AN5523

Application note

SPC58x Ethernet wake-up cases

Introduction

The purpose of this document is to show how to wake up an SPC58x automotive microcontroller MCU by using the Ethernet peripheral.

This application will focus on two main different approaches. The first one is fully based on the Ethernet controller's capability to manage the wake-up event, for instance the magic frame can wake up the controller.

The second approach shows how to enter a dedicated low-power mode available for this MCU family and wake up from the Ethernet related receive signals. To obtain more details on low-power states, the user can refer to the official technical note released for these products as well as to the related reference manual (see Section Appendix B Reference documents).

The entire application has been built on top of the SPC5Studio (https://www.st.com/en/development-tools/spc5-studio.html).

1 MAC controller overview

SPC58x automotive microcontrollers feature a quality-of-service 10/100 Mbit/s Ethernet peripheral that implements the Medium Access Control (MAC) layer plus several internal modules for standard and advanced network features that can be summarized into the following categories:

- MAC core and MAC Transaction Layer (MTL)
- DMA engine
- SMA interface
- MAC management counters
- Power Management Block (PMT)

Details about the Ethernet controller programming can be found in the Reference Manual (see Section Appendix B Reference documents).

2 PMT block

The Ethernet controller has its power management (PMT) block designed to support the reception of network (remote) wake-up packets and magic packets.

The hardware is able to generate interrupts for remote wake-up packets and magic packets that the MAC receives. When the power-down mode is enabled, the MAC drops all received packets. Refer to the related chapter in the microcontroller's reference manual (see Section Appendix B Reference documents) for all the necessary steps that must be followed to program the PMT block in order to manage the wake-up event.

This application will focus on the magic frame event.

Note: The magic packet feature is based on the Magic Packet Technology white paper.

3 Ethernet pin selection

The Ethernet controller 1 has been used because on this board the RXDATA pins can be mapped on the wake-up unit as shown in the table below.

Table 1. **Receive pins used to wake-up**

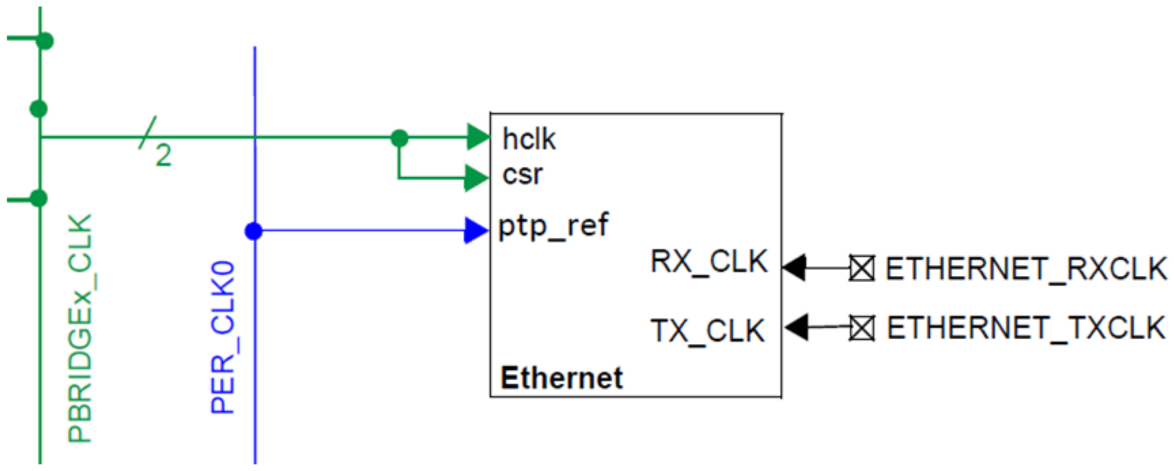
Port	SIUL MSCR#	MSCR SSS	Function	Module
PA[10]	973	00000001	RDATA0	Ethernet 1
PA[10]	-	-	INT23	Wake-up
PA[11]	974	00000001	RDATA1	Ethernet 1
PA[11]	-	-	INT24	Wake-up
PE[10]	976	00000001	RDATA3	Ethernet 1
PE[10]	-	-	INT25	Wake-up

Other signals are mapped according to the IO list available for the microcontroller. The SPC5Studio suite also offers a graphical interface to configure all the pins.

4 Ethernet clock

The PMT block does not perform the clock gate function, so the application is delegated to disable and enable the hclk; this is the clock provided to the Ethernet controller as shown in the figure below:

Figure 1. SPC58xEx/SPC58xGx Ethernet clocks



To gate off the clock, the related PCTL register must be properly programmed. The following code is an example of the API used to change the clock mode for the peripheral.

More details about the PCTL block can be found in the microcontroller's reference manual (see Section Appendix B Reference documents).

```
void ethernet_gate_clock(unsigned int mode)
{
  SPCSetPeripheralClockMode(SPC5_ETH1_PCTL,
    SPC5_ME_PCTL_RUN(mode) |
    SPC5_ME_PCTL_LP(mode));
}
```

Where, for this MCU and this Ethernet instance:

```
#define SPC5_ETH1_PCTL 233
```

5 Ethernet interrupts

The interrupt can be generated as a result of various events in the Ethernet IP controller, including the wake-up ones.

The following list shows two interrupts used in this application for the Ethernet 1.

- *218 vector* -> **sbd_intr_o**: this is the combined interrupts from all MAC/DMA/MTL sources.
- *219 vector* -> **pmt_intr_o**: the Ethernet module asserts this signal to exit from the power-down mode when the clock is gated off. This is synchronous to the rx clock.

