Using the Smart RF Adaptive Frequency Hopping Spread Spectrum (AFHSS) Firmware with the Reference Design Based on ATR2406 and ATmega88

Features

- Interfaces the ATR2406 With an AVR® Microcontroller
- Needs 1000 Words of Flash Memory
- Implements the Complete RF Flow
- Generates All Needed Timings
- Implements a Protocol Layer
- Implements All Necessary Low Level Drivers
- RAM Usage Reduced to a Minimum of Approximately 140 Bytes Plus Data Stack (36 Bytes)
- Implements an Adaptive FHSS System
- Three Different Hopping Schemes
  - Random Hopping
  - Adaptive Random Hopping
  - No Hopping
- Power Saving Features
- Easily Adjustable to the Necessary Effective Data Rate
- 33% of CPU Needed for RF RX Task
- 30% of CPU Needed for RF TX Task
- Achievable Effective Data Throughput: up to 384 Kbps
- Flexible Payload Length: 0 Bytes to 48 Bytes per Packet
- Needs Only One Hardware Timer and One USART Interface
- RF Task Completely Independent From Application Task
- RF Raw Data Rate: 1.152 Mbps
- Automatic Setup of RF Link
- Configures the ATR2406
- Synchronizes the RF Link (Time and Channel)
- True Random Generator for RF Purposes
- Automatic RF Packet Handling (Analysis and Setup)
- Four RF Test Modes
- Automatic Hardware Self Test
- Only Eight I/O Lines are Needed to Connect the ATR2406 to an MCU
- Configurable Hopping Rate (Up to 1000 Hops per Second)

ATR2406
Smart RF
AFHSS
Firmware

Application Note

Preliminary

Rev. 4858B–ISM–07/05

1. Description

This application note gives the user an overview of the usage of the advanced high-data-rate firmware with the reference design based on ATR2406 and ATmega88; Atmel's high-data-rate firmware is explained in detail. After reading this document, the user will be able to design software using the ATR2406 or to adjust the Atmel firmware to the specific needs of the application.

2. Introduction

First, this document describes the issues that have to be solved when designing an RF link. Second, the hardware is described and discussed with respect to advantages and disadvantages. Finally, the RF firmware is described in detail.

3. RF-specific Tasks and Problems

Transmitting data with the ATR2406 is not difficult, as data is simply shifted out bit-per-bit and then transmitted via the air. There are two important things that have to be considered, though. The first is that the data shifted into the ATR2406 has to be synchronous; this means each bit going into the ATR2406 must have a duration of multiples of 868.055 ns ($n \times 0.868055$ ns). The second point is that no gaps between the bytes of the data stream should occur.

Receiving the RF data is much more complex, as the receiving system has to be synchronized to the incoming RF data stream. This means the system must be able to find the first valid bit in the incoming RF data packet. That is one of the most important tasks regarding an RF link (when setting up the RF receiver, the RX Data pin monitors the noise on the channel to find the first valid bit of the packet).

4. Hardware Concept

The following sections give a short introduction into the chosen hardware concept and discuss the advantages and disadvantages of this concept.

4.1 USART Interface

The USART interface of the AVR microcontroller was chosen to serve the RF tasks, because the USART has the big advantage of an automatic hardware-supported synchronization on the incoming RF data stream. Therefore, this task does not impact the code space of the firmware or the CPU power required. The disadvantage of using the USART is that a start and a stop bit must be added to each byte of RF data.

4.1.1 Transmission of RF Data

During transmission, the RF data is simply written to the UART data register and then transferred to the ATR2406. The RF data stream must not be interrupted by gaps created through the data handling by the UART, which can easily occur if the UART is not double buffered in the transmit case, and must then be handled in software. Most microcontrollers contain a double-buffered UART, avoiding this problem in practice.

