CG1112 Engineering Principles and Practice II
Week 9 Studio 2
Power Management

Core Objectives:

- **C1. Introduction to Power Management**
- **C2. Exploring the Power Management Features on Your Raspberry Pi**
- **C3. Exploring the Power Management Features on Your Arduino Uno**
- **C4. Examining the Power Consumption of your RPLIDAR**

Team Setup:

- Work with your project team for this studio as you need the RPi, Arduino, and LiDAR.

Equipment Setup:

- Remember to bring the following:
  - Your fully assembled Alex consisting of Raspberry Pi, Arduino Uno, and RPLIDAR with the given microUSB-to-USB cable
  - USB PowerBank and AA batteries (fully charged ☺)
  - USB power measurement device (UT658) previously issued to your team
- You will need your **completed Alex.ino code** from **Week 8 Studio 1**, as well as the Alex-pi.cpp code given to you for Week 8 Studio 1.
- Ensure that your RPi's **Wi-Fi** configuration is working, and you can access it using **VNC**.

C1. Introduction to Power Management

Just like many other professions, social responsibilities are a vital part of the engineering profession. In Week 3 Studio 1, you have learnt about the importance of using non-toxic/reduced-toxicity materials in engineering products, which is also why we must always use lead-free solder despite their less-desirable qualities compared to leaded solder. In this studio, we focus on another important aspect of *environmental awareness* and *sustainability* – the need to design engineering solutions that are power-efficient! Not only would a power-efficient engineering product save on utility costs or run longer on batteries, it also helps create a healthier and more sustainable environment by reducing our carbon footprint.

Your Alex, being a search and rescue robot, needs to accomplish its mission in a time-critical manner. While time-criticality and power-efficiency may be conflicting requirements in some instances, it is the responsibility of every engineer to evaluate the design trade-offs and use his/her ingenuity to come up with a balanced solution.

Today's electronic products are complex and come with multiple subsystems. Not all the subsystems need to be powered on at the same time or be operated at their peak performance. If you look around you, almost all electronic products already come with some form of power management features that can turn off individual subsystems (e.g., your smartphone's screen turns off when in standby mode), or put them into low-power mode (e.g., a PC reducing the clock frequency of the microprocessor when there is not much processing task). In its basic configuration, your Alex consists of several electronic modules – Raspberry RPi, Arduino Uno, RPLIDAR, motor driver chip, etc. Many of these modules also come with built-in power management features (because they are designed by responsible engineers!)

In this studio, we shall explore some of the ways we could manage the power consumption of the RPi, the Arduino, and the RPLIDAR. It is important to note that this studio does not provide a full comprehensive list of all the power management features that are available in these modules. Its aim is to expose some of these possibilities to you, with the hope that you could further discover additional power saving methodologies on your own.

C2. Exploring the Power Management Features on Your Raspberry Pi

The Raspberry Pi, being a tiny computer, can consume more than 450 mA of current when in headless mode (using WiFi for remote access, no keyboard/mouse attached), even when it is idling. In this section, we will experiment with some of the ways to reduce the RPi's power consumption.

We will be using the USB power measurement device (UT658) that allows us to measure the instantaneous current drawn, as well as the voltage. It also allows us to measure the cumulative milli-ampere hour (mAh) that has been consumed (since reset). You can press the button on the UT658 for about 2 seconds to reset the mAh counter.

Disconnect your **Arduino Uno** and **RPLIDAR** from your RPi for this part of the studio.

1. **<u>Connect the UT658 to your USB PowerBank,</u>** then connect your RPi to the UT658. Turn on the USB PowerBank, and boot up your RPi. You can unplug your keyboard and mouse from your RPi once you can successfully access it using VNC. Unplugging these unnecessary peripheral devices from your RPi will reduce some power consumption.

2. Take note of the voltage and current readings on the UT658 once they have stabilized:
   Voltage reading: _____
   Current reading: _____
   What is the instantaneous power consumption calculated from the above readings?
   Instantaneous power: _____
   (Shame your teammates if they have forgotten the formula from CG1111. ☺)

3. We shall now attempt to turn off the HDMI port, since we are running headless RPi.
   Open a terminal, and enter: **`/usr/bin/tvservice -o`**
   Take note of the new current reading: _____

> **Note:**
>
> 1. You can turn the HDMI port back on by entering:
>   **`/usr/bin/tvservice -p`**
>   However, the screen may not be displayed correctly. But not to worry, everything will go back to normal after rebooting.
>
> 2. It is also possible to disable HDMI on boot by entering the command in
>   **`/etc/rc.local`**
>   However, this is not recommended as you won't be able to see what's going on if your WiFi or VNC fails for some reason.

