AN11583

Guide about how to port the Passive Target example from the NFC Reader Library to another MCU

Rev. 1.0 — 11 August 2014 **Application note**
302310 **COMPANY PUBLIC**

03Document information

Info	Content
Keywords	NFC Reader Library, LPCXpresso, LPC1769, LPC1227, SNEP, NDEF, PNEV512B, NFC, P2P,
Abstract	This document describes a way how to port a software project based on the NFC Reader Library 3.x from a microcontroller based on an ARM Cortex-M3 to a simpler microcontroller based on an ARM Cortex-M0. It also description which software modules are necessary to replace and which application variables and buffers must be reduced.

Revision history

Rev	Date	Description
1.0	20140811	Initial revision

Contact information

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

The software project used in this documentation to show the way of porting to another MCU is the “Passive Target” example for the PN512 NFC reader IC. The way pointed out here can be applied to any project based on the NFC Reader Library 3.x. No matter which reader IC is used.

The Passive Target example is the NFC P2P application based on SNEP. The application runs as either SNEP Server or SNEP Client and it demonstrates NDEF message transmission using SNEP.

The example has been created to run on NXP microcontroller LPC1769. The microcontroller is ARM Cortex-M3 core based, providing enough performance and FLASH and RAM required for hosting the Passive Target application.

The main goal of the document is to show and describe steps of application porting to a simpler and cheaper microcontroller. LPC1227 microcontroller produced by NXP based on ARM Cortex-M0 core has been chosen. The microcontroller type LPC1227 delivers sufficient computational performance and FLASH amount.

The secondary goal of the document is to show main principles of the Passive Target application, how it works and furthermore to clarify how overwriting some dedicated configuration parts influences behavior of the application.

In the porting section there are described modifications of selected parts of the application and MCU drivers which results to LPC1227 is able to host (and run) the Passive Target application. At the end there are listed some results of practical tests referring about RAM memory usage by the Passive Target application concluding into size limit for an NDEF message to be transferred.

2. Application description

2.1 Overview

2.1.1 SNEP Overview

Simple NDEF Exchange Protocol is a request/response protocol for exchanging NDEF messages over LLCP. SNEP uses underlying Connection Oriented LLCP link to exchange data packets with peer device. SNEP uses Connection Oriented LLCP link as its lower transport. Requests are always sent by the client and responses are sent by server.[8], [9] [11]. The software application is possible to find at the web site [13].

SNEP API is divided into three categories of functions:

Session Establishment:

This session starts with creating SNEP Server and Client then sends a connect request from Client to Server. The last action is acceptance of the incoming connection request by Server.

Data Exchange:

This is the main part of the application. There are PUT and GET **requests** sent from Client to Server and **responses** from Server to Client.

Session Release:

Disconnect is also necessary part of correct protocol performance. Client disconnects from Server and Server confirms the disconnection and provides any services for the Client no more until next connection request.

2.1.2 Example Setting Overview

The passive target application example demonstrates transmission of an NDEF message via SNEP packets. The data transfer capabilities are described in chapters 2.1.3 and 2.1.4.

2.1.2.1 Set Server or Client

Following defines are used to configure application as SNEP Client or SNEP Server.

```
1    #define SNEP_SERVER            /* Enable for Demo application as SNEP Server */
2    #define SNEP_CLIENT            /* Enable for Demo application as SNEP Client */
```

2.1.2.2 SNEP Server Service Name

Following defines are used to set the SNEP Server service name. If the SNEP Client application and the SNEP server application run at the same time while expected to be connected and communicate each other, they both must be set to the same service name.

```
3    #define DEFAULT_SERVER         /* Enable the SNEP Default Server  */
4    #define NON_DEFAULT_SERVER     /* Enable the SNEP Non-Default Server  */
```

Default Server

The name of the default Server service is: **urn:nfc:sn:snep** and it is defined in [8]

For this service name only PUT *request* is supported. GET request is not default supported and usual *response* is E0h – Not Implemented.

Non Default Server

The name of the non-default Server service is: **urn:nfc:xsn:nfc-forum.org:snep-validation.** This name represents the **Validation Server**. The Validation Server shall accept Put and Get *requests*. The behavior is defined in [12]

In our application the SNEP Server behavior is changed as follows. There is Data buffer where is stored received NDEF message. On receive the GET *request* from the Client, the Server sends the message present in the Data buffer encapsulated in *response*. In case the Data buffer is empty Server will send the default message in *response*.

The default message is defined follow:

```
5     uint8_t    Data[] = {'S','N','E','P','D','A','T','A','G','E','T','\0'};
```

2.1.3 SNEP Server capabilities

Passive Target example set as Server supports following NDEF message types:

1.) short TEXT message (up to 255Bytes)

2.) long TEXT message (from 256 to 1022Bytes[note1])

3.) URI message

These messages shall follow the standard NFC Forum well-known type [10]

NOTE1: the maximal length of received data depends on length of the:

```
6     uint8_t   TransferData[1024] = {0};      /**< Data Inbox Buffer */
```

It is declared in the *Example_Pn512_M3M0_P2P_Target.c* source file.

2.1.4 SNEP Client capabilities

Passive Target example set as Client supports following NDEF message types:

1.) RTD type - **short TEXT** message (up to 255Bytes)

2.) RTD type - **long TEXT** message (from 256 to 1023Bytes)

3.) RTD type - **URI** message

These messages shall follow the standard NFC Forum well-known type [see NFCForum-TS-RTD_1.0 (2006-07-24)]

Range of supported message types could be easily extended by adding following NDEF types:

4.) RTD type - long **TEXT** message (more than 1023Bytes, theoretical limit is FFFFFFFFh Bytes (32bit number), but in real 50kB file is possible transfer)

5.) MIME message format (multimedia data) **JPG**, **JPEG** and **PNG** picture

6.) MIME message format – **vCard**

7.) RTD type - **Smart Poster format** which combines different RTD types to one NDEF message.

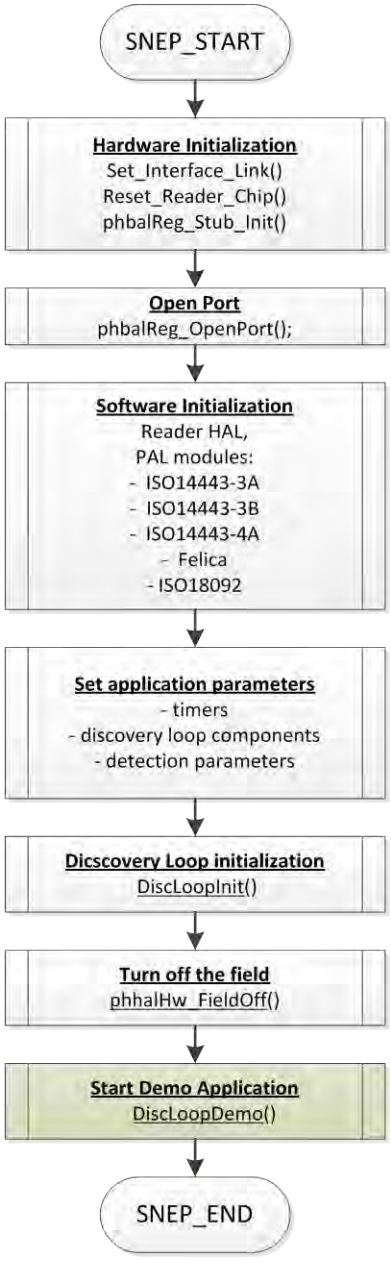
2.2 Passive Target Example Overview

This part consists mainly of the flowcharts which provide overview of the Passive Target Example behavior. The behavior is divided in to three parts.

