EE405 Electronic design lab weeks 4~5

Lab2 Experiment Guide: GPIO

Objectives

The goal of this lab is to control solenoid magnet using GPIO hardware in various ways: command lines using sysfs, shell script and C++ program. First of all, basic experiment using LEDs will be conducted as preparation. Then you will practice how to control solenoid magnet which will be used as end effector(i.e. gripper) of the robotic manipulator arm.

- Building a simple **switching circuit** for safe GPIO control
- Learning how to access hardware from software (**linux kernel**)
- How to control GPIO using code
    - by **command using sysfs**
    - by **shell script**
    - by **c++ file executable**

Problem statement

We'll implement an switching circuit using Beaglebone GPIO and transistor array, to implement LED light controller and solenoid magnet controller.

- Switching circuit here means a wired circuit which transistor used for electric switch operation.
- For the LED light controller, you will wire two LEDs to the switching circuit and drive them by commands with sys file system(sysfs), shell script and C++ program.
- For solenoid magnet controller, you will wire solenoid magnet to the switching circuit and drive them by C++ program. This will be directly applied to your robotic manipulator arm project.

Problem 2A.

Wire two LEDs using GPIO, with transistor array and resistors. Test commands with sysfs to control the USR0 LED on Beaglebone, and then test commands with sysfs for two hard-wired LED lights.

Problem 2B.

Control two LED lights wired on switching circuit using shell script.

Problem 2C.

Control two LED lights wired on switching circuit using C++ program.

Problem 2D.

Wire switching circuit for solenoid magnet with transistor array and resistors. Control the solenoid magnet using a C++ program.

Backgrounds

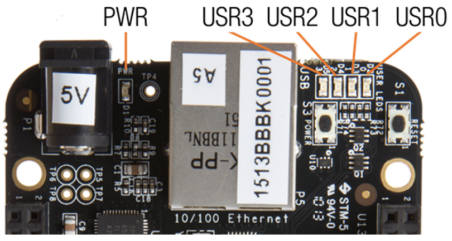
Hardware components summary

1. User LEDs on BeagleBone

Getting Started with BeagleBone [http://beagleboard.org/getting-started]

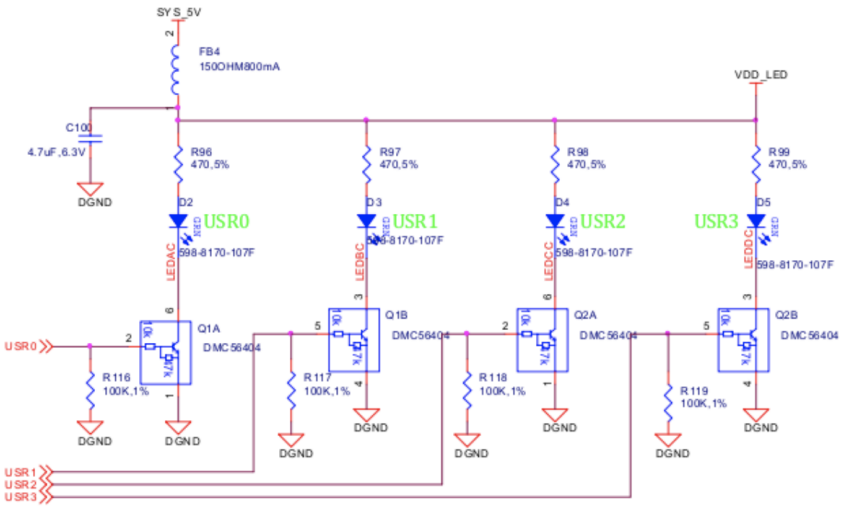
You'll see the PWR LED lit steadily. Within 10 seconds, you should see the other LEDs blinking in their default configurations.

