cairan *asam basa*, tergantung jenis yang akan di tentukan konsentrasi larutannya. Dan indikatornya yaitu *cairan fenolftalein, timolftalein, metal, timol dan alizarin*

2.1.2 Alat ukur otomatisasi piranti titrasi selain yang ada di Lab Pertanian UMY

Berikut ini adalah contoh alat ukur yang ada di pasaran tetapi belum di gunakan pada Lab Pertanian UMY dikarenakan harga yang terlalu mahal dan kepentingan kegunaan yang digunakan untuk praktikum, salah satu contoh alat yang ada di pasaran dapat di lihat pada gambar 2.1

ZDJ-4A Automatic Titrator



Gambar 2.1 *Automatic titator*

Harga : US $2,930

2.2 Dasar - dasar teori

2.2.1 Metode *titrasi*

Titrasi merupakan suatu metode yang bertujuan untuk menentukan banyaknya suatu larutan dengan konsenstrasi yang telah diketahui agar tepat habis bereaksi dengan sejumlah larutan yang dianalisis atau ingin diketahui kadarnya

Titrasi biasanya dibedakan berdasarkan jenis reaksi yang terlibat di dalam proses *titrasi,* sebagai contoh bila melibatan reaksi *asam basa* maka disebut sebagai *titrasi asam basa, titrasi redox* untuk *titrasi* yang melibatkan reaksi *reduksi oksidasi, titrasi kompleksometri* untuk *titrasi* yang melibatkan pembentukan reaksi kompleks dan lain sebagainya. (disini hanya dibahas tentang *titrasi asam basa*).

Zat yang akan ditentukan kadarnya disebut sebagai "*titrant*" dan biasanya diletakan di dalam *erlenmeyer,* sedangkan zat yang telah diketahui konsentrasinya disebut sebagai *"titer"* dan biasanya diletakkan di dalam "*buret*". Baik *titer* maupun *titrant* biasanya berupa larutan.

Prinsip *Titrasi Asam basa*

Titrasi asam basa melibatkan *asam* maupun *basa* sebagai *titer* ataupun *titrant. Titrasi asam basa* berdasarkan reaksi penetralan. Kadar larutan *asam* ditentukan dengan menggunakan larutan basa dan sebaliknya.

Titrant ditambahkan *titer* sedikit demi sedikit sampai mencapai keadaan *ekuivalen* (artinya secara *stoikiometri titrant* dan *titer* tepat habis bereaksi). Keadaan ini disebut sebagai "*titik ekuivalen*". *Stoikiometri* merupakan ilmu yang mempelajari dan menghitung hubungan kuantitatif dari reaktan dan produk dalam

Saat terjadinya titik *ekuivalen* maka proses *titrasi* dihentikan, kemudian kita mencatat volume *titer* yang diperlukan untuk mencapai keadaan tersebut. Dengan menggunakan data volume *titrant*, dengan volume dan *konsentrasi titer* maka kita bisa menghitung kadar *titrant*.

Cara Mengetahui *Titik Ekuivalen*

Ada dua cara umum untuk menentukan *titik ekuivalen* pada *titrasi asam basa*.

1. Memakai pH meter untuk memonitor perubahan pH selama *titrasi* dilakukan, kemudian membuat *plot* antara pH dengan volume *titrant* untuk memperoleh kurva *titrasi*. Titik tengah dari kurva *titrasi* tersebut adalah "*titik ekuivalen*".

2. Memakai *indikator asam basa*. Indikator ditambahkan pada *titrant* sebelum proses *titrasi* dilakukan. Indikator ini akan berubah warna ketika *titik ekuivalen* terjadi, pada saat inilah *titrasi* kita hentikan.

Umumnya cara kedua dipilih disebabkan kemudahan pengamatan, tidak diperlukan alat tambahan, dan sangat praktis.

Indikator *Asam-basa*

Indikator *asam-basa* adalah zat yang berubah warnanya atau membentuk *fluoresen* atau kekeruhan pada suatu range *(trayek) pH* tertentu. Indikator *asam-basa* terletak pada *titik ekivalen* dan ukuran dari *pH*. Zat-zat indikator dapat berupa asam atau basa, larut, stabil, dan menunjukkan perubahan warna yang kuat

serta biasanya adalah *zat organik.* Perubahan warna disebabkan oleh *resonansi isomer electron.*

Indikator *asam-basa* secara garis besar dapat diklasifikasikan dalam tiga golongan:

1. *Indikator ftalein dan indicator sulfoftalein*
2. *Indikator azo*
3. *Indikator trifenilmetana*

Indikator ftalein dibuat dengan *kondensasi anhidrida ftalein* dengan *fenol,* yaitu *fenoftalein.* Pada *pH 8,0-9,8* berubah warnanya menjadi merah.

Anggota-anggota lainya adalah : *o-cresolftalein, thimolftalein, -naftolftalein. Indikator sulfoftalein* dibuat dari *kondensasi anhidrida ftalein dan sulfonat.* Yang termasuk dalam kelas ini: *thymol blue, m-cresolpurple, chlorofenolred, bromofenolred, bromofenolblue, bromocresolred,* dan sebagainya. *Indikator azo,* diperoleh dari reaksi *amina romatik dengan garam dizonium,* misalnya: *methylyellow atau p-dimetil amino azo benzene.* Perubahan warna terjadi pada larutan asam kuat. *Metil-orange* tidak larut dalam air. Indikator yang lain yang [illegible]

Tabel 2.1. Daftar indikator *asam basa*

NAMA	pH RANGE	WARNA	TIPE(SIFAT)
Biru timol	1,2-2,8	merah - kuning	asam
Kuning metil	2,9-4,0	merah - kuning	basa
Jingga metil	3,1 – 4,4	merah - jingga	basa
Hijau bromkresol	3,8-5,4	kuning - biru	asam
Merah metil	4,2-6,3	merah - kuning	basa
Ungu bromkresol	5,2-6,8	kuning - ungu	asam
Biru bromtimol	6,2-7,6	kuning - biru	asam
Merah fenol	6,8-8,4	kuning - merah	asam
Ungu kresol	7,9-9,2	kuning - ungu	asam
Fenolftalein	8,3-10,0	t.b. - merah	asam
Timolftalein	9,3-10,5	t.b. – biru	asam
Kuning alizarin	10,0-12,0	kuning - ungu	basa

Indikator yang sering digunakan dalam *titrasi asam basa* yaitu indikator *fenolftalein.* Tabel 2.2 merupakan karakteristik dari indikator *fenolftalein.*

Tabel 2.2. Indikator yang sering digunakan pada proses *titrasi asam basa*

pH	< 0	0–8.2	8.2–12.0	>12.0
Kondisi	Sangat asam	Asam atau mendekati netral	Basa	Sangat basa
Warna	Jingga	Tidak berwarna	pink keunguan	Tidak berwarna
Gambar				

Untuk memperoleh ketepatan hasil titrasi maka titik akhir *titrasi* dipilih sedekat mungkin dengan titik *equivalen*, hal ini dapat dilakukan dengan memilih indikator yang tepat dan sesuai dengan *titrasi* yang akan dilakukan. Keadaan dimana *titrasi* dihentikan dengan cara melihat perubahan warna indikator disebut sebagai "titik akhir *titrasi*"

Pengembangan Proses titrasi mulai dari proses manual yang ada atau berlangsung hingga proses otomatis yang di bantu dengan alat Titrasi otomatis yang di *design* oleh mahasiswa teknik elektro UMY di Lab Pertanian UMY saya bagi menjadi 3 tahap proses, termasuk proses pengembangan saya adalah proses tahap 3 yang merupakan bagian dari pengembangan alat titrasi otomatis yaitu :

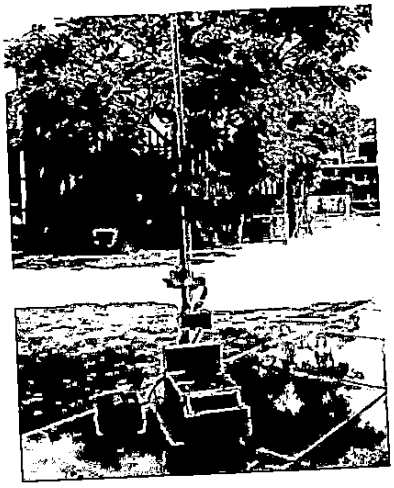
1. Proses manual
2. Proses otomatis 1
3. Proses otomatis 2

1. Proses Manual

Gambar 2.2 Proses manual titrasi asam basa

Pada gambar 2.2 dapat dilihat bahwa hal tersebut merupakan salah satu contoh gambar proses titrasi secara manual yang dimana terdapat kekurangan yaitu dimana manusia nya sendiri turut langsung atau menjadi bagian dari proses manual itu sendiri, baik sistem pengadukan dan settingan kran yang berubah – ubah, dan juga penghitungan manual

2. Proses menggunakan alat titrasi otomatis



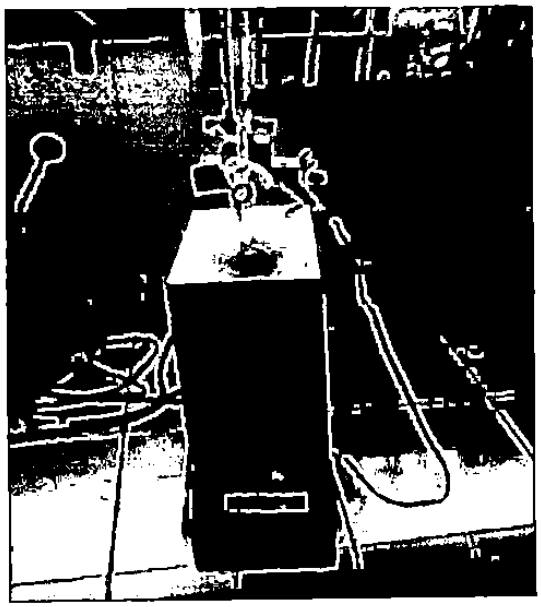
Gambar 2.3 Alat ukur Titrasi Otomatis 1

Gambar 2.3 merupakan alat ukur Titrasi otomatis yang pertama kali di rancangan oleh mahasiswa elektro UMY angkatan 2003, yang sempat di uji cobakan pada lab pertanian UMY, akan tetapi alat ini masih memiliki kekurangan yaitu :

1. Belum bisa beroperasi secara otomatis (atau sensor warna belum difungsikan)
2. Design elektronika yang masih banyak mengalami korslet terhadap *casing box*

3. Tidak bekerjanaya motor stepper sebagai fungsi otomatis penutup dan pembuka kran buret.

3. Proses Otomatis 2 (Pengembangan 1) oleh saya sendiri



Gambar 2.4 Alat ukur titrasi otomatis pengembangan 1

Gamba 2.4 merupakan alat ukur otomatis piranti titrasi yang merupakan pengembangan pertama oleh saya dengan mengikuti konsep rancangan pertama, dan pengembangan pertama ini saya mencoba untuk memperbaiki kekurangan yang ada pada konsep rancangan alat pertama yaitu dimana alat dapat difungsikan sebagai berikut :