2.2.1 Main-Loop Overview

Passive Target example – main loop is shown via flowchart in the figure:

Fig 1 - Main-Loop



(1) shadow item is enlarged in the next figures

Fig 1. Main-Loop

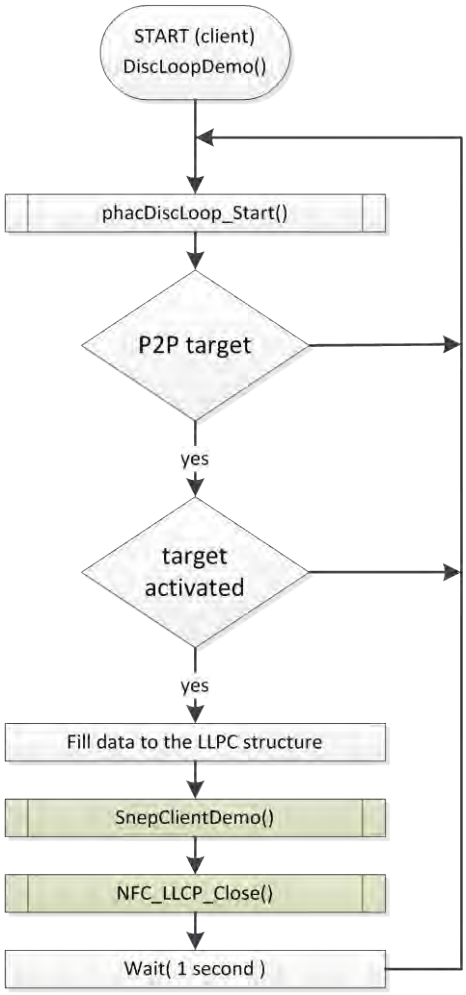
2.2.2 SNEP Client Overview

SNEP Client example application is shown via flowcharts in the figures:

Fig 2 - Client Discovery-Loop

Fig 3 - SNEP Client Demo

Close LLCP for Client



(1) shadow items are enlarged in the next figures

Fig 2. Client Discovery-Loop

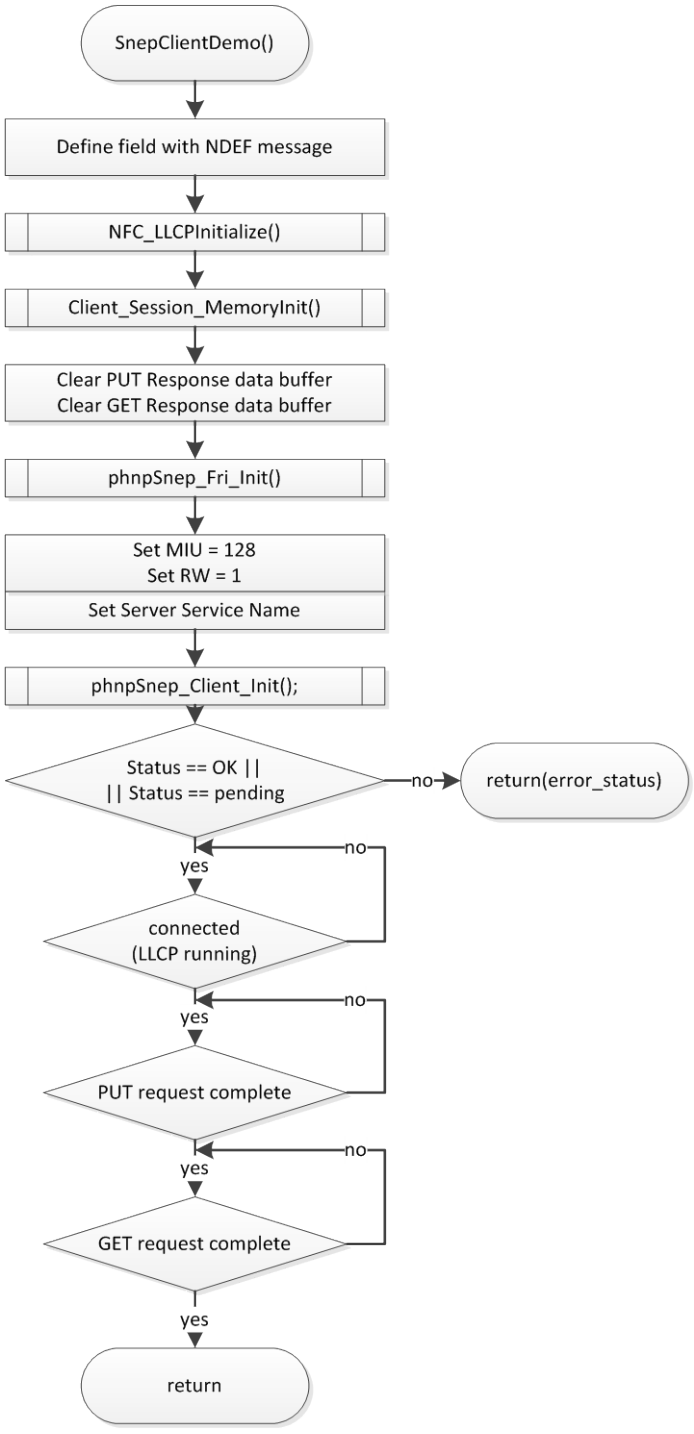


Fig 3. SNEP Client Demo

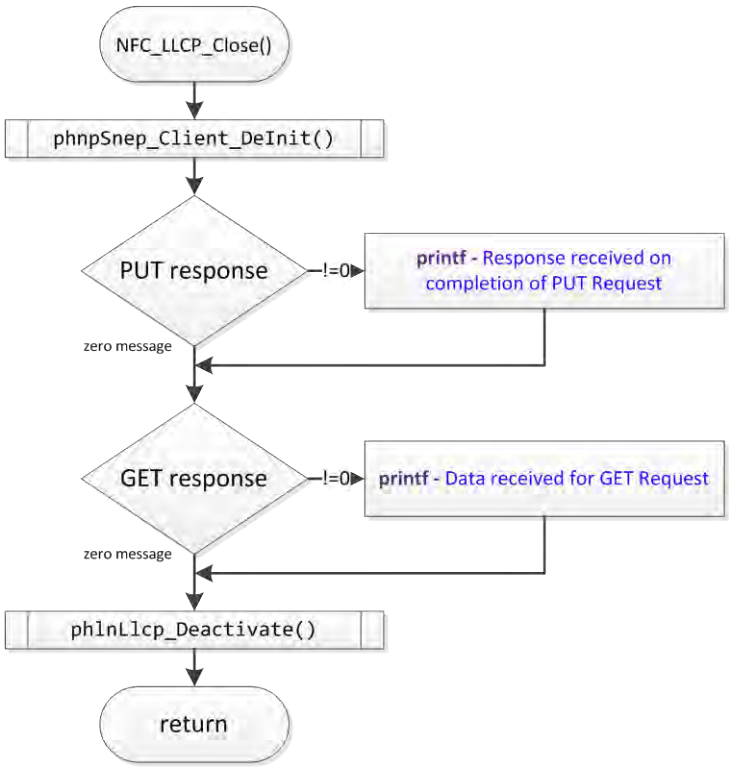


Fig 4. Close LLCP for Client

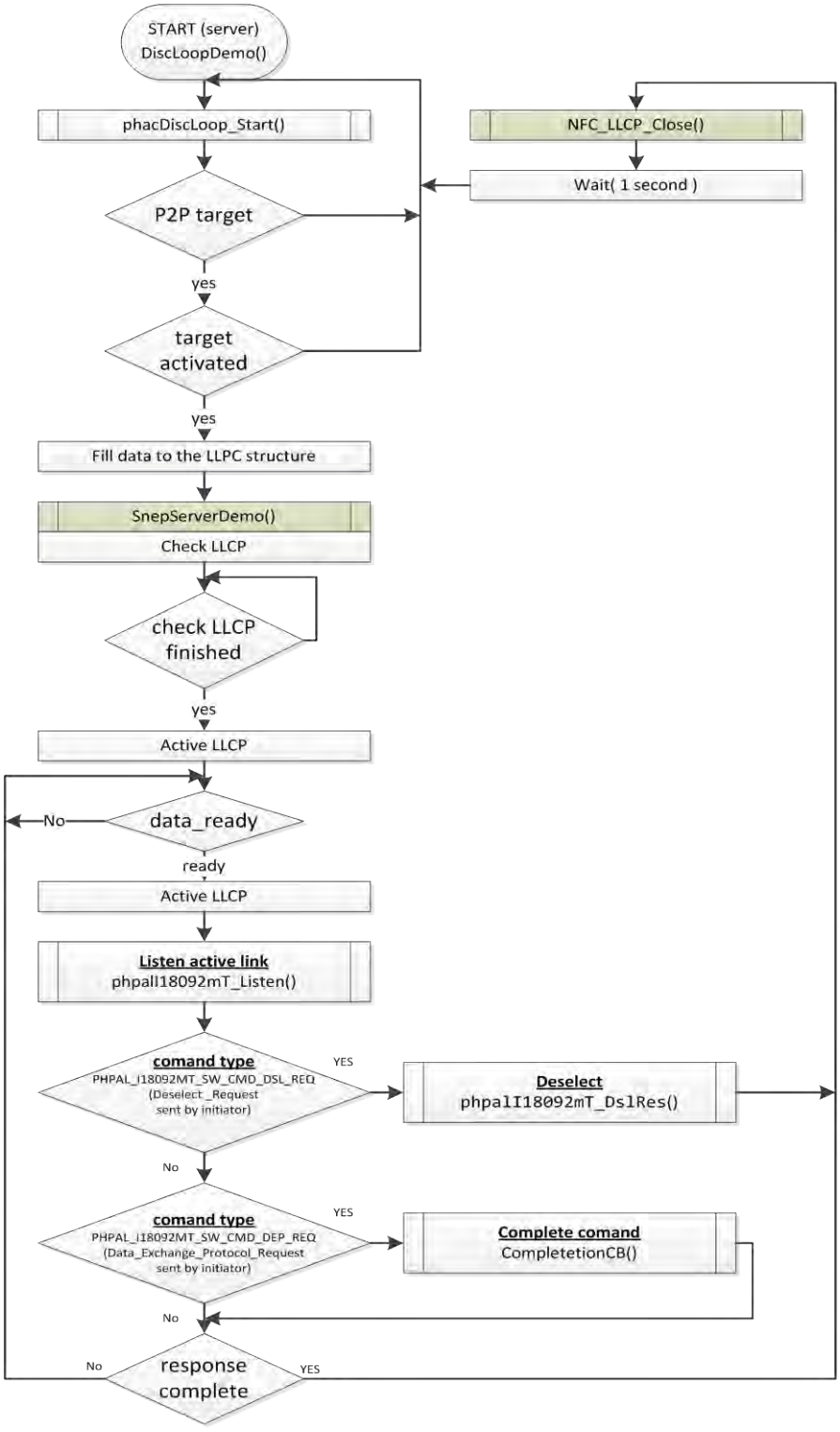
2.2.3 SNEP Server Overview

SNEP Server example application is shown via flowcharts in the figures:

Fig 5 - Server Discovery-Loop

Fig 6 - SNEP Server Demo

Fig 7 - Close LLCP for Server



(2) shadow items are enlarged in the next figures

Fig 5. Server Discovery-Loop

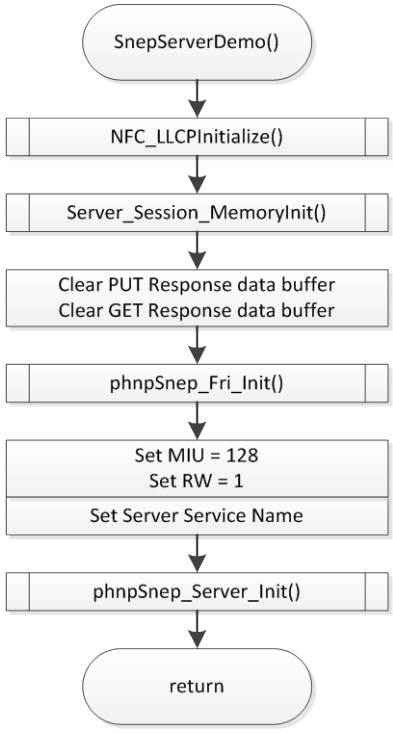


Fig 6. SNEP Server Demo

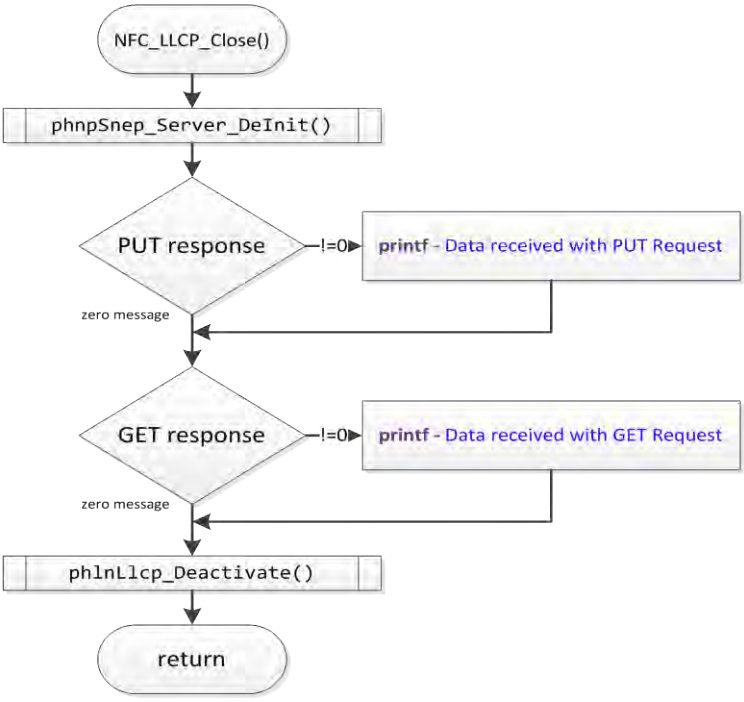


Fig 7. Close LLCP for Server

2.2.4 Software Structure

At the figure Fig 8 there is described the whole project structure. The structure consists of 2 libraries and of the application:

- NXP Reader Library (public release of the NFC library)
- LPC1xxx (library supports the microcontroller)
- P2P-PN512-Target (passive target application consists only of main.c file)

