ПЛИС фирмы Gowin Semiconductor: номенклатура, характеристики, разработка приложений.

Часть 4. Программные процессорные IP-ядра ПЛИС GOWIN, разработка программной и аппаратной части проектов ПЛИС

Павел Редькин

В предлагаемой статье рассмотрены порядок и особенности создания проектов ПЛИС GOWIN, содержащих программные процессорные IP-ядра в среде GOWIN FPGA Designer и встроенного программного обеспечения (ПО) для этих ядер в среде IDE ARM Keil MDK. Статья предназначена для разработчиков электронной аппаратуры на ПЛИС и студентов специальностей, связанных с цифровой электроникой.

Техническое задание и этапы разработки

В предыдущей части [1] цикла статей о ПЛИС GOWIN было описано создание простого проекта ПЛИС для отладочной платы TangNano 9K с ПЛИС GW1NR-9. Созданный проект однократно выводит некие константные данные на подключённый к ПЛИС символьный ЖКИ, совместимый со стандартом HD44780, а потом постоянно опрашивает линию GPIO ПЛИС, сконфигурированную в качестве входа, и в зависимости от значения внешнего уровня на ней выводит на ЖКИ одну из двух заданных последовательностей символов.

Теперь добавим к созданному проекту процессорное IP-ядро, что позволит значительно увеличить гибкость проекта. Процессорное IP-ядро будет выполнять встроенное программное обеспечение (ПО), которое мы параллельно разработаем в другой среде разработки, а также обслуживать свои периферийные интерфейсы, которые мы также добавим к нашему проекту. В ПЛИС GW1NR-9 отсутствуют аппаратные процессорные ядра, поэтому наше IP-ядро будет чисто программным, то есть синтезированным на базе стандартных ячеек логической матрицы ПЛИС.

Для начала сформулируем требования к функционалу программного процессорного IP-ядра, которое собираемся включить в проект.

1. Процессорное ядро должно иметь архитектуру ARM Cortex-M1, которой в IDE GOWIN FPGA Designer соответствует обозначение Gowin_EMPU_M1.
2. Процессорное ядро должно тактироваться от выходного синхросигнала системы PLL, уже имеющейся в нашем проекте ПЛИС.
3. Процессорное ядро должно иметь собственный отдельный отладочный интерфейс JTAG и поддерживать режим отладки через него.
4. Процессорное ядро должно обеспечивать поддержку обмена данными с внешними устройствами через свой периферийный интерфейс UART со скоростью обмена из стандартного ряда скоростей.
5. Процессорное ядро должно обеспечивать поддержку чтения и записи через свои периферийные линии GPIO. Количество таких линий в проекте зададим равным восьми (семь выходов, один вход). К шести из семи выходов на плате TangNano 9K подключены светодиоды с активным высоким уровнем выходного сигнала, к восьмому выходу ничего пока не подключено, а ко входу подключена пользовательская кнопка с активным низким уровнем входного сигнала.

Таблица 1. Количество системных ресурсов, необходимое для синтеза процессорного IP-ядра Cortex-M1 (Gowin_EMPU_M1) и периферии в ПЛИС GOWIN

Конфигурация программного процессорного IP-ядра Gowin_EMPU_M1 и его периферии в ПЛИС	Количество системных ресурсов ПЛИС, необходимое для синтеза данной конфигурации			
	Количество LUT	Количество регистров	Количество блоков BSRAM	Количество макросов DSP
Минимальная конфигурация программного IP-ядра Cortex-M1 без периферии	3034	1046	8	0
Заданная по умолчанию конфигурация программного IP-ядра Cortex-M1 без периферии	5237	2322	32	2
Заданная по умолчанию конфигурация программного IP-ядра Cortex-M1 со следующей периферией: GPIO, UART, таймер, охранный таймер (Watch Dog)	6961	3103	32	2
Заданная по умолчанию конфигурация программного IP-ядра Cortex-M1 со всей возможной периферией	21 029	12 929	48	2

Работа по добавлению программного процессорного IP-ядра в проект ПЛИС будет состоять из двух частей:

- разработка и компиляция программной части, то есть встроенного ПО процессорной системы, состоящей в ПЛИС из процессорного IP-ядра Cortex-M1 и его вышеперечисленной периферии (GPIO, UART);
- добавление к проекту ПЛИС процессорного IP-ядра Cortex-M1, его вышеперечисленной периферии и результата проведенной в предыдущем пункте компиляции встроенного ПО, затем итоговая компиляция всей этой сборки с последующей за-

грузкой выходного файла в конфигурационную память ПЛИС.

Первая часть задачи может быть выполнена в IDE ARM Keil MDK или альтернативно в IDE GOWIN MCU Designer. Выходные файлы у обеих IDE имеют один и тот же формат. Вторая часть задачи выполняется в GOWIN FPGA Designer.

Предварительно оценим достаточность аппаратных ресурсов используемой нами ПЛИС для выполнения поставленной задачи. В табл. 1, взятой из [2], указано количество системных ресурсов ПЛИС, необходимое для синтеза различных категорий конфигурации программного IP-ядра Cortex-M1 (Gowin_EMPU_M1) и его периферии. Как можно видеть из таблицы, ресурсы установленной на плате Tang Nano 9K ПЛИС GW1NR-9 по количеству LUT (8640) удовлетворяют категориям конфигурации IP-ядра включительно до «Заданная по умолчанию конфигурация программного IP-ядра Cortex-M1 со следующей периферией: GPIO, UART, таймер, охранный таймер (Watch Dog)». Оставшаяся часть ёмкости LUT ПЛИС GW1NR-9 нами будет использована для построения той части проекта ПЛИС, которая не входит в IP-ядро Cortex-M1.

Создадим на жёстком диске ПК каталог Proba_IP и скопируем туда созданный нами ранее в [1] проект ПЛИС fpga_project из каталога Proba. В этот проект ПЛИС в дальнейшем мы будем добавлять процессорное программное IP-ядро с периферией. В этом же каталоге Proba_IP создадим подкаталог FPGA_project_keil_soft. В этом подкаталоге мы будем создавать проект встроенного ПО для программного IP-ядра в среде IDE ARM Keil MDK. Архив готового проекта в виде файла Proba.zip доступен для загрузки с сайта журнала.

Разработка ПО для программного IP-ядра в IDE ARM Keil MDK

Программная среда IDE ARM Keil MDK от фирмы Keil пользуется заслуженной популярностью у программистов встраиваемых систем за простоту и удобство работы в ней, а также за гибкость применяемого инструментария. Начиная с версии V5.26 и выше. Среда IDE ARM Keil MDK поддерживает работу с аппаратными и программными процессорными IP-ядрами ПЛИС GOWIN.

