ECE 532 – Digital System Design

Group Report

Picture Processing Unit of the Nintendo Entertainment System

Project Team: Ricky Iu <993298980>
Esther Li <993372355>

Date: March 31, 2008.

Table of Content

Section 1 – Project Description

The purpose of this project is to create a Picture Processing Unit (PPU). The PPU is designed based on the one used in the famous Nintendo Entertainment System (NES). The PPU would have features such as background and sprite rendering and scrolling. The job of the PPU is to render the video information in an NES game cartridge, or in this case the ROM images file, into RGB data.

There are several changes in the final design compare to the initial concept described in the proposal. First, the external memory blocks are no longer needed, as the RGB data stream is directly feed to the SVGA controller. As a result, the Bit-mapped mode of the SVGA controller is no longer used. Only the SVGA timing generator portion of the block is retained. Second, the arbiter for the BRAM between the Microblaze and the PPU is actually not necessary since the BRAM ram provided by Xilinx is dual ported. The Microblaze and the PPU can access the RAM separately. Lastly, a second block ram is added for storing sprite attribute, which include horizontal and vertical position and the tile number of all 64 sprites. Since there are two block rams, the original BRAM is renamed to VRAM, which stores the name table and pattern table, and the new RAM is named SRAM, for storing sprite data. The changes are shown in the diagrams below.

Section 2 – System Block Diagram

Section 2.1 – Initial system block diagram (copied from project proposal)

