Часть II. 32-разрядные микроконтроллеры

Руководство к практикуму

КФУ - Казань - 2014

УДК 681.3.068

Печатается по решению редакционно-издательского совета

Института физики

Казанского федерального университета

Рецензенты

Гумеров Р.И.

Программируемые микроэлектронные системы. Лабораторный практикум.

Часть II. 32-разрядные микроконтроллеры. Руководство. – Казань: КФУ, 2014, -62 стр.

Аннотация:

Руководство предназначено для начального освоения современных тридцати двух разрядных МК на примере МК с архитектурами ARM от фирмы Atmel и xCore от xMOS. Освоение базируется на реализации примеров приложений, использующих периферийные модули и порты ввода/вывода; аппаратными платформами являются стартовые модули AS-sam7x и XK-1A. В руководстве даны описания архитектуры и системы команд для контроллеров типа AT91sam7x и XS1-L1, а также инструментария разработки приложений: *IAR Embedded Workbench для ARM, и xTimeComposer для МК от xMOS. Фирменная документация по разработке приложений на английском языке доступна через гиперссылки. Для студентов, обучающихся по направлению 011800.62.*

Оглавление

ПРОГРАММИРУЕМЫЕ МИКРОЭЛЕКТРОННЫЕ СИСТЕМЫ

МИКРОКОНТРОЛЛЕРЫ

Часть II. **32-*разрядные микроконтроллеры***

Введение

В этом разделе практикума мы будем изучать микроконтроллеры с архитектурой ARM и xCore. Архитектура ARM является наиболее распространенной сегодня архитектурой для 32-разрядных МК. Аббревиатуры ARM, CORTEX хорошо знакомы специалистам и просто интересующимся современной микропроцессорной техникой.

Микроконтроллеры от фирмы XMOS (Великобритания) замечательны тем, что имеют многоядерную архитектуру xCore, аппаратную многопоточность и обеспечивают параллельные вычисления.

В данном руководстве архитектура ARM рассматривается на примере микроконтроллера AT91Sam7x от фирмы ATMEL, а на основе чипа XS1-L1 мы рассмотрим микроконтроллеры с архитектурой xCore от компании XMOS.

ARM. Микропроцессорное ядро ARM – ключевой компонент многих 32-разрядных встраиваемых систем. Особенно привлекательна та область их применения, которая связана с встраиваемыми системами *реального времени*. Благодаря применению, развитию и адаптации фирмой «*Advanced RISC Machines»* (ARM) философии RISC процессоры с архитектурой ARM выполняют большинство команд за один такт, имеют впечатляющее быстродействие и время реакции на прерывание, а также высокую плотность кода. Компания «*ARM Limited»* занимается только проектированием и поддержкой ядра ARM, тогда как микропроцессоры и микроконтроллеры на основе этого ядра предлагает множество сторонних производителей. Ядро ARM прошло значительный эволюционный путь от своей первой реализации ARM1 в 1985 году до микропроцессоров на ядрах ARM11, работающих на частотах более 200 MHz и выполняющих

большинство команд за один такт. МК ARM имеют аппаратные блоки для операций с плавающей точкой и векторных вычислений, встроенную кэш-память, поддерживают сопроцессоры и могут быть использованы для построения многопроцессорных систем с общей памятью и при этом обладают очень низким энергопотреблением. Кроме того, наиболее развитые типы устройств семейства имеют аппаратную поддержку интерпретатора команд JAVA, содержат в себе блоки ускорения мультимедийных операций, а также обладают некоторыми свойствами сигнальных процессоров; есть блоки управления виртуальной памятью, и возможность установки полноценных операционных систем, например, ARM Linux или Windows CE. При всем при этом МК вместе со всей необходимой периферией занимает всего около квадратного сантиметра площади кристалла.

Наиболее распространенным ядром ARM является разработанное в 1995 году и постоянно совершенствуемое ядро ARM7 [1,2]. Благодаря сочетанию высокого быстродействия и весьма малого времени реакции на прерывания это ядро хорошо подходит для реализации систем реального времени.

В этой части руководства на основе микроконтроллеров AT91SAM7x256 [3] от фирмы Atmel мы рассмотрим особенности архитектуры ARM, возможности микроконтроллеров и инструментария разработки прикладных систем. Здесь будут кратко представлены архитектура и система команд ARM7TDMI, особенности периферийных устройств микроконтроллера AT91SAM7X256; даны принципиальные схемы и руководство пользователя для стартового набора разработчика AS-sam7x и среды разработки IAR.

Практическая часть - лабораторная работа - заключается в выполнении заданий связанных с разработкой приложения и получения «прошивки» состоящей из процедур инициализации процессорного ядра, используемой периферии и собственно прикладной задачи; в освоении среды разработки приложений IAR, где исходный код может быть написан на языке ассемблера

для ARM7TDMI, или на Си, C++. В настоящее время возможно получение сведений об основных элементах МК с архитектурой ARM7TDMI на русском языке на сайте

http://www.gaw.ru/html.cgi/txt/doc/micros/arm/arh_7dtmi/index.htm

xCore. Микроконтроллеры, предлагаемые британской фирмой XMOS (http://www.xmos.com), позиционируются как 32-разрядные МК, обладающие реальной многозадачностью, имеющие транспьютерную архитектуру в рамках одного мощного МК и предназначенные для использования в системах реального времени с дискретизацией 10 нс! С точки зрения программиста здесь новым является отсутствие привычных низкоуровневых схем, таких как DMA и обработчики прерываний, а также отказ от RTOS, поскольку ее функции могут выполняться микроконтроллером аппартно. Отсутствие прерываний и RTOS делают систему абсолютно предсказуемой по времени исполнения кода. Высокоскоростная межпроцессорная шина позволяет увеличить производительность системы за счет объединения ресурсов нескольких микроконтроллеров в единый вычислительный модуль. Профессиональная бесплатная среда разработки с поддержкой языков Си, C++, с расширением XC (похожим на элементы языка «Occam») и библиотекой интерфейсных модулей обладает уникальными технологиями: XMOS Timing Analyser и XScope, позволяют выделять наилучшие (или наихудшие) времена исполнения критичного по времени фрагмента кода, графически представлять структуру программы, проводить анализ во временной области. Свойства и особенности МК этого типа мы изучим на основе стартового комплекта разработчика XK-1A.

2.1.Микроконтроллеры ARM

2.1.1. Архитектура микроконтроллера AT91SAM7X256

Используемый в практикуме микроконтроллер AT91SAM7X256 (рис. 2.1.1) имеет:

- ядро ARM7TDMI
- 256 килобайт FLASH-памяти
- 64 килобайта оперативной SRAM-памяти
- блок управления памятью
- блок управления перезагрузками
- блок управления тактовой частотой
- блок управления питанием
- блок управления прерываниями
- модуль отладки
- таймер фиксированных периодических интервалов
- сторожевой таймер
- таймер реального времени
- два блока параллельного ввода/вывода
- контроллер прямого доступа к памяти
- порт устройства USB 2.0
- модуль Ethernet MAC
- контроллер синхронной последовательной шины
- два универсальных синхронных/асинхронных приёмопередатчика
- два модуля интерфейса SPI
- трёхканальный блок таймеров/счётчиков общего назначения
- четырёхканальный широтно-импульсный модулятор
- блок интерфейса TWI

- восьмиканальный 10-ти разрядный аналогово-цифровой преобразователь

Мы видим, что микроконтроллер AT91SAM7X256 это микропроцессорное ядро ARM7TDMI, скомпонованное в одном кристалле с быстродействующей FLASH-памятью (доступ в течении одного системного такта на частотах до 30МГц) и SRAM-памятью (доступ за один такт на максимальной частоте) и очень богатым набором периферийных устройств. Далее, начиная с модели программирования ядра, рассмотрим наиболее часто применяемые компоненты.

2.1.1.1. Ядро процессора.

Ядро МК AT91SAM7X - это ARM7TDMI, представляетсобой RISC процессор, основанный на архитектуре Фон-Неймана ARMv4T (рис. 2.1.4). Буквенное обозначение «TDMI» в названии ядра ARM7TDMI, согласно спецификации фирмы ARM, имеют следующую расшифровку:

T – наличие в составе ядра помимо базовой архитектуры ARM дополнительной архитектуры Thumb, что позволяет пользователю выбрать для своего приложения одно их двух состояний (систем команд): ARM или Thumb;

D – ядро имеет в своём составе дополнительные отладочные модули;

M – ядро имеет в своём составе дополнительный аппаратный умножитель, позволяющий выполнять команды умножения так называемой длинной формы (с 64-битным результатом);

I – ядро имеет в своём составе встроенную логику отладки (ICE).

Этот процессор может работать на частотах до 55МГц, обеспечивая при этом 0.9 миллиона команд в секунду на один МГц тактовой частоты. Данное ядро может декодировать два набора инструкций: ARM и THUMB: ARM представляет собой набор 32-битных инструкций, обеспечивающий высокую производительность; THUMB – это набор 16-битных инструкций, обеспечивающий высокую плотность кода. Ещё одной особенностью данного микропроцессора является трёхступенчатый конвейер:

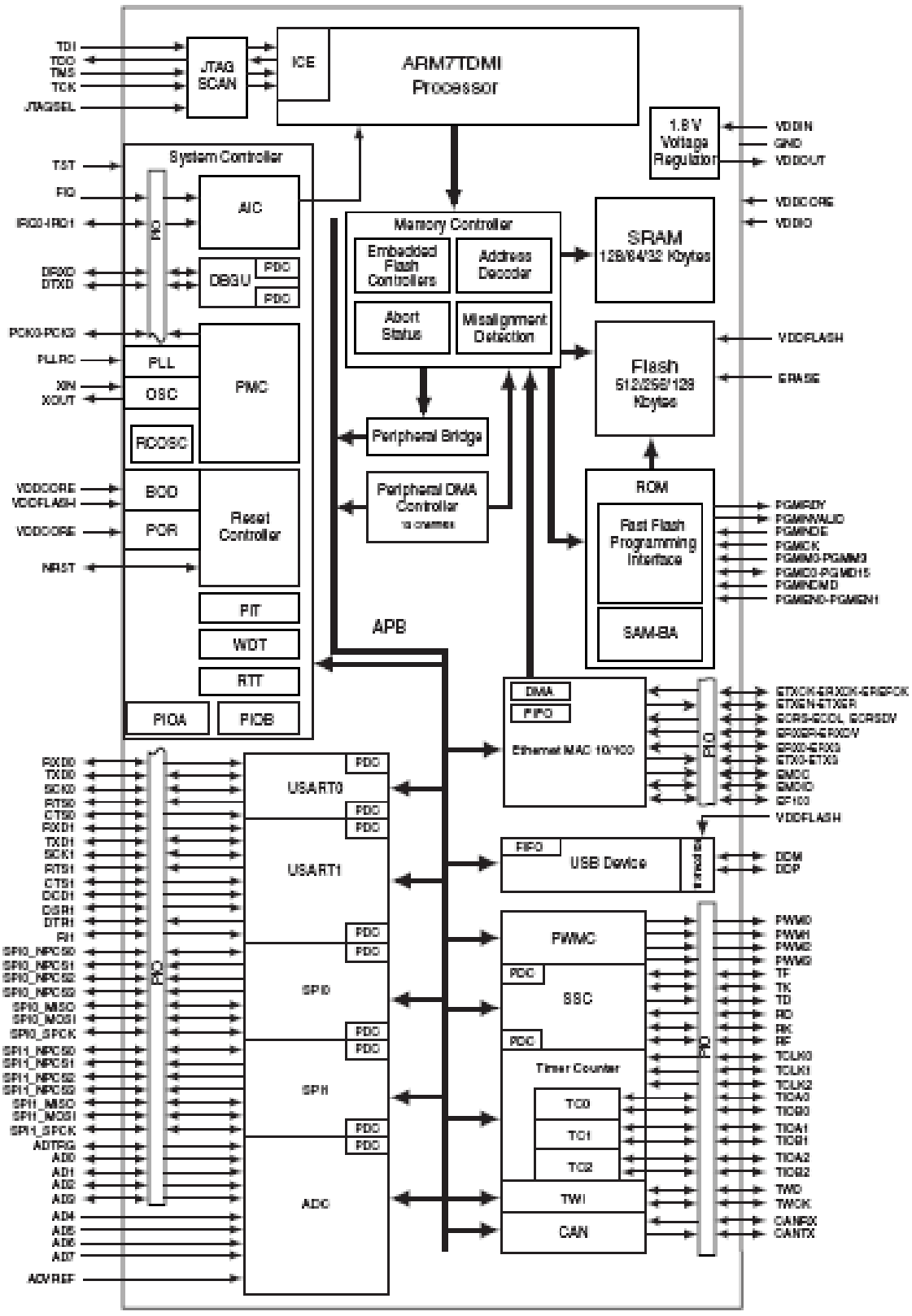


Рис. 2.1.1. Блок-схема МК AT91SAM7X.

в первом системном такте происходит извлечение команды из памяти, во втором происходит декодирование первой команды и одновременно извлечение следующей, в третьем – выполнение первой команды, декодирование второй и извлечение третьей.

ARM7TDMI имеет семь режимов процессора:

- USER – пользовательский режим
- FIQ – режим быстрого прерывания
- IRQ – режим прерывания
- SUPERVISOR – привилегированный защищённый режим
- ABORT – ошибка доступа к памяти
- SYSTEM – привилегированный пользовательский режим
- UNDEFINED – неизвестная инструкция

После перезагрузки процессор находится в режиме SUPERVISOR. Изменение режима может произойти под управлением программы, например, после того как операционная система закончила выполнение своих функций, она изменяет режим ядра на пользовательский и передаёт управление прикладной программе. А также в ответ на прерывание или исключение, например, ошибка доступа к памяти при извлечении инструкции или сохранении данных, или при вызове функций ОС для доступа к защищённым ресурсам с использованием специальной инструкции swi. Режим FIQ можно использовать для минимизации времени отклика на прерывание, режим ABORT – для реализации системы виртуальной памяти, а UNDEFINED – для поддержки сопроцессоров или программной эмуляции инструкций.

ARM7TDMI имеет в общей сложности 37 регистров (рис. 2.1.2.):

- 31 регистр общего назначения
- 6 регистров состояния

Тем не менее, только 16 регистров доступны пользователю в каждый момент времени, остальные используются для ускорения переключения контекста процессора при переходе из одного режима в другой. Так, например, каждый

режим имеет собственный регистр состояния и указатель стека, а режим FIQ – имеет дополнительно 5 собственных регистров общего назначения. Эти регистры называются “связанными”.

User and System Mode	Supervisor Mode	Abort Mode	Undefined Mode	Interrupt Mode	Fast Interrupt Mode
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8_FIQ
R9	R9	R9	R9	R9	R9_FIQ
R10	R10	R10	R10	R10	R10_FIQ
R11	R11	R11	R11	R11	R11_FIQ
R12	R12	R12	R12	R12	R12_FIQ
R13	R13_SVC	R13_ABORT	R13_UNDEF	R13_IRQ	R13_FIQ
R14	R14_SVC	R14_ABORT	R14_UNDEF	R14_IRQ	R14_FIQ
PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_SVC	SPSR_ABORT	SPSR_UNDEF	SPSR_IRQ	SPSR_FIQ

Рис. 2.1.2. Регистры ARM7TDMI.

Регистр 15 – это счётчик команд (PC – program counter), содержащий адрес извлекаемой команды. Из-за конвейера это будет адрес выполняемой команды минус 8 байт в режиме декодирования ARM или минус 4 байта в режиме декодирования THUMB, что важно учитывать при использовании PC для относительной адресации.

Регистр 14 (LR – link register) содержит адрес возврата из процедуры. При вызове процедуры ядро автоматически копирует PC в LR.

Регистр 13, по соглашению, в программах используется в качестве указателя стека (SP – stack pointer).

Регистр 16 – это регистр состояния процессора (SR – status register) (рис. 2.1.3). Биты 0-4 данного регистра определяют режим процессора. Бит 5 определяет набор декодируемых инструкций (ARM или THUMB). Биты 7 и 6 разрешают или запрещают соответственно прерывания и быстрые

прерывания. Биты 8-27 не используются. Биты 28-31 содержат флаги арифметических операций: 28 – флаг переполнения, 29 – флаг переноса, 30 – флаг нуля, 31 – флаг отрицательного значения.

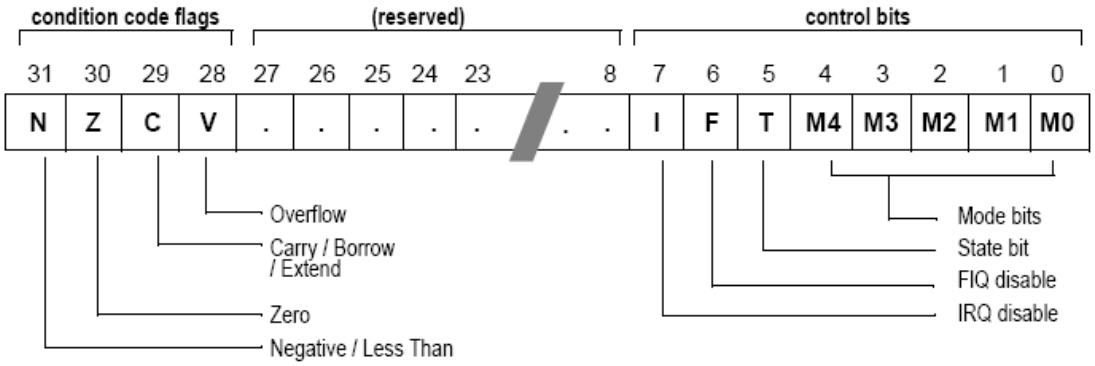


Рис. 2.1.3. Регистр состояния

Кроме того, каждый привилегированный режим, за исключением SYSTEM, имеет регистр сохранённого состояния, куда при смене режима ядро копирует регистр состояния прежнего режима.

Одной из отличительных особенностей архитектуры ядра является сдвигатель барабанного типа (barrel shifter). Благодаря его наличию, один из операндов любой арифметической операции может быть сдвинут вправо или влево на любое число битов в одном такте с основной операцией. Что ускоряет выполнение различных операций, например, умножение и деление на произвольную степень двойки.

2.1.1.2. Запуск

После перезагрузки микроконтроллер работает на частоте встроенного RC-осциллятора (slow clock), приблизительно равной 32кГц. Для работы на полной скорости необходимо сконфигурировать модуль главного осциллятора (main oscillator) и блок фазовой автоматической подстройки частоты (PLL – phase lock loop). Делается это через интерфейс модуля управления питанием (PMC – power management controller). Первый шаг – это активирование главного осциллятора и ожидание его стабилизации. Запись интервала времени стабилизации и бита MOSEN в регистр MOR блока PMC активирует главный осциллятор (main osc).

