BAB II
PEMOGRAMAN MIKROKONTROLER

II.1 Basis Kerja Digital

Sistem digital didefinisikan sebagai suatu sistem yang menggunakan teori dan teknik digital untuk disain sistem informasi serta komponen dan alat elektronik yang digunakan untuk kontrol digital. Dalam teknologi digital, *output* dan *input* berupa data digital. Dibandingkan analog, teknologi digital sangat serbaguna karena teknologi ini menggunakan sinyal digital yang lebih akurat, dapat diandalkan, dan bebas dari kesalahan. Sistem angka digital membentuk dasar pengertian teknologi digital.

Dalam konteks teknologi dan *engineering*, kuantitas adalah nilai atau suatu ukuran yang direpresentasikan oleh fungsi dari tanda-tanda numerik dengan menggunakan sistem angka. Contohnya, kuantitas dari tegangan dan arus yang melalui konduktor direpresentasikan menggunakan sistem angka. Manusia (dunia alamiah) menggunakan sistem angka desimal, yang mengakomodasi 10 digit (0, 1, 2, ... 9). Lain halnya dalam dunia buatan (*artificial*/komputer), peralatan komputer menggunakan dua digit, yaitu ”0” dan ”1” untuk merepresentasikan nilai sesuatu, karena mereka tidak terlalu ”pintar” dibanding manusia.

Selain dua jenis angka diatas (desimal dan biner), terdapat pula beberapa sistem angka lain seperti sistem angka oktal, heksadesimal, dan *Binary Coded Decimal* (BCD). Sistem angka oktal mengakomodasi delapan digit angka, yaitu 0, 1, 2, 3, 4, 5, 6, dan 7. Sistem angka heksadesimal mengakomodasi 16 digit yaitu 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, dan F. Sistem angka yang disebutkan di atas adalah sistem angka yang berbasis digital. Artinya bahwa semua sistem angka tersebut dapat direpresentasikan dengan hanya dua digit angka, yaitu 0 dan 1. Inilah yang disebut dengan Sistem Angka Digital (*digital number system*-DNS).

II.1.1 Bilangan Binari

II.1.1.1 Pengertian Bilangan Binari

Seperti yang telah dijelaskan diatas, bilangan Binari hanya terdiri dari dua buah digit yaitu ’0’ dan ’1’. Masing-masing digit ini disebut pula dengan bits. Sebuah sistem angka digital (DNS) dapat terdiri dari dua, tiga, atau empat bit binari sesuai dengan kombinasi yang diperlukan. Kombinasi yang dimaksudkan disini adalah :

- Kombinasi dua bits :

  00, 01, 10 atau 11 sehingga kombinasi yang didapatkan $2^2 = 4$

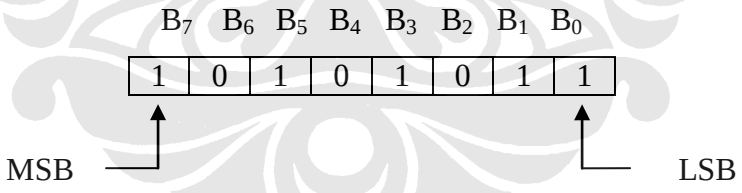
- Kombinasi tiga bits :

  000, 001, 010, 011, 100, 101, 110, atau 111 sehingga kombinasi yang didapatkan $2^3 = 8$

- Kombinasi empat bits :

  0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, atau 1111 sehingga kombinasi yang didapatkan $2^4 = 16$

Dalam sebuah DNS, gabungan dari delapan buah bits paling umum digunakan yang dikenal juga dengan sebutan *byte*. Sehingga 1 *byte* = 8 bits dengan susunan sebagai berikut :



Gambar II. 1 : Definisi Byte [7]

Dimana bagian bit paling kanan disbut pula dengan *Least Significant Bit* (LSB), sedangkan bit paling kiri disebut dengan *Most Significant Bit* (MSB).

II.1.1.2 Konversi Bilangan Desimal ke Bilangan Binari

Perbedaan mendasar antara bilangan desimal dengan bilangan binari adalah orde yang digunakan. Bilangan desimal menggunakan orde 10, sedangkan bilangan binari menggunakan orde 2. Untuk mengkonversi

bilangan desimal ke bilangan binari yang perlu dilakukan adalah membagi setiap bilangan desimal dengan faktor pembagi 2. Pada setiap proses pembagian perlu diperhatikan pula angka pembagi yang tersisa. Apabila tersisa 0, maka ditulis angka 0. Begitu pula apabila tersisa angka 1, maka ditulis pula angka 1. Proses pembagian dilanjutkan hingga bilangan yang terbagi bernilai 0. Untuk lebih jelasnya dapat memperhatikan ilustrasi konversi bilangan desimal 3442 ke dalam sebuah bilangan binari.

Tabel II. 1 : Tabel Konversi Desimal Setara Dengan Binari [7]

Dibagi 2	Hasil Pembagian	Sisa	Binari
3442	1711	0	
1711	855	1	
855	422	1	
422	211	0	
211	105	1	
105	52	1	$(110100110110)_2$
52	26	0	
26	13	0	
13	6	1	
6	3	0	
3	1	1	
1	0	0	

Berikut ini adalah contoh dalam mengkonversi bilang desimal 0.48 ke bilangan binari.

Tabel II. 2 : Tabel Konversi Desimal Setara Dengan Binari [7]

Dikalikan 2	Hasil Pengalian	Sisa	Binari
0.48	0.96	0	
0.96	1.92	1	
0.92	1.84	1	
0.84	1.68	1	
0.68	1.36	1	
0.36	0.72	0	$(0.011110101110)_2$
0.72	1.44	1	
0.44	0.88	0	
0.88	1.76	1	
0.76	1.52	1	
0.52	1.04	1	
0.04	0.08	0	

Sedangkan proses konversi dari bilangan binari kedalam bilangan desimal adalah dengan mengalikan bit pada bilangan binari dengan bilangan berorde dua sesuai dengan susunan dan urutan yang telah dijelaskan sebelumnya. Sebagai contoh konversi bilangan 1110 0011 adalah 227.

$1x2^7 + 1x2^6 + 1x2^5 + 0x2^4 + 0x2^3 + 0x2^2 + 1x2^2 + 1x2^0 =$

$128 + 64 + 32 + 0 + 0 + 0 + 2 + 1 = (227)_{10}$

II.1.2 Bilangan Oktal Dan Hexadesimal

Bilangan Oktal menggunakan angka 8 sebagai orde dan mengakomodasi bilangan digit 0, 1, 2, 3, 4, 5, 6, dan 7. Konversi bilangan oktal kedalam bilangan binari menggunakan tabel konversi bilangan 3-bit binari.

Tabel II. 3 : Tabel Konversi 3-Bits Binari Setara Dengan Digit Oktal [7]

Oktal Digits	3-Bit Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Sebagai contoh, bilangan oktal $(5237)_8$ setara dengan bilangan binari $(101010011111)_2$.

5 2 3 7 → Bilangan Oktal

↓ ↓ ↓ ↓

101 010 011 111 = $(101010011111)_2$

Proses konversi ini dapat berlaku pula sebaliknya (*vice versa*) untuk konversi dari bilangan binari ke bilangan oktal.

Sedangkan bilangan *hexadesimal* merupakan sistem angka digital yang bekerja dengan sistem angka digital lainnya. Perbedaan hanya terletak pada konfigurasi nilai yang menggunakan 4 buah bits. Proses pengubahan sama dengan sistem angka oktal, seperti ditunjukkan pada tabel konversi berikut :

Tabel II. 4 : Tabel Konversi 4-Bits Binari Setara Dengan Digit Hexadesimal [7]

Hexadecimal Digits	4-Bit Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Metode konversi yang digunakan juga sama dengan metode konversi yang digunakan pada bilangan oktal (*vice versa*).

