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

«ОРГАНИЗАЦИЯ ЭВМ И СИСТЕМ»

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.

Цель работы

Изучение порта UART JTAG и его практическое применение для взаимодействия процессорной системы с инструментальным компьютером через терминальное окно Intel Monitor Program.

Исходные файлы лабораторной работы

Программа, демонстрирующая использование порта JTAG UART в процессорной системе, входит в состав приложения IMP. Она доступна в разделе *Sample program* под именем *JTAG UART.s*. Её листинг с подробными комментариями приведен в приложении 2. Там же приводится фрагмент программы TEST DE 2-70 Media Computer, в котором выполняется взаимодействие с JTAG UART портом.

В файле jtag_uart содержатся две функции **putchar_uart** и **getchar_uart**. Первая функция **putchar_uart** проверяет поле wspace регистра управления порта. Если оно не равно нулю, значит место в буфере FIFO для записи есть, в этом случае байт из R4 пересылается в UART. Признаком успешного выполнения функции является возврат в регистре R2 нулевого значения. Если места в буфере FIFO нет, то возвращается -1. Функция будет полезна для написания функции вывода текстовой строки в терминальное окно инструментального ПК в первой части работы.

Вторая функция **getchar_uart** проверяет бит RAVAIL в регистре данных порта UART. Если данные для чтения в буфере FIFO есть, то считанный из него байт помещается в R2. Если данных в буфере нет, то в R2 возвращается -1. Функция будет полезна для написания функции ввода текстовой строки, печатаемой на клавиатуре инструментального ПК, в память процессорной системы. Такую функцию потребуется создать во второй части работы.

В файле display содержатся две подпрограммы **hex_display** и **led_display**, которые отображают содержимое регистра R4 на двух 7-сегментных индикаторах и на красных светодиодах, соответственно. Они будут полезны для изучения ASCII кодов символов, печатаемых на клавиатуре инструментального ПК и отправляемых в буфер FIFO порта JTAG UART.

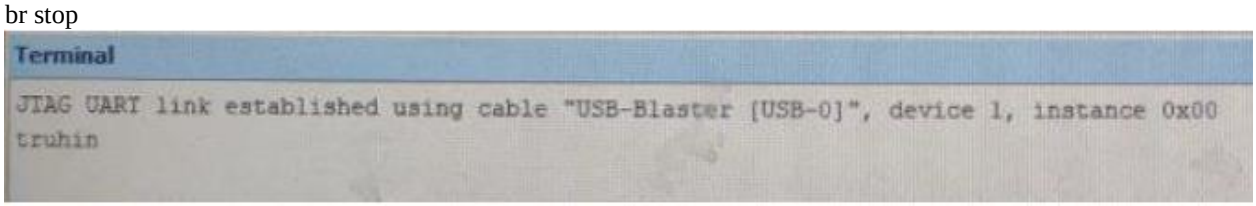
В файле **print_lcd** представлена одноименная программа, выполняющая вывод текста в режиме бегущей строки на LCD дисплей стенда. Адрес строки передается через регистр R4. Эта программа будет полезна для демонстрации прерываний, происходящих при работе с портом JTAG UART.

Выполнение заданий лабораторной работы

1. Запись в **UART JTAG** (вывод информации).

