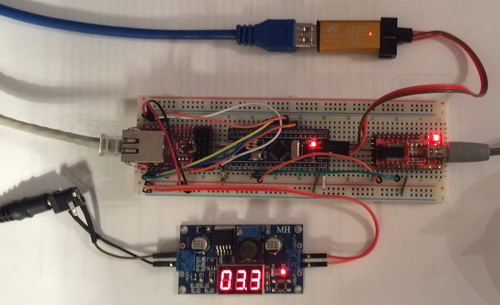
**LAN. ENC28J60. Connect external interrupts (EXTI)**

[*Previous lesson*](http://narodstream.ru/stm-urok-76-hal-ltdc-emwin-multilayer-transparency/) [*Programming STM32 MK*](http://narodstream.ru/programmirovanie-mk-stm32/) [*Next lesson*](http://narodstream.ru/stm-urok-78-hal-ltdc-emwin-window-manager/)

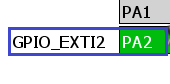
In  [**lesson 74,**](http://narodstream.ru/stm-urok-74-hal-exti-ili-vneshnie-preryvaniya/)  we figured out how to use and process external interrupts on the **STM32** microcontrollers , and we did it mainly in order to use this accumulated experience for programming our network module, since the **ENC28J60** chip , which is its heart, can produce a certain signal At the moment when the buffer for reading is full. This will allow us to eliminate errors that arise from processing incoming packets in an infinite loop, and to eliminate delays, since we will already process these packets immediately after they have appeared in the buffer.

First, I decided to put things in order a little bit in the connections. All that I can put on the breadboard. For this, to the USART adapter, I soldered the bottom of the pins, which also have the necessary contacts GND, TXD and RXD. Since on the LAN module, which we used before, there is no possibility to solder such lines, I took the proprietary module from OLIMEX, executed on the same chip, but already having contacts from below for layout on the breadboard and for now I will use it. ST-Link I will use cheap, because it has a smaller size and copes well with the assigned charms, only the power I removed from it, because I believe that the current that can supply its power supply is not enough for the full operation of the chip ENC28J60. Therefore, as a power supply, we will use a DC-DC down-converter for 3 amperes, which we will supply with a conventional 12 volt power supply with a 5.5 mm connector. You can use any power supply, if only it could withstand a good maximum load current (our 2 amps) and the voltage at its output to exceed 3.3 volts. Now everything looks quite culturally and we are sure that we will not fail. Also contact module**INT is** connected to the leg of the microcontroller **PA2** , since the task of our lesson is the use of external interrupts from the module. Here and so today looks like our scheme (click on the image to enlarge the image)

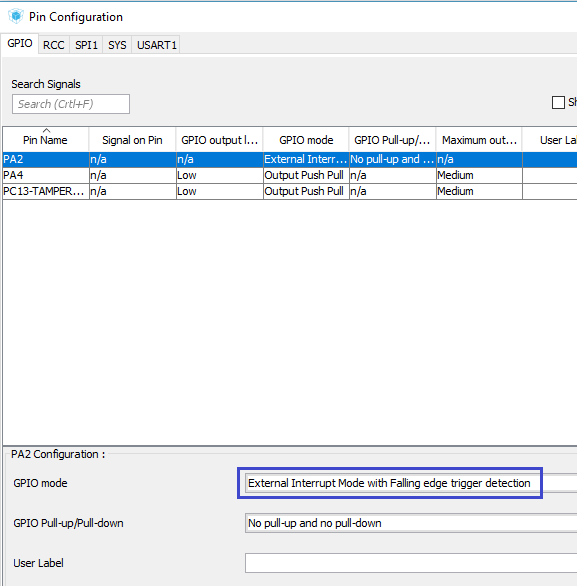
[](http://narodstream.ru/wp-content/uploads/2017/06/image00.jpeg)

The project will be created from the project **ENC28J60\_ARP**[**lesson 71**](http://narodstream.ru/stm-urok-68-lan-enc28j60-arp-chast-1/) and let's call it **ENC28J60\_INT**.

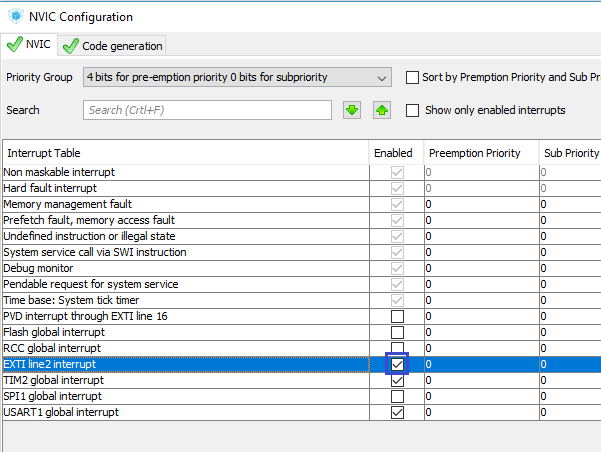
Open this project in the **Cube MX** and turn on the foot **PA2** external interrupts



Let's go to Configuration and configure on our leg external interrupts on the descending front in the GPIO section, the resistor does not tighten anywhere, since we do not have a button but quite conscious levels from the microcircuit



We'll go into **NVIC** and turn on global interrupts on **EXTI2**



Generate the project, open it in Keil, connect all our libraries, and also configure the programmer for auto-cutting. We will try to assemble the project, flash it and test the efficiency of pings and our ARP-requests in the terminal program.

