

Introduction to Embedded System Design
Dhanajay V. Gadre (NSUT)
Indian Institute of Technology Delhi
Lecture 33

Serial Communication Protocol, USCI Module in MSP430

Hello and welcome to a new session for this online course on Introduction to Embedded System Design, I am your instructor Dhanan V.Gadre. In this session, we are going to talk about one of the peripheral functions of a microcontroller namely, communicating with the outside world. Now, if you want to communicate a microcontroller wants to communicate with the outside world, there are many ways of communication, but the common methods require you to send the data and receive data in a serial fashion.


Why? Because it minimizes the number of connections that you would require for communication, the data in a microcontroller or in any computer is in the form of bytes or words and so, if you want to transmit a byte of data, otherwise you would require eight bits that is eight wires, one for each of the bits and then ground and some other handshakes signals as they call it.

So, the number of wires that would be required to connect between two devices would grow too much and this is not preferred and therefore, any communication that normally happens is in the form of a serial form that is the byte or word that you want to transmit is serialized one bit at a time and transmitted similarly it is received in that fashion. So, in this lecture, we are going to look at the various methods of communication with the outside world meaning with other devices which have similar capabilities of communication. How do we get our MSP430 microcontroller to communicate with them.

(Refer Slide Time: 02:03)

Communication Protocols

- A Communication protocol is a set of rules and formal descriptions defined for communicating information between two or more devices.
- In other words, they describe the format and rate in which the data will be sent from sender to receiver.





Now, this is defined as communication protocols. Communication protocol is simply a set of rules and formal descriptions, which are defined for communicating information between two devices. In other words, this communication protocol defines the format of the data that will be sent because you are sending parallel, you have parallel data, but you want to send it serially. So, you have to decide what is the format of that data and at what rate are you sending each of these bits. A communication protocol basically defines these rules.

(Refer Slide Time: 02:37)

Data Transmission Modes

- **Simplex:** Communication can occur only in one direction from Device A to Device B only.
- **Half Duplex:** Data transmission can occur between Device A and Device B, but only one at a time.
- **Full Duplex:** Data transmission can occur in both direction between Device A and B simultaneously.



Now, when we communicate between two devices, let me draw them here. For example, if I want to communicate between device A and device B and this could be electronic devices, humans and so on and so forth. Let us say, this is A and B and you want to communicate between them.

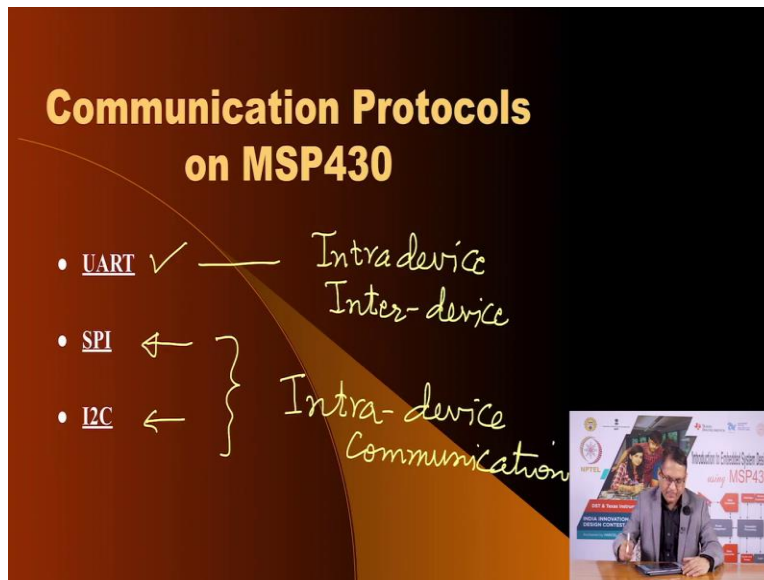
If you only transmit between, from A to B with no possibility of B responding to you this would be called simplex mode of communication. Then you could have half duplex in which yes it is possible for A to communicate information to B as well as for B to send back information to A, but not at the same time this is called half duplex and when you have full duplex, which means, when you transmit from A to B and B to A and these two transmissions can happen simultaneously this is called full duplex.

In real life in normal life we have examples for example, simplex communication happens if you are talking to somebody and there is no mechanism. For example, radio transmission, broadcast transmission suppose you are listening to FM channel, the transmitter is transmitting and you only receive.

On the other hand half duplex is what normally happens using a walkie-talkie. There is a button called push to talk, the one who pushes that button is able to transmit the rest everybody can receive and then when they want to talk, they push their push to talk button, then they talk and the receiver rest of the receivers can receive information and full duplex is what happens when you make a telephone call. Both the people can talk at the same time.

Of course, in the case of humans, both people talking at the same time, they may not listen to what the other person is saying. But that is a different matter. So, there are three modes of communication simplex, half duplex and full duplex.

(Refer Slide Time: 04:38)



Now on our MSP430 we have several communication protocols, several serial communication protocols and the most common of them is the UART. It stands for universal asynchronous receiver and transmitter. Then we also have a mechanism for higher speed of communication this is called Serial Peripheral Interface and the third method is called I square C, which is inter IC communication.

We can think of these you may recall that in, when we were talking about the six box model for embedded systems and we considered the various communication methods, we classified the communication methods in two ways, one was called communicating within a device and we call that intra device communication and then we had another mechanism called inter device communication that is, if there are two individual independent entities, if they want to communicate, what kind of protocols are suitable for communicating between two independent devices and we call that inter device communication.

Now, you are typically in suitable not only for intra device that is within a device. If there are devices within a, if there are building blocks within the device, they want to communicate serially you could use a UART, it is suitable for that and it is also suitable for communicating between two independent devices.

So, it is also suitable for inter device communication. SPI and I square C on the other hand are most suitable for intra device communication, intra device communication and we will see what

all, where all these two protocols are used. Now UART is the most common method of communication between a microcontroller and another microcontroller or a microcontroller and a desktop computer or many other examples.

In fact, you are using UART every time you connect your MSP 430 lunch box to your desktop computer it is using although on the desktop side it is using a USB interface but on the microcontroller side the USB protocol is converted into a UART protocol and appropriate pins of MSP430 which are able to communicate using the UART you make that connection and that is how the program is being downloaded from the desktop computer into the memory of the MSP 430 lunchbox microcontroller.

(Refer Slide Time: 07:23)

UART (Universal Asynchronous Receiver/Transmitter)

- Dedicated hardware meant for Serial Communication.
- Baud Rate of data transmission is known beforehand, which is defined by the user.
- UART data is organized into '*packets*'. Each packet contains 1 start bit, 5 to 9 data bits (depending on the UART settings), an optional *parity* bit, and 1 or 2 stop bits. Example format 9600 8N1.

UART uses dedicated hardware and it is communicating in a serial fashion. When you are transmitting data serially, you had to decide what is the duration of each data because each bit. Because we are saying the communication type is asynchronous, which means there is no accompanying clock with the data that is being sent.

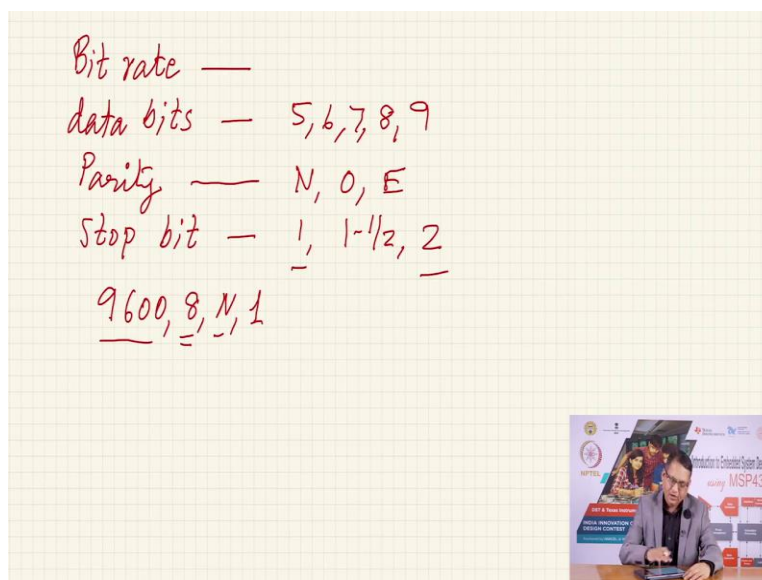
And therefore, some other method of knowing when one bit ends and the other bit starts, the next bit starts has to be decided and one method of doing that is to finalize and agree upon a certain time for which each bit will last and this rate, the resultant rate is called bit rate or baud rate and the user has to define this before any communication can happen and of course, both the devices must agree on that bit rate or baud rate.

Also, since you are sending data serially for historical reasons, in the olden times, you did not want to send eight bits of data. In fact, if you recall, ASCII standard is defined for seven bits, the original ASCII standard is seven bits. So, if you wanted to send ASCII information, you only want to send seven bits.

But these days you want to send entire eight bits of data and in fact, ASCII there is a advanced additional ASCII format which allows for eight bits and so you may choose to decide how many data bits will be sent in each transmission and together with the rest of the bits, we call this one data packet and this consists of 1 start bit between 5 to 9 data bits and then you can also have a parity bit, the parity bit allows you to detect whether in the transmission of the data bits if there has been any error because of noise.

It does not tell you which bit went wrong, it will only tell you if any bit went wrong and it can only tell if one bit went wrong, if two bits went wrong the parity error may not be able to tell you that possibility and then after the parity bit, you can add one or two stop bits and there is a method of specifying the format of your communication as an example, you could say that we are going to communicate at 96008 and one I will explain what this, what these numbers mean.

(Refer Slide Time: 09:54)



So, the first value is the bit rate or baud rate as they call it, often. Then the second value is data bits, number of data bits that you want to transmit, then the information about parity and the last is the stop bits. There is no option of specifying the number of start bits. The information

regarding start bit is fixed and it is only one start bit per packet transmission and so if I say my format is 9600, 8, N, 1, that means my bit rate is 9600 bits per second, I want to transmit eight data bits per transmission. I do not want to have any parity.

The options are, so betrayed we will see, what are the various options standard bit rates are available, data bits can be 5, 6, 7, 8 and in some microcontrollers it also offers nine bits of data. The parity bit there are three options no parity, odd parity or even parity and for stop bit, historically, it offers three options one stop bit, one and a half stop bit or two stop bits.

In the case of MSP 430 you have the options of one or two and you can choose whether you want to have parity or no parity at all and if you want to have a parity whether it is odd parity you want to include or even parity.

(Refer Slide Time: 11:28)

The slide displays a list of standard baud rates: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 11500. A red bracket groups the rates from 300 to 11500. A red arrow points from the 9600 rate to a handwritten calculation:
$$\text{Bit Time} = \frac{1}{9600} = 104 \mu\text{s}$$

In the bottom right corner, there is a small video inset showing a presenter in a grey jacket sitting at a desk with a laptop. Behind him is a banner for 'Production & Controller System Design using MSP430' with logos for 'MSP430' and 'Texas Instruments'.