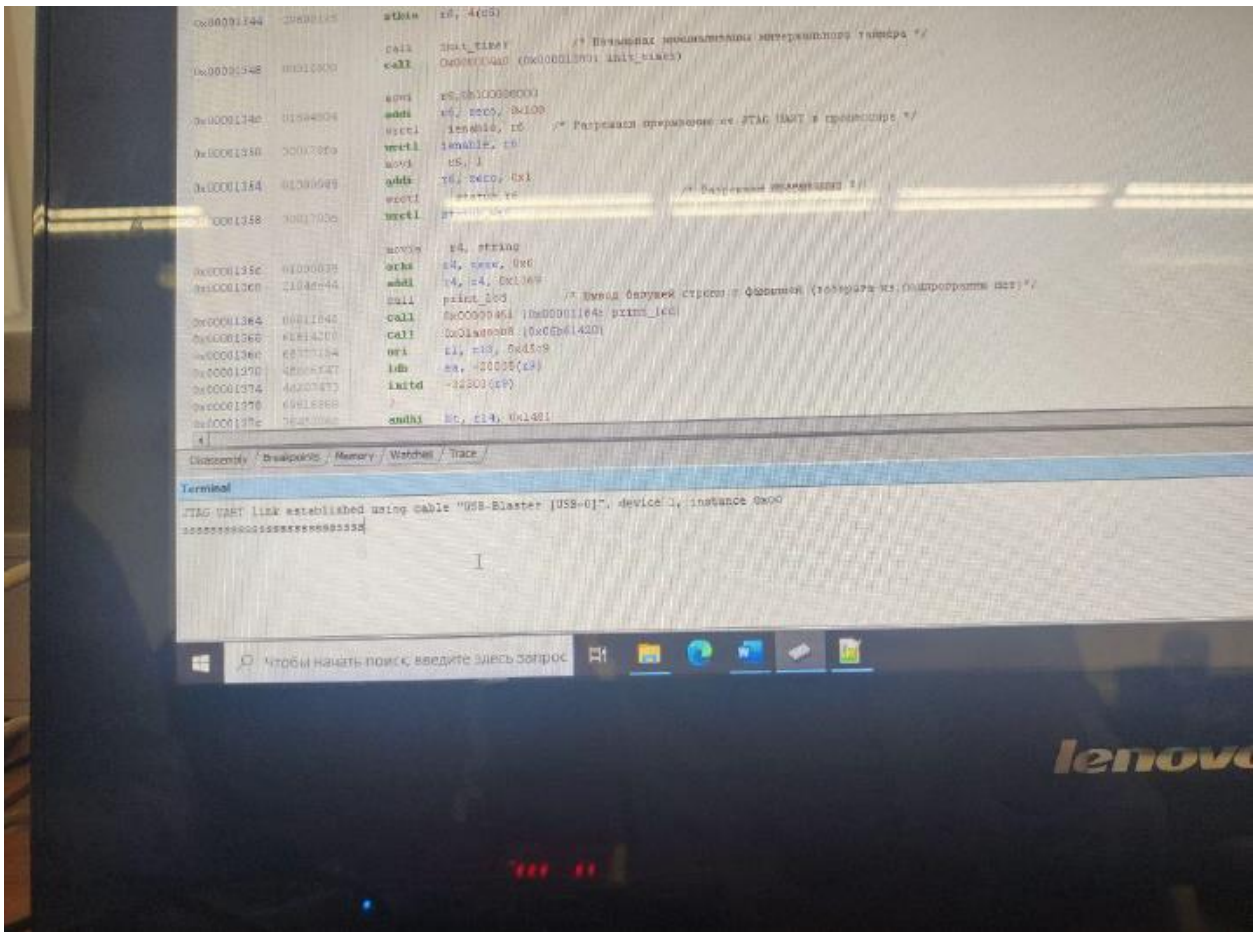
1.3. Написали программу, которая записывает ASCII код заглавной буквы наших фамилий (KOS) в порт UART JTAG. Функция выполняет анализ поля WSPACE в регистре управления UART, отражающего наличие свободного места в буфере FIFO для записываемых данных. Если свободное место в буфере есть, то символ записывается в буфер и в регистре R2 возвращается нулевое значение, свидетельствующее об успешном выполнении функции. В противном случае, если свободного места в буфере нет, функция возвращает в регистре R2 значение -1, свидетельствующее об неуспешном её завершении.

```
.equ STACK_BASE, 0x081ffffc /*Базовый адрес стека*/
.extern putchar_uart
.text
.global _start
_start:
movi r2, 0x777
movia sp, STACK_BASE /*Заносим в sp адрес последнего слова в SRAM*/
movi r4, 'K' /*Передаем в качестве параметра символ 'KOS'*/
call putchar_uart /*Выводим символ в UART*/
stop:
br stop
```



1.4. Модифицировали программу из пункта 1 таким образом, чтобы заданный символ выводился в порт UART многократно, то есть в цикле. Причем добавили программную задержку в цикл вывода.

```
.equ STACK_BASE, 0x081ffffc /*Базовый адрес стека*/
.extern putchar_uart
.text
.global _start
_start:
movia sp, STACK_BASE /*Заносим в sp адрес последнего слова в SRAM*/
movi r3, 0
movi r4, 'K' /*Передаем в качестве параметра символ 'K'*/
delay_start:
/*movia r10, 1*/
met:
/*subi r10, r10, 1
bne r10, zero, met*/
call putchar_uart /*Выводим символ в UART*/
bne r2, zero, stop
addi r3, r3, 1
br met
stop:
br stop /*бесконечный цикл*/
```

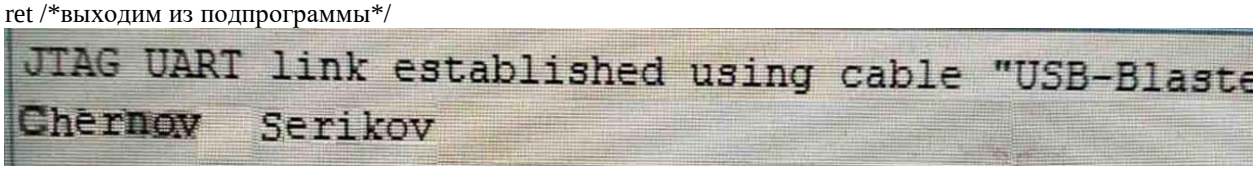


Заметим, что буфер FIFO никогда не переполняется, и программа выводит символ в терминальное окно IMP бесконечно. Буфер FIFO успевает вывести в терминальное окно значительно больше символов, чем в пункте 4, до того, как произойдет его переполнение.

