The processor has two independent, synchronous serial ports, SPORT0 and SPORT1, that provide an I/O interface to peripheral devices.

Each serial port has a set of control registers and data buffers. With a range of clock and frame synchronization options, the SPORTs support a variety of serial communication protocols and provide a glueless hardware interface to industry-standard data converters and CODECs.

The processor’s serial ports provide these features and capabilities:

- Two transmit and two receive channels per serial port.
  
  Each serial port can transmit and receive data simultaneously for full duplex operation.
- Inexpensive eight- or six-line connection to peripheral devices for two-way communication.
- Independent transmit and receive functions.
  
  Independent functioning provides greater flexibility for serial communications.
- Double buffering of data.
- Integral hardware for µ-law and A-law companding.
- Operation at processor’s full clock rate.
  
  This capability provides each with a maximum data rate of \( n \) Mbit/s, where \( n \) equals the processor’s input clock frequency.
• Core controlled interrupt-driven, single-word transfers to and from on-chip memory.

• DMA controller controlled block transfers to and from on-chip memory, including chained DMA operations of multiple data blocks.

• Three operation modes: standard, I^2S, and multichannel.

  In standard mode, one or both transmit channels can transmit, and one or both receive channels can receive.

  In I^2S mode, one or both transmit channels can transmit, and one or both receive channels can receive. Each channel either transmits or receives L and R channels.

  In both standard and I^2S modes, when both A and B channels are used, they transmit or receive data simultaneously, sending or receiving bit 0 on the same edge of the serial clock, bit 1 on the next edge of the serial clock, and so on.

  In multichannel mode, each SPORT can receive and transmit data selectively from channels of a time-division-multiplexed serial bit-stream—a useful option for T1 interfaces.

• Support for internally or externally generated serial clock and frame sync signals in a wide range of frequencies.

• Support for data words of 3- to 32-bits and MSB or LSB formats.
Figure 9-1. Serial port block diagram
Figure 9-1 on page 9-3 shows the architecture of each serial port and Table 9-1 lists and describes the pins.

Table 9-1. Serial port pins

<table>
<thead>
<tr>
<th>Function</th>
<th>SPORT0</th>
<th>SPORT1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Chn Transmitter data</td>
<td>DTOA</td>
<td>DTOB</td>
</tr>
<tr>
<td>B Chn Transmitter data</td>
<td>DT0A</td>
<td>DT0B</td>
</tr>
<tr>
<td>A Chn Transmit clock</td>
<td>TCLK0</td>
<td>TCLK1</td>
</tr>
<tr>
<td>B Chn Transmit frame sync/word select</td>
<td>TFS0</td>
<td>TFS1</td>
</tr>
<tr>
<td>A Chn Receive data</td>
<td>DR0A</td>
<td>DR0B</td>
</tr>
<tr>
<td>B Chn Receive data</td>
<td>DR1A</td>
<td>DR1B</td>
</tr>
<tr>
<td>A Chn Receive clock</td>
<td>RCLK0</td>
<td>RCLK1</td>
</tr>
<tr>
<td>B Chn Receive frame sync</td>
<td>RFS0</td>
<td>RFS1</td>
</tr>
</tbody>
</table>

A serial port receives serial data on its DR input and transmits serial data on its DT output. It can receive and transmit simultaneously for full duplex operation.

The processor always drives, never puts the DT pins in a high impedance state, except when a serial port is in multichannel mode and an inactive time slot occurs.

Serial communications are synchronized to a clock signal—a clock pulse must accompany every data bit. Each serial port can generate or receive its own transmit clock signal (TCLK) and receive clock signal (RCLK). You
configure internally-generated serial clock frequencies in a serial port’s TDIVx and RDIVx registers.

You can use frame synchronization to signal data, signaling either at the beginning of an individual word or at the beginning of a block of words. Configuration of the frame sync signals depends on the type of serial device connected to the processor. Each serial port can generate or receive its own transmit frame sync (TFS) signal and receive frame sync (RFS) signal. You configure internally-generated frame sync frequencies in a serial port’s TDIVx and RDIVx registers.

Figure 9-1 on page 9-3 shows the components of a serial port. The processor’s core writes data for transmission to the TX buffer. Serial port hardware compresses (optional) the data, then automatically transfers it to the transmit shift register. The transmit shift register shifts the data out on the SPORT’s DT pin synchronously to the TCLK transmit clock. When using framing signals, the TFS signal indicates the beginning of the serial word transmission. With serial port enabled (\(SPEN=1\)), the processor always drives the DT pin, unless the channel is operating in multichannel mode and an inactive time slot occurs. (For details, see “Multichannel Mode” on page 9-67.)

Likewise, the receive shift register shifts in data from the SPORT’s DR pin synchronously to the RCLK receive clock. When using framing signals, the RFS signal indicates the beginning of the serial word reception. When the receive shift register shifts in an entire word, serial port hardware expands (optional) the data, then automatically transfers it to the RX buffer.

Because the processor’s SPORTs are not UARTs, they cannot communicate with an RS-232 device or with any other asynchronous communications protocol.
Serial Port Connections

You can, however, implement RS-232-compatible communications with the processor. To do so, use two of the FLAG11-0 pins as asynchronous data receive and transmit signals. For details, see the appropriate chapter in Digital Signal Processing Applications Using The ADSP-2100 Family, Volume 2. Although these examples are 16-bit, fixed-point applications, you can easily modify the code to run on the ADSP-21065L.

SPORT Interrupts

Each serial port has a transmit DMA interrupt and a receive DMA interrupt. With serial port DMA disabled, interrupts occur for each data word the serial port transmits and receives. Table 9-2 shows the priority of the serial port interrupts.

Table 9-2. SPORT interrupts

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Function</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPR0I</td>
<td>SPORT0 receive DMA channels 0 and 1</td>
<td>Highest</td>
</tr>
<tr>
<td>SPR1I</td>
<td>SPORT1 receive DMA channels 2 and 3</td>
<td></td>
</tr>
<tr>
<td>SPT0I</td>
<td>SPORT0 transmit DMA channels 4 and 5</td>
<td></td>
</tr>
<tr>
<td>SPT1I</td>
<td>SPORT1 transmit DMA channels 6 and 7</td>
<td></td>
</tr>
<tr>
<td>EPOI</td>
<td>Ext. port buffer 0 DMA channel 8</td>
<td>Lowest</td>
</tr>
<tr>
<td>EPII</td>
<td>Ext. port buffer 1 DMA channel 9</td>
<td></td>
</tr>
</tbody>
</table>

1 Interrupt names are defined in the def21065L.h include file supplied with the ADSP-21000 Family Development Software.

SPORT interrupts occur on the second system clock (CLKin) after the serial port latches or drives out the last bit of the serial word.
**SPORT Reset**

You can reset the serial ports using either the hardware or the software method. Each method affects the serial ports differently.

Both methods disable the serial ports and clear the data buffer status bits. Re-enabling a serial port does not affect its data buffer status bits. But, regardless of whether a serial port is enabled or disabled, a write or read of its TX or RX buffers changes the corresponding data buffer status bits, incrementing or decrementing them, respectively. This is so, even when you write the RX buffer (increments the RXS status bits) or read the TX buffer (decrements the TXS status bits).

Table 9-3 shows the results of writing and reading full and empty TX and RX data buffers. Some results depend on the value of the BHD bit in the SYSCON register (see page 9-15 and page 9-86).

Table 9-3. Results of TX and RX writes and reads

<table>
<thead>
<tr>
<th>Operation</th>
<th>Full TX</th>
<th>Empty TX</th>
<th>Full RX</th>
<th>Empty RX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>Depends on BHD bit:</td>
<td>Increments status bits</td>
<td>Depends on BHD bit:</td>
<td>Increments status bits</td>
</tr>
<tr>
<td></td>
<td>• Hangs processor</td>
<td></td>
<td>• Hangs processor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Overwrites current contents of TX buffer</td>
<td></td>
<td>• Overwrites current contents of RX buffer</td>
<td></td>
</tr>
</tbody>
</table>
SPORT RESET

Table 9-3. Results of TX and RX writes and reads (Cont’d)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Full TX</th>
<th>Empty TX</th>
<th>Full RX</th>
<th>Empty RX</th>
</tr>
</thead>
</table>
| Read      | Decrements status bits | Depends on BHD bit:  
• Hangs processor  
• Reads invalid data | Decrements status bits | Depends on BHD bit:  
• Hangs processor  
• Reads invalid data |

When re-enabled (in the STCTLx or SRCTLx control register) after reset, a serial port configured for external clock and frame sync can start transmitting or receiving data two CLKIN cycles after becoming enabled.

Using the Hardware Reset Method

To perform a hardware reset, you use the processor’s RESET pin.

A hardware reset clears the STCTLx and SRCTLx control registers (including the SPEN enable bits) and the TDIVx and RDIVx frame sync divisor registers to disable the serial port.

This method aborts any ongoing operations.

Using the Software Reset Method

To perform a software reset, you clear the serial port’s enable bit (SPEN) in the STCTLx and SRCTLx control registers.

A software reset disables the serial port and clears all data buffer status bits.

This method aborts any ongoing operations.
SPORT Control Registers and Data Buffers

Each SPORT has a set of control and configuration registers and data buffers, as shown in Table 9-4. These registers and buffers are part of the IOP register set.

Table 9-4. SPORT control and data registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>STCTLx</td>
<td>SPORT transmit control register</td>
</tr>
<tr>
<td>TXz_1</td>
<td>Transmit data buffer</td>
</tr>
<tr>
<td>TDIVx</td>
<td>Transmit clock and frame sync divisors</td>
</tr>
<tr>
<td>MTCSx</td>
<td>Multichannel transmit select</td>
</tr>
<tr>
<td>MTCCSx</td>
<td>Multichannel transmit compand select</td>
</tr>
<tr>
<td>SRCTLx</td>
<td>SPORT receive control register</td>
</tr>
<tr>
<td>RXz_1</td>
<td>Receive data buffer</td>
</tr>
<tr>
<td>RDIVx</td>
<td>Receive clock and frame sync divisors</td>
</tr>
<tr>
<td>MRCSx</td>
<td>Multichannel receive select</td>
</tr>
<tr>
<td>MRCCSx</td>
<td>Multichannel receive companding select</td>
</tr>
<tr>
<td>KEYWDx</td>
<td>SPORT receive comparison register</td>
</tr>
<tr>
<td>IMASKx</td>
<td>SPORT receive comparison mask</td>
</tr>
</tbody>
</table>

1  x = Serial port 0 or 1; z = Channel A or B
SPORT Control Registers and Data Buffers

Table 9-5 shows the memory-mapped address and reset initialization value of each SPORT register. All of these registers are 32 bits wide.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Reset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STCTL0</td>
<td>0x00E0</td>
<td>0x0000 0000</td>
<td>SPORT0 transmit control reg-</td>
</tr>
<tr>
<td>SRCTL0</td>
<td>0x00E1</td>
<td>0x0000 0000</td>
<td>SPORT0 receive control reg-</td>
</tr>
<tr>
<td>TX0_A</td>
<td>0x00E2</td>
<td>None</td>
<td>SPORT0 transmit data buffer; A data</td>
</tr>
<tr>
<td>RX0_A</td>
<td>0x00E3</td>
<td>None</td>
<td>SPORT0 receive data buffer; A data</td>
</tr>
<tr>
<td>TDIVO</td>
<td>0x00E4</td>
<td>None</td>
<td>SPORT0 transmit divisor</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x00E5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDIVO</td>
<td>0x00E6</td>
<td>None</td>
<td>SPORT0 receive divisor</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x00E7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTCSO</td>
<td>0x00E8</td>
<td>None</td>
<td>SPORT0 multichannel transmit select</td>
</tr>
<tr>
<td>MRCSO</td>
<td>0x00E9</td>
<td>None</td>
<td>SPORT0 multichannel receive select</td>
</tr>
<tr>
<td>MTCCSO</td>
<td>0x00EA</td>
<td>None</td>
<td>SPORT0 multichannel transmit compand select</td>
</tr>
<tr>
<td>MRCCSO</td>
<td>0x00EB</td>
<td>None</td>
<td>SPORT0 multichannel receive compand select</td>
</tr>
<tr>
<td>KEYWDO</td>
<td>0x00EC</td>
<td>None</td>
<td>SPORT0 receive comparison register</td>
</tr>
</tbody>
</table>
Table 9-5. SPORT registers memory-mapped addresses and reset values

