

Note

This article is a part of **Arduino / ATmega328p Embedded C Firmware Programming Tutorial**. Consider exploring the course home page for articles on similar topics.



ARDUINO PROGRAMMING TUTORIAL

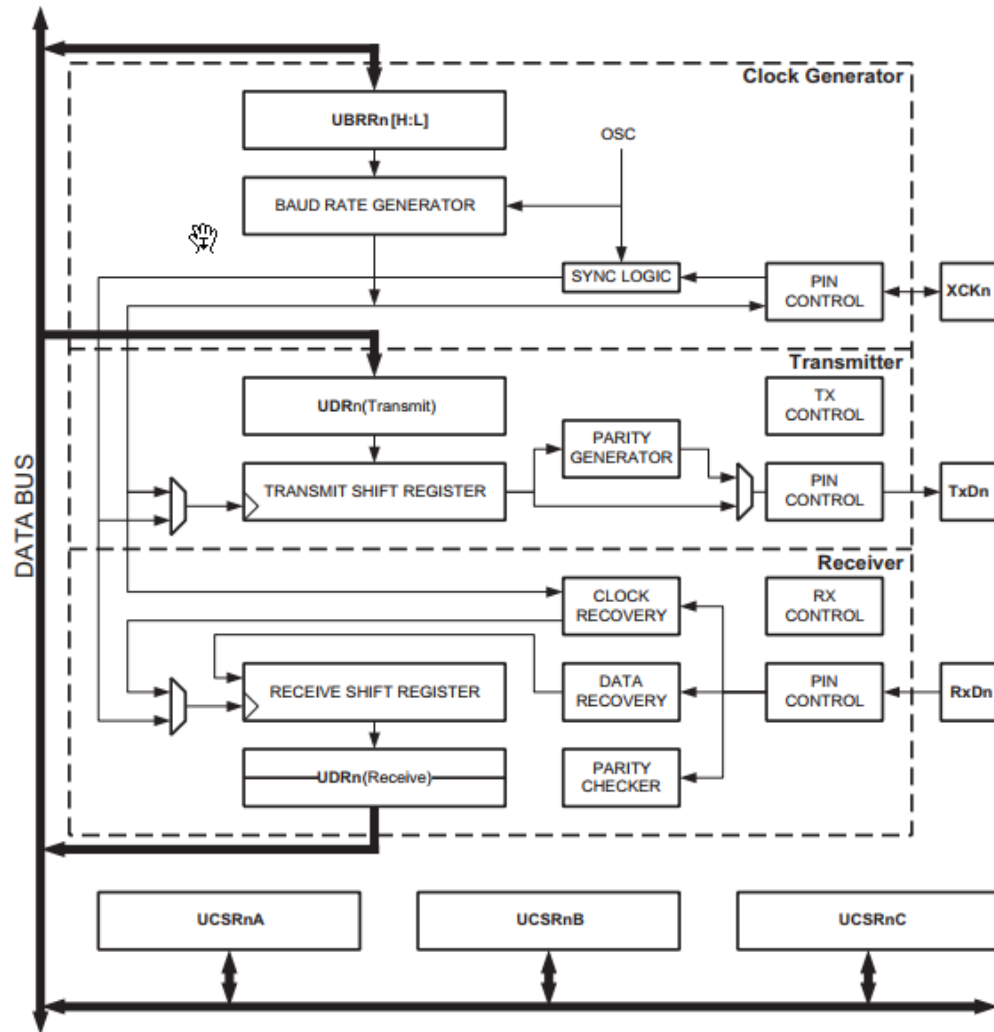
Arduino Tutorial Embedded C Register Level Arduino Master Class

Also visit the **Release Page** for **Register Level Embedded C Hardware Abstraction Library and Code for AVR**.

Arnab Kumar Das



The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device. The USART hardware in ATmega48A/PA/88A/PA/168A/PA/328/P is represented as USART0.



AVR USART Register Configuration

The major hardware components in USART are Clock Generator, Transmitter, and Receiver.

The Clock Generation logic consists of synchronization logic for external clock input used by synchronous slave operation, and the baud rate generator.

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.

(UDRn). The Receiver supports the same frame formats as the Transmitter and can detect Frame Error, Data OverRun, and Parity Errors.

Each of the hardware units needs to be configured by writing bits in their respective control registers. The USART supports four modes of operation: Normal asynchronous, Double Speed asynchronous, Master synchronous, and Slave synchronous mode.

What You Will Learn

- How to Program the UART in Arduino?
- How to do UART Programming in AVR ATmega328p?
- How to Transmit and Receive Data using UART communication in Arduino/ATmega328p?
- How to Program UART for Polling and Interrupt based communication?
- How to Transmit and Receive data to and fro Computer and Arduino/ATmega328p?