- USR0 is configured at boot to blink in a heartbeat pattern
- USR1 is configured at boot to light during microSD card accesses
- USR2 is configured at boot to light during CPU activity
- USR3 is configured at boot to light during eMMC accesses



2. Circuit for BeagleBone User LED

Four user LEDS are provided via GPIO pins on the processor. Figure below shows the LED circuitry. [See p. 44, 7.7.3 User LEDs, Beaglebone System Reference Manual A5].



- Four user LEDs are connected to GPIO pins as shown in the following Table.

Table 2.1 User LED Control

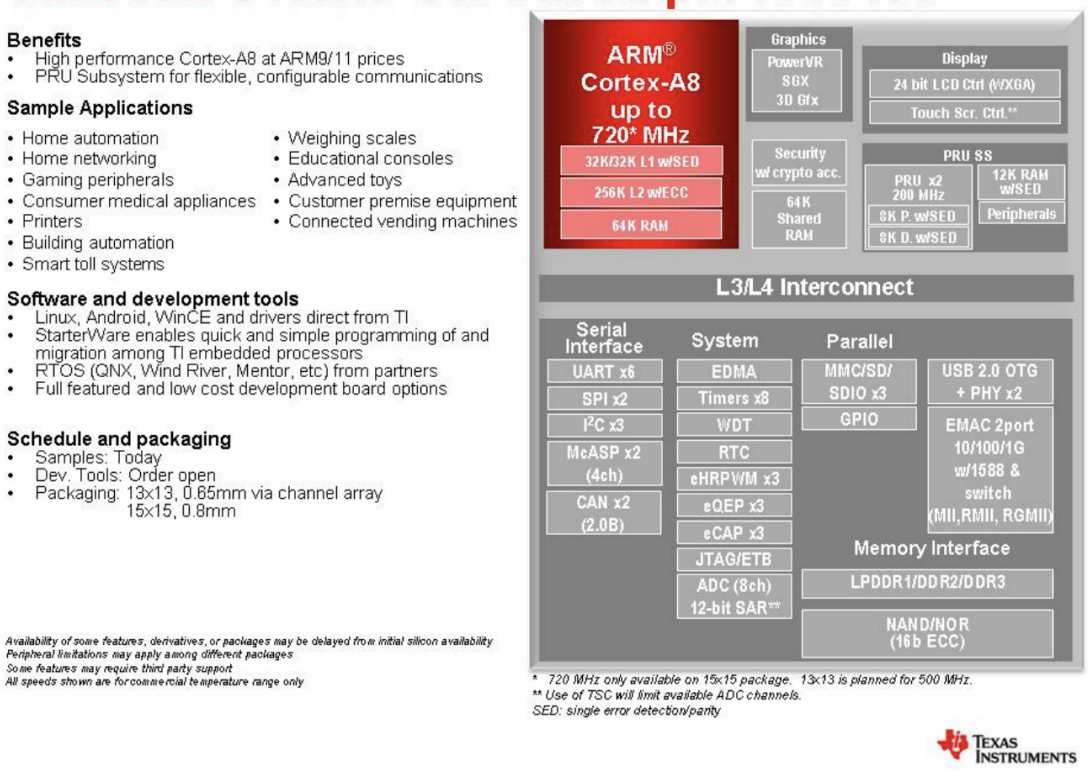
LED	GPIO
User 0	GPIO1_21
User 1	GPIO1_22
User 2	GPIO1_23
User 3	GPIO1_24

We are going to control the USR0 LED, which is connected to the processor via GPIO1_21.