II.2 Dasar Bahasa Pemrograman C

Bahasa C merupakan bahasa pemrograman tingkat tinggi (*High Level Programming Language*) yang telah banyak dikenal dan dapat digunakan untuk banyak aplikasi. Salah satu aplikasi dari bahasa pemrograman C adalah untuk melakukan pemrograman pada mikrokontroler dengan bantuan software tertentu. Setiap program yang menggunakan bahasa C setidaknya harus mempunyai satu fungsi, yaitu *main()*. Fungsi ini merupakan dasar dari program bahasa C dan juga merupakan titik awal pada saat kode program dieksekusi. Fungsi *main()* merupakan fungsi yang paling awal dieksekusi saat program dijalankan. Dalam beberapa kasus, fungsi *main()* hanya terdiri atas beberapa pernyataan yang berfungsi untuk menginisialisasi dan mengendalikan aliran program dari satu fungsi ke fungsi lainnya.

II.2.1 Variabel dan Konstanta

Data yang disimpan saat sebuah program dijalankan dapat berupa variabel atau konstanta. Variabel merupakan suatu nilai yang dapat berubah, sementara konstanta mempunyai nilai yang tetap. Variabel dan konstanta ini mempunyai bentuk yang beragam, sehingga penyimpanannya pun berbeda antara jenis yang satu dengan lainnya.

II.2.1.1 Tipe Variabel

Sebuah variabel dideklarasikan dengan menggunakan kata-kata yang menunjukkan jenis variabel tersebut dan ukurannya. Variabel dan konstanta akan disimpan dalam memori yang terbatas dari mikrokontroler sehingga perlu dipertimbangkan sedemikian rupa sehingga tidak membuang percuma memori yang ada. Berikut beberapa jenis variabel beserta ukurannya.

Tabel II. 5 : Jenis dan Ukuran Variabel [4]

Tipe	Ukuran (bit)	Range
bit	1	0,1
char	8	-128 sampai 127
unsigned char	8	0 sampai 255
signed char	8	-128 sampai 127
int	16	-32768 sampai 32767
short int	16	-32768 sampai 32767
unsigned int	16	0 sampai 65535
signed int	16	-32768 sampai 32767
long int	32	-2147483648 sampai 2147483647
unsigned long int	32	0 sampai 4294967295
signed long int	32	-2147483648 sampai 2147483647
float	32	±1.175e-38 sampai ±3.402e38
double	32	±1.175e-38 sampai ±3.402e38

Pendeklarasian suatu variabel dilakukan dengan penulisan,

```
unsigned char kotak;
int jambu;
long int bola_pejal;
```

II.2.1.2 Lingkup Variabel

Variabel dan konstanta harus dideklarasikan sesuai dengan kegunaanya. Lingkup dari variabel mengindikasikan kemampuan suatu variabel untuk diakses dalam sebuah program. Sebuah variabel dapat dinyatakan sebagai suatu variabel lokal maupun variabel global. Variabel lokal hanya dapat diakses pada fungsi dimana variabel tersebut dideklarasikan. Sementara variabel global dapat diakses oleh semua fungsi yang ada dalam program. Penempatan variabel lokal dan variabel global dapat dilihat pada contoh di bawah ini:

```
unsigned char global    //global variabel

void fungsi_x (void)
{
  int lokal_x;          //variabel lokal untuk
                        fungsi_x
       pernyataan;
       ....
}

void main ()
{
  char lokal_main;      //variabel lokal untuk
                        main()
  while(1)              //program berjalan terus
  ;
}
```

II.2.2 *Control Statement*

Control statement digunakan untuk mengatur aliran pengeksekusian program. Ada beberapa *control statement* yang terdapat dalam bahasa pemrograman C.

II.2.2.1 *While Loop*

Merupakan salah satu dasar dalam penegendalian aliran eksekusi program. Bentuk dari perintah *while* adalah;

```
while (kondisi){        atau        while (kondisi)
      pernyataan1;                        pernyataan;
      pernyataan2;
}
```

Ketika eksekusi dari program mencapai bagian awal dari perintah *while*, kondisi yang ada akan dievaluasi. Apabila kondisi tersebut BENAR (tidak nol) maka pernyataan yang terdapat didalam perintah *while* akan dieksekusi. Pernyataan yang terletak di dalam perintah *while* adalah pernyataan-pernytaan yang terdapat dalam tanda {} setelah perintah *while* (untuk pernyataan majemuk) atau pernyataan yang tepat berada setelah perintah *while* (pernyataan tunggal). Program akan berulang terus selama kondisi yang ada masih terpenuhi. Apabila kondisi sudah tidak terpenuhi maka pernyataan-pernyataan yang ada dalam perintah *while* akan dilewati.

II.2.2.2 *Do/While Loop*

Perintah ini hampir sama dengan perintah *while*, hanya saja kondisi baru dievaluasi setelah satu *loop* dilaksanakan. Artinya bahwa pernyataan yang ada dalam perintah *do/while* ini selalu dilaksanakan sekali terlebih dahulu sebelum diperiksa apakah kondisi yang disyaratkan terpenuhi atau tidak. Format dari perintah *do/while* adalah;

```
do                                    do
{                          atau           pernyataan;
      pernyataan1;                    while (kondisi);
      pernyataan2;
      ...
} while (kondisi);
```

II.2.2.3 *For Loop*

For loop digunakan untuk menjalankan suatu kumpulan perintah untuk jumlah pengulangan tertentu. Dapat dideskripsikan sebagai suatu proses inisialisasi, pengetesan, dan aksi yang menuju pada hasil yang memuaskan. Format dari perintah *for* adalah;

```
for (kds1;kds2;kds3)                  for(kds1;kds2;kds3)
{                          atau           pernyataan;
      pernyataan1;
      pernyataan2;
      ...
}
```

"kds1" hanya akan dieksekusi selama sekali saat masuk pertama kali dalam perintah for. Merupakan pernyataan yang digunakan untuk menginisialisasi keadaan dari "kds2".

"kds2" merupakan pernyataan yang berfungsi untuk menentukan saat untuk keluar dari perintah *for*. "kds3" merupakan kondisi yang digunakan untuk dapat memenuhi kondisi dari "kds2".

Perintah *for* ini dapat dijabarkan dalam bentuk perintah *while* sebagai berikut.

```
kds1;
while (kds2)
{
       pernyataan1;
       pernyataan2;
       ...
       kds3;
}
```

II.2.2.4 *If/Else*

Perintah *if/else* merupakan perintah yang digunakan untuk mengatur aliran eksekusi program berdasarkan kondisi yang ada.

Format perintah *if* ;

```
if (kondisi)                          if(kondisi)
{                          atau             pernyataan;
       pernyataan1;
       pernyataan2;
       ...
}
```

Pada perintah *if*, pernyataan-pernyataan yang ada di dalam perintah tersebut akan dieksekusi apabila kondisi yang ada telah tercapai.

Format perintah *if/else* ;

```
if (kondisi)                          if(kondisi)
{                          atau             pernyataan1;
       pernyataan1;                    else
       pernyataan2;                           pernyataan2;
}
else
{
       pernyataan3;
       pernyataan4;
}
```

Pada perintah *if/else*, pernyataan dalam *if* akan dieksekusi apabila kondisi tercapai. Kalau kondisi tidak tercapai, pernyataan dalam *else* yang akan dieksekusi.

