X32 cache memory and SDRAM controller

Matías Escudero Martínez

April 28, 2010

1 Introduction

This document describes the design and implementation of a memory module X32 softcore [1]. The final goal is to have the X32 running on the Trenz Electronic TE300 board [2]. This board has a Xilinx Spartan3E FPGA [4][5] connected to a DDR SDRAM chip MT46V32M16 [3] SDRAM memories are designed for high throughput, but the optimal performance only can be achieved when transferring big amounts of data. For example, accessing only one word takes around 20 cycles, but each extra word only takes one cycle more. This huge overhead forces to use a cache memory between the X32 and the SDRAM. This architecture also is more suitable for dealing with the different clock domains that the SDRAM interface requires for normal operation.

2 Cache organization

The amount of RAM available on the FPGA is 72KByte. Since most of the programs that are going to be loaded on the X32 are in the range of 10 to 32 KByte, plus the stack and some variables, we have decided to use a 64KByte direct mapped cache.

The value of the block size is coded with generics, so can be change any time by synthesizing again the softcore. Starting value is 32 words. That means 128 bytes. Writing or reading 32 words to or from the SDRAM takes 32 cycles plus the overhead, approximately 18 cycles more. Since the SDRAM works with 100MHz clock and the X32 with 50MHz clock, a cache miss is between 25 and 50 cycles (if writing the block back in the SDRAM is needed), from the microcontroller point of view. This means 1us delay.

Then the number of sets will be:

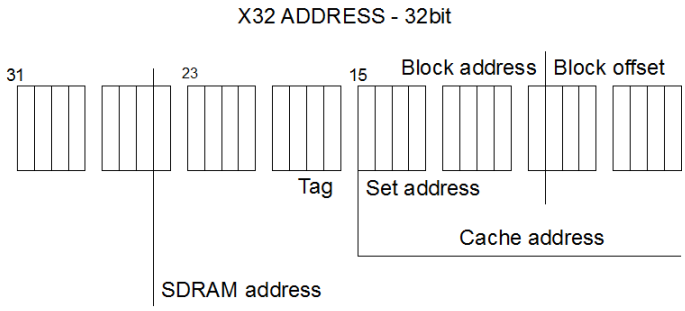

Figure 1: Cache addressing scheme.

$$N_{sets} = \frac{cache\_size}{block\_size} = \frac{2^{16}}{2^7} = 2^9$$

In order to know which block is loaded in each set of the cache, a Translation Lookaside Buffer (TLB) is needed. Such TLB has as many positions as number of sets in the cache. The width of each position is the width of the tag, that identifies the block, plus the dirty bit, that tells whether the block loaded in the cache was modified or not.

3 System overview

The X32 is connected to the memory module through the normal X32 memory bus [1]. The cache controller process the memory request and checks in the TLB whether the block corresponding to the requested address is loaded in the cache or not. Most of the times the block will be already on the cache and the cache controller will read or write the data with the correct alignment. The write operations also require to set the dirty bit to 1 in the TLB address corresponding to the written memory address. If the block is not in the cache, the cache controller must issue a cache block request to the sdram controller.

The SDRAM controller module will read in the TLB the dirty bit of the set corresponding to requested block. If the block that is loaded on the cache was modified, the SDRAM controller writes the block back to the memory. After that the SDRAM controller reads the requested block from the SDRAM memory and loads it on the cache. Finally, the tag of the new memory address is stored in the TLB.

