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

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

2025 г.

Описание лабораторной работы

Тема:

Изучение клавиатуры и подключение ее к процессорной системе DE 2-70 Media Computer с использованием интерфейса PS/2.

Цель:

Изучить интерфейс и контроллер PS/2 клавиатуры и её принцип работы.

Теоретический материал.

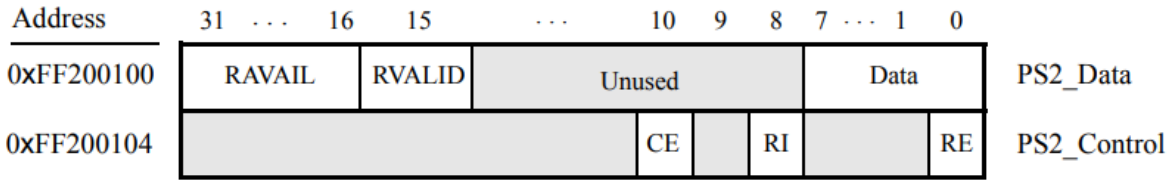
Порты PS/2 и буфер FIFO:

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.

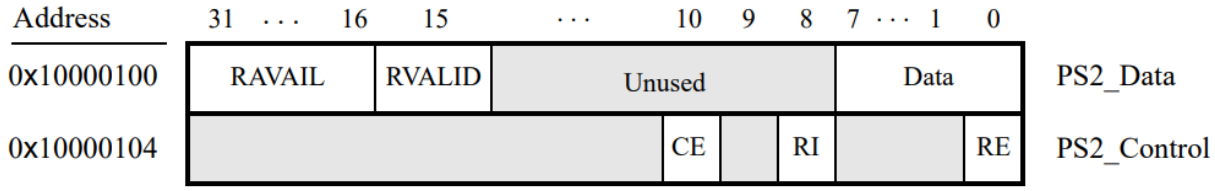


Рисунок 1. Регистры порта PS/2 (Computer System)

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:

PS2_from_FIFO_to_HEX

Содержит программу, предназначенную для чтения байтов данных из буфера FIFO контроллера PS/2 с выводом их на HEX индикаторы. Программа читает содержимое регистра данных порта PS/2, анализирует бит RValid прочитанного слова, и если он установлен, то отображает младший байт прочитанного слова на HEX индикаторах. Причем в начале своей работы программа гасит все индикаторы. Первый переданный устройством байт отображается на индикаторах HEX1, HEX0, второй- HEX3, HEX2, и так далее. Если индикаторов меньше, чем байт в буфере, выводимое значение будет смещаться вправо на 1 байт, а освободившиеся два последних HEX индикатора выведут следующий байт из буфера.

PS2_From_FIFO_to_OP_and_LCD

Содержит программу, которая пересылает содержимое буфера FIFO контроллера PS/2 в ОП процессорной системы и выводит его на LCD экран в верхней строке, а в нижней указывает количество выведенных на экран байт. Если все байты не помещаются, выведенное содержимое смещается влево, а справа выводится следующий байт.

PS2_Typematic_rate_Delay_to_JTAG

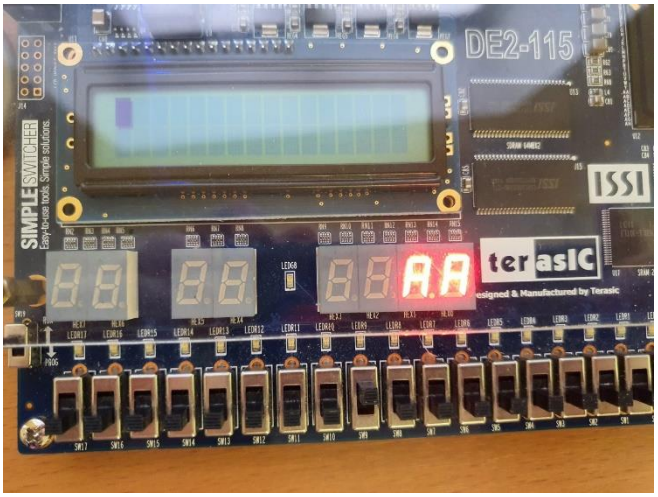
Содержит программу, предназначенную для выполнения пятой части работы. В ней вначале программируется интервальный таймер на непрерывную работу и разрешаются прерывания от клавиатуры. Прерывания будут происходить каждый раз, когда в буфере FIFO контроллера PS/2 будет появляться очередной байт, переданный клавиатурой. Обработчик прерывания считывает текущее значение интервального таймера (тайм-код), фиксируя тем самым момент поступления байта от клавиатуры, переданный клавиатурой байт данных и пересылает их в ОП в два массива. Тайм-коды, пересылаются в первый массив, начиная с адреса SDRAMTIME (по умолчанию 0x1000) пословно, а сами принятые от клавиатуры байты данных в другой, со смещением DELTA (по умолчанию 0x1000). После приема 20 байт (эта величина задана в виде константы NUM_OF_BYTES) работа обработчика завершается, а в терминал JTAG UART выводится частота повтора и задержку автоповтора символов.

Ход работы

Часть 1. Исследование команд управления клавиатурой.

Команда Reset (0xFF)

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



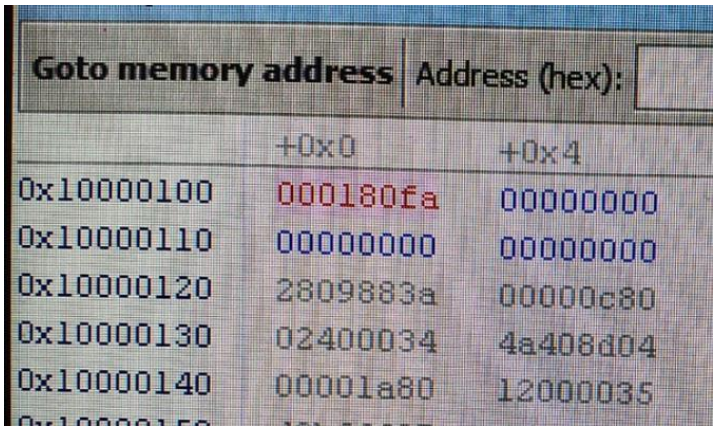


Рис. 3-4 Ответное сообщение при подаче команды Reset (0xFF)

Команда Resend (0xFE)

Далее мы нажимали клавиши клавиатуры и наблюдали ответ по адресу порта PS/2. Вот что мы получили:

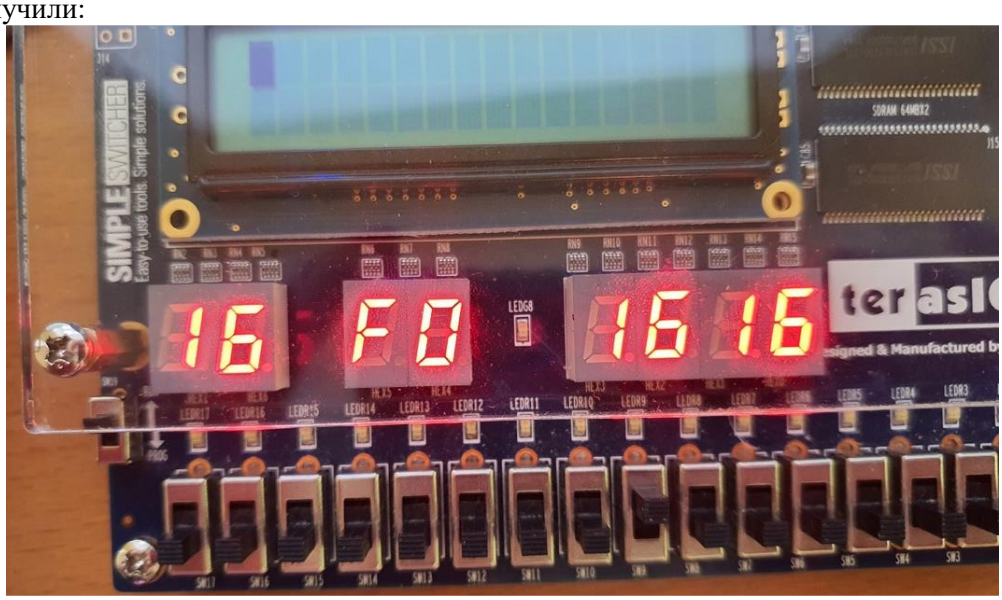


Рис. 5 Байты, считываемые в порт при нажатии определенных клавиш

Команда ReadID (0xF2)

Данная команда запрашивает 2 байта, которые являются идентификационным номером клавиатуры. Номер нашей клавиатуры: 0xAB 0x83 (171 131 в десятичной)

Команда Disable/Enable scanning (0xF4/0xF5)

Экспериментально было определено, как работают эти команды. После отправки команды 0xF5 в порт PS/2, при дальнейших нажатиях клавиш на клавиатуре буфер перестает принимать скан-коды нажатия клавиш. А после отправки команды 0xF4 в порт PS/2 прием скан-кодов нажатий и отпусканий клавиш возобновляется.

Таким образом 0xF5 запрещает прием скан-кодов клавиш, а 0xF4 разрешает прием скан-кодов клавиш.

При отправке данных при запрете сканирования клавиш, буфер продолжает получать данные, так-как бит RVALID устанавливается в 1 (рис. 4).

Команда Echo (0xEE)

Эта команда обратной связи, используемая для проверки правильности связи и функционирования между компьютером и подключенным устройством через интерфейс PS/2. Клавиатура отвечает тем же "Echo" (0xEE). Мы в этом экспериментально убедились.

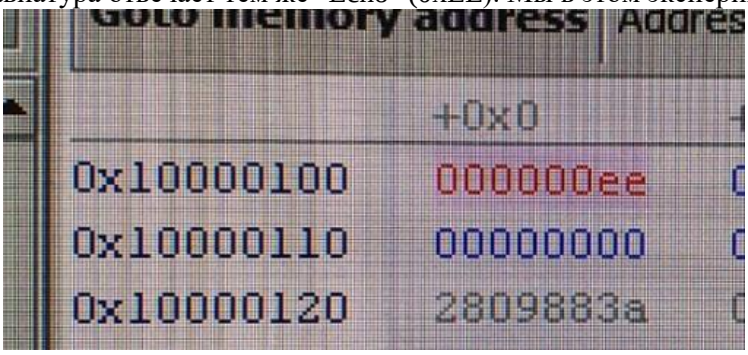
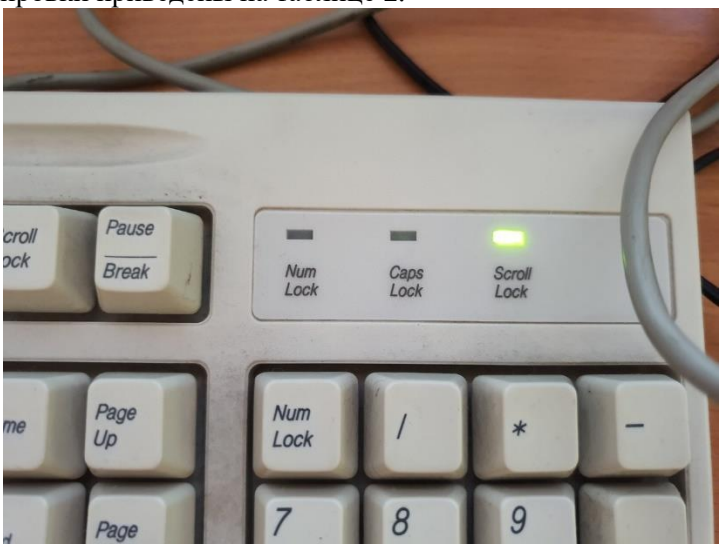


Рис. 6. ‘Echo’

Команда Set/Reset LEDs (0xED)

Эта команда зажигает на светодиоды Num Lock, Caps Lock, Scroll Lock на клавиатуре. Экспериментально были определены значения аргумента для разных светодиодов. Полученные кодировки приведены на таблице 2.



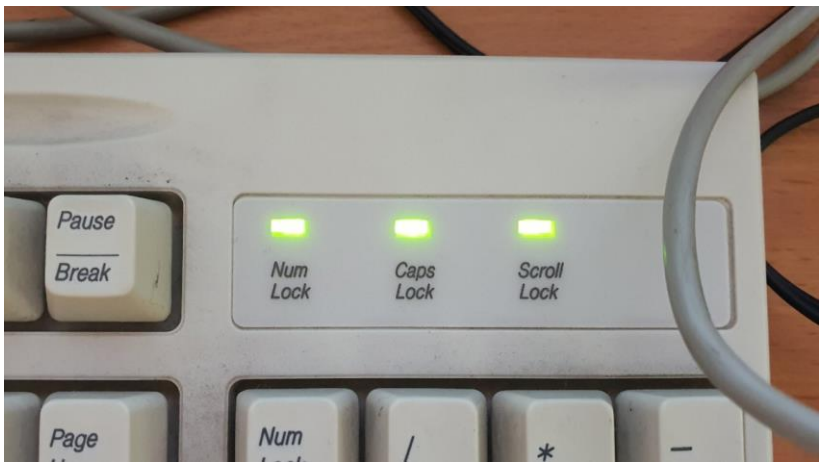


Рис. 6-7 Управление светодиодами на клавиатуре

Таблица 2. Кодировка различных комбинаций светодиодов на клавиатуре

Назначение\бит	2	1	0
Scroll Lock	0	0	1
Num Lock	0	1	0
Num Lock + Scroll Lock	0	1	1
Caps Lock	1	0	0
Caps Lock + Scroll Lock	1	0	1
Num Lock + Caps Lock	1	1	0
Все светодиоды	1	1	1

Команда Set Default (0xF6)

Устанавливает стандартные значения параметров клавиатуры (какие именно, выясняется в следующих частях).

