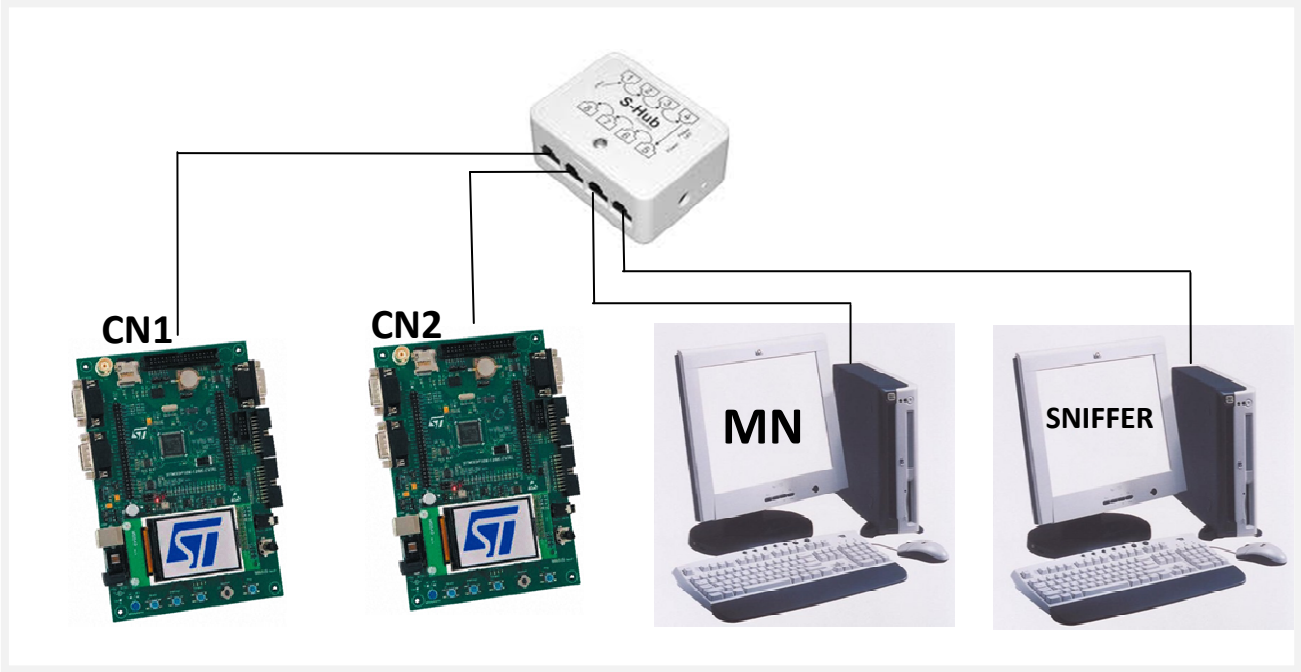
TUTORIAL FOR openPOWERLINK NETWORK DEPLOYMENT

This tutorial show you how to easily setup an Ethernet POWERLINK network using a MN running on Windows environment (Winpcap) and two CNs running on STM3210C-VAL/A. A sniffer (Wireshark for Windows) is connected to the POWERLINK network segment. Devices communicate each other by means of an hub (not a switch). You can change the topology adapting the code described below.



MN configuration

We used the MN of the openPOWERLINK 1.6 package (..\Examples\X86\Windows\VC9\), with some little changes for the above network topology. We choose relaxed network parameters that allow you to observe traffic between devices.

Initialization parameters

POWERLINK initialization parameters are into the `demo_main.c file.`

```
EplApiInitParam.m_dwCycleLen = 150000;     // required for error detection
EplApiInitParam.m_dwPresMaxLatency = 50000;  // const; only required for IdentRes
EplApiInitParam.m_dwAsndMaxLatency = 150000;   // const; only required for IdentRes
EplApiInitParam.m_dwLossOfFrameTolerance = 500000;
EplApiInitParam.m_dwAsyncSlotTimeout = 3000000;
EplApiInitParam.m_dwWaitSocPreq = 150000;
```

The same parameters have to be updated on the CN code as we will show you later.

