Name: ..... Student ID: .....

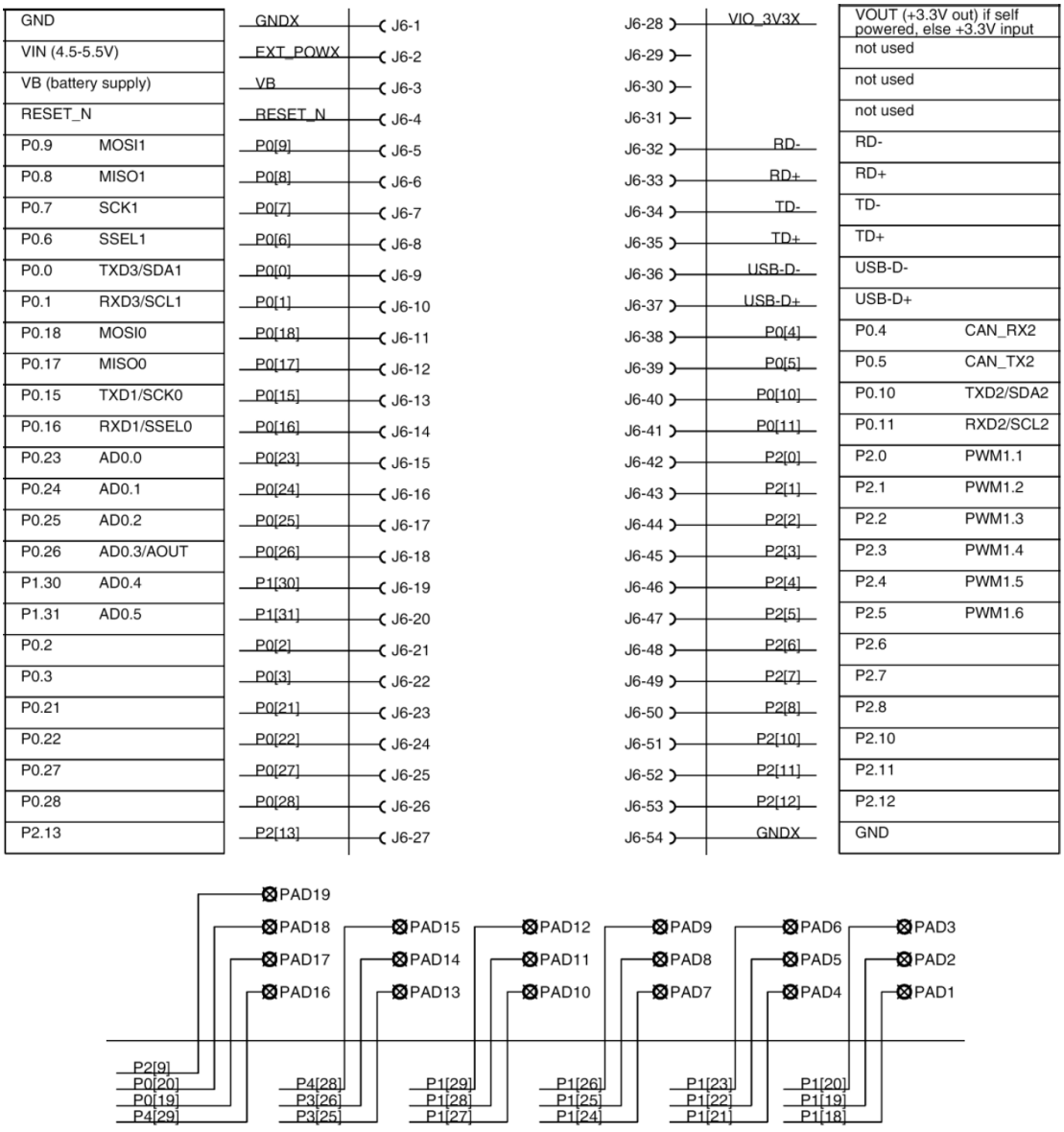
Name: ..... Student ID: .....

Laboratory 03

Introduction to GPIO

1. Introduction

GPIO stands for General **Purpose Input/Output**. This laboratory will teach you how to use the GPIO pins to switch a LED on and off and read the status of an IO pin.



With GPIO (General Purpose Input Output), you can interact with the environment, connecting up other devices and turning your microcontroller into something useful.

