DIPLOMARBEIT

FPGA-Basierte Studie moderner Computersysteme am Beispiel RISC-V: YARM

Höhere Technische Bundeslehr- und Versuchsanstalt Anichstraße

Abteilung

Elektronik und technische Informatik

Ausgeführt im Schuljahr 2019/20 von:

Armin Brauns 5AHEL

Daniel Plank 5BHEL

Betreuer/Betreuerin:

Dipl.-Ing. Christoph Schönherr

Projektpartner: IT-Syndikat, Verein zur Förderung des freien Zugangs zu technischer Fort- und Weiterbildung jeglicher Art

Ansprechpartner: Herr David Oberhollenzer

Innsbruck, am 27. März 2020

Abgabevermerk:

Datum:

Betreuer/in:

Gendererklärung

Aus Gründen der besseren Lesbarkeit wird in dieser Diplomarbeit die Sprachform des generischen Maskulinums angewendet. Es wird an dieser Stelle darauf hingewiesen, dass die ausschließliche Verwendung der männlichen Form geschlechtsunabhängig verstanden werden soll.

Kurzfassung/Abstract

Projektergebnis

Inhaltsverzeichnis

1 Aufgabenstellung

1.1 Daniel Plank

2 Planung

2.1 Grobdesign

2.1.1 Aufgabenstellung

Zur Klärung immerwiederkehrender Fragen im Umfeld des Vereins IT-Syndikat in Innsbruck sollen Beispiele und Dokumentation der Funktion moderner Prozessorperipherie und der Kommunikation selbiger mit dem Prozessorkern als MMIO [1] Angefertigt werden. Diese Dokumentation soll möglichst für alle Prozessorarchitekturen gültig und daher Architekturunabhängig sein. Die Dokumentation soll für Personen mit tieferem Verständniss von Hardware, sowie für Anfänger hilfreiche Aussagen über die Funktionswiese der Hardware liefern, welche sich aus einer seriellen Schnittstelle mit TIA-/EIA-232 Pegeln, einem Framebuffer[2] und einem Soundbuffer[3].

2.1.2 Umsetungsbeschreibung

Zu aller erst müssen, um die ziele Verstehen zu können, die benötigten Unterlagen beschaffen werden. Diese können in Papierform oder Digital vorhanden sein. Letztere dürften leichter in diesem Industriezweig aufzutreiben sein, jedoch muss mehr Acht gegeben werden auf die Korrektheit der Dokumente. Nach der Beschaffung der Dokumente sollen Beispielschaltungen entwickelt werden, damit die Funktionsweise auch tatsächlich verstanden werden kann. Dokumentationen ohne sinnvollen Praktischen Hintergrund sind meist recht unanschaulich. Nach Entwicklung der Schaltungen sollen diese sinnvoll begründet werden und dann Dokumentiert werden. Die Dokumentation soll mindestens aus Schaltungsbeschreibung, einer generellen Bauteilbeschreibung und den Ideen hinter der aktuellen Umsetzung bestehen.

[1] MMIO... Memory Mapped I/O

[2] Framebuffer...Ein Speicherbereich in welchen ein Bild geladen werden kann, welches dann ausgegeben werden

[3] Ähnlich einem Framebuffer nur dass PCM-Audio anstatt Bildern ausgegeben wird

2.2 Meilensteine

Anmerkung zu den Daten Die Daten wurden den Vorgaben entsprechend gewählt, so wurden mindestens 2 Meilensteine in das Sommersemester hineingelegt. Die Meilensteine wurden auf Dienstage gelegt, da an diesem Wochentag bekannt ist dass der Betreuer sich im Gebäude der HTBLuVA befindet. Die Meilensteintermine wurden möglichst Äquidistant über den zur Verfügung stehenden Zeitbereich verteilt.

2.2.1 1. Meilenstein - Beschaffung der Unterlagen

Datum: 2019-11-19

Der 1. Meilenstein beschäftigt sich mit der Beschaffung der Benötigten Unterlagen auf welchen die weitere Dokumentation basieren soll. Diese Unterlagen sollen den Aufbau einer Seriellen Schnittstelle beschreiben, den Aufbau eines Frame-Buffers und den Aufbau einer Sound-Karte. Die Unterlagen sollen auch Beispielschlatungen beinhalten.

Die Folgenden Tests zur Verifikation der bisherigen Arbeit wurden dafür definiert:

1. Testname: Testinhalt

FAILURE	SUCCESS

2.3 2. Meilenstein - Serielle Schnitstelle

Datum: 2019-01-10

Der 2. Meilenstein beschäftigt sich mit der Seriellen Schnittstelle, ihrer Schaltung und der Verifikation selbiger. Die Dokumentation ihrer Funktionsweise soll zu einem späteren Zeitpunkt erfolgen.

2.4 3. Meilenstein - Soundbuffer

Datum: 2019-02-04

Der 3. Meilenstein beschäftigt sich mit der Schaltungsentwicklung des Soundbuffers, der Funktionsweise eines Soundbuffers und der Schaltungsverifikation der entwickelten Schaltung.

2.5 4. Meilenstein - Dokumentation

Datum: 2019-03-10

Der 4. Meilenstein beschäftigt sich mit der Dokumentation der bisherigen Bauteile, namlich des Soundbuffers und der Seriellen Schnittstelle. Zu diesem Meilenstein soll die Dokumentation der Funktionsweise und der Implementation gefertigt werden.

2.6 4. Meilenstein - Framebuffer

Datum: 2019-04-14

Der 5. Meilenstein beschäftigt sich mit der Dokumentation eines Framebuffers und der rein theoretischen implementierung selbigens.

2.7 Zeitabschätzung

2.8 Aufwandsabschätzung

Die Aufwandsabschätzung beinhaltet nur die nach offiziellem Einreichen der Diplomarbeit zu vollziehenden Arbeiten.

