Embitude

Let's have fun with Embedded

I2C Training Exercise

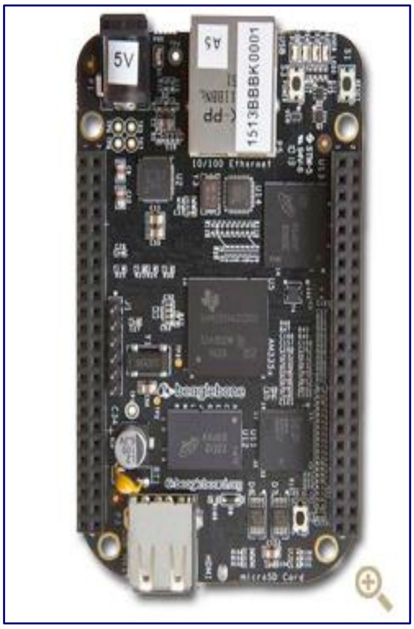
Introduction

The workshop contents (code, docs, ...) and more are available from the I2CD GIT Repo. You may clone the repo using " https://github.com/embitude/i2cd". More details on the contents can be obtained from its README.

You may follow the following steps on your Linux System:

```
$ cd
$ mkdir I2CD
$ cd I2CD
$ git clone https://github.com/embitude/i2cd
```

Click here to browse material.



Beagle Bone Black as shown above is the hardware used for the workshop. More details about it can be obtained from its Product Page.

Default Bootup Setup (One time)

Connect USB2TTL to BBB and System (as per I2CD/Docs/BBB_SRM.pdf pg 90) - Do not power BBB

If VM, switch USB2TTL to VM
On Linux system (install the minicom package if not already there):

```
sudo minicom -s and do the setup for baud 115200 bits 8n1 hw & sw flow control off
sudo minicom -o
```

Power on BBB to boot into Linux
Login into BBB as root

```
uname -r # Verify your original kernel version
df -h /boot/uboot | tail -1 | awk '{print $6}' # Determine / or /boot/uboot
poweroff
```

Further on Linux system (inside the I2CD folder):

```
$ git clone https://github.com/embitude/bbb-builds
$ cd bbb-builds
```

Steps to setup the toolchain (from the bbb-builds folder):

```
$ make install_toolchain
```

Then, logout and login back for its PATH activation.

Steps to install additional libraries (from the bbb-builds folder):

```
$ make install_libs
```

Embitude specific Setup (One time)

On Linux system (inside the bbb-builds folder):

```
$ make generate_prepare_usd
$ cd Utils
Connect uSD w/ Linux System
$ ./prepare_usd [-d] <usd_device_file> # -d for raw dump of MLO & u-boot.img
```

Insert uSD into BBB
Boot BBB w/ uSD by optionally keeping the lone black button pressed, while powering on
Login into BBB as root
Check for "4.19" or latest kernel using uname -r

References

https://www.i2c-bus.org/

Session 1 Assignments

Assignment 1: Configure & Build the Kernel for I2C Training

1. Navigate to the OS directory of bbb-builds

   ```
   $ cd bbb-builds/OS
   ```

2. Get the kernel source code

   ```
   $ wget https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/linux-4.19.103.tar.gz
   ```

3. Unpack the kernel source code

   ```
   $ tar -xvf linux-4.19.103.tar.gz
   ```

4. Get into the kernel source code

   ```
   $ cd linux-4.19.103/
   ```

5. Apply the patches

   ```
   $ patch -p1 < ../Patches/0001-Set-up-the-pin-mux-for-button-S2.patch
   $ patch -p1 < <i2cd_repo_path>/Patches/i2c_embitude.patch
   ```

6. Configure the kernel with already available configuration file for Beaglebone Black

   ```
   $ cp <i2cd_repo_path>/Configs/config.4.19.103.i2cd .config
   ```

7. Update the Kernel Makefile to cross compile for arm architecture
   Add following in Kernel Makefile (Search for ARCH and update as below)
   - CROSS_COMPILE=arm-linux-gnueabihf-
   - ARCH=arm
8. Finally, compile the kernel & dtb

