# Interfacing the QDR™ to the Delta39K™

## QDR™: An Introduction

With the continuos demand for higher performance data processing systems, memory devices are evolving to more closely match the needs of these applications. Specialized memory products that optimize memory bandwidth for a specific system architecture are successfully increasing overall performance in a variety of data processing systems. This application note introduces the QDR™ architecture, an SRAM architecture designed to improve the SRAM interface bandwidth by more than 4x the current solutions. This application note introduces the Delta39K™ and the QDR SRAM devices and provides details of interfacing the SRAM to the Delta39K CPLD.

## Delta39K™: An Introduction

Cypress's Delta39K CPLD family represents a major milestone in the programmable logic industry. Built with an advanced 0.18-µm process, the Delta39K family offers densities from 15K to 350K usable gates–including the world's largest CPLD with 5376 macrocells. Delta39K CPLDs provide more than five times the amount of embedded RAM compared to any other programmable logic device. Additionally, on-chip, dedicated FIFO flag and DualPort arbitration logic allows you to build fast and efficient memory functions without compromising any macrocell logic. There is also a Spread Aware™ PLL, which allows clock multiplication, division, and skew adjustment.

These devices also feature completely independent Core and I/O voltage networks to allow any combination of 1.8V, 2.5V and 3.3V on the I/O or Core $V_{CC}$ pins. Furthermore, the highly flexible I/O structure supports standard I/O levels (PCI, LVTTL, LVCMOS) as well as high performance memory and backplane interfaces including SSTL, HSTL and GTL+.

## What is a QDR™?

QDR is a family of synchronous SRAMs with an innovative architecture designed especially for High Performance Networking systems by the QDR Consortium, comprised of Cypress, IDT, NEC, Micron and Samsung.

QDR Stands for Quad Data Rate, which effectively means that 4 pieces of data can be transferred through the SRAM in a single clock cycle. The QDR Family includes two devices, which differ by the number of data words that will be bursted out on every access.

*Table 1* shows the current devices in the family.

Most of the existing SRAM solutions are relics from the PC era, with interfaces designed to move data efficiently for PC type single-I/O applications. In most networking applications continuous movement of data through the SRAM is a necessity. In such applications, there are continuous transitions between read and writes through the memory. Single-I/O devices like standard Synchronous Pipelined SRAMs do not perform well under these conditions. The NoBL™/ZBT™ family of SRAMs have optimized the synchronous SRAM architecture to allow no latency in the read/write transitions and have a 100% utilization of the I/O bus.

But in most networking applications, this improvement in throughput is not enough. So, applications that need to have the read and write done simultaneously will benefit tremendously from the QDR SRAM

**Table 1. Devices in the QDR Family**

| Device | SIze | Description |
|---|---|---|
| CY7C1302 | 512K x 18 | QDR - Burst of 2 |
| CY7C1304 | 512K x 18 | QDR - Burst of 4 |

Applications such as ATM switches and routers will benefit from the fact that simultaneous Reads and Write can be done on the SRAM with no latency and the data through the SRAM is guaranteed even for simultaneous access to the same address location.

For more details on the QDR refer to the application note "QDR: The Next Generation High Performance Networking SRAM,"

### CY7C1302

*Figure 1* shows the block diagram of the CY7C1302 QDR device. The QDR SRAM family has two members: the two-word burst device (CY7C1302) and the four word burst device (CY7C1304). The difference between the two is in terms

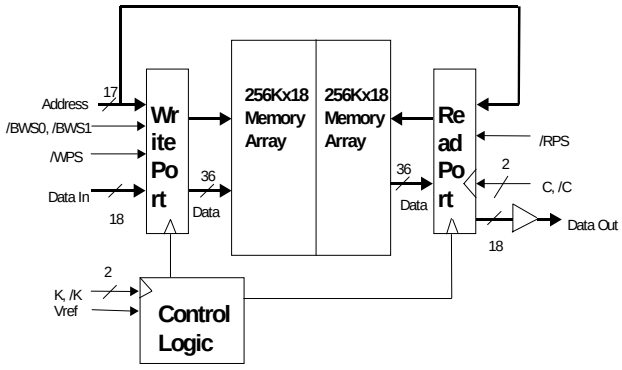



**Figure 1. Block Diagram of the CY7C1302.**

of the number of words of data that can be obtained from the SRAM on a single read or provided to the SRAM on a write.