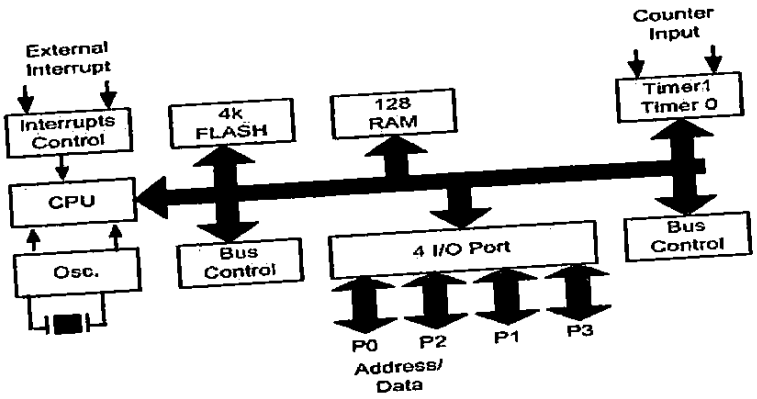
1. Sensor warna dapat bekerja dengan baik, dimana sensor warna dapat mendeteksi perubahan warna pertama kali.
2. Motor stepper dapat membuka dan meutup kran buret secara otomatis

2.2.2 Mikrokontroller AT89S51

Mikrokontroller AT89S51 adalah anggota dari keluarga *MCS-51* yang memiliki fasilitas antara lain:

- CPU dengan kapasitas *8-bit*
- *Boolean processing* dalam *bit (single bit logic)*
- On-Chip 4 Kbyte Program Memori
- Program memori dapat diperbesar hingga *64 Kbyte*
- On-Chip *128 byte* Data Memori
- Data memori dapat diperbesar hingga *64 Kbyte*
- Empat buah port masing-masing *8 bit*
- Dua buah Timer / Counter *16 bit*
- *UART full duplex*
- Lima interupsi vektor dengan 2 tingkatan prioritas

Arsitektural MCS-51 dapat dilihat pada gambar 2.5:



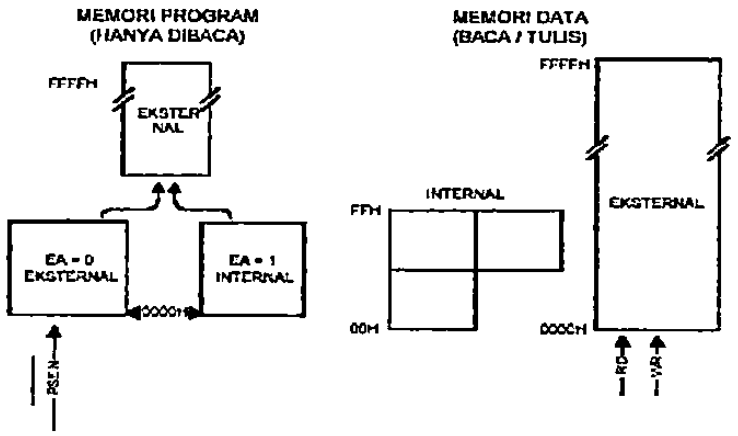
Gambar 2.5 Blok Diagram Inti AT89S51

Organisasi Memori MCS-51

Alokasi memori merupakan suatu hal yang cukup penting untuk diperhatikan pada penggunaan mikrokontroller.

1. Pemisahan antara Program Memori dan Data Memori

Semua perangkat MCS-51 memiliki ruang alamat tersendiri untuk Program Memori dan Data Memori, seperti terlihat pada gambar 2.3



Gambar 2.6 Struktur Memori MCS51

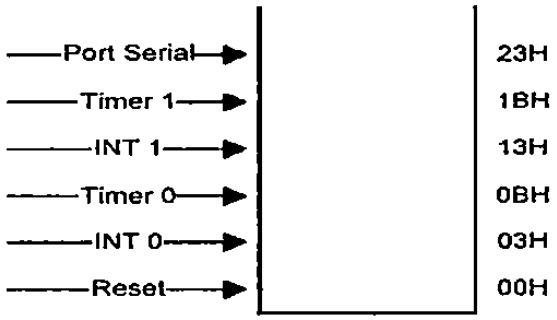
Pemisahan Program dan Data Memori ini memungkinkan pengaksesan Data Memori dengan pengalamatan 8 *bit*, sehingga dapat langsung disimpan dan dimanipulasi oleh mikrokontroller dengan kapasitas akses 8 *bit*. Namun demikian, untuk pengaksesan data memori dengan alamat 16 *bit*, harus dilakukan dengan menggunakan *register* DPTR *(Data Pointer)*.

Program memori hanya dapat dibaca saja (diletakkan pada *ROM/ EPROM*). Untuk membaca Program memori *eksternal*, mikrokontroller akan mengirim sinyal PSEN *(Program Store Enable)*. Sebagai data memori *eksternal*

dapat digunakan RAM *eksternal* (maksimum 64 *Kbyte*). Dalam pengaksesannya mikrokontroller akan mengirimkan sinyal RD (*Read,* melakukan operasi pembacaan data) dan WR (*Write*, melakukan operasi penulisan data). Bila diperlukan, program memori dan *eksternal* data dapat dikombinasikan dengan menyatukan sinyal RD dan PSEN ke dalam input gerbang AND dan menggunakan output dari gerbang tersebut sebagai sinyal *read* (baca) untuk program memori *eksternal* data.

2. Program Memori

Gambar 2.4 menunjukkan peta dari bagian bawah program memori. Setelah *reset,* CPU memulai eksekusi program dari lokasi alamat 0000H.



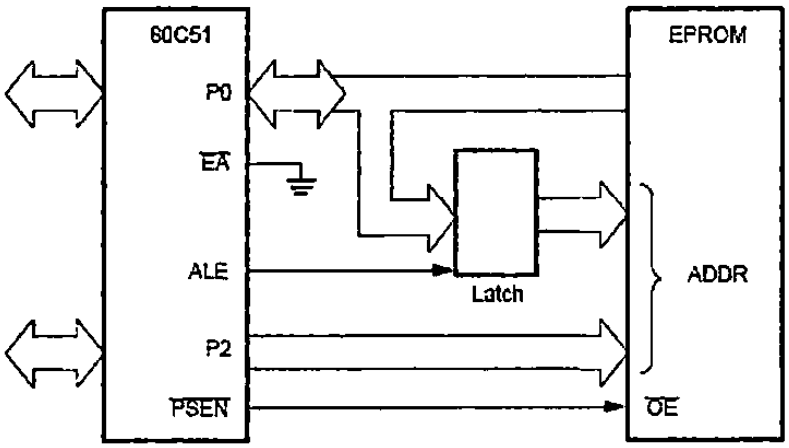
Gambar 2.7 Program memori *MCS51*

Setiap interupsi diberi lokasi tersendiri pada program memori (ketentuan tersebut dikenal sebagai *vektor* interupsi). Interupsi yang dilakukan menyebabkan CPU melompat ke alamat interupsi yang berisi sebuah rutin khusus dan kemudian mengeksekusinya. Jika program interupsi cukup pendek, maka program tersebut dapat dimulai pada alamat vektor interupsi yang ditentukan. Bila program interupsi cukup panjang, maka instruksi '*Jump*' dapat digunakan pada alamat

vektor interupsi tersebut, untuk melompat menuju lokasi awal dimana program interupsi dibuat.

Kapasitas 4K (atau 8K, atau 16K) dari program memori dapat difungsikan sebagai *on-chip* ROM *(internal)* atau sebagai ROM *eksternal.* Pemilihan ROM *internal* atau *eksternal* pada saat pembacaan oleh mikrokontroller, dilakukan dengan mengkondisikan pin EA (*External Access*) pada kondisi *high/ low* (EA = *high,* untuk ROM *internal*; EA = low, untuk ROM *eksternal*).

Sinyal pembacaan ROM *eksternal,* yaitu sinyal PSEN, digunakan untuk pengambilan semua program *eksternal*. Pada pembacaan program *internal* PSEN tidak diaktifkan.



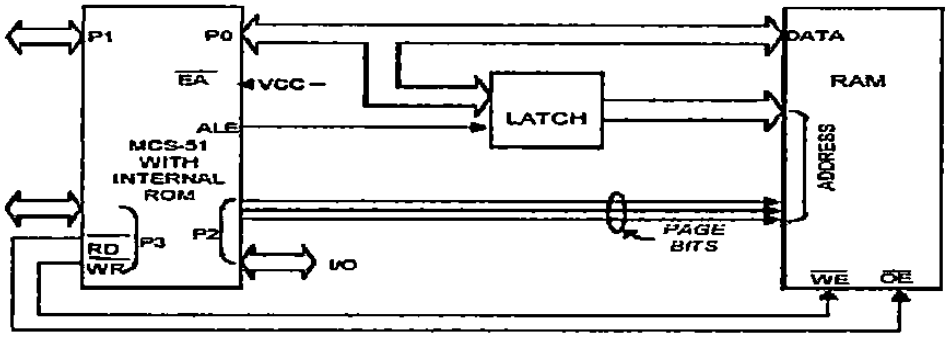
Gambar 2.8 Pengeksekusian program memori *internal*

Konfigurasi *hardware* untuk mengeksekusi program *eksternal* ditunjukkan pada Gambar 2.6, terlihat bahwa 16 jalur I/O (Port 0 dan 2) digunakan sebagai bus alamat dan bus data untuk menghubungi program memori *eksternal* (Port 0 *memultipleks* alamat dan data). Port 0 mengirim *byte* bawah *(low byte)* dari

Program *Counter* (PCL) sebagai alamat, kemudian Port 0 berada dalam kondisi mengambang menunggu datangnya *byte code* (program yang dibaca) dari program memori. Pada saat *byte* bawah dari program *counter* dianggap valid di dalam P0, maka sinyal *ALE (Address Latch Enable*) akan dikirim ke piranti *Address Latch.* Sementara itu Port 2 berisi alamat byte atas *(high byte)* dari Program *Counter* (PCH). Selanjutnya PSEN memberikan sinyal pada EPROM dan *byte code* langsung dapat diterima oleh mikrokontroller.

3. Data Memori

Data memori *internal* dan *external* (lihat gambar 2.5) dapat langsung digunakan oleh user. Jika dibutuhkan ukuran memori lebih besar dari 2 KB dapat menggunakan *eksternal* RAM dengan konfigurasi seperti yang ditunjukkan pada gambar 2.6.

