1 External Interrupt Problems with ESP866

1.1 External Interrupt Crashes ESP8266 if occurs during Connecting to WiFi

If an external (GPIO) interrupt occurs and the ESP8266 WiFi stack tries to connect to a Network, the ESP8266 WiFi Stack crashes usually. (Checked with Stack version 2.3.0 and 2.4.2)

The chapter “Demo Code” shows a simplified code example that generates the crash if an interrupt source is connected to the greenKeyInput pin (PIN_D1).

The chapter “Typical Exception Stack after Crash” shows the decoded function-call stack after the crash occurs.

Looking at some different exception stacks it is obvious that the crash always occurs during the function ESP8266WiFiMulti::run(). It was not investigated if the crash rate is 100% but it is quite high.

1.1.1 Demo Code

```
#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>
#include <WiFiUdp.h>

ESP8266WiFiMulti wifiMulti;

// Input-Output Definitions
static const byte greenKeyInput = PIN_D1;
static const byte LEDPin        = PIN_D2;

// Wifi States
static const int WIFI_INIT=0;
static const int WIFI_CONNECTING=1;
static const int WIFI_CONNECTED=2;
int wifiState=WIFI_INIT;

bool ledState=false;


void setup() {
  // GPIO setupt
  pinMode(greenKeyInput, INPUT);
  pinMode(LEDPin, OUTPUT);
  digitalWrite(LEDPin, ledState);
  Serial.begin(230400);
  delay(10);
  attachInterrupt(digitalPinToInterrupt(greenKeyInput), greenKey, FALLING);
}
```

```
void greenKey() {
  ledState=!ledState;
  digitalWrite(LEDPin, ledState);
}

void updateWifi(void)
{
  static unsigned long prevMillis=0;
  static const unsigned long delayMs=150;

  switch (wifiState)  {
    case WIFI_INIT:
      wifiState=WIFI_CONNECTING;
      wifiMulti.addAP("SSID", "Password"); // add Wi-Fi networks
      Serial.println("Wifi Connecting");
    break;
    case WIFI_CONNECTING:
      if (millis()-prevMillis > delayMs)  {
        //ETS_GPIO_INTR_DISABLE();
        bool wifiConnected=wifiMulti.run() == WL_CONNECTED;
        //ETS_GPIO_INTR_ENABLE();
        if (wifiConnected)  {
          wifiState=WIFI_CONNECTED;
        }
        prevMillis=millis();
        Serial.print(".");
      }
    break;
    default:
    case WIFI_CONNECTED:
        if (wifiMulti.run() != WL_CONNECTED)  {
          wifiState=WIFI_CONNECTING;
          Serial.println("Wifi Connection lost, try reconnect!");
        }
    break;
  }
}

void loop() {
  updateWifi();
  yield();
}
```

1.1.2 Typical Exception Stack after a crash

```
Exception 0: Illegal instruction

Decoding 45 results
0x402044bc: greenKey() at
/home/user/Projects/LEDMatrix/HSG_Stopuhr/FW/test/test.ino line 78
```