GPIO has two fundamental operating modes, **input** and **output**. **Input** lets you read the voltage on a pin, to see whether it's held low (0v) or high (3v) and deal with that information programmatically. **Output** lets you set the voltage on a pin, again either high or low. Every pin on the LPC can be used as a GPIO pin, and can be independently set to act as an input or output.

In this laboratory we will show how to complete 2 tasks, reading the state of a button, and making an LED blink. Using what you learn from that, you will be able to start building more complex devices, and even implementing simple communications protocols.

2. GPIO Output

GPIO output is a versatile and powerful tool, especially given that it takes little effort to use and control it. Once you have chosen a pin, the process to use it is straightforward:

- Tell the LPC that the pin should be used as an output
- Tell the LPC whether the pin should be held low or high.

2.1 Memory Mapped

In order to talk to the GPIO controller, or any other peripheral, we have to use *memory mapped registers*. When we try to read or write to a normal chunk of memory, the address and instruction are sent to the memory controller. The memory controller then either retrieves data from memory and places it into a register, or takes data from a register and writes it to somewhere in the memory. However, there are a number of privileged addresses, and when you try to read from or write to these a different pathway is taken. Here when the memory controller gets the instruction it notices the address is special, and instead of going to the memory module, it will forward the request to a register that is located in the relevant peripheral.

These registers all have different functions, each of which is detailed in the manual along with the register's memory address. The really important thing to notice is that you are not dealing with a normal piece of memory, and these *memory mapped* registers can act very differently. Unlike static memory where you can read or write pretty much anywhere, there are a number of these memory mapped registers which you can only read from. Trying to write to any of these will trap your system in a hard fault.

2.2 Blinking Lights

So let start with something simple: Connect up an LED to pin P0[9] and ground, making sure to place the proper current limiting resistor in series with it. To actually

turn on the LED you have to first **tell the LPC that the pin is to be used for output**, and then **set the state to be on**. If you look in the manual you will see that P0[9]'s direction is controlled by the 9th bit in a register name **FIO0DIR** located at 0x2009C000, and that setting it to **1** makes it an **output pin**. To set state of the pin to be on or off, set a register name **FIO0PIN** to be **1** or **0** respectively and **FIO0MASK** of that corresponding bit must be **0**. By default the reset values of **FIOxMASK** is **0**. (see Appendix for detail)

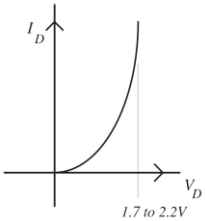
```
((uint32_t *) 0x2009C000) |= (1 << 9); // Set P0[9] to output
((uint32_t *) 0x2009C014) |= (1 << 9); // Turn On
((uint32_t *) 0x2009C014) &= ~(1 << 9); // Turn Off
```

Or use **LPCOpen API**

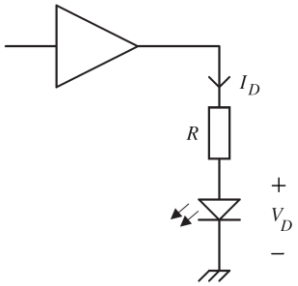
```
Chip_GPIO_SetDir(LPC_GPIO, 0, 9, 1);          // 1 for Output, 0 for input
Chip_GPIO_WritePortBit(LPC_GPIO, 0, 9, true);       // Turn on
Chip_GPIO_WritePortBit(LPC_GPIO, 0, 9, false);      // Turn off

/* Defined in gpio_17xx_40xx.h */
```

So now you should be able to make the light blink, or by varying the amount of time on and off.

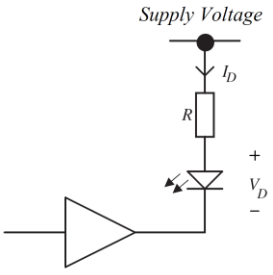
The LED (light emitting diode) has the voltage/current characteristic shown in a). A small forward voltage will cause very little current to flow. As the voltage increases there comes a point where the current suddenly starts flowing rather rapidly. Figures b) and c) show circuits used to make direct connections of LEDs to the output of logic gates, for example an LPC1769 pin configured as an output. Some LEDs, such as the ones recommended for the next program, have the series resistance built into them. They therefore don't need any external resistor connected



a) Led V-I Characteristic



b) Gate Output Sourcing Current to LED Load



c) Gate Output Sinking Current from LED

3. GPIO Input

Reading a pin uses the same registers we have already used, once a pin's mode is set to input in FIODIR, the corresponding bit in FIOPIN holds the currently read value. In addition to simply reading the registers to figure out the voltage on a pin, we have also got access to interrupts, which will notify your program when the state of a pin changes, while allowing you to do something else in the meantime.