Часть 3. Исследование скан-кодов нажатия и отпускания клавиш

В этой части мы использовали файл PS2_from_FIFO_to_HEX.s, чтобы выяснить, у каких клавиш какие скан-коды нажатия. Для этого мы запустили мы запустили программу, нажимали различные клавиши на клавиатуре и наблюдали скан-коды нажатия и отпускания клавиш на HEX-индикаторах (пример на рис. 2).

Команда Set Scan Code (0xF0)

С помощью этой команды мы выставляли различные режимы клавиатуры и проверяли скан-коды клавиш в этих режимах. Сначала в порт посылается сама команда 0xF0, затем байт-аргумент, который соответствует соответствующему режиму: 0x01 – первый, 0x02 – второй, 0x03 – третий. При этом, если подать аргумент 0x00, то в ответе получим два байта: байт подтверждения и байт, указывающий на номер текущего режима (рис. 5-8). Так, при установке настроек клавиатуры по умолчанию (команда 0xF6 из пред. пункта) и отправке в порт команды

0xF0 с аргументом 0, были получены следующие байты

Таким образом, команда была успешно выполнена, а клавиатура установлена во 2 режим. После ввода команды 0xF0 пришло подтверждение команды (fa), а затем 18001 – установленный режим.

Режим 1	18001
Режим 2	18002
Режим 3	18003

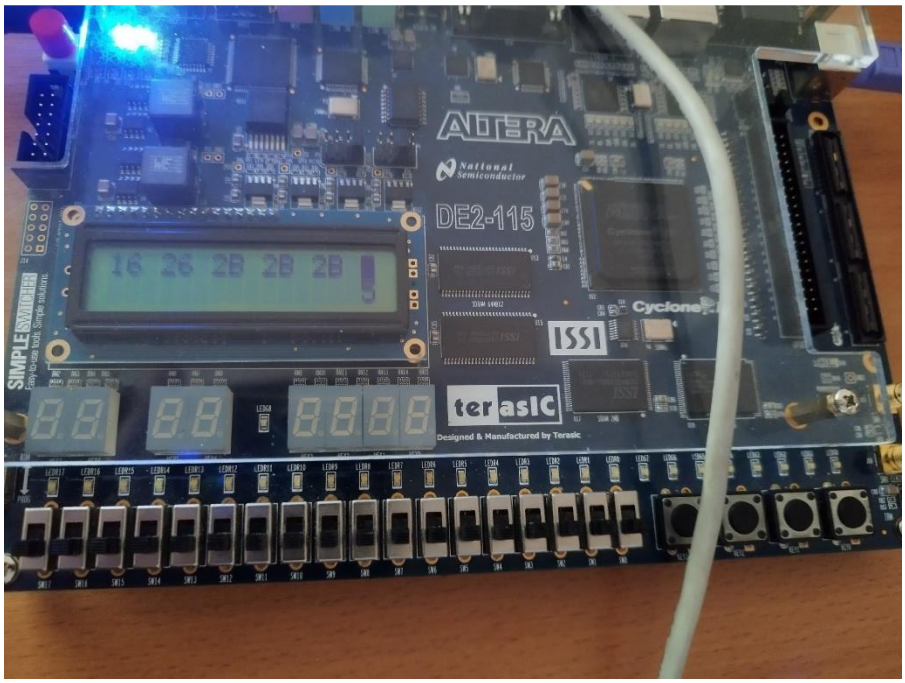


Рис. 10 Перезапуск программы командой continue

Таблица 3. Скан коды клавиш клавиатуры

Клавиша	Set #1		Set #2		Set #3	
	Press	Let	Press	Let	Press	Let
1	82	02	16	F0 16	16	
2	83	03	1e	F0 1e	1e	
3	84	04	26	F0 26	26	
4	85	05	25	F0 25	25	
5	86	06	2e	F0 2e	2e	
6	87	07	36	F0 36	36	
7	88	08	3d	F0 3d	3D	
8	89	09	3e	F0 3e	3E	
9	8a	0a	46	F0 46	46	
0	8b	0b	45	F0 45	45	
Enter	9c	1c	5a	F0 5a	5A	
Backspace	8e	0e	66	F0 66	66	
Left Alt	68	38	11	F011	19	F0 19
Right Alt	68 e0	38 e0	11 f0	E0 11	39	
Left Ctrl	9d	1d	14	F0 14	11	F0 11
Right Ctrl	9d e0	1d e0	14 f0	E0 14	58	
NL 1	CF	4F	69	F0 69	69	
NL 2	D0	50	72	F0 72	72	
NL 3	D1	51	7A	F0 7A	7A	
NL 4	CB	4B	6B	F0 6B	6B	
NL 5	CC	4C	73	F0 73	73	
NL 6	CD	4D	74	F0 74	74	
NL 7	C7	47	6C	F0 6C	6C	
NL 8	C8	48	75	F0 75	75	
NL 9	C9	49	7D	F0 7D	7D	
NL 0	D2	52	70	F0 70	70	

Далее программа была перезапущена с помощью команд AMP: “Restart” и “Continue”, а потом остановлена. Далее мы вернули начальные настройки клавиатуры отправкой команды 0xFF в порт PS/2. (рисунок 10)

Часть 4. Определение размера буфера FIFO в контроллере PS/2

Для выполнения данной части мы использовали файл PS2_From_FIFO_to_OP_and_LCD.s. Мы скомпилировали и запустили данную программу. И сразу же ее остановили. Программа очистила область памяти в диапазоне 0x1000 по 0x2000

Далее мы зажимали клавишу ‘1’ в течение 15 секунд. Затем мы проверили, сколько байтов записалось в буфер FIFO (рис. 12). Тоже самое мы проделали для клавиш ‘2’ и ‘f’ (рис. 13 и 14).

Состояние регистра PS2_Data после зажатия клавиш:

1	00b28016
2	00b18016
f	00b0802b

Таким образом, RAVAIL = 0xb0, то есть 178 байт в буфере при нажатии клавиш. Небольшие отклонения в значении RAVAIL при нажатиях различных клавиш может быть объяснено погрешностью при измерениях. Далее мы проделывали то же самое, но удерживали различные клавиши в течение 18 секунд. Получали в поле RAVAIL значение 0x100, соответствующе 256 байтам (рис. 14)

Определили размер буфера FIFO по полю RAVAIL: 0100805a

Заполнение буфера нажатием клавиш

Далее мы запустили программу. Программа считывала данные из буфера и заполняла ими очищенную область памяти. Мы наблюдали количество оставшихся байт в буфере по зеленым светодиодам. Далее мы наблюдали, что область памяти, начиная с адреса 0x1000, заполнилась 255 байтами. Причем каждый байт соответствует скан-коду нажатия клавиши Enter во втором режиме, которую мы и зажимали последней. Причем скан-кодов отпускания клавиши в памяти нет. Делаем вывод, что при зажатии клавиши буфер заполняется скан-кодами нажатия.

