МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего профессионального  
образования

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»**

**Факультет автоматики и электроники**

**Кафедра электроники**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К ДИПЛОМНОМУ ПРОЕКТУ НА ТЕМУ:**

Разработка макета для изучения микропроцессорных СБИС с использованием платформы

NI ELVIS II

Студент-дипломник: / Шлыгин А.С. /

Руководитель проекта: / Некрасов П.В. /

Рецензент проекта: / /

Заведующий кафедрой №3: / Барбашов В.М. /

Москва 2017 г.

Оглавление

[Введение. 4](#_Toc485206979)

[Глава 1. Лабораторный макет NI ELVIS II+. 5](#_Toc485206980)

[1.1 Программные устройства 5](#_Toc485206981)

[1.2 Аппаратные устройства 7](#_Toc485206982)

[Глава 2. Существующие макеты на базе ELVIS 9](#_Toc485206983)

[2.1 Знакомство с аппаратной средой. 9](#_Toc485206984)

[2.1.1 Аппаратные компоненты NI ELVIS II. 10](#_Toc485206985)

[2.1.2 Настольная рабочая станция NI ELVIS II. 10](#_Toc485206986)

[2.1.3 Задняя панель NI ELVIS II 12](#_Toc485206987)

[2.1.4 Нижняя панель NI ELVIS II 13](#_Toc485206988)

[2.1.5 Схема защиты NI ELVIS II 13](#_Toc485206989)

[2.1.6 Макетная плата NI ELVIS II 13](#_Toc485206990)

[2.1.7 Описание сигналов 14](#_Toc485206991)

[2.2 Виртуальные измерительные приборы 16](#_Toc485206992)

[2.2.1 Цифровой мультиметр 18](#_Toc485206993)

[2.2.2 Генератор стандартных сигналов (FGEN) 20](#_Toc485206994)

[2.2.3 Осциллограф (SCOPE) 21](#_Toc485206995)

[2.2.4 Анализатор АЧХ. 22](#_Toc485206996)

[2.2.5 Анализатор спектра (DSA) 24](#_Toc485206997)

[2.2.6 Цифровой ввод/вывод (DigIn/DigOut) 25](#_Toc485206998)

[Глава 3.Макет прошлый 27](#_Toc485206999)

[3.1 Выбор микроконтроллера 27](#_Toc485207000)

[3.2 Описание микроконтроллера STM32F103ZET6. 27](#_Toc485207001)

[3.2.1 Ядро ARM Cortex-M3 с интегрированной Flash и SRAM памятью 29](#_Toc485207002)

[3.2.2 Встроенная Flash-память 30](#_Toc485207003)

[3.2.3 Блок расчета CRC 30](#_Toc485207004)

[3.2.4 Встроенная SRAM-память 30](#_Toc485207005)

[3.2.5 FSMC 30](#_Toc485207006)

[3.2.6 Параллельный интерфейс работы с LCD 31](#_Toc485207007)

[3.2.7 Контроллер приоритетных векторных прерываний (NVIC) 31](#_Toc485207008)

[3.2.8 Контроллер внешних прерываний/событий (EXTI) 31](#_Toc485207009)

[3.2.9 Тактовый сигнал и запуск 31](#_Toc485207010)

[3.2.10 Режимы загрузки 32](#_Toc485207011)

[3.2.11 Схемы питания 32](#_Toc485207012)

[3.3 Выбор ПЛИС 35](#_Toc485207013)

[3.4 Описание ПЛИС 10M08SAU169 36](#_Toc485207014)

[3.4.1 Логические элементы и матрицы логических элементов 36](#_Toc485207015)

[Глава 4. Макет новый 38](#_Toc485207016)

[4.1 Используемые технологии при разработке 38](#_Toc485207017)

[4.2 Плата управления макетом на основе МК 39](#_Toc485207018)

[4.2.1 Этапы разработки платы управления на основе МК 40](#_Toc485207019)

[4.3 Апробация платы МК 41](#_Toc485207020)

[4.3.1 Тактирование МК 41](#_Toc485207021)

[4.3.2 АЦП 41](#_Toc485207022)

[4.3.3 Порты ввода-вывода 42](#_Toc485207023)

[4.3.4 Интерфейсы UART и SPI 42](#_Toc485207024)

[4.4 Плата управления макетом на основе ПЛИС 43](#_Toc485207025)

[4.4.1 Этапы разработки платы управления на основе ПЛИС 43](#_Toc485207026)

[Заключение. 45](#_Toc485207027)

[Список литературы 46](#_Toc485207028)

[Приложение А 47](#_Toc485207029)

[Приложение Б 52](#_Toc485207030)

[Приложение В 71](#_Toc485207031)

# Введение.

Для эффективного обучения студентов, кафедре электроники необходимо обеспечивать не только теоретическое обучение студентов, но и приобретение ими необходимых практических навыков.

На данный момент технологии изготовления позволяют выпускать микроконтроллеры и ПЛИС любой ценовой категории, что расширяет их сферу применения. Они находят свое применение в широком спектре устройств различной степени сложности – от ёлочных гирлянд до спутников и суперкомпьютеров. С другой стороны растут возможности микропроцессорных СБИС, что предъявляет к разработчикам более жесткие требования как в знаниях программирования, так и во внутренних процессах СБИС.

Широкая номенклатура микроконтроллеров и оценочных комплектов на их базе создают определенные трудности в изучении микропроцессорных систем. Для более удобного и быстрого освоения микроконтроллеров и ПЛИС необходимо как-либо стандартизовать и «привести к общему знаменателю» периферийные устройства и входы-выходы изучаемых программируемых СБИС. Это позволит изучать большое количество микроконтроллеров и ПЛИС используя одни и те же периферийные устройства. Такой подход предоставит возможность быстрого переключения между изучаемыми СБИС, а также возможность наглядного сравнения результатов, в том числе между различными микроконтроллерами или микроконтроллеров и ПЛИС, имеющие различный подход к программированию.

Еще одним немаловажным фактором в обучении является получение результатов и их осознание. При программировании порой сложно понять почему написанная программа работает неправильно или не работает вообще. Такие ситуации возникают из-за невозможности контролировать конечный результат не зависимо от качества его исполнения устройством. Чтобы избежать подобных ситуаций и предоставить возможность контролировать результат исполнения программы микроконтроллером или ПЛИС необходимо сразу несколько инструментов, такие как источник питания, осциллограф, генератор электрических сигналов, мультиметр. Такие приборы занимают много места на лабораторном столе и требуют минимальной, но все же подготовки для их применения.

Решением множества из перечисленных проблем можно назвать платформу NI ELVIS, имеющую все необходимые инструменты, но не имеющей периферийных устройств. Целью данной работы является создание макетной платы для изучения микроконтроллеров и ПЛИС.

# Лабораторный макет NI ELVIS II+.

Лабораторный макет NI ELVIS представляет собой программно-аппаратную платформу со множеством встроенных измерительных и контрольных устройств, управление которыми осуществляется с компьютера. Еще одной возможностью данной платформы является отсутствие встроенной периферии, которой можно управлять или наблюдать за её работой – платы с периферией необходимо покупать или создавать самостоятельно. Существует несколько готовых макетный плат с различными периферийными устройствами для обучения различным направлениям схемотехники и исследований физических процессов.

