Technical Note

High-Speed DRAM Controller Design

Introduction

Multiple ways to design DRAM controllers exist, each having its own advantages and disadvantages. The intent of this technical note is to identify and discuss five key areas of DRAM controller design, including controller requirements, controller clock domains, transmit circuits, capture circuits, strobe detection, and strobe delay circuits. As bandwidth requirements have increased for high-speed DRAM memory systems, such as high-speed networking, so, too, have the design challenges increased. For example, with DDR3 data rates pushing 1.6 GHz, designing the logic interface has become particularly challenging. And to deal with smaller data valid windows, many areas in the transmit and receive circuits need to be enhanced.

Controller Requirements

At the start of every DRAM controller design, it's important to balance the requirements of the system with its complexity (cost). To determine the controller CORE_CLK architecture, look at the command scheduling, power, and speed requirements. (Note that in the following section, the CORE_CLK refers to the base clock of the memory controller.) Two examples of a controller CORE_CLK architecture include a 1-to-1 DRAM to CORE_CLK ratio and a 2-to-1 DRAM to CORE_CLK ratio. Although many other possibilities exist, these are the two scenarios that we have seen most in this industry.

Employing the 1-to-1 ratio offers a command scheduling advantage with the increased controller clock frequency, but, depending on the speed needed versus the available process geometry, employing this ratio might not be possible. Moving to a 2-to-1 ratio reduces the CORE_CLK speed in relation to the DRAM clock speed, but this may reduce the command bandwidth. You can get around this by scheduling commands through a 2-to-1 serializer, so both commands can be sent at the same time. For example, you could schedule the RAS and CAS commands simultaneously. This type of approach also has a power advantage for the controller because of the reduced clock frequency.

Defining the interface width comes down to whether you want the complexity on the board design or in the controller. The overall data rate can easily be increased by increasing the overall width of the memory system. In some cases, this enables you to reduce the speed of the DRAM, but at the cost of increased pin count on the controller and additional routing headaches. If the overall DRAM speed need is beyond the current technology found in DRAM, you will be forced to go wider.

The other option to carefully examine is error correcting bits (such as x36, x72). From the DRAM point of view, with every process shrink, the soft error rate is coming down to a level that it makes sense to look carefully into whether you really need ECC. For more information on DRAM soft error rates, contact Micron.

One of the key design requirements in any DRAM system is determining the speed required and the corresponding DRAM technology that is needed in the application. As the speed requirements increase, the complexity of the DRAM controller also increases. At the higher clock rates of DDR3, a 1/4 clock might be required, depending on the process geometry being used. To deal with the extended burst length, the internal width of the data path may have to be increased. Basically, you are capturing data serially and then converting the data to parallel. Other complexities that are introduced include the need to have calibration or training of the I/O interface. As process geometries are reduced, training also might be needed at the slower speeds because of the increased variability found in the finer resolution geometries.

The number of address and command copies required is typically a trade-off between signal integrity and pin count. As the loading on the address and command pins continues to increase, a need for 2T style addressing may exist. 2T style addressing is done by lightly loading the CS# pin to the DRAM while the address and command pins are heavily loaded. The address and command are sent over 2 clock cycles with the CS# being used to gate the clock to the DRAM, as shown in Figure 1. In some cases, it's a good idea to have a mode in the chip where this can be turned on or off, depending on how fast the application is running, whether the controller is being designed for multiple applications, and what its loading conditions are.

If you look at some of the DDR3 controllers that are designed for high speed and are heavy loaded, some 3T addressing is happening. Increasing the load and reducing the signal rise time not only creates timing problems, but it also causes signal integrity problems. In the end, when the controller is being defined and the requirements are being set, signal integrity and timing need to be examined closely to ensure the right trade-offs are being made.

Figure 1: 2T Addressing Example

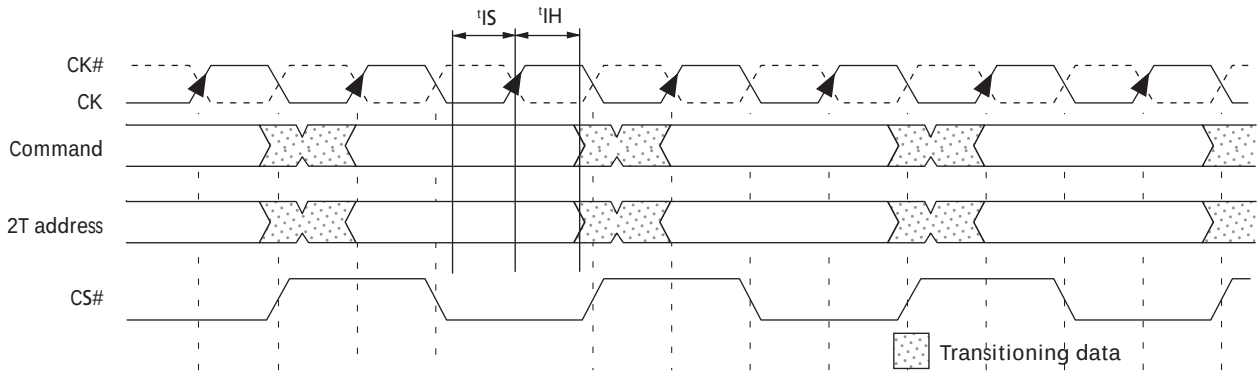
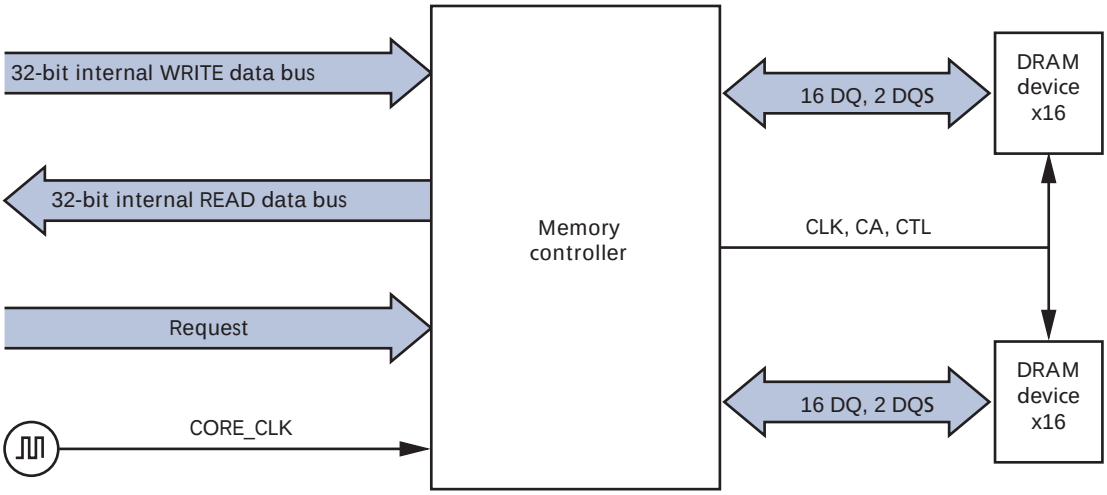


Figure 2 on page 3 illustrates a block diagram of a point-to-point DRAM system. The areas in the block diagram are the typical points that need to be defined early in the design process.

Figure 2: Controller Overview Example



DRAM Controller Clock Domains

Controller clock domains can be designed in many different ways. The one key area to consider here is the CORE_CLK phase alignment to the DRAM CLK. The CLK trees options are a function of what is available from the DLL/PLL in the physical interface. To look at an example, a 1x CORE clock can be selected and used to transmit address, command, and control signals to the DRAM. Adding a 2x phase-aligned clock from the DLL/PLL would typically be used to transmit the DRAM CLK and DQS signals. The 90 degree out-of-phase clock can then be used to transmit the DQ and DM signals. This type of approach, illustrated in Figure 3, is a typical approach and seems to work quite well because it can be more accurate to clock internal signals in the controller on rising edges (for example, because of less duty cycle dependency). In most cases, holding the duty cycle constant seems to be a bit more challenging than running 2 CLK trees out of phase.

Figure 3: Controller Clocks

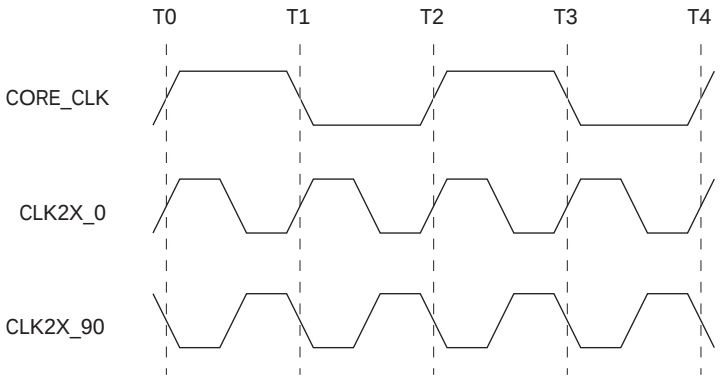
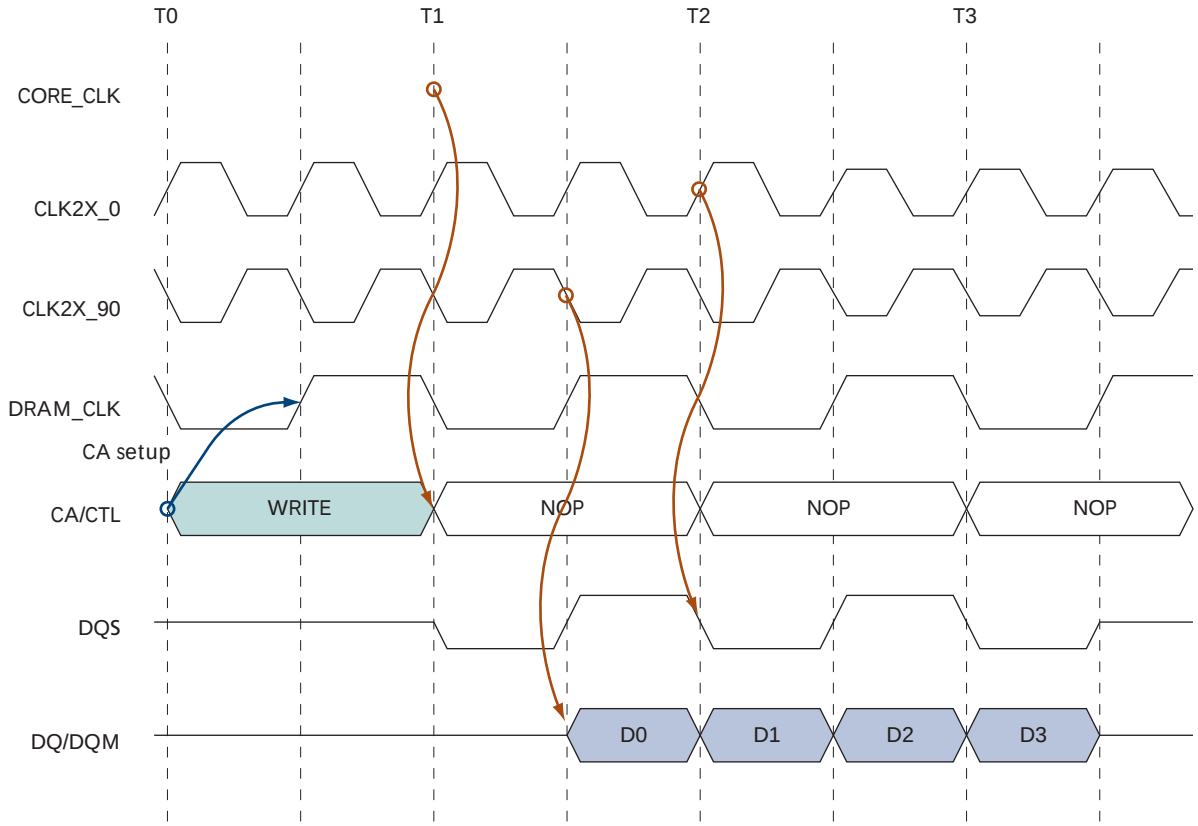


