**ИНСТРУКЦИЯ по РАБОТЕ с ПЛАТОЙ СОТА и КРОСС.**

**2015 год**

**Содержание**

1.Основные Термины.

1.2.Генерация SSH ключа и создание репозитория.

2.Аппаратный Состав компонентов СУЛ

2.1. Плата СОТА и СОТА+

2.1.1. Старт плат СОТА и СОТА+

2.1.2. Загрузка исполняемого образа СОТА и СОТА+

2.1.3. Карта eMMC памяти(ПЗУ) плат СОТА и СОТА+

2.1.4. Карта DDR памяти (ОЗУ) плат CОТА и СОТА+

2.1.5.

2.2 Плата КРОСС

2.3 Плата ЛИНК

2.4 Макетная плата Beagleone Rev.c

2.4.1 Сборка и запись на SD карту MLO,u-boot,Linux kernel,and file system.

3.Установка и Настройка Среды разработки для работы с ПО СОТА и КРОСС.  **–** **10 стр.**

3.1.Настройка #define

3.2.

3.3.

3.4.

3.5.

3.6.

4.Обновление ПО плат СОТА, СОТА+, КРОСС.

4.1.Прошивка стартового загрузчика МLO first loader

4.2.Прошивка приложения application

4.3.Прошивка таблиц: ТФА и Поправочных таблиц.

5.

6.Описание Библиотек входящих в состав ПО Starter-Ware

6.1.Библиотека mmcsdlib для работы с eMMC флэшкой.

6.1.1 Функционал

**1.Основные Термины.**

СУЛ – Система Управления Лучом

ПЛАТА СОТА - Предназначена для соединения с ППМ

ПЛАТА СОТА+ - Предназначения для соединения с ПМ модулем(имеет большие габаритные размеры)

КРОСС -

ЛИНК -

ТФА -

ВСК  **-** Встроенная система контроля.

БСО - Блок сигнальной обработки.

**1.Генерацию SSH ключа для доступа для доступа нового пользователя. к репозиторию. Из LINUX**

**1.1.Создаём ключ :**

ssh-keygen -t rsa -b 4096 -C "skdvoip@gmail.com"

Generating public/private rsa key pair.

Enter file in which to save the key (/root/.ssh/id\_rsa): skdbso

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in **skdbso.**

Your public key has been saved in **skdbso.pub.**

The key fingerprint is: **(дешифратор ключа ниже)**

8b:e6:48:29:15:63:26:33:87:ac:e9:a8:6b:dc:22:7c skdvoip@gmail.com

В итоге в папке получили два ключа:

**skdbso -** (приватный ключ)

**skdbso.pub -** (публичный ключ)

**1.2 добавляем приватный ключ в клиент ssh:**

ssh-add skdbso

**1.3 Добавляем ключ в АО”НПК Пеленгатор”**

Идём и добавляем наш публичный ключ в Пеленгатор.

**http://srv-gitlab-rts.npk-pelengator.ru/profile/keys**

2.Выкачиваем Репозиторий в каталог -> /home/pelengator/workspace

2.1 Создаём пользователя pelengator в linux

useradd –D

useradd pelengator

лучше добавлять пользователя через администратор YAST тогда будет всё ok

2.2 Выкачиваем Репозиторий из svn Пеленгатора

git init

git remote add origin [git@srv-gitlab-rts.npk-pelengator.ru:donskov\_yd/BSO\_SPO.git](mailto:git@srv-gitlab-rts.npk-pelengator.ru:donskov_yd/BSO_SPO.git)

git pull origin master

Нужно обязательно так как все пути сборки у них прописаны для этого каталога.

//Для загрузки образа нужно.

2.Аппаратный Состав компонентов СУЛ

**2.1** Плата СОТА и СОТА+**:**

**ЦП(CPU): Обозначение на схеме U19**

Тип (наименование)Texas Instrument AM3358 Sitara.