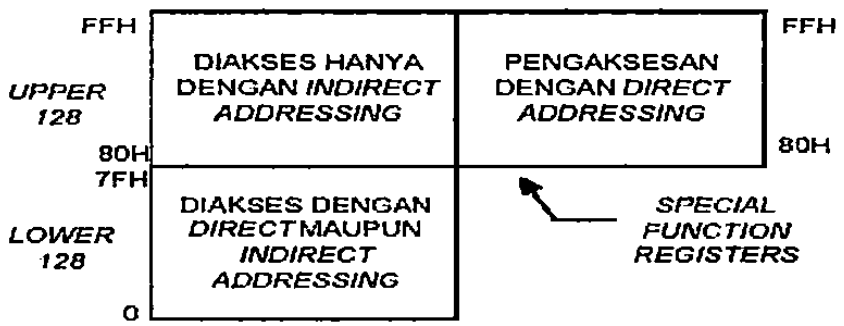


Gambar 2.9 Pengaksesan data memori *eksternal*

CPU mengeksekusi program dari ROM *internal.* Port 0 berfungsi sebagai bus alamat/ data (bersifat *multiplexer*) terhadap RAM dan 3 buah jalur dari Port 2 digunakan untuk pemilihan halaman dari RAM (RAM page). CPU mengaktifkan

sinyal RD dan WR berdasarkan kebutuhannya selama pengaksesan RAM *eksternal.*

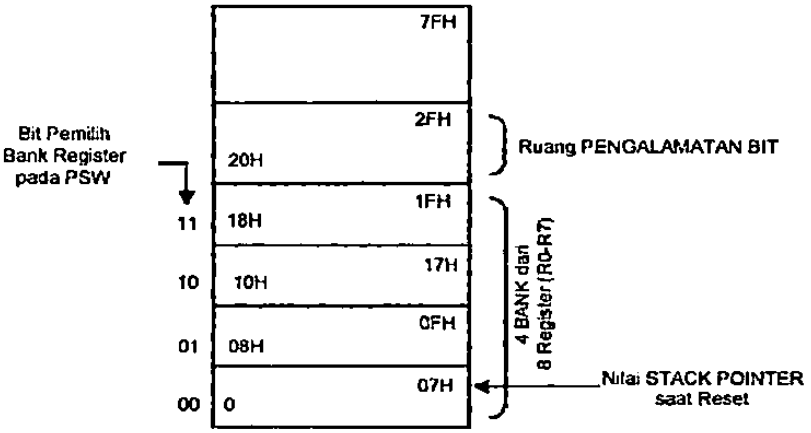
Data memori *eksternal* dapat mencapai 64KB. Pengalamatan data memori *eksternal* ini ada yang memerlukan lebar alamat cukup 1 *byte* saja atau dengan 2 *byte.* Pengalamatan dengan 1 *byte* sering digunakan asalkan satu atau lebih dari jalur I/O digunakan untuk memilih RAM *page*, seperti yang terlihat pada Gambar 2.7 Sedangkan pengalamatan 2 *byte* digunakan dengan catatan Port 2 digunakan untuk mengirim *high address byte* (*byte* alamat atas).



Gambar 2.10 Data Memori *Internal*

Data memori *internal* dipetakan seperti pada Gambar 2.7 Memori ini dibagi menjadi 3 blok, dimana secara umum dibedakan menjadi *Lower 128, Upper 128* dan ruang *Special Function Register (SFR).* Lebar alamat dari data memori *internal* selalu sebesar 1 *byte*, oleh karena itu kapasitas maksimum dari sebuah alamat data adalah 256 *byte.* Bagaimanapun, mode pengalamatan untuk *internal* RAM dapat diakomodasikan menjadi 384 *byte* dengan sedikit trik

Pengalamatan langsung yang lebih tinggi dari 7FH akan mengakses blok memori yang berbeda. Gambar 2.7 menunjukkan bagaimana *Upper* 128 dan ruang SFR menggunakan blok yang sama pada pengalamatan 80H sampai FFH, walaupun secara fisik keduanya adalah terpisah.



Gambar 2.11 Lower 128 dari RAM *internal*

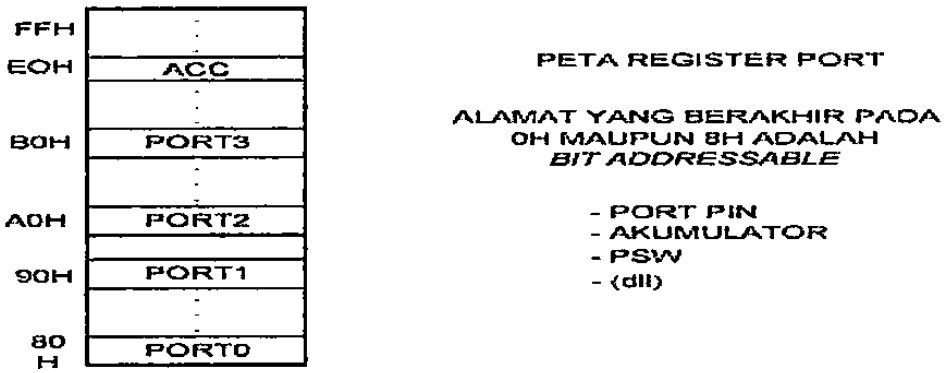
Blok *lower* 128 selalu tersedia pada semua piranti MCS-51 seperti terlihat pada Gambar 2.8. Lokasi di bawah 32 *byte* dikelompokan menjadi 4 buah *bank* dari 8 *register*. Program instruksi mengenalnya sebagai R0 sampai R7. Dua *bit* dalam PSW (*Program Status Word*) dipakai untuk memilih *bank-bank* mana yang akan digunakan. Hal ini akan menyebabkan penggunaan yang lebih efisien dari pengkodean, sebab dengan menggunakan cara ini akan didapatkan instruksi yang lebih pendek dari pada dengan menggunakan *direct addressing* (pengalamatan langsung).

16 *byte* berikutnya di atas *bank register* adalah ruang memori yang bersifat *bit-addressable* (dapat dialamati per *bit*). Pada *Instruction Set* dari MCS-51 terdapat instruksi-instruksi yang dapat mengolah *bit* tunggal dan sebanyak 128 *bit*

pada area ini dapat diakses secara langsung dengan menggunakan *Instruction Set. Bit* yang dapat diakses langsung ini adalah daerah dari 00H sampai 7FH.

Semua daerah dalam *lower* 128 dapat diakses secara *direct* maupun *indirect addressing* (pengalamatan langsung maupun tidak langsung). Sedangkan *Upper* 128 hanya dapat diakses menggunakan *indirect addressing. Upper* 128 dari RAM tidak digunakan pada AT89S51 tapi menggunakan RAM lain dengan kapasitas 256 *byte.*

Gambar 2.12 menunjukkan ruang SFR (*Special Function Register*). Dalam ruang SFR terdapat Port *latch, timer,* kontrol *peripheral* dan lain-lain. *Register-register* ini hanya dapat diakses dengan menggunakan *direct addressing.* Secara umum, seluruh keluarga MCS-51 memiliki ruang SFR yang sama dengan AT89S51 dan SFR tersebut diletakkan pada alamat yang sama.



Gambar 2.12 Ruang SFR

Sebanyak 16 alamat pada ruang SFR dapat dilakukan pengalamatan baik pengalamatan *bit* maupun *byte*. SFR yang bersifat *bit addressable* adalah alamat-

alamat yang berakhir dengan 000B. Bit yang dialamatkan pada area ini adalah mulai 80H sampai FFH.

Mode Pengalamatan *(Addressing Modes)*

Pada MCS-51 terdapat beberapa Mode Pengalamatan, yaitu :

a. *Direct Addressing*

Pada *direct addressing* instruksi yang dikeluarkan secara spesifik akan menyebutkan alamat dari *operand* yang diproses. Hanya *internal* data RAM dan SFR yang dapat diproses dengan menggunakan *direct addressing* ini.

b. *Indirect Addressing*

Pada *indirect addressing* instruksi yang dikeluarkan akan menyebutkan sebuah *register* yang berisi alamat dari *operand* yang akan diproses. Baik *internal* maupun *eksternal* RAM dapat diakses menggunakan *indirect addressing* ini.

Register alamat untuk 8 *bit* yang dapat dipakai adalah R0 dan R1 dari *bank register, atau Stack Pointer*. Pada pengalamatan 16 *bit* dapat menggunakan *register* DPTR (*Data Pointer*).

c. *Register Instruction*

Bank register berisi *register* R0 sampai R7 yang dapat diakses dengan instruksi-instruksi tertentu di mana hanya akan melibatkan 3 *bit register* spesifik yang berisi *opcode* dari instruksi. Instruksi yang mengakses *register* dengan cara ini akan lebih efisien, karena mode ini akan menghilangkan

bagian *byte* alamat. Saat instruksi ini dieksekusi, satu dari delapan *register* dari *bank register* akan diakses.

d. Register-Specific Instructions

Beberapa instruksi secara spesifik menunjuk sebuah *register*, sebagai contoh, beberapa instruksi selalu mengoperasikan *akumulator*, atau *Data Pointer*, jadi tidak ada bagian *byte* alamat yang dibutuhkan pada instruksi ini. *Opcode* dari instruksi inilah yang langsung menunjuk *register* yang dibutuhkan.

e. Immediate Constants

Sebuah konstanta dapat mengikuti *opcode* pada Program Memori, sebagai contoh:

MOV A,#100

Perintah di atas akan mengisi *Akumulator* dengan sebuah konstanta dengan nilai desimal 100, sedangkan untuk nilai Hexadesimalnya adalah 64H.

f. Indexed Addressing

Hanya program memori yang dapat diakses secara *indexed addressing* dan operasi yang dilakukan hanyalah membaca. Mode pengalamatan ini dimaksudkan untuk membaca tabel pada program memori. Sebuah *register* berbasis 16 *bit* (dapat DPTR atau Program *Counter*) menunjuk basis dari tabel dan akumulator akan diset dengan *entry* dari tabel. Alamat dari tabel *entry* pada program memori dibentuk dengan menambahkan data dari akumulator ke

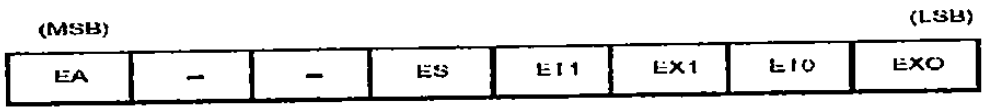
Tipe lain dari *indexed addressing* dapat dilihat dari instruksi "*Jump*". Pada kasus ini alamat tujuan yang dituju oleh instruksi *Jump* akan diproses sebagai penjumlahan terhadap basis *pointer* dan data akumulator.

Struktur Interupsi

Inti dari AT89S51 menyediakan 5 buah interupsi, yaitu dua buah interupsi *eksternal,* dua buah interupsi *timer* dan sebuah *serial port* interupsi. Di bawah akan dijelaskan lebih mendalam bagaimana cara memanfaatkan interupsi-interupsi ini.

a. Interrupt Enable

Tiap interupsi dapat secara individu dimatikan ataupun diaktifkan dengan menset *bit* yang sesuai pada SFR yang disebut IE *(Interrupt Enable).* Perlu diingat juga bahwa apabila *bit* EA pada IE diberi nilai 0, maka seluruh interupsi yang ada tak dapat digunakan seperti yang terlihat pada gambar 2.10



Gambar 2.13 *Interupt Enable*

Enable bit =1 , *enable* semua interupsi

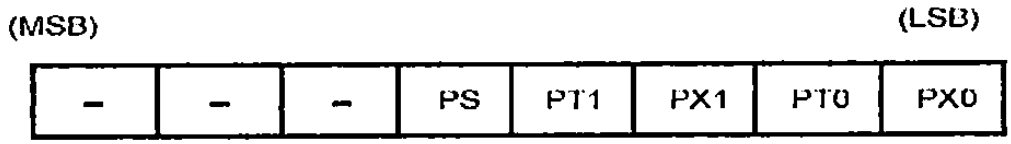
Tabel 2.3. IE *(Interrupt Enable)*

Simbol	Posisi	Fungsi
EA	IE.7	semua interupsi disable. Jika EA = 0, tidak ada interupsi yang dapat aktif. Jika EA=1, maka tiap interupsi dapat diaktifkan dengan menset bit-bit yang berpadanan.
-	IE.6	tidak digunakan
-	IE.5	tidak digunakan
ES	IE.4	Serial Port Interrupt enable bit
ET1	IE.3	Timer 1 Overflow Interrupt enable bit
EX1	IE.2	Interupsi Eksternal 1 enable bit
ET0	IE.1	Timer 0 Overflow Interrupt enable bit
EX0	IE.0	Interupsi Eksternal 0 enable bit

b. Interupsi Prioritas (*Interrupt Priority*)

Tiap interupsi juga dapat secara individu diprogram untuk satu atau dua prioritas *level* dengan menset *bit* pada SFR yang bernama IP *(Interrupt Priority).* Sebuah interupsi yang memiliki *level* Interupsi Prioritas yang rendah *(low priority interrupt)* dapat diganggu / di interupsi oleh interupsi yang memiliki prioritas yang tinggi *(high priority interrupt),* tapi tidak dapat diganggu oleh interupsi lain yang juga merupakan *low priority interrupt.* Sebuah *high priority interrupt* tidak dapat diganggu/di interupsi oleh interupsi manapun.