Here is a list of standard baud rates. Now the meaning of standard baud rates means that if you have a existing device, let us say a standard laptop or desktop, which would have offer certain serial communication protocols, it would offer the communication rates which can be from one of these numbers.

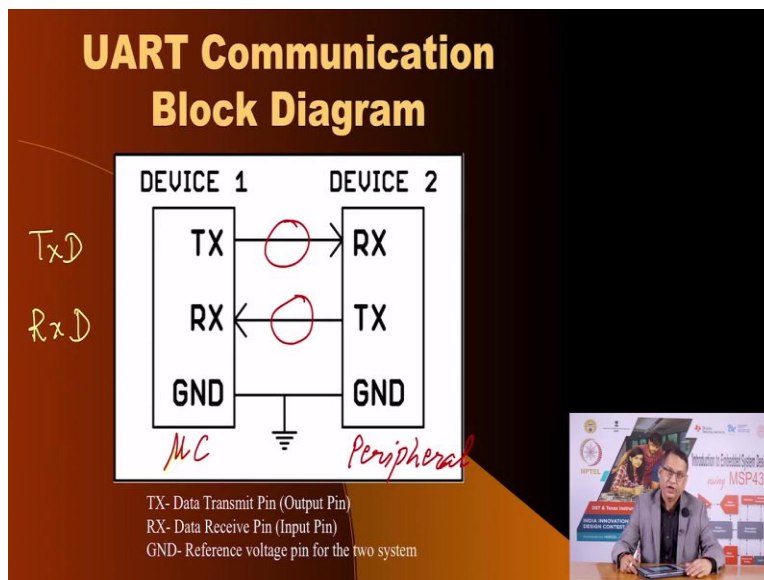
Of course, if both the devices that want to communicate using the UART protocol and both the devices are your devices, you can choose arbitrary rates of communication, there is nothing wrong with that, but if you want to communicate with the existing devices, then these are the

standard baud rates that you must choose from 300 bits per second, 600 bits per second so on and so forth.

If you see there is a factor of two as you go down this list the number of data bits per second they double and the maximum bit rate you can get is there is a zero missing here 115000. So, these are the standard baud rates that are available and these baud rates because, so if I say 9600 bits per second that means each bit is going to last for this duration, the value of bit time, bit time is equal to one by the bit rate, so it is 9600 and in this case this becomes 104 microseconds.

Now because there is a timing information involved, there has to be a mechanism for the microcontroller to generate this timing information, so that it can change each bit after 104 microseconds and of course the microcontroller has timing information in the form of various clock signals. In the case of our MSP430, we have A clock, we have SM clock and we have M clock and we can choose which of these clock signals will provide the timing information necessary for serial communication.

(Refer Slide Time: 13:44)



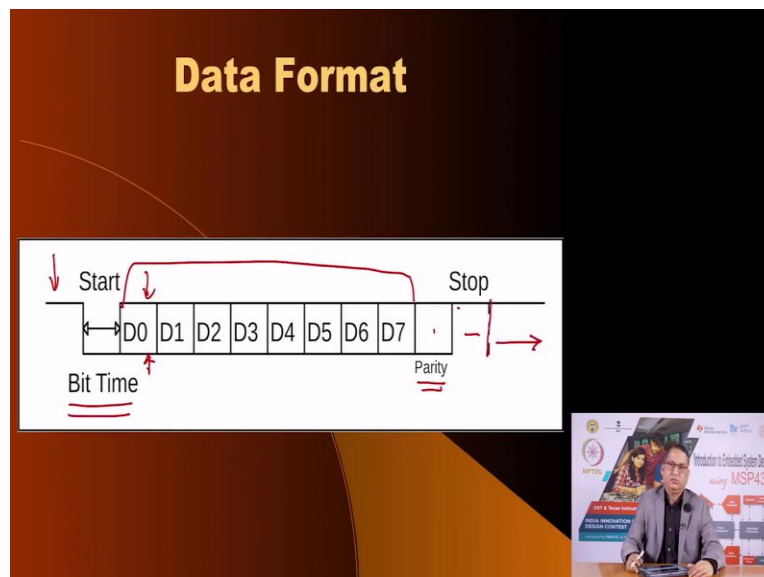
Now, the basic block diagram of communication between two devices is illustrated in this diagram, you have device one and your device two, the two pins of serial communication are called TX or oftentimes they are also called Tx D. I could also call this Tx D and the other pin for receiving is called Rx D.

So, you have to connect the transmitting bit to the receiving bit of the second device and vice versa and of course, you need a common ground connection. So, these are the three pins that are required for, minimum three pins that are required for communicating between two devices, whether one device could be a microcontroller is fine. It could be a microcontroller and the other device could be some peripheral or you could have both the devices as microcontrollers you, it could be two independent devices, they would want to communicate with each other, it could use the UART protocol.

Now, if the distance between these two devices is small, of the order of few inches, then you can simply take the transmitting pin of one device and connect it to the receiving pin of the other device and all this would work fine. Of course, the logic levels on these pins must match each other which means if this device is powered at 5 volts, then the second device should also be powered at 5 volts, if this device is powered at 3.3 the other should also be power at 3.3.

Now as the distance between the two devices increases and if the two devices are not being operated at a common or identical supply voltages, then making such a connection may not result in reliable communication and therefore, some sort of line drivers that is some sort of amplifiers on these lines will be required, so that the distance between the two devices does not induce noise in the data that is being transmitted and based on the type of line drivers that you include, will lead to different types of protocols. We will see what are those protocols that are available.

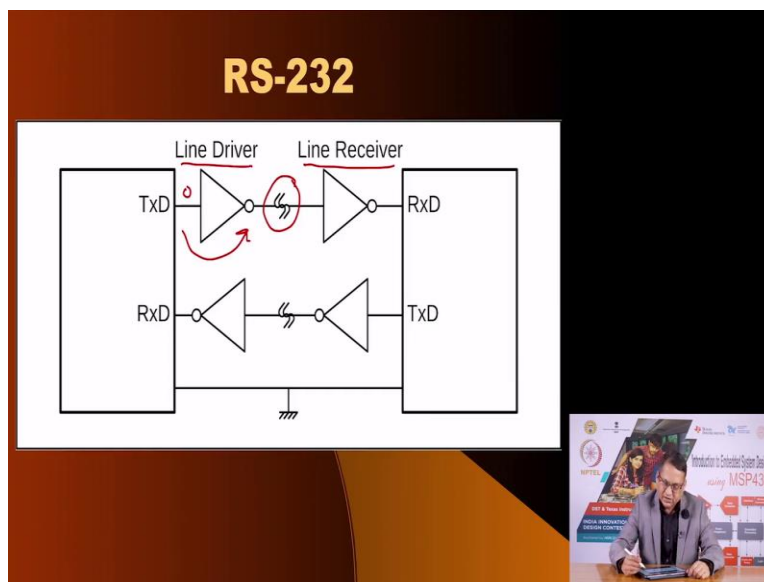
(Refer Slide Time: 15:47)



Now, when you transmit the data as I mentioned, at the beginning of the transmission, the logic on your output pin is logic one and when the device is ready to transmit it will take the TxD pin, it will take it low for a certain amount of time equal to bit time and then depending upon the value of the actual data bit, bit 0 that bit can go high or it can remain low depending upon the value of D0 and so on and so forth and you are going to transmit D0 D1 D2, all the way till the last bit and then depending upon the parity bit that is the parity of these bits can be stuffed can be added at the end of this.

After that you will add a stop bid which is offer logic high and then when the stop bit end you can start a new transmission. This is the format of data that is serially transmitted between one device and will be received by the second device. As I mentioned, we need line drivers so that data is effectively and noiselessly communicated from one end to the other end.

(Refer Slide Time: 17:06)




And one protocol which is very popular is called RS232. It requires line driver like this on this side and this here simply means that the distance between the transmitter and receiver can be large and then you need to have a line receiver IC or line receiver amplifier. As you see here, this line driver also inverts the logic that is if you send 0 here, the value will be inverted. But the voltage levels at which the RS232 line drivers work they operate as follows.

(Refer Slide Time: 17:37)

Input	RS232 o/p
0	+V $3V < V < 15V$
1	-V

3V ±5V
 ±10V

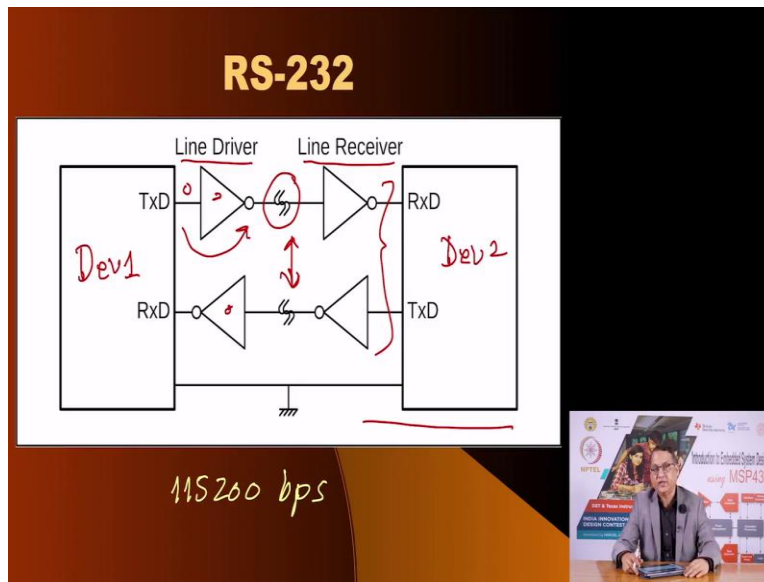
MAX232



If the input is 0, then the output RS232 is plus V some voltage V. If the input is 1, then it is minus V and what are these voltages? The value of V could be anything greater than 3 volts and less than 15 volts which means, this line driver and similarly this line receiver has to have a separate power supply.

For example, let me give an example. Your microcontroller can be working at 3 volts and the RS232 line transmitter and line driver and receiver can work at plus minus 5 volts or plus minus 10 volt. Now, how do you generate plus minus 5 volt from 3 volts? Either you supply individual voltages by polar voltages have this value or many of these line drivers have a mechanism have a built in DC to DC converter, which converts the supply voltage of 3 volt to 5 or 10 volts standard line driver ICs, transmitters and receivers are available. For example, there is IC called Max 232 this will convert the TTL or 3 volt logic levels into RS232 protocol.