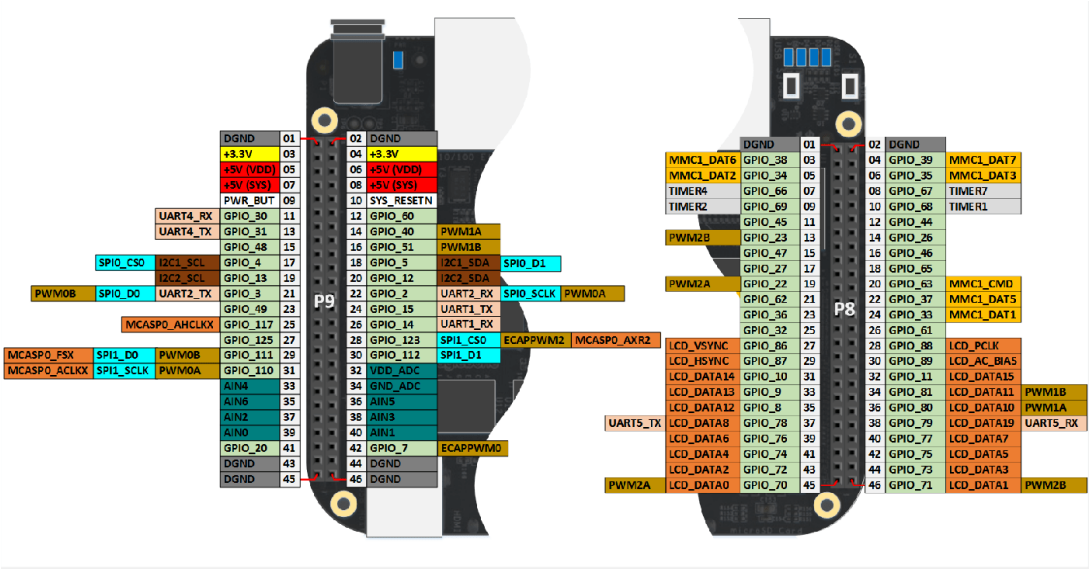
3. GPIO

GPIO(General-purpose input/output) means digital signal pin on electronic circuit board, can be used as both input or output.

- GPIO is controllable by software(in this case, sysfs command / shell, C++ executables).
- BeagleBone Black board supports GPIO by its digital pin.



Internal block diagram of AM3359 Processor [Refer http://www.ti.com/product/am3359]



BeagleBone Black pin map(including GPIO) [Refer https://kr.mathworks.com/help/supportpkg/beagleboneio/ug/beaglebone-black-pin-map.html]

- **Purpose of GPIO peripheral in AM335x processor**

The general-purpose interface combines four general-purpose input/output (GPIO) modules. Each GPIO module provides 32 dedicated general-purpose pins with input and output capabilities; thus, the

general-purpose interface supports up to 128 (4 × 32) pins. These pins can be configured for the following applications:

- Data input(capture) / output(drive)
- Keyboard interface with a debounce cell
- Interrupt generation in active mode upon the detection of external events. Detected events are processed by two parallel independent interrupt-generation submodules to support biprocessor operations.
- Wake-up request generation (in Idle mode) upon the detection of signal transition(s)

- **How to connect GPIO Output safely?**

Selection of the resistor is important for safe GPIO Connection. We are going to limit the output current to be LESS THAN 1mA, regardless of connected logic circuit. What resistor will be sufficient between GPIO output and connected logic circuit?

** GPIO output is 0V(OFF) or 3.3V(ON), and the connected logic circuit may be operated by +5V power.

In this case,

- GPIO Logic 0 output of 0V → Connected logic 0V. Regarded as logic 0.
- GPIO logic 1 output of 3.3V → Connected logic 3.3V. Regarded as logic 1 since 3.3V > threshold (half of 5V).
- We attach resistor of 5kΩ between GPIO output and external logic input. In this case, the maximum current is 3.3V / 5 kΩ = 0.66 mA < 1mA.

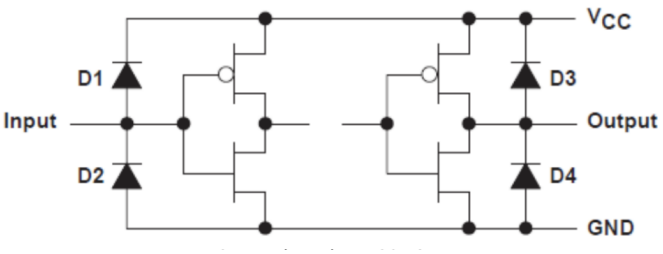
Therefore the solution is **GPIO output → 5KΩ resistor → External 5V logic input.**

- **How to connect GPIO Input safely?**

External logic may use +5 V power. The external logic output (0V / 5V) is to be connected to the GPIO input, which is NOT 5V TOLERANT.

