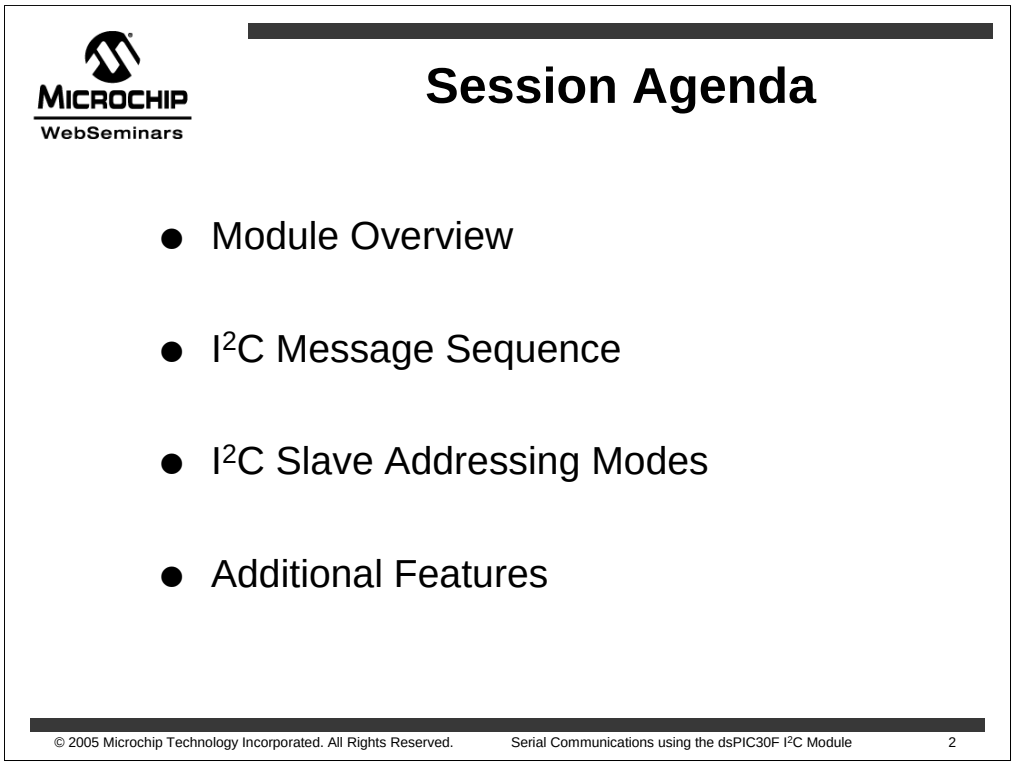
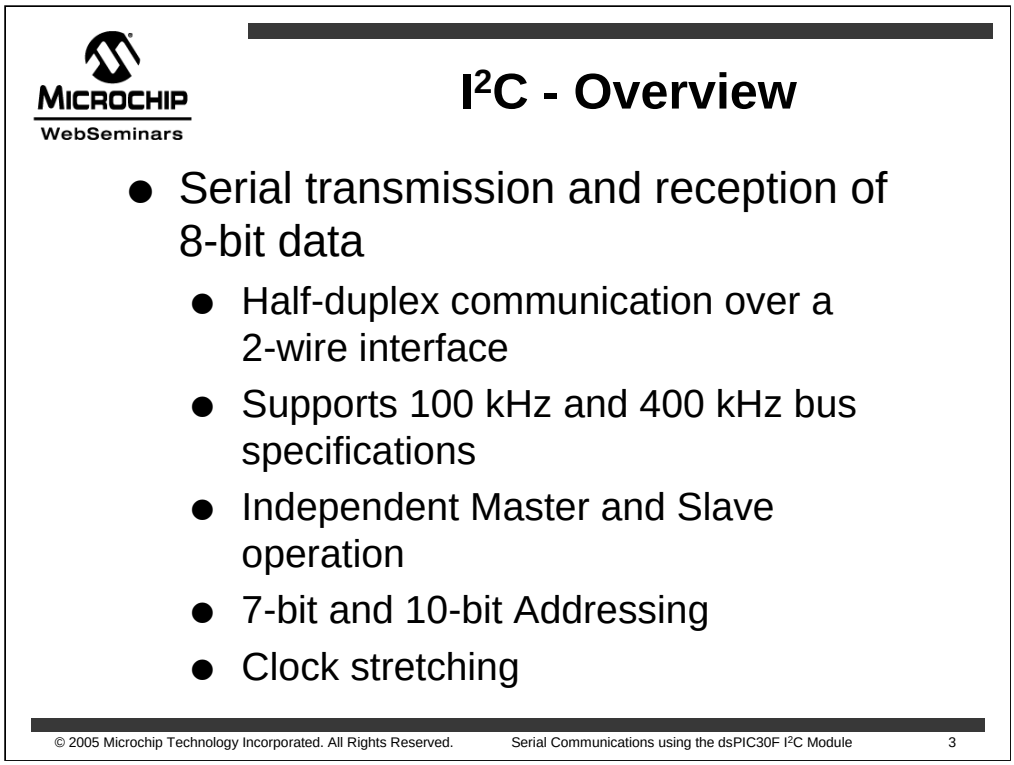
Welcome to the “Serial Communications using the dsPIC30F I²C Module” web seminar.



In today's session, we will start by outlining some key features of the Inter-Integrated Circuit, or I^2C, module in the dsPIC30F family of devices. We will then delve deeper into the processes of data transmission and reception through the I^2C, as well as various slave device addressing techniques. Finally, we will study some additional features of the I^2C module.



The I^2C module is used for transmission and reception of 8-bit data over a 2-wire interface consisting of a serial clock line, SCL, and a serial data line, SDA.

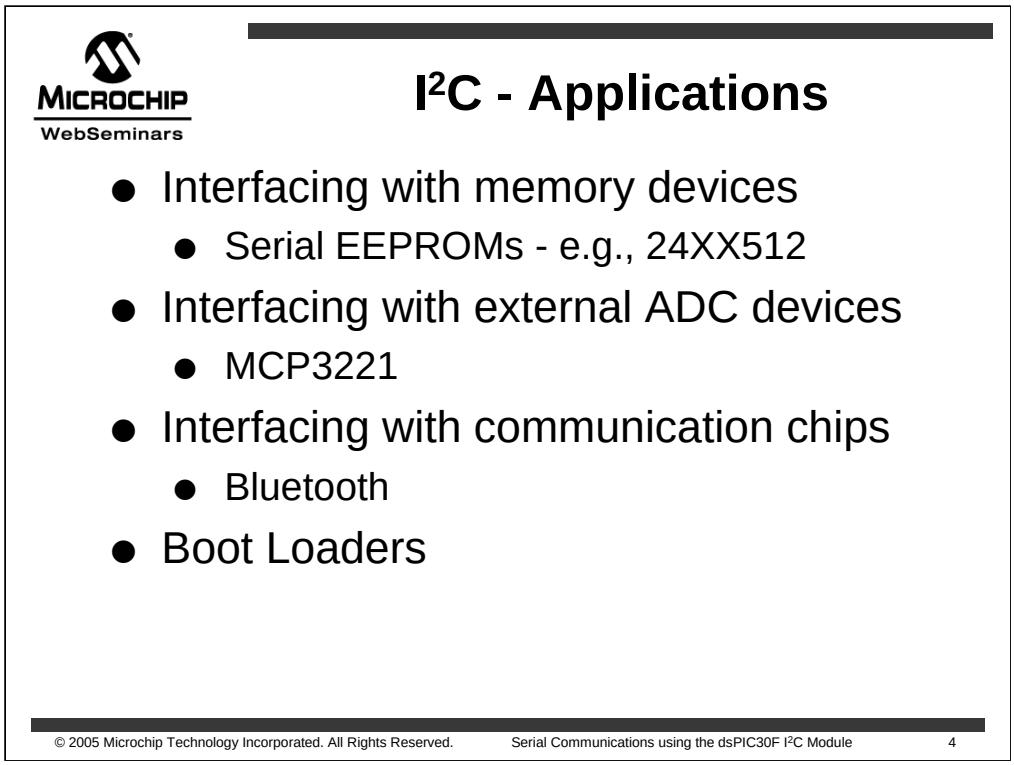
I^2C communication is half-duplex, which means either a transmission or a reception can occur at a time. It is also an example of Synchronous communication, as both communicating entities use the same serial clock signal.

The I^2C module may be used for communicating with external peripherals such as Analog-to-Digital or Digital-to-Analog Converters, external memory devices such as Serial EEPROMs, or other microcontrollers. It supports 100 kHz and 400 kHz baud rates as specified by the I^2C standard. In general, baud rates of up to 1 MHz are supported.

The Master and Slave logic in the I^2C module operate independently, rather than the user having to configure the module as either a Master or a Slave.

The I^2C module supports 7-bit or 10-bit Slave address detection, besides processing of special addresses such as General Call Address and supporting the Intelligent Peripheral Management Interface, or IPMI, addressing.

In addition, the module includes integrated signal conditioning features such as Slew Rate Control and support for SMBus signal levels.

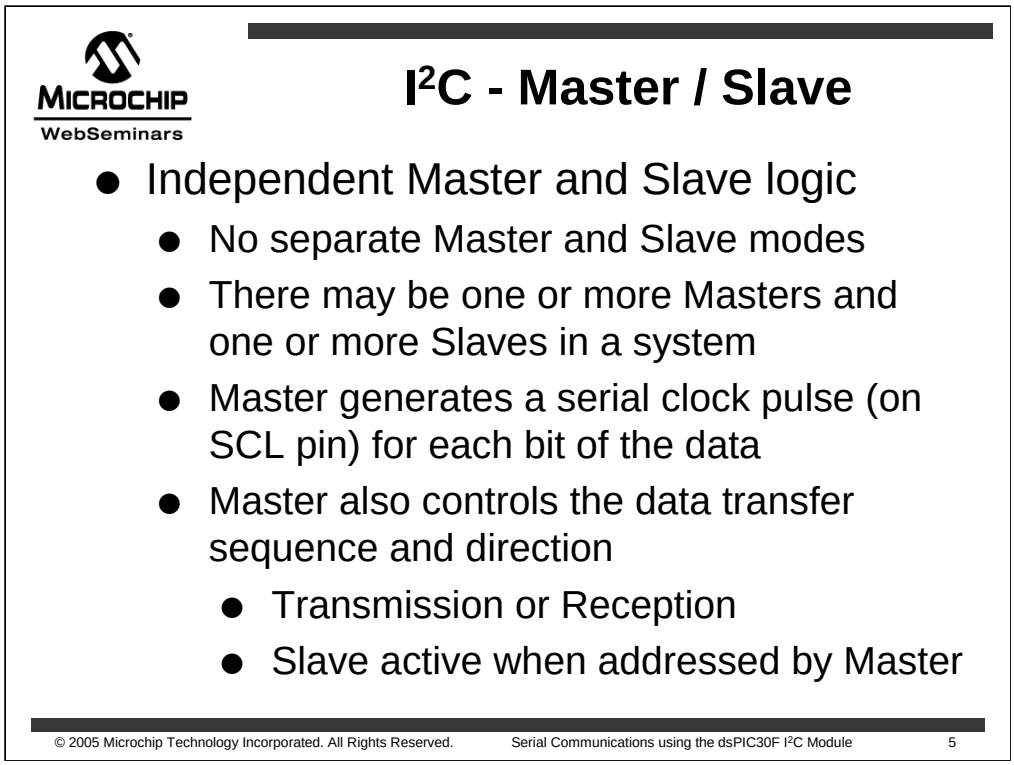


The I^2C module can be utilized to interface and communicate with a wide variety of external peripheral devices.

