Verwendete Software:

- rcX V2.0.4.5
- CD-NXLOM RCX-DNS 04'2008 V2.0
- CD-NXLOM RCX-DPS 05'2008 V2.0

Sachverhalt:
Mit Hilfe der rcX-Konfiguration für die PIO-Instanz Definition wurde ein weiter PIO Output Instanz für den Protokoll-Stack DeviceNet Slave erstellt. Das Ziel ist es die PIO0..7 für die verwendeten Protokoll-Stack als Output zu belassen und die PIO 8..30 für jeweilige Inputs oder Outputs zu verwenden.

```
/*
************************************************************
*   Definition of the PIO Instances
************************************************************
*/

STATIC CONST FAR RX_PIO_SET_T atrXPio[] =
{
  {{"SYSPIO",RX_PERIPHERAL_TYPE_PIO,0},
   {RX_PIO_VALUE_TYPE_ACTIVE_HIGH, NETX_PIO_OUT_EN, 0x000000FF},/* Optional Register to make PIO Pin to output at startup */
   {RX_PIO_VALUE_TYPE_NONE,NULL, 0x00000000},                  /* Optional Register to make PIO Pin to output at startup */
   {RX_PIO_VALUE_TYPE_ACTIVE_LOW, NETX_PIO_OUT},               /* PIO Register to set PIOs                               */
   {RX_PIO_VALUE_TYPE_ACTIVE_LOW, NETX_PIO_OUT},               /* PIO Register to clear PIOs                             */
   {RX_PIO_VALUE_TYPE_ABSOLUTE, NETX_PIO_IN},                  /* Register to get current input value of the PIOs        */
  },
  {{"SYSPIO_01",RX_PERIPHERAL_TYPE_PIO, 0},
   {RX_PIO_VALUE_TYPE_ACTIVE_HIGH, NETX_PIO_OUT_EN, 0x00FF0000},/* Optional Register to make PIO Pin to output at startup */
   {RX_PIO_VALUE_TYPE_NONE,NULL, 0x00000000},                  /* Optional Register to make PIO Pin to output at startup */
   {RX_PIO_VALUE_TYPE_ACTIVE_HIGH, NETX_PIO_OUT},              /* PIO Register to set PIOs                               */
   {RX_PIO_VALUE_TYPE_ACTIVE_HIGH, NETX_PIO_OUT},              /* PIO Register to clear PIOs                             */
   {RX_PIO_VALUE_TYPE_ABSOLUTE, NETX_PIO_IN},                  /* Register to get current input value of the PIOs        */
  }
};
```

Weiterhin wurden in dem Packet API Demo der Protokolstacks die PIO 16 und 23 innerhalb der Remote Resourcen der Task angelegt.

```
/* remote resources */
struct DEVICENET_DEMO_RSC_REM_Ttag
{
  /* dummy, for compilations */
  TLR_UINT            uDummy;

  TLR_HANDLE hPio16;              /* Handle for PIO16 */
  TLR_HANDLE hPio23;              /* Handle for PIO23 */
};
```

Diese Handle wurden im weiteren Verlauf mittels der rcX-Treiber Funktionalität „Programmable Input/Output Services“ initialisiert und identifiziert.

```
if((eRslt = Drv_PioIdentifyPio("SYSPIO_01", 0, &ptRsc->tRem.hPio16)) != TLR_S_OK)
{
  goto leave;
}
if((eRslt = Drv_PioInitializePio(ptRsc->tRem.hPio16)) != TLR_S_OK)
{
  goto leave;
}

if((eRslt = Drv_PioIdentifyPio("SYSPIO_01", 0, &ptRsc->tRem.hPio23)) != TLR_S_OK)
{
  goto leave;
}
if((eRslt = Drv_PioInitializePio(ptRsc->tRem.hPio23)) != TLR_S_OK)
{
  goto leave;
}
```

Innerhalb der Funktion „`TaskProcess_DeviceNETDemo`“ werden diese PIOs mit Hilfe der Funktionen „`Drv_PioSetOutputs`“ gesetzt und „`Drv_PioClearOutputs`“ rückgesetzt.

