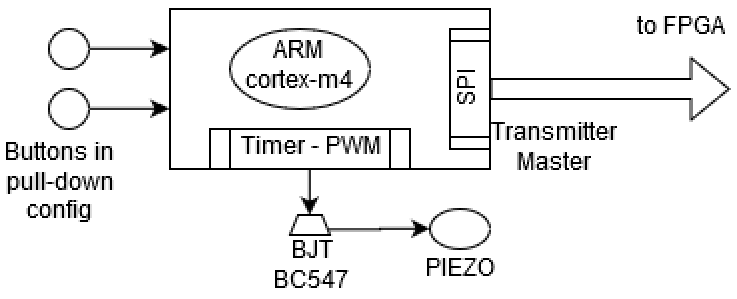
Micro-game board

The final product is a low-power microcontroller-based videogame platform with the size of a keychain that can be connected to an fpga-based extender device in order to project the screen to a monitor/television.
The pixel resolution of the platform is 128x64. The portable display size is 1.3 inch, while the resolution of the VGA output allowed by the extender device is 800x600@60Hz.

(photo of dev environment)

Microcontroller side:
The selected microcontroller is a STM32L476, where L stands for Low-power. It implements an ARM cortex M4 processor and operates at a max. frequency of 80 MHz.



- Display and SPI
The 128x64 screen is buffered in RAM as an 8 bit (unsigned int) array of length 1024. This buffer is sent through the SPI peripheral either to the FPGA or directly to the oled display. The SPI transmission takes place every 16 ms to ensure a screen refresh rate of 60 Hz, a timer is used to accurately enable the communication. Its baud rate is 10 MHz, thus the transmission of the entire buffer will last about 1 ms. This means that all the game logic must fit the 15 ms interval between screen refreshes.
The system clock is now configured @ 80 MHz, so quite a great amount of operations can be performed on such a small resolution display.
The SPI and timer peripherals have been used directly manipulating their configuration registers.

- Buttons
At the moment, two buttons have been mounted in a pull-down configuration. A timer is employed, in an interrupt-based fashion, to debounce them and to continuously check whether a button is actually pressed or not.

- Buzzer and PWM note-player
To play game sound, a piezo-electric buzzer has been employed. This load is driven by a BJT transistor that is controlled by a PWM signal coming from the MCU. Specifically, the PWM signal is generated using an internal timer. The period of this timer will determine the frequency of the resulting signal, thus it will determine the effective note being played by the buzzer. While active, the count value of the timer is compared to another configurable register, when these 2 are equal the PWM signal is toggled. When the counter value reaches the end of its period the PWM signal is toggled again. In this way, acting on that configurable register the PWM duty cycle is modified and the volume of the buzzer is regulated.
Another timer is used to control the duration of the output note: it is activated when the note starts to be transmitted and it expires according to the wanted duration for that note. When it expires, it

stops the frequency timer and itself.
In figure 2, the BJT configuration adopted is shown and the parameters of the circuit are represented. Given those values, a correct resistance Rb had to be selected. Two different resistance values were initially taken in consideration, both were valid according to the lower and upper bounds found.