(Refer Slide Time: 19:36)



Input	RS232 o/p
0	+V
1	-V

$3V < |V| < 15V$

3V $\pm 5V$
 $\pm 10V$

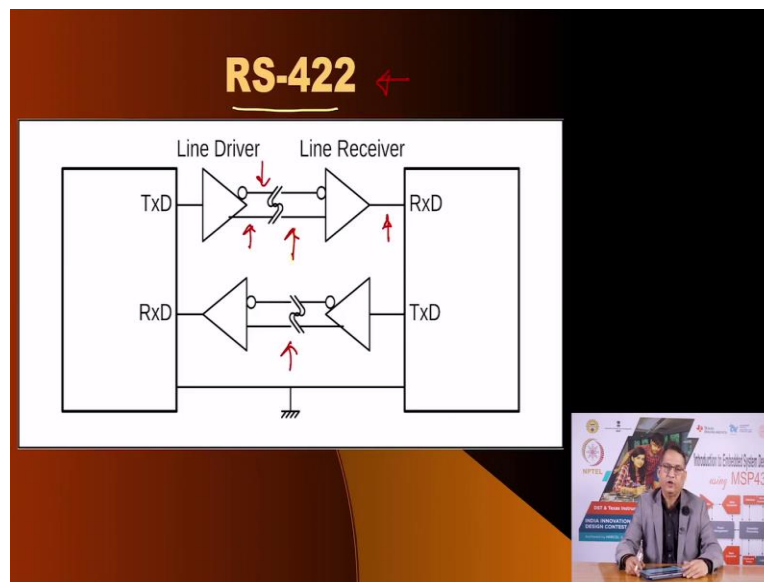
MAX232

On the receiving side it will receive the bipolar voltages and it will invert the logic and again it will give you the same logic level as required by this device. So, this is let us say this is device one and this is device two. Remember that this transmitter and receiver here are being, are connected to this device, so they may share the supply voltage from device to which could be 5 volts this could be 3 volts, this driver and receiver is connected very close to the device one and it may have a different supply voltage, it does not matter.

What matters is that the voltages on the RS232 signals will conform to these levels and they may be different and yet these two devices, these two line drivers and receivers will convert the RS232 levels into the logic levels corresponding to the actual device on their side.

Using this protocol, using this method you can have standard baud rates going up to 115200 that is 115200 bits per second. You can have the distance between the transmitter and receiver to few 10s or 100s of meters. Now as the distance between the transmitter and receiver increases the possibility of noise corrupting the data increases and one method of reducing that noise is to have a different kind of line driver and receiver one that is called as differential transmitter and receivers. It leads to a different protocol and that protocol is called RS422. But remember our UART here is still the same, it also has the same data format voltages are also the same.

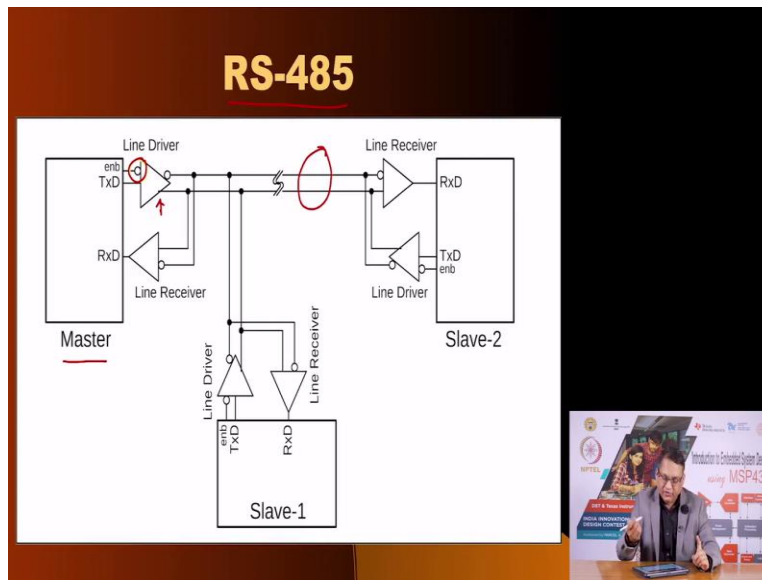
(Refer Slide Time: 21:36)



But the voltages that are available on these lines the serial communication lines now conform to RS422 format here. The line driver has a true and a inverted value and these are again received by the receiver it takes the difference of these two voltages and any common mode noise that may have corrupted the two data lines would be subtracted out and you would get your actual value back here and so, these differential, these are called differential transmitters and receivers, they are able to communicate between two devices for a longer distance and a higher bit rate.

Now, in all these discussions till now, we have seen that there is only one transmitter and one receiver that is, there is a transmitter on one end and receiver on the other and the other side has a transmitter and receiver. This allows for up to full duplex communication and what if you wanted to communicate between more than two devices like here we have device one and device two, what if I wanted to have one more device using these protocols it is not possible.

(Refer Slide Time: 22:56)



But there is a modification and there is a new method, a different method called RS 485. It is derivative or the RS422, except if you see here now, the line driver not only has a differential output, but it also has a enable pin. You see here it has a enable pin. Now if the pin is not enabled, the line driver would tristate the outputs meaning it would disconnect it from this cable. Similarly, the receiver has these two inputs connected and of course, if the transmitter is not transmitting anything, it will receive the value from somewhere else.

Now, using this enable mechanism, you can connect multiple devices. Now, the moment there are multiple devices, question arises as to who would be transmitting, because we still have a common set of wires and therefore, at any time only one person can transmit, if more than one device transmits it would lead to what is called as data corruption, it will lead to collision of data and you do not want collision of data.

And therefore one method of saving that possibility is to have a master slave protocol, where there is a master, the master would decide which slave can communicate back and this method is usually what we have in the classroom, the teacher is the master and all the students are the slaves, the teacher can allow a particular student to ask a question and till the permission comes, nobody can ask a question and using this protocol, you can have multiple devices connected to each other to create a sort of network of microcontrollers.

You may recall that in India there is a very popular program called ‘Kaun Banega Crorepati’, ‘Who Wants to Be a Millionaire’ international version of that program. In that, you see many of these participants, they offer a vote they are able to vote based on the person, requirement of the person on the hot seat.

How did they communicate their vote? if you could implement such a network using RS485 protocol where each of the viewer or each of the audience in the hall would have a keypad on their seat which has four pins four keys, ABCD to offer the options and all these chairs could be networked using RS485 protocol and it could be received at the desktop computer of the host to get the information of the votes and a simple network made out of RS485 can easily satisfy the requirements of audience voting, audience pole in the case of such a requirement.

(Refer Slide Time: 25:43)

Steps of Transmission

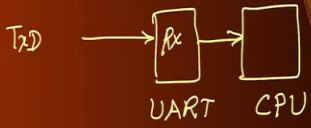
- UART module receives data to be sent from CPU in parallel form.
- UART module then adds start, stop and parity bits accordingly.
- Data is sent serially from Tx pin over the pin at predefined rate.

Now what are the steps of transmission, when you want to transmit data, you have a microcontroller that is the CPU and you have the UART. Of course in the microcontroller, this is your CPU and this is your UART. Both of these sub these building blocks are integrated in the same chip, but the CPU will transmit information of the data that it wants to transmit from the CPU to the UART. The UART will add the start bit, then it will serial it will send the start bit it will serialize the received data at the end of it, it may add a parity bit as well as top bit and this data will be sent on the Tx line TxD.


(Refer Slide Time: 26:29)

Steps of Reception

- Receiver receives data in form of packets and performs error checks using start, stop and parity bits.
- It then discards start, stop and parity bits, keeping only the data bits.
- Sends the data to CPU over data bus parallelly.



The diagram shows a hand-drawn flow: 'Tx' with an arrow pointing to a box labeled 'Rx' with 'UART' written below it. Another arrow points from the 'Rx' box to a box labeled 'CPU'.



In the case of reception, you reverse the format now the UART is going to receive data from transmitter here, it is going to come in to the Rx pin, Rx pin and this will communicate to the CPU. This is your CPU, this is your UART. The UART will receive the start bit it will start receiving the data bits and then the parity and the stop bit, it will strip off the start bit, parity bet, and stop bit.

Of course, it will use the parity bit received from the transmitter and compare it with a local computed parity bit, if they to match that means no error has been encountered in the transmission and then it will remove the data bits and present them to the CPU. This is how the reception happens.

(Refer Slide Time: 27:21)

SPI (Serial Peripheral Interface)

- A Full Duplex Synchronous Serial Communication Protocol
- Master can talk to many slaves but a slave can talk to only one master




So, this is as far as the UART protocol is concerned. The second protocol that is available on MSP430 is this Serial Peripheral Interface. This is a full duplex synchronous, the previous protocol was asynchronous, which means there was no separate clock that was being sent. But here you have to send a separate clock and you it is a, you can create a network of devices using SPI and you have a master and you can have one slave or multiple slaves and you can have full duplex communication.

(Refer Slide Time: 27:53)

Block Diagram

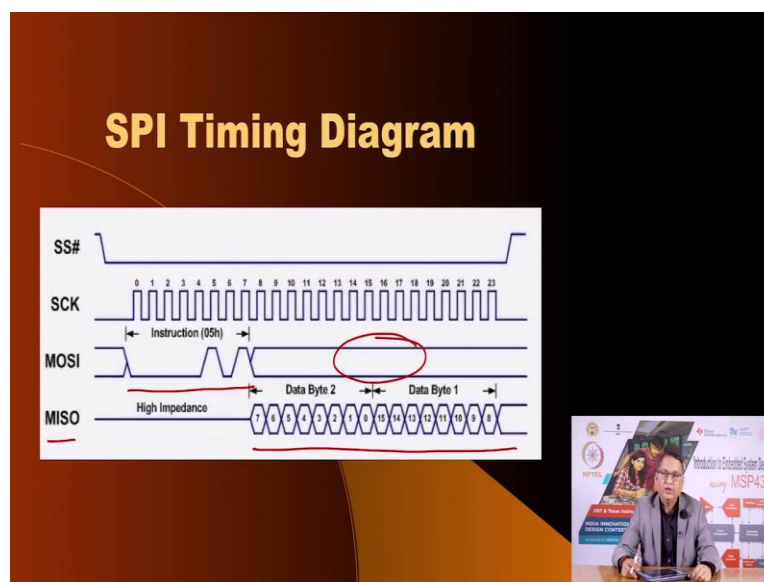
MASTER	SLAVE
MOSI	MOSI
MISO	MISO
SCK	SCK
CS/SS	CS/SS

MOSI- Master Out Slave In
MISO- Master In Slave Out
SCK- Serial Clock
CS/SS- Chip Select/ Slave Select



This is the signal connections between a master and a slave or you could have multiple slaves. You have MOSI that stands for ‘Master Out Slave In’ this is the receiver part master out and slave in is on the slave is receiving pin. ‘Master In Slave Out’ is a receiving pin on Master, but it is a transmitting pin on the slave. Then you have a ‘Serial Clock’ which goes from the master to the slave and you have a ‘Chip Select’ which enables a particular slave. So, if you have multiple slaves, you would need multiple chip select devices, so that you can select which slave you want to communicate with at any given point of time.