2.8.1 Stundenabschätzung der Teilaufgaben

Teilbereich	Aufgabe	Dauer[Stunden]
Allgemeines	Informationsbeschaffung zur Funktionsweise	2
Allgemeines	Bestückung der Backplane	1
Allgemeines	Zeichnen der Testplatinen	6
Allgemeines	Auswahl der Bauelemente der Testplatinen	3
Allgemeines	Bestücken der Testplatinen	3
Allgemeines	Verifikation der Backplane	2
Soundpuffer	Informationsbeschaffung zur Funktionsweise	2
Soundpuffer	Auswahl der Bauelemente	1
Soundpuffer	Verifikations-Schaltungsenticklung	4
Soundpuffer	Steckbrettaufbau und Fehlersuche	7
Soundpuffer	Steckbrettaufbau Test und Verifikation	3
Grafikpuffer	Infromationsbeschaffung zur Funktiosweise	4
Grafikpuffer	Schaltungsplanung zum Pufferwechsel	5
Grafikpuffer	Verifikations-Schalktungsentwicklung	7
Grafikpuffer	Auswahl der Bauelemente	1.5
Grafikpuffer	Steckbrettaufbau und Fehlersuche	10
Soundpuffer	Fertigen der Platinenzeichnung	2
Grafikpuffer	Fertigen der Platinenzeichnung	3
Soundpuffer	Fertigen der Platine	3.5
Grafikpuffer	Fertigen der Platine	2
Soundpuffer	Bestücken der Platine	1.5
Grafikpuffer	Bestücken der Platine	1.5
Soundpuffer	Verifikation mittels Testplatine	1.5
Grafikpuffer	Verifikation mittels Testplatine	1.5
Allgemeines	Entwicklung von Beispielbedienungen mittels Testplatinen	3
Mikrokontrolle	Entwicklung von Interface zwischen Backplane und Mikrokontroller	2
Software	Entwicklung von Demosoftware mit Soundpuffer	4
Software	Entwicklung von Demosoftware mit Grafikpuffer	7
Allgemeines	Allgemeiner Fehlerpuffer während der Entwicklung	15h
SUMME	**SUMME**	109
Dokumentation	Dokumentation	60
SUMME	**SUMME**	169

Tabelle 1: Stundenabschätzung Plank Daniel

Die Dokumentation wird mit 60h beschätzt, da diese die Gesamte Funktionsweise der Hardware abdeckt, und mehrere Messungen der Hardware beeinhlatet.

Teil I

A short introduction to VHDL

Designing a processor is a big task, and it's easiest to start very small. With software projects, this is usually in the form of a "Hello World" program - we will be designing a hardware equivalent of this.

3 Prerequisites

Other than a text editor, the following Free Software packages have to be installed:

`ghdl` **[1]** to analyze, compile, and simulate the design

`gtkwave` **[2]** to view the simulation waveform files

`yosys` **[3]** to synthesize the design

`ghdlsynth-beta` **[4]** to synthesize the design

`nextpnr-xilinx` **[5]** to place and route the design

`Project X-Ray` **[6]** for FPGA layout data and bitstream tools

`openFPGALoader` **[7]** to load the bitstream onto the FPGA

4 Creating a design

A simple starting design is an up/down counter. The following VHDL code describes the device:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity counter is
6      port (
7          clk       : in std_logic;
8          reset     : in std_logic;
9          enable    : in std_logic;
10         direction : in std_logic;
11
12         count_out : out std_logic_vector(7 downto 0)
```

```
    );
end counter;

architecture behaviour of counter is
    signal count : unsigned(7 downto 0) := (others => '0');
begin
    proc: process(clk)
    begin
        if reset then
            count <= (others => '0');
        elsif rising_edge(clk) and enable = '1' then
            if direction = '1' then
                count <= count + 1;
            else
                count <= count - 1;
            end if;
        end if;
    end process;

    count_out <= std_logic_vector(count);
end behaviour;
```

counter.vhd

In order to test this design, a test bench has to be created:

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

entity counter_tb is
end counter_tb;

architecture test of counter_tb is
    signal clk, reset, enable, direction : std_logic;
    signal s_count_out : std_logic_vector(7 downto 0);

    signal count_out : unsigned(7 downto 0);
begin
    uut: entity work.counter
        port map (
            clk       => clk,
            reset     => reset,
            enable    => enable,
            direction => direction,

            count_out => s_count_out
        );

    count_out <= unsigned(s_count_out);

    simulate: process
        begin
            clk <= '0';
            reset <= '1';
            enable <= '0';

            wait for 30 ns;
            assert count_out = 0;

```

```
35              reset <= '0';
36
37              clk <= '0';
38              wait for 10 ns;
39              clk <= '1';
40              wait for 10 ns;
41
42              assert count_out = 0;
43
44              enable <= '1';
45              direction <= '0';
46
47              clk <= '0';
48              wait for 10 ns;
49              clk <= '1';
50              wait for 10 ns;
51
52              assert count_out = 255;
53
54              direction <= '1';
55
56              clk <= '0';
57              wait for 10 ns;
58              clk <= '1';
59              wait for 10 ns;
60
61              clk <= '0';
62              wait for 10 ns;
63              clk <= '1';
64              wait for 10 ns;
65
66              assert count_out = 1;
67
68              wait for 30 ns;
69              wait;
70          end process;
71  end test;
```

counter_tb.vhd

5 Simulating a design

```
# analyze the design files
ghdl -a --std=08 *.vhd
# elaborate the test bench entity
ghdl -e --std=08 counter_tb
# run the test bench, saving the signal trace to a GHW file
ghdl -r --std=08 counter_tb --wave=counter_tb.ghw
# open the trace with gtkwave (using the view configuration in counter_tb.gtkw)
gtkwave counter_tb.ghw counter_tb.gtkw
```

