30 UART

Introduction 565
API Functions 565
Programming Example 589

30.1 Introduction

The Universal Asynchronous Receiver/Transmitter (UART) API provides a set of functions for using the Tiva UART modules. Functions are provided to configure and control the UART modules, to send and receive data, and to manage interrupts for the UART modules.

The Tiva UART performs the functions of parallel-to-serial and serial-to-parallel conversions. It is very similar in functionality to a 16C550 UART, but is not register-compatible.

Some of the features of the Tiva UART are:

- A 16x12 bit receive FIFO and a 16x8 bit transmit FIFO.
- Programmable baud rate generator.
- Automatic generation and stripping of start, stop, and parity bits.
- Line break generation and detection.
- Programmable serial interface
  - 5, 6, 7, or 8 data bits
  - even, odd, stick, or no parity bit generation and detection
  - 1 or 2 stop bit generation
  - baud rate generation, from DC to processor clock/16
- Modem control/flow control
- IrDA serial-IR (SIR) encoder/decoder.
- uDMA interface
- 9-bit operation

This driver is contained in `driverlib/uart.c`, with `driverlib/uart.h` containing the API declarations for use by applications.

30.2 API Functions

Functions

- void UART9BitAddrSend (uint32_t ui32Base, uint8_t ui8Addr)
- void UART9BitAddrSet (uint32_t ui32Base, uint8_t ui8Addr, uint8_t ui8Mask)
- void UART9BitDisable (uint32_t ui32Base)
- void UART9BitEnable (uint32_t ui32Base)
- void UARTBreakCtl (uint32_t ui32Base, bool bBreakState)
- bool UARTBusy (uint32_t ui32Base)

- int32_t UARTCharGet (uint32_t ui32Base)
- int32_t UARTCharGetNonBlocking (uint32_t ui32Base)
- void UARTCharPut (uint32_t ui32Base, unsigned char ucData)
- bool UARTCharPutNonBlocking (uint32_t ui32Base, unsigned char ucData)
- bool UARTCharsAvail (uint32_t ui32Base)
- uint32_t UARTClockSourceGet (uint32_t ui32Base)
- void UARTClockSourceSet (uint32_t ui32Base, uint32_t ui32Source)
- void UARTConfigGetExpClk (uint32_t ui32Base, uint32_t ui32UARTClk, uint32_t *pui32Baud, uint32_t *pui32Config)
- void UARTConfigSetExpClk (uint32_t ui32Base, uint32_t ui32UARTClk, uint32_t ui32Baud, uint32_t ui32Config)
- void UARTDisable (uint32_t ui32Base)
- void UARTDisableSIR (uint32_t ui32Base)
- void UARTDMADisable (uint32_t ui32Base, uint32_t ui32DMAFlags)
- void UARTDMAEnable (uint32_t ui32Base, uint32_t ui32DMAFlags)
- void UARTEnable (uint32_t ui32Base)
- void UARTEnableSIR (uint32_t ui32Base, bool bLowPower)
- void UARTFIFODisable (uint32_t ui32Base)
- void UARTFIFOEnable (uint32_t ui32Base)
- void UARTFIFOLevelGet (uint32_t ui32Base, uint32_t *pui32TxLevel, uint32_t *pui32RxLevel)
- void UARTFIFOLevelSet (uint32_t ui32Base, uint32_t ui32TxLevel, uint32_t ui32RxLevel)
- uint32_t UARTFlowControlGet (uint32_t ui32Base)
- void UARTFlowControlSet (uint32_t ui32Base, uint32_t ui32Mode)
- void UARTIntClear (uint32_t ui32Base, uint32_t ui32IntFlags)
- void UARTIntDisable (uint32_t ui32Base, uint32_t ui32IntFlags)
- void UARTIntEnable (uint32_t ui32Base, uint32_t ui32IntFlags)
- void UARTIntRegister (uint32_t ui32Base, void (*pfnHandler)(void))
- uint32_t UARTIntStatus (uint32_t ui32Base, bool bMasked)
- void UARTIntUnregister (uint32_t ui32Base)
- void UARTLoopbackEnable (uint32_t ui32Base)
- void UARTModemControlClear (uint32_t ui32Base, uint32_t ui32Control)
- uint32_t UARTModemControlGet (uint32_t ui32Base)
- void UARTModemControlSet (uint32_t ui32Base, uint32_t ui32Control)
- uint32_t UARTModemStatusGet (uint32_t ui32Base)
- uint32_t UARTParityModeGet (uint32_t ui32Base)
- void UARTParityModeSet (uint32_t ui32Base, uint32_t ui32Parity)
- void UARTRxErrorClear (uint32_t ui32Base)
- uint32_t UARTRxErrorGet (uint32_t ui32Base)
- void UARTSmartCardDisable (uint32_t ui32Base)
- void UARTSmartCardEnable (uint32_t ui32Base)
- bool UARTSpaceAvail (uint32_t ui32Base)
- uint32_t UARTTxIntModeGet (uint32_t ui32Base)
- void UARTTxIntModeSet (uint32_t ui32Base, uint32_t ui32Mode)

30.2.1 Detailed Description

The UART API provides the set of functions required to implement an interrupt-driven UART driver. These functions may be used to control any of the available UART ports on a Tiva microcontroller and can be used with one port without causing conflicts with the other port.

