От инженеров для инженеров

FPGA-Systems.ru

MicroBlaze: работа с прерываниями

Автор: **MetallFly**

Рецензент: KeisN13

2019 г.

Оглавление

Аннотация

В статье описано подключение к процессорной системе контроллера прерываний и двух источников прерываний в Vivado 2018.3, а также настройка и обработка этих прерываний. Будут рассмотрены основные моменты настройки IP ядра, его подключение к процессору Microblaze и написание кода для конфигурации и обработки прерываний.

Введение

Аппаратные прерывания – это необходимый инструмент при разработке рабочей процессорной системы. Они позволяют выполнить обработку специального кода при асинхронно наступающем событии. В софт-процессорной системе по типу Microblaze необходимо дополнительно добавить «внешний» контроллер прерываний, который будет генерировать нужные векторы прерываний. Для знакомства с подключением контроллера прерываний и обработкой этих прерываний мы добавим два источника прерываний: UART-Lite и GPIO.

Целью статьи является описание основных настроек контроллера прерываний для процессорной системы Microblaze и пояснение этапов подключения.

При настройке и отладке этого проекта использовалась среда Vivado 2018.3 и плата ARTY S7.

Настройка в Vivado

Для работы с контроллером прерываний нам необходимо для начала создать проект и добавить в него IP ядро процессора. Если вы не знаете, как это сделать, ознакомьтесь с предыдущими статьями цикла [1].

После добавления ядра, запустим Run Block Automation. Мы должны увидеть следующее окно (рисунок 1):

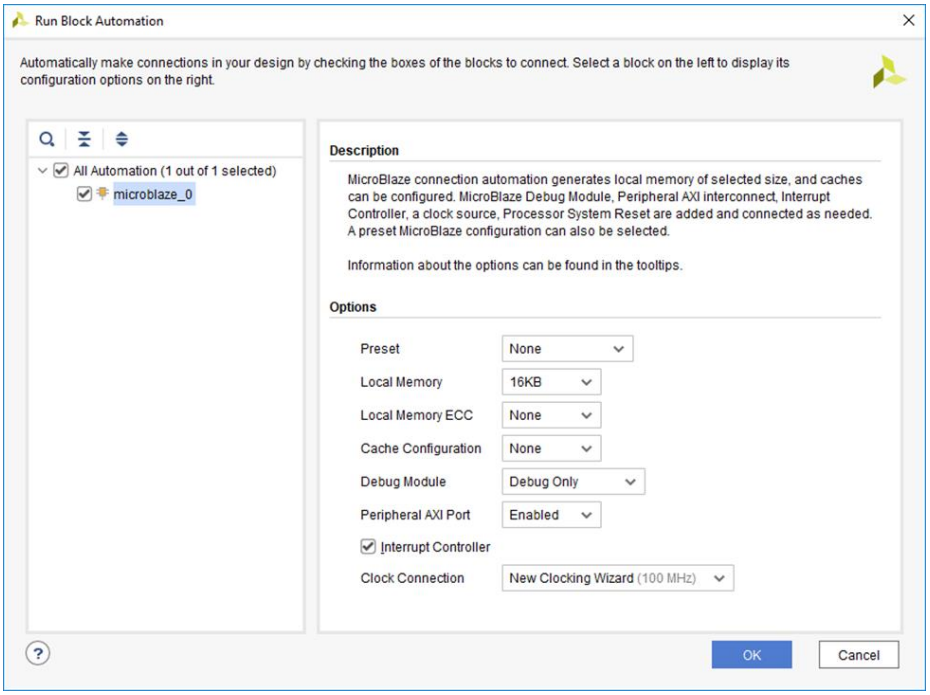

Рисунок 1 – Окно Run Block Automation

Установим Local Memory 16 KB, и поставим галочку напротив Interrupt Controller. Больше ничего менять не требуется и можно нажать OK. В рабочем поле Block Design должны появиться следующие модули (рисунок 2):