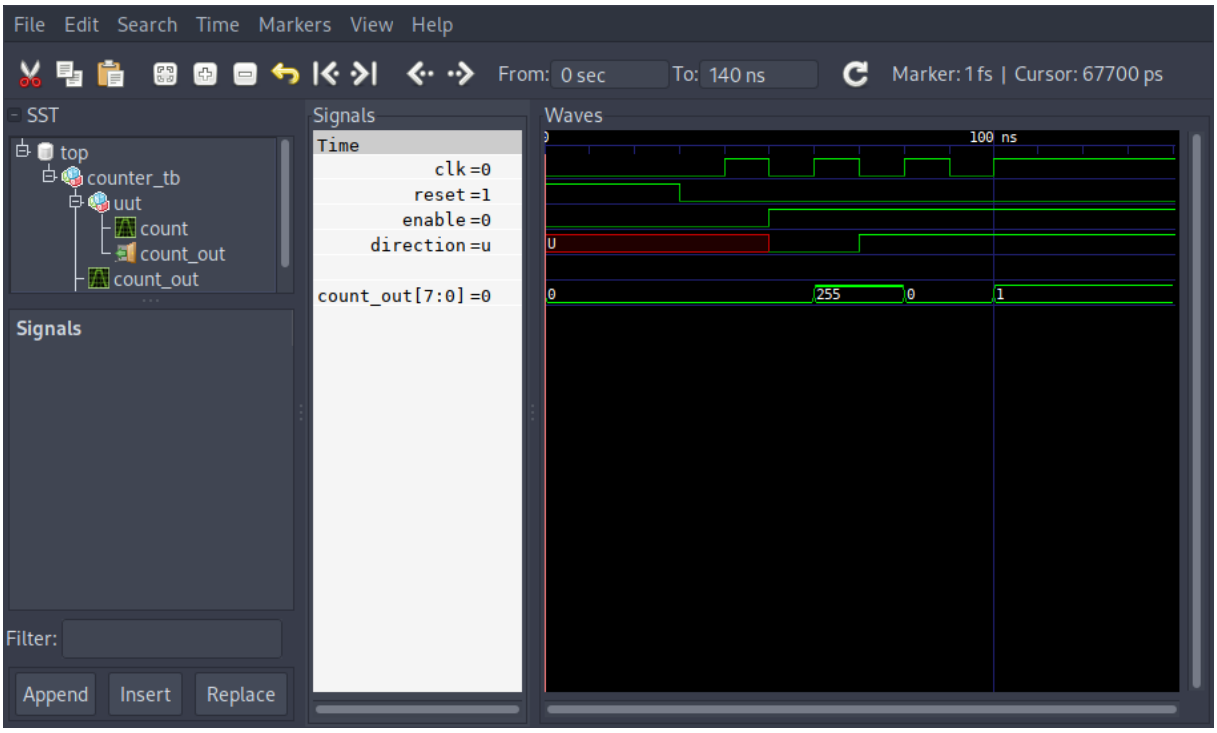

Abbildung 1: Screenshot of the resulting waveform in GTKWave

6 Synthesizing a design

An additional Xilinx Design Constraints (XDC) file is required to assign the signals to pins on the FPGA:

```
1  set_property LOC D9  [get_ports clk]
2  set_property LOC C9  [get_ports reset]
3  set_property LOC A8  [get_ports enable]
4  set_property LOC C11 [get_ports direction]
5
6  set_property LOC F6  [get_ports count_out[0]]
7  set_property LOC J4  [get_ports count_out[1]]
8  set_property LOC J2  [get_ports count_out[2]]
9  set_property LOC H6  [get_ports count_out[3]]
10 set_property LOC H5  [get_ports count_out[4]]
11 set_property LOC J5  [get_ports count_out[5]]
12 set_property LOC T9  [get_ports count_out[6]]
13 set_property LOC T10 [get_ports count_out[7]]
14
15 set_property IOSTANDARD LVCMOS33 [get_ports clk]
16 set_property IOSTANDARD LVCMOS33 [get_ports reset]
17 set_property IOSTANDARD LVCMOS33 [get_ports enable]
18 set_property IOSTANDARD LVCMOS33 [get_ports direction]
19 set_property IOSTANDARD LVCMOS33 [get_ports count_out[0]]
20 set_property IOSTANDARD LVCMOS33 [get_ports count_out[1]]
21 set_property IOSTANDARD LVCMOS33 [get_ports count_out[2]]
22 set_property IOSTANDARD LVCMOS33 [get_ports count_out[3]]
23 set_property IOSTANDARD LVCMOS33 [get_ports count_out[4]]
24 set_property IOSTANDARD LVCMOS33 [get_ports count_out[5]]
25 set_property IOSTANDARD LVCMOS33 [get_ports count_out[6]]
26 set_property IOSTANDARD LVCMOS33 [get_ports count_out[7]]
```

counter.xdc

```
# synthesize with yosys
yosys -m ghdl.so -p '
    ghdl --std=08 counter.vhd -e counter;
    synth_xilinx -flatten;
    write_json counter.json'
# place and route the design with nextpnr
nextpnr-xilinx --chipdb xc7a35tcsg324-1.bin --xdc counter.xdc --json counter.json --
    fasm counter.fasm
# convert the FPGA assembly to frames
fasm2frames.py --part xc7a35tcsg324-1 counter.fasm counter.frames
# convert the frames to a bitstream
xc7frames2bit --part-name xc7a35tcsg324-1 --frm-file counter.frames --output-file
    counter.bit
# upload the bitstream to the FPGA
openFPGALoader -b arty counter.bit
```

The current value of the counter is displayed in binary on the eight LEDs on the board. When switch 0 (enable) is in the high position, the counter can be advanced using button 0, with the direction set by switch 1. Button 1 resets the counter to zero.

Teil II

Meta

7 History

The project started out with the desire to build a CPU from scratch. Examples such as The NAND Game[8] and Ben Eater's Breadboard Computer series[9] served as inspirations and guidance during development.

At first, a design similar to Ben Eater's consisting solely of discrete integrated circuits was considered, but soon discarded in favor of an FPGA-based design. Designing the logic alone was a difficult task, implementing it in discrete hardware would have pushed the project far over the allotted maximum development time.

RISC-V was chosen as the instruction set architecture for the processor. Its modular design with a very small base instruction set make it easy to implement a basic processor that is still fully compatible with existing software and toolchains.