The UART API is broken into three groups of functions: those that deal with configuration and control of the UART modules, those used to send and receive data, and those that deal with interrupt handling.

The clock source for the baud rate generator is handled by the UARTClockSourceSet() and UARTClockSourceGet() functions.

Configuration and control of the UART are handled by the UARTConfigGetExpClk(), UARTConfigSetExpClk(), UARTDisable(), UARTEnable(), UARTParityModeGet(), and UARTParityModeSet() functions. The DMA interface can be enabled or disabled by the UARTDMAEnable() and UARTDMADisable() functions.

Sending and receiving data via the UART is handled by the UARTCharGet(), UARTCharGetNonBlocking(), UARTCharPut(), UARTCharPutNonBlocking(), UARTBreakCtl(), UARTCharsAvail(), and UARTSpaceAvail() functions.

Managing the UART interrupts is handled by the UARTIntClear(), UARTIntDisable(), UARTIntEnable(), UARTIntRegister(), UARTIntStatus(), and UARTIntUnregister() functions.

The 9-bit operation mode is handled by the UART9BitEnable(), UART9BitDisable(), UART9BitAddrSet(), and UART9BitAddrSend() functions.

The UARTConfigSet(), UARTConfigGet(), UARTCharNonBlockingGet(), and UARTCharNonBlockingPut() APIs from previous versions of the peripheral driver library have been replaced by the UARTConfigSetExpClk(), UARTConfigGetExpClk(), UARTCharGetNonBlocking(), and UARTCharPutNonBlocking() APIs, respectively. Macros have been provided in `uart.h` to map the old APIs to the new APIs, allowing existing applications to link and run with the new APIs. It is recommended that new applications utilize the new APIs in favor of the old ones.

30.2.2 Function Documentation

30.2.2.1 UART9BitAddrSend

Sends an address character from the specified port when operating in 9-bit mode.

Prototype:

```
void
UART9BitAddrSend(uint32_t ui32Base,
                 uint8_t ui8Addr)
```

Parameters:

ui32Base is the base address of the UART port.

ui8Addr is the address to be transmitted.

Description:

This function waits until all data has been sent from the specified port and then sends the given address as an address byte. It then waits until the address byte has been transmitted before returning.

The normal data functions (UARTCharPut(), UARTCharPutNonBlocking(), UARTCharGet(), and UARTCharGetNonBlocking()) are used to send and receive data characters in 9-bit mode.

Note:
The availability of 9-bit mode varies with the Tiva part in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:
None.

30.2.2.2 UART9BitAddrSet

Sets the device address(es) for 9-bit mode.

Prototype:
```
void
UART9BitAddrSet(uint32_t ui32Base,
                uint8_t ui8Addr,
                uint8_t ui8Mask)
```

Parameters:
ui32Base is the base address of the UART port.
ui8Addr is the device address.
ui8Mask is the device address mask.

Description:
This function configures the device address or range of device addresses that respond to requests on the 9-bit UART port. The received address is masked with the mask and then compared against the given address, allowing either a single address (if **ui8Mask** is 0xff) or a set of addresses to be matched.

Note:
The availability of 9-bit mode varies with the Tiva part in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:
None.

30.2.2.3 UART9BitDisable

Disables 9-bit mode on the specified UART.

Prototype:
```
void
UART9BitDisable(uint32_t ui32Base)
```

Parameters:
ui32Base is the base address of the UART port.

Description:
This function disables the 9-bit operational mode of the UART.

Note:
The availability of 9-bit mode varies with the Tiva part in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:
None.

30.2.2.4 UART9BitEnable

Enables 9-bit mode on the specified UART.

Prototype:
```
void
UART9BitEnable(uint32_t ui32Base)
```

Parameters:
ui32Base is the base address of the UART port.

Description:
This function enables the 9-bit operational mode of the UART.

Note:
The availability of 9-bit mode varies with the Tiva part in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:
None.

30.2.2.5 UARTBreakCtl

Causes a BREAK to be sent.

Prototype:
```
void
UARTBreakCtl(uint32_t ui32Base,
             bool bBreakState)
```

Parameters:
ui32Base is the base address of the UART port.
bBreakState controls the output level.

Description:
Calling this function with *bBreakState* set to **true** asserts a break condition on the UART. Calling this function with *bBreakState* set to **false** removes the break condition. For proper transmission of a break command, the break must be asserted for at least two complete frames.

Returns:
None.

30.2.2.6 UARTBusy

Determines whether the UART transmitter is busy or not.

Prototype:

```
bool
UARTBusy(uint32_t ui32Base)
```

Parameters:

ui32Base is the base address of the UART port.

Description:

This function allows the caller to determine whether all transmitted bytes have cleared the transmitter hardware. If **false** is returned, the transmit FIFO is empty and all bits of the last transmitted character, including all stop bits, have left the hardware shift register.