Apabila dua buah interupsi diterima oleh kontroller secara bersamaan maka interupsi yang memiliki *level* prioritas yang lebih tinggilah yang akan dilaksanakan, namun jika kedua interupsi tersebut memiliki *level* interupsi yang sama, maka urutan *internal polling* yang akan menentukan interupsi manakah yang akan dilayani. Jadi di dalam tiap *level* prioritas terdapat struktur prioritas kedua yang ditentukan oleh urutan *polling.* Seperti yang dilihat pada gambar 2.14



Gambar 2.14 *Interupt Priority*

Priority bit = 1 dianggap *high priority*

Priority bit = 0 dianggap *low priority*

Tabel 2.4. IP *(Interrupt Priority)*

Simbol	Position	Function
-	IP.7	Disediakan
-	IP.6	Disediakan
-	IP.5	Disediakan
PS	IP.4	Serial Port Interrupt Priority bit
PT1	IP.3	Timer 1 Interrupt priority bit
PX1	IP.2	External Interrupt 1 priority bit
PT0	IP.1	Timer 0 Interrupt priority bit
PX0	IP.0	External Interrupt 0 priority bit

Timer/Counter (T/C)

Mikrokontroller AT89S51 mempunyai dua buah *register Timer/Counter* 16 *bit, Timer* 0 dan *Timer* 1. Pada saat sebagai *Timer, register* naik satu (*increment*) setiap satu *cycle*. Jika digunakan *osilator* 12 Mhz, maka satu *cycle* sama dengan 1/12 frekuensi *osilator* = 1_s. Pada saat sebagai *counter, register* naik satu (*increment*) pada saat transisi 1 ke 0 dari *input eksternal*, T0 atau T1.

Apabila periode tertentu telah dilampaui, *Timer/Counter* segera

waktu telah selesai dilaksanakan. Periode waktu *Timer/Counter* secara umum ditentukan oleh persamaan berikut:

a. Sebagai T/C 8 *bit*

$$T = (255 - TLx) * 1\_s$$

Dimana TLx adalah isi *register* TL0 atau TL1.

b. Sebagai T/C 16 *bit*

$$T = (65535 - THx\ TLx) * 1\_s$$

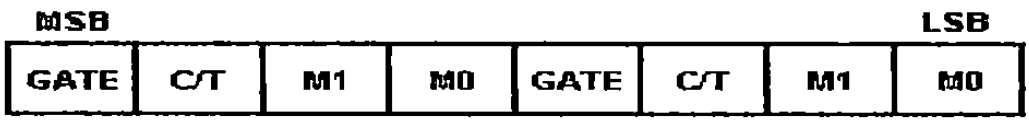
THx = isi register TH0 atau TH1

TLx = isi register TL0 atau TL1

Pengontrol kerja *Timer/Counter* ada pada *register timer control* (TCON). Adapun definisi dari *bit-bit* pada *timer control* adalah sebagai berikut:

Gambar 2.15 Definisi tiap *bit* pada *timer* kontrol

Pengontrol pemilihan mode operasi *Timer/Counter* ada pada *register timer mode* (TMOD). Definisi *bit-bitnya* dapat di perhatikan pada gambar 2.16



Gambar 2.16 Definisi *bit* pemilihan mode operasi *Timer/Counter*

Keterangan

GATE Gate control set. Timer/Counter "x" akan aktif jika *pin* "INTx" *high* dan kondisi *pin* "TRx" sedang *set. Gate control clear. Timer*"x" akan aktif jika "TRx" *set.*

C / T *Timer/Counter Selector. Clear* untuk mode *Timer* (*input* dari *internal clock*) dan *set* untuk mode *Counter* (*input* dari pin "Tx")

M1 *Bit* untuk memilih *mode timer/counter*

M0 *Bit* untuk memilih *mode timer/counter*

x = 0 atau 1.

Kombinasi M0 dan M1 adalah sebagai berikut:

Tabel 2.5. Kombinasi M0 dan M1

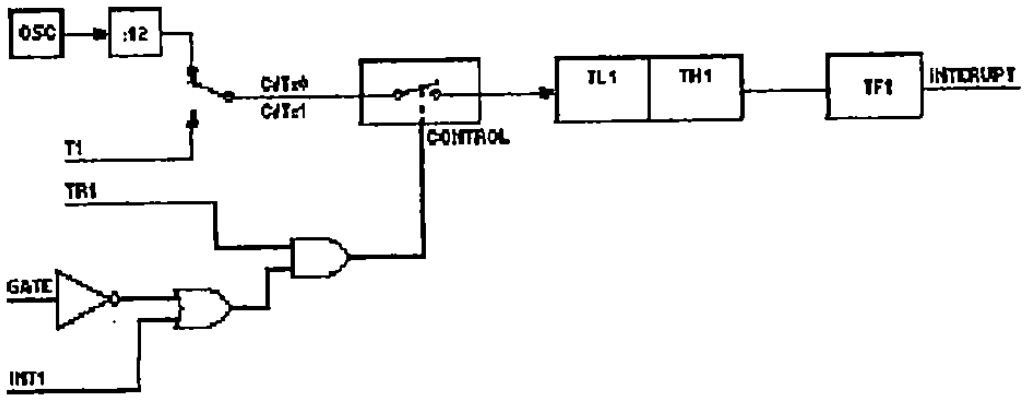
1	1	3	(*Timer* 0) TL0 adalah T/C 8 *bit* yang dikontrol oleh kontrol *bit* standar *Timer* 0. TH0 adalah *timer* 8 *bit* dan dikontrol oleh *bit Timer* 1 (*Timer* 1) *Timer-Counter* 1 tidak aktif

M1	M0	Mode	Operasi
0	0	0	*Timer* 13 *bit*
0	1	1	*Timer/Counter* 16 *bit*
1	0	2	*Timer/Counter auto reload* 8 *bit* (pengisian otomatis)

Mode 0

Pada mode ini *timer* bekerja sebagai *timer* 13 *bit* yang terdiri dari *counter* 8-*bit* dengan pembagi 32 (pembagi 5 *bit*). Gambar 2.14 menunjukkan diagram *timer* yang bekerja pada mode 0. Setelah perhitungan selesai, mikrokontroller akan mengeset *Timer Interrupt Flag* (TF1). Dengan membuat *GATE* = 1, *timer* dapat dikontrol oleh *input* dari luar (INT1), untuk fasilitas pengukuran lebar *pulsa.*

Register 13 *bit* yang digunakan terdiri dari 8 *bit* dari TH1 dan 5 *bit* bawah dari TL1 (*bit* 6,7,8 tidak digunakan).Mengeset TR1 tidak akan menghapus isi *register.* Operasi pada mode 0 untuk *Timer* 0 dan *Timer* 1 adalah sama.

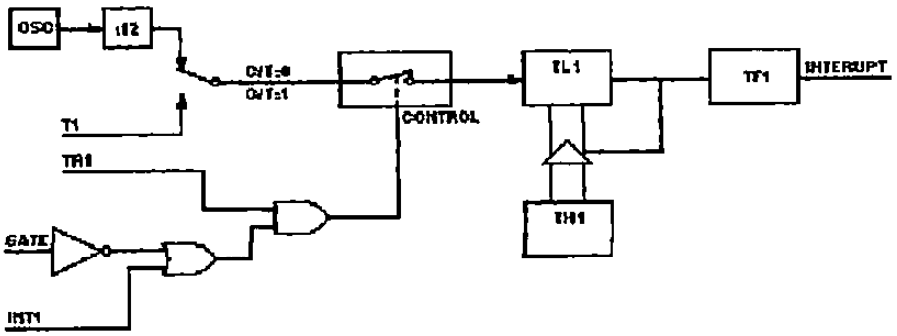


Mode 1

Mode 1 sama dengan mode 0 kecuali *register timer* akan bekerja dalam mode 16 *bit*.

Mode 2

Mode 2 menyusun *register timer* sebagai 8 *bit counter* (TL1) dengan kemampuan *reload* otomatis seperti yang ditunjukkan pada gambar 2.18. *Overflow* dari TL1 tidak hanya menset TF1 tetapi juga mengisi TL1 dengan isi TH1 yang diisi sebelumnya oleh *software*. Pengisian ulang ini tidak mengubah nilai TH1. Seperti yang terlihat pada gambar 2.18



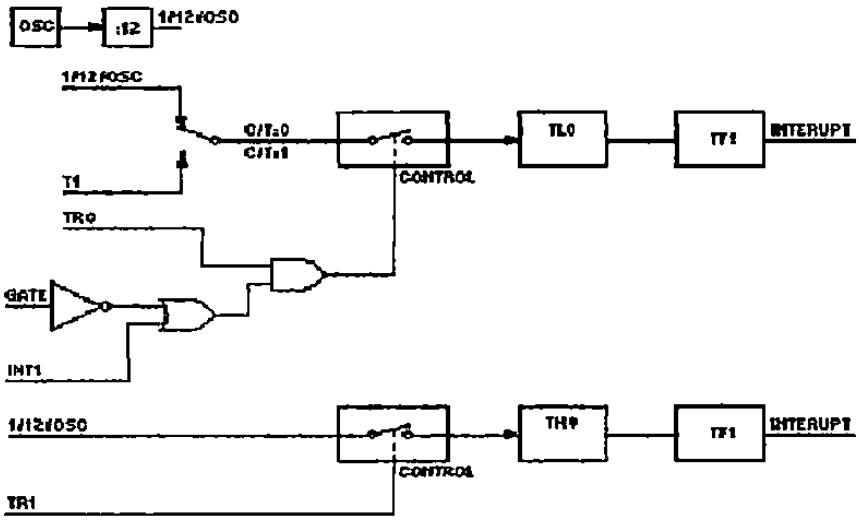
Gambar 2.18 *Timer/Counter* 1 bekerja dalam mode 2, sebagai T/C 8-*bit auto-reload*

Mode 3

Dalam operasi mode 3 *Timer* 1 akan berhenti, hitungan yang sedang berjalan dipegang. Efeknya sama seperti mengatur TR1 = 0. *Timer* 0 dalam mode 3 membuat TL0 dan TH0 sebagai dua *counter* terpisah. TL0 menggunakan

sebagai *timer* dan mengambil alih penggunaan TR1 dan TF1 dari *Timer* 1 dan sekarang TH0 mengontrol interupsi *Timer* 1.