4.1.2 Reception of RF Data

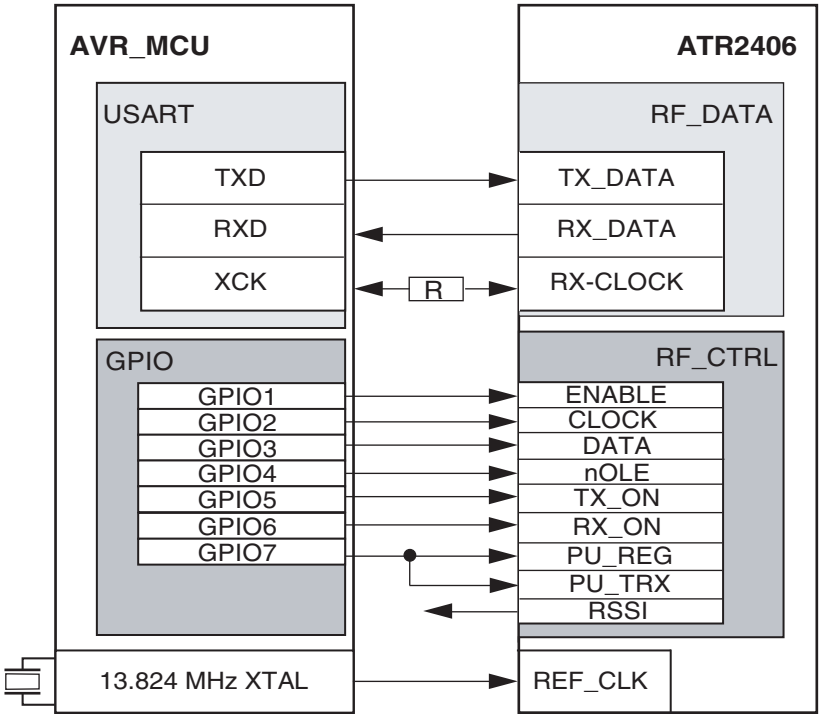
The reception of RF data via the UART is also not a complex task, as the synchronization on the incoming RF data is handled by hardware. This design of a receiving interface to the ATR2406 saves requiring additional code or CPU performance for synchronization. The only thing to do is to design a fast software, that is able to read the UART data register fast enough to prevent a possible buffer overrun of the peripheral. Most controllers use a double-buffered UART, avoiding this problem in practice.

4.1.3 Clock Rate Generation

The generation of the correct clock depends on the processor used, or on the possible prescalers of the UART baud-rate generator. If no prescaler is suitable, the user may try to use a synchronous UART instead of an asynchronous one.