Returns:

Returns **true** if the UART is transmitting or **false** if all transmissions are complete.

30.2.2.7 UARTCharGet

Waits for a character from the specified port.

Prototype:

```
int32_t
UARTCharGet(uint32_t ui32Base)
```

Parameters:

ui32Base is the base address of the UART port.

Description:

This function gets a character from the receive FIFO for the specified port. If there are no characters available, this function waits until a character is received before returning.

Returns:

Returns the character read from the specified port, cast as a *int32_t*.

30.2.2.8 UARTCharGetNonBlocking

Receives a character from the specified port.

Prototype:

```
int32_t
UARTCharGetNonBlocking(uint32_t ui32Base)
```

Parameters:

ui32Base is the base address of the UART port.

Description:

This function gets a character from the receive FIFO for the specified port.

Returns:

Returns the character read from the specified port, cast as a *int32_t*. A **-1** is returned if there are no characters present in the receive FIFO. The UARTCharsAvail() function should be called before attempting to call this function.

30.2.2.9 UARTCharPut

Waits to send a character from the specified port.

Prototype:

```
void
UARTCharPut(uint32_t ui32Base,
            unsigned char ucData)
```

Parameters:

ui32Base is the base address of the UART port.

ucData is the character to be transmitted.

Description:

This function sends the character *ucData* to the transmit FIFO for the specified port. If there is no space available in the transmit FIFO, this function waits until there is space available before returning.

Returns:

None.

30.2.2.10 UARTCharPutNonBlocking

Sends a character to the specified port.

Prototype:

```
bool
UARTCharPutNonBlocking(uint32_t ui32Base,
                       unsigned char ucData)
```

Parameters:

ui32Base is the base address of the UART port.

ucData is the character to be transmitted.

Description:

This function writes the character *ucData* to the transmit FIFO for the specified port. This function does not block, so if there is no space available, then a **false** is returned and the application must retry the function later.

Returns:

Returns **true** if the character was successfully placed in the transmit FIFO or **false** if there was no space available in the transmit FIFO.

30.2.2.11 UARTCharsAvail

Determines if there are any characters in the receive FIFO.

Prototype:

```
bool
UARTCharsAvail(uint32_t ui32Base)
```

Parameters:

ui32Base is the base address of the UART port.

Description:

This function returns a flag indicating whether or not there is data available in the receive FIFO.

Returns:

Returns **true** if there is data in the receive FIFO or **false** if there is no data in the receive FIFO.

30.2.2.12 UARTClockSourceGet

Gets the baud clock source for the specified UART.

Prototype:

```
uint32_t
UARTClockSourceGet(uint32_t ui32Base)
```

Parameters:

ui32Base is the base address of the UART port.

Description:

This function returns the baud clock source for the specified UART. The possible baud clock source are the system clock (**UART_CLOCK_SYSTEM**) or the precision internal oscillator (**UART_CLOCK_PIOSC**).

Note:

The ability to specify the UART baud clock source varies with the Tiva part in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:

None.

30.2.2.13 UARTClockSourceSet

Sets the baud clock source for the specified UART.

Prototype:

```
void
UARTClockSourceSet(uint32_t ui32Base,
                   uint32_t ui32Source)
```

Parameters:

ui32Base is the base address of the UART port.

ui32Source is the baud clock source for the UART.

Description:

This function allows the baud clock source for the UART to be selected. The possible clock source are the system clock (**UART_CLOCK_SYSTEM**) or the precision internal oscillator (**UART_CLOCK_PIOSC**).

Changing the baud clock source changes the baud rate generated by the UART. Therefore, the baud rate should be reconfigured after any change to the baud clock source.

Note:

The ability to specify the UART baud clock source varies with the Tiva part in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:

None.

30.2.2.14 UARTConfigGetExpClk

Gets the current configuration of a UART.

Prototype:

```
void
UARTConfigGetExpClk(uint32_t ui32Base,
                    uint32_t ui32UARTClk,
                    uint32_t *pui32Baud,
                    uint32_t *pui32Config)
```

Parameters:

ui32Base is the base address of the UART port.

ui32UARTClk is the rate of the clock supplied to the UART module.

pui32Baud is a pointer to storage for the baud rate.

pui32Config is a pointer to storage for the data format.

Description:

This function determines the baud rate and data format for the UART, given an explicitly provided peripheral clock (hence the ExpClk suffix). The returned baud rate is the actual baud rate; it may not be the exact baud rate requested or an “official” baud rate. The data format returned in *pui32Config* is enumerated the same as the *ui32Config* parameter of UARTConfigSetExpClk().

The peripheral clock is the same as the processor clock. The frequency of the system clock is the value returned by SysCtlClockGet() for TM4C123x devices or the value returned by SysCtlClockFreqSet() for TM4C129x devices, or it can be explicitly hard coded if it is constant and known (to save the code/execution overhead of a call to SysCtlClockGet() or fetch of the variable call holding the return value of SysCtlClockFreqSet()).

