**ИНСТРУКЦИЯ ПО РАБОТЕ с БСО ver0.0.8**

**300316.**

**Содержание**

1.Установка и настройка ХОСТ ПК

2. -

3. -

Сервер для прошивок: [\\srv-fs\WorkingDocs\РТС\TFTP](file:///\\srv-fs\WorkingDocs\РТС\TFTP)

-Наш tftp сервак для скачивания прошивок

**tftp 10.5.5.8**

проверяем статус соединения:  **status**

закачиваем файлы в папку из которой запустили: **get BCO\_meteo\_1\_0\_3.rbf**

**ReadMINE(для ведения bug\_report)**

**Имя п-ля:** shilyaev\_kd

Пароль: как локальный ПК

<http://srv-redmine-rts.npk-pelengator.ru/projects/ozp/issues?set_filter=1&tracker_id=6>

http://srv-redmine-rts.npk-pelengator.ru/projects/ozp/issues?set\_filter=1&tracker\_id=6

0.Работа с Переменными Окружения в командной строке u-boot

//Обновление Ядра

**~** это означает: /home/user/ ,/home/pelengator/ ,/home/Rouben/

**Подготовка Хост ПК**

Дополнительный пакеты, которые нужно поставить на Debian 7.8.0

Для x64 обязательно до gcc 32 g++32

-multistrap\_2.1.20\_all.deb

-qwt 6.0 (libqwt6)

-qemu-user-static

-lib32z1

-lib32z1-dev

-autostart-target\_1.02-1\_all.deb (возможно это для сборки файловой системы)

1.Подготовка Хост машины (ПК) к установке

*Важное замечание создание директорий производиться под локальным пользователем*

*pelengator* ***:***

**-home/pelengator/workspace**

**-home/pelengator/bin**

Монтирование директории совместно используемой с windows под root

**mount -t vboxsf Share /mnt/SHARE**

копирование и разархивирование компиляторов*производиться под локальным пользователем pelengator и всего остального*

**-gcc4.8.tar.gz и gcc4.9.tar.gz**

Есть один момент, что при переходе от root в Debian тереться путь к компилятору.

export PATH=$PATH:/home/pelengator/bin/gcc-linaro-arm-linux-gnueabihf-4.9-2014.09\_linux/bin

export ARCH=arm

export CROSS\_COMPILE=arm-linux-gnueabihf-

**1.Установка Компилятора на Хост ПК**

* 1. *Устанавливаем Компилятор*

-Скачиваем нужный компилятор.

-Компилятор нужно положить(туда): /home/kosta/bin

-Разархивируем компилятор в папке.

-tar –xvzf gcc-linaro-arm-linux-gnueabihf-4.9-2014.09\_linux.tar.gz

*1.2 Прописываем путь к компилятору в файле : /home/kosta/.profile*

Для компилятора: 4.9-2014.09

PATH=”$HOME/bin/gcc-linaro-arm-linux-gnueabihf-4.9-2014.09\_linux/bin/:$PATH”

Для компилятора: 4.8 пытаемся собрать.

PATH=”$HOME/bin/gcc-linaro-arm-linux-gnueabihf-4.8-2014.04\_linux/bin/:$PATH”

Путь:/home/pelengator/bin/gcc-linaro-arm-linux-gnueabihf-4.9-2014.09\_linux

**2.Сборка ядра Linux на ХОСТ ПК linux-socfpga для БСО.**

**Наименование модулей БСО при выводе температуры**

LF -Совычислитель (левый дальний) , LF CN -Резерв

LN -Совычислитель (левый ближний) , BGS -Плата Гетеродинов и синхронизации

ADC -АЦП , EXT0 -Резерв

FURY -Плата FURY, BCO -Плата BCO контрольная точка U52

RN -Совычислитель (правый ближний) ,LN CN -Резерв

RF -Совычислитель(правый дальний), ,RF CN -Резерв

IFACE -Плата IFACE ,RN CN -Резерв

EXT1 -Резерв ,BCO -Плата BCO,контрольная точка U53

**КАРТА ПАМЯТИ QSPI флэшки.**

В текущей версии БСО для монтирования 4 флэшек используется jffs2 (журналируемая) файловая система

Возможно проблемы с производительностью могут быть в этом.(нужно исследовать).

Размер = 64 М/байт = 0x0400\_0000= 671 088 64 байт. N25Q512A

0x00000------------------------------------------------------- **START of FLASH**

**preloader-mkpimage.bin**

0x40000 (256 К/байт) 262 144 ---------------------

**uboot-env.bin**

0x42000 (264 К/байт) 270 336 ----------------------

0x60000 (384 К/байт) 393 216 ----------------------

**u-boot.img**

0xF0000 (960 К/байт) 983 040 ----------------------

**socfpga.dtb**

0x200\_000 (2 М/байт) 2 097 152 ----------------------

**zImage (6 М/байт)**

0x800\_000 (8 М/байт) 8 388 608 ----------------------

**bco\_meteo.rbf (8 М/байт)**

0x0100\_0000 (16 М/байт) 16 772 16 ----------------------

**uRamdisk\_initramfs\_root (48 М/байт)**

0x0400\_0000(64 М/байт) 67 108 864 ----------------------**END of FLASH**

Чистим флэшку от 16М/байт до 64 М/байт

sf erase 0x01000000 0x800000 //стираем 8 М/байт работает.

