Section 13

ADSP-BF533 Serial Communications

BF533 Serial Communications

Three Serial Comm's Peripherals

- SPORTs (synchronous Serial PORTs)
  - High Speed (up to SCLK/2)
  - Two SPORTs (SPORT0 and SPORT1)
  - Typically used for interfacing with CODEC's and TDM data streams
- SPI (Serial Peripheral Interface)
  - Single High Speed SPI port (up to SCLK/4)
  - Typically used to interface with serial EPROMS, other CPUs, data converters, and displays
- UART (Universal Asynchronous Receiver/Transmitter)
  - Single PC-style UART port (baud rate up to SCLK/16)
  - Typically used for maintenance port, and interfacing with slow serial peripherals

ADSP-BF533 SPORTs

- Two synchronous serial ports
  - Fully independent receive and transmit channels - double buffered
  - Primary and Secondary Data RX/TX pins
  - Support up to 32-bit serial words
  - Internal or externally generated serial clocks and frame syncs
  - Programmable internal/external frame syncs
  - Built in hardware for u-law & A-law companding
  - Support for multichannel interfaces
  - I^2S signaling support
  - Generates optional interrupts
  - Separate Data and Error Interrupts
  - Operates up to ½ System bus clock rate (SCLK)

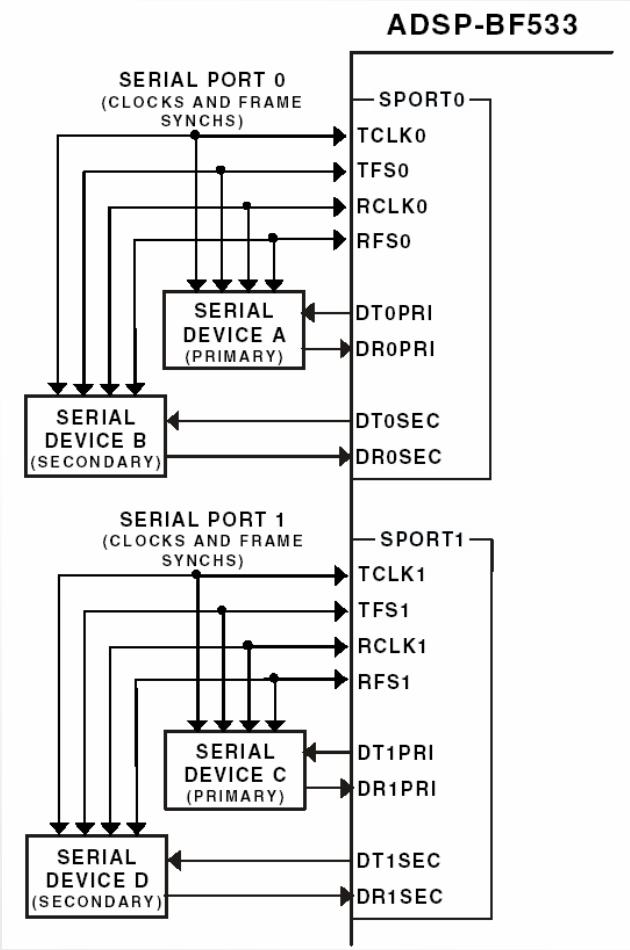
ADSP-BF533 Serial PORTs Features

- Interrupt-driven, single-word transfers to/from on-chip memory controlled by ADSP-BF533 core
- Block word transfers to/from memory controlled by DMA controller
- Several modes of operation
  - Programmable serial word length, 3 to 32-bits
  - Either MSB or LSB first
  - Early Frame Sync
  - Late Frame Sync
  - No Frame Sync
  - 128 time slot out of a 1024-channel window multi-channel capability for TDM interfaces
  - I2S capable operation

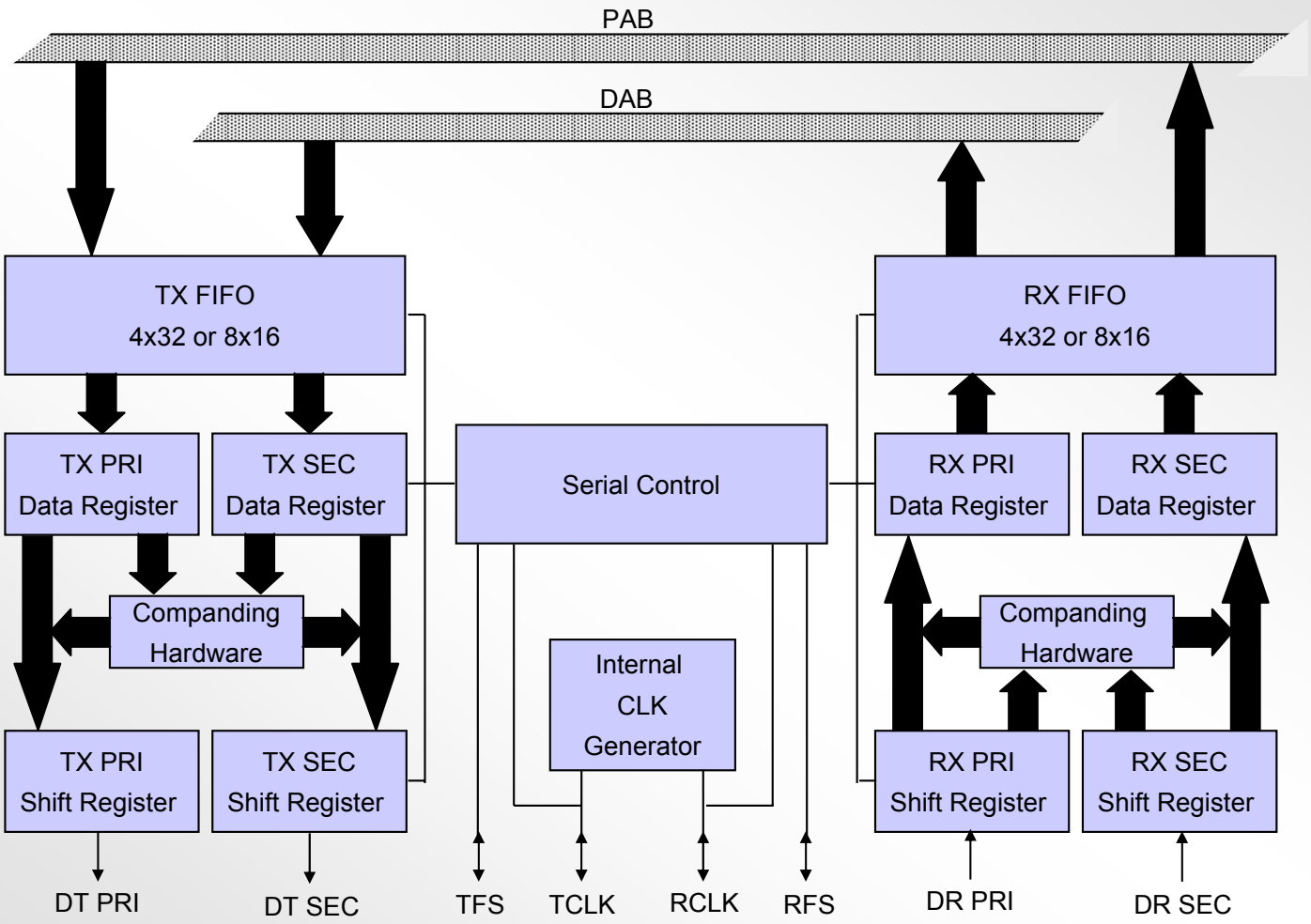
ADSP-BF533 SPORT Pins

Pin	Description
DTxPRI	Transmit Data Primary
DTxSEC	Transmit Data Secondary
TSCLKx	Transmit Clock
TFSx	Transmit Frame Sync
DRxPRI	Receive Data Primary
DRxSEC	Receive Data Secondary
RSCLKx	Receive Clock
RFSx	Receive Frame Sync

SPORT Interface



Serial Port - Block Diagram



SPORTx_TX Register and Transmit FIFO

ONLY PRIMARY ENABLED
DATA LENGTH <= 16 BITS

8 WORDS OF PRIMARY DATA IN FIFO

15	0
PRIMARY	W7
PRIMARY	W6
PRIMARY	W5
PRIMARY	W4
PRIMARY	W3
PRIMARY	W2
PRIMARY	W1
PRIMARY	W0

PRIMARY AND SECONDARY ENABLED
DATA LENGTH <= 16 BITS

4 WORDS OF PRIMARY DATA AND 4 WORDS OF SECONDARY DATA IN FIFO

15	0
SECONDARY	W3
PRIMARY	W3
SECONDARY	W2
PRIMARY	W2
SECONDARY	W1
PRIMARY	W1
SECONDARY	W0
PRIMARY	W0

ONLY PRIMARY ENABLED
DATA LENGTH > 16 BITS

4 WORDS OF PRIMARY DATA IN FIFO

15	0
PRIMARY	W3 LOW
PRIMARY	W3 HIGH
PRIMARY	W2 LOW
PRIMARY	W2 HIGH
PRIMARY	W1 LOW
PRIMARY	W1 HIGH
PRIMARY	W0 LOW
PRIMARY	W0 HIGH

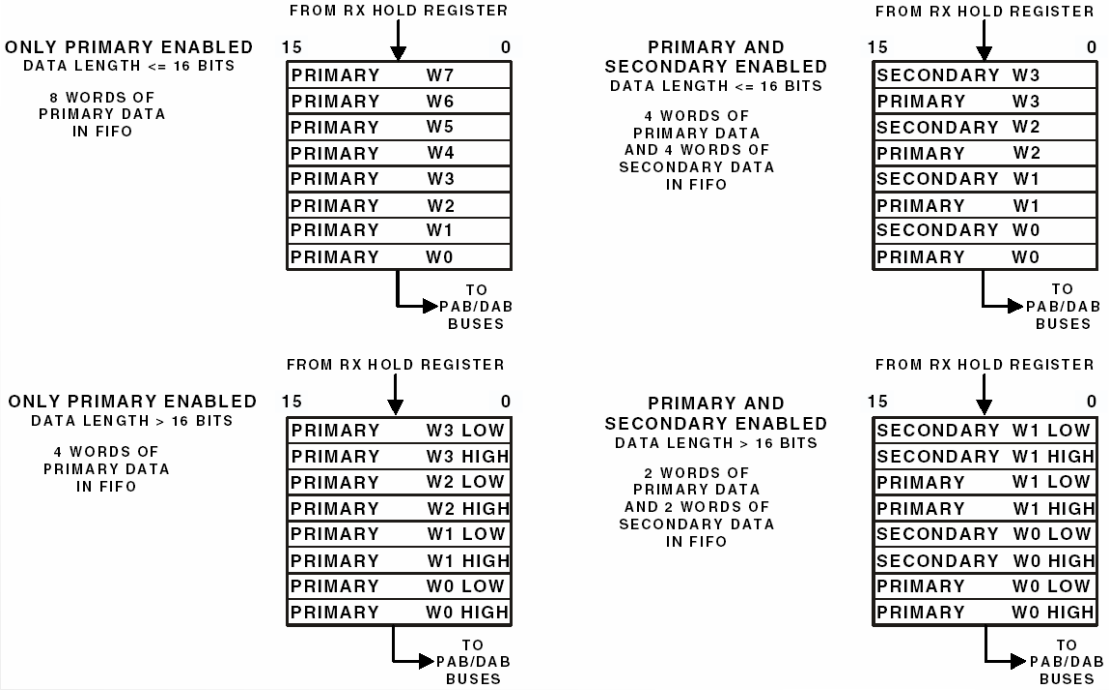
PRIMARY AND SECONDARY ENABLED
DATA LENGTH > 16 BITS

2 WORDS OF PRIMARY DATA AND 2 WORDS OF SECONDARY DATA IN FIFO

15	0
SECONDARY	W1 LOW
SECONDARY	W1 HIGH
PRIMARY	W1 LOW
PRIMARY	W1 HIGH
SECONDARY	W0 LOW
SECONDARY	W0 HIGH
PRIMARY	W0 LOW
PRIMARY	W0 HIGH

- **Writes to SPORTx_TX write to Transmit FIFO**
  - **Reads cause PAB bus error**
- **Transmit FIFO data ordering is dependant on TXSE and SLEN bits**
  - **TXSE = 1 enables Secondary (Primary always enabled)**
  - **SLEN selects word length (3<SLEN<32)**

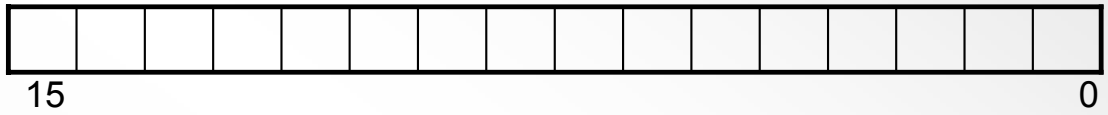
SPORTx_RX Register and Receive FIFO



- **Reads from SPORTx_RX read the Receive FIFO**
  - **Writes cause PAB bus error**
- **Receive FIFO data ordering is dependant on RXSE and SLEN bits**
  - **RXSE = 1 enables Secondary (Primary always enabled)**
  - **SLEN selects word length (3<SLEN<32)**

Serial Clock Divider

SPORTx_TCLKDIV and SPORTx_RCLKDIV are each 16-bit registers



- **Used for internally generated clock**
- **SPORTx_T/RCLK freq = $\frac{\text{SCLK frequency}}{2 * (\text{SPORTx\_T/RCLKDIV} + 1)}$**

Example:

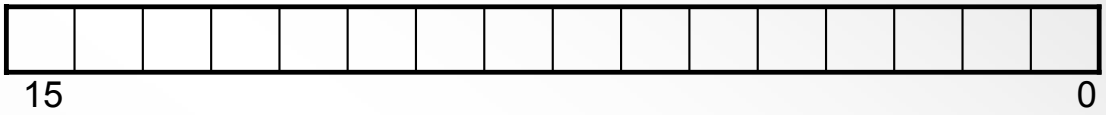
If SCLK is 133MHz, what RCLKDIV is required for a 13.3MHz RCLK rate?