Для образца при формировании структуры проекта в IDE ARM Keil

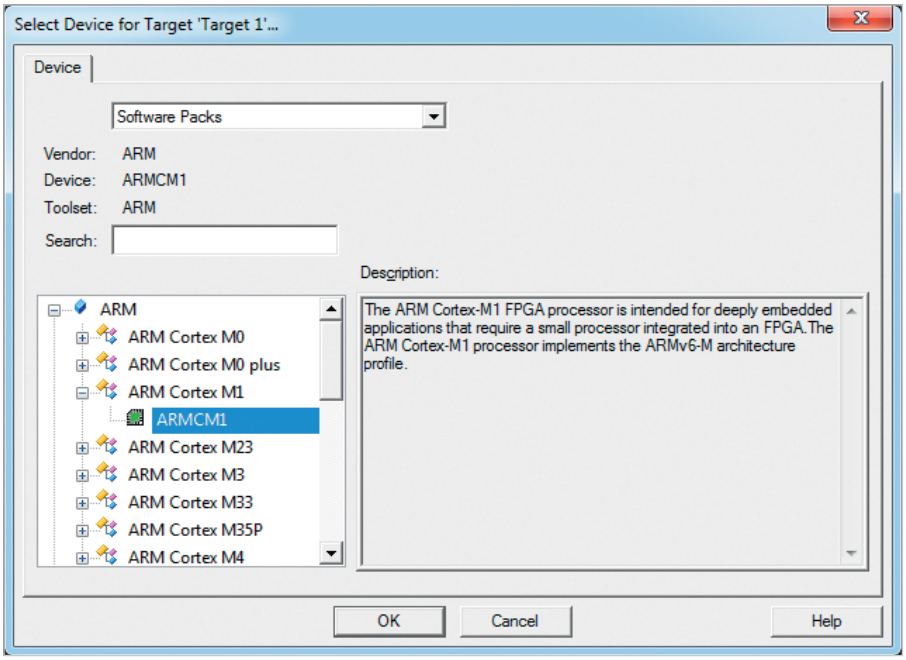

Рис. 1. Выбор платформы целевой системы

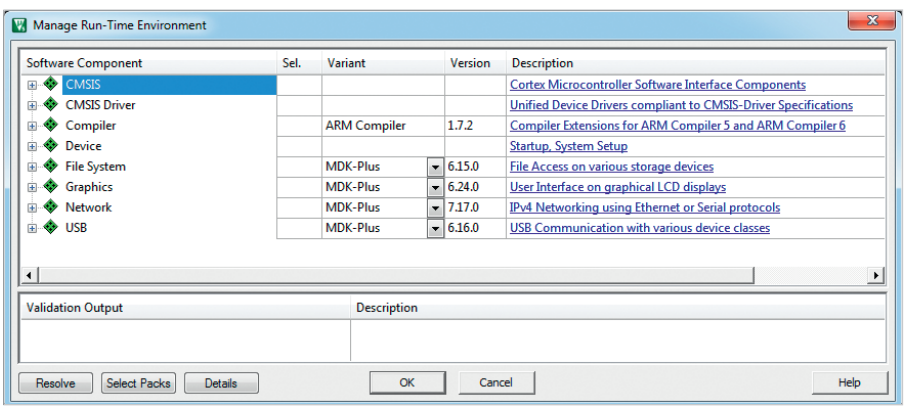

Рис. 2. Окно «Manage Run-Time Environment»

MDK рекомендуется взять модель Keil_RefDesign из архива «сервис пак» Gowin_EMPU_M1_Vx.x.x.zip, который можно загрузить с сайта GOWIN. Согласно рекомендуемой там структуре корневой каталог проекта должен содержать следующие подкаталоги:

- CORE (определения для ARM Cortex-M1 MCU);
- STARTUP (файлы Startup ядра MCU core);
- SYSTEM (определения периферийных регистров, инициализация системы, определения системных синхросигналов);
- PERIPHERAL (библиотеки драйверов функций периферии);
- USER (дизайн пользовательского приложения);
- PROJECT (служебные файлы проекта).

Перечисленные подкаталоги, кроме подкаталога PROJECT, следует заполнить файлами из соответствующих подкаталогов любого проекта, входящего в состав модели Keil _RefDesign. При этом файл пользовательского дизайна main.c в подкаталоге USER создаём, но пока оставляем пустым. Ниже в него будет добавлено содержимое дизайна нашего встроенного ПО.

Далее запускаем IDE ARM Keil MDK (автор использовал бесплатную версию v 5.37.0.0 с ограниченным размером встраиваемого кода) и создаём в нём новый проект. Для этого в главном меню выбираем: Project > New mVision Project. В открывшемся окне задаём имя файла проекта – FPGA_project_keil_soft.uvprojx и его расположение – подкаталог PROJECT в каталоге FPGA_project_keil_soft. Далее в автоматически открывшемся окне выбираем платформу целевой системы проекта, как показано на рис. 1: ARM Cortex M1 > ARMCM1. Далее автоматически открывается окно «Manage Run-Time Environment» (рис. 2), оставляем

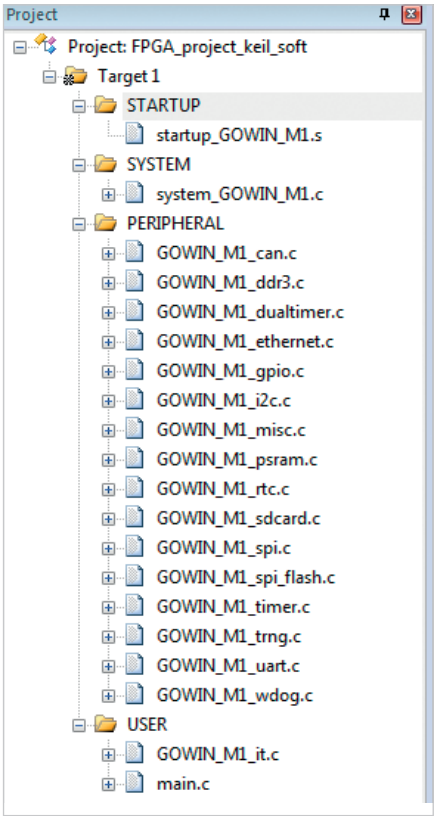

Рис. 3. Задание каталогов и файлов проекта

все его настройки заданными по умолчанию, кликаем на кнопке «OK». Проект создан.

Теперь задаём каталоги проекта, включаем туда ранее скопированные из архива Gowin_EMPU_M1_Vx.x.x.zip файлы, как показано на рис. 3.

Затем приступаем к настройке проекта, выбрав в главном меню: Project > Options for Target «Target 1». Это окно имеет несколько страниц (вкладок). На странице «Device» должно быть выбрано целевое устройство ARMCM1. Во вкладке задания параметров целевой системы «Target» задаём значения стартовых адресов и размеров различных типов памяти, доступной в целевой системе, как показано на рис. 4. Для процессорного ядра Gowin_EMPU_M1 внутренняя память программ или внешняя память программ является памятью ROM (IROM1 на рис. 4), а внутренняя память данных или внешняя память данных является памятью RAM (IRAM1 на рис. 4).

Необходимо задать начальные адреса и размеры ROM (внутренней памяти программ) и RAM (внутренняя память данных).

Начальный адрес и размер ROM задаются, исходя из следующих условий. В случае загрузки ПО из внешней SPI-Flash-памяти начальный адрес ROM задаётся равным 0x400. Размер ROM в этом случае необходимо установить согласно конфигурации размера встроенной памяти ITCM (память программ для процессорного IP-ядра в ПЛИС), задаваемого при разработке аппаратной части проекта ПЛИС, как будет показано ниже. В проектах из модели Keil_RefDesign задано значение 0x7C00, что соответствует размеру 31 744 байт.

В случае загрузки ПО из встроенной памяти ITCM начальный адрес ROM задаётся равным 0x0. Размер ROM в этом случае необходимо установить согласно конфигурации размера встроенной памяти ITCM, заданной при разработке аппаратной части проекта ПЛИС, как будет показано ниже. В проектах из модели Keil_RefDesign задано значение 0x8000, что соответствует 32 768 байт.

В нашем случае загрузка ПО будет производиться из встроенной памяти ITCM.

Начальный адрес и размер RAM задаются, исходя из следующих условий. Начальный адрес RAM всегда задаётся равным 0x20000000. Размер RAM необходимо установить согласно конфигурации размера встроенной памяти DTCM (память данных для процессорного IP-ядра в ПЛИС), задаваемого при разработке аппаратной части проекта ПЛИС, как будет показано ниже. В проектах из модели Keil_RefDesign задано значение 0x8000, что соответствует размеру 32 768 байт. Значение 0x4000 соответствует размеру 16 384 байт.

Заданные при разработке аппаратной части проекта ПЛИС размеры памяти ITCM и DTCM ограничены ресурсом памяти на кристалле ПЛИС.

Для ПЛИС GW1N-9/GW1NR-9/GW1N-9C/GW1NR-9C размер ITCM или размер DTCM может быть до 32 Кбайт. Если размер ITCM или размер DTCM в этих ПЛИС равен 32 Кбайт, то размер другого вида памяти не может превышать 16 Кбайт.

Для ПЛИС GW2AN-9X/GW2AN-1BX размер ITCM или размер DTCM может быть до 32 Кбайт. Если размер ITCM или размер DTCM в этих ПЛИС равен 32 Кбайт, то размер другого вида памяти не может превышать 16 Кбайт.

Для ПЛИС GW2A-18/GW2A-18C/GW2AR-18/GW2AR-18C/GW2ANR-18C размер ITCM или размер DTCM может быть до 64 Кбайт. Если размер ITCM или размер DTCM в этих ПЛИС равен 64 Кбайт, то размер другого вида памяти не может превышать 16 Кбайт.