0x40106646: interrupt_handler at
/home/user/.arduino15/packages/esp8266/hardware/esp8266/2.4.2/cores/esp8266/core
_esp8266_wiring_digital.c line 212
0x4010660c: interrupt_handler at
/home/user/.arduino15/packages/esp8266/hardware/esp8266/2.4.2/cores/esp8266/core
_esp8266_wiring_digital.c line 212
0x40238100: sleep_reset_analog_rtcreg_8266 at ?? line ?
0x40223f03: pp_attach at ?? line ?
0x40223f52: pp_attach at ?? line ?
0x4010106b: ppCalFrameTimes at ?? line ?
0x40238100: sleep_reset_analog_rtcreg_8266 at ?? line ?
0x40105744: spi_flash_erase_sector at ?? line ?
0x401067f4: pvPortZalloc at
/home/user/.arduino15/packages/esp8266/hardware/esp8266/2.4.2/cores/esp8266/heap
.c line 174
0x4022bd94: wifi_param_save_protect_with_check at ?? line ?
0x4022bd7d: wifi_param_save_protect_with_check at ?? line ?
0x4022be67: system_param_save_with_protect at ?? line ?
0x4022be4a: system_param_save_with_protect at ?? line ?
0x4022c334: wifi_station_ap_number_set at ?? line ?
0x40103523: lmacRecycleMPDU at ?? line ?
0x40106a05: __wrap_spi_flash_read at
/home/user/.arduino15/packages/esp8266/hardware/esp8266/2.4.2/cores/esp8266/core
_esp8266_phy.c line 267
0x4022c52c: wifi_station_ap_number_set at ?? line ?
0x4022c5fe: wifi_station_ap_number_set at ?? line ?
0x40237d60: sleep_reset_analog_rtcreg_8266 at ?? line ?
0x4022c5cd: wifi_station_ap_number_set at ?? line ?
0x4022c634: wifi_station_set_config at ?? line ?
0x40202f5c: ESP8266WiFiSTAClass::disconnect(bool) at
/home/user/.arduino15/packages/esp8266/hardware/esp8266/2.4.2/libraries/ESP8266W
iFi/src/ESP8266WiFiSTA.cpp line 558
0x40103266: lmacProcessTXStartData at ?? line ?
0x40102192: wDev_ProcessFiq at ?? line ?
0x40104bfd: ets_timer_disarm at ?? line ?
0x4022ba66: wifi_get_opmode at ?? line ?
0x4022cc2f: wifi_station_get_connect_status at ?? line ?
0x40202fc4: ESP8266WiFiSTAClass::status() at
/home/user/.arduino15/packages/esp8266/hardware/esp8266/2.4.2/libraries/ESP8266W
iFi/src/ESP8266WiFiSTA.cpp line 558
0x40100721: cont_continue at
/home/user/.arduino15/packages/esp8266/hardware/esp8266/2.4.2/cores/esp8266/cont
.S line 51
0x40202d53: ESP8266WiFiMulti::run() at
/home/user/.arduino15/packages/esp8266/hardware/esp8266/2.4.2/libraries/ESP8266W
iFi/src/ESP8266WiFiMulti.cpp line 175
0x401064da: millis at
/home/user/.arduino15/packages/esp8266/hardware/esp8266/2.4.2/cores/esp8266/core
_esp8266_wiring.c line 183
0x40202821: updateWifi() at
/home/user/Projects/LEDMatrix/HSG_Stopuhr/FW/test/test.ino line 99
0x40204214: esp_yield at
/home/user/.arduino15/packages/esp8266/hardware/esp8266/2.4.2/cores/esp8266/core
_esp8266_main.cpp line 91
0x402028d1: loop at /home/user/Projects/LEDMatrix/HSG_Stopuhr/FW/test/test.ino
line 131
0x402042a0: loop_wrapper at
/home/user/.arduino15/packages/esp8266/hardware/esp8266/2.4.2/cores/esp8266/core
_esp8266_main.cpp line 125
0x40100739: cont_wrapper at
/home/user/.arduino15/packages/esp8266/hardware/esp8266/2.4.2/cores/esp8266/cont
.S line 81

1.1.3 Solution / Workaround

A workaround is to avoid external interrupts during wifi connect. If that is not possible an other (maybe preferred) workaround is to block the external interrupts during the call to ESP8266WiFiMulti::run(). This can be done using the macros ETS_GPIO_INTR_DISABLE() and ETS_GPIO_INTR_ENABLE(). Example code is shown below:

```
ETS_GPIO_INTR_DISABLE();
bool wifiConnected=wifiMulti.run() == WL_CONNECTED;
ETS_GPIO_INTR_ENABLE();
```

After this modification the Example code above did not crash anymore. This has been tested with a quite long running setup were the interrupt input was connected to a signal generator that generates a high frequency interrupt load.

1.1.4 Are Interrupts lost due to the workaround?

A) How long does a call to ESP8266WiFiMulti::run() usually take? Measurements (see below) show about 100ms run time for a function call. All multiple interrupts that occur during that run-time are lost (blue channel)

=> That means if the external interrupt comes at a rate faster than 100ms interrupts will be lost

B) Are single interrupts lost if they hit the ESP8266 if the ESP8266WiFiMulti::run() is just running?

=> No, the device enters the interrupt service routine directly after the interrupts are enabled again. That means the interrupt is delayed but not lost.