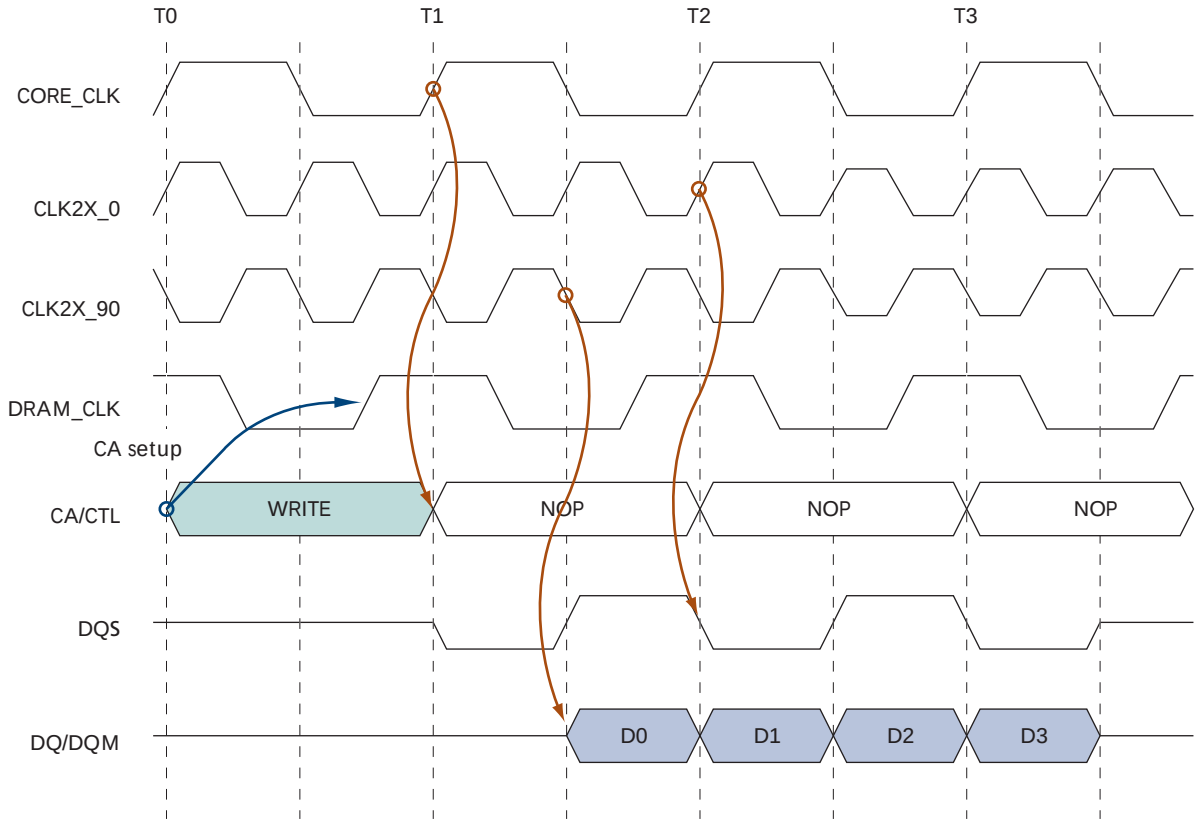
Figure 4 on page 4 shows the aliment of the CORE_CLK with the DRAM clock. Each clock is shown with the related reference point to the DRAM signals. The phase between the CORE_CLK and the DRAM clock can be changed if there is a delay line or a DLL that can be used to change this relationship.

Figure 4: Controller Clocks With DRAM I/F



In some cases, some command and address busses require additional setup at the higher speed. The relation between the CORE_CK and the 2x CLKs should stay the same to transmit the higher speed DQ and DQS signals. The DRAM_CLK can then be shifted to gain margin to setup on the address and command bus. Shifting the CORE_CLK to DRAM_CLK relationship adds margin to the address and command bus, which can help with loading conditions. The 2x CLKs are still used for the DQ and DQS signals. To make things a bit more flexible, you can add the capability to change this phase offset either at a boot-up or after training, but only if necessary.

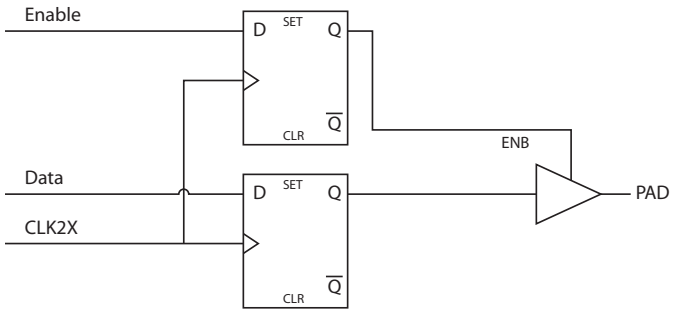
Figure 5: Added (3/4 CK) CA Setup



Transmit Circuits

You can design transmit circuits in various ways and with varying complexities. And, as with other elements of controller design, as speeds increase, the complexity of transmit circuit design also increases. The simplest approach to the transmit circuit is a single phase Tx, which uses two flip flops with a 2x input clock. See Figure 6. The 2x input clock enables the use of rising edges only, which makes this approach duty cycle independent. The symmetry in the flip flops is critical to the timing accuracy of the driver.

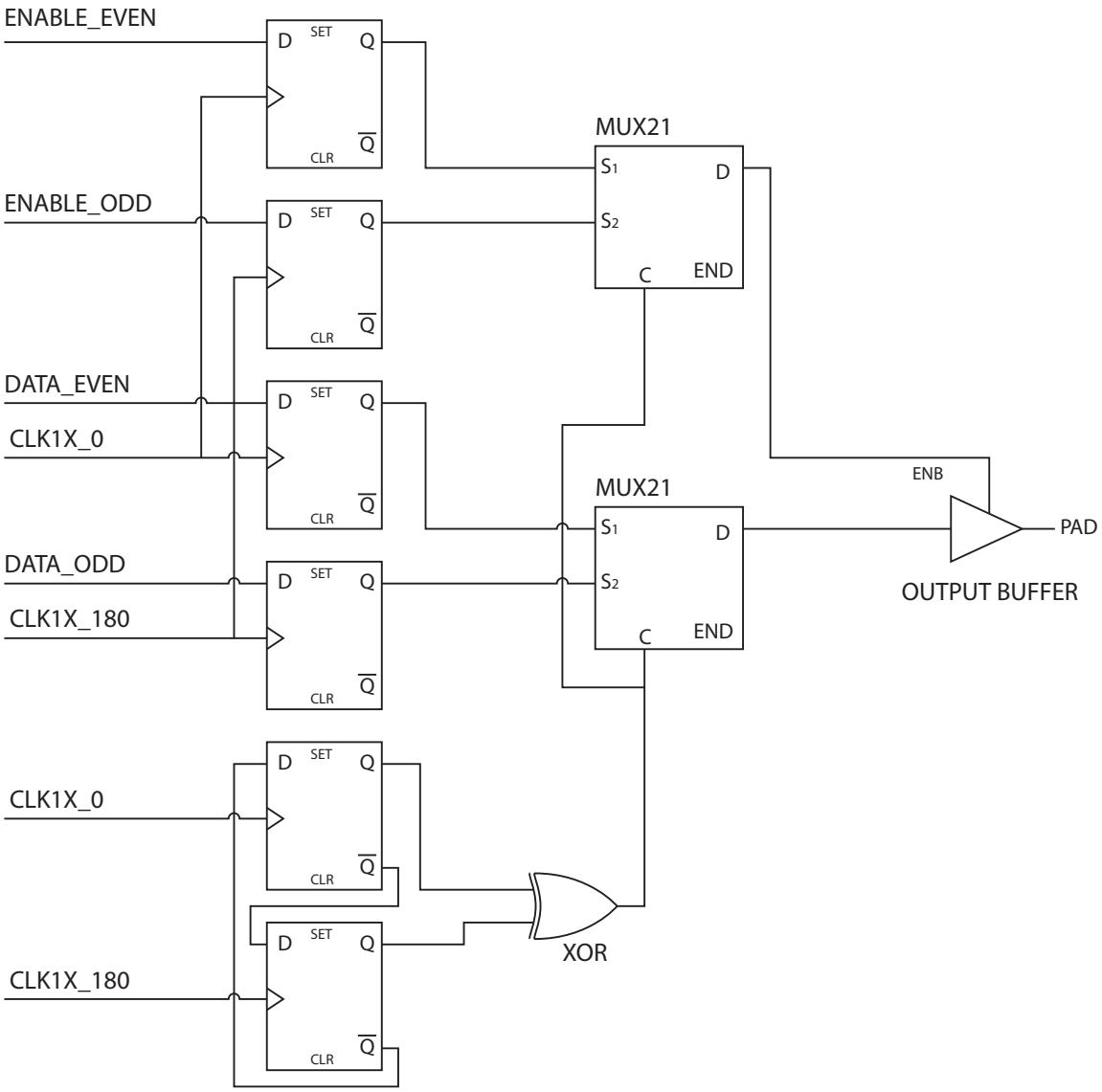
Figure 6: Single Phase Tx



If the speed is pushing the limits of the process technology and a 2x clock is no longer possible, you might need another approach. A slightly more complex approach to designing transmit circuits would be to use a 2-phase Johnson counter, as shown in Figure 7 on page 7. A Johnson counter is basically a shift register, where the complement of output from the last stage is used as an input to the first stage.