# Существующие макетные платы на базе платформы ELVIS

## Плата системы управления (Control System Design and Analysis)

Плата системы управления

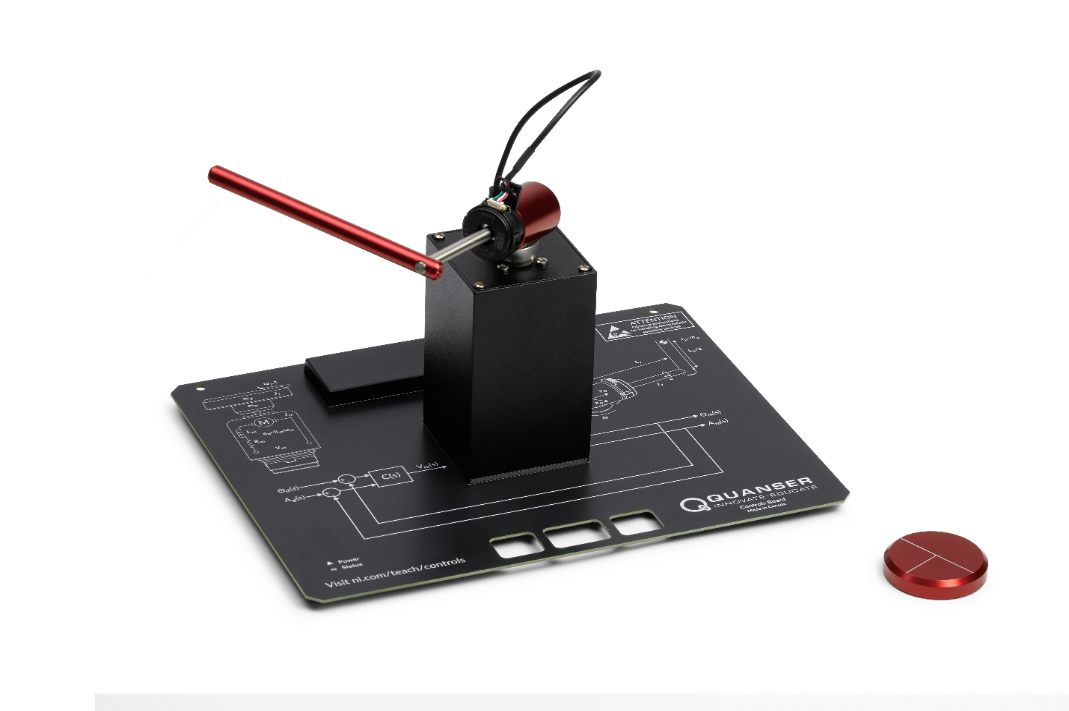
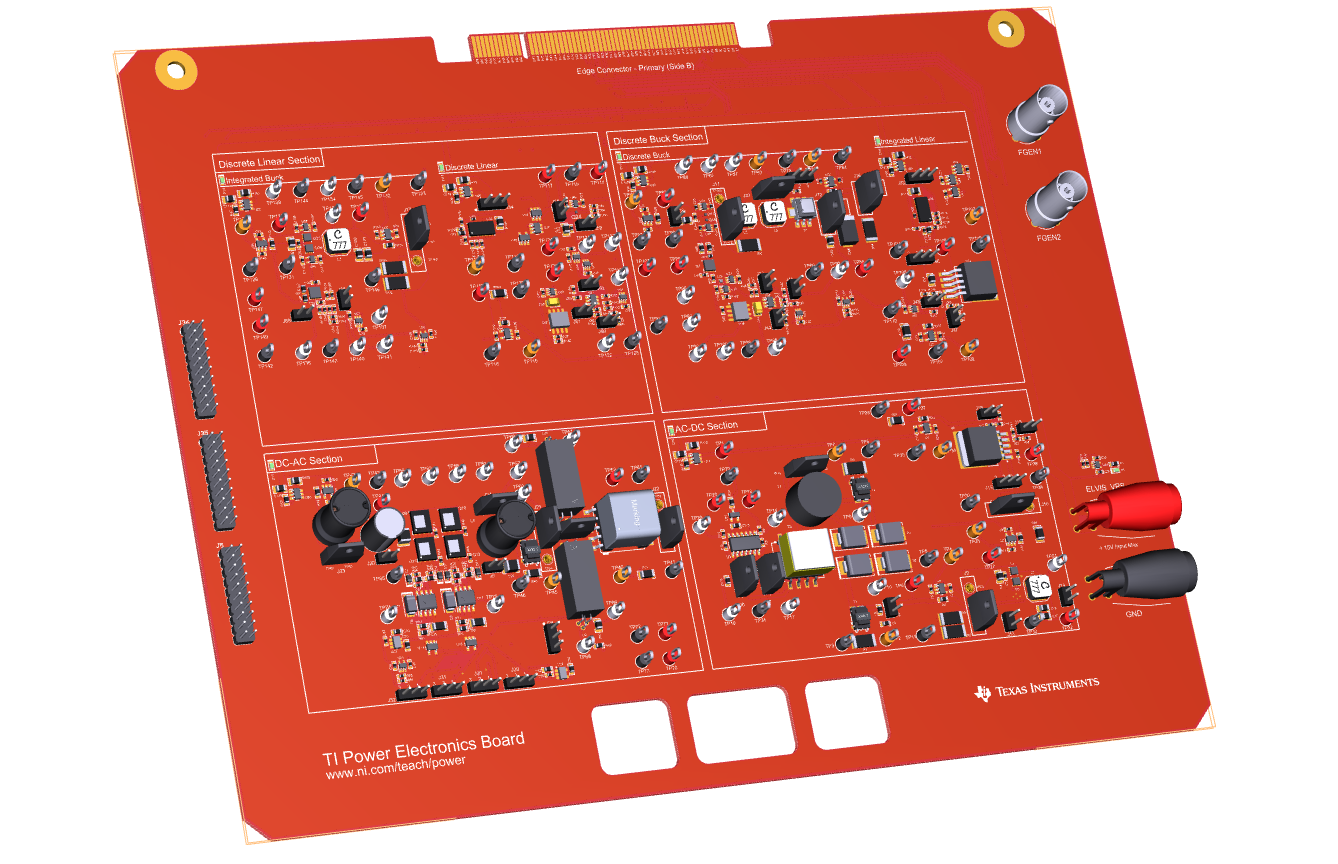


Рисунок 2.1. Плата системы управления

Плата представляет собой платформу для исследования практически всех аспектов современной теории управления от системного моделирования и ПИД-регулирования до стабильности и цифрового управления. Система состоит из детерминированного двигателя постоянного тока с датчиком высокого разрешения, а также крепления маятника для контроля баланса. Плата включает в себя следующие элементы:

Данная плата используется для изучения механизмов управления механическими приводами и стабилизации (балансировки) грузов, в том числе при изменении массы.

## Плата силовой электроники (Power Electronics)



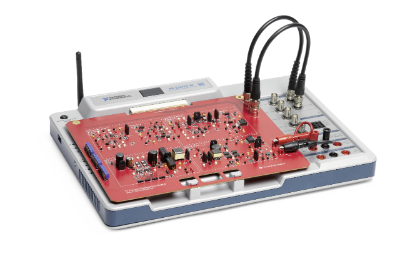


Рисунок 2.2. Плата силовой электроники

Плата силовой электроники предоставляет собой учебную практическую платформу для изучения собственно силовой электроники и управления питанием. Используя функциональные блоки, можно создавать конвертеры, регуляторы и преобразователи DC-AC и AС-DC. Данная плата позволяет получить представление о каждом компоненте в силовой электронике и о том, как он влияет на другие компоненты для создания единой системы при использовании стандартных схем.

## Плата мехатронного системного анализа (Mechatronic Systems Analysis)



Рисунок 2.3

Плата, изображенная на рисунке 2.3 , представляет собой универсальную систему, которая позволяет студентам исследовать и внедрять различные подкомпоненты мехатронной PWM системы, управления манипулятором и кинематикой, обработки изображений. Система состоит из 5-контактного разъема с прямым приводом, камеры с манипулятором, настраиваемых накладных изображений пространства и модифицируемых контроллеров LabVIEW.

## Плата анализа энергетических систем (Energy Systems Analysis)

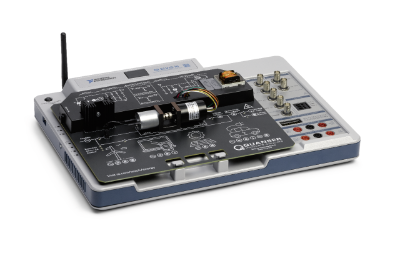


Рисунок 2.4

Плата Quanser Energy Systems, изображенная на рисунке 2.4, представляет собой систему, которая позволяет студентам исследовать и исследовать различные подкомпоненты электромеханической силовой системы. Диапазон температуры от выработки, выпрямления и инверсии переменного тока для увеличения мощности и повышения мощности постоянного тока. Система состоит из источника питания постоянного тока и трехфазного генератора переменного тока, трехфазного выпрямителя, источника питания с переключаемым режимом, инвертора и трансформатора, приемника постоянного тока и модифицируемых контроллеров LabVIEW. Плата может быть легко адаптирована к широкому спектру приложений для энергосистем, таких как производство энергии ветра и солнечной энергии, а также источники питания потребителей и электромобили.

## Плата основных мехатронных датчиков (Fundamentals of Mechatronic Sensors)

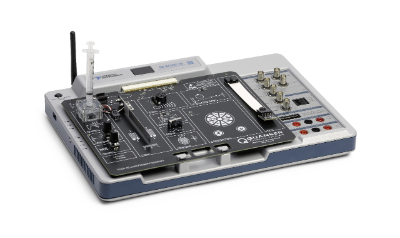


Рисунок 2.5

Доска Quanser Mechatronic Sensors Board, изображенная на рисунке 1, представляет собой прикладную плату, предназначенную для обучения и демонстрации основ общих датчиков, используемых в мехатронных приложениях. Он предоставляет студентам практический опыт измерения, калибровки и анализа следующих физических свойств / явлений: контакт, угловое смещение, расстояние и близость, давление, деформацию, температуру и инерционные измерения. Система работает с использованием ПК с LabVIEW 2018 и NI ELVIS III.

## Плата основных мехатронных приводов (Fundamentals of Mechatronic Actuators)



Рисунок 2.5

Доска Quanser Mechatronic Actuators, изображенная на рисунке 1, является идеальным инструментом для внедрения различных обычных исполнительных механизмов и демонстрации их преимуществ, взаимодействия и эксплуатации, а также соображений и ограничений конструкции. Темы включают двигатели постоянного тока, усилители мощности и управляемые положением приводы. Плата состоит из мотального двигателя постоянного тока с выбираемыми линейными и усиленными усилителями мощности PWM, бесщеточным двигателем постоянного тока, шаговым двигателем, встроенным сервомеханизмом и модифицируемыми контроллерами LabVIEW. Плата дает студентам опыт работы с рабочими приводами, устраняя сложности, связанные с подключением и настройкой схемы питания и управления. Сравнивают и контрастируют технологии приводов, позволяя пользователям комфортно принимать проектные решения о приводах и усилителях в реальных приложениях.

## Устройство и схемы

На кафедре был создан лабораторный макет для изучения микроконтроллерных систем с помощью платформы ELVIS. В его состав входят:

- микроконтроллер STM32F103;

- цифровая двенадцатикнопоная клавиатура;

- LCD-дисплей;

- динамик;

- четырехразрядный семисегментный индикатор;

- три программируемые с помощью микроконтроллера кнопки;

- вспомогательные устройства, обеспечивающие питание и и защиту платы.

Также предусмотрена возможность подключения к плате дополнительных переферийных устройств через несколько портов ввода-вывода.

Конструкция платы оказалась весьма неудачной: разбиение портов на цифровые, аналоговые и дополнительные сильно усложняет подключение дополнительных переферийных устройств и может приводить к ошибкам в подключении. Также, во время тестирования платы были обнаружены значительные ошибки в принципиальной схеме и расположении элементов перефирии. Так, например, на сдвиговые регистры, необходимые для работы LCD-дисплея и семисегментного индикатора подавалось напряжение питание, равное 3,3В вместо 5В.

## Тестовые программы и прошивка микроконтроллера.

Для тестирования платы в рабочем режиме были написаны несколько программ, каждая из которых отвечает за определенный модуль периферии: для LCD-дисплея, цифровой клавиатуры, семисегментного индикатора, динамика.

