TEXAS INSTRUMENTS

TMS320C54x Serial Ports

Addendum to the TMS320C54x User's Guide

User's Guide

1995 Digital Signal Processing Products

Texas Instruments

Printed in U.S.A., December 1995
D425005–9741 revision *

SPRU156

TMS320C54x Serial Ports User's Guide

Addendum to the TMS320C54x User's Guide

SPRU156
December 1995

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Preface

Read This First

The TMS320C54x is a fixed-point digital signal processor (DSP) in the TMS320 family. The purpose of this addendum to the TMS320C54x User's Guide is to serve as a reference for the TMS320C54x serial ports. Throughout this book, all references to the 'C54x apply to the TMS320C54x, as well as the TMS320LC54x and the TMS320VC54x, unless otherwise specified.

How to Use This User's Guide

The following table summarizes the 'C54x information contained in this addendum:

If you are looking for information about:	Turn to these sections:
Buffered serial port	Section 1.2, *Buffered Serial Port*
'C203 Serial Ports	Section 1.2, *Buffered Serial Port* Section 1.3, *Time-Division-Multiplexed (TDM) Serial Port*
'C209 Serial Ports	Section 1.1, *Synchronous Serial Port*
Synchronous serial port	Section 1.1, *Synchronous Serial Port*
TDM serial port	Section 1.3, *Time-Division-Multiplexed (TDM) Serial Port*

Related Documentation

The following books describe the ’C54x and related support tools. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477–8924. When ordering, please identify the book by its title and literature number.

TMS320C54x User’s Guide (literature number SPRU131) describes the TMS320C54x 16-bit, fixed-point, general-purpose digital signal processors. It describes the CPU architecture, memory organization, program control, instruction set, pipeline, and on-chip peripherals. Software and hardware applications are covered in dedicated chapters.

TMS320C54x Optimizing C Compiler User’s Guide (literature number SPRU103) describes the ’C54x C compiler. This C compiler accepts ANSI standard C source code and produces TMS320 assembly language source code for the ’C54x generation of devices.

TMS320C5xx C Source Debugger User’s Guide (literature number SPRU099) tells you how to invoke the ’C54x emulator, EVM, and simulator versions of the C source debugger interface. This book discusses various aspects of the debugger interface, including window management, command entry, code execution, data management, and breakpoints. It also includes a tutorial that introduces basic debugger functionality.

TMS320C54x Assembly Language Tools User’s Guide (literature number SPRU102) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the ’C54x generation of devices.

TMS320 Third-Party Support Reference Guide (literature number SPRU052) alphabetically lists over 100 third parties that provide various products that serve the family of ’320 digital signal processors. A myriad of products and applications are offered—software and hardware development tools, speech recognition, image processing, noise cancellation, modems, etc.

TMS320 Family Development Support Reference Guide (literature number SPRU011) describes the ’320 family of digital signal processors and covers the various products that support this product line. This includes code-generation tools (compilers, assemblers, linkers, etc.) and system integration and debug tools (simulators, emulators, evaluation modules, etc.). Also covered are available documentation, seminars, the university program, and factory repair and exchange.

Style and Symbol Conventions

This document uses the following conventions.

- Program listings and program examples are shown in a `special typeface` similar to a typewriter's typeface.

  Here is a segment of a program listing:

```
STL    A,*AR1+      ;Int_RAM(I)=0
RSBX   INTM         ;Globally enable interrupts
B      MAIN_PG      ;Return to foreground program
```

- In syntax descriptions, the instruction is in **bold typeface** font and parameters are in *italic typeface*. Portions of a syntax in **bold** should be entered as shown; portions of a syntax in *italics* describe the type of information that you specify. Here is an example of an instruction syntax:

  [*label*] **LMS** *Xmem*, *Ymem*

  LMS is the instruction, which has two parameters indicated by *Xmem*, and *Ymem*. When you use **LMS**, the parameters should be actual dual data-memory operand values. A comma and a space must separate the two values.

- In the assembly language instructions, when the word "or" is capitalized, it denotes a boolean operation. When it is lowercased it indicates selection. Here is an example of an instruction with OR and or:

  lk OR (src) → src or [,dst]

  This instruction ORs the value of lk with the contents of src. Then, it stores the result in src or dst depending on the syntax of the instruction.

- Square brackets (**[** and **]**) identify an optional parameter. If you use an optional parameter, specify the information within the brackets; do not type the brackets themselves. In the example above, instead of typing [*label*], specify a name for the label. When you specify more than one optional parameter from a list, separate them with a comma and a space.

Technical Articles

A wide variety of related documentation is available on digital signal processing. These references fall into one of the following application categories:

- General-Purpose DSP
- Graphics/Imagery
- Speech/Voice
- Control
- Multimedia
- Military
- Telecommunications
- Automotive
- Consumer
- Medical
- Development Support

In the following list, references appear in alphabetical order according to author. The documents contain beneficial information regarding designs, operations, and applications for signal-processing systems; all of the documents provide additional references. Texas Instruments strongly suggests that you refer to these publications.

***General-Purpose DSP**:*

1) Chassaing, R., Horning, D.W., *"Digital Signal Processing with Fixed and Floating-Point Processors"* , CoED, USA, Volume 1, Number 1, pages 1–4, March 1991.

2) Defatta, David J., Joseph G. Lucas, and William S. Hodgkiss, *Digital Signal Processing: A System Design Approach*, New York: John Wiley, 1988.

3) Erskine, C., and S. Magar, "Architecture and Applications of a Second-Generation Digital Signal Processor," *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, USA, 1985.

4) Essig, D., C. Erskine, E. Caudel, and S. Magar, "A Second-Generation Digital Signal Processor," *IEEE Journal of Solid-State Circuits*, USA, Volume SC–21, Number 1, pages 86–91, February 1986.

5) Frantz, G., K. Lin, J. Reimer, and J. Bradley, "The Texas Instruments TMS320C25 Digital Signal Microcomputer," *IEEE Microelectronics*, USA, Volume 6, Number 6, pages 10–28, December 1986.

6) Gass, W., R. Tarrant, T. Richard, B. Pawate, M. Gammel, P. Rajasekaran, R. Wiggins, and C. Covington, "Multiple Digital Signal Processor Environment for Intelligent Signal Processing," *Proceedings of the IEEE, USA*, Volume 75, Number 9, pages 1246–1259, September 1987.

7) Jackson, Leland B., *Digital Filters and Signal Processing*, Hingham, MA: Kluwer Academic Publishers, 1986.

8) Jones, D.L., and T.W. Parks, *A Digital Signal Processing Laboratory Using the TMS32010*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1987.

9) Lim, Jae, and Alan V. Oppenheim, *Advanced Topics in Signal Processing*, Englewood Cliffs, NJ: Prentice- Hall, Inc., 1988.

10) Lin, K., G. Frantz, and R. Simar, Jr., "The TMS320 Family of Digital Signal Processors," *Proceedings of the IEEE*, USA, Volume 75, Number 9, pages 1143–1159, September 1987.

11) Lovrich, A., Reimer, J., *"An Advanced Audio Signal Processor"* , Digest of Technical Papers for 1991 International Conference on Consumer Electronics, June 1991.

12) Magar, S., D. Essig, E. Caudel, S. Marshall and R. Peters, "An NMOS Digital Signal Processor with Multiprocessing Capability," *Digest of IEEE International Solid-State Circuits Conference*, USA, February 1985.

13) Oppenheim, Alan V., and R.W. Schafer, *Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1975 and 1988.

14) Papamichalis, P.E., and C.S. Burrus, "Conversion of Digit-Reversed to Bit-Reversed Order in FFT Algorithms," *Proceedings of ICASSP 89*, USA, pages 984–987, May 1989.

15) Papamichalis, P., and R. Simar, Jr., "The TMS320C30 Floating-Point Digital Signal Processor," *IEEE Micro Magazine*, USA, pages 13–29, December 1988.

16) Papamichalis, P.E., "FFT Implementation on the TMS320C30," *Proceedings of ICASSP 88*, USA, Volume D, page 1399, April 1988.

17) Parks, T.W., and C.S. Burrus, *Digital Filter Design*, New York, NY: John Wiley and Sons, Inc., 1987.

18) Peterson, C., Zervakis, M., Shehadeh, N., *"Adaptive Filter Design and Implementation Using the TMS320C25 Microprocessor"* , Computers in Education Journal, USA, Volume 3, Number 3, pages 12–16, July–September 1993.

19) Prado, J., and R. Alcantara, "A Fast Square-Rooting Algorithm Using a Digital Signal Processor," *Proceedings of IEEE*, USA, Volume 75, Number 2, pages 262–264, February 1987.

20) Rabiner, L.R. and B. Gold, *Theory and Applications of Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1975.

21) Simar, Jr., R., and A. Davis, “The Application of High-Level Languages to Single-Chip Digital Signal Processors,” *Proceedings of ICASSP 88*, USA, Volume D, page 1678, April 1988.

22) Simar, Jr., R., T. Leigh, P. Koeppen, J. Leach, J. Potts, and D. Blalock, “A 40 MFLOPS Digital Signal Processor: the First Supercomputer on a Chip,” *Proceedings of ICASSP 87*, USA, Catalog Number 87CH2396–0, Volume 1, pages 535–538, April 1987.

23) Simar, Jr., R., and J. Reimer, “The TMS320C25: a 100 ns CMOS VLSI Digital Signal Processor,” *1986 Workshop on Applications of Signal Processing to Audio and Acoustics*, September 1986.

24) Texas Instruments, *Digital Signal Processing Applications with the TMS320 Family*, 1986; Englewood Cliffs, NJ: Prentice-Hall, Inc., 1987.

25) Treichler, J.R., C.R. Johnson, Jr., and M.G. Larimore, *A Practical Guide to Adaptive Filter Design*, New York, NY: John Wiley and Sons, Inc., 1987.

***Graphics/Imagery**:*

1) Reimer, J., and A. Lovrich, “Graphics with the TMS32020,” *WESCON/85 Conference Record*, USA, 1985.

***Speech/Voice**:*

1) DellaMorte, J., and P. Papamichalis, “Full-Duplex Real-Time Implementation of the FED-STD-1015 LPC-10e Standard V.52 on the TMS320C25,” *Proceedings of SPEECH TECH 89*, pages 218–221, May 1989.

2) Gray, A.H., and J.D. Markel, *Linear Prediction of Speech*, New York, NY: Springer-Verlag, 1976.

3) Frantz, G.A., and K.S. Lin, “A Low-Cost Speech System Using the TMS320C17,” *Proceedings of SPEECH TECH ’87*, pages 25–29, April 1987.

4) Papamichalis, P., and D. Lively, “Implementation of the DOD Standard LPC–10/52E on the TMS320C25,” *Proceedings of SPEECH TECH ’87*, pages 201–204, April 1987.

5) Papamichalis, Panos, *Practical Approaches to Speech Coding*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1987.

6) Pawate, B.I., and G.R. Doddington, “Implementation of a Hidden Markov Model-Based Layered Grammar Recognizer,” *Proceedings of ICASSP 89*, USA, pages 801–804, May 1989.

7) Rabiner, L.R., and R.W. Schafer, *Digital Processing of Speech Signals*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1978.

8) Reimer, J.B. and K.S. Lin, "TMS320 Digital Signal Processors in Speech Applications," *Proceedings of SPEECH TECH '88*, April 1988.

9) Reimer, J.B., M.L. McMahan, and W.W. Anderson, "Speech Recognition for a Low-Cost System Using a DSP," *Digest of Technical Papers for 1987 International Conference on Consumer Electronics*, June 1987.

***Control**:*

1) Ahmed, I., "16-Bit DSP Microcontroller Fits Motion Control System Application," *PCIM*, October 1988.

2) Ahmed, I., "Implementation of Self Tuning Regulators with TMS320 Family of Digital Signal Processors," *MOTORCON '88*, pages 248–262, September 1988.

3) Allen, C. and P. Pillay, *"TMS320 Design for Vector and Current Control of AC Motor Drives"*, Electronics Letters, UK, Volume 28, Number 23, pages 2188–2190, November 1992.

4) Panahi, I. and R. Restle, *"DSPs Redefine Motion Control"*, Motion Control Magazine, December 1993.

5) Lovrich, A., G. Troullinos, and R. Chirayil, "An All-Digital Automatic Gain Control," *Proceedings of ICASSP 88*, USA, Volume D, page 1734, April 1988.

6) Ahmed, I., and S. Meshkat, "Using DSPs in Control," *Control Engineering*, February 1988.

7) Meshkat, S., and I. Ahmed, "Using DSPs in AC Induction Motor Drives," *Control Engineering*, February 1988.

8) Matsui, N. and M. Shigyo, *"Brushless DC Motor Control Without Position and Speed Sensors"*, IEEE Transactions on Industry Applications, USA, Volume 28, Number 1, Part 1, pages 120–127, January–February 1992.

9) Hanselman, H., "LQG-Control of a Highly Resonant Disc Drive Head Positioning Actuator," *IEEE Transactions on Industrial Electronics*, USA, Volume 35, Number 1, pages 100–104, February 1988.

10) Bose, B.K., and P.M. Szczesny, "A Microcomputer-Based Control and Simulation of an Advanced IPM Synchronous Machine Drive System for Electric Vehicle Propulsion," *Proceedings of IECON '87*, Volume 1, pages 454–463, November 1987.

11) Ahmed, I., and S. Lindquist, “Digital Signal Processors: Simplifying High-Performance Control,” *Machine Design*, September 1987.

***Multimedia**:*

1) Reimer, J., *“DSP-Based Multimedia Solutions Lead Way Enhancing Audio Compression Performance”* , Dr. Dobbs Journal, December 1993.

2) Reimer, J., G. Benbassat, and W. Bonneau Jr., *“Application Processors: Making PC Multimedia Happen”* , Silicon Valley PC Design Conference, July 1991.

***Military**:*

1) Papamichalis, P., and J. Reimer, “Implementation of the Data Encryption Standard Using the TMS32010,” *Digital Signal Processing Applications*, 1986.

***Telecommunications**:*

1) Ahmed, I., and A. Lovrich, “Adaptive Line Enhancer Using the TMS320C25,” *Conference Records of Northcon/86*, USA, 14/3/1–10, September/October 1986.

2) Casale, S., R. Russo, and G. Bellina, “Optimal Architectural Solution Using DSP Processors for the Implementation of an ADPCM Transcoder,” *Proceedings of GLOBECOM ’89*, pages 1267–1273, November 1989.

3) Cole, C., A. Haoui, and P. Winship, “A High-Performance Digital Voice Echo Canceller on a SINGLE TMS32020,” *Proceedings of ICASSP 86*, USA, Catalog Number 86CH2243–4, Volume 1, pages 429–432, April 1986.

4) Cole, C., A. Haoui, and P. Winship, “A High-Performance Digital Voice Echo Canceller on a Single TMS32020,” *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, USA, 1986.

5) Lovrich, A., and J. Reimer, “A Multi-Rate Transcoder,” *Transactions on Consumer Electronics*, USA, November 1989.

6) Lovrich, A. and J. Reimer, *“A Multi-Rate Transcoder”* , Digest of Technical Papers for 1989 International Conference on Consumer Electronics, June 7–9, 1989.

7) Lu, H., D. Hedberg, and B. Fraenkel, “Implementation of High-Speed Voiceband Data Modems Using the TMS320C25,” *Proceedings of ICASSP 87*, USA, Catalog Number 87CH2396–0, Volume 4, pages 1915–1918, April 1987.

8) Mock, P., “Add DTMF Generation and Decoding to DSP– µP Designs,” *Electronic Design*, USA, Volume 30, Number 6, pages 205–213, March 1985.

9) Reimer, J., M. McMahan, and M. Arjmand, “ADPCM on a TMS320 DSP Chip,” *Proceedings of SPEECH TECH 85*, pages 246–249, April 1985.

10) Troullinos, G., and J. Bradley, “Split-Band Modem Implementation Using the TMS32010 Digital Signal Processor,” *Conference Records of Electro/86 and Mini/Micro Northeast*, USA, 14/1/1–21, May 1986.

***Automotive**:*

1) Lin, K., “Trends of Digital Signal Processing in Automotive,” *International Congress on Transportation Electronic (CONVERGENCE ’88*), October 1988.

***Consumer**:*

1) Frantz, G.A., J.B. Reimer, and R.A. Wotiz, “Julie, The Application of DSP to a Product,” *Speech Tech Magazine*, USA, September 1988.

2) Reimer, J.B., and G.A. Frantz, “Customization of a DSP Integrated Circuit for a Customer Product,” *Transactions on Consumer Electronics*, USA, August 1988.

3) Reimer, J.B., P.E. Nixon, E.B. Boles, and G.A. Frantz, “Audio Customization of a DSP IC,” *Digest of Technical Papers for 1988 International Conference on Consumer Electronics*, June 8–10 1988.

***Medical**:*

1) Knapp and Townshend, “A Real-Time Digital Signal Processing System for an Auditory Prosthesis,” *Proceedings of ICASSP 88*, USA, Volume A, page 2493, April 1988.

2) Morris, L.R., and P.B. Barszczewski, “Design and Evolution of a Pocket-Sized DSP Speech Processing System for a Cochlear Implant and Other Hearing Prosthesis Applications,” *Proceedings of ICASSP 88*, USA, Volume A, page 2516, April 1988.

***Development Support**:*

1) Mersereau, R., R. Schafer, T. Barnwell, and D. Smith, “A Digital Filter Design Package for PCs and TMS320,” *MIDCON/84 Electronic Show and Convention*, USA, 1984.

2) Simar, Jr., R., and A. Davis, “The Application of High-Level Languages to Single-Chip Digital Signal Processors,” *Proceedings of ICASSP 88*, USA, Volume 3, pages 1678–1681, April 1988.

If You Need Assistance. . .

If you want to. . .	Do this. . .
Order Texas Instruments documentation	Call the Literature Response Center: **(800) 477–8924**
Obtain technical support, report suspected problems	Call the DSP hotline: **(713) 274–2320** Or send a FAX: **(713) 274–2324** **+33–1–3070–1032** (Europe) Or send email to: **4389750@mcimail.com.**
Obtain TI product updates, application software	Dial the TMS320 Bulletin Board Service (BBS): **(713) 274–2323** (24 hrs.) **+44–2–3422–3248** (Europe) Set your modem to 8 bits,1 stop bit, no parity. Supported speeds are from 300 to 14400 bps.
Access the TMS320 BBS from Internet	Connect via anonymous ftp to: **ftp.ti.com** (192.94.94.5), subdirectory /pub/mirrors
Report mistakes or offer suggestions regarding this document or any other TI documentation	Fill out and return the reader response card at the end of this book Or send your comments to: **Texas Instruments Incorporated** **Technical Publications Manager, MS 702** **P.O. Box 1443** **Houston, Texas 77251–1443** or send email to: **comments@books.sc.ti.com**

Contents

Figures

Tables

Examples

Chapter 1

Serial Ports

This chapter describes the serial port interfaces available in the ’C54x: synchronous serial port, buffered serial port (BSP), and time-division-multiplexed (TDM) serial port. The ’C54x devices contain a combination of serial port interfaces:

- The ’C541 devices contain two synchronous serial ports.
- The ’C542 and the ’C543 devices contain one buffered serial port and one TDM serial port.

Topic **Page**

1.1 Synchronous Serial Port

A full duplex (bidirectional) on-chip serial port provides direct communication with serial devices such as codecs, serial A/D (analog-to-digital) converters, and other serial systems. The serial port can also be used for intercommunication between processors in multi-processor systems.

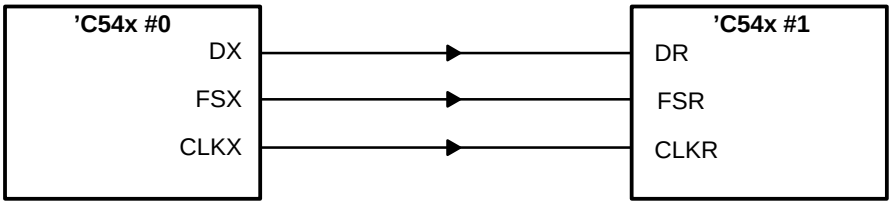
Both receive and transmit operations are double-buffered on the ’C54x, thus allowing a continuous communication stream (either 8- or 16-bit data packets). The continuous mode provides operation that once initiated requires no further frame synchronization pulses when transmitting at maximum packet frequency. The serial port is fully static and thus functions at arbitrarily low clocking frequencies. The maximum operating frequency of the serial port while using internal clocks is CLKOUT1/4 (5 Mbps at 50ns, 12.5 Mbps at 20ns). When the serial ports are in reset the device can be configured to shut off the serial port internal clocks, allowing the device to run in a lower power mode of operation.

1.1.1 Serial Port Operation

Three signals are necessary to connect two devices for data transmission. On the transmitting device, the transmit data signal (DX) sends the data, the transmit frame synchronization signal (FSX) initiates the transfer at the beginning of the packet, and the transmit clock signal (CLKX) clocks the bit transfer. The corresponding pins on the receiving device are DR, FSR and CLKR, respectively. Figure 1–1 shows these pins for two ’C54x serial ports connected for a one-way transfer from device 0 to device 1.

At reset, CLKX, CLKR, DR, FSX, and FSR become inputs and DX is set high-impedance.

Figure 1–1. One-Way Serial Port Transfer



Legend:

DX	Transmit data	**DR**	Receive data
FSX	Transmit frame synchronization	**FSR**	Receive frame synchronization
CLKX	Transmit clock	**CRKR**	Receive clock

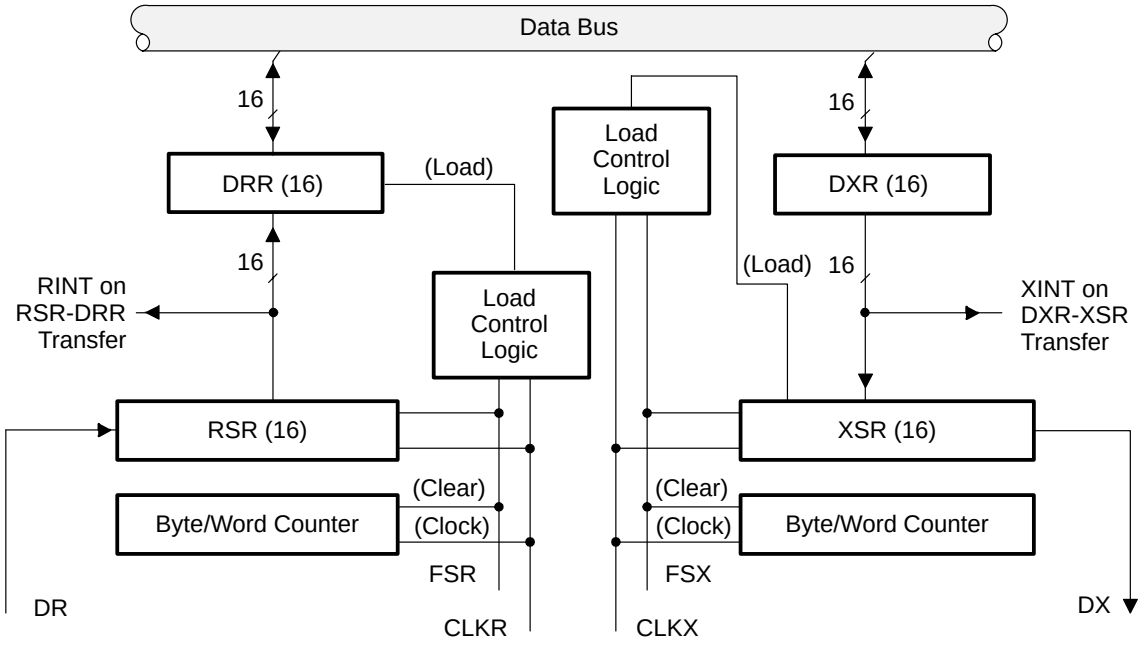
The serial port operates through three memory-mapped registers: the serial port control register (SPC), the data transmit register (DXR), and the data receive register (DRR). Additionally, the port uses two unmapped registers that permit double-buffering capability: the transmit shift register (XSR) and the receive shift register (RSR).

Table 1–1. Serial Port Registers

Registers	Description
SPC	Serial port control register
DXR	Data transmit register
DRR	Data receive register
XSR	Transmit shift register
RSR	Receive shift register

Figure 1–2 shows how the pins and registers are configured on the serial port and how the double buffering is implemented.

Figure 1–2. Serial Port Block Diagram



The SPC controls serial port operation; the functions of SPC bit fields are described in Figure 1–2. Transmit data is written to the DXR, while received data is read from the DRR. A transmit is executed by writing data to the DXR, which copies the data to XSR when the XSR is empty (the last word has been serially transmitted, that is, driven on the DX pin). The XSR manages the shifting of the data to the DX pin, thus allowing another write to DXR as soon as the DXR-to-XSR copy is completed.

Upon completion of the the DXR-to-XSR copy, a 0-to-1 transition occurs on the transmit ready XRDY bit in the SPC and generates a serial port transmit interrupt that signals that DXR is ready for a new word. The process is similar on the receive side. Data from the DR pin is shifted into the RSR, which copies it to the data receive register (DRR) from which it can be read. Upon completion of the RSR-to-DRR copy, a 0-to-1 transition occurs on the receive ready (RRDY) bit in the SPC and generates a serial port receive interrupt (RINT). Thus, the serial port is double-buffered because data can be transferred to or from DXR or DRR while another transmit or receive is being performed. Note that the transfer timing is synchronized by the frame sync pulse in burst mode and is discussed in more detail in subsection 1.1.2.

Figure 1–3 shows the 16-bit memory-mapped register that configures the serial port. Some of the bits are read-only while others are read/write.

Figure 1–3. Serial Port Control Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FREE	SOFT	RSRFULL	$\overline{\text{XSREMPTY}}$	XRDY	RRDY	IN1	IN0	RRST	XRST	TXM	MCM	FCM	FO	DLB	RES
R/W	R/W	R	R	R	R	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R

Note: R=Read, W=Write

Table 1–2. Serial Port Control Register Bits Summary

Bit	Name	Function
0	Reserved	Always read as 0.
1	DLB	The digital loopback mode bit can be used to put the serial port in digital loopback mode. When DLB=1, DR and FSR are connected to DX and FSX, respectively, through multiplexers, as shown in Figure 1–4. Additionally, CLKR is driven by CLKX if MCM=1. If DLB=1 and MCM=0, CLKR is taken from the CLKR pin of the device. This configuration allows CLKX and CLKR to be tied together externally and supplied by a common external clock source. If DBL=0, DR, FSR, and CLKR are taken from the respective device pins. Note that TXM must be set to 1 for proper operation in DLB mode. Note also that the FSX and DX signals appear on the device pins when DLB=1, but FSR and DR do not.
2	FO	The format bit specifies the word length of the serial port transmitter and receiver. If FO=0, data is transmitted and/or received as 16-bit words. If FO=1, data is transferred as 8-bit bytes. The data is transferred with the MSB first.
3	FSM	The frame synch mode bit specifies whether frame synchronization pulses are required for serial port operation. If FSM=1, a frame sync pulse is required on FSX/FSR for the transmission/reception of each word. When the serial port is operated in the continuous mode, FSM=0. Refer to subsection 1.1.2 for more details on frame sync signals.
4	MCM	The clock mode bit specifies the clock source for CLKX. If MCM=0, CLKX is taken from the CLKX pin. If MCM=1, CLKX is driven by an on-chip clock source having a frequency equal to one–fourth of CLKOUT. Note that if MCM=1 and DLB=1, a CLKR signal is also supplied by the internal source.