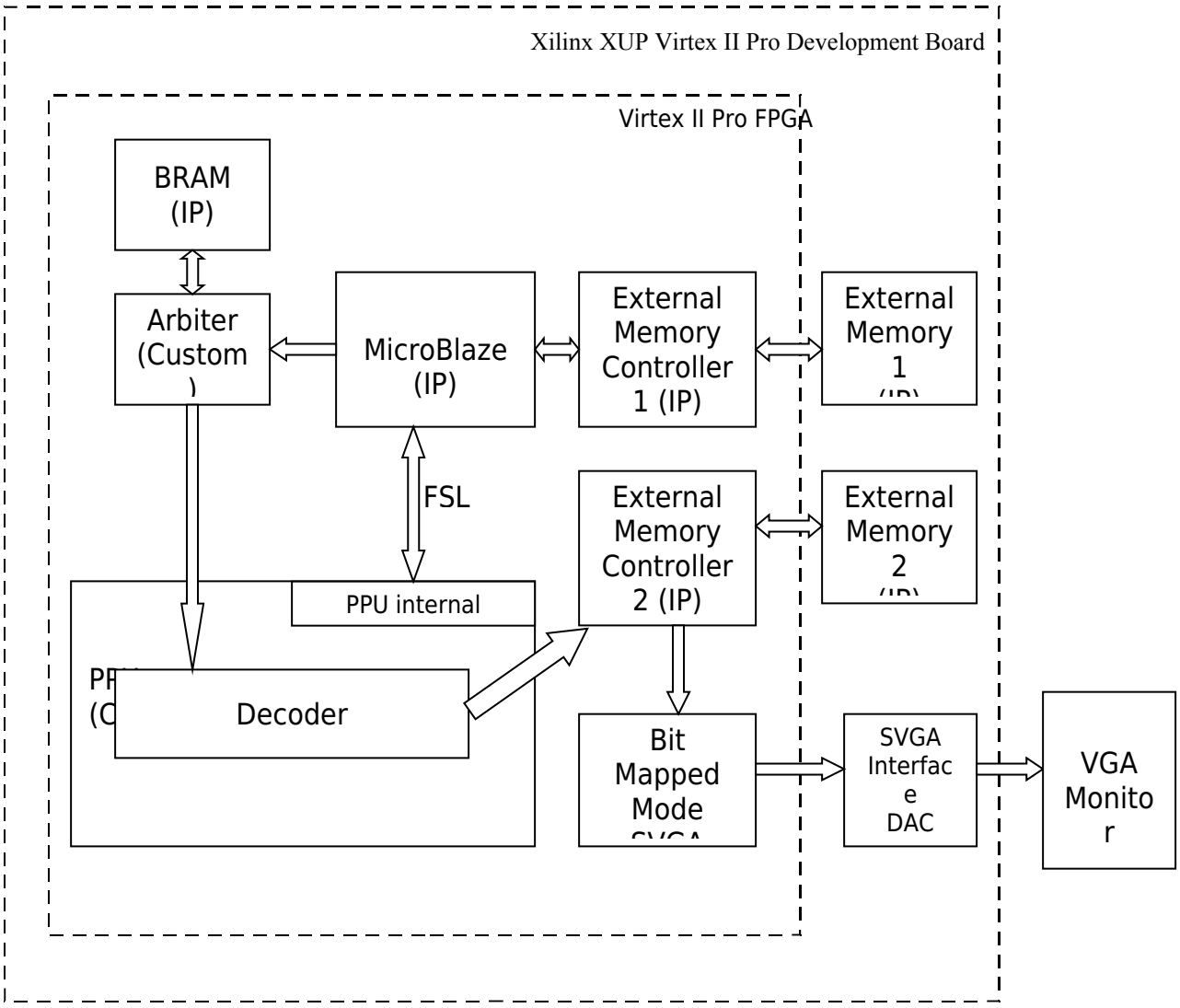

Fig 1.1 Proposed System Block Diagram

Section 2.2 – Final design system block diagram

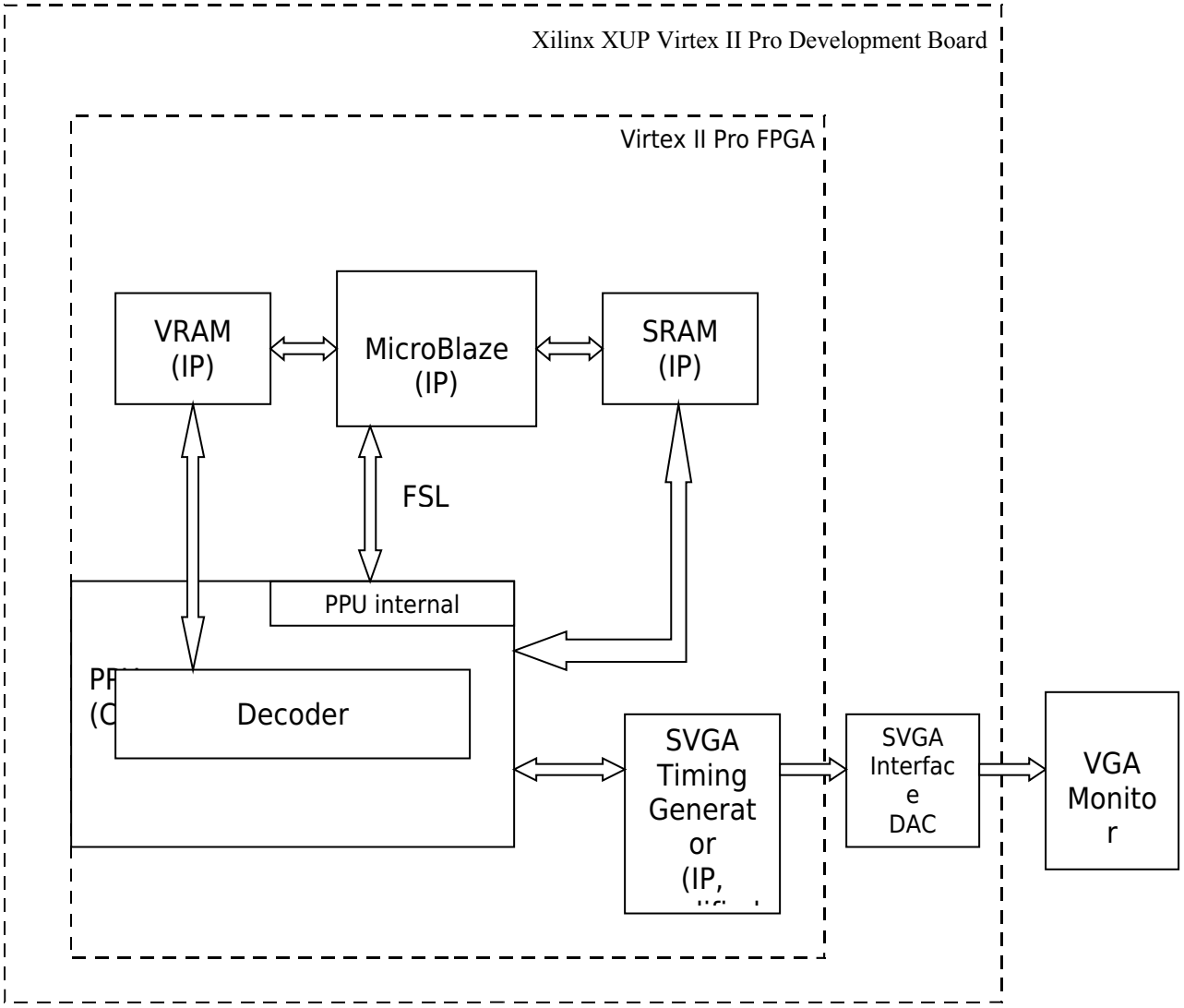

Fig 1.2 Final Design System Block Diagram

Section 2.3 – Brief Description of Blocks in Final Design

MicroBlaze(IP) - The MicroBlaze will be responsible for reading the NES ROM image files from the External Memory and write it into the BRAM. The MicroBlaze will also be used to control the PPU internal register via FSL.

VRAM(IP) – a block ram that stores all the data necessary for rendering the screen.

SRAM(IP) – a 2nd block ram for storing sprite attribute data.

PPU(Custom) – The PPU is a full custom block for rendering the graphics based on data stored in the VRAM and SRAM. The data format will be the same as those used in the real NES PPU.

Bit Mapped Mode SVGA and SVGA interface – Only the SVGA Timing generator is used in the final design. The resolution for the SVGA is 800x600 in our final design. The middle 512x480 will be the display, and rest will be always black.

Section 3 – Outcome

Most of the PPU features proposed were implemented successfully. The features are background and sprite rendering and background scrolling. Although the background scrolling and sprite rendering were a little more complex, they are able to operate smoothly in the final design. For farther description of how the PPU operate, refer to the Detailed Block Description below.

Unfortunately, due to the complicity of the sprite rendering, we did not have time to investigate the idea of getting graphic data from an NES ROM file. To compromise, we spent some time creating our own graphics, which include four backgrounds and a number of sprites.

