The ZTH1 Micro-Computer, v. 1.0

S. Morel, Zthorus-Labs, 2020-01-29

1 Introduction

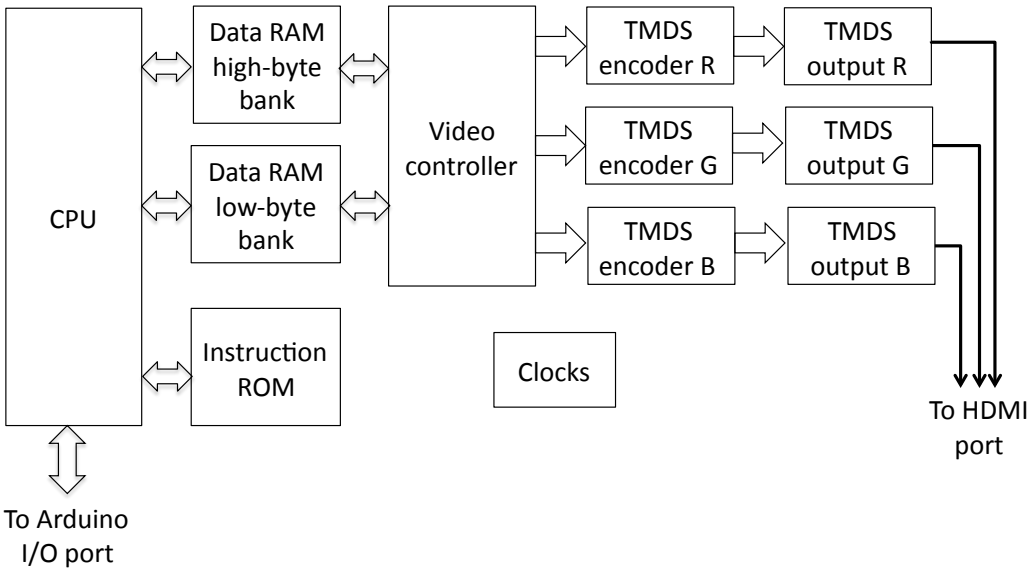
This document describes the architecture of the micro-computer that has been designed around the ZTH1 RISC CPU. This micro-computer can be synthesized on a Maximator board (based on an Altera MAX10 FPGA).

2 General characteristics

The ZTH1 micro-compter consists of the following components that are describes by the indicated VHDL files:

- A ZTH1 CPU (16-bit RISC, Harvard architecture) with instruction and data address buses reduced to 13 bits. The VHDL file describing the CPU is: `ZTH1_CPU.vhd`.
- A read-only memory (ROM) containing the instructions for the CPU, with a capacity of 8192 16-bit words of instructions. The ROM has been created from an IP core and consists of the files: `rom.cmp`, `rom.mif`, `rom.qip`, `rom.vhd`, `rom_inst.vhd`.
- A random-access memory (RAM) for the data, consisting of a bank of 8192 bytes for the "high-byte" part of the 16-bit words addressed by the CPU, and another bank of 8192 bytes for the "low-byte" part. The RAM can be simultaneously addressed by the CPU and by the video-controller by two separate access ports (an access port consists of: an address bus, a data bus, a read-enable signal and a write-enable signals). Because the high-byte and the low-byte banks usually contain different data at initialization, they are distinct entities and the files are: `ram_h.cmp`, `ram_l.cmp`, `ram_h.mif`, `ram_l.mif`, `ram_h.qip`, `ram_l.qip`, `ram_h.vhd`, `ram_l.vhd`, `ram_h_inst.vhd`, `ram_l_inst.vhd`.
- A video-controller allowing to display 128 lines of 192 pixels in 16 colors (individual color for each pixel, no "color-clash") among 16 millions colors available (each red, green and blue component of a color is coded on a byte). This video controller has also the ability to display up to 8 sprites over the image stored in RAM (see Section 4). The VHDL file describing the video-controller is: `video_controller.vhd`.
- An HDMI interface to convert in real-time the RGB and sync signals generated by the video-controller into TMDS signals that are sent to the HDMI port of the Maximator board (the sync signals are coded into the blue channel). The files describing this interface are: `tmds_encoder.vhd` and `tmds_output.vhd`. There are three instances of each file (one for each color component).
- A PLL (phase-locked loop) that generates (from the oscillator on the Maximator board) the 10-MHz clock for the CPU, the 25-MHz clock for the video-controller and the TMDS encoders, and the 250-MHz clock for the TMDS output shift-registers. This PLL has been created from an IP core and the files are: `clocks.cmp`, `clocks.ppf`, `clocks.qip`, `clocks.vhd`, `clocks_inst.vhd`.
- A description of the virtual printed-circuit board on which all the components above are connected and which is the top-level entity of the project: `zth1.vhd`.

The Intel Quartus Prime project to implement the ZTH1 micro-computer on the Maximator board requires also the files: `zth1.qpf`, `zth1.qsf`, `zth1.qws` . The compilation of the project requires 6086 logic elements (the MAX10 of the Maximator board has 8064 logic elements available).



Architecture of the ZTH1 micro-computer (components).)

3 Programming

A program for the ZTH1 micro-computer consists of op-codes of CPU instructions in the ROM (stored in the `ROM.mif` file), and, usually, data in the RAM (stored in the `ram_h.mif` and `ram_l.mif` files). These files can be generated by the ZTH1 assembler running on Linux (see GitHub repository: `zthorus/ZTH1-Assembler`). Once created they have to be copied in the folder of the ZTH1 Intel Quartus-Prime project. It is not necesary to fully recompile the project when updating `.mif` file: just select in the "Processing" menu of Intel-Quartus Prime: "Update Memory Initialization File" and then, in the same menu: "Start" → "Start Assembler". Use the "Programmer" tool to upload the updated `.sof` file into the FPGA.

4 The video-controller

The video-controller generates video frames on the HDMI port, from what is read from the RAM. The 128×192 image to be displayed is stored from address x0000 to address x17FF. To get the address of a pixel (x, y) (starting from $(0, 0)$ = top-left corner of the image), use the formula:

$$address = 48 \times y + x/4$$

This formula returns the address of a 16-bit word that codes a group of 4 pixels. The color of the leftmost pixel in the group is coded by the most-significant nibble (4 bits) of the word and the color of the rightmost pixel is coded by the least-significant nibble.

Because the graphic definition of the ZTH1 micro-computer is less than the definition of a standard HDMI monitor, each displayed pixel is actually made of 2 by 3 pixels. A border is displayed around the image to have it centered on the screen.

The colors are coded by the look-up table (LUT) located at address x1800 to x1817. The high-bytes of this memory zone correspond to the red, green and blue components of each color indexed from 0 to 7, while the low-bytes correspond to the same components for the colors indexed from 8 to 15 (in decimal).

As mentioned, the video-controller of the ZTH1 micro-computer has the ability to display sprites which can be used to program arcade video-games. Each sprite is a 8×8-pixel graphic entity. Sprites can represent characters, spaceships, aliens, bombs, etc... Each sprite has only one color (that shall exist in the LUT). In the bitmap of a sprite, a bit at one represents a pixel with this color, while a bit at zero is considered as a transparent pixel: the image stored in memory will appear through this pixel. Sprites can overlap on each other according to a particuliar order: given two sprites i and j, the sprite j will be at the foreground of the sprite i if $j > i$. The sprite 0 will remain anyway at the foreground of the displayed image. Each sprite is characterised by the following parameters:

- An "activity" bit. If this bit is at one, the sprite will be visible on the screen. If it is at zero, the sprite will not appear.
- The coordinates (x, y) (x from 0 to 191, y from 0 to 127) of the top-left corner of the sprite.
- The color of the sprite (from 0 to 15).
- The bitmap (8 bytes) of the graphic of the sprite. Each byte describes a line of 8 pixels. Note that this graphic shall be coded "horizontally mirrored" compared to what will be displayed. For example is a line of the bitmap is a dot at the extreme left (x80), then it shall be coded as x01.

The activity bit and the (x, y) coordinates are part of the same 16-bit word in the RAM.

The video-controller also watches for sprite collisions. The collision is detected only for the sprites 0 and 1 which are considered as "player sprites" (motion controlled by the player). The collision report for each of these sprite is a 4-bit nibble that can be read at any time from the RAM and which contains the following possible values (in decimal):

- 0: No collision.
- 1: Collision with a pixel of the image that has the color 1. The color 1 is special and can be used, for example, to represent the ground in a game (hence, it is possible to detect if a player sprite has hit the ground). It can take any RGB value.
- 2 to 7: Collision with sprite 2 to 7.