Prerequisite

- Knowledge of C/C++ programming

Hardware Bill of Materials

- **Arduino UNO**
- **USBasp** (Optional, when Arduino's serial programming is not used)

Software Bill of Materials

- Atmel Studio 7
- Arduino IDE
- Arduino Drivers
- USBasp Drivers (Needed when USBasp is used)

Schematic Connection

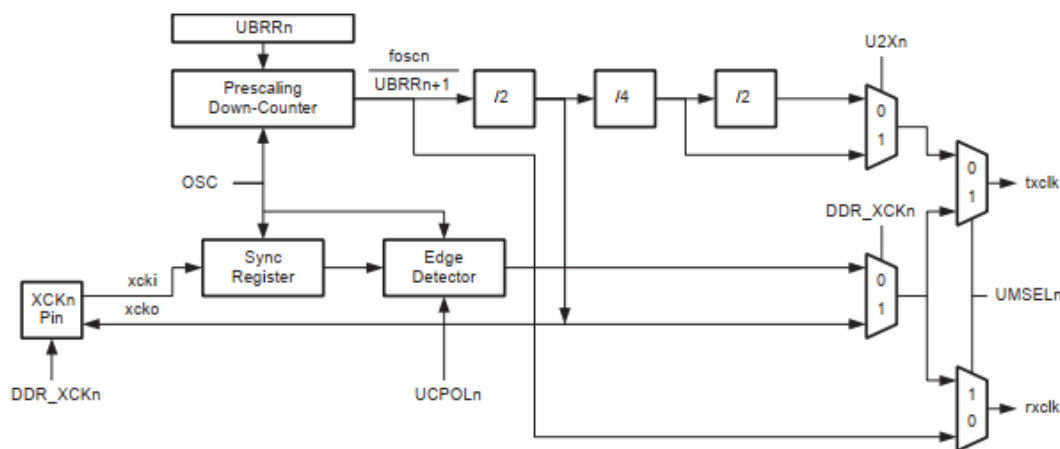
Arnab Kumar Das



In case you are not using the Arduino's serial programming to flash the microcontroller, you will also need an additional USBasp to connect your computer and Arduino board. In this condition, you need two USB connections to your Arduino. One via USBasp to flash and another one for serial communication.

USART Programming

The three major hardware components that need to be initialized before any communication are Clock Generator, Transmitter, and Receiver. The initialization process normally consists of setting the baud rate, setting frame format, and enabling the Transmitter or the Receiver.



Arduino UNO / Atmega328p USART Clock Circuit

The baud rate is generated from the system clock with the help of Prescaler and clock circuit. The USART Baud Rate Register (UBRR0) controls the programmable down counter / Prescaler to generate a particular clock signal. The down-counter, running at system clock (f_{osc}), is loaded with the UBRR0 value each time the counter has counted down to zero and generates a clock pulse.

Asynchronous Normal mode
(U2Xn = 0)

$$BAUD = \frac{f_{osc}}{16(UBRRn + 1)}$$

$$UBRRn = \frac{f_{osc}}{16BAUD} - 1$$

Arduino UNO / ATmega328p USART Baud Rate Calculation



Baud Rate (bps)	UBRR0	Error %
2400	416	-0.1
4800	207	0.2
9600	103	0.2
14.4k	68	0.6
19.2k	51	0.2
28.8k	34	-0.8
38.4k	25	0.2
57.6k	16	2.1
76.8k	12	0.2
115.2k	8	-3.5
230.4k	3	8.5
250k	3	0.0
0.5M	1	0.0
1M	0	0.0

So from the above table, it is easy to choose the available baud rates with their respective UBRR0. At 16Mhz the highest communication speed we can reach is 1Mbps.

The next step is to set the Frame Format using UCSR0C register. The USART accepts all 30 combinations of the following as valid frame formats:

- 1 start bit
- 5, 6, 7, 8, or 9 data bits

The next step is to enable the Transmitter and Receiver to use UCSRB register and load the UBR0 register with the data to transmit. In the case of reception, the UBR0 is read by the application.

NOTE: All the code below can be compiled and flashed both from Atmel Studio and Arduino IDE. Use any Serial Monitor at 9600bps, 1 Stop Bit, No Parity. I recommend using the Data Visualizer in Atmel Studio for Serial Port Terminal in case you are programming and flashing from Atmel Studio.

