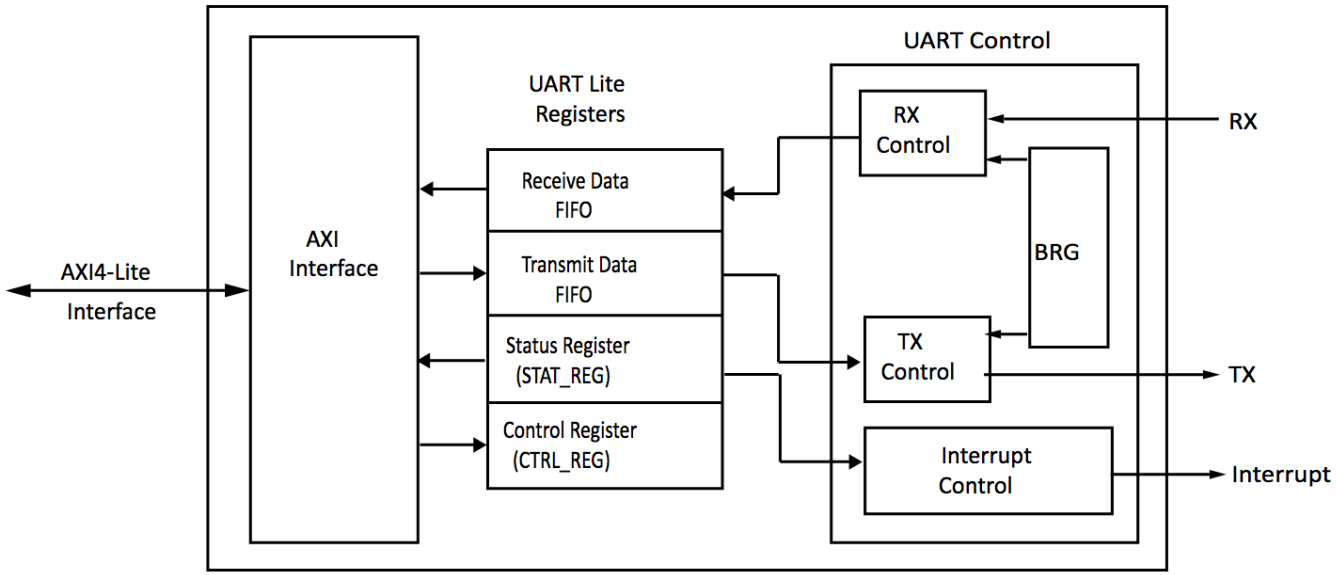
CSCE 4114
Uartlite

David Andrews
dandrews@uark.edu

Uartlite

- Soft IP version of a UART



16-character transmit and receive FIFO's
Configurable number of data bits (5-8)
Configurable Parity
Configurable Baud Rate

Uartlite

- Programmers View

Base Address + Offset (hex)	Register Name	Access Type	Default Value (hex)	Description
C_BASEADDR + 0x0	Rx FIFO	Read (1)	0x0	Receive Data FIFO
C_BASEADDR + 0x4	Tx FIFO	Write (2)	0x0	Transmit Data FIFO
C_BASEADDR + 0x8	STAT_REG	Read (1)	0x4	UART Lite Status Register
C_BASEADDR + 0xC	CTRL_REG	Write (2)	0x0	UART Lite Control Register

1. Writing of a read only register has no effect.
2. Reading of a write only register returns zero.

Uartlite

- Transmit/Receive bytes
- Tx/Rx Channels are 16 Deep FIFO's. Why ?