4. Your RPi B+ has an onboard chip – the Microchip LAN7515 – which is essentially a 4-port USB Hub controller chip. The RPi's Ethernet connectivity is actually implemented using Ethernet over USB. Hence, its Ethernet port also relies on this chip. Such an integrated approach helps to reduce the RPi's production cost, as well as its form factor. We shall now examine the power drawn by this chip by turning off its power. In your terminal, enter: **`echo '1-1' |sudo tee /sys/bus/usb/drivers/usb/unbind`**
   Take note of the new current reading: _____
   What is the amount of current reduction? _____

> **Note:**
>
> 1. The above step actually turns off all the USB ports, as well as the Ethernet capability. Your remote access is not affected since you are using WiFi.
>
> 2. Notice that this chip consumes about half of the RPi's idling power (wow!). Unfortunately, you cannot turn off this chip entirely for your Alex project, since you need your USB ports for the RPLIDAR, as well as the Arduino Uno. However, you might have future hobby projects that do not require Ethernet and USB; you can consider doing the above for such projects.

5. Now, we will turn the LAN7515 chip back on by entering:
   `echo '1-1' |sudo tee /sys/bus/usb/drivers/usb/bind`
   Grieve for the increase in power consumption again.

6. But alas, all is not lost! In the following, we will try to turn off the Ethernet function only. To do so, we need to install an additional Linux module, download a third party's source code from GitHub, compile it, and then run it. Enter the following in your terminal:
```
sudo apt-get update
sudo apt-get install libusb-dev
git clone https://github.com/codazoda/hub-ctrl.c
cd hub-ctrl.c
gcc -o hub-ctrl hub-ctrl.c -lusb
cp hub-ctrl ..
cd ..
```

7. We now issue the command to turn off the Ethernet port alone, without affecting the USB ports:
```
sudo ./hub-ctrl -h 0 -P 1 -p 0
```
Take note of the new current reading: _____
What is the amount of current reduction? _____

Note:

1. If you need to turn the Ethernet back on, enter the following:
```
sudo ./hub-ctrl -h 0 -P 1 -p 1
```

2. Some online resources claim that you can also use the above utility to turn off individual USB ports. We have tried but it didn't seem to work reliably with the RPi B+. You are left on your own to further explore if you can get it to work. If so, you can save additional power by turning off any unused USB ports!

At this point, we have reduced our RPi's idling power consumption by about 20%. We will end our exploration of RPi's power management features here. In the following, we list some other methodologies that can also reduce the RPi's power consumption, although not all of them would be useful for your Alex project's specific requirements and mission demands. You are encouraged to assess and explore whether they could be useful for your Alex project.

- Reducing the number of cores that can be activated.
- Lowering the maximum clock speed of its CPU.
- Lowering the minimum clock speed for its CPU (the power reduction seems insignificant if the workload is low in the first place).

- Changing the "scaling governor" setting; the scaling governor is a software kernel program that can switch the clock speed dynamically based on the processor load. Its setting can also be changed manually on the fly.
- Disabling Bluetooth (the power reduction seems insignificant, probably because we are using the same chip for WiFi, which we need in Alex).
- Shutting off the PWR and ACT LEDs.

You can now shut down your RPi. In the next activity, you will need to **move the UT658 to the USB port of your RPi**, so that it can directly measure the power requirements of the Arduino Uno connected to your RPi.

C3. Exploring the Power Management Features on Your Arduino Uno

Next, we move on to explore the power management and sleep modes that are built into the ATmega328P. The table below (from ATmega328P's datasheet) shows the six sleep modes that are available in the ATmega328P. Among these sleep modes, the "Idle" mode gives the least power savings (because it turns off the least peripherals), while the "Power-down" mode gives the most power savings (because it turns off the most peripherals, potentially drawing only a few μA!).

Active Clock Domains and Wake-up Sources in the Different Sleep Modes

	Active Clock Domains					Oscillators		Wake-up Sources							
	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other I/O	Software BOD Disable
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X	
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X(2)	X	X	X		
Power-down								X(3)	X				X		X
Power-save					X		X(2)	X(3)	X	X			X		X
Standby(1)						X		X(3)	X				X		X
Extended Standby					X(2)	X	X(2)	X(3)	X	X			X		X

Notes: 1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.

From the table, notice that each sleep mode specifies a list of clocks and oscillators that remain active, as well as a list of "wake-up sources" that can wake up the microcontroller unit (MCU). To assess whether a sleep mode would be suitable for a specific application, you need to consider whether its remaining active peripherals can *meet the applications' needs* while sleeping, and whether you have *the means to wake up the MCU from sleep*.