0x01800000 0x800000 //стираем 8 М/байт работает.

0x02000000 0x800000 //стираем 8 М/байт

0x02800000 0x800000 //стираем 8 М/байт

0x03000000 0x800000 //стираем 8 М/байт

0x03800000 0x800000 //стираем 8 М/байт

До конца флэшки 0x0400\_0000

//Сборка u-boot

**Первичный Загрузчик (**preloader-mkpimage.bin**)**

**MLO,SPL,**

**Основные функции:**

**-** настраивает вычислитель в соответствии с результатами работы редактора системных ресурсов

Quartus Qsys

- устанавливает тактовые частоты;

- конфигурирует периферийные устройства;

- инициализирует SDRAM и подсистему коррекции ошибок ECC;

- загружает по завершению работы из QSPI flash в SDRAM и передает управление вторичному

загрузчику u-boot.

**Как Получить для(Cyclone 5 and Arria 5):**

-H\W-инженер проектирует подсистему HPS в системном редакторе **Quartus::Qsys**. Результатом его работы является сгенерированный каталог **Handsoff files** где в закрытом формате содержится описание системы

-конфигурация периферийных устройств;

-мультиплексирование и свойства выходных пинов;

-тактовые частоты;

-настройки PLL;

-конфигурация;

-тайминги SDRAM, ECC.

**Путь где лежат готовые заготовки() папка hps\_isw\_handoff**

**/home/pelengator/altera/15.0/embedded/examples/hardware/cv\_soc\_devkit\_ghrd**

**/home/pelengator/altera/15.0/embedded/examples/hardware/cv\_soc\_devkit\_ghrd/hps\_isw\_handoff/soc\_system\_hps\_0**

Копируем папку из каталога Altera в наше расположение как по инструкции ЦРТС в папку

**/home/pelengator/workspace/uboot/B5\_uboot/hps\_isw\_handoff/avhps\_hps\_0**

**find . -print0 | xargs -0 touch**

**cd /home/pelengator/workspace/uboot/B5\_uboot**

запускаем

**/home/pelengator/workspace /altera/15.0/embedded/embedded\_command\_shell.sh**

выполняем

**bsp-editor**

New->HPS указываем путь к Handoff файлам.->>

**~/uboot/B5\_uboot/hps\_isw\_handoff/avhps\_hps\_0**

**Запись на QSPI первичного загрузчика из u-boot (загрузились с SD карты)**

**tftp 0x2000000 boc/bco/preloader-mkpimage.bin**

**sf probe 0**

**sf update 0x2000000 0x00000 $filesize**

**Вторичный Загрузчик (u-boot.img)**

**/home/pelengator/altera/15.0/embedded/embedded\_command\_shell.sh**

uEnv.txt ->file which will be used by u-boot to provide the necessary

commands to boot up in the preferred configuration

**Запись на QSPI вторичного загрузчика из u-boot (загрузились с SD карты)**

**Выполняем :**

**cd /home/pelengator/workspace/uboot/B5\_uboot/software/spl\_bsp**

**make ->**build preloader

**make uboot ->**build uboot

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

**tftp 0x2000000 boc/bco/u-boot.img**

**sf probe 0**

**sf erase 0x60000 0x50000**

**sf write 0x2000000 0x60000 $filesize**

**Переменные окружения (u-boot\_env.bin)**

**-расположение утилиты для генерации переменных окружения**

/home/pelengator/workspace/uboot/B5\_uboot/software/spl\_bsp/uboot-socfpga/tools/ mkenvimage

Нужно скопировать в папку чтобы ОС всегда видела наши переменные окружения

/usr/local/bin

----------------------------------------------------------------------------------------------------------------------

--для создание файла **u-boot\_env.bin** нужен текстовый файл

**bootenv.txt** в котром в определённом формате будут расположены наши переменные

bootargs=console=ttyS0,115200

bootcmd=tftp 22000000 uImage; bootm

[...]