- **Answer:**

$\text{RCLKDIV} = \frac{133\text{MHz}}{2 * 13.3\text{MHz}} - 1 = 4$

Frame Sync Divider

SPORTx_TFSDIV andSPORTx_RFSDIV are each 16-bit registers

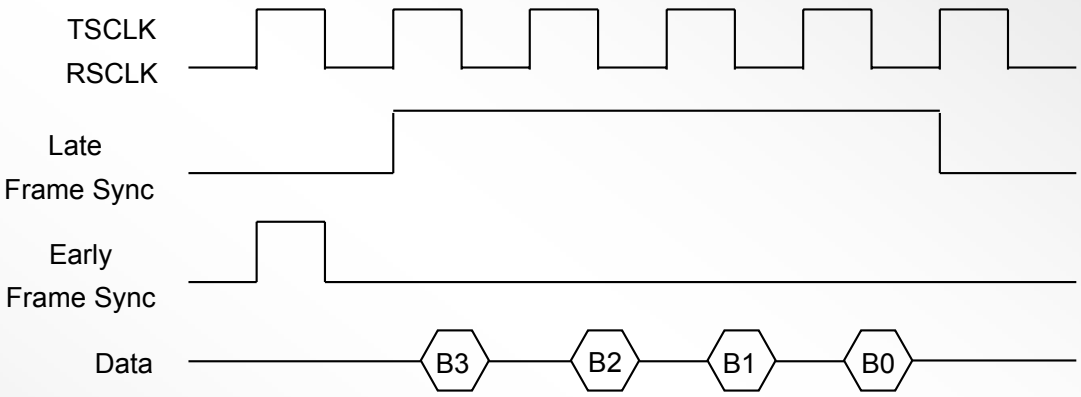


- **Used for internally generated frame syncs**
- **Number of cycles between FS assertions = T/RFSDIV + 1**
- **SPORTx_T/RFS freq = $\frac{\text{T/RSCLKx frequency}}{\text{SPORTx\_T/RFSDIV + 1}}$**
- **Example:**

  If RCLK is 13.3 MHz, what RFSDIV is required for a 48kHz RFS rate?
- **Answer:**

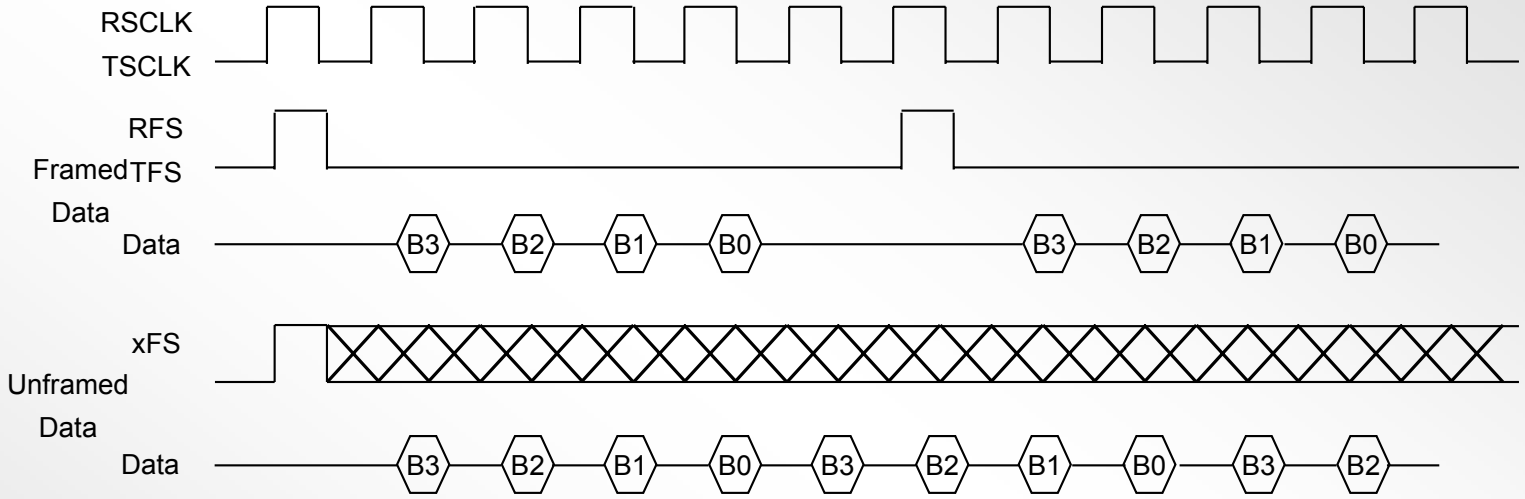
  RFSDIV = $\frac{13.3\ \text{MHz}}{48\text{kHz}}$ - 1 = 276

Serial Port Timing Characteristics
Early vs. Late Framing



- **Early framing: LAxFS=0**
  - **frame sync precedes data by one serial clock cycle.**
- **Late framing: LAxFS=1**
  - **frame sync checked on first bit only**
- **Data transmitted MSB first (xLSBIT=0) or LSB first (xLSBIT=1)**
- **Frame sync, TSCLK and RSCLK generated internally or externally**

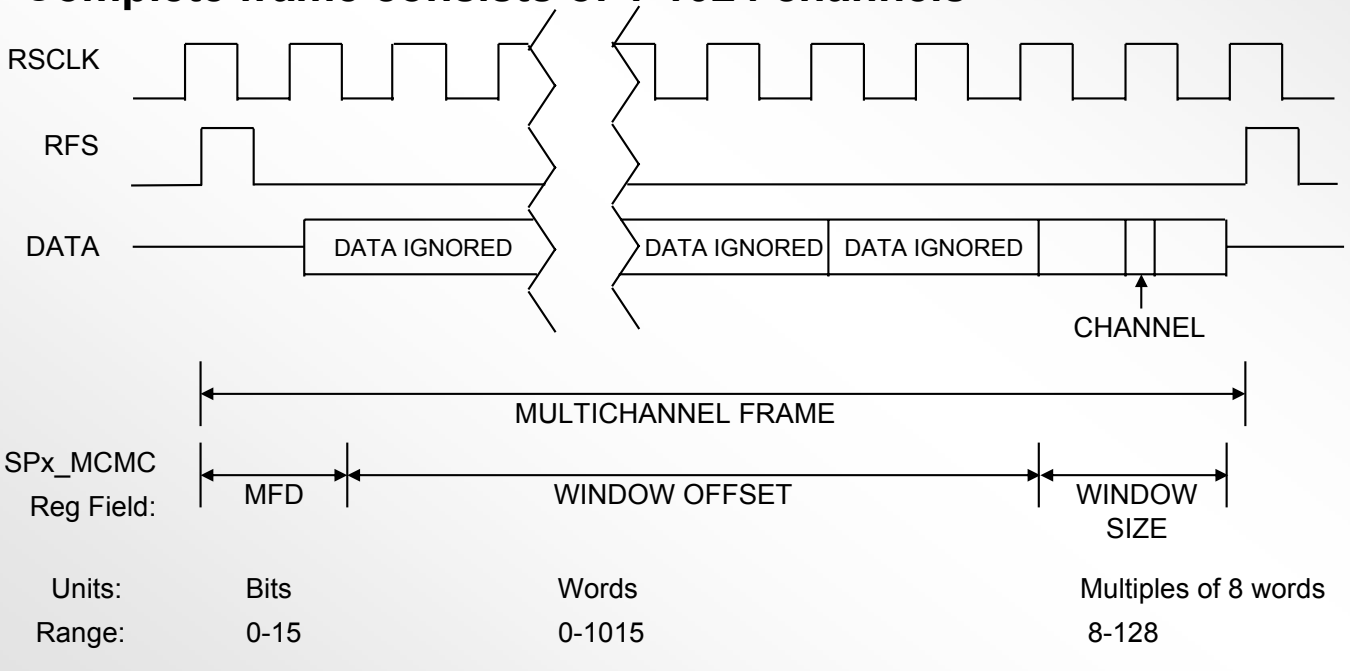
Serial Port Timing Characteristics Framed vs. Unframed Data



- **Framed mode: TFSR/RFSR = 1**
  - **Requires a framing signal for every word.**
- **Unframed mode: TFSR/RFSR = 0**
  - **Ignores framing signal after first word.**
- **Active low or active high frame syncs selected with LTFS and LRFS bits of SPORTx_TCR1 and SPORTx_RCR1 control registers**

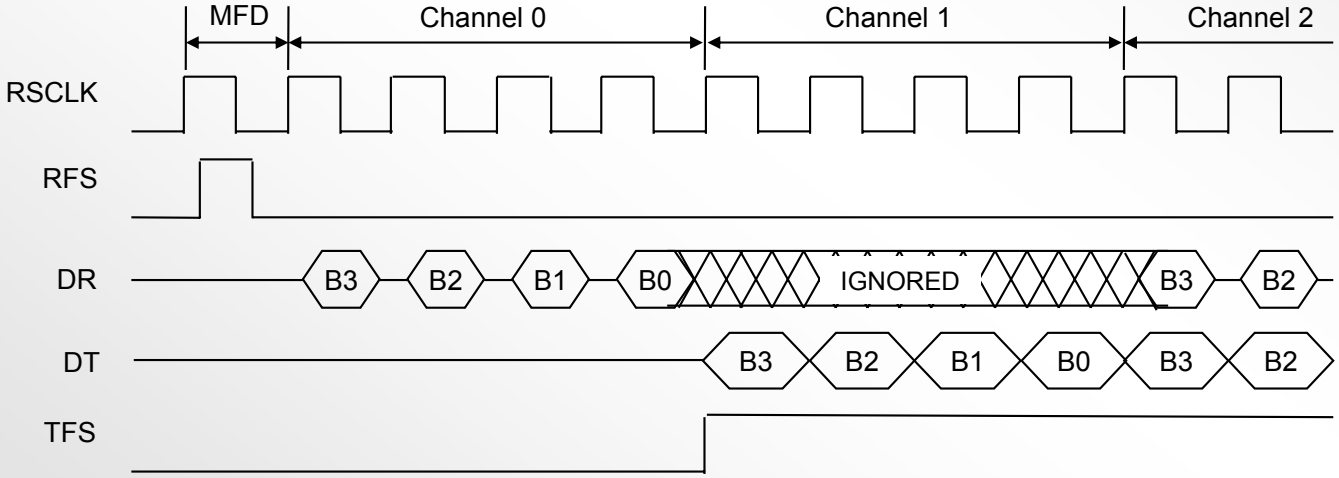
Multichannel Frame

- Contains more then one channel
- Specified by the window size and offset
- Complete frame consists of 1-1024 channels



Multichannel Operation

- TDM (time-division-multiplexed) method where serial data is sent/received on different channels sharing the same serial bus.
- TDM channels: 128 of 1024 total channels
- RFS signals start of frame
- TFS is used as Transmit Data Valid (TDV) for external logic. Active only during transmit channels
- 2D DMA features are useful create channel buffers in memory



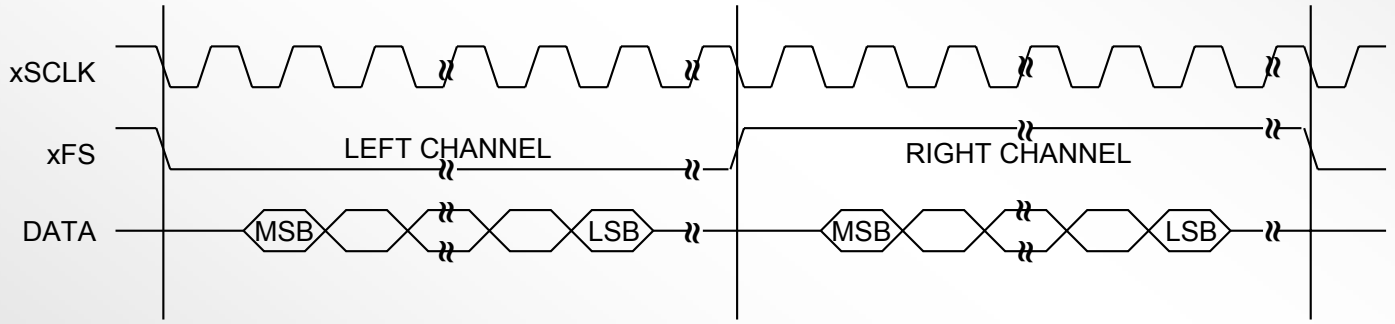
Example: Receive on channel 0 & 2, Transmit on channel 1

