European FPGA Developer contest 2019
Project: Digital billboard controller

Contributor:

Philippe THIRION

3 rue de Colmar

F-67118 Geispolsheim

France

General description

The digital billboard controller enables to display still images on a VGA monitor. The device contains a programming interface (SPI) mainly to load images in board memory. When an image is stored in memory, the programming interface could be disconnected from the system.

It's possible to load up to two different images in memory. A programming bit (PAGE) enables to switch between the images for displaying. This feature allows to load a new image while the current one is displayed.

Another feature is to disable display. The screen is black but the synchronization signals continue to be present. This feature is activated after system reset and avoid displaying random image before loading the first image in memory. The usage is to deactivate a programming bit (BLANKING) after loading the first image.

Detail description

The project uses an Arrow MAX1000 board which contains a MAX10 10M08SA device.

VGA format is the standard 640x480 at 60 Hz. Color depth is limited to 256 values.

The FPGA 's on chip memory is limited to 378 Kbit. To store an image, we need 2457600 bit (640 x 480 x 8). In conclusion, the internal memory size is not sufficient. Fortunately, MAX1000 board includes 64 Mbit SDRAM. Then it would be possible to store up to 25 different images. The need for this project is only to store two different images.

The FPGA contains three main blocks:

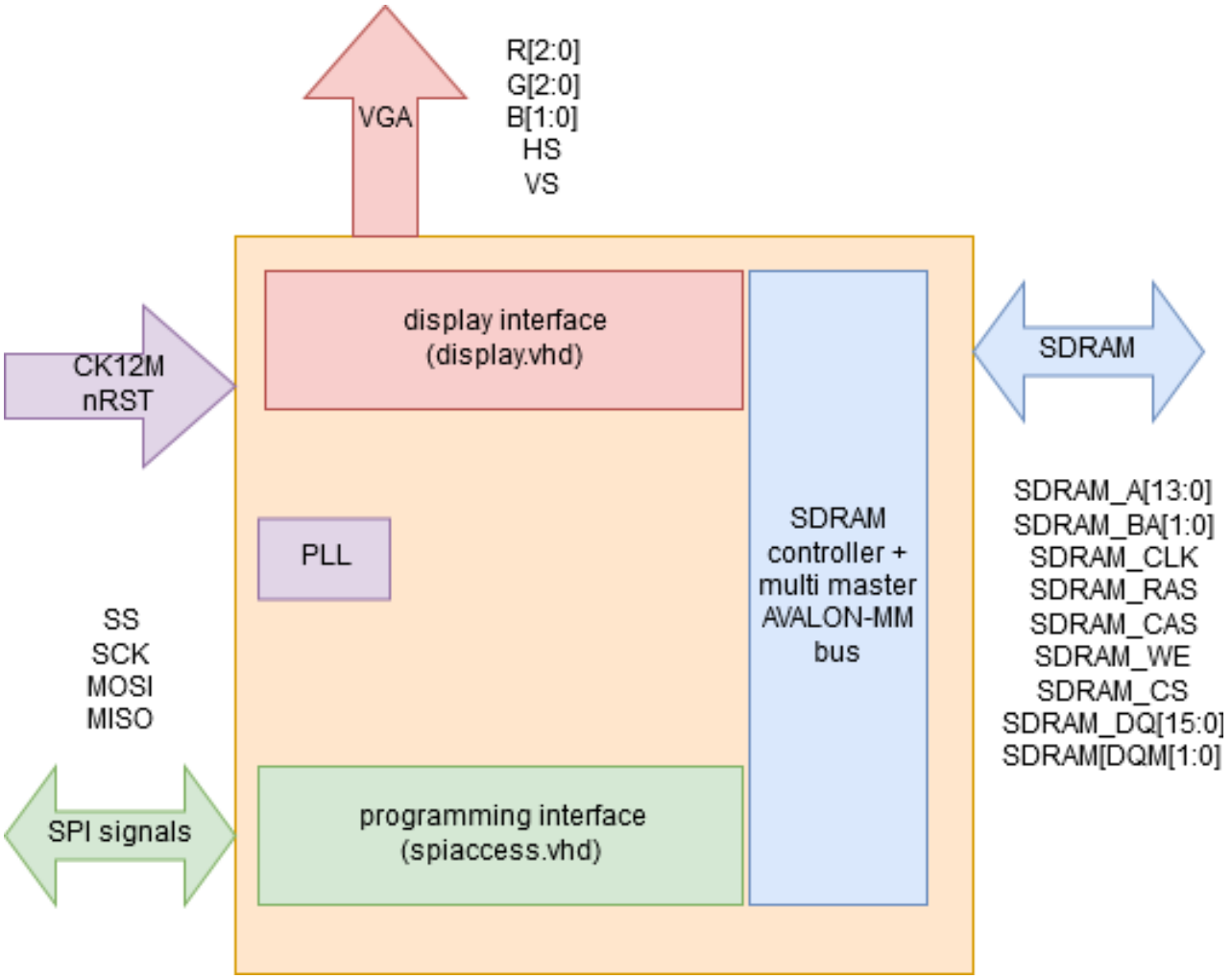
- SDRAM controller with multi master AVALON-MM bus.
- A programming interface to enable accesses to SDRAM, to a controller register or to read a status register.
- A display interface, to read pixel information in the SDRAM and output VGA signals.

