Software Examples Using ShockBurst™ Modes in nRF24L01 and nRF24LU1

nAN24-12

Application Note v1.0

February 2008

Liability disclaimer

Nordic Semiconductor ASA reserves the right to make changes without further notice to the product to improve reliability, function or design. Nordic Semiconductor ASA does not assume any liability arising out of the application or use of any product or circuits described herein.

Life support applications

These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Nordic Semiconductor ASA customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nordic Semiconductor ASA for any damages resulting from such improper use or sale.

Contact details

For your nearest dealer, please see http://www.nordicsemi.no

Receive available updates automatically by subscribing to eNews from our homepage or check our website regularly for any available updates.

Main office:

Otto Nielsen's vei 12
7004 Trondheim
Phone: +47 72 89 89 00
Fax: +47 72 89 89 89
www.nordicsemi.no

Revision History

Date	Version	Description
February 2008	1.0	

Contents

1 Introduction

This document uses software examples to describe how to use ShockBurst™ (SB), Enhanced ShockBurst™ (ESB) and Enhanced ShockBurst™ with Bidirectional data (PL), to transfer button status from one device to another. Also described in this document is how to port code to other hardware platforms.

The nAN24-12SW package contains source code that helps you set up the nRF24L01 and nRF24LU1 in the ShockBurst™ modes. The source code is written in C and contains all the files necessary for setting up the devices using only the Hardware Abstraction Layer (HAL). The source code can also be used as a reference when you are developing your own application.

You will need two Basic Feature Boards (BFB), which are included in the nRF24L01 and nRF24LU1 development kits, programmed with the nAN24-12 firmware to run the application.

nRF24L01 and nRF24LU1 use the following hex files for demonstration when plugged into the BFB:

- Basic Feature Board with C8051F320 + nRF24L01: nRF24L01.hex
- Basic Feature Board with nRF24LU1: nRF24LU1.hex

The BFB must be powered on, but does not need to be connected to any peripherals to run the application example. The buttons and LEDs on the BFB are used for user input and output.

1.1 Prerequisites

In order to fully understand the application note, a good knowledge of electronic and software engineering is required.

1.2 Writing conventions

This application note follows a set of typographic rules that makes the document consistent and easy to read. The following writing conventions are used:

- Commands and code are written in `Courier`.
- File names are written in **bold**.
- Cross references are underlined and highlighted in blue.

2 nAN24-12 contents and requirements

This chapter describes the contents of the nAN24-12SW package and the hardware that is required for demonstrating the different operational modes. Refer to chapter 6 for information on porting the source code to other hardware architectures.

2.1 Contents

The software package contains the following files:

- Documentation (/doc)
  - nAN24-12 Application Note
  - Windows help file (**help.chm**)
- Source code (/src)
  - The Keil project files for both nRF24L01 and nRF24LU1
  - Complete source code for the application
  - Complete source code for the necessary HAL libraries
- Firmware files (/hex)
  - The pre-compiled hex files for nRF24L01 and nRF24LU1

2.2 Required software

The application has been tested and compiled on KEIL µVision3 3.53.

2.2.1 nRF24L01 specific software

To program the SiLabs C8051F320 MCU, which is part of the nRF24L01 development kit, you can use either the built in functionality in the Keil software package or you can download the Flash Programming Utilities from SiLabs: http://www.silabs.com

2.2.2 nRF24LU1 specific software

Refer to the documentation for the nRF24LU1 Development Kit on how to program the nRF24LU1.

2.3 Required hardware

To work out of the box, this application needs either the development kit for nRF24L01 or nRF24LU1.

2.3.1 nRF24L01 specific hardware

To program the nRF24L01 development kit you need a programmer that supports the SiLabs C8051F320 MCU. The low cost USB debug adapter from SiLabs has been used during the development: http://www.silabs.com

You need to solder the four pin programming/debug interface on the dongle to a simple interface to support the 10 pin connector on the USB debug adapter. See Figure 1. for the schematic.