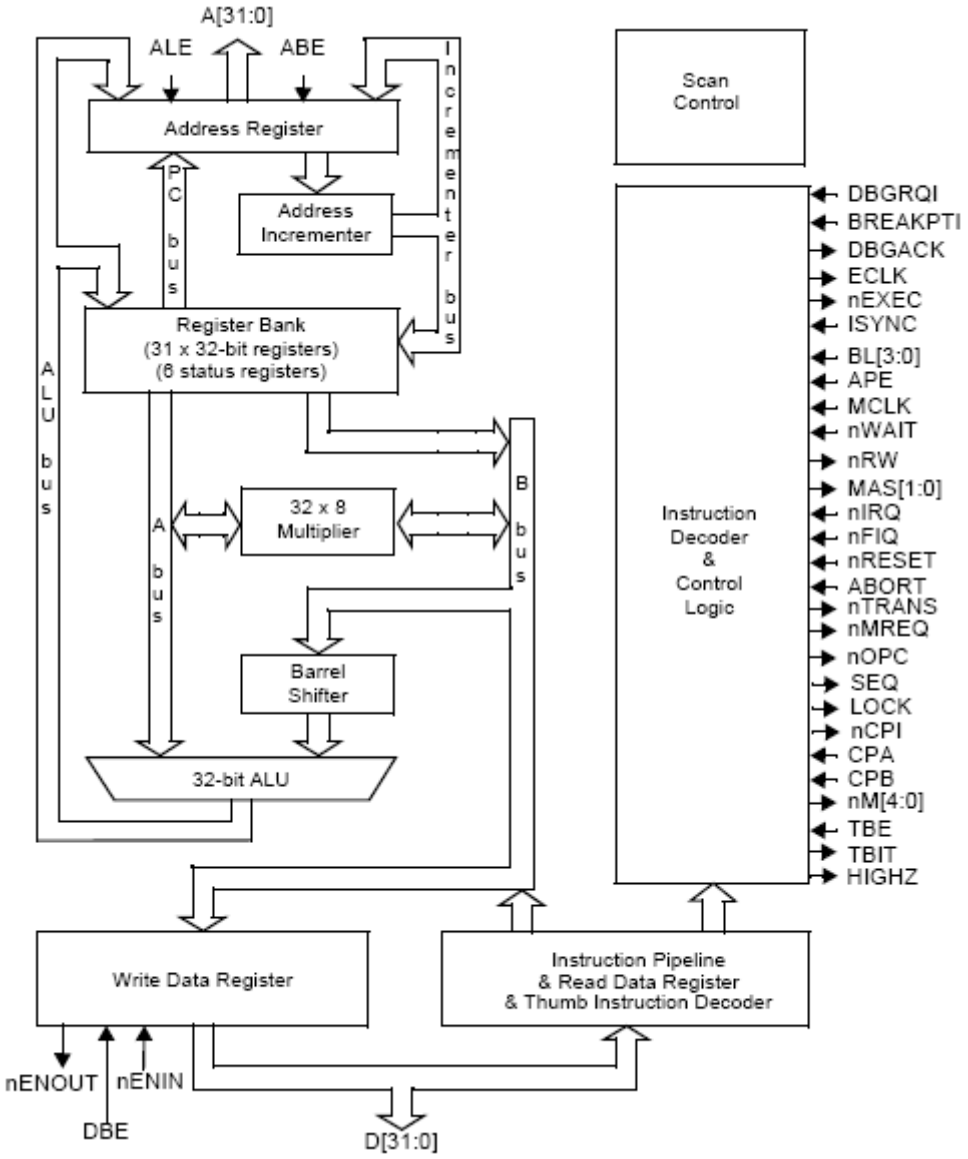


Рис. 2.1.4. Блок-схема ядра

Интервал времени стабилизации представляет собой число тактов внутреннего RC-осциллятора необходимых для стабилизации главного осциллятора. Значение этого временного интервала в миллисекундах можно найти в документации на микроконтроллер [3]. По истечении указанного числа тактов бит MOSCS регистра состояния PMC будет установлен.

После того, как главный осциллятор стабилизирован, можно приступать к конфигурированию PLL. PLL работает от тактовых импульсов главного осциллятора и состоит из двух последовательных блоков: первый

делит входную частоту, а второй умножает её. Значения, на которые эта частота делится и умножается (DIV и MUL factors), записываются в соответствующие битовые поля регистра PLLR блока PMC. Программа “PLL-calculator”, которая на основе заданной частоты главного осциллятора и требуемой выходной частоты PLL-блока высчитывает значения параметров DIV и MUL доступна на сайте фирмы ATMEL (www.atmel.com). Она позволяет получить наиболее близкое соответствие реальной выходной частоты и требуемой. Так для входной частоты 18.432МГц (частота кварца на нашей плате) и требуемой выходной равной 96МГц “PLL-calculator” дает значения для DIV равное 14, и 73 для MUL. По аналогии с главным осциллятором, время стабилизации PLL должно быть записано в соответствующее поле регистра PLLR блока PMC. И оно может быть с определено с помощью документации на микроконтроллер. После того как блок PLL запущен с необходимыми параметрами, МК дожидается его стабилизации. Это происходит путём опроса регистра состояния блока PMC.

На последнем этапе необходимо в качестве главной частоты микроконтроллера (main clock) выбрать выход блока PLL, а также, учитывая то, что частота ядра не должна превышать 55Мгц, задать корректное значение предделителя главной частоты. Это делается путём записи соответствующих значений в битовые поля регистра MCKR блока PMC. Причём, чтобы избежать работы ядра на недопустимых частотах, значение предделителя должно быть записано первым. Кроме того, перед выбором в качестве главной частоты микроконтроллера выхода PLL, необходимо позаботится о том, чтобы процессор и в дальнейшем мог извлекать команды из FLASH-памяти. Это обусловлено тем, что данная память не может обеспечить доступ в течение *одного* такта на частотах б*о*льших, чем 30МГц, поэтому при работе ядра на больших частотах (например, на частоте 48МГц), необходимо задать режим обращения к памяти, при котором процессор будет ждать один такт при чтении данных и два такта при записи. Конфигурирование числа тактов ожидания производится записью в

соответствующие битовые поля регистра FMR блока управления памятью (EFC – embedded flash controller).

2.1.1.3. Блок управления прерываниями

Блок управления прерываниями (AIC – advanced interrupt controller) предоставляет 8-ми уровневую приоритетную векторную систему прерываний, допускающую до 32 источников прерываний, каждое из которых может быть независимо сконфигурировано или запрещено (рис. 2.1.5).

AIC управляет состоянием выводов процессора nIRQ (standard interrupt request) и nFIQ (fast interrupt request) на основании состояния своих входов. Эти входы могут быть соединены с линиями прерываний встроенных периферийных устройств (внутренние источники прерываний), либо с портами ввода/вывода общего назначения (внешние источники прерываний).

Блок управления прерываниями позволяет назначить каждому источнику один из восьми уровней приоритета. Это означает, что более приоритетные прерывания будут обрабатываться даже во время обработки менее приоритетных.

Внутренние источники прерываний могут генерироваться только по состоянию уровней соответствующих линий прерываний или по фронту уровней на этих линиях. Внешние прерывания могут быть сконфигурированы как генерируемые на линии по высокому уровню сигнала или по низкому, по нарастающему фронту или по спаду. Кроме того, AIC предоставляет функцию “принудительное быстрое прерывание” (fast forcing), которая позволяет выбранным источникам прерываний генерировать быстрые прерывания процессора, а не обычные. Выбор условий генерации прерываний на каждом источнике производится записью соответствующего значения в поле SRCTYPE регистра SMR блока AIC. Каждое прерывание может быть запрещено или разрешено записью соответствующих битов в командные регистры AIC_IECR (Interrupt Enable Command Register) и

AIC_IDCR (Interrupt Disable Command Register) блока AIC, а маска разрешённых прерываний может быть считана из регистра AIC_IMR. Прерывания, сконфигурированные как генерируемые по фронту,

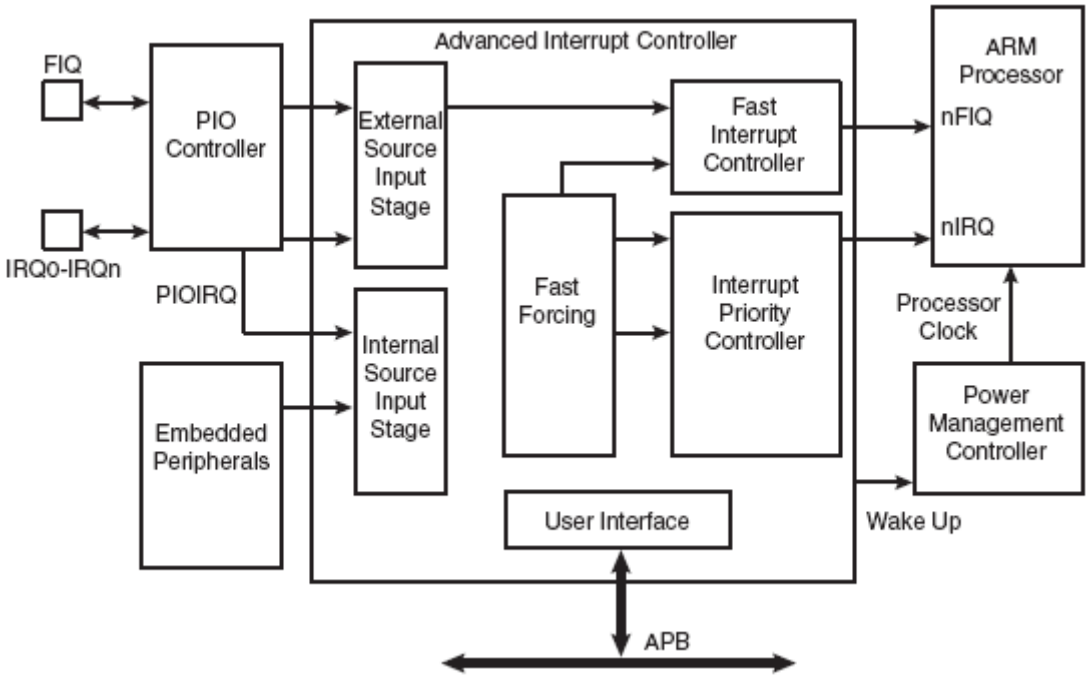


Рис. 2.1.5. Блок-схема AIC

автоматически очищаются, когда процессор входит в подпрограмму обработки прерывания и считывает регистр AIC_IVR (Interrupt Vector Register). Состояние каждого источника прерывания независимо от того, маскируется он или нет, может быть считано из регистра AIC_IPR (Interrupt Pending Register).

Адрес входа в процедуру обработки соответствующего прерывания записывается в регистр AIC_SVR1 – AIC_SVR31 (Source Vector Register 1-31). После того, как на какой-нибудь линии возникают условия прерывания, и если соответствующее прерывание разрешено, то блок управления прерываниями подает низкий уровень на линию nIRQ (или nFIQ – в случае быстрого прерывания) процессора, что заставляет процессор прервать выполнение текущей команды и загрузить в счётчик команд (регистр 15) адрес 0x18 (это адрес прерывания процессора ARM). По этому адресу должна располагаться команда “LDR PC,[PC,# -&F20]”, которая загружает в

счётчик команд адрес команды, находящийся в регистре AIC_IVR. В свою очередь блок AIC уже загрузил в регистр AIC_IVR содержимое одного из регистров AIC_SVR, соответствующего обнаруженному прерыванию. Таким образом, с помощью одной инструкции обеспечивается переход на соответствующую подпрограмму обработки прерывания.

2.1.1.4. Блок параллельного ввода/вывода

AT91SAM7X256 имеет два блока параллельного ввода/вывода (PIOC - Parallel Input/Output Controller) PIOAC и PIOBC, каждый из которых управляет 32-мя линиями ввода/вывода (рис. 2.1.6). Любая из них может служить в качестве линии общего назначения или выполнять функции связанные с работой других периферийных компонентов на чипе, например, по нарастающему уровню сигнала на ней может происходить захват какого-либо из таймеров/счётчиков.

Для каждой линии имеется возможность:

- генерации прерываний при изменениях уровня;
- подключения фильтра помех, блокирующего короткие импульсы;
- конфигурирования в качестве вывода с открытым стоком (open drain capability);
- подключения внутреннего подтягивающего резистора;
- контроля состояния линии в реальном времени.

Каждая из 32-х линий ввода/вывода ассоциирована с соответствующим битом в 32-битных регистрах блока параллельного ввода/вывода. Каждая линия ввода/вывода имеет внутренний подтягивающий резистор, подключение и отключение которого производится установкой битов соответствующей линии в регистрах PIO_PUER (Pull-up Enable Register) и PIO_PUDR (Pull-up Disable Resistor) соответствующего блока ввода/вывода. По умолчанию внутренние подтягивающие резисторы подключены на всех линиях. Также по умолчанию все линии работают как линии общего назначения. Установкой соответствующих битов в регистре PIO_PDR (PIO

Disable Register) блока PIOC линии предписывается выполнять первую из закреплённых за ней периферийных функций. Установкой битов линий в регистре PIO_ASR (A Select Register) блока PIOC линии предписывается

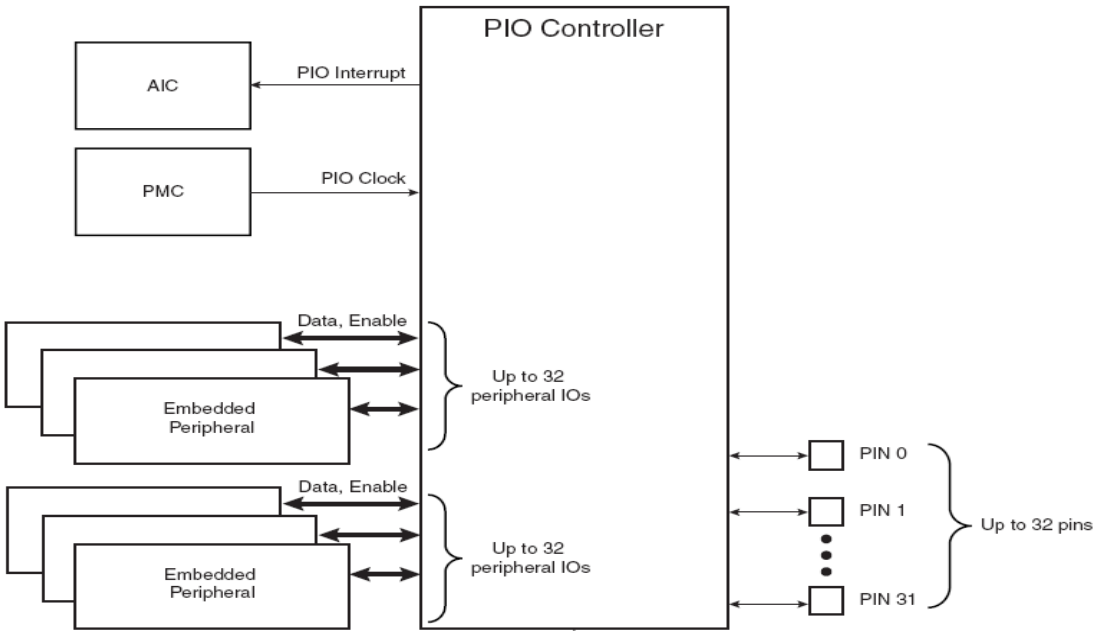


Рис. 2.1.6. Блок-схема PIOC.

выполнять вторую из закреплённых за ней на аппаратном уровне периферийных функций. Путём чтения значений регистров состояния блока параллельного ввода/вывода PIO_PSR (PIO Status Register) и PIO_ABSR (AB Select Status Register) можно узнать: используется ли линия в качестве линии общего назначения или выполняет какую-либо из периферийных функций.

Когда линия сконфигурирована как вывод общего назначения, её состояние можно контролировать записью бита, соответствующего линии, в регистры PIO_OER (Output Enable Register) и PIO_ODR (Output Disable Register). Текущий уровень на линии отображается в регистре PIO_OSR (Output Status Register).

Установка бита линии в регистре PIO_MDER (Multi-driver Enable Register) конфигурирует её как вывод с открытым коллектором (open drain).

Когда линия сконфигурирована как вход, запись бита, соответствующего линии, в регистр PIO_IFER (Input Filter Enable Register) блока PIOC включает фильтр помех, который отсеивает импульсы на линии, меньшие половины периода главной частоты (main clock).

2.1.1.5. Блок таймеров/счётчиков

Блок таймеров счётчиков (TC – timer/counter) включает в себя три независимых счётных канала (рис. 2.1.7). Каждый из которых может быть запрограммирован на выполнение различных функций, например, измерение частоты, счёт событий, измерение интервалов между событиями, генерацию импульсов, измерение необходимых задержек или широтно-импульсную модуляцию. Каждый счётный канал имеет три внешних входа, пять внутренних и два вывода общего назначения, которые могут быть сконфигурированы пользователем.

Каждый канал управляет свой линией прерывания блока AIC и имеет отдельный интерфейс программирования. Только два регистра блока таймеров/счётчиков относятся сразу ко всем каналам – это BCR (Block Control Register), запись в который запускает все таймеры одновременно, позволяя запускать их одной инструкцией, и BMR (Block Mode Register), он определяет от каких внешних входов работает каждый таймер/счётчик, позволяя объединять их в цепь (когда один таймер работает от импульсов, генерируемых другим). Все каналы блока таймеров/счётчиков идентичны и независимы. Каждый из них построен вокруг 16-битного счётчика. В обычном режиме значение счётчика увеличивается на единицу на каждом положительном фронте выбранной входной частоты. Когда счётчик достигает значения 0xffff, происходит переполнение, и он сбрасывается в ноль, одновременно с этим устанавливается бит COVFS в регистре состояния TC_SR (Status Register) соответствующего канала. Текущее значение счётчика доступно только для чтения в регистре TC_CV (Counter Value Register) соответствующего канала.

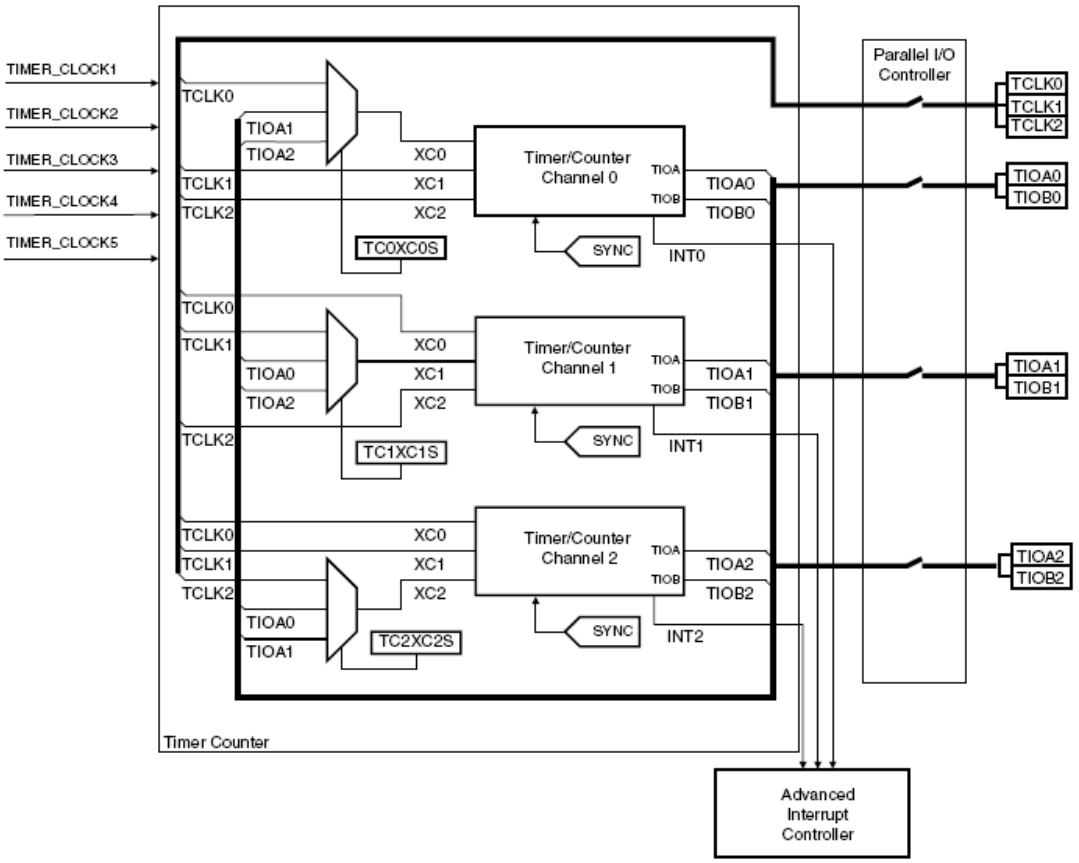


Рис. 2.1.7. Блок-схема TC.