Mode 3 diperlukan untuk aplikasi yang membutuhkan *ekstra Timer/Counter* 8 *bit*. Dengan *timer* 0 dalam mode 3, mikrokontroller AT89S51 seperti memiliki 3 T/C. Saat *Timer* 0 dalam mode 3, Timer 1 dapat dihidupkan atau dimatikan, atau dapat digunakan oleh *port serial* sebagai pembangkit *baudrate* dalam aplikasi *komunikasi serial*. Seperti yang terlihat pada gambar 2.19



Gambar 2.19 *Timer/Counter* 1 bekerja dalam mode 3, sebagai 2 T/C 8-bit

Menset *Timer/Counter 0 atau Timer/Counter 1*

Sebagai contoh untuk menjalankan *Timer/Counter* 0 dalam mode 8-*bit timer auto reload* dengan kontrol *internal* maka nilai yang harus dimasukkan ke *register* TMOD adalah 02H. Untuk menjalankan *Timer/Counter* 0 dalam mode 16 *bit counter* dengan kontrol *eksternal* maka nilai yang harus dimasukkan ke

Nilai yang dimasukkan ke *register* TMOD diatas hanya akan beroperasi pada satu *timer* bila diinginkan untuk menjalankan *timer* 1 dan *timer* 0 secara bersamaan, maka nilai diatas di OR kan dengan nilai yang diperlukan untuk mengoperasikan *Timer/Counter* 1.

Sebagai contoh T/C 1 pada mode *timer* 13-*bit* kontrol *internal* dan T/C 0 pada mode 8-*bit auto-reload* dengan kontrol *internal* maka nilai yang harus dimasukkan ke TMOD adalah 02H. Dalam mode kontrol *internal*, operasi *timer/counter* dihidup-matikan dengan mengeset *bit* "TRx" *(kontrol software)*. Pada kontrol *eksternal*, *timer* dihidup-matikan dengan memberikan logika 0 pada pin INT 0 (kontrol *hardware*).

Komunikasi *Serial*

Mikrokontroller AT89S51 dilengkapi dengan *port* komunikasi *serial full duplex*, yang berarti mikrokontroller ini dapat mengirimkan dan menerima data secara bersamaan. Data yang diterima pada komunikasi *serial* diberi *buffer* yang berarti kita dapat menerima *byte* yang kedua sebelum *byte* yang pertama dibaca pada *register* penerima *(receive register)*, tetapi bila *byte* kedua telah selesai diterima dan *byte* pertama belum diolah, maka salah satu *byte* akan hilang. *Register* penerima dan pengirim pada *port serial*, keduanya diakses pada SBUF *(Serial Buffer)*. Mengisi SBUF berarti mengisi *register* pengirim *(transmit register)* dan membaca SBUF berarti mengakses *register* penerima.

Register status dan kontrol *port serial* pada *Special Function Register*

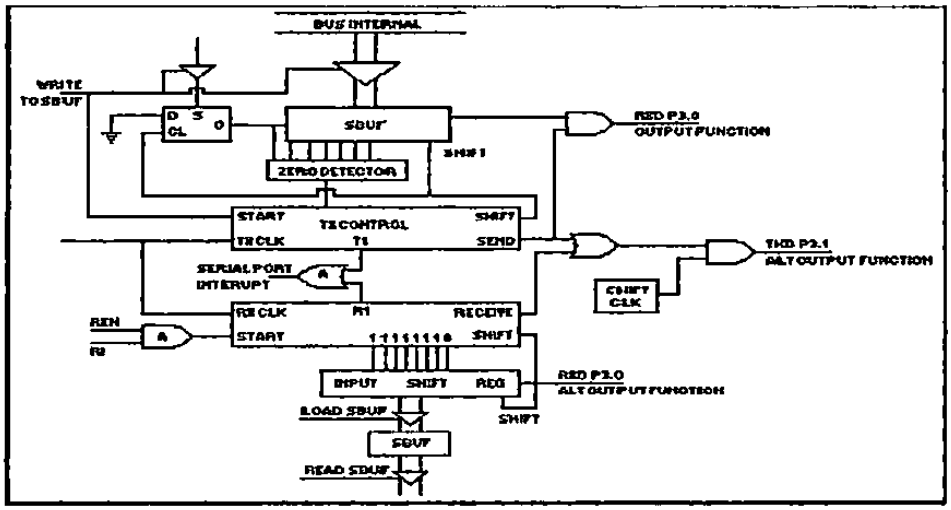
tidak hanya berisi *bit* untuk pengaturan mode saja, melainkan *bit* ke-9 untuk pengiriman dan penerimaan data (TB8 dan RB8) ada di *register* ini, serta *bit* interupsi *port serial* (T1 dan R1).

Gambar 2.20 SCON pada *Special Function Register*

Mode 0

Mode 0 adalah mode komunikasi *sinkron* dimana ada *clock* yang mensinkronkan antara waktu pengiriman dan penerimaan *bit* data. Mode ini *kompatible* dengan komunikasi *sinkron* SPI dari Motorolla atau *Microwire* dari *National Semiconductor.*

Mode 0 penggunaan data *serial* masuk dan keluar melalui *pin* RxD sedangkan *pin* TxD mengeluarkan *shift clock.* Data 8 *bit* dikirim / diterima dengan urutan yang pertama adalah LSB. *Baudrate* tetap yaitu pada 1/12 frekuensi



Gambar 2.21 *Serial Port* pada Mode 0

Mode 1

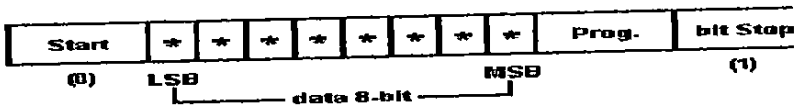
Sepuluh *bit* dikirim melalui TxD (P3.1) atau diterima melalui RxD (P3.0). Format data yang diterima pada mode 1 sebagai berikut:

Gambar 2.22 Format data Mode 1 yang diterima

Start bit selalu 0 dan *stop bit* selalu 1. Pada saat *stop bit* diterima, *bit* ini masuk ke RB8 pada *register* kontrol *serial* (SCON). *Baudrate* dapat diubah-ubah.

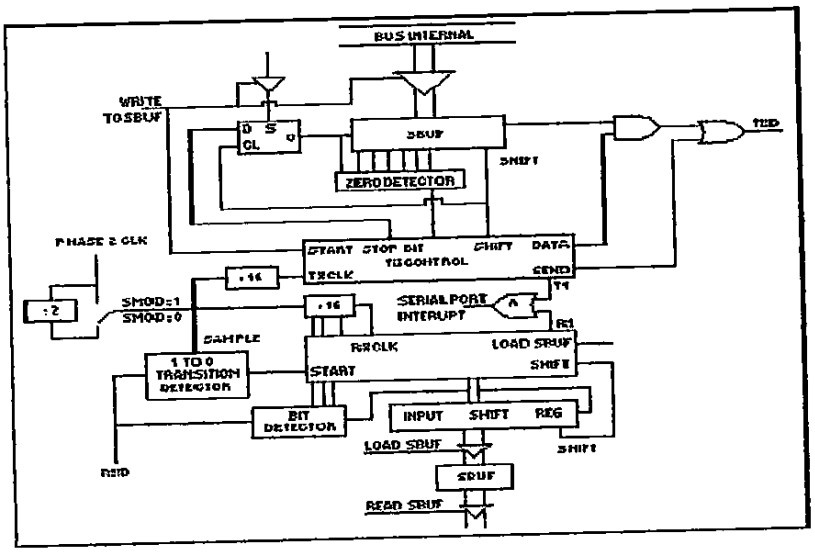
Mode 2

Sebelas *bit* dikirim melalui TxD atau diterima melalui RxD. Format datanya dapat dilihat pada gambar 2.23



Gambar 2.23 Format Data 11 bit Mode 2

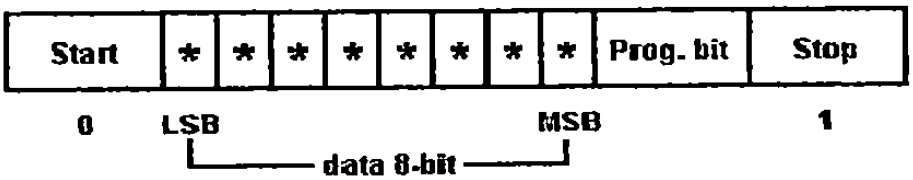
Start bit selalu 0 dan *Stop bit* selalu 1. Pada saat pengiriman data, *bit* data ke-9 (TB8 dalam SCON) dapat diprogram agar bernilai 0 atau 1. Sebagai contoh, *parity bit* (P di dalam PSW) dapat dipindahkan / *dicopy* ke TB8. Pada saat menerima data, *bit* ke-9 akan berada pada RB8 di SCON, ketika *stop bit* diabaikan. Fungsi dari *bit* ke 9 sangat penting pada aplikasi komunikasi *multi processor*. *Baudrate* dapat diprogram menjadi 1/32 atau 1/64 dari frekuensi *osilator*. Komunikasi *serial* pada mode 2 digambarkan pada gambar 2.24



Gambar 2.24 *Serial Port* pada Mode 2

Mode 3

Sebelas *bit* data dikirim melalui TxD atau diterima melalui RxD. Format datanya adalah sebagai berikut:



Gambar 2.25 Format Data 11 bit Mode 3

Secara *default* Mode 3 sama dengan mode 2 kecuali dalam hal *baudrate.* Dalam mode 3, *baudrate* adalah *variabel. Inisialisasi transmisi* data serial pada keempat mode pengiriman ditandai oleh instruksi yang menggunakan SBUF sebagai *register* tujuan. Penerimaan data dalam mode 0 mempunyai syarat dimana R1 = 0 dan REN = 1. Sedangkan dalam mode lain, ditentukan oleh *start bit* yang datang jika REN=1

Baudrate

Saat *Timer* 1 digunakan untuk menghasilkan *baudrate*, *baudrate* dalam mode 1 dan 3 ditentukan oleh *Timer* 1 *overflow rate* dan nilai dari SMOD.

Untuk lebih jelasnya lihat rumus di bawah ini :

$$\frac{2^{SMOD}}{}$$

Interupsi *Timer* 1 harus dinonaktifkan untuk aplikasi di atas. *Timer/Counter* ini sendiri dapat dikonfigurasikan dalam operasi sebagai *timer* atau *counter* dalam mode yang ada. Tapi pada kebanyakan aplikasi, *timer/counter* dikonfigurasikan sebagai *timer* pada mode *auto reload* (*high nibble* dalam TMOD=0010B). Pada kasus ini, rumus untuk *baudrate* adalah sebagai berikut:

$$Baut\ Rate\ Mode\ 1,3 = \frac{2^{SMOD}}{32} \times \frac{frekuensi\ osilator}{12 \times [256 - (THI)]}$$

Apabila *baudrate* yang ingin digunakan diketahui, maka nilai TH1 yang harus diisikan (*reload*) dapat dicari dengan persamaan berikut:

$$THI = \frac{256 - (K \times Frekuensi\ osilator)}{(384 \times Baut\ Rare)}$$

Nilai TH1 harus dalam bentuk *integer*. Pembulatan nilai TH1 pada nilai *integer* terdekat tidak akan menghasilkan *baudrate* yang dikehendaki. Dalam hal ini pemakai dapat mengganti *frekuensi* kristal. Sebagai *standar* komunikasi *asinkron baudrate* yang sering digunakan adalah: 1200, 2400, 9600, 19200 bps. Untuk mencapai ketelitian *baudrate* maka biasanya digunakan kristal dengan *frekuensi* 11.059.200 Hz.