4.2 Detailed Hardware Description

Figure 4-1. Hardware Concept

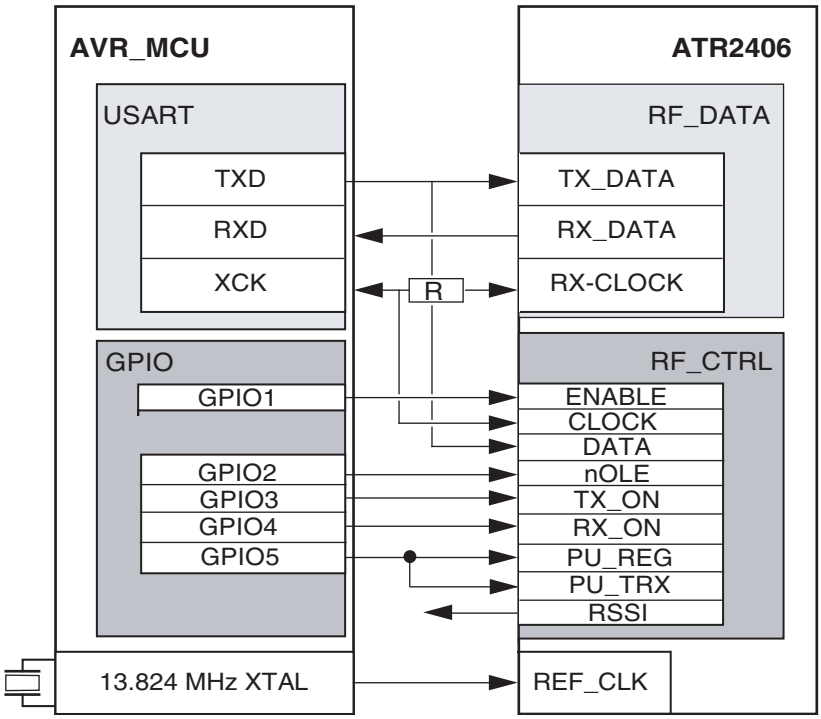


To exchange data via RF, the chips are connected via the USART interface of the AVR controller (Figure 4-1). To achieve the high data rate of 1.152 Mbps, the USART is used in synchronous mode, requiring the pin XCK. This pin is used for sampling the incoming data from the ATR2406. In transmit mode (as seen from the ATR2406) data is shifted out from the AVR TXD pin into the TX_DATA pin of the ATR2406. In this case the pin XCK is configured as output. As this would collide with the output pin CLK_REC of the ATR2406, a series resistor has to be put in this connection line. Also, the USART is configured in the MPC (Multi-processor Communication) Mode, a special mode that can be used to realize an automatic detection of a valid RF address while using only a minimum of CPU power. To transmit one byte of payload, three bits have to be added (start bit, stop bit and MPCM bit), giving each byte of data a transmit length of 11 bits and lasting (at a data rate of 1.152 Mbps) for 9.56 µs.

Configuring the ATR2406 is done via an SPI-like three wire interface (Clock, Data, Enable). Via this configuration, the ATR2406 is settled on the desired channel. As a special feature the ATR2406 has a programmable Gaussian filter, active only during the transmission of a data packet. The configuration word length is 16 bits for transmit and 25 bits for receive. The other control lines control the behavior of the RF chip.

The ATR2406 offers the possibility of evaluating an RSSI signal. This analog signal indicates the strength of the incoming RF signal and can, for example, be used to trigger the AVR internal analog comparator to enable the USART. The analog comparator should be used to obtain a quick response on the start of an RF data packet; however, if an exact value of the RSSI voltage is required, it is easy to pass the RSSI to the ADC via an AVR internal multiplexer. To evaluate the RSSI with this ADC needs much more time, of course, than doing the same with the analog comparator. The RF firmware does not use this feature because it is a system which is synchronized (that is, all the devices of the whole RF system are synchronized to the master). Using the ADC to evaluate the RSSI will decrease the sensitivity of the system. In systems that are not synchronized it is very helpful to have such an RSSI signal, as it can trigger the receiver to enable the reception of the RF packet.

Figure 4-2. Improved Hardware Concept



Another advantage of the whole RF system is that only one crystal is needed (at a frequency of 13.824 MHz). This crystal is connected to the AVR controller and the output is passed to the ISM transceiver chip. It is very important when starting the development of application software to have the correct settings of the AVR internal fuse bits (CKOPT fuse has to be set to have the full swing signal of the crystal). In order to reduce the I/Os needed to control the ATR2406, some of the lines are connected together as shown in Figure 4-2.

Figure 4-3. Reference Design Based on ATR2406 and ATmega88

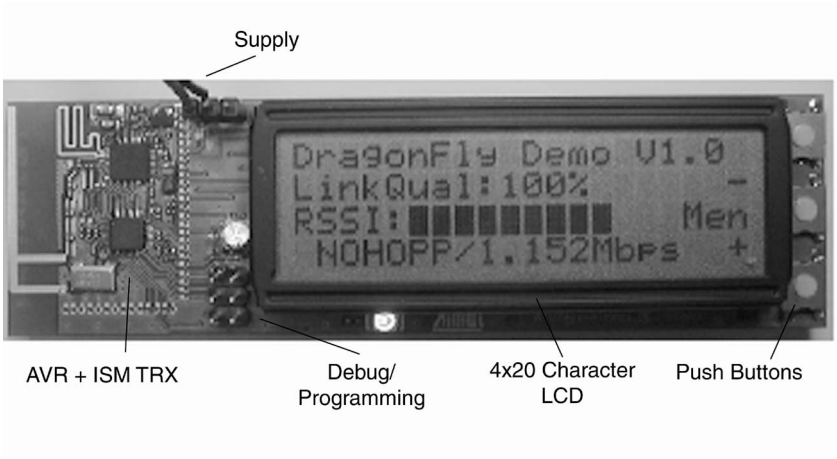
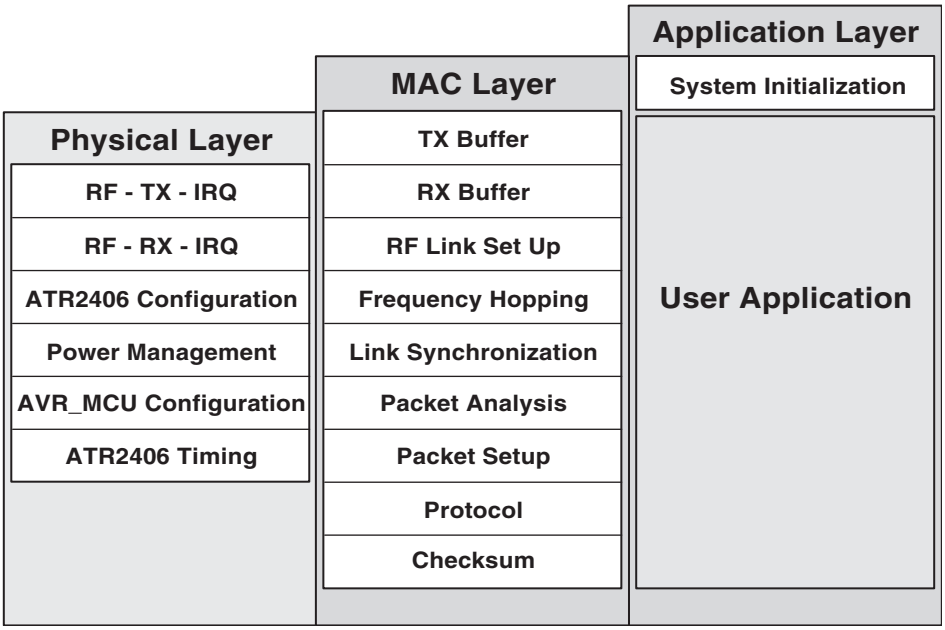