This approach uses multiple phases of the 1x clock, where the 1x clock is equal to the CORE_CLK. The Johnson counter keeps the circuit from being duty cycle dependent. Even and odd data use 0 and 180 degree 1x clock, respectively. The 0 and 180 clocks also feed and enable the counter and enable flip flops in the circuit. The outputs of the enable and data flip flops are then fed into a MUX for parallel to serial conversion. The data select to the MUX is triggered by the counter through an XOR gate. It is critical in this type of approach to keep the symmetry/delay through the counter, XOR, and MUX matched. Although Figure 7 shows a 2-phase Johnson counter, this could easily be expanded to 4 phases if needed. If this type of approach is needed, static timing analysis and SPICE simulation are required.

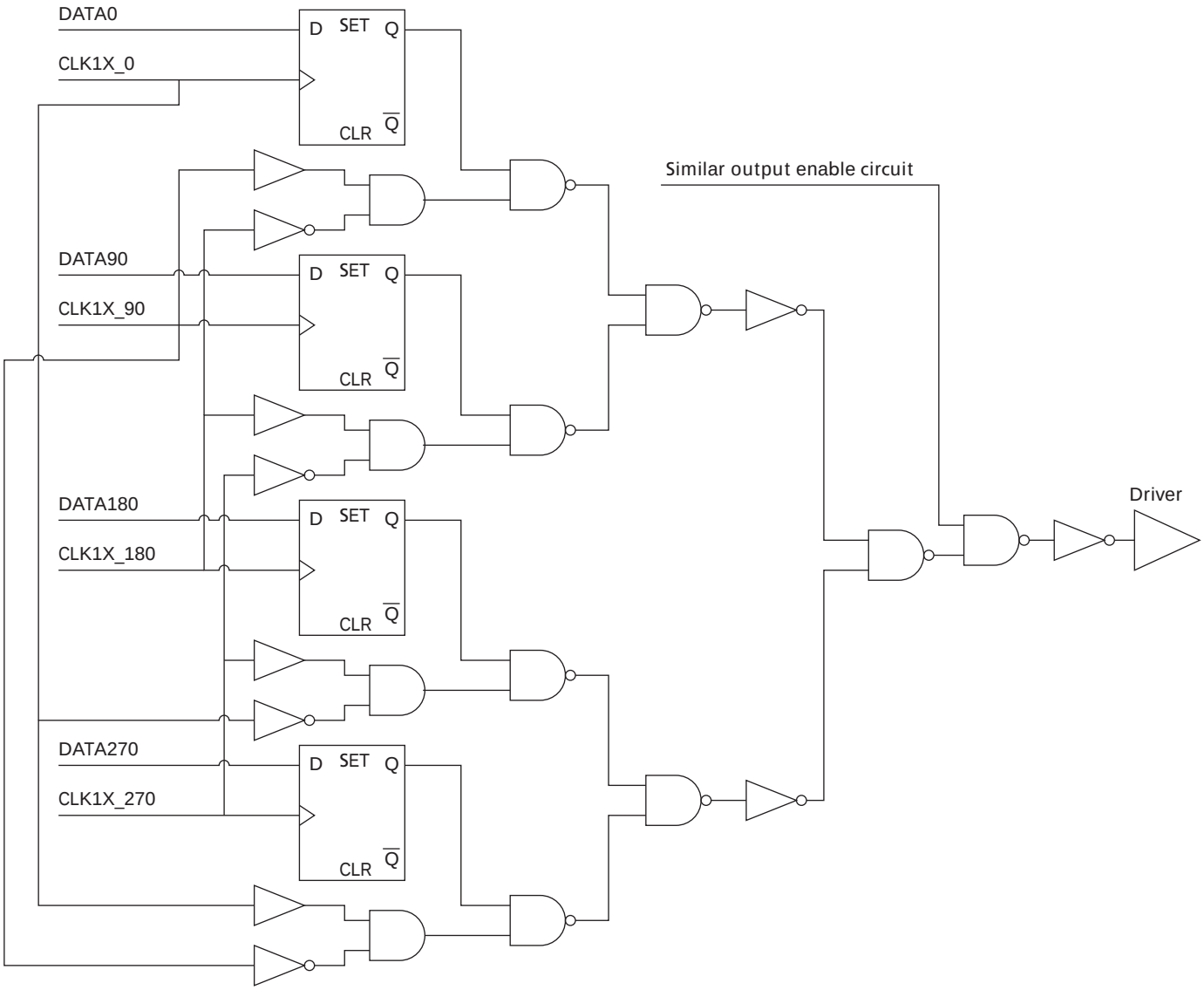
Figure 7: 2-Phase Johnson Counter



The last approach we will be discussing on the transmit circuit side is the 4-phase gated clock Tx, shown in Figure 8 on page 8. This is a good approach to use when the speed of the DRAM is pushing the process capability of the controller. Similar to the Johnson counter approach, the gated clock Tx does not require a 2x or 4x clock, but does require multiple phases of the 1x CORE_CLK. For the CLK/DQS signals, this approach uses 4 phases of the CORE_CLK, including 0, 90, 180, and 270. The DQ pins and DM pins also use 4 phases, but shifted by 45 degrees to center-align the data with DQS.

The 4-phase gated Tx is also duty cycle variation tolerant. The output enable circuit is similar to the one shown in the Johnson counter (see Figure 7 on page 7). The 4 flip flops toggle data out sequentially with the different phases of the clock, aligning data at the output buffer. With this approach, it is critical everything is balanced to minimize any offset in the clock tree and to avoid race conditions and glitches. The matching should be manageable because, in terms of layout, everything is in the same area. This approach requires full SPICE simulation to make sure there are no glitches.

Figure 8: 4-Phase Gated Clock Tx



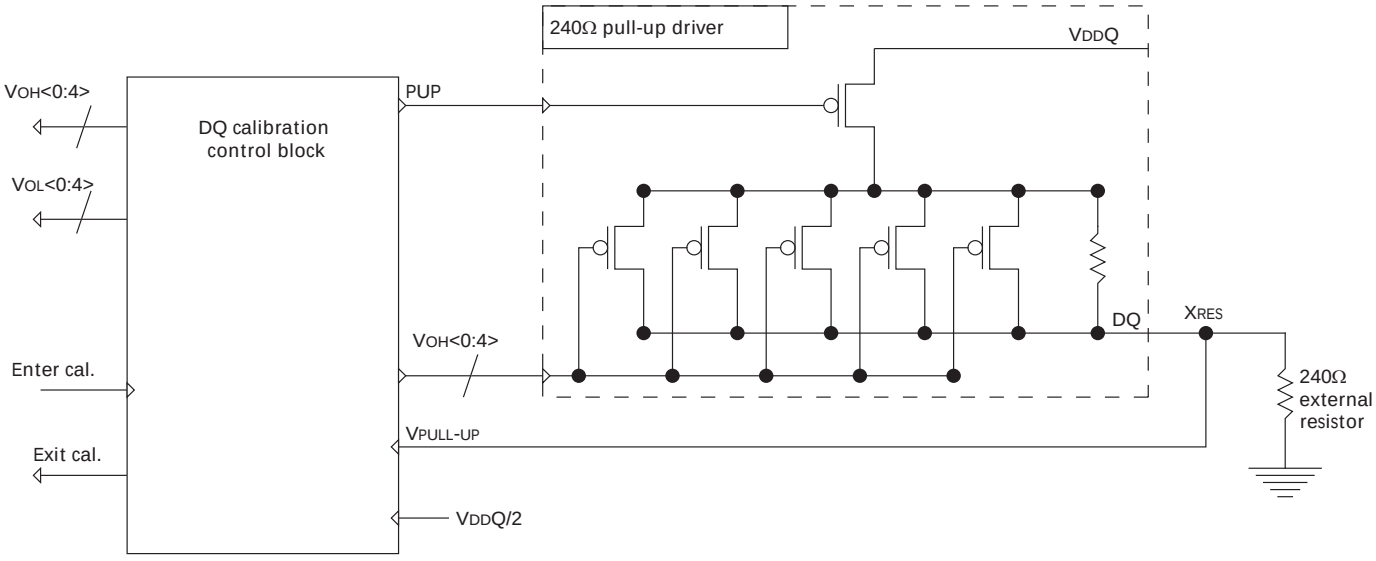
Calibration Circuit

After the DRAM bus speed reaches 533 Mb/s, calibrated output drivers greatly simplify the signaling problems that can occur at the system level design. A calibrated driver and on-die termination (ODT) require an external 1% precision resistor tied to Vss to calibrate the driver/ODT to a known value and to eliminate process variation that can be introduced during manufacturing. The calibration process requires the external resistor to be a multiple of the driver/ODT desired impedance.

As an example, if you need a 40Ω drive impedance with a 60Ω terminator, a 240Ω legs works perfectly so the external resistor would be 240Ω. The calibration process uses the programmable impedance control (PIC) circuit to calibrate both the output impedance and the ODT impedance. The PIC circuit contains one 240Ω pull-up leg, one 240Ω pull-down leg, and the DQ calibration control block, as shown in Figure 9 on page 9 and Figure 10 on page 10.

The DQ calibration block consists of an analog-to-digital converter (ADC), comparators, a majority filter, an internal reference voltage generator, and an approximation register. The 240Ω N-channel legs are identical to one of the several pull-down legs in the output driver, and the 240Ω P-channel leg matches one of the multiple pull-up legs. The pull-up legs can use a polyresistor that is slightly larger than the desired 240Ω value and has several P-channel devices to reduce the resistance of the leg and to tune the polyresistor to 240Ω. The large polyresistor is used to achieve a more linear pull-up and pull-down impedance for improved signal integrity at the system level. The pull-down leg is very similar to the pull-up leg, except using a large polyresistor with multiple N-channel devices for tuning the polyresistor to the desired 240Ω value.