-ревизия Архитектуры : ARMv7-A (ARM V7A8)

-ревизия Ядра: :Cortex-A8 revision r3p2

-плавающая точка :аппаратная поддержка VFPv3 floating point

-аппаратный сопроцессор SIMD инструкций : NEON core

-аппаратные сопроцессоры PRU 2 штуки: (ревизия) PRU-ICSS1. **PRU**

-ОЗУ (встроенное в ядро\_1 (on\_chip\_SRAM) ) -> 64 К/байт

-ОЗУ (встроенное в ядро\_2)(L3\_SRAM) -> 64 К/байт

**ОЗУ(DDR): Обозначение на схеме U20**

Тип:MT41K256M16HA-125IT:E **->**512 М/байт 4(Г/бит)

1.25ns @ CL = 11 (DDR3-1600) -125

**ПЗУ(eMMC):**

Тип: MTFC4GLVEA-0M WT

Ёмкость:4 Г/байт

**ПЛИС:**

-CPLD MAX10 10M08DCV81

**УПРАВЛЕНИЕ ПИТАНИЕМ:**

**ДАТЧИК ТЕМПЕРАТУРЫ:**

ADS1013 IRUG

**2.1.1 Старт плат СОТА и СОТА+**

Нижний уровень.

.СТАРТ в составе антенной системы. СУЛ + БСО

СТАРТ отдельно платы СОТА и СОТА+ (с точки зрения ARM CortexA8)

- Подали питание +27[В], первым запустился TPS65217C и подал питание

VDD\_DDR\_1.5[v] ,VDD\_MPU\_1.2[v],VDD\_CORE\_1.1[v] на CortexA8

- Первым делом в соответствии с Аппаратной конфигурацией определяемой пинами SYSBOOT[] мы пытаемся найти бинарный образ в emmc флэш ,данный вид загрузки называется (26.1.7 ***Memory Booting p.4923),***если мы находим образ специально подготовленный для emmc флэшки(\*) то копируем его во внутреннюю встроенную в кристалл **onchipRAM**  память 64+64=128 К/байт, начинаем его исполнение с функции main

Основной Функционал Первичного загрузчика:

СТАРТ отдельно платы СОТА и СОТА+ (с точки зрения msp430)

-

**2.1.2. Загрузка исполняемого образа СОТА и СОТА+**

-Первым делом наше приложение после старта должно скопировать из eMMC памяти и разместить в DDR (ОЗУ):Четыре таблицы ТФА(8Мб\*4=64Мбайт),

Четыре Поправочные таблиц(8Мб\*4=64 Мбайт).

**2.1.3. Карта eMMC памяти(ПЗУ) СОТА и СОТА+**

Работа с флэшкой eMMC происходит без файловой системы поблочно.

1 блок = 512 [байт] соответственно адресация у нас идёт номерами блоков.

Адрес начала блока [16 М/байт] = blk\_num = 32768(0x8000) (количество блоков)

Адрес начала блока [128 М/байт] = blk\_num = 262144(0x40000)(количество блоков)

Адрес начала блока [256 М/байт]= blk\_num = 524288(0x80000)(количество блоков)

Размер блока [8 М/байт] = blk\_len = 16384(0x4000) (количество блоков)

Общая ёмкость нашей eMMC флэш памяти = 4 Г/байт.

Для простоты 32 битной адресации, используем 1Г/байт = 1 073 7418 24 байт=

2097152 [блока] в нашей флэшке.