Data ports are separated for the read and write ports. The address lines are shared by both the read and write ports. Data is transferred in a double data rate manner (DDR) on both input and output ports. This allows four words of data to be transferred on every clock cycle.

### Timing

*Figure 2* shows the timing diagram for the CY7C1302, the two word burst device on the QDR.

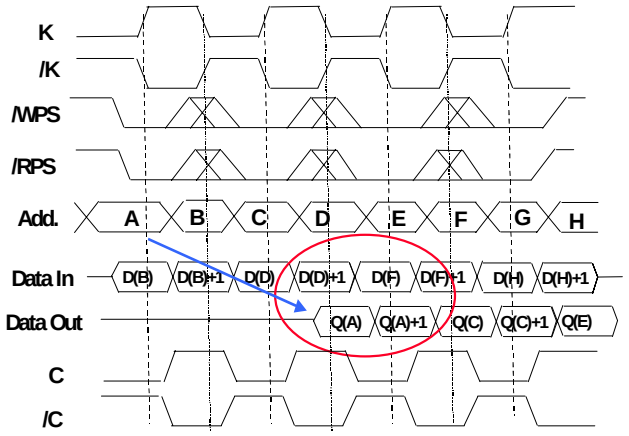



**Figure 2. Timing Diagram of the CY7C1302.**

In the first clock cycle, /RPS and /WPS are both asserted active LOW. The address for the read (A) is latched on the rising edge of the K clock and the address for the write (B) is latched on the rising edge of the /K clock.

The data for the write to address B, is latched on the rising edges of K clock (D(B)) and /K clock (D(B+1). The byte writes are also latched on the same clock as the data.

Read accesses to the CY7C1302 are conducted in 2 cycles (a 2 stage pipeline). During the first cycle the address is latched on the rising edge of K clock. The address is then presented to the memory. The next rising edge of K clocks out the first 18-bit data word (Q(A)). The next rising edge of /K clocks out the second 18-bit data word (Q(A+1)).

As seen from the timing diagram, one could start QDR read and write to the same address on the same cycle. When such an operation occurs, the QDR forwards the write data to the read port and ensures that valid data is driven out on the data bus. By doing so, Data coherency is guaranteed.

### Memory Controller for the QDR SRAMs

To simplify the process of designing an interface to the QDR SRAM, a memory controller was developed using the Delta 39K CPLD. A block diagram of the memory controller is shown in *Figure 3*.

