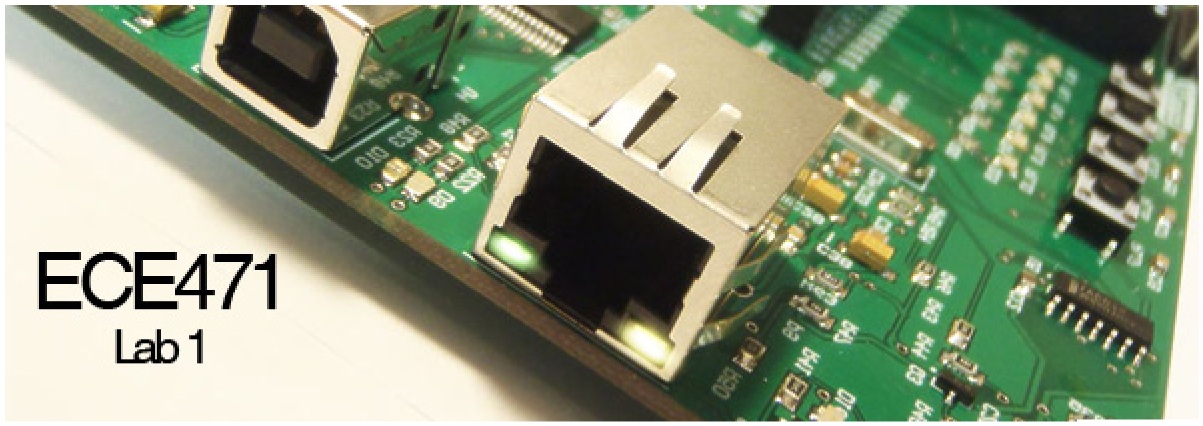


NDSU Embedded Systems II

Tact Buttons & LEDs

Objective: To obtain familiarity with the following:
- LPC Xpresso IDE
- ARM Cortex M3 Digital I/O Registers
- ECE471 Embedded Development Board

By
Nathan Zimmerman

Requirements for Lab1:

Introduction: The primary goal of the first 4 labs is to introduce students to the hardware, software, and tools that we will be using in this course. These labs will also be a crash course in basic C and ARM programming in order to ensure that we are all somewhat on the same page before we introduce more advance topics.

1. **Blink LEDs:** Use the FIOPIN register on the LPC1769 in order to blink all 8 of the LEDs on the ECE71 development board in a cylon sequence. The sequence should be [LED0, LED1, … LED6, LED7, LED6, LED5, … LED1] and repeat.
2. **Utilize a Tactile Push Button:** Use a tactile push button on the ECE471 development board in order to turn on and off the LED sequence. Use the FIOPIN register on the LPC1769 in order to read a digital input. This can be done via a software poll or an interrupt.

Extra Credit: LED Fading

Blinking LEDs might be abit too easy for some of you. There is a rather diverse level of embedded experience in this class. Consequently, in order to moderately challenge some of you, extra credit will be available in some of the labs. For this lab in particular, extra credit will be made available to those that complete the above requirements except with a slight modification. You are required to use PWM (pulse width modulation) in order to incorporate a LED “fading” or “breathing” effect. In other words, the transition between fully ON and fully off should be gradual. How you implement this is up to you.
One thing to note is that most of our senses are logarithmic. Our ears for example can barley perceive the difference between a single piezo electric buzzer and 2 buzzers. While the volume of sound emitting is approximately doubled, our ears most certainly don’t perceive this as the case. The same goes for our eyes. For example, with LEDs, the transition from 80-90% duty cycle will appear as a far smaller transition of brightness as opposed to the transition from 10-20% duty cycle. Consequently, you will need to implement some type of exponential function in order for the change in perceived brightness to be linear..

Note: If you choose to do this option you will still have to release a version of code implementing the original lab as well.