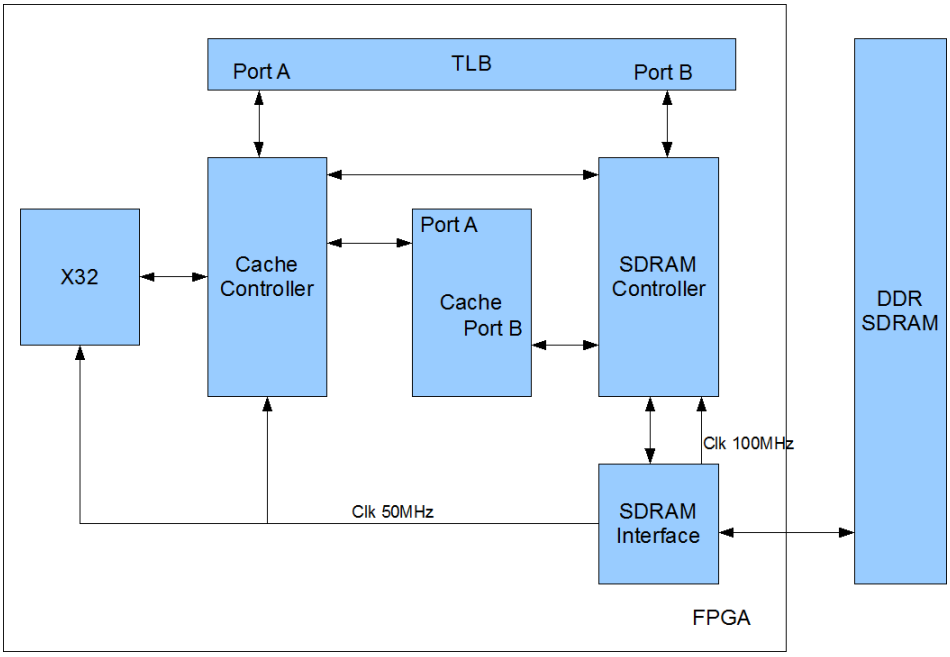

Figure 2: Cache Block Diagram.

4 Cache description

The cache is made out of four BRAMs. Each address of each bram contains one of the bytes of each word for the corresponding address. Thus, the cache can handle words of 32 bits while supporting writing masks.

5 Cache controller

The cache controller reads or writes the data in the cache following the orders from the X32. When the X32 issues a read operations, the address is stored. Then the cache controller goes to the SET_ADDRESS state. There it enables the cache and sets the address in the cache and puts the set bits in the address of the TLB. In the next clock cycle the controller goes to the READ_FIRST_WORD state and sets the address+1 in the cache and set bits corresponding to this new address in the TLB. If the tag that comes out of the TLB corresponds with the tag of the first address, then the cache request is not issued. If a second word is needed it moves to the READ_SECOND_WORD state, that is similar to the previous. In These

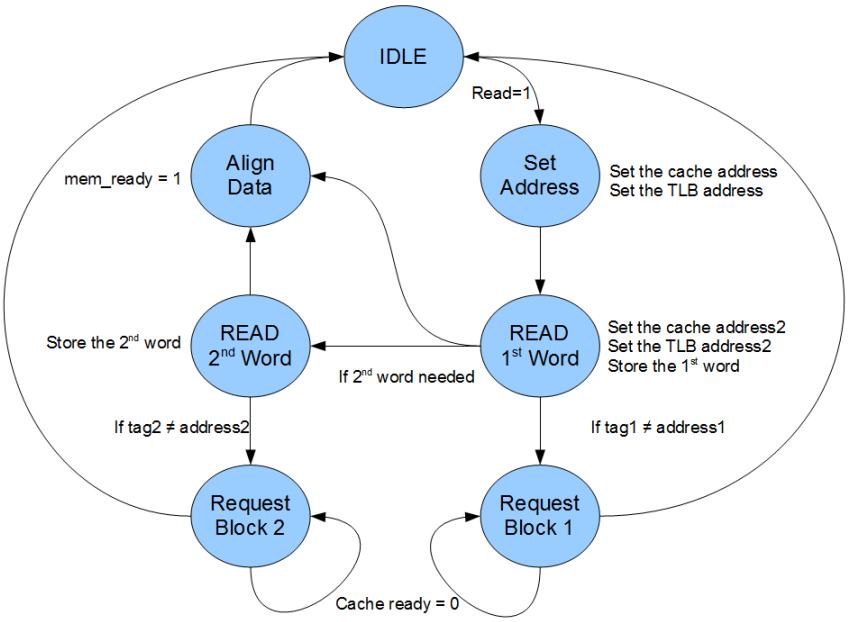

Figure 3: Read state machine.

states the first word and the second word are stored as input of the memory decoder module. This module aligns the data and takes care of the size of the memory operation (1, 2 or 4 bytes) and of the sign.

