TMS320C6727, TMS320C6726,

TMS320C6722

Digital Signal Processors

Silicon Errata

Silicon Revisions 1.0, 1.1

SPRZ232B
July 2005
Revised November 2005

REVISION HISTORY

This revision history highlights the technical changes made to SPRZ232A to generate SPRZ232B.

Scope: Silicon revision 1.1 silicon is now TMS (fully qualified production device).

Added Advisories 1.1.1, 1.1.2, 1.1.3, 1.1.4, and 1.1.5 (all of which are applicable to both Revision 1.1 silicon ***and*** Revision 1.0 silicon).

PAGE(S) NO.	ADDITIONS/CHANGES/DELETIONS
5	Table 1, Die PG Codes: − silicon revision 1.1 silicon is now TMS (fully qualified production device)
6	Added Advisory 1.1.1, SPI: Slave Mode Four-Pin Chip-Select and Five-Pin Anomalies [applicable to both Revision 1.1 silicon ***and*** Revision 1.0 silicon]
19	Added Advisory 1.1.2, SPI Slave Mode Only: Final SPIx_SOMI Bit has Short Hold Time [applicable to both Revision 1.1 silicon ***and*** Revision 1.0 silicon]
22	Added Advisory 1.1.3, SPI Master Mode: Do not Use WDELAY [applicable to both Revision 1.1 silicon ***and*** Revision 1.0 silicon]
22	Added Advisory 1.1.4, SPI Master Mode: Extra Step Required to Use CSHOLD [applicable to both Revision 1.1 silicon ***and*** Revision 1.0 silicon]
25	Added Advisory 1.1.5, SPI Master Mode: Do Not Use T2EDELAY and T2CDELAY [applicable to both Revision 1.1 silicon ***and*** Revision 1.0 silicon]

Contents

1 Introduction

This document describes the known exceptions to the functional specifications for the TMS320C6727, TMS320C6726, and TMS320C6722 digital signal processors.† [See the *TMS320C6727, TMS320C6726, TMS320C6722 Floating-Point Digital Signal Processors* data sheet (literature number SPRS268).]

The advisory numbers in this document are not sequential. Some advisory numbers have been moved to the next revision and others have been removed and documented in the user's guide. When items are moved or deleted, the remaining numbers remain the same and are not resequenced.

This document also contains "Usage Notes". Usage Notes highlight and describe particular situations where the device's behavior may not match presumed or documented behavior. This may include behaviors that affect device performance or functional correctness. These notes will be incorporated into future documentation updates for the device (such as the device-specific data sheet), and the behaviors they describe will not be altered in future silicon revisions.

1.1 Device and Development-Support Tool Nomenclature

To designate the stages in the product development cycle, TI assigns prefixes to the part numbers of all DSP devices and support tools. Each DSP commercial family member has one of three prefixes: TMX, TMP, or TMS (e.g., **TMS**320C6727). Texas Instruments recommends two of three possible prefix designators for its support tools: TMDX and TMDS. These prefixes represent evolutionary stages of product development from engineering prototypes (TMX/TMDX) through fully qualified production devices/tools (TMS/TMDS).

Device development evolutionary flow:

TMX Experimental device that is not necessarily representative of the final device's electrical specifications

TMP Final silicon die that conforms to the device's electrical specifications but has not completed quality and reliability verification

TMS Fully qualified production device

Support tool development evolutionary flow:

TMDX Development-support product that has not yet completed Texas Instruments internal qualification testing.

TMDS Fully qualified development-support product

TMX and TMP devices and TMDX development-support tools are shipped against the following disclaimer:

"Developmental product is intended for internal evaluation purposes."

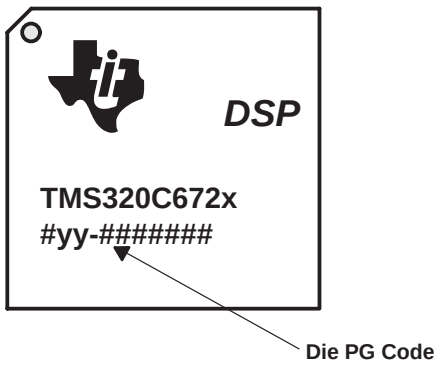
TMS devices and TMDS development-support tools have been characterized fully, and the quality and reliability of the device have been demonstrated fully. TI's standard warranty applies.

Predictions show that prototype devices (TMX or TMP) have a greater failure rate than the standard production devices. Texas Instruments recommends that these devices not be used in any production system because their expected end-use failure rate still is undefined. Only qualified production devices are to be used.

† Throughout the remainder of the document, TMS320C6727 (or C6727), TMS320C6726 (or C6726), and/or TMS320C6722 (or C6722) will be referred to as TMS320C672x (or C672x).

1.2 Revision Identification

The device revision can be determined by the Die PG code marked on the top of the package. The location of the Die PG code for the C672x packages is shown in Figure 1. Figure 1 also shows an example of the types of C672x package symbolization.



NOTE: Qualified devices are marked with the letters “TMS” at the beginning of the device name, while nonqualified devices are marked with the letters “TMX” or “TMP” at the beginning of the device name.

Figure 1. Example, Die PG Codes for TMS320C672x Device Packages

Silicon revision is identified by a code on the chip. The code is of the format #yy-#######. For example, if yy is “11”, then the silicon is revision 1.1, etc. Table 1 lists the silicon revisions associated with each die PG code for the C6727, C6726, and C6722 devices.

Table 1. Die PG Codes

Die PG Code (yy)	Silicon Revision	Comments
11	1.1	TMS320C6727GDH, TMS320C6727ZDH, TMS320C6726RFP, and TMS320C6722RFP
10	1.0	Initial silicon revision: TMX320C6727GDH, TMX320C6727ZDH, TMX320C6726RFP, and TMX320C6722RFP

2 Silicon Revision 1.1 Known Design Exceptions to Functional Specifications and Usage Notes

2.1 Usage Notes for Silicon Revision 1.1

Usage Notes highlight and describe particular situations where the device's behavior may not match presumed or documented behavior. This may include behaviors that affect device performance or functional correctness. These notes will be incorporated into future documentation updates for the device (such as the device-specific data sheet), and the behaviors they describe will not be altered in future silicon revisions.

There are currently no known usage notes for the C672x silicon revision 1.1 devices.

2.2 Silicon Revision 1.1 Known Design Exceptions to Functional Specifications

Advisory 1.1.1	*SPI: Slave Mode Four-Pin Chip-Select and Five-Pin Anomalies*

Revision(s) Affected: 1.1, 1.0

Details:

This advisory applies to both the SPI0 and SPI1 modules on the C672x DSP. The C672x SPI modules sample the $\overline{\text{SPIx\_SCS}}$ pin incorrectly, which can result in:

- the slave erroneously transferring data while it is deselected
- issues with the sharing of the $\overline{\text{SPIx\_ENA}}$ pin with other slave devices on the same SPI bus.

In four-pin with chip-select slave mode and five-pin slave mode, the SPI module provides a chip-select input ($\overline{\text{SPIx\_SCS}}$). The chip-select input facilitates the sharing of the same SPI bus with other slave devices.

The master asserts the chip select on the device with which it intends to communicate and de-asserts the chip select on other slave devices before it begins the SPI transfer. The deselected slave device(s) should then:

- ignore activity on the SPI bus while $\overline{\text{SPIx\_SCS}}$ remains de-asserted
- place their SPIx_SOMI pin in a high-impedance state to avoid conflicting with transmit data from the selected slave
- de-assert their $\overline{\text{SPIx\_ENA}}$ output by either 3-stating the pin or driving it high. (SPIINT0.ENABLEHIGHZ controls the choice of 3-stating the pin or driving it high)

The C672x SPI modules do not correctly implement the $\overline{\text{SPIx\_SCS}}$ function. This can lead to the C672x SPI receiving data erroneously when it is not selected. In five-pin mode, it can also lead to a conflict on the $\overline{\text{SPIx\_ENA}}$ pin.

The SPI module samples the $\overline{\text{SPIx\_SCS}}$ pin only at the beginning of a transfer. Once $\overline{\text{SPIx\_SCS}}$ is asserted, the SPI enters a transfer state and remains in this state until the transfer completes, regardless of the state on the $\overline{\text{SPIx\_SCS}}$ pin during the transfer.†

† In four-pin with chip-select mode. In five-pin mode, when the slave services the SPI transmit buffer is also a factor (see Figure 3).

SPI: Slave Mode Four-Pin Chip-Select and Five-Pin Anomalies (Continued)

If the master de-asserts $\overline{\text{SPIx\_SCS}}$ and selects a different slave device, the C672x SPI still completes the transfer and, in error, receives the data intended for the other slave device. Also, transmit data is consumed in error and must be re-transmitted. However, it appears as if no data is transmitted. This is due to $\overline{\text{SPIx\_SCS}}$ de-assertion asynchronously placing the SPIx_SOMI in a high-impedance state. This also means that the deselected slave SPI does not conflict with transmit data from the selected device.

In five-pin mode, the $\overline{\text{SPIx\_SCS}}$ pin is also supposed to force the slave to de-assert its $\overline{\text{SPIx\_ENA}}$ output pin to allow multiple slaves to share the same handshake line. Only the selected slave device should assert $\overline{\text{SPIx\_ENA}}$.

