Министерство науки и высшего образования Российской Федерации
Мытищинский филиал
Федерального государственного автономного образовательного учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ __КОСМИЧЕСКИЙ_____

КАФЕДРА ___К-3_____

отчет

К ЛАБОРАТОРНОЙ РАБОТЕ

по ДИСЦИПЛИНЕ

«ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА ЭВМ»

Студент __К3-76Б__ _____Сериков Д.Е._____
(Группа) (И.О.Фамилия)
Студент __К3-76Б__ _____Чернов В.Д._____
(Группа) (И.О.Фамилия)

Преподаватель _____Н.В.Ефремов_____
(И.О.Фамилия)

2025 г.

Описание лабораторной работы

Тема:

Изучение манипулятора типа мышь. Подключение его к процессорной системе DE 2-115 Media Computer с использованием интерфейса PS/2

Цель:

Изучить интерфейс и контроллер PS/2, мыши и её принцип работы.

Теоретический материал.

Порты PS/2 и буфер FIFO:

Компьютер DE2-70 имеет два порта PS/2, которые могут подключаться к стандартным клавиатурам или мышам с интерфейсом PS/2. Каждый порт включает буфер First-In-First-Out (FIFO) объемом 256 байт для хранения данных, полученных от подключенного устройства PS/2.

Регистр PS2_Data:

Программный интерфейс порта PS/2 включает два регистра (Рисунок 1), одним из которых является регистр PS2_Data. Этот регистр можно читать и записывать. Когда бит RVALID (бит 15) равен 1, чтение из этого регистра извлекает данные из начала буфера FIFO (поле Data), а также количество записей в буфере FIFO (поле RAVAIL). Чтение из регистра PS2_Data уменьшает поле RAVAIL на 1. *Запись в PS2_Data можно использовать для отправки команды устройству PS/2*.

Рисунок 1. Регистры порта PS/2 (Computer System)

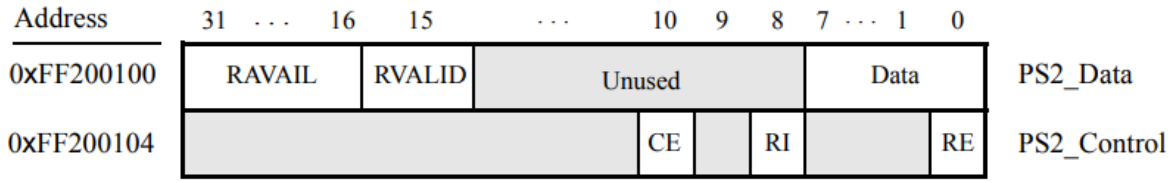
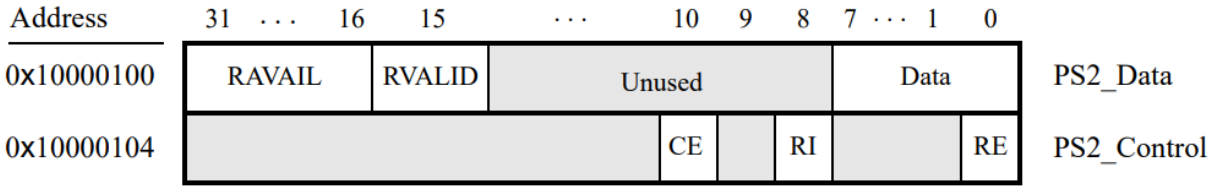


Рисунок 2. Регистры порта PS/2 (Media Computer System)



Регистр PS2_Control:

Другим регистром является регистр PS2_Control. Его можно использовать для разрешения прерываний от порта PS/2, установив поле RE (Receive Enable) в значение 1. Когда RAVAIL (количество записей в буфере FIFO) больше 0, генерируется прерывание. Когда происходит прерывание, бит RI (Receive Interrupt) устанавливается в 1. Его можно сбросить, опустошив буфер FIFO. Поле CE (Command Error) в регистре PS2_Control указывает, произошла ли ошибка при отправке команды устройству PS/2.

Используемые файлы программ:

Файл PS2_X_Y_Packets_Rate_to_JTAG_and_LedR

содержит программу, которая пересылает содержимое буфера FIFO контроллера PS/2 в ОП процессорной системы. Вначале, программа разрешает прерывание от таймера и задает для него временной интервал 3 сек. При получении первого пакета начинается отсчет времени. Программа сохраняет содержимое буфера FIFO в ОП, подсчитывает количество пакетов, принятых от мыши, определяет перемещение по осям X и У. После завершения заданного временного интервала обработчик прерывания определяет частоту дискретизации мыши, поделив общее число принятых пакетов на длительность интервала. Результирующие значения перемещения по обеим осям, число пакетов и частота дискретизации выводятся в терминальное окно приложения AMP (IFPGAMP). Частота дискретизации также выводится на красные светодиоды.

Ход работы

Часть 1. Режимы работы мыши и команды управления манипулятором.

Команда Reset (0xFF)

Используя вкладку Memory, мы отправили команду Reset (0xFF) в порт данных мыши. Мы использовали кнопку Refresh, чтобы просмотреть ответное сообщение мыши. Мы получили код 0xFA, что соответствует успешному выполнению команды, то есть выполнена перезагрузка клавиатуры. На рис. 3 приведен результат.