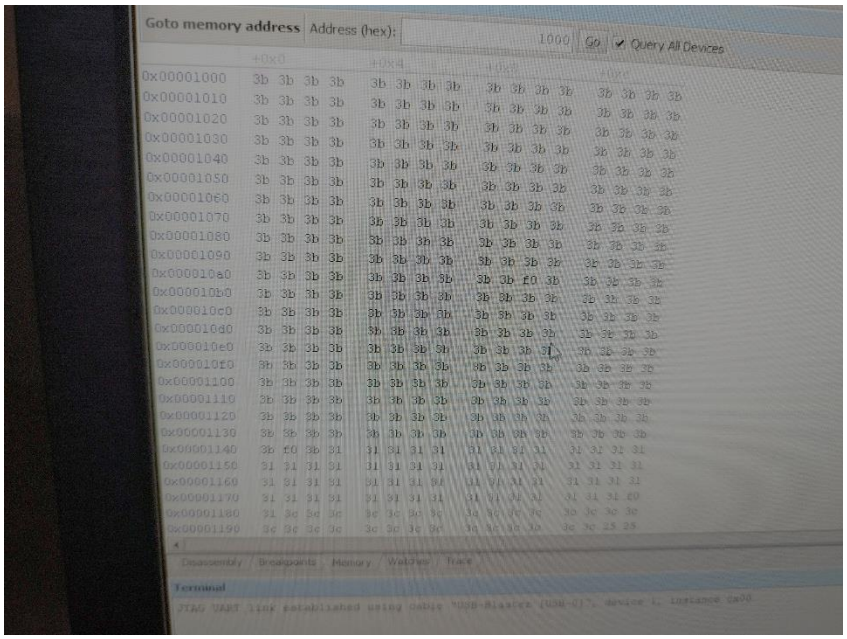


Рис. 11 Выделенный участок памяти заполнился скан-кодами нажатия клавиш

Часть 5. Определение условия формирования прерывания от клавиатуры. Определение временных параметров автоповтора клавиатуры

В данной части мы использовали файл PS2_Typematic_rate_Delay_to_JTAG. Далее мы убедились, что бит прерывания RE в регистре управления порта PS/2 установлен (рис. 16).

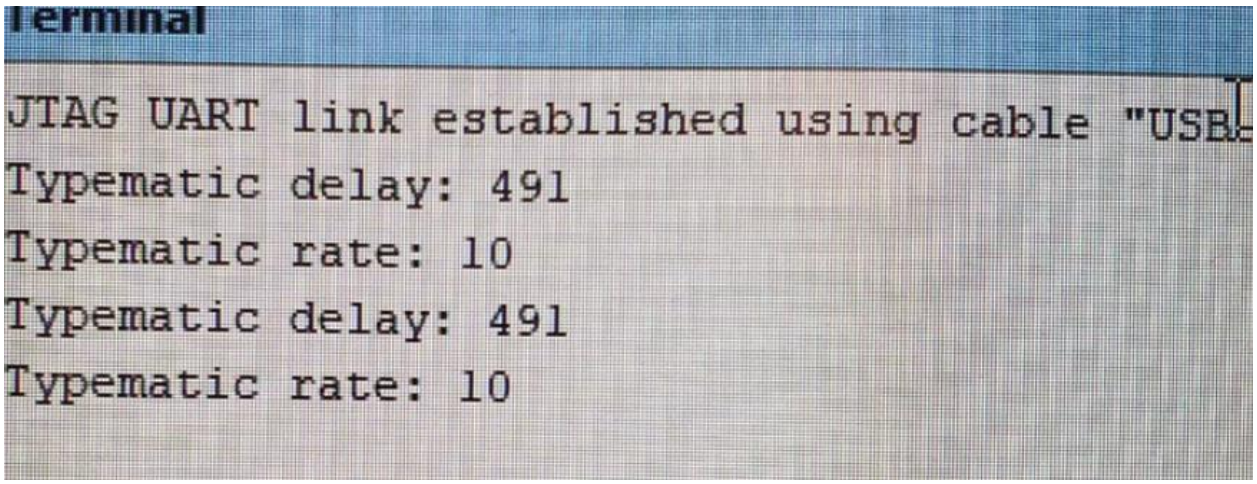


Рис. 12 Два эксперимента с зажатием клавиши Enter. Один и тот же результат

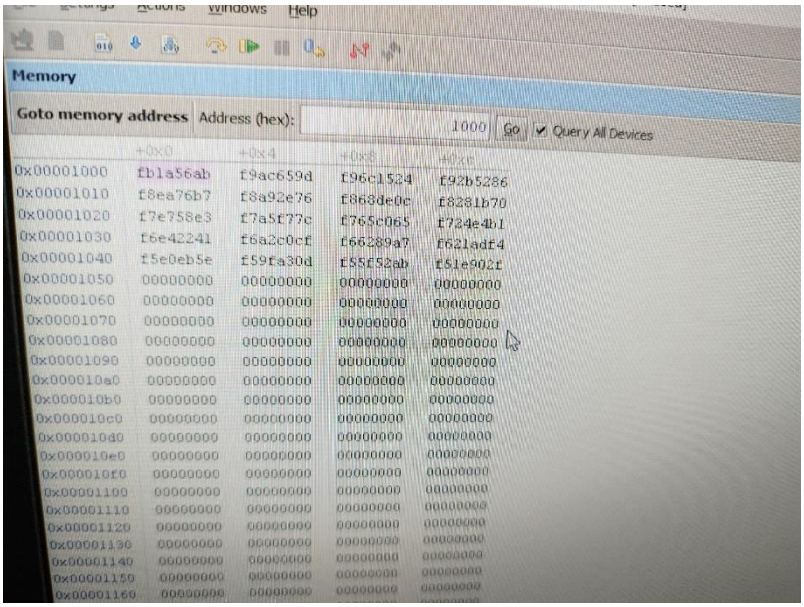
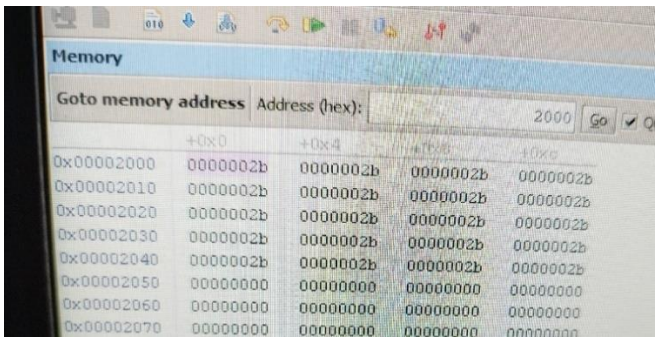


Рис. 13 Состояние порта PS/2 после нажатия клавиши на клавиатуре

Таким образом, бит RI устанавливается в 1, когда значение поля RAVAIL не равно 0. Также мы убедились в том, что бит RI можно очистить очисткой буфера FIFO (рис. 18)