The following code shows the interrupt service routines associated to the two interrupts signals mentioned in the previous list and the related low level APIs invoked to service the events.

```
IRQ_HANDLER(SPC5_ETH1_CORE_HANDLER) {
    IRQ_PROLOGUE();

    dwmac_core_dma_main_isr(&drv_instance[SPC5_ETH1_INSTANCE]);

    IRQ_EPILOGUE();
}

IRQ_HANDLER(SPC5_ETH1_PMT_HANDLER) {
    IRQ_PROLOGUE();

    dwmac_pmt_irq_handler(&drv_instance[SPC5_ETH1_INSTANCE]);

    IRQ_EPILOGUE();
}
```

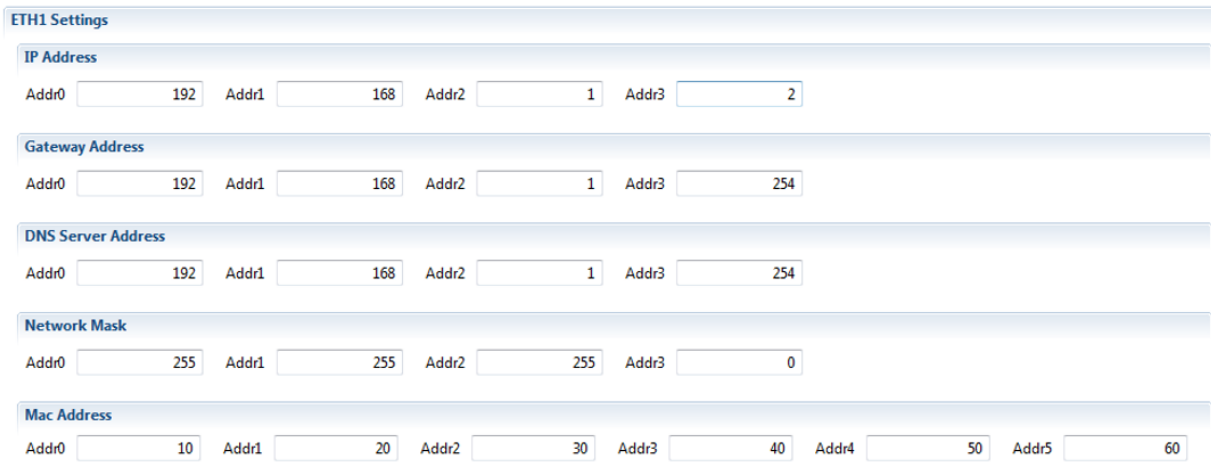
The `dwmac_core_dma_main_isr` will be invoked on sbd_intr_o while `dwmac_pmt_irq_handler` on pmt_intr_o.

6 FreeRTOS network configuration

The application is based on the FreeRTOS operating system with the TCP stack. As soon as the FreeRTOS TCP network starts, the Ethernet instance 1 is activated and configured. The *0a:14:1e:28:32:3c* MAC address is assigned to the interface. The default SPC5Studio TCP stack configuration has been adopted for this project.

The following picture shows the SPC5Studio graphical interface used for configuring the Ethernet at the time of writing,.

Figure 2. **SPC5Studio Network component**



ETH1 Settings

IP Address

Addr0	192	Addr1	168	Addr2	1	Addr3	2

Gateway Address

Addr0	192	Addr1	168	Addr2	1	Addr3	254

DNS Server Address

Addr0	192	Addr1	168	Addr2	1	Addr3	254

Network Mask

Addr0	255	Addr1	255	Addr2	255	Addr3	0

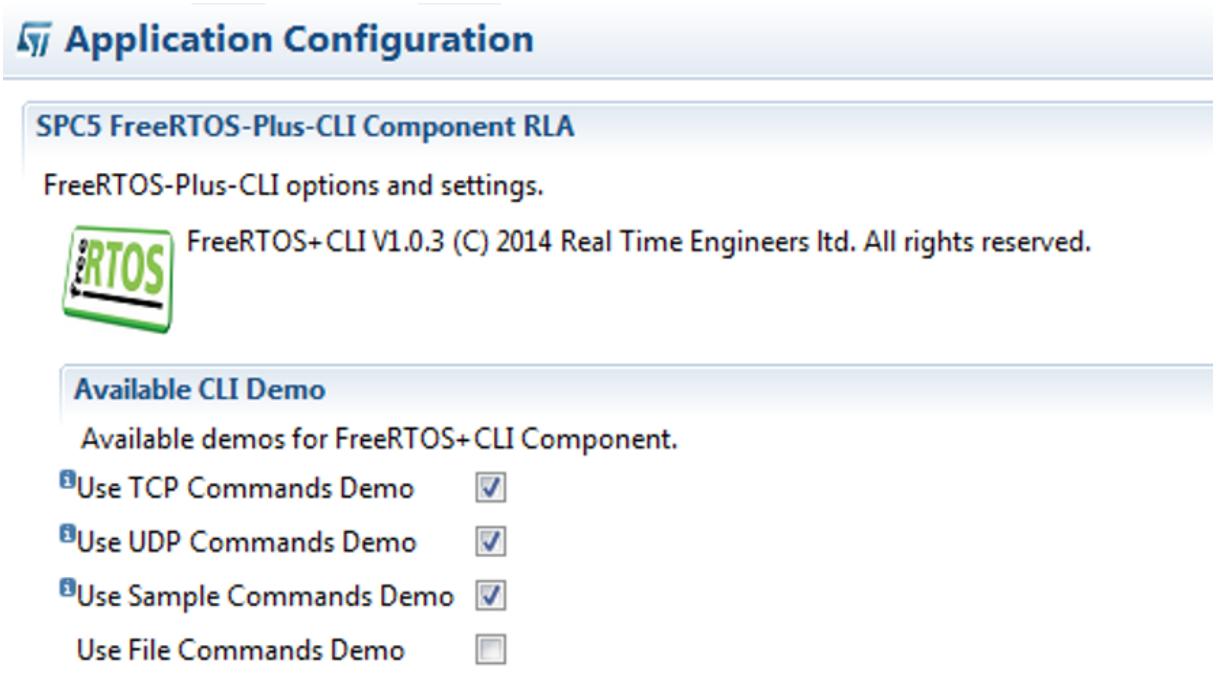
Mac Address

Addr0	10	Addr1	20	Addr2	30	Addr3	40	Addr4	50	Addr5	60

7 FreRTOS CLI command

The CLI support must be enabled in FreeRTOS to offer a *TELNET* connection where the user can execute several commands available in the suite as well as their own commands, for example, to enter in low-power state and set up the wake-up mode.

Figure 3. **SPC5Studio FreeRTOS+CLI wizard**



Application Configuration

SPC5 FreeRTOS-Plus-CLI Component RLA

FreeRTOS-Plus-CLI options and settings.

FreeRTOS+CLI V1.0.3 (C) 2014 Real Time Engineers ltd. All rights reserved.

Available CLI Demo

Available demos for FreeRTOS+CLI Component.

Use TCP Commands Demo	✓
Use UDP Commands Demo	✓
Use Sample Commands Demo	✓
Use File Commands Demo	

All the application registers need CLI commands as shown below when the Ethernet link is established. This function can be called inside the `main`.

```
void vDWMACEthCLICommands( void )
{
    …
    FreeRTOS_CLIRegisterCommand( &xEthWolParameter );
    FreeRTOS_CLIRegisterCommand( &xEthStbyParameter );
}
```

The `FreeRTOS_CLIRegisterCommand` is implemented in the FreeRTOS_CLI.c file.