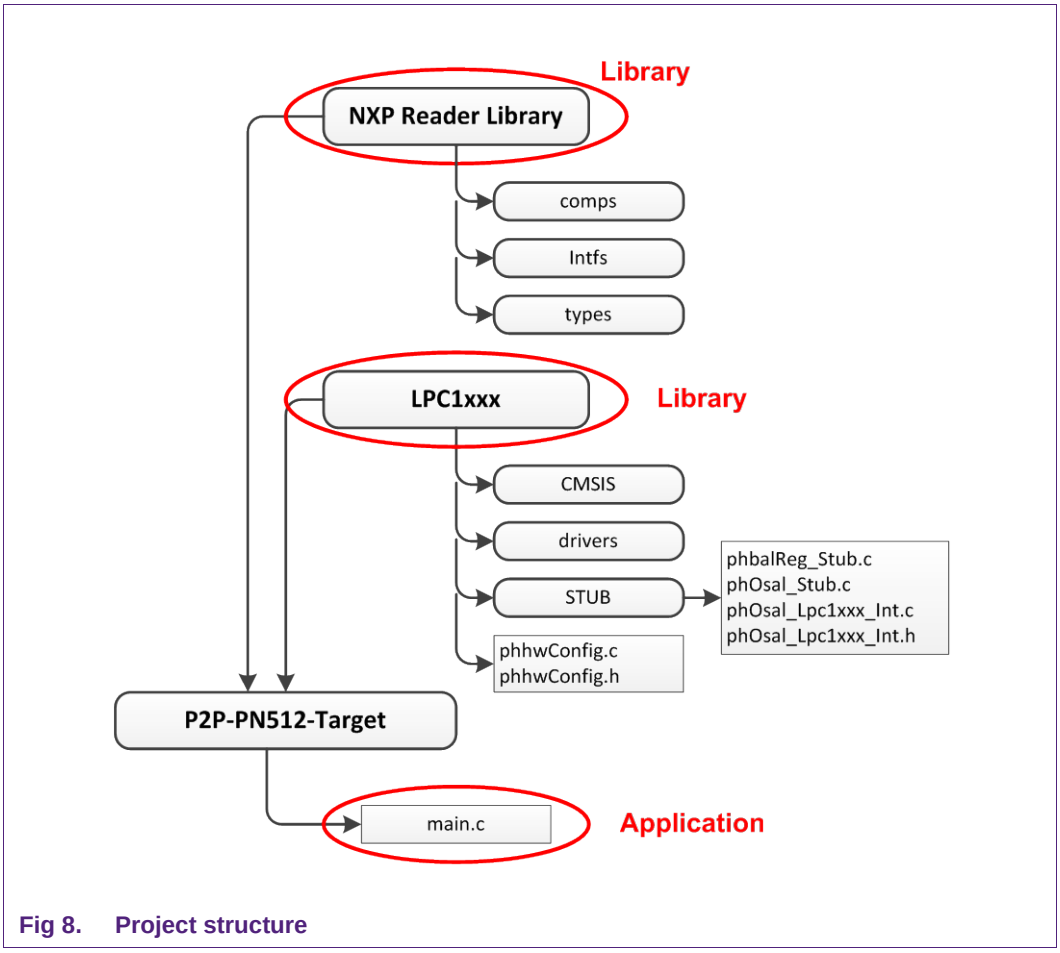


Fig 8. Project structure

Project is built from Nxp Reader Library, one of the LPC1xxx library and from application (*main.c file*). The Nxp Reader Library supports the SNEP Client/Server application and it needs to be always included. LPC1xxx library selection depends on used MCU and is shown in the figure Fig 9 and MCU setting is shown in the figure Fig 10. Source code of the application and its main() function is placed in the *main.c* file.