1.7. Написали программу, которая выводит в терминальное окно IMP некоторое сообщение. Сообщение поместили в сегменте данных программы. Для этого использовали директиву. asciz ассемблера. Программа анализирует считываемый из строки байт, и если он равен нулю, то вывод строки завершается.

```
//main.s
.equ STACK_BASE, 0x081ffffc /*Базовый адрес стека*/
.text
.global _start
_start:
movia sp, STACK_BASE /*Заносим в sp адрес последнего слова в SRAM*/
movi r4, string /*Параметр - адрес строки в сегменте данных*/
call puts /*Вывод строки в UART*/
stop:
br stop /*бесконечный цикл*/
.data
string:
.asciz "Chernov Serikov" /*Строка для вывода*/
.end
//puts.s
.equ STACK_BASE, 0x081ffffc /*Базовый адрес стека*/
.extern putchar_uart /*запись в UART*/
/***************************** Секция кода *********************************/
.text
```

```
/*****************************************************************************
* Функция вывода в UART нуль-терминированной строки
* Параметр:
В r4 - адрес строки, оканчивающейся '\0'
*****************************************************************************/
.global puts
puts:
subi sp, sp, 12 /*создаем стековый кадр*/
stw ra, 8(sp)
stw r4, 4(sp)
stw r16, 0(sp)
mov r16, r4 /*сохраняем базовый адрес строки в r16*/
br 1f /*сразу переходим к проверке условия*/
0:
call putchar_uart
bne r2, r0, 0b /*если буфер переполнен, пытаемся повторить*/
addi r16, r16, 1 /*переходим к рассмотрению следующего байта*/
1:
ldb r4, (r16) /*если мы дошли до нулевого байта*/
bne r4, r0, 0b /*то выходим из цикла*/
ldw ra, 8(sp) /*восстанавливаем регистры из стека*/
ldw r4, 4(sp)
ldw r16, 0(sp)
addi sp, sp, 12
ret /*выходим из подпрограммы*/
```



Чтение из **UART JTAG (ввод информации).**

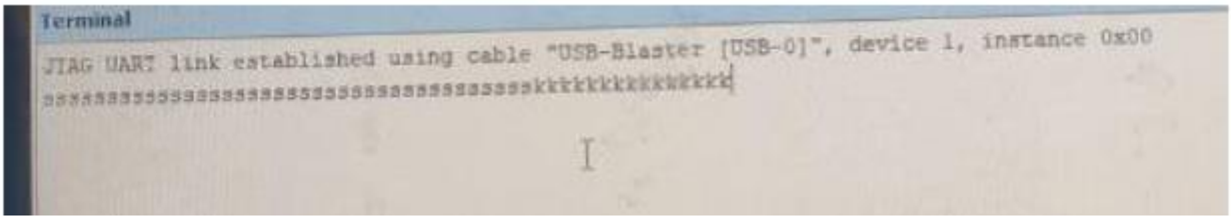
2.1. Написали программу, которая выполняет чтение из порта UART JTAG одного байта с помощью функции **getchar_uart**.

/main.s

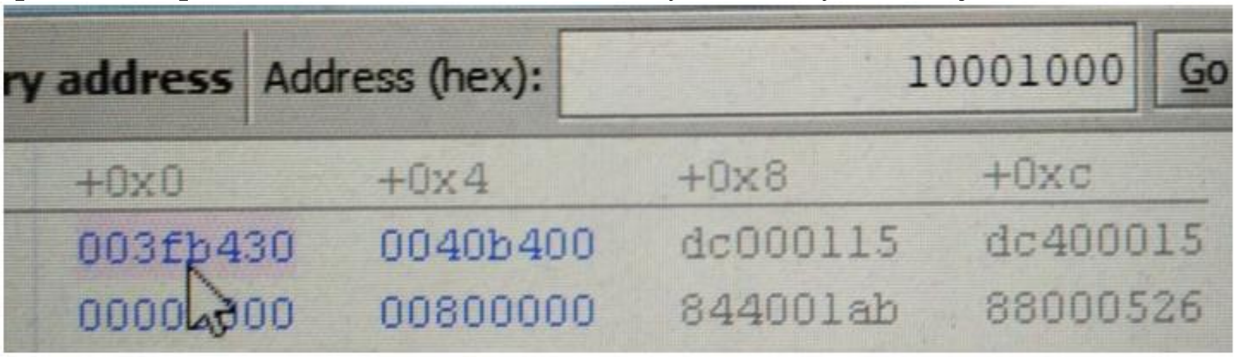
```
.equ STACK_BASE, 0x081ffffc /*Базовый адрес стека*/
.equ EOF, -1 /*Сообщение об ошибке, гарантированно не совпадает с кодами символов*/
.extern putchar_uart /*запись в UART*/
.extern getchar_uart /*чтение из UART*/
.text
.global _start
_start:
movia sp, STACK_BASE /*Заносим в sp адрес последнего слова в SRAM*/
movi r5, EOF /*сохраняем в r5 код EOF*/
0:
call getchar_uart /*считываем байт из UART*/
beq r2, r5, 0b /*если символов в буфере нет (EOF), повторяем*/
mov r4, r2 /*считанный байт заносим в регистр-параметр*/
1:
call putchar_uart /*отправляем байт обратно в UART*/
bne r2, r0, 1b /*если буфер отправки переполнен, ждем*/
br 0b /*ожидаем ввода нового символа*/
.end
```

```
.equ JTAG_UART_BASE, 0x10001000 /*Базовый адрес JTAG UART*/
.equ EOF, -1 /*Сообщение об ошибке, гарантированно не совпадает с кодами символов*/
.text
/UART.s/
* Функция записи символа в JTAG UART (неблокирующая)
* Соответствует NIOS 2 ABI
* Сигнатура для C: int putchar_uart(int symbol);
* Аргумент:
* В r4 - код записываемого сивола
* Возвращаемое значение в r2:
* 0 - успешное завершение
* EOF - неудача (буфер записи JTAG UART переполнен)
**************************************************************/
.global putchar_uart
putchar_uart:
subi sp, sp, 12 /*создаем стековый кадр*/
stw ra, 8(sp)
stw r16, 4(sp)
stw r17, 0(sp)
movia r16, JTAG_UART_BASE /*заносим базовый адрес JTAG UART в r16*/
ldhuio r17, 6(r16) /*проверяем поле WSPACE регистра управления*/
beq r17, r0, overflow /*выходим, если буфер JTAG UART переполнен*/
stbio r4, 0(r16) /*записываем байт в JTAG UART*/
mov r2, r0 /*возвращаемый код результата 0*/
br 0f
overflow:
movi r2, EOF /*возвращаемый код EOF*/
0:
ldw ra, 8(sp) /*восстанавливаем регистры из стека*/
ldw r16, 4(sp)
ldw r17, 0(sp)
addi sp, sp, 12
ret /*выходим из подпрограммы*/
/**************************************************************
* Функция считывания символа из JTAG UART (неблокирующая)
* Сигнатура для C: int getchar_uart();
*
* Возвращаемое значение в r2:
* 0..255 - считанный байт
* EOF - неудача (буфер записи JTAG UART переполнен)
**************************************************************/
.global getchar_uart
getchar_uart:
subi sp, sp, 16 /*создаем стековый кадр*/
stw ra, 12(sp)
stw r16, 8(sp)
stw r17, 4(sp)
stw r18, 0(sp)
movia r16, JTAG_UART_BASE /*заносим базовый адрес JTAG UART в r16*/
ldwio r17, 0(r16) /*считываем регистр данных JTAG UART*/
andi r18, r17, 0x8000 /*проверяем значение бита RVALID регистра данных*/
beq r18, r0, empty /*если данных нет, то выходим*/
andi r2, r17, 0xFF /*записываем принятый байт в регистр возвращаемого результата*/
br 0f
empty:
movi r2, EOF /*иначе возвращаем EOF*/
0:
ldw ra, 12(sp) /*восстанавливаем регистры из стека*/
ldw r16, 8(sp)
ldw r17, 4(sp)
ldw r18, 0(sp)
addi sp, sp, 16
ret /*выходим из подпрограммы*/
```

