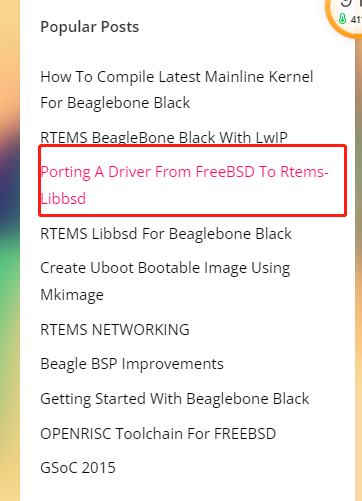
# Porting a driver from FreeBSD to rtems-libbsd

<http://ragustechblog.blogspot.com/2015/06/porting-driver-from-freebsd-to-rtems.html>



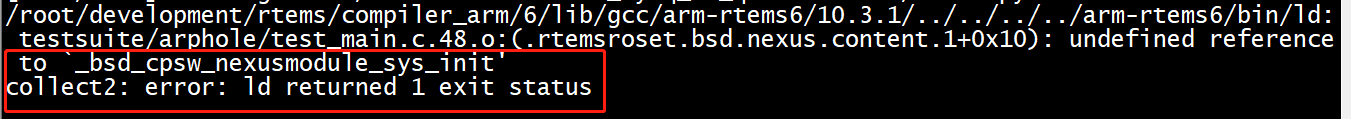
in this post I will explain the necessary steps needed to port a ethernet driver from FreeBSD to rtems-libbsd.  There are few changes we need to make to be able to compile the driver.  
  
I will be using cpsw ethernet driver for BBB as reference.  Follow the below steps  
Whenever we edit a header file or c file, we need to add the changes under #if defined  \_\_rtems\_\_.  
The code on FreeBSD needs to go under the #else part.

# STEPS

Step1  
1. Copy the driver files from FreeBSD to appropriate destination in rtems-libbsd  
     I copied it to "freebsd/sys/arm/ti/cpsw"

**step2**

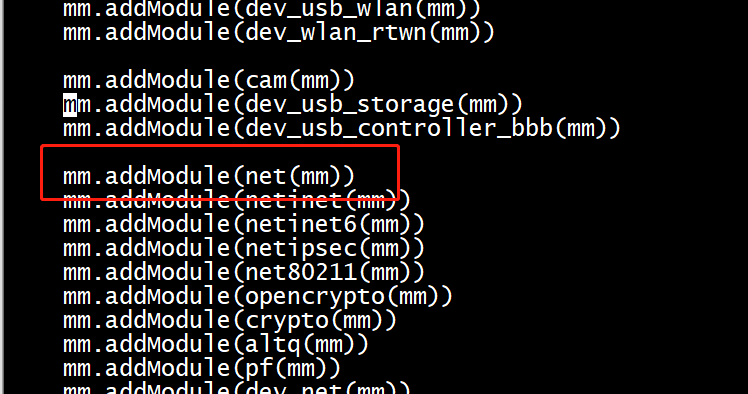
2.  Include the added files in Makefile  
     "LIB\_C\_FILES += freebsd/sys/arm/ti/cpsw/if\_cpsw.c"

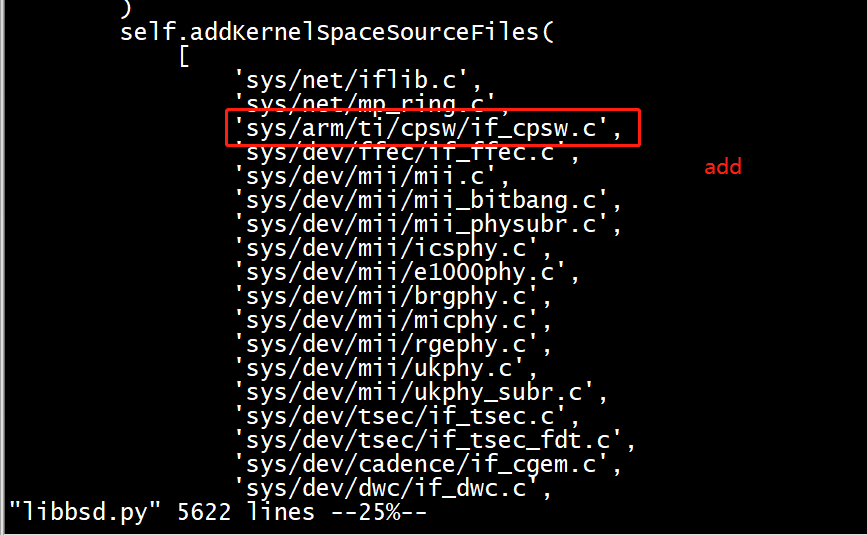
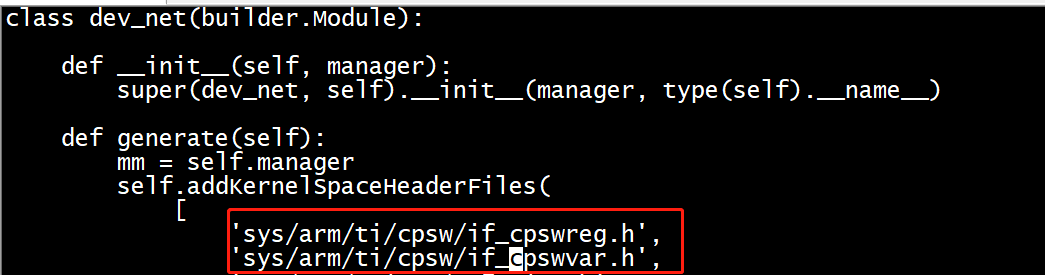