# Разработка плат управления макетом

## Используемые технологии при разработке

Для разработки плат управления использовался программный комплекс KiCad. KiCad — свободный программный комплекс класса EDA с открытым исходным кодом. Разрабатывается специалистами из Laboratoire des Images et des Signaux и ЦЕРН.

Состоит из следующих компонентов:

* kicad — менеджер проектов;
* eeschema — редактор электрических схем;
  + встроенный редактор символов схем (библиотечных компонентов);
* pcbnew — редактор печатных плат;
  + встроенный редактор образов посадочных мест (библиотечных компонентов);
  + 3D Viewer — 3D-просмотрщик печатных плат на базе OpenGL (часть pcbnew);
* gerbview — просмотровщик файлов Gerber (фотошаблонов);
* cvpcb — программа для выбора посадочных мест, соответствующих компонентам на схеме;
* wyoeditor — текстовый редактор для просмотра отчётов.

Комплекс обладает полным набором возможностей, необходимых для эффективного проектирования электронных устройств, включая:

* Построение принципиальных схем радиоэлектронных устройств
* Проверка правил построения схем, таких как конфликты подключения контактов и отсутствующие соединения
* Выгрузка списков электрических цепей в форматах PSpice, Cadstar, PcbNew, XML
* Создание списков компонентов (BOM)
* Обширная библиотека компонентов, с возможностью простого создания новых элементов, и редактирования уже имеющихся
* Интерактивный роутер, с поддержкой Push and shove, подстройкой абсолютной и относительной длины дорожек для дифференциальных сигналов
* Автоматическое построение 3D-моделей разрабатываемых печатных плат
* Поддержка автоматизации задач при помощи python-скриптов
* Встроенный SPICE-симулятор разрабатываемых схем

## Плата управления макетом на основе МК

Плата управления макетом на основе МК разрабатывалась с учетом следующих ограничений:

* Размер платы должен составлять 96\*108 мм
* На плате должен находиться универсальный разъем для подключения к лабораторному макету
* Должно обеспечиваться разделение цифровой и аналоговой части платы для изоляции цифровых шумов
* Плата должна соответствовать 4 классу точности

В процессе разработки так же учитывались следующие моменты:

* Должна обеспечиваться совместимость с микроконтроллерами семейств STM32F1xx, STM32F20x, STM32F4xx посредством установки перемычек на печатной плате
* Должна обеспечиваться возможность выбора способа загрузки микроконтроллера при помощи установки перемычек на линиях BOOT0 и BOOT1.
* Должна обеспечиваться возможность выбора опорного напряжения АЦП при помощи перемычки. Возможные значения должны включать в себя напряжение питания платы и отдельный источник опорного напряжения.
* Должен присутствовать разъем внутрисхемной отладки JTAG, соответствующий стандарту ULINK2 10 Pin.
* Должна присутствовать отдельная кнопка сброса микроконтроллера, помимо основной линии сброса на универсальном разъеме, для обеспечения возможности использования платы вне макета.

### Этапы разработки платы управления на основе МК

Для разработки платы управления на основе МК были изучены документы, представленные производителем устройства.

Разработан модуль KiCad, соответствующий универсальному разъему лабораторного макета.

Составлена принципиальная схема устройства, учитывающая поставленные критерии. Так же учтены рекомендации производителя по обеспечению цифрового питания устройства, аналогового питания, источнику опорного напряжения, подключению тактовых осцилляторов, сигнала сброса.

Проведена трассировка платы управления, учитывающая поставленные критерии. Большая часть нижнего слоя платы залита земляным полигоном, сшитым переходными отверстиями с земляными полигонами верхнего слоя, для обеспечения пути минимального сопротивления для обратных токов сигналов. Минимизированы пересечения сигнальных дорожек.

Подготовлены слои шелкографии, описывающие имеющиеся разъемы на плате, а также назначение перемычек припоя для выбора типа установленной микросхемы. Подробно описано назначение каждого контакта универсального разъема, и соответствующий ему контакт микроконтроллера.

Таблицы сопоставления контактов МК контактам универсального разъема и схемы платы находятся в приложении А.

## Апробация платы МК

Для апробации платы с МК был собран макет, в который вошли:

* Плата управления на основе МК
* Потенциометр
* Адаптер USB-UART
* Светодиод RGB
* Модуль трансивера ISM-диапазона с интерфейсом SPI

Так же использовался осциллограф и цифровой мультиметр.

Было написано тестовое приложение, покрывающее некоторые возможности управляющей платы.

Исходные коды приложения расположены в приложении Б.

### Тактирование МК

Для тактирования МК использовался кварцевый резонатор на 8 МГц. Конечная частота достигалась при помощи ФАПЧ. На вывод МК подавался сигнал ШИМ заданной при помощи делителя частоты и выбранной скважности.

Корректность выхода МК на режим работы от внешнего резонатора проверялся при помощи отладчика по отсутствию флага ошибки в соответствующем регистре, и при помощи осциллографа, по достижению сигнала ШИМ частоты, соответствующей работе на заданной частоте при указанном делителе.

### АЦП

Работа АЦП проверялась при помощи подачи на вход сигнала с выхода делителя потенциометра. Была проверена работа как с опорным напряжением, равным напряжению питания, так и со взятым с отдельного источника опорного напряжения. Результаты преобразования отражались в скважность ШИМ-сигнала.

Корректность проверялась при помощи отладчика, путем сверки данных с АЦП и показаний мультиметра, а также путем визуальной инспекции изменения скважности ШИМ.

### Порты ввода-вывода

Порты ввода-вывода проверялись при помощи осциллографа путем визуальной инспекции изменения уровня сигнала на соответствующих портах при подаче на указанные порты последовательность двоичных значений.

### Интерфейсы UART и SPI

Работоспособность интерфейса UART проверялась при помощи адаптера USB-UART, путем подачи команд управления частотой и скважностью ШИМ на плату МК. Корректность контролировалась как при помощи отладчика, так и при помощи визуальной инспекции изменения параметров ШИМ на экране осциллографа.

Интерфейс SPI проверялся при помощи отладчика путем чтения и записи данных в регистры трансивера ISM-диапазона.

## Плата управления макетом на основе ПЛИС

Плата управления макетом на основе ПЛИС разрабатывалась с учетом следующих ограничений:

* Размер платы должен составлять 96\*108 мм
* На плате должен находиться универсальный разъем для подключения к лабораторному макету
* Должно обеспечиваться разделение цифровой и аналоговой части платы для изоляции цифровых шумов
* Плата должна соответствовать 5 классу точности

В процессе разработки так же учитывались следующие моменты:

* Должна обеспечиваться возможность выбора способа конфигурации ПЛИС при помощи установки перемычки на линии Config\_Sel.
* Должна обеспечиваться возможность выбора опорного напряжения АЦП при помощи перемычки. Возможные значения должны включать в себя напряжение питания платы и отдельный источник опорного напряжения.
* Должен присутствовать разъем внутрисхемной отладки JTAG, соответствующий стандарту ULINK2 10 Pin.
* Должна присутствовать отдельные кнопки сброса ПЛИС, помимо основной линии сброса на универсальном разъеме, для обеспечения возможности использования платы вне макета.

### Этапы разработки платы управления на основе ПЛИС

Для разработки платы управления на основе ПЛИС были изучены документы, представленные производителем устройства.

Разработан модуль KiCad, соответствующий микросхеме 10M08SAU169.

Составлена принципиальная схема устройства, учитывающая поставленные критерии. Так же учтены рекомендации производителя по обеспечению цифрового питания устройства, аналогового питания, источнику опорного напряжения, подключению тактовых генераторов, сигнала сброса.

Проведена трассировка платы управления, учитывающая поставленные критерии. Учтены рекомендации производителя по трассировке корпусов BGA. Большая часть нижнего слоя платы залита земляным полигоном, сшитым переходными отверстиями с земляными полигонами верхнего слоя, для обеспечения пути минимального сопротивления для обратных токов сигналов. Минимизированы пересечения сигнальных дорожек.

Подготовлены слои шелкографии, описывающие имеющиеся разъемы на плате. Подробно описано назначение каждого контакта универсального разъема, и соответствующий ему контакт ПЛИС.

Таблицы сопоставления контактов ПЛИС контактам универсального разъема и схемы платы находятся в приложении А.

# Заключение.

Проведен анализ современных тенденций по цифровым устройствам управления. Выбраны наиболее перспективные представители направлений микроконтроллеров и ПЛИС.

Изучена документация на выбранные устройства. В соответствие с поставленными требованиями и рекомендациями производителей разработаны платы управления лабораторным макетом. Для платы на основе микроконтроллера достигнута совместимость с рядом устройств различных серий.

Произведен монтаж и апробация платы на основе микроконтроллера. Написано ПО, позволяющее протестировать широкий набор возможностей платы.

