RINGKASAN

Ronny Ari Setiawan, Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya, Mei 2016, Rancang Bangun *Driver* Motor DC *Servo* Berbasis Komunikasi Serial pada Robot *Humanoid* KRSI, Dosen Pembimbing: Nanang Sulistiyanto dan Akhmad Zainuri.

Kontes Robot Seni Indonesia merupakan suatu ajang kompetisi perancangan, pembuatan, dan pemrograman robot yang disertai dengan unsur-unsur seni tari yang telah terkenal di Indonesia. Setiap tim diharuskan membuat sendiri robot otomatis berbentuk *humanoid.* Terdapat dua jenis motor DC *servo* yang sering digunakan sebagai aktuator robot *humanoid* yaitu motor DC *servo* yang dikendalikan secara serial dan paralel. Aktuator yang digunakan oleh Tim Robot Teknik Elektro Universitas Brawijaya adalah motor DC *servo* paralel. Namun penggunaan motor DC *servo* paralel memiliki kekurangan yaitu membutuhkan banyak pin untuk mengendalikan banyak motor DC *servo*. Berdasarkan permasalahan tersebut maka dalam penelitian ini dirancanglah sebuah kontroler untuk motor DC *servo* paralel agar dapat dikendalikan secara serial.

Metode yang digunakan yaitu komunikasi serial *multi-poin*. Komponen utama yang digunakan pada rangkaian *Serial Driver* yaitu mikrokontroler ATmega8 dengan eksternal *clock* 16 MHz. Protokol komunikasi serial yang digunakan yaitu komunikasi serial UART dengan *baudrate* 1 Mbps. Data yang ditransmisikan disusun membentuk suatu paket data yang terdiri dari 3 byte data yaitu karakter *header*, ID, dan Sudut. *Header* yang digunakan yaitu karakter '@'. ID berisi alamat dari masing-masing rangkaian *Serial Driver* yang berupa bilangan desimal 1-6. Sudut berisi nilai masukan pada rangkaian *Serial Driver* untuk menggerakkan motor DC *servo*, nilai yang diizinkan yaitu 0-120 desimal yang merepresentasikan sudut 0^o-120^o.

Hasil yang didapat dari penelitian ini yaitu rangkaian *Serial Driver* mampu membangkitkan sinyal PWM dengan selisih antara perancangan dan pengukuran yaitu 0-5 µs. Selisih sudut rata-rata antara perancangan dan pengukuran yang dihasilkan oleh masing – masing Motor DC Servo 1 sampai 6 secara berurutan yaitu 0,8 ^{o}, 7,7 ^{o} 1,2 ^{o}, 1,2^o, 0,9 ^{o} dan 7,6 ^{o}. Semakin banyak jumlah tahap yang digunakan untuk mengontrol motor DC servo mencapai sudut target maka gerakannya semakin halus.

Kata Kunci : *Serial Driver*, motor DC *servo, multi-point*.

SUMMARY

***Ronny Ari Setiawan**, Department of Electrical Engineering, Faculty of Engineering, Brawijaya University, May 2016. Designing of DC Servo Motor Driver Based on Serial Communication on Humanoid Robot KRSI, Academic Supervisor: Nanang Sulistiyanto and Akhmad Zainuri.*

Indonesian Arts Robot Contest is a competition for designing, implementing, and programming robots accompanied with elements of art and culture of Indonesia, especially the dance that has been well known in the motherland. Each team is required to create their humanoid robot that is capable to perform dance according to the music automatically. There are two types of DC servo motor which is often used as actuator on Humanoid Robot. They are DC servo motor that controlled serially and parallel. Based on these problems, in this study designed a controller for parallel DC servo motor to be controlled serially.

The method that used in Serial Driver circuit is multi-point serial communication. The main components that used in the Serial Driver circuit is ATmega8 microcontroller with 16 MHz external clock. Serial communication protocol that used is the UART with 1 Mbps baudrate. Transmitted data are arranged in a data packet. One data packet is 3 bytes in size and consisting of 3 characters. They are header, ID, and SUDUT. Header that used is '@' character. ID contains the address of each Serial Driver circuit which is have number 1-255. SUDUT contains input value for Serial Driver circuit to drive a DC servo motor which is have value 0-120 that represents the angle 0^o-120^o.

The results of this research are Serial Driver circuit capable to generate PWM signal with the difference between designing and measuring 0-5 µs. Each DC servo motor 1 until 6 has an average difference degree between designing and measuring 0.8^o, 7.7^o, 1.2^o, 1.2^o, 0.9^o and $7,6^o$ respectively. If DC servo motor uses much number of step to control it, then its motion is soft.

Keywords: Serial Driver, DC servo motor, multi-point.

PENGANTAR

Puji syukur Penulis panjatkan kepada Tuhan Yang Maha Esa karena berkat limpahan rahmat dan hidayat-Nya Penulis dapat menyelesaikan Laporan Skripsi yang berjudul ”Rancang Bangun *Driver* Motor DC *Servo* Berbasis Komunikasi Serial pada Robot *Humanoid* KRSI”. Laporan ini dibuat dengan tujuan untuk untuk memenuhi persyaratan memperoleh gelar Sarjana Teknik pada Jurusan Teknik Elektro Konsentrasi Teknik Elektronika Fakultas Teknik Universitas Brawijaya Malang.

Dalam penyusunan Laporan ini tidak sedikit hambatan yang penulis hadapi, namun penulis menyadari bahwa kelancaran dam penyusunan Laporan ini berkat bantuan, dorongan, dan bimbingan dari berbagai pihak baik secara langsung maupun tidak langsung, untuk itu penulis menyampaikan terima kasih yang sebesar-besarnya kepada:

1. Bapak, Ibu dan seluruh anggota keluarga, atas dukungan dan doa yang telah diberikan.
2. Yang terhormat Bapak M. Aziz Muslim, ST., MT., Ph.D. selaku Ketua Jurusan Teknik Elektro Universitas Brawijaya
3. Yang terhormat Ibu Nurrusa’adah selaku Ketua Kelompok Dosen Konsentrasi Elektronika Jurusan Teknik Elektro Universitas Brawijaya.
4. Yang terhormat Bapak Nanang Sulistiyanto selaku dosen pembimbing 1 yang selalu memberikan bimbingan, arahan, dan motivasi dalam penyusunan Skripsi ini.
5. Yang terhormat Bapak Akhmad Zainuri selaku dosen pembimbing 2 yang selalu memberikan dukungan, semangat, dan solusi dalam penyusunan Skripsi ini.
6. Teman-teman Tim Robot angkatan 2012 Fikrul, Agus, Firman, Septian, Sofyan, Reza, Ricky, Anggi, Wiwin, dan Juli atas dukungan dan bantuannya.
7. Teman-teman Lab. Desain dan Prototipe 2012 Juddin, Guntoro, Wildan, Bagus, Dianata, Bidin dan Rifqa atas dukungan, bantuan dan semangat yang telah diberikan.
8. Teman-teman Tim KRSI 2016 Surya, Itsna, Yuda, Dion, Wahyu, Tomy, dan Fauzi.
9. Teman-teman TEUB tercinta terutama teman-teman Paket B Konsentrasi Teknik Elektronika yang selalu memberikan semangat, dorongan dan bantuan pikiran.

Penulis berharap semoga laporan ini dapat memberikan manfaat dan dapat dijadikan referensi di masa yang akan datang. Penulis sadar bahwa laporan ini masih banyak kekurangan dan jauh dari sempurna, oleh karena itu kritik dan saran yang membangun penulis harapkan demi kesempurnaan laporan ini.

Malang, 1 Agustus 2016

Penulis

DAFTAR ISI

PENGANTAR i
DAFTAR ISI ii
DAFTAR TABEL iv
DAFTAR GAMBAR v
DAFTAR LAMPIRAN vi
BAB I PENDAHULUAN 1
1.1. Latar Belakang 1
1.2. Rumusan Masalah 2
1.3. Batasan Masalah 2
1.4. Tujuan 2
1.5. Manfaat 2
BAB II TINJAUAN PUSTAKA 5
2.1. Kontes Robot Seni Indonesia (KRSI) 5
2.2. Robot *Humanoid* 6
2.3. Komunikasi Serial 6
2.4. Motor DC *Servo* 10
2.5. Mikrokontroler ATmega8 11
2.5.1. Komunikasi USART 13
2.5.2. Pulse Width Modulation (PWM) 16
BAB III METODE PENELITIAN 19
3.1. Penentuan Spesifikasi Alat 19
3.2. Perancangan dan Pembuatan Alat 19
3.2.1. Diagram Blok 20
3.2.2. Perancangan Rangkaian *Mainboard Serial Driver* 21
3.2.3. Perancangan *Software* 22

3.2.4. Perancangan UART 23
3.2.5. Perancangan Paket Data 24
3.2.6. Perancangan PWM 25
3.2.7. Perancangan konversi sudut ke PWM 28
3.3. Pengujian Alat 29
3.3.1. Pengujian Transmisi Data 29
3.3.2. Pengujian Sinyal PWM 30
3.3.3. Pengujian Motor DC *Servo* 31
3.3.4. Keseluruhan Sistem 32
BAB IV HASIL DAN PEMBAHASAN 35
4.1. Pengujian Transmisi Data 35
4.2. Pengujian Sinyal PWM 36
4.3. Pengujian Motor DC *Servo* 38
4.4. Pengujian Keseluruhan Sistem 40
BAB V PENUTUP 45
5.1. Kesimpulan 45
5.2. Saran 45
DAFTAR PUSTAKA 46

DAFTAR TABEL

Tabel 3.1. Mode Operasi Timer 1 26
Tabel 3.2. Hasil perhitungan nilai OCR1x 28
Tabel 4.1. Hasil Pengujian Transmisi Data 36
Tabel 4.2. Data Hasil Pengujian Sinyal PWM 37
Tabel 4.3. Hasil Pengujian Pertama Sudut Pergerakan Motor DC *Servo* 39
Tabel 4.4. Data hasil pengujian pertama sudut motor DC *servo* pada lengan robot 41

DAFTAR GAMBAR

Gambar 2.1. Transmisi data paralel ke serial 7
Gambar 2.2. Proses transmisi data asynchronous..... 8
Gambar 2.3. Komunikasi Serial Mode *Simplex* 9
Gambar 2.4. Komunikasi Serial Mode *Full Duplex*..... 9
Gambar 2.5. Komunikasi Serial Mode Half Duplex 10
Gambar 2.6. Konfigurasi Pin Pengkabelan Motor *Servo* 11
Gambar 2.7. Sinyal kontrol motor DC *servo*..... 11
Gambar 2.8. Pin-pin ATmega8 kemasan 32-pin SMD 12
Gambar 2.9 Diagram blok arsitektur USART ATmega8 15
Gambar 2.10 Format frame data pada komunikasi serial asinkron 16
Gambar 2.11 Pulsa pada mode Fast PWM..... 17
Gambar 3.1 Diagram blok keseluruhan sistem..... 20
Gambar 3.2 Diagram blok *Serial Driver* 20
Gambar 3.3 Rangkaian *mainboard Serial Driver* 21
Gambar 3.4. Diagram alir software 23
Gambar 3.5. Format Paket Data 25
Gambar 3.6. Format frame 1 byte data..... 25
Gambar 3.7. Sinyal kontrol motor DC *servo*..... 26
Gambar 3.8. Skema Pengujian Transmisi Data..... 30
Gambar 3.8. Format *frame* pengiriman data..... 30
Gambar 3.9. Skema Pengujian Rangkaian Serial Controller..... 31
Gambar 3.10. Tampilan sinyal PWM pada *software* PCLAB PCSU1000..... 31
Gambar 3.11. Skema Pengujian Motor DC *Servo*..... 32
Gambar 4.1. Hasil Pembacaan Data dengan Osiloskop 35
Gambar 4.2. Hasil pembacaan sinyal PWM menggunakan osiloskop 37
Gambar 4.3. Grafik selisih sudut perancangan dan pengukuran 40
Gambar 4.3 Pergerakan Motor DC *servo* 4 (siku) (a) 1 tahap, (b) 20 tahap..... 42
Gambar 4.4 Pergerakan Motor DC *servo* 6 (pergelangan tangan) (a) 1 tahap, (b) 20 tahap 42
Gambar 4.5. Gerakan *Sembahan* (a) gerakan manusia dan (b) gerakan robot 43
Gambar 4.6. Gerakan *Encot* (a) gerakan manusia dan (b) gerakan robot..... 43
Gambar 4.7. Gerakan *Panggel* (a) gerakan manusia dan (b) gerakan robot..... 44
Gambar 4.8. Gerakan *Kengser* (a) gerakan manusia dan (b) gerakan robot..... 44

DAFTAR LAMPIRAN

Lampiran 1. Hasil Rancangan Alat 47

Lampiran 2. Pengujian motor DC *servo* pada lengan robot 50

Lampiran 3. Listing Program 54

Lampiran 4. Datasheet ATmega8 56

Lampiran 5. Datasheet LED SMD 0603 85

Lampiran 6. Datasheet Motor DC *Servo* HS5085 87

BAB I
PENDAHULUAN

1.1. Latar Belakang

Dewasa ini perkembangan dunia robotika telah mengalami peningkatan yang sangat pesat. Robot tidak lagi dianggap sebagai mesin namun juga produk teknologi yang memiliki kecerdasan dan dinilai efektif dalam mengerjakan tugas yang memiliki risiko tinggi yang sulit dijalankan oleh manusia. Selain itu, pemanfaatan robot tidak hanya terbatas pada bidang industri saja namun juga di berbagai bidang kehidupan manusia.

Perkembangan dunia robotika di Indonesia juga mengalami perkembangan pesat yang ditandai dengan semakin banyaknya penyelenggaraan Kontes Robotika oleh pemerintah. Salah satu kontes robotika yang paling bergengsi di Indonesia yaitu Kontes Robot Indonesia yang diselenggarakan oleh Direktorat Jendral Pendidikan Tinggi (Dirjen DIKTI). Dalam Kontes Robot Indonesia terdapat 4 bidang perlombaan yaitu Kontes Robot ABU Indonesia (KRAI), Kontes Robot Pemadam Api (KRPAI), Kontes Robot Seni Indonesia (KRSI), dan Kontes Robot Sepak Bola Indonesia (KRSBI) (Dikti, 2014:5).

Kontes Robot Seni Indonesia merupakan suatu ajang kompetisi perancangan, pembuatan, dan pemrograman robot yang disertai dengan unsur-unsur seni dan budaya bangsa Indonesia khususnya seni tari yang telah terkenal di Indonesia. Setiap tim diharuskan membuat sendiri robot otomatis berbentuk humanoid yang mampu melakukan gerakan tari mengikuti alunan musik pengiring saat lomba berlangsung. Beberapa poin penilaian yang dilakukan dewan juri berdasarkan panduan Kontes Robot Seni Indonesia tahun 2015 meliputi kepiawaian gerakan tari di setiap zona, sinkronisasi gerakan dengan musik, dan keatraktifan gerakan serta keluwesan robot dalam melakukan tarian. Robot dapat dirancang menggunakan aktuator gerak berupa elektromotor, sistem *pneumatic* maupun sistem hidrolik (Dikti, 2014:7).

Aktuator gerak jenis elektromotor yang sering digunakan pada robot humanoid terdapat dua jenis yaitu motor DC *servo* yang dikontrol secara paralel dan motor DC *servo* yang dikontrol secara serial. Kedua jenis motor DC *servo* tersebut memiliki karakteristik masing-masing dari segi torsi dan kecepatannya. Motor DC *servo* paralel memiliki torsi 4,3 kg.cm dan kecepatan 0,13 s/60^o sedangkan motor DC *servo* seri memiliki torsi 2 kg.cm dan kecepatan 0,269 s/60^o (Lee, 2007:1).

Jenis aktuator yang digunakan oleh tim Robot Teknik Elektro Universitas Brawijaya adalah motor DC *servo* paralel. Hal ini dikarenakan robot yang dibuat memiliki massa

cukup besar sehingga membutuhkan aktuator yang memiliki torsi hingga mencapai 4,3 kg.cm. Namun, penggunaan motor DC *servo* paralel memiliki beberapa kerugian yaitu membutuhkan banyak pin mikrokontroler untuk mengontrol lebih dari satu motor DC *servo* karena setiap motor DC *servo* paralel membutuhkan satu pin mikrokontroler dan membutuhkan kabel yang cukup panjang untuk menjangkau sendi terjauh dari mikrokontroler.

Berdasarkan masalah-masalah tersebut maka dalam skripsi ini dirancang sebuah *driver* untuk motor DC *servo* paralel agar dapat dikendalikan secara serial. Dengan menggunakan sistem ini diharapkan instalasi kabel pada robot menjadi lebih mudah, simpel dan rapi serta gerakan robot menjadi lebih luwes.

1.2. Rumusan Masalah

Berdasarkan latar belakang dapat disusun rumusan masalah sebagai berikut:

1. Bagaimana metode pengontrolan motor DC *servo* paralel agar dapat dikontrol secara serial.
2. Bagaimana merancang dan membuat kontroler untuk motor DC *servo* paralel agar dapat dikontrol secara serial.
3. Bagaimana unjuk kerja kontroler motor DC *servo* paralel secara serial dalam menerima data, membangkitkan sinyal PWM dan menggerakkan motor DC servo.

1.3. Batasan Masalah

Dengan mengacu pada permasalahan yang telah dirumuskan, maka hal-hal yang berkaitan dengan alat akan diberi batasan sebagai berikut:

1. Jumlah pergerakan yang diteliti berjumlah 6 yaitu pada satu bagian tangan saja.
2. Mekanik lengan robot yang digunakan tidak dibahas dalam penelitian ini.
3. Blok master yang digunakan tidak dibahas dalam penelitian ini.

1.4. Tujuan

Tujuan dari pembuatan alat ini adalah merancang dan membuat *driver* untuk motor DC *servo* paralel pada robot *humanoid* KRSI agar dapat dikontrol secara serial.

1.5. Manfaat

Manfaat yang dapat diambil dari penerapan alat ini yaitu dapat menggerakkan banyak motor DC *servo* secara serial sehingga menghemat penggunaan pin I/O

mikrokontroler, dapat meringkas instalasi kabel yang rumit dan panjang sehingga pergerakan robot menjadi lebih luwes.

BAB II
TINJAUAN PUSTAKA

2.1. Kontes Robot Seni Indonesia (KRSI)

Kontes Robot Seni Indonesia merupakan suatu ajang kompetisi perancangan dan pembuatan robot yang disertai dengan unsur-unsur seni dan budaya bangsa yang telah terkenal di bumi pertiwi. KRSI pertama kali diselenggarakan pada tahun 2009 yang mengangkat tema "Robot Penari Jaipong", tahun 2010 dengan mengangkat tema "Robot Penari Pendet", tahun 2011 dengan mengangkat tema "Robot Penari Kelono Topeng", tahun 2012 mengangkat tema "Robot Penari Piring", tahun 2013 mengangkat tema "Robot Anoman Duto", tahun 2014 mengangkat tema "Robot Penari Legong Keraton", dan pada tahun 2015 mengangkat tema "Bambangan Cakil". Untuk KRSI 2015, tema yang diangkat adalah "Robot Penari Legong Keraton". Kegiatan KRSI 2015 ini dilaksanakan bersamaan dengan pelaksanaan Kontes Robot Indonesia (KRI) tingkat Regional dan KRI tingkat Nasional pada tanggal 11-14 Juni 2015 yang dikoordinasi dan didanai oleh Direktorat Penelitian dan Pengabdian kepada Masyarakat, Direktorat Jenderal Pendidikan Tinggi Kementrian Pendidikan dan Kebudayaan bekerjasama dengan institusi Perguruan Tinggi yang ditunjuk (Dikti, 2014: 5).

Tujuan dari kontes robot ini adalah untuk menumbuh kembangkan kreatifitas dan minat para mahasiswa dalam teknologi maju, khususnya teknologi robotika yang selain diperuntukkan bagi industri juga diharapkan dapat membantu kegiatan manusia sehari-hari serta seni budaya khususnya seni tari. Setiap tim peserta yang terdiri dari 3 (tiga) mahasiswa dengan seorang dosen pembimbing, diwajibkan untuk membuat satu atau beberapa robot yang terkoordinasi untuk menampilkan seni budaya yang diinginkan sesuai tema kontes (Dikti, 2014: 20).

Robot penari yang digunakan harus menyerupai struktur tubuh manusia dengan tinggi 50 ±5 cm diukur di posisi kepala. Robot harus memiliki bagian yang dapat disebut sebagai sistem kaki, tubuh, tangan dan kepala. Jumlah derajat kebebasan masing-masing sistem robot minimal 21 (dua puluh satu). Berat robot total maksimum adalah 30kg. Rentang kaki atau tangan robot maksimal tidak boleh lebih 60 cm diukur dari ujung jari tangan/kaki kanan ke kiri ketika membuka tangan/kaki selebar-lebarnya. Lebar telapak kaki maksimum 150 cm^2 berbentuk elips, lingkaran atau persegi empat (Dikti, 2014: 22).

Gerak tari harus diselaraskan dengan irama musik pengiring tari "Bambangan Cakil". Waktu yang disediakan untuk setiap unjuk kebolehan tari dalam lomba ini adalah empat (4) menit sesuai dengan panjang atau durasi irama gamelan pengiring. Dalam waktu

empat (4) menit, musik pengiring akan berhenti sebanyak satu kali selama 10-15 detik. Setiap tim pada setiap game diberikan kesempatan *retry*. Setiap *retry* akan dikenakan hukuman pengurangan nilai (*penalty*) (Dikti, 2014:25).

2.2. Robot *Humanoid*

Dunia Robotika telah mengalami kemajuan yang sangat pesat. Saat ini robot dibuat tidak hanya digunakan untuk melakukan tugas-tugas yang bersifat otomatis namun juga untuk berinteraksi dengan manusia. Fungsi dari robot telah berubah sangat drastis dari otomasi di bidang industri menjadi interaksi dengan manusia. Sebagai contoh yaitu robot yang membantu manusia melakukan aktifitas sehari-hari seperti di kantor, rumah dan rumah sakit sangat diinginkan (Tsai, 2006:2002).

Seiring dengan perkembangan zaman, robot kini memiliki program dan fungsi yang lebih kompleks. Robot dapat dikelompokkan menjadi beberapa jenis sesuai dengan spesifikasinya, misalkan fungsi dan alat gerak. Sesuai fungsinya, ada beberapa jenis robot, diantaranya robot penjelajah, robot industri, robot pemadam api, robot medis, robot nano, robot perang, dan robot kompetisi. Jenis robot sesuai dengan alat geraknya adalah robot beroda dan robot berkaki.

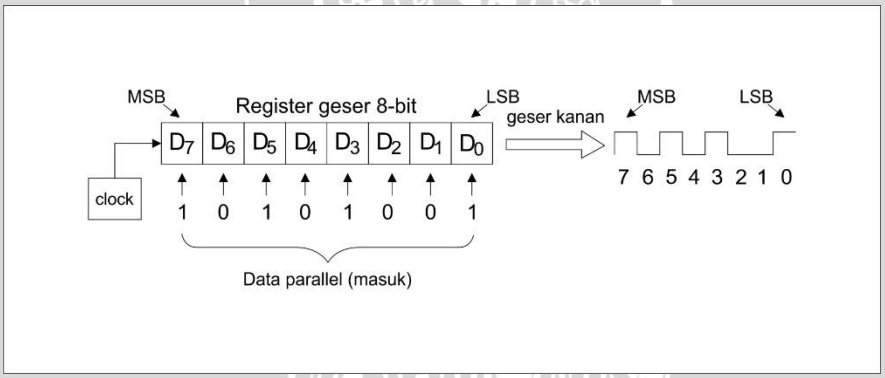
Robot *humanoid* adalah robot yang termasuk dalam kategori robot berkaki yang memiliki bentuk struktural menyerupai manusia. Robot *humanoid* memiliki penampilan keseluruhan yang didasarkan pada bentuk tubuh manusia, yaitu memiliki dua buah kaki, dua buah tangan, badan dan kepala. Robot *humanoid* memiliki stuktur tangan yang tersusun atas tiga tulang sendi yaitu sendi pada bahu, siku dan pergelangan tangan. Bahu dapat bergerak bebas dengan sudut pergerakan 270^o *pitch*, 120^o *yaw* dan 180^o *roll*. Siku hanya dapat bergerak dengan sudut pergerakan sebesar 90^o *pitch* saja untuk mengimbangi bahu. Pergelangan tangan memiliki pergerakan yang bebas untuk mengimbangi siku dan bahu dengan sudut pergerakan 180^o *roll*, 180^o *pitch* dan 120^o *yaw* (Tsai, 2006:2003).

2.3. Komunikasi Serial

Komunikasi serial adalah komunikasi dimana data dikirimkan bit per bit. Sehingga apabila sebuah data 8-bit hendak dikirimkan, kedelapan bitnya akan ditransmisikan satu per satu secara berturut-turut melalui sebuah jalur. Hal ini berarti bahwa sebuah data harus dipecah menjadi bit-bit pembentuknya agar dapat ditransmisikan dan kemudian disusun kembali menjadi data yang sama ketika diterima. Komunikasi serial selain relatif lebih murah, juga memberikan jangkauan transmisi yang lebih panjang dari komunikasi

paralel. Bandingkan dengan komunikasi paralel yang hanya bisa 1 hingga 2 meter saja, dengan komunikasi serial maka jangkauan tersebut bisa berlipat-lipat ganda. Misalnya dengan menggunakan standar komunikasi serial EIA RS-232 yang lebih dikenal dengan standar RS-232, dapat melakukan transmisi sejauh kurang lebih 50 *feet* dengan *baud rate* 9600. Transmisi yang lebih jauh dapat dilakukan pada data *rate* yang lebih rendah, dan jarak transmisi menjadi lebih pendek pada data *rate* yang lebih besar dari 9600 baud.

Kita ketahui bahwa prosesor mengolah data secara paralel. Karena data dalam prosesor (CPU) diproses dalam bentuk paralel, maka transfer data *input/output* serial harus dimulai dan diakhiri dengan data paralel. Pada dasarnya konversi paralel ke serial mudah dilakukan. Data paralel dimuat ke dalam sebuah *register* geser (*shift register*), kemudian *register* geser diberikan *clock*. Data kemudian dikeluarkan dari *register* geser mulai LSB (1 bit untuk setiap siklus *clock*). Bit pertama dari sebuah transmisi serial adalah LSB (*least significant bit*). Bit kedua adalah bit LSB selanjutnya, dan seterusnya. Bit data yang terakhir adalah MSB (*most significant bit*). Konversi paralel ke serial ditunjukkan pada Gambar 2.1.



Gambar 2.1. Transmisi data paralel ke serial
Sumber : Syahrul, 2014: 458

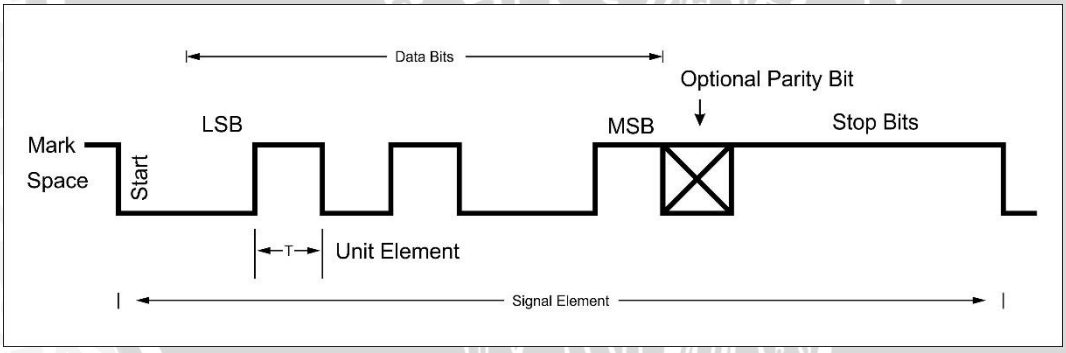
Penerimaan data serial dan konversinya ke data paralel merupakan operasi kebalikannya. Data serial digeser ke dalam sebuah register geser, kemudian setelah semua bit di-clock ke dalam register geser maka data diambil secara paralel untuk ditransfer ke prosesor. Jadi dengan kata lain digunakan dua register geser, yang pertama register jenis PISO (*parallel in parallel out*) dan di sisi yang lain digunakan jenis SIPO (*serial in parallel out*).

Peralatan atau *device* yang melakukan konversi paralel ke serial dan konversi serial ke paralel salah satunya disebut *universal asynchronous receiver-transmitter* (UART). Selain itu terdapat pula yang *disebut universal synchronous asynchronous receiver-*

transmitter (USART), dimana selain dapat melakukan komunikasi serial asinkron juga dapat dengan cara sinkron. Baik cara sinkron maupun asinkron keduanya sama-sama mempunyai keuntungan dan kelemahan masing-masing (Syahrul, 2014: 459).

Metode *Asynchronou*s

Metode *asynchronous* adalah metode pengiriman/penerimaan data secara serial dimana data yang dikirim/diterima tidak disinkronkan dengan sinyal *clock*. Pada metode asynchronous, data yang ditransmisikan memiliki bit penanda pada bit awal dan bit terakhir untuk melakukan sinkronisasi data dan juga kecepatan transmisi data dapat disesuaikan. Sebagai contoh pada proses pentransmisian karakter J dengan format ASCII 7 bit, ketika karakter tersebut dikirim, diawali oleh start bit berupa "space" (binary 0) kemudian diikuti parity bit dan satu atau lebih stop bit seperti ditunjukkan pada Gambar 2.2. Stop bit selalu berupa "mark" atau logika 1. Bagian penerima mendeteksi start bit melalui tidak adanya transisi dari mark hingga space kemudian menterjemahkan 7 bit setelahnya sebagai karakter. Jika terdapat lebih banyak karakter yang dikirimkan maka proses akan diulangi. Bagian pengirim dan penerima memiliki clock masing-masing, namun kecepatan yang digunakan harus sama (Sinnema, 1986: 30).



Gambar 2.2. Proses transmisi data asynchronous

Sumber: Sinnema, 1986: 30

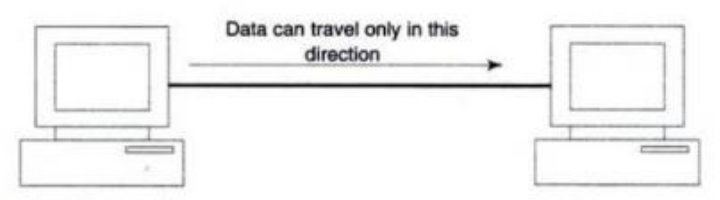
Metode *Synchronous*

Metode *synchronous* merupakan metode pengiriman/penerimaan data secara serial dimana data yang dikirim/diterima sinkron dengan sinyal *clock*. Pada metode synchronous tidak menggunakan start dan stop bit untuk melakukan sinkronisasi data. Metode synchronous menggunakan sumber clock yang sama pada bagian pengirim dan penerima untuk melakukan sinkronisasi dan identifikasi data. Penerima dapat mengenali kode unik yang terdapat pada bit data yang diterima kemudian mengunci dan mengidentifikasi data tersebut. Bagian penerima diberikan clock yang kecepatannya sama

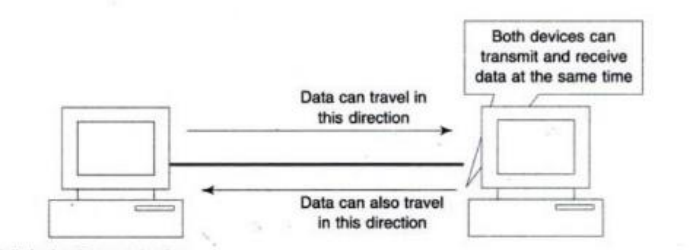
dengan bagian pengirim. Clock ini disebut juga dengan bit sinkronisasi (Sinnema, 1986: 32).

Terdapat tiga mode komunikasi serial yang biasa digunakan yaitu *simplex*, *half duplex*, dan *full duplex*. Komunikasi *simplex* merupakan komunikasi serial searah yang bersifat permanen. Komunikasi serial dengan mode *simplex* juga merupakan koneksi serial pertama antar komputer. Proses pengiriman atau penerimaan data pada mode *simplex* bersifat satu arah saja. Pengirim hanya bisa mengirim data dan penerima hanya bisa menrima data, sehingga tidak bisa dilakukan komunikasi interaktif antara pengirim dan penerima. Gambar 2.3 menunjukkan ilustrasi komunikasi *simplex*.

Komunikasi mode *full duplex* merupakan komunikasi serial dua arah yang dapat dilakukan secara bersamaan. Biasanya pada komunikasi mode *full duplex* menggunakan dua jalur data. Dalam mode ini, baik pengirim maupun penerima dapat melakukan transmisi data pada waktu yang sama. Contoh penerapan komunikasi *full duplex* yaitu pada teknologi telepon. Gambar 2.4 menunjukkan ilustrasi komunikasi *full duplex*.



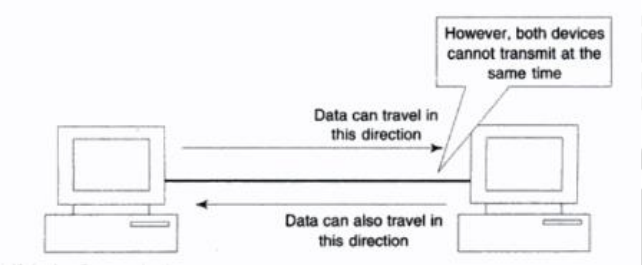
Gambar 2.3. Komunikasi Serial Mode *Simplex*
Sumber : Godbole, 2002: 65



Gambar 2.4. Komunikasi Serial Mode *Full Duplex*
Sumber : Godbole, 2002: 66

Dalam komunikasi *half duplex* antara pengirim dan penerima informasi berkomunikasi secara bergantian namun tetap berkesinambungan. Jalur yang digunakan dalam mode *half duplex* hanya satu, namun jalur tersebut dapat digunakan untuk mengirim dan menerima data secara bergantian. Biasanya salah satu bagian pengirim atau penerima akan memberikan kode tertentu untuk memulai komunikasi agar data yang ditransmisikan tidak bertabrakan dan dapat disampaikan pada masing-masing bagian.