Overall, two major improvements can be made to the current design. First, the resolution of the SVGA can be changed to 256x240 to match the logical resolution of the Nintendo PPU. In the current design, the SVGA resolution is set to 800x600(monitor resolution) and only the middle 512x480 area is used. Each pixel is rendered twice in the horizontal and twice in the vertical to double the screen size from 256x240 to 512x480. This work around was done since initially we have trouble modifying the SVGA timing to operate in the resolution we want, and we did not want to spend too much time in this as we needed to work on the actually PPU block.

The second improvement will be setting up the compact flash memory to store a NES ROM file, which contains actual graphic data from a NES game. This would involve writing software to copy the data in the ROM file to the VRAM and SRAM, so the PPU can access them. Other improvements would be to implement more features of the actually NES PPU, such as sprite flipping bits. For all the features in the NES PPU refer to [1].

Figure 3.1 PPU final demo screenshot

Section 4 – Detail Block Description

MicroBlaze(IP) – The main purpose of the MicroBlaze is to write graphic data into the VRAM and SRAM. In our demo program, the name table and pattern table are predefined, and it is then written to the both BRAMs at the beginning of the program. After the background is stored, the program will dynamically change the scrolling offset through FSL, and modifying the sprite location to create

animations. The base address of the VRAM is mapped to 0x4200000 of the OPB and the base address of the SRAM is mapped to 0x42010000 of the OPB.

VRAM (IP) – The VRAM stores video data in the following format for the PPU to process:

(1) Pattern Table – Contains 256 tiles used for Background and Sprites. Each tile consists of an 8x8 pixel bitmap with 2bit depth (4 colors).