For Tiva parts that have the ability to specify the UART baud clock source (via UARTClockSourceSet()), the peripheral clock can be changed to PIOSC. In this case, the peripheral clock should be specified as 16, 000, 000 (the nominal rate of PIOSC).

Returns:

None.

30.2.2.15 UARTConfigSetExpClk

Sets the configuration of a UART.

Prototype:

```
void
UARTConfigSetExpClk(uint32_t ui32Base,
                    uint32_t ui32UARTClk,
                    uint32_t ui32Baud,
                    uint32_t ui32Config)
```

Parameters:

ui32Base is the base address of the UART port.

ui32UARTClk is the rate of the clock supplied to the UART module.

ui32Baud is the desired baud rate.

ui32Config is the data format for the port (number of data bits, number of stop bits, and parity).

Description:

This function configures the UART for operation in the specified data format. The baud rate is provided in the *ui32Baud* parameter and the data format in the *ui32Config* parameter.

The *ui32Config* parameter is the logical OR of three values: the number of data bits, the number of stop bits, and the parity. **UART_CONFIG_WLEN_8**, **UART_CONFIG_WLEN_7**, **UART_CONFIG_WLEN_6**, and **UART_CONFIG_WLEN_5** select from eight to five data bits per byte (respectively). **UART_CONFIG_STOP_ONE** and **UART_CONFIG_STOP_TWO** select one or two stop bits (respectively). **UART_CONFIG_PAR_NONE**, **UART_CONFIG_PAR_EVEN**, **UART_CONFIG_PAR_ODD**, **UART_CONFIG_PAR_ONE**, and **UART_CONFIG_PAR_ZERO** select the parity mode (no parity bit, even parity bit, odd parity bit, parity bit always one, and parity bit always zero, respectively).

The peripheral clock is the same as the processor clock. The frequency of the system clock is the value returned by SysCtlClockGet() for TM4C123x devices or the value returned by SysCtlClockFreqSet() for TM4C129x devices, or it can be explicitly hard coded if it is constant and known (to save the code/execution overhead of a call to SysCtlClockGet() or fetch of the variable call holding the return value of SysCtlClockFreqSet()).

The function disables the UART by calling UARTDisable() before changing the the parameters and enables the UART by calling UARTEnable().

For Tiva parts that have the ability to specify the UART baud clock source (via UARTClockSourceSet()), the peripheral clock can be changed to PIOSC. In this case, the peripheral clock should be specified as 16, 000, 000 (the nominal rate of PIOSC).

Returns:

None.

30.2.2.16 UARTDisable

Disables transmitting and receiving.

Prototype:

```
void
UARTDisable(uint32_t ui32Base)
```

Parameters:
ui32Base is the base address of the UART port.

Description:
This function disables the UART, waits for the end of transmission of the current character, and flushes the transmit FIFO.

Returns:
None.

30.2.2.17 UARTDisableSIR

Disables SIR (IrDA) mode on the specified UART.

Prototype:
```
void
UARTDisableSIR(uint32_t ui32Base)
```

Parameters:
ui32Base is the base address of the UART port.

Description:
This function disables SIR(IrDA) mode on the UART. This function only has an effect if the UART has not been enabled by a call to UARTEnable(). The call UARTEnableSIR() must be made before a call to UARTConfigSetExpClk() because the UARTConfigSetExpClk() function calls the UARTEnable() function. Another option is to call UARTDisable() followed by UARTEnableSIR() and then enable the UART by calling UARTEnable().

Note:
The availability of SIR (IrDA) operation varies with the Tiva part in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:
None.

30.2.2.18 UARTDMADisable

Disable UART uDMA operation.

Prototype:
```
void
UARTDMADisable(uint32_t ui32Base,
               uint32_t ui32DMAFlags)
```

Parameters:
ui32Base is the base address of the UART port.
ui32DMAFlags is a bit mask of the uDMA features to disable.

Description:
This function is used to disable UART uDMA features that were enabled by UARTDMAEnable(). The specified UART uDMA features are disabled. The *ui32DMAFlags* parameter is the logical OR of any of the following values:

- **UART_DMA_RX** - disable uDMA for receive
- **UART_DMA_TX** - disable uDMA for transmit
- **UART_DMA_ERR_RXSTOP** - do not disable uDMA receive on UART error

Returns:
None.

30.2.2.19 UARTDMAEnable

Enable UART uDMA operation.

Prototype:
```
void
UARTDMAEnable(uint32_t ui32Base,
              uint32_t ui32DMAFlags)
```

Parameters:
ui32Base is the base address of the UART port.
ui32DMAFlags is a bit mask of the uDMA features to enable.

Description:
The specified UART uDMA features are enabled. The UART can be configured to use uDMA for transmit or receive and to disable receive if an error occurs. The *ui32DMAFlags* parameter is the logical OR of any of the following values:

- **UART_DMA_RX** - enable uDMA for receive
- **UART_DMA_TX** - enable uDMA for transmit
- **UART_DMA_ERR_RXSTOP** - disable uDMA receive on UART error

Note:
The uDMA controller must also be set up before DMA can be used with the UART.