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Reset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMASK0</td>
<td>0x00ED</td>
<td>None</td>
<td>SPORT0 receive comparison mask register</td>
</tr>
<tr>
<td>TX0_B</td>
<td>0x00EE</td>
<td>None</td>
<td>SPORT0 transmit data buffer: B data</td>
</tr>
<tr>
<td>RX0_B</td>
<td>0x00EF</td>
<td>None</td>
<td>SPORT0 receive data buffer: B data</td>
</tr>
<tr>
<td>STCTL1</td>
<td>0x00F0</td>
<td>0x0000 0000</td>
<td>SPORT1 transmit control register</td>
</tr>
<tr>
<td>SRCTL1</td>
<td>0x00F1</td>
<td>0x0000 0000</td>
<td>SPORT1 receive control register</td>
</tr>
<tr>
<td>TX1_A</td>
<td>0x00F2</td>
<td>None</td>
<td>SPORT1 transmit data buffer: A data</td>
</tr>
<tr>
<td>RX1_A</td>
<td>0x00F3</td>
<td>None</td>
<td>SPORT1 receive data buffer: A data</td>
</tr>
<tr>
<td>TDIV1</td>
<td>0x00F4</td>
<td>None</td>
<td>SPORT1 transmit divisor</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x00F5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>0x00F6</td>
<td>None</td>
<td>SPORT1 receive divisor</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x00F7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTCS1</td>
<td>0x00F8</td>
<td>None</td>
<td>SPORT1 multichannel transmit select</td>
</tr>
<tr>
<td>MRCS1</td>
<td>0x00F9</td>
<td>None</td>
<td>SPORT1 multichannel receive select</td>
</tr>
<tr>
<td>MTCCS1</td>
<td>0x00FA</td>
<td>None</td>
<td>SPORT1 multichannel transmit compand select</td>
</tr>
</tbody>
</table>
SPORT Control Registers and Data Buffers

Table 9-5. SPORT registers memory-mapped addresses and reset values

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Reset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRCCS1</td>
<td>0x00FB</td>
<td>None</td>
<td>SPORT1 multichannel receive compand select</td>
</tr>
<tr>
<td>KEYWD1</td>
<td>0x00FC</td>
<td>None</td>
<td>SPORT1 receive comparison register</td>
</tr>
<tr>
<td>IMASK1</td>
<td>0x00FD</td>
<td>None</td>
<td>SPORT1 receive comparison mask register</td>
</tr>
<tr>
<td>TX1_B</td>
<td>0x00FE</td>
<td>None</td>
<td>SPORT1 transmit data buffer; B data</td>
</tr>
<tr>
<td>RX1_B</td>
<td>0x00FF</td>
<td>None</td>
<td>SPORT1 receive data buffer; B data</td>
</tr>
</tbody>
</table>

To program the SPORT control registers, you write to the appropriate address in memory. Applications can use the symbolic names of the registers and individual control bits. The file `def21065L.h`, provided in the INCLUDE directory of the ADSP-21000 Family Development Software, contains the #define definitions for these symbols. See Appendix E, Control and Status Registers, in ADSP-21065L SHARC Technical Reference, for a listing of the file's contents.

All control and status bits in the SPORT registers are active high unless otherwise noted.

Because the SPORT registers are memory-mapped, you cannot write them with data coming directly from memory. Instead, you must write or read them from or to the processor's core registers, usually one of the Register File's general-purpose universal registers (R15–R0).

External devices, such as another ADSP-21065L or a host, can write and read the SPORT control registers to set up a serial port DMA operation, for example.
When changing operating modes, write the serial port’s control register, STCTLx or SRCTLx, with all 0s to clear it before you write the new mode to the register.

**Register Writes and Effect Latency**

The processor completes internal writes to SPORT registers at the end of the same CLKIN cycle in which they begin. So the newly written value is available in the register on the next cycle. But when a write to one of the STCTLx or SRCTLx control registers immediately follows a read of the same register, the write takes at least two cycles to finish.

After a write to a SPORT register, control and mode bit changes take effect by the end of the second CLKIN cycle after the write has finished. Two CLKIN cycles after they are enabled (in the STCTLx or SRCTLx register), the serial ports can start transmitting or receiving, losing no serial clock cycles from that point on.

**Transmit and Receive Data Buffers (TX, RX)**

TX0_A and TX0_B are the transmit data buffers for SPORT0, and TX1_A and TX1_B are the transmit data buffers for SPORT1. Either the DMA controller or the processor’s core program must load these 32-bit buffers with the data to transmit.

RX0_A and RX0_B are the receive data buffers for SPORT0, and RX1_A and RX1_B are the receive data buffers for SPORT1. The receive shift register automatically loads these 32-bit buffers when the serial port has received an entire word. The receive and transmit buffers right-justify words containing less than thirty-two bits.

**TX Buffer Operation**

Because they have a data register and an output shift register, the TX buffers behave like two-location FIFOs (see Figure 9-1 on page 9-3).
SPORT Control Registers and Data Buffers

You can store only two 32-bit words in a TX buffer at a time. When the TX buffer is loaded and the serial port has transmitted the previous word, the TX buffer automatically loads its contents into the transmit shift register. This transfer generates an interrupt, signaling that the TX buffer is not full and ready to accept the next word. When serial port DMA is enabled or the corresponding mask bit in the IMASK register is set, this interrupt does not occur.

When a transmit frame synch occurs and the TX buffer contains no new data, the processor sets the transmit underflow status bit (TUVF) in the transmit control register. The TUVF status bit is sticky (the application must explicitly clear the bit), and you must disable the serial port to clear it.

RX Buffer Operation

Because they have two data register and an input shift register, the RX buffers behave like three-location FIFOs (see Figure 9-1 on page 9-3).

You can store two 32-bit words in an RX buffer while the receive shift register is shifting in a third word. The third word overwrites the second if the processor’s core or the DMA controller has not read the first word. When this occurs, the processor sets the receive overflow status bit (ROVF) in the receive control register. The RX buffer can receive almost three entire words without an internal read before overflow occurs. The processor generates the overflow status on the last bit of third word. The ROVF status bit is sticky, and you must disable the serial port to clear it.

When the RX buffer has received a word (the buffer is not empty), it generates an interrupt. When serial port DMA is enabled or the corresponding bit in the IMASK register is set, the processor masks this interrupt.

Reading and Writing RX, TX

If the processor’s core attempts to read from an empty RX buffer or to write to a full TX buffer, the processor delays the access until the external...
I/O device accesses the buffer. This delay is called a core processor hang. To avoid hanging the processor’s core, read the buffer’s full or empty status (in STCTLx or SRCTLx) before accessing a TX or RX buffer. To prevent this type of hang condition globally, set the BHD (Buffer Hang Disable) bit in the SYSCON register (see Table 9-3 on page 9-7).

The processor updates the status bits in STCTLx and SRCTLx during core reads and writes, even when the serial port is disabled. For details, see page 9-7.

Make sure your application disables a serial port when it writes to the serial port’s RX buffer or reads from the serial port’s TX buffer; for example, if it tests the results of companding.

**Transmit and Receive Control Registers (STCTL, SRCTL)**

The main control registers for each serial port are the transmit control register, STCTLx, and the receive control register, SRCTLx. See Table 9-6 and Table 9-7 on page 9-21 for the bit definitions of these registers. For default bit values, see Figure 9-2 on page 9-18, Figure 9-3 on page 9-19, Figure 9-4 on page 9-20, Figure 9-5 on page 9-23, Figure 9-6 on page 9-24, and Figure 9-7 on page 9-25. Some bit definitions depend on the mode of operation for which the serial port is configured.

Table 9-6. STCTLx transmit control bits

<table>
<thead>
<tr>
<th>Bit</th>
<th>I²S Mode</th>
<th>Standard Mode</th>
<th>Multichannel Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SPEN_A</td>
<td>SPEN_A</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
<td>DTYPE</td>
<td>DTYPE</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
<td>DTYPE</td>
<td>DTYPE</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
<td>SENDN</td>
<td>SENDN</td>
</tr>
</tbody>
</table>
## SPORT Control Registers and Data Buffers

Table 9-6. STCTLx transmit control bits (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>( I^2S ) Mode</th>
<th>Standard Mode</th>
<th>Multichannel Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>SLEN(_0)</td>
<td>SLEN(_0)</td>
<td>SLEN(_0)</td>
</tr>
<tr>
<td>5</td>
<td>SLEN(_1)</td>
<td>SLEN(_1)</td>
<td>SLEN(_1)</td>
</tr>
<tr>
<td>6</td>
<td>SLEN(_2)</td>
<td>SLEN(_2)</td>
<td>SLEN(_2)</td>
</tr>
<tr>
<td>7</td>
<td>SLEN(_3)</td>
<td>SLEN(_3)</td>
<td>SLEN(_3)</td>
</tr>
<tr>
<td>8</td>
<td>SLEN(_4)</td>
<td>SLEN(_4)</td>
<td>SLEN(_4)</td>
</tr>
<tr>
<td>9</td>
<td>PACK</td>
<td>PACK</td>
<td>PACK</td>
</tr>
<tr>
<td>10</td>
<td>MSTR</td>
<td>ICLK</td>
<td>Reserved</td>
</tr>
<tr>
<td>11</td>
<td>OPMODE</td>
<td>OPMODE</td>
<td>OPMODE</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
<td>CKRE</td>
<td>CKRE</td>
</tr>
<tr>
<td>13</td>
<td>Reserved</td>
<td>TFSR</td>
<td>Reserved</td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
<td>ITFS</td>
<td>Reserved</td>
</tr>
<tr>
<td>15</td>
<td>DITFS</td>
<td>DITFS</td>
<td>DITFS</td>
</tr>
<tr>
<td>16</td>
<td>L(__)FIRST</td>
<td>LTFS</td>
<td>LTFS</td>
</tr>
<tr>
<td>17</td>
<td>Reserved</td>
<td>LAFS</td>
<td>Reserved</td>
</tr>
<tr>
<td>18</td>
<td>SDEN(_A)</td>
<td>SDEN(_A)</td>
<td>SDEN(_A)</td>
</tr>
<tr>
<td>19</td>
<td>SCHEN(_A)</td>
<td>SCHEN(_A)</td>
<td>SCHEN(_A)</td>
</tr>
<tr>
<td>20</td>
<td>SDEN(_B)</td>
<td>SDEN(_B)</td>
<td>MFD</td>
</tr>
<tr>
<td>21</td>
<td>SCHEN(_B)</td>
<td>SCHEN(_B)</td>
<td>MFD</td>
</tr>
</tbody>
</table>
## Table 9-6. STCTLx transmit control bits (Cont'd)

<table>
<thead>
<tr>
<th>Bit</th>
<th>I²S Mode</th>
<th>Standard Mode</th>
<th>Multichannel Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>FS_BOTH</td>
<td>FS_BOTH</td>
<td>MFD</td>
</tr>
<tr>
<td>23</td>
<td>Reserved</td>
<td>Reserved</td>
<td>MFD</td>
</tr>
<tr>
<td>24</td>
<td>SPEN_B</td>
<td>SPEN_B</td>
<td>CHNL</td>
</tr>
<tr>
<td>25</td>
<td>Reserved</td>
<td>Reserved</td>
<td>CHNL</td>
</tr>
<tr>
<td>26</td>
<td>TUVF_B</td>
<td>TUUF_B</td>
<td>CHNL</td>
</tr>
<tr>
<td>27</td>
<td>TXS_B</td>
<td>TXS_B</td>
<td>CHNL</td>
</tr>
<tr>
<td>28</td>
<td>TXS_B</td>
<td>TXS_B</td>
<td>CHNL</td>
</tr>
<tr>
<td>29</td>
<td>TUUF_A</td>
<td>TUUF_A</td>
<td>TUUF_A</td>
</tr>
<tr>
<td>30</td>
<td>TXS_A</td>
<td>TXS_A</td>
<td>TXS_A</td>
</tr>
<tr>
<td>31</td>
<td>TXS_A</td>
<td>TXS_A</td>
<td>TXS_A</td>
</tr>
</tbody>
</table>
SPORT Control Registers and Data Buffers

Figure 9-2. STCTLx transmit control register—Standard mode
Figure 9-3. STCTLx transmit control register—$I^2S$ mode
Figure 9-4. STCTLx transmit control register—multichannel mode
Table 9-7. SRCTLx transmit control bits

<table>
<thead>
<tr>
<th>Bit</th>
<th>I²S Mode</th>
<th>Standard Mode</th>
<th>Multichannel Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SPEN_A</td>
<td>SPEN_A</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
<td>DTYPE</td>
<td>DTYPE</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
<td>DTYPE</td>
<td>DTYPE</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
<td>SENDN</td>
<td>SENDN</td>
</tr>
<tr>
<td>4</td>
<td>SLEN0</td>
<td>SLEN0</td>
<td>SLEN0</td>
</tr>
<tr>
<td>5</td>
<td>SLEN1</td>
<td>SLEN1</td>
<td>SLEN1</td>
</tr>
<tr>
<td>6</td>
<td>SLEN2</td>
<td>SLEN2</td>
<td>SLEN2</td>
</tr>
<tr>
<td>7</td>
<td>SLEN3</td>
<td>SLEN3</td>
<td>SLEN3</td>
</tr>
<tr>
<td>8</td>
<td>SLEN4</td>
<td>SLEN4</td>
<td>SLEN4</td>
</tr>
<tr>
<td>9</td>
<td>PACK</td>
<td>PACK</td>
<td>PACK</td>
</tr>
<tr>
<td>10</td>
<td>MSTR</td>
<td>ICLK</td>
<td>ICLK</td>
</tr>
<tr>
<td>11</td>
<td>OPMODE</td>
<td>OPMODE</td>
<td>OPMODE</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
<td>CKRE</td>
<td>CKRE</td>
</tr>
<tr>
<td>13</td>
<td>Reserved</td>
<td>RFSR</td>
<td>Reserved</td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
<td>IRFS</td>
<td>IRFS</td>
</tr>
<tr>
<td>15</td>
<td>Reserved</td>
<td>Reserved</td>
<td>IMODE</td>
</tr>
<tr>
<td>16</td>
<td>L_FIRST</td>
<td>LRFS</td>
<td>LRFS</td>
</tr>
</tbody>
</table>
### SPORT Control Registers and Data Buffers

