Development of a Simple Serial Communication Protocol for Microcontrollers (SSCPM)

Chetan Patil

Electronics & Communication, National Institute of Technology, Surathkal, Karnataka, India
chetankpatil@ieee.org

Abstract- The paper describes development of a simple communication protocol meant for communication between the microcontrollers, separated by a small distance, of the order of a few meters. The following protocol which I have named as Simple Serial Communication Protocol for Microcontrollers (SSCPM) can transfer small amount of data between the microcontrollers at a reasonable speed of up to 4 kbits/sec between 2 microcontrollers, separated by up to 3 or 4 meters. This communication is asynchronous and serial transfer. The main advantage of this protocol is that it does not require any inbuilt or external peripherals, which is the case in other protocols. This protocol requires just a single wire between the 2 microcontrollers and a simple half duplex communication is possible between the two systems at any point of time. The user can just apply the simple code that has been developed in C language so that it becomes independent of the operating platform. The code is just additional functions that are added to the main operating code and can be called at any point of the code to transfer or receive the data from the neighbouring microcontroller. Since the protocol is based on soft transmission of the code, the efficiency of the protocol is partially dependent on the frequency at which the two microcontrollers are working. There is no necessity that the two microcontrollers operate at the same frequency, but evidently higher the operating frequency, greater the data transfer rate between the two. The maximum data transfer rate is governed by the microcontroller with the lower clock frequency. The paper describes how the protocol has been developed so far and tested on the microcontrollers. The paper also gives possible extensions to the development of the protocol which include auto baud rate detection, increasing the range of distance between the two microcontrollers, increasing the data transfer rate and other features. However the protocol described in this paper is a possible alternative to the scenario where simple data transfer without much complexity is essential and does not essentially replace or improve the existing advanced communication protocols. Since this is open source protocol and no hardware requirement, the protocol can be of great encouragement to those who wish to develop applications without licensing or buying other hardware.

Index Terms- Serial communication Protocol, communication protocol for Microcontroller, Atmega32

1. INTRODUCTION

There have been several communication protocols in the embedded systems like RS-232, Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), Controller Area Network (CAN) and many more. But most of these protocols require a prerequisite hardware and sometimes licence for the copyrighted code for the protocol. Hence in simple applications where this amount of complexity in terms of communication as well as rights to use is not essential and only some kind of basic communication is essential, there is a need to develop a simpler interface to overcome this existing complexity. The best alternative would be to develop a new protocol which may not be the fastest or most efficient, but definitely the simplest, cheapest and open source. Making this protocol independent of the hardware platform/requirement would increase the flexibility of the protocol to a great extent. The development of such a protocol has been brought out in this paper which elaborates the details of the protocol, the advantages, the limitations and further possible improvements of the protocol.