Reset parameters of CNs

To allow the MN to comunicate with CNs it must send them some commands for resetting their state machines. It is necessary to update the demo_main file changing `EplApiWriteLocalObject` instructions that write the 0x1F81 index.

case kEplNmtGsResetCommunication:

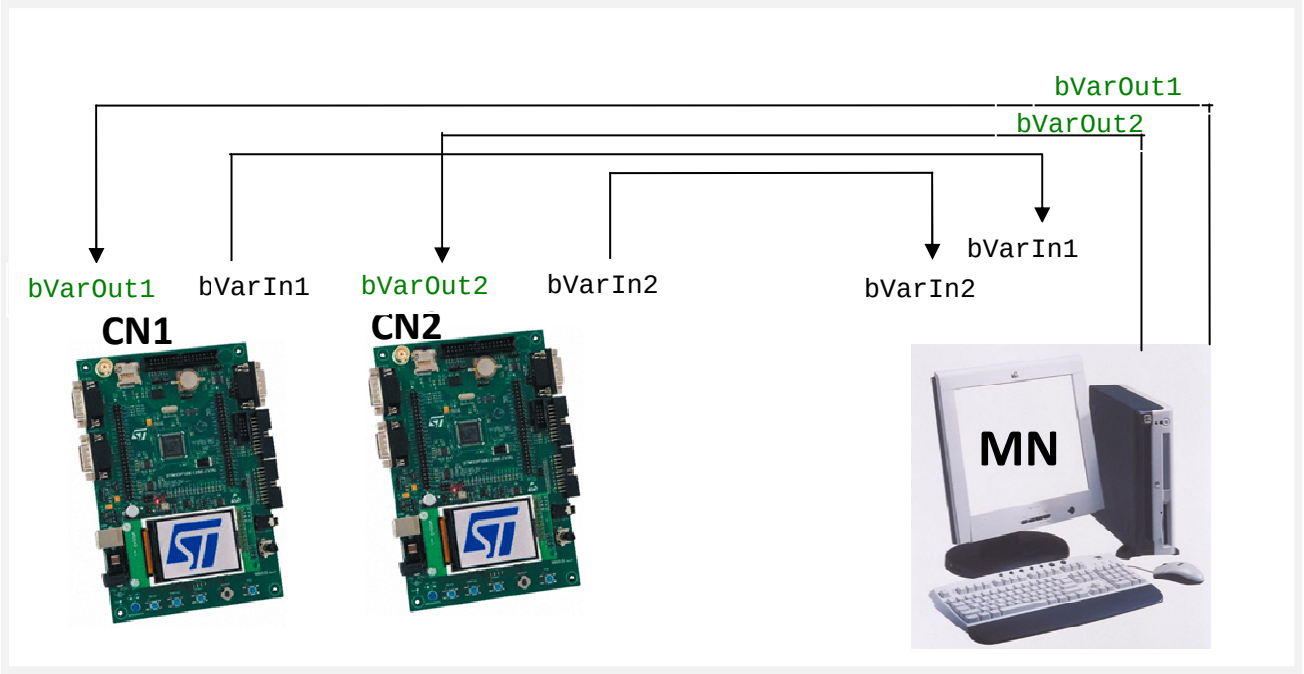
```
{
    DWORD   dwNodeAssignment;


    // configure OD for MN in state ResetComm after reseting the OD
    // TODO: setup your own network configuration here
    dwNodeAssignment = (EPL_NODEASSIGN_NODE_IS_CN | EPL_NODEASSIGN_NODE_EXISTS);    //
0x00000003L
    EplRet = EplApiWriteLocalObject(0x1F81, 0x01, &dwNodeAssignment, sizeof
(dwNodeAssignment));
    EplRet = EplApiWriteLocalObject(0x1F81, 0x02, &dwNodeAssignment, sizeof
(dwNodeAssignment));
    dwNodeAssignment = (EPL_NODEASSIGN_MN_PRES | EPL_NODEASSIGN_NODE_EXISTS);       //
0x00010001L
    EplRet = EplApiWriteLocalObject(0x1F81, 0xF0, &dwNodeAssignment, sizeof
(dwNodeAssignment));
                    // continue
}
```