Polling Transmission

Polling transmission is the simplest method of transmission where the application software keeps monitoring the status of the USART transmitter hardware and loads a byte of data into the USART buffer UDR0 only when the hardware is ready for transmission. This wastes CPU time in constantly monitoring the status of UDRE0 bit of UCSRA register.

```

1  /*
2  * usart.c
3  *
4  * Created : 15-08-2020 07:24:45 PM
5  * Author  : Arnab Kumar Das
6  * Website : www.ArnabKumarDas.com
7  */
8
9  #define F_CPU 16000000UL // Defining the CPU Frequency
10
11 #include <avr/io.h>      // Contains all the I/O Register Macros
12 #include <util/delay.h>  // Generates a Blocking Delay
13
14 #define USART_BAUDRATE 9600 // Desired Baud Rate
15 #define BAUD_PRESCALER (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)
16
17 #define ASYNCHRONOUS (0<<UMSEL00) // USART Mode Selection
18
19 #define DISABLED (0<<UPM00)
20 #define EVEN_PARITY (2<<UPM00)
21 #define ODD_PARITY (3<<UPM00)
22 #define PARITY_MODE DISABLED // USART Parity Bit Selection
23
24 #define ONE_BIT (0<<USBS0)
25 #define TWO_BIT (1<<USBS0)
26 #define STOP_BIT ONE_BIT // USART Stop Bit Selection
27
28 #define FIVE_BIT (0<<UCSZ00)
29 #define SIX_BIT (1<<UCSZ00)
30 #define SEVEN_BIT (2<<UCSZ00)
31 #define EIGHT_BIT (3<<UCSZ00)
32 #define DATA_BIT EIGHT_BIT // USART Data Bit Selection
33
34 void USART_Init( )
35 {
36     // Set Baud Rate
37 }

```

Arnab Kumar Das

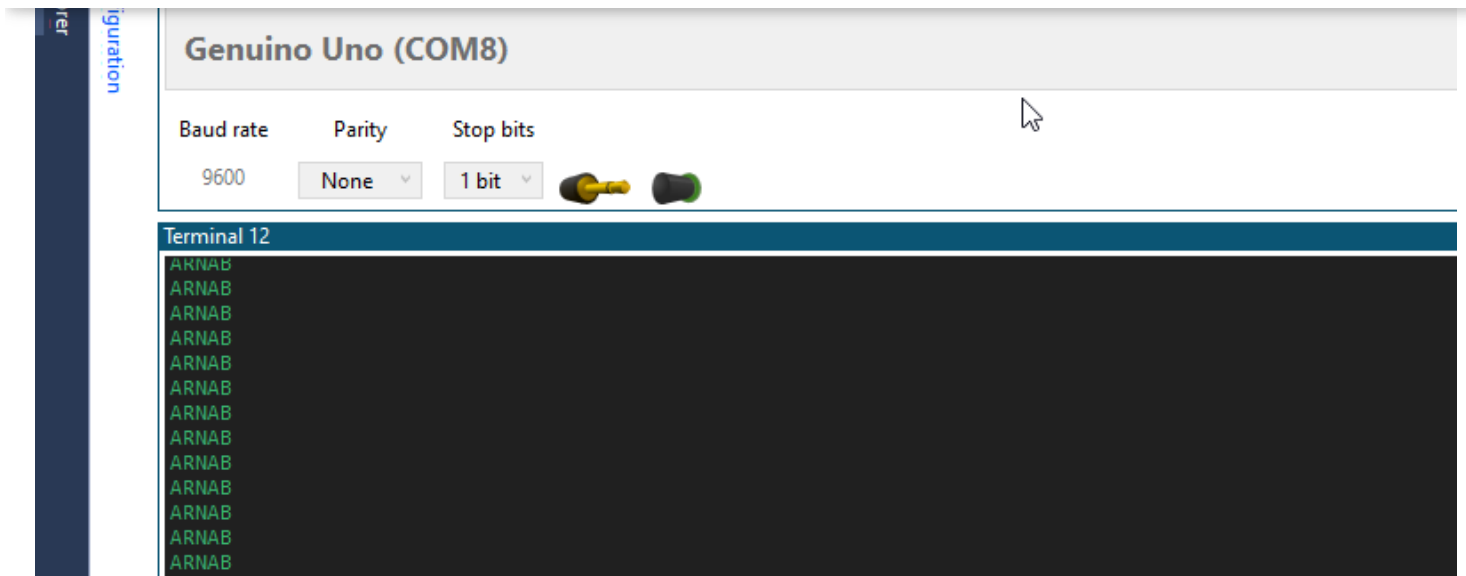


```
42  UCSRC = ASYNCHRONOUS | PARITY_MODE | STOP_BIT | DATA_BIT;  
43  // Enable Receiver and Transmitter  
44  UCSRB = (1<<RXEN0) | (1<<TXEN0);  
45  }  
46  
47  void USART_TransmitPolling(uint8_t DataByte)  
48  {  
49      while ((UCSR0A & (1<<UDRE0)) == 0) {}; // Do nothing until UDR is ready  
50      UDR0 = DataByte;  
51  }  
52  
53  int main()  
54  {  
55      USART_Init();  
56      while (1)  
57      {  
58          USART_TransmitPolling('A');  
59          USART_TransmitPolling('R');  
60          USART_TransmitPolling('N');  
61          USART_TransmitPolling('A');  
62          USART_TransmitPolling('B');  
63          USART_TransmitPolling('\n');  
64          _delay_ms(1000);  
65      }  
66      return 0;  
67  }
```