Счётчик может быть сброшен, то есть можно задать его обнуление. Сброс происходит на следующем после задания сброса обнаруженном положительном фронте выбранной входной частоты. Выбор входной частоты осуществляется записью необходимых значений в соответствующие битовые поля регистра BMR. Кроме того, импульсы выбранной частоты могут быть инвертированы, что позволяет вести счёт на отрицательных фронтах. Также в таймерах/счётчиках предусмотрена функция выборки входной частоты (burst function). Если эта функция активна, то счёт производится только на тех входных импульсах, у которых на ассоциированных с функцией выборки линиях ввода/вывода присутствуют высокие уровени. Таймеры/счётчики могут работать в двух основных режимах: режиме генерации импульсов и режиме захвата.

Таймер/счётчик будет работать в режиме захвата (capture operating mode), если параметр WAVE в регистре TC_CMR (Channel Mode Register) сброшен в ноль. В этом режиме выводы общего назначения счётного канала (TIOA и TIOB) сконфигурированы как входы, а регистры A и B (RA и RB) используются как регистры захвата, то есть в них может быть загружено значение счётного регистра при возникновении заданных условий на входах TIOA и TIOB. Это позволяет использовать данный режим для измерения характеристик импульсов, таких как частота, период, коэффициент заполнения, и сдвиг фаз.

Если параметр WAVE регистра TC_CMR установлен, то таймер/счётчик работает в режиме генерации импульсов (waveform operating mode). Так как теперь выводы общего назначения счётного канала сконфигурированы как выходы, данный режим используется для генерации двух широтно-импульсных модулированных сигнала одинаковой частоты с независимо программируемыми коэффициентами заполнения, или для генерации различного рода одиночных или повторяющихся импульсов. В данном режиме регистры RA, RB и RC используются как регистры сравнения, то есть, при достижении счётчиком значений, записанных в этих регистрах, возможно изменение состояния выходов TIOA и TIOB, причём, регистр RA используется для контроля линии TIOA, регистр RB для контроля TIOB, а регистр RC может использоваться для изменения состояния обоих выходов. Поведение таймера/счетчика определяется 2-х битовым полем WAVESEL регистра TC_CMR. Возможны четыре случая:

- 00 – счётчик инкрементируется от 0 до 0xffff, затем сбрасывается;
- 10 – счётчик инкрементируется от 0 до значения в RC, затем сбрасывается;
- 01 – счётчик инкрементируется от 0 до 0xffff, затем декрементируется до 0;
- 11 – счётчик инкрементируется от 0 до значения в RC, затем декрементируется до 0.

2.1.1.6. Модуль Ethernet MAC

Данный модуль позволяет осуществить управление доступом к среде передачи (Media Access Control – MAC) в соответствие со стандартами 10/100 Ethernet и IEEE 802.3 Standard. Он состоит из блока проверки адресов, блока регистров контроля и статистики, блока передачи и приёма, а также блока прямого доступа к памяти (рис. 2.1.8).

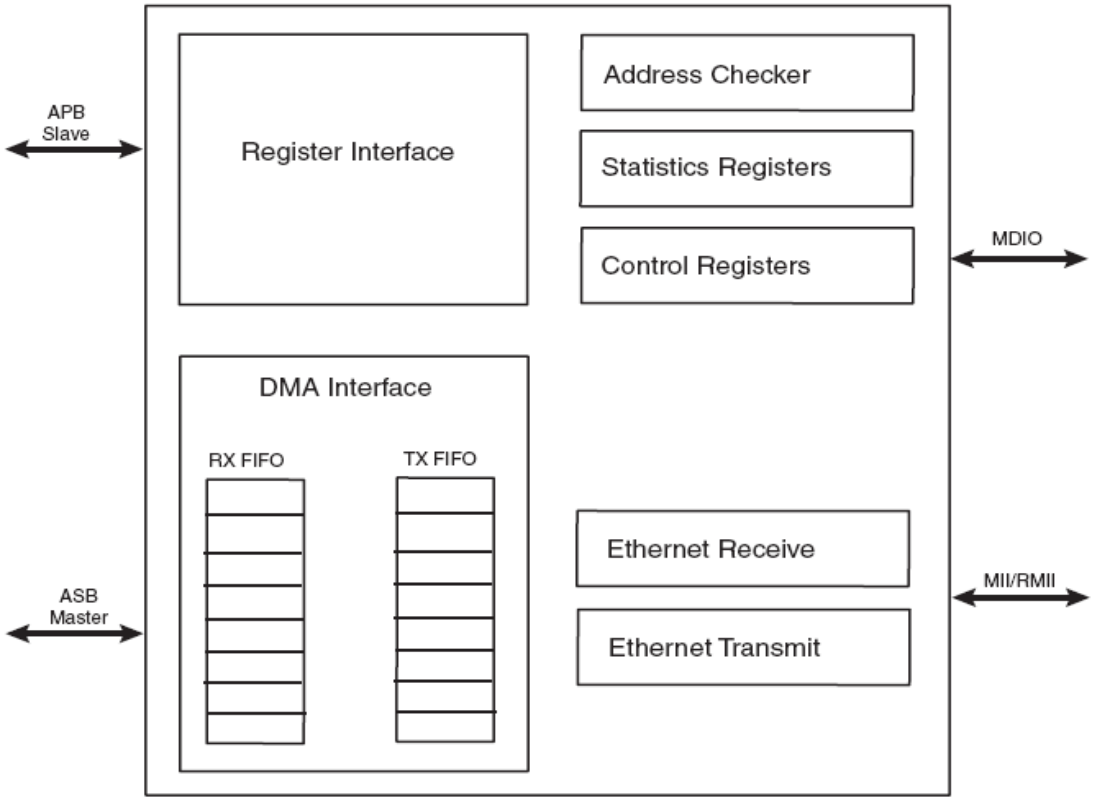


Рис. 2.1.8. Блок-схема EMAC.

Данный модуль обеспечивает интерфейс микроконтроллера и чипа физического уровня доступа к среде (PHY). Он может работать с двумя видами чипов физического уровня стандарта IEEE 802.3: *clause 45 PHY* и *clause 22 PHY*. Модуль EMAC взаимодействует с чипом PHY посредством независимого от среды передачи интерфейса MII (Media Independent Interface). Для передачи или приёма данных перед программированием EMAC соответствующий PHY должен быть корректно проинициализирован.

На используемой плате AS-Sam7X установлен чип физического уровня фирмы Davicom DM9161AC. Режим работы данного чипа устанавливается во время его перезагрузки в зависимости от состояния его выводов, которые подсоединены к линиям ввода/вывода общего назначения микроконтроллера AT91SAM7X256. Поэтому перед программированием модуля EMAC необходимо с помощью блока управления перезагрузками AT91SAM7X256 перезагрузить чип DM9161AC, подав импульс низкого уровня на линию перезагрузки микроконтроллера (микроконтроллер при этом не перезагружается – опция блока управления перезагрузками), одновременно, с помощью блока параллельного ввода/вывода установить выводы чипа PHY в необходимое положение. После того, как чип PHY проинициализирован, задействованные при перезагрузке линии ввода/вывода можно использовать по любому назначению.

Итак, модуль EMAC состоит из нескольких блоков. Путём записи необходимых значений в регистры блока контроля выбирается режим работы модуля EMAC, скорость, дуплекс, MAC-адрес, производится взаимодействие с чипом PHY, инициируется отправка фреймов и разрешается их копирование в память при приёме. В регистры блока статистики заносится разного рода служебная информация интерфейса Ethernet, а также информация об ошибках.

Блок приёма проверяет фреймы на наличие корректного заголовка, отсутствие ошибок контроля избыточным кодом, корректную длину и выравнивание. После чего предоставляет полученный фрейм блоку проверки адреса. Последний, в свою очередь, принимает решение о копировании фрейма в память и в случае успеха предоставляет фрейм модулю прямого доступа к памяти, который копирует его в программно выделенный буфер в памяти.

Блок передачи получает фрейм от блока прямого доступа к памяти, добавляет заголовок и контрольную сумму Ethernet, а также PAD если необходимо, и предаёт фрейм в соответствии со стандартом CSMA/CD

(carrier sense multiple access with collision detect). Если во время передачи случается ошибка, то после передачи jam-секции фрейм обрывается, и соответствующая информация заносится в один из регистров блока статистики.

Для того чтобы передавать и получать фреймы, необходимо предварительно выделить память для буферов приёма и передачи, сконструировать в памяти таблицу дескрипторов для каждого буфера и записать соответствующие указатели в регистры блока контроля модуля EMAC.

2.1.3. Программирование МК ARM

Данные flash-микроконтроллеры могут быть перепрограммированы несколько тысяч раз и хранят программу в своей памяти несколько десятков лет. Микроконтроллеры AT91… поддерживают программирование программаторами очень многих стандартов, позволяющими программирование кристаллов в промышленных масштабах при конвейерном производстве. Но для нас важна возможность их программирования непосредственно в готовом устройстве, для чего кампания Atmel поставляет свои микроконтроллеры с записанной в ROM память чипа программой SAM-BA – Smart ARM Microcontroller - Boot Assistance (рис. 2.1.9). Кроме того, на сайта www.atmel.com бесплатно доступна программа для OS Windows с аналогичным названием (SAM-BA for Windows), которая обеспечивает интерфейс пользователя с микроконтроллером, обмениваясь данными с программой в ROM-памяти, что делает возможным программирование контроллера с помощью USB или Com-порта.

SAM-BA позволяет записывать в FLASH-память микроконтроллера бинарный код команд микропроцессора ARM из указанного *.bin файла. Но чтобы данный код получить необходим кросс-компилятор позволяющий

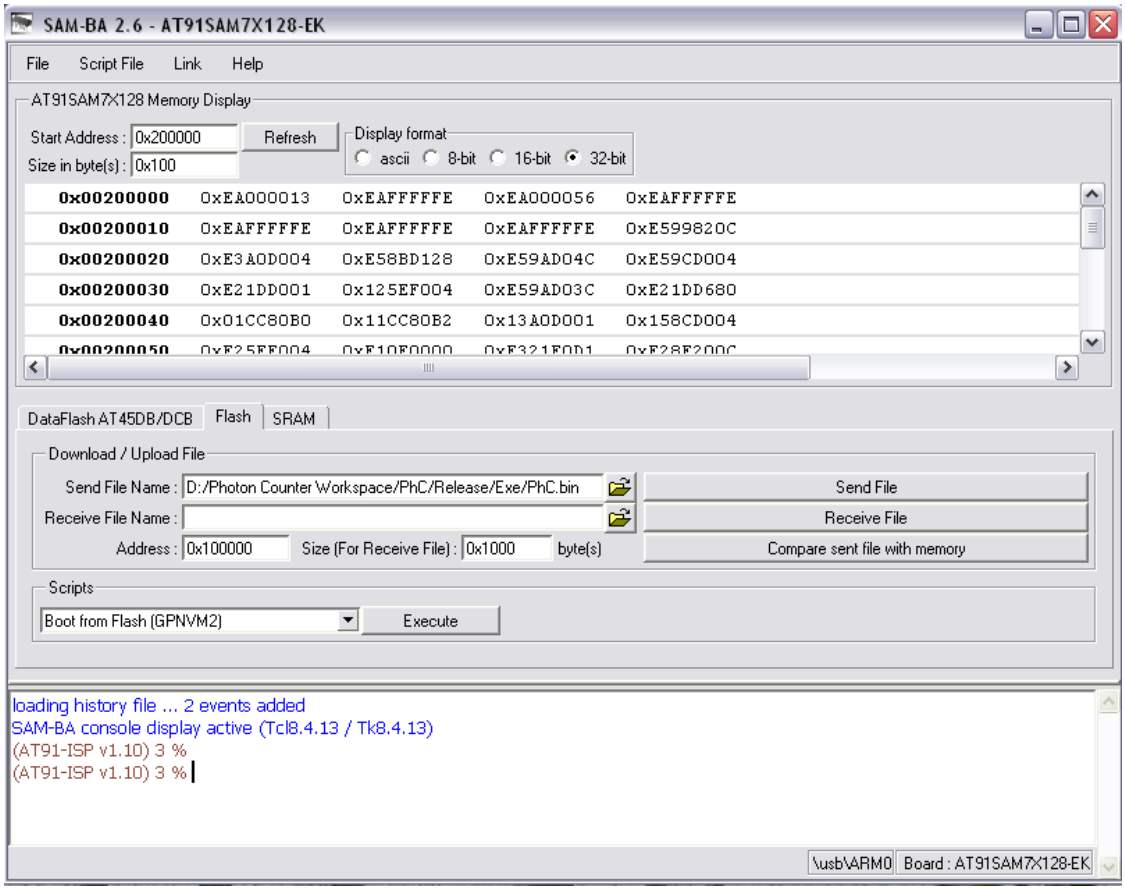


Рис. 2.1.9. Программа SAM-BA.

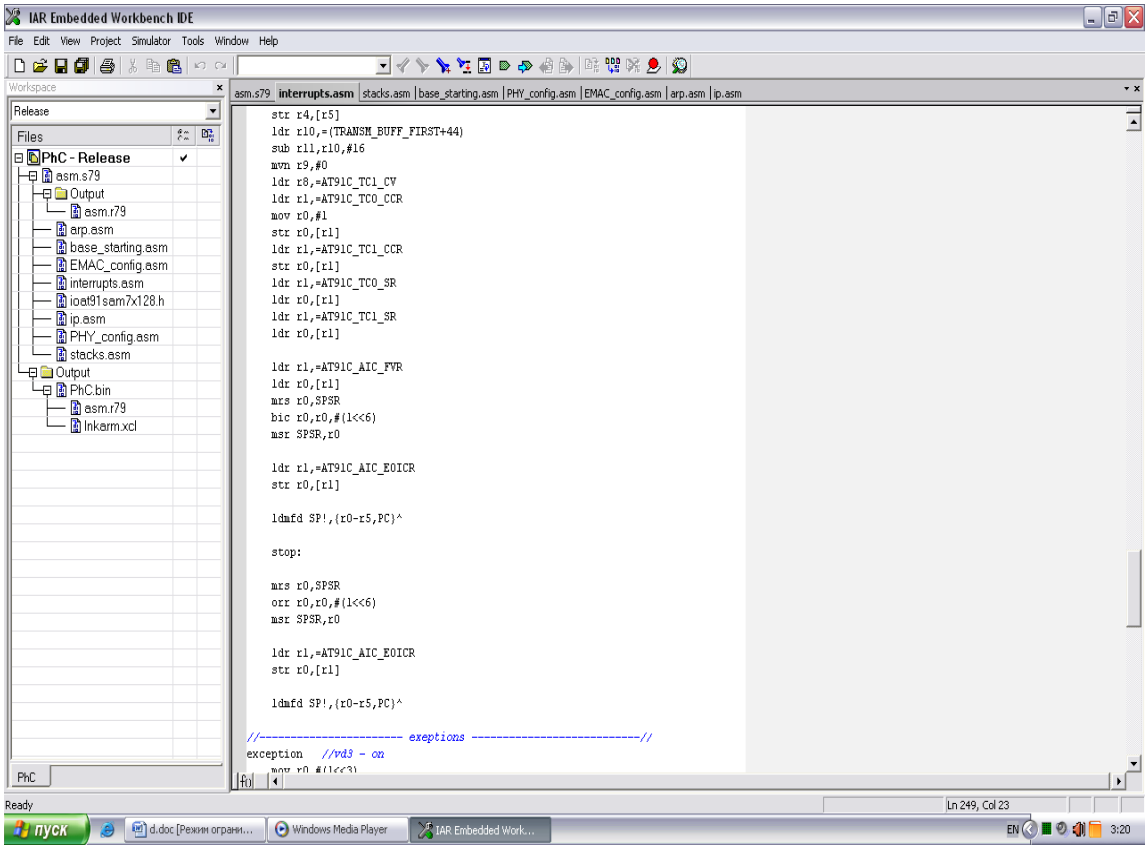


Рис. 2.1.10. Консоль среды разработчика IAR.

писать программы на каком-нибудь языке программирования, например, ассемблере или Си, и генерирующий .bin файл с кодом для ARM. Мы для создания пользовательского приложения и получения необходимого *.bin файла будем использовать некоммерческую версию интегрированной среды разработки IAR Embedded Workbench (рис.2.1.10). Программировать приложения можно как на языке ассемблера (ARM-ассемблер), так и на Си. В данном практикуме используем ассемблер.

Пример:

Написать программу, результатом которой являлось бы попеременное мигание двух светодиодов на плате AS-sam7x с периодом 1сек.

Рис. 2.1.11. Блок-алгоритм примера

2. Создание проекта в среде IAR.