(Refer Slide Time: 28:33)



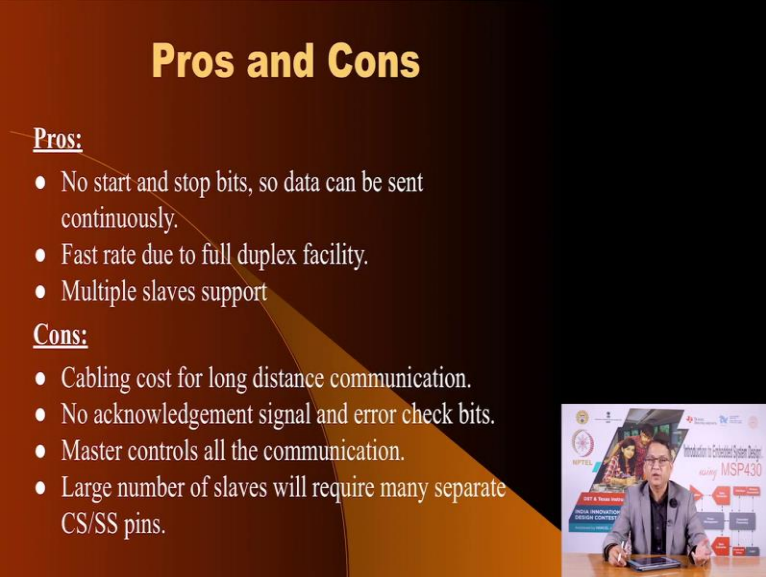
The timing diagram of the various signals that is chip select serial clock, MOSI, MISO is shown here. It starts from the master and then once the master has sent the data here, this is the data being sent by the master the MISO at that time will be in high impedance state and once the data has been transmitted, it may expect the slave to respond back and this is the data received from the slave.

In this way, you can have communication, full duplex communication and this can overlap you can also have data being transmitted from the master to the slave at that time, synchronized with the clock signal and using this one or multiple devices can communicate with a MSP430 are similar microcontrollers.

The clock rate is quite high, it can go up to 4 megahertz or even more and therefore, the data rate of communication between two devices using the SPI protocol is much higher compared to the

data rate that you get on UART. But remember, SPI is not used for communicating between two devices. It is used between for communication between within the device between the microcontroller and a peripheral device.

(Refer Slide Time: 29:52)



Pros and Cons

Pros:

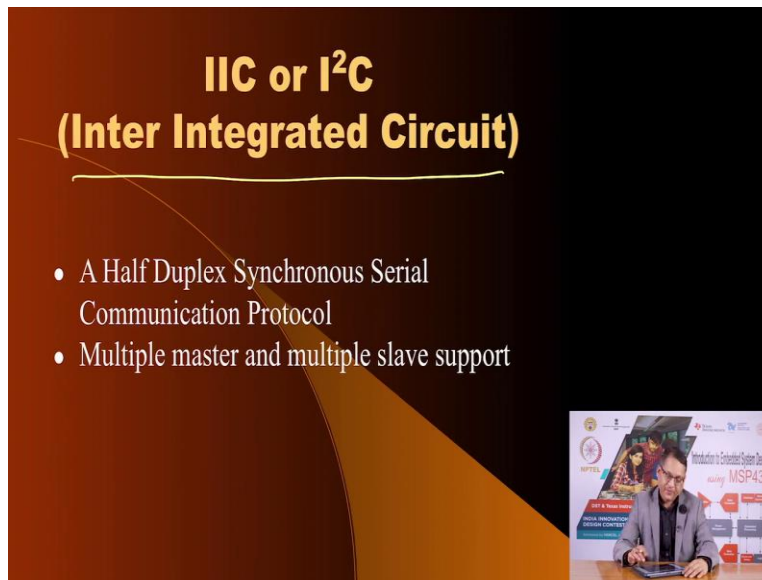
- No start and stop bits, so data can be sent continuously.
- Fast rate due to full duplex facility.
- Multiple slaves support

Cons:

- Cabling cost for long distance communication.
- No acknowledgement signal and error check bits.
- Master controls all the communication.
- Large number of slaves will require many separate CS/SS pins.

Now the advantages and disadvantages of this protocol are listed here. Since there is no start and stop bits the effective number of bits is, you know does not have that over it. So, your data rate will be higher. You can support multiple slaves. But you cannot use this for long distance communication between two devices and if you have multiple slaves, it will require multiple chip select pins from the master that is the only, you know disadvantage of this method.

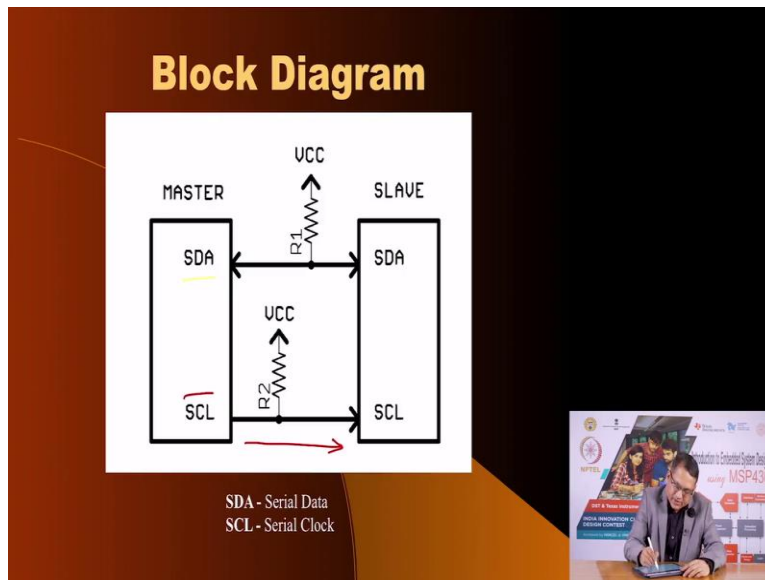
(Refer Slide Time: 30:23)



The third method of communication that MSP430 offers is called inter IC communication, as you see here, is called I square C. It is a half duplex communication protocol and you can connect multiple devices multiple slaves to the microcontroller and you can collect up to seven additional devices. That is the beauty of this protocol.

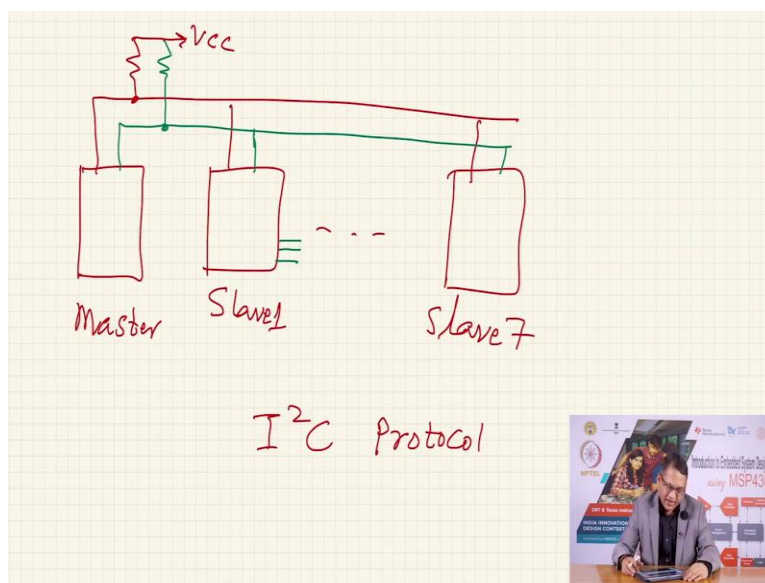
The communication rate is not as high as you would get on the SPI but higher than what you can achieve on the UART and therefore this is very, very preferred when you have to connect multiple devices and you do not have many pins because on SPI bus you can connect many devices. But as you increase the number of devices the number of chip select pins will linearly increase. That is not the case with I square C.

(Refer Slide Time: 31:13)



Here is what is required you only have two pins, one is called serial data and the second is called serial clock, The clock starts from the master goes to the slave and the data is bidirectional, you the master starts the communication the slave receives it, and then if required the slave can respond. Now, if you want to have multiple such devices, you can connect them to the same pins in the following way.

(Refer Slide Time: 31:40)

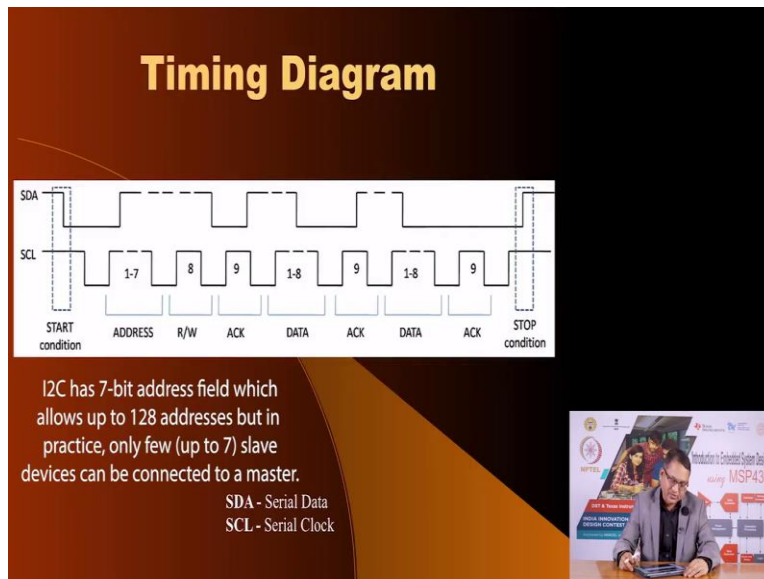


Here is your master, this is your master and we are talking of I square C protocol. You could have slave 1 and up to slaves 7 and using only two wires. Let me draw the second wire with a

different color you can have and you would, of course need a pull up resistor on both of these wires.

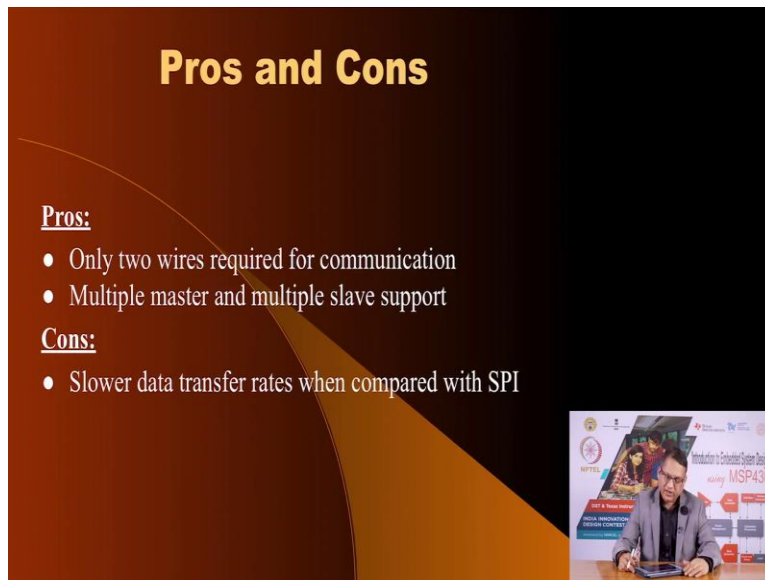
Now, how does a particular slave get selected? Well, each of the slaves has an address, Master address 0 and then you have slave address from 1 to 7 and the transmission includes the address of the slave and how does the slave know its address? Well, in many of these, if this is a peripheral IC, it would have a mechanism either that addresses hardwired into the IC or oftentimes there are three address lines and you can set these three address lines to any particular combination that you would like to have.

(Refer Slide Time: 33:01)



Here is the timing diagram of communication on the I square C bus, you have to initiate a start condition which is when the serial clock pin is high and there is a high to low transition on the SDA bus and then you have certain number of bits and then you end that transmission with a stop conditions in which the clock pin is high and you have a low to high transition on the SDA pin this will terminate that channel that packet of data and the slave can then respond.

