schoolRISCV + VGA

https://github.com/DigitalDesignSchool/ce2020labs/tree/master/day_4

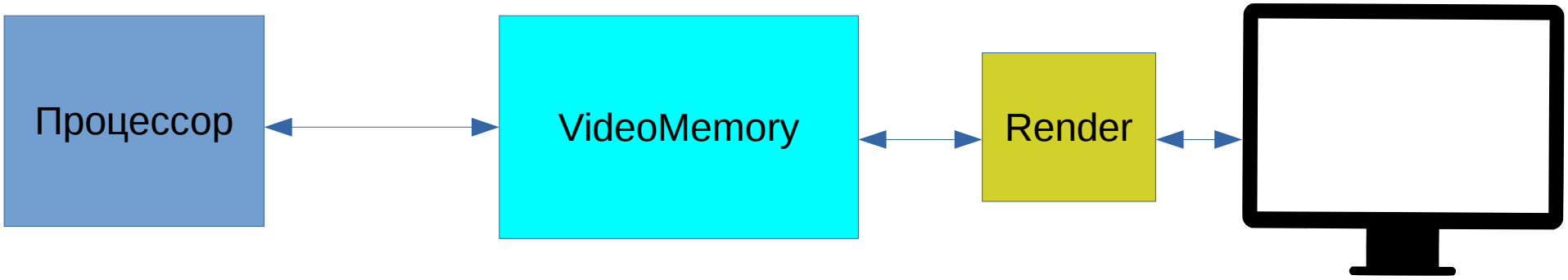
Dmitry Smekhov, 2021

Цели лабораторной работы

Подключить SchoolRISCV к VGA

- Создать интерфейс для подключение внешних устройств
- Создать компонент для управления спрайтами
- Создать программу пример
- Проверить работу на плате
- Проверить работу на модели

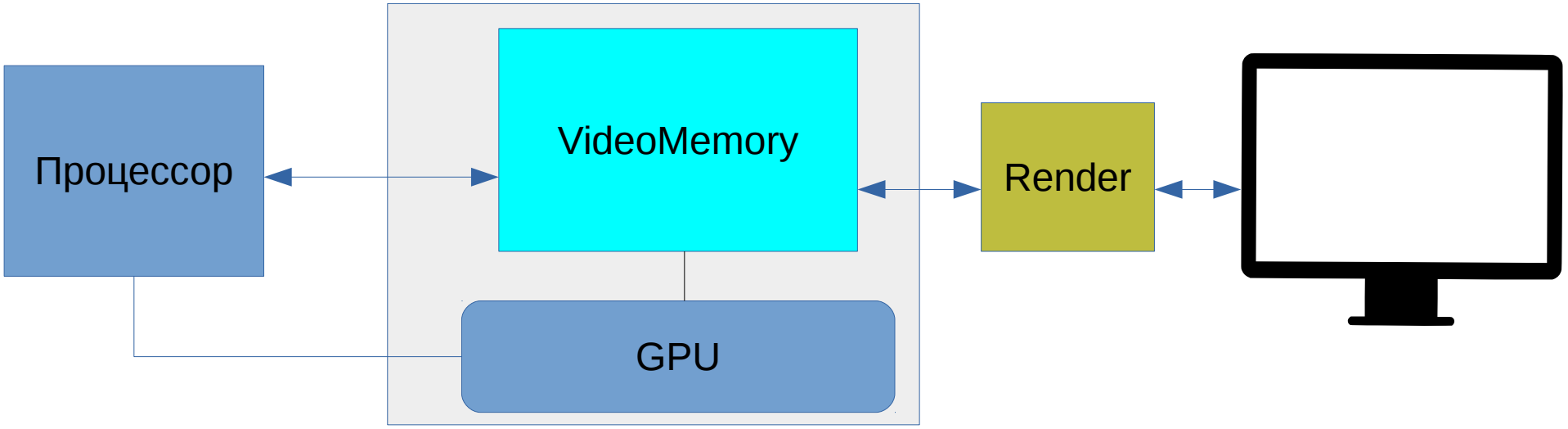
Варианты подключения - видеопамять



- VideoMemory — хранит содержимое экрана
- Render — отображает содержимое памяти на мониторе
- Процессор имеет доступ к видеопамяти