- External logic output 0 (0V) → GPIO input 0V. Regarded as logic 0. OK.
- External logic output 1 (5V) → GPIO input 5V. Beaglebone uses 3.3V. **BURNS BEAGLEBONE!**

Why external 5V to GPIO input burns Beaglebone? In figure below, suppose the Input (in the left side) is driven by 5V logic output. The voltages in pins of protection diode D1 are anode voltage of 5V and cathode voltage of 3.3V (Vcc), and hence producing voltage drop of 1.7V. The current through D1 became very large and it will burn out!



We'll attach resistor of 5kΩ between external logic output and GPIO input.

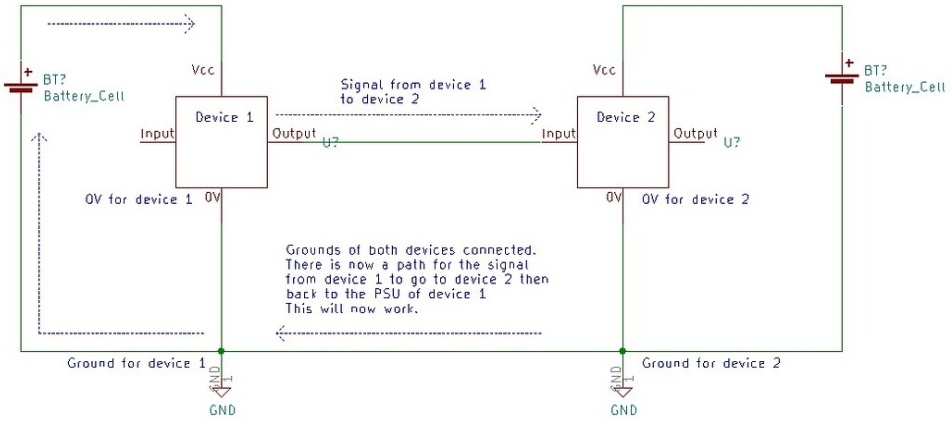
In this case, the diode D1 is forward biased with 0.7V voltage drop with current (5–3.3–0.7)V / 5kΩ = 0.2mA. The voltage at the Input (GPIO input pin) became Vcc + 0.7 = 4V. This will in effect make the pin 5V-tolerant for digital I/O.

Thus, the solution becomes **External 5V logic output → 5 kΩ resistor → GPIO input!**

- **Common ground for embedded board**

As mentioned above, while wiring an electric circuit with a Beaglebone board and additional external power supply, you should be aware of the voltage difference between GPIO port and external power supply source.

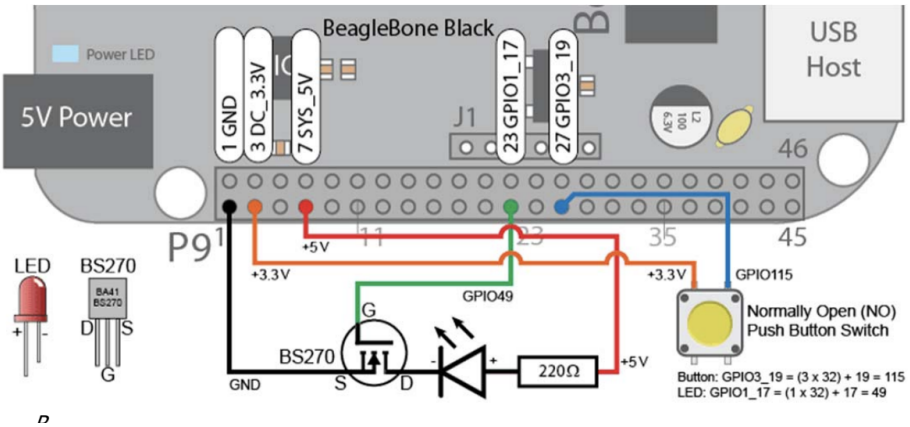
In this case, designating common ground is very important. For better understanding, please refer to the following embedded board forum about common ground;
[https://forum.arduino.cc/t/common-ground-and-why-you-need-one/626215]