Contoh pengunaan komunikasi mode *half duplex* yaitu pada *walkie talky* dimana salah satu pengguna harus menekan tombol untuk berbicara dan pengguna lain hanya dapat mendengarkan. Gambar 2.5 menunjukkan ilustrasi komunikasi *half duplex* (Sinnema, 1986: 34).



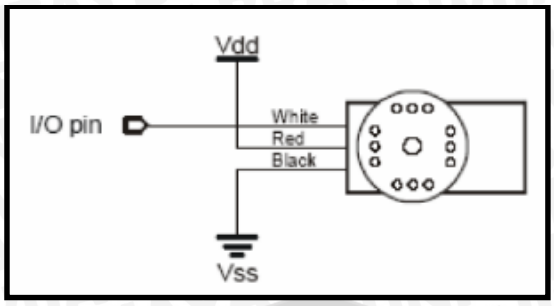
Gambar 2.5. Komunikasi Serial Mode Half Duplex
Sumber : Godbole, 2002: 67

2.4. Motor DC *Servo*

Berbeda dengan motor DC dan motor Stepper, motor DC *servo* adalah sebuah motor dengan sistem *closed feedback* di mana posisi dari motor akan diinformasikan kembali ke rangkaian kontrol yang ada di dalam motor DC *servo*. Motor DC *servo* terdiri atas sebuah motor, serangkaian *internal gear*, potensiometer dan rangkaian kontrol. Potensiometer berfungsi untuk menentukan batas sudut putaran motor DC *servo*. Sedangkan sudut sumbu motor DC *servo* diatur berdasarkan lebar pulsa yang dikirim melalui kaki sinyal dari kabel motor.

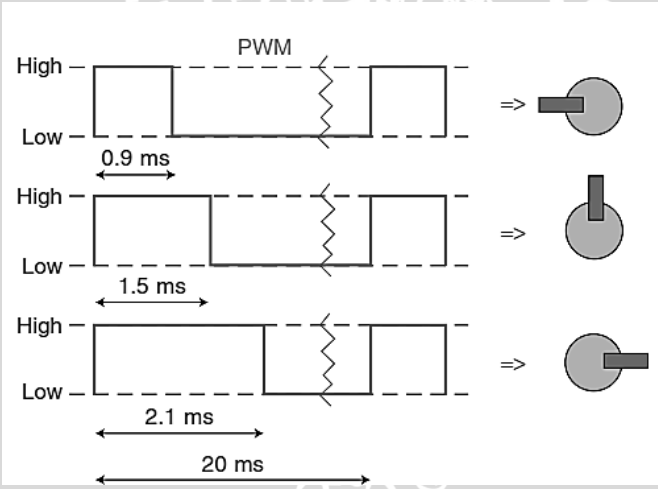
Motor *servo* mampu bekerja dua arah (CW dan CCW), arah dan sudut pergerakan rotornya dapat dikendalikan hanya dengan memberikan pengaturan *duty cycle* sinyal PWM pada bagian pin kontrolnya. Motor *servo* merupakan motor yang berputar lambat, dimana biasanya ditunjukkan oleh rate putarannya yang lambat, namun demikian memiliki torsi yang kuat karena roda gigi dalamnya. Karakteristik motor *servo* adalah sebagai berikut :

- Memiliki 3 jalur kabel : *power*, *ground*, dan *control* seperti ditunjukkan dalam Gambar 2.6.
- Pin kontrol untuk mengendalikan posisi.
- Operasional dari motor *servo* dikendalikan oleh sebuah pulsa selebar ± 20 ms, dimana lebar pulsa antara 0.5 ms dan 2 ms menyatakan akhir dari range sudut maksimum.
- Konstruksi di dalamnya meliputi *internal gear*, potensiometer, dan *feedback control*.



Gambar 2.6. Konfigurasi Pin Pengkabelan Motor *Servo*
Sumber : Syahrul, 2014:449.

Pengaturan sudut motor DC *servo* diperlukan untuk mengetahui gerakannya dan pulsa yang harus diberikan untuk bergerak ke kanan atau bergerak ke kiri. Gambar 2.7 menunjukkan teknik PWM untuk mengatur sudut motor DC *servo*. Dalam Gambar 2.7 diasumsikan bahwa saat diberikan sinyal periodik dengan lebar 0,9 ms maka motor DC *servo* akan bergerak dengan sudut 0^o, jika diberi sinyal 1,5 ms maka motor DC *servo* akan bergerak dengan sudut 90^o, dan jika diberi sinyal 2,1 ms maka motor DC *servo* akan bergerak dengan sudut 180^o.



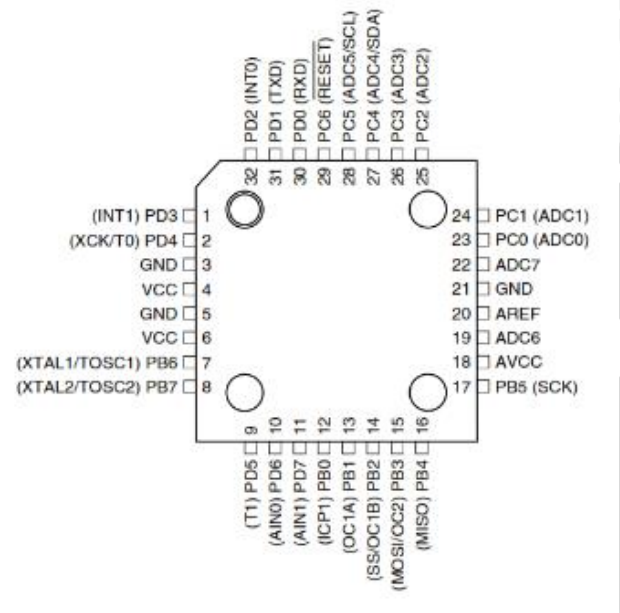
Gambar 2.7. Sinyal kontrol motor DC *servo*
Sumber : Pinckney, 2006: 28

2.5. Mikrokontroler ATmega8

ATmega8 merupakan salah satu jenis mikrokontroler CMOS 8-bit dengan konsumsi daya rendah buatan Atmel yang berbasis arsitektur RISC (*Reduced Instruction Set Computer*). ATmega8 memiliki *throughput* mendekati 1 MIPS per MHz sehingga mampu mengeksekusi banyak instruksi hanya dalam satu siklus *clock*. Beberapa periferal yang dimiliki Atmega8 yaitu dua timer/counter dengan resolusi 8 bit, satu timer/counter dengan resolusi 10 bit, tiga biah channel PWM, 8 channel ADC dengan resolusi 10 bit, antarmuka

komunikasi serial USART, antarmuka komunikasi serial SPI, dll. Tegangan kerja yang dimiliki ATmega8 yaitu 4,5 V – 5,5 V. Jumlah pin yang dimiliki ATmega8 yaitu 28 pin untuk kemasan PDIP dan 32 pin untuk kemasan LQFP (Atmel, 2009:1).

Pin-pin pada Atmega8 dengan kemasan 32-pin LQFP ditunjukkan oleh Gambar 2.8.



Gambar 2.8. Pin-pin ATmega8 kemasan 32-pin SMD

Sumber : Atmel, 2009: 2

Dari gambar diatas dapat dijelaskan fungsi masing-masing pin Atmega8 sebagai berikut:

1. VCC merupakan *pin* yang berfungsi sebagai masukan catu daya.
2. GND merupakan *pin Ground.*
3. *Port* B (PB0…PB7) merupakan *pin input/output* dua arah dan *pin* fungsi khusus, seperti Port B0 (ICP1 (*Timer/counter input capture pin*), Port B1 (OC1A (*Timer/counter* 1 *output compare* A *match output*)), Port B2 (OC1B (*Timer/counter* 1 *output compare* B *match output*) dan SS (SPI *slave select input*)), Port B3 (OC2 (*timer/counter 2 compare match output*) dan MOSI (SPI *bus master output/slave input*)), Port B4 (MISO (SPI *bus master input/slave output*)), Port B5 (SCK (SPI *bus serial clock*)), Port B6 (TOSC1 (*Timer Oscilator* pin1) dan XTAL1), Port B7 (TOSC2 (*Timer oscillator* pin2) dan XTAL2).
4. *Port* C (PC0…PC6) merupakan *pin input/output* dua arah dan *pin* fungsi khusus, seperti Port C0-C3 (ADC), Port C4 (ADC dan SDA (*Two-Wire* serial bus data input/output line), Port C5 (ADC dan SCL (*Two-Wire serial bus clock line*)) dan Port C6 merupakan pin reset.

5. *Port* D (PD0...PD7) merupakan saluran masukkan/keluaran dua arah dan juga mempunyai fungsi khusus. Fungsi khusus dari Port D diantaranya adalah : Port D0 (RXD (USART *input pin*)), Port D1 (TXD (USART *output pin*)), Port D2 (INT0 (Eksternal *interupt 0 input*)), Port D3 (INT1 (Eksternal *interupt 1 input*)), Port D4 (T0 (*timer/counter0* eksternal *counter input*) & XCK (USART eksternal *clock input/output*), Port D5 (T1 (*timer/counter* eksternal *counter input*)), Port D6 (AIN0 (*Analog comparator positive input*) & INT2 (*External interrupt 2 input*), Port D7 (AIN1 (*Analog comparator negative input*) & (OC0 (*Timer/counter0 output compare match output*)).
6. ADC6-ADC7 merupakan pin masukan ADC
7. *Reset* merupakan *pin* yang digunakan untuk mereset mikrokontroler
8. XTAL1 dan XTAL3 merupakan *pin* masukan *clock* eksternal.
9. AVCC merupakan *pin* masukan tegangan ADC.
10. AREF merupakan *pin* masukan tegangan referensi ADC (Atmel, 2009:5).

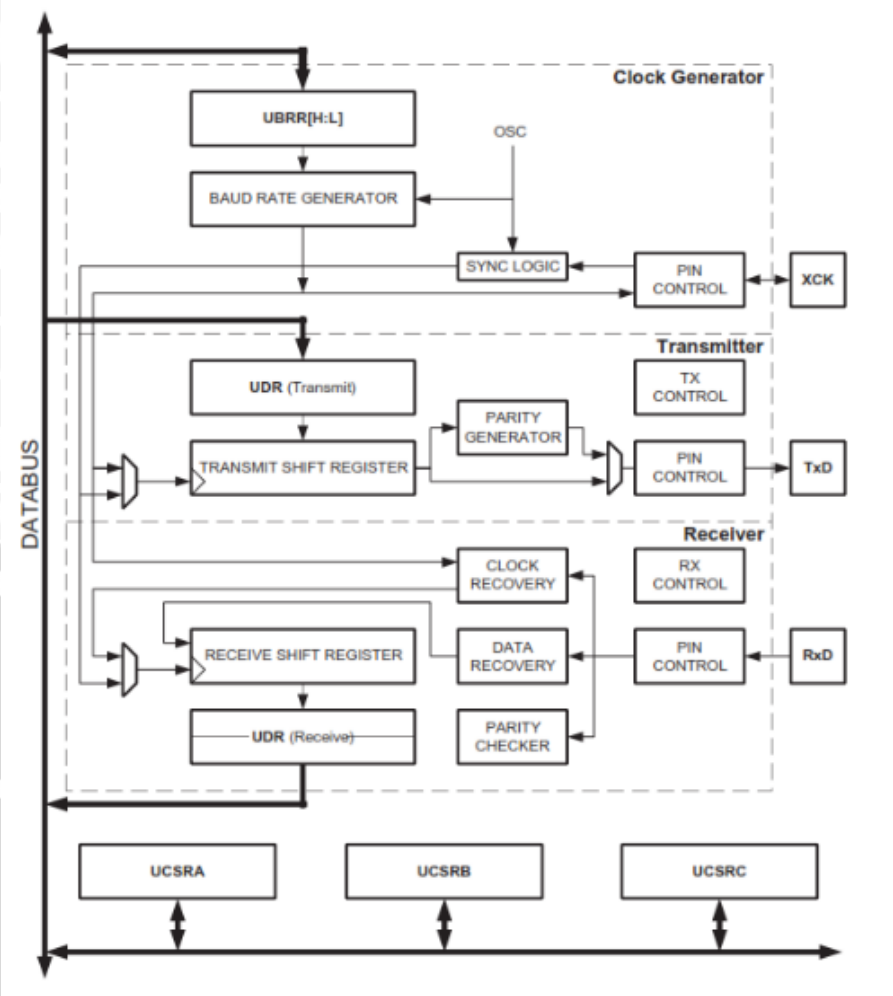
2.5.1. Komunikasi USART

Komunikasi USART (*Universal Synchronous Asynchronous Receiver and Transmitter*) merupakan salah satu fasilitas komunikasi serial yang disediakan oleh mikrokontroler AVR ATmega8. Periferal USART ATmega8 dapat digunakan pada ketiga mode komunikasi serial yaitu simplex, half duplex maupun full duplex. Terdapat dua mode operasi yang tersedia yaitu komunikasi serial sinkron dan asinkron. Adapun fitur lain dari peripheral USART ATmega8 yaitu memiliki baud rate generator dengan resolusi tinggi, tersedia serial frame dengan 5,6,7,8 atau 9 data bit dan 1 atau 2 stop bit, deteksi data overrun dan framing error, double speed pada mode Asinkron, dll (Atmel, 2009: 133).

Komunikasi serial sinkron adalah komunikasi antara mikrokontroler dengan *device* lain dimana sinyal *clock* yang digunakan antara *transmitter* dan *receiver* berasal dari satu sumber *clock*. Sedangkan komunikasi serial asinkron masing-masing memiliki sumber *clock* sendiri. Komunikasi USART dapat dilakukan dalam mode *full duplex* (dua arah) antara *transmitter* dan *receiver*. Komunikasi serial yang banyak digunakan adalah mode *asinkron*, dimana untuk menjaga sinkronisasi antara *transmitter* dan *receiver* maka digunakan teknik pembingkaian data *(framing data)* menggunakan bit *start* dan *stop* pada awal dan akhir setiap *byte* data dalam rangkaian transmisinya. Kecepatan data untuk komunikasi serial asinkron jauh lebih lambat daripada komunikasi sinkron, tetapi

penanganannya lebih sederhana dan hanya menggunakan kawat tunggal antara *transmitter* dan *receiver*. Pada komunikasi serial sinkron, sinkronisasi antara *transmitter* dan *receiver* harus dijaga dengan sinyal "*sync*" yaitu dengan menggunakan sumber *clock* yang sama antara dua device. Kecepatan transfer datanya jauh lebih tinggi daripada teknik asinkron tetapi memerlukan dua saluran kawat yaitu untuk sinyal data dan sinyal *clock* (Syahrul, 2014:461).

Perangkat keras USART dapat dibagi menjadi tiga bagian/blok besar yaitu *transmitter, receiver* dan sumber *clock* seperti ditunjukkan pada Gambar 2.9. Bagian *Transmitter* berhubungan dengan pengiriman data pada pin TX. Perangkat yang sering digunakan seperti *register* UDR sebagai tempat penampung data yang akan ditransmisikan, *flag* TXC sebagai akibat dari data yang ditransmisikan telah sukses, dan *flag* UDRE sebagai indikator jika UDR kosong dan siap untuk diisi data yang akan ditransmisikan lagi. Bagian *Receiver* berhubungan dengan penerimaan data dari pin RX. Perangkat yang sering digunakan seperti *register* UDR sebagai tempat penampung data yang telah diterima, dan *flag* RXC sebagai indikator bahwa data telah sukses diterima. Bagian Pembangkit *Clock* berhubungan dengan kecepatan transfer data (*baud rate*), *register* yang bertugas menentukan *baud rate* adalah *register* pasangan UBRR. Bagian *Transmitter* berhubungan dengan pengiriman data pada pin TX. Perangkat yang sering digunakan seperti *register* UDR sebagai tempat penampung data yang akan ditransmisikan, *flag* TXC sebagai akibat dari data yang ditransmisikan telah sukses, dan *flag* UDRE sebagai indikator jika UDR kosong dan siap untuk diisi data yang akan ditransmisikan lagi. Bagian *Receiver* berhubungan dengan penerimaan data dari pin RX. Perangkat yang sering digunakan seperti *register* UDR sebagai tempat penampung data yang telah diterima, dan *flag* RXC sebagai indikator bahwa data telah sukses diterima. Bagian Pembangkit *Clock* berhubungan dengan kecepatan transfer data (*baud rate*), *register* yang bertugas menentukan *baud rate* adalah *register* pasangan UBRR (Winoto, 2010:147).



Gambar 2.9 Diagram blok arsitektur USART ATmega8

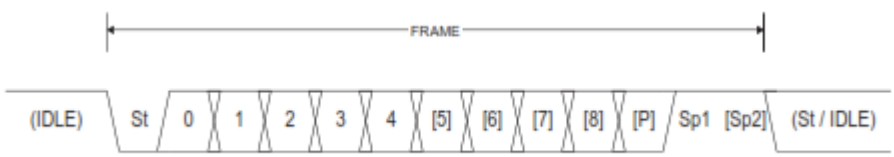
Sumber : Atmel, 2009: 133

Seperti yang sudah dijelaskan sebelumnya, terdapat teknik pembingkaian data (framing data) untuk menjaga sinkronisasi antara transmistter dan receiver pada komunikasi serial asinkron. Sebuah bingkai data serial (serial frame) dapat diartikan sebagai sebuah karakter yang tersusun oleh data bit, bit sinkronisasi (start bit dan stop bit) dan parity bit yang bersifat opsional. Format frame yang diijinkan memiliki susunan sebagai berikut:

- 1 star bit
- 5,6,7,8 atau 9 data bit
- Even atau parity bit, atau tidak dimasukkan
- 1 atau 2 stop bit

Sebuah frame data diawali dengan start bit kemudian diikuti data bit dengan total maksimum 9 bit yang diawali LSB hingga MSB. Jika parity bit diaktifkan, maka terletak

setelah data bit. Frame data diakhiri dengan 1 atau 2 stop bit. Gambar 2.10 menunjukkan ilsutrasi susunan frame pada komunikasi serial asinkron.



Gambar 2.10 Format frame data pada komunikasi serial asinkron

Sumber : Atmel, 2009: 137

Keterangan :

St : bit Start (berlogika low)

(n) : bit data (0 hingga 8)

P : bit Parity (odd atau even)

Sp : bit Stop (berlogika high)

IDLE : tidak ada transmisi data (berlogika high) (Atmel, 2009: 137).

2.5.2. Pulse Width Modulation (PWM)

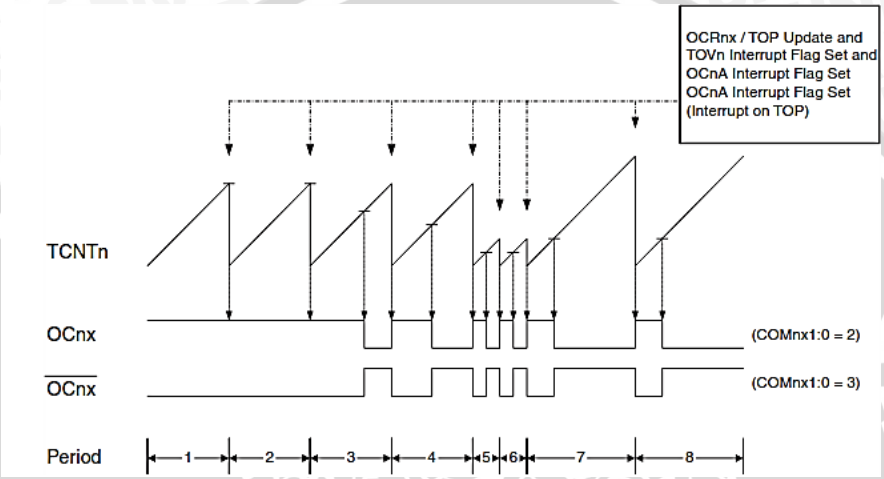
Salah satu fitur yang dimiliki AVR ATmega8 adalah adanya PWM (*Pulse Width Modulation*) yang berfungsi untuk menghasilkan pulsa-pulsa yang dapat diatur lebarnya berdasarkan penggunaan *Timer/Counter* yang sudah ada. Pulsa PWM merupakan salah satu cara yang baik dan mudah dalam mengemudikan atau mengatur kecepatan motor DC, mengatur pemacuan dan penyalaan led, dan mengontrol pergerakan motor DC *servo*.

Dalam mendesain pembangkit sinyal PWM dasarnya adalah menggunakan *Timer/Counter*. Pada mikrokontroler ATmega8 terdapat dua buah *Timer/Counter* 8-bit yaitu *Timer/Counter* 0 dan *Timer/Counter* 2, dan satu buah *Timer/Counter* 16-bit yaitu *Timer/Counter 1*. Salah satu mode *Timer/Counter* yang bisa digunakan untuk membangkitkan sinyal PWM adalah mode *fast* PWM, seperti yang terdapat dalam Gambar 2.11. *Timer/Counter* yang biasa digunakan yaitu *Timer/Counter* 1 yang memiliki resolusi 16 bit (Syahrul, 2014:348).

Timer/Counter1 dalam mode *fast* PWM digunakan untuk mengendalikan t_{on} dan t_{off} melalui *register* pembanding OCR1A atau OCR1B yang akan berakibat kepada besar *duty cycle* yang dihasilkan. Dalam mode *fast* PWM sifat cacahan *register* pencacah TCNT1 mencacah dari *BOTTOM* (0x0000) terus mencacah naik (*counting-up*) hingga mencapai *TOP* (nilai maksimum yang ditentukan sesuai resolusi yang diinginkan,

misalnya resolusi 10-bit maka nilai *TOP*=0x01FF) kemudian mulai dari *BOTTOM* lagi dan begitu seterusnya atau yang dinamakan *single slope* (satu arah cacahan).

Resolusi *fast* PWM dapat ditentukan dengan resolusi yang sudah tetap seperti 8-, 9-,10-bit atau bisa kita tentukan melalui *register* ICR1 atau OCR1A. Resolusi minimal yang diizinkan adalah 2-bit (ICR1 atau OCR1A diisi 0x0003), dan resolusi maksimal yang diizinkan adalah 16-bit (ICR1 atau OCR1A diisi 0xFFFF) (Winoto, 2010:133).



Gambar 2.11 Pulsa pada mode Fast PWM

Sumber : Atmel, 2009:90

UNIVERSITAS BRAWIJAYA

BAB III
METODE PENELITIAN

Kajian dalam penelitian ini bersifat aplikatif, yaitu perencanaan, pembuatan dan perealisasian alat agar dapat bekerja sesuai hasil perencanaan dengan mengacu pada rumusan masalah. Langkah-langkah yang dilakukan untuk merealisasikan sistem yang dirancang meliputi penentuan spesifikasi alat, perancangan dan pembuatan alat, pengujian dan pengambilan kesimpulan.

3.1. Penentuan Spesifikasi Alat

Spesifikasi alat secara global perlu ditentukan terlebih dahulu sebagai acuan untuk mendapatkan sistem yang sesuai dengan keinginan dan dapat bekerja secara efektif dan efisien. Alat yang dirancang hanya terbatas pada satu lengan robot yang tersusun atas 6 buah motor DC *servo*. Spesifikasi alat yang direncanakan yaitu sebagai berikut:

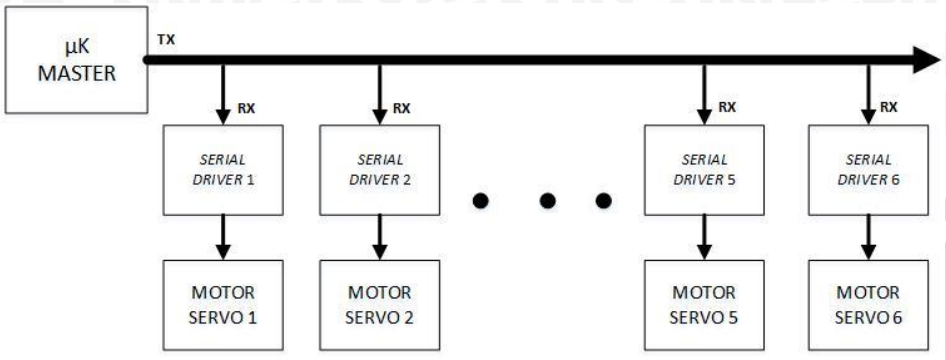
1. Sumber tegangan yang dibutuhkan memiliki range antara 4,8 - 6,0 V, oleh karena itu digunakan sumber tegangan berupa baterai *Lithium Polymer* 2S 7,4 V 2200 mAH.
2. Motor DC *servo* memiliki torsi minimal 1 kg.cm, oleh karena itu digunakan motor DC *servo* bertipe *Hitec* HS-5685 yang memiliki torsi minimal 3,6 kg.
3. Rangkaian catu daya mampu menurunkan tegangan sumber menjadi 6,0 V dengan arus 2 A, oleh karena itu digunakan rangkaian regulator *Switching* LM2576 *Simple Switcher*
4. Rangkaian kontroler mampu menghasilkan sinyal PWM dengan frekuensi 50 Hz dan mampu melakukan komunikasi serial dengan perangkat lain, oleh karena itu kontroler yang digunakan adalah mikrokontroler ATmega8

3.2. Perancangan dan Pembuatan Alat

Perancangan "Rancang Bangun *Driver* Motor DC *Servo* Berbasis Komunikasi Serial pada Robot *Humanoid* KRSI" dilakukan secara bertahap sehingga memudahkan dalam analisis setiap blok maupun secara keseluruhan. Adapun perancangan untuk membangun sistem ini yaitu sebagai berikut:

3.2.1. Diagram Blok

Diagram blok untuk keseluruhan sistem pada penelitian ini ditunjukkan pada Gambar 3.1. Metode yang digunakan yaitu metode komunikasi serial *multipoint* yang dapat menghubungkan banyak item hanya dengan menggunakan 2 kabel. Pada penelitian ini terdapat 6 buah item yang dihubungkan secara serial.



Gambar 3.1 Diagram blok keseluruhan sistem

Seperti ditunjukkan pada Gambar 4.1, keseluruhan sistem pada penelitian ini terdiri atas 1 buah blok *master* dan 6 buah blok *slave* yang terhubung secara seri. Namun blok *master* tidak dibahas dalam penelitian ini karena blok *master* dapat berupa perangkat keras PC ataupun mikrokontroler yang sifatnya untuk mengirim perintah ke blok *slave*. Keenam blok *slave* memiliki struktur yang sama, yang membedakan antara satu *slave* dengan *slave* lainnya yaitu ID yang ditanamkan untuk masing-masing *slave*. Penjelasan untuk masing-masing blok yaitu sebagai berikut:

a. *Master* akan mengirim paket data berupa ID *slave* dan nilai untuk menggerakkan motor DC *servo*
b. Data dengan ID yang sesuai dengan *Serial Driver* akan dimasukkan pada variabel untuk membangkitkan sinyal PWM penggerak motor DC *servo*. Jika tidak ada ID yang sesuai dengan *Serial Driver* maka variabel untuk membangkitkan sinyal PWM diisi dengan nilai sebelumnya.

Adapun diagram untuk blok *slave* ditunjukkan pada Gambar 3.2.

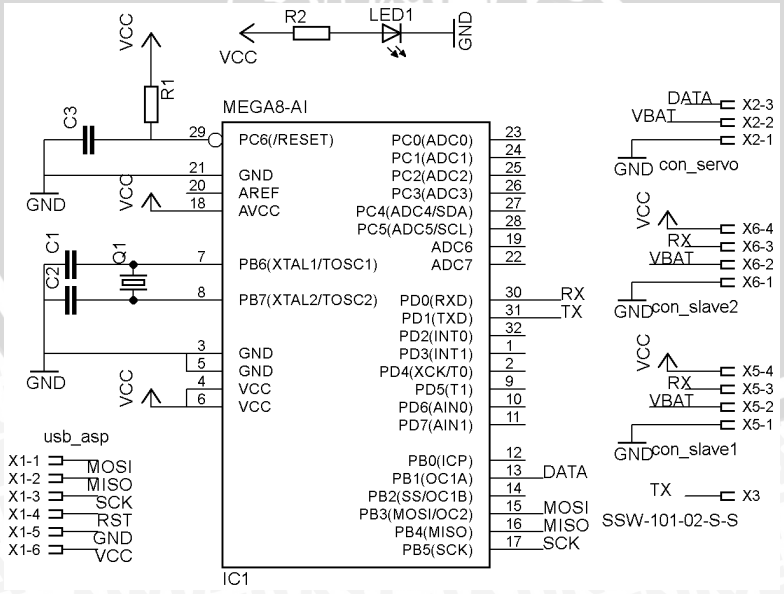
Gambar 3.2 Diagram blok *Serial Driver*

Penjelasan untuk cara kerja blok *slave Serial Driver* yaitu, pertama mikrokontroler menunggu apakah ada paket data yang diterima melalui pin RX. Paket data tersebut berisi ID dan nilai sudut. Paket data yang diterima kemudian diperiksa apakah memiliki ID yang sama. Jika paket data tersebut memiliki ID yang sama maka nilai sudut yang mengikutinya akan dikonversi menjadi nilai tertentu dan dimasukkan pada variabel yang mengatur lebar sinyal high PWM. Sinyal PWM yang dibangkitkan akan digunakan untuk menggerakkan motor DC *servo*. Jika ID-nya berbeda maka variabel yang mengatur lebar sinyal high PWM akan diisi nilai sebelumnya.

3.2.2. Perancangan Rangkaian *Mainboard Serial Driver*

Rangkaian *mainboard Serial Driver* merupakan rangkaian utama sistem *Serial Driver* pada penelitian ini. Rangkaian ini berupa minimum sistem ATmega8 yang dilengkapi dengan beberapa fitur yaitu 1 pin *Timer/Counter1* untuk antarmuka motor DC *servo*, 1 pin *receiver* antarmuka USART, 6 pin untuk antarmuka USB ASP *programmer* serta konektor catu daya. Gambar 3.3 berikut menunjukkan rangkaian *mainboard* ATmega8.

Pada rangkaian *mainboard* ATmega8 terdapat dua buah catu daya yaitu catu daya untuk motor DC *servo* dan catu daya untuk mikrokontroler. Catu daya untuk motor DC *servo* menggunakan tegangan 6 V dan untuk mikrokontroler menggunakan tegangan 5V.



Gambar 3.3 Rangkaian *mainboard Serial Driver*

Sumber *clock* eksternal yang digunakan yaitu *crystal* dengan frekuensi 16 MHz. Dua kapasitor non polar yang dihubungkan secara paralel pada kedua ujung kaki *crystal* memiliki kapasitansi sebesar 22 pF. Rangkaian reset yang digunakan terdiri dari satu buah resistor 47 kΩ dan kapasitor 1nF yang terhubung ke pin reset mikrokontroler ATmega8. Led yang dihubungkan dengan sumber tegangan digunakan sebagai indikator apakah terdapat tegangan yang menyuplai rangkaian. Agar led tidak rusak maka perlu diberi resistor untuk menghambat arus yang melewati led. Berdasarkan *datasheet* arus maksimum pada led yaitu 20 mA, sehingga dengan menggunakan hukum ohm dapat dihitung nilai resistor yang dibutuhkan seperti ditunjukkan pada Persamaan 3-2.

$$\boldsymbol{V = R\,x\,I} \tag{3-2}$$

$$\boldsymbol{R = \frac{V}{I}}$$

$$\boldsymbol{R = \frac{5\,V}{10\,mA}}$$

$$\boldsymbol{R = 250\ \Omega}$$

Karena resistor 250 Ω tidak tersedia di pasaran maka digunakan resistor 270 Ω. Jika nilai resistor 270 Ω dimasukkan kembali pada persamaan 3-2 maka akan diketahui arus yang melewati led yaitu 18 mA.

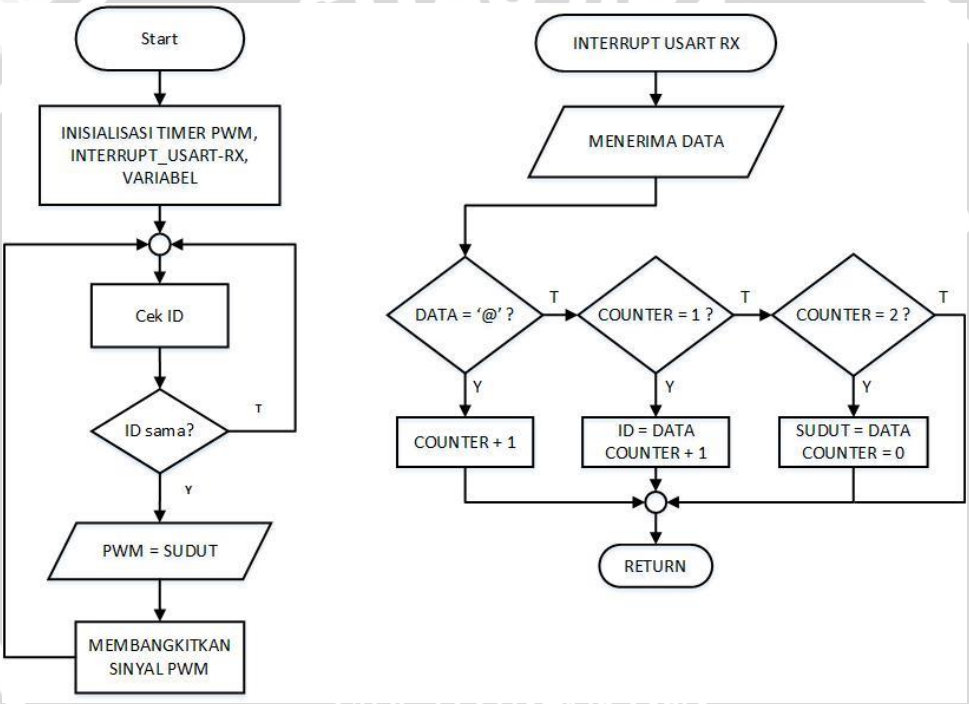
3.2.3. Perancangan *Software*

Perancangan *Software* dilakukan dengan merancang diagram alir (*flowchart*) terlebih dahulu. Diagram alir ini berfungsi sebagai alur kerja untuk setiap perangkat keras yang dikendalikan oleh mikrokontroler ATmega8 serta proses-proses perhitungan yang dikerjakan oleh mikrokontroler. Bahasa pemrograman yang digunakan dalam program utama adalah bahasa C dengan menggunakan *compiler* AVR Studio 7.0. Diagram alir perancangan *software* ditunjukkan oleh Gambar 3.4.