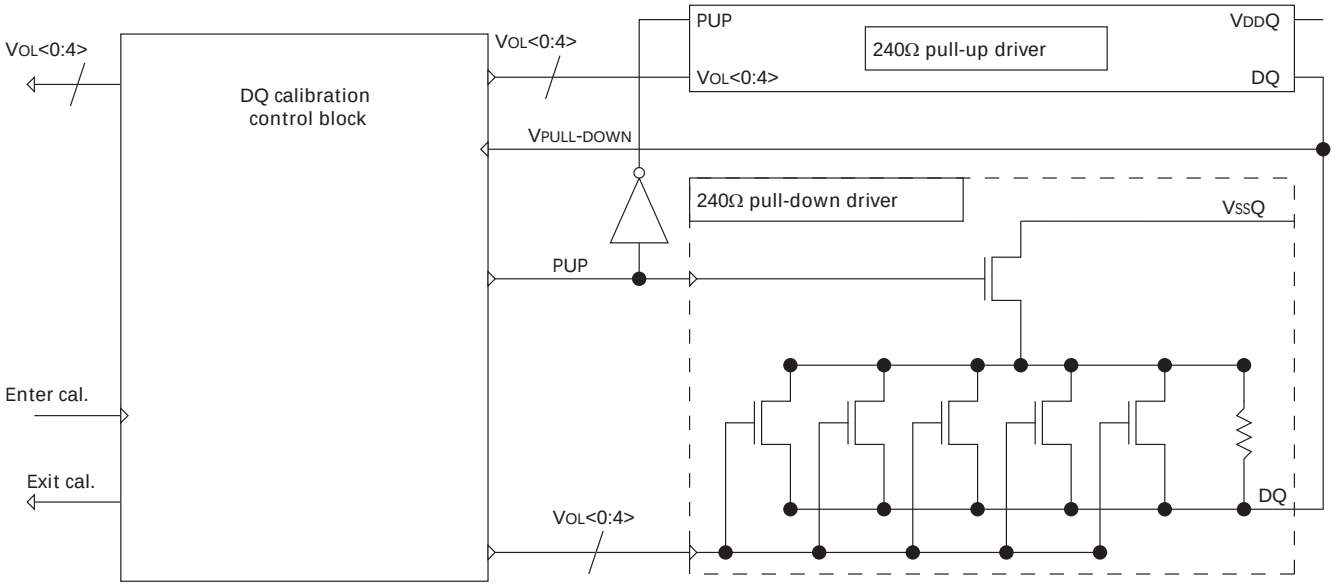
Figure 9: 240Ω Pull-Up Calibration Block



The calibration process starts by pulling the PUP line LOW (Figure 9), which pulls the pull-up leg to VDDQ. The VPULL-UP line is used to compare the voltage at the XRES point in Figure 9 to an internally generated reference voltage, VDDQ/2, with the comparator located inside the DQ calibration control block. The P-channel tuning devices are then individually turned on using the VOH signals until the voltage at XRES is equal to VDDQ/2. The VOH codes (number of devices needed to calibrate to 240Ω) generated by the PIC are stored in the internal approximation register and sent to each of the pull-up legs of the output driver at the appropriate times.

The VOH codes generated in the first stage of calibration are sent from the approximation register in the PIC to one 240Ω pull-up leg to calibrate the 240Ω pull-down leg. The PUP signal from the PIC turns on both the pull-down leg and the one pull-up leg, and an inverter is used to turn on the pull-up leg (see Figure 10 on page 10). The comparator is used again to compare the voltage on the VPULL-DOWN line (see Figure 10) to the reference voltage set at VDDQ/2, generating the VOL code, which is stored in an approximation register inside the PIC and sent to each of the output driver's pull-down legs at the appropriate time. After the VOH and VOL codes are obtained and the values are stored in the approximation register circuit inside the PIC, the calibration process is complete.

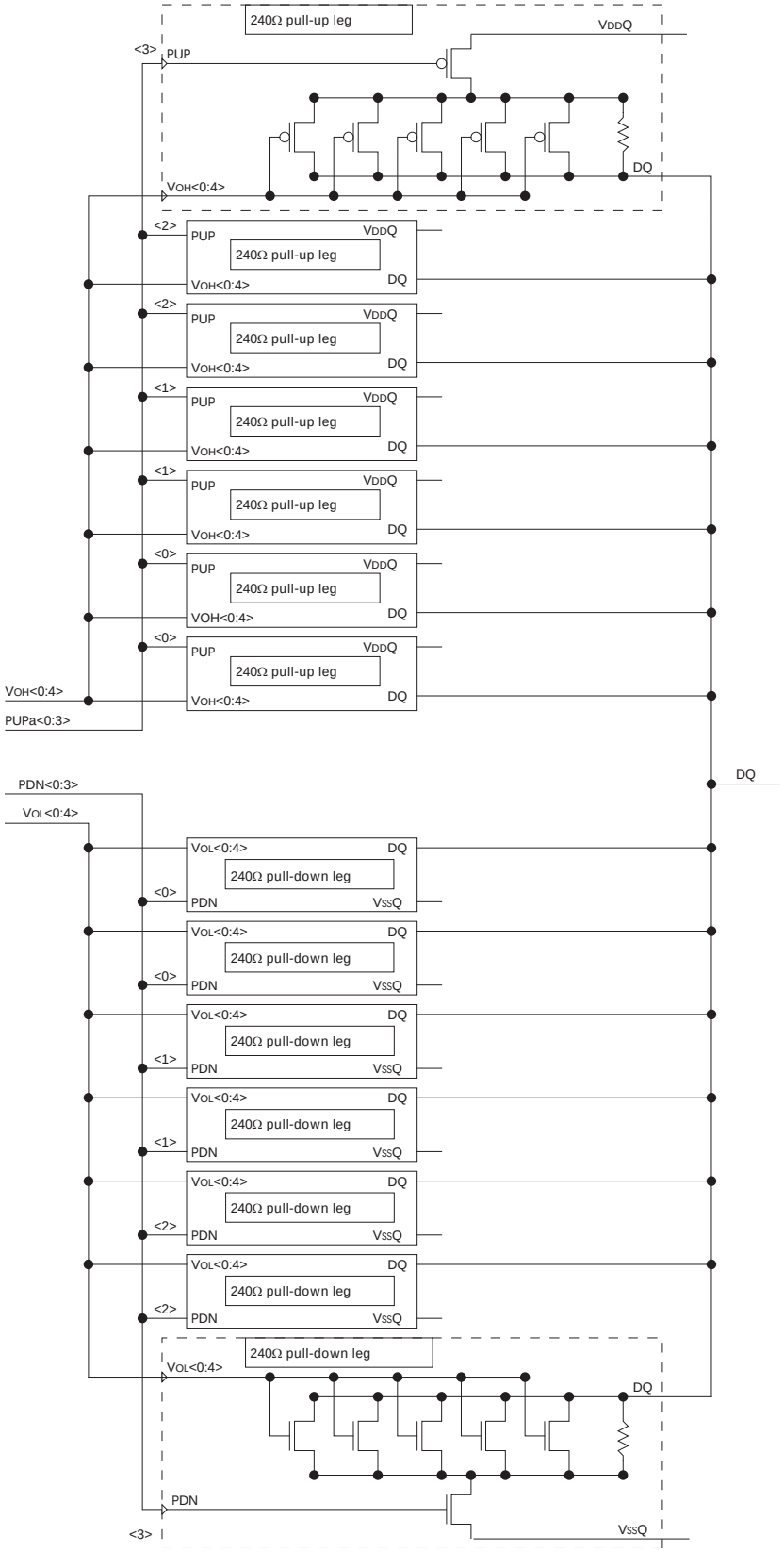
Figure 10: 240Ω Pull-Down Calibration Block



Output Driver/ODT

The output driver in this example is a standard push-pull stile driver, similar to what is found in some DRAM drivers seen today. It includes multiple legs used to tune the drive impedance of the byte lane [data (DQ) pins, data strobe (DQS) pins, data mask (DM), and ODT]. Figure 11 on page 11 illustrates the driver that is used in some DRAM memory systems. The pull-up and pull-down of the output driver consists of seven 240Ω legs that are identical to one of the 240Ω pull-up or pull-down legs in the PIC circuit. The seven legs in parallel make up a 34Ω pull-up driver. Two legs of the pull-up driver and two legs of the pull-down driver can then be used for a 60Ω ODT. The number of legs in the driver that are turned on in parallel will determine the drive strength or ODT impedance. Each stage of the pull-up driver is programmed with the VOH codes generated by the comparator in the PIC circuit to achieve a calibrated resistance on (RON) impedance value of the output driver and a calibrated ODT value.

Figure 11: Driver and ODT



Each of the seven stages of the pull-down driver is programmed using the VOL codes generated by the PIC circuit, which are similar to the codes sent to the pull-up legs. The seven parallel 240Ω legs are also used in parallel for the 34Ω pull-down driver. After the process variations have been eliminated by the PIC circuit, the only other factors that need to be addressed are voltage and temperature variation over time that are always present in the system. To address this, the drivers/ODT can be periodically recalibrated to minimize the fluctuations in the pull-up and pull-down drive currents caused by voltage and temperature variation. The PIC could be set to recalibrate any time a system reset, delay lock loop (DLL) reset, or a large phase shift occurs in the DLL phase circuit. Following one of these events, the PIC circuit starts the recalibration process. When the bus is quiet, the VOH and VOL codes from the approximation register in the PIC are sent to the driver and ODT, enabling extremely fast updates at any time.

Capture Circuits

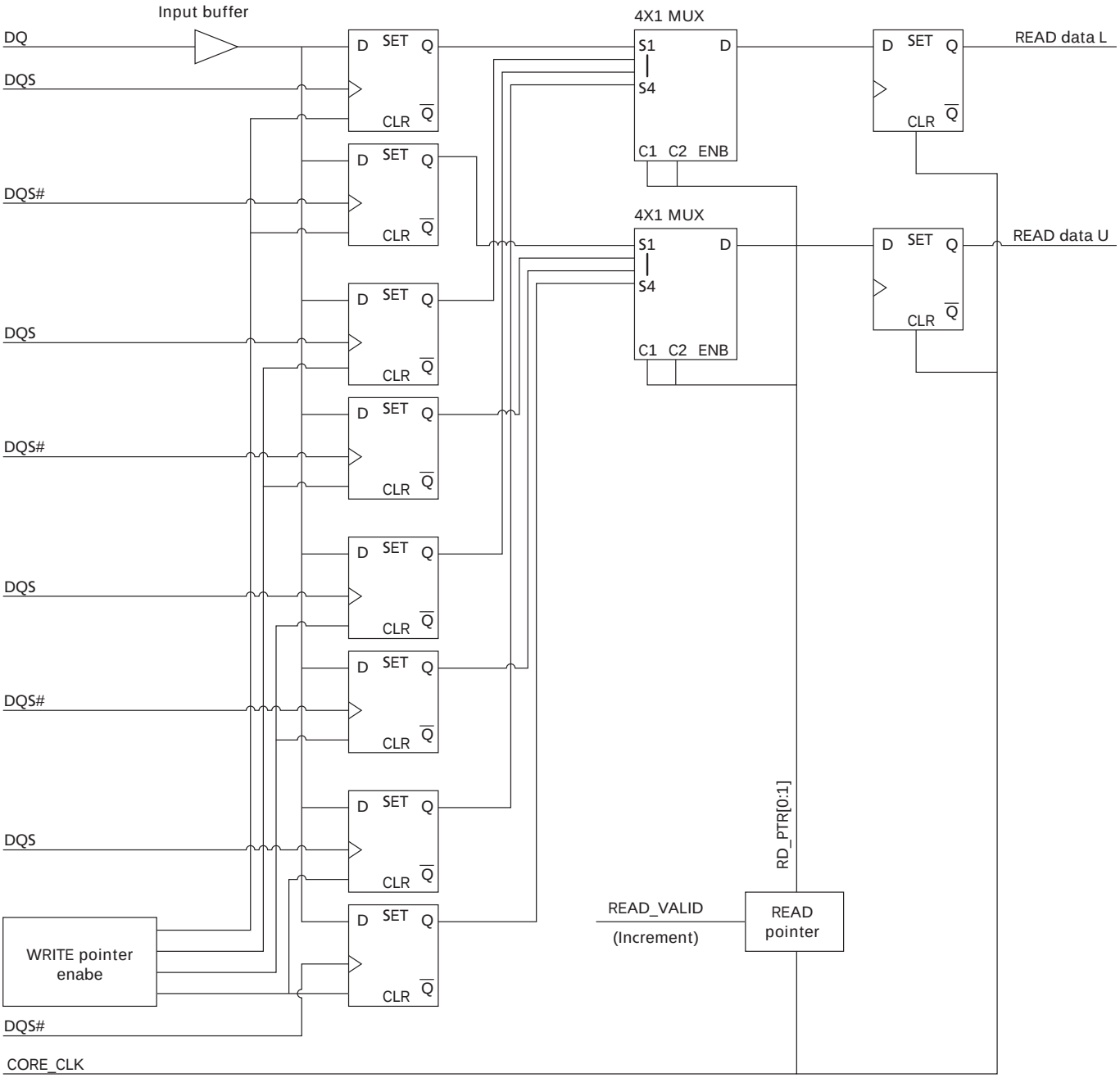
There are a couple of ways to address the capture circuit using an internal clock or strobe-based data capture. Both techniques have advantages and disadvantages. Internal clock, commonly called “free running,” uses an internally generated reference clock to capture data and ignores the DQS signal being sent by the DRAM. This type of approach definitely simplifies the DQS domain to CORE_CLK domain crossing. This approach can also save pin count.

