Digilent Nexys-3 Cellular RAM Controller Reference Design Overview

General Overview

This document describes a reference design of the Cellular RAM (or PSRAM – Pseudo Static RAM) controller for the Digilent Nexys-3 development board with Spartan-6 FPGA. The reference design at hand is a video pipeline with a soft-core processor control. The video part of the design displays an 800x600 pixel image stored in the PSRAM. The processor has full access to the PSRAM memory, as well as to PS/2 keyboard interface and 7-segment LED indicators.

The aim of this design is to demonstrate the usage of the PSRAM memory on a specific development board. It is not intended to be educational in any way, nor does it pretend to be very efficient or fault-free, again, in any way.

Design features:

- VESA 800x600 @ 60Hz video standard, 8-bit colour depth
- Synchronous single and burst PSRAM access with fixed latency @ 50MHz
- Dual-port frontend for the PSRAM memory controller
- picoBlaze soft-core processor
- PS/2 keyboard interface with software access
- 7-segment LED driver with software access
- Verilog HDL (PS/2 interface is in VHDL)

Functional overview

This design is comprised of three distinct parts: the PSRAM memory controller (with a separate dual-port frontend), the video pipeline (with VGA timing generator) and the picoBlaze soft-core processor (with peripherals):

- The memory controller provides a simple asynchronous memory access to the application by taking care of all timing and signalling considerations. The attached dual-port frontend allows two separate applications to use the memory controller without considering each other (read further to see how this can actually negatively affect the result).
- The video pipeline reads portions of image from PSRAM memory and buffers them in local block-RAM (BRAM) memory. The VGA timing generator reads the pixel data from this buffer and sends them to VGA screen. True dual-port nature of BRAM allows these two components to work simultaneously.
- The processor can access (read and write) any memory location, hence it has the ability to read and modify the image stored in the PSRAM. The processor also has access to two peripheral devices – the PS/2 keyboard interface and 7-segment LED indicators.

Detailed overview

Video pipeline

The video standard used in this design is VESA 800x600 @ 60Hz. It means the image size is 800 by 600 pixels, and the image on the screen is refreshed 60 times per second. It provides a good enough

resolution and quality for many applications. This standard requires, that the pixel clock frequency would be 40MHz, which is a very convenient number because the on-chip digital clock manager (DCM) can produce this frequency (exactly) from the reference 100MHz, provided on-board.

The external PSRAM memory is of a pseudo-static type, capable of operating asynchronously. Its access interface in this mode is very simple; however the access time is at least 70ns, which is equivalent to just over 14MHz. Even though two "pixels" (assuming 8-bit colour depth) can be stored in each memory word, it is by far insufficient to match the 40MHz pixel clock requirement of the VGA timing generator. The memory is therefore used in synchronous mode, in which it can operate at a frequency of up to 80MHz.

In synchronous mode the memory can be accessed (read or written) either in single on burst modes, where single-access is a variation of burst-access with burst length equal to one. Burst mode allows to access up to 128 memory cells sequentially – one cell per clock cycle, without having to reinitiate the operation. The first address is provided during read or write request; it is then being incremented automatically by the memory.

Although all PSRAM memory cells are addressed linearly, the memory is arranged in rows; each row contains 128 cells. Burst-access can only last until the end of a row, irrespectively of where it started. If the application is continuing a burst-access over the end of a row, the operation will abort at the end of the row and a new operation will have to be initiated to continue. The PSRAM controller provided in this design does not check for the end-of-row condition. It is up to the application to make sure that burst-accesses do not exceed ends of rows. Failure to do so will result in the controller entering an undefined state, from where it may possibly not recover unless restarted.