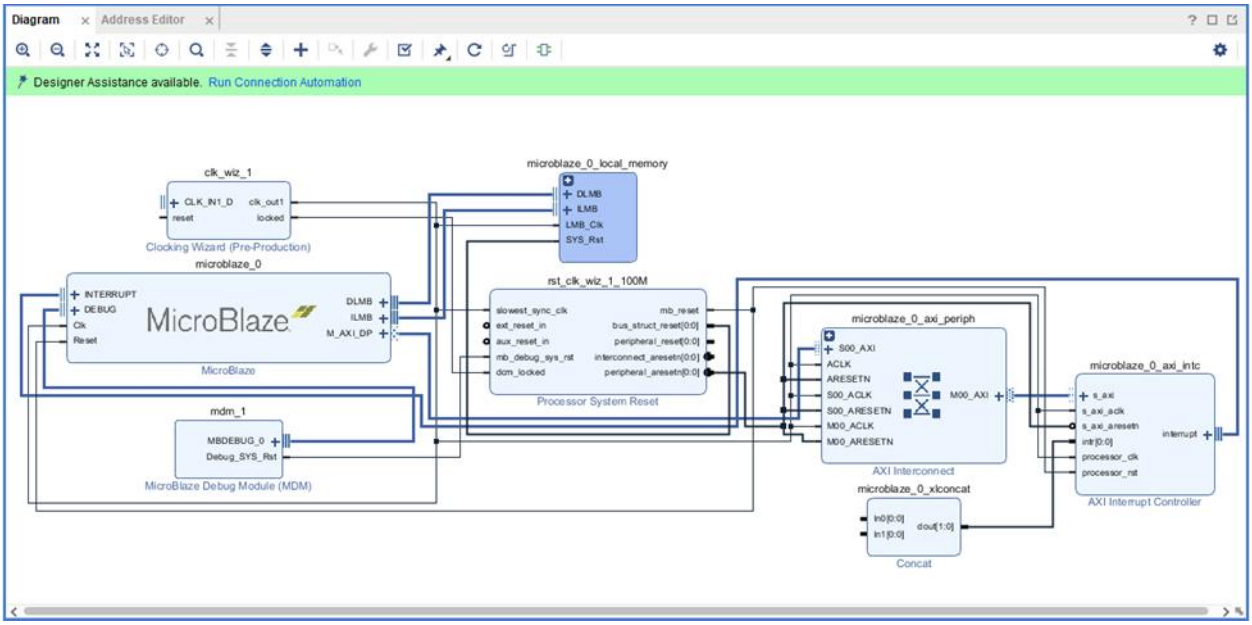

Рисунок 2 – Окно Block Design после запуска автоматизации

Как мы можем заметить, доступ к регистрам контроллера прерываний осуществляется по шине AXI. Для подключения нескольких источников прерываний используется модуль Concat, объединяющий отдельные сигналы в шину.

Если ваша процессорная подсистема уже была настроена, и вам только понадобилось добавить прерывания в свой проект, то найти контроллер можно в каталоге IP ядер (рисунок 3).

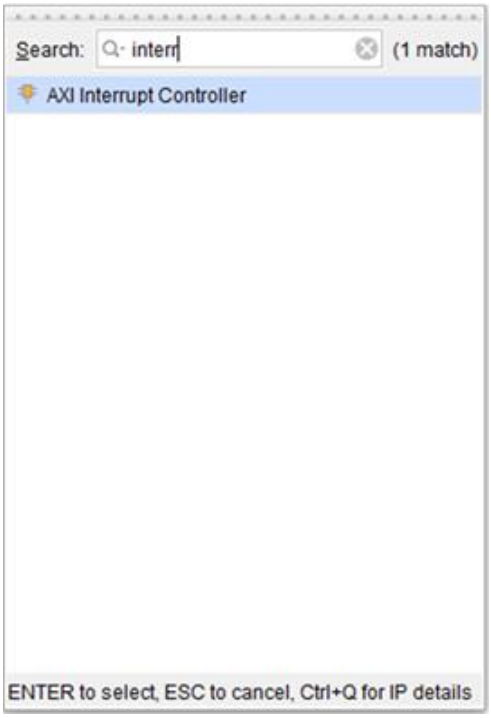

Рисунок 3 – Поиск контроллера прерываний

Добавляем его в поле Block Design и дважды кликаем по нему мышкой. Должно открыться окно его настройки, представленное на рисунке 4.

