Essentials of Microcontroller Use Learning about Peripherals: GPIO

Essentials of Microcontroller Use Learning about Peripherals: 1 of 6

In this six-part series, we look at on-chip peripheral functions for the effective use of MCUs. In particular, we will look at some core peripherals common on a wide variety of MCUs.

CPU and Memory are the Brains. Peripherals are the Brawn.

MCUs (Microcontrollers) are widely used to control electronics devices of all types. As we explained in our earlier **"Introduction to Microcontroller"** (/support/engineer-school/mcu-01-basic-structure-operation) series, an MCU consists of a CPU (central processing unit), memory, and additional circuitry that implement a variety of peripheral support functions (see Figure 1). The CPU operates by reading programs and following instructions: instructions to read data, to carry out calculations and comparisons, to generate other operations based on results of comparisons, and so on. The memory's role is to store not just data, but also the program itself.

The MCU additionally includes circuitry that implements a variety of peripheral functions, enabling easier deployment in a variety of settings. An MCU typically includes a variety of I/O (input and output) ports, for example, to facilitate signal flow between the CPU and external sensors and switches. It also usually includes one or more ADCs (analog/digital converters) to convert incoming analog signals into digital values, and one or more DACs (digital/analog converters) to convert digital values into output analog signals. These I/O ports and converters enable use of a variety of signal types.

Figure 1: MCU's Internal Configuration(Conceptual)

Another ubiquitous peripheral is the RTC (real-time clock), which is used to enable accurate time measurements and time-of-day monitoring, and is widely utilized by processes that refer to or are dependent on time. Still another common peripheral is the UART (universal

refer to or are dependent on time. Still another common peripheral is the UART (universal asynchronous receiver transmitter), used to convert parallel signals into serial, and serial into parallel.

In this series, we will use the Renesas RX63N MCU to demonstrate the basics of on-chip peripherals. The MCU we use shall be mounted on a GR-SAKURA board, so that we can better explain how the program drives the MCU's operations.

Important Peripheral: GPIO Ports

A GPIO (general-purpose input/output) port handles both incoming and outgoing digital signals. As an input port, it can be used to communicate to the CPU the ON/OFF signals received from switches, or the digital readings received from sensors. As an output port, it can be used to drive outside operations based on CPU instructions and calculation results—for example, to drive an LED display based on calculation results, or to output drive signals to a motor.

The GPIO is referred to as "general purpose" because each pin can be freely set to function as either an input or an output. In early MCUs, each port was either exclusively input or exclusively output. A GPIO is flexible, however. If it has 8 pins, you can set them as best suits your needs: 4 input and 4 output, or 7 input and 1 output, or any other combination.

Note that while programs read, write, and operate on digital values (0s and 1s), external devices often use signal levels: LOW voltage and HIGH voltage. The GPIO handles the necessary conversions in both directions. Let's look at the basic registers[1] used by the RX63N's GPIO (see Figure 2).

---

1. Registers are special, rapidly accessible, dedicated memory circuits located within the CPU or within peripheral circuitry. They are used to store calculation results, CPU execution states, and other information crucial to program execution. As internal real estate is precious, only a limited number of registers are available; because of their location, however, they can be written and read much more quickly than regular memory. Registers come in two types: "special registers" that are limited to a specific use, and "general-purpose registers" that can be used freely for a variety of purposes.

Figure 2: Basic Structure of GPIO(Conceptual)

- Port Direction Register (PDR)
  Sets the direction of each GPIO pin; either input or output.
- Port Input Data Register (PIDR)
  Shows status of the input pins. For each pin, input of a LOW signal sets the

corresponding register value to 0; input of a HIGH signal sets the value to 1. The CPU reads this register in order to learn the most recent signal levels. Values are not saved; each time the CPU reads the register, it will reflect the current signal states.

- Port Output Data Register (PODR)
  To output data through the output pins, the CPU writes the output values to the register. A value of 0 is converted into a LOW output; 1 is converted into HIGH output. As with regular memory, the values written here are retained until overwritten. This means that the pin output level will also be maintained until the value is changed.

Programming the GPIO with the GR-SAKURA

To get some hands-on knowledge of how a GPIO port works, let's write a program that responds to the pressing of a switch by lighting up an LED. We will use the **Sakura "Digital I/O" library** (/products/gadget-renesas/reference/gr-sakura/library-digitalio) to facilitate our programing.