LEDs management block is also included to show activity (SPI accesses …).

An “on chip” PLL allows to generate a pixel clock (at 25.2 MHz) and two ram clocks (at 72 MHz).

The pixel clock is only used in the display interface.

The first ram clock drives SDRAM controller with multi master AVALON-MM block, the programming interface and a part of the display interface. The second ram clock drives directly the clock of the SDRAM component on the board with a phase of -5 ns.



SDRAM controller with multi master AVALON-MM bus block

This block is created using the QSYS tool available in Quartus prime 17.0.

This block contains:

- A SDRAM controller
- An AVALON-MM pipeline bridge for the programming interface.
- An AVALON-MM pipeline bridge for the display interface block.

AVALON-MM pipeline bridge for the programming interface enable to perform single write and read accesses to the SDRAM.

AVALON-MM pipeline bridge for the display interface block is designed to perform only 8 words burst read accesses to the SDRAM.

SDRAM data bus is 16 bits wide. So, all accesses will be always 16 bits wide. This means that we transfer always two pixels.

All these features could be found in the sdram.qsys file.

Programming interface block

This block (spiaccess.vhd) contains:

- An SPI slave front end module (clocked with SCK): spislave.vhd
- A register addressing module: spiregister.vhd
- An interface with AVALON-MM bus: avalon_access.vhd

The register addressing module resynchronizes the signals coming from SPI slave module and enable access to:

- AVALON-MM interface (write and read)
- A control register (write and read)
- A status register (only read)

A SDRAM read access is performed in three steps:

- Write the address to AVALON-MM interface.
- Perform a read command.
- Read the data stored in the read data register.

A SDRAM write access is performed in two steps:

- Write the address to AVALON-MM interface.
- Write the data to AVALON-MM interface.

An auto increment address mechanism is provided to avoid writing the address at each access.

Control register mapping (read and write):

	Name	Description
Bit 7-3	unused	
Bit 2	AINC	Auto increment address feature. After reset value is 1 (activated)
Bit 1	PAGE	Select image page to display. After reset value is 0. 0: SDRAM address from 0x000000 1: SDRAM address from 0x100000 (bit20)
Bit 0	BLANKING	Black image is displayed. After reset value is 1 (activated)

Status register mapping (only read):

	Name	Description
Bit 7	UNDERRUN	Underrun detected in display module. Cleared after status register reading
Bit 6-1	unused	Always at 0
Bit 0	DATA DONE	AVALON-MM interface availability: 0: transfer in progress to SDRAM (read or write) 1: available for new transfer.

