School of Advanced Technology

CST8227 Lab 3: Simple Series Circuits, Pulse-Width Modulation, Interrupt Service Routines

Lab Objectives:

1. Prototype a simple series circuit, sending digital and analog outputs to a tri-color LED.
2. Use a polling loop to read a momentary contact switch.
3. Code an interrupt service routine to read a momentary contact switch.

Required Equipment:

- Computer with Arduino IDE & Teensy extensions installed and working
- Teensy board and USB cable
- Tricolour LED
- Push-button switch
- Resistors of different values, based on calculations you have designed for.

References and Resources:

- You will need to memorize the "resistor colour code" (BBROYGBVGW, silver, gold)
- Uploaded reference textbooks: “Beginning Arduino” and “Getting Started with Arduino” (For examples and explanations of how to set pin modes and reliably read a switch)
- Layout of breadboard: electrical connectivity among rows & columns
- Spec sheet for ARMCortex-M4 and the tricolour LED.
- PartsList-Explanations file with part numbers, for referencing data sheets
- Arduino API references for digitalRead(), digitalWrite(), delay() and analogWrite() and attachInterrupt().

Task 1: Confirm maximum safe current limits for the ARMCortex-M4

1. Locate the data sheet for the Teensy 3.2 board processor, the document entitled “K20P64M72SF1.pdf” (in the “Data Sheets” link in Brightspace). The PJRC (the microcontroller board) website specifies the exact part number for the processor. Alternative means of tracking down the processor data sheet would be to use a magnifying glass to see what part number is printed on the chip, and then search the Internet for this part. You could also go to www.freescale.com (the manufacturer) and search for the keyword/part number for the processor.

2. There is a lot of information in a data sheet – you have to decide what information is important for the task. There are at least two sections of interest for now:

   i. Open the PDF document in Adobe Acrobat Reader. In the “Bookmarks” area on the left side of the document, expand the “Ratings” section, then go to “Voltage and current operating ratings” (Section 4.4). The table of data lists ***absolute maximum*** operating parameters. Two notable parameters are I_D and I_{DD}. I_D is the *maximum* allowable current for any single pin on

the ARM chip. I_{DD} is the ***maximum*** allowable current that is allowed to pass through the entire ARM chip at any one time – kind of like a "global" current variable.

ii. In the "Bookmarks" area on the left side of the document, expand the "General" section, and go to Section 5.2.3, "Voltage and current operating behaviors", Table 4. "Operating behaviors" implies voltages and currents to expect under 'normal' operating circumstances. There is a current listing for a high output voltage V_{OH} and a low output voltage V_{OL}. The magnitude of the current is the same in both cases of V_{OH} and V_{OL}, but the signs are (+) or (-). The sign refers to the direction of a current, whether or not the current is travelling into the processor from the pin, or travelling out of the pin from the processor. The electronics terminology that refer to the current direction are the 'sinking' and 'sourcing' of current. Find the current that corresponds with **high drive strength** and 2.7 V ≤ $\mathbf{V_{DD}}$ ≤ 3.6V, when $\mathbf{V_{DD}}$ = 3.3V, and use it as the targeted current when calculating a suitable size resistor in Task 2.

Task 2: Obtain Operating Parameters for Tri-colour LED

1. Locate the data sheet for the Tri-colour LED, the document entitled "599R2GBC-CC_TRI-COLOR_LED.pdf" (in the "Data Sheets" link in Brightspace).

2. The forward voltage drop (and brightness) is different for each of the three LEDs. Find the forward voltage (V_F) to expect for each colour, at the targeted operating current. Now use Ohm's Law and Kirchoff's Voltage Law, to calculate the correct size resistor, which will be used in your breadboarded circuit.

Task 3: Verify Basic Operation [Demo #1]

1. You are the circuit designer, but here are some conventions to follow:
   i. Use red for wires that are connected directly to a power source.
   ii. Use black or green for wires that are connected directly to GND.
   iii. Use different colored wires for control or communication signals.
   iv. Route wires around chips, not over.
   v. Use as short a wire as possible to complete a connection.