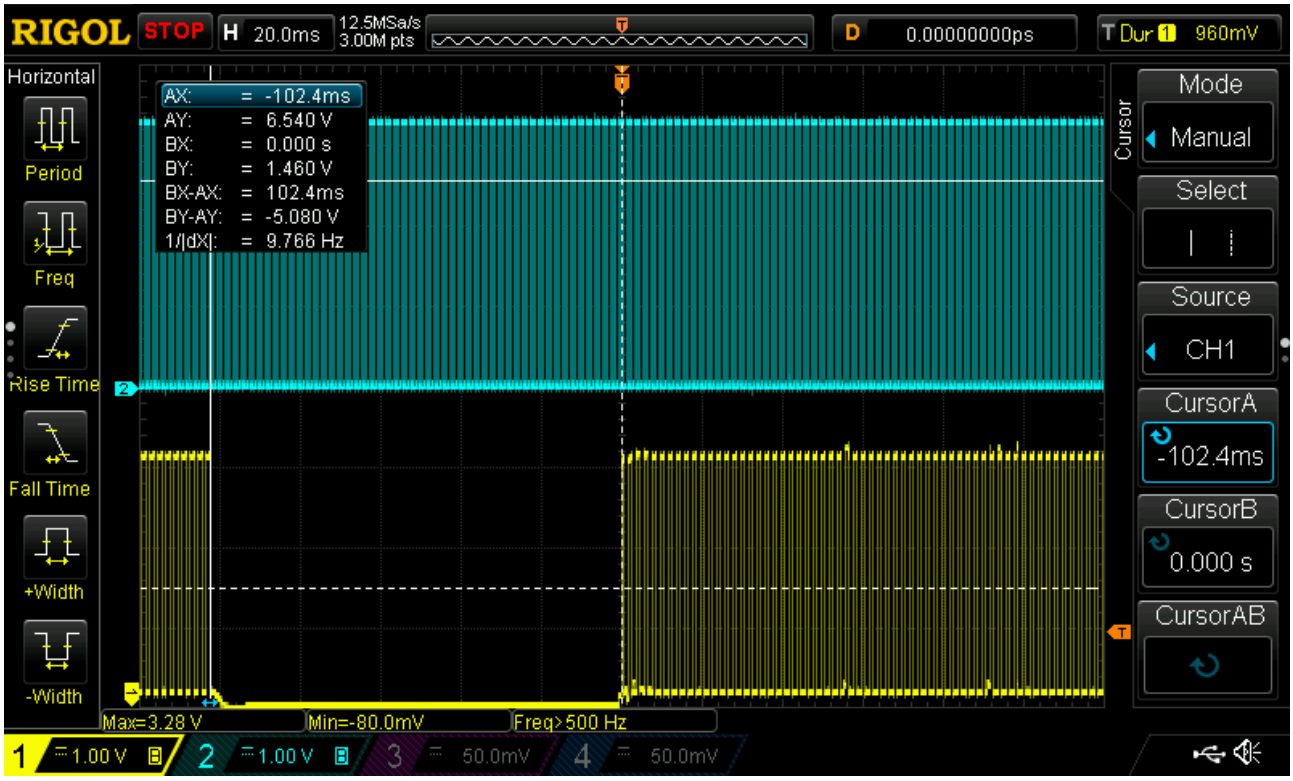

Illustration 1: Measured run-time of the ESP8266WiFiMult::run() function (blue channel Interrupt input, yellow channel LED output)

Illustration 2: Single Interrupts are not lost if hitting the device during ESP8266WiFiMult::run(). Blue channel interrupt input, yellow channel LED output)

1.2 Other Problems with External Interrupts

1.2.1 No reliable Edge Detection if rise/fall times are slow

Interrupts can be defined as “FALLING”, “RISING” or “CHANGE”. That impacts the trigger of the interrupt. Setting an interrupt to “FALLING” (as in the example code) would mean that the interrupt is only generated at the falling edge of the input signal. Unfortunately that is not always true.

The edge detection seems to work pretty well if the signal rise times are fast. But already at rise-times in the range of 1ms the picture changes. Such rise times are not unusual especially if passive (RC) de-bouncing circuits are used for external key inputs.

In the test a clean signal, coming directly out of a waveform generator was connected to the interrupt input. The rise-time of the pulse was set to 2ms. The figure below shows the wrongly generated interrupts at the rising edge (interrupt was configured to “FALLING”).

Illustration 3: Wrong generated Interrups, generated not at falling edges. Blue: Interrupt input, Yellow: LED output

1.2.2 Solution / Workaround

A first solution seems to be to check inside the interrupt service routine if the signal at the input pin is currently at low level. That's a required condition for a falling edge interrupt.

