Quad Data Rate™ (QDR™) SRAM Clocking Scheme

The Cypress QDR™ Device

Most traditional synchronous SRAMs have a single clock input controlling both the input and output registers of the device. This approach works well in systems where the flight time from each SRAM to the receiving device is small when compared to the access time of the SRAM. When frequencies increase and access times shrink flight times become significant. SRAMs farthest from the controller have the most difficult task during read operations. The first disadvantage is that the SRAM farthest from the controller receives the clock signal to trigger the output register during a read access a certain amount of time after each of the other SRAMs residing closer to the controller. In *Figure 1*, it takes an additional 300 ps for the clock signal to reach SRAM #2 after reaching SRAM #1. The next disadvantage is that it will also take longer for the data generated by SRAM #2 to reach the controller than the data generated by SRAM #1. This, again, is due to the flight time difference between each SRAM and the controller. When the flight time of each SRAM becomes significant when compared to the access time of the SRAM, another approach is required.

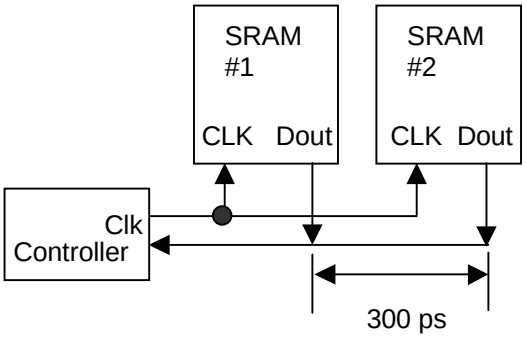

Figure 1. Flight Time Delays Both the Clock and Output Data

The QDR SRAM family addresses this issue by providing a set of both input clocks (K/K#) and output clocks (C/C#). These independent clocks allow the user the ability to eliminate the flight time differences back to the controller, which is described below.

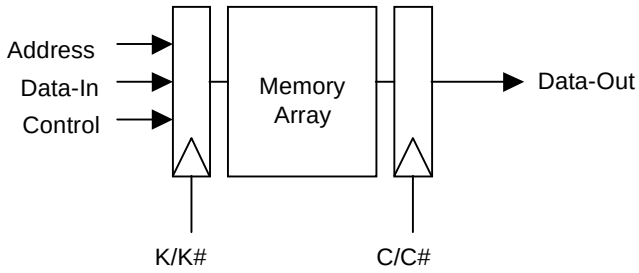

Figure 2. QDR SRAMs Have Both Input and Output Clocks

Using the Separate Input and Output Clocks

One way of using the separate input and output clocks of the QDR SRAMs is shown in *Figure 3*. It is important to note that the clock generated by the controller is fed to K/K# of the SRAM. This same clock signal is allowed to propagate past the farthest SRAM and is looped back to the controller. On the way back to the controller it is fed into the SRAM again and used as the output clock, C/C#. Of course, additional clocks may be needed to reduce loading and provide the signal integrity required. Each system designer will have to make his or her own trade-off.

First, let's evaluate the input timing to the SRAM. If care is taken to closely match the trace lengths of the signals from the controller to the SRAM (Address, Data, Control, and clocks), then the only difference between the signals at the pins of each SRAM and those at the pins of the controller is the time skew caused by the flight time. Let's take an example by assuming that an address is generated at t=0 ns and a clock is generated at t=1.0 ns. Both have a flight time to the SRAM of 0.2 ns. The address will arrive at the SRAM at t=0.2 ns and the clock will arrive at 1.2 ns. The timing relationship between these two signals is maintained since both need to travel the same distance. This is true for all signals generated from the same controller if the trace lengths are tightly controlled. Therefore, generating the clock with the address, data, and control signals destined for the SRAM eliminates the flight time differences to each SRAM.