Table 9-7. SRCTLx transmit control bits (Cont’d)

<table>
<thead>
<tr>
<th>Bit</th>
<th>I2S Mode</th>
<th>Standard Mode</th>
<th>Multichannel Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Reserved</td>
<td>LAFS</td>
<td>Reserved</td>
</tr>
<tr>
<td>18</td>
<td>SDEN_A</td>
<td>SDEN_A</td>
<td>SDEN_A</td>
</tr>
<tr>
<td>19</td>
<td>SCHEN_A</td>
<td>SCHEN_A</td>
<td>SCHEN_A</td>
</tr>
<tr>
<td>20</td>
<td>SDEN_B</td>
<td>SDEN_B</td>
<td>IMAT</td>
</tr>
<tr>
<td>21</td>
<td>SCHEN_B</td>
<td>SCHEN_B</td>
<td>Reserved</td>
</tr>
<tr>
<td>22</td>
<td>SPL</td>
<td>SPL</td>
<td>Reserved</td>
</tr>
<tr>
<td>23</td>
<td>Reserved</td>
<td>MCE</td>
<td>MCE</td>
</tr>
<tr>
<td>24</td>
<td>SPEN_B</td>
<td>SPEN_B</td>
<td>NCH</td>
</tr>
<tr>
<td>25</td>
<td>Reserved</td>
<td>Reserved</td>
<td>NCH</td>
</tr>
<tr>
<td>26</td>
<td>ROVF_B</td>
<td>ROVF_B</td>
<td>NCH</td>
</tr>
<tr>
<td>27</td>
<td>RXS_B</td>
<td>RXS_B</td>
<td>NCH</td>
</tr>
<tr>
<td>28</td>
<td>RXS_B</td>
<td>RXS_B</td>
<td>NCH</td>
</tr>
<tr>
<td>29</td>
<td>ROVF_A</td>
<td>ROVF_A</td>
<td>ROVF_A</td>
</tr>
<tr>
<td>30</td>
<td>RXS_A</td>
<td>RXS_A</td>
<td>RXS_A</td>
</tr>
<tr>
<td>31</td>
<td>RXS_A</td>
<td>RXS_A</td>
<td>RXS_A</td>
</tr>
</tbody>
</table>
Figure 9-5. SRCTLx receive control registers—Standard mode
SPORT Control Registers and Data Buffers

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Status *</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>RX A Data Buffer</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>OOMODE Operation Mode</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>MSTR Master/Slave mode</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>PACK 16/32-bit packing</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>SPL SPORT Loopback</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>ROVF A RX A Overflow</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>SDEN A SPORT Rcv DMA chaining enable A</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>L_FIRST Rcv left chn. first</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>RXS A Status *</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>RXS A RX A Data Buffer</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>SLEN Serial Word Length -1</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>SPEN A SPORT Enable A</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>RXS B Status *</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>L_FIRST Rcv left chn. first</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>ROVF B RX A Overflow</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>SLEN Serial Word Length -1</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>SPEN B SPORT Enable B</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>SLEN Serial Word Length -1</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>MSTR Master/Slave mode</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>PACK 16/32-bit packing</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>OOMODE Operation Mode</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>RXS A Status *</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>OOMODE Operation Mode</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>RXS B Status *</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>L_FIRST Rcv left chn. first</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>ROVF B RX A Overflow</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>SLEN Serial Word Length -1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>SPEN B SPORT Enable B</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>RXS A RX A Data Buffer</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>SLEN Serial Word Length -1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>OOMODE Operation Mode</td>
<td>0</td>
</tr>
</tbody>
</table>

* Status is read-only

Figure 9-6. SRCTLx receive control registers—I^2S mode
### Serial Ports

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RXS_A (Status *)</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>RX A Data Buffer</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>RX A Underflow</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>ROVF_A (Status *) (sticky)</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>RX A Underflow</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>NCH</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>MCE</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>IMAT</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>OPMODE</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>PACK</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>CKRE</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>ICLK</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>SLEN</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>DATA_TYPE</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>SENDN</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>LRFS</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>SDEN_A</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>SCHEN_A</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>NCH</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>PACK</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>CKRE</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>ICLK</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>SLEN</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>DATA_TYPE</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>SENDN</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>LRFS</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>SDEN_A</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>SCHEN_A</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>NCH</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>PACK</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>CKRE</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>ICLK</td>
<td>0</td>
</tr>
</tbody>
</table>

* Status is read-only

---

**Figure 9-7. SRCTLx receive control registers—multichannel mode**
SPORT Control Registers and Data Buffers

Bit definitions of the STCTLx and SRCTLx control register parameters are:

**CHNL**
Current channel selected.
Multichannel mode only. STCTLx register.
Read-only, sticky status bits.
Identifies the currently selected transmit channel slot (0 to 31).

**CKRE**
Frame sync clock edge.
Standard and multichannel modes only. STCTLx and SRCTLx registers.
Selects the active edge of the serial port clock on which to sample or drive data and frame syncs.
In standard mode only, you can set this parameter separately for transmit and receive channels.
0 = Falling edge
1 = Rising edge
(Frame sync is level-sensitive, not edge-sensitive.)

**DITFS**
Data independent TFS.
All operation modes. STCTLx register.
Selects when the processor generates the transmit frame sync signal.
Serial Ports

0 = Data dependent TFS.

TFS signal generated only when new data is in SPORT channel’s transmit data buffer. Applications must also program the TDIV register.

1 = Data independent TFS.

TFS signal generated regardless of the validity of the data present in SPORT channel’s transmit data buffer. The processor generates the TFS signal at the frequency specified by the value you load in the TDIV register.

DTYPE
Data type.

Standard and multichannel modes only. STCTLx and SRCTLx registers.

Selects the companding and MSB format of serial words loaded into the TX and RX buffers. (The transmit shift register does not 0-fill or sign-extend TX data words.)

Selection differs between modes.

For standard mode, selection of companding mode and MSB format are exclusive:

00 = Right justify; fill unused MSBs with 0s.
01 = Right justify; sign-extend into unused MSBs.
10 = Compand using μ-law. (Primary channels only)
11 = Compand using A-law. (Primary channels only)

For multichannel mode, selection of companding mode and MSB format are independent:
**SPORT Control Registers and Data Buffers**

- **x0**  = Right justify; fill unused MSBs with 0s.
- **x1**  = Right justify; sign-extend into unused MSBs.
- **0x**  = Compand using $\mu$-law.
- **1x**  = Compand using $A$-law.

**FS_BOTH**
Frame sync both.

- $I^2S$ and standard modes only. STCTLx register.
Selects when during transmission to issue the word select.
- **0**  = Issue word select if data in either transmit channel.
- **1**  = Issue word select only if data in both transmit channels.

**ICLK**  Transmit and receive clock sources.

Standard and multichannel modes only. STCTLx and SRCTLx registers.
Selects the clock source to use to transmit and to receive data. In standard mode only, you can set this parameter separately for transmit and receive channels.
- **0**  = Use an external clock.
- **1**  = Use processor's internal clock.

**IMAT**  Receive comparison accept data.

Multichannel mode only. SRCTLx register.
Selects the method to use for evaluating whether to accept received data.
Serial Ports

0 = Accept the received data if the KEYWD compares false.
1 = Accept the received data if the KEYWD compares true.

**IMODE**
Receive comparison enable.
Multichannel mode only. SRCTLx register.
Enables and disables the receive comparison option.
0 = Disable receive comparison.
1 = Enable receive comparison.

**IRFS** RFS source.
Standard and multichannel modes only. SRCTLx register.
Selects the source to generate frame sync signals for received data.
0 = Use external source.
1 = Use processor’s internal serial clock.

**ITFS** TFS source.
Standard mode only. STCTLx register.
Selects the source to generate frame sync signals for transmit data.
0 = Use external source.
1 = Use processor’s internal serial clock.

**LAFS** Late TFS/RFS.
Standard mode only. STCTLx and SRCTLx registers.
SPORT Control Registers and Data Buffers

Selects when to generate the receive frame sync signal.

0 = Generate early, during the serial clock cycle immediately preceding the first data bit.
1 = Generate late, during the first bit of each data word.

L_FIRST
Left/right channel transmit/receive first.

I^2S mode only. STCTLx and SRCTLx registers.

Selects which I^2S channel to transmit or receive first.

0 = Right channel first.
1 = Left channel first.

LRFS Active state RFS.

Standard and multichannel modes only. SRCTLx register.

Selects the logic level of the received frame sync signals. Active high (0) is the default.

0 = Active high.
1 = Active low (inverted).

LTFS Active state TFS.

Standard and multichannel modes only. STCTLx register.

Selects the logic level of the transmit frame sync signals. Active high (0) is the default.
Serial Ports

0 = Active high.
1 = Active low (inverted).

MCE Multichannel mode enable.
Standard and multichannel modes only. SRCTLx register.
One of two configuration bits that enable and disable multichannel mode on receive serial port channels. See also, OPMODE.
0 = Disable multichannel operation.
1 = Enable multichannel operation if \( OP\text{MODE} = 0 \).

MFD Multichannel frame delay.
Multichannel mode only. STCTLx register.
Sets the interval, in number of serial clock cycles, between the transmit frame sync pulse and the first data bit. Provides support for different types of T1 interface devices.
Valid values range from 0 to 15.
0 = No delay; frame sync pulse concurrent with first data bit.
1:15 = Corresponding number of intervening serial clock cycles.

MSTR
SPORT transmit and receive master mode.
\(^3\)S mode only. STCTLx and SRCTLx registers.
Selects the clock and word-select source for transmitting or for receiving.
SPORT Control Registers and Data Buffers

0 = Use external clock and word-select source; transmitter or receiver is slave.

1 = Use internal clock and word-select source; transmitter or receiver is master.

**NCH** Number of channel slots.

Multichannel mode only. SRCTLx register.

Selects the number of channel slots (maximum of 32) to use for multichannel operation.

Use this formula to calculate the value for NCH:

\[
NCH = \text{Actual number of channel slots} - 1.
\]

Valid values for actual number of channel slots range from 1 to 32.

**OMODE**

SPORT operation mode.

All operation modes. STCTLx and SRCTLx registers.

Enables and disables I^2^S operation mode. When this bit is set, the processor ignores the MCE bit.

0 = Disable I^2^S mode.

Depending on the MCE bit, sets the channel in either standard mode or multichannel mode.

1 = Enable I^2^S mode.

**PACK** Packing 16/32 bit.

All operation modes. STCTLx and SRCTLx registers.
Serial Ports

Selects whether the serial port packs external words of 16 bits or less into internal 32-bit words and vice versa.

0 = Disable packing.
1 = Enable packing.

**RFSR** RFS requirement.

Standard mode only. SRCTLx register.

Selects whether receive serial port communications require frame sync signals.

0 = Not required.

1 = Every data word requires a frame sync signal.

(Only a single frame sync signal required to initiate communications; ignored after first bit received.)

**ROVF** Receive overflow status.

All operation modes. SRCTLx register.

Read-only, sticky status bit.

Indicates when the channel has received new data while the RXS buffer is full. New data overwrites existing data.

0 = No new data.
1 = New data.

**RXS** Receive data buffer status.

All operation modes. SRCTLx register.
SPORT Control Registers and Data Buffers

Read-only, sticky status bit.
Indicates the status of the channel’s receive buffer contents.
00 = Buffer empty.
01 = Reserved.
10 = Buffer partially full.
11 = Buffer full.

SCHEN
SPORT DMA chaining.
All operation modes for primary (A) SPORT channels. I2S and standard modes only for secondary (B) SPORT channels. STCTLx and SRCTLx registers.
Enables and disables SPORT DMA chaining.
0 = Disable DMA chaining.
1 = Enable DMA chaining.

SDEN
SPORT DMA enable.
All operation modes for primary (A) SPORT channels. I2S and standard modes only for secondary (B) SPORT channels. STCTLx and SRCTLx registers.
Enables and disables SPORT DMA.
0 = Disable DMA.
1 = Enable DMA.
**SENDN**

Endian data word format.

Standard and multichannel modes only. STCTLx and SRCTLx registers.

Selects whether the serial word is transmitted or received MSB or LSB first.

0 = MSB first.

1 = LSB first.

**SLEN** Serial word length.

All operation modes. STCTLx and SRCTLx registers.

Selects the number of bits the serial word contains. The SPORTs handle serial words containing from 3 to 32 bits.

Use this formula to calculate the value for SLEN:

\[
SLEN = \text{Actual serial word length} - 1
\]

\[ SLEN \neq 0 \text{ or } 1 \]

**SPEN** SPORT enable.

I²S and standard modes only. STCTLx and SRCTLx registers.

Enables and disables the SPORT. Performs a software reset.
SPORT Control Registers and Data Buffers

0 = Disable SPORT.
    Aborts any ongoing operation and clears the status bits.
1 = Enable SPORT.
    SPORTS ready to transmit or receive two cycles after enabling.

**SPL**  SPORT loopback mode.

I²S and standard modes only. SRCTLx register.

Sets the channel in or out of loopback mode. Loopback mode enables developers to run internal tests and to debug applications.

0 = Disable loopback mode.
1 = Enable loopback mode.