Uartlite: Control Register

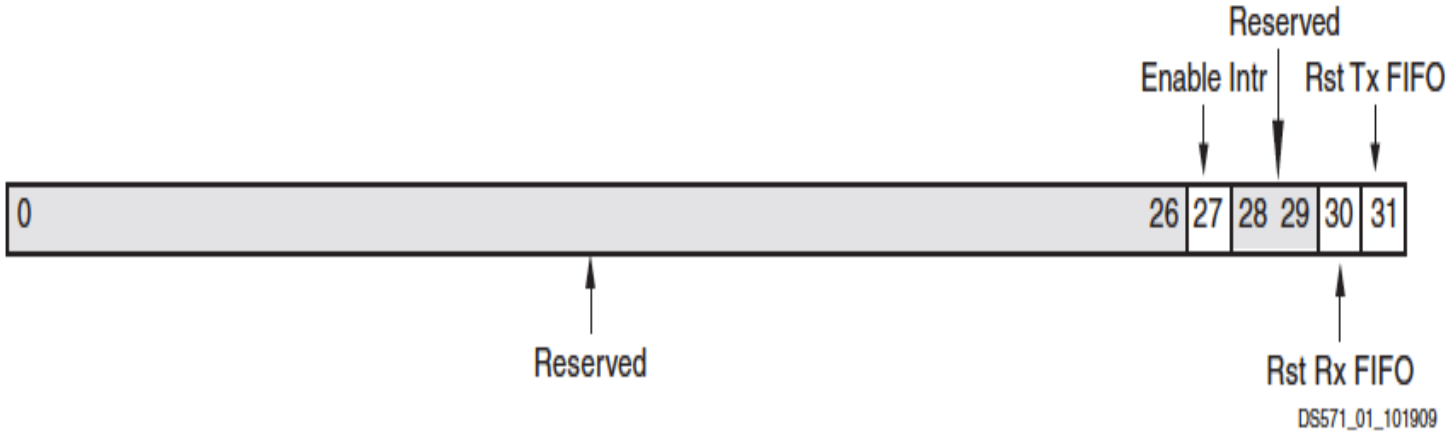


Figure 4: **UART Lite Control Register**

- Enable Intr: 1=Enabled, 0=Disabled
- Rst Tx FIFO: 1=Reset, 0=nothing
- Rst Rx FIFO: 1=Reset, 0=nothing

Uartlite: Control Register

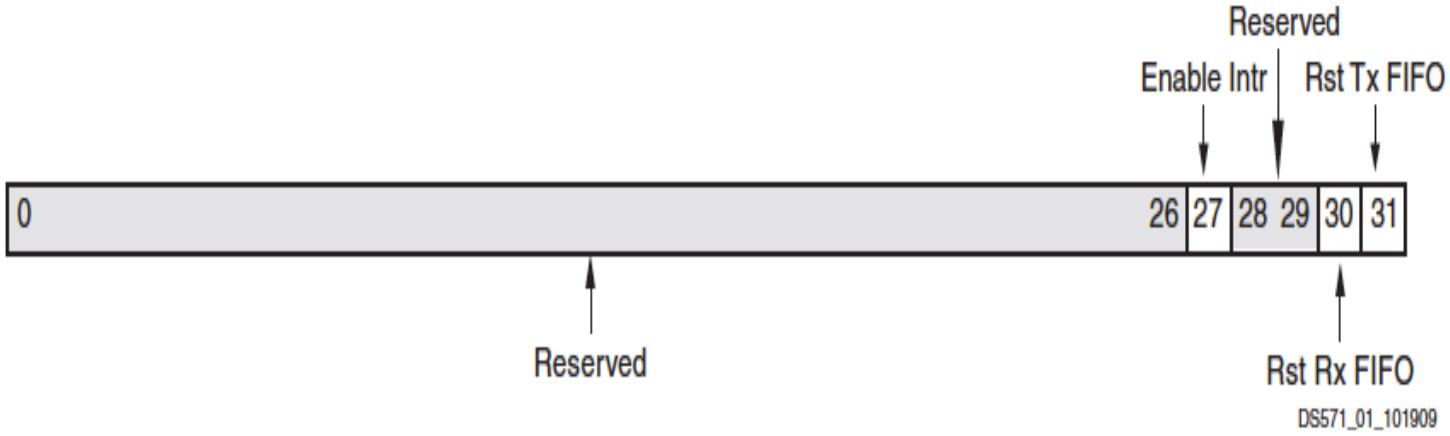


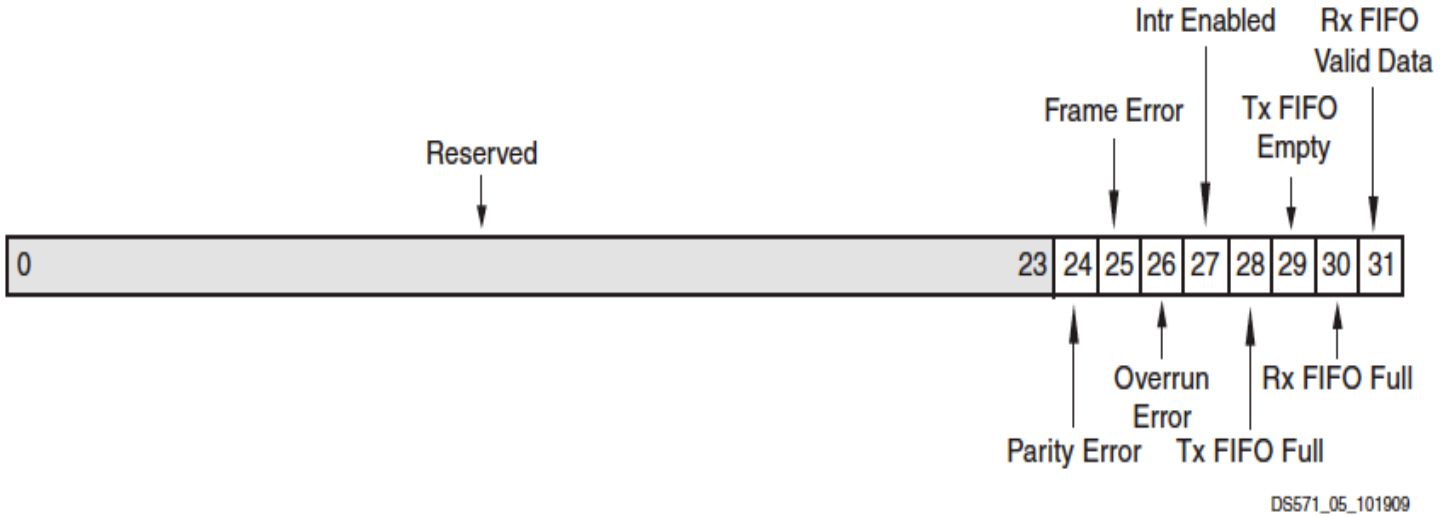
Figure 4: **UART Lite Control Register**

Int mask = ox?