Selain dua bentuk di atas, kombinasi dari perintah *if/else* juga dapat dibentuk seperti format di bawah ini;

```
if (kondisi 1)
      pernyataan1;

else if (kondisi2)
      pernyataan2;

else if (kondisi3)
      pernyataan3;

else
      pernyataan4;
```

II.3 Mikrokontroler

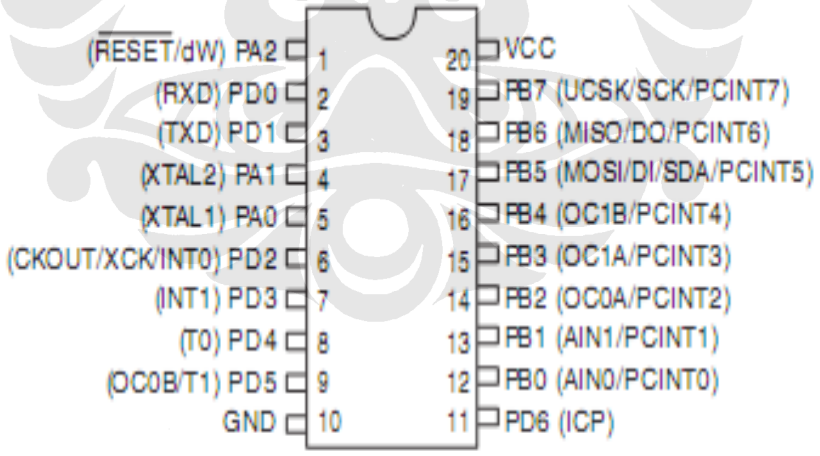
Mikrokontroler adalah sebuah sirkuit terintegrasi secara digital. Mikrokontroler dipergunakan dalam otomasi dan aplikasi kontrol. Saat ini domain aplikasi mikrokontroler sudah meluas dalam sektor komputasi. Banyak mikrokontroler ditambahkan pada modem, *disk drive*, *floppy drive*, dan sebagainya. Mikrokontroler terdiri utamanya dari CPU dan *peripheral* tambahan dalam arsitekrturnya. CPU terdiri dari ALU, *register*, *buffer*, *instuction decoder*, *control unit*, *Program Counter*, *Stack Pointer*, *interrupt controller*, SID/SOD unit, dan bus. CPU merupakan fungsi dasar yang umum yang dibutuhkan untuk menginterpretasi dan mengeksekusi instruksi atau program. Unit-unit tambahannya meliputi :

- *Analog to Digital Converter* (ADC)
- *Programable Timer*
- *Watchdog Timer*
- *Interrupt Controller*
- *Pulse Width Modulation* (PWM)
- *Phase Locked Loop*
- *Memory Controller*
- Tipe Data dan Mode Pengalamatan

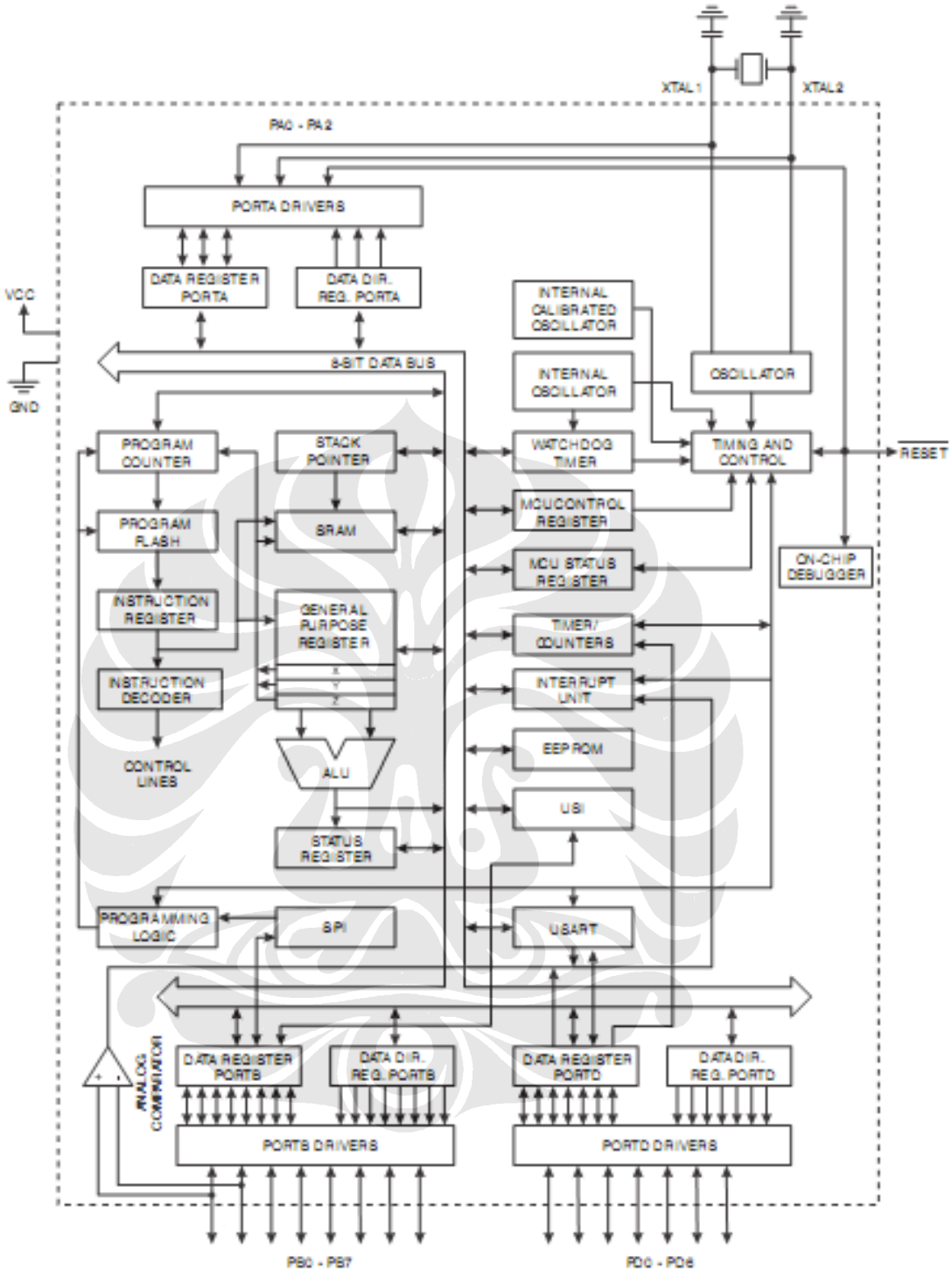
- I/O Port
- EPROM, ROM, dan sebagainya

Obyek dimana sistem kontrol ini dibuat disebut *plant*. Suatu *plant* dapat dikendalikan menggunakan mikrokontroler untuk mengerjakan pekerjaan tertentu apabila *plant* tersebut telah dilengkapi dengan sensor dan aktuator yang terhubung secara logis melalui mikrokontroler. Dalam sebuah sistem kontrol berbasis mikrokontroler, kebutuhan akan sirkuit-sirkuit *eksternal* seperti pengkondisian sinyal dan unit-unit tambahan lainnya sangat sedikit. Disain sistem kontrol berbasis mikrokontroler sangat tangguh dan dapat diandalkan karena tidak lagi memerlukan *peripheral* tambahan. *Peripheral* tambahan ini sudah dibangun sebagai satu kesatuan di dalam unit mikrokontrolernya. Mikrokontroler dapat diandalkan karena memiliki lebih sedikit koneksi tambahan dan antarmuka (*interface*) yang dibutuhkan, sehingga mengurangi ukuran *part*, menambah performa dan efisiensi, serta biaya implementasi secara keseluruhan yang rendah [3].