The following code shows the standby command implementation.

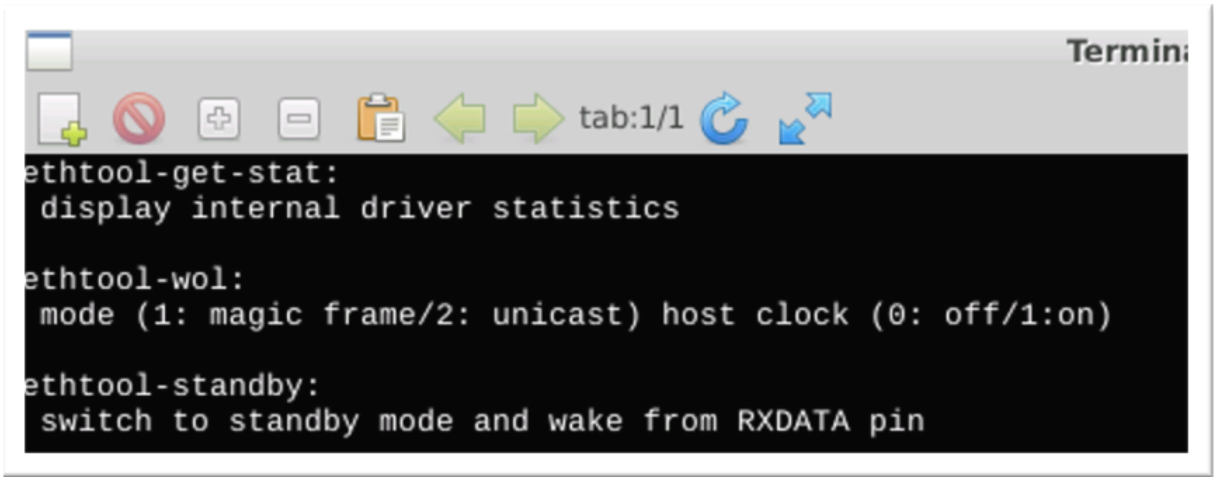
```
static BaseType_t prvStby( char *pcWriteBuffer, size_t xWriteBufferLen,
   const char *pcCommandString )
{
   const char *pcParameter;
   BaseType_t xParameterStringLength, xReturn = pdFALSE;
   static BaseType_t lParameterNumber = 0;

   sprintf( pcWriteBuffer, "---------- Standby Mode ----------\r\n" );
   MainGoStandBy(); /* Main routine to enter in standby mode. */
   return pdFALSE;
}

static const CLI_Command_Definition_t xEthStbyParameter =
{
  "ethtool-standby",
  "\r\nethtool-standby:\r\n switch to standby mode and wake from RXDATA
    pin\r\n",
   prvStby, /* The function to run. */
   0 /* No parameters are expected. */
};
```

The following code shows the WoL command implementation.

```
static BaseType_t prvWoL( char *pcWriteBuffer, size_t xWriteBufferLen, const char
*pcCommandString )
{
const char *pcParameter;
BaseType_t xParameterStringLength, xReturn = pdFALSE;
static BaseType_t lParameterNumber = 0;

    ( void ) pcCommandString;
    ( void ) xWriteBufferLen;
    configASSERT( pcWriteBuffer );

    memset( pcWriteBuffer, 0x00, xWriteBufferLen );

    if( lParameterNumber == 0 )
    {
        sprintf( pcWriteBuffer, "---------- Wake-up On LAN ----------\r\n" );
        lParameterNumber = 1L;
        return pdTRUE;
    }
    pcParameter = FreeRTOS_CLIGetParameter(
            pcCommandString, lParameterNumber, &xParameterStringLength);

    if( lParameterNumber == 1L ) {
        wol =  atoi(pcParameter);
        if (wol == 1)
            sprintf( pcWriteBuffer, "Magic Packet\r\n" );
        else if (wol == 2)
            sprintf( pcWriteBuffer, "Global Unicast\r\n" );
        else
            sprintf( pcWriteBuffer, "Invalid Configuration\r\n" );
        lParameterNumber++;
        xReturn = pdTRUE;
    } else if( lParameterNumber == 2L ) {
        clk =  atoi(pcParameter);
        if (clk == 1)
            sprintf( pcWriteBuffer, "Host Clock ON\r\n" );
        else
            sprintf( pcWriteBuffer, "Host Clock OFF\r\n" );
        xReturn = pdFALSE;
        lParameterNumber = 0;
        /* Invoked Network callbabk to go in suspend mode. */
        xNetworkInterfaceSetWol(wol, clk);
    }
    return xReturn;
}

static const CLI_Command_Definition_t xEthWolParameter =
{
"ethtool-wol",
"\r\nethtool-wol:\r\n mode (1: magic frame/2: unicast) host clock (0:off/1:on)\r\n",
prvWoL, /* The function to run. */
2
};
```

This is an example of Telnet connection to the target: 192.168.1.2 on TCP port 23.

Figure 4. Telnet connection to the target



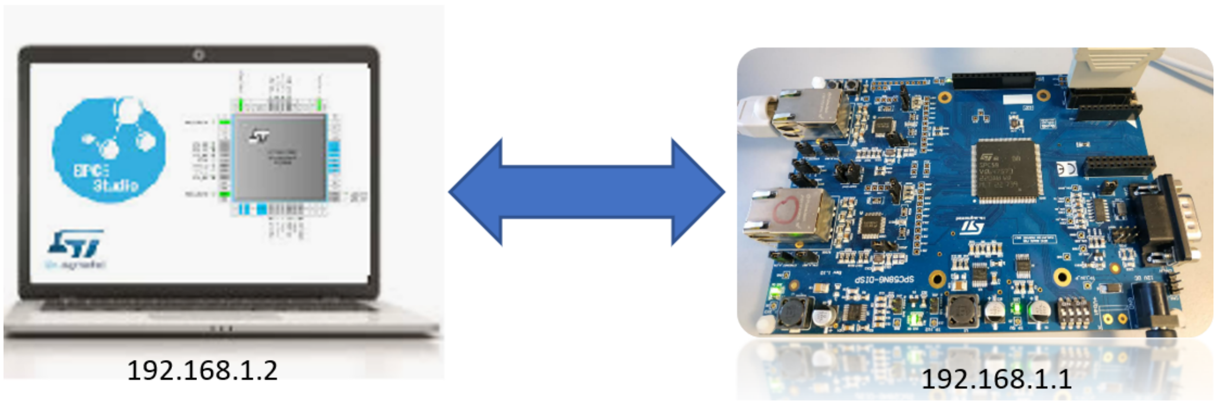
```
ethtool-get-stat:
 display internal driver statistics

ethtool-wol:
 mode (1: magic frame/2: unicast) host clock (0: off/1:on)

ethtool-standby:
 switch to standby mode and wake from RXDATA pin
```

8 The magic packet

The *magic packet* is a broadcast frame containing anywhere within its payload 6 bytes of all 255 (*FF:FF:FF:FF:FF:FF*), followed by sixteen repetitions of the target computer's 48-bit MAC address, for a total of 102 bytes.