В платах СОТА и СОТА+ карта памяти (первый бета вариант имеет следующий вид):

**----------------------------- blk\_start =0x0(0) emmc\_mem\_start =0x0(0 байт)**

Первичный Загрузчик можно будет потом расширить до [512 К/байт]

MLO first Loader

**-----------------------------blk\_end =0x200(512) emmc\_mem\_end =0x40000(262144 байт)**

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**-----------------------------blk\_start=0x400(1024) emmc\_mem\_start =0x80000(524288 байт)**

Исполняемое

Приложение mem\_len=[512 К/байт], mem\_size 0x00080000(524288) , blk\_len 1024(0x400)

(Application)

**-----------------------------blk\_end =0x800(2048) emmc\_mem\_end =0x100000 (1048576 байт )**

**Таблицы Фаз и Амплитуд**

**Отступаем 16 М/Байт в eMMC флэш и начинаем таблицы общий размер которых 64М/байт:**

Таблица фаз и Амплитуд №1 **blk\_start =0x8000** **blk\_cnt=**0x4000 **blk\_end=0xC000**

Таблица фаз и Амплитуд №2 **blk\_start =0xC000** **blk\_cnt=**0x4000 **blk\_end=0x10000**

Таблица фаз и Амплитуд №3 **blk\_start =0x10000** **blk\_cnt=**0x4000 **blk\_end=0x14000**

Таблица фаз и Амплитуд №4 **blk\_start =0x14000** **blk\_cnt=**0x4000 **blk\_end=0x18000**

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**Поправочные Таблицы**

**Отступаем 128 [М/Байт] и начинаем таблицы:**

Поправочная Таблица №1 **blk\_start =**0x40000 **blk\_cnt=**0x4000 **blk\_end=**0x44000

Поправочная Таблица №2 **blk\_start =**0x44000 **blk\_cnt=**0x4000 **blk\_end=**0x48000

Поправочная Таблица №3 **blk\_start =**0x48000 **blk\_cnt=**0x4000 **blk\_end=**0x4C000

Поправочная Таблица №4 **blk\_start =**0x4C000 **blk\_cnt=**0x4000 **blk\_end=**0x50000

ЛОГ куда пишем последние события в системе:

**Отступаем 256 [М/Байт] и начинаем Лога:**

**1 ЛОГ**

Функционал:

-прошита mlo и application и ревизия

**blk\_start\_log1 =**0x80000 **blk\_cnt\_log1=**0x1 **blk\_end\_log1=**0x80001 [512 байт]

**2 ЛОГ**

Функционал:

**blk\_start\_log2 =**0x80002 **blk\_cnt\_log2=**0x1 **blk\_end\_log2=**0x80003 [512 байт]

**2.1.4. Карта DDR памяти(ОЗУ) СОТА и СОТА+**

Используется DDR3 память при старте исполняемое приложение(Apllication) копируется из eMMC в DDR память и начинает там исполняться, следующий этап оно должно скопировать таблицы из eMMc в DDR.

Тип:MT41K256M16HA-125IT:E **->**512 М/байт 4(Г/бит)

Длинна одной таблицы 8М/байт =8 388 608 байт = 0x00800000 (hex)

Длинна области хлама 128 м/байт = 134 217 728 байт = 0x08000000 (hex)

Стартовый Адрес памяти нашего приложения: 0x80000000 (entry\_point)

**Приложение(Application)** 0x80000000 - 0x80080000 [512 К/байт]

**Отступаем 16 М/Байт начинаем таблицы:**

**Таблица фаз и Амплитуд №1** 0x81000000 - 0x81800000 [8/Мбайт]

**Таблица фаз и Амплитуд №2** 0x81800000 - 0x82000000 [8/Мбайт]

**Таблица фаз и Амплитуд №3** 0x82000000 - 0x82800000[8/Мбайт]

**Таблица фаз и Амплитуд №4** 0x82800000 **-** 0x83000000[8/Мбайт]

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

**Поправочная Таблица №1** 0x83000000 - 0x83800000 [8/Мбайт]

**Поправочная Таблица №2** 0x83800000 – 0x84000000 [8/Мбайт]

**Поправочная Таблица №3** 0x84000000 – 0x84800000 [8/Мбайт]

**Поправочная Таблица №4** 0x84800000 – 0x85000000 [8/Мбайт]

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Область используемая для отступаем 128 М/байт = 0x08000000