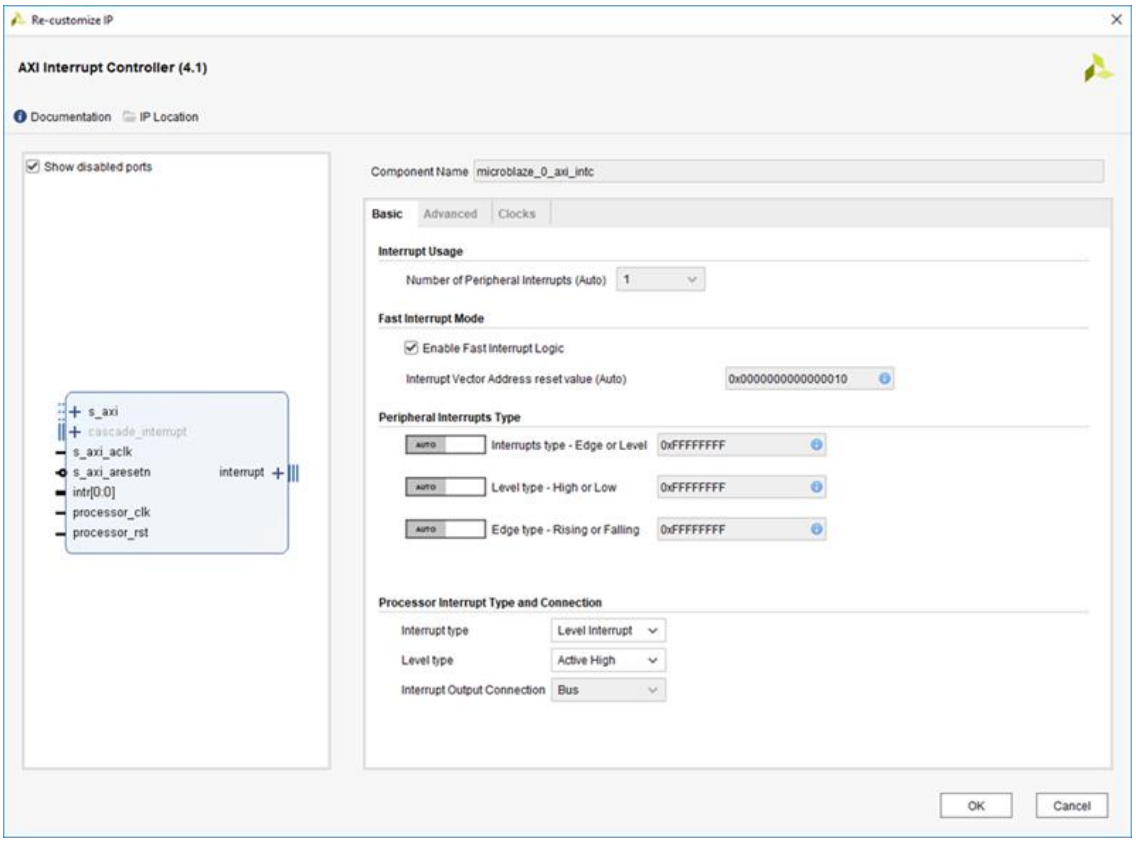

Рисунок 4 – Окно настройки контроллера прерываний

Большинство опций оставим по умолчанию, так как они сами будут настроены в процессе запуска автоматизации. Установим только галочку напротив Fast Interrupt mode, что в свою очередь автоматически переключит тип подключения к процессору с отдельного сигнала на шину. Это позволит ускорить переход процессором к обработчику прерываний в результате передачи вектора непосредственно по этой шине. Прерывание подтверждается сигналом processor_ack [2]. Нажимаем OK. Не забудем также настроить Clock Wizard.

Теперь добавим периферию, которая будет источником прерываний. Пусть это будут UART Lite и GPIO, к которому подключены кнопки и светодиоды (я уверен, на вашей отладочной плате они точно будут). В UART поменяем только Baud Rate на 115200. Для GPIO же установим следующие настройки (рисунок 5):