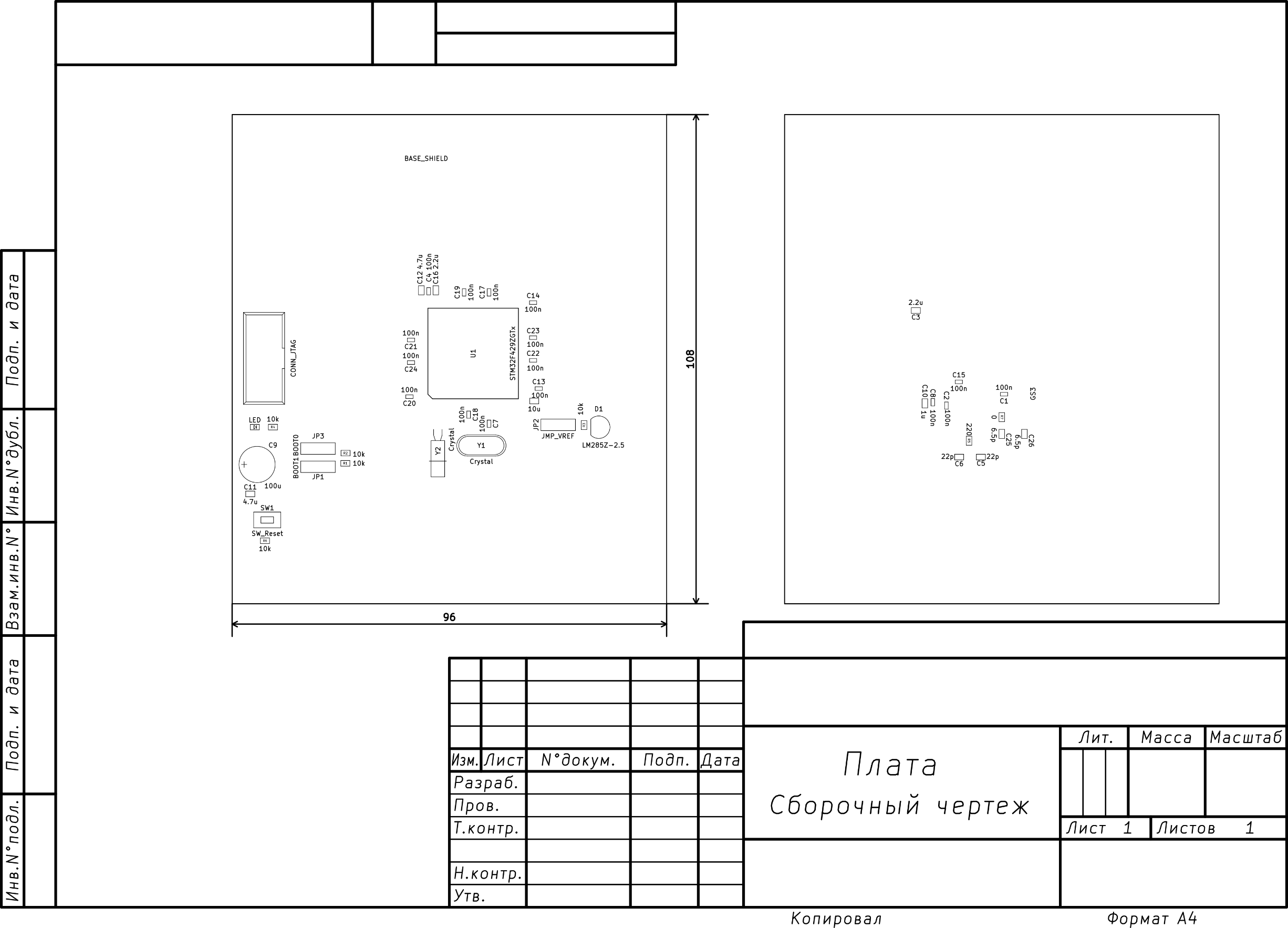
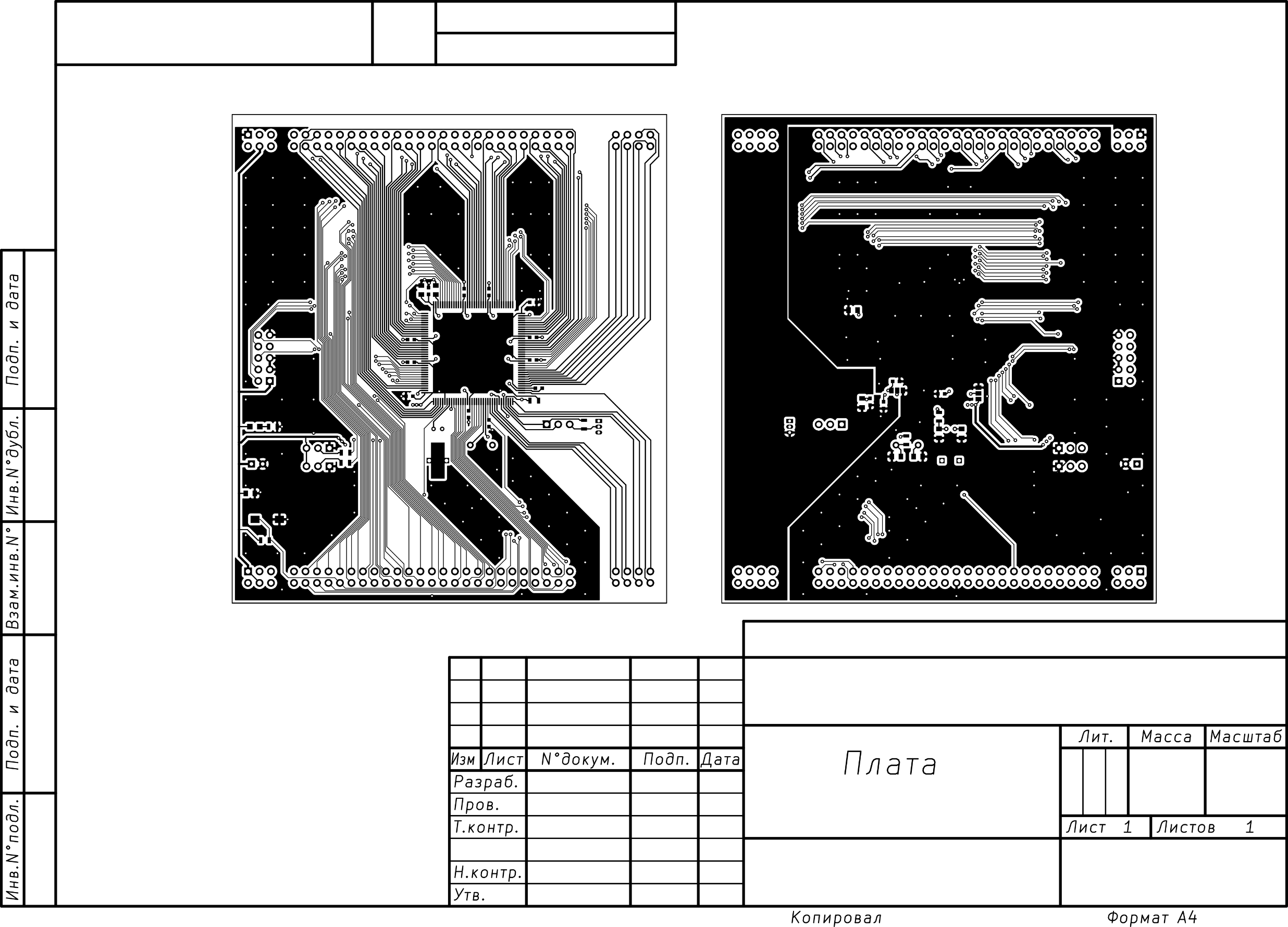
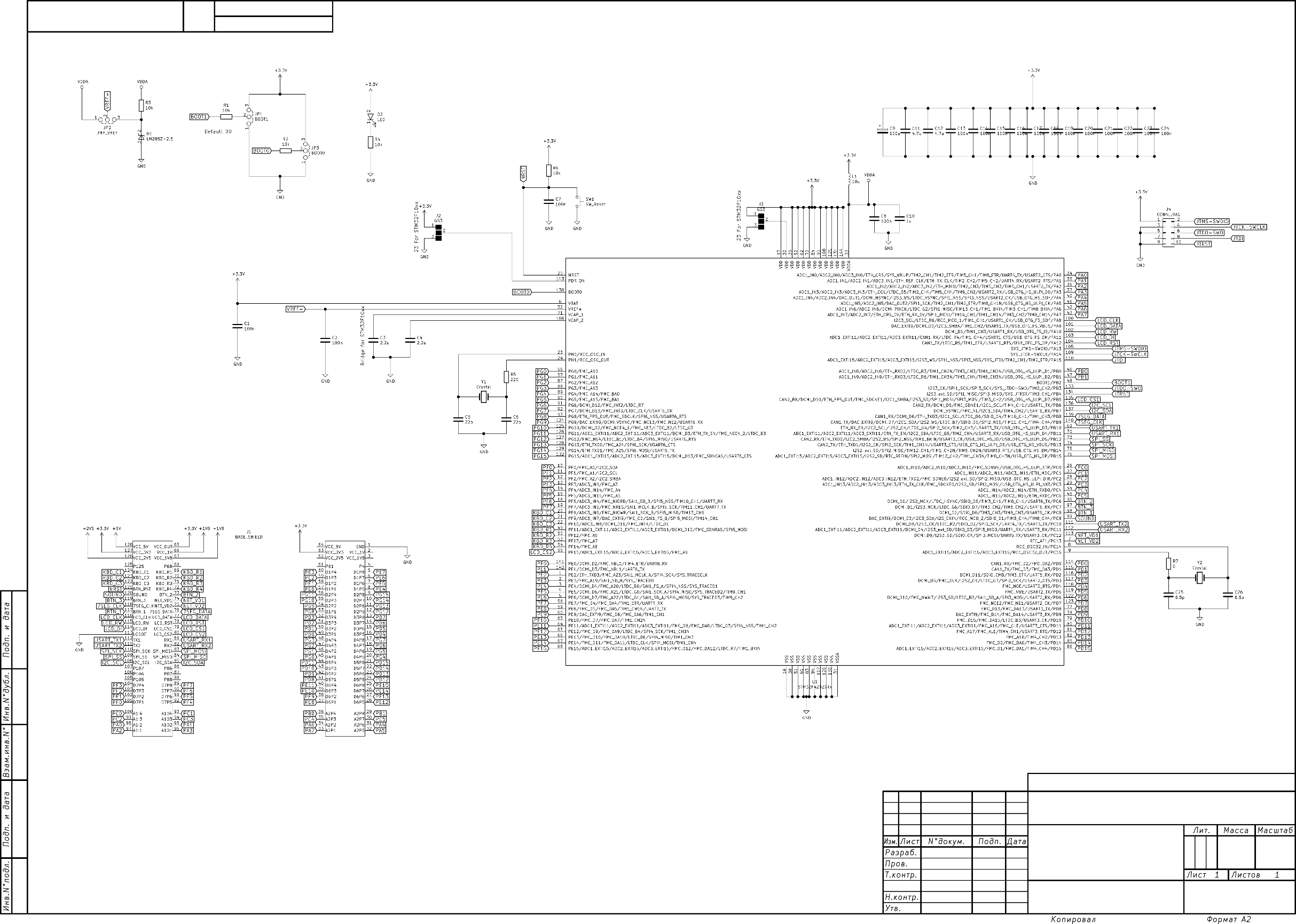
# Список литературы

