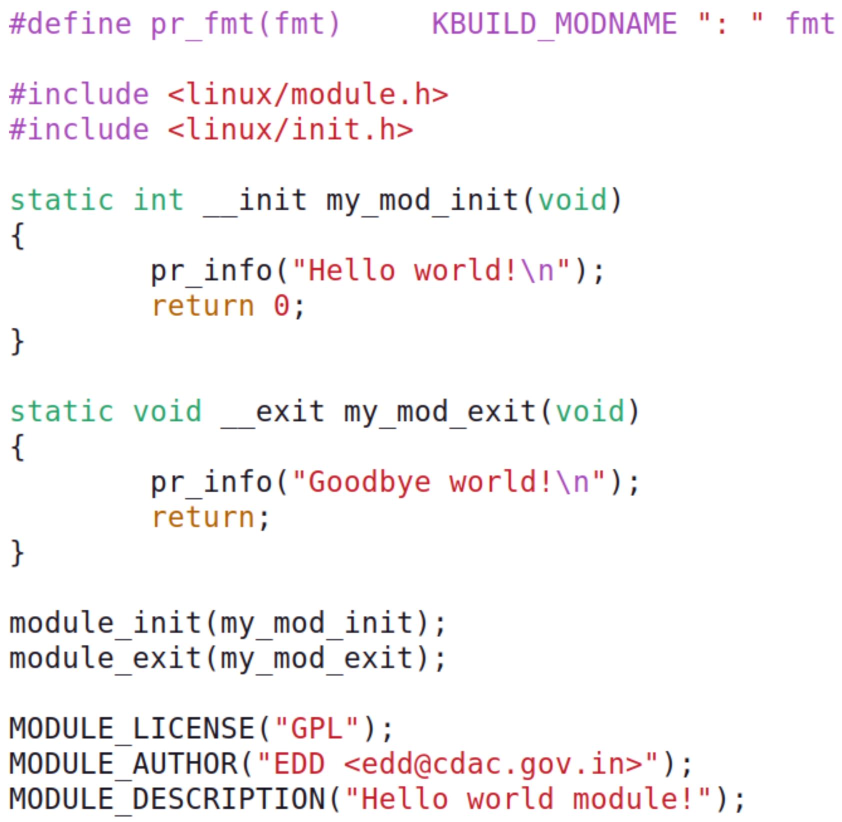
EMBEDDED DEVICE DRIVERS

Linux Device Drivers on Beaglebone Black

Kernel module: Basics

- Kernel module skeleton



```
#define pr_fmt(fmt)     KBUILD_MODNAME ": " fmt

#include <linux/module.h>
#include <linux/init.h>

static int __init my_mod_init(void)
{
        pr_info("Hello world!\n");
        return 0;
}

static void __exit my_mod_exit(void)
{
        pr_info("Goodbye world!\n");
        return;
}

module_init(my_mod_init);
module_exit(my_mod_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("EDD <edd@cdac.gov.in>");
MODULE_DESCRIPTION("Hello world module!");
```

- Header files
- Init / Exit concept
- init/exit macros
  - __init, __exit
  - module_init()
  - module_exit()
- Metadata macros
  - MODULE_LICENSE
  - MODULE_AUTHOR
  - MODULE_DESCRIPTION

Kernel module: Headers

- Note that kernel cannot use standard C libs
  - Since it is an independent piece of software
  - Same goes for all its modules

- It has its own headers
  - Come from <linux/…>
  - In the kernel source tree

- All modules need

  <linux/module.h>
  <linux/init.h>

- Also note ***pr_info()*** – the kernel's version of "printf"
  - No floating point support

Kernel module: init/exit macros

- Kernel modules are not sequential code
    - Most modules
        - Initialize some resource / hardware
        - Handle requests to deal with that resource
        - Have to clean up if the resource is not needed any more
    - Often, the module "registers" itself with the kernel
        - Saying that "I will handle this resource from now on"

- This leads to the concept of
    - Init
        - This function is called when the module is "loaded"
        - __init and module_init() tell the kernel about this function

        #define __init   __section(.init.text)

    - Exit
        - This function is called when the module is "unloaded"
        - __exit and module_exit() tell the kernel about this function

        #define __exit   __section(.exit.text)

