Introduction

The DAC348x family is a family of digital-to-analog converter (DAC) devices with many digital features such as FIFO, interpolation filters, quadrature modulator correction circuits, and mixers. It also has flexibility for multi-device synchronization to ensure output phase alignment for each device. In order to operate the DAC correctly, the internal digital logics must be synchronized correctly at start-up. The intend of this application note is to provide references for the digital blocks in the DAC348x devices and describes the steps needed to synchronize these blocks. The same principle can be applied to other Texas Instruments (TI) DAC devices such as the DAC328x family and the DAC317x family.

Key Words:
DAC3482, DAC3484, DAC34H84, DAC34SH84, DAC3282, DAC3283, DAC3174, FIFO, Synchronization.

Acronyms:
PVT = shift in process, voltage, and temperature.

FIFO = first in first out.
RP = read pointer of the FIFO.
WP = write pointer of the FIFO.

ISTR = input pointer strobe (DAC34H84 and DAC34SH84 control input).
FRAME = similar to ISTR. Also establishes data boundaries. (DAC3282, DAC3283, DAC3482, and DAC3484 control input).
SYNC = functions similarly as input pointer strobe. This input also synchronizes the on-chip PLL N-divider to synchronize the multi-device PLLs.
OSTR = output pointer strobe.

DACCLK = DAC sampling clock. This is the final DAC update rate (F_{DAC}) after the interpolation stages.
DATACLK = clock used to latch the LVDS data input.
FIFO-Out Clock = FIFO output pointer clock, where N is the interpolation factor. It could be DACCLK/N or DACCLK/2N depending on the DAC configuration mode.

Reset/Synchronization Sources = sources to synchronize the digital logics. These are typically FRAME, ISTR, SYNC, OSTR, and SIF_SYNC.

1. FIFO Architecture for Synchronization

The DAC348x family (along with the DAC328x and DAC317x family) has new FIFO architecture to allow multiple devices synchronization. This FIFO architecture ensures that the latency of each DAC device is the same, which allows the multiple DAC outputs to be phase aligned. Multi-device synchronization is useful for systems such as beam-forming, active antenna array systems, diversity, and etc.

1.1 The purpose of the FIFO

Typically, the modern digital communication transmitter system consists of clock generation source, digital data source, and digital to analog converters (DAC). The DAC accepts the data from the data source, and the DAC clock provides the time reference to latch the data into the DAC. TI datasheet has specifications of the setup and hold timing requirements (i.e. t_S and t_H, respectively) for the input data and the DAC clock. Figure 1 below shows the setup and hold time diagram.

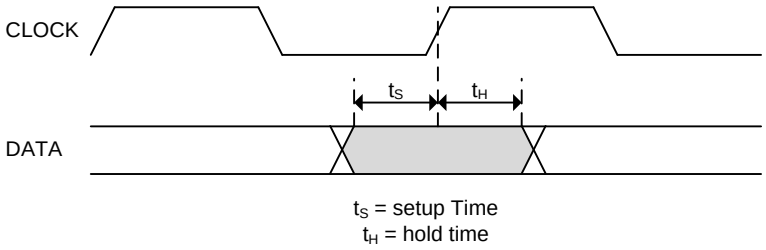


Figure 1. Setup and Hold Time Diagram

System designers will need to make sure the data source and clock source are aligned to meet the DAC setup and hold time requirement for correct output waveform. The time alignment (T_{ALIGN}) must be maintained over the entire system operating range. Any shift in the data source or the DAC clock could cause bit errors because the data cannot be registered correctly.

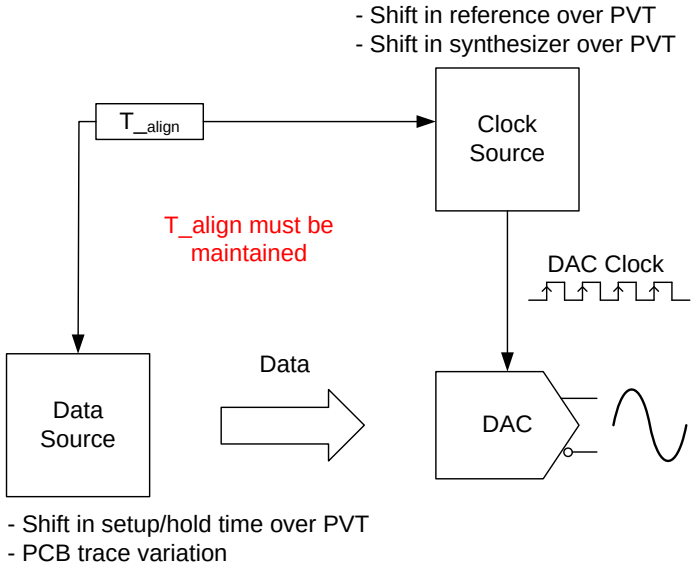


Figure 2. Typical Alignment Requirement for Data and Clock Sources

Figure 2 above shows some of the challenges for meeting the requirements include the shift of reference and clock synthesizer phase variation in clock source over process, voltage, and temperature (PVT), shift of data source output timing specification over PVT, and addition of PCB trace delay variations. These variables add system timing constraints and present constant challenges to designers when ensuring bit-error free operation.

To relax the system timing requirement, most of the TI interpolation and straight DACs have integrated FIFO (first in first out). The FIFO is located at the boundary between the input data and the internal DAC digital blocks. The memory depth is usually 8 or 16 samples deep, and two pointers (write and read pointers) provide reference to the memory during normal operation. Warning flags such as collision alarm are determined by these two pointers.

The purpose of the FIFO can be portrayed as an elastic buffer, which means the current data sample does not have to be written and read at the same time. The write operation can occur some times before the read operation as long as the time variation between the two operations is within the memory depth. As shown in Figure 3, the ultimate goal for the FIFO is to absorb the timing variations between the input data stream and the DAC sampling clock, thus relaxing the system timing.

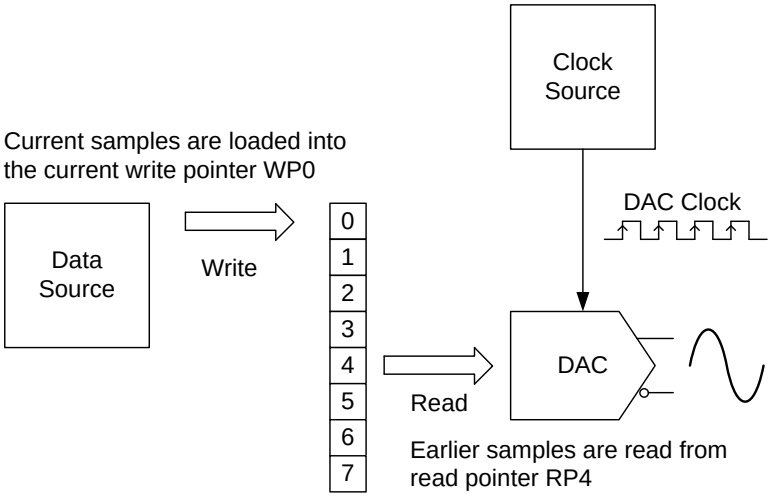


Figure 3. The Use of FIFO to Relax Alignment Requirement

Texas Instruments DAC328x, DAC317x, and DAC348x families require two clock sources: the DATACLK and the DACCLK, in order to function correctly. The DATACLK provides the time reference to latch in the digital data correctly, while the DACCLK provides the clock to the DAC cores and internal digital logics. As shown in Figure 4 of a typical system, the FPGA or custom ASIC provides the data stream along with the DATACLK. The clock generation and distribution device, such as Texas Instruments CDCE62005 or LMK04800 family, provides the reference clock to the FPGA and also the DACCLK source to the DAC.

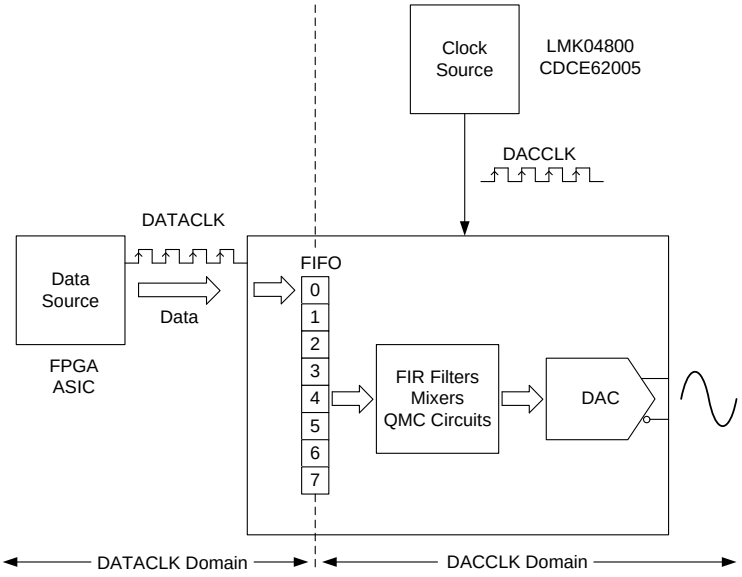


Figure 4. Typical System Implementation for DAC Devices with FIFO

If the DACs do not have a FIFO, then the system must provide the DATACLK and DACCLK, which are precisely aligned. The threshold for the alignment is typically listed as $T_{\_ALIGN}$ requirement in the datasheet. With the FIFO bypassed in the DAC, the input data must be handed to the internal digital logics within a certain time frame in order for the DAC to operate properly. The $T_{\_ALIGN}$ requirement basically meets the timing constraints for the internal digital logics. For a typical system, this $T_{\_ALIGN}$ requirement adds additional design time and cost since designers must account for the delays in the data source in order to match the alignment requirement properly.

1.2 FIFO Operation

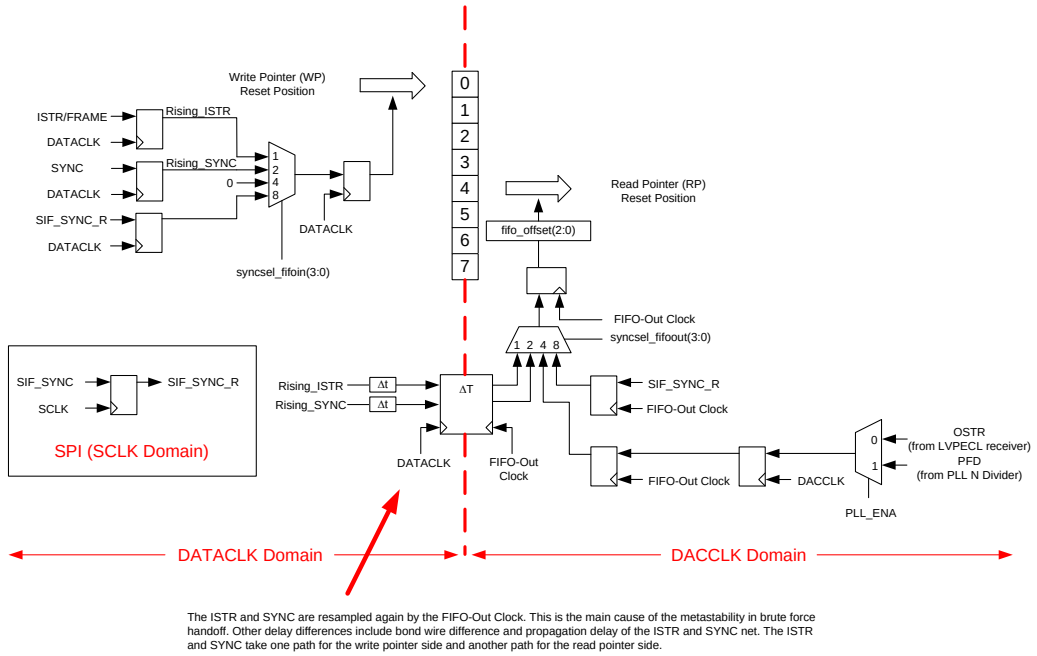
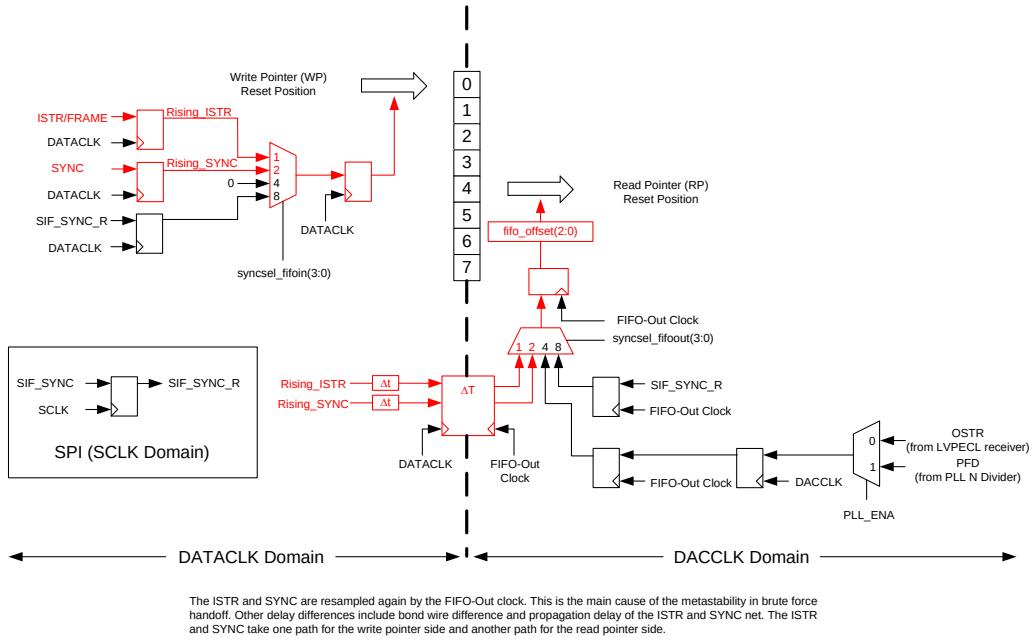