SPI operation format (mode 0) - SCK frequency up to 36 MHz (72 MHz / 2)

Opcode	Byte1	Byte2	Byte3	Operation
0x01	MSB	LSB	-	Write SDRAM data
0x02	MSB	XSB	LSB	Write SDRAM address
0x03	-	-	-	Read SDRAM command
0x30	(MSB)	(LSB)	-	Read data read
0x40	(DATA)	-	-	Read status register
0x05	DATA	-	-	Write control register
0x50	(DATA)	-	-	Read control register

Trick: Opcode most significant nibble decoding is used for reading accesses to fetch data before shift out.

Display interface block

This block contains mainly:

- An AVALON-MM read interface: avalon_read.vhd
- A FIFO 8x16 to retrieve 8 words burst data: fifo_control.vhd + DPRAM
- Another FIFO 8x16 to solve clock domain crossing cdcfifo8.vhd
- A VGA controller vga_controller.vhd
- A RGB interface module rgbout.vhd

Each time the first FIFO is detected empty, a request to read a burst of 8 words is send to AVALON-MM.

At the end of the image frame, during active vertical synchronization, FIFO's and AVALON-MM interface are cleared to reinitialize image frame and AVALON-MM address is reset at the beginning of the page.

CDCFIFO enables synchronization pixel information from ram clock domain at 72 MHz to pixel clock at 25.2 MHz.

RGB interface splits 16-bit word into two bytes for displaying.

The incoming bandwidth (from SDRAM side) is 16 x 72 = 1152 Mb/s and the output bandwidth (VGA side) is maximum 8 x 25.2 = 201.6 Mb/s. So, we hope to have never underrun condition on the RGB interface. A mechanism is implemented to detect underrun condition. This detection information is readable from the programming interface.

LED display

Each LED light on during one second if some events are detected:

LED1	Underrun detected in display side.
LED2	SPI activity detected with SDRAM.
LED3	Activity detected between SDRAM and display side.

HDL tests

Some VHDL test benches have been developed to test the different part of the design.

Perhaps some tests need to be adapted slightly to match recent module interface changes.

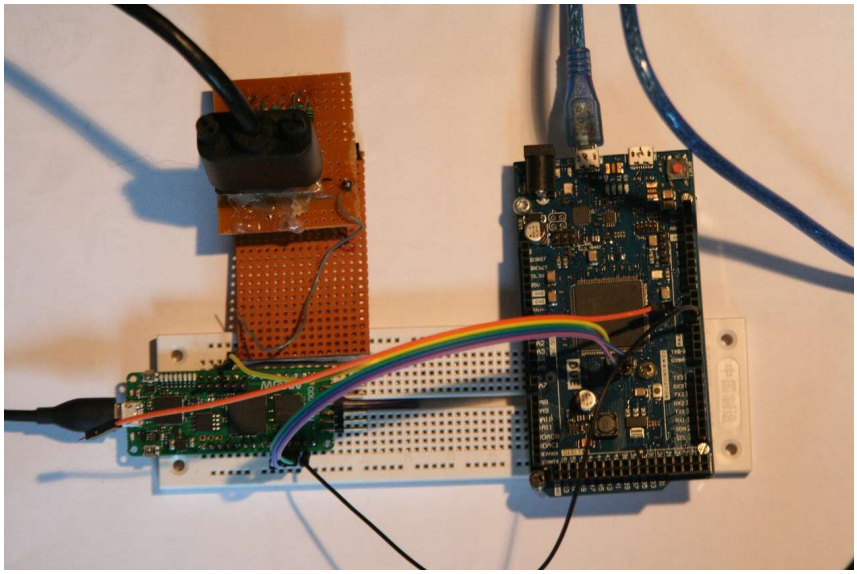
For simulation and VHDL tests, I used open source GHDL et GTKWAVE software.