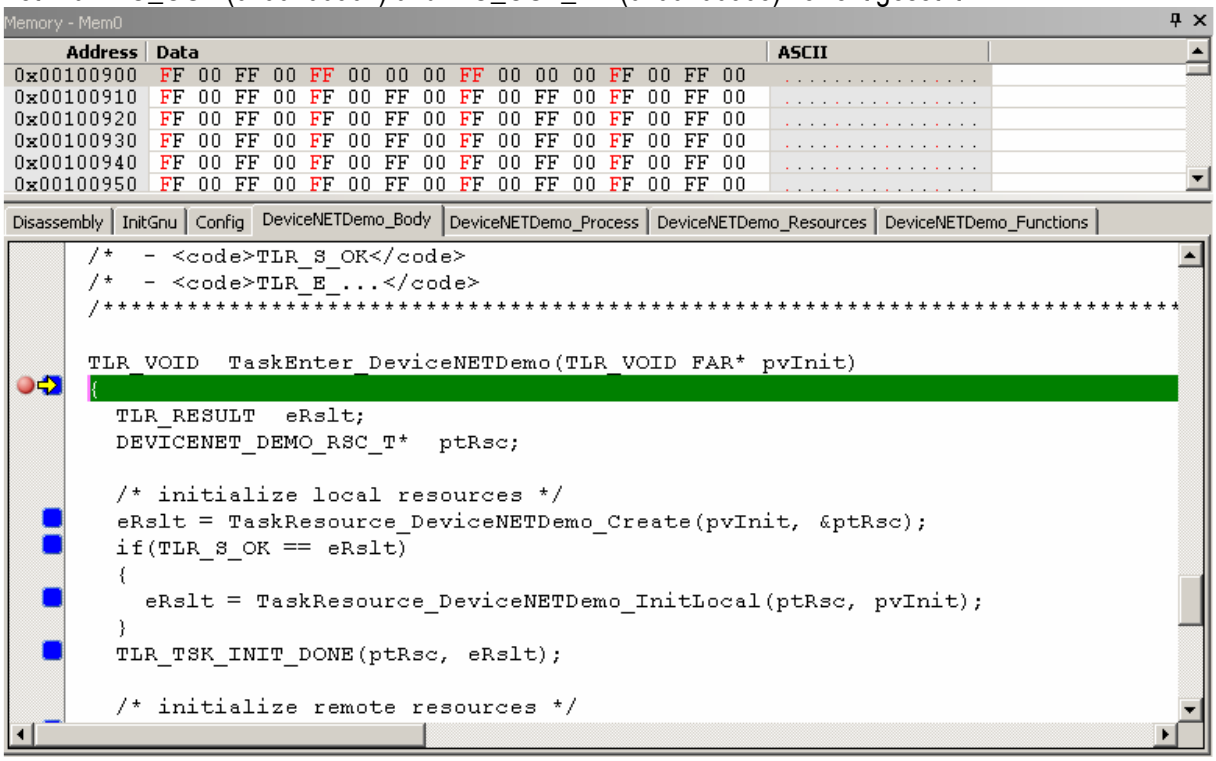
```
/* PIO Handling */
rX_SysSleepTask(2000);
eRslt = Drv_PioSetOutputs(ptRsc->tRem.hPio16,0x00010000);     /* Set the PIO16 */
eRslt = Drv_PioSetOutputs(ptRsc->tRem.hPio23,0x00800000);     /* Set the PIO23 */

rX_SysSleepTask(2000);
eRslt = Drv_PioClearOutputs(ptRsc->tRem.hPio16,0x00010000);   /* Reset the PIO16 */
eRslt = Drv_PioClearOutputs(ptRsc->tRem.hPio23,0x00800000);   /* Reset the PIO23 */
```