//Создаём файл

mkenvimage -s 0x1000 -o uboot-env.bin bootenv.txt

**tftp 0x2000000 boc/bco/** uboot-env.bin

sf probe 0

sf erase 0x40000 0x1000000

sf write **0x2000000 0x40000 0x1000**

**или**

**sf update 0x2000000 0x40000 $filesize**

---------------------------------------------------------------------------------------------------------------------

--**-расположение утилиты для генерации uImage**

/home/pelengator/workspace/uboot/B5\_uboot/software/spl\_bsp/uboot-socfpga/tools/mkimage

Нужно скопировать в папку чтобы ОС всегда видела наши переменные окружения

/usr/local/bin

-----------------------------------------------------------------------------------------------------------------------------------------------------------

//Доработка U-boot для загрузки ramboot файловой системы

-Корректировать строку она сделана для загрузки и монтирования ext2 на флэшке

bootargs=console=ttyO0,115200n8 root=/dev/ram0 rw rootfstype=ext2 //должно быть

bootargs=console=ttyO0,115200n8 root=/dev/ram0 rw rootfstype= squashfs

bootargs=console=ttyO0,115200n8 root=/dev/ram0 rw rootfstype= **ramfs**

//что происходит на БСО в uboot.

**1. bootargs=ramdisk\_size=140000 mem=1G root=/dev/ram rw rootfstype=ramfs rootwait console=ttyS0,115200 coherent\_pool=1M ecc=on ip=192.168.193.168:192.168.193.193::255.255.255.0::eth0:off**

**2.bootcmd=run fpgaqspiload; run qspiload; run ramqspiload; run ramboot**

fpgaload=tftp ${fpgadata} ${tftpdir}${fpgaimage}; fpga load 0 ${fpgadata}; run bridge\_enable\_handoff;

fpgaqspiload=sf probe 0:0; sf read ${fpgadata} ${qspifpgaaddr} ${fpgadatasize}; fpga load 0 ${fpgadata}; run bridge\_enable\_handoff

**3.** **ramboot=setenv bootargs ramdisk\_size=140000 mem=1G root=/dev/ram rw rootfstype=ramfs rootwait console=ttyS0,115200 coherent\_pool=1M ecc=on ip=${ipaddr}:${serverip}::${netmask}::eth0:off; bootz ${loadaddr} ${rdiskaddr} ${fdtaddr}**

find . -print0 | xargs -0 touch ->корректность установки времени

/home/pelengator/altera/15.0/embedded/embedded\_command\_shell.sh

u-boot-socfpga ->uboot-socfpga

make uboot

**Сборка Device Tree (dtbimage.bin) из dtc файла.**

**/home/pelengator/workspace/linux-socfpga/scripts/dtc**  ->**исполняемый файл**

**/home/pelengator/workspace/linux-socfpga/arch/arm/boot/dts** ->**конфиги**

**socfpga\_cyclone5\_socdk.dts**

**socfpga\_cyclone5\_sockit.dts**

**//Для БСО**

/home/pelengator/workspace/linux-socfpga/scripts/dtc/dtc -I dts -O dtb -o /home/pelengator/workspace/dtbimage.bin ./socfpga\_arria5\_furia.dts

**//Для Сyclone 5 SOC\_EDS**

/home/pelengator/workspace/linux-socfpga/scripts/dtc/dtc -I dts -O dtb -o /home/pelengator/workspace/dtbimage.bin /home/pelengator/workspace/linux-socfpga/arch/arm/boot/dts/socfpga\_cyclone5\_socdk.dts

!!!//работает!!!

/home/pelengator/workspace/linux-socfpga-ltsi/scripts/dtc/dtc -I dts -O dtb -o /home/pelengator/workspace/dtbimage.bin /home/pelengator/workspace/linux-socfpga-ltsi/arch/arm/boot/dts/socfpga\_cyclone5\_socdk.dts

/home/pelengator/workspace/linux-socfpga/scripts/dtc/dtc -I dts -O dtb -o /home/pelengator/workspace/dtbimage.bin /home/pelengator/workspace/linux-socfpga/arch/arm/boot/dts/soc.dts

./home/pelengator/workspace/linux-socfpga/scripts/dtc/dtc -I dtb -O dts -o soc.dts socfpga.dtb

/home/pelengator/workspace/linux-socfpga/scripts/dtc/dtc -I dtb -O dts -o soc.dts socfpga.dtb

**tftp 0x2000000 boc/bco/** dtbimage.bin

**tftp 0x2000000 boc/bco/**socfpga.dtb

**sf probe 0**

**sf erase 0xf0000 0x10000**

**sf write 0x2000000 0xf0000 $filesize**

**//run fpgaload ->загрузка с tftp**

//run fpgaqspiload -> pfuheprf c qspi

//Сборка файловой системы из armhf-main.tgz

Берём с диска архив platform.tar.gz ->разархивируем его в /home/pelengator/workspace/platform

Создаем директорию platform-build в home/pelengator/workspace

/home/pelengator/workspace/platform/armhf.sh /home/pelengator/workspace/platform-build/

Создать директорию /tmp/fs в /home/pelengator/altera/15.0/embedded/embedded\_command\_shell.sh

**перекомпилить ядро командами**

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- mrproper

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- socfpga\_pelengator2\_defconfig