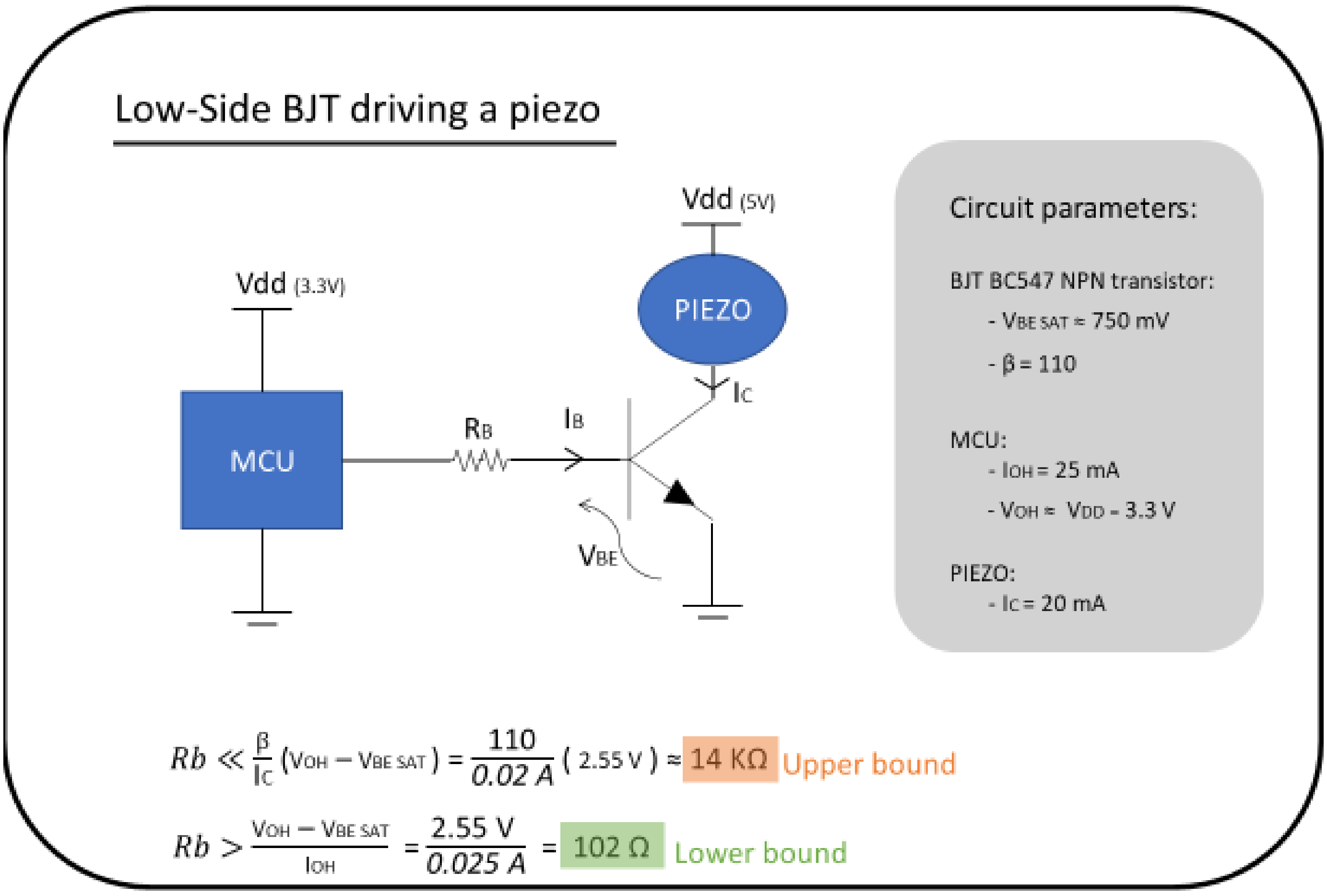
1 – **Rb = 1 KΩ** -> Strong saturation, more power consumption and louder piezo sound.

This implies, **Ib** = $\frac{V_{OH} - V_{BE\,SAT}}{Rb}$ = 2.55 V / 1KΩ = **2.5 mA >> 0.18 mA = Ic / β**

2 – **Rb = 10 KΩ** -> Weak saturation, quieter piezo sound.

Implies **Ib** = 2.55 V / 10 KΩ = **0.25 mA > 0.18 mA**

Finally, the resistor inducing a weaker saturation of the BJT was selected because the final result was still good.



Circuit parameters:

BJT BC547 NPN transistor:
- $V_{BE\,SAT} \approx 750$ mV
- $\beta = 110$

MCU:
- $I_{OH} = 25$ mA
- $V_{OH} \approx V_{DD} = 3.3$ V

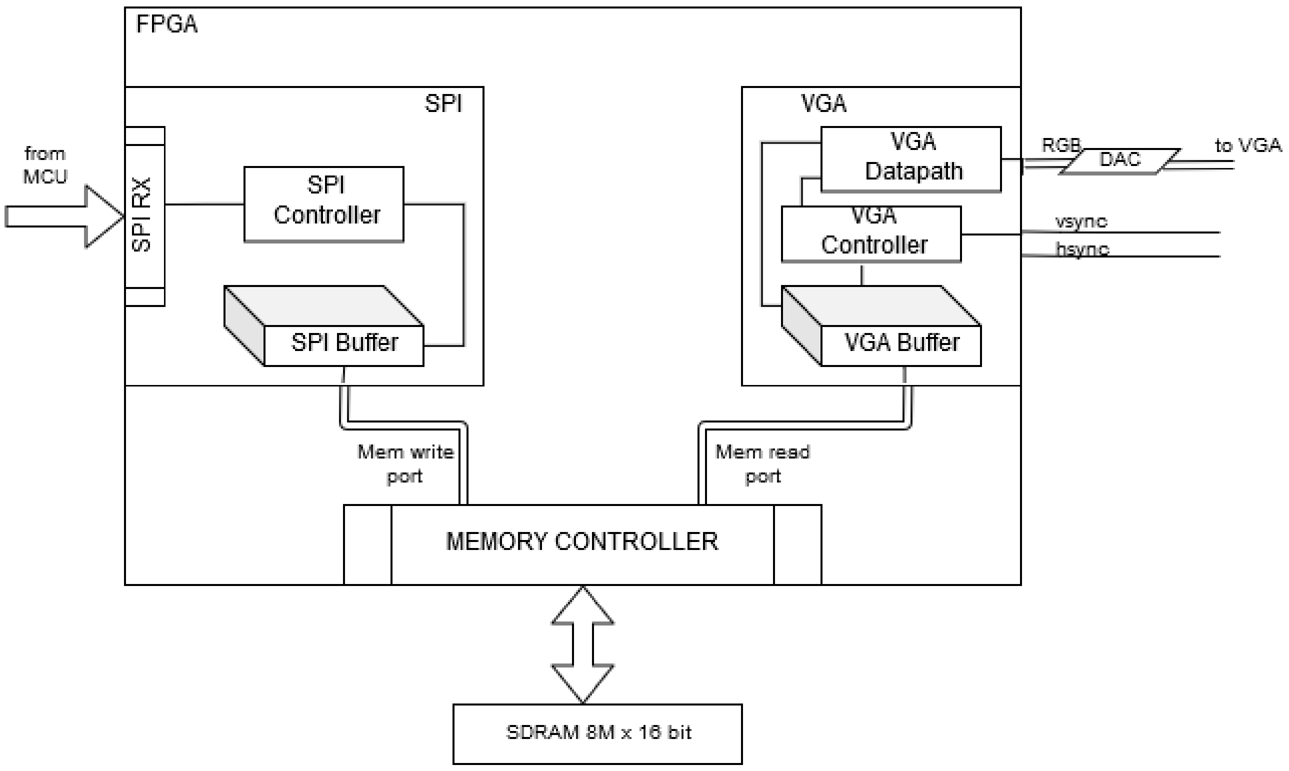
PIEZO:
- $I_C = 20$ mA

$$Rb \ll \frac{\beta}{I_C}(V_{OH} - V_{BE\,SAT}) = \frac{110}{0.02\,A}(2.55\,V) \approx 14\,K\Omega \quad \text{Upper bound}$$

$$Rb > \frac{V_{OH} - V_{BE\,SAT}}{I_{OH}} = \frac{2.55\,V}{0.025\,A} = 102\,\Omega \quad \text{Lower bound}$$