Gordon Göhrmann

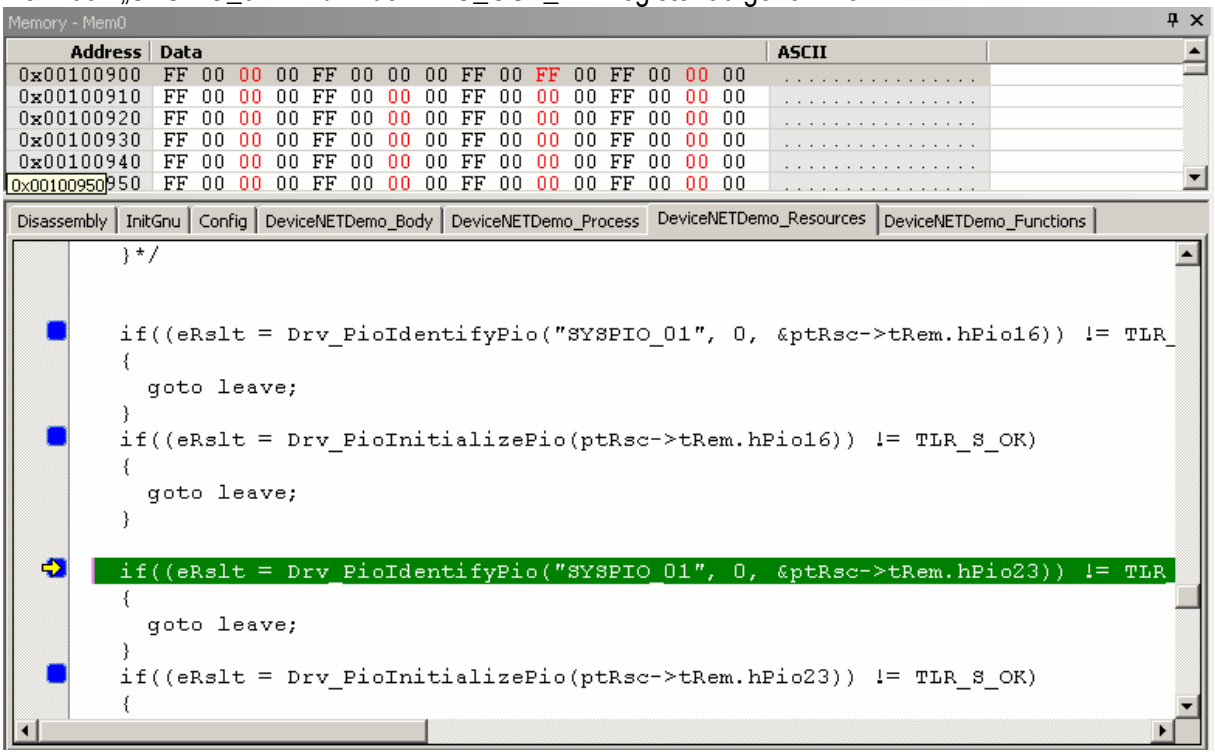
Der Protokoll-Stack DeviceNet Slave verwendet die rcX-Treiber Funktionalität „LED Services - Header File: AP_Led.h“ für die MNS LED. Die Initialisierung und Identifikation der LED´s erfolgt innerhalb der Lib „libdnsfal.a“. Ebenfalls erfolgt das Setzen der LED je Status des Stack aus der Library heraus.

```
/*****************************************************************************
 function:    DnsFalHw_Init()
 description:

 global:      none
 input:       TLR_VOID
 output:      none
 return:      TLR_RESULT
*****************************************************************************/
TLR_RESULT DnsFalHw_Init( DNS_FAL_RSC_T FAR*  ptRsc )
{
  TLR_RESULT eResult = TLR_S_OK;

  if( (TLR_S_OK == Drv_LedIdentifyLed("MNSGRN",0,&ptRsc->tHw.hMnsLedGrn)) &&
      (TLR_S_OK == Drv_LedIdentifyLed("MNSRED",0,&ptRsc->tHw.hMnsLedRed)) )
  {
     if( (TLR_S_OK == Drv_LedInitializeLed(ptRsc->tHw.hMnsLedGrn)) &&
         (TLR_S_OK == Drv_LedInitializeLed(ptRsc->tHw.hMnsLedRed)) )
    {
      ptRsc->tHw.fMNSLed = TLR_TRUE;
    }
  }
  else
  {
    /* Check if Two LEDs are configured */
    if( (TLR_S_OK == Drv_LedIdentifyLed("COMYEL",0,&ptRsc->tLoc.hComLedYel)) &&
        (TLR_S_OK == Drv_LedIdentifyLed("COMRED",0,&ptRsc->tLoc.hComLedRed)) )

    {
          /* Initialize LEDs    */
      if( (TLR_S_OK == Drv_LedInitializeLed(ptRsc->tLoc.hComLedYel)) &&
          (TLR_S_OK == Drv_LedInitializeLed(ptRsc->tLoc.hComLedRed)) )
      {
        //ptRsc->tHw.fMNSLed = TLR_TRUE;
      }
      else
      {
        //ptRsc->tHw.fMNSLed = TLR_TRUE;
      }
    }
  }

  /* Clear LEDs */
  DnsFalHw_ClrErrLed( ptRsc );
  DnsFalHw_ClrStaLed( ptRsc );

  return (eResult);
}
```