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- socfpga\_defconfig

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- zImage LOADADDR=0x8000

в .../linux-socfpga/arch/arm/boot

появится файл zImage

3) Скопировать zImage на таргет командой

scp ./zImage [/root@192.168.193.168:/mnt/f1p0/](mailto:/root@192.168.193.168:/mnt/f1p0/)

4) Зайти на таргет по SSH и прошить его на флеш, как мы его прошивали командой

./fcp -v -o 0x200000 zImage /dev/mtd0

updkernel=tftp ${loadaddr} ${tftpdir}${bootimage}; sf probe 0:0; sf update ${loadaddr} ${qspibootimageaddr} ${bootima gesize}

--**Перепрошивка u-boot каким образом выполнить её если случайно затёрли?**

--Установка Сервера tftp

setenv tftpdir (директория откуда будем грузиться)

setenv serverip (ip\_adress сервера tftp)

saveenv

Перестарт

1) boot

2) по питанию

--**Настройка FPGA**

setenv fpgaimage (имя файла прошивки ПЛИС).rbf

--**Настройка Ядра**

setenv fpgaimage (имя файла прошивки ПЛИС).rbf

--Настройка Файловой системы.

setenv fpgaimage (имя файла прошивки ПЛИС).rbf

**Работа с Регистровой Спецификация из под LINUX доступ к ПЛИС.**

**--FPGA\_BRIDGE\_ADDR -> 0xFF200000**

**--FPGA\_X-Device\_ADDR -> 0xFF205000 addr[17:12] биты 5-ку**

**--FPGA\_X-Device\_ADDR -> 0xFF205000 addr[7:2] биты 7-2**

**Команды чтения Регистров**

./devmem2 <адрес регистра>

**Команда записи в Регистры**

./devmem2 <адрес регистра> w <значение>

**#define GEN\_DEV\_DEVICE\_ID\_OFFS ( 0x26 \* 4 ) /\* 0x26 \* 4 = 0x98 0xff200098 \*/**

**(0x0D \*4) =0x34 = 0xff200034, = 0xFF205034**

**Пример получение ревизии проекта**

**#define GEN\_DEV\_PROJECT\_ID\_OFFS ( 0x20 \* 4 ) /\* 0x20 \* 4 = 0x80 0xff200080 \*/**

./devmem2 0xff205080

**-Проверка ревизии** **прошивки FPGA**

-в командной строке linux ввести команду

./devmem2 0xff205080 после ввода команды мы увидим ревизию прошивки ПЛИС

./devmem2 0xff205080 после ввода команды мы увидим ревизию прошивки ПЛИС

**-Сборка Отладочной(ЦЕЛЕВОЙ) версии проекта МЕТЕО**

toolsmeteo/src/common/thread\_param.h

#define PIM\_ON или PIM\_OFF

-Чтение файлов с SSD

На хост ПК

./client\_16mb.elf -i192.168.193.168 -p24577 –bh0 –eh3b9dff m7.b

парсим 31 положение луча.

./bin31.elf –f<имя\_файла>

killall mnggeneral ->убиваем главный диспетчер.

test\_supervisor –t (смотрим температуру)

**Косяк с формулой перерасчёта мощности в [Дб] в диспетчере метео.**

Файл:/toolsmeteo/src/mngmeteo/mngmeteo.cpp

Функция:discreteToDb(uint32\_t \*buf,uint32\_t bufSize)

receivedPower.push\_back(20\*log10(double)buf(i)-240)

2.1- *Установка и Распаковка Ядра*

*Где Лежат конфиги Ядра.*

*/home/pelengator/workspace/linux-socfpga/arch/arm/configs*

*Расположение* socfpga\_pelengator\_defconfig (сборка ядра для git ЦРТС)

socfpga\_pelengator2\_defconfig(сборка ядра для Пеленгатор)

*отличие заключается в строке CONFIG\_LOCALVERSION=”+”*

tar –xvf linux-socfpga.tar //архив ядра Linux для Пеленгатора.

Переходим в директорию где находиться наше ядро.

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- mrpropper //очистка ядра

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- socfpga\_pelengator2\_defconfig //конфиг

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- zImage LOADADDR=0x8000 //сборка

make -j 4 uImage make -j 4 uImage

**Выход:** Получаем файл **zImage** который является у нас ядром. Если нужно обновить.

**3.сборка драйверов нижнего уровня модулей ядра bso\_driver на ХОСТ ПК**

3.1 - Установка и Распаковка драйверов ядра

Нужно создать папку : /home/kosta/bso\_ltib

Туда кладём: tar –xvf bco\_driver.tar.gz //драйвер для БСО.

3.2 - Сборка драйверов должна идти под user kosta.

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- KDIR=/home/pelengator/workspace/linux-socfpga clean

//очистка исполняемых файлов модуля ядра - \*.ko

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- KDIR=/home/pelengator/workspace/linux-socfpga

//сборка заново.