**TFSR**  Transmit frame sync requirement.

Standard mode only. STCTLx register.

Selects whether transmit serial port communications require frame sync signals.

0 = Not required.
    (Only a single frame sync signal required to initiate communications; ignored after first bit transmitted.)
1 = Every data word requires a frame sync signal.

**TUVF**  Transmit underflow status.

All operation modes. STCTLx register.

Read-only, sticky status bit.
Indicates whether the TFS signal (from internal or external source) occurred while the TXS buffer was empty. The SPORTs transmit data whenever they detect a TFS signal.

0 = No TFS signal occurred.
1 = TFS signal occurred.

**TXS**  Transmit data buffer status.

All operation modes. STCTLx register.

Read-only, sticky status bit.

Indicates the status of the channel’s transmit buffer contents.

00 = Buffer empty.
01 = Reserved.
10 = Buffer partially full.
11 = Buffer full.

---

Hereafter in this chapter, unless referring to a specific case, registers and control parameters are referred to by the descriptive part of their symbolic names only or with x or _z included to indicate serial port and/or channel specification, respectively. (For example, SRCTLx, SPEN, or SCHEN_Z.)

However to use the symbolic names in your application, you must write the correct symbolic name in its entirety. For example, SPEN_A or SPEN_B, not SPEN or SPEN_Z; STCTL1 or STCTL0, not STCTLx or STCTI.
SPORT Control Registers and Data Buffers

Control Register Status Bits

The STCTLx and SRCTLx status bits are read-only, sticky bits that provide information about the status of a particular SPORT channel.

The STCTLx and SRCTLx status bits are:

- **CHNL** Current Channel Selected status bits
- **ROVF** Receive Overflow status bit
- **RXS** Receive Data Buffer status bits
- **TUVF** Transmit Underflow status bit
- **TXS** Transmit Data Buffer status bits

Current Channel Selected Status Bits (CHNL)

During multichannel operation, the CHNL status bits indicate which of the thirty-two channel slots (CHNL_{31:0}) the serial port is currently selected.

Receive Overflow Status Bit (ROVF)

The processor sets the ROVF bit whenever the serial port receives new data while the RX buffer is full. In this case, the new data overwrites the existing data.

Receive Data Buffer Status Bits (RXS)

The RXS status bits indicate whether the RX buffer is full (11), empty (00), or partially full (10).

You can test the RXS status bits to determine if the RX data buffer has free space or if it contains data. To test for space, test for RXS_0=0. To test for data, test for RXS_1=1.
Serial Ports

Transmit Underflow Status Bit (TUVF)

The processor sets the TUVF bit whenever the TFS signal occurs (generated either internally or by an external source) while the TX buffer is empty.

You can suppress this behavior when using internally generated TFS. To do so, you clear the DITFS control bit (DITFS=0). Setting DITFS to 0 selects data-dependent frame syncs. In this mode, the processor generates the transmit frame sync signal (TFS) only when the TX buffer contains new data, so the serial port transmits new data only.

Setting DITFS to 1 selects data-independent frame syncs. In this mode, the processor generates the TFS signal whether or not the TX buffer contains new data, and the serial port transmits the contents of the TX buffer regardless. Typically, serial port DMA keeps the TX buffer full, and when the DMA operation finishes, the serial port continuously transmits the last word in the TX buffer.

Transmit Data Buffer Status Bits (TXS)

The TXS status bits indicate whether the TX data buffer is full (11), empty (00), or partially full (10).

You can test the TXS status bits to determine if the TX data buffer has free space or if it contains data. To test for space, test for TXS0=0. To test for data, test for TXS1=1.

Clock and Frame Sync Frequencies (TDIV, RDIV)

The TDIV and RDIV registers contain divisor values, which determine the frequencies at which internally generated clocks and frame syncs operate.
SPORT Control Registers and Data Buffers

Figure 9-8 shows and Table 9-8 lists and defines the contents of the TDIV0 and TDIV1 registers.

Table 9-8. Transmit divisor register bit fields

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-0</td>
<td>TCLKDIV</td>
<td>Transmit clock divisor</td>
</tr>
<tr>
<td>31-16</td>
<td>TFSDIV</td>
<td>Transmit frame sync divisor</td>
</tr>
</tbody>
</table>

Figure 9-8. TDIVx transmit divisor registers
Figure 9-9 shows and Table 9-9 lists and defines the contents of the RDIV0 and RDIV1 registers.

Table 9-9. Receive divisor register bit fields

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-0</td>
<td>RCLKDIV</td>
<td>Receive clock divisor</td>
</tr>
<tr>
<td>31-16</td>
<td>RFSDIV</td>
<td>Receive frame sync divisor</td>
</tr>
</tbody>
</table>

The TCLKDIV and RCLKDIV bit fields specify the number of times to divide the processor’s system clock (CLKIN) to generate the transmit and receive clocks. The divisor is a 16-bit value, which provides a wide range of serial clock rates.
SPORT Control Registers and Data Buffers

Use this equation to calculate the serial clock frequency:

\[
\text{serial clock frequency} = \frac{2 \times f_{\text{CLKIN}}}{(x_{\text{CLKDIV}} + 1)}
\]

\(f_{\text{CLKIN}}\) is the 1x frequency for the processor, and \(x_{\text{CLKDIV}}\) is at least equal to 1.

Use this equation to calculate the value of \(x_{\text{CLKDIV}}\), given the \(f_{\text{CLKIN}}\) frequency and target serial clock frequency:

\[
x_{\text{CLKDIV}} = \frac{2 \times f_{\text{CLKIN}}}{\text{serial clock frequency}} - 1
\]

When frame sync is internally generated, \(T_{\text{FSDIV}}\) and \(R_{\text{FSDIV}}\) specify the number of transmit or receive clock cycles the processor counts before it generates a TFS or RFS pulse. You can use a frame sync this way to initiate periodic transfers. The processor counts serial clock cycles whatever the clock source, internal or external.

Use this equation to calculate the number of serial clock cycles between frame sync pulses:

\[
\text{No. cycles between frame sync assertions} = x_{\text{FSDIV}} + 1
\]

Use this equation to determine the value of \(x_{\text{FSDIV}}\), given the serial clock frequency and target frame sync frequency:

\[
x_{\text{FSDIV}} = \frac{\text{serial clock frequency}}{\text{frame sync frequency}} - 1
\]
Serial Ports

The frame sync is continuously active if xFSDIV=0. However, to avoid causing an external device to abort the current operation or causing other unpredictable results, use a value for xFSDIV such that

\[ \text{FSDIV} \geq \text{SLEN} - 1 \]

(Use the value of the SLEN field in the transmit or receive control register.)

If not using the serial port, you can use the xFSDIV divisor as a counter for dividing an external clock or for generating a periodic pulse or periodic interrupt. For this function, the serial port must be enabled.

Restrictions on Using Maximum Clock Rate

A delay occurs between the arrival of the transmit clock signal at the TCLKx pin and the output of serial data. This delay may limit the operating speed of the receiver. For exact timing specifications, see the data sheet.

For reliable operation, we recommend that you use full-speed, serial clocks only when receiving with an externally generated clock and externally generated frame sync (ICLK=0, IRFS=0).

Externally-generated, late transmit frame syncs (LAFS) experience a similar delay between their arrival and data output, which can also limit the maximum speed of serial clocks. For exact timing specifications, see the data sheet.

Although the serial ports handle words with lengths of three to thirty-two bits, transmitting or receiving words smaller than four bits at the processor's full serial clock rate may cause loss of data when DMA chaining is enabled. Chaining takes over the processor's internal I/O bus for several cycles while the DMA controller loads new TCB parameters. During this period, receive data in the RX buffer may be overwritten.
Data Word Formats

The DTYPE, PACK, SENDN, and SLEN bits of the STCTLx and SRCTLx control registers format data words transmitted through the serial ports.

Data Type (DTYPE)

The DTYPE field of the STCTLx and SRCTLx control registers, shown in Table 9-10, specifies the justification format and the companding format of the data when the serial port is configured for standard or multichannel operation.

For standard operation, the DTYPE field specifies one of four data formats. Data justification and companding formats are separate and exclusive options.

Table 9-10. Data formats for nonmultichannel operation

<table>
<thead>
<tr>
<th>DTYPE</th>
<th>Data Formatting</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Right justify; fill unused MSBs with zeros (0)</td>
</tr>
<tr>
<td>01</td>
<td>Right justify; extend sign into unused MSBs</td>
</tr>
<tr>
<td>10</td>
<td>Compand using µ-law</td>
</tr>
<tr>
<td>11</td>
<td>Compand using A-law</td>
</tr>
</tbody>
</table>

The RX and TX shifter registers apply these formats to serial data words when they are loaded into the RX and TX buffers. (Since only the significant bits of the serial data word are transmitted, the TX shift register does not actually zero-fill or sign-extend TX data words.)

For multichannel operation, the DTYPE field specifies one of four data types, as shown in Table 9-11. Because the justification and companding
format options function independently, the low bit specifies the justification format, and the high bit specifies the companding format:

Table 9-11. Data formats for multichannel operation

<table>
<thead>
<tr>
<th>DTYPE</th>
<th>Data Formatting</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>Right justify; fill unused MSBs with zeros (0)</td>
</tr>
<tr>
<td>x1</td>
<td>Right justify; extend sign into unused MSBs</td>
</tr>
<tr>
<td>0x</td>
<td>Compand using µ-law</td>
</tr>
<tr>
<td>1x</td>
<td>Compand using A-law</td>
</tr>
</tbody>
</table>

The multichannel compand select registers, MTCCSx and MRCCSx, enable companding on specific transmit and receive channel slots. (For details, see “Channel Selection Registers (MTCSx, MRCSx, MTCCSx, MRCCSx)” on page 9-72.) Linear transfers occur on a channel slot that is active and has companding disabled. Companded transfers occur on a channel slot that is active and has companding enabled.

In STCTLx, bit 0 of DTYPE selects transmit sign extension for all transmit channels. In SRCTLx, bit 0 of DTYPE selects receive sign extension for all receive channels. With bit 0 set, sign extension occurs on selected channels that have companding disabled. If this bit is cleared, the data word contains 0s in its MSBs.

**Companding**

Companding (compressing and expanding) is the process of logarithmically encoding and decoding data to minimize the number of bits that must be transmitted.

The processor’s serial ports support the two most widely used companding algorithms—A-law and µ-law—according to ITU G.711 specification. In standard and multichannel modes, you can select a companding algorithm
Data Word Formats

independently for each SPORT. (In standard mode, only the primary channels support companding.) The DTYPE field in the STCTLx and SRCTLx control registers selects the companding algorithm.

With companding enabled, the data in the Rx0_A or Rx1_A buffer is the right-justified, sign-extended expanded value of the eight LSBs received. Likewise, a write to Tx0_A and Tx1_A compresses the 32-bit value into eight LSBs (sign-extended to the width of the transmit word) before transmission. If the 32-bit value is greater than the 13-bit A-law or 14-bit µ-law maximum, the TX buffer automatically compresses it to the maximum value.

Because the values in the TX and RX buffers are companded in place, you can use the companding hardware without transmitting (or receiving) data, for example, during testing or debugging. This operation requires a single cycle of overhead.

To compand data in place:

1. Enable companding.
   
   Set the DTYPE field of the STCTLx transmit control register appropriately.

2. Write a 32-bit data word to TX.
   
   (Companding is calculated in this cycle.)

3. Wait one cycle.
   
   You can either insert a NOP instruction or not. Either way, the processor's core is held off for one cycle. (This delay enables the serial port companding hardware to reload TX with the companded value.)

4. Read the 8-bit companded value from TX.
To expand data in place, use the same procedure, but replace TX with RX. When performing this procedure, make sure to set the serial word length (SLEN) in the SRCTLx control register appropriately.

With companding enabled, interfacing the processor’s serial ports to a code requires little additional programming effort. With companding disabled, two formats for received data words of fewer than 32 bits are available (for details, see “Data Type (DTYPE)” on page 9-44).

**Data Packing and Unpacking (PACK)**

You can pack received data words of sixteen bits or less into 32-bit internal data words, and unpack 32-bit internal data words into 16-bit data words for transmission.

The PACK bit in the SRCTLx and STCTLx control registers enable word packing and unpacking.

In SRCTLx:

PACK=1  Pack two words received successively into a single 32-bit word.

In STCTLx:

PACK=1  Unpack each 32-bit word into two 16-bit words and transmit.

Packing right-justifies the first 16-bit (or smaller) data word in bits 15-0 of the packed word and right-justifies the second 16-bit (or smaller) word in bits 31-16. This procedure reverses during transmit (unpacking) operations.

You can compand and pack/unpack data concurrently.
Data Word Formats

With packing enabled, 32-bit packed words, not each 16-bit data word, generates the transmit and receive interrupts.

⚠️ Using short word space addresses, you can read and write 16-bit data words that have been packed into 32-bit words and stored in normal word space in internal memory.

Endian Format (SENDN)

Endian format determines whether the processor transmits the serial word MSB-first or LSB-first.

The SENDN bit in the STCTLx and SRCTLx control registers select endian format.

- **SENDN**\_\_z=0 Transmit or receive serial words MSB-first.
- **SENDN**\_\_z=1 Transmit or receive serial words LSB-first.