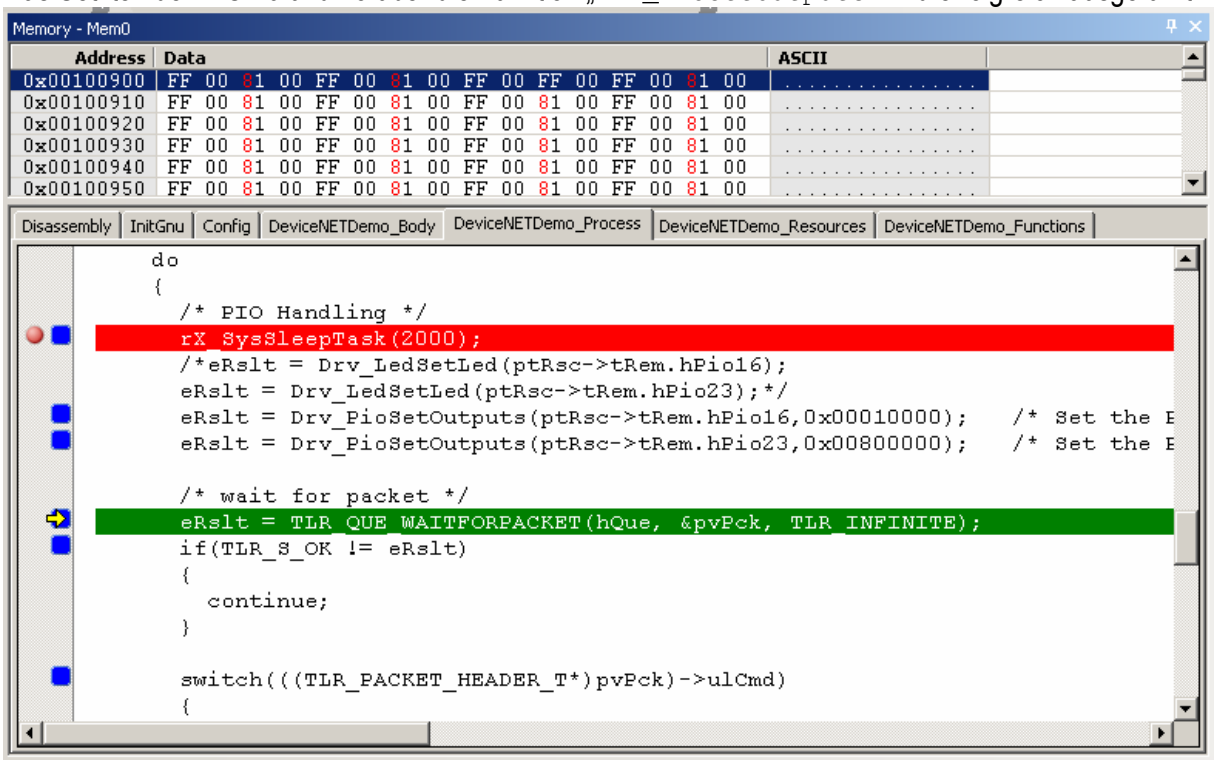
Programmablauf:
Der DeviceNET Stack initialisiert und identifiziert die LED´s für die PIO Instanz „SYSPIO“ vor dem Erzeugen der Packet API Demo Task „`TaskEnter_DeviceNETDemo`“. Hierbei sind die Register des netX für PIO_OUT (0x00100904) und PIO_OUT_EN (0x00100908) korrekt gesetzt.