(Refer Slide Time: 33:34)



Pros and Cons

Pros:

- Only two wires required for communication
- Multiple master and multiple slave support

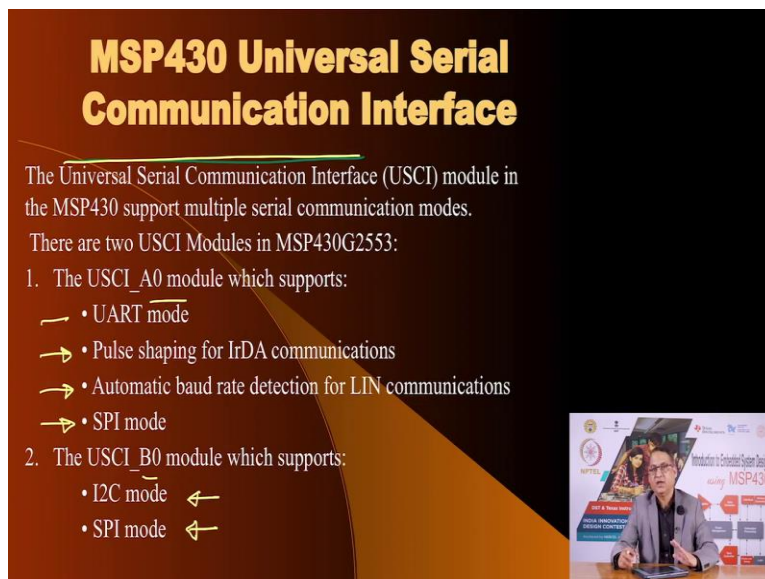
Cons:

- Slower data transfer rates when compared with SPI

The slide features a dark background with a large orange and yellow curved graphic on the left. A small inset image in the bottom right shows a man in a grey jacket sitting at a desk with a laptop, with a presentation board behind him that includes the text 'Introduction to Connected System Design using MSP430'.

Only two devices to pin two wires are required, so it is a advantage and you can have a master and multiple slaves, but the rate of communication is much lower compared to what you can achieve with the SPI.

(Refer Slide Time: 33:49)



MSP430 Universal Serial Communication Interface

The Universal Serial Communication Interface (USCI) module in the MSP430 support multiple serial communication modes.

There are two USCI Modules in MSP430G2553:

1. The USCI_A0 module which supports:
 - UART mode
 - Pulse shaping for IrDA communications
 - Automatic baud rate detection for LIN communications
 - SPI mode
2. The USCI_B0 module which supports:
 - I2C mode
 - SPI mode

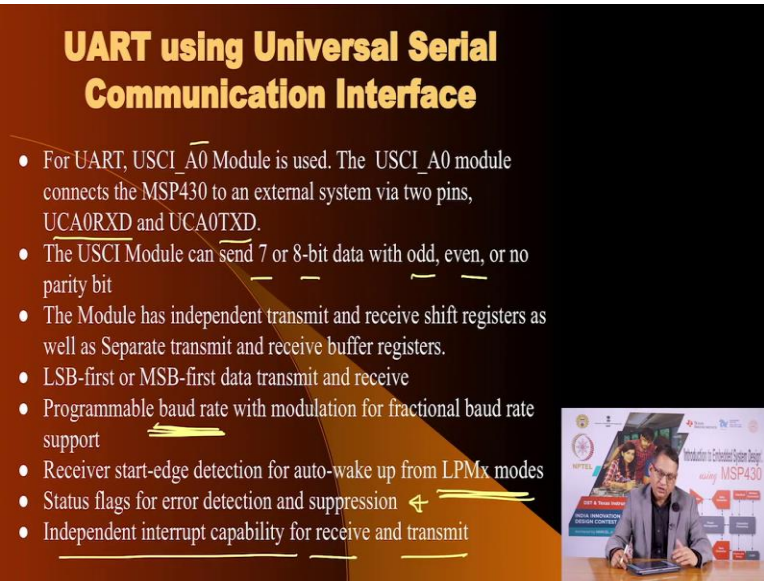
The slide features a dark background with a large orange and yellow curved graphic on the left. A small inset image in the bottom right shows a man in a grey jacket sitting at a desk with a laptop, with a presentation board behind him that includes the text 'Introduction to Connected System Design using MSP430'.

Now, let us look at the serial communication capabilities within MSP430 in the context of these three protocols and we will see how the peripheral, serial communication peripherals on MSP430 allows you to select one of these three protocols and so on. Now, there are two serial communication interface modules in MSP430G2553 that we are using and they are called

Universal Serial Communication Interface. The reason why they are called universal because each of these modules offer multiple protocols there is support for multiple protocols.

The A0 module offers UART it has a infrared data communication as well as a LIN protocol and the SPI mode and whereas the B0 module offers only the I square C and SPI mode. So, if you want to have one UART and one I square C it is possible. If you want to have two SPI it is possible, if you want to have one UART and one SPI it is possible. So, combinations of these protocols can be implemented on MSP430 microcontroller.

(Refer Slide Time: 35:00)



UART using Universal Serial Communication Interface

- For UART, USCI_A0 Module is used. The USCI_A0 module connects the MSP430 to an external system via two pins, UCA0RXD and UCA0TXD.
- The USCI Module can send 7 or 8-bit data with odd, even, or no parity bit
- The Module has independent transmit and receive shift registers as well as Separate transmit and receive buffer registers.
- LSB-first or MSB-first data transmit and receive
- Programmable baud rate with modulation for fractional baud rate support
- Receiver start-edge detection for auto-wake up from LPMx modes
- Status flags for error detection and suppression
- Independent interrupt capability for receive and transmit

The slide features a dark background with yellow and white text. A small inset image in the bottom right corner shows a man in a grey jacket sitting at a desk with a laptop, with a presentation slide visible behind him. The slide behind him has the text 'Introduction to Computer System Design using MSP430' and 'MSP430'.

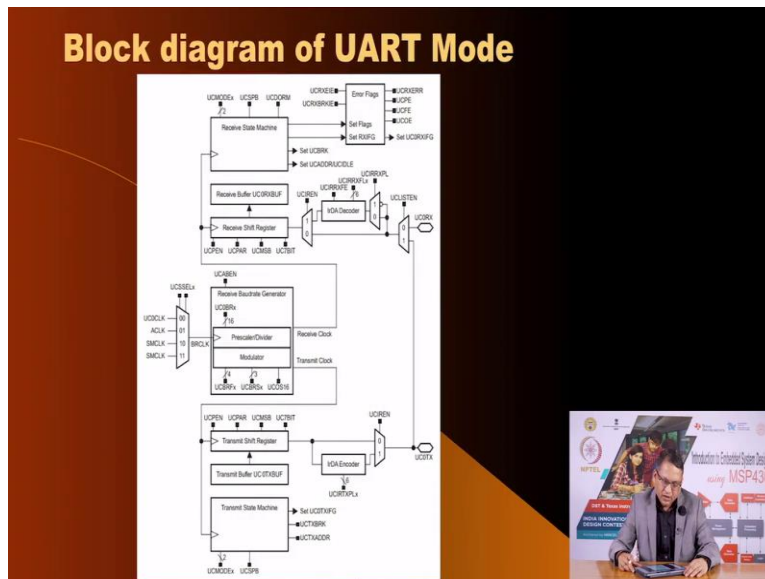
Now for UART, you had to use the A module, A0 module and it uses two pins those two pins will be listed on the microcontroller as RxD and DxD. Of course, these pins are mapped on available port pins, the module can send eight or seven or eight bits of data with odd even or no parity, it has independent transmit and receive shift registers, which is how it is able to communicate in a full duplex fashion.

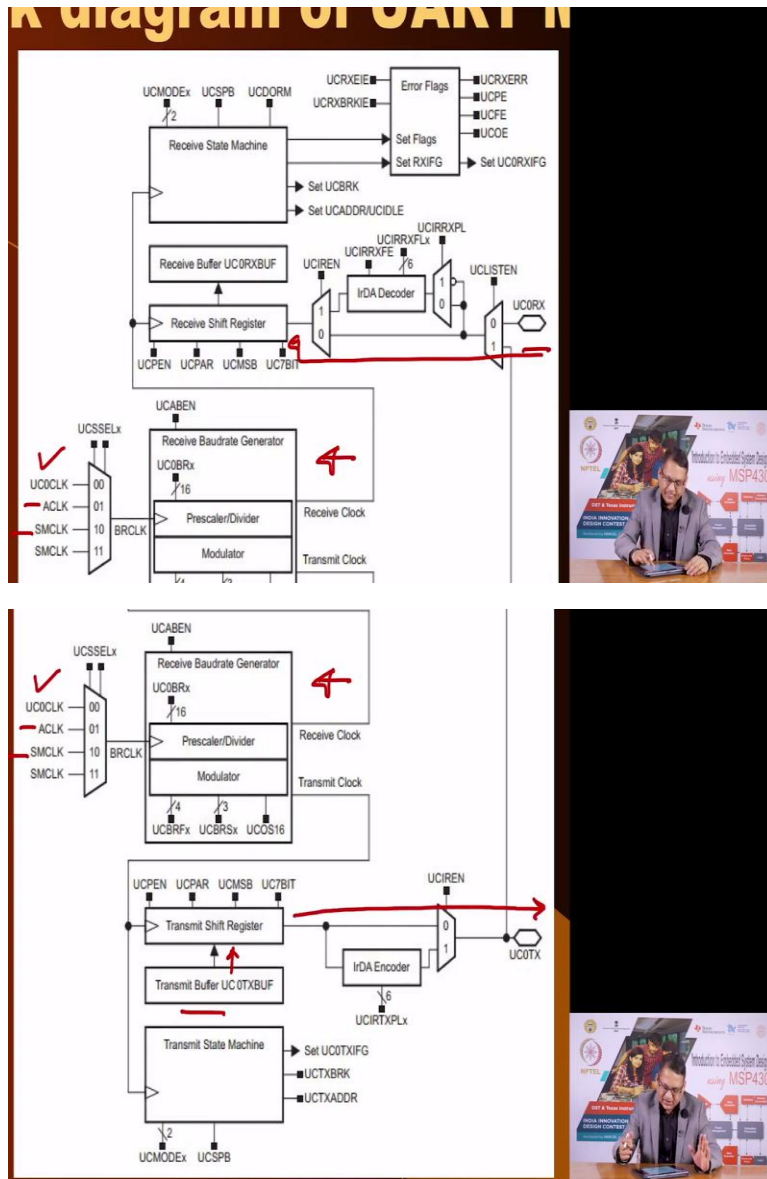
Remember, you are sending parallel data from the CPU to the peripheral and the peripheral converts that parallel data into a serial bit stream. So, essentially the UART interface acts like a serial register meaning either it works parallel in serial out for the transmitter and for the receiver it is like serial in parallel out.

You can decide whether you want to send the LSB first or MSB First, the standard method is to send the least significant bit first you have programmable baud rate, which means you can change for given clock signals available on your microcontroller, you can change the bit rate to support any of these standard bit rates.