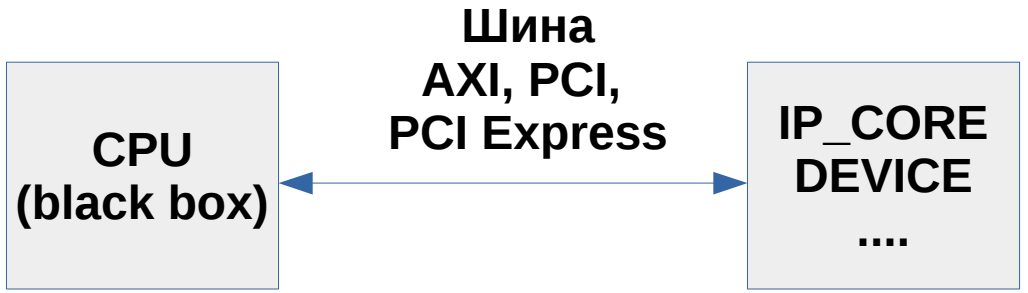
Недостаток - процессор занимается переписыванием пикселей

Варианты подключения - GPU



GPU — освобождает процессор от части операций

Вариант подключения к процессору



Адрес	Регистр
0x8000A	REG_A
0x8000B	STATUS
0x8000C	RZA5

Как правило, внешнее устройство или IP Core имеет некоторое количество регистров которое отображается на адресное пространство процессора.

Названия регистров, их адреса, назначения битов определяются фантазиями разработчика.

Успешные фантазии становятся корпоративными стандартами.

Команды обращения к памяти

Команды LOAD, STORE

LW, LH, LB — чтение слова 32, 16, 8 бит из памяти

SW, SH, SB — запись слова 32, 16, 8 бит в память

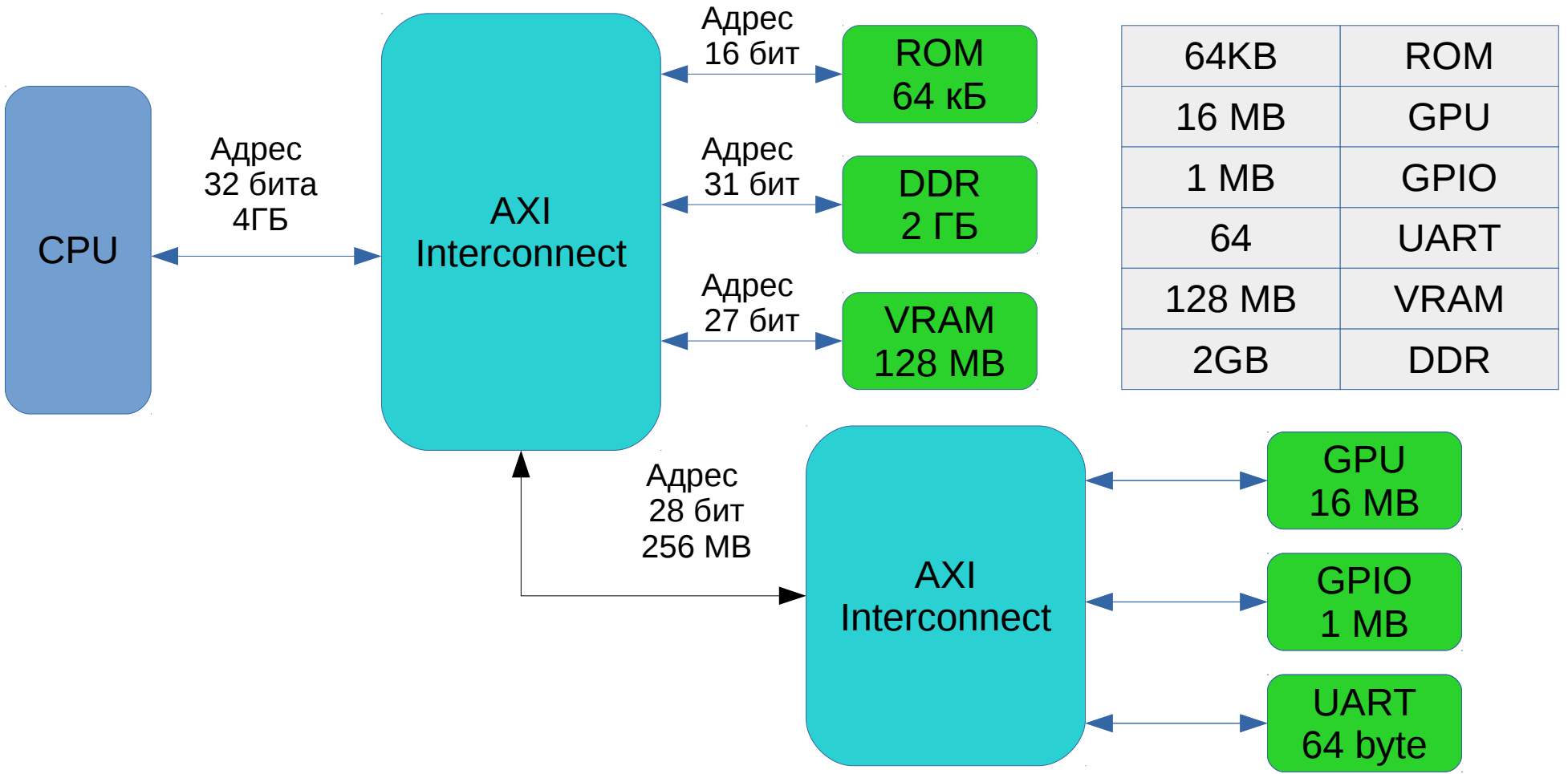
Пример:

LW A0, 100(A1)

На внешнюю шину процессора будет выставлен адрес **A1+100**, прочитанное слово будет записано в регистр **A0**

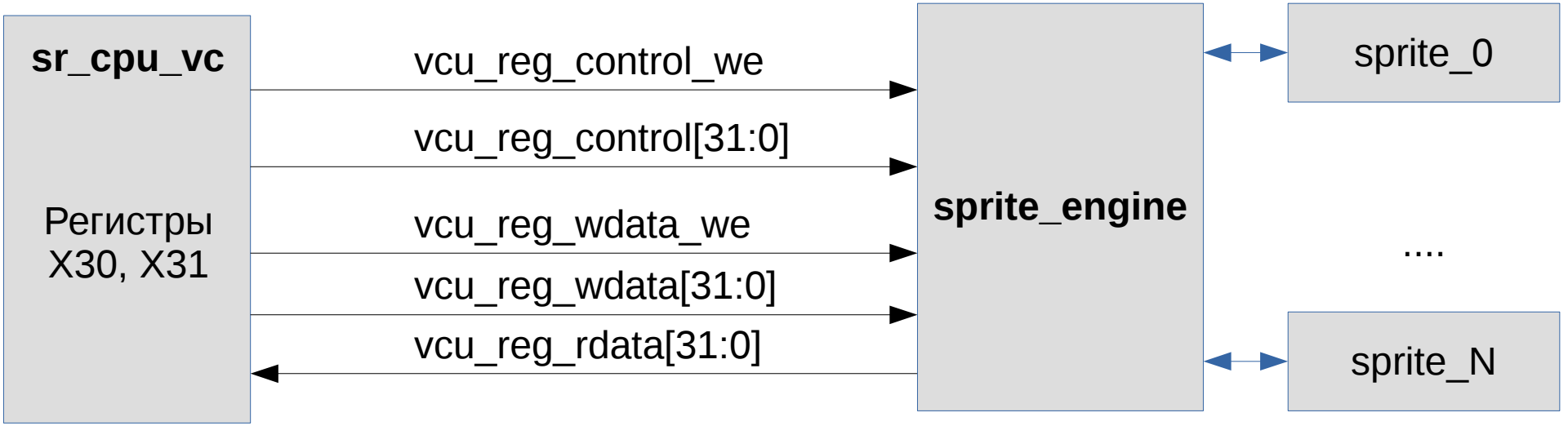
Появляется понятие адреса, а значит автоматически появляется адресное пространство

Адресное пространство — пример



64KB	ROM
16 MB	GPU
1 MB	GPIO
64	UART
128 MB	VRAM
2GB	DDR

Структура подключения SchoolRISCV



Проведено грубое вмешательство в структуру процессора.