Each memory word is 16 bits wide. The Nexys-3 board has only 8 colour signals (3 red, 3 green and 2 blue), so it is rational to store two pixel colour values in every memory cell. Such arrangement also allows speeding up reads and writes if operating on 8-bit data. However, computing pixel addresses becomes slightly more complex. In addition, writing a single 8-bit datum is not so straightforward any more. There are two options to do this (actually, you would always choose just one of them):

- first read a memory cell, update the corresponding byte and write it back
- use additional LB (lower byte enable) and UB (upper byte enable) memory signals to indicate which byte you wish to write

Even with burst-access it is not possible to read the memory continuously indefinitely. We must, however, provide an uninterrupted stream of pixel data to the VGA generator. One of the ways to do so is to read the memory on average at a faster rate than the VGA generator reads pixel colour values. In this case a buffer is required between the memory and the VGA generator. This scheme is implemented in this design. The memory frontend of the video pipeline is reading the memory in burst mode at a frequency of 50MHz, two "pixels" at a time, and stores them into on-chip block-RAM memory. The VGA frontend of the video pipeline is using the second port of BRAM buffer to read pixel colour values at a frequency of 40MHz, two at a time, and sends the corresponding byte to the screen.

In order for this arrangement to run properly, the whole buffer is filled with pixel data at system start-up. Then every time the VGA frontend of the video pipeline reads the middle or the last word

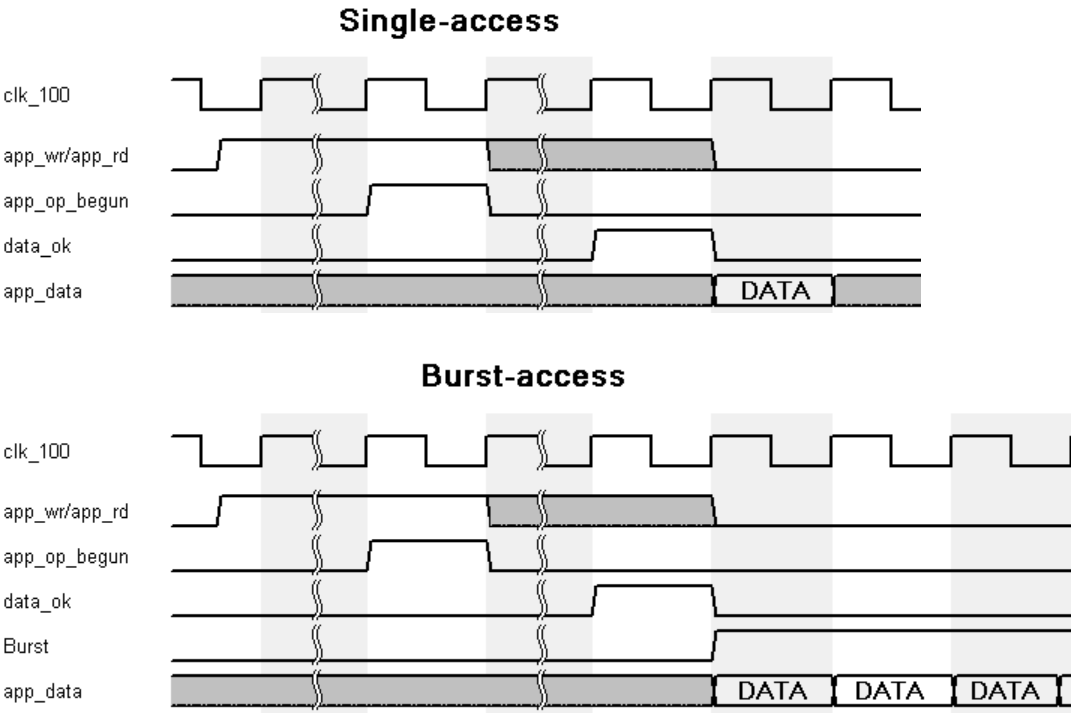
from the buffer, the memory frontend initiates a burst read to fill the recently read half of the buffer with fresh pixel colour values. The address counters of the two frontends are synchronized, and because the memory frontend is operating at a faster frequency than the VGA frontend (50MHz vs. 40Mhz) and is fetching two pixel colour values at a time, this arrangement functions correctly and leaves certain time slots for the second application (the processor) to access the memory.

As mentioned previously, the image size is 800 x 600 pixels with an 8-bit colour depth. Each memory cell contains colour data for two pixels. The video pipeline is reading the memory starting from address 0 to address (800 x 600 / 2 = 240000) in decimal. Make sure you set proper offsets when preloading the memory.