SPORT I²S Operation

- **Industry standard developed by Philips for stereo transmission of audio over a 3-wire interface**
- **Data always transmits in MSB format**
- **Can select either DMA-driven or interrupt driven transfers**
- **Consists of Serial Clock, Word Select and Data**
- **SPORT data programmability either allows up to:**
  - 4 I²S transmitters for 8 output audio channels
  - 4 I²S receivers for 8 input audio channels

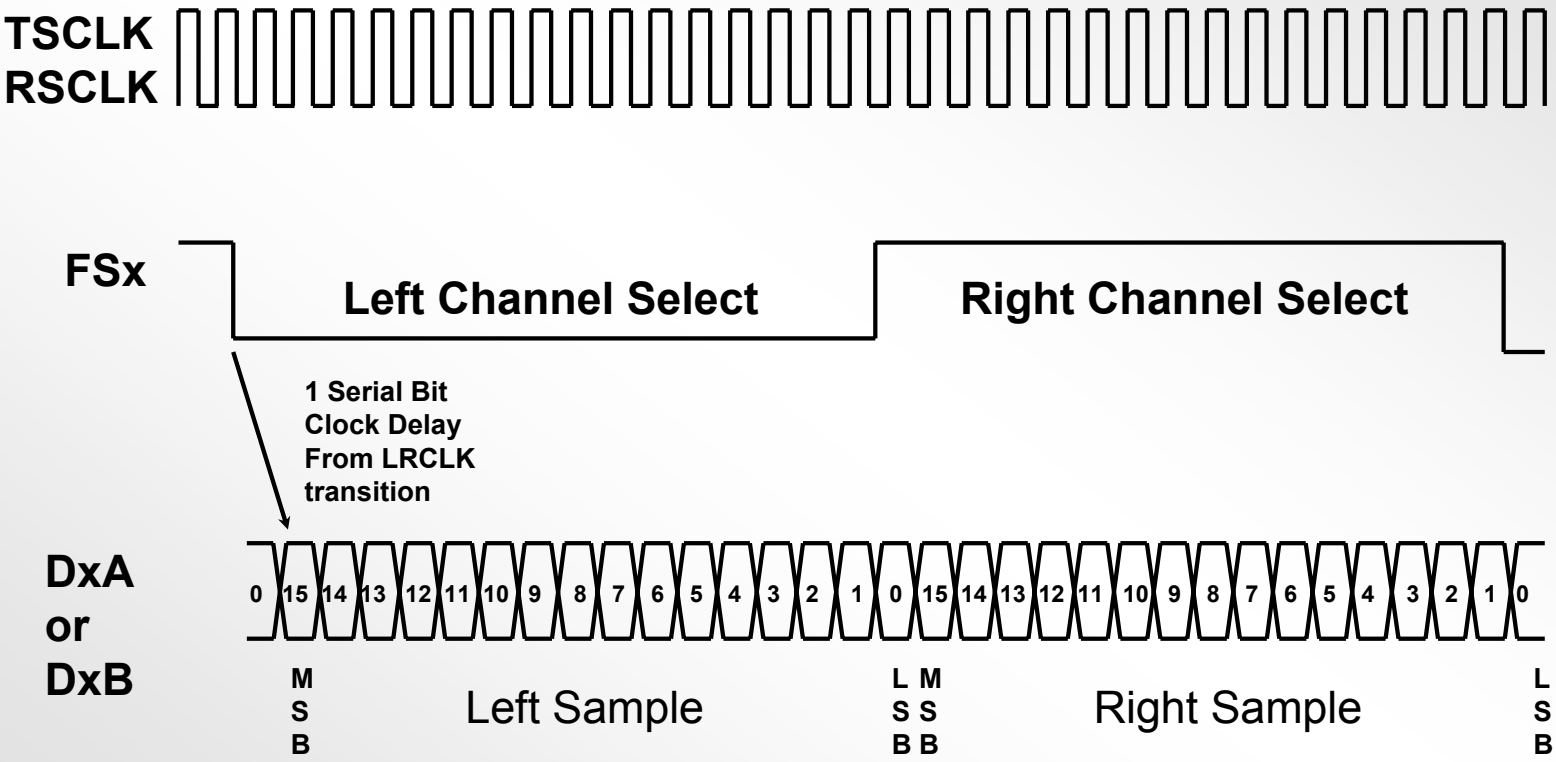
I²S Operation

- Supports up to 8 I²S stereo devices
  - 4 Transmit, 4 Receive
- Frame sync pins become word select signals
- Word select changes state one SCLK period before MSB is transmitted



I2S MODE – 3 TO 32 BITS PER CHANNEL

I²S Serial Protocol



Setting I2S Mode

Bit Field	Stereo Audio Serial Scheme		
	I^2S	Left-Justified	DSP Mode
RSFSE	1	1	0
RRFST	0	0	0
LARFS	1	0	1
LRFS	1	0	0
RFSR	1	1	1
RCKFE	0	0	0
SLEN	2 - 31	2 - 31	2 - 31
RLSBIT	0	0	0
RFSDIV (If internal FS is selected.)	2 - Max	2 - Max	2 - Max
RXSE (Secondary Enable is available for RX and TX.)	X	X	X

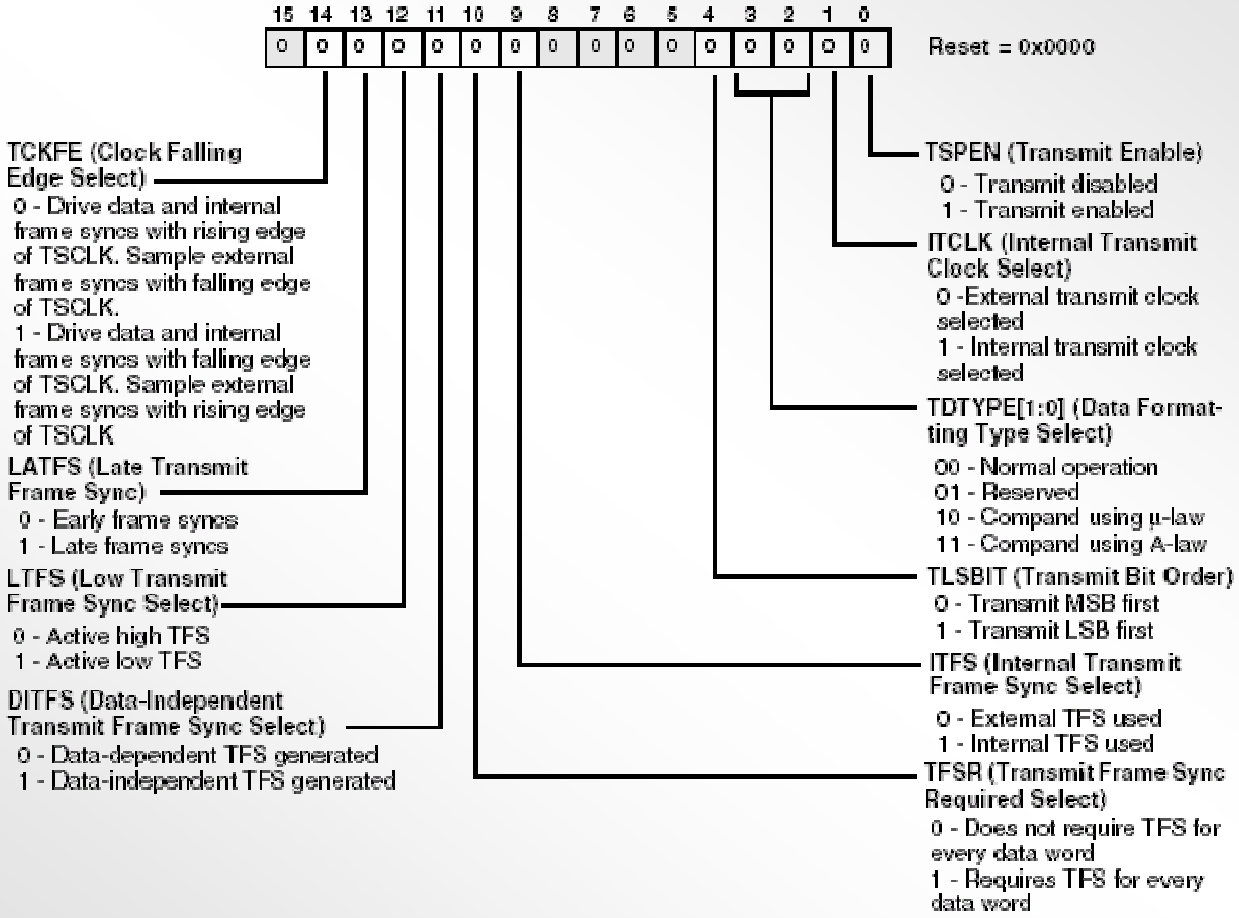
- There are similar bits in the Transmit control register

ADSP-BF533 SPORT MMRs

- Access serial port data through memory-mapped registers: SPORTx_TX, SPORTx_RX
- Configure SPORTx through memory-mapped control registers:
  - SPORTx_TCR1/2 - Transmit Control Register 1 and 2
  - SPORTx_TSCLKDIV - Transmit Clock Divisor
  - SPORTx_TFSDIV - Transmit Frame Sync Divisor
  - SPORTx_RCR1/2 - Receive Control Register 1 and 2
  - SPORTx_RSCLKDIV - Receive Clock Divisor
  - SPORTx_RFSDIV - Receive Frame Sync Divisor
  - SPORTx_MCMC1/2 - Multichannel Configuration 1 and 2
  - SPORTx_MRCS0-3 - Multichannel Channel Select
  - SPORTx_MTCS0-3 - Multichannel Channel Select

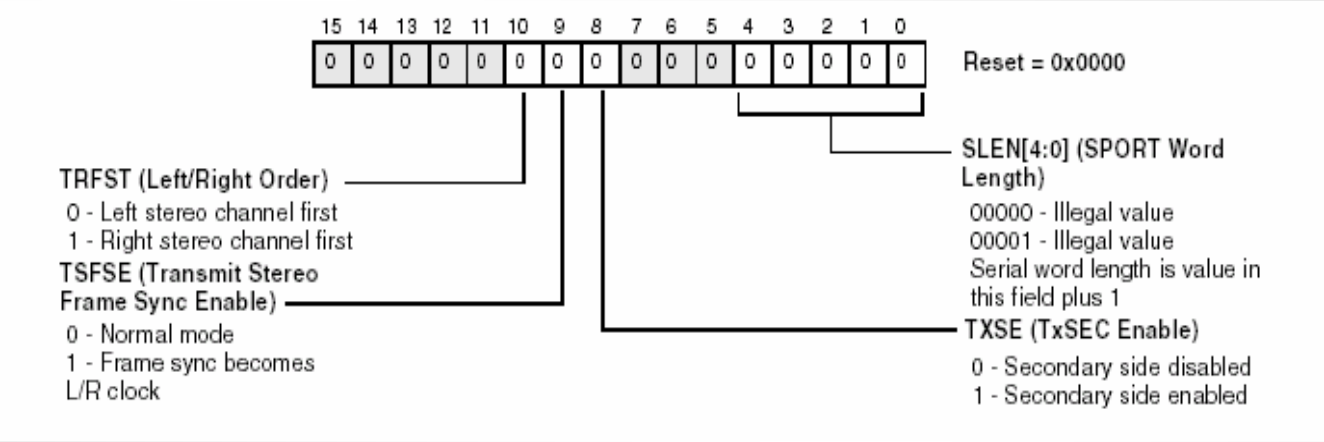
SPORTx Transmit Configuration Registers

SPORTx Transmit Configuration 1 Register (SPORTx_TCR1)