Figure 4-3 shows a picture of the reference design based on ATR2406 and ATmega88. Only eight I/O lines of the MCU (ATmega88) are needed to control the ATR2406. The three push buttons are used to navigate between the different available menus. To display information about the status of the RF link or the operation mode of the ATR2406, a 4 x 20 character LCD is used. Figure 4-3 shows the value of the RSSI and the quality of the RF link being monitored. The RF raw data rate and the hopping scheme currently in use are displayed on the last line. The two connectors on the PCB are for voltage supply to the reference design based on ATR2406 and ATmega88, and to debug or program the AVR microcontroller.

5. Software Concept

Figure 5-1. Software Concept

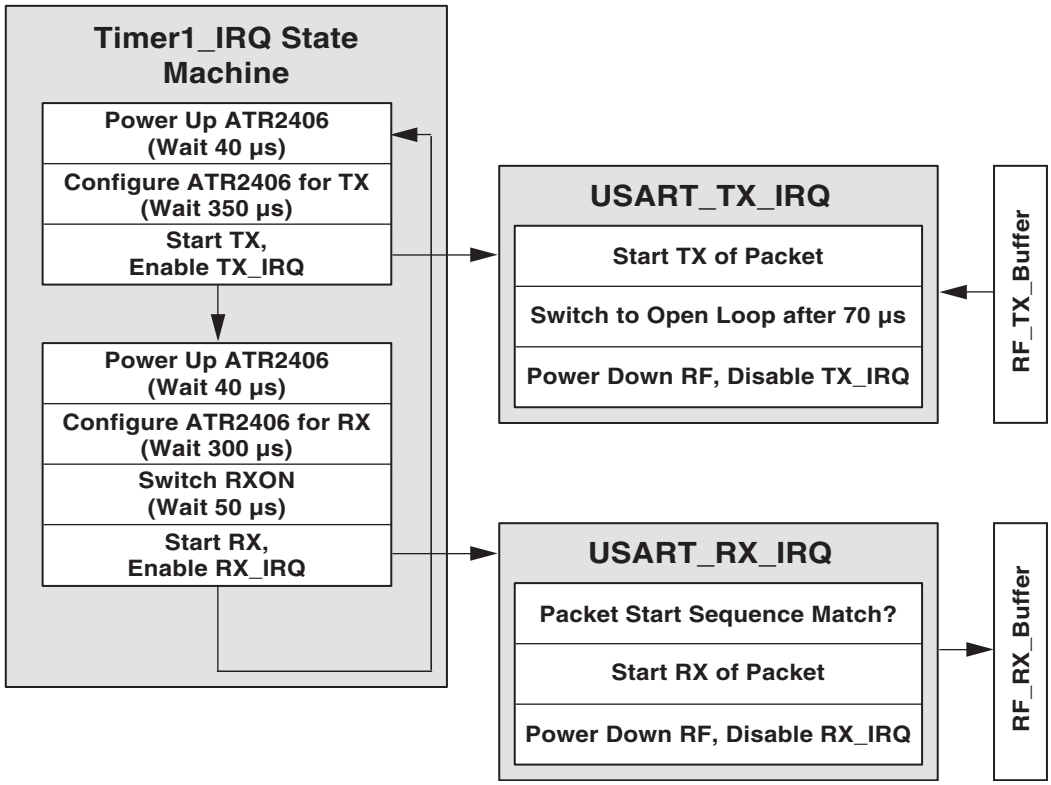


The RF firmware implements three main layers to set up the RF link and to exchange data via RF. The lowest layer (the physical layer) is responsible for the exchange of the RF data and the configuration of the ATR2406. The second layer (the MAC layer) handles the transmit and receive buffers, holding the RF data. The MAC layer also generates the timing needed to control the RF chip, and analyzes the received packet and the setup of a response packet.

The top layer of the RF firmware (the application layer) serves the special needs of the application and is completely independent of all the RF-related tasks. This independence makes it easy to implement the application layer, as the user does not need to worry about anything related to the RF task, unless the application uses other interrupt sources. (In this case the user has to make sure that the RF interrupts do not collide with the application interrupts.) The only thing the user needs to do, regarding the RF task, is to adjust the desired effective data rate (in Kilobytes per second) and hopping scheme in the RF firmware. This is done in the global definition file *defines.h*. To realize the RF specific tasks, some hardware resources of the AVR microcontroller are needed. These resources include two peripherals, the USART interface and one timer, as well as Flash and RAM. About 1000 instruction words of Flash are necessary, and 140 bytes of RAM, which is due mostly to the RX and TX buffers for the RF data. The rest of the MCU peripherals and memories can be freely used by the application-specific tasks.

6. RF Software Flow

Figure 6-1. Timer1 State Machine



The basic element of the RF software flow is the AVR internal timer, Timer1. This 16-bit timer is used to generate all the necessary timings to exchange data via RF. Figure 6-1 on page 6 draws a rough picture of the tasks controlled by Timer1. This timer controls the whole RF software flow and should not be changed by the user, as this could lead to a corrupted RF link. "RF Task Timing" on page 8 shows the basic calculations the RF timing is based on. The data exchange via the RF link is handled through the USART RX and TX interrupts. To achieve better system performance, these USART interrupts are written in assembler. Adapting the firmware to another processor of the AVR family is relatively easy – look for the location of the used USART interface (whether the USART