A standard magic packet has some basic limitations, for example: the destination MAC address must be known, there is no delivery confirmation and the hardware must support the related detection. To test the magic frame, a private network has been set up. This can prevent the magic frame from being dropped by other network components like the router.

Figure 5. Network setup



8.1 Generating a magic frame

Different operating systems offer tools to generate this special packet; on the Linux machine, which has been used for this application as the reference host, the following tools can be installed (these are available on several Linux distributions):

- `wol` - Wake On LAN client
- `ether-wake` - A tool to send a Wake-On-LAN "Magic Packet"

More details about these commands can be found in the related manual pages.

The user can switch to different systems and other tools to generate the magic frame.

9 Application details

The application covers the following wake-up user cases:

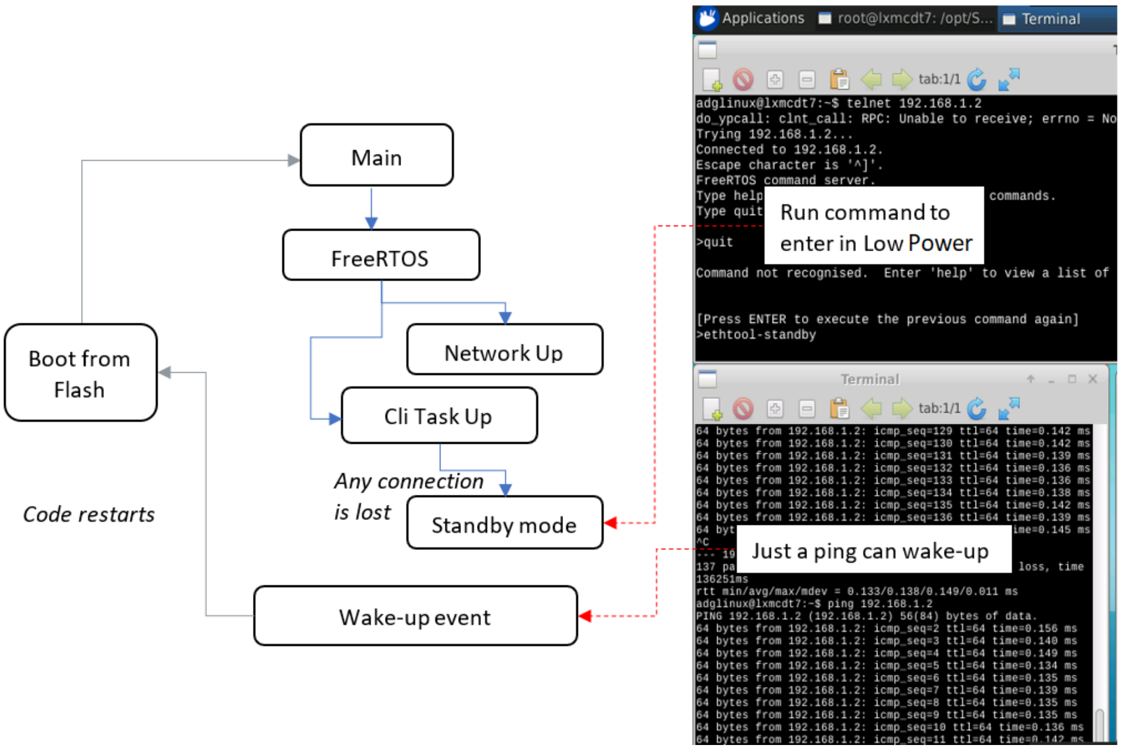
- The system goes in STANDBY mode, the wake-up occurs from RX data pins.
- The Ethernet controller enters power-down mode and can be woken up by using the magic packet or unicast frames. In this test, the host clock can be switched off as well.

As a prerequisite for all cases the PHY transceiver must be kept enabled.

9.1 STANDBY test

After the application starts and the whole network is configured via the command, line support, the `ethtool-standby` invokes a dedicated function to program the WakeUp unit and switch to STANDBY mode. Any frames passed by the transceiver to the Ethernet will wake up the system. In this example the ping command is used.

Figure 6. STANDBY flow diagram



```
adglinux@lxmcdt7:~$ telnet 192.168.1.2
do_ypcall: clnt_call: RPC: Unable to receive; errno = No
Trying 192.168.1.2...
Connected to 192.168.1.2.
Escape character is '^]'.
FreeRTOS command server.
Type help                                  commands.
Type quit

>quit

Command not recognised.  Enter 'help' to view a list of


[Press ENTER to execute the previous command again]
>ethtool-standby
```

```
64 bytes from 192.168.1.2: icmp_seq=129 ttl=64 time=0.142 ms
64 bytes from 192.168.1.2: icmp_seq=130 ttl=64 time=0.142 ms
64 bytes from 192.168.1.2: icmp_seq=131 ttl=64 time=0.139 ms
64 bytes from 192.168.1.2: icmp_seq=132 ttl=64 time=0.136 ms
64 bytes from 192.168.1.2: icmp_seq=133 ttl=64 time=0.136 ms
64 bytes from 192.168.1.2: icmp_seq=134 ttl=64 time=0.138 ms
64 bytes from 192.168.1.2: icmp_seq=135 ttl=64 time=0.142 ms
64 bytes from 192.168.1.2: icmp_seq=136 ttl=64 time=0.139 ms
64 byt                                       ime=0.145 ms
^C
--- 19
137 pa                                        loss, time
136251ms
rtt min/avg/max/mdev = 0.133/0.138/0.149/0.011 ms
adglinux@lxmcdt7:~$ ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.156 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.140 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=64 time=0.149 ms
64 bytes from 192.168.1.2: icmp_seq=5 ttl=64 time=0.134 ms
64 bytes from 192.168.1.2: icmp_seq=6 ttl=64 time=0.135 ms
64 bytes from 192.168.1.2: icmp_seq=7 ttl=64 time=0.139 ms
64 bytes from 192.168.1.2: icmp_seq=8 ttl=64 time=0.135 ms
64 bytes from 192.168.1.2: icmp_seq=9 ttl=64 time=0.135 ms
64 bytes from 192.168.1.2: icmp_seq=10 ttl=64 time=0.136 ms
64 bytes from 192.168.1.2: icmp_seq=11 ttl=64 time=0.142 ms
```

9.1.1 Programming the wake-up unit

This hardware module is programmed to wake up as soon as the data is present on the Ethernet receive signals: RXDATA0, RXDATA1 and RXDATA3. These are the only signals mapped in this configuration and for this hardware setup.