Linking process variables

The network is configured in a way the MN executes the control loop using input variables from CNs (we simulate a sensor behaviour by which pushbuttons on the eval-board). The MN computes outputs and send them to CNs that adjust the actuators behaviour (we simulate the ouput by writing on the eval-board lcd).

By linking a variable to an object of the OBD we can read and write objects without invoking APIs. This operation is strictly connected to the specific industrial application. We used a simple configuration as illustrated in the figure above:



The linking procedure is carried out into MN and CNs demo_main files. In order to modify variables it is necessary editing OBD as we will show you later.

```
// link process variables used by CN to object dictionary
ObdSize = sizeof(bVarOut1);
uiVarEntries = 1;
EplRet = EplApiLinkObject(0x6200, &bVarOut1, &uiVarEntries, &ObdSize, 0x01);
if (EplRet != kEplSuccessful)
{
    goto Exit;
}

// link process variables used by CN to object dictionary
ObdSize = sizeof(bVarOut2);
uiVarEntries = 1;
EplRet = EplApiLinkObject(0x6203, &bVarOut2, &uiVarEntries, &ObdSize, 0x01);
if (EplRet != kEplSuccessful)
{
    goto Exit;
}

  // link process variables used by CN to object dictionary
ObdSize = sizeof(bVarIn1);
uiVarEntries = 1;
EplRet = EplApiLinkObject(0x6000, &bVarIn1, &uiVarEntries, &ObdSize, 0x01);
if (EplRet != kEplSuccessful)
```

```
    {
        goto Exit;
    }

      // link process variables used by CN to object dictionary
    ObdSize = sizeof(bVarIn2);
    uiVarEntries = 1;
    EplRet = EplApiLinkObject(0x6003, &bVarIn2, &uiVarEntries, &ObdSize, 0x01);
    if (EplRet != kEplSuccessful)
    {
        goto Exit;
    }
```

Object Dictionary configuration

You have to edit the `objdict.h` file for mapping the variables above.

We have choosed those settings:

```
bVarIn1  -> index:0x6000 subindex: 1

bVarIn2  -> index:0x6003 subindex: 1

bVarOut1 -> index:0x6200 subindex: 1

bVarOut2 -> index:0x6203 subindex: 1
```

To link bVarIn1 to the 0x6000 index you must declare this index into the OBD by editing objdict.h with

```
        EPL_OBD_BEGIN_INDEX_RAM(0x6000, 0x02, NULL)
            EPL_OBD_SUBINDEX_RAM_VAR(0x6000, 0x00, kEplObdTypUInt8,
kEplObdAccConst, tEplObdUnsigned8, number_of_entries, 0x1)
            EPL_OBD_SUBINDEX_RAM_USERDEF(0x6000, 0x01, kEplObdTypUInt8,
kEplObdAccVPRW, tEplObdUnsigned8, Sendb1, 0x0)
        EPL_OBD_END_INDEX(0x6000)
```

To link bVarIn2 to the 0x6003 index

```
        EPL_OBD_BEGIN_INDEX_RAM(0x6003, 0x02, NULL)
            EPL_OBD_SUBINDEX_RAM_VAR(0x6003, 0x00, kEplObdTypUInt8,
kEplObdAccConst, tEplObdUnsigned8, number_of_entries, 0x1)
            EPL_OBD_SUBINDEX_RAM_USERDEF(0x6003, 0x01, kEplObdTypUInt8,
kEplObdAccVPRW, tEplObdUnsigned8, Sendb1, 0x0)
        EPL_OBD_END_INDEX(0x6003)
```