Useful Resources:

Schematic: This will be useful in order to obtain the LPC1768 port pins required in order to operate the various peripherals on the ECE471 embedded development board
Link:

LPC1769 Datasheet: Standard microcontrollers will generally come with two important resource documents for specifications. One of these documents is the datasheet which will generally contain the electrical hardware specifications of the device. The other primary resource document is the user manual which will generally contain extensive microprocessor register descriptions.
Link: http://www.nxp.com/documents/user_manual/UM10360.pdf

LPC1769 User Manual: See above for description.
Link: http://www.nxp.com/documents/user_manual/UM10360.pdf

LPC Xpresso IDE:

The Xpresso IDE (Integrated Development Environment) is a “free” eclipse based GNU-C compiler for NXP microcontrollers. The extreme advantage of this complier is that it is free and can be downloaded to your own personal PC which I would highly recommend. Currently our school computers are imprisoned by CEA tech and are rendered virtually useless for any type of practical development. The disadvantage of the Xpresso IDE is that it is code size limited to 128kB and that its licensing restrict its use for commercial development. However, 128kB should be more than enough coding space for the these labs.

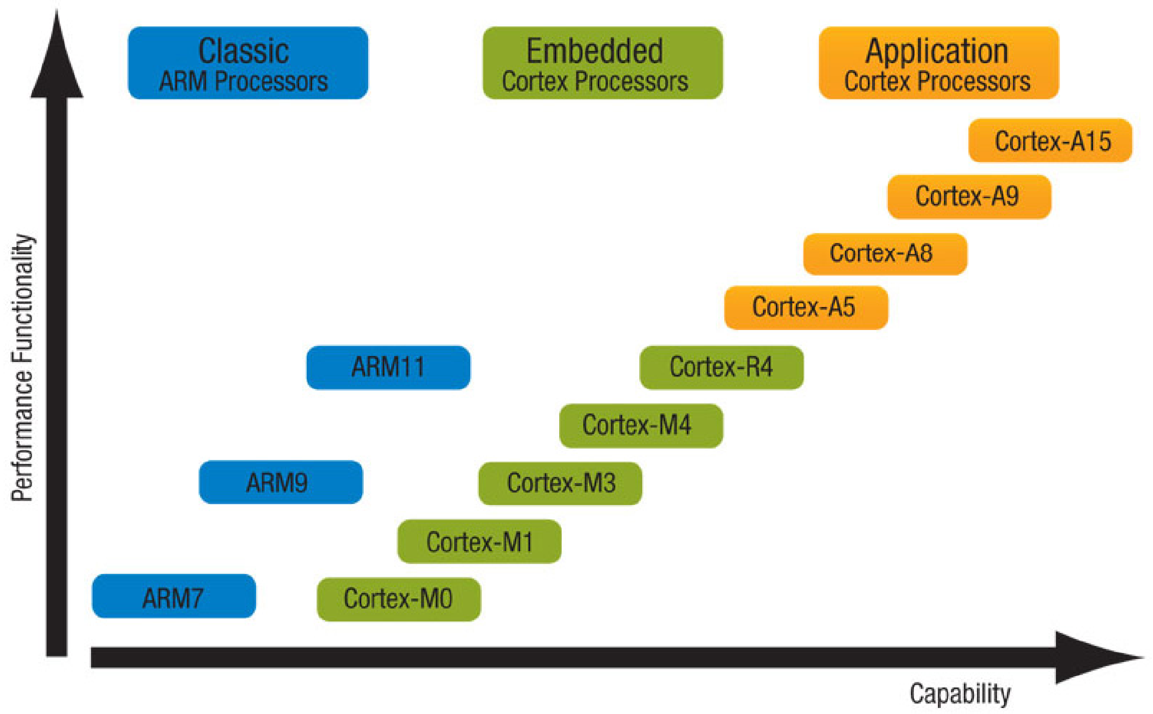
Link: http://lpcxpresso.code-red-tech.com/LPCXpresso/

What is ARM?