There is a mechanism to exit the low power modes once serial communication data is received. There are flags which detect if some error has happened in the communication and there are independent interrupt capabilities for receiving and transmitting functions of the Universal Serial Communication Interface on MSP430.

(Refer Slide Time: 36:38)





Here is a block diagram. Let this block diagram not intimidate you. I am going to go through that. So, you see the most important part is the part which is the baud rate generator here. It gets, it has a possibility of selecting a source of clock. There is a internal clock generator you could choose or you could choose the a clock, auxiliary clock or you could choose the SM clock, this with the help of certain registers will divide the clock frequency down to get the bit rate clock.

Now, this clock is shared to the receiving section as well as the transmitting section, in the receiving section, the data that is coming from the RX pin goes into the shift register and once the data is received, it is transferred to the buffer receiver buffer and then the microcontroller

will get to know that a data byte is available on the receive buffer and then it can be transferred to the CPU.

On the transmitting side, the CPU writes data into the transmit buffer from the transmit buffer it is transferred into the shift register, so that the data is serialized and sent out to the TX pin. That is essentially the entire simplified explanation of how the UART module on MSP430 works. Of course, to be able to do that it requires many-many registers and we will just go through those registers.

(Refer Slide Time: 38:12)

UART Character Format

- UART Format consists of a start bit, seven or eight data bits, an even/odd/no parity bit, an address bit (address-bit mode), and one or two stop bits. The UCMSB bit controls the direction of the transfer and selects LSB or MSB first.


The diagram illustrates the UART character format. It shows a sequence of bits: a Start Bit (ST), followed by data bits (D0 to D7), an optional Address Bit (PA), and one or two Stop Bits (SP1, SP2). A red arrow indicates the direction of data flow from the start bit to the data bits. Callouts provide specific bit settings: [2nd Stop Bit, UCSPB = 1], [Parity Bit, UCPEM = 1], [Address Bit, UCMODEx = 10], [8th Data Bit, UC7BIT = 0], and [Optional Bit, Condition].

The UART format as I mentioned can be D0 down to the last bit and then you have parity bits and so on and stop bits.

(Refer Slide Time: 38:19)

Setting UART Baud Rate

- UCA0BR0, UCA0BR1 and UCA0MCTL registers values are set as per the baud rate clock source and baud rate requirement.
- Values for the setting baud rate are provided in device user guide.




You have baud rate selector there are three register baud rate 0, register baud rate 1 and modulation control registers and the values that these registers must be initialized. The user manual has a useful guide which tells you, if this is your clock source and you want to communicate at certain baud rate what should be the values that you should load into the baud rate register 0, baud rate register 1 and the modulation control registers.

(Refer Slide Time: 38:49)

Example register values for baud rate

Table 15-4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0

BRCLK Frequency [Hz]	Baud Rate [Baud]	UCBRx	UCBRsx	UCBRFx	Maximum TX Error (%)		Maximum RX Error (%)	
32,768	1200	27	2	0	-2.8	1.4	-5.9	2.0
32,768	2400	13	6	0	-4.8	6.0	-9.7	8.3
32,768	4800	6	7	0	-12.1	5.7	-13.4	19.0
32,768	9600	3	3	0	-21.1	15.2	-44.3	21.3
1,048,576	9600	109	2	0	-0.2	0.7	-1.0	0.8
1,048,576	19200	54	5	0	-1.1	1.0	-1.5	2.5
1,048,576	38400	27	2	0	-2.8	1.4	-5.9	2.0
1,048,576	56000	18	6	0	-3.9	1.1	-4.6	5.7
1,048,576	115200	9	1	0	-1.1	10.7	-11.5	11.3
1,048,576	128000	8	1	0	-8.9	7.5	-13.8	14.8
1,048,576	256000	4	1	0	-2.3	25.4	-13.4	38.8
1,000,000	9600	104	1	0	-0.5	0.6	-0.9	1.2
1,000,000	19200	52	0	0	-1.8	0	-2.6	0.9




Here are examples that if your clock is 32768 and you want to do at 9600 what values should be stored in the baud rate registers and the modulation registers.

(Refer Slide Time: 39:03)

USCI Tx Interrupt

- The USCI has one interrupt vector for transmission and one interrupt vector for reception.
- The UCA0TXIFG interrupt flag is set by the transmitter to indicate that UCA0TXBUF is ready to accept another character.
- An interrupt request is generated if UCA0TXIE and GIE are also set.
- UCA0TXIFG is automatically reset if a character is written to UCA0TXBUF.




Then once you have, you want to transmit data, you can do that with the help of an interrupt, there is an interrupt vector for transmit which means the moment the output buffer is empty, it will generate an interrupt to the CPU and the CPU can then load a new value into the buffer and there is a flag interrupt flag which tells whether the transmitter is ready and you can also enable and disable this interrupt.

(Refer Slide Time: 39:33)

USCI Rx Interrupt

- The UCA0RXIFG interrupt flag is set each time a character is received and loaded into UCA0RXBUF.
- An interrupt request is generated if UCA0RXIE and GIE are also set.
- UCA0RXIFG and UCA0RXIE are reset by a system reset PUC signal.
- UCA0RXIFG is automatically reset when UCA0RXBUF is read.




You also have a independent interrupt for the receiver receiving function and whenever the interrupt flag is set, that means a new character is available in the buffer and the CPU can read it.

The interrupt request is generated if this flag is one and you have enabled the global interrupts and you reset it by resetting this flag and it is also reset on power up clear condition and this flag is automatically reset when you read this buffer register. So, when you read it, you do not have to clear that flag, the error flag is automatically cleared.

(Refer Slide Time: 40:11)

USCI Registers for UART Mode

Register	Short Form	Register Type	Address	Initial State
USCI_A0 control register 0	UCA0CTL0	Read/write	060h	Reset with PUC
USCI_A0 control register 1	UCA0CTL1	Read/write	061h	001h with PUC
USCI_A0 Baud rate control register 0	UCA0BR0	Read/write	062h	Reset with PUC
USCI_A0 baud rate control register 1	UCA0BR1	Read/write	063h	Reset with PUC
USCI_A0 modulation control register	UCA0MCTL	Read/write	064h	Reset with PUC
USCI_A0 status register	UCA0STAT	Read/write	065h	Reset with PUC
USCI_A0 receive buffer register	UCA0RXBUF	Read	066h	Reset with PUC
USCI_A0 transmit buffer register	UCA0TXBUF	Read/write	067h	Reset with PUC
USCI_A0 Auto baud control register	UCA0ABCTL	Read/write	06Dh	Reset with PUC
USCI_A0 IrDA transmit control register	UCA0IRCTL	Read/write	05Eh	Reset with PUC
USCI_A0 IrDA receive control register	UCA0IRRCTL	Read/write	05Fh	Reset with PUC
SFR interrupt enable register 2	IE2	Read/write	001h	Reset with PUC
SFR interrupt flag register 2	IFG2	Read/write	003h	00Ah with PUC




These are the various registers for UART module, there is a control register 0 and 1 for A0 there is a baud rate register as I mentioned baud rate register 01 and a modulation control register. This status register will tell you, what is the state of transmission and reception, whether a new byte has been received or the old byte has been transmitted.

If there is an error in those transmissions, then you have receive buffer and a transmit buffer you have a, there is a mechanism to automatically detect the baud rate of incoming data and you can use that, then there are registers for infrared communication and then you have interrupt enable register as well as inter flag registers.

(Refer Slide Time: 40:59)

Register 0

	7	6	5	4	3	2	1	0
	UCPEN	UCPAR	UCMSB	UCTBIT	UCSPB	UCMODEx		UCSYNC
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
UCPEN	Bit 7	Parity enable						
		0 Parity disabled						
		1 Parity enabled. Parity bit is generated (UCA1TXD) and expected (UCA1RXD). In address-bit multiprocessor mode, the address bit is included in the parity calculation.						
UCPAR	Bit 6	Parity select. UCPAR is not used when parity is disabled.						
		0 Odd parity						
		1 Even parity						
UCMSB	Bit 5	MSB first select. Controls the direction of the receive and transmit shift register.						
		0 LSB first						
		1 MSB first						
UCTBIT	Bit 4	Character length. Selects 7-bit or 8-bit character length.						
		0 8-bit data						
		1 7-bit data						
UCSPB	Bit 3	Stop bit select. Number of stop bits.						
		0 One stop bit						
		1 Two stop bits						
UCMODEx	Bits 2-1	USCI mode. The UCMODEx bits select the asynchronous mode when UCSYNC = 0.						
		00 UART mode						
		01 Idle-line multiprocessor mode						
		10 Address-bit multiprocessor mode						
		11 UART mode with automatic baud rate detection						
UCSYNC	Bit 0	Synchronous mode enable						
		0 Asynchronous mode						
		1 Synchronous mode						




I strongly recommend that you go through these slides to understand the operation of the control register. For example, here you are selecting whether you want the parity to be enabled, do you want the MSB first or the LSB first, what is the size of the data transmission seven bits or eight bits, whether you want a one stop bit or two stop bits, whether you want to operate in the UART mode and so on and whether you are doing synchronous or asynchronous mode of communication.

(Refer Slide Time: 41:29)

UCA0CTL1: USCI_A0 Control Register 1

	7	6	5	4	3	2	1	0
	UCSSELx	UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST	
	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1
UCSSELx	Bits 7-6	USCI clock source select. These bits select the BRCLK source clock.						
		00 UCLK						
		01 ACLK						
		10 SMCLK						
		11 SMCLK						
UCRXEIE	Bit 5	Receive erroneous-character interrupt-enable						
		0 Erroneous characters rejected and UCA1RXIFG is not set						
		1 Erroneous characters received will set UCA1RXIFG						
UCBRKIE	Bit 4	Receive break character interrupt-enable						
		0 Received break characters do not set UCA1RXIFG						
		1 Received break characters set UCA1RXIFG						
UCDORM	Bit 3	Dormant. Puts USCI into sleep mode						
		0 Not dormant. All received characters will set UCA1RXIFG						
		1 Dormant. Only characters that are preceded by an idle-line or with address bit set will set UCA1RXIFG. In UART mode with automatic baud rate detection only the combination of a break and synch field will set UCA1RXIFG.						
UCTXADDR	Bit 2	Transmit address. Next frame to be transmitted will be marked as address depending on the selected multiprocessor mode.						
		0 Next frame transmitted is data						
		1 Next frame transmitted is an address						
UCTXBRK	Bit 1	Transmit break. Transmits a break with the next write to the transmit buffer. In UART mode with automatic baud rate detection 05h must be written into UCA1TXBUF to generate the required break/synch fields. Otherwise 0h must be written into the transmit buffer.						
		0 Next frame transmitted is not a break						
		1 Next frame transmitted is a break or a break/synch						
UCSWRST	Bit 0	Software reset enable						
		0 Disabled. USCI reset released for operation.						
		1 Enabled. USCI logic held in reset state.						



Then the control one register allows you to select what sort of clock source you would like to have and so on.