Instead of the correct behavior, the C672x SPI module drives the $\overline{\text{SPIx\_ENA}}$ with its actual ready status whenever it is in the transfer state. This means while the SPI slave is in the transfer state erroneously, it drives its actual ready status on the $\overline{\text{SPIx\_ENA}}$ line instead of indicating “not ready”.

The most common reason for the SPI slave to enter the transfer state erroneously is due to the timing of the master de-asserting $\overline{\text{SPIx\_SCS}}$ at the end of a valid transfer (or series of transfers).

The C672x SPI module begins sampling the $\overline{\text{SPIx\_SCS}}$ line to begin a new transfer almost immediately after the final receive edge of the SPIx_CLK. In most cases, it is not feasible for the master to de-assert $\overline{\text{SPIx\_SCS}}$ in time to avoid an additional erroneous transfer following a series of valid transfers.

The SPI module does not meet the timing requirements that are outlined in the C672x data sheet SPI parameter 26, $t_{d(SPC\_SCSH)S}$, which specifies a **minimum** delay time from the final SPIx_CLK edge until the de-assertion of $\overline{\text{SPIx\_SCS}}$. Instead, in order to avoid a subsequent erroneous transfer, the timing requirements outlined in Figure 2 and Figure 3 must be met. These place a requirement on the **maximum** delay time for $\overline{\text{SPIx\_SCS}}$ de-assertion.

SPI: Slave Mode Four-Pin Chip-Select and Five-Pin Anomalies (Continued)

Figure 2 illustrates the timing requirement on the de-assertion of the $\overline{\text{SPIx\_SCS}}$ pin in the four-pin with chip-select mode. If the master does not meet this requirement, the slave will enter the transfer state again and receive the next word transferred on the SPI bus as well as consume any data already written to the transmit buffer.

Note that in most cases, the device system clock (P) is between 6 ns to 8 ns and the delay is negative. This means the master must actually de-assert the $\overline{\text{SPIx\_SCS}}$ pin before the final receive clock edge. But the slave also 3-states its SPIx_SOMI output when $\overline{\text{SPIx\_SCS}}$ is de-asserted, so the master may not be able to receive the final data bit correctly if it de-asserts $\overline{\text{SPIx\_SCS}}$ early. (Timing requirements of SPI master devices vary; so, consult the data sheet of the particular master.)

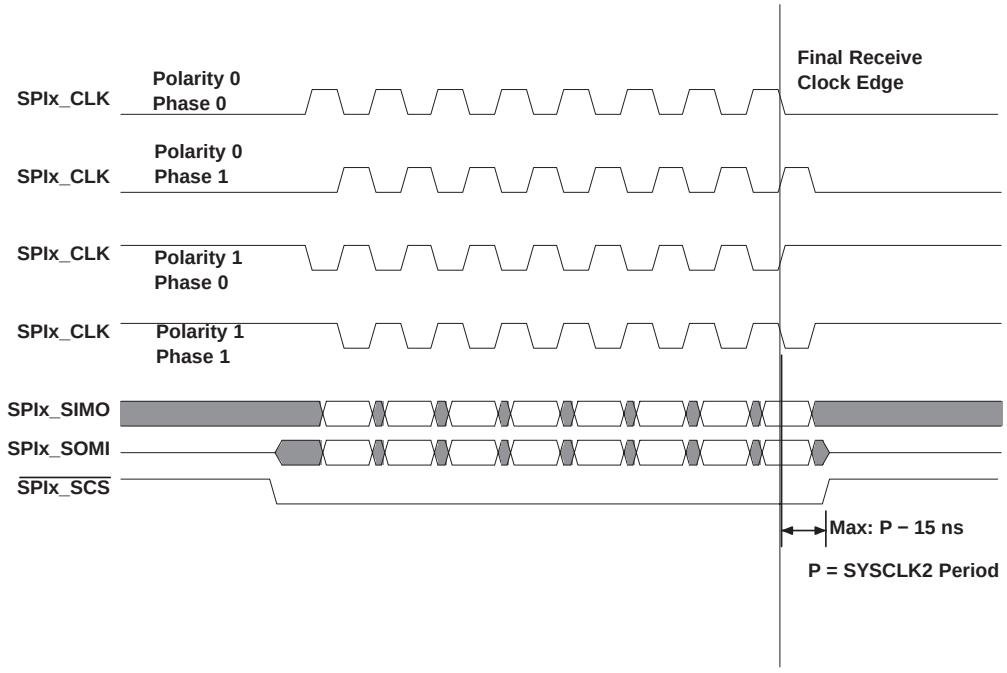


Figure 2. Timing Requirement of $\overline{\text{SPIx\_SCS}}$ De-assertion in Four-Pin With Chip-Select Mode

SPI: Slave Mode Four-Pin Chip-Select and Five-Pin Anomalies (Continued)

Figure 3 illustrates the timing requirement on the de-assertion of $\overline{\text{SPIx\_SCS}}$ in five-pin mode. The timing requirements of this mode are looser than those in the four-pin with chip-select mode.

In five-pin mode, the slave delays entering the transfer state until $\overline{\text{SPIx\_SCS}}$ is asserted and the transmit buffer (SPIDAT0 or SPIDAT1) has been written with data for the next transfer. This provides additional time for the master to de-assert $\overline{\text{SPIx\_SCS}}$. As long as the master de-asserts $\overline{\text{SPIx\_SCS}}$ 2P + 15 ns before the slave writes to SPIDAT0 or SPIDAT1, the slave will not enter the transfer state until $\overline{\text{SPIx\_SCS}}$ is asserted again, and the erroneous transfer may be avoided.

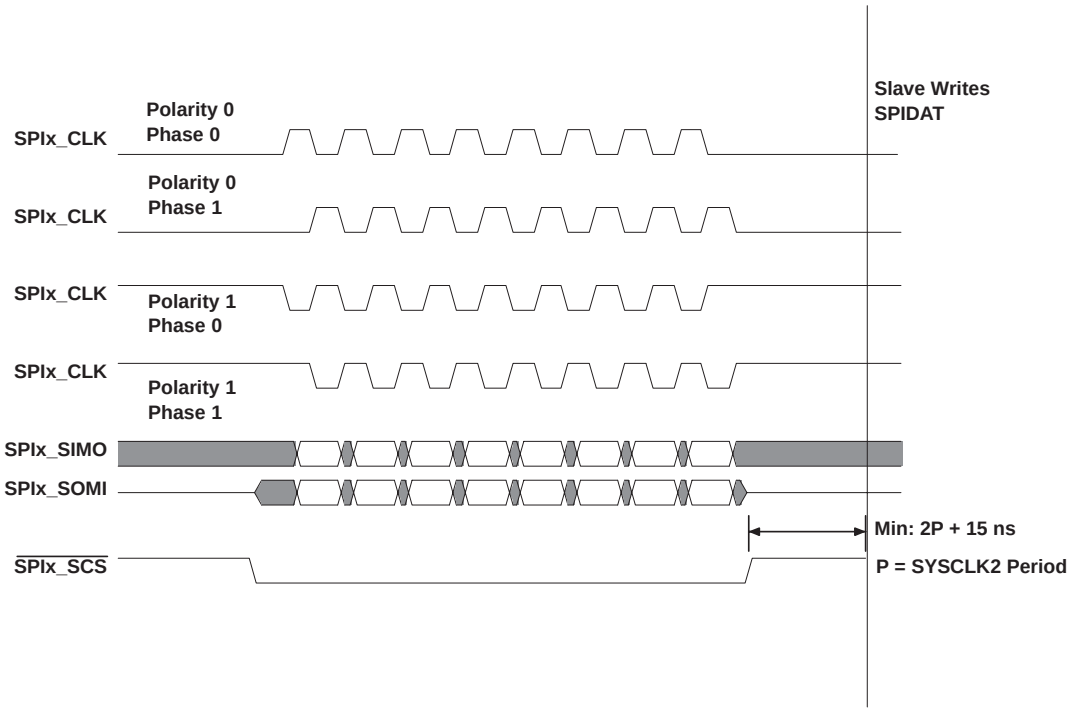


Figure 3. Timing Requirement of $\overline{\text{SPIx\_SCS}}$ De-assertion in Five-Pin Mode

SPI: Slave Mode Four-Pin Chip-Select and Five-Pin Anomalies (Continued)

The second most common reason for an erroneous transfer is a glitch on the $\overline{\text{SPIx\_SCS}}$ pin while the slave SPI is idling between transfers. Since the slave SPI samples $\overline{\text{SPIx\_SCS}}$ using SYSCLK2, any low-going glitch that lasts longer than a SYSCLK2 period will cause the SPI slave to enter the transfer state. Additionally, any glitch that is shorter than a SYSCLK2 period may still cause the SPI slave to enter the transfer state; but in this case, it depends upon whether or not the slave captures the glitch.

There are several ways that the manner in which the slave samples $\overline{\text{SPIx\_SCS}}$ can create a system-level problem. These are:

1. A glitch on $\overline{\text{SPIx\_SCS}}$ may cause the slave SPI to receive data erroneously.
2. The timing of $\overline{\text{SPIx\_SCS}}$ de-assertion may cause the slave to receive data erroneously.
3. In both of the above cases, if the slave does not receive enough SPIx_CLK clocks while deselected to **complete** the erroneous transfer before being selected again, it will lose synchronization with the master.
4. If the slave is deselected and is in the transfer state in error, but there are no clocks on SPIx_CLK, then no improper transfer will occur.
5. In five-pin mode, the slave drives the $\overline{\text{SPIx\_ENA}}$ pin incorrectly while it is in the transfer state and $\overline{\text{SPIx\_SCS}}$ is de-asserted. This can be a problem if multiple slave devices share the $\overline{\text{SPIx\_ENA}}$ pin.