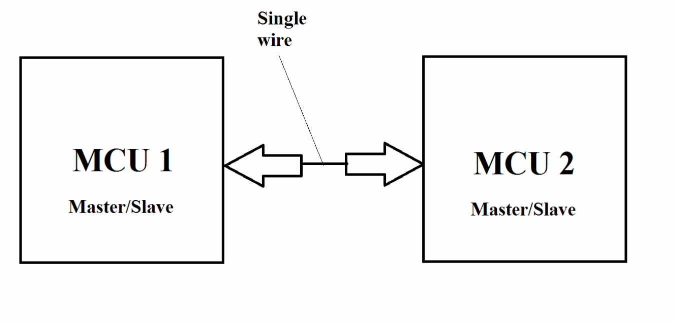
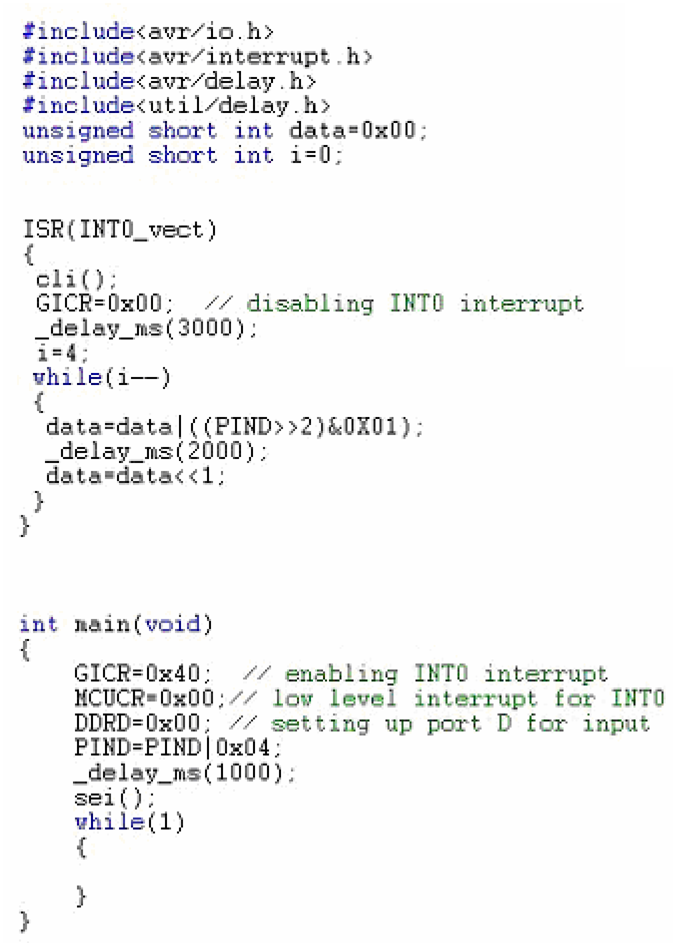


Figure1. Block diagram of SSCPM

2. DETAILS

The development of the protocol has been conducted in several phases. The first phase was to implement a simple data reception code in Atmega 32 using interrupt.
Atmega 32 has 2 external interrupts on PORTD INT0 and INT1 which can be invariably used as either interrupt pins or as input/output pins. The INT0 of PORTD i.e. PD2 was used for this purpose which can be interchangeably used as interrupt pin or IO pin. The advantage of using the interrupt is the time and power saved because of unnecessary polling.



```
#include<avr/io.h>
#include<avr/interrupt.h>
#include<avr/delay.h>
#include<util/delay.h>
unsigned short int data=0x00;
unsigned short int i=0;


ISR(INT0_vect)
{
 cli();
 GICR=0x00;  // disabling INT0 interrupt
 _delay_ms(3000);
 i=4;
 while(i--)
 {
  data=data|((PIND>>2)&0X01);
  _delay_ms(2000);
  data=data<<1;
 }
}


int main(void)
{
    GICR=0x40;  // enabling INT0 interrupt
    MCUCR=0x00;// low level interrupt for INT0
    DDRD=0x00; // setting up port D for input
    PIND=PIND|0x04;
    _delay_ms(1000);
    sei();
    while(1)
    {

    }
}
```

The above code demonstrates the usage of PD2 to receive one byte of data from pin PD2. The data was set using the switch connected to a pull up resistor for testing purpose. In the latter part this data will be provided by another microcontroller which intends to transmit the required data to the host microcontroller. The data supplied by the user through the switch is around 2000ms for each bit and the same is applicable for the start bit. Initially the required interrupts for PD2 is enabled.