We will begin the program by using the library's pinMode (pin-no, mode) function to set one of the port pins to input mode, and another pin to output mode. We only have to do this once, since our program will use the same pins settings throughout.

To read the incoming value (signal level) at the input pin, we will use the digitalRead function. To output a signal from the output pin, we will use the digitalWrite function. In each case, a numerical value of 0 corresponds to a LOW signal; a 1 is a HIGH signal. When coding these, you can use either the number or the level, since they both have the same functional meaning: 0 is the same as LOW; 1 is the same as HIGH.

For our input, we will use the switch on the GR-SAKURA board. And for our output, we will use one of the GR-SAKURA's LEDs. When writing the program, we do not need to specify the pin numbers connected to these devices. Instead of pin numbers, we can use "PIN_SW" to designate the pin connected to the switch, and PIN_LEDx (where x is 0 to 3) to designate the pin connected to the corresponding LED.

For more information about the GR-SAKURA's configuration, refer to these links.

Hardware specifications (http://sakuraboard.net/gr-sakura_en.html)

Schematic(PDF) (http://sakuraboard.net/gr_sakura_schematic_mono_en.pdf)

Figure 3 shows the code for a program that turns the designated LED (LED 0) on and off in accordance with the switch value, where the GPIO port is used both to get the switch setting

and to send the LED ON/OFF signal. The program starts with a setup function that calls the pinMode function (1), which sets the pin connected to the switch to input mode, and the pin connected to the LED to output mode. We use a setup function here, as the pin mode settings must be made only once.

The program then proceeds to a loop function, which reads the switch state and turns the LED on if the switch is pressed, and off if the switch is not being pressed. This function uses the digitalRead (PIN_SW) function to determine the state of the switch. The switch on the

GR-SAKURA puts out a HIGH signal (through a pull-up resistance) when not being pressed, and a LOW signal (connected to GND) when in pressed position.

What we want to do is to turn the LED on when digitalRead (PIN_SW) returns LOW (2). To turn on the LED (D1 on the board), we simply need to call digitalWrite (PIN_LED0, HIGH) (3).

And when the switch is not being pressed, we turn the LED off by calling digitalWrite (PIN_LED0, LOW) (4), so that the corresponding GPIO pin outputs a LOW.

Try running the above program. Before running it, of course, you will need to compile it—which you can do using the GR-SAKURA online compiler. For information about using this compiler, refer to the second and fourth sessions of our Build-Your-Own Challenge, Introductory Gadget Building series.

Introductory Gadget Building Part 2: A Closer Look at the GR-SAKURA Board, and Getting Ready (/support/engineer-school/mcu-application-02-gr-sakura)

Introductory Gadget Building Part 4: Creating the Software, and Completing the Project! (/support/engineer-school/mcu-application-04-gr-sakura)

```
/*GR-SAKURA Sketch Template Version: V1.08*/
#include <rxduino.h>
#define CHATTERING 50
void setup()
{
        pinMode(PIN_SW,INPUT);                ...........(1)
        pinMode(PIN_LED0,OUTPUT);
}
void loop()
{
        if(digitalRead( PIN_SW) == LOW)       ...........(2)
        {
```

```
                digitalWrite(PIN_LED0, HIGH);...........(3)
                delay(CHATTERING);           ...........(5)
        }else
        {
                digitalWrite(PIN_LED0, LOW); ...........(4)
        }
}
```

Figure 3: Sample Program

Programming Hint: Preventing Problems Caused by "Chattering"

A switch operates by completing (closing) or blocking (opening) an electrical circuit: current flows when the switch is closed, and does not flow when it is open. Because a switch is mechanical device, however, it cannot be relied on to change the circuit state in a way that is instantaneous and clean. Rather, movement of the switch will always generate some degree of rapid vibration, which results in a brief intermediate "chattering" stage where the circuit cycles rapidly on and off before stabilizing in the correct state. You may want to try running the following program (Figure 4) to get a better understanding of chattering in action.

```
/*GR-SAKURA Sketch Template Version: V1.08*/
#include <rxduino.h>
void setup( )
{
        pinMode(PIN_SW, INPUT);
        pinMode(PIN_LED0,OUTPUT);
}
void loop( )
{
        if(digitalRead(PIN_SW) == LOW )
                {
                        digitalWrite(PIN_LED0 , digitalRead(PIN_LED0)
== 0 ? 1: 0);
                        while(digitalRead(PIN_SW) == 0);
                }
}
```

Figure 4: Sample Program (Chattering)

*The purpose of this program is to explain the principle, so it only takes into account the chattering when the switch is pressed.

chattering when the switch is pressed.