1. Ingar Fredriksen, Pal Kastnes, «Choosing a MCU for your next design; 8 bit or 32 bit?», Atmel, 2014
2. National Instruments, Комплект виртуальных измерительных приборов для учебных лабораторий NI ELVIS II, Руководство пользователя, учебный центр "Центр технологий National Instruments", 2008
3. ST Microelectronics, AN2867 Oscillator design guide for STM8S, STM8A and STM32 microcontrollers, Rev 10, 2015
4. ST Microelectronics, AN4488 Getting started with STM32F4xxxx MCU hardware development, Rev 6, 2016
5. ST Microelectronics, High-density performance line ARM®-based 32-bit MCU with 256 to 512KB Flash, USB, CAN, 11 timers, 3 ADCs, 13 communication interfaces, Rev 12, 2015
6. ST Microelectronics, ARM Cortex-M4 32b MCU+FPU, 225DMIPS, up to 2MB Flash/256+4KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 20 comm. interfaces, camera & LCD-TFT, Rev 9, 2016
7. ST Microelectronics, RM0008 STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM®-based 32-bit MCUs, Rev 16, 2015
8. Intel, M10-OVERVIEW MAX 10 FPGA Device Overview, 2017
9. Intel, M10-DATASHEET MAX 10 FPGA Device Datasheet, 2017
10. Intel, MAX 10 FPGA Device Architecture, 2016
11. Intel, UG-M10ADC MAX 10 Analog to Digital Converter User Guide, 2017
12. Intel, UG-M10CONFIG MAX 10 FPGA Configuration User Guide, 2017
13. Xilinx, UG1099 Recommended Design Rules and Strategies for BGA Devices, 2016

# Приложение А

Таблица А.1 Соответствие контактов универсального разъема контактам МК

|  |  |  |  |
| --- | --- | --- | --- |
| Номер контакта | Название сигнала | Контакт МК | Комментарий |
| 1 | GND | GND |  |
| 2 | VCC\_1V | NC | Нет подключения |
| 3 | VCC\_1V8 | NC | Нет подключения |
| 4 | NC | NC |  |
| 5..8 | D1P8..5 | PE7..4 |  |
| 9..12 | D2P8..5 | PG15..12 |  |
| 13..16 | D3P8..5 | PD7..4 |  |
| 17..20 | D4P8..5 | PG7..4 |  |
| 21..24 | D5P8..5 | PD15..12 |  |
| 25..28 | D6P8..5 | PE15..12 |  |
| 29..32 | A2P8 | PB1 | ADC\_IN9 |
| 30 | A2P7 | PC5 | ADC\_IN15 |
| 31 | A2P6 | PA4 | ADC\_IN4/DAC\_OUT1 |
| 32 | A2P5 | PA5 | ADC\_IN5/DAC\_OUT2 |
| 33 | A2P1 | PA7 | ADC\_IN7 |
| 34 | A2P2 | PA6 | ADC\_IN\_6 |
| 35 | A2P3 | PC4 | ADC\_IN14 |
| 36 | A2P4 | PB0 | ADC\_IN8 |
| 37..40 | D6P1..4 | PE8..11 |  |
| 41..44 | D5P1..4 | PD8..11 |  |
| 45..48 | D4P1..4 | PG0..3 |  |
| 49..52 | D3P1..4 | PD0..3 |  |
| 53..56 | D2P1..4 | PG8..11 |  |
| 57..60 | D1P1..4 | PE0..3 |  |
| 61 | NC | NC |  |
| 62 | VCC\_2V5 | NC | Нет подключения |
| 63 | VCC\_3V3 | VCC |  |
| 64 | VCC\_5V | NC | Нет подключения |
| 65 | VCC\_OUT | VCC |  |
| 66 | VCC\_1V | NC | Нет подключения |
| 67 | VCC\_1V8 | NC | Нет подключения |
| 68 | NC | NC |  |
| 69..72 | KBD\_R1..4 | PF11..14 |  |
| 73 | BTN\_2 | PC6 |  |
| 74..75 | NET\_VD1..2 | PC12..13 |  |
| 76 | 7\_SEG\_DATA | PB8 |  |
| 77 | LCD\_DATA | PA9 | USART1\_TX |
| 78 | LCD\_RST | PA12 |  |
| 79 | LCD\_CS1 | PB5 |  |
| 80 | LCD\_CS2 | PF15 |  |
| 81 | USART\_RX1 | PB11 | USART3\_RX |
| 82 | USART\_RX2 | PC11 | USART3\_RX/UART4\_RX |
| 83 | SPI\_MOSI | PB15 | SPI2\_MOSI |
| 84 | SPI\_MISO | PB14 | SPI2\_MISO |
| 85 | I2C\_SDA | PB7 | I2C1\_SDA/USART1\_RX |
| 86..88 | NC | NC |  |
| 89..92 | D8P8..5 | PF7..4 |  |
| 93 | A1O4 | PC1 | ADC\_IN11 |
| 94 | A1O3 | PC3 | ADC\_IN13 |
| 95 | A1O2 | PA1 | ADC\_IN1 |
| 96 | A1O1 | PA3 | ADC\_IN3 |
| 97 | A1I1 | PA2 | ADC\_IN2 |
| 98 | A1I2 | PA0 | ADC\_IN0 |
| 99 | A1I3 | PC2 | ADC\_IN12 |
| 100 | A1I4 | PC0 | ADC\_IN10 |
| 101..104 | D8P1..4 | PF0..3 |  |
| 105..107 | NC | NC |  |
| 108 | I2C\_SCL | PB6 | I2C1\_SCL/USART1\_TX |
| 109 | SPI\_SS | PB12 |  |
| 110 | SPI\_SCK | PB13 |  |
| 111 | UART\_TX2 | PC10 | USART3\_TX/UART4\_TX |
| 112 | UART\_TX1 | PB10 | USART3\_TX |
| 113 | LCD\_OE | GND |  |
| 114 | LCD\_DI | PA11 |  |
| 115 | LCD\_RW | PA10 |  |
| 116 | LCD\_CLK | PA8 | USART1\_CK |
| 117 | BTN\_1 | PC8 |  |
| 118 | 7\_SEG\_CLK | PB9 |  |
| 119 | BTN\_3 | PC7 |  |
| 120 | SOUND | PC9 | TIM3\_CH4 |
| 121 | BTN\_RST | RESET |  |
| 122..124 | KBD\_C3..1 | PF10..8 |  |
| 125 | NC | NC |  |
| 126 | VCC\_2V5 | NC | Нет подключения |
| 127 | VCC\_3V3 | VCC | VCC |
| 128 | VCC\_5V | NC | Нет подключения |



# Приложение Б

Исходные коды стандартных библиотек и библиотек CMSIS не приведены.

#### Macros.h

#ifndef \_SYS\_MACROS\_H

#define \_SYS\_MACROS\_H

#define SET\_BIT(REG, BIT) ((REG) |= (BIT))

#define CLEAR\_BIT(REG, BIT) ((REG) &= ~(BIT))

#define READ\_BIT(REG, BIT) ((REG) & (BIT))

#define CLEAR\_REG(REG) ((REG) = (0x0))

#define WRITE\_REG(REG, VAL) ((REG) = (VAL))

#define READ\_REG(REG) ((REG))

#define MODIFY\_REG(REG, CLEARMASK, SETMASK) WRITE\_REG((REG), (((READ\_REG(REG)) & (~(CLEARMASK))) | (SETMASK)))

#define POSITION\_VAL(VAL) (\_\_CLZ(\_\_RBIT(VAL)))

#endif //SYS\_MACROS

#### Queue\_circular.h

#include <stdint.h>

#ifndef \_QUEUE\_CIRCULAR\_H

#define \_QUEUE\_CIRCULAR\_H

#define queue\_define(qtype, qsize, qname); \

typedef struct \

{ \

queue\_hdr\_t hdr; \

qtype items[qsize]; \

} qname;

#define queue\_init(q) \

{ \

q.hdr.front = q.hdr.rear = 0; \

q.hdr.size = sizeof(q.items) / sizeof(q.items[0]); \

}

#define queue\_put(q, item) \

{ \

q.items[q.hdr.rear] = item; \

q.hdr.rear = (q.hdr.rear + 1) % q.hdr.size; \

}

//q.hdr.rear = (q.hdr.rear + 1) & (q.hdr.size-1);

#define queue\_get(q, item) \

{ \

item = q.items[q.hdr.front]; \

q.hdr.front = (q.hdr.front + 1) % q.hdr.size; \

}

//q.hdr.front = (q.hdr.front + 1) & (q.hdr.size-1);

#define queue\_peek(q, item) \

{ \

item = q.items[q.hdr.front]; \

}

#define queue\_front(q, item) \

item = q.items[q.hdr.front]

#define queue\_empty(q) (q.hdr.front == q.hdr.rear)

#define queue\_full(q) ((q.hdr.rear + 1) % q.hdr.size == q.hdr.front)

//#define queue\_full(q) ((q.hdr.rear + 1) & (q.hdr.size-1) == q.hdr.front)

//private, do not access directly

typedef struct

{

volatile uint8\_t front;

volatile uint8\_t rear;

volatile uint8\_t size;

} queue\_hdr\_t;

#endif //\_QUEUE\_CIRCULAR\_H

#### App\_module.h

#include <stdint.h>

#include "stm32f10x.h"

#include "macros.h"

#ifndef \_APP\_MODULE\_H

#define \_APP\_MODULE\_H

typedef struct

{

void (\*init)(void\*);

void (\*deinit)(void\*);