Proses yang pertama kali dilakukan adalah inisialisasi *Timer/Counter1*, *interrupt uart receiver*, dan variabel-variabel yang digunakan. *Timer/Counter1* diatur pada mode *fast* PWM untuk mengendalikan motor DC *servo*. *Interrupt* usart *receiver* digunakan untuk menginterupsi program ketika terdapat data yang diterima melalui pin RX ATmega8.

Setelah inisialisasi, proses selanjutnya yaitu menunggu apakah terdapat data yang diterima melalui pin RX, apabila ada maka sub rutin *interrupt* usart akan menginterupsi program utama. Data yang diterima berupa paket data sederhana dengan susunan karakter

'@' - data Id - data nilai sudut. Untuk memisahkan paket data tersebut dilakukan proses pemilihan kondisi. Apabila data yang diterima berupa karakter '@' maka variabel counter ditambah 1. Apabila data yang diterima bukan karakter '@' dan variabel counter bernilai 1 maka data akan dimasukkan pada variabel ID. Apabila data yang diterima bukan karakter '@' dan variabel counter bernilai 2 maka data akan dimasukkan pada variabel Sudut dan variabel counter diberi nilai 0. Setelah didapatkan data Id dan nilai sudut, proses selanjutnya yaitu memeriksa data pada variabel ID apakah memiliki Id yang sesuai dengan *Serial Driver*. Jika sesuai maka data pada variabel Sudut akan dikonversi menjadi nilai digital untuk membangkitkan sinyal pwm dan menggerakkan motor DC *servo*.



Gambar 3.4. Diagram alir software

3.2.4. Perancangan UART

Perancangan ini menjelaskan tentang penyusunan komunikasi UART ATmega8 yang digunakan pada penelitian ini. Dalam penelitian ini dibutuhkan sistem yang dapat menerima data secara cepat, akurat dan tidak membutuhkan banyak jalur data. Salah satu metode penerimaan data yang dapat digunakan yaitu komunikasi serial UART yang dimiliki oleh ATmega8. Komunikasi serial UART memiliki keunggulan yaitu hanya membutuhkan satu jalur data untuk melakukan komunikasi satu arah (*simpleks*) dan kecepatan transfer datanya dapat diatur hingga maksimal 1 Mbps. Untuk menghindari adanya data yang hilang pada saat proses transmisi maka digunakan fitur *interrupt* UART

receiver. Fitur ini akan menginterupsi proses yang sedang berlangsung apabila terdapat data yang diterima melalui pin RX ATmega8.

Terdapat dua faktor yang perlu diperhitungkan sebelum melakukan inisialisasi *interrupt* UART *receiver* pada program yaitu pengaturan *baudrate* dan *clock* eksternal karena kedua faktor tersebut sangat berpengaruh pada besarnya *error* keberhasilan data yang diterima. Berdasarkan tabel pemilihan *baudrate* dan frekuensi *clock* eksternal yang telah tersedia pada datasheet ATmega8, *baudrate* tertinggi yaitu 1 Mbps yang memiliki arti 1 bit data dapat dikirimkan dalam waktu 1 µs. Pada penelitian ini *baudrate* yang digunakan yaitu 1 Mbps karena membutuhkan kecepatan yang tinggi dalam proses transmisi data. Jika mengacu pada tabel tersebut, untuk mendapatkan *baudrate* 1 Mbps dapat menggunakan frekuensi *clock* eksternal 2 MHz, 7,3728 MHz, 8 MHz, 14,7456 MH, dan 16 MHz. Namun, frekuensi yang memiliki *error* 0% yaitu pada frekuensi 16 MHz. Sehingga frekuensi *clock* eksternal yang baik digunakan untuk memperoleh *baudrate* 1 Mbps dengan *error* 0 % yaitu 16 MHz.

Setelah menentukan *baudrate* dan frekuensi *clock* eksternal, langkah selanjutnya yaitu menghitung nilai *register* UBRR yang berfungsi untuk mengatur nilai *baudrate* pada program. Nilai *register* UBRR dapat dihitung menggunakan Persamaan (3-3).

$$\boldsymbol{UBRR = \frac{f_{clk\_I/O}}{16\,x\,BAUD} - 1} \qquad (3\text{-}3)$$

UBRR adalah *register* yang mengatur *baudrate*. $f_{clk\_I/O}$ adalah frekuensi clock eksternal. Baud adalah *baudrate* yang ingin digunakan. Dengan memasukkan nilai $f_{clk\_I/O}$ = 16 MHz dan *baudrate* 1 Mbps maka didapatkan nilai UBRR = 0.

3.2.5. Perancangan Paket Data

Perancangan ini menjelaskan tentang format paket data yang digunakan pada proses transmisi data dari blok master ke rangkaian serial kontroler. Protokol yang digunakan yaitu komunikasi serial UART. Untuk mempercepat proses transmisi data maka paket data yang digunakan harus sesederhana mungkin dan ukurannya sekecil mungkin. Terdapat dua parameter yang dibutuhkan oleh rangkaian *Serial Driver* agar dapat menggerakan motor DC *servo* yaitu Id dan Sudut. Sehingga perlu disusun paket data yang dapat memuat dua parameter tersebut.

Berdasarkan pertimbangan tersebut, disusunlah paket data yang berisi 3 *byte* data yaitu header, id, dan sudut seperti ditunjukkan pada Gambar 3.5. *Header* berfungsi sebagai penanda karakter pertama dari paket data yang ditransmisikan. Id berfungsi

sebagai alamat pengiriman data dimana setiap rangkaian *Serial Driver* memiliki alamat yang berbeda-beda. Sudut berfungsi sebagai data masukan untuk menggerakkan motor DC *servo*. Satu *byte* data menggunakan *frame* yang terdiri atas 10 bit data yaitu 1 *start* bit, 8 bit data, dan 1 *stop* bit seperti ditunjukkan pada Gambar 3.6. *Baudrate* yang digunakan yaitu 1 Mbps. Waktu yang dibutuhkan untuk mengirim satu bit data dapat dihitung menggunakan persamaan (3-12).

$$t_{bit\ data} = \frac{1}{baudrate} \quad (3\text{-}12)$$

$$t_{bit\ data} = \frac{1}{1000000}$$

$$t_{bit\ data} = 1\ \mu s$$

$t_{bit\ data}$ merupakan waktu yang dibutuhkan untuk mengirim 1 bit data. Baudrate adalah nilai *baudrate* yang digunakan. Jika 1 bit data dikirim dalam waktu 1µs maka waktu yang dibutuhkan untuk mengirim 1 paket data yang terdiri dari 30 bit data adalah 30 µs.

@	ID	SUDUT

Gambar 3.5. Format Paket Data

@ merupakan karakter header. ID merupakan alamat pengiriman berupa bilangan decimal 1-255. SUDUT merupakan variabel masukan untuk menggerakkan motor DC servo berupa bilangan decimal 0-120.

St	0	1	2	3	4	5	6	7	Sp

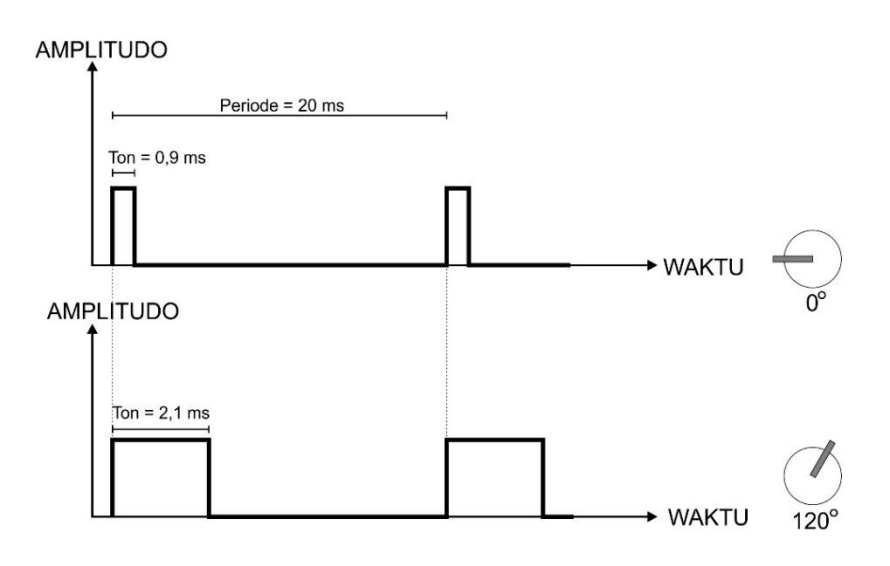
Gambar 3.6. Format frame 1 byte data

St merupakan *start bit* yang selalu berlogika *low*. 0-7 merupakan data 8 bit. Sp merupakan *stop bit* yang selalu berlogika *high*

3.2.6. Perancangan PWM

Perancangan ini menjelaskan pembangkitan sinyal PWM untuk mengendalikan motor DC *servo* menggunakan fitur *timer/counter* yang dimiliki ATmega8. Pada penelitian ini *timer/counter* yang digunakan yaitu *timer/counter1* karena memiliki resolusi 16 bit. Berdasarkan *datasheet*, motor DC *servo* tipe HS5085 dapat bergerak apabila diberikan sinyal PWM dengan frequensi 50 Hz (20 ms) dengan *duty cycle* antara 4,5 % - 10,5 %. Ilustrasi pemberian sinyal PWM ditunjukkan pada Gambar 3.7. Motor DC *servo* tipe HS5085 memiliki kemampuan untuk membentuk sudut dari 0^o – 120^o.

Untuk dapat bergerak membentuk sudut tertentu *duty cycle* dari sinyal PWM perlu diubah-ubah, dalam hal ini dengan mengatur lebar T_{ON} seperti pada Gambar 3.7. T_{ON} yang dibutuhkan agar dapat menghasilkan *duty cycle* 4,5 % adalah 900 µs dan *duty cycle* 10,5 % adalah 2100 µs. Sehingga bisa diasumsikan untuk bergerak pada sudut 0^o diperlukan T_{ON} 900 µs dan sudut 120^o diperlukan T_{ON} 2100 µs.



Gambar 3.7. Sinyal kontrol motor DC *servo*

Tabel 3.1. Mode Operasi Timer 1

No	Mode Operasi	TOP Value	Update OCR1A
0	Normal	0XFFFF	Immidiet
1	PWM Phase Correct 8-bit	0X00FF	TOP
2	PWM Phase Correct 9-bit	0X01FF	TOP
3	PWM Phase Correct 10-bit	0X03FF	TOP
4	CTC	OCRIA	Immidiet
5	Fast PWM 8-bit	0X00FF	BOTTOM
6	Fast PWM 9-bit	0X01FF	BOTTOM
7	Fast PWM 10-bit	0X03FF	BOTTOM
8	PWM Phase & Frequency Correct	ICR1	BOTTOM
9	PWM Phase & Frequency Correct	OCR1A	BOTTOM
10	PWM Phase Correct	ICR1	TOP
11	PWM Phase Correct	OCR1A	TOP
12	CTC	ICR1	Immidiet
13	Tidak digunakan	-	-
14	Fast PWM	ICR1	BOTTOM
15	Fast PWM	OCR1A	BOTTOM

Berdasarkan spesifikasi yang dibutuhkan maka *timer/counter* yang digunakan adalah *timer/counter1* dengan mode operasi nomor 14 seperti yang ditunjukkan pada Tabel 3.1. Mode operasi 14 merupakan *timer/counter1* pada mode *fast* PWM dengan resolusi 16 bit dan nilai TOP yang dapat diatur menggunakan *register* ICR1. Nilai TOP ini digunakan untuk menghasilkan sinyal PWM dengan frekuensi 50 Hz pada saat terjadi *overflow*. Lebar T_{ON} dapat diatur menggunakan *register* OCR1A pada saat terjadi *compare match*. Nilai TOP dan OCR1A perlu diketahui untuk membangkitkan sinyal PWM yang sesuai dengan tipe motor DC *servo* yang digunakan. Persamaan (3-4) dan (3-5) dapat digunakan untuk menghitung nilai TOP.

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{Nx(1+TOP)} \quad (3\text{-}4)$$

$$TOP = \frac{f_{clk\_I/O}}{N\ x\ f_{OCnxPWM}} - 1 \quad (3\text{-}5)$$

dimana $f_{OCnxPWM}$ adalah frekuensi PWM pada mode *output compare*. Dengan mengetahui frekuensinya, maka periode output *compare* adalah satu dibagi frekuensinya. $f_{clk\_I/O}$ adalah frekuensi *clock* yang digunakan, dalam penelitian ini menggunakan frekuensi *clock* eksternal 16 MHz. Nilai N adalah *prescaler* atau pembagi, dalam penelitian ini N yang digunakan adalah 8. TOP adalah nilai pada *Output Compare Register*. Sehingga dengan memasukkan nilai-nilai yang sudah diketahui didapatkan nilai TOP = 39999.

Untuk mendapatkan nilai OCR1A dapat menggunakan Persamaan *duty cycle* (3-6) dan (3-7):

$$DC = \left[\frac{(OCR1A+1)}{(TOP+1)}\right] x\ 100\% \quad (3\text{-}6)$$

$$OCR1A = \left[\frac{DC\ x\ (TOP+1)}{100\%}\right] - 1 \quad (3\text{-}7)$$

DC merupakan *duty cycle* sinyal pwm. Duty cycle juga dapat dicari dengan Persamaan (3-8):

$$DC = \frac{T_{ON}}{T_P} x\ 100\% \quad (3\text{-}8)$$

Dimana T_{ON} adalah lama waktu pada saat logika 1, dalam hal ini yaitu pada rentang 900 µs – 2100 µs. T_P adalah lama periode satu gelombang pulsa, dalam hal ini yaitu 20 ms. Dengan mensubtitusi Persamaan (3-8) ke dalam Persamaan (3-7) maka Persamaan (3-7) menjadi seperti Persamaan (3-9).

$$OCR1A = \left[\frac{T_{ON}}{T_P}\ (TOP + 1)\right] - 1 \qquad (3\text{-}9)$$

Dengan mensubtitusikan variabel-vaiabel yang sudah diketahui maka akan didapatkan nilai OCR1A. Tabel 3.2 menunjukkan beberapa nilai OCR1A dengan beberapa variasi nilai T_{ON}. Sehingga untuk mengatur lebar T_{ON} 900 µs nilai OCR1A yang digunakan adalah 1799 dan untuk lebar T_{ON} 2100 µs nilai OCR1A yang digunakan adalah 4199.

Tabel 3.2. Hasil perhitungan nilai OCR1x

T_{ON} (µs)	OCR1x
900	1799
1100	2199
1300	2599
1500	2999
1700	3399
1900	3799
2100	4199

3.2.7. Perancangan konversi sudut ke PWM

Perancangan ini menjelaskan tentang konversi dari sudut menjadi nilai OCR1A untuk membangkitkan sinyal PWM. Berdasarkan perancangan paket data sebelumnya, data yang diterima oleh rangkaian *Serial Driver* yaitu berupa nilai sudut dengan rentang 0 – 120, sehingga perlu dilakukan pengubahan dari nilai sudut menjadi nilai OCR1A yang sudah dihitung pada perancangan sinyal PWM. Berdasarkan perancangan sinyal PWM, lebar T_{ON} yang dibutuhkan yaitu 900 µs – 2100 µs. Dengan mengasumsikan nilai sudut 0^o sama dengan lebar T_{ON} 900 µs dan 120^o sama dengan lebar T_{ON} 2100 µs maka dapat dicari hubungan antara nilai sudut dengan nilai OCR1A. Jika nilai sudut dijadikan sebagai parameter masukan dan nilai PWM sebagai parameter keluaran maka hubungan antara nilai sudut dan nilai OCR1A dapat dicari menggunakan persamaan garis lurus yang melalui dua buah titik seperti Persamaan (3-10).

$$\frac{y-y_1}{y_2-y_1} = \frac{x-x_1}{x_2-x_1} \qquad (3\text{-}10)$$

Dimana y adalah keluaran proses konversi berupa nilai PWM, x adalah masukan proses konversi berupa nilai sudut, y_1 adalah batas bawah parameter keluaran, y_2 adalah batas atas parameter keluaran, x_1 adalah batas bawah parameter masukan dan x_2 adalah batas atas parameter masukan. Setelah parameter-parameter yang telah diketahui dimasukkan ke dalam persamaan maka akan didapat hubungan antara nilai sudut dengan nilai OCR1A seperti ditunjukkan oleh Persamaan (3-11).

$$y = 20x + 1799$$

$$Nilai\ OCR1x = 20(Nilai\ Sudut) + 1799 \quad (3\text{-}11)$$

Persamaan (3-11) kemudian dimasukkan ke program untuk melakukan konversi dari nilai sudut menjadi nilai OCR1A. Tabel 3.4 menunjukkan beberapa hasil konversi dari nilai sudut menjadi nilai OCR1A yang dihitung secara manual. Sehingga berdasarkan Tabel 3.3 dapat diketahui untuk bergerak pada sudut 0º menggunakan nilai OCR1A 1799 dan pada sudut 120º menggunakan nilai OCR1A 4199.

Tabel 3.3. Hasil perhitungan konversi nilai sudut ke nilai OCR1A

Sudut (º)	OCR1A
0	1799
20	2199
40	2599
60	2999
80	3399
100	3799
120	4199

3.3. Pengujian Alat

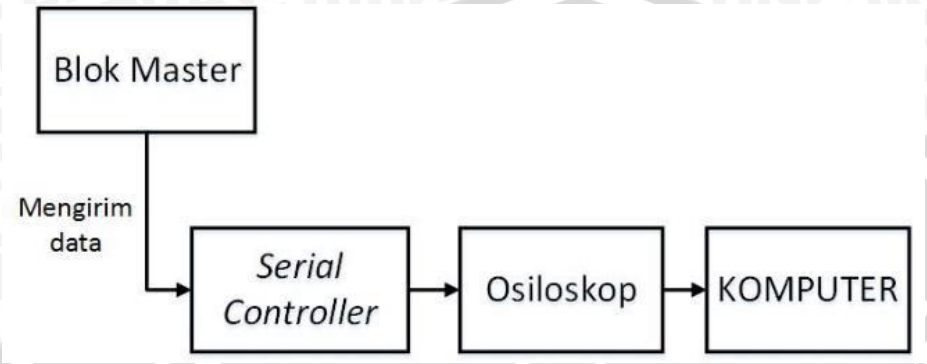
Untuk menganalisis kinerja alat apakah sesuai dengan yang direncanakan maka perlu dilakukan pengujian sistem. Pengujian dilakukan pada masing-masing blok perancangan *hardware* serta pengujian keseluruhan untuk mengetahui apakah *software* berjalan dengan baik atau tidak. Pengujian tersebut meliputi pengujian transmisi data, sinyal PWM, motor DC *servo*, motor DC *servo* pada lengan robot dan pengujian keseluruhan.

3.3.1. Pengujian Transmisi Data

Pengujian ini bertujuan untuk mengetahui keberhasilan transmisi data dari blok *master* ke rangkaian *Serial Driver*. Alat bantu yang digunakan dalam pengujian ini yaitu *usb asp programmer*, osiloskop digital dan komputer. Osiloskop digital yang digunakan adalah osiloskop Velleman PCLAB PCSU1000. Skema pengujian transmisi data ditunjukkan pada Gambar 3.7.

Pertama blok *master* mengirimkan karakter tertentu ke rangkaian *Serial Driver* secara bergantian. Karakter yang dikirim berupa karakter '@' dan '1 – 120'. Data dikirim ke rangkaian *Serial Driver* melalui pin Tx ATmega8 dengan *Baudrate* 1 Mbps. Data yang diterima kemudian dikirim kembali melalui pin Tx rangkaian *Serial Driver*. Probe Osiloskop dihubungkan pada pin Tx tersebut untuk membaca data yang dikirim. Setiap

data yang dikirim memiliki format *frame* seperti ditunjukkan pada Gambar 3.8. Data yang dibaca oleh osiloskop akan ditampilkan pada komputer dalam bentuk sinyal kotak yang merepresentasikan data dalam bentuk bilangan biner 10 bit. Data tersebut terdiri dari 1 bit start, 8 bit data, dan 1 bit stop. Bit *start* selalu berlogika *low* dan bit *stop* berlogika *high*.



Gambar 3.8. Skema Pengujian Transmisi Data

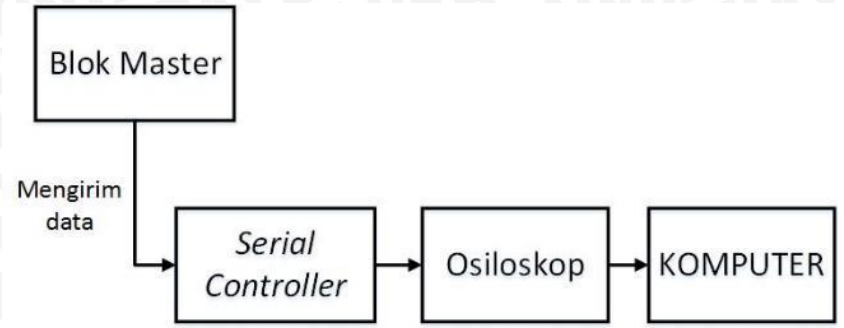
St	0	1	2	3	4	5	6	7	Sp

Gambar 3.8. Format *frame* pengiriman data.

St merupakan *start bit* yang selalu berlogika *low*. 0-7 merupakan data 8 bit. Sp merupakan *stop bit* yang selalu berlogika *high*

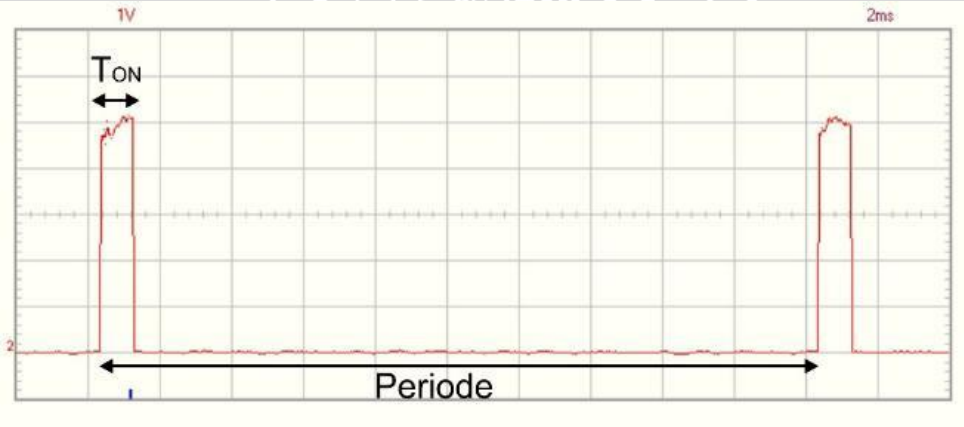
3.3.2. Pengujian Sinyal PWM

Pengujian ini bertujuan untuk mengetahui pengaruh perubahan nilai data yang diterima melalui komunikasi UART terhadap nilai Ton pada sinyal PWM penggerak motor DC *servo*. Alat bantu yang digunakan pada pengujian ini yaitu usb asp programmer, osiloskop digital dan komputer. Osiloskop digital digunakan untuk membaca bentuk sinyal PWM yang dihasilkan oleh rangkaian *Serial Driver* dan komputer digunakan untuk menampilkan hasil pembacaan osiloskop. Osiloskop digital yang digunakan adalah osiloskop Velleman PCLAB PCSU1000. Skema pengujian rangkaian *Serial Driver* ditunjukkan pada Gambar 3.9.



Gambar 3.9. Skema Pengujian Rangkaian Serial Controller

Pertama blok *master* mengirimkan paket data ke rangkaian *Serial Driver* dengan nilai SUDUT yang diubah-ubah. Nilai SUDUT yang diterima rangkaian *Serial Driver* dimasukkan pada persamaan konversi sudut ke PWM untuk membangkitkan sinyal PWM. Sinyal PWM dikeluarkan pada PORTC 0 mikrokontroler ATmega8. Probe Osiloskop dihubungkan pada PORTC 0 dan ground untuk melihat bentuk sinyal PWM yang dibangkitkan. Sinyal yang dibaca oleh osiloskop ditampilkan pada komputer melalui *software* PCLAB PCSU1000. Setelah sinyal muncul pada *software* PCLAB PCSU1000 kemudian data ditahan dan disimpan dalam bentuk gambar pada komputer. Dari gambar yang dihasilkan oleh *software* PCLAB kemudian didapatkan lebar T_{ON} dan Periode seperti ditunjukkan pada Gambar 3.10.

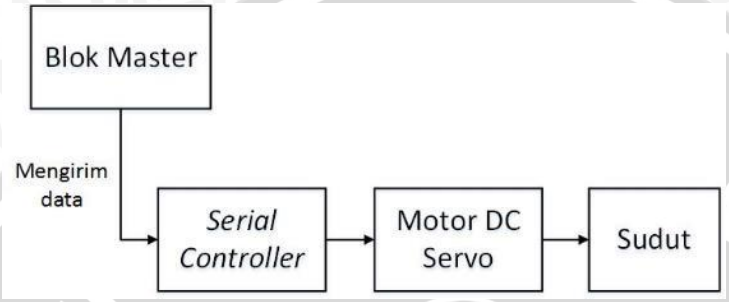


Gambar 3.10. Tampilan sinyal PWM pada *software* PCLAB PCSU1000

3.3.3. Pengujian Motor DC *Servo*

Pengujian ini bertujuan untuk mengetahui apakah motor DC *servo* dapat bergerak pada sudut tertentu sesuai dengan sinyal PWM yang diberikan. Peralatan pendukung yang digunakan yaitu baterai, modul regulator DC to DC *stepdown*, rangkaian blok *master*, dan busur derajat. Baterai yang digunakan *lithium polymer* 2S 7,4 V 2000 mAh. modul regulator DC to DC *stepdown* terdapat dua jenis yaitu modul regulator LM2576 *simple*

switcher dan regulator *linear* L7805. Modul regulator LM2576 *simple switcher* diatur hingga memiliki output 6 V untuk mensuplai motor DC *servo* dan modul regulator *linear* L7805 untuk mensuplai rangkaian serial controller. Busur derajat dipasang pada *horn* atau poros motor DC *servo* dan disesuaikan pada sudut 0^o. Busur derajat tersebut berfungsi untuk mengamati sudut pergerakan motor DC *servo*. Skema pengujian motor DC *servo* ditunjukkan pada Gambar 3.11.



Gambar 3.11. Skema Pengujian Motor DC *Servo*

Langkah yang dilakukan pada pengujian ini sama persis dengan pengujian PWM, namun parameter yang diamati yaitu pergerakan motor DC *servo* melalui busur derajat. Pengaturan sudut 0^o motor DC *servo* diwakilkan pada lebar T_{ON} 900 µs. Sudut teori dapat dihitung menggunakan persamaan linear konversi sudut (3-11) pada sub bab 3.2.7. Sudut yang terbaca melalui busur derajat kemudian dicatat untuk dianalisis. Sudut yang diuji yaitu 0^o – 120^o sebanyak 25 variasi sudut dengan interval 5^o. Pengujian dilakukan sebanyak 2 kali untuk tiap sudut yang diuji. Dari data hasil pengujian akan dicari besarnya selisih antara sudut perancangan dengan sudut pengukuran, kemudian dilakukan rata-rata selisih yang didapatkan untuk mengetahui karakteristik masing-masing motor DC servo yang digunakan.

3.3.4. Keseluruhan Sistem

Pengujian ini bertujuan untuk membuktikan bahwa tiap blok yang telah diuji dapat dirangkai menjadi satu sistem yang utuh dan dapat bekerja sesuai dengan perencanaan. Pengujian dibagi menjadi dua bagian yaitu pengujian sudut motor DC *servo* pada lengan robot dan pengujian motor DC *servo* pada lengan robot membentuk gerakan tarian. Peralatan pendukung yang digunakan yaitu baterai, modul regulator DC to DC *stepdown*, dan rangkaian blok *master*. Baterai yang digunakan *lithium polymer* 2S 7,4 V 2000 mAh. Modul regulator DC to DC *stepdown* yang digunakan terdapat dua jenis yaitu modul regulator LM2576 *simple switcher* dan regulator *linear* L7805. Modul regulator LM2576

simple switcher diatur hingga memiliki output 6 V untuk mensuplai motor DC *servo* dan modul regulator *linear* L7805 untuk mensuplai rangkaian *Serial Driver*.

Parameter yang diuji antara lain data dapat diterima oleh semua rangkaian *Serial Driver* dan semua motor DC *servo* dapat bergerak sesuai perintah dari blok *master*. Sistem yang akan diuji dirangkai berdasarkan diagram blok pada Gambar 3.1. Algoritma yang digunakan disusun berdasarkan flowchart pada Gambar 3.5. Sesuai dengan diagram blok perancangan, blok master akan mengirim paket data ke semua rangkaian *Serial Driver* dengan format '@'-'ID'-'SUDUT' seperti yang sudah dijelaskan pada sub bab 3.2.5. Rangkaian *Serial Driver* dengan ID yang sama akan mengkonversi SUDUT menjadi nilai pembangkit sinyal PWM dan menggerakkan motor DC *servo*.

Pada pengujian sudut motor DC *servo* pada lengan robot, pengujian dilakukan dengan 2 variasi yaitu menggerakkan motor DC servo pada sudut tertentu dengan 1 kali tahap dan 20 kali tahap. Pengujian diulang sebanyak 2 kali untuk setiap sudut yang diuji. Blok master akan mengirimkan data sudut 0^o – 120^o dengan interval 20^o untuk masing-masing jumlah tahap. Dari data hasil pengujian akan dicari besarnya selisih antara sudut perancangan dengan sudut pengukuran, kemudian dilakukan rata-rata selisih yang didapatkan untuk mengetahui pengaruh jumlah tahap yang digunakan terhadap gerakan motor DC servo dan sudut yang dihasilkan.

Pada pengujian gerakan tarian, blok master akan mengirimkan data gerakan tertentu seperti ditunjukkan pada gambar referensi. Parameter yang diuji yaitu apakah lengan robot dapat bergerak membentuk gerakan tarian seperti gambar yang dijadikan sebagai referensi.

UNIVERSITAS BRAWIJAYA

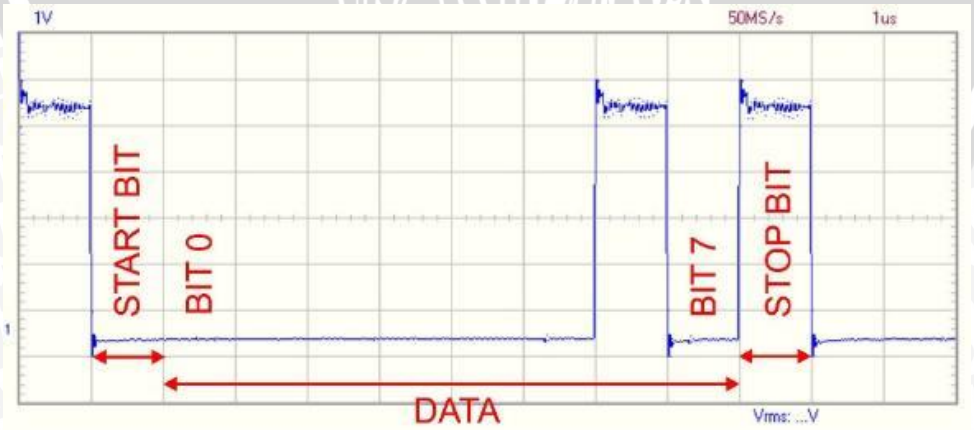
BAB IV
HASIL DAN PEMBAHASAN

Pengujian dan analisis dilakukan untuk menganalisis alat yang telah dirancang dan diimplementasikan apakah telah bekerja sesuai dengan perancangan. Pengujian dilakukan tiap-tiap blok dengan tujuan untuk mengamati apakah tiap rangkaian sudah sesuai dengan perancangan, kemudian dilanjutkan dengan pengujian secara keseluruhan sistem. Adapun pengujian yang telah dilakukan sebagai berikut:

1. Pengujian Transmisi Data
2. Pengujian PWM
3. Pengujian motor DC *servo*
4. Pengujian keseluruhan sistem

4.1. Pengujian Transmisi Data

Pengujian dilakukan dengan mengirim karakter '@', bilangan desimal 1-120 dengan 18 variasi data. Gambar 4.1 merupakan salah satu contoh hasil pembacaan karakter '@' oleh osiloskop. Karakter '@' merupakan kode ASCII dengan bilangan desimal 64. Oleh karena data yang ditampilkan osiloskop berupa sinyal kotak yang merepresentasikan bilangan biner, maka untuk mempermudah mengetahui apakah data sudah benar, data harus dikonversi menjadi bilangan biner terlebih dahulu. Jika '@' dikonversi menjadi kode biner maka hasilnya yaitu 0b01000000. Data diawali dengan bit *start* yang selalu berlogika *low* kemudian diikuti bit data ke 0 hingga ke 7 dan diakhiri dengan bit *stop* yang berlogika *high*. Hasil pengujian ditunjukkan pada Tabel 4.1.