For example, it can be used to program and read external memory devices, such as the 24xx512 Serial EEPROM from Microchip, or to interface with external Analog-to-Digital Converters, such as the MCP3221 from Microchip.

Another popular use of the I^2C module is as a control channel for dedicated communication processors running a Bluetooth protocol stack for wireless communications.

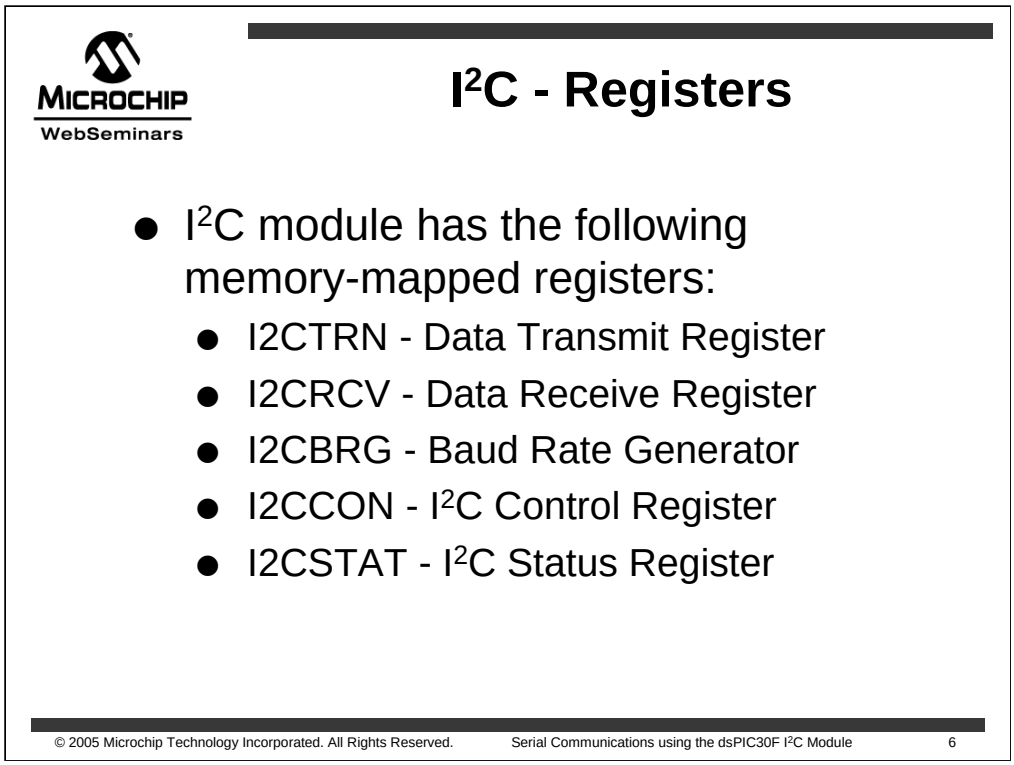
Last but not the least, I^2C is a very efficient communication channel for a Boot Loader program running on the dsPIC, to exchange command and response messages with an external Host device.



The I²C module does not have separate Master and Slave modes; Master or Slave states are defined by the actions performed by the module at any given time. In a system, there may be multiple Masters and Slaves, though only one Master may control the bus at a time.

The Master is responsible for generating serial clock pulses on the SCL line that will be used to shift out data through the SDA line. The Master also determines whether the data will be transmitted by the Master and received by the Slave, or transmitted by the Slave and received by the Master.

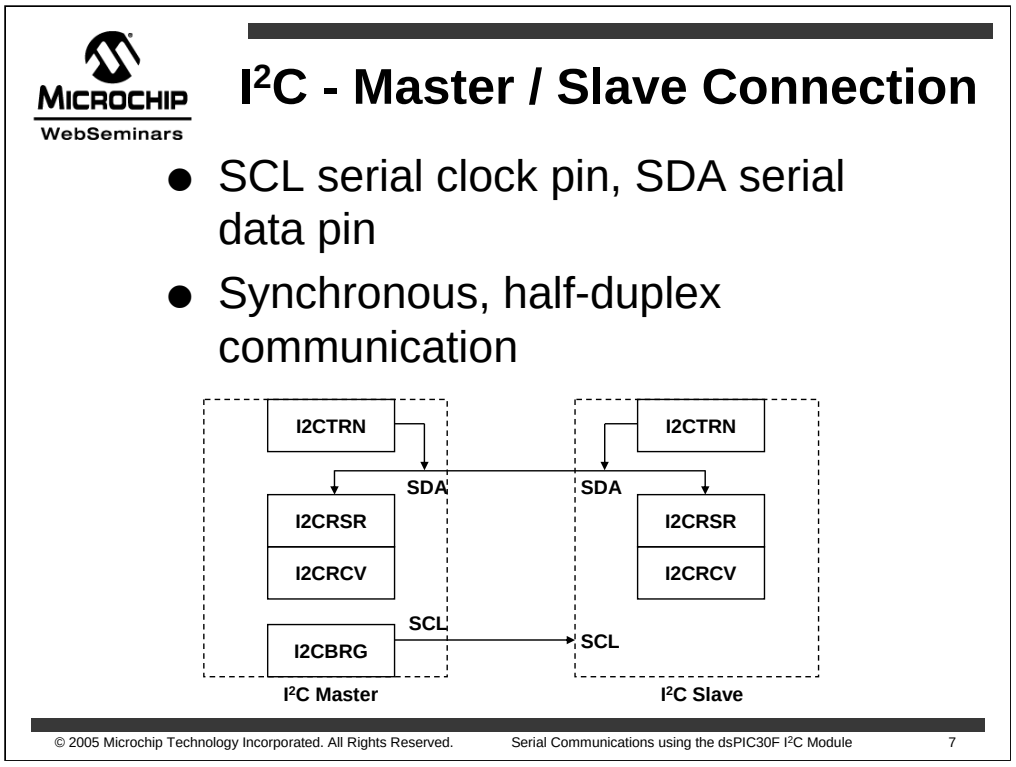
Slave operations are activated only when the Slave logic detects that the Master has addressed it.



The I²C module has a few control and status registers to control and monitor the functioning of the module. These include a Transmit Register and a Receive Register, as well as a Baud Rate Register.

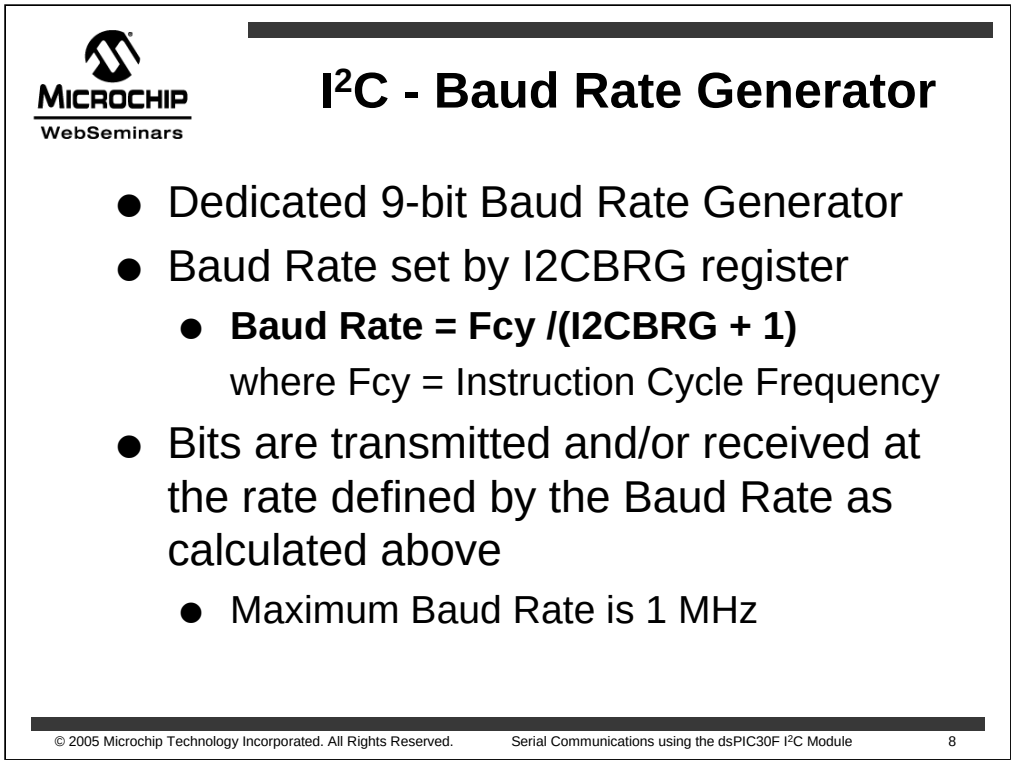
The I²C Control Register has several bits to configure various parameters of the module such as addressing format, as well as to initiate specific real-time operating conditions such as START or STOP. All these bits can be read and written by the user.

The I²C Status Register mainly contains status flag bits that indicate various conditions that occur during a message sequence, as well as certain error conditions like overflows. Most of these bits are read-only.



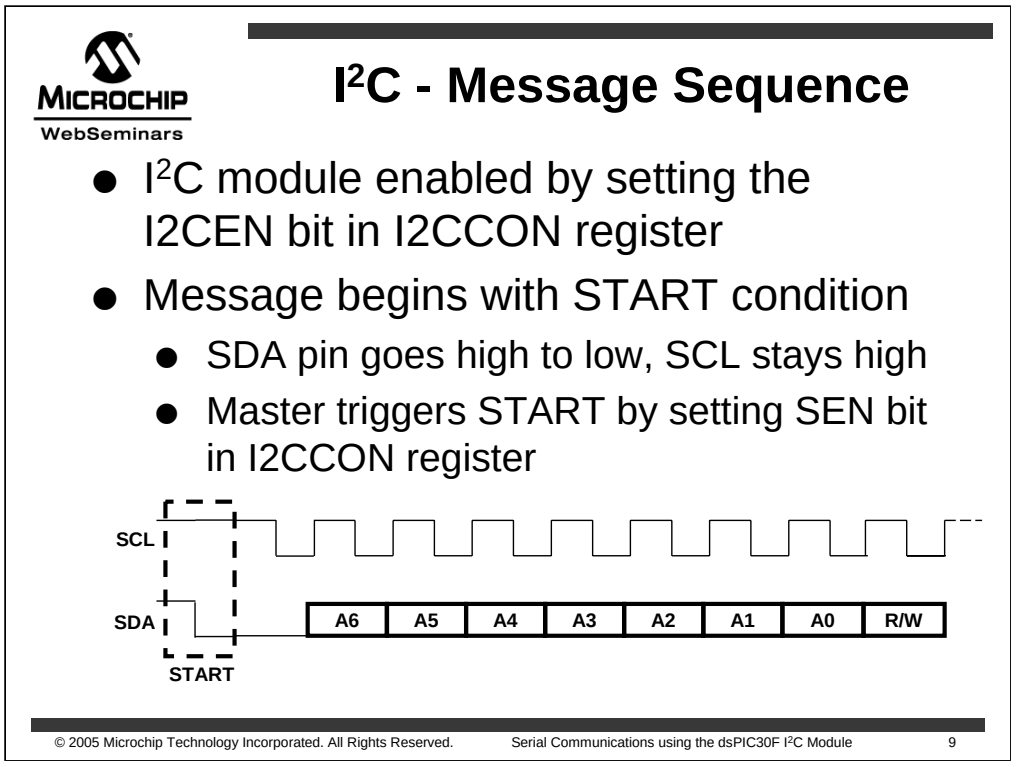
As shown in this figure, two devices communicating through I^2C are inter-connected with their SCL and SDA pins tied together. Depending on the data transfer direction, data is either shifted out through the I^2C Transmit Register, which also serves as a Transmit Shift Register, or shifted in through the I^2C Receive Shift Register.