.end



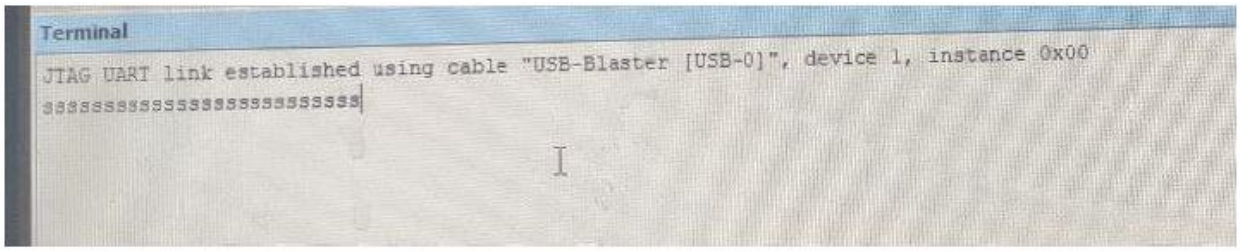
2.3. Установили курсор мыши в терминальное окно IMP и напечатали на клавиатуре цифру 0. После остановки программы в контрольной точке проанализировали поле RAVAIL, используя вкладку Memory.



2.4. Модифицировали программу из предыдущего пункта таким образом, чтобы читаемые из UART байты дополнительно отображались на двух HEX индикаторах и на красных светодиодах.

```
.equ STACK_BASE, 0x081ffffc /*Базовый адрес стека*/
.equ EOF, -1 /*Сообщение об ошибке, гарантированно не совпадает с кодами символов*/
.extern putchar_uart /*запись в UART*/
.extern getchar_uart /*чтение из UART*/
.extern hex_display отображение на семисегментных индикаторах*/
.extern led_display отображение на светодиодах*/
.text
.global _start
_start:
movia sp, STACK_BASE /*Заносим в sp адрес последнего слова в SRAM*/
movi r5, EOF /*сохраняем в r5 код EOF*/
0:
call getchar_uart /*считываем байт из UART*/
beq r2, r5, 0b /*если символов в буфере нет (EOF), повторяем*/
mov r4, r2 /*считанный байт заносим в регистр-параметр*/
call hex_display отображение на семисегментных индикаторах*/
call led_display отображение на светодиодах*/
1:
call putchar_uart /*отправляем байт обратно в UART*/
bne r2, r0, 1b /*если буфер отправки переполнен, ждем*/
br 0b /*ожидаем ввода нового символа*/
.end
```

2.5. Внесли изменения в программу из предыдущего пункта таким образом, чтобы напечатанная на клавиатуре инструментального ПК текстовая строка целиком пересылалась обратно в порт UART JTAG только после ввода символа Enter.



Ввод из **UART JTAG** в режиме прерывания.

