Министерство образования и науки РФ

Федеральное государственное автономное образовательное учреждение высшего профессионального образования "Казанский (Приволжский) федеральный университет"

ИНСТИТУТ ФИЗИКИ

КАФЕДРА РАДИОЭЛЕКТРОНИКИ

Специальность: 011800.62 – Радиофизические методы по областям применения; радиофизические измерения

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА НА ТЕМУ:

«ФОРМИРОВАТЕЛЬ СИГНАЛОВ УПРАВЛЕНИЯ С ТОЧНОЙ ВРЕМЕННОЙ ПРИВЯЗКОЙ ДЛЯ АППАРАТУРЫ ТЕЛЕСКОПА РТТ-150 НА ОСНОВЕ МИКРОКОНТРОЛЛЕРА XMOS»

Работа завершена:

"__"_____2015 г. _____(Д.С.Ломовцев)

Работа допущена к защите:

Научный руководитель
к.ф - м.н.,
доц. каф. радиоэлектроники
"__"_____2015 г. _____(Р.И.Гумеров)

Заведующий кафедрой
Д.ф - м.н.,
профессор каф. радиоэлектроники
"__"_____2015 г. _____(М.Н.Овчинников)

Казань - 2015

СОДЕРЖАНИЕ

ВВЕДЕНИЕ……3
Глава 1. Инструментальная основа……4
1.1. Микроконтроллер xCORE……4
1.2. Плата XP-MC-CTRL-L2……7
1.3. XTAG……11
1.4. Сетевая модель OSI……11
1.5. Точная временная привязка……18
1.6. Навесное оборудование……20
Глава 2. Среда разработки……24
2.1. xTIMEcomposer studio……24
2.2. XTA……25
2.3. xSOFTip……26
2.4. xSCOPE……27
2.5. Отладчик……28
2.6. Симулятор……29
2.7. Программирование в xTIMEcomposer studio……29
2.8. HTML и CSS……34
Глава 3. Разработка……37
3.1. Описание……37
3.2. Алгоритм работы……38
3.3. Программный код……39
3.3.1 Формирователь импульсов……41
3.3.2 Интерфейс……44
3.3.3 ФАПЧ и временная привязка……48
3.3.4 Функция запуска……49
3.3.5 Функция main……50
3.4. Тестирование устройства……51
ЗАКЛЮЧЕНИЕ……55
СПИСОК ЛИТЕРАТУРЫ……56
ПРИЛОЖЕНИЕ 1……58
ПРИЛОЖЕНИЕ 2……71

ВВЕДЕНИЕ

На стандартном приборе вы получите стандартные результаты. Если вы хотите получать новые данные, лидировать в исследованиях, вы должны уметь совершенствовать свои инструменты.

В соответствии с развитием измерительных и IT технологий, на телескопе РТТ-150 КФУ регулярно модернизируется навесное оборудование и средства управления им. Астрономические данные имеют ценность только тогда, когда они привязаны к шкале звездного или всемирного времени UTC.

Целью настоящей работы, является создание устройства, которое позволяло бы с высокой точностью выполнять привязку к временной шкале, как работу навесного оборудования, так и телескопа в целом. В этом актуальность данной работы.

В качестве инструментальной основой данного устройства было решено использовать новейший многоядерный микроконтроллер с архитектурой xCORE.

Для реализации поставленной цели необходимо было сделать следующее:

1. изучить аппаратную платформу для реализации требуемого устройства и среду разработки для неё;
2. рассмотреть цепи запуска навесной аппаратуры телескопа РТТ-150 и определить наиболее употребляемые режимы запуска;
3. написать программный код, реализующий функции устройства;
4. обеспечить удаленное управление устройством по сети ethernet;
5. провести лабораторные испытания устройства.

ГЛАВА 1. ИНСТРУМЕНТАЛЬНАЯ ОСНОВА

1.1 Микроконтроллер xCORE

XMOS — британская fabless-компания, которая занимается разработкой многопоточных многоядерных процессоров, предназначенных для решения нескольких задач в режиме реального времени.

В компании разработали новое поколение 32-разрядных, многоядерных, многопоточных встраиваемых процессоров [1,2,3] с возможностью легкого масштабирования через межпроцессорную шину, которые предназначены для выполнения нескольких задач в реальном времени, цифровую обработку сигналов, управление различными процессами одновременно. Их отличает событийная модель управления процессами, плотность команд RISC-архитектуры, вычислительные возможности DSP и гибкость периферии FPGA. Многопоточная архитектура XMOS обеспечивает параллельное выполнение 8 задач одним ядром в реальном масштабе времени, при этом каждый поток выполняется не реже, чем раз в 11 нс.

Процессоры выполняются в выводных и безвыводных корпусах с количеством ядер 1, 2 и 4. Кроме того, недавно появилась новая линейка продукции: в кристалле реализован независимый генератор 12-битный АЦП, физический уровень USB 2.0 High Speed, что позволяет подключать к процессорам XS1-S USB-периферию на частоте 400 МГц без дополнительных микросхем. К процессорам прилагаются готовые примеры решений и стартовые комплекты.

Отдельная особенность XMOS — возможность объединения нескольких процессоров благодаря межпроцессорной связи XLink, обеспечивающей с быстродействием 1+ GBPS для решения одной сложной задачи на нескольких ядрах.

В микроконтроллерах, 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.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.

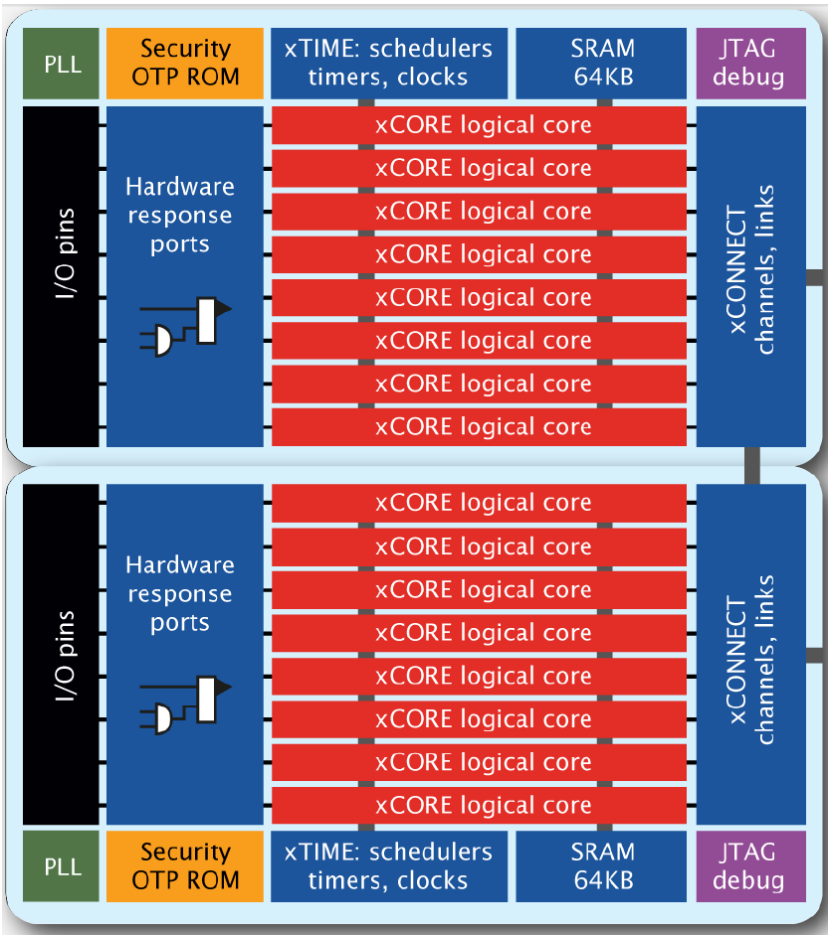


Рис. 1. Архитектура xCORE

Особенности:

- вычислительная мощность устройств от 4 до 16 ядер или от 400 до 1000 MIPS;
- гибкость реализации устройств;
- низкая задержка, отклик быстрее в 100 раз реальных I/O;
- ограниченное время выполнения;
- интеграция DSP с набором родных инструкций 32/64бит;
- безопасный OTP для защиты IP;
- бесплатные инструменты для программирования xSOFTip в xTIMEcomposer studio.

Процессоры XMOS совмещают в себе функции DSP, ASICs и FPGA решений с возможностью программирования через единый интерфейс C, XC или C++.

Микроконтроллеры XMOS нацелены на такие рынки, как: системы управления двигателями, управление светодиодными решениями, построение High Quality Audio систем.

Разработка программного обеспечения осуществляется с помощью среды разработки XDE (на основе платформы Eclipse), которая имеет универсальный набор возможностей: инструменты для симуляции и отладки программы, логический анализатор, приложение xSCOPE, утилиты для симуляции работы с платами, возможность работы с помощью командной строки или с помощью графического интерфейса. Есть версии для операционных систем Windows, Mac OS и Linux. Свободный доступ на имеющиеся библиотеки модулей xSOFTip, готовые простые приложения, проекты других пользователей. Актуальные версии находятся на хостинге GitHub (github.com/xcore).

Разработка программы осуществляется на языках C, C++ или XC. XC является версией языка C, разработанной XMOS. В нём используется тот же самый синтаксис и большая часть типов данных.

1.2 Плата XP-MC-CTRL-L2

Демонстрационная плата XP-MC-CTRL-L2 [4,5,6,7], изображена на рис. 2, очень хорошее решение для управления двигателями. Гибкость реализации устройств, низкая задержка и детерменизм делают микроконтроллеры xCORE идеальным выбором для проектов управления двигателями. Многоядерный подход позволяет строить многоосные системы с высокой скоростью и высоким разрешением, использующий передовые алгоритмы и с добавлением на свой выбор дополнительных функций, от промышленных сетей и ethercat до функционального LCD дисплея, все на одном устройстве.

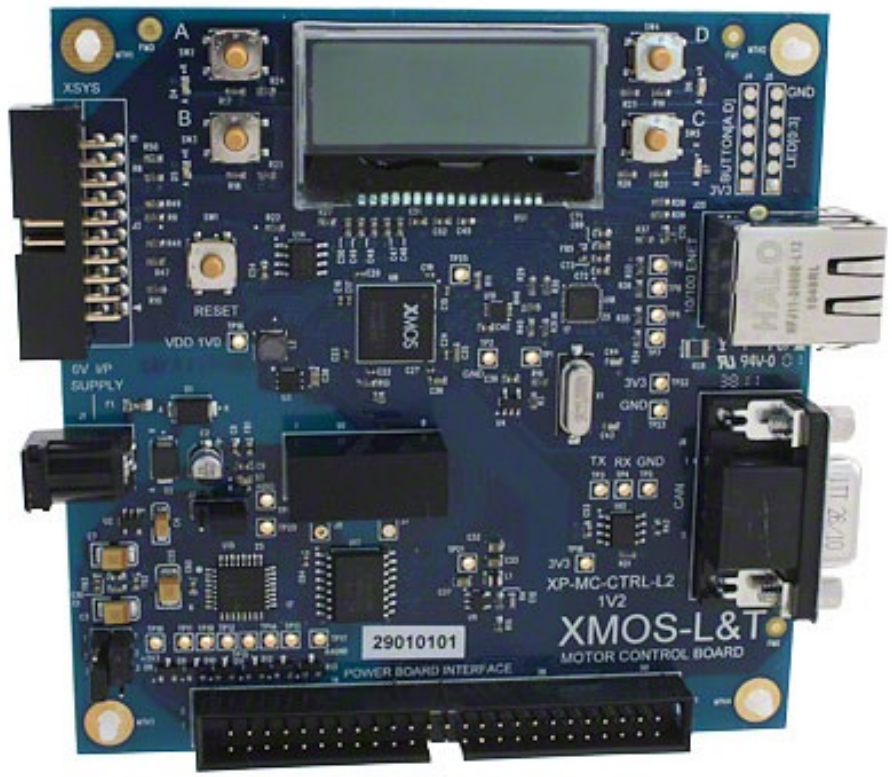


Рис. 2. Плата XP-MC-CTRL-L2

Из чего состоит плата:

1. многоядерный микроконтроллер XMOS xCORE XS1-L16;
2. LCD дисплей;
3. 4 кнопки;
4. 4 led диода;
5. кнопка сброса;
6. Ethernet;
7. CAN;
8. 25 MHz кварцевый резонатор;
9. выход питания;

10. АЦП для моторов;
11. преобразователь напряжения;
12. 20-ти пиновый вывод, для интерфейса XTAG;
13. 50-ти пиновый вывод, для подключения power board, на которой расположены двигатели.

Основные особенности:

- контроль многоосных и сложных двигателей (разрешение ШИМ 4 нс; 125 кГц ток контура скорости);
- обильные вычислительные мощности (многоосный контроль; сложные, высокоскоростные алгоритмы; сети и коммуникации);
- высокий уровень программирования (C и C++; Eclipse IDE);
- аппаратный анализ/симуляция (статическое время и логический анализ).

На рис. 3 и 4, приведена принципиальная схема платы XP-MC-CTRL-L2, согласно которой определяются контакты необходимой периферии для обеих плит.

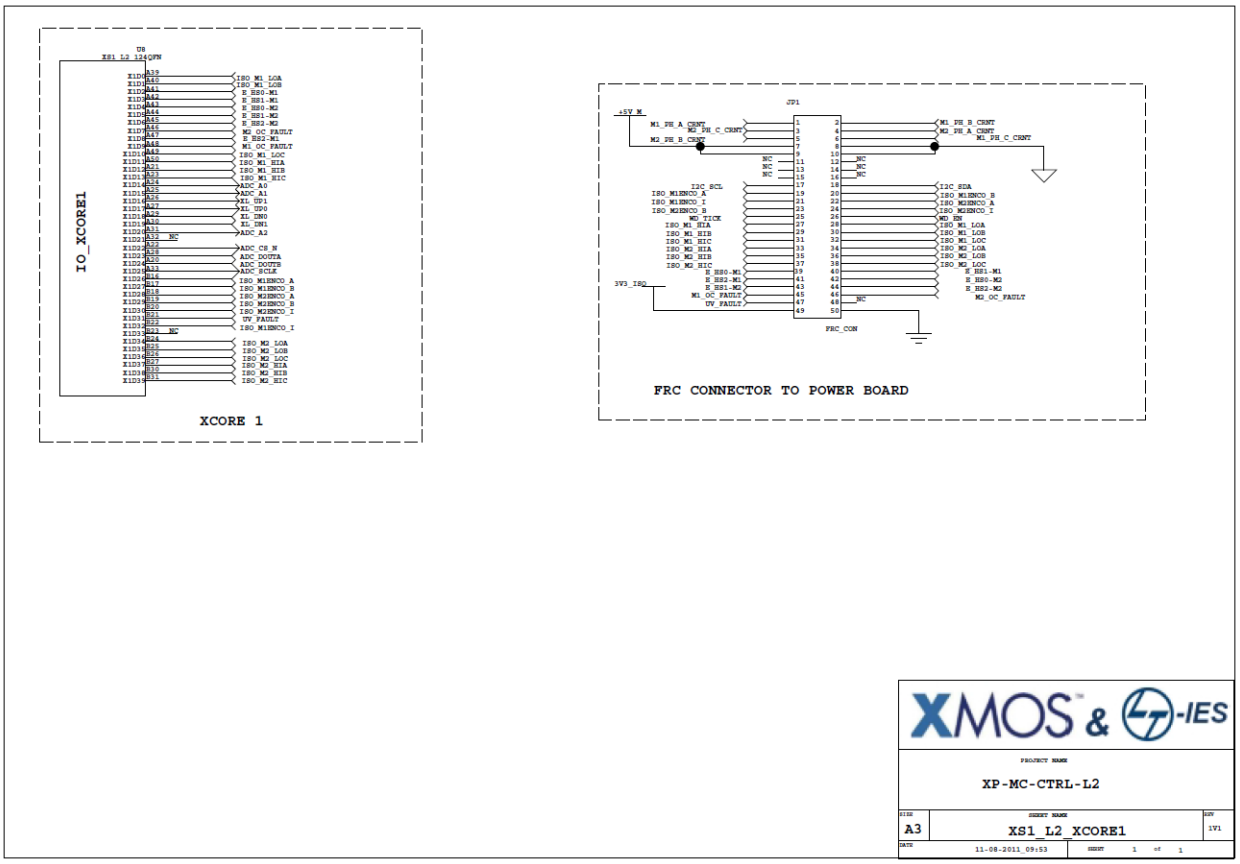


Рис. 3. Описание выводов

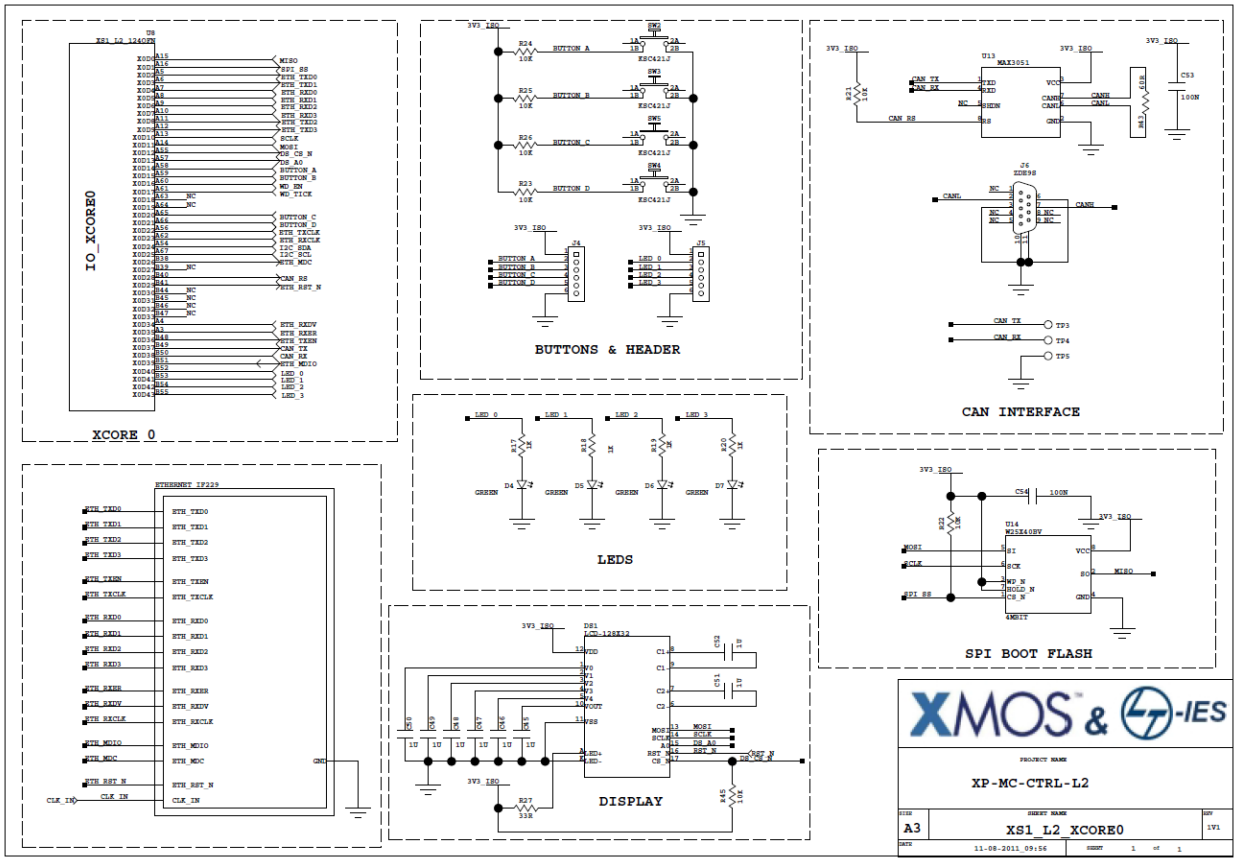


Рис. 4. Принципиальная схема периферии плиты

Решение управления двигателями от XMOS приносят преимущество многоядерных микроконтроллеров xCORE для широкого круга рынка, таких как робототехника, промышленное управление, бытовых товаров и игрушек.

Он отлично подойдет для тех, кто использует простой двигатель на постоянном токе или для реализации сложных алгоритмов управления HighEnd PMSM или SRM.

Поскольку устройства xCORE выполняют задачи детерминировано, они идеально подходят для задач реального времени. Мы можем выполнять несколько циклов быстрого управления и создать точный ШИМ с разрешением 4 нс. Можно реализовать алгоритмы высокой производительности, или добавить дополнительные оси управления, особенности человеко-машинного интерфейса и сетевые функции, такие как Ethernet и CAN.

Помимо того, что эта плата очень мощная, она очень проста в использовании. Можно работать в полностью знакомой среде, основанной на Eclipse, используя языки C и C++. Для упрощения разработки предоставляется широкий спектр проверенных программных блоков из библиотеки xSOFTip, а также уникальные инструменты для статического анализа времени, прототипирования и высокоскоростные внутрисхемные измерительные

приборы, чтобы быть уверенным что мы не пропустили дедлайн выполнения задачи в реальном времени.

Многоядерных подход xCORE позволяет легко разделить свою разработку: ядра могут быть назначены на такие задачи, как PWM, запуск петли управления, считывание значений АЦП, реализация периферийного устройства связи или эксплуатация сетевого стека.

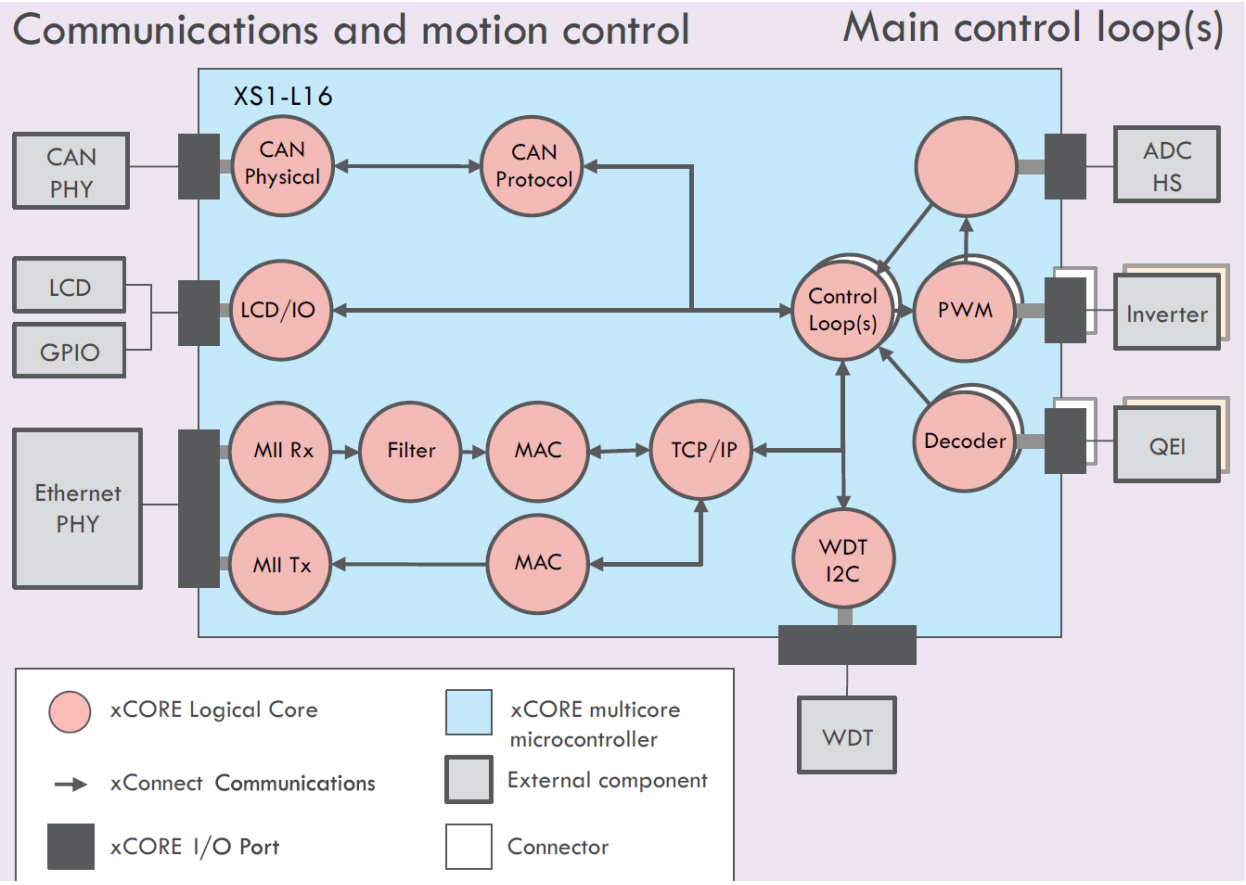


Рис. 5. Коммуникации и управление

xCORE многоядерные микроконтроллеры предлагают функциональность как у DSP, которая может легко справиться с самыми сложными проблемами управления двигателем. Даже после учета программного обеспечения, необходимого для основного управления двигателем и коммуникационных функций, устройства имеют 100 с MIPS в запасе из доступных для осуществления сложных алгоритмов управления.