Data received in the I^2C Receive Shift Register is transferred to the I^2C Receive Register, which is memory-mapped and therefore user-accessible.



The I^2C module contains a dedicated 9-bit Baud Rate Generator, which is used to provide timing synchronization of data transfer and all other I^2C events. The Baud Rate Generator is memory-mapped, and can be configured by the user to generate serial clocks of up to 1 MHz. In I^2C systems, 100 kHz and 400 kHz are the most common bit rates.

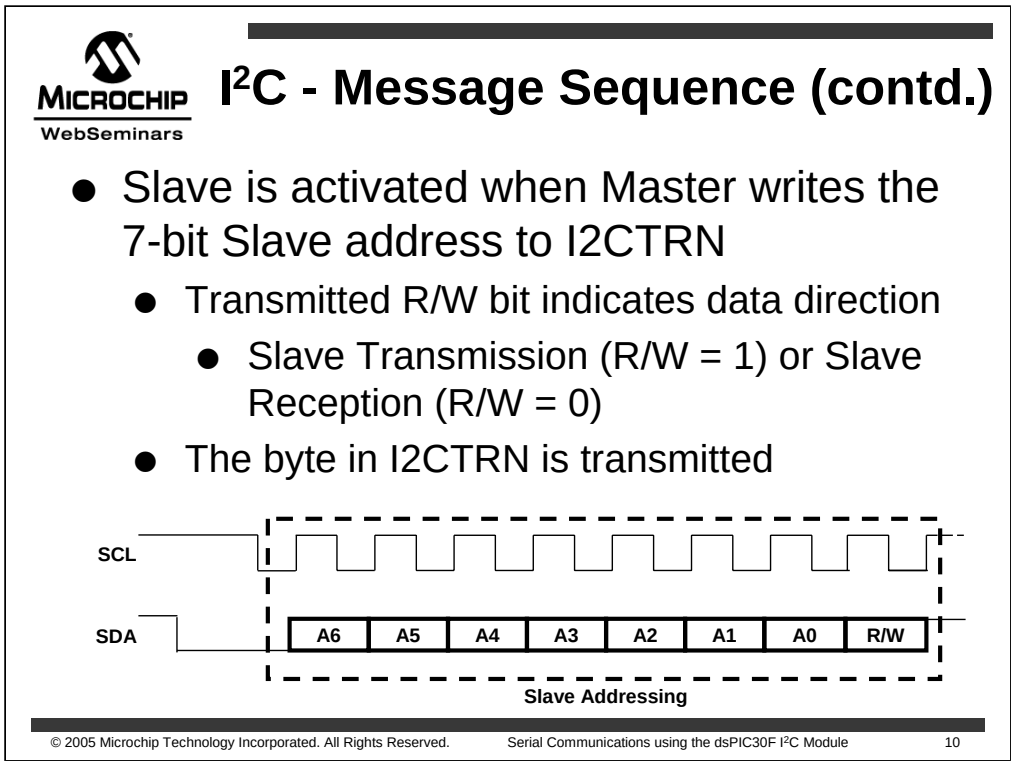
If an I^2C module only performs Slave operations, then the Baud Rate Generator need not be initialized, since a Master in the system would provide the serial clock pulses required for communications.



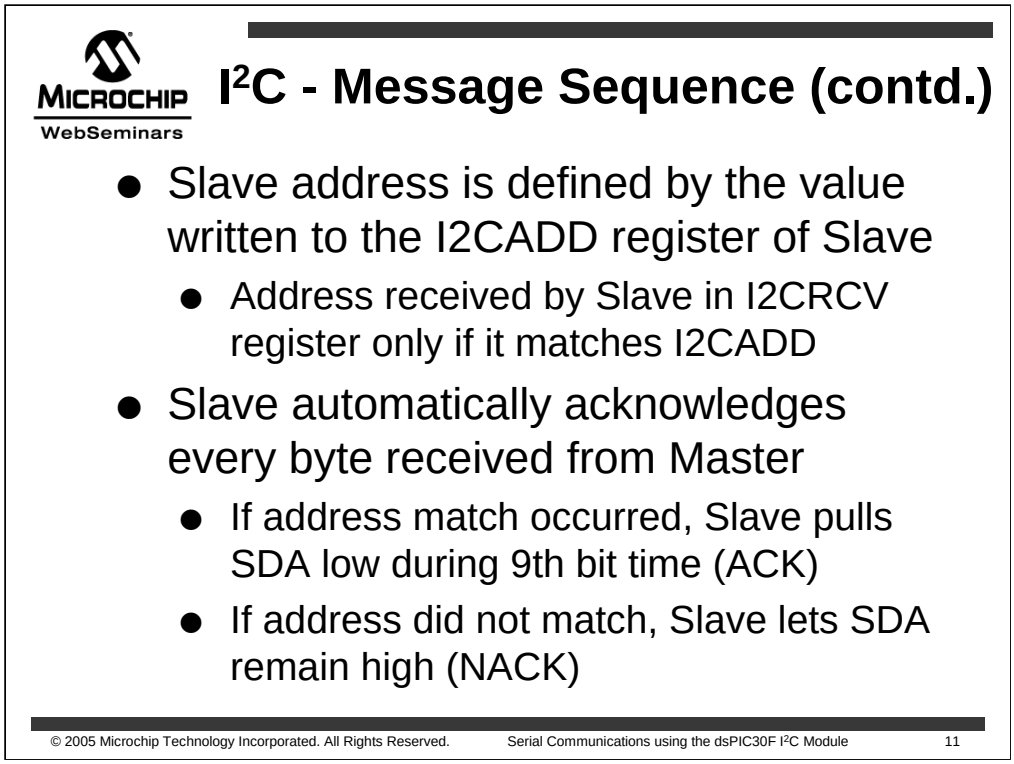
Let us take a look at the various phases of a data transfer over an I²C channel.

Setting the I²C Enable bit in the I²C Control Register enables the module. Subsequently, an I²C device can initiate communications on the Serial Data line by setting the START Enable bit in the I²C Control Register, thereby initiating a Master START sequence.

A START condition is characterized by the Serial Data line going to a low logic level while the Serial clock line is still high. Once a START condition has occurred, the Slave waits for the Master to send a Slave address byte over the Serial Data line.



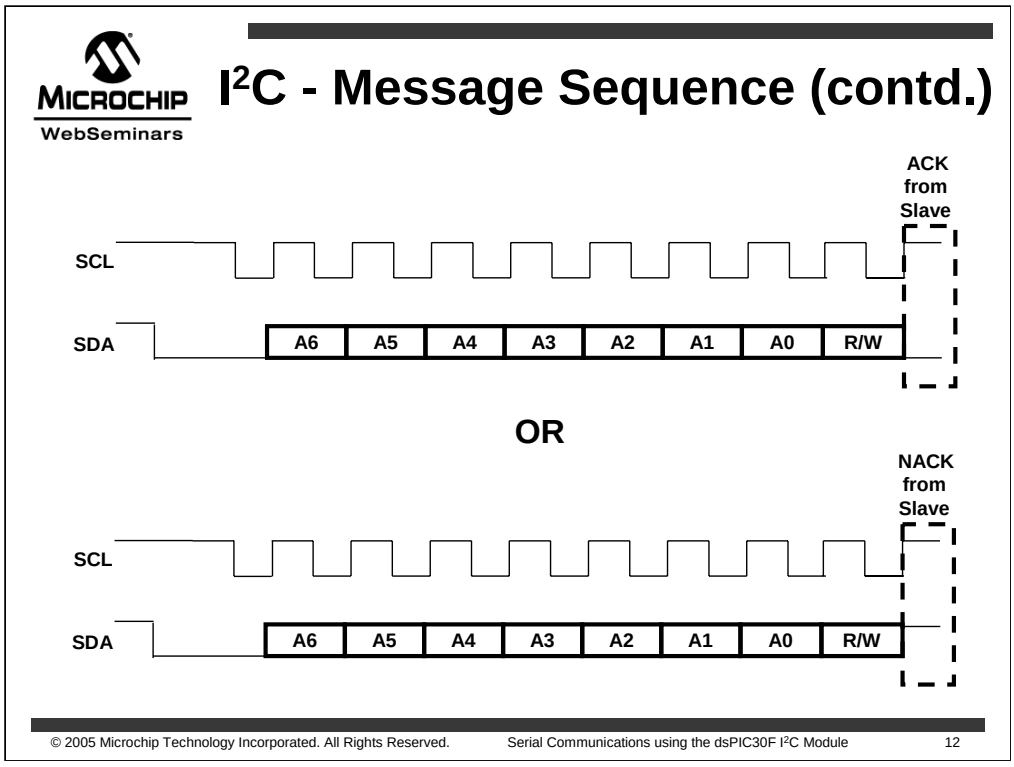
The Master writes the Slave address byte to the I²C Transmit Register. This byte is immediately transmitted, starting with the MSB and ending with the LSB. The address byte consists of 7 address bits, followed by a ‘Read-Write’ bit which signals to the Slave whether the Master wants to initiate a reception or a transmission; in other words, it signals the required data transfer direction. If the Read-Write bit is cleared, it implies that the Master wants to send data to the Slave. If the Read-Write bit is set, the Master is expecting to receive data from the Slave.