*cntrl_reg &= mask;

to Disable Interrupts, reset receive FIFO

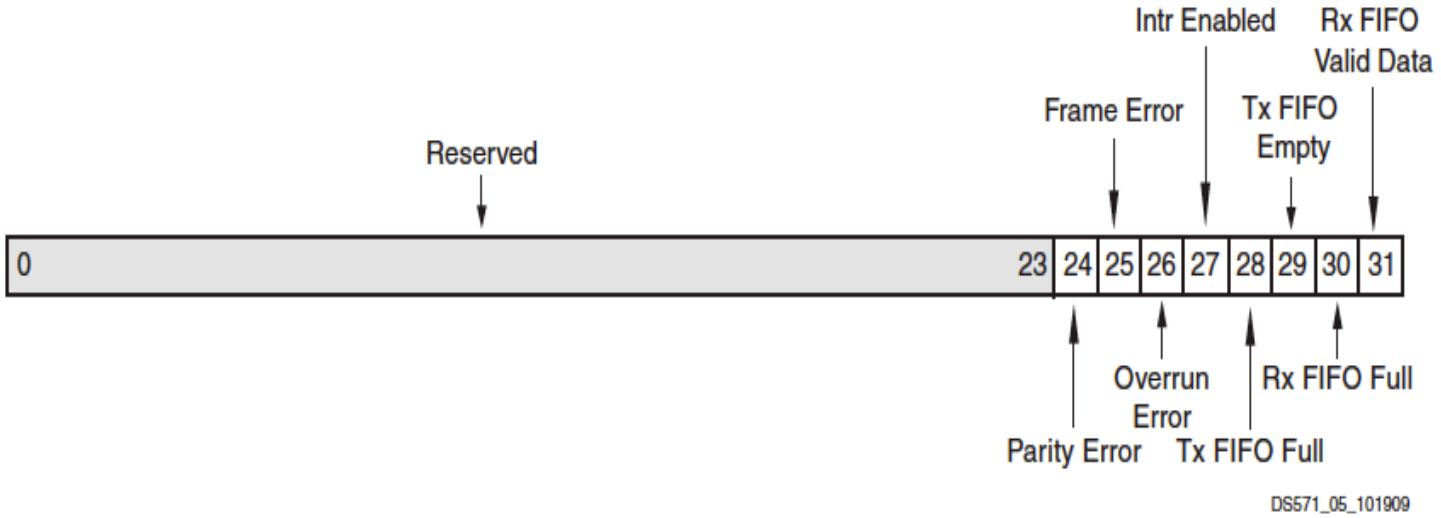
Uartlite: Status Register



Errors: Parity, Frame, Overrun

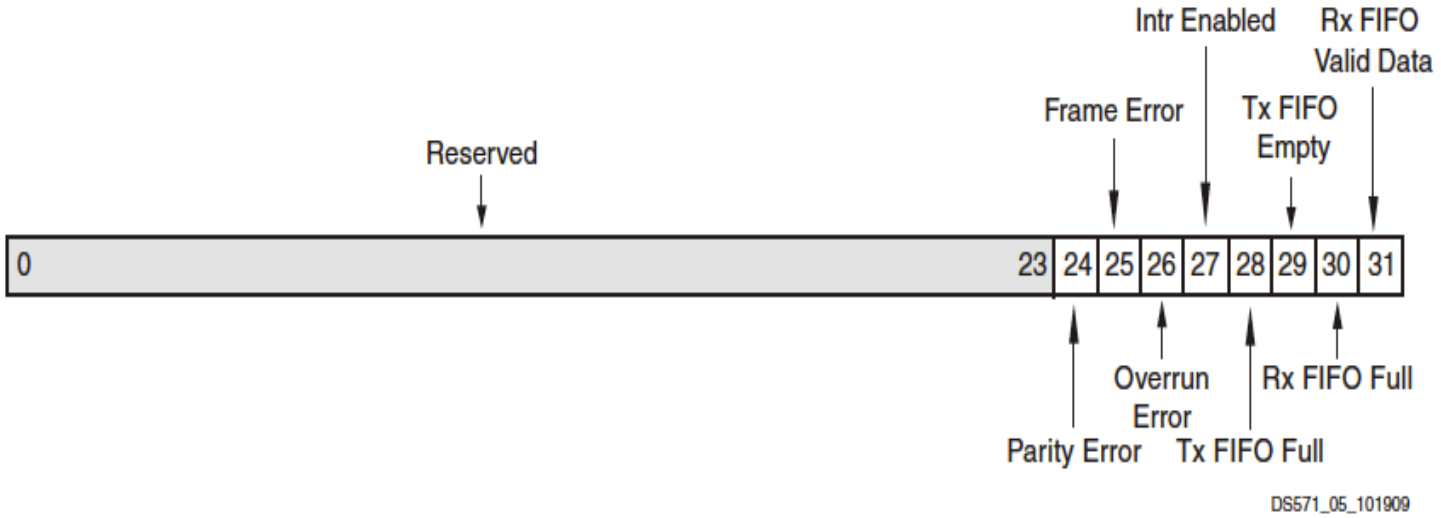
Status: intr enabled,Tx,Rx FIFO's

