Laboratory 2 report

Cédric Hölzl Antoine Brunner

March 2020

1 VHDL system design

In the first part of the laboratory, we had to implement the statistic computations, the level adjuster in VHDL.

For the statistics computation, we didn't do anything incredible ;). In the component, there are three registers: one for the maximum, one for the minimum, and one for the sum. Those registers are updated with the value from *pix_data* when the signal *valid* is 1. They are reset when the signal *pix_sof* is 1, so that the statistics for the new frame can be computed.

The level adjuster was just a matter of finding a formula that does what we want. The component takes as input a 14-bit value that is in the range $[raw\_min, raw\_max]$, and we would like to remap those values to the full range $[0, 2^{14} - 1]$. If we were able to use floating point arithmetic, what we would do the following:

$$x \mapsto \frac{x - min}{max - min} \cdot (2^{14} - 1)$$

That is, we would first divide $x - min$ by $max - min$ to have a value between 0 and 1. Then, we would multiply that by $2^{14} - 1$ to get the result. But since we are using integer arithmetic, we cannot perform the first division in this way. The trick is to first multiply by $2^{14} - 1$, which results in a 27-bit value, and then make an integer division by $max - min$.

Once we realized that little trick, the rest was a matter of translating it to VHDL, which was fairly easy, since we had already been given the division component.

2 C application design

As for the first lab, the C wasn't too extensive, we used IORD/IOWR to interact with the right registers (or bits) for the different function: in one case writing a value, in another one checking a bit and the last one looping while a bit is 1. We also completed the main loop of the application, with it working as follows: While no error occurs capture and wait.

3 QSys system integration

The second part of the lab consisted of connecting all the hardware component together using *QSys*. We didn't encounter any major problems in that part. Some small issues that we had was that we were setting wrong directions for some signals, but that wasn't really hard to fix. Note

that we used the automatic memory mapping from QSys to let him decide where our components were mapped in memory. The memory mapping was then exported to the software part through the file system.h. Figure 1 shows how that mapping was chosen by QSys and exported to the file *system.h*.

Figure 1: At the top, the QSys editor. At the bottom, the *system.h* file that was automatically generated. It can be seen that the memory mapping was taken by QSys.

Another problem that we had was that we forgot to instantiate the components of the system in the VHDL entity that *QSys* generated, because we initialy thought it was also done automaticly.

Results

In this section we just present a few images that we captured using the thermal camera, in Figure 2

Figure 2: The images that we captured using the thermal camera. From left to right: Face with glasses, Glass Bottles, Inside a Computer with CPU and GPU (where we can see on the left a column of chokes and capacitors and on the right a crystal both of them being a major source of heat. We can also see on the bottom right an SSD).

4 Appendix: Code

In this appendix, we have put the code that implements what was described in the previous sections, if you prefer to read from the PDF. In order not to make the report too long, we have only included the changes that we made, not the full files.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity lepton_stats is
    port(
        clk        : in  std_logic;
        reset      : in  std_logic;
        pix_data   : in  std_logic_vector(13 downto 0);
        pix_valid  : in  std_logic;
        pix_sof    : in  std_logic;
        pix_eof    : in  std_logic;
        stat_min   : out std_logic_vector(13 downto 0);
        stat_max   : out std_logic_vector(13 downto 0);
        stat_sum   : out std_logic_vector(26 downto 0);
        stat_valid : out std_logic);
end lepton_stats;

architecture rtl of lepton_stats is

    -- The accumulated sum, min and max of the pixel values
    signal curr_min : unsigned(13 downto 0);
    signal curr_max : unsigned(13 downto 0);
    signal curr_sum : unsigned(26 downto 0);

    -- The next value of the registers
    signal next_min : unsigned(13 downto 0);
    signal next_max : unsigned(13 downto 0);
    signal next_sum : unsigned(26 downto 0);

begin

    -- This is the synchronous transition logic
    transition_logic : process(clk, reset)
    begin
        if reset = '1' then
            curr_sum <= (others => '0');
            curr_min <= (others => '0');
            curr_max <= (others => '0');
        elsif rising_edge(clk) then
            curr_min <= next_min;
```

