Simple button / LED sample code with indication feature - how ensure successful data transmission ?

Answered

Karsten Schmidt
asked this on May 27, 2014, 16:12

Share Tweet 0 Like 0

Hi there!

I wrote the sample code below to switch a LED on/of by pressing a button and send one indication Byte $00 / $01 to a remote device.

When button is pressed and released a little bit faster the remote device does not recognize the indication Byte. Please refer the video link.

http://youtu.be/d8jpJ5BJnhM

What could be the reason for this behavior and how it could be solved?

Many thanks!

```
# Hardware description
# Inputs: P0_0 || P1_2, P0_1
# P0_0 and P1_2 are wired parallel --> Button P0_0
# P0_1 --> Button P0_1
# Port 0 causes interrupt at L/H edge, Port 1 causes interrupt at H/L edge

# Outputs: P1_0 and P1_1 (LED's)
# Yellow LED P1_0 correspond to Button P0_0 when pressed or released
# Green LED P1_1 stays always ON when Button P0_1 pressed one time

# Messages: Indication
# "0" when Button P0_0 is pressed
# "1" when Button P0_0 is released


#Boot event listener
event system_boot(major, minor, patch, build, ll_version, protocol_version, hw)

   call gap_set_mode(gap_general_discoverable, gap_undirected_connectable)

   #P1_0 und P1_1 at High (both LED's off)
   call hardware_io_port_write(1,$3,$3)

   #P1_2 (P1_2 to P1_7) as Input, P1_0 and P1_1 as Output
   call hardware_io_port_config_direction(1,$3)

   #P0_0 and P0_1 (all GPIOs of Port 0) as Input
   call hardware_io_port_config_direction(0,$0)

   #P1_2 (all Inputs of Port 1) Pull down
   call hardware_io_port_config_pull(1,0,0)
   #P0_0 and P0_1 (all    Inputs of Port 0) Pull down
   call hardware_io_port_config_pull(0,0,0)

   #Interrupt of Port 0 at L/H
   call hardware_io_port_irq_direction(0,0)
   #Interrupt of Port 1 at H/L
   call hardware_io_port_irq_direction(1,1)
   #Interrupt of Port 0, P0_0 and P0_1 enable
   call hardware_io_port_irq_enable(0,$03)
   #Interrupt of Port 1, P1_2 enable
   call hardware_io_port_irq_enable(1,$04)

end

#HW interrupt listener

event hardware_io_port_status(delta, port, irq, state)
   # if Port 1 causes interrupt it means Button P0_0 was released
   if port = 1 then
      #P1_0 at High (yellow LED off)
      call hardware_io_port_write(1,$1,$1)
      #Send 0
      call attributes_write(xgatt_data,0,1,0)
   end if

   # if Port 0 causes interrupt the source can be P0_0 or P0_1
```

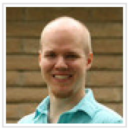
Support

```
    if port = 0 then
        # if interrupt source is P0_0 it means Button P0_0 was pressed
        if (irq & $01) = 1 then
            #P1_0 at Low (yellow LED on)
            call hardware_io_port_write(1,$1,$0)
            #Send 1
            call attributes_write(xgatt_data,0,1,1)
        else
        # if interrupt source is not P0_0 it must be P0_1 it means Button P0_1 was pressed
            #P1_1 at Low (green LED on)
            call hardware_io_port_write(1,$2,$0)
        end if
    end if

end

event connection_disconnected(connection, reason)

    call gap_set_mode(gap_general_discoverable, gap_undirected_connectable)
end
```

Button_GATT.zip

0 people would like this to be answered. Be the first!

Comments


Greg Rowberg

Karsten,

Based on what is shown in the video, it looks like you might be having an issue with button bouncing. During the mechanical transition as the button is pressed, there can be some logic noise before the electrical connection becomes solidly established (or released). You might try debouncing the button by adding something like this:

```
dim last_press

event hardware_io_port_status(delta, port, irq, state)
    if (delta-last_press) >= 3277 || (delta < last_press && ($7FFFFFFF - last_press + delta) > 3277) then
        last_press = delta

        # if Port 1 causes interrupt it means Button P0_0 was released
        if port = 1 then
            #P1_0 at High (yellow LED off)
            call hardware_io_port_write(1,$1,$1)
            #Send 0
            call attributes_write(xgatt_data,0,1,0)
        end if

        # if Port 0 causes interrupt the source can be P0_0 or P0_1
        if port = 0 then
            # if interrupt source is P0_0 it means Button P0_0 was pressed
            if (irq & $01) = 1 then
                #P1_0 at Low (yellow LED on)
                call hardware_io_port_write(1,$1,$0)
                #Send 1
                call attributes_write(xgatt_data,0,1,1)
            else
            # if interrupt source is not P0_0 it must be P0_1 it means Button P0_1 was pressed
                #P1_1 at Low (green LED on)
                call hardware_io_port_write(1,$2,$0)
            end if
        end if
    end if
end
```

May 29, 2014, 05:49

Karsten Schmidt

Hi Greg!

Thanks a lot. It's a great idea to use the internal timer as debouncing solution.

But there is still a problem. Once the button is pressed for longer then 100ms and released for longer then 100ms it works fine.

If the button is pressed and released faster then 100ms an interrupt on Port 0 accours and the yellow LED remains on. But this action should be ignored if the time between press and release is below the debounc time. How this could be achieved?

