Hardware ve Software UART Iletisim Arayüzleri

/*
 * Mehmet Ozgan <mozgan @ gmail.com>
 * 20.08.2011, Wien
 */

Index

0. Baslarken

Bu dökümanda Atmel AVR 8-bitlik mikro islemcilerde UART iletisim arayüzünün nasil oldugunu görecegiz. Ancak islenecek olan adimlar algoritmik oldugundan farkli mikro iclemci, pic v.b. kullanilacagi vakit o islemciye -eger var ise- özel register ayarlari eklenmelidir.

1. UART (Universal Asynchronous Receiver Transmitter)

UART, PC ve μC arasindaki seri iletisim arayüzüdür. Bu iletisim 5 ile 9 bit arasi data uzunlugu tasima özelligine sahiptir. Ancak genellikle kullanim sirasinda 8 veya 9 bit tercih edilir.

2. USART (Universal Synchronous/Asynchronous Receive Transmitter)

Yeni tip mikro islemcilerde bulunan, daha gelismis bir iletisim arayüzüdür ve ek olarak senkronize edilmis veri tasima özelligi sunmaktadir.

3. U(S)ART Genel

U(S)ART iletisim arayüzü kullanilirken öncelikle veri tasima hizi (BAUD) ayarlanir. Örnegin, iletisim hizi 9600 BAUD olan bir seri iletisim arayüzü saniyede 960 Byte göndermektedir (eger 1 adet Stop-Bit göze alinirsa).

Bir byte uzunlugunda veri icin 1 Start-Bit, 8 Data-Biti ve 1-2 adet Stop-Bit gönderilir. Genellikle 2. Stop-Bit Verici'in Alici'dan daha hizli oldugu durumlardaki UART uygulamalarinda kullanilir. UART Data formati genelde su sekilde yazilmaktadir:
- 8N1 : 8 Data-Bit, No Parity, 1 Stop-Bit
- 9E1 : 9 Data-Bit, Even Parity, 1 Stop-Bit
- 9O2 : 9 Data-Bit, Odd Parity, 2 Stop-Bit
- ...

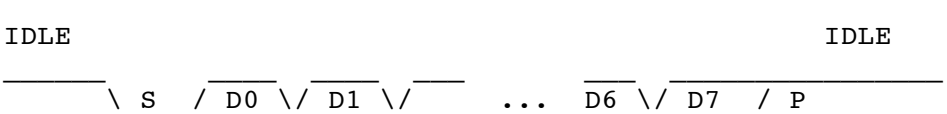
Asenkron veri aktariminin hatasiz (ya da en az hata ile) calisabilmesi icin Verici ve Alici'nin iletisim hizlari ayni (hic degilse cok yakin) olarak ayarlanmalidir. Örnegin, 10 bitlik bir veri paketi en fazla %4'lük hata payina kadar cikabilmektedir. Güvenli bir data aktarimi icin %2'lik hata payina kadar

göz ardi edilebilir. Tabii bu gecikme zamaninda olusan hata payi icerisinde yedekleme süresi, calistirma ve durdurma süreleri v.b. de barinmaktadir.

Kullanim alanlari:
- PC ile iletisim
- Robot ici alt sistemler arasi iletisim
- Bootloader
- SPI iletisim arayüzü
- Wireless, ethernet iletisimi
- ...

4. U(S)ART Iletisimi (Teori)

Asenkron bir seri iletisiminde bir byte uzunlugunda veri gönderilirken asagidaki gibi bir yapi görülmektedir:



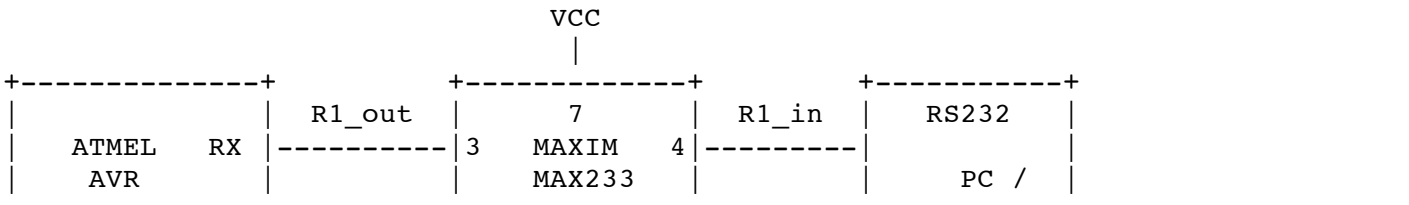
S ... Start-Bit
D0:7 ... Data Bitleri
P ... Stop-Bit