Returns:
None.

30.2.2.20 UARTEnable

Enables transmitting and receiving.

Prototype:
```
void
UARTEnable(uint32_t ui32Base)
```

Parameters:
ui32Base is the base address of the UART port.

Description:
This function enables the UART and its transmit and receive FIFOs.

Returns:
None.

30.2.2.21 UARTEnableSIR

Enables SIR (IrDA) mode on the specified UART.

Prototype:

```
void
UARTEnableSIR(uint32_t ui32Base,
              bool bLowPower)
```

Parameters:

ui32Base is the base address of the UART port.

bLowPower indicates if SIR Low Power Mode is to be used.

Description:

This function enables SIR (IrDA) mode on the UART. If the *bLowPower* flag is set, then SIR low power mode will be selected as well. This function only has an effect if the UART has not been enabled by a call to UARTEnable(). The call UARTEnableSIR() must be made before a call to UARTConfigSetExpClk() because the UARTConfigSetExpClk() function calls the UARTEnable() function. Another option is to call UARTDisable() followed by UARTEnableSIR() and then enable the UART by calling UARTEnable().

Note:

The availability of SIR (IrDA) operation varies with the Tiva part in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:

None.

30.2.2.22 UARTFIFODisable

Disables the transmit and receive FIFOs.

Prototype:

```
void
UARTFIFODisable(uint32_t ui32Base)
```

Parameters:

ui32Base is the base address of the UART port.

Description:

This function disables the transmit and receive FIFOs in the UART.

Returns:

None.

30.2.2.23 UARTFIFOEnable

Enables the transmit and receive FIFOs.

Prototype:

```
void
UARTFIFOEnable(uint32_t ui32Base)
```

Parameters:
ui32Base is the base address of the UART port.

Description:
This functions enables the transmit and receive FIFOs in the UART.

Returns:
None.

30.2.2.24 UARTFIFOLevelGet

Gets the FIFO level at which interrupts are generated.

Prototype:
```
void
UARTFIFOLevelGet(uint32_t ui32Base,
                 uint32_t *pui32TxLevel,
                 uint32_t *pui32RxLevel)
```

Parameters:
ui32Base is the base address of the UART port.
pui32TxLevel is a pointer to storage for the transmit FIFO level, returned as one of **UART_FIFO_TX1_8**, **UART_FIFO_TX2_8**, **UART_FIFO_TX4_8**, **UART_FIFO_TX6_8**, or **UART_FIFO_TX7_8**.
pui32RxLevel is a pointer to storage for the receive FIFO level, returned as one of **UART_FIFO_RX1_8**, **UART_FIFO_RX2_8**, **UART_FIFO_RX4_8**, **UART_FIFO_RX6_8**, or **UART_FIFO_RX7_8**.

Description:
This function gets the FIFO level at which transmit and receive interrupts are generated.

Returns:
None.

30.2.2.25 UARTFIFOLevelSet

Sets the FIFO level at which interrupts are generated.

Prototype:
```
void
UARTFIFOLevelSet(uint32_t ui32Base,
                 uint32_t ui32TxLevel,
                 uint32_t ui32RxLevel)
```

Parameters:
ui32Base is the base address of the UART port.
ui32TxLevel is the transmit FIFO interrupt level, specified as one of **UART_FIFO_TX1_8**, **UART_FIFO_TX2_8**, **UART_FIFO_TX4_8**, **UART_FIFO_TX6_8**, or **UART_FIFO_TX7_8**.
ui32RxLevel is the receive FIFO interrupt level, specified as one of **UART_FIFO_RX1_8**, **UART_FIFO_RX2_8**, **UART_FIFO_RX4_8**, **UART_FIFO_RX6_8**, or **UART_FIFO_RX7_8**.

Description:
This function configures the FIFO level at which transmit and receive interrupts are generated.

Returns:
None.

30.2.2.26 UARTFlowControlGet

Returns the UART hardware flow control mode currently in use.

Prototype:
```
uint32_t
UARTFlowControlGet(uint32_t ui32Base)
```

Parameters:
ui32Base is the base address of the UART port.

Description:
This function returns the current hardware flow control mode.

Note:
The availability of hardware flow control varies with the Tiva part and UART in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:
Returns the current flow control mode in use. This value is a logical OR combination of values **UART_FLOWCONTROL_TX** if transmit (CTS) flow control is enabled and **UART_FLOWCONTROL_RX** if receive (RTS) flow control is in use. If hardware flow control is disabled, **UART_FLOWCONTROL_NONE** is returned.

30.2.2.27 UARTFlowControlSet

Sets the UART hardware flow control mode to be used.

Prototype:
```
void
UARTFlowControlSet(uint32_t ui32Base,
                   uint32_t ui32Mode)
```

Parameters:
ui32Base is the base address of the UART port.

ui32Mode indicates the flow control modes to be used. This parameter is a logical OR combination of values **UART_FLOWCONTROL_TX** and **UART_FLOWCONTROL_RX** to enable hardware transmit (CTS) and receive (RTS) flow control or **UART_FLOWCONTROL_NONE** to disable hardware flow control.