Для приложений, требующих высокой степени интеграции, поддерживает широкий спектр стандартов связи, включая EtherCAT, Ethernet и CAN. xSOFTip модули для функций HMI помогут завершить разработку. И подход является масштабируемым. Ядра могут быть выделены, чтобы изменить количество осей, повысить скорость и точность контура управления, а также, чтобы обеспечить дальнейшую связь интерфейсов.

1.3 XTAG

XTAG [8] обеспечивает высокую скорость, низкую задержку моста между USB на хост-ПК и быстрого подключения JTAG (до 10Mbps) на наше устройство. Обеспечена молниеносно быстрая отладка функций, таких как пошаговая отладка. В дополнение к возможности отладки, поддерживаются порты UART и xCONNECT, обеспечивая связь с xCORE во время выполнения. Эти порты ключ к инструменту xSCOPE, который может записывать данные приложений на скорости до 1,25 MSPS.

Рис. 6. Модуль XTAG

Начальный уровень xTAG-2 обеспечивает все возможности отладки и поддержки для инструмента xSCOPE. Он входит в большинство комплектов для разработчиков от XMOS. Также предоставляется xTAG-PRO, который добавляет аналоговую область и динамический мониторинг питания для всех особенностей xTAG-2. Можно использовать инструменты xTIMEcomposer для мониторинга энергопотребления xCORE устройства и мониторинга поведения нашего приложения, используя библиотеку xSCOPE и симулятор; а также проверку требований синхронизации, используя анализатор синхронизации XMOS.

1.4 Сетевая модель OSI

Поскольку управление устройством, задания режимов его работы, должно выполняться по сети Ethernet (TCP/IP), рассмотрим особенности работы. В основе архитектуры сети Ethernet лежит модель OSI [9].

Сетевая модель OSI (англ. open systems interconnection basic reference model — базовая эталонная модель взаимодействия открытых систем, сокр. ЭМВОС; 1978 г) — сетевая модель стека сетевых протоколов OSI/ISO (ГОСТ Р ИСО/МЭК 7498-1-99).

В связи с затянувшейся разработкой протоколов OSI, в настоящее время основным используемым стеком протоколов является TCP/IP, разработанный ещё до принятия модели OSI и вне связи с ней.

Табл. 1. Модель OSI

Модель OSI				
Уровень (layer)		Тип данных (PDU)	Функции	Примеры
Host layers	7. Прикладной (application)	Данные	Доступ к сетевым службам	HTTP, FTP, SMTP
	6. Представительский (представления) (presentation)		Представление и шифрование данных	ASCII, EBCDIC, JPEG
	5. Сеансовый (session)		Управление сеансом связи	RPC, PAP
	4. Транспортный (transport)	Сегменты (segment)	Прямая связь между конечными пунктами и надежность	TCP, UDP
Media layers	3. Сетевой (network)	Пакеты (packet)/ Дейтаграммы (datagram)	Определение маршрута и логическая адресация	IPv4, IPv6, IPsec, AppleTalk
	2. Канальный (data link)	Биты (bit)/ Кадры (frame)	Физическая адресация	PPP, IEEE 802.2, L2TP, ARP
	1. Физический (physical)	Биты (bit)	Работа со средой передачи, сигналами и двоичными данными	DSL, USB

В литературе наиболее часто принято начинать описание уровней модели OSI с 7-го уровня, называемого прикладным, на котором пользовательские приложения обращаются к сети. Модель OSI заканчивается 1-м уровнем — физическим, на котором определены стандарты, предъявляемые независимыми производителями к средам передачи данных:

- тип передающей среды (медный кабель, оптоволокно, радиоэфир и др.);
- тип модуляции сигнала;
- сигнальные уровни логических дискретных состояний (нуля и единицы).

Любой протокол модели OSI должен взаимодействовать либо с протоколами своего уровня, либо с протоколами на единицу выше и/или ниже своего уровня. Взаимодействия с протоколами своего уровня называются горизонтальными, а с уровнями на единицу выше или ниже — вертикальными. Любой протокол модели OSI может выполнять только функции своего уровня и не может выполнять функций другого уровня, что не выполняется в протоколах альтернативных моделей.

Каждому уровню с некоторой долей условности соответствует свой операнд — логически неделимый элемент данных. Этим операндом на отдельном уровне можно оперировать в рамках модели и используемых протоколов. На физическом уровне мельчайшая единица — бит, на канальном уровне информация объединена в кадры, на сетевом — в пакеты (датаграммы), на транспортном — в сегменты. Любой фрагмент данных, логически объединённых для передачи — кадр, пакет, датаграмма — считается сообщением. Именно сообщения в общем виде являются операндами сеансового, представительского и прикладного уровней.

К базовым сетевым технологиям относятся физический и канальный уровни.

Прикладной уровень (уровень приложений; англ. application layer) — верхний уровень модели, обеспечивающий взаимодействие пользовательских приложений с сетью:

- позволяет приложениям использовать сетевые службы:
  - удалённый доступ к файлам и базам данных,
  - пересылка электронной почты;
- отвечает за передачу служебной информации;
- предоставляет приложениям информацию об ошибках;
- формирует запросы к уровню представления.

Протоколы прикладного уровня: RDP, HTTP, SMTP, SNMP, POP3, FTP, XMPP, OSCAR, Modbus, SIP, TELNET и другие.

Представительский уровень (уровень представления; англ. presentation layer) обеспечивает преобразование протоколов и кодирование/декодирование данных. Полученные с прикладного уровня запросы приложений, на уровне представления преобразуются в формат для передачи по сети, а полученные из сети данные преобразуются в формат приложений. На этом уровне может осуществляться сжатие/распаковка или шифрование/дешифрование, а также перенаправление запросов другому сетевому ресурсу, если они не могут быть обработаны локально.

Уровень представлений представляет собой промежуточный протокол для преобразования информации из соседних уровней. Это позволяет осуществлять обмен между приложениями на разнородных компьютерных системах, прозрачным для приложений образом. Уровень представлений обеспечивает форматирование и преобразование кода. Форматирование кода используется для того, чтобы гарантировать приложению поступление информации для обработки, которая имела бы для него смысл. При необходимости, этот уровень может выполнять перевод из одного формата данных в другой.

Уровень представлений имеет дело не только с форматами и представлением данных, он также занимается структурами данных, которые используются программами. Таким образом, уровень 6 обеспечивает организацию данных при их пересылке.

Чтобы понять, как это работает, представим, что имеются две системы. Одна использует для представления данных расширенный двоичный код обмена информацией EBCDIC, например, это может быть мейнфрейм компании IBM, а другая — американский стандартный код обмена информацией ASCII (его используют большинство других производителей компьютеров). Если этим двум системам необходимо обменяться информацией, то нужен уровень представлений, который выполнит преобразование и осуществит перевод между двумя различными форматами.

Другой функцией, выполняемой на уровне представлений, является шифрование данных, которое применяется в тех случаях, когда необходимо защитить передаваемую информацию от доступа несанкционированными получателями. Чтобы решить эту задачу, процессы и коды, находящиеся на уровне представлений, должны выполнить преобразование данных. На этом уровне существуют и другие подпрограммы, которые сжимают тексты и преобразовывают графические изображения в битовые потоки, так что они могут передаваться по сети.

Стандарты уровня представлений также определяют способы представления графических изображений. Для этих целей может использоваться формат PICT — формат изображений, применяемый для передачи графики QuickDraw между программами.

Другим форматом представлений является тэгированный формат файлов изображений TIFF, который обычно используется для растровых изображений с высоким разрешением. Следующим стандартом уровня представлений, который может использоваться для графических изображений, является стандарт, разработанный Объединенной экспертной группой по фотографии

(Joint Photographic Expert Group); в повседневном пользовании этот стандарт называют просто JPEG.

Существует другая группа стандартов уровня представлений, которая определяет представление звука и кинофрагментов. Сюда входят интерфейс электронных музыкальных инструментов (англ. Musical Instrument Digital Interface, MIDI) для цифрового представления музыки, разработанный экспертной группой по кинематографии стандарт MPEG, используемый для сжатия и кодирования видеороликов на компакт-дисках, хранения в оцифрованном виде и передачи со скоростями до 1,5 Мбит/с, и QuickTime — стандарт, описывающий звуковые и видео элементы для программ, выполняемых на компьютерах Macintosh и PowerPC.

Протоколы уровня представления: AFP — Apple Filing Protocol, ICA — Independent Computing Architecture, LPP — Lightweight Presentation Protocol, NCP — NetWare Core Protocol, NDR — Network Data Representation, XDR — eXternal Data Representation, X.25 PAD — Packet Assembler/Disassembler Protocol.

Сеансовый уровень (англ. session layer) модели обеспечивает поддержание сеанса связи, позволяя приложениям взаимодействовать между собой длительное время. Уровень управляет созданием/завершением сеанса, обменом информацией, синхронизацией задач, определением права на передачу данных и поддержанием сеанса в периоды неактивности приложений.

Протоколы сеансового уровня: ADSP (AppleTalk Data Stream Protocol), ASP (AppleTalk Session Protocol), H.245 (Call Control Protocol for Multimedia Communication), ISO-SP (OSI Session Layer Protocol (X.225, ISO 8327)), iSNS (Internet Storage Name Service), L2F (Layer 2 Forwarding Protocol), L2TP (Layer 2 Tunneling Protocol), NetBIOS (Network Basic Input Output System), PAP (Password Authentication Protocol), PPTP (Point-to-Point Tunneling Protocol), RPC (Remote Procedure Call Protocol), RTCP (Real-time Transport Control Protocol), SMPP (Short Message Peer-to-Peer), SCP (Session Control Protocol), ZIP (Zone Information Protocol), SDP (Sockets Direct Protocol)..

Транспортный уровень (англ. transport layer) модели предназначен для обеспечения надёжной передачи данных от отправителя к получателю. При этом уровень надёжности может варьироваться в широких пределах. Существует множество классов протоколов транспортного уровня, начиная от протоколов, предоставляющих только основные транспортные функции (например, функции передачи данных без подтверждения приема), и заканчивая протоколами, которые гарантируют доставку в пункт назначения нескольких пакетов данных в надлежащей последовательности, мультиплексируют несколько потоков данных, обеспечивают механизм

управления потоками данных и гарантируют достоверность принятых данных. Например, UDP ограничивается контролем целостности данных в рамках одной датаграммы, и не исключает возможности потери пакета целиком, или дублирования пакетов, нарушение порядка получения пакетов данных. TCP обеспечивает надёжную и непрерывную передачу данных, исключающую потерю данных или нарушение порядка их поступления или дублирования. Может перераспределять данные, разбивая большие порции данных на фрагменты и наоборот склеивая фрагменты в один пакет.

Протоколы транспортного уровня: ATP (AppleTalk Transaction Protocol), CUDP (Cyclic UDP), DCCP (Datagram Congestion Control Protocol), FCP (Fiber Channel Protocol), IL (IL Protocol), NBF (NetBIOS Frames protocol), NCP (NetWare Core Protocol), SCTP (Stream Control Transmission Protocol), SPX (Sequenced Packet Exchange), SST (Structured Stream Transport), TCP (Transmission Control Protocol), UDP (User Datagram Protocol).

Сетевой уровень (англ. network layer) модели предназначен для определения пути передачи данных. Отвечает за трансляцию логических адресов и имён в физические, определение кратчайших маршрутов, коммутацию и маршрутизацию, отслеживание неполадок и «заторов» в сети.

Протоколы сетевого уровня маршрутизируют данные от источника к получателю. Работающие на этом уровне устройства (маршрутизаторы) условно называют устройствами третьего уровня (по номеру уровня в модели OSI).

Протоколы сетевого уровня: IP/IPv4/IPv6 (Internet Protocol), IPX (Internetwork Packet Exchange, протокол межсетевого обмена), X.25 (частично этот протокол реализован на уровне 2), CLNP (сетевой протокол без организации соединений), IPsec (Internet Protocol Security). Протоколы маршрутизации - RIP (Routing Information Protocol), OSPF (Open Shortest Path First).

Канальный уровень (англ. data link layer) предназначен для обеспечения взаимодействия сетей на физическом уровне и контроля за ошибками, которые могут возникнуть. Полученные с физического уровня данные, представленные в битах, он упаковывает в кадры, проверяет их на целостность и, если нужно, исправляет ошибки (формирует повторный запрос поврежденного кадра) и отправляет на сетевой уровень. Канальный уровень может взаимодействовать с одним или несколькими физическими уровнями, контролируя и управляя этим взаимодействием.

Спецификация IEEE 802 разделяет этот уровень на два подуровня: MAC (англ. media access control) регулирует доступ к разделяемой физической среде, LLC (англ. logical link control) обеспечивает обслуживание сетевого уровня.

На этом уровне работают коммутаторы, мосты и другие устройства. Говорят, что эти устройства используют адресацию второго уровня (по номеру уровня в модели OSI).

Протоколы канального уровня: ARCnet, ATM, Controller Area Network (CAN), Econet, Ethernet, Ethernet Automatic Protection Switching (EAPS), Fiber Distributed Data Interface (FDDI), Frame Relay, High-Level Data Link Control (HDLC), IEEE 802.2 (provides LLC functions to IEEE 802 MAC layers), Link Access Procedures, D channel (LAPD), IEEE 802.11 wireless LAN, LocalTalk, Multiprotocol Label Switching (MPLS), Point-to-Point Protocol (PPP), Point-to-Point Protocol over Ethernet (PPPoE), Serial Line Internet Protocol (SLIP, устарел), StarLan, Token ring, Unidirectional Link Detection (UDLD), x.25, ARP.

В программировании этот уровень представляет драйвер сетевой платы, в операционных системах имеется программный интерфейс взаимодействия канального и сетевого уровней между собой. Это не новый уровень, а просто реализация модели для конкретной ОС. Примеры таких интерфейсов: ODI, NDIS, UDI.

Физический уровень (англ. physical layer) — нижний уровень модели, который определяет метод передачи данных, представленных в двоичном виде, от одного устройства (компьютера) к другому. Составлением таких методов занимаются разные организации, в том числе: Институт инженеров по электротехнике и электронике, Альянс электронной промышленности, Европейский институт телекоммуникационных стандартов и другие. Осуществляют передачу электрических или оптических сигналов в кабель или в радиоэфир и, соответственно, их приём и преобразование в биты данных в соответствии с методами кодирования цифровых сигналов.

На этом уровне также работают концентраторы, повторители сигнала и медиаконвертеры.

Функции физического уровня реализуются на всех устройствах, подключенных к сети. Со стороны компьютера функции физического уровня выполняются сетевым адаптером или последовательным портом. К физическому уровню относятся физические, электрические и механические интерфейсы между двумя системами. Физический уровень определяет такие виды сред передачи данных как оптоволокно, витая пара, коаксиальный кабель, спутниковый канал передач данных и т. п. Стандартными типами сетевых интерфейсов, относящимися к физическому уровню, являются: V.35, RS-232, RS-485, RJ-11, RJ-45, разъемы AUI и BNC.

Протоколы физического уровня: IEEE 802.15 (Bluetooth), IRDA, EIA RS-232, EIA-422, EIA-423, RS-449, RS-485, DSL, ISDN, SONET/SDH, 802.11 Wi-Fi,

Etherloop, GSM Um radio interface, ITU и ITU-T, TransferJet, ARINC 818, G.hn/G.9960.

Поскольку наиболее востребованными и практически используемыми стали протоколы (например TCP/IP), разработанные с использованием других моделей сетевого взаимодействия, далее необходимо описать возможное включение отдельных протоколов других моделей в различные уровни модели OSI.

Семейство TCP/IP имеет три транспортных протокола: TCP, полностью соответствующий OSI, обеспечивающий проверку получения данных; UDP, отвечающий транспортному уровню только наличием порта, обеспечивающий обмен датаграммами между приложениями, не гарантирующий получения данных; и SCTP, разработанный для устранения некоторых недостатков TCP, в который добавлены некоторые новшества. (В семействе TCP/IP есть ещё около двухсот протоколов, самым известным из которых является служебный протокол ICMP, используемый для внутренних нужд обеспечения работы; остальные также не являются транспортными протоколами).

1.5 Точная временная привязка сигналов

Точность временной привязки сигналов формируемых устройством на основе микроконтроллеров, зависит от точности (жесткости) системы реального времени, которая используется в нашей задаче.

Система реального времени (СРВ) [10] — это система, которая должна реагировать на события во внешней по отношению к системе среде или воздействовать на среду в рамках требуемых временных ограничений. Оксфордский словарь английского языка говорит об СРВ как о системе, для которой важно время получения результата. Другими словами, обработка информации системой должна производиться за определённый конечный период времени, чтобы поддерживать постоянное и своевременное взаимодействие со средой. Естественно, что масштаб времени контролирующей системы и контролируемой ей среды должен совпадать.

Под реальным временем понимается количественная характеристика, которая может быть измерена реальными физическими часами, в отличие от логического времени, определяющего лишь качественную характеристику, выражаемую относительным порядком следования событий. Говорят, что система работает в режиме реального времени, если для описания работы этой системы требуются количественные временные характеристики.

Процессы (задачи) систем реального времени могут иметь следующие характеристики и связанные с ними ограничения:

- дедлайн (англ. deadline) — критический срок обслуживания, предельный срок завершения какой-либо работы;
- латентность (англ. latency) — время отклика (время задержки) системы на внешние события;
- джиттер (англ. jitter) — разброс значений времени отклика. Можно различить джиттер запуска (англ. release jitter) — период времени от готовности к исполнению до начала собственно исполнения задачи и джиттер вывода (англ. output jitter) — задержка по окончании выполнения задачи. Джиттер может возникать под влиянием других, одновременно исполняемых задач.

В моделях систем реального времени могут фигурировать и другие параметры, например, период и количество итераций (для периодических процессов), нагрузка (англ. load) — количество команд процессора в худшем случае.

В зависимости от допустимых нарушений временных ограничений системы реального времени можно поделить на системы жёсткого реального времени, для которых нарушения равнозначны отказу системы, и системы мягкого реального времени, нарушения характеристик которых приводят лишь к снижению качества работы системы.

Следует заметить, что определение жёсткого реального времени ничего не говорит об абсолютном значении времени отклика: это могут быть как миллисекунды, так и недели. Требования к системам мягкого реального времени можно задать только в вероятностных терминах, например, как процент откликов, выданных в установленные временные рамки. Интересно, что при проектировании предварительные расчёты легче выполнить для системы жёсткого реального времени, чем получить, например, долю выполняемых в срок задач в системе мягкого реального времени, поэтому разработчики таких систем часто пользуются инструментами и методиками для проектирования систем жёсткого реального времени.

События реального времени могут относиться к одной из трёх категорий:

- Асинхронные события — полностью непредсказуемые события. Например, вызов абонента телефонной станции.
- Синхронные события — предсказуемые события, случающиеся с определённой регулярностью. Например, вывод аудио и видео.
- Изохронные события — регулярные события (разновидность асинхронных), случающиеся в течение интервала времени. Например, в мультимедийном приложении данные аудиопотока должны прийти за время прихода соответствующей части потока видео.

С развитием технологий системы реального времени нашли применения в самых различных областях. Особенно широко СРВ применяются в промышленности, включая системы управления технологическими процессами, системы промышленной автоматики, SCADA-системы, испытательное и измерительное оборудование, робототехнику. Применения в медицине включают в себя томографию, оборудование для радиотерапии, прикроватный мониторинг. СРВ встроены в периферийные устройства компьютеров, телекоммуникационное оборудование и бытовую технику, такую как лазерные принтеры, сканеры, цифровые камеры, кабельные модемы, маршрутизаторы, системы для видеоконференций и интернет-телефонии, мобильные телефоны, микроволновые печи, музыкальные центры, кондиционеры, системы безопасности. На транспорте СРВ применяются в бортовых компьютерах, системах регулирования уличного движения, управлении воздушного движения, аэрокосмической технике, системе бронирования билетов и т. п. СРВ находят применения и в военной технике: системах наведения ракет, противоракетных системах, системах спутникового слежения.

Примеры систем, работающих в режиме реального времени:

- АСУ ТП химического реактора;
- бортовая система управления космического аппарата;
- АСНИ в области ядерной физики;
- система обработки аудио- и видеопотоков при трансляции в прямом эфире;
- интерактивная компьютерная игра.

1.6 Навесное оборудование

На телескопе РТТ-150 КФУ имеется 2 камеры прямого изображения iKon DW436 [11] и iXon DU888 [12], предназначенные для получения прямых изображений участков неба, с целью определения блеска и положения звезд.

На рис. 7, изображена камера iKon DW436. Её разрешение составляет 2048x2048 пикселей.

Рис. 7. Внешний вид камеры iKon DW436

На рис. 8, изображена камера iXon DU888.

Рис. 8. Внешний вид камеры iXon DU888

Камера содержит следующее:

- датчик EMCCD с предварительным усилителем;
- 16-битный, аналого-цифровой преобразователь, чтобы оцифровывать данные из плат аналоговых контроллеров;
- датчик температуры с предварительным усилителем;
- термоэлектрический кулер и схема охлаждения;
- входные и выходные разъемы.

Камера может быть присоединена к микроскопу, или другому оптическому устройству, для сбора данных.

Разрешение камеры 1024x1024 пикселей.

Режимы работы камер:

1) Одиночный

Одиночная развертка - самый простой режим сбора, в которой система выполняет одну развертку ПЗС.

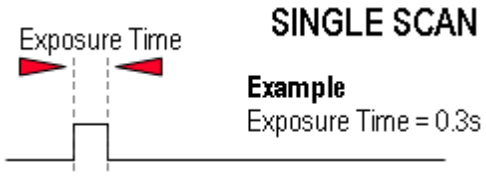


Рис. 9. Одиночный режим

2) Видео

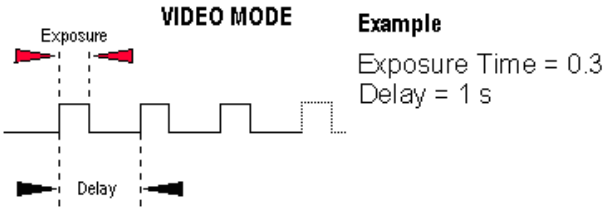


Рис. 10. Видео режим

Режим доступен только на камере iXon DU888.

3) Накопление

Можно выбрать следующие параметры в диалоговом окне настройки:

- время экспозиции;
- период накопления. Этот параметр доступен, только если выбран внутренний запуск;
- количество накоплений.