- Запись в регистр X30 — формирование **vcu_reg_wdata**[31:0] и **vcu_reg_data_we**
- Чтение регистра X30 — чтение с шины **vcu_reg_rdata**[31:0]
- Запись в регистр X31 — формирование сигналов **vcu_reg_control**[31:0] и **vcu_reg_control_we**

Регистры X30, X31 имеют альтернативные названия T5 и T6 по соглашению о вызовах функций для компилятора

Последствия вмешательства

- Реализовано управление внешним устройством без команд обращения к памяти
  (в SchoolRISCV не реализованы команды обращения к памяти LB, LH, LW, SB, SH, SW)
- Доступно два регистра на запись и один регистр на чтение
- Доступ к регистрам реализован через регистры процессора T5 и T6
- Доступны все операции обращения к регистрам
- **Регистры T5 и T6 не могут больше использоваться как регистры общего назначения**
- Требуется провести анализ всей используемой экосистемы на возможность применения

- В нашем примере такое вмешательство допустимо.

Изменения в процессоре

Компонент sr_cpu_vc

```
// VCU register
always @(posedge clk) begin
    if( ~rst_n )
        vcu_reg_control <= 0;
    else if( regWrite & rd==5'b11111 )
        vcu_reg_control <= wd3;

    if( rd==5'b11111 )
        vcu_reg_control_we <= regWrite;
    else
        vcu_reg_control_we <= 0;

    if( ~rst_n )
        vcu_reg_wdata <= 0;
    else if( regWrite & rd==5'b11110 )
        vcu_reg_wdata <= wd3;

    if( rd==5'b11110 )
        vcu_reg_wdata_we <= regWrite;
    else
        vcu_reg_wdata_we <= 0;

end
```

Адреса регистров
t5 — адрес 30
t6 — адрес 31

```
module sm_register_file_vc
(
    input         clk,
    input  [ 4:0] a0,
    input  [ 4:0] a1,
    input  [ 4:0] a2,
    input  [ 4:0] a3,
    output [31:0] rd0,
    output [31:0] rd1,
    output [31:0] rd2,
    input  [31:0] wd3,
    input         we3,

    input   [31:0]  vcu_reg_rdata

);
    reg [31:0] rf [31:0];

    assign rd0 = (a0 != 0) ? rf [a0] : 32'b0;
    assign rd1 = (a1 != 0) ? rf [a1] : 32'b0;
    assign rd2 = (a2 != 0) ? rf [a2] : 32'b0;

    always @ (posedge clk) begin
        if( a3!=5'b11110 )
            if(we3) rf [a3] <= wd3;

        rf[5'b11110] <= vcu_reg_rdata;
    end
endmodule
```

Варианты обращения к регистрам

Требования:

- Запись константы
- Запись из регистра

Команды:

- ADDI rd, zero, const_12 - запись константы размером 12 бит
- LUI rd, const_20 - запись константы 20 бит в старшие разряды
- ADDI rd, rs, 0 - запись значения из регистра rs

Псевдокоманда

- LI const_32 - запись константы 32 бит, одна или две инструкции

Псевдокоманда LI является комбинацией команд ADDI и LUI. В зависимости от константы выполняется либо ADDI, либо LUI, либо обе.

Команды **SPRITE_ENGINE** - Вариант 1

Регистр **VCU_REG_CONTROL**

31:24 - SPRITE_NUM	23:20	19:10 - Y	9:0 - X

Недостаток 1 — требуется несколько команд для комбинирования X,Y

Недостаток 2 — требуется две команды для записи значения в регистр

Недостаток 3 — сначала записываются биты 31:12, затем 11:0

Команды **SPRITE_ENGINE** - Вариант 2

Регистр **VCU_REG_CONTROL**

11 - START	10:8 - SEL	7:0 - SPRITE_NUM

Регистры **VCU_REG_WDATA** и **VCU_REG_RDATA**

11:0 - VAL

Назначение VAL зависит от поля SEL и SPRITE_NUM
000 — X, 001 — Y
010 — DX, 011 — DY, 100 - COUNT

SPRITE_NUM - номер спрайта, значение 255 - VGA_STATUS

Адресное пространство **SPRITE_ENGINE**

Адрес (SPRITE_NUM)	Название	Описание
0	SP_TORPEDO	Торпеда
255	VGA_STATUS	Регистр развёртки

Если добавить спрайты или регистры, то они займут дополнительные строки в таблице

Адресное пространство **SPRITE**

Адрес (SEL[2:0])	Название	Описание
0	X	Координата X
1	Y	Координата Y

Предполагается добавить ещё три регистра
DX, DY - приращение координат
CNT - количество шагов

