

CMPE-460
Interface & Digital Electronics
Laboratory Manual

Department of Computer Engineering
Rochester Institute of Technology

Contents

Submission Guidelines	iii
Timeline & Resources	v
1 Intro to K64F GPIO and Keil MicroVision	1
2 UART Driver	9
3 Characterization of OPB745	13
4 UART Over Bluetooth	21
5 K64 Timers, Interrupts, and Analog-to-Digital Converter	27
6 Motor Control	41
7 Filter Design and Simulation	51
8 Heartrate Monitor	55
A PSpice Parametric AC Sweep	63

Submission Guidelines

These instructions apply to all submissions in the course

1. Everything must be in a PDF, except code
2. You must include *all* of the required items for full credit, make it look professional
3. Everything must be typed and proofread. *NOTHING HANDWRITTEN WILL BE ACCEPTED FOR POINTS*
4. Submit all code to MyCourses
5. Submit grading sheets to MyCourses appended to the back of the report or worksheet for that exercise (scan it in). They must be signed by the lab instructor or a TA to get any credit on the exercise
6. Submit the worksheets and reports to MyCourses
7. All entries not submitted by the beginning of the next lab are considered late, unless otherwise specified
8. *DO NOT ZIP YOUR FILES.*
9. If you have any questions, email *ALL* TAs for your section

Timeline & Resources

The schedule for the labs is as follows:

Week	Lab	
1	Lab 1	Intro to K64F and Keil MicroVision
2	Lab 2	UART Driver
3	Lab 3	Characterization of OPB745
4	Lab 4	Instrumentation Amplifier and Wheatstone Bridge
5	Lab 5	K64 Timers, Interrupts, and Analog-To-Digital Converter
6	Lab 5	K64 Timers, Interrupts, and Analog-To-Digital Converter
7	Lab 6	Motor Control
8	Lab 6	Motor Control
9	Lab 7	Filter Design and Simulation
10	Lab 8	Heartrate Monitor
11	Lab 8	Heartrate Monitor
12	Lab 8	Heartrate Monitor
13		
14		

This GitLab repository contains all resources you will need for the labs:

https://kgcoe-git.rit.edu/CMPE_460/CMPE_460_Files

Lab 1 Intro to K64F GPIO and Keil MicroVision

Introduction

The Kinetis K64F from Freescale Semiconductors belongs to a family of low-power 32-bit MCUs based on the ARM[®] Cortex-M4 core. Its evaluation platform has a 120 MHz version of the MCU, with 1 MB flash memory and 256 KB RAM. Interfaces and hardware include an RGB LED, accelerometer and magnetometer, two user push buttons, an ADC and DAC, I2C, SPI, Ethernet, and the ability to connect to secondary modules. Such add-ons include an RF module, and a Bluetooth module, one of which will be used later in the course. The K64F is considered an ideal development platform for “Internet of Things” development, and is currently the most powerful Freedom Board available.

Files

https://kgcoe-git.rit.edu/CMPE_460/CMPE_460_Files/tree/master/Lab_1

Objectives

In this exercise, you are to familiarize yourself with the FRDM-K64F and Keil uVision IDE by writing code to utilize the K64 GPIO ports. The GPIOs will be used to control three LEDs using input from one pushbutton. The information required to complete this lab can be found in the K64 reference manual and other provided documents. Reading through the following material is highly recommended.

Relevant Chapters from the K64F Reference Manual

- Chapter 10: Signal Multiplexing and Signal Descriptions
- Chapter 11: Port Control and Interrupts (PORT)
- Chapter 12: System Integration Module (SIM)
- Chapter 55: General Purpose Input/Output (GPIO)

Materials

- K64F Board (and Micro USB cable)

FRDM-K64F Setup


- There are two USB connectors - use the one labeled ‘SDA USB’, next to the ‘Reset’ button.
 - The USB connectors are somewhat fragile and can easily be broken off, please take extra care when handling the board with the USB cable plugged in.
- Update your board’s firmware by doing the following:
 1. Plugging in the K64F board

2. Opening the folder to view the files and double-click on 'MBED'
3. If 'MBED' does not appear:
 - (a) Download the 0226_k20dx128_k64f.0x5000.bin file from the MyCourses under Labs → NXP Board Firmware
 - (b) Unplug the K64F
 - (c) Hold down the 'Reset' button while plugging the board back in
 - (d) Open the folder to view the files in 'BOOTLOADER'
 - (e) Drag and drop the file downloaded earlier into the folder
 - (f) Unplug the K64F and plug it in again, this time not holding 'Reset'
 - (g) Open the folder to view the files and double-click on 'MBED'
4. Unplug the board

Notes on Keil μ Vision

- The Keil software is fully installed on the Freescale/NXP Lab Computers you will be using.
 - You may download and run Keil from your personal computer if you would like to work on exercises outside of the lab. Use this to your advantage and start the exercises early. All exercises will be within the size limit allowable by the version of Keil on the lab computers.
1. On your personal machine, update Keil to version 5.13, which can be found at <http://www2.keil.com/mdk5/>
 2. On your personal machine, update the Windows serial driver, which can be found at <https://developer.mbed.org/handbook/Windows-serial-configuration>

Create a Project and Run

1. Start Keil μ Vision V5.13 (Should be in the Start Menu → All Programs)
2. Click Project → New μ Vision Project, then navigate to your personal storage device. Storing code on the lab machines is prohibited. Create a new folder on your storage device. Enter the folder, type in the name of your μ Vision project, and hit OK.
3. To select the target device, expand NXP → expand K60 Series → select MK64FN1M0xxx12, and hit OK. If this does not show up as an option:
 1. Close the Project Wizard
 2. Click this icon:  in the top bar
 3. On the left side of the Pack Installer Wizard navigate to NXP → K60 Series → MK64FN1M0xxx12
 4. Once that has been clicked the Device Specific packages should appear on the right pane
 5. Install all of the device specific packages, wait until they've all completed (you may need to click through some dialogs), and return to creating a project.
4. A window pops up to manage the runtime environment. Expand "CMSIS", Select CORE check box. Expand "Device", Select Startup check box and click on OK.

5. In the Project window on the left, expand Target1 and right click on Source Group1. Add a new item to Source Group 1 by right clicking and selecting “Add New Item to Group”.
6. Select a C file, and type in the name of your source file. Click Add.
7. Copy and paste the source code stub provided on in the git repo linked in the Files section into the C file.
 - (a) To log into the git repo click the link provided in the Files section
 - (b) Log in using your RIT username and password
8. Build the code by clicking on the Build button (F7) in the upper left. There should be 0 warnings and 0 errors. If this is not the case, check your configuration.
9. Plug in the K64F using the USB port labeled SDA next to the reset button.
10. Set up the K64F for debug by clicking the magic wand button to the right of the “Target 1” field. It will have a pop-up label of “options for target” when you scroll over it.
11. Click on “Debug” tab. Select the radio button “Use”, and then select “CMSIS-DAP Debugger”. Click on “Settings” to the right of “CMSIS-DAP Debugger”.
12. Using the “Port” drop-down menu on the left hand side of the window, select “SW”. At this point, in the SWDevice window, under device name, you need to see “ARM Core Sight SW-DP”. Hit OK twice to close both windows.
 - (a) If not, make sure all embed windows are closed. Try again once they are.
 - (b) Unplug and plug the USB cable back in with the target window closed and try again.

Use GPIO for Push-Button Input and LED Control

1. Return to the C file and add appropriate comments at the top of the file: exercise name, your name, and a short description.
2. All the signals used in the laboratory experiment will be done through Port B, Port C and Port E as described below. You will be using these ports to toggle and light up three LEDs and enable a push button.
 - (a) SIM_SCGCx are the registers that control all the internal clock gates on the K64, which are by default disabled to save power. They must be manually enabled for use.
 - (b) Look up the System Clock Gating Control Register 5 (SIM_SCGC5) in Chapter 12 in the reference manual on how to enable and disable clocks. This register specifically handles all the clock gates for the specific ports to be used with a few other features that you should take some time to investigate in the reference manual.
 - For example `SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK; // Enables Clock on PORTB.`
 - (c) Enable clocks on Port B, Port C, and Port E as specified in the code.
 - Use the masks for the SIM_SCGC5 for each port, which is located in the header file: “MK64F12.h”.
 - (d) If you access the peripheral registers before enabling their corresponding Clocks, the MCU will throw a hard fault, and become non responsive. This should be visible in the debugger.

3. Refer to Chapters 10 & 13 in the FRDM-K64F User Manual to find out what pins the three built-in LEDs and one built-in push button are connected to. The pushbutton we will be working with is SW2 and is connected to PTC6. The LEDs are referenced as Blue, Red and Green and are accessible through PTB21, PTB22, and PTE26. The K64 Documentation can be found in MyCourses under Reference → FRDMK64UG.
4. The pin naming convention is as follows: “PTA0 = Port A, Pin 0”. Copy down the pin numbers for the LEDs and push button (also make sure these are documented in your code):

Table 1.1: Table for Recording Ports & Pins for LEDs

	Port	Pin	Name
Red LED			
Green LED			
Blue LED			
Switch 2			

5. Enable the LED and button control signals by setting the Pin Control Register (PORTx_PCRn) for each LED pin to GPIO Pin Mux Control.
 - (a) Assign the LED pins found above to the value PORT_PCR_MUX(1), which enables GPIO mode.
 - (b) For example, “PORTA_PCR11 = PORT_PCR_MUX(1);” wakes up the 11th bit of PORT A to work in GPIO mode.
6. Set the bits of each Port to input/output mode according to the requirements of the lab. If set to output, the pin can then be set, cleared, or toggled. If set to input, the pin can be read. (Chapter 55 of K64 reference manual).
 - (a) Configure GPIO pins for Input or Output using the GPIO Data Direction Register (PDDR).
 - (b) Make sure to use the correct Port for each pin.
 - (c) To load data into the direction register, use a simple shift left operation for each of the desired port pins. [e.g. GPIOA_PDDR = (1 << n), where n is the LED pin from PORTA that we want to set to output. You can OR as many of these together in one line as GPIOA_PDDR = (1 << 11) — (1 << 12) to set pins 11 and 12 as output.]
7. Write code to turn on the LEDs and to control them using the pushbutton. (Chapter 55 of K64 reference manual).
 - (a) Drive the output to the LEDs high by writing a ‘1’ to the GPIOx_PDOR register for each LED, initializing them to the ‘off’ state. Keep in mind that the LEDs are active low.
 - (b) Note, when controlling the GPIO, we will be using five commands:
 - (i) GPIOx_PDOR = <value>; this sets the entire register to <value>
 - (ii) GPIOx_PSOR = <value>; this sets all bits in the register for which <value> has a ‘1’ bit (in binary)
 - (iii) GPIOx_PCOR = <value>; this clears all bits in the register for which <value> has a ‘1’ bit (in binary)

- (iv) `GPIOx_PTOR = <value>;` this toggles all bits in the register for which `<value>` has a '1' bit (in binary)
 - (v) `if (GPIOx_PDIR == <value>);` use this to test an input register. For example, to test if PORTA11 is high, type `if((GPIOA_PDIR & (1 << 11)) == 0)`
 - (c) To download and run the code:
 - (i) Hit the 'Build' button (F7)
 - (ii) Hit the 'Load' button (F8)
 - If you get an error that looks like: "Cannot access memory", follow the instructions at this link to flash the Hello World binary to your board. <https://os.mbed.com/platforms/FRDM-K64F/#flash-a-project-binary>
 - (iii) Hit the 'Reset' button on the K64F. The reset button is the button next to the USB cable. (Note, if you are debugging, when you first step into the code, the board is automatically reset for you.)
 - (d) Create an infinite loop that will make the LED yellow whenever the SW2 pushbutton is pressed. The SW2 pushbutton is labeled as SW2 on the board.
 - (i) Note: to get yellow, the red and green LED's are on, and the blue LED is off.
 - (ii) To make the yellow LED stay on, add a loop that increments a long variable 2,000,000 times.
 - There should be nothing inside this for loop.
8. Modify the code such that (using the loop delay from above to keep each color on):
- (a) The first time you hit SW2 the LED cycles from red, then green, then blue, then turns off.
 - (b) The next time you hit the SW2 button, the LED cycles from cyan, then magenta, then yellow, then turns off.
 - (c) The next time you hit the SW2 button, it displays white then turns off.
 - (d) The next time you hit the SW2 button, the pattern restarts at red, then green, then blue then turns off.