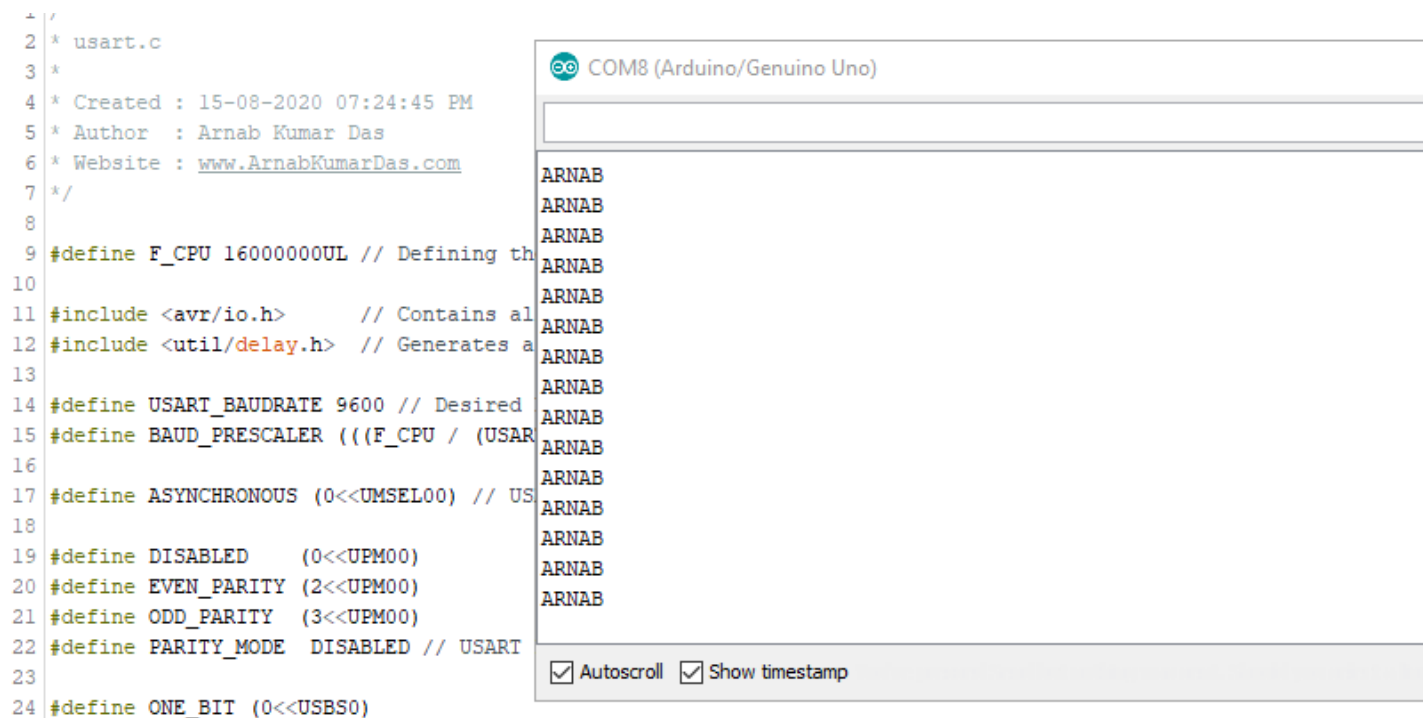
The output of the above code is "ARNAB" every 1 Second in the Serial Monitor.

```
ARNAB  
ARNAB  
ARNAB
```