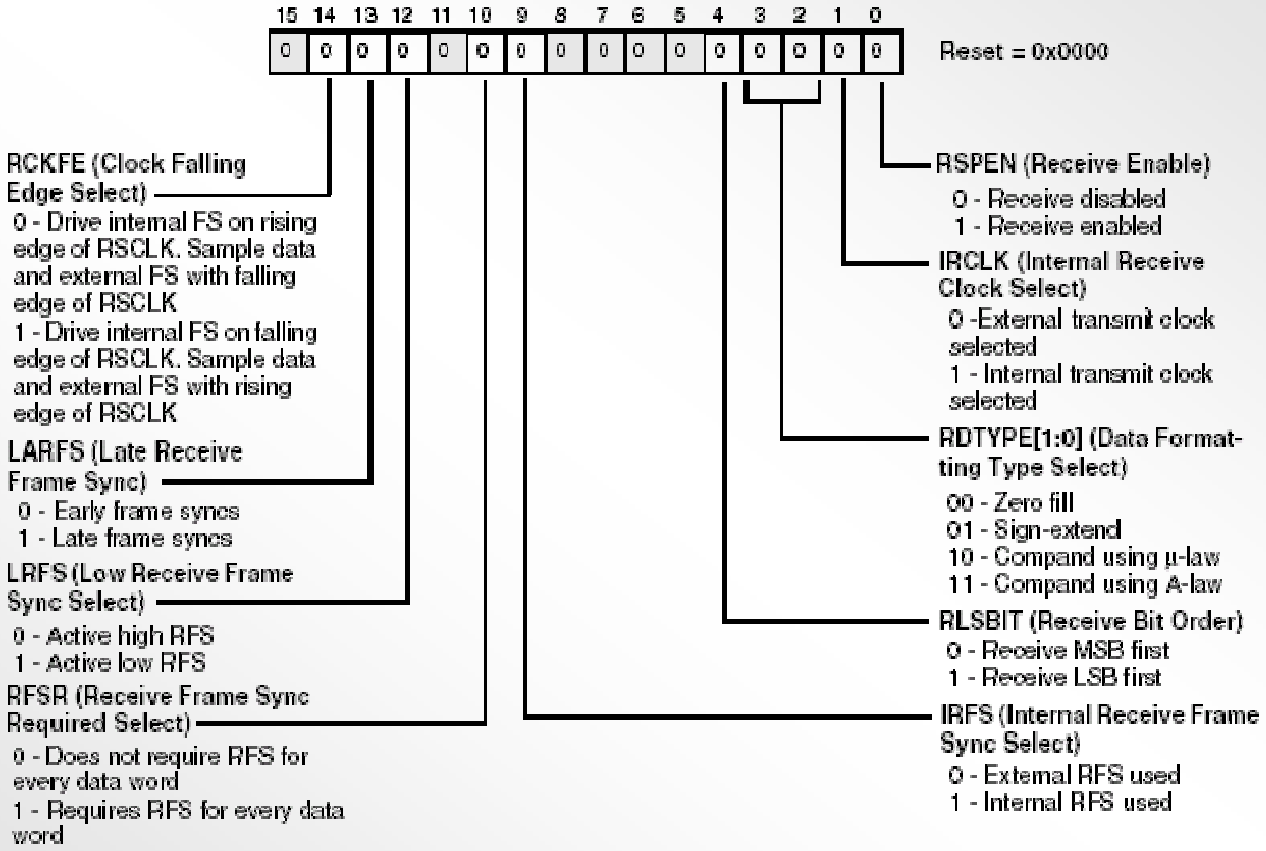
SPORTx Transmit Configuration Registers

SPORTx Transmit Configuration 2 Register (SPORTx_TCR2)



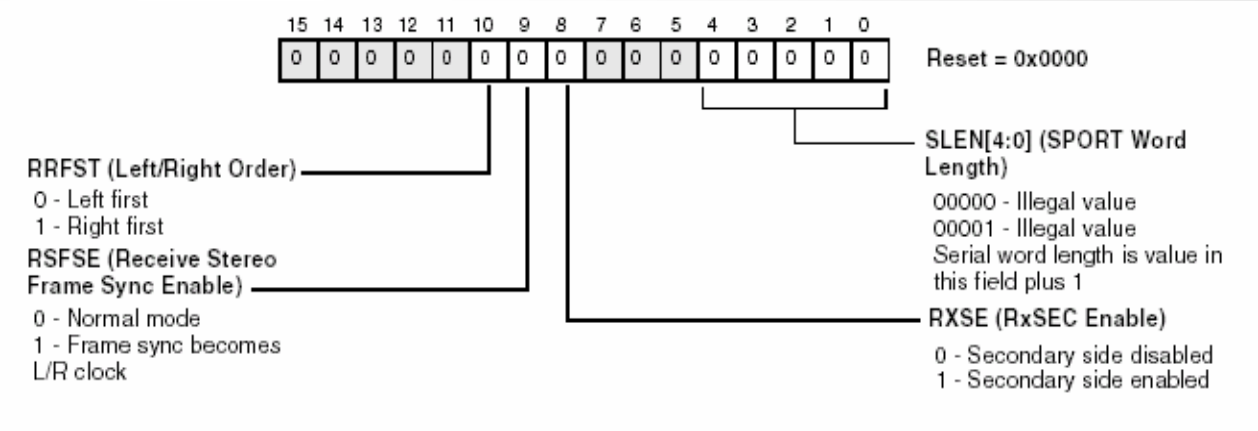
SPORTx Receive Configuration Registers

SPORTx Receive Configuration 1 Register (SPORTx_RCR1)



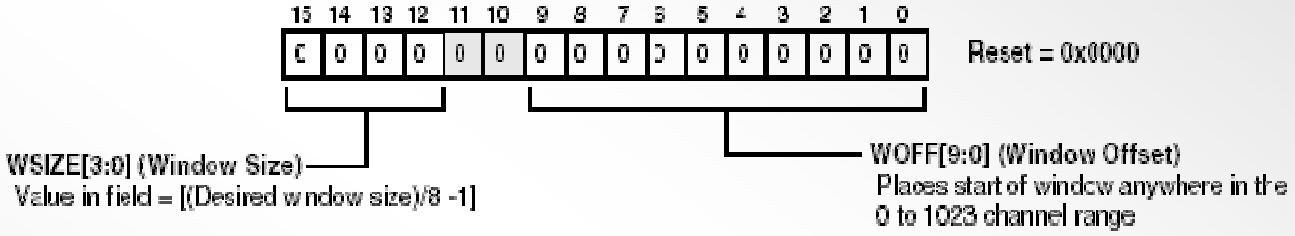
SPORTx Receive Configuration Registers

SPORTx Receive Configuration 2 Register (SPORTx_RCR2)

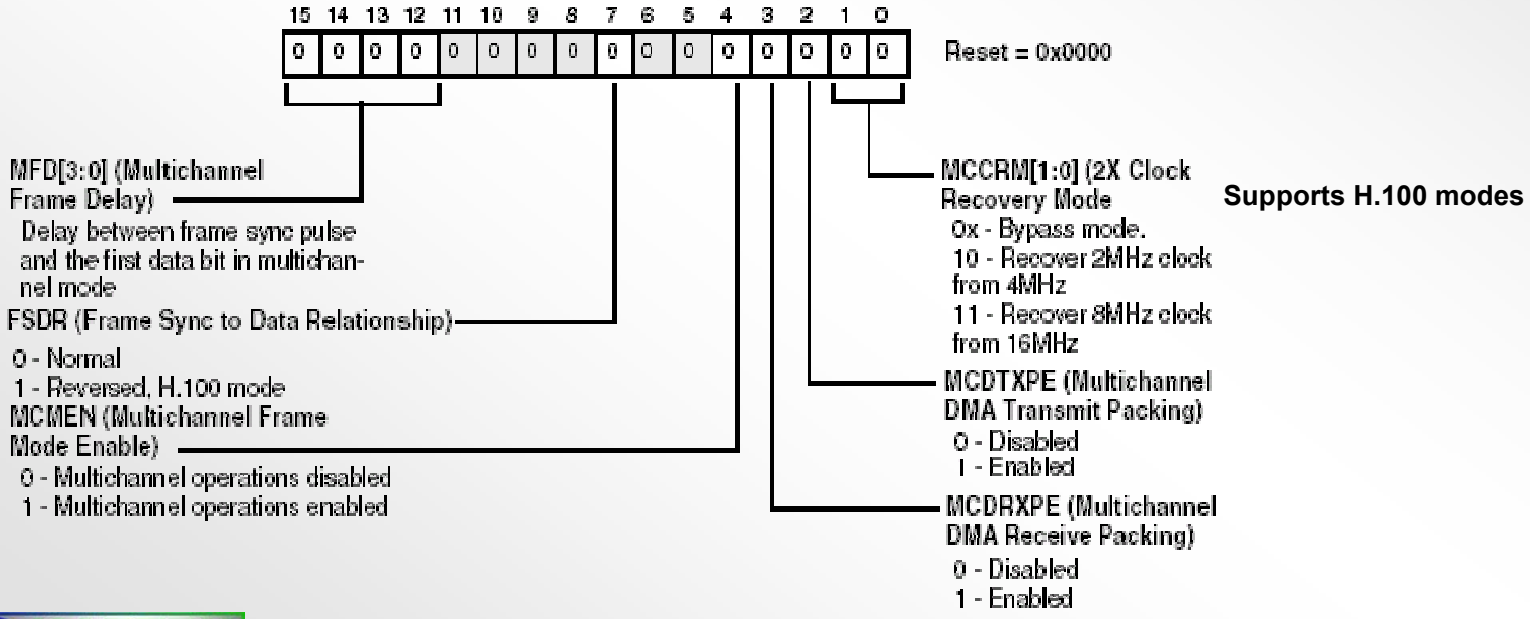


Multi-Channel Registers

SPORTx Multichannel Configuration Register 1 (SPORTx_MCMC1)



SPORTx Multichannel Configuration Register 2 (SPORTx_MCMC2)



Supports H.100 modes

Serial Peripheral Interface (SPI)

ADSP-BF533 SPI Features

- **One SPI-Compatible Port**
- **4 Pin Interface (MOSI, MISO, ~SPISS, SCK)**
- **Master and Slave Mode Operation**
  - Supports Multimaster Environments
- **Can Use 8 GP Flag Pins As Slave-Select Lines**
  - 1 Slave Select *Input* Pins
  - 7 Slave Select *Output* Pins
- **Gated SPI Clock (Only Active During Transfers)**
- **DMA Support**
  - One DMA Channel (Transmit or Receive)
- **Programmable Baud Rate**
- **Programmable Clock Polarity and Phase**
- **Programmable Serial Word Length (8 or 16 Bits)**

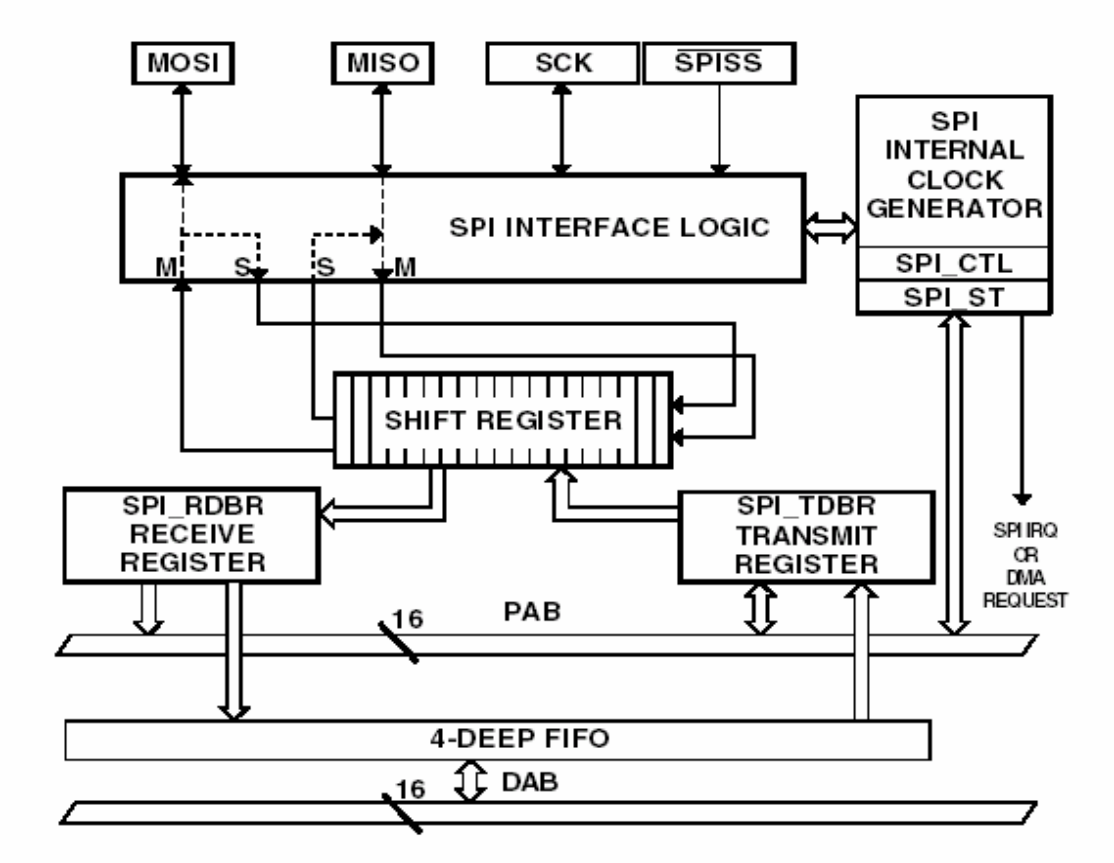
Pin Descriptions And Uses

- **Serial Peripheral Interface Clock (SCK)**
  - Driven By The Master Device
  - Cycles Once For Every Bit Transmitted
    - Programmed Baud Rate
  - Gated Clock – Only Active During Transfers
  - Ignored By Slave Devices With Inactive Slave-Select
  - Shifts Data Out On One Edge And Samples On The Other
  - Polarity And Phase Are Programmable
- **Master Out Slave In (MOSI) / Master In Slave Out (MISO)**
  - Bi-directional I/O Data Pins
    - Direction Depends On Whether Device Is Master Or Slave