Lab Worksheet

Please, submit one lab worksheet per individual. The Lab worksheet for this week should include:

1. Title of the Lab, your name, date the lab was performed, course name, TA's name and instructor's name.
2. Two line description of lab
3. List out what each of the below GPIO commands do:
 - `GPIOx_PODR = <value>`
 - `GPIOx_PSOR = <value>`
 - `GPIOx_PCOR = <value>`
 - `GPIOx_PTOR = <value>`
 - `if(GPIOx_PDIR == <value>)`
4. Sign off sheet

Submit the **completed** worksheet and code to the MyCourses dropbox by the beginning of the next lab. (Note: Worksheets and reports will always be due by the beginning of the lab after it was scheduled to be completed, unless specified otherwise)

Exercise 1: Intro to K64F GPIO and Keil MicroVision

Student's Name:_____

Section:_____

Demo		Point Value	Points Earned	Date
Demo	Port & Pin Table	10		
	Yellow LED Loop	25		
	LED Cycle	40		

To receive any grading credit students must earn points for both the demonstration and the report.

Exercise 1: Intro to K64F GPIO and Keil MicroVision

Report		Point Value	Points Earned	Comments
Report	Lab Description	10		
	Question	15		
Total for prelab, demo, and report		100		

Lab 2 UART Driver

Introduction

For this exercise, you will be writing a UART driver for the K64. UART is short for Universal Asynchronous Receiver Transmitter, and it functions to translate data between parallel and serial forms. UART is commonly used in conjunction with the RS-232 communication standard. Data format and transmission speeds are configurable and all electrical signaling is handled by an external driver circuit. For more information on the basics of UART, please refer to http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter

Files

https://kgcoe-git.rit.edu/CMPE_460/CMPE_460_Files/tree/master/Lab_2

Objectives

You are expected to write the methods for initialization of the UART interface, writing a character and reading a character. You will then connect the K64 to the computer through the UART-to-USB dongle provided to you for serial communication and prove that characters are being written to the terminal upon key press. For the second part of the exercise, you shall rewrite the main.c file to allow the user to enter a sentence whose maximum length is decided by the TA/Instructor when you are demonstrating this part. Characters should be displayed on the screen as they're being entered, and hitting the return key must allow the user to enter the next sentence.

Relevant References

- Chapters 10, 11, 12, and 52 of the K64 Reference Manual

Materials

- K64F Board (and Micro USB cable)

Prelab Activities

We have already learnt from Lab 1 that before working with the ports, the clock to the ports need to be enabled and the Port Control register has to be configured to the proper mode. Go through the K64 Reference Manual and figure out what needs to be done for you to access the UART0 module on the board. Write down the appropriate registers needed, as well as the ports and pins for the transmit and receive peripherals.

Project Setup

1. Create a new project in Keil μ Vision
2. Download the provided code 'lab2.template.c' from the git repo linked in the Files section and add it to your 'main.c' in the project.

3. Download the provided 'uart_template.c' from the git repo and add it as an additional file called 'uart.c' in your project

Procedure

Part 1

1. You must complete the methods `uart_init()`, `uart_putchar(char)`, `uart_getchar()`, `uart_put(char*)` in `uart.c` with the help of the below instructions . Note: when setting/clearing register bits, use the `#define` values provided by the microcontroller. For example, when enabling the clock for PORTC, use "`SIM_SCGC5 |= SIM_SCGC5_PORTC_MASK;`" instead of "`SIM_SCGC5 |= 0x800u;`". These `#define` values are found in `MK64F.h`.
2. Begin by reading Chapter 52 in the K64 Reference Manual for the general UART initialization procedure. Refer to the following steps for detail:
 - (a) Configure the port control register of the two UART0 pins for the UART module to function correctly. (pinout section 10.3 of K64 reference manual)
 - (b) Disable transmitter and receiver of the UART until the UART is properly configured with the necessary settings. This can be done by writing to the control register 2 of UART module, `UARTx_C2`, where x is the chosen UART module number.
 - (c) Use the default settings for UART module by writing 0x00 to `UARTx_C1` (the default configuration being 8 Data bits, No Parity, 1 Stop bit at 50MHz)
 - (d) Use 9600 Baud rate. Read 52.4.3 for Baud Rate Generation. Use the example provided below along with the suggested reading to understand the procedure to be followed for configuring the UART module to a specific baud rate.
 - $ubd = \text{UART module clock} - (16 * \text{baud rate})$
 - (e) Using the default system clock value for UART module clock (this value is defined as `SYS_CLOCK` in the template) and default serial communication baud rate (9600) calculate `ubd` in the above equation. This `ubd` contains the SBR values.
 - (f) Distribute this `ubd` in `UARTx_BDH` and `UARTx_BDL` bits that hold the SBR values. The baud rate fractional divisor, `BRFD`, can be used to fine tune the baud rate value and bring it closer to the desired value and is configured by writing to `BRFA` (Baud rate fine adjust) in `UARTx_C4`. `BRFD` is 1/32 of `BRFA`'s value. This aim of using `BRFD`(`BRFA`) is to counter the loss due to truncation of `ubd` into an integer. The value of `brfa` can be calculated by the formula below
 - $brfa = ((\frac{SYS_Clock * 32}{BAUD_rate * 16}) - (ubd * 32))$

Write this `brfa` value into the corresponding bits of `UARTx_C4`. After configuring the settings for the UART module, you should turn the receiver and transmitter of the UART module back on by using `UARTx_C2`.
3. The remaining methods should do the following:
 - (a) For `uart_getchar()`: Wait for a character to be received on the specified UART. You can do so by waiting until the `RDRF` bit in `UARTx_S1` is 1. Then return the register containing the character, `UARTx_D`.

- (b) For `uart_putchar()`: While the TDRE bit in the `UARTx_S1` is 0, the UART is busy. As soon as it's not busy, we can use it, and send the character to the terminal by filling the register `UARTx_D` with the 8-bit character you wish to send.
4. Using these functions, print "IDE: Lab 2 Demonstration by <Your name>" to the PuTTY terminal.
 - (a) Connect the K64 board to the computer using the USB cable provided on each computer.
 - (b) Open PuTTY
 - (c) Select the 'Serial' radio button
 - (d) The serial port can be determined by taking a look at the 'Ports' section in the Device Manager. (A simple way to bring up the device manager is to search for it in the search box after clicking on the start icon. Sometimes the Ports section may not be visible in which case you can go to "View" in the device manager and highlight "Show hidden devices" option. You don't need to be logged in as the Admin to open or see the contents of the Device Manager.) Note: use the mbed serial port.
 - (e) Set the Baud Rate to 9600 and hit the enter key to connect
 - (f) Download the code into the flash memory of K64 and start the execution by hitting the 'Reset' button.
 - (g) Demonstrate Part 1 to the TA/Instructor
 - (h) Take a screenshot of the terminal output for your lab report. Please invert the image for clarity.

Part 2

1. Create a new project and copy the existing code into the sources directory.
2. The `uart.c` file should initialize the clock, UART pins and UART interface as done in Part 1
3. Your new source file should allow the user to:
 - (a) Enter up to a maximum character length for a given sentence which is decided by the TA/instructor while giving a demonstration.
 - (b) Press return to finalize the sentence and be prompted to enter another, which means that the sentence needn't exceed the max character length for a prompt to enter a new sentence in a new line.
4. Open a terminal as before and demonstrate that the software is working correctly on the K64
5. Take a screenshot of the terminal output for your lab worksheet

Lab Worksheet

Please submit one lab worksheet per individual. The worksheet for this exercise should include:

1. Title of the Exercise, your name, date the lab was performed, course name, TA's name and instructor's name.
2. Screen shots of terminal from Part 1 and Part 2
3. Sign off – A TA needs to sign off on Part 1 and Part 2. Include two lines that say:
_____Part 1 of Exercise 2 is working correctly
_____Part 2 of Exercise 2 is working correctly

Submit the **completed** worksheet and all code to the MyCourses dropbox by the beginning of the next Lab time.

Lab 3 Characterization of OPB745

Introduction

Optical measurement and isolation has the advantage that it does not require physical contact with a system, and thus is much less likely to disturb the system being measured. For example, it is possible to use optical measurements to measure the distance between two objects, eliminating the need for a physical “measuring tape”. Opto-isolators are useful when physical isolation between two systems is required. Isolation may be required when:

1. An application requires safe, accurate measurement of DC and low frequency voltage or current in the presence of high common-mode voltage.
2. It is necessary to receive/transmit signals at high impedance in noisy environments.
3. An application must perform a measurement, and, for safety reasons, must guarantee that voltage and current leakage will remain below certain thresholds.

Opto-isolators are a common element found in medical electronic equipment, power plants, automated test equipment, industrial controls, and portable field equipment.

Objectives

For this exercise, you are to characterize the OPB745 photo-transducer included in your lab kit, testing both its ability to accurately measure the distance to a reflecting surface and its ability to perform as an opto-isolator. Several components of this lab experiment are open-ended, and can be completed as desired. This exercise will test your ability to develop and define the required elements of design.

Materials

- From Kit:
 - OPB745 and associated extension wires
 - SN7406N Inverter
 - 74LS14N Inverter

- Extra:

Note: If you cannot get a ride to purchase these materials, Rick Tolleson may be able to sign out a pre-built measurement device for this lab

- About a foot of opaque PVC piping
- About a foot of opaque PVC (black or white, can also use black sprinkler) pipe with a slightly smaller diameter
- Reflective surface material (e.g. reflective tape or heavy-duty aluminum foil)
- A round piece of thick cardboard with duct tape or dark foam
- A printout of a millimeter ruler

Prelab Activities

Warning: Extensive preparation is required for this exercise. Please, give yourself plenty of time.

- Design and build a test environment for the OPB745 that will isolate the sensor from external lighting and will allow you to accurately adjust the distance between a reflective surface and the OPB745's sensor head. Examples of the isolation chamber are shown in Figures 3.1 and 3.2. When designing the chamber, keep in mind that:
 1. It should provide excellent isolation from all ambient light sources. Precise measurement and construction is crucial to providing an acceptable characterization of the sensor. For this, use of an opaque material such as PVC piping is suggested. About a foot of the outer shell is required. All cuts on the PVC pipe need to be square (right angles to long edge of pipe). If you don't have access to the tools to cut the pipe, you can get help at the Machine Shop or Construct Lab.
 - Note: To fully block the light, use either a black outer tube, or cover the outside in opaque black tape.
 2. It should allow the user to adjust the distance between OPB745 and reflecting surface. Use of a smaller diameter, opaque PVC pipe is highly recommended. The outer and inner pipes should fit snugly but allow for movement. Use lubricant sparingly if desired, ensuring that lubricant does not reach the reflective surface at the end of the inner pipe.
 3. It should provide the ability to accurately measure the distance between the OPB745's sensor head and a reflective surface with an error margin of $\pm 0.5mm$. A round piece of thick cardboard with duct tape or dark foam cut in a round can be used to hold the opto-isolator in place and prevent light infiltration. Cut a slit in the center of the round foam to allow for passage of the OPB745. Attach a printout of a millimeter ruler to the inner PVC piping that will allow for the user to measure the displacement of the reflective surface.
 4. Attach a reflective surface on one end of the inner pipe. A piece of heavy-duty aluminum foil is sufficient. When applying the reflective surface to the end of the inner tube, ensure that it is flat so the tubes can still glide in and out easily. When adjusting the distance using the inner PVC tube, take care to not push the surface too far into the chamber, as the case of the opto-isolator may damage the surface of the aluminum, or change its shape.