As seen, the actions of reading from the cache and reading from the TLB are done in parallel. For writing operations this is not possible, so those actions will be pipelined. When the X32 issues a write operation, the address is stored and the controller moves to the next state. In the ALIGN_DATA state, the memory decoder splits the word to be written in two words if needed. Those words will be written in consecutive memory addresses. In this state the controller also request the corresponding tag from the TLB. In the next state the output tag of the TLB is compared with the tag of the first address. If it is correct, the controller moves to the next state, where the first word is written in the cache. In this state the tag of the second memory address is also checked. If it is correct and writing a second word is needed, the controller moves to the next state, otherwise the write operation is finished and it goes to IDLE state.

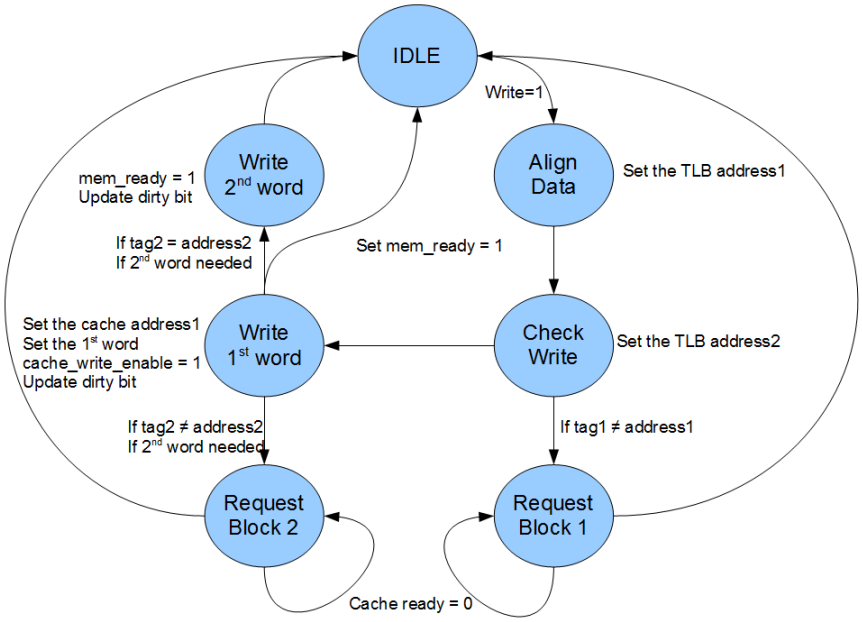

Figure 4: Write state machine.

6 SDRAM controller

The functionality of the SDRAM controller is quite simple. When a cache block request is issued, the controller looks at the TLB if the block that is already loaded was modified. In that case, the controller issues a write command and writes the first address of the block in the SDRAM interface. When the SDRAM interface asserts the command ack, the SDRAM controller writes the following addresses of the block, and the data. When it reaches the end of the block, the SDRAM controller asserts the burst done for one cycle and waits the command ack signal goes to zero. After that the controller starts a reading burst, that is the same as the previous one, but waiting for the data valid signal that comes from the interface. When this signal is asserted, the controller starts writing the incoming data in the cache.

The main complexity of this module is due to the different clocks. The SDRAM interface works on one side with the falling edge of a 100MHz clock for the commands and the address path. On the other side the data path works with the rising edge of the same clock shifted 90 degrees. Thus, some additional registers are needed for adapting this two clocks. For more details about the SDRAM interface look at [6]