Arnab Kumar Das



Arduino UNO / Atmega328p USART Atmel Studio Output



Arduino UNO / Atmega328p USART Arduino IDE Output

Polling Reception

Polling reception is the simplest method of reception where the application software keeps monitoring the status of the USART receiver hardware and reads the data from the USART buffer UDR0 only when the hardware has received a byte of data. This wastes CPU time in constantly monitoring the status of RXC0 bit of UCSRA register.

```

/*
 * usart.c
 *
 * Created : 15-08-2020 07:44:46 PM
 * Author  : Arnab Kumar Das
 * Website : www.ArnabKumarDas.com
 */

#define F_CPU 16000000UL // Defining the CPU Frequency

#include <avr/io.h> // Contains all the I/O Register Macros
#include <util/delay.h> // Generates a Blocking Delay

#define USART_BAUDRATE 9600 // Desired Baud Rate
#define BAUD_PRESCALER (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

#define ASYNCHRONOUS (0<<UMSEL00) // USART Mode Selection

#define DISABLED (0<<UPM00)
#define EVEN_PARITY (2<<UPM00)
#define ODD_PARITY (3<<UPM00)
#define PARITY_MODE DISABLED // USART Parity Bit Selection

#define ONE_BIT (0<<USBS0)
#define TWO_BIT (1<<USBS0)
#define STOP_BIT ONE_BIT // USART Stop Bit Selection

#define FIVE_BIT (0<<UCSZ00)
#define SIX_BIT (1<<UCSZ00)
#define SEVEN_BIT (2<<UCSZ00)
#define EIGHT_BIT (3<<UCSZ00)
#define DATA_BIT EIGHT_BIT // USART Data Bit Selection

void USART_Init()
{
    // Set Baud Rate
    UBRRH = BAUD_PRESCALER >> 8;
    UBRRL = BAUD_PRESCALER;

    // Set Frame Format
    UCSRC = ASYNCHRONOUS | PARITY_MODE | STOP_BIT | DATA_BIT;

    // Enable Receiver and Transmitter
    UCSRB = (1<<RXEN0) | (1<<TXEN0);
}

uint8_t USART_ReceivePolling()
{
    uint8_t DataByte;
    while ((UCSR0A & (1<<RXC0)) == 0) {}; // Do nothing until data have been received
    DataByte = UDR0;
    return DataByte;
}

int main()
{
    DDRB |= 1 << 5; // Configuring PB5 / D13 as Output

```



```

63     LocalData = USART_Receive();
64     if (LocalData == 'a')
65     {
66         PORTB |= 1<<5;    // Writing HIGH to glow LED
67     }
68     else
69     {
70         PORTB &= ~(1<<5); // Writing LOW
71     }
72     _delay_ms(1000);
73 }
74 return 0;
}

```

Polling Loopback

The loopback test is a great way to verify any communication channel. A loopback test of USART will verify both the reception and transmission side of the code. A loopback test sends back the same data that is received. The below code will echo back the same character that is sent from the serial terminal.

```

/*
 * usart.c
 *
 * Created : 15-08-2020 08:34:15 PM
 * Author  : Arnab Kumar Das
 * Website : www.ArnabKumarDas.com
 */

#define F_CPU 16000000UL // Defining the CPU Frequency

#include <avr/io.h>      // Contains all the I/O Register Macros
#include <util/delay.h>  // Generates a Blocking Delay

#define USART_BAUDRATE 9600 // Desired Baud Rate
#define BAUD_PRESCALER (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

#define ASYNCHRONOUS (0<<UMSEL00) // USART Mode Selection

#define DISABLED (0<<UPM00)
#define EVEN_PARITY (2<<UPM00)
#define ODD_PARITY (3<<UPM00)
#define PARITY_MODE DISABLED // USART Parity Bit Selection

#define ONE_BIT (0<<USBS0)
#define TWO_BIT (1<<USBS0)
#define STOP_BIT ONE_BIT // USART Stop Bit Selection

#define FIVE_BIT (0<<UCSZ00)
#define SIX_BIT (1<<UCSZ00)
#define SEVEN_BIT (2<<UCSZ00)

```

```

36 // Set Baud Rate
37 UBR0H = BAUD_PRESCALER >> 8;
38 UBR0L = BAUD_PRESCALER;
39
40 // Set Frame Format
41 UCSRC = ASYNCHRONOUS | PARITY_MODE | STOP_BIT | DATA_BIT;
42
43 // Enable Receiver and Transmitter
44 UCSRB = (1<<RXEN) | (1<<TXEN);
45 }
46
47 uint8_t USART_ReceivePolling()
48 {
49     uint8_t DataByte;
50     while ((UCSR0A & (1<<RXC0)) == 0) {}; // Do nothing until data have been received
51     DataByte = UDR0;
52     return DataByte;
53 }
54
55 void USART_TransmitPolling(uint8_t DataByte)
56 {
57     while ((UCSR0A & (1<<UDRE0)) == 0) {}; // Do nothing until UDR is ready
58     UDR0 = DataByte;
59 }
60
61 int main()
62 {
63     USART_Init();
64     char LocalData;
65     while (1)
66     {
67         LocalData = USART_ReceivePolling();
68         USART_TransmitPolling(LocalData);
69     }
70     return 0;
71 }

```

Interrupt Transmission

In polling the CPU waste valuable time monitoring the USART registers. This valuable time could be used in the execution of other instructions. This problem is solved by interrupt based transmission. Below code transmits using interrupts. The code transmits the character 'a' endlessly while the D13 LED on Arduino UNO keeps blinking. The CPU keeps performing the LED blinking in an infinite loop and every time the transmission finishes an interrupt is generated to state that the UDR0 buffer is ready to receive new data. The CPU pauses the LED blinking and serves the ISR.

```

/*
* usart.c

