1. Работа с регистрами
   1. \*\_clk\_on. Регистр устанавливает сколько тактов светодиод остается включенным, после этих тактов включаются остальные регистры.
   2. \*\_clk\_off. Регистр устанавливает сколько тактов светодиод остается выключенным после \*\_clk\_on. После этих тактов включаются остальные регистры.
   3. \*\_wait\_before\_start. Ожидание тактов от старта периода от старта периода экстраполяции.
   4. \*\_line\_order. Необходим чтобы задать последовательность включения светодиодов. Если значение равно, значит они начинают тактирвоаться одновременно согласно значениями в управляющих регистрах.
   5. \*\_chg\_on. [12:0] – 13 битный регистр Этот регистр задает динамическое смещение \*\_clk\_on во время работы. Старший 13 бит регистра задает знак +/- вычитать это значение или прибавлять.
   6. \*\_chg\_off. [12:0] - 13 битный регистр. Этот регистр задает динамическое смещение \*\_clk\_off во время раобты. Старший 13 бит регистра задает знак.
   7. \*\_count\_before\_change. Этот регистр задает ожидание в тактах для работы динамических регистров.

В прошивку на остаточную память fpga защит логический анализатор работы линейки. Чтобы начать работать загрузить stp файл в инструмент логического анализатора и нажать ожидание. И вызвать линукс программу запускающую работу, тогда произойдет захват данных и можно посмотреть как управляется подсветка согласно выбранному режиму в момент экстраполяции и опроса чтения значения с линейки.

1. Виртуальная машина
   1. Компиляция и замена ядра.

cd ~/rel\_socfpga/myremote/

sh ~/intelFPGA\_pro/19.1/embedded/embedded\_command\_shell.sh

export ARCH=arm

export CROSS\_COMPILE=arm-linux-gnueabihf-

export LOADADDR=0x8000

make socfpga\_defconfig – по необходимости

make menuconfig – выбрать необходимые настройки ядра(если нужно)

make –j 6 zImage – 6 количество физ ядер в процессоре ускоряет компиляцию

итоговый zImage переносишь на флешку

если de0nano-soc в сети можно использовать

scp zImage [root@192.168.0.X:~/](about:blank)

mount /dev/mmcpblk0p1 flash

mv zImage flash/zImage

reboot

* 1. Компиляция модуля ядра.

cd ~/custom\_kernel\_module/

dma\_alloc.c – редактируешь

make

дальше перенос таким же способом, как в замене ядра. Только файл с рашсирением \*.ko

1. Запуск тестовых программ

Insmod dma\_alloc.ko

LD\_LIBRARY\_PATH=/usr/arm-linux-gnueabihf/lib/

export LD\_LIBRARY\_PATH

Тестовая программа работает с регистрами

\*(uint32\_t \*) reset\_enable = 1; // перед началом работы надо во всех модулях установить первичные значения

\*(uint32\_t \*) reset\_enable = 0;// и выйти из режима сброса для работы

// необходимо установить нужный режим работы

\*(uint32\_t \*) mode\_cyclic = 1; // режим цикличной заполнения оперативной памяти

\*(uint32\_t \*) state\_cyclic\_index\_in\_buffer; // index in buffer 14 битный

\*(uint32\_t \*) state\_cyclic\_index\_num; // index 28 бит тоже самое что в буфере, только с запасом для обнаружения ошибки

\*(uint32\_t \*) mode\_cyclic = 0; // режим одноразового заполнения по выполнению загорается светодиод на плате

\*(uint32\_t \*) sensor\_on = 0; // включение модуля сенсора

\*(uint32\_t \*) dma\_on = 0; // включение модуля передачи в ram

Остальные по тактированию смотреть пункт 1.