**0x88000000 – 0x8D000000 Хлам и всякое разное**

2.2 Плата КРОСС

**2.3 Плата ЛИНК**

**2.4 Макетная плата Beagleone Rev.c**

**2.4.1 Сборка и запись на SD карту MLO,u-boot,Linux kernel,and file system.**

- Подготовка SD карты для u-boot

Нужно отформатировать SD карту в формате FAT32 и сделать один раздел

с параметрами PRIMARY и ACTIVE иначе грузиться не будет

* Очистка eMMC флэщки из под u-boot загрузчика который записан

На SD карту только для **beagleBone Rev.c**

Версии u-boot U-Boot 2014.07 (Oct 28 2015 - 14:38:52)

-Грузимся с SD карты в командной строке U-BOOT набираем команды:

**1. mmc dev 1 (выбираем устройстов 1 eMMC флэш)**

-switch to partitions #0, OK

-mmc1(part 0) is current device

**2. Операция Стирания флэшки**

//blk //cnt

**mmc erase 0x0 0x200**  // Стираем первичный загрузчик

**mmc erase 0x400 0x400** // Стираем приложение

**mmc erase 0x80000 0x1** // Стираем ЛОГ1

DRAM bank = 0x00000000

-> start = 0x80000000

-> size = 0x20000000

Пишем рабочий образ на eMMC флэшку

Операция записи eMMC флэшки из под u-boot

--EMMC

mmc dev 1

mmc rescan

**----------------------MLO**

fatload mmc 0 0x80000000 MLO (если с SD карты)

tftp 0x80000000 boc/bco/MLO (если с tftp)

mmc erase 0x0 0x200

mmc write 0x80000000 0x0 0x200

**---------------------ENV**

fatload mmc 0 0x81000000 uEnv.txt (если с SD карты)

tftp 0x80010000 boc/bco/uEnv.txt (если с tftp)

mmc erase 0x800 0x10

mmc write 0x80100000 0x800 0x10

**--------------------------------------------------------------------------------------------**

**----------------------u-boot**

fatload mmc 0 0x81000000 u-boot.img (если с SD карты)

tftp 0x80100000 boc/bco/u-boot.img (если с tftp)

mmc erase 0x300 0x400

mmc write 0x80100000 0x300 0x400

ENV

Операция очистки от uEnv.txt

U-Boot# mmc rescan

U-Boot# fatload mmc 0 0x81000000 uEnv.txt

U-Boot# env import -t 0x81000000 $filesize

U-Boot# boot

**3.Установка и Настройка Среды разработки для работы с ПО СОТА и КРОСС.**

**3.1. Установка #Define**

Где прописываем: Properties for arm\_skd\_syl ->Arm Compiler ->Advanced Options ->Predefined Symbols -> PRU0\_ADD (чтобы включить PRUмодуль).

**3.2 Сборка Проекта и Отладка с Поддержкой ARM + PRU ядер.**

Нужно в проекте arm\_skd\_syl выполнить настройки:

0.Определить #define PRU0\_ADD 1 в файле armskd\_syl\_main.c

1.Определить Predefined Symbols -> PRU0\_ADD в настройках проекта.

2.Определить в RUN ->Debug Configurations выбрать проект arm\_skd\_syl

**Target Configuration:** C:\Share\sota\_skd\gel\_config\syl\_A8andPRU.ccxml

**Выбрать галочки:** CortxA8 и PRU\_0.

${command} ${flags} ${output\_flag} ${output} ${inputs} ${PROJECT\_LOC}\PRU\_HexUtil.cmd

**3.3 Сборка Проекта и Отладка только ARM ядер.**

StarterWare – Development Tools

Free bare-bones software suite for ARM9, Cortex-A8 and C6000 DSP

Не используем ОС используем открытый пакет Starter-ware

Версия Среды разработки: Code Composer Studio Version: 6.1.0.00104

Версия компилятора ARM ядра: arm\_5.1.6

Версия компилятора PRU ядер: ti-cgt-pru\_2.1.1

В starterware бинарный образ который получается в результате компиляции не содержит заголовка eMMC raw header который позволяет грузиться без наличия файловой системы на носителе, в данном случае eMMC флэш ,после старта CPU ARMv7 ищет его в 0 блоке. Если не находит, то пытается найти загрузочную запись файловой системы.