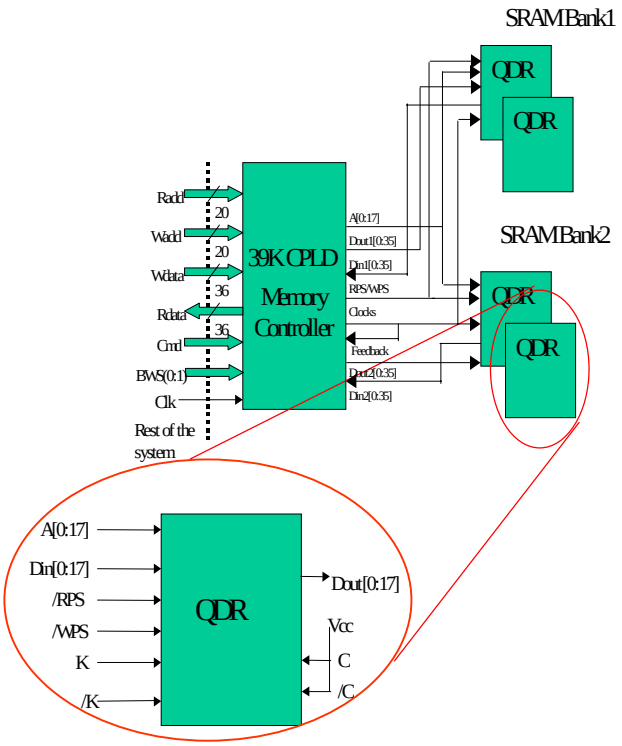



**Figure 3. Memory Controller Interface.**

The memory controller is designed to control four SRAMs. The four QDRs are split into two SRAM banks each consisting of two QDRs. Each of the SRAM banks receives 36-bit data and a 18 bit address along with all the control signals and clocks. The SRAMs form two banks of 512Kx 36.

The memory controller generates all the control signals for the memory array, It supports concurrent double data rate operation on all the inputs and outputs, it also allows byte write operation to the memory bank.Memory Controller for the QDR™ SRAMs.

The memory controller assumes that the QDR SRAMs are in the single clock mode. This can simplify the memory interface. It operates at 71 MHz, allowing a bandwidth of 10.22 Gbits/sec. The memory controller has independent read and write state machines. The memory controller is a command based interface with a two bit command input.

## VHDL Implementation

Appendix 2 shows the VHDL Implementation of the interface. The interface is tuned to be able to read and write 144 bits of data in each clock cycle at 71 MHz, i.e. 72 bits of data during a write and 72 bits of data during a read.

## Implementation Details

Appendix 1 shows the logic block diagram of the Memory Controller. The memory interface is composed of four 36-bit

buses and one 18-bit address bus and the host interface with the 72-bit data bus and an 18-bit address bus.

The CPLD operates internally at 142 MHz. Externally, the buses need only operate at 71 MHz, since the DDR interface transfers data on both the leading and trailing edges. The 72-bit read data path from the host is internally split into two 36-bit sections and latched by separate registers. These registers are clocked at 142 MHz, allowing one to send or receive data on both edges of the clock.

The different blocks in the memory interface are discussed below:

### 1. The PLL

The PLL takes the original clock signal from the user and splits it into the global clock (glbclock), the 2x global clock (i2clk) and the address clock (address_clk). The 2x global clk is used to generate the “k” and “kn” signal which clocks the QDR SRAM. Like the 2x global clock, the address clock is twice as fast as the original clock. The address clock is offset by 45 degrees from the K clock. The address clock is used to generate the address for the QDR. The 45 degree phase allows the generation of the QDR address ahead of the K clock. This allows the address to meet the setup time requirements of the QDR.

The “kfb” signal serves as the external feedback for the PLL. With the external feedback connected to the output clock of the QDR SRAM, the PLL can deskew the signals between the QDR SRAM and QDR controller. Without board deskewing, there could be the potential problem of reading and writing data incorrectly.

### 2. K Generator

Kn is the inverted signal of K. They are external signals sent to the QDR SRAM to trigger the reading and writing of data. Though the frequency of both K and Kn is the same as the glbclock signal, the rising edge of both K and Kn are used as the read and write trigger for the QDR SRAM. *Figure 4* shows the DDR operation.

Two clocks must be shifted to accommodate the delays on the board, this is done by using the PLL. Since outputs from the PLL cannot connect directly to an external signal, an alternate means of generating the K signal must be used. The K generator generates the K and Kn signal from the rising edge of the i2clk. During each rising edge of i2clk, the K and Kn signals are inverted with respect to itself. K and Kn are two times slower than i2clk. Since the K and Kn signal should be at the same speed as the glbclock, this signal generation scheme fits the design perfectly.