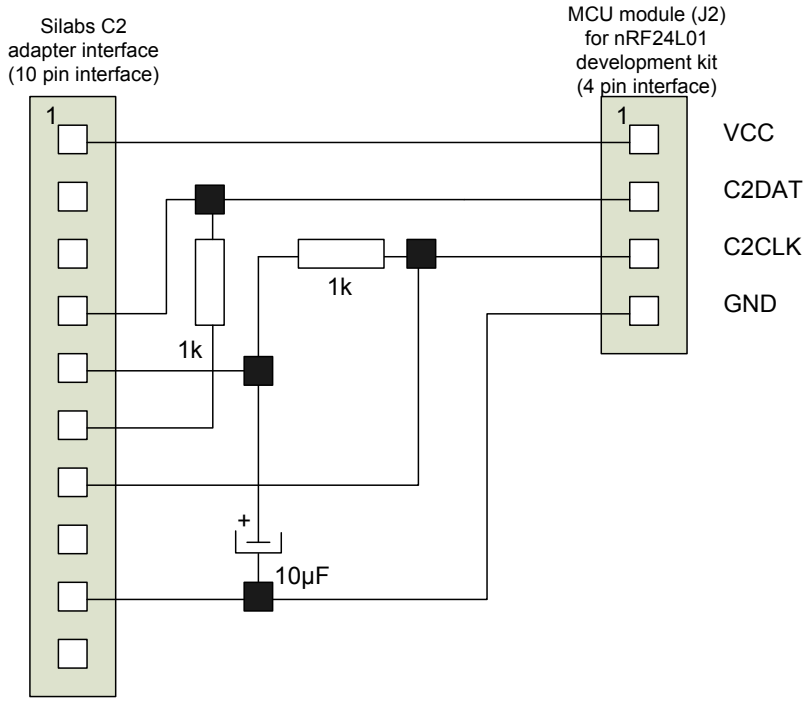


Figure 1. SiLabs modification

2.3.2 nRF24LU1 specific hardware

All the necessary hardware for the nRF24LU1 is included in the nRF24LU1 Development Kit.

3 SW architecture

The software (SW) architecture is shown in Figure 2. A source and header file represent each block. See the help documentation for more information on the prototypes and macros.

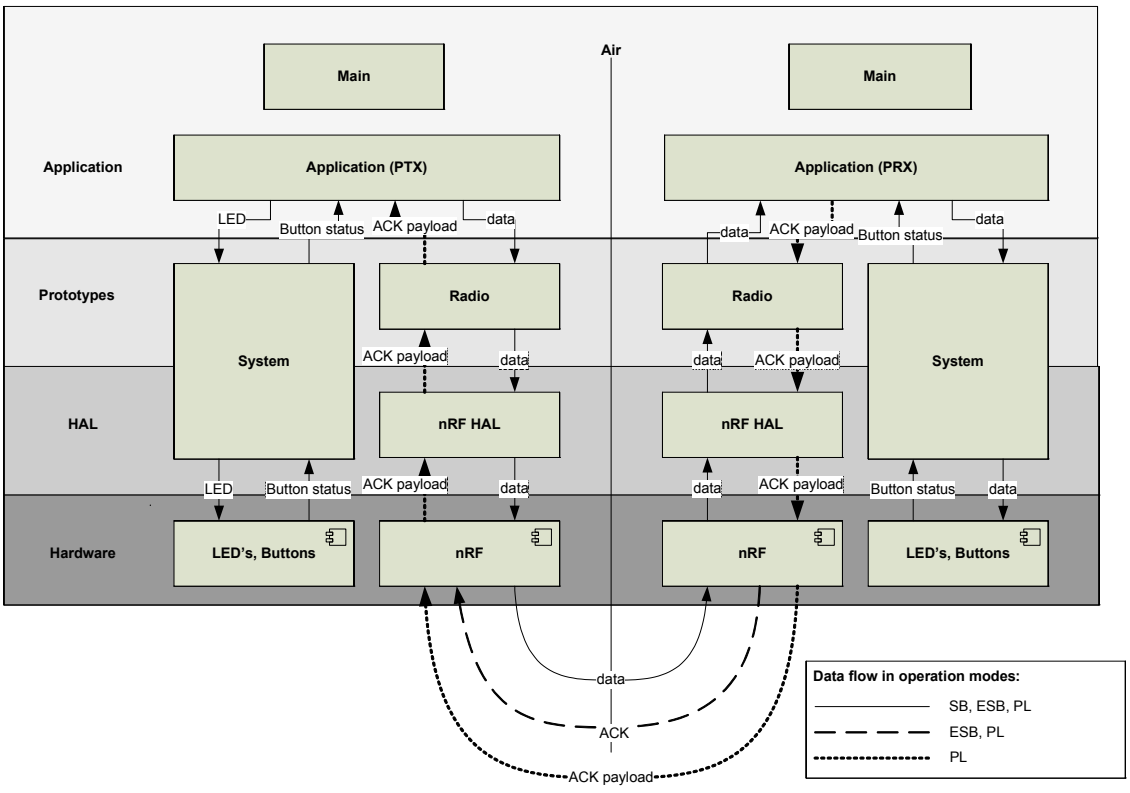


Figure 2. SW Architecture

3.1 Main

The main application is a state machine controlling your choices of operational modes and initialization. It starts up, initializes the microcontroller, and waits for you to select the application mode before initializing the radio and starting the application. After starting the application, the main application releases all control.

The main application defines the following prototypes and typedefs:

Prototypes:

```
state_t get_next_state (state_t current_state);
void wait_for_button_release (void);
void show_status (state_t operation);
```

Typedef:

```
enum {
  DEVICE_IDLE = 0,
  DEVICE_PRX_IDLE,
  DEVICE_PTX_IDLE,
  DEVICE_PRX_SB,
  DEVICE_PRX_ESB,
  DEVICE_PRX_PL,
  DEVICE_PTX_SB,
  DEVICE_PTX_ESB,
  DEVICE_PTX_PL,
  NO_CHANGE
} state_t;
```

3.2 Application

The application handles the logic of the program. It contains the logic for both the Primary Transceiver (PTX) and the Primary Receiver (PRX) in separate functions. There is one implementation for each of the operational modes.