For example, it is tempting to consider using the "power-down" sleep mode for your Alex as it gives maximum power savings. However, the I/O is not working under this mode, and your RPi won't be able to wake up the Arduino by sending serial data through USART. Although INT1, INT0 and Pin Change Interrupt (PCINT) could potentially be used as wakeup sources, you need to be aware that the RPi's GPIO runs on 3.3 V, while the Arduino Uno's GPIO runs on 5 V. (Using a logic level converter would introduce additional power consumption!) In addition, INT0 and INT1 are already used by Alex's wheel encoders.

In this part of the studio, **we will only explore the "Idle" sleep mode** as it is convenient to wake up your Alex's Arduino whenever your RPi sends serial data to it. You are free to consider other sleep modes in your project if you can get them to work. However, we don't advise spending time on this because the additional power saving you can achieve is really insignificant compared to the power consumed by the RPi and the RPLIDAR.

SMCR – Sleep Mode Control Register

The **Sleep Mode Control Register (SMCR)** contains control bits for power management:

Bit	7	6	5	4	3	2	1	0	
0x33 (0x53)	–	–	–	–	SM2	SM1	SM0	SE	SMCR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The sleep mode is selected by setting the appropriate **Sleep Mode Select Bits (SM[2:0])** according to the following table:

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Standby(1)
1	1	1	External Standby(1)

The **Sleep Enable (SE)** bit must be written to '1' to allow the MCU to enter the sleep mode when the SLEEP instruction is executed. To avoid the MCU entering the sleep mode unless it is the programmer's intention, it is recommended to write the SE bit to '1' just before the execution of the SLEEP instruction and to clear it immediately after waking up.

PRR – Power Reduction Register

When putting the MCU to sleep, we should also try to further minimize power by disabling functions/modules that are not needed. The Power Reduction Register (PRR) provides a method to stop the clock to individual peripherals to reduce power consumption. The following shows the layout of the PRR:

Bit	7	6	5	4	3	2	1	0	
(0x64)	PRTWI	PRTIM2	PRTIM0	–	PRTIM1	PRSPI	PRUSART0	PRADC	PRR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 (PRTWI): Power Reduction TWI**
  Writing a '1' to this bit shuts down the Two Wire Interface (TWI)/I2C.

- **Bit 6 (PRTIM2): Power Reduction Timer/Counter2**
  Writing a '1' to this bit shuts down the Timer/Counter2 module.

- **Bit 5 (PRTIM0): Power Reduction Timer/Counter0**
  Writing a '1' to this bit shuts down the Timer/Counter0 module.

- **Bit 4: Reserved**
  This bit will always be read as '0'.

- **Bit 3 (PRTIM1): Power Reduction Timer/Counter1**
  Writing a '1' to this bit shuts down the Timer/Counter1 module.

- **Bit 2 (PRSPI): Power Reduction Serial Peripheral Interface (SPI)**
  Writing a '1' to this bit shuts down the SPI by stopping the clock to the module.

- **Bit 1 (PRUSART0): Power Reduction USART0**
  Writing a '1' to this bit shuts down the USART by stopping the clock to the module.

- **Bit 0 (PRADC): Power Reduction ADC**
  Writing a '1' to this bit shuts down the ADC. The ADC must be disabled before shutdown. To disable ADC, we must write a '0' to bit 7 of ADCSRA register.

Watchdog Timer (WDT)

The WDT is a timer counting the cycles of a separate on-chip 128kHz oscillator. It gives an interrupt or a system reset when the counter reaches a given time-out value. In normal operation mode, the system needs to restart the counter before the time-out value is reached,

otherwise an interrupt or system reset will be issued. This interrupt can be used to wake the device from sleep-modes, and also as a general system timer. One example is to limit the maximum time allowed for certain operations, giving an interrupt when the operation has run longer than expected. In System Reset mode, the WDT gives a reset when the timer expires. This is typically used to prevent system hang-up in case of runaway code, which is particularly important for remote systems (such as satellites in space) that you cannot physically access to reset them.

If the WDT is not needed in the application, the module should be turned off to save power. The following provides the Arduino code for turning off the WDT. If you wish to understand more about the meaning of the different bit settings, you can refer to the section "**System Control and Reset**" in your ATmega328P's datasheet.

