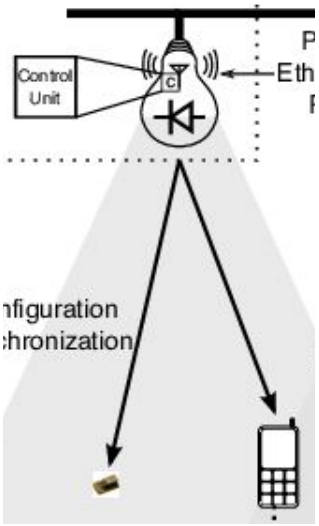
EE107 Project 1

Blinkenlight Communication

Due Mon April 15th at 10pm. Please submit your source code and result as a single .zip file on Piazza.

Overview

Communicating with visual light is one of the basic modes of wireless communication that has been around since the primitive humans. Back then, long distance communication was achieved by encoding messages in puffs of smoke, and transmitting them (naturally) by reflecting light from the off of the smoke particles. Needless to say, this form of communication had a very slow data rate (i.e., bits per minute).



In this project[1], we will be building a relatively high data rate, short-range visual light communication system from scratch. The transmitter will be a set of blinky LEDs on your embedded system; namely, the ring if 16 LEDs Charlieplexed on your TinyLED daughterboard, and the receiver will be the video camera on your smartphone. You will send 8-digit SUID (like 01234567) by blinking LEDs.

We'll provide Arduino template project for this assignment .You will need to build the drivers for each component first, then you can build the final Blinkenlight communication application. Specifically, the project will involve the following steps, each of which is described in detail in this specification:

1. **Setup your development environment -** Since this is your first project you will need to setup your system to compile and program the Atmel SAM21 (Cortex M0+) MCU on the TinyZero.
2. **Implement the LED driver** - This driver will give you a simple function to call to select which LED of the 16 Charlieplexed LEDs, if any, is illuminated.
3. **Implement the Timer driver** - This driver will provide an abstraction of the Timer peripheral so you can use it in your Blink app to control when a new set of bits is appearing on the LEDs.
4. **Implement the Blinkenlight Communication app -** This application will use the blink library to blink the light at different rates.

Resources (Important!)

For this project, the most important documents you need to understand the circuits and components are:

- Arduino template project

[1] This project is based on Aaron's CSE190 and the idea is inspired by Luxapose by Pannuto et al. The diagram above is borrowed from their work.

- TinyZero schematic
- TinyLED 16 schematic
- SAMD21 datasheet (optional)
- SAM CMSIS header files (optional)
- System Initialization and Clock Setup (optional)
- NVIC header file (optional)

The datasheet covers multiple variants of the SAM21, **we are using the SAMD21G18A package.** The CMSIS headers also come with headers for multiple variants of the SAM21, you will only need the headers in the samd21 folder. For all of the projects in the class, you will want to refer to the MCU memory map in the header file: `samd21/include/samd21g18a.h` This contains all of the definitions of the variables (pointers and structs) that you will use to access the registers to communicate with the MCU's peripherals.

During the course of the project, please post any questions you have to the class discussion forum so we can share the knowledge with all the other students.

Grading Criteria

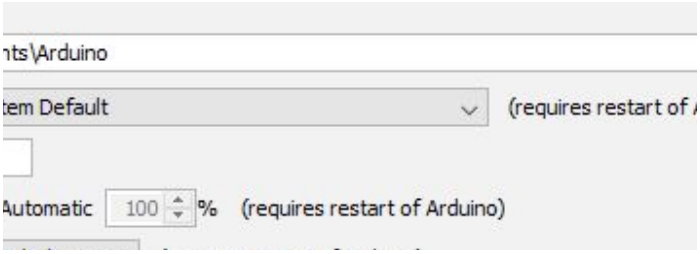
- 50% the blink application outputs your 8-digit ID correctly when flashed to a test board
- 10% the LED driver is implemented correctly
- 10% the Timer interrupt is implemented correctly
- 10% the Blink app is implemented correctly
- 20% your code is code is clear and confusing portions are well commented

Step 0: Setup the development environment

First, Download the Arduino IDE. We will use this IDE because it comes with all of the tools that need to build and deploy C and C++ applications onto the TinyZero board. Specifically, the Arduino IDE includes binaries of the ARM g++ toolchain (compiler and linker), BOSSA USB programmer, and USB drivers built for your development platform of choice (Win32/64, Linux, MacOS). It also automatically calls these tools with the correct parameters when you build and deploy your application.