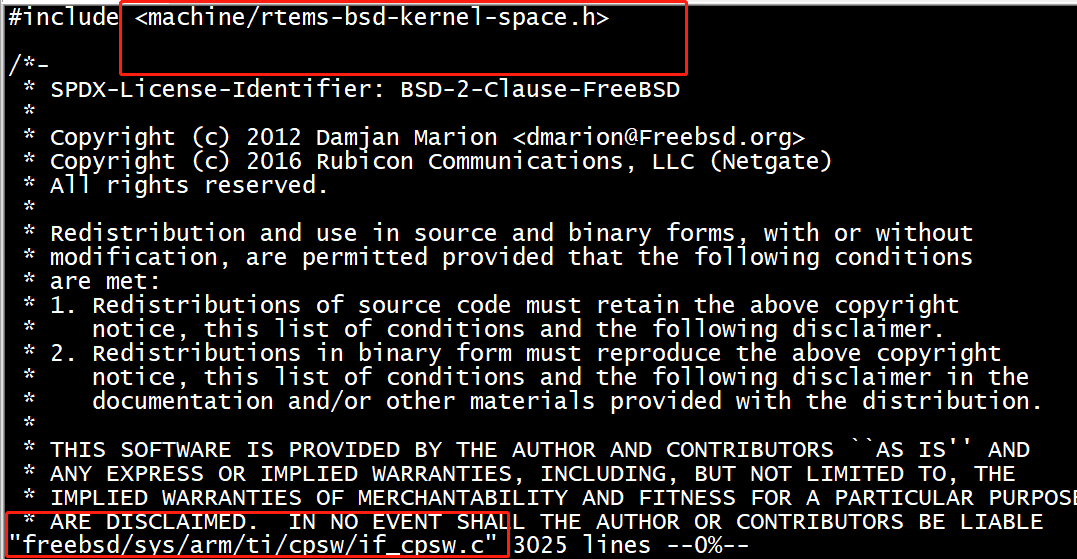
### Waf & libbsd.py

**参考[RTEMS USB support for BBB](https://blog.css8.cn/post/3577340.html)**

If adopt waf, you sould do



3.  Add the following header file for any c file you are including  
     "#include <machine/rtems-bsd-kernel-space.h>"  
  
4. At this stage, your driver is included for compilation. start compilation  
    "make clean && make"  
  
5. You will get several errors. You will need to resolve the errors one by one. The errors that come  
     are dependent on the driver. I will explain some of the common errors.

### Common Issues

     a.   Sysctl

          The FreeBSD driver usually add sysctls to get stats and adjust variables.  RTEMS do not  
          support these. We need to remove all functions and variables related to sysctl handling.

     b.  OFW/FDT

          Open Firmware(OFW)/ Flattened Device Tree (FDT) dependency. FreeBSD supports FDT  
          involves supplying device tree information during boot time using a special configuration file.  
          RTEMS do not have this support yet, we need to remove those dependencies. Commonly they  
          are seen in probe and attach functions  
  
     c. In attach functions, we may get the phy address from FDT. This phy address is used during  
         mii\_attach. We can give MII\_PHY\_ANY in mii\_attach. It will loop through all the possible  
         addresses and will find the correct phy.  
  
     4. There may some other dependencies on other drivers or some other modules which is defined  
          FreeBSD but not needed in RTEMS. You can define equivalent functions as per the  
          requirement. I defined my own functions to read the mac address.