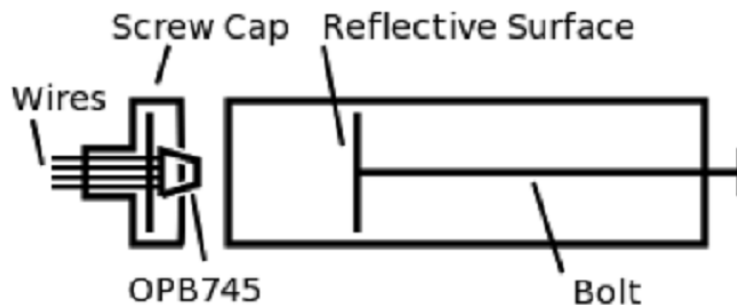


Figure 3.1: Optoisolator Diagram

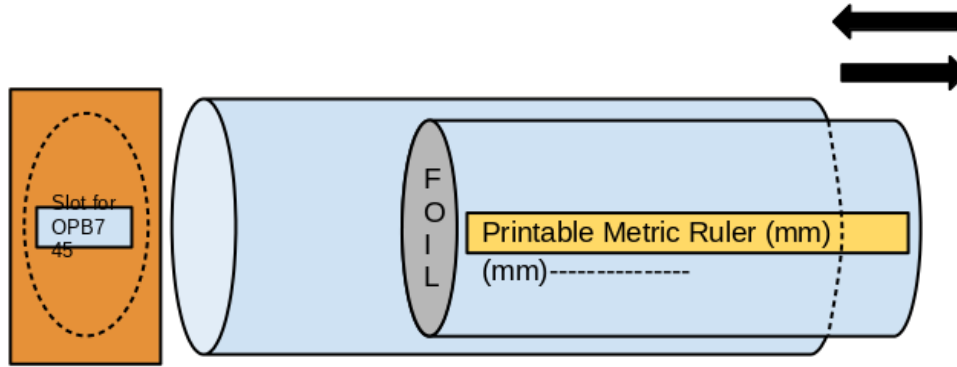


Figure 3.2: Optoisolator Diagram

- Calculate the resistor value, R_F , for Parts A and B. Use an input voltage of 5V and allow for 40mA of current to pass through R_F to the input diode. The resistor value for Part B will not be the same as you need to consider the voltage drop across the 7406 inverter in addition to that of the diode. **Show your resistor values to the Lab Instructor or TA before continuing the exercise.**

Procedure

Part 1

1. Connect the provided extension wires to the OPB745's anode, cathode, emitter and collector pins to provide enough slack for connection to a breadboard. For pin specifics, refer to the datasheet, which can be found in the MyCourses.
2. Using the isolation chamber constructed in the pre-lab portion of the exercise, the attached OPB745, and the R_F value from pre-lab, construct the circuit shown in Figure 3.3 on a breadboard.

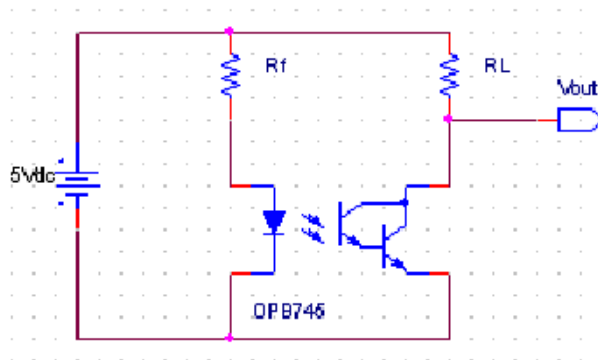


Figure 3.3: OPB745 Circuit for Part 1

3. Apply power to the circuit as specified. Varying the distance from the opto-isolator to the reflective surface by changing the position of the inner PVC pipe, record the measurements in

Table 3.1 for an RL value of $10\text{ k}\Omega$. Repeat for an RL value of $20\text{ k}\Omega$. Measure voltage across the photo-transistor and current across the RL resistor.

Table 3.1: Example Table for Recording OPB745 Measurements

	$R_{L1} = 10\text{ k}\Omega$		$R_{L2} = 20\text{ k}\Omega$	
Distance (mm)	$V_{out}(V)$	$I_{RL}(mA)$	$V_{out}(V)$	$I_{RL}(mA)$
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
20				
25				
30				
35				
40				
45				
50				

- Graph the current vs. distance and voltage vs. distance results for each resistor. The resulting current graph should resemble the one found in the OPB745 data sheet entitled, “Normalized Collector Current vs. Object Distance”.
- Demonstrate the resulting graphs to a TA or Lab Instructor.

Part 2

Note: The measurement distance (position of tube) to use in Part B is the optimal distance from Part A

Perform the following twice, once with $R_L = 10\text{ k}\Omega$ and once with $R_L = 20\text{ k}\Omega$

- Construct the circuit shown in Figure 3.4 using the test chamber and attached OPB745, a 7406 inverter package, a 74LS14 inverter package, and the resistor value, R_F calculated in the pre-lab.

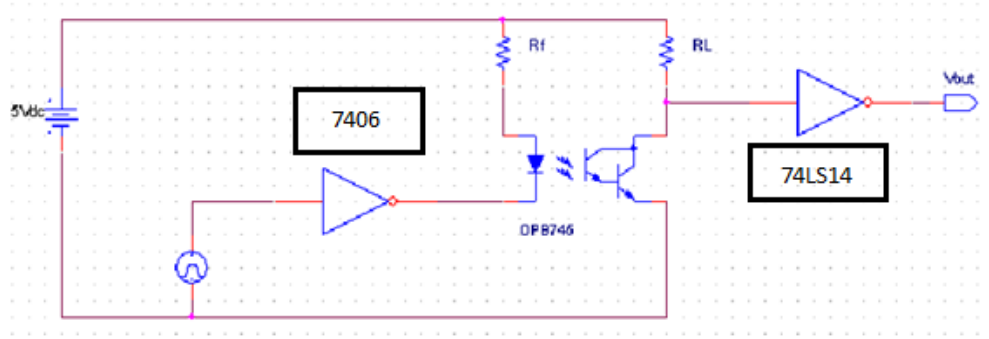


Figure 3.4: OPB745 Circuit for Part 2

2. Apply $5V_{DC}$ as indicated, and attach a waveform generator to the input end of the 7406 inverter. Generate a 50% duty cycle square wave at 100Hz. (Make sure to use the output from the TTL logic port)
3. Take voltage readings with the oscilloscope probe placed between the waveform generator and the 7406 inverter. This will provide a V_{in} value. Another probe should be placed after the 74LS14 inverter to provide a V_{out} value.
4. Increment the frequency by 100Hz at a time to find the maximum frequency at which the output wave retains its square wave characteristics. You will have to readjust the view on the oscilloscope.
5. Take an oscilloscope screen capture of the resulting waveform, and save it to USB drive. Be sure to include the frequency measurements of both channels in the screenshot.
 - Press the 'Save' button on the side, under 'File'
 - Insert a USB drive
 - Select 'Save' in the sub-menu
 - Select the desired format and specify a filename
 - Hit 'Save'
6. Demonstrate results for the Instructor or TA. Be able to explain to this person in your own words what is being accomplished in this portion of the experiment.

Questions

In the lab we do not use a 7406 inverter on the output; instead we use the 74LS14 with Schmitt trigger. Why do we need to do this, and what is the difference between the two?

Lab Report

Follow the Lab Report Guidelines on MyCourses under ‘Reference Material’

1. Title Page
2. Short Abstract
3. Design Methodology
4. Results and Analysis
 - (a) A completed version of Table 3.1
 - (b) A graph of the data recorded while measuring the relationship between distance and output voltage
 - (c) A graph of the data recorded while measuring the relationship between distance and current
 - (d) Oscilloscope captures taken while finding f_{max} , showing both the input and output of your circuit. These captures must include frequency measurements of both channels.
5. Questions
6. Conclusion
7. Completed Lab 3 Sign Off Sheet

Exercise 3: Characterization of OPB745

Student's Name:_____ Section:_____

PreLab		Point Value	Points Earned	Comments
PreLab	Optoisolator	10		
	Resistor Calculations	10		

Demo		Point Value	Points Earned	Date
Demo	Part A	20		
	Part B	20		

To receive any grading credit students must earn points for both the demonstration and the report.

Exercise 3: Characterization of OPB745

Report		Point Value	Points Earned	Comments
Abstract		5		
Design Methodology	Discussion	5		
Results and Analysis	Completed Table	5		
	Distance v Vout Graph	5		
	Distance v Current Graph	5		
	Oscilloscope Captures	5		
Conclusion		5		
Question		5		
Total for prelab, demo, and report		100		

Lab 4 UART Over Bluetooth

Introduction

Wireless communication is becoming more prominent as time goes on, and one of those methods of communication is BLE, or Bluetooth Low Energy, which is a more energy-efficient version of standard Bluetooth. Bluetooth is a very versatile method of communication, as it can transmit data wirelessly with minimal loss of fidelity while remaining rather power-efficient. It is also, in many cases, rather simple to set up, especially when compared to certain other methods of wireless communication.

Objective

In this exercise you are to familiarize yourself with Bluetooth controllers and communication over Bluetooth using UART. You will be controlling the color of the RGB LED on the K64F using Bluetooth, as well as setting up a basic “chatroom” between your computer and a phone.

Relevant References

- Chapters 10, 11, 12, and 52 of the K64F Reference Manual

Materials

- K64F Board (and Micro USB cable)
- H-10 Bluetooth Module
- A Smartphone (you will need to install a couple of apps)

Prelab Activities

Create a new UART controller file (you can start with the program you wrote in Lab 2) which uses UART3. Please refer to the K64F Reference Manual for the appropriate GPIO pins and be sure to initialize all aspects of the driver correctly. Make sure to take note of which pins are the transmit and receive pins for UART3. You will need to use these functions in conjunction with your original UART functions for a later part of this exercise, so make sure to change the function names accordingly.

Procedure

Pairing the Bluetooth Module to Your Phone

To connect the HM-10 to your K64F, use the same jumper wires that were used in Lab3. Connect the Vcc of the HM-10 to 3.3V on the board. Connect GND to GND, TX to the UART3 RX that you found in the prelab, and RX to UART3 TX.