is located in the normal IO space or in the extended IO space). In the original firmware, the USART is located in the extended IO space. Therefore, the instructions *IN* and *OUT* cannot be used, so they are replaced by *LDS* and *STS*. Changing the location of the USART from the extended I/O space to the normal requires that the user replace *STS* by the instruction *OUT* and *LDS* by *IN*. Nothing else in the two interrupt handlers needs to change. These three interrupts, USART_TX, USART_RX and Timer1, form the backbone of the complete RF task, so it is important that the user clearly understand them before writing custom code or making complex adjustments to the RF task. If, on the other hand, the user intends to use the RF firmware as is (that is, only write application-specific software or change the effective RF data rate), there is no need to completely understand these three interrupt handlers.

The following steps outline the software flow of the RF task.

1. The RF chip starts in the power down state and the next necessary step is the transmission of RF data.
2. The next execution of the Timer1 interrupt will power up the ATR2406.
3. After applying power to the chip, the user has to wait 40 µs before the configuration process can be started. So, the compare register of Timer1 is set so that the next Timer1 interrupt will trigger the system 40 µs later. The time between the interrupts can be used for the application software.
4. When the 40 µs have elapsed, the desired channel is calculated. This depends on the hopping scheme in use. The RF firmware offers two different hopping schemes (no hopping and random hopping). Random hopping is usually the best choice, as it is the most robust concerning blocked or disturbed channels. Once the channel is calculated, the ATR2406 is configured. The USART is also configured in the necessary manner (Synchronous mode, 1.152 Mbps, 1N9, MPC Mode). Note that the RF address of the whole system depends on the channel used, therefore the actual channel used has to be put in the high byte of the address word.
5. The interrupt routine is left and the Timer1 compare register is increased to trigger the system again in 350 µs. At this time, the transmission of the TX buffer (the data packet) is started by switching TXON to "1" and writing the first two bytes (double-buffered) of the packet to the USART data register. This also starts the USART TX interrupt. The compare register of Timer1 is increased so that the next interrupt triggers the system after the time necessary to achieve the desired effective RF data rate.
6. After having completely transmitted the packet, the ATR2406 is powered down and put into sleep mode. The transmission of the packet is completed.
7. The next execution of the Timer1 interrupt will start the reception of data via RF by powering up the RF chip. As in the transmit case, the chip needs 40 µs before it can be configured. The correct channel has to be calculated again and the calculated address has to be set in the variable *my_rf_adr_h*, to match the correct start sequence of the packet. After configuring the ATR2406, it is necessary to wait for 300 µs before the chip can be switched to receive mode.