As a starting point, a Terasic DE0 development board[4] containing an Altera Cyclone III[5] FPGA was borrowed from the school's inventory. It was used to implement a first version of the core.

The only method of synthesis for Altera devices is to use the proprietary Quartus IDE. However, the last version of Quartus to support the Cyclone III series of FPGAs (version 13.1) had already been out of date for several years at the start of the project. Because of this and the increasing resource demand of the developing core, an Arty A7-35T development board[6] with a Xilinx Artix-7[7] FPGA was ordered from Digilent.

The two FPGAs compare as follows:

	Altera EP3C16	Xilinx XC7A35T
Logic Elements	15000	33280
Multipliers	56	90
Block RAM (kb)	504	1800
PLLs	4	5
Global clocks	20	32

The periphery on the development boards:

	Terasic DE0	Digilent Arty A7-35T
Switches	10	4
Buttons	3	4
LEDs	10 + 4x 7-segment	4 + 3 RGB
GPIOs	2x 36	4x PMOD + chipKIT
Memory	8MB SDRAM	256MB DDR3L
Others	SD card, VGA	Ethernet

While the Digilent board offers fewer IO options, the DDR3 memory can be interfaced using Free memory cores and allows for much larger programs to be loaded, possibly even a full operating system. The missing VGA port has been substituted by a HDMI-compatible DVI interface that is accessible through one of the high-speed PMOD connectors.

[4] https://www.terasic.com.tw/cgi-bin/page/archive.pl?No=364

[5] https://www.intel.com/content/www/us/en/products/programmable/fpga/cyclone-iii.html

[6] https://store.digilentinc.com/arty-a7-artix-7-fpga-development-board-for-makers-and-hobbyists/

[7] https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html

8 Tooling

FPGA design is done using a Hardware Description Language (HDL). The two most well-known HDLs are Verilog and VHDL (VHSIC (Very high speed integrated circuit) HDL). As part of our studies at HTL, we exclusively worked with VHDL. For this reason, and because VHDL offers a better type system, it was chosen as the language of choice for the project.

8.1 Vendor Tools

The conventional way to work with FPGA designs is to use the FPGA vendor's development solution for simulation, synthesis and place-and-route. All of these tools are proprietary software specialized to a certain FPGA manufacturer, so a change of hardware also requires changing to a completely different software solution.

Vendor tools are usually free-of-charge for basic usage, but this also means there is no guaranteed support. During the development of this project, several bugs and missing features were found in vendor tools that required workarounds.

8.2 Free Software Tools

A somewhat recent development is the creation of Free Software[8] FPGA toolchains. A breakthrough was achieved by Claire (formerly Clifford) Wolf in 2013 with yosys[3], [10], a feature-complete Verilog synthesis suite for Lattice's `iCE40` FPGA series. Since then, both yosys and place-and-route tools like nextpnr[11] have matured, however Lattice's iCE40 and ECP5 remained the only supported FPGA architectures for place-and-route.

Thus, two obstacles remained for Free toolchains to be viable for this project: synthesizing *from* VHDL code and synthesizing *to* Artix-7 FPGAs. During the development of the project, both of these were solved: Tristan Gingold released ghdlsynth-beta[4], a bridge between GHDL[1] and yosys allowing VHDL to be synthesized just the same as Verilog, and Dave Shah added Xilinx support to nextpnr[5]. The latter was preceded by many months of volunteer work reverse-engineering the Xilinx bitstream format as part of *Project X-Ray*[6].

[8] "Free Software" refers to software that grants its user the freedom to share, study and modify it - see `https://www.fsf.org/about/what-is-free-software`.

With these two pieces in place, the project was switched over to a completely Free toolchain, removing any depencies on vendor tools:

- yosys, with ghdl as a frontend for processing VHDL, is used to synthesize the design
- nextpnr-xilinx, together with the Project X-Ray database, is used for place-and-route
- tools from Project X-Ray are used to convert the routed design to a bitstream
- openFPGALoader is used to transfer the bitstream to the FPGA via JTAG

9 Peripherals

9.1 UART

9.2 DVI graphics

The graphics submodule consists of a VGA timing generator, a text renderer with a font ROM, and a DVI encoder frontend:

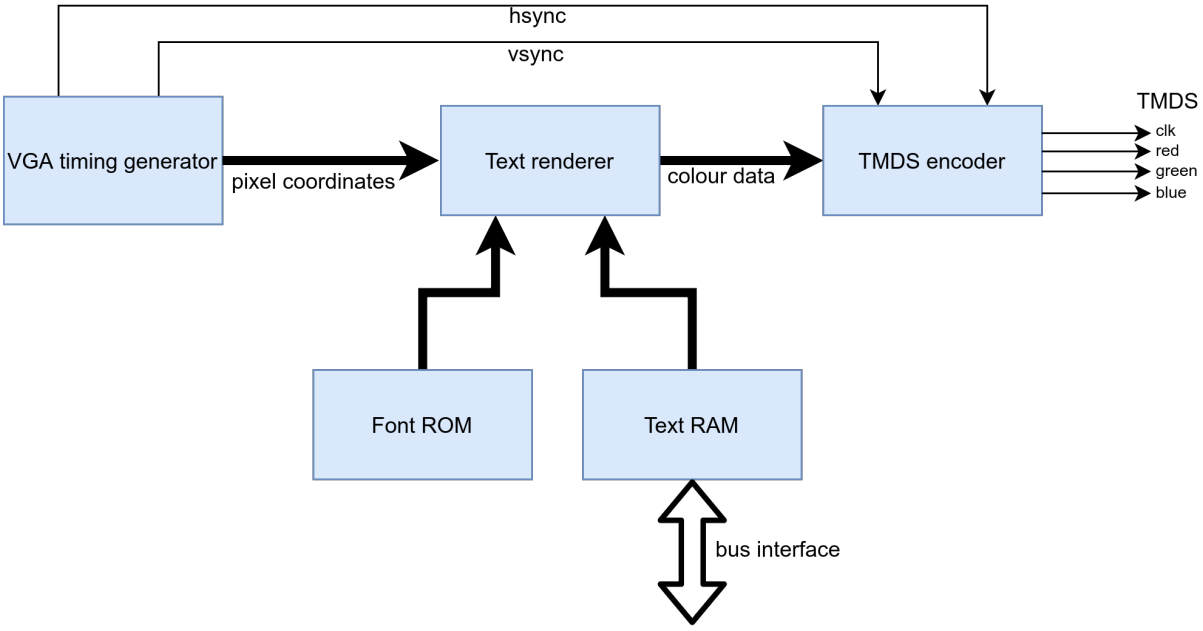

Abbildung 2: Block diagram of the video core

9.2.1 VGA timing

The timing of VGA signals dates back to analog monitors. Even though this original purpose is only very rarely used nowadays, the timing remained the same for analog and digital DVI all the way to modern HDMI.