The application defines the following external prototypes:

Prototypes (ShockBurst™):

```
void device_ptx_mode_sb (void);
void device_prx_mode_sb (void);
```

Prototypes (Enhanced ShockBurst™):

```
void device_ptx_mode_esb (void);
void device_prx_mode_esb (void);
```

Prototypes (Enhanced ShockBurst™ with bidirectional data):

```
void device_ptx_mode_pl (void);
void device_prx_mode_pl (void);
```

3.3 System

The system controls the interface to the MCU and the development kit. All hardware interaction is performed through these functions.

The following external prototypes and macros are defined in the system:

Prototypes:

```
void device_boot_msg (void);
void delay_10ms (void);
void delay_100ms (void);
void start_timer (uint16_t time);
void wait_for_timer (void);
bool timer_done (void);
```

Macros:

```
LEDx_ON (); (One each for LED1, LED2 and LED3)
LEDx_OFF (); (One each for LED1, LED2 and LED3)
LEDx_BLINK (); (One each for LED1, LED2 and LED3)
LED_ALL_OFF ();
T0_START ();
T0_STOP ();
T1_START ();
T1_STOP ();
```

3.4 Radio

The radio contains all the necessary functions to send and receive packages. Each operation mode (ShockBurst™, Enhanced ShockBurst™, and Enhanced ShockBurst™ with bidirectional data) has a separate initiation function.

The radio defines the following external prototypes, constants and typedefs:

Prototypes:

```
void radio_send_packet (uint8_t *packet, uint8_t length);
radio_status_t radio_get_status (void);
void radio_set_status (void);
uint8_t radio_get_pload_byte (uint8_t byte_index);
void radio_irq (void);
```

ShockBurst™:

```
void radio_init_sb (uint8_t *address, hal_nrf_operation_mode_t
operational_mode);
```

Enhanced ShockBurst™:

```
void radio_init_esb (uint8_t *address, hal_nrf_operation_mode_t
operational_mode);
```

Enhanced ShockBurst™ with bidirectional data:

```
void radio_init_pl (uint8_t *address, hal_nrf_operation_mode_t
operational_mode);
```

Constants:

```
RF_CHANNEL 40;
RF_POWER_UP_DELAY 2;
PAYLOAD_LENGTH 2;
RF_RETRANSMITS 15;
RF_TRANS_DELAY 250;
```

Typedef:

```
enum {
  RF_IDLE,
  RF_MAX_RT,
  RF_TX_DS,
  RF_RX_DR,
  RF_TX_AP,
  RF_BUSY
} radio_status_t;
```

3.5 nRF HAL

The nRF HAL (Hardware Abstraction Layer) contains all the functions that abstracts communication with the selected radio. These are specific for the chosen radio, but implement the same interface, **hal_nrf.h**.

4 Structure

The source code is divided into logical parts for easy reading. To enable easy porting to different hardware platforms, the hardware specific files are placed in separate subfolders for each architecture. The default supported platforms are:

- SiLabs C8051F320+nRF24L01
- nRF24LU1

There are also separate folders for each operational mode (ShockBurst™, Enhanced ShockBurst™, and Enhanced ShockBurst™ with bidirectional data)

For information on porting, see chapter 6.