Arduino Code for Turning Off the Watchdog Timer (WDT):

```
void WDT_off(void)
{
      /* Global interrupt should be turned OFF here if not
      already done so */

      /* Clear WDRF in MCUSR */
      MCUSR &= ~(1<<WDRF);

      /* Write logical one to WDCE and WDE */
      /* Keep old prescaler setting to prevent unintentional
      time-out */
      WDTCSR |= (1<<WDCE) | (1<<WDE);

      /* Turn off WDT */
      WDTCSR = 0x00;

      /* Global interrupt should be turned ON here if
      subsequent operations after calling this function DO
      NOT require turning off global interrupt */
}
```

Activity: Putting Your Alex's Arduino into "Idle" Sleep Mode

In the following, we will go through an example of putting your Alex's Arduino into the "Idle" sleep mode whenever the following two conditions are met: (i) *there is no USART serial data* and (ii) *the motors are stopped*. It will wake up from the sleep mode when USART serial data arrives.

Make sure that you have your **completed Alex.ino code** from **Week 8 Studio 1**, as well as the **Alex-pi.cpp code** given to you for Week 8 Studio 1.

1. If you have not yet moved the UT658 from the PowerBank's port to the USB port of your RPi, do it now. Then connect your Arduino to the UT658, and power on your RPi again. Use VNC to remotely access your RPi via WiFi.

2. Using the Arduino IDE in your RPi, open your **completed** Alex.ino code from the end of Week 8 Studio 1. It should still be OK if you have already made additional modification since that studio, so long as your Alex.ino on the Arduino can receive commands from the Alex-pi running on your RPi.

3. In your Alex.ino, add the following header file. The "sleep.h" header file contains the definition of the function sleep_cpu(), which puts the ATmega328P into sleep.

```
#include <avr/sleep.h>
```

4. Next, add the following mask definitions into your Alex.ino.

```
#define PRR_TWI_MASK             0b10000000
#define PRR_SPI_MASK             0b00000100
#define ADCSRA_ADC_MASK          0b10000000
#define PRR_ADC_MASK             0b00000001
#define PRR_TIMER2_MASK          0b01000000
#define PRR_TIMER0_MASK          0b00100000
#define PRR_TIMER1_MASK          0b00001000
#define SMCR_SLEEP_ENABLE_MASK   0b00000001
#define SMCR_IDLE_MODE_MASK      0b11110001
```

5. Add the WDT_off() code (given to you above) into your Alex.ino as well.

6. You will now write a function named as setupPowerSaving(). Follow the guidelines given in the comments below. Note that we intentionally made Arduino Uno's Pin 13 an output pin, and write a logic LOW to it so as to keep the LED tied to Pin 13 off. If left as input pin, the voltage fluctuates with noise and the LED is usually ON all the time, wasting power.

```
void setupPowerSaving()
{
  // Turn off the Watchdog Timer

  // Modify PRR to shut down TWI

  // Modify PRR to shut down SPI
```

```
  // Modify ADCSRA to disable ADC,
  // then modify PRR to shut down ADC

  // Set the SMCR to choose the IDLE sleep mode
  // Do not set the Sleep Enable (SE) bit yet

  // Set Port B Pin 5 as output pin, then write a logic LOW
  // to it so that the LED tied to Arduino's Pin 13 is OFF.
}
```

7. Call the function setupPowerSaving() within setup() of your Alex.ino.

8. Now, write a function named as putArduinoToIdle(). Follow the guidelines given in the comments below. Note that we shut down the timers before the MCU sleeps so as to save power, but power them up after the MCU wakes from sleep because we need the timers to be active for Alex's PWM operation.

```
void putArduinoToIdle()
{
  // Modify PRR to shut down TIMER 0, 1, and 2

  // Modify SE bit in SMCR to enable (i.e., allow) sleep

  // The following function puts ATmega328P's MCU into sleep;
  // it wakes up from sleep when USART serial data arrives
  sleep_cpu();

  // Modify SE bit in SMCR to disable (i.e., disallow) sleep

  // Modify PRR to power up TIMER 0, 1, and 2
}
```

Note: By the way, if we do not turn off Timer0, it triggers an interrupt every 1.024 ms and will wake up the Arduino from "Idle" sleep mode. This interrupt is used to implement functions like delay() in Arduino.

9. Finally, modify loop() in your Alex.ino so that it calls the function putArduinoToIdle() under the **right conditions**. Recall that our objective in this part of the studio is to put your Alex's Arduino into sleep whenever **the motors are stopped**. It wakes up from the sleep mode when **USART serial data arrives**.
   Hint: If you do it correctly, it only requires a single if() statement inside loop().

10. Compile the Alex.ino code and upload it to the Arduino.

11. Now, run Alex-pi in terminal: **`./Alex-pi`**

12. Examine the current measured by the UT658, and note down its value: _____

13. Try to command Alex to go forward like in Week 8 Studio 1. Satisfy yourself that your Arduino is still able to respond to commands from the RPi, even though it goes to sleep often (whenever the motor stops and there is no further serial data received).