Gambar 4.1. Hasil Pembacaan Data dengan Osiloskop

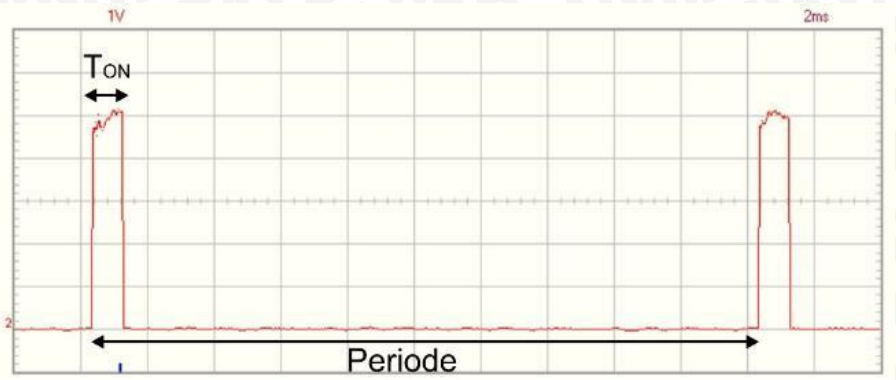
Tabel 4.1. Hasil Pengujian Transmisi Data

NO	DATA KIRIM (DEC)	DATA TERIMA (v)										KETERANGAN	
		St	0	1	2	3	4	5	6	7	Sp	biner (7-0)	DEC
1	1	0	5	0	0	0	0	0	0	0	5	00000001	1
2	2	0	0	5	0	0	0	0	0	0	5	00000010	2
3	3	0	5	5	0	0	0	0	0	0	5	00000011	3
4	4	0	0	0	5	0	0	0	0	0	5	00000100	4
5	5	0	5	0	5	0	0	0	0	0	5	00000101	5
6	6	0	0	5	5	0	0	0	0	0	5	00000110	6
7	10	0	0	5	0	5	0	0	0	0	5	00001010	10
8	20	0	0	0	5	0	5	0	0	0	5	00010100	20
9	30	0	0	5	5	5	5	0	0	0	5	00011110	30
10	40	0	0	0	0	5	0	5	0	0	5	00101000	40
11	50	0	0	5	0	0	5	5	0	0	5	00110010	50
12	60	0	0	0	5	5	5	5	0	0	5	00111100	60
13	70	0	0	5	5	0	0	0	5	0	5	01000110	70
14	80	0	0	0	0	0	5	0	5	0	5	01010000	80
15	90	0	0	5	0	5	5	0	5	0	5	01011010	90
16	100	0	0	0	5	0	0	5	5	0	5	01100100	100
17	110	0	0	5	5	5	0	5	5	0	5	01101110	110
18	120	0	0	0	0	5	5	5	5	0	5	01111000	120

Berdasarkan hasil pengujian yang dilakukan, data yang dikirim dari master berhasil diterima oleh rangkaian *Serial Driver* secara akurat. Penggunaan *baudrate* sebesar 1 Mbps tidak berpengaruh terhadap keakuratan hasil pengiriman data. Masing-masing bit dalam satu karakter yang terdiri dari 10 bit dikirimkan dalam waktu 1 µs. Sehingga untuk mengirimkan satu karakter membutuhkan waktu 10 µs.

4.2. Pengujian Sinyal PWM

Pengujian dilakukan dengan mengirimkan data nilai sudut sebanyak 25 variasi dimulai dari 0^o – 120^o. Sinyal PWM yang diuji berdasarkan nilai sudut yang dikirim dari blok *master* dengan rentang sudut 0-120^o. Nilai sudut kemudian dikonversi menggunakan persamaan 3.11 dan menghasilkan nilai OCR1A dengan rentang 1799-4199. Nilai – nilai tersebut dipilih karena mewakili karakteristik T_{ON} yang biasa digunakan untuk pengendalian motor DC *servo*. Gambar 4.2 merupakan salah satu contoh hasil pembacaan sinyal PWM dengan T_{on} 900 µs dan periode 20 ms. Data hasil pengujian ditunjukkan pada Tabel 4.2.



Gambar 4.2. Hasil pembacaan sinyal PWM menggunakan osiloskop

Tabel 4.2. Data Hasil Pengujian Sinyal PWM

Data Terima (°)	T_{ON} Perancangan (µs)	T_{ON} Pengukuran (µs)					
		SD1	SD2	SD3	SD4	SD5	SD6
0	900	900	900	900	900	900	900
5	950	952	952	952	952	952	952
10	1000	1000	1000	1000	1000	1000	1000
15	1050	1050	1050	1055	1055	1055	1050
20	1100	1100	1100	1100	1100	1100	1100
25	1150	1150	1150	1150	1150	1150	1150
30	1200	1200	1200	1200	1200	1200	1200
35	1250	1250	1255	1250	1255	1255	1255
40	1300	1300	1300	1300	1300	1300	1300
45	1350	1350	1345	1345	1345	1345	1345
50	1400	1400	1400	1400	1400	1400	1400
55	1450	1450	1450	1450	1450	1450	1450
60	1500	1500	1500	1500	1500	1500	1500
65	1550	1545	1550	1550	1550	1550	1550
70	1600	1600	1600	1600	1600	1600	1600
75	1650	1655	1650	1650	1655	1655	1655
80	1700	1700	1700	1700	1700	1700	1700
85	1750	1750	1750	1750	1750	1750	1750
90	1800	1795	1800	1800	1800	1800	1800
95	1850	1850	1850	1850	1850	1850	1850
100	1900	1900	1900	1900	1900	1900	1900
105	1950	1950	1950	1950	1950	1950	1950
110	2000	2000	2000	2000	2000	2000	2000
115	2050	2050	2050	2050	2050	2050	2050
120	2100	2100	2100	2100	2100	2100	2100

*SD = Rangkaian *Serial Driver*

Berdasarkan data hasil pengujian pada Tabel 4.2, masing-masing rangkaian *Serial Driver* dapat membangkitkan sinyal PWM dengan rentang T_{on} 900 – 2100 µs. Dari 25

variasi data yang diuji terdapat beberapa perbedaan antara hasil perancangan dan pengukuran. Perbedaan tersebut terdapat pada lebar Ton 950 µs, 1050 µs, 1250 µs, 1350 µs, 1550 µs, 1650 µs, dan 1800 µs dengan selisih antara 0-5 µs. Namun selisih tersebut tidak berpengaruh pada sudut yang akan dihasilkan motor DC *servo* karena berdasarkan persamaan (3-9) pada perancangan PWM dan persamaan (3-11) pada perancangan konversi sudut, perubahan sudut 1^o terjadi setiap selisih 10 µs.

4.3. Pengujian Motor DC *Servo*

Pengujian dilakukan dengan mengirimkan data nilai sudut sebanyak 25 variasi dimulai dari 0^o – 120^o. Data akan dikonversi oleh rangkaian *Serial Driver* menjadi nilai pembangkit sinyal PWM untuk menggerakkan motor DC *servo*. Pengujian dilakukan sebanyak 2 kali untuk tiap sudut yang diuji. Data hasil pengujian ditunjukkan pada Tabel 4.3 dan grafik pada Gambar 4.3. Data yang ditunjukkan pada tabel 4.3 merupakan data hasil pengujian pertama, data hasil pengujian kedua tidak ditampilkan karena memiliki hasil yang hampir sama dengan hasil pengujian pertama. Hasil pengujian yang dilakukan sebanyak 25 variasi kemudian dirata-rata untuk mengetahui karakteristik masing-masing motor DC servo yang digunakan.

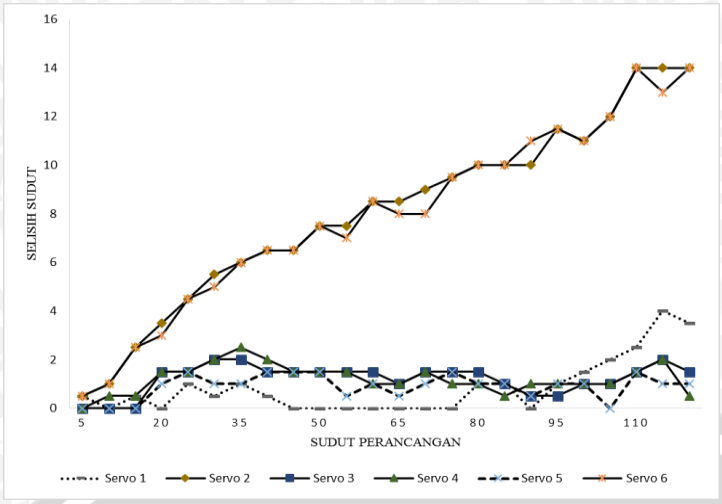
Berdasarkan hasil pengujian pada Tabel 4.3, motor DC *servo* dapat bergerak dengan rentang sudut 0^o-120^o. Sinyal PWM yang dapat digunakan yaitu pada rentang lebar T_{ON} 900 – 2100 µs. Sudut 0^o diwakili oleh lebar T_{ON} 900 µs dan sudut 120^o diwakili oleh lebar T_{ON} 2100 µs. Dari hasil pengujian didapatkan selisih sudut antara perancangan dan pengukuran yang berbeda-beda untuk setiap motor DC *servo* seperti ditunjukkan pada Gambar 4.3. Motor DC *servo* 1 memiliki selisih sudut antara $0{,}5^o$ - 4^o, motor DC *servo* 2 antara $0{,}5^o$ - 14^o, motor DC *servo* 3 antara $0{,}5^o$ - 2^o, motor DC *servo* 4 antara $0{,}5^o$ - $2{,}5^o$, motor DC *servo* 5 antara $0{,}5^o$ - $1{,}5^o$ dan motor DC *servo* 6 antara $0{,}5^o$ - 14^o. Dari besarnya selisih sudut yang didapat, motor DC *servo* yang digunakan dalam penelitian ini dapat dikategorikan menjadi dua jenis yaitu motor DC *servo* yang ideal dan tidak ideal. Motor DC *servo* ideal memiliki selisih sudut kurang dari 4^o sedangkan motor DC *servo* tidak ideal memiliki selisih sudut lebih dari 4^o. Yang termasuk dalam motor DC *servo* ideal yang digunakan dalam penelitian ini yaitu motor DC *servo* 1, 3, 4, dan 5, sedangkan yang tidak ideal yaitu motor DC *servo* 2 dan 6. Motor DC *servo* yang ideal seharusnya dapat bergerak dari 0^o – 120^o pada rentang lebar T_{ON} 900 – 2100 µs, namun untuk motor DC *servo* yang tidak ideal bergerak dari 0^o – 134^o pada rentang lebar T_{ON} yang sama.

Tabel 4.3. Hasil Pengujian Pertama Sudut Pergerakan Motor

ke	Sudut Perancangan	Sudut Pengukuran						Selisih					
		MS1	MS2	MS3	MS4	MS5	MS6	MS1	MS2	MS3	MS4	MS5	MS6
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	5	4,5	5,5	5,0	5,0	5,0	5,5	0,5	0,5	0	0	0	0,5
3	10	10	11	10	10,5	10	11	0	1	0	0,5	0	1
4	15	14,5	17,5	15	15,5	15	17,5	0,5	2,5	0	0,5	0	2,5
5	20	20	23,5	21,5	21,5	21,0	23	0	3,5	1,5	1,5	1,0	3
6	25	26	29,5	26,5	26,5	26,5	29,5	1	4,5	1,5	1,5	1,5	4,5
7	30	30,5	35,5	32	32	31	35	0,5	5,5	2	2	1	5
8	35	36	41	37	37,5	36,0	41	1	6	2	2,5	1	6
9	40	40,5	46,5	41,5	42	41,5	46,5	0,5	6,5	1,5	2	1,5	6,5
10	45	45	51,5	46,5	46,5	46,5	51,5	0	6,5	1,5	1,5	1,5	6,5
11	50	50	57,5	51,5	51,5	51,5	57,5	0	7,5	1,5	1,5	1,5	7,5
12	55	55	62,5	56,5	56,5	55,5	62	0	7,5	1,5	1,5	0,5	7
13	60	60	68,5	61,5	61	61,0	68,5	0	8,5	1,5	1	1,0	8,5
14	65	65	73,5	66,0	66,0	65,5	73,0	0	8,5	1,0	1,0	0,5	8,0
15	70	70	79	71,5	71,5	71,0	78	0	9	1,5	1,5	1,0	8
16	75	75	84,5	76,5	76	76,5	84,5	0	9,5	1,5	1	1,5	9,5
17	80	81	90	81,5	81	81,0	90	1,0	10	1,5	1	1,0	10
18	85	86	95	86,0	85,5	86	95	1,0	10	1	0,5	1	10
19	90	90	100	90,5	91	90,5	101	0	10	0,5	1,0	0,5	11
20	95	96	106,5	95,5	96	96	106,5	1,0	11,5	0,5	1,0	1,0	11,5
21	100	101,5	111	101	101	101	111	1,5	11	1	1	1	11
22	105	107	117	106	106	105	117	2	12	1	1	0	12
23	110	112,5	124	111,5	111,5	111,5	124	2,5	14	1,5	1,5	1,5	14
24	115	119	129	117	117	116	128	4	14	2	2	1	13
25	120	123,5	134	121,5	119,5	121	134	3,5	14	1,5	0,5	1,0	14
Rata-rata								0,8	7,7	1,2	1,2	0,9	7,6

*MS = Motor DC Servo

Hasil pengujian kedua tidak ditampilkan karena memiliki hasil yang hampir sama dengan pengujian pertama



Gambar 4.3. Grafik selisih sudut perancangan dan pengukuran

4.4. Pengujian Keseluruhan Sistem

Pengujian ini bertujuan untuk membuktikan bahwa tiap blok yang telah diuji dapat dirangkai menjadi satu sistem yang utuh dan dapat bekerja sesuai dengan perencanaan. Seperti yang telah dijelaskan pada sub bab 3.3.5, pengujian dibagi menjadi dua yaitu pengujian sudut dan pengujian gerakan. Pengujian sudut dilakukan sebanyak 2 kali dengan memberikan sudut 0º-120º pada setiap motor DC *servo* dengan 2 variasi waktu. Pengujian gerakan dilakukan dengan memberikan sudut-sudut yang akan membentuk gerakan tertentu.

Pada pengujian sudut, 2 variasi waktu yang digunakan yaitu satu kali tahap dengan jeda 50 ms untuk mencapai sudut target dan 20 kali tahap dengan jeda 50 ms untuk mencapai sudut target. Berdasarkan hasil pengujian sudut, didapatkan data sudut seperti ditunjukkan pada Tabel 4.4. Data yang ditunjukkan pada tabel 4.4 merupakan data hasil pengujian pertama, data hasil pengujian kedua tidak ditampilkan karena memiliki hasil yang hampir sama dengan hasil pengujian pertama. Hasil pengujian juga ditunjukkan pada Gambar 4.3 – 4.4 untuk melihat pergerakan motor DC *servo*. Berdasarkan data hasil pengujian pada Tabel 4.4, selisih antara sudut perancangan dan pengukuran pada motor DC *servo* 1 dengan 1 tahap memiliki rata-rata sebesar 1,57º, pada motor DC *servo* 2 sebesar 1,43º, pada motor DC *servo* 3 sebesar 2,71º, pada motor DC *servo* 4 sebesar 2,14º, pada motor DC *servo* 5 sebesar 0,57º, dan pada motor DC *servo* 6 sebesar 2,29º. Sedangkan selisih antara sudut perancangan dan pengukuran pada motor DC *servo* 1 dengan 20 tahap memiliki rata-rata sebesar 2,71º, pada motor DC *servo* 2 sebesar 1,71º, pada motor DC *servo* 3 sebesar 2,93º, pada motor DC *servo* 4 sebesar 2,43º, pada motor DC *servo* 5 sebesar 1,07º, dan pada motor DC *servo* 6 sebesar 2,5º.

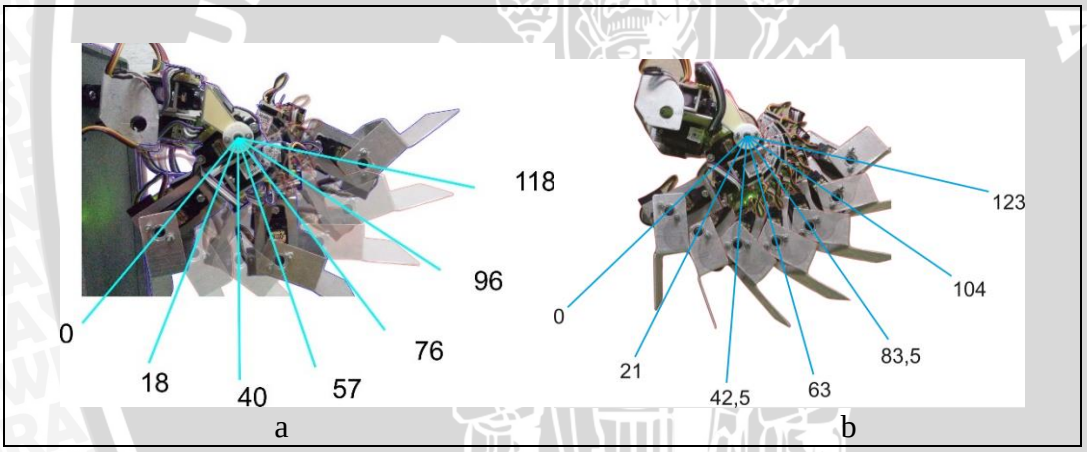
Tabel 4.4. Data hasil pengujian pertama sudut motor DC *servo* pada lengan

	Perc. ke	Sudut Perancangan	Sudut Pengukuran						Selisih					
			MS1	MS2	MS3	MS4	MS5	MS6	MS1	MS2	MS3	MS4	MS5	MS6
1 step	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	2	20	22	19	22	18	21	22	2	1	2	2	1	2
	3	40	43	41	42	40	41	43	3	1	2	0	1	3
	4	60	61	61	62	57	61	62	1	1	2	3	1	2
	5	80	80	81	72	76	81	84	0	1	8	4	1	4
	6	100	102	103	96	96	100	103	2	3	4	4	0	3
	7	120	123	123	121	118	120	122	3	3	1	2	0	2
	Rata - rata								1,57	1,43	2,71	2,14	0,57	2,29
	Perc. ke	Sudut Perancangan	Sudut Pengukuran						Selisih					
			MS1	MS2	MS3	MS4	MS5	MS6	MS1	MS2	MS3	MS4	MS5	MS6
20 step	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	2	20	22	21	22,5	21,0	21	22,5	2	1	2,5	1	1	2,5
	3	40	44	42	44	42,5	40,5	42,5	4	2	4	2,5	0,5	2,5
	4	60	63	61	64	63	61	63,5	3	1	4	3	1	3,5
	5	80	82	82	83	83,5	81,5	83	2	2	3	3,5	1,5	3
	6	100	104	102	103	104	102	103	4	2	3	4	1,5	3
	7	120	124	124	124	123	122	123	4	4	4	3	2	3
	Rata - rata								2,71	1,71	2,93	2,43	1,07	2,50

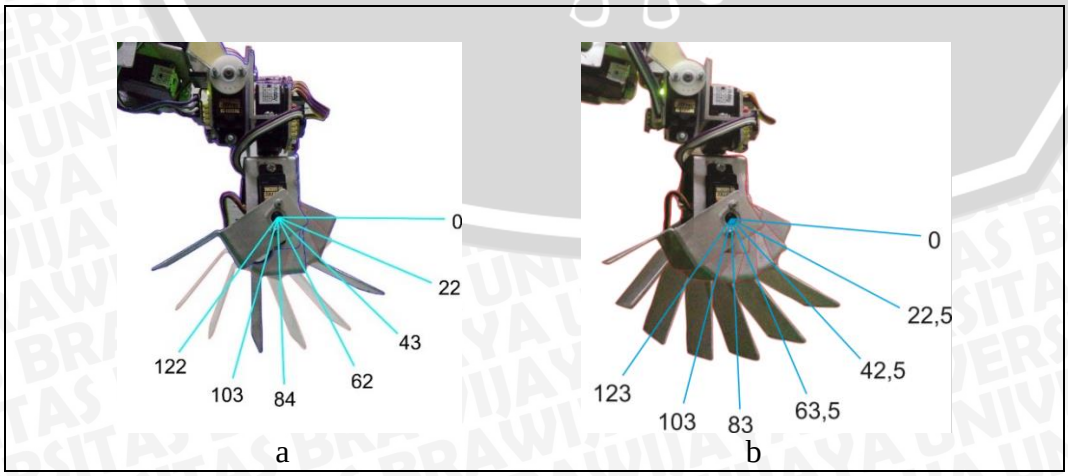
*MS = Motor DC Servo

Hasil pengujian kedua tidak ditampilkan karena memiliki hasil yang hampir sama dengan pengujian pertama

Motor DC *servo* yang digerakkan dengan 20 tahap memiliki pergerakan lebih halus namun menghasilkan selisih rata-rata lebih besar jika dibandingkan dengan pergerakan motor DC *servo* dengan 1 tahap. Selisih tersebut terjadi karena adanya penambahan dan pembulatan hasil pembagian tahap yang digunakan. Namun selisih yang dihasilkan dapat diabaikan karena dalam Kontes Robot Seni Indonesia yang menjadi poin penting adalah keluwesan dan kehalusan gerakan robot. Sehingga dapat diambil kesimpulan jumlah tahap yang digunakan untuk menggerakkan motor DC *servo* mempengaruhi kecepatan dan pergerakan motor DC *servo* untuk mencapai sudut target. Semakin banyak jumlah tahap yang digunakan maka waktu yang dibutuhkan untuk mencapai sudut target semakin lama (gerakan motor DC *servo* lambat) dan pergerakannya semakin halus dan sebaliknya apabila jumlah tahap yang digunakan semakin sedikit maka waktu yang dibutuhkan untuk mencapai sudut target semakin cepat (gerakan motor DC *servo* cepat) dan pergerakannya tidak halus.



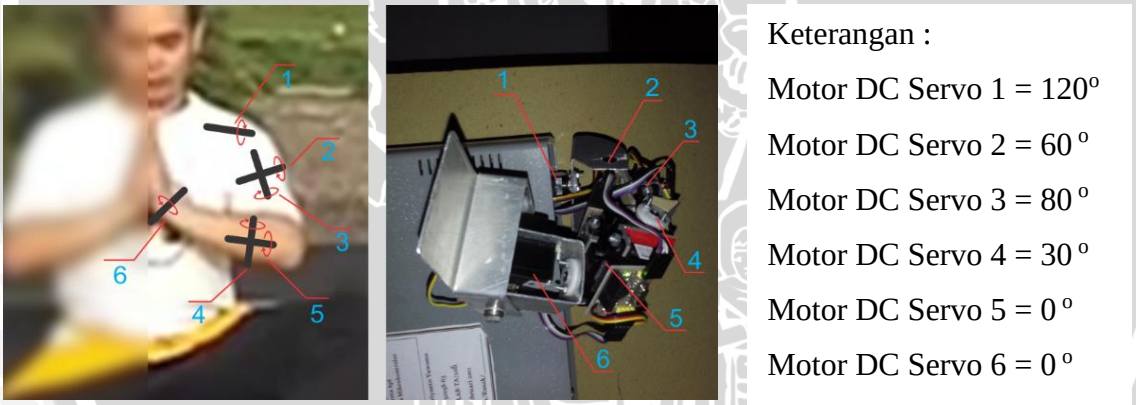
Gambar 4.3 Pergerakan Motor DC *servo* 4 (siku) (a) 1 tahap, (b) 20 tahap



Gambar 4.4 Pergerakan Motor DC *servo* 6 (pergelangan tangan) (a) 1 tahap, (b) 20 tahap

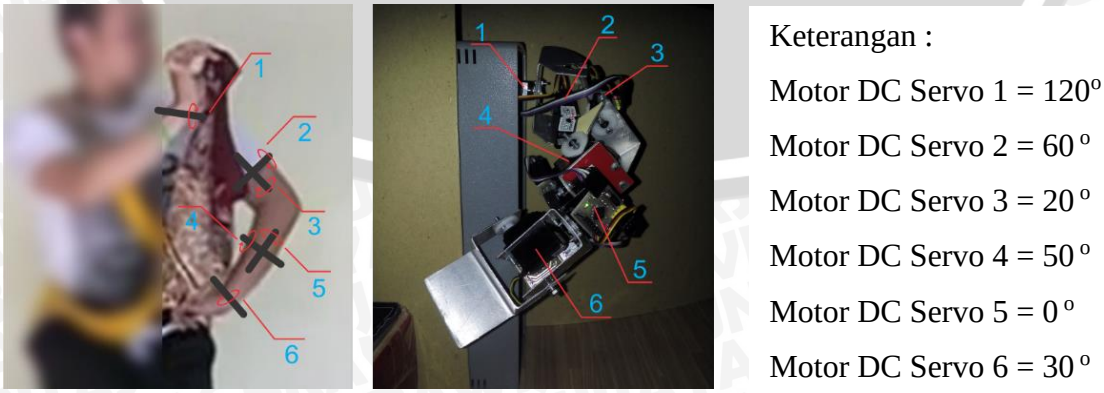
Hasil pengujian gerakan lengan robot ditunjukkan pada Gambar 4.5 – 4.8. Pada pengujian ini parameter yang dinilai yaitu apakah lengan robot dapat membentuk gerakan tarian. Berdasarkan hasil pengujian tersebut, motor DC *servo* yang telah disusun membentuk lengan robot dapat bergerak membentuk gerakan seperti gerakan referensi. Namun gerakan yang dibentuk tidak bisa sama persis seperti referensi dikarenakan keterbatasan sendi pada lengan robot.

Berdasarkan hasil pengujian keseluruhan dapat diambil kesimpulan masing-masing rangkaian *Serial Driver* dapat menerima dan menggerakkan motor DC *servo* sesuai data yang dikirim dari blok master. Semakin banyak jumlah tahap yang digunakan untuk mencapai sudut target maka gerakan motor DC *servo* semakin halus dan lambat. Gerakan-gerakan yang dibentuk dapat mengikuti gerakan referensi. Sehingga rangkaian *Serial Driver* ini dapat diterapkan pada robot *humanoid* KRSI. Instalasi kabel juga menjadi lebih ringkas dan untuk mengontrol 6 buah motor DC *servo* dapat dilakukan menggunakan 1 pin mikrokontroler.



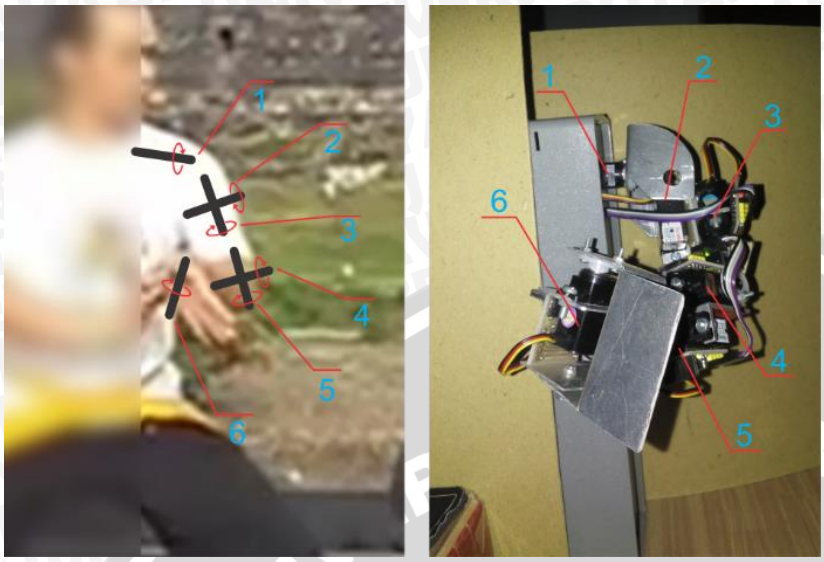
a b

Gambar 4.5. Gerakan *Sembahan* (a) gerakan manusia dan (b) gerakan robot



a b

Gambar 4.6. Gerakan *Encot* (a) gerakan manusia dan (b) gerakan robot



Keterangan :

Motor DC Servo 1 = 90º

Motor DC Servo 2 = 60 º

Motor DC Servo 3 =100 º

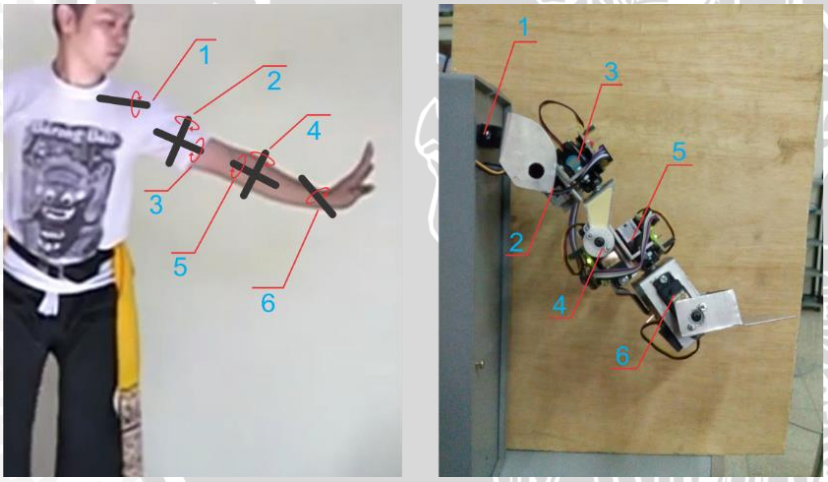
Motor DC Servo 4 = 25 º

Motor DC Servo 5 = 0 º

Motor DC Servo 6 = 0 º

a b

Gambar 4.7. Gerakan *Panggel* (a) gerakan manusia dan (b) gerakan robot



Keterangan :

Motor DC Servo 1 = 60º

Motor DC Servo 2 = 20 º

Motor DC Servo 3 =60 º

Motor DC Servo 4 = 120 º

Motor DC Servo 5 = 60 º

Motor DC Servo 6 = 0 º

a b

Gambar 4.8. Gerakan *Kengser* (a) gerakan manusia dan (b) gerakan robot

BAB V
PENUTUP

5.1. Kesimpulan

Berdasarkan rumusan masalah, perancangan dan hasil pengujian maka dapat diambil kesimpulan sebagai berikut

1. Motor DC *servo* paralel dapat dikontrol secara serial menggunakan metode komunikasi serial *multipoint*, komunikasi *simplex* dan pengiriman data menggunakan metode paket data yang terdiri dari 1 *byte Header*, ID dan Sudut.
2. Rangkaian *Serial Driver* dapat dirancang menggunakan *peripheral* UART mikrokontroler ATmega8 dengan *Baudrate* 1 Mbps dan PWM dengan resolusi 16 bit.
3. Rangkaian *Serial Driver* mampu menerima data secara akurat dengan waktu tunggu 150 µs tiap *Serial Driver* dari 6 *Serial Driver*, mampu membangkitkan sinyal PWM dengan selisih antara perancangan dan pengukuran 0-5 µs, mampu menggerakkan motor DC *servo* dengan selisih terbesar 4^o untuk motor DC servo ideal dan selisih diatas 4^o untuk motor DC servo tidak ideal. Semakin banyak jumlah tahap yang digunakan untuk mencapai sudut target maka gerakan motor DC *servo* semakin halus.

5.2. Saran

Adapun saran untuk pengembangan mengenai penelitian serupa yaitu sebagai berikut:

1. Perlu dilakukan penyempurnaan pada algoritma tahap agar tidak terjadi pembulatan disetiap hasil bagi jumlah tahap yang digunakan sehingga pergerakan motor DC *servo* menjadi lebih akurat.

DAFTAR PUSTAKA

Atmel. 2009. *8-bit AVR with 8K Bytes In-System Progammable Flash ATmega8*. San Jose: ATMEL

Dikti. 2014. *Panduan KRSI 2015-ver Okt 2014*. Jakarta: Direktorat Jenderal Pendidikan Tinggi Kementrian RisetTeknologi dan Pendidikan Tinggi

Godbole Achyut S. 2002. *Data Communications and Networks*. New Delhi: Tata McGraw-Hill

Lee Jun Hee. 2007. *General Specification of HS-5085MG Digital Micro Servo*. Korea: Hitec Rcd Korea Inc.

Moon Yongseon, Ko Nak Yong, Bae Youngchul. 2009. *The Design of Humanoid Robot Arm Based on Morphological and Neurological Analysis of Human Arm*. Shanghai: InTech

Pinckney, Nathaniel. 2006. *Pulse-Width Modulation For Microcontroller Servo Control*. IEEE Potentials. 25(1):27–29.

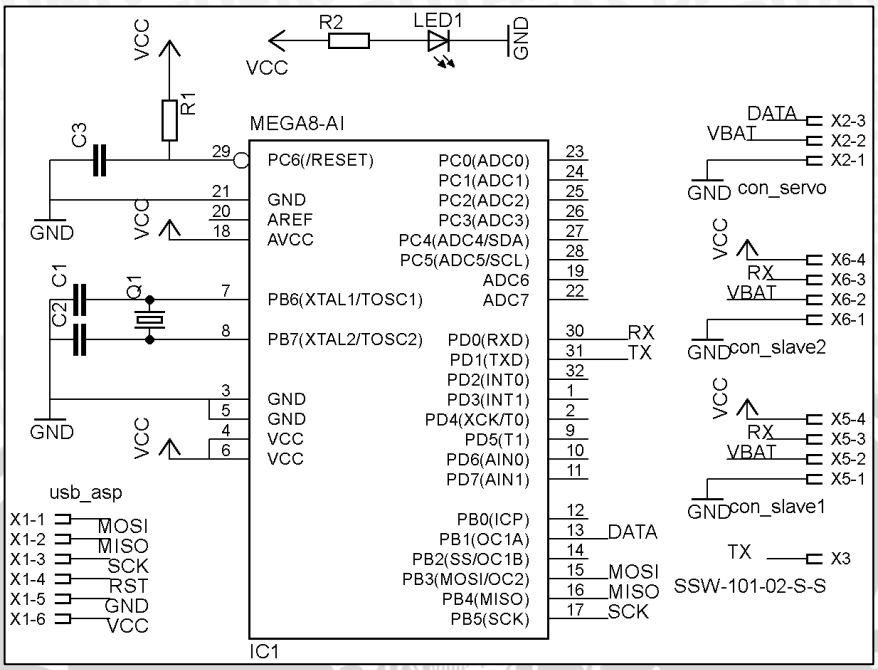
Sinnema William, McGovern Tom. 1986. *Digital, Analog, and Data Communication Second Edition*. United State of America: Prentice Hall Inc.