Mikrokontroler yang digunakan dalam sistem *Automatic Cruise Control* ini adalah jenis ATTiny 2313.



Gambar II. 2 : Mikrokontroler ATTiny 2313 [3]



Gambar II. 3 : Diagram Blok ATTiny 2313 [3]

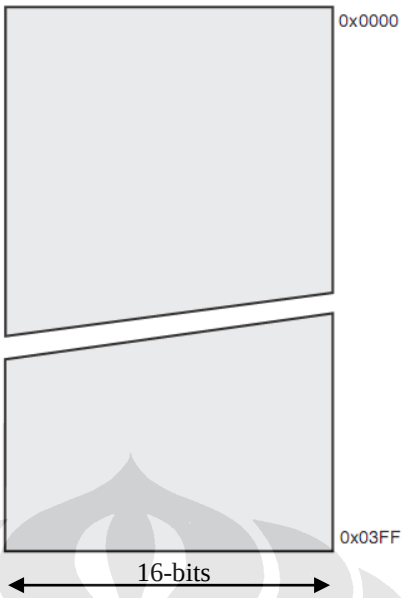
Mikrokontroler jenis ATTiny 2313 ini merupakan jenis mikrokontroler 8-bit AVR. Fitur-fitur yang terdapat di dalamnya antara lain;

- Memori :
    - ✓ 2K *bytes* untuk *In-System Self Programmable Flash* dengan ketahanan untuk *write/erase* sampai 10.000 kali.
    - ✓ 128 *bytes* untuk *In-System Programmable EEPROM* dengan ketahanan untuk *write/erase* sampai 10.000 kali.
    - ✓ 128 *bytes* internal SRAM.
- Tegangan yang dibutuhkan saat bekerja : 2.7 - 5.5 Volt.
- Fitur *peripheral*;
    - ✓ 1 buah *Timer/Counters* 8-bit dengan *prescaler* dan *compare mode* yang terpisah.
    - ✓ 1 buah *Timer/Counters* 16-bit dengan *prescaler, compare mode*, dan *capture mode* yang terpisah.
    - ✓ 4 buah *PWM Channels*
    - ✓ *Analog Comparator*
    - ✓ *Watchdog Timer*
    - ✓ USI – *Universal Serial Interface*

II.4 *Memory*

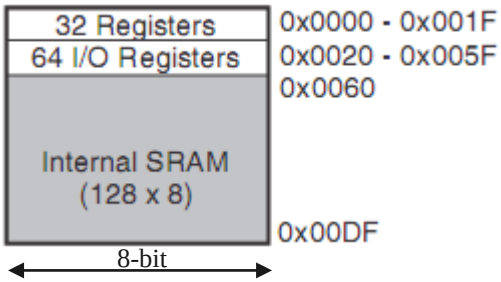
Sebuah mikrokontroler mempunyai beberapa jenis memori yang kapasitasnya berbeda, tergantung spesifikasi dari mikrokontroler tersebut. Pada mikrokontroler ATTiny 2313 sendiri terdapat tiga jenis memori, yaitu Flash memori, SRAM (Data) memori, dan EEPROM memori.

Flash memori merupakan jenis memori *non-volatile*, yaitu memori yang tetap menahan data yang disimpannya, walaupun sumber tenaga untuk menghidupkan mikrokontroler dimatikan [4]. Memori jenis ini biasanya digunakan untuk menyimpan *executable code* dan konstanta karena data ini harus tetap berada di memori walaupun sumber tenaga telah dimatikan. Flash memori berbentuk blok-blok yang mempunyai lebar 16-bit dan dimulai dengan lokasi 0x0000. Kapasitas keseluruhannya tergantung jenis dari mikrokontrolernya. Untuk ATTiny 2313, Flash memori mempunyai kapasitas sampai 2K bytes.



Gambar II. 4 : Peta Flash Memori ATTiny 2313 [3]

Data memori pada mikrokontroler jenis Atmel AVR, terdiri atas tiga bagian yang terpisah, yaitu *General-Purpose Working Register*, *I/O Register*, dan *internal SRAM*. *General-Purpose Working Register* digunakan untuk menyimpan variabel lokal dan data sementara lainnya yang digunakan oleh program saat program sedang berjalan. *I/O register* digunakan sebagai perangkat antarmuka (*interface*) dengan peralatan I/O dan *peripheral-peripheral* yang terintegrasi dalam mirokontroler. Sedangkan *internal SRAM* digunakan untuk menyimpan variabel umum (*general variables*) dan juga untuk *processor stack*. Pada ATTiny 2313, 32 blok pertama digunakan untuk *General-Purpose Working Register*, 64 blok untuk *I/O register*, dan 128 blok untuk *internal SRAM*.



Gambar II. 5 : Peta Data Memori ATTiny 2313 [3]

EEPROM memori merupakan jenis memori *non-volatile*, sama dengan Flash memori. Walaupun EEPROM memori juga memiliki kemampuan *write/erase* seperti halnya memori jenis lain, tetapi memori ini jarang digunakan untuk menyimpan variabel umum. Hal ini dikarenakan EEPROM memori sangat lambat dalam proses penyimpanannya (*write*). Oleh karena itu, memori ini biasanya digunakan untuk menyimpan variabel yang harus dapat dijaga nilainya meskipun terjadi penurunan daya.

II.5 Port Input/Output Paralel

Port input/output (I/O) paralel adalah perlengkapan I/O yang paling umum. Setiap port I/O paralel memiliki tiga *register* I/O yang berhubungan, yaitu:

- *data direction register* (DDRx), dimana *x* adalah A, B, C, dan seterusnya tergantung dari mikrokontroler dan port paralel yang digunakan.
- *port driver register* (PORTx)
- *port pin register* (PINx)

Tujuan DDRx adalah untuk menentukan bit yang mana dari suatu port digunakan sebagai input, dan bit yang mana digunakan sebagai output. Bit input dan output dapat ditentukan sesuai keinginan programer. Bila mikrokontroler di-reset, mikrokontroler membersihkan semua bit DDRx ke logika 0, men-set semua bit port sebagai input. Men-set suatu bit DDRx ke logika 1 membuat bit port yang bersangkutan menjadi mode output. Contohnya, men-set dua *least significant bit* (LSB) dari DDRA menjadi logika 1 dan bit yang lain ke logika 0 akan men-set dua bit LSB port A menjadi output dan bit yang lain menjadi input.

Contoh program untuk menulis ke bit output di PORTA adalah :

```
PORTA = 0x02;  //men-set bit kedua dari port A dengan logika
               1 dan membersihkan bit yang lain menjadi
               logika 0
```

Contoh program untuk membaca bit input PORTA adalah :

```
x = PINA;      //membaca keseluruhan 8 bit dari port A.
```

Pada contoh tersebut, "x" merupakan nilai dari seluruh bit di PORTA, input dan output, karena register PIN merefleksikan nilai dari seluruh bit dari port

tersebut. Pin port input adalah tersebar/mengambang, sehingga tidak perlu sebuah resistor *pull-up* diasosiasikan terhadap pin port. Mikrokontroler dapat memberikan resistor *pull-up* jika diinginkan dengan memberikan logika 1 pada bit yang bersangkutan dari PORTx. Contohnya :

```
DDRA = 0xC0;   //2 bit paling atas port A sebagai output, 6
               bit terendah sebagai input

PORTA = 0x03;  //membolehkan pull-up internal pada dua bit
               terendah
```

Secara umum walaupun bervariasi pada mikrokontroler yang berbeda, pin port mampu menyediakan 20mA. Artinya bahwa port dapat secara langsung menghidupkan LED.