сборка и прошивка (first bootloader)

-добавить в Code Composer Studio проект Project -> Import CCS Projects

//Путь откуда импортируем CCS проект

C:\Share\starterwarefree-code\build\armv7a\cgt\_ccs\am335x\beaglebone\bootloader

\*для сборки boot предварительно должны собрать(если нужно) статические библиотеки (lib)

-rts7A8\_A\_le\_v3\_eabi.lib

-utils.lib

-platform.lib

-libmmcsd.lib

-drivers.lib

//Где расположены HEX массивы

eMMC\_raw\_header.h ->сам заголовок размер массива 512 байт.

//После того как мы собрали boot идём в директорию:

C:\Share\starterwarefree-code\binary\armv7a\cgt\_ccs\am335x\beaglebone\bootloader\Debug\_MMCSD

Там файл: boot\_ti.bin (это наш бинарник с заголовочной информацией от TI)

Записываю своё приложение для App (Application)

C:\Share\sota\_skd\arm\_skd\_syl\Debug\arm\_skd\_syl\_ti.bin

Превращаем его в массив данных.

//Потом можно либо его через функции f\_read ,(считать) я его просто конверчу в HEX массив

И добавляю заголовок RAW header.

1.2\_сборка и прошивка (приложения Application)

-ревизии

По умолчанию приложения Starter Ware собираються без raw eMMC header

-ревизии

-заголовок (eMMC raw header)

- включить режим 8 битной адресации для eMMC (на будущее)

-сборка приложения **- arm\_skd\_syl**

**- arm\_skd\_syl**

- Старт Приложения.

Адрес входа в функцию main\*()

shilyaev\_kd

megabridge\_5084

Адрес корпоративного Сервера.

<http://srv-gitlab-rts.npk-pelengator.ru/shemanaev_as/cross>

http://srv-gitlab-rts.npk-pelengator.ru/u/shilyaev

<http://srv-gitlab-rts.npk-pelengator.ru/u/doroshenko_kp>

[k.d.shilyaev@npk-pelengator.ru](mailto:k.d.shilyaev@npk-pelengator.ru)

Расположение проектов на корпоративном svn сервере.

git@srv-gitlab-rts.npk-pelengator.ru:shilyaev/sota\_skd.git

git@srv-gitlab-rts.npk-pelengator.ru:shilyaev/starterwarefree-code.git

git@srv-gitlab-rts.npk-pelengator.ru:shilyaev/pru-software-support-package.git

Настрйока CodeCOmposerStudio для проектов. Файл инициализации no\_config

Проверка что всё работает:

Run -> Debug Configurations ->arm\_skd\_syl (Main) ->Target Configuration

C:\Share\sota\_skd\gel\_config\syl\_no\_config.ccxml (внизу нажимаем кнопку debug)

Последовательность работы с Ядрами:

CONNECT -> CORTEX A8

Scripts -> AM335x System Initialization -> AM335x\_Beagleblack\_initialization

Scripts -> PRU\_CAPE -> PRU\_Cape\_Init.

Должны получить такой вывод.

После это можно работать я ядрами.

-Работа с Таймерами

Загрузчик MLO

On power on, the MLO is first detected and copied and executed from the OCMC0 RAM. The MLO then copies the application image (app) from the card to the DDR and passes the control to the application. On Successful boot, following messages appear on the serial console.