We take the same two registers as before FIODIR and FIOPIN, and use them slightly differently.

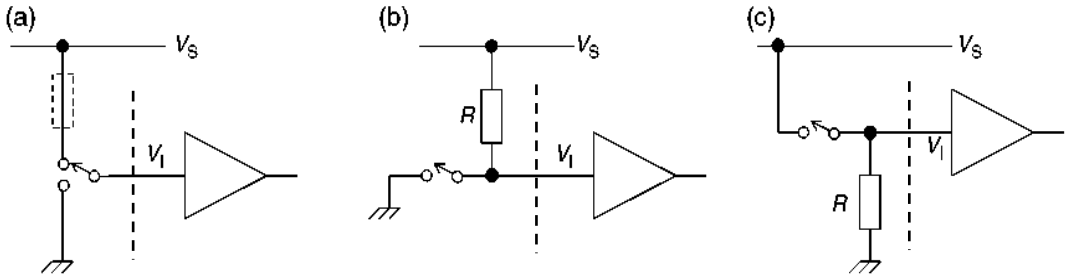
```
LPC_GPIO0->FIODIR &= ~(1 << 7); // Set P0[7] to Input
PinState = (LPC_GPIO0->FIOPIN >> 7) & 1; // Get P0[7] State
```

or

```
Chip_GPIO_SetDir(LPC_GPIO, 0, 7, 0);          // Used P0[7] for input
Chip_GPIO_GetPinState(LPC_GPIO_T *pGPIO, uint8_t port, uint8_t pin)
```

Connecting switches to a digital system

We can use ordinary electromechanical switches to create logic levels, which will satisfy the logic level requirements seen earlier. Three commonly used ways are shown here.



a) Single-pole double-throw (SPDT) connection.

b) Single-pole single-throw (SPST) with pull-up resistor.

c) SPST with pull-down resistor

Note: the diagrams show a logic buffer, which can in each case be a microcontroller input pin.

Read from a GPIO port

To use a GPIO port as input to read out a button it is needed to set the GPIO port as input with the **Chip_GPIO_SetDir** command. To check if an IO port is high or low the command **Chip_GPIO_GetPinState(** ***GPIO*****,** ***portnumber, bitposition);*** is used. The command returns a 1 or a 0. To read out IO pin 0.7 the command is:

```
bool switch = Chip_GPIO_GetPinState(LPC_GPIO, 0, 7);
```

When connecting a button, normally a pull-up or a pull-down resistor is needed. A pull-up or pull-down resistor is a high value resistor (around 10K ohm) going from a IO port to the Vcc or GND. This is to make sure that an IO port is always 3.3V or 0V in case that a button is not connected. Else, if the button is not pressed, the IO port is connected to nothing and it's not sure what value the IO port is. **The LPC1769 micro-controller has build-in pull-up and pull-down resistors.** By default after reset, all I/O ports is configured to input pull-up state.

4. Experiment

```
Chip_GPIO_Init(0);      // Initialize GPIO Port 0
Chip_GPIO_SetDir(LPC_GPIO, 0, 8, 1);// Port 0 Pin 8 for output

Chip_GPIO_SetDir(LPC_GPIO, 0, 9, 1);
Chip_GPIO_SetDir(LPC_GPIO, 0, 7, 0); // Port 0 pin 7 for input

bool switch = Chip_GPIO_GetPinState(LPC_GPIO, 0, 7);
int i;
for(i=0; i<10000000; i++);    // delay for debouncing switch

Chip_GPIO_WritePortBit(LPC_GPIO, 0, 8, true);
```

Modify the sample code to perform the following:

4.1 Connect 8 LEDs to PO[7:0] and make all of them flash **simultaneously** at frequency 5 Hz

4.2 Make the LEDs flash sequentially, continuously from **left to right** at frequency 5 Hz

4.3 Connect SPDT switch (SW1) to PO[8] when switch to logic H the LEDs flash from **left to right** continuously. Inversely, when switch to logic L the LEDs flash from **right to left**. Both are at frequency 5 Hz

5. Appendix

Table 102. 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

[1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.

6. References

[1] UM10360 LPC176x/5x User manual, NXP, April 2014

[2] Rohit Ramesh, Introduction to Embedded Systems Development, http://www.cs.umd.edu/~rohit/ESDbook.pdf

[3] Rob Toulson and Tim Wilmshurst, Fast and Effective Embedded Systems Design, Elsevier, 2012