7 Clock and reset implementation

Since the system uses two 100MHz clocks for the SDRAM interface and one 50 MHz clock for the X32 and the peripherals, a Digital Clock Manager (DCM) is needed for frequency synthesis. The DCM takes the clock signal coming from the outside of the FPGA (system clock or *sys_clk*) with a frequency of 100 MHz and generates 3 different clock signals: two 100 MHz clocks with 0 degrees phase (*clk0*) and 90 degrees phase (*clk90*), and the 50 MHz clock (*clk50MHz*). *clk0* and *clk90* are required for the SDRAM interface and controller but all the other parts of the system work with *clk50MHz*. The DCM *locked* signal is asserted when the output clocks are engage with the input clock, both in phase and frequency. When the FPGA is programmed, this signal is zero and it can be used for system reset. *reset* signal is active when the DCM is not locked, and 200us after the DCM locks the phase. Thus, no hardware reset is available for the user.

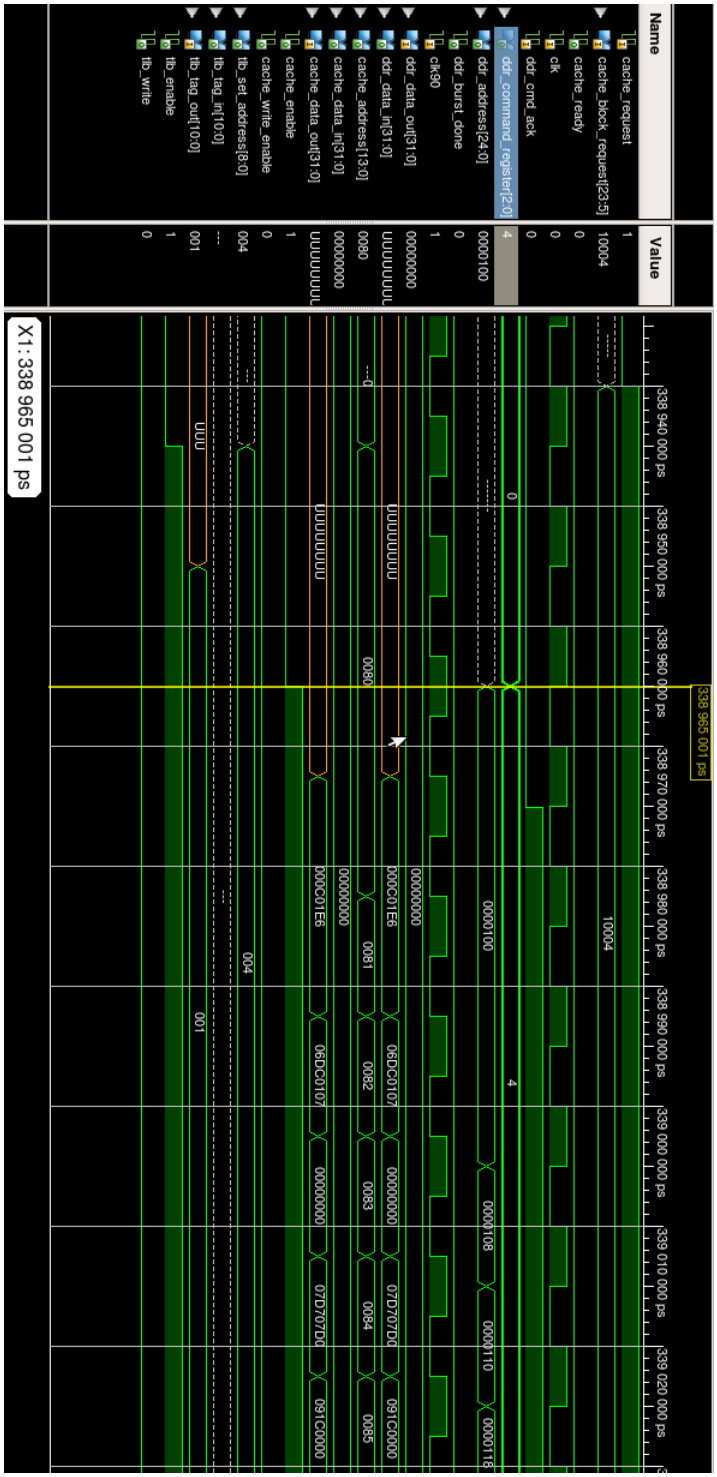

Figure 5: The SDRAM controller starts writing to SDRAM after the cache request.