Рис. 14 Буфер FIFO очищен, бит RI сброшен

Далее мы запустили программу и зажимали клавишу клавиатуры до вывода на терминальное окно AMP временных характеристик клавиатуры (рис. 19)



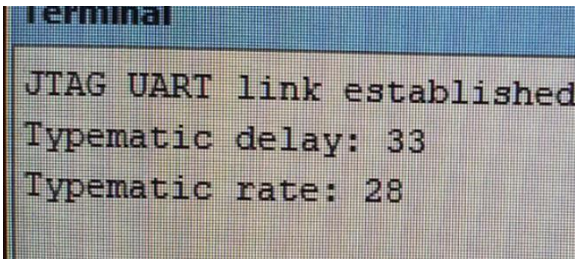


Рис. 15-16 Временные характеристики клавиатуры

Листинги программного файлов работы

Листинг 1. PS2_From_FIFO_to_HEX.s

```
.equ STACK_BASE,    0x081ffffc        /*Базовый адрес стека*/
.equ PS2_DATA,                  0x10000100
.equ HEX_DATA_L,    0x10000020
.equ HEX_DATA_H,    0x10000030
# Задержка перед сбросом состояния 7-сегментных индикаторов.
.equ HEX_RESET_DELAY, 0x7A120
# Значения для отображения 16-х цифр на 7-сегментных
индикаторах.
.equ HEX_0, 0x3F
.equ HEX_1, 0x06
.equ HEX_2, 0x5B
.equ HEX_3, 0x4F
.equ HEX_4, 0x66
.equ HEX_5, 0x6D
.equ HEX_6, 0x7D
.equ HEX_7, 0x07
.equ HEX_8, 0x7F
.equ HEX_9, 0x6F
.equ HEX_A, 0x77
.equ HEX_B, 0x7C
.equ HEX_C, 0x39
.equ HEX_D, 0x5E
.equ HEX_E, 0x79
.equ HEX_F, 0x71

.text
.global _start
_start:
        # Начальный адрес стека.
        movia sp, STACK_BASE
        # Очистка 7-сегментых индикаторов.
        call CLEAR_HEX_DISPLAY
        # Сброс состояния 7-сегментных индикаторов.
reset_hex:
        # Данные буфера FIFO, отображаемые на 7-
сегментных индикаторах.
        mov r8, zero
        # Количество байт данных, отображаемых на 7-
сегментных индикаторах.
        mov r9, zero
reset_hex_delay:
        # Задержка перед сбросом состояния 7-сегментных
индикаторов.
        movia r10, HEX_RESET_DELAY
read_loop:
        # Сброс состояния, если задержка равна 0.
        ble r10, zero, reset_hex
        # Чтение регистра PS2_Data в r2.
        call READ_PS2_DATA
        # Выделение 15 бита RVALID.
        srli r11, r2, 15
        andi r11, r11, 1
        # Декремент задержки.
        subi r10, r10, 1
        # Ожидание установки RVALID в 1.
        beq r11, zero, read_loop
        # Выделение поля (байта) DATA.
        andi r6, r2, 0xFF
        # Добавление байта к отображаемым данным буфера
FIFO.
        mov r4, r8
        mov r5, r9
        call APPEND_BYTE
        mov r8, r2
        mov r9, r3
        # Очистка 7-сегментных индикаторов.
        call CLEAR_HEX_DISPLAY
        # Вывод данных буфера FIFO на 7-сегментные
индикаторы.
        mov r4, r8
        # Кол-во 16-х цифр = кол-во байт данных * 2.
        muli r5, r9, 2
        # Вывод на индикаторы.
        call HEX_DISPLAY
br reset_hex_delay

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

/* Добавляет байт в конец (левую часть) исходного значения. */
APPEND_BYTE:
        # Сохранение регистров.
        stwio r8, (sp)
        mov r2, r4
        mov r3, r5
        movi r8, 3
        # Если длина последовательности < 4, пропускаем
смещение.
        bleu r3, r8, APPEND_BYTE_BODY
        # Смещаем последовательность влево.
        srli r2, r2, 8
        # Декриментируем длину последовательности.
        subi r3, r3, 1
APPEND_BYTE_BODY:
        # Определяем смещение.
        muli r8, r3, 8
        # Смещаем добавляемый байт влево.
        sll r8, r6, r8
        # Добавляем байт к значению.
        or r2, r2, r8
        # Инкриментируем длину последовательности.
        addi r3, r3, 1
        # Восстановление регистров.
        ldwio r8, (sp)
        ret

/* Очищает 7-сегментные индикаторы. */
CLEAR_HEX_DISPLAY:
        # Сохранение регистров.
        stwio r8, (sp)
        # Очистка первой четверки индикаторов.
        movia r8, HEX_DATA_L
        stwio zero, (r8)
        # Очистка первой четверки индикаторов.
        movia r8, HEX_DATA_H
        stwio zero, (r8)