Figure 5. DAC348x FIFO Structure

Figure 5 above shows the generic FIFO architecture of the DAC348x family. Similar concepts can be applied to the DAC328x and DAC317x families. As mentioned previously, the FIFO is located between the boundary of the DATACLK and DACCLK domain. The FIFO is 8 samples deep, where the write pointer (WP) and read pointer (RP) indicate the current memory location. The write pointer can be synchronized by ISTR/FRAME, SYNC, or SIF_SYNC. The read pointer can be synchronized by ISTR/FRAME, SYNC, SIF_SYNC, or OSTR.

Among the DAC328x, DAC317x, and DAC348x family, the most innovative feature for the FIFO is the ability to select and use different synchronization sources to synchronize the FIFO. Depending on the implementation method, the overall latency of the DAC can be controlled. The following sections describe the operations of each mode:

1.3.1 Single Sync Source Mode



The ISTR and SYNC are resampled again by the FIFO-Out clock. This is the main cause of the metastability in brute force handoff. Other delay differences include bond wire difference and propagation delay of the ISTR and SYNC net. The ISTR and SYNC take one path for the write pointer side and another path for the read pointer side.

Figure 6. Single Sync Source Mode

In Single Sync Source mode (shown in Figure 6), the FIFO write and read pointers are reset from the same source, either LVDS ISTR/FRAME or LVDS SYNC. The FIFO read pointer reset signal is handed off between the two clock domains (DATACLK and DACCLK) by simply re-sampling the write pointer reset signal (shown in Figure 7). Since the two clocks are asynchronous, there is a small but distinct possibility of a meta-stability during the signal handoff.

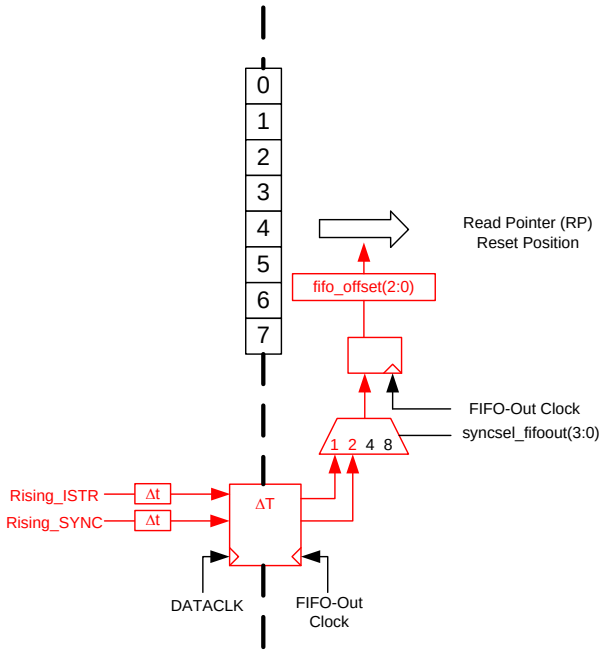


Figure 7. Write Pointer Synchronization Signal Handoff across the FIFO

The meta-stability of the signal handoff is shown below. The zero to one transition of the ISTR signal, for example, is sampled by the DATACLK to reset the write pointer. The same ISTR signal crosses the clock domain and is sampled by the FIFO-Out Clock. The FIFO-Out clock is derived internally from the DACCLK, and is related to the DAC setting and interpolation factor. The table below shows the FIFO-Out Clock rate related to the DAC sampling rate and interpolation factor, N. For instance, N = 2 for 2x interpolation.

Device	FIFO Out Clock
DAC3484	DACCLK/N
DAC3482 Byte Wide Mode	DACCLK/N
DAC3482 Word Wide Mode	DACCLK/2/N
DAC34H84	DACCLK/N
DAC34SH84	DACCLK/N

Table 1. FIFO-Out Clock Rate

Since the ISTR signal is asynchronous to the FIFO-Out clock, the signal could be sampled either one FIFO-Out Clock cycle earlier or one cycle later. The diagram shown below has two FIFO-Out Clock possibilities. FIFO-Out Clock (2) arrives earlier than FIFO-OUT Clock (1) by small time difference. The read pointer reset now happens at a later time. Moreover, the consequence is that the read pointer cannot be precisely reset at some defined time instant, and this poses a limitation for multi-device synchronization because the latency of multiple DAC devices may have variations due the uncertainty of read pointer position reset time instance.

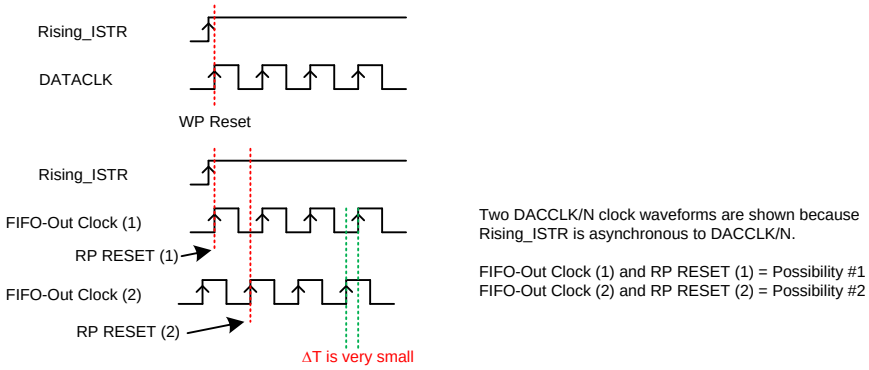


Figure 8. Meta-stability of the Synchronization Signal Handoff across the FIFO

The FIFO in this mode can still absorb the data delay differences due to variations in the digital source output paths or board level wiring. However, it is not possible to ensure that multiple DAC devices have the same latency because the read pointer location of the DAC devices cannot be precisely aligned. For instance, Figure 9 below shows two DAC devices in Single Sync Source mode. The first DAC (DAC#1) resets the read pointer to position number four, RP4, immediately upon the first rising edge of FIFO-OUT Clock because the ISTR transition has enough setup and hold time. This is the ideal case where

both the read pointer and write pointer are reset to the default position at the same time instance.

For the second DAC (DAC#2), the read pointer position resets to RP4 on the second rising edge of FIFO-OUT Clock. The ISTR transition does not have enough setup and hold time for the first rising edge of FIFO-OUT Clock and have to wait for the second rising clock edge to register the reset.

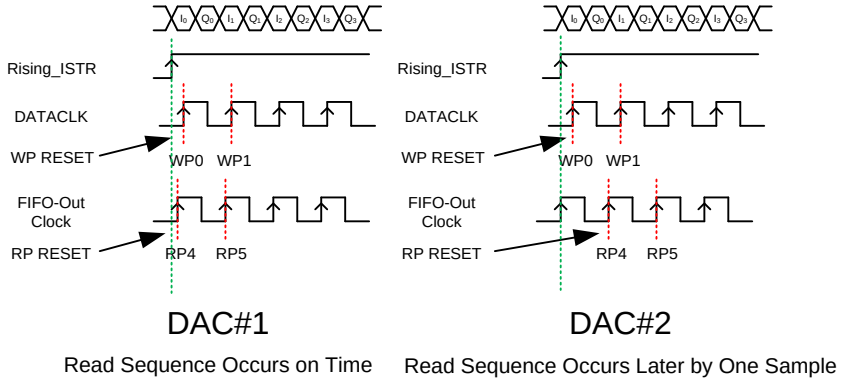


Figure 9. Reset Time Difference of FIFO Read Pointer Position in Single Sync Source Mode

When comparing between the two DACs, the read pointer position slips by one sample. The two diagrams below elaborate the write and read sequence over the next eight time instance. For DAC #1, data S0 is loaded into write memory location #0 at time instance #0, and also write sequence #0. The read memory location #4 does not have any data at the time until four sequences later. After eight sequences, write sequence #8 load data S8 into write memory location #8, and earlier sample S4 is loaded into the DAC at time instance #8.

**Note: During DAC initialization, the DAC output is held at mid-code by driving the TXENABLE pin LOW. The DAC can start transmitting the data once the FIFO are filled with proper samples by pulling TXENABLE pin HIGH.

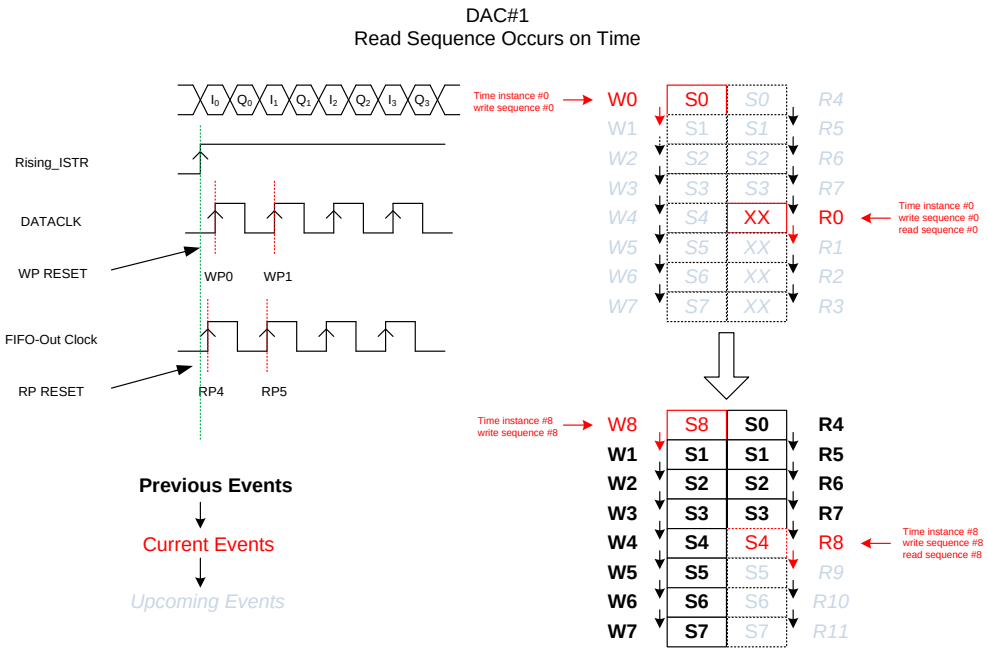


Figure 10. DAC#1 FIFO Write/Read Sequence (Read Sequence Occurs On Time)

The same derivation is done on the second case DAC#2. For this case, the read pointer reset does not occur until time instance #1 and write sequence #1 complete. Therefore, the read pointer reset is delayed by one sample. After eight sequences, the earlier data S3 is loaded into the DAC at time instance #8. Note that DAC#1 and DAC#2 have one sample difference at the same exact time instance.