Word Length (SLEN)

The serial ports handle word lengths that range from three to thirty-two bits.

The five-bit SLEN field in the STCTLx and SRCTLx control registers configures the word length. The processor uses this value to determine how many bits to shift into or out of the shift register.
Serial Ports

The value of SLEN is equal to the word length minus one:

\[ SLEN = \text{Serial Word Length} - 1 \]

\[ \text{SLEN} \neq 0 \text{ or } 1 \]

The RX and TX buffers right-justify words smaller than thirty-two bits, so they occupy the least significant bit positions.

Transmitting or receiving words smaller than four bits at the processor’s full clock rate can cause loss of data when DMA chaining is enabled. Because chaining takes over the processor’s internal I/O bus for several cycles while the DMA controller loads new TCB chain parameters, received data in the RX buffer may be overwritten during this period.
Clock Signal Options

Each serial port has a transmit clock signal TCLKx and a receive clock signal RCLKx.

The ICLK and CKRE bits of the STCTLx and SRCTLx control registers configure the clock signals for standard and multichannel operation modes only.

The ICLK bits select the source of the transmit and receive clock signals. The CKRE bits select which clock edge (rising or falling) to use for synchronizing transmit and receive frames and for sampling data.

You configure the serial clock frequency in the TDIVx and RDIVx registers.

To use a single clock for both input and output, tie the receive clock pin to the transmit clock pin.

Internal vs. External Clocks

You can configure an internal or external clock source for the transmit and receive operations independently. The ICLK bit in the STCTLx and SRCTLx control registers selects the clock source.

When ICLK=1, the processor generates the clock signal, and the TCLKx or RCLKx pins are output pins.

The value of the serial clock divisor TCLKDIV or RCLKDIV in the TDIVx or RDIVx register, sets the clock frequency.
Serial Ports

When $I_{CLK}=0$, the processor accepts the clock signal as an input on the TCLKx or RCLKx pins.

In this mode, the processor ignores the serial clock divisors in the TDIVx and RDIVx registers.

The processor does not require synchronization between an externally generated serial clock and its system clock.
Frame Sync Options

Framing signals indicate the beginning of each serial word transfer. For frame sync operation, the processor supports a variety of framing options. Framing options on transmit and receive serial port channels are independent and configured separately in the STCTRLx and SRCTLx control registers.

The processor supports these frame sync options:

- Frame sync requirement (TFSR/RFSR)
- Frame sync source (ITFS/IRFS)
- Frame sync active state (LTFS/LRFS)
- Frame sync clock edge (CKRE)
- Frame sync insert (LAFS)
- Frame sync data dependency (DITFS)

Frame Sync Requirement (TFSR/RFSR)

Using frame sync signals is optional in serial port communications. In standard mode only, the TFSR (transmit frame sync required) and RFSR (receive frame sync required) control bits determine whether frame sync signals are required.

When $TFSR=1$ or $RFSR=1$, every data word requires a frame sync signal. To enable continuous transmissions from the ADSP-21065L, the processor must load each new data word into the TX buffer before shifting out and transmitting the last bit of the previous word. (See “Frame Sync Data Dependency (DITFS)” on page 9-57.)

When $TFSR=0$ or $RFSR=0$, data words do not require the corresponding frame sync signal, but initiating communications requires a single frame
After the processor transfers the first bit, it ignores the frame sync signal and continuously transmits data words unframed.

When DMA is enabled with frame syncs not required, chaining may hold off DMA requests or the DMA controller may not service requests frequently enough to guarantee continuous, unframed data flow.

Figure 9-10 on page 9-54 shows framed serial transfers, which have the following characteristics:

- TFSR and RFSR bits in STCTLx, SRCTLx control registers determine framed or unframed mode.
- Framed mode requires a framing signal for every word. Unframed mode ignores the framing signal after the first word.
- Unframed mode is appropriate for continuous reception.
- Active-low or active-high frame syncs selected with LTFS and LRFS bits of STCTLx, SRCTLx control registers.
Figure 9-10. Framed vs. unframed data

Frame Sync Options

In standard mode and multichannel mode (receive only), you can configure an internal or external frame sync source for transmit and receive operations independently.

When ITFS=1 or IRFS=1, the processor generates the corresponding frame sync signal internally, and the TFSx pin or RFSx pin becomes an output pin. The value of the frame sync divisor TFSDIV or RFSDIV in the TDIVx or RDIVx registers determines the frequency of the frame sync signal.

When ITFS=0 or IRFS=0, the processor accepts the corresponding frame sync signal as an input on the TFSx pin or RFSx pin and ignores the frame sync divisors in the TDIVx or RDIVx register.
All of the various frame sync options are available whether the signal is generated internally or externally.

**Frame Sync Active State (LTFS/RTFS)**

In standard mode and multichannel mode, you can configure the logic level of frame sync signals for active high operation or for active low (inverted) operation.

When \( \text{LTFS} = 0 \) or \( \text{LRFS} = 0 \), the corresponding frame sync signal is active high. This value is the default configuration, and a processor reset initializes the LTFS and LRFS bits to 0.

When \( \text{LTFS} = 1 \) or \( \text{LRFS} = 1 \), the corresponding frame sync signal is active low.

**Frame Sync Clock Edge (CKRE)**

In standard mode and multichannel mode, you can configure on which edge of serial port clock signals the processor samples data and frame syncs—either on the rising edge or on the falling edge.

For transmit data and frame syncs, setting \( \text{CKRE} = 1 \) selects the rising edge of TCLKx. \( \text{CKRE} = 0 \) selects the falling edge of TCLKx. Data and frame sync signals change state on whatever clock edge is not selected.

For receive data and frame syncs, setting \( \text{CKRE} = 1 \) causes the processor to clock data in on the rising edge of RCLKx. \( \text{CKRE} = 0 \) causes the processor to clock data in on the falling edge of RCLKx.

If you connect the transmit and receive functions of two serial ports together, make sure you configure the connections with the same value for CKRE, so the processor drives internally generated signals on one edge and samples received signals on the opposite edge.
Frame Sync Options

Frame Sync Insert (LAFS)

In standard mode, you can configure when the processor generates frame sync signals (for multichannel mode, MFD=1, see page 9-67). Frame sync signals can occur during the first bit of each data word (late) or during the serial clock cycle immediately preceding the first bit (early).

Setting LAFS=0 selects early frame sync mode (normal operation). In this mode, the first bit of the transmit data word is available (and the first bit of the receive data word is latched) in the serial clock cycle after the frame sync is asserted, and the frame sync is not checked again until the entire word has been transmitted (or received). (In multichannel operation, this occurs when frame delay is 1.)

In early frame sync mode, if data transmission is continuous (the first bit of the next word immediately follows the last bit of each word), the frame sync signal occurs during the last bit of each word. In early frame sync mode, the processor asserts internally generated frame syncs for one clock cycle.

Setting LAFS=1 selects late frame sync mode. In this mode, the first bit of the transmit data word is available (and the first bit of the receive data word is latched) in the same serial clock cycle that the frame sync is asserted. (In multichannel operation, this occurs when frame delay is 0.)

Serial clock edges latch receive data bits, but the frame sync signal is checked during the first bit of each word only. In late frame sync mode, the processor continues to assert internally generated frame syncs for the entire length of the data word. Externally generated frame syncs are checked during the first bit only.
Serial Ports

Figure 9-11 illustrates the two modes of frame signal timing:

- LAFS bits of STCTLx, SRCTLx control registers. LAFS=0 for early frame syncs, LAFS=1 for late frame syncs.
- Early framing: frame sync precedes data by one cycle. Late framing: frame sync checked on first bit only.
- Data transmitted MSB-first (SENDN=0) or LSB-first (SENDN=1).
- Frame sync and clock generated internally or externally.

![Diagram of frame signal timing]

Figure 9-11. Normal vs. alternate frame

Frame Sync Data Dependency (DITFS)

In all operation modes, you can configure the conditions that govern when the processor outputs internally-generated transmit frame sync (TFS) signal.

Normally, the processor outputs a TFS only when the TX buffer has data ready to transmit (data-dependent transmit frame sync). DITFS (data-
Frame Sync Options

independent transmit frame sync) mode enables the processor to continuously generate the TFS signal, with or without new data.

When \( DITFS=0 \), the processor outputs TFS only when the TX buffer contains a new data word. Once loaded into the TX buffer, a new data word is transmitted two cycles after the processor generates the next TFS. Data-dependent mode provides the method to transmit data at specific times only.

When \( DITFS=1 \), the processor outputs TFS at its programmed interval, regardless of whether new data is available in the TX buffer. In data-independent mode, with each assertion of TFS, the processor transmits whatever data is present in the TX buffer. When old data is retransmitted, the processor sets the transmit underflow status bit (TUVF) in the STCTLx control register. The processor also sets the TUVF status bit if the TX buffer does not have new data when an externally generated TFS occurs. In data-independent mode, the first internally generated TFS is delayed until data has been loaded into the TX buffer.

With \( DITFS=1 \), initiating a transfer requires a single write to the TX data register.
Standard Mode

In standard mode, you can enable either one or both of the SPORTs’ transmit channels. The frame sync source determines their transmit configuration.

When using both transmitters simultaneously, both TX buffers must contain data. For continuous transmission, both TX buffers must contain new data.

The receiving SPORT receives on both Rx_A and Rx_B. But only a SPORT with DMA enabled generates DMA requests or DMA interrupts upon receiving data.

Each SPORT transmit and receive channel has its own channel enable, DMA enable, and chaining enable bits in its STCTLx and SRCTLx control register.

The SPORTs support companding on the primary channels, Tx_A and Rx_A, only.

Enabling Standard Mode (OPMODE, MCE)

You enable standard mode with the OMODE and MCE bits (STCTLx and SRCTLx). To do so, set both bits to 0.

Frame Sync Configuration (FS_BOTH)

In standard mode, FS_BOTH (STCTLx) specifies when the processor generates the transmit frame sync signal.

FS_BOTH=0 Generate frame sync when data is available in either transmit channel.

FS_BOTH=1 Generate frame sync only when data is available in both transmit channels.
Standard Mode

When both transmitters are transmitting simultaneously (FS_BOTH=1), the processor generates frame syncs only when both transmitters contain data. For continuous transmission when using both transmitters simultaneously, both transmitters must contain new data.

To implement this mode, you must also configure the processor for data-independent TFS and as TFS source:

\[
\begin{align*}
\text{DITFS} &= 1 \\
\text{ITFS} &= 1
\end{align*}
\]

Setting the Serial Clock Frequency (CLKDIV)

You can set the serial clock frequency for the processors internal clocks. For details see, “Clock and Frame Sync Frequencies (TDIV, RDIV)” on page 9-39.


I²S Mode

I²S mode supports the Inter-IC sound bus protocol developed for exchanging audio data between digital audio processors over a serial link.

The I²S bus transmits audio data and control signals over separate lines. The data line carries two multiplexed data channels, the left channel and the right channel.

In I²S mode:

- Both SPORT transmit channels (Tx_A and Tx_B) always transmit simultaneously, each transmitting left and right I²S channels.
- Both SPORT receive channels (Rx_A and Rx_B) always receive simultaneously, each receiving left and right I²S channels.
- Data always transmits in MSB format.
- You can select either DMA-driven or interrupt-drive data transfers.
- TFS and RFS are the transmit and receive word select signals.
- Multichannel operation and companding are not supported.

Each SPORT transmit and receive channel has its own channel enable, DMA enable, and chaining enable bits in its STCTLx and SRCTLx control register.

Setting the Internal Serial Clock Rate

You can program the serial clock rate (xCLKDIV value) for internal clocks in the CLKDIV registers. For details, see “Clock and Frame Sync Frequencies (TDIV, RDIV)” on page 9-39.
I²S Mode

In I²S mode, you must load both the TDIV register and the RDIV register with the same value as SLEN. For example, for 8-bit data words (SLEN=7), you must set TDIV = 7 and RDIV = 7.

I²S Control Bits

Several bits in the STCTLx and SRCTLx control registers enable and configure I²S operation:

- Operation mode (OPMODE)
- Multichannel enable (MCE)
- Word length (SLEN)
- I²S channel transfer order (L_FIRST)
- Frame sync (word select) generation (FS_BOTH)
- Master mode enable (MSTR)
- DMA enable (SDEN)
- DMA chaining enable (SCHEN)

Enabling I²S mode (OPMODE, MCE)

You enable I²S mode with the OPMODE and MCE bits (STCTLx and SRCTLx). With SPENx=1, set

```
OPMODE=1  Enable I²S mode
MCE=0     Disable multichannel mode
```
Serial Ports

Setting the Word Length (SLEN)

The SPORTs handle data words containing from 3 to 32 bits. You can set the number of bits transmit and receive data words contain. For details, see “Word Length (SLEN)” on page 9-48.

The transmitter always sends the MSB of the next word one clock cycle after the word select (TFS) signal changes.

In I²S mode, you must load the FSDIV register with the same value as SLEN. For example, for 8-bit data words (SLEN=7), you must set FSDIV= 7. For details, see “Clock and Frame Sync Frequencies (TDIV, RDIV)” on page 9-39.

Selecting the I²S Transmit and Receive Channel Order (L_FIRST)

You can configure which I²S channel each SPORT channel transmits or receives first. By default, the SPORT channels transmit and receive on the right I²S channel first. The left and right I²S channels are time-duplexed data channels.