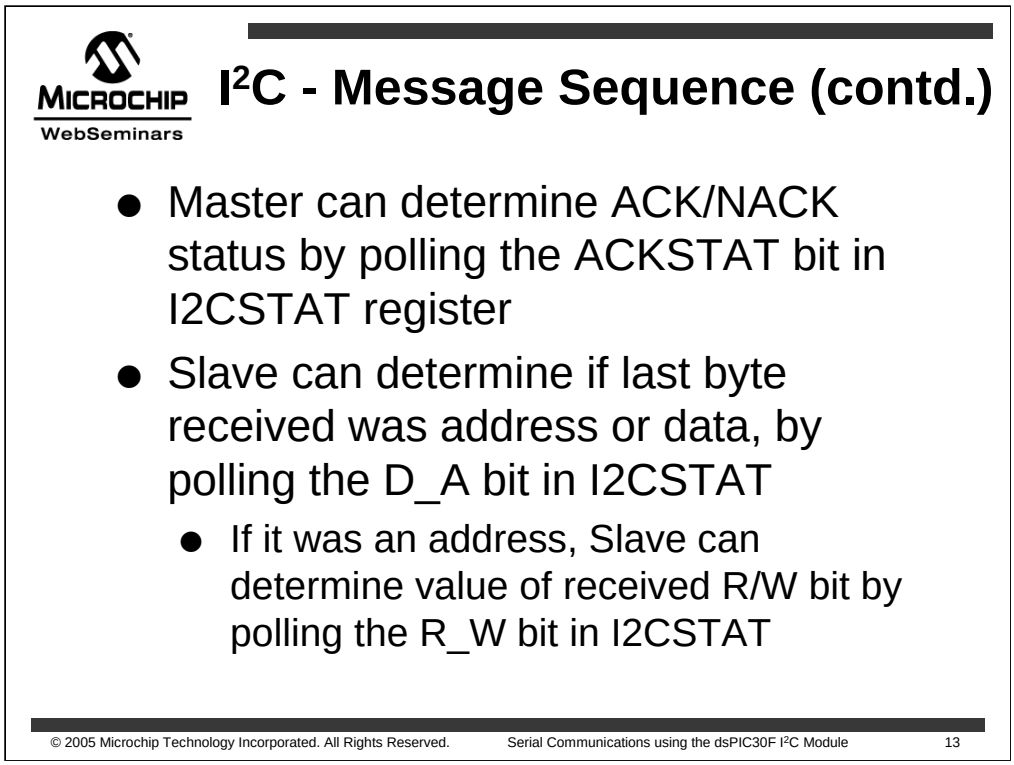
Prior to any communication, the Slave software must assign a Slave address to the device by writing to the I^2C Address Register. During communication, when an address byte is received from the Master, the Slave hardware compares this received address with the I^2C Address Register. If the two addresses match, then the Slave hardware automatically concludes that this particular Slave device is being addressed by the Master.

Whenever an address match occurs, the received address byte is transferred to the user-accessible I^2C Receive Register. During the ninth Serial Clock pulse, the Slave drives the Serial Data line low, which signifies an ‘Acknowledge’ condition.

However, if the received address did not match the pre-assigned I^2C Address Register value, then the Slave allows the Serial Data line to float high, indicating a ‘Not Acknowledge’ condition. In this case, the address byte is not transferred to the I^2C Receive Register, and the Slave hardware continues to wait for an address match.

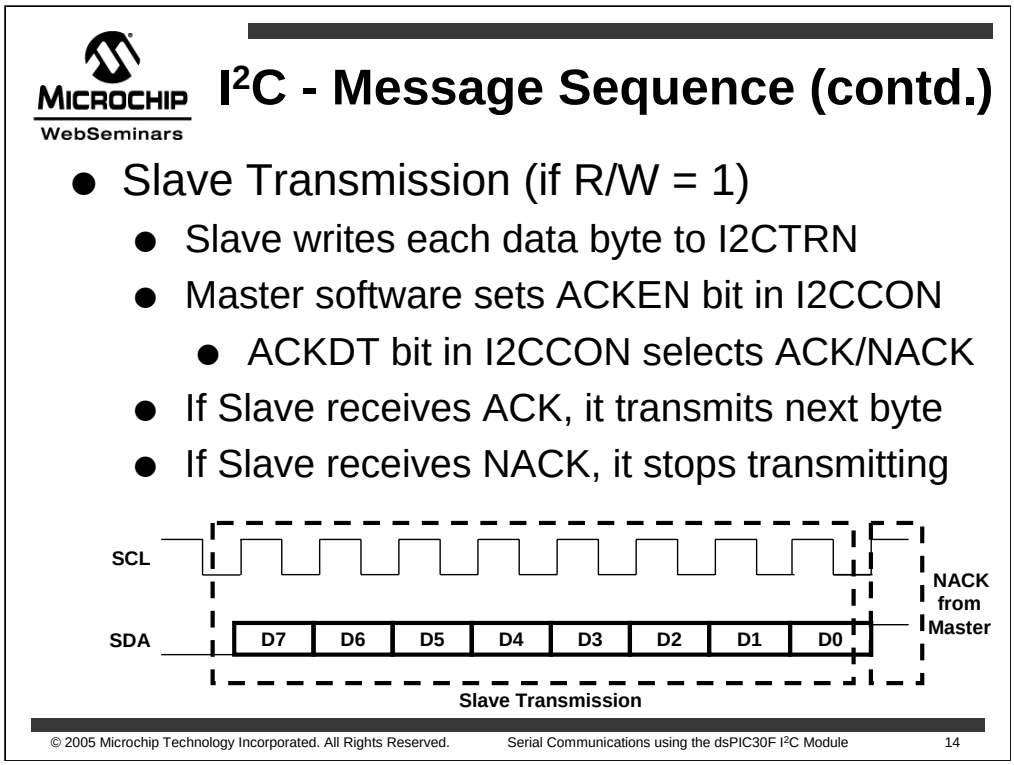


These timing diagrams illustrate the sequence of events that occur during a Slave addressing sequence.



Now, how does the Master ascertain if an Acknowledge was received? It can do so by reading the value of the Acknowledge Status bit in the I^2C Status Register. This bit reflects the state of the Serial Data line during the 9th clock pulse of the Address transmission sequence. Therefore, a value of zero indicates an Acknowledge received from the Slave.

Whenever a byte is transferred to the I^2C Receive Register, the Slave software needs to determine if the byte received was address or data, and if it was an address, whether the Master requested a Read or a Write. This may be accomplished by first reading the Data Or Address bit in the I^2C Status Register. If this bit is found to be cleared, the Slave software should then read the Read-Write bit in the same register. If the Read-Write bit is set, then the Slave prepares to send data to the Master, whereas if the bit is cleared, then the Slave waits for one or more data bytes from the Master.



If a Read-Write bit value of ‘1’ was received from the Master, a Slave Transmission is initiated.

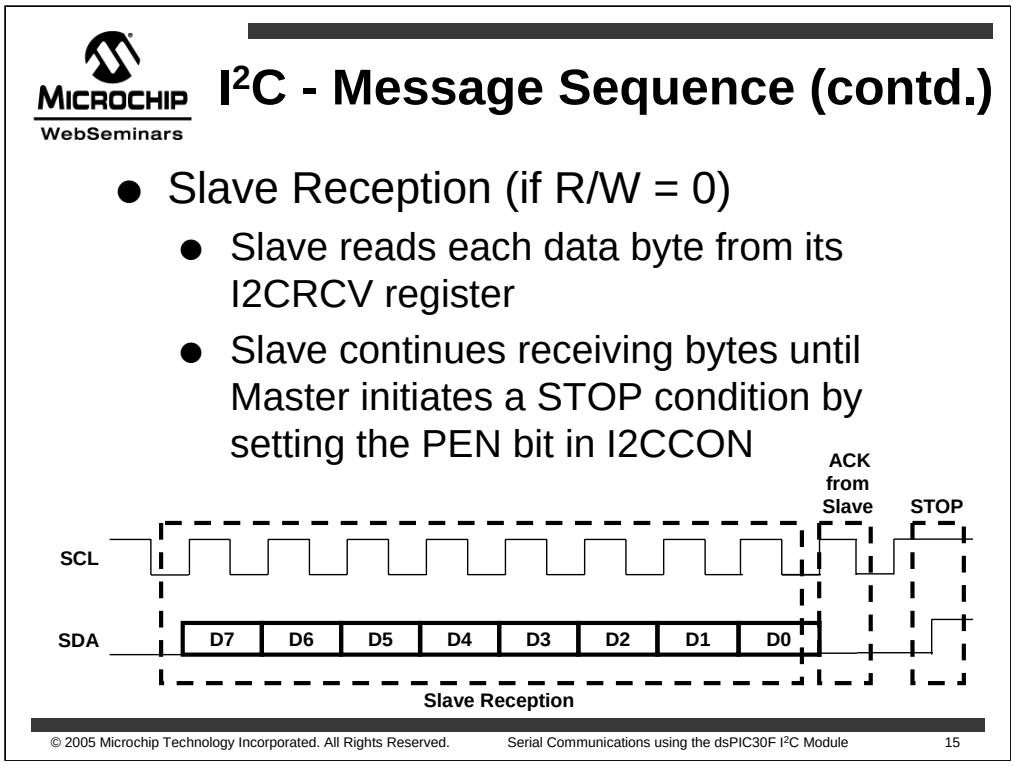
The Slave responds to a Slave Transmission request by writing a data byte to its I²C Transmit Register. The data is shifted out using a series of 8 Serial Clock pulses generated by the Master.

On the 9th Serial Clock pulse, the Master software initiates either an Acknowledge or a Not Acknowledge condition. An Acknowledge condition at this stage indicates that the Master received the data and wants the Slave software to continue transmitting data. A Not Acknowledge condition indicates that the Master wants the Slave software to stop transmitting data.

The Master software can generate an Acknowledge by first clearing the Acknowledge Data bit in the I²C Control Register and then setting the Acknowledge Sequence Enable bit in the same register.

In order to generate a Not Acknowledge, the Master software must first set the Acknowledge Data bit and then set the Acknowledge Sequence Enable bit.

Note that an Acknowledge or Not Acknowledge sequence from the Master is always controlled by software, whereas an Acknowledge or Not Acknowledge sequence from the Slave is automatically generated by the hardware.

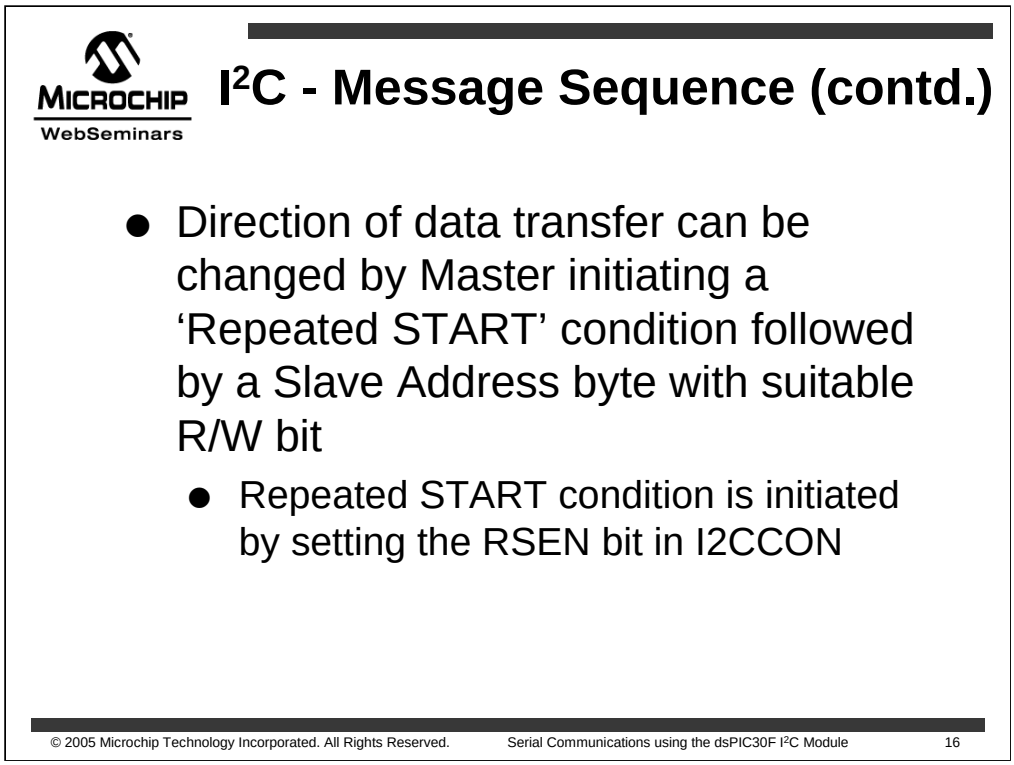


If a Read-Write bit value of ‘0’ was received from the Master, a Slave Reception is initiated.

