Radiocrafts

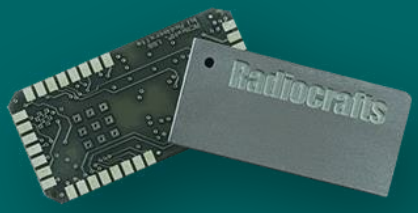
Embedded Wireless Solutions

Fast to Market. Proven Quality.

AN063: GPIO Features

By: Thomas S. Knutsen

2023-03-15



GPIO features in RIIM

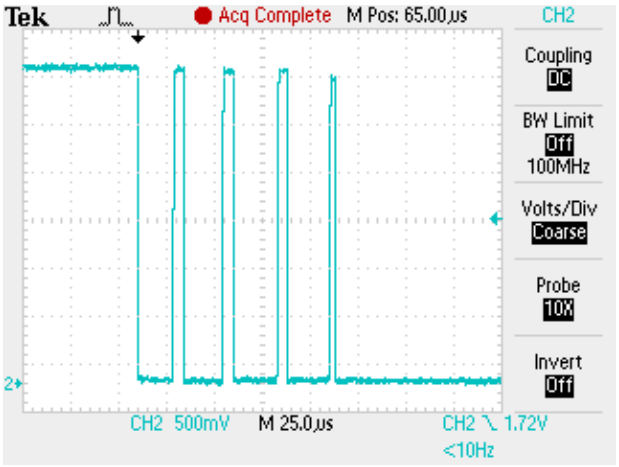
By Thomas S. Knutsen

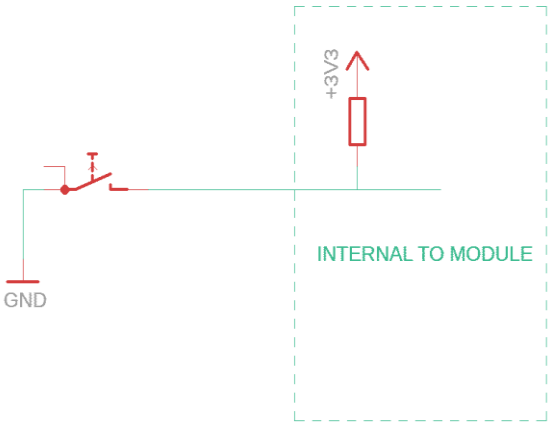
Background

There are a lot of features supported by the GPIO ports in RIIM. In this Application Note we show examples of how we solve some common problems, including use of common functions like Interrupts and PWM.

Debouncing using an interrupt and timer

All common types of push buttons will, when connected to a pin on a module, have some quite visible bounces when switching:





When connected to an input of a digital device that does edge detection, these spikes will each count as a button push.

When using modules that have ICI (Radiocrafts Intelligent C-programmable I/O) it is possible to use an edge interrupt and a timer to archive debouncing. This code for RIIM outlines the basics:

```
#include "RIIM_UAPI.h"
static uint8_t debounceTimer;

void debounce()
{
    if(GPIO.getValue(GPIO_5)== 0)
    {
       Util.printf("debounced\n");
    }
    Timer.stop(debounceTimer);

}

void interruptHandler(enum GPIO_Pin pin)
{
    Util.printf("interrupt\n");
    Timer.start(debounceTimer);
}

RIIM_SETUP()
{
    // Start as a router, leaf or border router.
    const uint8_t nwKey[16]={0,1,2,3,4,5,6,7,8,9,9,11,12,13,14,15};
    Network.setNWKey((uint8_t*)nwKey);
    Network.startMeshRouter();

    debounceTimer = Timer.create(ONE_SHOT, 100, debounce); // define the debounce
timer
    GPIO.setDirection(GPIO_5, INPUT); // using GPIO5 for the button input
    GPIO.setHandler(GPIO_5, FALLING_EDGE, PULL_UP, interruptHandler); // define
the interrupt handler
}
```