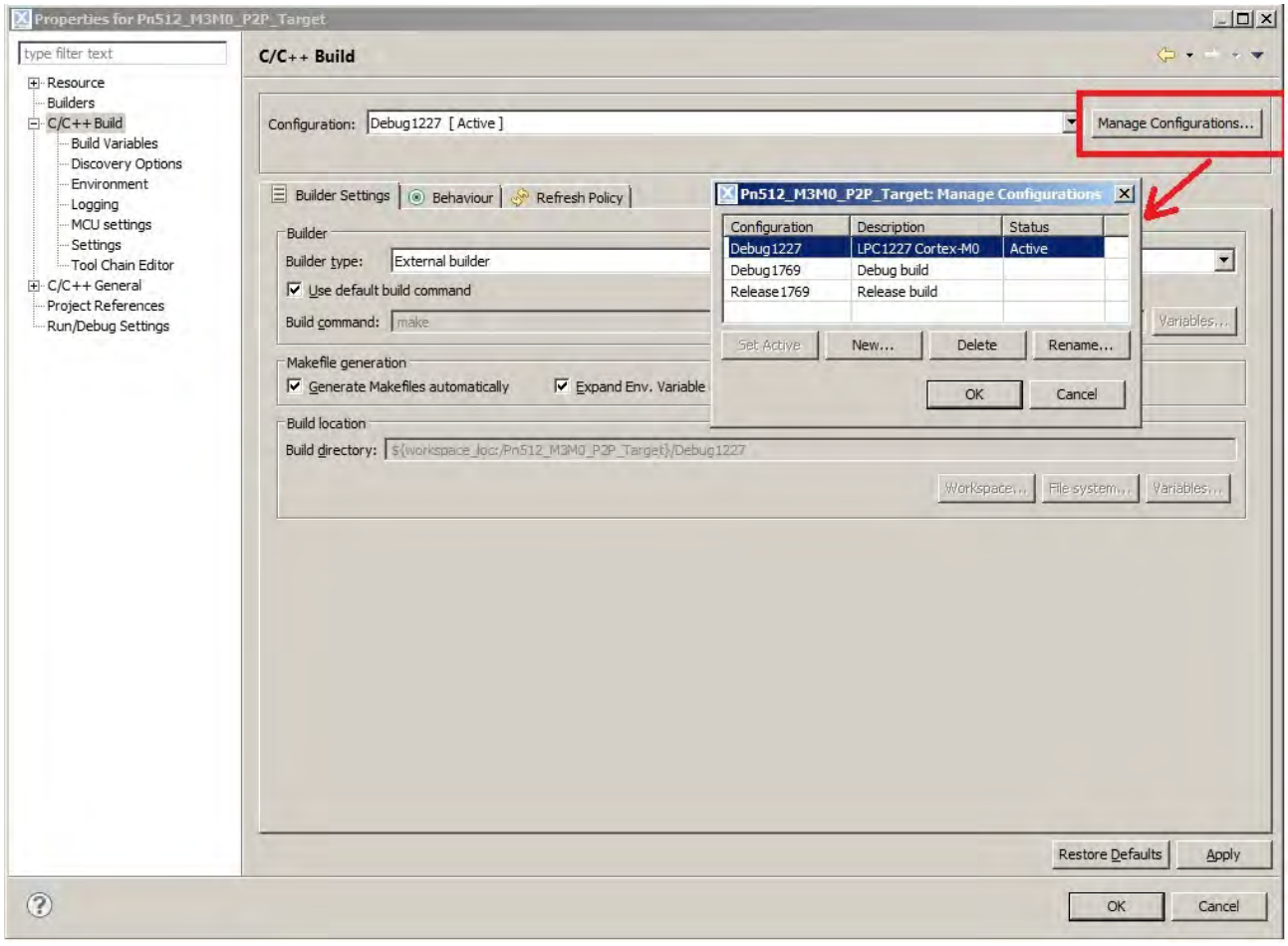


Fig 9. MCU library selection

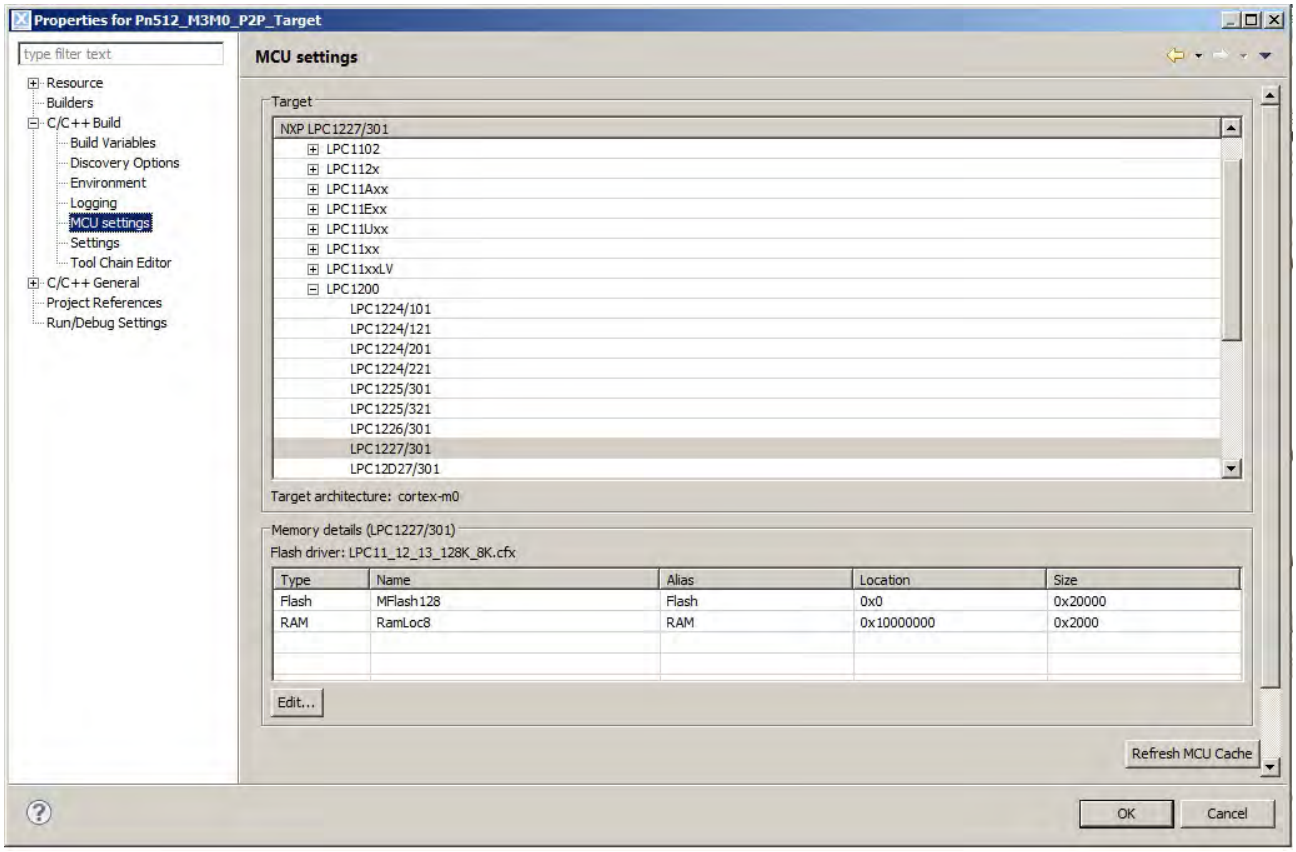


Fig 10. MCU settings

3. Application Porting

3.1 Overview

The Passive Target demo application has been created with an intention to be run on conjugation of LPCXpresso LPC1769 board revision B [5] with Blueboard PNEV512B version v1.5 [3]. The LPC1769 is MCU based on Cortex-M3 core.

The target SNEP application can be ported for simpler Cortex-M0 core. This application note demonstrates porting to LPC1227 MCU, which is placed on LPCXpresso LPC1227 board [6].

The porting requires changes in following modules:

BAL – communication between MCU and reader chip

OSAL – solution of timer and memory management

GPIO – processor I/O pins driving (module dependent on MCU)

The porting requires adding of following modules:

Hardware module – Interrupt handlers and I/O pins configuration

3.1.1 BAL

The Bus Abstraction Layer (BAL) ensures correct communication interface between the master device and the reader chip [1]. In this case the master is microcontroller MCU.

The original BAL module consists of the generic module and specific modules. MCU uses the STUB module. This module is designed to complement its own functionality and it is placed in the NXP Reader Library at following place:

```
.\NXP-Reader-Library\comps\phbalReg\src\Stub\
```

The name of the file is: *phbalReg_Stub.c.*

The main principle how to custom the BAL-STUB module for the typical type of the MCU is to copy this source file to the STUB part of the LPC1xxx library (see the project structure at the picture Fig 8). Then add or enlarge the functionality of each required function inside the module.

The advantage of this change is that it is not necessary to change the names of variables and structures and functions by themselves.

The Passive Target example has implemented only SPI communication interface.

The source file *phbalReg_Stub.c* needs to be enhanced of cases to be able to call functions of the added SPI module.

Example how to port BAL functions

There is the description how to port function - *phbalReg_Stub_Exchange()* from **LPC1768** library to the **LPC1227** library. Changes are marked by underline green lines.

Original function *phbalReg_Stub_Exchange()* in the LPC1769 library

```
7     phStatus_t phbalReg_Stub_Exchange(
8                                       phbalReg_Stub_DataParams_t * pDataParams,
9                                       uint8_t * pTxBuffer,
10                                      uint16_t wTxLength,
```

```
11                                  uint16_t wRxBufSize,
12                                  uint8_t * pRxBuffer,
13                                  uint16_t * pRxLength
14                                  )
15  {
16      uint16_t xferLen;
17      SSP_DATA_SETUP_Type xferConfig;
18
19      xferConfig.length = wTxLength;
20      xferConfig.rx_data = pRxBuffer;
21      xferConfig.tx_data = pTxBuffer;
22
23      if(pDataParams->bHalType == PHBAL_REG_HAL_HW_RC523)
24      {
25          LPC_GPIO0->FIOCLR = 0x00000040;
26          xferLen = SSP_ReadWrite (LPC_SSP1, &xferConfig, SSP_TRANSFER_POLLING);
27          LPC_GPIO0->FIOSET = 0x00000040;
28      }
29      else if(pDataParams->bHalType == PHBAL_REG_HAL_HW_RC663)
30      {
31          LPC_GPIO0->FIOCLR = 0x00000400;
32          xferLen = SSP_ReadWrite (LPC_SSP1, &xferConfig, SSP_TRANSFER_POLLING);
33          LPC_GPIO0->FIOSET = 0x00000400;
34      }
35      else
36      {
37          return PH_ADD_COMPCODE(PH_ERR_INTERNAL_ERROR, PH_COMP_BAL);
38      }
39
40      if (xferLen != wTxLength)
41      {
42          return PH_ADD_COMPCODE(PH_ERR_INTERFACE_ERROR, PH_COMP_BAL);
43      }
44
45      *pRxLength = xferLen;
46
47      return PH_ADD_COMPCODE(PH_ERR_SUCCESS, PH_COMP_BAL);
48  }
```

Function *phbalReg_Stub_Exchange()* after porting to LPC1227 library:

```
49  phStatus_t phbalReg_Stub_Exchange(
50                                  phbalReg_Stub_DataParams_t * pDataParams,
51                                  uint8_t * pTxBuffer,
52                                  uint16_t wTxLength,
53                                  uint16_t wRxBufSize,
54                                  uint8_t * pRxBuffer,
55                                  uint16_t * pRxLength
56                                  )
57  {
```

```
58        uint16_t xferLen;
59        SSP_DATA_SETUP_Type xferConfig;
60
61        xferConfig.length = wTxLength;
62        xferConfig.rx_data = pRxBuffer;
63        xferConfig.tx_data = pTxBuffer;
64
65        if(pDataParams->bHalType == PHBAL_REG_HAL_HW_RC523)
66        {
67          GPIOSetValue(PORT0, PIN SSEL, SSEL ASR); // CS on
68          xferLen = SSP_ReadWrite (LPC SSP, &xferConfig, SSP_TRANSFER_POLLING);
69          GPIOSetValue(PORT0, PIN SSEL, SSEL DEASR);     // CS off
70        }
71        else if(pDataParams->bHalType == PHBAL_REG_HAL_HW_RC663)
72        {
73          GPIOSetValue(PORT0, PIN SSEL, SSEL ASR); // CS on
74          xferLen = SSP_ReadWrite (LPC SSP, &xferConfig, SSP_TRANSFER_POLLING);
75          GPIOSetValue(PORT0, PIN SSEL, SSEL DEASR);     // CS off
76        }
77        else
78        {
79            return PH_ADD_COMPCODE(PH_ERR_INTERNAL_ERROR, PH_COMP_BAL);
80        }
81
82        if (xferLen != wTxLength)
83        {
84            return PH_ADD_COMPCODE(PH_ERR_INTERFACE_ERROR, PH_COMP_BAL);
85        }
86
87        *pRxLength = xferLen;
88
89        return PH_ADD_COMPCODE(PH_ERR_SUCCESS, PH_COMP_BAL);
90    }
```

3.1.2 OSAL

The Operating System Abstraction Layer (OSAL) supports Timers usage and RAM (de)allocation [1].

The original OSAL module consists of the generic module and specific modules. MCU uses the STUB module. This module is designed to complement its own functionality and it is placed in the NXP Reader Library at following place:

```
.\NXP-Reader-Library\comps\phOsal\src\Stub\
```

The name of the file is *phOsal*_Stub.c.

The main principle how to custom the OSAL-STUB module for the typical type of the MCU is to copy this source file to the STUB part of the LPC1xxx library (see the project structure at the picture Fig 8). Then add or enlarge the functionality of each required function inside the module.