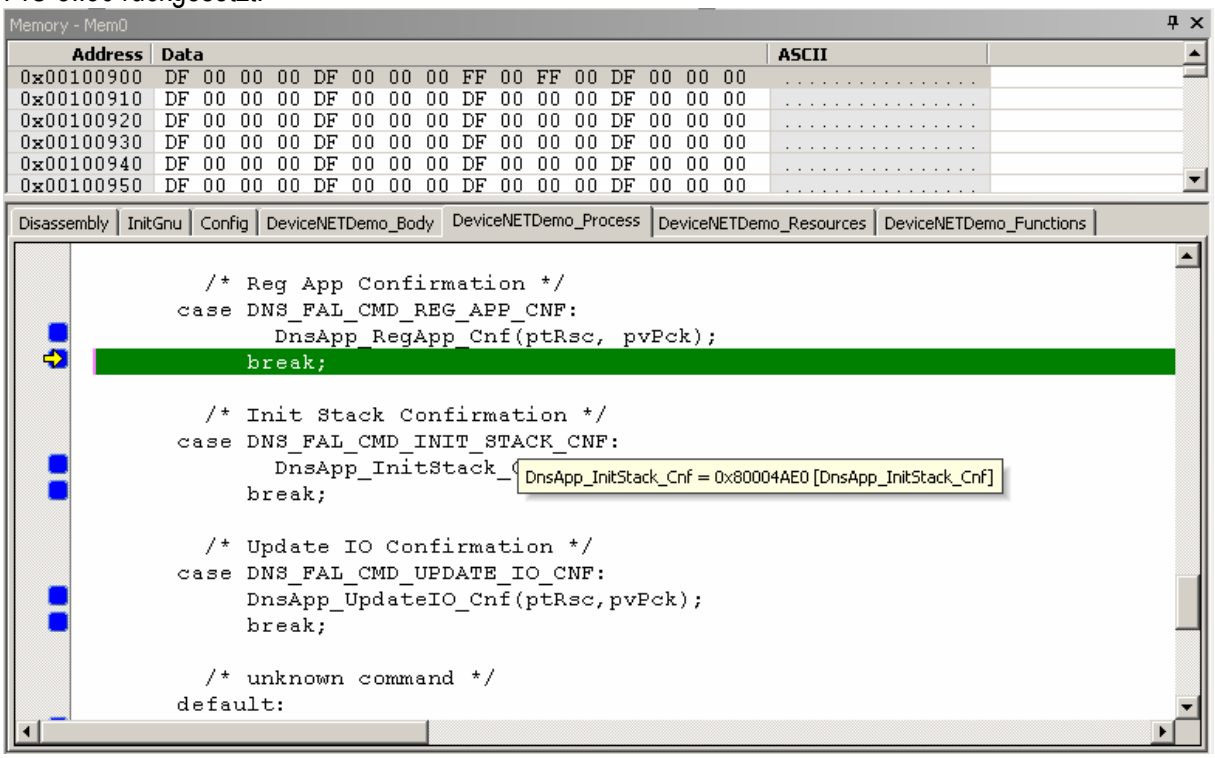
Im weiteren Verlauf werden die PIO 16 und 23 der PIO Instanz Definition „SYSPIO_01“ identifiziert und initialisiert. Hier sind wiederum das Verhalten und die Ausgabe völlig korrekt. Die weitere PIO-Instanz Definition „SYSPIO_01“ wird in dem PIO_OUT_EN Register aufgenommen.



Das Setzten der PIO 16 und 23 über die Funktion „`Drv_PioSetOutputs`“ wird erfolgreich ausgeführt.



Im nächsten Schritt wird vom Stack mit Hilfe der Warmstartparameter initialisiert. Dabei wird die MNS LED je nach Status Rot oder Gelb gesetzt. Durch setzen der PIO 4 und 5 werden die weiteren Output PIO 8..30 rückgesetzt.



Gordon Göhrmann

Resultat:
Es ist nicht möglich die PIO 0...30 zu instanziieren, um diese für diverse Aufgaben zu nutzten, da durch den jeweiligen Protokoll-Stack zu unterschiedlichen Laufzeiten das komplette PIO_OUT Register beeinflusst wird. Weiterhin ist es auch nicht möglich auf die Treiber Funktionalität „LED Services - Header File: AP_Led.h“ um zu steigen, da hier ebenfalls eine gegenseitige Beeinflussung der Register stattfindet. Beim Protokoll-Stack PROFIBUS werden die Status-LED über den Middleware rcX Treiber „`Mid_LedCreateLed`“ gesetzt. Auch mit diesem Protokoll-Stack wurden beide Varianten der Treiber erfolglos geprüft.