If everything works, then we will continue to write the code.

Comment on the call of the network polling function in an infinite loop, since we will now interrogate our network in the external interrupt handler

/\* USER CODE BEGIN 3 \*/

**//net\_poll();**

}

/\* USER CODE END 3 \*/

Now create the interrupt handler at the bottom of the main module

//----------------------------------------------------------

**void HAL\_GPIO\_EXTI\_Callback(uint16\_t GPIO\_Pin)**

**{**

**}**

/\* USER CODE END 4 \*/

In the body of this handler, we filter the events on a particular foot and call the network polling function

void HAL\_GPIO\_EXTI\_Callback(uint16\_t GPIO\_Pin)

{

**if(GPIO\_Pin== GPIO\_PIN\_2){**

**net\_poll();**

**} else{**

**\_\_NOP();**

**}**

}

And, most importantly, we need not forget that interrupts also need to be solved by the chip itself. Once we wrote this code, but then commented on it.

Let's go to the module enc28j60.c and uncomment this line in the initialization

enc28j60\_SetBank(ECON1);

**enc28j60\_writeOp(ENC28J60\_BIT\_FIELD\_SET,EIE,EIE\_INTIE|EIE\_PKTIE);**

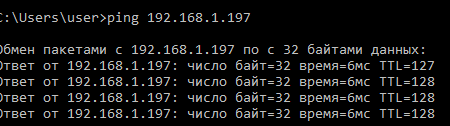
Since we came here, we will also remove the delay after the sending of the package

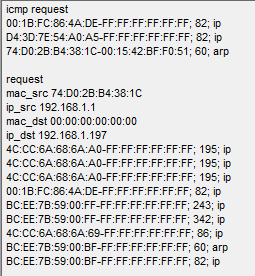
  enc28j60\_writeOp(ENC28J60\_BIT\_FIELD\_SET, ECON1,ECON1\_TXRTS);

~~HAL\_Delay(1);~~

}

Let's try to collect the code and ping our board, and also to interrogate some address using the ARP-request in the terminal program





Everything is working. Now the packages are processed by us, when necessary.

Let's take a little bit of optimization of our network processing, since we spend a lot of time to send bytes to USART, and we do this even before sending response packets, that is, not on time, thereby delaying the processing of the package.

Let's start by processing ARP REQUEST packages.

To return to the **net.c** file in the eth\_read function and first comment out everything related to USART

if(frame->type==ETH\_ARP)

{

**// sprintf(str1,"%02X:%02X:%02X:%02X:%02X:%02X-%02X:%02X:%02X:%02X:%02X:%02X; %d; arp\r\n",**

**// frame->addr\_src[0],frame->addr\_src[1],frame->addr\_src[2],**

**// frame->addr\_src[3],frame->addr\_src[4],frame->addr\_src[5],**

**// frame->addr\_dest[0],frame->addr\_dest[1],frame->addr\_dest[2],**

**// frame->addr\_dest[3],frame->addr\_dest[4],frame->addr\_dest[5],len);**

**// HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

  res = arp\_read(frame,len-sizeof(enc28j60\_frame\_ptr));

  if(res==1)

  {

    arp\_send(frame);

  }

  else if(res==2)

  {

    arp\_table\_fill(frame);

  }

}

if(frame->type==ETH\_IP)

{

**// sprintf(str1,"%02X:%02X:%02X:%02X:%02X:%02X-%02X:%02X:%02X:%02X:%02X:%02X; %d; ip\r\n",**

**// frame->addr\_src[0],frame->addr\_src[1],frame->addr\_src[2],**

**// frame->addr\_src[3],frame->addr\_src[4],frame->addr\_src[5],**

**// frame->addr\_dest[0],frame->addr\_dest[1],frame->addr\_dest[2],**

**// frame->addr\_dest[3],frame->addr\_dest[4],frame->addr\_dest[5],len);**

**// HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

  ip\_read(frame,len-sizeof(ip\_pkt\_ptr));

}

Also, if suddenly there comes a flight package, which is not related to either ARP or IP, we will do a separate processing for it

**else**if(frame->type==ETH\_IP)

{

  // sprintf(str1,"%02X:%02X:%02X:%02X:%02X:%02X-%02X:%02X:%02X:%02X:%02X:%02X; %d; ip\r\n",

  // frame->addr\_src[0],frame->addr\_src[1],frame->addr\_src[2],

  // frame->addr\_src[3],frame->addr\_src[4],frame->addr\_src[5],

  // frame->addr\_dest[0],frame->addr\_dest[1],frame->addr\_dest[2],

  // frame->addr\_dest[3],frame->addr\_dest[4],frame->addr\_dest[5],len);

  // HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);

  ip\_read(frame,len-sizeof(ip\_pkt\_ptr));