2. Examine the pinouts from the tri-colour LED data sheet. Take care to note how to identify the common (i.e. GND or *cathode*) pin and also the pins that correspond to each of the anode connections of the three LEDs. Install a 220Ω resistor in series with each color lead of the LED. Connect the GND lead of the tricolour LED to the GND rail. Photos and graphics of sample circuits are posted with the lab write up in Brightspace. Make certain that the 3.3V and GND pins of the teensy are connected to their respective rails on the breadboard. If you are uncertain about this step please ask. Accidently shorting the 3.3V pin to the GND pin could destroy parts of the board.

3. Modify your basic LED blinking program from Lab 2. Use the digitalWrite() function to continuously cycle each colour (R, G, B) in an ON/OFF pattern; each LED being individually illuminated, with the other two colours being OFF.

Task 4: Use Analog Outputs for Variable Brightness [Demo #2]

Many pins on the Teensy have multiple operating modes. Some pins are capable of generating "analog" outputs. These can be used in combination with delays to generate smoothly varying colours from the tri-colour LED.

1. Consult the pinout card you received with the Teensy to locate the pins marked "PWM". These are the pins capable of generating an "analog" output.

2. Rewire your circuit, if necessary, so that the three colours are driven by PWM capable pins. (In your code in the previous task, did you hard-code your pin numbers? If you did, that was a bad plan! Use meaningful variable names for pins such as int greenLED = 5; in this way, when there are pin re-assignments, updating code is minimized).

3. Modify your program to use the analogWrite() function with values from 0-255 to control each LED. Verify your understanding of this function by activating one LED at a time. A repetition structure that uses the counter to serve as the argument for the duty cycle in the analogWrite() function is useful.

4. All colours are made from some proportion of the three primary colours, red, green and blue. Modify your program to create different colours by combining red, green, and blue in different proportions of PWM. For precise colour replication, consult an HTML hexadecimal colour reference source of your choice. If you recall how HTML colours are made, each RGB portion of the colour wheel is represented as a one byte hexadecimal number from $(00)_{16}$ to $(FF)_{16}$ which is the equivalent of $(0)_{10}$ to $(255)_{10}$. Use the analogWrite() function to create a smooth transition (over some significant fraction of a second) between each of the colour combinations. When you are designing your code, think about where the delay() functions should be located. Should they be located between each change in the analogWrite() value or should you be varying only one value at a time or multiple values at a time?

4. Throw out your lava lamp and show off your creation to your friends. Demo this to the lab instructor.

Task 5: Reading from a Push-Button Switch (PBS) to Control Colour-cycling [Demo #3]

The goal is to start with automatic colour-cycling, press the PBS to "freeze" the light, and then change to a "manual" mode where there is a single change (jumping) from one colour to another on the second and each subsequent press of the BPS for 10 clicks.

1. Check the textbooks or Internet for examples of using a push-button switch (PBS)
   - Beginning Arduino, page 38, Project 4 – Interactive Traffic lights
   - Getting Started with Arduino, page 42, using a pushbutton to control the LED.

2. Note the use in both examples of a "pull-down" resistor to make sure the input pin is **not** affected by stray voltage (e.g. static electricity). It is also possible to arrange the resistor in a "pull-up" configuration.

3. Modify your code to check the state of the push-button switch (PBS). The first time the PBS is pressed, “freeze” the current colour (i.e. call delay() function). On the second and subsequent presses, jump to a new (random) colour and freeze on the new colour. Reset the behaviour after 10 clicks of the PBS. Demo this to the lab instructor.

Task 6: Implement an Interrupt [Demo #4]

1. Repeat Task 5, but this time implement an ISR (interrupt service routine). Demo this to the lab instructor.

Summary of Demos:

Complete the following demos: [2 marks each item]

i. basic Operation (Demo#1)
ii. variation of brightness using PWM (Demo#2)
iii. reading from a switch to control cycling (Demo#3)
iv. implement an interrupt service routine (ISR) to react to the push-button switch (Demo#4)

Deliverable:

Upload the following to Brightspace LMS before the due date: [2 marks each item]

i. Use your mobile phone to take a static photo (no video please) of your circuit, as demo’ed in Task 5 or Task 6, featuring a push-button switch (PBS).

Reference Photo:

Compare your photo to mine: