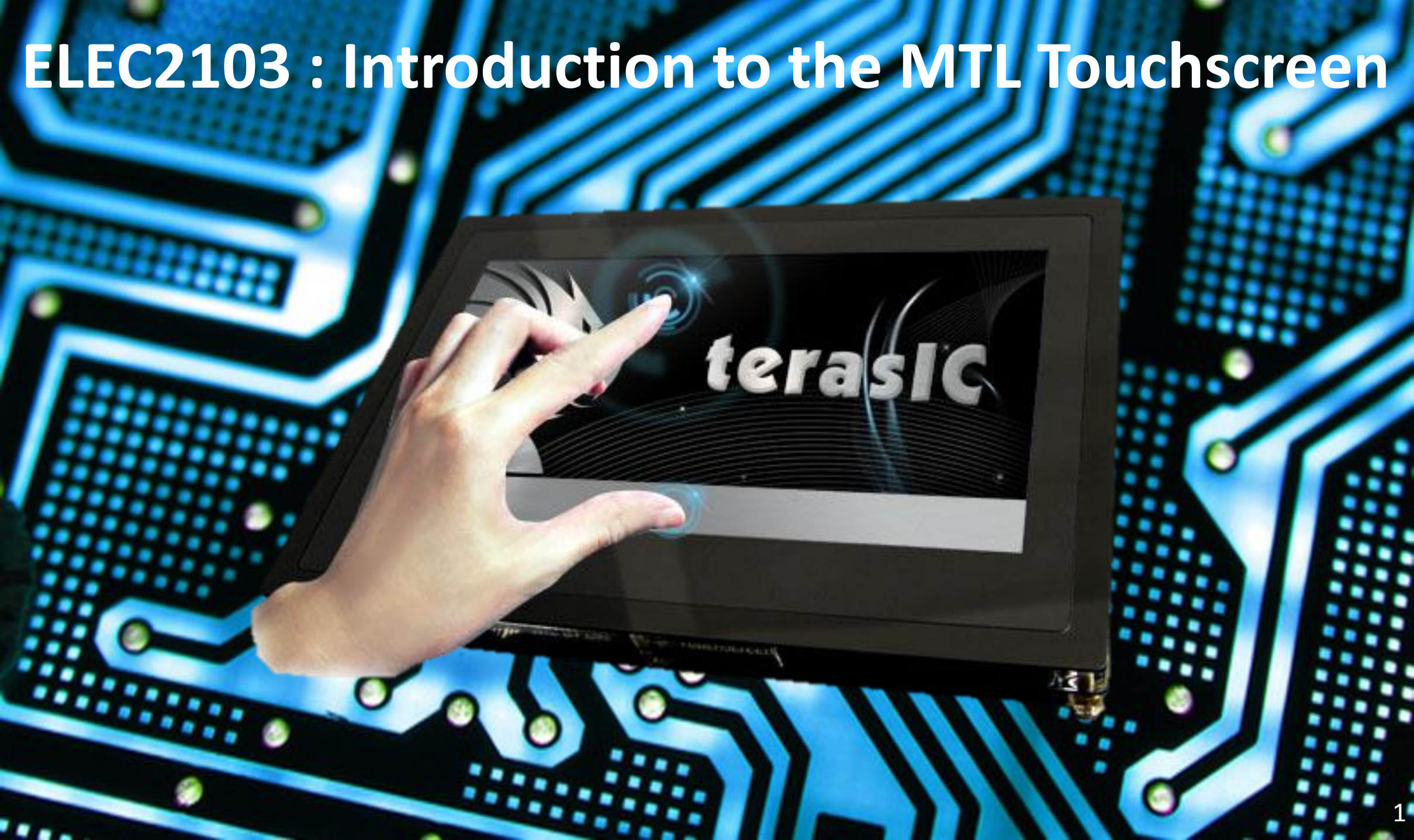
ELEC2103 : Introduction to the MTL Touchscreen



Introduction to the MTL Touchscreen: Plan

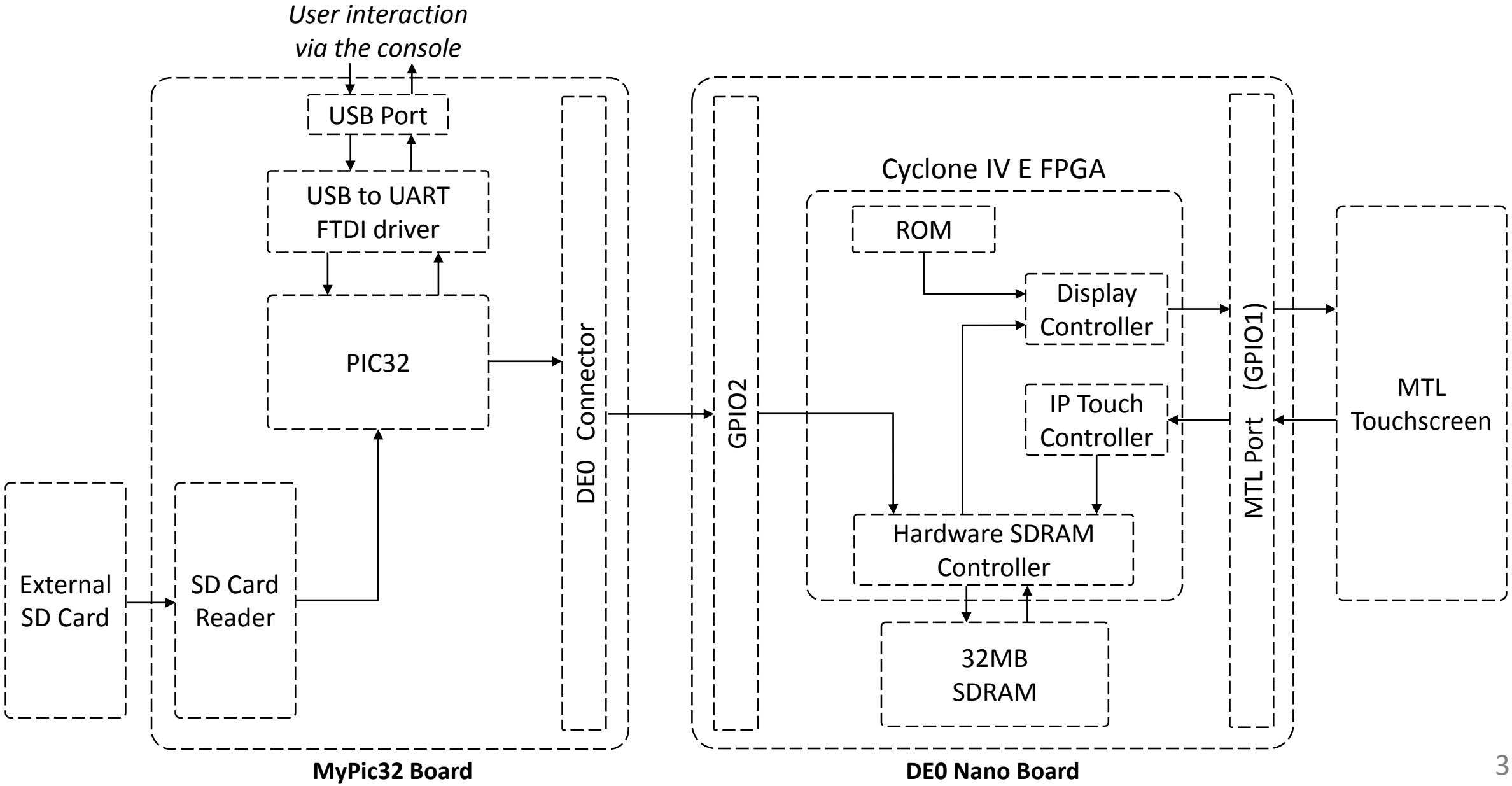
A. Functional overview of the complete system

B. Description of three key points

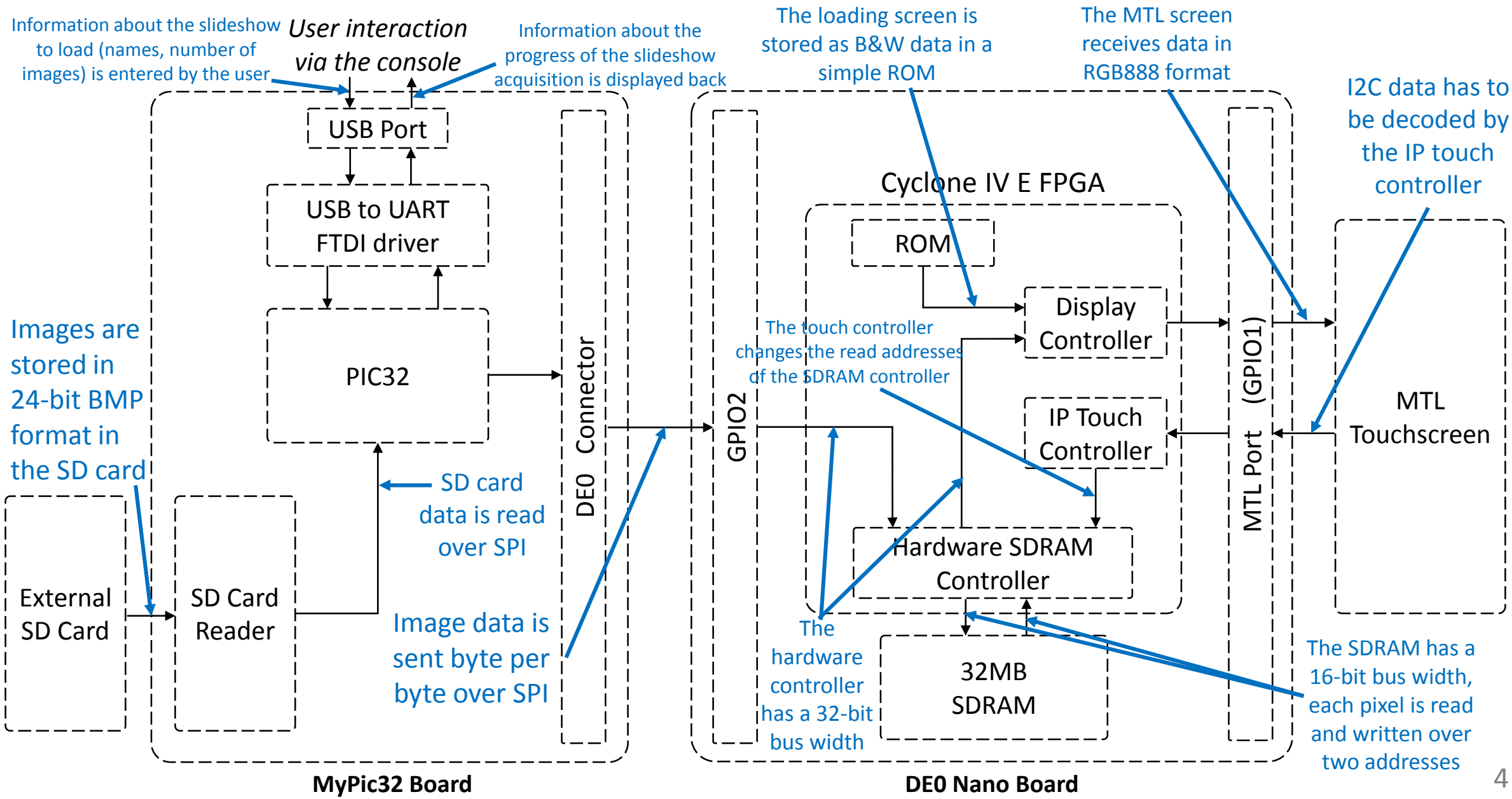
- the commands of the MTL touchscreen,
- the BMP file format,
- the hardware SDRAM controller.

Note : This presentation is only intended to give you the big picture.
For minor elements and implementation details, please refer to the code: it has been thoroughly commented.

A. Functional overview of the complete system

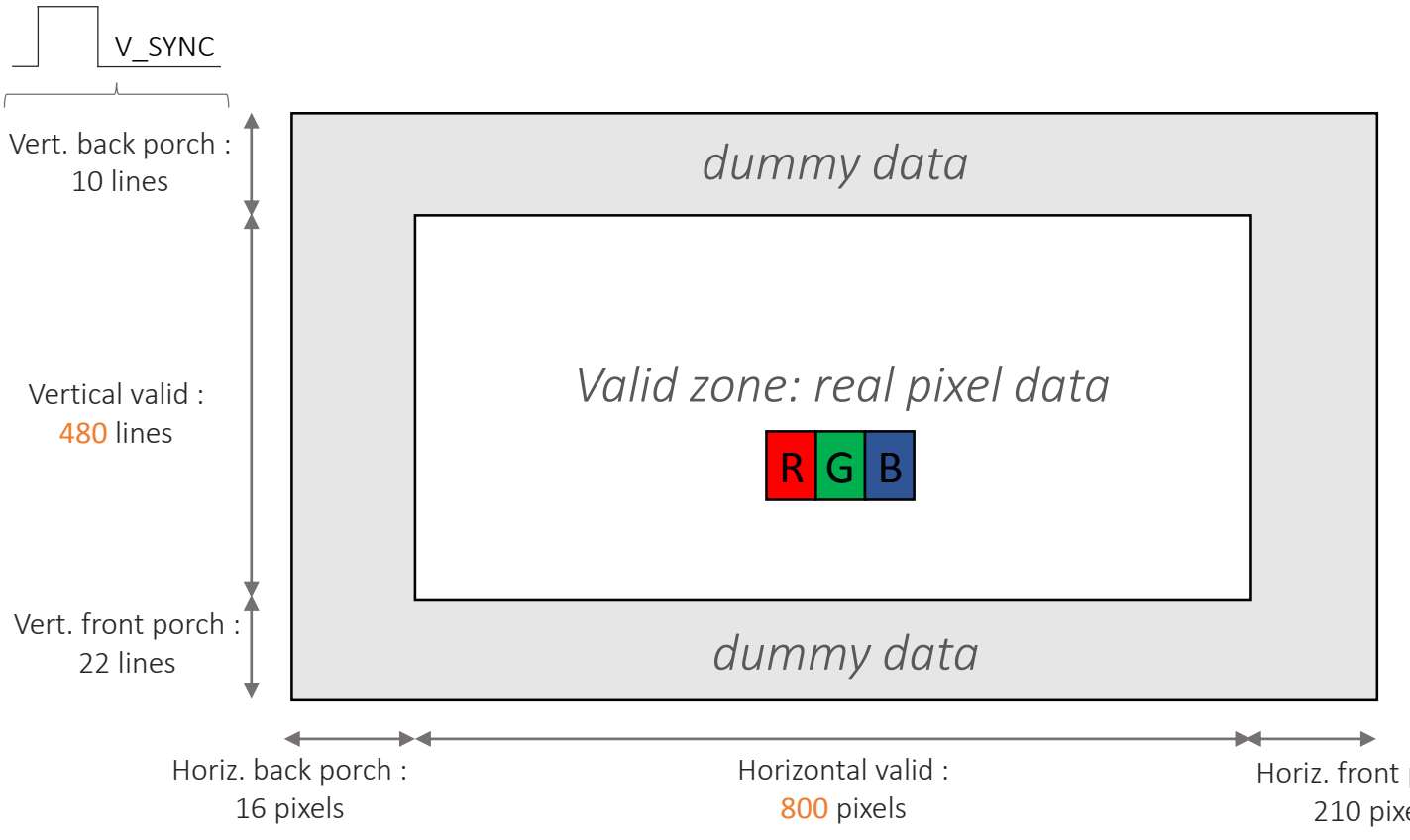


A. Functional overview of the complete system



B. Description of three key points

How to control the 800x480 LCD screen



What is needed to control the LCD screen (RGB888):

- CLK 1-bit clock signal (33 MHz for the MTL),
- H_SYNC 1-bit horizontal sync. signal,
- V_SYNC 1-bit vertical sync. signal,
- RED 8-bit red value of the current pixel,
- GREEN 8-bit green value of the current pixel,
- BLUE 8-bit blue value of the current pixel.

↳ One could also use RGB565 control and lose a bit of color resolution.