Uartlite: Status Register



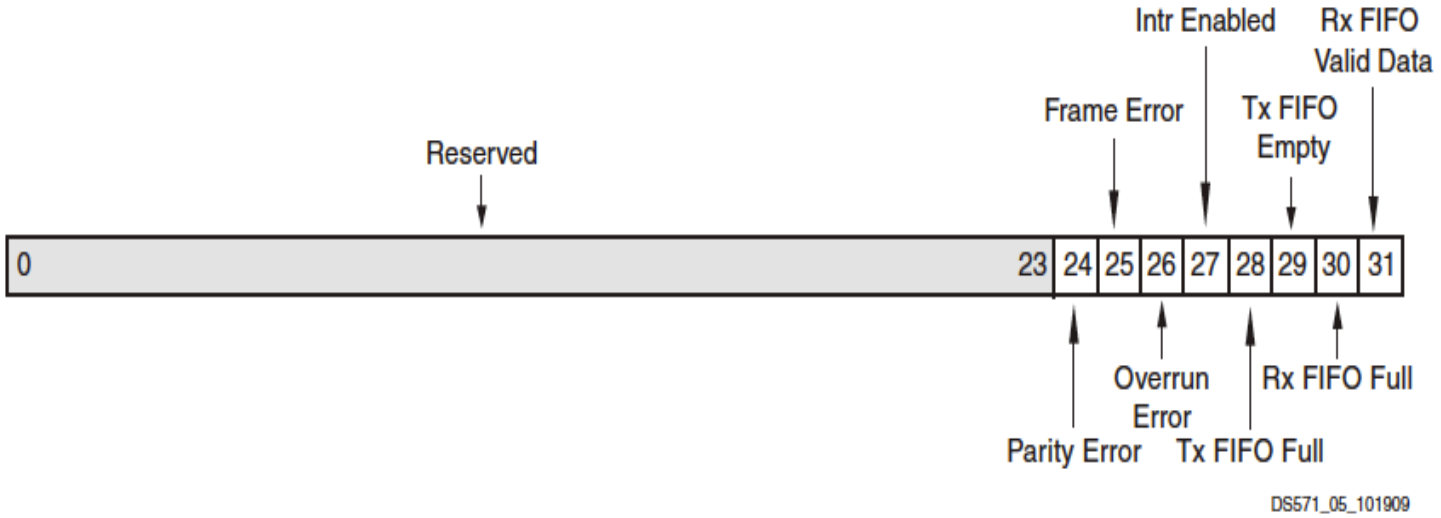
Parity Error: Does not match parity

Uartlite: Status Register



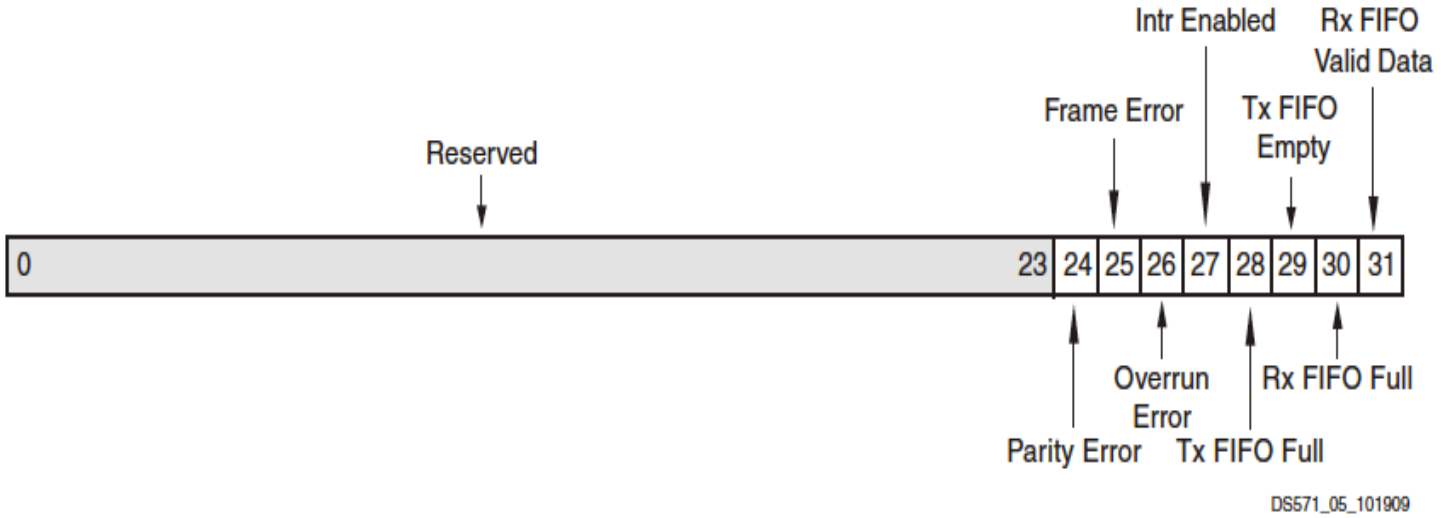
Frame Error: "0" Detected in stop bit position

