Bluetooth disk-on-key: Final project for the Advanced computer system class TAU winter 2016/2017

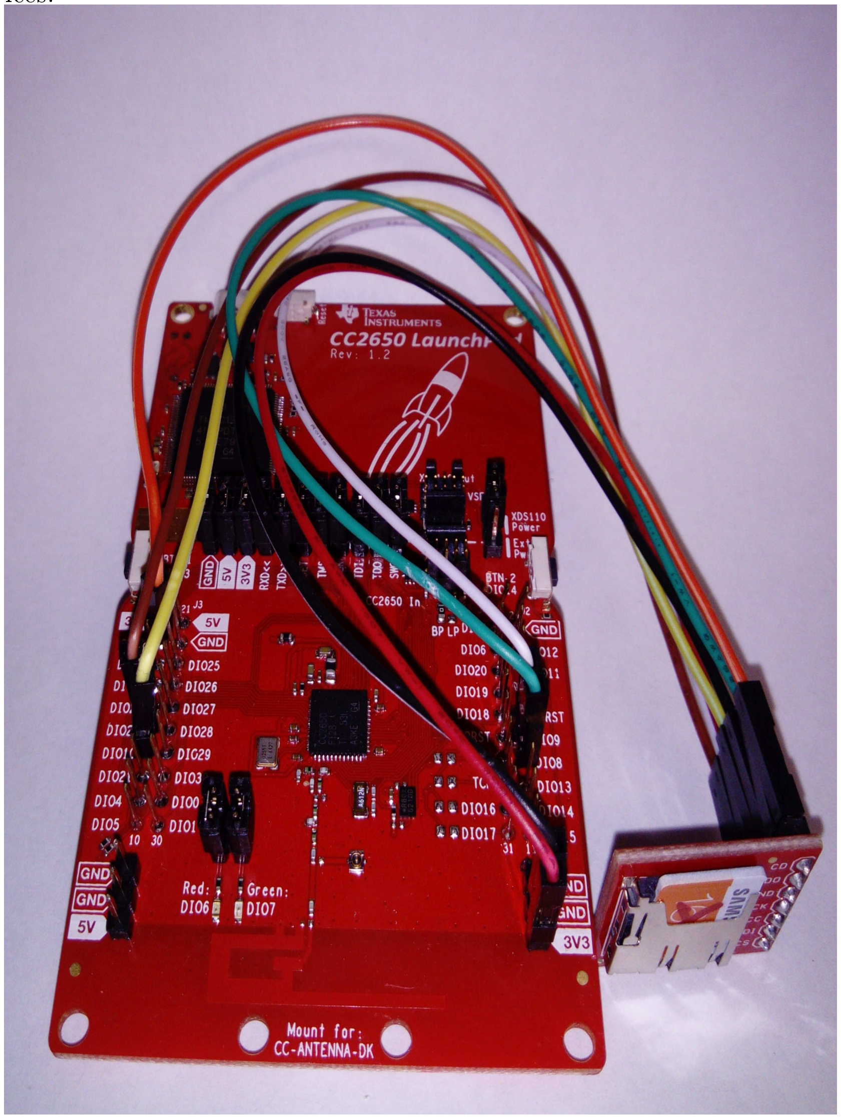
Lev Pachmanov
Noam Nissan
Supervisor: Prof. Sivan Toledo

March 11, 2017

Contents

1 Introduction

As a project in Advance computer system class in IoT and embedded systems we have decided to create a proof of concept of a disk on key that communicates over Bluetooth. The general idea is a disk on key that can stay in your pocket while connecting it to a different PC or even your smart phone. The reason for choosing Bluetooth as the wireless interface is the ease of pairing it with other devices.



Project guidelines addressing

1. Use hardware and operating system used in class - we use the TI CC2650 launchpad and the TI-RTOS operating system
2. Software development challenge and hardware interfacing - We plug an SD card reader and interact with it via SPI, we also implement the protocol of NBD3.3 on the board.
3. Real need or Real users - A wireless mobile storage can be very use full.

2 Implementation Overview

The system is made of two parts, the DOK side and the user side. The DOK side is made of the TI CC2650 launchpad connected to a microSD card reader which is used for the storage. The launchpad was programmed with both SD card interface and Bluetooth interface for our needs. For the user side we used a Linux laptop with integrated/external Bluetooth chip. To be able to support any file system desired we decided the communication between the PC and the board will be on the block device layer. We use a library called NBD[4] (Network Block Device) which creates a Linux block device that its implementation is over the network, i.e translates block reads and writes into network communication. Finally we create another component on the PC that transforms network traffic coming from the NBD into Bluetooth traffic with the ti board and the program on the board in turn takes the NBD protocol and transform it into reads and write with the SD card.

3 Components Overview

In this part we discuss every component of the system in terms of structure and functionality.

3.1 MicroSD slot

We used the microSD Transflash Breakout made by Sparkfun[1].

SPI The Interface is connected to the launchpad with simple wires in the following manner (see picture above)

- CD - DIO23
- DATA OUT - DIO8
- GND - VDD
- SCK(clock) - DIO10
- VCC - VCC(3.3v)

- DATA IN - DIO9
- CHIP SELECT - DIO24

We could make the SPI interface faster by using 4 data wires instead of two but we didn't had one and this was not the bottleneck anyway.

SD We used an SD card that supports physical layer protocol v2.0[3]. We follow the protocol for transitioning the card to SPI mode. The code have not been tested for older versions.