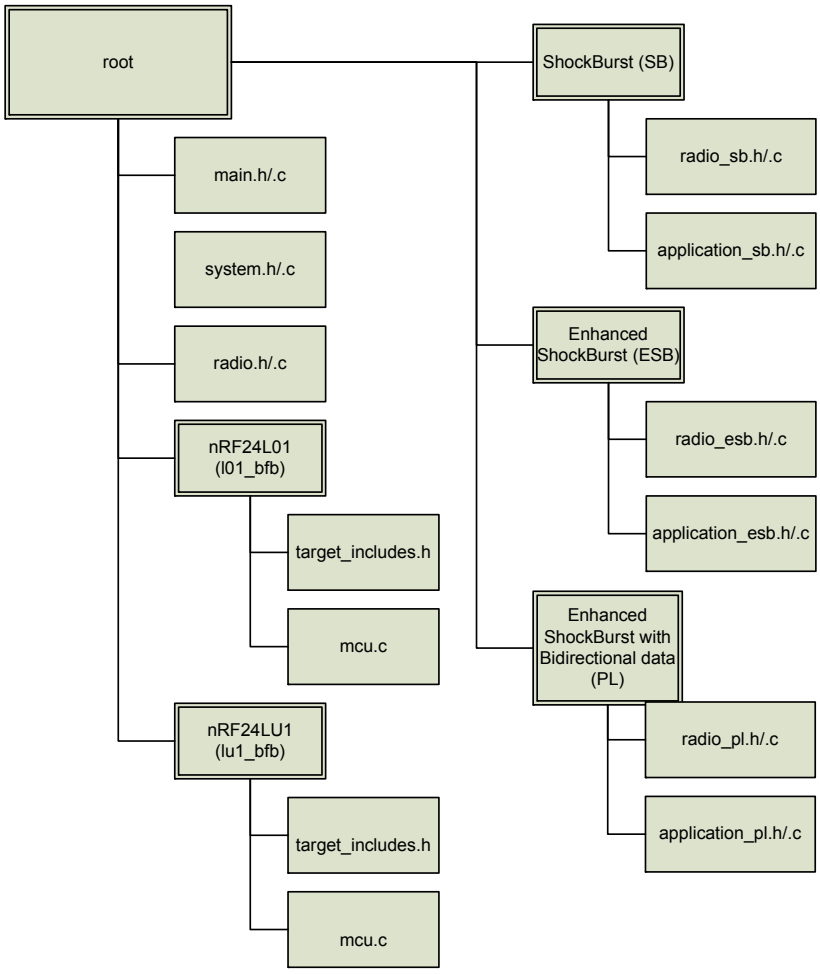


Figure 3. File and folder structure

5 Application

This chapter describes how to operate the application. How to choose an operational mode is described in section 5.1.The different operational modes are described in sections 5.2 to section 5.4.

5.1 Easy setup for demonstration

Figure 4. shows how to set up a board in each mode. The first step is to choose between PTX and PRX mode and the second step is to set up the desired operation mode. For example, the board can be set up to PTX ShockBurst™ mode by pressing Button 1 twice. After you set up the operation mode, the LED stays lit for three seconds to indicate that the set up is successful.

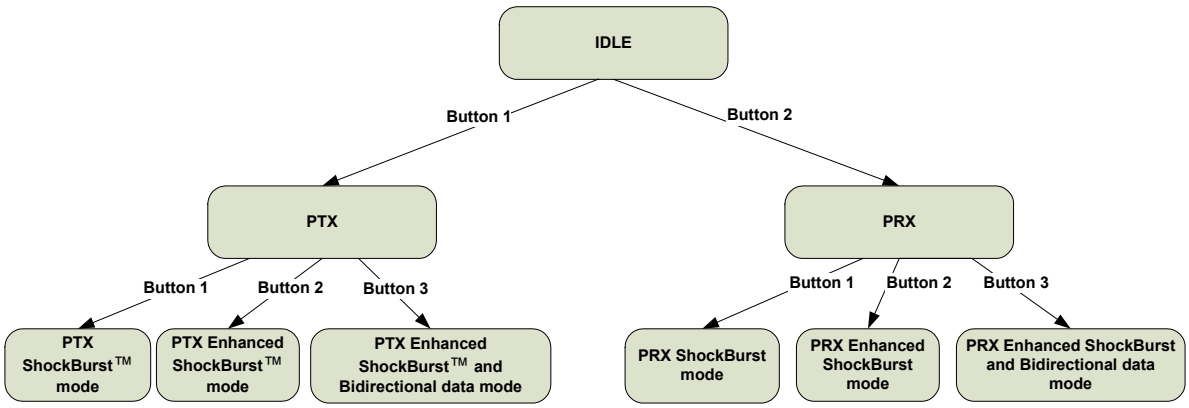
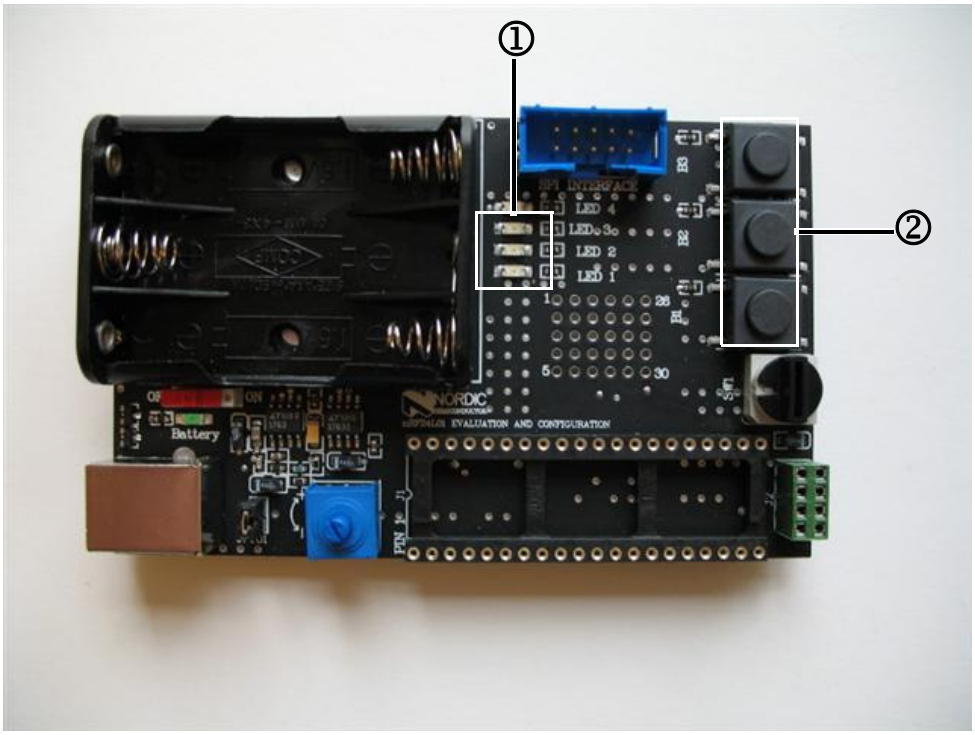