After the wake-up event, the application will start from scratch from the Flash device.

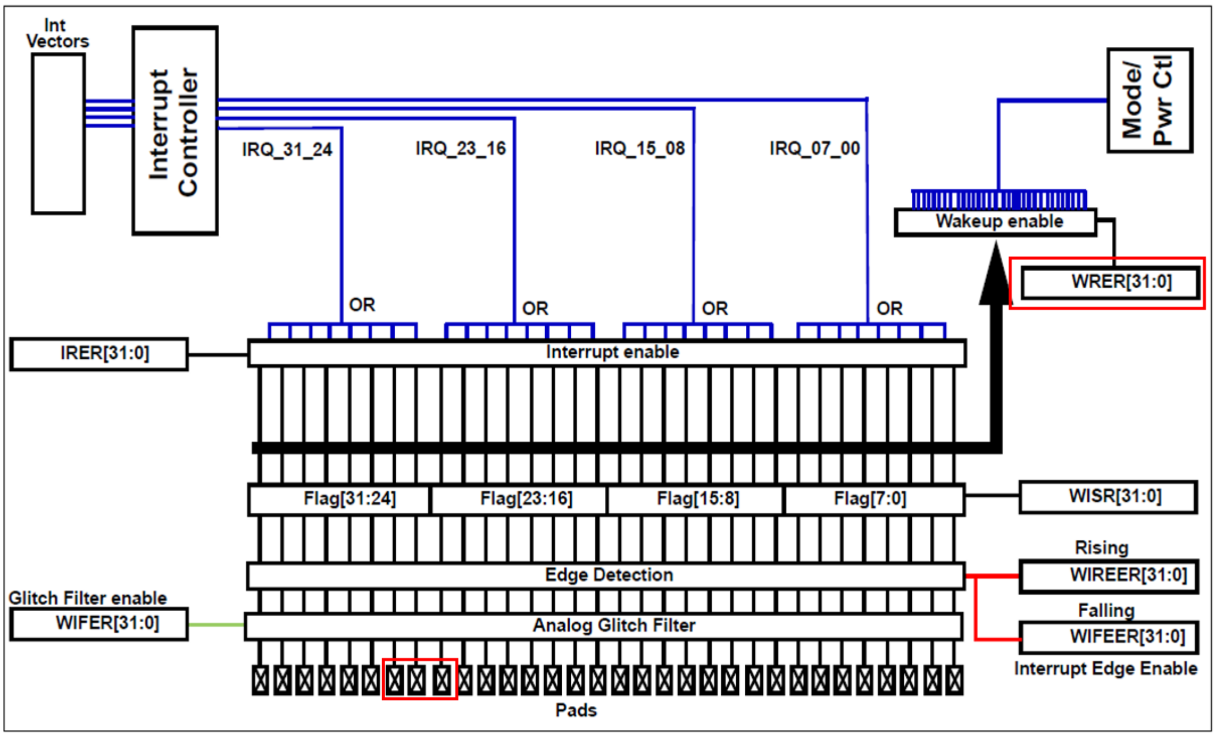
The configuration can be changed to use the STOP and HALT modes, paying attention to program the low-power states as expected and with the proper parameters: for instance keeping the Flash enabled.

The wake-up unit could raise an interrupt, for example when the HALT mode is configured by programming: WKPU.IRER.B.EIRE<xx> = 1;

For further details about the wake-up unit and related programming, refer to the official technical note and reference manual (see Section Appendix B Reference documents).

This application only covers the STANDBY user case.

Figure 7. Wake-up unit



This is the function to enter in STANDBY mode:

```
void MainGoStandBy(void)
{
    /* Peripheral OFF in every mode */
    MC_ME.RUN_PC[0].R = SPC5_ME_RUN_PC0_BITS;
    MC_ME.RUN_PC[1].R = SPC5_ME_RUN_PC0_BITS;
    MC_ME.RUN_PC[2].R = SPC5_ME_RUN_PC0_BITS;
    clockInit();
    MC_ME.RUN_MC[3].R = 0x00130030; /* Stop PLL */
    clockInit();
    MC_ME.RUN_MC[3].R = 0x00130010; /* Stop XOSC */
    clockInit();

    /* Configure wake-up line 23 (on ETH1: RXDATA0 - PA[11] pad) to wake-up */
    WKPU.WIFER.B.IFE23 = 1;    /* Filter Enabled*/
    WKPU.WIFEER.B.IFEE23 = 1;    /* Enable Wake-up on Falling Edge */
    WKPU.WIREER.B.IREE23 = 1;    /* Enable Rising Edge Wake-up Interrupt */
    /* WKPU.IRER.B.EIRE23 = 1;      Wake-up Interrupt Request Enable (HALT mode) */
    WKPU.WRER.B.WRE23 = 1;         /* Wake-up request enable */

    /* RDATA1 can be also configured to wakeup */
    WKPU.WIFER.B.IFE24 = 1;    /* Filter Enabled*/
    WKPU.WIFEER.B.IFEE24 = 1;    /* Enable Wake-up on Falling Edge */
    WKPU.WIREER.B.IREE24 = 1;    /* Enable Rising Edge Wake-up Interrupt */
    /* WKPU.IRER.B.EIRE24 = 1;       Wake-up Interrupt Request Enable (HALT mode) */
    WKPU.WRER.B.WRE24 = 1;         /* Wake-up request enable */

    /* RDATA3 can be also configured to wakeup */
    WKPU.WIFER.B.IFE25 = 1;        /* Filter Enabled*/
    WKPU.WIFEER.B.IFEE25 = 1;    /* Enable Wake-up on Falling Edge */
    WKPU.IRER.B.EIRE25 = 1;        /* Wake-up Interrupt Request Enable */
    WKPU.WRER.B.WRE25 = 1;         /* Wake-up request enable */

    WKPU.WISR.R = 0xffffffff;
    if (SPCSetRunMode(SPC5_RUNMODE_STANDBY0) == CLOCK_FAILED)
      SPC5_CLOCK_FAILURE_HOOK();
}
```

Where:

- `SPCSetRunMode` is the API to switch the system to the specified run mode;
- `clockInit` is for the clock initialization.

