Experiment 2: CMSIS and Digital Output

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
141	Sep. 12, 2014		H

Contents

1 Objectives

- Introduce CMSIS: Cortex Microcontroller Software Interface Standard
- Data structures and pointers in C
- Using external peripherals

2 Parts List

- LPC1769 LPCXpresso board
- USB A-Type to Mini-B cable
- Breadboard
- LEDs
- 330-Ohm Resistors
- Jumper wires

3 Background

3.1 CMSIS

> The *Cortex Microcontroller Software Interface Standard* (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series [...]. The CMSIS enables consistent device support and simple software interfaces to the processor and the peripherals, simplifying software re-use [...].
>
> — ARM Ltd. *CMSIS: Introduction*

The CMSIS components are: CMSIS-CORE, CMSIS-Driver, CMSIS-DSP, CMSIS-RTOS API, CMSIS-Pack, CMSIS-SVD, CMSIS-DAP, CMSIS-DAP [cmsis-intro].

The most relevant component to us is CMSIS-CORE [cmsis-core].

> CMSIS-CORE implements the basic run-time system for a Cortex-M device and gives the user access to the processor core and the device peripherals. In detail it defines:
>
> - **Hardware Abstraction Layer (HAL)** for Cortex-M processor registers with standardized definitions for the SysTick, NVIC, System Control Block registers, MPU registers, FPU registers, and core access functions.
> - **System exception names** to interface to system exceptions without having compatibility issues.
> - **Methods to organize header files** that makes it easy to learn new Cortex-M microcontroller products and improve software portability. This includes naming conventions for device-specific interrupts.
> - **Methods for system initialization** to be used by each MCU vendor. For example, the standardized `SystemInit()` function is essential for configuring the clock system of the device.
> - **Intrinsic functions** used to generate CPU instructions that are not supported by standard C functions.
> - A variable to determine the **system clock frequency** which simplifies the setup the SysTick timer.
>
> — ARM Ltd. *CMSIS-CORE: Overview*

CMSIS provides abstraction at the chip level only. Other libraries provide more extensive APIs for additional peripherals and board features, but are usually less generic and more vendor-specific [lpcx-cmsis].

3.2 Data Structures in C

In the C programming language, the `struct` keyword is used to define a complex data type as a group of variables. The resulting data type can then be used to declare variables, each of which would contain all of the listed variables in the structure definition.

Tip
A C structure variable references a contiguous block of physical memory.

Defining a Structure and Declaring a Variable

```
struct point {
    int x;
    int y;
};

struct point p;
```

In the example above, `p` is an instance of the `struct point` structure.

Structure Aliases Using typedef It is possible to use a shorter name to identify the structure type using the `typedef` keyword. The following example results in a variable `p` that is identical to the `p` variable declared in the previous example.

```
struct point {
    int x;
    int y;
};

typedef struct point Point;

Point p;
```

Note
C is case-sensitive. In the example above, `point` is different from `Point`.

Combining typedef and struct The `struct` and `typedef` statements can be combined into a single statement.

```
typedef struct point {
    int x;
    int y;
} Point;
```

In fact, when combined, the name immediately following the `struct` keyword, also known as the *structure tag*, can be removed.

```
typedef struct {
    int x;
    int y;
} Point;
```

Accessing Fields In the examples above, the variable `p` is of type `Point`, and thus contains two integer fields, `x` and `y`. To access the fields, the dot operator is used (`.`).

```
p.x = 5;
int z = p.y;
```

Pointers to Structures It is common to refer to structure variables by their address, or pointer, instead of the variable itself. This is especially useful when passing a structure instance as an argument to a function to avoid copying possibly-large variables during the function call.

Note
C passes arguments by value, not by reference. Pointers can be used to pass arguments by reference.

Pointers to structure variables are also useful for declaring another structure instance as a field within the structure.

Pointers to structures are used like any other pointers. The `&` operator retrieves the address of a variable, which can be stored in a pointer variable. The `*` operator is used to declare a pointer variable, and to dereference a pointer in order to access the variable it points to.

```
Point p;                /* an instance variable */
Point *pointer = &p;    /* a pointer to the same instance */
p.x = 5;
(*pointer).x = 5;       /* Identical to the previous statement */
```

Because it is very common to refer to structures using pointers instead of structure variables, a special operator, the arrow (`->`), is available to access a field of a structure using its pointer.

```
p.x = 6;
pointer->x = 6;         /* Identical to the previous statement */
```

Unions A union in C is a data type that stores different data types in the same memory location. There are two main uses of unions:

1. Storing mutually-exclusive data. If you never need to store both variables `a` and `b` at the same time, you can define them using a union so that they use the same memory space. This also applies if you want to declare a generic variable that can have multiple types.
2. Accessing the same data in different ways, or as different data types. For example:

   ```
   union {
       uint32_t x;
       struct {
           uint16_t xL;
           uint16_t xH;
       };
   };
   ```

   Here, `x` refers to a 32-bit integer, whereas `xL` and `xH` refer to the low and high 16 bits of that 32-bit integer, respectively. Changing the value of `x` would also change the values of `xL` and `xH`, depending on which bits have changed.