Для ПЛИС GW2A-55/GW2A-55C/GW2AN-55C размер ITCM или размер DTCM может быть до 256 Кбайт. Если размер ITCM или размер DTCM в этих ПЛИС равен 256 Кбайт, то размер другого вида памяти не может превышать 16 Кбайт.

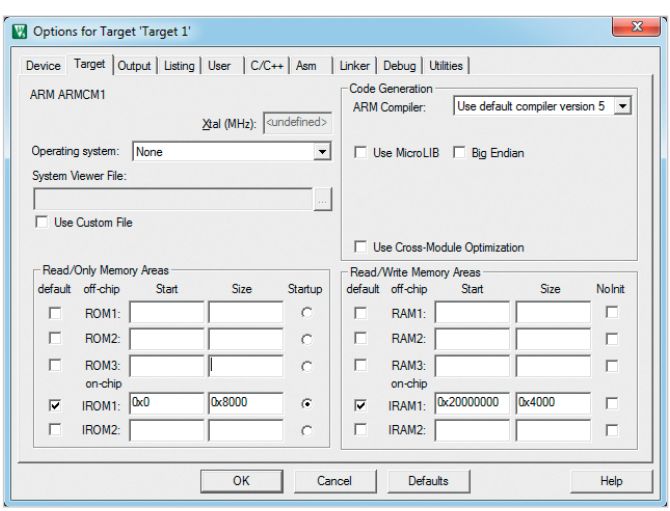

Рис. 4. Страница «Target» окна настройки проекта

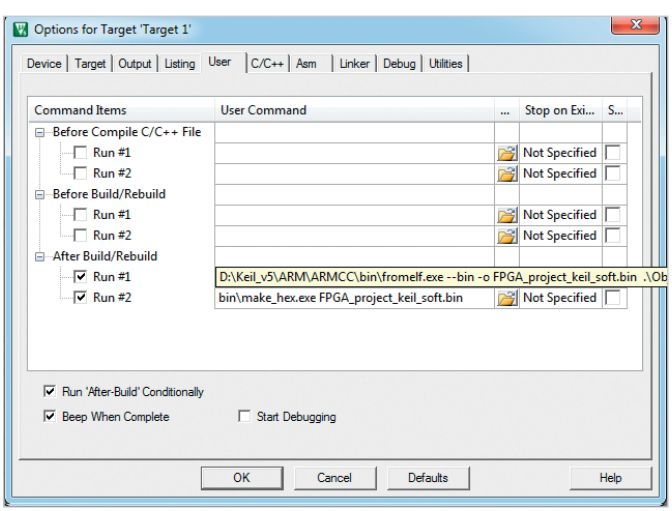

Рис. 5. Страница «User» окна настройки проекта

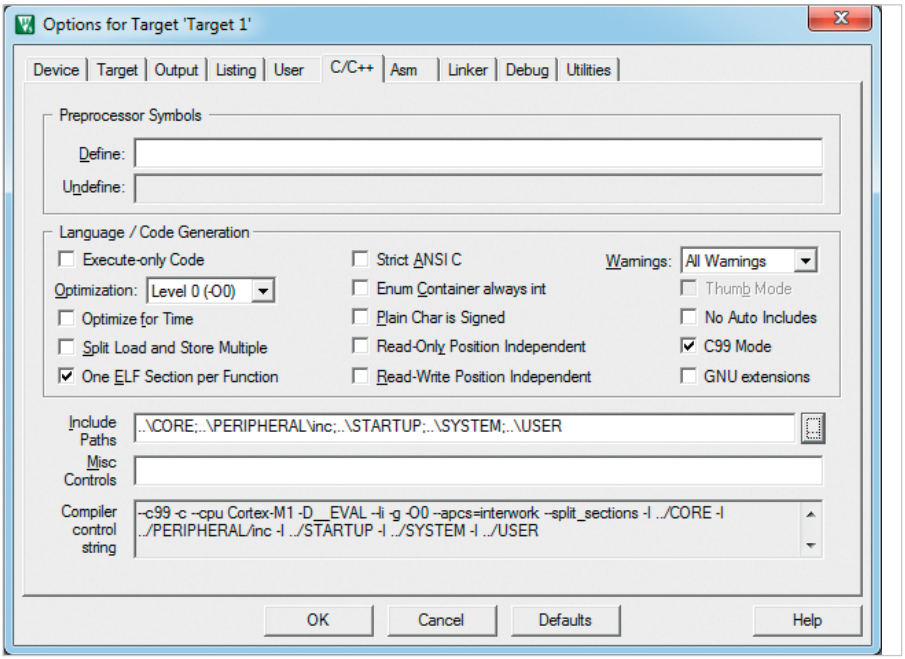

Рис. 6. Страница «C/C++» окна настройки проекта

Как показано на рис. 4, размер ITCM задаём равным 32 Кбайт, а размер DTCM – равным 16 Кбайт.

На страницах «Output», «Listing» окна настройки проекта все параметры оставляем заданными по умолчанию.

Настраиваем формат выходных файлов нашего проекта. По умолчанию IDE ARM Keil MDK генерирует выходной файл проекта с расширением *.axf. Однако для загрузки пользовательского ПО процессорного IP-ядра в ПЛИС GOWIN нам необходимо иметь комплект из четырёх загрузочных файлов в формате, распознаваемом ПЛИС: itcm0–itcm3. Эти файлы могут быть сформированы только из выходного файла проекта IDE ARM Keil MDK с расширением *.bin. Таким образом, нам необходимо выходной файл проекта с расширением *.axf сначала преобразовать в файл с расширением *.bin, а затем уже файл с расширением *.bin преобразовать в четыре загрузочных файла ПЛИС itcm0–itcm3. Для первого преобразования используется утилита fromelf.exe, которая имеется в составе IDE ARM Keil MDK. Для второго преобразования используется специальная утилита make_hex.exe из состава архива «сервис пак» Gowin_EMPU_M1_Vx.x.x.zip. Добавляем каталог bin, содержащий утилиту make_hex.exe с её библиотеками, в каталог PROJECT нашего проекта, где находится выходной файл с именем проекта и расширением *.bin.

На странице «User» окна настройки проекта добавляем две пользовательские команды, выполняемые после сборки проекта, следующего формата:

```
Run #1: fromelf.exe --bin -o
bin-file axf-file
Run #2: make_hex.exe bin-
file
```

В нашем проекте вместо формальных имен bin-file и axf-file указываем реальные имена наших файлов (FPGA_project_keil_soft.bin и FPGA_project_keil_soft.axf), также указываем реальные пути к утилитам fromelf.exe и make_hex.exe на диске ПК. У автора эти команды имеют следующий вид:

```
Run #1: D:\Keil_v5\ARM\
ARMCC\bin\fromelf.exe --bin -o
FPGA_project_keil_soft.bin.
\Objects\FPGA_project_keil_
soft.axf
Run #2: bin\make_hex.exe
FPGA_project_keil_soft.bin
```

Помечаем добавленные команды галочками, как показано на рис. 5.

Настраиваем параметры компиляции нашего проекта. На странице «C/C++» окна настройки проекта устанавливаем настройки, как показано на рис. 6. В частности, указываем пути к каталогам проекта в формате, показанном на рис. 7.

На страницах «Asm», «Linker» окна настройки проекта все параметры оставляем заданными по умолчанию.

На странице «Debug» окна настройки проекта устанавливаем настройки, как показано на рис. 8. В частности, указываем тип аппаратного отладчика-программатора, который в дальнейшем будем использовать для отладки

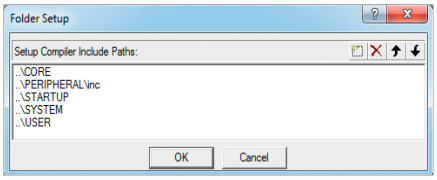

Рис. 7. Задание путей к каталогам проекта

кода в нашем программном процессорном IP-ядре, например, J-LINK/J-TRACE Cortex.

Теперь наполним наш проект конкретным содержанием в части пользовательского дизайна. В файле main.c из подкаталога USER напишем несложный пользовательский код, реализующий обмен данными через UART, опрос входа подключения кнопки и выдачу сигналов на выходы подключения светодиодов. В программе производится начальная инициализация процессорного ядра с помощью функции SystemInit(), начальная инициализация UART для скорости обмена 115 200 бит/с с помощью функции UartInit() и начальная инициализация линий GPIO с помощью функции GPIOInit(). Все перечисленные функции содержатся в скопированных из Keil_RefDesign файлах system_GOWIN_M1.c, GOWIN_M1_uart.c, GOWIN_M1_gpio.c, входящих в состав нашего проекта.