3.1. Реализовали блокирующий ввод текстовой строки, печатаемой на клавиатуре инструментального компьютера, в режиме прерывания. Для этого основная программа разрешает прерывания по чтению от UART JTAG и выполняет вывод строки с нашими фамилиями на дисплей LCD в режиме бегущей строки. Ввод текста в терминальном окне инструментального компьютера приводит к прерыванию основной программы, и обработчик прерывания далее осуществляет ввод строки в ОП и вывод в UART JTAG всей строки, пока не обнаружит символ конца строки. В это время вывод бегущей строки на LCD дисплей прекратиться.

```
/main.s
.equ STACK_BASE, 0x081ffffc /*Базовый адрес стека*/
.text
.global _start
_start:
movia sp, STACK_BASE /*Заносим в sp адрес последнего слова в SRAM*/
movie r5, JTAG_UART_BASE
movi r6, 1
stbio r6, 4(r5) /* Разрешаем прерывания по чтению от JTAG UART */
movi r6,0b100000000
wrctl ienable, r6 /* Разрешаем прерывания от JTAG UART в процессоре */
movi r6, 1
wrctl status,r6 /* Разрешаем прерывания */
movia r4, string
call print_lcd /* Вывод бегущей строки с фамилией (возврата из подпрограммы нет)*/
.data
string:
.asciz "Chernov Serikov"
.end
/interruptHandler
* ОБРАБОТЧИК СБРОСА
* "ax" требуется для того, чтобы определить секцию как исполняемую.
* AMP автоматически размещает секцию сброса по адресу, определяемому в настройках
* процессора в SOPC Builder.
*/
.section .reset, "ax"
movia r2, _start
jmp r2 /* Переходим в основную программу */
/*****************************************************************************
* ОБРАБОТЧИК ИСКЛЮЧЕНИЙ
* "ax" требуется для того, чтобы определить секцию как исполняемую.
* AMP автоматически размещает секцию сброса по адресу, определяемому в настройках
```

```
* процессора в SOPC Builder.
*/
.section .exceptions, "ax"
.global EXCEPTION_HANDLER /*Определяем процедуру как глобальную*/
subi sp, sp, 16 /* Изменяем адрес указателя стека */
stw et, 0(sp) /*Сохраняем содержимое регистра et в стеке*/
rdctl et, ipending
beq et, zero, SKIP_EA_DEC /* Если прерывание не внешнее, то переходим на SKIP_EA_DEC */
subi ea, ea, 4 /* декрементируем регистр ea на 1 команду */
SKIP_EA_DEC:
stw ea, 4(sp) /* Сохраняем регистры в стеке */
stw ra, 8(sp)
stw r22, 12(sp)
rdctl et, ipending
bne et, zero, CHECK_LEVEL_0 /* Если прерывание внешнее, то переходим на CHECK_LEVEL_0*/
NOT_EI: Прерывание произошло в случае встречи невыполнимой команды*/
br END_ISR /* или команды TRAP */
CHECK_LEVEL_0: /* Проверка, является ли прерывание прерыванием от таймера IRQ0 */
andi r22, et, 0b1 /* Если бит 0b1 регистра et не равен 1, то переходим */
beq r22, zero, CHECK_LEVEL_1 /* к проверке, является ли прерывание прерыванием от кнопок*/
/*call TIMER_ISR*/ /* Обработчик прерывания от таймера*/
br END_ISR
CHECK_LEVEL_1: /*Проверка, является ли прерывание прерыванием от JTAG UART IRQ10*/
andi r22, et, 0b100000000
beq r22, zero, END_ISR /* Если бит 0b100000000 регистра et не равен 1, то */
call JTAG_UART_ISR /* Вызов обработчика прерываний от JTAG UART*/
END_ISR:
ldw et, 0(sp) /* Восстанавливаем значения регистров из стека */
ldw ea, 4(sp)
ldw ra, 8(sp)
ldw r22, 12(sp)
addi sp, sp, 16
eret
.end
/JTAG_UART_isr.s
/*****************************************************************************
*****************************************************************************/
.global JTAG_UART_ISR
JTAG_UART_ISR:
subi sp, sp, 8 /*Сохраняем регистры в стеке */
stw ra, 4(sp)
stw r4, 0(sp)
movia r4, string /*Адрес места для вводимой строки - в r4*/
call gets /*ввод строки из UART до символа перевода строки (LF)*/
call puts /*вывод нуль-терминированной строки в UART*/
ldw ra, 4(sp) /*возвращаем регистры из стека*/
ldw r4, 0(sp)
addi sp, sp, 8
ret
.data
string: .skip 256, '^' /*Место в памяти, выделенное для вводимой строки*/
.end
```

3.2. Модифицировали программу из предыдущего пункта таким образом, чтобы теперь ввод строки стал неблокирующим. То есть, после ввода каждого символа обработчик прерывания возвращает управление основной программе. Для этого надо было изменить JTAG_UART_isr.s.

```
.equ EOF, -1 /*Сообщение об ошибке, гарантированно не совпадает с кодами символов*/
/*********************************************************************************
* Процедура обработки прерываний от JTAG UART
*********************************************************************************/
.global JTAG_UART_ISR
JTAG_UART_ISR:
subi sp, sp, 8 /*Сохраняем регистры в стеке */
stw ra, 4(sp)
movi r5, EOF /*сохраняем в r5 код EOF*/
0:
call getchar_uart /*считываем байт из UART*/
beq r2, r5, exit /*если символов в буфере нет (EOF), выходим*/
mov r4, r2 /*считанный байт заносим в регистр-параметр*/
call hex_display /*выводим на hex-индикаторы*/
call led_display /*выводим на светодиоды*/
1:
call putchar_uart /*отправляем байт обратно в UART*/
bne r2, r0, 1b /*если буфер отправки переполнен, ждем*/
br 0b /*ожидаем ввода нового символа*/
exit:
ldw ra, 4(sp) /*возвращаем регистры из стека*/
ldw r4, 0(sp)
addi sp, sp, 8
ret /*возврат из обработчика прерывания*/
.end
```

Ввод из UART JTAG в режиме прерывания с использованием вложенного прерывания таймера.