ARM is an acronym that stands for Advanced RISC Machine. In practice, ARM is a microcontroller architecture created by a company called ARM Holdings. ARM Holdings licenses its cores to various chip manufactures such as NXP, Freescale, TI…ect to create the end of line microcontrollers that are available on the market. **ARM's customers reported 7.9 billion ARM processors shipped, representing 95% of smartphones, 90% of hard disk drives, 40% of digital televisions and set-top boxes, 15% of microcontrollers and 20% of mobile computers.**   As a result of this popularity and industry demand, we will be using an ARM microcontroller in the ECE471 class.

What is a RISC Processor? Read here: http://en.wikipedia.org/wiki/RISC
More information on ARM processors: http://en.wikipedia.org/wiki/ARM

Where does the LPC1679 ARM Cortex-M3 rank among other ARM processors?



LPC1769 Vs. PIC18F4620

Previously in ECE376, a microchip PIC18F4620 was used. In this class we will be using the LPC1669 ARM Cortex M3 microcontroller in hopes to bring you one step closer to what is used in industry. Below is a spec comparison between the 2 processors.

Features	LPC1769	PIC18F4620
Instruction Size:	32 Bit	8 Bit
Core CLK Speed	120MHz	40MHz
Flash Size	512KB	64KB
RAM Size	64K	3.8K
GPIO	70	36
Primary Voltage	3.3V	5V
A/D Resolution	12 BIT	10 BIT
Connectivity	CAN, Ethernet, I^{2}C, IrDA, Microwire, SPI, SSI, UART/USART, USB OTG	I^{2}C, SPI, UART/USART
Price QTY 1000	6.11	4.88

LPC1769 Basic I/O Registers

Introduction: A GPIO pin stands for a General Purpose Input Output pin. As the name suggests, these pins have oodles of functionality packed into them. A GPIO can function as an analog input, analog output, digital input, digital output, SPI data line, USB data line…ect. In order to configure these IO for our specific application, the LPC1769 has a series of IO registers to manipulate the GPIOs. A few of the registers relevant to the lab are listed below and for a complete list, see the LPC1769 user manual.

Below is a snapshot of the user manual describing some basic IO registers on the LPC1769. I suggest looking it up yourself in the manual as opposed to trying to read the below image.