BEFORE PAIRING

Each HM-10 will come having the same Bluetooth name, making it difficult for a full lab to connect to each user's correct device. To rename the device do the following:

- In the UART3_init function which was created in the prelab, after the UART3 has been initialized, send the string "AT+NAME `unique_name`;"

Android

1. Download the app "Serial Bluetooth Terminal" from the Google Play Store
2. Connect your HM-10 to your K64F, being sure to use the correct pins for interfacing with UART3
3. You should see a red light on the Bluetooth module flashing once it has power
4. Open the "Serial Bluetooth Terminal" app on your phone and go to "Devices" by tapping the bars in the top-left corner of the screen, and then tapping "Devices"
5. Make sure you have "Bluetooth LE" selected and the HM-10 should show up in the list of available devices, again as "HMSoft"
6. Tap the HM-10 in the list. The bar next to it should turn green
7. Go back to the terminal by tapping the bars in the top-left corner of the screen and then selecting "Terminal"
8. To connect to the HM-10, tap on the icon in the top bar that looks like two wires facing into each other
9. You should see some scrolling messages about trying to connect to the HM-10. Assuming this was done right, you should connect rather quickly

iPhone

1. Download the app "HM10 Bluetooth Serial Lite" from the App Store
2. Connect your HM-10 to your K64F, being sure to use the correct pins for interfacing with UART3
3. You should see a red light on the Bluetooth module flashing once it has power
4. In the app tap "Settings" in the top-right of the screen
5. Set the "Append To Message" to "Both NL & CR" and tap "Done" in the top-left
6. Once back at the main terminal, tap "Connect" in the top-left
7. Wait for the list to populate. You should see a device called "HMSoft", tap that and wait for it to connect

Part 1 - LED Control

1. Once connected to your HM-10 through your phone, use your `uart_put` and `uart_getchar` methods from the prelab to try sending and receiving a message, the same was as was done in Lab 2. You may need to send a blank line before you get the prompt on your phone's serial terminal.
2. Once you have confirmed that you can send to your K64F using the Bluetooth controller, use what you did in Labs 1 and 2 to control the color of the RGB LED on the K64F.
 - The LED should change color based on the code it is being sent:
 - 0 → OFF
 - 1 → RED
 - 2 → BLUE
 - 3 → GREEN
3. Once this is done, show your TA or Lab Instructor and move on the Part 2

Part 2 - Computer to Phone “Chatroom”

1. Create a new Project called “uart_chatroom”
2. In this project make a `uart.c` and `uart.h` file which allows you to use both UART0 and UART3
3. Create a program which allows you to get messages of up to a certain length from your phone on your computer, and vice versa
 - This program will not necessarily need to function like a more realistic chat client. Since your code will be running sequentially, you will only be able to send and receive from each device when that specific set of UART commands is run in your main function. As a result of this, please set your code up as follows:
 - (a) Send from phone to computer
 - (b) Send from computer to phone
 - (c) Repeat

Questions

Research independently to find the answers to the following questions.

- What is a Bluetooth device's Service UUID for? What is the Service UUID for the HM-10?
- What are a Bluetooth device's Read and Write Characteristics for? What are they for the HM-10?

Lab Worksheet

Please submit one worksheet per person. The worksheet for this week should include:

1. Title of the Lab, your name, date the lab was performed, course name, TA's name and instructor's name.
2. Two line description of lab
3. High level description of how code works
4. Answer to in-lab questions
5. Attached the completed Lab Sign Off Sheet to the back of the worksheet

Exercise 4: UART Over Bluetooth

Student's Name:_____ Section:_____

PreLab		Point Value	Points Earned	Comments
PreLab	UART3 Code	20		

Demo		Point Value	Points Earned	Date
Demo	Part 1	30		
	Part 2	30		

To receive any grading credit students must earn points for both the demonstration and the report.

Exercise 4: UART Over Bluetooth

Report		Point Value	Points Earned	Comments
Lab Description		5		
Code Description		5		
Question	Question 1	5		
	Question 2	5		
Total for prelab, demo, and report		100		

Lab 5 K64 Timers, Interrupts, and Analog-to-Digital Converter

Introduction

Interrupts

Interrupts are important events that must be recognized and serviced by any program. The microcontroller has hardware and software features to allow these important events. Interrupts are an important asset when writing high performance applications. You may use them in embedded systems to synchronize the microcontroller with slower I/O (such as serial interface), and in some applications with a real-time operating system, interrupts are used to carry out all processing tasks. The ARM Cortex-M Series contains a Nested Vector Interrupt Controller (NVIC). The NVIC in the Cortex-M4 supports up to 240 interrupts with 256 levels of priority each. The NVIC configuration in the K64 only supports up to 16 priority levels and uses only 95 interrupts. The K64's NVIC information can be found in Chapter 3.2.2 (Page 73) in the Kinetis K64 Reference Manual. Serial communication is used when data is sent in sequence, one bit at a time, over a specified channel or bus. It is used in extended communication sessions and computer networks due to its improved signal integrity and increasing transmission speeds. One could continuously poll to see if new bytes are ready to be transferred, or use interrupts whenever a byte is ready. The K64 has a serial port (RS-232) available to communicate using hardware interrupts.

Timers

Many embedded systems require precise timing to generate events at certain times. To generate events at the correct time, one of the features included in microcontrollers are timers. Timers can be used to generate interrupts at specific intervals or even be used as pulse counters if supported by the microcontroller. Most timers are based on a free-running counter that increments every clock cycle. When a timer has been started, it will continue running until it is stopped or reaches a maximum value (which can be programmed). Interrupts can be thrown when the timer has reached this value. Counter registers range from 32-bits down to 16-bits depending on the module. Because of the limited bits and high clock speeds, timers often include pre-scalars to slow down the clock, increasing the length of time you can count at the expense of resolution.

The Kinetis K64 contains several timer modules, each with their own set of features. The first timer module is the ARM System Tick Timer. The ARM System Tick Timer is present in many or all ARM cores. It features a 24-bit countdown timer and is often used in Real-time Operating Systems (RTOS) because of its high priority.

The Programmable Delay Block (PDB) is designed to be used alongside the analog features in the K64. It can be used to automatically trigger interrupts related to Analog-to-Digital (ADC) conversions, and Digital-to-Analog (DAC) and Comparator (CMP) updates. This timer is ideal for sampling and generating analog signals that require precise timing. The K64 has only one PDB module. The counter in the PDB counts up and will continue counting until it is explicitly stopped. All of the trigger outputs in the PDB share the same time base (they all compare their values to the PDB Counter).

The FlexTimer Module (FTM) is a very simple timer built upon the one used in the HCS08 (NXP's 8 bit microcontrollers). It can be used for input capture and output compare as well as generating PWM signals. This timer is ideal for motor control and timing associated with external inputs and outputs.

The Periodic Interrupt Timer (PIT) is a simple timer that can be used to generate interrupts and trigger Direct Memory Access (DMA) channels.

The Low Power Timer (LPTMR) is a simple timer that can operate through most system reset events. It can be used as a time of day counter or as a pulse counter.

The Real Time Clock (RTC) is a timer used to keep track of seconds. It is powered by an independent power supply (usually a coin cell) and can continue running while the microcontroller is unpowered. The K64 also includes two watchdog timers. The first is a timer for internal software such that if it overflows, the MCU will be reset. The second is an external watchdog which is designed to monitor external circuits and send an external reset if it happens to overflow.

Analog to Digital Converter

The Analog to Digital Converter (ADC) on the K64 contains many features such as differential mode, Programmable Gain Amplifier (PGA), Automatic compare function, and more. These features allow for more flexibility and helps offload software processing by implementing the features in hardware. The ADC can be configured in a number of ways to suit the application's requirements.

Digital to Analog Converter

The 12-bit Digital to Analog Converter (DAC) is a simple peripheral that can be used for various applications. The DAC features a configurable 16-word data buffer and Direct Memory Access (DMA) operations that helps reduce processor intervention. The DAC can select between two reference voltages: DACREF_1, and DACREF_2. DACREF_1 is connected to VREF_OUT which is set at 1.2V. DACREF_2 is connected to VDDA which is set at 3.3V.

ADC with Programmable Delay Block (PDB)

Instead of using a software triggered ADC conversion, the PDB can be used to trigger ADC conversions. The K64 has only one PDB with two trigger outputs, one for each ADC. Each trigger output on the K64 has two pre-trigger outputs which are used for selecting the status and control register that will be used for that conversion. Figure 5.1 shows the PDB delay registers that will send a trigger to the ADC to begin a conversion on the associated ADC status and control register. This allows easy sampling of multiple channels.

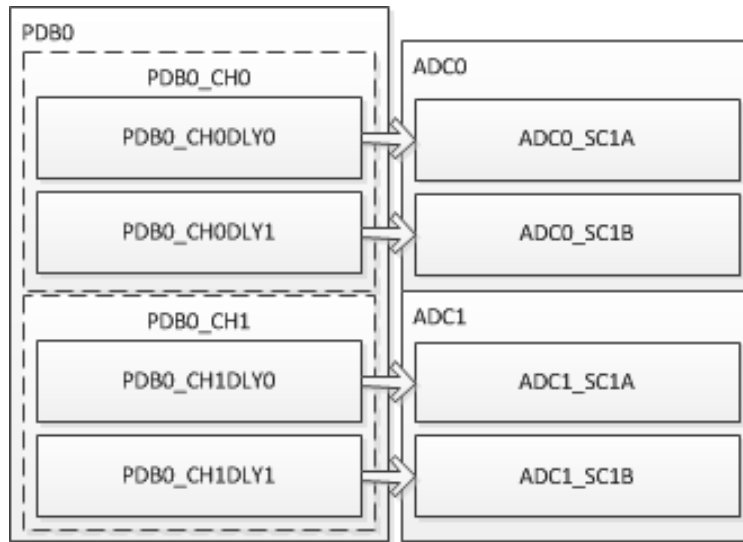


Figure 5.1: PDB Pre-Trigger Mapping

CDS Cell

A CDS Cell is a simple photo-electronic device that provides a variable resistance value to a circuit. This value decreases with increasing light and increases with decreasing light, exhibiting photoconductivity. The component is made of a high resistance semiconductor material that absorbs photons, giving electrons enough energy to jump bands, thereby conducting electricity and allowing more current to flow through a circuit.

Objectives

The objectives of this exercise are to understand how to enable and use interrupts for various modules, to use a timer to generate and measure time, to successfully digitize an analog signal that is passed through a photo-electronic device, and to read analog input from a line scan camera. In Part 1 of this exercise, you are to use the two pushbuttons on the K64 to do two things. The first button will be used to toggle an LED which will blink on or off every second using the PDB as the timer. The second button will use a timer to measure how long the button has been pressed down. After the button is released, the time in milliseconds is to be displayed in a terminal. We will be using the PDB timer to toggle the LED. Since there is only one PDB timer, we will then use the FTM to time the hold-down time for the second pushbutton. In Part 2, you will digitize an analog signal that is passed through a photo-electronic device, display binary logic to a terminal through a UART interface, and see the effects of aliasing on the data. In Part 3, you will combine your understanding of timers, interrupts, and analog to digital conversion to interface a line scan camera with your NXP car.

Files

https://kgcoe-git.rit.edu/CMPE_460/CMPE_460_Files/tree/master/Lab_5

Relevant Reference Manual Chapters

- Chapter 3.2.2: Nested Vectored Interrupt (NVIC) Configuration
- Chapter 3.7.1: 16-bit SAR ADC with PGA Configuration
- Chapter 10: Signal Multiplexing and Signal Descriptions (GPIO)
- Chapter 11: Port control and interrupts (PORT)
- Chapter 35: Analog-to-Digital Converter (ADC)
- Chapter 39: Programmable Delay Block (PDB)
- Chapter 55: General purpose input/output (GPIO)

Materials

- K64F Board and USB cable
- TMP36
- Line Scan Camera
- NXP Cup Car

Prelab Activities

Warning: Extensive preparation is required for this exercise. Please, give yourself plenty of time. Complete the following steps using registers found in the K64 Reference Manual:

- Enable the clock for the PDB0 module
- Enable the PDB0 IRQ, use `NVIC_EnableIRQ` macro
- Enable the clock for the FTM0 module
- Enable the FTM0 IRQ, use `NVIC_EnableIRQ` macro
- Configure SW3 to be interrupted on the rising edge
- Configure SW2 to be interrupted on both rising and falling edge
- Enable the clock for ADC1 module
- Enable the ADC1 IRQ, use `NVIC_EnableIRQ` macro
- Read result from ADC1 (*hint: fill in the blank*)
`unsigned short i = _____ > > 4;`
- Set DAC output value (12 bits) to variable `i` above (*hint: use DAC0, result is spread over two 8 bit registers*)

It is highly encouraged to look at the next parts of this exercise and implement the rest of the configurations for the modules we will use

Procedure

Part 1 - Timers and Interrupts

1. Create a new project in Keil μ Vision5.13.0
2. Copy contents from main_timer_template.c to your main.c, copy uart.c from Lab 2 into your project
3. Copy the LED_Init and Button_Init functions from Lab 1 to your main.c, in the function initGPIO()
4. Add initialization to SW3 in Button_init()
5. Call, the uart_init() function
6. At the end of the initGPIO() function set SW3 to cause an interrupt when pressed, and set SW2 to cause an interrupt when pressed or released (see Page 283 of the reference manual for more detail)
7. Create the isr.c and isr.h files.
 - (a) Copy isr_template.c to a new file in your project called isr.c
 - (b) Create a new header file called isr.h and add safety defines and function declarations to isr.h:

```
#ifndef ISR_H_
#define ISR_H_
void PDB0_IRQHandler(void);
void FTM0_IRQHandler(void);
void PORTA_IRQHandler(void);
void PORTC_IRQHANDLER(void);
#endif /* ifndef ISR_H_ */
```

8. Create uart.h
 - (a) Modify code below to add function declarations as appropriate for your uart code:

```
#ifndef UART_H
#define UART_H

#include <stdint.h>

void put(char *ptr_str);
void putnumU(int i);
void uart_init(void);
uint8_t uart_getchar(void);
void uart_putchar(char ch);

#endif /* ifndef UART_H */
```

9. Anytime an interrupt is triggered for the peripheral (timer, switch, etc), its IRQ Handler will be called based on its NVIC mapping (done in the initialization of each peripheral). Each IRQ handler will clear the interrupt and determine what do. The stubbed out ISR functions (IRQ

Handlers) for each timer and switch can be found in `isr.c`. Using the pseudo code below, set up each ISR.