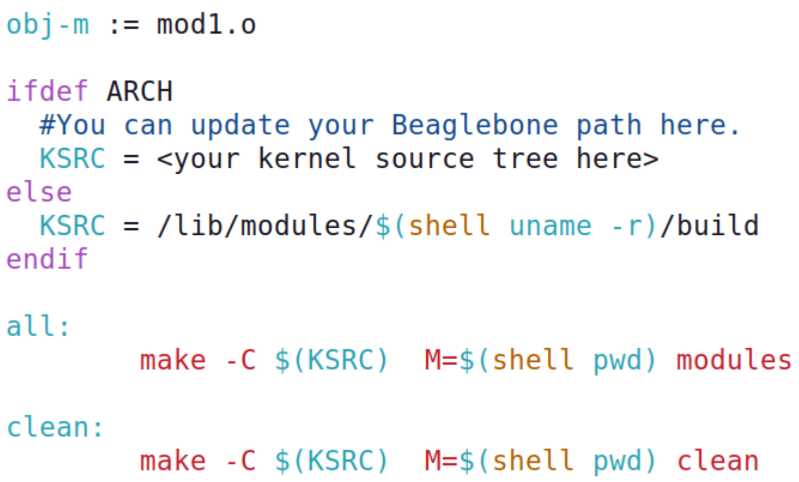
Kernel module: Metadata

- Kernel module code usually contains metadata
  - *MODULE_LICENSE*
    - Some options: “GPL”, “GPL v2”, “Dual BSD/GPL”, “Proprietary”
    - Any license which is proprietary “taints” the kernel
  - *MODULE_AUTHOR*
    - Name (and email) of the module author – for support/reference
  - *MODULE_DESCRIPTION*
    - A one-liner telling the world what this module does

  - *MODULE_VERSION*
  - *MODULE_ALIAS*
  - *MODULE_DEVICE_TABLE*

Kernel module: Compilation

- Makefile for module compilation



```
obj-m := mod1.o

ifdef ARCH
  #You can update your Beaglebone path here.
  KSRC = <your kernel source tree here>
else
  KSRC = /lib/modules/$(shell uname -r)/build
endif

all:
        make -C $(KSRC)  M=$(shell pwd) modules

clean:
        make -C $(KSRC)  M=$(shell pwd) clean
```

- obj-m
  - Compile as a kernel module
- ARCH
  - Read the ARCH from the command line
  - Else assume its host!
- KSRC
  - Location of the kernel source tree
- make command
  - *-C $(KSRC)*
    - Enter the KSRC directory
  - *M=$()*
    - Use the present directory to compile the module

Kernel module: *obj-<X>*

- Options for the ***obj-<X>*** variable:
  - *obj-m*
    - Compile as a loadable kernel module
  - *obj-y*
    - Compile as a built-in part of the kernel
  - *obj-n*
    - Exclude from the build process (don’t compile)

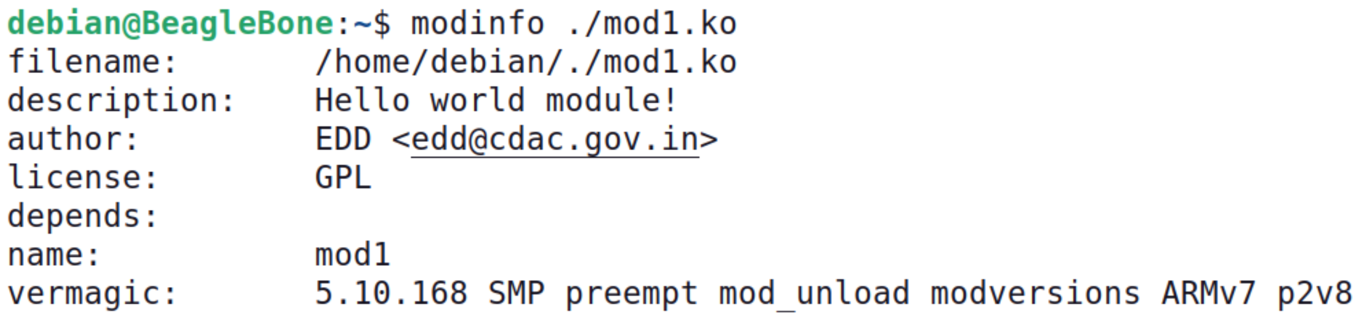
  - General usage in kernel tree:

    obj-$(CONFIG_MY_MODULE) := …

Kernel module: The .ko file

- Refer the ***mod1*** directory – ***mod1.c***
- The output from the kernel module compile
  - *mod1.ko*
    - Run ***file*** on this to see what type of file it is
    - Also

      $ ***modinfo*** *./mod1.ko*
    - Ensure the .ko file is for the ARM (BBB)!



```
debian@BeagleBone:~$ modinfo ./mod1.ko
filename:       /home/debian/./mod1.ko
description:    Hello world module!
author:         EDD <edd@cdac.gov.in>
license:        GPL
depends:
name:           mod1
vermagic:       5.10.168 SMP preempt mod_unload modversions ARMv7 p2v8
```