```

```

#define F_CPU 16000000UL // Defining the CPU Frequency

#include <avr/io.h> // Contains all the I/O Register Macros
#include <util/delay.h> // Generates a Blocking Delay
#include <avr/interrupt.h> // Contains all interrupt vectors

#define USART_BAUDRATE 9600 // Desired Baud Rate
#define BAUD_PRESCALER (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

#define ASYNCHRONOUS (0<<UMSEL00) // USART Mode Selection

#define DISABLED (0<<UPM00)
#define EVEN_PARITY (2<<UPM00)
#define ODD_PARITY (3<<UPM00)
#define PARITY_MODE DISABLED // USART Parity Bit Selection

#define ONE_BIT (0<<USBS0)
#define TWO_BIT (1<<USBS0)
#define STOP_BIT ONE_BIT // USART Stop Bit Selection

#define FIVE_BIT (0<<UCSZ00)
#define SIX_BIT (1<<UCSZ00)
#define SEVEN_BIT (2<<UCSZ00)
#define EIGHT_BIT (3<<UCSZ00)
#define DATA_BIT EIGHT_BIT // USART Data Bit Selection

#define RX_COMPLETE_INTERRUPT (1<<RXCIE0)
#define DATA_REGISTER_EMPTY_INTERRUPT (1<<UDRIE0)

volatile uint8_t USART_TransmitBuffer; // Global Buffer

void USART_Init()
{
    // Set Baud Rate
    UBRR0H = BAUD_PRESCALER >> 8;
    UBRR0L = BAUD_PRESCALER;

    // Set Frame Format
    UCSR0C = ASYNCHRONOUS | PARITY_MODE | STOP_BIT | DATA_BIT;

    // Enable Receiver and Transmitter
    UCSR0B = (1<<RXEN0) | (1<<TXEN0);

    // Enable Global Interrupts
    sei();
}

void USART_TransmitInterrupt(uint8_t Buffer)
{
    USART_TransmitBuffer = Buffer;
    UCSR0B |= DATA_REGISTER_EMPTY_INTERRUPT; // Enables the Interrupt
}

int main()
{
    DDRB |= 1 << 5; // Configuring PB5 / D13 as Output

```

```

71     write (1)
72     {
73         PORTB |= 1<<5; // Writing HIGH to glow LED
74         _delay_ms(500);
75         PORTB &= ~(1<<5); // Writing LOW
76         _delay_ms(500);
77     }
78
79     return 0;
80 }
81
82 ISR(USART_UDRE_vect)
83 {
84     UDR0 = USART_TransmitBuffer;
85     //UCSR0B &= ~DATA_REGISTER_EMPTY_INTERRUPT; // Disables the Interrupt, uncomment
86 }

```

Interrupt Reception

Interrupt reception behaves exactly the same as polling reception but in the case of interrupt reception. The CPU is busy looping an infinite loop and whenever data is received in USART Buffer an interrupt is thrown and the CPU serves it and toggles the LED accordingly. The CPU doesn't have to monitor the USART register bits to check the status of the reception.

```

1  /*
2  * usart.c
3  *
4  * Created : 15-08-2020 09:34:44 PM
5  * Author  : Arnab Kumar Das
6  * Website : www.ArnabKumarDas.com
7  */
8
9  #define F_CPU 16000000UL // Defining the CPU Frequency
10
11 #include <avr/io.h> // Contains all the I/O Register Macros
12 #include <util/delay.h> // Generates a Blocking Delay
13 #include <avr/interrupt.h> // Contains all interrupt vectors
14
15 #define USART_BAUDRATE 9600 // Desired Baud Rate
16 #define BAUD_PRESCALER (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)
17
18 #define ASYNCHRONOUS (0<<UMSEL00) // USART Mode Selection
19
20 #define DISABLED (0<<UPM00)
21 #define EVEN_PARITY (2<<UPM00)
22 #define ODD_PARITY (3<<UPM00)
23 #define PARITY_MODE DISABLED // USART Parity Bit Selection
24
25 #define ONE_BIT (0<<USBS0)
26 #define TWO_BIT (1<<USBS0)
27 #define STOP_BIT ONE_BIT // USART Stop Bit Selection