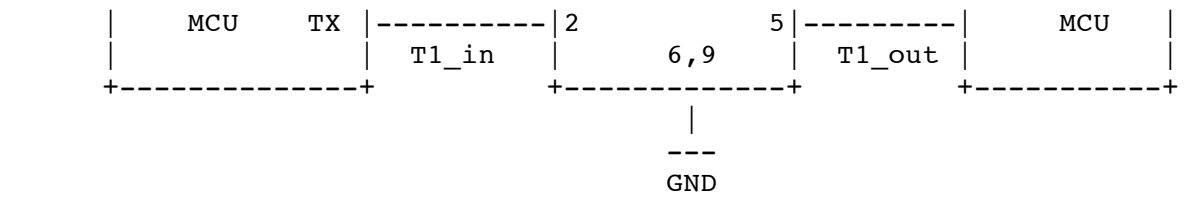
Bu durmda Verici'in TX (Transmit) pini Alici'nin RX (Receive) pini ile devamli olarak iletisim halindedir. Ve sinyal iletisimin IDLE durumunda iken HIGH konumunda bulunur. Eger veri aktarimi basliyor ise, Verici Start-Bit olarak sinyalin konumunu LOW yapar ve Alici bunu farkettiginde dinlemeye gecer. Eger hardware uart kullaniliyor ise tüm islemler islemci icerisinde otomatik olarak gerceklesir. LOW sinyalini algilayan Alici, gelen veri icin kullanacagi uart veri kütügüne (UART Data Register) sinyalin durumuna göre (LOW veya HIGH) 0 veya 1 bitlerini kaydeder. Eger iletisim sirasinda herhangi bir hata olusmus ise Stop-Bit Gönderici tarafindan gönderilemeyecegi gibi Alici tarafindan da alinamayacaktir. Bu durumda Alici'nin kilitlenme ihtimali olacagindan, ya disaridan bir buton ile islemci resetlenir, veya iceriden bir kesme ile islem sonlandirilir. Fakat Software UART ile bir seri iletisim kurulmus ise, veri aktarimi icin ayrilmis olan süre doldugunda zamanlayici (TIMER) kesme (Interrupt) bayragini kaldirir ve islemci kilitlenmekten kurtulur.

NOT: Seri iletisim sirasinda öncelikle LSB (en alt düzeyde bulunan) bit gönderilir.

5. U(S)ART'in RS232 Seri Yoluna Fiziksel Baglantisi

Donanim (Hardware) seviyesinde bir iletisim kullanilacagi vakit elbette ki bir de fiziksel olarak elektronik baglanti olacaktir. Bu belgenin amaci elektronik olmadigi gibi, iletisim icin yazilacak olan programin ve islemcinin iletisim kurarken nasil calistigini anlamak amaciyla teorik acidan fiziksel baglantilari bir nebze bilmenin yarari bulunmaktadir. Bir AVR islemci ve PC arasindaki fiziksel katmani sematik olarak asagidaki gibi gösterebiliriz:





6. Register Tanimi

- UDRn: UART Data Register
- UCSRnX: USART Control and Status Register X
- UBRRnL - UBRRnH: USART Baud Rate Register (LOW ve HIGH)

... n bir tam sayidir ve kullanilacak olan U(S)ART'i belirler.
... Tanimlanmis olan X, registerin adini belirtir ve A,B,C seklindedir.

NOT: Kullanilacak olan register isimleri icin islemcinin katologuna bakilmasi gerekmektedir. Örnegin, UART icin Data Register "UDR" iken, USART icin ayni register ismi "UDR0:1" veya -yukarida ki gibi- "UCSRnX" olarak adlandirilir.

7. BAUD Hesaplama

$$UBRR = \frac{MCU\_hizi\ (Hz)}{16 * Baudrate} - 1$$

- Örnegin, 8 MHz MCU hizindaki islemcinin 9600 Baudrate UART hizi icin UBRR register degeri:

$$UBRR = \frac{8.000.000\ Hz}{16 * 9600} - 1 = 51\ (00110001)$$

* U(S)ART Hata Analizi: Bir seri iletisim yapilirken Verici ve Alici'nin U(S)ART hizi hesaplamasinda tam degerler ideal olarak alinmaktadir. Yukarida verdigimiz örnekte aslinda UBRR 51,0833 gibi bir deger olmalidir. Ancak tam sayi degerler kullanildigi icin UBBR 51 degerini almistir.
Asagidaki formül kullanilarak hata analizi yapilir:

$$[\%]\ Hata\_baud = \left(\frac{UBRR\_kullanilan + 1}{UBRR\_hesaplanan + 1} - 1\right) * 100$$

Fakat bu hata payini daha da aza indirip, islemcinin katologundaki ön görülen degerleri elde etmek icin program yazarken asagidaki formülü kullanacagiz:

$$UBRR = \frac{MCU\_hizi\ (Hz) + (Baudrate * 8)}{(16 * Baudrate)} - 1$$

Program yazarken UBRR hesabi icin genellikle makro kullanilir. Asm ve C kodu

olarak su sekilde yazilir:

* ASM:

```
.equ    F_CPU = 8000000     ; MCU Freq.
.equ    BAUD  = 9600        ; Baudrate

.equ    UBRR_VAL    = ((F_CPU + BAUD*8) / (BAUD * 16) - 1)
.equ    BAUD_REAL   = (F_CPU / (16 * (UBRR_VAL + 1)))
.equ    BAUD_ERROR  = ((BAUD_REAL * 1000) / BAUD - 1000)

.if ((BAUD_ERROR < -10) || (BAUD_ERROR > 10))   ; max. +/-10 hata payi
        .error "Sistem Hatasi: Baudrate Hata Yüzdesi %1'den Daha Büyük!"
.endif
```