Pin Descriptions And Uses (Cont)

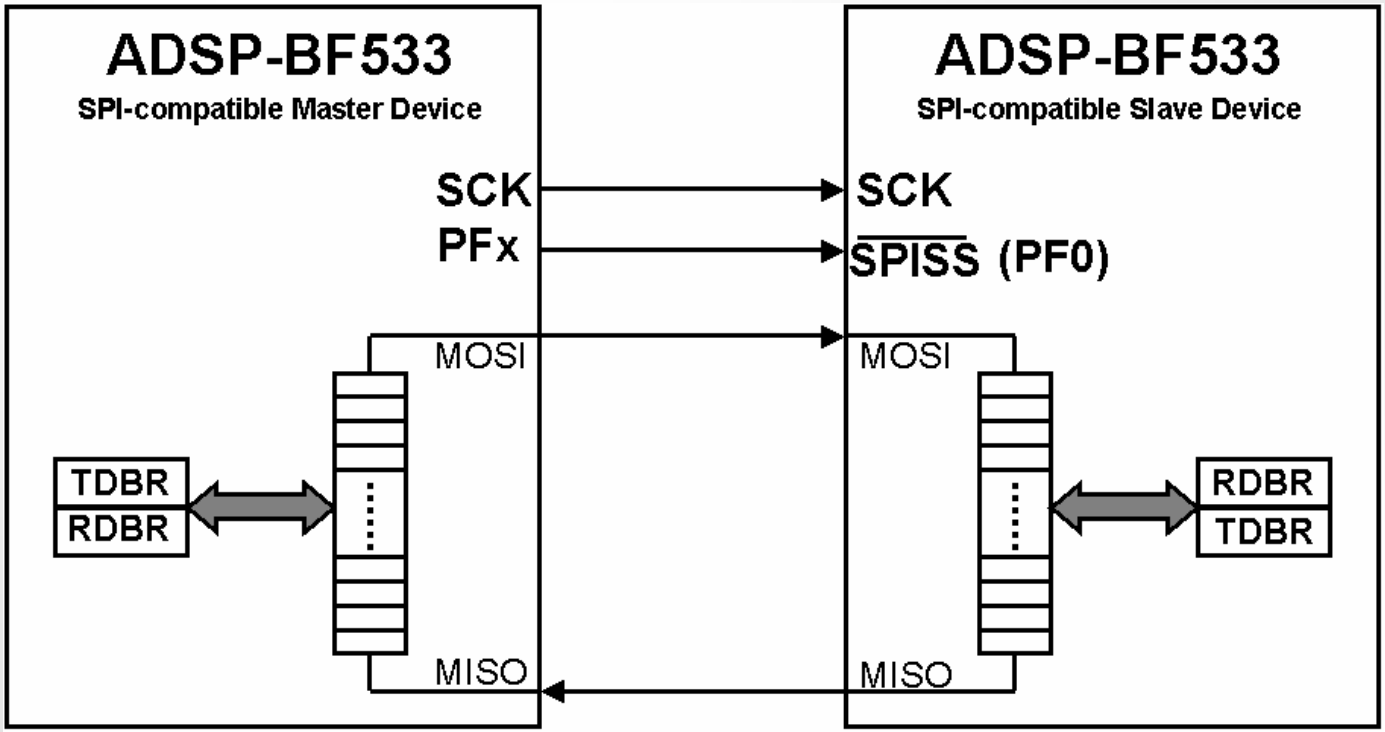
- **Serial Peripheral Interface Slave-Select (~SPISS, PF0)**
  - For An SPI Slave Device
    - Active Low Device Select Input Signal Provided By Master
  - For An SPI Master Device
    - Error-Detection Pin
      - Useful In A Multi-Master Environment
      - If Asserted, Another Device Is Trying To Be The Master
      - Feature Is Enabled By Setting PSSE Bit In SPI_CTL Register
  - Monitor ~SPISS Value In FIO_FLAG_S, FIO_FLAG_C, and FIO_FLAG_D Registers

ADSP-BF533 SPI Diagram



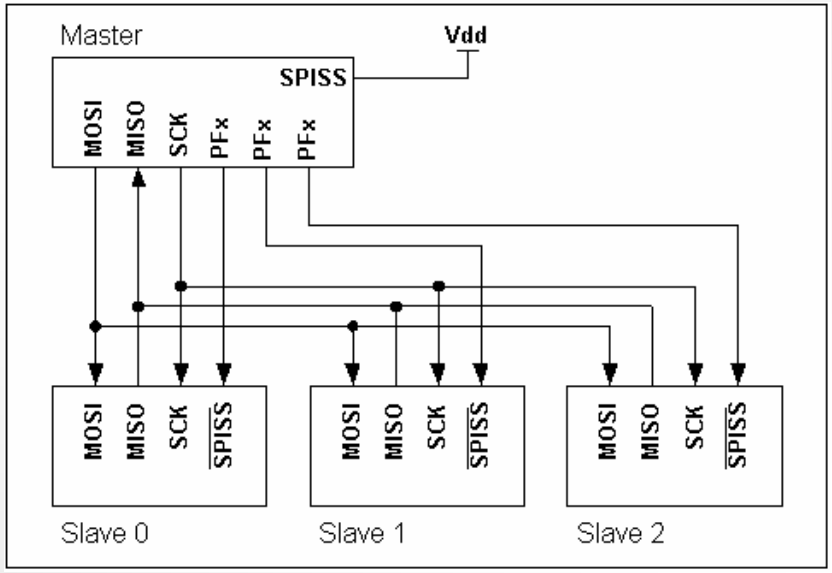
A Closer Look At How The Data Is Moved

- Shift Registers Simultaneously Shift Data In And Out



Serial Peripheral Interface Generic Example

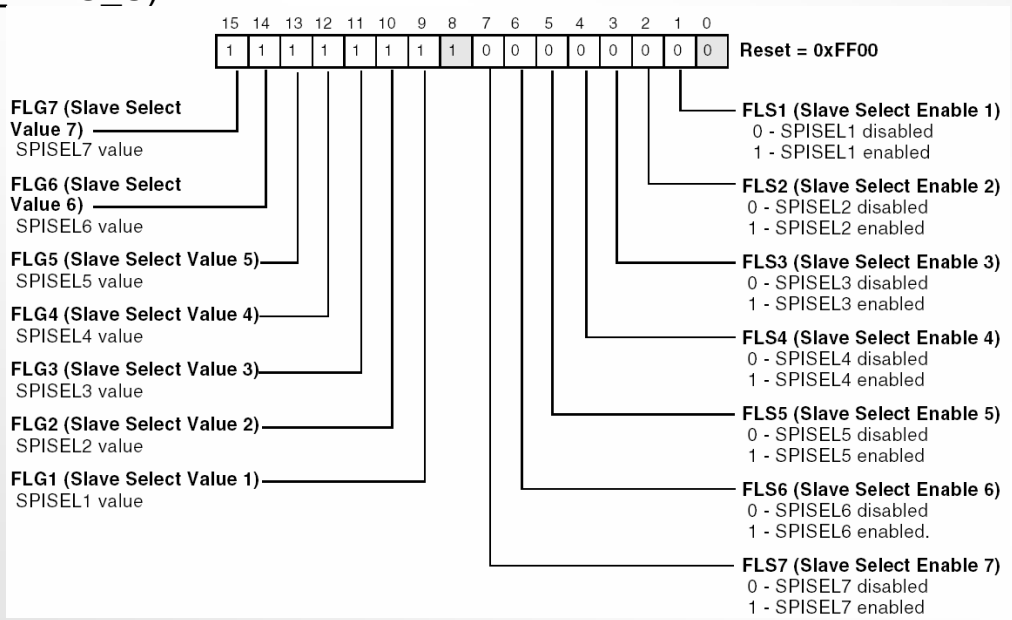
- 4-Wire Synchronous, Full-Duplex Interface



For Broadcast Write, All PFx Pins on Master Asserted and Only 1 Slave Sends Data Over MISO

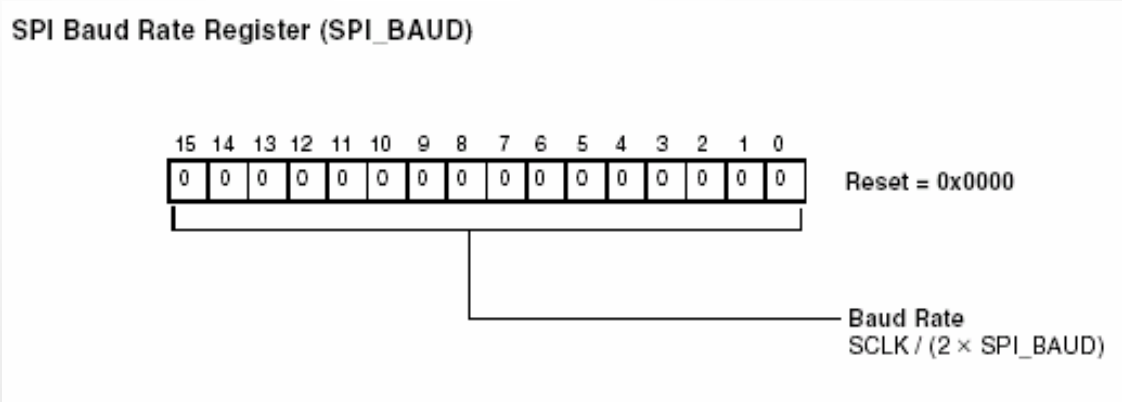
SPI_FLG Register

- **Write SPI Flag Register (SPI_FLG) (Master Only)**
  - 7 Control Flags = 7 Possible Slave Devices
  - Written 1st - Slaves Are Deselected During Master Configuration
  - Clock Phase Bit (CPHA) In SPI_CTL Determines Handling Of Slaves
  - Unused Flags Controlled By Flag Registers (FIO_FLAG_D, FIO_FLAG_C, FIO_FLAG_S)

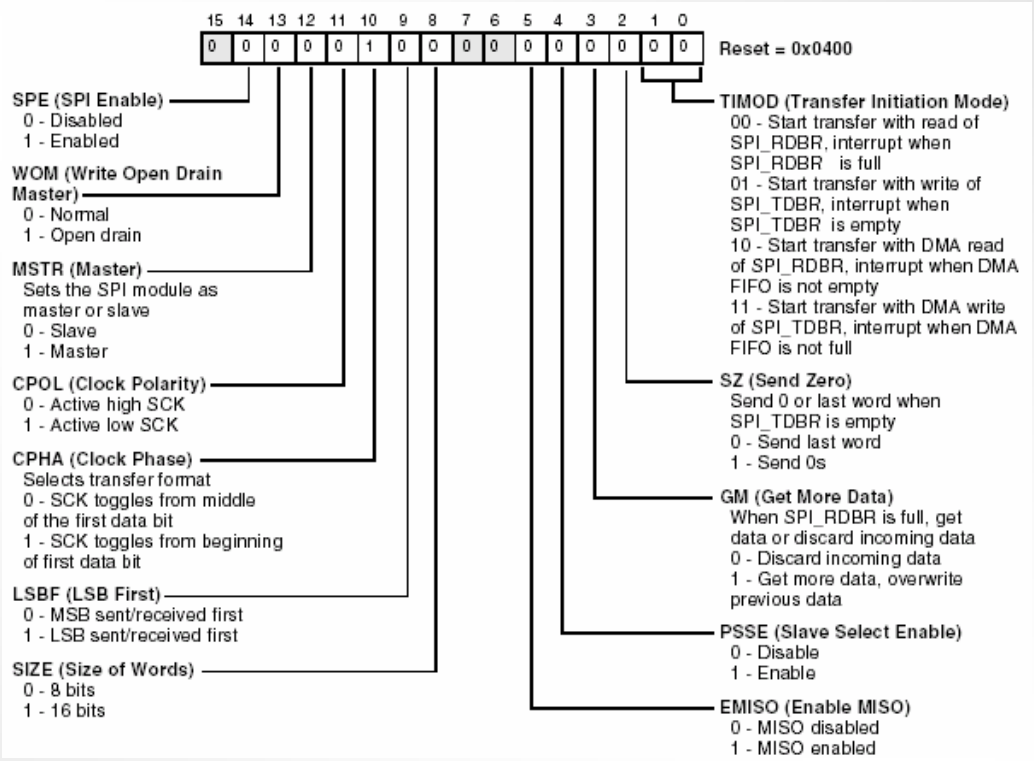


SPI_BAUD Register

- **Write SPI Baud Rate Register (SPI_BAUD) (Master Only)**
  - Slave Devices Ignore Writes To SPI_BAUD
  - Writing 0 or 1 Disables SPI Clock
    - Maximum SPI Clock Is One-Fourth System Clock (SCLK)