Table 1–2. Serial Port Control Register Bits Summary (Continued)

Bit	Name	Function
5	TXM	The transmit mode bit configures the FSX pin as an input (TXM=0) or as an output (TXM=1). When TXM=1, frame sync pulses are generated internally when data is transferred from the DXR to XSR to initiate data transfers. The internally generated framing signal is synchronous with respect to CLKX. When TXM=0, the transmitter idles until a frame synch pulse is supplied on the FSX pin.
6	XRST RRST	The transmit reset and receive reset signals reset the transmitter and receiver, respectively. If the SPC is to be modified to reconfigure the serial port, a total of two writes should be made to the SPC. The first write should write 0s to XRST and RRST and the desired configuration to bits 1–5. The second write should write 1s to XRST and RRST, taking the serial port out of reset. When a 0 is written to either of these bits, activity in the corresponding section of the serial port halts. Note that when XRDY=0, writing a 0 to XRST generates a transmit interrupt. When XRST=0,RRST=0, and MCM=0, the internal clocks to the serial ports are shut off, allowing the device to run in a lower power mode of operation.
8 9	IN0 IN1	The input 0 bit and input 1 bit allow the CLKR and CLKX pins to be used as bit inputs. IN0 and IN1 reflect the current levels of the CLKR and CLKX pins, respectively, of the device. The levels on these pins can be read through the SPC. They can be tested by using the BIT, BITF, or BITT instruction. Note that there is a latency of between 0.5 and 1.5 CLKOUT cycles in length from CLKR/CLKX switching to the new CLKR/CLKX value being represented in the SPC.
10 11	RRDY XRDY	Receive ready and transmit ready bits. A transition from 0 to 1 of the RRDY bit indicates that the receive shift register (RSR) has been copied to the DRR and that the data can be read. A receive interrupt is generated upon the transition. A transition from 0 to 1 of the XRDY bit indicates that the DXR contents have been copied to the XSR and that data is ready to be loaded with a new data word. A transmit interrupt is generated upon the transition. These bits can be polled in software in lieu of using serial port interrupts.
12	$\overline{\text{XSREMPTY}}$	The transmit shift register empty flag. This bit indicates whether the transmitter has experienced underflow. Underflow occurs when two conditions are satisfied: 1) the XSR empties, and 2) the DXR has not been reloaded since the last DXR-to-XSR transfer. Note that underflow does not constitute an error condition in burst mode. If another frame synch pulse occurs before writing the DXR, while in burst mode, the previous data in the XSR is shifted out the DX pin. Writing to DXR inactivates the $\overline{\text{XSREMPTY}}$ bit =0 indicates underflow.
13	RSRFUL	The receive shift register full flag. This bit indicates whether the receive has experienced overrun. Overrun occurs when three conditions are satisfied: 1) RSR is full, 2) the DRR has not been read since the last RSR-to-DRR transfer, and 3) a frame sync pulse appears on FSR. Note that condition 3 applies only when FSM=1. When FSM=0, only the first two conditions apply. When RSRFUL=1, the receiver halts and waits for the DRR to be read. The data in the RSR is preserved, but any data sent on DR while the receiver is halted, is lost. Reading DRR, device reset, and serial port reset each clear the RSRFUL bit. RSRFUL=1 indicates overflow.

Table 1–2. Serial Port Control Register Bits Summary (Continued)

Bit	Name	Function
14	SOFT	The soft bit. This bit is enabled when the FREE bit is 0. If FREE=0, the SOFT bit selects immediate stop, if 0, or stop after word completion if 1. SOFT=0 upon reset.
15	FREE	The free run bit. If FREE=1, free run is selected, regardless of the value of the SOFT bit. If FREE=0, the SOFT bit selects the emulation mode as described above. FREE=0 upon reset.

Bit 0 is reserved and is read as 0. The format bit FO, bit 1 of the SPC, specifies whether data is transmitted as 16-bit words (FO=0) or 8-bit bytes (FO=1). Note that in the latter case, only the lower byte of whatever is written to DXR on the transmitter is transmitted and the lower byte of whatever is read from DRR on the receiver is received. To transmit a whole 16-bit word in b-bit byte mode on the transmitter, two writes to DXR are necessary, with the appropriate shifts of the value because the upper 8 bits written to DXR are ignored. Similarly, to receive a whole 16-bit word in 8-bit mode on the receiver, two reads from DRR are necessary, with the appropriate shifts of the value, because the upper 8 bits in DRR are random values.

The source device for the clock for serial port transfers is set by bit 4 (MCM) of the SPC register. If MCM=1 then CLKX is configured as an output and is driven by an internal clock source with a frequency equal to 1/4 of CLKOUT. If MCM=0, CLKX is configured as an input and thus accepts an external clock. Note that the CLKR pin is always configured as an input.

The source device for the frame synchronization pulse is set with the TXM bit, (bit5). Like MCM, if TXM=1, the FSX pin is configured as an output and drives a pulse at the beginning of every transmit. If TXM=0, FSX is configured as an input and accepts an external frame sync signal. Note that the FSR pin is always configured as an input.

The reset of the serial port for both transmitter and receiver is done by the XRST bit and RRST bit, bit 6 and 7, respectively. These signals are active low, so that if XRST= RRST= 0, the serial port is in reset. To modify SPC to configure the serial port, a total of two writes to the SPC are necessary. The first write should write zeros to the XRST and RRST and the desired configuration bits 1–5. While maintaining the desired configuration bits, the second write should write ones to XRST, and RRST, bits, taking the serial port out of reset. Note that these bits can be reset individually if desired. When a zero is written to either of these bits, activity in the corresponding section of the serial port stops. When XRST=0 and RRST=0, the particular internal clocks to the serial port are shut off. This minimizes the switching and allows the device to operate on lower

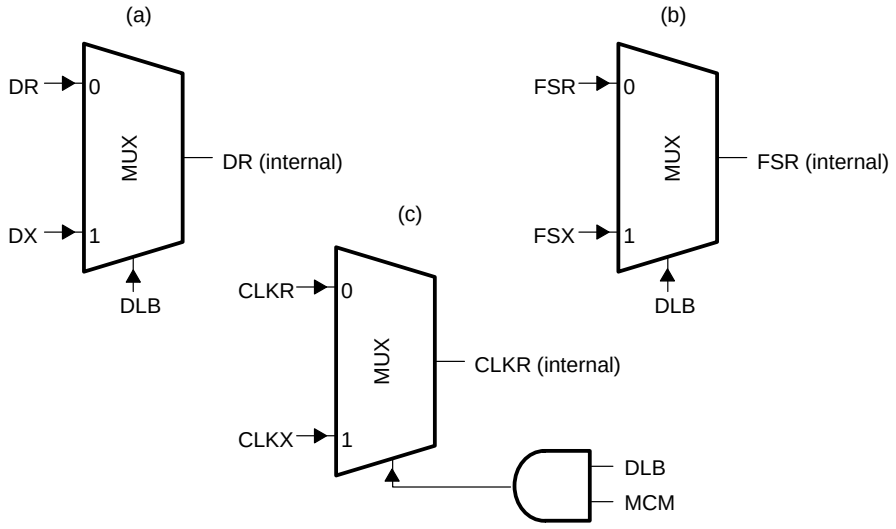
power consumption (as long as the CLKX bit is configured as an input, that is, with MCM=0.

The FSM bit (bit 3) specifies whether frame syncs are needed in consecutive serial port transmits. If FSM=1, a frame sync is required for every transfer and the mode is referred to as **burst mode**, because there can be periods of inactivity on the serial port between transmits. the frequency of packet writes to DXR is called packet frequency. The packets can be 8 or 16 bits long, depending on FO.

As the packet frequency increases, it reaches a maximum that is equivalent to 8 or 16 clock cycles, depending on FO. Note that this cycle count corresponds to 32 or 64 instruction cycles on CPU, again depending on FO if internal 'C54x clocks are used. Thus, if transmitting at maximum rate for more than one transmission, the frame sync signal becomes extraneous. The **continuous mode** of operation (FSM=1) is then the mode that requires only an initial frame sync pulse, as long as a write to DXR for transmit, or a read from DRR for receive, is executed during each transmission. the timing of both modes is discussed in detail in subsections 1.1.2 and 1.1.3.

The DLB bit, (bit 1) is a digital loop back mode that allows testing of the serial port code with just one device. When DLB=1, DR and FSR are connected to DX and FSX, respectively, through multiplexers, as shown in Figure 1–4.

Figure 1–4. Receiver Signal MUXes



CLKR is driven by CLKX if MCM=1. But if MCM=0 while DLB=1, then CLKR is taken from the CLKR pin. This allows for external clock generation of these signals during digital loopback mode. If DLB=0, then normal operation occurs where DR, FSR, and CLKR are all taken from their respective pins.

Bits 10–13 in the SPC are read-only status bits that indicate various states in serial port operation. Writes and reads to the serial port can be synchronized by polling RRDY and XRDY, (bits 10 and 11, respectively) or by using the interrupts that they generate. A transition from 0 to 1 of the RRDY bit indicates that the RSR has been copied to the DRR and that the received data can be read. A receive interrupt (RINT) is generated upon this transition. A transition from 0 to 1 of the XRDY bit indicates that the DXR contents have been copied to the XSR and that DXR is ready to be loaded with a new data word. A transmit interrupt (XINT) is generated upon this transition. Polling these bits in software can either substitute for or complement the use of serial port interrupts. In other words, both polling and interrupts can be used together, if so desired. $\overline{\text{XSREMPTY}}$ (bit 12) indicates whether the transmitter has experienced underflow. (When =0, it is active).

The following three situations cause the flag to become active:

- ❑ DXR has not been loaded since the last DXR-XSR transfer and XSR empties. (The actual transition of occurs after the last bit has been shifted out of XSR.)
- ❑ Serial port is reset (XRST=0).
- ❑ Device is reset.

When $\overline{\text{XSREMPTY}}$ is active, the transmit side of the serial port halts, thus driving no value (the DX pin is in the high-impedance state). An exception occurs in burst mode with external frame syncs, as explained in subsection 1.1.2. Note that underflow is not an error condition in burst mode, although it is in the continuous mode (error conditions are further discussed in subsection 1.1.4). The $\overline{\text{XSREMPTY}}$ flag becomes inactive ($\overline{\text{XRSEMPTY}}$=1) when:

- ❑ A write to DXR occurs.
- ❑ The RSRFUL bit, (bit 13) indicates whether the receiver has experienced overrun. (When RSRFULL=1, it is active.)

Overrun occurs when:

- ❑ The DRR has not been read since the last RSR-to-DRR transfer.
- ❑ RSR is full.
- ❑ A frame sync pulse appears on FSR.

Note that in continuous mode (FSM=0), only the first two conditions apply; therefore, RSRFULL transitions after the last bit has been shifted out. When RSRFULL=1, the receiver halts and waits for DRR to be read. The data in RSR is preserved, but any new data driven on the DR pin while the receiver is halted is lost.

The RSRFULL flag becomes inactive (RSRFULL=0) under the following three conditions:

- DRR is read.
- Serial port is reset (RRST=0).
- Device is reset.

IN0 and IN1 (bit 8 and 9) in the SPC allow the CLKR and CLKX pins to be used as bit inputs. IN0 and IN1 reflect the current levels of the CLKR and CLKX pins. The levels on the pins can be read by reading the SPC. they can be tested by using the BIT, BITF or BITT instructions. Note that there is a latency of between 0.5 and 1.5 CLKOUT cycles in length from CLKR/CLKX switching to the new CLKR/CLKX value being represented in the SPC. Note that if the serial port is put into reset, IN0 and IN1 can be used as bit inputs and DRR and DXR as general–purpose registers.

SOFT and FREE (bits 14 and 15, respectively) are special emulation bits that determine the state of the serial port clock when a breakpoint is encountered in the high-level language debugger. If the FREE bit is set to 1, then, upon a software breakpoint, the clock continues to run (that is, free runs) and data is shifted out. In this case, SOFT is a *don't care*. But if FREE is 0, then SOFT takes effect. If SOFT=0, then the clock immediately stops, thus aborting any transmission. If the SOFT bit is 1, the particular transmission continues until completion of the word, and then the clock halts. The options are as follows:

FREE	SOFT	
1	X	Free run
0	0	Immediate stop
0	1	Stop after completion of word

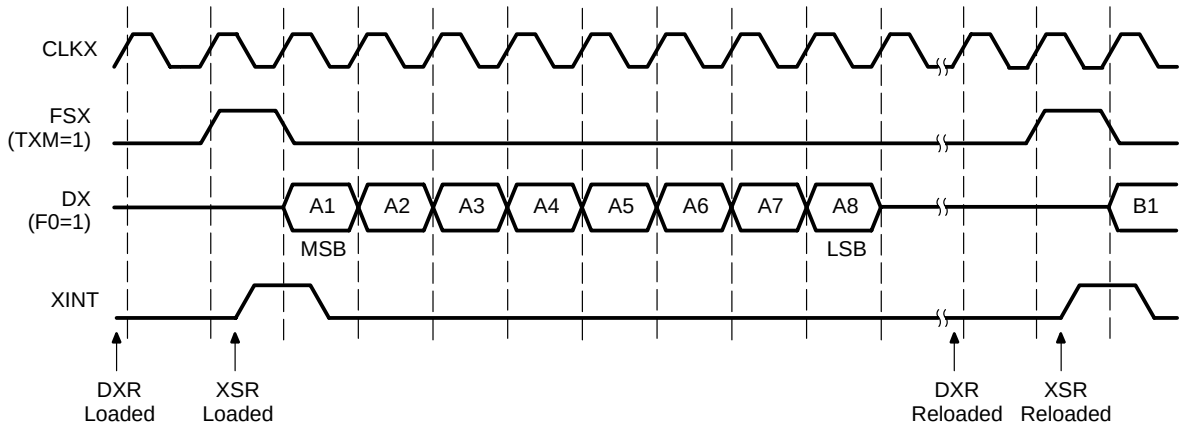
The receive side functions in a similar fashion. Note that if an option besides *immediate stop* is chosen, the receiver continues running and an overflow error is possible. The default value for these bits is *immediate stop*.

1.1.2 Transmit and Receive Operations (Burst Mode)

In burst mode operation, there are periods of serial port inactivity between packet transmits. The data packet is marked by the frame sync pulse on FSX. On the transmit device, the transmission is initiated by a write to DXR. The value in DXR is shifted to XSR; upon a frame sync pulse on FSX (generated internally or externally depending on TXM), the value in XSR is shifted out and driven on the DX pin. If DXR is reloaded before the old DXR contents have been transferred to XSR, the old DXR contents are overwritten. The DXR is

copied to the XSR only if the XSR is empty and the DXR has been loaded since the last DXR to XSR transfer. The DXR should be written to only if XRDY=1, which is guaranteed if the DXR write is made in response to a transmit interrupt or polling XRDY. The timing for the serial port transmit is shown in Figure 1–5.

Figure 1–5. Burst-Mode Serial Port Transmit Operation



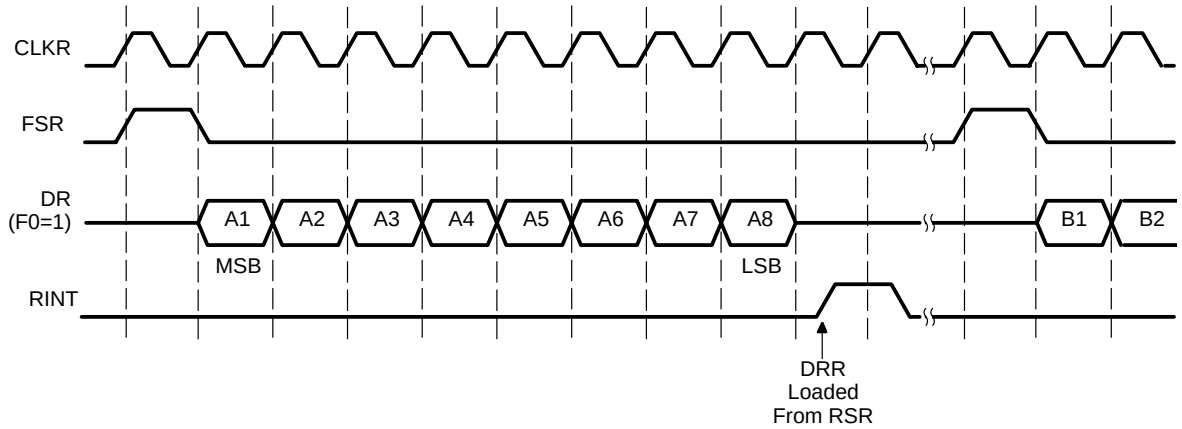
Note in the following discussion that the timings are slightly different for internally (TMX=1, FSX is an output) and externally (TMX=0, FSX is an input) generated frame syncs. This distinction is made because in the former case, the frame sync pulse is generated by the transmitting device as a direct result of a write to DXR. In the latter case, there is no such direct effect. Instead, the transmitting device must write to DXR and wait for an externally generated frame sync.

If frame sync pulse are internally generated (TXM=1), then after a write to DXR, a frame sync pulse is generated on the next rising edge of CLKX (For externally generated frame syncs the following events will occur whenever the frame sync pulse appears by the rising edge of CLKX after a write to DXR). Then on the next falling edge of CLKX, XSR is loaded with the value from DXR, and XRDY goes high, generating a transmit interrupt (XINT). On the next rising edge of the CLKX cycle, the first data bit (MSB first) is driven on the DX pin. With the fall of the frame sync pulse, the rest of the bits will be shifted out. (therefore, the first bit could have variable length if the frame sync is generated externally and does not fall within one CLKX cycle. Internally generated frame syncs are guaranteed by ’C54x timings.)

When all the bits are transferred, the DX pin enters the high-impedance state. Note that if DXR had not been loaded when XINT was generated, the $\overline{\text{XSREMPTY}}$ flag would become active (go low), indicating underflow. Thus, there is a 2-CLKX cycle latency (approximately) after DXR is loaded, before

the data is driven on the line, assuming that the frame sync pulse is generated internally (TXM=1). If the pulse is externally generated, this latency does not exist, and the timing specifications are relaxed. With externally generated frame sync, if the $\overline{\text{XSREMPTY}}$ flag is active and a frame sync pulse is generated, any old data in the DXR is transmitted. This is explained in detail in subsection 1.1.3.

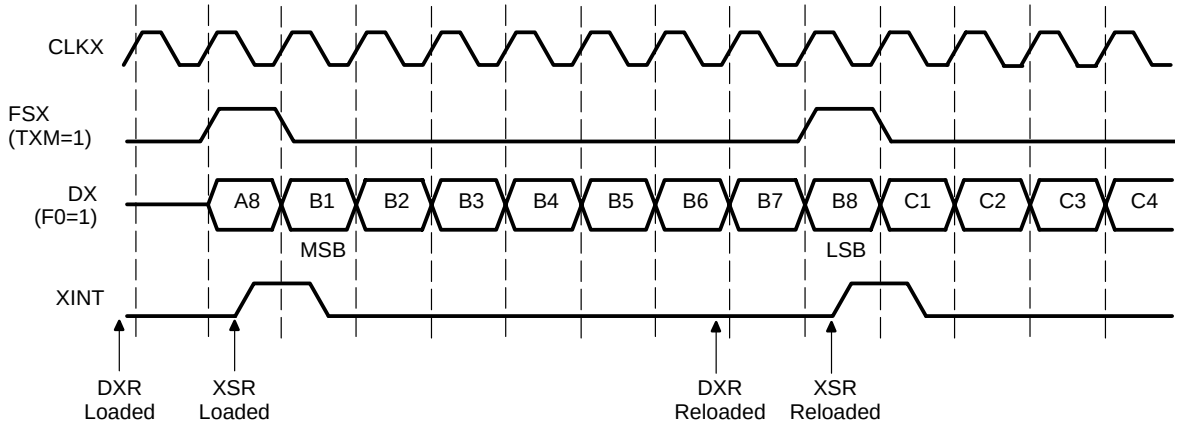
Figure 1–6. Burst-Mode Serial Port Receive Operation



The shifting into RSR begins on the falling edge of the CLKR cycle after the frame sync has gone low. After all the bits have been received, the contents of the RSR are transferred to the DRR on the falling edge of CLKR and RRDY goes high, generating a receive interrupt (RINT), as shown in Figure 1–6. Note that if the DRR from the previous receive had not been read and a frame sync appears, the RSRFULL flag would go high. This condition is an actual error and introduces questions of the serial port's behavior under various error situations: for example, the appearance of frame sync during a receive. Various error conditions are discussed in subsection 1.1.4.

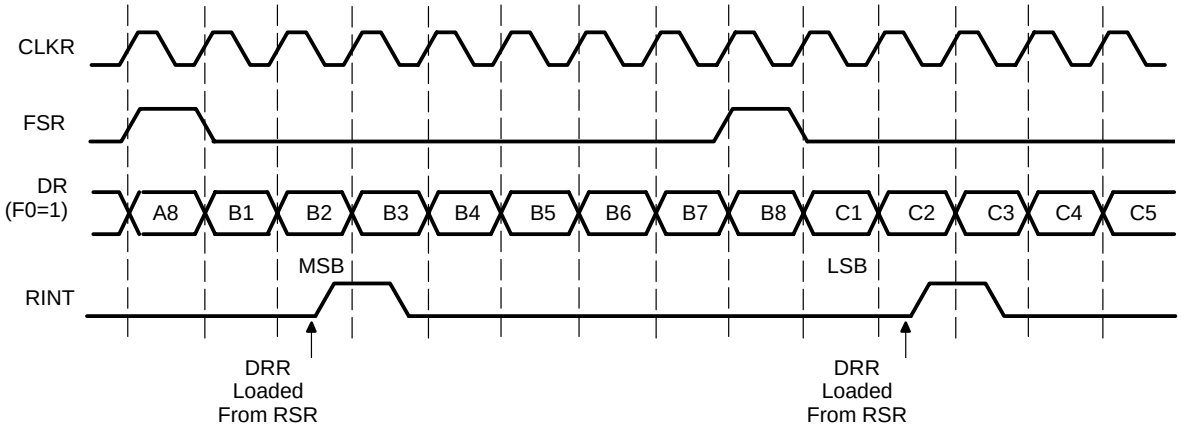
Note that if the packet frequency is increased, the inactivity period between the data packets for adjacent transfers decreases to zero. This corresponds to a minimum period between frame sync pulses (equivalent to 8 or 16 CLKX/R cycles, depending on FO) that corresponds to a maximum packet frequency at which the serial port can operate. At maximum packet frequency (see Figure 1–7), the timing looks like a compressed version of the one in Figure 1–6.

Figure 1–7. Burst-Mode Serial Port Transmit at Maximum Packet Frequency



The data bits in consecutive packets are transmitted continuously with no inactivity in between the bits. The frame sync pulse overlaps the last bit transmitted in the previous packet. The receive side in Figure 1–8 looks similar.

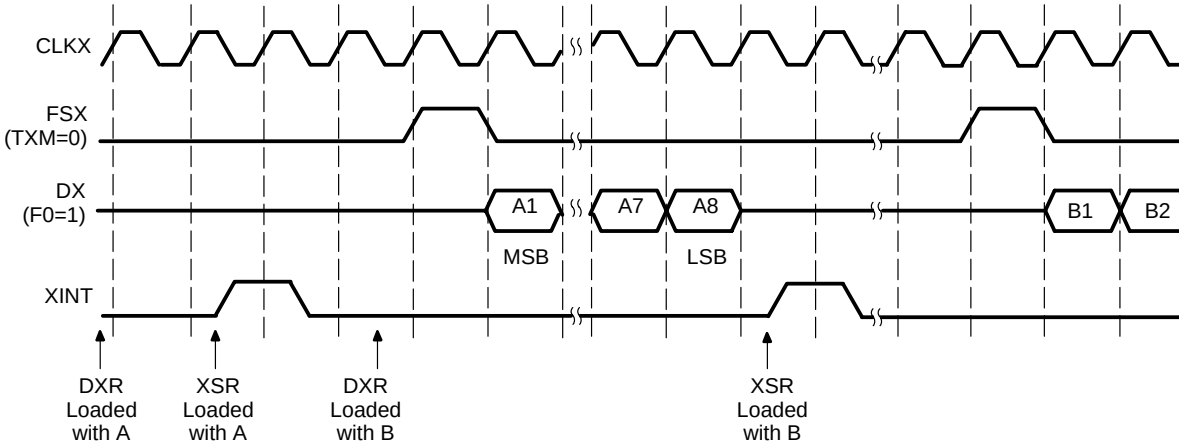
Figure 1–8. Burst-Mode Serial Port Receive at Maximum Packet Frequency



The maximum packet frequency transfer looks like a compressed version of burst mode with no periods of inactivity. The frame sync pulse overlaps the first bit transmitted.

Figure 1–7 and Figure 1–9 show the transfer of multiple data packets at maximum packet frequency; the frame sync appears to be extraneous information. Since the data packets are transmitted at a constant rate, the CLK provides enough timing information for the transfer and permits a continuous stream of data. Theoretically, only an initial frame sync signal needed to initiate the multi-packet transfer. This continuous mode is supported by the ’C54x serial port and is discussed in subsection 1.1.3.

Figure 1–9. Burst-Mode Serial Transmit Operation With Delayed Frame Sync in External Frame Sync Mode



The operation of the serial port with external frame sync is similar to that with internal frame sync. Events occur when the external frame sync appears. When the external frame sync is delayed, however, the double buffer is filled and frozen until the delayed frame sync appears, as shown in Figure 1–9. When the delayed frame sync occurs, A is transmitted on DX; after the transmit, a DXR-to-XSR copy of B occurs, and XINT is generated. The next frame sync after the delayed frame sync caused B to be transmitted on DX. Note that when the loading of B into DXR occurs, a DXR-to-XSR copy of B does not occur, and XINT is not generated because A has not been transmitted on DX. Any subsequent writes to DXR before the delayed frame sync occurs would overwrite DXR.

1.1.3 Transmit and Receive Operations (Continuous Mode)

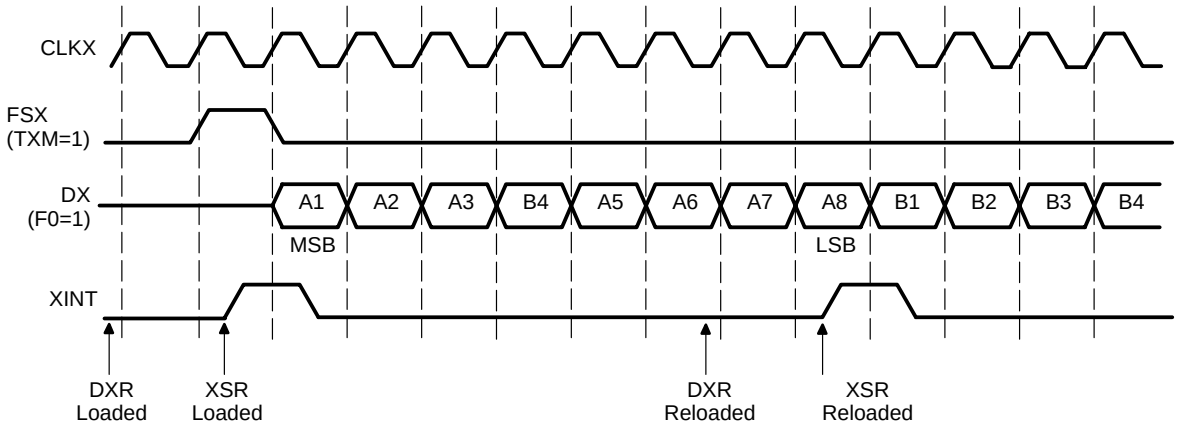
In the continuous mode, the frame sync signal on FSX/FSR is not necessary for consecutive packet transfers at maximum packet frequency after the initial pulse. Continuous mode is selected by setting FSM=0. Upon the first store to DXR in continuous mode, a frame sync is generated for the first transmission and then no more. As long as DXR is updated once every transmission, the continuous mode continues. Falling to update causes the serial port to halt, as in the burst mode case (The $\overline{\text{XSREMPTY}}$ flag becomes asserted etc.). If DXR is written to after the halt, the device restarts the continuous mode transmit and generates an FSX, assuming that the frame sync is internally generated. The difference between transmits using internal and external frame syncs is similar to the one discussed in subsection 1.1.2

If the frame syncs are externally generated (TXM=0), then DXR should be loaded, and the appearance of an external frame sync on the FSX pin restarts

a new continuous mode transmit. If the DXR has not been updated with external frame sync, the DX pin remains in the high-impedance state. This is different from the burst mode operation, which is covered in subsection 1.1.2. The continuous mode can be discontinued—in other words changed to burst mode—only by a serial-port or device reset. Changing the FSM bit during transmit or halt is not guaranteed to switch to burst mode.