* C :

```
#ifndef F_CPU
#warning "F_CPU tanimlanmamis ve 8 MHz olarak alinacaktir!"
#define F_CPU       8000000UL
#define BAUD        9600

#define UBRR_VAL    ((F_CPU + BAUD*8) / (BAUD * 16) - 1);
#define UBRR_REAL   (F_CPU / (16 * (UBRR_VAL + 1)));
#define UBRR_ERROR  ((BAUD_REAL * 1000) / BAUD - 1000);

#if ((BAUD_ERROR < -10) || (BAUD_ERROR > 10))   ; max. +/-10 hata payi
        #error Sistem Hatasi: Baudrate Hata Yüzdesi %1'den Daha Büyük!
#endif
```

8. Hardware UART

Hardware UART tamamen donanimsal olarak islemektedir. Yani, programci islemcinin iletisim hizini ve iletisimde kullanilacak olan verinin boyutunu v.b. ayarladigi vakit, tek yapilmasi gereken data göndermek ve almaktir. Islemci su sekilde ayarlanabilir:

* ASM:

```
.def    temp = r16

; Baud'u ayarlama
ldi     temp, HIGH(UBRR_VAL)
out     UBRRH, temp
ldi     temp, LOW(UBRR_VAL)
out     UBRRL, temp

; Data formati: 8N1
ldi     temp, ( (1 << URSEL) | (1 << UCSZ1) | (1 << UCSZ0) )
out     UCSRC, temp

; Transmit (TX) ve Receive (RX) bitlerini aktif duruma getir.
; Bu satirlar hem Verici hem de Alici durumu icin ayni anda ayarlandi.
; Bu yüzden kullanilan µC hem Verici hem de Alici konumundadir.
sbi     UCSRB, TXEN
sbi     UCSRB, RXEN
```

* C:

```
void
```

```
uart_init(void)
{
        /* Baud ayarlaniyor */
        UBRRH = (UBRR_VAL >> 8);
        UBRRL = (UBRR_VAL & 0xFF);

        /* Data formati: 8N1 */
        UCSRC = ( (1 << URSEL) | (1 << UCSZ1) | (1 << UCSZ0) );

        /* Transmit (TX) ve Receive (RX) iletisimini aktiflestir */
        UCSRB = (1 << TXEN) | (1 << RXEN);
}
```

UART'i ayarladiktan sonra artik μC bir bilgisayar veya baska bir μC ile iletisime gecebilir. Su durumda Verici "Hello World!" yazisini asagidaki gibi gönderilecektir ve program kodu su sekildedir:

* ASM:

```
; Flash'a yazilacak olan yigin
str:    .db "Hello World!"

;
; Programin bu kisminda yukaridaki U(S)ART ayari bulunur.
;

loop:
        /* Z Pointer'i Flash'a yükle */
        ldi     zl, LOW(str * 2)
        ldi     zh, HIGH(str * 2)
        rcall   uart_puts
        rjmp loop

uart_puts:
        lpm                     ; Flash'tan bir sonraki bayti yükle
        and     r0, r0          ; = NULL ??
        breq    uart_puts_end   ; Eger islem NULL ise Flash bos ve iletisimi
                                ; sonlandir

uart_puts_wait:
        sbis    UCSRA, UDRE     ; 1 Byte gönderilmeye hazir olana kadar bekle
        rjmp    uart_puts_wait

        out     UDR, r0         ; Yigindaki karakterleri tek tek gönder
        adiw    zl:zh, 1        ; Z Pointer'i bir arttir ve U(S)ART'i
                                ; sonraki karaktere hazirla
        rjmp    uart_puts

uart_puts_end:
        ret
```

* C :

```
/*
 * U(S)ART'a tek karakter gönderir
 */
void
uart_putch(const char ch)
{
```

```
        /* Bir sonraki Byte gönderilmeye hazir olana kadar bekle */
        while ( !(UCSRA & (1 << UDRE)) )
                ;

        /* ch karakterini UDR registerine yaz */
        UDR = ch;
}

/*
 * U(S)ART'a string gönderir
 */
void
uart_puts(const char *str)
{
        /* str yigini sonlanana kadar icinde barindirdigi tüm karakterleri
         * teker teke U(S)ART'a gönderir. Ne vakit son karakter olarak '\0'
         * gelir, o zaman while-döngüsü terk edilir.
         */
        while (*str)
                uart_putch(*str++);
}
```

Tabii ki iletisimde bir Verici oldugu gibi bir de Alici olacaktir. Fakat Alici eger PC degil ise, yahut gönderilen yigini gösterebilecek bir görsel aygit (örnegin LCD) yok ise, alinan karakterler ancak ve ancak cikis olarak ayarlanan bir PORT'tan cikis yapilacaktir (veya baska bir Alici'ya gönderilecektir). Bu durumda, yalnizca her karakteri ASCII kodunu iceren bir PORT görüntüsüne sahip olunacaktir. Eger PORT'un uclarina (pinlere) LED baglanmis ise, alinan karakterin ASCII koduna göre bunlardan kimisi yanacaktir, kimisi ise sönük kalacaktir. Örnegin, 'A' karakteri alinmis ise, ASCII kodu 65 ve PORT'un görüntüsü su sekilde konumlanacaktir: 0100001.
Alici'nin kodu da asagidaki gibidir:

* ASM:

```
.def    temp = r16

ldi     temp, 0xFF
out     DDRD, temp      ; PORT D'nin tüm pinleri cikis olarak ayarlandi

ldi     temp, 0x00

;
; Programin bu kisminda yukaridaki U(S)ART ayari bulunur.
;

recv_loop:
        sbis    UCSRA, RXC      ; Bir veri gelene kadar bekle
        rjmp    recv_loop

        in      temp, UDR       ; Gelen veriyi gecici temp registerine kopyala
        out     PORTD, temp     ; ve veriyi PORT D'den cikis yap
        rjmp    recv_loop
```

* C:

```
uint8_t
uart_getch(void)
{
        /* U(S)ART'a bir veri gelene kadar bekle */
```

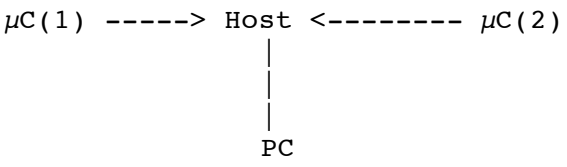
```
        while ( !(UCSRA & (1 << RXC)) )
                ;

        /* Alinan verisi cagirilan fonksiyona döndür */
        return UDR;
}
```

Gördügümüz bu Hardware UART iletisiminde yalnizca birisi "Hello World!" gönderirken, digeri de bu yigini almaktadir.

Biraz daha gelismis bir iletisim agi düsünür isek, örnegin; host görevini gören ve kendisine veri gönderen 2 tane μC ile PC'yi bilgilendiren bir sistemin iletisimi nasil kurulacaktir?

Sema ile gösterecek olur isek, su sekilde bir ag kurulabilir:



Farzedelim ki bu sistem su sekilde isliyor:

- μC(1) devamli olarak ortamin (yahut bir aygitin) sicakligini ölcüyor, ve eger degisim olur ise Host'a gönderiyor.

- μC(2) de her saniye tarih bilgisini (GG.AA.YYYY - SS:DD:ss) olarak Host'a gönderiyor.

- Host ise her μC(1)'den aldigi "her degisim ile" tam o vakit μC(2)'den en son alinan (min. 1 saniye önceki) tarih bilgisini PC'ye gönderiyor.

Iletisimin saglikli olabilmesi icin, μC(1), μC(2) ve Host'un ayni yapiya sahip veri tabanini kullanmalari gerekmektedir. Mesela;

```
typedef union
{
        int             sender;     /* μC(1) = 1, μC(2) = 2 */

        uint16_t        sicaklik;
        struct {
                unsigned int    gun;
                unsigned int    ay;
                unsigned int    yil;
                unsigned int    saat;
                unsigned int    dakika;
                unsigned int    saniye;
        } tarih;
} data_t;
```

Su durumda, eger bir μC data_t dipinde bir veri gönderdigi vakit Host öncelikle "sender" kismina bakacak ve gönderilen verinin kime ait oldugunu algilayacaktir. Eger μC(1) bir veri göndermis ise "sicaklik" önemli olacagindan Host diger bilgiler ile ilgilenmeyecektir. Eger μC(2) veri göndermis ise Host "tarih" bilgisini alacak fakat sicaklik kismi ile ilgilenmeyecektir.

Bu sistemin saglikli bir sekilde iletisim kurabilmesi icin Verici ve Alici'nin TX ve RX pinlerini dinlemesinin yaninda biraz daha farkli bir algoritma

gelisilmesi gerekmektedir. Cünkü, μC(1) ve μC(2) ayni anda Host'a veri gönderdikleri vakit, Host yalnizca bir veriyi degerlendirebilecektir. Bu problemin çözülebilmesi icin de gönderilen verilerin birisine öncelik vermek gerekmektedir. Bizim sistemimizde önceligi μC(1)'in verisi almalidir, cünkü PC basinda bulunan kisiyi en cok ilgilendirecek olan ortamin (veya ölcülen aygitin) sicakligi kismidir. Bu yüzden Host, μC(1)'in önceligini (priority) yükseltmektedir.

Eger sistemimizin calisma durumunu biraz daha gelistirmek istersek, ve asagidaki durumlar ek olarak eklenir ise:

- Ölcülen aygitin sicakligi belirli bir seviyeye ciktiginda Host'un μC(1)'e bir sinyal gönderip bagli oldugu aygiti sonlandirmasi gerekmektedir.