8. After setting RXON, a wait of 50 µs is required before it is possible receive data. In this state of the interrupt handler, the USART is enabled in RX mode and the Timer1 compare register is increased to trigger the system again when the maximum possible time for a correctly received packet has elapsed.
9. The next step is to check if data has been properly received. In case of correct reception, the system has to be resynchronized to the RF master (if it is in Slave Mode). Also, the response packet is calculated and put into the TX buffer.

To get an exact understanding of the Timer1 interrupt service routine, it is a good idea to have a look into the code. There are many comments in the code to help with the necessary understanding.

6.1 RF Task Timing

```
#define DATA_RATE 24000 //define datarate in effective bps
T_CLICK            1000 / 1728
T_SLOT             ((1000 / (DATA_RATE / 48)) * 1728)  //slot time in us
T_TX2RX            T_SLOT - T_PWR_UP_TX - T_LOOP_TX
T_RX2TX            T_SLOT - T_PWR_UP_RX - T_LOOP_RX - T_RX_ON - T_MAX_PAC_RX
T_START_SYNC       T_SLOT - 906 - 31 //[31 == calc time!!(17.94us)]
T_SYNC             T_SLOT - 906
POLL_LOOPS         ((T_SLOT + 1728) / 1728)
T_PWR_UP_TX        70       //70 == 40us(40.51)
T_LOOP_TX          605      //605 == 350us(350.11)
T_PWR_UP_RX        70       //70 == 40us(40.51)
T_LOOP_RX          518      //518 == 300us(299.77)
T_RX_ON            87       //87 == 50us(50.35)
T_MAX_PAC_RX       1124     //1124 == 650us(650.46)
```

7. USART RX IRQ

The USART RX IRQ interrupt handler for the reception of the RF data is written in assembler.

1. Push the used registers onto the stack.
2. Read the contents of the USART data register.
3. Check for the correct start sequence of the RF data packet.
4. If the sequence is correct
   a. All the received bytes are stored in the RX buffer until a valid end of frame is received.
   b. A timestamp is taken to keep the systems in synchronization.
   c. The checking of checksum is also performed inside this interrupt.
   d. Before the interrupt routine is left, the pushed registers are restored.

To completely understand all the things done in the interrupt, it is necessary to be familiar with the instruction set of the AVR microcontrollers.

8. USART DRE IRQ

The USART DRE IRQ interrupt handler is used for the transmission of the RF data.

1. The used registers must be put onto the stack.
2. The pointer on the correct byte of the TX buffer is set up.
3. The switching between open and closed loop are handled.
4. Setting and resetting of the multi processor communication bit in the UCSR1A register (the ninth bit of data) is done.
5. After putting the correct byte onto the USART data register the registers are restored and the interrupt routine is left.

9. Achieved RF Performance

The main focus for the development of the RF firmware is system performance, to give the application as much of the CPU power as possible to deal with the application specific tasks. For a better understanding of the numbers given in the features list of the RF firmware, see this example.

Assumption: Effective RF data rate :192 kbps (24 KBytes of RF data per second)

CPU power needed for RX: 33%

CPU power needed for TX: 30%

RF Slot time: 2000 µs

Resulting in every 4 ms, 66 Bytes of RF data have to transmitted, and every 4 ms, 58 Bytes of RF data have to be received. Duration of one byte is 11 × (1 / 1.152 MHz) = 9.55 µs.