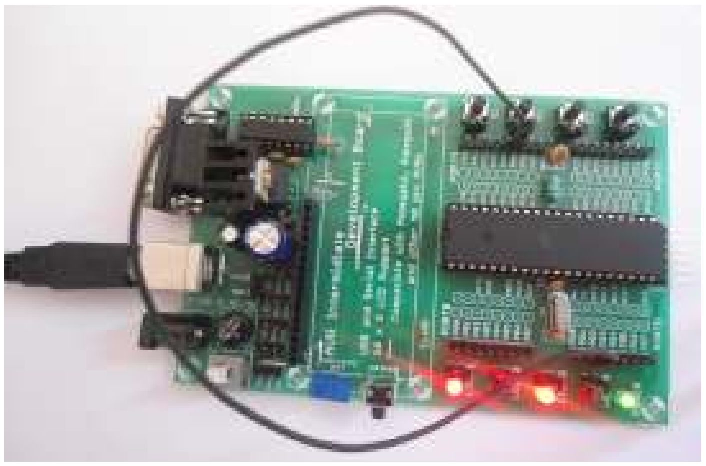


Figure 2. Experimental setup to test data reception

The figure above indicates the test setup of the host microcontroller. The above setup has Atmega32 as the host microcontroller and is used to verify the code and analyse the same. The output pins of the microcontroller are connected to the LEDs to display the most significant 4 bits of the data, for verification purpose. The voltage levels of the Atmega 32 are 5V for high and 0V for low. These levels might for different microcontrollers like for MSP430 series, it is 3.3V for high and 0V for low. However the functionality of the code remains unchanged. Here, PD2 has been enabled as an external interrupt which triggers for low level. This is done by setting the GICR and MCUCR registers as shown in the above code. After the initialization the processor is ready to process the interrupt on request. When the user grounds the PD2 pin for an instant, the ISR is called and the data reception initiates. The microcontroller waits for 3000ms so that it is in the middle of the first data bit and then samples the data. Thereafter the microcontroller waits for 2000ms each time and samples the data. The details of the data pattern diagram will make it clear of how this code works for the input data.

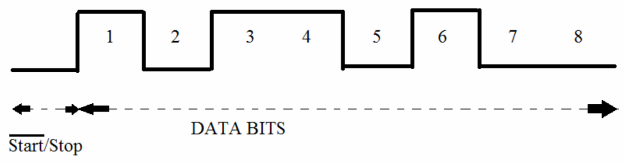


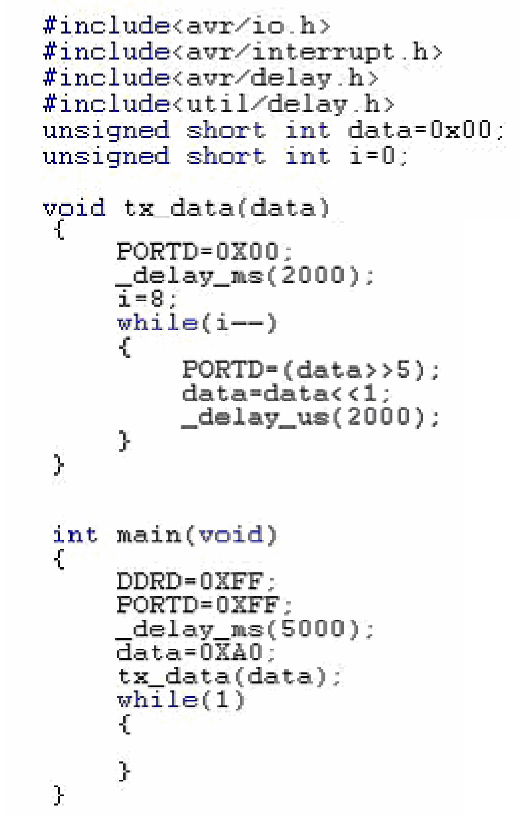
Figure3. Generic data pattern of the SSCPM protocol

The above figure indicates the simplicity of the coding of the data byte that has to be transmitted. Consider that a single data that is 8 bits of data have to be transmitted corresponding to the data byte, then a start byte is appended to the 8 bits which makes the total bits to be transmitted equal to 9 bits for each transmission. The 1st bit corresponds to the start/stop bit. The low indicates the commencement of the transmission of the data since the low level input is used to trigger the INT0 interrupt of the host microcontroller. Soon after this bit is received by the host microcontroller, it disables any further interrupts and starts to receive the data. A global short unsigned integer is used to store the value of the received data and is processed or stored in the memory as per the user's requirement.

The above experiment gave successful results. This however was using input provided by the user i.e. the user fed the data input to the microcontroller. In practice another microcontroller transmits this data and hence the same has to be devised. For the test setup the duration of each bit was 2s. However in practice, the duration of the data bit can be as low as 100 us. The lower the data bit duration, higher is the rate of data transfer. However the lower limit of the data bit duration is governed by the clock frequency of the microcontroller, the details of which will be dealt in the latter part.

Now, the next phase is to generate the data from a microcontroller using the same data pattern diagram. Another microcontroller i.e. Atmega32 is used for this purpose. This Atmega32 is operating at a clock frequency of 16 MHz using an external crystal. The Atmega32 of the host microcontroller is operating at a clock frequency of 1 MHz using internal clock.