Although these reasons are compelling, the downside is quite significant. This method reduces the size of the data valid window by as much as 25 percent. The voltage and temperature variation also becomes a real problem. Generally speaking, in DRAM development and design, it is very difficult to match output DQ signals to input clock. This variation tends to move quite a bit, which was the main reason the DRAM architecture went to a strobe-based data capture. On the DRAM side, the DQS signal is no different than a DQ, so as the DQ pins move cycle to cycle, the DQS signal will also move in the same fashion. For high-speed applications, using the internal clock to capture data is not recommended.

Using the DQS signal to latch data eliminates the temperature and voltage variation over time along with the natural movement introduced by the DRAM. The overall data valid window is increased, and as the speeds continue to increase, every picosecond counts. After the CK to DQS relationship from the DRAM reaches 1/2 the bit time, strobe-based data capture is the only option. Outside of the increased data valid window, the DQS data capture scheme also reduces the complexity of routing on the PCB because only the byte lane needs to be matched routed with this style of capture logic.

When using the strobe-based data capture scheme, it introduces the problem of crossing clock domains in the controller. A good way to handle this is with a circular FIFO structure (also called a ring buffer). See Figure 12 on page 13. There are two flip flops per entry that deal with the data on rising and falling data elements. The WRITE pointer increments with the DQS signal to fill the flip flops with incoming data from the DRAM. Moving the data from the FIFO is done with the READ pointer, which is incremented with the CORE_CLK. If more margin is needed between the CORE_CLK and the DQS domain, stages can be added to the FIFO. By adding stages to the FIFO, the handoff from the CORE_CLK domain can be delayed until you know that all the data from the burst resides in the FIFO. This eliminates ^{t}DQSCK from the timing budget. The minimum depth of the FIFO for arbitrary phase alignment is two times the DRAM-to-CORE_CLK ratio. The recommended depth is three or four times the DRAM-to-CORE_CLK ratio for added margin.

Figure 12: Circular FIFO

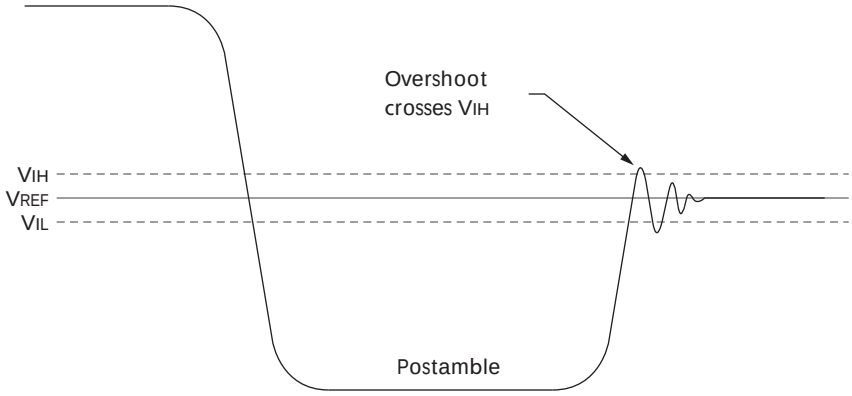


After the data has been transferred into the 4-to-1 MUX with the WRITE pointer, the READ pointer then shifts the data into the CORE_CLK domain in the DQIL and DQIU flip flops. This type of scheme works well when the relationship between the DQS and the CORE_CLK is mesochronous. The READ delay, or outer loop delay, is the number of clock cycles from when the READ command leaves the CORE_CLK domain until the READ data returns to the CORE_CLK domain. The READ pointer is then set to increment a number of CORE_CLK cycles after the READ command leaves the CORE_CLK domain.

The number of CORE_CLK cycles for the delay can be fixed or programmable through a register to set the READ delay. A programmable delay works best for debugging and flexibility with multiple DRAM latencies.

The READ and WRITE pointers need to be reset at power up. You can also reset the pointers during idle cycles, but it is recommended to disable this for debugging because hard fails may not occur if a single edge is corrupted. The DQS and DQS# are gated through the strobe detect circuit to guarantee that good edges on the strobe are always seen by the WRITE pointers. If the strobe detect circuit produces bad edges, data capture will be corrupt until the pointers are reset. The strobe detection and strobe delay circuits are critical in this type of circuit. Figure 13 illustrates an example of a bad edge. In this example, the DQS signal rings back through VIH on the postamble. This can cause the capture circuit to read this as a valid edge on the DQS and overwrite data in the FIFO if the strobe detection circuit is not turned off.

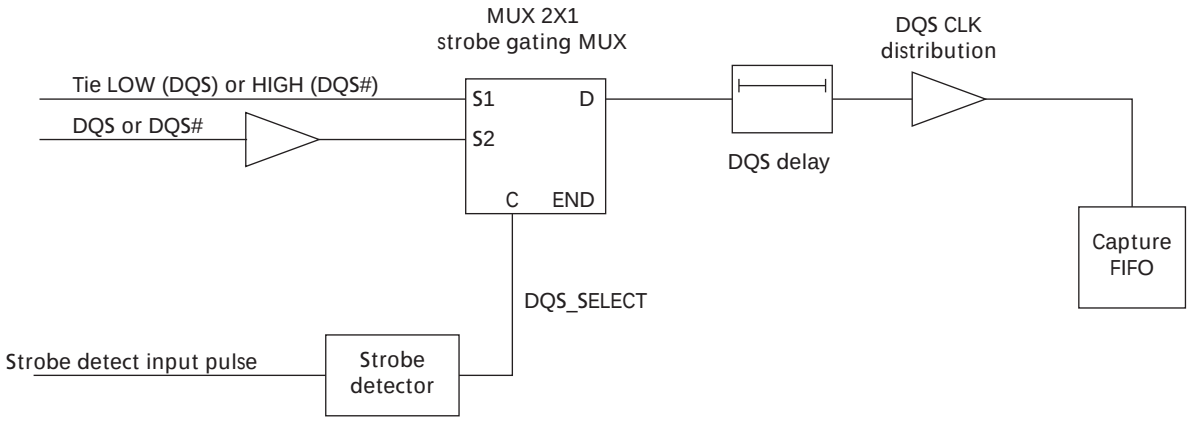
Figure 13: Postamble Glitch



Strobe Detection and Delay Circuits

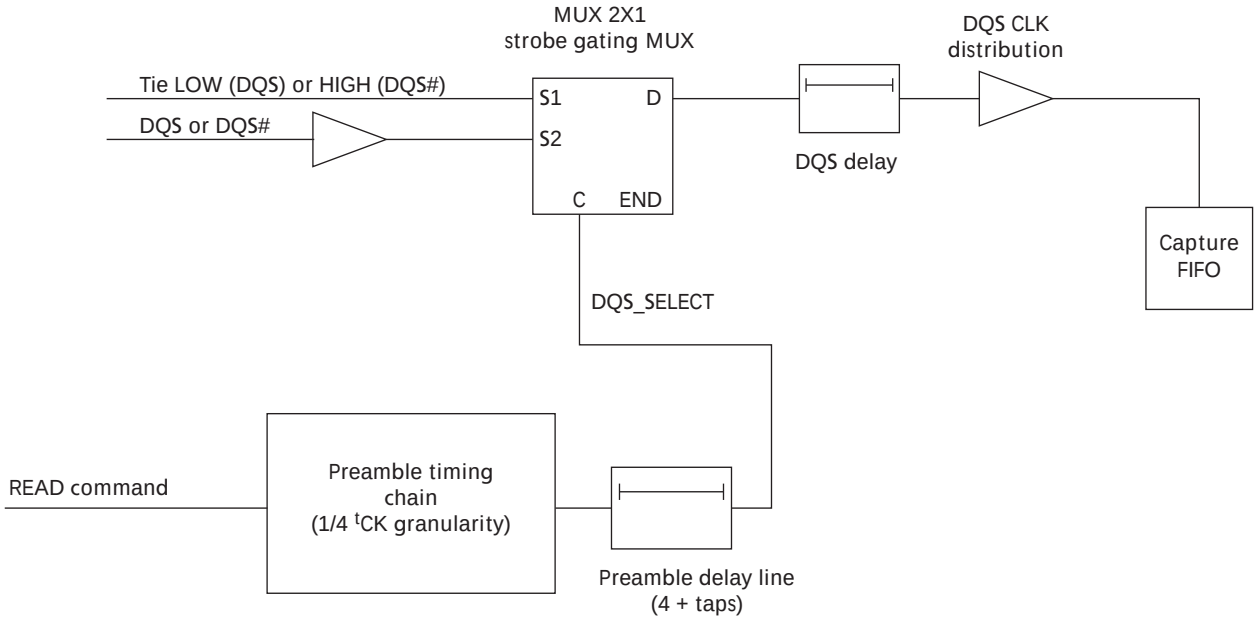
Now we need a way to enable and disable the DQS and DQS# signals during the preamble and postamble, respectively. At a high level, a 2-1 multiplexer is added in the DQS/DQS# path directly behind the input receiver to enable and to disable the DQS and DQS# signals. Multiple ways exist to enable and disable the DQS and DQS# signals. We will first look at the enable circuit and then look at the disable circuits.

Figure 14: Strobe Detect - Top Level



Starting from Figure 14, add a simple self-timed chain to enable during the preamble. This circuit is similar to READ delay by counting clock cycles from the time the READ command leaves the CORE_CLK domain to the center of the preamble. By using both the CLK2X_0 and the CLK2X_90, you can achieve 1/4 clock granularity. A small delay line can then be added with four or more tap points for added granularity. The delay line can also be used to calibrate the DQS to DQ relationship at power up. This type of training is recommended when you move to the faster DDR2 speed grades and higher. The calibration can also be done in the lab and then set in the bios if power-up calibration is not possible.