Description:
This function configures the required hardware flow control modes. If *ui32Mode* contains flag **UART_FLOWCONTROL_TX**, data is only transmitted if the incoming CTS signal is asserted. If *ui32Mode* contains flag **UART_FLOWCONTROL_RX**, the RTS output is controlled by the hardware and is asserted only when there is space available in the receive FIFO. If no hardware flow control is required, **UART_FLOWCONTROL_NONE** should be passed.

Note:
The availability of hardware flow control varies with the Tiva part and UART in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:
None.

30.2.2.28 UARTIntClear

Clears UART interrupt sources.

Prototype:
```
void
UARTIntClear(uint32_t ui32Base,
             uint32_t ui32IntFlags)
```

Parameters:
ui32Base is the base address of the UART port.
ui32IntFlags is a bit mask of the interrupt sources to be cleared.

Description:
The specified UART interrupt sources are cleared, so that they no longer assert. This function must be called in the interrupt handler to keep the interrupt from being triggered again immediately upon exit.

The *ui32IntFlags* parameter has the same definition as the *ui32IntFlags* parameter to UARTIntEnable().

Note:
Because there is a write buffer in the Cortex-M processor, it may take several clock cycles before the interrupt source is actually cleared. Therefore, it is recommended that the interrupt source be cleared early in the interrupt handler (as opposed to the very last action) to avoid returning from the interrupt handler before the interrupt source is actually cleared. Failure to do so may result in the interrupt handler being immediately reentered (because the interrupt controller still sees the interrupt source asserted).

Returns:
None.

30.2.2.29 UARTIntDisable

Disables individual UART interrupt sources.

Prototype:
```
void
UARTIntDisable(uint32_t ui32Base,
               uint32_t ui32IntFlags)
```

Parameters:
ui32Base is the base address of the UART port.
ui32IntFlags is the bit mask of the interrupt sources to be disabled.

Description:

This function disables the indicated UART interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

The *ui32IntFlags* parameter has the same definition as the *ui32IntFlags* parameter to UARTIntEnable().

Returns:

None.

30.2.2.30 UARTIntEnable

Enables individual UART interrupt sources.

Prototype:

```
void
UARTIntEnable(uint32_t ui32Base,
              uint32_t ui32IntFlags)
```

Parameters:

ui32Base is the base address of the UART port.

ui32IntFlags is the bit mask of the interrupt sources to be enabled.

Description:

This function enables the indicated UART interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

The *ui32IntFlags* parameter is the logical OR of any of the following:

- **UART_INT_9BIT** - 9-bit Address Match interrupt
- **UART_INT_OE** - Overrun Error interrupt
- **UART_INT_BE** - Break Error interrupt
- **UART_INT_PE** - Parity Error interrupt
- **UART_INT_FE** - Framing Error interrupt
- **UART_INT_RT** - Receive Timeout interrupt
- **UART_INT_TX** - Transmit interrupt
- **UART_INT_RX** - Receive interrupt
- **UART_INT_DSR** - DSR interrupt
- **UART_INT_DCD** - DCD interrupt
- **UART_INT_CTS** - CTS interrupt
- **UART_INT_RI** - RI interrupt

Returns:

None.

30.2.2.31 UARTIntRegister

Registers an interrupt handler for a UART interrupt.

Prototype:

```
void
UARTIntRegister(uint32_t ui32Base,
                void (*pfnHandler)(void))
```

Parameters:

ui32Base is the base address of the UART port.

pfnHandler is a pointer to the function to be called when the UART interrupt occurs.

Description:

This function does the actual registering of the interrupt handler. This function enables the global interrupt in the interrupt controller; specific UART interrupts must be enabled via UARTIntEnable(). It is the interrupt handler's responsibility to clear the interrupt source.

See also:

IntRegister() for important information about registering interrupt handlers.

Returns:

None.

30.2.2.32 UARTIntStatus

Gets the current interrupt status.

Prototype:

```
uint32_t
UARTIntStatus(uint32_t ui32Base,
              bool bMasked)
```

Parameters:

ui32Base is the base address of the UART port.

bMasked is **false** if the raw interrupt status is required and **true** if the masked interrupt status is required.

Description:

This function returns the interrupt status for the specified UART. Either the raw interrupt status or the status of interrupts that are allowed to reflect to the processor can be returned.

Returns:

Returns the current interrupt status, enumerated as a bit field of values described in UARTIntEnable().

30.2.2.33 UARTIntUnregister

Unregisters an interrupt handler for a UART interrupt.

Prototype:

```
void
UARTIntUnregister(uint32_t ui32Base)
```

Parameters:
ui32Base is the base address of the UART port.

Description:
This function does the actual unregistering of the interrupt handler. It clears the handler to be called when a UART interrupt occurs. This function also masks off the interrupt in the interrupt controller so that the interrupt handler no longer is called.

See also:
IntRegister() for important information about registering interrupt handlers.

Returns:
None.