Рис. 3 Ответное сообщение при подаче команды Reset (0xFF)

Команда Status Request (0xE9)

Ответ со статусом мыши

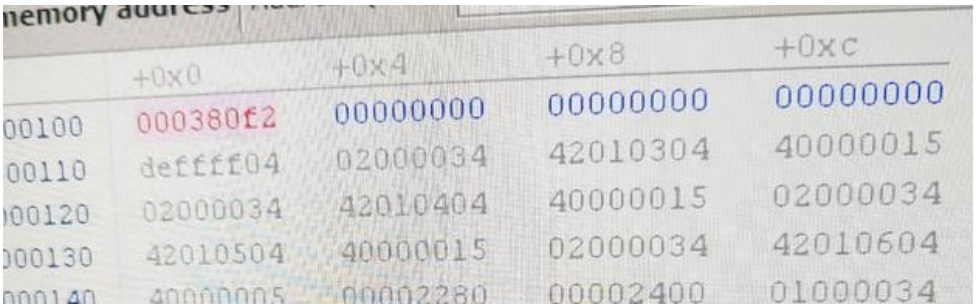


Рис. 4 Ответное сообщение при подаче команды Status Request (0xE9)

Команда Enable Data Reporting (0xF4)

Данная команда запрашивает разрешает отправку отчетов работы мыши

Экспериментально было определено, как работают эти команды. После отправки команды 0xF4 в порт PS/2, при дальнейших нажатиях мышка реагирует и отправляет данные.

Команда Set Stream Mode (0xEA)

Переключает мышь в потоковый режим работы

Экспериментально было определено, что манипулятор не присылает данные в этом режиме до отправки команды на разрешение чтения

Команда Read Data (0xEB)

Разрешает чтение потока данных мышки

Реакция на перемещение и нажатие кнопок

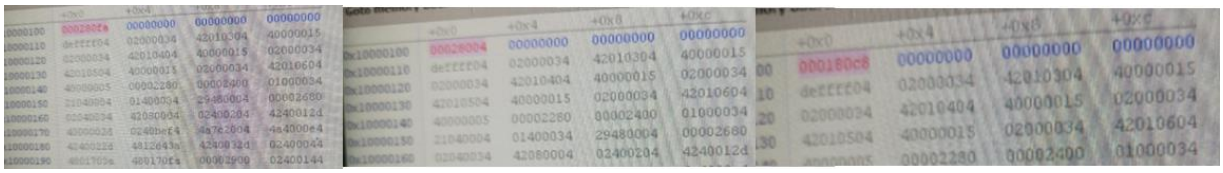


Рис. 5 Данные с информацией о перемещении мышки и нажатии клавиш

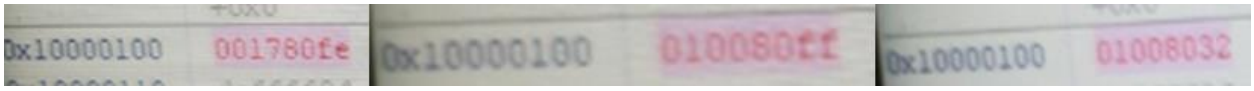


Рис. 6 Данные с информацией о перемещении мышки и нажатии клавиш

Экспериментально было определено, что в удалённом режиме только по команде eb отправляет пакет из 3 байтов

Еще раз запрашиваем статус мыши:

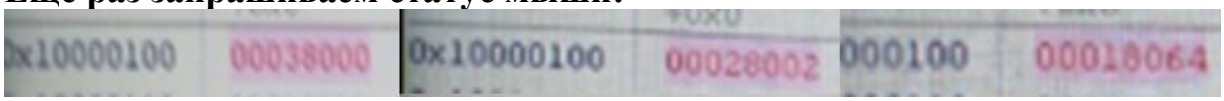


Рис. 8 результат запроса статуса мышки

Экспериментально было определено что последний байт в отчете о состоянии это частота дискретизации

Часть 2. Исследование потокового режима работы мыши и формата пересылаемых пакетов

Инициализировали мышь для ее использования с колесом Прокрутки последовательно изменяя ее частоту опроса.

выполнив вращение колесика на меньшую величину.
Диапазон для перемещения колесика от 255 до -256
Установив минимальную частоту дискретизации, равную десяти отчетам в секунду. Использовав команду
Set Sample Rate (0xF3) в порте данных мыши.