После запуска IAR Embedded Workbench IDE будет открыто окно с предложениями: открыть новый проект, добавить существующий, или открыть примеры приложений. Выбираем создание нового проекта. Тогда откроется следуещее окно:

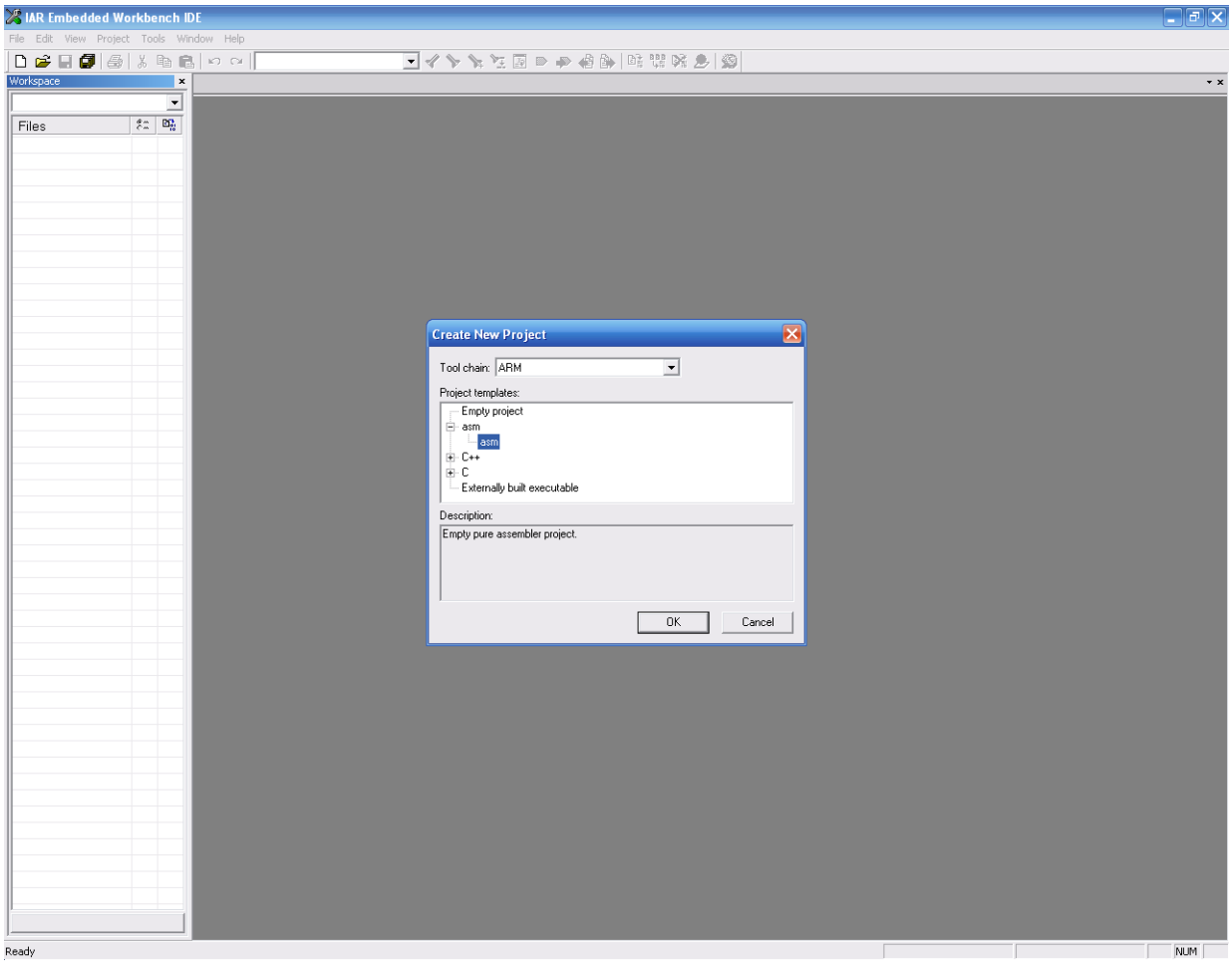


Рис. 2.1.12. IAR EWB, окно создания проекта

Здесь выбираем язык программирования; в нашем случае - asm (ассемблер). После этого появляется файловое окно, где указываем имя и местоположение нашего проекта. В результате откроется Workspace, а в окне редактора шаблон кода для ассемблера.

3. Настройка проекта.

Для прикладных пользовательских проектов опции могут быть заданы для узлов проекта всех уровней. Сначала производится задание общих опций, например, таких, как конфигурация процессора. Задание опций проекта производится в следующей последовательности: выбирается позиция

компиляции нужного проекта, например, **Release – Debug** в окне рабочей области, а в меню среды выбирается **Project>Options.** После этого откроется страница **Target** в категории **General Options** (см. рисунок ниже).

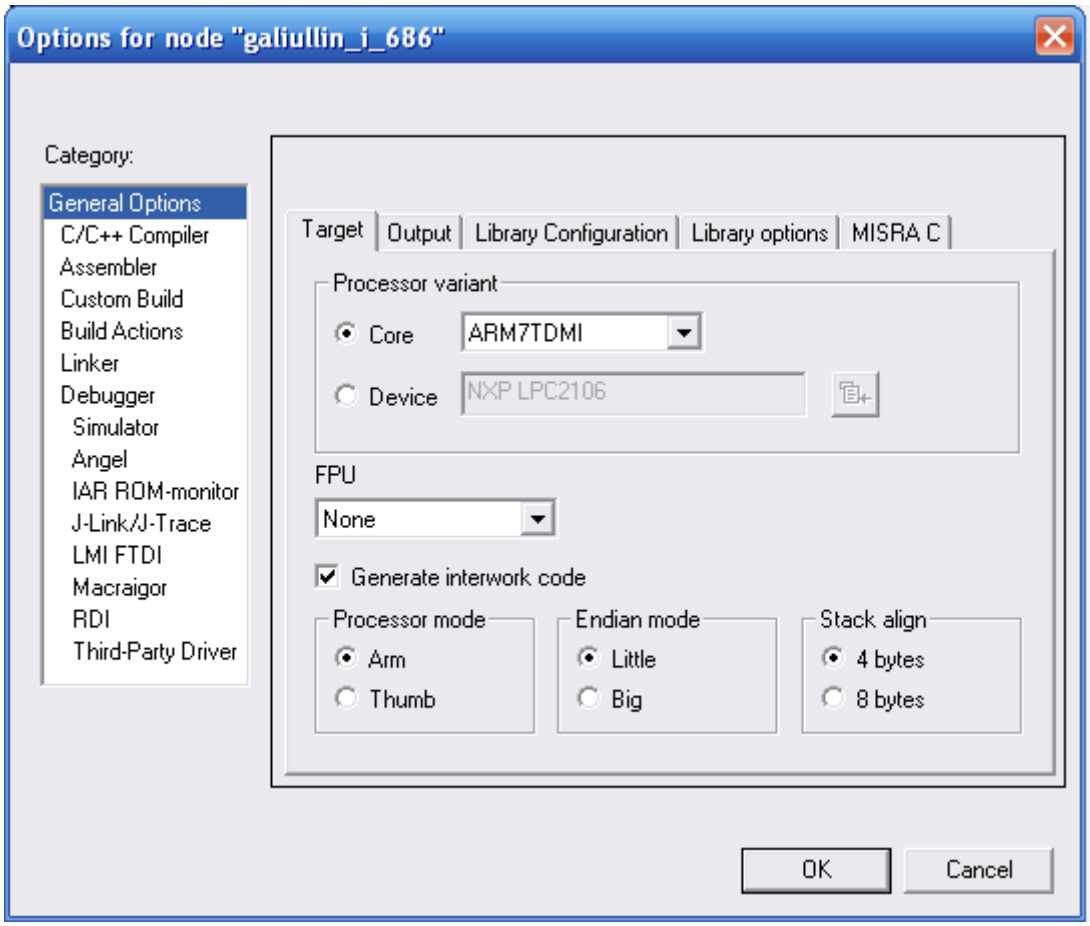


Рис. 2.1.13. Окно настройки опций

Установка требуемых опций интуитивно понятна.

4. Написание программы.

Конфигурирование режима User и IRQ

```
Reset:
 /*IRQ*/

 ldr r0,=(AT91C_ISRAM+AT91C_ISRAM_SIZE)  // Загружаем в регистр значение для указателя вершины стека
 mrs r1,CPSR                             // Далее, копируем содержимое регистра CPSR в рабочий регистр r1,
 bic r1,r1,#0x1f                         //  используя для этого специальную команду mrs
 orr r1,r1,#10010b                       //  с помощью логических команд записываем необходимое значение
 msr CPSR,r1                             //  загружаем полученное значение обратно в регистр
 mov SP,r0                               //  указываем вершину стека

 /*User*/

 sub r0,r0,#256
 mrs r1,CPSR                              // Аналогичные процедуры для инициализации режима User
 bic r1,r1,#0x1f
 orr r1,r1,#10000b
 msr CPSR,r1
 mov SP,r0
```

Конфигурирование флэш-памяти

```
ldr r0,=(1<<8)|(73<<16)          // Запись (1<<8) в регистр режима флэш-памяти означает задание количества тактовых циклов
ldr r1,=AT91C_MC_FMR             // в состоянии ожидания флэш-памяти: 2 цикла операции записи, 3 цикла операции чтения
str r0,[r1]                      // запись в 16 бит определяет количество циклов задающего генератора в одной мс
```

Запуск главного генератора

```
ldr r0,=1|(0xff<<8)                 // Устанавливаем бит MOSCEN (1=основной генератор разрешён)
ldr r1,=AT91C_CKGR_MOR              // в 8 бите определяется количество медленных циклов
str r0,[r1]                         // Когда бит MOSCEN установлен, флаг MOSCS в регистре PMS_SR автоматически устанавливается,
ldr r1,=AT91C_PMC_SR                // как только истекает время запуска основного генератора
M0_Stable:                          // Ожидание запуска
  ldr r0,[r1]
  ands r0,r0,#1
  beq M0_Stable
```

Запуск блока ФАПЧ

```
ldr r0,=(0x3f<<8)|(72<<16)|14       // Запись значений DIV и MUL: значение DIV- 0..7 биты, MUL- 16..26 биты
ldr r1,=AT91C_CKGR_PLLR             // С 8 по 13 бит- PLLCOUNT: определяет количество медленных тактовых циклов,
str r0,[r1]                         // прошедших до установки бита LOCK регистра PMC_SR с момента записи в регистр CKGR_PLLR
ldr r1,=AT91C_PMC_SR
PLL_Stable:                         // Ожидание стабилизации PLL
  ldr r0,[r1]
  ands r0,r0,#(1<<3)
  beq PLL_Stable
```

Выбор предделителя частоты ядра

```
mov r0,#(1<<2)                      // Выбранная тактовая частота/2
ldr r1,=AT91C_PMC_MCKR
str r0,[r1]
ldr r1,=AT91C_PMC_SR
MCP_Stable:                         // Ожидание стабилизации
  ldr r0,[r1]
  ands r0,r0,#(1<<3)
  beq MCP_Stable
```

Запуск ядра

```
mov r0,#3|(1<<2)                    // Выбранная тактовая частота/2
ldr r1,=AT91C_PMC_MCKR
str r0,[r1]
ldr r1,=AT91C_PMC_SR
MCK_Stable:                         // Ожидание стабилизации
  ldr r0,[r1]
  ands r0,r0,#(1<<3)
  beq MCK_Stable
```

Основное тело программы

```
  ldr r1,=(1<<20)                   // Регистры служат для зажигания и гашения светодиодов
  ldr r2,=(1<<21)

cycle:
 ldr t0,=(0x4C4B40)                 // Записываем значение для задержки
 mov t1,#1
 mov r3,r1
 mov r1,r2
 mov r2,r3
 ldr r7,=(1<<3)
 ldr r8,=AT91C_PMC_PCER             // Разрешаем тактирование
 str r7, [r8]
 ldr r8,=AT91C_PIOB_ODR             // Команда запрещения линии вывода B
 str r1, [r8]
 ldr r8,=AT91C_PIOB_OER             // Команда разрешения линии вывода B
 str r2, [r8]
delay:
 subs t0,t0,t1                      // секундная задержка
 bne delay
 b cycle
```

Рис. 2.1.14. Пример листинга проекта

код программы:

```
#define t0  r4
 #define t1  r5
 #define aTC r6
 #include "ioat91sam7x128.h"


CODE32

 b Reset
 undef_vector:
 b undef_vector
 swi_vector:
```

```
 b swi_vector
 pabt_vector:
 b pabt_vector
 dabt_vector:
 b dabt_vector
 rsvd_vector:
 b rsvd_vector
 ldr PC, [PC,# -0xF20]
 b.

Reset:

 /*IRQ*/
ldr r0,=(AT91C_ISRAM+AT91C_ISRAM_SIZE)
mrs r1,CPSR
 bic r1,r1,#0x1f
 orr r1,r1,#10010b
 msr CPSR,r1
 mov SP,r0

 /*User*/
 sub r0,r0,#256
 mrs r1,CPSR
 bic r1,r1,#0x1f
 orr r1,r1,#10000b
 msr CPSR,r1
 mov SP,r0

// запуск ядра на полной скорости:

   /* конфигурирование флэш-памяти */
 ldr r0,=(1<<8)|(73<<16)
 ldr r1,=AT91C_MC_FMR
 str r0,[r1]

/* запуск основного генератора */
 ldr r0,=1|(0xff<<8)
 ldr r1,=AT91C_CKGR_MOR
 str r0,[r1]
 ldr r1,=AT91C_PMC_SR
MO_Stable:
  ldr r0,[r1]
  ands r0,r0,#1
  beq MO_Stable

 /* запуск блока ФАПЧ */
 ldr r0,=(0x3f<<8)|(72<<16)|14
 ldr r1,=AT91C_CKGR_PLLR
 str r0,[r1]
 ldr r1,=AT91C_PMC_SR
PLL_Stable:
  ldr r0,[r1]
```

```
  ands r0,r0,#(1<<3)
  beq PLL_Stable

  /* выбор предделителя частоты ядра */
 mov r0,#(1<<2)
 ldr r1,=AT91C_PMC_MCKR
 str r0,[r1]
 ldr r1,=AT91C_PMC_SR
MCP_Stable:
  ldr r0,[r1]
  ands r0,r0,#(1<<3)
  beq MCP_Stable

 /* запуск ядра */
 mov r0,#3|(1<<2)
ldr r1,=AT91C_PMC_MCKR
 str r0,[r1]
 ldr r1,=AT91C_PMC_SR
MCK_Stable:
  ldr r0,[r1]
  ands r0,r0,#(1<<3)
  beq MCK_Stable

/*****************************************/
 // Основная программа
  ldr r1,=(1<<20)
  ldr r2,=(1<<21)

cycle:
 ldr t0,=(0x4C4B40)
 mov t1,#1
 mov r3,r1
 mov r1,r2
 mov r2,r3
 ldr r7,=(1<<3)
 ldr r8,=AT91C_PMC_PCER
 str r7, [r8]
 ldr r8,=AT91C_PIOB_ODR
 str r1, [r8]
 ldr r8,=AT91C_PIOB_OER
 str r2, [r8]
delay:
 subs t0,t0,t1
 bne delay
 b cycle

 END
```

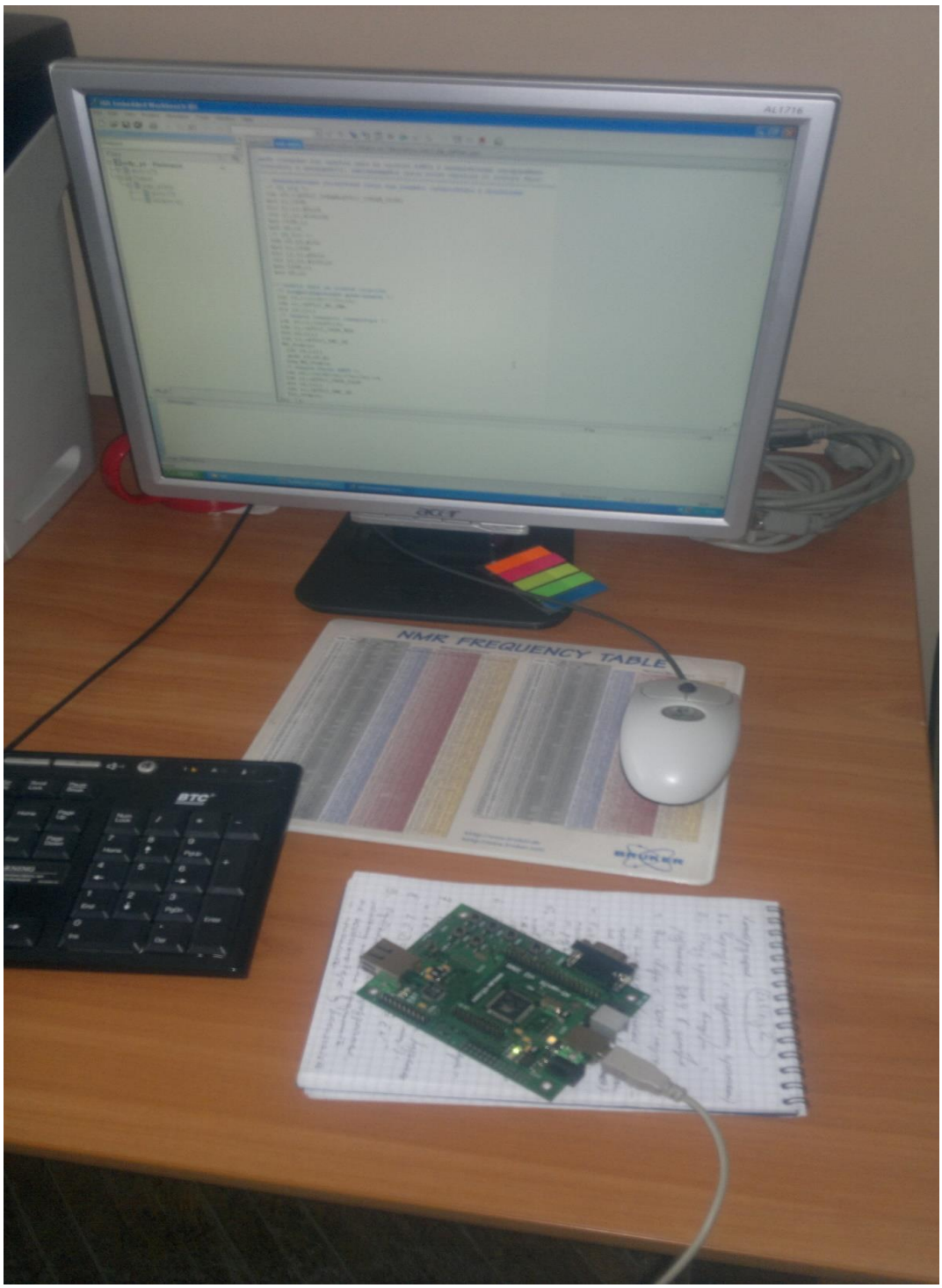


Рис. 2.1.15. Модуль AS-sam7x на рабочем месте.

2.2. Микроконтроллеры xCore

В МК построенных по архитектуре xCore можно обнаружить множество элементов, присутствующих в операционных системах реального времени (RTOS). Например, формирователи задач, таймеры, связь каналов, а также логическое разделение ядер процессора для задач реального времени. Здесь системы реального времени более предсказуемы, масштабируемы и распознаются намного быстрее, чем обычные RTOS основанные на системах с последовательными ядрами, говорится производителем.

МК типа XS1 представляют собой комбинацию некоторого числа xCore процессоров в «чипе» (рис.2.2.1), каждый из которых обладает собственной памятью, и кроме того, они обеспечивают прямую поддержку для параллельных процессов (многопоточность), коммуникации и ввод/вывод. Высокопроизводительные переключатели обеспечивают коммуникации между процессорами, а межкристальные XMOS «линки» дают возможность простого конструирования системы из нескольких «чипов». Изделия типа XS1 на практике, во многих случаях могут с успехом заменять устройства жесткой (или программируемой) логики.