### Simplebus vs nexus

         RTEMS has a well defined nexus bus support. It will be the one that will be attaching to root and other drivers will be attaching to this bus. Simplebus interface is not present in RTEMS. Simplebus is usually the bus related to OFW interface. So when we have to port a driver based on simplebus we have two options

1. Port simplebus interface to rtems-libbsd

2. Modify the driver to nexus bus

I took the option to modify the bus to use the nexus bus.  For this you need to make the following changes

1. In the DRIVER\_MODULE definition, we use nexus instead of simplebus

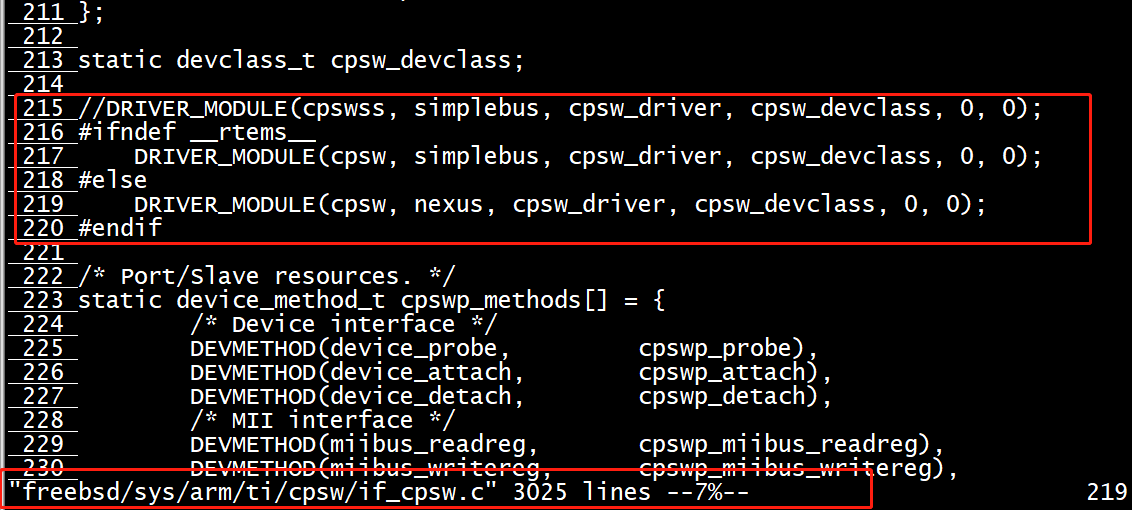
    #ifndef \_\_rtems\_\_

    DRIVER\_MODULE(cpsw, simplebus, cpsw\_driver, cpsw\_devclass, 0, 0);

    #else

    DRIVER\_MODULE(cpsw, nexus, cpsw\_driver, cpsw\_devclass, 0, 0);

    #endif



2. Next thing we need to define the resource structure which will inform the driver the base address

     to be used for bus handles and irq to be used for the device. The resource should have the

     following details the memory map address and the  irq and their numbers to be used by the

     driver. Cpsw driver starts at 0x4a100000 and have 4 irq's starting at 40.

     In the file [rtemsbsd/include/bsp/nexus-devices.h](https://github.com/ragunath3252/rtems-libbsd/commit/6b1aa217580f5b3162b60ecfdc86e75251ba5283" \l "diff-5241dafbec607e4b3db0d47fd8ae4148)

    #elif defined(LIBBSP\_ARM\_BEAGLE\_BSP\_H)

     static const rtems\_bsd\_device\_resource cpsw0\_res[] = {

        {

                .type = RTEMS\_BSD\_RES\_MEMORY,

                .start\_request = 0,

                .start\_actual = 0x4a100000 /\*Memory map address of the ethernet\*/

        }, {

                .type = RTEMS\_BSD\_RES\_IRQ,

                .start\_request = 0,

                .start\_actual = 0x28

        },

           {

                .type = RTEMS\_BSD\_RES\_IRQ,

                .start\_request = 1,

                .start\_actual = 0x29

        },

           {

                .type = RTEMS\_BSD\_RES\_IRQ,

                .start\_request = 2,

                .start\_actual = 0x2a

        },

           {

                .type = RTEMS\_BSD\_RES\_IRQ,

                .start\_request = 3,

                .start\_actual = 0x2b

        }

};