Равномерно перемещая мышь влево в течении трех секунд установили, что начальная позиция мыши на момент обновления равна 0, движение на себя будет интерпретировано как отрицательное смещение по оси Y, а движение влево как отрицательное смещение по оси X, соответственно движениями в противоположные стороны будут достигнуты положительные смещения

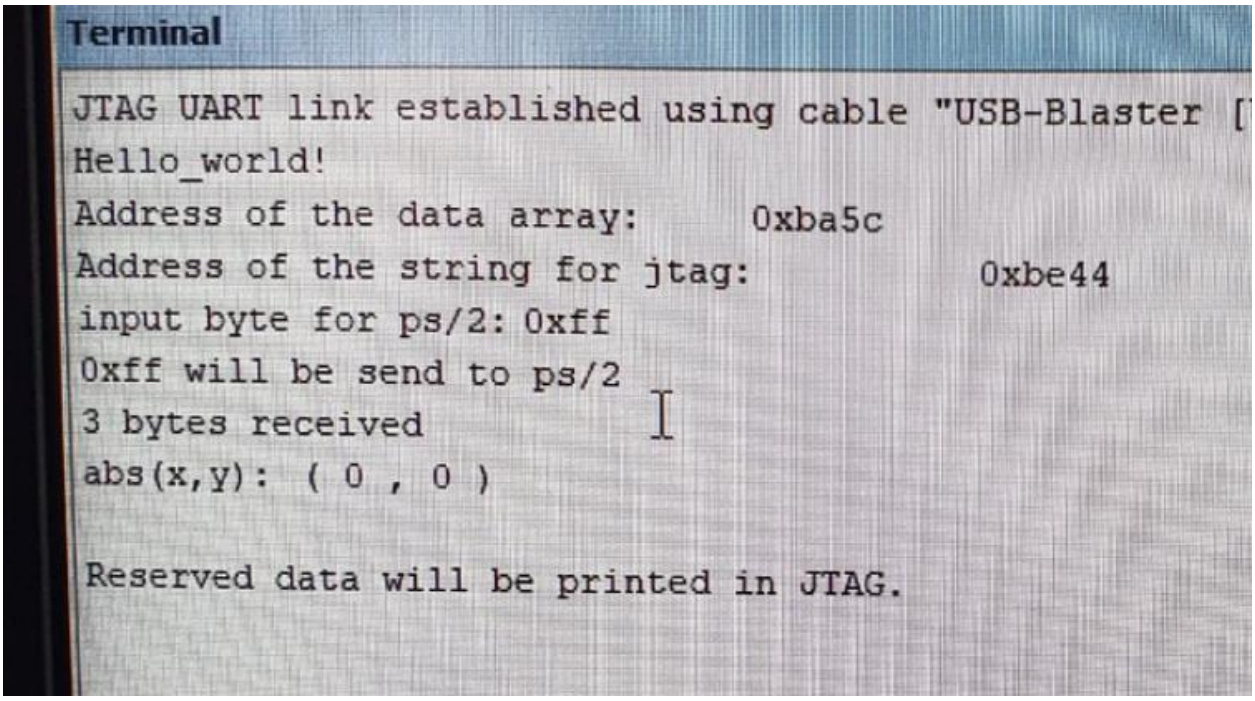


Рис. 7 вывод программы с координатами перемещения мышки

Часть 3. Определение разрешающей способности мыши и её временных характеристик

Установили минимальное разрешение, соответствующее одному значению на миллиметр, отправив в порт данных мыши команду
Set Resolution (0xE8), наблюдайте ответный байт от мыши, после чего передайте байт с аргументом для этой команды (0x00).

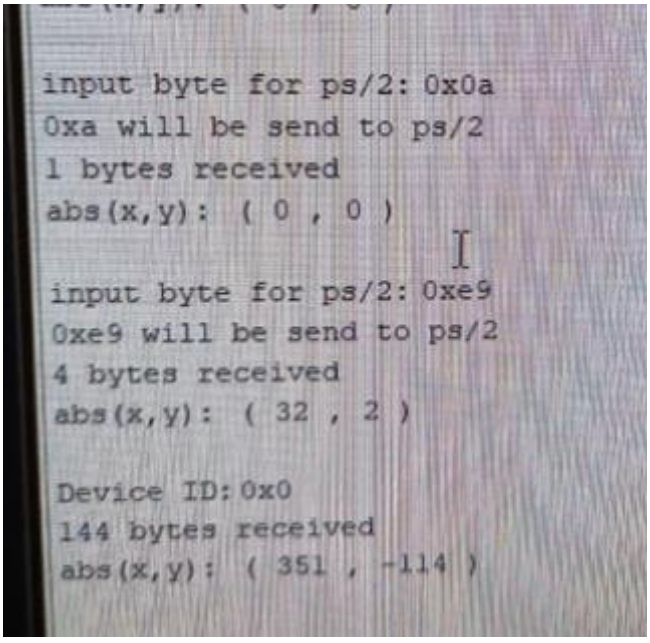


Рис. 8 вывод программы с координатами перемещения мышки

Экспериментально было установлено, что коэффициент домножения 2.8 в формуле для пересчета в длину. Не корректен и подбирается для каждой мышки отдельно.