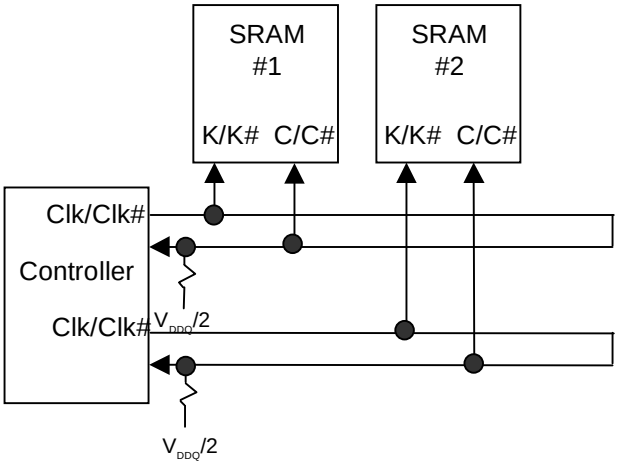

Figure 3.

Now, lets take a look at the output timing in this configuration. First, notice that the clocks loop back to the controller after passing the last SRAM. On the way back to the controller, the output clock signals (C and C#) are delivered to the SRAM. By looping back the clock, the output clock is allowed to trigger the register of the SRAM farthest from the controller in advance of the other SRAM closer to the controller. By doing so, the device farthest away is given a "head start" during read accesses when compared to the other SRAMs closer to the controller. In fact, the "head start" is the same as the flight time difference back to the controller. *This technique will result in the data from all the SRAMs arriving at the controller at essentially the same time.*

Latching Data At The Controller

The QDR family is able to provide an extremely high bandwidth of data. In order to take advantage of this, the user must be able to reliably latch the data at the receiving device. This section describes one method for latching the data provided by the SRAM.

In the implementation described above, the clock used to trigger the output register of the QDR SRAM is also fed back to the controller. The clock-to-data relationship at the pins of the SRAM is shown below at 167 MHz. Since both the data and clock have the same flight time back to the controller, the timing relationship between the two is maintained. As can be seen, clock #1 is responsible for clocking data "A" out of the SRAM. Data "A" is valid t_{CO} after the rising edge of the output clock C. Clock #2 (the rising edge of C# since the device only recognizes rising edges of the clocks) is responsible for clocking out data "A+1". In addition, the SRAM will "hold" data "A" for Tdoh (1.2 ns) from clock #2.

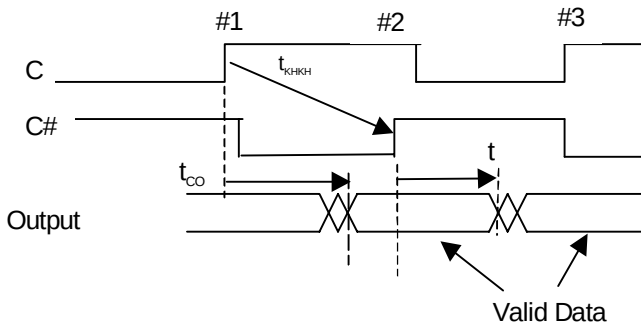

Figure 4. Timing Diagram

Let's assume that the QDR SRAM is being used at 167 MHz (t_{CYC}=6.0 ns). In addition, Tkhkh (rising edge of K to rising edge of K#) is a minimum of 2.7 ns. In addition, we will assume that t_{CO}=2.5 ns and t_{DOH}=1.2 ns. All of these are the worst case spec limits for each parameter. Therefore, the worst case "window" of valid data is:

Window of Valid Data = $(t_{KHKH}-t_{CO})+t_{DOH}$

In this example, the window of valid data = (2.7 ns-2.5 ns)+1.2 ns=1.4 ns.

As can be seen in *Figure 4*, the 1.4 ns "window" of valid data is guaranteed in relation to the rising edge of C#. The data will be available 0.2 ns before the rising edge of C# (since t_{CO} is 0.2 ns less than the minimum t_{KHKH}) and held for 1.2 ns afterwards (the guaranteed value of t_{DOH}). It is important to note that clock #1 is responsible for clocking out data "A", but in fact may not be the best clock to latch the data at the controller. In this example, C#, the clock on which data "A" is held from, is the best clock with which to latch the data. Once the data is latched inside the controller, it can be synchronized with the controller clock.

At slower frequencies it may be advantageous to use a single clock to trigger both the input and output registers of the device. In this case, the device would look like a standard synchronous SRAM and the same timing analysis techniques would apply.