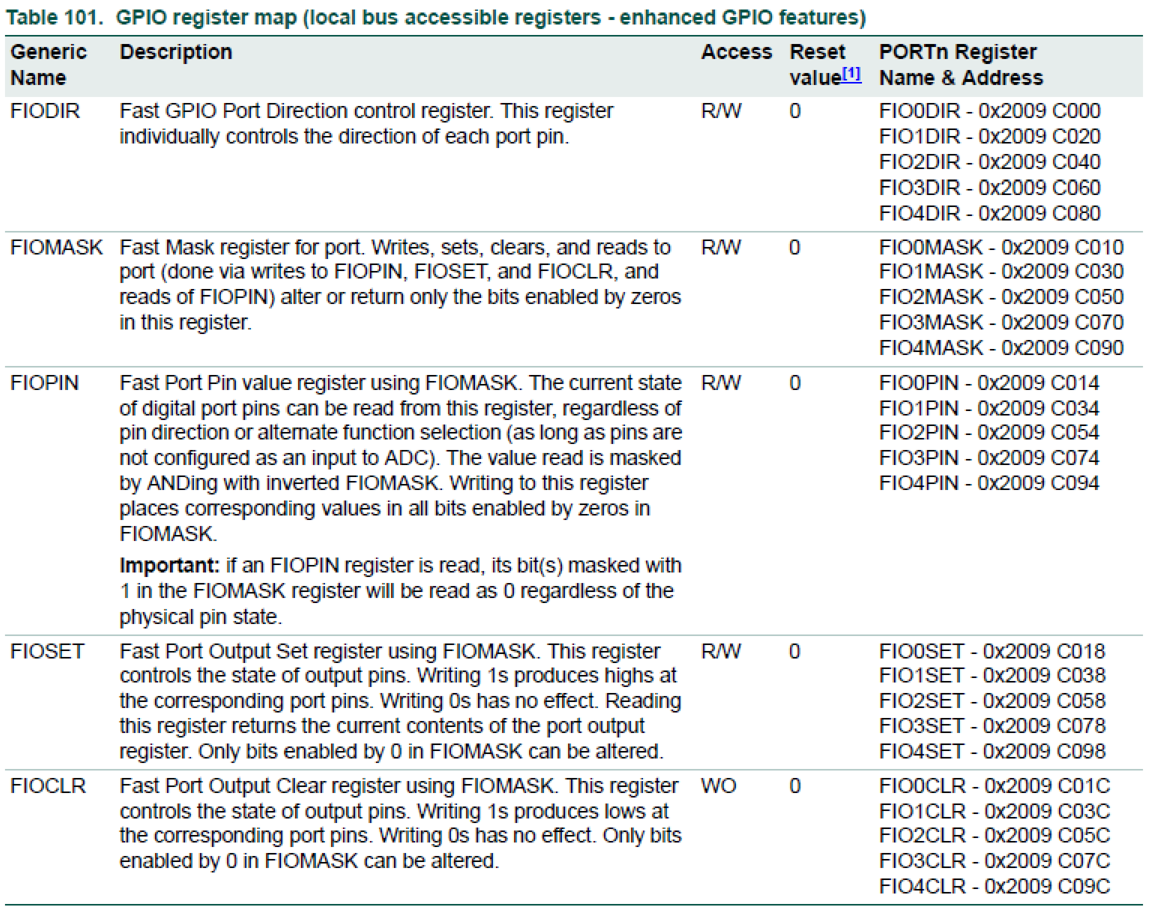


Table 101. GPIO register map (local bus accessible registers - enhanced GPIO features)

Generic Name	Description	Access	Reset value[1]	PORTn Register Name & Address
FIODIR	Fast GPIO Port Direction control register. This register individually controls the direction of each port pin.	R/W	0	FIO0DIR - 0x2009 C000 FIO1DIR - 0x2009 C020 FIO2DIR - 0x2009 C040 FIO3DIR - 0x2009 C060 FIO4DIR - 0x2009 C080
FIOMASK	Fast Mask register for port. Writes, sets, clears, and reads to port (done via writes to FIOPIN, FIOSET, and FIOCLR, and reads of FIOPIN) alter or return only the bits enabled by zeros in this register.	R/W	0	FIO0MASK - 0x2009 C010 FIO1MASK - 0x2009 C030 FIO2MASK - 0x2009 C050 FIO3MASK - 0x2009 C070 FIO4MASK - 0x2009 C090
FIOPIN	Fast Port Pin value register using FIOMASK. The current state of digital port pins can be read from this register, regardless of pin direction or alternate function selection (as long as pins are not configured as an input to ADC). The value read is masked by ANDing with inverted FIOMASK. Writing to this register places corresponding values in all bits enabled by zeros in FIOMASK. **Important:** if an FIOPIN register is read, its bit(s) masked with 1 in the FIOMASK register will be read as 0 regardless of the physical pin state.	R/W	0	FIO0PIN - 0x2009 C014 FIO1PIN - 0x2009 C034 FIO2PIN - 0x2009 C054 FIO3PIN - 0x2009 C074 FIO4PIN - 0x2009 C094
FIOSET	Fast Port Output Set register using FIOMASK. This register controls the state of output pins. Writing 1s produces highs at the corresponding port pins. Writing 0s has no effect. Reading this register returns the current contents of the port output register. Only bits enabled by 0 in FIOMASK can be altered.	R/W	0	FIO0SET - 0x2009 C018 FIO1SET - 0x2009 C038 FIO2SET - 0x2009 C058 FIO3SET - 0x2009 C078 FIO4SET - 0x2009 C098
FIOCLR	Fast Port Output Clear register using FIOMASK. This register controls the state of output pins. Writing 1s produces lows at the corresponding port pins. Writing 0s has no effect. Only bits enabled by 0 in FIOMASK can be altered.	WO	0	FIO0CLR - 0x2009 C01C FIO1CLR - 0x2009 C03C FIO2CLR - 0x2009 C05C FIO3CLR - 0x2009 C07C FIO4CLR - 0x2009 C09C