- **GPIO to LEDs Circuit**

Similar to the User LEDs circuit in Beaglebone, we are going to construct circuit with 2 LEDs for this lab on Breadboard. LEDs are driven by GPIO also, and connection from Beaglebone to Breadboard can be done using two I/O connectors P8/P9 as follows:

- AM3359 CPU → GPIO → P8/P9 → Transistor array IC → Light LEDs with R



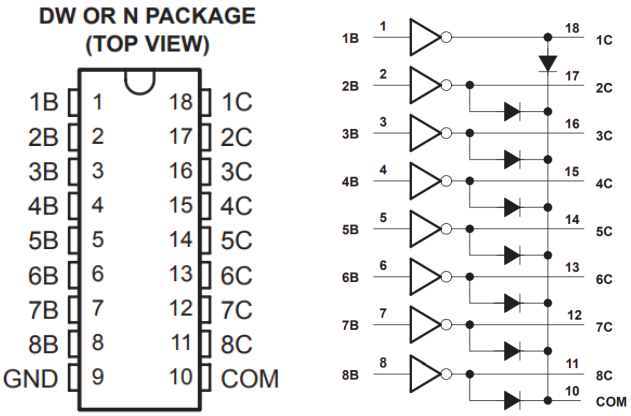
- Typical external ED circuit for Beaglebone can be found on the Web as follows [http://derekmolloy.ie/kernel-gpio-programming-buttons-and-leds/].
- Instead of discrete transistor, we are going to use uLN2803AN, which is an 8-unit Darlington transistor array up to 500mA in DIP type package.

Use two GPIOs to **control two LEDs each**. We selected **GPIO0_30 and GPIO0_31** as follows:

- **Wiring for Light 1:**
  GPIO0_30(P9.11) → R1 → uLN2803AN B input; uLN2803AN OC output → R2 → Cathode of LED; Anode of LED → 5V

- **Wiring for Light 2:**
  GPIO0_31(P9.13) → R1 → uLN2803AN B input; uLN2803AN OC output → R2 → Cathode of LED; Anode of LED → 5V

  ** Please refer to uLN2803AN datasheet to distinguish input/output pin



We select R1 as 5kΩ for safety. Since GPIO output can be either 0V or 3.3V, even when the other side of resistor is shorted to ground or Vcc of 5V, the output current of GPIO is less than 1mA.

We need to select the value of R2 to limit the current of LED to be less than the specified. Determine the value of R2, assuming the voltage drop of the LED is 3.5V.

4. User LED control using command (Prob 2A)

- Four User LEDs does appear as directories in /sys/class/leds as

  beaglebone:green:usr0@ beaglebone:green:usr1@ beaglebone:green:usr2@ beaglebone:green:usr3@

- In the directory beaglebone:green:usr0, you can control USR0 LED by writing to each file:

  trigger, brightness, delay_on, and delay_off.