Figure 15: Strobe Detect – Preamble Turn On



In the postamble disable circuit, it is critical to avoid glitches. Referring back to Figure 13 on page 14, this type of ringing in an early ^{t}DQSS situation can cause the receiver to latch data if the DQS and DQS# are not turned off in time. This has been seen in multiple controller and DRAM devices over the years starting with DDR1. Three methods will be discussed, including the timing chain, VREF offset, and self-timed postamble method.

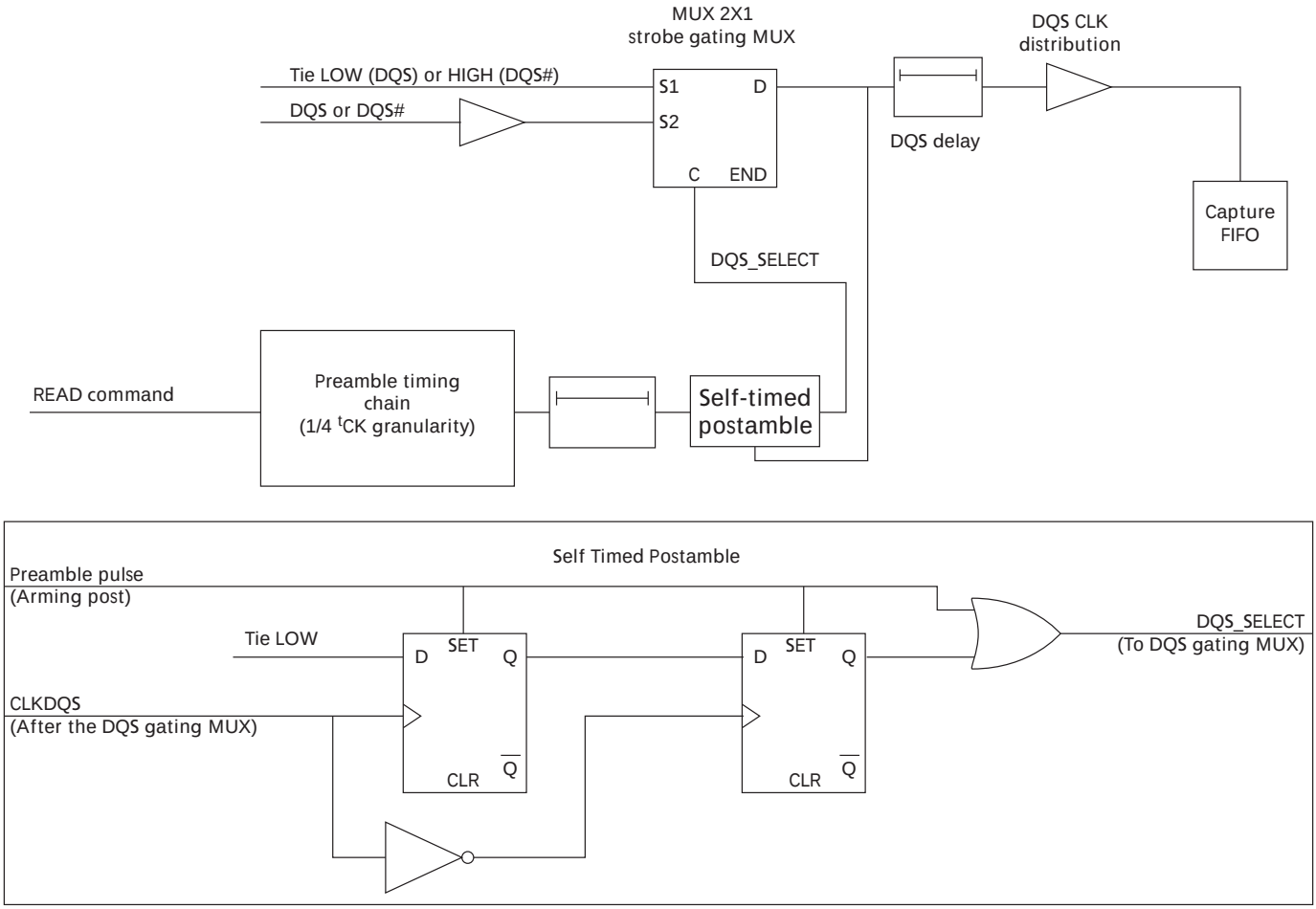
The timing chain method works in the same fashion as the preamble enable, but with a smaller data valid window to the size of the specified for the postamble. The timing chain is set to detect the postamble at BL/2 + 1/4 ^{t}CK after the preamble detect circuit has been detected. This is quite a simple approach with no special timing considerations, but because of the short postamble, it is not recommended for higher speed designs. If this method is implemented, it is critical that the DQS and DQS# detection circuit is off before the ^{t}DQSS MIN time frame.

The VREF offset method has been tried a few times with varying results. The idea behind this circuit is to offset the internal VREF signal so a tri-stated DQS gets interpreted by the input buffer as a logic LOW with DQS# a logic HIGH. The detector is then left on from the preamble to BL/2 + 1/2 ^{t}CK. The preamble/turn-on can be relaxed to extend the strobe detect pulse. The pointers can be reset when there is no READ data on the bus. This approach is also simple to implement, but it has multiple drawbacks. If the internal VREF has been offset, the loss of voltage margin typically rules out this type of circuit. Due to the voltage margin issues, this circuit would not be recommended for any speed.

The self-timed postamble method does add complexity, but it also solves most of the problems created by higher speed DRAM. The self-timed postamble method uses the preamble pulse to “arm” a self-timed circuit with a pulse width of BL/2 - 1 ^{t}CK in length. This allows the last rising edge of DQS# or the last falling edge of DQS to turn off the strobe detect circuit. By having the final edge turn off the strobe detect circuit, any ringing in the postamble will be ignored. The circuit then stays off until the next preamble detect is hit. The basic methodology here is to count edges and to turn off on the last edge.

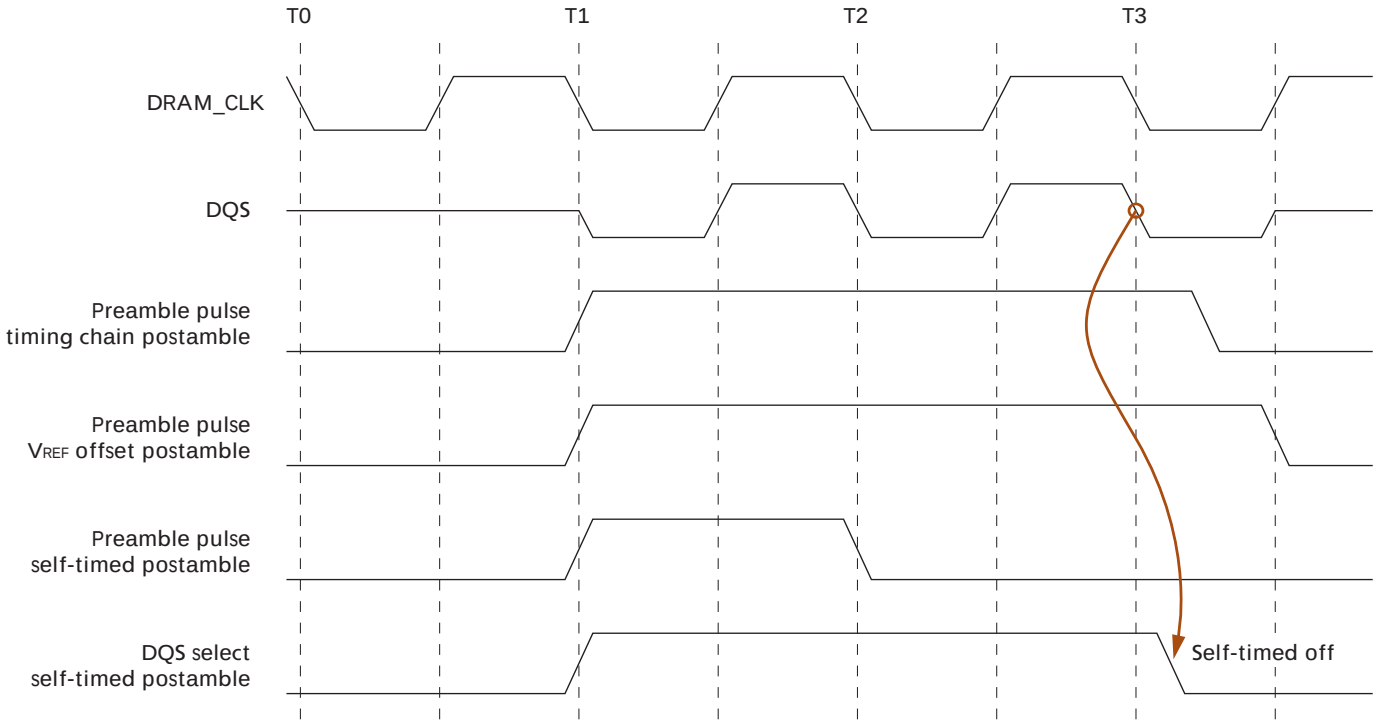
This method also eliminates the voltage margin issues you get from the VREF offset method described earlier. A couple of design challenges exist when implementing the self-timed postamble. You have to be careful about asynchronous design and timing, and you do have some design-for-test considerations. In addition, to avoid glitches, you must pay special attention to the asynchronous path through the flip flops.

Figure 16: Self-Timed Postamble



The strobe detect circuit is inserted between the programmable delay line and the MUX, as shown in Figure 16. The timing diagram in Figure 17 on page 18 shows the internal signals for all the preamble and postamble detect circuits. As seen in the timing diagram, the self-timed postamble turns the DQS detection off with the last falling edge on DQS.