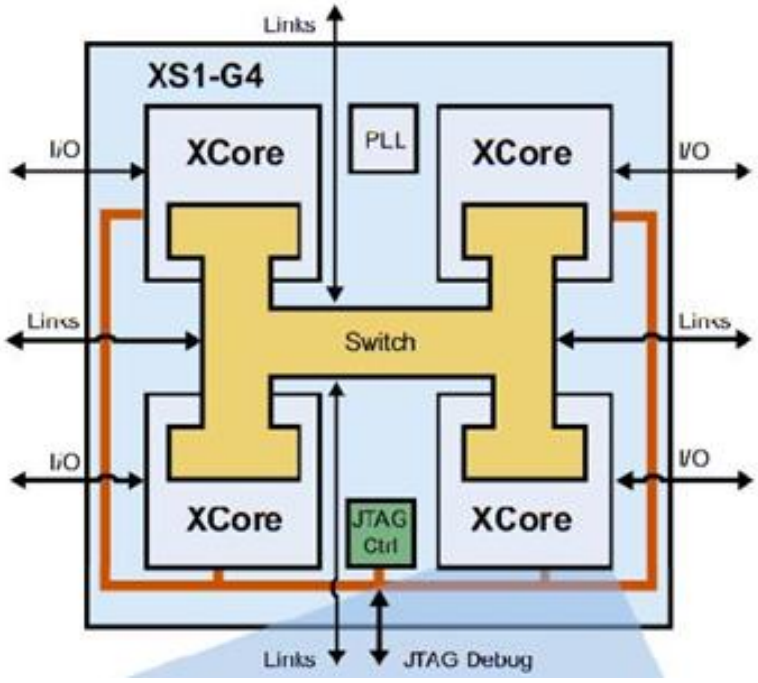


Рис. 2.2.1. Блок-схема 4-х ядерного МК

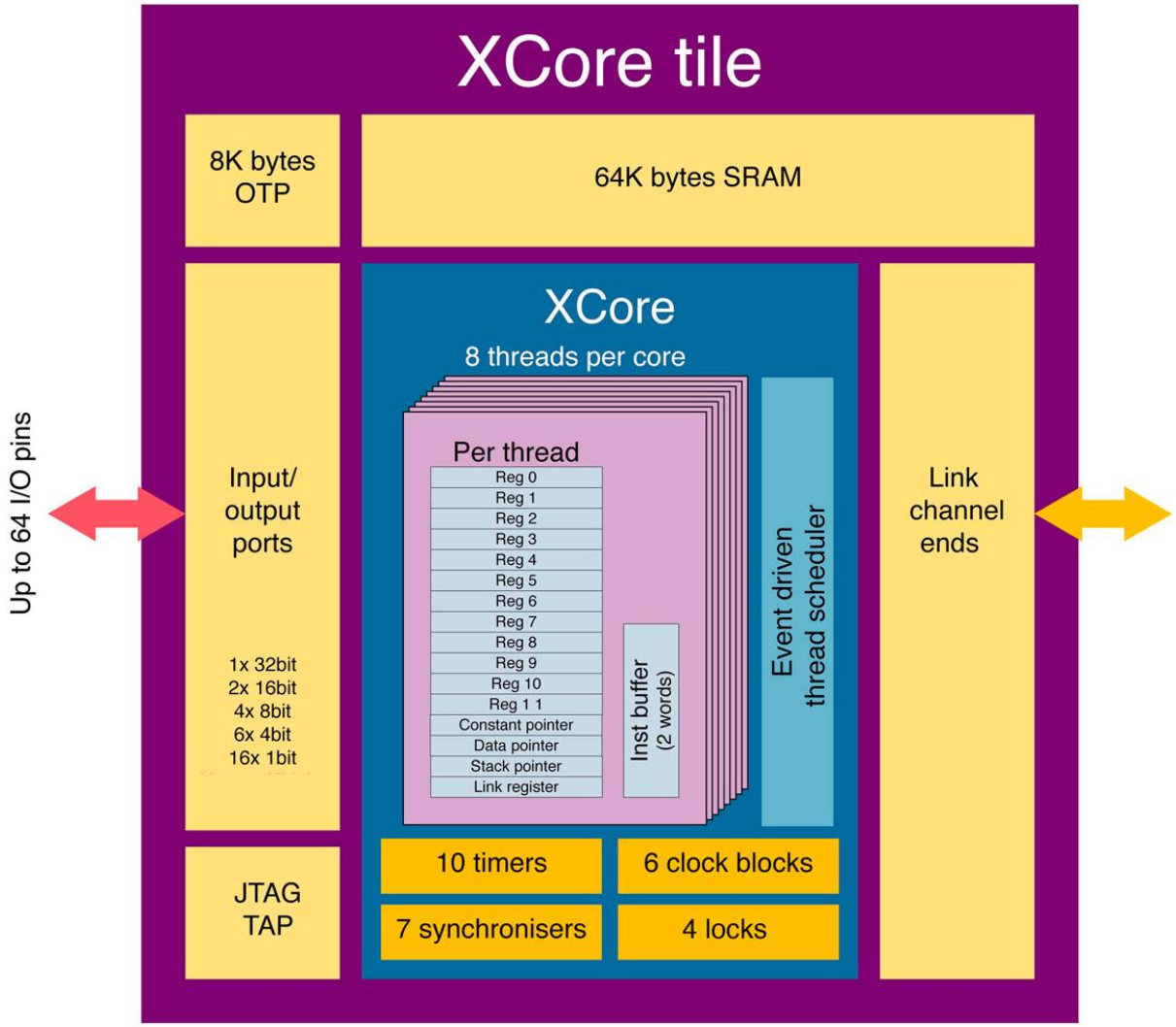


Рис.2.2.2. Архитектура многопоточного ядра xCore

2.2.1. Параллельные потоки

Каждое ядро xCore имеет аппаратную поддержку выполнения некоторого числа параллельных потоков (рис.2.2.2):

- набор основных регистров для каждого потока;
- в планировщик потоков, который динамически выбирает поток для выполнения;
- набор синхронизаторов для синхронизации потоков;
- набор каналов для связи с другими потоками;
- набор портов для ввода и вывода;
- набор таймеров для управления потоками в режиме реального времени;

- набор тактовых генераторов для обеспечения синхронизации ввода-вывода с внешний средой.

Для управления потоками есть команды инициализации, завершения, старта, синхронизации и остановки потоков; также есть команды обеспечивающие ввод-вывод и связь между потоками.

Набор потоков на каждом ядре xCore может быть задействован:

- для реализации контроллера, где ввод-вывод выполняется одновременно с выполнением прикладных программным;
- для обеспечения коммуникации ядер (потоков) или ввода-вывода совместно с текущей обработкой;
- для того, чтобы скрыть задержки интерконнекта, позволяя одним потокам продолжить работу, в то время как другие находятся в ожидании связи с удаленными ядрами XCore.

В системе команд МК есть инструкции, которые позволяют потокам общаться между собой и выполнять вход и выход. Эти инструкции:

- обеспечивают коммуникации, управляемые событиями, и ввод-вывод с ожиданием автоматически распланированных потоков;
- потоковую поддержку пакетированной или синхронизированной связи между потоками в любом месте системы;
- предоставляют процессору возможность перехода в режим «idle» с отключением тактирования, при том условии, что все потоки находятся в режиме ожидания и, тем самым, экономя энергопотребление;
- разрешают конвейеризацию коммуникаций и буферизацию ввода-вывода.

2.2.2. Особенности системы команд

Главные особенности системы команд xCore заключаются в следующем.

- Короткие команды обеспечивают эффективный доступ к стеку и к другим областям данных, а также обеспечивают эффективное ветвление и вызов

подпрограмм. Основанием для выбора коротких команд были экстенсивные оценки, опирающиеся на требования современных компиляторов.

- Адресация памяти байтовая, но доступ должен быть выровнен по естественным границам. Например, для 32-разрядных слов два младших значащих бита адреса должны быть нулями.

2.2.3. Модель многоядерного программирования

В устройствах xCore программы состоят из множества задач, идущих параллельно. Конкурирующая параллельная задача управляет своим собственным состоянием и ресурсами и взаимодействует посредством связей с другими. Задачи определяются также, как и любые функции в Си, например:

```
void task1(int x, int a[20]) { ...  }
```

Никаких специальных ключевых слов не требуется и, в общем-то, любая функция может быть представлена как задача. Задачи могут иметь любые аргументы, но, как правило, не имеют возвращаемой величины. Чаще всего задачи не возвращают вообще ничего и содержат бесконечный цикл:

```
void task1(args) {
  ... initialization ...
  while (1) {
    ... main loop ...
  }
}
```

Задачи планируются к выполнению в функции *main* программы посредством конструкции *par* :

```
int main(void) {
 par {
  task1( args );
  task2( args );
  task3( args );
  }
}
```

Здесь для конфигурации задач можно ввести аргументы. Каждая задача будет выполняться параллельно через логическое ядро xCore устройства. Компилятор автоматически проверяет какое количество ядер на плите

использовано и выдаст сообщение об ошибке, если слишком много. Также автоматически компилятор устанавливает необходимое количество стеков и размер памяти для каждой задачи, и сообщает сколько в целом памяти использовано.

Задачи взаимодействуют через операции определенных интерфейсов. Интерфейс предоставляет здесь наиболее структурированный и гибкий метод межзадачного взаимодействия. Интерфейс определяет вид взаимодействия, который может происходить между задачами, и информацию, которая идет с ними. Например, следующее объявление интерфейса определяет два типа взаимодействия:

```
interface my_interface {
  void fA(int x, int y);
  void fB(float x);
};
```

Типы взаимодействия определены как Си функции. Функции интерфейса могут принимать любые значения, которые могут принимать Си функции. Аргументы определяют: какая информация будет послана, когда произойдет взаимодействие между задачами. Если аргумент типа массив задан для связи внутри плиты, то массив не копируется, а компилятор просто устанавливает указатель на массив, предоставляющий приемнику доступ к массиву в памяти.

Связь выполняется на ненаправленном соединении, которое подчиняется протоколу интерфейса. Один конец соединения объявляется клиентом, и он отправляет сообщения, а другой конец-сервером, который, в свою очередь, принимает эти сообщения. Для посылки сообщения, задача может иметь аргумент, который будет клиентским концом интерфейсового соединенияи и использовать его различно для посылки сообщений:

```
void task1(interface my_interface client c)
{
  // c is the client end of the connection,
  // let's communicate with the other end.
  c.fA(5, 10);
}
```

Код может ожидать взаимодействия на серверном конце с помощью концепции *select*. *Select* ждет пока взаимодействие не будет с инициировано с другого конца:

```
void task2(interface my_interface server c)
{
  // wait for either fA or fB over connection c.
  select {
  case c.fA(int x, int y):
    printf("Received fA: %d, %d\n", x, y);
    break;
  case c.fB(float x):
    // handle the message
    printf("Received fB: %f\n", x);
    break;
  }
}
```

Отметим, как *select* позволяет нам обрабатывать несколько различных типов сообщений. Языковое расширение также позволяет выделять локальные переменные для входной информации, которую может принять обработчик сообщений, используя различные типы интерфейсов и взаимодействий. Как только одно из взаимодействий будет инициализировано и *select* обработает событие, код продолжится.

Выполнение задач можно сложить вместе объявлением экземпляра интерфейса и установкой его аргументом обеих задач:

```
int main(void)
{
  interface my_interface c;
  par {
    task1(c);
    task2(c);
  }
  return 0;
}
```

Типы функций указывают устройству что есть - сервер, а что - клиент. Компилятор также проверяет, чтобы каждая связь точно получила один серверный и один клиентский конец.

2.2.4. Плиты, расположение задач и совместная многозадачность

Устройства xCore делятся на плиты – самостоятельные МК на кристалле, которые могут сообща использовать память и порты. В свою очередь, каждая плита делится на несколько логических ядер. Когда вы в функции *main* используете выражение *par*, каждая задача может быть размещена в свою плиту, например:

```
int main(void) {
  par {
    on tile[0]: task1( args );
    on tile[0]: task2( args );
    on tile[1]: task3( args );
  }
}
```

2.2.5. Защита памяти

Во время написания параллельных приложений вопрос о безопасности обращения к памяти становится очень важным. Многоядерное xMOS-расширение для Си помогает решить это следующим образом:

1. Доступ ко всем массивам и указателям ограничен.
2. По умолчанию нет доступа к выполнению параллельных задач для одного и того же объема памяти, на одной и той же плите.

Проверка массивов памяти и указателей означает то, что вы можете написать обычный код на Си для доступа к памяти:

```
void double_and_copy(int a[], int *p, int n)
 {
  for (int i = 0; i < n; i++) {
    a[i] = 2*p[i];
  }
}
```

Каждый массив и каждый указатель связаны с диапазоном памяти, к которому разрешен доступ. Если код доступа вне этого диапазона, то это вызовет прерывание. Кроме того, прерывание вызовут переполнение буфера, другие классы трудно отслеживаемых дефектов кода, которые обусловливают состояние гонки.

Состоя́ние го́нки (англ. *race condition*) — ошибка проектирования многопоточной системы или приложения, при которой работа системы или приложения зависит от того, в каком порядке выполняются части кода. Состояние гонки — «плавающая» ошибка, проявляющаяся в случайные моменты времени и «пропадающая» при попытке её локализовать.

Для того, чтобы избежать случайных состояний гонки не следует получать непосредственный доступ к одной и той же глобальной переменной в двух разных задачах. Например, следующий код вызовет ошибку во время компиляции:

```
int g;
void task1() {
  ...
  g = 5;
  ...
}
void task2() {
  ...
  x = g;
  ...
}

int main() {
  par {
    task1();
    task2();
  }
}
```

2.2.6. Разделение информации между задачами.

Несмотря на то, что неуправляемое разделение информации между задачами не разрешается компилятором, вы, тем не менее, можете разделять информацию с помощью различных механизмов:

- с помощью специальных указателей - подвижных или небезопасных, которые позволяют устанавливать указатели между задачами с различной степенью безопасности обращения к памяти;
- с помощью распространяющих задач, которые позволяют задачам делиться локальными переменными с множеством других задач через

взаимодействие, позволяя компилятору реализовывать это как закрытый защищенный доступ, обеспечивая доступ каждой из задач.

2.2.7. Доступ к аппаратным таймерам и портам

Аппаратура xCore имеет 32-битный 100 МГц счетчик опорной тактовой частоты, который может быть использован в вашем проекте. Первый способ получить доступ к таймерам - это вызов delay_ фунции в “timer.h” . Но это позволяет лишь сделать задержку на определенный интервал времени (на протяжении которого логическое ядро, на котором выполняется код, отключится).

Для более гибкого использования таймеров вы можете объявить переменную типа *timer*. Чтение точного времени с таймера происходит с помощью специального оператора:

```
timer tmr;
unsigned t;
...
// Read the current time
tmr :> t;
```

Таймеры могут использоваться для инициализации задач так же, как это делает интерфейс связи, с помощью конструкции *select*:

```
// Wait for a transaction with another task *or* a timeout.
tmr :> t;
select {
 case i.fA(int x):
   // handle message from other task
   ...
   break;
 case tmr when timerafter(t + 1000) :> int x:
    // 100us have passed
   ...
   break;
}
```

Таймеры могут быть также использованы в программном коде для периодических задач:

```
void task1() {
  timer tmr;
  unsigned t;
```

```
  tmr :> t;
  // Do some work every 2ms
  for (;;) {
    select {
    case tmr when timerafter(t) :> int x:
      // Do the processing
      ...
      t = t + 200000;
      break;
    }
  }
}
```

Вы можете произвольно смешивать различные тайм-ауты и события взаимодействия задач в *select* ,чтобы управлять разными событиями в одной задаче.

Получить доступ к устройствам вввода-вывода и к портам блока можно с помощью объявления переменной типа *port*. Ввод и вывод порта выполняется с использованием специальных операторов ввода-вывода:

```
port p;
unsigned x;

// input from a port
p :> x;

// output to a port.
p <: x;
```

Порт ввода может также использоваться для инициации событий в задаче, добавлением оператора case в оператор select:

```
select {
  ...
  // trigger an event when the pins have a
  // particular value.
  case p when pinseq(1) :> int x:
    // handle port event ...
    break;
}
```

В комплекте с «xMOScomposer» поставляются обширные библиотеки для работы с аппартными портами, включая «десериализацию» (преобразование

последовательного кода в параллельный), «сериализацию» (параллельного в последовательный) и квитирование для того, чтобы работать с быстрыми протоколами ввода-вывода.

2.2.8. Программные средства разработки

Программное обеспечение разработки приложений, его «окружение» обеспечивают поддержку аппаратных средств МК xCore и делают достаточно простой реализацию задач реального времени в виде параллельной системы. Среда разработки «*xTIMEComposerStudio*» полностью согласована со стандартами компиляторов для Си и C++, имеет стандартные языковые библиотеки, интегрированную среду разработки (IDE), программный симулятор, отладчик, инструментарий для среды исполнения программы и трассировки, а также временной анализатор статического кода (XTA). Помогают разработчикам приложений пользоваться возможностями аппаратных средств МК простые, но в тоже время мощные многоядерные языковые расширения для Си. Эти расширения, называемые xC, содержат средства для построения решений, основанных на параллелизме и связи ядер (потоков), точном выборе времени входа – выхода и безопасного управления памятью.

2.2.9. Разработка приложений в XDE (xMOS Development Enviroment)

Среда разработки *xTIMEcomposer Studio* реализована на платформе *Eclipce* и предоставляет много разнообразных и полезных ресурсов разработчику.

2.2.9.1. Импорт и экспорт проекта в xTIMEcomposer Studio

xTIMEcomposer Studio позволяет легко обмениваться проектами с другими разработчиками через импорт и экспорт проектов.

Импорт проекта: Чтобы импортировать проект выполните следующие действия:

- Выберите **File -> Import.**
- Дважды щелкните на **General option** и выберите **Existing Projects** в **Workspace,** затем нажмите кнопку **Next.**
- В диалоговом окне **Import** нажмите кнопку **Browse.**

- Выберите архив для импорта и нажмите кнопку **Open.**
- Нажмите **Finish.**

Экспорт проекта: Чтобы экспортировать проект нужно выполнить следующее:

- Выберите **File -> Export.**
- Дважды щелкните на **General option**, выберите **Archive File** и нажмите кнопку **Next.**
- Выберите проекты, которые вы хотите экспортировать.
- Введите имя архива в текстовом поле **To archive file**
- Нажмите **Finish.**

2.2.9.2. *Разработка приложений с использованием xSOFTip*

XMOS предоставляет библиотеку проверенных *xSOFTip* блоков, которые содержат интерфейсы, такие как USB, Ethernet, последовательные порты, а также DSP и протокольные функции. Для просмотра доступных блоков из библиотеки *xSOFTip*, используется *xSOFTip Explorer* (рис.2.2.3), графический инструмент, предоставляющий единое место, чтобы просмотреть, выбрать и настроить библиотеку на конкретную задачу с помощью *xSOFTip* блоков, позволяя сконцентрироваться на приложении. Она доступна как отдельный инструмент, а также интегрирована в *xTIMEcomposer Studio*.