} app\_module\_typedef;

#endif //\_QUEUE\_CIRCULAR\_H

#### Adc\_module.h

#include <stdint.h>

#include "app\_module.h"

#ifndef \_ADC\_MODULE\_H

#define \_ADC\_MODULE\_H

typedef struct

{

app\_module\_typedef app\_module;

uint8\_t (\*\_ready)(void);

void (\*\_read)(void);

uint32\_t (\*\_get)(void);

uint32\_t (\*\_callback)(uint32\_t);

} adc\_module\_typedef;

extern adc\_module\_typedef adc\_module;

#endif //\_ADC\_MODULE\_H

#### Blink\_module.h

#include <stdint.h>

#include "app\_module.h"

#ifndef \_BLINK\_MODULE\_H

#define \_BLINK\_MODULE\_H

typedef struct

{

app\_module\_typedef app\_module;

void (\*\_set)();

void (\*\_reset)();

void (\*\_toggle)();

} blink\_module\_typedef;

extern blink\_module\_typedef blink\_module;

#endif //\_BLINK\_MODULE\_H

#### Pwm\_module.h

#include <stdint.h>

#include "app\_module.h"

#ifndef \_PWM\_MODULE\_H

#define \_PWM\_MODULE\_H

typedef struct

{

app\_module\_typedef app\_module;

void (\*\_setPeriod)(uint16\_t);

void (\*\_setPulse)(uint16\_t);

void (\*\_timerCallback)(void);

void (\*\_compareCallback)(void);

} pwm\_module\_typedef;

extern pwm\_module\_typedef pwm\_module;

#endif //\_UART\_MODULE\_H

#### Uart\_module.h

#include <stdint.h>

#include "app\_module.h"

#ifndef \_UART\_MODULE\_H

#define \_UART\_MODULE\_H

typedef struct

{

app\_module\_typedef app\_module;

void (\*\_send)(void\*, int);

void (\*\_receive)(void\*, int);

uint8\_t (\*\_hasData)(void);

} uart\_module\_typedef;

extern uart\_module\_typedef uart\_module;

#endif //\_UART\_MODULE\_H

#### Spi\_module.h

#include <stdint.h>

#include "app\_module.h"

#ifndef \_SPI\_MODULE\_H

#define \_SPI\_MODULE\_H

typedef struct

{

app\_module\_typedef app\_module;

void (\*\_send)(void\*, int);

void (\*\_receive)(void\*, int);

uint8\_t (\*\_hasData)(void);

} spi\_module\_typedef;

extern spi\_module\_typedef spi\_module;

#endif //\_SPI\_MODULE\_H

#### Adc\_module.c

#include <adc\_module.h>

static volatile uint8\_t adc\_ready\_flag = 0;

static volatile uint32\_t adc\_value;

static void adc\_init(void\* params)

{

RCC->APB2ENR |= RCC\_APB2ENR\_IOPAEN;

MODIFY\_REG ((GPIOA->CRL), (GPIO\_CRL\_MODE1 | GPIO\_CRL\_CNF1),

(0x0)); //Keep at 0 for analog input

RCC->APB2ENR |= RCC\_APB2ENR\_ADC1EN; //Enable ADC1 clock

ADC1->CR2 |= ADC\_CR2\_ADON; //Wake up ADC

ADC1->CR2 |= ADC\_CR2\_RSTCAL; //Reset calibration;

while(ADC1->CR2 & ADC\_CR2\_RSTCAL); //Wait for calibration to reset

ADC1->CR2 |= ADC\_CR2\_CAL; //Start calibration;

while(ADC1->CR2 & ADC\_CR2\_CAL); //Wait for calibration

MODIFY\_REG(ADC1->SMPR2, (ADC\_SMPR2\_SMP1), (ADC\_SMPR2\_SMP1\_1)); //13.5 cycles sampling

ADC1->SQR1 &= ~(ADC\_SQR1\_L); //Clear sequence length to keep adc single-shot

ADC1->SQR3 |= ADC\_SQR3\_SQ1\_0; //Set first conversion to first channel (0-th reset state)

ADC1->CR1 |= ADC\_CR1\_EOCIE; //Enable end of conversion interrupt

ADC1->CR2 |= ADC\_CR2\_EXTSEL; //111 to SWSTART

NVIC\_EnableIRQ(ADC1\_2\_IRQn); //Enable ADC interrupt vector

}

static void adc\_deinit(void\* params)

{

RCC->APB2ENR &= ~RCC\_APB2ENR\_ADC1EN; //Disable ADC1 clock

}

static uint8\_t adc\_ready()

{

return adc\_ready\_flag;

}

static void adc\_read()

{

adc\_ready\_flag = 0;

ADC1->CR2 |= ADC\_CR2\_ADON; //Start conversion

}

static uint32\_t adc\_get()

{

while(!adc\_ready\_flag);

return adc\_value;

}

void ADC1\_2\_IRQHandler()

{

if (ADC1->SR & ADC\_SR\_EOC)

{

ADC1->SR &= ~ADC\_SR\_EOC;

adc\_ready\_flag = 1;

adc\_value = ADC1->DR & (0x0000FFFF);

if (adc\_module.\_callback)

adc\_module.\_callback(adc\_value);

}

}

adc\_module\_typedef adc\_module =

{

{

adc\_init,

adc\_deinit

},

adc\_ready,

adc\_read,

adc\_get

};

#### Pwm\_module.c

#include <stdint.h>

#include "pwm\_module.h"

//MODE [1:0]

//00: Input mode (reset state)

//01: Output mode, max speed 10 MHz.

//10: Output mode, max speed 2 MHz.

//11: Output mode, max speed 50 MHz

//CNF [1:0]

//In input mode (MODE[1:0]=00):

//00: Analog mode

//01: Floating input (reset state)

//10: Input with pull-up / pull-down

//11: Reserved

//In output mode (MODE[1:0] > 00):

//00: General purpose output push-pull

//01: General purpose output Open-drain

//10: Alternate function output Push-pull

//11: Alternate function output Open-drain

static void pwm\_init(void\* params)

{

RCC->APB2ENR |= RCC\_APB2ENR\_IOPBEN;

MODIFY\_REG ((GPIOB->CRL), (GPIO\_CRL\_MODE1 | GPIO\_CRL\_CNF1),

(GPIO\_CRL\_CNF1\_1 //Alternate Function Output Push-pull

| GPIO\_CRL\_MODE1\_1)); //Output mode 2 MHz

RCC->APB1ENR |= RCC\_APB1ENR\_TIM3EN; //Enable timer3 clock

MODIFY\_REG(TIM3->CR1, (TIM\_CR1\_DIR | TIM\_CR1\_ARPE), (TIM\_CR1\_ARPE));

WRITE\_REG(TIM3->PSC, 1024);

MODIFY\_REG(TIM3->CCMR2, (TIM\_CCMR2\_OC4M | TIM\_CCMR2\_OC4PE | TIM\_CCMR2\_OC4FE),

(TIM\_CCMR2\_OC4M\_2 | TIM\_CCMR2\_OC4M\_1 | //PWM Mode 1

TIM\_CCMR2\_OC4PE)); //OC1 Preload Enable

TIM3->ARR = 512; //Period

TIM3->CCR4 = 128; //Compare

TIM3->EGR |= TIM\_EGR\_UG;

TIM3->CR1 |= TIM\_CR1\_CEN; //Enable timer

TIM3->CCER |= TIM\_CCER\_CC4E;

TIM3->DIER = TIM\_DIER\_UIE | TIM\_DIER\_CC4IE; //Enable interrupts on update and CC1

NVIC\_EnableIRQ(TIM3\_IRQn); // Enable TIM3 interrupt vector

}

static void pwm\_deinit(void)

{

TIM3->CR1 &= ~TIM\_CR1\_CEN;

}

static void pwm\_setPeriod(uint16\_t val)

{

TIM3->ARR = val;

}

static void pwm\_setPulse(uint16\_t val)

{

TIM3->CCR4 = val;

}

void TIM3\_IRQHandler()

{

static int p = 512;

static int o = 0;

static int delta = -2;

if(TIM3->SR & TIM\_SR\_UIF) // if UIF flag is set

{

TIM3->SR &= ~TIM\_SR\_UIF; // clear UIF flag

if (pwm\_module.\_timerCallback)

pwm\_module.\_timerCallback();

}

if(TIM3->SR & TIM\_SR\_CC4IF)

{

TIM3->SR &= ~TIM\_SR\_CC4IF;

if (pwm\_module.\_compareCallback)

pwm\_module.\_compareCallback();

}

}

pwm\_module\_typedef pwm\_module =

{

{

pwm\_init,

pwm\_deinit

},

pwm\_setPeriod,

pwm\_setPulse

};

#### Blink\_module.c

#include <stdint.h>

#include "blink\_module.h"

#define BLINK\_PORT GPIOG

#define BLINK\_SETPIN GPIO\_BSRR\_BS12

#define BLINK\_CLEARPIN GPIO\_BSRR\_BR12

#define BLINK\_DATAPIN GPIO\_ODR\_ODR12

static uint16\_t val;

static void init\_blink(void\* params)

{

RCC->APB2ENR |= RCC\_APB2ENR\_IOPGEN;

BLINK\_PORT->CRL = 0x22222222; //All pins output 2 MHz

BLINK\_PORT->CRH = 0x22222222;

val = 1;

//MODIFY\_REG ((BLINK\_PORT->CRH), (GPIO\_CRH\_MODE12 | GPIO\_CRH\_CNF12), (GPIO\_CRH\_MODE12\_1));

//MODIFY\_REG ((BLINK\_PORT->CRH), (GPIO\_CRH\_MODE8 | GPIO\_CRH\_CNF8), (GPIO\_CRH\_MODE8\_1));

}

static void deinit\_blink(void)

{

}

static void set\_blink(void)

{

//BLINK\_PORT->ODR = 0xAAAA;