- (a) For `PDB_Timer()`:
 - (i) clear the interrupt in register `PDB0_SC`
 - (ii) toggle the output state for `LED1`
 - (b) For `FTM_Timer()`:
 - (i) Clear the interrupt in register `FTM0_SC`
 - (ii) If `switch2` has set the global variable to signal a button press increment the global counter variable; else do nothing
 - (c) For `Switch3`: (toggle blinking LED upon press)
 - (i) Clear the interrupt
 - (ii) If the timer is enabled, disable the timer; else, enable the timer and start it with a trigger
 - (d) For `Switch2`: (prints the button's time held down upon release)
 - (i) Clear the interrupt
 - (ii) If `SW2` was being pressed
 - 1. Set a global variable to affect the `timer2` function
 - 2. Reset the `FlexTimer`
 - 3. Reset the timer counter
 - 4. Turn on the blue LED while the button is pressed
 - (iii) Else:
 - 1. Reset the global variable to affect the `timer2` function
 - 2. Turn off the blue LED while button is up
 - 3. Print the result as "Button held for xx milliseconds!"
- Note: you will need to use your own put functions from the UART lab

10. Set up the PDB. You will only need to use the `PDB0_SC`, `PDB0_MOD`, `PDB0_CNT`, and `PDB0_IDLY` registers. Use a search feature to find the register names in the K64 Reference Manual.

- (a) Determine the frequency you want the counter to increment. You need to take into consideration the counter's size, the MCU's peripheral clock speed, and the division factors that are available.
- (b) Once you have determined the frequency, configure the Modulus register so you have the correct period.
- (c) The Interrupt Delay Register (`PDB0_IDLY`) determines at what point during the counting sequence the PDB will generate an interrupt. Change this value accordingly.
- (d) Configure the Status and Control register. Configure the PDB to use software trigger so you can start the PDB with a line of code. Do not trigger the PDB until the end of initialization. Make sure you set the `LDOK` field, so that it will load in the `IDLY` and `MOD` values you set.
- (e) Enable interrupts on the PDB in the NVIC and set up an ISR for it.
- (f) Send a software trigger to the PDB (in the Status and Control register) to start the timer.

- (g) The above steps should be added to `initPDB()` in `main.c`.
- 11. To set up the Flex Timer, write code based on the following pseudo code in `initFTM()` in `main.c`:
 - (a) Enable the clock to the FTM0 module
 - (b) Turn off FTM write protection
 - (c) Divide the input clock down by $2^{FTM0_CLK_PRESCALE}$
 - (d) Reset the FTM counter to zero
 - (e) Set the overflow rate to $\frac{(DEFAULT_SYSTEM_CLOCK)}{\frac{128}{1000}}$
 - (f) Enable the interrupt mask
 - (g) Select the system clock
- 12. Turn on interrupts by using the `NVIC_EnableIRQ` macro on each IRQ needed in `initInterrupts()` function in `main.c`

Demonstrate full functionality of both timers by loading software onto the K64, connecting it via the serial port, and pressing SW2 and SW3 with a lab instructor or TA. Take a screenshot of the terminal for your report.

Part 2 - Analog to Digital Converter

ADC Channels and Pins Each ADC has multiple channels that are connected to various MCU pins. The K64F dedicates six pins (A0:A5) for analog input signals on PTB2, PTB3, PTB10, PTB11, PTC11, and PTC10 respectively.

Analog to Digital Converter (ADC) The reference manual provides initialization information starting in **Chapter 35**. Additional ADC Hardware Trigger options can be found in the `SIM_SOPT7` register.

Programmable Delay Block (PDB) The PDB in this lab is configured to trigger the ADC at a 1 KHz frequency. To generate trigger outputs, the pre-trigger output was enabled and the delay 0 register set to 0. Every time the PDB's counter reaches 0, the trigger fires.

1. Create a new project in Keil μ Vision 5.13.0
2. Download the code for the Analog to Digital Converter from MyCourses
3. Complete the `ADC1_Init()` function
 - (a) Replace the (Insert your code here.) in `main.c` with the appropriate initialization code
 - (b) Be sure to enable interrupts and hardware triggering

Note: The ADC status control registers needed for this exercise are located in **Section 35.3.1 (Page 831)**
4. Take the UART code from Exercise 2 and import it to the ADC project
5. Copy the contents from `main_timer_template.c` to your `main.c`

6. In your main.c file enable the UART pins and initialize the UART
7. In a 'for' loop print out decimal and hexadecimal representations of register ADC1_RA. The data result register (RA) contains the result of the ADC conversion of Channel 1 for this specific case.
8. On the breadboard create the simple circuit shown in Figure 5.2. Both power and ground can be provided from the K64 **in this case**.

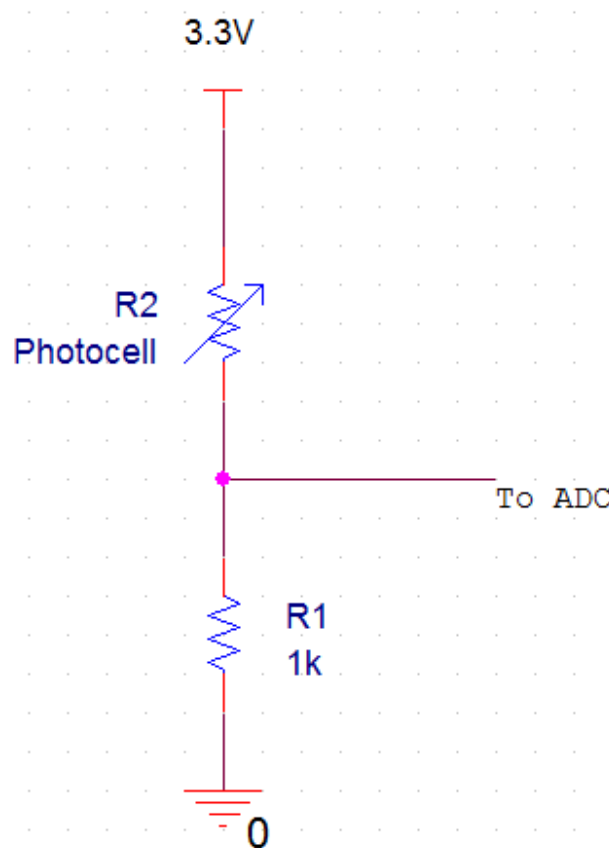


Figure 5.2: Photocell Circuit

9. Connect the middle node of the photocell circuit to the ADC at pin ADC0_DP0 (look at the wiring schematic for confirmation of this). The simple voltage divider based on the variance of the CDS cell will provide a variable analog value that will be observable through the UART interface, similar to previous lab exercises.
10. **Demonstrate your working circuit to your Lab Instructor or a TA**
11. Repeat the procedure with the TMP36 temperature sensor (this device outputs voltage directly, no need to construct a circuit). Display the temperature in both $^{\circ}C$ and $^{\circ}F$
12. **Demonstrate your CDS and temperature sensor working with your Lab Instructor or a TA**

Part 3 - NXP Car Camera

Line Scan Camera The NXP Car Kit comes with an independent line scan module that uses a light array and lens that returns an analog 1-D view slice of what it sees. The timing diagram for the camera can be seen in Figure 5.3. The camera has three inputs:

1. CK (clock) - latches SI and clocks pixels out (low to high) continuous signal
2. SI (serial input to sensor) begins a scan / exposure discrete pulses, pulse must go low before rising edge of next clock pulse
3. AO (analog output) - Analog pixel input from the sensor (0-Vdd) or tri-stated

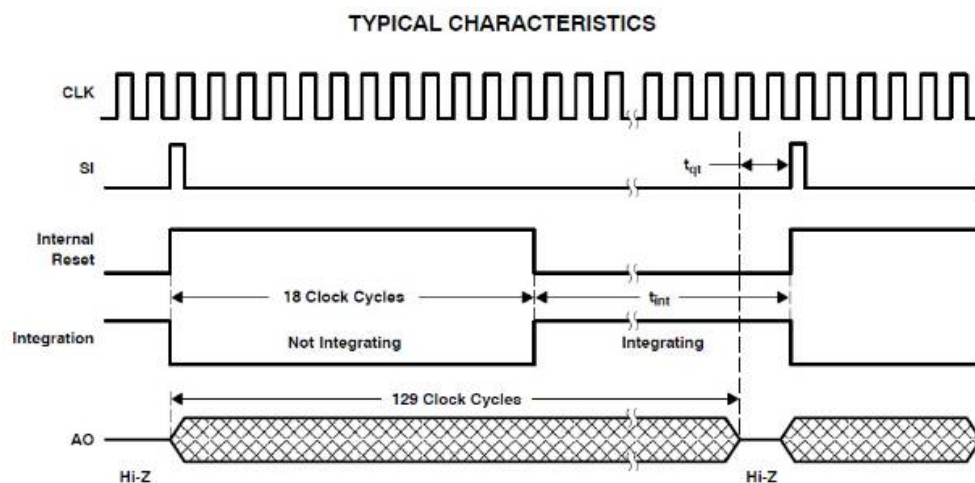


Figure 5.3: Line Scan Camera Timing Diagram

1. Create a new project in Keil μ Vision 5.13.0
2. Write code to interface with the Line Scan Camera using the timer modules and the ADC. Use camera_FTM_blank.c as a template.

Some notes to consider:

- Usage document: <https://community.freescale.com/docs/DOC-1030>
- Do not exceed over 100ms for the integration time otherwise the capacitors will saturate
- Each set of readings will take 128 cycles, each cycle is one piece of the input
- Use the FTM timer to toggle your clock signal and SI pulse on a GPIO at a period of $10\mu s$
- Use the PIT timer module to determine the integration period

Mounting Mount the camera on a dowel on the front of the car. Play around with the height and camera angle to find the best position for it to find a line in front of the car. If your 3D printed mounts are unusable inform your professor or TA.

Verification To verify that proper inputs are being received, use the provided MATLAB template (plot_cameras_serial_blank.m) to read in values from the camera using the COM port on the computer and UART module on the K64. (Note: You may need to add start and stop values to your incoming data). Graph these values using the MATLAB plot function. Output the values coming from the camera to an oscilloscope to verify the correctness of the MATLAB script and A2D conversion on the K64. Along with the raw data plot, plot both a low pass filter on the data as well as an edge detection plot (binary).