To select the channel order, set the L_FIRST bit:

- L_FIRST=0 Transmit or receive on right channel first.
- L_FIRST=1 Transmit or receive on left channel first.

Selecting the Frame Sync options (FS_BOTH)

The processor uses TFS and RFS as transmit and receive word select signals. You can configure when the processor generates the transmit word select signal based on the data in the transmit channels.
I²S Mode

**FS_BOTH=0** Generate word select signal if either transmit channel contains data.

**FS_BOTH=1** Generate word select signal only if both transmit channels contain data.

The word select signal changes one clock cycle before the MSB of the data word transmits, enabling the slave transmitter to derive synchronous timing of the serial data and enabling the receiver to store the previous data word and clear its input for the next one.

When using both transmitters (FS_BOTH=1) and MSTR=1 and DITFS=0, the processor generates a frame sync signal only when both transmit buffers contain data because both transmitters share the same CLKDIV and TFS. So, for continuous transmission, both transmit buffers must contain new data. To enable continuous transmission when only one transmit buffer contains new data, set \( \text{FS\_BOTH}=0 \).

When using both transmitters and MSTR=1 and DITFS=1, the processor generates a frame sync signal at the frequency set by FSDIV=x whether or not the transmit buffers contain new data. In this case, the processor ignores the FS_BOTH bit. The DMA controller or the application is responsible for filling the transmit buffers with data.

Enabling SPORT Master Mode (MSTR)

You can configure the SPORTs transmit and receive channels for master or slave mode. In master mode, the processor generates the word select and serial clock signals for the transmitter or receiver internally. In slave mode, an external source generates the word select and serial clock signals for the transmitter or receiver.

**MSTR=0** Use external word select and clock source; transmitter or receiver is slave.

**MSTR=1** Use processor’s internal clock for word select and clock source; transmitter or receiver is master.
Enabling SPORT DMA (SDEN)

You can enable or disable DMA independently on any of the SPORT’s transmit and receive channels.

- SDEN\_z=0: Disables DMA and set channel in interrupt-driven data transfer mode.
- SDEN\_z=1: Enable DMA and set channel in DMA-driven data transfer mode.

**Interrupt-Driven Data Transfer Mode.** In this mode, both transmitters share a common interrupt vector and both receivers share a common interrupt vector.

The SPORT generates an interrupt whenever the transmit buffer has a vacancy or whenever the receive buffer has data. To determine the source of an interrupt, applications must check the TXSx or RXSx data buffer status bits, respectively.

**DMA-Driven Data Transfer Mode.** Each transmitter and receiver has its own set of DMA registers. (For details, see Chapter 6, DMA.) The same DMA channel drives both the left and right I²S channels for the transmitter or for the receiver. The software application must demultiplex the left and right channel data received by the RX buffer.

Both transmitters share a common interrupt vector and both receivers share a common interrupt vector. The DMA controller generates an interrupt at the end of DMA transfer only.
**I²S Mode**

Figure 9-12 shows the relationship between FS (word select), serial clock, and I²S data. Timing for word select is the same as for frame sync. (Note that this example uses early frame sync.)

![I²S Mode Diagram](image)

For WS = 0, X0 = left

Figure 9-12. Word select timing in I²S mode
Multichannel Mode

The processor’s serial ports support multichannel operation, which enables a SPORT to communicate in a time-division-multiplexed (TDM) serial system.

In multichannel communications, each data word in the serial bit stream occupies one channel slot. Data word 0 occupies channel slot 0, data word 1 occupies channel slot 1, ..., and data word \( n \) occupies channel slot \( n \). In this way, each data word in the stream belongs to the next consecutive channel slot so that, for example, a 24-word block of data contains one word for each of 24 channel slots.

A SPORT can automatically select words for particular channel slots while ignoring others. The processor supports up to thirty-two channel slots for transmitting or receiving—each SPORT can receive and transmit data selectively from any of the thirty-two channel slots.

On each channel slot, a SPORT can:

- Transmit data
- Receive data
- Transmit and receive data
- Do nothing

Multichannel mode also supports data companding and DMA transfers.

In this mode only, if the SPORT is enabled, the processor puts the DT pin in a high-impedance state when an inactive channel slot occurs.

⚠️ In multichannel mode, the TCLKx pin is always an input and must connect to its corresponding RCLKx pin.
Multichannel Mode

Figure 9-13 shows example timing for a multichannel transfer, which has the following characteristics:

- Uses TDM method, where serial data is sent or received on different channel slots sharing the same serial bus.
- The number of channel slots is selected with the NCH bits of SRCTLx: NCH=(# of channels) – 1.
- Can independently select transmit and receive channels.
- RFS signals start of frame.
- TFS is used as “Transmit Data Valid” for external logic; active only during transmit channels.
- Example: Receive on channels 0 and 2 and transmit on channels 1 and 2.

Figure 9-13. Multichannel operation
Frame Syncs in Multichannel Mode

All receiving and transmitting devices in a multichannel system must have the same timing reference. The RFS signal provides this reference, indicating the start of a block (or frame) of multichannel data words.

With multichannel mode enabled, the SPORT’s transmitter and receiver both use RFS as a frame sync, whether RFS is internally or externally generated. The RFS signal synchronizes the channel slots and restarts each multichannel sequence. RFS assertion occurs at the beginning of the channel 0 data word.

TFS functions as a transmit data valid signal, which is active during transmission of an enabled word. Since the processor puts the serial port’s DTx pin in a high-impedance state when the time slot is inactive, the TFS signal specifies whether or not the processor is driving the DTx pin. The processor drives TFS in multichannel mode, whether or not \( \text{ITFS}=0 \) (external TFS source).

Transmission begins after the TX transmit buffer is loaded and the processor generates the TFS signal. With serial port DMA enabled, transmission can occur several cycles after the multichannel transmission is enabled. If your application requires a deterministic start time, have it preload the TX buffer.

In multichannel mode, TFS remains unconnected normally, and the serial ports’ RFS pins connect together.

Multichannel Control Bits

Several bits in the STCTLx and SRCTLx control registers enable and configure multichannel operation:

- Operation mode (OPMODE)
- Multichannel enable (MCE)
Multichannel Mode

- Number of channel slots (NCH)
- Current channel slot indicator (CHNL)
- Multichannel frame delay (MFD)
- Channel slot transmit/receive select (MTCS/MRCS)
- Channel slot transmit/receive compand select (MTCCS/MRCCS)

Operation Mode (OPMODE)

The operation mode bit enables and disables I2S mode and redefines the SPORT control bits accordingly. The multichannel enable (MCE) bit affects SPORT operation only when I2S mode is disabled.

Multichannel Enable (MCE)

Setting the MCE bit enables multichannel mode only when OPMODE=0.

\[
\begin{align*}
\text{MCE=1} & \quad \text{Enable multichannel operation.} \\
\text{MCE=0} & \quad \text{Disable all multichannel operations.}
\end{align*}
\]

Multichannel operation activates three cycles after MCE is set. Internally generated frame sync signals activate four cycles after MCE is set.

Setting the MCE bit enables multichannel operation for the SPORTs primary set of transmit and receive channels. Therefore, if the receiving SPORT is in multichannel mode, the transmitting SPORT is too.
Number of Channel Slots

The five-bit NCH field (SRCTLx) sets the number of channel slots to use in multichannel operation. Set NCH to the actual number of channels minus one:

\[ NCH = \text{Number of Channels} - 1 \]

The SPORTs support up to thirty-two channel slots.

Current Channel Selected

The five-bit CHNL field (STCTLx) indicates which channel slot is currently selected during multichannel operation. This field is a read-only status indicator. CHNL(4:0) increments modulo NCH(4:0) as the SPORT services each channel slot.

Multichannel Frame Delay

The four-bit MFD field (STCTLx) specifies a delay, in number of serial clock cycles, between the frame sync pulse and the first data bit in multichannel mode. Multichannel frame delay enables the processor to work with different types of T1 interface devices.

- **MFD=0**: No delay; frame sync concurrent with the first data bit.
- **MFD=x**: Frame sync delayed \( x \) clock cycles.

The maximum is 15 clock cycles. Because blocks of data occur back to back, new frame sync may occur before data from the last frame has been received.

When the processor is RFS source in a multiprocessor system and the system's serial clock is equal to CLKin (processor clock), use an MFD \( \geq 1 \). Otherwise, the system's master processor will not recognize the first frame sync after multichannel operation has been enabled. (It will, however, recognize all succeeding frame syncs.)
Multichannel Mode

Channel Selection Registers (MTCSx, MRCSx, MTCCSx, MRCCSx)

You can enable and disable specific channel slots individually to select which words are received and transmitted during multichannel communications.

The processor transmits and receives only data words from enabled channel slots and ignores data words on disabled channel slots. The SPORTs support a maximum of thirty-two channel slots for transmitting and for receiving.

The multichannel selection registers enable and disable (activate and deactivate) individual transmit and receive channel slots and enable and disable companding on them. Table 9-12 lists the registers for each serial port.

Table 9-12. Multichannel selection register definitions

<table>
<thead>
<tr>
<th>Register</th>
<th>Selects...</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTCSx</td>
<td>Multichannel transmit select. Specifies the active transmit channels.</td>
</tr>
<tr>
<td>MRCSx</td>
<td>Multichannel receive select. Specifies the active receive channels.</td>
</tr>
<tr>
<td>MTCCSx</td>
<td>Multichannel transmit compand select. Specifies which active channels are companded.</td>
</tr>
<tr>
<td>MRCCSx</td>
<td>Multichannel receive compand select. Specifies which active receive channels are companded.</td>
</tr>
</tbody>
</table>

Each register has thirty-two bits that correspond to the thirty-two channel slots. Setting a bit activates the corresponding channel slot, so the SPORT
Serial Ports

selects the data word it contains from the multiple-word data block. For example, setting bit 0 selects data word 0, setting bit 12 selects data word 12, and so on.

Setting a particular bit to 1 in the MTCSx register causes the SPORT to transmit the data word in that channel slot’s position in the data stream. Clearing the bit to 0 puts the SPORT’s DT (data transmit) pin into Hi-Z during the time of that channel slot.

Setting a particular bit to 1 in the MRCSx register causes the SPORT to receive the data word in that channel slot’s position in the data stream. The processor loads the received word into the RX buffer. Clearing the bit to 0 causes the SPORT to ignore the data.

You can also select companding on a per channel basis. The MTCCSx and MRCCSx registers specify companding for any active channel slots. Setting a bit to 1 in these registers causes the SPORT to compand the data word using either the A-law or µ-law companding algorithm. All channels configured for companding must use the same companding algorithm. (To select the companding algorithm, see “Data Type (DTYPE)” on page 9-44).

SPORT Receive Comparison Registers (KEYWDx and IMASKx)

In SPORT multichannel mode (MCE=1), the 32-bit receive comparison (KEYWDx) and receive comparison mask (IMASKx) registers aid multiprocessor communications.

The KEYWDx register stores the pattern against which to match the incoming data. The corresponding IMASKx register specifies which bits in the received data to compare. Setting a bit in IMASKx to 1 masks the corresponding bit in the KEYWDx register, removing it from comparison.

The receiving processor compares the received data with the data pattern in its KEYWDx register. Depending on the results, the processor accepts
the received data or ignores it. On acceptance, depending on the SDENx setting in SRCTLx, the receiver either requests a DMA transfer to internal memory or generates an interrupt.

These bits (SRCTL) control the operation of the receive comparison in multichannel mode, as shown in Table 9-13.

Table 9-13. SRCTL control bits for receive comparison

<table>
<thead>
<tr>
<th>IMODE</th>
<th>IMAT</th>
<th>Selects…</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>Receive comparison disabled.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Accept receive data if the KEYWD compares false.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Accept receive data if the KEYWD compares true.</td>
</tr>
</tbody>
</table>

With receive comparison enabled, companding is disabled on both transmitter and receiver.

The MTCCSx register, which selects multichannel companding when receive comparison is disabled, determines whether the DSP performs a KEYWD comparison for the enabled received channel slots.

\[
\text{MTCCSx}=0 \quad \text{On channel slot } x, \text{ disable receive comparison and always accept received data.} \\
\text{MTCCSx}=1 \quad \text{On channel slot } x, \text{ enable comparison and accept or reject received data} \text{ based on comparison results and IMAT value (SRCTLx).}
\]

The receive comparison feature enables the SPORT to determine whether to generate either a DMA request or an interrupt when received data matches a specified condition on a specified channel. Otherwise, every time it received data the SPORT would have to interrupt the processor,
which would have to determine whether the data was meant for it. And SPORT data is often in transit to other than the processor. With the receive comparison feature, you can program a SPORT on a particular processor to interrupt only on messages meant for its processor.

For example, consider two ADSP-21065s (A and B) which use SPORT0 in multichannel mode for interprocessor communications. Processors A and B use channel slots 0 and 1, respectively, to transmit control information between them. To transmit data, processor A uses channel slots 2 through 16, and processor B uses channel slots 17 through 31.

Because channel slots 0 and 1 carry control information between the processors, receive comparison on incoming data is enabled only on these channel slots. Initially, receive may be disabled on channel slots 2 through 31. In this example, the programmed key word for processor B to compare against is **START TRANSMIT TO B**.