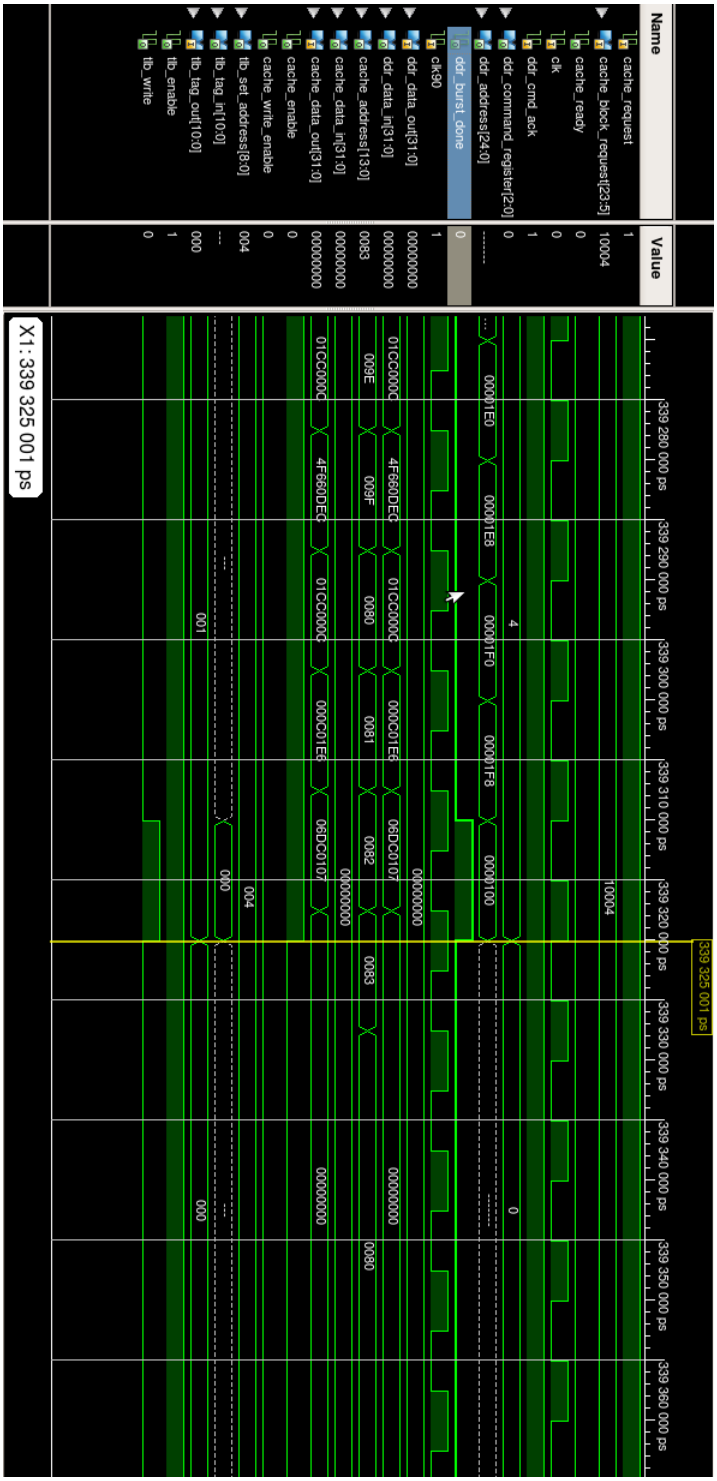

Figure 6: End of the writing operation.