(2) Name Table – Each frame contains 32x30 tiles (256x240 pixels) with 8bit tile name.

(3) Attribute Table – 2bit color information. Combine with the 2bit in Pattern table to create a total of 16 different colors. Each frame contains 16x15 attribute blocks while each attribute block defines the color of a 16x16 pixel bitmap.

(4) Palette memory – Color map; it contains separate entries for background and sprite.

The pattern table, name table and attribute table together would provide enough information to the PPU to render the background image. The base address of VRAM offsets are as shown below:

VRAM offset:

0000h-0FFFh Pattern Table 0

1000h-1FFFh Pattern Table 1

2000h-23FFh Name Table 0 and Attribute Table 0 (32x30 BG Map)

2400h-27FFh Name Table 1 and Attribute Table 1 (32x30 BG Map)

2800h-2BFFh Name Table 2 and Attribute Table 2 (32x30 BG Map)

2C00h-2FFFh Name Table 3 and Attribute Table 3 (32x30 BG Map)

3F00h-3F1Fh Background and Sprite Palettes

SRAM(IP) – A block ram for storing sprite attribute data. It is capable of displaying up to 64 sprite patterns on screen at a time. The sprite offset defines the sprite's vertical/horizontal positions, the tile

name defined in the pattern table and background priority of the sprite. The sprite attribute together with the Pattern Table will be used to render the sprites. For more information see [1][2][3].

SRAM offset:

00h-FFh - Sprite Attributes (256 bytes, for 64 sprites / 4 bytes per sprites)

Byte 0 – Vertical Position (0 to 239)

Byte 1 – Tile Number (0 to 256, sprite share the same pattern table as the background)

Byte 2 – bit 5 Background Priority (0-Sprite in front of background, 1-sprite behind BG, sprite disabled)

bit 1-0 Sprite palette

Byte 3 – Horizontal Position (0 to 255)

Bit Mapped Mode SVGA and SVGA interface – the SVGA Timing generator generate timing externally for the VGA monitor, and also internally for the PPU. Depending on pixel and line count, it will generate signal that tell the PPU to fetch background, sprite attribute, sprite pattern or palette. The resolution for the SVGA is 800x600. The middle 512x480 will be the display for the PPU, and rest of the screen is not used and will be always black.

PPU(Custom) – The PPU is a full custom block for rendering the graphics based on data stored in the VRAM and SRAM. The data format will be the same as those used in the real NES PPU.

The PPU is divided into five sub-modules. They are ppu_ctrl, color_mux, sprite_ctrl, sprite_render and sprite_pix_mux.

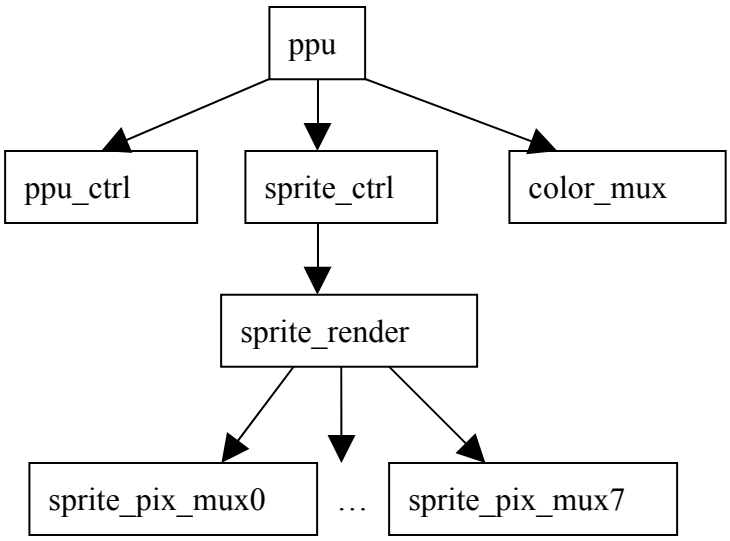

Figure 4.1 PPU hierarchy

ppu

This is the top level of the PPU. It is responsible for fetching data from the VRAM and SRAM and handles the interconnection between the sub modules. It also output the RGB data stream to the SVGA controller.