In analog screens, the electron beams (one for each primary color red, green and blue) scan across the screen a single horizontal line at a time while being modulated by the color values, resulting in a continuous mixture of all three components. When a beam reaches the end of a scanline, it continues outside the visible area for a small distance (the “Front Porch”), is then sent to the beginning of the next line by a pulse of the hsync (Horizontal Sync) signal, and draws the next line after another short off-screen period (the “Back Porch”).

The same applies to vertical timings: after the beam reaches the end of the last line, a few off-screen Front Porch lines follow, then a pulse of the vsync (Vertical Sync) signal sends the beam to the top of the screen, where the first line of the next frame is drawn after several invisible Back Porch lines.

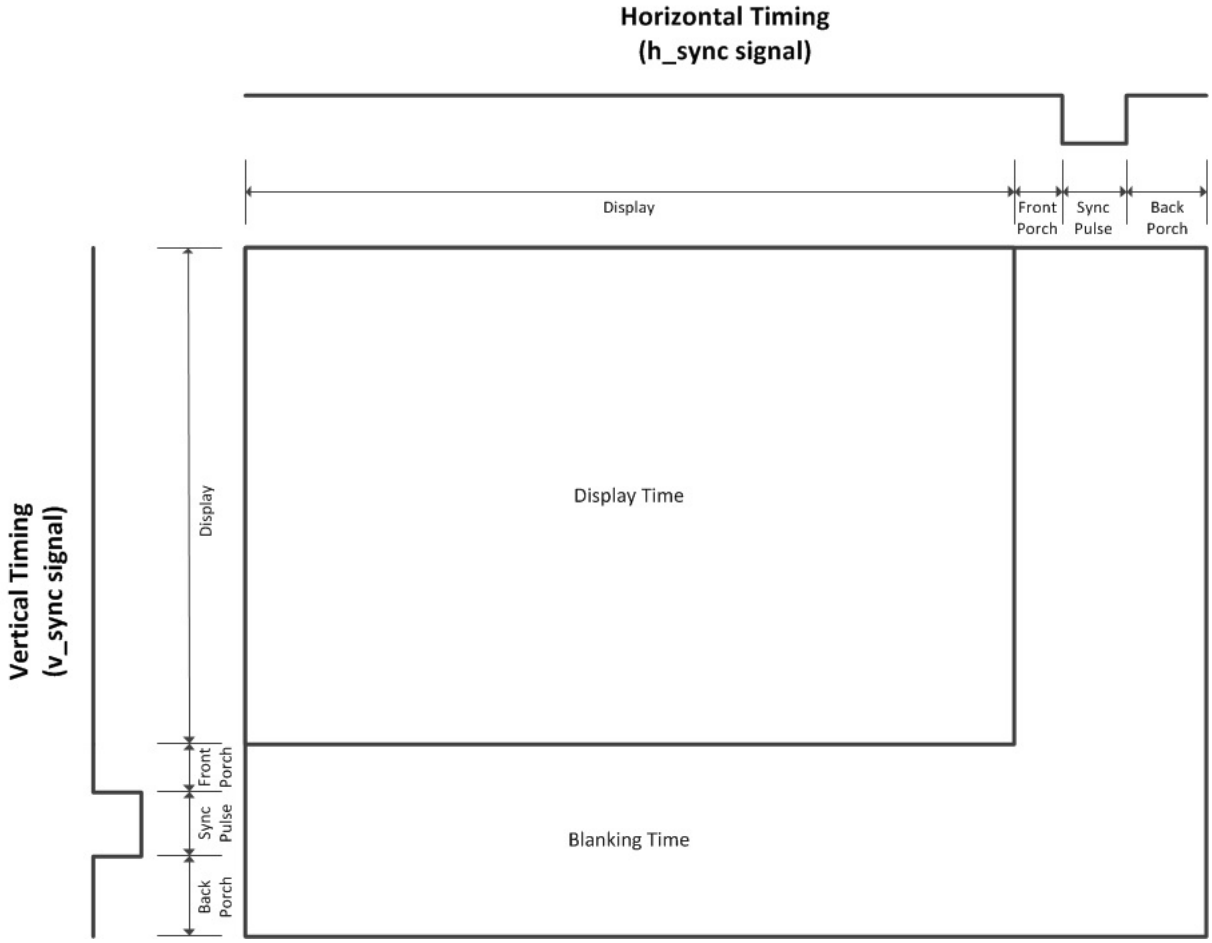

Abbildung 3: Diagram of VGA timing intervals

The VGA timing module generates these hsync and vsync signals, along with a blanking signal (active during any front porch, sync and back porch) and, while in the visible area (i.e. not blanking), the row and column of the current pixel relative to the visible area.

9.2.2 Text renderer

The text renderer converts a logical representation of a character, such as its ASCII code (henceforth referred to as its *codepoint*) to a visual representation (a *glyph*). This conversion is achieved using a *font*, a mapping of codepoints to glyphs.