Syahrul. 2014. "*Pemrograman Mikrokontroler AVR Bahasa Assembly dan C"*. Bandung: Informatika

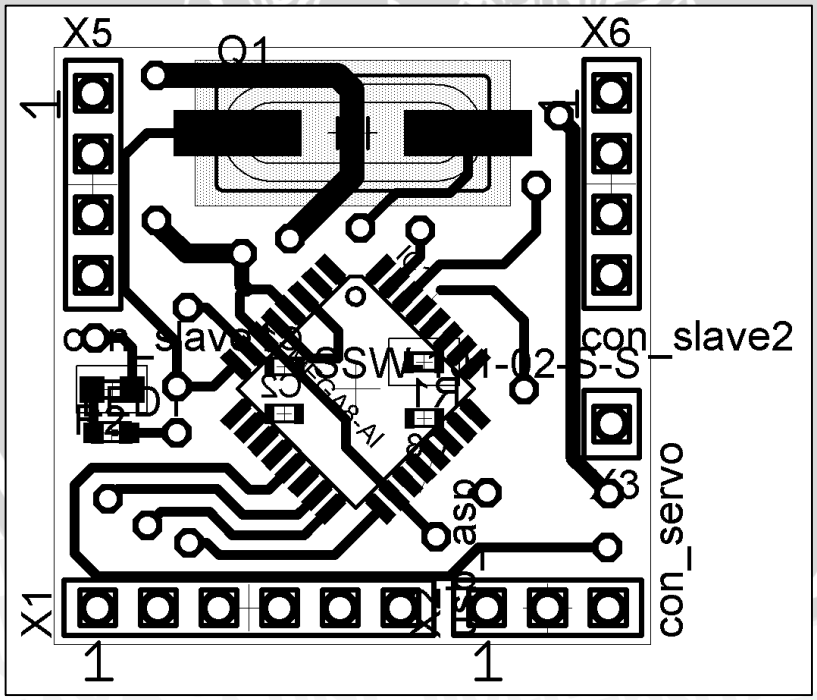
Tsai James, Lai C. 2006. *Design and Control of a Humanoid Robot*. IEEE/RSJ International Conference on Intelligent Robots and Systems. 9-15 Oktober 2006.

Winoto Ardi. 2010. *Mikrokontroler AVR ATmega8/16/32/8535 dan Pemrogramannya dengan Bahasa C pada Win AVR*. Bandung : Informatika

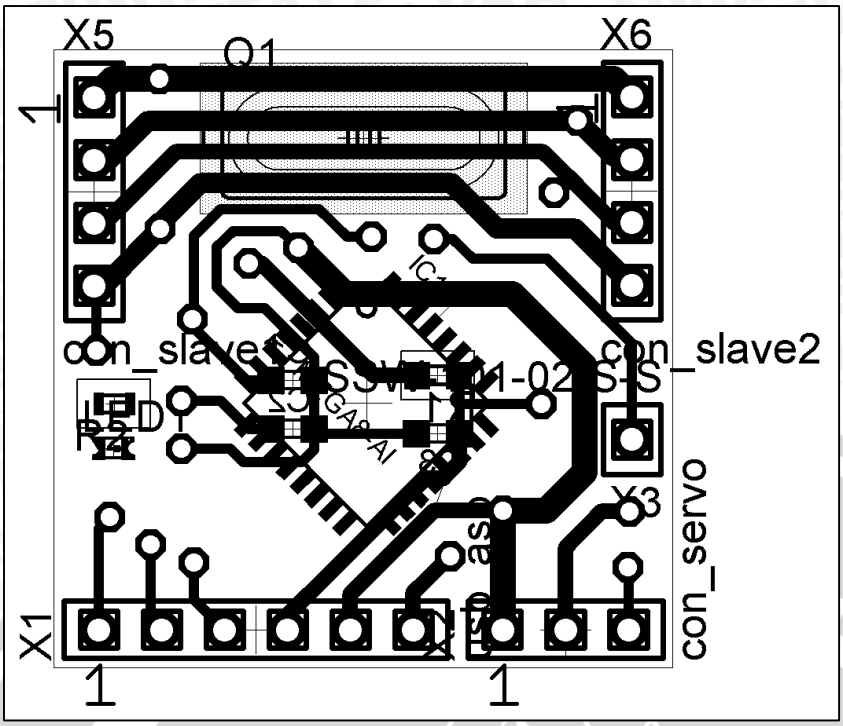
Lampiran 1. Hasil Rancangan Alat



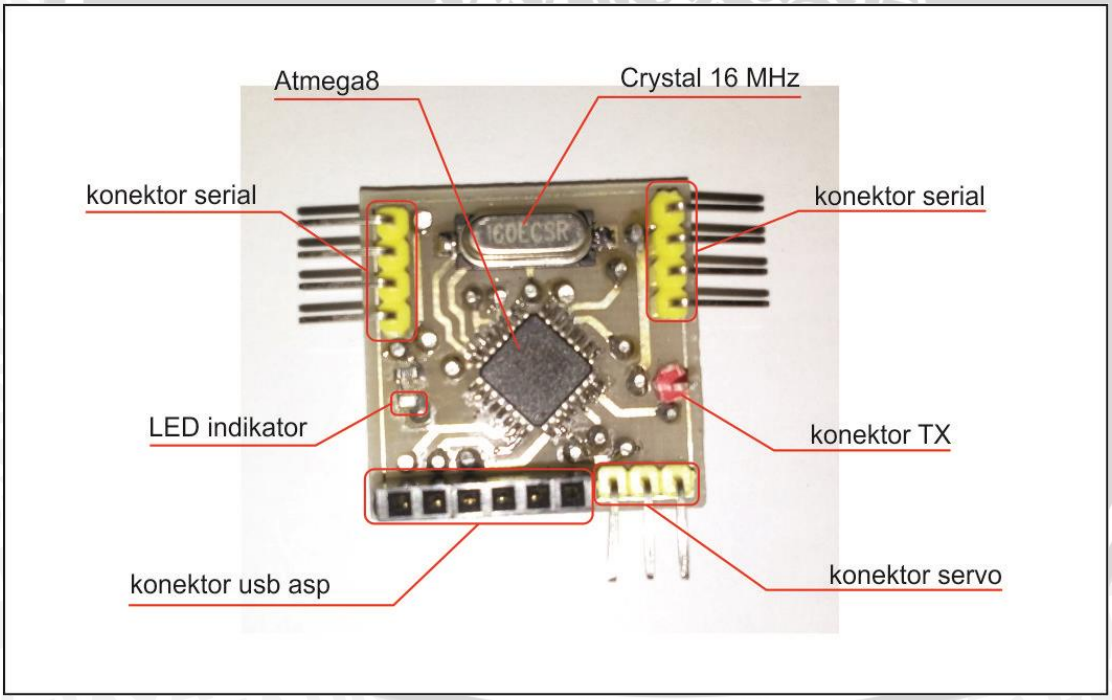
Gambar 1. Skematik Rangkaian *Serial Driver*



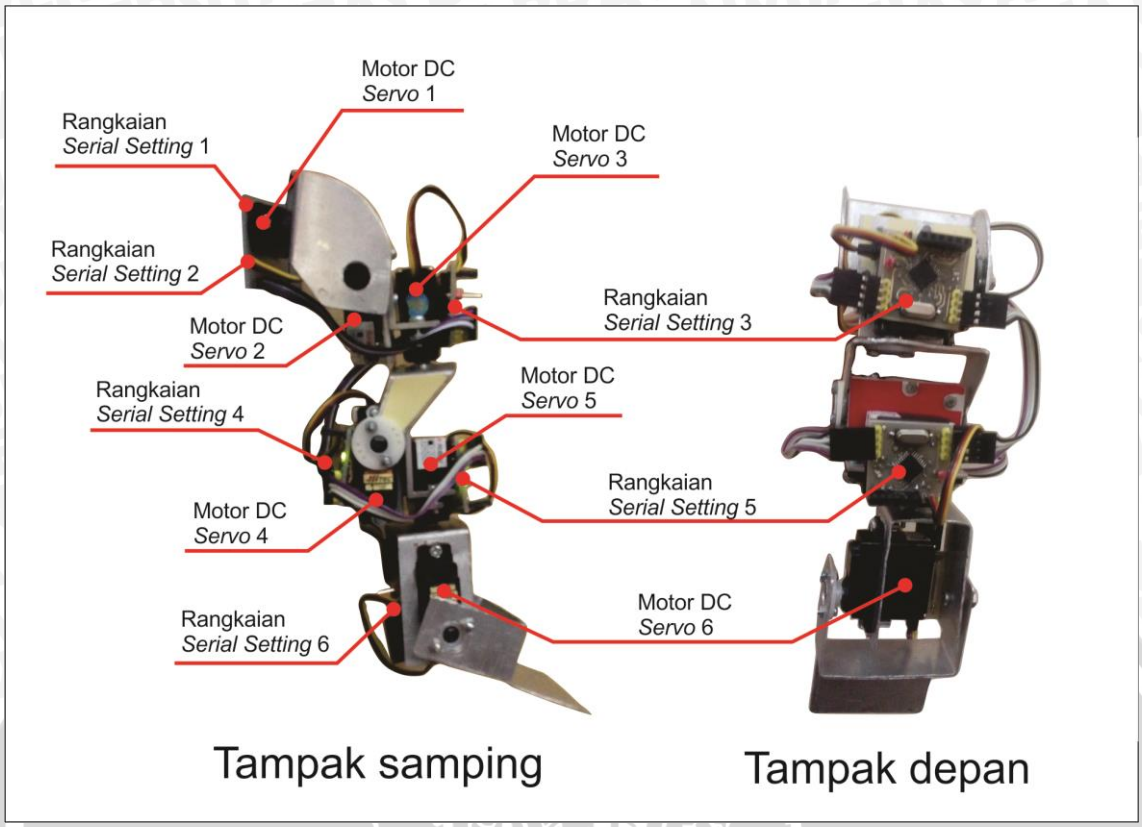
Gambar 2. Layout PCB Top Layer Rangkaian *Serial Driver*



Gambar 3. Layout PCB Bottom Layer Rangkaian *Serial Driver*

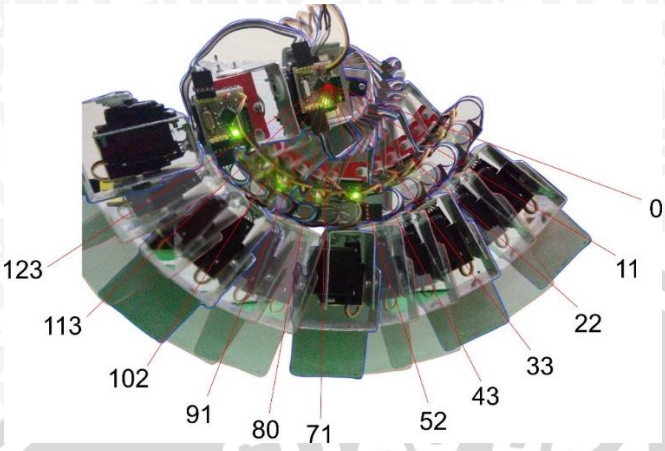


Gambar 4. PCB Rangkaian *Serial Driver*

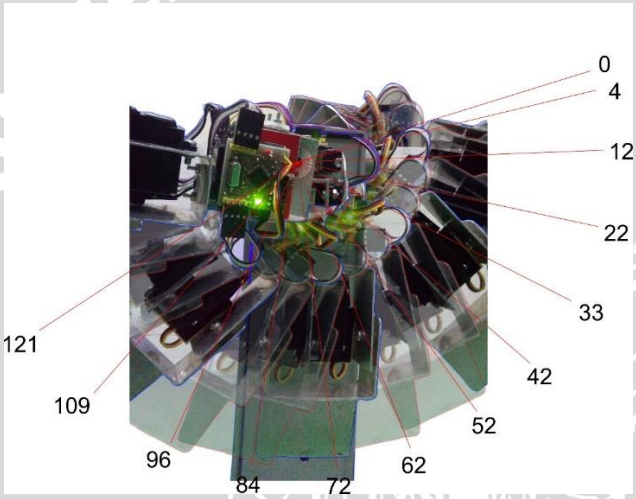


Gambar 5. Rangkaian *Serial Driver* terpasang pada lengan robot

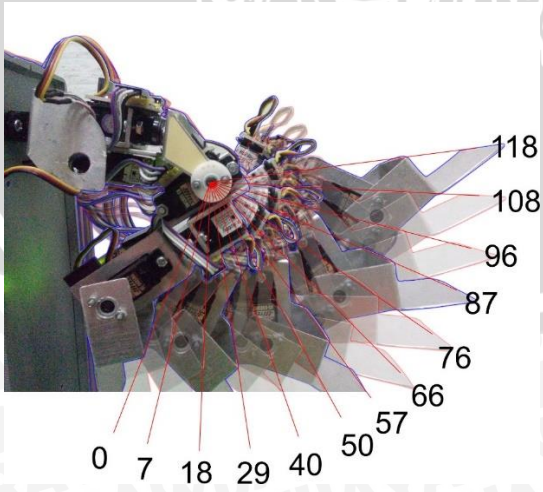
Lampiran 2. Pengujian motor DC *servo* pada lengan robot



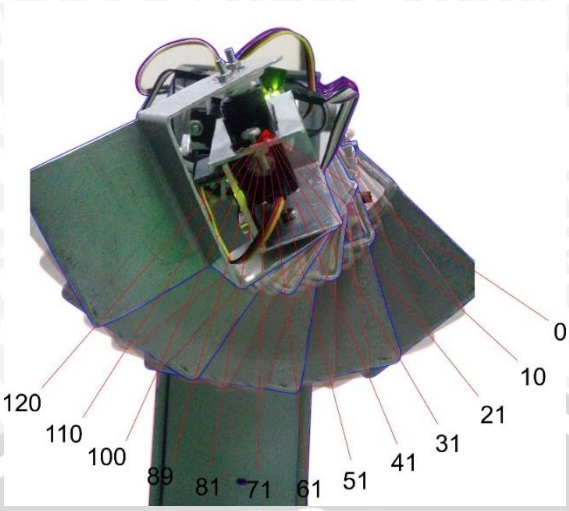
Gambar 6. Pengujian motor DC *servo* 1 pada lengan robot dengan 1 step



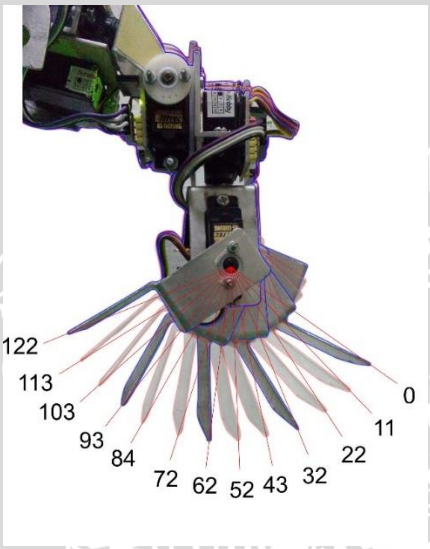
Gambar 7. Pengujian motor DC *servo* 2 pada lengan robot dengan 1 step



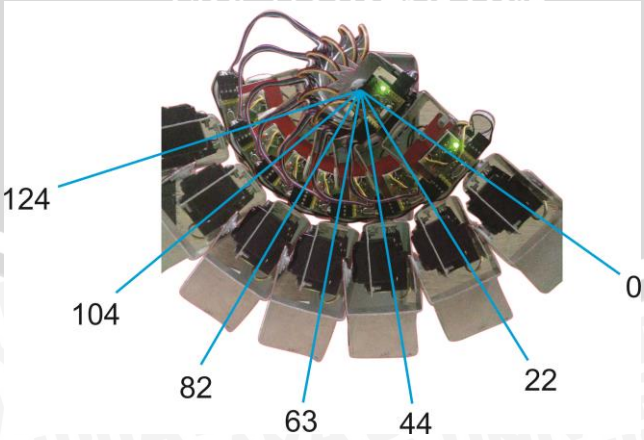
Gambar 8. Pengujian motor DC *servo* 3 pada lengan robot dengan 1 step



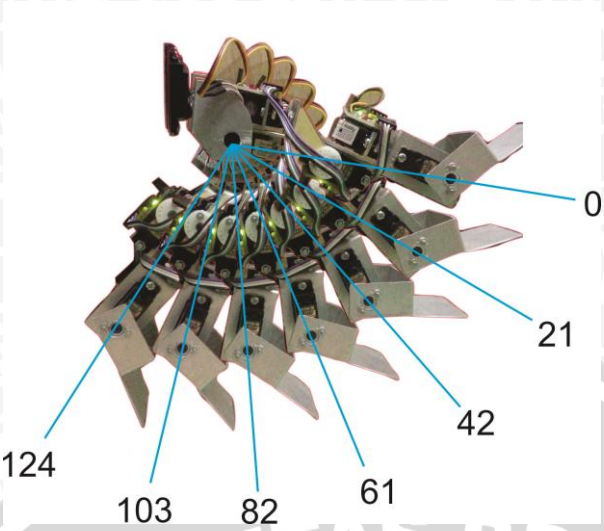
Gambar 9. Pengujian motor DC *servo* 4 pada lengan robot dengan 1 step



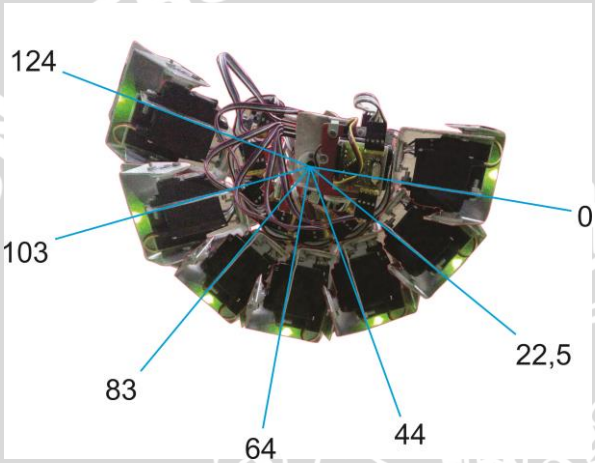
Gambar 10. Pengujian motor DC *servo* 5 pada lengan robot dengan 1 step



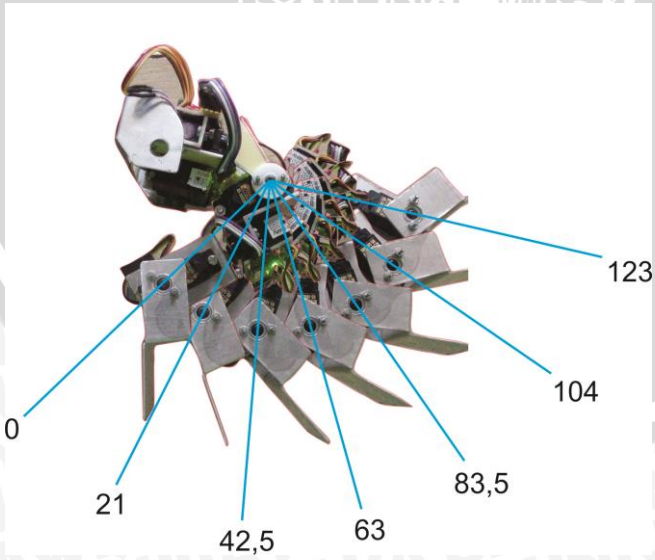
Gambar 11. Pengujian motor DC *servo* 1 pada lengan robot dengan 20 step



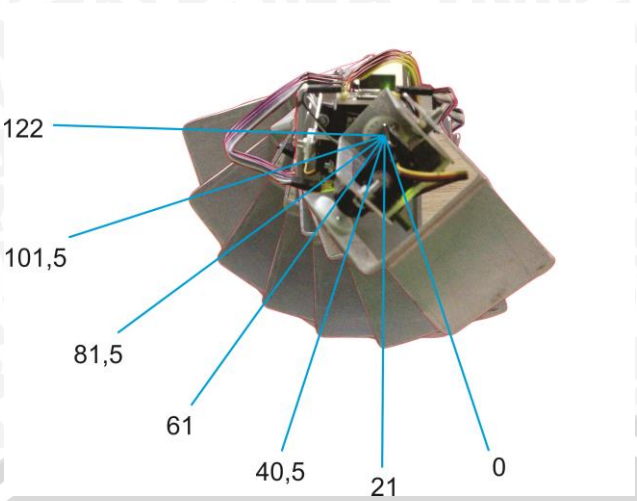
Gambar 12. Pengujian motor DC *servo* 2 pada lengan robot dengan 20 step



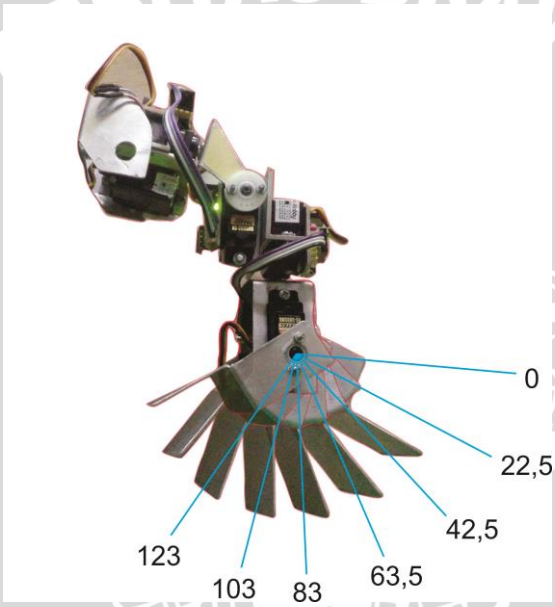
Gambar 13. Pengujian motor DC *servo* 3 pada lengan robot dengan 20 step



Gambar 14. Pengujian motor DC *servo* 4 pada lengan robot dengan 20 step



Gambar 15. Pengujian motor DC *servo* 5 pada lengan robot dengan 20 step



Gambar 16. Pengujian motor DC *servo* 6 pada lengan robot dengan 20 step

Lampiran 3. Listing Program

```
#ifndef F_CPU
#define F_CPU 16000000UL
#endif

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <util/delay.h>

unsigned int hitung;
unsigned int id;
uint8_t c;
volatile unsigned int
data;
volatile uint8_t a,b;
volatile char tampung[30];


void USARTInit(uint16_t
UBRRVAL)
{
  // set baud rate
  UBRRL = UBRRVAL;
  UBRRH = (UBRRVAL>>8);

  UCSRC =
(1<<URSEL)|(1<<UCSZ1)|(1<<
UCSZ0);

  // enable receiver and
transmitter and interrupt
rx
  UCSRB =
(1<<RXCIE)|(1<<TXEN)|(1<<R
XEN);

}
void
USARTWriterChar(unsigned
char data)
{
  while(!(UCSRA &
(1<<UDRE)))
  {

  }
  UDR=data;
}

void
USARTWriteString(char*
str)
{
  int len = strlen(str);
  while(len--)
  {

  USARTWriterChar(*str++);
       while(!(UCSRA &
(1<<UDRE)));
  }
}

unsigned char
USARTReaderChar()
{
  while(!(UCSRA &
(1<<RXC)))
  {

  }
  return UDR;
}

void USARTReadString(char
*buffer){
  char tmp;
  bacaLagi:
  tmp = USARTReaderChar();
  *buffer++ = tmp;
  if(tmp != '\0'){
       goto bacaLagi;
  }
}

// this function is called
by printf as a stream
handler
int
usart_putchar_printf(char
var, FILE *stream) {
  // translate \n to \r
for br@y++ terminal
  if (var == '\n')
USARTWriterChar('\r');
  USARTWriterChar(var);
  return 0;
}

static FILE mystdout =
FDEV_SETUP_STREAM(usart_pu
tchar_printf, NULL,
_FDEV_SETUP_WRITE);
```

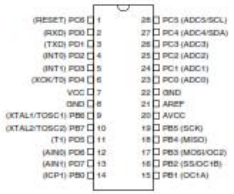
```
void init_IO()
{
  DDRC=0xff;
}
void init_PWM()
{
  DDRB |= 1<<PB1;
  TCCR1A |=
(0<<COM1A1)|(0<<COM1A0)|(0
<<COM1B1)|(0<<COM1B0)|(1<<
WGM11)|(0<<WGM10);
  TCCR1B |=
(1<<WGM13)|(1<<WGM12)|(0<<
CS12)|(1<<CS11)|(0<<CS10);
  TIMSK |=
(1<<OCIE1A)|(1<<TOIE1);
  ICR1 = 39999;
  //OCR1A = 1799;
}
ISR(TIMER1_OVF_vect)
{
  PORTC=0xFF;

}
ISR(TIMER1_COMPA_vect)
{

  PORTC=0x00;
}
void gerakServo(unsigned
int nilai);
void terimaData();

int main(void)
{
  stdout = &mystdout;

  init_IO();
  init_PWM();
  USARTInit(0); // 16MHz
baudrate 1M
  sei ();
   /* Replace with your
application code */
   while (1)
   {
      if (id == 1) // **
      {
gerakServo(data);
      }

   }
}

ISR (USART_RXC_vect)
{
char tmp;
  tmp = USARTReaderChar();
  USARTWriterChar(tmp);
  if (tmp == '@')
  {
      b++;

  } else
  if (b == 1)
  {
      b++;
      id = tmp;

  } else
  if (b == 2)
  {
      data = tmp;
      b=0;

  }

}

void gerakServo(unsigned
int nilai_servo)
{
  hitung =
((nilai_servo*20)+1799);
  OCR1A = hitung;
}
```

Catatan :

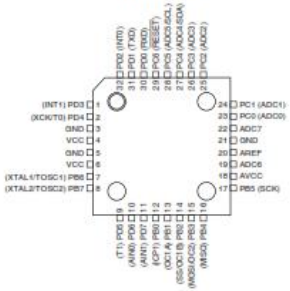
**id disesuaikan dengan rangkaian Serial Controller, masing-masing rangkaian Serial Controller memiliki id yang berbeda-beda. Pada penelitian ini terdapat 6 buah rangkaian Serial Controller sehingga id yang digunakan yaitu 1 – 6.

Pin Configurations

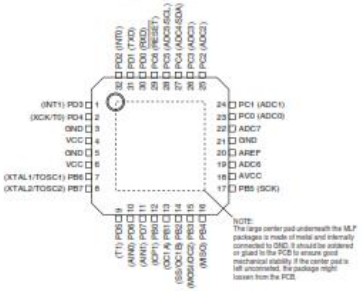
PDIP



TQFP Top View



MLF Top View



Features

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 130 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory segments
  - 8K Bytes of In-System Self-programmable Flash program memory
  - 512 Bytes EEPROM
  - 1K Byte Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler, one Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Three PWM Channels
  - 8-channel ADC in TQFP and QFN/MLF package
    Eight Channels 10-bit Accuracy
  - 6-channel ADC in PDIP package
    Six Channels 10-bit Accuracy
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Five Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, and Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-lead PDIP, 32-lead TQFP, and 32-pad QFN/MLF
- Operating Voltages
  - 2.7 - 5.5V (ATmega8L)
  - 4.5 - 5.5V (ATmega8)
- Speed Grades
  - 0 - 8 MHz (ATmega8L)
  - 0 - 16 MHz (ATmega8)
- Power Consumption at 4 Mhz, 3V, 25°C
  - Active: 3.6 mA
  - Idle Mode: 1.0 mA
  - Power-down Mode: 0.5 µA

8-bit AVR® with 8K Bytes In-System Programmable Flash

ATmega8*
ATmega8L*

* Not recommended for new designs.

Rev.2486V–AVR–05/09

Interrupts

This section describes the specifics of the interrupt handling performed by the ATmega8. For a general explanation of the AVR interrupt handling, refer to "Reset and Interrupt Handling" on page 14.

Interrupt Vectors in ATmega8

Table 18. Reset and Interrupt Vectors

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

Notes: 1. When the BOOTRST Fuse is programmed, the device will jump to the Boot Loader address at reset, see "Boot Loader Support – Read-While-Write Self-Programming" on page 209.
2. When the IVSEL bit in GICR is set, Interrupt Vectors will be moved to the start of the boot Flash section. The address of each Interrupt Vector will then be the address in this table added to the start address of the boot Flash section.

Table 19 shows reset and Interrupt Vectors placement for the various combinations of BOOTRST and IVSEL settings. If the program never enables an interrupt source, the Interrupt Vectors are not used, and regular program code can be placed at these locations. This is also the case if the Reset Vector is in the Application section while the Interrupt Vectors are in the boot section or vice versa.

16-bit Timer/Counter1

The 16-bit Timer/Counter unit allows accurate program execution timing (event management), wave generation, and signal timing measurement. The main features are:

- **True 16-bit Design (i.e., allows 16-bit PWM)**
- **Two Independent Output Compare Units**
- **Double Buffered Output Compare Registers**
- **One Input Capture Unit**
- **Input Capture Noise Canceler**
- **Clear Timer on Compare Match (Auto Reload)**
- **Glitch-free, Phase Correct Pulse Width Modulator (PWM)**
- **Variable PWM Period**
- **Frequency Generator**
- **External Event Counter**
- **Four Independent Interrupt Sources (TOV1, OCF1A, OCF1B, and ICF1)**

Overview

Most register and bit references in this section are written in general form. A lower case "n" replaces the Timer/Counter number, and a lower case "x" replaces the Output Compare unit channel. However, when using the register or bit defines in a program, the precise form must be used i.e., TCNT1 for accessing Timer/Counter1 counter value and so on.

A simplified block diagram of the 16-bit Timer/Counter is shown in Figure 32. For the actual placement of I/O pins, refer to "Pin Configurations" on page 2. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the "16-bit Timer/Counter Register Description" on page 96.

The Input Capture Register can capture the Timer/Counter value at a given external (edge triggered) event on either the Input Capture Pin (ICP1) or on the Analog Comparator pins (see "Analog Comparator" on page 193). The Input Capture unit includes a digital filtering unit (Noise Canceler) for reducing the chance of capturing noise spikes.

The TOP value, or maximum Timer/Counter value, can in some modes of operation be defined by either the OCR1A Register, the ICR1 Register, or by a set of fixed values. When using OCR1A as TOP value in a PWM mode, the OCR1A Register can not be used for generating a PWM output. However, the TOP value will in this case be double buffered allowing the TOP value to be changed in run time. If a fixed TOP value is required, the ICR1 Register can be used as an alternative, freeing the OCR1A to be used as PWM output.

Definitions

The following definitions are used extensively throughout the document:

Table 35. Definitions

BOTTOM	The counter reaches the *BOTTOM* when it becomes 0x0000.
MAX	The counter reaches its *MAX*imum when it becomes 0xFFFF (decimal 65535).
TOP	The counter reaches the *TOP* when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be one of the fixed values: 0x00FF, 0x01FF, or 0x03FF, or to the value stored in the OCR1A or ICR1 Register. The assignment is dependent of the mode of operation.

Compatibility

The 16-bit Timer/Counter has been updated and improved from previous versions of the 16-bit AVR Timer/Counter. This 16-bit Timer/Counter is fully compatible with the earlier version regarding:

- All 16-bit Timer/Counter related I/O Register address locations, including Timer Interrupt Registers.
- Bit locations inside all 16-bit Timer/Counter Registers, including Timer Interrupt Registers.
- Interrupt Vectors.

The following control bits have changed name, but have same functionality and register location:

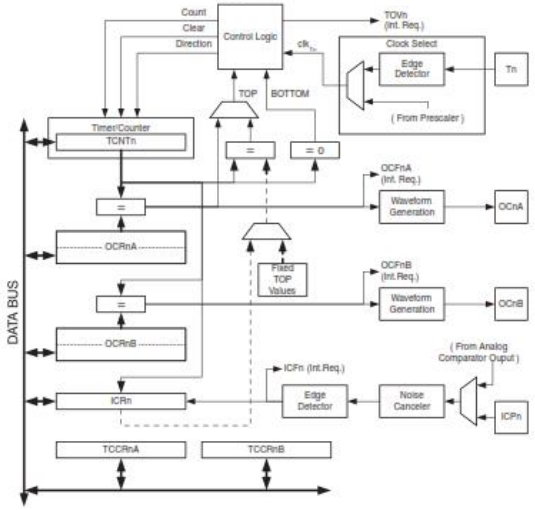
- PWM10 is changed to WGM10.
- PWM11 is changed to WGM11.
- CTC1 is changed to WGM12.

The following bits are added to the 16-bit Timer/Counter Control Registers:

- FOC1A and FOC1B are added to TCCR1A.
- WGM13 is added to TCCR1B.

The 16-bit Timer/Counter has improvements that will affect the compatibility in some special cases.

Figure 32. 16-bit Timer/Counter Block Diagram[1]



Note: 1. Refer to "Pin Configurations" on page 2, Table 22 on page 58, and Table 28 on page 63 for Timer/Counter1 pin placement and description.

Registers

The *Timer/Counter* (TCNT1), *Output Compare Registers* (OCR1A/B), and *Input Capture Register* (ICR1) are all 16-bit registers. Special procedures must be followed when accessing the 16-bit registers. These procedures are described in the section "Accessing 16-bit Registers" on page 79. The *Timer/Counter Control Registers* (TCCR1A/B) are 8-bit registers and have no CPU access restrictions. Interrupt requests (abbreviated to Int.Req. in the figure) signals are all visible in the *Timer Interrupt Flag Register* (TIFR). All interrupts are individually masked with the *Timer Interrupt Mask Register* (TIMSK). TIFR and TIMSK are not shown in the figure since these registers are shared by other timer units.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the T1 pin. The Clock Select logic block controls which clock source and edge the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the clock select logic is referred to as the timer clock (clk_{T1}).

The double buffered Output Compare Registers (OCR1A/B) are compared with the Timer/Counter value at all time. The result of the compare can be used by the waveform generator to generate a PWM or variable frequency output on the Output Compare Pin (OC1A/B). See "Output Compare Units" on page 84. The Compare Match event will also set the Compare Match Flag (OCF1A/B) which can be used to generate an Output Compare interrupt request.

Accessing 16-bit Registers

The TCNT1, OCR1A/B, and ICR1 are 16-bit registers that can be accessed by the AVR CPU via the 8-bit data bus. The 16-bit register must be byte accessed using two read or write operations. The 16-bit timer has a single 8-bit register for temporary storing of the High byte of the 16-bit access. The same temporary register is shared between all 16-bit registers within the 16-bit timer. Accessing the Low byte triggers the 16-bit read or write operation. When the Low byte of a 16-bit register is written by the CPU, the High byte stored in the temporary register, and the Low byte written are both copied into the 16-bit register in the same clock cycle. When the Low byte of a 16-bit register is read by the CPU, the High byte of the 16-bit register is copied into the temporary register in the same clock cycle as the Low byte is read.

Not all 16-bit accesses uses the temporary register for the High byte. Reading the OCR1A/B 16-bit registers does not involve using the temporary register.