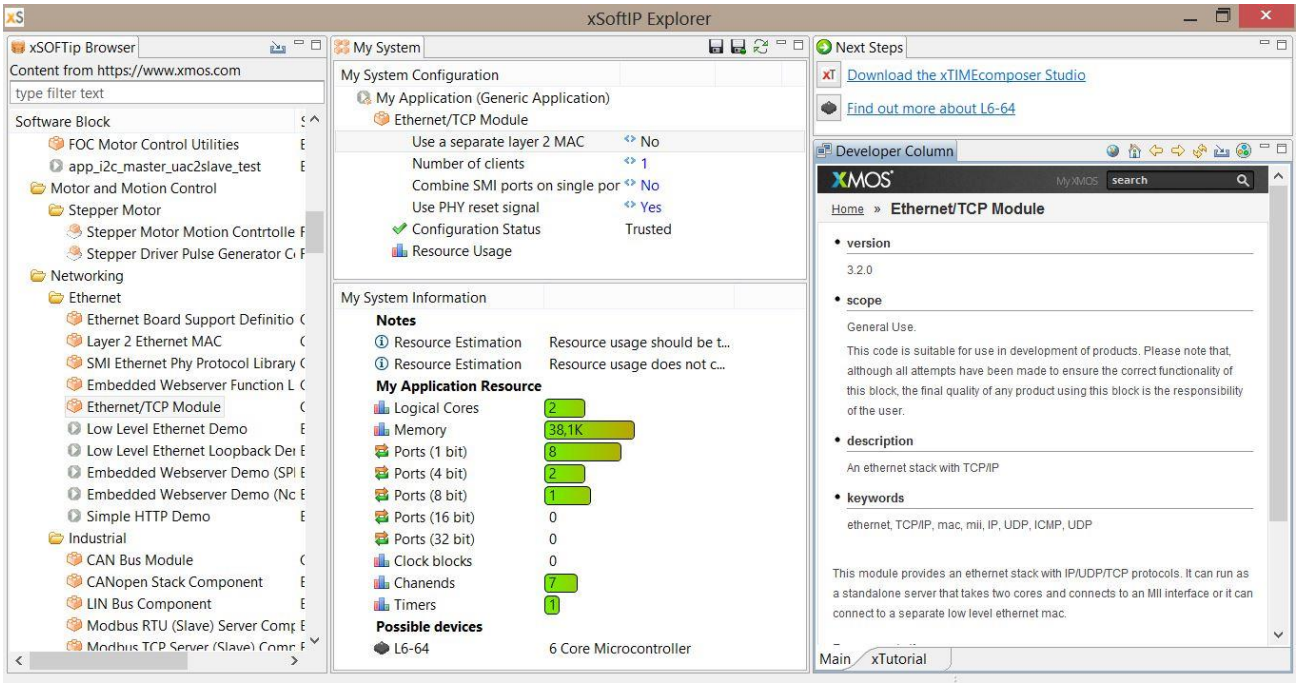


Рис.2.2.3. Окно *xSoftiP Explorer*

xSOFTip Explorer имеет четыре окна:

- *xSOFTip Browser* - показывает компоненты *xSOFTip,* которые можно выбрать для проекта.
- *My System Configuration* – показывает выбранные компоненты.
- *System Information* – показывает ресурсы (память, логические ядра, порты, таймеры-счетчики), выбранных *xSOFTip* компонентов и многоядерные микроконтроллеры XMOS которые подходят для создаваемого приложения.
- *Developer Column* - online - документация о *xSOFTip*, инструментах и xCore многоядерных микроконтроллерах.

Каждая компонента *xSOFTip* имеет область, которая показывает статус *xSOFTip* компоненты:

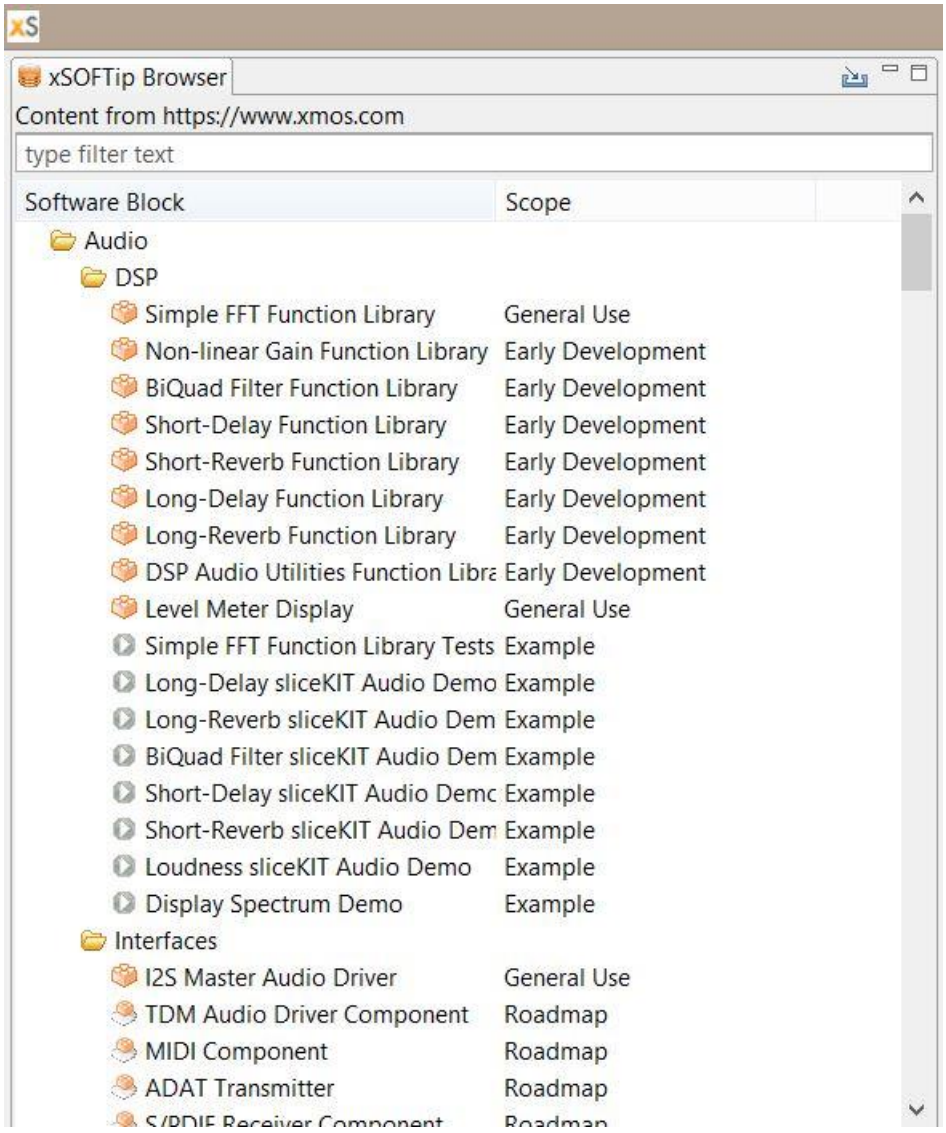


Рис.2.2.4. *xSoftip Browser*

General Use: Полная информация, ресурс доступен. Все было сделано, чтобы обеспечить правильную функциональность этого блока.

Early Development: *xSOFTip* подходит для использования в разработке продуктов и является полностью функциональной. Тем не менее, необходимо позаботиться о проверке продукта с помощью этого блока программного обеспечения. Информация о ресурсах доступна.

Experimental: x*SOFTip* находится на экспериментальной стадии. Код существует, но нет в комплекте. Информация о ресурсах может быть недоступна.

Roadmap: *xSOFTip* находится на стадии развития XMOS. Информационный ресурс существует для этого *xSOFTip*, но код не доступен.

Open Source Community: *xSOFTip* была разработана сообществом Open Source. Информация о ресурсах может быть недоступна.

При выборе компоненты в окне *xSOFTip Browser* информация о ней отображается в колонке *Developer Column*, *с описанием того,* что она делает, ее особенности и какие xKIT средства разработки подходят для использования с этим *xSOFTip.* Дополнительная информация об отдельных параметрах и конфигурациях может быть отображена в *Developer Column*.

Добавление *xSOFTip* компоненты в проект.

Чтобы добавить *xSOFTip* компоненту в проект нужно выполнить следующие действия:

1. перетащите *xSOFTip* компоненту в *System Configuration* или дважды щелкните по компоненте в *xSOFTip Browser* (Рис.2.2.5.).
2. Выберите версию компоненты, которую необходимо импортировать, все ранее выпущенные версии должны отображаться. Самая последняя версия отображается по умолчанию.
3. Выберите проект, в который нужно добавить ***xSOFTip*** компоненту.
4. Нажмите **Finish.**

По мере добавления *xSOFTip* компонентов в *System Configuration*, окно *System Information* (рис.2.2.6.) показывает совокупное число ресурсов, выбранных пользователем.

Logical Cores: 32-bit ядра микроконтроллера. XMOS многоядерные микроконтроллеры включают 4, 6, 8, 10, 12 и 16-ядерные устройства.

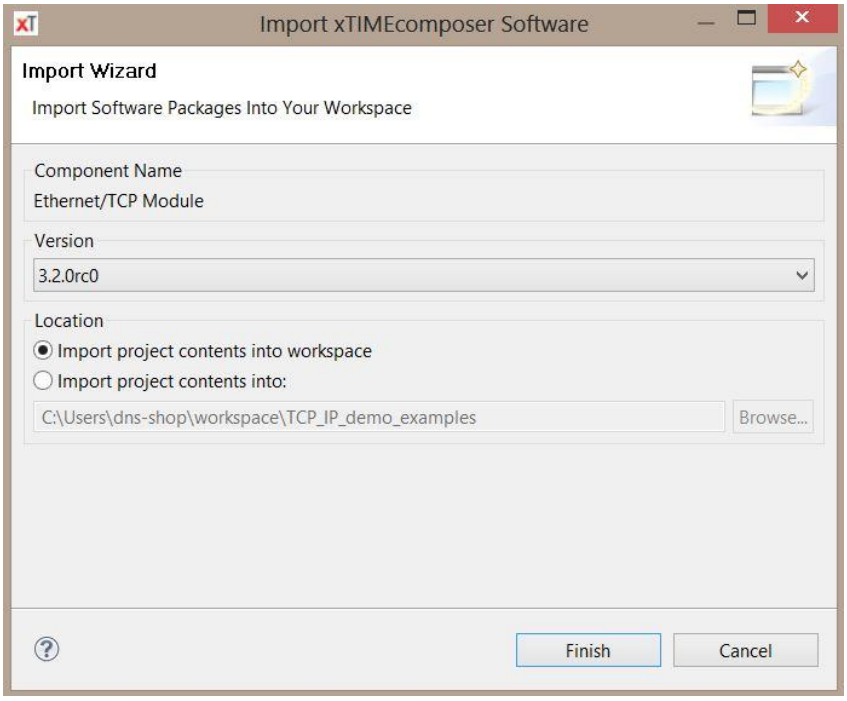


Рис.2.2.5. Окно *Import Wizard*

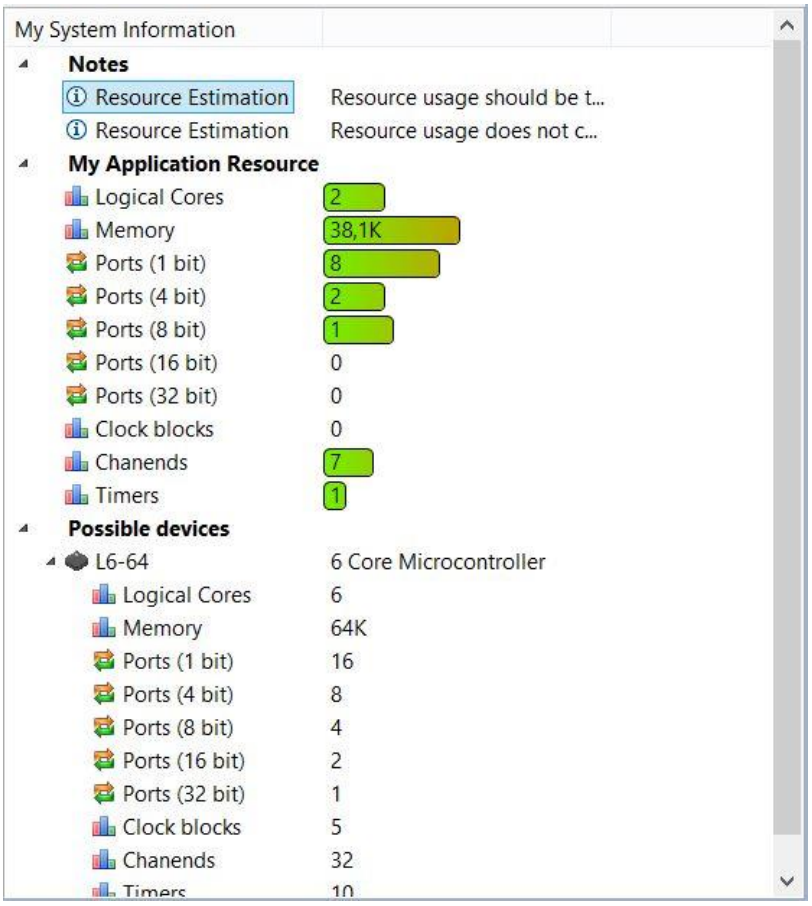


Рис.2.2.6. Окно System Information

Ports - порты ввода/вывода XMOS микроконтроллеров, подключенные к контактам; позволяют отправлять и получать данные через «пины» с очень низкой задержкой. Порты доступны различной ширины (разрядности): 1-битовый порт ввода / вывода подключен к одному контакту чипа, 4-битный порт подключен к четырем контактам.

Clock Blocks - используются для синхронизации ввода / вывода.

Chanends - концы канала - являются частью системы XConnect, позволяя ядрам отправлять сообщения друг другу через низкую латентность каналов XConnect.

Timers - таймеры - используются программным обеспечением для управления временем исполнения кода и синхронизации. Таймеры запускаются от тактирующей последовательности МК 100 МГц, что дает дискретность установки 10 нс.

Перечень возможных устройств отображается в нижней части окна *System Information***.** Показывает, какие типы многоядерных микроконтроллеров xCore предназначены для выбранных *xSOFTip* компонентов.

Некоторые компоненты имеют настраиваемые параметры, которые можно изменить, после того, как они будут добавлены в окно *System Configuration* (рис.2.2.7).

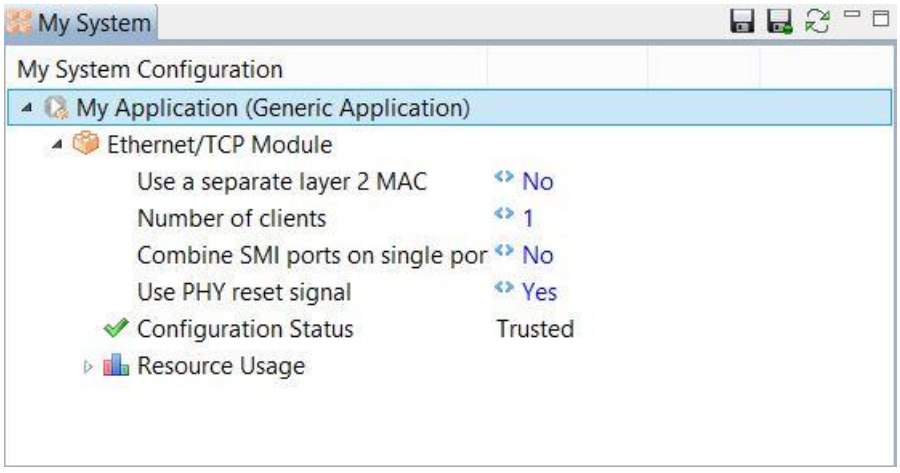


Рис.2.2.7. Окно *System Configuration*

После изменения конфигурации использование ресурсов обновляется в окне *System Information***.**

2.2.9.3. *Создание проекта из прилагаемых компонентов*

Среда разработчика позволяет автоматически создать проект из компонентов в окне *System Configuration.*

1. Нажимаем **Generate Project,** кнопка в верхней части ***System Configuration.***
2. Вводим имя для проекта в окне **Generate Project.**
3. Выбираем плату из списка оборудования.
4. Нажимаем **Finish.**

xTIMEcomposer Studio создает проект с выбранными *xSOFTip* компонентами. *xSOFTip* компоненты написаны на Си, их можно корректировать под свои конкретные требования или добавить собственные функции на Си.

2.2.9.4. Компиляция проекта

Чтобы построить проект выбираем **Project -> Build Project,** далее выбираем **Debug** или **Release.** В созданном проекте для определения конфигурации настроек, используемых компилятором, *xTIMEcomposer Studio* использует ***Makefile.***.

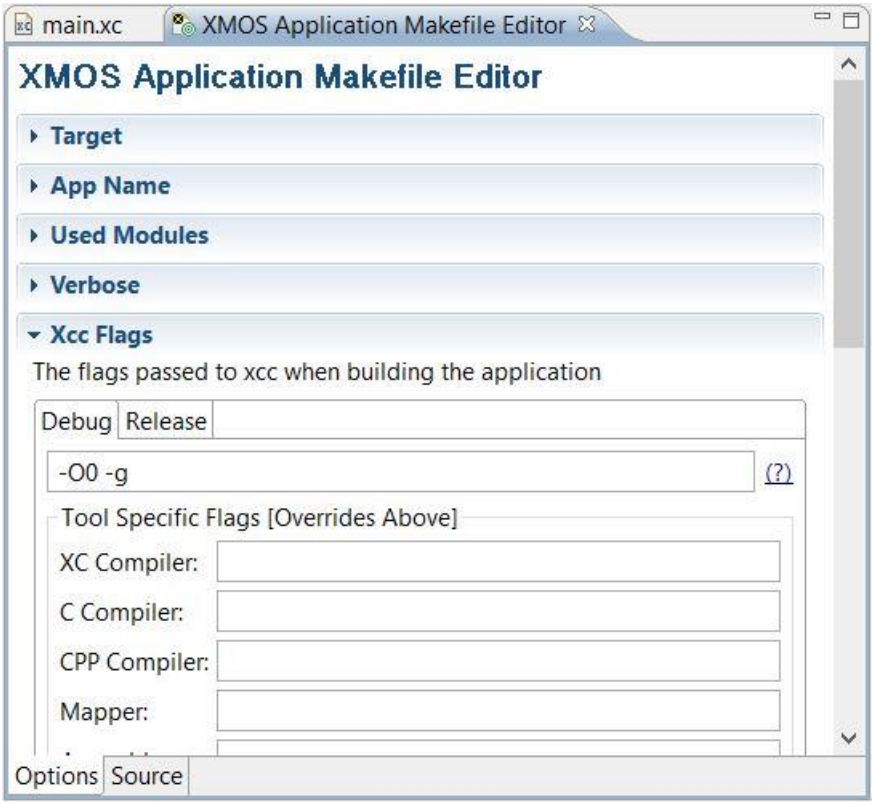


Рис.2.2.8. Редактор Makefile

Дважды щелкнув по ***Makefile*** проекта ***в Project Explorer***, откроем его редактор (рис.2.2.8.), в котором можно задать параметры компилятора
Если нет ошибок в программе, то после компиляции *xTIMEcomposer Studio* добавляет скомпилированный двоичный файл ***Binaries*** в папку проекта. Ошибки выводятся на консоль и подсвечиваются красным цветом; если дважды щелкнуть мышкой по сообщению об ошибке, то ее местоположение откроется в редакторе. Компилятор *xTIMEcomposer Studio* может управляться директивами *# pragma* для кодов написанных на Си и XC.

2.2.10. Временной анализ

Инструмент *xCore Timing Analyzer (XTA)* позволяет определить время необходимое для выполнения кода на целевой платформе. В связи с детерминированной природой архитектуры *xCore* инструментарий *XTA* позволяет оценить наибольший и наименьший интервал времени необходимый для выполнения фрагментов кода. В сочетании с указанными пользователем требованиями инструмент может во время компиляции определить: все ли критичные по времени исполнения участки кода будут гарантированно выполнены в пределах заданных временных интервалов. Для загрузки программы под управлением *XTA* нужно выполнить следующие действия:

1. Выбрать проект в ***Project Explorer.***
2. Выбрать **Run -> Time Configurations.**
3. В левой части панели дважды щелкнуть **XCore Application**; *xTIMEcomposer Studio* создаст новую конфигурацию и отобразит параметры по умолчанию в правой части панели.
4. В текстовом поле **Name** нужно ввести имя для конфигурации.
5. Чтобы сохранить конфигурацию и запустить анализатор *XTA* - нажать **Time**.

xTIMEcomposer Studio загружает программу в анализатор синхронизации и открывает его в *Timing perspective* (рис.2.2.9.).