Bu durumda, μC(1) de Host gibi hem Alici hem de Verici durumuna gececektir. Ancak Host'tan gelecek olan sinyali devamli olarak bekledigi vakit ölcümünde veya diger islemlerinde bir aksaklik cikma olasiligi da dogacaktir. Bu problemin üstesinden gelmenin yegane yolu μC(1)'de RX pini icin U(S)ART Receive Interrupt Routine kurulmasi. Yani, Host gönderecegi bir sinyal ile μC(1)'in o an yaptigi tüm islemleri kesip U(S)ART Interrupt bayragini kaldirmasi ve bagli oldugu aygiti sonlandirmasi gerekmektedir. Alici'nin Interrupt yolu ile bilgi almasini asagidaki gibi bir kod yazilabilir:

* ASM:

```
;
; BAUD hesabi
;

.org    0x00
        rjmp    main

; U(S)ART Alici durumu icin Interrup Vektörü
.org    URXCaddr
        rjmp    int_recv

main:
        ; Stack Pointer
        ldi     temp, HIGH(RAMEND)
        out     SPH, temp
        ldi     temp, LOW(RAMEND)
        out     SPL, temp

;
; Yukarida tanimlanan U(S)ART ayari
;

        sbi     UCSRB, RXEN         ; Receive (RX) icin Interrupt (Kesme)

        sei                         ; Tüm kesmeler aktif

loop:
        ; - sonsuz döngü -
        ; eger yapilmasi gereken baska islemler var ise onlar eklenebilir
        rjmp    loop

int_recv:
        push    temp        ; temp icindeki bilgileri Stack'ta saklar
        in      temp, UDR   ; Alinan verisi okur

        ; Alinan veri ile yapilmasi gerekenler yapilir
```

```
        pop     temp        ; Stack icinde saklanan temp bilgisi geri yüklenir
        reti                ; Kesmeden geri dönüs
```

```
* C:
void
uart_init(void)
{
        /*
         * Yukarida tanimlanan U(S)ART ayarlari
         */

         /* U(S)ART Receive (RX) Interrupt Routine */
         UCSRB |= (1 << RXCIE);
}

int
main(void)
{
        /* mcu ve diger port ayarlari */

        uart_init();

        /* Tüm tanimlanan kesmeler aktif */
        sei();

        /* Sonsuz döngü */
        while(1)
                ;

        return 0;
}

SIGNAL(SIG_USART_RECV)
{
        /*
         * Alinan veri ile yapilmasi gerekenler yapilir
         */
}
```

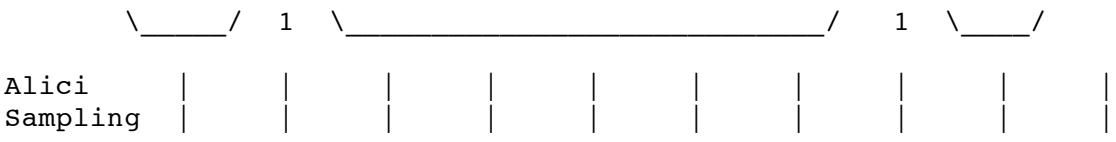
NOT: C icin tanimli olan Interrupt isimleri icin lütfen islemcinin katogouna veya http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html adresine bakiniz. Tanimli olan iki farkli kesme vektörleri icin SIGNAL ve ISR isimleri de farklilik arz etmektedir. Tabii ki kullanilan islemcinin özelligine göre de farkli isimlendirmeler bulunmaktadir.

9. Software UART

Seri iletisimde kullanilan diger bir arayüz de Software UART'tir. Bu tür iletisim bir vakit icerisinde yalnizca tek yönlü (Half Duplex) olarak calismaktadir. Hardware UART'tan farkli olarak, Software UART'i ayarlarken Alici ve Verici'yi zamanlayici (Timer) kullanarak senkronize edip veri tasinmasini bu sekilde saglamaktir.