To do a 16-bit write, the High byte must be written before the Low byte. For a 16-bit read, the Low byte must be read before the High byte.

The following code examples show how to access the 16-bit Timer Registers assuming that no interrupts updates the temporary register. The same principle can be used directly for accessing the OCR1A/B and ICR1 Registers. Note that when using "C", the compiler handles the 16-bit access.

Assembly Code Example[1]

```
...
; Set TCNT1 to 0x01FF
ldi r17,0x01
ldi r16,0xFF
out TCNT1H,r17
out TCNT1L,r16
; Read TCNT1 into r17:r16
in r16,TCNT1L
in r17,TCNT1H
...
```

C Code Example[1]

```
unsigned int i;
...
/* Set TCNT1 to 0x01FF */
TCNT1 = 0x1FF;
/* Read TCNT1 into i */
i = TCNT1;
...
```

Note: 1. See "About Code Examples" on page 8.

The assembly code example returns the TCNT1 value in the r17:r16 Register pair.

It is important to notice that accessing 16-bit registers are atomic operations. If an interrupt occurs between the two instructions accessing the 16-bit register, and the interrupt code updates the temporary register by accessing the same or any other of the 16-bit Timer Registers, then the result of the access outside the interrupt will be corrupted. Therefore, when both the main code and the interrupt code update the temporary register, the main code must disable the interrupts during the 16-bit access.

The following code examples show how to do an atomic read of the TCNT1 Register contents. Reading any of the OCR1A/B or ICR1 Registers can be done by using the same principle.

Assembly Code Example[1]

```
TIM16_ReadTCNT1:
  ; Save Global Interrupt Flag
  in r18,SREG
  ; Disable interrupts
  cli
  ; Read TCNT1 into r17:r16
  in r16,TCNT1L
  in r17,TCNT1H
  ; Restore Global Interrupt Flag
  out SREG,r18
  ret
```

C Code Example[1]

```
unsigned int TIM16_ReadTCNT1( void )
{
  unsigned char sreg;
  unsigned int i;
  /* Save Global Interrupt Flag */
  sreg = SREG;
  /* Disable interrupts */
  _CLI();
  /* Read TCNT1 into i */
  i = TCNT1;
  /* Restore Global Interrupt Flag */
  SREG = sreg;
  return i;
}
```

Note: 1. See "About Code Examples" on page 8.

The assembly code example returns the TCNT1 value in the r17:r16 Register pair.

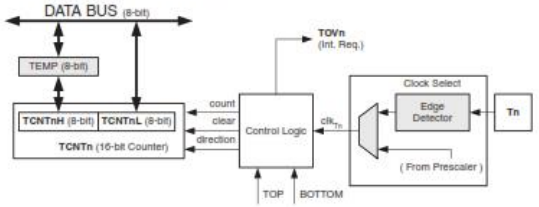
Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the clock select logic which is controlled by the *clock select* (CS12:0) bits located in the *Timer/Counter Control Register B* (TCCR1B). For details on clock sources and prescaler, see "Timer/Counter0 and Timer/Counter1 Prescalers" on page 74.

Counter Unit

The main part of the 16-bit Timer/Counter is the programmable 16-bit bi-directional counter unit. Figure 33 shows a block diagram of the counter and its surroundings.

Figure 33. Counter Unit Block Diagram



Signal description (internal signals):

count	Increment or decrement TCNT1 by 1.
direction	Select between increment and decrement.
clear	Clear TCNT1 (set all bits to zero).
clk_{T1}	Timer/Counter clock.
TOP	Signalize that TCNT1 has reached maximum value.
BOTTOM	Signalize that TCNT1 has reached minimum value (zero).

The 16-bit counter is mapped into two 8-bit I/O memory locations: *counter high* (TCNT1H) containing the upper eight bits of the counter, and *Counter Low* (TCNT1L) containing the lower eight bits. The TCNT1H Register can only be indirectly accessed by the CPU. When the CPU does an access to the TCNT1H I/O location, the CPU accesses the High byte temporary register (TEMP). The temporary register is updated with the TCNT1H value when the TCNT1L is read, and TCNT1H is updated with the temporary register value when TCNT1L is written. This allows the CPU to read or write the entire 16-bit counter value within one clock cycle via the 8-bit data bus. It is important to notice that there are special cases of writing to the TCNT1 Register when the counter is counting that will give unpredictable results. The special cases are described in the sections where they are of importance.

Depending on the mode of operation used, the counter is cleared, incremented, or decremented at each *timer clock* (clk_{T1}). The clk_{T1} can be generated from an external or internal clock source, selected by the *clock select* bits (CS12:0). When no clock source is selected (CS12:0 = 0) the timer is stopped. However, the TCNT1 value can be accessed by the CPU, independent of whether clk_{T1} is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the *Waveform Generation mode* bits (WGM13:0) located in the *Timer/Counter Control Registers* A and B (TCCR1A and TCCR1B). There are close connections between how the counter behaves (counts) and how waveforms

The following code examples show how to do an atomic write of the TCNT1 Register contents. Writing any of the OCR1A/B or ICR1 Registers can be done by using the same principle.

Assembly Code Example[1]

```
TIM16_WriteTCNT1:
  ; Save Global Interrupt Flag
  in  r18,SREG
  ; Disable interrupts
  cli
  ; Set TCNT1 to r17:r16
  out TCNT1H,r17
  out TCNT1L,r16
  ; Restore Global Interrupt Flag
  out SREG,r18
  ret
```

C Code Example[1]

```
void TIM16_WriteTCNT1( unsigned int i )
{
  unsigned char sreg;
  unsigned int i;
  /* Save Global Interrupt Flag */
  sreg = SREG;
  /* Disable interrupts */
  _CLI();
  /* Set TCNT1 to i */
  TCNT1 = i;
  /* Restore Global Interrupt Flag */
  SREG = sreg;
}
```

Note: 1. See "About Code Examples" on page 8.

The assembly code example requires that the r17:r16 Register pair contains the value to be written to TCNT1.

Reusing the Temporary High Byte Register

If writing to more than one 16-bit register where the High byte is the same for all registers written, then the High byte only needs to be written once. However, note that the same rule of atomic operation described previously also applies in this case.

are generated on the Output Compare Outputs OC1x. For more details about advanced counting sequences and waveform generation, see "Modes of Operation" on page 88.

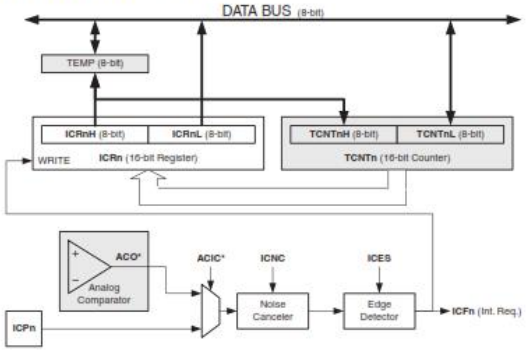
The *Timer/Counter Overflow* (TOV1) fLag is set according to the mode of operation selected by the WGM13:0 bits. TOV1 can be used for generating a CPU interrupt.

Input Capture Unit

The Timer/Counter incorporates an Input Capture unit that can capture external events and give them a time-stamp indicating time of occurrence. The external signal indicating an event, or multiple events, can be applied via the ICP1 pin or alternatively, via the Analog Comparator unit. The time-stamps can then be used to calculate frequency, duty-cycle, and other features of the signal applied. Alternatively the time-stamps can be used for creating a log of the events.

The Input Capture unit is illustrated by the block diagram shown in Figure 34. The elements of the block diagram that are not directly a part of the Input Capture unit are gray shaded. The small "n" in register and bit names indicates the Timer/Counter number.

Figure 34. Input Capture Unit Block Diagram



When a change of the logic level (an event) occurs on the *Input Capture Pin* (ICP1), alternatively on the *Analog Comparator Output* (ACO), and this change confirms to the setting of the edge detector, a capture will be triggered. When a capture is triggered, the 16-bit value of the counter (TCNT1) is written to the *Input Capture Register* (ICR1). The *Input Capture Flag* (ICF1) is set at the same system clock as the TCNT1 value is copied into ICR1 Register. If enabled (TICIE1 = 1), the Input Capture Flag generates an Input Capture interrupt. The ICF1 Flag is automatically cleared when the interrupt is executed. Alternatively the ICF1 Flag can be cleared by software by writing a logical one to its I/O bit location.

Reading the 16-bit value in the *Input Capture Register* (ICR1) is done by first reading the Low byte (ICR1L) and then the High byte (ICR1H). When the Low byte is read the High byte is copied into the High byte temporary register (TEMP). When the CPU reads the ICR1H I/O location it will access the TEMP Register.

The ICR1 Register can only be written when using a Waveform Generation mode that utilizes the ICR1 Register for defining the counter's TOP value. In these cases the *Waveform Generation mode* (WGM13:0) bits must be set before the TOP value can be written to the ICR1 Register. When writing the ICR1 Register the High byte must be written to the ICR1H I/O location before the Low byte is written to ICR1L.

For more information on how to access the 16-bit registers refer to "Accessing 16-bit Registers" on page 79.

Input Capture Pin Source

The main trigger source for the Input Capture unit is the *Input Capture Pin* (ICP1). Timer/Counter 1 can alternatively use the Analog Comparator Output as trigger source for the Input Capture unit. The Analog Comparator is selected as trigger source by setting the *Analog Comparator Input Capture* (ACIC) bit in the *Analog Comparator Control and Status Register* (ACSR). Be aware that changing trigger source can trigger a capture. The Input Capture Flag must therefore be cleared after the change.

Both the *Input Capture Pin* (ICP1) and the *Analog Comparator Output* (ACO) inputs are sampled using the same technique as for the T1 pin (Figure 30 on page 74). The edge detector is also identical. However, when the noise canceler is enabled, additional logic is inserted before the edge detector, which increases the delay by four system clock cycles. Note that the input of the noise canceler and edge detector is always enabled unless the Timer/Counter is set in a Waveform Generation mode that uses ICR1 to define TOP.

An Input Capture can be triggered by software by controlling the port of the ICP1 pin.

Noise Canceler

The noise canceler improves noise immunity by using a simple digital filtering scheme. The noise canceler input is monitored over four samples, and all four must be equal for changing the output that in turn is used by the edge detector.

The noise canceler is enabled by setting the *Input Capture Noise Canceler* (ICNC1) bit in *Timer/Counter Control Register B* (TCCR1B). When enabled the noise canceler introduces additional four system clock cycles of delay from a change applied to the input, to the update of the ICR1 Register. The noise canceler uses the system clock and is therefore not affected by the prescaler.

Using the Input Capture Unit

The main challenge when using the Input Capture unit is to assign enough processor capacity for handling the incoming events. The time between two events is critical. If the processor has not read the captured value in the ICR1 Register before the next event occurs, the ICR1 will be overwritten with a new value. In this case the result of the capture will be incorrect.

When using the Input Capture interrupt, the ICR1 Register should be read as early in the interrupt handler routine as possible. Even though the Input Capture interrupt has relatively high priority, the maximum interrupt response time is dependent on the maximum number of clock cycles it takes to handle any of the other interrupt requests.

Using the Input Capture unit in any mode of operation when the TOP value (resolution) is actively changed during operation, is not recommended.

Measurement of an external signal's duty cycle requires that the trigger edge is changed after each capture. Changing the edge sensing must be done as early as possible after the ICR1 Register has been read. After a change of the edge, the Input Capture Flag (ICF1) must be cleared by software (writing a logical one to the I/O bit location). For measuring frequency only, the clearing of the ICF1 Flag is not required (if an interrupt handler is used).

Output Compare Units

The 16-bit comparator continuously compares TCNT1 with the *Output Compare Register* (OCR1x). If TCNT equals OCR1x the comparator signals a match. A match will set the *Output Compare Flag* (OCF1x) at the next timer clock cycle. If enabled (OCIE1x = 1), the Output Compare Flag generates an Output Compare interrupt. The OCF1x Flag is automatically cleared when the interrupt is executed. Alternatively the OCF1x Flag can be cleared by software by writ-

written first. When the High byte I/O location is written by the CPU, the TEMP Register will be updated by the value written. Then when the Low byte (OCR1xL) is written to the lower eight bits, the High byte will be copied into the upper 8-bits of either the OCR1x buffer or OCR1x Compare Register in the same system clock cycle.

For more information of how to access the 16-bit registers refer to "Accessing 16-bit Registers" on page 79.

Force Output Compare

In non-PWM Waveform Generation modes, the match output of the comparator can be forced by writing a one to the *Force Output Compare* (FOC1x) bit. Forcing Compare Match will not set the OCF1x Flag or reload/clear the timer, but the OC1x pin will be updated as if a real Compare Match had occurred (the COM1x1:0 bits settings define whether the OC1x pin is set, cleared or toggled).

Compare Match Blocking by TCNT1 Write

All CPU writes to the TCNT1 Register will block any Compare Match that occurs in the next timer clock cycle, even when the timer is stopped. This feature allows OCR1x to be initialized to the same value as TCNT1 without triggering an interrupt when the Timer/Counter clock is enabled.

Using the Output Compare Unit

Since writing TCNT1 in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNT1 when using any of the Output Compare channels, independent of whether the Timer/Counter is running or not. If the value written to TCNT1 equals the OCR1x value, the Compare Match will be missed, resulting in incorrect waveform generation. Do not write the TCNT1 equal to TOP in PWM modes with variable TOP values. The Compare Match for the TOP will be ignored and the counter will continue to 0xFFFF. Similarly, do not write the TCNT1 value equal to BOTTOM when the counter is downcounting.

The setup of the OC1x should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC1x value is to use the Force Output Compare (FOC1x) strobe bits in Normal mode. The OC1x Register keeps its value even when changing between Waveform Generation modes.

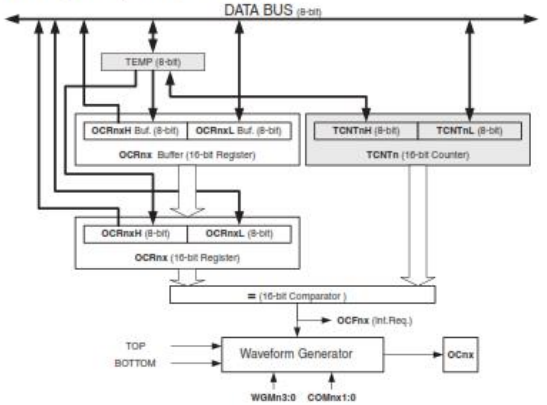
Be aware that the COM1x1:0 bits are not double buffered together with the compare value. Changing the COM1x1:0 bits will take effect immediately.

ing a logical one to its I/O bit location. The waveform generator uses the match signal to generate an output according to operating mode set by the *Waveform Generation mode* (WGM13:0) bits and *Compare Output mode* (COM1x1:0) bits. The TOP and BOTTOM signals are used by the waveform generator for handling the special cases of the extreme values in some modes of operation (See "Modes of Operation" on page 88.)

A special feature of Output Compare unit A allows it to define the Timer/Counter TOP value (i.e. counter resolution). In addition to the counter resolution, the TOP value defines the period time for waveforms generated by the waveform generator.

Figure 35 shows a block diagram of the Output Compare unit. The small "n" in the register and bit names indicates the device number (n = 1 for Timer/Counter 1), and the "x" indicates Output Compare unit (A/B). The elements of the block diagram that are not directly a part of the Output Compare unit are gray shaded.

Figure 35. Output Compare Unit, Block Diagram



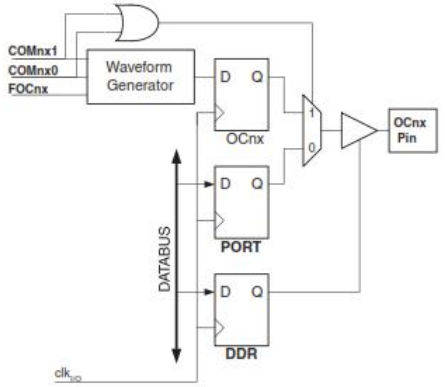
The OCR1x Register is double buffered when using any of the twelve *Pulse Width Modulation* (PWM) modes. For the normal and *Clear Timer on Compare* (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR1x Compare Register to either TOP or BOTTOM of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCR1x Register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR1x Buffer Register, and if double buffering is disabled the CPU will access the OCR1x directly. The content of the OCR1x (Buffer or Compare) Register is only changed by a write operation (the Timer/Counter does not update this register automatically as the TCNT1 and ICR1 Register). Therefore OCR1x is not read via the High byte temporary register (TEMP). However, it is a good practice to read the Low byte first as when accessing other 16-bit registers. Writing the OCR1x Registers must be done via the TEMP Register since the compare of all 16-bit is done continuously. The High byte (OCR1xH) has to be

Compare Match Output Unit

The *Compare Output mode* (COM1x1:0) bits have two functions. The waveform generator uses the COM1x1:0 bits for defining the Output Compare (OC1x) state at the next Compare Match. Secondly the COM1x1:0 bits control the OC1x pin output source. Figure 36 shows a simplified schematic of the logic affected by the COM1x1:0 bit setting. The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O Port Control Registers (DDR and PORT) that are affected by the COM1x1:0 bits are shown. When referring to the OC1x state, the reference is for the internal OC1x Register, not the OC1x pin. If a System Reset occur, the OC1x Register is reset to "0".

Figure 36. Compare Match Output Unit, Schematic



The general I/O port function is overridden by the Output Compare (OC1x) from the waveform generator if either of the COM1x1:0 bits are set. However, the OC1x pin direction (input or output) is still controlled by the *Data Direction Register* (DDR) for the port pin. The Data Direction Register bit for the OC1x pin (DDR_OC1x) must be set as output before the OC1x value is visible on the pin. The port override function is generally independent of the Waveform Generation mode, but there are some exceptions. Refer to Table 36, Table 37 and Table 38 for details.

The design of the Output Compare Pin logic allows initialization of the OC1x state before the output is enabled. Note that some COM1x1:0 bit settings are reserved for certain modes of operation. See "16-bit Timer/Counter Register Description" on page 96.

The COM1x1:0 bits have no effect on the Input Capture unit.

Compare Output Mode and Waveform Generation

The waveform generator uses the COM1x1:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COM1x1:0 = 0 tells the waveform generator that no action on the OC1x Register is to be performed on the next Compare Match. For compare output actions in the non-PWM modes refer to Table 36 on page 97. For fast PWM mode refer to Table 37 on page 97, and for phase correct and phase and frequency correct PWM refer to Table 38 on page 98.

A change of the COM1x1:0 bits state will have effect at the first Compare Match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOC1x strobe bits.

Modes of Operation

The mode of operation (i.e., the behavior of the Timer/Counter and the Output Compare pins) is defined by the combination of the *Waveform Generation mode* (WGM13:0) and *Compare Output mode* (COM1x1:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM1x1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM1x1:0 bits control whether the output should be set, cleared or toggle at a Compare Match. See "Compare Match Output Unit" on page 87.

For detailed timing information refer to "Timer/Counter Timing Diagrams" on page 95.

Normal Mode

The simplest mode of operation is the *Normal* mode (WGM13:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 16-bit value (MAX = 0xFFFF) and then restarts from the BOTTOM (0x0000). In normal operation the *Timer/Counter Overflow Flag* (TOV1) will be set in the same timer clock cycle as the TCNT1 becomes zero. The TOV1 Flag in this case behaves like a 17th bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV1 Flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime.

The Input Capture unit is easy to use in Normal mode. However, observe that the maximum interval between the external events must not exceed the resolution of the counter. If the interval between events are too long, the timer overflow interrupt or the prescaler must be used to extend the resolution for the capture unit.

The Output Compare units can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

Clear Timer on Compare Match (CTC) Mode

In *Clear Timer on Compare* or CTC mode (WGM13:0 = 4 or 12), the OCR1A or ICR1 Register are used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT1) matches either the OCR1A (WGM13:0 = 4) or the ICR1 (WGM13:0 = 12). The OCR1A or ICR1 define the top value for the counter, hence also its resolution. This mode allows greater control of the Compare Match output frequency. It also simplifies the operation of counting external events.

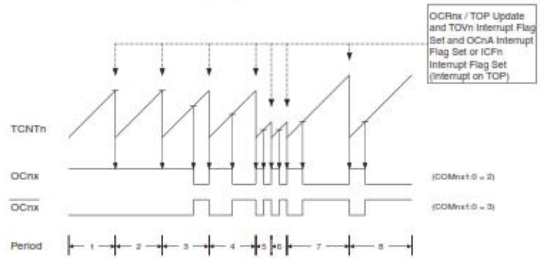
The timing diagram for the CTC mode is shown in Figure 37. The counter value (TCNT1) increases until a Compare Match occurs with either OCR1A or ICR1, and then counter (TCNT1) is cleared.

imum resolution is 16-bit (ICR1 or OCR1A set to MAX). The PWM resolution in bits can be calculated by using the following equation:

$$R_{FPWM} = \frac{\log(TOP+1)}{\log(2)}$$

In fast PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGM13:0 = 5, 6, or 7), the value in ICR1 (WGM13:0 = 14), or the value in OCR1A (WGM13:0 = 15). The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 38. The figure shows fast PWM mode when OCR1A or ICR1 is used to define TOP. The TCNT1 value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT1 slopes represent compare matches between OCR1x and TCNT1. The OC1x Interrupt Flag will be set when a Compare Match occurs.

Figure 38. Fast PWM Mode, Timing Diagram

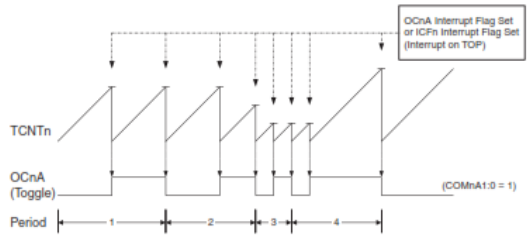


The Timer/Counter Overflow Flag (TOV1) is set each time the counter reaches TOP. In addition the OCF1A or ICF1 Flag is set at the same timer clock cycle as TOV1 is set when either OCR1A or ICR1 is used for defining the TOP value. If one of the interrupts are enabled, the interrupt handler routine can be used for updating the TOP and compare values.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a Compare Match will never occur between the TCNT1 and the OCR1x. Note that when using fixed TOP values the unused bits are masked to zero when any of the OCR1x Registers are written.

The procedure for updating ICR1 differs from updating OCR1A when used for defining the TOP value. The ICR1 Register is not double buffered. This means that if ICR1 is changed to a low value when the counter is running with none or a low prescaler value, there is a risk that the new ICR1 value written is lower than the current value of TCNT1. The result will then be that the counter will miss the Compare Match at the TOP value. The counter will then have to count to the MAX value (0xFFFF) and wrap around starting at 0x0000 before the Compare Match can occur. The OCR1A Register, however, is double buffered. This feature allows the OCR1A I/O location to be written anytime. When the OCR1A I/O location is written the value written will be put into the OCR1A Buffer Register. The OCR1A Compare Register will then be updated with the value in the Buffer Register at the next timer clock cycle the TCNT1 matches TOP. The update is done at the same timer clock cycle as the TCNT1 is cleared and the TOV1 Flag is set.

Figure 37. CTC Mode, Timing Diagram



An interrupt can be generated at each time the counter value reaches the TOP value by either using the OCF1A or ICF1 Flag according to the register used to define the TOP value. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing the TOP to a value close to BOTTOM when the counter is running with none or a low prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written to OCR1A or ICR1 is lower than the current value of TCNT1, the counter will miss the Compare Match. The counter will then have to count to its maximum value (0xFFFF) and wrap around starting at 0x0000 before the Compare Match can occur. In many cases this feature is not desirable. An alternative will then be to use the fast PWM mode using OCR1A for defining TOP (WGM13:0 = 15) since the OCR1A then will be double buffered.

For generating a waveform output in CTC mode, the OC1A output can be set to toggle its logical level on each Compare Match by setting the Compare Output mode bits to toggle mode (COM1A1:0 = 1). The OC1A value will not be visible on the port pin unless the data direction for the pin is set to output (DDR_OC1A = 1). The waveform generated will have a maximum frequency of $f_{OC1A} = f_{clk\_I/O}/2$ when OCR1A is set to zero (0x0000). The waveform frequency is defined by the following equation:

$$f_{OCnA} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCRnA)}$$

The N variable represents the prescaler factor (1, 8, 64, 256, or 1024).

As for the Normal mode of operation, the TOV1 Flag is set in the same timer clock cycle that the counter counts from MAX to 0x0000.

Fast PWM Mode

The *fast Pulse Width Modulation* or fast PWM mode (WGM13:0 = 5, 6, 7, 14, or 15) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC1x) is cleared on the Compare Match between TCNT1 and OCR1x, and set at BOTTOM. In inverting Compare Output mode output is set on Compare Match and cleared at BOTTOM. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct and phase and frequency correct PWM modes that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), hence reduces total system cost.

The PWM resolution for fast PWM can be fixed to 8-, 9-, or 10-bit, or defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or OCR1A set to 0x0003), and the max-

Using the ICR1 Register for defining TOP works well when using fixed TOP values. By using ICR1, the OCR1A Register is free to be used for generating a PWM output on OC1A. However, if the base PWM frequency is actively changed (by changing the TOP value), using the OCR1A as TOP is clearly a better choice due to its double buffer feature.

In fast PWM mode, the compare units allow generation of PWM waveforms on the OC1x pins. Setting the COM1x1:0 bits to 2 will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x1:0 to 3. See Table 37 on page 97. The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OC1x). The PWM waveform is generated by setting (or clearing) the OC1x Register at the Compare Match between OCR1x and TCNT1, and clearing (or setting) the OC1x Register at the timer clock cycle the counter is cleared (changes from TOP to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot (1 + TOP)}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCR1x Register represents special cases when generating a PWM waveform output in the fast PWM mode. If the OCR1x is set equal to BOTTOM (0x0000) the output will be a narrow spike for each TOP+1 timer clock cycle. Setting the OCR1x equal to TOP will result in a constant high or low output (depending on the polarity of the output set by the COM1x1:0 bits.)

A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC1A to toggle its logical level on each Compare Match (COM1A1:0 = 1). This applies only if OCR1A is used to define the TOP value (WGM13:0 = 15). The waveform generated will have a maximum frequency of $f_{OC1A} = f_{clk\_I/O}/2$ when OCR1A is set to zero (0x0000). This feature is similar to the OC1A toggle in CTC mode, except the double buffer feature of the Output Compare unit is enabled in the fast PWM mode.

Phase Correct PWM Mode

The *phase correct Pulse Width Modulation* or phase correct PWM mode (WGM13:0 = 1, 2, 3, 10, or 11) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is, like the phase and frequency correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC1x) is cleared on the Compare Match between TCNT1 and OCR1x while upcounting, and set on the Compare Match while downcounting. In inverting Output Compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

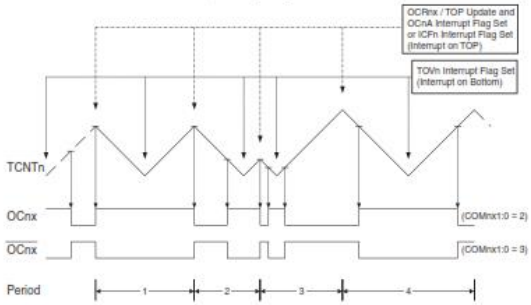
The PWM resolution for the phase correct PWM mode can be fixed to 8-, 9-, or 10-bit, or defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or OCR1A set to 0x0003), and the maximum resolution is 16-bit (ICR1 or OCR1A set to MAX). The PWM resolution in bits can be calculated by using the following equation:

$$R_{PCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In phase correct PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGM13:0 = 1, 2, or 3), the value in ICR1 (WGM13:0 = 10), or the value in OCR1A (WGM13:0 = 11). The counter has then reached the TOP and changes the count direction. The TCNT1 value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 39. The figure shows phase correct PWM mode when OCR1A or ICR1 is used to define TOP. The TCNT1 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The

diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT1 slopes represent compare matches between OCR1x and TCNT1. The OC1x Interrupt Flag will be set when a Compare Match occurs.

Figure 39. Phase Correct PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV1) is set each time the counter reaches BOTTOM. When either OCR1A or ICR1 is used for defining the TOP value, the OC1A or ICF1 Flag is set accordingly at the same timer clock cycle as the OCR1x Registers are updated with the double buffer value (at TOP). The Interrupt Flags can be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a Compare Match will never occur between the TCNT1 and the OCR1x. Note that when using fixed TOP values, the unused bits are masked to zero when any of the OCR1x Registers are written. As the third period shown in Figure 39 illustrates, changing the TOP actively while the Timer/Counter is running in the Phase Correct mode can result in an unsymmetrical output. The reason for this can be found in the time of update of the OCR1x Register. Since the OCR1x update occurs at TOP, the PWM period starts and ends at TOP. This implies that the length of the falling slope is determined by the previous TOP value, while the length of the rising slope is determined by the new TOP value. When these two values differ the two slopes of the period will differ in length. The difference in length gives the unsymmetrical result on the output.

It is recommended to use the Phase and Frequency Correct mode instead of the Phase Correct mode when changing the TOP value while the Timer/Counter is running. When using a static TOP value there are practically no differences between the two modes of operation.

In phase correct PWM mode, the compare units allow generation of PWM waveforms on the OC1x pins. Setting the COM1x1:0 bits to 2 will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x1:0 to 3. See Table 38 on page 98. The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OC1x). The PWM waveform is generated by setting (or clearing) the OC1x Register at the Compare Match between OCR1x and TCNT1 when the counter increments, and clearing (or setting) the OC1x Register at Compare Match between OCR1x and TCNT1 when

the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnxPCPWM} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot TOP}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCR1x Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR1x is set equal to BOTTOM the output will be continuously low and if set equal to TOP the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

If OCR1A is used to define the TOP value (WMG13:0 = 11) and COM1A1:0 = 1, the OC1A output will toggle with a 50% duty cycle.

Phase and Frequency Correct PWM Mode

The *phase and frequency correct Pulse Width Modulation*, or phase and frequency correct PWM mode (WGM13:0 = 8 or 9) provides a high resolution phase and frequency correct PWM waveform generation option. The phase and frequency correct PWM mode is, like the phase correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC1x) is cleared on the Compare Match between TCNT1 and OCR1x while upcounting, and set on the Compare Match while downcounting. In inverting Compare Output mode, the operation is inverted. The dual-slope operation gives a lower maximum operation frequency compared to the single-slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

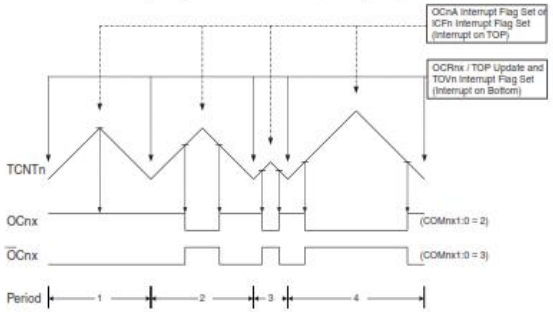
The main difference between the phase correct, and the phase and frequency correct PWM mode is the time the OCR1x Register is updated by the OCR1x Buffer Register, (see Figure 39 and Figure 40).

The PWM resolution for the phase and frequency correct PWM mode can be defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or OCR1A set to 0x0003), and the maximum resolution is 16-bit (ICR1 or OCR1A set to MAX). The PWM resolution in bits can be calculated using the following equation:

$$R_{PFCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In phase and frequency correct PWM mode the counter is incremented until the counter value matches either the value in ICR1 (WGM13:0 = 8), or the value in OCR1A (WGM13:0 = 9). The counter has then reached the TOP and changes the count direction. The TCNT1 value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct and frequency correct PWM mode is shown on Figure 40. The figure shows phase and frequency correct PWM mode when OCR1A or ICR1 is used to define TOP. The TCNT1 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT1 slopes represent compare matches between OCR1x and TCNT1. The OC1x Interrupt Flag will be set when a Compare Match occurs.

Figure 40. Phase and Frequency Correct PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV1) is set at the same timer clock cycle as the OCR1x Registers are updated with the double buffer value (at BOTTOM). When either OCR1A or ICR1 is used for defining the TOP value, the OC1A or ICF1 Flag set when TCNT1 has reached TOP. The Interrupt Flags can then be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a Compare Match will never occur between the TCNT1 and the OCR1x.

As Figure 40 shows the output generated is, in contrast to the Phase Correct mode, symmetrical in all periods. Since the OCR1x Registers are updated at BOTTOM, the length of the rising and the falling slopes will always be equal. This gives symmetrical output pulses and is therefore frequency correct.

Using the ICR1 Register for defining TOP works well when using fixed TOP values. By using ICR1, the OCR1A Register is free to be used for generating a PWM output on OC1A. However, if the base PWM frequency is actively changed by changing the TOP value, using the OCR1A as TOP is clearly a better choice due to its double buffer feature.

In phase and frequency correct PWM mode, the compare units allow generation of PWM waveforms on the OC1x pins. Setting the COM1x1:0 bits to 2 will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x1:0 to 3. See Table 38 on page 98. The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OC1x). The PWM waveform is generated by setting (or clearing) the OC1x Register at the Compare Match between OCR1x and TCNT1 when the counter increments, and clearing (or setting) the OC1x Register at Compare Match between OCR1x and TCNT1 when the counter decrements. The PWM frequency for the output when using phase and frequency correct PWM can be calculated by the following equation:

$$f_{OCnxPFCPWM} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot TOP}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCR1x Register represents special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR1x is set equal to BOTTOM the

output will be continuously low and if set equal to TOP the output will be set to high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

If OCR1A is used to define the TOP value (WGM13:0 = 9) and COM1A1:0 = 1, the OC1A output will toggle with a 50% duty cycle.

Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock (clk_{T1}) is therefore shown as a clock enable signal in the following figures. The figures include information on when Interrupt Flags are set, and when the OCR1x Register is updated with the OCR1x buffer value (only for modes utilizing double buffering). Figure 41 shows a timing diagram for the setting of OCF1x.