4.1. Модифицировали программу чтения из порта JTAG UART так, чтобы набранный на клавиатуре ПК символ выводился в терминальное окно ПК каждые 500мс.

```
/mais.s
.equ STACK_BASE, 0x081ffffc /*Базовый адрес стека*/
.equ JTAG_UART_BASE, 0x10001000 /*Базовый адрес JTAG UART*/
.text
.global _start
_start:
movia sp, STACK_BASE /* Заносим в sp адрес последнего слова в SRAM */
movia r5, JTAG_UART_BASE
movi r6, 1
stbio r6, 4(r5) /* Разрешаем прерывания по чтению от JTAG UART */
call init_timer /* Начальная инициализация интервального таймера */
movi r6,0b100000000
wrctl ienable, r6 /* Разрешаем прерывания от JTAG UART в процессоре */
movi r6, 1
```

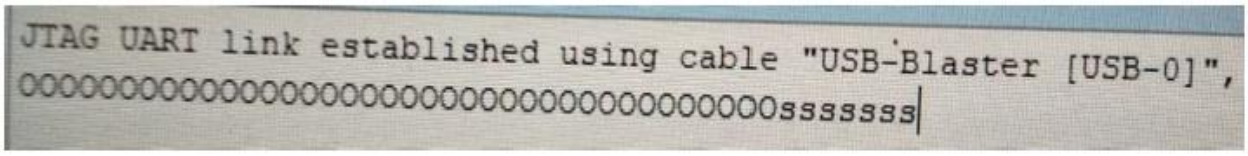
```
wrctl status,r6 /* Разрешаем прерывания */
movia r4, string
call print_lcd /* Вывод бегущей строки с фамилией (возврата из подпрограммы нет)*/
.data
string:
.asciz "Kolchenko Osokin Smurenkov"
.end
/display.s
.equ HEX7_4_BASE, 0x10000030
.equ HEX3_0_BASE, 0x10000020
.equ LEDS_BASE, 0x10000000 /*Адрес светодиодов*/
.text
/*****************************************************************************
* Подпрограмма, которая выводит значение на 7-сегментные индикаторы
* Число для вывода передается через регистр r4
* Адрес сегментов HEX3..0 определяется константой HEX3_0_BASE
* Адрес сегментов HEX7..4 определяется константой HEX7_4
*****************************************************************************/
.global hex_display
hex_display:
subi sp, sp, 20
stw r10, 16(sp) /*Сохраняем используемые регистры в стеке*/
stw r11, 12(sp)
stw r12, 8(sp)
stw r13, 4(sp)
stw r14, 0(sp)
/*Выводим число на сегмент HEX0*/
movie r12,DECODE_TABLE/*Записываем в r12 адрес памяти, с которого начинается таблица
декодирования*/
movie r14,HEX3_0_BASE /*Записываем в r14 адрес семисегментных индикаторов*/
mov r10, r4 /*Переписываем число для вывода*/
andi r10, r10, 0x000F /*Получаем младшие 4 бита числа, которые будут являться смещением в таблице
декодирования*/
add r11, r12, r10 /*Получаем адрес ячейки памяти, в которой содержится представление числа для 7-
сегментных индикаторов*/
ldb r13,0(r11)/*В P13 считываем из памяти представление числа для 7-сегментных индикаторов*/
stb r13, 0(r14) /*Выводим число на дисплей*/
/*Выводим число на сегмент HEX1*/
andi r10, r10, 0x00F0
srli r10,r10,4 /*Сдвигаем результат вправо*/
mov r11,r0
add r11, r12, r10
ldb r13, 0(r11)
stb r13, 1(r14)
ldw r10, 16(sp) /*Сохраняем используемые регистры в стеке*/
ldw r11, 12(sp)
ldw r12, 8(sp)
ldw r13, 4(sp)
ldw r14, 0(sp)
addi sp, sp, 20
ret
/*Таблица декодирования для 7-сегментных индикаторов. Каждый бит кодирует 1 сегмент индикатора*/
DECODE_TABLE:
.byte 0b00111111, 0b00000110, 0b01011011, 0b01001111
.byte 0b01100110, 0b01101101, 0b01111101, 0b00000111
.byte 0b01111111, 0b01101111, 0b01110111, 0b01111100
.byte 0b00111001, 0b01011110, 0b01111001, 0b01110001
.global led_display
led_display:
subi sp, sp, 4
stw r16, 0(sp)
movia r16, LEDS_BASE
stwio r4 ,0(r16) /* Выводим результат на светодиоды */
```

```
ldw r16, 0(sp)
addi sp, sp, 4
ret
.end
/interruptHandler.s
/*******************************************************************************
* ОБРАБОТЧИК СБРОСА
* "ax" требуется для того, чтобы определить секцию как исполняемую.
* AMP автоматически размещает секцию сброса по адресу, определяемому в настройках
*/
.section
.reset, "ax"
movie r2, _start
jmp r2 /* Переходим в основную программу */
/*******************************************************************************
* ОБРАБОТЧИК ИСКЛЮЧЕНИЙ
* "ax" требуется для того, чтобы определить секцию как исполняемую.
* AMP автоматически размещает секцию сброса по адресу, определяемому в настройках
* процессора в SOPC Builder.
*/
.section .exceptions, "ax"
.global EXCEPTION_HANDLER /*Определяем процедуру как глобальную*/
EXCEPTION_HANDLER: /*Процедура обработки прерываний*/
subi sp, sp, 16 /* Изменяем адрес указателя стека */
stw et, 0(sp) /*Сохраняем содержимое регистра et в стеке*/