Запись значений — варианты 1 и 2

Вариант 1

```
ADDI  t6, zero,  0x00     # подготовка к записи X
ADDI  t5, zero,  22       # запись X
ADDI  t6, zero,  0x100    # подготовка к записи Y
ADDI  t5, zero,  44       # запись Y
```

Мы предполагаем что координаты будут записываться парой, поэтому можно сделать автоматическое увеличение поля **SEL**

Вариант 2

```
ADDI  t6, zero,  0x00     # подготовка к записи X,Y
ADDI  t5, zero,  22       # запись X
ADDI  t5, zero,  44       # запись Y
```

Запись значений — варианты 3 и 4

Вариант 3 — использование псевдокоманд

```
LI     t6,  0x00      # подготовка к записи X
MV   t5, a0          # запись X
MV   t5, a1          # запись Y
```

Эти команды будут преобразованы в ADDI

Вариант 4 — значение регистра с приращением

```
ADDI  t6, zero,  0x00      # подготовка к записи X,Y
ADDI  t5, a0,    10        # запись X
ADDI  t5, a1,    22        # запись Y
```

Синхронизация с развёрткой

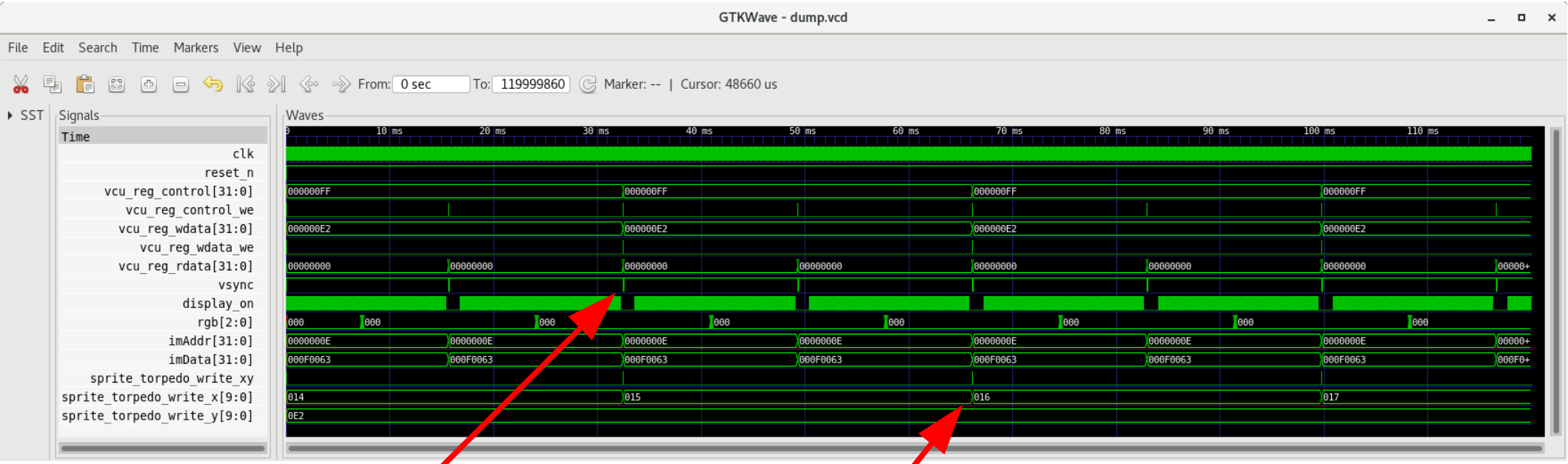
Алгоритм синхронизации:

- Запись любого значения в регистр **VCU_REG_WDATA** при установленном SPRITE_NUM=0xFF

- Чтение регистра **VCU_REG_RDATA** при установленном SPRITE_NUM=0xFF до появления 1

  Значение 1 появится только после формирования сигнала vsync, это начало нового кадра