```

```

32 #define EIGHT_BIT (1<<UCSZ00)
33 #define DATA_BIT  EIGHT_BIT  // USART Data Bit Selection
34
35 #define RX_COMPLETE_INTERRUPT      (1<<RXCIE0)
36 #define DATA_REGISTER_EMPTY_INTERRUPT (1<<UDRIE0)
37
38 volatile uint8_t USART_ReceiveBuffer; // Global Buffer
39
40 void USART_Init()
41 {
42     // Set Baud Rate
43     UBRR0H = BAUD_PRESCALER >> 8;
44     UBRR0L = BAUD_PRESCALER;
45
46     // Set Frame Format
47     UCSR0C = ASYNCHRONOUS | PARITY_MODE | STOP_BIT | DATA_BIT;
48
49     // Enable Receiver and Transmitter
50     UCSR0B = (1<<RXEN0) | (1<<TXEN0);
51
52     //Enable Global Interrupts
53     sei();
54 }
55
56 int main()
57 {
58     DDRB |= 1 << 5; // Configuring PB5 / D13 as Output
59     USART_Init();
60     UCSR0B |= RX_COMPLETE_INTERRUPT;
61     while (1)
62     {
63     }
64     return 0;
65 }
66
67 ISR(USART_RX_vect)
68 {
69     USART_ReceiveBuffer = UDR0;
70     if (USART_ReceiveBuffer == 'a')
71     {
72         PORTB |= 1<<5;    // Writing HIGH to glow LED
73     }
74     else
75     {
76         PORTB &= ~(1<<5); // Writing LOW
77     }
78 }
79 }

```

Interrupt Loopback

The below example works exactly like polling loopback but here the CPU doesn't waste time in checking the status of the USART registers.

```

5  * Author   : Arnab Kumar Das
6  * Website  : www.ArnabKumarDas.com
7  */
8
9  #define F_CPU 16000000UL // Defining the CPU Frequency
10
11 #include <avr/io.h>      // Contains all the I/O Register Macros
12 #include <util/delay.h>  // Generates a Blocking Delay
13 #include <avr/interrupt.h> // Contains all interrupt vectors
14
15 #define USART_BAUDRATE 9600 // Desired Baud Rate
16 #define BAUD_PRESCALER (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)
17
18 #define ASYNCHRONOUS (0<<UMSEL00) // USART Mode Selection
19
20 #define DISABLED (0<<UPM00)
21 #define EVEN_PARITY (2<<UPM00)
22 #define ODD_PARITY (3<<UPM00)
23 #define PARITY_MODE DISABLED // USART Parity Bit Selection
24
25 #define ONE_BIT (0<<USBS0)
26 #define TWO_BIT (1<<USBS0)
27 #define STOP_BIT ONE_BIT // USART Stop Bit Selection
28
29 #define FIVE_BIT (0<<UCSZ00)
30 #define SIX_BIT (1<<UCSZ00)
31 #define SEVEN_BIT (2<<UCSZ00)
32 #define EIGHT_BIT (3<<UCSZ00)
33 #define DATA_BIT EIGHT_BIT // USART Data Bit Selection
34
35 #define RX_COMPLETE_INTERRUPT (1<<RXCIE0)
36 #define DATA_REGISTER_EMPTY_INTERRUPT (1<<UDRIE0)
37
38 volatile uint8_t USART_ReceiveBuffer; // Global Buffer
39
40 void USART_Init()
41 {
42     // Set Baud Rate
43     UBRR0H = BAUD_PRESCALER >> 8;
44     UBRR0L = BAUD_PRESCALER;
45
46     // Set Frame Format
47     UCSR0C = ASYNCHRONOUS | PARITY_MODE | STOP_BIT | DATA_BIT;
48
49     // Enable Receiver and Transmitter
50     UCSR0B = (1<<RXEN0) | (1<<TXEN0);
51
52     // Enable Global Interrupts
53     sei();
54 }
55
56 int main()
57 {
58     USART_Init();
59     UCSR0B |= RX_COMPLETE_INTERRUPT;
60     while (1)
61     {
62

```

Arnab Kumar Das



```
68     USART_ReceiveBuffer = UDR0;  
69     UDR0 = USART_ReceiveBuffer;  
70 }
```

Categories: ARDUINO TUTORIAL – THE INDUSTRIAL AND PROFESSIONAL WAY

**Crazy Engineer**

MAKER - ENGINEER - YOUTUBER

10 Comments

**Kevin Walton** · April 2, 2021 at 3:22 pm

Great work, will give this a go, thank you. A version for bit banging Serial laid out so simply and easy to understand would be a great addition?

[↩ REPLY](#)**Crazy Engineer** · April 3, 2021 at 1:14 am

Thank You, for spending time on the website. Thanks for the Idea. I will implement it soon.

[↩ REPLY](#)**BALAJI** · September 20, 2021 at 12:32 am

Thank you . You have written it in the best way possible.

Arnab Kumar Das



Excellent Work! This is one of the rare blogs written with so much clarity. Thanks for sharing y our knowledge...

[↩ REPLY](#)

Filip · January 8, 2022 at 2:47 am

Thanks a lot for what all the bits of each register do and possible combinations

[↩ REPLY](#)

Takunda Nyamashuka · September 5, 2022 at 9:27 pm

Thanks. Quite informative. Hope you make more programing tutorials for AVR processors, par ticularly atmega328

[↩ REPLY](#)

MQ · December 21, 2022 at 7:21 am

I tested the polling loopback, there was nothing in Arduino IDE, Putty and Data Visualizer.