        # Восстановление регистров.
        ldwio r8, (sp)
        ret

/*      Возвращает адрес 7-сегментного индикатора. */
HEX_DISPLAY_ADDR:
        # Сохранение регистров.
        stwio r8, (sp)
        stwio r9, -4(sp)
        # Маска на 3 бита (7 = 0b111).
        andi r8, r4, 7
        # Выбор регистра данных.
        movi r9, 3
```

```
        bgtu r8, r9, HEX_DISPLAY_ADDR_H
HEX_DISPLAY_ADDR_L:
        movia r2, HEX_DATA_L
        add r2, r2, r4
        br HEX_DISPLAY_ADDR_END
HEX_DISPLAY_ADDR_H:
        movia r2, HEX_DATA_H
        add r2, r2, r4
        subi r2, r2, 4
HEX_DISPLAY_ADDR_END:
        # Восстановление регистров.
        ldwio r8, (sp)
        ldwio r9, -4(sp)
        ret

/*      Выводит цифры на индикаторы. */
HEX_DISPLAY_DIGIT:
   # Сохранение регистров.
        stwio ra, (sp)
        stwio r2, -4(sp)
        stwio r8, -8(sp)
        stwio r9, -12(sp)
        subi sp, sp, 16
        # Получение адреса индикатора в r2.
        mov r8, r4
        mov r4, r5
        call HEX_DISPLAY_ADDR
        mov r4, r8
        # Маска на 16-ю цифру.
        andi r8, r4, 0xF
        # Получение значения для отображения цифры на
индикаторе.
        movia r9, HEX_DIGITS
        add r8, r8, r9
        ldb r8, (r8)
        call HEX_SAVE
        # Отображение последовательности цифр на
индикаторе.
        stwio r8, (r2)
        # Восставновление регистров.
        addi sp, sp, 16
        ldwio ra, (sp)
        ldwio r2, -4(sp)
        ldwio r8, -8(sp)
        ldwio r9, -12(sp)
        ret

/*      Выводит значение на 7-сегментные индикаторы. */
HEX_DISPLAY:
        # Сохранение регистров.
        stwio ra, (sp)
        stwio r4, -4(sp)
        stwio r5, -8(sp)
        stwio r8, -12(sp)
        subi sp, sp, 16
        # r8 хранит кол-во отображаемых цифр.
        mov r8, r5
        # r5 хранит текущий номер индикатора.
        mov r5, zero
HEX_DISPLAY_LOOP:
        # Цикл отображения цифр.
        beq r5, r8, HEX_DISPLAY_END
        call HEX_DISPLAY_DIGIT
        srli r4, r4, 4
        addi r5, r5, 1
        br HEX_DISPLAY_LOOP
HEX_DISPLAY_END:
        # Восстановление регистров.
        addi sp, sp, 16
        ldwio ra, (sp)
        ldwio r4, -4(sp)
        ldwio r5, -8(sp)
        ldwio r8, -12(sp)
        ret

/* Возвращает последовательность значений для отображения
на индикаторах */
HEX_SAVE:
        # Сохранение регистров.
        stwio ra, (sp)
        stwio r4, -4(sp)
        subi sp, sp, 8
        # Определяем смещение для записи значения в
последовательность.
        mov r4, zero
        beq r5, zero, HEX_SAVE_0
        movi r4, 1
        beq r5, r4, HEX_SAVE_1
        movi r4, 5
        beq r5, r4, HEX_SAVE_1
        movi r4, 2
        beq r5, r4, HEX_SAVE_2
        movi r4, 6
        beq r5, r4, HEX_SAVE_2
        movi r4, 3
        beq r5, r4, HEX_SAVE_3
        movi r4, 7
        beq r5, r4, HEX_SAVE_3
        movi r4, 4
        beq r5, r4, HEX_SAVE_0
HEX_SAVE_0:
        mov r12, zero
        or r12, r12, r8
        br HEX_SAVE_EX
HEX_SAVE_1:
        slli r8, r8, 8
        or r12, r12, r8
        br HEX_SAVE_EX
HEX_SAVE_2:
        slli r8, r8, 16
        or r12, r12, r8
        br HEX_SAVE_EX
HEX_SAVE_3:
        slli r8, r8, 24
        or r12, r12, r8
HEX_SAVE_EX:
        mov r8, r12
        # Восстановление регистров.
        addi sp, sp, 8
        ldwio ra, (sp)
        ldwio r4, -4(sp)
        ret

.data
HEX_DIGITS:
        .byte HEX_0, HEX_1, HEX_2, HEX_3, HEX_4, HEX_5,
HEX_6, HEX_7, HEX_8, HEX_9, HEX_A, HEX_B, HEX_C, HEX_D,
HEX_E, HEX_F

.end
```