Figure 41. Timer/Counter Timing Diagram, Setting of OCF1x, no Prescaling

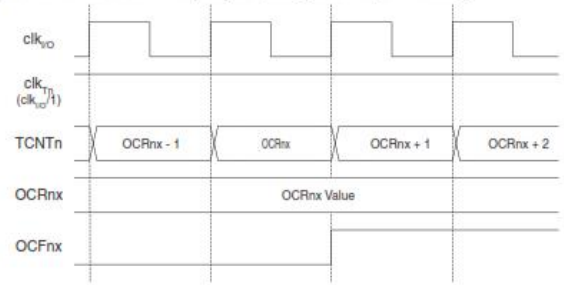


Figure 42 shows the same timing data, but with the prescaler enabled.

Figure 42. Timer/Counter Timing Diagram, Setting of OCF1x, with Prescaler ($f_{clk\_I/O}/8$)

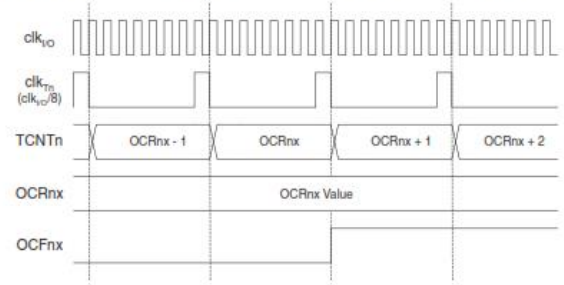


Figure 43 shows the count sequence close to TOP in various modes. When using phase and frequency correct PWM mode the OCR1x Register is updated at BOTTOM. The timing diagrams

will be the same, but TOP should be replaced by BOTTOM, TOP-1 by BOTTOM+1 and so on. The same renaming applies for modes that set the TOV1 Flag at BOTTOM.

Figure 43. Timer/Counter Timing Diagram, no Prescaling

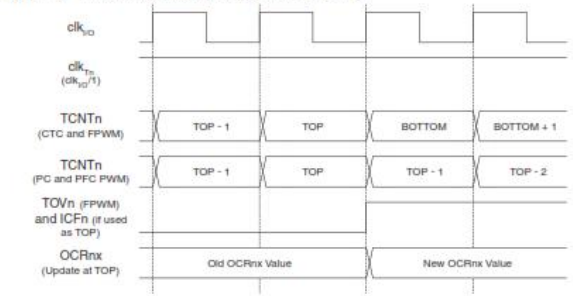
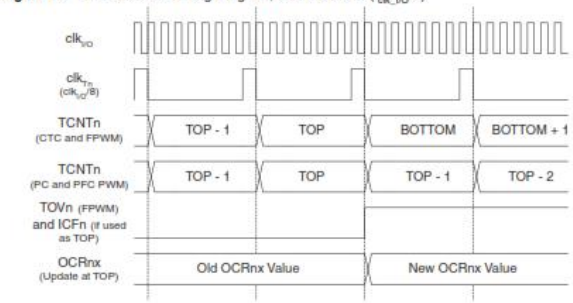


Figure 44 shows the same timing data, but with the prescaler enabled.

Figure 44. Timer/Counter Timing Diagram, with Prescaler ($f_{clk\_I/O}/8$)



16-bit Timer/Counter Register Description

Timer/Counter 1 Control Register A – TCCR1A

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	

Initial Value 0 0 0 0 0 0 0 0

- **Bit 7:6 – COM1A1:0: Compare Output Mode for channel A**
- **Bit 5:4 – COM1B1:0: Compare Output Mode for channel B**

The COM1A1:0 and COM1B1:0 control the Output Compare Pins (OC1A and OC1B respectively) behavior. If one or both of the COM1A1:0 bits are written to one, the OC1A output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COM1B1:0 bit are written to one, the OC1B output overrides the normal port functionality of the I/O pin it is connected to. However, note that the *Data Direction Register* (DDR) bit corresponding to the OC1A or OC1B pin must be set in order to enable the output driver.

When the OC1A or OC1B is connected to the pin, the function of the COM1x1:0 bits is dependent of the WGM13:0 bits setting. Table 36 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to a normal or a CTC mode (non-PWM).

Table 36. Compare Output Mode, Non-PWM

COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on Compare Match
1	0	Clear OC1A/OC1B on Compare Match (Set output to low level)
1	1	Set OC1A/OC1B on Compare Match (Set output to high level)

Table 37 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to the fast PWM mode.

Table 37. Compare Output Mode, Fast PWM[1]

COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at BOTTOM, (non-inverting mode)
1	1	Set OC1A/OC1B on Compare Match, clear OC1A/OC1B at BOTTOM, (inverting mode)

Note: 1. A special case occurs when OCR1A/OCR1B equals TOP and COM1A1/COM1B1 is set. In this case the Compare Match is ignored, but the set or clear is done at BOTTOM. See "Fast PWM Mode" on page 89. for more details.

Table 38 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to the phase correct or the phase and frequency correct, PWM mode.

Table 38. Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM[1]

COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 9 or 14: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match when up-counting. Set OC1A/OC1B on Compare Match when downcounting.
1	1	Set OC1A/OC1B on Compare Match when up-counting. Clear OC1A/OC1B on Compare Match when downcounting.

Note: 1. A special case occurs when OCR1A/OCR1B equals TOP and COM1A1/COM1B1 is set. See "Phase Correct PWM Mode" on page 91. for more details.

- **Bit 3 – FOC1A: Force Output Compare for channel A**
- **Bit 2 – FOC1B: Force Output Compare for channel B**

The FOC1A/FOC1B bits are only active when the WGM13:0 bits specifies a non-PWM mode. However, for ensuring compatibility with future devices, these bits must be set to zero when TCCR1A is written when operating in a PWM mode. When writing a logical one to the FOC1A/FOC1B bit, an immediate Compare Match is forced on the waveform generation unit. The OC1A/OC1B output is changed according to its COM1x1:0 bits setting. Note that the FOC1A/FOC1B bits are implemented as strobes. Therefore it is the value present in the COM1x1:0 bits that determine the effect of the forced compare.

A FOC1A/FOC1B strobe will not generate any interrupt nor will it clear the timer in Clear Timer on Compare Match (CTC) mode using OCR1A as TOP.

The FOC1A/FOC1B bits are always read as zero.

- **Bit 1:0 – WGM11:0: Waveform Generation Mode**

Combined with the WGM13:2 bits found in the TCCR1B Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see Table 39. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare Match (CTC) mode, and three types of Pulse Width Modulation (PWM) modes. (See "Modes of Operation" on page 88.)

Table 39. Waveform Generation Mode Bit Description

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation[1]	TOP	Update of OCR1x	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP

Table 39. Waveform Generation Mode Bit Description

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation[1]	TOP	Update of OCR1x	TOV1 Flag Set on
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

Timer/Counter 1 Control Register B – 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	

- **Bit 7 – ICNC1: Input Capture Noise Canceler**

Setting this bit (to one) activates the Input Capture Noise Canceler. When the noise canceler is activated, the input from the Input Capture Pin (ICP1) is filtered. The filter function requires four successive equal valued samples of the ICP1 pin for changing its output. The Input Capture is therefore delayed by four Oscillator cycles when the noise canceler is enabled.

- **Bit 6 – ICES1: Input Capture Edge Select**

This bit selects which edge on the Input Capture Pin (ICP1) that is used to trigger a capture event. When the ICES1 bit is written to zero, a falling (negative) edge is used as trigger, and when the ICES1 bit is written to one, a rising (positive) edge will trigger the capture.

When a capture is triggered according to the ICES1 setting, the counter value is copied into the Input Capture Register (ICR1). The event will also set the Input Capture Flag (ICF1), and this can be used to cause an Input Capture Interrupt, if this interrupt is enabled.

When the ICR1 is used as TOP value (see description of the WGM13:0 bits located in the TCCR1A and the TCCR1B Register), the ICP1 is disconnected and consequently the Input Capture function is disabled.

- **Bit 5 – Reserved Bit**

This bit is reserved for future use. For ensuring compatibility with future devices, this bit must be written to zero when TCCR1B is written.

- **Bit 4:3 – WGM13:2: Waveform Generation Mode**

See TCCR1A Register description.

- **Bit 2:0 – CS12:0: Clock Select**

The three clock select bits select the clock source to be used by the Timer/Counter, see Figure 41 and Figure 42.

Table 40. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

If external pin modes are used for the Timer/Counter1, transitions on the T1 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

Input Capture Register 1 – ICR1H and ICR1L

Bit	7	6	5	4	3	2	1	0	
	ICR1[15:8]								ICR1H
	ICR1[7:0]								ICR1L
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	

The Input Capture is updated with the counter (TCNT1) value each time an event occurs on the ICP1 pin (or optionally on the Analog Comparator Output for Timer/Counter1). The Input Capture can be used for defining the counter TOP value.

The Input Capture Register is 16-bit in size. To ensure that both the high and Low bytes are read simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary High byte Register (TEMP). This temporary register is shared by all the other 16-bit registers. See "Accessing 16-bit Registers" on page 79.

Timer/Counter Interrupt Mask Register – TIMSK[1]

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	–	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Note: 1. This register contains interrupt control bits for several Timer/Counters, but only Timer1 bits are described in this section. The remaining bits are described in their respective timer sections.

- **Bit 5 – TICIE1: Timer/Counter1, Input Capture Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Input Capture Interrupt is enabled. The corresponding Interrupt Vector (see "Interrupts" on page 46) is executed when the ICF1 Flag, located in TIFR, is set.

- **Bit 4 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare A match interrupt is enabled. The corresponding Interrupt Vector (see "Interrupts" on page 46) is executed when the OCF1A Flag, located in TIFR, is set.

- **Bit 3 – OCIE1B: Timer/Counter1, Output Compare B Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare B match interrupt is enabled. The corresponding Interrupt Vector (see "Interrupts" on page 46) is executed when the OCF1B Flag, located in TIFR, is set.

- **Bit 2 – TOIE1: Timer/Counter1, Overflow Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Overflow Interrupt is enabled. The corresponding Interrupt Vector (see "Interrupts" on page 46) is executed when the TOV1 Flag, located in TIFR, is set.

Timer/Counter Interrupt Flag Register – TIFR[1]

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	–	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Note: 1. This register contains flag bits for several Timer/Counters, but only Timer1 bits are described in this section. The remaining bits are described in their respective timer sections.

- **Bit 5 – ICF1: Timer/Counter1, Input Capture Flag**

Timer/Counter 1 – TCNT1H and TCNT1L

Bit	7	6	5	4	3	2	1	0	
	TCNT1[15:8]								TCNT1H
	TCNT1[7:0]								TCNT1L
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	

The two *Timer/Counter* I/O locations (TCNT1H and TCNT1L, combined TCNT1) give direct access, both for read and for write operations, to the Timer/Counter unit 16-bit counter. To ensure that both the high and Low bytes are read and written simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary High byte Register (TEMP). This temporary register is shared by all the other 16-bit registers. See "Accessing 16-bit Registers" on page 79.

Modifying the counter (TCNT1) while the counter is running introduces a risk of missing a Compare Match between TCNT1 and one of the OCR1x Registers.

Writing to the TCNT1 Register blocks (removes) the Compare Match on the following timer clock for all compare units.

Output Compare Register 1 A – OCR1AH and OCR1AL

Bit	7	6	5	4	3	2	1	0	
	OCR1A[15:8]								OCR1AH
	OCR1A[7:0]								OCR1AL
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	

Output Compare Register 1 B – OCR1BH and OCR1BL

Bit	7	6	5	4	3	2	1	0	
	OCR1B[15:8]								OCR1BH
	OCR1B[7:0]								OCR1BL
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	

The Output Compare Registers contain a 16-bit value that is continuously compared with the counter value (TCNT1). A match can be used to generate an Output Compare Interrupt, or to generate a waveform output on the OC1x pin.

The Output Compare Registers are 16-bit in size. To ensure that both the high and Low bytes are written simultaneously when the CPU writes to these registers, the access is performed using an 8-bit temporary High byte Register (TEMP). This temporary register is shared by all the other 16-bit registers. See "Accessing 16-bit Registers" on page 79.

This flag is set when a capture event occurs on the ICP1 pin. When the Input Capture Register (ICR1) is set by the WGM13:0 to be used as the TOP value, the ICF1 Flag is set when the counter reaches the TOP value.

ICF1 is automatically cleared when the Input Capture Interrupt Vector is executed. Alternatively, ICF1 can be cleared by writing a logic one to its bit location.

- **Bit 4 – OCF1A: Timer/Counter1, Output Compare A Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the Output Compare Register A (OCR1A).

Note that a Forced Output Compare (FOC1A) strobe will not set the OCF1A Flag.

OCF1A is automatically cleared when the Output Compare Match A Interrupt Vector is executed. Alternatively, OCF1A can be cleared by writing a logic one to its bit location.

- **Bit 3 – OCF1B: Timer/Counter1, Output Compare B Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the Output Compare Register B (OCR1B).

Note that a Forced Output Compare (FOC1B) strobe will not set the OCF1B Flag.

OCF1B is automatically cleared when the Output Compare Match B Interrupt Vector is executed. Alternatively, OCF1B can be cleared by writing a logic one to its bit location.

- **Bit 2 – TOV1: Timer/Counter1, Overflow Flag**

The setting of this flag is dependent of the WGM13:0 bits setting. In normal and CTC modes, the TOV1 Flag is set when the timer overflows. Refer to Table 39 on page 98 for the TOV1 Flag behavior when using another WGM13:0 bit setting.

TOV1 is automatically cleared when the Timer/Counter1 Overflow Interrupt Vector is executed. Alternatively, TOV1 can be cleared by writing a logic one to its bit location.

USART

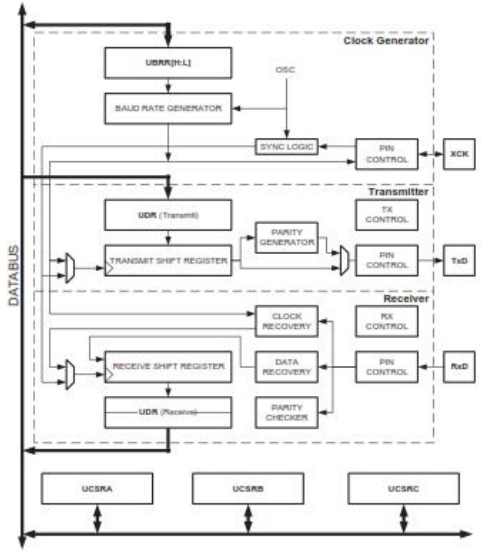
The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly-flexible serial communication device. The main features are:

- **Full Duplex Operation (Independent Serial Receive and Transmit Registers)**
- **Asynchronous or Synchronous Operation**
- **Master or Slave Clocked Synchronous Operation**
- **High Resolution Baud Rate Generator**
- **Supports Serial Frames with 5, 6, 7, 8, or 9 Databits and 1 or 2 Stop Bits**
- **Odd or Even Parity Generation and Parity Check Supported by Hardware**
- **Data OverRun Detection**
- **Framing Error Detection**
- **Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter**
- **Three Separate Interrupts on TX Complete, TX Data Register Empty and RX Complete**
- **Multi-processor Communication Mode**
- **Double Speed Asynchronous Communication Mode**

Overview

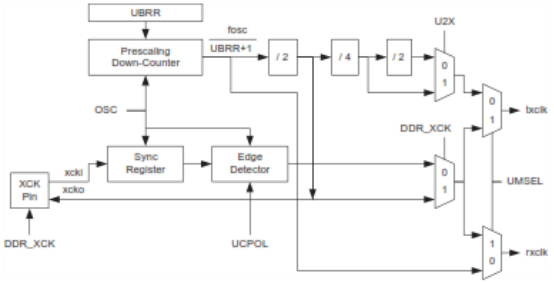
A simplified block diagram of the USART Transmitter is shown in Figure 61. CPU accessible I/O Registers and I/O pins are shown in bold.

Figure 61. USART Block Diagram[1]



Note: 1. Refer to "Pin Configurations" on page 2, Table 30 on page 64, and Table 29 on page 64 for USART pin placement.

Figure 62. Clock Generation Logic, Block Diagram



Signal description:

- **txclk** Transmitter clock. (Internal Signal)
- **rxclk** Receiver base clock. (Internal Signal)
- **xcki** Input from XCK pin (internal Signal). Used for synchronous slave operation.
- **xcko** Clock output to XCK pin (Internal Signal). Used for synchronous master operation.
- **fosc** XTAL pin frequency (System Clock).

Internal Clock Generation – The Baud Rate Generator

Internal clock generation is used for the asynchronous and the Synchronous Master modes of operation. The description in this section refers to Figure 62.

The USART Baud Rate Register (UBRR) and the down-counter connected to it function as a programmable prescaler or baud rate generator. The down-counter, running at system clock (fosc), is loaded with the UBRR value each time the counter has counted down to zero or when the UBRRL Register is written. A clock is generated each time the counter reaches zero. This clock is the baud rate generator clock output (= fosc/(UBRR+1)). The Transmitter divides the baud rate generator clock output by 2, 8, or 16 depending on mode. The baud rate generator output is used directly by the Receiver's clock and data recovery units. However, the recovery units use a state machine that uses 2, 8, or 16 states depending on mode set by the state of the UMSEL, U2X and DDR_XCK bits.

Table 52 contains equations for calculating the baud rate (in bits per second) and for calculating the UBRR value for each mode of operation using an internally generated clock source.

The dashed boxes in the block diagram separate the three main parts of the USART (listed from the top): Clock generator, Transmitter and Receiver. Control Registers are shared by all units. The clock generation logic consists of synchronization logic for external clock input used by synchronous slave operation, and the baud rate generator. The XCK (transfer clock) pin is only used by synchronous transfer mode. The Transmitter consists of a single write buffer, a serial Shift Register, Parity Generator and control logic for handling different serial frame formats. The write buffer allows a continuous transfer of data without any delay between frames. The Receiver is the most complex part of the USART module due to its clock and data recovery units. The recovery units are used for asynchronous data reception. In addition to the recovery units, the Receiver includes a parity checker, control logic, a Shift Register and a two level receive buffer (UDR). The Receiver supports the same frame formats as the Transmitter, and can detect Frame Error, Data OverRun and Parity Errors.

AVR USART vs. AVR UART – Compatibility

The USART is fully compatible with the AVR UART regarding:

- Bit locations inside all USART Registers.
- Baud Rate Generation.
- Transmitter Operation.
- Transmit Buffer Functionality.
- Receiver Operation.

However, the receive buffering has two improvements that will affect the compatibility in some special cases:

- A second Buffer Register has been added. The two Buffer Registers operate as a circular FIFO buffer. Therefore the UDR must only be read once for each incoming data! More important is the fact that the Error Flags (FE and DOR) and the ninth data bit (RXB8) are buffered with the data in the receive buffer. Therefore the status bits must always be read before the UDR Register is read. Otherwise the error status will be lost since the buffer state is lost.
- The Receiver Shift Register can now act as a third buffer level. This is done by allowing the received data to remain in the serial Shift Register (see Figure 61) if the Buffer Registers are full, until a new start bit is detected. The USART is therefore more resistant to Data OverRun (DOR) error conditions.

The following control bits have changed name, but have same functionality and register location:

- CHR9 is changed to UCSZ2.
- OR is changed to DOR.

Clock Generation

The clock generation logic generates the base clock for the Transmitter and Receiver. The USART supports four modes of clock operation: normal asynchronous, double speed asynchronous, Master synchronous and Slave Synchronous mode. The UMSEL bit in USART Control and Status Register C (UCSRC) selects between asynchronous and synchronous operation. Double speed (Asynchronous mode only) is controlled by the U2X found in the UCSRA Register. When using Synchronous mode (UMSEL = 1), the Data Direction Register for the XCK pin (DDR_XCK) controls whether the clock source is internal (Master mode) or external (Slave mode). The XCK pin is only active when using Synchronous mode.

Figure 62 shows a block diagram of the clock generation logic.

Table 52. Equations for Calculating Baud Rate Register Setting

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

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps).

BAUD Baud rate (in bits per second, bps)

f_{OSC} System Oscillator clock frequency

UBRR Contents of the UBRRH and UBRRL Registers, (0 - 4095)

Some examples of UBRR values for some system clock frequencies are found in Table 60 (see page 159).

Double Speed Operation (U2X)

The transfer rate can be doubled by setting the U2X bit in UCSRA. Setting this bit only has effect for the asynchronous operation. Set this bit to zero when using synchronous operation.

Setting this bit will reduce the divisor of the baud rate divider from 16 to 8, effectively doubling the transfer rate for asynchronous communication. Note however that the Receiver will in this case only use half the number of samples (reduced from 16 to 8) for data sampling and clock recovery, and therefore a more accurate baud rate setting and system clock are required when this mode is used. For the Transmitter, there are no downsides.

External Clock

External clocking is used by the Synchronous Slave modes of operation. The description in this section refers to Figure 62 for details.

External clock input from the XCK pin is sampled by a synchronization register to minimize the chance of meta-stability. The output from the synchronization register must then pass through an edge detector before it can be used by the Transmitter and Receiver. This process introduces a two CPU clock period delay and therefore the maximum external XCK clock frequency is limited by the following equation:

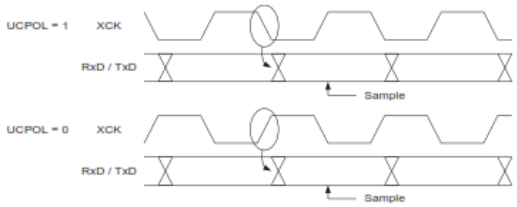
$$f_{XCK} < \frac{f_{OSC}}{4}$$

Note that f_{osc} depends on the stability of the system clock source. It is therefore recommended to add some margin to avoid possible loss of data due to frequency variations.

Synchronous Clock Operation

When Synchronous mode is used (UMSEL = 1), the XCK pin will be used as either clock input (Slave) or clock output (Master). The dependency between the clock edges and data sampling or data change is the same. The basic principle is that data input (on RxD) is sampled at the opposite XCK clock edge of the edge the data output (TxD) is changed.

Figure 63. Synchronous Mode XCK Timing



The UCPOL bit UCRSC selects which XCK clock edge is used for data sampling and which is used for data change. As Figure 63 shows, when UCPOL is zero the data will be changed at rising XCK edge and sampled at falling XCK edge. If UCPOL is set, the data will be changed at falling XCK edge and sampled at rising XCK edge.

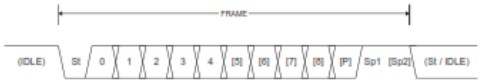
Frame Formats

A serial frame is defined to be one character of data bits with synchronization bits (start and stop bits), and optionally a parity bit for error checking. The USART accepts all 30 combinations of the following as valid frame formats:

- 1 start bit
- 5, 6, 7, 8, or 9 data bits
- no, even or odd parity bit
- 1 or 2 stop bits

A frame starts with the start bit followed by the least significant data bit. Then the next data bits, up to a total of nine, are succeeding, ending with the most significant bit. If enabled, the parity bit is inserted after the data bits, before the stop bits. When a complete frame is transmitted, it can be directly followed by a new frame, or the communication line can be set to an idle (high) state. Figure 64 illustrates the possible combinations of the frame formats. Bits inside brackets are optional.

Figure 64. Frame Formats



St Start bit, always low.

(n) Data bits (0 to 8).

P Parity bit. Can be odd or even.

Sp Stop bit, always high.

Assembly Code Example[1]

```
USART_Init:
  ; Set baud rate
  out   UBRRH, r17
  out   UBRRL, r16
  ; Enable receiver and transmitter
  ldi   r16, (1<<RXEN)|(1<<TXEN)
  out   UCSRB,r16
  ; Set frame format: 8data, 2stop bit
  ldi   r16, (1<<URSEL)|(1<<USBS)|(3<<UCSZ0)
  out   UCSRC,r16
  ret
```

C Code Example[1]

```
#define FOSC 1843200// Clock Speed
#define BAUD 9600
#define MYUBRR FOSC/16/BAUD-1
void main( void )
{
  ...
  USART_Init ( MYUBRR );
  ...
}
void USART_Init( unsigned int ubrr)
{
  /* Set baud rate */
  UBRRH = (unsigned char)(ubrr>>8);
  UBRRL = (unsigned char)ubrr;
  /* Enable receiver and transmitter */
  UCSRB = (1<<RXEN)|(1<<TXEN);
  /* Set frame format: 8data, 2stop bit */
  UCSRC = (1<<URSEL)|(1<<USBS)|(3<<UCSZ0);
}
```

Note: 1. See "About Code Examples" on page 8.

More advanced initialization routines can be made that include frame format as parameters, disable interrupts and so on. However, many applications use a fixed setting of the Baud and Control Registers, and for these types of applications the initialization code can be placed directly in the main routine, or be combined with initialization code for other I/O modules.

IDLE No transfers on the communication line (RxD or TxD). An IDLE line must be high.

The frame format used by the USART is set by the UCSZ2:0, UPM1:0 and USBS bits in UCSRB and UCSRC. The Receiver and Transmitter use the same setting. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the Receiver and Transmitter.

The USART Character SiZe (UCSZ2:0) bits select the number of data bits in the frame. The USART Parity mode (UPM1:0) bits enable and set the type of parity bit. The selection between one or two stop bits is done by the USART Stop Bit Select (USBS) bit. The Receiver ignores the second stop bit. An FE (Frame Error) will therefore only be detected in the cases where the first stop bit is zero.

Parity Bit Calculation

The parity bit is calculated by doing an exclusive-or of all the data bits. If odd parity is used, the result of the exclusive or is inverted. The relation between the parity bit and data bits is as follows:

$$P_{even} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

P_{even} Parity bit using even parity.

P_{odd} Parity bit using odd parity.

d_n Data bit n of the character.

If used, the parity bit is located between the last data bit and first stop bit of a serial frame.

USART Initialization

The USART has to be initialized before any communication can take place. The initialization process normally consists of setting the baud rate, setting frame format and enabling the Transmitter or the Receiver depending on the usage. For interrupt driven USART operation, the Global Interrupt Flag should be cleared (and interrupts globally disabled) when doing the initialization.

Before doing a re-initialization with changed baud rate or frame format, be sure that there are no ongoing transmissions during the period the registers are changed. The TXC Flag can be used to check that the Transmitter has completed all transfers, and the RXC Flag can be used to check that there are no unread data in the receive buffer. Note that the TXC Flag must be cleared before each transmission (before UDR is written) if it is used for this purpose.

The following simple USART initialization code examples show one assembly and one C function that are equal in functionality. The examples assume asynchronous operation using polling (no interrupts enabled) and a fixed frame format. The baud rate is given as a function parameter. For the assembly code, the baud rate parameter is assumed to be stored in the r17:r16 Registers. When the function writes to the UCSRC Register, the URSEL bit (MSB) must be set due to the sharing of I/O location by UBRRH and UCSRC.

Data Reception – The USART Receiver

The USART Receiver is enabled by writing the Receive Enable (RXEN) bit in the UCSRB Register to one. When the Receiver is enabled, the normal pin operation of the RxD pin is overridden by the USART and given the function as the Receiver's serial input. The baud rate, mode of operation and frame format must be set up once before any serial reception can be done. If synchronous operation is used, the clock on the XCK pin will be used as transfer clock.

Receiving Frames with 5 to 8 Data Bits

The Receiver starts data reception when it detects a valid start bit. Each bit that follows the start bit will be sampled at the baud rate or XCK clock, and shifted into the Receive Shift Register until the first stop bit of a frame is received. A second stop bit will be ignored by the Receiver. When the first stop bit is received (i.e., a complete serial frame is present in the Receive Shift Register), the contents of the Shift Register will be moved into the receive buffer. The receive buffer can then be read by reading the UDR I/O location.

The following code example shows a simple USART receive function based on polling of the Receive Complete (RXC) Flag. When using frames with less than eight bits the most significant bits of the data read from the UDR will be masked to zero. The USART has to be initialized before the function can be used.

Assembly Code Example[1]

```
USART_Receive:
  ; Wait for data to be received
  sbis UCSRA, RXC
  rjmp USART_Receive
  ; Get and return received data from buffer
  in   r16, UDR
  ret
```

C Code Example[1]

```
unsigned char USART_Receive( void )
{
  /* Wait for data to be received */
  while ( !(UCSRA & (1<<RXC)) )
        ;
  /* Get and return received data from buffer */
  return UDR;
}
```

Note: 1. See "About Code Examples" on page 8.

The function simply waits for data to be present in the receive buffer by checking the RXC Flag, before reading the buffer and returning the value.

Receiving Frames with 9 Data Bits

If 9-bit characters are used (UCSZ=7) the ninth bit must be read from the RXB8 bit in UCSRB **before** reading the low bits from the UDR. This rule applies to the FE, DOR and PE Status Flags as well. Read status from UCSRA, then data from UDR. Reading the UDR I/O location will change the state of the receive buffer FIFO and consequently the TXB8, FE, DOR, and PE bits, which all are stored in the FIFO, will change.

Receive Compete Flag and Interrupt

The USART Receiver has one flag that indicates the Receiver state.

The Receive Complete (RXC) Flag indicates if there are unread data present in the receive buffer. This flag is one when unread data exist in the receive buffer, and zero when the receive buffer is empty (i.e., does not contain any unread data). If the Receiver is disabled (RXEN = 0), the receive buffer will be flushed and consequently the RXC bit will become zero.

When the Receive Complete Interrupt Enable (RXCIE) in UCSRB is set, the USART Receive Complete Interrupt will be executed as long as the RXC Flag is set (provided that global interrupts are enabled). When interrupt-driven data reception is used, the receive complete routine must read the received data from UDR in order to clear the RXC Flag, otherwise a new interrupt will occur once the interrupt routine terminates.

Receiver Error Flags

The USART Receiver has three error flags: Frame Error (FE), Data OverRun (DOR) and Parity Error (PE). All can be accessed by reading UCSRA. Common for the error flags is that they are located in the receive buffer together with the frame for which they indicate the error status. Due to the buffering of the error flags, the UCSRA must be read before the receive buffer (UDR), since reading the UDR I/O location changes the buffer read location. Another equality for the error flags is that they can not be altered by software doing a write to the flag location. However, all flags must be set to zero when the UCSRA is written for upward compatibility of future USART implementations. None of the error flags can generate interrupts.

The Frame Error (FE) Flag indicates the state of the first stop bit of the next readable frame stored in the receive buffer. The FE Flag is zero when the stop bit was correctly read (as one), and the FE Flag will be one when the stop bit was incorrect (zero). This flag can be used for detecting out-of-sync conditions, detecting break conditions and protocol handling. The FE Flag is not affected by the setting of the USBS bit in UCSRC since the Receiver ignores all, except for the first, stop bits. For compatibility with future devices, always set this bit to zero when writing to UCSRA.

The Data OverRun (DOR) Flag indicates data loss due to a Receiver buffer full condition. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the Receive Shift Register, and a new start bit is detected. If the DOR Flag is set there was one or more serial frame lost between the frame last read from UDR, and the next frame read from UDR. For compatibility with future devices, always write this bit to zero when writing to UCSRA. The DOR Flag is cleared when the frame received was successfully moved from the Shift Register to the receive buffer.

The Parity Error (PE) Flag indicates that the next frame in the receive buffer had a parity error when received. If parity check is not enabled the PE bit will always be read zero. For compatibility with future devices, always set this bit to zero when writing to UCSRA. For more details see "Parity Bit Calculation" on page 138 and "Parity Checker" on page 147.

The following code example shows a simple USART receive function that handles both 9-bit characters and the status bits.

Assembly Code Example[1]

```
USART_Receive:
  ; Wait for data to be received
  sbis UCSRA, RXC
  rjmp USART_Receive
  ; Get status and ninth bit, then data from buffer
  in   r18, UCSRA
  in   r17, UCSRB
  in   r16, UDR
  ; If error, return -1
  andi r18,(1<<FE)|(1<<DOR)|(1<<PE)
  breq USART_ReceiveNoError
  ldi  r17, HIGH(-1)
  ldi  r16, LOW(-1)
USART_ReceiveNoError:
  ; Filter the ninth bit, then return
  lsr  r17
  andi r17, 0x01
  ret
```

C Code Example[1]

```
unsigned int USART_Receive( void )
{
  unsigned char status, resh, resl;
  /* Wait for data to be received */
  while ( !(UCSRA & (1<<RXC)) )
        ;
  /* Get status and ninth bit, then data */
  /* from buffer */
  status = UCSRA;
  resh = UCSRB;
  resl = UDR;
  /* If error, return -1 */
  if ( status & (1<<FE)|(1<<DOR)|(1<<PE) )
    return -1;
  /* Filter the ninth bit, then return */
  resh = (resh >> 1) & 0x01;
  return ((resh << 8) | resl);
}
```

Note: 1. See "About Code Examples" on page 8.

The receive function example reads all the I/O Registers into the Register File before any computation is done. This gives an optimal receive buffer utilization since the buffer location read will be free to accept new data as early as possible.

Parity Checker

The Parity Checker is active when the high USART Parity mode (UPM1) bit is set. Type of parity check to be performed (odd or even) is selected by the UPM0 bit. When enabled, the Parity Checker calculates the parity of the data bits in incoming frames and compares the result with the parity bit from the serial frame. The result of the check is stored in the receive buffer together with the received data and stop bits. The Parity Error (PE) Flag can then be read by software to check if the frame had a parity error.

The PE bit is set if the next character that can be read from the receive buffer had a parity error when received and the parity checking was enabled at that point (UPM1 = 1). This bit is valid until the receive buffer (UDR) is read.

Disabling the Receiver

In contrast to the Transmitter, disabling of the Receiver will be immediate. Data from ongoing receptions will therefore be lost. When disabled (i.e., the RXEN is set to zero) the Receiver will no longer override the normal function of the RxD port pin. The Receiver buffer FIFO will be flushed when the Receiver is disabled. Remaining data in the buffer will be lost

Flushing the Receive Buffer

The Receiver buffer FIFO will be flushed when the Receiver is disabled (i.e., the buffer will be emptied of its contents). Unread data will be lost. If the buffer has to be flushed during normal operation, due to for instance an error condition, read the UDR I/O location until the RXC Flag is cleared. The following code example shows how to flush the receive buffer.

Assembly Code Example[1]

```
USART_Flush:
  sbis UCSRA, RXC
  ret
  in   r16, UDR
  rjmp USART_Flush
```

C Code Example[1]

```
void USART_Flush( void )
{
  unsigned char dummy;
  while ( UCSRA & (1<<RXC) ) dummy = UDR;
}
```

Note: 1. See "About Code Examples" on page 8.