Board test

Test environment

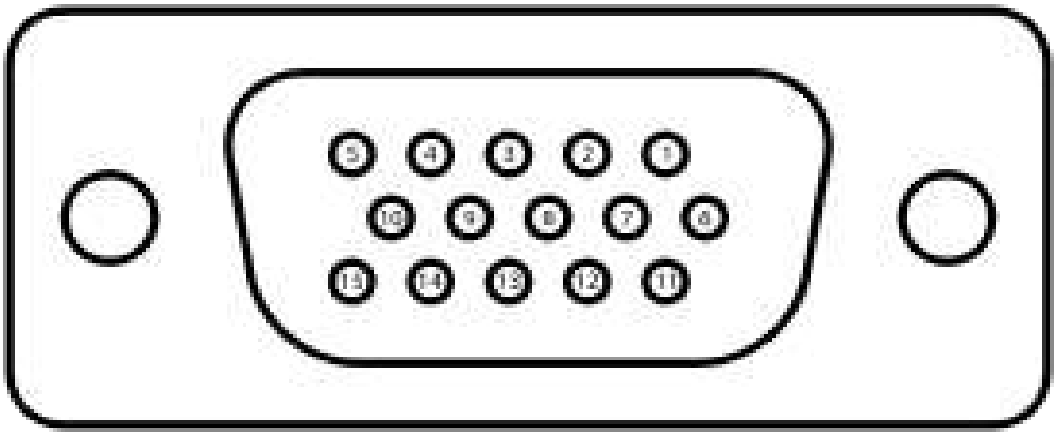
Board test platform contains:

- An Arrow MAX1000 board
- An Arduino due
- A self-made board to link the RGB FPGA outputs to a standard VGA connector



Self-made board connections:

FPGA signal	Resistor (Ohm)	VGA connector pin	Description
R[0]	2K	1	Red
R[1]	1K	1	
R[2]	500	1	
G[0]	2K	2	Green
G[1]	1K	2	
G[2]	500	2	
B[0]	1K	3	Blue
B[1]	500	3	
HS	220	13	Horizontal synchro
VS	220	14	Vertical synchro
GND	0	5,6,7,8,10	+ shield



Arduino due

The Arduino Due is a microcontroller board based on the Atmel SAM3X8E ARM Cortex-M3 CPU. It is an Arduino board based on a 32-bit ARM core microcontroller. Arduino due contains natively a SPI master interface and deals with 3.3v signals. Popular Arduino SDK provides all necessary libraries to program easily this component. A drawback is that the program code is limited to 512 Kbytes. Then, it's impossible to store two different images in the program flash without using a trick!

Demo test

The purpose is to display two different images on a VGA monitor. Each image is displayed alternatively for 10 seconds.

The scenario is:

- With blanking activated, load first image in SDRAM
- Deactivated blanking
- Loading second image on the other page
- Every 10 seconds switching between first and second image.

Preprocessing

The first operation is to be able to create an 640x480 bytes array with RGB information.

I used python code with PIL (or pillow) library. This library allows to process image files in different format (gif, png …) and extracts RGB information. So, I can easily create my 640x480 bytes array (see image.py).

Unfortunately, only 512Kb are available in the Arduino due flash for programming and SRAM size is ridiculous (64 Kb).

RGB information is often a sequence of same value and a RLE (run length encoding) algorithm can reduce the array size in a lossless way for at least 50%!