3.2 CC2650 Launchpad

We use the Texas Instruments CC2650 Launchpad[2] with standart configuration.

3.2.1 IDE

We used the Code Composer Studio v7.1, when we refer project_zero_stack and project_zero_app we mean the sample projects provided by TI inside CCS.

3.2.2 Source description

The board is flashed with two CCS programs:

- BLE stack, namely the project_zero_stack project without modifications.
- The main project is based on project_zero_app to which we added the following modules:
  - Spp_ble_server.c [7] implements serial connection server over the SerialPortService included in the project_zero_app[6]. we modified the module to interact with NBD.
  - Sd.c module - interfaces with the external SD card over SPI.
  - Nbd.c module - implements an NBD client on the launchpad
  - Nbd_task.c module - handles the message queues of incoming/outgoing NBD messages.

3.2.3 Program architecture

The program is composed from several tasks each handles asynchronously messages assigned to it in its message queue and in between wait on a semaphore. Tasks pass messages to other tasks by adding them to their queue and posting their semaphore. The tasks in the program are:

- NBD task
- Spp task

- GAP task
- BLE stack task
- Idle task

GAP Responsible of the basic BLE GAP communication, report services the board exports.

Spp task This task runs the logic of spp_ble_server.c that implements UART over BLE, it treats ble characteristics write into UART input and converts UART output into write notify, specifically in our case, input packets are queued to the NBD module which interprets the input as NBD read/write request, same for the other direction.

NBD task The NBD task translates UART data into NBD messages and vice versa. It interacts with the NBD module which in turn interacts with the SD module for read/write actions.

BLE_stack task Runs the BLE back-end.

3.3 Peer side(PC) - NBD

The NBD is an open source project. NBD is composed from a Linux kernel module that is a part of the mainline kernel and a usermode client side and a server side. The PC using the DOK is required to have the NBD module loaded and run the client side daemon. The server side is implemented on the CC2650 Launchboard.

3.4 Peer side(PC) - Socket

A python script that connects the NBD socket to ble session. For the sake of the completeness of the project this script does the bare minimum namely transferring data from the socket to the BLE and vice versa.

4 Challenges

- Sd card won't respond to CMD8 - find out we used an SD card that answers protocol v1.10 while CMD8 exists from v2.0.
- Sd card returns unexpected data from CMD8 - turns out SD card manufactured by SanDisk does not behave well in SPI mode.
- Reading some data from the card, overriding with the different data, rerunning the program and reading the old data - if you don't plug the board off the power source between runs some data stays cached.

- SPI interaction won't work from main function - it uses a semaphore that can only be taken from a task.
- Program crashed from a heap allocation failure at an early stage - after a lot of effort we have managed to have a heap of 6KB that can handle at most 1 NBD request on the board at a time.
- Writing to an unmmaped area which result in a segmentation fault actually crashes later at an unrelated point - The writes are buffered so the program will crash only when the actual write happens and not when it executes the writing opcode - at the beginning of the program we write to the Auxiliary Control HW register to disable write buffering
- Handle SPI commands from the context of BLE message handling is very slow - moved it to another task (nbd_task.c).
- BLE attributes are 20 bytes size at max, we need to split the NBD messages and blocks of data (512 bytes) which creates a lot of overhead.
- BLE messages handeling is not in the order they were sent - add a sequence number which increases the overhead.
- Debugging challenges:
  - If compiling with size optimizations debugging is more difficult, breakpoints stop in a different place than requested.
  - Can't switch between tasks in debugger.
  - When program hits a breakpoint Bluetooth gets disconnected.
  - Debug prints aren't shown for some reason so we don't have that.
  - Board does not support more than 4 breakpoints at a time.

5 Build and run instructions

1. Prepare a PC with Bluetooth support running Linux with kernel 3.10 and above, For example Ubuntu 14.04 or new version.
2. Checkout the source[8]
3. Download the NBD module by using git submodule or download directly. If the master branch does not compile in later steps try the premade tarball
4. Connect the TI CC2650 with the SD card reader as instructed above.
5. Insert a microSD card manufactured by **any company other than Sandisk**, preferably a new card of 4GB storage or more.
6. Flash the Board with the project_zero_stack_cc2650 and project_zero_app_cc2650launchxl

7. follow the NBD instructions to compile the client side daemon on load the kernel module.
8. on the PC side
    (a) python bt_nbd/bt_socket.py (as root)
    (b) nbd/nbd_client -N client localhost 33333 (as root)
9. on the board side:
    (a) run project_zero_app_cc2650launchxl

6 Performance

The disk on key support Arbitrary storage amount limited by the bounds of the SD card and the block device driver. however the speed turned up to be very slow at a rate of 1KB/s. The main bottlenecks are the small amount of RAM on the launchpad allowing us to handle only 1 request at the time and the time for a BLE packet to go over the air.

References

[1] https://www.sparkfun.com/products/544

[2] http://www.ti.com/product/CC2650

[3] http://users.ece.utexas.edu/~valvano/EE345M/SD_Physical_Layer_Spec.pdf

[4] https://nbd.sourceforge.io/

[5] http://www.ti.com/lit/ug/tidu997a/tidu997a.pdf

[6] http://www.ti.com/lit/ug/tidua63/tidua63.pdf

[7] TI BLE examples - spp_ble_server.c

[8] Our source code