**ROM Bootloader (RBL)**

*On Power-up execution begins from on-chip ROM which:*

*1.Checks the boot setting (configured by DIP switches)*

*2.Loads the secondary bootloader from the respective medium to on-chip RAM*

*3.Hands over control to the on-chip RAM*

**On-chip RAM Bootloader**

*The secondary bootloader then:*

*1.Initialises PLLs*

*2.Enables Peripheral Clocks*

*3.Initialises DDR*

*4.Sets the operating frequency (up to 720 MHz)*

*5.Loads the application from respective medium to off-chip DDR*

*6.Hands over control to DDR*

**StarterWare Application**

*The StarterWare Application code is then executed from DDR*

Runtime support library

rtsv7A8\_A\_le\_v3\_eabi.lib

to be re-built with the ARMv3 options (check the <project>.cmd build file to see the library dependencies ) :

Connect to the Virtual serial Port on PC (115200, 8, N, 1)

Ключи Компиляции:

**4.Обновление ПО плат СОТА,СОТА+,КРОСС.**

4.1 Прошивка СОТА и СОТА+ rev.00

В платах СОТА и СОТА+ есть следующие виды информации, которые надо

Прошивать и обновлять по ходу работы.

-Первичный загрузчик MLO bootloader

-Исполняемое приложение Application

-Четыре таблицы фаз и амплитуд для каждого канала размером 8Мб \*4 =64 Мб

-Четыре поправочные таблицы для каждого канала 8Мб\*4=64 Мб

4.2.Прошивка Первичного загрузчика MLO и исполняемого приложения Application

4.2.1 Через консоль rs-232

Когда пришла пустая плата СОТА или СОТА+ в ней небудет исполняемого образа

Тогда мы должны загрузиться через TeraTerm VT протоколу Xmodem

Мы для загрузки используем файл boot.bin

4.3.2 Через jtag и Code Composer Studio

4.3.Прошивка Таблиц ТФА и Поправочных таблиц.

4.3.1 Через консоль rs-232

4.3.2 Через jtag и Code Composer Studio

Прежде чем прошить таблицы нужно понять какой у нас порядковый номер платы

СОТА или СОТА+, в соответствии с этим нужно выбрать номера таблиц который мы шьём.

Прошивание образа производиться через jtag для этого открываем проект wdtReset с помощью #define в файле wdtReset.c определяем какую конкретно таблицу собираемся шить и последовательность,прошивки

Загружаем и располагаем таблицы в памяти DDR как написано в разделе (2.1.4. Карта DDR памяти(ОЗУ) СОТА и СОТА+)

Как прошивать таблицы в Память ARM cortex A8(на первых порах)

Memory Browser -> Load Memory -> обязательно File Type: Ti Raw Data (если выбрать что-то другое то получим косяк с файлами)

Для ускорения записи выбираем режим 64 бита тогда 8 Мбайт копируются относительно

быстро.

Start Address определяем где у нас лежат таблицы.

**4.1 Обновление через jtag отладчик СОТА и СОТА+,КРОСС.**

**4.2 Обновление через UART отдельно СОТА и СОТА+,КРОСС.**

**4.3 Обновление в составе Антенной системы (СУЛ).**

6.Описание Библиотек входящих в состав ПО Starter-Ware

Минимальная единица информации работы с флэшкой это 512 байт можем

записать и прочитать.

6.1.Библиотека mmcsdlib для работы с eMMC флэшкой

6.1.1 Функционал

В файле syl\_mmcsd\_proto.c находиться вся реализация функций работы с eMMC.

В файле syl\_mmcsd\_proto.h находиться прототипы функций работы c eMMC.

**-Запись eMMC**

Первой вызывается функция:

**mmc\_bwrite**(**int** dev\_num, **unsigned** **int** start, **unsigned** **int** blkcnt, **const** **void** \*src)

Второй вызывается функция:

**mmc\_write\_blocks**(**struct** mmc \*mmc,**unsigned** **int** start,**unsigned** **int** blkcnt\_ext, **const** **void** \*src)

**-Чтение eMMC**

Первой вызывается функция:

**static** **unsigned** **int** **mmc\_bread**(**int** dev\_num, **unsigned** **int** start, **unsigned** **int** blkcnt, **void** \*dst)

Второй вызывается функция:

**static** **int** **mmc\_read\_blocks**(**struct** mmc \*mmc, **void** \*dst, **unsigned** **long** **long** start,**unsigned** **long** **long** blkcnt)

**-Стирание eMMC**

**unsigned** **int** **mmc\_berase**(**int** dev\_num, **unsigned** **int** start,**unsigned** **int** cnt /\*lbaint\_t blkcnt\*/)

**static** **unsigned** **int** **mmc\_erase\_t**(**struct** mmc \*mmc, **unsigned** **int** start, **unsigned** **int** blkcnt)

6.2.