Листинг 2. PS2_From_FIFO_to_OP_and_LCD.s

```
.equ STACK_BASE,    0x081ffffc
.equ PS2_DATA,      0x10000100
.equ LED_R,                 0x10000000
.equ LED_G,                 0x10000010
.equ LSD_ADDR_COMM,         0x10003050
.equ LSD_ADDR_VAL,          0x10003051
# Адреса памяти для записи данных.
.equ SDRAM, 0x1000
.equ SDRAM_EDGE, 0x2000
.equ TEMP, 0x5000
# Команда установки курсора в первую строку.
.equ LSD_POINTER_FIRST,0x80
# Команда установки курсора во вторую строку.
.equ LSD_POINTER_SECOND,0xc0
# Команда сброса экрана.
.equ LSD_DROP,0b00000001
# Значение пробела в аски.
.equ ASCII_WHITESPACE,0x20
# Значение нуля в аски.
.equ ASCII_NULL,0x30
# Значение буквы A в аски.
.equ ASCII_A,0x37

.text
.global _start
_start:
        # Начальный адрес стека.
        movia sp, STACK_BASE
        # Очистка сегмента памяти для записи данных из
буфера.
        movia r4, SDRAM
        movia r5, SDRAM_EDGE
        call CLEAR_MEM
        # Очистка LCD.
        call CLEAR
        # Адрес порта PS2.
        movia r3, PS2_DATA
        # Адреса LED.
        movia r7, LED_G
        # Количество символов, отображаемых на LCD.
        mov r11, r0
        # Адрес ОП для записи данных из буфера FIFO.
        movi r12, SDRAM
        # Адрес ОП для вывода на LCD.
        movi r13, SDRAM
```

```
        # Адрес ОП для временного хранения данных.
        movi r15, TEMP
read_ps2_buffer:
        # Погасить LED.
        stwio zero, (r7)
        # Маска 15 бита для проверки порта.
        movia r4, 0x8000
        # Маска для отсечения 1 байта регистра.
        movia r6, 0xff
        # Чтение PS2_DATA.
        ldwio r10, (r3)
        # Выделение 15 бита для проверки наличия записи.
        and r5,r10,r4
        bne r5, r4, read_ps2_buffer
        # Помещаем в r2 PS2_DATA * маска(0xff).
        and r2, r10, r6
        # Записываем r2 -> ОП с адресом r12.
        stb r2, (r12)
        # Смещаемся на 1 байт.
        addi r12, r12, 1
        mov r5, r10
        srli r5, r5, 16
        # Зажигание диодов.
        stwio r5, (r7)
        # Выводим на строку максимум 5 байт.
        movi r20, 5
        beq r11, r20, SHIFT
        # Выводим строку со следующим байтом ОП.
        addi r11, r11, 1
        br PRINT
SHIFT:
        # Если строка LCD заполнена, то смещаем адрес ОП
для вывода на LCD.
        addi r13, r13, 1
PRINT:
        mov r5, r13
        mov r6, r11
        call PRINTLCD
br read_ps2_buffer

/* Очистка области памяти. */
CLEAR_MEM:
        # Сохранение регистра.
        stwio r4, (sp)
CLEAR_MEM_LOOP:
        stwio zero, (r4)
        addi r4, r4, 4
        blt r4, r5, CLEAR_MEM_LOOP
        # Восстановление регистра.
        ldwio r4, (sp)
        ret

/* Вывод на LCD содержимого области памяти. */
PRINTLCD:
        # Сохранение регистров.
        stwio r9, (sp)
        stwio r11, -4(sp)
        stwio r7, -8(sp)
        stwio r5, -12(sp)
        stwio r4, -16(sp)
        stwio r10,-20(sp)
        stwio r12,-24(sp)
        stwio r13,-28(sp)
        stwio r14,-32(sp)
        stwio r15,-36(sp)
        stwio r16,-40(sp)
        stwio ra, -44(sp)
        subi sp, sp, 48
        # Очистка LCD.
        call CLEAR
        # Обнуляем счетчик.
        movia r4, 0
        movia r14,0
        mov r16,r15
LOOP:
        # Загрузка слова из области памяти.
        ldb r7, (r5)
        # Если счетчик = количеству символов, то выходим.
        beq r4, r6, DONE
        stb r7, (r16)
        # Увеличиваем счетчик 1.
        addi r4, r4, 1
        # Считываем следующий байт.
        addi r5, r5, 0x1
        movia r10,5
        bge r4,r10,ONLAST
        # Записываем следующий байт.
        addi r16, r16, 0x1
ONLAST:
        call RENDER
        br LOOP
DONE:
        # Восстановление регистров.
        addi sp, sp, 48
        ldwio r9, (sp)
        ldwio r11, -4(sp)
        ldwio r7, -8(sp)
        ldwio r5, -12(sp)
        ldwio r4, -16(sp)
        ldwio r10,-20(sp)
        ldwio r12,-24(sp)
        ldwio r13,-28(sp)
        ldwio r14,-32(sp)
        ldwio r15,-36(sp)
        ldwio r16,-40(sp)
        ldwio ra, -44(sp)
        ret

/* Получаем ASCII значение символа. */
GETASCII:
        movia r10,0x9
        # Если значение больше 9, приводим к букве, иначе к
цифре.
        ble  r9, r10,TONUM1
        # Добавляем A в ASCII чтобы получить представления
значения ячейки памяти в ASCII.
        addi r9, r9, ASCII_A
        ret
TONUM1:
        # Добавляем 0 в ASCII чтобы получить представления
значения ячейки памяти в ASCII.
        addi r9, r9, ASCII_NULL
        ret

/* Очистка LCD экрана */
CLEAR:
        # Кладем в r9 команду очищения экрана.
        movia r9,LSD_DROP
        # Кладем в r11 адрес команд LCD.
        movia r11, LSD_ADDR_COMM
        stbio r9, (r11)
        ret

/* Вывод в конце второй строки LCD количества выведенных
символов. */
PRINTCOUNT:
        mov r7,r6
        mov r12,r6
        movia r13,0
GETNUMBERSTOSTACK:
        # Если все проверили - выходим.
        beq r7,zero, MOVECURSOR
        movia r10,10
        # Получаем остаток от деления на 10.
        div r7,r7,r10
        mul r9,r7,r10
        sub r9,r12,r9
        mov r12,r7
        stwio r9,(sp)
        subi sp, sp, 4
        addi r13,r13,1
        br GETNUMBERSTOSTACK
MOVECURSOR:
        movia r10,LSD_POINTER_SECOND
        addi r10,r10,0x10
        sub r10,r10,r13
        # Кладем в r9 команду сдвига.
        mov  r9, r10
        # Сдвигаем экран.
        movia r11, LSD_ADDR_COMM
        # Сдвигаем курсор.
        stbio r9, (r11)
COUNTPRINT:
        # Если все проверили - выходим.
        beq r13,zero, END
        addi sp, sp, 4
        ldwio r9,(sp)
        addi r9, r9, ASCII_NULL
        # Получаем адрес индикатора.
        movia r11, LSD_ADDR_VAL
        # Печатаем цифру.
        stbio r9, (r11)
        addi r13,r13,-1
        br COUNTPRINT
END:
        # Кладем в r9 команду сдвига.
        movia  r9, LSD_POINTER_FIRST
        # Сдвигаем экран.
        movia r11, LSD_ADDR_COMM
        # Сдвигаем курсор.
```

```
        stbio r9, (r11)
        ret

RENDER:
        # Сохранение регистра.
        stwio ra, (sp)
        subi sp, sp, 4
        # Очистка LCD.
        call CLEAR
        # Вывод количества символов.
        call PRINTCOUNT
        movia r13,0
        mov r12,r15
RENDERLOOP:
        movia r10,5
        beq r13, r4, ENDRENDER
        beq r13,r10, ENDRENDER
        ldb r7, (r12)
        # Получаем второй символ байта данных, с помощью
маски.
        andi r9, r7, 0xf0
        # Сдвиг вправо чтобы получить нужный код.
        srli r9, r9, 0x4
        # Получаем ASCII значение.
        call GETASCII
        # Получаем адрес индикатора.
        movia r11, LSD_ADDR_VAL
        # Печатаем цифру.
        stbio r9, (r11)
        # Получаем первый символ байта данных, с помощью
маски.
        andi r9, r7, 0xf
        # Получаем ASCII значение.
        call GETASCII
        # Получаем адрес индикатора.
        movia r11, LSD_ADDR_VAL
        # Печатаем цифру.
        stbio r9, (r11)
        # Символ пробела.
        movia r9,  ASCII_WHITESPACE
        # Печатаем цифру.
        stbio r9, (r11)
        # Задержка.
        addi r12,r12,1
        addi r13,r13,1
        br RENDERLOOP
ENDRENDER:
        # Задержка.
        movia r9, 0
DELAY:
        # Вычитаем в цикле 1.
        addi r9,r9,-1
        # Если стало равно 0, то переходим к следующей
цифре.
        bgt r9,zero, DELAY
        movia r10,5
        movia r7,0
        blt r4,r10,RETURNRENDER
        movia r10,4
        mov r12,r15
SWAP:
        beq r7,r10,RETURNRENDER
        addi r13,r12,0x1
        ldb r9, (r13)
        stb r9, (r12)
        addi r12,r12,0x1
        addi r7,r7,1
        br SWAP
RETURNRENDER:
        # Восстановление регистра.
        addi sp, sp, 4
        ldwio ra, (sp)
        ret

.end
```