Timer 1 dapat menghasilkan *baudrate* yang sangat rendah dengan mengabaikan *Timer 1 interrupt enable* dan mengkonfigurasikan *timer* sebagai 16 *bit timer* (*high nibble* dari TMOD = 0001B) dan menggunakan interupsi *Timer* 1 untuk melakukan *16 bit software reload*

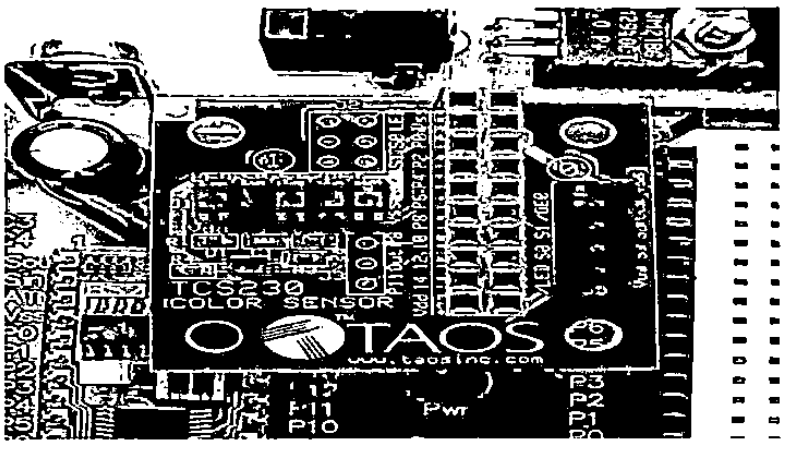
Baudrate pada mode 2 tergantung dari isi *bit* SMOD dalam *Special Function Register*, PCON. Jika SMOD = 0 (nilai pada saat *reset*) maka *baudrate*nya 1/64 dari *frekuensi osilator*. Jika SMOD = 1 , *baudrate*nya = 1/32 *frekuensi osilator.*

$$Baut\ Rate\ Mode\ 2 = \frac{2^{SMOD}}{64} \times frekuensi\ osilator$$

2.2.3 Sensor Warna TCS230 dengan Adapter AppMod

Seperangkat modul sensor warna TCS230 terdiri dari dua cetakan papan *circuit* dan sebuah panjang pendek kabel pita. Kedua kabel pita ini terhubung dalam satu paket.

Papan *adapter* AppMod (memiliki satu konektor *pass-thru*) dapat dihubungkan secara langsung ke konektor *AppMod Parallax*, sama seperti pada sebuah Papan Edukasi. Dimana konektor tersebut harus ditancapkan sesuai dengan urutan *pin* pada papan *adapter* dan hal ini juga sama pada Papan Edukasi.



Gambar 2.26 Papan AppMod diinstall secara benar

Modul sensor dihubungkan ke adapter AppMod menggunakan kabel pita berjajar 6 (150 mm). kabel ini akan dikunci dengan sebuah blok konektor satu lubang untuk setiap kabel. Hal ini mengacu pada urutan *pin* 2x5 *header* di kedua papan. Demikian selanjutnya sampai 9 pin dihubungkan, kabel tidak boleh dihubungkan secara sembarangan, dan tiap ujung-ujungnya harus sesuai apabila dihubungkan dengan *boardnya.*

Adapter AppMod memiliki kedua *header* sebagai tempat kabel ditancapkan. Mereka dialamati sebagai 0 (J1) dan 1 (J1), dengan alamat tertera pada papan. Hal ini dimaksudkan untuk membimbing dan mengakomodasikan agar dua modul dapat digabungkan menjadi satu, jika mau, atau untuk penggunaan lain seperti pada saat digunakan dengan *BOE-Bot.*

Memulai

Salin program berikut ini ke dalam *BASIC Stamp Editor*

```
'{$STAMP BS2}
EN          con         1
A0          con         2
S0          con         3
S1          con         4
S2          con         5
S3          con         6
nLED        con         7
OUT         con         8

pRED        con         12
pGREEN      con         6
pBLUE       con         5

RED         var         word
GREEN       var         word
BLUE        var         word

Start:      low         A0
            high        S0
            high        S1
            low         nLED
            high        EN

MainLp:     gosub       Color
            debug       "R", dec3 RED
            debug       " G", dec3 GREEN
            debug       " B", dec3 BLUE
            debug       cr
            goto        MainLp

Color:      low         S2
            low         S3
            count       OUT, pRED, RED
            high        S3
            count       OUT, pBLUE, BLUE
            high        S2
            count       OUT, pGREEN, GREEN
            return

            end
```

Gambar 2.27 Program *Stamp BS2*

Penggunaan modul sensor tertancap dan sebuah sebuah kabel *serial* terhubung dari PC ke Papan Edukasi, nyalakan Papan Edukasi, dan jalankan program yang telah kamu masukkan. Matikan tegangan, hubungkan kabel pita ke J5 modul sensor *adapter* AppMod, dan nyalakan tegangan kembali. Dua hal yang akan terjadi.

1) LED putih pada modul sensor akan menyala
2) Kamu akan melihat keluaran melalui *terminal* separti berikut:

R123 G065 B200

R120 G060 B187 Dst

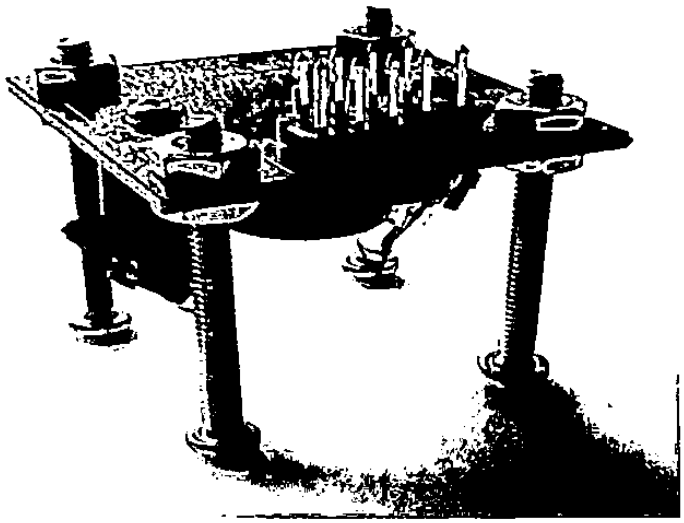
Angka-angka ini mewakilkan nilai untuk komponen berwarna merah, hijau, dan biru dari sensor warna yang terlihat. Sekarang, ambil modul sensor dan balikkan pada sebuah kertas berwarna. Gerakkan modul tersebut ke atas dan ke bawah ketika kamu melihat adanya keluaran. Nilainya akan bertambah naik, sampai maksimum, kemudian akan berkurang lagi. Tandai modul ketika berada dipuncak. Pada saat tersebut papan *circuit* akan berada kira-kira satu inci di atas permukaan kertas. Tandai juga dimana pada point ini titik dari dua LED digabung menjadi satu. Posisi ini yang akan selalu terjadi pada saat pembacaan warna dilakukan.

Kamu dapat dengan mudah membuat kaki-kaki kecil untuk modul sensor menggunakan komponen-komponen yang didapat dari toko *hardware*:

4 ea. 4-40 x 1¼" (or M3 x 35mm) *pan-head machine screws*,

8 ea. 4-40 (or M3) hex n

Pasang kaki-kaki tersebut seperti ditunjukkan:



Gambar 2.28 Mencocokkan *stand made* dari mesin *screw*

Tinggi yang kamu inginkan dapat diset dengan melonggarkan baut, menurunkan atau menaikkan tiap kaki-kaki *mur*, kemudian kencangkan kembali bautnya.

Alternatif lain, ***TCS230 Deluxe Mounting Kit*** terdiri dari beberapa perlengkapan, *bracket* dan pengatur jarak untuk lubang pada modul sensor untuk setiap tujuan yang berbeda *(kedua BOE-Bot on dan off)*

Hardware

TCS230 memiliki urutan *photodetektor* terdiri dari sebuah lampu berwarna merah,hijau, atau *filter* berwarna biru, atau tidak ada filter *(clear)*. Filter dari setiap warna didistribusikan menggunakan sistem *array* untuk memperkecil bias antar warna. Didalam modul ini sebuah *osilator* yang menghasilkan gelombang kotak yang frekuensinya proporsional dengan intensitas dari cahaya warna yang berbeda. Akan ada satu keluaran tiga kondisi stabil dari *osilator,* dan warna yang akan dibaca dipilih menggunakan dua jalur alamat: S2 dan S3. Sebagai tambahan

dimungkinkan untuk memprogram nilai bagi dari *osilator* menggunakan dua jalur tambahan, **SO** dan **S1**. Berikut ini setting dari jalur *control* dan fungsi-fungsinya:

Tabel 2.6. Jalur *control* dan fungsi-fungsinya

S0	S1	Divide	S2	S3	Color
0	0	Pwr.DOWN	0	0	Red
0	1	1:50	0	1	BLUE
1	0	1:5	1	0	Clear
1	1	1:1	1	1	Green

Sebagian besar jalur kontrol dan data IC TCS230 dapat diakses langsung menggunakan *BASIC Stamp port* melalui papan *adapter* AppMod. Salah satu pengecualian adalah jalur **/OE** (keluaran *active-low enable*), yang dikodekan untuk setiap kabel *header* (J1 = **Unit 1**, dan J5 = **Unit 0**) menggunakan **A0** dan **EN** dari *BASIC Stamp*. Sebagai tambahan, setiap jalur **/OE** modul sensor selanjutnya menjadi gerbang LED *enable*, jadi LED pada papan tidak dapat hidup jika keluarannya *disable*. Logika tambahan ini menyimpulkan dua hal:

1) Mencegah terjadinya *crash* pada keluaran jika dua modul sensor dihubungkan pada adapter AppMod
2) Mencegah terjadinya aliran arus masuk yang akan menyebabkan terjadinya dua LED menyala secara bersamaan

Urutan *port BASIC Stamp* ditunjukkan pada table berikut:

Tabel 2.7. Urutan port BASIC Stamp

Signal	Default BS 2 pin	Option
EN	P1	
A0	P2	

S0	P3	
S1	P4	N/C
S2	P5	N/C
S3	P6	
/LED	P7	N/C
OUT	P8	P11

Urutan *table "option"* di atas diperuntukkan J2 dan J3 pada papan AppMod. Dengan memotong jalur pada J2 untuk **S0** dan **S1**. Sinyal-sinyal ini akan terdefault *"high"*, atau mereka diikat pada *level* yang berbeda J1 pada modul sensor. Memotong jalur **/LED** akan menkondisikan LED "*on*" kapanpun modul sensor dipilih. Memotong jalur ini akan membebaskan jalur **P3**, **P4**, dan **P7** untuk tujuan yang lain. Sebagai tambahan, jika **P8** digunakan, keluaran dapat dialihkan ke **P11** dengan memotong jalur pada J3 dan menj*umper* kaki tengah dengan kaki terluar.