II.6 Fungsi *Interrupt* Dan *Reset*

II.6.1 *Interrupt*

Sebuah *interrupt* adalah perubahan aliran, atau interupsi pada operasi program yang disebabkan oleh sumber eksternal atau internal dari suatu hardware. *Interrupt* memiliki efek akan memanggil suatu fungsi lain untuk dieksekusi bila *interrupt* itu sendiri terjadi. Hasilnya *interrupt* akan menyebabkan aliran eksekusi fungsi program utama berhenti, sementara fungsi *interrupt* yang disebut *Interrupt Service Routine* (ISR) dieksekusi. Setelah ISR selesai dieksekusi, aliran fungsi program utama akan berlanjut kembali dari saat program tersebut diinterupsi.

Interrupt akan menyebabkan status register dan program counter disimpan di *stack*, dan berdasarkan sumber *interrupt*, program counter akan diberi suatu nilai dari tabel alamat-alamat register. Alamat-alamat ini dianggap sebagai vektor. Ketika aliran program utama sudah diarahkan kembali oleh vektor *interrupt*, aliran program tersebut dapat dikembalikan ke operasi normal melalui suatu mesin instruksi RETI (*RETurn from Interrupt*). Instruksi RETI mengembalikan status register kepada nilai sebelum terinterupsi dan meletakkan program counter pada mesin instruksi selanjutnya setelah instruksi yang diinterupsi.

Banyaknya jumlah *interrupt* yang tersedia pada mikrokontroler tergantung dari ukuran mikrokontroler tersebut, semakin besar mikrokontroler semakin banyak *interrupt* yang tersedia. Untuk mengetahui suatu mikrokontroler memiliki *interrupt* atau tidak dapat dilihat dari header file di awal program, yang juga merupakan definisi dari mikrokontroler tersebut. Header file ini spesifik untuk tiap-tiap jenis mikrokontroler.

ISR merupakan fungsi yang dipanggil oleh sistem *interrupt* bila *interrupt* terjadi. ISR dideklarasikan menggunakan kata-kata *interrupt* sebagai tipe fungsi. Kata *interrupt* ini diikuti dengan indeks yang merupakan vektor lokasi sumber *interrupt*. Contoh penulisan ISR adalah :

```
interrupt [EXT_INT0] void external_int0 (void)
{
    //berisi fungsi yang dipanggil bila interrupt pada
    interrupt 0 terjadi akibat sumber dari luar
}
```

atau

```
interrupt [TIM_OVF] void timer0_overflow (void)
{
    //berisi fungsi yang dipanggil bila timer 0 overflow
    terjadi
}
```

ISR dapat dieksekusi kapan saja bila sumber *interrupt* telah diinisialisasi dan *interrupt global* diperbolehkan. ISR tidak bisa mengembalikan suatu nilai karena secara teknis tidak ada "*caller*" dan ISR selalu dideklarasikan sebagai tipe ***void***. *Interrupt* dapat membuat eksekusi yang *real-time*.

Pada tiap jenis mikrokontroler, semua *interrupt* memiliki prioritas yang sama. Tidak diperbolehkan suatu *interrupt* menginterupsi *interrupt* yang lain tetapi dimungkinkan dua *interrupt* muncul berurutan. Apabila kasus seperti ini terjadi, suatu skema *arbitrasi* atau skema prioritas disediakan untuk menentukan *interrupt* mana yang dieksekusi lebih dulu. Bila tidak ada skema ini, *interrupt* dengan vektor yang memiliki nilai yang lebih rendah akan dieksekusi lebih dulu. Pilihan sumber *interrupt* dan nilai dar vektor-vektornya

tergantung pada mikrokontroler yang digunakan. Pada ATTiny 2313, vektor dari masing-masing *interrupt* dijelaskan dalam tabel di bawah ini;

Tabel II. 6 : Vektor Interrupt pada ATTiny 2313 [3]

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset, and Watchdog Reset
2	0x0001	INT0	External Interrupt Request 0
3	0x0002	INT1	External Interrupt Request 1
4	0x0003	TIMER1 CAPT	Timer/Counter1 Capture Event
5	0x0004	TIMER1 COMPA	Timer/Counter1 Compare Match A
6	0x0005	TIMER1 OVF	Timer/Counter1 Overflow
7	0x0006	TIMER0 OVF	Timer/Counter0 Overflow
8	0x0007	USART0, RX	USART0, Rx Complete
9	0x0008	USART0, UDRE	USART0 Data Register Empty
10	0x0009	USART0, TX	USART0, Tx Complete
11	0x000A	ANALOG COMP	Analog Comparator
12	0x000B	PCINT	Pin Change Interrupt
13	0x000C	TIMER1 COMPB	Timer/Counter1 Compare Match B
14	0x000D	TIMER0 COMPA	Timer/Counter0 Compare Match A
15	0x000E	TIMER0 COMPB	Timer/Counter0 Compare Match B
16	0x000F	USI START	USI Start Condition
17	0x0010	USI OVERFLOW	USI Overflow
18	0x0011	EE READY	EEPROM Ready
19	0x0012	WDT OVERFLOW	Watchdog Timer Overflow

Interrupt harus diinisialisasi sebelum aktif atau bisa diguanakan. Penginisialisasian *interrupt* merupakan proses dua langkah. Langkah pertama adalah membuka *interrupt* yang akan aktif dan langkah kedua adalah dengan membolehkan secara global *interrupt* yang telah dibuka.

Membuka *interrupt* dilakukan dengan memberikan nilai 1 pada kontrol register sesuai dengan *interrupt* yang akan digunakan. Kontrol register tersebut tersebut adalah *General Interrupt Mask* (GIMSK). GIMSK digunakan untuk membolehkan *interrupt* eksternal. Mengatur bit INT0 akan membolehkan *interrupt* 0 eksternal, dan mengatur bit INT1 akan membolehkan *interrupt* 1. Mengatur keduanya akan membolehkan keduanya. Bit ini diibaratkan sebagai

sebuah topeng yang menutupi *interrupt*. Pada GIMSK, bit INT0 biasanya terletak pada bit ke 6, dan INT1 biasanya terletak pada bit ke 7.

Ketika sinyal muncul di pin *interrupt*, sinyal tersebut secara logis pemrograman di-AND-kan dengan bit *interrupt* di GIMSK. Jika hasilnya 1, *interrupt* diperbolehkan muncul. Jika bit *interrupt* adalah 0, hasilnya secara logis pemrograman adalah 0 dan *interrupt* tidak diperbolehkan muncul.

Langkah kedua untuk membolehkan *interrupt* adalah dengan men-set bit *global interrupt* pada *Status Register* (SREG) di prosesor. Hal ini bisa dilakukan dengan membuat :

```
#asm("sei")
```

Kode ini dimasukkan ke bahasa pemrograman C di tempat di mana bit *global interrupt* diperbolehkan. Kode ini menggunakan pengarah compiler "`#asm`" untuk memasukkan ke dalam program instruksi bahasa pemrograman *assembly* SEI. Instruksi tersebut digunakan untuk men-set bit *global interrupt*.

Pada ISR, definisi [EXT_INTx] datang dari file `#include` dan INTx memiliki angka vektor tertentu. Compiler menggunakan informasi ini untuk meletakkan lompatan relatif ke fungsi ISR pada tempat yang tepat di tabel vektor *interrupt*.