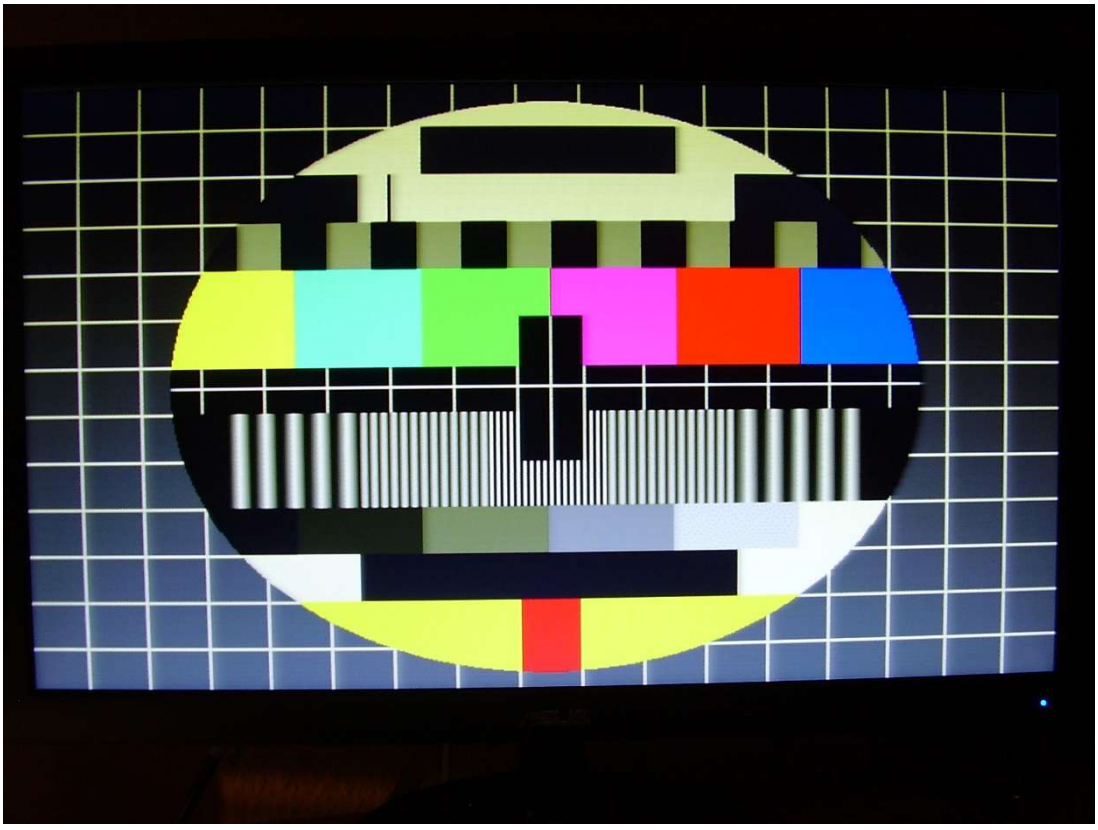
In the image2.py, in addition, I implement a custom RLE encoding algorithm.

Arduino program

The image array then is inserted in the Arduino program and I develop a C++ object to decode the image array on the flow (one pixel at once).

Files are loadimage.ino, rle_stream.h and rle_stream.cpp.

Results

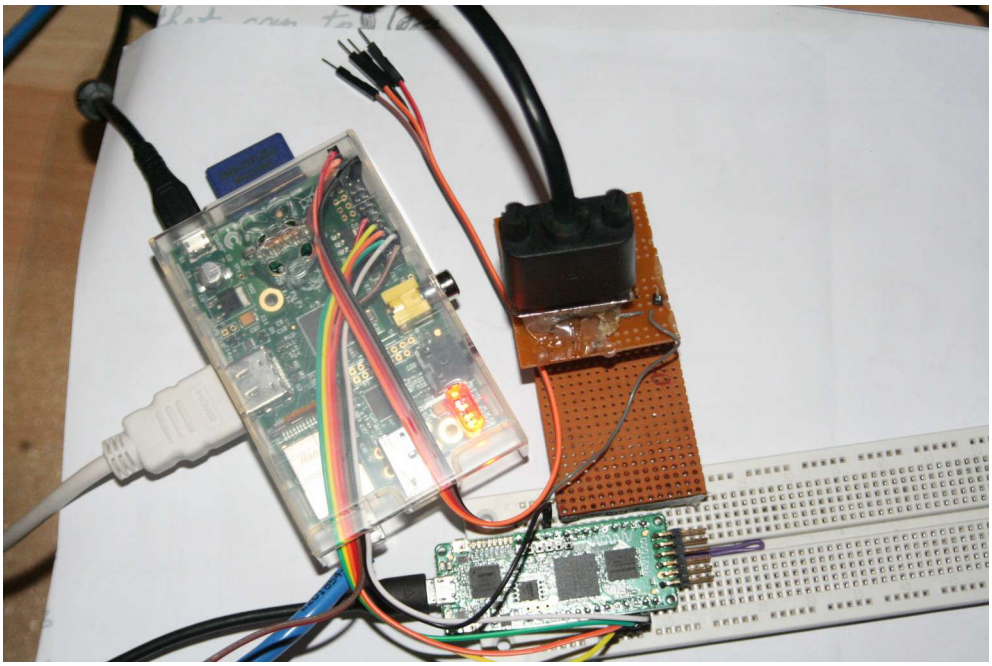


Test board with raspberry pi

Arduino due is not adapted to continue developing applications for this project. All platform with SPI master feature could be a potential candidate!