В основном цикле программы опрашивается вход GPIO, к которому на плате TangNano 9K подключена кнопка, и, в зависимости от её состояния, выдаются разные сигналы управления на выходы GPIO, к которым на плате TangNano 9K подключены светодиоды. Также в зависимости от состояния кнопки через UART выдаются сообщения «Button_ON» или «Button_OFF». Также в основном цикле проверяется, не принят ли байт данных через UART, и, если принят, производится его сравнение с заданной константой (символ «A»). В случае совпадения через UART выдаётся сообщение «OK» и устанавливается высокий уровень на одном выходе GPIO (вывод 86 ПЛИС на плате TangNano 9K), в случае несовпадения – сообщение «ERR» и низкий уровень на указанном выходе GPIO.

Как уже было сказано, функция SystemInit() является стандартной и содержится в файле system_GOWIN_M1.c. Однако в нашем случае в этот файл необходимо внести следующие правки. Значение частоты синхросиг-

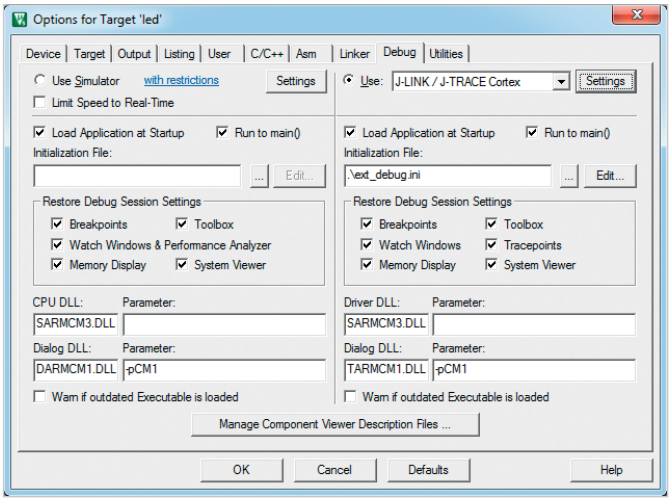

Рис. 8. Страница «Debug» окна настройки проекта

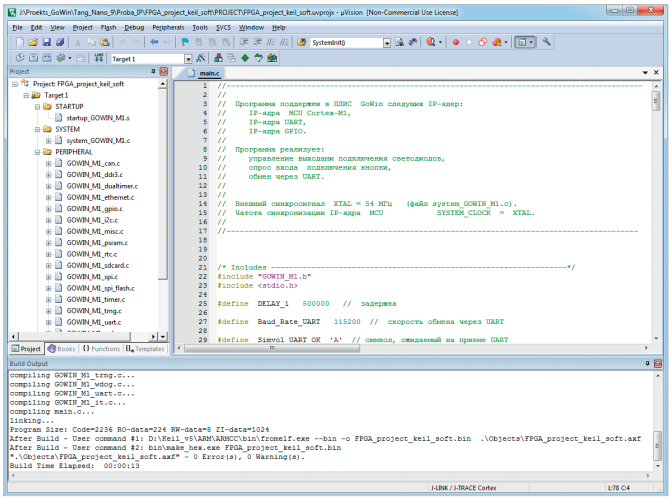

Рис. 9. Безошибочное завершение компиляции проекта

нала процессорного IP-ядра, по умолчанию равное 50 МГц, необходимо задать равным 54 МГц, как показано ниже:

```
#define XTAL (54000000UL) /* Oscillator frequency */
#define SYSTEM_CLOCK XTAL /* 54MHz */
```

Это связано с тем, что выходная частота системы PLL нашего проекта ПЛИС, которую предполагается использовать в качестве синхросигнала процессорного IP-ядра, была нами задана при конфигурировании PLL равной 54 МГц – удвоенному значению частоты синхросигнала ПЛИС (27 МГц), как описано в [1].

При обращении к линиям GPIO в программе используются имена GPIO_Pin_0 – GPIO_Pin_15, так как процессорное IP-ядро Gowin_EMPU_M1 поддерживает в ПЛИС периферийное IP-ядро GPIO, состоящее из 16 линий GPIO. Установка и сброс выходов GPIO в программе осуществляются с помощью стандартных функций GPIO_SetBit() и GPIO_ResetBit() соответственно. Чтение входа GPIO в программе осуществляется с помощью функции GPIO_ReadInputDataBit().

Проверка принятия байта данных через UART осуществляется с помощью следующей программной конструкции:

```
//-------------------------------------------------------------------------
#define Simvol_UART_OK 'A' // символ, ожидаемый на приёме UART
// Проверяем, не принят ли символ через UART
if ((UART0->STATE & UART_STATE_RXBF))
{
  DATA_UART_RX = UART_ReceiveChar(UART0);
// Если принят, сохраняем его
// проверяем, не ожидаемый ли символ принят
 if (DATA_UART_RX == Simvol_UART_OK)
 {
 UART_Send_OK (); // если ожидаемый, отвечаем OK
   GPIO_SetBit(GPIO0,GPIO_Pin_15); // уст. контрольный выход ПЛИС
 }

 else
 {
 UART_Send_ERR (); // если другой, отвечаем ERR
  GPIO_ResetBit(GPIO0,GPIO_Pin_15); // сбр. контрольный выход ПЛИС
 }

 }
//-------------------------------------------------------------------------
```

Передача байта данных через UART осуществляется с помощью стандартной функции UART_SendChar().

При старте программа выдаёт через UART начальное приветствие: «Test IP-Core soft Keil».

После завершения написания исходного кода необходимо произвести его компиляцию. Для этого в главном меню нужно выбрать: Project > Rebuild all target files или кликнуть на иконке на инструментальной панели. После завершения компиляции необходимо убедиться в отсутствии ошибок и замечаний, информация о которых выводится в поле Build Output в нижней части окна IDE ARM Keil MDK, как показано на рис. 9. В частности, без ошибок и замечаний должны быть выполнены две пользовательские команды, исполняемые после сборки проекта. Информация об их выполнении предваряется в поле Build Output следующими префиксами.

```
After Build - User command #1:
After Build - User command #2:
```

В заключение необходимо убедиться, что в каталоге PROJECT нашего проекта появились четыре выходных файла с именами itcm0–itcm3, которые далее будут использоваться в качестве входных в проекте ПЛИС с программным процессорным IP-ядром.

Для получения дополнительной информации по использованию IDE ARM Keil MDK для разработки встроенного ПО процессорных IP-ядер ПЛИС GOWIN рекомендуется обратиться к источнику [3].

Разработка аппаратной части проекта ПЛИС с программным IP-ядром в GOWIN FPGA Designer

В среде IDE GOWIN FPGA Designer открываем проект ПЛИС fpga_project из каталога Proba. В главном меню «Tools» выбираем инструмент «IP-Core Generator», далее в открывшемся окне выбираем: Soft IP-Core > Microprocessor System > Soft-Core-MCU > Gowin_EMPU_M1. При выборе курсором позиции Gowin_EMPU_M1 справа появляется описание этого процессорного IP-ядра, как показано на рис. 10.

