Embitude

Let's have fun with Embedded

I2C Training Assignments

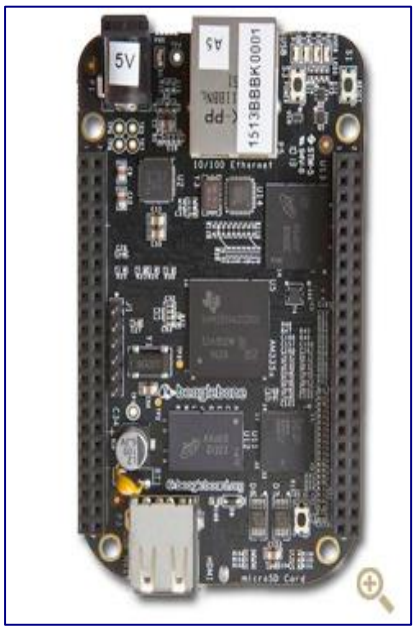
Introduction

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)