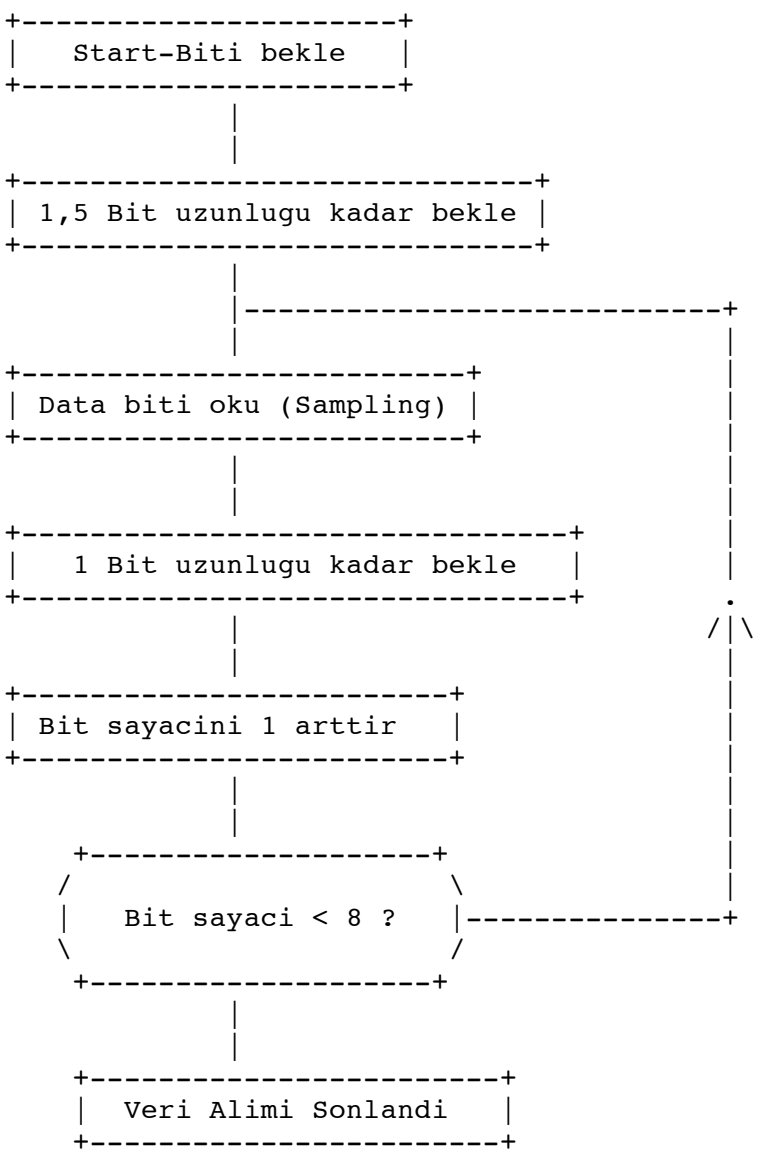
```
* Software UART ile ASCII 'A' (ASCII: 65) (Hex: $41) gönderimi

IDLE                                                        IDLE
_______      ___                                    ____       _________
       \ Start /   \   0     0     0     0     0   /    \   0  /  STOP
```

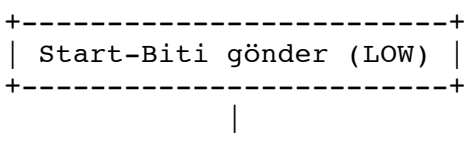


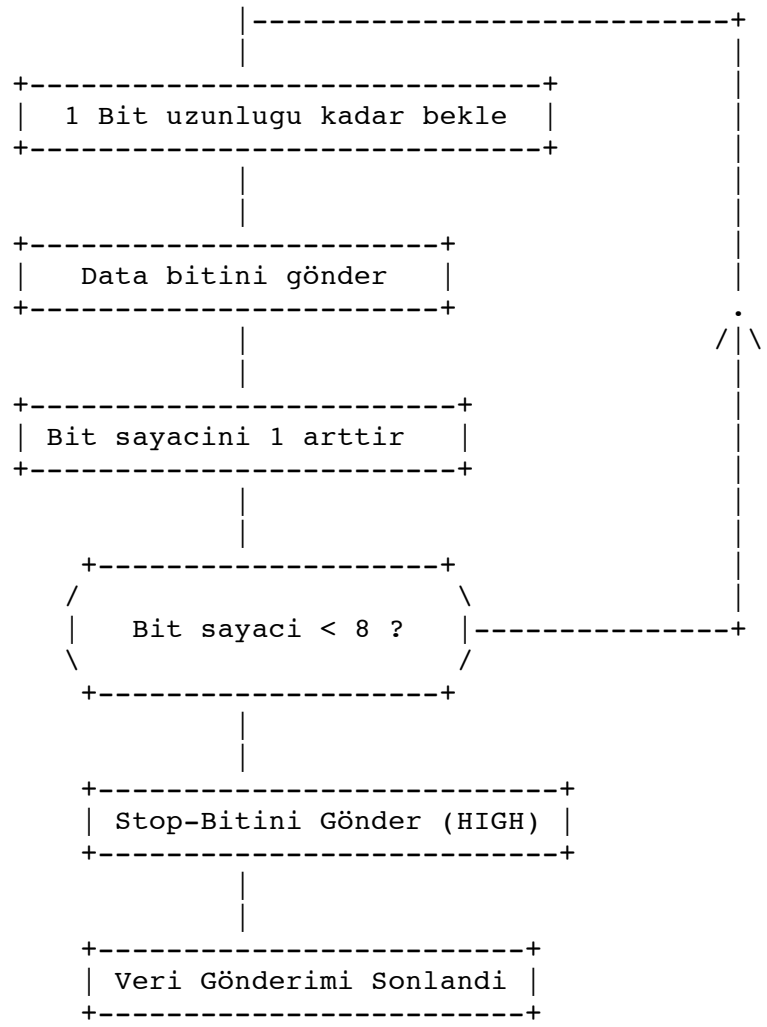
Yukarida ki örnekte de görüldügü gibi, IDLE durumunda olan RX pini HIGH olarak aktif haldedir. LOW konumuna gectigi vakit islemci gelen sinyalin Start-Bit oldugunu algilar ve periyodik olarak kurulan zamanlayici ile üretilen her kesmede gelen bitleri okumaya baslar. Alinacak olan verinin uzunluguna göre de (örnegin, 8N1) okunan bitler kaydedilir ve en son olarak Stop-Bit kontrolü yapilip alinan veri islenmeye hazir hale gelir.