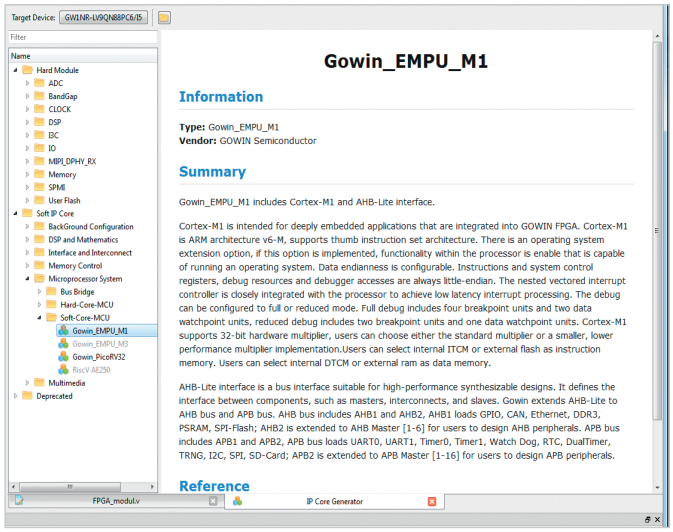

Рис. 10. Выбор IP-ядра Gowin_EMPU_M1 в «IP-Core Generator»

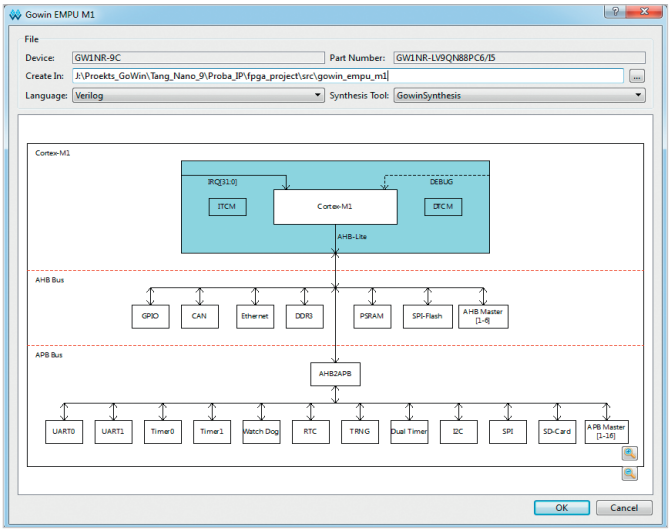

Рис. 11. Структура IP-ядра Gowin_EMPU_M1

Слева на рисунке представлен перечень поддерживаемых инструментом «IP-Core Generator» в ПЛИС GOWIN аппаратных модулей и программных IP-ядер. Позиции доступных для синтеза в нашей ПЛИС модулей и IP-ядер отображаются чёрным шрифтом, недоступных – серым.

Дважды кликаем на позиции Gowin_EMPU_M1, после чего открывается окно, показанное на рис. 11. Окно отражает структурную схему процессорного IP-ядра Gowin_EMPU_M1 и его периферии, подключаемой к ядру по шинам AHB и APB. Как можно видеть из схемы, для синтеза в качестве периферийных интерфейсов процессорного IP-ядра потенциально доступен достаточно широкий набор интерфейсных модулей: GPIO, CAN, Ethernet, DDR3, PSRAM, SPI-Flash, UART, I2C, SPI, таймеры общего назначения, охранный таймер, модуль часов реального времени RTC, интерфейс SD-карты, модули ведущих устройств для шин AHB и APB и пр. Белый цвет прямоугольников интерфейсных модулей на схеме означает, что они пока ещё не выбраны для синтеза в данном проекте и настройки для них ещё не заданы.

В верхней части окна указаны заданные по умолчанию имя нашего процессорного IP-ядра с периферией (gowin_empu_m1), путь к нему и язык синтеза ядра – Verilog. Оставляем эти настройки без изменения.

Сначала настраиваем процессорное IP-ядро. В окне со схемой наводим курсор на прямоугольник с надписью Cortex-M1 и дважды кликаем на нём. Открывается окно задания параметров ядра Cortex-M1, показанное на рис. 12.

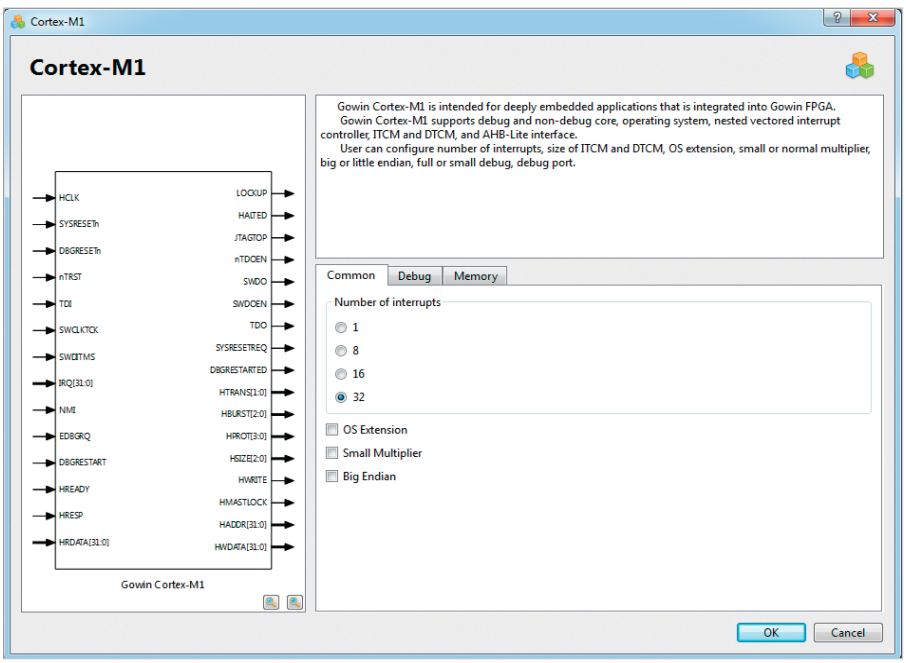

Рис. 12. Окно задания параметров ядра Cortex-M1

В этом окне три страницы: «Common», «Debug» и «Memory».

На странице «Common» снимаем установленную по умолчанию галочку на опции «OS Extension», как показано на рис. 12.

На странице «Debug» оставляем все настройки заданными по умолчанию, как показано на рис. 13, таким образом, разрешаем поддержку отладки ПО процессорного IP-ядра через интерфейсы JTAG и SW.

На странице «Memory» задаём объём памяти ITCM максимально возможным для нашей ПЛИС – 32 Кбайт. Объём памяти DTCM также задаём максимально возможным для нашей ПЛИС – 16 Кбайт. Ставим галочку на позиции «Initialize ITCM», затем в ставшем активном поле «ITCM Initialization Path» указываем с помощью кнопки ... путь к каталогу PROJECT с файлами itcm0–itcm3, ранее сгенерированными нами в проекте FPGA_project_keil_soft. При синтезе IP-ядра эти файлы автоматически включаются в него. Необходимо