### 3. Data Parser

Triggered by the global clock, the data parser parses 72-bit data written into each of the QDR SRAM banks into two 36-bit sections. The parsed sections are then stored into the signal “input data” and “inputdata_1”. The signal “input data” contains the first 36 bits of the input while the signal “inputdata_1” contains the last 36 bits. This enables the QDR controller to send the input data in two 36-bit fragments to the QDR SRAM banks at DDR.

The parser also latches the write data address and the read data address into an internal register. This will prevent the address data from being corrupted due to signal fluctuations later on.

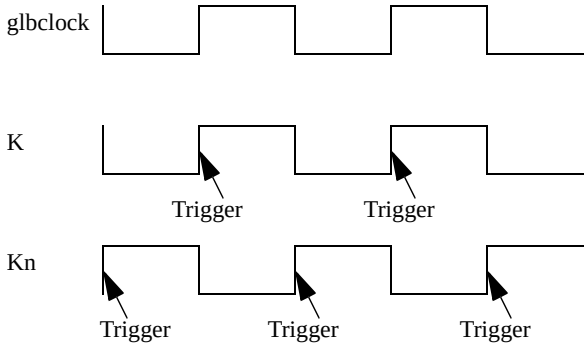



**Figure 4. Generation of the QDR Clocks.**

### 4. QDR Transmitter

The primary function of the QDR transmitter is to transmit data and address data to the QDR during a data write. It is synchronized with the address clock.

When the signal “K” is LOW, the QDR SRAM banks are ready to receive the first 36-bits of the input data. At the same time the address for reading data occupies the 18-bit address data line. When the signal “K” is HIGH, the QDR SRAM is ready to transmit the last 36 bits of the input data. At the same time the address for reading data occupies the 18-bit address data line. This is done to satisfy the timing requirements of the QDR SRAM (see *Figure 2*).

### 5. QDR Receiver

The main functionality of the receiver is to receive and integrate the received data back to its original form. The QDR receiver is synchronized with the rising edge of the address clock. When the signal “K” is LOW, the data received from the QDR SRAM banks are placed into the first 36-bits of the output data. When the signal “K” is HIGH, the data received is placed into the last 36-bits of the output data.

Since this QDR interface controls two QDR SRAM banks simultaneously, the data from the first QDR bank is placed into the first 72 bits of the output data and the data from the second QDR SRAM bank is placed into the last 72 bits.

## Flags and Commands

When the “PLL” signal is HIGH, it indicates that a PLL lock is achieved and it is ready to transfer data to the QDR SRAM. Any transfer of data prior to the lock will result in data being lost.

The “cmd” signal controls the read and write enable for the QDR SRAMs. When cmd (0) is equal to “0”, it indicates read is enabled. When cmd (1) is equal to “0”, it indicates that write is enabled.

“BWS” is the bit wise select of the QDR SRAMs. It selects which set of bits are to be read. BSW(0) controls the first 9-bits of the 18-bit data going in to each of the QDR SRAMs. The next 9 bits is controlled by BSW(1). When “BSW” is set

to LOW, all 144 bits of data will flow as if the “BSW” control is not present at all.

## Timing

All CY7C1302 signals are registered in the I/O buffers and use HSTL buffers. For the write-cycle timing, all signals must meet the set-up and hold time requirements of the QDR SRAM. The timings that are relevant to the write cycle time are: the propagation delays from the Delta39K (Clock to Output), the board wiring delay, and the QDR memory set-up timing. Those delays must total less than the cycle time of the write operation:

Tco(FPGA) + Tpd(Board) + Tsu(QDR SRAM) < Tcyc(write cycle)

5.8 ns + 0.6 ns +0.8 ns = 7.2 ns < Tcyc (7.5 ns)

The clock-to-out and QDR set-up time values are 2.5 ns and 0.8 ns respectively. So there is a good margin for board delay. The QDR memory has a hold-time requirement of 0.5 ns.