Диаграмма работы

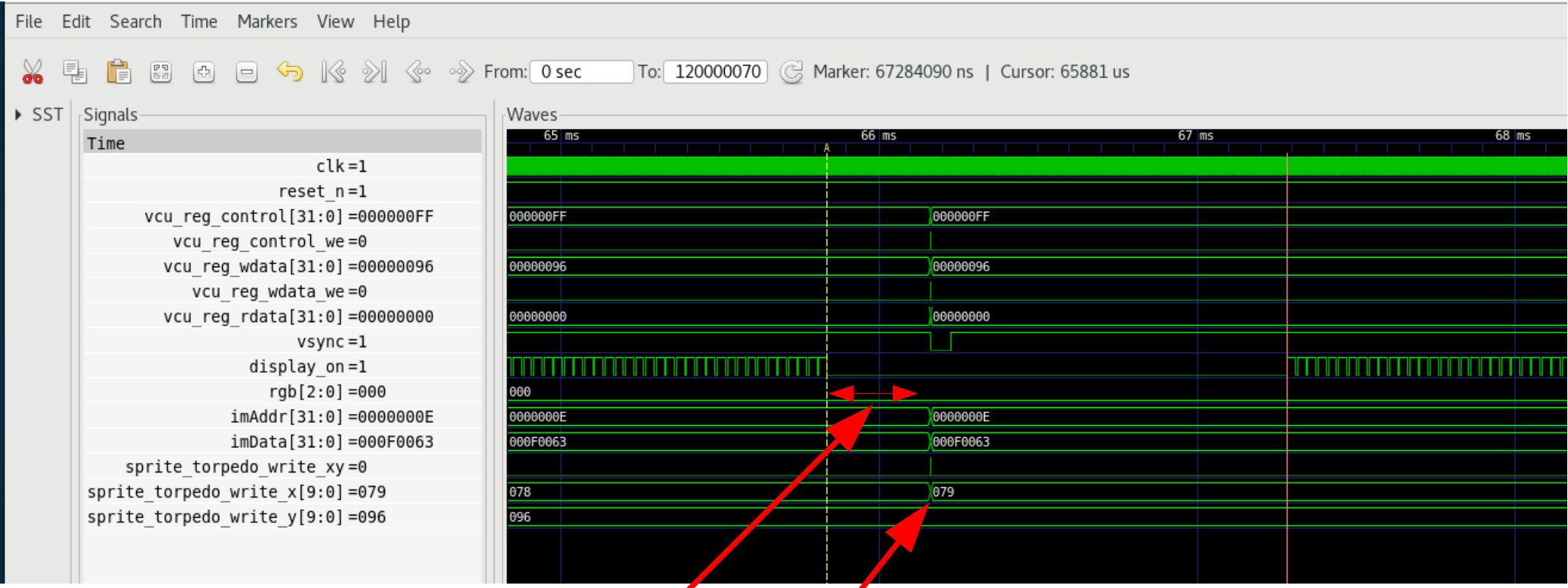


Начало отображения
нового кадра

Изменение координат

Программа изменяет координаты
каждый второй кадр

Диаграмма работы



Резерв времени

Работа программы

Между кадрами ~ 1.4 ms

Структура каталогов day_4

- **common** — общие файлы
- **doc** — полезная документация
- **program**
  - **01_sprite_move** — программа для Lab1
  - **common** — общие файлы для сборки программы
- **lab1** - лабораторная работа 1
  - **rzrd** — файлы для сборки на RZ-EazyFPGA
    - **run** — каталог сборки
  - каталоги для других плат
- **lab2** - лабораторная работа 2
  - **rzrd** — файлы для сборки на RZ-EazyFPGA
    - **run** — каталог сборки
  - каталоги для других плат
- **scripts**
  - create_run_directories.bash - создание структуры каталогов для сборки проектов

Компонент lab1/risk_game_top

- Создан на основе **game_top** из day_2/.../lab_2_game
- Закомментирован ряд компонентов
- Используется **sprite_torpedo**, управление от SchoolRISCV
- Добавлен **risk_sprite_engine** - управление спрайтами
- Добавлен **sm_rom** — память программ, размер увеличен до 128 слов
- Добавлен **sr_cpu_vc** — модифицированный SchoolRISCV

Программа 01_sprite_move

- Строки 8-14 - инициализация перемещения
- Строки 17-19 — запись координат в VCU
- Сроки 21-30 — пропуск двух кадров
- Строка 32 — вычисление нового значения
- Строка 33 — проверка достижения конечного значения
- Строка 34 — всё начинаем заново

Внимание!