The Slave responds to a Slave Reception request by simply waiting for the Master to transmit data. The data is shifted in using a series of 8 Serial Clock pulses generated by the Master, and the Slave hardware always generates an Acknowledge condition during the 9th Serial Clock pulse. The received data can be read by the Slave software from the I^2C Receive Register.

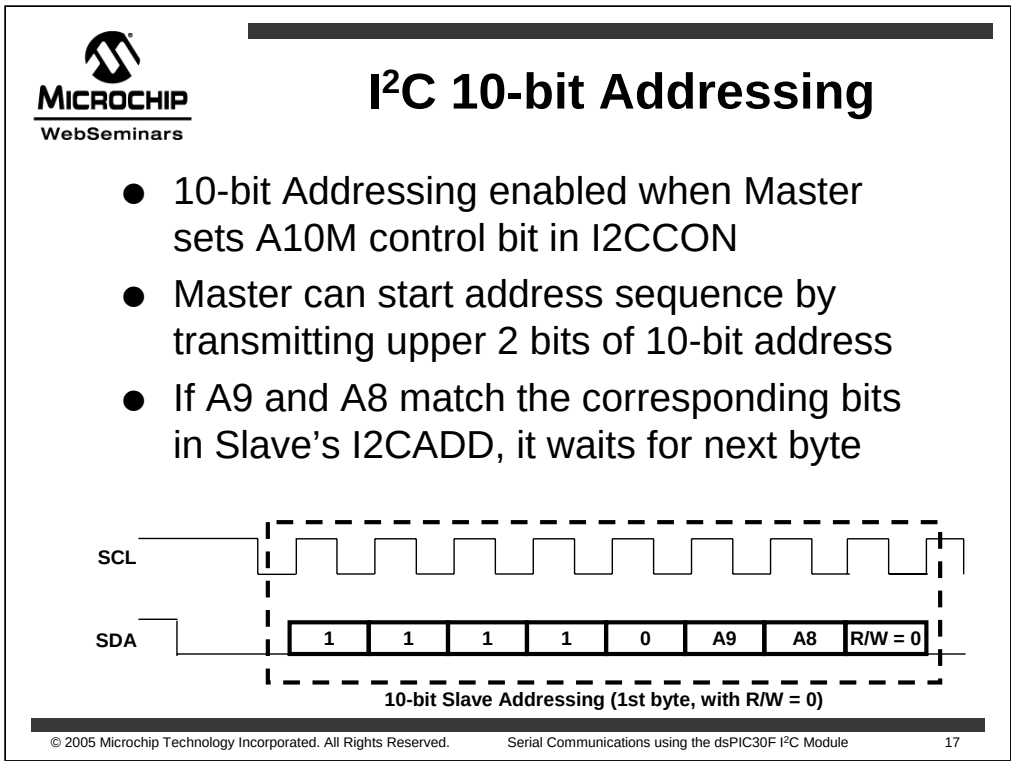
At this point, the Master can continue to send data to the Slave, and the Slave will continue receiving the data. After receiving an Acknowledge from the Slave, if the Master wants to stop transmitting data, then the Master software should initiate a STOP sequence.

A STOP condition is characterized by the Serial Data line going to a high logic level while the Serial Clock line is still high. Once a STOP condition has occurred, the Slave stops expecting data from the Master, and instead, waits for the Master to send a Slave address byte.



At any time during the data transfer sequence, the Master can reverse the direction of data transfer by performing a Repeated Start, or RESTART, condition. This is initiated by setting the Restart Enable bit in the I2CCON register.

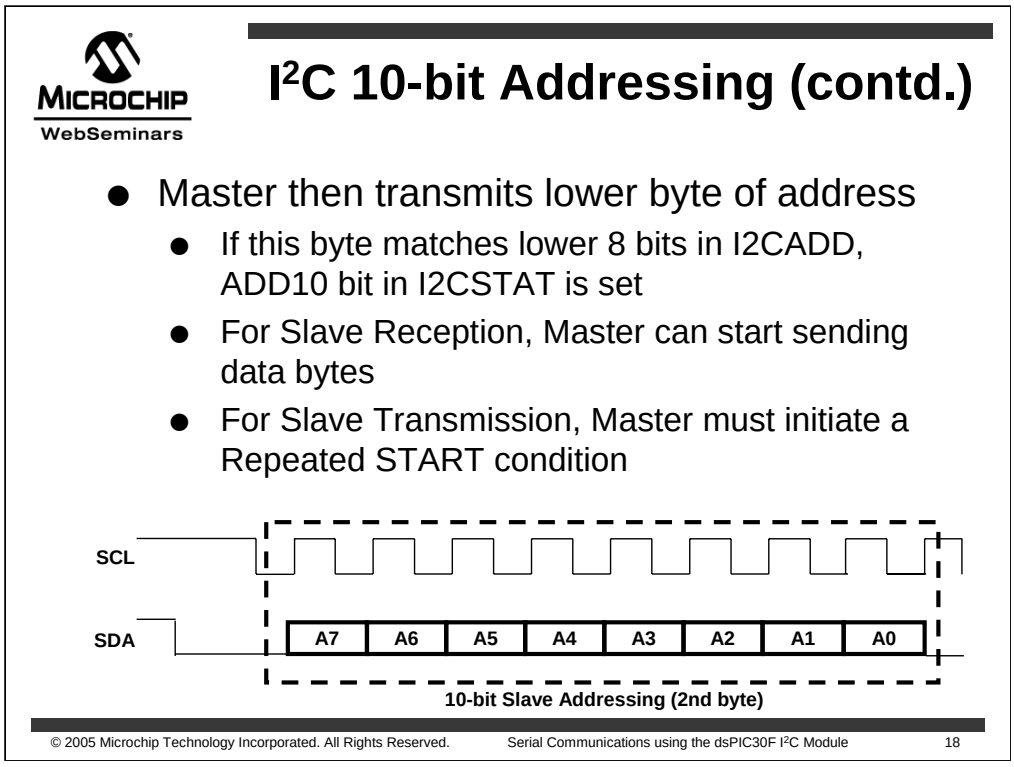
After the RESTART condition is completed, the Master would need to resend the address byte with the required Read-Write bit value.



Besides detection of 7-bit Slave Addresses, the I²C module also supports 10-bit Slave Addressing. This special mode is enabled by setting the 10-bit Slave Address Enable bit in the I²C Control Register. In this case, the Slave needs to initialize its I²C Address Register with a 10-bit value.

An I²C Master can initiate a 10-bit addressing sequence by transmitting a byte which consists of a pre-defined upper address preamble '11110', the upper two bits of the 10-bit Slave address, and a cleared Read-Write bit. The Read-Write bit must be clear at this stage because the address has not been completely transmitted yet.

When the Slave hardware detects this specific '11110' bit pattern, it compares the upper two bits of the received Address with the upper 2 bits of the I²C Address Register contents. If the bits matched, the Slave hardware sends an Acknowledge, otherwise it sends a Not Acknowledge.

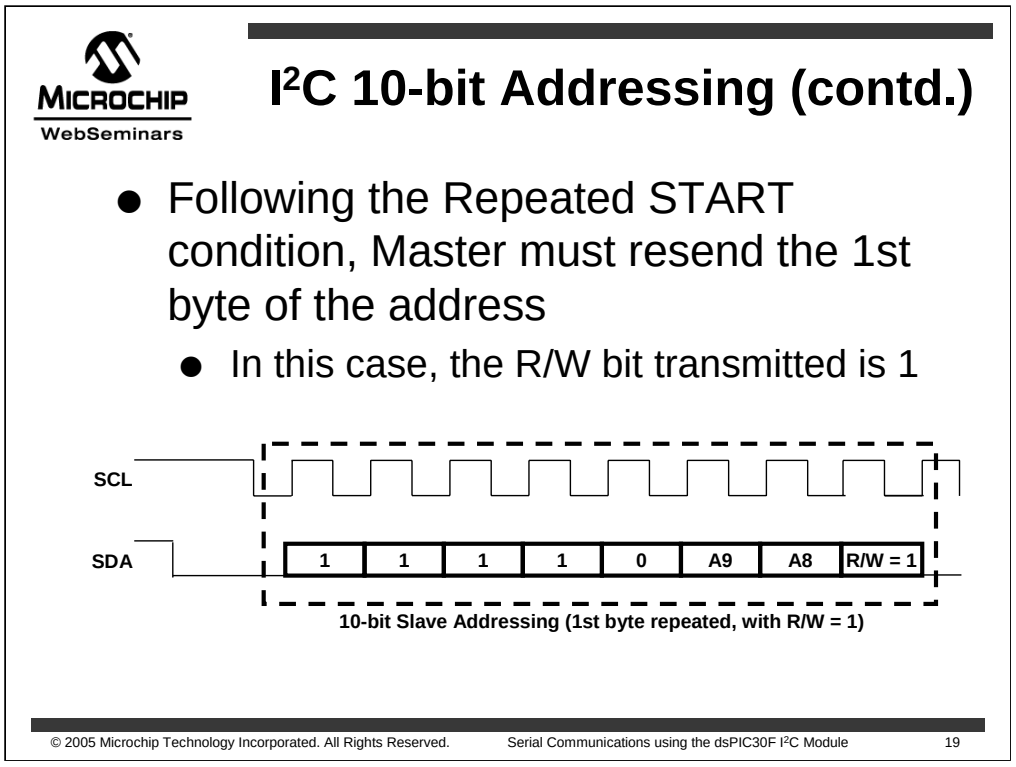


Following the transmission of the first address byte, the Master transmits a second byte, which contains the lower 8 bits of the Slave address.

The Slave hardware compares these 8 bits with the lower 8 bits of the I^2C Address Register, and if a match occurred, it again sends an Acknowledge. Moreover, if all 10 bits of the received address matched those of the I^2C Address Register, then the 10-bit Address Status bit in the I^2C Status Register is set.