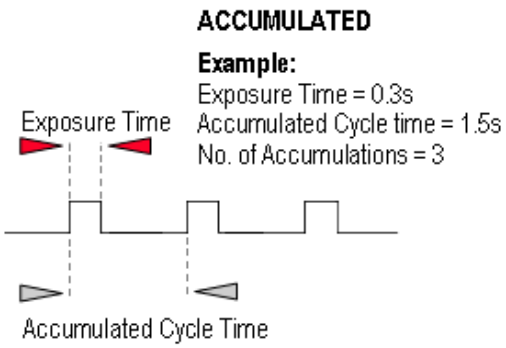


Рис. 11 Режим накопления

4) Кинетическая серия

Оборудование в режиме кинетическая серия управляется при помощи двух цифровых входов. На первый подается специальный длинный импульс, длительность которого равна длительности всего эксперимента. На второй подаются сами импульсы.

В диалоговом окне настройка можно ввести следующие параметры:

- время экспозиции;
- период кинетической серии: время между началом и окончанием каждого импульса кинетической серии;
- количество в кинетической серии.

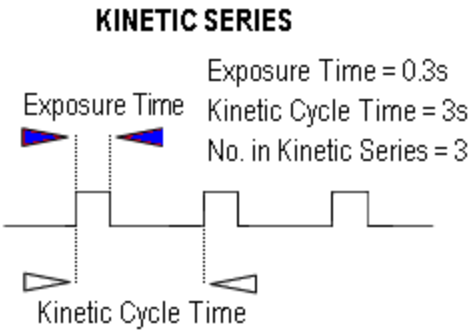


Рис. 12. Режим кинетическая серия

ГЛАВА 2 СРЕДА РАЗРАБОТКИ

2.1 xTIMEcomposer studio

Среда разработки xTIMEcomposer studio [13] обеспечивает привычную встроенную среду разработки программного обеспечения, основанную на Eclipse. Можно запрограммировать многоядерный микроконтроллер xCORE с использованием языков высокого уровня, таких как ANSI, C и C++, или при необходимости, сборку в любой комбинации, что соответствует современным потребностям. Чтобы программирование xCORE было еще проще, в компании XMOS было создано несколько простых расширений языка C. Оно позволяет взять под контроль расширенные возможности многоядерных микроконтроллеров, такие как определение параллелизма потоков данных и событий. Для людей, знакомы с языком C, расширение языка окажется простым в использовании.

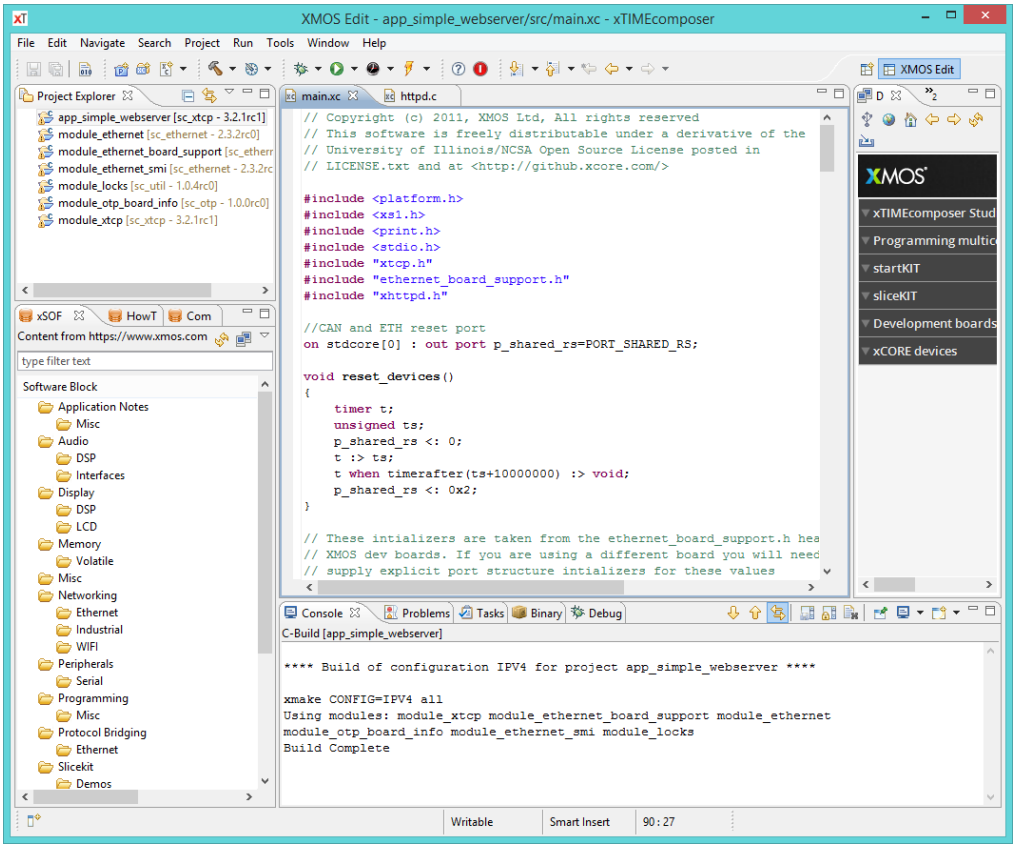


Рис. 13. xTIMEcomposer studio

Функциональные проблемы могут быть выявлены и зафиксированы с помощью отладчика XMOS GNU Debugger (GBD), в то время как уникальный статический анализатор времени XTA (XMOS Timing Analyzer) можно использовать для проверки того, что все требования режима реального времени

будут выполнены без необходимости использования сложных и трудоемких натурных динамических испытаний.

Что бы точно знать, что происходит внутри процессора и анализировать поведение системы можно воспользоваться инструментом xSCOPE для сбора данных от запущенного приложения в режиме реального времени.

Во время разработки, можно загрузить программу с хост-ПК через JTAG. В изготовленной конструкции необходимо уметь программировать свой образ во FLASH. Инструменты XMOS и функции безопасности позволяют шифровать программы на флэш-памяти и включать безопасный загрузчик.

2.2 XTA

Устройства xCORE обладают однозначно низкой задержкой и определенными сроками выполнения. Это делает их идеальными для программирования систем жесткого реального времени. Разработка систем реального времени и критических систем безопасности могут построить контуры управления, которые обрабатывают несколько одновременных задач. Можно быть уверенным, что сроки выполнения никогда не будут превышены.

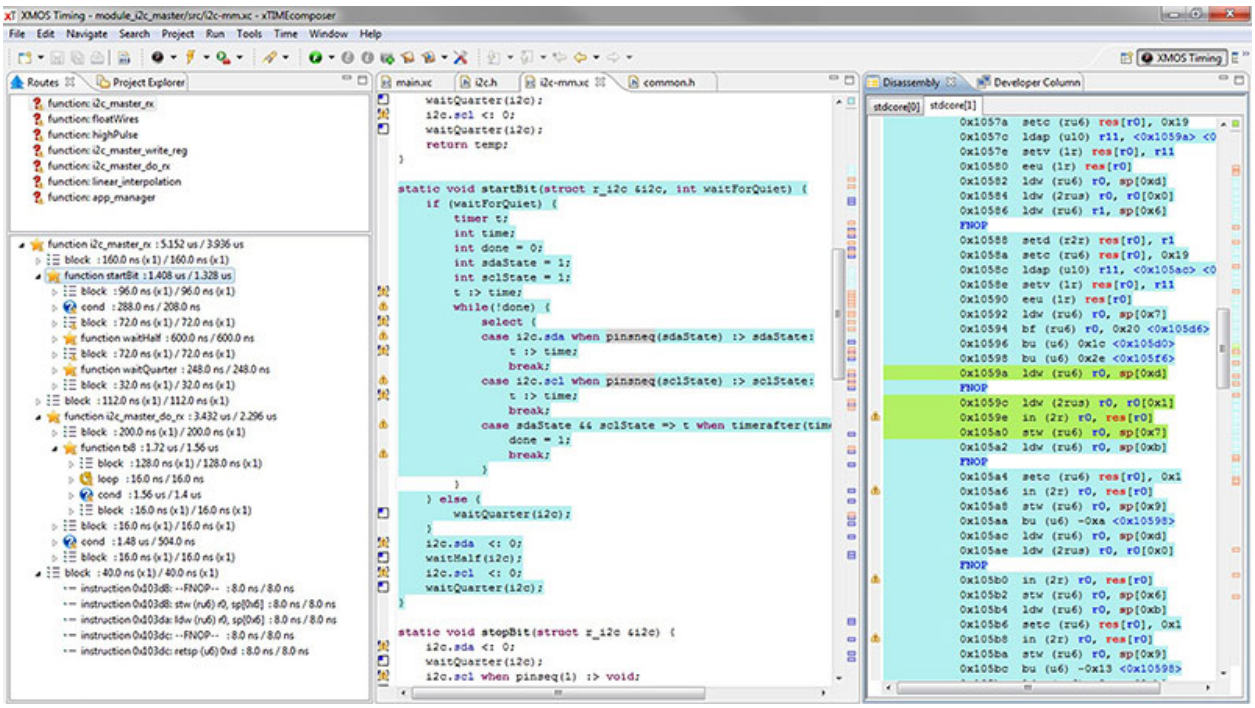


Рис. 14. XTA

Чтобы убедиться, что выбор времени был правилен, был создан такой инструмент, как временной анализатор XMOS XTA (XMOS Timing Analyzer). XTA работает на основе анализа "Application Binary" и сообщает худший и лучший случай синхронизации пути через код. Это устраняет необходимость в испытательном стенде и снижает требовательность к вычислительной

мощности. Можно точно знать, что гарантированные сроки выполнения задач всегда будут выполнены. О любом провисании в сроках сообщает инструмент, который позволяет добавить больше функциональных возможностей, или уменьшить рабочую частоту для экономии энергии.

XTA может быть использован в одном из двух режимов. Во первых, можно просматривать двоичный файл для графического исследования временных путей между двумя точками в программном коде. Режим просмотра отображает структуру кода, лучший и наихудший сроки выполнения, и гистограмму с сроками пути.

Второй режим работы - скриптом. Он работает путем тестирования временного утверждения в коде против анализируемого времени выполнения. Затем компилятор сообщает о принятии положительного либо отрицательного решения для каждого утверждения.

XTA является уникальным инструментом, который работает с многоядерными микроконтроллерами xCORE и доступен для бесплатного скачивания с официального сайта, как часть программного обеспечения xTIMEcomposer studio.

2.3 xSOFTip

xSOFTip explorer представляет собой графическое окно в студии, в котором представлен список простых приложений и готовых блоков кода, что упрощает написание собственного приложения.

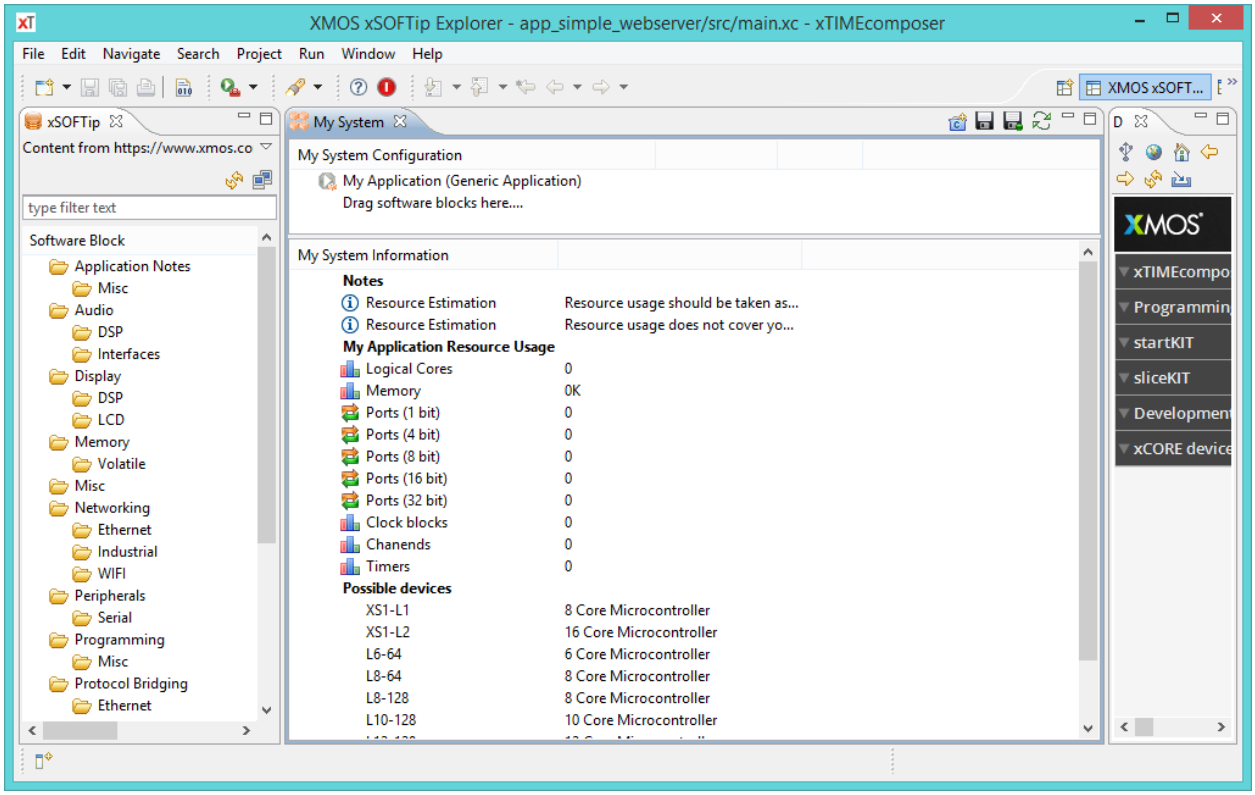


Рис. 15. xSOFTip

xSOFTip включает документацию для каждого блока, что облегчает понимание для чего данный блок нужен, какие у него возможности и как его использовать. xSOFTip является инструментом для быстрого и простого монтажа и настройки нашего оборудования, так как нам это требуется.

Помимо того, что он включен в студию, xSOFTip Explorer также доступен как самостоятельный инструмент.

2.4 xSCOPE

Отладка в цепи может быть сложной задачей, так как система реального времени подвержена событиям реального времени, из-за чего может быть трудно отследить причины непредвиденного поведения системы. Пошаговый разбор кода в отладчике бессмыслен, так как системы жесткого реального времени сразу ломаются, если сроки выполнения были пропущены.

Это означает, что отладка в цепи требует способ мониторинга системы и просмотр данных не используя программный код. Значит должен осуществляться такой способ сбора данных, который не будет влиять на сроки выполнения и позволит работать системы без изменения её поведения. Также необходимо, чтобы наблюдались переменные значения, пины ввода/вывода, контур управления и наблюдать непосредственно потоки данных. Все это, реализовано в xSCOPE.

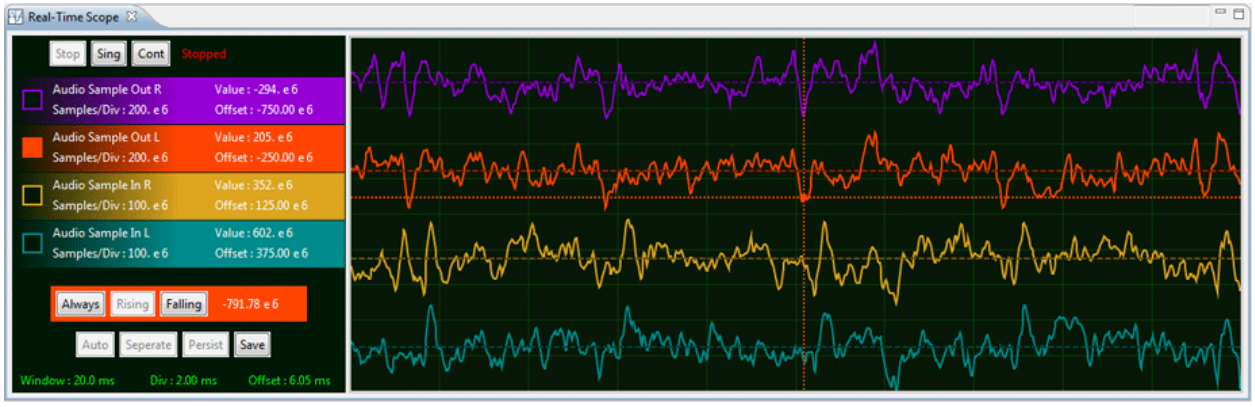


Рис. 16. Real-Time Scope

После подключения xSCOPE ведет себя также как и обычный осциллограф. Имеется контроль времени, несколько входных каналов, масштабирование, триггер. Каждое окно показывает данные пользователя, форматирование, основной идентификатор, в случае исполнения одно и того же участка кода несколькими ядрами, а также время кратное 10 нс. Все это позволяет с легкостью наладить работу системы реального времени.

В отличии от традиционных механизмов трассировки xSCOPE обрабатывает большой объем данных с очень высокой производительностью.

xSCOPE может работать в автономном режиме, собирая данные и сохраняя в файле, что позволяет анализировать их в xTIMEcomposer studio или в программах от других производителей. Эта также позволяет отлаживать разработку в облачных системах.

2.5 Отладчик

Отладчик позволяет отлаживать программу непосредственно на микроконтроллера через модуль XTAG, аналог интерфейса JTAG.

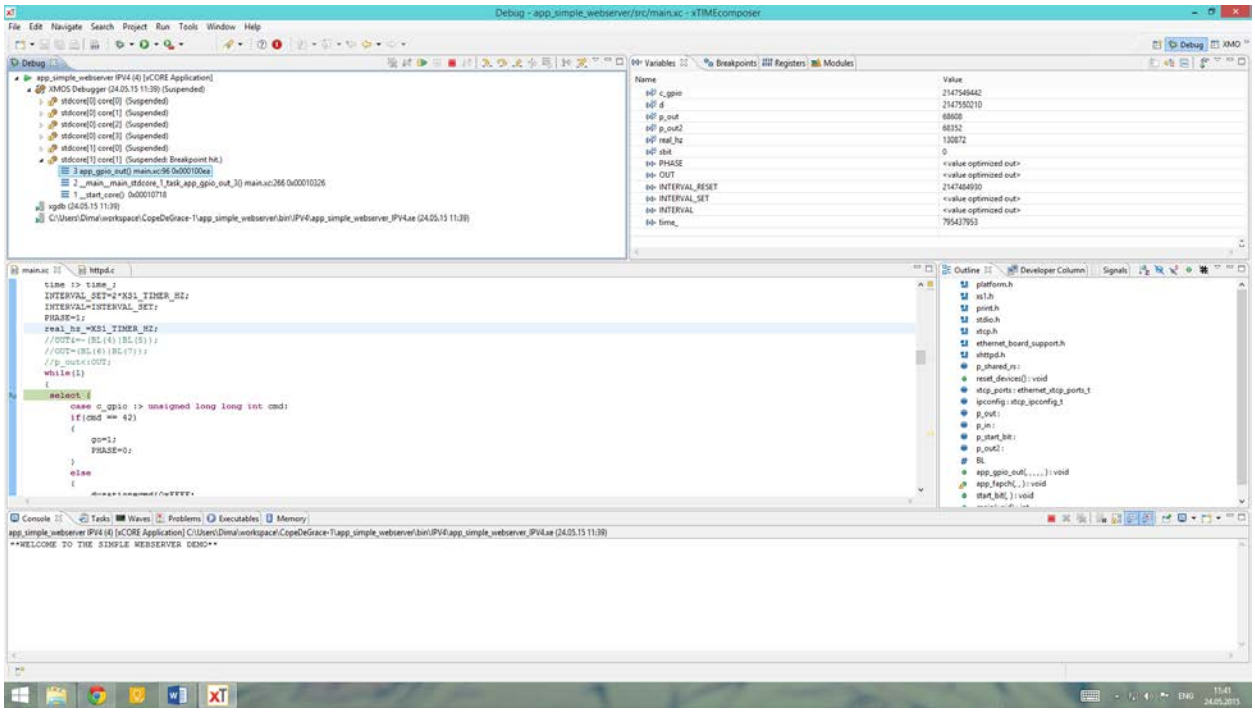


Рис. 17. Отладчик

В главном окне отображается построчное выполнение кода. Можно устанавливать точки остановки в коде. Это позволяет отслеживать наступление определенных событий и просматривать к каким результатам они приводят.

Во вкладке Memory видна карта памяти энергонезависимой памяти программ и оперативно запоминающего устройства.

Вкладка Variables дает обзор переменных текущей функции и значения глобальных переменных, которые были отмечены в данном окне для отслеживания. Отображение значения переменных может зависеть от уровня оптимизации кода.

Во вкладке Disassembly показан код в дизассемблере, которые показывает выполнение кода на машинном уровне, с передачей значений в регистры.

Значения основных регистров выводятся во вкладке Registers.

2.6 Симулятор (xSIM)

Симулятор позволяет отлаживать программу на программном уровне без аппаратного соединения. Код выполняется на компьютере, а не на микроконтроллере. При этом все необходимые функции отладки остаются работоспособными.

Потактовый симулятор модели всей системы xCORE включает: пакеты, узлы, плиты, ядра, средства, ссылки, порты и память. Симулятор может подключать также внешние компоненты к контактам/ссылкам построив испытательный стенд или смоделировать всю систему, используя конфигурацию замыкания, определяемые пользователем модули и испытательные стенды.

xSIM включает в себя множество вариантов трассировки и VCD (Value change dump - значение изменений дампа) механизм трассировки, который связывает выход к источникам кода и выдает файл для автономного анализа в xTIMEcomposer студии или сторонних утилит. Если нужно еще больше информации, xSIM обеспечивает интерфейс трассировки плагина, который может быть использован для написания пользовательских форматов файлов.

2.7 Программирование в xTIMEcomposer studio

Для программирования устройств от XMOS был разработан язык XC [14].

XC предназначен для программирования в режиме реального времени встроенных параллельных процессоров, но специально ориентированные на архитектуре процессора XMOS xCORE.

XC это императивный язык программирования, основанный на особенности организации параллелизма языка Occam, а также на синтаксисе и последовательных особенностях языка C. Это обеспечивает примитивные особенности, которые соответствуют различным предоставляемым архитектурным ресурсам, в частности: концы каналов, затворы, порты и таймеры.

В сочетании с процессорами xCORE, XC используется для построения встраиваемых систем с уровнем I/O, производительность в режиме реального времени и вычислительной мощности, как правило, приписываемых ПЛИС или ASIC устройств.

Программа XC выполняется на плите xCORE. Каждая плита содержит от одного до восьми процессорных ядер, а также ресурсы, которые могут быть разделены между ядрами, в том числе ввод/вывод и память. Все плиты соединены коммуникационной сетью, что позволяет любой плите взаимодействовать с другой плитой. В частности, целевая система задается во время компиляции и компилятор гарантирует, что достаточное количество

плит, ядер и ресурсов доступны для выполнения компилируемой программы. На рис. 18, показано окно консоли после сборки проекта. Здесь видны параметры плит, количество используемых ядер, каналов, таймеров и памяти.