Decoding logic ditunjukkan sebagai berikut:

Tabel 2.8. *Decoding Logic*

A0	EN	/LED	/0E0	LED0	/0E1	LED1
0	0	0/1	disable	Off	disable	off
0	1	0	enable	On	disable	off
0	1	1	enable	Off	disable	off
1	0	0/1	disable	Off	disable	off
1	1	0	disable	Off	enable	on
1	1	1	disable	Off	enable	off

Modul sensor dapat juga beroperasi tanpa *adapter* AppMod. Dalam hal ini, sinyal **/OE** dari TCS230 dikontrol langsung menggunakan alamat logika yang ditunjukkan di atas. LED akan tetap "*on*" jika kondisi **/OE** berada pada *aktif low*

bagaimanapun juga. Walaupun IC TCS230 dapat beroperasi pada tegangan 2.7 sampai 5.5V, modul **Vdd** harus berada pada +5.0VDC.

Sebagai tambahan, modul sensor dapat beroperasi hanya menggunakan kaki luar pada J2 (**Vdd,S3, S2,Out, Vss**). Pada saat baris mengambang, **/OE** berada pada kondisi *low,* sama dengan **/LED. S0** dan **S1** pada posisi tetapi tiap-tiapnya dapat diikat *low* secara individu pada J2. Hal ini untuk menfasilitasi operasi dengan syarat sinyal minimum dan mengizinkan koneksi melalui sebuah kabel flesibel tipis (contoh DigiKey part number A9BBG-0506F-ND).

Data Optik

Data optik didukung lensa 5.3 mm dan sebuah pengatur jarak (25 mm) dari permukaan terdepan pada papan *circuit*, modul sensor dapat *"see"* sebuah area kotak 5/32 berkisar sekitar (4 mm) pada satu sisi. Ini berarti bahwa variasi warna pada kecakupan tersebut akan terdeteksi oleh TCS230.

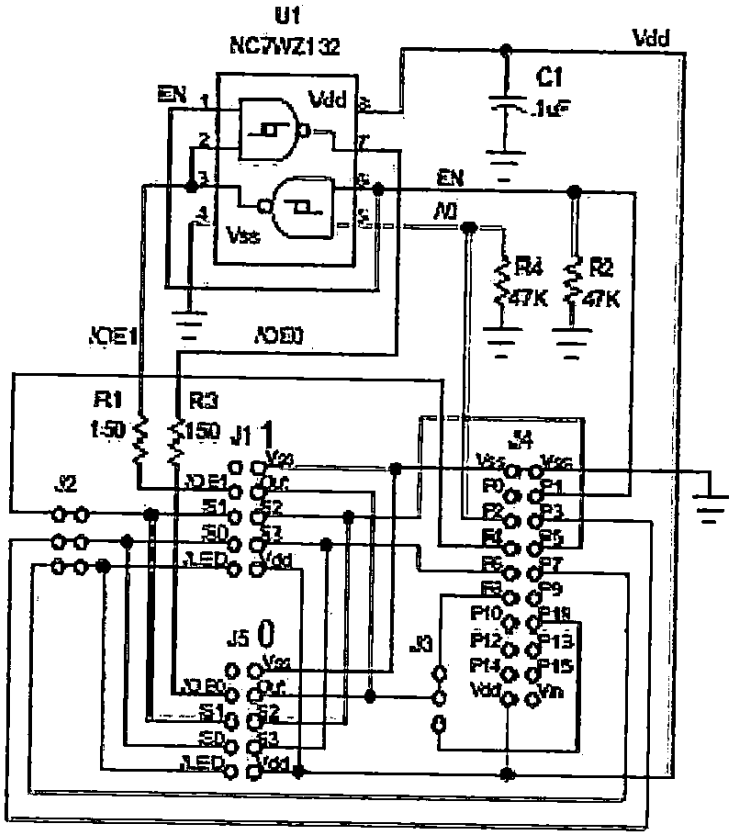
Respon *spectral* dari setiap sistem sensor warna adalah sebuah fungsi yang tidak hanya direspon oleh sensor, tetapi juga dari sistem optik dan pencahayaan.

Software

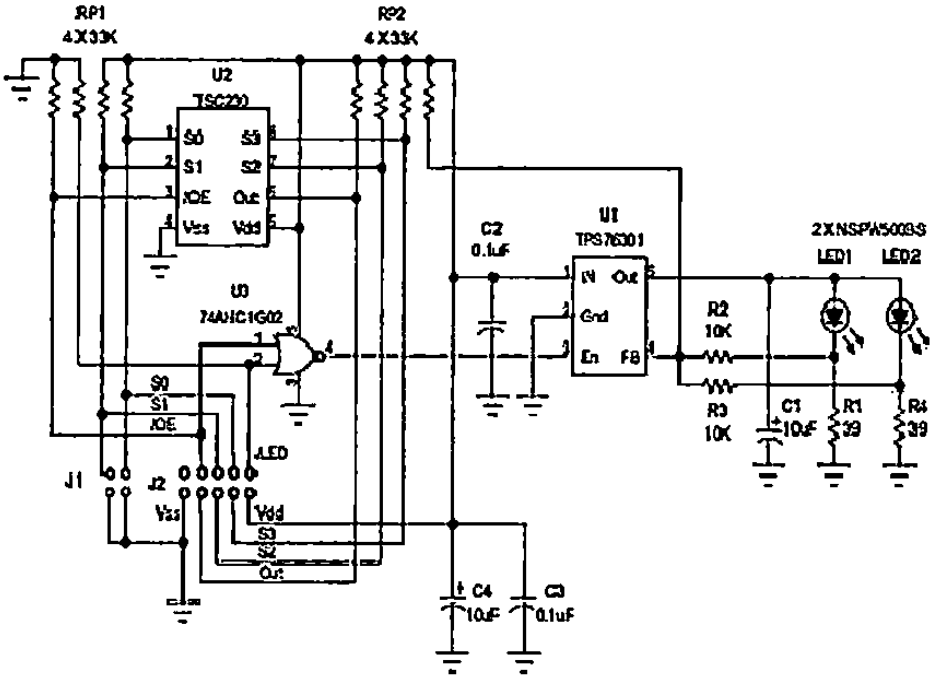
Ketika bekerja dengan *BASIC Stamp*, *frekuensi* keluaran TCS230 dapat dibaca dengan menghitung kondisi *Stamp*, seperti ditunjukkan kode contoh pada halaman-halaman depan dari *sheet* ini. Pada contoh ini, **S0** dan **S1** diset "*high*", untuk mengaktifkan rata keluaran cepat dari TCS230. Bagaimanpun juga, nilai maksimum dari rata keluaran ini adalah 600 KHz atau lebih maksimum dari

intensitas cahaya, dan lebih cepat dari yang dapat dihitung oleh BS2. Jadi **S0** dan **S1** harus diset berdasarkan intensitas cahaya maksimum yang diinginkan dari subjek yang akan di periksa.

Dalam hal untuk menghemat daya - khusus ketika baterai dioperasikan kamu dapat mengoperasikan LED hanya ketika membaca warna. Hal ini karena LED dapat menyala secara tiba-tiba tanpa ada waktu pemanasan seperti lampu biasa. Hal ini dapat diselesaikan dengan membiarkan **/LED** *low* dan *strobing* **EN** *high* atau hanya *menstrobing* **/LED** *low* ketika **EN** *high.*



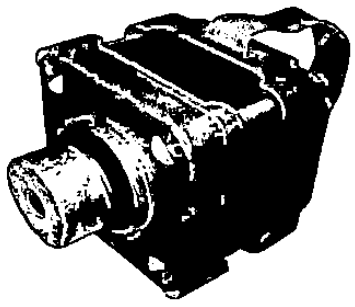
Gambar 2.29 Skema *adapter* AppMod



Gambar 2.30 Skema modul sensor

2.2.4 Motor *Stepper*

Motor *stepper* adalah perangkat elektromekanis yang bekerja dengan mengubah pulsa *elektronis* menjadi gerakan mekanis *diskrit*. Motor *stepper* bergerak berdasarkan urutan *pulsa* yang diberikan kepada motor. Karena itu, untuk menggerakkan motor *stepper* diperlukan pengendali motor *stepper* yang membangkitkan *pulsa-pulsa periodik*. Penggunaan motor *stepper* memiliki beberapa keunggulan dibandingkan dengan penggunaan motor DC biasa.



Gambar 2.31 Contoh motor *stepper*

Keunggulannya antara lain adalah :

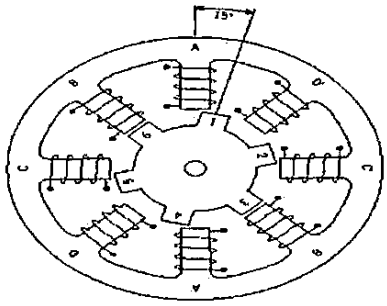
- Sudut *rotasi* motor *proporsional* dengan *pulsa* masukan sehingga lebih mudah diatur.
- Motor dapat langsung memberikan *torsi* penuh pada saat mulai bergerak
- Posisi dan pergerakan repetisinya dapat ditentukan secara *presisi*
- Memiliki respon yang sangat baik terhadap mulai, *stop* dan berbalik (perputaran)
- Sangat *realiable* karena tidak adanya sikat yang bersentuhan dengan *rotor* seperti pada motor DC
- Dapat menghasilkan perputaran yang lambat sehingga beban dapat dikopel langsung ke porosnya
- *Frekuensi* perputaran dapat ditentukan secara bebas dan mudah pada *range* yang luas.

Pada dasaranya terdapat 3 tipe motor *stepper* yaitu:

1. Motor *stepper* tipe *Variable reluctance* (VR)

Motor *stepper* jenis ini telah lama ada dan merupakan jenis motor yang secara struktural paling mudah untuk dipahami. Motor ini terdiri atas sebuah *rotor* besi lunak dengan beberapa gerigi dan sebuah lilitan *stator*. Ketika lilitan *stator* diberi energi dengan arus DC, kutub-kutubnya menjadi

kutub *stator*. Berikut ini adalah penampang melintang dari motor *stepper* tipe *variable reluctance* (VR):

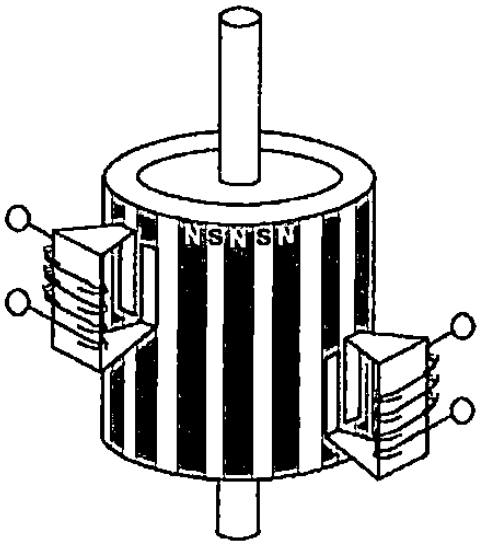


Gambar 2.32 Penampang melintang dari motor *stepper* tipe *variable reluctance* (VR)

2. Motor *stepper* tipe *Permanent Magnet* (PM)

Motor *stepper* jenis ini memiliki *rotor* yang berbentuk seperti kaleng bundar (*tin can*) yang terdiri atas lapisan magnet permanen yang diselang-seling dengan kutub yang berlawanan (perhatikan gambar 2.30). Dengan adanya magnet permanen, maka intensitas *fluks magnet* dalam motor ini akan meningkat sehingga dapat menghasilkan *torsi* yang lebih besar. Motor jenis ini biasanya memiliki *resolusi* langkah *(step)* yang rendah yaitu antara $7{,}5^0$ hingga 15^0 per langkah atau 48 hingga 24 langkah setiap putarannya.