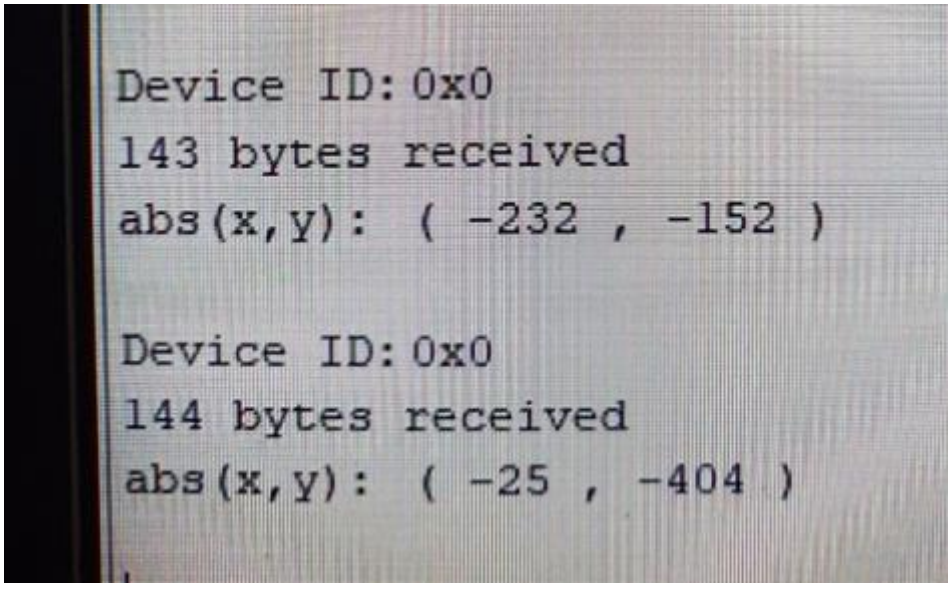


Рис. 9 Результаты при непрерывном движении мыши на протяжении 3 секунд.

Установив максимальное разрешение, соответствующее восьми значениям на миллиметр, отправив в порт данных мыши команду Set Resolution (0xE8), наблюдайте ответный байт от мыши, после чего передайте

байт с аргументом для этой команды (0x03).

Было установлено что это значение так же уникально для каждой мышки.

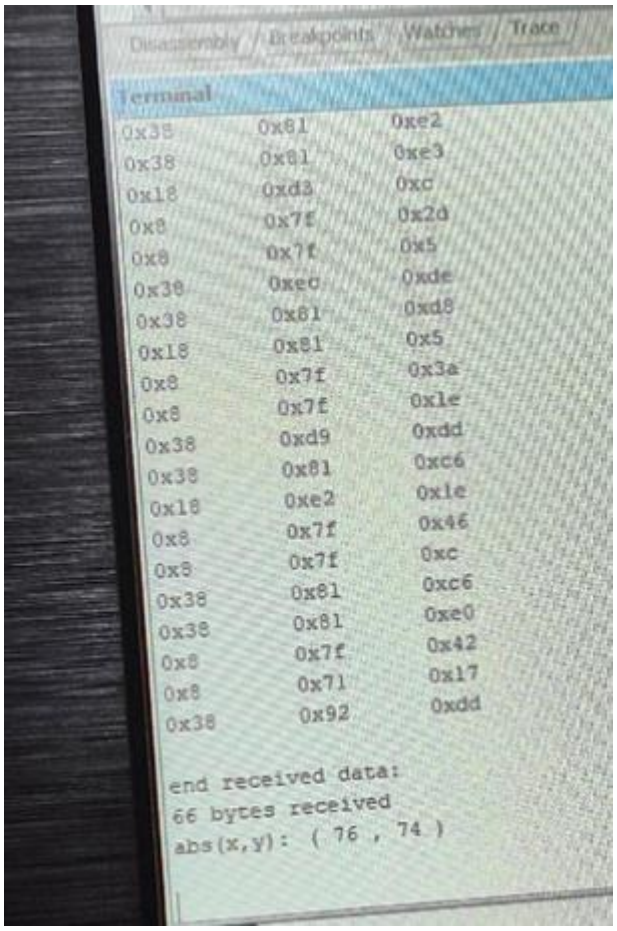


Рис. 10 Вывод в терминал при непрерывном движении мыши.

Часть 4. Исследование параметра масштабирования мыши

При попытке изменения масштабирования мышки мы столкнулись с тем что некоторые мышки включая нашу экспериментальную не изменяют масштаб после получения команды.

Проделав все предыдущие пункты, мы получили те же результаты.

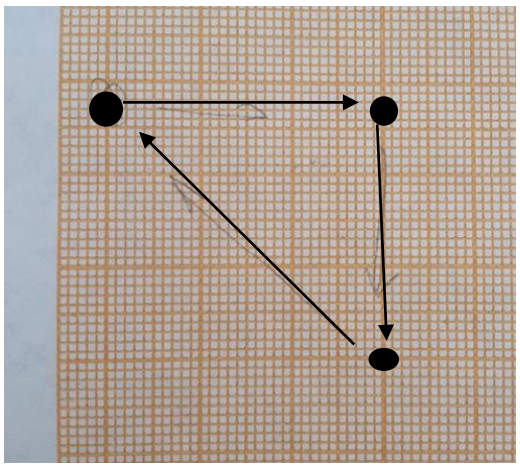


Рис. 11 Траектория движения мыши на миллиметровке (начало в верхней левой точке)