Workaround:
Es wird mit ausschließlich einer PIO Instanz Definition gearbeitet.

```
/*
************************************************************
*   Configuration of the PIO Instances
************************************************************
*/
STATIC CONST FAR RX_PIO_SET_T atrXPio[] =
{
  {{"SYSPIO", RX_PERIPHERAL_TYPE_PIO, 0},
  {RX_PIO_VALUE_TYPE_ACTIVE_HIGH, NETX_PIO_OUT_EN, 0x00FF00FF}, /* Optional Register to make PIO Pin to output at startup */
  {RX_PIO_VALUE_TYPE_NONE, NULL, 0x00000000},                   /* Optional Register to make PIO Pin to output at startup */
  {RX_PIO_VALUE_TYPE_ACTIVE_LOW, NETX_PIO_OUT},                 /* PIO Register to set PIOs   */
  {RX_PIO_VALUE_TYPE_ACTIVE_LOW, NETX_PIO_OUT},                 /* PIO Register to clear PIOs */
  {RX_PIO_VALUE_TYPE_ABSOLUTE, NETX_PIO_IN},                    /* Register to get current input value of the PIOs */
  }
};
```

Hierbei wird eine gegenseitige Beeinflussung verhindert. Der Nachteil besteht darin, dass alle benötigten weiteren PIO 8..30 den definierten Startup-Status einnehmen. Da unsere Status- und Sys-LED low-aktiv verwendet werden, werden die PIO 0...7 auf high gezogen, um die LED´s im Startup zu disablen. Somit sind die weiteren PIO aber ebenfalls nur im Startup Status High verwendbar. Der Anwender ist somit gezwungen die verwendeten Atoren dahingehend auszulegen. (Bspw. Relais)

Ursache:
Die PIOs werden innerhalb der der Funktion „`HalPioInitialize`“ des rcX initialisiert. Hier besteht eine IF-Anweisung, wodurch alle PIO für den Startup Status in die der Konfiguration angegebenen Ausgangs-Status gesetzt werden. Prinzipiell ist der Fehler auf UND- /ODER-Verknüpfte Anwendungen zurück zu führen.

```
/*  ============================================================================================

  Function: HalPioInitialize
  ============================================================================================ */
RX_RESULT FAR HalPioInitialize(RX_PIO_T FAR* ptPio)
{
  unsigned long ulIrqSave;
  lock_irq_save(ulIrqSave);
  /* Switch off the outputs first of all */
  if(ptPio->tCfg.tClr.eTyp == RX_PIO_VALUE_TYPE_ACTIVE_HIGH) {
    POKE_AND(ptPio->tCfg.tClr.uReg,~(ptPio->tCfg.tMod.uVlu));
  } else {
    POKE_OR(ptPio->tCfg.tClr.uReg,ptPio->tCfg.tMod.uVlu);
  }

  /* Make the Mode selection */
  switch(ptPio->tCfg.tMod.eTyp) {
    case RX_PIO_VALUE_TYPE_ACTIVE_HIGH:
      POKE_AND(ptPio->tCfg.tMod.uReg,~ptPio->tCfg.tMod.uVlu);
      POKE_OR(ptPio->tCfg.tMod.uReg,ptPio->tCfg.tMod.uVlu);
    break;
    case RX_PIO_VALUE_TYPE_ACTIVE_LOW:
      POKE_AND(ptPio->tCfg.tMod.uReg,ptPio->tCfg.tMod.uVlu);
      POKE_OR(ptPio->tCfg.tMod.uReg,~ptPio->tCfg.tMod.uVlu);
    break;
    case RX_PIO_VALUE_TYPE_ABSOLUTE:
      POKE(ptPio->tCfg.tMod.uReg,ptPio->tCfg.tMod.uVlu);
    break;
    default:
    break;
  }
  lock_irq_restore(ulIrqSave);
  return(RX_OK);
}
```