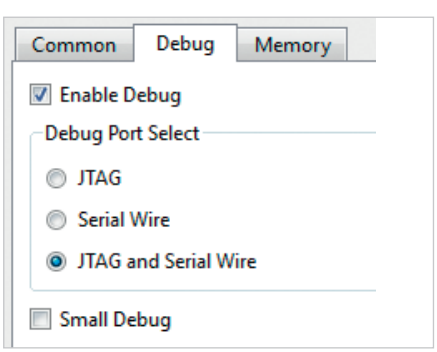

Рис. 13. Страница «Debug» окна задания параметров ядра

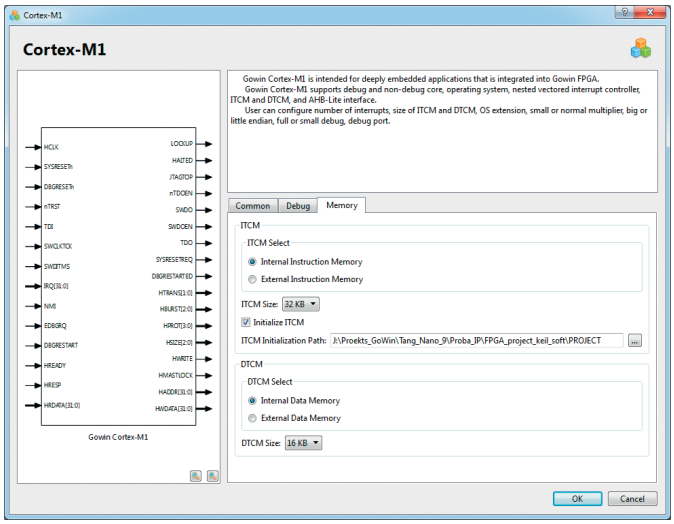

Рис. 14. Страница «Memory» окна задания параметров ядра

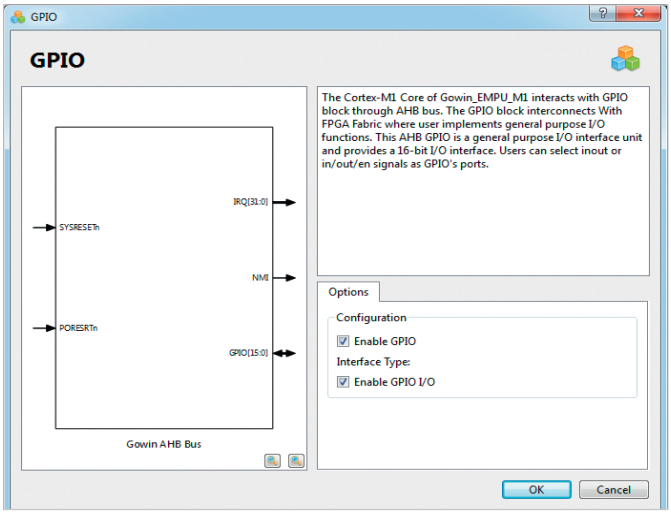

Рис. 15. Окно задания параметров GPIO

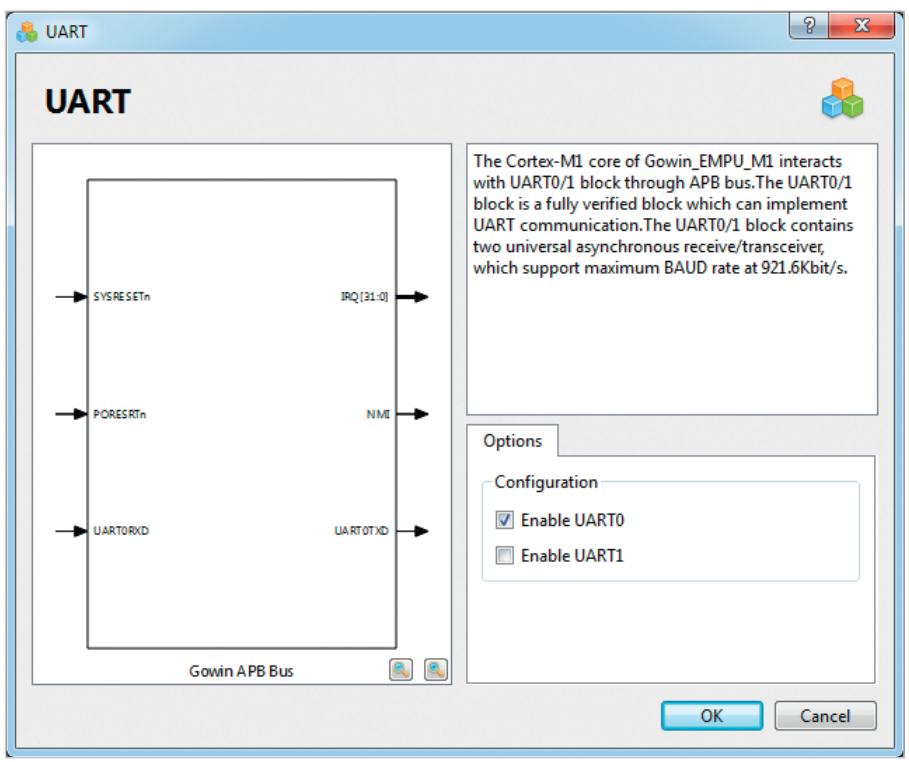

Рис. 16. Окно задания параметров UART

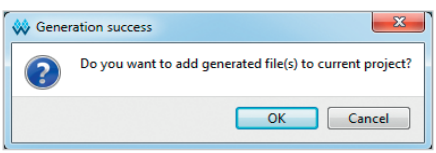

Рис. 17. Предложение включить синтезированное ядро в проект ПЛИС

заметить, что при внесении каких-либо изменений во встроенное ПО IP-ядра в IDE ARM Keil MDK и соответственно новой генерации файлов itcm0–itcm3 придётся заново синтезировать в IDE GOWIN FPGA Designer всё процессорное IP-ядро с его периферией. Затем потребуется заново пересобрать весь проект ПЛИС с новым процессорным IP-ядром.

Вид настроенной страницы «Memory» показан на рис. 14. Кликаем на кнопке «OK». Процессорное IP-ядро настроено, путь к его встроенному ПО задан.

Теперь настроим необходимую нам периферию: GPIO и UART.

В окне со схемой (рис. 11) наводим курсор на прямоугольник с надписью «GPIO» и дважды кликаем на нём. Открывается окно задания параметров модуля GPIO, показанное на рис. 15. В этом окне ставим галочки в позициях «Enable GPIO» и «Enable GPIO I/O», тем самым разрешая IP-ядру поддержку 16 двунаправленных линий GPIO (GPIO[15:0]). Как именно мы будем использовать эти линии (в качестве входов или выходов), будет показано ниже. Кликаем на кнопке «OK». Модуль GPIO настроен.

В окне со схемой (рис. 11) наводим курсор на прямоугольник с надписью UART и дважды кликаем на нём. Открывается окно задания параметров модуля UART, показанное на рис. 16. В этом окне ставим галочки в позиции «Enable UART0», тем самым разрешая IP-ядру поддержку одного из двух имеющихся модулей UART – UART0. Кликаем на кнопке «OK». Модуль UART настроен.

После настройки модулей GPIO и UART их прямоугольники на схеме (рис. 11) меняют цвет с белого на зелёный, в знак того, что они выбраны для синтеза и настроены. Кликаем на кнопке «OK» в окне на рис. 11. Запускается процесс синтеза процессорного IP-ядра и его периферии, который занимает некоторое время и завершается открытием окна с предложением включить синтезированное ядро в наш проект ПЛИС (рис. 17). Соглашаемся – кликаем на кнопке «OK».

После этого открываются следующие автоматически созданные файлы:

- файл синтезированного IP-ядра с периферией на языке Verilog HDL gowin_empu_m1.v. Заметим, что исходный код в нём недоступен для просмотра и копирования (зашифрован);
- файл шаблона заголовка модуля синтезированного IP-ядра на языке Verilog HDL gowin_empu_m1_tmp.v, предназначенный для копирования в буфер и вставки в файл верхнего уровня нашего проекта ПЛИС в качестве модуля нижнего уровня.

Выглядит шаблон, как показано на рис. 18а. В нём имеются цепи GPIO (inout [15:0] GPIO), входы синхросигнала и сброса IP-ядра (input HCLK и input hwRstn), цепи отладочного интерфейса JTAG (inout JTAG_3 – inout JTAG_18), вход и выход линий приёма и передачи UART0 (input UART0RXD и output UART0TXD), а также пара служебных выходных сигналов ядра. Последние мы использовать не будем. Из шаблона необходимо скопировать исходный код и вставить его в наш файл модуля верхнего уровня проекта (FPGA_modul.v), отредактировав его путём задания ему уникального имени, а также имён входных и выходных сигналов, как показано на рис. 18б. В качестве синхросигнала модуля IP-ядра используем выходной сигнал нашей системы PLL (clkout_o_PLL). Остальные цепи и сигналы модуля IP-ядра привязываем к цепям и сигналам нашего проекта, как показано на рис. 19. Цепи GPIO_io[0]-GPIO_io[5] назначаем для управления выходами ПЛИС led_0-led_5, к которым на плате TangNano 9K подключены светодиоды. Цепь GPIO_io[6] назначаем для приёма сигнала от входа ПЛИС button_0, к которому на плате TangNano 9K подключена кнопка. Цепь GPIO_io[15] назначаем для управления выходом ПЛИС Output_upr_ind_LCD.

