nRF24Z1 Controlled by External MCU

nAN24-10

1. Motivation

The nRF24Z1 is a complex chip capable of transmitting digital audio signals, establishing links, and maintaining quality of service in internal firmware. However, the chip must be instructed which address and frequency scheme to use, when to go into power-down mode, and how to interact with a user interface and external chips. This document describes a software package that lets you access internal registers in the nRF24Z1 from a microcontroller (MCU). The software is written in C for easy development and porting. The program functions may be used stand-alone in an MCU program. A debug program is included so that you may call the functions from a terminal emulator running on a PC.

2. Limitations

All functions in this document are written for nRF24Z1 firmware version 2.0 or later. The 2-wire slave interface (described below) is only supported in Engineering Sample 2 and later versions of the nRF24Z1 chip. Nordic Semiconductor is not responsible for the porting of the MCU program to other processor platforms.

3. Typical System

The supporting MCU program is written for a system with user and control interfaces at the audio transmitter (ATX) and/or at the audio receiver (ARX). In this general case, the ATX features the nRF24Z1 in transmit mode, an MCU, an EEPROM[1] and a digital audio source (ADC, S/PDIF interface, I2S interface). The ARX features the nRF24Z1 in receive mode, an EEPROM, and a digital audio destination (amplifier, volume control, DAC etc). The system is controlled by the MCU in the ATX. (The MCU program may be adapted to go into an MCU in the ARX.)

Here are some applications that may be realized with this general system:

- CD/MP3 player (ATX) with wireless headphones (ARX)
- HiFi stereo headphones with line-in on the ATX
- Wireless microphone

[1] The EEPROM in the ATX is required for engineering sample versions of nRF24Z1. With the production-version of the chip, it is not strictly required, as initial register values may be loaded by the MCU program. However, it is recommended to design the ATX with room for the EEPROM.

4. Program Features

Version 1.0 of the MCU program features functions for nRF24Z1 register access and debug. Later versions are in development, where a full API will be provided. Version 1.0 of the program is split in the following four parts:

1. General and nRF24Z1 specific functions in `main.h`, `main.c`, `z1slaveio.h`, `z1slaveio.c`
2. MCU specific functions in `mcu.h`, `mcu_atmega16.c`
3. DAC specific functions in `dac.h`, `dac_max9850.c`
4. Debug functions in `uartdebug.h`, `uartdebug.c`

The program lets you control internal registers on a byte-by-byte level. It also features higher-level code to control the volume of a MAX9850 DAC and headphone amplifier. User interfaces and control interfaces to the audio source are not supported in this version.

Most files are general purpose. There are four exceptions:

- Code that depends on the MCU and its architecture are in `mcu_xxxx.c`, where `xxxx` represents the MCU name. As a programmer, you will need to consult the technical documentation of your MCU to port the required functions.
- `mcu.h` holds compiler and architecture dependant `#include` lines. You will have to port this file as well. There should be no need to change the function definitions in `mcu.h` when you port the program to a different MCU.
- Code that depends on your DAC is collected in `dac_yyyy.c`. You should not change `dac.h`. That file is included in the rest of the program. The DAC datasheet holds information about how to configure other DACs than the described MAX9850.
- `makefile` may have to be changed to work with your compiler. It also has to list which source files you want to compile.

4.1. General and nRF24Z1 Specific Functions

A `main()` program initiates the necessary hardware and starts a debug interface. `main.h` includes global configuration settings. The most important setting is to select *either* the SPI *or* the 2-wire slave interface of the nRF24Z1 to be used with the MCU code.

These are the nRF24Z1 specific functions in `z1slaveio.h`:

```
char z1_singleread (char adr);
```
Reads and returns a single byte from an internal register (`adr`) in nRF24Z1 using its SPI or 2-wire slave interface (and corresponding master interface of the MCU)

```
void z1_singlewrite (char adr, char data);
```
Writes a single byte (`data`) to an internal register (`adr`) in nRF24Z1

```
void z1_multiread (char startadr, char endadr);
```
Reads multiple consecutive bytes (`startadr` through `endadr`) from nRF24Z1 internal registers into the global array `slaveinbuf[]`.

```
void z1_multiwrite (char startadr, char endadr);
```
Writes multiple consecutive bytes (`startadr` through `endadr`) into nRF24Z1 internal register from the global array `slaveoutbuf[]`.

```
char z1_flagready(char flag);
```

Returns `OKAY` if a given transfer register (`flag`) indicates that the nRF24Z1 is ready to accept new data. Returns `TIMEOUT` if the transfer register did not indicate accept for new data in `MAXPOLLITER` iterations.

```
void z1_setflag(char flag);
```
Tells a given transfer register (`flag`) that the new values in the according set of nRF24Z1 registers are to be used across the wireless link

```
char z1_haslink(void);
```
Returns 1 (true) if there is a valid audio link present. Returns 0 (false) otherwise.

```
char z1_read_eeprom(char adrh, char adrl);
```
Access a Microchip 25AA640 or compatible EEPROM located on the master SPI bus of the ARX. Return the byte in 16-bit address `adrh_adrl`.

```
char z1_write_eeprom(char adrh, char adrl, char data);
```
Write `data` to address the 16-bit address location `adrh_adrl` in a Microchip 25AA640 or compatible EEPROM on the master SPI bus of the ARX.