The advantage of this change is that it is not necessary to change the names of variables and structures and functions by themselves.

Example how to port OSAL functions

There is the description how to port function *phOsal_Stub_Timer_Create()* from the **LPC1768** library to the **LPC1227** library. Changes are marked by underline green lines.

Original function *phOsal_Stub_Timer_Create()* in the LPC1769 library

```
91   phStatus_t phOsal_Stub_Timer_Create(
92                                       phOsal_Stub_DataParams_t   *pDataParams,
93                                       uint32_t                       *pTimerId
94                                       )
95   {
96       phOsal_Lpc17xx_Int_Timer_Create(pDataParams, pTimerId);
97
98       if (*pTimerId == -1)
99          {
100         /* No timer found, need to return error */
101         return PH_ADD_COMPCODE(PH_OSAL_ERR_NO_FREE_TIMER,  PH_COMP_OSAL);
102         }
103      else
104         {
105         return PH_ADD_COMPCODE(PH_ERR_SUCCESS, PH_COMP_OSAL);
106         }
107  }
```

Function *phOsal_Stub_Timer_Create()* after porting to LPC1227 library:

```
108  phStatus_t phOsal_Stub_Timer_Create(
109                                      phOsal_Stub_DataParams_t *pDataParams,
110                                      uint32_t * pTimerId
111                                      )
112  {
113      phOsal_Lpc12xx_Int_Timer_Create(pDataParams, pTimerId);
114
115      if (*pTimerId == -1)
116         {
117         /* No timer found, need to return error */
118         return PH_ADD_COMPCODE(PH_OSAL_ERR_NO_FREE_TIMER,  PH_COMP_OSAL);
119         }
120      else
121         {
122         return PH_ADD_COMPCODE(PH_ERR_SUCCESS, PH_COMP_OSAL);
123         }
124  }
```

3.1.3 MCU drivers

Drivers are necessary to support and handle all on chip peripherals of the MCU. Drivers are part of the CMSIS [14]. For the Passive Target example only three drivers are used:

SPI

GPIO

Timers

3.1.4 Hardware module

Name of this module is **phhwConfig**. It consists of functions which directly control the hardware MCU functionalities.

There are interrupt handlers for timers and for handling the interrupt request from reader chip when data are received.

There is also reset pin setting and interface selection pins setting (both for reader chip control)

3.1.5 Timers overview

First we have to recognize the “software (system) timer” and the “hardware timer” for timer name.

Hardware timer:

This is physical realization of the timer. Hardware time is directly on the MCU chip and it has own control register, clock source, counter register, match registers and clock prescale register. Hardware timer should be used for system timer realization or it can be used directly by application.

Software timer:

This is software realization of the timer. Software timer is represents by own Timer_ID, and expiration time or delay time (and for operation system also timer type). There is also possibility to execute some system or application function within the timer usage. Software timer is possible to manage by following commands of the OSAL API for our Passive Target example:

Create, Delete, Init, Start, Stop, Reset, Execute Callback, Wait.

There are 4 hardware timers in both MCU.

LPC1769 timers:

4x 32 bits hardware timers

LPC1227 timers:

2x 32bits hardware timers

2x 16bits hardware timers.

3.1.6 Usage of the timers

In the OSAL module there is general approach regarding to the timers: each software timer is represents by one hardware timer only and each hardware timer could provide one software timer at a certain time.

There are 4 software timers each one corresponding to a separate hardware timer. *The order doesn’t matter for timers with Timer_ID = 0 to 2.*

Passive Target example uses 2 software timers directly for P2P and LLCP layer establishment. One is used as LTO timer in case the LLCP is activated, the second is RTOX in case only 18092-DEP layers is activated. The 3rd software timer is free and could be used for user application like software or hardware timer. The 4th software timer is assigned by Timer_ID = 4. This one is separated from timers group. It is specially used to perform time delay represented by the function **phOsal_Timer_Wait**. Time delay is used directly in Discovery Loop library part. Time delay may be used in main loop (user application) part where time delay is required before performing next loop and the Discovery Loop doesn't run.

The Table 1 describes timers usage and naming

Table 1. Timers naming and usage

Timer_ID	Timer name LPC1769	Timer name LPC1227	Usage
0	Timer0	Timer32_0	LLCP, 18092 or may be used by user
1	Timer1	Timer32_1	
2	Timer2	Timer16_0	
3	Timer3	Timer16_1	WAIT

3.1.7 Timers Porting

System_clock is the internal clock frequency set by system registers of the MCU and the peripherals are controlled by this one.

Porting of 32bits timer to 32bit timers is without necessity of any modification of the OSAL functions code.

32bits timer is able to perform time counting in the range from 2 us to 89 s for 48MHz system clock frequency.

16bits timer provides time counting in range from 2 us to 1365 us for 48MHz system clock frequency.

Porting of 32bits timer to 16bits timer is a problem and has to be handled by time value range checking and setting the prescaler of the timer. There is different approach in solution for microsecond timer and millisecond timer.

Microsecond timer: realization is shown at the figure Fig 11. Time value is limited to 1000us.

Millisecond timer: realization is shown at the figure Fig 12. Match register of the timer is fulfilled by default value which represents 1000us. Timer prescaler is configured to a value corresponding to one millisecond as one tick at the timer counts register. By this solution 16bits timer is able to provide time amounts in range from 1ms to 65535 ms.