During the read cycle, data must meet the setup and hold time of the FPGA:

Tco (QDR SRAM) + Tpd (board) + Tsu (Delta39K) < Tcyc (read cycle)

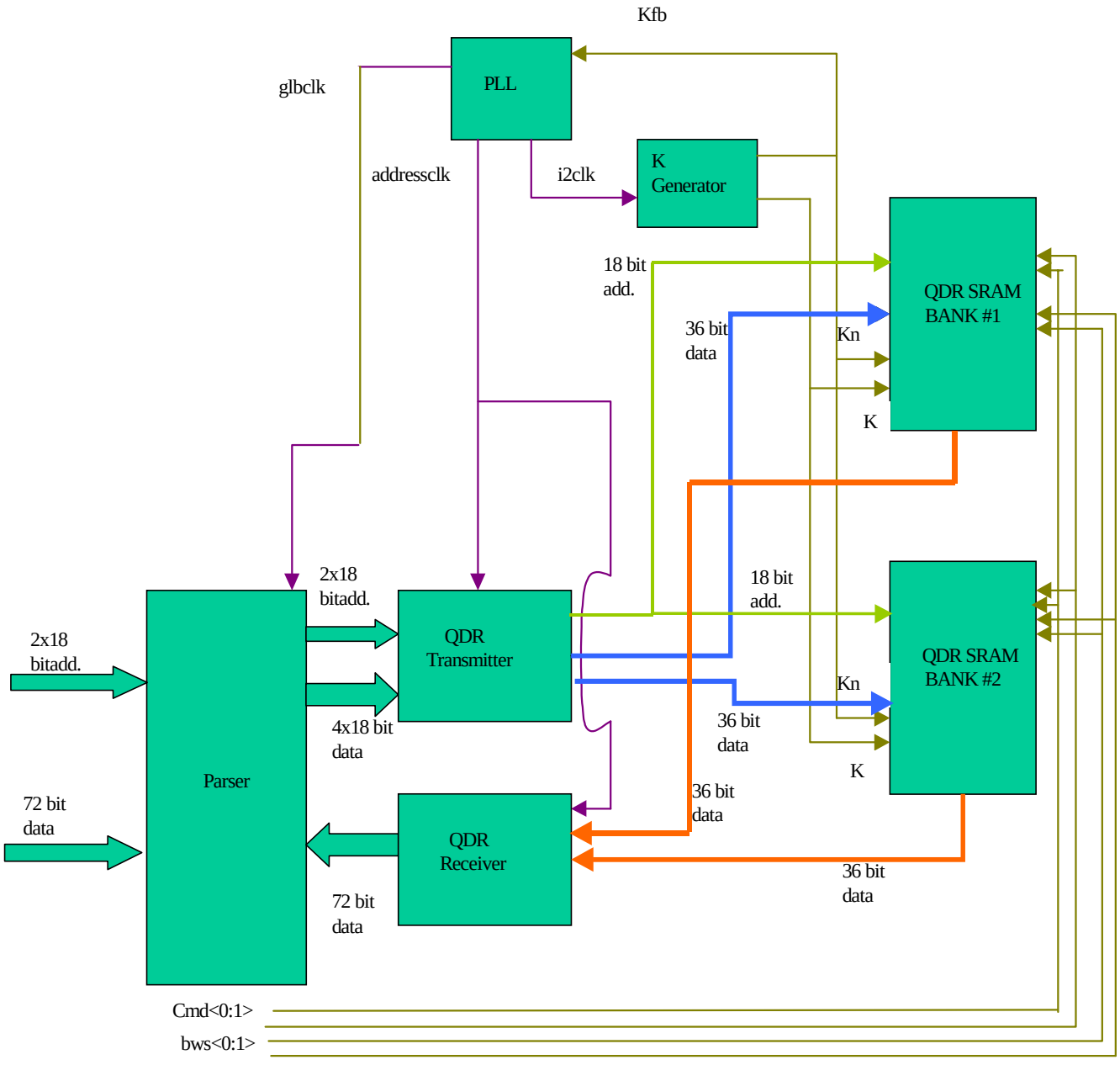
2.5 ns + 0.6 ns + 3.2 ns = 6.3 ns < Tcyc (7.5 ns)

The set-up time requirement for this implementation is 3.2 ns. Along with a clock-to-out timing on the QDR SRAM of 2.5 ns, this demand permits a good margin for operation at 66 MHz.