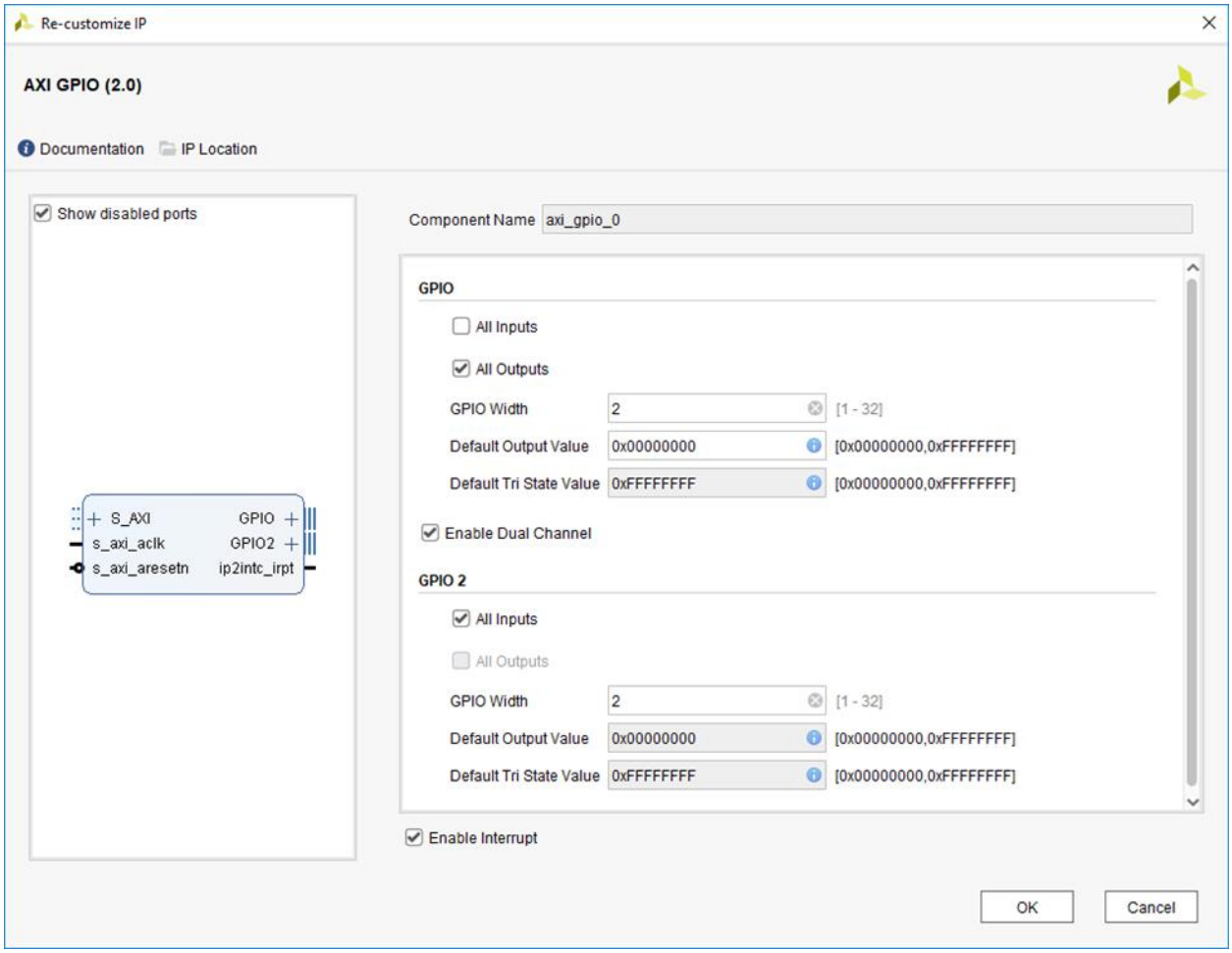

Рисунок 5 – Настройка UART Lite

Здесь мы разрешили прерывания, добавили два канала и установили их ширину и направление. На второй канал мы подключим наши кнопки. Это нужно для того, чтоб не детектировались изменения сигналов на выходных портах, и связано это с устройством

прерываний в AXI GPIO [3] (в первый раз я сам столкнулся с таким парадоксом, но, если я ошибаюсь, пусть меня поправят). Нажимаем OK.

Теперь нам необходимо подключить прерывания к контроллеру прерываний, а также периферию к шине AXI. Для этого воспользуемся предоставленный модулем конкатенации, чьей ширины сейчас как раз хватает для подключения прерываний (рисунок 6).