Time necessary for the TX purpose: 64 × 9.55 µs × 0.3 = 183.5 µs

Time necessary for the RX purpose: 56 × 9.55 µs × 0.33 = 176.5 µs

So, 360 µs are needed to deal with the RF-related tasks in a complete time of 4000 µs. Therefore, the average necessary CPU power to achieve an effective average data rate of 192 Kbps is 360 µs/4000 µs = 9.0%. Now, it is also necessary to include the CPU power needed to handle the Timer1 interrupt. Roughly, for the whole RF task, 10% of CPU power is needed. That means the application software has up to 90% of CPU power to serve its special needs. This would be the same as clocking the CPU at 12.5 MHz, if there were no need of serving anything related to the RF task.

10. Module Overview

The following section gives an overview on the released modules and functions inside the RF firmware. There are 6 modules and 8 header files included in the RF firmware.

10.1 VARIS.C

Module *varis.c* includes definitions for all necessary global variables.

volatile uc_8 rf_tx_buffer[64]@0x0C00; Buffer holding the data intended to be transmitted

volatile uc_8 rf_rx_buffer[64]@0x0C40; Buffer holding the received data

volatile uc_8 rf_counter; Counter for transmitting and receiving RF data

volatile uc_8 timer_state; Control variable for the RF-task state machine

volatile uc_8 rf_rx_status; Status of the RF receiver interrupt handler

volatile uc_8 checksumm; Checksum of the received RF packet

volatile ui_16 timestamp@0x0BFE; Timestamp used to keep the RF systems synchronized

volatile uc_8 timestamp_lb@0x0BFE; Timestamp for accessing via in-line assembler

volatile uc_8 timestamp_hb@0x0BFF; Timestamp for accessing via in-line assembler

uc_8 conf2406; Variable holding the configuration of the RF chip
(GF settings, output power, etc.)

uc_8 volatile rf_status; Status of the RF task

uc_8 random; Variable holding a random number

uc_8 channel; Actual channel used by the RF link

uc_8 channel_width; Variable holding the jumping width for the frequency hopping

uc_8 my_rf_adr_l; RF low address of the system

uc_8 my_rf_adr_h; RF high address of the system

10.2 INIT.C

The task of module *init.c* is to initialize the hardware of the AVR microcontroller and to assign the default values to the global variables, if necessary. The module includes two functions, the function *init_avr* is used for the initialisation of the AVR peripherals and the function *init_varis* has to set the default values of the global variables.

10.3 MISC.C

Module *misc.c* includes three functions, *get_random*, *copy_buffer* and *wait_n_10us*.

- The function *get_random* is used to initialize the global random variable. This is done by sampling the RX data pin of the RF chip (this pin monitors noise while in receive mode and no signal is present, therefore it is an easy and convenient way to generate a random value).
- To provide a powerful method for copying two buffers, the routine *copy_buffer* was implemented in in-line assembler so that it would require only a few instruction cycles.
- The third function, *wait_n_10us,* provides the possibility of generating a configurable wait time (in multiples of 10 µs).

10.4 RF_CTRL.C

File *rf_ctrl.c* is the most important file, handling the basics of the complete RF task. To do this there are 4 interrupt functions included in this module to control the behavior of the ATR2406. Three interrupt handlers are needed for the RF data. One for receiving data (*USART1_RXC_IRQ*) and two for the transmission of RF data (*USART1_TXC_IRQ* and *USART1_DRE_IRQ*). Each of the three interrupt handlers is written in assembly language, to improve the performance of the whole system. The last function is the Timer1 interrupt, used to control the RF flow via a state machine.

The other 6 functions are needed to configure and control the ATR2406 and the RF task.

- *start_timer1* sets up the 16-bit wideTimer1 as required
- *setup_tmr1_scanning* sets Timer1 up for the necessary channel scanning while setting up the RF link
- *reset_rf* is used to bring all the RF related variables to their default values
- *pwr_up_atr2406* applies power to the RF chip
- *pwr_dwn_atr2406* powers down the RF chip and puts it in sleep mode
- *init_atr2406* is needed to configure the ATR2406 for the desired channel, mode and output power

10.5 PROT.C

The purpose of *prot.c* is to analyze the received data and set up the packets. It is also used to establish the connection between the systems included in the RF link.