**Инсталяция драйвера (модуля ядра БСО)**

insmod bco\_drv.ko

./nodes.sh

**Выход:**

Главный Драйвер модуля Ядра**:**

**-**bco\_drv.ko

**Видимо тесты нижнего уровня интерфейсных плат:**

**//max10\_test**

test\_c\_1.elf- Проверяет работу ioctl для MAX10 на RCVC. Пишет и читает регистр RCVC.

test\_iface\_1.elf-Пишет данные в arinc 429 TX0. Проверят код возврата.

test\_iface\_2.elf-Пишет данные в arinc 429 TX0.

Host с Windows используя sw "Преобразователь последовательного кода" и девайс "Контур НИИРС" читает поток.

test\_iface\_3\_rx0.elf-Читает данные из arinc 429 RX0 и записывает данные в файл.

Host с Windows используя sw "Преобразователь последовательного кода" и девайс "Контур НИИРС" пишет данные.

test\_iface\_3\_rx1.elf-Читает данные из arinc 429 RX1 и записывает данные в файл.

Host с Windows используя sw "Преобразователь последовательного кода" и девайс "Контур НИИРС" пишет данные.

test\_iface\_uart\_1.elf-Пишет данные в RS422 uart0 , читает из RS422 uart1. Используем TX - RX перемычку и читем что написали.

test\_iface\_uart\_2\_3.elf-

Пишет данные в RS485 uart2 , читает из RS485 uart3. Используем TX - RX перемычку и читем что написали.

test\_supvs\_1.elf-Пишет в решистры MAX10 SUPVS (supervisor)

test\_supvs\_2.elf-Посылает programming file в софею 0 и измеряет время доставки

**//meteo \_test**

test\_iface\_mfi\_read.elf

test\_iface\_mfi\_write.elf

test\_iface\_uart\_0.elf-Читает сырые данны из bins

test\_antsys\_wr.elf-Пишет в антенную систему (RS485 uart2) , читает из антенной системы(RS485 uart3). Используем TX - RX перемычку и читем что написали.

test\_meteo\_timer155.elf-Читает отладочный Метео Таймер (155 миллисекундного интервала)

test\_meteo\_04\_708.elf- Пишет бинарный поток в arinc 708 tx0.

test\_meteo\_03.elf-Это тотже тест test\_meteo\_02.c , использующий sw врапперы

test\_meteo\_02.elf- Читает поток метео в разных режимах. Режимы вводятся через командную строку.

test\_meteo\_01.elf-Читает поток метео

**компилятор:**

arm-linux-gnueabihf-gcc (crosstool-NG linaro-1.13.1-4.8-2014.04 - Linaro GCC 4.8-2014.04) 4.8.3 20140401 (prerelease)

**4.Cборка Диспетчера верхнего уровня. (Приложение Пользователя Linux) для ARM платформы БСО.**

3.1 - Установка и Распаковка Диспетчера Верхнего Уровня

установить cmake 2.8.9 для возможности выполнения сборки.

**-** libarinc//библиотека arinc

**-** libmeteo\ameteo //библиотека meteo

**-** toolsmeteo //набор тестов для meteo.

Входят поддиректории.

1.asdatareader,

2.beaming,

3.binsdatareader,

4.common,

5.hwf,

6.mfidatareader,

7.mnggeneral, --видимо главный диспетчер с него стартовать.

8.mngmeteo,

9.uartreaderbase.

**3.2 *Сборка нужно поменять пользователя на root иначе не собирается.***

Переходим в директорию libarinc выполняем скрипт

***build\_arm.sh***

Переходим в директорию libmeteo\ameteo выполняем скрипт

***build\_arm.sh***

*Перходим в директорию toolsmeteo*

***build\_arm.sh***

**Выход:**

**------libarinc**

libarinc429.a

libarinc708.a

main\_test (исполняемый файл) приложение.

**-----libameteo\ameteo.**

libameteo.a

libameteo.a

**Инструкция по Загрузке и работе с БСО с точки зрения Программиста**

Примечание – если модуль ядра уже установлен, то повторная установка не требуется.

#### Запустить на выполнение скрипт «./test\_256\_4.sh».

#### Проверить отсутствие сообщений об ошибках в окне консоли вовремя и по результатам выполнения скрипта.

Проверка корректности чтения данных с накопителя:

#### Открыть на ПК консоль БСО и установить модуль ядра, выполнив команду «insmod sata\_driver.ko»

Примечание – если модуль ядра уже установлен, то повторная установка не требуется.

#### Запустить задачу на выполнение путем ввода команды «./serv\_10\_16mb.elf».

#### Открыть консоль ПК и запустить сценарий на выполнение, введя команду «./cl\_6001-3se-p.sh».

Примечание – результатам запуска в консоли ПК команды «./cl\_6001-3se-p.sh» является копирование файлов с твердотельного накопителя данных БСО на внутреннюю память ПК. Типичное время копирования файлов составляет 1 ч. 15 мин.