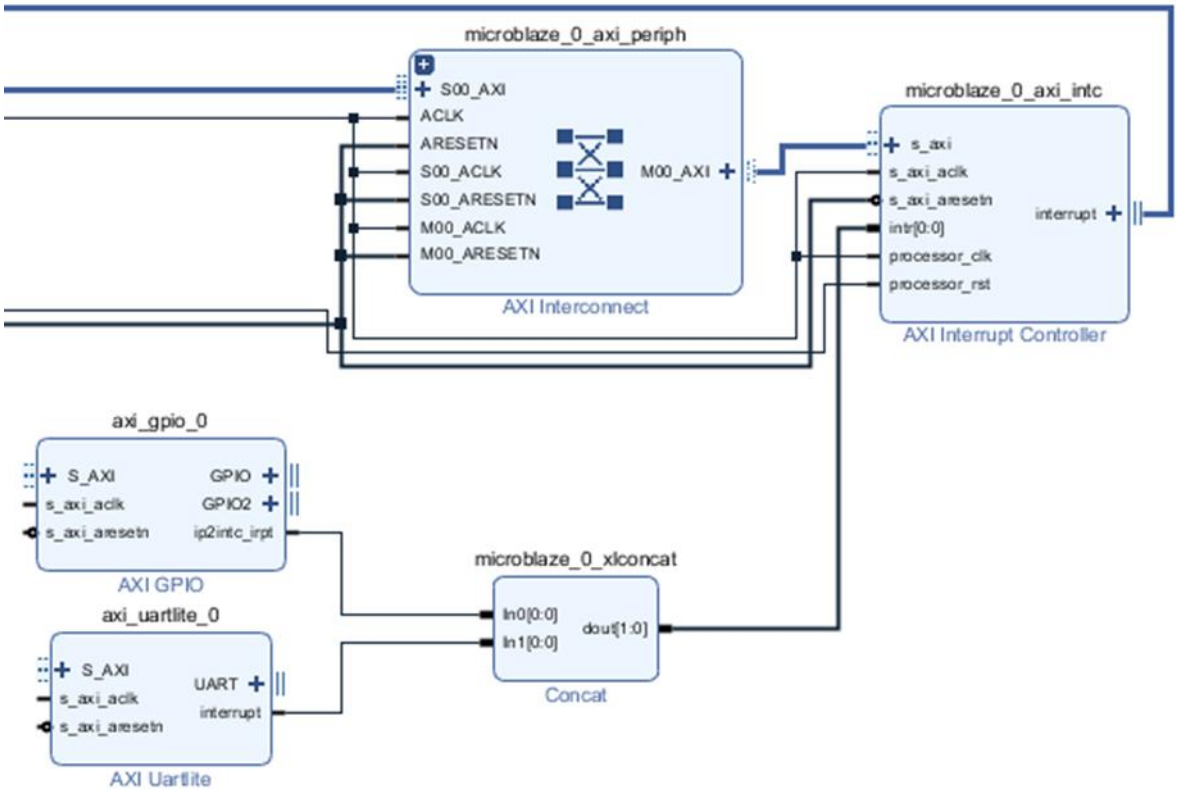

Рисунок 6 – Подключение источников прерываний

Приоритет прерываний определяется позицией подключаемого источника. Младший бит будет обладать высшим приоритетом.

Теперь запустим автоматизацию подключения периферии и выведем наружу сигналы UART, GPIO и CLK. Также не забывайте не оставлять в воздухе сигналы reset и подтянуть их с помощью Constant к “1”. После нее запустим последовательно Regenerate Layout и Validate Design. Поле Block Design будет выглядеть следующим образом (рисунок 7):

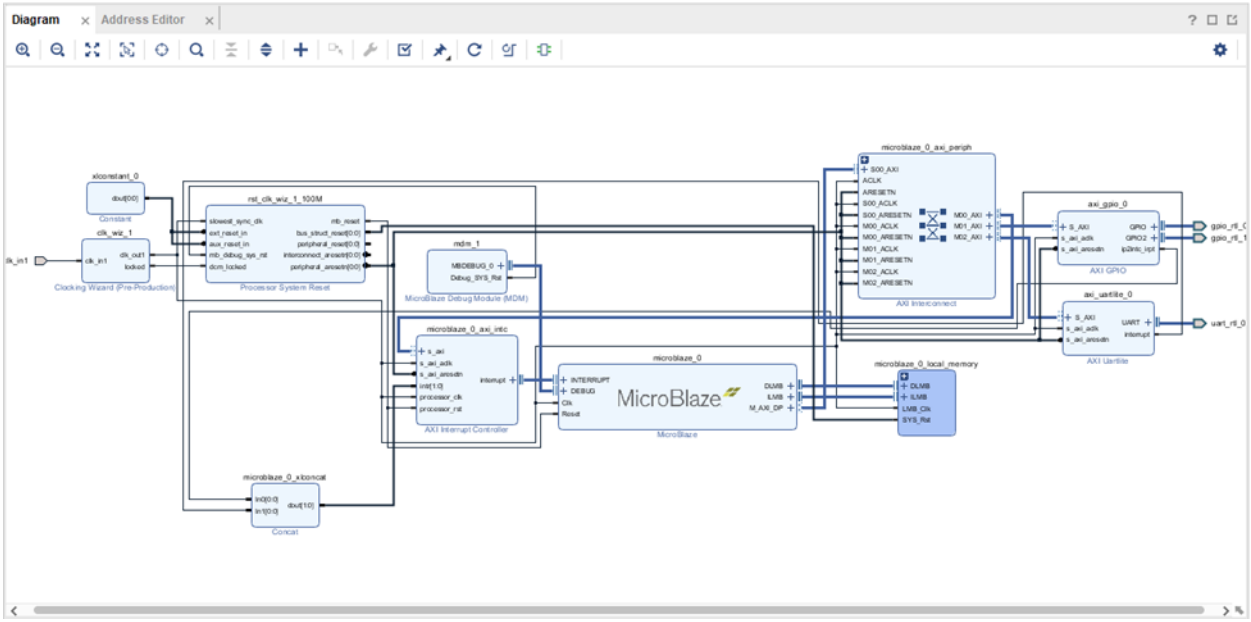

Рисунок 7 – Поле Block Design

Можно снова открыть настройки контроллера прерываний и посмотреть, как они изменились (Подсказка: прерывания от портов теперь генерируются по фронту сигнала). Теперь самостоятельно запустим синтез, подключим источник тактирования, кнопки и светодиоды и сгенерируем Bitstream. После выполним Export Hardware и запустим SDK.