Setting a register value

Introduction: Now that we have the register descriptions, lets attempt to set one to a particular value.

Example Register: FIODIR This Fast GPIO Port Direction control register allows a GPIO pin to be defined as a digital output. If not defined, the pin is a digital input.
Below is the example of how to set a register. Setting this register defines an arbitrary port pin (P1.23) as a digital output.

```
LPC_GPIO1->FIODIR |= (1 << 23) ;
```

Now, the above code is quite basic but may not be familiar to some. Therefore, we will break it down further. We will break it down into 3 sections. If you already understand the above material skip to the next section.
1: LPC_GPIO1->FIODIR
2: |= (1 << 23);

1: Memory Addresses: Essentially all we are trying to do here is write to a register which is a memory address on the LPC1769. "LPC_GPIO1->FIODIR" references a specific spot in memory which is **0x2009 C020** according the User Manual. Consequently, "**LPC_GPIO1->FIODIR**" is simply a memory address of a register. To demonstrate this, look at the code below.
The following code is equivalent.

```
LPC_GPIO1->FIODIR |= (1 << 23) ;
(*(__IO uint32_t *)  ( 0x2009C020)) |= (1<<23) ;
```

Luckily for us, the Xpresso IDE has memory address pre defined for us such that we don't have to use pointers everywhere in order to manipulate registers. These registers are defined in **LPC17xx.h**
However, understanding what the code actually doing always helps for understanding.

Setting a register value Part 2

LPC_GPIO1->FIODIR |= (1 << 23) ;

2: Bitwise Operators: |= is a bitwise operator. Bitwise operators are used to simplify bit manipulation. A practical application of bitwise operators is to set specific bits in a register while maintaining the present value of others.

For example, say we have the following arbitrary byte address

Bit#	76543210	
Bits	xxxxxxxxx	x= unkown/don't care

If we did the following operation LPC_GPIO1->FIODIR **=** 0x8; that memory address would now be set to the following: (*Note this is an example, FIOPIN is actually a 32bit register)

Bit#	76543210	
Bits	00001000	x= don't care

While this does work, more often than not we are going to want to preserve the current values of a register. Generally one will want to write to a select few bits in a register as opposed to the whole 32bit register. If we use the bitwise OR assignment operator **|=** , with LPC_GPIO1->FIODIR **|=** 0x8 we would obtain the following result.

Bit#	76543210	
Bits	xxxxx1xxx	x= don't care

Or likewise if we want to set a specific value to 0 in a register we would use the following.
LPC_GPIO1->FIODIR &= ~0x8;

Bit#	76543210	
Bits	xxxxx0xxx	x= don't care

Now for part 3, (1<<23) is simply the value 0x1 shifted over 23 places. This is much simpler to read than the hex equivalent of (1<<23) which is 0x800000. These compound assignment operators are frequently used by other coders to simplify their code. As a result of this, learning these compound assignment operators will help significantly in understanding general C code due to their frequent use.

Ensure to review your C operators!