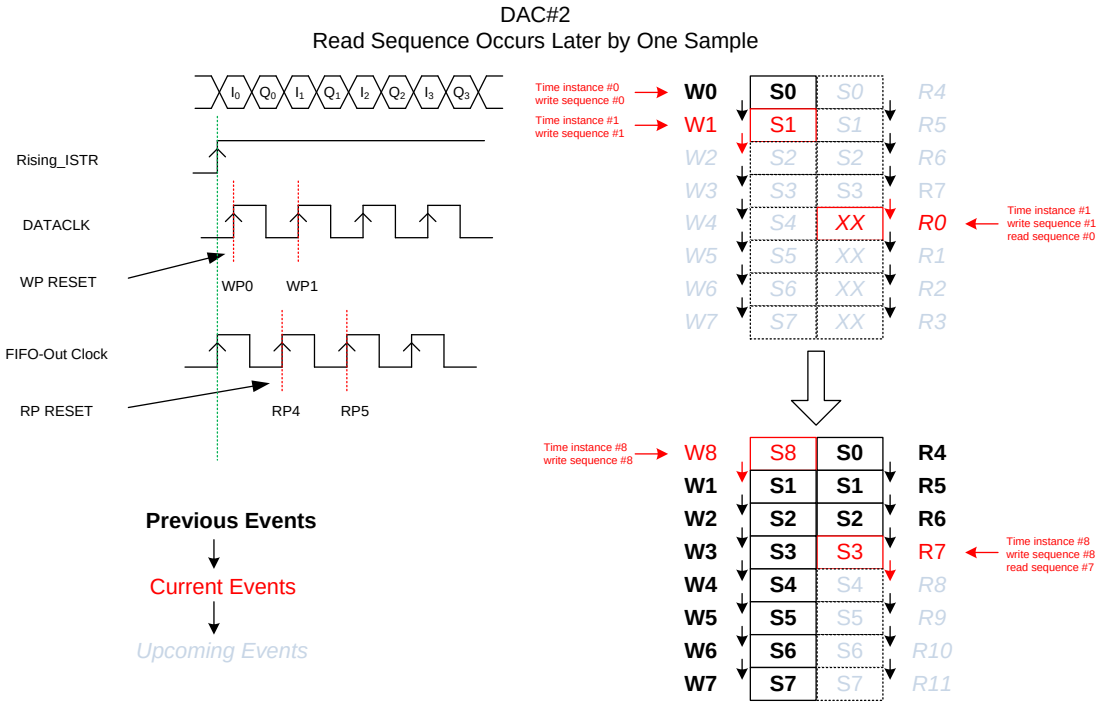


Figure 11. DAC#2 FIFO Write/Read Sequence (Read Sequence Occurs Later by One Sample)

The synchronization signal such as ISTR/FRAME or SYNC has to cross the DATACLK domain to the DACCLK domain, and the clock domain crossing could introduce timing variation in time unit of DACCLK. Along with the reset time meta-stability of the FIFO read pointer, the total latency variation could have time units of both the FIFO-Out Clock period and DACCLK period. Furthermore, although less likely, other factors such as DATACLK to DACCLK skew, PVT, and internal signal transition variation due to internal net bond wire differences can introduce additional latency control variables.

Ultimately, if the Single Sync Source mode is used, the write pointer and read pointer delta among the DAC devices cannot be precisely controlled, and the overall latency of all the DAC devices cannot be matched. Although this mode is very suitable for simple system implementation without additional timing control, it is not practical for applications requiring multi-device synchronization or precise latency control. To facilitate latency synchronization, TI recommends Dual Sync Sources mode.

1.3.2 Single Sync Source mode with SIF_SYNC

Besides synchronizing the FIFO write and/or read pointers, the ISTR/FRAME or the SYNC signal also aligns the FIFO's internal data formatter to ensure the I and Q signals are aligned properly. For instance, Figure 12 below shows the data format of the DAC3484 in byte wide interface mode. The rising edge of DATACLK occurs at upper eight bits of sample A and sample B (i.e. A[15:8] and B[15:8]), while the falling edge of DATACLK occurs at lower eight bits of sample A and sample B (i.e. A[7:0] and B[7:0]). The zero to one transition of ISTR/FRAME or SYNC signal is used to indicate the beginning of sample A. If the FRAME or SYNC signal does not make the transition

correctly, the internal pattern could be swapped since the rising edge of DATACLK could also occur at the beginning of sample B. Same concept also applies to DAC328x and DAC317x family.

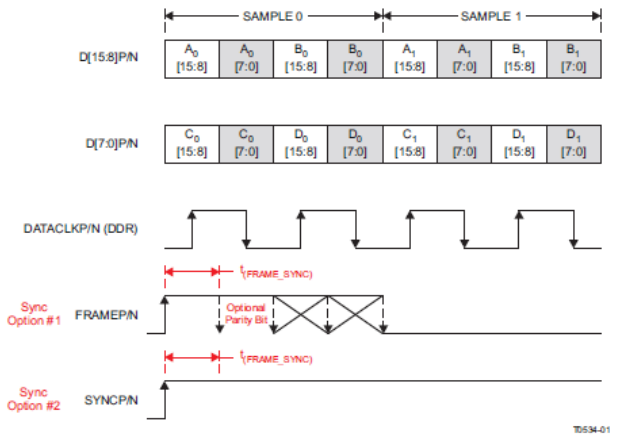


Figure 12. Byte Wide Data Transmission Format (copy from Figure 53 of SLAS749C)

For the DAC3482 in word wide mode, DAC34H84, and DAC34SH84, each sample is located on the distinctive DATACLK edge as shown in Figure 13. Even if the ISTR/FRAME or SYNC signal is not present, the samples will not be swapped in position.

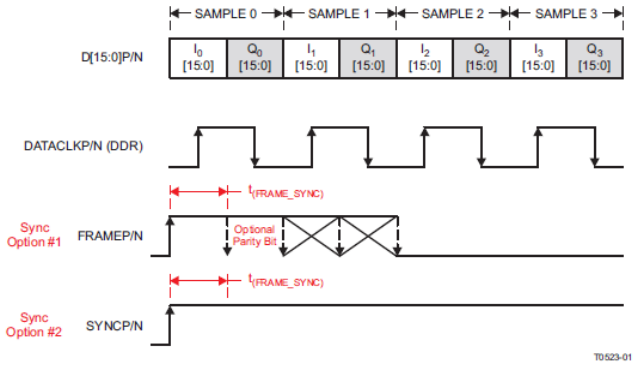


Figure 51. Word-Wide Data Transmission Format

Figure 13. Word Wide Data Transmission Format (copy from Figure 51 of SLAS748)

In some applications where the FPGA or ASIC has limited IO pins, the control signal can be eliminated by using the SIF_SYNC SPI register bit. The FIFO (and other parts of the circuit such as NCO, QMCs, etc) can be synchronized by a zero to one transition of the SPI register bit – SIF_SYNC. Figure 14 below shows the signal path for the SIF_SYNC. A zero-to-one transition of the SIF_SYNC bit in CONFIG31 will be first registered by the SPI bus in the SCLK domain. The signal is then re-sampled by the DATACLK to synchronize the FIFO write pointer position. It is also re-sampled by the FIFO-OUT Clock to synchronize the FIFO read pointer position.

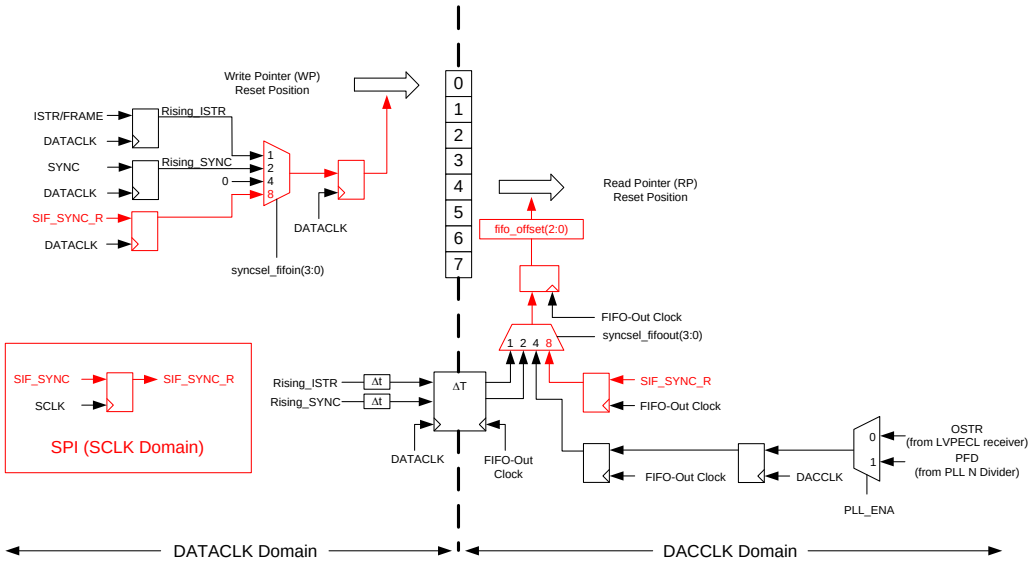


Figure 14. Single Sync Mode Using SIF_SYNC

Since the SIF_SYNC signal is asynchronous to both the DATACLK and FIFO-OUT Clock, not only it is possible that the read pointer position may vary from time-to-time and device-to-device, the sample sequence loaded into the FIFO memory may also be different. The example below (shown in Figure 15) demonstrates this idea.

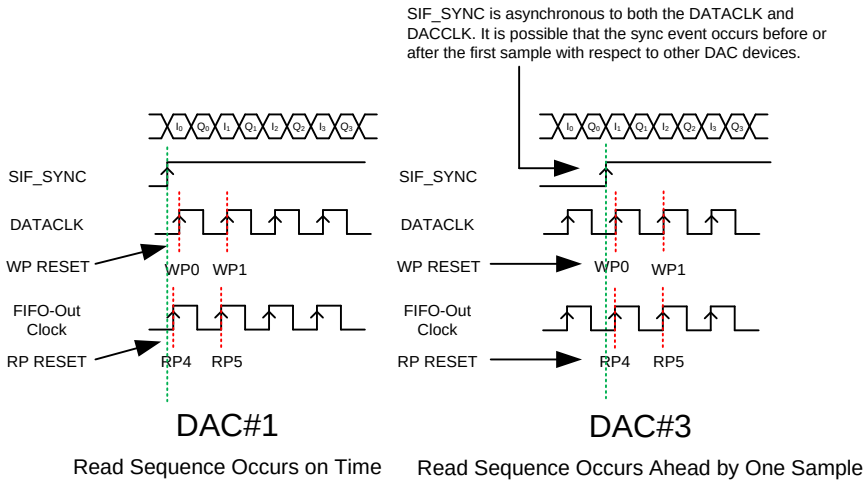


Figure 15. Reset Time Difference of FIFO Read Pointer in SIF_SYNC Mode

The ideal DAC#1 (also shown in the previous section) has both write pointer and read pointer reset at sample S0. In DAC#3, the SIF_SYNC source arrived one sample later. The write pointer and read pointer reset to default at S1 instead of S0, and effectively, DAC#3 load the same memory into the FIFO one sample earlier than DAC#1 (as shown in Figure 16). Comparison between Figure 10 and Figure 16 indicates the sample time difference.