To link bVarOut1 to the 0x6200 index

```
        EPL_OBD_BEGIN_INDEX_RAM(0x6200, 0x02, NULL)
            EPL_OBD_SUBINDEX_RAM_VAR(0x6200, 0x00, kEplObdTypUInt8,
kEplObdAccConst, tEplObdUnsigned8, number_of_entries, 0x1)
            EPL_OBD_SUBINDEX_RAM_USERDEF(0x6200, 0x01, kEplObdTypUInt8,
kEplObdAccVPRW, tEplObdUnsigned8, Sendb1, 0x0)
        EPL_OBD_END_INDEX(0x6200)
```

To link bVarOut2 to the 0x6203 index

```
EPL_OBD_BEGIN_INDEX_RAM(0x6203, 0x02, NULL)
            EPL_OBD_SUBINDEX_RAM_VAR(0x6203, 0x00, kEplObdTypUInt8,
kEplObdAccConst, tEplObdUnsigned8, number_of_entries, 0x1)
            EPL_OBD_SUBINDEX_RAM_USERDEF(0x6203, 0x01, kEplObdTypUInt8,
kEplObdAccVPRW, tEplObdUnsigned8, Sendb1, 0x0)
        EPL_OBD_END_INDEX(0x6203)
```

More than defining objects, you have to specify how they must be incapsulated into TPDOs when transmitted, or how they are transferred from an RPDO to the OBD. Every PDO travels throw a virtual channel described with some entries of the OBD (communication parameters, mapping parameters). The transmitting part of the channel is described by a TPDO, the receiving one is described by an RPDO.

This is the TX part of a PDO channel. A PDO with payload 6200/01(bVarOut1) has to be transferred from the MN to the CN1

```
       // Object 1800h: PDO_TxCommParam_00h_REC
        EPL_OBD_BEGIN_INDEX_RAM(0x1800, 0x03, EplPdouCbObdAccess)
            EPL_OBD_SUBINDEX_RAM_VAR(0x1800, 0x00, kEplObdTypUInt8,
kEplObdAccConst, tEplObdUnsigned8, NumberOfEntries, 0x02)
            EPL_OBD_SUBINDEX_RAM_VAR(0x1800, 0x01, kEplObdTypUInt8,
kEplObdAccRW, tEplObdUnsigned8, NodeID_U8, 0x01)
            EPL_OBD_SUBINDEX_RAM_VAR(0x1800, 0x02, kEplObdTypUInt8,
kEplObdAccRW, tEplObdUnsigned8, MappingVersion_U8, 0x00)
        EPL_OBD_END_INDEX(0x1800)

        // Object 1A00h: PDO_TxMappParam_00h_AU64
        EPL_OBD_BEGIN_INDEX_RAM(0x1A00, 0x03, EplPdouCbObdAccess)
            EPL_OBD_SUBINDEX_RAM_VAR(0x1A00, 0x00, kEplObdTypUInt8,
kEplObdAccRW, tEplObdUnsigned8, NumberOfEntries, 0x01)
            EPL_OBD_SUBINDEX_RAM_VAR(0x1A00, 0x01, kEplObdTypUInt64,
kEplObdAccRW, tEplObdUnsigned64, ObjectMapping, 0x0008000000016200LL)
            EPL_OBD_SUBINDEX_RAM_VAR(0x1A00, 0x02, kEplObdTypUInt64,
kEplObdAccRW, tEplObdUnsigned64, ObjectMapping, 0x00)
        EPL_OBD_END_INDEX(0x1A00)
```