BLINK\_PORT->ODR = val;

val = (val << 1) | (val >> (sizeof(val)\*8 - 1));

//BLINK\_PORT->BSRR = BLINK\_SETPIN;

}

static void reset\_blink(void)

{

//BLINK\_PORT->ODR = 0x5555;

//BLINK\_PORT->BSRR = BLINK\_CLEARPIN;

}

static void toggle\_blink(void)

{

//BLINK\_PORT->ODR ^= BLINK\_DATAPIN;

}

blink\_module\_typedef blink\_module =

{

{

init\_blink,

deinit\_blink

},

set\_blink,

reset\_blink,

toggle\_blink

};

#### Uart\_module.c

#include "uart\_module.h"

#include "queue\_circular.h"

#define UART\_RX\_BUFFER\_SIZE 128

#define UART\_TX\_BUFFER\_SIZE 128

queue\_define(uint8\_t, UART\_RX\_BUFFER\_SIZE, rx\_queue\_t);

queue\_define(uint8\_t, UART\_TX\_BUFFER\_SIZE, tx\_queue\_t);

static volatile rx\_queue\_t rx\_queue;

static volatile tx\_queue\_t tx\_queue;

static volatile int rxn = 0;

static void init\_uart(void\* params)

{

queue\_init(rx\_queue); //Initialize circular queues

queue\_init(tx\_queue);

RCC->APB2ENR |= RCC\_APB2ENR\_IOPAEN; //Enable clock to IO port A

MODIFY\_REG ((GPIOA->CRH), (GPIO\_CRH\_MODE9 | GPIO\_CRH\_CNF9),

(GPIO\_CRH\_CNF9\_1 | //Alternate Function Output Push-pull

GPIO\_CRH\_MODE9\_1)); //Output mode 2 MHz

MODIFY\_REG ((GPIOA->CRH), (GPIO\_CRH\_MODE10 | GPIO\_CRH\_CNF10),

(GPIO\_CRH\_CNF10\_0)); //Floating input (reset state)

RCC->APB2ENR |= RCC\_APB2ENR\_USART1EN; //Enable USART1 clock

USART1->CR1 |= USART\_CR1\_UE; //Enable USART

USART1->CR1 &= ~USART\_CR1\_M; //0: 1 start, 8 data, n stop; 1: 1 start, 9 data, n stop

USART1->CR2 &= ~USART\_CR2\_STOP; //00: 1 stop bit; 01: 0.5 stop bits; 10: 2 stop bits; 11: 1.5 stop bits

//BAUD = Fck / (16\*USARTDIV)

//For 9600 @ 48MHz => 48000000/(16\*9600) = 0d312.5

//Fraction = 16\*0d0.5 = 0d8 = 0x8

//Mantissa = 0d312 = 0x138

USART1->BRR = 0x1388;

USART1->CR1 |= USART\_CR1\_RXNEIE; //Enable RX Not Empty interrupt

USART1->CR1 |= USART\_CR1\_RE; //Enable Receiver

NVIC\_EnableIRQ(USART1\_IRQn); //Enable UART interrupt

}

static void deinit\_uart(void)

{

GPIOC->BSRR = GPIO\_BSRR\_BR13;

}

static void send\_uart(void\* data, int len)

{

uint8\_t \*dtmp = (uint8\_t\*)data;

while(len--)

{

while (queue\_full(tx\_queue)); //Wait until there is some space in tx buffer

queue\_put(tx\_queue, \*dtmp++);

}

USART1->CR1 |= USART\_CR1\_TXEIE; //Enable TX Empty interrupt

USART1->CR1 |= USART\_CR1\_TCIE; //Enable Transmission Complete interrupt

USART1->CR1 |= USART\_CR1\_TE; //Enable Transmitter

}

static void receive\_uart(void\* data, int len)

{

uint8\_t \*dtmp = (uint8\_t\*)data;

while (len--)

{

while (queue\_empty(rx\_queue)); //Wait until there is some data in rx buffer

queue\_get(rx\_queue, \*dtmp++);

}

}

static uint8\_t hasData\_uart(void)

{

return !queue\_empty(rx\_queue);

}

void USART1\_IRQHandler()

{

if (USART1->SR & USART\_SR\_ORE)

{

GPIOB->BSRR = GPIO\_BSRR\_BS8;

}

while(USART1->SR & USART\_SR\_RXNE)

{

//USART1->SR &= ~USART\_SR\_RXNE;

queue\_put(rx\_queue, USART1->DR);

rxn++;

}

if(USART1->SR & USART\_SR\_TXE)

{

USART1->SR &= ~USART\_SR\_TXE;

if (!queue\_empty(tx\_queue)) //If TX buffer not empty

{

queue\_get(tx\_queue, USART1->DR); //Send next char

}

}

if(USART1->SR & USART\_SR\_TC)

{

USART1->SR &= ~USART\_SR\_TC;

if (queue\_empty(tx\_queue))

{

USART1->CR1 &= ~USART\_CR1\_TXEIE; //Disable TX Empty interrupt

USART1->CR1 &= ~USART\_CR1\_TCIE; //Disable Transmission Complete interrupt

USART1->CR1 &= ~USART\_CR1\_TE; //Disable Transmitter

}

}

}

uart\_module\_typedef uart\_module =

{

{

init\_uart,

deinit\_uart

},

send\_uart,

receive\_uart,

hasData\_uart

};

#### Spi\_module.c

#include <spi\_module.h>

#include "queue\_circular.h"

#define SPI\_RX\_BUFFER\_SIZE 32

#define SPI\_TX\_BUFFER\_SIZE 32

queue\_define(uint8\_t, SPI\_RX\_BUFFER\_SIZE, rx\_queue\_t);

queue\_define(uint8\_t, SPI\_TX\_BUFFER\_SIZE, tx\_queue\_t);

static volatile rx\_queue\_t rx\_queue;

static volatile tx\_queue\_t tx\_queue;

static volatile int rxn = 0;

static void spi\_init(void\* params)

{

queue\_init(rx\_queue); //Initialize circular queues

queue\_init(tx\_queue);

RCC->APB2ENR |= RCC\_APB2ENR\_IOPAEN; //Enable clock to IO port A

MODIFY\_REG ((GPIOA->CRL), (GPIO\_CRL\_MODE4 | GPIO\_CRL\_CNF4),

(GPIO\_CRL\_CNF4\_1 | //Alternate Function Output Push-pull

GPIO\_CRL\_MODE4\_0)); //Output mode 10 MHz

MODIFY\_REG ((GPIOA->CRL), (GPIO\_CRL\_MODE5 | GPIO\_CRL\_CNF5),

(GPIO\_CRL\_CNF5\_1 | //Alternate Function Output Push-pull

GPIO\_CRL\_MODE5\_0)); //Output mode 10 MHz

MODIFY\_REG ((GPIOA->CRL), (GPIO\_CRL\_MODE6 | GPIO\_CRL\_CNF6),

(GPIO\_CRL\_CNF6\_0)); //Floating input (reset state)

MODIFY\_REG ((GPIOA->CRL), (GPIO\_CRL\_MODE7 | GPIO\_CRL\_CNF7),

(GPIO\_CRL\_CNF7\_1 | //Alternate Function Output Push-pull

GPIO\_CRL\_MODE7\_0)); //Output mode 10 MHz

RCC->APB2ENR |= RCC\_APB2ENR\_SPI1EN; //Enable SPI Clock

MODIFY\_REG(SPI1->CR1, (SPI\_CR1\_BR),

SPI\_CR1\_BR\_2); // BR 100: Fclk/32

//2. Select the CPOL and CPHA bits to define one of the four relationships between the data transfer and the serial clock (see Figure 239).

SPI1->CR1 |= SPI\_CR1\_MSTR; //Master mode

SPI1->CR2 |= SPI\_CR2\_SSOE; //Hardware control of Slave Select

//SPI1->CR2 |= SPI\_CR2\_TXEIE; //Enable TX Empty interrupt

SPI1->CR2 |= SPI\_CR2\_RXNEIE; //RX Not Empty interrupt

SPI1->CR1 |= SPI\_CR1\_SPE; //Enable SPI

NVIC\_EnableIRQ(SPI1\_IRQn); //Enable SPI interrupt

}

static void spi\_deinit(void)

{

}

static void spi\_send(void\* data, int len)

{

uint8\_t \*dtmp = (uint8\_t\*)data;

while(len--)

{

while (queue\_full(tx\_queue)); //Wait until there is some space in tx buffer

queue\_put(tx\_queue, \*dtmp++);

}

SPI1->CR2 |= SPI\_CR2\_TXEIE; //Enable TX Empty interrupt

}

static void spi\_receive(void\* data, int len)

{

uint8\_t \*dtmp = (uint8\_t\*)data;

while (len--)

{

while (queue\_empty(rx\_queue)); //Wait until there is some data in rx buffer

queue\_get(rx\_queue, \*dtmp++);

}

}

static uint8\_t spi\_hasData(void)

{

return !queue\_empty(rx\_queue);

}

void SPI1\_IRQHandler()

{

while(SPI1->SR & SPI\_SR\_RXNE)

{

queue\_put(rx\_queue, SPI1->DR);

rxn++;

}

if(SPI1->SR & SPI\_SR\_TXE)

{

SPI1->SR &= ~SPI\_SR\_TXE;

if (!queue\_empty(tx\_queue)) //If TX buffer not empty

{

queue\_get(tx\_queue, SPI1->DR); //Send next char

}

else

SPI1->CR2 &= ~SPI\_CR2\_TXEIE; //Disable TX Empty interrupt

}

}

spi\_module\_typedef spi\_module =

{

{

spi\_init,

spi\_deinit

},

spi\_send,

spi\_receive,

spi\_hasData

};

#### main.c

#include <stddef.h>

#include "stm32f10x.h"

#include "app\_module.h"

#include "pwm\_module.h"

#include "blink\_module.h"