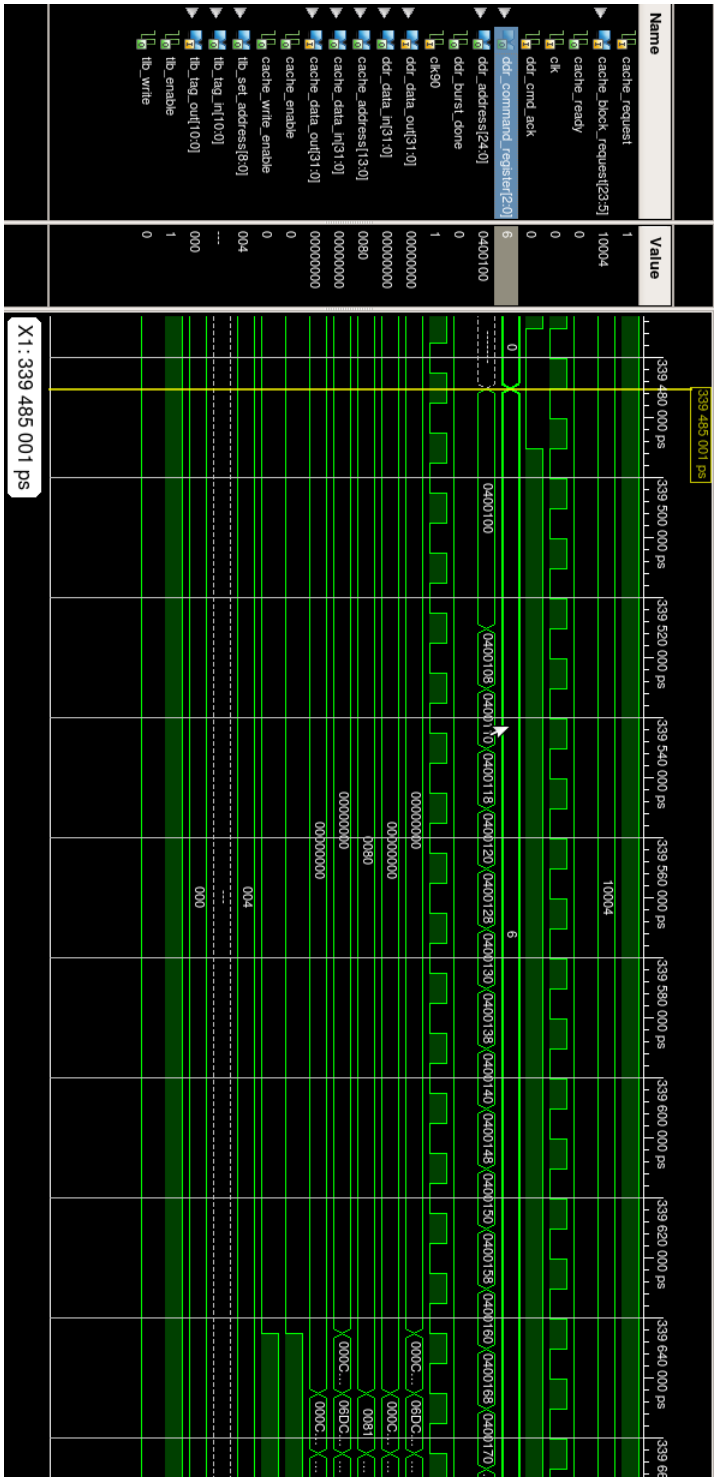

Figure 7: SDRAM controller starts reading from the SDRAM.