Demonstrate your MATLAB and Oscilloscope Output to your Lab Instructor or a TA

Lab Report

Follow the general Lab Report Guidelines on MyCourses. In addition to handing in the printed copy of your report during the next lab period, please add any source code that changed in this exercise, along with your lab report, to the Dropbox on MyCourses.

1. Title Page
2. Short Abstract
3. Design Methodology
 - (a) Describe what was done, why it was done, and how it was done. It is expected that all initialization sequences and procedures are explained in depth in this section.
 - (b) Describe any problems encountered and the fixes performed.
4. Results
 - (a) A terminal screenshot of the UART output for Part 1.
 - (b) A terminal screenshot of the UART output for Part 2.
 - (c) A screenshot of the MATLAB output for Part 3.
5. Analysis
 - (a) What is the smallest amount of time you could measure? Longest amount of time?
 - (b) You need to describe what speed the PDB timer is operating at, with MOD=32,000 how often are ADC measurements made, and calculate the highest frequency you should be able to read in with no aliasing.
6. Questions
 - (a) Why do the IRQ flags need to be cleared in the ISR functions?
 - (b) Which timer would you use to measure time while power is off?
 - (c) Why does the LDOK field for the PDB0_SC register need to be asserted?
7. Conclusion

Exercise 5: K64 Timers, Interrupts, and Analog-to-Digital Converter

Student's Name: _____ Section: _____

PreLab		Point Value	Points Earned	Comments
PreLab	Prelab C Code	10		

Demo		Point Value	Points Earned	Date
Demo	Functionality with SW2	10		
	Functionality with SW3	5		
	ADC Functionality with CdS	10		
	ADC Functionality with Temp Sensor	5		
	Linescan Camera MATLAB and Oscilloscope	10		

To receive any grading credit students must earn points for both the demonstration and the report.

Exercise 5: K64 Timers, Interrupts, and Analog-to-Digital Converter

Report		Point Value	Points Earned	Comments
Abstract		5		
Design Methodology	Discussion	10		
Results and Analysis	Part 1 Screenshot	5		
	Part 2 Screenshot	5		
	Camera Plot	5		
	Timing Explanations	3		
Conclusion		5		
Questions	Question a	4		
	Question b	4		
	Question c	4		
Total for prelab, demo, and report		100		

Lab 6 Motor Control

Introduction

Microcontrollers such as our K64F operate at low voltages and low currents. Many real world devices and switches operate at much higher voltages and currents. For example, the motors on a typical wheelchair have a typical operating condition of 24V at 20Amps, yet can be controlled with only a 3.3V, 20 mA signal. The amplification of the microcontroller signal using transistors or MOSFET's, or usage of relays is a solution for simple on-off circuits. Motor control can be more complex. For example, DC motors require continuous voltage variation while stepper motors require pulsed timing patterns. Further, both need not only speed control, but direction control.

Files

https://kgcoe-git.rit.edu/CMPE_460/CMPE_460_Files/tree/master/Lab_6

Objectives

For this exercise, you will understand how to amplify low power signals from the microcontroller to drive higher power devices. Further, this lab will explore how to drive motors using H-bridge circuits.

Prelab Activities

1. This lab will use Lynxmotion GHM-01 gear head DC motors. The characterization data of the gearbox output is specified with the curves in Figure 6.1. The figures include the gear box. The gear reduction is 30:1. From the curves below, estimate:
 - (a) Motor stall torque in oz-in and in kg-cm before reduction
 - (b) Maximum current draw
 - (c) Maximum motor turn speed before reduction
 - (d) Maximum torque in oz-in after gear reduction
 - (e) Maximum turn speed in kRPM after gear reduction

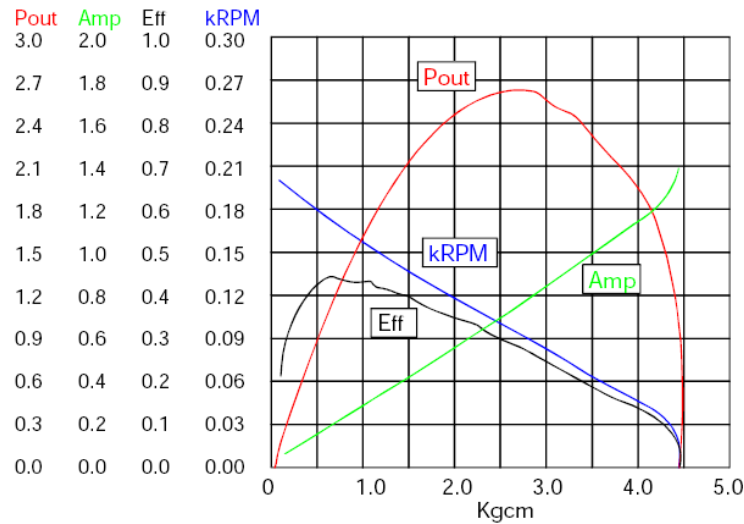


Figure 6.1: Motor Characterization Curves

2. A typical microcontroller can source or sink only about 10-25mA per IO pin. As such, if we were to generate a 5V, 20mA PWM signal from the K64, it would not provide sufficient power to turn our DC motor. To solve this problem, transistors, H-bridges, and motor controllers are used. Each of these amplify the output signal from the IO port to the appropriate voltage and current necessary to drive the motor. For example, we could use NPN and PNP transistors to construct our own H-bridge. In the circuit in Figure 6.2, by having IO A high and IO B low, the motor turns in one direction, and swapping their polarities makes the motor turn in the other direction. Answer the following questions based on Figure 6.2.

- (a) What is the purpose of the diodes in this circuit?
- (b) What is the purpose of the capacitor in this circuit?
- (c) Why are the transistors in pairs?
- (d) Why use 2N3904/2N3906 for some transistors and TIP31/TIP42 for others?

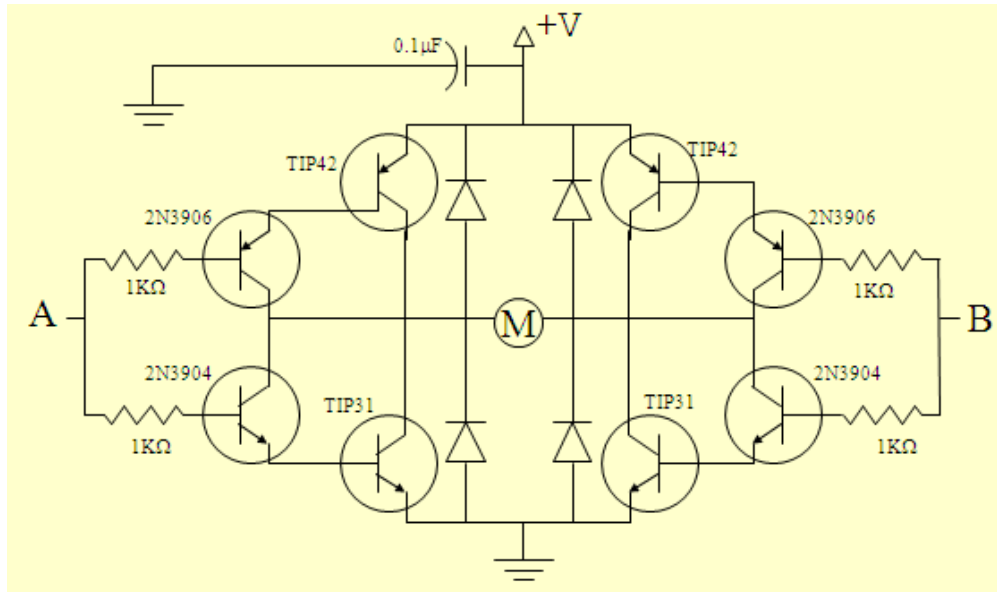


Figure 6.2: DC-Motor Control H-Bridge

Procedure

1. Create a new project in Keil μ Vision 5.13.0
2. Copy PWM.c, PWM.h, and your uart code into the project and start with main_template_Lab7.c
3. Use the sample code to generate a 20% duty cycle signal at 10KHz. The PWM signals are generated on PTC4 and PTC3- consult the K64 Reference Manual to find the corresponding pin on the K64 board. **Display on oscilloscope and demonstrate to your lab instructor or a TA to be checked off.**
4. Explain how the code works in your report and attach oscilloscope screenshot in your report.
5. The circuit shown in Figure 6.2 is offered as a prepackaged integrated circuit such as the L293NE/SN754410 which contains four half bridges. Two half bridges make a full H-bridge, so one L293NE/SN754410 IC can drive two DC motors. For simplicity, we'll only drive one motor, making connections as per the L293NE/SN754410 datasheet and class lecture slides. Connect the microcontroller generated PWM signals to input pins on the chip (use left side). Connect the enable pin and Vcc1 to +5V from the K64. Connect Vcc2 to +10V from power supply. Note: to drive the motor clockwise, one of the two signals is set to ground or 0% duty cycle; to change direction, swap which signal (input pin) is PWM and which is ground.
6. Before connecting your circuit to the motor, verify the signal on the output pins L293NE/SN754410 on the oscilloscope. Attach a screenshot showing the input 3.3V waveform and output 10V waveform on the oscilloscope. After verification, go ahead and connect to the DC motor.
7. In your report, explain how to change speed and direction of turn of the DC motor.
8. Drive the provided DC motor with a 10KHz, 20% duty cycle in either direction.
9. In the main() function, write an infinite while loop of code that does the following:

- Generates a 0% to 100% duty cycle in forward direction
- Generates a 100% down to 0% duty cycle in the forward direction
- Generates a 0% to 100% duty cycle in reverse direction
- Generates a 100% down to 0% duty cycle in the reverse direction

Put a $\sim 10ms$ delay between each duty cycle and verify the code is working with an oscilloscope, then attach the H-bridge circuit

10. **Demonstrate the above code that turns the motor at different speeds and direction to the lab instructor or a TA before continuing.**
11. The above method requires two PWM lines, one for forward, and one for reverse. Describe in your lab report an alternate method in your report that uses only one PWM line and one GPIO line.
12. Stepper motors are good for precise positioning and when there is a need for strong holding torque. You are provided with a sample stepper motor. Is it unipolar or bipolar?
13. There are eight wires coming from the stepper motor. Note that some of the stepper motors have a gray ribbon cable, others have colored wires. Aside from that, the motors are identical. We could use the tips in the lecture notes to find each of the four coils, but, if you did, you would find we have 4 independent coils. This makes it difficult to figure out which are the A/C and which are the B/D coils. So, using Table 6.1, connect the stepper motor to the ULN2068B/ULN2083A Darlington array (noting that common wires of stepper motor need to be connected to Vss, not to ground). **Set Vss to +5V from external power supply.** Starting with the red wire (if gray ribbon wire, pick either end to start), the motor leads are in the order **C G D G G A G B**, where all the wires labeled “G” which should go to Vss, and the lettered wires are driven from the microcontroller via Darlington transistor switches. Be very careful to have the order of the lettered wires correct. For example, the first C wire on the stepper gets connected to the line marked C (1b). Make sure ground of power supply is connected to ground of microcontroller. Use PortD Pins [0,1,2,3] to drive the Darlington transistor switches.

Note: The stepper motor draws $\sim 1A$ of current and the ULN2068 ULN2083A/ULN2083A chips are only designed to handle $\sim 0.5A$. Do not run the stepper motor for more than 10-15 seconds at a time to prevent overheating and damage to the circuit components. The ULN2068/ULN2083A can get hot enough to burn to the touch, so be careful.