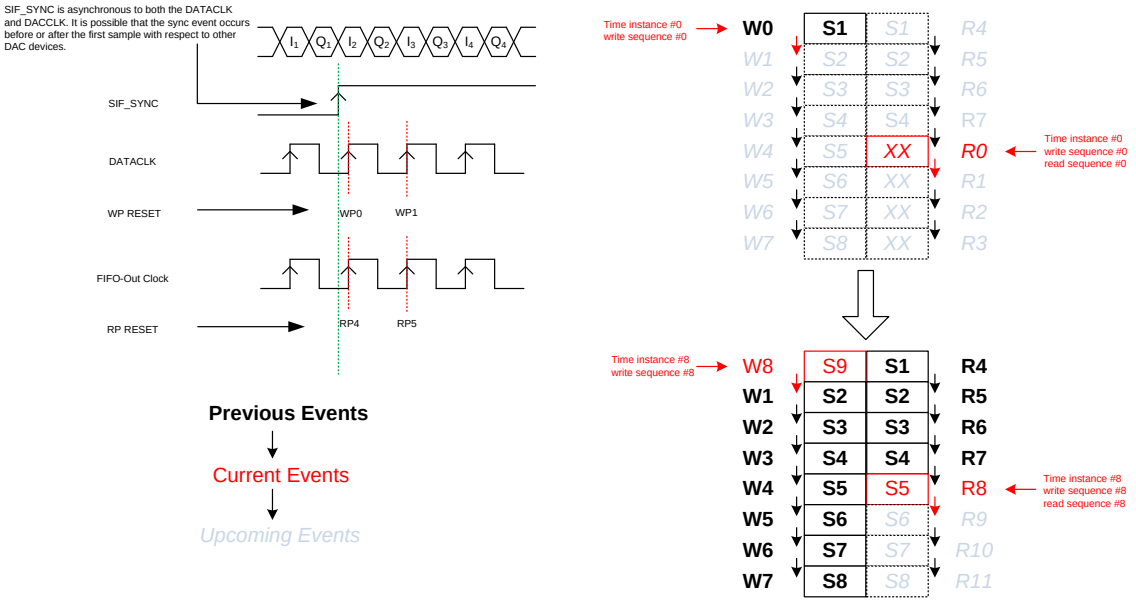


Figure 16. DAC#3 FIFO Write/Read Sequence (Read Sequence Occurs Ahead by One Sample)

Primarily, this mode allows the user to save I/O pins in the FPGA or ASIC. With the existing read pointer uncertainty and also the addition of write sequence uncertainty (due to the asynchronous nature of SIF_SYNC to both the DATACLK and the DACCLK), latency control remains a challenge and is not practical in this mode. If the system requires multi-device synchronization, TI recommends the Dual Sync Sources mode.

1.4 Optimize FIFO location for Single Sync Source Mode

The default FIFO write pointer position of WP0 and read pointer position RP4 creates the optimal FIFO setup. The read pointer position is placed in the middle of the FIFO to absorb timing variations in either the write direction or the read direction shown in Figure 17. An example for the worst case timing variation in the write direction would be the DATACLK and the data stopped completely while the read operation continues with the DACCLK running. In the default setup, the FIFO could absorb at most three samples before collision. The same principle can be applied to the worst case timing variation in the read direction where the DACCLK and the read operation stop completely while the write operation continues. In this case, the FIFO could also absorb three samples before collision. Since the FIFO loops around, the optimal write pointer position is in the beginning and the read pointer position is in the middle.

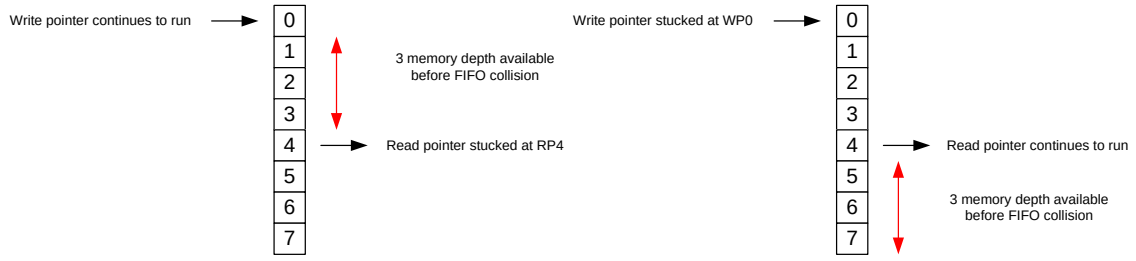


Figure 17. FIFO Memory Buffer for Both the Write and Read Direction

Since the Single Sync Source mode is not recommended for applications requiring precise latency control, the following section only describes the optimal setup for system with only single DAC device or multi-DAC devices system without the need for latency synchronization. As mentioned in the previous section, the Single Sync Source mode could cause the read pointer position to slip in either direction. For the SIF_SYNC operation, both the write pointer position and read pointer position could slip in either direction. Therefore, it is possible that the spacing between the read pointer position and write pointer position is not optimal at start-up.

To help designers optimize the FIFO setup at start-up, the DAC families includes FIFO alarms to indicate the pointer position delta. The three primary alarms are FIFO 2-away, FIFO 1-away, and FIFO collision. FIFO 2-away indicates absolute read and write pointer position difference (i.e. either write to read pointer position or read to write pointer position) have two spaces. FIFO 1-away indicates the absolute pointer position difference have one space. FIFO collision indicates that the FIFO pointers have collided and requires re-synchronization. Figure 18 shows the possible conditions to trigger the alarms.

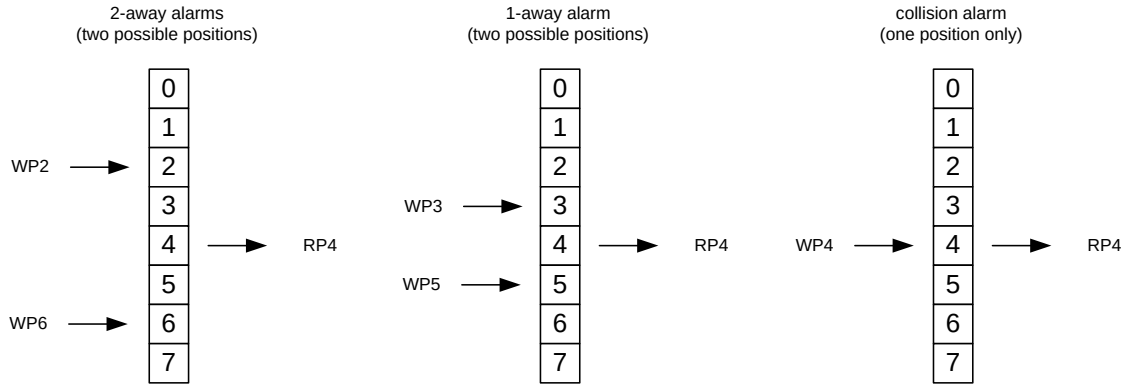


Figure 18. Possible Alarm Conditions (Assuming RP4 Occurs)

The alarm registers in the DAC family have a memory system and do not self-clear upon system recovery. This is important for designer if the system diagnosis algorithm only consisted of polling architecture (as oppose to both polling and interrupt service routine****). If the alarms do self-clear upon system recovery, there are possibilities that the polling interval missed the alarm, and the system will not be able to detect the alarm and issue remedies for the error.

****the DAC family do have dedicated alarm signal output to trigger interrupt service routine (ISR) if the system design requires such provision.

Upon system power up, the alarm registers could detect various alarms due to power up transitions. For instance, the DACCLK could be running before the DATACLK, and the FIFO alarms could trigger. Therefore, it is important to clear the alarm registers by writing all zeros to the alarms registers before reading back the register. After the system clocks and the DAC have settled, the system diagnosis algorithm should check the alarms registers. If the FIFO alarms remain triggered, the read pointer position (programmed in

the FIFO offset register) can be adjusted accordingly to clear the alarms and optimize the FIFO spacing.

Table 2 shows the typical FIFO alarm behavior of each read pointer position after the initial start-up synchronization. The experiment consists of five independent tests of eight different start-ups. For each start-up, the FIFO write pointer position is always reset to WP0, and the FIFO read pointer position is programmed from RP0 to RP7. Therefore, different FIFO alarms will trigger at each start-up.

Read Pointer Position	Test #1	Test #2	Test #3	Test #4	Test #5
RP0	collision	collision	collision	collision	collison
RP1	1	1	1	1	1
RP2	2	2	2	2	2
RP3	no	no	no	no	no
RP4	no	no	no	no	no
RP5	no	no	no	no	no
RP6	2	2	2	2	2
RP7	1	1	1	1	1

Table 2. FIFO Alarm Distribution vs. FIFO Offset Location (Single Sync Source Mode – ISTR only)

When compared among the power up status with various FIFO offset position, the data has shown the optimal "green" region where the least amount of FIFO alarms occurred. The actual "green" region may shift due to the design of Single Sync Source mode, DATACLK to DACCLK skew, sequence of DATACLK and DACCLK presence, and PVT. If the optimal region of FIFO offset is found, the FIFO offset adjustment process can quickly yield the best FIFO pointer position.

Table 3 shows the test result of Single Sync Source mode with SIF_SYNC. Note that the SIF SYNC only mode has more variations of the alarm distribution due to the SIF_SYNC being asynchronous to both the DATACLK and DACCLK.

Read Pointer Position	Test #1	Test #2	Test #3	Test #4	Test #5
RP0	1 away	1 away	1 away	1 away	collision
RP1	2 away	1 away	2 away	1 away	2 away
RP2	0	0	2 away	0	0
RP3	0	0	0	0	0
RP4	0	0	0	0	0
RP5	2 away	0	2 away	0	0
RP6	2 away	2 away	2 away	2 away	1 away
RP7	1 away	1 away	1 away	1 away	1 away

Table 3. FIFO Alarm Distribution vs. FIFO Offset Location (Single Sync Source Mode – SIF_SYNC only)

The following software loop demonstrates the alarm checking algorithm and the associated FIFO offset adjustment for optimal write and read pointer spacing. Each path starts off with the default FIFO offset and the checking of the FIFO alarms. Depending

on the alarms detected, the FIFO offset is adjusted accordingly. After the adjustment, the alarms are rechecked again to ensure the FIFO has the optimal spacing. Since the FIFO pointers in this mode have uncertainties in the exact pointer position, the loop has implemented additional error checking and FIFO offset corrections. Each path can be re-used depending on the alarm status.

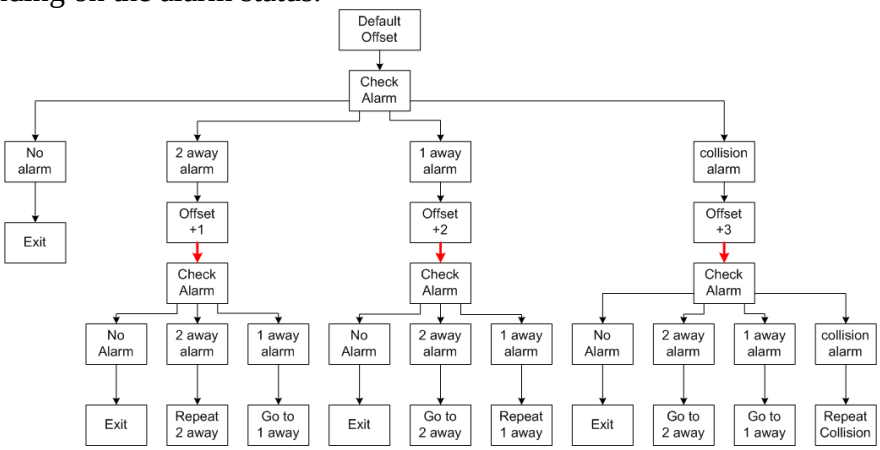


Figure 19. FIFO Offset Adjustment Algorithm

The ultimate goal for FIFO position optimizing in Single Sync Source mode is to adjust the read pointer such that it has enough space from the write pointer in either direction of the FIFO space movement. This will be the best approach for system with only single DAC device or multi-DAC devices system without the need for latency control.

1.5 Dual Sync Sources Mode

The Dual Sync Sources mode is the recommended mode of operation for those applications that require precise control of the output timing and latency. In Dual Sync Sources mode, the FIFO write and read pointers are synchronized independently by two sources. The FIFO write pointer is reset using the LVDS ISTR/FRAME or SYNC signal, and the FIFO read pointer is reset using the OSTR signal (either from external LVPECL source or the on-chip PLL's internal PFD frequency). The OSTR signal can control the latency (or phase) of the output for either a single chip or multiple chips. Multiple DAC devices can be fully synchronized in this mode.