Запускаем общую сборку проекта ПЛИС (синтез и компиляцию) кликом на иконке «Run All» на панели инструментов: . В случае безошибочного завершения сборки можно переходить к загрузке проекта в конфигурационную память целевой ПЛИС, как описано в [1]. В ходе загрузки в конфигурационную память ПЛИС будут автоматически загружены как файлы встроенного ПО IP-ядра (itcm0–itcm3), так и файл битового потока проекта ПЛИС, включающего IP-ядро и его периферию (fpga_project.fs).

```
//Copyright (C)2014-2022 Gowin Semiconductor Corporation.
//All rights reserved.
//File Title: Template file for instantiation
//GOWIN Version: GowinSynthesis V1.9.8.08
//Part Number: GW1NR-LV9QN88PC6/I5
//Device: GW1NR-9C
//Created Time: Thu Jun 15 12:03:10 2023

//Change the instance name and port connections to the signal names
//--------Copy here to design--------

    Gowin_EMPU_M1_Top your_instance_name(
        .LOCKUP(LOCKUP_o), //output LOCKUP
        .HALTED(HALTED_o), //output HALTED
        .GPIO(GPIO_io), //inout [15:0] GPIO
        .JTAG_3(JTAG_3_io), //inout JTAG_3
        .JTAG_4(JTAG_4_io), //inout JTAG_4
        .JTAG_5(JTAG_5_io), //inout JTAG_5
        .JTAG_6(JTAG_6_io), //inout JTAG_6
        .JTAG_7(JTAG_7_io), //inout JTAG_7
        .JTAG_8(JTAG_8_io), //inout JTAG_8
        .JTAG_9(JTAG_9_io), //inout JTAG_9
        .JTAG_10(JTAG_10_io), //inout JTAG_10
        .JTAG_11(JTAG_11_io), //inout JTAG_11
        .JTAG_12(JTAG_12_io), //inout JTAG_12
        .JTAG_13(JTAG_13_io), //inout JTAG_13
        .JTAG_14(JTAG_14_io), //inout JTAG_14
        .JTAG_15(JTAG_15_io), //inout JTAG_15
        .JTAG_16(JTAG_16_io), //inout JTAG_16
        .JTAG_17(JTAG_17_io), //inout JTAG_17
        .JTAG_18(JTAG_18_io), //inout JTAG_18
        .UART0RXD(UART0RXD_i), //input UART0RXD
        .UART0TXD(UART0TXD_o), //output UART0TXD
        .HCLK(HCLK_i), //input HCLK
        .hwRstn(hwRstn_i) //input hwRstn
    );

//--------Copy end-------------------
```

```
// Модуль  IP-ядра  Cortex-M1 с IP-ядрами   UART0  и GPIO
// GPIO – двунаправленные линии
    Gowin_EMPU_M1_Top Gowin_EMPU_M1_Top_1
    (
        .LOCKUP(LOCKUP_o), //output LOCKUP
        .HALTED(HALTED_o), //output HALTED

        .GPIO(GPIO_io), //inout [15:0] GPIO

        .JTAG_3(TRST_JTAG_Soft), //inout JTAG_3
        .JTAG_4(GND_Wire), //inout JTAG_4
        .JTAG_5(TDI_JTAG_Soft), //inout JTAG_5
        .JTAG_6(GND_Wire), //inout JTAG_6
        .JTAG_7(TMS_JTAG_Soft), //inout JTAG_7
        .JTAG_8(GND_Wire), //inout JTAG_8
        .JTAG_9(TCLK_JTAG_Soft), //inout JTAG_9
        .JTAG_10(GND_Wire), //inout JTAG_10
        .JTAG_11(RTCK_JTAG_Soft), //inout JTAG_11
        .JTAG_12(GND_Wire), //inout JTAG_12
        .JTAG_13(TDO_JTAG_Soft), //inout JTAG_13
        .JTAG_14(GND_Wire), //inout JTAG_14
        .JTAG_15(RST_JTAG_Soft), //inout JTAG_15
        .JTAG_16(GND_Wire), //inout JTAG_16
        .JTAG_17(GND_Wire), //inout JTAG_17
        .JTAG_18(GND_Wire), //inout JTAG_18

        .UART0RXD(UART0RXD_i), //input UART0RXD
        .UART0TXD(UART0TXD_o), //output UART0TXD

        .HCLK(clkout_o_PLL), //input HCLK
        .hwRstn(nrst) //input hwRstn
    );
```

Рис. 18. а) Шаблон заголовка модуля нижнего уровня IP-ядра, б) отредактированный заголовок модуля нижнего уровня IP-ядра

```
//------------------------------------------------------------------------------------------------------------------------
// IP-ядро MPU и его обслуживание
//------------------------------------------------------------------------------------------------------------------------

wire  GND_Wire;  //  цепь  общего провода

wire[15:0] GPIO_io; //   цепи входов/выходов GPIO

assign GND_Wire = 0;  //  управление цепью общего провода



// управление светодиодами из IP-ядра
assign led_0 = GPIO_io[0]; // управление светодиодом от GPIO  IP-ядра  MCU
assign led_1 = GPIO_io[1]; // управление светодиодом от GPIO  IP-ядра  MCU
assign led_2 = GPIO_io[2]; // управление светодиодом от GPIO  IP-ядра  MCU
assign led_3 = GPIO_io[3]; // управление светодиодом от GPIO  IP-ядра  MCU
assign led_4 = GPIO_io[4]; // управление светодиодом от GPIO  IP-ядра  MCU
assign led_5 = GPIO_io[5]; // управление светодиодом от GPIO  IP-ядра  MCU


 // чтение состояния кнопки в IP-ядро MCU
assign  GPIO_io[6] = button_0;

// управление выходом управления индикации LCD  из IP-ядра MCU
assign  Output_upr_ind_LCD = GPIO_io[15]; // управление цепью управления индикации LCD



// UART0  в  проекте ПЛИС  не обслуживается, его линии с входов/выходов ПЛИС транзитом идут в IP-ядро MPU
```

Рис. 19. Привязка к цепям и сигналам проекта сигналов IP-ядра

Отладка ПО для программного IP-ядра в IDE ARM Keil MDK

После завершения загрузки проекта в целевую ПЛИС встроенное ПО процессорного IP-ядра начинает выполняться одновременно с остальным проектом ПЛИС по сигналу системного сброса. Поскольку при синтезе процессорного IP-ядра мы задали возможность поддержки отладки ПО процессорного IP-ядра Gowin_EMPU_M1 через интерфейсы JTAG и SW, то теперь можно проверить возможность этой отладки «в железе», подключив к нашему макету внешний аппаратный JTAG-отладчик. В качестве такового в [3] рекомендуется использовать отладчик J-LINK или U-LINK. Автор использовал недорогой китайский клон отладчика J-LINK, подключённый к макету с ПЛИС через JTAG, а к ПК хоста – через USB. Для проверки работы периферийного модуля UART0 процессорного IP-ядра макет с ПЛИС также можно подключить к COM-порту ПК через интерфейс RS-232.

Для проверки работы отладчика откроем наш проект FPGA_project_keil_soft в IDE ARM Keil MDK, выберем в главном меню: Project > Options for Target 'Target 1 и перейдём на страницу «Debug» (рис. 8). Далее кликнем на кнопке «Settings», после чего откроется окно настроек аппаратного отладчика J-LINK/J-TRACE Cortex, показанное на рис. 20. На странице «Debug» этого окна в поле «J-Link/J-Trace Adapter» отображается информация о подключённом в данный момент к ПК аппаратном отладчике J-LINK: его серийный номер, название модели, версии прошивки, а также выбранный по умолчанию отладочный интерфейс (JTAG) и максимальная поддерживаемая отладчиком скорость обмена через него (5 МГц). Последние два параметра доступны для редактирования пользователем – вместо JTAG можно выбрать интерфейс SW, который также поддерживается ядром Gowin_EMPU_M1, а скорость обмена выбрать из выпадающего списка. Для улучшения стабильности отладки скорость обмена через JTAG понизим до 1 МГц.