Tüm bu islemleri algoritmik olarak gösterecek olur isek, Alici su sekilde calismaktadir:



Ayni sekilde, Verici de Timer kullanarak benzer bir algoritma ile verileri gönderir:





Bir bitin gönderilmesi icin gerekli olan zamani hesaplarken islemcinin calisma frekansi ve U(S)ART'in tasima hizi kullanilir.

$$Z_{bit} = \frac{F_{CPU}\ (Hz)}{BAUDRATE}$$

Örnegin, 8 Mhz ve 9600 Baud hizinda calisan islemci ve U(S)ART icin gerekli olan bir bit tasima süresi -yaklasik olarak- 833 salinimdir (cycle). Ve Alici ile Verici'nin zamanlayici periyodu da bu salinim süresine göre ayarlanir.

Alici ve Verici icin Software U(S)ART'ta farkli olarak islemcinin TX ve RX pinlerinin bulundugu PORT bacaklarinin durumlari kontrol edilmektedir. Bu yüzden öncelikle bu pinleri -Data Direction Register icerisinde-, TX cikis (output) olarak, RX ise giris (input) olarak ayarlanmalidir.

```
void
rx_init(void)
{
        /*
         * RX pin giris (input) olarak ayarla ve
         * pull-up direncini (HIGH) aktif hale getir (IDLE durumu icin)
         */
        SOFT_UART_RX_DDR &= ~(1 << SOFT_UART_RX_PIN);
        SOFT_UART_RX_PORT |= (1 << SOFT_UART_RX_PIN);
}
```

```
void
tx_init(void)
{
        /*
         * TX pinini cikis (output) olarak ayarla ve
         * pini HIGH aktif hale getir (IDLE durumu icin)
         */
        SOFT_UART_TX_DDR |= (1 << SOFT_UART_TX_PIN);
        SOFT_UART_TX_PORT |= (1 << SOFT_UART_TX_PIN);
}
```

Simdi de Alici'da kullanilacak olan Sampling islemi icin zamanlayiciyi (Timer) MCU ve U(S)ART iletisim hizlarina göre ayarlanacak.

```
void
recv_timer(void)
{
        /* Timer ayari yapilirken tüm kesmeler kapatiliyor */
        cli();

        /* Timer/Counter Control Register A */
        TCCR1A = 0x00;

        /* Timer/Counter Control Register B */
        /*
         * Timer CTC Mode: TOP = OCR1A, TOV = MAX
         * Timer Prescaler: 8 (CLK_io / 8)
         * ICES1 : Input Capture Edge (Falling Edge)
         * ICNC1 : Input Capture Noise Cancaler
         */
         TCCR1B = (1 << WGM12) | (1 << CS10) | (1 << ICES1) | (1 << ICNC1);

         /* Timer Interrupt periyodu icin gerekli olan üst sinir */
         OCR1A = (uint16_t) ((uint32_t)F_CPU / BAUDRATE);

        /* Interrupt Capture Interrupt Enable */
         TIMSK |= (1 << TICIE1);

        /* Input Capture Flag, Output Compare B Match Flag */
         TIFR |= (1 << ICF1) | (1 << OCF1B);
}
```

Ve Verici ile Alici'nin sekron halde calismasi icin Verici icin de bir zamanlayici ayari yapilmalidir.

```
/* output data */
static volatile uint16_t out_frame;

void
trans_timer(void)
{
        /* Timer ayari yapilirken tüm kesmeler kapatiliyor */
        cli();

        /* Timer/Counter Control Register A */
        TCCR1A = 0x00;

        /* Timer/Counter Control Register B */
        /*
         * Timer CTC Mode: TOP = OCR1A, TOV = MAX
```

```
         * Timer Prescaler: 8 (CLK_io / 8)
         * ICES1 : Input Capture Edge (Falling Edge)
         * ICNC1 : Input Capture Noise Cancaler
         */
        TCCR1B = (1 << WGM12) | (1 << CS10) | (1 << ICES1) | (1 << ICNC1);

        /* Timer Interrupt periyodu icin gerekli olan üst sinir */
        OCR1A = (uint16_t) ((uint32_t)F_CPU / BAUDRATE);

        /* Input Capture Interrupt deaktif */
        TIMSK &= ~(1 << TICIE1);

        /* Output Compare A Match Flag */
        TIFR |= (1 << OCF1A);

        out_frame = 0x0000;
}
```

Timer ve U(S)ART ayarlandiktan sonra istedigimiz veriyi gönderip alabiliriz.
Verici bir karakteri göndermek icin su sekilde programlanir:

```
void
uart_putc(const char c)
{
        /*
         * Gönderilecek olan verinin yapisi:
         * IDLE - P.7.6.5.4.3.2.1.0.S - IDLE
         * P: Stop-Bit (HIGH), S: Start-Bit (LOW)
         */
        out_frame = (3 << 9) | (((uint8_t) c) << 1);

        /* Output Compare A Match Interrupt Enable */
        TIMSK |= (1 << OCIE1A);

        /* Output Compare A Match Flag */
        TIFR = (1 << OCF1A);

        /* Kesmeleri aktiflestir */
        sei();
}
```