Написание С кода в SDK

Теперь создадим пустой проект. Если кто не помнит: File->New->Application Project. После добавим Source файл и в нем создадим функцию main со следующим кодом:

```
#include <string.h>
#include "xgpio.h"
#include "xuartlite.h"
#include "xintc.h"

#define	LENGHT	32

#define	BTN1	0x01
#define	BTN2	0x02

#define	LED1	0x01
#define	LED2	0x02


XGpio Gpio;
XUartLite Uart;
XIntc Intc;
```

```
u8 GpioIntrFlag = 0;
u8 UartSendIntrFlag = 0;
u8 UartRecvIntrFlag = 0;

u8 SendBuffer[LENGHT];
u8 ReceiveBuffer[LENGHT];

u8 ReceiveBufferPtr = 0;

void Initialization();

void GpioHandler(void *CallBackRef);

void UartSendHandler(void *CallBackRef, unsigned int EventData);

void UartRecvHandler(void *CallBackRef, unsigned int EventData);

int main()
{

	u8 sendStrLen = 0;

	Initialization();

	XUartLite_Recv(&Uart, (ReceiveBuffer + ReceiveBufferPtr), 1);

	while(1)
	{

		if(UartRecvIntrFlag)
		{
			UartRecvIntrFlag = 0;

			if(strcmp("LED On", (char*)ReceiveBuffer) == 0)
			{
				strcpy((char*)SendBuffer, "LED is On\r");
				XGpio_DiscreteSet(&Gpio, XGPIO_IR_CH1_MASK, LED2);
			}
			else if(strcmp("LED Off", (char*)ReceiveBuffer) == 0)
			{
				strcpy((char*)SendBuffer, "LED is Off\r");
				XGpio_DiscreteClear(&Gpio, XGPIO_IR_CH1_MASK, LED2);
			}
			else
			{
				strcpy((char*)SendBuffer, "Unknown command: ");
				strcat((char*)SendBuffer, (char*)ReceiveBuffer);
				strcat((char*)SendBuffer, "\r");
			}

			sendStrLen = strlen((char*)SendBuffer);

			XUartLite_Send(&Uart, SendBuffer, sendStrLen);
```

```
		}

		switch(GpioIntrFlag)
		{
			case BTN1:
			{
				GpioIntrFlag = 0;

				strcpy((char*)SendBuffer, "Button 1 is pressed\r");

				sendStrLen = strlen((char*)SendBuffer);

				XUartLite_Send(&Uart, SendBuffer, sendStrLen);

				break;
			}

			case BTN2:
			{
				GpioIntrFlag = 0;

				XGpio_DiscreteWrite(&Gpio, XGPIO_IR_CH1_MASK,
XGpio_DiscreteRead(&Gpio, XGPIO_IR_CH1_MASK) ^ LED1);

				break;
			}

		}

	}

	return 0;
}

void Initialization()
{

	XGpio_Initialize(&Gpio, XPAR_GPIO_0_DEVICE_ID);

	XUartLite_Initialize(&Uart, XPAR_UARTLITE_0_DEVICE_ID);

	XIntc_Initialize(&Intc, XPAR_INTC_0_DEVICE_ID);

	XIntc_Connect(&Intc, XPAR_INTC_0_GPIO_0_VEC_ID, (Xil_ExceptionHandler)GpioHandler,
&Gpio);

	XIntc_Connect(&Intc, XPAR_INTC_0_UARTLITE_0_VEC_ID,
(XInterruptHandler)XUartLite_InterruptHandler, &Uart);

	XIntc_Enable(&Intc, XPAR_INTC_0_GPIO_0_VEC_ID);

	XIntc_Enable(&Intc, XPAR_INTC_0_UARTLITE_0_VEC_ID);
```

```
	XIntc_Start(&Intc, XIN_REAL_MODE);

	XGpio_InterruptEnable(&Gpio, XGPIO_IR_CH2_MASK);

	XGpio_InterruptGlobalEnable(&Gpio);

	XUartLite_SetSendHandler(&Uart, UartSendHandler, &Uart);

	XUartLite_SetRecvHandler(&Uart, UartRecvHandler, &Uart);

	XUartLite_EnableInterrupt(&Uart);

	Xil_ExceptionInit();

	Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
(Xil_ExceptionHandler)XIntc_InterruptHandler, &Intc);

	Xil_ExceptionEnable();

}

void GpioHandler(void *CallbackRef)
{

	XGpio *GpioPtr = (XGpio *)CallbackRef;

	GpioIntrFlag = XGpio_DiscreteRead(GpioPtr, XGPIO_IR_CH2_MASK);

	XGpio_InterruptClear(GpioPtr, XGPIO_IR_CH2_MASK);

}

void UartSendHandler(void *CallBackRef, unsigned int EventData)
{
	UartSendIntrFlag = 1;
}

void UartRecvHandler(void *CallBackRef, unsigned int EventData)
{

	if(ReceiveBuffer[ReceiveBufferPtr] == '\0')
	{
		ReceiveBufferPtr = 0;
		UartRecvIntrFlag = 1;
	}
	else if(++ReceiveBufferPtr == 32)
	{
		ReceiveBufferPtr = 0;
	}

	XUartLite_Recv(&Uart, (ReceiveBuffer + ReceiveBufferPtr), 1);
}
```