Background fetches – There are 240 scan lines in total. Each scan line will have 256(32x8) pixels. Four memory fetches are needed for each of the 8 pixels in the scan line. The five fetches include one name table bytes, one attribute table byte and two pattern table bytes. Since there are one cycle delay between the request and the data, a total of five cycles is needed for rendering 8 pixels.

Fetch cycle 1- request for name table data

Fetch cycle 2- request for attribute table data, name table data valid

Fetch cycle 3- request for bit0 of background pattern, attribute data valid

Fetch cycle 4- request for bit1 of background pattern, bit0 pattern data valid

Fetch cycle 5- bit1 pattern data valid

The name table data is used as the address for the pattern table fetch. The attribute table byte and the pattern table bytes are stored into shift registers and they are shifted out bit by bit for each pixel. The attribute data is delayed for two cycles and the first pattern table byte is delayed for one cycle. This will line up the two attribute table bits and the two pattern table bits. The combined four bits will determine the color of a pixel.

Palette fetches – Palette data is fetched at the end of each frame. Eight memory fetches, 4 bytes per fetch. Each byte contains one color, which result in 32 colors, 16 for background and 16 for sprites.

Sprite attribute fetches – the 64 sprite attributes are fetched one line before the scan line that they'll be appear in. This is done for every scan line.

Sprite pattern fetches – since during the PPU active, the VRAM is occupied by the background fetch, the sprite pattern fetches will be done before. The data will be stored into an array of flop ram, and will be used as they are rendering in the scan line.

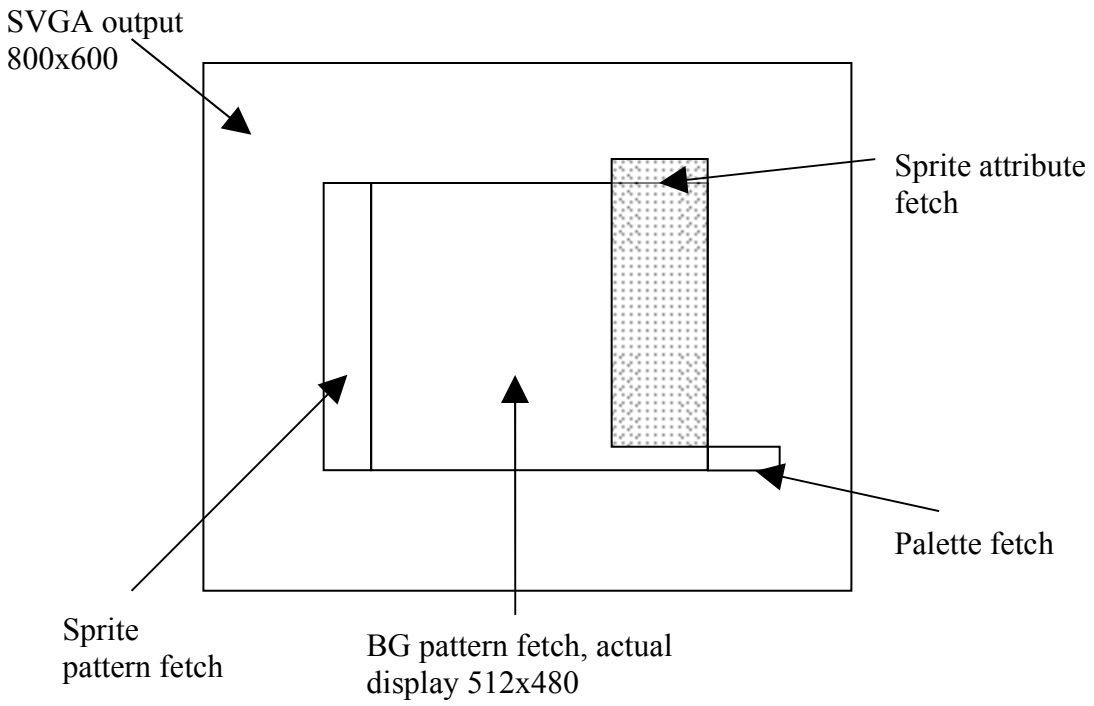

Figure 4.2 Fetch timing (not in scale)

ppu_ctrl

This module is responsible for reading from the FSL and writing to the PPU control register. In the final design, there are three registers. The three registers can be set using one single nfslput() by the MicroBlaze.