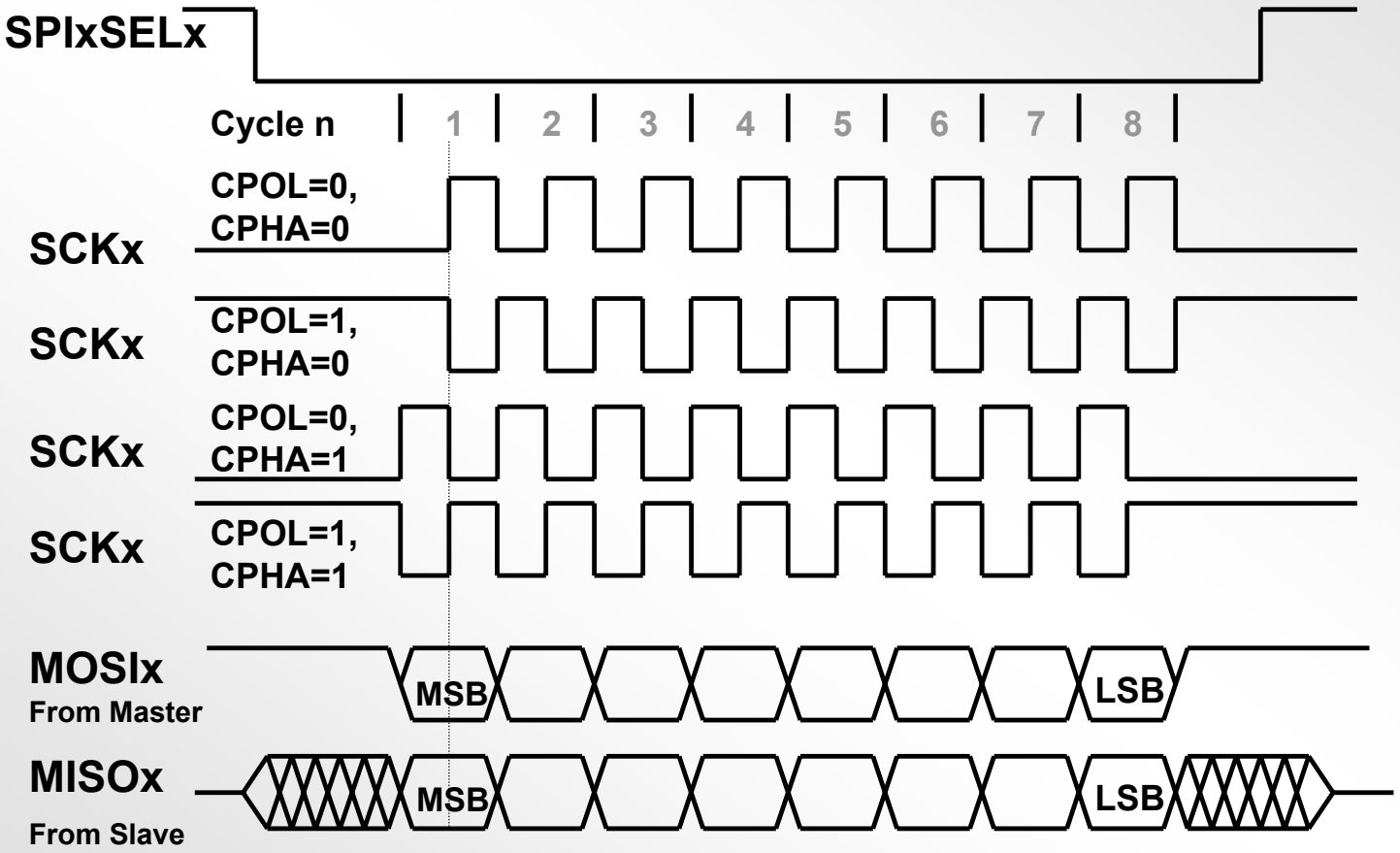
SPI Control Register



- **CPHA Bit Also Controls Whether Hardware or Software is Responsible For Toggling Slave Select Signals In SPI_FLG**
  - **0 – Hardware Toggles Active Slave Selects Between Words**
  - **1 – Slave Selects Controlled In Software**

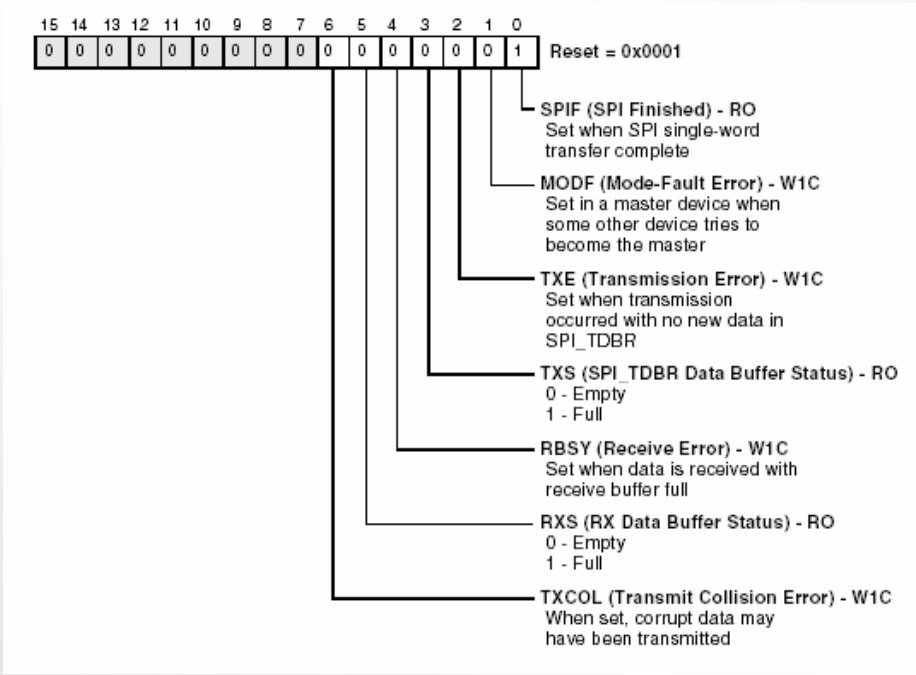
SPI Transfer Protocol

- Example of transfer



Additional ADSP-BF533 SPI Port Registers

- **Receive Data Buffer Shadow (SPI_SHADOW)**
  - **Useful For Debugging, Always Contains Same Data As RDBR**
  - **Reading SPI_SHADOW Does Not Affect The System**
- **SPI Status Register (SPI_STAT)**



SPI DMA Registers

- Refer to DMA section for description
- SPI DMA Register Names (DMA5 is default DMA channel for SPI)
  - DMA5_CONFIG – Configuration
  - DMA5_CURR_DESC_PTR – Current Descriptor Pointer
  - DMA5_NEXT_DESC_PTR – Next Descriptor Pointer
  - DMA5_START_ADDR – Start Address
  - DMA5_CURR_ADDR – Current Address
  - DMA5_X_COUNT – Inner-Loop Count
  - DMA5_CURR_X_COUNT – Current Inner-Loop Count
  - DMA5_Y_COUNT – Outer-Loop Count (2D)
  - DMA5_Y_COUNT – Current Outer-Loop Count (2D)
  - DMA5_X_MODIFY – Inner-Loop Stride
  - DMA5_Y_MODIFY – Outer-Loop Stride
  - DMA5_IRQ_STATUS – Interrupt Status
  - DMA5_PERIPHERAL_MAP – Peripheral Mapping

SPI Interrupts

- **Handling Depends On Transfer Initiate Mode (TIMOD) Selected In SPI_CTL Register**

TIMOD	Function	Transfer Initiated Upon	Action, Interrupt
00	Transmit and Receive	Initiate new single-word transfer upon read of SPI_RDBR and previous transfer completed	Interrupt active when receive buffer is full Read of SPI_RDBR clears interrupt
01	Transmit and Receive	Initiate new single-word transfer upon write to SPI_TDBR and previous transfer completed	Interrupt active when transmit buffer is empty Writing to SPI_TDBR clears interrupt
10	Receive with DMA	Initiate new multiword transfer upon enabling SPI for DMA mode. Individual word transfers begin with a DMA read of SPI_RDBR, and last transfer completed.	Interrupt active when DMA FIFO is not empty. Interrupt clears when DMA FIFO is empty.
11	Transmit with DMA	Initiate new multi-word transfer upon enabling SPI for DMA mode. Individual word transfers begin with a DMA write to SPI_TDBR, and last transfer completed.	Interrupt active when DMA FIFO is not full. Interrupt clears when DMA FIFO is full.

```
P0.H = HI(SPI_RDBR);
P0.L = LO(SPI_RDBR);
R0 = W[P0]
```

```
R0 = 0xFEED (z);
P0.H = HI(SPI_TDBR);
P0.L = LO(SPI_TDBR);
W[P0] = R0;
```

```
R0 = 0x0001;
P0.H = HI(DMA5_IRQ_STATUS);
P0.L = LO(DMA5_IRQ_STATUS);
W[P0] = R0;
```

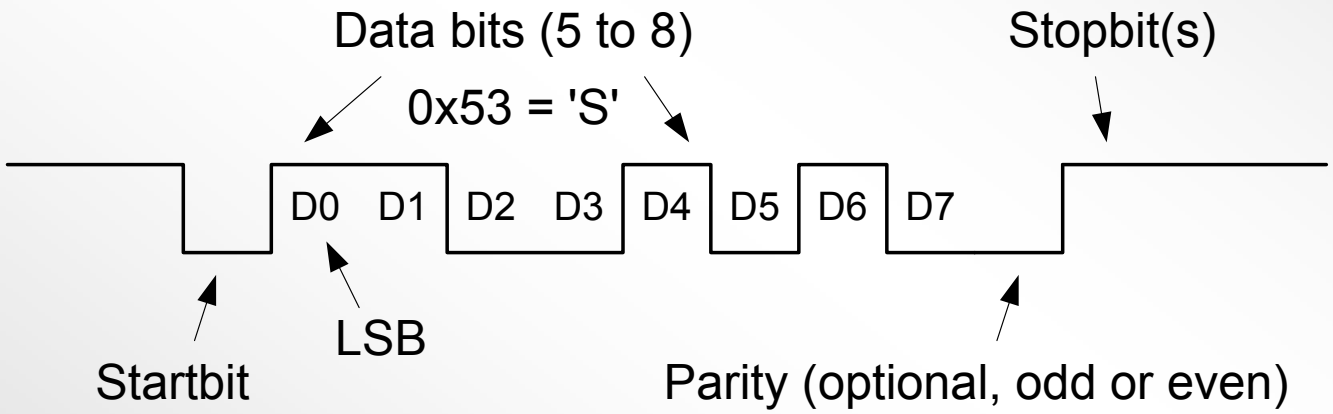
UART

ADSP-BF533 UART Feature Overview

- One UART module
- Industrial Standard 16450 compliant
  - 5-8 data bits
  - 1, 1½ or 2 stop bits
  - None, even or odd parity
  - Baud rate = SCLK/(16*DIVISOR)
  - Loopback mode
- Supports half-duplex IrDA SIR (9.6/115.2 Kbps rate)
- Autobaud detection support through the use of the Timers
- Separate TX and RX DMA support (Register-based and Descriptor-based)

The Universal Asynchronous Protocol

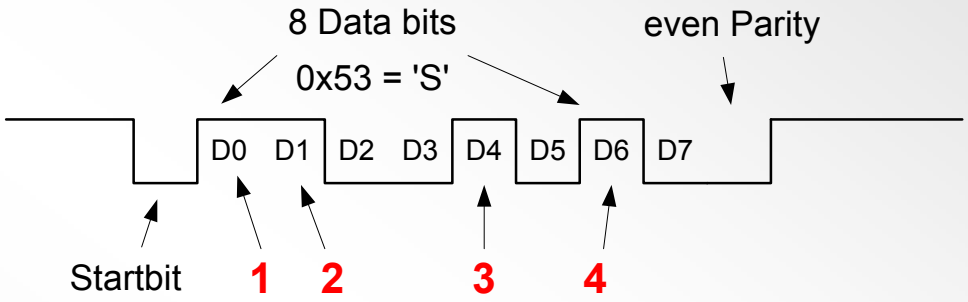
- **2-Signal Asynchronous, Full-Duplex Interface**
- **Common settings for Transmit and Receive for the UART**
- **LSB always sent first**