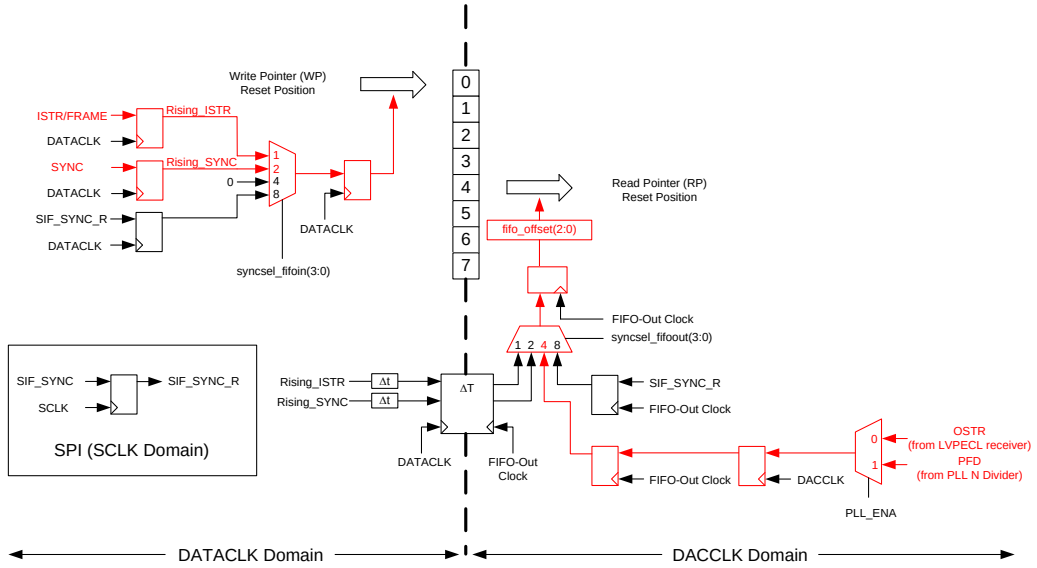


Figure 20. Dual Sync Sources Mode

The ISTR or SYNC signal should come from the data source and has the same timing relationship as the DATACLK, and the OSTR signal comes from the clock source and has the same timing relationship as DACCLK. This implementation does not need to re-sample the reset signal in two different clock domains, thus eliminating the possibility of meta-stability during reset signal handoff.

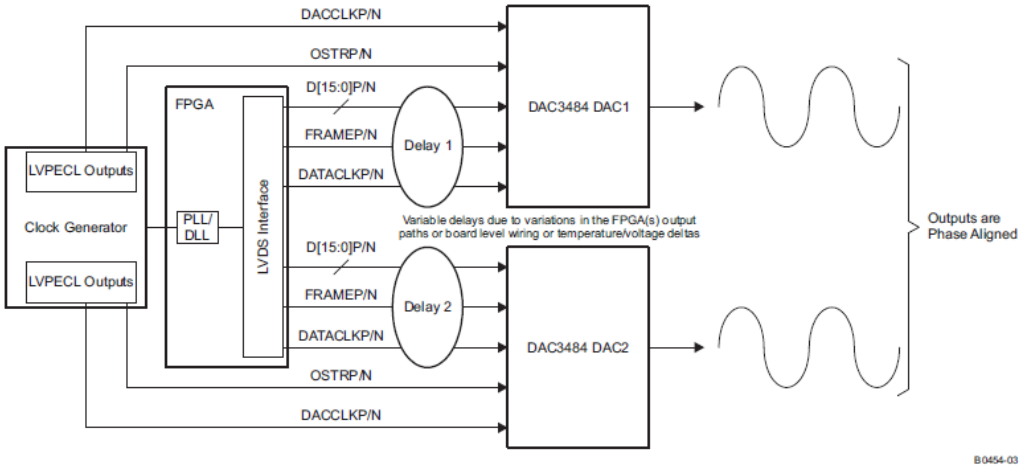


Figure 60. SynchronizationSystem in Dual Sync Sources Mode with PLL Bypassed
Figure 21. Typical Dual Sync Sources System Setup (Figure 60 of SLAS749).

The two key principles in achieving the multi-device synchronization are the following:

1. The data is loaded into the FIFO in the same order and same FIFO location for all the DAC devices. The ISTR/FRAME or SYNC signal needs to be aligned with the first sample properly in order to load the data in the correct sequence.

2. All the DAC devices read same FIFO location at the same time to achieve the same latency. In order to meet this condition, the OSTR and DACCLK signals must have the same delay among the DAC devices in order to reset the FIFO read pointer at the same time.

   The following example demonstrates this idea. Shown in Figure 22, the system contains two data sources and two DAC devices. Each data path to the DAC device has different time delay due to shift in data source timing over PVT and PCB trace variations. Path #1 has less delay than Path #2. To keep the analysis simple, this example utilizes DAC input with double data rate format, which latches one channel data on the rising edge of DATACLK and another channel data on the falling edge of DATACLK (i.e. DAC3482 Word Wide Input Format or the DAC34H84 Input Format). The time difference between the paths is two DATACLK cycles, or two samples in double data rate (DDR) fashion, which is shown in Figure 23.

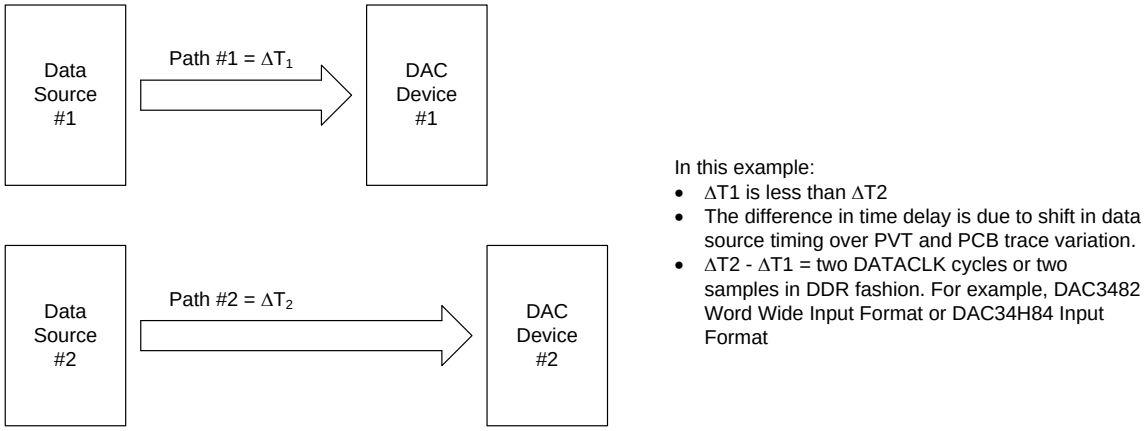


Figure 22. Multi-Device Synchronization Example:
Path#2 Has Additional Two Sample Delay when Compared to Path#1

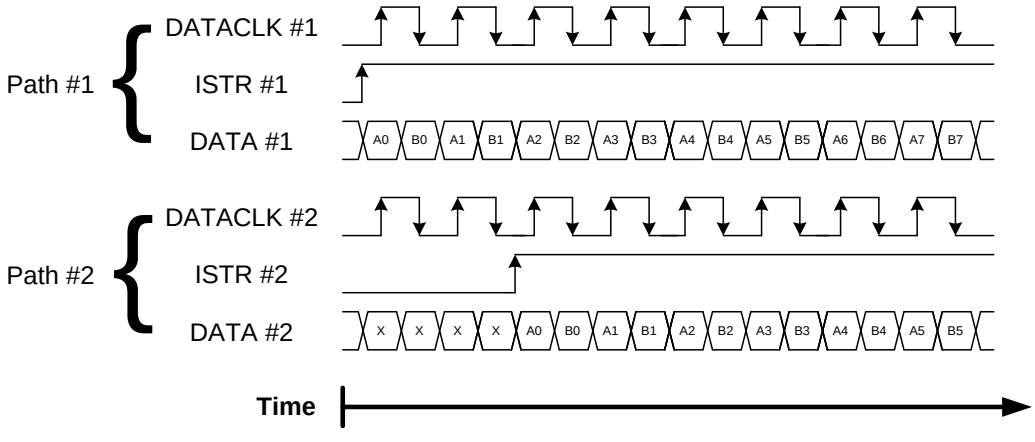


Figure 23. Multi-Device Synchronization Example:
Data Sequence vs. Time between Path#1 and Path#2

The rising edge of ISTR signal has two purposes: to indicate the first sample of the data sequence, and to reset the FIFO write pointer position to default WP0. This indication allows the data of different data paths to be loaded in the same sequence regardless of the data path delay. At this point, this implementation allows the system to meet the first requirement of multi-device synchronization.

To continue the demonstration, Figure 24 shows FIFO write process of the system. Three snap-shots of the FIFO are demonstrated. The first snap-shot shows the time instance #2 where Path#1 is loading sample S2, while Path#2 is loading the initial sample, S0. The second and third snap-shots show the subsequent FIFO states three and six samples later, respectively.

The first and second snap-shots have demonstrated that the data sequences of the two paths are loaded into the FIFO in the same order. Once the FIFOs of the two paths have been fully loaded, the same data sample can be released from the FIFO at the same time. The synchronization of the FIFO release time is based on the OSTR signal. Per the second requirement of multi-device synchronization, the system implementation must guarantee that the OSTR and DACCLK timing are the same for all the DAC devices in the system. Once this requirement is met, all the DAC devices can read the same data sequence at the same time. Since the rest of the digital logics after the FIFO operate in the DACCLK domain, the DAC devices in the system can achieve the same latency.

To simplify the analysis, the FIFO read process is not presented until the third snap-shot of the FIFO. In actual system implementation, the FIFO write and read sequence is synchronized during the DAC initialization stage, and the TXENABLE of the DAC is held LOW to disable the DAC output. In this process, the FIFO memory can be flushed out without the DAC outputting transient waveforms, which may damage subsequent signal chain devices. Once the FIFO is initialized properly with the proper samples loaded, the TXENABLE can be pulled HIGH to enable the DAC output for transmission.

Ultimately, the FIFO absorbs the input path delay difference to allow equal latency after the FIFO. This example has input path delay difference of two samples, which is within the FIFO depth. If the input path delay is greater than the FIFO depth, then the FIFO cannot absorb the delay difference, and latency alignment cannot be achieved.

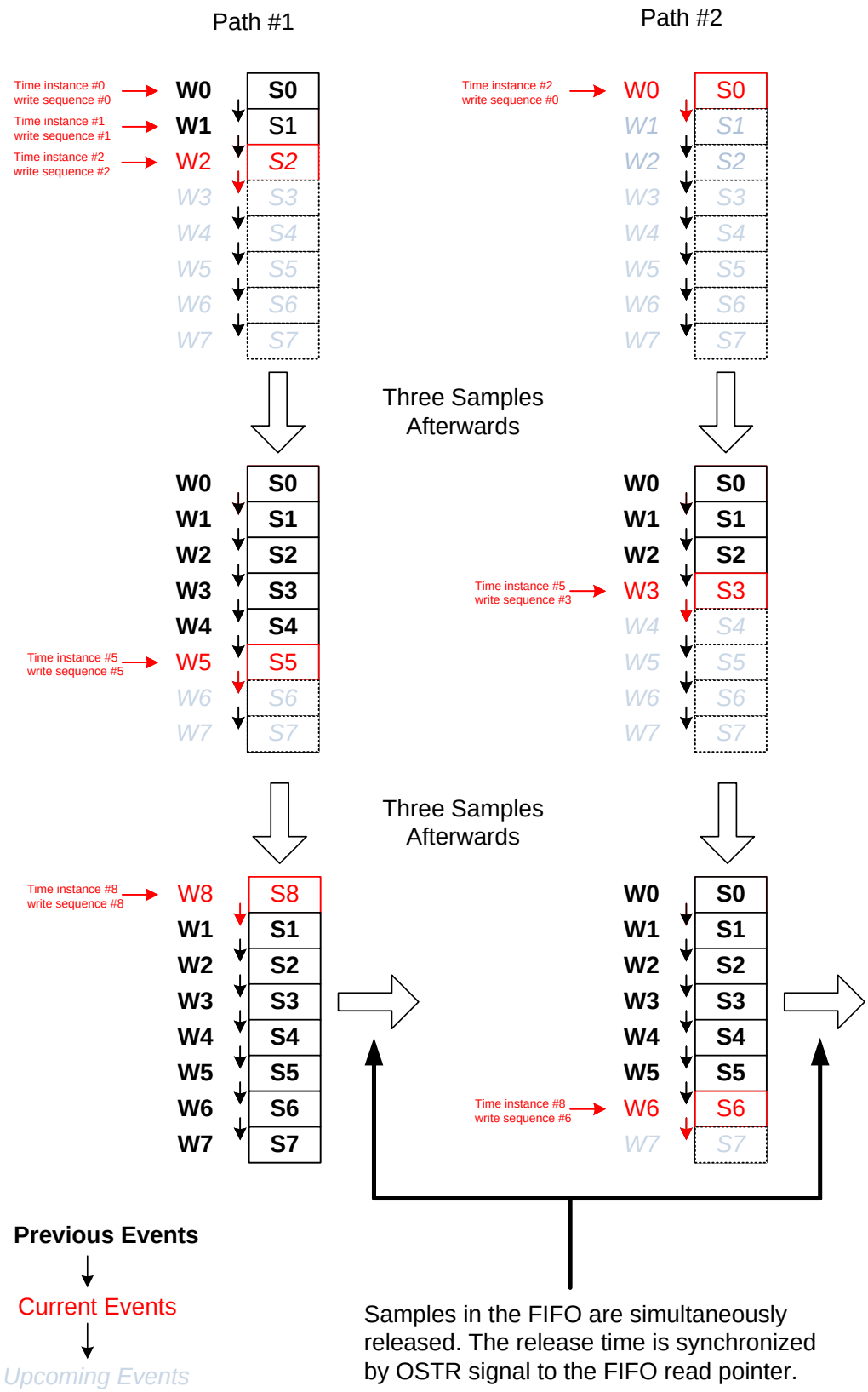


Figure 24. Multi-Device Synchronization Example: FIFO State Snap-shots

Beside data, control signals such as SYNC and ISTR, are treated as data and can be read from the FIFO at the same time instance. An example is shown in Figure 25, where the rising edge of SYNC pulse is registered at the same time sample S4 is latched. When all the DAC devices in the system release RP4 in the FIFO, the SYNC pulse stamp will also be released. This stamp will be registered by the appropriate digital logics (based on DAC programming) and used for logic synchronization. This is useful for synchronizing internal digital logics such as NCO and QMC. These circuits can process the signal amplitude and phase. By setting these circuits to look for the synchronized control signals, multiple devices can adjust the signal amplitude and phase at the same precise moment.