Both of these APIs are implemented in the SPC5Studio. An important goal of this user case is to obtain the best low-power condition while entering in standby mode. This application does not show any measurement of low-power consumption, please refer to official channel to get the criteria and the related figures. When resuming one of the following conditions must be true, this means that the wake-up has been actually done by one of the expected RXDATA lines.

- `WKPU.WISR.B.EIF23 == 1`
- `WKPU.WISR.B.EIF24 == 1`
- `WKPU.WISR.B.EIF25 == 1`

9.2 Wake-up from magic frame

The Ethernet controller can enter in power down mode by enabling the `PWRDWN` bit in the `MAC_PMT_CONTROL_STATUS` register as described in the reference manual (see Section Appendix B Reference documents).

When entering in power down the Ethernet receiver drops all received packets until it receives the expected magic packet or remote wake-up packet.

This application tests the magic packet and shows how to also use the global unicast mode as a wake up event.

When configuring the unicast mode, every incoming frame will wake-up the controller, e.g.: a ping to the target.

When configuring magic packet, only this broadcast frame will be able to wake up the Ethernet.

The RX_CLK from the MII interface must be kept enabled in these modes. Refer to the related technical note for MII interface (see Section Appendix B Reference documents).

In the Application, after programming WoL, the TX_CLK can be turned off (as well as the PTP clock) but the `MAC_CONFIGURATION.B.RE` must be 1.

When the Ethernet goes in POWERDOWN the application covers the following scenarios:

- The host clock is not gated-off
  – the **sbd_intr_o** interrupt is raised to wake-up. As soon as the magic packet is received, the Interrupt service routine (ISR) will check the `MAC_INTERRUPT_STATUS.B.PMTIS`, clearing the status and re-enabling the DMA, TX_CLK and transmission. The Network will restart working.
- The host lock is gated-off
  – the **pmt_intr_o** is the interrupt that will be raised to wake-up. This has to enable the peripheral via PCTL in the ISR.

The MCU reference manual describes in the related chapter all the mandatory steps to do before and after entering in power-down (see Section Appendix B Reference documents).

9.2.1 Controller in power-down

As soon as the whole network stack starts, by using the CLI command, the low-level driver can be invoked to program. The desired wake-up mode in the related Ethernet registers. The controller will enter in power down so any network activity will stop. Before entering in power down, different approaches can be followed, e.g. the low level driver can release its own resources which were allocated (descriptor rings, buffers). The application is designed to invoke a suspend private callback in order to do other jobs (i.e. turn off another peripheral or subsystem).

After receiving the magic packet, the related interrupt will be raised so the interrupt service routine will resume the Ethernet. If the software had previously released the Ethernet resources, at this stage everything must be restored in order to make the whole network stack work again.

If the host clock has been switched off, the ISR will have to enable it.

The following figure summarizes the flow sequence followed for putting the controller in power down.