## Conclusion

The QDR controller is optimized to interface the QDR with the Delta39K CPLD. This Controller allows operation at 66 Mhz with the 39K CPLD. This solution provides a data rate of 133 Mhz. Further improvements to the controller are ongoing, please contact Cypress Applications for more details on the latest controller.

## Appendix 1: Logic Block Diagram of Interface;



## Appendix 2. VHDL

```
--*************************************************************************
**
--*************************************************************************
**
-- This model is the property of Cypress Semiconductor Corp. and is
protected**
-- by the US copyright laws, any unauthorized copying and distribution
is    **
-- prohibited. Cypress reserves the right to change any of the func-
tional    **
-- specifications without any prior notice.Cypress is not liable for any
**
-- damages which may result from the use of this functional model.
**
--                    **
--                    **
-- File name:QDR_39K_interface.vhdl     **
--                    **
-- Date         :March 08, 2001     **
--                    **
-- Model        :QDR - 39K Interface **
--                    **
-- Revision  :New     **
--                    **
-- Queries   :MPD Applications     **
--               Ph#: (408)-943-2821     **
--               e-mail: mpd_apps@cypress.com     **
--                    **
-- Comments: This is a functional model with most of the timings**
--                 and closely emulates the actual device. Select the **
--                 timing bin which you use from the below table.     **
--                 The model would still function even if the timings **
--                 are violated, but these are notified.          **
--*************************************************************************
**
--*************************************************************************
**
--4 QDR Controller
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE WORK.pll.all;
--LIBRARY cypress;
--USE cypress.rtlpkg.all;

ENTITY qdr_intf IS
  PORT(       clk     : IN  std_logic;              -- CLK:  System Clock
LVTTL
              rst     : IN  std_logic;              -- RST:  System Reset
LVTTL
              dll     : OUT std_logic;              -- dll: dll Status        LVTTL
-- FPGA Interface Signals
                 cmd     : IN  std_logic_vector( 1 DOWNTO 0); -- CMD:
Access control word    LVTTL
              bws       : IN  std_logic_vector( 3 DOWNTO 0); -- BWS:
Bytewide select        LVTTL
              rad       : IN  std_logic_vector(17 DOWNTO 0); -- RAD:
Read Address           LVTTL
              rda     : OUT std_logic_vector(143 DOWNTO 0); -- RDA:
Read Data              LVTTL
                wad       : IN  std_logic_vector(17 DOWNTO 0); --
WAD:  Write Address          LVTTL
              wda     : IN  std_logic_vector(143 DOWNTO 0); -- WDA:
Write Data             LVTTL
              rdy       : OUT std_logic;                  -- RDY:  Requested
Data Ready   LVTTL
-- QDR SRAM Address, Data and Control Signals
              k : OUT std_logic;
              kn    : OUT std_logic;                    -- K/KN: QDR SRAM
2-Phase Clock HSTL
              kfb       : IN  std_logic;                  -- KFB:  QDR Clock
dll Feedback HSTL
              SRAM1_dq  : OUT std_logic_vector(35 DOWNTO 0); --
DQ:   Data to QDR SDRAM1      HSTL
              SRAM1_di : IN std_logic_vector(35 DOWNTO 0); -- DI:
Data from QDR SRAM1     HSTL
              SRAM2_dq  : OUT std_logic_vector(35 DOWNTO 0); --
DQ:   Data to QDR SDRAM2      HSTL
              SRAM2_di : IN std_logic_vector(35 DOWNTO 0); -- DI:
Data from QDR SRAM2     HSTL
              ad  : OUT std_logic_vector(17 DOWNTO 0); -- AD:   Ad-
dress to QDR SRAM    HSTL
              SRAM1_bwsn: OUT std_logic_vector( 1 DOWNTO 0); --
BWSN: Bytewide Select SRAM1   HSTL