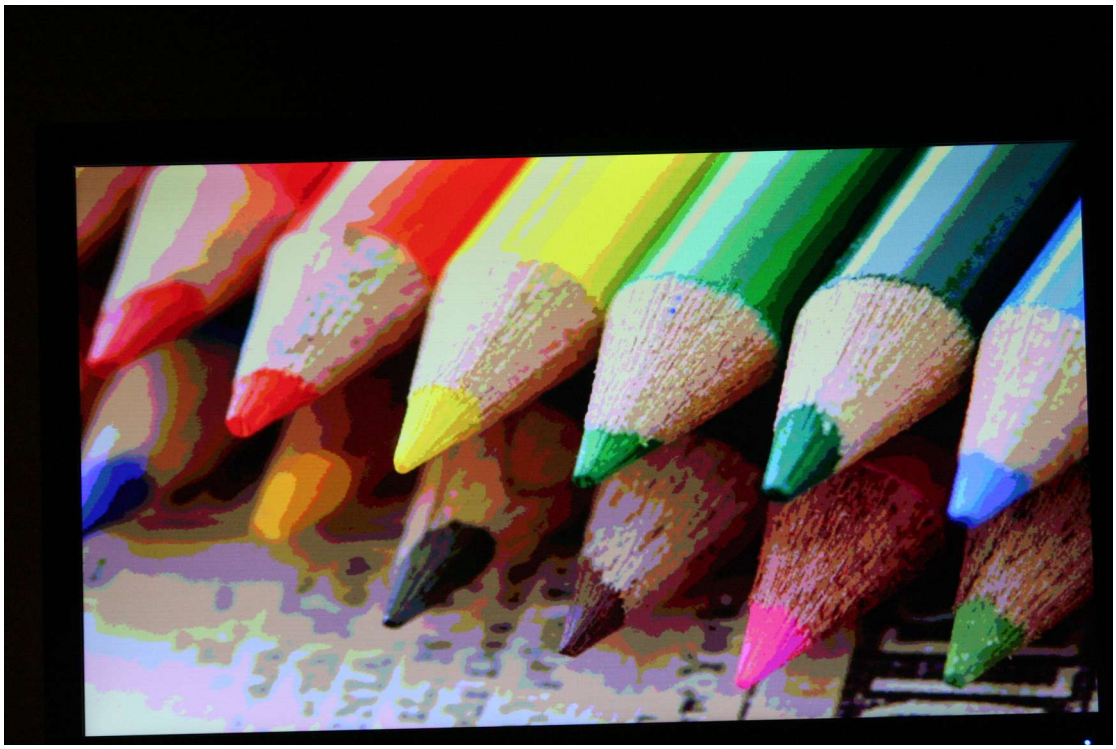
Using a raspberry pi enables to have more available memory. So I replace the arduino due by a raspberry pi.

Image processing and spi accesses can be done in a same python program. You need to install spidev package and python PIL library. So you can input directly 640x480 gif or png images. The program loadimage.py manages also automatic switching between pages.



Limitation

Using only 256 colors limits to display correctly all kind of images:



And now?

Here I stop my work on this project for this contest. But the story could continue ..

- In the IoT domain, using ESP32 or ESP8266 could be a solution to manage remotely several digital billboards board with wifi support.
- We use only two pages to store images. Some slight modifications in the VHDL should allow to address up to 25 different images.
- 256 colors is too few to display full colored image. Storing 65536 colors image should give best results but need more memory, working probably with a higher ram clock frequency and a DAC component to transform digital outputs to analog VGA signal.