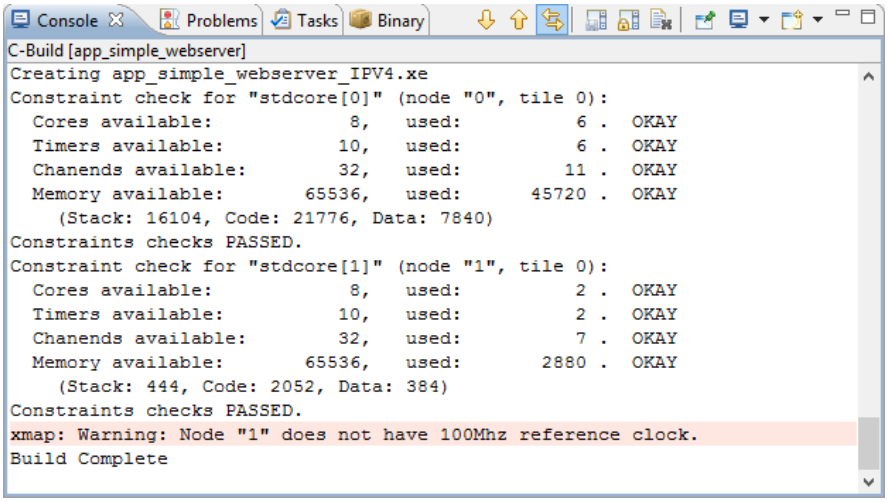


Рис. 18. Окно консоли после сборки проекта

Особенности XC

Операторы в XC выполняются последовательно (как они находятся в C), так что в исполнении:

f(); g();

функция g выполняется только один раз, при этом выполнение функции f завершено. Набор операторов, может выполняться параллельно с использованием *par*, так что:

par { f(); g(); }

говорит о том, что f и g должны быть выполнены одновременно. Выполнение параллельного оператора завершается только тогда, когда каждый из компонентов оператора завершился. Компонент оператора называется задачей в XC.

Так как обмен переменных может привести к условиям гонки и недетерминированному поведению, XC обеспечивает соблюдение параллельной дизъюнктности. Дизъюнктность означает, что переменная, изменяемая в одном компоненте оператора из par, не может быть использована в любой другом операторе.

Параллельные операторы могут быть написаны с репликатора, в аналогичный цикл, так что многие случаи подобных задач могут быть созданы, без того, чтобы писать каждый из них по отдельности, так что запись:

par (size_t i=0; i<4; ++i)

f(i);

эквивалентно:

par { f(0); f(1); f(2); f(3); }

Задачи в параллельном операторе выполняются созданием темы на процессоре, выполняющем оператор. Задачи могут быть размещены на различных плитах с помощью префикса tile. В следующем примере:

```
par {
  on tile[0] : f();
  par (size_t i=0; i<4; ++i)
    on tile[1].core[i] : g();
}
```

задача f находится на любом доступном ядре плиты 0, а экземпляры задач g размещены на 0, 1, 2 и 3 ядре плиты 1. Размещением задач ограничивается основная функция программы XC. Концептуально это потому, что, когда программа XC составлена, она разделяется на своем верхнем уровне, в отдельно исполняемые программы для каждой плиты.

Коммуникации

Параллельные задачи могут общаться друг с другом, используя интерфейсы или каналы связи.

Интерфейс определяет набор типов транзакций, где каждый тип определяется как функция с параметром и возврата типов. Когда две задачи соединяются через интерфейс, то один работает в качестве сервера, а другой в качестве клиента. Клиент имеет возможность инициировать транзакцию с соответствующего сервера, с синтаксисом, аналогичным вызову обычной функции. Это взаимодействие можно рассматривать как удаленный вызов процедур.

Каналы связи обеспечивают более простой способ общения между задачами, чем интерфейсы. Канал соединяет две задачи и позволяет им получать и отправлять данные, используя в <: и из:> операторы соответственно. Связь осуществляется только тогда, когда вход соответствует с выходом, и при этом каждая сторона ожидает готовности, это также синхронизирует задачи. Например:

```
chan c;
int x;
par {
  c <: 42;
  c :> x;
}
```

Обработка событий

Выражение дожидается появления события для выбора (состояния). Это похоже на процесс чередования в языке программирования параллельных процессов «Occum». Каждый компонент выбирает событие, например отклик интерфейса, сигналы входа канала или входного порта и связанное с ним действие. Когда выбор сделан, он ждет, пока первое событие не включится, а затем выполняет действия соответствующие этому событию. В следующем примере:

```
select {
  case left :> v:
    out <: v;
    break;
  case right :> v:
    out <: v;
    break;
}
```

Выбранный оператор «сливает» данные из левого и правого каналов на вывод.

Синхронизация

Каждая плита имеет опорную частоту, которая может быть доступна с помощью переменных таймера. Выполнение операции вывода на таймер считывает текущее время в циклах. Например, для расчета времени выполнения функции f:

```
timer t;
uint32_t start, end;
t :> start;
f();
t :> end;
printf("Elapsed time %u s\n", (end-start)/CYCLES_PER_SEC);
```

где CYCLES_PER_SEC определяется как количество циклов в секунду.

Таймеры могут быть также использованы в некоторых операторах, чтобы вызывать события.

Порты ввода вывода

Переменные типа порта обеспечивают доступ к пинам ввода/вывода на устройства xCORE в XC. Порты имеют формат кратный степени двойки, что

позволяет иметь одно и то же число бит для ввода и вывода в каждом цикле. Здесь используются те же входные и выходные операторы : <: и:>.

Следующая программа постоянно считывает значение в один порт и выводит его на другой:

```
#include <xs1.h>
in port p = XS1_PORT_1A;
out port q = XS1_PORT_1B;
int main (void) {
  bool b;
  while (1) {
    p :> b;
    q <: b;
  }
}
```

Мультиплексирование задач на ядрах

По умолчанию, каждая задача отображается в одном ядре. Так как количество ядер ограничено (восемь на плите в современных устройствах xCORE), XC предоставляет два способа расположения задач по ядрам и позволяет более эффективно использовать имеющиеся ядра.

Задачи сервера, которые состоят из бесконечного цикла содержащего оператор выбора, могут быть помечены как комбинируемые с атрибутом [[combinable]]. Это позволяет компилятору объединить две или более комбинируемые задачи для запуска на одном и том же ядре, путем слияния в один выбор (select).

Комбинируемые задачи, за исключением случая обработки функции транзакции, могут быть отмечены атрибутом [[distributable]] - распределяемый. Это позволяет компилятору преобразовать отдельные случаи в местные вызовы функций.

Доступ к памяти

XC имеет две модели доступа к памяти: безопасные и небезопасные. Безопасный доступ установлен по умолчанию, который проверяет что сделано следующее:

- доступ к памяти не происходит за пределами их границ;
- не созданы псевдонимы памяти;
- не созданы висячие указатели.

Эти гарантии будут достигнуты через комбинации различных видов указателей (ограниченных, сглаженных, подвижных), статической проверки во время компиляции и проверок во время выполнения программы.

2.8 HTML и CSS

HTML [15] (от англ. HyperText Markup Language — «язык гипертекстовой разметки») — стандартный язык разметки документов во Всемирной паутине. Большинство веб-страниц содержат описание разметки на языке HTML (или XHTML). Язык HTML интерпретируется браузерами, полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства.

Язык HTML является приложением SGML (стандартного обобщённого языка разметки) и соответствует международному стандарту ISO 8879.

Язык XHTML является более строгим вариантом HTML, он следует всем ограничениям XML и, фактически, XHTML можно воспринимать как приложение языка XML к области разметки гипертекста.

Во всемирной паутине HTML-страницы, как правило, передаются браузерам от сервера по протоколам HTTP или HTTPS, в виде простого текста или с использованием шифрования.

Язык HTML был разработан британским учёным Тимом Бернерсом-Ли приблизительно в 1986—1991 годах в стенах ЦЕРНа в Женеве в Швейцарии. HTML создавался как язык для обмена научной и технической документацией, пригодный для использования людьми, не являющимися специалистами в области вёрстки. HTML успешно справлялся с проблемой сложности SGML путём определения небольшого набора структурных и семантических элементов — дескрипторов. Дескрипторы также часто называют «тегами». С помощью HTML можно легко создать относительно простой, но красиво оформленный документ. Помимо упрощения структуры документа, в HTML внесена поддержка гипертекста. Мультимедийные возможности были добавлены позже.

Изначально язык HTML был задуман и создан как средство структурирования и форматирования документов без их привязки к средствам воспроизведения (отображения). В идеале, текст с разметкой HTML должен был без стилистических и структурных искажений воспроизводиться на оборудовании с различной технической оснащённостью (цветной экран современного компьютера, монохромный экран органайзера, ограниченный по размерам экран мобильного телефона или устройства и программы голосового воспроизведения текстов). Однако современное применение HTML очень далеко от его изначальной задачи. Например, тег <TABLE> предназначен для

создания в документах таблиц, но часто используется и для оформления размещения элементов на странице. С течением времени основная идея платформонезависимости языка HTML была принесена в жертву современным потребностям в мультимедийном и графическом оформлении.

HTML — теговый язык разметки документов. Любой документ на языке HTML представляет собой набор элементов, причём начало и конец каждого элемента обозначается специальными пометками — тегами. Элементы могут быть пустыми, то есть не содержащими никакого текста и других данных (например, тег перевода строки
). В этом случае обычно не указывается закрывающий тег. Кроме того, элементы могут иметь атрибуты, определяющие какие-либо их свойства (например, размер шрифта для элемента font). Атрибуты указываются в открывающем теге.

CSS [16] (англ. Cascading Style Sheets — каскадные таблицы стилей) — формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

Преимущественно используется как средство описания, оформления внешнего вида веб-страниц, написанных с помощью языков разметки HTML и XHTML, но может также применяться к любым XML-документам, например, к SVG или XUL.

CSS используется создателями веб-страниц для задания цветов, шрифтов, расположения отдельных блоков и других аспектов представления внешнего вида этих веб-страниц. Основной целью разработки CSS являлось разделение описания логической структуры веб-страницы (которое производится с помощью HTML или других языков разметки) от описания внешнего вида этой веб-страницы (которое теперь производится с помощью формального языка CSS). Такое разделение может увеличить доступность документа, предоставить большую гибкость и возможность управления его представлением, а также уменьшить сложность и повторяемость в структурном содержимом. Кроме того, CSS позволяет представлять один и тот же документ в различных стилях или методах вывода, таких как экранное представление, печатное представление, чтение голосом (специальным голосовым браузером или программой чтения с экрана), или при выводе устройствами, использующими шрифт Брайля.

Правила CSS пишутся на формальном языке CSS и располагаются в таблицах стилей, то есть таблицы стилей содержат в себе правила CSS. Эти таблицы стилей могут располагаться как в самом веб-документе, внешний вид которого они описывают, так и в отдельных файлах, имеющих формат CSS. (По сути, формат CSS — это обычный текстовый файл. В файле .css не содержится ничего, кроме перечня правил CSS и комментариев к ним.)

То есть, эти таблицы стилей могут быть подключены, внедрены в описываемый ими веб-документ четырьмя различными способами:

- когда таблица стилей находится в отдельном файле, она может быть подключена к веб-документу посредством тега <link>, располагающегося в этом документе между тегами <head> и </head>. (Тег <link> будет иметь атрибут href, имеющий значением адрес этой таблицы стилей). Все правила этой таблицы действуют на протяжении всего документа;
- когда таблица стилей находится в отдельном файле, она может быть подключена к веб-документу посредством директивы @import, располагающейся в этом документе между тегами <style> и </style> (которые, в свою очередь, располагаются в этом документе между тегами <head> и </head>) сразу после тега <style>, которая также указывает (в своих скобках, после слова url) на адрес этой таблицы стилей. Все правила этой таблицы действуют на протяжении всего документа;
- когда таблица стилей описана в самом документе, она может располагаться в нём между тегами <style> и </style> (которые, в свою очередь, располагаются в этом документе между тегами <head> и </head>). Все правила этой таблицы действуют на протяжении всего документа;
- когда таблица стилей описана в самом документе, она может располагаться в нём в теле какого-то отдельного тега (посредством его атрибута style) этого документа. Все правила этой таблицы действуют только на содержимое этого тега.

В первых двух случаях говорят, что к документу применены внешние таблицы стилей, а во вторых двух случаях — внутренние таблицы стилей.

Для добавления CSS к XML-документу, последний должен содержать специальную ссылку на таблицу стилей.

ГЛАВА 3. РАЗРАБОТКА

3.1 Описание

Наше устройство должно будет формировать сигналы управления для телескопа РТТ-150, и выдавать сигнал, во время длительности экспозиции, о том, что они еще идут. Задаваться параметры будут в окне браузера, поэтому используется интерфейс ethernet. Временная привязка осуществляется при помощи внешнего сигнала от GPS. Устройство, с заданными параметрами, будет ждать старт-бита от телескопа для начала работы.

На рис. 19, показана логика подключения устройства.

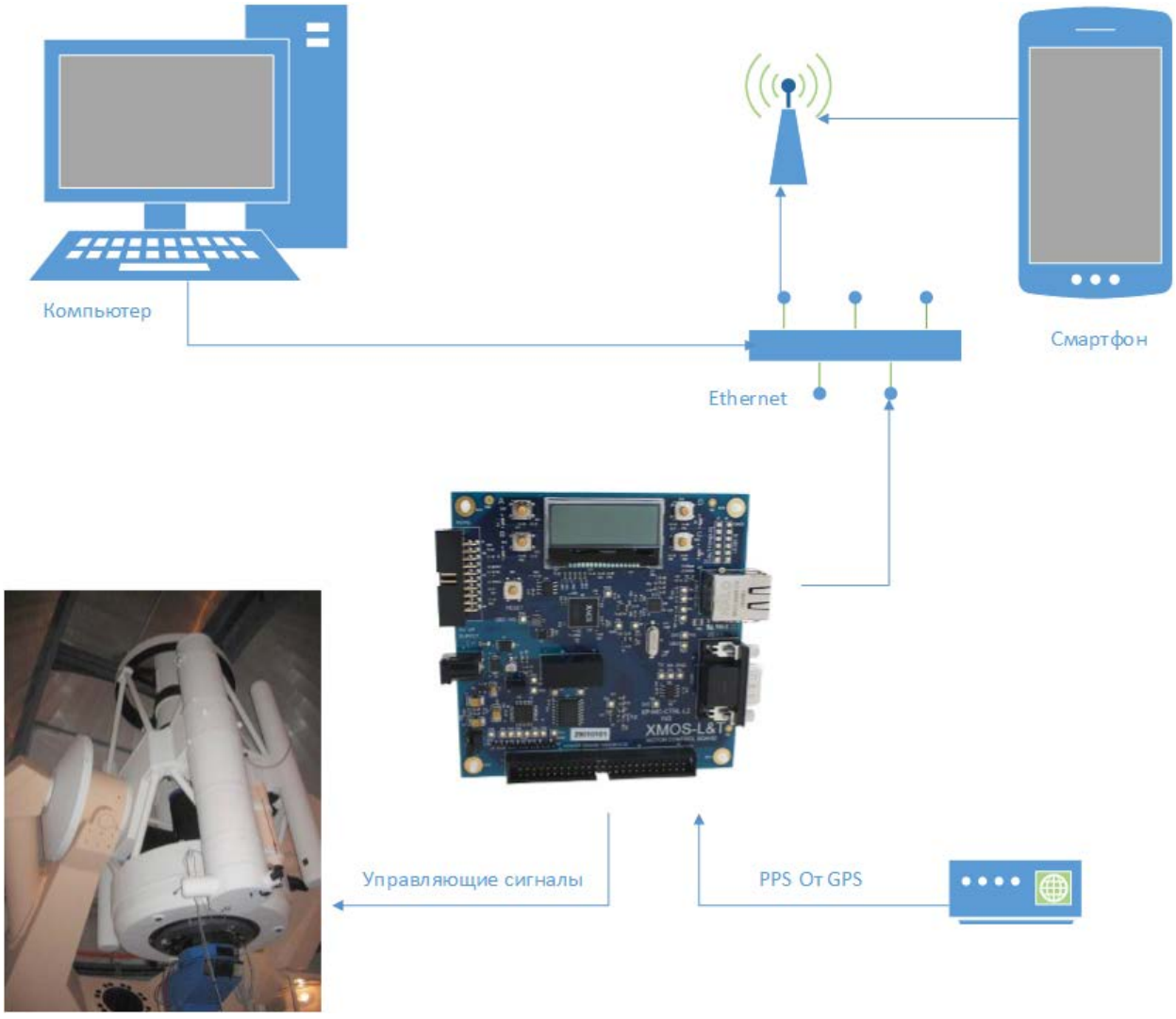


Рис. 19. Схема подключения

3.2 Алгоритм работы

До написания программного кода, определимся с логикой работы программы и нарисуем блок-схему работы устройства.

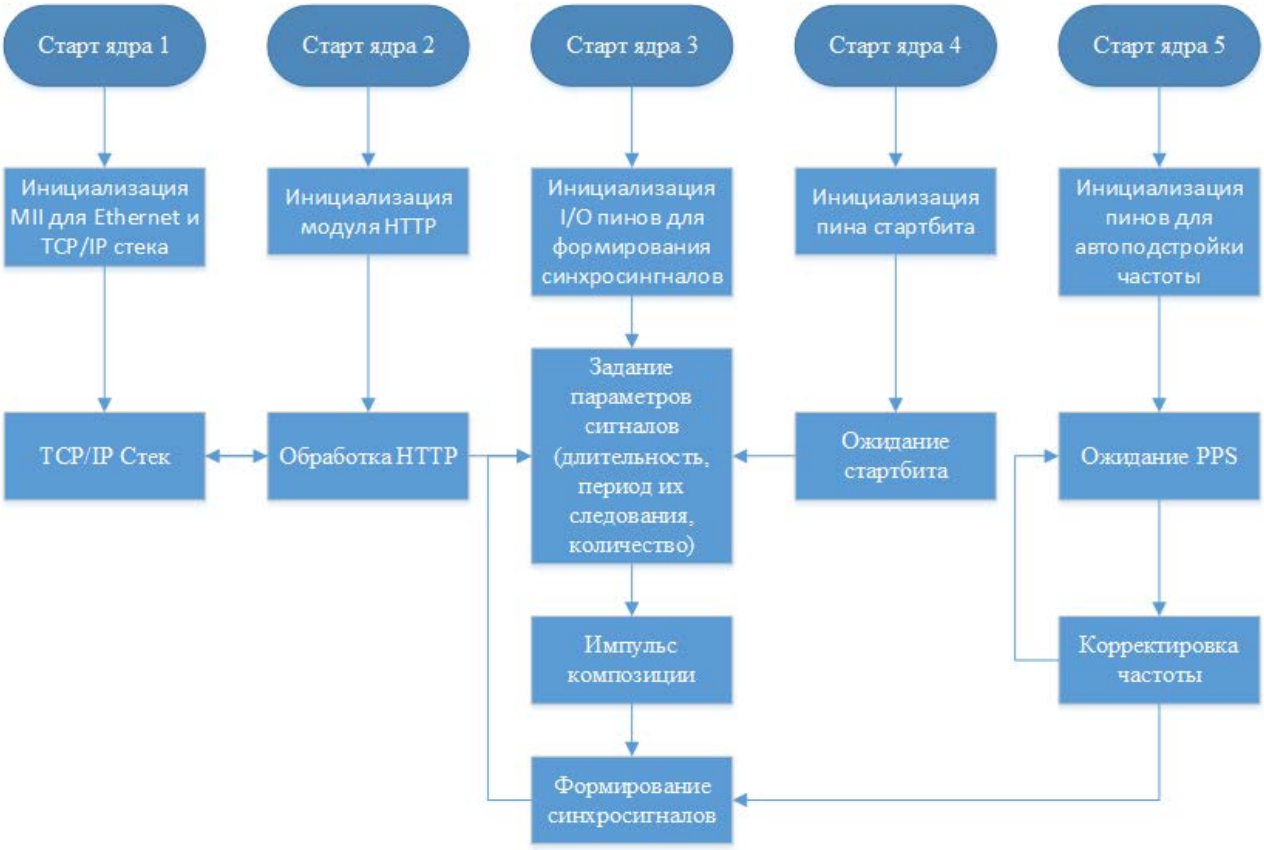


Рис. 20. Блок-схема устройства

Логика работы следующая. Стартуют ядра, на которых выполняются отдельные задачи. Ядро первое отвечает за инициализацию MII для Ethernet и работу TCP/IP стека. Второе ядро инициализирует модуль HTTP и потом обрабатывает HTTP протокол, то есть здесь у нас реализуется html-страница, на которой мы задаем параметры. Третье ядро получив по каналу данные со второго ядра, ждет старт-бита с четвертого ядра. Получив его, начинает формировать импульс композиции, а также сами синхросигналы для телескопа. На пятом ядре должна быть реализована фазовая автоподстройка частоты, которая считывает сигнал 1 PPS с GPS приемника, корректирует частоту и посылает данные, о корректировке частоты, на третье ядро. По окончании композиции импульсов, программа возвращается в исходное состояния ожидания данных и старт бита. Если не пришли новые данные с html странички, то при появления старт-бита начнется вновь формироваться композиция синхроимпульсов, с предыдущими параметрами.

3.3 Программный код

Код написан в соответствии с блок схемой приведенной на рис. 20, реализация веб-интерфейса выполнена на основе кода Simple HTTP Demo [17] который поставляется с инструментом xSOFTip. Он использует блоки технологий ethernet, tcp/ip [18] и реализует простой веб-сервер.

Так как код Simple HTTP Demo написан для набора плат SliceKit, а мы используем XP-MC-CTRL-L2, код пришлось модернизировать.

Этапы написания кода:

- откроем xTIMEcomposer studio, во вкладке xSOFTip найдем приложение Simple HTTP Demo, импортируем его к себе в окно проектов. Далее мы видим, что появилась 1 папка с проектом и 6 папок с модулями, которые задействуются в данном проекте;
- открыв папку с проектом можно увидеть файлы документации, папку подключаемых библиотек, папку с кодом программы, папку "installed targets", makefile;
- в файле makefile записано, какая используется плата, какие модули подключены для работы приложения. Мы видим, что указана плата SLICEKIT-L2:
  TARGET = SLICEKIT-L2
  и используются модули module_xtcp и module_ethernet_board_support:
  USED_MODULES = module_xtcp module_ethernet_board_support;
- напротив "TARGET = " укажем нашу плату, XP-MC-CTRL-L2. С помощью данной команды исполняемый файл будет искать установленные платы в папке installed targets. Так как там файла для нашей платы нет, закинем папку с ним туда:
  C:\Program Files (x86)\XMOS\xTIMEcomposer\Community_13.2.2\targets;
- папка должна содержать файл с расширением xn. Далее мы, в коде программы, объявляем порт PORT_SHARED_RS и напишем функцию сброса порта Ethernet;

```
//ETH reset port
on stdcore[0] : out port p_shared_rs=PORT_SHARED_RS;

void reset_devices()
{
    timer t;
    unsigned ts;
    p_shared_rs <: 0;
    t :> ts;
    t when timerafter(ts+10000000) :> void;
    p_shared_rs <: 0x2;
}
```

- добавим её в главную функцию main, а также заменим везде tile[] на stdcore[];

```
int main(void) {
       chan c_xtcp[1];

      par
      {
          on stdcore[0]: reset_devices();
          // The main ethernet/tcp server
          on ETHERNET_DEFAULT_TILE:
             ethernet_xtcp_server(xtcp_ports,
                                  ipconfig,
                                  c_xtcp,
                                  1);


          // The webserver
          on stdcore[0]: xhttpd(c_xtcp[0]);

      }
      return 0;
}
```

- однако этого не достаточно чтобы программа скомпилировалась. Дело в том что, по каким то причинам, не был выпущен обновленный xn файл для нашего устройства, в котором учитывалась бы унификация для объявления задач на ядрах посредством функции tile[]. Поэтому, используя старый файл, нам надо везде писать функцию stdcore[], а также изменить в библиотеке ethernet_board_support в папке XP-MC-CTRL-L2 в файле ethernet _board_conf.h;

```
#ifndef __ethernet_board_defaults_h__
#define __ethernet_board_defaults_h__

#define ETHERNET_DEFAULT_TILE stdcore[0] //заменили вместо tile[0]
#define ETHERNET_DEFAULT_PHY_ADDRESS 0x0

#endif // __ethernet_board_defaults_h__
```

- компилируем наше приложение и записываем на нашу плату через интерфейс XTAG. После запуска, устройство выдает в консоль свой IP-адрес, вбив который в браузер мы перейдем на сгенерированную html страничку "Hello world".