4.2. MCU Specific Functions

The program is intended for easy porting to multiple MCU architectures. All functions whose implementations vary with MCU architecture are defined in `mcu.h`. Everything that depends on specific MCU hardware or special hardware pins has to be put here. The implementations are put in a file named `mcu_xxxx.c`, where `xxxx` represents which MCU is used. As a naming example, consider `mcu_atmega16.c` for the Atmel ATmega16.

When MCU specific functions are used by the rest of the program, `mcu.h` is included. If an engineer decides to write `mcu_xxxx.c` for a chosen MCU, the code must be compatible with `mcu.h`. The supplied `makefile` must also be edited.

These are the functions defined in `mcu.h`:

```
void mcu_init(void);
```
Configures the MCU itself.

```
void mcu_wait_ms(int time);
```
Halts the MCU for the number of milliseconds give in `time`.

```
void mcu_spi_master_init(void);
```
Initiates the MCU's master SPI interface. The program uses either SPI or 2-wire as a communication channel between the MCU and nRF24Z1. In `main.h`, define either `SLAVE_SPI` or `SLAVE_2W`.

```
void mcu_spicycle(char startadr, char endadr);
```

Transmits data between the SPI interface and global buffers `slaveinbuf[]` and `slaveoutbuf[]`. `startadr` and `endadr` are modified with the MSB being 0 for an SPI read access and 1 for an SPI write access.

`void mcu_2w_master_init(void);`
Initiates the MCU's master 2-wire interface.

`void mcu_2wread(char startadr, char endadr);`
Transmits data from the 2-wire slave interface of nRF24Z1 to global buffer `slaveinbuf[]`.

`void mcu_2wwrite(char startadr, char endadr);`
Transmits data from global buffer `slaveoutbuf[]`to the 2-wire slave interface of nRF24Z1.

`void mcu_uart_init(void);`
Initiates a UART debug interface for 57600 bps, 8 bits data, 1 stop bit, no parity.

Beware that these are only the functions that are visible from other files in the program. It may be required to change additional support functions defined in `mcu_xxxx.c`.

4.3. DAC Specific Functions

Just like various MCUs may be used by the rest of the program, DAC specific functions are all collected in one file so that the other source files have a consistent set of DAC commands. The functions are defined in `dac.h` and implemented in `dac_yyyy.c`. The file supported MAX9850 has `dac_max9850.c` associated with it. The engineer wishing to use a DAC that is not supported must make a new `dac_yyyy.c` file and edit the supplied `makefile`. All functions must be compatible with `dac.h` in order to work with rest of the program.

These are the functions in `dac.h`:

`char dac_init(void);`
Initializes the DAC. Returns `OKAY` or `TIMEOUT`.

`char dac_volume_up(void);`
`char dac_volume_down(void);`
Increase or decrease the volume. The functions use the global variable `volume` defined in `main.h`. The value of `volume` are only altered by volume control functions. The exception is that `main.c` sets it to `0xFF` at power-on reset in order to tell `dac_init()` to initialize it. The functions return `OKAY` or `TIMEOUT`.

4.4. Debug Functions

A UART debugging subsystem has been included with the program. The program gives direct access to all nRF24Z1 ATX registers. It also initiates and controls the volume of a DAC in the ARX.

The most basic functions of the debug subsystem depend on MCU hardware, and are thus defined in `mcu.h` and implemented in `mcu_xxxx.c`. The other parts are in `uartdebug.h` and `uartdebug.c`. Macros to support the `printf()` functions of various compilers are written in `uartdebug.h`.

These are the externally visible functions in `uartdebug.h`:

`void db_singleread(char adr);`
Read a single byte from an internal nRF24Z1 register (`adr`) and prints it to the UART interface.

`void db_singlewrite(char adr, char data);`
Writes a single byte (`data`) to an internal nRF24Z1 register (`adr`) and prints the result to the UART interface.

`void db_multiread(char startadr, char endadr);`
Reads multiple consecutive nRF24Z1 internal registers (`startadr` through `endadr`) and prints the result to the UART interface

`void db_multiwrite(char startadr, char endadr);`
Prompts for new data to write into consecutive nRF24Z1 internal registers. Calls `db_multiread()` with the same addresses after it is done.

`void db_hex(void);`
A hex debug interface that lets the engineer inspect all nRF24Z1 internal registers. The debug interface commands are explained in the help text in `uartdebug.c`. As an alternative to breakpoints, you may insert a call to `db_hex()` in your code.

The hex debug interface will prompt (`>`) for your command. Here are some use examples:

`> 14¶`
Displays the content of nRF24Z1 internal address 0x14

`> 14=3b¶`
Writes 0x3b to nRF24Z1 internal address 0x14. Then displays the content of address 0x14 for verification

`> 14:18¶`
Displays the content of nRF24Z1 internal address 0x14 through 0x18. At most `SLAVEBUFSIZE` (defined in `main.h`) bytes may be transferred at a time.