Uartlite: Status Register



Overrun Error: Rx buffer full when new transmission received

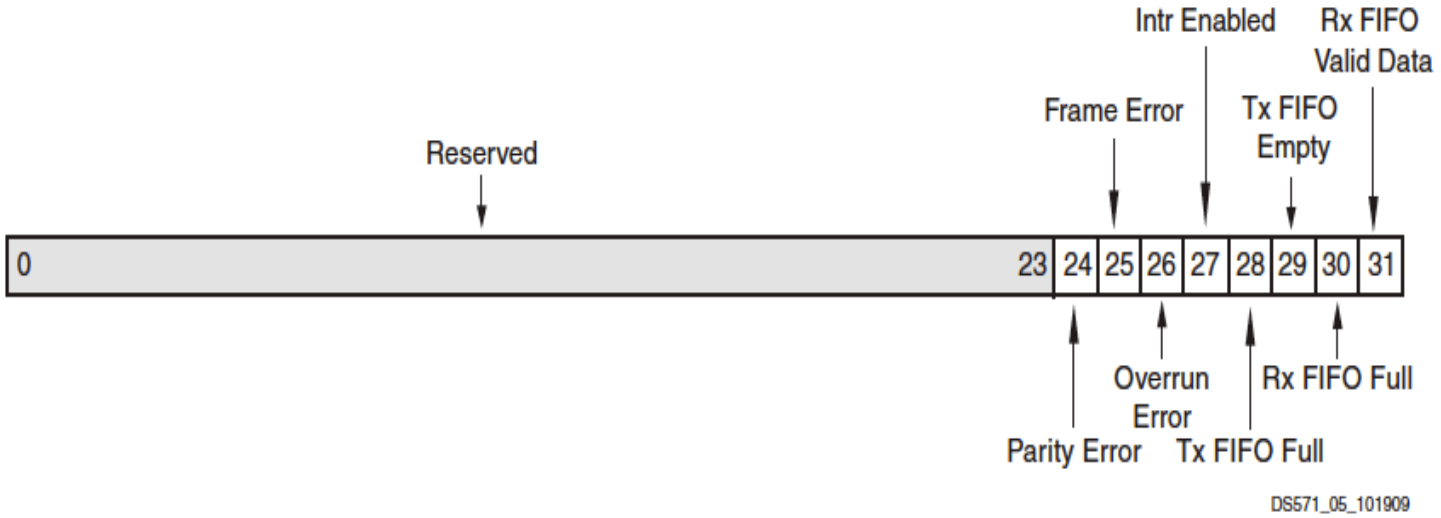
Uartlite: Status Register



Overrun Error: Rx buffer full when new transmission received

Last transmission is lost !