#### Проверить отсутствие сообщений об ошибках в процессе копирования данных с БСО (твердотельного накопителя данных) на ПК.

#### Проверить наличие на ПК файлов:

* lba\_00000000\_039c0fff.b
* lba\_039c1000\_07381fff.b
* lba\_07382000\_0ad42fff.b
* lba\_0ad43000\_0e703fff.b
* lba\_0e704000\_120c4fff.b
* lba\_120c5000\_15a85fff.b
* lba\_15a86000\_19446fff.b
* lba\_19447000\_1ce07fff.b

#### Для проверки корректности скопированной с носителя данных БСО информации ввести в консоль ПК команду «./check-3se-p.sh».

**ТЕСТЫ БСО Верхний Уровень.**

**Где расположены:**

Расположение тестов:home\pelengator\workspace\toolsmeteo\tests

**anglecor** - проверка процедуры коррекции луча для проверки необходимы файлы исходных данных и файл результатов, которые формируются в модели matlab.

Программа берет данные углов берутся из исходного файла и формирует файл результатов. Далее с помощью команды diff необходимо сравнить результат работы программы и результат работы модели матлаб

**binstest** - проверка приема данных от БИНС. Выводит на экран значения тангажа и угла крена, полученные от БИНС.

**mfi055057test** - проверка отправки сообщений 055 и 057 на МФИ.

Программа поочередно отправляет сообщения 055 для азимутов -30..30, а затем отправляет сообщения 057 для углов места -8..8.

Программа показывает, что после получения сообщения 057 МФИ перестает отображать графическую развертку.

**mfi055test** - проверка отправки сообщений 055 на МФИ.

Программа поочередно отправляет сообщения 055 для азимутов -30..30.

**mfi055test2** - проверка отправки сообщений 055 на МФИ одной пачкой

Программа отправляем большое количество сообщений 055 с периодом 5 секунд.

Программа показывает следующие дефекты МФИ: пропадание части сообщений 055 при смене режима, переход в режим Готов при отсутствии сообщений 055 больше 3 секунд.

**mfi429test** - проверка приема сообщений 270, 271, 272 от МФИ

Программа выводит на печать сообщения Arinc 429, и результат их распаковки.

**scanplantest** - проверка процедуры планирования сканирования

Программа выводит на печать рассчитанные азимуты и углы места для ГРП и ВРП.

**-.УСТАНОВКА РЕЖИМА МЕТЕО Автоматическая Загрузка БСО**

-Что нужно поменять.

-Прошивка Центральной ПЛИС FPGA Arria5(FURY2.1) файлом bco\_meteo.rbf(lkz)

-Изменить файл deploy.sh для Автоматической загрузки драйверов. (для метео).  
 **1.1 Отредактируйте начало ./deploy.sh так или возьмите его в аттачменте**  
#!/bin/bash  
# BOARD for SATA  
#arm039 workaround for SATA  
# should be direct connection  Board <-> PC. Any changing IP addresses can to impact to eth #neagotiations.  
**mknod /dev/satadev c 200 0  
/mnt/f1p0/bco\_test/nodes.sh**  
#ip addr del 192.168.193.91/24 dev eth0  
**insmod /mnt/f1p0/bco\_test/bco\_drv.ko  
insmod /mnt/f1p0/sata\_test/sata\_driver.ko**  
**/mnt/f1p0/mnggeneral**  
**exit 0**  
==================================================  
 **1.2 Отредактируйте  переменные окружения u-boot для загрузки   bco\_meteo.rbf    в  FPGA**  
setenv tftpdir boc/bco/  
setenv serverip 192.168.193.168  
setenv fpgaimage bco\_meteo.rbf  
saveenv

**2.УСТАНОВКА РЕЖИМА ТПО**

**2.1.Что нужно**

-Прошивка Центральной ПЛИС FPGA Arria5(FURY2.1) файлом BCO.rbf (для ТПО)

-Изменить файл deploy.sh для Автоматической загрузки драйверов. (для ТПО)

Для того чтобы вернуть обратно   ТРО

Отредактируйте начало ./deploy.sh так или возьмите его в аттачменте

#!/bin/bash  
# BOARD for SATA  
#arm039 workaround for SATA  
# should be direct connection  Board <-> PC. Any changing IP addresses can to impact to eth #neagotiations.