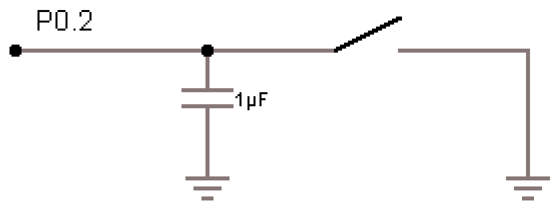
http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B

Compound Assignment Operators

Operator name	Syntax	Meaning
Addition assignment	a **+=** b	a **=** a **+** b
Subtraction assignment	a **-=** b	a **=** a **-** b
Multiplication assignment	a ***=** b	a **=** a ***** b
Division assignment	a **/=** b	a **=** a **/** b
Modulo assignment	a **%=** b	a **=** a **%** b
Bitwise AND assignment	a **&=** b	a **=** a **&** b
Bitwise OR assignment	a **|=** b	a **=** a **|** b
Bitwise XOR assignment	a **^=** b	a **=** a **^** b
Bitwise left shift assignment	a **<<=** b	a **=** a **<<** b
Bitwise right shift assignment	a **>>=** b	a **=** a **>>** b

Digital Input Reading

Example Switch Circuit on ECE471 Board



Note: There is no external pull up resistor in the schematic. Consequently, the GPIO pin P0.2 would always read low even when the button is not pressed unless an internal pull up is enabled. This is an example of one of the many features of a GPIO pin is being able to enable internal pull up and pull down resistors. Using internal resistors allows numerous external components to be saved on a particular design. Luckily for you in this case, the internal pullup resistors are enabled by default on the LPC1769. However, depending on your various application you may want a pull up, pull down, or no internal resistor enabled. Read the LPC1769 user manual for more information.

Software poll of a digital input example: Note the FIOPIN register is both a read and write register. We can write to FIOPIN in order to set GPIO that are defined as digital outputs. In addition, we can read FIOPIN to determine the value of digital inputs. Example code below.

```
IF ( (LPC_GPIO0->FIOPIN) == 0x53) {
      //Do something
}
```

Lab Report Guidelines:

All students will be expected to turn in a purty and FORMAL lab report in order to receive any credit for their lab. This lab report model is used frequently in industry and is inccrrreeeeeeedddiiiibbbllly useful. Your boss will ask you for these constantly.

Example Report Structure:

Introduction:

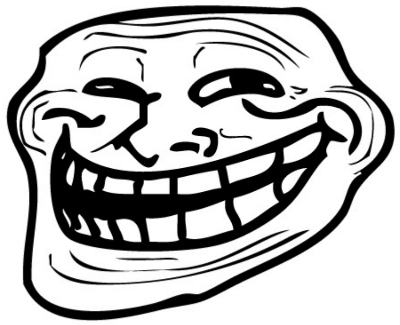
Purpose: It is expected that you include a purpose your lab in your report. However, do note there is no purpose to this report.

Materials Used: Ensure to copy and paste the following materials into your lab report or else I'll have no idea of what you used.

- **Keyboard**
- **Computer**
- **~~Brain~~**
- **Xpresso Board**

Theory:

Report: Write a 50 page double sided report in pen (NO ERASING!!!) of how the following image makes you feel.



Lab Report Guidelines:

Option B & C:

1. Upload your code files to your assigned GIT repo. You must first create an account on github and email me your account name in order to gain access to your repo.

2. If you are having trouble getting GIT to work initially, email me your code at NDSU.ECE471@GMAIL.COM. The code should be attached in the form of .c and .h files or in a complete zip archive . Your email title should be : Lab_Number : Myname. For example: Lab_1 : Nathan Zimmerman. This method of turning in assignments is cumbersome and will not be available long term so attempt to get GIT working ASAP.

   You are allowed to work in groups but you are all expected to turn in your own individual code. You should write the code yourself and it should not be a duplication of someone elses code.

Grading:

100%	**: Code runs and by majority meets requirements**
50%	**: Code does not meet requirements or does not compile but a reasonable effort was made to complete the lab.**
0%	**: Code was not turned in or significantly falls short of meeting requirements.**