Note : All of this is already implemented as a module in **`mtl_controller.sv`**.
But if you want to have a look at some typical timing diagrams, you can read chapter 1 of the application note "*QVGA TFT-LCD direct drive using the STM32F10xx FSMC peripheral*" in the project file folder.
The screen which is described is not the same as the MTL and the context is different, but it will give you a good overview.
For more details about the LCD timing parameters, refer to section 3.2 of the MTL datasheet, also in the project file folder.

B. Description of three key points

How data is retrieved from the BMP files

Using data in the RGB888 format for the LCD screen, it seems quite natural to use the 24-bit BMP file format.

The BMP file contains:

- some general information about the picture, the main ones are
  - the file size at byte number 2,
  - the number of columns at byte number 18,
  - the number of rows at byte number 22,
  - the number of bits per pixel (here, 24) at byte 46.
- then each pixel's data sequentially
  - first byte is blue,
  - second byte is green,
  - third and last byte is red.

383201	383202	…	384000
…	…		…
801	802	…	1600
1	2	…	800

Pixel order in the BMP file format : example for a 800x480 picture.

With 3 bytes for each of the 800x480 pixels, the BMP files weigh more than 1 MB!

One can observe that pixels are displayed from top left to down right on the LCD screen.
The order of the lines in the BMP file format has to be mirrored: this is done in the C algorithm which is run by the PIC32 when data is extracted from the BMP files on the SD card.

B. Description of three key points

How to read from and write to the SDRAM

The SDRAM controller has two read ports and two write ports.
Only one of each is used in the project code, their corresponding interfaces are shown below.

Write port (x is 1 or 2) :

WRx_DATA	- 32-bit data to be written to the SDRAM,
WRx	- 1-bit trigger: indicates when WRx_DATA is valid and can be pushed onto the internal write FIFO,
WRx_ADDR	- 24-bit base (minimum) address to write to,
WRx_MAX_ADDR	- 24-bit maximum address to write to,
WRx_LENGTH	- 9-bit number of addresses whose data will be stored in the write FIFO before being written,
WRx_LOAD	- when asserted, the internal write FIFO is cleared and the address bounds are registered,
WRx_CLK	- clock signal associated with the port and its signals.

Read port (x is 1 or 2) :

RDx_DATA	- 32-bit data which has been read from the SDRAM,
RDx	- 1-bit trigger: indicates when RDx_DATA is updated and can be pulled from the internal read FIFO,
RDx_ADDR	- 24-bit base (minimum) address to read from,
RDx_MAX_ADDR	- 24-bit maximum address to read from,
RDx_LENGTH	- 9-bit number of addresses whose data will be stored in the read FIFO, waiting to appear on RDx_DATA,
RDx_LOAD	- when asserted, the internal read FIFO is cleared and the address bounds are registered,
RDx_CLK	- clock signal associated with the port and its signals.

B. Description of three key points

How to read from and write to the SDRAM

Some important elements :

- Data is stored in the SDRAM as 16bits/address, the interface of the SDRAM controller has been made 32 bits so you don't have to worry about the details.
- Two addresses are used for each pixel (which contains 24 bits of data), so be cautious there is a factor 2 between anything expressed as a number of pixels and the **xxx_ADDR**, **xxx_MAX_ADDR** and **xxx_LENGTH** signals.
- Addresses are not individually accessible: they are always read/written sequentially. **x_ADDR** and **x_MAX_ADDR** only give bounds! The current address is internal to the controller: you have to keep track of what you do.
- On the write side, when you send data to the controller, everything is stored in a write FIFO. The controller waits for enough data in the FIFO to write [*WRx_LENGTH*] addresses to the SDRAM in a burst: nothing is written before that.
  If you want to write each pixel as it arrives, you must have [*WRx_LENGTH*] equal to 2 (recall: two addresses written per pixel).
- It behaves in a similar way on the read side, but here [*RDx_LENGTH*] addresses are read in the SDRAM and the values are stored in the read FIFO, this is done *before* you actually assert the **RDx** trigger. When you do, data will be available instantly.
- The controller internally decides for itself what to do and when (writing, reading, refreshing data,…), all of this is completely transparent to the user. Quite handy, isn't it?
  In fact, that wouldn't be possible without the guarantee that reading and writing are purely sequential.