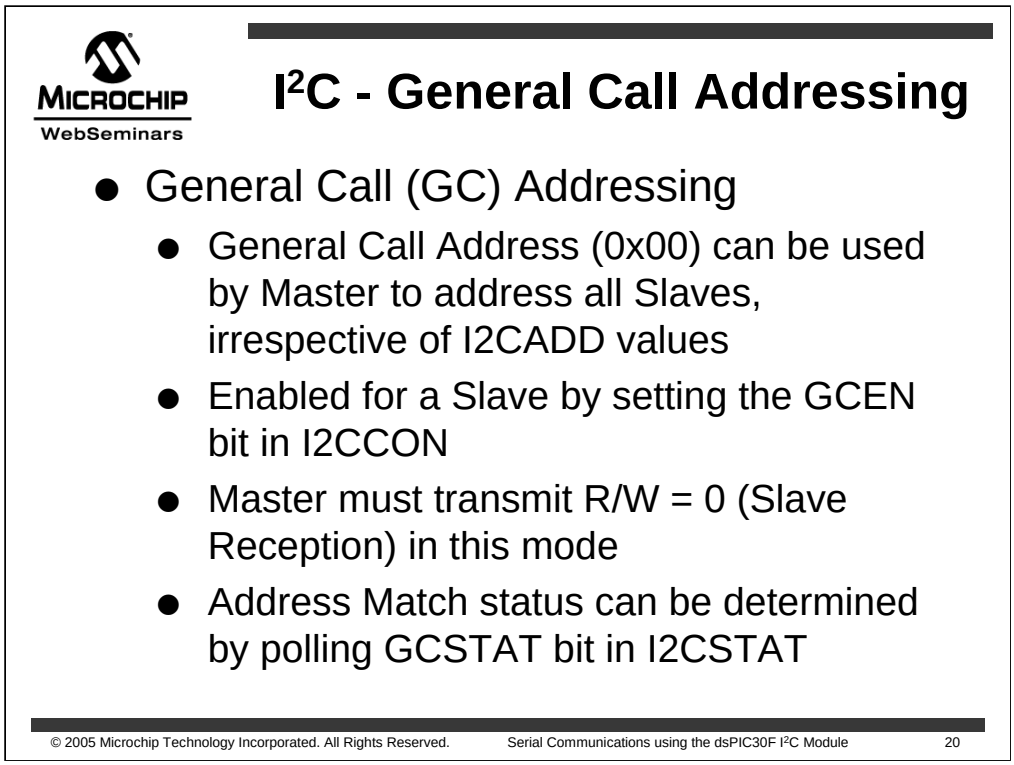
Now, if Slave Reception is required, the Master can simply start transmitting data bytes to the Slave without re-sending either of the address bytes, since a Read-Write bit value of ‘0’ has already been sent.

However, if Slave Transmission is required, the Master is required to initiate a Repeated START sequence, which is used to invert the data transfer direction.



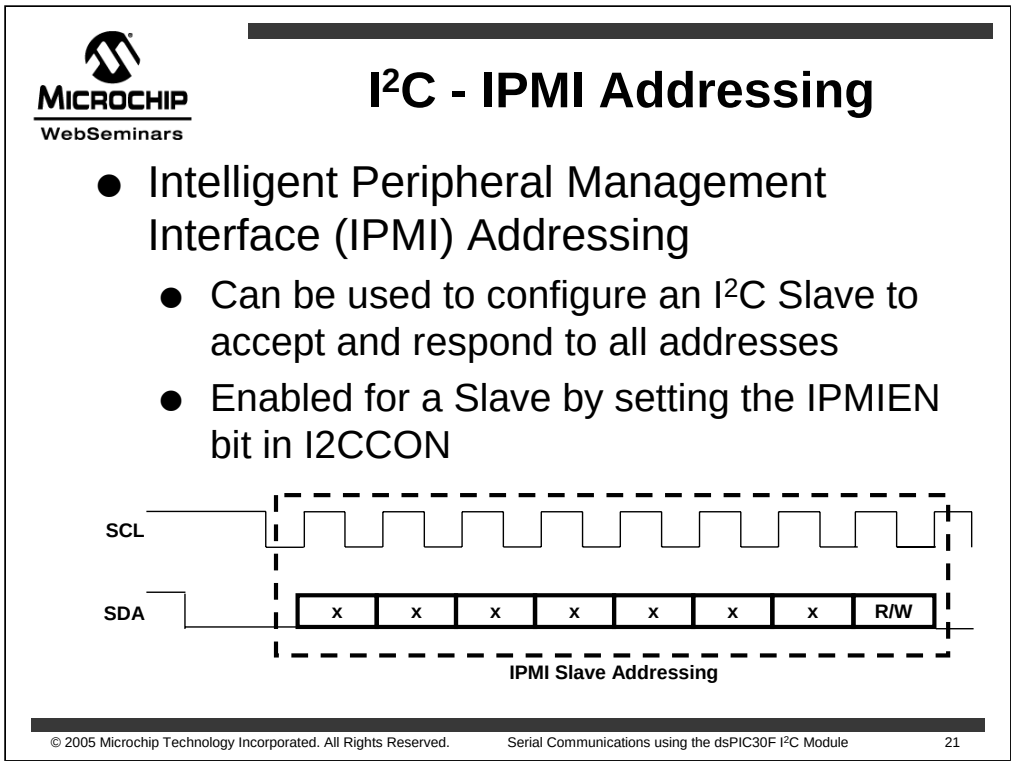
After the Repeated START condition, the Master software should re-transmit only the first Address byte, as shown. However, this time the Read-Write bit should be set, not cleared, in order to request a Slave Transmission.

The transmission or reception of data bytes can now proceed, and continues until the Master generates a STOP condition.



The I^2C module includes support for a special address known as the General Call Address. This address, which consists of seven zeroes and a cleared Read-Write bit, signals to every Slave in the system that it is being addressed, irrespective of the value of its internal I^2C Address Register.

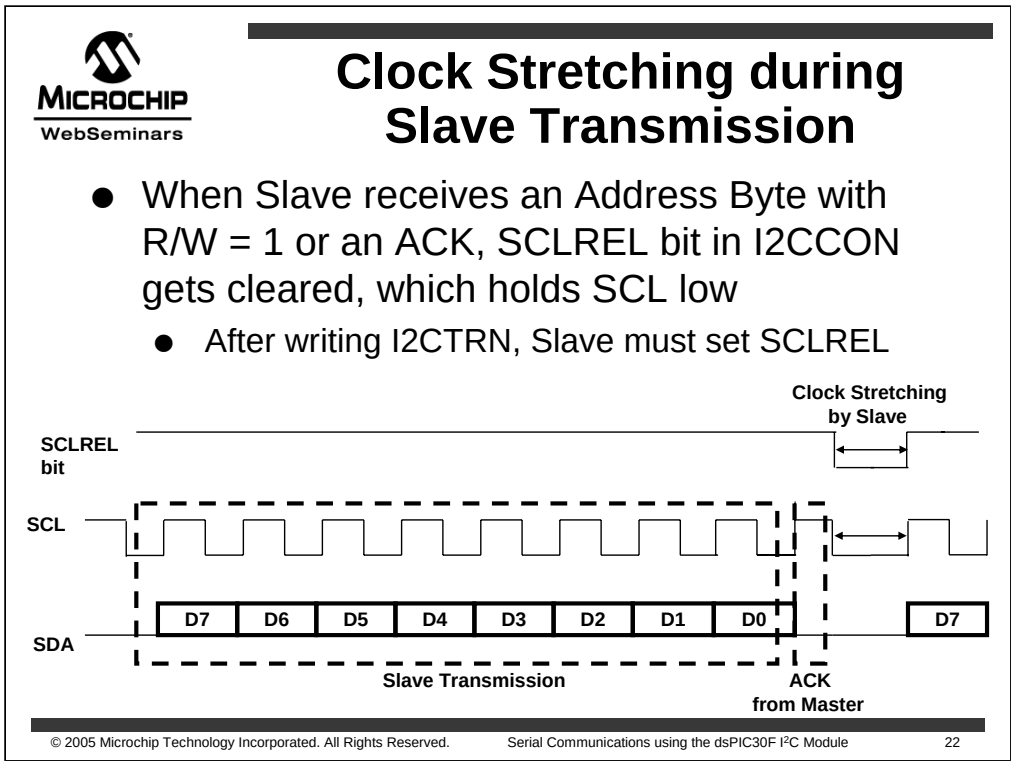
General Call Addressing can be enabled by setting the General Call Enable bit in the I^2C Control Register. When a General Call Address is received by a Slave, an Acknowledge condition is generated and the General Call Status bit in the I^2C Status Register is set.



Another special Slave Addressing feature in the I²C module is the Intelligent Peripheral Management Interface, or IPMI, mode. The IPMI bus uses I²C nodes as message repeaters in a distributed network. To allow a node to repeat all messages, the Slave module must accept all messages, regardless of the device address.

If IPMI mode is enabled by setting the IPMI Enable bit in the I²C Control Register, the Slave acknowledges and responds to any Slave address received, irrespective of the I²C Address Register value.

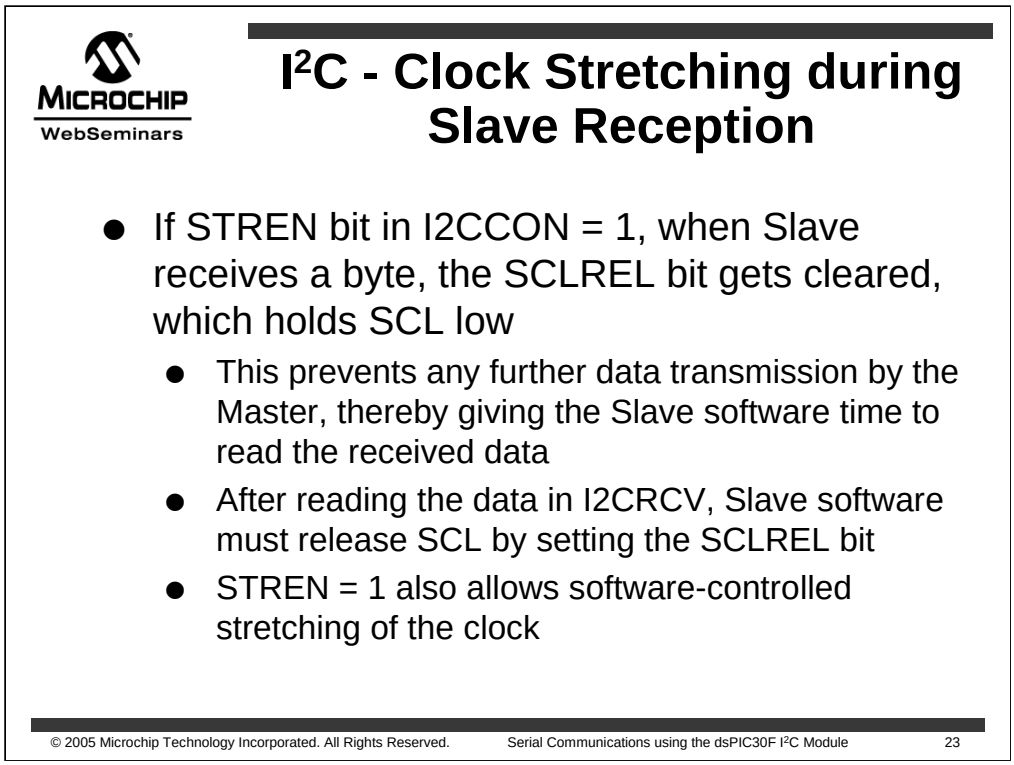
Besides its primary usage in an IPMI bus, this feature can also be used in any system by a device for monitoring messages intended for other recipients.



The I²C module contains a special feature called Clock Stretching. Clock Stretching is a mechanism by which a Slave can temporarily hold the Serial Clock line low, thereby allowing it sufficient time to perform certain operations expected of it.