Interrupt eksternal dapat diatur untuk bisa terpicu oleh *falling edge* atau *rising edge*. Pengaturannya ada pada *MCU Control Register* (MCUCR).

Bit	7	6	5	4	3	2	1	0	
					ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Gambar II. 6 : Definisi MCUCR ATTiny 2313 [3]

Pada mikrokontroler ATTiny 2313, kontrol bit untuk *interrupt* 0 ada pada bit 0 dan 1, yang diwakili oleh *Interrupt Sense Control* (ISC) 00 dan ISC 01. Kontrol bit untuk *interrupt* 1 ada pada bit 2 dan 3, yang diwakili oleh ISC 10 dan ISC 11.

Tabel II. 7 : Definisi ISC11 dan ISC10 [3]

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

Tabel II. 8 : Definisi ISC01 dan ISC00 [3]

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

II.6.2 Reset

Reset adalah *interrupt* dengan angka paling rendah. Ini juga merupakan *interrupt* spesial, karena selalu lebih diutamakan dari seluruh *interrupt* yang sedang berjalan. Ada tiga sumber dapat mengakibatkan reset muncul, yaitu *logic low* diberikan pada pin reset eksternal selama lebih dari 50 ns, sebagai bagian dari urutan tenaga mikrokontroler, dan *timeout* dari *watchdog timer*. Reset digunakan untuk me-preset mikrokontroler ke keadaan yang telah diketahui sehingga mikrokontroler dapat mulai kembali mengeksekusi program yang terletak di lokasi 0x000 di memori kode.

Kondisi mikrokontroler yang mengikuti reset akan berbeda tergantung dari mikrokontroler yang digunakan. Secara umum kondisi-kondisi tersebut adalah :

- seluruh *peripheral* termasuk *watchdog timer* dilumpuhkan
- seluruh port paralel di-*set* ke input
- seluruh *interrupt* dilumpuhkan

Dengan melumpuhkan seluruh *peripheral* dan *interrupt*, mikrokontroler dapat memulai eksekusi program tanpa lompatan yang tidak diharapkan. Men-set port paralel ke mode input memastikan bahwa port dan perlengkapan

eksternalnya tidak akan mencoba untuk mengarahkan port pin ke level yang berlawanan, yang bisa merusak port pin.

Salah satu hal yang dapat mengakibatkan reset muncul adalah *timeout* pada *watchdog timer*. *Watchdog timer* adalah perlengkapan keamanan yang didesain agar mikrokontroler me-reset pada saat mikrokontroler "bingung" atau "kehilangan arah" atau melakukan sesuatu yang lain selain menjalankan program yang harus dijalankan.

Watchdog timer aktif saat terjadi keadaan *timeout*. Ketika program beroperasi secara normal, program secara konsisten me-reset *watchdog timer* untuk mencegah *timeout*. Jika program "kehilangan arah", *timeout* akan muncul, dan mikrokontroler akan di-reset. Teorinya bahwa reset ini akan mengembalikan program beroperasi secara nomal.

Untuk menginisialisasi *watchdog timer,* bit-bit register pada *Watchdog Timer Control Register* (WDTCR) harus di-set ke satu. Bit-bit WDTCR antara lain:

- bit 0 : WDP0
- bit 1 : WDP1
- bit 2 : WDP2
- bit 3 : WDE
- bit 4 : WDTOE

Tabel II. 9 : WDTCR [4]

Bit	Nama	Penjelasan
WDTOE	*Watchdog Timer Off Enable*	Memperbolehkan melumpuhkan WDT
WDE	*Watchdog Timer Enable*	Mengaktifkan WDT
WDPx	*Watchdog Prescaler* bit x	Mengatur periode *timeout* untuk WDT

Osilator yang memberi detak pada WDT terpisah dari sistem clock. Frekuensinya tergantung pada tegangan yang diberikan ke mikrokontroler. Dengan 5 volt diberikan ke Vcc, frukuensinya mendekati 1 MHz, dan pada 3 volt frekuensinya mendekati 350 kHz. Hal ini membuat situasi dimana, walaupun *watchdog timer presaler* bit di WDTCR di-set untuk menentukan

timeout, waktu *timeout* sebenarnya yang diperlukan akan berbeda tergantung Vcc yang diberikan ke mikrokontroler.

Tabel II. 10 : Pemilihan Periode Watchdog Timer [4]

WDP2	WDP1	WDP0	Timeout @5 V V_{cc}	Timeout @ 3 V V_{cc}
0	0	0	15 ms	47 ms
0	0	1	30 ms	94 ms
0	1	0	60 ms	190 ms
0	1	1	120 ms	380 ms
0	0	0	240 ms	750 ms
1	0	1	490 ms	1500 ms
1	1	0	970 ms	3000 ms
1	1	1	1900 ms	6000 ms

Tabel ini hanya pendekatan, karena frekuensi osilator sangat tergantung dan merupakan pendekatan terhadap tegangan yang diberikan ke Vcc. WDT di-reset oleh eksekusi yang disebut instruksi `#asm("wdr")`. Instruksi ini merupakan instruksi kode *assembly* yang tidak terdapat di bahasa pemrograman C, dan harus dieksekusi sebelum WDT memiliki kesempatan untuk *timeout*. WDT bisa dilumpuhkan dengan cara memberikan suatu kondisi dimana bila kondisi tersebut terjadi, eksekusi pelumpuhan WDT akan dilakukan. Proses pelumpuhan ini terdiri atas dua langkah. Langkah pertama adalah dengan men-*set* bit WDTOE dan bit WDE menjadi tidak aktif, diikuti dengan menbersihkan bit WDE pada langkah berikutnya. Melumpuhkan WDT adalah sebuah operasi yang rumit untuk mencegah WDT dilumpuhkan secara tidak sengaja oleh operasi program yang tidak menentu. Normalnya WDT bila diaktifkan, tidak pernah dilumpuhkan lagi, karena tujuan mengaktifkan WDT adalah untuk melindungi mikrokontroler dari pemrosesan program yang tak menentu.

II.7 *Timer/Counter*

Timer/counter mungkin merupakan *peripheral* rumit yang paling sering digunakan di mikrokontroler. *Timer/counter* sangat serba guna, dapat digunakan

untuk mengukur waktu, PWM, kecepatan, frekuensi, atau menyediakan sinyal *output*. Walau digunakan dalam mode yang sangat berbeda yaitu, mengukur waktu dan menghitung, *timer/counter* merupakan penghitung biner simpel. Ketika digunakan untuk mengukur waktu, penghitung biner ini menghitung lama periode waktu yang diberikan pada inputnya. Pada mode menghitung, *timer/counter* menghitung kejadian atau pulsa. Contohnya, penghitung biner memiliki 1 ms pulsa sebagai inputnya, periode waktu dapat diukur dengan memulai *counter* pada awal suatu kejadian dan berhenti pada akhir kejadian. Akhir dari penghitungan merupakan jumlah milidetik yang sudah berlalu selama kejadian. Ketika *timer/counter* digunakan sebagai penghitung, kejadian yang mau dihitung diberikan ke input penghitung biner, dan jumlah kejadian yang muncul dihitung.

Mikrokontroler memiliki 8-bit dan 16-bit timer/counter. Hal yang sangat penting adalah mengetahui kapan penghitung mencapai nilai maksimum penghitungan dan berulang kembali. Pada 8-bit, hal ini terjadi bila penghitungan mencapai nilai 255, dan pulsa berikutnya akan mengakibatkan penghitung mulai dari 0 lagi. Pada 16-bit, hal yang sama akan terjadi setalah mencapai 65535. Kejadian penghitung-ulangan ini sangat penting bagi program agar dapat secara akurat membaca hasilnya. Faktanya, penghitung-ulangan ini sangat penting karena *interrupt* disediakan sehingga muncul ketika *timer/counter* menghitung-ulang.