Figure 4. Setting up the board for demonstration

Mode	LED status		
	LED 1	**LED 2**	**LED 3**
Idle mode	ON	ON	ON
PTX	OFF	OFF	OFF
PRX	OFF	OFF	ON
PTX ShockBurst™	ON	OFF	OFF
PRX ShockBurst™	ON	OFF	ON
PTX Enhanced ShockBurst™	OFF	ON	OFF
PRX Enhanced ShockBurst™	OFF	ON	ON
PTX Enhanced ShockBurst™ and Bidirectional data	ON	ON	OFF
PRX Enhanced ShockBurst™ and Bidirectional data	ON	ON	ON

Table 1. LED status of each mode



① LEDs (1,2,3) ② Buttons (1,2,3)

Figure 5. nRF24LU1 Basic Feature Board (BFB)

5.2 ShockBurst™

You can set up the PTX and PRX to ShockBurst™ with no advanced features enabled. The PTX is configured without auto retransmit, and the PRX is configured without auto ack. ShockBurst™ is used to broadcast data.

ShockBurst™ is demonstrated by pressing Button 1 on the PTX which then turns on the LED 1 on the PRX.

See Figure 6. (a) for the flow of PTX in ShockBurst™. Press Button 1 twice to get to this mode.

See Figure 6. (b) for the flow of PRX in ShockBurst™. Press Button 2 followed by Button 1 to get to this mode.

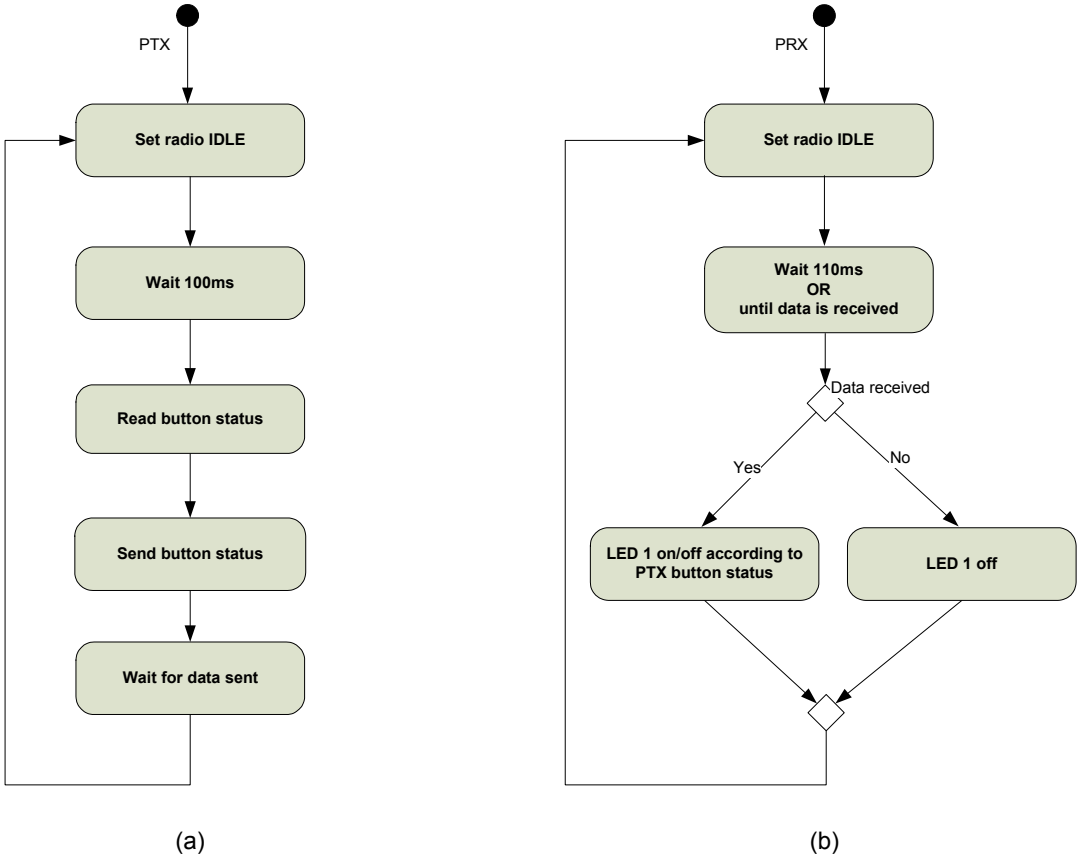


Figure 6. Application flow in ShockBurst™

5.3 Enhanced ShockBurst™

You can set up the PTX and PRX to Enhanced ShockBurst™ with advanced features enabled. The PTX is configured with auto retransmit, and the PRX is configured with auto acknowledgement.