During Slave Transmission, the Slave hardware always holds the clock line low every time it receives a Slave Transmission request from the Master, and also every time a data byte sent by the Slave is acknowledged by the Master. This indicates to the Master that the Slave is in the process of writing data to its I²C Transmit Register, and that it requires some time to do so. A Clock Stretching condition is indicated by a zero value in the Serial Clock Release bit in the I²C Control Register.

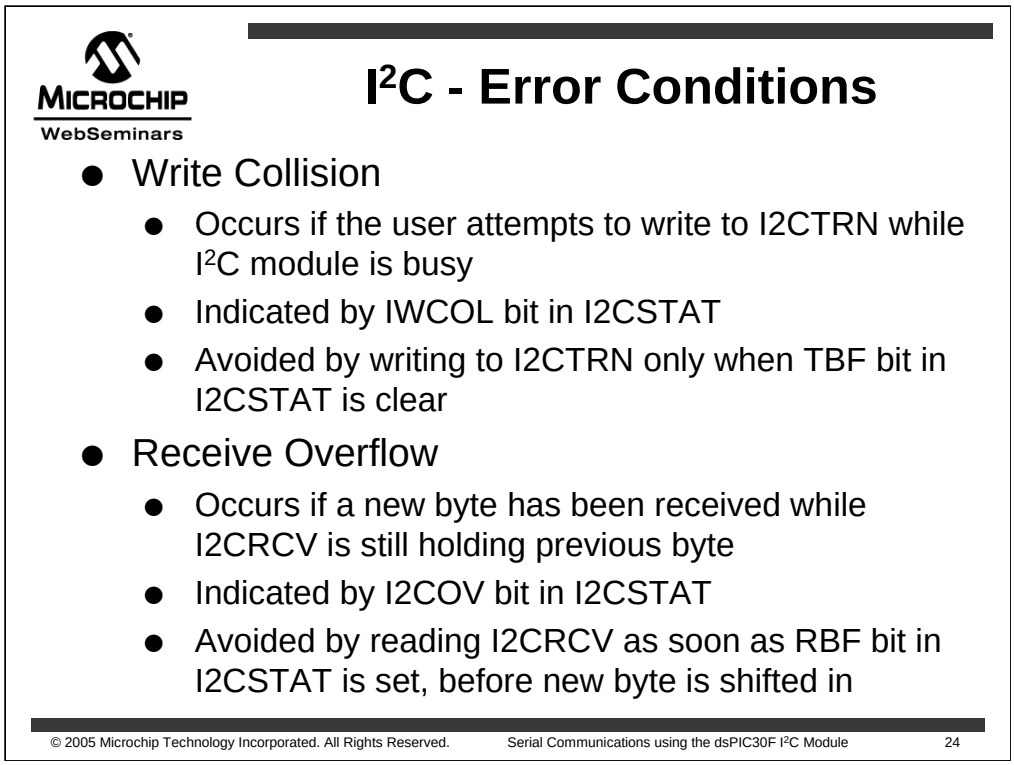
After the Slave software writes to the I²C Transmit Register, it must set the Serial Clock Release bit, thereby releasing the Serial Clock line and allowing data transmission to proceed.



During Slave Reception, the Slave hardware can optionally hold the clock line low every time it receives a data byte in its I^2C Receive Register. This indicates to the Master hardware that the Slave is in the process of reading the received data, and that it requires some time to do so. A Clock Stretching condition is indicated by a zero value in the Serial Clock Release bit in the I^2C Control Register. This optional Clock Stretching during Slave Reception can be enabled by setting the Serial Clock Stretch Enable bit in the I^2C Control Register.

After the Slave software reads the I^2C Receive Register, it must set the Serial Clock Release bit, thereby releasing the Serial Clock line and allowing data transmission to proceed.

Moreover, if the Serial Clock Stretch Enable bit is set, then the Serial Clock can be manually held low at any time by the Slave software, simply by clearing the Serial Clock Release bit.



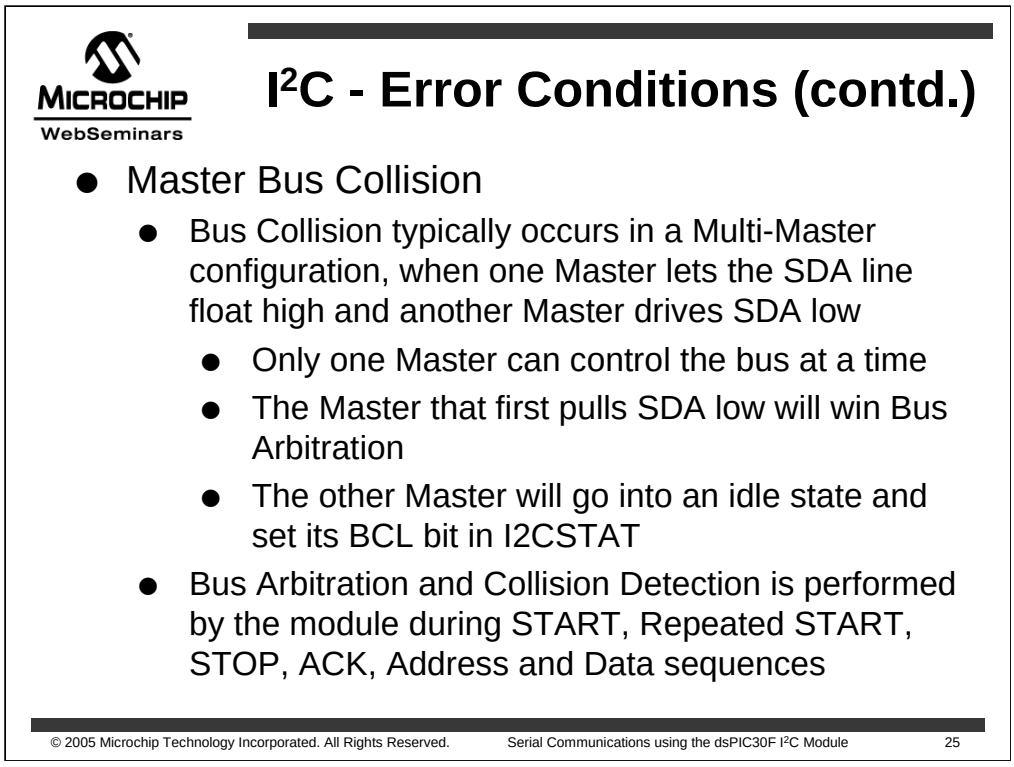
There are various error conditions that are flagged by the I²C module. These include Write Collision, Receive Overflow and Bus Collision.

A Write Collision occurs if the user writes to the I²C Transmit Register while a transmission is still going on. When this happens, the Write Collision Status bit in the I²C Status Register is set, and the attempted write is ignored.

To avoid Write Collision errors, the user should inspect the Transmit Buffer Full bit in the I²C Status Register. The I²C Transmit Register should be written to only when this bit is zero.

A Receive Overflow is generated when the I²C Receive Register contains some received data that has not been read yet, and a new data byte is completely shifted in through the I²C Receive Shift Register. In such cases, the newly received byte is discarded, and the Receive Overflow bit in the I²C Status Register is set.

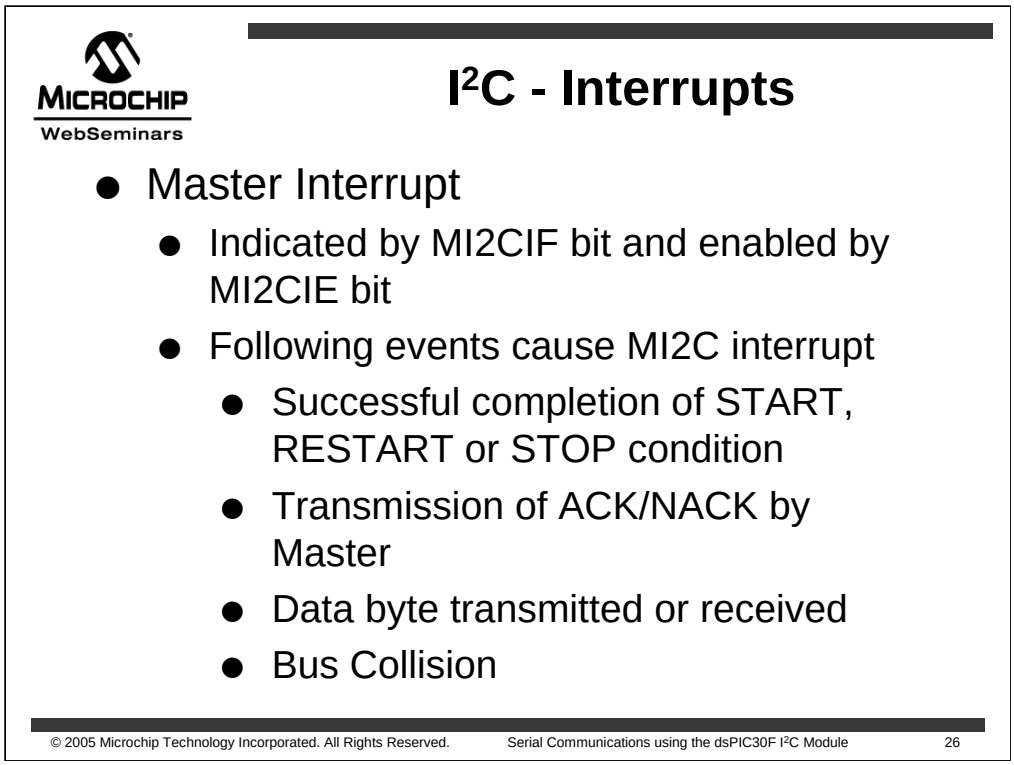
Receive Overflow errors can be avoided by reading the I²C Receive Register as soon as an address or data byte is received, that is, as soon as the Receive Buffer Full bit in the I²C Status Register is set.



The I²C module supports multi-master operation. A key feature that enables multi-master operation to function correctly is detection of Bus Collision errors.