This has been deliberately done to check for possible problems in synchronization of the two systems. The following code was used corresponding to transfer data to the host microcontroller. The code was compiled using AVR Studio 4 and the snapshot of the same has been given here. For other microcontroller families, we can use a different platform for compiling our code.



```
#include<avr/io.h>
#include<avr/interrupt.h>
#include<avr/delay.h>
#include<util/delay.h>
unsigned short int data=0x00;
unsigned short int i=0;

void tx_data(data)
 {
     PORTD=0X00;
     _delay_ms(2000);
     i=8;
     while(i--)
     {
         PORTD=(data>>5);
         data=data<<1;
         _delay_us(2000);
     }
 }


 int main(void)
 {
     DDRD=0XFF;
     PORTD=0XFF;
     _delay_ms(5000);
     data=0XA0;
     tx_data(data);
     while(1)
     {

     }
 }
```

The above code shows how the data can be transmitted to the host microcontroller using appropriate timing delays. Since it is the testing phase, the microcontroller is coded only with the data transmission and not data reception. The output of this microcontroller is given to the PD2 of the first (host) microcontroller. It is essential to note that both the microcontrollers might have an independent source of voltage but the grounds of both the systems have to be short. This is the case even in most other common protocols but nevertheless an important point to consider.

It is important to note that though we are declaring the entire PORTD (D0 through D7) to be the output ports while initializing, we are using only the pin PD2 for the transmission of the data. The rest of the pins of the PORTD can be used as per the user's requirement and correspondingly configured as input or output pins. _delay_ms( ) is a function that is available from the library avr/delay.h and can provide fairly accurate delays of the order of milliseconds. Similarly _delay_us( ) is used in the latter part of the development which can give delay of the order of hundreds of microseconds. These functions differ according to the compiling platform and the library used and hence has to suitably replaced in case of different microcontroller.

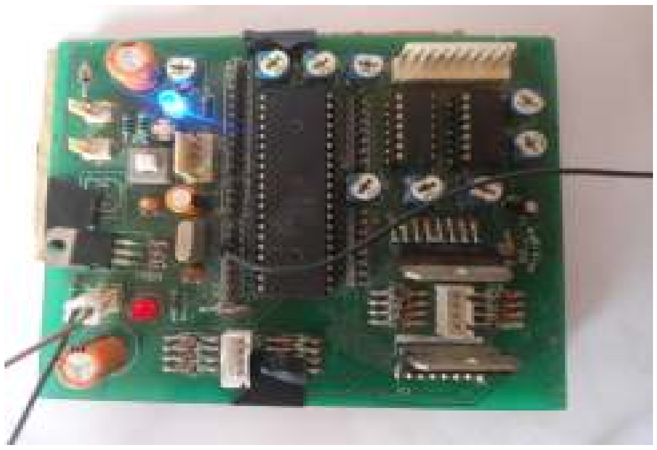


Figure 4. Experimental setup to transmit data to the host

The above figure indicates the setup of another microcontroller which wishes to transmit the data to the host microcontroller and the output of this microcontroller was verified on a LED and the timing of each bit transmitted data bit was verified to be correct. The successful transmission of the data from the above microcontroller paved way to check if the data was intercepted by the host microcontroller appropriately and the experiment proved successful i.e. the data received by the host microcontroller was the same as the transmitted data.

The next phase was to make both the systems completely independent and use the transmission and reception as an interrupt function whenever required. For this the functions depicted in the first and second phase i.e. to transmit and to receive were integrated into one single block of code wherein the user can fit in any application code along with this part of the code without any hindrance to the main application code. Since the data reception is interrupt driven, the microcontroller does not waste time in polling which is the case in some of the communication protocols. More important was to reduce the data bit duration to a great extent so that higher data rates could be verified. For this purpose the duration of each data bit was reduced to 500 us including the start/stop bit. The timing diagram below depicts clearly the implementation of the same.