The code (not working) would look like that:

```
void greenKey() {
  if ( digitalRead(greenKeyInput)==LOW )  {  // double check for falling edge
  }
}
```

Unfortunately that does not work. It seems that the falling edge interrupt is already triggered before the digialRead() function reads the signal as low. So the following modification has been done to the code:

```
// one NOP takes  1/80000000 of a second if ARM is running at 80MHz
#define NOP __asm__ __volatile__ ("nop")  // 12ns delay

void greenKey() {
  for (int i=0; i<50; i++) NOP;  // small delay gives the pin time to change
  if ( digitalRead(greenKeyInput)==LOW )  {  // double check for falling edge
    ledState=!ledState;
    digitalWrite(LEDPin, ledState);
  }
}
```

A small delay before reading the digital input is required. A delay of 50x NOP's (means 600ns) was already sufficient in this test-setup. In the real project it might be necessary to increase this delay due to noise, different rise-times and rising edge styles. This checking for falling edges creates limitations on the low-period of the signal at the external input. The low period has to be long enough to not loose interrupts.

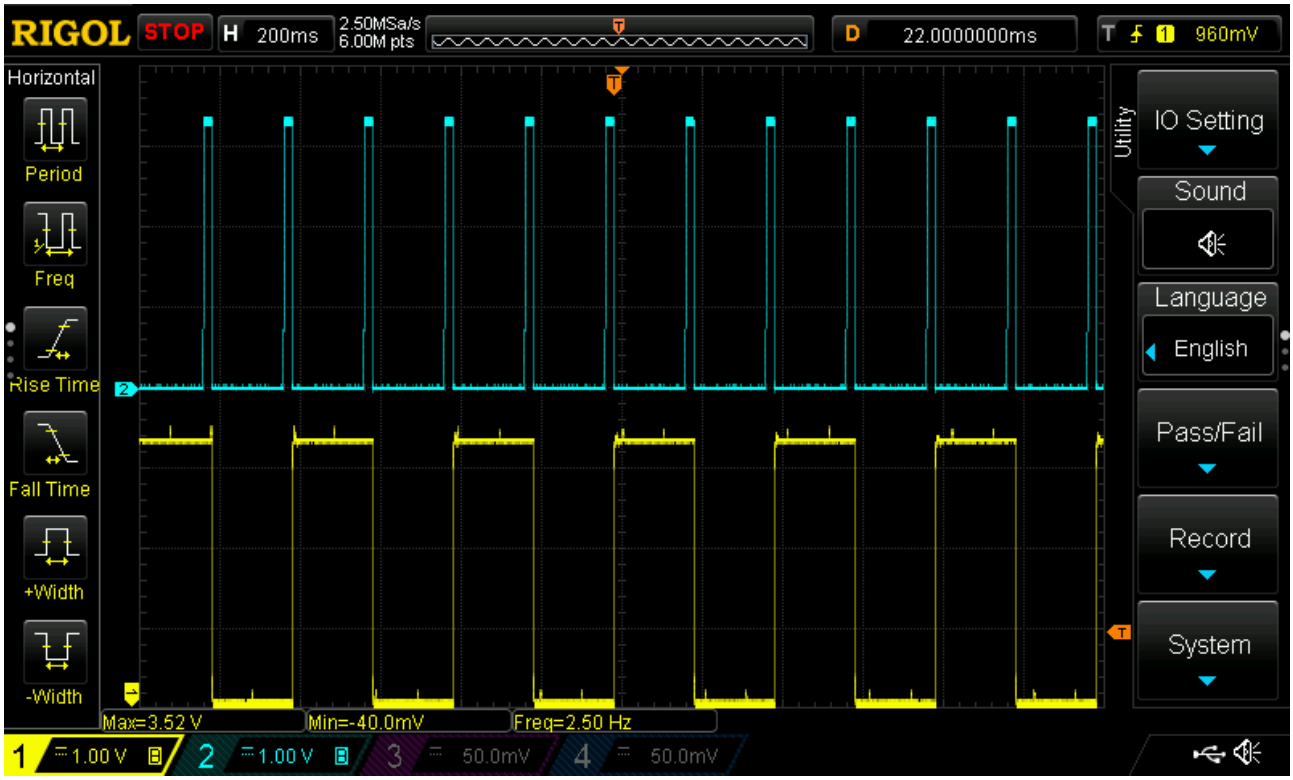

Illustration 4: Finally Interrupts are generated as expected. (blue: interrupt input, yellow: LED output)

1.3 Conclusion

There are quite some significant limitations with external interrupts and the ESP8266 WiFi Stack:

- Fast interrupt rates at external inputs are not possible if WiFi is not connected. Such interrupts will be lost due to the used workaround that disables the external interrupt during the function ESP8266WiFiMulti::run().
- Short interrupt pulses are not possible also, as the interrupt service routine must check if there was really a falling edge. That means the input signal has still to be low if the interrupt service routine starts.