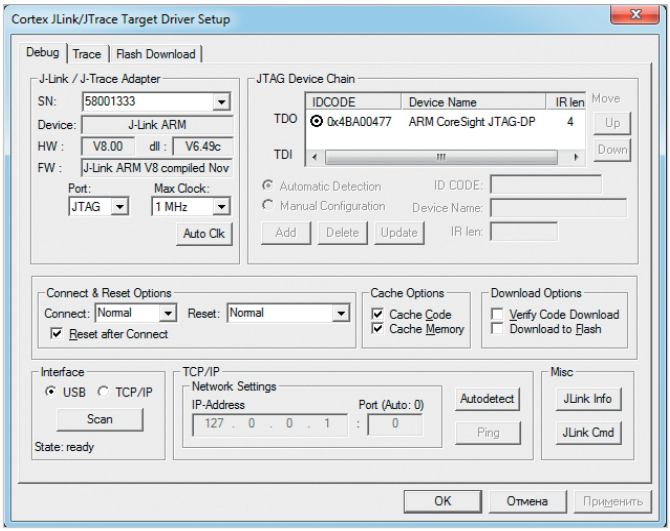

Рис. 20. Окно настроек отладчика J-LINK/J-TRACE Cortex

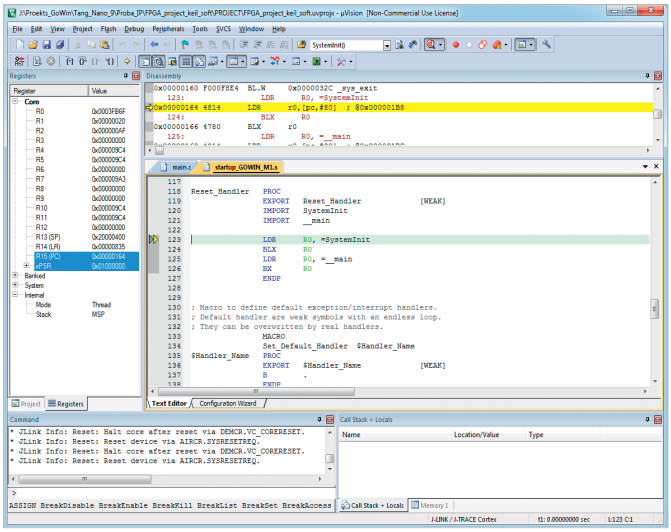

Рис. 21. Окно программного отладчика в сессии отладки

В поле «JTAG Device Chain» отображается информация о подключённой к отладчику целевой системе, считанная отладчиком из неё: код идентификации (IDCODE) и точное наименование процессорного ядра целевой системы. Из рис. 20 мы видим, что отладчик обнаружил и опознал подключённую целевую систему как ядро с архитектурой ARM. Если информация о целевой системе в поле «JTAG Device Chain» отсутствует, значит, отладчик её «не видит». В этом случае необходимо проверить подключение отладчика через JTAG к макету и наличие на макете питания. (Питание на макет поступает через интерфейс USB платы TangNano 9K, который должен быть подключён к любому порту USB хоста или к источнику +5 В). Если всё в порядке, кликом на кнопке «OK» закрываем окно настроек отладчика, следующим кликом на «OK» закрываем окно опций проекта.

Запускаем сессию отладки путём выбора в главном меню: Debug > Start/Stop Debug Session. Открывается окно программного отладчика, показанное на рис. 21. В нём доступны все основные средства отладки, включая возможность просмотра содержимого регистров процессорного ядра и памяти. В сессии отладки можно производить сброс процессорного ядра целевой системы (кнопка RST на инструментальной панели), запускать встроенное ПО на непрерывное выполнение (кнопка), останавливать выполнение ПО (кнопка), а также производить пошаговое выполнение с показом захода управления в функции или без показа (кнопки). При этом можно убедиться, что запущенное на выполнение встроенное ПО выполняется IP-ядром в реальном времени и в соответствии с алгоритмом функционирования, описанным выше. Выход из сессии отладки осуществляется путём повторного выбора в главном меню: Debug > Start/Stop Debug Session. Таким образом, в общем и в целом, программное процессорное IP-ядро ведёт себя как аппаратное.

Здесь необходимо подчеркнуть один важный момент. В описанных выше проектах на базе единой логической матрицы ПЛИС платы TangNano 9K синтезированы две разные системы, каждая из которых располагает своими собственными аппаратными ресурсами:

- собственно проект ПЛИС, управляющий ЖКИ и опрашивающий один вход GPIO (Input_upr_ind_LCD – выв. 85 ПЛИС на плате TangNano 9K);
- процессорное IP-ядро Gowin_EMPU_M1 со своей периферией, осуществляющее обмен через UART с внешними устройствами, управляющее шестью выходами GPIO, к которым на плате TangNano 9K подключены светодиоды, опрашивающее один вход GPIO, к которому на плате TangNano 9K подключена кнопка, а также управляющее ещё одним выходом GPIO (Output_upr_ind_LCD – выв. 86 ПЛИС на плате TangNano 9K).

Проект ПЛИС, кроме того, обеспечивает для IP-ядра Gowin_EMPU_M1 синхросигнал и сигнал сброса. Кроме как по питанию, сбросу и синхросигналам внутри ПЛИС, эти системы более никак не связаны. В представленных макете и проектах они взаимодействуют только с помощью внешней короткозамкнутой перемычки между выводами 85 и 86 ПЛИС на плате TangNano 9K (рис. 13 из [1]), по которой логический сигнал Output_upr_ind_LCD из IP-ядра поступает на логический вход проекта ПЛИС Input_upr_ind_LCD, тем самым обеспечивая IP-ядру доступ к управлению ЖКИ. Это, конечно же, крайне неудобный, медленный и затратный в плане расхода линий GPIO способ взаимодействия. Чтобы реализовать взаимодействие между процессорным IP-ядром и проектом ПЛИС внутри самой ПЛИС, необходимо в состав проекта ПЛИС включить интерфейс ведомого устройства одной из шин APB или AHB. В этом случае IP-ядро Gowin_EMPU_M1, для которого доступен периферийный модуль – ведущее устройство (Master) на шине APB или AHB (рис. 11), сможет взаимодействовать с проектом ПЛИС непосредственно. Однако описание порядка разработки такого решения выходит за рамки данной статьи.

Литература

1. *Редькин П.* ПЛИС фирмы Gowin Semiconductor: номенклатура, характеристики, разработка приложений. Часть 3. Инструментальные средства программной поддержки ПЛИС GOWIN, разработка аппаратной части проектов // Современная электроника. 2023. № 6. С. x - x;.
2. Gowin_EMPU_M1 Hardware Design. Reference Manual. IPUG531-1.9E, 10/12/2021.
3. Gowin_EMPU_M1 IDE Software. Reference Manual. IPUG536-1.8E, 10/12/2021.

НОВОСТИ МИРА

Китай ответит на санкции США самым передовым литографом, работающим на ускорителе частиц. С его помощью Китай сможет создавать 2-нм чипы

Как пишет South China Morning Post, Китай изучает новые возможности производства чипов внутри страны. И одной из таких является передовой литограф, работающий на ускорителе частиц. Команда из Университета Цинхуа уже ведёт активные переговоры с властями нового района Сюнган в провинции Хэбэй на севере Китая о выборе строительной площадки для передового проекта. В планах – построить ускоритель частиц с длиной контура 100–150 метров, в котором луч ускорителя превратится в высококачественный источник света для производства очень «тонких» чипов.

В отличие от мирового лидера в производстве литографов, нидерландской Advanced Semiconductor Materials Lithography (ASML), выступающей за уменьшение размеров машин для изготовления чипов, китайский проект подразумевает строительство огромной фабрики, на которой разместят несколько литографических машин вокруг одного ускорителя. Такой подход обеспечит массовость и дешевизну производства однокристальных систем.

В настоящее время 7-нанометровые чипы производятся с применением EUV-литографии, а ASML со своими машинами доминирует на рынке (за прошлый год компания поставила 180 EUV-литографов, в текущем году должна поставить ещё 60). И хотя эта технология является самой изученной и, можно сказать, надёжной, китайцы решили пойти другим путём. Работы над альтернативным проектом в Университете Цинхуа под руководством профессора Тана Чуаньсяня идут с 2017 года, в них принимает участие и Huawei. Ключевой принцип – создание лазерного источника света по технологии устойчивого микрогруппирования (Steady-State Microbunching, SSMB). Впервые принципы SSMB описал профессор Чжао Ву из Стэнфордского университета в 2010 году. Чжао – бывший ученик известного физика Ян Чжэньнина.

SSMB предполагает использование энергии заряженных частиц в качестве источника излучения с узким рассеянием.

Устройство может генерировать высококачественное излучение от терагерцовых волн с длиной волны 0,3 мм до волн EUV с длиной волны 13,5 нм, пояснил Чжао.

Иными словами, SSMB даёт практически идеальный источник света, который имеет более высокую среднюю мощность в сравнении с EUV-литографией

И хотя учёные понимают всю важность их разработки для Китая, создание работающего SSMB-литографа – это перспектива не ближайшего времени.

– До нашей независимой машины EUV-литографии предстоит пройти ещё долгий путь, но источники EUV-света на базе SSMB дают нам альтернативу санкционным технологиям, – сказал профессор Тан.

– Для создания пригодной к использованию системы литографии требуются постоянные технологические инновации на основе источников света SSMB и сотрудничество с добывающими и перерабатывающими предприятиями, – добавил он.

industry-hunter.com