The transmit timing in continuous mode is shown in Figure 1–10.

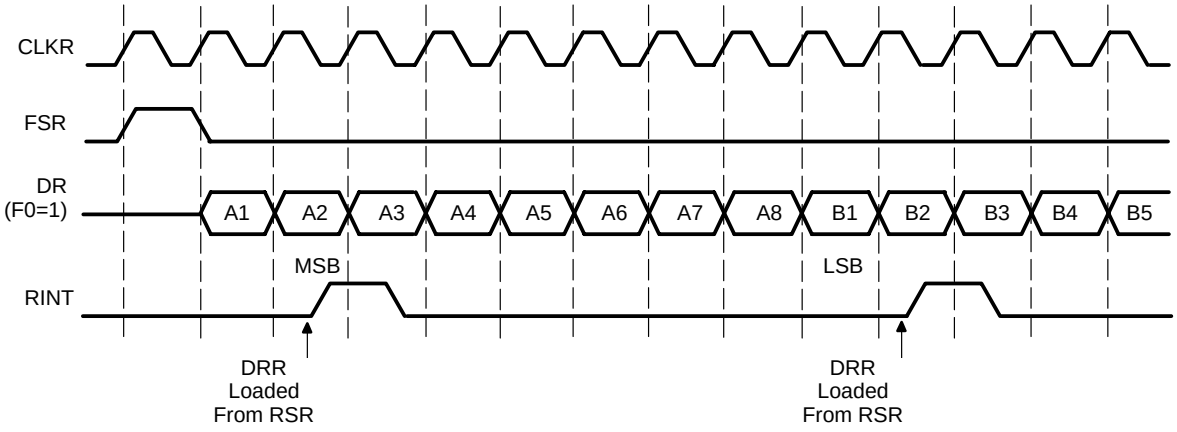
Figure 1–10. Serial Port Transmit Continuous Operation



Transmit timing in continuous mode is similar to the continuous stream in Figure 1–9. The major difference is the lack of a frame sync pulse after the initial one. As long as DXR is updated once per transmission, this mode will continue. Overwrites to DXR behave just as in burst mode. The data written last is transmitted. XSR operation is not disturbed. An external FSX pulse on the line will abort the present transmission, cause one data packet to be lost, and initiate a new continuous mode transmit. this is explained in more detail in subsection.

The receive operation is similar to the transmit operation. After the initial frame sync pulse on FSR, no more frame syncs are needed. This mode will continue as long as DRR is read every transmission. If it is not read, the serial port receive will halt (RSRFULL flag becomes active). Reading DRR will restart the continuous mode as soon as a frame sync is received. The continuous mode must be discontinued with a serial port or device reset. The receive timing can be seen in Figure 1–11.

Figure 1–11. Serial Port Receive Continuous Operation



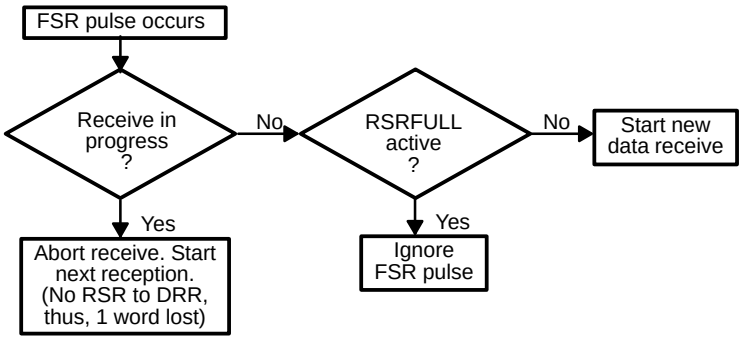
Receive timing in continuous mode is similar to the continuous stream in Figure 1–9. The major difference is the lack of a frame sync pulse after the initial one. If a pulse occurs on FSR during transmission (an error), then the receive operation is aborted, one packet is lost, and a new receive cycle is begun. This is discussed in more detail on page 1-16.

1.1.4 Error Conditions

Error conditions result from an unprogrammed event occurring to the serial port. These conditions are operational aberrations such as overrun, underflow, or a frame sync pulse during a transmission. You may need to understand how the serial port handles these errors and the state it acquires during these error conditions. Because they differ slightly in burst and continuous modes, the error conditions are discussed separately.

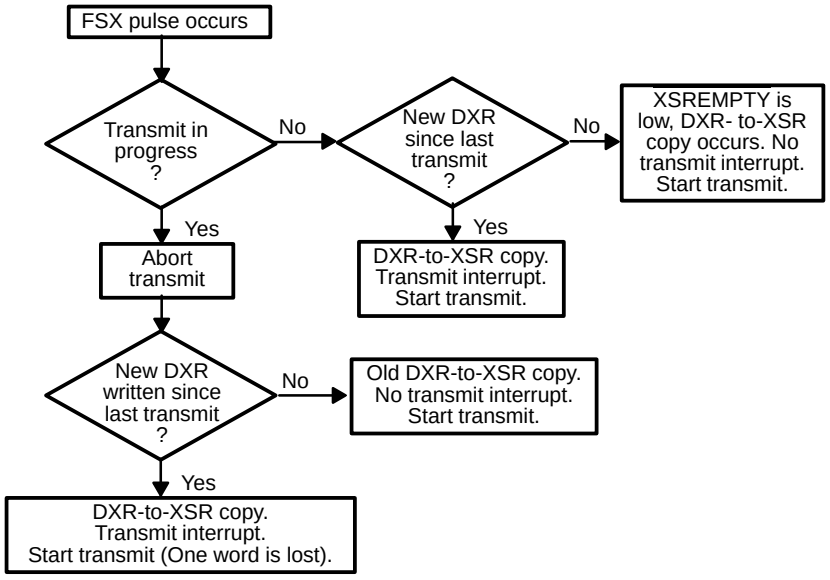
In burst mode, the first error condition (discussed in subsection 1.1.2) is the RSRFULL flag. Basically, this flag occurs when the device has not read incoming data and more data is being sent, which is indicated by a frame sync pulse on FSR. The processor halts serial port receives until DRR is read. Thus, any further data sent is lost. If receive errors continue, and the frame sync occurs during a receive (that is, data is being shifted into RSR from DR pin), then the present receive is aborted and a new one begins. Thus, the data that was being loaded into RSR is lost, but the data in DRR is not. No RSR-to-DRR copy occurs. Figure 1–12 shows the serial port receive side behavior for a frame sync pulse during a receive and includes nonerror situations.

Figure 1–12. Receive Error (Normal or Burst Mode)



Transmit errors in burst mode result when a frame sync occurs during various conditions. Underrun in burst mode is not considered an error and is explained in subsection . If a transmission is in progress (that is, XSR data is being driven on the DX pin) when the frame sync pulse occurs, then the present transmit is aborted, and data in the XSR is lost. Then, whatever data is in the DXR at the time of the frame sync pulse is transferred to XSR (DXR-to-XSR copy) for transmitting. However, a transmit interrupt XINT is generated only if the DXR has been written to after the last transmit. Also, if $\overline{\text{XSREMPTY}}$ is active and a frame sync pulse appears, the old data in DXR is shifted out. Figure 1–13 summarizes serial port transmit behavior with error (and nonerror) conditions.

Figure 1–13. Transmit Error (Normal or Burst Mode)



In continuous mode, errors take on a broader meaning. Data transfer is supposed to be occurring at all times in continuous mode. Thus, underflow (=0)

is considered an error in continuous mode because data is not being transmitted. As in burst mode, overrun is an error, and both of these cause the serial port receive or transmit sections to halt. The operation of both these flags is explained in subsection in the and RSRFULL flags description. Underflow and overrun errors are not fatal; they can not corrected by reading DRR of writing to DXR. In a write to DXR to deactivate , either a frame sync pulse is generated (if FSM=1) or required (if FSM=0). On the receive side, however, after DRR is read to deactivate RSRFULL, a frame sync pulse is not required. The receive side of the serial port keeps track of the word (either 8– or 16–bit) boundary, even though it is not receiving data. When the RSRFULL flag is deactivated by a read from DRR, the receiver begins the read from the correct bit.

Another cause for error is the appearance of frame syncs during a transmission. After the initial frame sync in continuous mode, no others should occur. When a frame sync pulse occurs during a transmit, the current transmit operation(that is, serially driving XSR data onto DX pin) is aborted, and data in XSR is lost. A new transmit cycle is initiated., as long as the DXR is updated once per transmission afterward. During a receive in continuous mode, the situation is similar: if a frame sync pulse occurs, one packet of data (8–bit byte or 16–bit word, depending on FO) is lost. The RSR bit counter is reset, so the data that was being shifted into RSR from the DR pin is lost. Data then driven on DR is shifted into RSR. Therefore, the frame sync during transmission chart for continuous mode looks like the left half of the burst mode charts in Figure 1–5 and Figure 1–6 because a receive or transmit is always in progress.

Figure 1–14 and Figure 1–15 show receive and transmit errors for continuous mode. Note that if a frame sync occurs after deactivating the RSRFULL flag by reading DRR but before the beginning of the next word (either 8- or 16-bit) boundary, a receive abort condition occurs. Also, note a major difference in the transmit continuous mode error compared with transmit burst mode error. If $\overline{\text{XSREMPTY}}$ is active in continuous mode and an external frame sync occurs, no old data is transmitted. Instead, since underflow in continuous mode is considered an error, the frame sync pulse is ignored, and the DX pin remains in the high-impedance state.

Figure 1–14. Receive Error (Continuous Mode)

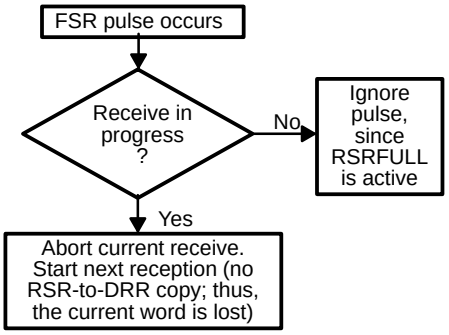
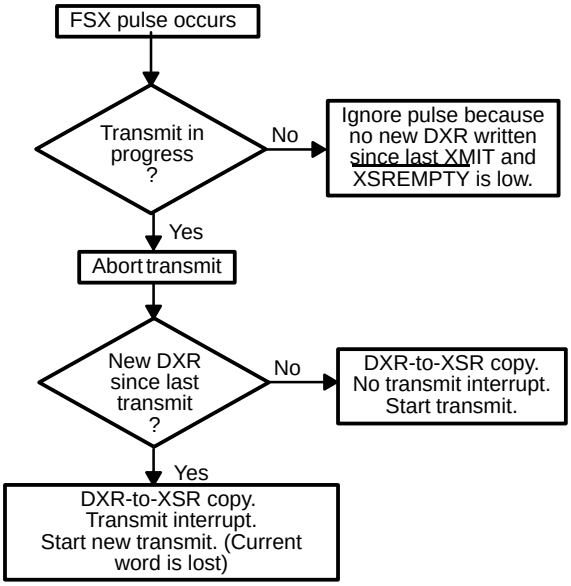


Figure 1–15. Transmit Error (Continuous Mode)



1.1.5 Example

The code examples that follow show a one-way transmit from device 0 to device 1 of an arithmetic sequence of numbers. The numbers are written in each device in a block from 9000h to b000h in data memory. Device 0 waits in a BIO loop for a ready to receive signal (XF) from device 1 and initializes the transfer with a value of zero. Only device 0's transmit interrupt is enabled; its transmit ISR writes the value it will send into its own memory.

Example 1–1. One-Way Transmit Operation from Device 0 to Device 1—Transmit Side

```
* Device 0 – Transmit side
:      :            :
                                ;Setup SPC0 as CLK source
                                ;and internal frame sync
       STM    #0038h,spc0       ;Set TXM=MCM=FSM=1,
                                ;DLB=FO=0.
                                ;and put SP0 into reset
                                ;(XRST=RRST=0)
       STM    #00F8h,SPC0       ;Take SP0 out of reset
                                ;Setup interrupts
       STM    #0FFFFh,IFR       ;Clear IFR
       STM    #020h,IMR         ;Turn on XINT0
       RSBX   1,INTM            ;Enable interrupts

ILOOP      BC     SENDZ,BIO     ;Wait for ready–to–receive
           B      ILOOP         ;from other device

SENDZ      LD     #0,A          ;First transmit/write
                                ;Value is 0.
           STM    #9000h,AR7    ;Setup where to write
           STL    A,*AR7        ;Write first value
           STLM   A,DXR0        ;Transmit first value

SELF1      B      SELF1         ;Wait for interrupts
XMT_ISR    LDM    AR7,A         ;Check if past 0x0b000
           SUB    #0B000h,A     ;i.e. end of block
           BC     END_SERP,AGEQ ;Go to tight loop if so
           ADDM   #1,*AR7       ;Add one
           LD     *AR7+,A
           STLM   A,DXR0        ;Transmit value
           RETE

END_SERP   B      END_SERP      ;Sit in tight loop after
                                ;block is complete.
:      :            :
```

The code in device 1 follows. It sends a ready-to-receive signal (XF) to device 0. Only its receive interrupt is masked and its receive ISR reads from the DRR, writes to the block, and checks if it has reached the end of the block.

Example 1–2. One-Way Transmit Operation from Device 0 to Device 1—Receive Side

```
*Device 1 – Receive
                                ;Set SP0 as CLK, frame
                                ;sync receive
          STM    #0008h,SPC0    ;Set TXM=MCM=DLB=FO=0, FSM=1
                                ;And put SP0 into reset
                                ;(XRST=RRST=0)

          STM    #00C8h,SPC0    ;Take SP0 out of reset
                                ;Setup interrupts
          STM    #0FFFFh,IFR    ;Clear IFR
          STM    #010h,IMR      ;Turn on RINT0
          RSBX   1,INTM         ;Enable interrupts
          STM    #9000h,AR7     ;Setup where to write
                                ;received data
          SSBX   1,XF           ;Signal ready to receive

SELF1  B  SELF1                 ;Wait for interrupts

RCV_ISR
          LDM    DRR0,A         ;Load received value
          STL    A,*AR7+        ;Write to memory block
          LDM    AR7,A          ;Check if past 0x0b000
          SUB    #0B000h,A      ;i.e. end of block
          BC     END_SERP,AGEQ  ;go to tight loop if so
          RETE

END_SERP  B  END_SERP           ;Sit in tight loop after
                                ;block is complete.
```

1.2 Buffered Serial Port (BSP)

The Buffered Serial Port (BSP) is comprised of a full-duplex, double-buffered Serial Port Interface (SPI) and an Autobuffering Unit (ABU). The SPI block of the BSP is an enhanced version of the one on existing 320C54x devices. The ABU is a new block which allows the SPI to read/write directly to 'C54x internal memory independent of the CPU. This results in a minimum overhead for SPI transactions and faster data rates.

When auto-buffering capability is disabled (Standard Mode), transfers with SPI are done under user control (software). In this mode, the ABU is transparent; the word based interrupts (WXINT and WRINT) provided by the SPI are sent to the CPU as transmit interrupt (XINT) and receive interrupt (RINT). When auto-buffering is enabled, word transfers can be done directly between SPI and 'C54x internal memory using the ABU embedded address generators.

The on-chip full duplex serial interface (SPI) provides direct communication with serial devices such as codecs, serial A/D converters, and other serial devices with a minimum of external hardware. The double-buffered SPI allows transfer of a continuous communication stream (8-,10-,12- or 16-bit data packets). Data packets are directed by a frame synchronization pulse for every packet in Burst Mode.

In Continuous Mode, serial operation, once initiated, requires no further frame synchronization pulses. Frame signal as well as a frequency programmable serial clock can be provided by the SPI for transmission. Polarity of frame and clock strobes can be programmed . The SPI is fully static and thus will function at arbitrarily low clocking frequencies. The maximum operating frequency is CLKOUT (28.6 Mbit/s at 35ns, 40 Mbit/s at 25 ns). SPI transmit section includes a PCM (Pulse Coded Modulation) mode that allows easy interface with a PCM line.

The ABU has its own set of circular addressing registers with corresponding address generation units. Memory for the buffers reside in 2K words of 'C54x internal memory. The length and starting addresses of the buffers are user programmable. A buffer empty/full interrupt can be posted to the CPU. Buffering can be easily halted thanks to an auto disabling capability. autobuffering capability can be separately enabled for transmit and receive sections. When autobuffering is disabled, operation is similar to 'C54x standard serial port operation.

1.2.1 BSP Configuration for ’C542

- Memory-mapped registers**:**

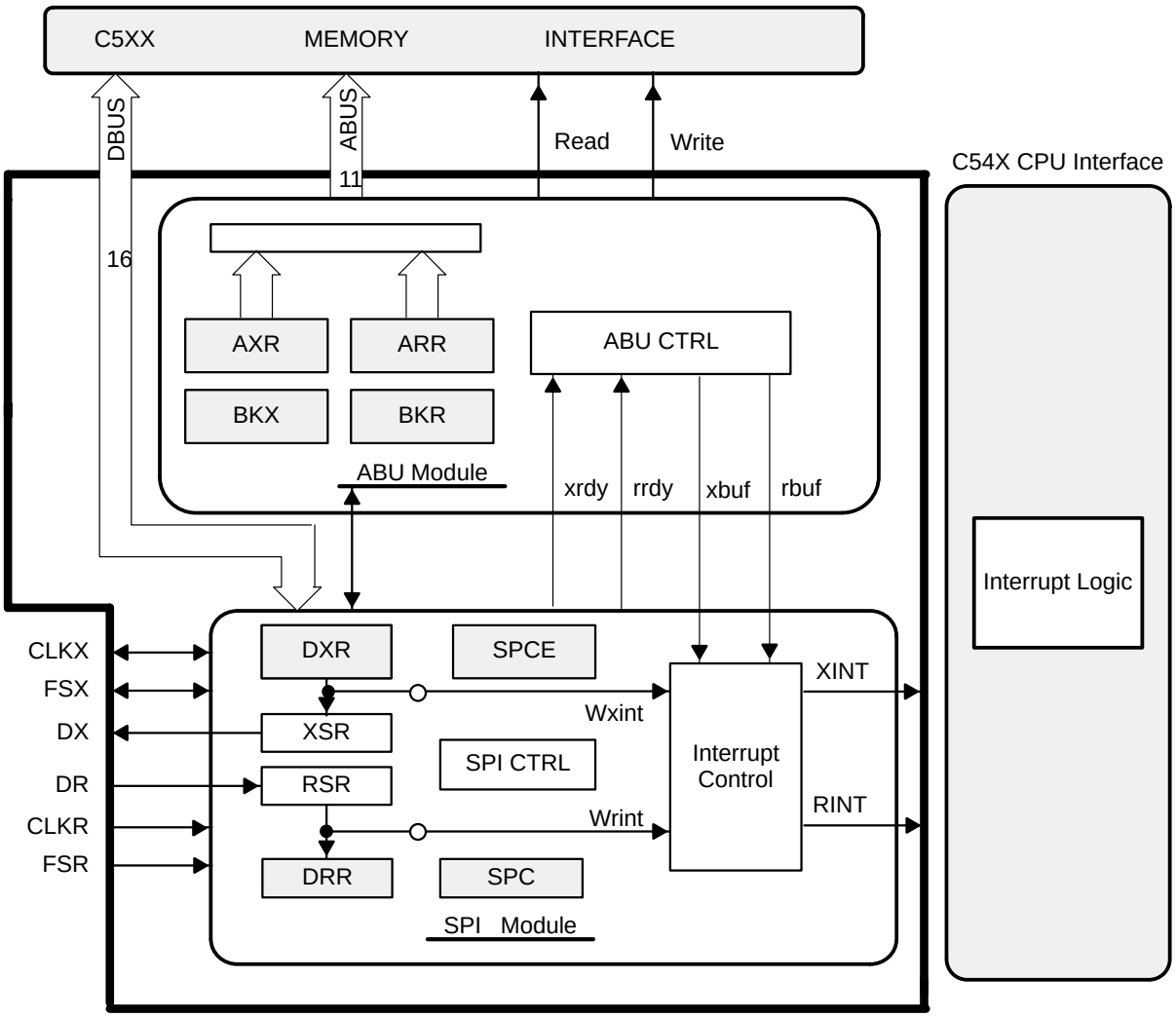
DRR	Add 20h	Data Receive Register
DXR	Add 21h	Data Transmit Register
SPC	Add 22h	Serial Port Control Register
SPCE	Add 23h	BSP Control Extension Register
AXR	Add 38h	ABU (autobuffering unit) Transmit Address Register
BKX	Add 39h	ABU Transmit Buffer Size Register
ARR	Add 3Ah	ABU Receive Address Register
BKR	Add 3Bh	ABU Receive Buffer Size Register

- Memory for the buffers:

  2k words of on-chip RAM located at address range 0800h–0FFFh

- Interrupt locations:
  - Receive interrupt: vector location is 50h, mask/flag bit is located at IMR/IFR[4:4]
  - Transmit interrupt: vector location is 54h, mask/flag bit is located at IMR/IFR[5:5]

Figure 1–16. Buffered Serial Port Block Diagram and Registers



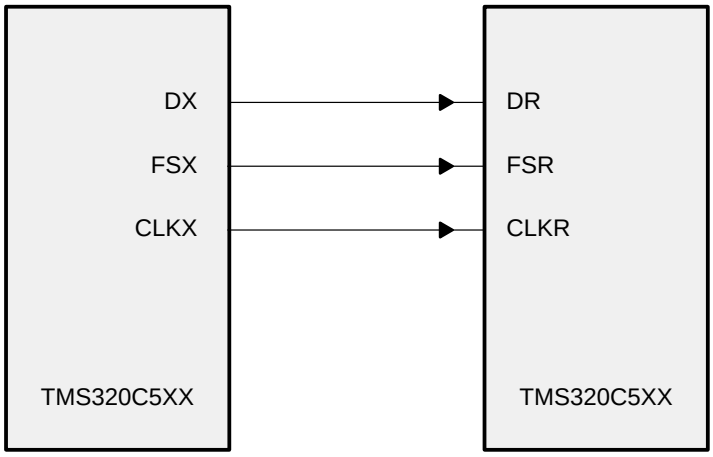
Legend:

XSR	16-bit Data Transmit Shift Register
DXR	16-bit Data Transmit Register
AXR	11-bit Address Transmit Register
BKX	11-bit Transmit Buffer Size Register
RSR	16-bit Data Receive Shift Register
DRR	16-bit Data Receive Register
ARR	11-bit Address Receive Register
BKR	11-bit Receive Buffer Size Register
SPC	16-bit Serial Port Interface Control Register
SPCE	16-bit Serial Port Interface Control Register Extension and Autobuffering Unit Control

1.2.2 Serial Port Interface (SPI) Operation

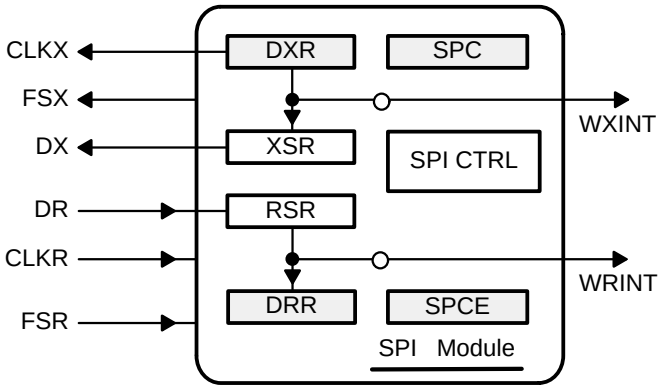
Three lines are necessary for transmission between the transmitting device and receiving device. On the transmission side, the Transmit Frame Synchronization Signal pin (FSX) is the control pin that initiates the transfer. The Transmit Clock Signal pin (CLKX) clocks the transmitted data that are sent on the Serial Data Signal pin (DX). Corresponding pins on the receiver side are the Receive Frame Synchronization Signal (FSR),the Receive Clock Signal pin (CLKR),and the Received Serial Data Signal pin (DR). Figure 1–17 is showing two ’C54x serial ports connected for a half duplex transmission. In this setup, the transmitting ’C54x is providing Transmit Frame and Transmit Clock.

Figure 1–17. Half-Duplex Communication Between Two TMS320C54xs



SPI uses four memory-mapped registers (SPC, SPCE, DXR and DRR) shown in Figure 1–18 and two other registers (XSR and RSR) that are not accessible to user but permit the double-buffering capability.

Figure 1–18. SPI Module Block Diagram



The SPI control registers (SPC and SPCE) control the operation. Transmit data is written to the Data Transmit Buffer (DXR) while received data is read from the Data Receive Register (DRR). A transmit is performed by writing data to the DXR which will copy the data to the XSR (Transmit Shift Register) when the XSR will be requested (Frame Synchronization occurrence for instance) to shift out the data on the DX pin thus allowing another write to DXR. The process is similar on the receive part. Data from DR pin is shifted into the Receive Shift Register (RSR) that copies it to the Data Receive Register from which it may be read. Thus the serial port is double-buffered because data may be transferred to/from DXR/DRR while another transmission/reception is being performed. If the serial port is not being used, the DXR and the DRR registers can be used as general purpose registers. In this case, FSR should be connected to a logic low to prevent a possible receive operation from being initiated. Note that when autobuffering is enabled for Transmit or Receive the DXR or DRR registers cannot be accessed by user program. DRR register cannot be written when receiver is enabled (RRST=1).

1.2.2.1 Serial Port Interface Control Registers

The SPC control register is a 16-bit memory mapped register that configures the SPI. Some of the bits are read-only while others are read/write. The SPCE control register is a 16-bit extension control read/write register. The 10 LSBs of SPCE are dedicated to SPI control whereas the 6 MSBs are used for Autobuffering Unit Control (see paragraph 3.2). Figure 1–19 shows the SPC bit positions, and Figure 1–20 shows the SPCE bit positions. A summary of each bit is given in Table 1–3 for SPC and in Table 1–4 for SPCE. SPC register format is identical to ’C54x standard serial port SPC register.

Figure 1–19. SPC Register

7	6	5	4	3	2	1	0
RRST	XRST	TXM	MCM	FSM	FO	DLB	RES
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R

15	14	13	12	11	10	9	8
FREE	SOFT	RSRFULL	XSREMPTY	XRDY	RRDY	IN1	IN0
R/W	R/W	R	R	R	R	R	R

Note: Note: R=Read W=Write

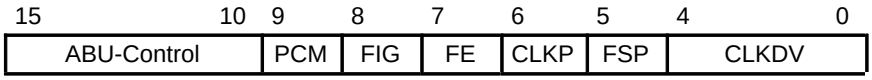
Table 1–3. SPC Register Bits Summary

Bit	Name	Function
0	Reserved	Always read as 0
1	DLB	Digital Loop Back Mode Bit. When DLB=1, DR and FSR are connected to DX and FSX as shown in Figure 1–21. Additionally, CLKR is driven by CLKX if MCM=1 or is taken from the CLKR pin of the device if MCM=0 allowing CLKX and CLKR to be tied together externally and supplied by a common external clock source. Note that TXM must be set to one for proper operation in DLB mode. Note also that the FSX and DX signals appear on the device pins in DLB mode but FSR and DR do not.
2	FO	The FO Format Bit in conjunction with FE (Format Extension Bit) of SPCE register specifies the word length. When FO FE = 00 format is 16-bit words, when FO FE = 01 format is 10-bit words, when FO FE = 10 format is 8-bit words and when FO FE = 11 format is 12-bit words. Note that for 8-,10-,12-bit words, received words are right justified and sign bit is extended to form a 16-bit word. Words to transmit must be right justified
3	FSM	The Frame Sync Mode (FSM) specifies whether frame synchronization pulses are required for serial port operation. If FSM=1, a frame pulse is required on FSX/FSR for the transmission/ reception of each word (Burst Mode). When the serial port is operating in continuous mode, FSM=0.
4	MCM	The Clock Mode Bit (MCM) specifies the clock source for CLKX. If MCM=0, CLKX is taken from the CLKX pin. If MCM=1, CLKX is driven by an on-chip source having a frequency equal to 1/(CLKDV+1) of CLKOUT. CLKDV programmable division factor is specified in SPCE register. Note that if MCM=1 and DLB=1, a CLKR signal is also supplied by the internal source.
5	TXM	The Transmit Mode Bit (TXM) configures the FSX as an input (TXM=0) or as an output (TXM=1). When TXM=1, frame sync pulses are generated internally when data is transferred from DXR to XSR to initiate transfer. The internally generated frame sync signal is synchronous with respect to CLKX. When TXM=0, FSX is supplied on the FSX pin by external device.
6 7	XRST RRST	The Transmit Reset (XRST) and Receive Reset (RRST) signals reset the transmitter and receiver respectively. If the SPC is to be modified to reconfigure the serial port, a total of two writes should be made to the SPC. The first write should write zeroes to XRST and RRST and the desired configuration to bits 1–5. The second write should write ones to XRST and RRST, taking the SPI out of reset. When a zero is written to either of these bits, activity in the corresponding section halts. When XRST=0 and RRST=0, the internal clocks to the serial port are shut off, allowing the device to run in a lower power mode of operation. Notes: 1– Writing a zero to XRST clears the XSREMPTY bit and sets the XRDY bit to 1. 2–Writing a zero to RRST clears the RSRFULL bit and RRDY bit. 3–Writing DXR while XRST =0 has no effect on XRDY bit.

Table 1–3. SPC Register Bits Summary

Bit	Name	Function
8 9	IN0 IN1	The Input 0 Bit (IN0) and Input 1 Bit (IN1) allow the CLKR and CLKX pins to be used as bit inputs. The IN0 and IN1 bits of SPC are reflections of the current levels on the CLKR and CLKX pins of the device. The levels on these pins can be read by reading the SPC. They can be tested by using the PLU or the BIT or BITT instructions. Note that as IN0/IN1 are sampled with CPU clock, CLKX/CLKR frequency must be at least two times CLKOUT frequency for valid IN0/IN1 values. Note that IN0 and IN1 can be used as bit inputs if SPI is not used.
10 11	RRDY XRDY	Receive Ready and Transmit Ready Bits. A transition from 0 to 1of the RRDY bit indicates that the received data in RSR has been copied to DRR and that data can be read. In standard mode, a receive interrupt is generated upon this transition. When received data in DRR is read (by software in standard mode or by ABU in auto-buffering mode) the RRDY is reset to 0. RRDY is set to 0 upon device reset and SPI receive reset (RRST=0). A transition from 0 to 1 of the XRDY bit indicates that the new contents of DXR have been copied to XSR and that DXR is ready to be loaded with a new data word. In standard mode a transmit interrupt is generated upon the transition. When DXR is written (by software in standard mode or by ABU in auto-buffering mode), the XRDY bit is reset to 0. XRDY is set to 1 upon device reset and SPI transmit reset (XRST=0).
12	XSREMPTY	Transmit Shift Register Empty Flag. Note:XSREMPTY=0 upon device reset, SPI transmit reset (XRST=0).
13	RSRFULL	Receive Shift Register Full Flag. This bit is set to one when a word has been shifted in the RSR register and current DRR value has not yet been read (RRDY=1). If RSRFULL=1, receiver halts and waits for DRR to be read; the data in RSR is preserved but any data sent on DR is lost. Note:RSRFULL=0 upon device reset, SPI receive reset (RRST=0).
14	SOFT	SOFT bit. This bit is enabled when the FREE bit is 0. If FREE=0 the SOFT bit selects immediate stop if 0, stop after word transmit completion if 1. See page 10. SOFT=0 upon device reset.
15	FREE	FREE bit. If FREE=1 free run is selected regardless of the value of SOFT bit. If FREE=0 the SOFT bit selects the emulation mode as described above. See page 10. FREE=0 upon device reset.

Figure 1–20. SPCE Register

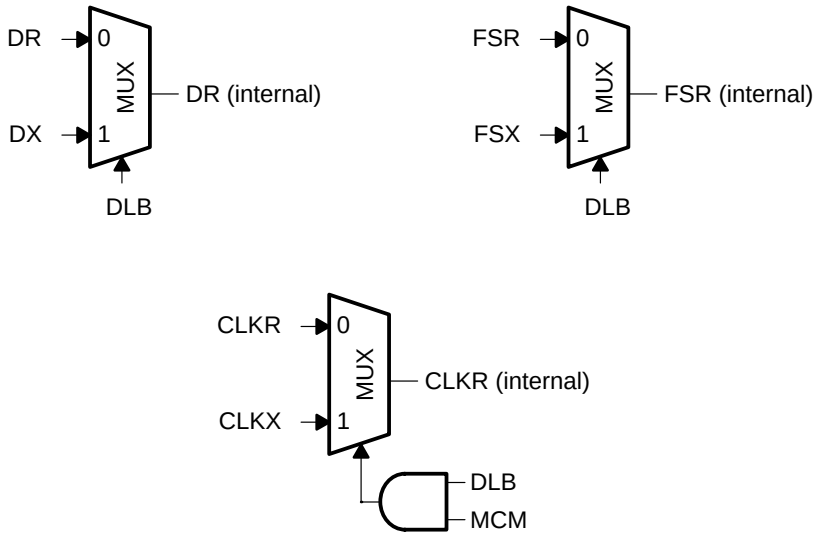


Note: R=Read W=Write

Table 1–4. SPCE Register Bits Summary

Bit	Name	Function
0–4	CLKDV	Internal Transmit Clock Division Factor. When MCM bit of SPC register is set to 1, CLKX is driven by an on-chip source having a frequency equal to 1/(CLKDV+1) of CLKOUT. CLKDV range is [0–31]. When CLKDV is odd or equal to zero, CLKX duty cycle is 50/50; when CLKDV is even value (CLKDV=2p), CLKX high state duration is (p+1)/(p) cycles and low state duration is (p)/(p+1) cycles when polarity (CLKP) is 0/1. CLKDV value upon device reset is 3.
5	FSP	Frame Sync Polarity Bit. When FSP=1, frame sync pulses (FSX and FSR) are active low. When FSP=0, frame sync pulses are active high. FSP=0 upon device reset.
6	CLKP	Clocks Polarity Bit When CLKP=1, data is sampled by receiver on CLKR rising edge and sent by transmitter on CLKX falling edge. When CLKP=0 data is sampled by receiver on CLKR falling edge and sent by transmitter on CLKX rising edge. CLKP=0 upon device reset.
7	FE	Format Extension Bit. The FE Format Bit in conjunction with FO of SPC register specifies the word length. When FO FE = 00 format is 16-bit words, when FO FE = 01 format is 10-bit words, when FO FE = 10 format is 8-bit words and when FO FE = 11 format is 12-bit words. Note that for 8-,10-,12-bit words, received words are right justified and sign bit is extended to form a 16-bit word. Words to transmit must be right justified. FE =0 upon device reset.
8	FIG	Frame Ignore Bit. This control bit is operating only in transmit continuous mode with external frame and in receive continuous mode. When set to 1 frame pulses following first frame pulse that initiates SPI operation are ignored. When set to 0, frame pulses following first frame restart SPI. Upon device reset FIG=0.
9	PCM	Pulse Coded Modulation Bit Mode. This mode is active when PCM=1and affects transmitter only. In PCM mode, DXR is transmitted only if its 15th bit is set to 0. If this bit is set to 1, DXR is not transmitted and DX is put in high impedance during transmission period. DXR to XSR transfer is not affected by PCM bit value.
15–10	ABU–C	Reserved for autobuffering unit control .See paragraph 3.2).

Figure 1–21. Receiver Signal Multiplexers for Digital Loop Back



Upon device reset the values of the SPI registers are:

- DXR = 0000h
- DRR = 0000h
- SPC = 000010XR00000000b
- SPCE = 000000000000011b

X and R are respectively the current levels on the CLKX and CLKR pins.

SOFT and FREE (bits 14 and 15 of SPC control register) are special emulation bits that determine the state of the serial port clock when a breakpoint is encountered in the high-level language debugger. If the FREE bit is set to one, then upon a software breakpoint, the clocks continue to run (that is, free runs) and data is shifted out or shifted in. In this case, SOFT is a *don't care.*

If FREE is 0, then SOFT takes effect. If SOFT =0 ,then the clocks immediately stop. If the SOFT bit is one, the current transmission continues until completion of the word, and then the transmit clock halts, the receiver is not affected in this case. The options are as follows:

FREE	SOFT	
1	X	Free run
0	0	Immediate stop
0	1	Stop after completion of word

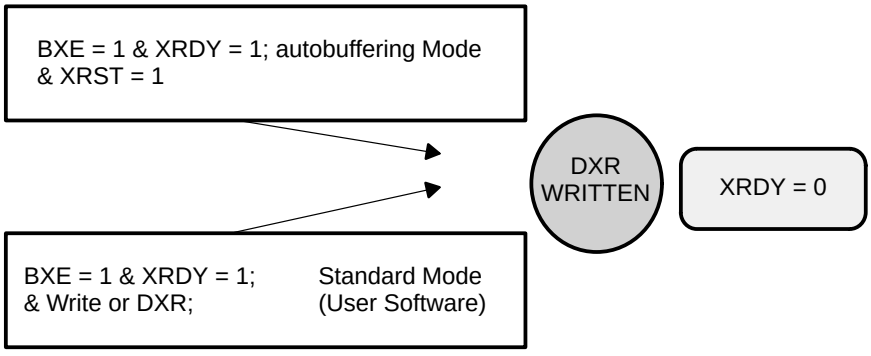
The default values for these bits is immediate stop (FREE=0,SOFT=0).

1.2.2.2 Serial Point Interface Transmit Operation

Two processes take place to ensure transmission,a process of filling DXR register under control of the CPU (user software) in Standard Mode or under control of the autobuffering unit (ABU) in auto buffered mode and a process of emptying DXR register under control of SPI. In order to synchronize and monitor these processes,two signals are used in SPC register. These signals are XRDY (Transmit Ready) and XSREMPTY (Transmit Shift Register Empty). Filling process is illustrated with Figure 1–22, and SPI emptying process is illustrated with Figure 1–23 (case is Burst Mode with External frame).When DXR is written (filling process), XRDY signal is cleared so that SPI process is informed that a new value is ready to be transmitted. When SPI is transferring this new value to transmit shift register (XSR), XRDY bit is set to 1. This transfer occurs upon various circumstances,depending on which type of Transmission Mode is selected. At the same time transfer occurs,a Transmit Request is sent to CPU as interrupt (XINT) in Standard Mode (BXE=0 in ABUC control register) or to the ABU (xrdy) Buffered Mode (BXE = 1 in ABUC register).In Standard Mode,the DXR should be written only if XRDY =1,which is guaranteed if the DXR write is made in response to a transmit interrupt or polling XRDY.

Transmit Shift Register Empty (XSREMPTY) info can be used to monitor state of XSR register. This bit can only be written by SPI. When a new word (XRDY=0) has been transferred from DXR to XSR,this bit is set to 1 while at the same time XRDY is set to 1.When new word has been shifted on transmit line (DX),this bit is set to 0 if another transmission with a new word is not following the current transmission.

Figure 1–22. DXR Filling Process



The SPI supports two modes of transmission,Burst Mode and Continuous Mode which are here below detailed. For all flowcharts and timing diagrams used for description and featuring frame and clock events, assumption is

made that clock polarity (CLKP) is 0 and frame polarity (FSP) is 0. Operation with other polarity can be derived easily. Table 1–5 is giving the list of signals and registers that are used for the transmit flowcharts.

All modes can be operated with external clock (MCM=0) or with internal clock (MCM=1). When clock is internal, frequency can be user programmed thanks to CLKDV field of SPCE register Frequency is equal to 1/(CLKDV+1) of CLKOUT.

All modes can also be operated in Pulse Coded Modulation Mode when PCM bit of SPCE register is set to 1. When PCM is set to 1,word to transmit is shifted out to transmit data line (DX) if bit at position 15 (MSB) is set to 0,if this bit is 1,DX line is high impedance driven during the transmission period.

Table 1–5. Transmit Flowcharts Signals and Registers

Name	Description
FSX	Transmit Frame Sync signal (internal or external)
CLKX	Transmit Serial Clock (internal or external)
DX	Transmit Serial Data Signal.
DXR	Data Transmit Memory-mapped register.
XSR	Transmit Shift Register.
XRDY	Transmit Ready status/control bit of SPC register.
XSREMPTY	Transmit Shift Register Empty status bit of SPC register.
FIG	Frame Ignore control bit of SPCE register.
XFRA	Transmit Frame Acknowledge internal signal. This bit manages sync pulse acceptance/non acceptance during continuous mode when FIG=0/1. XFRA=1 upon device reset or upon reset of Transmit Section of SPI (XRST=0).
XCOUNT	Internal Transmit Counter for data shifting.
XBITS	Number of bits to transmit (8,10,12 or 16).
FSXLOW	Internal Signal for FSX state monitoring. FSXLOW = 1 upon device reset or upon reset of SPI Transmit Section (XRST=0)
PCM	Pulse Coded Modulation bit of SPCE register.
XHZ	Internal signal for transmission of high impedance state in PCM mode when word is not transmitted.
XREQ	Transmission request (XINT in standard mode, request to ABU in auto buffered mode).

Burst Mode (FSM=1) With External Frame SYNC (TXM=0)

In this mode, data packet is marked by the Fame Sync Pulse on FSX. Some periods of inactivity may then occur between packets. The flowchart of Figure 1–23 is showing detailed SPI process embedding timing aspects. Figure 1–24 and Figure 1–25 show timing aspects for both short duration frame signal and long duration frame signal. SPI operation starts when frame signal becomes active,this event being sampled on falling edge of the transmit clock. In the TRANS state (on the next rising edge of transmit clock),transfer from DXR to Transmit Shift Register (XSR) is performed and if contents of DXR is new (XRDY=0) then XRDY and XSREMPTY are set to 1 and request for next word to transmit (XREQU) is sent to CPU as XINT interrupt (Standard Mode) or to the ABU (Auto buffered Mode).If DXR contents is not new (XRDY=1) then the same value will be transmitted and no Transmit Request will be generated. At the same time, DX is driven with first bit (MSB) to transmit. DX will be driven with this first bit as long as Frame Sync Pulse is high. Following bits are transmitted (SHIFT state) starting on next rising edge of Transmit Clock following Frame Sync Pulse low event ,with event being detected on a Transmit Clock falling edge. After all bits have been transmitted (TX_END state),DX is high impedance driven and XSREMPTY is set to 0.Note that this last state may not be reached if a new Frame Sync Pulse has occurred. This is illustrated with Figure 1–26 (Continuous transmission in Burst Mode) and Figure 1–27 (Transmission Abort). If a Frame Sync Pulse occurs during transmission transmission is restarted (TRANS state),the transmitted word is the same or a new one depending on DXR written with next word or not. Figure 1–28 and Figure 1–29 are showing respectively XSREMPTY activation and XSREMPTY deactivation.

Figure 1–23. SPI Transmit Process for Burst Mode With External Frame (FSM=1 and TXM=0)

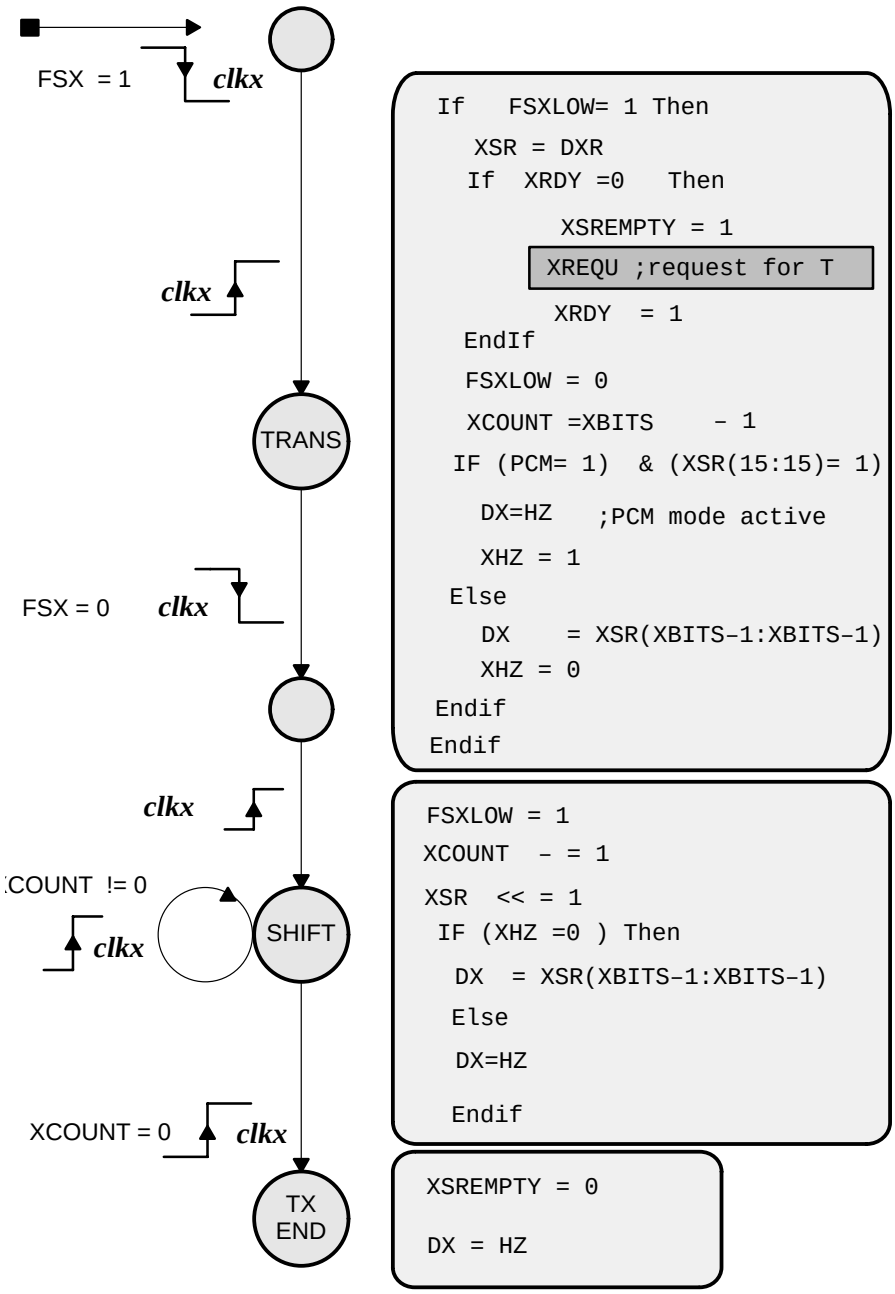
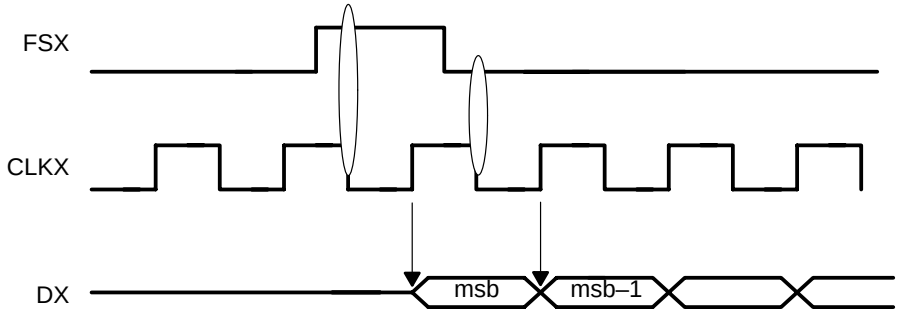


Figure 1–24. Short FSX Pulse

(a) Short FSX Pulse With FSX Going Low When CLKX Is High



(b) Short FSX Pulse With FSX Going Low When CLKX Is Low

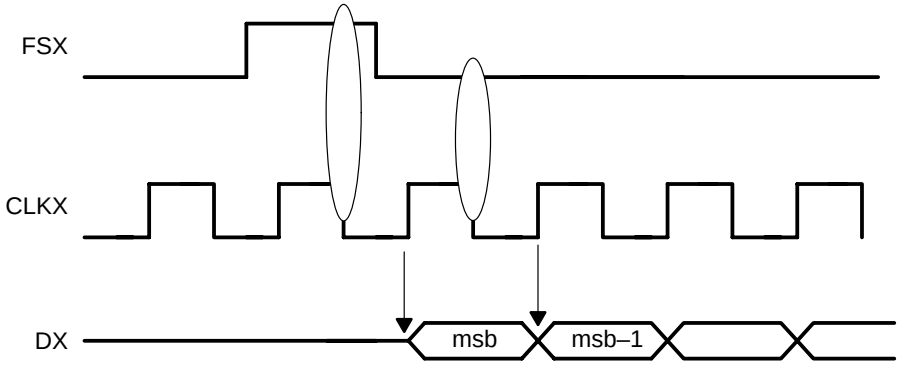
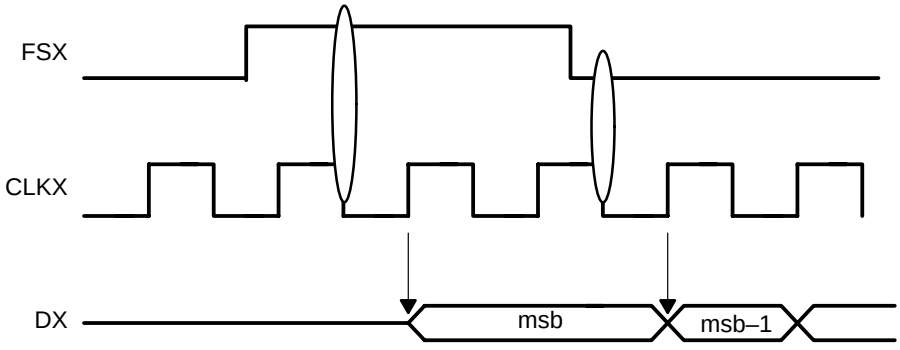


Figure 1–25. Long FSX Pulse

(a) Long FSX Pulse With FSX Going Low When CLKX Is High



(b) Long FSX Pulse With FSX Going Low When CLKX Is Low

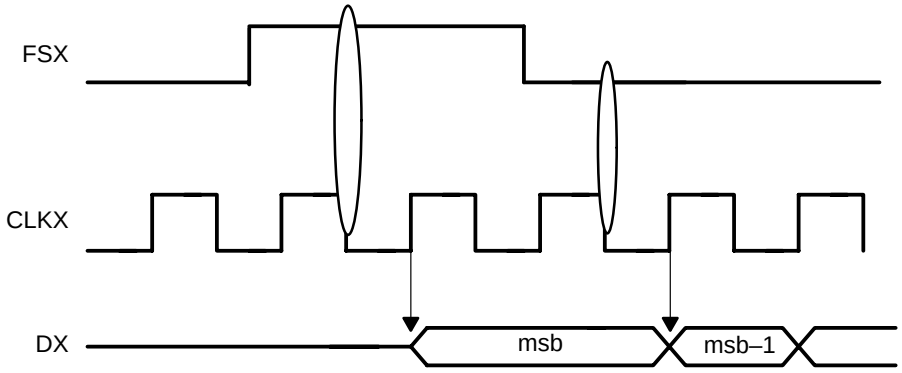


Figure 1–26. Transmit Burst Mode With External Frame in Continuous Operation

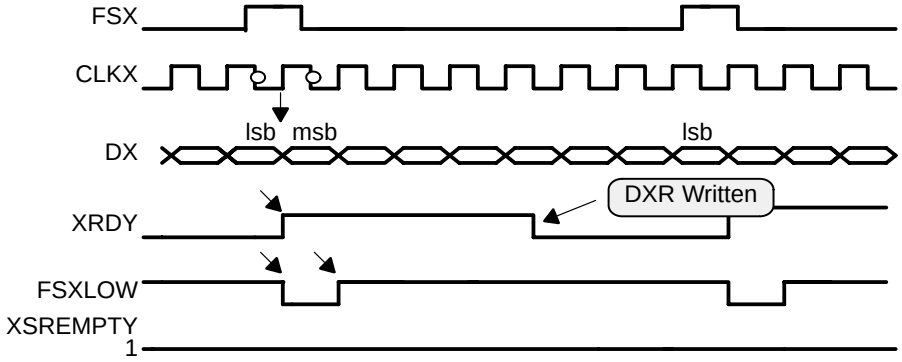


Figure 1–27. Transmit Burst Mode With External Frame (Transmit Aborts—Format Is 8 Bits)

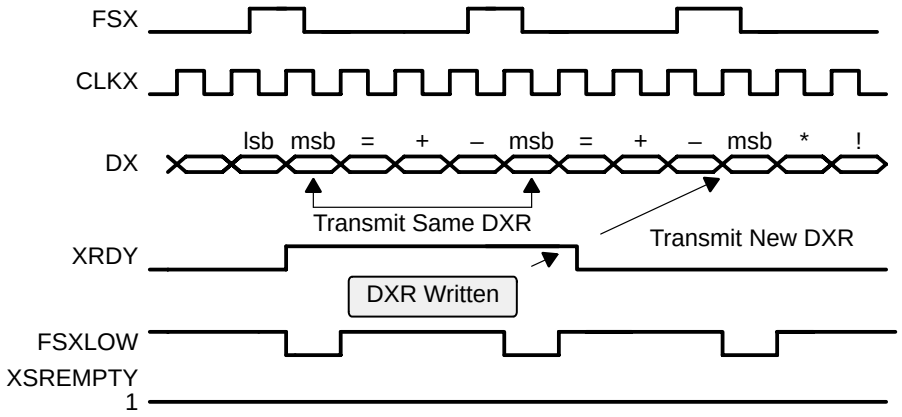


Figure 1–28. Transmit Burst Mode With External Frame—XSREMPTY Activation (Format Is 8 Bits)

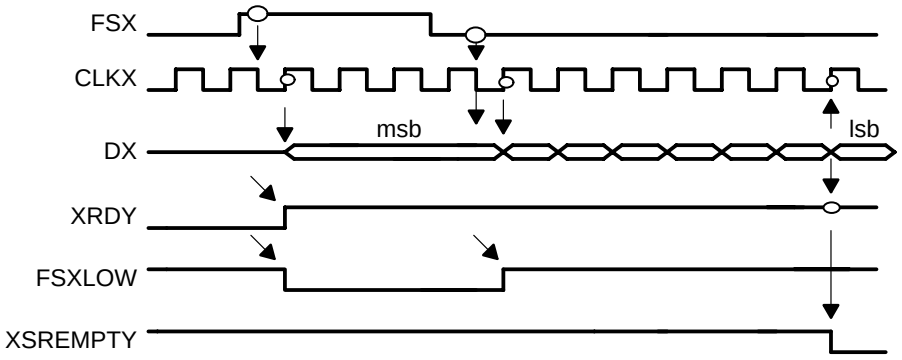
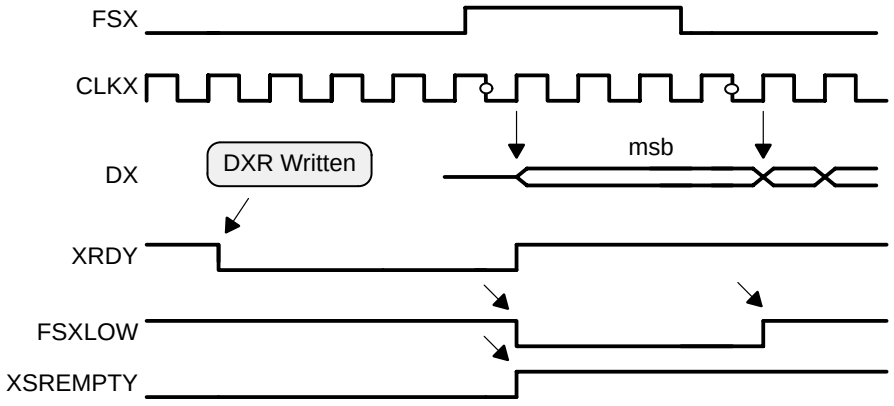


Figure 1–29. Transmit Burst Mode With External Frame—XSREMPTY Deactivation



Burst Mode (FSM=1) With Internal Frame SYNC (TXM=1)

In this mode, Frame Sync Pulse is generated internally (Figure 1–30) at the Transmit Serial Clock rising edge following a write to DXR register (XRDY going low).The Frame Sync Pulse is active during one full clock period. For further steps, SPI process is similar to External Frame mode. Thanks to double-buffering capability, continuous transmission can be done if DXR is updated in Transmit Interrupt (XINT) service routine. In Standard Mode,transmission can be initiated by an external event (external interrupt for instance) or by serial port receive interrupt (RINT). In Auto Buffered Mode, this mode will result in a continuous transmission with frame generated by SPI at each transmission start.

Figure 1–30. SPI Transmit Process for Burst Mode With Internal Frame (FSM=1 and TXM=1)

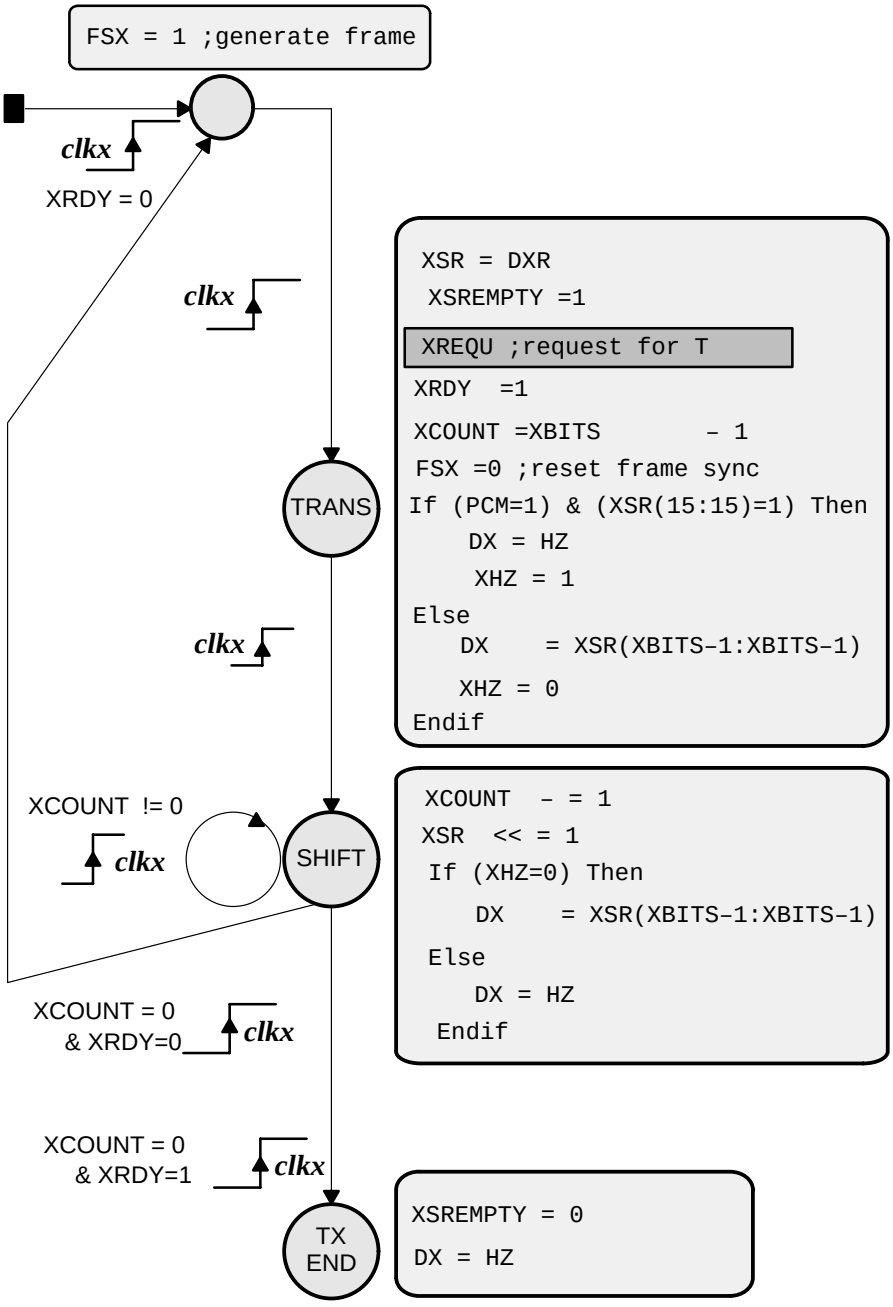
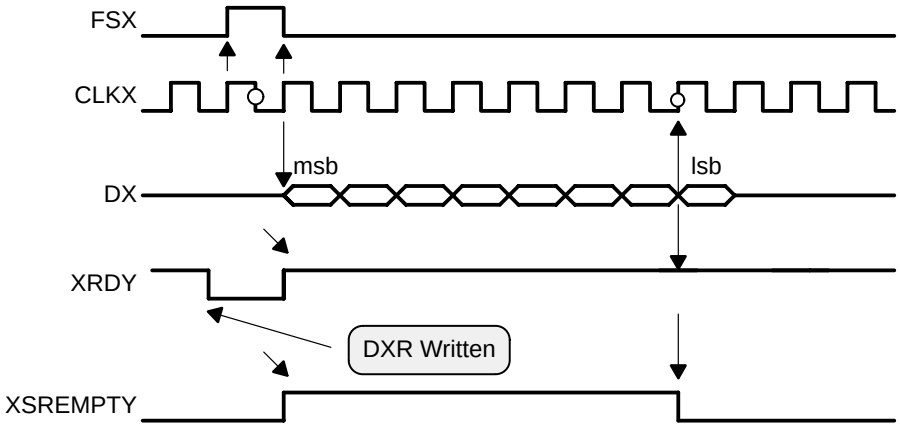


Figure 1–31. Transmit Burst Mode Internal Frame(Format Is 8 Bits)



Continuous Mode (FSM=0) With External Frame SYNC (TXM=0)

In the continuous mode with external frame sync (Figure 1–33, Figure 1–34, and Figure 1–35), only the first frame sync signal is necessary to start and transmit consecutive packets. As long as DXR is updated once every transmission, the continuous mode will continue. Failing to update will cause the SPI process to reach the SPI_END state (DX high impedance driven and XSREMPTY = 0). In this case,new frame sync pulse is required to restart SPI. If a frame sync pulse occurs after the initial one, SPI is restarted if FIG bit (Frame Ignore Bit) of SPCE register is 0; if FIG=1, this frame sync will be ignored. Setting FIG bit to 1 allows, for instance, transmitting in a continuous way in 16 bits format whereas frame sync occur every 8,10, or 12 bits. This can result in significant gain for buffer size in auto-buffered mode and standard mode and significant CPU cycles gain in standard mode. Figure 1–35 is showing example with 16 bits format and sync pulse every byte.

Figure 1–32. SPI Transmit Process for Continuous Mode With External Frame (FSM=0 and TXM=0)

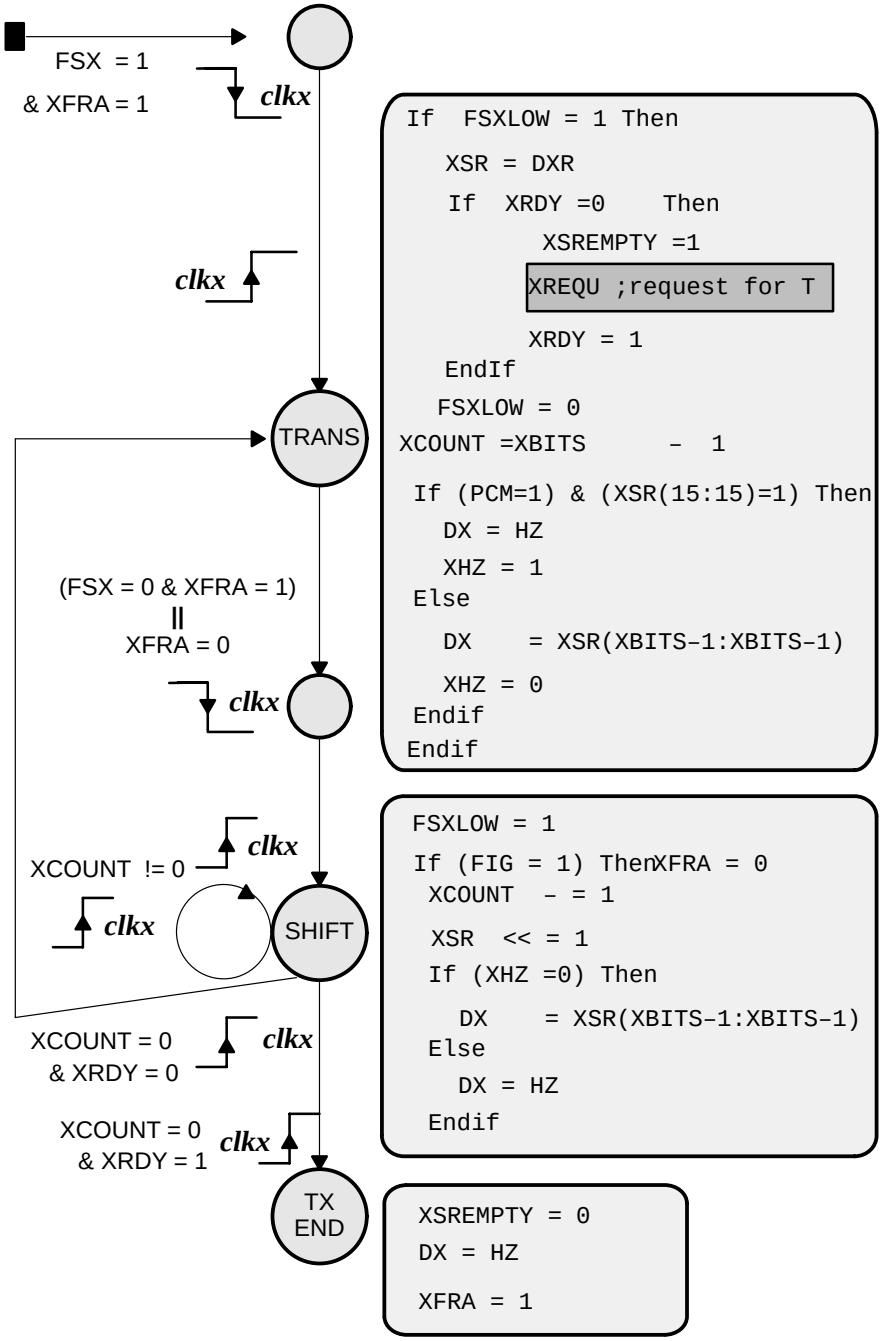


Figure 1–33. Transmit Continuous Mode With External Frame (Format Is 8 Bits)

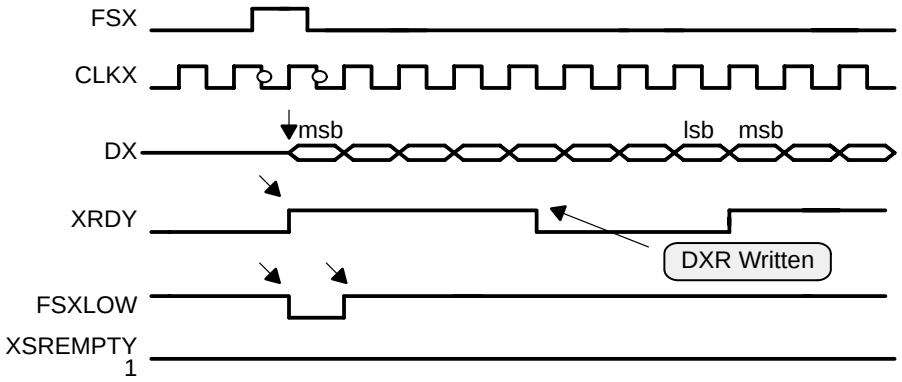


Figure 1–34. Transmit Continuous Mode With External Frame Transmission Stop (Format Is 8 Bits)

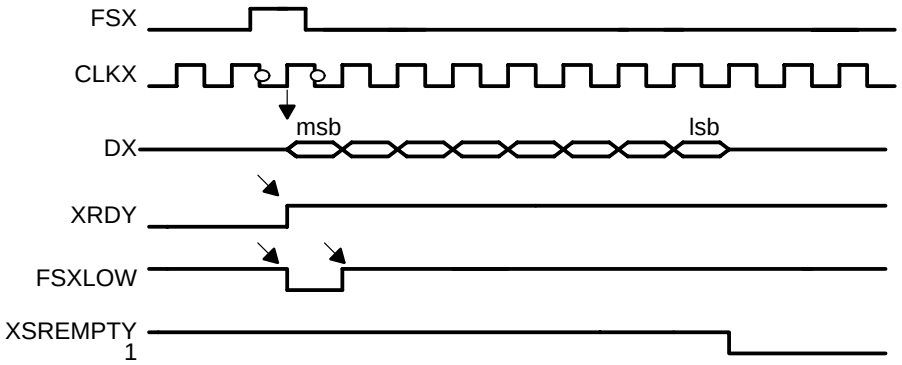
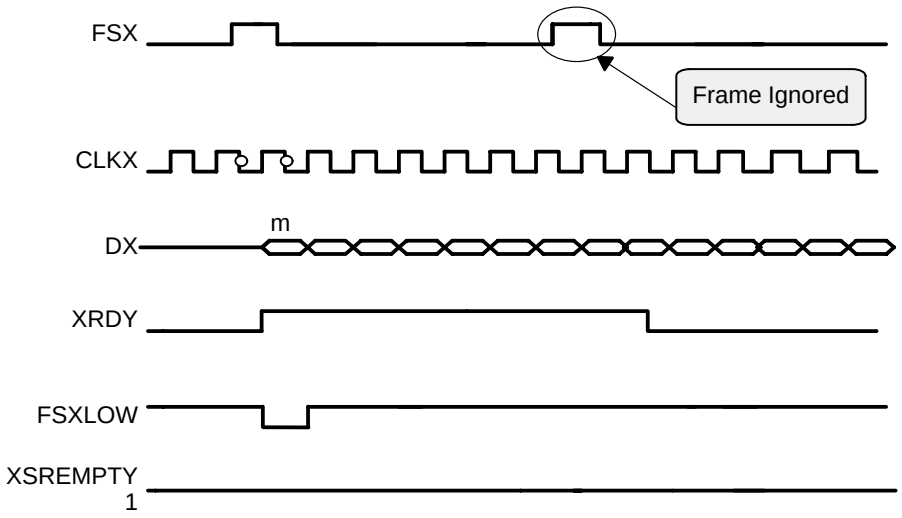


Figure 1–35. Transmit Continuous Mode With External Frame and FIG =1 (Format Is 16-bits)



Continuous Mode (FSM=0) With Internal Frame SYNC (TXM=1)

In this mode, Frame Sync Pulse is generated internally (Figure 1–36 and Figure 1–37) at the Transmit Serial Clock rising edge following a write to DXR register (XRDY going low). The Frame Sync Pulse is active during one full clock period. For further steps, SPI process is similar to External Frame mode.

As long as DXR is updated once every transmission,the continuous mode will continue. Failing to update will cause the SPI process to reach the SPI_END state (DX high impedance driven and XSREMPTY = 0). In this case, SPI will be restarted as soon as DXR will be written (XRDY going to 0).

Figure 1–36. Transmit Continuous Mode With Internal Frame(FSM=0 and TXM=1)

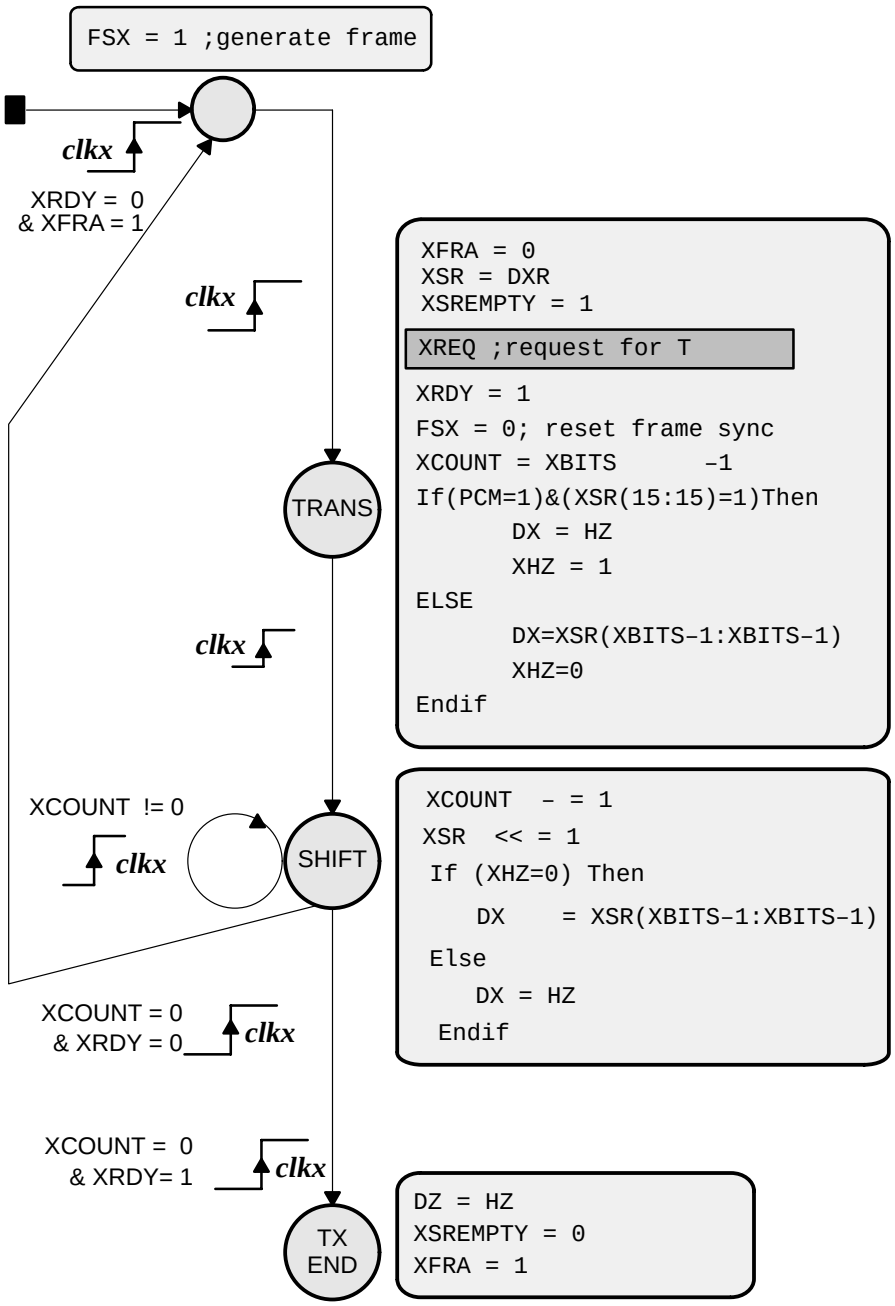
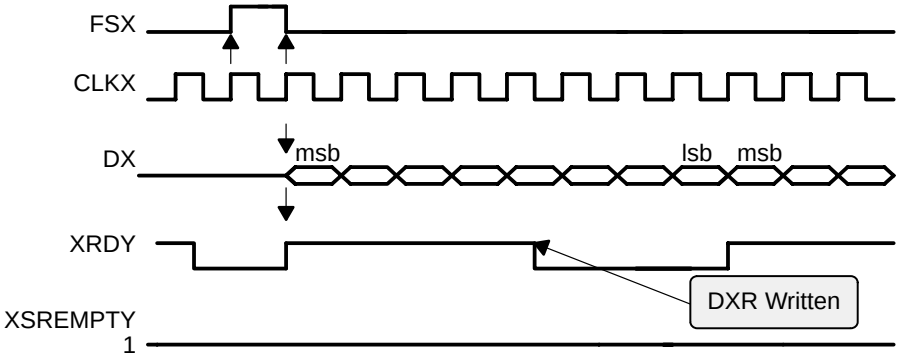


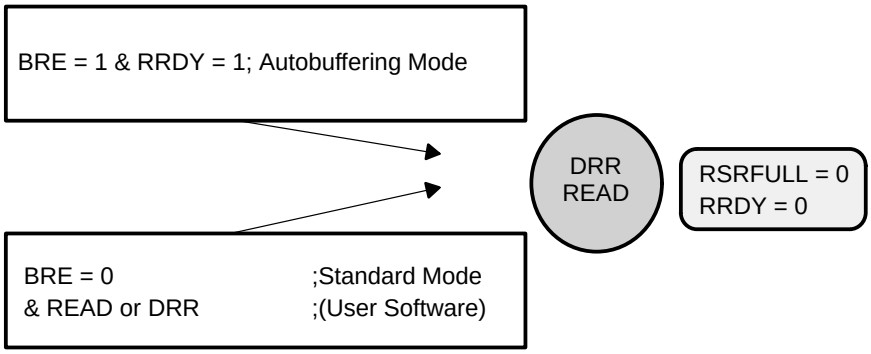
Figure 1–37. Transmit Continuous Mode With Internal Frame (Format Is 8 Bits)



1.2.2.3 Serial Port Interface Receive Operation

Two processes take place to ensure reception,a process of filling DRR register under control of the SPI and a process of emptying DRR register under control of user software in Standard Mode or under control of the autobuffering unit (ABU) in auto buffered mode. In order to synchronize and monitor these processes,two signals are used in SPC register. These signals are RRDY (Receive Ready) and RSRFULL (Receive Shift Register Full).Emptying process is illustrated with Figure 1–38 and SPI filling process is illustrated with Figure 1–39 (case is Burst Mode). When DRR is read (emptying process), RRDY signal is cleared and RSRFULL bit is cleared so that SPI process is informed that word has been read and new one can be received. When SPI is transferring this new word from RSR (Receive Shift Register) to DRR (Figure 1–39), RRDY bit is set to 1. This transfer occurs only if RSRFULL bit is cleared (Figure 1–39). If transfer cannot be performed because RSRFULL is still 1, then SPI process ends and can only be restarted if DRR is read. RSRFULL =1 may then be considered as overflow event. At the same time a transfer occurs, a Receive Request is sent to CPU as interrupt (RINT) in Standard Mode (BRE=0 in SPCE control register) or to the ABU in Auto Buffered Mode (BRE= 1 in BUFC register). In Standard Mode, the DRR should be read only if RRDY =1, which is guaranteed if the DRR read is made is response to a receive interrupt or polling RRDY.

Figure 1–38. DRR Emptying Process



The SPI supports two modes of reception,Burst Mode and Continuous Mode which are here below detailed. For all flowcharts and timing diagrams used for description and featuring frame and clock events,assumption is made that clock polarity (CLKP) is 0 and frame polarity (FSP) is 0.Operation with other polarity can be derived easily. Table 1–6 is giving the list of signals and registers that are used for the receive flowcharts.

Table 1–6. Receive Flowcharts Signals and Registers

Name	Description
FSR	Receive Frame Sync signal
CLKR	Receive Serial Clock .
DR	Receive Data Signal.
DRR	Data Receive Register.
RSR	Receive Shift Register.
RRDY	Receive Ready status/control bit of SPC register.
RSRFULL	Receive Shift Register Full bit of SPC register
FIG	Frame Ignore control bit of SPCE register.
RFRA	Receive Frame Acknowledge internal signal. This bit manages sync pulse acceptance/non acceptance during continuous mode when FIG=0/1 RFRA=1 upon device reset or upon reset of SPI Receive section (RRST=0).
RCOUNT	Internal Receive Counter for data shifting.
RBITS	Number of bits to receive (8,10,12 or 16).
RREQ	Receive Request (RINT in standard mode,request to ABU in auto buffered mode).

Burst Mode (FSM=1)

In this mode,data packet is marked by the Fame Sync Pulse on FSR. Some periods of inactivity may then occur between packets. The flowchart of Figure 1–39 is showing detailed SPI process embedding timing aspects. Figure 1–40 and Figure 1–41 show timing aspects for both short duration frame signal and long duration frame signal. SPI operation starts when frame signal becomes active,this event being sampled on falling edge of the receive clock. In the RX_START internal receive counter (RCOUNT) is loaded with number of bits to receive (8,10,12, or 16 bits).First bit (MSB) is shifted into Receive Shift Register at the falling edge of Receive Clock sampling Frame Sync Pulse (FSR) at low level. After all bits have been received, if RRDY bit is 0 (last DRR has been read), RRDY is set to 1 and Receive Interrupt (RINT) is sent to CPU in standard mode (BRE=0) or a request for emptying DRR is sent to ABU in auto buffered mode (BRE=1); if RRDY is still equal to 1, RSRFULL bit is set to 1. SPI is then restarted when next Frame Sync occurs only if RSRFULL is 0. Thanks to double buffering,reception can be maintained continuously (Figure 1–42). If a Frame Sync Pulse occurs during reception (Figure 1–43) reception is restarted and the bits that have been shifted in the RSR for the aborted reception are lost.

Figure 1–44 and Figure 1–45 are showing RSRFULL activation and RSRFULL deactivation, respectively.

Figure 1–39. Receive Burst Mode (FSM=1)

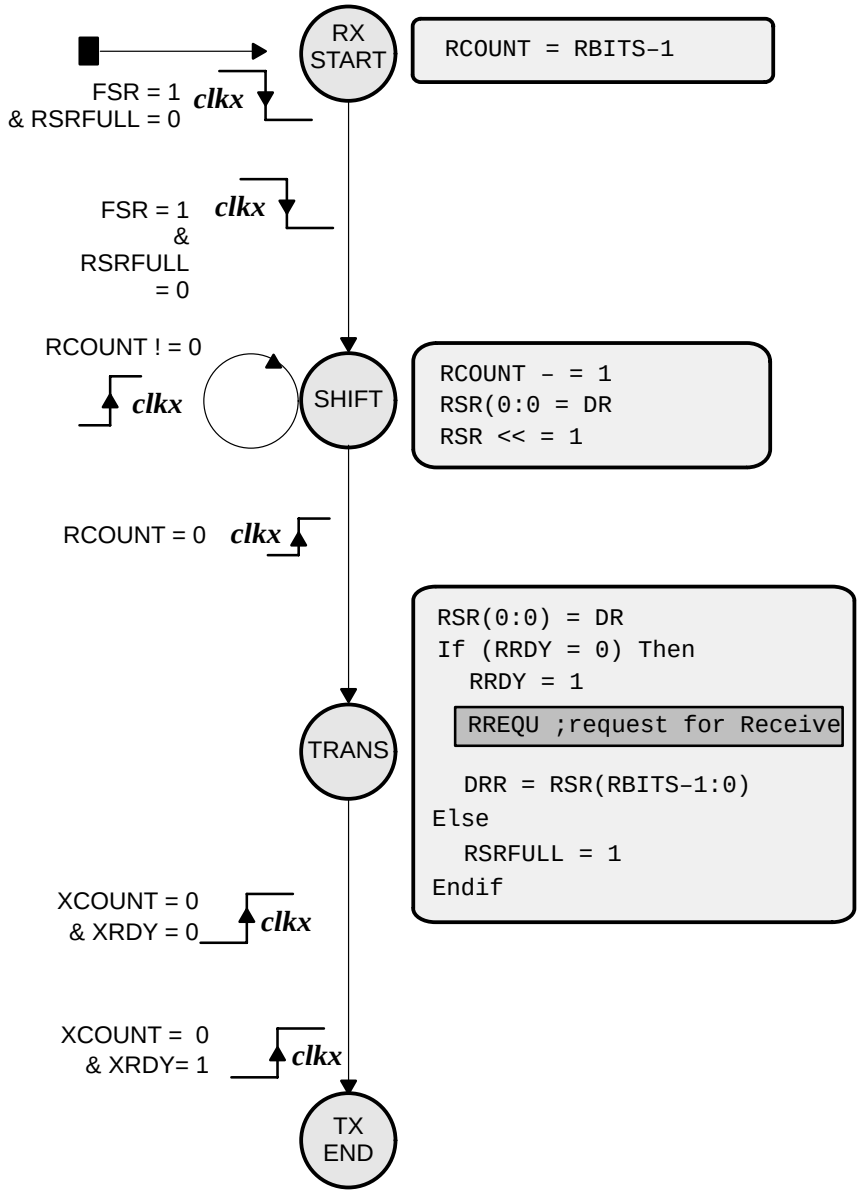
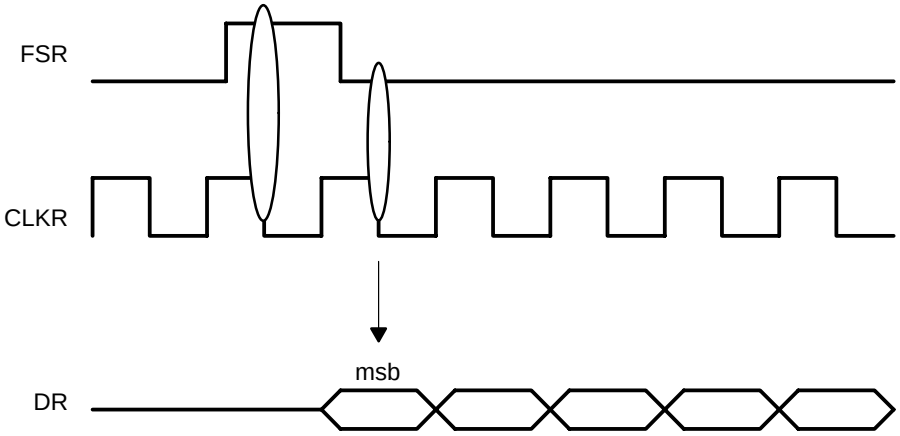


Figure 1–40. Short FSR Pulse

(a) Short FSR Pulse With FSR Going Low When CLKR Is High



(b) Short FSR Pulse With FSR Going Low When CLKR Is Low

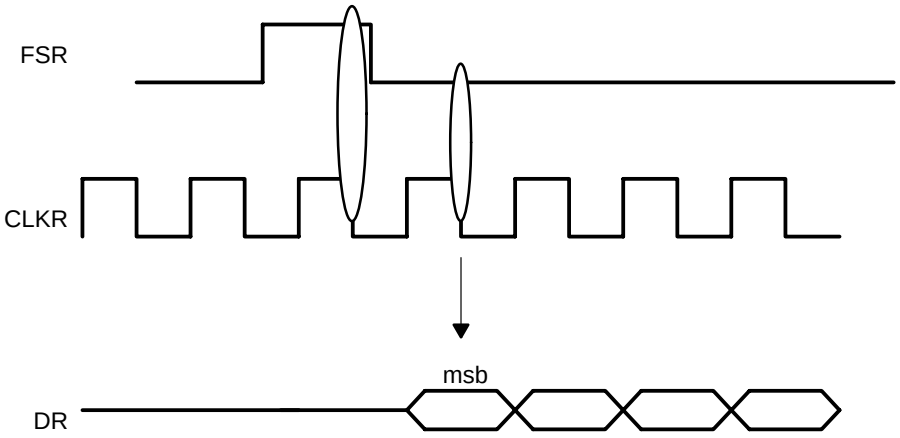
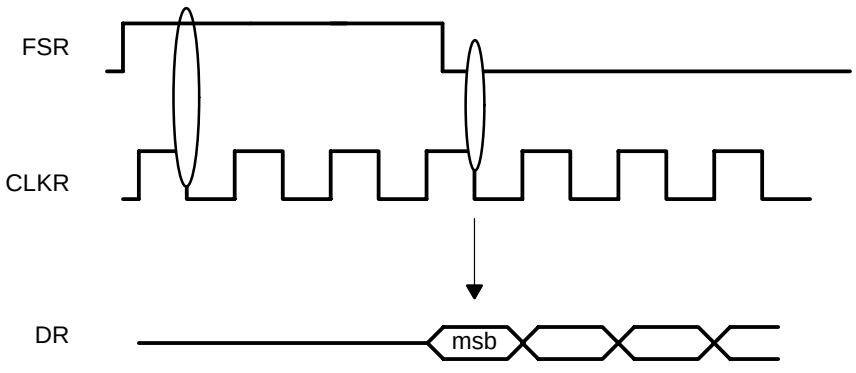


Figure 1–41. Long FSR Pulse

(a) Long FSR Pulse With FSR Going Low When CLKR Is High



(b) Long FSR Pulse With FSR Going Low When CLKR Is Low

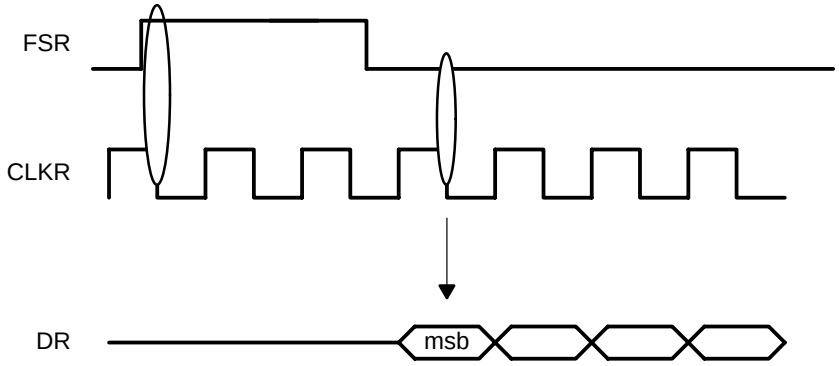


Figure 1–42. Receive Burst Mode—Continuous Reception (Format Is 8 Bits)

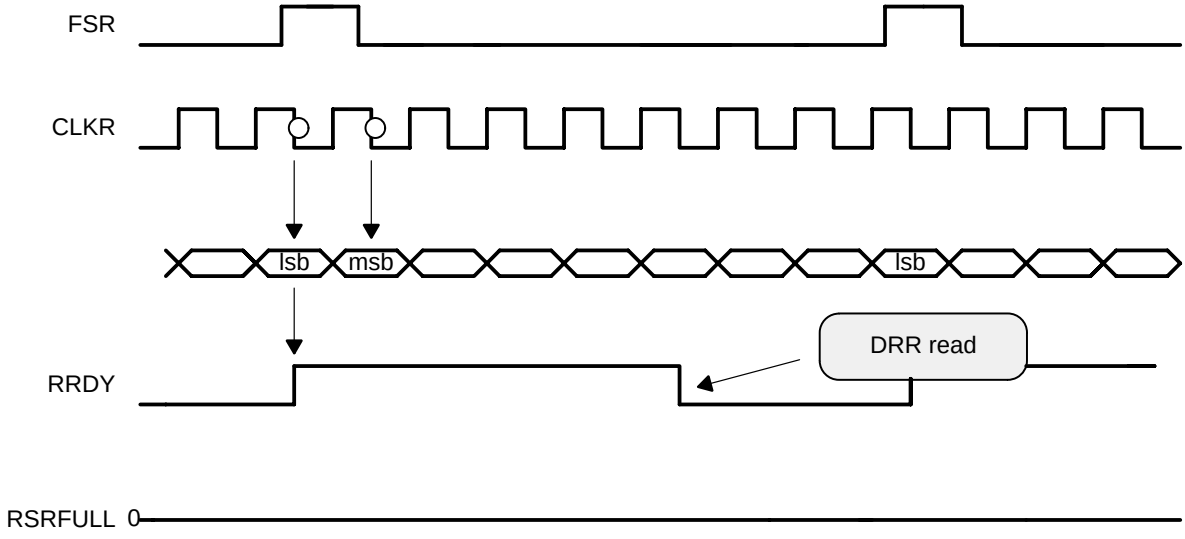


Figure 1–43. Receive Burst Mode With Reception Aborts

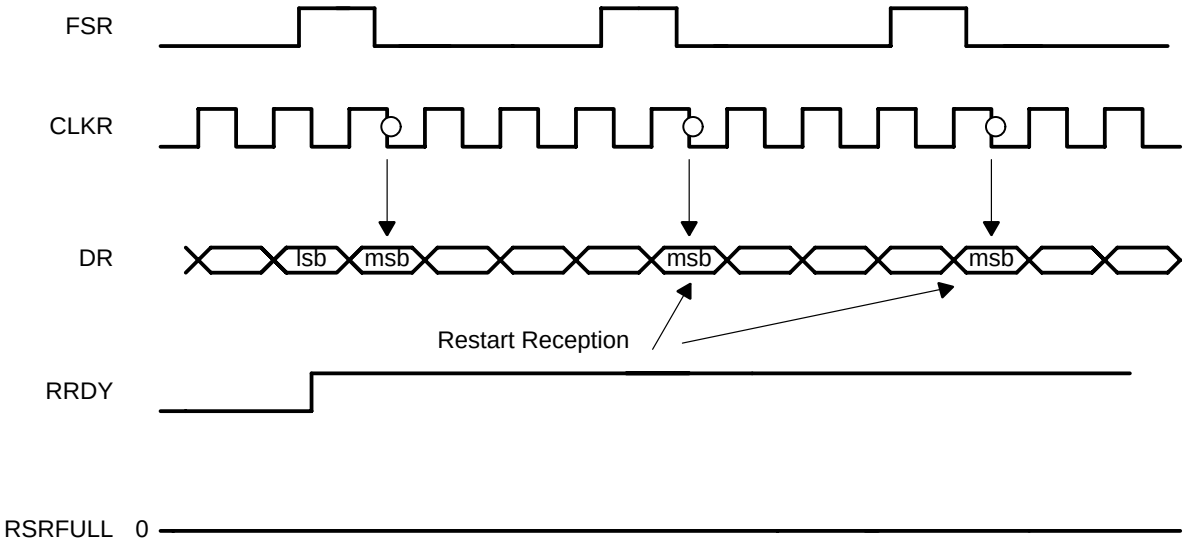


Figure 1–44. Receive Burst Mode—RSRFULL Activation (Format Is 8 Bits)

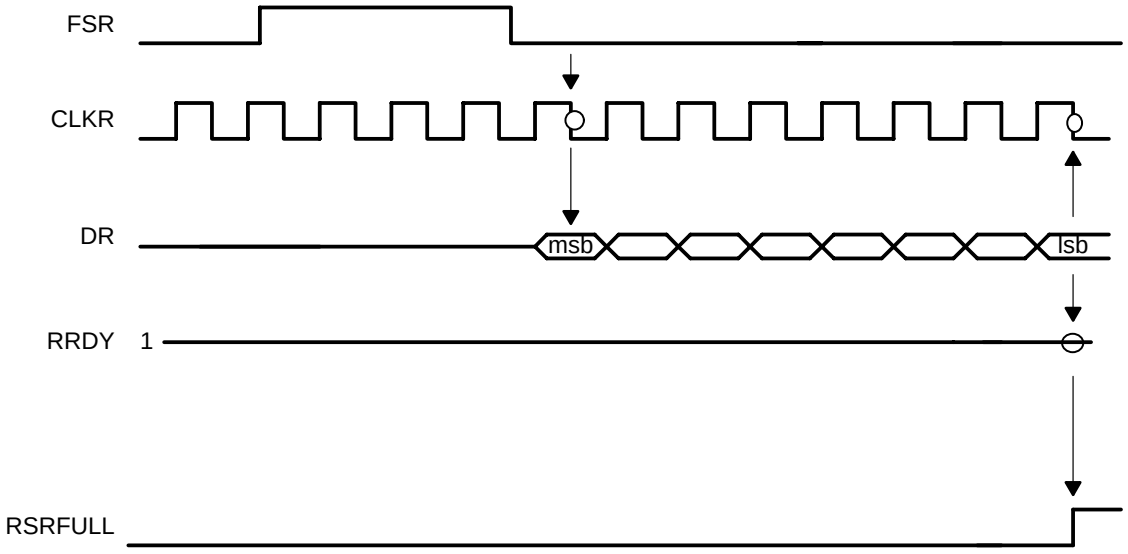
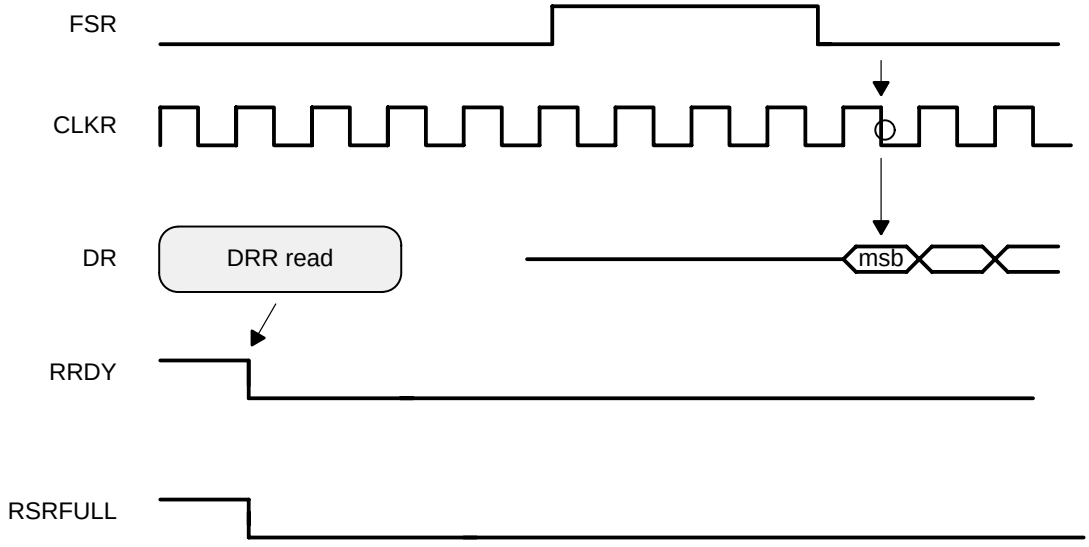


Figure 1–45. Receive Burst Mode—RSRFULL Deactivation



Continuous Mode (FSM=0)

In the continuous mode with external frame sync (Figure 1–46, Figure 1–47, Figure 1–48), only the first frame sync signal is necessary to start and receive consecutive packets. As long as DRR is read every reception,the continuous mode will continue. Failing to read will cause the SPI process to reach the RX_END state (RSRFULL =1).In this case,read of DRR and new frame sync pulse are required to restart SPI. If a frame sync pulse occurs after the initial one,SPI is restarted if FIG bit (Frame Ignore Bit) of SPCE register is 0;if FIG=1,this frame sync will be ignored. Setting FIG bit to 1 allows,for instance,receiving in a continuous way in 16 bits format whereas frame sync occur every 8,10, or 12 bits. This can result in significant gain for buffer size in auto-buffered mode and standard mode and significant CPU cycles gain in standard mode. Figure 1–49 is showing example with 16 bits format and sync pulse every byte.

Figure 1–46. Receive Continuous Mode (FSM=0)

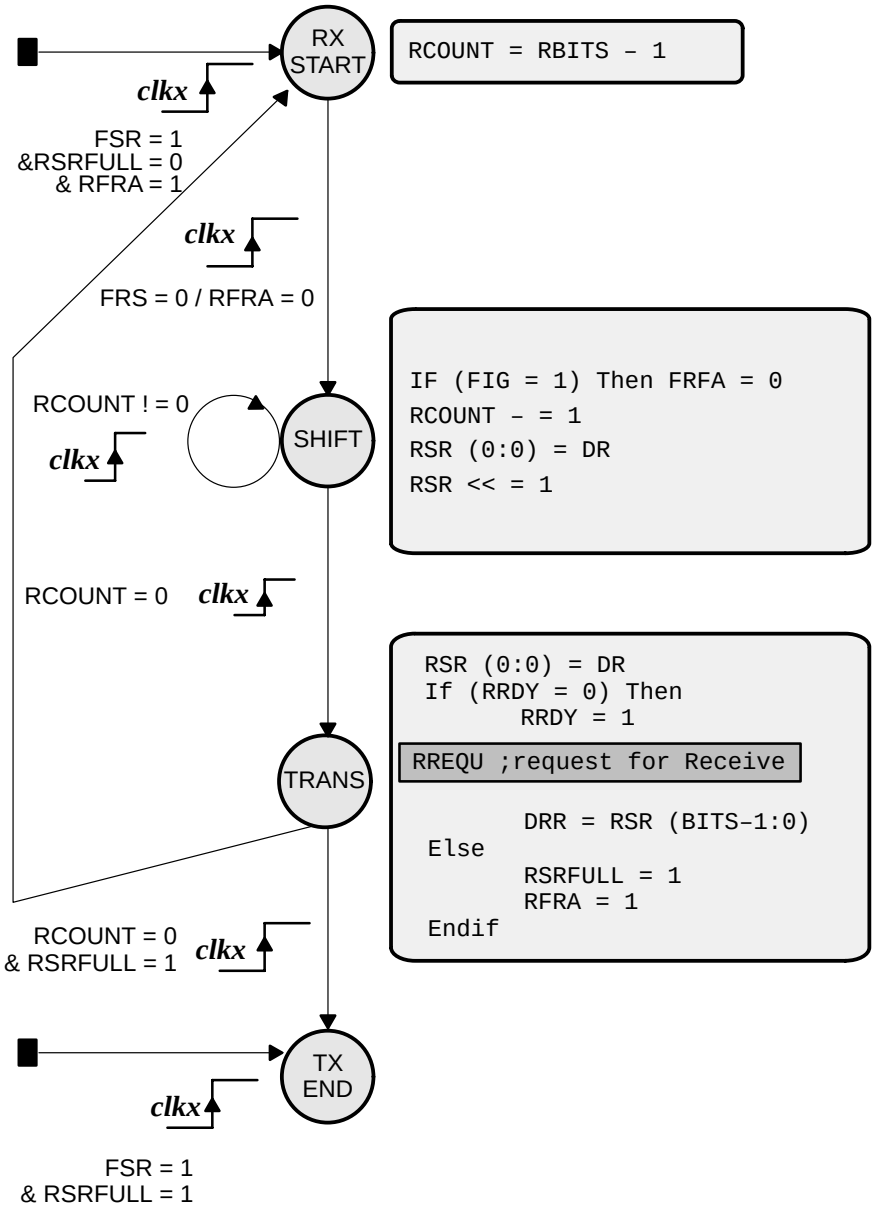


Figure 1–47. Receive Continuous Mode (Format Is 8 Bits)

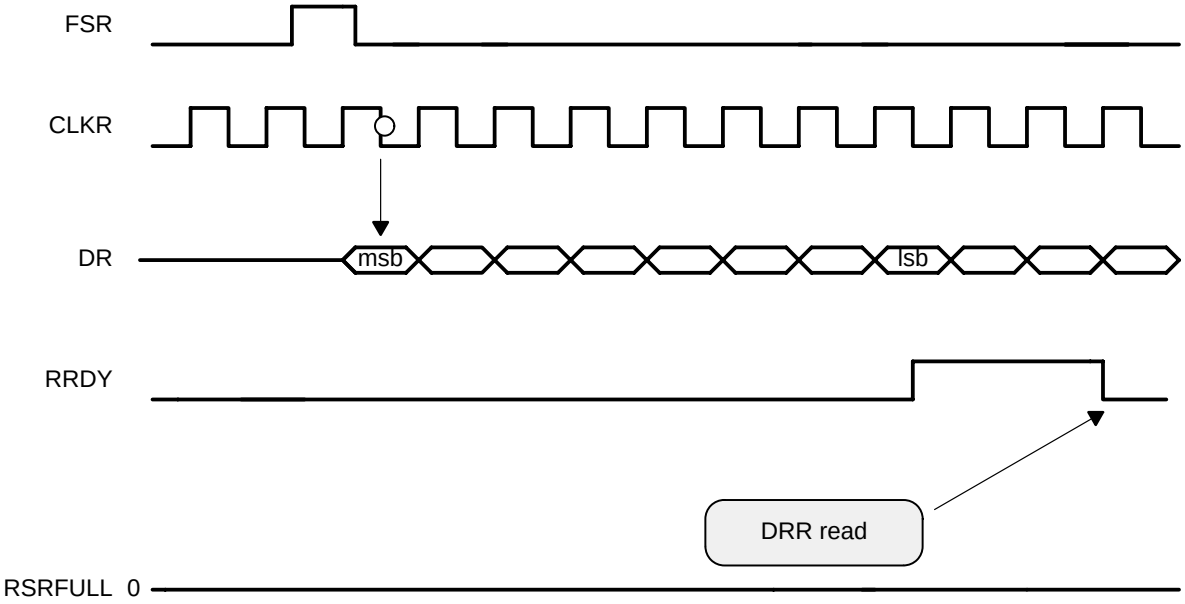


Figure 1–48. Receive Continuous Mode—Stop (Format Is 8 Bits)

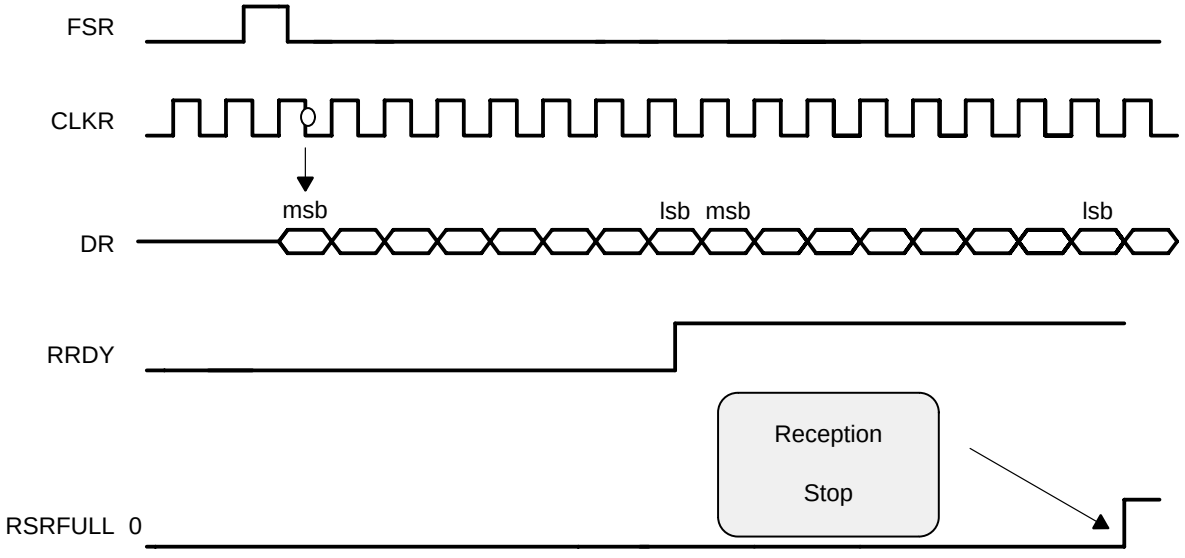
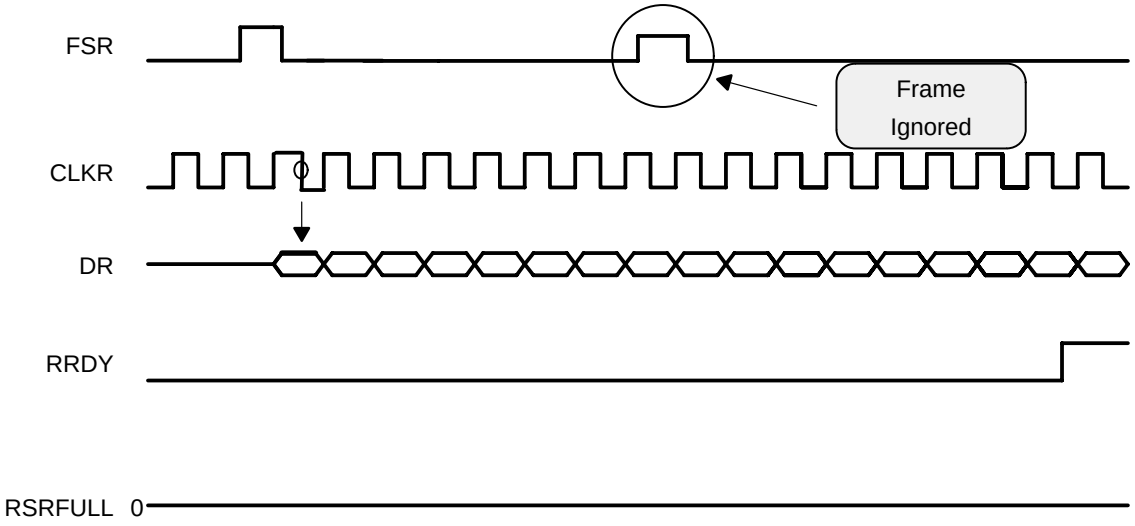


Figure 1–49. Receive Continuous Mode With FIG=1 (Frame Ignore, Format Is 16-Bits)



1.2.3 Autobuffering Unit (ABU)

Figure 1–50 is showing block diagram of Autobuffering Unit. The ABU is using five memory-mapped registers (SPCE, AXR, BKX, ARR and BKR). The Serial Port Control Extension Register (SPCE) controls the operation. The Address Transmit Register (AXR) and Block Size Transmit Register (BKX) associated with a circular addressing logic allow address generation for reading word to be transferred from ’C54x internal memory to SPI Data Transmit Register (DXR). Address Receive Register (ARR) and Block Size Receive Register (BKR) associated with a circular addressing logic allow address generation for writing Data Receive Register (DRR) of SPI in ’C54x internal memory. If the Autobuffering is not used,the 11 bits AXR, BKX, ARR and BKR registers can be used as general purpose registers. An interrupt mechanism is implemented in the ABU in order to interrupt CPU when transmit /receive buffer has been halfway or entirely emptied/filled. This mechanism features an auto disabling capability.

ABU transmit and receive parts can be enabled separately. When a part is disabled, it operates in Standard Mode. In standard mode, the ABU is transparent for the corresponding part. When autobuffering is enabled, the corresponding data serial port register (DXR or DRR) is no more available for access as memory mapped register by software.

Burst Mode and Continuous Mode as described in SPI operation can be run in conjunction with autobuffering capability.

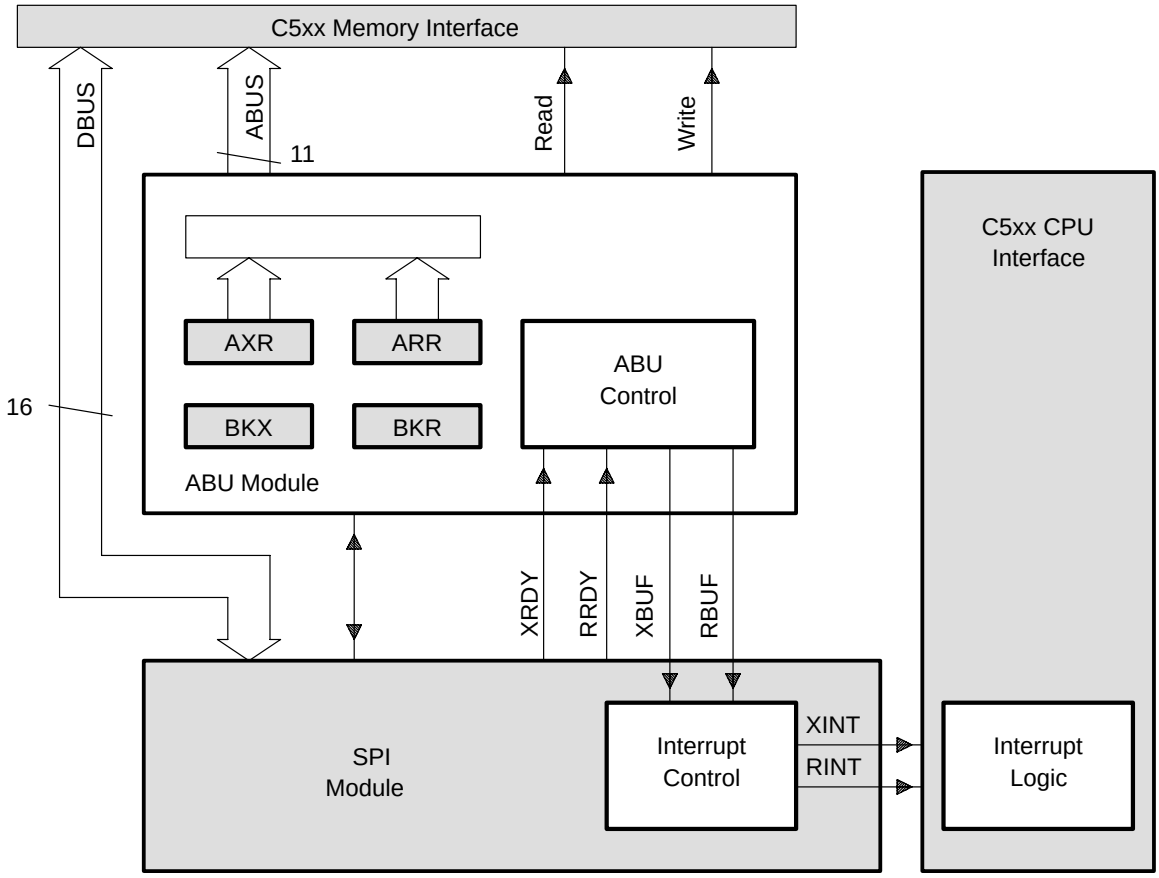
The internal ’C54x memory used for autobuffering consists in a 2K word block of dual access memory that can be configured by user as data or data/program, like the other dual access blocks. When the ABU is enabled, this memory block can still be addressed by CPU for accessing data or/and program. If the 2K word block is addressed at the same time by CPU and ABU, a memory access conflict may occur in some cases which fall basically in two categories:

- An ABU transmit access occurs at the same time as a CPU access via D or P internal buses (simple, dual, long data read or program read)
- An ABU receive access occurs at the same time as a CPU access via C or E internal buses (write or dual read operations)

These conflicts are automatically solved with one cycle penalty at the CPU side; priority is given to the ABU. Other combinations of ABU/CPU accesses at the same memory block don’t generate a conflict. Also, no conflict appears when ABU and CPU access different memory blocks.

When the ABU is enabled for transmit and receive and the transmit and receive requests from SPI happen at same time, the transmit request has priority over the receive request.

Figure 1–50. ABU Block Diagram



1.2.3.1 Autobuffering Control Register (SPCE)

Six bits in the SPCE control register configure the ABU. Some of the bits are read-only while others are read/write. Figure 1–51 shows the bit positions. A summary of each bit is given in Table 1–7.

Figure 1–51. ABUC Register

15	14	13	12	11	10	9 ... 0
HALTR	RH	BRE	HALTX	XH	BXE	SPI_Control
R/W	R	R/W	R/W	R	R/W	

Legend: **R** Read; **W** Write

Table 1–7. ABUC Register

Bit	Name	Function
9–0	SPIC	Serial Port Interface Control Bits (see paragraph 2.2)
10	BXE	Transmit Autobuffering Enable Bit. When BXE=1, autobuffering is enabled for transmitter. When BXE=0, autobuffering is disabled and SPI operates in Standard Mode. BXE=0 upon device reset.
11	XH	XH bit indicates which half of transmit buffer has been transmitted. For instance, it can be read when XINT interrupt occurs (interrupt program or IFR polling). XH=0 indicates that first half of buffer has been transmitted, XH=1 indicates that the second half of buffer has been transmitted. Device reset clears XH bit.
12	HALTX	When HALTX=1, autobuffering is halted when current half of buffer has been transmitted. At same time, BXE bit of SPCE register is cleared and serial port operation continues operation in standard mode. Device reset clears HALTX bit.
13	BRE	Receive Autobuffering Enable Bit. When BRE=1, autobuffering is enabled for receiver. When BRE=0, autobuffering is disabled and SPI operates in Standard Mode. BRE=0 upon device reset.
14	RH	RH bit indicates which half of receive buffer has been received It can be read when RINT interrupt occurs (interrupt program or IFR polling). RH=0 indicates that first half of buffer has been received, RH=1 indicates that the second half of buffer has been received. Device reset clears RH bit.
15	HALTR	When HALTR=1, autobuffering is halted when current half of buffer has been received. At same time, BRE bit of SPCE register is cleared and serial port continues operation in standard mode. Device reset clears HALTR bit.

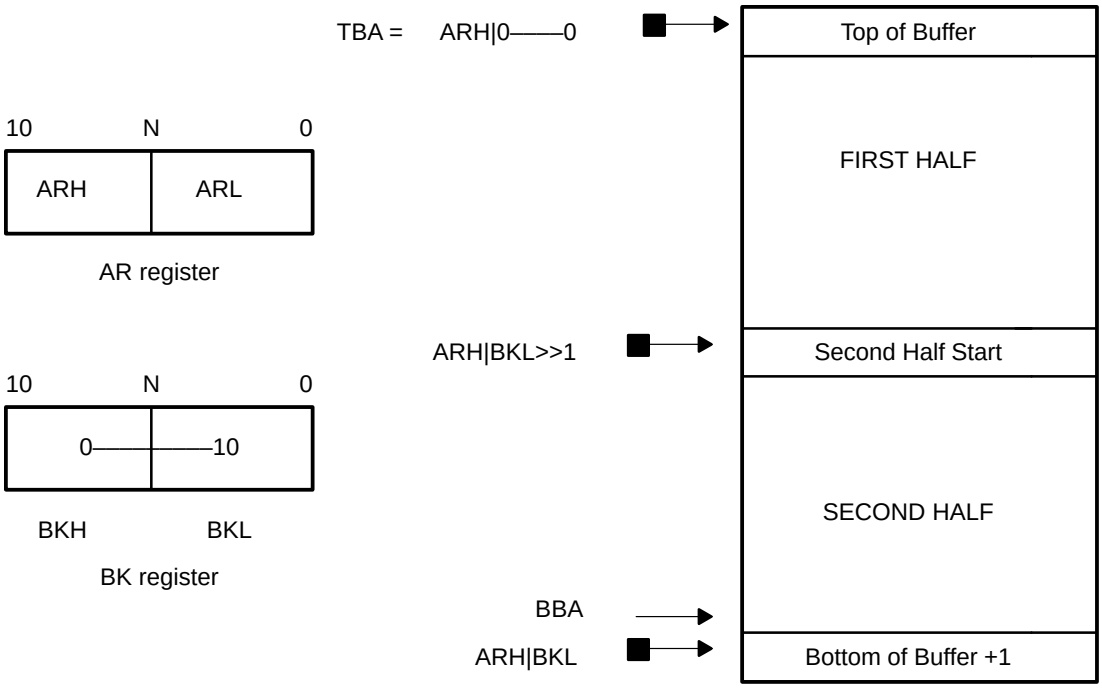
The value of the SPCE upon device reset is 0000000000000011b

1.2.3.2 Autobuffering Process

Buffers for autobuffering reside in an internal 2K words block of ’C54x internal memory. Top address and size of buffers within this block can be user programmable thanks to 11-bit address registers (AXR or ARR) and 11-bit block size registers (BKX and BKR). Transmit and receive can reside in same area which allows transmitting a buffer while receiving in same buffer, in overlapping areas or in different areas. Note that when the 11-bit memory mapped registers are read to form a 16-bit word, the 5 most significant bits are read as zeroes.

Circular addressing is key for auto buffered operation. Mechanism is the same for transmit and receive. For each direction (transmit or receive), two registers are specifying buffer size and current address in buffer. These registers are BKX and ARX for transmit, BKR and ARR for receive. The two registers (BK and AR) are fully specifying top and bottom of buffer. Figure 1–52 illustrates the relationship between the block size register (BK type), the address register (AR type), the bottom of the circular buffer (BBA) and the top of the circular buffer (TBA). The BK register contains the exact size of buffer. It can be split into two parts; higher part (BKH) corresponds to the BK most significant bits range with all bits equal to 0, the lower part is remaining bits with a 1 at the most significant bit position (position N). The N bit position is defining two parts (ARH and ARL) in address register. Top of buffer address (TBA) is defined as concatenation of ARH with (N+1) bits equal to 0 as least significant bits. The bottom of buffer address (BBA) is concatenation of ARH and BKL minus one (Figure 1–52). A circular buffer of size BK must then start on an N-bit boundary (i.e.,N least significant bits of address register are 0) where N is smallest integer that satisfies 2**(N+1) > (BK) or must starts at address 0 of ’C54x internal memory block. The buffer consists of two halves, address range for first half is [0,BKL/2–1] and [BKL/2,BKL–1] for second half (Figure 1–52).

Figure 1–52. Circular Addressing Registers



Notes: 1) Minimum size for BK is 2.

2) Maximum size for BK is 2047, such buffer needs to start at relative address 0x0000.

3) A buffer of size 1024 needs to start at relative address 0x0000.

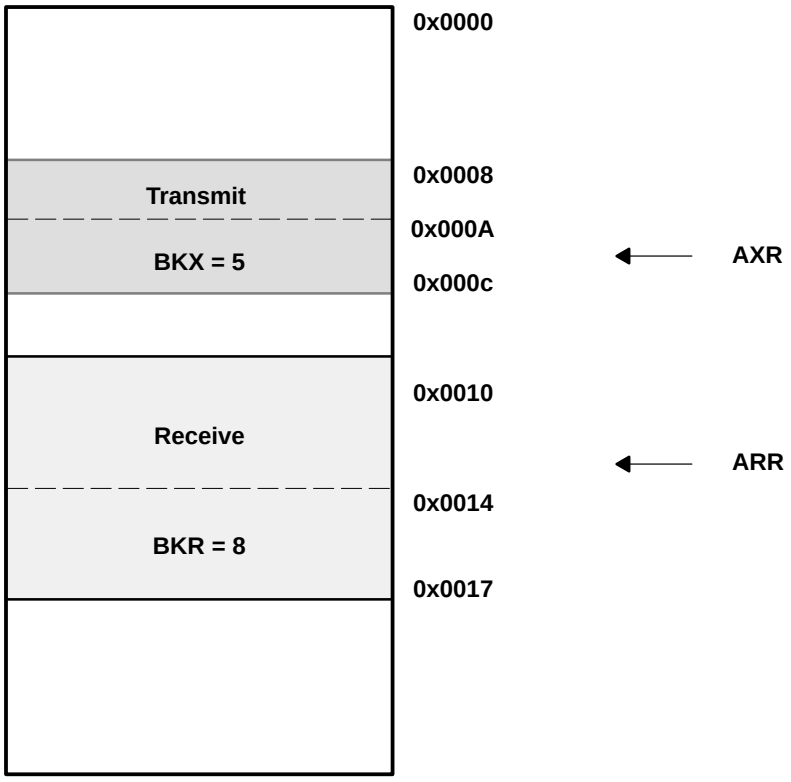
4) If address pointer (AXR or ARR) is initialized with value that does not fit with acceptable buffer range, the pointer will be incremented until it meets the next permitted buffer start address (see example below).

Example for a transmit buffer of size 5 (BKX=5) and a receive buffer of size 8 (BKR=8) is described in Figure 1–53. Transmit buffer may start at any relative address that is a multiplier of 8 (address 0x0000,0x0008,0x0010,0x0018 ..., 0x07f8), receive buffer may start at any relative address that is a multiplier of 16 (0x0000,0x0010,0x0020 .... ,0x07f0). In this example, transmit buffer starts at relative address 0x0008 and receive buffer starts at relative address 0x0010.

AXR pointer may be initialized with any value in [0x0008–0x000c] range and ARR pointer may be initialized with any value in [0x0010–0x0017] range.

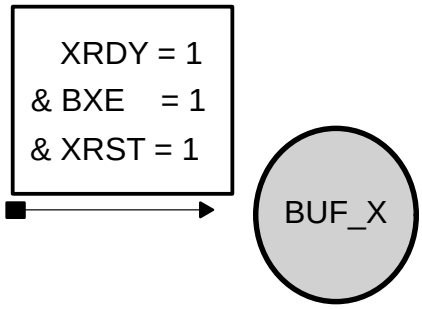
In this example, if AXR is initialized with value 0x000D (not in a modulo 5 acceptable buffer), it will be incremented until it reaches address 0x0010 which is acceptable value for a modulo 5 buffer.

Figure 1–53. Example for Transmit Buffer and Receive Buffer Mapping



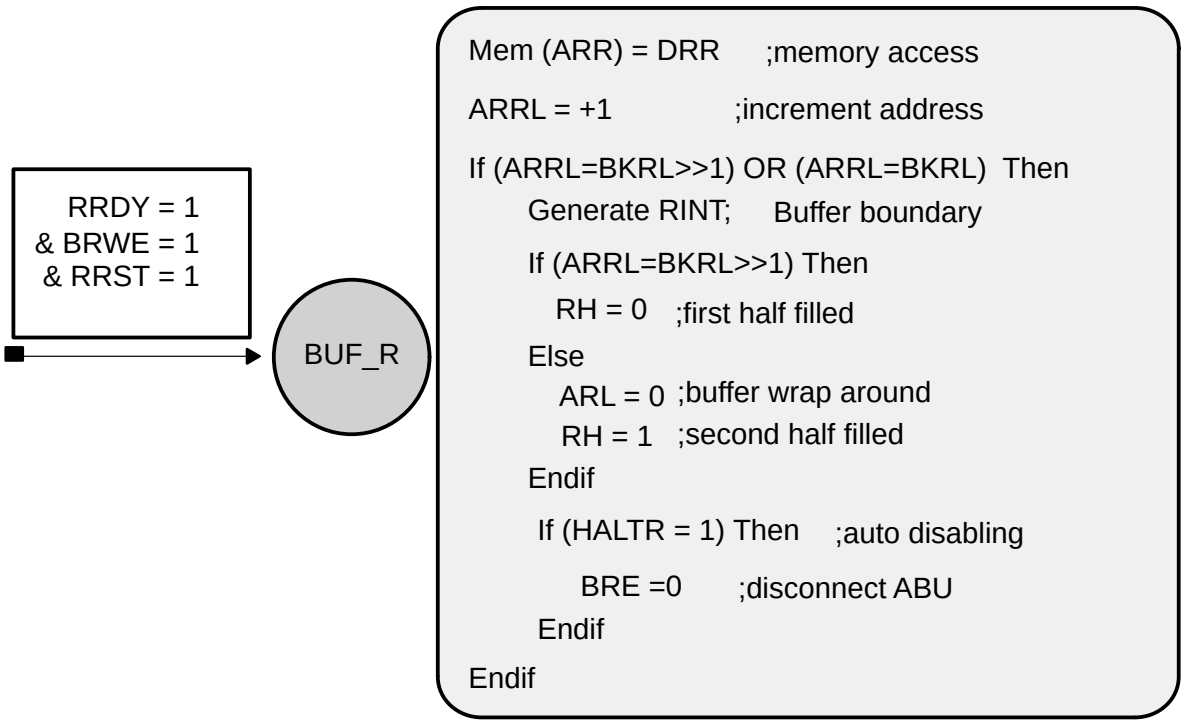
The autobuffering process is described in Figure 1–54 for transmit and in Figure 1–55 for receive. When process is activated upon request from SPI (XRDY=1 or RRDY=1), four actions are performed, first is 'C54x internal memory access, second is address register update, third is decision for interrupt and last is auto disabling management. Interrupt is generated when first half of buffer or second half of buffer is processed. Flags (RH and XH) in SPCE register allow user to know which half has been processed when interrupt occurs. When the auto disabling is on (HALT bit set), then when next buffer boundary is found, the autobuffering enable bit of SPCE (BXE or BRE) is cleared so that the autobuffering is stopped and does not generate any further request. When transmit autobuffering is stopped, some data are still to be sent (current XSR contents and last value loaded in DXR). Then SPI operation must not be stopped. If user wants to know that transmission is actually ended, he can test the condition XRDY=1 and XSREMPTY=0 which will occur after last bit has been transmitted.

Figure 1–54. Autobuffering Process for Transmit




```
DXR = Mem (AXR);memory access
AXRL = +1            ;increment address
If (AXRL=BKXL>>1) OR (AXRL=BKL)  Then
      Generate XINT;  Buffer boundary
      If (AXRL=BKXL>>1) Then
         XH = 0   ;first half emptied
      Else
         AXL = 0   ;buffer wrap around
         XH = 1    ;second half emptied
      Endif
       If (HALTX = 1) Then
          BXE  =0    ;disconnect ABU
      Endif
Endif
```

Figure 1–55. Autobuffering Process for Receive



1.2.4 Initialization of BSP Operation

In order to start or restart BSP operation in a given mode, following steps are to be done through DSP software. Assumption is made that transmit and receive interrupts are used, polling of IFR (Interrupt flag register) could also be used. Both transmit and receive sections can be initialized at same time.

- Transmit Interrupt Initialization
  - Set INTM to 1 to have global disabling of interrupts
  - Enable transmit interrupt in IMR (Interrupt Mask register)
- Transmit mode selection
  - Configure mode by writing SPC register with XRST =0
  - Configure additional modes and ABU if required by writing to SPCE register

  Once mode has been set up, internal hardware is initialized. The serial port clock is provided externally if mode is internal clock (MCM=1) and only if receiver is started (RRST = 1) The transmit line (DX) is in high impedance and the FSX line is in high impedance.

- Start Operation
  - Clear any transmit pending interrupt in IFR (Interrupt Flag Register)
  - Initialize address pointers (ABU or auxiliary registers)
  - Start transmit operation by writing XRST=1 is SPC register
  - In non auto buffered mode, load first word to transmit in DXR. In auto buffered mode, this will be automatically done.
  - Set INTM to 0 to have global enabling of interrupts
- Receive Interrupt Initialization
  - Set INTM to 1 to have global disabling of interrupts
  - Enable receive Interrupt in IMR (Interrupt Mask register)
- Receive Mode Selection
  - Configure mode by writing SPC register with RRST =0
  - Configure additional modes and ABU if required by writing SPCE register

  Once mode has been set up, internal hardware is initialized.
- Start operation
  - Clear any receive pending interrupt in IFR (Interrupt Flag Register)
  - Initialize address pointers (ABU or auxiliary registers)
  - Start receive operation by writing RRST=1 is SPC register
  - Set INTM to 0 to have global enabling of interrupts

1.2.4.1 Clock Synchronization

The SPI (Serial Port Interface) is operating with clocks that can be asynchronous with 'C54x internal clock and that can reach frequencies as high as 'C54x operating frequency.

In order to achieve this performance, transmit or receive operation start event done by software (transition of XRST or RRST from 0 to 1) needs to be re-synchronized with clock (CLKX or CLKR). Once this synchronization has occurred, the SPI can operate with respect to the clock (CLKX or CLKR).

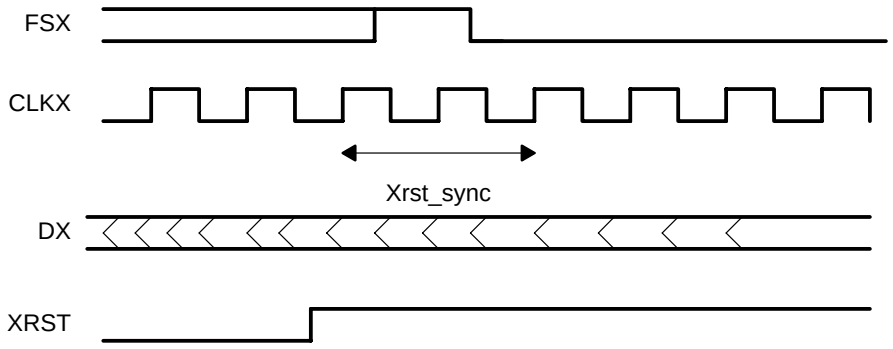
In order to achieve synchronization, two clock periods (of CLKX or CLKR) are needed after 0 to 1 transition of XRST or RRS. For transmit, this is valid with external or internal clock. For all modes except transmit with internal frame (TXM=1), these two cycles will result in frame signal (FSX or FSR) to be

ignored if the frame signal events (transitions) occur during these two cycles. For transmit mode with internal frame (TXM=1), if data is ready to be transmitted after XRST 0 to 1 transition, frame will be constructed after two CLKX period delay for the first transmission.

Examples here below are illustrating ignored frame pulses due to synchronization. Polarities are FSP=0 and CLKP=0.

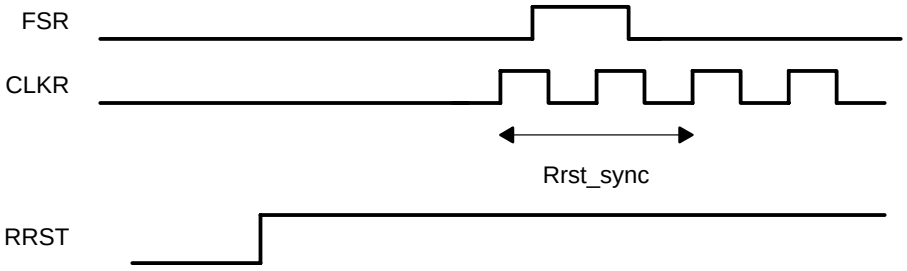
Example in Figure 1–56 is an external frame (TXM=1) and external clock mode (MCM=0) transmit operation. In this example, the FSX pulse occurs during the two cycles synchronization window, resulting in frame to be ignored (FSX pulse will not initiate transmission, DX stays high impedance).

Figure 1–56. FSX Pulse Occurs During Synchronization Window



Example in Figure 1–57 is a receive operation. In this example, the FSR pulse occurs during the two cycles synchronization window, resulting in frame to be ignored (FSR pulse will not initiate reception). In this case CLKR clock is applied in conjunction with FSR (CLKR is not active before FSR pulse), next FSR pulse following RRST 0 to 1 transition is ignored.

Figure 1–57. FSR Pulse Occurs During Synchronization Window



1.2.4.2 Memory-Mapped Registers and Interrupt Locations

Memory-mapped registers:

DRR	Address 0x20	Data Receive Register
DXR	Address 0x21	Data Transmit Register
SPC	Address 0x22	Serial Port Control Register
SPCE	Address 0x23	BSP Control Extension Register
AXR	Address 0x38	ABU Transmit Address Register
BKX	Address 0x39	ABU Transmit Buffer Size Register
ARR	Address 0x3A	ABU Receive Address Register
BKR	Address 0x3B	ABU Receive Buffer Size Register

Interrupt locations:

- The receive interrupt vector is located at address 0x50. Its associated mask/flag bit is number 4 within IMR[0:15] and IFR[0:15] registers.
- The transmit interrupt vector is located at address 0x54. Its associated mask/flag bit is number 5 within IMR[0:15] and IFR[0:15] registers.

1.2.4.3 Examples

For all examples given here below, entry conditions are: page is page 0 (DP=0) and interrupts are disabled (INTM=1).

Standard Mode

Example 1–3.Transmit Burst Mode With External Frame
(External Clock Format =10 bits; Polarities=0; PCM Mode Off)

```
ORM #020h,IMR      ;enable transmit interrupt (XINT)
STM #XTOP,AR1      ;init pointer with top of buffer
                   ;address
LD *AR1+,A         ;load accumulator with first word to
                   ;transmit
STM #08h,SPC       ;reset and configure serial port SPC
STM #080h,SPCE     ;configure serial port SPCE register
ORM #020h,IFR    ;clear any latched transmit interrupt
ORM #040h,SPC      ;start SPI transmit part
STLM A,DXR         ;load first word to transmit in data
                   ;transmit register (makes XRDY=0
                   ;so that interrupt is generated when
                   ;FSX gets active)
RSBX   INTM        ;enable interrupts
```

Example 1–4. Transmit Continuous Mode With Internal Frame (Internal Clock = 1/16 CLKOUT Frequency; Polarities = 0; Format= 12 Bits; PCM Mode Off)

```
ORM #020h,IMR      ;enable transmit interrupt (XINT)
STM #XTOP,AR1      ;init pointer with top of buffer
                   ;address
LD *AR1+,A         ;load accumulator with first word to
                   ;transmit
STM #034h,SPC      ;reset and configure serial port SPC
STM #08Fh,SPCE     ;configure serial port SPCE register
ORM #020h,IFR      ;clear any latched transmit interrupt
ORM #040h,SPC      ;start SPI transmit part
STLM A,DXR         ;load first word to transmit in data
                   ;transmit register (this makes XRDY=0
                   ;so that interrupt is generated when
                   ;FSX gets active). This write
                   ;generates FSX.
RSBX   INTM           ;enable interrupts
```

Example 1–5. Receive Burst Mode (Format= 16 Bits; Polarities=1)

```
ORM #010h,IMR      ;enable receive interrupt
STM #RTOP,AR2      ;init pointer with top of buffer
                   ;address
STM #08h,SPC       ;reset and configure serial port SPC
STM #060h,SPCE     ;configure serial port SPCE register
ORM #010h,IFR      ;clear any latched receive interrupt
ORM #080h,SPC      ;start SPI receive part
RSBX INTM             ;enable interrupts
```

Example 1–6. Receive Continuous Mode (Polarities = 1; Format= 8 Bits; Frame Ignore Is Set)

```
ORM #010h,IMR      ;enable receive interrupt
STM #RTOP,AR2      ;init pointer with top of buffer
                   ;address
STM #04h,SPC       ;reset and configure serial port SPC
STM #0160h,SPCE    ;configure serial port SPCE register
ORM #010h,IFR      ;clear any latched receive interrupt
ORM #080h,SPC      ;start SPI receive part
RSBX   INTM           ;enable interrupts
```

Autobuffering Mode

Example codes are shown below for initialization of SPI and ABU.

Example 1–7. Transmit Burst Mode With External Frame,External Clock Format= 10 bits, Polarities=0, PCM Mode Off

```
   ORM #020h,IMR      ;enable transmit interrupt (XINT)
   STM #08h,SPC       ;configure serial port SPC register
                      ;(XRST=0)
   STM #01480h,SPCE   ;configure serial port SPCE register
   STM #XTOP,AXR      ;init address of buffer start in AXR
   STM #XSIZE,BKX     ;init size of buffer
   ORM #020h,IFR      ;clear any latched transmit interrupt
   ORM #040h,SPC      ;start SPI transmit part (SPI=1)
RSBX  INTM            ;enable interrupts
```

Transmit autobuffering is on and transmission halts when half buffer is transmitted (HALTX=1).

Example 1–8. Receive Continuous Mode Polarities = 1, Format= 8 Bits, Frame Ignore Is Set, Receive Autobuffering Is Enabled

```
   ORM #010h,IMR      ;enable receive interrupt (RINT)
   STM 04h,SPC        ;reset and configure serial port SPC
                      ;(RRST=0)
   STM #02160h,SPCE   ;configure serial port SPCE register
   STM RTOP,ARR       ;init pointer with top of buffer
                      ;address
   STM #RSIZE,BKR     ;init size of receive buffer
   ORM v#010h,IFR     ;clear any latched receive interrupt
   ORM #080h,SPC      ;start SPI receive part
RSBX  INTM            ;enable interrupts
```

1.2.5 Operation During IDLE2

IDLE2 is a power down mode that is invoked by executing the “IDLE 2” instruction. During IDLE2 there is a complete shutdown of the core CPU and all CPU activities are halted. The on-chip peripheral circuits (timer, standard serial port) are also stopped.

The Buffered Serial Port is an exception. The receive side of the Buffered Serial Port will continue to operate during IDLE2. The transmit side of the Buffered Serial Port will continue to operate during IDLE2 at the following conditions:

- The serial port clock CLKX and frame FSX are external signals.
- The serial port clock CLKX is an external signal, the serial port frame FSX is internal and is generated before IDLE2 instruction is executed, and transmission is set in continuous mode.

To ensure that FSX pulse is generated, there must be a minimum of two cycles in the assembler code between the instruction which starts the transmission and the “IDLE 2” instruction. Once IDLE2 mode is entered, no extra FSX pulses will be generated from the DSP.

In autobuffering mode, the XINT transmit interrupt and RINT receive interrupt can be used to terminate the IDLE2 power-down mode. These interrupts are generated when first half of buffer or second half of buffer is processed.

When the BSP is programmed in standard mode, XINT and RINT interrupts can also be used to terminate IDLE2 state. In standard mode, these interrupts are generated when one word is processed.

In either case (autobuffering or standard mode), XINT or RINT must be enabled in IMR register to wake up the whole device. If INTM=0, the processor enters the corresponding interrupt service routine. If INTM=1, the processor continues with the instruction following the IDLE2 instruction.

1.3 Time-Division-Multiplexed (TDM) Serial Port

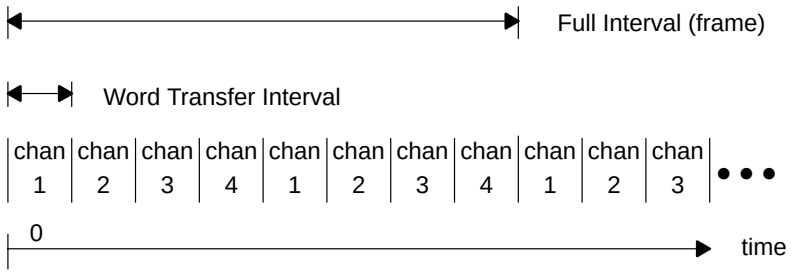
The TDM (time-division-multiplexed) serial port allows the ‘C54x device to communicate serially with up to seven other devices. The TDM port, therefore, provides a simple and efficient interface for multiprocessing applications.

The TDM serial port is a superset of the serial port described in Section 1.1. By means of the TDM bit in the TSPC control register, the port can be configured in multiprocessing mode (TDM=1) or stand-alone mode (TDM=0). When in stand-alone mode, the port operates as described in Section 1.1. When in multiprocessing mode, the port behaves as described in this section. The port can be shut down for low power consumption via the XRST and RRST bits, as described in Section 1.1.

1.3.1 Basic TDM Operation

Time-division multiplexing is the division of time intervals into a number of subintervals, with each subinterval representing a communications channel according to a prespecified arrangement. Figure 1–58 shows a 4-channel TDM scheme. Note that the first time slot is labeled chan 1 (channel 1), the next chan 2 (channel 2), etc. Channel 1 is active during the first communications period and during every fourth period thereafter. The remaining three channels are interleaved in time with channel 1, as shown in the figure.

Figure 1–58. Time-Division Multiplexing

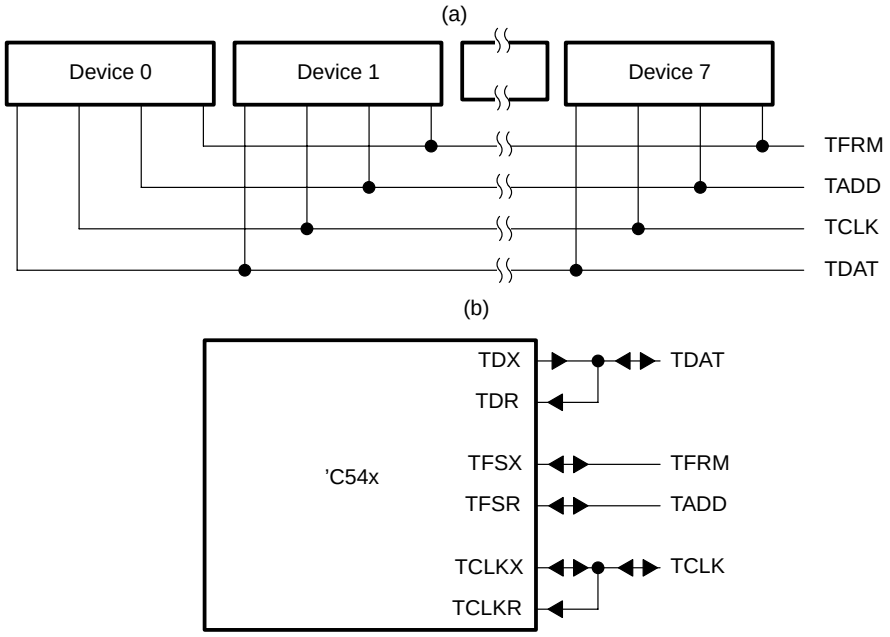


The ’C54x TDM port uses eight TDM channels. Which device is to transmit, and which device or devices is/are to receive for each channel may be independently specified. This results in a high degree of flexibility in interprocessor communications.

1.3.2 TDM Serial Port Interface Operation

Figure 1–59(a) shows the ’C54x TDM port architecture. Up to eight devices can be placed on the four-wire serial bus. This four-wire bus consists of a conventional serial port’s bus of clock, frame, and data (TCLK, TFRM, and TDAT) wires plus an additional wire (TADD) that carries the device addressing information. Note that the TDAT and TADD signals are bidirectional signals and are often driven by different devices on the bus during different time slots within a given frame of operation.

Figure 1–59. TDM Four-Wire Bus



The TADD line, which is driven by a particular device for a particular time slot, determines which device(s) in the TDM configuration should execute a valid TDM receive during that time slot. This is similar to a valid serial port read operation, as described in Section 1.1, except that some corresponding TDM registers are named differently. The TDM receive register is TRCV, and the TDM receive shift register is TRSR. Data is transmitted on the bidirectional TDAT line.

Note that in Figure 1–59(b). the device TDX and TDR pins are tied together externally to form the TDAT line. Also, note that only one device can drive the data and address line (TDAT and TADD) in a particular slot. All other devices’ TDAT and TADD outputs should be in the high-impedance state during that slot, which is accomplished through proper programming of the TDM port con-

trol registers (this is described in detail later in this section). Meanwhile, in that particular slot, all the devices (including the one driving that slot) sample the TDAT and TADD lines to determine if the current transmission represents valid data to be read by any one of the devices on the bus (this is also discussed in detail later in this section). When a device recognizes an address to which it is supposed to respond, a valid TDM read then occurs, the value is transferred from TRSR to the TRCV register. A receive interrupt is generated, which indicates that TRCV has valid receive data and can be read.

All TDM port operations are synchronized by the TCLK and TFRM signals. Each of them are generated by only one device (typically the same device), referred to as the TCLK and TFRM source(s). The word master is not used here because it implies that one device controls the other, which is not the case, and TCSR must be set to prevent slot contention. Consequently, the remaining devices in the TDM configuration use these signals as inputs. Figure 1–59(b) shows that TCLKX and TCLKR are externally tied together to form the TCLK line. Also, TFRM and TADD originate from the TFSX and TFSR pins respectively. This is done to make the TDM serial port also easy to use in standard mode.

TDM port operation is controlled by six memory-mapped registers. The layout of these registers is shown in Figure 1–60. The TRCV and TDXR registers have the same functions as the DRR and DXR registers respectively, described in Section 1.1. The TSPC register is identical to the SPC register except that bit zero serves as the TDM mode enable control bit in the TSPC. This bit configures the port in TDM mode (TDM=1), or stand-alone mode (TDM=0), in which the port operates as a standard serial port as described in Section 1.1. Refer to subsection 1.2.2 for additional information about the function of the bits in these registers.

Figure 1–60. TDM Port Registers

	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TRCV	Receive Data															
TDXR	Transmit Data															
TSPC	FREE	SOFT	X	X	XRDY	RRDY	IN1	IN0	RRST	XRST	TXM	MCM	FSM	FO	DLB	TDM
TCSR	X	X	X	X	X	X	X	X	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
TRTA	TA7	TA6	TA5	TA4	TA3	TA2	TA1	TA0	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0
TRAD	X	X	X2	X1	X0	S2	S1	S0	A7	A6	A5	A4	A3	A2	A1	A0

When TDM mode is selected, the DLB and FO bits in the TSPC are hard-configured to 0, resulting in no access to the digital loopback mode and in a fixed word length of 16 bits (a different type of loopback is discussed in the example in subsection 1.3.5). Also, the value of FSM does not affect the port when TDM=1, and the states of the underflow and overrun flags are indeterminate

(subsection 1.3.4 explains how errors are handled in TDM mode). If TDM=1, changes made to the contents of the TSPC become effective upon completion of channel 7 of the current frame. Thus the TSPC value cannot be changed for the current frame; any changes made will take effect in the next frame.

The source device for the TCLK and TFRM timing signals is set by the MCM and TXM bits, respectively. The TCLK source device is identified by setting the MCM bit of its TSPC register to 1. Typically, this device is the same one that supplies the TDM port clock signal TCLK. The TCLKX pin is configured as an input if MCM=0 and an output if MCM=1. In the latter case (internal 'C54x clock), the device whose MCM=1 supplies the clock (TCLK frequency=one fourth of CLKOUT1 frequency) for all devices on the TDM bus. The clock can be supplied by an external source if MCM=0 for all devices. TFRM can also be supplied externally if TXM=0. An external TFRM, however, must meet TDM receive timing specifications with respect to TCLK for proper operation. No more than one device should have MCM or TXM set to 1 at any given time. The specification of which device is to supply clock and framing signals is typically made only once, during system initialization.

The TDM channel select register (TCSR) of a given device specifies in which time slot(s) that device is to transmit. A 1 in any one or more of bits 0–7 of the TCSR sets the transmitter active during the corresponding time slot. Again, a key system-level constraint is that no more than one device can transmit during the same time slot; devices do ***not*** check for bus contention, and slots must be consistently assigned. As in TSPC operation, a write to TCSR during a particular frame is valid only during the next frame. However, a given device can transmit in more than one slot. This is discussed in more detail in subsection 5.6.3, with an emphasis on the utilization of TRTA, TDXR, and TCSR in this respect.

The TDM receive/transmit address register (TRTA) of a given device specifies two key pieces of information. The lower half specifies the receive address of the device, while the upper half of TRTA specifies the transmit address. The receive address (LSB RA0 to MSB RA7 refer to Figure 1–60) is the 8-bit value that a device compares to the 8-bit value it samples on the TADD line in a particular slot to determine whether it should execute a valid TDM receive. The receive address, therefore, establishes the slots in which that device may receive, dependent on the addresses present in those slots, as specified by the transmitting devices. This process occurs on each device during every slot.

The transmit address, (LSB TA0 to MSB TA7, refer to Figure 1–60), is the address that the device drives on the TADD line during a transmit operation on an assigned slot. The transmit address establishes which receiving devices may execute a valid TDM receive on the driven data.

Only one device at a time can drive a transmit address on TADD. Each processor bit-wise-logically-ANDs the value it samples on the TADD line with its receive address (RA7–RA0). If this operation results in a nonzero value, then a valid TDM receive is executed on the processor(s) whose receive addresses match the transmitted address. Thus, for one device to transmit to another, there must be at least one bit in the upper half of the transmitting device's TRTA (the transmit address) with a value of 1 that matches one bit with a value of 1 in the lower half of TRTA (the receive address) of the receiving device. This method of configuration of TRTA allows one device to transmit to one or more devices, and for any one device to receive from one or more than one transmitter. This can also allow the transmitting device to control which devices receive, without the receive address on any of the devices having to changed.

The TDM receive address register (TRAD) contains various information regarding the status of the TADD line which can be polled to verify the previous values of this signal and to verify the relationship between instruction cycles and TDM port timing.

Bits 13–11 (X2–X0) contain the current slot number value, regardless of whether a valid data receive was executed in that slot or not. This value is latched at the middle of the slot and retained only until the middle of the slot.

Bits 10–8 (S02–S0) hold the number of the last slot plus one (module eight) in which data was received (that is, if the last valid data read occurred in slot five in the previous frame, these bits would contain the number six). This value is latched during the TDM receive interrupt (TRNT) at the end of the slot in which the last valid data receive occurred, and maintained until the end of the next slot in which a valid receive occurs.

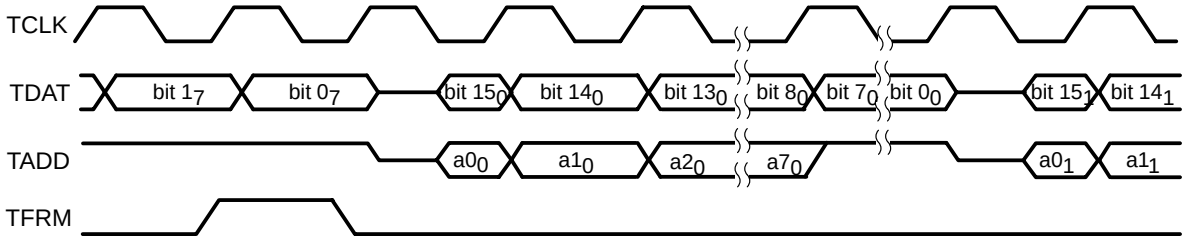
Bits 7–0 (A7–A0) hold the last address sampled on the TADD line, regardless of whether a valid data receive was executed or not. This value is latched halfway through each slot (so the value on the TADD may be shifted in) and maintained until halfway through the next slot, whether a valid receive is executed or not.

1.3.3 TDM Mode Transmit and Receive Operations

Figure 1–61 shows the timing for TDM port transfers. The TCLK and TFRM signals are generated by the timing source device. The TCLK frequency is one fourth the frequency of CLKOUT1 if generated by a 'C54x device. The TFRM pulse occurs every 128 TCLK cycles, and is timed to coincide with bit zero of slot seven, or with the last bit of the previous frame. The relationship of TFRM and TCLK allows 16 data bits for each of eight time slots to be driven on the TDAT line, which also permits the processor to execute a maximum of 64

instructions during each slot, assuming that a ’C54x internal clock is used. Beginning with slot zero and with the MSB first, the transmitter drives 16 data bits for each slot, with each bit having a duration of one TCLK cycle, with the exception of the first bit of each slot, which lasts only half of one bit time. Note that data is both clocked onto the TDAT line by the transmitting device and sampled from the TDAT line by receiving devices on the rising edge of TCLK (refer to Appendix A for detailed interface timings).

Figure 1–61. Serial Port Timing in TDM Mode



Simultaneous with data transfer, the transmitting device also drives the TADD line with the transmit address for each slot. This information, unlike that on TDAT, is only one byte long and is transmitted with the LSB first for the first half of the slot. During the second half of the slot (that is, the last eight TCLK periods) the TADD line is driven low. The TDM receive logic samples the TADD line only for the first eight TCLK periods, ignoring it during the second half of the slot. Therefore, the transmitting device (if not a ’C54x) could drive TADD high or low during that time period. Note that, like TDAT, the first TADD bit transmitted lasts for only one half of one TCLK cycle.

If no device on the TDM bus is configured to transmit in a slot (that is, none of the devices has a 1 for the corresponding slot in their TCSR register), that slot is considered empty. In an empty slot, both TADD and TDAT are high impedance. This condition has the potential for spurious receives, however, because TDAT and TADD are always sampled, and a device performs a valid TDM reception if its receive address matches the address on the TADD line. To avoid spurious reads, a pull-down resistor ***must*** be tied to the TADD line. This causes the TADD line to read low on empty slots. Otherwise, any noise on the TADD line that happens to match a particular receive address would result in a spurious read. If power dissipation is a concern and the resistor is not desired, then an arbitrary processor with transmit address equal to 0h can drive empty slots by writing to TDXR in those slots. Slot manipulation is explained later in this section. The 1-kΩ resistor is not required on the TDAT line.

An empty TDM slot can result in the following cases: the first obvious case, as mentioned above, occurs when no device has its TCSR configured to transmit in that slot. A second more subtle case occurs when TDXR has not been

loaded before a transmit slot in a particular frame. This may also happen when the TCSR contents are changed, since the actual TCSR contents are not updated until the next TFRM pulse occurs. Therefore, any subsequent change takes effect only in the next frame. The same is true for the receive address (the lower half of TRTA). The transmit address (upper half of TRTA), however, and TDXR, clearly, may be changed within the current frame for a particular slot, assuming that the slot has not yet been reached when the instruction to load the TRTA or TDXR is executed. Note that it is not necessary to load the transmit address each time TDXR is loaded; when a TDXR load occurs and a transmission begins, the current transmit address in TRTA is transmitted on TADD.

The current slot number may be obtained by reading the X2–X0 bits in TRAD. This affords the flexibility of reconfiguring the TDM port on a slot-by-slot basis, and even slot sharing if desired. The key to utilizing this capability is to understand the timing relationship between the instructions being executed and the frame/slots of the TDM port. If the TDM port is to be manipulated on a slot-by-slot basis, changes must be made to appropriate registers quickly enough for the desired effect to take place at the desired time. It is also important to take into account that the TCSR and the receive address (lower half of TRTA) take effect only at the start of a new frame, while the transmit address (upper half of TRTA) and TDXR (transmit data) can take effect at the start of a new slot, as mentioned previously.

If you change the transmit address on the fly, be careful not to corrupt the receive address, since both addresses are located in the TRTA register. This location is to maintain the convention of allowing the transmitting device to specify the receiving devices.

1.3.4 TDM Serial Port Interface Error Conditions

Because of the nature of the TDM architecture, with the ability for one processor to transmit in multiple slots, the concepts of overflow and underrun become indeterminate. Therefore, the overrun and underflow flags are not active in TDM mode.

In the receiver, if TRCV has not been read and a valid receive operation is initiated (because of the value on TRTA and the device's receive address), the present value of TRCV is overwritten; the receiver is ***not*** halted. On the other hand, if TDXR has not been updated before a transmission, the TADD or TDAT lines are not driven, and these pins remain in the high-impedance state. This mode of operation prevents spurious transmits.

If a TFRM pulse occurs at an improper time during a frame, the TDM port is not able to continue functioning properly, since slot and bit numbers become

ambiguous when this occurs. Therefore, only one TFRM should occur every 128 TCLK cycles. Unlike the serial port, the TDM port cannot be reinitialized with a frame sync pulse during transmission.

1.3.5 Example of TDM Serial Port Interface Operation

The following is an example of TDM port operation, showing the contents of some of the key device registers involved, and explaining the effect of this configuration on port operation. In this example, eight devices are connected to the TDM port as shown in Figure 1–59.

Table 1–8 shows the TADD value during each of the eight channels given the transmitter and receiver designations shown. This example shows the configuration for eight devices to communicate with each other. In this example, device zero broadcasts to all other device addresses during slot zero. In subsequent frames, devices one through seven each communicate to one other processor.

Table 1–8. Interprocessor Communications Scenario

Channel	TADD Data	Transmitter Device	Receiver Device(s)
0	0FEh	0	1–7
1	40h	7	6
2	20h	6	5
3	10h	5	4
4	08h	4	3
5	04h	3	2
6	02h	2	1
7	01h	1	0

Table 1–9 shows the TDM port register contents of each device that results in the scenario given in Table 1–8. Device zero provides the clock and frame control signals for all channels and devices. The TCSR and TRTA register contents specify which device is to transmit on a given channel and which devices are to receive.

Table 1–9. TDM Register Contents

Device	TSPC	TRTA	TCSR
0	xxF9h	0FE01h	xx01h
1	xxC9h	0102h	xx80h
2	xxC9h	0204h	xx40h
3	xxC9h	0408h	xx20h
4	xxC9h	0810h	xx10h
5	xxC9h	1020h	xx08h
6	xxC9h	2040h	xx04h
7	xxC9h	4080h	xx02h

In this example, the transmit address of a given device (the upper byte of TRTA) matches the receive address (the lower byte of TRTA) of the receiving device. Note, however, that it is not necessary for the transmit and receive addresses to match exactly; the matching operation implemented in the receiver is a bitwise AND. Thus, it is only necessary that one bit in the field matches for a receive to occur. The advantage of this scheme is that a transmitting device can select the device or devices to receive its transmitted data by simply changing its transmit address (as long as each devices' receive address is unique, the receive address of the receiving device does not need to be changed). In the example, device zero can transmit to any combination of the other devices by merely writing to the upper byte of TRTA. Therefore, if a transmitting device changed its TRTA to 08001h on the fly, it would transmit only to device seven.

A device may also transmit to itself, because both the transmit and receive operations are executed on the rising edge of TCLK (see Appendix A for detailed TDM interface timings). To enable this type of loopback, it is necessary to use the standard TDM port interface connections as shown in Figure 1–59. Then, if device zero has a TRTA of 00101h, it would transmit only to itself.

Another example of TDM port operation is shown in the code sequence below, in which a one-way transmit of a sequence of values from device 0 to device 1 is shown. The values are stored in each device in a block from 4000h to 6000h in data memory. Device zero transmits in slot zero and has a transmit address of 01h. It waits in a $\overline{\text{BIO}}$ loop for a ready to receive signal (XF) from device one, and initializes the transfer data with a value of zero. Only its transmit interrupt is enabled, and its transmit ISR writes the value it will send into its own memory.

Example 1–9. One-Way Transmit Operation from Device 0 to Device 1—Transmit Side

```
* Device 0 – Transmit side
                .
                .
                .
        STM     1h, TCSR          ;Setup TCSR to xmt on slot 0
        STM     100h, TRTA        ;Setup transmit address
                                  ;Setup TSPC as TCLK, TFRM
                                  ;source
        STM     0039h, TSPC       ;Set TXM=MCM=FSM=TDM=1,
                                  ;DLB=FO=0.
                                  ;And put TDM into reset
                                  ;(XRST=RRST=0)
        STM     00F9h, TSPC       ;Take TDM out of reset
                                  ;Setup interrupts
        STM     0ffffh, IFR       ;Clear IFR
        STM     080h, IMR         ;Turn on TXNT
        RSBX    INTM              ;Enable interrupts
TILOOP  BC      TSENDZ, BIO       ;Wait for ready–to–
        B       TILOOP            ;receive from other device
TSENDZ  LD      #0, A             ;First transmission/write
                                  ;Value is 0.
        STM     4000h, AR7        ;Setup where to write
        STL     A, *AR7           ;Write first value
        STLM    A, TDXR           ;Transmit first value

SELF2   B       SELF2             ;Wait for interrupts

TXMT_ISR
        LDM     AR7, A            ;Check if past 0x6000
        SUB     #6000h, A         ;i.e. end of block
        BC      END_TDMP, AGEQ    ;Go to tight loop if so.
                                  ;Add one and transmit
        LD      *AR7+, A          ;Load value
        ADD     #1                ;Add one
        STL     A, *AR7           ;Write value
        STLM    A,  TDXR          ;Transmit value
        RETE
END_TDMP   B    END_TDMP          ;Sit in tight loop after
                                  ;block is complete.
                     .
                     .
                     .
```

The code in device one follows. It has a receive address of 01h and sends a ready-to-receive signal ($\overline{XF}$) to device zero. Only its receive interrupt is enabled, and its receive ISR reads from the TDRR, writes to the block, and then checks to see if it has reached the end of the block.

Example 1–10. One-Way Transmit Operation from Device 0 to Device 1—Receive Side

```
                        .
                        .
                        .
*Device 1 – receive side
        STM   0h, TCSR          ; Setup TCSR to xmt on no
                                ; slots
        STM   001h, TRTA        ; Setup receive address
                                ; Set TDM as TCLK, TFRM
                                ; receive
        STM   0009h, TSPC       ; Set TXM=MCM=DLB=FO=0,
                                ; FSM=TDM=1.
                                ; And put TDM into reset
                                ; (XRST=RRST=0)
        STM   00C9h, TSPC       ; Take TDM out of reset
        STM   0ffffh, IFR       ; Clear IFR
        STM   040h, IMR         ; Mask on TRNT
        RSBX  INTM              ; Enable interrupts
        STM   4000h, AR7        ; Setup where to write
                                ; received data
        RSBX  XF                ; Signal ready to receive
SELF2   B     SELF2             ; Wait for interrupts
TRCV_ISR
        LDM   TRCV, A           ; Load received value
        STL   A, *AR7+          ; Write to memory block
        LDM   AR7, A            ; Check if past 0x6000
        SUB   #6000h, A         ; i.e. end of block
        BC    END_TDMP, AGEQ    ; Go to tight loop if so
        RETE
END_TDMP   B  END_TDMP          ; Sit in tight loop after
                                ; block is complete.
```

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.