Figure 17: Strobe Detect Timing

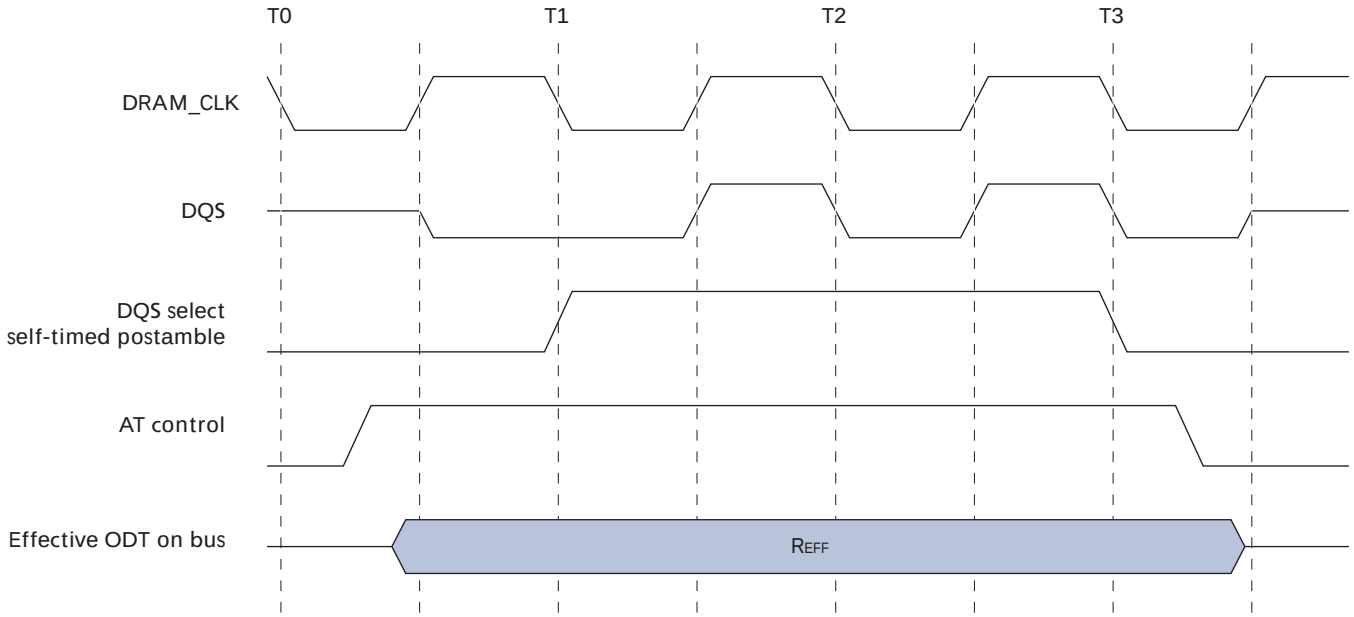


ODT Control

ODT is typically required as the speeds increase into the DDR2 and DDR3 range. The impedance of the ODT is typically a trade-off between signal integrity and power consumption. In most cases, the Rtt value is slightly higher in comparison with the RON value of the DRAM. By doing this, the signaling will not be as clean, but the gain in voltage margin helps close timing. An even higher Rtt can be chosen to reduce power like 150Ω. The real question comes down to whether it's good enough to still close timing and to burn as little power as possible.

To control the ODT, you can use a simple timing chain approach, leveraging the preamble turn-on circuit as a reference point to bracket DQ and DQS. An effective reference point is the center of the READ preamble. As an example, assume ODT control uses a 1/8 to 1/4 clock delay before the ODT is on the bus. A typical delay for turning on the termination would be 3/4 of clock before the preamble with the off happening 1/4 clock after the burst length divided by 2. The termination needs to be on for the entire length of the preamble and postamble to cover the strobe being driven by the DRAM and possibly by a second rank of DRAM.

Figure 18: ODT Timing



Strobe Delay

The data from the DRAM comes edge-aligned with the strobe and needs to be shifted by 90 degrees to center-align the data valid window to the strobe. There are several different methods to accomplish this task, including board trace delay, fixed controller silicon delay, and a programmable delay line. With clock speeds over 200 MHz, it is recommended you use a programmable delay line.

A board trace delay is a very simple approach that stays constant over temperature and voltage. The largest problem with this type of approach is that it limits the design to one frequency, and there is no way to calibrate the data valid window. A board delay also can remove margin on the WRTIE side of the timing budget.

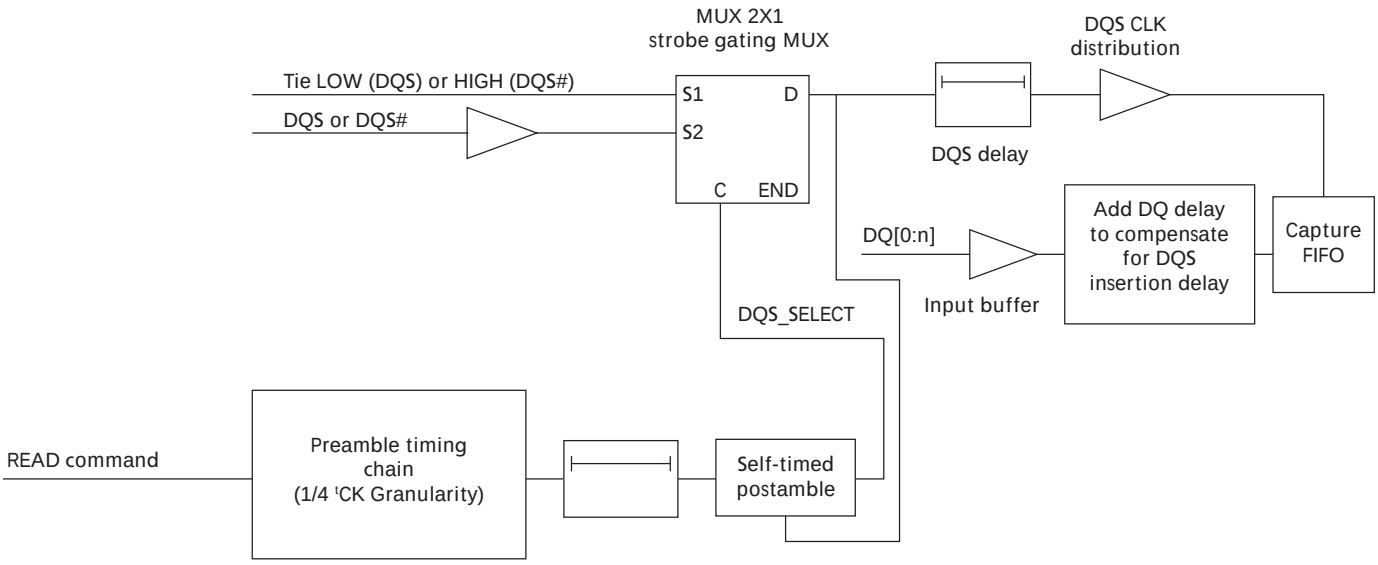
A fixed silicon delay is a common approach seen in quite a few designs today. The problem with this type of approach is the process dependency on the controller. Voltage of temperature variation will have a profound effect on the location of the data valid window in relation to the strobe. The process dependencies are on the manufacturing process of the DRAM and the controller.

The process dependencies can be removed with a programmable delay line. The programmable delay line enables the controller to calibrate the DQ to strobe relationship to find the center of the data eye for every part. This can be done at power up, or it can be done real-time to adjust for voltage and temperature variation. To track this dynamically, you need to monitor phase shifts in the DLL or PLL. If there are a certain number of phase shifts, the temperature or voltage has changed, which can trigger the recalibrate process. The granularity of the calibration can be everything from a per byte lane to per bit. If a per bit training is required to close timing, this can be done with one DLL and multiple slave DLLs.

The programmable delay line also adds flexibility to the board design relaxing the trace matching requirements with the ability to train out mismatch in the byte lane or even in the DQ in the byte lane if the training is done on a per DQ basis. The programmable

delay does add insertion delay into the READ data path, but at high speeds, this typically is a good trade-off. Figure 19 illustrates where in the READ data path the programmable delay line would reside between the MUX and the DQS clock distribution tree.

Figure 19: Strobe Delay



In Figure 19, the insertion delay is also shown between the input buffer and the capture FIFO. This is needed as the speeds increase and as the DQS insertion delay compared with the DQ insertion delay approaches the desired 90 degrees. Delay is added to the DQ data path to compensate for the added delay of the DQS signal. The type of delay to use depends on the application and how much flexibility is needed in the controller.

The simplest approach is to use a fixed silicon delay. The only drawback with this method comes from the added latency if the controller is designed to run at multiple speeds. When you add the delay, it needs to be done for the fastest speed the controller could be run at. Lack of added control and minimum die penalty are major advantages to this type of control.

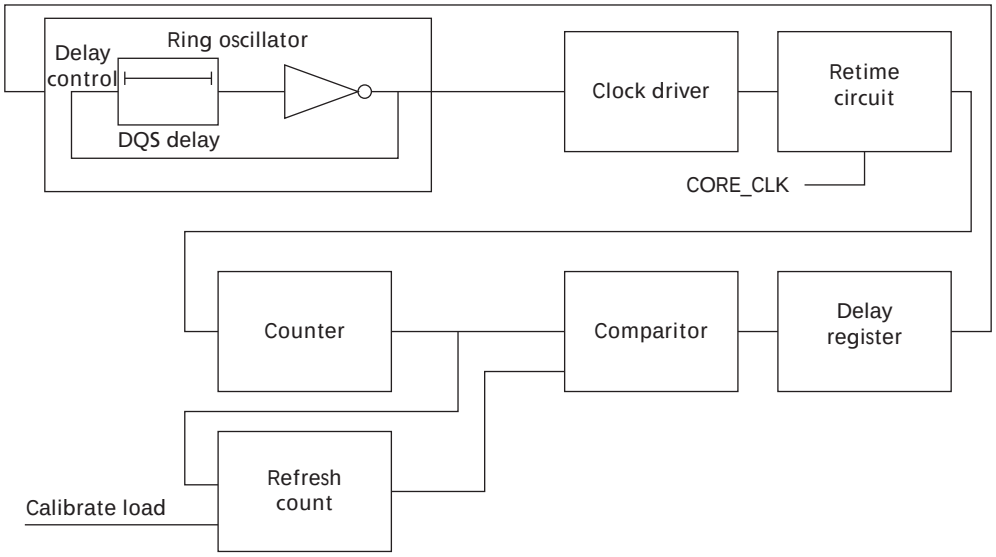
The programmable delay line removes the increased latency issue found with the fixed delay at the slower speeds if the controller is designed to run at multiple speed ranges. The added complexity can be mitigated by using a shortened version of the DQS delay line. The programmable delay line should be designed to match the delay seen on the strobe through the DQS delay line. The difficulty comes from making the two delay lines symmetric.

A locking strobe delay becomes necessary when process, voltage, and temperature variations of the strobe delay line break the timing budget. This can happen with data rates above 400 Mb/s, but it is application-dependent. To lock the strobe delay, you can use a couple of different approaches, such as analog or a digital delay line. In most cases, a digital delay line works the best. The major advantages of a digital delay line over the analog style delay line come from the design difficulties and larger die penalty area. The power on the digital delay line can also be drastically reduced by using one master delay line with multiple slave delay lines. Two types of digital delay elements, including the ring oscillator and the phase control clock, and their advantages and disadvantages will be described.

Ring Oscillator

The master delay line is used to create a ring oscillator for control of the delay line. The ring oscillator is divided down and retimed to the CORE_CLK. The CORE_CLK is then used to count the high time of the divided ring oscillator. The physical count will be directly proportional to the delay in the delay line and can then be compared to slave or reference count. If the count is larger, the reference delay is incremented. If the count is smaller, the reference delay is decremented. Careful consideration needs to be made on the updates to avoid glitches. One way to avoid a glitch is to wait for AUTO REFRESH command periods for updates. This will ensure the data bus is idle.