Table 6.1: Darlington Connectors

From Microcontroller	To Stepper
PORTD0	A, 1a
PORTD1	B, 2a
PORTD2	C, 1b
PORTD3	D, 2b

14. Modify your main function with sample code below, demonstrate turning the stepper motor at different speeds and directions to the lab instructor or a TA. Referring back to the class notes, which stepping mode does this code use?

```

// Enable clocks on Port D

//Configure the Signal Multiplexer for the Port D GPIO Pins

// Configure the GPIO Pins for Output

int forward = 1;
int phase = 0;

while(true){
    // Turn off all coils, Set GPIO pins to 0

    //Set one pin high at a time
    if(forward){
        if(phase == 0){/*turn on coil A*/; phase++;} //A, 1a
        else if(phase == 1){ /*turn on coil B*/; phase++;} //B,2a
        else if (phase == 2) { /*turn on coil C*/; phase++;} //C,1b
        else { /*turn on coil D*/; phase=0;} //D,2b
    }
    else { //reverse
        if (phase == 0) { /*turn on coil D*/; phase++;} //D,2b
        else if (phase == 1) { /*turn on coil C*/; phase++;} //C,1b
        else if (phase == 2) { /*turn on coil B*/; phase++;} //B,2a
        else { /*turn on coil A*/; phase=0;} //A,1a
    }
    //Note- you need to write your own delay function
    delay(10); //smaller values=faster speed
}

```

15. Your NXP Cup Car has two DC motors and a servo. Create a new project which can drive the DC motors using one timer module and the servo on a second timer module. Run the DC motors at 10KHz using the sample code from the DC motor section of this lab. Run the servo at 50Hz on a second timer (you will have to change frequency and duty cycle of sample code. Hint: Most servo motors use a standardized control signal. Google it). **Display on oscilloscope and demonstrate to TA to be checked off before hooking up to the NXP Car. Then display your NXP Cup Car rear motors and servo being driven simultaneously. (It is okay if the two DC motors have the same duty cycle for this lab.)**

Shield Pinout

The adapter board sits on top of the K64F and fully covers all headers. The following table displays which pins on the motor shield connect to which pins on the K64F.

A1, A2, B1, B2 are tied directly to the PWM input on the motor shield. EN A and EN B are tied to the H-Bridge enable pins on the motor shield. In order to use one of the motor outputs the respective enable pin must be set to logic 1. For example, to use h-bridge A, PTB2 must be in GPIO mode and set high.

P1(PTE24) is tied directly to an IR receiver on the motor shield. This pin should only be used as an input from this device as the IR receiver cannot be disabled.

Table 6.2: Motor Shield to K64F Pinout

Motor Shield Pin	K64F Pin
P0	PTB18
P1	PTE24(IR)
P2	PTC9
P3	PTD0
P4	PTE25
P5	PTB11
P6	PTB10
P7	PTB19
P21	PTC10
P22	PTC1(A1)
P23	PTC2(A2)
P24	ADC1_DP0
P25	PTB2(EN_A)
P26	PTB3(EN_B)
P27	RST
P28	PTC3(B1)
P29	PTC4(B2)
SCL	PTA2
SDA	PTA1
IDSC	DAC0_OUT
CE1	ADC1_DP1
CE0	PTB20
IDSD	PTC11
SCK	PTD3
MISO	PTD2
MOSI	PTD1
Rx	PTC16
Tx	PTC17

The following table contains additional adapter pins for use with the camera and servo.

Table 6.3: Camera & Servo Pins

Header Pin	K64F Pin
SIG	PTC8
CLK	PTB9
SI	PTB23
AO	ADC_DP0

Lab Worksheet

Please submit one worksheet per group. The worksheet for this week should include:

1. Title of the Lab, your name, date the lab was performed, course name, TA's name and instructor's name.
2. Two line description of lab
3. Neat, well annotated circuit schematics (using a schematic capture program like Orcad) with actual resistor values.
4. Answers to prelab questions
5. Oscilloscope captures
6. High level description of how code works
7. Answer to in-lab questions
8. Attached the completed Lab Sign Off Sheet to the back of the worksheet

Exercise 6: Motor Control

Student's Name: _____ Section: _____

PreLab		Point Value	Points Earned	Comments
PreLab	Motor Calculations	10		
	H-Bridge Questions	10		

Demo		Point Value	Points Earned	Date
Demo	20% Duty Cycle at 10kHz	10		
	DC Motor Functionality	5		
	Stepper Motor Functionality	5		
	NXP Cup Signal Generation	5		
	Simultaneous NXP Car Motors	15		

To receive any grading credit students must earn points for both the demonstration and the report.

Exercise 6: Motor Control

Report		Point Value	Points Earned	Comments
Exercise Description		5		
Circuit Schematics		10		
Oscilloscope Captures	20% Duty Cycle	5		
	NXP Car	5		
Code Explanation		5		
Questions		10		
Total for prelab, demo, and report		100		

Lab 7 Filter Design and Simulation

Introduction

A frequency-selective filter is an electrical network designed to pass signals in a specific frequency range, called the pass band, and to attenuate signals at all other frequencies, called the stop band. Realizable filters are also characterized by a transition band between the pass band and stop band, whose exact boundary locations depend upon the specific filter approximation. An ideal filter has zero amplitude loss in the pass band and infinite loss in the stop band with linear phase throughout. However, the ideal filter is physically unrealizable because practical filters are represented by ratios of polynomials that cannot possess the discontinuities required for sharply defined boundaries and a zero-width transition band. Realizable filters are therefore an approximation, to within some specified set of tolerances, to the ideal.

For instrumentation purposes, filter requirements are usually low pass in nature because of the typically encountered low-frequency signals and the general requirement to eliminate higher frequency interference, plus the prevention of signal heterodyning (aliasing) with the sampling frequency in sampled-data systems. However, high-pass and band-reject filters are also occasionally required as well as the band-pass filter.

The Butterworth filter has a relatively flat pass band and an approximately linear phase, and the biquad band-pass filter can achieve a relatively high Q that other band-pass filter configurations are unable to achieve.

Objectives

- In this lab, you are to design and simulate several passive filters, and then design and simulate a Butterworth low-pass active filter and a biquad band-pass active filter. All simulations are done with OrCAD Capture CIS-Lite.
- Students are to implement their simulated Butterworth low-pass active filter.

Materials

- 2 to 4 TLC277CP 0-5V OpAmps

Procedure

This part is to be done individually

1. If not familiar with OrCAD Capture CIS – Lite, or if you would like to install the software on your own laptop, follow the instructions at: https://www.youtube.com/watch?v=SSTy_pisHdE&feature=youtu.be
2. Design a first order RC passive low pass filter, $R = 10k\Omega$, $f_c = 1kHz$. What is C ? Show the circuit and magnitude/phase plots from $1Hz$ to $1MHz$. (All plots should be log frequency with f_c identified, and the magnitude plot response should be $dB(\frac{V_{out}}{V_{in}})$).
3. Fix $R = 10k\Omega$, then vary $C = [100pf, 1nf, 10nf, 100nf]$ using a parametric AC sweep, and show magnitude/phase plots from $1Hz$ to $1MHz$, labeling each curve. (All plots should be

log frequency with f_c identified, and the magnitude plot response should be $\text{dB}(\frac{V_{out}}{V_{in}})$. See the Appendix if you are not sure how to do a parametric AC sweep in PSPICE. What is the slope of each of the (straight line portion) of each magnitude plot curve?

4. Design a first order RC passive low pass filter, $R = 10k\Omega$, $C = 10nf$, ($f_c = 1.59kHz$). Show the circuit and magnitude/phase plots from $1Hz$ to $1MHz$.
 - (a) Modify the circuit to be a second order RC passive low pass filter by adding a new stage before the stage you just designed. If f_c were to remain unchanged, what are R and C for this new first stage? What is the new $f(-3dB)$? Show the circuit and magnitude/phase plots from $1Hz$ to $100MHz$.
 - (b) Modify the circuit to be a third order RC passive low pass filter by adding a new stage before the two stages you just designed. If f_c were to remain unchanged, what are R and C for this new first stage? What is the new $f(-3dB)$? Show the circuit and magnitude/phase plots from $1Hz$ to $100MHz$. (All plots should be log frequency with f_c identified, and the magnitude plot response should be $\text{dB}(\frac{V_{out}}{V_{in}})$).
 - (c) What would happen as more stages are added?
 - (d) Can you think of an ideal component/circuit to put between each stage?
5. Design a Sallen-Key 2nd order low pass filter, gain of 2, $f_c = 1kHz$. For this design, $R1 = R2 = RA = 1k\Omega$, and $C1 = C2$. For the op-amp, and all op-amps used in this lab, please use the EVAL library, part uA741. Show the circuit and show magnitude/phase plots from $1Hz$ to $1MHz$. (All plots should be log frequency with f_c identified, and the magnitude plot response should be $\text{dB}(\frac{V_{out}}{V_{in}})$).
6. Design a one op-amp second order VCVS (Butterworth) low-pass filter with a cutoff frequency of $10Hz$ and a pass-band gain of 10. The design values and schematics can be found from the appropriate design chart in the ACTIVE FILTER DESIGN CHARTS document available on MyCourses. You must use a K-parameter of more than 10, but less than 25, because the 351 and 741 op-amps have very high input impedance. Show the circuit and show magnitude/phase plots from $1Hz$ to $100Hz$. (All plots should be log frequency with f_c identified, and the magnitude plot response should be $\text{dB}(\frac{V_{out}}{V_{in}})$). Note: You will be building the circuits in this step as part of Lab 8 and you will be comparing your theoretical results to your actual results.
7. Design a three op-amp biquad band-pass filter with a center frequency of $500Hz$, a Q of 25, and a pass-band gain of 10. Again, select a K-parameter of more than 10 but less than 25. Use the same design chart document as for the low-pass filter to find circuit schematics and component values. Show the circuit and show magnitude/phase plots from half the center frequency to double the center frequency. Use 1000 data points per decade. (All plots should be log frequency with f_c identified, and the magnitude plot response should be $\text{dB}(\frac{V_{out}}{V_{in}})$).

This part is to be done in a group

1. Build the Butterworth filter you've designed using the TI TLC277CP op-amp chips. Each chip contains two op-amps, neither of which needs a calibration circuit to compensate for the input voltage offset. The rail voltage for these op-amps is 5V and 0V.

Fine tune your filter circuit to meet your original design specifications. Make sure to measure and record the values of each component used in your filters.

Circuit Setup Notes:

- Capacitors have a very large tolerance. You must use a capacitance meter to measure each capacitor, and you may need to align several capacitors in series/parallel to achieve your desired capacitances. The capacitance can be measured using the multimeters mounted on plexiglass located under the upper portion of your work bench.
 - Electrolytic capacitors will not work at the frequencies that we are using in this experiment. Be sure to use the ceramic capacitors
 - Take the time to come up with a neat breadboard layout. It will be easier to debug if the components are not all clumped together.
2. You must measure and graph the magnitude and phase of your filter as a function of the input frequency. Measure the magnitude and phase of your filter over the same ranges used in your simulations.

Measurement Notes:

- When measuring the response of your filter, make sure to obtain more measurements near the corner/center frequency of your filters.
 - The keypad on the function generator (SFG-2110) can be used to select the exact frequency. Punch in the value and then select the desired order of magnitude (Hz, kHz, or MHz).
 - An input peak-to-peak voltage of 300 mV will work well for these filters. For the SFG-2110 Function Generator, pull out the amplitude knob and put it in the 9 o'clock position. Adjust the knob until the peak-to-peak voltage is about 300mV.
 - A DC offset will need to be added to the input signal to keep it away from the supply rails. For the SFG-2110 Function Generator, pull out the Offset knob and put it in the 1 o'clock position. Adjust the knob until the input signal is not hitting a supply rail.
 - Configure the oscilloscope to measure the peak-to-peak voltage of the input and output signals and the phase difference between the input and output signals.
3. Graph your measured frequency response and compare this result to your simulated response. The design charts used for the prelab include equations defining the transfer functions for your filters. It is recommended, though not required, that you plot your measured results directly on top of a plot of the mathematical frequency response of your filter so that you can easily see how your results compare. A tool like MATLAB makes this type of plotting easy.