}

**else**

**{**

**sprintf(str1,"%02X:%02X:%02X:%02X:%02X:%02X-%02X:%02X:%02X:%02X:%02X:%02X; %d; %04X",**

**frame->addr\_src[0],frame->addr\_src[1],frame->addr\_src[2],frame->addr\_src[3],frame->addr\_src[4],frame->addr\_src[5],**

**frame->addr\_dest[0],frame->addr\_dest[1],frame->addr\_dest[2],frame->addr\_dest[3],frame->addr\_dest[4],frame->addr\_dest[5],**

**len, be16toword(frame->type));**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)"\r\n",2,0x1000);**

**}**

Come now arp.c file to function **arp\_read** and also uberom branch in terms  **ARP\_REQUEST** anything related to **USART**

if(msg->op==ARP\_REQUEST)

{

**// sprintf(str1,"\r\nrequest\r\nmac\_src %02X:%02X:%02X:%02X:%02X:%02X\r\n",**

**// msg->macaddr\_src[0],msg->macaddr\_src[1],msg->macaddr\_src[2],**

**// msg->macaddr\_src[3],msg->macaddr\_src[4],msg->macaddr\_src[5]);**

**// HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**// sprintf(str1,"ip\_src %d.%d.%d.%d\r\n",**

**// msg->ipaddr\_src[0],msg->ipaddr\_src[1],msg->ipaddr\_src[2],msg->ipaddr\_src[3]);**

**// HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**// sprintf(str1,"mac\_dst %02X:%02X:%02X:%02X:%02X:%02X\r\n",**

**// msg->macaddr\_dst[0],msg->macaddr\_dst[1],msg->macaddr\_dst[2],**

**// msg->macaddr\_dst[3],msg->macaddr\_dst[4],msg->macaddr\_dst[5]);**

**// HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**// sprintf(str1,"ip\_dst %d.%d.%d.%d\r\n",**

**// msg->ipaddr\_dst[0],msg->ipaddr\_dst[1],msg->ipaddr\_dst[2],msg->ipaddr\_dst[3]);**

**// HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

  res=1;

}

Now we go to the **arp\_send** function and here after sending the packet we will display all the information and show it in the terminal, not forgetting that the source with the receiver has now been swapped

  eth\_send(frame,sizeof(arp\_msg\_ptr));

**sprintf(str1,"%02X:%02X:%02X:%02X:%02X:%02X(%d.%d.%d.%d)-",**

**frame->addr\_dest[0],frame->addr\_dest[1],frame->addr\_dest[2],frame->addr\_dest[3],frame->addr\_dest[4],frame->addr\_dest[5],**

**msg->ipaddr\_dst[0],msg->ipaddr\_dst[1],msg->ipaddr\_dst[2],msg->ipaddr\_dst[3]);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**sprintf(str1,"%02X:%02X:%02X:%02X:%02X:%02X(%d.%d.%d.%d) arp request\r\n",**

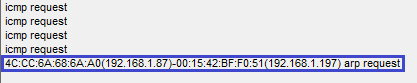
**frame->addr\_src[0],frame->addr\_src[1],frame->addr\_src[2],frame->addr\_src[3],frame->addr\_src[4],frame->addr\_src[5],**

**msg->ipaddr\_src[0],msg->ipaddr\_src[1],msg->ipaddr\_src[2],msg->ipaddr\_src[3]);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

}

Sending a ping to our board, we will have to come at least one ARP-request to the terminal program



Let's further optimize the request with the **ICMP** response in the same way .

In function **icmp\_read** also we will comment out all superfluous

if((len>=sizeof(icmp\_pkt\_ptr))&&(icmp\_pkt->msg\_tp==ICMP\_REQ))

{

**//sprintf(str1,"icmp request\r\n");**

**//HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

  icmp\_pkt->msg\_tp=ICMP\_REPLY;

And we will display all the information in the same function, but only after sending the package (mac-addresses in this case are no longer needed)

  ip\_send(frame,len+sizeof(ip\_pkt\_ptr));

**sprintf(str1,"%d.%d.%d.%d-%d.%d.%d.%d icmp request\r\n",**

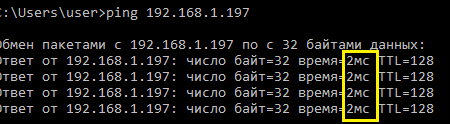
**ip\_pkt->ipaddr\_dst[0],ip\_pkt->ipaddr\_dst[1],ip\_pkt->ipaddr\_dst[2],ip\_pkt->ipaddr\_dst[3],**

**ip\_pkt->ipaddr\_src[0],ip\_pkt->ipaddr\_src[1],ip\_pkt->ipaddr\_src[2],ip\_pkt->ipaddr\_src[3]);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

}

We will collect the code, we'll sew the controller and see the result in the terminal program after sending the ping to our module, and also see the response time



We see that the response time has become much less, although using AVR was even less. Perhaps this is due to the code of the HAL library, in avr we accessed the registers directly.

In this lesson, we solved 2 tasks at once: using external interrupts to receive and process packets from the network module in a timely manner, and optimizing the code by eliminating unnecessary parcels in the USART bus.

Thank you all for attention!