SPI: Slave Mode Four-Pin Chip-Select and Five-Pin Anomalies (Continued)

Figure 4 illustrates the case where the slave SPI captures a glitch on the chip-select line and enters the transfer state erroneously. If transfers to another slave device follow this event, then the C672x slave will erroneously receive the first character transmitted to the other selected slave device. Since $\overline{\text{SPIx\_SCS}}$ is de-asserted, the C672x SPI holds its SPIx_SOMI pin in a high-impedance state and does not interfere with the data transmitted back to the SPI bus master by the selected slave device. After the erroneous transfer completes, the slave remains idle until the next assertion of $\overline{\text{SPIx\_SCS}}$. (See Figure 6 for the case where there are not enough clocks to complete the erroneous transfer.)

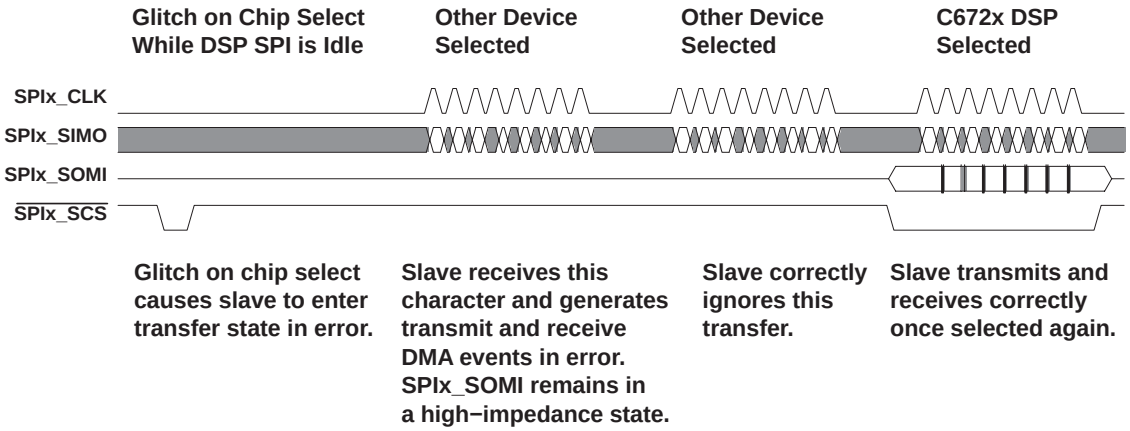


Figure 4. Glitch on Chip Select Causing Erroneous Transfer

Figure 5 illustrates almost the same case as Figure 4, except the slave device enters the transfer state due to the master failing to meet the timing requirements for de-assertion of the slave $\overline{\text{SPIx\_SCS}}$ input following a valid transfer as illustrated in Figure 2 and Figure 3.

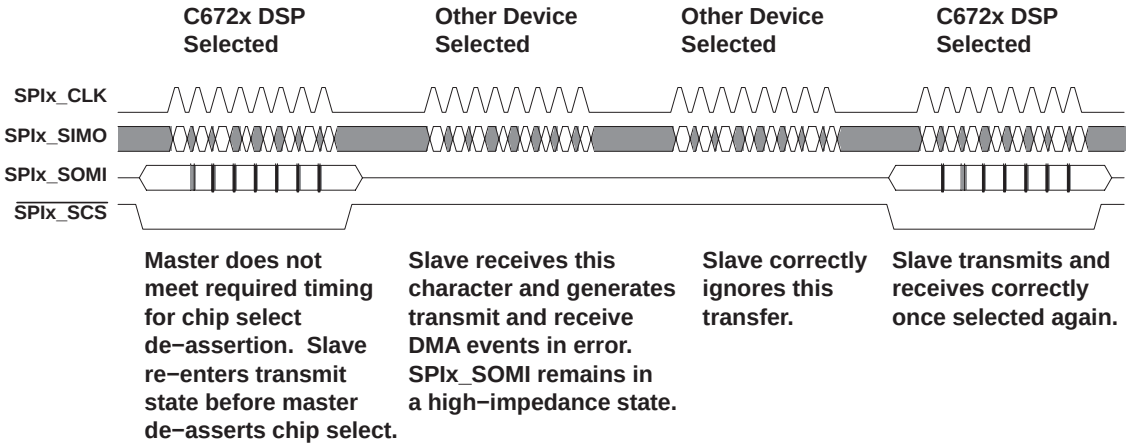


Figure 5. Erroneous Transfer Due to Master De-asserting Chip Select Late

SPI: Slave Mode Four-Pin Chip-Select and Five-Pin Anomalies (Continued)

Figure 6 illustrates the case where the slave enters the transfer state in error due to the master de-asserting $\overline{\text{SPI\_SCS}}$ too late to meet the timing requirements. (Note: the same issue applies if the slave device enters the transfer state due to a glitch on $\overline{\text{SPIx\_SCS}}$.) If there are not enough clocks for the completion of the erroneous transfer while the slave is deselected, the slave will remain in the transfer state. When selected, the slave will first complete the erroneous transfer using the first bits from the valid transfer. The slave then begins another transfer in the middle of the valid transfer, and synchronization to the master is lost. Data received while synchronization is lost will be invalid.

The slave will regain synchronization only after $\overline{\text{SPIx\_SCS}}$ is de-asserted and there is a sufficient number of clocks on to allow the slave to complete an invalid transfer and enter the idle state before the master asserts $\overline{\text{SPIx\_SCS}}$ again.

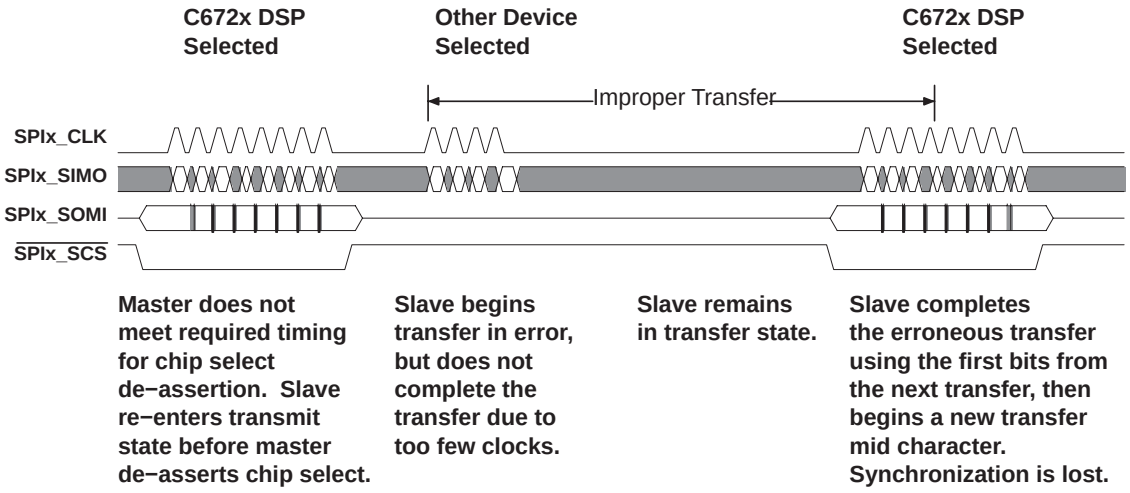


Figure 6. Loss of Synchronization to Character Boundary

SPI: Slave Mode Four-Pin Chip-Select and Five-Pin Anomalies (Continued)

Figure 7 illustrates the case where the slave enters the transfer state in error due to the master de-asserting $\overline{\text{SPI\_SCS}}$ too late to meet the timing requirements. (Note: the same issue applies if the slave device enters the transfer state due to a glitch on $\overline{\text{SPIx\_SCS}}$.)

If there are no clocks on the slave SPIx_CLK line while $\overline{\text{SPIx\_SCS}}$ is de-asserted, then even though the slave is in the transfer state during this time period, no shifting occurs. This means that when the master selects the slave again for another transfer, this transfer will complete correctly. In four-pin mode, the error would go undetected. In five-pin mode, only $\overline{\text{SPIx\_ENA}}$ remains a problem (see Figure 8).

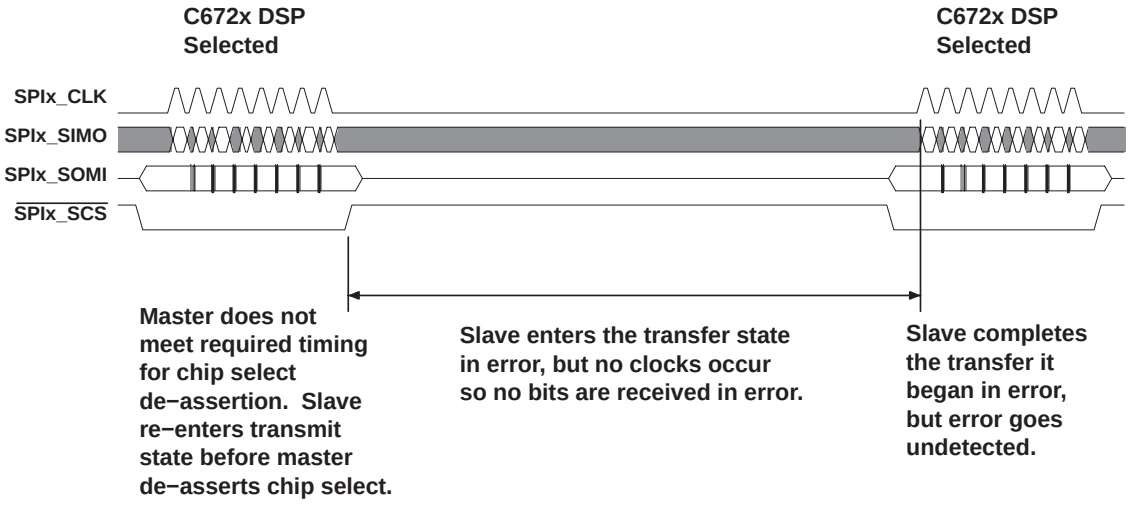


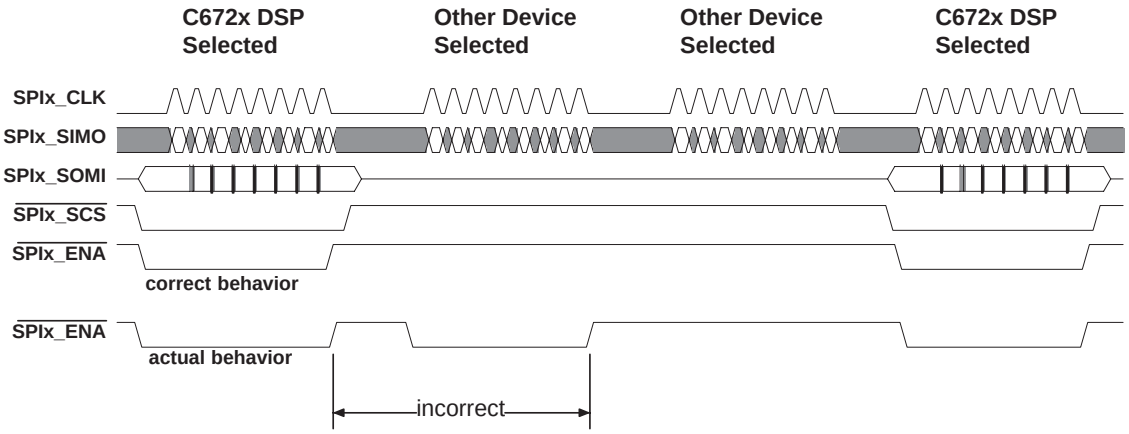
Figure 7. Slave Enters Transfer State in Error, but Error Goes Undetected

SPI: Slave Mode Four-Pin Chip-Select and Five-Pin Anomalies (Continued)

Figure 8 illustrates the relationship between the issues described in Figure 4−Figure 7 and the SPIx_ENA pin in five-pin slave mode. When any of the issues illustrated in Figure 4−Figure 7 occur and the slave enters the transfer state incorrectly, the slave drives SPIx_ENA incorrectly while it remains in the transfer state. Instead of indicating that it is not ready as it should whenever SPIx_SCS is de-asserted, the slave will indicate its actual ready state until it completes the erroneous transfer and enters the idle state again.

This will typically only cause a problem when sharing SPIx_ENA, since the de-selected slave may force the enable line to indicate ready even if the slave actually selected is not ready.

Note that Figure 8 illustrates the case where SPIINT0.ENABLEHIGHZ has been set to drive the SPIx_ENA pin in a push-pull mode. However, the same issue applies when configuring SPIINT0.ENABLEHIGHZ for open-drain mode, except instead of driving SPIx_ENA high as shown in Figure 8, the slave places the pin in a high-impedance state to de-assert SPIx_ENA.



The slave indicates its actual ready state while it is in the transfer state, even if it is in the transfer state in error due to the issues sampling chip select. This behavior is incorrect.

Figure 8. Slave Drives Enable Line Incorrectly During the Time it is in the Transfer State Erroneously

SPI: Slave Mode Four-Pin Chip-Select and Five-Pin Anomalies (Continued)

Workaround: There are several possible workarounds to this issue. These include:

1. In four-pin mode, meet the timing requirements of Figure 2.
2. In five-pin mode, meet the timing requirements of Figure 3.
3. Add an external logic gate to qualify SPIx_CLK (and $\overline{\text{SPIx\_ENA}}$ if required) by $\overline{\text{SPIx\_SCS}}$.
4. Increase the character length of the C672x SPI and make the corresponding change on the master side. Pad the transmit and receive words appropriately to fill out the increased character length. De-assert $\overline{\text{SPIx\_SCS}}$ early.

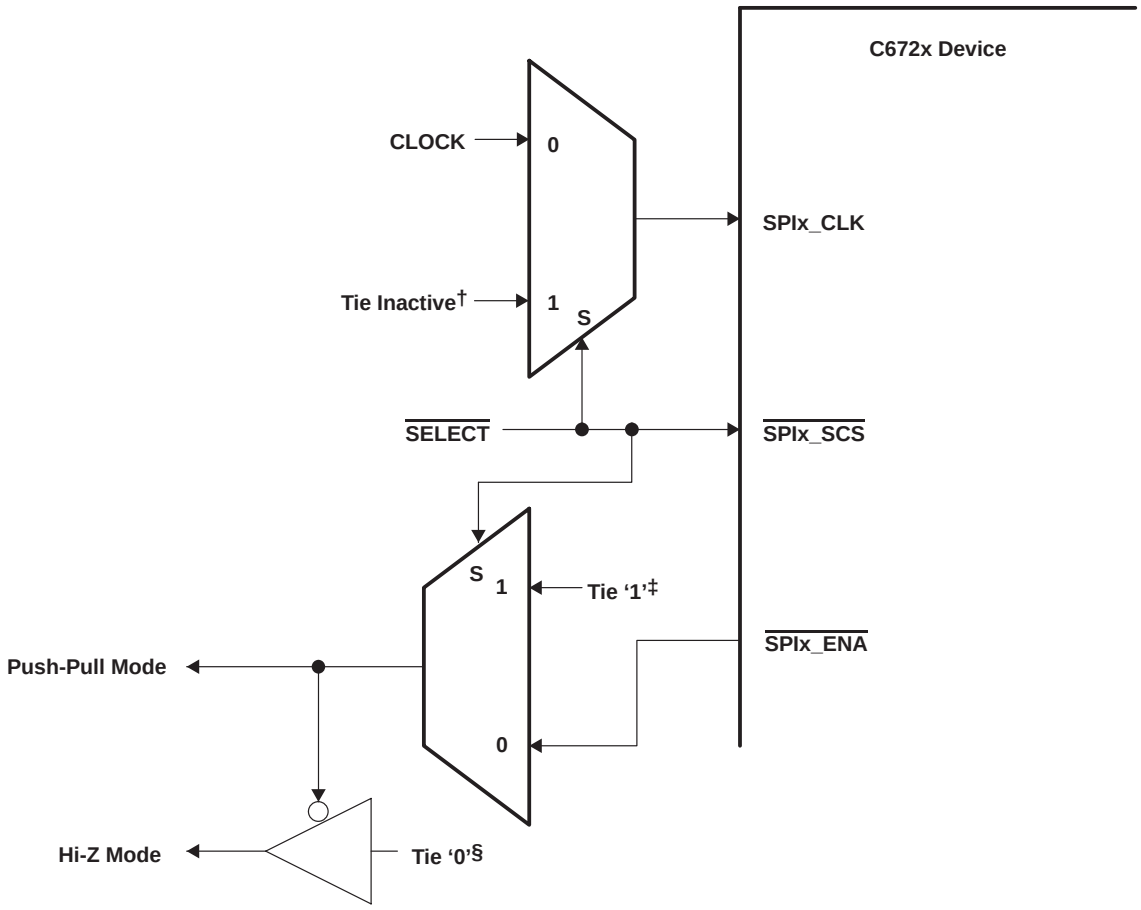
The first two workarounds are possibly the simplest, but may also be the most difficult to implement depending upon the capabilities of the SPI master. The first three workarounds require a separate workaround for Advisory 1.1.2. Only the fourth workaround addresses this advisory as well as Advisory 1.1.2.

Workaround 1 and Workaround 2 are self-explanatory (timing requirements must be met).

The third workaround involves the addition of external logic gates. Figure 9 illustrates this workaround circuit. The external circuit forces the SPIx_CLK line inactive whenever the SPI bus $\overline{\text{SELECT}}$ line for the C672x device is inactive. While this does not prevent the SPI slave from entering the transfer state in error, it does ensure that no clocks will reach SPIx_CLK while $\overline{\text{SPIx\_SCS}}$ is de-asserted. In other words, at the DSP pins this circuit creates the situation illustrated in Figure 7.

Figure 9 also shows the external logic required to correct the behavior of the $\overline{\text{SPIx\_ENA}}$ pin in case five-pin mode is required and it is also necessary to share a single $\overline{\text{SPIx\_ENA}}$ line among multiple slave devices. When using this logic, SPIINT0.ENABLEHIGHZ must be configured to drive $\overline{\text{SPIx\_ENA}}$ out of the DSP actively, even if the final interface to the SPI bus is open-drain.

SPI: Slave Mode Four-Pin Chip-Select and Five-Pin Anomalies (Continued)



† Inactive Level
‡ '1' if Polarity = 1
§ '0' if Polarity = 0

Figure 9. Workaround 3 Using External Logic Gates

The fourth workaround is a modification of the SPI bus character length and takes advantage of the ability to program the slave SPI character length with values up to 31. The SPI character length in slave mode is programmed using the bit field in SPIFMT0.CHARLEN[4:0]. The SPI documentation says that values of 0x11 through 0x1F have an indeterminate result. The SPI will actually interpret these values from 17 to 31 bits correctly; however, the SPI shift registers are limited to 16 bits, so normally the SPI cannot make use of these values.

SPI: Slave Mode Four-Pin Chip-Select and Five-Pin Anomalies (Continued)

Figure 10 illustrates the operation of the SPI module in 3-pin mode with SPIFMT0.CHARLEN[4:0] programmed with a value greater than 0x10. In this case, the value of this field is 0x18 for a character length of 24 bits. This means that the slave SPI requires 24 clocks on SPIx_CLK to complete a single transfer. However, from an internal viewpoint, the slave only writes one 16-bit word to SPIDAT0/SPIDAT1 at the beginning of the transfer and only reads one 16-bit word from SPIBUF at the end of the transfer. As illustrated in Figure 10, the data written to SPIDAT0/SPIDAT1 will appear on the SPIx_SOMI pin during the first 16 SPI clock periods, followed by 8 indeterminate bits. Also, the data received in the SPIBUF register is the last 16 bits that appear on the SPIx_SIMO pin. The first 8 bits are shifted in but eventually discarded by the slave.

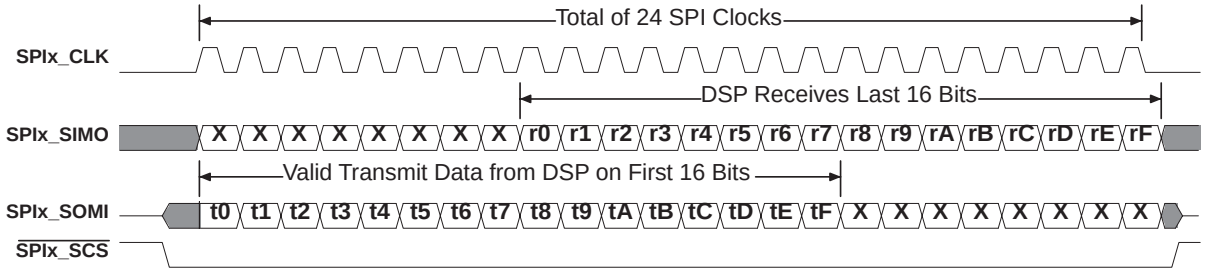


Figure 10. Slave SPI Operation With Character Length of 24 Bits

Figure 11 illustrates the suggested workaround sequence, assuming that the SPI master is a microcontroller using an 8-bit character length.

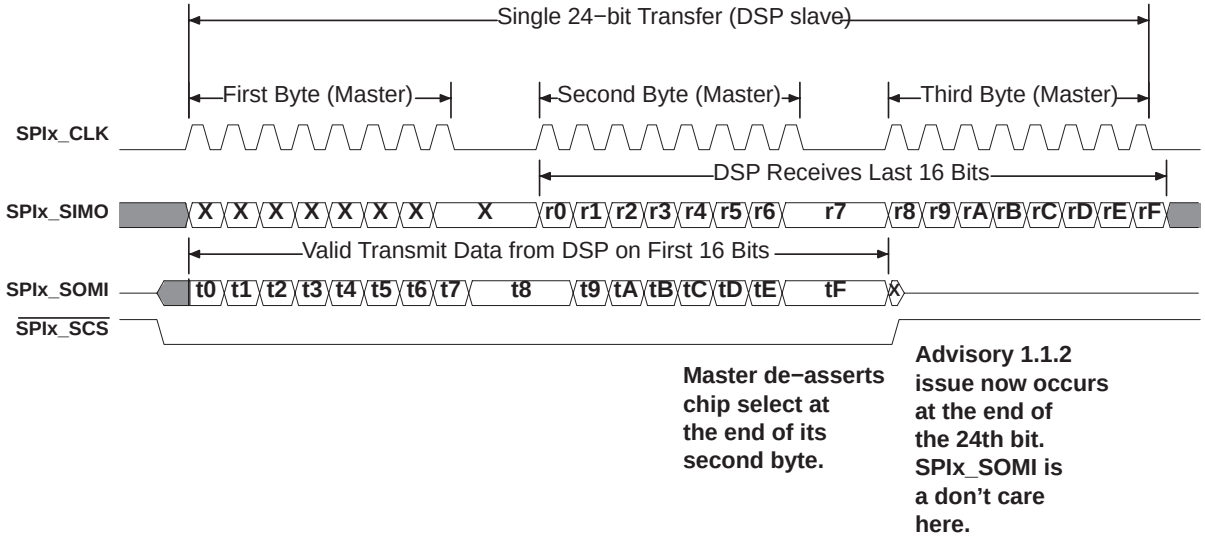


Figure 11. Complete 16-Bit Transfer in 24-Bit Character Using Fourth Workaround

SPI: Slave Mode Four-Pin Chip-Select and Five-Pin Anomalies (Continued)

The microcontroller should view the transfer as 3 bytes. The microcontroller should transmit a dummy byte followed by 2 data bytes (r0−r7 and r8−rF in Figure 11). It should receive two valid data bytes (t0−t7 and t8−tF in Figure 11) and discard the third data byte it receives.

The microcontroller should also de-assert the $\overline{\text{SPIx\_SCS}}$ line between the second and third bytes. This will ensure that it meets the requirement on the de-assertion of $\overline{\text{SPIx\_SCS}}$ (see Figure 2 and Figure 3) because in this case, it will be de-asserting $\overline{\text{SPIx\_SCS}}$ eight SPI clocks before the end of the transfer from the point of view of the DSP slave SPI.

The slave transmits all its valid data bits during the first sixteen clocks and the master discards the final eight bits (their value is a “don’t care”). So, it is fine for the slave to place SPIx_SOMI in a high-impedance state during the final eight bits of the transfer.

This workaround prevents the slave from entering the transfer state in error due to the timing requirements for de-assertion of $\overline{\text{SPIx\_SCS}}$. However, it does not protect against the issue of a glitch on the $\overline{\text{SPIx\_SCS}}$ line initiating a transfer. The system design needs to ensure that a glitch on $\overline{\text{SPIx\_SCS}}$ does not occur.

Assuming this can be done, then this workaround also avoids any problems with $\overline{\text{SPIx\_ENA}}$, since these problems only occur when the slave SPI has entered the transfer state in error.

This approach also resolves the issue discussed in Advisory 1.1.2 because the final bit transmitted by the DSP moves from the sixteenth bit position (‘tF’) to the 24th bit position, which is a “don’t care” in this approach.

NOTE: The fourth workaround needs to be removed once silicon is available to correct the issue described in this advisory. The reason is that once this defect is corrected, the C672x SPI module will not respond to the final eight bit clocks supplied by the master with $\overline{\text{SPIx\_SCS}}$ de-asserted.

Advisory 1.1.2 *SPI Slave Mode Only: Final SPIx_SOMI Bit has Short Hold Time*

Revision(s) Affected: 1.1, 1.0

Details:

According to the *TMS320C6727, TMS320C6726, TMS320C6722 Floating-Point Digital Signal Processors* data sheet (literature number SPRS268), the output hold time on the SPIx_SOMI pin when the SPI is in slave mode is $0.5*t_{c(SPC)S}$ − 10 ns on all bits except for the final bit. The final bit is supposed to be held until the slave writes new data into the SPIDAT0/SPIDAT1 register, typically several hundred nanoseconds when serviced by the dMAX.

However, after the final receive edge on the SPI clock, the SPI automatically copies the receive data buffer back into the transmit shift register.† The new value of the transmit shift register appears on the SPIx_SOMI pin immediately after the copy occurs. This means that the output hold time actually provided by the slave SPI is limited to the time it takes for the copy to complete.

For the C672x device, this time is 1.5*SYSCLK2 periods. With a typical SYSCLK2 period of 8 ns (based on 250-MHz SYSCLK1 and 125-MHz SYSCLK2), the resulting output hold time is only 12 ns.

Figure 12 illustrates the data sheet specification for the output hold time on the final bit (for the SPI in slave mode) versus the hold time on actual silicon.

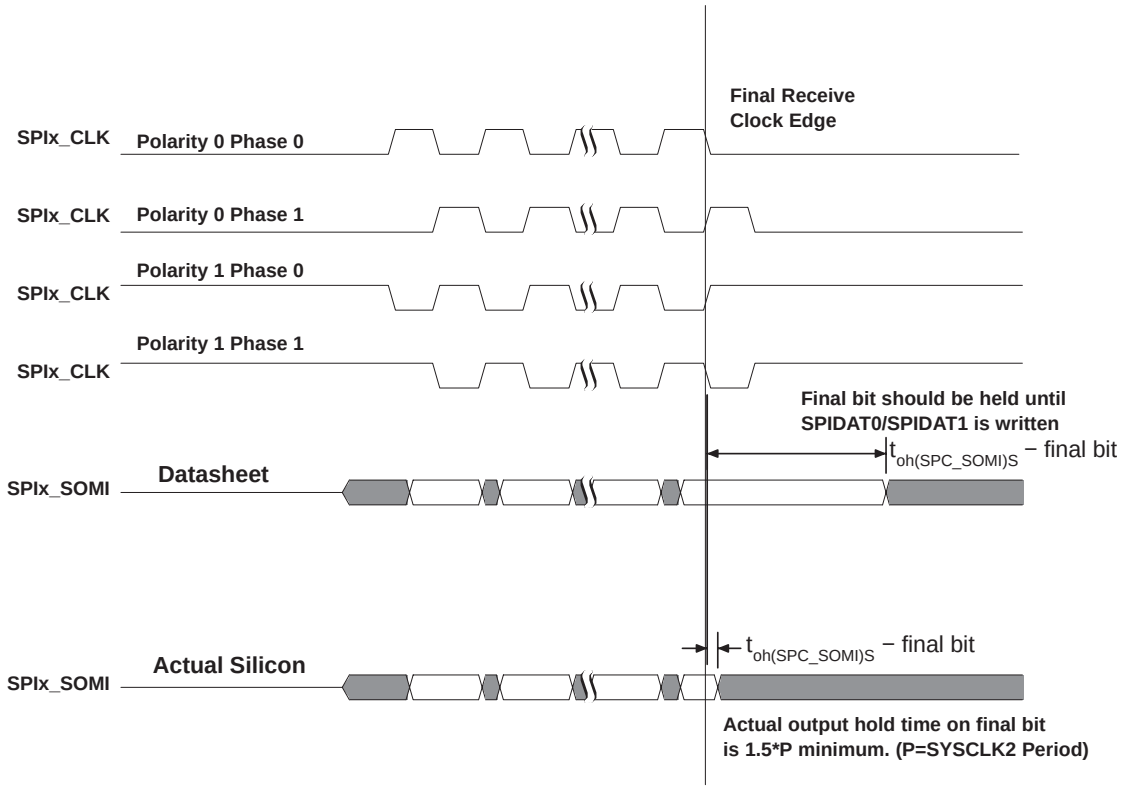


Figure 12. Actual Output Hold Time on Slave SPIx_SOMI Final Bit Versus Data Sheet

† It is implemented this way to maintain the illusion of a single shift register for both transmit and receive, even though there are actually separate shift registers inside the module.

SPI Slave Mode Only: Final SPIx_SOMI Bit has Short Hold Time (Continued)

This may not be sufficient hold time for many master SPI devices. Consult the data sheet of the master device to determine whether or not there is a hold time issue.

Workaround: If the master does require additional hold time, then there are several options.

One workaround is to select a master device that has a hold time requirement on data shifted into its SPI port that is less than the time provided by the DSP. For example, some master SPI devices have an input hold time requirement of ‘0 ns’.

If the master device is already chosen, and it requires more hold time than the DSP provides, then several additional options are available:

- A small gap between the DSP output hold time specification in slave mode and the requirements of the master can be worked around by adding additional delay to the SPIx_CLK and SPIx_SOMI connections between the master and slave devices. This can be accomplished using logic gates or an R−C network. See Figure 13.
- A large output delay can be worked around by implementing a software SPI on the master microcontroller and sampling the SPIx_SOMI pin from the DSP before the final receive clock edge.
- Another option to work around a large gap in specifications is to implement the fourth workaround suggested in Advisory 1.1.1. That workaround also corrects the issue described in this advisory by making the final bit transmitted by the slave SPI a “don’t care”, in which case it does not matter whether or not it is received correctly by the master device.

SPI Slave Mode Only: Final SPIx_SOMI Bit has Short Hold Time (Continued)

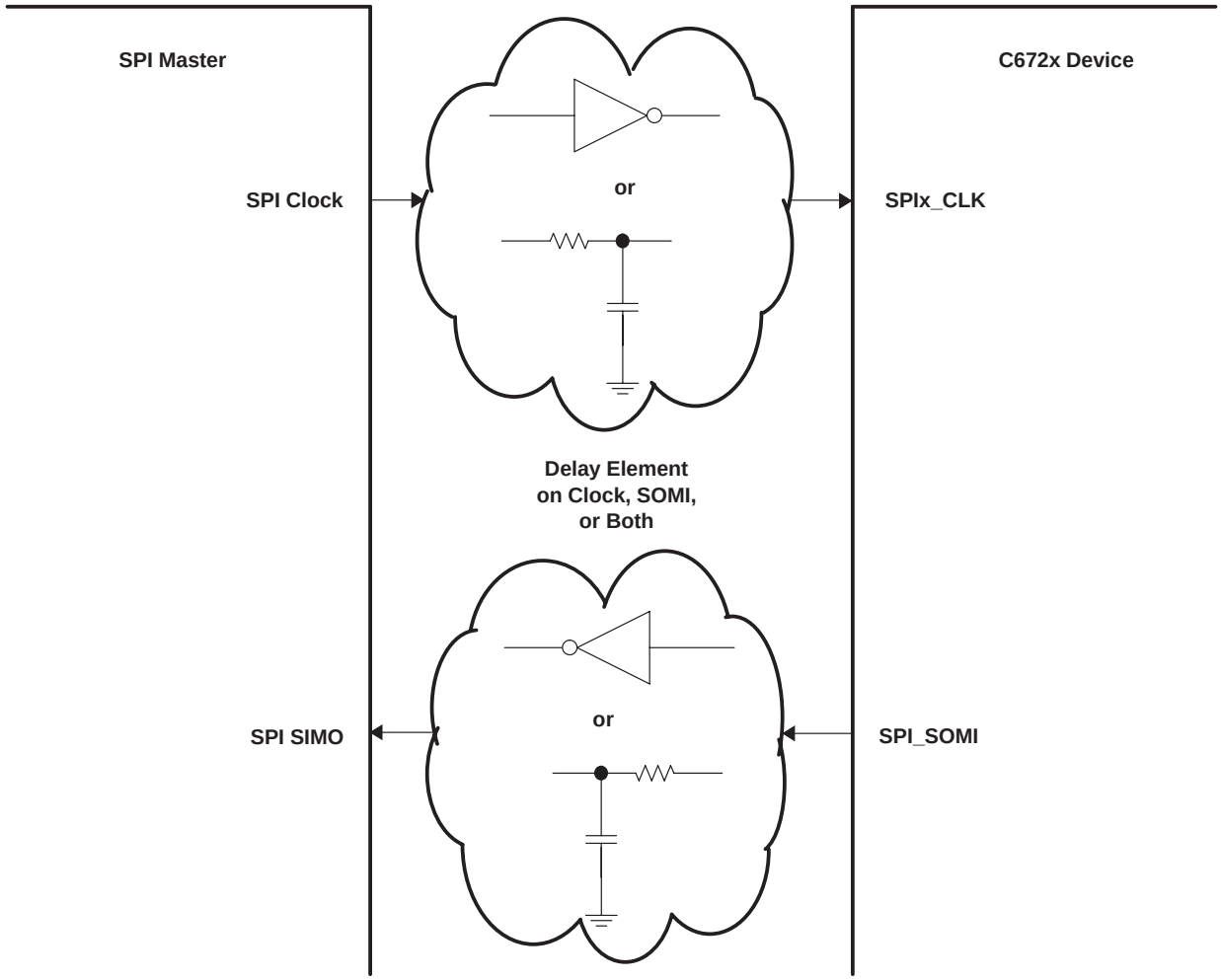


Figure 13. Additional Hold Time Through Delay on Clock and/or SOMI

Advisory 1.1.3 — *SPI Master Mode: Do not Use WDELAY*

Revision(s) Affected: 1.1, 1.0

Details: In master mode, the SPI supports a 6-bit field, SPIFMTx.WDELAY[5:0], which sets a minimum delay value in SYSCLK2 cycles between transfers. Writing to SPIDAT1 with the SPIDAT1.WDEL field activates the delay.

For slave devices that require a delay between transfers, the WDELAY field should simplify servicing the slave with a DMA engine such as dMAX.

While the WDELAY feature does actually ensure a minimum delay between transfers if enabled, it is not useful in the current implementation because the SPI discards any data written to the SPIDAT0/SPIDAT1 register before the delay counter expires.

Workaround: If WDELAY is used, ensure that the DMA or CPU delays writing new data to the SPI transmit buffer until after the WDELAY counter expires

Note that implementing such a workaround by itself will ensure the desired delay, even without the WDELAY counter. Therefore, the recommendation is to implement any required delay in software and **avoid** using the WDELAY counter

Advisory 1.1.4 — *SPI Master Mode: Extra Step Required to Use CSHOLD*

Revision(s) Affected: 1.1, 1.0

Details: The SPI module chip-select hold (CSHOLD) feature allows the device to instruct the SPI to keep the chip-select pin asserted between transfers. This feature applies in master mode and is enabled by writing a ‘1’ to SPIDAT1.CSHOLD (bit 28).

When data is written to the SPIDAT1 register with the CSHOLD bit set to ‘1’, the master is supposed to keep the $\overline{\text{SPIx\_SCS}}$ pin asserted after the transfer completes. When data is written to the SPIDAT1 register with CSHOLD set to ‘0’, the master is supposed to de-assert the $\overline{\text{SPIx\_SCS}}$ pin after the transfer completes.

For example, assume that the device needs to send two 16-bit words (0x1234 and 0x5678) to a SPI slave that requires its chip select to remain asserted between the transfers. This is a common requirement when communicating with SPI memory devices.

According to the SPI specification, the sequence:

- Write 0x10001234 to SPIDAT1 for transmission of 0x1234 (CSHOLD = 1)
- Write 0x00005678 to SPIDAT1 for transmission of 0x5678 (CSHOLD = 0)

should produce the expected result as illustrated in Figure 14.

SPI Master Mode: Extra Step Required to Use CSHOLD (Continued)

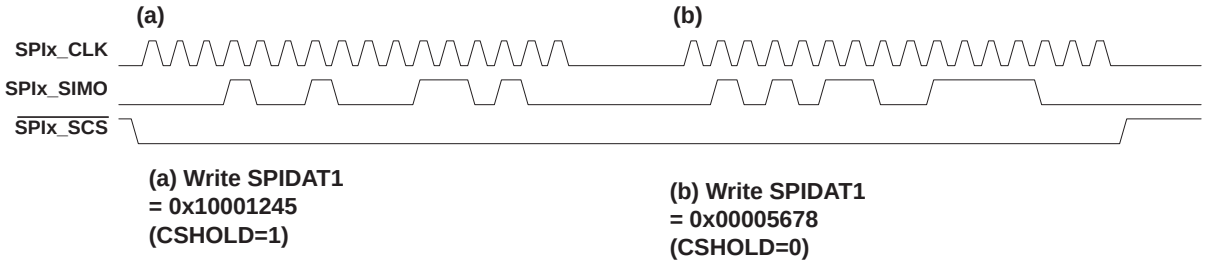


Figure 14. Expected CSHOLD Behavior

Instead, what actually occurs is that SPIx_SCS is momentarily de-asserted at the beginning of the second write, as illustrated in Figure 15.

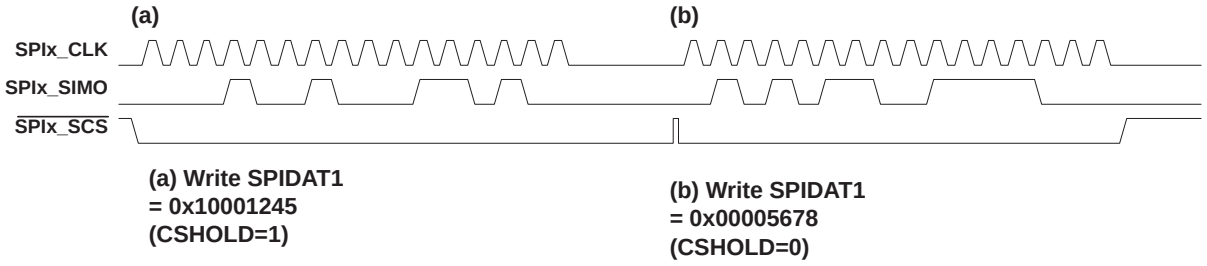


Figure 15. Actual CSHOLD Behavior—32-Bit Writes to SPIDAT1

Both Figure 14 and Figure 15 assume that SPIDAT1 is written using a single 32-bit write instruction. If SPIDAT1 is instead written using an 8-bit or 16-bit instruction to write to the CSHOLD field, followed by a 16-bit write to the transmit shift register field of SPIDAT1, then what actually occurs is illustrated in Figure 16. This is the same case as illustrated in Figure 15 except that the de-assertion of SPIx_SCS lasts for the duration between writing to a '0' to the CSHOLD field and writing new data to the transmit shift register.

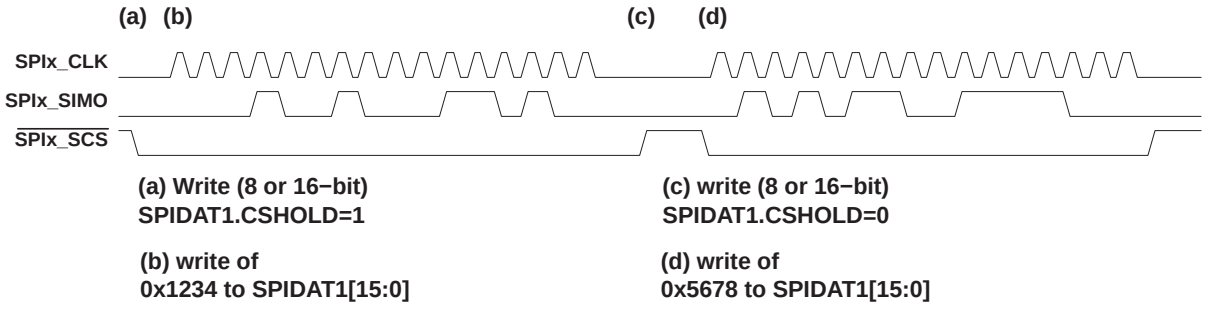


Figure 16. Actual CSHOLD Behavior—32-Bit Writes to SPIDAT1

SPI Master Mode: Extra Step Required to Use CSHOLD (Continued)

Workaround: For each word in the sequence of words during which $\overline{\text{SPIx\_SCS}}$ should be held low, write to the SPIDAT1 register with the CSHOLD bit set to '1'. Follow this by a write to only the CSHOLD field of SPIDAT1, setting CSHOLD = 0 to de-assert $\overline{\text{SPIx\_SCS}}$. See Figure 17 for an illustration.

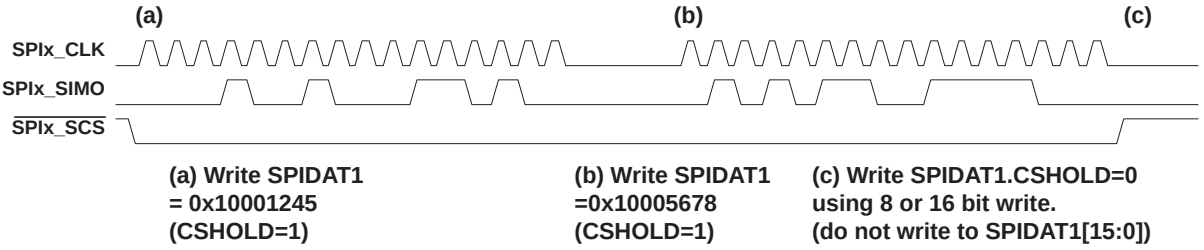


Figure 17. Workaround Assuming 32-Bit Writes to SPIDAT1 Followed by a Write Only to CSHOLD

Alternatively, write to the CSHOLD field only of SPIDAT1 before and after the transfer to toggle the $\overline{\text{SPIx\_SCS}}$ pin. During the transfer, write only to the data field of SPIDAT[15:0] using 16-bit (halfword) write commands. See Figure 18 for an illustration.

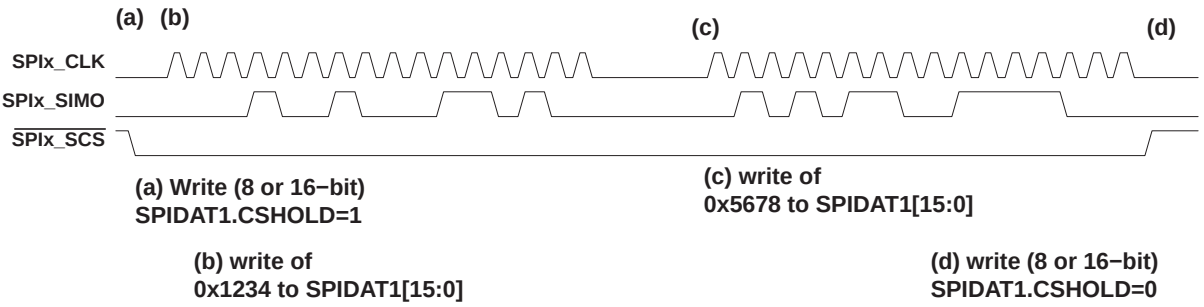


Figure 18. Workaround Assuming Halfword Writes to SPIDAT1

Advisory 1.1.5	*SPI Master Mode: Do Not Use T2EDELAY and T2CDELAY*

Revision(s) Affected: 1.1, 1.0

Details:

In master mode with four-pin and five-pin options, the SPI module provides four delay fields to specify additional timing relationships between the SPIx_CLK and the $\overline{\text{SPIx\_SCS}}$ and $\overline{\text{SPIx\_ENA}}$ pins. These fields are part of the SPIDELAY register.

The SPIDELAY.C2TDELAY and SPIDELAY.C2EDELAY fields specify additional timings at the start of the SPI transfer. These fields are safe to use.

However, a hazard exists with the use of SPIDELAY.T2EDELAY and SPIDELAY.T2CDELAY, which specify additional timings at the end of an SPI transfer.

If these fields are enabled and the CPU or DMA writes new data to SPIDAT0 or SPIDAT1 during the time period specified by these delay counters, then the SPI internal state machines will lock up.

Workaround:

SPIDELAY.T2EDELAY provides for error-checking by specifying a maximum delay time from the final SPIx_CLK to the slave device de-asserting $\overline{\text{SPIx\_ENA}}$. This feature provides error-checking for a specific error condition, slave de-synchronization. This condition occurs if the slave SPI misses one or more SPIx_CLK edges. Error-checking for this condition can be disabled without any other effect to SPI communications. Therefore, it is **strongly recommended** that SPIDELAY.T2EDELAY be set to 0x00; otherwise, there is a risk of the SPI module becoming locked up.

The SPIDELAY.T2CDELAY provides an automatic delay between the final SPIx_CLK edge and the master de-asserting the $\overline{\text{SPIx\_SCS}}$ field. It is also **strongly recommended** that SPIDELAY.T2CDELAY be set to 0x00; otherwise, there is a risk of the SPI module becoming locked up.

If the slave device requires more hold time between the final SPIx_CLK edge and the master de-assertion of $\overline{\text{SPIx\_SCS}}$ with SPIDELAY.T2CDELAY set to 0, then this delay can be generated by:

- Begin the transfer with the SPIDAT1.CSHOLD field set to ‘1’.
- After the transfer completes, implement the required delay using the DSP CPU or dMAX.
- Set the SPIDAT1.CSHOLD field to ‘0’ to de-assert the $\overline{\text{SPIx\_SCS}}$ pin. (Use an 8- or 16-bit write to avoid writing to SPIDAT1[15:0] and initiating another transfer in the process.)

3 Silicon Revision 1.0 Known Design Exceptions to Functional Specifications and Usage Notes

3.1 Usage Notes for Silicon Revision 1.0

There are currently no known usage notes for the C672x silicon revision 1.0 devices.

3.2 Silicon Revision 1.0 Known Design Exceptions to Functional Specifications

Advisory 1.0.1	*Oscillator and Clock Input: Device Start-up Issue*

Revision(s) Affected: 1.0

Details:

A problem in the OSCIN and CLKIN path can result in the device internal clocks being disabled after power up.

The most noticable indicator of this problem is the lack of any I/O activity after power up (except for the crystal oscillator which is not affected). Specifically, the EM_CLK output pin does not toggle when this problem occurs.

Both the OSCIN and CLKIN paths of the C672x include a special de-glitch cell (labeled “DG” in Figure 19). The de-glitch cell is designed to filter out short glitches that might occur due to system-level noise and to prevent these glitches from disturbing the PLL during normal operation, improving the robustness of the C672x devices.

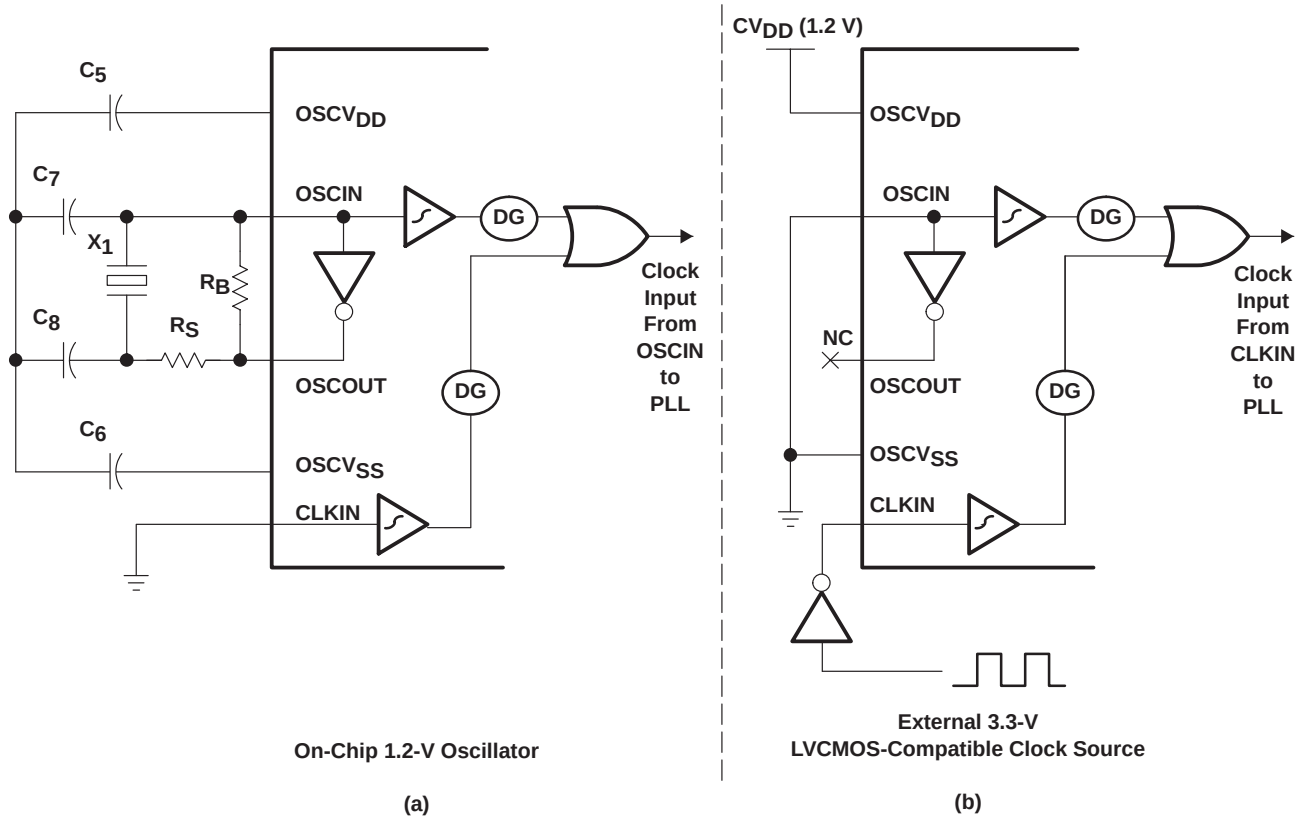


Figure 19. OSCIN and CLKIN Input Paths

Oscillator and Clock Input: Device Start-up Issue (Continued)

However, the de-glitch cell on Revision 1.0 of the C672x devices has a defect that can leave its output in an uninitialized state until an edge appears on its input.

This creates a problem given that one of the two input sources of the C672x clock must be tied low. The input which is tied low does not see any edges upon power up, leaving its corresponding de-glitch circuit in an uninitialized state.

If the uninitialized de-glitch circuit should power up in a logic “1” state, then it effectively blocks the other clock source from propogating past the OR gate shown in Figure 19.

Note that the de-glitch circuit is not initialized by the C672x $\overline{\text{RESET}}$ pin, so that once the C672x powers up in a bad state, the only way to clear the state is to cycle power again. Even cycling power again may not clear the condition.

Also note that it is not possible to “screen” C672x devices for this problem, since the bias of the de-glitch to power up in a logic “0” versus logic “1” state is not predictable and will be sensitive to changes in the operating conditions (e.g., voltage, temperature) of the device.

This problem will be corrected on a future revision of the C672x device.

Workaround: For Revision 1.0 silicon, the following workaround is needed to ensure reliable power up.

The workaround to this problem is to provide at least one edge on the “tied low” pin during reset so that the low level on this pin can propogate through the de-glitch circuit to the OR gate.

This means that the circuit of Figure 19(a) should be modified to provide one or more edges on the CLKIN pin while $\overline{\text{RESET}}$ is low. Likewise, in Figure 19(b), the OSCIN pin requires one or more edges while $\overline{\text{RESET}}$ is low.

After $\overline{\text{RESET}}$ is released, the unused clock input must be driven to a steady logic “0” so that the clock input to the PLL remains periodic.

An external device such as a Microcontroller or CPLD might be used to provide this functionality, especially if it controls $\overline{\text{RESET}}$ to the C672x device.

Alternatively, a simple circuit can be constructed from a 74LV132 (Quad NAND gate with Schmidt-Trigger inputs) to correct the issue. This circuit is illustrated in Figure 20 and Figure 21. The circuit provides a clock of approximatetly 1.5 MHz while $\overline{\text{RESET}}$ is low and drives a low-level output when $\overline{\text{RESET}}$ is high. The two circuits are identical except for the extra resistor divider in Figure 21, which is required because the DSP OSCIN pin operates from CV_{DD}.

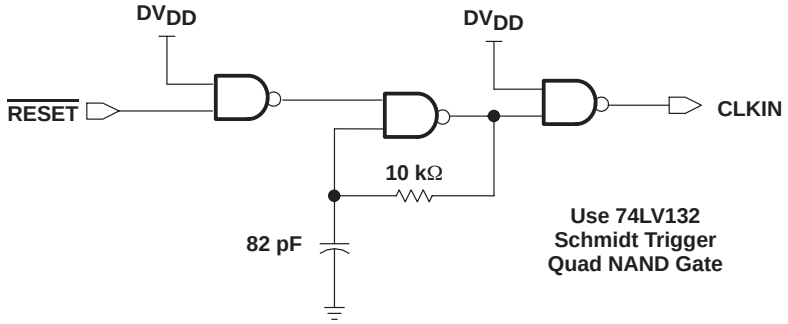


Figure 20. Example Workaround Circuit if Crystal Oscillator is Used as C672x Clock Input

Oscillator and Clock Input: Device Start-up Issue (Continued)

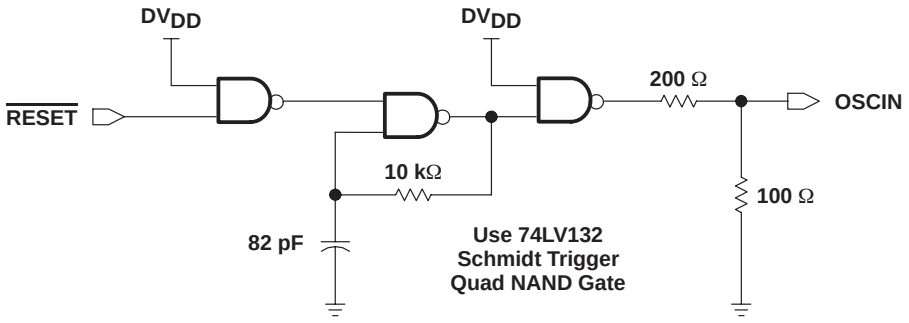


Figure 21. Example Workaround Circuit if CLKIN Pin is Used as C672x Clock Input

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265