Lab Worksheet

Please submit one worksheet per person. The worksheet for this week should include:

1. Title of the Lab, your name, date the lab was performed, course name, TA's name and instructor's name.
2. Two line description of lab
3. Schematics for each circuit with all components labelled
4. Magnitude/phase plots for each circuit with f_c identified
5. Answers to specific questions throughout the exercise
6. A signoff for the functioning Butterworth filter

Lab 8 Heartrate Monitor

Introduction

A frequency-selective filter is an electrical network designed to pass signals in a specific frequency range, called the pass band, and to attenuate signals at all other frequencies, called the stop band. Realizable filters are also characterized by a transition band between the pass band and stop band, whose exact boundary locations depend upon the specific filter approximation. An ideal filter has zero amplitude loss in the pass band and infinite loss in the stop band with linear phase throughout. However, the ideal filter is physically impossible to realize because practical filters are represented by ratios of polynomials that cannot possess the discontinuities required for sharply defined boundaries and a zero-width transition band. Realizable filters are therefore an approximation, to within some specified set of tolerances, to the ideal.

For instrumentation purposes, filter requirements are usually low pass in nature because of the typically encountered low-frequency signals and the general requirement to eliminate higher-frequency interference, plus the prevention of signal heterodyning (aliasing) with the sampling frequency in sampled-data systems. However, high-pass and band-reject filters are also occasionally required as well as the band-pass filter.

A heart rate monitor should be designed to be portable and intelligent enough to monitor the vital signs of an arbitrary individual, regardless of their state of health. Healthcare professionals should be able to gain access to that real-time data from the medical instrumentation through computer networks, internet, or wireless communications. The medical instrumentation is an embedded data acquisition system which utilizes a microcomputer to process the digital data obtained from the sensor network. In order to process the raw data, the analog signal must first be converted to digital format and then processed by the microcomputer at a specified sampling rate. The processed digital signal can then be fed into a digital to analog converter to restore it, if desired.

Prior to any MCU interaction, however, the analog signal must first be acquired, amplified, and conditioned for use. The opto-isolator is able to sense small changes and variations in reflected light as it transmits through blood vessels, reflects off the bone, and transmits through blood vessels again. We will be measuring the reflected light in the fingertip, but we could also measure transmitted light in the fingertip or ear, or measure reflected light off of any bone rich with blood vessels, such as the forehead. In this manner, an analog signal representing the user's heart rate may be acquired. From previous experience, the OPB745 has been shown to be quite noisy. In order to remedy this, a filter must be designed and placed prior to amplification that will allow for a cleaner analog signal to pass into amplification. The second stage of the circuit will provide an adequate amount of gain, which will ideally be adjustable for purposes of testing. Stage three of the circuit may consist of any other necessary signal conditioning, such as elimination of noise provided by ambient light sources. The final stage of signal conditioning may necessitate shifting the level of the analog signal to that which is acceptable by the MCU. Figure 8.1 shows the basic Block Diagram of the Heart Rate Monitor (HRM) system. Figure 8.2 shows a more in-depth Block Diagram, similar to how you will be creating yours.

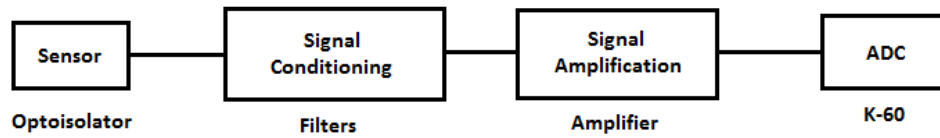


Figure 8.1: HRM Simple Block Diagram

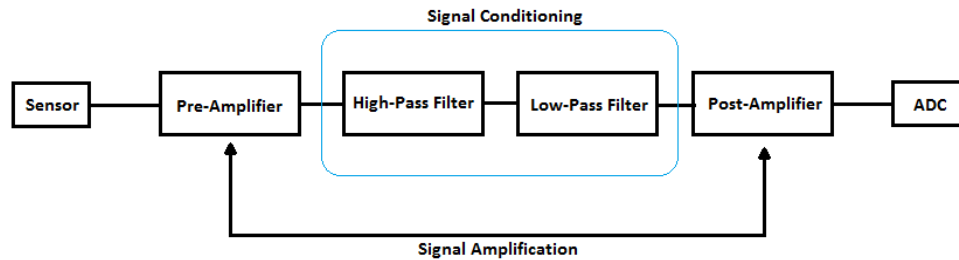


Figure 8.2: HRM Complete Block Diagram

Printed Circuit Board (PCB) design and layout is a crucial step in realizing a project's hardware implementation. PCBs are compact and robust enough to embed into a finished product. To design these boards, various design tools can be used including Cadence Allegro PCB Designer, Altium, and Eagle PCB. In this lab, Eagle PCB will be used to design a custom PCB, check for errors, estimate parts cost, and then submit the board (but not place order) to a fabrication house.

Objectives

- In this lab, students will utilize the OPB745 to sense the blood flow in their finger tip, amplify the signals, filter out the noise, and condition it to be compatible to the input voltage level of the ADC of K64 MCU.
- Students will build the heartbeat monitor using a staged approach to design, and testing as each of the following stages are implemented: sensor, (optional pre-amplifier), high pass filter, low pass filter, conditioning the output to within 0 – 3.3 V. An example block diagram is shown in Figure 8.2.
- Students will digitize the heart beat signal at a sampling rate of 1 sample per millisecond utilizing ADC algorithms.
- Understand how to design and order a custom PCB.

Materials

- OPB745 optical isolator

Procedure

1. Perform independent research on using sensors such as OBP745 for heartbeat measurement.
2. Draw the designed schematics neatly with OrCAD stage by stage. Show the component names and measured values. Construct each block as given in the lectures (PPG Introduction), and test it with an oscilloscope with both channels, one for the input of the block, and one for the output of the block. Verify the function of each block properly before you connect them together. (Note- you must use a virtual reference circuit.)
3. Connect a one hertz signal at the mV level to the input of the first stage pre-amplifier. Trace block by block, verifying the desired gain specification. For the filter stage, verify the corner frequency and the gain. Make sure the output from the filter is within 0 to 3.3 Volts.
4. Connect a 100Ω resistor to the output of the low pass filter, and connect the voltage surge protection circuit with two Schottky rectifiers as shown in Figure 8.3.

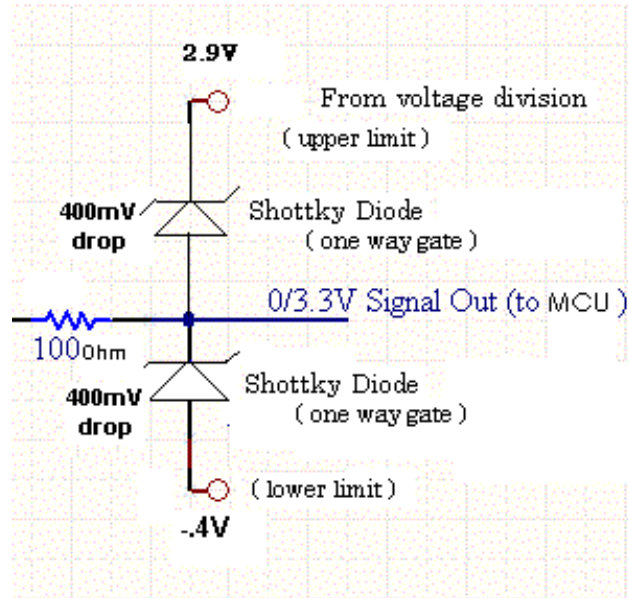


Figure 8.3: Schottky Diode Protection Circuit

5. Connect the output to your ADC and write a C code program to calculate the beat per minute (BPM) and output the message through the serial port as: `My Heart Rate is xx BPM`
6. It will be useful to implement some form of software filtering to the calculated BPM measurement (ie, measurements shouldn't jump from 30 bpm \rightarrow 300 bpm)
7. Demo your successful design to your lab instructor or a TA
8. Design a custom PCB using Eagle, show schematic and top and bottom board views. Show the lab instructor that ratsnest has no airwires and DRC shows no errors.
9. Construct a parts inventory of parts used on Eagle. Include the Eagle part description and library, as well as the cost and weblink on where to purchase.

10. Submit order to OSH Park fabrication services, create a screen capture of OSH Park PCB Order, but do not place the order. Include a total cost estimate for parts and PCB.

Lab Report

Follow the Lab Report Guidelines on MyCourses under ‘Reference Material’

1. Title Page
2. Short Abstract
3. Design Methodology
4. Results and Analysis
 - (a) Schematics neatly showing your designed filters including the actual physical values of each component used.
 - (b) The transfer functions of each filter.
 - (c) A comparison of each filter’s simulated and measured magnitude and phase frequency response.
 - (d) An explanation of any discrepancies between the simulated and measured response.
 - (e) You will write a full project report that includes *necessary background research* into the proper heartbeat schematic selection, the schematic capture of the complete heart beat circuit, OrCad transient simulations of each stage of the circuit, C code for the ADC to digitize the signal, and any relevant oscilloscope and PuTTY terminal screen captures, OSH Park order, and parts list. Figures should all be placed in an Appendix with explanation of each. Make sure this lab report includes background research (including references) you have done to select the final circuit choice.
5. Questions
6. Conclusion
7. Completed Exercise 8 Sign Off Sheet

Exercise 8: Heartrate Monitor

Student's Name: _____ Section: _____

Demo		Point Value	Points Earned	Date
Demo	Butterworth Filter	10		
	Pspice and Plots for HRM	10		
	Completed HRM	20		
	PCB Design and Layout	10		

To receive any grading credit students must earn points for both the demonstration and the report.

Exercise 8: Heartrate Monitor

Report		Point Value	Points Earned	Comments
Abstract		4		
Design Methodology	Discussion	8		
Results and Analysis	Schematics	4		
	Transfer Functions	4		
	Filter Comparison	6		
	Schematic and Background Research	8		
	OrCad Simulations	3		
	PCB Information	4		
Conclusion		4		
Questions		5		
Total for prelab, demo, and report		100		

Appendix A PSpice Parametric AC Sweep

1. Design and layout your circuit, then verify performance with an AC sweep Bode Plot
2. Choose the single component you want to vary. Change its value in the schematic from a fixed number to a variable text by double clicking on the value (click on the value, not the label) of the component, then in the Value field type a string in curly braces, ie: {myCapValues}. The curly braces tells OrCad Lite to treat this component as a parameter
3. In the Place Part menu, highlight the SPECIAL library, then select a part called PARAM. Place this part anywhere on your schematic.
4. Double-click on PARAMETERS part that appeared on your schematic.
5. Click New Property, assign the Name the same as your Value field from step 2 (ie: myCapValues) without the curly braces, and give it a default value. Hit OK.
6. Find the new column that was added, left-click and choose Display. Change the Display Format to 'Name and Value'. Hit OK.
7. Go back to your schematic, add any markers if need be.
8. Choose PSpice→New Simulation Profile, give it a name, then hit Create.
9. In the Simulation Settings, under Analysis tab, for Analysis type, choose 'AC Sweep/Noise'. Assign a start and end frequency, and enter the number of points/decade.
10. While still in the Analysis tab, under Options, click on Parametric sweep. Under sweep variable, click 'Global parameter'. Under Parameter name, enter your Value field from step 2 (ie: myCapValues) without the curly braces. In the sweep type field, either assign a start/end/increment, or use a space separated value list.
11. Click Apply and OK.
12. Pspice → Run to start a new simulation

Notes