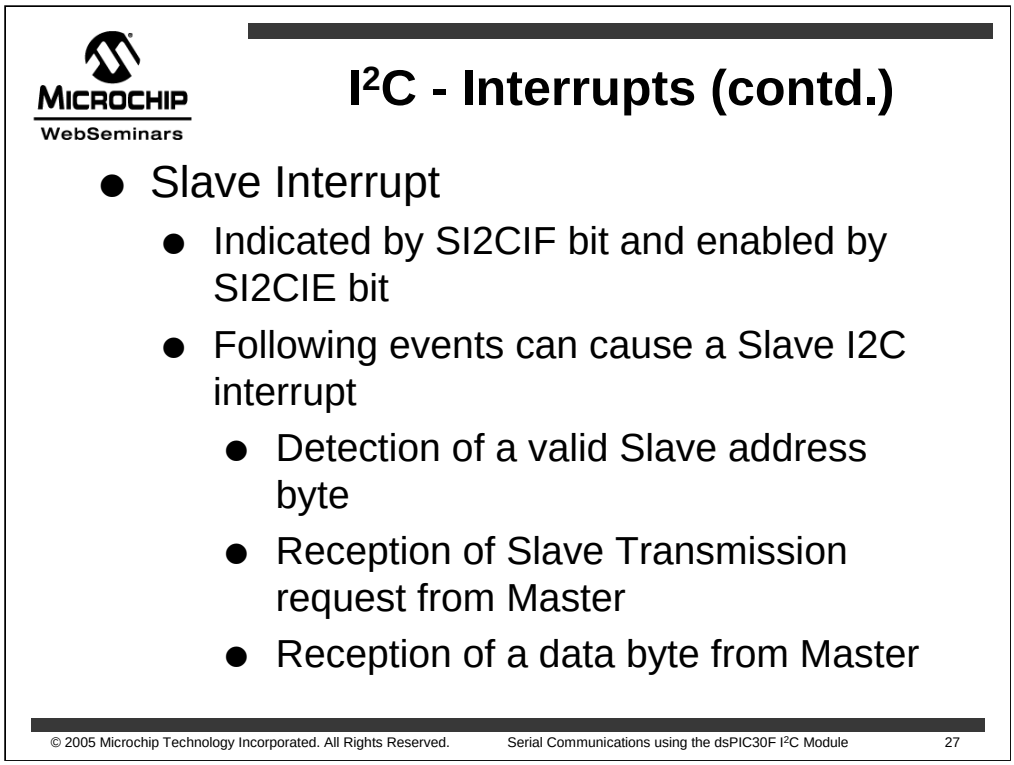
A Bus Collision occurs when one Master in the system lets the Serial Data line float high, but detects a low logic level on the line. This indicates that another Master in the system has already gained control of the bus. The first Master then sets the Bus Collision Status bit in its I²C Status Register. A process known as Bus Arbitration causes the first Master to withdraw from the bus and wait for the bus to be idle before attempting any communication. This ensures robust operation, by allowing only one Master to control the bus at any given time.

Bus Arbitration is performed by the Master hardware at various stages of communication, such as START, Repeated START, STOP, Acknowledge, Slave Addressing and Data Transmission or Reception.

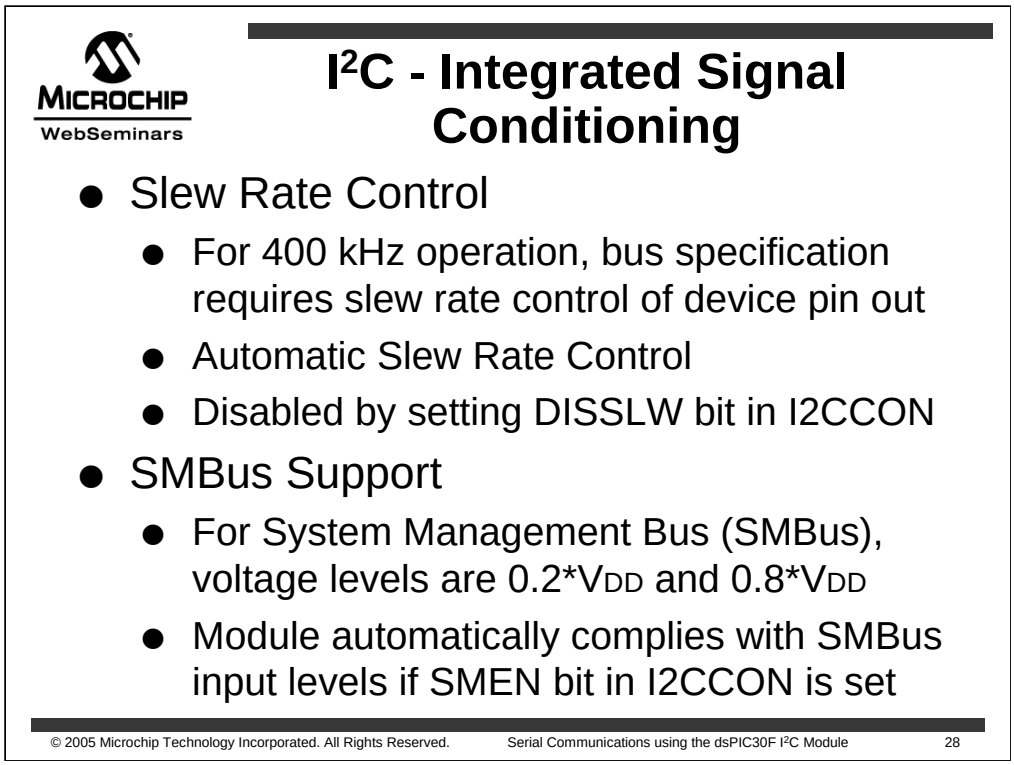


The I^2C module supports two interrupts, Master I^2C Interrupt and Slave I^2C Interrupt. As with other modules in the dsPIC30F, each of these two interrupts is serviced if it is enabled and its priority is higher than the current CPU interrupt priority.

The Master I^2C Interrupt is generated for events related to Master operations. The various events that can cause a Master I^2C Interrupt are listed here. These include events such as data transmission and reception, special sequences such as START, STOP and ACK, as well as Master Bus Collision.



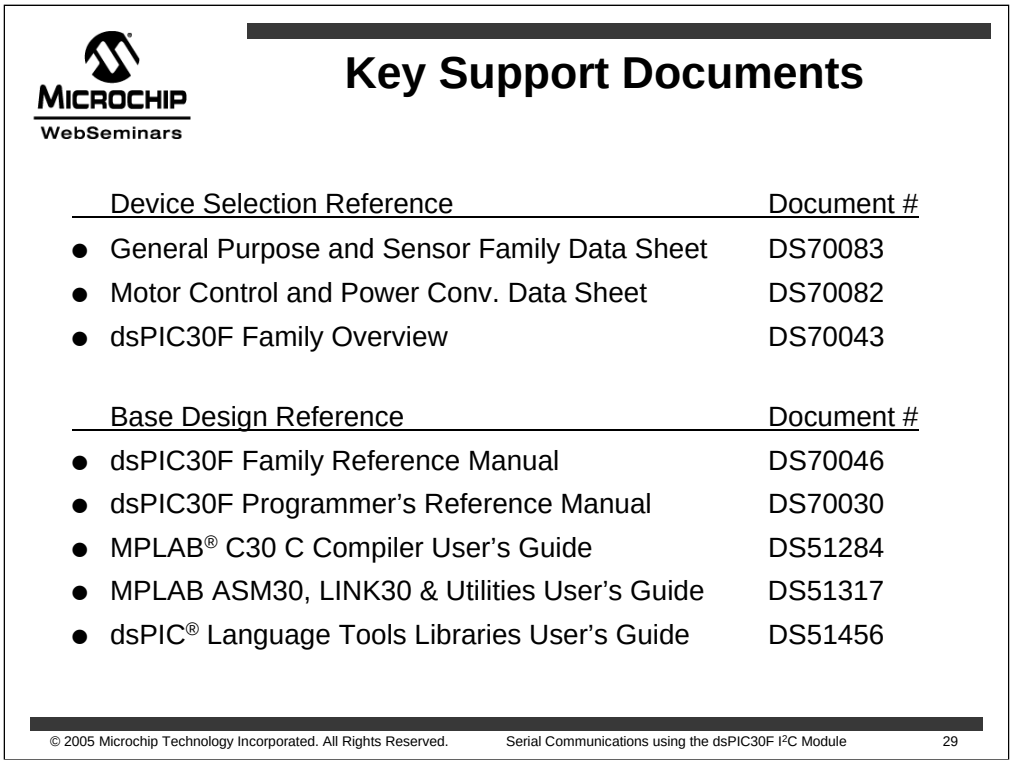
The Slave I^2C Interrupt is generated for events related to Slave operations. Events such as Slave address match and data reception can cause a Slave I^2C Interrupt.



The I^2C hardware includes certain integrated signal conditioning features. For example, the Serial Clock and Data pins have an input filter to prevent glitches in the signal from affecting module operation.

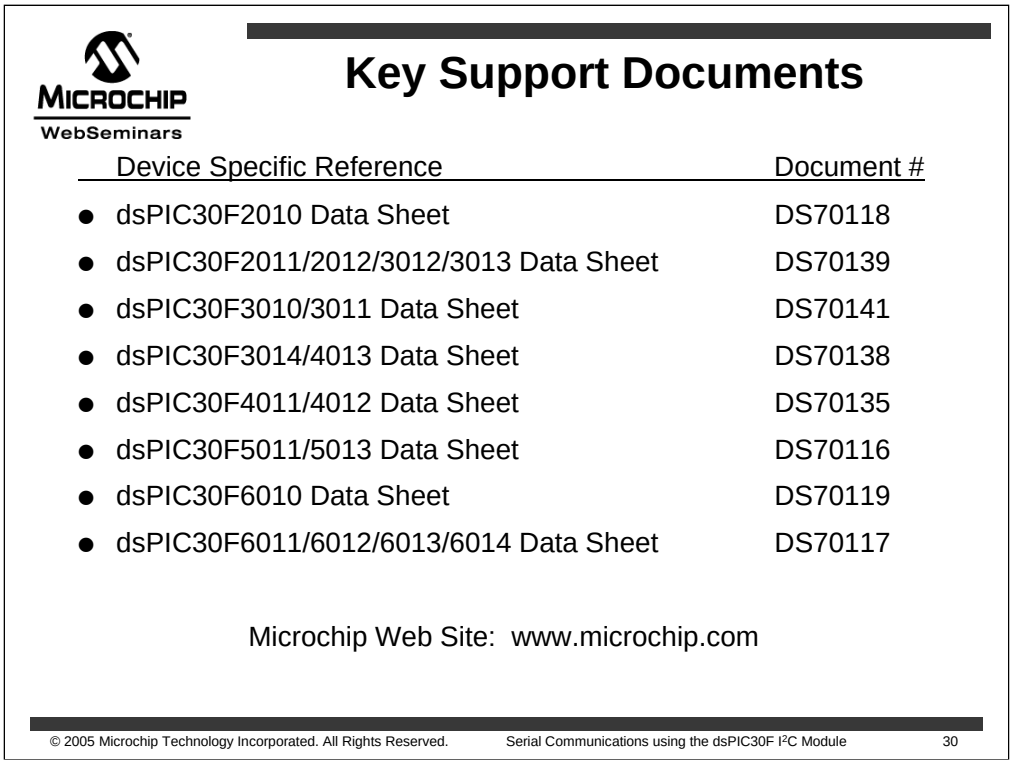
When operating on a 400 kHz bus, the I^2C specification requires slew rate control of the device pin output. This slew rate control is integrated within the dsPIC30F devices. Slew rate control is active when the Disable Slew Rate Control bit in the I^2C Control Register is cleared. For communication speeds other than 400 kHz, the user should set the Disable Slew Rate Control bit, thereby disabling this feature.

A System Management Bus, or SMBus, system requires low and high voltage levels for its inputs that are different from those in a normal I^2C system. The I^2C module can optionally change the input signal levels so that they conform with the SMBus specifications. This signal conditioning feature is enabled by setting the SMBus Input Levels bit in the I^2C Control Register.



For more information, here are references to some important documents that contain a wealth of information about the dsPIC30F family of devices.

The Family Reference Manual contains detailed information about the architecture and peripherals, whereas the Programmer's Reference Manual contains a thorough description of the instruction set.



For device-specific information such as pinout diagrams, packaging and electrical characteristics, the device datasheets listed here are the best source of information.

All these documents can be obtained from the Microchip web site shown, by clicking on the “dsPIC® Digital Signal Controllers” or “Technical Documentation” link.