#include "uart\_module.h"

#include "adc\_module.h"

#include "spi\_module.h"

#include "MFRC522.h"

int UART\_ReceiveToken(void\*, uint8\_t);

void UART\_TransmitString(void\*);

void ADC\_Callback(uint32\_t);

void PWM\_TimerCallback(void);

void PWM\_CompareCallback(void);

void PCD\_Init(void);

volatile uint16\_t period = 512;

volatile int16\_t pulse = 128;

volatile int16\_t delta = 0;

volatile int16\_t direction = 1;

int main(void)

{

adc\_module.\_callback = ADC\_Callback;

pwm\_module.\_compareCallback = PWM\_CompareCallback;

pwm\_module.\_timerCallback = PWM\_TimerCallback;

app\_module\_typedef\* modules[] = {

(app\_module\_typedef\*)&blink\_module,

(app\_module\_typedef\*)&pwm\_module,

(app\_module\_typedef\*)&uart\_module,

(app\_module\_typedef\*)&adc\_module,

(app\_module\_typedef\*)&spi\_module

};

int modules\_len = sizeof(modules)/sizeof(modules[0]);

for (int i=0; i<modules\_len; i++)

{

modules[i]->init(NULL);

}

PCD\_Init();

blink\_module.\_set();

char\* hello = "Hello World!\r";

uart\_module.\_send(hello, 13);

char msgbuf[32];

int msglen;

char\* n;

while(1)

{

if (uart\_module.\_hasData())

{

msglen = UART\_ReceiveToken(msgbuf, 31);

if (strcmp(msgbuf, "PWM") == 0)

{

msglen = UART\_ReceiveToken(msgbuf, 31);

period = strtol(msgbuf, n, 10);

if (pulse > period)

pulse = period;

//msglen = UART\_ReceiveToken(msgbuf, 31);

//pulse = strtol(msgbuf, n, 10);

if (period == 0)

continue;

pwm\_module.\_setPeriod(period);

//pwm\_module.\_setPulse(pulse);

UART\_TransmitString("PWM OK\r");

}

}

}

}

void PWM\_TimerCallback()

{

blink\_module.\_reset();

adc\_module.\_read();

pulse += delta;

if (pulse > period)

{

delta \*= -1;

pulse = period;

}

if (pulse < 0)

{

delta \*= -1;

pulse = 0;

}

pwm\_module.\_setPulse(pulse);

}

void PWM\_CompareCallback()

{

blink\_module.\_set();

}

void ADC\_Callback(uint32\_t value)

{

uint16\_t val = ((double)period / (double)0xFFF) \* value / 32.0;

//uint16\_t val = ((double)period / (double)0xFFF) \* 0x400 / 32.0;

if (delta > 0)

delta = val;

else

delta = val \* -1;

//TIM3->CCR1 = val;

}

int UART\_ReceiveToken(void \*data, uint8\_t maxlen) //maxlen should be 1 less than buffer size to allow zero-termination

{

uint8\_t \*dtmp = (uint8\_t\*)data;

uint8\_t curlen = 0;

uint8\_t chr;

uint8\_t exit = 0;

while (maxlen--)

{

uart\_module.\_receive(&chr, 1);

switch (chr) {

case ' ':

case '\t':

case '\r':

case '\n':

if (curlen > 0)

exit = 1;

}

if (exit)

break;

\*dtmp++ = chr;

curlen++;

}

\*dtmp = 0; //terminate string

return curlen;

}

void UART\_TransmitString(void \*data) //Zero-terminated string

{

uint32\_t len = 0;

uint8\_t \*dtmp = (uint8\_t\*)data;

while(\*dtmp != 0)

{

len++;

\*dtmp++;

}

uart\_module.\_send(data, len);

}

# Приложение В

Таблица В.1 Соответствие контактов универсального разъема контактам ПЛИС

|  |  |  |  |
| --- | --- | --- | --- |
| Номер контакта | Название сигнала | Контакт МК | Комментарий |
| 1 | GND | GND |  |
| 2 | VCC\_1V | NC | Нет подключения |
| 3 | VCC\_1V8 | NC | Нет подключения |
| 4 | NC | NC |  |
| 5 | D1P8 | G12 |  |
| 6 | D1P7 | H10 |  |
| 7 | D1P6 | J13 |  |
| 8 | D1P5 | J10 |  |
| 9..12 | D2P8 | K12 |  |
| 10 | D2P7 | L12 |  |
| 11 | D2P6 | M12 |  |
| 12 | D2P5 | N11 |  |
| 13 | D3P8 | L11 |  |
| 14 | D3P7 | M10 |  |
| 15 | D3P6 | K10 |  |
| 16 | D3P5 | M9 |  |
| 17 | D4P8 | M8 |  |
| 18 | D4P7 | K8 |  |
| 19 | D4P6 | K7 |  |
| 20 | D4P5 | N6 |  |
| 21 | D5P8 | J6 |  |
| 22 | D5P7 | M5 |  |
| 23 | D5P6 | K5 |  |
| 24 | D5P5 | M4 |  |
| 25 | D6P8 | N3 |  |
| 26 | D6P7 | L3 |  |
| 27 | D6P6 | M2 |  |
| 28 | D6P5 | L2 |  |
| 29 | A2P8 | K2 |  |
| 30 | A2P7 | J2 |  |
| 31 | A2P6 | H4 |  |
| 32 | A2P5 | H2 |  |
| 33 | A2P1 | H1 |  |
| 34 | A2P2 | H3 |  |
| 35 | A2P3 | J1 |  |
| 36 | A2P4 | K1 |  |
| 37 | D6P1 | L1 |  |
| 38 | D6P2 | M1 |  |
| 39 | D6P3 | N2 |  |
| 40 | D6P4 | M3 |  |
| 41 | D5P1 | L4 |  |
| 42 | D5P2 | N4 |  |
| 43 | D5P3 | L5 |  |
| 55 | D5P4 | N5 |  |
| 45 | D4P1 | K6 |  |
| 46 | D4P2 | M7 |  |
| 47 | D4P3 | N7 |  |
| 48 | D4P4 | N8 |  |
| 49 | D3P1 | N9 |  |
| 50 | D3P2 | L10 |  |
| 51 | D3P3 | N10 |  |
| 52 | D3P4 | M11 |  |
| 53 | D2P1 | N12 |  |
| 54 | D2P2 | M13 |  |
| 55 | D2P3 | L13 |  |
| 56 | D2P4 | K13 |  |
| 57 | D1P1 | K11 |  |
| 58 | D1P2 | J12 |  |
| 59 | D1P3 | H13 |  |
| 60 | D1P4 | G13 |  |
| 61 | NC | NC |  |
| 62 | VCC\_2V5 | NC | Нет подключения |
| 63 | VCC\_3V3 | VCC |  |
| 64 | VCC\_5V | NC | Нет подключения |
| 65 | VCC\_OUT | VCC |  |
| 66 | VCC\_1V | NC | Нет подключения |
| 67 | VCC\_1V8 | NC | Нет подключения |
| 68 | NC | NC |  |
| 69 | KBD\_R1 | F10 |  |
| 70 | KBD\_R2 | F12 |  |
| 71 | KBD\_R3 | E10 |  |
| 72 | KBD\_R4 | E13 |  |
| 73 | BTN\_2 | D13 |  |
| 74 | NET\_VD1 | C13 |  |
| 75 | NET\_VD2 | C12 |  |
| 76 | 7\_SEG\_DATA | B12 |  |
| 77 | LCD\_DATA | A12 |  |
| 78 | LCD\_RST | B10 |  |
| 79 | LCD\_CS1 | A10 |  |
| 80 | LCD\_CS2 | C9 |  |
| 81 | USART\_RX1 | A9 |  |
| 82 | USART\_RX2 | A8 |  |
| 83 | SPI\_MOSI | D7 |  |
| 84 | SPI\_MISO | A7 |  |
| 85 | I2C\_SDA | A6 |  |
| 86..88 | NC | NC | Нет подключения |
| 89 | D8P8 | A5 |  |
| 90 | D8P7 | A4 |  |
| 91 | D8P6 | A3 |  |
| 92 | D8P5 | A2 |  |
| 93 | A1O4 | C1 | ADC1IN5 |
| 94 | A1O3 | B1 | ADC1IN6 |
| 95 | A1O2 | F1 | ADC1IN7 |
| 96 | A1O1 | E1 | ADC1IN8 |
| 97 | A1I1 | E4 | ADC1IN4 |
| 98 | A1I2 | E3 | ADC1IN3 |
| 99 | A1I3 | C2 | ADC1IN2 |
| 100 | A1I4 | D1 | ADC1IN1 |
| 101 | D8P1 | B2 |  |
| 102 | D8P2 | B3 |  |
| 103 | D8P3 | B4 |  |
| 104 | D8P4 | B5 |  |
| 105..107 | NC | NC | Нет подключения |
| 108 | I2C\_SCL | B6 |  |
| 109 | SPI\_SS | E6 |  |
| 110 | SPI\_SCK | B7 |  |
| 111 | UART\_TX2 | E7 |  |
| 112 | UART\_TX1 | E8 |  |
| 113 | LCD\_OE | GND |  |
| 114 | LCD\_DI | D9 |  |
| 115 | LCD\_RW | C10 |  |
| 116 | LCD\_CLK | A11 |  |
| 117 | BTN\_1 | C11 |  |
| 118 | 7\_SEG\_CLK | B11 |  |
| 119 | BTN\_3 | B13 |  |
| 120 | SOUND | D12 |  |
| 121 | BTN\_RST | DEV\_CLRn |  |
| 122 | KBD\_C3 | D11 |  |
| 123 | KBD\_C2 | E12 |  |
| 124 | KBD\_C1 | F13 |  |
| 125 | NC | NC | Нет подключения |
| 126 | VCC\_2V5 | NC |  |
| 127 | VCC\_3V3 | VCC |  |
| 128 | VCC\_5V | NC |  |