Figure 20: Ring Oscillator

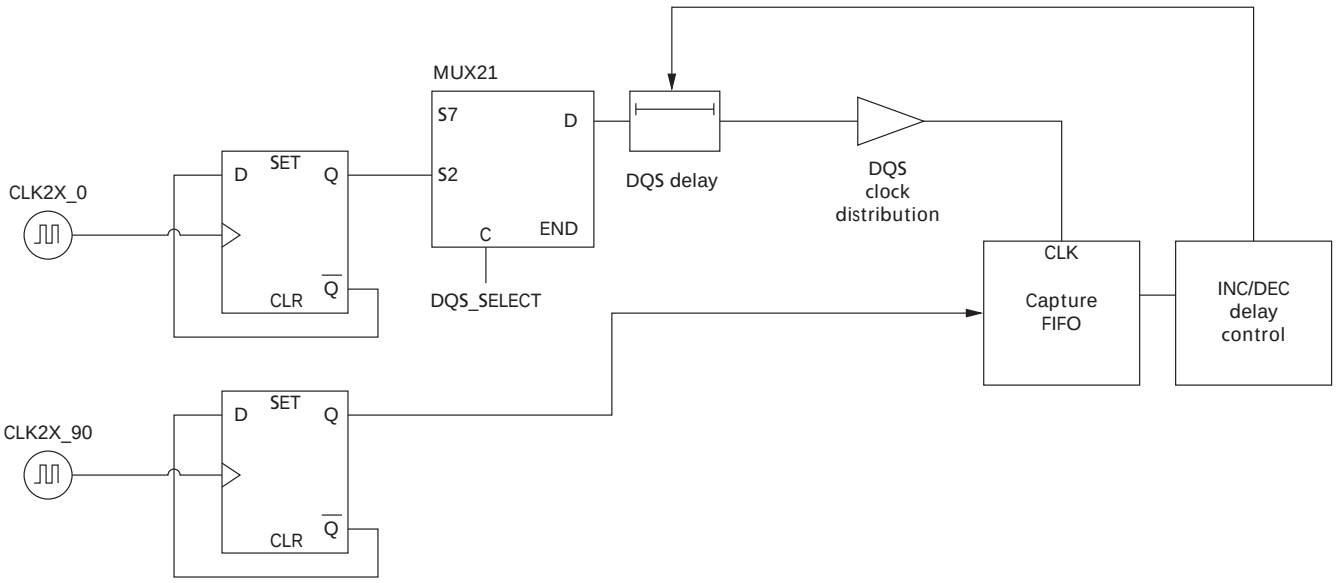


The ring oscillator allows for any insertion delay to be locked so the controller is not tied to a frequency. In this relatively straightforward approach, the 90 degree out-of-phase controller clock is no longer needed. If implementing a ring oscillator style approach, pay careful attention to DQS distribution copies into the oscillator. If this proves to be difficult, one alternative is to ignore or model the DQS distribution nets. The typical oscillating frequency is roughly 2x the DRAM clock, but it can be increased if better resolution is required.

Phase Clock Control

The phase clock control uses the actual DQS to DQ circuitry to capture toggling inputs 90 degrees out of phase. The capture FIFO is then used as a phase detector and increments or decrements as needed to maintain a constant delay. The target delay should be the same as the 90 degree delay needed for the master locking circuit. If this is out of phase, the accuracy of the slave delay will suffer. Figure 21 on page 22 shows the phase clock control added to the top level functional block diagram of the strobe detect and shift circuit.

Figure 21: Phase Clock Control



The phase clock control is simple to implement because the design already exists. The other major advantage to this type of approach is that it is very well-matched using the actual capture gates, which will track well over voltage and temperature variations. The drawbacks to the phase clock control include the added die area and the loss of accuracy if the target delay is not matched to 90 degrees. This may be mitigated if the controller has other clock phases available that better match the accrual target phase offset.

At the end of the day, both the phase clock control and the ring oscillator are solid approaches with advantages and disadvantages. The decision on which approach to use is design-dependent and application-dependent.

Data Training

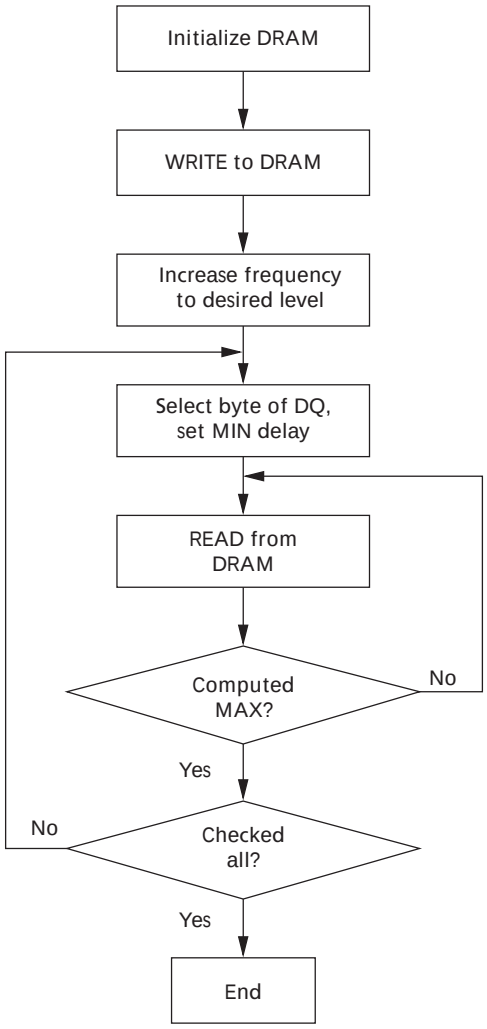
Data training can be accomplished for both the READ and WRITE sides of the data path. Unfortunately, the JEDEC-standard DDR2 and DDR3 devices do not have a built-in mechanism to make this sequence straightforward. The following method has been implemented in some controllers to deal with the fast data rates that would be seen in the high end of DDR3, but it would also work with DDR2 if additional timing margin is needed. To start the data training, the READ side from the DRAM must be calibrated first.

It is difficult to calibrate the READ data path without having the WRTIE side calibrated, and you do not know whether the WRITE was successful to the DRAM. To overcome this issue, the DRAM device can be powered on at a reduced speed and then initialized to lock the DLL. The reduced speed should be at a point that is low enough that you are comfortable the data will be 100 percent correct. After the DRAM device is ready for operations, the controller needs to write a known pattern to the DRAM. With the data now stored in the DRAM device, the clock frequency needs to be changed to the accrual operating frequency of the system. With a stable clock, the DLL can be reset through the DRAM's mode register.

If changing the clock frequency is not an option, you can also do the training in two stages. The first stage would write a data pattern, such as 0xC, that would not stress the bus. This doubles the size of the DVW, allowing the controller to find a point in the window that is known good for the second stage of training with a PRBS pattern.

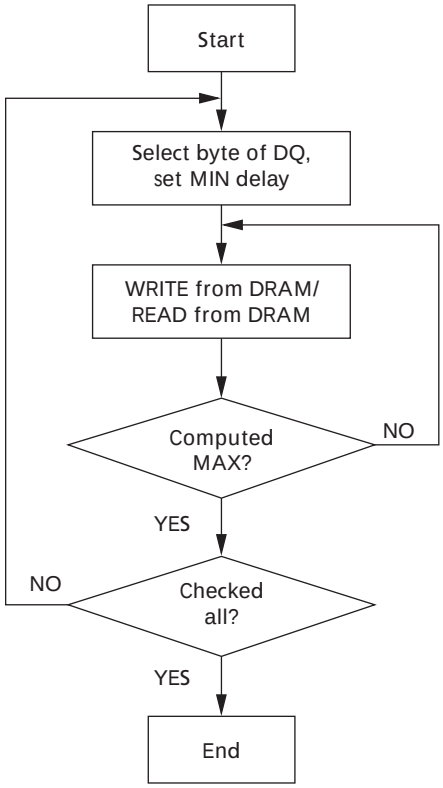
With the system running at speed and the DLL locked, the controller needs to set the tap points on the delay line used for calibration at the minimum delay. The controller now issues normal READ commands to the DRAM. REFRESH commands may still be needed to ensure the data in the array is not corrupted. After the READ command is latched and if the delay line is not at the maximum value, the delay line is incremented by one, and another READ is issued. After each READ command, the controller needs to continue incrementing the delay line until the maximum is reached. Keep in mind the first few and the last few stages of the delay line might produce incorrect data. The controller needs to keep track of the number of passing delay stages with a phase detector and phase interpolator with the DQS being placed in the middle of the data eye. Figure 22 shows a flow chart of the READ data training process.

Figure 22: READ Data Training Flow Chart



After the READ data training is complete, the WRITE side of the data bus can be calibrated. The WRITE calibration is performed in a similar fashion to the READ training except without the frequency change because the READ side of the bus is already calibrated. To start the WRITE training, the controller selects the byte or DQ to start training. The delay line is set to a MIN value, and WRITE commands are issued to the DRAM at full speed. The controller increments through the delay line until the last tap point is hit by issuing WRITE followed by READ commands. Figure 23 shows a flow chart of the WRITE data training process.

Figure 23: WRITE Training Data Flow Chart



Physical Integration

In the physical integration of the DRAM controller, keep in mind the following circuit recommendations. The placement of the DQ/DQS byte lane layout macros should duplicate for each DQ, DQS, CLK, and command/address. As the speeds increase, fixed and matched routing should also be implemented to reduce on-chip skew in the routing.

After you decide which Tx circuit approach to use, make sure you use the same Tx circuit for the DQ, DQS, CLK, command, and address so that the out delays match. For example, if you use different Tx circuits for different outputs, such as a complex Tx circuit for the high-speed pins (DQ) and a simpler approach for the lower speed pins (address), you risk introducing delays between the address capture and the data capture.

Also, the DQS clock trees should be the same for each byte. Differences in the bytes can be tolerated if the strobe to DQ relationship is calibrated on the byte lane or per bit basis. Failing to do so can cause a loss of margin in the DQS to CORE_CLK domain crossing.

To avoid on-die process temperature and voltage variation across different bytes, the master delay line should be physically placed as close to the DDR interface as possible and centered between the byte lanes. This can be a problem if certain areas of the silicon get hotter in relation to other areas. Voltage variation can be seen when the local power bus drops in certain areas of the die.

Conclusion

The logical interface of high speed DRAM controller is becoming particularly challenging as the speeds increase moving to DDR2 and DDR3. With data rates increasing as we move to DDR2 and DDR3, special attention needs to be given to the transmit and receive circuits of the DRAM-based controllers to ensure proper operation of the memory system. And though this technical note has described several techniques that can be used in DRAM-based controllers, it doesn't cover the only way to design DRAM-based controllers to run at DDR2 and DDR3 speeds.

8000 S. Federal Way, P.O. Box 6, Boise, ID 83707-0006, Tel: 208-368-3900
prodmktg@micron.com www.micron.com Customer Comment Line: 800-932-4992