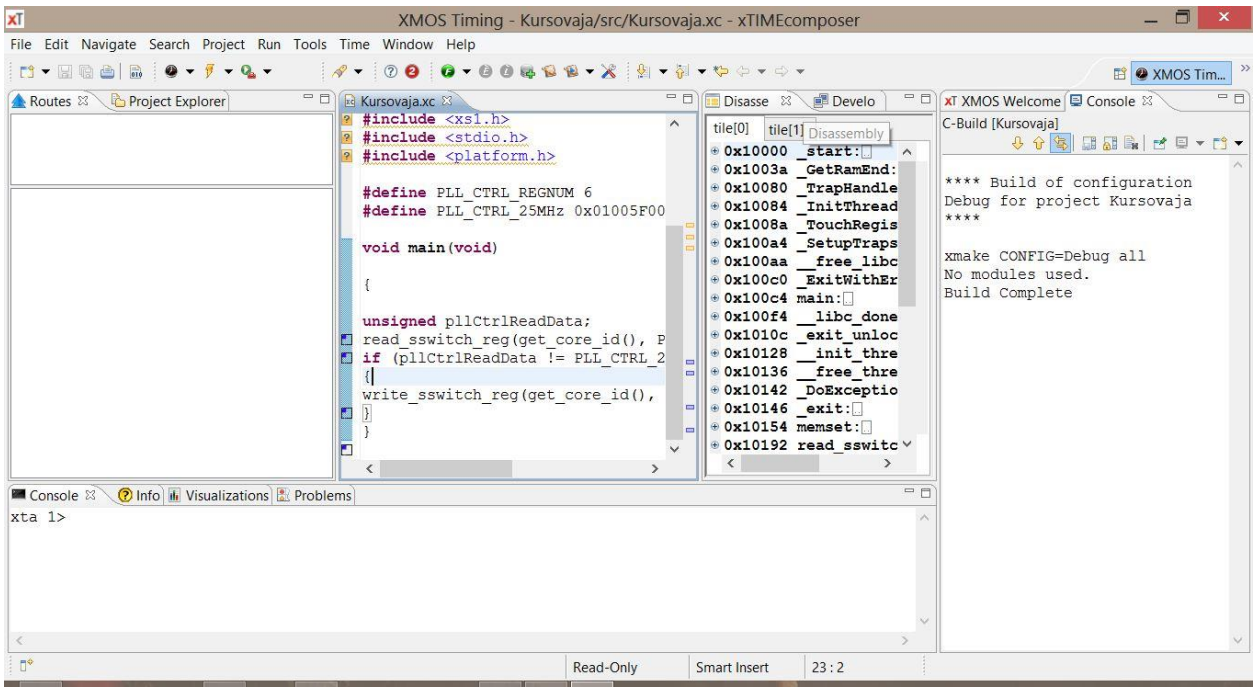


Рис.2.2.9.

Чтобы указать маршрут и проанализировать его, нужно выполнить следующие действия:

1. Щелкнуть правой кнопкой мыши по конечной точке маркера в поле редактора и выбрать **Set from endpoint.** В правой верхней четверти маркера *xTIMEcomposer Studio* отобразит зеленую точку.
2. Щелкнуть правой кнопкой мыши на конечной точке маркера и выбрать **Set to endpoint.** *xTIMEcomposer Studio* отобразит красную точку в нижней правой четверти маркера.
3. Нажать **Analyze Endpoints -** кнопка на главной панели инструментов. *xTIMEcomposer Studio* анализирует все пути по указанному маршруту, который отображается в нижней панели **Routes.**
4. Чтобы проанализировать время необходимое для выполнения функций, нажать **Analyze Function (**кнопка на главной панели инструментов) и выбрать функцию из раскрывшегося списка.

Routes отображает структуру маршрута. Каждый раз, при анализе маршрута в верхней панели добавляется запись. Нажатием на маршрут

можно просмотреть его в нижней панели. Он представлен в виде следующих узлов:

- Функция исходного кода.
- Список узлов, которые выполняются в определенной последовательности.
- Набор узлов, которые выполняются условно.
- Цикл, состоящий из последовательности узлов, в которых последний узел может перейти обратно к первому узлу.
- Блок, содержащий последовательность линейных команд.
- Одиночная машинная команда.

Visualizations обеспечивает графическое представление маршрута**.** Вкладка **Structure** представляет собой маршрут в виде линии, которая идет слева направо (рис.2.2.10). Маршрут разветвляется на несколько путей, и все пути соединяются в конце. В лучшем случае путь отображается зеленым цветом, в худшем случае - красным, а все другие пути серым цветом.

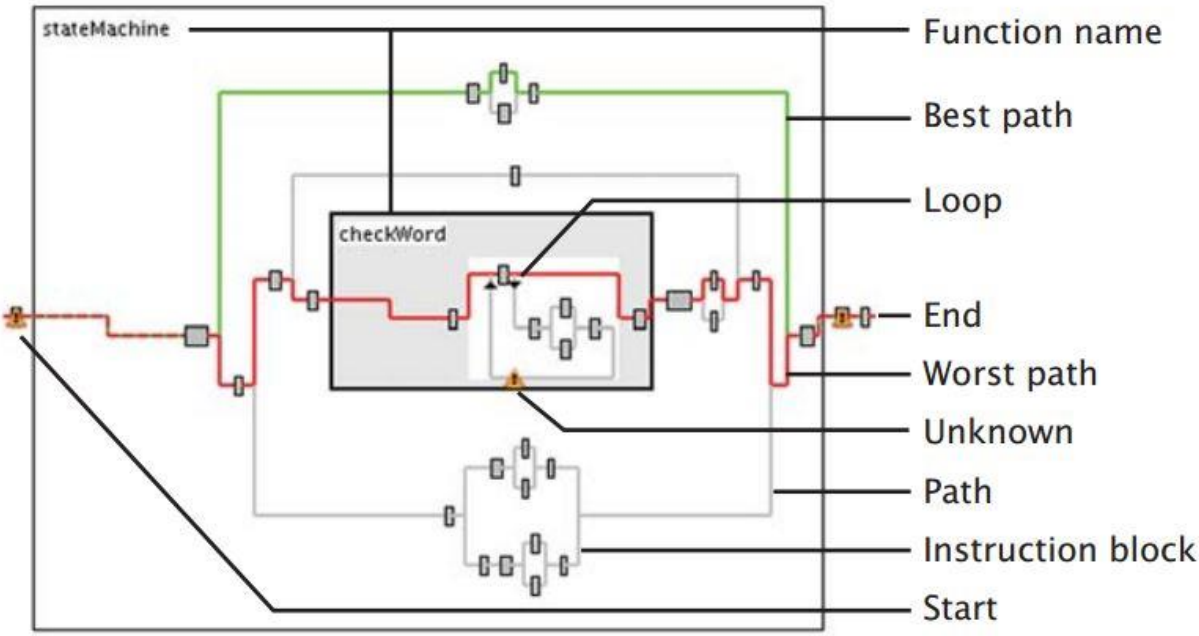


Рис.2.2.10. Маршрут

Наведением курсора на узел можно инициировать краткий отчет о его временных свойствах. При некоторых условиях анализатор не сможет показать время без дополнительной информации. Примеры этих условий:

- Маршрут содержит команду ввода / вывода, которая может быть приостановлена на неопределенный отрезок времени.
- Маршрут содержит цикл с условием выхода в зависимости от данных.
- Путь реализуется только при возникновении ошибки и поэтому время не является критичным.

В этих случаях временному анализатору можно предоставить дополнительную информацию о выполнении программы. Благодаря этой дополнительной информации анализатор может показать, что требование к заданному времени маршрута удовлетворяется. Информация, которую можно предоставить включает в себя:

- **Число итераций цикла;** здесь для отображения диалогового окна нужно щелкнуть правой кнопкой мыши на узле цикла и выбрать **Set loop iterations**, ввести максимальное число циклов и нажать **ОК.**
- **Максимальное время паузы для команды ввода/вывода**; здесь требуется щелкнуть правой кнопкой мыши по узлу команд и выбрать **Set instruction time** для отображения диалогового окна, затем ввести значение и выбрать единицы измерения времени/скорости (например, наносекунды или МГц) и нажать **ОК.**
- **Исключение пути из маршрута -** щелкнуть правой кнопкой мыши на узле и выбрать **Exclude.**

По умолчанию анализатор времени предполагает, что маршрут всегда следует по той ветви, которая принимает наибольшее время для выполнения. Если вы знаете, что это не так, например, путем просмотра во время моделирования или формального анализа программы, то можно уточнить параметры, используемые анализатором. Уточнения можно сделать включив:

- **определение абсолютного времени для выполнения вызова функции -** щелкните правой кнопкой мыши по функции узла и выберите **Set function time,** откроется диалоговое окно, введите время и нажмите **ОК**;
- **определение абсолютного времени для пути -** выберите путь, затем, нажав на два узла команд, щелкните правой кнопкой мыши и выберите **Set path time**; откроется диалоговое окно, введите время и нажмите **ОК**;
- **определение условного узла циклов:** по умолчанию анализатор предполагает, что условный узел всегда идет по пути, который занимает наибольшее время для выполнения. Чтобы указать число циклов выполнения целевого узла, щелкните правой кнопкой мыши на целевом узле и выберите **Set loop path,** откроется диалоговое окно; введите число итераций и нажмите **ОК**.

После того, как будут указаны временные требования к программе можно генерировать скрипт, который проверяет эти требования во время компиляции. Чтобы создать скрипт, который проверяет все временные требования, указанные в **Routes,** нужно выполнить следующие действия:

1. Нажать кнопку **Generate Script**.
2. В текстовом окне **Script location** ввести имя файла для скрипта. Имя файла должно иметь расширение **.xta.**
3. Чтобы изменить имена директив, добавленных в исходный файл, нужно поменять их значение в поле **Pragma name.**
4. Нажать кнопку **ОК** для сохранения скрипта и обновления своего исходного кода. *xTIMEcomposer Studio* добавляет скрипт в проект и открывает его в редакторе.

При компиляции программы требования к анализу времени исполнения кода проверяются, и любые ошибки выводятся, как ошибки компиляции. Дважды щелкнув по ошибке можно получить информацию о некорректных командах в скрипте.

2.2.11. Подготовка проекта

Проект может содержать исходные Си-файлы с включением (или без включения) многоядерного расширения. Для использования этого расширения нужно вместо вашего исходного файла с расширением *.c использовать файл с расширением *.xc. Компилятор среды *xTIMEcomposerStudio* автоматически установит расширение этого файла. Таким образом, приложение будет содержать код сочетающий xC и Си.

Примеры:

1. *Hello World.*

Начнем с привычной программы „hello world”:

```
#include <stdio.h>
int main() {
  printf("Hello World\n");
  return 0;
}
```

Видим, что все точно так же, как на Си. Отметим только, что вывод на консоль "Hello World" возможен только при подключенном адаптере отладки.

2. *Мигающие LED.*

Это код, реализует управление светодиодами на плате XK-1A:

```
#include <xs1.h>
#include <timer.h>

port p = XS1_PORT_1A;

int main() {
  while (1) {
    p <: 0;
    delay_milliseconds(200);
    p <: 1;
    delay_milliseconds(200);
  }
}
```

Этот пример использует тип *port,* который является частью многоядерного расширения xC. Этот тип инициализирует здесь блок портов *1A*. Взаимное соответствие между портом и внешними выводами даются в таблице данных для конкретного xMOS устройства. Оператор "<: " передает значение в порт. В примере также использована функция *delay_milliseconds*, которая находится в библиотеках, прилагаемых к с устройствам.

При компиляции проекта устройство отслеживает ресурсы, которые вы используете. Например, если скомпилировать эту программу с опцией *report* то получится следующий вывод:

```
Constraint check for "tile[0]" (node "0", tile 0):
Cores available:        8,   used:        1 .  OKAY
Timers available:       10,  used:        1 .  OKAY
Chanends available:     32,  used:        0 .  OKAY
Memory available:     65536, used:     1176 .  OKAY
(Stack: 336, Code: 720, Data: 120)
Constraints checks PASSED.
```

Здесь также можно видеть, что компилятор указывает на то, сколько *точно* используется памяти. Многоядерное расширение разработано таким образом, чтобы всегда можно было видеть эту информацию, даже если параллельно запущена задача, использующая тот же объем памяти.

3. *Мигающие LED с настройкой режима.*

Теперь у нас имеется достаточно информации, чтобы написать простое многоядерное приложение. В этом приложении будет две задачи: одна заставляет светодиоды мигать, другая изменяет периодичность этого мигания.

В первую очередь рассмотрим задачу "flasher", которая, используя периодические события формируемые таймером, отвечает через определенный интерфейс другой задаче, предназначенной для изменения периода мигания:

```
#include <xs1.h>
#include <timer.h>
#include <stdio.h>
```

```
interface flasher_if {
  void set_period(unsigned period);
};

void flasher(port p, server interface flasher_if i)
{
  int val = 0;
  timer tmr;
  int t;
  unsigned period = 1000;
  tmr :> t;
  while (1) {
    select {
    case tmr when timerafter(t + period) :> void:
      p <: val;
      val = ~val;
      t += period;
      break;
    case i.set_period(unsigned new_period):
      period = new_period;
      break;
    }
  }
}
```

Вторая задача будет использовать соединение с предыдущей, чтобы изменять период мигания:

```
void myprog(client interface flasher_if i) {
  while (1) {
    printf("Setting period to 1000 timer ticks");
    i.set_period(1000);
    delay_seconds(5);
    printf("Setting period to 5000 timer ticks");
    i.set_period(5000);
    delay_seconds(5);
  }
}
```

А последняя запускает обе задачи параллельно, используя выражение *par*:

```
port p = XS1_PORT_1A;

int main() {
  interface flasher_if i_flasher;
  par {
    flasher(p, i_flasher);
    myprog(i_flasher);
  }
  return 0;
}
```

2.2.12. Построение проекта и запуск программы; прошивка платы

Чтобы проверить работоспособность программы, пронаблюдать её действие на плате xk-1A, необходимо сначала создать проект. Для это, открыв xTIMEcomposer Studio создаем новый xTIMEcomposer проект.

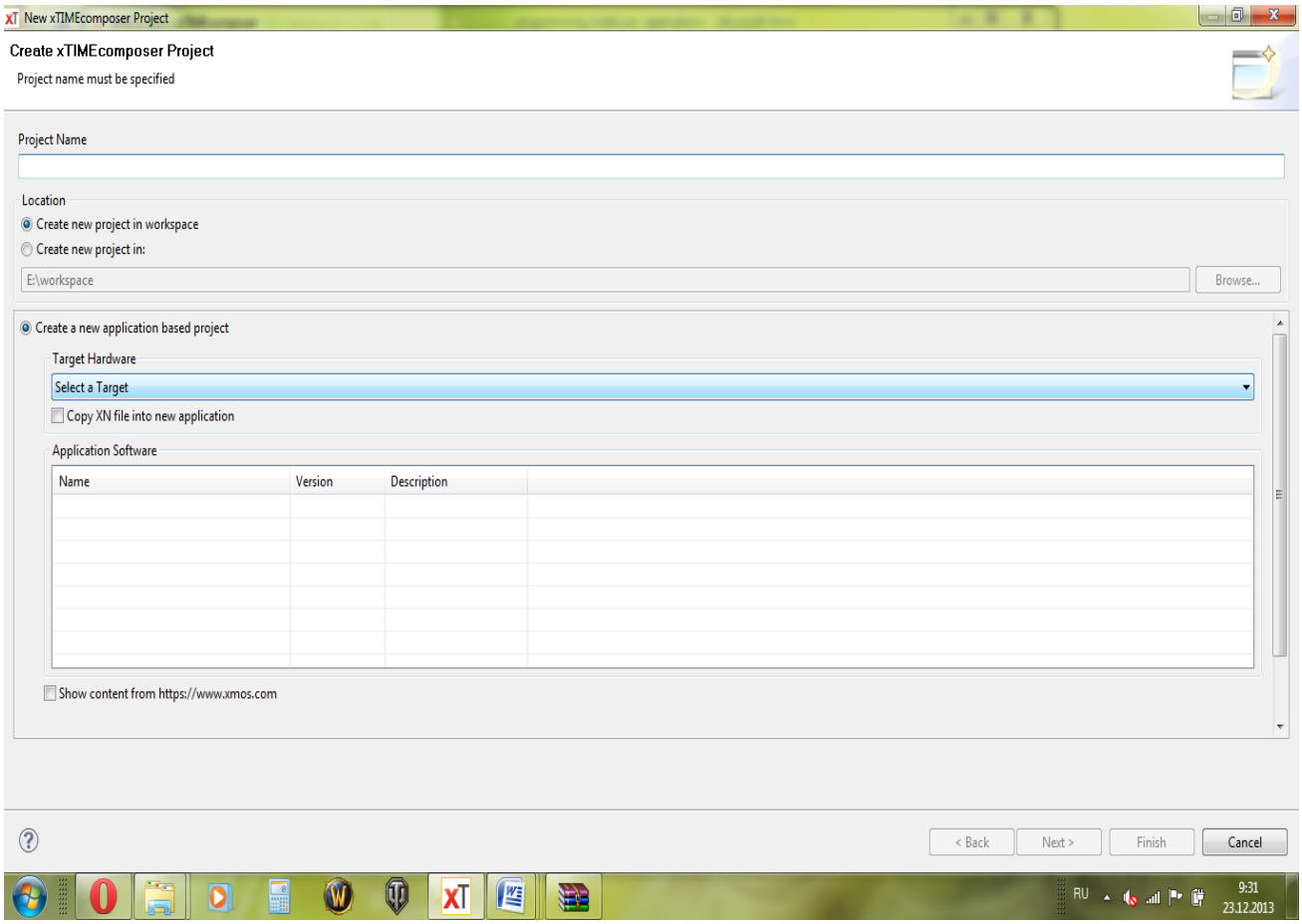
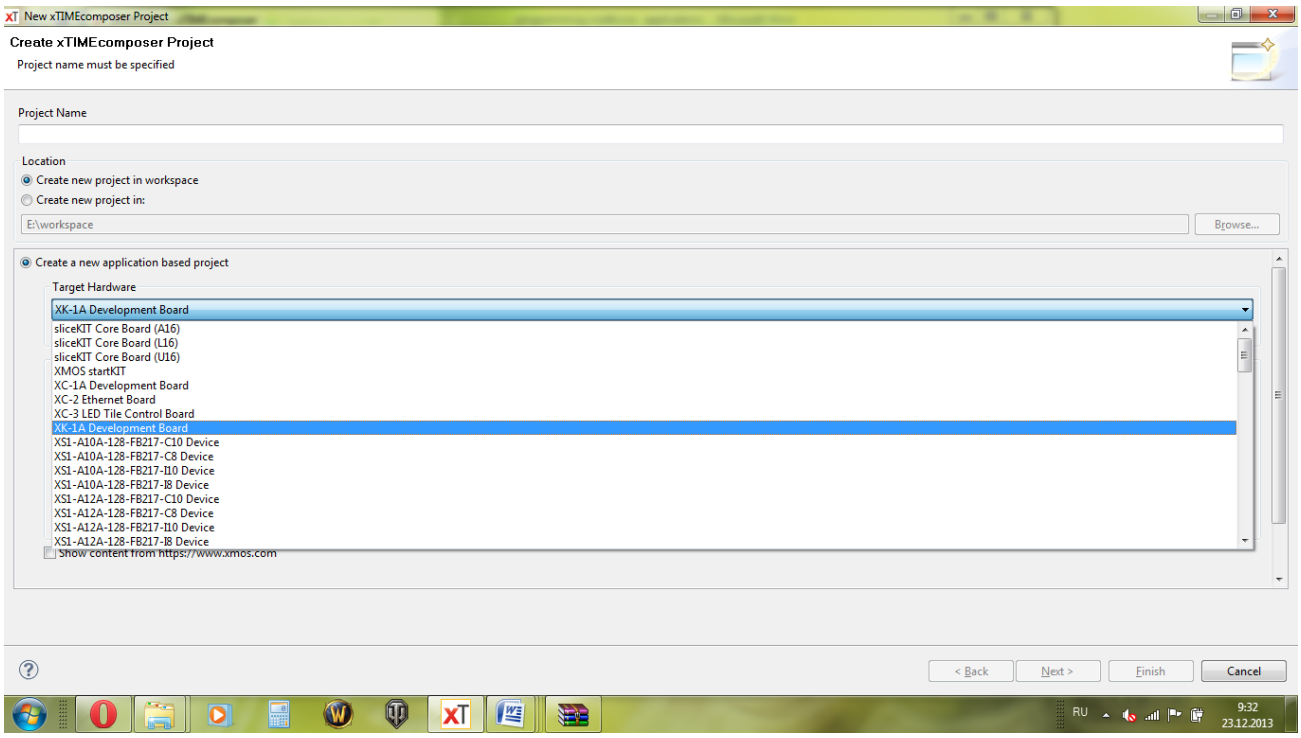