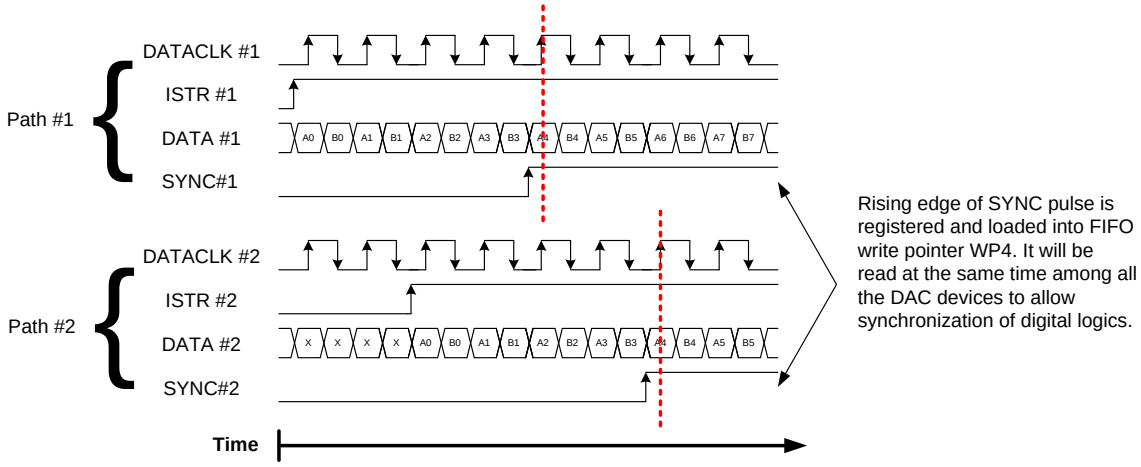


Figure 25. Multi-Device Synchronization Example: SYNC Pulse Loading for Logic Synchronization

For the DAC348x family configured as external clock mode, the external clock synthesizer and distribution circuits such as the LMK04800 family and the CDCE62005 will need to ensure the OSTR and DACCLK signals going to each DAC348x device have zero skew. The timing diagram is shown in Figure 26

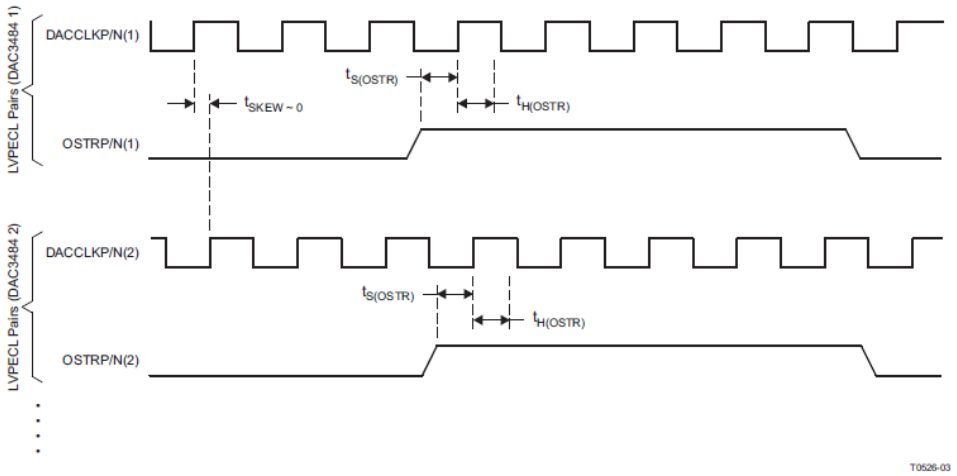


Figure 61. Timing Diagram for LVPECL Synchronization Signals

Figure 26. Timing Diagram for LVPECL Synchronization Signals of DAC3484 (Figure 61 of SLAS749)

For the DAC348x family with PLL mode enabled, multiple DAC348x device on-chip PLLs must be synchronized in order to have zero skew OSTR and DACCLK signals. As shown in Figure 27, the on-chip PLL is synchronized by the rising edge of LVDS SYNC signal going to the N-divider circuit. The idea is that if the reference frequencies of the PLLs are aligned in time, then the OSTR and DACCLK signals will also be aligned.

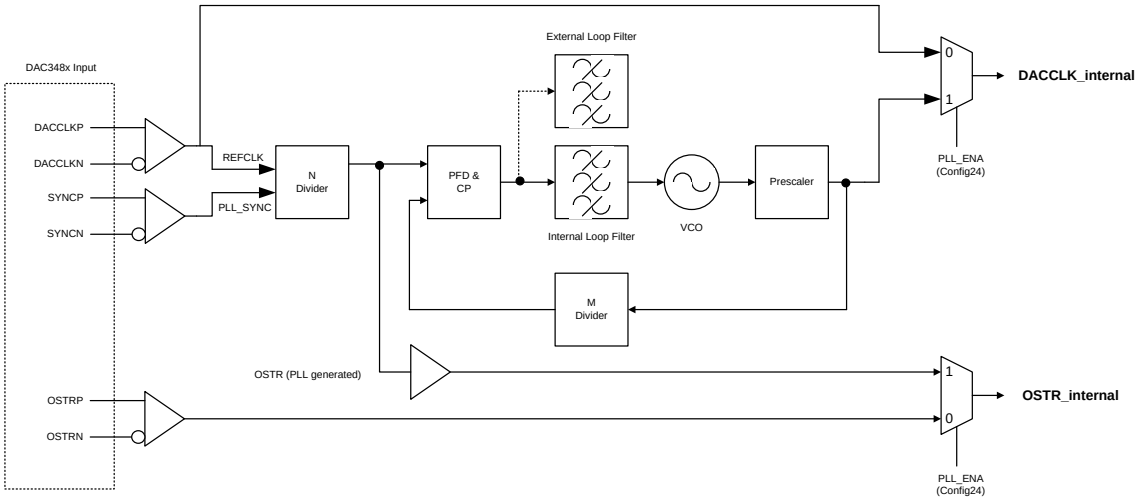


Figure 27. DAC348x On-Chip PLL DACCLK and OSTR Generation

One important note about the OSTR signal in PLL mode is that the OSTR is the PFD frequency after the N-divider. Therefore, the reference clock (REFCLK), N-divider, and M-divider must be configured to meet the OSTR frequency requirement and also the PLL lock requirement.

1.6 Optimal FIFO position setup for Dual Sync Sources mode

In Dual Sync Sources mode, the FIFO read pointers of multiple DAC devices can start at the same location at the same exact time instance given that the OSTR and DACCLK signals among all the devices have the same delay. At the data input side, as long as the load sequences of each DAC devices are the same, the FIFO's memory depth can absorb some variations at the data input side. The FIFO has limited length and has to absorb timing variations from both input and output pointer side. Therefore, it is important that the FIFO position starts in an optimal position.

In the previous example of Path#1 and Path#2, both paths have to have the same read pointer position in order to have the same latency. However, the write side of Path#1 has less delay than Path#2, and Path#1 will load data into the FIFO earlier than Path#2. Therefore, Path#1 has narrower FIFO write to read pointer position gap than Path#2.

The FIFO offset adjustment procedure is shown in Figure 28. The green arrow shows the write to read pointer gap, while the red arrow shows the read to write pointer gap. When the pointer gap is narrower, the chance of FIFO collision is higher. Counting from the write pointer to read pointer, Path#1 with read pointer position of 4 only has one memory gap. If somehow the DATACLK or DACCLK has interruption or the skew has changed, for instance, the clock generator started to shift the delay of the OSTR and DACCLK signals over PVT, the FIFO read pointer could reset earlier and cause collision.

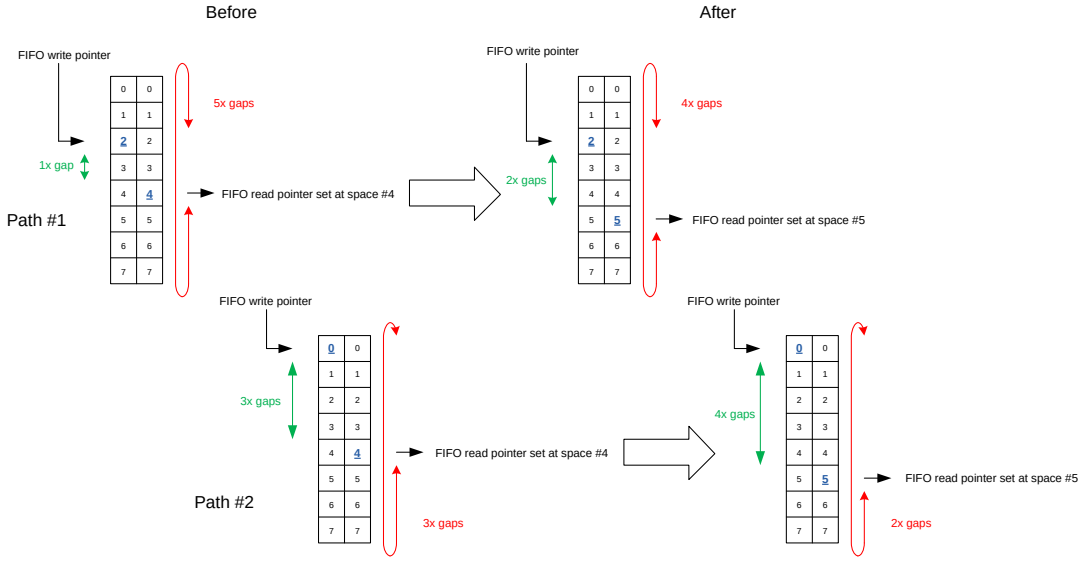


Figure 28. FIFO Offset Adjustment for Dual Sync Sources Mode

By changing FIFO read pointer position from RP4 to RP5, the write pointer to read pointer gap (shown in green) has increased from one to two in Path#1. The read pointer to write pointer gap (shown in red) has decreased from five gaps to four gaps, but it is the best option without disturbing the setup for Path#2. If the read pointer position is increased further, then Path#2 will not have enough read pointer to write pointer gap. If either the DATACLK or DACCLK is disturbed, the FIFO does not have much buffer and

collides with higher probability. As mentioned in the previous section, designers should check the FIFO alarms upon start-up to ensure each device has optimal FIFO position.

2 Digital Logic Synchronization

Besides the FIFO, other digital logics such as clock divider, data formatter, mixers, and quadrature modulator correction features, require initialization during start-up to ensure proper operation. Each of the digital blocks is described in this section.

2.1 FIFO and data formatter (configured and synchronized upon initial start-up):

The FIFO synchronization method would depend on the end application requirement. Depending on the Single Sync Source mode or Dual Sync Sources mode configuration, the *syncsel_fifoin(3:0)* and the *syncsel_fifoout(3:0)* in Config32 register can be programmed accordingly.

For the DAC3482 and DAC3484, the data interface options can be either 16-bit word wide interface or 8-bit byte wide interface. The FIFO includes a data formatter to format the FIFO memory accordingly. The synchronization source can be selected by *syncsel_dataformatter(1:0)* in Config31, with the option including the FRAME or SYNC. As mentioned previously, the rising edge of FRAME or SYNC can be used to establish data boundary to ensure the data are interpreted correctly.