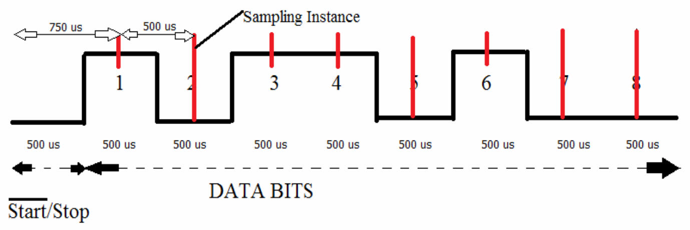


Figure5. Timing details of the data transmitted and received

The above figure shows the timing diagram of the data bits. The start bit is a low for 500 us, which triggers the interrupt on the host microcontroller. The receiving microcontroller senses this interrupt and enters the ISR and performs the data reception function using the same pin PD2 as input pin. After the interrupt is triggered, the microcontroller waits for 750 us so that it samples the mid value of the first bit. This reduces the probability of data sampling error and increases data transfer efficiency. Thereafter it samples the data after every 500 us so that it is sampling the middle of the interval of each data bit.

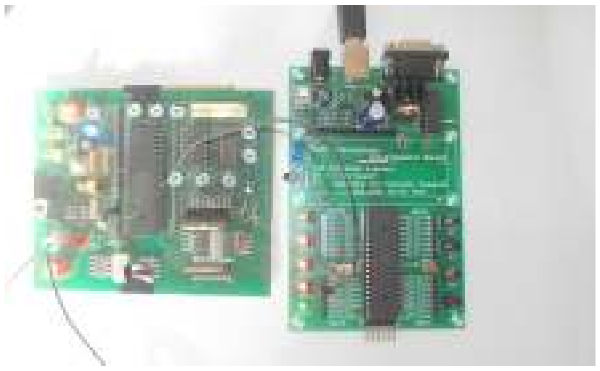


Figure6. Complete setup of the system for demonstration

The figure above shows the final setup of the system containing 2 microcontrollers connected by just a single wire, which is connected to PD2 of each microcontroller. Also, both the systems have different power sources but their grounds are short externally. The data transferred can be verified by displaying on the LEDs of either microcontroller. The setup clearly shows the compactness of the system and independence of any external hardware resources or extra wires. Thus both transmission and reception are possible by the single wire between the two microcontrollers.

The code below explains the exact details of the protocol.

```
#include<avr/io.h>
#include<avr/interrupt.h>
#include<avr/delay.h>
#include<util/delay.h>
unsigned short int data=0x00;
unsigned short int i=0;

ISR(INT0_vect)
{
    cli();   //disable interrupts
 GICR=0x00;  // disabling INT0
 _delay_us(750);
 i=8;
 while(i--)
 {
  data=data|((PIND>>2)&0X01);
  _delay_us(500);
  data=data<<1;
 }

    sei();      // enabling interrupts
    GICR=0x40;  // enabling INT0
}

void tx_data(data)
{
    cli();
    DDRD=0XFF;
    PORTD=0X00;
    _delay_us(500);
    i=8;
    while(i--)
    {
        PORTD=(data>>5);
        data=data<<1;
        _delay_us(500);
    }
    DDRD=0X00;
    sei();
}


int main(void)
{
    GICR=0x40;  // enabling INT0
    MCUCR=0x00;// low level interrupt for INT0
    DDRD=0x00; // setting up port D for input
    PIND=PIND|0x0F;
    _delay_ms(1000);//setup time before enabling interrupt
    sei(); //enable interrupt
    while(1)
    {

        //user appplication code here

    }
}
```

The above code depicts the exact working of the code. It would be essential to note that when the microcontroller receives the interrupt i.e. to receive the data, it enters the ISR and first disables any further interrupts since while data transfer is taking place, the microcontroller might mistake it for interrupt and cause complication in interrupt servicing. After the reception of all the 8 data bits, the microcontroller enables the interrupt on the pin PD2 again so that any further request for data reception can be honoured correspondingly. Another important feature to note is that, the pin PD2, serves as input pin all the while, but only during data transmission it changes into an output pin and transmits all the data bits and then after the completion of all the data transmission, it converts to an input pin again. This is the main reason for the flexibility available in this protocol. The code clearly depicts the compactness and flexibility of the data transfer system in case of the SSCPM.