3.3.1 Формирователь импульсов

Напишем функцию, которая будет формировать импульсы и выдавать их на вход. Инициализируем выходной пин, с которого будут сниматься импульсы, а также пин экспозиции:

```
on stdcore[1]: port p_out=PORT_M2_LO_C;
on stdcore[1]: port p_out2=PORT_M2_LO_B;
```

(Детальное описание функций кода и устройства, представлено в данной работе по требованию руководителя в связи необходимостью использования его в качестве технического описания).

Все используемые пины, 50 пинового разъема, показаны на рис. 21.

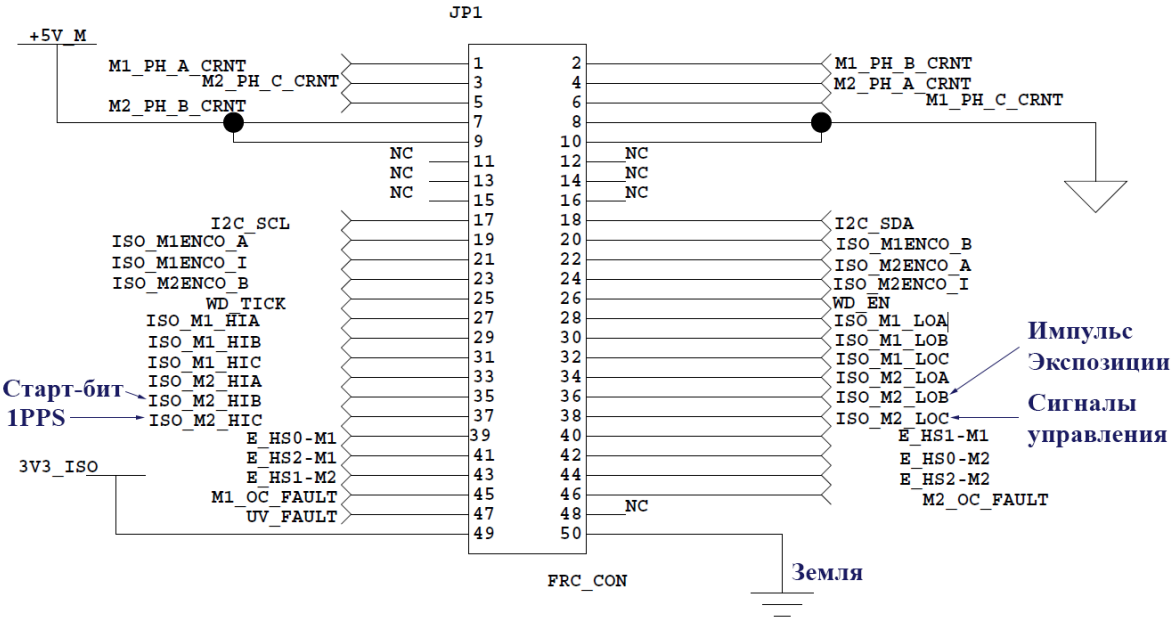


Рис. 21. 50 пиновый разъем

Суть этой функции составляет бесконечный цикл, в котором мы ожидаем выбора нескольких инструкций. Выбор первый, осуществляется передача данных по каналу c_gpio. Это наши длительность, количество и период импульсов. Расфасуем эти данные по переменным, и установим период и длительность импульсов, используя таймер XS1_TIMER_HZ. Выбор второй, пришла поправка на частоту по каналу real_hz. Выбор третий, пришел старт-бит по каналу sbit. Здесь записываем в переменную go количество символов, а также выставляем на пин экспозиции логическую единицу, тем самым сигнализируя о начале экспозиции. Выбор четвертый, переменная go не равна нулю. Здесь осуществляется формирование импульсов.

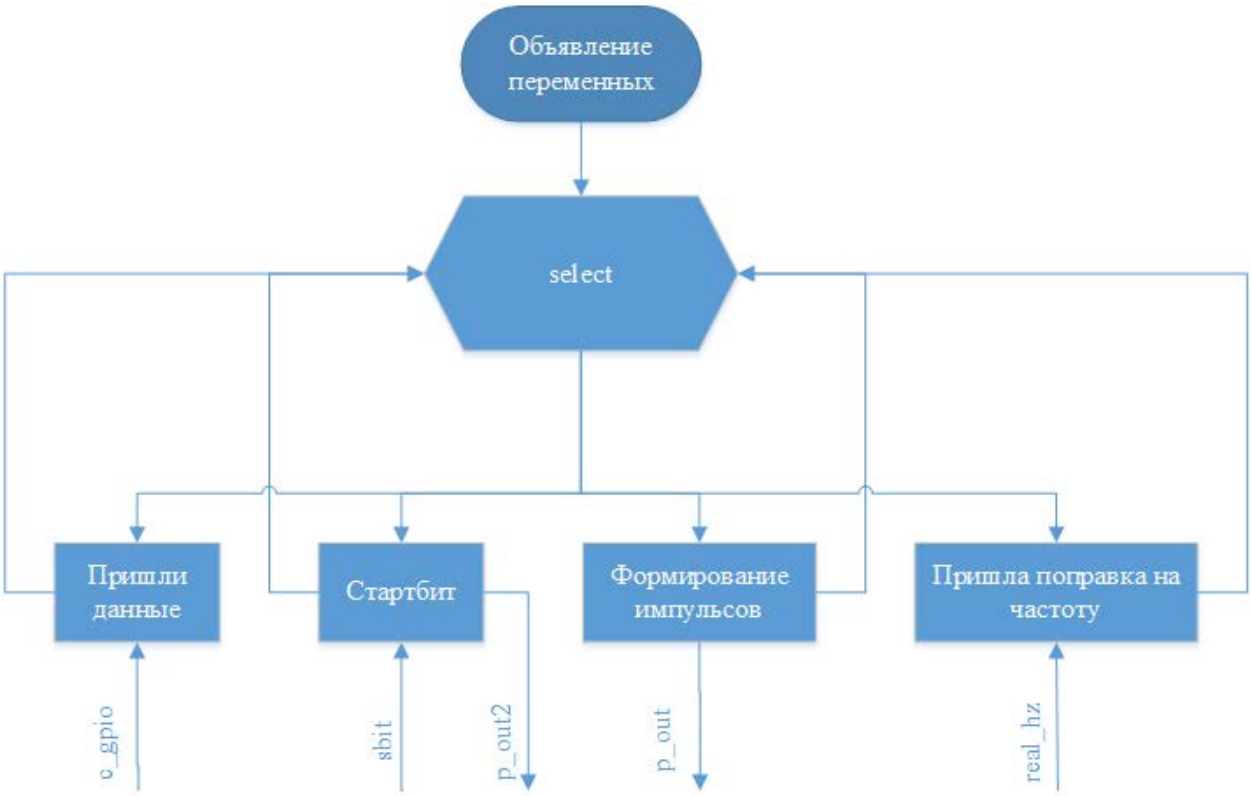


Рис. 22. Блок-схема функции app_gpio_out

```
void app_gpio_out(chanend c_gpio, chanend d, out port p_out, out port p_out2,
chanend sbit,chanend real_hz)
{
    timer time; //объявление таймера
    unsigned time_; //переменная для хранения промежуточных состояний таймера
    unsigned INTERVAL; //длительность интервала
    unsigned INTERVAL_SET; //период
    unsigned INTERVAL_RESET; //длительность импульса
    unsigned char OUT; //состояние пина
    unsigned char startbit=0;
    unsigned char PHASE; //шаг внутри одного периода
    unsigned long long int real_hz_; //частота с учетом поправок
    int go=0;
    int prego=300; //количество импульсов
    int duration=0; //длительность
    //устанавливаем начальные значения
    time :> time_;
```

Умножение на 2,5 происходит потому, что опорная частота плиты 1 равна 250 МГц.

```
    INTERVAL_RESET=2.5*(XS1_TIMER_HZ/1000)*(10);
    INTERVAL_SET=(2.5*((XS1_TIMER_HZ/1000)*(100))-INTERVAL_RESET);
    INTERVAL=INTERVAL_SET;
    PHASE=1;
    real_hz_=2.5*XS1_TIMER_HZ;
    while(1)
    {
     select {
         case c_gpio :> unsigned long long int cmd:
         if(cmd == 42) //если 42 значит пришла команда на остановку
         {
             go=1;
```

```
            PHASE=0;
        }
        else //если нет то пришли данные
        {
            duration=cmd&0xFFFF;
            printint(duration);
            printstrln(" ДЛИТЕЛЬНОСТЬ\n");
            cmd=cmd>>16;
            prego=cmd&0xFFFF;
            printint(prego);
            printstrln(" КОЛИЧЕСТВО\n");
            cmd=cmd>>16;
            printint(cmd);
            printstrln(" ПЕРИОД\n");
            INTERVAL_RESET=2.5*(XS1_TIMER_HZ/1000)*(duration&0xFFFF);
            INTERVAL_SET=(2.5*((XS1_TIMER_HZ/1000)*(cmd&0xFFFF))-
INTERVAL_RESET);
        }
        break;
        //пришла новая поправка на частоту
        case real_hz :> unsigned long long int real_hz__:
        real_hz_=real_hz__;
        time:>time_;
        INTERVAL=0;
        PHASE=1;
        break;
        //пришел стартбит можно начать запуск
        case sbit :> int sbit1:
        startbit=sbit1;
        break;
        //Если стартбит пришел, ждем 1PPS и начинаем
        case d :> int cmd_d:
        if(cmd_d==0)
        {
            if(startbit)
            {
            if(prego)
            {
                go=prego;
                //prego=0;
                time:>time_;
                INTERVAL=0;
                //выставляем импульс композиции
                p_out2<:1;
            }
            startbit=0;
            }

        }
        break;
```

Изменяем состояние пина, если пришло время.

```
        case go => time when timerafter(time_+INTERVAL):>void:
        time:>time_; //записываем текущий момент времени
        //смотрим состояние пина и меняем его
        if(PHASE)
        {
            PHASE=0;
            INTERVAL=(real_hz_*INTERVAL_RESET)/(250000000);
            OUT=1;
        }
        else
```

```
        {
            PHASE=1;
            INTERVAL=(real_hz_*INTERVAL_SET)/(250000000);
            OUT=0;
            go--;
            //если закончились импульсы
            if(!go)
            {
                OUT=0;
                //убираем импульс композиции
                p_out2<:0;
            }
        }
        p_out<:OUT;
        break;
     }
    }
}
```

3.3.2 Интерфейс

После того как мы получили IP-адрес и ввели его в окно браузера, мы должны получить страничку с тремя полями "Период импульсов", "Количество", "Длительность импульсов" и кнопкой "отправить". После нажатия на кнопку "отправить" должны получить новую страничку "Данные переданы" и кнопку "остановить". При нажатии на эту кнопку импульсы прекращаются, а в браузере вновь появляется первая страница.

Пишем обе страницы на html, с использованием css. Пример первой страницы:

```
<!DOCTYPE html>
<html><head><meta http-equiv="Content-Type" content="text/html;
charset=windows-1251"><title>Задание параметров</title>
<style type="text/css">
      div {
            width: 250px;
            height: 22px;
            padding: 5px;
      }
</style>
</head>
<body style="align-content: center;">

      <img
src="http://hsto.org/getpro/habr/post_images/5c1/050/fa3/5c1050fa3eb9a7c0
7663b481881e0577.png" alt="zagolovok" style="max-width: 50px;">
      <form name="form" action="" method="post" style="height: 250px;
position: relative;">
            <div style="background-color: #7777FF; height: 100%; border-
radius: 20px;">

                  <div>Период между импульсами:</div>
                  <div>
                        <input type="text" name="period" size="20"> мс</p>
                  </div>
                  <div>Количество импульсов:</div>
                  <div>
                        <input type="text" name="count" size="20">
```

```
                </div>

                <div>Длительность импульсов:</div>

                <div>
                     <input type="text" name="duration" size="20">
                </div>

                <div style="position: relative; top: 20px;">
                     <input type="submit" value="Отправить"
style="height: 25px; width: 187px;">
                </div>
           </div>
     </form>
</body>
</html>
```

Страницы мы помещаем в char page[] и char page2[], для этого каждую строку надо поместить в кавычки “ “, а также добавляем к каждой такой кавычки, имеющейся в строке, обратный слэш \, чтобы не нарушать синтаксис. Например:

```
"<form name=\"form\"action=\"\"method=\"post\">"
```

Функция void_httpd_recv получив HTTP запрос, вызывает функцию parse_http_request для анализа полученных данных. Поэтому нам надо модернизировать функцию parse_http_request, а именно реализовать процесс считывания отправленной информации с первой страницы, и считывание нажатия кнопки, об остановки экспозиции импульсов, со второй страницы.

```
// Parses a HTTP request for a GET
void parse_http_request(httpd_state_t *hs, char *data, int len)
{
    volatile char * xx;
    volatile unsigned long long int i;
    volatile int j;
    volatile unsigned long long int count;
    volatile unsigned long long int duration;
  // Return if we have data already
  if (hs->dptr != NULL)
    {
      return;
    }

  // Test if we received a HTTP GET request
  if (strncmp(data, "GET ", 4) == 0)
    {
      // Assign the default page character array as the data to send
      hs->dptr = &page[0];
      hs->dlen = strlen(&page[0]);
    }
```

Получив HTTP запрос GET, мы ищем совпадение строк period, count и duration в наших данных. Если таких строк нет, значит пришла форма данных со второй страницы, а это значит, что пора остановить генерацию импульсов. Здесь также реализуется начальная генерация первой страницы.

```
else
  {
    xx=strstr(data, "period");
    if(xx != NULL)
    {

        xx+=strlen("period=");
        i=0;
        j=0;
        while((*xx>='0')&&(*xx<='9')&&(j<5))
        {
            j++;
            i=i*10;
            i+=(*xx-'0');
            xx++;
        }
        if(i>0xFFFF)
            i=0xFFFF;
        i=i*0x10000;

        xx=strstr(data,"count");
        xx+=strlen("count=");
        count=0;
        j=0;
        while((*xx>='0')&&(*xx<='9')&&(j<5))
        {
            j++;
            count=count*10;
            count+=((*xx)-'0');
            xx++;
        }
        i|=(count&0xFFFF);
        xx=strstr(data,"duration");
        xx+=strlen("duration=");
        duration=0;
        j=0;

               while((*xx>='0')&&(*xx<='9')&&(j<5))
                  {
                      j++;
                      duration=duration*10;
                      duration+=((*xx)-'0');
                      xx++;
                  }
```

Данные о периоде, количестве и длительности импульсов записываются в одну переменную i, чтобы отправить их по каналу c_gpio на ядро формирования сигналов.

```
        i=i*0x10000;
        i|=(duration&0xFFFF);

        set_gpio_state(c_gpio,i);
        hs->dptr=&page2[0];
        hs->dlen=strlen(&page2[0]);
    }
```

Данные о прекращении сигналов также посылаются по этому каналу. Реализовано это следующим образом, мы отправляем число 42 на ядро формирования, и если число 42 то записываем в переменную go единицу, а PHASE устанавливаем в ноль, что обеспечит остановку импульсов. Можно не

бояться не корректного поведения устройства, если число 42 придет в виде информации о сигналах, это будет лишь означать что период и количество их равно нулю, что эквивалентно тому, что никаких импульсов не задано.

```
        else
        {
        i=42;
        set_gpio_state(c_gpio,i);
        hs->dptr=&page[0];
        hs->dlen=strlen(&page[0]);
        }
        // We did not receive a get request, so do nothing
    }
}
```

На рис. 23, показаны страницы, для ввода параметров и для остановки импульсов соответственно.

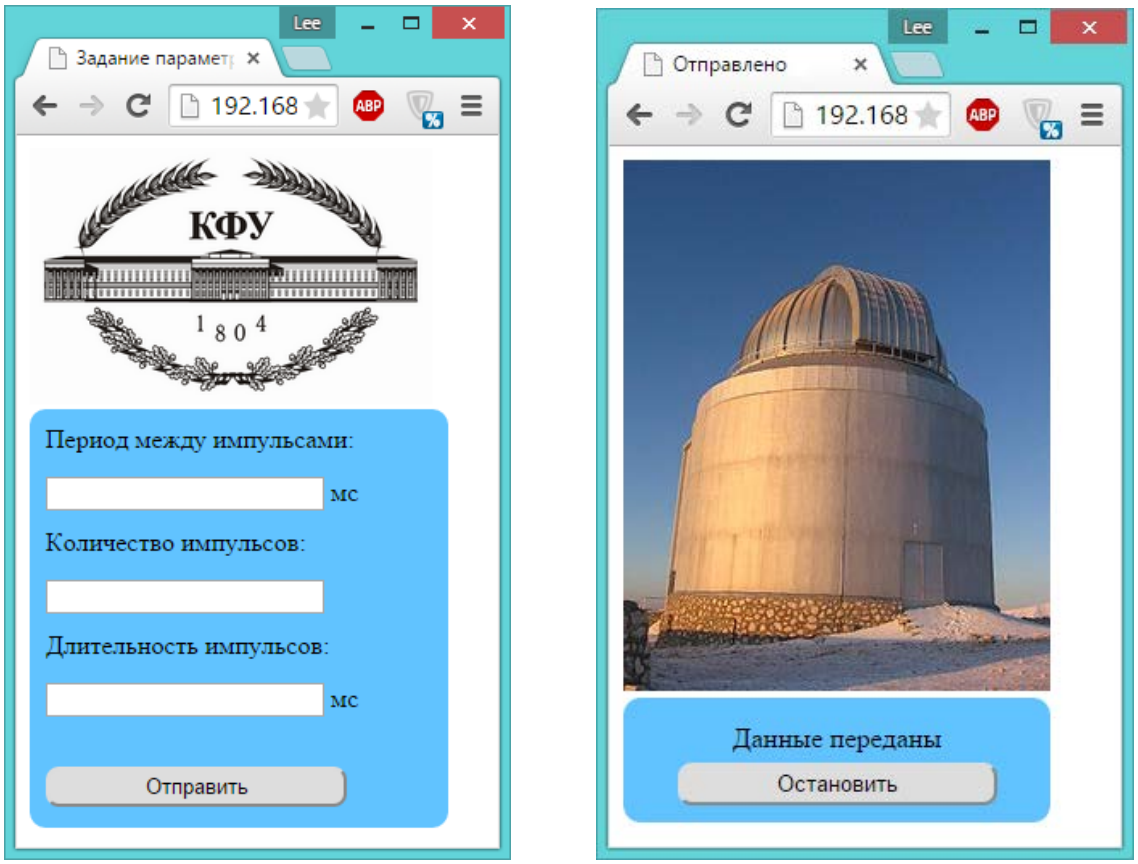


Рис. 23. Страницы ввода параметров и остановки импульсов

Выбор управления, используя передачу команд по протоколу HTTP, был выбран в результате следующих соображений:

1. отсутствие платформенной привязки;
2. отсутствие привязки к операционной системе на верхнем уровне;
3. не требуется установка программного обеспечения;
4. легкость в использовании.

3.3.3 ФАПЧ и временная привязка

Данная функция реализует подсчитывание времени между секундными метками 1PPS, и на основе этого формирует частоту, с которой надо формировать импульсы.

Инициализируем вывод, с которого будем считывать 1PPS:

```
on stdcore[1]: port p_in=PORT_M2_HI_C;
```

Если импульс 1PPS начался, то берем значение, которое насчитал таймер, затем высчитываем время, которое таймер насчитал за время между 1PPS. Поправку отправляем на другое ядро. Если явно ошибочное значение насчитал таймер, например, был пропущен импульс, то нужно отправить предыдущее значение. После этого записываем новое значение для нового отсчета импульса.

На рис. 24 показана блок-схема функции app_fapch.

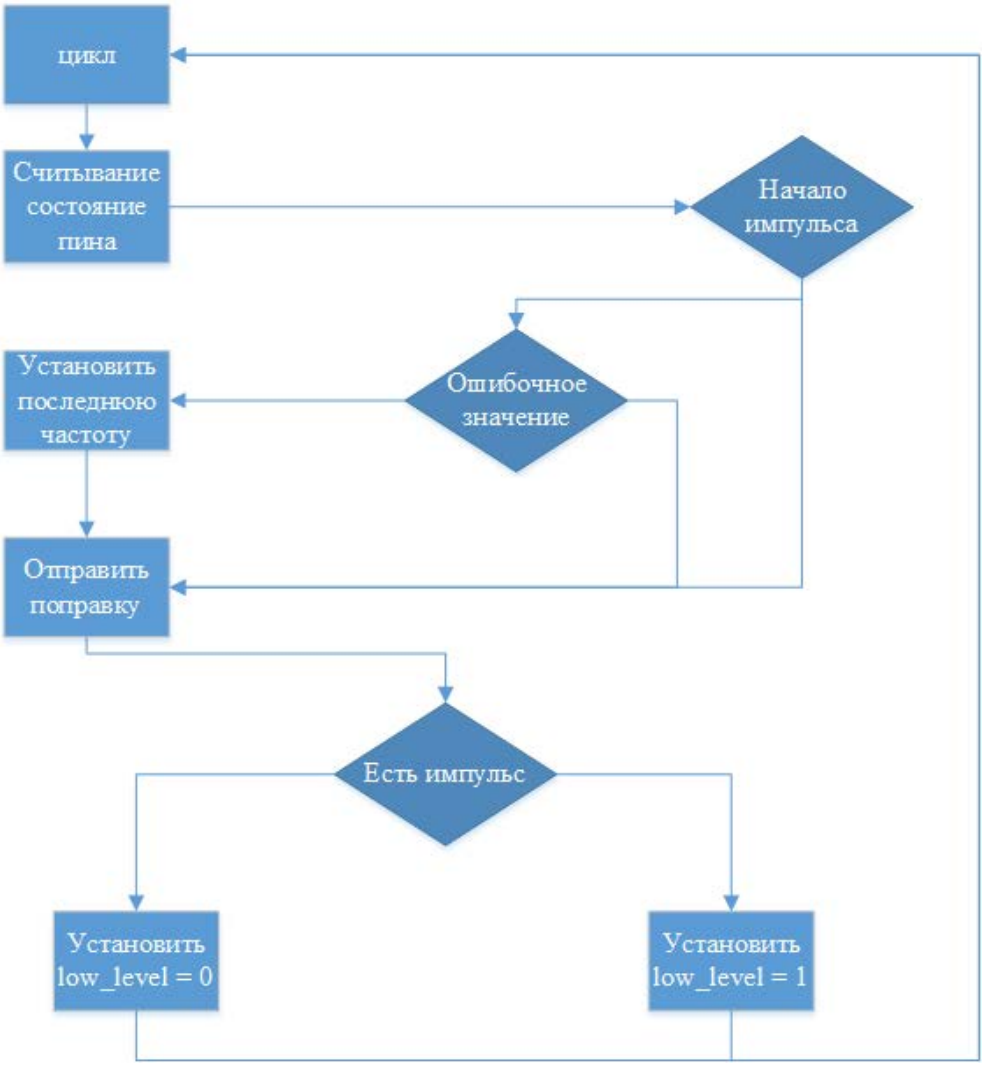


Рис. 24. Блок-схема функции app_fapch

```
void app_fapch(in port p_in, chanend d, chanend real_hz)
{
    timer time_fapch; //таймер для автоподстройки
    unsigned time_o; //начало интервала
    unsigned time_t; //конец интервала
    unsigned long long int realfapch;
```

```
    unsigned long long int realfapch_s;
    int pps;
    int low_level=1;
    time_fapch :> time_o;
    time_t=time_o+250000000;
    realfapch=250000000;
    realfapch_s=250000000;
    while(1)
    {
        //считываем состояние пинов
        p_in:>pps;
        d<:pps;
        if(((low_level)||(pps))==0)
        {
            time_fapch:>time_t; //берем значение которое насчитал таймер
            realfapch=time_t; //высчитываем сколько таймер насчитал
            realfapch-=time_o;
            //если явно ошибочные значения, то ошибка
            if( ( realfapch > 500000000 ) || ( realfapch < 125000000 ) )
            {
                realfapch=realfapch_s;
            }
            real_hz<:realfapch; //отправить поправку
            time_fapch:>time_o;
            //запоминаем корректное значение частоты
            realfapch_s=realfapch;
        }
        if(pps) //есть импульс 1PPS или нет
        {
            low_level=0;
        }
        else
        {
            low_level=1;
        }
    }
}
```

3.3.4 Функция запуска

Когда данные заданы через html страничку и устройство готово к работе, оно должно дожидаться старт-бита от телескопа чтобы начать выдавать экспозицию импульсов.