To check for this keyword, processor B:

1. Sets the KEYWDx register to **START TRANSMIT TO B**.
2. In IMASKx, sets bits 31:16 to 0 and sets bits 15:0 to 1
   
   This step enables receive comparison on bits 31:16 only. Assume that the code for **START TRANSMIT TO B** uses bits 31:16 only and that bits 15:0 indicate the transmission source and data channel slots.
3. Sets the IMODE and IMAT (SRCTLx) bits to 1.
   
   This step enables the SPORT to generate either an interrupt or DMA request only if the incoming data matches the KEYWDx.
4. Sets bits 0 and 1 of MTCCSx to 1 and clears the remaining bits 31:2.
   
   This step enables comparison only on channel slots 0 and 1.
Communication between the two processors follows this sequence:

1. Until it receives the `START TRANSMIT TO B` keyword, processor B ignores all transmissions that it receives.

2. To initiate transmission to B, processor A sends the `START TRANSMIT TO B` keyword on channel slot 0.

3. When processor B’s receive comparison logic recognizes the `START TRANSMIT TO B` keyword, the SPORT interrupts its processor.

4. Processor B analyzes the remaining 16-bits and determines that the transmit source is processor A and that the data is on channel slots 2:16.

5. Because processor A is using channel slots 2 through 16 to transmit data, processor B enables receive channel slots 2 through 16 and sends a `READY TO RECEIVE DATA` message to processor A on channel slot 1.

6. After receiving this message, processor A sends the data on channel slots 2 through 16.

If the transfer protocol uses a fixed number of bytes in each message, to confirm that the data transferred accurately, processor B can return a checksum message to processor A after receiving A’s message.
Moving Data Between SPORTs and Memory

You can transfer transmit and receive data between the SPORTs and on-chip memory in one of two ways: with single-word, core transfers or with DMA block transfers. Both methods are interrupt-driven and use the same internally generated interrupts.

When serial port DMA is disabled (STCTLx or SRCTLx), the SPORT generates an interrupt every time it receives a data word or starts to transmit a data word.

SPORT DMA provides a mechanism for receiving or transmitting an entire block of serial data before the SPORT generates the interrupt. The processor’s on-chip DMA controller handles the DMA transfer, enabling the core to continue executing program until the entire block of data has been transmitted or received. Service routines that operate on blocks of data instead of single words significantly reduce overhead.

DMA Block Transfers

The processor’s on-chip DMA controller enables automatic DMA transfers between internal memory and the two serial ports.

Eight DMA channels support serial port operations—each SPORT has two channels for receiving data and two channels for transmitting data. Table 9-14 on page 9-78 lists each serial port DMA channels and its data buffer.
Because of their relatively low service rate and their inability to hold off incoming data, the SPORT DMA channels have higher priority than external port DMA channels. Because they have higher priority, the DMA controller performs SPORT DMA transfers first when it receives multiple DMA requests in the same cycle.

Although DMA transfers always use 32-bit words, the serial ports can handle word sizes from 3 to 32 bits. If serial words are 16 bits or smaller, the SPORTs can pack them into 32-bit words for each DMA transfer. You configure packing with the PACK bit in the STCTLx and SRCTLx control registers.

Table 9-14. DMA serial port channels

<table>
<thead>
<tr>
<th>Channel</th>
<th>Data Buffer</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rx0_A</td>
<td>SPORT0 Receive, A data</td>
<td>Highest</td>
</tr>
<tr>
<td>2</td>
<td>Rx0_B</td>
<td>SPORT0 Receive, B data</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Rx1_A</td>
<td>SPORT1 Receive, A data</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Rx1_B</td>
<td>SPORT1 Receive, B data</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Tx0_A</td>
<td>SPORT0 Transmit, A data</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Tx0_B</td>
<td>SPORT0 Transmit, B data</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Tx1_A</td>
<td>SPORT1 Transmit, A data</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Tx1_B</td>
<td>SPORT1 Transmit, B data</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>EPB0</td>
<td>External port FIFO buffer 0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>EPB1</td>
<td>External port FIFO buffer 1</td>
<td>Lowest</td>
</tr>
</tbody>
</table>
With packing enabled (PACK=1), the SPORT generates the transmit and receive interrupts for the 32-bit packed words, not for each 16-bit serial word.

The following sections describe serial port DMA operations. For details on other DMA operations, see Chapter 6, DMA.

**Setting Up DMA on SPORT Channels**

Each SPORT DMA channel has an enable bit SDEN in its STCTLx and SRCTLx control registers.

When DMA is disabled for a particular channel, the SPORT generates an interrupt every time it receives a data word or starts transmitting a data word (see “Single-Word Transfers” on page 9-86).

Each channel also has a DMA chaining enable bit SCHEN in its STCTLx and SRCTLx control registers. For details, see “SPORT DMA Chaining” on page 9-85.

To set up a serial port DMA channel, you write a set of memory buffer parameters to the SPORT DMA parameter registers shown in Table 9-15.

<table>
<thead>
<tr>
<th>Table 9-15. SPORT DMA parameter registers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Register</strong></td>
</tr>
<tr>
<td><strong>SPORT Rx_X Channels</strong></td>
</tr>
<tr>
<td>IIRx_X</td>
</tr>
<tr>
<td>IMRx_X</td>
</tr>
<tr>
<td>CRx_X</td>
</tr>
<tr>
<td>CPRx_X</td>
</tr>
</tbody>
</table>
Moving Data Between SPORTs and Memory

Table 9-15. SPORT DMA parameter registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPRx_X</td>
<td>DMA channel x general purpose</td>
</tr>
<tr>
<td>IITx_X</td>
<td>DMA channel x index; Start address for data buffer</td>
</tr>
<tr>
<td>IMTx_X</td>
<td>DMA channel x modify; Address increment</td>
</tr>
<tr>
<td>CTx_X</td>
<td>DMA channel x count; Number of words to transmit</td>
</tr>
<tr>
<td>CPTx_X</td>
<td>DMA channel x chain pointer; Address next set of data buffer parameter</td>
</tr>
<tr>
<td>GPTx_X</td>
<td>DMA channel x general purpose</td>
</tr>
</tbody>
</table>

You must load the II, IM, and C registers with a starting address for the buffer, an address modifier, and a word count, respectively. You can program these registers from the processor or from an external processor.

Once you set up and enable serial port DMA, the processor’s DMA controller automatically transfers received data words in the RX buffer to the buffer in internal memory. Likewise, when the serial port is ready to transmit data, the DMA controller automatically transfers a word from internal memory to the TX buffer. The controller continues these transfers until the entire data buffer is received or transmitted—when the count register reaches zero.

When the count register of an active DMA channel reaches zero (0), the SPORT generates the corresponding interrupt.
SPORT DMA Parameter Registers

A DMA channel consists of a set of parameter registers that implement a data buffer in internal memory and the hardware that the serial port uses to request DMA service.

The parameter registers for each SPORT DMA channel are shown in Table 9-15 on page 9-79. These registers are part of the processor’s memory-mapped IOP register set, and their addresses are shown in Table 9-16 on page 9-82.

The DMA channels operate similarly to the processor’s data address generators (DAGs). Each channel has an index register (II) and a modify register (IM) for setting up a data buffer in internal memory. You must initialize the index register with the starting address of the data buffer. After it transfers each serial I/O word to or from the SPORT, to generate the address for the next DMA transfer, the DMA controller adds the modify value to the index register. The modify value in the IM register is a signed integer, which provides capability for both incrementing and decrementing the buffer pointer.

Each DMA channel has a count register C, which you must initialize with a word count that specifies the number of words to transfer. The count register decrements after each DMA transfer on the channel. When the word count reaches zero, the SPORT generates the interrupt for the channel and automatically disables the DMA channel.

Each SPORT DMA channel also has a chain pointer register CP and a general-purpose register GP. The CP register functions in chained DMA operations (see “SPORT DMA Chaining” on page 9-85), and you can use the GP register for any purpose.
## Moving Data Between SPORTs and Memory

### Table 9-16. Addresses of DMA parameter registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>DMA Chn.</th>
<th>SPORT Chn.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIR0B</td>
<td>0x0030</td>
<td>1</td>
<td>Rx0_B</td>
</tr>
<tr>
<td>IMR0B</td>
<td>0x0031</td>
<td>1</td>
<td>Rx0_B</td>
</tr>
<tr>
<td>CR0B</td>
<td>0x0032</td>
<td>1</td>
<td>Rx0_B</td>
</tr>
<tr>
<td>CPR0B</td>
<td>0x0033</td>
<td>1</td>
<td>Rx0_B</td>
</tr>
<tr>
<td>GPR0B</td>
<td>0x0034</td>
<td>1</td>
<td>Rx0_B</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x0035 - 0x0036</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMASTAT</td>
<td>0x0037</td>
<td></td>
<td>DMA channel status register</td>
</tr>
<tr>
<td>IIR1B</td>
<td>0x0038</td>
<td>3</td>
<td>Rx1_B</td>
</tr>
<tr>
<td>IMR1B</td>
<td>0x0039</td>
<td>3</td>
<td>Rx1_B</td>
</tr>
<tr>
<td>CR1B</td>
<td>0x003A</td>
<td>3</td>
<td>Rx1_B</td>
</tr>
<tr>
<td>CPR1B</td>
<td>0x003B</td>
<td>3</td>
<td>Rx1_B</td>
</tr>
<tr>
<td>GPR1B</td>
<td>0x003C</td>
<td>3</td>
<td>Rx1_B</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x003D - 0x003F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIEP0</td>
<td>0x0040</td>
<td>8</td>
<td>EPB0</td>
</tr>
<tr>
<td>IMEP0</td>
<td>0x0041</td>
<td>8</td>
<td>EPB0</td>
</tr>
<tr>
<td>CEP0</td>
<td>0x0042</td>
<td>8</td>
<td>EPB0</td>
</tr>
<tr>
<td>CPEP0</td>
<td>0x0043</td>
<td>8</td>
<td>EPB0</td>
</tr>
<tr>
<td>GPEP0</td>
<td>0x0044</td>
<td>8</td>
<td>EPB0</td>
</tr>
</tbody>
</table>
Table 9-16. Addresses of DMA parameter registers  (Cont’d)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>DMA Chn.</th>
<th>SPORT Chn.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1EP0</td>
<td>0x0045</td>
<td>8</td>
<td>EPB0</td>
</tr>
<tr>
<td>EMEP0</td>
<td>0x0046</td>
<td>8</td>
<td>EPB0</td>
</tr>
<tr>
<td>ECEP0</td>
<td>0x0047</td>
<td>8</td>
<td>EPB0</td>
</tr>
<tr>
<td>IIEP1</td>
<td>0x0048</td>
<td>9</td>
<td>EPB1</td>
</tr>
<tr>
<td>IMEP1</td>
<td>0x0049</td>
<td>9</td>
<td>EPB1</td>
</tr>
<tr>
<td>CEP1</td>
<td>0x004A</td>
<td>9</td>
<td>EPB1</td>
</tr>
<tr>
<td>CPEP1</td>
<td>0x004B</td>
<td>9</td>
<td>EPB1</td>
</tr>
<tr>
<td>GPEP1</td>
<td>0x004C</td>
<td>9</td>
<td>EPB1</td>
</tr>
<tr>
<td>E1EP1</td>
<td>0x004D</td>
<td>9</td>
<td>EPB1</td>
</tr>
<tr>
<td>EMEP1</td>
<td>0x004E</td>
<td>9</td>
<td>EPB1</td>
</tr>
<tr>
<td>ECEP1</td>
<td>0x004F</td>
<td>9</td>
<td>EPB1</td>
</tr>
<tr>
<td>IIT0B</td>
<td>0x0050</td>
<td>5</td>
<td>Tx0_B</td>
</tr>
<tr>
<td>IMT0B</td>
<td>0x0051</td>
<td>5</td>
<td>Tx0_B</td>
</tr>
<tr>
<td>CTOB</td>
<td>0x0052</td>
<td>5</td>
<td>Tx0_B</td>
</tr>
<tr>
<td>CPT0B</td>
<td>0x0053</td>
<td>5</td>
<td>Tx0_B</td>
</tr>
<tr>
<td>GPT0B</td>
<td>0x0054</td>
<td>5</td>
<td>Tx0_B</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x0055 - 0x0057</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIT1B</td>
<td>0x0058</td>
<td>7</td>
<td>Tx1_B</td>
</tr>
<tr>
<td>IMT1B</td>
<td>0x0059</td>
<td>7</td>
<td>Tx1_B</td>
</tr>
</tbody>
</table>
### Moving Data Between SPORTs and Memory

Table 9-16. Addresses of DMA parameter registers (Cont’d)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>DMA Chn.</th>
<th>SPORT Chn.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT1B</td>
<td>0x005A</td>
<td>7</td>
<td>Tx1_B</td>
</tr>
<tr>
<td>CPT1B</td>
<td>0x005B</td>
<td>7</td>
<td>Tx1_B</td>
</tr>
<tr>
<td>GPT1B</td>
<td>0x005C</td>
<td>7</td>
<td>Tx1_B</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x005D - 0x005F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIR0A</td>
<td>0x0060</td>
<td>0</td>
<td>Rx0_A</td>
</tr>
<tr>
<td>IMR0A</td>
<td>0x0061</td>
<td>0</td>
<td>Rx0_A</td>
</tr>
<tr>
<td>CROA</td>
<td>0x0062</td>
<td>0</td>
<td>Rx0_A</td>
</tr>
<tr>
<td>CPR0A</td>
<td>0x0063</td>
<td>0</td>
<td>Rx0_A</td>
</tr>
<tr>
<td>GPR0A</td>
<td>0x0064</td>
<td>0</td>
<td>Rx0_A</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x0065 - 0x0067</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIR1A</td>
<td>0x0068</td>
<td>2</td>
<td>Rx1_A</td>
</tr>
<tr>
<td>IMR1A</td>
<td>0x0069</td>
<td>2</td>
<td>Rx1_A</td>
</tr>
<tr>
<td>CR1A</td>
<td>0x006A</td>
<td>2</td>
<td>Rx1_A</td>
</tr>
<tr>
<td>CPR1A</td>
<td>0x006B</td>
<td>2</td>
<td>Rx1_A</td>
</tr>
<tr>
<td>GPR1A</td>
<td>0x006C</td>
<td>2</td>
<td>Rx1_A</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x006D - 0x006F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIT0A</td>
<td>0x0070</td>
<td>4</td>
<td>Tx0_A</td>
</tr>
<tr>
<td>IMT0A</td>
<td>0x0071</td>
<td>4</td>
<td>Tx0_A</td>
</tr>
<tr>
<td>CTOA</td>
<td>0x0072</td>
<td>4</td>
<td>Tx0_A</td>
</tr>
</tbody>
</table>
In chained DMA operations, the processor’s DMA controller automatically sets up another DMA transfer when the contents of the current buffer have been transmitted (or received). The chain pointer register (CP) functions as a pointer to the next set of buffer parameters stored in memory. The DMA controller automatically downloads these buffer parameters to set up the next DMA sequence. For details, see Chapter 6, *DMA*.