Memory controller

The dual-port frontend for the memory controller provided in this design has a very simple (and limited) principle of operation. If two applications (the processor and the video pipeline) are requesting memory access simultaneously, it will give priority to the video pipeline. No memory access will be given to any application until the current operation (if such exists) is finished. The user must make sure that the second application does not monopolize the memory by long subsequent burst operations, otherwise the video pipeline will starve for pixel colour data and the picture on the screen will degrade.

The memory controller provides a simple asynchronous memory access interface to the application. The application must issue a read or write request signal and the address and wait for “op_begun” signal, after which the application may take down the request signals and address. Operation is finished when “op_done” signal is asserted. In case of a read, data is available on the next clock cycle after “data_ok” assertion. In case of a write, data must be provided on the next clock cycle after “data_ok” assertion. If case of a burst-access the “burst” signal must be asserted after “data_ok” goes high and kept high until the end of burst. Of course, data must be read in or sent out every next clock cycle.



Processor application

From the software point of view, the processor has access to any memory location. It cannot be done with a single instruction, however. The program must provide a 23-bit address, a 16-bit data + 4 control signals (read strobe, write strobe, upper byte enable, lower byte enable) to the memory in order to access it. This is done by writing to certain registers which are associated with output port of the processor. The mapping is as follows:

Port ID	Register			
0	WR	data [7:0]	RD	data [7:0]
1	WR	data [15:8]	RD	data [15:8]
2	WR	address [7:0]	RD	
3	WR	address [15:8]	RD	
4	WR	address [22:16]	RD	
5	WR	control signals	RD	
6	WR	operation done	RD	operation done
7	WR	7-segment LEDs 1,2	RD	scan code
8	WR	7-segment LEDs 3,4	RD	

As soon as a read or write strobe bit appears in the control register, the FSM initiates the corresponding operation. These control bits are then cleared automatically. When the operation is complete, the FSM writes 0x01 to register “op_done”. This register must be cleared by the program. It may therefore take quite some time to access just a single memory cell – first, registers need to be loaded with the corresponding values (6 in the worst case), they need to be sent out (again, all 6 in the worst case). In case of a read operation one must also monitor the “op_done” register in a loop, and then read the data in from two external registers. Memory access latency must also be added. Therefore, from the software point of view, a read operation takes about 40 clock cycles to complete.

In a similar fashion, i.e. through registers mapped to processor ports, software can control the four 7-segment LED indicators. They are configured to support only hexadecimal digits.

The PS/2 interface controller constantly monitors keyboard activity, registers the message, extracts the information byte and generates an interrupt to the processor. When the processor receives the interrupt, the program can retrieve the code from port 7. When a key is pressed the keyboard transmits its respective code. When the key is released the keyboard transmits 0xF0 followed by the key’s code. For certain keys, called extended, 0xE0 precedes the keys’ codes. This means that the processor will receive 3 or 5 interrupts for each key. Situation will change if a single key is held down pressed or if a key is used in combination with another key, e.g. shift + a.

Known issues

1. There appears to be a 1 cycle data delay after the VGA generator is enabled. This is possibly due to the 16-bit -> 8-bit buffer register.
2. Sometimes certain pixels on the screen seem to blink or change colour. Dunno why.

Possible improvements

1. Improve dual-port frontend priority management.
2. Add end-of-row check for PSRAM controller.

Comments on implementation

1. The design was implemented using Xilinx ISE 12.3
2. In project file hierarchy you may see that module “BSCAN_BLOCK_inst” is missing. You may safely continue without it.
3. Synthesis will warn that “Input <instruction<0:11>> is never used”. This appears to be a bug but not in the design. You may safely ignore this warning.
4. Synthesis will warn that “Node <input_buffer_0> is unconnected”. This is a small coding issue. You may safely ignore this warning.
5. Implementation will warn that “read_strobe_flop” has unconnected output pin. This is because this pin is not used in this design. You may safely ignore this warning.
6. Implementation will warn that “k_write_strobe_flop” has unconnected output pin. This is because this pin in not used in this design. You may safely ignore this warning.
7. Implementation will warn that “interrupt_ack_flop” has unconnected output pin. This is because this pin is not used in this design. You may safely ignore this warning.

Feedback

Please send your feedback/bug reports to vadim.pesonen@ati.ttu.ee