#mknod /dev/satadev c 200 0  
#/mnt/f1p0/bco\_test/nodes.sh  
##ip addr del 192.168.193.91/24 dev eth0  
#insmod /mnt/f1p0/bco\_test/bco\_drv.ko  
#insmod /mnt/f1p0/sata\_test/sata\_driver.ko  
#/mnt/f1p0/mnggeneral  
#exit 0

**2.2 Отредактируйте  переменные окружения u-boot для загрузки   bco.rbf    в  FPGA**  
setenv tftpdir boc/bco/  
setenv serverip 192.168.193.168  
setenv fpgaimage bco.rbf  
saveenv

После загрузки ядра

На Хост ПК(Debian whezzy) необходимо запустить: testhfamber

На БСО (Debian whezzy) необходимо запустить : soc\_bos\_rf с параметром –g(возможно)

Интеграции модуля ядра bco\_driver.ko с SW верхнего уровня

В секции инклудов включить.

#include "bco\_syst\_devices.h" содержит все имена файлов устройств

#include "bco\_syst\_ioctl.h" обеспечивает команды для чтения/записи регистров Максов и структуру для обмена с драйвером через ioctl()

#include "max10\_supvs.h" обеспечивают карту всех регистров и их адресов

#include "max10\_iface.h"

#include "max10\_rcvc.h"

#include "max10\_bgs.h"

#include "meteo.h"

#include "gen.h"

Включить в компиляцию файл

bco\_integraion.c обеспечивает правильные файловые дескрипторы для кода в bco\_inline.h

например так : $(CC) -Wall -lrt -O2 -I./ ./meteo\_tests/test\_meteo\_03.c ./meteo\_tests/bco\_integration.c -o ./meteo\_tests/test\_meteo\_03.elf

В секции инклудов включить #include "bco\_inline.h" обеспечивает

- inline функции для чтения записи регистров Максов,

write\_register , read\_register , write\_register\_block , read\_register\_block

- мапирование устройств в интерфейсы, которые включают несколько разных устройств или каналов

а также фунции для открытия и закрытия интерфейсов

**Для МФИ**

------------------------------------------------------

open\_mfi();

read( fd\_read\_mfi\_0, buf, count );

read( fd\_read\_mfi\_1, buf, count );

write( fd\_write\_mfi, buf, count );

close\_mfi();

------------------------------------------------------

**Для БИНС**

------------------------------------------------------

open( fd\_bins );

read( fd\_bins, buf, count );

write( fd\_bins, buf, count );

close( fd\_bins );

------------------------------------------------------

**Для Антенной Системы**

------------------------------------------------------

open\_antsys();

read( fd\_read\_antsys, buf, count );

write( fd\_write\_antsys, buf, count );

close\_antsys();

------------------------------------------------------

В текст программы включить

в начале - open\_sys\_maxes()

в конце - close\_sys\_maxes()

это обеспечит открытие устройства Супервайзера первым и его закрытие последним

**ТПО**

Структура Директорий для ТПО

dmitry@dmitry-ws:~/workspace/pelengator$ tree -d

├── bos\_lib

│   ├── bin

│   ├── fury-dev

│   │   ├── inc

│   │   └── src

│   ├── inc

│   └── max10\_controller

│   ├── inc

│   └── src

├── soc\_tests

│   ├── soc\_bos\_rf

│   │   ├── Debug

│   │   │   └── src

│   │   │   ├── jsmn

│   │   │   └── rf\_tests

│   │   └── src

│   │   ├── jsmn

│   │   └── rf\_tests

│   ├── test-interface

│   │   └── src

│   └── test-supervisor

│   └── src

└── testhfamber

└── src

├── chphasemodel

├── cplot

├── csensibilitymodel

├── cwidget

├── plotdlg

└── resources

└── images

31 directories

**U-BOOT**

3 directories, 20 files

rouben@arm039:~/uboot/B5\_uboot/software/spl\_bsp$ cd ..

rouben@arm039:~/uboot/B5\_uboot/software$ cd ..

rouben@arm039:~/uboot/B5\_uboot$ tree

.

├── build.sh

├── fury\_20\_eth0.patch

├── hps\_isw\_handoff

│   └── avhps\_hps\_0

│   ├── alt\_types.h

│   ├── avhps\_hps\_0.hiof

│   ├── emif.xml

│   ├── hps.xml

│   ├── id

│   ├── sdram\_io.h

│   ├── sequencer\_auto\_ac\_init.c

│   ├── sequencer\_auto.h

│   ├── sequencer\_auto\_inst\_init.c

│   ├── sequencer.c

│   ├── sequencer\_defines.h

│   ├── sequencer.h

│   ├── system.h

│   ├── tclrpt.c

│   └── tclrpt.h

├── Makefile.diff

├── prg.sh

├── prg-u.sh

└── software

└── spl\_bsp

├── build.sh

├── fury\_20\_eth0.patch

├── generated

│   ├── build.h

│   ├── iocsr\_config\_arria5.c

│   ├── iocsr\_config\_arria5.h

│   ├── pinmux\_config\_arria5.c

│   ├── pinmux\_config.h

│   ├── pll\_config.h

│   ├── reset\_config.h

│   └── sdram

│   └── sdram\_config.h

├── Makefile

├── Makefile.diff

├── Makefile.orig

├── preloader.ds

├── preloader-mkpimage.bin

├── prg.sh

├── prg-u.sh

├── settings.bsp

├── u-boot.bin

├── uboot.ds

└── uboot-socfpga -> /home/rouben/workspace/u-boot-socfpga

7 directories, 40 files

**Привет! БСО ТПО разных версий.**

Необходимо поменять в библиотеке bos\_lib функция static int max10\_rcvc\_init() val = 1; на val = 0; Затем Собрать библиотеку и Собрать приложение