rdctl et, ipending
beq et, zero, SKIP_EA_DEC /* Если прерывание не внешнее, то переходим на SKIP_EA_DEC */
subi ea, ea, 4 /* декрементируем регистр ea на 1 команду */
SKIP_EA_DEC:
stw ea, 4(sp) /* Сохраняем регистры в стеке */
stw ra, 8(sp)
stw r22, 12(sp)
rdctl et, ipending
bne et, zero, CHECK_LEVEL_0
/* Если прерывание внешнее, то переходим на CHECK_LEVEL_0*/
NOT_EI: /* Прерывание произошло в случае встречи невыполнимой команды*/
br END_ISR /* или команды TRAP */
CHECK_LEVEL_0:/* Проверка, является ли прерывание прерыванием от таймера IRQ0 */
andi r22, et, 0b1 /* Если бит 0b1 регистра et не равен 1, то переходим */
beq r22, zero, CHECK_LEVEL_1
/* к проверке, является ли прерывание прерыванием от JTAG UART*/
call TIMER_ISR
br END_ISR
andi r22, et, 0b100000000
beq r22, zero, END_ISR
/* Если бит 0b100000000 регистра et не равен 1, то */ выходим из обработчика прерываний*/
call JTAG_UART_ISR/* Вызов обработчика прерываний от JTAG UART*/
END_ISR:
ldw et, 0(sp) /* Восстанавливаем значения регистров из стека */
ldw ea, 4(sp)
ldw ra, 8(sp)
ldw r22, 12(sp)
addi sp, sp, 16
eret
.end
/timer.s
.equ TIMER_BASE, 0x10002000
.equ TIMER_START_VALUE, 25000000
.extern symbol
.text
/*******************************************************************************
* Программа инициализации интервального таймера
*/
```

```
.global init_timer
init_timer:
subi sp, sp, 12 /*сохраняем регистры в стеке*/
stw ra, 8(sp)
stw r16, 4(sp)
stw r17, 0(sp)
movia r16, TIMER_BASE /*базовый адрес интервального таймера в r16*/
movia r17, TIMER_START_VALUE /*начальное значение счетчика*/
sthio r17, 8(r16) /*запись в младшую половину счетчика*/
srli r17, r17, 16 /*сдвигаем, чтобы получить старшее полуслово начального значения*/
sthio r17, 12(r16) /*запись в старшую половину счетчика*/
movi r17, 0b111 /*формируем команду инициализации (START + CONT + ITO)*/
sthio r17, 4(r16) /*записываем команду в регистр управления*/
ldw r16, 4(sp)
ldw r17, 0(sp)
addi sp, sp, 12
ret /*возврат из подпрограмы*/
/*****************************************************************************
* Обработчик прерываний от таймера
* Прерывания происходят раз в 500 миллисекунд, последний считанный символ
* выводится в JTAG UART
*/
.global TIMER_ISR
TIMER_ISR:
subi sp, sp, 25 /*сохраняем регистры в стеке*/
stw ra, 16(sp)
stw r2, 12(sp)
stw r14, 8(sp)
stw r16, 4(sp)
stw r17, 0(sp)
stw r18, 20(sp)
movia r16, TIMER_BASE
sthio zero, 0(r16) /*сбрасываем бит "time out" регистра status*/
movia r17, symbol
ldb r4, 0(r17) /*считываем последний введенный символ из памяти*/
movia r23, 0x0a
beq r4, r23, skip_met
0:
call putchar_uart /*отправляем символ в UART*/
bne r2, r0, 0b /*если буфер отправки переполнен, ждем*/
skip_met:
ldw ra, 16(sp) /*возвращаем регистры из стека*/
ldw r2, 12(sp)
ldw r14, 8(sp)
ldw r16, 4(sp)
ldw r17, 0(sp)
ldw r18, 20(sp)
addi sp, sp, 25
ret /*возврат из обработчика*/
/jtag_uart_isr.s
.equ EOF, -1 /*Сообщение об ошибке, гарантированно не совпадает с кодами символов*/
/*****************************************************************************
* Процедура обработки прерываний от JTAG UART
*****************************************************************************/
.global JTAG_UART_ISR
JTAG_UART_ISR:
subi sp, sp, 8 /*Сохраняем регистры в стеке */
stw ra, 4(sp)
stw r4, 0(sp)
rdctl r4, ienable /*если произошло прерывание от JTAG UART*/
ori r4, r4, 0b1 /*разрешаем прерывания от таймера IRQ0*/
wrctl ienable, r4
movi r5, EOF /*сохраняем в r5 код EOF*/
```

```
0:
call getchar_uart /*считываем байт из UART*/
beq r2, r5, exit /*если символов в буфере нет (EOF), выходим*/
mov r4, r2 /*считанный байт заносим в регистр-параметр*/
call hex_display /*выводим на hex-индикаторы*/
call led_display /*выводим на светодиоды*/
movia r2, symbol
stb r4, 0(r2)
br 0b /*ожидаем ввода нового символа*/
exit:
ldw ra, 4(sp) /*возвращаем регистры из стека*/
ldw r4, 0(sp)
addi sp, sp, 8