   ```
   $ make zImage
   $ make dtbs
   ```

9. Transfer the newly build kernel & dtb to the target board

   ```
   $ mount /dev/mmcblk0p1 /mnt (On board)
   $ scp arch/arm/boot/zImage root@<board_ip>:/mnt/ (On the system)
   $ scp arch/arm/boot/dts/am335x-boneblack.dtb root@<board_ip>:/mnt/ (On the system)
   ```

10. Unmount & Reboot the board to boot up with updated kernel

    ```
    $ umount /mnt (On board)
    $ reboot (On board)
    ```

11. Verify if the kernel is updated

    ```
    $ uname -a (Should show the latest kernel build time)
    ```

Assignment 2: Building the Linux I2C framework independent driver

1. Complete all the TODOs in the low_level_driver.c & i2c_char.c
2. Compile & transfer i2c.ko to the target platform
   $ insmod i2c.ko (This should create the corresponding device file)
   $ cat & echo on the device file should invoke the i2c_receive() & i2c_transmit() functions respectively
3. $ Share the screenshot for the output & also the diff w.r.t original code

Session-2 Assignments (I2C Driver Initialization)

Assignment 1: I2C Module Initialization

1. Complete all the todos related to initialization in P02_i2c_init/low_level_driver.c
2. Share the diff w.r.t original low_level_driver.c (Use git diff low_level_driver.c to get the diff)
3. Integrate the Session-1 Assignment-2 changes over the low_level_driver.c.
4. Compile and code & generate i2c.ko

Assignment 2: Sending a bytes over the I2C bus

1. Complete all the todos in i2c_transmit() of P02_i2c_init/low_level_driver.c
2. Share the diff w.r.t to the original
3. Compile the code and transfer it to the board.
4. $ insmod i2c.ko
5. echo 1 > /dev/i2c_drv0 (Should get XRDY and ARDY event)
6. Share the screenshot of the output.

Session-3 Assignments (I2C Transactions & Accessing Eeprom)

Assignment 1: Sending multiple bytes over the I2C bus

1. Enhance the low level driver in Session 2, assignment #2 to transmit multiple bytes on the I2C bus.
2. Share the code snippet for the i2c_transmit() function and screenshot of the o/p.

Assignment 2: Receiving the data from Eeprom

1. Complete all the todos in i2c_receive() of P03_i2c_txrx/lower_level_driver.c
2. Modify the i2c_transmit() from above assignment to send only 2 bytes, representing the address of the eeprom location to be read.
3. Modify my_read() in i2c_char.c to invoke i2c_transmit() followed by i2c_receive.
4. Share the screenshot of o/p and i2c_receive() function.

PS To cross-verify the output, follow the below instructions:

Verifying the Eeprom Contents

1. cd <Kernel Source Path> (linux-4.19.103)
2. $ make menuconfig
   Device Drivers > I2C support > I2C Hardware Bus support
   Select OMAP I2C adapter as M
   Exit menuconfig
3. $ make drivers/i2c/busses/i2c-omap.ko
4. $ scp drivers/i2c/busses/i2c-omap.ko root@192.168.7.2:
   Execute the below commands on BBB
5. $ insmod i2c-omap.ko (make sure that custom driver i2c.ko is removed)
6. $ cat /sys/bus/i2c/drivers/at24/0-0050/eeprom | hexdump -C (reads the eeprom contents)
7. $ cat /sys/bus/i2c/drivers/at24/0-0050/eeprom > eeprom.bin (Redirects the eeprom contents to the file)
8. $ cat eeprom.bin > /sys/bus/i2c/drivers/at24/0-0050/eeprom (Updating the eeprom)

Session-4 Assignments (Device Model)

References

https://www.kernel.org/doc/html/latest/driver-api/driver-model/platform.html (Device Model)
https://www.kernel.org/doc/html/latest/driver-api/driver-model/index.html (Device Model)
https://lwn.net/Articles/448499/ (Platform Driver)
https://lwn.net/Articles/448502/ (Platform devices and Device tree)
https://elinux.org/Device_Tree_Usage (Device Tree)