Для начала надо инициализировать вывод, с которого будем считывать старт-бит:

```
    on stdcore[1]: port p_start_bit=PORT_M2_HI_B;

void start_bit(in port p_start_bit, chanend sbit)
{
    int sb;
    while(1)
    {
        p_start_bit:>sb;
        if(sb)
        {
            sbit<:sb;
        }
    }
}
```

3.3.5 Функция main

Основная функция main показана на листинге. В начале функции объявляются все используемые каналы, для связи наших функций. Далее идет функция par, которая объявляет параллелизм. В ней следуют выполняемые функции на отдельных ядрах.

Следует обратить внимание на то, что одни функции используют stdcore[0], а другие stdcore[1]. Это связано с тем, что управление периферией распределено между ядрами плит. Так, например, пины ввода/вывода контролируются ядрами первой плиты, а Ethernet, TCP/IP стек и HTTP контролируются ядрами нулевой плиты.

```
// Program entry point
int main(void) {
       chan c_xtcp[1];
       chan c_gpio;
       chan sbit;
       chan d;
       chan real_hz;

      par
      {
          on stdcore[0]: reset_devices();
          // The main ethernet/tcp server
          on ETHERNET_DEFAULT_TILE:
             ethernet_xtcp_server(xtcp_ports,
                                  ipconfig,
                                  c_xtcp,
                                  1);


          // Вебсервер
          on stdcore[0]: xhttpd(c_xtcp[0],c_gpio);
          // Формирователь импульсов
          on stdcore[1]: app_gpio_out(c_gpio,d,p_out,p_out2,sbit,real_hz);
          // ФАПЧ
          on stdcore[1]: app_fapch(p_in,d,real_hz);
          // ожидание старт бита
          on stdcore[1]: start_bit(p_start_bit,sbit);
      }
      return 0;
}
```

3.4 Тестирование устройства

После разработки устройства были проведено тестирование его и сняты осциллограммы требуемых от устройства характеристик.

Осциллограммы были получены с помощью двухканального осциллографа, на первом канале наблюдается управляющие синхросигналы, на втором канале наблюдается 1PPS от GPS приемника. Уровень сигналов на осциллограммах 4 вольта для 1PPS и 3.3 вольта для синхросигналов.

Параметры работы для тестирования, были заданы следующие: период следования импульсов 1000 мс, количество импульсов 1000, длительность импульсов 10 мкс.

Первая характеристика, это уход частоты микроконтроллера от заявленной при выключенной функции автоподстройки частоты. На рис. 25 показан начальный момент времени, когда были заданы параметры и был сгенерирован первый импульс. На рис. 26, 27, 28 показаны 3,6,11 импульсы соответственно.

Видно, что каждую секунду наши импульсы откланяются от импульса 1PPS, на 25 мкс.

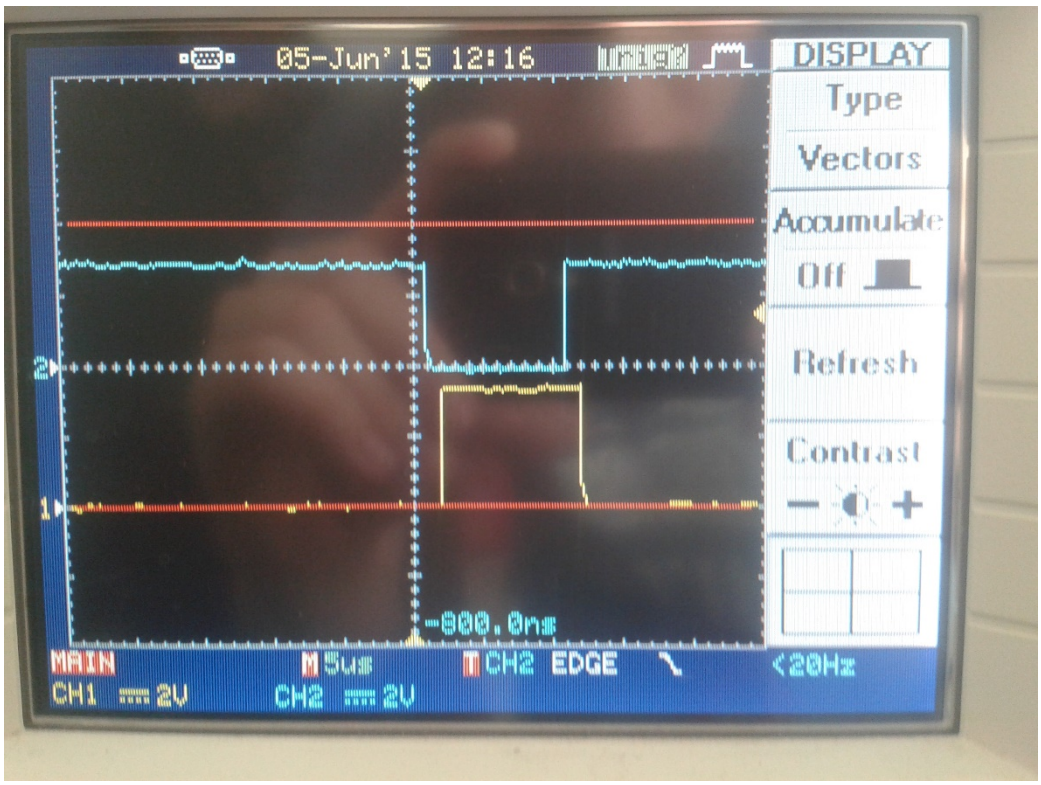


Рис. 25. Первый импульс при выключенной АПЧ

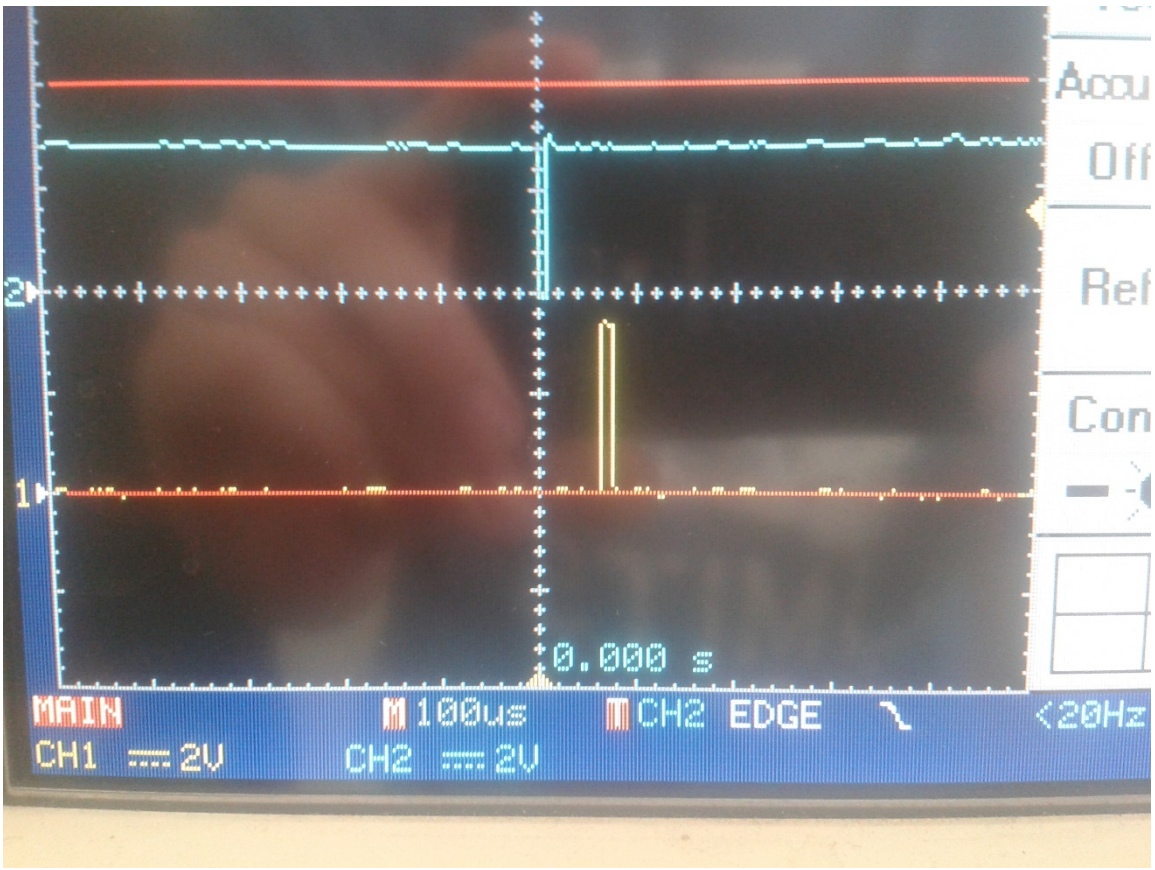


Рис. 26. Третий импульс, при выключенной АПЧ

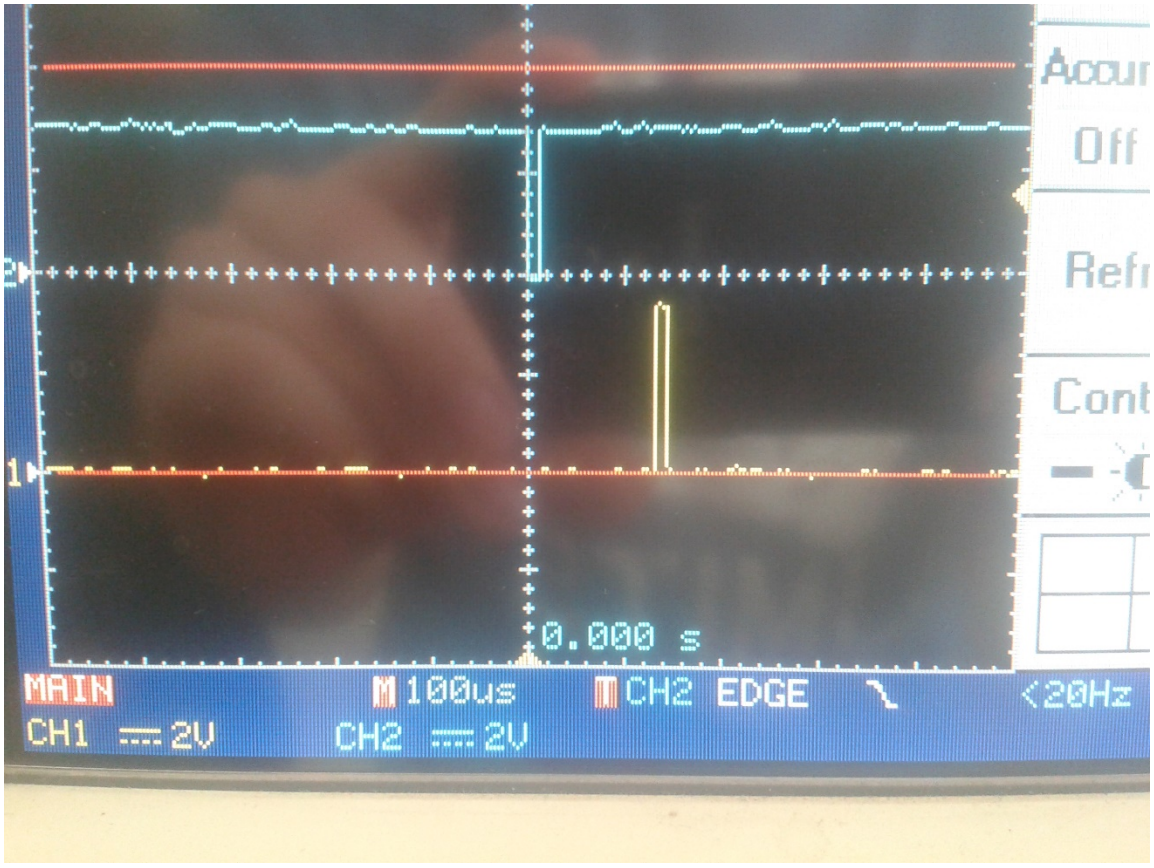


Рис. 27. Шестой импульс, при выключенной АПЧ

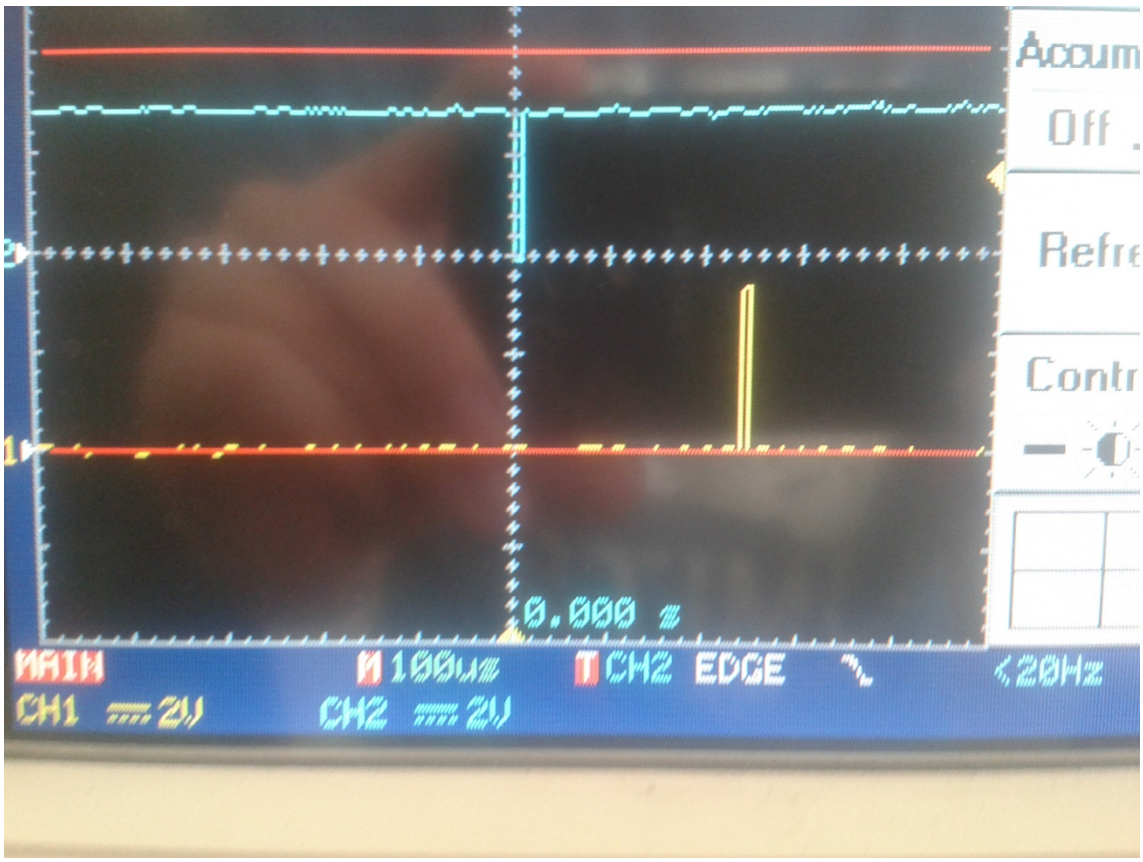


Рис. 28. Одиннадцатый импульс, при выключенной АПЧ

В результате полученных осциллограмм, мы выявили, что, при выключенной функции автоподстройки частоты, каждую секунду время на нашем устройстве отклоняется от реального всемирного времени на 25 мкс.

Далее включим функцию автоподстройки частоты и получим новые осциллограммы. В начальный момент времени картина была такая же, как на рис. 25.

На рис. 29, показана осциллограмма через минуту после запуска, а на рис. 30, последний импульс в серии.

Видно, что функция автоподстройки частоты работает.

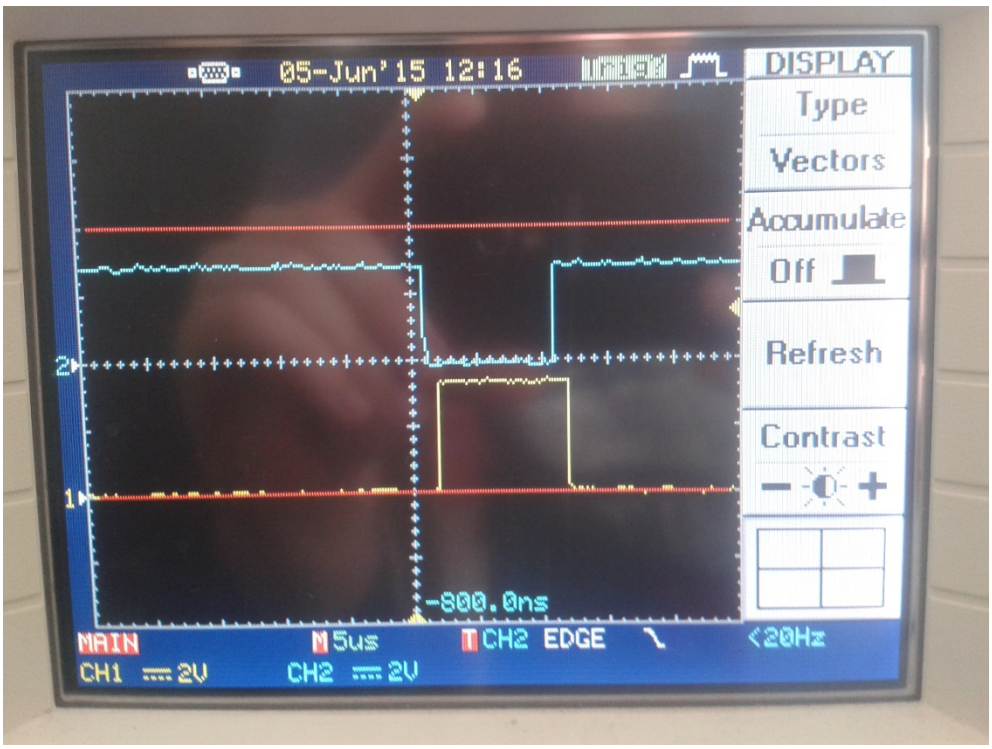


Рис. 29. Импульс спустя минуту, при включенной АПЧ

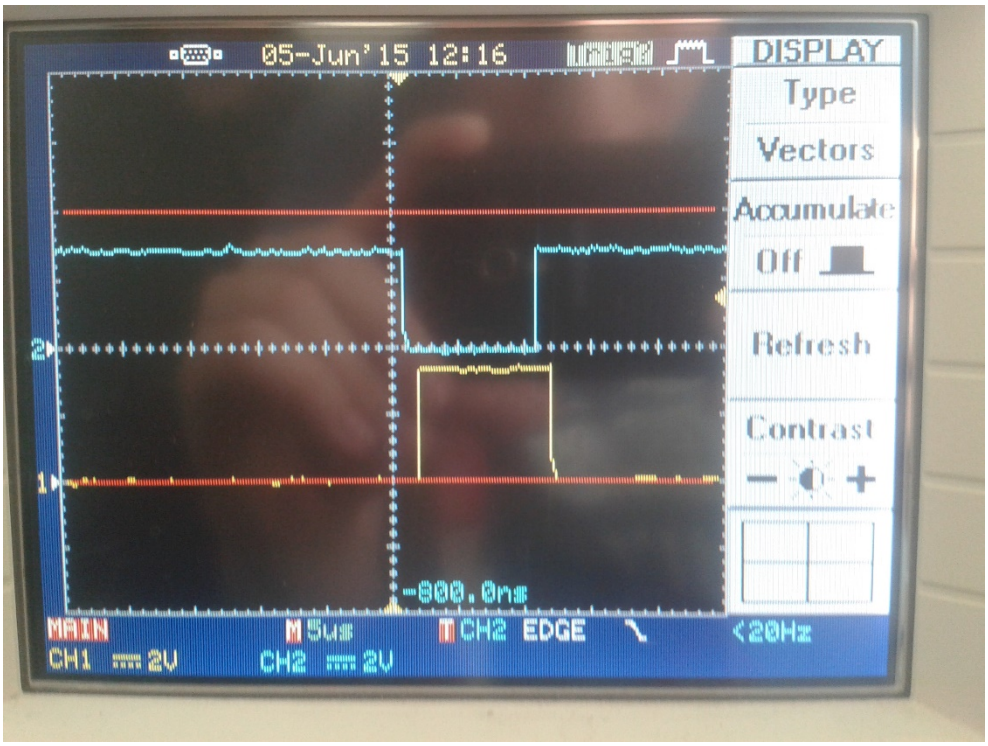


Рис. 30. Последний импульс в серии, при включенной АПЧ

ЗАКЛЮЧЕНИЕ

В результате данной работы, сделано следующее:

1. Рассмотрены цепи запуска навесного научного оборудования телескопа РТТ-150;
2. Выбрана плата XP-MC-CTRL-L2 с микроконтроллером xCORE и обширной периферией, способной выполнять поставленные задачи;
3. Изучена среда разработки выбранного нами микроконтроллера;
4. Разработан алгоритм, написан программный код, реализующий функции устройства;
5. Написан интерфейс взаимодействия с устройством на языке HTML с применением CSS;
6. Проведены лабораторные испытания разработанного устройства.

В ходе работы я обнаружил несоответствие фирменного программного модуля подключения к сети ethernet с нашим устройством XP-MC-CTRL-L2. Несоответствие заключалось в том, что не был выпущен обновленный файл платы, в котором бы учитывалась унификация для объявления задач на ядрах посредством слова tile[]. Для этого мне пришлось изменить в библиотеке ethernet_board_support, указание плиты по умолчанию для ethernet, с tile[] на stdcore[].