As showed with TPDOs, also RPDOs have to be described . This RPDO is received wrapped by a Pres frame sent from CN1(**NodeID_U8, 0x01**). The payload is interpreted using the **0x0008000000016000LL** code that says: “Read the payload from bit 0 (low order bits 8-11), copy it to the 6000/01 object” (this object is mapped by bVarIn1 variable).

```
        // Object 1400h: PDO_RxCommParam_00h_REC
        EPL_OBD_BEGIN_INDEX_RAM(0x1400, 0x03, EplPdouCbObdAccess)
            EPL_OBD_SUBINDEX_RAM_VAR(0x1400, 0x00, kEplObdTypUInt8,
kEplObdAccConst, tEplObdUnsigned8, NumberOfEntries, 0x02)
            EPL_OBD_SUBINDEX_RAM_VAR(0x1400, 0x01, kEplObdTypUInt8,
kEplObdAccRW, tEplObdUnsigned8, NodeID_U8, 0x01)
            EPL_OBD_SUBINDEX_RAM_VAR(0x1400, 0x02, kEplObdTypUInt8,
kEplObdAccRW, tEplObdUnsigned8, MappingVersion_U8, 0x00)
        EPL_OBD_END_INDEX(0x1400)

        // Object 1600h: PDO_RxMappParam_00h_AU64
        EPL_OBD_BEGIN_INDEX_RAM(0x1600, 0x03, EplPdouCbObdAccess)
            EPL_OBD_SUBINDEX_RAM_VAR(0x1600, 0x00, kEplObdTypUInt8,
kEplObdAccRW, tEplObdUnsigned8, NumberOfEntries, 0x01)
```

```
            EPL_OBD_SUBINDEX_RAM_VAR(0x1600, 0x01, kEplObdTypUInt64,
kEplObdAccRW, tEplObdUnsigned64, ObjectMapping, 0x0008000000016000LL)
            EPL_OBD_SUBINDEX_RAM_VAR(0x1600, 0x02, kEplObdTypUInt64,
kEplObdAccRW, tEplObdUnsigned64, ObjectMapping, 0x00)
        EPL_OBD_END_INDEX(0x1600)
```

CN configuration

The demo uses 2 CNs. If you want to increment or decrement this number, you have to edit EplCfg.h changing the n parameter.

```
#ifdef CNn
      #define NODEID 0xn
      #define IP_ADDR 0xc0a8640n
      static  BYTE abMacAddr[] = {0x00, 0x03, 0xc0, 0xa8, 0x64, 0x0n};
      #define DW_DEVICE_TYPE                    -1
      #define DW_VENDOR_ID                      -1
      #define DW_PRODUCT_CODE                   -1
      #define DW_REVISION_NUMBER                -1
      #define DW_SERIAL_NUMBER                  -1
#endif
```

With n ∈[1,239]

It is important to define the macro CNn where n is the node ID of the CN for which you would to run the firmware.

```
#define CNn
```

Those parameters are set in the same way of the MN (init params of demo_main). They are the same for all CNs of the network segment.

```
#define DW_FEATURE_FLAGS                  -1
#define DW_CYCLE_LEN                      150000//5000
#define UI_ISOCHR_TX_MAX_PAYLOAD    100
#define UI_ISOCHR_RX_MAX_PAYLOAD    100
#define DW_PRES_MAX_LATENCY         50000
#define UI_PREQ_ACT_PAYLOAD_LIMIT   36
#define UI_PRES_ACT_PAYLOAD_LIMIT   36
#define DW_ASND_MAX_LATENCY               150000
#define UI_MULTIPL_CYCLE_CNT        0
#define UI_ASYNC_MTU                      1500
#define UI_PRESCALER                      2
#define DW_LOSS_OF_FRAME_TOLERANCE  500000      //900000000
#define DW_ASYNC_SLOT_TIMEOUT       3000000     //10000000
#define DW_WAIT_SOC_PREQ             150000//       0
```

The next step is to edit the VariableMapping() function. Process variables are linked to OBD entries by invoking the `EplApiLinkObject` API. Since the same firmware runs on CNs, it is necessary to bound CN specific code by `ifdef/endif constructs.`

```
#ifdef CNn
EplRet = EplApiLinkObject(0x6200, &bVarOut1_l, &uiVarEntries, &ObdSize, 0x01);
#endif
```

Finally you have to define how to assign values to linked process variables bounding the code as we showed above.

```
tEplKernel PUBLIC AppCbSync(void)
```

This callback is called by openPOWERLINK during the isochronous phase, at right instants that don't interfere with the PDO exchange.