```
            curr_max <= next_max;
            curr_sum <= next_sum;
        end if;
    end process;

    -- This is the combinatorial transition logic
    next_min <=
        curr_min when pix_valid = '0' else
        unsigned(pix_data) when pix_sof = '1' else
        curr_min when unsigned(pix_data) >= curr_min else
        unsigned(pix_data);

    next_max <=
        curr_max when pix_valid = '0' else
        unsigned(pix_data) when pix_sof = '1' else
        curr_max when unsigned(pix_data) <= curr_max else
        unsigned(pix_data);

    next_sum <=
        curr_sum when pix_valid = '0' else
        unsigned((26 downto 14 => '0') & pix_data) when pix_sof = '1' else
        curr_sum + unsigned((26 downto 14 => '0') & pix_data);

    -- This is the synchronous output logic
    output_logic : process(clk, reset)
    begin
        if rising_edge(clk) then
            stat_valid <= pix_eof;
        end if;
    end process;

    -- This is the combinatorial output logic
    stat_min <= std_logic_vector(curr_min);
    stat_max <= std_logic_vector(curr_max);
    stat_sum <= std_logic_vector(curr_sum);

end rtl;
```

Listing 1: The code written in lepton_stats.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity level_adjuster is
    port(
        clk             : in  std_logic;
```

```
        raw_pixel      : in  std_logic_vector(13 downto 0);
        raw_max        : in  std_logic_vector(13 downto 0);
        raw_min        : in  std_logic_vector(13 downto 0);
        raw_sum        : in  std_logic_vector(26 downto 0);
        adjusted_pixel : out std_logic_vector(13 downto 0));
end level_adjuster;

architecture rtl of level_adjuster is
    component lpm_divider
        port(
            clock    : in  std_logic;
            denom    : in  std_logic_vector(13 downto 0);
            numer    : in  std_logic_vector(27 downto 0);
            quotient : out std_logic_vector(27 downto 0);
            remain   : out std_logic_vector(13 downto 0));
    end component;

    -- Intermediate signals needed by the divider
    signal numer : std_logic_vector(27 downto 0);
    signal denom : std_logic_vector(13 downto 0);
    signal quot  : std_logic_vector(27 downto 0);

begin

    -- Computation of the intermediate signals
    numer <= std_logic_vector((13 downto 0 => '1') * (unsigned(raw_pixel) -
    ↪  unsigned(raw_min)));
    denom <= std_logic_vector(unsigned(raw_max) - unsigned(raw_min));

    -- We compute the remaineder of (x - min) / (max - min)
    divider : lpm_divider port map(
        clock => clk,
        numer => numer,
        denom => denom,
        quotient => quot,
        remain => open
    );

    -- And we only keep the LSB of the quotient (we know the MSB must be 0)
    adjusted_pixel <=
        (adjusted_pixel'range => '0') when denom = (denom'range => '0') else
        quot(13 downto 0);

end rtl;
```

Listing 2: The code written in level_adjuster.vhd

```
 do{
    lepton_start_capture(&lepton);
    lepton_wait_until_eof(&lepton);
}while(lepton_error_check(&lepton));
```

Listing 3: The code written in app.c

```
void lepton_start_capture(lepton_dev *dev) {
        IOWR_16DIRECT(dev->base, LEPTON_REGS_COMMAND_OFST, 0x1);
}

bool lepton_error_check(lepton_dev *dev) {
        return (IORD_16DIRECT(dev->base, LEPTON_REGS_STATUS_OFST) & 0x2) != 0;
}

void lepton_wait_until_eof(lepton_dev *dev) {
        while(IORD_16DIRECT(dev->base, LEPTON_REGS_STATUS_OFST) & 0x1);
}
```

Listing 4: The code written in lepton.c