30.2.2.34 UARTLoopbackEnable

Enables internal loopback mode for a UART port

Prototype:
```
void
UARTLoopbackEnable(uint32_t ui32Base)
```

Parameters:
ui32Base is the base address of the UART port.

Description:
This function configures a UART port in internal loopback mode to help with diagnostics and debug. In this mode, the transmit and receive terminals of the same UART port are internally connected. Hence, the data transmitted on the UnTx output is received on the UxRx input, without having to go through I/O's. UARTCharPut(), UARTCharGet() functions can be used along with this function.

Returns:
None.

30.2.2.35 UARTModemControlClear

Clears the states of the DTR and/or RTS modem control signals.

Prototype:
```
void
UARTModemControlClear(uint32_t ui32Base,
                      uint32_t ui32Control)
```

Parameters:
ui32Base is the base address of the UART port.
ui32Control is a bit-mapped flag indicating which modem control bits should be set.

Description:
This function clears the states of the DTR or RTS modem handshake outputs from the UART.

The *ui32Control* parameter is the logical OR of any of the following:

- **UART_OUTPUT_DTR** - The modem control DTR signal
- **UART_OUTPUT_RTS** - The modem control RTS signal

Note:
The availability of hardware modem handshake signals varies with the Tiva part and UART in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:
None.

30.2.2.36 UARTModemControlGet

Gets the states of the DTR and RTS modem control signals.

Prototype:
```
uint32_t
UARTModemControlGet(uint32_t ui32Base)
```

Parameters:
ui32Base is the base address of the UART port.

Description:
This function returns the current states of each of the two UART modem control signals, DTR and RTS.

Note:
The availability of hardware modem handshake signals varies with the Tiva part and UART in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:
Returns the states of the handshake output signals. This value is a logical OR combination of values **UART_OUTPUT_RTS** and **UART_OUTPUT_DTR** where the presence of each flag indicates that the associated signal is asserted.

30.2.2.37 UARTModemControlSet

Sets the states of the DTR and/or RTS modem control signals.

Prototype:
```
void
UARTModemControlSet(uint32_t ui32Base,
                    uint32_t ui32Control)
```

Parameters:
ui32Base is the base address of the UART port.
ui32Control is a bit-mapped flag indicating which modem control bits should be set.

Description:
This function configures the states of the DTR or RTS modem handshake outputs from the UART.

The *ui32Control* parameter is the logical OR of any of the following:

- **UART_OUTPUT_DTR** - The modem control DTR signal
- **UART_OUTPUT_RTS** - The modem control RTS signal

Note:
The availability of hardware modem handshake signals varies with the Tiva part and UART in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:
None.

30.2.2.38 UARTModemStatusGet

Gets the states of the RI, DCD, DSR and CTS modem status signals.

Prototype:
```
uint32_t
UARTModemStatusGet(uint32_t ui32Base)
```

Parameters:
ui32Base is the base address of the UART port.

Description:
This function returns the current states of each of the four UART modem status signals, RI, DCD, DSR and CTS.

Note:
The availability of hardware modem handshake signals varies with the Tiva part and UART in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:
Returns the states of the handshake output signals. This value is a logical OR combination of values **UART_INPUT_RI**, **UART_INPUT_DCD**, **UART_INPUT_CTS** and **UART_INPUT_DSR** where the presence of each flag indicates that the associated signal is asserted.

30.2.2.39 UARTParityModeGet

Gets the type of parity currently being used.

Prototype:
```
uint32_t
UARTParityModeGet(uint32_t ui32Base)
```

Parameters:
ui32Base is the base address of the UART port.

Description:
This function gets the type of parity used for transmitting data and expected when receiving data.

Returns:
Returns the current parity settings, specified as one of **UART_CONFIG_PAR_NONE**, **UART_CONFIG_PAR_EVEN**, **UART_CONFIG_PAR_ODD**, **UART_CONFIG_PAR_ONE**, or **UART_CONFIG_PAR_ZERO**.

30.2.2.40 UARTParityModeSet

Sets the type of parity.

Prototype:
```
void
UARTParityModeSet(uint32_t ui32Base,
                  uint32_t ui32Parity)
```

Parameters:
ui32Base is the base address of the UART port.
ui32Parity specifies the type of parity to use.

Description:
This function configures the type of parity to use for transmitting and expect when receiving. The *ui32Parity* parameter must be one of **UART_CONFIG_PAR_NONE**, **UART_CONFIG_PAR_EVEN**, **UART_CONFIG_PAR_ODD**, **UART_CONFIG_PAR_ONE**, or **UART_CONFIG_PAR_ZERO**. The last two parameters allow direct control of the parity bit; it is always either one or zero based on the mode.

Returns:
None.

30.2.2.41 UARTRxErrorClear

Clears all reported receiver errors.

Prototype:
```
void
UARTRxErrorClear(uint32_t ui32Base)
```

Parameters:
ui32Base is the base address of the UART port.

Description:
This function is used to clear all receiver error conditions reported via UARTRxErrorGet(). If using the overrun, framing error, parity error or break interrupts, this function must be called after clearing the interrupt to ensure that later errors of the same type trigger another interrupt.