`> 14-18¶`
Prompts for new content to be written to internal addresses 0x14 through 0x18. Then displays the contents of address 0x14 through 0x18 for verification. At most `SLAVEBUFSIZE` (defined in `main.h`) bytes may be transferred at a time.

`> H`
Displays help information.

`> p`
Initiates the DAC. This requires that a supported DAC is placed on the nRF24Z1 ARX board.

`> +`
Increases DAC volume. Requires supported DAC.

`> -`
Decreases DAC volume. Requires supported DAC.

5. Development Environment Setup

The MCU program is intended to work on different compilers and MCU platforms. This section describes various setups. It is not required that you use the same MCU and compiler as the program was developed on. However, Nordic Semiconductor is not able to support program development on other MCUs.

5.1. Atmel ATmega16 in STK500 Board, WinAVR Compiler

The MCU program in version 1.0 was developed on an ATmega 16L in an Atmel STK500 demo board. This section describes the setup that was used during development. Please consult your local Atmel distributors if you wish to purchase a demo board or if you have questions.

The WinAVR compiler is a GCC port for Atmel's AVR MCUs. It can be found at:
http://winavr.sourceforge.net/

To program and configure the STK500 and Atmel MCUs, we recommend that you install AVR Studio 4 which can be found on:
http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725

STK500 is described on this web page:
http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2735

On the PC, the debug terminal should be set up for 57600 baud, 8 data bits, 1 stop bit. This version of the MCU program uses a terminal for reading and writing configuration bytes to/from the nRF24Z1. In your production version, this debugging is omitted. We recommend Tera Term as terminal emulator. It may be found at:
http://hp.vector.co.jp/authors/VA002416/teraterm.html

The `main_atmega16.hex` file is compiled for an Atmel ATmega16L microcontroller. To run this .hex file, you will need to purchase an ATmega16L (L means it is low-voltage tolerant) in a 40-pin DIP package and install it in the STK500. The chip is described here:
http://www.atmel.com/dyn/products/product_card.asp?part_id=2010

After installing and purchasing the required software and equipment, you need to set up the STK500 as follows:

- Use a power supply of 9-12V. When you power up the STK500, the green led next to the VTARGET jumper must be lit. The board tolerates both positive and negative polarity at the center pin of the power supply connector.
- With the power off, insert the ATmega16L in SCKT3100A3.
- RX232 Spare RXD connects to PD0 (headers near the pushbuttons)
- RX232 Spare TXD connects to PD1 (headers near the pushbuttons)
- VTARGET connected
- AREF connected
- RESET connected
- XTAL1 connected
- OSCSEL connected pins 1-2
- BSEL2 open
- PJUMP both open
- ISP6PIN connects to red socket with 6-pin flat cable
- Set the SSEL jumper on the ATX board to "ON"
- A special SPI cable connects between the STK500 and the nRF24Z1 ATX evaluation board. (See Table 1)

The SPI cable connects the ATX SPI slave connector to PORTB on the STK500. Table 1 shows the connection:

STK500 pin number	ATX pin number	STK500 pin name	ATX pin name
NC	1	NC	VDD
8	2	PB7	SSCK
6	3	PB5	SMOSI
5	4	PB4	SCSN
7	5	PB6	SMISO
9	6	GND	GND

Table 1: *SPI cable for STK500*

If you rather want to use 2-wire, (with ES2 or later versions of the nRF24Z1), do the following:

- Place 10kΩ pull-up resistors in R12 and R34 on the nRF24Z1 ATX board
- Set the SSEL jumper to "OFF"
- Set the MZ_ADR jumper according to the definitions in the 2-wire part of `mcu_xxxx.c`
- Connect PC0 and PC1 on the STK500 to pins 1 and 3, respectively, on JP7 on the nRF24Z1 ATX board.
- In `main.h`, comment the line "`#define SLAVE_SPI`" and uncomment the line "`#define SLAVE_2W`"
- Recompile the MCU program

Connect a serial cable between a COM port on your PC and "RS232 CTRL" on the STK500. You will now need to start AVRstudio on your PC. Please click the "AVR" chip symbol on

the toolbar of AVRstudio. An interface to the board should now open. Choose “Advanced” and click “Read Signature” to verify that the ATmega16L is detected.

You will need to set up the voltages by clicking “Board”. Set the voltages at 3.2V. Click “Write voltages”.

You also have to set up the oscillator on the STK500. In the “Oscillator and ISP Clock” sub window, choose STK500 Osc = 3.686MHz. Then click “Write” in the same sub window.

You are now ready to program the ATmega16L microcontroller. Please go to “Program”. In the “Flash” sub window, browse to `main_atmega16.hex` by clicking the “…” button. Next, press “Program” to set up the microcontroller.

After programming the MCU, exit AVRstudio and connect the serial cable from the COM port of the PC to “RS232 SPARE” on the STK500. Start your favorite terminal emulator program. The debug terminal should be set up for 57600 baud, 8 data bits, and 1 stop bit.

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 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.

Application Note, Revision: 1.0, Date: 2005-07-25.

Application Note order code: nAN24-10

YOUR NOTES