Поскольку информация по данным типам контроллеров и среде разработки, имеется, в основном, на сайте производителя и на английском языке, поэтому потребовалось определенное время для того чтобы корректно освоить всю необходимую информацию.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Казанский федеральный университет. Институт физики. [Электронный ресурс] Р.И. Гумеров. «Программируемые микроэлектронные системы. Лабораторный практикум. Часть II. 32-разрядные микроконтроллеры. Руководство.» – Казань: КФУ, 2014, - 62 с. - Режим доступа: http://www.kpfu.ru, свободный. - Загл. с экрана. - Яз. рус. (дата обращения 5.03.2015)
2. Сайт компании «XMOS Company» [Электронный ресурс]. XS1-L02A-QF124 Darasheet 2012. - 25 с. - Режим доступа: http://www.xmos.com, свободный. - Загл. с экрана. - Яз. англ. (дата обращения 6.03.2015)
3. Сайт компании «XMOS Company» [Электронный ресурс]. XS1-L System Specification 2012. - 35 с. - Режим доступа: http://www.xmos.com, свободный. - Загл. с экрана. - Яз. англ. (дата обращения 6.03.2015)
4. Сайт компании «XMOS Company» [Электронный ресурс]. Xmos Motor Control Solutions Overview (1.1) 2012. - 2 с. - Режим доступа: http://www.xmos.com, свободный. - Загл. с экрана. - Яз. англ. (дата обращения 7.03.2015)
5. Сайт компании «XMOS Company» [Электронный ресурс]. L2 Control Board Quick Start Guide 2012. - 4 с. - Режим доступа: http://www.xmos.com, свободный. - Загл. с экрана. - Яз. англ. (дата обращения 7.03.2015)
6. Сайт компании «XMOS Company» [Электронный ресурс]. Motor Control Platform Software Guide 2012. - 49 с. - Режим доступа: http://www.xmos.com, свободный. - Загл. с экрана. - Яз. англ. (дата обращения 7.03.2015)
7. Сайт компании «XMOS Company» [Электронный ресурс]. Motor Control - Control Board Schematics 2012. - 6 с. - Режим доступа: http://www.xmos.com, свободный. - Загл. с экрана. - Яз. англ. (дата обращения 7.03.2015)
8. Сайт компании «XMOS Company» [Электронный ресурс]. XTAG-2 Hardware Manual 2009. - 25 с. - Режим доступа: http://www.xmos.com, свободный. - Загл. с экрана. - Яз. англ. (дата обращения 9.03.2015)
9. Википедия [Электронный ресурс]. Сетевая модель OSI. - Режим доступа: https://ru.wikipedia.org/wiki/Сетевая_модель_OSI, свободный. - Загл. с экрана. - Яз. рус. (дата обращения 2.04.2015)
10. Википедия [Электронный ресурс]. Система реального времени - Режим доступа: https://ru.wikipedia.org/wiki/Система_реального_времени, свободный. - Загл. с экрана. - Яз. рус. (дата обращения 2.04.2015)

11. Сайт компании «ANDOR» [Электронный ресурс]. iKon-L Manual version 1.1 User guide 2008. - 166 с. - Режим доступа: http://www.andor.com, свободный. - Загл. с экрана. - Яз. англ. (дата обращения 3.04.2015)
12. Сайт компании «ANDOR» [Электронный ресурс]. iXon Manual version 3.6 User guide 2007. - 171 с. - Режим доступа: http://www.andor.com, свободный. - Загл. с экрана. - Яз. англ. (дата обращения 3.04.2015)
13. Сайт компании «XMOS Company» [Электронный ресурс]. xTIMEcomposer User Guide 2014. - 395 с. - Режим доступа: http://www.xmos.com, свободный. - Загл. с экрана. - Яз. англ. (дата обращения 6.04.2015)
14. Сайт компании «XMOS Company» [Электронный ресурс]. XC Programming Guide 2013. - 66 с. - Режим доступа: http://www.xmos.com, свободный. - Загл. с экрана. - Яз. англ. (дата обращения 7.04.2015)
15. Википедия [Электронный ресурс]. HTML - Режим доступа: https://ru.wikipedia.org/wiki/HTML, свободный. - Загл. с экрана. - Яз. рус. (дата обращения 9.04.2015)
16. Википедия [Электронный ресурс]. CSS - Режим доступа: https://ru.wikipedia.org/wiki/CSS, свободный. - Загл. с экрана. - Яз. рус. (дата обращения 9.04.2015)
17. Сайт компании «XMOS Company» [Электронный ресурс]. Ethernet Slice Simple Webserver Application Quickstart 2014. - 12 с. - Режим доступа: http://www.xmos.com, свободный. - Загл. с экрана. - Яз. англ. (дата обращения 17.04.2015)
18. Сайт компании «XMOS Company» [Электронный ресурс]. Ethernet TCP/IP component programming guide 2014. - 33 с. - Режим доступа: http://www.xmos.com, свободный. - Загл. с экрана. - Яз. англ. (дата обращения 17.04.2015)

main.xc

```
// Copyright (c) 2011, XMOS Ltd, All rights reserved
// This software is freely distributable under a derivative of the
// University of Illinois/NCSA Open Source License posted in
// LICENSE.txt and at <http://github.xcore.com/>

#include <platform.h>
#include <xs1.h>
#include <print.h>
#include <stdio.h>
#include "xtcp.h"
#include "ethernet_board_support.h"
#include "xhttpd.h"

//ETH reset port
on stdcore[0] : out port p_shared_rs=PORT_SHARED_RS;

void reset_devices()
{
    timer t;
    unsigned ts;
    p_shared_rs <: 0;
    t :> ts;
    t when timerafter(ts+10000000) :> void;
    p_shared_rs <: 0x2;
}

// These intializers are taken from the ethernet_board_support.h header
for
// XMOS dev boards. If you are using a different board you will need to
// supply explicit port structure intializers for these values
ethernet_xtcp_ports_t xtcp_ports =
    {on ETHERNET_DEFAULT_TILE: OTP_PORTS_INITIALIZER,
     ETHERNET_DEFAULT_SMI_INIT,
     ETHERNET_DEFAULT_MII_INIT_lite,
     ETHERNET_DEFAULT_RESET_INTERFACE_INIT};

#if IPV6
xtcp_ipconfig_t ipconfig = {
  0,
  {{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}},
  {{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}},
  {{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}},
};
#else
// IP Config - change this to suit your network.  Leave with all
// 0 values to use DHCP
xtcp_ipconfig_t ipconfig = {
  { 0, 0, 0, 0 }, // ip address (eg 192,168,0,2)
  { 0, 0, 0, 0 }, // netmask (eg 255,255,255,0)
  { 0, 0, 0, 0 }  // gateway (eg 192,168,0,1)
};
#endif
```

```
//////////////////////////////////////////////////////////////////////////
//////////////////////////////
//с этого вывода снимаем сигналы управления
on stdcore[1]: port p_out=PORT_M2_LO_C;
//сюда поступает 1PPS
on stdcore[1]: port p_in=PORT_M2_HI_C;
//стартбит
on stdcore[1]: port p_start_bit=PORT_M2_HI_B;
//экспозиция
on stdcore[1]: port p_out2=PORT_M2_LO_B;

//функция формирования синхросигналов
void app_gpio_out(chanend c_gpio, chanend d, out port p_out, out port
p_out2, chanend sbit,chanend real_hz)
{
    timer time; //объявление таймера
    unsigned time_; //переменная для хранения промежуточных состояний
таймера
    unsigned INTERVAL; //длительность интервала
    unsigned INTERVAL_SET; //период
    unsigned INTERVAL_RESET; //длительность импульса
    unsigned char OUT; //состояние пина
    unsigned char startbit=0;
    unsigned char PHASE; //шаг внутри одного периода
    unsigned long long int real_hz_; //частота с учетом поправок
    int go=0;
    int prego=300; //количество импульсов
    int duration=0; //длительность
    //устанавливаем начальные значения
    time :> time_;
    INTERVAL_RESET=2.5*(XS1_TIMER_HZ/1000)*(10);
    INTERVAL_SET=(2.5*((XS1_TIMER_HZ/1000)*(100))-INTERVAL_RESET);
    INTERVAL=INTERVAL_SET;
    PHASE=1;
    real_hz_=2.5*XS1_TIMER_HZ;
    while(1)
    {
     select {
         case c_gpio :> unsigned long long int cmd:
         if(cmd == 42) //если 42 значит пришла команда на остановку
         {
             go=1;
             PHASE=0;
         }
         else //если нет то пришли данные
         {
             duration=cmd&0xFFFF;
             printint(duration);
             printstrln(" ДЛИТЕЛЬНОСТЬ\n");
             cmd=cmd>>16;
             prego=cmd&0xFFFF;
             printint(prego);
             printstrln(" КОЛИЧЕСТВО\n");
             cmd=cmd>>16;
             printint(cmd);
             printstrln(" ПЕРИОД\n");
             INTERVAL_RESET=2.5*(XS1_TIMER_HZ/1000)*(duration&0xFFFF);
```

```
            INTERVAL_SET=(2.5*((XS1_TIMER_HZ/1000)*(cmd&0xFFFF))-
INTERVAL_RESET);
        }
        break;
        //пришла новая поправка на частоту
        case real_hz :> unsigned long long int real_hz__:
        real_hz_=real_hz__;
        time:>time_;
        INTERVAL=0;
        PHASE=1;
        break;
        //пришел стартбит можно начать запуск
        case sbit :> int sbit1:
        startbit=sbit1;
        break;
        //Если стартбит пришел, ждем 1PPS и начинаем
        case d :> int cmd_d:
        if(cmd_d==0)
        {
            if(startbit)
            {
            if(prego)
            {
                go=prego;
                prego=0;
                time:>time_;
                INTERVAL=0;
                //выставляем импульс композиции
                p_out2<:1;
            }
            startbit=0;
            }

        }
        break;
        //меняем состояние пина если пришло время
        case go => time when timerafter(time_+INTERVAL):>void:
        time:>time_; //записываем текущий момент времени
        //смотрим состояние пина и меняем его
        if(PHASE)
        {
            PHASE=0;
            INTERVAL=(real_hz_*INTERVAL_RESET)/(250000000);
            OUT=1;
        }
        else
        {
            PHASE=1;
            INTERVAL=(real_hz_*INTERVAL_SET)/(250000000);
            OUT=0;
            go--;
            //если закончились импульсы
            if(!go)
            {
                OUT=0;
                //убираем импульс композиции
                p_out2<:0;
            }
```

```
            }
            p_out<:OUT;
            break;
        }
    }
}

// ФАПЧ
void app_fapch(in port p_in, chanend d, chanend real_hz)
{
    timer time_fapch; //таймер для автоподстройки
    unsigned time_o; //начало интервала
    unsigned time_t; //конец интервала
    unsigned long long int realfapch;
    unsigned long long int realfapch_s;
    int pps;
    int low_level=1;
    time_fapch :> time_o;
    time_t=time_o+250000000;
    realfapch=250000000;
    realfapch_s=250000000;
    while(1)
    {
        //считываем состояние пинов
        p_in:>pps;
        d<:pps;
        if(((low_level)||(pps))==0)
        {
            time_fapch:>time_t; //берем значение которое насчитал таймер
            realfapch=time_t; //высчитываем сколько таймер насчитал
            realfapch-=time_o;
            //если явно ошибочные значения, то отправить последнюю
нормальную частоту, которая была корректной
            if( ( realfapch > 500000000 ) || ( realfapch < 125000000 ) )
            {
                realfapch=realfapch_s;
            }
            //отправить поправку
            real_hz<:realfapch;
            time_fapch:>time_o;
            //запоминаем корректное значение частоты
            realfapch_s=realfapch;
        }
        if(pps) //есть импульс 1PPS или нет
        {
            low_level=0;
        }
        else
        {
            low_level=1;
        }
    }
}

//старт-бит
void start_bit(in port p_start_bit, chanend sbit)
{
    int sb;
```

```
    while(1)
    {
        p_start_bit:>sb;
        if(sb)
        {
            sbit<:sb;
        }
    }
}

//////////////////////////////////////////////////////////////////////////
//////////////////////////////
// Program entry point
int main(void) {
       chan c_xtcp[1];
       chan c_gpio;
       chan sbit;
       chan d;
       chan real_hz;

     par
     {
          on stdcore[0]: reset_devices();
          // The main ethernet/tcp server
          on ETHERNET_DEFAULT_TILE:
             ethernet_xtcp_server(xtcp_ports,
                                  ipconfig,
                                  c_xtcp,
                                  1);


          // вебсервер
          on stdcore[0]: xhttpd(c_xtcp[0],c_gpio);
          on stdcore[1]:
app_gpio_out(c_gpio,d,p_out,p_out2,sbit,real_hz);
          on stdcore[1]: app_fapch(p_in,d,real_hz);
          on stdcore[1]: start_bit(p_start_bit,sbit);
     }
     return 0;
}
```

httpd.c

```
// Copyright (c) 2011, XMOS Ltd, All rights reserved
// This software is freely distributable under a derivative of the
// University of Illinois/NCSA Open Source License posted in
// LICENSE.txt and at <http://github.xcore.com/>

#include <string.h>
#include <stdlib.h>
#include <print.h>
#include "xtcp_client.h"
#include "httpd.h"
#include "xhttpd.h"


// Наши html страницы
```

```
char page[] = "<!DOCTYPE html>"
        "<html><head><meta http-equiv=\"Content-Type\"
content=\"text/html; charset=windows-1251\"><title>Задание
параметров</title>"
        "<style type=\"text/css\">"
        "div {"
        "width: 250px;"
        "height: 22px;"
        "padding: 5px;"
        "}"
        "</style>"
        "</head>"
        "<body style=\"align-content: center;\">"
        "<img src=\"http://www.legalstudies.ru/events/images/logos/kfu-
logo.gif\" alt=\"КФУ\" style=\"max-width: 250px;\">"
        "<form name=\"form\" action=\"\" method=\"post\" style=\"height:
250px; position: relative;\">"
        "<div style=\"background-color: #61c3ff; height: 100%; border-
radius: 10px;\">"
        "<div>Период между импульсами:</div>"
        "<div>"
        "<input type=\"text\" name=\"period\" size=\"20\"> мс"
        "</div>"
        "<div>Количество импульсов:</div>"
        "<div>"
        "<input type=\"text\" name=\"count\" size=\"20\">"
        "</div>"
        "<div>Длительность импульсов:</div>"
        "<div>"
        "<input type=\"text\" name=\"duration\" size=\"20\"> мс"
        "</div>"
        "<div style=\"position: relative; top: 20px;\">"
        "<input type=\"submit\" value=\"Отправить\" style=\"height: 25px;
width: 187px; border-radius: 8px;\">"
        "</div>"
        "</div>"
        "</form>"
        "</body>"
        "</html>";


char page2[] = "HTTP/1.0 200 OK\nServer: xc2/pre-1.0
(http://xmos.com)\nContent-type: text/html\n\n"
        "<!DOCTYPE html>"
        "<html><head>"
        "<meta http-equiv=\"content-type\" content=\"text/html;
charset=windows-1251\">"
        "<title>Отправлено</title>"
        "</head>"
        "<body>"
        "<img
src=\"https://upload.wikimedia.org/wikipedia/ru/thumb/b/bc/Russian_turkis
h_telescope_rtt150.jpg/250px-Russian_turkish_telescope_rtt150.jpg\"
alt=\"РТТ-150\" style=\"max-width: 250px;\">"
        "<div style=\"margin: relative; width: 230px; padding: 10px;
background-color: #61c3ff; height: 100%; border-radius: 10px; text-align:
center\">"
        "<div style=\"padding: 5px;\">Данные переданы</div>"
```

```
        "<form name=\"form2\" action=\"\" method=\"post\">"
        "<input name=\"knopka\" value=\"Остановить\" type=\"submit\"
style=\"height: 25px; width: 187px; border-radius: 8px;\">"
        "</form>"
        "<div>"
        "</body>"
        "</html>";


// Maximum number of concurrent connections
#define NUM_HTTPD_CONNECTIONS 10


// Structure to hold HTTP state
typedef struct httpd_state_t {
  int active;      //< Whether this state structure is being used
                   //  for a connection
  int conn_id;     //< The connection id
  char *dptr;      //< Pointer to the remaining data to send
  int dlen;        //< The length of remaining data to send
  char *prev_dptr; //< Pointer to the previously sent item of data
} httpd_state_t;

httpd_state_t connection_states[NUM_HTTPD_CONNECTIONS];
////

chanend c_gpio;

void init_web_c_gpio(chanend c_gpio_)
{
    c_gpio=c_gpio_;
}

// Initialize the HTTP state
void httpd_init(chanend tcp_svr)
{
  int i;
  // Listen on the http port
  xtcp_listen(tcp_svr, 80, XTCP_PROTOCOL_TCP);

  for ( i = 0; i < NUM_HTTPD_CONNECTIONS; i++ )
    {
      connection_states[i].active = 0;
      connection_states[i].dptr = NULL;
    }
}
////


// Parses a HTTP request for a GET
void parse_http_request(httpd_state_t *hs, char *data, int len)
{
    volatile char * xx;
    volatile unsigned long long int i;
    volatile int j;
    volatile unsigned long long int count;
    volatile unsigned long long int duration;
  // Return if we have data already
```

```
if (hs->dptr != NULL)
  {
    return;
  }

// Test if we received a HTTP GET request
if (strncmp(data, "GET ", 4) == 0)
  {
    // Assign the default page character array as the data to send
    hs->dptr = &page[0];
    hs->dlen = strlen(&page[0]);
  }
else
  {
    xx=strstr(data, "period");
    if(xx != NULL)
    {

        xx+=strlen("period=");
        i=0;
        j=0;
        while((*xx>='0')&&(*xx<='9')&&(j<5))
        {
            j++;
            i=i*10;
            i+=(*xx-'0');
            xx++;
        }
        if(i>0xFFFF)
            i=0xFFFF;
        i=i*0x10000;

        xx=strstr(data,"count");
        xx+=strlen("count=");
        count=0;
        j=0;
        while((*xx>='0')&&(*xx<='9')&&(j<5))
        {
            j++;
            count=count*10;
            count+=((*xx)-'0');
            xx++;
        }
        i|=(count&0xFFFF);
        xx=strstr(data,"duration");
        xx+=strlen("duration=");
        duration=0;
        j=0;

               while((*xx>='0')&&(*xx<='9')&&(j<5))
                  {
                      j++;
                      duration=duration*10;
                      duration+=((*xx)-'0');
                      xx++;
                  }
        i=i*0x10000;
        i|=(duration&0xFFFF);
```

```
            set_gpio_state(c_gpio,i);
            hs->dptr=&page2[0];
            hs->dlen=strlen(&page2[0]);
        }
        else
        {
        i=42;
        set_gpio_state(c_gpio,i);
        hs->dptr=&page[0];
        hs->dlen=strlen(&page[0]);
        }
        // We did not receive a get request, so do nothing
    }
}
//:



// Receive a HTTP request
void httpd_recv(chanend tcp_svr, xtcp_connection_t *conn)
{
  struct httpd_state_t *hs = (struct httpd_state_t *) conn->appstate;
  char data[XTCP_CLIENT_BUF_SIZE];
  int len;

  // Receive the data from the TCP stack
  len = xtcp_recv(tcp_svr, data);

  // If we already have data to send, return
  if ( hs == NULL || hs->dptr != NULL)
    {
      return;
    }

  // Otherwise we have data, so parse it
  parse_http_request(hs, &data[0], len);

  // If we are required to send data
  if (hs->dptr != NULL)
    {
      // Initate a send request with the TCP stack.
      // It will then reply with event XTCP_REQUEST_DATA
      // when it's ready to send
      xtcp_init_send(tcp_svr, conn);
    }
  ////
}


// Send some data back for a HTTP request
void httpd_send(chanend tcp_svr, xtcp_connection_t *conn)
{
  struct httpd_state_t *hs = (struct httpd_state_t *) conn->appstate;

  // Check if we need to resend previous data
  if (conn->event == XTCP_RESEND_DATA) {
```

```
    xtcp_send(tcp_svr, hs->prev_dptr, (hs->dptr - hs->prev_dptr));
    return;
  }

  // Check if we have no data to send
  if (hs->dlen == 0 || hs->dptr == NULL)
    {
      // Terminates the send process
      xtcp_complete_send(tcp_svr);
      // Close the connection
      xtcp_close(tcp_svr, conn);
    }
  // We need to send some new data
  else {
    int len = hs->dlen;

    if (len > conn->mss)
      len = conn->mss;

    xtcp_send(tcp_svr, hs->dptr, len);

    hs->prev_dptr = hs->dptr;
    hs->dptr += len;
    hs->dlen -= len;
  }
  ////
}


// Setup a new connection
void httpd_init_state(chanend tcp_svr, xtcp_connection_t *conn)
{
  int i;

  // Try and find an empty connection slot
  for (i=0;i<NUM_HTTPD_CONNECTIONS;i++)
    {
      if (!connection_states[i].active)
        break;
    }

  // If no free connection slots were found, abort the connection
  if ( i == NUM_HTTPD_CONNECTIONS )
    {
      xtcp_abort(tcp_svr, conn);
    }
  // Otherwise, assign the connection to a slot        //
  else
    {
      connection_states[i].active = 1;
      connection_states[i].conn_id = conn->id;
      connection_states[i].dptr = NULL;
      xtcp_set_connection_appstate(
           tcp_svr,
           conn,
           (xtcp_appstate_t) &connection_states[i]);
    }
}
```

```
// Free a connection slot, for a finished connection
void httpd_free_state(xtcp_connection_t *conn)
{
  int i;

  for ( i = 0; i < NUM_HTTPD_CONNECTIONS; i++ )
    {
      if (connection_states[i].conn_id == conn->id)
        {
          connection_states[i].active = 0;
        }
    }
}
////


// HTTP event handler
void httpd_handle_event(chanend tcp_svr, xtcp_connection_t *conn)
{
  // We have received an event from the TCP stack, so respond
  // appropriately

  // Ignore events that are not directly relevant to http
  switch (conn->event)
    {
    case XTCP_IFUP: {
      xtcp_ipconfig_t ipconfig;
      xtcp_get_ipconfig(tcp_svr, &ipconfig);

#if IPV6
      unsigned short a;
      unsigned int i;
      int f;
      xtcp_ipaddr_t *addr = &ipconfig.ipaddr;
      printstr("IPV6 Address = [");
      for(i = 0, f = 0; i < sizeof(xtcp_ipaddr_t); i += 2) {
        a = (addr->u8[i] << 8) + addr->u8[i + 1];
        if(a == 0 && f >= 0) {
          if(f++ == 0) {
            printstr("::");
           }
        } else {
            if(f > 0) {
              f = -1;
            } else if(i > 0) {
                printstr(":");
            }
          printhex(a);
        }
      }
      printstrln("]");
#else
      printstr("IP Address: ");
      printint(ipconfig.ipaddr[0]);printstr(".");
      printint(ipconfig.ipaddr[1]);printstr(".");
      printint(ipconfig.ipaddr[2]);printstr(".");
```

```
      printint(ipconfig.ipaddr[3]);printstr("\n");
#endif
      }
      return;
    case XTCP_IFDOWN:
    case XTCP_ALREADY_HANDLED:
      return;
    default:
      break;
    }

  // Check if the connection is a http connection
  if (conn->local_port == 80) {
    switch (conn->event)
      {
      case XTCP_NEW_CONNECTION:
        httpd_init_state(tcp_svr, conn);
        break;
      case XTCP_RECV_DATA:
        httpd_recv(tcp_svr, conn);
        break;
      case XTCP_SENT_DATA:
      case XTCP_REQUEST_DATA:
      case XTCP_RESEND_DATA:
          httpd_send(tcp_svr, conn);
          break;
      case XTCP_TIMED_OUT:
      case XTCP_ABORTED:
      case XTCP_CLOSED:
          httpd_free_state(conn);
          break;
      default:
        // Ignore anything else
        break;
      }
    conn->event = XTCP_ALREADY_HANDLED;
  }
  ////
  return;
}
////
```

httpd.h

```
// Copyright (c) 2011, XMOS Ltd, All rights reserved
// This software is freely distributable under a derivative of the
// University of Illinois/NCSA Open Source License posted in
// LICENSE.txt and at <http://github.xcore.com/>

#ifndef _httpd_h_
#define _httpd_h_

#include "xtcp_client.h"

void httpd_init(chanend tcp_svr);
void httpd_handle_event(chanend tcp_svr,
REFERENCE_PARAM(xtcp_connection_t, conn));
```