The advantages of using Enhanced ShockBurst™ are:

- One way communication.
- Auto acknowledge and auto retransmit data.

Enhanced ShockBurst™ is demonstrated by pressing Button 1 on the PTX, which then turns on LED 1 on the PRX. The PTX indicates successfully sent data by turning LED 2 on, or if max retransmission occurs this is indicated by the PTX turning LED 3 on.

See Figure 7. (a) for the flow of PTX in Enhanced ShockBurst™. Press Button 1 followed by Button 2 to get to this mode.

See Figure 7. (b) for the flow of PRX in Enhanced ShockBurst™. Press Button 2 twice to get to this mode.

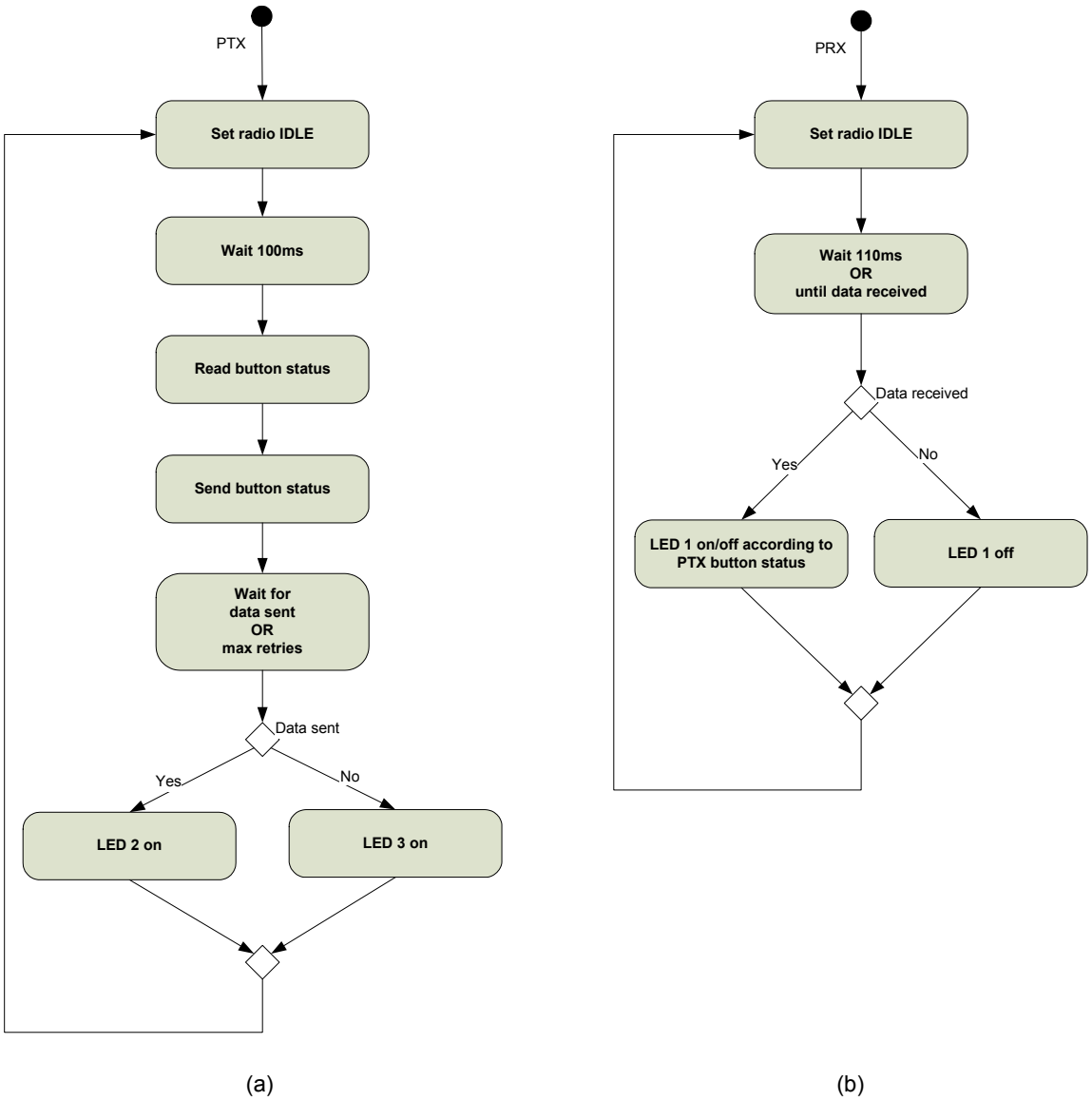


Figure 7. Application flow in Enhanced ShockBurst™

5.4 Enhanced ShockBurst™ with bidirectional data

You can set up the PTX and PRX to Enhanced ShockBurst™ with advanced features enabled. The PTX is configured with auto retransmit and ack payload, and the PRX is configured with auto ack and ack payload.