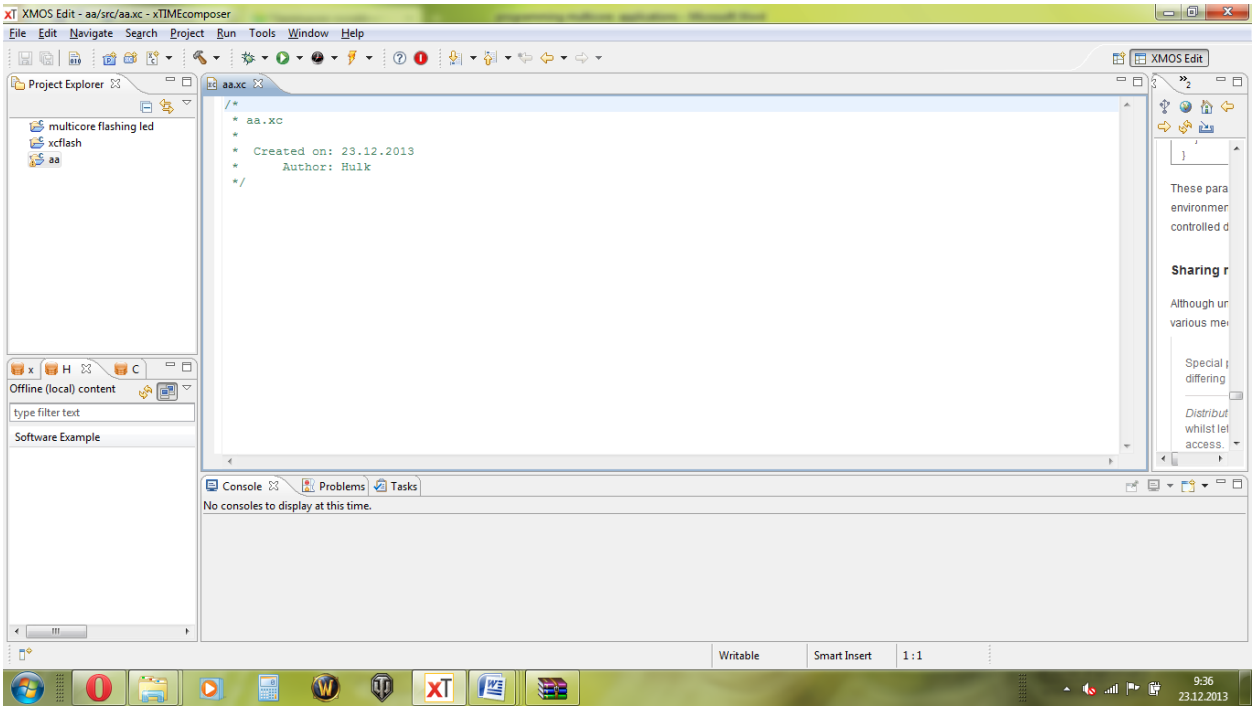
Рис.2.2.11. Окно создания проекта

В поле project name вводим имя будущего проекта. В поле select hardware выбираем целевую плату (в нашем случае это xk-1A):



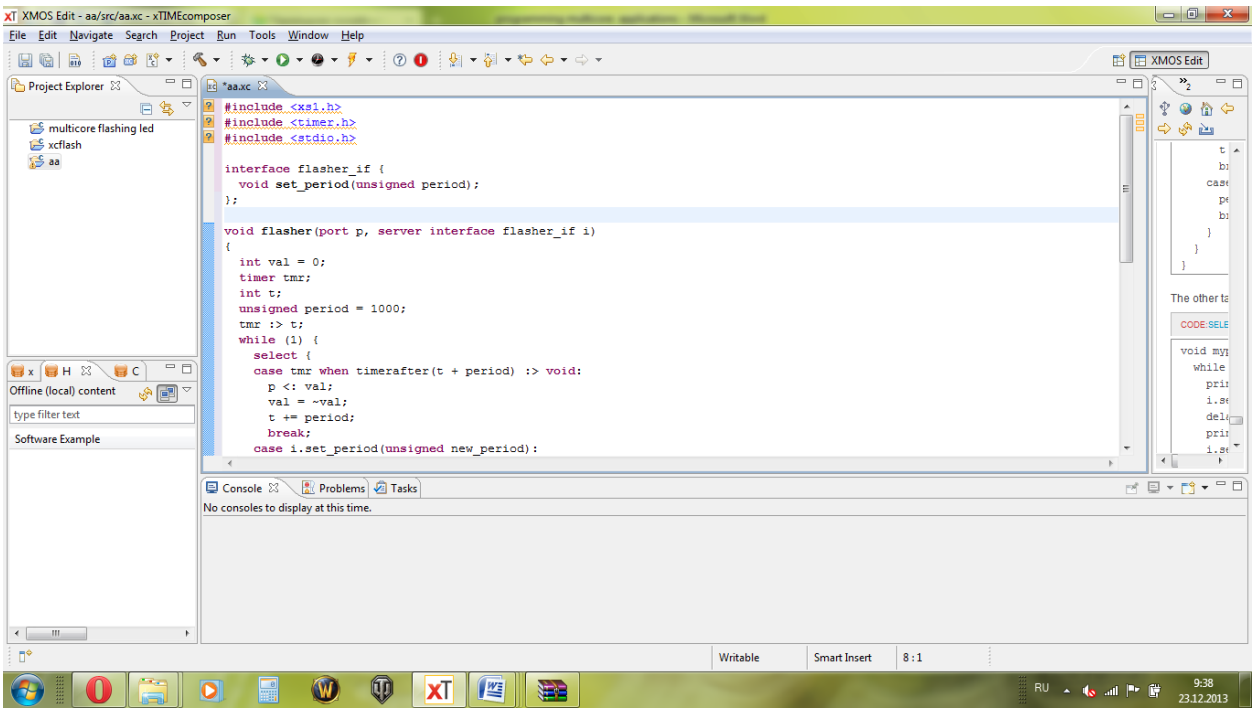
Устанавливаем галочку у “Copy XN file into a new application”. В поле ниже выбираем empty XC file и нажимаем finish.

После этого видим следующее окно:



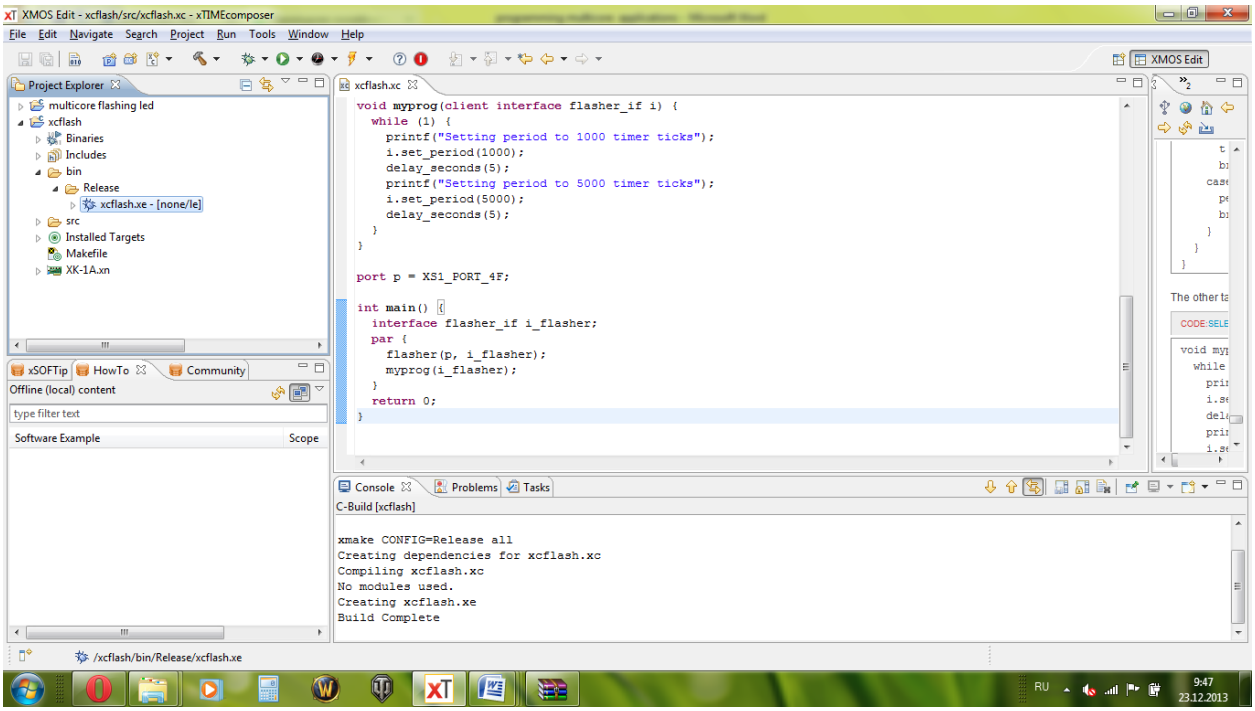
Далее в поле текстового редактора среды xTimecomposerStudio копируем код программы:

```
#include <xs1.h>
#include <timer.h>
#include <stdio.h>

interface flasher_if {
  void set_period(unsigned period);
};

void flasher(port p, server interface flasher_if i)
{
  int val = 0;
  timer tmr;
  int t;
  unsigned period = 1000;
  tmr :> t;
  while (1) {
    select {
    case tmr when timerafter(t + period) :> void:
      p <: val;
      val = ~val;
      t += period;
      break;
    case i.set_period(unsigned new_period):
      period = new_period;
      break;
    }
  }
}

void myprog(client interface flasher_if i) {
  while (1) {
    printf("Setting period to 1000 timer ticks");
    i.set_period(1000);
    delay_seconds(5);
    printf("Setting period to 5000 timer ticks");
    i.set_period(5000);
    delay_seconds(5);
  }
}

port p = XS1_PORT_4F;

int main() {
  interface flasher_if i_flasher;
  par {
    flasher(p, i_flasher);
    myprog(i_flasher);
  }
  return 0;
}
```

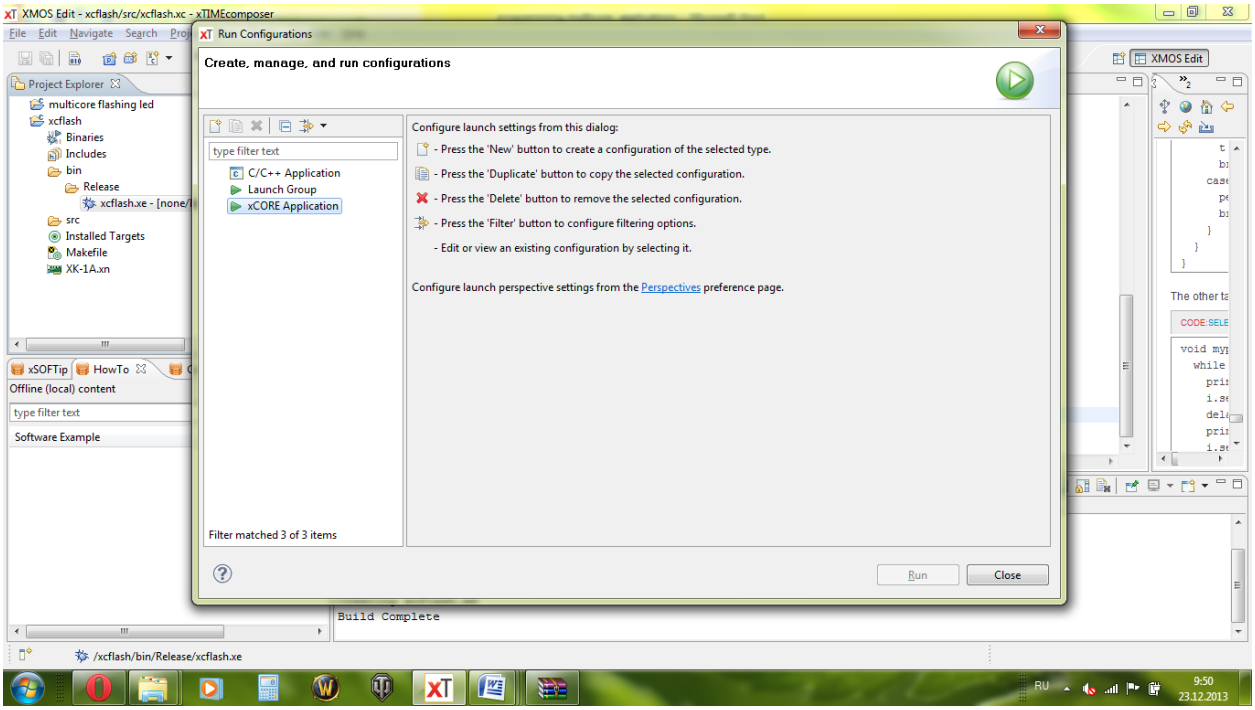


После чего выполняем построение. Для этого нажимаем на стрелку возле значка молоточка и выбираем тип построения «Release». В консоли снизу будет указано Build Complete.

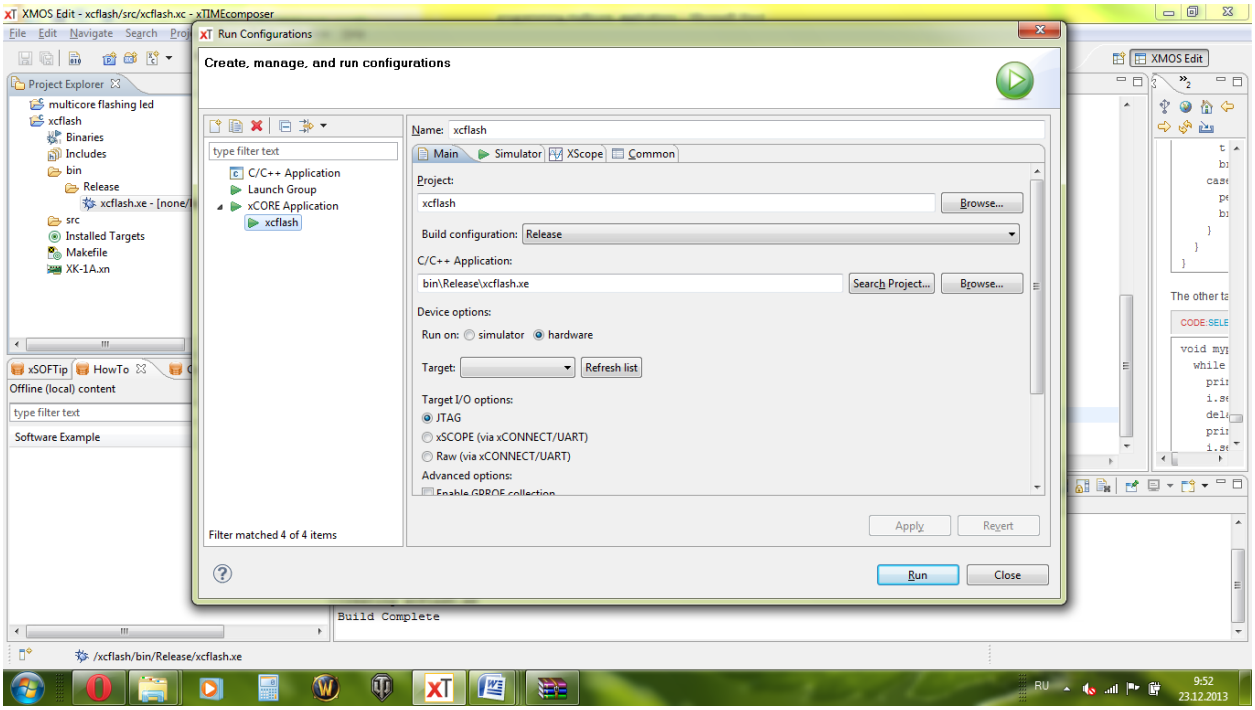
Далее, открываем в project explorer папку проекта, в нашем случае это xcflash. Далее находим и открываем: bin/Release/ и выделяем левой кнопкой мыши файл .xe:



Таким образом, указав для какого режима выполняется запуск и «прошивка», выполняем эти процедуры. Для этого в верхней строке меню открываем Run/run configurations, затем открывается окно:



В этом окне открываем раздел xCore application двойным нажатием левой кнопки мыши:



В разделе device option выбраем hardware. Для Target I/O options определяем соответствующее устройство (например, JTAG). И нажимаем кнопку run.

После этого произойдет прошивка платы, и можно будет наблюдать действие программы на плате xk-1A.

2.2.13. Задания

1. Используя модуль xk-1A и среду xTimecomposerStudio построить приложения из примеров 1, 2 и 3. Загрузить в память и инициировать работу этих приложений.
2. “Прошить" приложение в Flash-память контроллера.
3. В примере 3 изменить режим мигания светодиодов в коде программы. Построить проект и убедиться в изменении режима мигания на плате xk-1A.

Литература.

1. ARM7TDMI (Thumb) Data Sheet. / Atmel ES2.-1999.-204p.
2. ARM Architecture Reference Manual. / ARM Limited.-2006.-811p.
3. AT91 ARM THUMD – Based Microcontrollers AT91SAM7X512/256/128 Preliminary. / Atmel inc.-2007.-671p.
4. Sloss, A.N., ARM System Developer’s Guide: Designing and Optimizing System Software. / Andrew N.Sloss, Dominic Symes, Chris Wright, CA:Elsevier.-2004.-703p.
5. xMOS, XK-1A Development Board Tutorial [Электронный ресурс]-электрон.текстовые данные// XK-1A Hardware Manual URL: http://www.xmos.com/published/xk1ahw
6. . XMOS, XK-1A Development Board Tutorial [Электронный ресурс] - электрон.текстовые данные // https://www.xmos.com/published/xmos-programming-guide?version=latest
7. xTIMEcomposer User Guide REV 13.0.0 Publication Date: 2013/11/14 [Электронный ресурс] – электрон.текстовые данные, XMOS © 2013, All Rights Reserved // https://www.xmos.com/download/public/xTIMEcomposer-User-Guide%2813.0.0%29.pdf