DMA chaining occurs independently for the transmit and receive channels of each serial port. Each SPORT DMA channel has a chaining enable bit SCHEN (STCTLx and SRCTLx).
Moving Data Between SPORTs and Memory

\( \text{SCHEN}_z = 0 \) Disable DMA chaining

\( \text{SCHEN}_z = 1 \) Enable DMA chaining

(You can also write all 0s to the address field of the chain pointer register (CP) to disable chaining.)

Single-Word Transfers

The SPORTs can also transfer individual data words, generating interrupts for each 32-bit word transfer.

When a serial port is enabled and DMA is disabled (STCTLx or SRCTLx), the SPORT generates DMA interrupts whenever:

- The RX buffer has received an entire word.
- The TX buffer is not full.

This behavior enables you to use single-word interrupts to implement interrupt-driven I/O on the serial ports.

Whenever the processor’s core program reads a word from a serial port’s RX buffer or writes a word to its TX buffer, make sure it checks the buffer’s full/empty status first to avoid hanging the core. (This can happen to an external device too, such as a host processor, when it is reading or writing a serial port buffer.) To check buffer status, read the RXS bits or the TXS bits in the SRCTLx or STCTLx control register.

Reading from an empty RX buffer or writing to a full TX buffer causes the processor (or external device) to hang, waiting for the status to change. To prevent this hang condition, in the SYSCON register, set the BHD (Buffer Hang Disable) bit to 1.

The processor updates the status bits in STCTLx and SRCTLx during core reads and writes, even when the serial port is disabled. For details, see page 9-7.
Multiple interrupts can occur if both SPORTs transmit or receive data in the same cycle. You can mask out any interrupt in the IMASK register. If you re-enable the interrupt in IMASK later, clear the corresponding interrupt latch bit in IRPTL in case the interrupt occurred while it was masked.

With serial port data packing enabled \((\text{PACK}=1)\), the SPORT generates the transmit and receive interrupts for the 32-bit packed words, not for each 16-bit serial word.
SPORT Loopback

In standard and I^2S modes, the SPL bit (SPORT loopback mode) in the SRCTLx control register configures the serial port for internal loopback connection. SPORT loopback mode enables you to test the serial port’s internal operation.

- SPL=0 Disable SPORT loopback mode.
- SPL=1 Enable SPORT loopback mode.

With loopback enabled, the DRx, RCLKx, and RFSx signals of the SPORT’s receive section internally connect to the DTx, TCLKx, and TFSx signals of the transmit section. The DTx, TCLKx, and TFSx signals are active and available at their respective pins, while the processor ignores the DRx, RCLKx, and RFSx pins.

In loopback mode, you can use only the transmit clock and transmit frame sync options, and you must make sure that you set up the serial port correctly in the STCTLx and SRCTLx control registers.

Loopback mode does not support multichannel operation.

SPORT Pin Driver Considerations

The processor has very fast drivers on all output pins, including the serial ports. If connections on the data, clock, or frame sync lines are longer than six inches, we recommend that you use a series termination for strip lines on point-to-point connections. Because of the edge rates, this hardware may be necessary even for low-speed serial clocks.
SPORT Programming Examples

The processor provides three ways to control serial port communications and memory-to-SPORT data transfers:

- Single-word transfers under core processor control with no interrupts.
- Single-word transfers under core processor control with interrupts.
- DMA transfers with interrupts.

The three examples presented next illustrate each of these methods. Each example uses SPORT0 to transmit eight 32-bit words from a data buffer in internal memory.

Each of the three control schemes also operates in multichannel mode and with any of the serial clock and frame sync options.

Single-Word Transfers Without Interrupts

The processor’s core will stall (i.e. hang) when it attempts to write data to a full TX buffer or read data from an empty RX buffer. This provides a very simple method of controlling the SPORT—placing the instruction that writes data to TX or reads data from RX in a loop. Program execution will stall at this instruction, until the SPORT is ready to transmit new data or has received new data.

Listing 9-1 on page 9-90 shows the code for this example, which sets up a loop to transmit data out of SPORT0. Although this technique provides a very simple programming solution, it prevents the processor’s core from handling any other tasks while waiting for the serial port. The interrupt-driven technique described in the following section alleviates this.
SPORT Programming Examples

Listing 9-1. SPORT transmit example code

/*
SPORT Transmit Example: Uses the feature that the processor core will
stall when attempting to write to a full TX register. This example
sets up a loop to transmit the data in the memory buffer source.
*/

#define N 8
#include "def21065L.h" /* Use symbolic register name */

.segment/dm dm32_b1; /* Data segment name described in ldf file */
.var source[N]= 0x11111111, 0x22222222, 0x33333333, 0x44444444
0x55555555, 0x66666666, 0x77777777, 0x88888888;
.endseg;

.segment/pm rst_svc; /* Reset vector from ldf file */
nop; /* First location is used for booting */
.jumpstart:
.endseg;

/* Main Routine */

.segment/pm pm48_1b0; /* Main code segment from ldf file. */

.start:r0=0x00270007; /* TDIV0 register: TCLKDIV=7,TFSDIV=39 */
.dm(TDIV0)=r0; /* sclock=CLKIN/8, framerate=sclock/20 */

.r0=0x000064f1; /* STCTL0 register */
.dm(STCTL0)=r0; /* SPEN=1, (SPORT enabled) */
/* SLEN=15, (16-bit word) */
/* ICLK=1, (internal tx clock) */
/* TFSR=1, (require TFS) */
/* ITFS=1, (internal TFS) */
/* DITFS=0, (data-depedent FS) */

.b0=source; /* Pointer to source; i0=b0 automatically */
.10=@source;
Serial Ports

Listing 9-2 shows the code for this example. Note that the interrupt used is the SPORT0 Transmit DMA Channel interrupt (SPT0I)—when serial port DMA is disabled, this interrupt becomes a single-word transmit interrupt.

Listing 9-2. SPORT interrupt-driven transmit example
/*
2106x Interrupt Driven SPORT Transmit Example
This example uses interrupts to notify the core when new data is required. The buffer “source” is transmitted.
/*
#define N 8
#include “def21065L.h”/*Use symbolic register names */
.segment/dm dm32_b1; /* Data segment name described in ldf file */
SPORT Programming Examples

```
.var source[N]= 0x11111111, 0x22222222, 0x33333333, 0x44444444
0x55555555, 0x66666666, 0x77777777, 0x88888888
.endseg;

.segment/pm rst_svc: /* Reset vector from ldf file. */
nop; /* First location is used for booting */
.jump start;
.endseg;

.segment/pm spt0_svc: /* SPORTO TX interrupt vector. */
.jump s0tx;
/* Main routine
.segment/pm pm48_1b0: /*Main code segment from ldf file */

.start: r0=0x00270007; /* TDIVO register: TCLKDIV=7,TFSDIV=39 */
dm(TDIV0)=r0; /* sclock=2CLKIN/8, framerate=sclock/20 */
r0=0x000064f1; /* STCTLO register */
dm((STCTLO)=r0 /* SPEN=1, (SPORT enabled) */
/* SLEN=15, (16-bit word) */
/* ICLK=1, (internal tx clock) */
/* TFSR=1, (require TFS) */
/* ITFS=1, (internal TFS) */
/* DITFS=0, (data dependent FS) */
b0=source; /* Pointer to source; i0=b0 automatically. */
l0=@source;

.bit set imask SPT0I;/* Enable SPORTO TX interrupt */
.bit set model IRPTN;

.r0=dm(i0,1); /* Write first value to TX0 to kick off SPORT */
dm((TX0_A)=r0;

.wait: idle; /* Wait for SPORTO TX interrupts. */
.jump wait;

/* SPORTO Transmit Interrupt Routine
.s0tx: rti (db);
r0=dm(i0,1); /* Get data from source buffer */
dm((TX0_A)=ro; /* Write transmit register */
.endseg;
/* */
```

This example shows how to use the processor’s on-chip DMA controller to handle serial port I/O. The DMA controller performs the data transfers between internal memory and the SPORTs, providing the most efficient way to handle input and output of multiple-word blocks of data. Once it has been set up, the DMA controller operates independently from the processor’s core. It interrupts core execution only when an entire block of data has been received (or transmitted). This frees the core to continue with other tasks.

Listing 9-3 shows the code for this example, which uses the serial port’s loopback mode. The program first sets up the SPORT1 DMA channels by loading values into the DMA parameter registers, then writes to the SRCTL1 and STCTL1 registers and waits to be interrupted.

Listing 9-3. SPORT DMA-driven loopback example

```c
/*
ADSP-21065L DMA-Driven SPORT Loopback Example:
This example sets up a SPORT DMA transfer and receive for serial port 1 in the loopback mode. The buffer "source" is DMAed out of the sport.
The loopback DMA programming mode internally attaches DT1, TFS1, and TCLK1 to DR1, RFS1, and RCLK1. The receive DMA places the data in the buffer "destination".
*/
#define N 8
#include "def21065L.h" /* Use symbolic register names */
.segment/dm dm32_b1; /* Data segment name described in ldf. file.*/
.var source[N]= 0x11111111, 0x22222222, 0x33333333, 0x44444444,
0x55555555, 0x66666666, 0x77777777, 0x88888888;
.var destination[N];
.endseg;
```
SPORT Programming Examples

```
.segment/pm rst_svc: /* Reset vector from ldf. file.*/
    nop: /* First location is used for booting.*/
    jump start:
.endseg;

.segment/pm spr1_svc; /* SPORT1 rx interrupt vector.*/
    jump s1rx:
.endseg;

* main routine *

.segment/pm pm48_1b0; /* Main code segment from ldf. file */
start:
    r0=source;
    dm(I16)=r0; /* Set DMA tx index to start of source buffer */
    r0=destination;
    dm(I12)=r0; /* Set DMA rx index to start of dest. buffer */
    r0=1;
    dm(IM6)=r0; /* Set DMA modify (stride) to 1. */
    dm(IM2)=r0;
    r0=@source;
    dm(C6)=r0; /* Set DMA count to length of data buffer */
    dm(C2)=r0;
    r0=0x004421f1; /* SRCTL1 Register: */
    dm(SRCTL1)=r0; /* SPEN=1, (SPORT1 enabled) */
    /* SLEN=31. (32-bit word) */
    /* RFSR=1. (require RFS) */
    /* SDEN=1, (rx DMA enable) */
    /* SPL=1, (loop back DT to DR & TFS to RFS) */
    r0=0x00270007; /* TDIV0 Register: TCLKDIV=7, TFSDIV=39 */
    dm(TDIV1)=r0; /* sclock=2CLKIN/8, framerate=sclock/2 0 */
    r0=0x000465f1; /* STCTL1 Register: */
    dm(STCTL1)=r0; /* SPEN=1, (SPORT1 enabled) */
    /* SLEN=31, (32-bit word) */
    /* ICLK=1, (internal tx clock) */
    /* TFSR=1, (require TFS) */
    /* ITFS=1, (internal TFS) */
    /* DITFS=0, (data dependent FS), all other bits=0 */
    /* SDEN=1, (tx dma enable), this kicks it off */
```
Serial Ports

bit set imask SPR1I; /* Enable SPORT1 rx interrupt */
bit set model IRPTEN; /* Global interrupt enable */

wait:idle; /* Wait for SPORT1 rx interrupt */
jump wait; /* Ends up here after entire DMA complete */

/*________________________SPORT1 Receive Interrupt Routine________________________*/
s1rx:rti; /* This interrupt will occur only once */
.endseg;

/*______________________________________________________________________________*/