U(S)ART'a gönderilmek istenilen her karakter "out_frame" isminde global bir
degiskene kaydedilir. Ardindan zamanlayici icin kesme (interrupt) ayarlanip
aktif hale getirilir, ve her kesme oldugu vakit Interrupt fonksiyonu cagirilir,
TX pini verinin ASCII kodundaki bit durumuna göre HIGH veya LOW hale getirilir.

```
SIGNAL(SIG_OUTPUT_COMPARE1A)
{
        uint16_t data = out_frame;

        /*
         * Eger gönderilen bit '1' ise TX pini HIGH konumuna gecer,
         * eger '0' ise LOW ile konumlandirilir.
         */
        if (data & 0x0001)
                SOFT_UART_TX_PORT |= (1 << SOFT_UART_TX_BIT);
        else
                SOFT_UART_TX_PORT &= ~(1 << SOFT_UART_TX_BIT);

        /*
         * Eger gönderilen data son bite ulasmis ise
```

```
         * Output Compare 1A Interrupt Routine deaktif edilir.
         */
        if (0x0001 == data)
                TIMSK &= ~(1 << OCIE1A);

        /*
         * Gönderilen veri bir bit saga kaydirilarak
         * bir sonraki bit gönderilemk icin hazir edilir.
         */
        out_frame = (data >> 1);
}
```

Gönderilen bir bit Alici tarafindan alinmaya basladiginda öncelikle Input Capture Interrupt bayragi aktif olunur ve Sampling yapilacak olan periyod ayarlanir. Ardindan zamanlayici periyodik olarak Output Compare Interrupt kesmesini her cagirdiginda RX pininin durumu okunur. Stop-Bit geldigi vakit kaydedilmis olan tüm pin durumlari gönderilmis olan veriyi icerir.

```
/* Global degiskenler */
static volatile uint16_t in_frame;
static volatile uint8_t in_bits, received;
static volatile uint8_t in_data;

SIGNAL(SIG_INPUT_CAPTURE1)
{
        /*
         * Kesme oldugu anda ICR1 ve OCR1A kütükleri
         * icerisinde bulunan degerler kaydedilir.
         */
        uint16_t icr1 = ICR1;
        uint16_t ocr1a = OCR1A;

        /*
         * Iletisim hizini ayarlarken OCR1A'da sakladigimiz periyodun yarisini
         * ICR1 üzerine ekleyip Sampling islemi yapacak olan OCR1B'ye yüklenir,
         * ve böylece zamanlayicinin periyodu ayarlanmis olur.
         */
        uint16_t ocr1b = icr1 + ocr1a / 2;

        if (ocr1b >= ocr1a)
                ocr1b -= ocr1a;

        /* Sampling icin periyot ayarlanir */
        OCR1B = ocr1b;

        /* Output Compare B Match Flag */
        TIFR = (1 << OCF1B);

        /*
         * Veri alimi basladigi anda Input Capture icin kesme kapatilir
         * Sampling icin olan zamanlayici kesmesi aktif edilir.
         */
        TIMSK &= ~(1 << TICIE1);
        TIMSK |= (1 << OCIE1B);

        in_frame = 0;
        in_bits = 0;
}

SIGNAL(SIG_OUTPUT_COMPARE1B)
{
```

```
        uint16_t data = (in_frame >> 1);

        /* Eger RX pini HIGH aktif ise data icine bu bit icin '1' eklenir */
        if (SOFT_UART_RX_PIN & (1 << SOFT_UART_RX_BIT))
                data |= (1 << 9);

        uint8_t bits = (in_bits + 1);

        /*
         * Alinan tüm bit sayisi eger 10 ise ve Stop-Bit te Alici'Y ulasmis ise
         * zamanlayici kapatilir ve Input Capture tekrar aktiflestirilir.
         * Eger bit sayaci hala 10. bite ulasmamis ise bir sonraki gelecek olan
         * bit beklenir.
         */
        if (10 == bits) {
                if ((data & 0x0001) == 0) {
                        if (data >= (1 << 9)) {
                                in_data = (data >> 1);
                                received = 1;
                        }
                        TIMSK &= ~((1 << OCIE1B) | (1 << TICIE1));
                        TIFR = (1 << ICF1);
                }
        } else {
                in_bits = bits;
                in_frame = data;
        }
}


int
uart_getc(void)
{
        while (!received)
                ;

        received = 0;

        return (int) in_data;
}
```

Software UART yöntemi kullanilarak kontrol edilecek olan sistemler gelistirilip, bu gelismis sistemlerin birbirleri arasindaki iletisim protokolleri icin de düzenli calisabilecek bir ag topolojisi tasarlanabilir.

10. Bitirirken

Bu belgeyi hazirlarken Almanca ders notlari ve Ingilizce kaynaklardan yararlandigim icin -su an- literatürde kullnilan Türkce terimlerde bir hata yapmis isem bu hatamin mazur görülmesini istirham ederim.

11. Kaynaklar

- http://www.mikrocontroller.net/
- http://www.rn-wissen.de/
- Atmel AVR Application Notes