6.3.

6.4.

6.5.

6.6.

driver.lib – device abstraction layer APIs for supported peripherals

Driver library contents:

–Peripheral configuration, access, and control

•peripheralInit(…)

•peripheralEnable(…) and peripheralDisable()

•peripheralIo(…)

System control

•peripheralReset(…)

•peripheralClocking(…)

platform.lib – board-specific code for configuring

peripherals

Platform library contents:

–Pinmux configuration API

•Supports EVM (including all example applications)

•Peripheral-specific APIs (ex. I2CPinMuxSetup(…))

•Complete GPIO muxing not supported

•Locks/unlocks the write-protection of the SYSCFG module (i.e. kick registers)

system.lib – ARM/DSP core specifics like interrupt controller code

System library contents:

–Cache configuration

–ARM and DSP interrupt controller

•Enable/disable interrupts

•Specify user callback functions

–Exception Handler definitions

•Define exception interrupt vector table

–Initialization & Startup Code

•IVT initialization

utils.lib – miscellaneous functionality for StarterWare applications

–UART-based STDIO replacement (puts, gets, etc.)

–Delay functionality (specified as real time, not just cycles)

–Periodic “tick” with user-specified callback function

–Command line argument handling

**-Библиотеки входящие в состав проекта. нужно закинуть в SVN**

**Для Работы с ARM ядром и перефирией**

**Расположение Include** файлов

**"${CG\_TOOL\_ROOT}/include" файлов**

**"C:\Share\starterwarefree-code\include"**

**"C:\Share\starterwarefree-code\include\hw"**

**"C:\Share\starterwarefree-code\include\armv7a"**

**"C:\Share\starterwarefree-code\include\armv7a\am335x"**

**"C:\Share\starterwarefree-code\mmcsdlib\include"**

**-drivers (третьим drivers)**

**-mmcsdlib(моя билиотека для работы с eMMC флэшкой)**

**-platform (Вторым папку platform)**

**-system (Первым собираем папку system)**

**-utils (четвёртым utils делаем)**

**Для Работы с PRU сопроцессорами.**

Расположение Lib файлов для Линковщика.

"rtsv7A8\_A\_le\_v3\_eabi.lib" //скорее всего нужно его вначало так там функции необходимые

1."C:\Share\starterwarefree-code\binary\armv7a\cgt\_ccs\mmcsdlib\Debug\libmmcsd.lib"

2."C:\Share\starterwarefree-code\binary\armv7a\cgt\_ccs\am335x\drivers\Debug\drivers.lib"

3."C:\Share\starterwarefree-code\binary\armv7a\cgt\_ccs\am335x\system\_config\Debug\system.lib"

4."C:\Share\starterwarefree-code\binary\armv7a\cgt\_ccs\am335x\beaglebone\platform\Debug\platform.lib"

5."C:\Share\starterwarefree-code\binary\armv7a\cgt\_ccs\utils\Debug\utils.lib"

Нужно пересобирать библиотеку:

Rtsv7A8\_A\_le\_v3\_eab

-Размещение в памяти таблица фаз и амплитуд (ТФА);

Размещение в памяти таблицы управления элементом (ТУЭР)

**ТАЙМЕРЫ.**

//Моя библиотека для Таймеров где я определяю функции extern для таймеров.

"C:\Share\starterwarefree-code\include\delay.h

Где лежат Определния функция работы с таймерами

// C:\Share\starterwarefree-code\platform\beaglebone\sysdelay.c