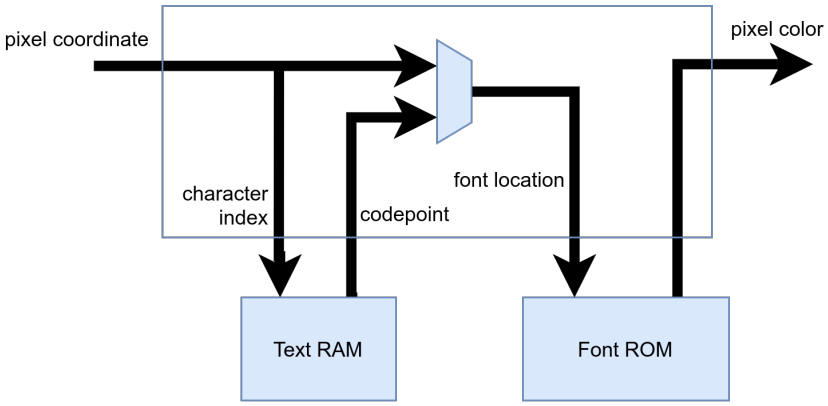

Abbildung 4: Block diagram of the text renderer

First, the current pixel coordinate (created by the VGA timing generator) is split up into two parts: the character index, which specifies the on-screen character the pixel belongs to, and the offset of the pixel in this character. The character index is passed to the text RAM, which contains the codepoint for each on-screen character. This codepoint, along with the pixel offset, is looked up in the font ROM to determine the color of the pixel.

9.2.3 TMDS encoder

DVI and HDMI are serial digital transmission standards. Three data lines (corresponding to red, green, and blue channels) along with a clock line transmit all color information as well as synchronization signals. The encoding used for these signals is Transition-minimized differential signaling (TMDS). It is a kind of 8b/10b encoding (transforming every 8-bit chunk of data into a 10-bit chunk) that is designed to minimize the number of changes of the output signal.

9.3 Ethernet

The Arty development board contains an RJ-45 Ethernet jack connected to an Ethernet PHY, which exposes a standardized media-independent interface (MII) to the FPGA. The LiteEth core[12], which is released under a Free Software license, is used to integrate the Ethernet interface into the SoC.

9.4 WS2812 driver

A hardware driver for WS2812 serially-addressable RGB LEDs is also included in the SoC. It was developed independently as part of the curriculum at HTL and later incorporated into the SoC.

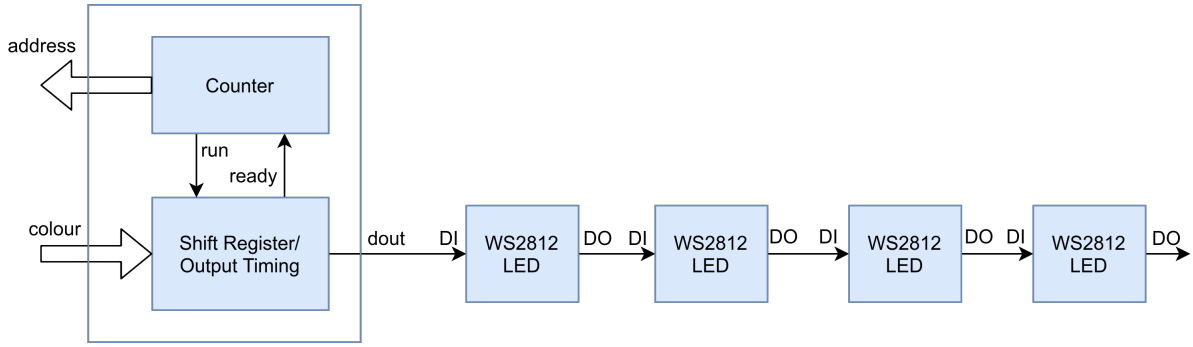

Abbildung 5: Block diagram of the WS2812 driver

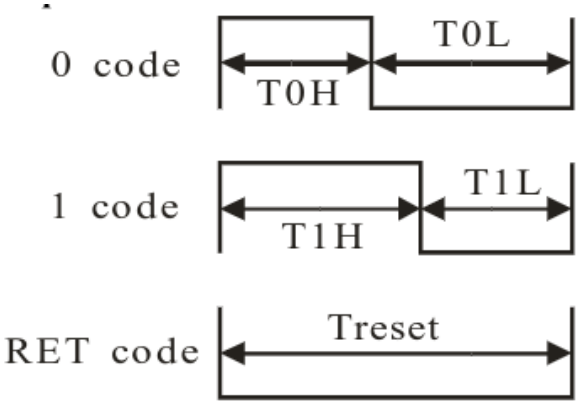

Abbildung 6: Timing diagram for the WS2812 serial protocol

The driver is designed to be attached to external circuitry that provides color data for any given LED index (address). This can either be discrete logic that generates the color value from the address directly, or a memory that stores a separate color value for each address.

The LEDs are controlled using a simple one-wire serial protocol. After a reset (long period of logic 0), the data for all LEDs is transmitted serially in one single blob. Each LED consumes and stores the first 24 bits of the stream and applies them as its color value (8 bits each for red, green, blue), all following bits are passed through unmodified. The second LED thus uses the first 24 bits of the stream it receives, but since the first LED already dropped its data, these are actually the second set of 24 bits of the source data.

Every bit is encoded as a period of logic 1, followed by a period of logic 0. The timing of these sections determines the value, see 6.

The exact timing differs between models, so all periods can be customized using generics in the VHDL entity.

9.5 DRAM

9.6 External Bus

Bridging the internal SoC bus with the external peripheral bus requires a few steps. For one, the external data bus is bidirectional, so tri-state outputs must be used on the FPGA. In addition, the internal bus arbitrates components using addresses alone, while the external bus uses chip enable signals and overlapping address spaces.

Due to a mistake in the adapter board layout, the nibbles of the address and data buses are reversed (MSB to LSB are pins 7 to 0 on the FPGA, but 3 to 0 followed by 7 to 4 on the board). Thanks to the completely arbitrary mapping of FPGA pins, this can be mitigated without using any additional resources.

10 Testing

10.1 RISC-V Compliance Tests

The RISC-V Compliance Test Suite[13] can be used to empirically confirm the correct functionality of a RISC-V processor. It consists of a series of programs that perform some operations related to a specific feature, then write some result data to a memory region. This memory region is then compared to a “golden signature”, which was produced by a processor implementation that is known to be correct.

The initial implementation of the compliance tests uncovered several bugs in the processor core:

- The bitshift instructions (SLL, SRL, SRA, etc.) must, according to the RISC-V standard, only use the lower 5 bits of the second operand as a shift offset. The implementation used all 31 bits instead, causing a test failure.
- Reading a signed value of a size less than 32 bits from memory would not perform proper sign extension. For example, reading a byte value of 0xFF (-1) would result in an expanded machine word of 0x0000_00FF (255) instead of 0xFFFF_FFFF.
- The `SLTIU` (Set less than immediate; unsigned) instruction compares a given register with a constant provided as part of the instruction (the immediate). While the comparison is unsigned, the 12-bit immediate must be sign-extended as if it were a signed integer. The implementation wrongly assumed that the sign-extension should be unsigned as well.
- The Instruction Set Manual specifies exceptions that must be raised when a misaligned memory access occurs. These exceptions were not yet implemented, but since the compliance tests check for them, the functionality was added to make the tests pass.

Since these tests are easily automated, they were added to the GitLab Continuous Integration (CI) configuration. Whenever a new git commit is pushed to GitLab, the tests are run automatically, and any failures are reported to the responsible committer via email.

Teil III

The Core

The core implements the rv32i architecture as specified by the RISC-V standard.

It is constructed according to the traditional RISC pipeline:

Fetch fetches the next instruction from memory.

Abbildung 7: Block diagram of the CPU core

Decode decodes the instruction into its constituent parts. At the same time, operand values are loaded from any required registers.

Execute performs the action required by the instruction, such as math performed by the Arithmetic Logic Unit (ALU) or writing to Control and Status Registers (CSRs).

Memory loads values from or stores values to the system's main memory or interacts with memory-mapped hardware devices.

Writeback stores a potential result value from Execute or Memory stages to the destination register.

11 Overview

12 Control

```
entity control is
    generic (
        RESET_VECTOR : yarm_word
    );
    port (
        clk   : in std_logic;
        reset : in std_logic;

        fetch_enable    : out std_logic;
        fetch_ready     : in std_logic;
        fetch_instr_out : in yarm_word;

        decoder_enable         : out std_logic;
        decoder_instr_info_out : in instruction_info_t;

        registers_data_a : in yarm_word;
        registers_data_b : in yarm_word;

        alu_enable_math : out std_logic;
        alu_math_result : in yarm_word;
        alu_valid       : in std_logic;
        alu_enable_cmp  : out std_logic;
        alu_cmp_result  : in compare_result_t;

        csr_enable           : out std_logic;
        csr_ready            : in std_logic;
        csr_data_read        : in yarm_word;
        csr_increase_instret : out std_logic;

        datamem_enable         : out std_logic;
        datamem_ready          : in std_logic;

        alignment_raise_exc : out std_logic;
```

```
34          alignment_exc_data  : out exception_data_t;
35
36          registers_read_enable  : out std_logic;
37          registers_write_enable : out std_logic;
38
39          -- TRAP CONTROL
40
41          may_interrupt     : out std_logic;
42          -- the stage that will receive an interrupt exception
43          interrupted_stage : out pipeline_stage_t;
44
45          do_trap     : in std_logic;
46          trap_vector : in yarm_word;
47
48          trap_return_vec : in yarm_word;
49          return_trap     : out std_logic;
50
51          -- instruction info records used as input for the respective stages
52          stage_inputs : out pipeline_frames_t
53      );
54  end control;
```

control.vhd

The control unit is responsible for coordinating subcomponents and the data flow between them. Internally, it is based on `instruction_info_t` structures, which contain all the information required to pass an instruction along the different pipeline stages. Before the fetch stage, when an instruction is first scheduled, it contains only the instruction's address (because nothing else is known about it). Then, information is added incrementally by the different stages.

13 Decoder

```
1   entity decoder is
2       port (
3           clk    : in std_logic;
4           enable : in std_logic;
5
6           async_addr_rs1 : out register_addr_t;
7           async_addr_rs2 : out register_addr_t;
8
9           alu_muxsel_a    : out mux_selector_t;
10          alu_muxsel_b    : out mux_selector_t;
11          alu_muxsel_cmp2 : out mux_selector_t;
12
13          csr_muxsel_in : out mux_selector_t;
14
15          instr_info_in  : in instruction_info_t;
16          instr_info_out : out instruction_info_t;
17
18          raise_exc : out std_logic;
19          exc_data  : out exception_data_t
20      );
```

```
21 end decoder;
```

decoder.vhd

The decoder receives an instruction and interprets it. Among others, it determines

- The source and destination register addresses
- The pipeline stages that need to be run for the instruction
- The ALU operation, if any
- Whether the instruction should branch, and if so, under what condition

14 Registers

```
1  entity registers is
2      port (
3          clk   : in std_logic;
4
5          read_enable  : in std_logic;
6          write_enable : in std_logic;
7
8          addr_a : in register_addr_t;
9          addr_b : in register_addr_t;
10         addr_d : in register_addr_t;
11
12         data_a : out yarm_word;
13         data_b : out yarm_word;
14         data_d : in yarm_word
15     );
16 end registers;
```

registers.vhd

The registers store the 32 general-purpose values required by rv32i (each 32-bit wide). They are accessible through two read ports and one write port. As specified by the RISC-V standard, the first register (`x0`) is hard-wired to 0, and any writes to it are ignored.

15 Arithmetic and Logic Unit (ALU)

```vhdl
entity alu is
    port (
        clk : in std_logic;

        enable_math : in std_logic;
        valid       : out std_logic;
        operation   : in alu_operation_t;
        a, b        : in yarm_word;
        math_result : out yarm_word;

        -- compare inputs
        -- do signed comparisons
        enable_cmp : in std_logic;
        cmp_signed : in std_logic;
        cmp1, cmp2 : in yarm_word;
        cmp_result : out compare_result_t
    );
end alu;
```

alu.vhd

The ALU contains a math/logic unit as well as a comparator. It is used both explicitly by instructions such as `add` or `shiftl`, as well as to add offsets to base addresses for memory instructions and to decide whether an instructions should branch.