Berikut ini adalah ilustrasi sederhana dari motor *stepper* tipe *permanent magnet*. Lihat gambar 2.33

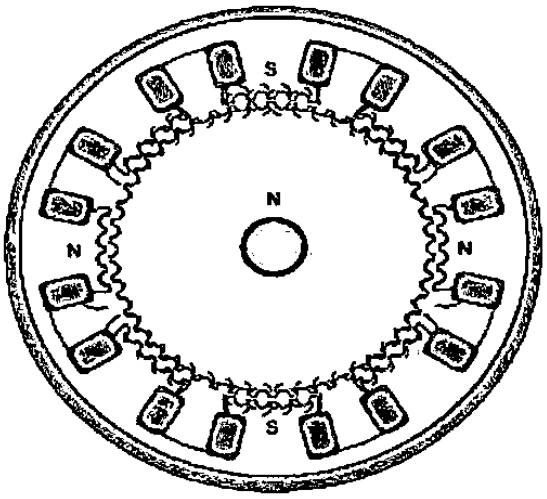


Gambar 2.33 Ilustrasi sederhana dari motor *stepper* tipe *permanent magnet* (PM)

3. Motor *stepper* tipe *Hybrid* (HB)

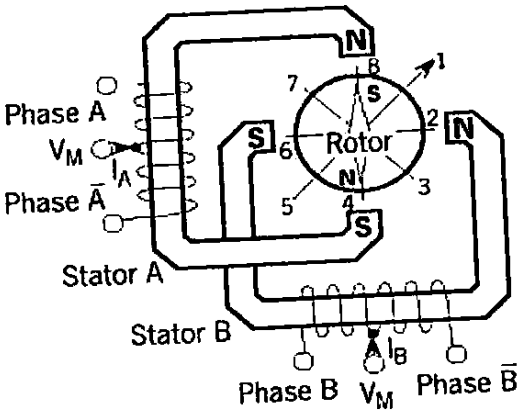
Motor *stepper* tipe *hybrid* memiliki struktur yang merupakan kombinasi dari kedua tipe motor *stepper* sebelumnya. Motor *stepper* tipe *hybrid* memiliki gigi-gigi seperti pada motor tipe VR dan juga memiliki magnet permanen yang tersusun secara *aksial* pada batang porosnya seperti motor tipe PM. Motor tipe ini paling banyak digunakan dalam berbagai aplikasi karena kinerja lebih baik. Motor tipe *hybrid* dapat menghasilkan resolusi langkah yang tinggi yaitu antara 3.6^0 hingga 0.9^0 per langkah atau 100-400 langkah setiap putarannya

Berikut ini adalah penampang melintang dari motor *stepper* tipe *hybrid* yang dapat dilihat pada gambar 2.34



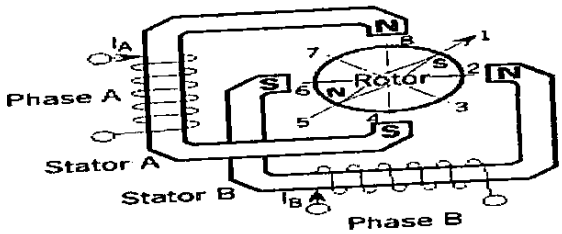
Gambar 2.34 Penampang melintang dari motor *stepper* tipe *hybrid*

Berdasarkan metode perancangan rangkaian pengendalinya, motor *stepper* dapat dibagi menjadi jenis *unipolar* dan *bipolar*. Rangkaian pengendali motor *stepper unipolar* lebih mudah dirancang karena hanya memerlukan satu *switch / transistor* setiap lilitannya. Untuk menjalankan dan menghentikan motor ini cukup dengan menerapkan pulsa *digital* yang hanya terdiri atas tegangan *positif* dan nol (*ground*) pada salah satu *terminal* lilitan (*wound*) motor sementara *terminal* lainnya dicatu dengan tegangan *positif* konstan (V_M) pada



Gambar 2.35 Motor *stepper* dengan lilitan *unipolar*

Untuk motor *stepper* dengan lilitan *bipolar*, diperlukan sinyal pulsa yang berubah-ubah dari *positif* ke *negatif* dan sebaliknya. Jadi pada setiap *terminal* lilitan (A & B) harus dihubungkan dengan sinyal yang mengayun dari *positif* ke *negatif* dan sebaliknya (perhatikan gambar 2.36). Karena itu dibutuhkan rangkaian pengendali yang agak lebih kompleks daripada rangkaian pengendali untuk motor *unipolar*. Motor *stepper bipolar* memiliki keunggulan dibandingkan dengan motor *stepper unipolar* dalam hal *torsi* yang lebih besar untuk ukuran yang sama.



Gambar 2.36 Motor *stepper* dengan lilitan *bipolar*

2.2.5 Sensor *Optocoupler*

Optocoupler adalah suatu piranti yang terdiri dari 2 bagian yaitu *transmitter* dan *receiver*, yaitu antara bagian cahaya dengan bagian deteksi sumber cahaya terpisah. Biasanya *optocoupler* digunakan sebagai *saklar elektrik*, yang bekerja secara otomatis.

Optocoupler adalah suatu komponen penghubung *(coupling)* yang bekerja berdasarkan picu cahaya *optic*. *Optocoupler* terdiri dari dua bagian yaitu :

1. Pada *transmitter* dibangun dari sebuah LED infra merah. Jika dibandingkan dengan menggunakan LED biasa, LED infra merah memiliki ketahanan yang lebih baik terhadap sinyal tampak. Cahaya yang dipancarkan oleh LED infra merah tidak terlihat oleh mata telanjang.
2. Pada bagian *receiver* dibangun dengan dasar komponen *phototransistor*. *Phototransistor* merupakan suatu *transistor* yang peka terhadap tenaga cahaya. Suatu sumber cahaya menghasilkan energi panas, begitu pula dengan *spektrum* infra merah. Karena *spektrum* infra merah mempunyai efek panas yang lebih besar dari cahaya tampak, maka *phototransistor* lebih peka untuk menangkap *radiasi* dari sinar infra merah.

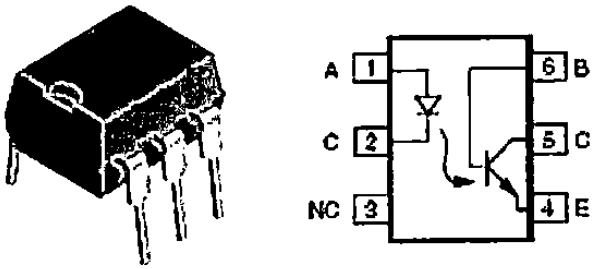
Ditinjau dari penggunaanya, fisik *optocoupler* dapat berbentuk bermacam-macam. Bila hanya digunakan untuk mengisolasi *level* tegangan atau data pada sisi *transmitter* dan sisi *receiver*, maka *optocoupler* ini biasanya dibuat dalam bentuk *solid (tidak ada ruang antara LED dan phototransistor)*. Sehingga sinyal

listrik yang ada pada *input* dan *otput* akan *terisolasi.* Dengan kata lain *optocoupler* ini digunakan sebagai *optoisolator* jenis *IC.*

Prinsip kerja dari *optocoupler* adalah :

- Jika antara *phototransistor* dan LED terhalang maka *phototransistor* tersebut akan *off* sehingga *output* dari *kolektor* akan berlogika *high.*
- Sebaliknya jika antara *phototransistor* dan LED tidak terhalang maka *phototransistor* tersebut akan *on* sehingga *output*-nya akan berlogika *low.*

Biasanya dipasaran *optocoupler* tersedia dengan tipe 4N25 / 4N35 ini mempunyai tegangan *isolasi* 7500 volt dengan kemampuan maksimal LED dialiri arus *forward* sebesar 3A.

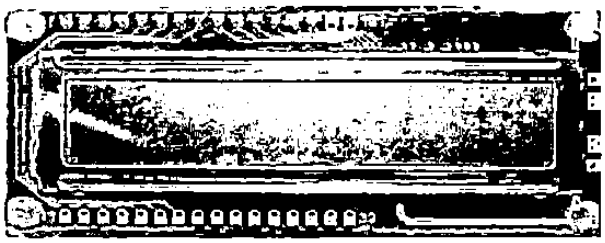


Gambar 2.37 *Optocoupler* tipe 4N25

2.2.6 LCD

Modul LCD *Character* dapat dengan mudah dihubungkan dengan

mempunyai lebar *display* 2 baris 16 kolom atau biasa disebut sebagai LCD karakter 2x16, dengan 16 *pin* konektor, yang didefinisikan sebagai berikut:



Gambar 2.38 Modul LCD Karakter 2x16

Tabel 2.9. Pin dan Fungsi LCD

PIN	Name	Function
1	VSS	*Ground voltage*
2	VCC	+5V
3	VEE	*Contrast voltage*
4	RS	*Register Select* 0 = *Instruction Register* 1 = *Data Register*
5	R/W	*Read/ Write,* to *choose write or read mode* 0 = *write mode* 1 = *read mode*
6	E	*Enable* 0 = *start to lacht data to LCD character* 1= *disable*
7	DB0	LSB
8	DB1	-
9	DB2	-
10	DB3	-
11	DB4	-
12	DB5	-
13	DB6	-
14	DB7	MSB
15	BPL	*Back Plane Light*
16	GND	*Ground voltage*

Display karakter pada LCD diatur oleh *pin* EN, RS, dan RW. Jalur EN dinamakan *Enable*. Jalur ini digunakan untuk memberitahu LCD bahwa anda sedang mengirimkan sebuah data. Untuk mengirimkan data ke LCD, maka melalui program EN harus dibuat logika *low* "0" dan set pada dua jalur *control* yang lain RS dan RW. Ketika dua jalur yang lain telah siap, *set* EN dengan logika "1" dan tunggu untuk sejumlah waktu tertentu (Sesuai dengan *datasheet* dari LCD tersebut). Dan berikutnya *set* EN ke logika *low* "0" lagi. Jalur RS adalah jalur *Register Select*. Ketika RS berlogika *low* "0", data akan dianggap sebagai sebuah perintah atau intruksi khusus (seperti *clear screen*, posisi kursor dll). Ketika RS berlogika "1", data yang dikirim adalah data *text* yang akan di tampilkan pada *display* LCD. Sebagai contoh, untuk menampilkan huruf "T" pada layar LCD maka RS harus di *set* logika *high* "1" jalur RW adalah jalur *control Red/White*. Ketika RW berlogika *low* "0" , maka informasi pada *bus* akan dituliskan pada layar LCD. Ketika RW berlogika "1", maka program akan melakukan pembacaan memori dari LCD. Sedangkan pada aplikasi umum *pin* RW selalu diberi logika *low* "0". Pada akhirnya, bus data terdiri dari 4 atau 8 jalur (bergantung pada mode operasi yang dipilih oleh *user*). Pada kasus *bus* data 8 bit, jalur disebut sebagai DB0 s/d DB7.