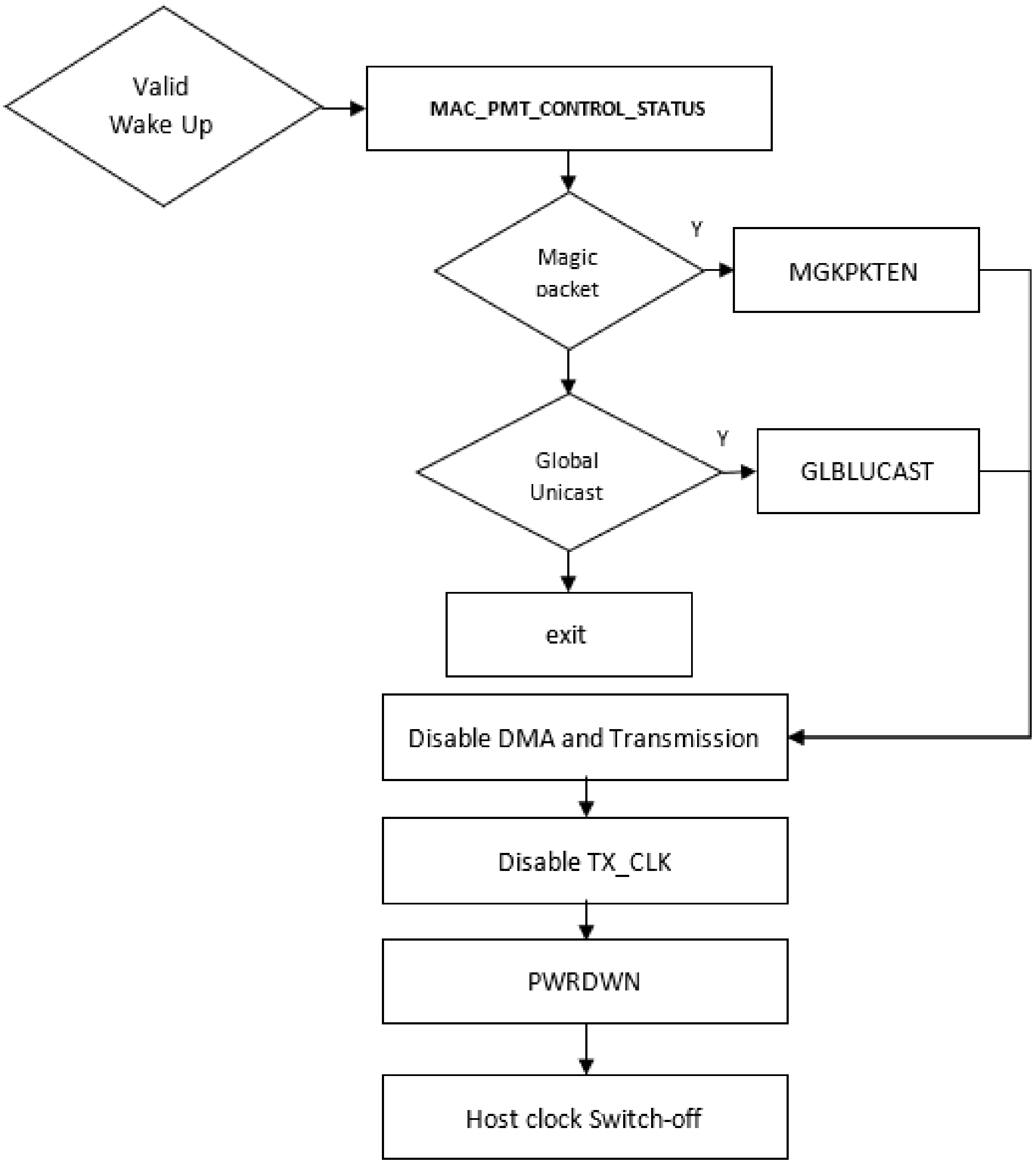


Figure 8. **WoL flow diagram**

The code below is an example of the entry point in the low-level driver to program the wake-up mode and enter in power down.

```
void dwmac_netdrv_setwol( netdrv_ops_t *ops, BaseType_t WakeMode, BaseType_t clk_en)
{
    dwmac_qos_t *dwmac;
    dwmac = (dwmac_qos_t *) ops->priv;

    if (WakeMode != NO_WOL) {
        unsigned int ret;

        dwmac->pmt = WakeMode;

        /* Set the WoL mode and enable the power-down for the core. */
        ret = dwmac->mac->pmt(dwmac);
        if (!ret) {

            /* If available, suspend can do further job... */
            if (ops->suspend != NULL)
                ops->suspend();
     /* Disabling host clock, the PMT interrupt will wake-up
      * the system on magic frame event and the Network will
      * start from scratch.
      * The core interrupt will not be raised in this case.
      */
            if ((clk_en == 0) && (ops->hclk_en != NULL)) {
                ops->hclk_en(false);
                return;
            }
            /* Reception must be kept enabled. */
            dwmac->reg->MAC_CONFIGURATION.B.TE = 0;
            dwmac->dma->start_tx(dwmac, false, DMA_CH0);
            dwmac->dma->start_rx(dwmac, false, DMA_CH0);
            /* driver resources could be released at this point... */
        }
    }
}
```

9.2.2 Wake-up with clock disabled

In this scenario, before entering in power down, the host clock is switched off by using the PTCL (as detailed in the related paragraph).

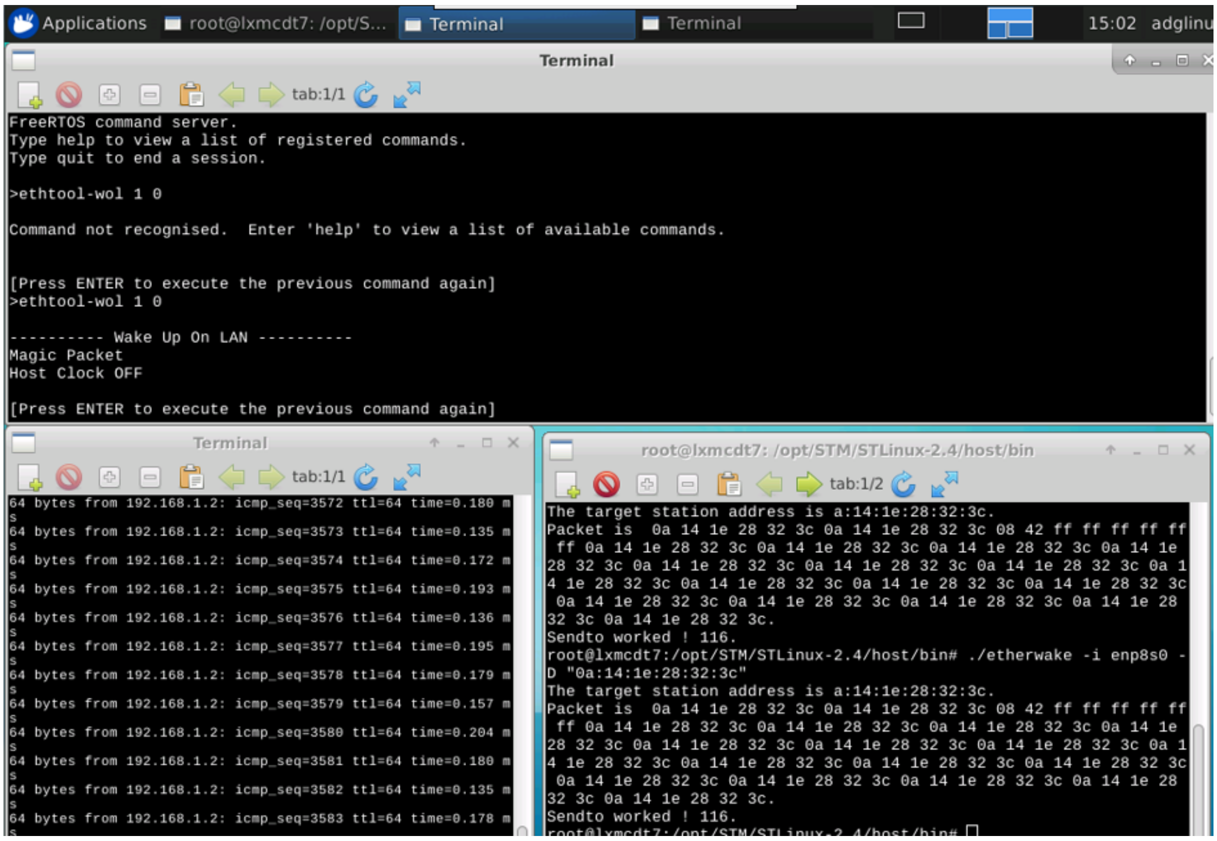
None of the Ethernet registers will be accessible anymore. Only pmt_intr_ocan will wake up the controller.

As soon as the magic frame is detected, the related ISR will enable the clock and resume the interface as shown in the function below:

```
static void dwmac_pmt_irq_handler(dwmac_qos_t *dwmac)
{
       uint32_t status;

       netdrv_ops_t *ops = (netdrv_ops_t *) dwmac->netdev;

       dwmac->core_irq_stat.pmt_irq++;

       if (ops->resume != NULL) {
            ops->resume();
       }
       if (ops->hclk_en != NULL) {
                 ops->hclk_en(true);
       }
}
```

Figure 9. Host view to program the target to send the magic packet shows an example of how to enter in power down with the clock switched off. In another terminal it is visible when the magic frame is sent to wake up the target.