> Type "f" followed by <enter>, then key in the speed (in percent) and distance to travel in cm separated by a space, and press <enter>. For example:
>
> 80 50
>
> Your Arduino must still be able to respond with "**Command OK**".

14. Next, exit from Alex-pi. Then comment out the function call to putArduinoToIdle() from loop() in Alex.ino, recompile it, and upload it to the Arduino. Rerun Alex-pi. Note down the current measured by the UT658 (this time is without entering "Idle" sleep mode): Measured current without sleep mode: _____

(NOTE: YOU MUST EXIT Alex-pi IN ORDER TO UPLOAD SKETCHES TO THE ARDUINO. IF YOU DO NOT EXIT Alex-pi, YOUR ARDUINO IDE WILL HANG WHEN UPLOADING SKETCHES.)

Congratulations! Your Arduino is now "greener" (no pun intended) than before!

C4. Examining the Power Consumption of your RPLIDAR

In Week 7 Studio 1, we have seen that the RPLIDAR is a power-hungry device that drains your PowerBank rapidly. So, how much power is your RPLIDAR using? In this part of the studio, you will be examining the power consumption of your RPLIDAR under different operating modes. We will be using the SDK that you have used in Week 7 Studio 1.

1. Unplug your Arduino Uno from the UT658. You can leave your Arduino Uno unplugged for the rest of this studio.

2. Plug in your RPLIDAR into the UT658. Observe that its motor starts to spin. In this state, the RPLIDAR's scanner module is in a SLEEP mode. Note down the current measurement on the UT658:

Current when motor is spinning, and scanner in SLEEP mode: _____

3. Next, we will make a minor modification to an app called "ultra_simple" that is provided in RPLIDAR's SDK. Using your RPi's File Manager, navigate to the folder that contains the main.cpp of the ultra_simple app, e.g.:
   /home/pi/Desktop/rplidar_sdk_v1.5.7/sdk/app/ultra_simple

4. Double-click on the main.cpp file to open it. Scroll to the bottom of the file, and add a 10 second delay above the line **RPlidarDriver::DisposeDriver(drv);**

```
delay(10000);
RPlidarDriver::DisposeDriver(drv);
```

5. Save the main.cpp file. Using the terminal, navigate to the SDK's subfolder, e.g.:
   /home/pi/Desktop/rplidar_sdk_v1.5.7/sdk

6. Type "**make**" and <enter>. Compilation should start automatically. There may be some warning messages, but no compilation error.

7. Now, navigate your terminal to the subfolder:
   /home/pi/Desktop/rplidar_sdk_v1.5.7/sdk/output/Linux/Release
   Assuming that your RPLIDAR is named by your RPi as "/dev/ttyUSB0", enter the following:
   ./ultra_simple /dev/ttyUSB0

8. At this time, your RPLIDAR should begin to perform scanning and report back the data continuously. Note down the current measurement on the UT658:
   Current when motor is spinning, and scanner in WORK mode: _____

9. Next, we will measure the current when the motor is stopped. You have **10 seconds** after exiting the scanning operation to take the reading. When you are ready, press <**Ctrl-c**> to stop the scanning and also the motor. Note down the current measurement on the UT658:
   Current when both motor and scanner are STOPPED: _____

> Notice that the RPLIDAR starts its motor again after the 10 seconds delay. This seems to be due to a bug in the RPLIDAR's firmware. The motor is restarted when the RPlidarDriver object is deleted in the DisposeDriver().

10. You now know the power consumption of your RPLIDAR in each of the three modes of operation! For the Alex project, it will be good if you can switch between the different operating modes to conserve power. We will leave this part to your own creativity.

Conclusion

Now that you have gained a better understanding of some possibilities for managing the power consumption of your RPi, Arduino Uno, and the RPLIDAR, you are more adept at designing an Alex that is power-effiicient!

Besides managing the power consumption of your Alex's hardware, you also need to consider the efficiency/complexity of your algorithms when you program your Alex. Inefficient algorithms that waste computational power may also contribute significantly to the overall power consumption.

As a responsible engineer, you shall evaluate the different design trade-offs, and come up with a balanced solution that meets the project requirements while managing Alex's power consumption the best way you can.

References / Resources:

1. Raspberry Pi Forum (https://www.raspberrypi.org/forums/viewtopic.php?t=152692)
2. How to power off Raspberry Pi 3 USB or Ethernet ports (http://embeddedapocalypse.blogspot.com/2016/10/how-to-power-off-raspberry-pi-3-usb-or.html)
3. ATmega328P Datasheet, Section 14