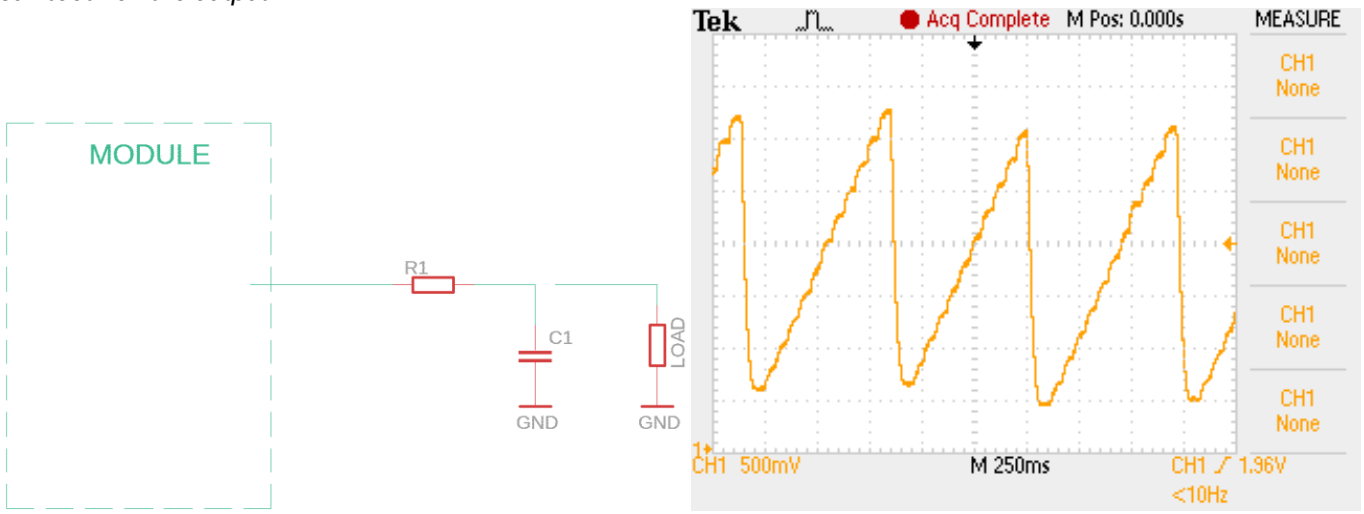
In RIIM_setup we define a timer that is a single shot timer that runs for 100ms. Port GPIO_5 is setup as an input with interrupt on falling edge and internal pullup. When this interrupt is triggered, it runs the interruptHandler routine that starts the debounce timer. After 100ms this timer expires and the routine debounce is executed. The Util.printf statements are there to show the triggering of the interrupt and the expiration of the debounce timer.

Pulse With Modulation

PWM is useful to get an analog voltage from a digital pin, without the use of a DAC, dimming of LEDs, clocking of external circuitry etc. ICI has PWM implemented, and an example is shown here:

```
#include "RIIM_UAPI.h"
uint8_t TimerHandle;
uint16_t i;

void TimerPWM()
{
    i = i+20;
    if(i > 1000){i = 0;}
    GPIO.startPWM(GPIO_6, 300, i);
}

RIIM_SETUP()
{
    // Start as a router, leaf or border router.
    const uint8_t nwKey[16]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
    Network.setNWKey((uint8_t*)nwKey);

    GPIO.setDirection(GPIO_6, OUTPUT);

    TimerHandle=Timer.create(PERIODIC, 50, TimerPWM);
    Timer.start(TimerHandle);
    i = 0;
    return UAPI_OK;
}
```

In this application we define a timer to trigger each 50ms. This increases the PWM duty cycle in order to get a sawtooth on the output.



Analog to Digital converter

Using the analog to digital converter can be useful to sample an analog sensor or measure the battery voltage.

```
#include "RIIM_UAPI.h"
uint8_t TimerHandle;
uint8_t i;
static uint32_t milliVolts;

void Timer()
{
    ADC.init(ADC0, ADC_FIXED_REFERENCE, ADC_SAMPLING_DURATION_10_6_US);
    ADC.convertToMicroVolts(ADC0, &milliVolts);
    milliVolts /= 1000;
    Util.printf("%i\n", milliVolts);
}

RIIM_SETUP()
{
    // Start as a router, leaf or border router.
    const uint8_t nwKey[16]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
    Network.setNWKey((uint8_t*)nwKey);

    ADC.init(ADC0, ADC_FIXED_REFERENCE, ADC_SAMPLING_DURATION_10_6_US);

    GPIO.setDirection(GPIO_5, OUTPUT);
    GPIO.setValue(GPIO_5, HIGH);

    TimerHandle=Timer.create(PERIODIC, 500, Timer);
    Timer.start(TimerHandle);
    return UAPI_OK;
}
```