- During the design section, you have to proceed followings:
  - GPIO via the Shell Command Line and sysfs.
  - Flashing the user LEDs.
- For details, read the following sections(web page) and test by yourself:
- EBC Exercise 10: Flashing an LED [http://elinux.org/EBC_Exercise_10_Flashing_an_LED]
- Lab 2 procedure Step 1, 2 (Problem 2A)

5. GPIO LED control using command and sysfs

- gpio-sysfs

  gpio-sysfs is the preferred method of GPIO interfacing from user space. Use this unless you absolutely need to update multiple GPIOs simultaneously or you must use a kernel version before 2.6.27. The full documentation is alongside the gpio-framework documentation:

  - [https://www.kernel.org/doc/Documentation/gpio/sysfs.txt]
  - [https://www.kernel.org/doc/Documentation/gpio/gpio.txt]
- During the design section, you have to:
- Wire 2 LED lights and transistor array with Beaglebone GPIO ports on the breadboard to realize switching circuit.
- Control LED lights using command and sysfs.
  - How to use GPIO signals in Linux [https://developer.ridgerun.com/wiki/index.php/How_to_use_GPIO_signals]
  - For detailed procedure, see Lab2 Procedures.

6. LED control with shell script (Prob 2B)

The shell provides you with an interface to the UNIX system. It reads input from you and executes the programs you specified. While the programs are executing, it displays their output. The real power of the shell lies in the fact that it is much more than a command interpreter. It is also a powerful programming language, complete with conditional statements, loops, and functions.

- We need two files in the design section:
- ui_control_light.sh: Test light control along with user input.
- loop_control_lights.sh: Test time consumption of the ON/OFF light control loop for N times.
- Please refer to [Lab 2 Procedure Step 3. Shell program for Lights LED] during your design section.

For your code design,

- ui_control_light.sh will be given. Try to understand the principle of the code by writing down the given shell script.
- loop_control_lights.sh will be given as skeleton code. Follow the instructions written on the skeleton code to complete them.

You might utilize the shell commands used on Prob 2A. After completing code writing, make it into executables and execute them to control LED lights.

- For the basic knowledge of the bash scripting please refer to following links:
- Ubuntu documentation - Beginners/BashScripting [https://help.ubuntu.com/community/Beginners/BashScripting]
    - Especially, Sections "Scripting" to "Functions".
- Bash test operators [https://kapeli.com/cheat_sheets/Bash_Test_Operators.docset/Contents/Resources/Documents/index]

7. LED control using cpp and sysfs (Prob 2C)

We are going to control LED lights by managing GPIO signal using C++ and sysfs.

- We need four files:
- gpio_control.hpp: Define functions to manage GPIO signal [Given already]
- gpio_control.cpp: The codes consists of the body of the gpio_control.hpp
- test_light_control.cpp: Test light control along with user input.
- loop_light_control.cpp: Test time consumption of the ON/OFF light control loop for N times.
- Please refer following for your code:
- Access GPIO from Linux user space [http://falsinsoft.blogspot.kr/2012/11/access-gpio-from-linux-user-space.html]
- How to use GPIO signals in Linux, C/C++ [https://www.ics.com/blog/how-control-gpio-hardware-c-or-c]

Basically, gpio_control.hpp and skeleton code for gpio_control.cpp will be given.

- We are going to complete skeleton codes in the design section.
- You might use VSCode cmake extension to generate CMakeLists.txt and cross-compile your C++ files. Please refer to cmake on VSCode on the Lab1 Procedure.

8. switching circuit for solenoid magnet (Prob 2D)

We are going to wire switching circuit to control solenoid magnet.

- before wiring switch circuit, please revisit [3. GPIO] section of this documentation to remind precautions for wiring an circuit containing external additional power supply.
- This switching circuit has almost similar design with the one of Problem 2B/2C, but you have to utilize 12V external power supply to satisfy the appropriate voltage of solenoid magnet.

We will use **GPIO_30 to control solenoid magnet**.

- **Wiring for solenoid magnet:**

- Electric switch input part: GPIO0_30(P9.11) → R3 → uLN2803AN B input
- Power supply part: 12V Power supply(+) → R4 → Solenoid magnet* → uLN2803AN C output
- Common ground part: GPIO GND → uLN2803AN GND; 12V Power supply(-) → uLN2803AN GND;

  *You can connect solenoid magnet on the circuit in any direction.