The advantages of using the Enhanced ShockBurst™ with bidirectional data are:

- Bidirectional communication
- Auto acknowledge and auto retransmit data
- Dynamic payload length
- Payload with acknowledgement

Enhanced ShockBurst™ with Bidirectional data is demonstrated by pressing either Button 1 on the PTX, which then turns on LED 1 on the PRX or, by pressing Button 1 on the PRX, which then turns on LED1 on the PTX. Both the PTX and the PRX indicate successfully sent data by turning LED 2 on, or if max retransmission occurs this is indicated by LED 3 on.

See Figure 8. (a) for the flow of PTX in Enhanced ShockBurst™ with Bidirectional data. Press Button 1 followed by Button 3 to get to this mode.

See Figure 8. (b) for the flow of PRX in Enhanced ShockBurst™ with Bidirectional data. Press Button 2 followed by Button 3 to get to this mode.

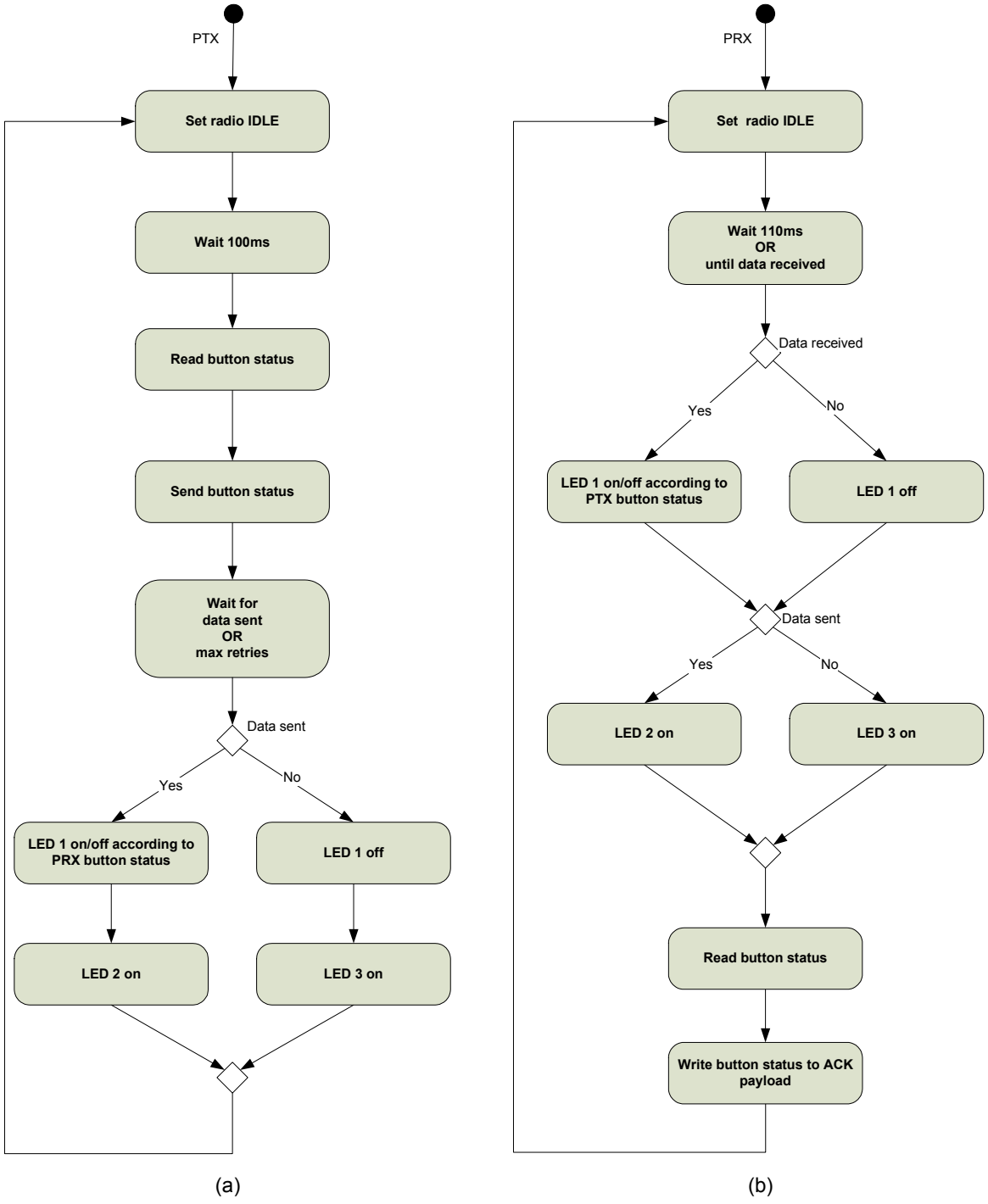


Figure 8. Application flow in Enhanced ShockBurst™ with bidirectional data

6 Porting the source code to another hardware architecture

You must re-map the hardware to port this application to another hardware architecture. Most of the hardware dependent parts of the code are in the MCU specific subfolders. The hardware dependent files for the nRF24LU1 are located in the **lu1_bfb** folder. The hardware dependent files for the nRF24L01 are located in the **l01_bfb** folder. When porting to another MCU, a new folder must be made containing the following files:

- mcu.c
- target_includes.h

If you want a new Keil project, you should base it on the **nRF24LU1.uV2** with the following changes:

1. Update the device architecture to your target device.
2. Name of the compiled hex file should reflect the architecture.
3. Paths should include:
   a. The new folder for the MCU.
   b. The correct HAL library for the radio architecture.
4. Update the HAL group to contain the HAL files for the architecture.
   a. `hal_nrf_hw.c` (implementation of `hal_nrf_rw` from **hal_nrf.h**).
   b. `hal_nrf_x.c` (*x* indicating MCU. Implementation of **hal_nrf.h**).

6.1 mcu.c

This file needs one function: `void system_init (void);` This function should set up three inputs and three outputs (B1-3 and LED1-3 respectively).

6.2 target_includes.h

This file contains most of the hardware dependent settings. The hardware abstractions that must be ported are in Table 2.

Name	Description
LED_ON	The value that should be sent on the output to turn on a LED
LED_OFF	The value that should be sent on the output to turn off a LED
LED1	Address of LED1 (bit address)
LED2	Address of LED2 (bit address)
LED3	Address of LED3 (bit address)
B1	Address of B1 (bit address)
B2	Address of B2 (bit address)
B3	Address of B3 (bit address)
B_PRESSED	Value on input that indicate button pressed
RADIO_ACTIVITY	The way to check if the radio is active without using interrupts
RESET_RADIO_ACTIVITY	How to reset the radio detection so that a new packet can be detected
TIMER1_OVERFLOW	Check for Timer 1 overflow
CYCLES_PR_MS	The number clock cycles a timer counts that make up a millisecond
MAX_RUNTIME	How many milliseconds the timer can run on a 16-bit counter (0xFF / CYCLES_PR_MS)
GLOBAL_INT_ENABLE	Enables all interrupts
GLOBAL_INT_DISABLE	Disables all interrupts

Name	Description
T0_START	Starts Timer 0
T0_STOP	Stops Timer 0
T1_START	Starts Timer 1
T1_STOP	Stops Timer 1
T1_MODE1	Sets timer 1 up in mode 1 (16-bit timer)
T1_SET_LB	Sets the low byte on timer 1
T1_SET_HB	Sets the high byte on timer 1
INTERRUPT_T0	The interrupt vector for timer 0

Table 2. Hardware dependencies

6.3 Interrupt

If the MCU does not have an interrupt vector for timer 0, some of the LED functionality will not work. The workaround for this problem is to change the `LEDx_BLINK` functions in **system.h** to call `LEDx_TOGGLE` instead and remove the `t0_ov_interrupt()` function from **system.c**.

No other interrupts are used in this application. The interrupt status for timer 1 and radio are polled, but not handled as interrupts.

7 Troubleshoot

Q: The board does not flash after I reset. What could be the problem?

A: There could be two reasons why this is happening:

i. The board may not be powered up. In this case, ensure that the board is powered up through either the USB or the battery.

ii. The program may not be loaded. In this case, flash the development kit with the correct HEX file for your MCU and radio.

Q: LED 1 on the receiver does not light up when I press Button 1 on the transmitter. What could be the problem?

A: There could be three reasons why this is happening:

i. The button may not be fully pressed. In this case, press the button harder.

ii. The receiver or transmitter may not be powered up. In this case, power up and configure.

iii. The receiver and transmitter are in non-compatible operational modes. Restart and configure to the correct operational mode.

Q: ACK is not receiving on the transmitter (LED 3 flashing).

A: There could be two reasons why this is happening:

i. The receiver may be out of range. In this case, bring the receiver into range.

ii. The receiver is in a different operational mode. In this case, restart and configure to correct operational mode.

8 Glossary of terms

Term	Definition
ACK	Acknowledgment
BFB	Basic Feature Board
Enhanced ShockBurst™ (ESB)	A packet based data link layer that features automatic packet assembly and timing, automatic acknowledgement and retransmissions of packets.
Enhanced ShockBurst™ with Bidirectional data (PL)	A packet based data link layer that features automatic packet assembly and timing, automatic acknowledgement, retransmissions of packets and transmission of data from PRX to PTX in ACK so that no mode change is required.
HAL	Hardware Abstraction Layer
HW	Hardware
MCU	Microcontroller
PRX	Primary receiver
PTX	Primary transceiver
SDK	Software Development Kit
ShockBurst™ (SB)	This mode gives you automatic packet transmission at a high data rate.