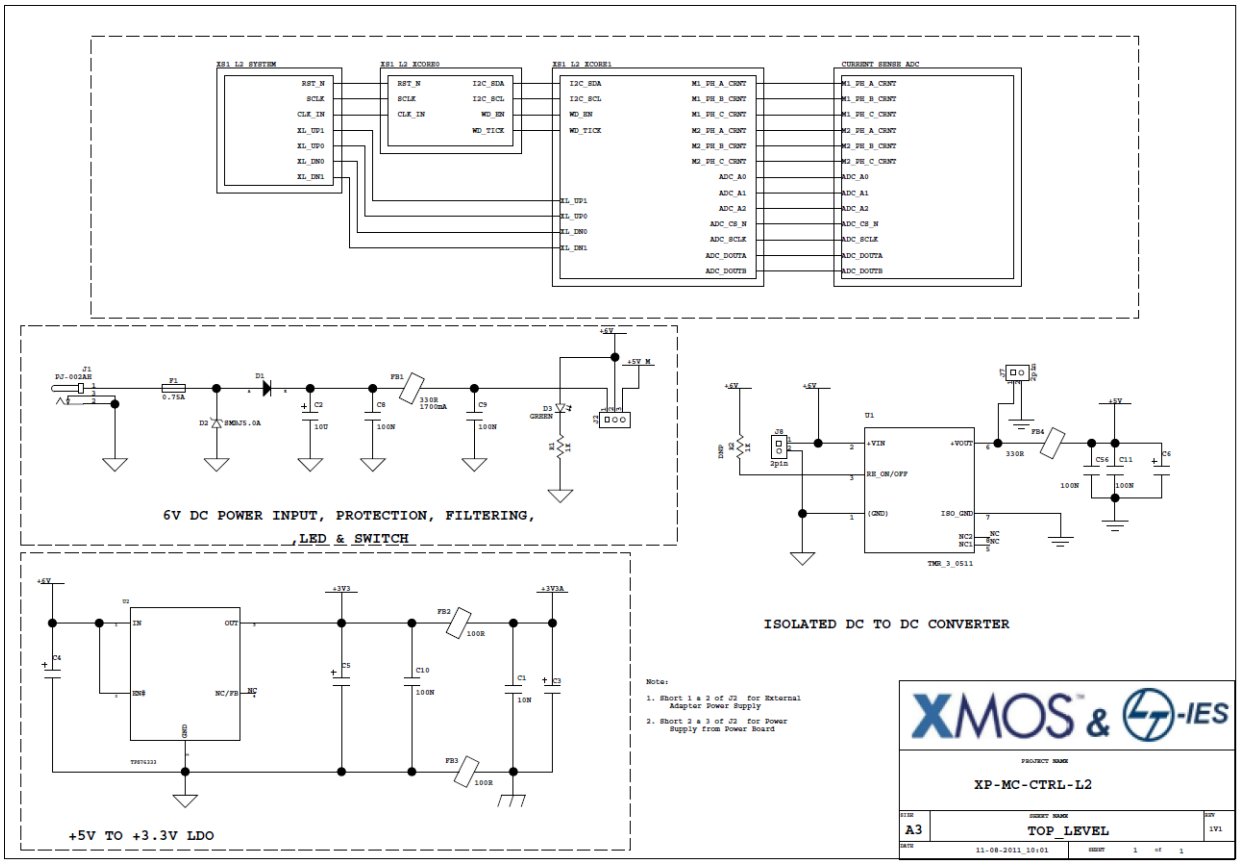
```
void init_web_c_gpio(chanend c_gpio);

#endif // _httpd_h_
```

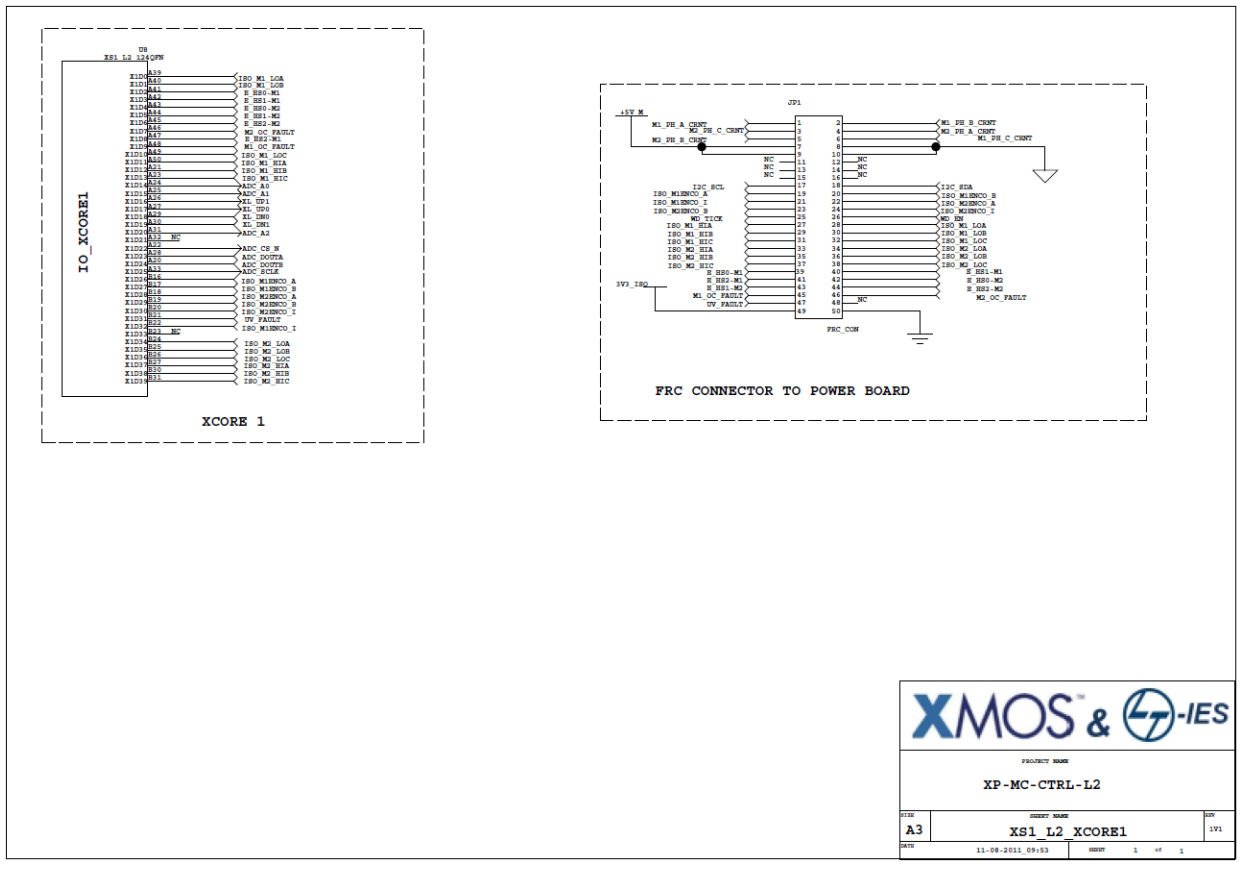
xhttpd.h

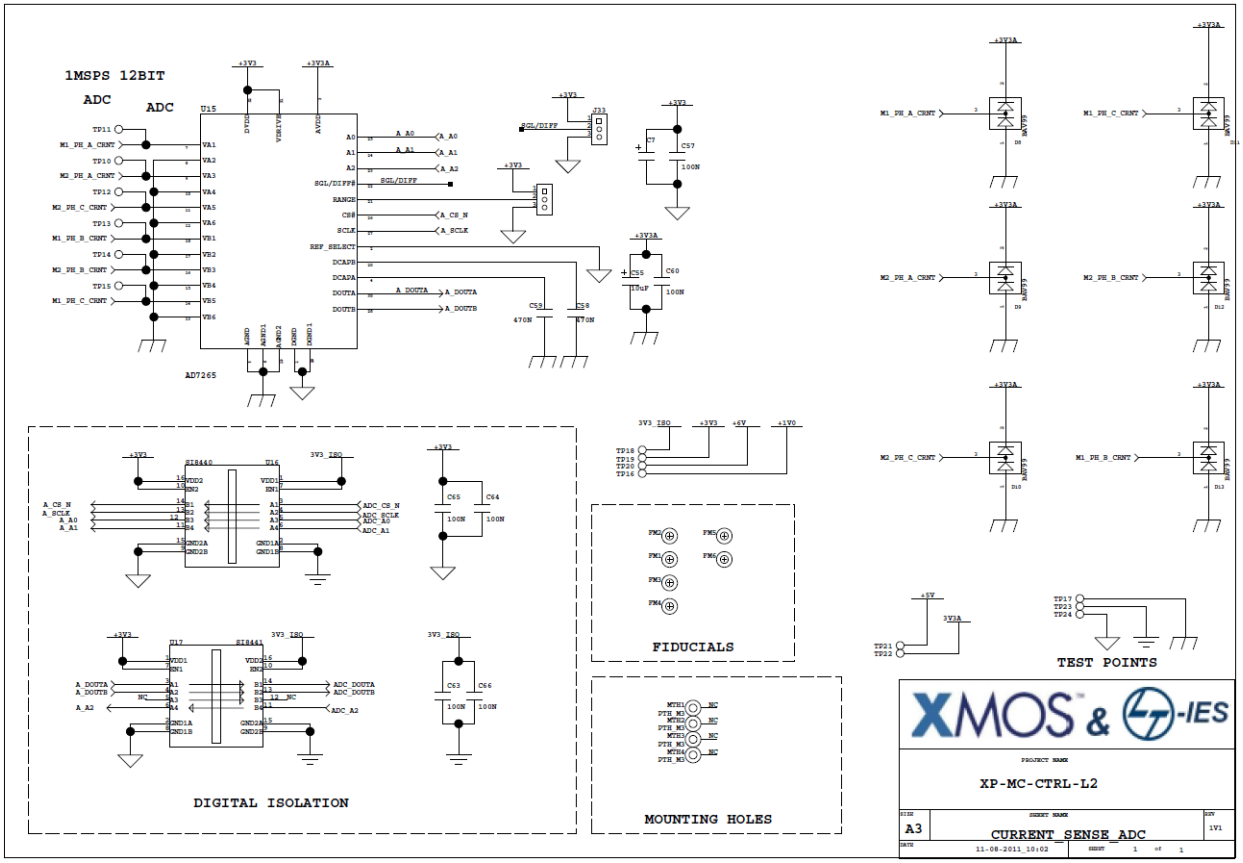
```
// Copyright (c) 2011, XMOS Ltd, All rights reserved
// This software is freely distributable under a derivative of the
// University of Illinois/NCSA Open Source License posted in
// LICENSE.txt and at <http://github.xcore.com/>

#ifndef _xhttpd_h_
#define _xhttpd_h_

void xhttpd(chanend tcp_svr, chanend c_gpio);
void set_gpio_state(chanend c_gpio, unsigned long long int s);
#endif // _xhttpd_h_
```

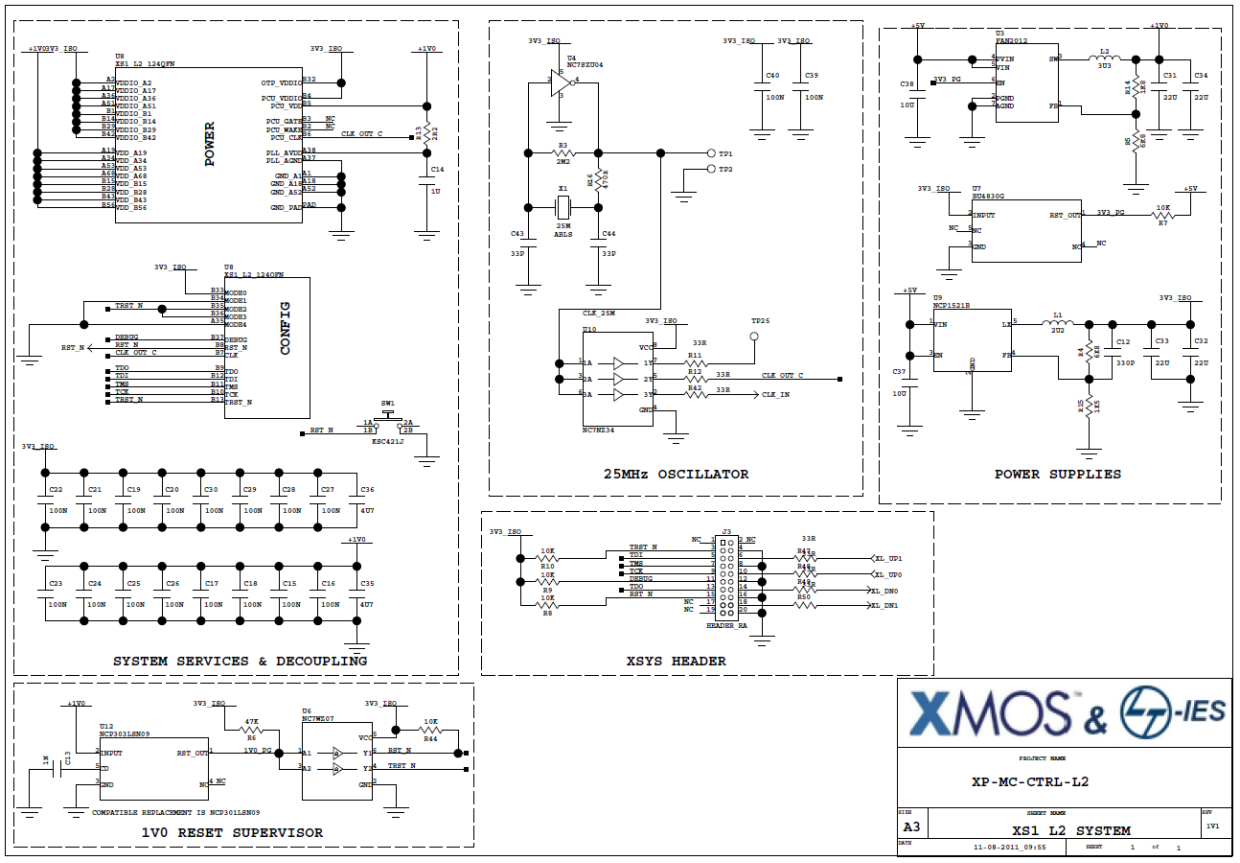
xhttpd.xc

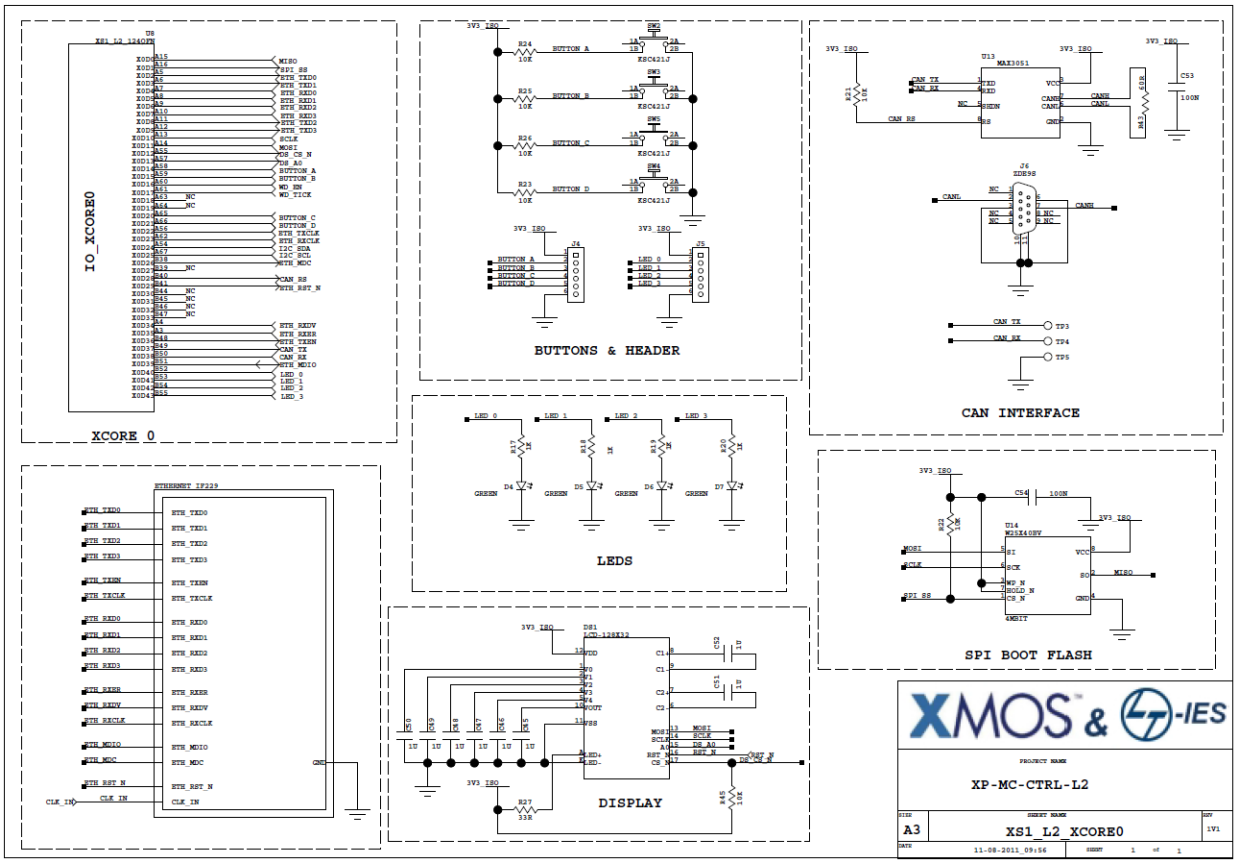
```
// Copyright (c) 2011, XMOS Ltd, All rights reserved
// This software is freely distributable under a derivative of the
// University of Illinois/NCSA Open Source License posted in
// LICENSE.txt and at <http://github.xcore.com/>

#include <xs1.h>
#include <print.h>
#include "httpd.h"
#include "xtcp_client.h"

// The main webserver thread
void xhttpd(chanend tcp_svr, chanend c_gpio)
{
  xtcp_connection_t conn;
  printstrln("**WELCOME TO THE SIMPLE WEBSERVER DEMO**");
  // Initiate the HTTP state
  httpd_init(tcp_svr);

  init_web_c_gpio(c_gpio);
  // Loop forever processing TCP events
  while(1)
    {
      select
        {
        case xtcp_event(tcp_svr, conn):
          httpd_handle_event(tcp_svr, conn);
          break;
        }

    }
}

void set_gpio_state(chanend c_gpio, unsigned long long int s)
{
    c_gpio<:s;
}
```

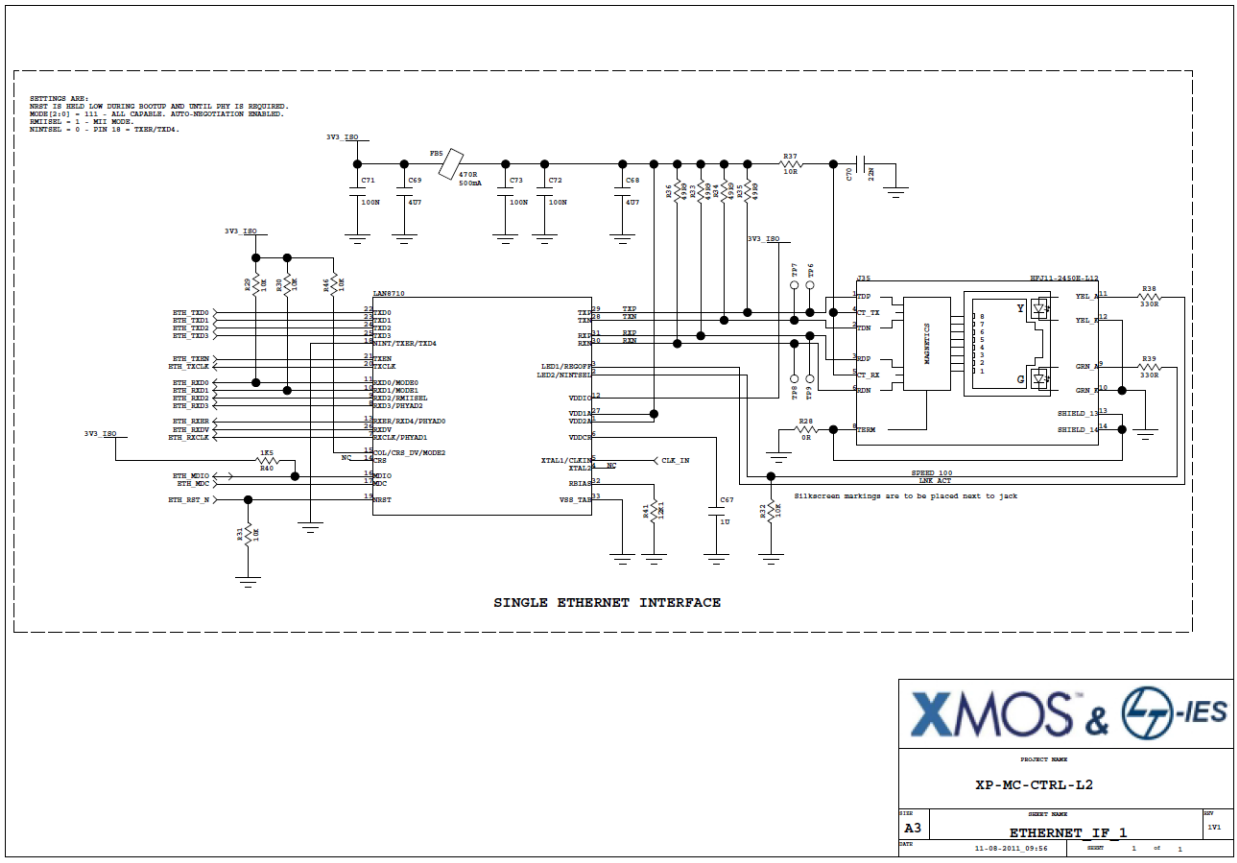

XS1_L2_SYSTEM
XS1_L2_XCORE0
XS1_L2_XCORE1
CURRENT_SENSE_ADC
6V DC POWER INPUT, PROTECTION, FILTERING, LED & SWITCH
ISOLATED DC TO DC CONVERTER
+5V TO +3.3V LDO
Note:
1. Short 1 a 2 of J2 for External Adapter Power Supply
2. Short 2 a 3 of J2 for Power Supply from Power Board
XMOS & XA-IES
PROJECT NAME
XP-MC-CTRL-L2
SIZE
A3
SHEET NAME
TOP_LEVEL
REV
1V1
DATE
11-08-2011_10:01
SHEET 1 of 1


XCORE 1
FRC CONNECTOR TO POWER BOARD
XMOS & XA-IES
PROJECT NAME
XP-MC-CTRL-L2
SIZE
A3
SHEET NAME
XS1_L2_XCORE1
REV
1V1
DATE
11-08-2011_09:53
SHEET 1 of 1


1MSPS 12BIT
ADC
ADC
AD7265
FIDUCIALS
TEST POINTS
DIGITAL ISOLATION
MOUNTING HOLES
XMOS & XYLINX
XP-MC-CTRL-L2
A3
CURRENT_SENSE_ADC
11-08-2011_10:02


POWER
CONFIG
SYSTEM SERVICES & DECOUPLING
25MHz OSCILLATOR
POWER SUPPLIES
XSYS HEADER
1V0 RESET SUPERVISOR
XMOS & XYLINX
XP-MC-CTRL-L2
A3
XS1 L2 SYSTEM
11-08-2011_09:55


IO_XCORE0
XCORE_0
BUTTONS & HEADER
CAN INTERFACE
LEDS
SPI BOOT FLASH
DISPLAY
ETHERNET_IF229
XMOS & XTIMES
XP-MC-CTRL-L2
XS1_L2_XCORE0
A3
11-08-2011_09:56


SETTINGS ARE:
NRST IS HELD LOW DURING BOOTUP AND UNTIL PHY IS REQUIRED.
MODE[2:0] = 111 - ALL CAPABLE. AUTO-NEGOTIATION ENABLED.
EMIISEL = 1 - MII MODE.
NINTSEL = 0 - PIN 18 = TXER/TXD4.
LAN8710
HFJ11-2450E-L12
Silkscreen markings are to be placed next to jack
SINGLE ETHERNET INTERFACE
XMOS & XTIMES
XP-MC-CTRL-L2
ETHERNET_IF_1
A3
11-08-2011_09:56