Теперь разберем по порядку некоторые строчки нашего кода. Здесь мы создали программную модель портов, контроллера прерываний и UART.

```
XGpio Gpio;
XUartLite Uart;
XIntc Intc;
```

Затем создали буферы приемника и передатчика, а также флаги для индикации произошедшего прерывания.

```
u8 GpioIntrFlag = 0;
u8 UartSendIntrFlag = 0;
u8 UartRecvIntrFlag = 0;

u8 SendBuffer[LENGHT];
u8 ReceiveBuffer[LENGHT];

u8 ReceiveBufferPtr = 0;
```

Создали прототипы функции инициализации и функций обработчиков прерываний, на которые будет ссылаться наша программа после перехода по векторам прерываний.

```
void Initialization();

void GpioHandler(void *CallBackRef);

void UartSendHandler(void *CallBackRef, unsigned int EventData);

void UartRecvHandler(void *CallBackRef, unsigned int EventData);
```

Вызов следующих функций происходит в функции Initialization() и, как следует из названия, инициализирует порты, контроллер прерываний и UART.

```
XGpio_Initialize(&Gpio, XPAR_GPIO_0_DEVICE_ID);

XUartLite_Initialize(&Uart, XPAR_UARTLITE_0_DEVICE_ID);

XIntc_Initialize(&Intc, XPAR_INTC_0_DEVICE_ID);
```

Следующими функциями мы подключаем в таблице векторов прерываний наши обработчики прерываний, причем GpioHandler мы определяем сами, в то время как InterruptHandler является библиотечной функцией, что, однако, не мешает нам использовать свой

обработчик вместо нее. После этого разрешаем эти прерывания в контроллере прерываний и запускаем его.

```
XIntc_Connect(&Intc, XPAR_INTC_0_GPIO_0_VEC_ID, (Xil_ExceptionHandler)GpioHandler,
&Gpio);

XIntc_Connect(&Intc, XPAR_INTC_0_UARTLITE_0_VEC_ID,
(XInterruptHandler)XUartLite_InterruptHandler, &Uart);

XIntc_Enable(&Intc, XPAR_INTC_0_GPIO_0_VEC_ID);

XIntc_Enable(&Intc, XPAR_INTC_0_UARTLITE_0_VEC_ID);

XIntc_Start(&Intc, XIN_REAL_MODE);
```

Далее мы разрешаем прерывания в модулях портов и UART и также подключаем отдельные обработчики прерываний для окончания приема и окончания передачи.

```
XGpio_InterruptEnable(&Gpio, XGPIO_IR_CH2_MASK);

XGpio_InterruptGlobalEnable(&Gpio);

XUartLite_SetSendHandler(&Uart, UartSendHandler, &Uart);

XUartLite_SetRecvHandler(&Uart, UartRecvHandler, &Uart);

XUartLite_EnableInterrupt(&Uart);
```

В самом конце мы разрешаем прерывания и подключаем таблицу обработки исключений уже в самом процессоре.

```
Xil_ExceptionInit();

Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
(Xil_ExceptionHandler)XIntc_InterruptHandler, &Intc);

Xil_ExceptionEnable();
```

В функции GpioHandler мы считываем состояние второго порта GPIO и запоминаем его состояние во флаге прерываний. В конце очищаем флаг прерывания.

```
XGpio *GpioPtr = (XGpio *)CallbackRef;

GpioIntrFlag = XGpio_DiscreteRead(GpioPtr, XGPIO_IR_CH2_MASK);

XGpio_InterruptClear(GpioPtr, XGPIO_IR_CH2_MASK);
```

В UartRecvHandler обрабатываем окончание приема данных по UART. Данная функция будет вызвана после приема последнего байта в буфер приемника, чей размер мы указываем во время инициализации приема. Если мы приняли окончание строки, то устанавливаем флаг UartRecvIntrFlag и устанавливаем указатель буфера ReceiveBufferPtr на его начало. Также проверяем, заполнился ли буфер приемника, и, если заполнился, переводим указатель на его начало. Затем инициализируем прием одного байта по UART по адресу начала буфера приемника со смещением ReceiveBufferPtr.

```
if(ReceiveBuffer[ReceiveBufferPtr] == '\0')
{
	ReceiveBufferPtr = 0;
	UartRecvIntrFlag = 1;
}
else if(++ReceiveBufferPtr == 32)
{
	ReceiveBufferPtr = 0;
}

XUartLite_Recv(&Uart, (ReceiveBuffer + ReceiveBufferPtr), 1);
```

Обработчик прерывания по окончанию передачи мы не используем, потому его код ограничивается только установкой флага прерывания UartSendIntrFlag.