ret /*возврат из обработчика прерывания*/
.data
.global symbol
symbol:
.byte 0
.end
/string_io.s
.extern putchar_uart /*запись в UART*/
.extern getchar_uart /*чтение из UART*/
/*************************** Секция кода *********************************/
.text
/*****************************************************************************
* Функция вывода в UART нуль-терминированной строки
* Соответствует NIOS 2 ABI
* Сигнатура для C: void puts(char* string);
* Параметр:
* В r4 - адрес строки, оканчивающейся '\0'
*****************************************************************************/
.global puts
puts:
subi sp, sp, 12 /*создаем стековый кадр*/
stw ra, 8(sp)
stw r4, 4(sp)
stw r16, 0(sp)
mov r16, r4 /*сохраняем базовый адрес строки в r16*/
br 1f /*сразу переходим к проверке условия*/
0:
call putchar_uart /*выводим текущий символ в UART*/
bne r2, r0, 0b /*если буфер переполнен, пытаемся повторить*/
addi r16, r16, 1 /*переходим к рассмотрению следующего байта*/
1:
ldb r4, (r16) /*если мы дошли до нулевого байта*/
bne r4, r0, 0b /*то выходим из цикла*/
ldw ra, 8(sp) /*восстанавливаем регистры из стека*/
ldw r4, 4(sp)
ldw r16, 0(sp)
addi sp, sp, 12
ret /*выходим из подпрограммы*/
/*****************************************************************************
* Функция ввода из UART строки символов до перехода но новую строку '\n' (LF)
* Нулевой байт в конец строки добавляется автоматически
* Соответствует NIOS 2 ABI
```

```
* Сигнатура для C: void gets(char* string);
* Параметр:
* В r4 - указатель на область памяти, в которой сохраняется строка
*******************************************************************************/
.global gets
gets:
subi sp, sp, 20 /*создаем стековый кадр*/
stw ra, 16(sp)
stw r4, 12(sp)
stw r16, 8(sp)
stw r17, 4(sp)
stw r18, 0(sp)
mov r16, r4 /*сохраняем адрес начала памяти для строки в r16*/
movi r17, EOF /*сохраняем в r17 код EOF*/
movi r18, '\n' /*сохраняем в r18 код перехода на новую строку '\n'*/
0:
call getchar_uart /*считываем байт из UART*/
beq r2, r17, 0b /*если символов в буфере нет (EOF), повторяем*/
mov r4, r2
call hex_display /*эхопечать кода принятого символа на hex-индикаторах*/
call led_display /*эхопечать кода принятого символа на светодиодах*/
beq r2, r18, 1f /*если текущий считанный символ - это переход на новую строку, выходим*/
stb r2, (r16) /*сохраняем текущий символ в память*/
addi r16, r16, 1 /*увеличиваем адрес в строке*/
br 0b /*переход к считыванию следующего символа*/
1:
stb r0, (r16) /*дополняем строку нулевым байтом*/
ldw ra, 16(sp) /*восстанавливаем регистры из стека*/
ldw r4, 12(sp)
ldw r16, 8(sp)
ldw r17, 4(sp)
ldw r18, 0(sp)
addi sp, sp, 20
ret /*выходим из подпрограммы*/
```



Вывод из процессной системы в UART JTAG с использованием вложенного прерывания.

5.1. Экспериментально определили условие формирования прерывания текущей программы при выполнении вывода символьной информации в терминальное окно инструментального компьютера. Для этого в основной программе разрешили прерывания по записи от порта JTAG UART и выполнили запись в буфер FIFO, используя программу из пункта 4 части 1. Как мы видели ранее, выполнение этой программы приводит к переполнению буфера FIFO, предназначенного для записи. Следовательно, должно произойти прерывание программы. Обработчик прерывания должен вывести на экран LCD предупреждающее сообщение, проверить, освободился ли буфер FIFO, и, если да, вернуть управление основной программе. Таким образом, обработчик прерывания будет срабатывать как предохранитель, принудительно приостанавливая процесс дальнейшего заполнения символьной информацией буфера FIFO, позволяя тем самым в это время его освободить.

Вывод

В ходе выполнения лабораторной работы были успешно изучены и практически применены принципы работы интерфейса **JTAG UART** для организации двустороннего обмена данными между процессорной системой стенда и инструментальным компьютером. их настройки.