3. Define the following macro below the resource structure

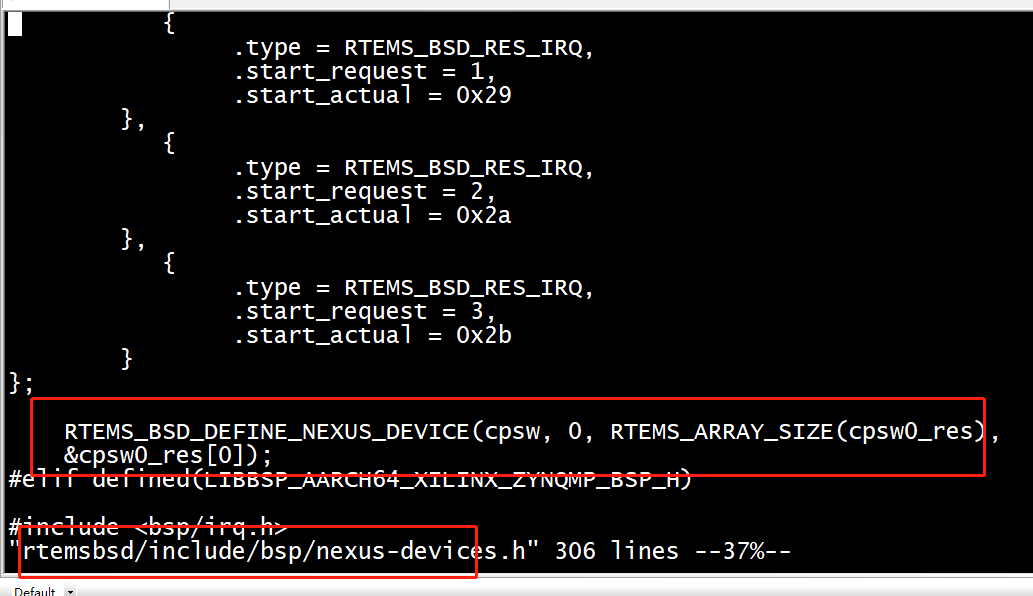
     RTEMS\_BSD\_DEFINE\_NEXUS\_DEVICE(cpsw, 0, RTEMS\_ARRAY\_SIZE(cpsw0\_res),

    &cpsw0\_res[0]);

     This is needed because, we use SYSINIT feature of FreeBSD. With this definition, we will be

      automatically loaded during boot time. For more details see libbsd.txt file.

With these changes we moved to nexus bus.



4. You may need to port the phy driver also from FreeBSD. For me I needed smscphy. So I added smscphy.c to rtems-libbsd and included it in compilation.

5. We need to make sure  the phy device get attached to miibus. for this we define

     SYSINIT\_DRIVER\_REFERENCE(smscphy, miibus) in the same nexus device file.

With this we should be able to form a proper device tree.

### LOG

Below is the log I got. Ignore the Failed to read PHY messages. Since we gave MII\_PHY\_ANY during attach, it tries different address so we getting failed messages

## Transferring control to RTEMS (at address 80000000) ...

RTEMS Beagleboard: am335x-based

\*\*\* LIBBSD INIT 1 TEST \*\*\*

RTEMS Shell on /dev/console. Use 'help' to list commands.

[/] # nexus0: <RTEMS Nexus device>

cpsw0: <3-port Switch Ethernet Subsystem> on nexus0

00:00:00 cpsw\_attach

cpsw0: device parent is present

cpsw0: CPSW SS Version 1.12 (0)

cpsw0: Initial queue size TX=128 RX=384

cpsw0: MAC HI ce04a578

cpsw0: MAC Low f531

devctl: !system=IFNET subsystem=cpsw0 type=ATTACH

cpsw0: Ethernet address: -2144597976

cpsw0: Failed to read from PHY.

cpsw0: Failed to read from PHY.

cpsw0: Failed to read from PHY.

cpsw0: Failed to read from PHY.

cpsw0: Failed to read from PHY.

cpsw0: Failed to read from PHY.

miibus0: <MII bus> on cpsw0

smscphy0: <SMC LAN8710A 10/100 interface> PHY 0 on miibus0

smscphy0:  10baseT, 10baseT-FDX, 100baseTX, 100baseTX-FDX, auto

devctl: +smscphy0 at phyno=0 oui=0x800f model=0xf rev=0x1 on miibus0

devctl: +miibus0 at   on cpsw0

devctl: +cpsw0 at   on nexus0

devctl: +nexus0 at   on root0

devctl: !system=IFNET subsystem=lo0 type=ATTACH

IRQ defined can be seen using irq command

[/] #  irq

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

                             INTERRUPT INFORMATION

--------+----------------------------------+---------+------------+------------

 VECTOR | INFO                             | OPTIONS | HANDLER    | ARGUMENT

--------+----------------------------------+---------+------------+------------

     40 | cpsw0                            |  UNIQUE | 0x800d8788 | 0x802c5428

     41 | cpsw0                            |  UNIQUE | 0x800d8788 | 0x802c5460

     43 | cpsw0                            |  UNIQUE | 0x800d8788 | 0x802c5498

     67 | Clock                             |  UNIQUE | 0x800d7684 | 0x00000000

     72 | NS16550                       |  SHARED | 0x800d945c | 0x00000000

--------+----------------------------------+---------+------------+------------