Sebagian besar jenis mikrokontroler memiliki dua buah 8-bit timer (Timer 0 dan Timer 2) dan satu buah 16-bit timer (Timer 1). Sedangkan untuk Mikrokontroler ATTiny 2313 sendiri hanya memiliki satu buah 8-bit timer (Timer 0) dan satu buah 16-bit timer (Timer 1)

II.7.1 Timer/Counter *Presaler* Dan Pemilihan Input

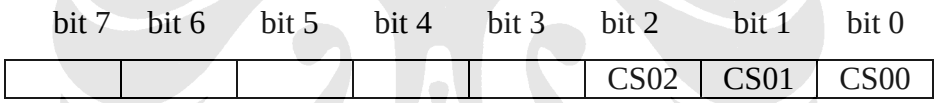
Timer/counter unit dapat menggunakan berbagai jenis frekuensi internal yang diturunkan dari sistem *clock* sebagai inputnya, atau dapat juga mendapatkan input dari pin eksternal. *Timer Counter Control Register* (TCCRx) yang berhubungan dengan timer memiliki bit *Counter Select* (CSx2, CSx1, dan CSx0) yang mengatur input mana yang digunakan sebagai *Counter Spesifik*.

Potongan program berikut menunjukkan sebuah contoh bagaimana menginisalisasi timer 0 untuk menggunakan sistem *clock* dibagi dengan 8 sebagai sumber *clock* (bit pemilih *counter* adalah tiga LSB dari TCCR0).

```
TCCR0 = 0x02; //timer 0 menggunakan sistem clock/8.
```

II.7.2 Timer 0

Timer 0 secara umum merupakan 8-bit timer, tetapi bisa berbeda tergantung mikrokontroler yang digunakan. Timer 0 memiliki fungsi umum *timer/counter*, tetapi lebih sering digunakan untuk membuat waktu dasar atau "denyut" untuk program. *Timer Control Counter Register* 0 (TCCR0) mengontrol fungsi timer 0 dengan memilih sumber *clock* untuk diberikan ke timer 0. Definisi bit untuk timer 0 adalah : bit 0 untuk CS00, bit 1 untuk CS01, bit 2 untuk CS02.



Gambar II. 7 : Definisi bit Timer 0 [4]

Tabel II. 11 : Definisi TCCR0 [4]

CS02	CS01	CS00	Interrupt Function
0	0	0	Stop, Timer 0 is stopped
0	0	1	System Clock, CK
0	1	0	System Clock / 8, CK / 8
0	1	1	System Clock / 64, CK / 64
1	0	0	System Clock / 256, CK / 256
1	0	1	System Clock / 1024, CK / 1024
1	1	0	External Pin T0, counts a falling edge
1	1	1	External Pin T0, counts a falling edge

II.7.3 Timer 1

Timer 1 merupakan *peripheral* yang lebih serbaguna dan lebih rumit dibandingkan dengan Timer 0. Timer 1 memiliki sebuah register 16-bit penangkap input dan dua buah register pembanding output. Register penangkap

input (*input capture register*-ICR) digunakan untuk mengukur ketebalan detak (*pulse width*) atau waktu penangkapan (*capturing time*). Register pembanding output (*output compare register*-OCR) digunakan untuk menghasilkan frekuensi atau detak dari timer/counter ke sebuah pin output di mikrokontroler.

Timer 1 juga secara konsep berbeda dari Timer 0. Timer 0 biasanya dihentikan, dimulai, di-*reset*, dan sebagainya dalam penggunaannya. Timer 1 biasanya dibiarkan berjalan. Hal ini membuat beberapa perbedaan yang harus diperhatikan dalam penggunaannya.

II.7.3.1 *Prescaler* Dan Pemilihan Timer 1

Timer 1 merupakan penghitung biner yang menghitung kelajuan atau interval waktu tergantung dari sinyal yang diberikan pada inputnya, seperti Timer 0. Timer 1 dikontrol melalui sebuah register yang disebut *Timer Counter Control Register* 1 (TCCR1). TCCR1 ini disusun oleh dua register yaitu, TCCR1A dan TCCR1B. TCCR1A mengatur mode pembanding dan mode penghasil ketebalan denyut (*pulse width modulation*-PWM) dari Timer 1. TCCR1B mengatur *prescaler*, input *multiplexer* dari Timer 1, dan juga mode *input capture*. Berikut ini adalah definisi bit TCCR1B :

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Gambar II. 8 : Definisi bit TCCR1B [3]

Tabel II. 12 : Deskripsi bit TCCR1B [4]

Bit	Fungsi
ICNC1	Input Capture Noise Canceller (1 = enable)
ICES1	Input Capture Edge Select (1 = rising edge, 0 = falling edge)
CTC1	Clear Timer/Counter on Compare Match (1 = Enable)
CS12	Counter Input Select Bits (definisi sama denga Timer 0)
CS11	
CS10	

Pemilihan bit TCCR1B mengatur input ke Timer 1 sama seperti Timer 0. Bit-bit pengaturnya menyediakan sinyal dengan cara yang sama seperti Timer 0.

II.7.3.2 *Timer 1 Input Capture Mode*

Mengukur periode waktu di Timer 0 melibatkan memulai waktu pada permulaan kejadian, memberhentikan pada akhir kejadian, dan membaca waktu kejadian dari *timer control register*. Aktivitas ini ditangani dengan cara yang berbeda pada Timer 1 karena Timer 1 selalu bekerja. Untuk mengukur kejadian, waktu pada Timer 1 ditangkap atau ditahan pada permulaan kejadian, waktu tersebut juga ditangkap pada akhir kejadian, dan keduanya dikurangkan untuk mendapatkan periode waktu kejadian. Aktivitas ini diatur oleh *Input Capture Register* (ICR1).

ICR1 adalah register 16-bit yang akan menangkap pembacaan aktual Timer 1 ketika mikrokontroler menerima sinyal tertentu. Sinyal ini dapat berupa *rising edge* atau *falling edge* yang diberikan pada *input capture* pin (ICP) mikrokontroler. Pilihan rising edge atau falling edge diatur pada *Input Capture Edge Select* bit (ICES1). Memberi nilai pada bit ICES1 dengan 1 akan mengakibatkan ICR1 menangkap waktu pada *rising edge* dan memberi nilai pada bit ICES1 dengan 0 akan membuat ICR1 menangkap waktu pada *falling edge*.

Karena hanya ada satu *capture register* pada Timer 1, data yang ditangkap harus segera dibaca segera setelah data tersebut ditangkap untuk mencegah data berikutnya yang ditangkap menimpa dan menghancurkan data sebelumnya. Untuk mencapai hal ini. Sebuah *interrupt* disediakan ketika data tersebut ditangkap oleh ICR1. Setiap *interrupt* terjadi, program harus menentukan apakah sinyal *interrupt* merupakan awal atau akhir suatu kejadian yang dihitung waktunya, sehingga program bisa memperlakukan data di ICR1 secara benar.

Timer 1 juga menyediakan fitur *Noise Canceller*, untuk mencegah gangguan yang tidak diinginkan pada sinyal yang diberikan ke ICP sehingga dapat menghindari penangkapan data pada waktu yang salah. Ketika fitur ini

aktif, ICP harus tetap pada level aktif (tinggi pada *rising edge*, atau rendah pada *falling edge*) untuk empat sampel yang saling berurutan, sebelum mikrokontroler akan memperlakukan pemicu sebagai legitimasi dan menangkap data. Hal ini mencegah noise memicu *capture register*. Mengaktifkan *input capture noise canceller* (ICNC1) di TCCR1B akan mengaktifkan fitur ini.