The desired operation is for the LED to come on and go off smoothly as the switch is pressed and released. But is this what will always happen? If the switch signal is read while chattering is in progress, results are unpredictable. That's why we introduced a "chattering" delay into the program shown in Figure 3.

When the program first detects that the switch has been pressed, it suspends operation for 50 ms (5). If this step is not taken, there is a risk that, an instant after detecting a switch

press and turning on the light, the program will then falsely detect that the switch has been released (because of chattering) and output a HIGH, briefly turning the lamp off again. A brief suspension is therefore introduced to allow the switch circuit some time to stabilize.

To sum up: In this session, we have looked at the structure and use of an on-chip GPIO port. This program uses LED 0, which is one of the four available LEDs on the GR-SAKURA board. You could of course choose to use a different LED; you can select any LED from 0 to 3. You could also rewrite the program's conditional judgment code to change the conditions for lighting the LED: for example, you could set the LED to light up after two presses (remaining dark after the first press), and then to turn back off at the third press. And while this lesson is limited to I/O devices that reside on the GR-SAKURA board, you could of course use external devices instead: for example, connecting an external digital-output sensor to the board's input, and connecting up a buzzer or similar device to take the place of the on-board LED.

In our next session, we will look at how an on-chip timer is used to support time-related processing. We hope you will join us.

Module List

1. MCU Programming: Basics (1) GPIO
2. **MCU Programming: Basics (2) Timers** (/support/engineer-school/mcu-programming-peripherals-02-timer)
3. **MCU Programming: Basics (3) Serial Communication** (/support/engineer-school/mcu-programming-peripherals-03-serial-communication)
4. **MCU Programming: Basics (4) Interrupts** (/support/engineer-school/mcu-programming-peripherals-04-interrupts)
5. **MCU Programming: Basics (5) Programming [1 of 2]** (/support/engineer-school/mcu-programming-peripherals-05)
6. **MCU Programming: Basics (6) Programming [2 of 2]** (/support/engineer-school/mcu-programming-peripherals-06)

Corporate

Overview (/us/en/about/profile)

Careers (https://jobs.renesas.com/)

Investors (/us/en/about/investor-relations)

News (/us/en/about/press-center)

Sustainability (/us/en/about/company/sustainability)

Contact (/us/en/contact-us)

Blogs (/us/en/blogs)

Videos (/us/en/about/press-room/videos)

Website Feedback (/us/en/websitefeedback)

Top Tools

e² studio (/us/en/software-tool/e-studio)

CS+ (/us/en/software-tool/cs)

MCU / MPU Selection Tool (/us/en/products/microcontrollers-microprocessors/mcu-mpu-selection-tool)

iSim:PE Offline Simulation Tool (/us/en/software-tool/isimpe-offline-simulation-tool)

PowerCompass Multi-Rail Design Tool (https://powercompass.renesas.com/)

PowerNavigator (/us/en/software-tool/powernavigator-software)

Timing Commander (/us/en/products/clocks-timing/timing-commander-software-download-resource-guide)

Lab on the Cloud (/us/en/labonthecloud)

Buy/Sample

Sales Support (/us/en/contact-us)

Free Sample Request (/us/en/buy-sample/free-samples)

Check Product Availability (/us/en/buy-sample/check-inventory)

Sales and Distributor Directory (/us/en/buy-sample/locations)

ROM Ordering (/us/en/products/rom-ordering)

Language

English (https://www.renesas.com/us/en/support/engineer-school/mcu-programming-

English (https://www.renesas.com/us/en/support/engineer-school/mcu-programming-peripherals-01-gpio)

中文 (https://www.renesas.com/us/zh/support/engineer-school/mcu-programming-peripherals-01-gpio)

日本語 (https://www.renesas.com/us/ja/support/engineer-school/mcu-programming-peripherals-01-gpio)

Region

Americas

©2022 Renesas Electronics Corporation.