(Refer Slide Time: 41:38)

UCA0BR0: USCI_A0 Baud Rate Control Register 0

7	6	5	4	3	2	1	0
UCA0BR0							
rw							

UCA0BR1: USCI_A0 Baud Rate Control Register 1

7	6	5	4	3	2	1	0
UCA0BR1							
rw							

UCA0BR1: USCI_A0 Baud Rate Control Register 1. Clock prescaler setting of the baud rate generator. The 16-bit value of (UCA0BR0 * UCA0BR1 + 256) forms the prescaler value.

Then these are baud rate registers, registers 0 and 1. We will see a code example to understand these registers better.

(Refer Slide Time: 41:46)

UCA0MCTL: USCI_A0 Modulation Control Register

7	6	5	4	3	2	1	0
UCBRFx				UCBSRx		UCOS16	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCBRFx: Bits 7-4. First modulation stage select. These bits determine the modulation pattern for BITCLK16 when UCOS16 = 1. Ignored with UCOS16 = 0. Table 15-3 shows the modulation pattern.

UCBSRx: Bits 3-1. Second modulation stage select. These bits determine the modulation pattern for BITCLK. Table 15-2 shows the modulation pattern.

UCOS16: Bit 0. Oversampling mode enabled.
 0 Disabled
 1 Enabled

You have the modulation control register, the modulation control register is there, so that the bit rate which is generated by dividing the system clock or whatever clock source you have chosen.

If there is a error then the modulation control register will allow you to minimize the effect of that error.

(Refer Slide Time: 42:06)

UCA0STAT: USCI_A0 Status Register

7	6	5	4	3	2	1	0
UCLISTEN	UCFE	UCOE	UCPE	UCBRK	UCRERR	UCADDR UCIDLE	UCBUSY
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCLISTEN Bit 7: Listen enable. The UCLISTEN bit selects loopback mode.
 0 Disabled
 1 Enabled. UCA0TXD is internally fed back to the receiver.

UCFE Bit 6: Framing error flag.
 0 No error
 1 Character received with low stop bit

UCOE Bit 5: Overrun error flag. This bit is set when a character is transferred into UCA0RXBUF before the previous character was read. UCOE is cleared automatically when UCA0RXBUF is read, and must not be cleared by software. Otherwise, it will not function correctly.
 0 No error
 1 Overrun error occurred

UCPE Bit 4: Parity error flag. When UCPEM = 0, UCPE is read as 0.
 0 No error
 1 Character received with parity error


UCBRK Bit 3: Break detect flag.
 0 No break condition
 1 Break condition occurred

UCRERR Bit 2: Receive error flag. This bit indicates a character was received with error(s). When UCRERR = 1, on or more error flags (UCFE, UCPE, UCOE) is also set. UCRERR is cleared when UCA0RXBUF is read.
 0 No receive errors detected
 1 Receive error detected

UCADDR Bit 1: Address received in address-stroke multiprocessor mode.
 0 Received character is data
 1 Received character is an address

UCIDLE: Idle line detected in idle-line multiprocessor mode.
 0 No idle line detected
 1 Idle line detected

UCBUSY Bit 0: USCI busy. This bit indicates if a transmit or receive operation is in progress.
 0 USCI inactive
 1 USCI transmitting or receiving



This is the status register. Status register which tells you whether data has been, whether there is a parity error, whether receive any other error on the receiving function and so on.

(Refer Slide Time: 42:20)

UCA0RXBUF: USCI_A0 Receive Buffer Register


7	6	5	4	3	2	1	0
UCA0RXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

UCA0RXBUFx Bits 7:0: The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCA0RXBUF resets the receive-error bits, the UCADDR or UCIDLE bit, and UCA0RXIFG. In 7-bit data mode, UCA0RXBUF is LSB justified and the MSB is always reset.

UCA0TXBUF: USCI_A0 Transmit Buffer Register

7	6	5	4	3	2	1	0
UCA0TXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

UCA0TXBUFx Bits 7:0: The transmit data buffer is user accessible and holds the data waiting to be moved into the transmit shift register and transmitted on UCA0TXD. Writing to the transmit data buffer clears UCA0TXIFG. The MSB of UCA0TXBUF is not used for 7-bit data and is reset.



Then these are the receive buffers, here and the transmit buffer transmits so that you can receive and transmit data. These are the two registers in which you are going to write data for transmission and you are going to read that register whenever serial data is received.


(Refer Slide Time: 42:36)

IE2: Interrupt Enable Register 2

		7	6	5	4	3	2	1	0	
									UCAITXIE	UCAIRXIE
									rw-0	rw-0
Bits 7-2		These bits may be used by other modules (see the device-specific data sheet).								
UCAITXIE	Bit 1	USCI_A0 transmit interrupt enable								
	0	Interrupt disabled								
	1	Interrupt enabled								
UCAIRXIE	Bit 0	USCI_A0 receive interrupt enable								
	0	Interrupt disabled								
	1	Interrupt enabled								

IFG2: Interrupt Flag Register 2

		7	6	5	4	3	2	1	0	
									UCAITXIFG	UCAIRXIFG
									rw-1	rw-0
Bits 7-2		These bits may be used by other modules (see the device-specific data sheet).								
UCAITXIFG	Bit 1	USCI_A0 transmit interrupt flag. UCAITXIFG is set when UCAITXBUFF is empty.								
	0	No interrupt pending								
	1	Interrupt pending								
UCAIRXIFG	Bit 0	USCI_A0 receive interrupt flag. UCAIRXIFG is set when UCAIRXBUFF has received a complete character.								
	0	No interrupt pending								
	1	Interrupt pending								




You have the Interrupt enable register 2, and interrupt flag register 2, you can disable or enable those interrupts through these registers.

(Refer Slide Time: 42:47)

Selecting UART function on P1.1 and P1.2

DIR SEL1 SEL2

		P1.x (IO)	1	0	0	0	0	
P1.1 TAB.0/	TAB.0	1	1	0	0	0	0	
	TAB.CC0A	0	1	0	0	0	0	
	UCAIRXD/	UCAIRXD	from USCI	1	1	0	0	
	UCAIRSM/	UCAIRSM	from USCI	1	1	0	0	
	A1 ^(?)	A1	X	X	X	1 (y = 1)	0	
	CA1	CA1	X	X	X	0	1 (y = 1)	
	Pin Disc	Capacitive sensing	X	0	1	0	0	
	P1.2 TAB.1/	P1.x (IO)	1	0	0	1	0	0
		TAB.1	1	1	0	0	0	0
		TAB.CC1A	0	1	0	0	0	0
UCAITXD/		UCAITXD	from USCI	1	1	0	0	
UCAIRSM/		UCAIRSM	from USCI	1	1	0	0	
A2 ^(?)		A2	X	X	X	1 (y = 2)	0	
CA2		CA2	X	X	X	0	1 (y = 2)	
Pin Disc		Capacitive sensing	X	0	1	0	0	



And here is now, as I mentioned, the transmit and receive pins are shared with port pins. In the case of MSP430G2553 the transmitting pin is on, receiving pin is on P1.1, here p1.1 and the transmittal pin here is on P1.2. So you have to, the direction is automatically determined when you write into the USCI register.

But the cell one and cell two bits you have to for RxD you have to write one and one here and for transmitting also have to write one there one here, once you write these registers you are set the two pins P1.1 and P1.2 are configured for receiving and transmitting serial data using URAT by writing appropriate numbers into the baud rate register, you have decided the baud rate by writing into the interrupt registers, you have decided whether you want to interrupt on transmission or reception and by writing into the other control register you are set and then, let us see an example. What we are going to do is we are going to take the MSP430 lunch box.

We are going to communicate to PC on the PC, we will show what kind of software, you should install, so that you can transmit data from the PC, you type any key on the keyboard of the laptop or desktop, that value will be encoded serialized and sent and the microcontroller MSP430 lunch box will receive it. It will increment the value and send it back and the reason why we have chosen to increment the value is, so that if by mistake the RX and TX pins are shorted. If you just use a echo function that is if I send A, I should get back A.

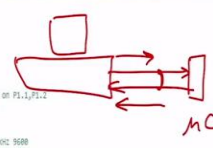
(Refer Slide Time: 44:39)

Example Code : Hello Serial

```


1 #include <msp430.h>
2
3 /**
4  * @brief
5  * These settings are yet enabling uart on Lunchbox
6  */
7 void register_settings_for_UART()
8 {
9     PSEL1 = BIT1 + BIT2; // Select UART RX/TX function on P1.1,P1.2
10    PSEL2 = BIT1 + BIT2;
11
12    UC0CTL1 |= UCSSEL_1; // UART Clock -> ACLK
13    UC0BR0 = 3; // Baud Rate Setting for 320Kc 9600
14    UC0BR1 = 0; // Baud Rate Setting for 320Kc 9600
15    UC0MCTL = UC0RS_0 + UC0RS_3; // Modulation Setting for 320Kc 9600
16    UC0CTL1 &&= ~UC0SWRST; // Initialize UART module
17    IE2 |= UC0RXIE; // Enable RX interrupt
18 }
19
20 /*@brief entry point for the code*/
21 void main(void)
22 {
23     WDTCTL = WDTHW + WDTHOLD; // Stop watchdog (not recommended for code in production and devices working in field)
24
25     register_settings_for_UART();
26
27     __bis_sr_register(LPM0_bits + GIE); // Enter LPM0, Enable Interrupt
28 }
29
30 /**
31  */
32 * @brief
33  * Interrupt vector for UART on Lunchbox
34  */
35 #pragma vector=USCI0RX_VECTOR // UART RX Interrupt vector
36 __interrupt void USCI0RX_ISR(void)
37 {
38     while (!IFG1UC0RXIFG); // Check if TX is ongoing
39     UC0TXBUF = UC0RXBUF + 1; // TX -> Received Char + 1
40 }

```



MC

A - B
1 -> 2



That is if here is your PC. Let us say here is your PC and from this PC, this is your microcontroller, this is the transmit pin, receive pin if by chance they get shorted. Then if this fellow sends a value it will be received back and you might think, oh it is working fine, but it could be because the pins are shorted.

So, I do not recommend a echo function. I have a modified echo function which means whatever value is sent. I increment it and send it back and if you transmit information using ASCII protocol you know that the code for A, if you add one to it you get a ASCII code for B. If you send 1 you will get 2 and so on. So, this is a very good test visually, you can decipher whether the value that you are transmitting, whether they are being received properly or not.

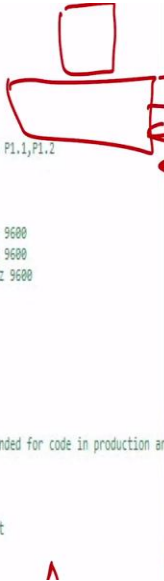
(Refer Slide Time: 45:31)

```
/**
 * @brief
 * These settings are wrt enabling uart on Launchbox
 */
void register_settings_for_UART()
{
    P1SEL = BIT1 + BIT2; // Select UART RX/TX function on P1.1,P1.2
    P1SEL2 = BIT1 + BIT2;

    UCARCTL1 |= UCSSSEL_1; // UART Clock -> ACLK
    UCABRR0 = 3; // Baud Rate Setting for 32kHz 9600
    UCABRR1 = 0; // Baud Rate Setting for 32kHz 9600
    UCARNTL = UCBRF_0 + UCBR5_3; // Modulation Setting for 32kHz 9600
    UCARCTL1 &= ~UCSWRST; // Initialize UART Module
    IE2 |= UCABRXIE; // Enable RX interrupt
}

/*@brief entry point for the code*/
void main(void)
{
    WDCTL = WDTPW + WDTHOLD; // Stop Watchdog (Not recommended for code in production an
    register_settings_for_UART();
    _bis_sr_register(LPM0_bits + GIE); // Enter LPM0, Enable Interrupt
}

```



Let us go through that it is a just a single page of code, you have in the main program, all you have done is you have, reset the stop the watchdog timer as we always do and then we are calling a sub routine for setting the UART registers here is that register, in this the P cell 1 and P cell 2 bits are written appropriately, so that you can map the transmit and receive functions on P 1.1 and 1.2.