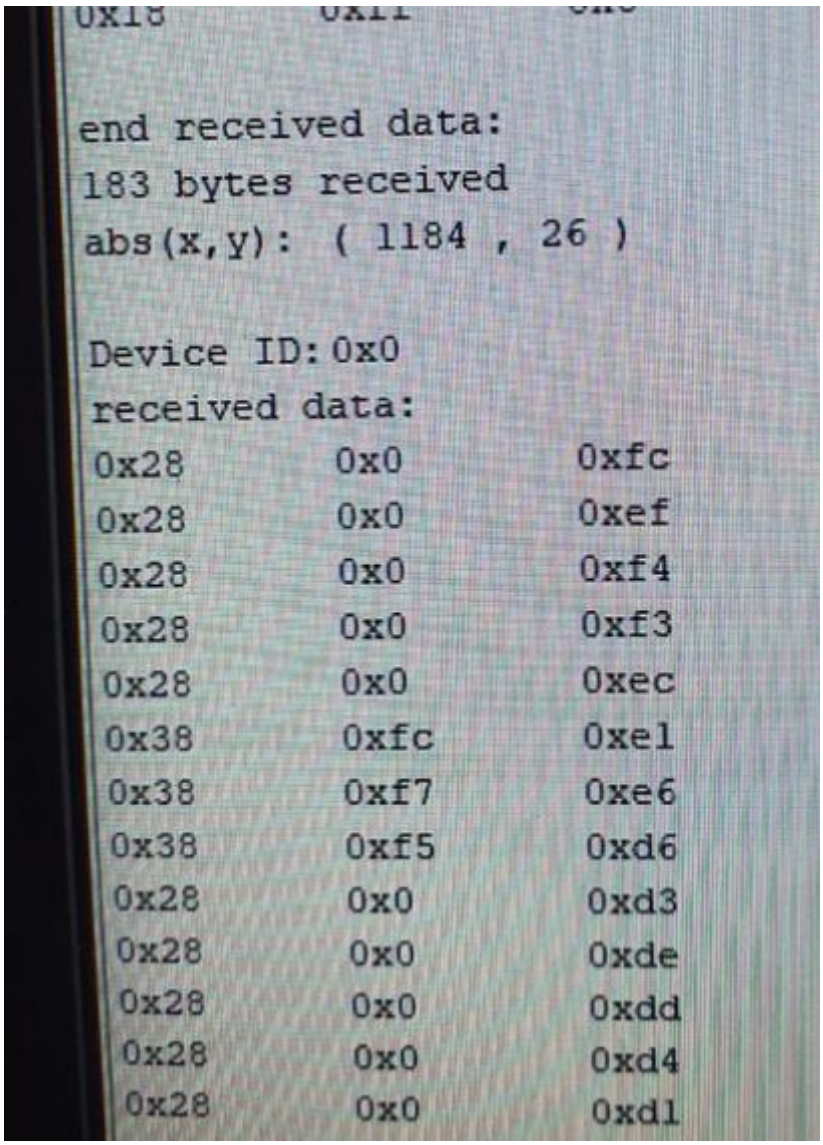


Рис. 12 Движение от точки 1 (левой верхней) до точки 2 (верхней правой)

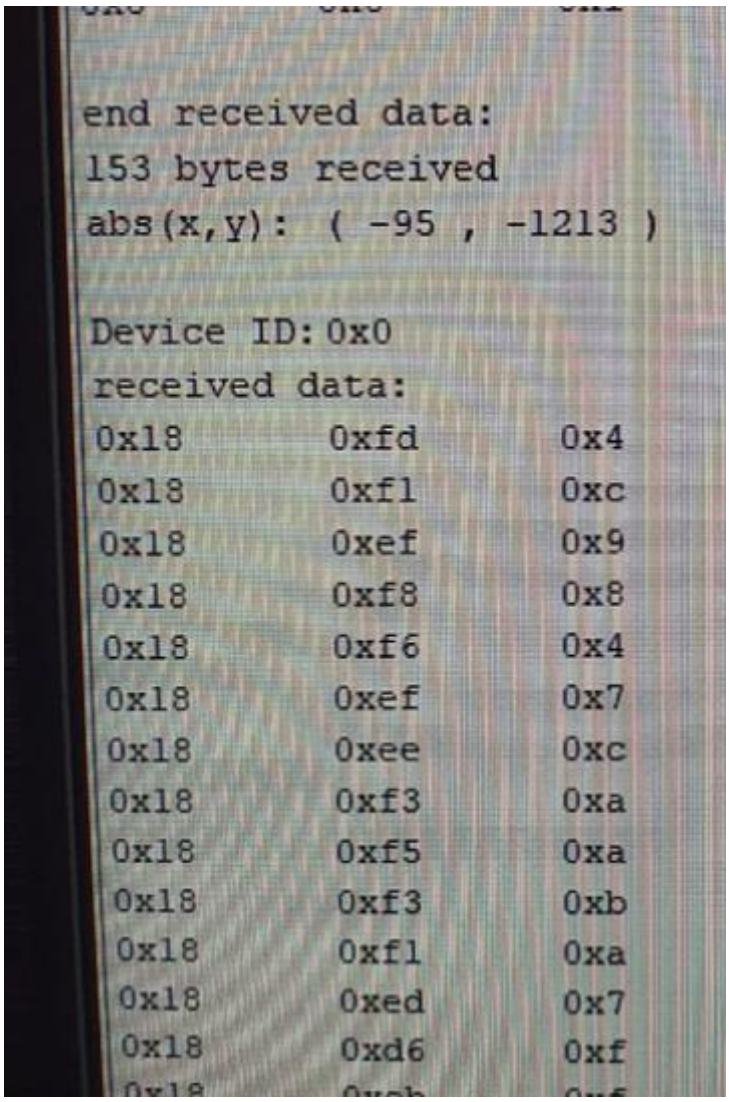


Рис. 13 Движение от точки 2 (верхней правой) до точки 3 (нижней)