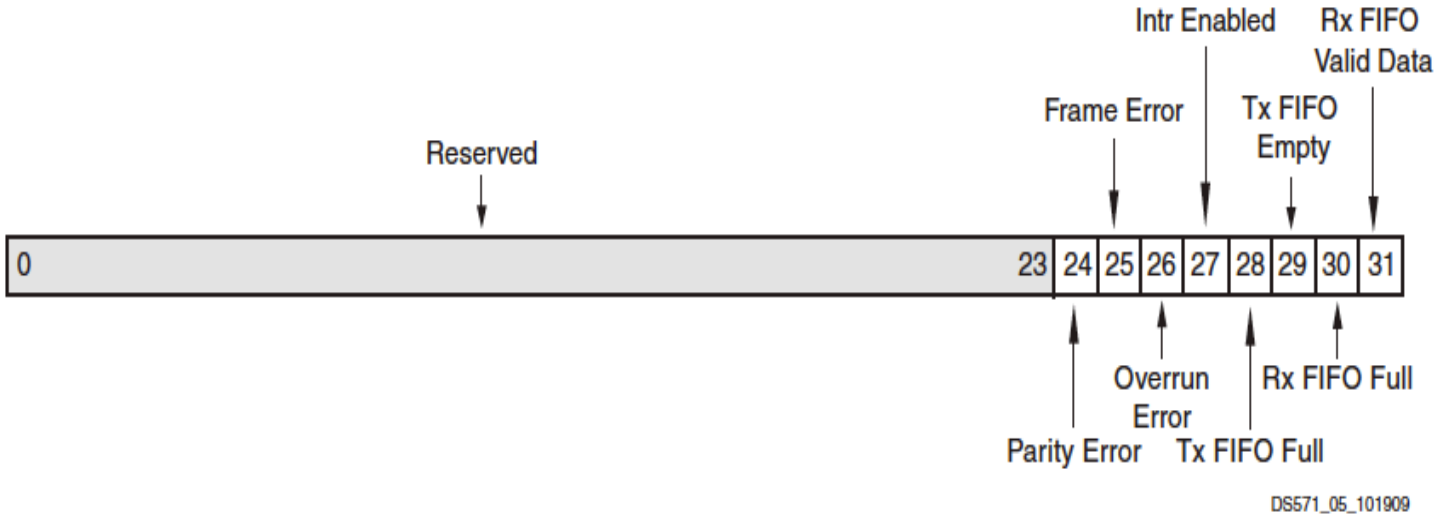
Uartlite: Status Register



Tx FIFO Full: No room left for another transmission character !

you routinely check this before transmitting

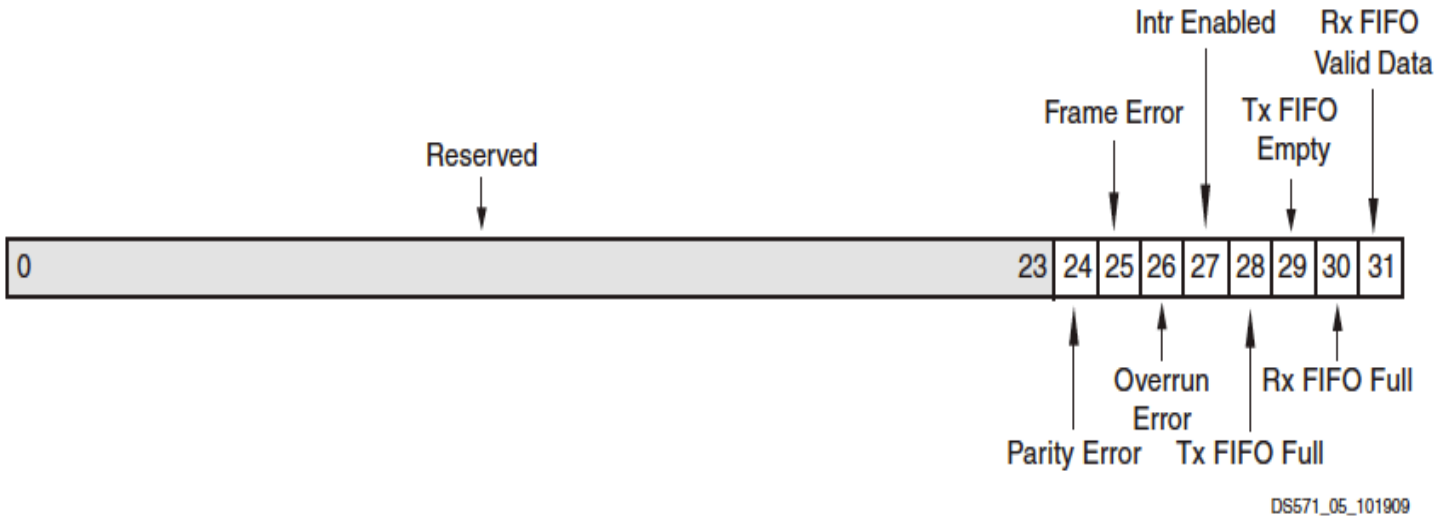
Uartlite: Status Register



Tx FIFO Empty: All Characters sent !

Tells you everything has been received. How about received correctly ?