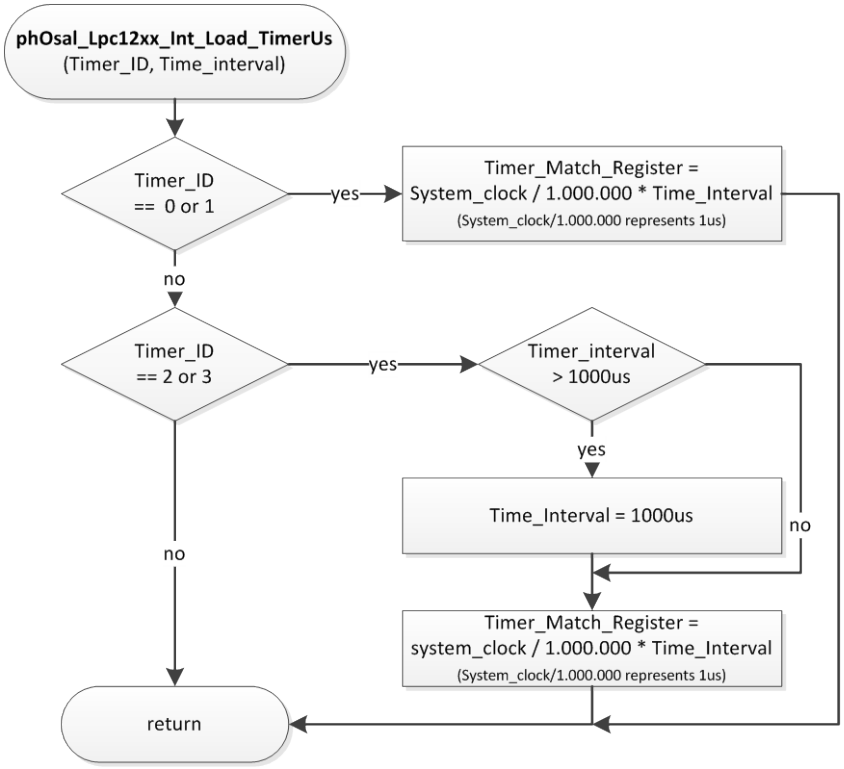


Fig 11. Porting 32 to 16 bits timer – time in microseconds

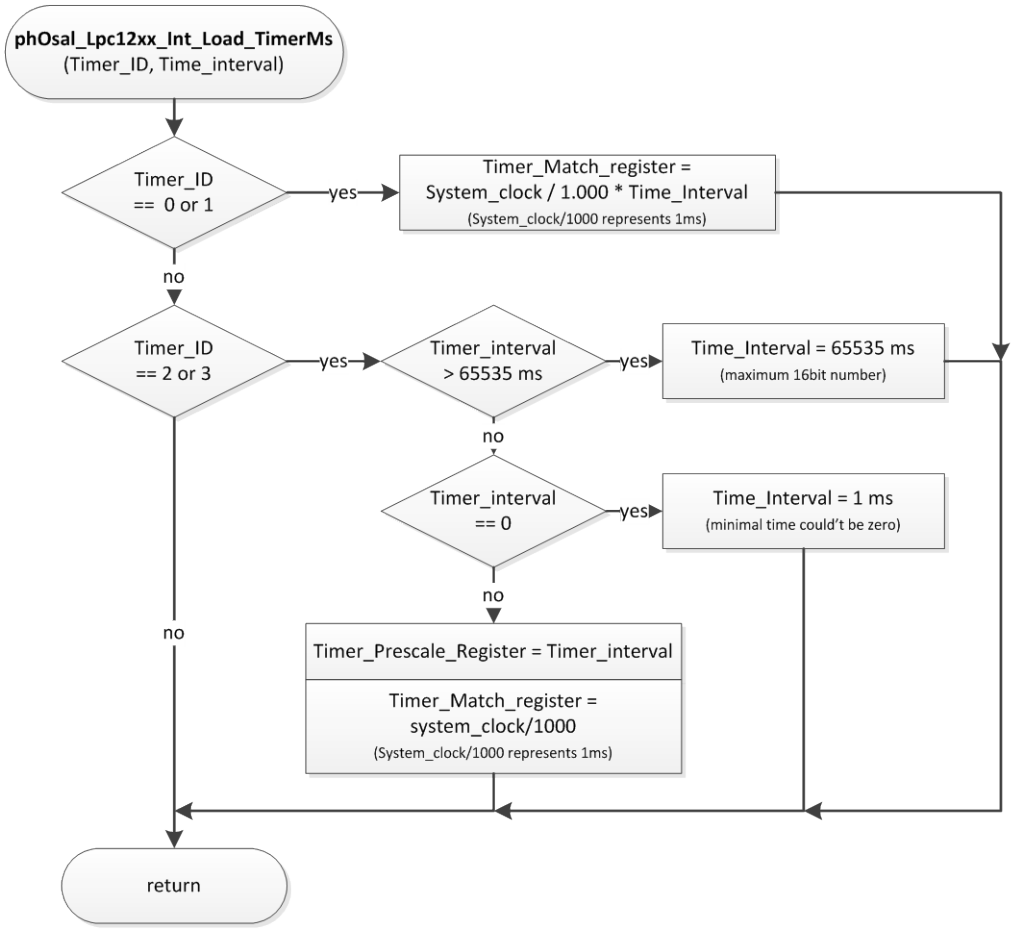


Fig 12. Porting 32 to 16 bits timer – time in miliseconds

3.1.8 Global Variables

LPC1769 and LPC1227 are different types of MCU. However both are ARM Cortex core. The project building is the same or very similar. Main difference is in FLASH and RAM memory amount. Next Table 2 describes these differences.

Table 2. RAM and FLASH amount overview

	FLASH	RAM
LPC1227	128 kB	8 kB
LPC1769	512 kB	64kB

Whole source code uses up to 80kB of FLASH memory after project build (for both MCUs). The FLASH memory is sufficient.

In case the Passive Target example is built for the LPC1769 MCU the RAM usage is 10,5kB and there is not calculated space which is used during operation mode where dynamically allocated variables and buffers also spend some space of RAM. Finally the RAM usage is about 11kB. For that reason we have to focus on reduction of global static variables (buffers), which are allocated in the RAM memory.

3.1.8.1 Reduction of global buffers

There are follow global variables in the Passive Target project used by LPC1769 MCU. These variables can be resized to keep full functionality of the application. All the variables represent data buffers.

Server buffers

Here are data buffers of HAL and LLCP and their size can be reduced in following way:

Before reduction:

```
uint8_t         bHalBufferTx[400];          /**< HAL TX buffer */
uint8_t         bHalBufferRx[400];          /**< HAL RX buffer */
uint8_t         bRxBuffer[256];             /**< LLCP TX buffer */
uint8_t         bTxBuffer[256];             /**< LLCP RX buffer */
```

After reduction:

```
uint8_t         bHalBufferTx[128];          /**< HAL TX buffer */
uint8_t         bHalBufferRx[128];          /**< HAL RX buffer */
uint8_t         bRxBuffer[128];             /**< LLCP TX buffer */
uint8_t         bTxBuffer[256];             /**< LLCP RX buffer */
```

Here are data buffers for responses transmission (answer of server for client request).

Before reduction:

```
uint8_t         PutResponseBuffer[1500];    /**< PUT Response Buffer */
uint8_t         GetResponseBuffer[256];     /**< GET Response Buffer */
```

After reduction:

```
uint8_t         PutResponseBuffer[128];    /**< PUT Response Buffer */
uint8_t         GetResponseBuffer[128];     /**< GET Response Buffer */
```

Here are data buffers storing and internal handling with transmitted NDEF messages.

Before reduction:

```
uint8_t         workingBuffer[256];         /**< Server Working Buffer (MIU * RW) + MIU */
uint8_t         snepWorkingBuffer[128];     /**< Working Data Buffer */
uint8_t         sConnWorkingBuffer[128];    /**< Connection Data Buffer */
uint8_t         pChunkingBuffer[128];       /**< Chunking Data Buffer */
uint8_t         TransferData[2048] = {0};   /**< Data Inbox Buffer */
uint8_t         gSnepPacketBuffer[1100];    /**< SNEP Packet Data Buffer */
```

After reduction:

```
uint8_t         workingBuffer[128];         /**< Server Working Buffer (MIU * RW) + MIU */
uint8_t         snepWorkingBuffer[128];     /**< Working Data Buffer */
uint8_t         sConnWorkingBuffer[128];    /**< Connection Data Buffer */
uint8_t         pChunkingBuffer[128];       /**< Chunking Data Buffer */
uint8_t         TransferData[1024] = {0};   /**< Data Inbox Buffer */
uint8_t         gSnepPacketBuffer[256];    /**< SNEP Packet Data Buffer */
```

Client Selected Variables

Here are data buffers of HAL and LLCP and their size can be reduced as follows:

Before reduction:

```
25  uint8_t         bHalBufferTx[400];      /**< HAL TX buffer */
26  uint8_t         bHalBufferRx[400];      /**< HAL RX buffer */
27  uint8_t         bRxBuffer[256];         /**< LLCP TX buffer */
28  uint8_t         bTxBuffer[256];         /**< LLCP RX buffer */
```

After reduction:

```
29  uint8_t         bHalBufferTx[128];      /**< HAL TX buffer */
30  uint8_t         bHalBufferRx[128];      /**< HAL RX buffer */
31  uint8_t         bRxBuffer[128];         /**< LLCP TX buffer */
32  uint8_t         bTxBuffer[256];         /**< LLCP RX buffer */
```

Here are data buffers for responses transmission (answer of server for client request).

Before reduction:

```
33  uint8_t         PutResponseBuffer[256];    /**< PUT Response Buffer */
34  uint8_t         GetResponseBuffer[1500];   /**< GET Response Buffer */
```

After reduction:

```
35  uint8_t         PutResponseBuffer[128];    /**< PUT Response Buffer */
36  uint8_t         GetResponseBuffer[128];   /**< GET Response Buffer */
```

Here are data buffers storing and internally handling with transmitted NDEF messages.

Before reduction:

```
37  uint8_t         WorkingBuffer[256];        /**< Server Working Buffer (MIU * RW) + MIU */
38  uint8_t         ResponseBuffer[1500];      /**< Response Buffer */
39  uint8_t         pChunkingBuffer[128];      /**< Chunking Buffer of size RemoteMiu or MIU */
40  uint8_t         gSnepPacketBuffer[1100];   /**< SNEP Packet Data Buffer*/
```

After reduction:

```
41  uint8_t         WorkingBuffer[256];        /**< Server Working Buffer (MIU * RW) + MIU */
42  uint8_t         ResponseBuffer[256];      /**< Response Buffer */
43  uint8_t         pChunkingBuffer[128];      /**< Chunking Buffer of size RemoteMiu or MIU */
44  uint8_t         gSnepPacketBuffer[256];   /**< SNEP Packet Data Buffer*/
```

3.1.9 Reduced Variables in Practice

There have been performed lots of tests to setup optimal size of the buffers mentioned in chapter 3.1.8.1. Here is a summary.