Строки 25 и 26 это компенсация задержки передачи значения **SEL** для корректного чтения **VCU_REG_RDATA** со значением флага синхронизации вертикальной развёртки

```
5
6           .text
7
8   start:  li   a0, 20     # start X
9           li   a1, 600    # stop  X
10          li   a2, 1    # step  X
11
12          li   a3, 226    # start Y
13          li   a4, 400    # stop  Y
14          li   a5, 6      # step  Y
15
16
17  move_x: addi t6, zero, 0x000
18          addi t5, a0, 0
19          addi t5, a3, 0
20
21          addi a6, zero, 0
22          addi a7, zero, 2
23
24  loop_vsync: addi t6, zero, 0x0FF
25          addi t6, zero, 0x0FF
26          addi t6, zero, 0x0FF
27  wait_vsync0: beq  t5, zero, wait_vsync0
28
29          addi a6, a6, 1
30          bne  a6, a7, loop_vsync
31
32          add a0, a0, a2
33          bne a0, a1, move_x
34
35          beq zero, zero, start
36
```

Лабораторная работа № 1

Задание:

- Просмотреть код проекта
- Запустить на плате — убедиться что спрайт двигается слева направо
- Переделать программу чтобы спрайт двигался по периметру экрана
- Изменить скорость и траекторию перемещения, например ромб или треугольник
- Уменьшить число шагов перемещения, запустить симулятор, убедиться в изменении координат

Полезные ссылки:

Assemble Language https://web.eecs.utk.edu/~smarz1/courses/ece356/notes/assembly/

Reference Card: https://www.cl.cam.ac.uk/teaching/1617/ECAD+Arch/files/docs/RISCVGreenCardv8-20151013.pdf

В процессоре реализованы только следующие команды: **add, or, srl, sltu, addi, lui, beq, bne**

Доступны псевдокоманды: **li, mv**

Добавляем опрос кнопок

- **risk_game_top** — добавляем порт key_sw
- **risk_sprite_engine** — добавляем чтение состояния кнопок через vcu_reg_rdata
- Адресное пространство **SPRITE_ENGINE** - изменяется **VGA_STATUS**
- Адресное пространство **BOARD** — добавляются регистры **KEY0...KEY3**
- Программа 02_sprite_p — перемещение спрайта при нажатии кнопок

Адресное пространство **SPRITE_ENGINE**

Адрес (SPRITE_NUM)	Название	Описание
0	SP_TORPEDO	Торпеда
255	BOARD	Управление платой

VGA_STATUS изменился на BOARD

Возможно добавление регистров управления светодиодами и семисегментными индикаторами

Адресное пространство **BOARD**

Адрес (SEL[2:0])	Название	Описание
0	VSYNC	Развёртка
1	KEY0	Кнопка 0
2	KEY1	Кнопка 1
3	KEY2	Кнопка 2
4	KEY3	Кнопка 3

Значение 1 в регистрах KEY0...KEY3 — кнопка нажата

Опрос кнопок

Недостаток -
очень большой код для опроса кнопок

Возможен другой вариант:

- один регистр для всех кнопок
- один раз прочитать значение
- четыре сравнения с шаблоном

Ещё вариант - внутри SPRITE_ENGINE сформировать новые значения для A2, A3

```
line0_wait_vsync:
                beq  t5, zero, line0_wait_vsync

                li a2, 0
                li a3, 0

check_key0:     addi t6, zero, 0x1FF
                addi t6, zero, 0x1FF    # delay for update new value of SPRITE_NUM
                addi t6, zero, 0x1FF

                beq t5, zero, check_key1
                li a2, 1

check_key1:     addi t6, zero, 0x2FF
                addi t6, zero, 0x2FF    # delay for update new value of SPRITE_NUM
                addi t6, zero, 0x2FF

                beq t5, zero, check_key2
                li a2, -1

check_key2:     addi t6, zero, 0x3FF
                addi t6, zero, 0x3FF    # delay for update new value of SPRITE_NUM
                addi t6, zero, 0x3FF

                beq t5, zero, check_key3
                li a3, 1

check_key3:     addi t6, zero, 0x4FF
                addi t6, zero, 0x4FF    # delay for update new value of SPRITE_NUM
                addi t6, zero, 0x4FF

                beq t5, zero, change_xy
                li a3, -1

change_xy:      add a0, a0, a2  # change X
                add a1, a1, a3  # change Y
```

Лабораторная работа № 2

Задание:

- Просмотреть код проекта
- Запустить на плате — убедиться что спрайт перемещается при нажатии на кнопки

Домашнее задание:

- Добавить управление вторым спрайтом
- Проект выложить на GitHub