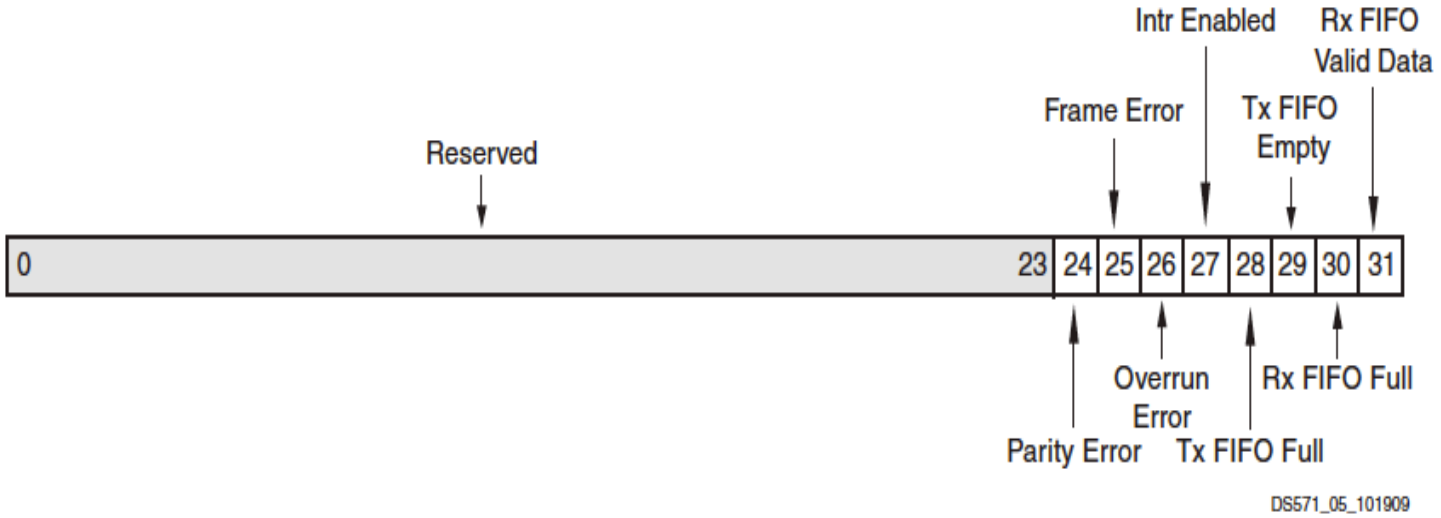
Uartlite: Status Register



Rx FIFO Full: No more room in your buffer.

Need to read data to avoid overrun errors !

Uartlite: Status Register



Rx FIFO Valid Data: Data is there and ready !

Can keep reading until Flag==0: -> you have read everything that is waiting

some addresses/bit masks

Important memory offsets (in decimal):

UART_RX_FIFO offset is 0 (Used to read Rx_FIFO values, read-only

UART_TX_FIFO offset is 4 (Used to write Tx_FIFO values, write only

UART_STATUS_REG offset is 8 (Used to check UART status, read only)

UART_CONTROL_REG offset is 12 (Used to configure UART, write-only)

some addresses/bit masks

Important memory offsets (in decimal):

UART_RX_FIFO offset is 0 Used to read Rx_FIFO values, read-only

UART_TX_FIFO offset is 4 Used to write Tx_FIFO values, write only

UART_STATUS_REG offset is 8 Used to check UART status, read only

UART_CONTROL_REG offset is 12 Used to configure UART, write-only

Important bit-masks (in decimal):

TX_FIFO_FULL 8	Used to check if the Tx_FIFO is full
TX_FIFO_EMPTY 4	Used to check if the Tx_FIFO is empty
RX_FIFO_FULL 2	Used to check if the Rx_FIFO is full
RX_FIFO_VALID 1	Used to check if the Rx_FIFO has data

How do I send information ?

1) Wait for Tx_FIFO status to be NOT FULL
2) Write character to Tx_FIFO

How do I send information ?

1) Wait for Tx_FIFO status to be NOT FULL
2) Write character to Tx_FIFO

- You write the pseudo C code……..

How do I send information ?

1) Wait for Tx_FIFO status to be NOT FULL
2) Write character to Tx_FIFO

- pseudo C code……..

```
while (UART_STATUS == TX_FIFO_FULL) { };
        TX_FIFO = my_char;
```

Assembler….

Assembler….

```
.set UART_base, 1080033280 /* 0x40600000 */
```

Assembler….

```
.set UART_base, 1080033280 /* 0x40600000 */

# Put UART's base address into r6
      addik r6, r0, UART_base
```

Assembler….

```
.set UART_base, 1080033280 /* 0x40600000 */

# Put UART's base address into r6
      addik r6, r0, UART_base


# Move character (byte) into r8 from character parameter (r5)
      addik r8, r5, 0
```

Assembler….

1) Wait for Tx_FIFO status to be NOT FULL

Assembler….

1) Wait for Tx_FIFO status to be NOT FULL

```
.set TX_FIFO_FULL,      8      /* Bit-mask for checking FIFO fullness */
```

Assembler….

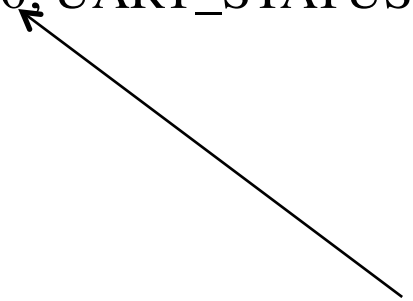
1) Wait for Tx_FIFO status to be NOT FULL

```
.set TX_FIFO_FULL,      8      /* Bit-mask for checking FIFO fullness */

# Wait until UART's TX FIFO is not full
myPrintCharLoop:
        lwi r7, r6, UART_STATUS_REG_OFFSET
```

Assembler….