- please refer to the specification of solenoid magnet on following URL link(12V / 0.26A / 2.5W) [https://negamy.com/product/cl-p-20-15-holding-electric-magnet-lifting-2-5kg-25n-solenoid-sucker-electromagnet-dc-6v-12v-24v-non-standard-custom/]

9. Gripper control using cpp and sysfs (Prob 2D)

To utilize it later, we are going to make cpp files that one turns on and the other turns off the solenoid magnet.

- We need two files:
- gripper_control.hpp: Define functions to ON/OFF solenoid magnet [Given already]
- gripper_control.cpp: Actual body of solenoid control functions to switch on the gripper
- gripper_test.cpp: file tests functions defined in gripper_control.cpp

```
// File gripper_control.hpp
// Function definitions for gripper_control.cpp

int gripper_open(unsigned int gpio);
// open the GPIO port for the solenoid magnet

int gripper_on(unsigned int fd_gpio);
// turn on the solenoid magnet

int gripper_off(unsigned int fd_gpio);
// turn off the solenoid magnet

int gripper_close(unsigned int gpio, unsigned int fd_gpio);
// close the GPIO port for the solenoid magnet
```

- Design functions defined on the gripper_control.cpp
  - You might have to use the basic GPIO control functions declared in gpio_control.hpp
- Complete writing given skeleton code of the gripper_test.cpp to test the functions defined on the gripper_control.hpp
  - gripper_test will 1) open the GPIO port, 2) turn on the solenoid magnet for several seconds and 3) turn off and close the GPIO port.
- You might use VSCode cmake extension to generate CMakeLists.txt and cross-compile your C++ files. Please refer to cmake on VSCode on the Lab1 Procedure.

Preparation

List of components

- Router and an ethernet cable
- IBM PC with Linux
- External power supply(only for Prob 2D)
- Beaglebone set: Beaglebone embedded board, USB cable, 1 ethernet cable, 4(or More) GB microSD card, microSD card reader
- Breadboard
- Electronic parts
    - CP41B-WES-CK0P0154(DIP LED White, 30 mA, 2EA)
    - uLN2803AN(Darlington TR array, 18 pin DIP, 1EA)
    - CL-20/15(Electric magnet, 12V / 260mA, 1EA)

Lab Procedures

- See the documentation "EE405 Lab2 Procedure."

Final Report

Discussion for the following questions should be included in the report.

1. [Shell script vs. C program]

   Compare Shell script and C program to control LEDs. Please include the advantages and disadvantages of each on the report.

2. [Transistor array]

   Why do we require a transistor array to drive LEDs? Can you drive 3W LEDs using uLN2803?

3. [Solenoid magnet circuit]

   Today we applied external power supply to control the solenoid magnet.

   Write some basic idea to form an electronic circuit to control solenoid magnet when you apply voltage 1) using beaglebone's own GPIO signal and 2) using external power supply.

   Please include what you have to consider to apply stable electric power and control the solenoid magnet.

4. [End effector]

   Solenoid magnet in this experiment will be used as a 'gripper' for a robotic manipulator project, in other words, an end effector.

   Please refer to the article follows, which explains about various kinds of the end effector. [https://www.universal-robots.com/blog/robot-grippers-explained/]

Choose one of the end effectors, and describe the differences in expected driving methods and relative advantage/disadvantages compared to solenoid magnet gripper.

5. [Discussion - own topic]

   Set your own topic to discuss related to Lab 2, and explain summary of your search result.

References

[1] Beaglebone System Reference Manual, https://cdn-shop.adafruit.com/datasheets/BBB_SRM.pdf

[2] AM3359 Datasheet, AM335x ARM Cortex-A8 Microprocessors (MPUs) (Rev. J), http://www.ti.com/lit/ds/symlink/am3359.pdf

[3] Technical Reference Manual - AM335x ARM Cortex-A8 Microprocessors (MPUs) Technical Reference Manual (Rev. O), http://www.ti.com/lit/ug/spruh73o/spruh73o.pdf