Kernel module: Load / Unload

- Transfer the .ko file to BBB using ssh

- Load the module into the running kernel

  $ sudo insmod ./mod1.ko

- Watch the output of dmesg / serial output

- Confirm the module is loaded

  $ lsmod | grep mod1

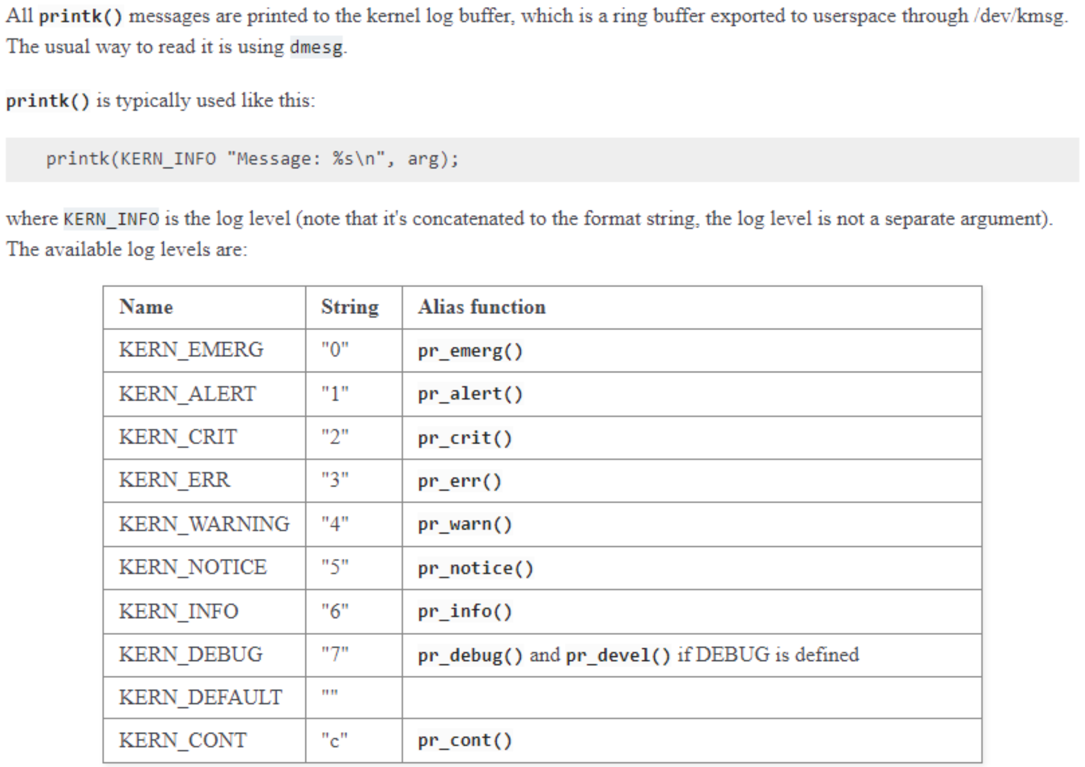
- Unload the module from the kernel

  $ sudo rmmod mod1

- Again, watch the output of dmesg / serial output

Kernel module: *printk / pr_**

- Earlier versions used *printk*; now we use *pr_**



All **printk()** messages are printed to the kernel log buffer, which is a ring buffer exported to userspace through /dev/kmsg. The usual way to read it is using `dmesg`.

printk() is typically used like this:

```
    printk(KERN_INFO "Message: %s\n", arg);
```

where `KERN_INFO` is the log level (note that it's concatenated to the format string, the log level is not a separate argument). The available log levels are:

Name	String	Alias function
KERN_EMERG	"0"	**pr_emerg()**
KERN_ALERT	"1"	**pr_alert()**
KERN_CRIT	"2"	**pr_crit()**
KERN_ERR	"3"	**pr_err()**
KERN_WARNING	"4"	**pr_warn()**
KERN_NOTICE	"5"	**pr_notice()**
KERN_INFO	"6"	**pr_info()**
KERN_DEBUG	"7"	**pr_debug()** and **pr_devel()** if DEBUG is defined
KERN_DEFAULT	""	
KERN_CONT	"c"	**pr_cont()**

A word about the pr_fmt macro

- Kernels have lots of modules
  - All modules write to kernel buffer
    - Read by dmesg
  - Becomes confusing to search
    - For ‘our’ module’s output

- What if we prefix our prints with our module name?
  - This is via the ***pr_fmt()*** macro

    #define pr_fmt(fmt)   KBUILD_MODNAME “: ” fmt

  - This prefixes all our module’s prints
    - With our module’s name (mod1)

      mod1: Hello world!

Kernel module: Loadability

- Note that the ability of loading/unloading modules
  - Can be set up when the kernel is compiled
  - For module load support
    - *CONFIG_MODULES=y*
  - For module unload support
    - *CONFIG_MODULE_UNLOAD=y*
  - For forcibly unloading modules (even when in use)
    - *MODULE_FORCE_UNLOAD=y*

- All these would be set up during

  $ make menuconfig

THANK YOU!