(Refer Slide Time: 45:54)

```

// Register address for UART0
PISSEL = BIT1 + BIT2; // Select UART RX/TX function on P1.1,P1.2
PISSEL2 = BIT1 + BIT2;

UCABCTL1 |= UCSSEL_1; // UART Clock -> ACLK
UCABRR0 = 3; // Baud Rate Setting for 32kHz 9600
UCABRR1 = 0; // Baud Rate Setting for 32kHz 9600
UCABMCTL = UCBRF_0 + UCBR3; // Modulation Setting for 32kHz 9600
UCABCTL1 &= ~UCSWRST; // Initialize UART MODULE
IE2 |= UCABRXIE; // Enable RX interrupt

// Brief entry point for the code
} main(void)

WDCTL = WDTMW + WDTWOLD; // Stop Watchdog (Not recommended for code in production and de

register_settings_for_UART();

_bis_sr_register(LPM0_bits + GIE); // Enter LPM0, Enable Interrupt

// Brief
Interrupt Vector for UART on LunchBox


```

Handwritten annotations on the code include red arrows pointing to specific lines, a red box around the initialization of the UART module, and a red box around the interrupt enable line. A red box also highlights the comment about the Watchdog. A red box highlights the register settings function call. A red box highlights the LPM0 register setting. A red box highlights the interrupt vector comment. A red box highlights the interrupt vector comment.

Handwritten calculations on the right side of the code show:

$$\frac{32768}{9600} = 3.41$$

Handwritten notes include "A - B" and "1 -> 2".



Then you have chosen to use the UART clock from ACLK, in our case ACLK is coming from the low frequency crystal oscillator that is 32768 the bit rate values are stored in these 2 registers value of 3. That is 32768 divided by, 9600. You will get a value of 3.41. So, this 3 you are writing into one of the registers. This 0.41 into 8. The resultant value you write into the modulation, register here.

The three value and then you initialize the UART module by writing into the control register and then you enable the receiving interrupt. So, what we are doing is the MSP430 lunch box is once it is programmed, it is waiting for byte to be received from the transmitter from the external world, in this case the PC, once that value is received, it will. We will see what it does.

(Refer Slide Time: 46:55)

```
4 UCARXCTL = UCARXCTL0 + UCARXCTL1; // MODULATION SETTING FOR 52KHZ 9600
5 UCARCTL1 &= ~UCSWRST; // Initialize UART Module
7 IE2 |= UCARXIE; // Enable RX interrupt
8 }
9 }
10 /*@brief entry point for the code*/
11 void main(void)
12 {
13     WDCTL = WDTPW + WDTHOLD; // Stop Watchdog (Not recommended for code in production a
14     register_settings_for_UART();
15     _bis_SR_register(LPM0_bits + GIE); // Enter LPM0, Enable Interrupt
16 }
17 }
18 }
19 /**
20  * @brief
21  * Interrupt Vector for UART on LunchBox
22  */
23 #pragma vector=USCI0RX_VECTOR // UART RX Interrupt Vector
24 _interrupt void USCI0RX_ISR(void)
25 {
26     while (!(IFG2&UCARXIFG)); // Check if TX is ongoing
27     UCARXBUF = UCARXBUF + 1; // TX -> Received Char + 1
28 }
```

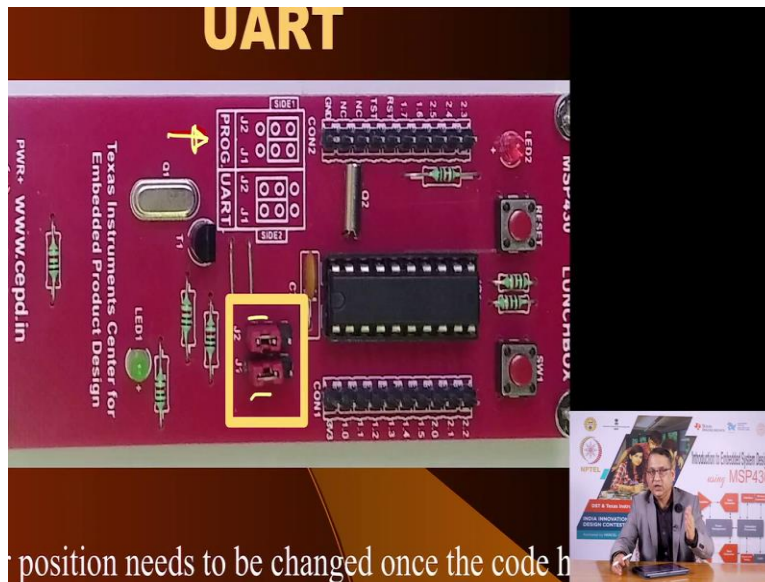
Handwritten annotations on the code:
- Red arrows point to line 5, line 14, line 26, and line 27.
- Red text "9600" is written next to line 4.
- Red text "3.41" is written next to line 15.
- Red text "A" and "1" are written next to lines 14 and 27 respectively, with arrows pointing to the corresponding code lines.

So once the, and then UART is programmed then it just sets the registers in the status registers to status register to enable the global interrupts and when a byte is received it will generate a interrupt vector. Here, we are going to go into the receive vector. In that receive vector, we are waiting, we have received the byte but since we want be increment and transmit it, we want to make sure that the transmit buffer is available.

So, wait if the transmit buffer is not available. Once that is available you simply write into the transmit buffer, the value that you get from the receive buffer plus 1. This is the entire code, I strongly recommend that you build and download this code into your MSP430 lunch box. Now, there is a little catch here, because our MSP430 lunch box. Since we are using the bootloader mechanism, the pins for bootloader serial communication are different from the two UART pins that is TxD and RxD.

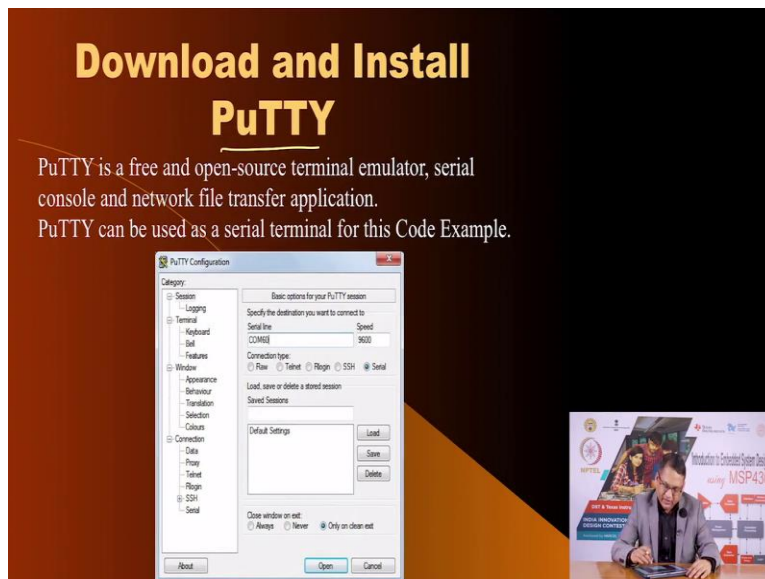
So, we have jumpers on the MSP430 lunchbox, which you must modify, after you have downloaded the program from the laptop or desktop into the lunch box, then you should remove the jumpers and place them in the appropriate location, which I will show you in the next photograph and then you should invoke a program on your laptop which I will show.

(Refer Slide Time: 48:20)



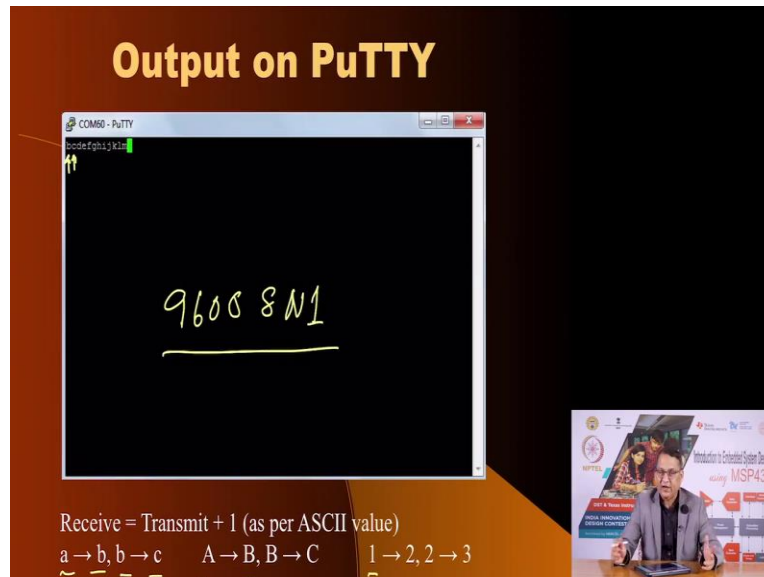
So, first of all this is the jumper settings for programming, for programming download the programming program from the desktop into the microcontroller the jumper settings there are three pins here in this, these two should be shorted. When you are downloading the program. Once the program is downloaded. You have to remove this shorting jumpers and you connect them on this side, on these two sides. Now, your microcontroller is ready to communicate with the PC using UART.

(Refer Slide Time: 48:53)



And what do you do, you install a software called PuTTY. PuTTY allows you to select the baud rate and the rest of the data bits and then when you have initialize it

(Refer Slide Time: 49:04)



And when you transmit this is what it receives. So, we sent A, we got B. We sent B we got C, and so on. So, if you send A you will get B, if you send B you will C, you will send a capital A you will get capital B and so on. You will send 1 you will get 2 and so on.

So, if this works on your PuTTY on your laptop or desktop that means congratulations the microcontroller is working as expected. It is communicating with the PC at the rate of 96008 N 1 protocol and of course you can experiment with other bit rates to see whether that is possible.

So, in this lecture, we have seen how MSP430 microcontroller can communicate using a variety of protocols, but because of the limitation of the coverage that we can do in this course, we have restricted ourselves to communicating using only the UART.

You are free to read the datasheet to understand how the SPI and I square C bus works and you could use your MSP430 for communicating with external devices within the same microcontroller system, embedded system to utilize the SPI and I square C bus also and you can use the UART to communicate between two microcontrollers or between one microcontroller, and one PC as we have illustrated here.

The code example that we have shared can be extended for use between two microcontrollers and so on. So, with this, I finish this lecture here and I will see you soon with a new lecture. Thank you.