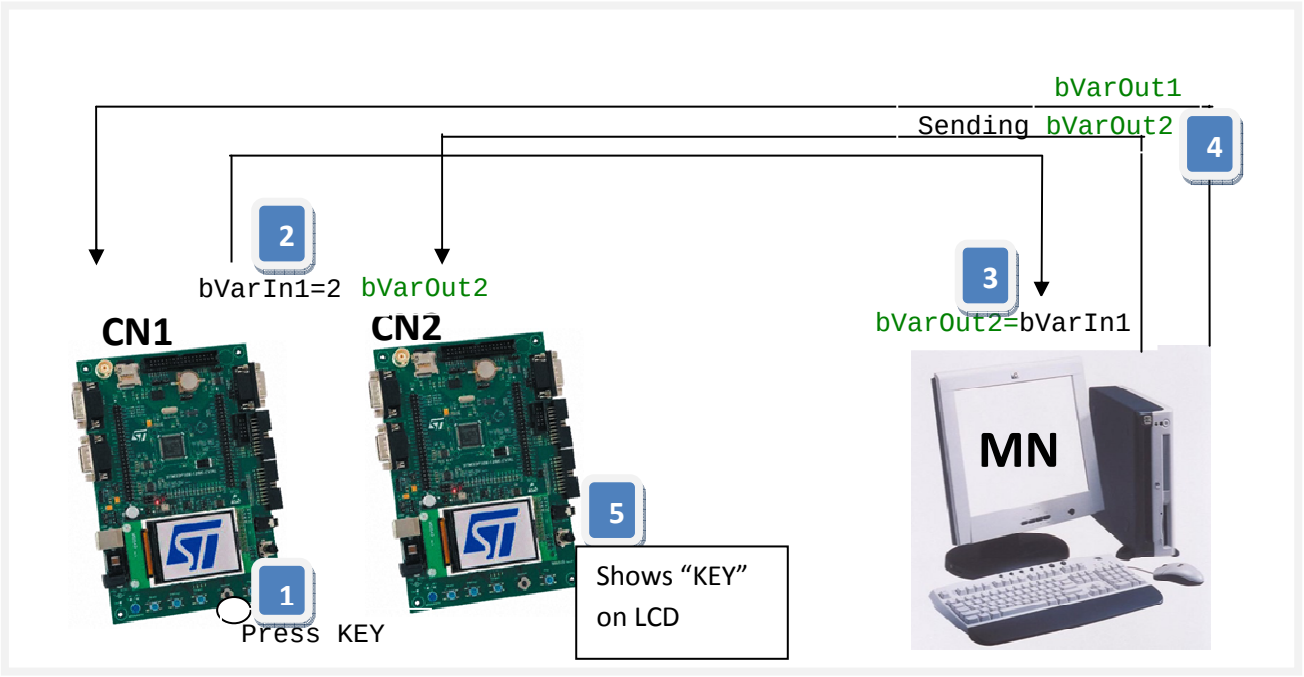
Demo application

The demo application concerns of a diversification among CN1 and CN2 roles. If you push a button on CN1 (key or wakeup), “key” or “wakeup” strings are showed by the CN2 LCD.