16 Control and Status Registers (CSR)

```vhdl
entity csr is
    generic (
        HART_ID : integer
    );
    port (
        clk    : in std_logic;
        reset  : in std_logic;
        enable : in std_logic;
        ready  : out std_logic;

        instr_info_in : in instruction_info_t;
        data_write    : in yarm_word;
        data_read     : out yarm_word;

        increase_instret : in std_logic;

        external_int : in std_logic;
        timer_int    : in std_logic;
        software_int : in std_logic;

        interrupts_pending : out yarm_word;
        interrupts_enabled : out yarm_word;
        global_int_enabled : out std_logic;
        mtvec_out : out yarm_word;
        mepc_out  : out yarm_word;

        do_trap     : in std_logic;
```

```
28          return_m_trap : in std_logic;
29          mepc_in       : in yarm_word;
30          mcause_in     : in yarm_trap_cause;
31          mtval_in      : in yarm_word;
32
33          raise_exc : out std_logic;
34          exc_data  : out exception_data_t
35      );
36  end csr;
```

csr.vhd

The control and status registers contain configurations relevant to the core itself. For example, they can be used to control interrupts.

17 Memory Arbiter

```
1   entity memory_arbiter is
2       port (
3           clk   : in std_logic;
4           reset : in std_logic;
5
6           fetch_enable    : in std_logic;
7           fetch_ready     : out std_logic;
8           fetch_address   : in yarm_word;
9           fetch_instr_out : out yarm_word;
10
11          fetch_raise_exc : out std_logic;
12          fetch_exc_data  : out exception_data_t;
13
14          datamem_enable        : in std_logic;
15          datamem_ready         : out std_logic;
16          datamem_instr_info_in : in instruction_info_t;
17          datamem_read_data     : out yarm_word;
18
19          datamem_raise_exc : out std_logic;
20          datamem_exc_data  : out exception_data_t;
21
22          -- little-endian memory interface, 4 byte address alignment
23          MEM_addr        : out yarm_word;
24          MEM_read        : out std_logic;
25          MEM_write       : out std_logic;
26          MEM_ready       : in std_logic;
27          MEM_byte_enable : out std_logic_vector(3 downto 0);
28          MEM_data_read   : in yarm_word;
29          MEM_data_write  : out yarm_word
30      );
31  end memory_arbiter;
```

memory_arbiter.vhd

Since both fetch and memory stages need to access the same system memory, access

to this common resource has to be controlled. The memory arbiter acts as a proxy for both fetch and data memory requests and stalls either until the other one completes.

18 Exception Control

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

        fetch_raise_exc : in std_logic;
        fetch_exc_data  : in exception_data_t;

        -- synchronous exceptions
        decoder_raise_exc : in std_logic;
        decoder_exc_data  : in exception_data_t;

        csr_raise_exc : in std_logic;
        csr_exc_data  : in exception_data_t;

        alignment_raise_exc : in std_logic;
        alignment_exc_data  : in exception_data_t;

        datamem_raise_exc : in std_logic;
        datamem_exc_data  : in exception_data_t;

        -- interrupts
        global_int_enabled : in std_logic;
        interrupts_enabled : in yarm_word;
        interrupts_pending : in yarm_word;

        -- stage inputs for return address + trap value (instruction)
        stage_inputs      : in pipeline_frames_t;
        interrupted_stage : in pipeline_stage_t;

        may_interrupt : in std_logic;
        do_trap       : out std_logic;
        trap_cause    : out yarm_trap_cause;
        trap_address  : out yarm_word;
        trap_value    : out yarm_word
    );
end exception_control;
```

exception_control.vhd

Several components in the core may raise a synchronous exception when an unexpected error (such as a malformed instruction or an unaligned memory access) occurs. Additionally, asynchronous interrupts (like from a timer or a UART) can be triggered externally. When an exception or an enabled interrupt is registered, program flow is diverted to the trap handler, defined using the machine trap vector (`mtvec`) CSR.

19 Erklärung der Eigenständigkeit der Arbeit

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe. Meine Arbeit darf öffentlich zugänglich gemacht werden, wenn kein Sperrvermerk vorliegt.

Ort, Datum	Armin Brauns
Ort, Datum	Daniel Plank

I Abbildungsverzeichnis

II Tabellenverzeichnis

III Listings

IV Literaturverzeichnis

Gingold, Tristan: ghdl. URL: https://github.com/ghdl/ghdl.

Bybell, Tony: GTKWave. URL: http://gtkwave.sourceforge.net.

Contributors, Various: Yosys - Yosys Open SYnthesis Suite. URL: https://github.com/YosysHQ/yosys.

Gingold, Tristan: ghdlsynth-beta. URL: https://github.com/tgingold/ghdlsynth-beta.

Shah, David: nextpnr-xilinx. URL: https://github.com/daveshah1/nextpnr-xilinx.

SymbiFlow: Project X-Ray. URL: https://github.com/SymbiFlow/prjxray.

Goavec-Merou, Gwenhael: openFPGALoader. URL: https://github.com/trabucayre/openFPGALoader.

Kjær, Olav Junker: The Nand Game. URL: http://nandgame.com.

Eater, Ben: Building an 8-bit breadboard computer! 2016. URL: https://www.youtube.com/playlist?list=PLowKtXNTBypGqImE405J2565dvjafglHU.

Clifford Wolf, Johann Glaser: “Yosys - A Free Verilog Synthesis Suite”. 2013. URL: http://www.clifford.at/yosys/files/yosys-austrochip2013.pdf.

Contributors, Various: nextpnr - a portable FPGA place and route tool. URL: https://github.com/YosysHQ/nextpnr.

Kermarrec, Florent: LiteEth. URL: https://github.com/enjoy-digital/liteeth.

Jeremy Bennett, Lee Moore: RISC-V Compliance Task Group. URL: https://github.com/riscv/riscv-compliance.