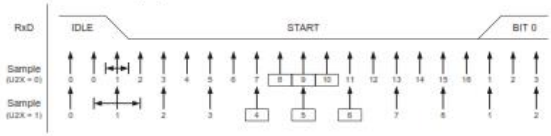
Asynchronous Data Reception

The USART includes a clock recovery and a data recovery unit for handling asynchronous data reception. The clock recovery logic is used for synchronizing the internally generated baud rate clock to the incoming asynchronous serial frames at the RxD pin. The data recovery logic samples and low pass filters each incoming bit, thereby improving the noise immunity of the Receiver. The asynchronous reception operational range depends on the accuracy of the internal baud rate clock, the rate of the incoming frames, and the frame size in number of bits.

Asynchronous Clock Recovery

The clock recovery logic synchronizes internal clock to the incoming serial frames. Figure 65 illustrates the sampling process of the start bit of an incoming frame. The sample rate is 16 times the baud rate for Normal mode, and eight times the baud rate for Double Speed mode. The horizontal arrows illustrate the synchronization variation due to the sampling process. Note the larger time variation when using the Double Speed mode (U2X = 1) of operation. Samples denoted zero are samples done when the RxD line is idle (i.e., no communication activity).

Figure 65. Start Bit Sampling

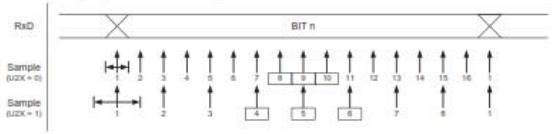


When the clock recovery logic detects a high (idle) to low (start) transition on the RxD line, the start bit detection sequence is initiated. Let sample 1 denote the first zero-sample as shown in the figure. The clock recovery logic then uses samples 8, 9 and 10 for Normal mode, and samples 4, 5 and 6 for Double Speed mode (indicated with sample numbers inside boxes on the figure), to decide if a valid start bit is received. If two or more of these three samples have logical high levels (the majority wins), the start bit is rejected as a noise spike and the Receiver starts looking for the next high to low-transition. If however, a valid start bit is detected, the clock recovery logic is synchronized and the data recovery can begin. The synchronization process is repeated for each start bit.

Asynchronous Data Recovery

When the Receiver clock is synchronized to the start bit, the data recovery can begin. The data recovery unit uses a state machine that has 16 states for each bit in Normal mode and eight states for each bit in Double Speed mode. Figure 66 shows the sampling of the data bits and the parity bit. Each of the samples is given a number that is equal to the state of the recovery unit.

Figure 66. Sampling of Data and Parity Bit



The decision of the logic level of the received bit is taken by doing a majority voting of the logic value to the three samples in the center of the received bit. The center samples are emphasized on the figure by having the sample number inside boxes. The majority voting process is done as follows: If two or all three samples have high levels, the received bit is registered to be a logic 1. If two or all three samples have low levels, the received bit is registered to be a logic 0. This majority voting process acts as a low pass filter for the incoming signal on the RxD pin. The recovery process is then repeated until a complete frame is received. Including the first stop bit. Note that the Receiver only uses the first stop bit of a frame.

Figure 67 shows the sampling of the stop bit and the earliest possible beginning of the start bit of the next frame.

Table 53. Recommended Maximum Receiver Baud Rate Error for Normal Speed Mode (U2X = 0)

D# (Data+Parity Bit)	R_{slow}(%)	R_{fast}(%)	Max Total Error (%)	Recommended Max Receiver Error (%)
5	93,20	106,67	+6.67/-6.8	± 3.0
6	94,12	105,79	+5.79/-5.88	± 2.0
7	94,81	105,11	+5.11/-5.19	± 2.0
8	95,36	104,58	+4.58/-4.54	± 2.0
9	95,81	104,14	+4.14/-4.19	± 1.5
10	96,17	103,78	+3.78/-3.83	± 1.5

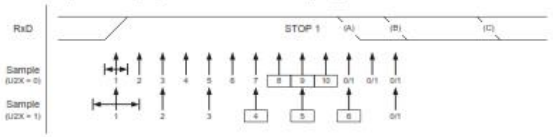
Table 54. Recommended Maximum Receiver Baud Rate Error for Double Speed Mode (U2X = 1)

D# (Data+Parity Bit)	R_{slow}(%)	R_{fast}(%)	Max Total Error (%)	Recommended Max Receiver Error (%)
5	94,12	105,66	+5.66/-5.88	± 2.5
6	94,92	104,92	+4.92/-5.08	± 2.0
7	95,52	104,35	+4.35/-4.48	± 1.5
8	96,00	103,90	+3.90/-4.00	± 1.5
9	96,39	103,53	+3.53/-3.61	± 1.5
10	96,70	103,23	+3.23/-3.30	± 1.0

The recommendations of the maximum Receiver baud rate error was made under the assumption that the Receiver and Transmitter equally divides the maximum total error.

There are two possible sources for the Receivers Baud Rate error. The Receiver's system clock (XTAL) will always have some minor instability over the supply voltage range and the temperature range. When using a crystal to generate the system clock, this is rarely a problem, but for a resonator the system clock may differ more than 2% depending of the resonators tolerance. The second source for the error is more controllable. The baud rate generator can not always do an exact division of the system frequency to get the baud rate wanted. In this case an UBRR value that gives an acceptable low error can be used if possible.

Figure 67. Stop Bit Sampling and Next Start Bit Sampling



The same majority voting is done to the stop bit as done for the other bits in the frame. If the stop bit is registered to have a logic 0 value, the Frame Error (FE) Flag will be set.

A new high to low transition indicating the start bit of a new frame can come right after the last of the bits used for majority voting. For Normal Speed mode, the first low level sample can be at point marked (A) in Figure 67. For Double Speed mode the first low level must be delayed to (B). (C) marks a stop bit of full length. The early start bit detection influences the operational range of the Receiver.

Asynchronous Operational Range

The operational range of the Receiver is dependent on the mismatch between the received bit rate and the internally generated baud rate. If the Transmitter is sending frames at too fast or too slow bit rates, or the internally generated baud rate of the Receiver does not have a similar (see Table 53) base frequency, the Receiver will not be able to synchronize the frames to the start bit.

The following equations can be used to calculate the ratio of the incoming data rate and internal Receiver baud rate.

$$R_{slow} = \frac{(D+1)S}{S-1+D\cdot S+S_F} \qquad R_{fast} = \frac{(D+2)S}{(D+1)S+S_M}$$

D Sum of character size and parity size (D = 5- to 10-bit)

S Samples per bit. S = 16 for Normal Speed mode and S = 8 for Double Speed mode.

S_F First sample number used for majority voting. S_F = 8 for Normal Speed and S_F = 4 for Double Speed mode.

S_M Middle sample number used for majority voting. S_M = 9 for Normal Speed and S_M = 5 for Double Speed mode.

R_{slow} is the ratio of the slowest incoming data rate that can be accepted in relation to the Receiver baud rate. R_{fast} is the ratio of the fastest incoming data rate that can be accepted in relation to the Receiver baud rate.

Table 53 and Table 54 list the maximum Receiver baud rate error that can be tolerated. Note that Normal Speed mode has higher toleration of baud rate variations.

Accessing UBRRH/UCSRC Registers

The UBRRH Register shares the same I/O location as the UCSRC Register. Therefore some special consideration must be taken when accessing this I/O location.

Write Access

When doing a write access of this I/O location, the high bit of the value written, the USART Register Select (URSEL) bit, controls which one of the two registers that will be written. If URSEL is zero during a write operation, the UBRRH value will be updated. If URSEL is one, the UCSRC setting will be updated.

The following code examples show how to access the two registers.

Assembly Code Examples[1]

```
...
; Set UBRRH to 2
ldi r16,0x02
out UBRRH,r16
...
; Set the USBS and the UCSZ1 bit to one, and
; the remaining bits to zero.
ldi r16,(1<<URSEL)|(1<<USBS)|(1<<UCSZ1)
out UCSRC,r16
...
```

C Code Examples[1]

```
...
/* Set UBRRH to 2 */
UBRRH = 0x02;
...
/* Set the USBS and the UCSZ1 bit to one, and */
/* the remaining bits to zero. */
UCSRC = (1<<URSEL)|(1<<USBS)|(1<<UCSZ1);
...
```

Note: 1. See "About Code Examples" on page 8.

As the code examples illustrate, write accesses of the two registers are relatively unaffected of the sharing of I/O location.

Read Access

Doing a read access to the UBRRH or the UCSRC Register is a more complex operation. However, in most applications, it is rarely necessary to read any of these registers.

The read access is controlled by a timed sequence. Reading the I/O location once returns the UBRRH Register contents. If the register location was read in previous system clock cycle, reading the register in the current clock cycle will return the UCSRC contents. Note that the timed sequence for reading the UCSRC is an atomic operation. Interrupts must therefore be controlled (e.g., by disabling interrupts globally) during the read operation.

The following code example shows how to read the UCSRC Register contents.

Assembly Code Example[1]

```
USART_ReadUCSRC:
  ; Read UCSRC
  in r16,UBRRH
  in r16,UCSRC
  ret
```

C Code Example[1]

```
unsigned char USART_ReadUCSRC( void )
{
  unsigned char ucsrc;
  /* Read UCSRC */
  ucsrc = UBRRH;
  ucsrc = UCSRC;
  return ucsrc;
}
```

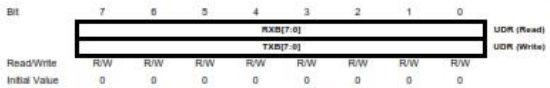
Note: 1. See "About Code Examples" on page 8.

The assembly code example returns the UCSRC value in r16.

Reading the UBRRH contents is not an atomic operation and therefore it can be read as an ordinary register, as long as the previous instruction did not access the register location.

USART Register Description

USART I/O Data Register – UDR



The USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or UDR. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDR Register location. Reading the UDR Register location will return the contents of the Receive Data Buffer Register (RXB).

For 5-, 6-, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver.

The transmit buffer can only be written when the UDRE Flag in the UCSRA Register is set. Data written to UDR when the UDRE Flag is not set, will be ignored by the USART Transmitter. When

Writing this bit to one will reduce the divisor of the baud rate divider from 16 to 8 effectively doubling the transfer rate for asynchronous communication.

- **Bit 0 – MPCM: Multi-processor Communication Mode**

This bit enables the Multi-processor Communication mode. When the MPCM bit is written to one, all the incoming frames received by the USART Receiver that do not contain address information will be ignored. The Transmitter is unaffected by the MPCM setting. For more detailed information see "Multi-processor Communication Mode" on page 151.

USART Control and Status Register B – UCSRB

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – RXCIE: RX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the RXC Flag. A USART Receive Complete interrupt will be generated only if the RXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXC bit in UCSRA is set.

- **Bit 6 – TXCIE: TX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the TXC Flag. A USART Transmit Complete interrupt will be generated only if the TXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXC bit in UCSRA is set.

- **Bit 5 – UDRIE: USART Data Register Empty Interrupt Enable**

Writing this bit to one enables interrupt on the UDRE Flag. A Data Register Empty interrupt will be generated only if the UDRIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDRE bit in UCSRA is set.

- **Bit 4 – RXEN: Receiver Enable**

Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxD pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FE, DOR and PE Flags.

- **Bit 3 – TXEN: Transmitter Enable**

Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxD pin when enabled. The disabling of the Transmitter (writing TXEN to zero) will not become effective until ongoing and pending transmissions are completed (i.e., when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted). When disabled, the Transmitter will no longer override the TxD port.

- **Bit 2 – UCSZ2: Character Size**

The UCSZ2 bits combined with the UCSZ1:0 bit in UCSRC sets the number of data bits (Character Size) in a frame the Receiver and Transmitter use.

- **Bit 1 – RXB8: Receive Data Bit 8**

RXB8 is the ninth data bit of the received character when operating with serial frames with nine data bits. Must be read before reading the low bits from UDR.

- **Bit 0 – TXB8: Transmit Data Bit 8**

TXB8 is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. Must be written before writing the low bits to UDR.

data is written to the transmit buffer, and the Transmitter is enabled, the Transmitter will load the data into the Transmit Shift Register when the Shift Register is empty. Then the data will be serially transmitted on the TxD pin.

The receive buffer consists of a two level FIFO. The FIFO will change its state whenever the receive buffer is accessed. Due to this behavior of the receive buffer, do not use Read-Modify-Write instructions (SBI and CBI) on this location. Be careful when using bit test instructions (SBIC and SBIS), since these also will change the state of the FIFO.

USART Control and Status Register A – UCSRA

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- **Bit 7 – RXC: USART Receive Complete**

This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e. does not contain any unread data). If the Receiver is disabled, the receive buffer will be flushed and consequently the RXC bit will become zero. The RXC Flag can be used to generate a Receive Complete interrupt (see description of the RXCIE bit).

- **Bit 6 – TXC: USART Transmit Complete**

This flag bit is set when the entire frame in the Transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer (UDR). The TXC Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXC Flag can generate a Transmit Complete interrupt (see description of the TXCIE bit).

- **Bit 5 – UDRE: USART Data Register Empty**

The UDRE Flag indicates if the transmit buffer (UDR) is ready to receive new data. If UDRE is one, the buffer is empty, and therefore ready to be written. The UDRE Flag can generate a Data Register Empty interrupt (see description of the UDRIE bit).

UDRE is set after a reset to indicate that the Transmitter is ready.

- **Bit 4 – FE: Frame Error**

This bit is set if the next character in the receive buffer had a Frame Error when received (i.e., when the first stop bit of the next character in the receive buffer is zero). This bit is valid until the receive buffer (UDR) is read. The FE bit is zero when the stop bit of received data is one. Always set this bit to zero when writing to UCSRA.

- **Bit 3 – DOR: Data OverRun**

This bit is set if a Data OverRun condition is detected. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the Receive Shift Register, and a new start bit is detected. This bit is valid until the receive buffer (UDR) is read. Always set this bit to zero when writing to UCSRA.

- **Bit 2 – PE: Parity Error**

This bit is set if the next character in the receive buffer had a Parity Error when received and the parity checking was enabled at that point (UPM1 = 1). This bit is valid until the receive buffer (UDR) is read. Always set this bit to zero when writing to UCSRA.

- **Bit 1 – U2X: Double the USART transmission speed**

This bit only has effect for the asynchronous operation. Write this bit to zero when using synchronous operation.

USART Control and Status Register C – UCSRC

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

The UCSRC Register shares the same I/O location as the UBRRH Register. See the "Accessing UBRRH/UCSRC Registers" on page 152 section which describes how to access this register.

- **Bit 7 – URSEL: Register Select**

This bit selects between accessing the UCSRC or the UBRRH Register. It is read as one when reading UCSRC. The URSEL must be one when writing the UCSRC.

- **Bit 6 – UMSEL: USART Mode Select**

This bit selects between Asynchronous and Synchronous mode of operation.

Table 55. UMSEL Bit Settings

UMSEL	Mode
0	Asynchronous Operation
1	Synchronous Operation

- **Bit 5:4 – UPM1:0: Parity Mode**

These bits enable and set type of Parity Generation and Check. If enabled, the Transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The Receiver will generate a parity value for the incoming data and compare it to the UPM0 setting. If a mismatch is detected, the PE Flag in UCSRA will be set.

Table 56. UPM Bits Settings

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

- **Bit 3 – USBS: Stop Bit Select**

This bit selects the number of stop bits to be inserted by the trAnsmitter. The Receiver ignores this setting.

Table 57. USBS Bit Settings

USBS	Stop Bit(s)
0	1-bit
1	2-bit

- **Bit 2:1 – UCSZ1:0: Character Size**

The UCSZ1:0 bits combined with the UCSZ2 bit in UCSRB sets the number of data bits (Character Size) in a frame the Receiver and Transmitter use.

Table 58. UCSZ Bits Settings

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

- **Bit 0 – UCPOL: Clock Polarity**

Examples of Baud Rate Setting

For standard crystal and resonator frequencies, the most commonly used baud rates for asynchronous operation can be generated by using the UBRR settings in Table 60. UBRR values which yield an actual baud rate differing less than 0.5% from the target baud rate, are bold in the table. Higher error ratings are acceptable, but the Receiver will have less noise resistance when the error ratings are high, especially for large serial frames (see "Asynchronous Operational Range" on page 149). The error values are calculated using the following equation:

$$\text{Error}[\%] = \left(\frac{\text{BaudRate}_{\text{Closest Match}}}{\text{BaudRate}} - 1\right) \cdot 100\%$$

Table 60. Examples of UBRR Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	f_{osc} = 1.0000 MHz				f_{osc} = 1.8432 MHz				f_{osc} = 2.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	**25**	**0.2%**	**51**	**0.2%**	**47**	**0.0%**	**95**	**0.0%**	**51**	**0.2%**	**103**	**0.2%**
4800	**12**	**0.2%**	**25**	**0.2%**	**23**	**0.0%**	**47**	**0.0%**	**25**	**0.2%**	**51**	**0.2%**
9600	6	-7.0%	**12**	**0.2%**	**11**	**0.0%**	**23**	**0.0%**	**12**	**0.2%**	**25**	**0.2%**
14.4k	3	8.5%	8	-3.5%	**7**	**0.0%**	**15**	**0.0%**	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	**5**	**0.0%**	**11**	**0.0%**	6	-7.0%	**12**	**0.2%**
28.8k	1	8.5%	3	8.5%	**3**	**0.0%**	**7**	**0.0%**	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	**2**	**0.0%**	**5**	**0.0%**	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	**1**	**0.0%**	**3**	**0.0%**	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	**2**	**0.0%**	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	**0**	**0.0%**	**1**	**0.0%**	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	**0**	**0.0%**	–	-	–	–
250k	–	–	–	–	–	–	–	–	–	–	**0**	**0.0%**
Max (1)	62.5 kbps		125 kbps		115.2 kbps		230.4 kbps		125 kbps		250 kbps	

1. UBRR = 0, Error = 0.0%

This bit is used for Synchronous mode only. Write this bit to zero when Asynchronous mode is used. The UCPOL bit sets the relationship between data output change and data input sample, and the synchronous clock (XCK).

Table 59. UCPOL Bit Settings

UCPOL	Transmitted Data Changed (Output of TxD Pin)	Received Data Sampled (Input on RxD Pin)
0	Rising XCK Edge	Falling XCK Edge
1	Falling XCK Edge	Rising XCK Edge

USART Baud Rate Registers – UBRRL and UBRRHs

Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	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	
	0	0	0	0	0	0	0	0	

The UBRRH Register shares the same I/O location as the UCSRC Register. See the "Accessing UBRRH/UCSRC Registers" on page 152 section which describes how to access this register.

- **Bit 15 – URSEL: Register Select**

This bit selects between accessing the UBRRH or the UCSRC Register. It is read as zero when reading UBRRH. The URSEL must be zero when writing the UBRRH.

- **Bit 14:12 – Reserved Bits**

These bits are reserved for future use. For compatibility with future devices, these bit must be written to zero when UBRRH is written.

- **Bit 11:0 – UBRR11:0: USART Baud Rate Register**

This is a 12-bit register which contains the USART baud rate. The UBRRH contains the four most significant bits, and the UBRRL contains the eight least significant bits of the USART baud rate. Ongoing transmissions by the Transmitter and Receiver will be corrupted if the baud rate is changed. Writing UBRRL will trigger an immediate update of the baud rate prescaler.

Table 61. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 3.6864 MHz				f_{osc} = 4.0000 MHz				f_{osc} = 7.3728 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	95	0.0%	191	0.0%	103	0.2%	207	0.2%	191	0.0%	383	0.0%
4800	47	0.0%	95	0.0%	51	0.2%	103	0.2%	95	0.0%	191	0.0%
9600	23	0.0%	47	0.0%	25	0.2%	51	0.2%	47	0.0%	95	0.0%
14.4k	15	0.0%	31	0.0%	16	2.1%	34	-0.8%	31	0.0%	63	0.0%
19.2k	11	0.0%	23	0.0%	12	0.2%	25	0.2%	23	0.0%	47	0.0%
28.8k	7	0.0%	15	0.0%	8	-3.5%	16	2.1%	15	0.0%	31	0.0%
38.4k	5	0.0%	11	0.0%	6	-7.0%	12	0.2%	11	0.0%	23	0.0%
57.6k	3	0.0%	7	0.0%	3	8.5%	8	-3.5%	7	0.0%	15	0.0%
76.8k	2	0.0%	5	0.0%	2	8.5%	6	-7.0%	5	0.0%	11	0.0%
115.2k	1	0.0%	3	0.0%	1	8.5%	3	8.5%	3	0.0%	7	0.0%
230.4k	0	0.0%	1	0.0%	0	8.5%	1	8.5%	1	0.0%	3	0.0%
250k	0	-7.8%	1	-7.8%	0	0.0%	1	0.0%	1	-7.8%	3	-7.8%
0.5M	–	–	0	-7.8%	–	–	0	0.0%	0	-7.8%	1	-7.8%
1M	–	–	–	–	–	–	–	–	–	–	0	-7.8%
Max (1)	230.4 kbps		460.8 kbps		250 kbps		0.5 Mbps		460.8 kbps		921.6 kbps	

1. UBRR = 0, Error = 0.0%

Table 62. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 8.0000 MHz				f_{osc} = 11.0592 MHz				f_{osc} = 14.7456 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	-7.8%	6	5.3%
0.5M	0	0.0%	1	0.0%	–	–	2	-7.8%	1	-7.8%	3	-7.8%
1M	–	–	0	0.0%	–	–	–	–	0	-7.8%	1	-7.8%
Max (1)	0.5 Mbps		1 Mbps		691.2 kbps		1.3824 Mbps		921.6 kbps		1.8432 Mbps	

1. UBRR = 0, Error = 0.0%

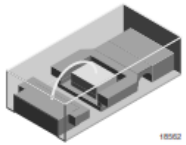
Table 63. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 16.0000 MHz				f_{osc} = 18.4320 MHz				f_{osc} = 20.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	**416**	**-0.1%**	**832**	**0.0%**	**479**	**0.0%**	**959**	**0.0%**	**520**	**0.0%**	**1041**	**0.0%**
4800	**207**	**0.2%**	**416**	**-0.1%**	**239**	**0.0%**	**479**	**0.0%**	**259**	**0.2%**	**520**	**0.0%**
9600	**103**	**0.2%**	**207**	**0.2%**	**119**	**0.0%**	**239**	**0.0%**	**129**	**0.2%**	**259**	**0.2%**
14.4k	68	0.6%	**138**	**-0.1%**	**79**	**0.0%**	**159**	**0.0%**	**86**	**-0.2%**	**173**	**-0.2%**
19.2k	**51**	**0.2%**	**103**	**0.2%**	**59**	**0.0%**	**119**	**0.0%**	**64**	**0.2%**	**129**	**0.2%**
28.8k	34	-0.8%	68	0.6%	**39**	**0.0%**	**79**	**0.0%**	42	0.9%	**86**	**-0.2%**
38.4k	**25**	**0.2%**	**51**	**0.2%**	**29**	**0.0%**	**59**	**0.0%**	32	-1.4%	**64**	**0.2%**
57.6k	16	2.1%	34	-0.8%	**19**	**0.0%**	**39**	**0.0%**	21	-1.4%	42	0.9%
76.8k	**12**	**0.2%**	**25**	**0.2%**	**14**	**0.0%**	**29**	**0.0%**	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	**9**	**0.0%**	**19**	**0.0%**	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	**4**	**0.0%**	**9**	**0.0%**	4	8.5%	10	-1.4%
250k	**3**	**0.0%**	**7**	**0.0%**	4	-7.8%	8	2.4%	**4**	**0.0%**	**9**	**0.0%**
0.5M	**1**	**0.0%**	**3**	**0.0%**	–	–	4	-7.8%	–	–	**4**	**0.0%**
1M	0	0.0%	1	0.0%	–	–	–	–	–	–	–	–
Max (1)	1 Mbps		2 Mbps		1.152 Mbps		2.304 Mbps		1.25 Mbps		2.5 Mbps	

1. UBRR = 0, Error = 0.0%

Lampiran 5. Datasheet LED SMD 0603

Ultrabright 0603 SMD LED



DESCRIPTION

The new 0603 LED series have been designed in the smallest SMD package. This innovative 0603 LED technology opens the way to

- smaller products of higher performance
- more design in flexibility
- enhanced applications

The 0603 LED is an obvious solution for small-scale, high power products that are expected to work reliability in an arduous environment.

The reflector inside this package is filled with a mixture of epoxy and yellow converter.

This yellow converter converts the blue emission partially to yellow, which mixes the remaining blue to give white.

PRODUCT GROUP AND PACKAGE DATA

- Product group: LED
- Package: SMD 0603
- Product series: standard
- Angle of half intensity: ± 80°

FEATURES

- High efficient InGaN technology
- Smallest SMD package 0603 with exceptional brightness 1.6 mm x 0.8 mm x 0.6 mm (L x W x H)
- High reliability lead frame based
- Temperature range -40 °C to +100 °C
- Chromaticity coordinate categorized according to CIE1931 per packing unit
- Typical color temperature 5500 K
- EIA and ICE standard package
- Compatible to IR reflow soldering
- Available in 8 mm tape reel
- Preconditioning according to JEDEC® level 2
- ESD-withstand voltage: up to 1 kV according to JESD22-A114-B
- AEC-Q101 qualified
- Material categorization: for definitions of compliance please see www.vishay.com/doc?99912

APPLICATIONS

- Automotive: backlighting in dashboards, switches, and keypads
- Telecommunication: indicator and backlighting in telephone and fax
- Backlighting for audio, and video equipment
- Backlighting in office equipment
- Indoor and outdoor message boards
- Flat backlight for LCDs, switches, and symbols

PARTS TABLE

PART	COLOR	LUMINOUS INTENSITY (mcd)			at I_F (mA)	COORDINATE (x, y)			at I_F (mA)	FORWARD VOLTAGE (V)			at I_F (mA)	TECHNOLOGY
		MIN.	TYP.	MAX.		MIN.	TYP.	MAX.		MIN.	TYP.	MAX.		
VLMW11R2S2-5K8L-08	White	140	-	280	10	-	0.33, 0.33	-	10	2.9	-	4.0	20	InGaN / yellow converter

ABSOLUTE MAXIMUM RATINGS (T_{amb} = 25 °C, unless otherwise specified)
VLMW11..

PARAMETER	TEST CONDITION	SYMBOL	VALUE	UNIT
Reverse voltage [1]	I_R max. = 10 µA	V_R	5	V
DC forward current	$T_{amb} \leq 60$ °C	I_F	20	mA
Surge forward current	$t_p \leq 10$ µs	I_{FSM}	0.1	A
Power dissipation		P_V	80	mW
Junction temperature		T_j	110	°C
Storage temperature range		T_{stg}	-40 to +100	°C
Operating temperature range		T_{amb}	-40 to +100	°C
Thermal resistance junction/ambient	mounted on PC board (pad size > 16 mm²)	R_{thJA}	480	K/W

Note

[1] Driving the LED in reverse direction is suitable for short term application

OPTICAL AND ELECTRICAL CHARACTERISTICS (T_{amb} = 25 °C, unless otherwise specified)
VLMW11.., WHITE

PARAMETER	TEST CONDITION	PART	SYMBOL	MIN.	TYP.	MAX.	UNIT
Luminous intensity	I_F = 10 mA	VLMW11R2S2	I_V	140	-	280	mcd
Chromaticity coordinate x acc. to CIE 1931	I_F = 10 mA	VLMW11	x	-	0.33	-	
Chromaticity coordinate y acc. to CIE 1931	I_F = 10 mA	VLMW11	y	-	0.33	-	
Angle of half intensity	I_F = 10 mA		φ	-	± 80	-	deg
Forward voltage	I_F = 20 mA		V_F	2.9	-	4.0	V
Temperature coefficient of V_F	I_F = 10 mA		TC_{VF}	-	-3	-	mV/K
Temperature coefficient of I_V	I_F = 10 mA		TC_{IV}	-	-0.4	-	%/K

LUMINOUS INTENSITY CLASSIFICATION

GROUP	LIGHT INTENSITY (mcd)		
STANDARD	OPTIONAL	MIN.	MAX.
R	2	140	180
S	1	180	224
	2	224	280

CROSSING TABLE

VISHAY	OSRAM
VLMW11R2S2	LWL28G-R2S2

Note

- Luminous intensity is tested at a current pulse duration of 25 ms and an accuracy of ± 11 %.
  The above type numbers represent the order groups which include only a few brightness groups. Only one group will be shipped on each reel (there will be no mixing of two groups on each reel). In order to ensure availability, single brightness groups are not be orderable.
  In a similar manner for colors where wavelength groups are measured and binned, single wavelength groups will be shipped on any one reel.
  In order to ensure availability, single wavelength groups are not be orderable.

CHROMATICITY COORDINATED GROUPS FOR WHITE SMD LED

	X	Y		X	Y
5L	0.291	0.268	7L	0.330	0.330
	0.285	0.279		0.330	0.347
	0.307	0.312		0.347	0.371
	0.310	0.297		0.345	0.352
5K	0.296	0.259	7K	0.330	0.310
	0.291	0.268		0.330	0.330
	0.310	0.297		0.338	0.342
	0.313	0.284		0.352	0.344
6L	0.310	0.297	8L	0.345	0.352
	0.307	0.312		0.347	0.371
	0.330	0.347		0.367	0.401
	0.330	0.330		0.364	0.380
6K	0.313	0.284	8K	0.352	0.344
	0.310	0.297		0.338	0.342
	0.330	0.330		0.364	0.380
	0.330	0.310		0.360	0.357

Note

- Chromaticity coordinate groups are tested at a current pulse duration of 25 ms and a tolerance of ± 0.01.

TYPICAL CHARACTERISTICS (T_{amb} = 25 °C, unless otherwise specified)

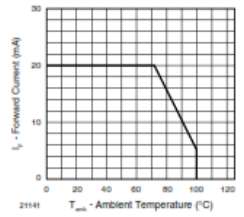


Fig. 1 - Forward Current vs. Ambient Temperature

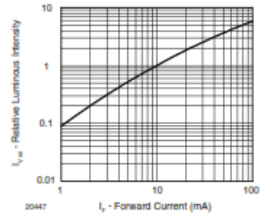


Fig. 4 - Relative Luminous Intensity vs. Forward Current

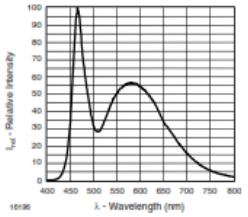


Fig. 2 - Relative Intensity vs. Wavelength

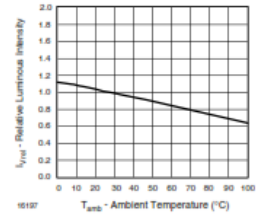


Fig. 5 - Relative Luminous Intensity vs. Ambient Temperature

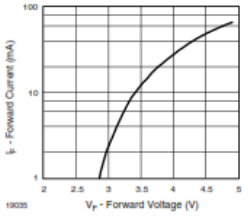


Fig. 3 - Forward Current vs. Forward Voltage

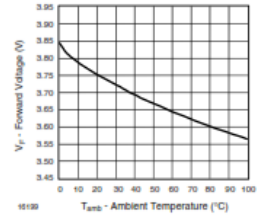


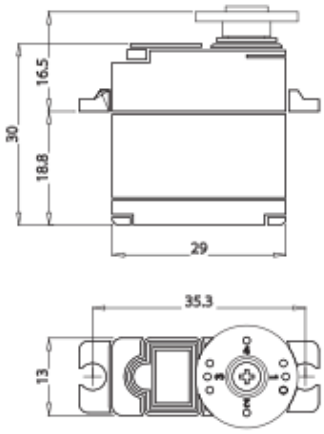
Fig. 6 - Forward Voltage vs. Ambient Temperature

Lampiran 6. Datasheet Motor DC *Servo* HS5085

PREPARED BY JUN HEE, LEE
BEGIN DATE: OCT 01, 2007
UPDATA:JAN 03, 2008

GENERAL SPECIFICATION OF HS-5085MG DIGITAL MICRO SERVO

1.TECHNICAL VALUE		
CONTROL SYSTEM	:+PULSE WIDTH CONTROL 1500usec NEUTRAL	
OPERATING VOLTAGE RANGE	:4.8V TO 6.0V	
OPERATING TEMPERATURE RANGE	:-20° TO +60°(-4°F TO +140°F)	
TEST VOLTAGE	:AT 4.8V	AT 6.0V
OPERATING SPEED	:0.17sec/60° AT NO LOAD	0.13sec/60° AT NO LOAD
STALL TORQUE	:3.6kg.cm(50.0oz.in)	4.3kg.cm(59.71oz.in)
STANDING TORQUE	:-	-
IDLE CURRENT	:3mA AT STOPPED	3mA AT STOPPED
RUNNING CURRENT	:230mA/60° AT NO LOAD RUNNING	290mA/60° AT NO LOAD RUNNING
STALL CURRENT	:1,700mA	2,150mA
DEAD BAND WIDTH	:2usec	2usec
OPERATING TRAVEL	:40°/ONE SIDE PULSE TRAVELING 400usec	
DIRECTION	:CLOCK WISE/PULSE TRAVELING 1500 TO 1900usec	
MOTOR TYPE	:CORED METAL BRUSH	
POTENTIOMETER TYPE	:2 SLIDER/DIRECT DRIVE	
AMPLIFIER TYPE	:DIGITAL AMPLIFIER & MOSFET DRIVER	
DIMENSIONS	:29x13x30mm(1.14x0.51x1.18in)	
WEIGHT	:21.9g(0.77oz)	
BALL BEARING	:SINGLE/MR106	
GEAR MATERIAL	:1 RESIN & 4 METAL	
HORN GEAR SPLINE	:24 SEGMENTS/Ø5.76	
SPLINED HORNS	:MICRO/M-I, M-O, M-X	
CONNECTOR WIRE LENGTH	:250mm(9.84in)	
CONNECTOR WIRE STRAND COUNTER	:20EA	
CONNECTOR WIRE GAUGE	:28AWG	



2.FEATURES
- HIGH PERFORMANCE DIGITAL AMPLIFIER
- LIGHTWEIGHT METAL GEARS WITH BALL BEARING

3.APPLICATIONS
- FOR SMALL MODELS

HITEC RCD KOREA INC.