Листинг 3. PS2_Typematic_rate_Delay_to_JTAG.s

```
.equ STACK_BASE,      0x081ffffc
.equ PS2_DATA,                  0x10000100
.equ TIMER,                                 0x10002000
.equ JTAG_UART_BASE, 0x10001000
.equ DATA_MASK, 0xff
.equ RVALID_MASK, 0x8000
.equ NUM_OF_BYTES, 20
# Адрес памяти для записи данных.
.equ SDRAMTIME_START, 0x1000
# Смещение для адреса данных.
.equ DELTA, 0x1000

.text
.org 0x20
handler:
        # Сохраняем регистры.
        stwio r8, (sp)
        stwio r9, -4(sp)
        stwio r10, -8(sp)
        stwio r11, -12(sp)
        # Если прерывание не внешнее, то выходим.
        rdctl r10, ctl4
        beq r10, r0, exit
        # Декрементируем регистр ea на 1 команду.
        subi ea, ea, 4
        # Если прерывание не от клавиатуры, то выходим.
        andi r11, r10, 0x80
        beq r11, r0, exit
        # Игнорируем младший байт 0xFA (ACK) or 0xFE (Resend).
        movia r8, PS2_DATA
        ldwio r9, (r8)
        andi r9, r9, DATA_MASK
        # Если 0xFA (ACK) or 0xFE (Resend), то выходим.
        movia r10, 0xFA
        movia r11, 0xFE
        beq r9, r10, exit
        beq r9, r11, exit
        # Если BYTES_LEFT = 0, то запрещаем прерывания от
клавиатуры и выходим.
        movia r9, PS2_DATA
        movia r8, BYTES_LEFT
        ldwio r10, (r8)
        bne r10, r0, continue
        stbio r0, 4(r9)
        br exit
continue:
        # Уменьшаем счетчик обработанных байт.
        subi r10, r10, 1
        stwio r10, (r8)
        # Получет значение времени таймера.
        movia r8, TIMER
        stwio r0, 0x10(r8)
        ldhuio r10, 0x10(r8)
        ldhuio r11, 0x14(r8)
        # Записываем значение времени таймера в ОП.
        movia r8, SDRAMTIME
        ldwio r9, (r8)
        sth r10, (r9)
        sth r11, 2(r9)
        # Смещаем адрес SDRAMTIME на 4 байта.
        addi r9, r9, 4
        stwio r9, (r8)
        # Получаем значение байта PS2 в r10.
        movia r8, PS2_DATA
        ldwio r10, (r8)
        andi r10, r10, DATA_MASK
        # Записываем значения байта в ОП.
        movia r8, SDRAMKEY
        ldwio r9, (r8)
        stwio r10, (r9)
        # Смещаем адрес SDRAM на 4 байта.
        addi r9, r9, 4
        stwio r9, (r8)
exit:
        # Восстанавливаем регистры.
        ldwio r8, (sp)
        ldwio r9, -4(sp)
        ldwio r10, -8(sp)
        ldwio r11, -12(sp)
        eret

.global _start
_start:
        # Начальный адрес стека.
        movia sp, 0x3fffffC
        # Адреса сегментов памяти.
        movi r8, SDRAMTIME_START
        addi r9, r8, DELTA
        addi r10, r9, DELTA
        addi r11, r10, DELTA
        # Инициализация переменных.
        movia r12, SDRAMTIME
        stwio r8, (r12)
        movia r12, SDRAMKEY
        stwio r9, (r12)
        movia r12, SDRAMDIFF
        stwio r10, (r12)
        movia r12, BYTES_LEFT
        movia r13, NUM_OF_BYTES
        stwio r13, (r12)
```

```
        # Очистка сегментов памяти.
        mov r4, r8
        mov r5, r9
        call CLEAR_MEM
        mov r4, r9
        mov r5, r10
        call CLEAR_MEM
        mov r4, r10
        mov r5, r11
        call CLEAR_MEM
        # Адрес регистра PS2_Data
        movia r2, PS2_DATA
        # Подготовка масок.
        movia r4, RVALID_MASK
        movia r6, DATA_MASK
flush_buffer_init:
        # Очистка буфера.
        ldwio r10, (r2)
        and r5,r10,r4
        beq r5, r4, flush_buffer_init
        # Разрешаем прерывания.
        movi r15,0x80
        wrctl ienable,r15
        movi r7,1
        wrctl status,r7
        # Разрешить прерывания от клавиатуры в контроллере.
        movi r9, 1
        stbio r9, 0x4(r2)
        # Настройка таймера.
        movia r8, TIMER
        # Команда STOP.
        movi r3, 0b1000
        # Остановка таймера.
        sthio r3, 4(r8)
        sthio r0, (r8)
        # Максимальное значение.
        movia   r18, 0xffff
        sthio   r18,8(r8)
        sthio   r18,0xC(r8)
        # Команда START.
        movi r3, 0b0100
        # Запуск таймера.
        sthio r3, 4(r8)
WAIT_LOOP:
        # Ожидание, пока прочитаются все байты.
        movia r3, BYTES_LEFT
        ldwio r3, (r3)
        bne r3, r0, WAIT_LOOP
        # Адрес начала сегмента памяти с таймкодами.
        movia r4, SDRAMTIME_START
        # Адрес конца сегмента памяти с таймкодами.
        movia r2, SDRAMTIME
        ldwio r2, (r2)
        subi r9, r2, 4
        # Адрес сегмента памяти с задержками.
        addi r5, r4, DELTA
        addi r5, r5, DELTA
LOOP_DIFF:
        # Запись задержек клавиатуры.
        ldwio r6, (r4)
        addi r4, r4, 4
        ldwio r7, (r4)
        sub r8, r6, r7
        movia r10, 50000
        divu r8, r8, r10
        stwio r8, (r5)
        addi r5, r5, 4
        bltu r4, r9, LOOP_DIFF
        # Вывод typematic delay и typematic rate в консоль.
        movia r4, TYPEMATIC_DELAY_STR
        # Отправляем строку в JTAG UART.
        call LOG_STR
        movia r8, SDRAMTIME_START
        addi r8, r8, DELTA
        addi r8, r8, DELTA
        ldwio r4, (r8)
        # Печать символов.
        call LOG_NUM
        # Печать пробела.
        call LOG_LINE
        movia r4, TYPEMATIC_RATE_STR
        # Отправляем строку в JTAG UART.
        call LOG_STR
        addi r8, r8, 4
        ldwio r4, (r8)
        movi r5, 1000
        div r4, r5, r4
        # Печать символов.
        call LOG_NUM
        # Печать пробела.
        call LOG_LINE
        br endless

/* Очистка области памяти. */
CLEAR_MEM:
        # Сохранение регистра.
        stwio r4, (sp)
CLEAR_MEM_LOOP:
        stwio zero, (r4)
        addi r4, r4, 4
        blt r4, r5, CLEAR_MEM_LOOP
        # Восстановление регистра.
        ldwio r4, (sp)
        ret

/* Отправляет символ из r4 в JTAG UART. */
LOG_CHAR:
        # Сохранение регистров.
        stwio r8, (sp)
        stwio r9, -4(sp)
        stwio 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.
        ldwio r8, (sp)
        ldwio r9, -4(sp)
        ldwio r10, -8(sp)
        ret

/* Отправляет число из r4 в JTAG UART. */
LOG_NUM:
        # Сохранение регистров.
        stwio ra, (sp)
        stwio r4, -4(sp)
        stwio r8, -8(sp)
        stwio r9, -12(sp)
        stwio r10, -16(sp)
        stwio r11, -20(sp)
        subi sp, sp, 24
        mov r8, r4
        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.
        stwio 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
        ldwio r4, (sp)
        addi r4, r4, 0x30
        call LOG_CHAR
        bne sp, r11, LOG_NUM_DIGITS
        # Восстановление регистров.
        addi sp, sp, 24
        ldwio ra, (sp)
        ldwio r4, -4(sp)
        ldwio r8, -8(sp)
        ldwio r9, -12(sp)
        ldwio r10, -16(sp)
        ldwio r11, -20(sp)
        ret

/* Отправляет строку по адресу r4 в JTAG UART. */
LOG_STR:
        # Сохранение регистров.
        stwio ra, (sp)
        stwio r4, -4(sp)
```

```
        stwio r8, -8(sp)
        subi sp, sp, 12
        # Адрес текущего символа в r8.
        mov r8, r4
LOG_STR_LOOP:
        # Печать символов.
        ldb r4, (r8)
        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
        ldwio ra, (sp)
        ldwio r4, -4(sp)
        ldwio r8, -8(sp)
        ret

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

endless:
        br endless

.data
```

Выводы

В процессе работы мы изучили порт PS/2, подключать через него клавиатуру, а также изучили команды для работы с клавиатурой, ее характеристики и их настройки.