Passive Target Server

Complete Passive Target Server example **built** requires **6874 Bytes of RAM**. Only **Server** application **requires 2718 Bytes** from this space. There **remains** about **160 Bytes** of RAM for other usage in the application.

Passive Target Client

Complete Passive Target Client example **built** requires **6082 Bytes of RAM**. Only **Client** application **requires 2482 Bytes** from this space. There **remains** about **600 Bytes** of RAM for other usage in the application.

All these values have been taken from LPCXpresso IDE environment after project build and during full operation mode.

4. Build configuration

4.1 Hardware Development Kits

Passive Target software example is directly designed (including the porting) for the following hardware development tools:

LPCXpresso LPC1769	microcontroller development kit [5]
LPCXpresso LPC1200	microcontroller development kit [6]
Blueboard PNEV512B v1.5	contactless card reader board [3]

4.2 Software Development tools

Passive Target software example has been developed using:

LPCXpresso v6.0.4_159 IDE	[7]

4.3 Debug Build Configuration

Passive Target example is built directly under the debug mode. This mode allows a user to print out debug messages on console, optimize and reduce code size, debug process including a detailed view on all registers and peripherals, view a memory.

Tool Settings of the IDE must be set as follows:

Common Symbols (LPC1227 and LPC1769)

DEBUG	- allows use debug mode and messages
__USE_CMSIS	- using CMSIS drivers support for the core and whole registers set of the MCU.

Basic symbols are commonly added.

Symbols of LPC1227:

__DISABLE_WATCHDOG	- watchdog module must be disabled for debug configuration.

NOTE: LPC1227 is industries oriented MCU and watch dog timer is allowed by default.

Optimization

-O1	This is first level of optimization and the code size reduction is sufficient for correct application operation.

NOTE: Higher optimization (eg. –O2 or -Os) caused linker troubles. It means during operations directly for LPC1227 MCU fatal error has occurred and the software has been interrupted from running. The same fail occurred also on the –Os which is optimization for size.

Debugging

Maximum	the maximal debug support
-g3	Level 3 includes extra information, such as all the macro definitions present in the program.

5. Abbreviations

Table 3. Abbreviations

Acronym	Description
ATQA	Answer To Request, type A
API	Application Programming Interface
BAL	Bus Abstraction Layer
CMSIS	Cortex Microcontroller Software Interface Standard
FLASH	an electronic non-volatile computer storage medium
GPIO	General Purpose Input Output
HAL	Hardware Abstraction Layer
HW	Hardware
IC	Integrated Circuit
IRQ	Interrupt Request
KUC	Key Usage Counter
LLCP	Logical Link Control Protocol
LTO	Link Timeout
MCU	Microcontroller Unit
MF	MIFARE
MFC	MIFARE Classic
MFUL	MIFARE Ultralight
MIME	Multipurpose Internet Mail Extensions
NAD	Node Address
NAK	Negative Acknowledge
NDEF	NFC Data Exchange Format
NFC	Near Field Communication
OSAL	Operating System Abstraction Layer
P2P	Peer to Peer
PAL	Protocol Abstraction Layer
RAM	Random Access Memory
RTD	NFC Record Type Definition
SAK	Select Acknowledge, type A
SAM	Secure Access Module
SNEP	Simple NDEF Exchange Protocol
SPI	Serial Peripheral Interface
SW	Software
UID	Unique Identifier
URI	Uniform Resource Identifier

6. References

[1] **NXP Reader Library** - link to the software package of the library
http://www.nxp.com/documents/software/200312.zip

[2] **User Manual P2P Library CLRC663, PN512**
http://www.nxp.com/documents/user_manual/UM10721.pdf

[3] **Blueboard PNEV512B** – evaluation board
http://www.nxp.com/demoboard/PNEV512B.html

[4] **Blueboard CLEV663B** – evaluation board
http://www.nxp.com/demoboard/CLEV663B.html

[5] **LPCXpresso 1769** – low-cost development platform
http://www.nxp.com/demoboard/OM13000.html

[6] **LPCXpresso 1200** – low-cost development platform
http://www.nxp.com/demoboard/CLEV663B.html

[7] **LPCXpresso** IDE - software development tool
www.nxp.com/redirect/lpcware.com/lpcxpresso

[8] **SNEP** overview and specification
http://www.nxp.com/redirect/members.nfc-forum.org/specs/spec_list/prots

[9] **NDEF** overview and specification
http://www.nxp.com/redirect/members.nfc-forum.org/specs/spec_list/prots

[10] **RTD** overview and specification
http://www.nxp.com/redirect/members.nfc-forum.org/specs/spec_list/prots

[11] **LLCP** overview and specification
http://www.nxp.com/redirect/members.nfc-forum.org/specs/spec_list/prots

[12] **SNEP Validation Specification** is in preliminary version and no public release is available

[13] **Passive Target** – software application

[14] **CMSIS** – microcontroller software interface

http://www.nxp.com/redirect/lpcware.com/content/faq/lpcxpresso/cmsis-support

http://www.nxp.com/redirect/arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php

7. Legal information

7.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

7.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

7.3 Licenses

Purchase of NXP ICs with NFC technology

Purchase of an NXP Semiconductors IC that complies with one of the Near Field Communication (NFC) standards ISO/IEC 18092 and ISO/IEC 21481 does not convey an implied license under any patent right infringed by implementation of any of those standards.

7.4 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

8. List of figures

Fig 1. Main-Loop 7
Fig 2. Client Discovery-Loop 8
Fig 3. SNEP Client Demo 9
Fig 4. Close LLCP for Client 10
Fig 5. Server Discovery-Loop 11
Fig 6. SNEP Server Demo 12
Fig 7. Close LLCP for Server 12
Fig 8. Project structure 13
Fig 9. MCU library selection 14
Fig 10. MCU settings 15
Fig 11. Porting 32 to 16 bits timer – time in microseconds 22
Fig 12. Porting 32 to 16 bits timer – time in miliseconds 23

9. List of tables

Table 1. Timers naming and usage....21
Table 2. RAM and FLASH amount overview 23
Table 3. Abbreviations 29

10. Contents

1. Introduction 3
2. Application description 4
2.1 Overview 4
2.1.1 SNEP Overview 4
2.1.2 Example Setting Overview 4
2.1.2.1 Set Server or Client 4
2.1.2.2 SNEP Server Service Name 4
2.1.3 SNEP Server capabilities 5
2.1.4 SNEP Client capabilities 5
2.2 Passive Target Example Overview 7
2.2.1 Main-Loop Overview 7
2.2.2 SNEP Client Overview 8
2.2.3 SNEP Server Overview 10
2.2.4 Software Structure 13
3. Application Porting 16
3.1 Overview 16
3.1.1 BAL 16
3.1.2 OSAL 18
3.1.3 MCU drivers 20
3.1.4 Hardware module 20
3.1.5 Timers overview 20
3.1.6 Usage of the timers 20
3.1.7 Timers Porting 21
3.1.8 Global Variables 23
3.1.8.1 Reduction of global buffers 24
3.1.9 Reduced Variables in Practice 26
4. Build configuration 27
4.1 Hardware Development Kits 27
4.2 Software Development tools 27
4.3 Debug Build Configuration 27
5. Abbreviations 29
6. References 30
7. Legal information 31
7.1 Definitions 31
7.2 Disclaimers 31
7.3 Licenses 31
7.4 Trademarks 31
8. List of figures 32
9. List of tables 33
10. Contents 34

For more information, visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 11 August 2014
302310
Document identifier: AN11583