              SRAM2_bwsn: OUT std_logic_vector( 1 DOWNTO 0); --
BWSN: Bytewide Select SRAM2   HSTL
              wpsn: OUT std_logic;                  -- WPSN: Write Port
Select      HSTL
              rpsn: OUT std_logic
              );

END qdr_intf;

ARCHITECTURE cypress OF qdr_intf IS
              constant zero : std_logic_vector (17 DOWNTO 0) := (oth-
ers => '0');
              signal k_int: std_logic;
              signal i2clk, address_clk : std_logic;
              signal SRAM1_inputdata,SRAM2_inputdata:
std_logic_vector(35 DOWNTO 0);
              signal SRAM1_inputdata_1,SRAM2_inputdata_1:
std_logic_vector(35 DOWNTO 0);
              signal write_address_sync, read_address_sync :
std_logic_vector(17 DOWNTO 0);
              signal datastore_clk : std_logic;
              signal glbclk: std_logic;
              signal rdaint_hi : std_logic_vector(71 downto 0);
              signal rdaint_lo : std_logic_vector(71 downto 0);
              signal rdaint_tsu_SRAM1,rdaint_tsu_SRAM2:
std_logic_vector(35 downto 0);
              signal sram1_dq_tco, sram2_dq_tco:
std_logic_vector(35 downto 0);
              signal sram1_dq_tco2,sram2_dq_tco2:
std_logic_vector(35 DOWNTO 0);


BEGIN

              process (i2clk, rst,k_int)
              BEGIN

-- Rising edge of K and Kn being initialized from the rising edge of i2clk


              if rst = '1' then
              k_int <= '0';

              elsif (i2clk'event and i2clk = '1') then
              k_int <= not k_int;
              k <= not k_int;
              kn <= k_int;
              end if;
               end process;



dll1: cy_c39kpll
              -- changed frequency output for i2clk,addressclk and gl-
bclk
              generic map(
              feedback       => EXTERNAL,-- optional
              multiply       => 2,-- optional
              gclk0_phase    => 0,-- optional
              gclk0_divide   => 1,-- optional
              gclk1_phase    => 45,-- optional
              gclk1_divide   => 1,-- optional
              gclk2_phase    => 0,-- optional
              gclk2_divide   => 2,-- optional
              gclk3_phase    => 0,-- optional
              gclk3_divide   => 1 -- optional
              )
              port map(
              pll_in         => clk,
              ext_fdbk       => kfb,-- optional
              lock_detect    => dll,-- optional
              gclk0          => i2clk,-- optional
              gclk1          => address_clk,-- optional
```

```
                gclk2       => glbclk,-- optional
                gclk3       => datastore_clk -- optional
                );

 -- ---------------------------------------------- --
 -- QDR SRAM Signals not used                    --
 -- ========================                     --
 -- 1. C and C_bar should be tied to Vdd placing  --
 --    the QDR SRAM in single clock mode         --
 --                                              --
 -- CMD Bit Assignment                           --
 -- ==================                           --
 -- CMD(0) 0: Read  1: No operation on read port  --
 -- CMD(1) 0: Write 1: No operation on write port --
 -- ---------------------------------------------- --

-- Byte Write Enable Control Signal for SRAM bank 1 and 2
                SRAM1_bwsn(0) <= bws (0);
                SRAM1_bwsn(1) <= bws (1);
                SRAM2_bwsn(0) <= bws (2);
                SRAM2_bwsn(1) <= bws (3);

                -- the data storing device for clk
                process (glbclk)
                BEGIN

                if rising_edge (glbclk) then
                  write_address_sync <= wad;
                read_address_sync <= rad;
-- Data written on to the  36 bits data buses. On every rising edge of
K and Kn the data on two buses is sent out to the 2 SRAM banks

                  SRAM1_inputdata <= wda (143 downto 108);