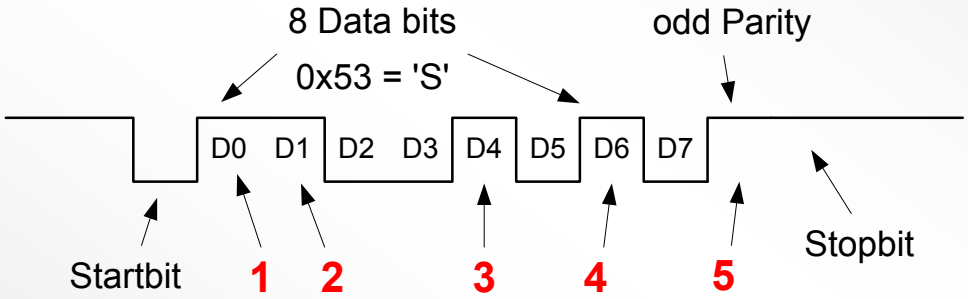
- **UART itself generates the logical TTL bit stream**
- **External Level-Shifter / Inverter / Transformer required**

Parity Bit

- **Optional Parity Bit Helps To Detect Corrupted Data**
- **Counts ALL the High Data Bits Including the Parity Bit**
- **Even Parity (low)**



- **Odd Parity (high)**

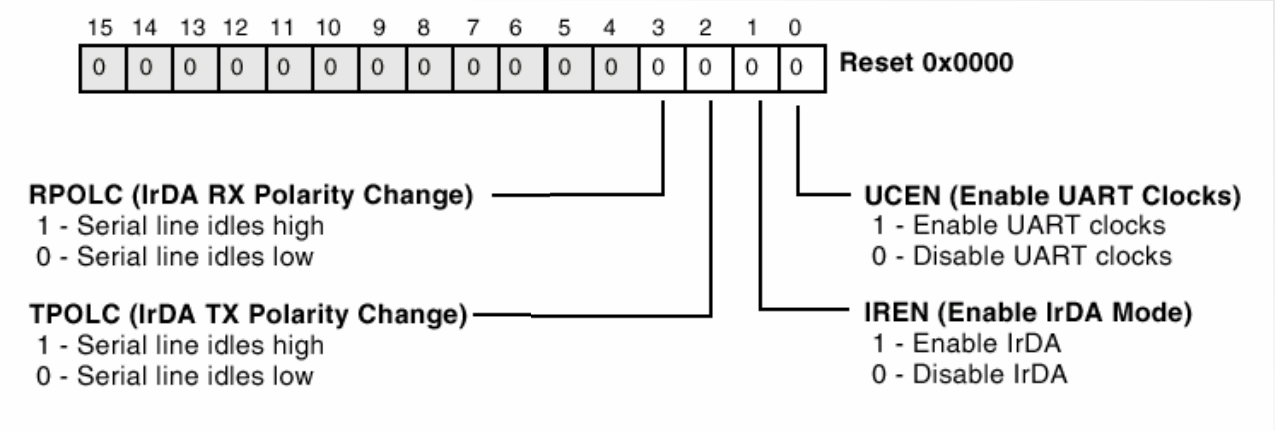


- **Optionally, Parity Bit may stick (mark or space)**

UART Control Registers

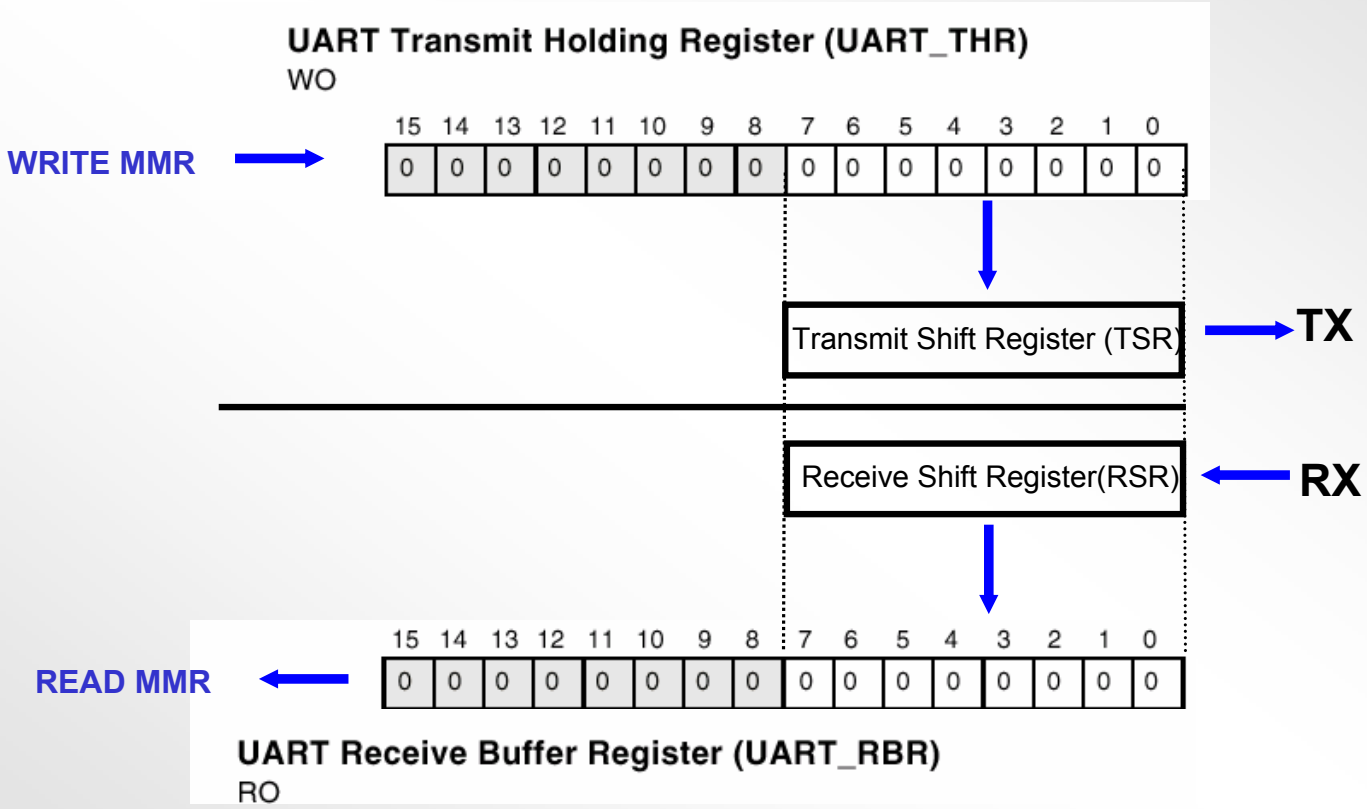
- Global Control Register
  - UART_GCTL
- Data Registers (buffered)
  - UART_THR – Transmit holding register
  - UART_RBR – Receive buffer register
- Frame Format and Status Registers
  - UART_LCR – Line control register
  - UART_LSR – Line status register
- Loopback Mode Control Register
  - UART_MCR – Modem Control Register
- Interrupt Control Registers
  - UART_IER – Interrupt enable register
  - UART_IIR – Interrupt identification register
- Bit Rate Control Registers
  - UART_DLL – Divisor latch low-byte
  - UART_DLH – Divisor latch high byte

Global Control Register UART_GCTL

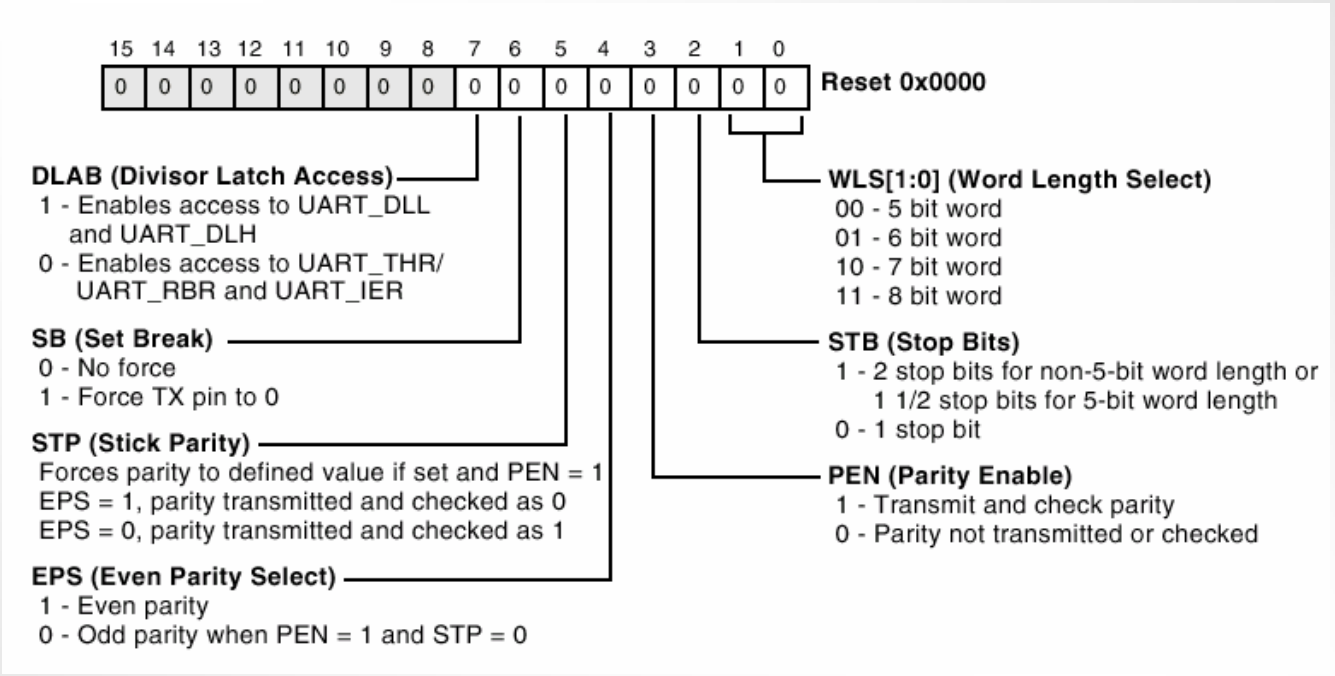


- **Must Enable UART Clocks (UCEN) before any other UART operation**
- **Typically when using IrDA Mode RPOLC=1 and TPOLC=0**

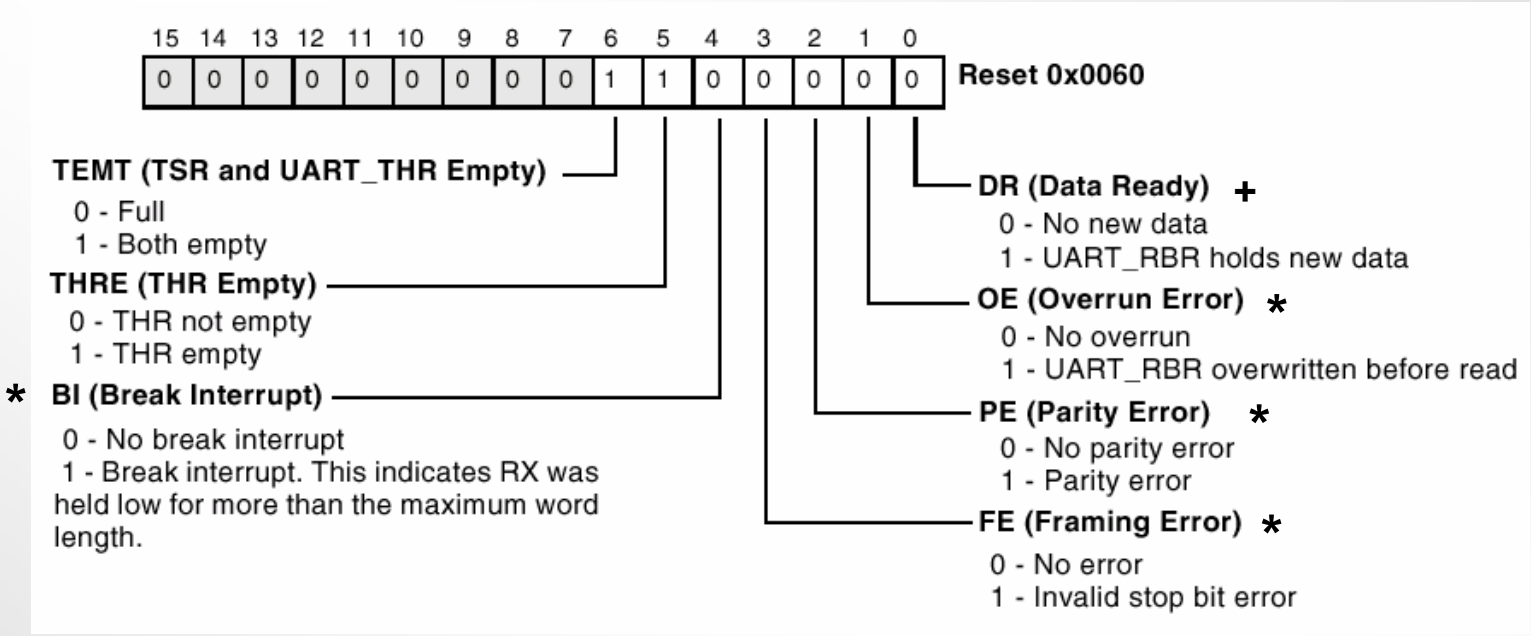
Buffered Transmit and Receive Channels



Line Control Register (UART_LCR)