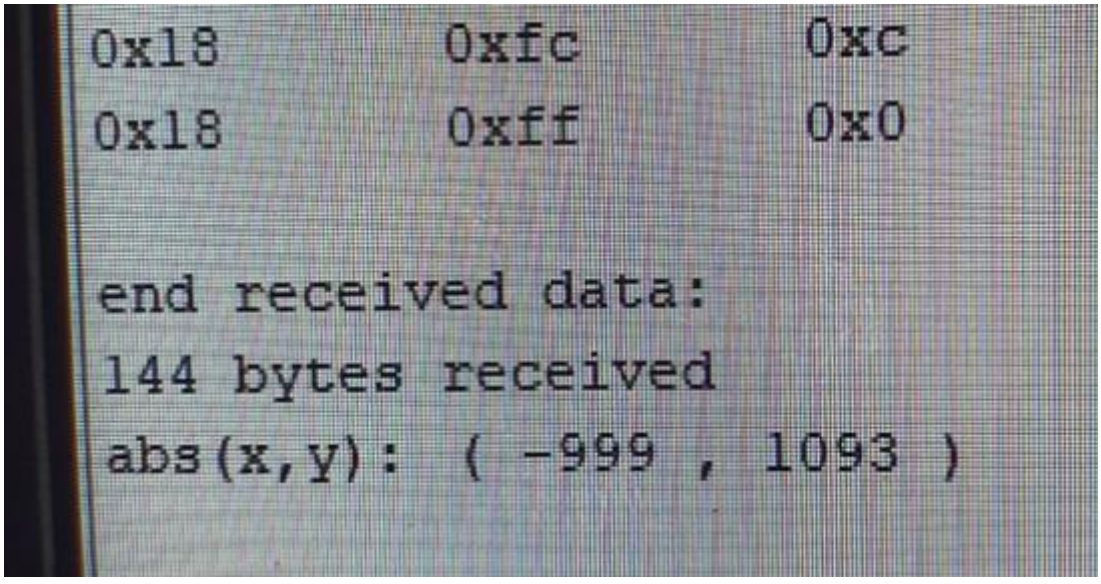


Рис. 14 Движение от точки 3 (нижней) до исходной точки 1 (верхней левой)

Выводы

В процессе работы мы изучили порт PS/2, подключили через него мышку, а также изучили команды для работы с мышкой, ее характеристики и их настройки.