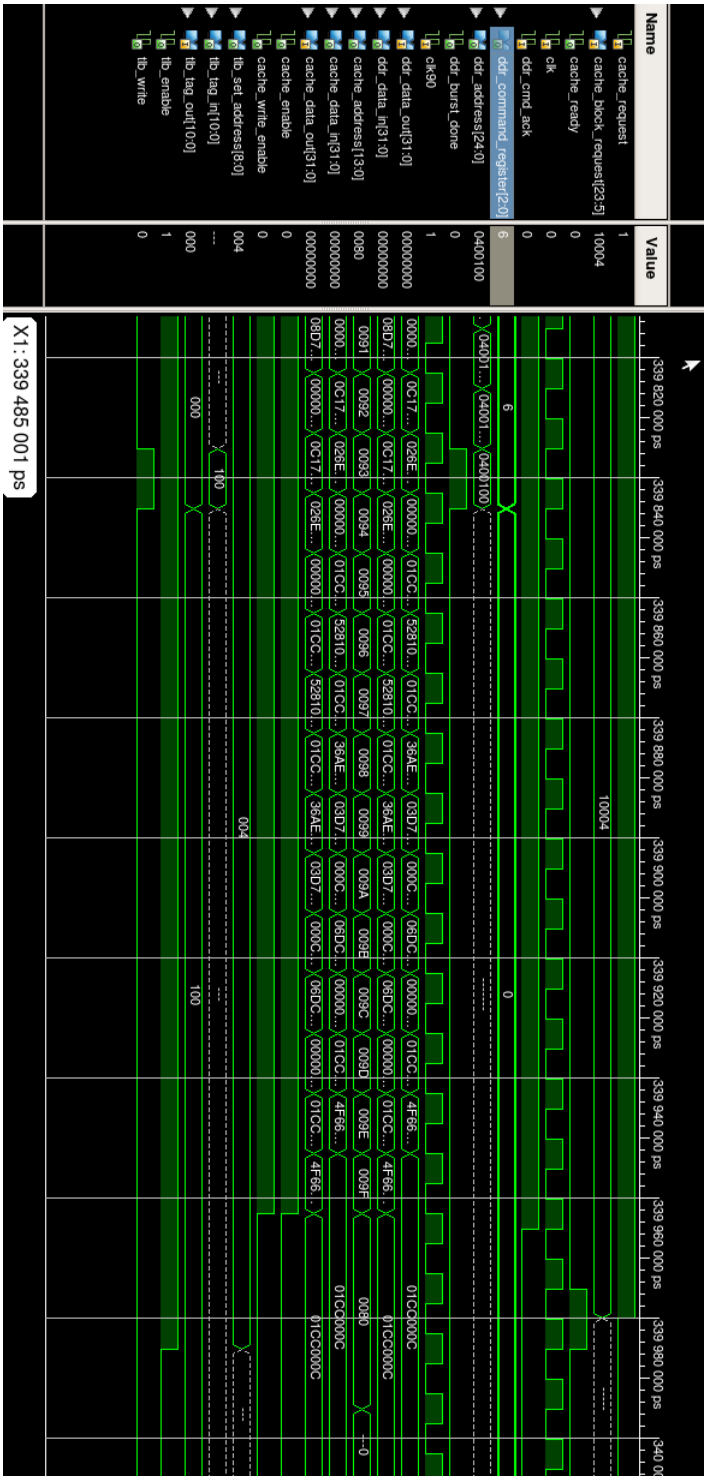

Figure 8: End of the reading operation.

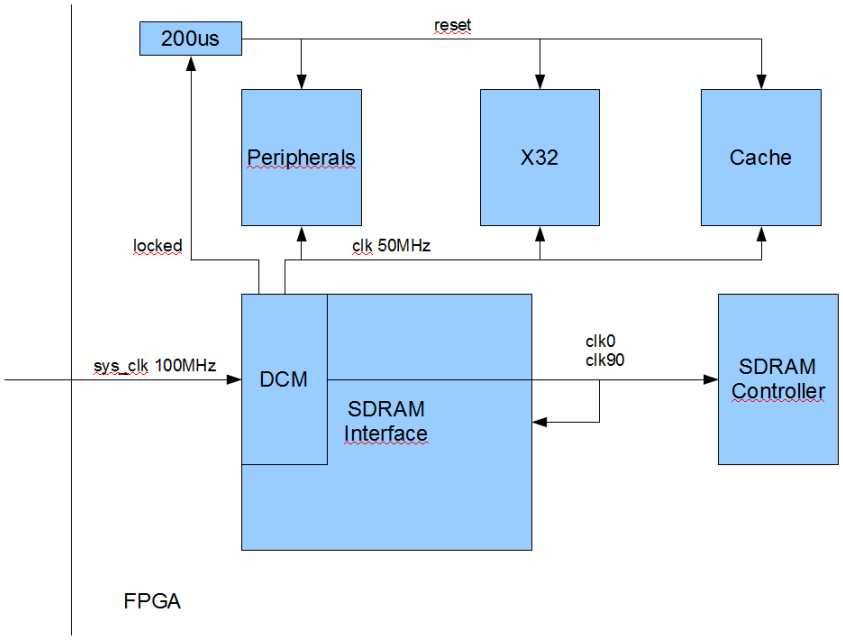

Figure 9: Clock and reset tree.