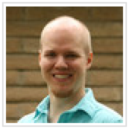
Please refer the video below. For a better presentation I have set the timer to 200ms.

http://www.youtube.com/watch?v=wgai0h-qWCU

Kind regards,

Karsten

June 3, 2014, 13:25



Greg Rowberg

Karsten,

There are two options I can think of that might be worth trying. One option would be to decrease that software timer from 3277 (100ms) to 328 (10ms) and see if you get more consistent results. You could experiment in that range and see if there's a value that is low enough to not prevent a wanted interrupt while pressing rapidly, but higher than the rate at which the button bounces.

The other option would be a simple hardware modification, adding a capacitor between the pin (+) and ground (-). The short interval of time it takes to charge and discharge the capacitor with each button press should negate any bounce in the button. For more information, see this article: http://www.ikalogic.com/de-bouncing-circuits/

June 5, 2014, 06:56

Markus Schlüter

HI there,

I would like to know if this works with multiple buttons at one port, too.

Right now I want to add different kind of buttons and switches to p1_2 - p1_7 and want to check all their states.

```
############################

#only the necessary part of my scrip

############################



dim powerstate

event system_boot(major, minor, patch, build, ll_version, protocol, hw)

   call gap_set_adv_parameters(320, 480, 7)
   call gap_set_mode(gap_general_discoverable, gap_undirected_connectable)
   call sm_set_bondable_mode(1)

   powerState = 0

end



event hardware_io_port_status(delta, port, irq, state)
   call hardware_io_port_read(1, 252)(result_port, tmp_port, tmp_data)
   call attributes_write(digital_input, 0, 1, tmp_data)

   if powerState = 0 then
          call hardware_io_port_config_irq(1, 252, 1)
          powerState = 1
   else
          call hardware_io_port_config_irq(1, 252, 0)
          powerState = 0
   end if

end
```

Right now I can read the state of multiple buttons, but I have to debounce them in hardware.

I didn't find a way to transfer your debouncing solution to my script therefore I want to ask you for your help.

Kind regards,

Markus

February 20, 2015, 11:14

Jeff Rowberg
Bluegiga
Technologies

Hello Markus,

Answer

Debouncing in hardware is typically much more reliable and straightforward than trying to do so in software, and I would highly recommend keeping it that way in your design.

As for the code that you posted, the following line is not necessary and will only take extra time:

```
    call hardware_io_port_read(1, 252)(result_port, tmp_port, tmp_data)
```

The reason for this is that the "**hardware_io_port_status**" event already contains the complete port logic state inside the "**state**" parameter of the event. This will be an 8-bit value with the current logic state of all pins on the port that triggered the interrupt. There is no need to re-read the port again. You can simply use the value contained in the "**state**" variable as the one that you write to the attribute.

February 20, 2015, 18:02

Paolo Anterri

Hi everyone, I tried the example proposed by Karsten/Greg and it works fine. My question is: is it possible to do this with by only one input instead by two inputs using the button P0_0?

I have the need to send an enable (through a push button) to BLE113 module. If the input (P0_x) goes High (from Low) a led turns ON and it stays ON until the enable stays HIGH. But when the enable goes Low (from High) the led turns OFF. Everything should be handled by a single input (P0_x) and not from two parallel input like the example above.

Thanks for attention, best regards

PAOLO A.

April 28, 2015, 10:42

Jeff Rowberg
Bluegiga
Technologies

Hello Paolo,

Answer

This cannot be done without polling (which is very inefficient power-wise) due to a hardware limitation in the chipset. You cannot have CHANGE interrupts on a single pin, only rising-edge or falling-edge interrupts are supported, not both at the same time. The only alternative is to have an interrupt on the rising edge and then a soft timer continuously polling the state of the pin until a logic LOW state is detected. Using two separate pins is much more power-efficient, if you have the pin available.

April 29, 2015, 00:32

Paolo Anterri

HI Jeff

thanks for your reply. Your suggestion sounds like the following snippet source code:

```
....

# Start a 10-ms repeating timeout soft timer
call hardware_set_soft_timer(328, 0, 0)          -> put this call into the event interrupt listener

...

# Soft timer event listener
event hardware_soft_timer(handle)

      call hardware_io_port_read(0, 1)  # read P0_1 pin status

       if data = 0 then
            # P0_x was LOW, now set HIGH
           call gap_set_mode(0, 0)  # Stop advertisements and allow the device to go to PM3 sleep mode
       end if

end
```

Is that right?

Best regards

Paolo A.

April 29, 2015, 20:30

Jeff Rowberg
Bluegiga
Technologies

Hi Paolo,

That is basically correct, except that your call to "**hardware_io_port_read**" does not select the correct pin mask (it should be "2" or "$02" for P0_1), and you do not seem to capture the return value. Something like this is required, assuming you have declared "**ret_result**", "**ret_port**", and "**ret_data**" as user variables:

```
call hardware_io_port_read(0, $02)(ret_result, ret_port, ret_data)
```

You can then check the value of "**ret_data**", and if it is 0, then the pin is low.

April 29, 2015, 20:33

Paolo Anterri

Hi Jeff,

GREAT! It works!

Thank you for your fast reply.

See you later...

Paolo A.

April 29, 2015, 20:55