Bit 28: 1 – disable PPU, 0 – enable PPU

Bit 22: 1 – vertical scroll using two name table, 0 – scroll using one name table

Bit 21-12: vertical offset (0 to 239)

Bit 10: 1 – horizontal scroll using two name table, 0 – scroll using one name table

Bit 9-0: horizontal offset (0 to 255)

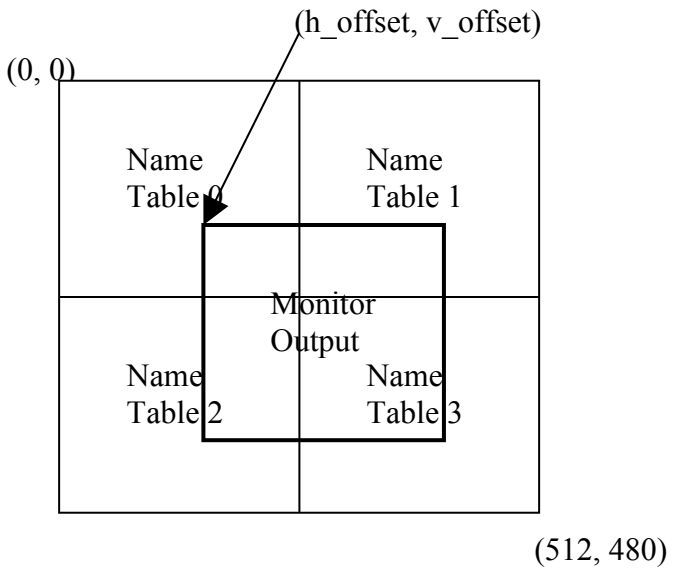

Figure 4.3 Scrolling in PPU

color_mux

This module consists of a number of multiplexers. The palette is stored into an array of flop ram. The appropriate RGB data is 'muxed' out base on the input attribute and pattern bits.

sprite_ctrl

This module is responsible for the sprite rendering. It searches for sprite that will be appearing on the next scan line. The sprite attribute will be stored into the registers if they will be appearing on the next line. The PPU support a maximum of eight sprites per scan line. Only the eight sprites with the highest priority will be rendered. Sprite 0 with the lower address has the highest priority and sprite 64 has the lowest.

The sprite_ctrl also generate a sprite enable signal for each of the eight sprites when they are within the scan range.

sprite_render

This module will chooses which sprite will be output to the color_mux base on the enable signals for each sprite. If the sprites are overlapped, the sprite with the higher priority will be output.

sprite_pix_mux

Choose which bit of the eight pattern data bits to output. There are a total of eight sprite_pix_mux for the eight sprite that will be appearing in the scan line.

Section 5 – Description of Design Tree

Top Level:

system.mhs – top level hardware descriptions
system.mss – top level software descriptions
data/
    system.ucf – user constraints file and pin specifications

Software Modules:

code/
    ppu.c – demo contain a custom animation using the PPU.

Hardware Modules:

pcores/
    ppu_v1_00_a – Verilog implementation of the PPU core
    COLOR_CHAR_MODE_SVGA_CTRL_v1_00_b – bit-mapped mode SVGA controller

Section 6 – References

1. Technical detail of the PPU

http://nesdev.parodius.com/2C02%20technical%20reference.TXT

2. Overall architecture of the NES

http://nocash.emubase.de/everynes.htm

3. Provide links to various documents related to NES

http://nesdev.parodius.com

4. Xilinx XUP Virtex II Pro Development board user guide

http://www.xilinx.com/univ/XUPV2P/Documentation/XUPV2P_User_Guide.pdf

5. Virtex II Pro complete data sheet

http://www.xilinx.com/support/documentation/data_sheets/ds083.pdf