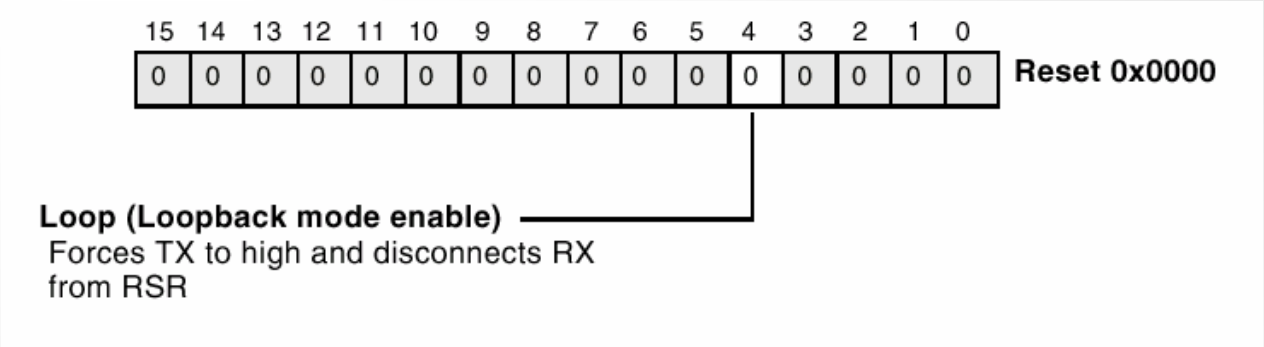
Line Status Register (UART_LSR)



* Status Bit Cleared When UART_LSR Register Read

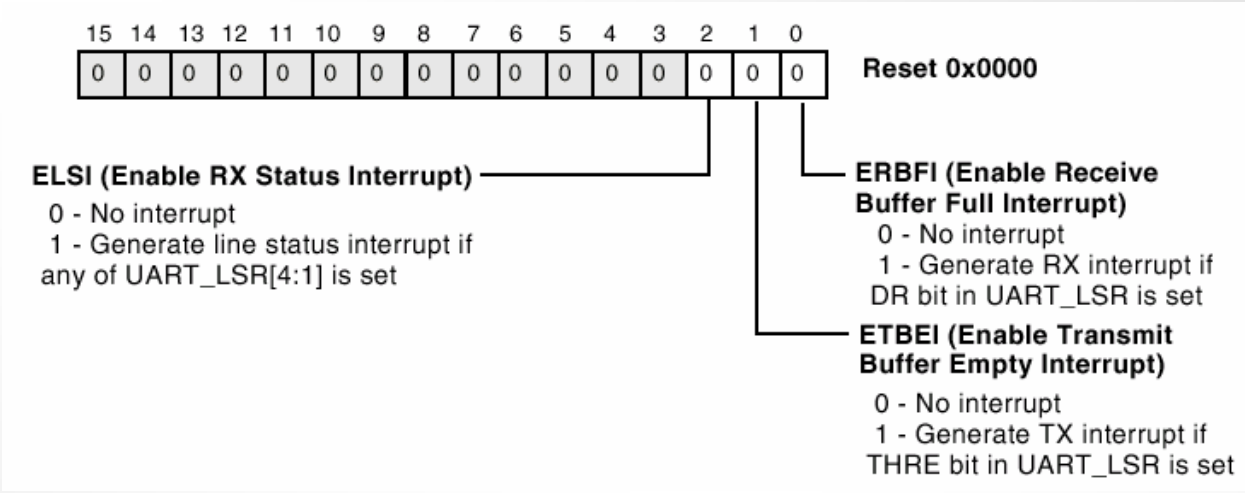
\+ Status Bit Cleared When UART_RBR Register Read

Modem Control Register (UART_MCR)



- Used for testing purposes

Interrupt Enable Register (UART_IER)



- **Non-DMA Mode**
  - **Must enable corresponding bits in UART_IER and poll SIC_ISR or enable interrupt in SIC_MASK.**
- **DMA Mode**
  - **ERBFI and ETBEI must be enabled to act as DMA request lines.**
  - **Can enable ELSI interrupt.**

Example of Receive Polling – Part 1

```
// UART Global Control Register
        p0.l = lo(UART_GCTL);
        p0.h = hi(UART_GCTL);

// Enable UART Clocks!!
        r1 = UCEN(z);
        w[p0] = r1;

// Parity Enable, Word Length = 8 bit
        r1 = PEN | WLS(8) (z);
        w[p0+UART_LCR-UART_GCTL] = r1;

// Enable Receive Buffer Full and Receive Status Interrupts
        r1 = ERBFI | ELSI (z);
        w[p0+UART_IER-UART_GCTL] = r1;

// System Interrupt Status Register
        p2.l = lo(SIC_ISR);
        p2.h = hi(SIC_ISR);
```

Example of Receive Polling – Part 2

```
// Poll SIC_ISR
receive_polling:
        r2 = w[p2] (z);
        CC = bittst (r2, bitpos (IRQ_UART_RX));
        if !CC jump receive_polling;
data_ready:
        csync;

// Read Status
        r1 = w[p0+UART_LSR-UART_GCTL] (z);

// Read Data
        r0 = w[p0+UART_RBR-UART_GCTL] (z);
// If Line Error
        CC = bittst (r2, bitpos (IRQ_UART_ERROR));
        if CC jump error_handler;

// Save Received word to Memory
        [i0++] = r0;
        jump receive_polling;
```

Mixing Non-DMA Mode and DMA Mode

- Non-DMA Mode Uses Different Synchronization Mechanisms Than The DMA Mode Does
  - Application Must Not Mix The Two Mechanisms
- To Switch From One Mode To The Other, The Program Should Wait Until The Ongoing Transfer Has Been Finished

Programmable Bit rate

- Bitrate is derived from peripheral clock (SCLK)
- Applies to TX and RX

$$Bitrate = \frac{SCLK}{16 \times DIVISOR}$$

- DIVISOR is a 16-Bit register formed by two byte registers
  - DLL And DLH (DIVISOR = 65536 when DLL = DLH = 0)
  - Resets To 0x0001
- To read/write DLL And DLH, the Divisor Latch Access Bit (DLAB) in the Line Control Register (LCR) register must be set

Bit rate Deviations

- **DIVISOR is a fraction of SCLK**

$$\frac{133MHz}{16 \times 866} = 9599 \neq 9600$$

Error = 0.013%

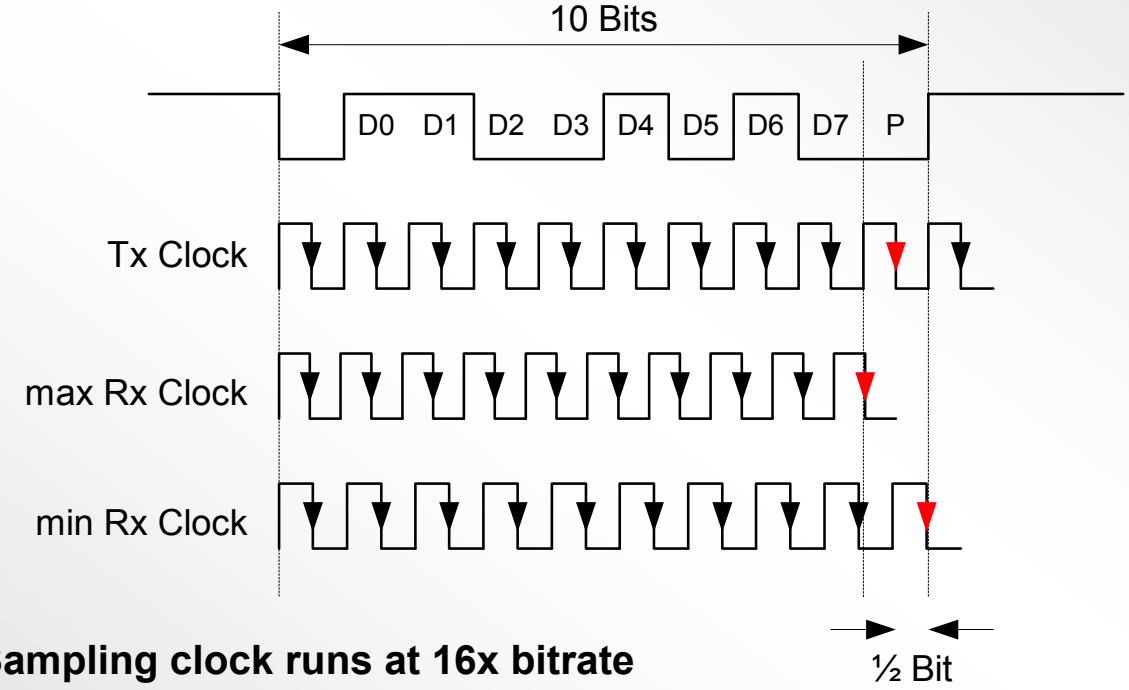
- **Timing Deviation**

10 x 12.288MHz

BAUD RATE	SCLK		
	133 MHz	120 MHz	122.88 MHz
9600	.013 %	.032%	0
19200	.013 %	.096%	0
38400	.022 %	.160%	0
57600	.218 %	.160%	0
115200	.218 %	.160%	0

Receiver Clock Synchronization

- **In theory a delta of up to 5% would not fail, but ...**

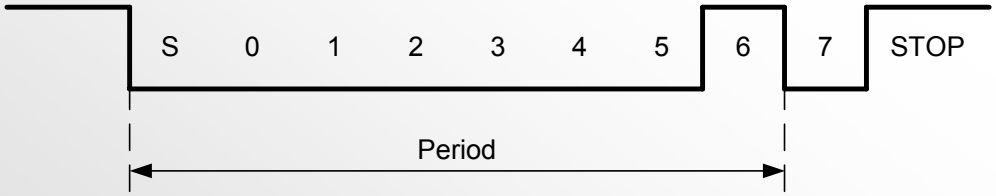


- **Sampling clock runs at 16x bitrate**
- **Receive Filter removes spikes of less than 2x sampling clocks**
- **Also signal rise times and ringing reduces max allowed error rate**

Autobaud Detection

- **Supported by Peripheral Timers**
  - **Set TIN_SEL bit in TIMERx_CONFIG register to sample UART RX pin instead of TMRx pin.**
  - **Use WDTH_CAP mode**
  - **Capture pulse width or periods (recommended)**

Example: `@´ = ASCII 0x40



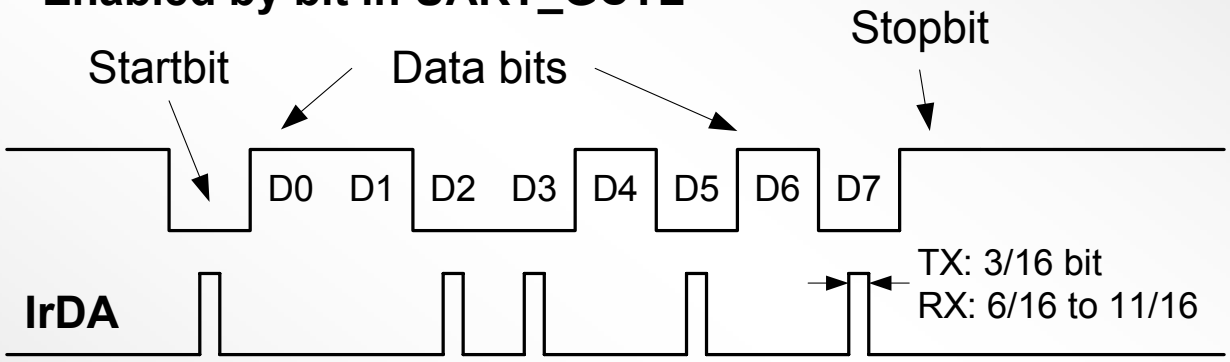
$$Bitrate = \frac{8}{Period} \cdot SCLK$$

$$Bitrate = \frac{SCLK}{16 \times DIVISOR}$$

$$DIVISOR = \frac{PERIOD}{16 \times 8}$$

IrDA Support on UART

- **Meets Half-duplex Infrared Standard IrDA SIR (9.6/115.2 Kbps rate)**
- **Enabled by bit in UART_GCTL**



- **Return-to-zero-inverted Modulation**
- **Off-chip Infrared Tranceiver required**