                SRAM1_inputdata_1 <= wda (107 downto 72);


                SRAM2_inputdata <= wda (71 downto 36);
                SRAM2_inputdata_1 <= wda (35 downto 0);


                  end if;
                END process;

-- Write and Read enable signals being assigned Common to all
SRAM's

                wpsn <= cmd(1);
                rpsn <= cmd(0);

--Reading and Writing DAta in to SRAM
                process (address_clk)
                BEGIN

                if rising_edge (address_clk) then
                -- setting for the time just before k rises
                if k_int = '0' then -- that means k is turning to '0' when
process is over

-- Sending read address on rising edge of K

                ad <= read_address_sync;

-- Sending higher order byte of Write Data to Write Data register

                SRAM1_dq_tco <= SRAM1_inputdata;
                SRAM2_dq_tco <= SRAM2_inputdata;

                -- setting for the time just before k falls
                elsif k_int = '1' then

-- Sending write address on rising edge of K

                ad <= write_address_sync;

-- Sending lower order byte of Write Data to Write DAta register

                SRAM1_dq_tco <= SRAM1_inputdata_1;
                SRAM2_dq_tco <= SRAM2_inputdata_1;
                end if;
                end if;
                end process;

--extra cycle of latency to decrease tcs on the CPLD

                process (address_clk)
                BEGIN
                if (address_clk'event and address_clk = '1') then

-- Read data being assigned to registers to give extra cycle of latency

                rdaint_tsu_SRAM1 <= SRAM1_di;
                rdaint_tsu_SRAM2 <= SRAM2_di;

-- Write DAta given to the two SRAM banks from the Write data register

                SRAM1_dq_tco2 <= SRAM1_dq_tco;
                SRAM2_dq_tco2 <= SRAM2_dq_tco;

                SRAM1_DQ <= SRAM1_dq_tco2;
                SRAM2_DQ <= SRAM2_dq_tco2;

                end if;
                 end process;


                process (address_clk, k_int)
                BEGIN
                if (address_clk'event and address_clk = '1') then
                 if  (k_int = '0') then
                -- that means k is turning to '1' when process is over
D(A+1)

-- Read DAta being assigned to another set of registers on rising edge
of K

                rdaint_hi(71 downto 36) <= rdaint_tsu_SRAM1;
                rdaint_hi(35 downto 0) <= rdaint_tsu_SRAM2;

                end if;
                end if;
                 end process;



                process (address_clk, k_int)
                BEGIN
                if (address_clk'event and address_clk = '1') then
                if (k_int = '1') then
                -- then k is going to be 0  D(A)

-- Read DAta being assigned to another set of registers on rising edge
of Kn

                rdaint_lo(71 downto 36) <= rdaint_tsu_SRAM1;
                rdaint_lo(35 downto 0) <= rdaint_tsu_SRAM2;

                end if;
                end if;
                 end process;



                process (i2clk)
                BEGIN

                if rising_edge (i2clk) then
                -- that means k is turning to '1' when process is over
D(A+1)

-- Read DAta assigned to the CPLD read data output from the read
data registers

                if k_int = '0' then

                rda (143 downto 108) <= rdaint_hi(71 downto 36);
                rda (107 downto 72) <= rdaint_lo(71 downto 36);
```

```
            -- setting for the time just before k falls

            elsif k_int = '1' then  -- then k is going to be 0  D(A)

            rda (71 downto 36) <= rdaint_hi(35 downto 0);
            rda (35 downto 0) <= rdaint_lo(35 downto 0);

            end if;

            end if;
             end process;

end cypress;
```

QDR, Delta39K, and NoBL are trademarks of Cypress Semiconductor Corporation.
ZBT is a trademark of IDT.