Figure 9. Host view to program the target to send the magic packet



```
FreeRTOS command server.
Type help to view a list of registered commands.
Type quit to end a session.

>ethtool-wol 1 0

Command not recognised.  Enter 'help' to view a list of available commands.


[Press ENTER to execute the previous command again]
>ethtool-wol 1 0

---------- Wake Up On LAN ----------
Magic Packet
Host Clock OFF

[Press ENTER to execute the previous command again]
```

```
64 bytes from 192.168.1.2: icmp_seq=3572 ttl=64 time=0.180 m
s
64 bytes from 192.168.1.2: icmp_seq=3573 ttl=64 time=0.135 m
s
64 bytes from 192.168.1.2: icmp_seq=3574 ttl=64 time=0.172 m
s
64 bytes from 192.168.1.2: icmp_seq=3575 ttl=64 time=0.193 m
s
64 bytes from 192.168.1.2: icmp_seq=3576 ttl=64 time=0.136 m
s
64 bytes from 192.168.1.2: icmp_seq=3577 ttl=64 time=0.195 m
s
64 bytes from 192.168.1.2: icmp_seq=3578 ttl=64 time=0.179 m
s
64 bytes from 192.168.1.2: icmp_seq=3579 ttl=64 time=0.157 m
s
64 bytes from 192.168.1.2: icmp_seq=3580 ttl=64 time=0.204 m
s
64 bytes from 192.168.1.2: icmp_seq=3581 ttl=64 time=0.180 m
s
64 bytes from 192.168.1.2: icmp_seq=3582 ttl=64 time=0.135 m
s
64 bytes from 192.168.1.2: icmp_seq=3583 ttl=64 time=0.178 m
s
```

```
The target station address is a:14:1e:28:32:3c.
Packet is  0a 14 1e 28 32 3c 0a 14 1e 28 32 3c 08 42 ff ff ff ff ff
 ff 0a 14 1e 28 32 3c 0a 14 1e 28 32 3c 0a 14 1e 28 32 3c 0a 14 1e
28 32 3c 0a 14 1e 28 32 3c 0a 14 1e 28 32 3c 0a 14 1e 28 32 3c 0a 1
4 1e 28 32 3c 0a 14 1e 28 32 3c 0a 14 1e 28 32 3c 0a 14 1e 28 32 3c
 0a 14 1e 28 32 3c 0a 14 1e 28 32 3c 0a 14 1e 28 32 3c 0a 14 1e 28
32 3c 0a 14 1e 28 32 3c.
Sendto worked ! 116.
root@lxmcdt7:/opt/STM/STLinux-2.4/host/bin# ./etherwake -i enp8s0 -
D "0a:14:1e:28:32:3c"
The target station address is a:14:1e:28:32:3c.
Packet is  0a 14 1e 28 32 3c 0a 14 1e 28 32 3c 08 42 ff ff ff ff ff
 ff 0a 14 1e 28 32 3c 0a 14 1e 28 32 3c 0a 14 1e 28 32 3c 0a 14 1e
28 32 3c 0a 14 1e 28 32 3c 0a 14 1e 28 32 3c 0a 14 1e 28 32 3c 0a 1
4 1e 28 32 3c 0a 14 1e 28 32 3c 0a 14 1e 28 32 3c 0a 14 1e 28 32 3c
 0a 14 1e 28 32 3c 0a 14 1e 28 32 3c 0a 14 1e 28 32 3c 0a 14 1e 28
32 3c 0a 14 1e 28 32 3c.
Sendto worked ! 116.
root@lxmcdt7:/opt/STM/STLinux-2.4/host/bin# 
```

9.2.3 Wake-up with clock enabled

In this case, when the host controller clock is kept enabled, it is important to notice that the wake-up event will not be managed by the pmt_intr_o.

When the magic frame is received, the sbd_intr_o signal will be asserted and the main interrupt service routine (`dwmac_core_dma_main_isr`) will be invoked.

This is the code portion which manages the code designed to run the wake-up in this scenario:

```
static void dwmac_core_dma_main_isr( dwmac_qos_t *dwmac )
{
    if( dwmac->mac->isr )
    {
        uint32_t Core_Irq_status;

        Core_Irq_status = dwmac->mac->isr( dwmac );
        if (Core_Irq_status == core_irq_pmt) {
            netdrv_ops_t *ops = (netdrv_ops_t *) dwmac->netdev;
            dwmac->dma->start_tx(dwmac, true, DMA_CH0);
            dwmac->dma->start_rx(dwmac, true, DMA_CH0);

            if (ops->resume != NULL)
            ops->resume();
…
```

This is the low-level driver function that checks the status register:

```
 static int dwmac_qos_core_isr(dwmac_qos_t * priv)
{
  uint32_t status = priv->reg->MAC_INTERRUPT_STATUS.B.PMTIS;
  int ret = core_irq_done;

  /* Clear the interrupt by reading the PMT status register
   * This is raised by sbd_intr_o so when clock is not gated off.
   */
  if (priv->pmt && status) {
    status = priv->reg->MAC_PMT_CONTROL_STATUS.R;

    /* Enable the Tx again. */
     priv->reg->MAC_CONFIGURATION.B.TE = 1;
     priv->reg->MAC_PMT_CONTROL_STATUS.R = 0;
     ret = core_irq_pmt;
  }
…
```

Note: *In this configuration the registers of the Ethernet controller will not be cleared.*

10 Conclusions

Microcontrollers can be woken up in several different ways depending on the application configuration and the expected power profiles. After a wake-up event different activites can be also performed according to the application needs.

The goal of this document is to provide a simple guide on how to wake up a microcontroller using Ethernet, and does not provide any power consumption data since the platform is not optimized for this purpose.

The user can also exploit different strategies starting from this application note, for instance using the Ethernet RXDATA for STOP and HALT modes. Wake-up from a remote packet event can be also implemented by following the guide provided in the reference manual (see Section Appendix B Reference documents).

Although this application has been implemented for the SPC58NG platforms, it can be ported on almost all SPC58 microcontrollers.

Appendix A Acronyms and abbreviations

Table 2. Acronyms

Abbreviation	Complete name
MAC	Medium Access Control
WOL	Wake-up On LAN
PMT	Power Management
CLI	Command line interface
MII	Media Independent Interface
DMA	Direct memory address
MTL	MAC transaction layer
ISR	Interrupt service routine
MCU	Micro-Controller Unit

Appendix B Reference documents

- RM0407 Reference manual
- OPEN Sleep/Wake-up Specification
- TN1271 Technical note SPC58xB/C/G Low Power Modes
- TN1305 Technical note Network Management Interfaces

Revision history

Table 3. **Document revision history**

Date	Version	Changes
04-Dec-2020	1	Initial release.

Contents

List of tables

List of figures