8 Bootloader

In the previous versions of X32 the bootloader program was stored in a ROM, that actually was a Block RAM with write port unconnected. For the new X32 memory system we have decided to do without the ROM and spend almost all the BRAM blocks in the cache. The bootloader program is stored in the cache when the FPGA is programmed, but once the bootloader is modified it will not be possible to retrieve the original code. The only reliable way to reload the bootloader is by reprogramming the FPGA.

A simple hardware reset would not be a guarantee of relaible behaviour because the bootloader might be overwrited or corrupted. This is the reason why this kind of reset is not implemented on TE300 board.

9 Implementation

Device Utilization after Map stage

```
Number of BUFGMUXs                        3 out of 24     12%
Number of DCMs                            1 out of 8      12%
Number of MULT18X18SIOs                   4 out of 36     11%
Number of RAMB16s                        33 out of 36     91%
Number of Slices                       3123 out of 14752  21%
   Number of SLICEMs                    120 out of 7376    1%

Number of LOCed Slices                    65 out of 3123    2%
   Number of LOCed SLICEMs                43 out of 120    35%


Logic Utilization:
  Number of Slice Flip Flops:         2,496 out of  29,504    8%
  Number of 4 input LUTs:             4,464 out of  29,504   15%

Logic Distribution:
  Number of occupied Slices:          3,123 out of  14,752   21%
    Number of Slices containing
                only related logic:   3,123 out of   3,123  100%
    Number of Slices containing
                unrelated logic:          0 out of   3,123    0%

  Total Number of 4 input LUTs:       4,818 out of  29,504   16%
    Number used as logic:             4,254
    Number used as a route-thru:        354
    Number used as 16x1 RAMs:            64
```

```
Number used for Dual Port RAMs:      64
  (Two LUTs used per Dual Port RAM)
Number used as Shift registers:      82
```

Timing results after Place and Route:

Constraint	Period Requirement	Actual Period		Timing Errors		Paths Analyzed	
		Direct	Derivative	Direct	Derivative	Direct	Derivative
sys_clk_in	10.000ns	4.800ns	9.988ns	0	0	0	5072657
clk_100MHz	10.000ns	9.918ns	N/A	0	0	2809	0
clk_100MHz_90	10.000ns	9.774ns	N/A	0	0	1245	0
clk_50MHz	20.000ns	19.975ns	N/A	0	0	5068603	0

10 Results

Thanks to this cache most of the times a memory access only takes between 4 or 5 cycles. In comparison, we have achieved an speedup of 5x compared with the version without cache, and an speedup of 1.6x compared with the version of the NEXSYS board. The main drawback is that is not possible to have a bootloader on the FPGA, and the monitor program is now loaded in the cache. If the monitor is overwritten at any time, the only way to retrieve it is by reprogramming the FPGA.

References

[1] X32 Design. http://x32.ewi.tudelft.nl/woutersen_thesis.pdf.

[2] TE300 board documents: http://www.trenz-electronic.de/support/download-area/te0300-spartan-3e-series.html

[3] Micron MT46V32M16 description: http://download.micron.com/pdf/datasheets/dram/ddr/512MBDDRx4x8x16.pdf

[4] Xilinx Spartan3E datasheet: http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf

[5] Xilinx Spartan3E user guide: http://www.xilinx.com/support/documentation/user_guides/ug331.pdf

[6] SDRAM interface generated by MIG 2.3: http://www.xilinx.com/support/documentation/ip_documentation/ug086.pdf