Уровни работы разработанного ПО:

1. Загрузчик

Preloader – предзагрузчик, все файлы для него генерируются из проекта Quartus в qsys-builer, папка для котрого лежит в user\hps\_isw\_handoff\soc\_hps\_0 . Предзагрузчик содержит в себе софт, который подгружает все необходимое для работы загрузчика и в нем настраивается откуда будет загружен загрузчик с раздела флешки с конкретным именем или какой-то области на флешке, область оперативной памяти или флеш памяти. В нашем случае раздел на флешке файл u-boot.img, который собирается после preloader. (см. видео/ Как работать с проектом и ввести разработку пункт 1)

U-boot – загрузчик, использует u-boot.scr (скомпилированный boot.script скрипт) который указывает файл загрузочной прошивки, дерева устройств, включает мосты FPGA-SDRAM, LWHPS-FPGA. Первый мост используется в реализации, как DMA драйвер который собранные данные с датчика пишет, используя этот мост в ddr3. LWHPS-FPGA, используется для контрольных регистров, которые управляют аппаратной реализацией fpga, которая работает с ацп и линейкой (управляющие регистры светодиодов, старта, адреса старта записи DMA, более подробно см. 1 страницу) и platform designer для того чтобы увидеть адресс доступа к регистру, в случае, если он не указан в исходниках пользовательской программы.

1. linux

Socfpga.dtb – файл дерева устройств linux, вслучае если бы в модуле было бы использовано прерывание, по передаче, оно бы обновилось от поумолчанию, в котором нужно было бы описать дополнительно устройства номера прерываний согласно спецификации. В нашем случае, от стандарта были изменены только установочные значения для мостов, чтобы их можно было разблокировать для просмотра их состояния из linux. Сам dtb же собирается там же, где собирается ядро linux, необязательно все усложнять по инструкции с rocketboards при работе с ним. Достаточно через find найти \*cyclone\*.dts выбрать нужны редактируемый (он вложенный и в исходнике написан #include в зависимости от того что добавляется и редактируется), в нашем случае это socfpga\*cyclone\*de\*.dts вроде был, либо просто \*cyclone5\*. И из директории после введенных комманд 1 страницы пункта 2а до make комманды, вводится комманда make socfpga\_cyclone5\_de0\_sockit.dtb . (https://habr.com/ru/company/metrotek/blog/235707/)

zImage – скомплириованное linux ядро (1 страница пункт 2а).

1. fpga

De0\_nano\_soc.rbf – скомпилированный sof файл прошивки средствами Quartus prime. Файл ,который создается из Verilog кода, по сути файл с логикой dma и линейки, который шьется при загрузке u-boot в fpga часть автоматически.

1. Rootfs – debian

Rootfs - . <https://software.intel.com/en-us/node/721452> .

Также был настроен swap файл. Но стоит учитывать, что скорости записи на флешку медленные и если что-то будет попадать в swap то, это будет сказываться на производительности.

Работа с линейкой

1. В случае поиска информации при разработке использовались следующие ресурсы:  
   1. [https://rocketboards.org](https://rocketboards.org/)  
   2. <https://habr.com/ru/users/des333/posts/>  
   3. <https://habr.com/ru/post/264515/>  
   4. <https://habr.com/ru/post/267273/>
2. Спецификации intel от 2018 года по шине Avalon <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf>
3. Cyclone intel handbook – в зависимости от задачи часто приходится сверять информацию, так или иначе.

Как работать с rocketboards:

1. Важно понимать, что все что написано на этом ресурсе, это не всегда актуальное и исчерпывающее руководство, следовательно из всех статей надо выделять суть и делать «просто так» шаг за шагом смысла нет никакого с большой вероятностью ничего не получится.
2. В серии workshop статей, описано почти все от форматирования флешки, как и в статьях у des333 до сборки linux. (В нашем случае с проектом опытном путем был подобран socfpga и он совпал с тем, что используют в статьях на хабре и в сообществе русскоязычном. <https://github.com/altera-opensource/linux-socfpga/branches> ). Соответственно ресурс следует использовать, найдя информацию из workshop (<https://rocketboards.org/foswiki/Documentation/AlteraSoCWorkshopSeries>), актуальную или несколько из <https://rocketboards.org/foswiki/Main/GettingStarted>. По результатам появляющихся ошибок выявить и решить причину самостоятельно.