Assignment 1: Enable the platform driver support

1. Complete all the todos in low_level_driver.c & low_level_device.c under P04_device_model. Refer platform_driver.c & platform_device.c
2. Compile and transfer i2c.ko & low_level_device.ko
3. When both modules are loaded, the probe for low_level_driver should get invoke. Also, the device file should get created and if any of the driver/device is removed, the device file should get deleted
4. cat /dev/i2c_drv0 should read the contents of the eeprom
5. Share the screenshot for the o/p and also the diff for the code

Assignment 2: Enable the dtb support in Linux

Part(i) – Update DTB

1. Modify am335x_boneblack.dts under kernel_source_path/linux-4.19.103/arch/arm/boot/dts/ to add the device node under '/' (root node). Add the reg & clock-frequency properties. Refer i2c0 node in kernel_source_path/linux-4.19.103/arch/arm/boot/dts/am33xx.dtsi.
2. cd kernel_source_path/linux-4.19.10/
3. make dtbs
4. mount_boot (On target)
5. scp arch/arm/boot/dts/am335x-boneblack.dtb root@192.168.7.2:/boot/
6. reboot (On target)

Part(ii) – Update Driver

1. Complete all the todos in P04_device_model/low_level_driver_dtb.c. Refer gpio_dtb.c
2. Compile and transfer i2c_dtb.ko to the board. On insmod the probe should get invoke and device file should get created under /dev/
3. cat /dev/i2c_drv0 should be access the eeprom
4. Share the screenshot for the o/p and also the code changes.

Session-5 Assignments (I2C Subsystem)

Assignment 1: Dummy Client & Adapter Driver

Part(i) – Update DTB

1. Add the node for i2c-dummy in kernel_source/linux-4.19.103/arch/arm/boot/dts/am335x-boneblack.dts (Refer slides for node details)
2. cd kernel_source/linux-4.19.103
3. make dtbs
4. mount_boot (on board)
5. scp arch/arm/boot/dts/am335x-boneblack.dtb root@192.168.7.2:/boot/

6. reboot (on board)

Part(ii) – Update Driver

1. Complete all the todos in dummy_client.c and dummy_adap.c under P05_i2c_subsystem (Refer slides for the APIs to register the adapter and client)
2. Transfer the dummy_client.ko & dummy_adap.ko to the board.
3. The probe for dummy_client.ko should get invoked once both the drivers are present
4. cat /dev/dmy_i2c0 should invoke the dummy_i2c_xfer in dummy_adap and print the buffer contents sent from the client driver
5. echo 1 > /dev/dum_i2c0 should invoke dummy_i2c_xfer of dummy_adap. The buffer contents should be printed in client driver

Session-6 Assignments (I2C Integration)

Assignment 1: Optimize the controller driver & remove the hardcoding in controller driver

1. Complete all the todos in i2c_char.c & low_level_driver.c under P06_i2c_integration
2. Make sure there is a corresponding node for the platform driver in the dtb
3. Compile & transfer the file i2c.ko to the board.
4. Perform the read/write operations as usual and share the share the screenshot.
5. Share the diff w.r.t original

Assignment 2: Integrate the controller & client driver with the I2C Framework

1. Complete all the todos in i2c_adap.c & i2c_client.c under P06_i2c_integration
2. Make sure there is a corresponding node for the controller & client driver in the dtb
3. Compile & transfer i2c_adap.ko and i2c_client.ko.
4. The probe should get invoke when both the drivers are in place.
5. Share the screenshot of the o/p & diff w.r.t original.

Session-7 Assignments (Interrupts)

Assignment 1: Enabling the Interrupt

Part(i) – Update DTB

1. Add the interrupts property in i2c controller node at arch/arm/boot/dts/am335x-boneblack.dts
2. $ make dtbs
3. mount_boot (on board)
4. scp arch/arm/boot/dts/am335x-boneblack.dtb root@192.168.7.2:/boot/
5. reboot (on board)

Part(ii) – Update the Controller Driver

1. Complete all the todos in P07_i2c_interrupts/i2c_adap.c
2. Compile and transfer the file on the board.
3. The irq should get registered upon insmod.
4. Modify i2c_client.c, so that draining comes into picture.
5. share the o/p and diff w.r.t to original code.