Figure - Low side transistor configuration

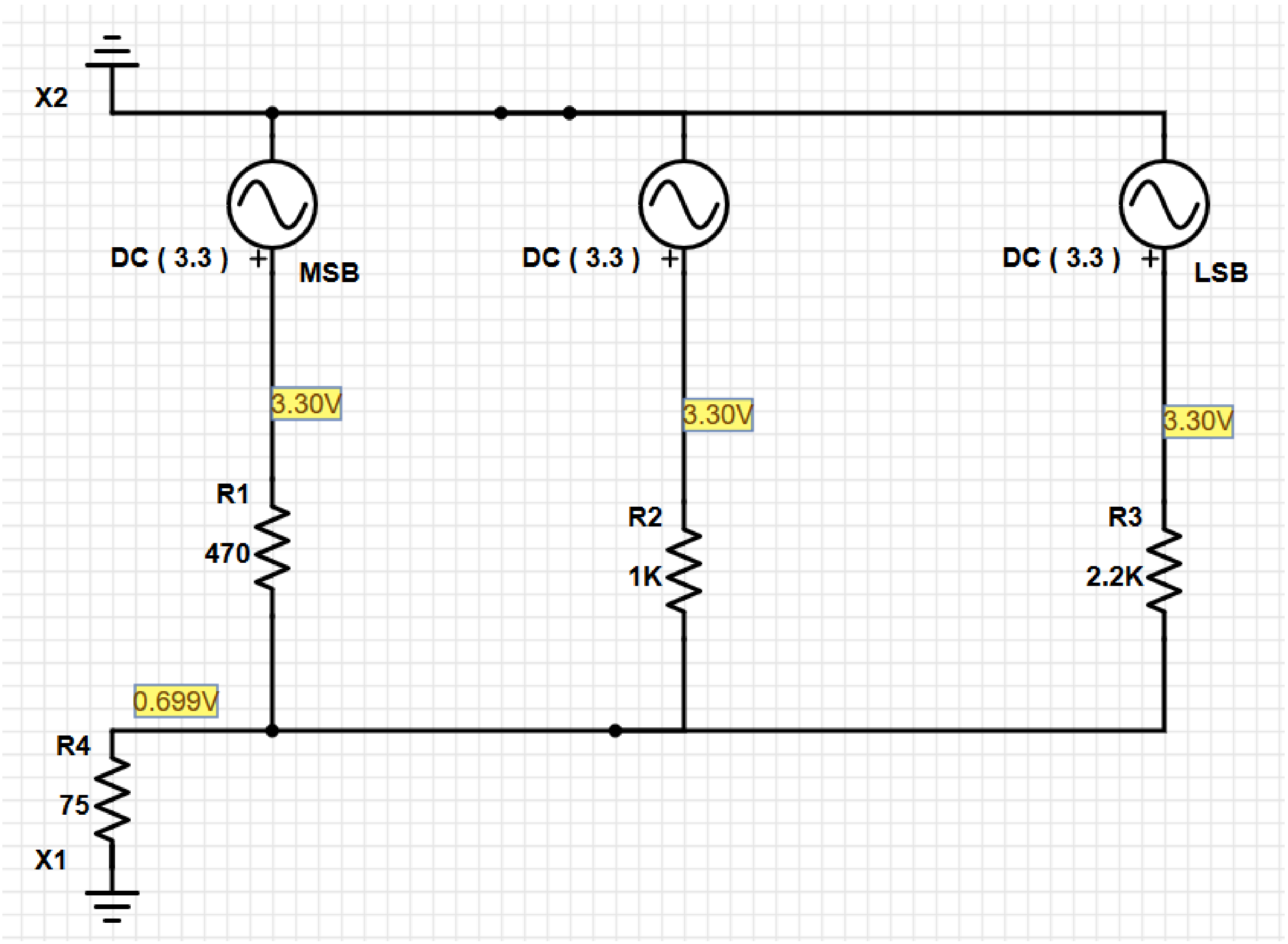
FPGA side:
The selected FPGA is Xilinx Spartan-3e xc3s500e, containing 500k logic elements. The external oscillator adopted to synchronize the circuit has a frequency of 50 MHz. Internally, the FPGA doubles this frequency exploiting PLLs, so almost the entire digital circuit implemented is driven by a 100 MHz clock.



- SPI
An SPI slave receiver (***SPI RX***) has been implemented. It is in charge of updating the shift register with the SPI input bit. When a word is completely received, it is sent to the SPI controller. The ***SPI controller*** has the role to decide what the received word must be used for. In general there are two types of words: data and commands. The first word must be a command, the interpretation of the following data is based on that command. A command should basically indicate whether the following data are referred to a configuration register of the entire device or to the display buffer. At the moment all the configuration registers are hardwired, so the controller is quite useless and it always interpret the incoming words as data for the screen. Data words for the screen are stored into an ***SPI Buffer***, then when the buffer is almost full it is discharged by writing each data word to the memory, through the memory controller.
In conclusion, this part of the design receives the entire display’s pixels from the MCU and writes them into the SDRAM.

- VGA and DAC
The ***VGA Controller*** is the arbiter of this part of design. In fact, it coordinates every operation regarding the VGA. It is responsible for the output of the Horizontal Sync signal and the Vertical Sync signal, also it informs the other components whether the display is enabled, i.e. pixels must be printed onto the screen, and whether an entire page has been completely sent to VGA.
Each time the ***VGA Buffer*** is empty, it loads a row of pixels from the memory. It is discharged by the ***VGA Datapath***, which at each clock cycle output the RGB signals according to the pixel to be

printed. Also, when an entire screen has been transmitted the buffer is flushed.
The signals containing RGB information are connected to a binary weighted ***DAC*** circuit. Resistors' chosen values provide an output voltage within the range 0 – 0.7 V, as required by VGA standards. Also, 3 digital input bit have been chosen, so 8 different analog voltages can be output, resulting in 8 different colors. The implemented circuit is here represented.



- Memory controller
The Memory controller manages the SDRAM chip. This component handles the SDRAM control signals including: data, address, clock, command (cke, cs, cas, ras), lqm and uqm (data masking). It provides the rest of digital circuits with two memory interfaces, one for reading and one for writing from/to a specific memory location. It also sends refresh requests when necessary.
Reads are implemented in burst, if more than one data is requested. Writes, on the other hand, must be singularly performed. The SDRAM clock is at 100 MHz as the rest of the circuit, but it is phase shifted to ensure enough setup time on the other input signals.
Finally, there is not an explicit memory arbiter but write requests have higher priority than reads. This was possible in this application because the write requests are much less frequent than read requests and because they fit in the interval that goes from one read request to the next one.