[↩ REPLY](#)

MQ · December 21, 2022 at 7:25 am

TX polling and TX interrupt were OK.

[↩ REPLY](#)

MQ · December 21, 2022 at 7:30 am

interrupt loopback was the same with polling loopback, there was nothing shown.

[↩ REPLY](#)

MQ · December 26, 2022 at 8:49 am

I tested all the 6 programs, they all works.

[↩ REPLY](#)

Leave a Reply

Name *

Arnab Kumar Das



Website

What's on your mind?

POST COMMENT

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)

Projects

[Arduino Tutorial – The Industrial and Professional Way](#)[DIY Project](#)[Event](#)[Line Follower Robot](#)[Online Courses](#)[Platforms](#)[Arduino](#)[Atmel](#)[Autodesk](#)[Everything ESP](#)[Nordic Semiconductor](#)[Nvidia](#)

Arnab Kumar Das



Raspberry Pi Pico RP2040

Topics

Automotive

Benchmark

Computer Numerical Control (CNC)

Electronics

Home Automation

Internet of Things (IOT)

Printed Circuit Board (PCB)

Programming

Related Posts



[ARDUINO, ARDUINO TUTORIAL – THE INDUSTRIAL AND PROFESSIONAL WAY, ATMEL](#)

Analog Comparator : Arduino / ATmega328p

AVR Analog Comparator / Arduino / ATmega328p Microcontroller | Embedded C Register Level Programming Tutorial | AVR Analog Comparator Tutorial



[ARDUINO, ARDUINO TUTORIAL – THE INDUSTRIAL AND PROFESSIONAL WAY, ATMEL](#)

AVR Interrupt and External Interrupt : Arduino / ATmega328p

AVR Interrupt and External Interrupt / Arduino / ATmega328p Microcontroller | Embedded C Register Level Programming Tutorial | AVR Internal Interrupt and External Interrupt Tutorial



ARDUINO, ARDUINO TUTORIAL – THE INDUSTRIAL AND PROFESSIONAL WAY, ATMEL

AVR System Clock : Arduino / ATmega328p

AVR System Clock Distribution and Control / Arduino / ATmega328p Microcontroller | Embedded C Register Level Programming Tutorial | AVR 8-Bit ATmega328P System Clock Tutorial

Explore Catagory

Arduino Arduino Tutorial – The Industrial and Professional Way Atmel Autodesk Automotive Benchmark Computer Numerical Control (CNC) DIY Project Electronics Event Everything ESP Home Automation Internet of Things (IOT) Line Follower Robot Nordic Semiconductor Nvidia Online Courses OpenCV Platforms Printed Circuit Board (PCB) Programming Raspberry Pi Raspberry Pi Pico RP2040 Topics

Arnab Kumar Das

[DIY LONG BACKUP UPS](#)[THE BIRDS OF THE SHOLA SKY ISLAND](#)[IOT SMART DUSTBIN : ESP32 – SIM800L](#)[LINE FOLLOWER ROBOT : ESP32 – QTR-8RC – PID LINE FOLLOWER ROBOT V1](#)[LINE FOLLOWER ROBOT : ESP8266 – QTR-8RC – PID LINE FOLLOWER ROBOT V1](#)[NVIDIA JETSON AGX XAVIER VS RASPBERRY PI 4 BENCHMARK](#)

Thank You For Visiting

- ♦ Hi, I am Arnab Kumar Das aka. Crazy Engineer.
- ♦ Maker 🛠️ - Professional Engineer 🛠️ - YouTuber 📺
- ♦ I love making projects related to Embedded Software, Electronics, Remote Control Vehicles, CNC, Mechatronics, DIY Projects, Woodworking and more!
- ♦ On this website, you will find Project Blog, Tutorials and DIY Guides.
- ♦ If you find any resource useful please share them with others so that they can also get benefited.
- ♦ You can use the social media buttons to directly share this webpage.
- ♦ *Copyright © 2014-2024 Arnab Kumar Das. All Rights Reserved. This Website uses Analytics and Cookies.*

[ABOUT](#)[ARDUINO TUTORIAL – THE INDUSTRIAL AND PROFESSIONAL WAY](#)[ARNAB KUMAR DAS](#)[BASKET](#)[CHECKOUT](#)[CONTACT](#)[HOME](#)[LATEST POSTS](#)[LINE FOLLOWER ROBOT TUTORIAL](#)[MY ACCOUNT](#)[NRF52840 IOT TUTORIAL](#)[PROJECT BLOG](#)[PROJECT PHOTOS](#)[PROJECT VIDEOS](#)[RELEASE PAGE](#)[STORE](#)[TUTORIALS](#)

Hestia | Developed by Themelsle

Arnab Kumar Das