Теперь рассмотрим код программы, ограничивающийся бесконечным циклом while. В первом условии проверяется флаг прерывания по окончанию приема, устанавливающийся при получении символа конца строки. Если флаг установлен, то он сразу же сбрасывается. Затем принятая строка сравнивается с шаблонными командами "LED On" и "LED Off", и если одна из них совпадает, то включается или выключается диод под номером 2 и буфер передатчика заполняется ответом "LED is On\r" или "LED is Off\r" соответственно. Если команда не совпадает, то буфер заполняется ответом "Unknown command: " и содержимым буфера приемника. Затем с помощью функции strlen мы узнаем длину строки в буфере передатчика и инициализируем передачу этой строки через UART.

```
if(UartRecvIntrFlag)
{
	UartRecvIntrFlag = 0;

	if(strcmp("LED On", (char*)ReceiveBuffer) == 0)
	{
		strcpy((char*)SendBuffer, "LED is On\r");
		XGpio_DiscreteSet(&Gpio, XGPIO_IR_CH1_MASK, LED2);
	}
	else if(strcmp("LED Off", (char*)ReceiveBuffer) == 0)
	{
		strcpy((char*)SendBuffer, "LED is Off\r");
		XGpio_DiscreteClear(&Gpio, XGPIO_IR_CH1_MASK, LED2);
	}
	else
	{
		strcpy((char*)SendBuffer, "Unknown command: ");
		strcat((char*)SendBuffer, (char*)ReceiveBuffer);
		strcat((char*)SendBuffer, "\r");
	}

	sendStrLen = strlen((char*)SendBuffer);

	XUartLite_Send(&Uart, SendBuffer, sendStrLen);
}
```

В операторе switch проверяется состояние флага прерывания от портов. Если источником прерывания выступает первая кнопка, то буфер передатчика заполняется строкой "Button 1 is pressed\r" и инициализируется передача по UART этой строки. Если источником прерывания выступает вторая кнопка, то изменяется состояние светодиода номер 2.

```
switch(GpioIntrFlag)
{
	case BTN1:
	{
		GpioIntrFlag = 0;

		strcpy((char*)SendBuffer, "Button 1 is pressed\r");

		sendStrLen = strlen((char*)SendBuffer);

		XUartLite_Send(&Uart, SendBuffer, sendStrLen);

		break;
	}

	case BTN2:
	{
		GpioIntrFlag = 0;
```

```
                XGpio_DiscreteWrite(&Gpio, XGPIO_IR_CH1_MASK, XGpio_DiscreteRead(&Gpio,
XGPIO_IR_CH1_MASK) ^ LED1);

                break;
        }
}
```

Стоит отметить, что мы не пытались бороться с дребезгом контакта на кнопке, потому бывает, что прерывания возникают по нескольку раз подряд. Так как у нас еще достаточно логики, я предложу решить проблему аппаратно. Добавим модуль со следующим кодом для входных портов GPIO:

```
module debounce #(
   //In MHz
   parameter freq = 100
)(

   input             clk,
   input             rst_n,
   input             btn_i,
   output             btn_o
   );

   reg     [31 : 0]    cnt;

   assign  btn_o       = btn_i & cnt >= freq * 10_000;

   always @ (posedge clk)
   begin
      if(!rst_n)
      begin
         cnt        <= 32'b0;
      end
      else
      begin
         if(btn_i & cnt < freq * 10_000)
         begin
            cnt     <= cnt + 1'b1;
         end
         if(!btn_i)
         begin
            cnt     <= 32'b0;
         end
      end
   end

endmodule
```

Этот модуль добавляет задержку срабатывания фронта сигнала перед входом порта на 10 мс. Этого как раз должно хватить, чтобы все переходные процессы закончились, и в то же время 10 мс не будут заметны человеческому глазу.

При работе с терминалом также не забывайте добавлять символ окончания строки, если ваш терминал этого не делает автоматически.

Выводы

Мы рассмотрели с вами подключение прерываний периферии к контроллеру прерываний, настроили их программно, написали обработчики. В результате наша программа по командам зажигает и гасит светодиод, уведомляет нас о нажатии кнопки, а также переключает состояние другого светодиода по нажатию второй кнопки. Также мы рассмотрели один из способов борьбы с дребезгом контакта с помощью программируемой логики.

Список литературы

[1]. Разработка процессорной системы на базе софт-процессора MicroBlaze в среде Xilinx Vivado IDE/HLx. Часть 1.

[2]. PG099 - AXI Interrupt Controller

[3]. PG144 - AXI GPIO v2.0