Returns:
None.

30.2.2.42 UARTRxErrorGet

Gets current receiver errors.

Prototype:

```
uint32_t
UARTRxErrorGet(uint32_t ui32Base)
```

Parameters:

ui32Base is the base address of the UART port.

Description:

This function returns the current state of each of the 4 receiver error sources. The returned errors are equivalent to the four error bits returned via the previous call to UARTCharGet() or UARTCharGetNonBlocking() with the exception that the overrun error is set immediately when the overrun occurs rather than when a character is next read.

Returns:

Returns a logical OR combination of the receiver error flags, **UART_RXERROR_FRAMING**, **UART_RXERROR_PARITY**, **UART_RXERROR_BREAK** and **UART_RXERROR_OVERRUN**.

30.2.2.43 UARTSmartCardDisable

Disables ISO7816 smart card mode on the specified UART.

Prototype:

```
void
UARTSmartCardDisable(uint32_t ui32Base)
```

Parameters:

ui32Base is the base address of the UART port.

Description:

This function clears the SMART (ISO7816 smart card) bit in the UART control register.

Note:

The availability of ISO7816 smart card mode varies with the Tiva part and UART in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:

None.

30.2.2.44 UARTSmartCardEnable

Enables ISO7816 smart card mode on the specified UART.

Prototype:

```
void
UARTSmartCardEnable(uint32_t ui32Base)
```

Parameters:

ui32Base is the base address of the UART port.

Description:
This function enables the SMART control bit for the ISO7816 smart card mode on the UART. This call also sets 8-bit word length and even parity as required by ISO7816.

Note:
The availability of ISO7816 smart card mode varies with the Tiva part and UART in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:
None.

30.2.2.45 UARTSpaceAvail

Determines if there is any space in the transmit FIFO.

Prototype:
```
bool
UARTSpaceAvail(uint32_t ui32Base)
```

Parameters:
ui32Base is the base address of the UART port.

Description:
This function returns a flag indicating whether or not there is space available in the transmit FIFO.

Returns:
Returns **true** if there is space available in the transmit FIFO or **false** if there is no space available in the transmit FIFO.

30.2.2.46 UARTTxIntModeGet

Returns the current operating mode for the UART transmit interrupt.

Prototype:
```
uint32_t
UARTTxIntModeGet(uint32_t ui32Base)
```

Parameters:
ui32Base is the base address of the UART port.

Description:
This function returns the current operating mode for the UART transmit interrupt. The return value is **UART_TXINT_MODE_EOT** if the transmit interrupt is currently configured to be asserted once the transmitter is completely idle - the transmit FIFO is empty and all bits, including any stop bits, have cleared the transmitter. The return value is **UART_TXINT_MODE_FIFO** if the interrupt is configured to be asserted based on the level of the transmit FIFO.

Note:
The availability of end-of-transmission mode varies with the Tiva part in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:
Returns **UART_TXINT_MODE_FIFO** or **UART_TXINT_MODE_EOT**.

30.2.2.47 UARTTxIntModeSet

Sets the operating mode for the UART transmit interrupt.

Prototype:
```
void
UARTTxIntModeSet(uint32_t ui32Base,
                 uint32_t ui32Mode)
```

Parameters:
ui32Base is the base address of the UART port.
ui32Mode is the operating mode for the transmit interrupt. It may be **UART_TXINT_MODE_EOT** to trigger interrupts when the transmitter is idle or **UART_TXINT_MODE_FIFO** to trigger based on the current transmit FIFO level.

Description:
This function allows the mode of the UART transmit interrupt to be set. By default, the transmit interrupt is asserted when the FIFO level falls past a threshold set via a call to UARTFIFOLevelSet(). Alternatively, if this function is called with *ui32Mode* set to **UART_TXINT_MODE_EOT**, the transmit interrupt is asserted once the transmitter is completely idle - the transmit FIFO is empty and all bits, including any stop bits, have cleared the transmitter.

Note:
The availability of end-of-transmission mode varies with the Tiva part in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns:
None.

30.3 Programming Example

The following example shows how to use the UART API to initialize the UART, transmit characters, and receive characters on a TM4C123x device.

```
//
    // Enable the UART0 module.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

    //
    // Wait for the UART0 module to be ready.
//
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_UART0))
    {
    }

// Initialize the UART.  Set the baud rate, number of data bits, turn off
// parity, number of stop bits, and stick mode. The UART is enabled by the
```

```
    // function call.
//
UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 38400,
                    (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                     UART_CONFIG_PAR_NONE));

//
// Check for characters.  Spin here until a character is placed
// into the receive FIFO.
//
while(!UARTCharsAvail(UART0_BASE))
{
}

//
// Get the character(s) in the receive FIFO.
//
while(UARTCharGetNonBlocking(UART0_BASE))
{
}

//
// Put a character in the output buffer.
//
UARTCharPut(UART0_BASE, 'c'));

//
// Disable the UART.
//
UARTDisable(UART0_BASE);
```