The data formatter is not needed for the DAC34H84 and DAC34SH84 since the 32-bit interface forces the A and C channels always latched on the rising edge of DATACLK and B and D channels always latched on the falling edge of DATACLK.

The DAC3482 word wide interface operates in the similar fashion and some users may choose to use SIF_SYNC to synchronize the FIFO. In this case, the *syncsel_dataformatter(1:0)* in Config31 can be set to “10” or “11” option for no sync options.

2.2 Clock Divider (configured and synchronized upon initial start-up):

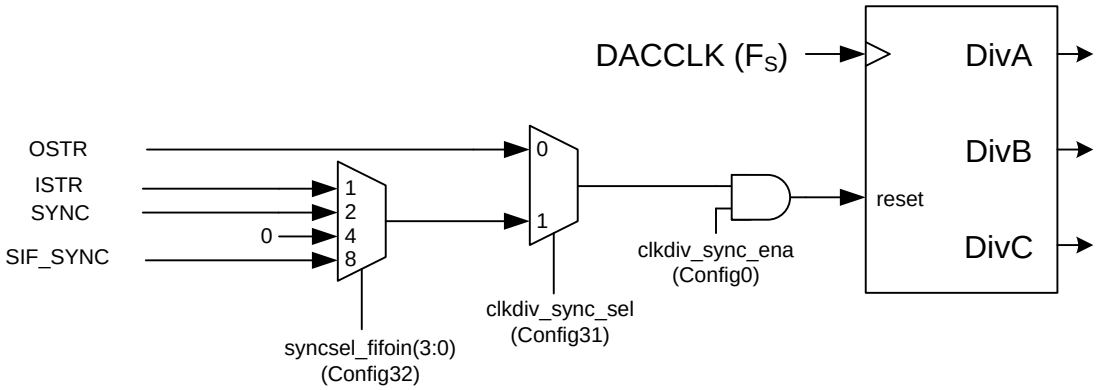


Figure 29. Simplified Clock Divider Block Diagram

The purpose of the clock divider is to provide divided-down clocks. Figure 29 above shows the simplified clock divider structure. The source of the clock divider comes from the DACCLK, and the clock divider can be reset by the OSTR signal or the FIFO input pointer reset source.

The clock divider needs to be synchronized at start-up along with the FIFO and data formatter (if needed). The clock divider synchronization source will be the same as the FIFO output pointer synchronization source. In Single Sync Source mode, the clock divider will be synchronized by ISTR, SYNC, or SIF_SYNC. In Dual Sync Sources mode, the clock divider will be synchronized by the OSTR signal (either externally provided to the DAC348x or internally generated by the PLL PFD).

2.3 Double Buffered Registers (adjusted during system operation).

FIR filters and coarse mixers have pre-set algorithm values. Once these circuits are enabled at start-up along with start-up synchronization routine (i.e. FIFO and clock divider synchronization), additional synchronization is not required. Since the NCO and QMC logics have programmable algorithms, the circuits require additional synchronization of the double-buffered register. A double-buffered register is designed to prevent instantaneous change to logic behavior when the desired logic behavior requires more than one serial port interface (SPI) configuration. For instance, the NCO register requires two 16-bit frequency information and one 16-bit phase information. The SPI register is 16-bit per register, and if the NCO logic changes instantaneously for every SPI write transaction, the output frequency may change three times over the programming period before settling to the final output frequency. The same principle applies to QMC gain/phase adjustment and QMC offset adjustment circuits. The double-buffered registers prevent unintentional gain, phase, and/or offset corrections during the programming of the register.

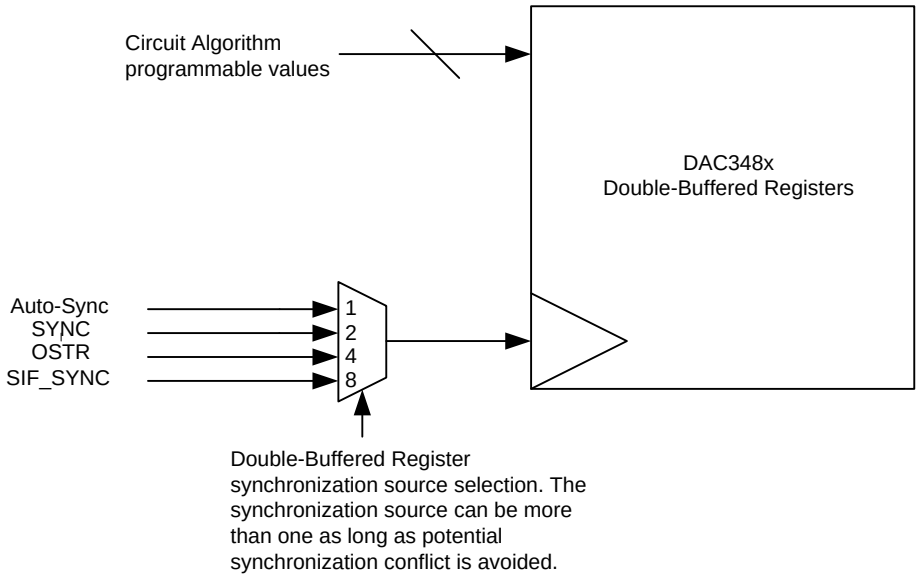


Figure 30. Double-Buffered Registers

For flexibility, the double-buffered register can have one or more synchronization sources, and the user should plan out the synchronization method. The sources can be SIF_SYNC, SYNC, OSTR, or auto-sync. For the synchronization using the SIF_SYNC, SYNC, and OSTR, the corresponding zero to one transition of the signal will create a synchronization event. For auto-sync selection, the synchronization will occur only when specific registers are written, and the register may not necessary be the last sequence of registers. Therefore, users will need to design their start-up sequence to write to auto-sync register specifically to synchronize the double buffered circuit. The table below shows the double buffered synchronization source select register for each digital block and the corresponding auto-sync register, and the detailed description of synchronization sources can be found on Table 4 and Table 5.

Digital Blocks	Double-Buffered Synchronization Source Selection	Auto-Sync Register
QMC Gain/Phase AB Channels	Config30	Config16
QMC Gain/Phase CD Channels	Config30	Config17
QMC Offset AB Channels	Config30	Config8
QMC Offset CD Channels	Config30	Config10
NCO Mixer AB Channels	Config31	Config18
NCO Mixer CD Channels	Config31	Config19

Table 4. DAC348x Double-Buffered Registers and the Associated Auto-Sync Registers

Synchronization Source Selection	Synchronization Source	Description
8	SIF_SYNC	0->1 Transition of the SIF_SYNC bit in Config31, bit2
4	SYNC	0->1 Transition on the LVDS SYNC. FIFO must be enabled
2	OSTR	0->1 Transition on the OSTR signal (either externally provided on the LVPECL OSTRp/n source or internally generat
1	Auto-Sync from Register Write	writing to the auto-sync register

Table 5. DAC348x Doube-Buffered Synchronization Sources Description

Programmable digital logics such as NCO and QMC circuits allow the system to change the DAC output values as the DAC is running. The output adjustments require the double-buffered synchronization sources as mentioned above, and these sources can be the same sources for the FIFO, data formatter, and the clock divider synchronization sources. Accidental reset to the FIFO, data formatter, and the clock divider may cause unexpected output behavior. Therefore, it is important for users to avoid this type of conflicts in system design.

For instance, when the LVDS SYNC or external LVPECL OSTR signal is used to synchronize the double-buffered registers, the occurrence of the zero to one signal transition must match the clock timing (i.e. DATACLK for SYNC and DACCLK for OSTR) and must not disrupt the FIFO flow. The design of the signal transition must occur at the beginning of the eight sample spaces.

If the programming of the double-buffered logics could affect the FIFO, data formatter, or clock divider operation, a good design practice is to disable FIFO, data formatter, and clock divider synchronization when the same synchronization sources are shared. After

the initial synchronization of the FIFO, data formatter, and clock divider, these circuits can be programmed to not pay attention to the synchronization sources.

Finally, the easiest way is to plan out the synchronization signal such that the synchronization sources for the double-buffered registers are different than the FIFO, the data formatter, and the clock divider. For instance, the auto-sync register write feature is a dedicate synchronization source for the double-buffered registers.

2.4 NCO Accumulator (adjusted during system operation)

The NCO accumulator is used to generate the desired sine and cosine term from a look-up table. After loading the frequency and phase information into the NCO double buffered registers, the NCO accumulator requires a reset to ensure that a correct, known state is initialized. The NCO block diagram is shown in Figure 31.

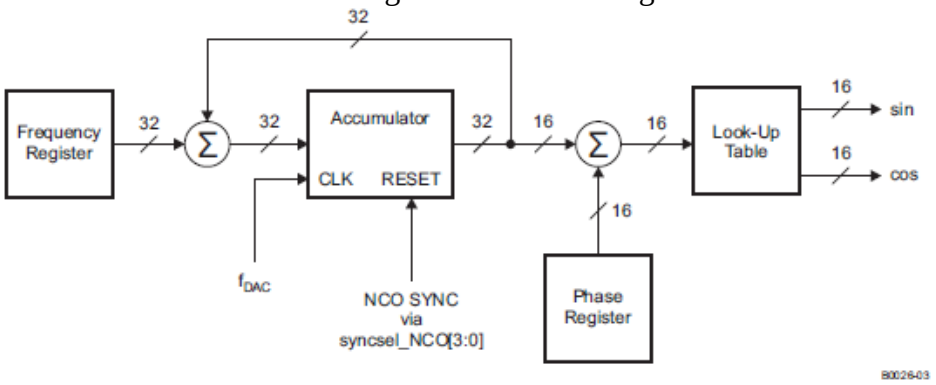


Figure 73. NCO Block Diagram

Figure 31. NCO Block Diagram (Figure 73, SLAS749)

The following table shows the synchronization sources for the NCO accumulator.

Synchronization Source Selection	Synchronization Source	Description
8	SIF_SYNC	0->1 Transition of the SIF_SYNC bit in Config31, bit2
4	SYNC	0->1 Transition on the LVDS SYNC. FIFO must be enabled
2	OSTR	0->1 Transition on the OSTR signal (either externally provided on the LVPECL OSTRp/n source or internally genera
1	ISTR	0->1 Transition on the LVDS SYNC. FIFO must be enabled

Table 6. Synchronization Sources for NCO Accumulator

Per register Config21 of the DAC348x, the sources can be SIF_SYNC, SYNC, OSTR, or ISTR. If the end application does not require specific timing or phase synchronization, the zero-to-one transition of the SIF_SYNC bit is the easiest to implement and sufficient for most of the general purpose NCO accumulator initialization. While the ISTR, SYNC, and/or OSTR can be used for accumulator reset, the primary intention of these synchronization methods is to ensure the NCOs of multiple DAC devices reset at the same instance of time to achieve output phase alignment.

As mentioned in section 1.5, the ISTR and/or SYNC are treated as data in the FIFO. If the ISTR and/or SYNC data are loaded to the devices of the same system in the correct sequence, then the signal data will be read at the same time instance. The signal data will reset the NCO accumulator at the same time instance, thus achieving synchronization.

When the OSTR option is selected, the synchronization signal is either the external OSTR signal or the internally generated OSTR signal from the on-chip PLL N-divider. As part of the multi-device synchronization requirement, the OSTR signals to the multiple devices must be time aligned. Therefore, the NCO accumulators of multiple devices will also be aligned.

Synchronizing the NCO accumulator may pose some issues due to the repeating nature of some synchronization sources and the looping structure of the NCO accumulator. When the synchronization signal is repetitive (i.e. ISTR and OSTR signals used in Dual Sync Sources mode), then the NCO frequency must be some integer multiple of the synchronization frequency. As shown in Figure 32, if the periodic synchronization signal does not allow the complete cycling of the NCO accumulator, the output of the NCO will be distorted.