3.3 Accessing Registers Using CMSIS

When using CMSIS, you don't need to know register addresses, which implies that you don't need to use the `#define` directive to name the registers as you did in Experiment 1. Instead, you use the `#include` directive to include the `lpc17xx.h` header file, which contains all the register address definitions for the LPC17xx family of microcontrollers. When you use the LPCXpresso IDE to create a CMSIS project, the IDE generates a basic source file which already includes this header file.

Structures and Pointers The CMSIS header file, `lpc17xx.h`, organizes the registers into logical groups based on their functions, using C structures. First, a structure is defined by listing its fields. Then, a pointer is defined for each needed instance of that structure, pointing to the starting address of the instance, as documented in the microcontroller manual.

For example, the names of the pointers to the structure instances for the five GPIO ports are:

`LPC_GPIO0` for port 0

`LPC_GPIO1` for port 1

`LPC_GPIO2` for port 2

`LPC_GPIO3` for port 3

`LPC_GPIO4` for port 4

These pointers are defined in the `lpc17xx.h` file as follows:

```
#define LPC_GPIO0  ((LPC_GPIO_TypeDef *)  LPC_GPIO0_BASE)
#define LPC_GPIO1  ((LPC_GPIO_TypeDef *)  LPC_GPIO1_BASE)
#define LPC_GPIO2  ((LPC_GPIO_TypeDef *)  LPC_GPIO2_BASE)
#define LPC_GPIO3  ((LPC_GPIO_TypeDef *)  LPC_GPIO3_BASE)
#define LPC_GPIO4  ((LPC_GPIO_TypeDef *)  LPC_GPIO4_BASE)
```

where `LPC_GPIO_TypeDef` is the name of the structure, which is defined earlier in the file to describe the registers related to GPIO ports, and `LPC_GPIO0_BASE` through `LPC_GPIO4_BASE` are fixed addresses, also defined earlier in the header file, at which the registers for each port start. Other structures are also defined for registers related to functions other than GPIO.

Fields are Registers For each instance of a structure, such as `LPC_GPIO0`, you can access a register by accessing the corresponding field in that structure instance. For example, the three registers used in Experiment 1 are defined in the aforementioned `LPC_GPIO_TypeDef` structure as the following fields:

1. `FIODIR`
2. `FIOSET`
3. `FIOCLR`

Each of these registers is accessible within the structure instance of each port.

Important
Therefore, when using CMSIS, you need to know two names to access a register:

1. The name of the pointer to the structure instance.
2. The name of the field within the structure, corresponding to the desired register.

EXAMPLES

- To control the direction of pins in port 2, use the following register:

```
LPC_GPIO2->FIODIR
```

- To set pins in port 1, use the following register:

```
LPC_GPIO1->FIOSET
```

3.4 Connecting External LEDs

- What is an LED?
- How does an LED work?
- What is the maximum voltage that an LED can tolerate?
- If the output voltage is higher than the LED maximum voltage, what should you do?
- An LED should be connected to an output GPIO pin.

GPIO, Revisited The GPIO mode is available in all I/O pins. A GPIO pin is one that can be used as a digital input or digital output. Obviously, you need to choose the direction of the pin to determine whether it is going to be used as input or output. In this experiment, we will choose the direction to make the required pin work as GPO (General-Purpose Output). In this case (GPO), you need a command to set this output pin to HIGH (1), and a command to Clear it to LOW (0).

4 Tasks

4.1 Hardware:

1. Find out which I/O Pins you *can* use, and choose specific ones.
2. Connect the LEDs using a proper current-limiting resistor.

4.2 Software

4.2.1 Import CMSIS Libraries

1. Run LPCXpresso IDE. Choose a new workspace, e.g. `cmsis_workspace`.
2. Click *Quickstart Panel* > *Import project(s)*
3. In the *Project archive (zip)* dialog, click *Browse* next to the *Archive* field, and choose:

   ```
   C:\nxp\LPCXpresso_7.3.0_186\lpcxpresso\Examples\NXP\LPC1000\LPC17xx\ ↩
       LPC17xx_LatestCMSIS_Libraries.zip
   ```

   Then click *Next*.
4. Keep both projects selected: `CMSIS_CORE_LPC17xx` and `CMSIS_DSPLIB_CM3`, and click *Finish*.

4.2.2 Implement LED Scrolling

1. Write a program that makes it look like the light is scrolling through 5 LEDs that are connected externally. The scroll effect can be achieved by turning LEDs ON and OFF sequentially.
2. Implement light scrolling using a discrete shift register chip and a simpler program.

5 Resources

[cmsis-intro]
ARM Ltd. *CMSIS Introduction - Cortex Microcontroller Software Interface Standard*. Version 4.1. 13 June 2014.
http://www.keil.com/pack/doc/CMSIS/General/html/index.html

[cmsis-core]
ARM Ltd. *CMSIS-CORE Overview - CMSIS-CORE support for Cortex-M processor-based devices*. Version 3.30. 13 June 2014
http://www.keil.com/pack/doc/CMSIS/Core/html/index.html

[lpcx-cmsis]
LPCXpresso Support. *CMSIS support in LPCXpresso*. 7 May 2014.
http://www.lpcware.com/content/faq/lpcxpresso/cmsis-support