Приложение

Листинг файла PS2_X_Y_Packets_Rate_to_JTAG_and_LedR.s

```
.equ LED_R, 0xFF200000
.equ PS2_DATA, 0xFF200100
.equ TIMER, 0xFF202000
.equ JTAG_UART_BASE, 0xFF201000
.equ PACKETS_START, 0x1000
.equ PACKETS_END, 0x2000
# Время работы таймера в секундах.
.equ TIMER_DURATION, 3

.text
.org 0x20
handler:
    # Сохранение регистров.
    stw ra, (sp)
    stw r4, -4(sp)
    stw r8, -8(sp)
    stw r9, -12(sp)
    subi sp, sp, 16
    # Выход, если прерывание не является внешним.
    rdctl r8, ctl4
    beq r8, zero, handler_end
    # Декремента ea на 1 команду.
    subi ea, ea, 4
    # Выход, если прерывание не от таймера.
    andi r8, r8, 0b1
    beq r8, zero, handler_end
    # Выход, если программа завершилась.
    movia r8, PROG_ENDED
    ldb r8, (r8)
    bne r8, zero, handler_end
    # Печать абсолютного смещения по X.
    movia r4, ABS_X_STR
    call LOG_STR
    movia r4, ABS_X
    ldw r4, (r4)
    call LOG_NUM
    call LOG_LINE
    # Печать абсолютного смещения по Y.
    movia r4, ABS_Y_STR
    call LOG_STR
    movia r4, ABS_Y
    ldw r4, (r4)
    call LOG_NUM
    call LOG_LINE
    # Печать общего количества считанных пакетов.
    movia r4, TOTAL_PACKETS_STR
    call LOG_STR
    movia r4, TOTAL_PACKETS
    ldw r4, (r4)
    call LOG_NUM
    call LOG_LINE
    # Печать частоты дискретизаци.
    movi r8, TIMER_DURATION
    div r8, r4, r8
    movia r4, SAMPLING_RATE_STR
    call LOG_STR
    mov r4, r8
    call LOG_NUM
    call LOG_LINE
    # Вывод частоты дискретизации на красные
светодиоды.
    movia r8, LED_R
    stwio r4, (r8)
    # Установка флага завершения программы.
    movia r8, PROG_ENDED
    movi r9, 1
    stb r9, (r8)
handler_end:
    # Восстановление регистров.
    addi sp, sp, 16
    ldw ra, (sp)
    ldw r4, -4(sp)
    ldw r8, -8(sp)
    ldw r9, -12(sp)
    eret

.global _start
_start:
    # Начальный адрес стека.
    movia sp, 0x3FFFFFC
    # Инициализация переменных.
    movia r8, ABS_X
    stw zero, (r8)
    movia r8, ABS_Y
    stw zero, (r8)
    movia r8, TOTAL_PACKETS
    stw zero, (r8)
    movia r8, PROG_ENDED
    stb zero, (r8)
    # Очистка красных светодиодов.
    call CLEAR_LED_R
    # Очистка буфера.
    call CLEAR_PS2_BUFF
    # Очистка ОП для записи пакетов.
    movia r4, PACKETS_START
    movia r5, PACKETS_END
    call CLEAR_MEM
    # Адрес начала регистров таймера.
    movia r8, TIMER
    # Остановка таймера.
    movi r9, 0b1000
    sthio r9, 0x4(r8)
    # Сброс поля TO таймера.
    sthio zero, (r8)
    # Установка длительности работы таймера = 50МГц
* TIMER_DURATION.
    movia r9, 50000000
    muli r9, r9, TIMER_DURATION
    # Запись младшего полуслова длительности.
    sthio r9, 0x8(r8)
    # Запись старшего полуслова длительности.
    srli r9, r9, 16
    sthio r9, 0xC(r8)
    # Разрешение внешних прерываний.
    movi r9, 0b1
    wrctl status, r9
    # Разрешение прерываний от таймера.
    wrctl ienable, r9
    # Чтение первого пакета мыши в r2.
    call READ_PS2_PACKET
    # Запуск таймера STOP = 0, START = 1, CONT = 0, ITO =
1.
    movi r9, 0b0101
    sthio r9, 0x4(r8)
    # Начальный адрес пакетов в ОП.
    movia r8, PACKETS_START
    # Общее количество считанных пакетов.
    mov r9, zero
    # Абсолютное смещение по X.
    mov r10, zero
    # Абсолютное смещение по Y.
    mov r11, zero
while:
    # Адрес записи пакета в ОП.
    muli r12, r9, 4
    add r12, r8, r12
```

```
            # Запись пакета в ОП.
            stw r2, (r12)
            # Выделение 1 байта с флагами.
            andi r12, r2, 0xFF
            # Выделение 2 байта с относительным смещением
по X.
            srli r2, r2, 8
            andi r13, r2, 0xFF
            # Относительное смещение по X со знаком.
            slli r14, r12, 4
            andi r14, r14, 0x100
            sub r13, r13, r14
            # Увеличение абсолютного смещения по X.
            add r10, r10, r13
            movia r13, ABS_X
            stw r10, (r13)
            # Выделение 3 байта с относительным смещением
по Y.
            srli r2, r2, 8
            andi r13, r2, 0xFF
            # Относительное смещение по Y со знаком.
            slli r14, r12, 3
            andi r14, r14, 0x100
            sub r13, r13, r14
            # Увеличение абсолютного смещения по Y.
            add r11, r11, r13
            movia r13, ABS_Y
            stw r11, (r13)
            # Увеличение количества считанных пакетов.
            addi r9, r9, 1
            movia r13, TOTAL_PACKETS
            stw r9, (r13)
            # Чтение следующего пакета.
            call READ_PS2_PACKET
        br while



    /* Очищает красные светодиоды. */
    CLEAR_LED_R:
        # Сохранение регистра.
        stw r8, (sp)
        movia r8, LED_R
        stwio zero, (r8)
        # Восстановление регистра.
        ldw r8, (sp)
        ret

    /* Очищает буфер FIFO. */
    CLEAR_PS2_BUFF:
        # Сохранение регистров.
        stw ra, (sp)
        stw r2, -4(sp)
        subi sp, sp, 8
    CLEAR_PS2_BUFF_LOOP:
        # Чтение регистра PS2_Data в r2.
        call READ_PS2_DATA
        # Выделение 15 бита RVALID.
        andi r2, r2, 0x8000
        # Если RVALID = 1, то продолжаем чтение.
        bne r2, zero, CLEAR_PS2_BUFF_LOOP