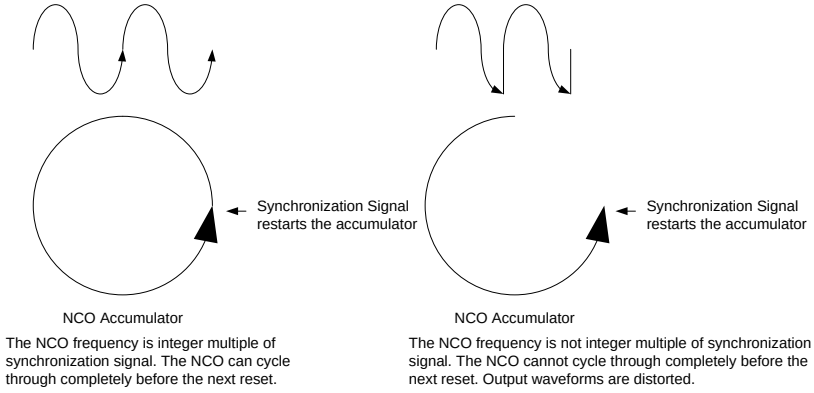


Figure 32. NCO Accumulator Operation with Repetitive Synchronization Signals

The requirements for the NCO accumulator synchronization signals:

- **Single Pulse Signal**
  There is no requirement for NCO frequency to be evenly divisible by F_{DAC} if a single pulse OSTR signal is used to reset the FIFO and NCO during initialization.
- **Periodic Signal**:
  The periodic synchronization signals are used in Dual Sync Sources mode where the ISTR (or SYNC) and OSTR are used to synchronized the FIFO write and read pointers. In this case, the synchronization signal frequency must meet both the FIFO block requirement and the NCO accumulator requirement.
  - FIFO: Refer to the DAC348x family device datasheet for FIFO synchronization requirements.
  - NCO accumulator: The synchronization signal frequency must be integer divisible by the NCO frequency. This allows the NCO counter to complete the cycle at least once without interruption.
  - To meet both requirements, the synchronization signal frequency must be the least common multiple (LCM) of both the FIFO block requirement and NCO accumulator requirement.
  - The example below shows the DAC34H84 NCO using the OSTR signal when the DAC34H84 is in Dual Sync Sources mode.

$$F_{OSTR\_FIFO} = \frac{F_{DAC}}{n \times interpolation \times 8} \text{ given } n \text{ is an integer}$$

$$F_{OSTR\_NCO\_SYNC} = \frac{F_{NCO}}{m} = \frac{F_{DAC}}{\frac{F_{DAC} \times m}{F_{NCO}}} \text{ given } m \text{ is an integer}$$

$$F_{OSTR\_FIFO\_NCO\_SYNC} = \frac{F_{DAC}}{LCM(n \times interpolation \times 8, \frac{F_{DAC} \times m}{F_{NCO}})} \text{ given } \frac{m \times F_{DAC}}{F_{NCO}} \text{ is an integer to find LCM.}$$

$$F_{OSTR\_FIFO\_NCO\_SYNC} = \frac{F_{DAC}}{n \times interpolation \times 8 \times \frac{F_{DAC} \times m}{F_{NCO}}}$$

$$= \frac{F_{NCO}}{m \times n \times interpolation \times 8}$$

Table 7 below summarizes the synchronization needed for the DAC348x device logics.

Circuits	FIFO	Data Formatter	Clock Divider	NCO Accumulator	NCO Double Buffered	QMC Gain/Phase
Sync Sources	ISTR/FRAME	ISTR/FRAME	ISTR/FRAME	SIF_SYNC	SIF_SYNC	SIF_SYNC
	SYNC	SYNC	SYNC	SYNC	SYNC	SYNC
	SIF_SYNC		SIF SYNC	OSTR	OSTR	OSTR
	OSTR		OSTR	ISTR	Auto-sync from register write	Auto-sync from register write

Table 7. DAC348x Device Logics Synchronization Sources

3. Start-up Sequence

The section goes over the recommended start-up sequence for the DAC348x family. The important steps are to synchronize the FIFO, clock divider, and the data formatter (if needed) as the first step. The programmable logics such as QMC and NCO can be synchronized afterwards as the DAC is running. Designers must be careful to avoid accidental synchronizations to unintended logics in order to prevent unexpected output behavior. An example start-up sequence is listed at the end of this application note.

1. Ensure the TXENABLE or TXENA pin are low. This disables any potential data transmission to the output.
2. Supply voltages to DACVDD, DIGVDD, CLKVDD, VFUSE, AVDD, IOVDD, and PLLAVDD. These supplies can be powered up simultaneously or in any order. There are no specific requirements on the ramp rate for the supplies.
3. Provide all LVPECL inputs: DACCLKP/N and the optional OSTRP/N. These inputs can also be provided after the SIF register programming.
4. Toggle the RESETB pin for a minimum 25 ns active low pulse width.
5. Program the SIF registers.
6. FIFO configuration needed for synchronization:
    a. Program *syncsel_fifoin(3:0)* (config32, bit<15:12>) to select the FIFO input pointer sync source.

b. Program *syncsel_fifoout(3:0)* (config32, bit<11:8>) to select the FIFO output pointer sync source.

c. Program syncsel_dataformatter(1:0) (config31, bit<3:2>) to select the FIFO Data Formatter sync source.

7. Clock divider configuration needed for synchronization:

   a. Program *clkdiv_sync_sel* (config32, bit<0>) to select the clock divider sync source.

   b. Program *clkdiv_sync_ena* (config0, bit<2>) to "1" to enable clock divider sync.

8. Provide all LVDS inputs (DAB[15:0]P/N, DCD[15:0]P/N, DATACLKP/N, ISTRP/N, SYNCP/N and PARITYP/N) simultaneously. Synchronize the FIFO and clock divider by providing the pulse or periodic signals needed. For example, provide a pulse on the ISTRP/N LVDS pair and the pulse on the OSTRP/N LVPECL pair in Dual Sync Source Mode.

9. FIFO and clock divider configurations after all the sync signals have provided the initial sync pulses needed for synchronization:

   a. The clock divider operates in the DACCLK domain and provides the divided-down clocks for the digital circuits inside the DAC. Therefore, for Single Sync Source Mode where the clock divider sync source is either ISTRP/N or SYNCP/N, clock divider syncing must be disabled after DAC34H84 initialization and before the data transmission by setting *clkdiv_sync_ena* (config0, bit 2) to "0". Enabling the clock divider syncing at all time for Single Sync Source Mode is not recommended due to the possible phase ambiguity between the DATACLK and DACCLK clock domains

   b. For Dual Sync Source Mode, where the clock divider sync source is from the OSTR signal (either from external OSTRP/N or internal PLL N divider output), the clock divider syncing may be enabled at all time.

   c. Optionally, disable FIFO syncing by setting *syncsel_fifoin(4:0)* and *syncsel_fifoout(4:0)* to "0000" after the FIFO input and output pointers are initialized.

10. Enable transmit of data by asserting the TXENABLE/TXENA pin or set *sif_txenable* to "1".

EXAMPLE START-UP ROUTINE

Device Configuration:

f_{DATA} = 737.28MSPS

Interpolation = 2x

Input data = baseband data

f_{OUT} = 122.88MHz

PLL = Enabled

Full Mixer = Enabled

NCO = Enabled

Dual Sync Sources Mode

PLL Configuration:

f_{REFCLK} = 737.28MHz at the DACCLKP/N LVPECL pins

$f_{DACCLK} = f_{DATA}$ x Interpolation = 1474.56MHz

$f_{VCO} = 2 \times f_{DACCLK}$ = 2949.12MHz (keep f_{VCO} between 2.7GHz to 3.3GHz)

PFD = f_{OSTR} = 46.08MHz

N = 16, M = 32, P = 2, single charge pump

PLL_VCO(5:0) = "011100" (28)

NCO Configuration:

f_{NCO} = 122.88MHz

f_{NCO_CLK} = 1474.56MHz

freq = f_{NCO} x 2^32 / 1228.8
= 357913941
= 0x15555555

phaseaddAB(31:0) and/or phaseaddCD(31:0) = 0x15555555

NCO synchronization method = rising edge of LVDS SYNC

Step	Read/Write	Address	Value	Description
1	N/A	N/A	N/A	Set TXENA Low
2	N/A	N/A	N/A	Power Up the device
3	N/A	N/A	N/A	Apply LVPECL DACCLKP/N for PLL reference clock
4	N/A	N/A	N/A	Toggle RESETB pin
5	Write	0x00	0xF19F	QMC offset and correction enabled, 2x int, FIFO enabled, Alarm enabled, clock divider sync enabled, inverse sinc filter enabled.
6	Write	0x01	0x040E	Single parity enabled, FIFO alarms enabled (2 away, 1 away, and collision).
7	Write	0x02	0x7052	Output shut-off when DACCLK gone,

				DATACLK gone, and FIFO collision. Mixer block with NCO enabled, twos complement.
8	Write	0x03	0xA000	Output current set to 20mAFS with internal reference and 1.28kohm R_{BIAS} resistor.
9	Write	0x07	0xD8FF	Un-mask FIFO collision, DACCLK-gone, and DATACLK-gone alarms to the Alarm output.
10	Write	0x08	N/A	Program the desired channel A QMC offset value. **(Causes Auto-Sync for QMC AB-Channels Offset Block)**
11	Write	0x09	N/A	Program the desired FIFO offset value and channel B QMC offset value.
12	Write	0x0A	N/A	Program the desired channel C QMC offset value. **(Causes Auto-Sync for QMC CD-Channels Offset Block)**
13	Write	0x0B	N/A	Program the desired channel D QMC offset value.
14	Write	0x0C	N/A	Program the desired channel A QMC gain value.
15	Write	0x0D	N/A	Coarse mixer mode not used. Program the desired channel B QMC gain value.
16	Write	0x0E	N/A	Program the desired channel B QMC gain value.
17	Write	0x0F	N/A	Program the desired channel C QMC gain value.
18	Write	0x10	N/A	Program the desired channel AB QMC phase value. **(Causes Auto-Sync QMC AB-Channels Correction Block)**
19	Write	0x11	N/A	Program the desired channel CD QMC phase value. **(Causes Auto-Sync for the QMC CD-Channels Correction Block)**
20	Write	0x12	N/A	Program the desired channel AB NCO phase offset value. **(Causes Auto-Sync for Channel AB NCO Mixer)**
21	Write	0x13	N/A	Program the desired channel CD NCO phase offset value. **(Causes Auto-Sync for Channel CD NCO Mixer)**
22	Write	0x14	0x5555	Program the desired channel AB NCO frequency value
23	Write	0x15	0x1555	Program the desired channel AB NCO frequency value
24	Write	0x16	0x5555	Program the desired channel CD NCO frequency value
25	Write	0x17	0x1555	Program the desired channel CD NCO frequency value
26	Write	0x18	0x2C50	PLL enabled, PLL N-dividers sync enabled, single charge pump, prescaler = 2.

27	Write	0x19	0x20F4	M = 32, N = 16, PLL VCO bias tune = "01"
28	Write	0x1A	0x7010	PLL VCO coarse tune = 28
29	Write	0x1B	0x0800	Internal reference
30	Write	0x1E	0x9999	QMC offset AB, QMC offset CD, QMC correction AB, and QMC correction CD can be synced by sif_sync or auto-sync from register write
31	Write	0x1F	0x4440	Mixer AB and CD values synced by SYNCP/N. NCO accumulator synced by SYNCP/N. FIFO data formatter synced by ISTRP/N.
32	Write	0x20	0x2400	FIFO Input Pointer Sync Source = ISTR FIFO Output Pointer Sync Source = OSTR (from PLL N-divider output) Clock Divider Sync Source = OSTR
33	N/A	N/A	N/A	Provide all the LVDS DATA and DATACLK Provide rising edge FRAMEP/N and rising edge SYNCP/N to sync the FIFO input pointer and PLL N-dividers.
34	Read	0x18	N/A	Read back pll_lfvolt(2:0). If the value is not optimal, adjust pll_vco(5:0) in 0x1A.
35	Write	0x05	0x0000	Clear all alarms in 0x05.
36	Read	0x05	N/A	Read back all alarms in 0x05. Check for PLL lock, FIFO collision, DACCLK-gone, DATACLK-gone, etc. Fix the error appropriately. Repeat step 34 and 35 as necessary.
37	Write	0x1F	0x4442	Sync all the QMC blocks using sif_sync. These blocks can also be synced via auto-sync through appropriate register writes.
38	Write	0x00	0xF19B	Disable clock divider sync.
39	Write	0x1F	0x4448	Disable FIFO data formatter sync. Set sif_sync to "0" for the next sif_sync event.
40	Write	0x20	0x0000	Disable FIFO input and output pointer sync.
41	Write	0x18	0x2450	Disable PLL N-dividers sync.
42	N/A	N/A	N/A	Set TXENA high. Enable data transmission.