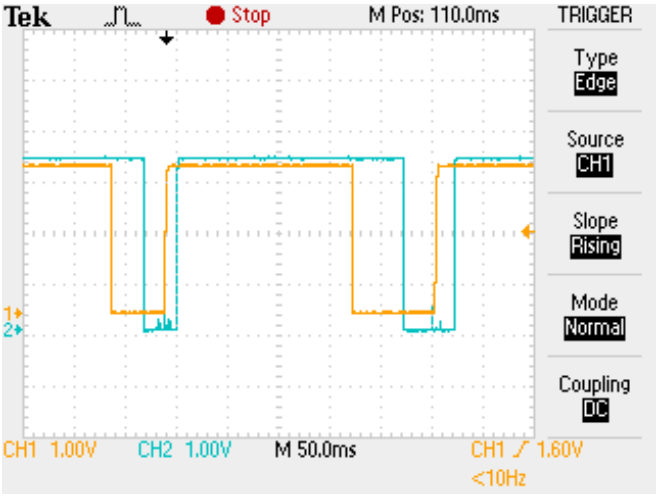
In this particular case GPIO_5 is set high, so that it works with the Sensorboard and the LDR (Light Dependent Resistor).

Reading quadrature encoder

A quadrature encoder can be read by using an interrupt on a flank on A, then determining if the B channel is leading or lagging.

```
#include "RIIM_UAPI.h"
uint8_t TimerHandle;
uint8_t i;
static uint32_t milliVolts;

void GPIOHandler(enum GPIO_Pin pin)
{
   if(GPIO.getValue(GPIO_6)==1)
  {
     i=i+1;
  }  else
  {
     i = i-1;
  }
  Util.printf("%i\n", i);
}

RIIM_SETUP()
{
    // Start as a router, leaf or border router.
    const uint8_t nwKey[16]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
    Network.setNWKey((uint8_t*)nwKey);

    GPIO.setDirection(GPIO_6, INPUT);
    GPIO.setPull(GPIO_6, PULL_UP);
    GPIO.setDirection(GPIO_5, INPUT);
    GPIO.setHandler(GPIO_5, RISING_EDGE, PULL_UP, GPIOHandler);

    i = 100;
    return UAPI_OK;
}
```

The quadrature encoder may need de-bouncing, either in software using a timer or with a hardware RC network.



Exceptional low power consumption

Leaving the ports floating leads to higher power consumption than what you may desire. To get as low power consumption as possible, each of the unused ports should be put in a known state, for instance as an input with a pull down:

```
RIIM_SETUP()
{
    // Start as a router, leaf or border router.
    const uint8_t nwKey[16]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
    Network.setNWKey((uint8_t*)nwKey);

    GPIO.setPull(GPIO_3, PULL_DOWN);
    GPIO.setDirection(GPIO_2, INPUT);
    GPIO.setPull(GPIO_2, PULL_DOWN);
    GPIO.setDirection(GPIO_1, INPUT);
    GPIO.setPull(GPIO_1, PULL_DOWN);
    GPIO.setDirection(GPIO_0, INPUT);
    GPIO.setPull(GPIO_0, PULL_DOWN);

    return UAPI_OK;
}
```

Shown here is an example with 4 ports. This should be applied to all GPIO ports that are not in use.

Document Revision History

Document Revision	Changes
1.0	Document created

Disclaimer

Radiocrafts AS believes the information contained herein is correct and accurate at the time of this printing. However, Radiocrafts AS reserves the right to make changes to this product without notice. Radiocrafts AS does not assume any responsibility for the use of the described product; neither does it convey any license under its patent rights, or the rights of others. The latest updates are available at the Radiocrafts website or by contacting Radiocrafts directly.

As far as possible, major changes of product specifications and functionality, will be stated in product specific Errata Notes published at the Radiocrafts website. Customers are encouraged to check regularly for the most recent updates on products and support tools.

Trademarks

RC232™ is a trademark of Radiocrafts AS. The RC232™ Embedded RF Protocol is used in a range of products from Radiocrafts. The protocol handles host communication, data buffering, error check, addressing and broadcasting. It supports point-to-point, point-to-multipoint and peer-to-peer network topologies.

All other trademarks, registered trademarks and product names are the sole property of their respective owners.

Life Support Policy

This Radiocrafts product is not designed for use in life support appliances, devices, or other systems where malfunction can reasonably be expected to result in significant personal injury to the user, or as a critical component in any life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness. Radiocrafts AS customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Radiocrafts AS for any damages resulting from any improper use or sale.

Radiocrafts Support:

Knowledge base: https://radiocrafts.com/knowledge-base/
Application notes library: https://radiocrafts.com/resources/application-notes/
Whitepapers: https://radiocrafts.com/resources/articles-white-papers/
Technology overview: https://radiocrafts.com/technologies/
RF Wireless Expert Training: https://radiocrafts.com/resources/rf-wireless-expert-training/

Contact Radiocrafts

Sales requests: https://radiocrafts.com/contact/