In a production environment, you would be manually setting compiler arguments to achieve the best performance, and modifying the programming environment to match your deployment (e.g., you may not want to require your users to connect their device to USB in order to flash updates). Therefore, we will enable the verbose output of the compiler and programmer so there will be no magic: you will see exactly how the Arduino IDE is calling the debugger and compiler.

To enable verbose output, go to the preferences dialog (File -> Preferences) and enable the following options:



You then **must setup the Ardunio IDE to program a TinyZero board**. Please follow these instructions from the friendly folks at TinyCircuits (the designer and manufacturer of our boards).

If you are running the Arduino IDE on Linux, you may have to add your user to the `dialout` group so you do not have to run the IDE as root to access the TinyZero over USB. You can do so by running the following command in the shell.

```
$ sudo usermod -a -G dialout user
```

Step 1: Implement the LED Circle driver

The LED driver will provide a basic abstraction of the LEDs on the 16 LED daughterboard so it is easy for other drivers and libraries to control them. To do this you will need to know how to use the GPIO (general purpose input/output) pins. This microcontroller has 38 GPIO pins, but we will only be using 5 for this project.

We recommend making a copy of this template project as a starting point. It has all the MCU-specific USB code included so you can reprogram your board with the Ardunio IDE even when it is running your program. It even demonstrates how to use GPIO to turn off all of the LEDs on the 16 LED daughterboard.

To see all the other registers you can set, check out port.h in the SAM header file directory. Here are some **optional** readings:

Relevant datasheets: SAMD21 MCU datasheet **-** Chapter 23 (PORT)
Relevant header files:

- `samd21/include/component/port.h`
- `samd21/include/pio/samd21g18a.h`

Textbook: Programming Embedded Systems, Ch 2 and Ch 7

You will implement this LED function turn on the selected LED on the display: Note that you will have to implement Charlieplexing with the GPIO pins (i.e., you will need to use the high-impedance input state).

Step 2: Implement the Timer driver

The timer driver will provide an abstraction of the timer peripheral that makes it easy for other drivers and libraries to use it. You will use the timer driver in the Blink app to create a timer that fires once every *N* milliseconds. Download Adafruit zero timer library https://github.com/adafruit/Adafruit_ZeroTimer and install it as .zip library from "Sketch" -> "Include library" -> "Add .ZIP library".

Relevant documents: SAMD21 MCU datasheet

- Chapter 30 (TC)
- Chapter 14 (Clock system)
- Chapter 15 (GCLK)

Relevant libraries:

- **Adafruit zero timer library** https://github.com/adafruit/Adafruit_ZeroTimer

Textbook: Programming Embedded Systems Ch. 7 and Ch. 8

Step 3: Implement the Blink app

This is the main code that you will implement for this project. The led and timer libraries provide a convenient interface for setting up blinking LEDs.

Use the functions from steps 1 and 2 to make the 16 LED array display your 8-digit SUID, 4 digits at a time. E.g. display "0123" as "0000 0001 00010 0011" for the first **0.1** second, display "4567" in the second **0.1** second, then turn off all LEDs in the third **0.1** second and keep looping. You should use timer rather than delay() for accurate timing. Verify that you're sending correctly by recording the LEDs with you cellphone at 30/60fps.The VLC media player should be used to test your video because you can use the "e" key to advance one frame at a time during playback.

File that will be graded: Compress your **code** as well as 3 second **video recording** of the LED as 01234567.zip (replace "01234567" with your 8-digit SUID)

Common problems:

1. If you killed your MCU and it won't reflash, you can force it into bootloader mode by switching off the board, and switching it back on while holding down the boot button on the back of the board.

2. If the interrupt handler is not being called and you think it should be, it could be because you have not enabled the interrupt controller, you also may not have enabled interrupts in the CPU.

3. You can't use delay() in interrupt handle function. The watchdog may reset if do that.

4. Some LEDs may be dim so you can test in a less bright environment.