The organization of the RAM used by the video-controller is a little complex and can be described by the following table:

Address	**High-byte**	**Low-byte**
x0000 to x17FF	High nibble: color of pixel x mod 4 = 0, Low nibble: color of pixel x mod 4 = 1	High nibble: color of pixel x mod 4 = 2, Low nibble: color of pixel x mod 4 = 3
x1800	Red, color 0	Red, color 8
x1801	Green, color 0	Green, color 8
x1802	Blue, color 0	Blue, color 8
x1803	Red, color 1	Red, color 9
x1804	Green, color 1	Green, color 9
x1805	Blue, color 1	Blue, color 9
x1806	Red, color 2	Red, color 10
x1807	Green, color 2	Green, color 10
x1808	Blue, color 2	Blue, color 10
x1809	Red, color 3	Red, color 11
x180A	Green, color 3	Green, color 11
x180B	Blue, color 3	Blue, color 11
x180C	Red, color 4	Red, color 12
x180D	Green, color 4	Green, color 12
x180E	Blue, color 4	Blue, color 12
x180F	Red, color 5	Red, color 13
x1810	Green, color 5	Green, color 13
x1811	Blue, color 5	Blue, color 13
x1812	Red, color 6	Red, color 14
x1813	Green, color 6	Green, color 14
x1814	Blue, color 6	Blue, color 14
x1815	Red, color 7	Red, color 15
x1816	Green, color 7	Green, color 15
x1817	Blue, color 7	Blue, color 15
x1818	Most-significant bit = activity of sprite 0, rest of byte = y coordinate of sprite 0	x coordinate of sprite 0
x1819	Most-significant bit = activity of sprite 1, rest of byte = y coordinate of sprite 1	x coordinate of sprite 1
x181A	Most-significant bit = activity of sprite 2, rest of byte = y coordinate of sprite 2	x coordinate of sprite 2
x181B	Most-significant bit = activity of sprite 3, rest of byte = y coordinate of sprite 3	x coordinate of sprite 3
x181C	Most-significant bit = activity of sprite 4, rest of byte = y coordinate of sprite 4	x coordinate of sprite 4
x181D	Most-significant bit = activity of sprite 5, rest of byte = y coordinate of sprite 5	x coordinate of sprite 5

Address	High-byte	Low-byte
x181E	Most-significant bit = activity of sprite 6, rest of byte = y coordinate of sprite 6	x coordinate of sprite 6
x181F	Most-significant bit = activity of sprite 7, rest of byte = y coordinate of sprite 7	x coordinate of sprite 7
x1820 to x1823	Free	Free
x1824	Free	High nibble: collision report for sprite 1, low nibble: collision report for sprite 0
x1825	Free	Free
x1826	Color of sprite 0	Color of sprite 4
x1827	Color of sprite 1	Color of sprite 5
x1828	Color of sprite 2	Color of sprite 6
x1829	Color of sprite 3	Color of sprite 7
x182A to x1831	Bitmap of sprite 0	Free
x1832 to x1839	Bitmap of sprite 1	Free
x183A to x1841	Bitmap of sprite 2	Free
x1842 to x1849	Bitmap of sprite 3	Free
x184A to x1851	Bitmap of sprite 4	Free
x1852 to x1859	Bitmap of sprite 5	Free
x185A to x1861	Bitmap of sprite 6	Free
x1862 to x1869	Bitmap of sprite 7	Free

5 I/O lines for peripherals

On the Maximator implementation, the ZTH1 has 4 input lines and 4 output lines (which can be read or write using the INP and OUT instructions) that are connected to the Arduino I/O port of the board:

ZTH1 line	MAX10 pin name	Arduino port pin
Input 0	J16	D2
Input 1	H15	D3
Input 2	H16	D4
Input 3	G15	D5
Output 0	G16	D6
Output 1	F16	D7
Output 2	E15	D8
Output 3	E16	D9

Moreover, the pin L16 (Arduino D0) triggers the interrupt vector 0 (reset = jump to instructions at address x0000) and the pin J15 (Arduino D1) triggers the interrupt vector 1 (jump to instructions at address x0008). Each interrupt is triggered when its pin is connected to the ground of the board.

___oOo___