If you push the key button the value 2 is written on bVarIn1

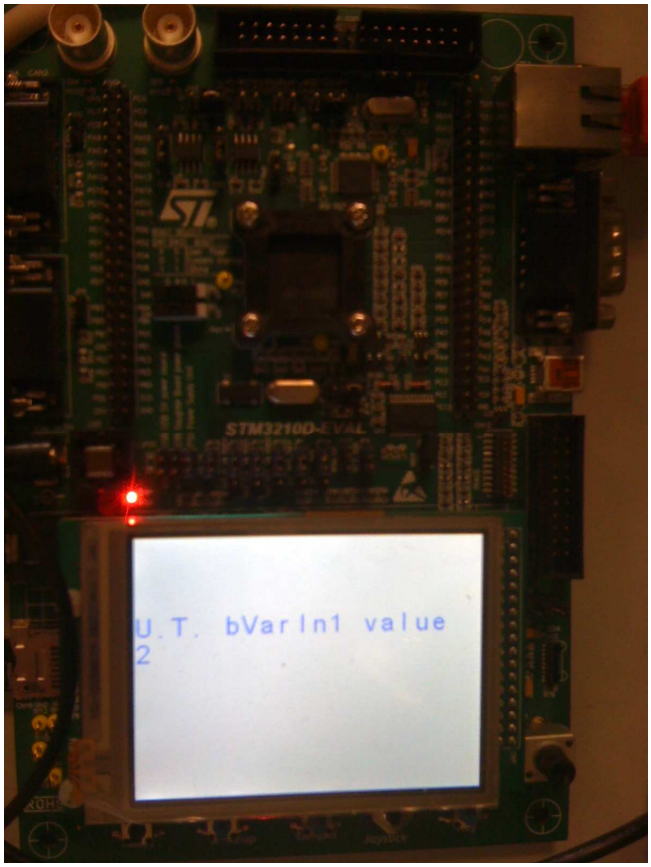
If you push the Wakeup button the value 2 is written on bVarIn1

If you don’t press any button the value 0 is written on bVarIn1



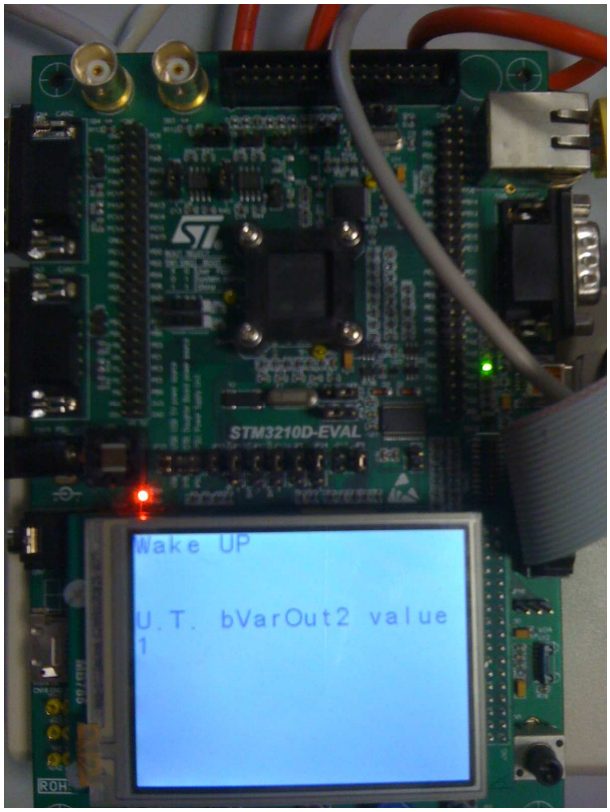
This action demonstrates you that CN1 writes the input variable `bVarIn1` and the MN, that runs the control loop, generates an output for the CN2 writing bVarOut2 (for semplicity only esegue `bVarOut2=bVarIn1` is carried out).

This demo also shows you multitasking features of our porting using the freeRTOS kernel. On CN1 a user task communicates with the task running openPOWERLINK, and the former invokes EplApiReadLocalObject() for reading the `bVarIn1` input variable value. Since the user task invokes the reading API with a relatively wide period, it is necessary manteining pressed the push button in order to detect a value on the LCD (`1:Wakeup, 2:Key`) because to "not pressed buttons(value 0)" is not associated any string on the LCD. The read value by the user task (U.T) is showed with "`U.T. bVarIn1 value`" on `the LCD.`



Similarly as you saw on CN1, CN2 has got a user task which request to openPOWERLINK reading the output variable `bVarOut2`.

The value read by the user task is showed with the `“U.T. bVarOut2 value”` label. If you want a realistic representation of the read value you must mantein pressed the push buttons on CN1.



Massimo Trubia

Alessio Tognazzolo