Как работать с linux:

1. При работе с модулем ядра использовать Documentation в папке с исходниками ядра.
2. Искать готовые решения для переработки в примерах модулей которые билдятся, это наиболее эффективное, содержательное и актуальное руководство к версии исходников с которыми ведется работа. Искать используя комманды find и grep –R.
3. Для осознания того, как работают примеры из исходного кода репозитория linux использовать справочные ресурсы по linux, такие как:
   1. <http://larionov.mytomsk.ru/translate/DMA-API.html>
   2. <https://elixir.bootlin.com/linux/latest/source/Documentation/DMA-API-HOWTO.txt>
   3. <https://www.xml.com/ldd/chapter/book/ch13.html>

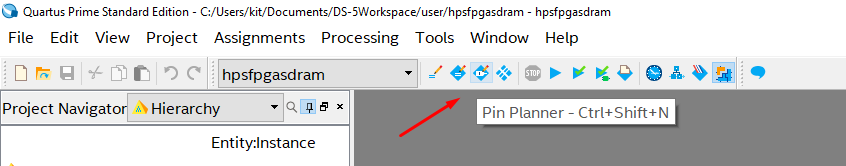
Как работать с проектом и вести разработку:

1. Общий step-by-step <https://rocketboards.org/foswiki/Documentation/EmbeddedLinuxBeginnerSGuide>

<https://rocketboards.org/foswiki/Documentation/AVCVGSRDLTS>

(просто как справку).

1. Для общего понимания, если это первый опыт, стоит на примере светодиодов, в ускоренном режиме просто посмотреть. <https://www.youtube.com/watch?time_continue=10&v=zJ6PgFIw_b8> . Все что там делается описано в <https://bitlog.it/hardware/building-embedded-linux-for-the-terasic-de10-nano-and-other-cyclone-v-soc-fpgas/> . <https://gist.github.com/addisonElliott/8c229e47184a724d1a00328110dc094c> . Отсюда была взята работа с lw2hps на основе которых были построены все управляющие и контрольные регистры.
2. <https://www.terasic.com.tw/attachment/archive/941/DE0-Nano-SoC_User_manual_rev.C1.pdf> - расположение pinout. В quartus настраивается из Pin Planner



Пины, которые назначаются указываются в модуле самого верхнего уровня. После синтеза в pinplanner их можно найти и настроить на нужную ногу.

1. Туториал по настройке логического анализатора: <https://marsohod.org/11-blog/213-signaltap>
2. Прошивка создается в формате rbf-x16. Прошивать её удобно следующим образом:
   1. Создаем прошивку либо из интерфейса file-convert, generate, все как в видео
   2. scp firmware.rbf [root@192.168.0.X:~/](about:blank)
   3. mount /dev/mmcpblk0p1 flash
   4. mv firmware.rbf flash/de0\_nano\_soc.rbf # принципиально название de0\_nano\_soc.rbf или перепишите u-boot.scr на основе переданного boot.script и скомпилируйте и замените его.
   5. reboot

Таким образом, файл скомпилированноый на виртуалке передается по сети в плату и после перезагрузки все автоматом подхватывается, операций по передергиванию проводов и флешек нету.

Описание работы разработанного продукта:

При загрузке модуля ядра, модуль ядра выделяет память для DMA (при необходимости увеличить этот объем надо пересобрать ядро linux с параметрами расширения CMA (coherent memory allocation) надо найти этот параметр после команды make menuconfig по инструкции как при сборке ядра. Выделив память и заполнив её инкрементирующимся значением (для проверки по умолчанию он заполняет выделенную область). Модуль считается загруженным. Дальше программа из пользовательского пространства открывает модуль ядра, как файл и использует два механизма для работы с ним. Первый механизм построен на ioctl (<https://www.opennet.ru/docs/RUS/lkmpg/node18.html>). Этот механизм, используется для передачи стартового физического адреса выделенной области памяти, который в дальнейшем передается в контрольный регистр для FPGA DMA. Второй механизм mmap из пользовательского пространства можно просматривать для целей отладки записанные значения. В дальнейшем пользовательская программа в зависимости от тех значений которые она устанавливает в контрольно-статусные регистры управляет работой линейки.

Пароль от виртуалки: 1234