3. CONCLUSION

The tests conducted on the system proved to be successful. The microcontrollers under consideration were

1. Atmega 32 operating at 1MHz internal clock
2. Atmega 32 operating at 16 MHz external clock

Though the microcontrollers were operating at different frequency and also not synchronized with clock, there was no data error even at data rates of up to 2 kBps and efficiency proved to be almost 100%. Further increase in the data transfer rate gave an error of around 15%, which is very evident since the data bit duration approaches the clock frequency closer and closer.

The main advantages of the newly developed protocol are:

1. Simplicity in the communication channel as this protocol requires just one wire i.e. medium of transmission/reception and can render to both reception and transmission
2. No hardware resources are required by the microcontroller which is not the case in many protocols. Thus a microcontroller with no specific peripherals can use this medium of communication
3. No licensing/permission for using the protocol. In case of most of other protocols, a specified prices has to be paid as licence/fees
4. No polling, since the communication is interrupt driven and hence considerable amount of power is saved
5. Can be extended to any microcontroller family irrespective of RISC or CISC
6. Platform independent since the code is in C language (high level language)
7. Very effective where small amount of data is required to be transferred
8. Works very efficiently where distance between the two systems is not very large

Some disadvantages of the protocol are:

1. Data transfer rate is dependent on operating frequency of the slowest microcontroller; hence the data rate can drop considerably with decrease in the operating frequency of the microcontroller
2. Data error may increase considerably with larger amount of data to be transferred
3. The protocol has not been tested for larger distance and hence can be a barrier in some cases
4. This is a half duplex communication and sometimes pose a problem where in simultaneous read/write becomes essential
5. Dependant on software/timer delays and so might result in inaccuracy if the coding is not done efficiently. However at any point of time, this issue can be resolved by increasing the duration of the data bits

Possible improvement in the protocol

1. The protocol discussed as per the above code requires that the both the transmitter and the receiver are aware of the bit rate so that the delays can be set appropriately by each microcontroller. However auto baud rate detection can be implemented to resolve this issue. For this there are 2 alternatives, first is to transmit a series of ones and zeros alternatively and the receiving microcontroller can measure the average bit duration of each bit using level triggered timers and is a one-time. The second alternative is to transfer an additional high bit after the start bit during each transmission so that the timer of the receiving microcontroller can calculate the baud rate and the main advantage is that the baud rate can be changed by the transmitting microcontroller at any time.
2. The data register illustrated in the code is a single 8 bit register. However it would be advantageous to declare an array of 8 bit memory locations as data buffer so that after receiving each byte the data can be loaded into this data buffer. This data buffer can prove to be useful when the transmitter transmits a sequence of data continuously to the receiving microcontroller. Similarly an output buffer can also be designated to transmit a sequence of data.
3. The signals considered in the experiment are 5V for high and 0V for low as per the Atmega 32 levels. However the range of distance of communication can be increased to considerable extent by increasing the voltage difference level using amplifiers or level shifter. However, this can cause the cost to go up but this is not essential most of the times since the protocol is meant for communication between systems which are at a close distance.
4. The model has been explained with only 2 systems in consideration; however the same can be extended to multiple systems using the same single wire to interface with all the systems. For this the transmitting microcontroller would first send the byte containing the address of the particular system and then the particular system would authorize itself to be the receiver and receive the corresponding data.

With the development of the technology, better and more efficient techniques have been developed to break the barriers of communication. However it is equally important to look for possible simpler solutions which can be applied to systems where highly complex form of communication systems are unnecessary and prove to be redundant. This protocol is a step in that direction and further development in the protocol can be utilized as a reliable and simple to use communication protocol in all future embedded systems.

REFERENCES

[1] L.R. Thebaud, "Systems and Methods with Identity Verification by Comparison and Interpretation of Skin Patterns Such as Fingerprints," US Patent No. 5,909,501, 1999.
[2] Fang Yi-yuan; Chen Xue-jun; Design and Simulation of UART Serial Communication Module Based on VHDL, Intelligent Systems and Applications (ISA), 2011 3rd International Workshop
[3] Roxel D.E., Serial *Interfaces for Minicomputers*, IEEE transactions on Computers