II.7.3.3 *Timer 1 Output Compare Mode*

Output Compare Mode digunakan oleh mikrokontroler untuk menghasilkan sinyal output. Output bisa merupakan gelombang kotak atau asimetris, dan bisa bervariasi frekuensi dan simetrinya. *Output compare mode* merupakan kebalikan dari *input capture mode*. Pada *input capture mode*, sinyal eksternal mengakibatkan waktu di timer ditangkap atau ditahan di ICR. Pada *output compare mode*, program mengeluarkan *output compare register*. Nilainya dibandingkan dengan nilai-nilai pada register *timer/ counter* dan *interrupt* muncul bila nilai keduanya cocok. Interrupt ini bertindak sebagai alarm yang mengakibatkan mikrokontroler mengeksekusi suatu fungsi, relatif terhadap sinyal yang dihasilkan saat dibutuhkan.

Untuk menghasilkan *interrupt*, *output compare mode* dapat secara otomatis men-set, membersihkan, atau *toggle* spesifik port pin output. Untuk Timer 1, *output compare mode* diatur oleh *Timer Counter Control Register* 1A (TCCR1A).

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Gambar II. 9 : Definisi bit TCCR1A [3]

COM1A0 & COM1A1 mengatur fungsi mode pembanding untuk OC1A (Output Compare pin). COM1B0 & COM1B1 mengatur fungsi mode pembanding untuk OC1B

Tabel II. 13 : Deskripsi bit TCCR1A [4]

COM1x1	COM1x0	Fungsi (x adalah A atau B)
0	0	Tidak ada output
0	1	Hasil yang dibandingkan *toggle line* OC1x
1	0	Hasil yang dibandingkan membersihkan line OC1x ke bit 0
1	1	Hasil yang dibandingkan men-*set* line OC1x ke bit 1

Bit pengontrol mode pembanding menentukan aksi apa yang akan diambil ketika *match* (kecocokan) timbul antara register pembanding dan register timer. Pin output yang bersangkutan dapat tidak terpengaruh, ter-*toggle*, ter-*set*, atau terbersihkan. Kecocokan juga mengakibatkan *interrupt* muncul. Tujuan dari ISR adalah untuk me-*reset* atau me-*reload* register pembanding untuk kecocokan berikutnya yang akan muncul.

II.7.3.4 Timer 1 Pulse Width Modulation (PWM)

Mode PWM adalah salah satu metode penyediaan konversi analog ke digital. PWM adalah skema dimana siklus kerja dari suatu gelombang kotak output dari mikrokontroler divariasikan untuk menyediakan output DC yang berbeda dengan menyaring bentuk gelombang output aktual untuk mendapatkan DC rata-rata. Pembedaan siklus kerja, atau proporsi siklus yang tinggi, akan membedakan tegangan DC rata-rata pada bentuk gelombang. Bentuk gelombang kemudian disaring dan digunakan untuk mengatur peralatan analog, membuat sebuah DAC (*Digital to Analog Converter*).

Salah satu metode membuat PWM dengan Timer 1 adalah dengan menggunakan *output compare register*, setiap kecocokan terjadi, pembedaan jumlah inkremen yang di-*reload* akan membuat bentuk gelombang PWM. Timer 1 menyediakan metode *built-in* untuk menghasilkan PWM tanpa harus menggunakan register pembanding. Timer 1 mengubah mode operasinya untuk menghasilkan PWM. Ketika beroperasi pada mode PWM, timer 1 menghitung atas dan bawah, sehingga sulit untuk menggunakan mode-mode lain. Selama mode PWM, timer 1 menghitung

dari nol sampai nilai atas dan turun lagi sampai nol. Nilai atas ditentukan oleh resolusi yang diinginkan. PWM menyediakan resolusi 8-bit, 9-bit, atau 10-bit sesuai yang ditentukan oleh PWM select bit di TCCR1A. Definisi PWM select bit adalah :

Tabel II. 14 : Deskripsi bit PWM dalam TCCR1A [4]

PWM Select Bit		**Resolusi PWM**	**Nilai Atas Timer**
PWM 11	**PWM 10**		
0	0	PWM Disable	
0	1	8-bit	255 (0xff)
1	0	9-bit	511 (0x1ff)
1	1	10-bit	1023 (0x3ff)

Tabel di atas menunjukkan resolusi yang dipilih akan menentukan nilai atas bagi counter untuk menghitung dari bawah sampai atas dan ke bawah lagi, dan juga akan mempengaruhi frekuensi dari bentuk gelombang PWM yang dihasilkan. Sebagi contoh, memilih resolusi 9-bit akan menghasilkan hitungan sampai 511 dan frekensi PWM dapat dihitung dengan cara (frekuensi sistem clock adalah 8 MHz) [4]:

$$f_{PWM} = f_{system\ clock} / (\text{prescaler} * 2 * \text{nilai atas}) \quad (II.1)$$

$$f_{PWM} = 8\ MHz / (8 * 2 * 511) = 978.5\ Hz \quad (II.2)$$

Resolusi adalah akurasi dari PWM. Dalam mode 8-bit, PWM dikontrol dalam 256, dalam mode 9-bit, PWM dikontrol dalam 512, dan dalam mode 10-bit, PWM dikontrol dalam 1024. Pada PWM, resolusi harus dibedakan dari frekuensi untuk menentukan pilihan optimum.

Siklus kerja aktual yang dioutputkan dalam mode PWM tergantung dari nilai yang diberikan kepada *output compare register*. Dalam mode normal PWM, ketika *counter* menghitung ke bawah, PWM men-*set* bit output dalam *match* dengan OCR, dan bila menghitung ke atas, PWM membersihkan bit output dalam *match* dengan OCR. Dalam keadaan ini,

memberi nilai OCR dengan, misalnya, 20% dari nilai atas akan menghasilkan 20 % bentuk gelombang siklus kerja.

Dimungkinkan juga untuk menyediakan PWM yang terbalik untuk aplikasi seperti mengatur terangnya LED yang terhubung langsung pada pin, memberikan *output compare register* sampai 80% nilai atas pada mode terbalik akan menghasilkan 80% gelombang kotak siklus kerja rendah. Men-*set* frekensi output yang tepat membutuhkan pemadupadanan *prescaler* dan resolusi untuk mendapatkan sedekat mungkin frekuensi yang diinginkan.

II.8 LCD

LCD digunakan sebagai peralatan output dan display. Untuk mengaktifkan LCD perlu diaktifkan fungsi LCD pada status register di prosesor. Hal ini bisa dilakukan dengan membuat :

```
#asm
   .equ __lcd_port=0x18 ;PORTB
#endasm
#include <lcd.h>
```

Kode ini dimasukkan ke bahasa pemrograman C di awal program. Kode ini menggunakan pengarah compiler "`#asm`" untuk memasukkan ke dalam program, instruksi bahasa pemrograman *assembly*. Pin yang digunakan adalah keseluruhan pin pada suatu port, kecuali pin 3. Pada mikrokontroler ATTiny, hanya port B yang dapat digunakan untuk menampilkan LCD. Untuk menggunakan port B bit yang diaktifkan adalah 0x18. Langkah berikutnya adalah menginisialisasi jumlah karakter setiap baris yang sesuai dengan LCD yang digunakan, pada ***main***(). Contohnya :

```
lcd_init(16);
```

Pada contoh ini jumlah karakter per baris yang digunakan adalah 16 karakter.