        # Восстановление регистров.
        addi sp, sp, 8
        ldw ra, (sp)
        ldw r2, -4(sp)
        ret

    /* Очистка области памяти с адреса r4 по r5 */
    CLEAR_MEM:
        # Сохранение регистра.
        stw r4, (sp)
    CLEAR_MEM_LOOP:
        # Очистка памяти.
        stw zero, (r4)
        addi r4, r4, 4
        # Если очистили не всю область памяти, то
продолжаем.
        blt r4, r5, CLEAR_MEM_LOOP
        # Восстановление регистра.
        ldw r4, (sp)
        ret

    /* Записывает значение регистра PS2_Data в r2. */
    READ_PS2_DATA:
        movia r2, PS2_DATA
        ldwio r2, (r2)
        ret

    /* Возвращает байты пакета мыши PS2 в r2. */
    READ_PS2_PACKET:
        # Сохранение регистров.
        stw ra, (sp)
        stw r8, -4(sp)
        stw r9, -8(sp)
        stw r10, -12(sp)
        subi sp, sp, 16
        # Номер байта в пакете.
        mov r8, zero
        # Результат (байты пакета).
        mov r9, zero
    READ_PS2_PACKET_LOOP:
        # Чтение регистра PS2_Data в r2.
        call READ_PS2_DATA
        # Выделение 15 бита RVALID.
        srli r10, r2, 15
        andi r10, r10, 1
        # Если RVALID = 0, то ожидаем следующий байт.
        beq r10, zero, READ_PS2_PACKET_LOOP
        # Выделение поля DATA.
        andi r2, r2, 0xFF
        # Сдвиг значения влево на (номер байта * 8 бит).
        muli r10, r8, 8
        sll r2, r2, r10
        # Добавление байта к результату.
        or r9, r9, r2
        # Инкремент номера байта.
        addi r8, r8, 1
        # Если номер байта < 3, то продолжаем чтение.
        cmplti r10, r8, 3
        bne r10, zero, READ_PS2_PACKET_LOOP
        # Запись результата в r2.
        mov r2, r9
        # Восстановление регистров.
        addi sp, sp, 16
        ldw ra, (sp)
        ldw r8, -4(sp)
        ldw r9, -8(sp)
        ldw r10, -12(sp)
        ret

    /* Отправляет символ из r4 в JTAG UART. */
    LOG_CHAR:
        # Сохранение регистров.
        stw r8, (sp)
        stw r9, -4(sp)
        stw r10, -8(sp)
        # Загрузка базового адреса JTAG UART в r8.
        movia r8, JTAG_UART_BASE
        # Чтение регистра управления.
        ldwio r9, 4(r8)
        # Выделение поля WSPACE (доступное место в
буфере записи).
        srli r9, r9, 16
        # Запись символа в поле DATA регистра данных.
        stbio r4, (r8)
        # Восстановление регистров.
        ldw r8, (sp)
        ldw r9, -4(sp)
```

```
        ldw r10, -8(sp)
        ret

    /* Отправляет число из r4 в JTAG UART. */
    LOG_NUM:
        # Сохранение регистров.
        stw ra, (sp)
        stw r4, -4(sp)
        stw r8, -8(sp)
        stw r9, -12(sp)
        stw r10, -16(sp)
        stw r11, -20(sp)
        subi sp, sp, 24
        # Адрес текущего символа в r8.
        mov r8, r4
        # Число 10 в r9.
        movi r9, 10
        # Изначальный адрес стека в r11.
        mov r11, sp
        # Проверка является ли число положительным.
        bge r8, zero, LOG_NUM_POSITIVE
        # Если число отрицательное, печатаем минус.
        movi r4, 0x2D
        call LOG_CHAR
        # И делаем число положительным.
        muli r8, r8, -1
    LOG_NUM_POSITIVE:
        # Получаем остаток от деления на 10 в r4.
        div r10, r8, r9
        mul r4, r10, r9
        sub r4, r8, r4
        # Добавляем цифру в стек.
        stw r4, (sp)
        subi sp, sp, 4
        # Цикл пока результат деления != 0.
        mov r8, r10
        bne r8, zero, LOG_NUM_POSITIVE
    LOG_NUM_DIGITS:
        # Печать цифр.
        addi sp, sp, 4
        ldw r4, (sp)
        addi r4, r4, 0x30
        call LOG_CHAR
        bne sp, r11, LOG_NUM_DIGITS
        # Восстановление регистров.
        addi sp, sp, 24
        ldw ra, (sp)
        ldw r4, -4(sp)
        ldw r8, -8(sp)
        ldw r9, -12(sp)
        ldw r10, -16(sp)
        ldw r11, -20(sp)
        ret

    /* Отправляет строку по адресу r4 в JTAG UART. */
    LOG_STR:
        # Сохранение регистров.
        stw ra, (sp)
        stw r4, -4(sp)
        stw r8, -8(sp)
        subi sp, sp, 12
        # Адрес текущего символа в r8.
        mov r8, r4
    LOG_STR_LOOP:
        # Печать символов.
        ldb r4, (r8)
        # Если по адресу текущего символа пусто, то
выходим.
        beq r4, zero, LOG_STR_END
        call LOG_CHAR
        addi r8, r8, 1
        br LOG_STR_LOOP
    LOG_STR_END:
        # Восстановление регистров.
        addi sp, sp, 12
        ldw ra, (sp)
        ldw r4, -4(sp)
        ldw r8, -8(sp)
        ret

    /* Отправляет символ перехода на новую строку в JTAG
UART. */
    LOG_LINE:
        # Сохранение регистров.
        stw ra, (sp)
        stw r4, -4(sp)
        subi sp, sp, 8
        # Печать пробела.
        movi r4, 0xA
        call LOG_CHAR
        # Восстановление регистров.
        addi sp, sp, 8
        ldw ra, (sp)
        ldw r4, -4(sp)
        ret

    .data
    ABS_X:
        .word 0
    ABS_Y:
        .word 0
    TOTAL_PACKETS:
        .word 0
    PROG_ENDED:
        .byte 0
    ABS_X_STR:
        .asciz "x: "
    ABS_Y_STR:
        .asciz "y: "
    TOTAL_PACKETS_STR:
        .asciz "packets: "
    SAMPLING_RATE_STR:
        .asciz "sampling rate: "

    .end
```