1) Wait for Tx_FIFO status to be NOT FULL

```
.set TX_FIFO_FULL,      8      /* Bit-mask for checking FIFO fullness */

# Wait until UART's TX FIFO is not full
myPrintCharLoop:
        lwi r7, r6, UART_STATUS_REG_OFFSET
```



Base addr of uart already in r6

Assembler….

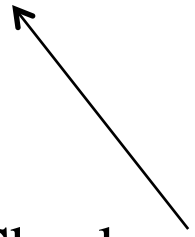
1) Wait for Tx_FIFO status to be NOT FULL

```
.set TX_FIFO_FULL,      8      /* Bit-mask for checking FIFO fullness */

# Wait until UART's TX FIFO is not full
myPrintCharLoop:
        lwi r7, r6, UART_STATUS_REG_OFFSET

# Mask out the TX_FIFO_FLAG
        andi r7, r7, TX_FIFO_FULL
```

Assembler….

1) Wait for Tx_FIFO status to be NOT FULL

```
.set TX_FIFO_FULL,      8      /* Bit-mask for checking FIFO fullness */

# Wait until UART's TX FIFO is not full
myPrintCharLoop:
        lwi r7, r6, UART_STATUS_REG_OFFSET

# Mask out the TX_FIFO_FLAG
        andi r7, r7, TX_FIFO_FULL
```



Checks only Tx_FIFO_Full Flag

Assembler….

1) Wait for Tx_FIFO status to be NOT FULL

```
.set TX_FIFO_FULL,      8      /* Bit-mask for checking FIFO fullness */

# Wait until UART's TX FIFO is not full
myPrintCharLoop:
        lwi r7, r6, UART_STATUS_REG_OFFSET

# Mask out the TX_FIFO_FLAG
        andi r7, r7, TX_FIFO_FULL

# Loop back if it is not-zero
        bnei r7 myPrintCharLoop
        nop
```

Assembler….

1) Wait for Tx_FIFO status to be NOT FULL

```
.set TX_FIFO_FULL,      8     /* Bit-mask for checking FIFO fullness */

# Wait until UART's TX FIFO is not full
myPrintCharLoop:
        lwi r7, r6, UART_STATUS_REG_OFFSET

# Mask out the TX_FIFO_FLAG
        andi r7, r7, TX_FIFO_FULL

# Loop back if it is not-zero
        bnei r7 myPrintCharLoop
        nop
```

Assembler….

2) Send Character

```
.set UART_TX_FIFO_OFFSET, 4 /* transmit FIFO, write only */
```

Assembler….

2) Send Character

```
.set UART_TX_FIFO_OFFSET, 4 /* transmit FIFO, write only */

# Send character to the FIFO
      swi r8, r6, UART_TX_FIFO_OFFSET
```

How do I receive ?

1) Wait for Rx_FIFO status to be VALID
2) Read a character from the Rx_FIFO

- Pseudo code……

How do I receive ?

1) Wait for Rx_FIFO status to be VALID
2) Read a character from the Rx_FIFO

- Pseudo code……

```
while (UART_STATUS !=RX_FIFO_VALID) { };
        my_char = RX_FIFO;
```

Assembler

1) Wait for Rx_FIFO status to be VALID

Assembler

1) Wait for Rx_FIFO status to be VALID

```
# Put UART's base address into r6
    addik r6, r0, UART_base
```

Assembler

1) Wait for Rx_FIFO status to be VALID

```
# Put UART's base address into r6
    addik r6, r0, UART_base

# Wait until UART's RX FIFO is not empty
myGetCharLoop:
            lwi r7, r6, UART_STATUS_REG_OFFSET
```

Assembler

1) Wait for Rx_FIFO status to be VALID

```
# Put UART's base address into r6
    addik r6, r0, UART_base

# Wait until UART's RX FIFO is not empty
myGetCharLoop:
            lwi r7, r6, UART_STATUS_REG_OFFSET

    # Mask out the RX_FIFO_DATA_VALID_FLAG
            andi r7, r7, RX_FIFO_VALID
```

Assembler

1) Wait for Rx_FIFO status to be VALID

```
# Put UART's base address into r6
    addik r6, r0, UART_base

# Wait until UART's RX FIFO is not empty
myGetCharLoop:
            lwi r7, r6, UART_STATUS_REG_OFFSET

    # Mask out the RX_FIFO_DATA_VALID_FLAG
            andi r7, r7, RX_FIFO_VALID

    # Loop back if it is zero
            beqi r7 myGetCharLoop
            nop
```

Assembler

2) Get Character

Assembler

2) Get Character

```
# Loop is over

# Get character from the RX-FIFO
        lwi r5, r6, UART_RX_FIFO_OFFSET
```