Полный дифф коммита прилагаю

dmitry@dmitry-ws:~/workspace/bos\_lib$ git diff 6818de7ae diff --git a/max10\_controller/src/max10\_rcvc.c

b/max10\_controller/src/max10\_rcvc.c

index 029a3d8..fdb9ac1 100644

--- a/max10\_controller/src/max10\_rcvc.c

+++ b/max10\_controller/src/max10\_rcvc.c

@@ -84,7 +84,7 @@ int max10\_rcvc\_destroy ()

static int max10\_rcvc\_init(){

unsigned char val = 0, i = 0;

- val = 1;

+ val = 0; //changed from 1 to 0 in MAX

if (max10\_ctrl\_write\_reg (max10\_rcvc, RCVC\_REG\_CTRL, &val, 1)) return -1;

val = 3;

if (max10\_ctrl\_write\_reg (max10\_rcvc, RCVC\_REG\_ADR, &val, 1)) return -1; dmitry@dmitry-ws:~/workspace/bos\_lib$

bco\_meteo\vlib\_local\meteo\_dsp\meteo\_to\_hps.vh - заголовочный файл с регистровой моделью ПЦОС для обмена информацией с процессором

Передача пакетов с результатами обработки реализована с помощью механизма X-device

Результаты ПЦОС в виде амплитуд (модулей выходного комплексного сигнала), т.е. расчет десятичного логарифма возложен на процессор

Регистр ARINC\_CTRL выделен для передачи 32-битного слова данных с информацией о режиме работы (диапазоне дальности) от индикатора.

Переданное слово данных по процессорной шине инициирует запуск цикла зондирования.

По готовности результатов обработки модуль ПЦОС выставляет прерывание процессору, после чего процессору необходимо за время между циклами зондирования вычитать пакет данных из fifo X-device модуля.

После освобождения fifo (заполненность проверяется по чтению регистра IN\_FIFO\_STATUS\_2) процессор должен сбросить прерывание (регистр INT\_ACK) и войти в режим ожидания следующего прерывания

Структура передаваемого в процессор пакета, состоящего из 32-битных слов:

- Заголовок 16'h\_ffff (старшие 16 бит) и длина пакета (младшие 16 бит)

- Данные в виде положительных чисел, модулей комплексного выходного сигнала (кол-во слов указано в длине пакета и зависит от режима работы)

Таблица 5 – Длина пакета в зависимости от режима и диапазона дальности

|  |  |  |
| --- | --- | --- |
| Режим и диапазон дальности | Управляющие сигналы на ПЦОС (cм «Регистровая спецификация ПЦОС МЕТЕО.docx») | Длина пакета |
| ВРП/ГРП, 5 км | hor/ver = 1, vsk = 0, diap[5:0] = 000001 | 30 |
| ВРП/ГРП, 10 км | hor/ver = 1, vsk = 0, diap[5:0] = 000010 | 63 |
| ВРП/ГРП, 20 км | hor/ver = 1, vsk = 0, diap[5:0] = 000100 | 24 |
| ВРП/ГРП, 50 км | hor/ver = 1, vsk = 0, diap[5:0] = 001000 | 64 |
| ВРП/ГРП, 100 км | hor/ver = 1, vsk = 0, diap[5:0] = 010000 | 131 |
| ВСК | hor/ver = 0, vsk = 1, diap[5:0] = xxxxxx | 3 |

**Патчи на код. Патч.1(БИНС) -> mngmeteo.cpp**

void MngMeteoImpl::setASParam(double azIn, double elIn)

{

// Устанавливаем параметры луча в ПЛИС (Установка параметров луча осуществляется в аппаратном проекте в соответствие с временной диаграммой)

// MngAfar::setASParam(azIn, elIn) ; //comment

MngAfar::setASParam(angleCorrectionMeteoOutputData.azOut,angleCorrectionMeteoOutputData.elOut) ;

}

**Патчи на код. Патч.2 -> soc.cpp**

void SocImpl::socMain()

{

if (flush)

{

//m\_paramLogs[i].ofs << resString << std::endl ; //1 log закоментировать запись на диск

Запись на флешку spi. //серъёзно грузит софт 60~80 %

}

// Записываем данные

processMsg(msg) ; //2 закоментировать лог не вызывалось в этой же функции

Запись на флешку spi. //серъёзно грузит софт 60~80 %

}

Наложение Патчей на ядро