10.6 APPL.C

Module *appl.c* includes *main*, where the user can place the application software. There are no restrictions for the main function, as the complete RF task is independent from all the other tasks. It is important to consider the usage of other interrupts. The RF interrupt can trigger the system every 9.5 µs, so other interrupts must not take longer than 9.5 µs. If any other interrupt could use more system time than 9.5 µs, the global interrupt flag has to be set in the interrupt service routine to re-enable the RF interrupts. As long as this restriction is kept in mind, there should be no problem developing application software that uses the RF firmware to exchange data via RF.

10.7 DEFINES.H

The file *defines.h* is very important, as the user can use it to adjust the RF task to the specific needs. It is possible to select the desired frequency hopping scheme out of three predefined ones.

//#define ADAPTIVE – FHSS System performs adaptive hopping through the channels

The user chooses a hopping scheme by uncommenting it. The most powerful is the random hopping scheme. Another possible adjustment is the selection of the desired effective average data rate. By manipulating the line *#define DATA_RATE 24000//define the needed data rate in effective bytes per second* the user can choose any data rate desired. The maximum data rate is 48 Kilobytes per second (if the RF chip is not powered down during operation) or 24 Kilobytes per second (if the ATR2406 is powered down during the RF cycles).

11. Necessary AVR Resources

The RF firmware requires some RAM and Flash memory, as well as some peripheral support. The necessary Flash space for a minimal implementation of the RF firmware is about 1120 words (2240 Bytes), about 55% of the Flash size of an ATmega48, the smallest available AVRMega processor. This Flash space includes the required “C Start up”, without which the size of the code is reduced to only 1000 words (2000 Bytes), meaning that the RF firmware itself is only about 49% of the space available in an ATmega48. To achieve high system performance the RF task is supported by two peripherals, the USART and Timer1. Timer1 is used to generate all the needed RF timings and the USART is used to exchange RF data between the AVR and the ATR2406.

There are eight I/O lines needed to control the ATR2406. To achieve this low I/O usage some of the signals have to be connected together, or eleven I/O lines would be needed.

Table 11-1. AVR Resources

RF Task	Application Task
16-bit Timer (1 IRQ source)	Two 8-bit timer
No EEPROM	Complete EEPROM
2 KByte of Flash	2 KB/6 KB/14 KB of Flash
154 Bytes of SRAM (depends mainly on RF buffer length)	454 Bytes/970 Bytes of SRAM
USART interface (3 IRQ sources)	SPI interface, all ADC (8 channels)
8 GPIOs	13 GPIOs
-	TWI interface, all PWM (6 channels)
-	Watchdog timer

12. Using the Reference Design Based on ATR2406 and ATmega88

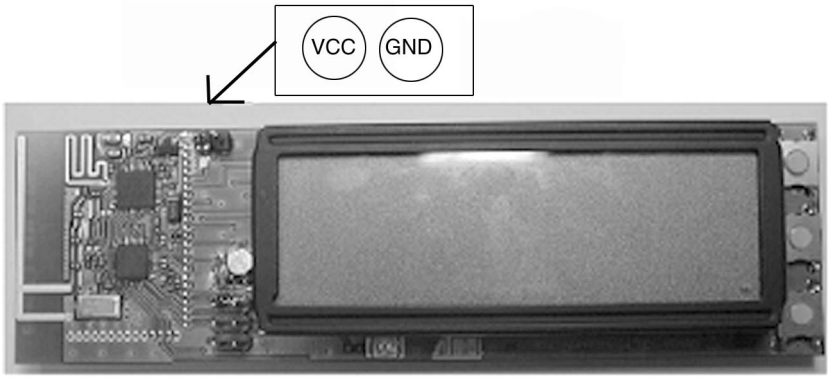
12.1 Setting up the Reference Design Board

The reference design board is shipped flashed with Atmel's advanced smart RF firmware, but you should check the web site for a newer version of the firmware and flash it to the AVR microcontroller, if available.

12.1.1 Powering Up the Reference Design Board

To power up the reference design board, connect 3.8V – 5.0V to the supply connector at the top of the PCB. Figure 12-1 shows the correct way to connect the supply voltage (either three AAA-batteries or a power supply) to the reference design board. After the reference design board is powered up, the different menus offered by the application software can be navigated.

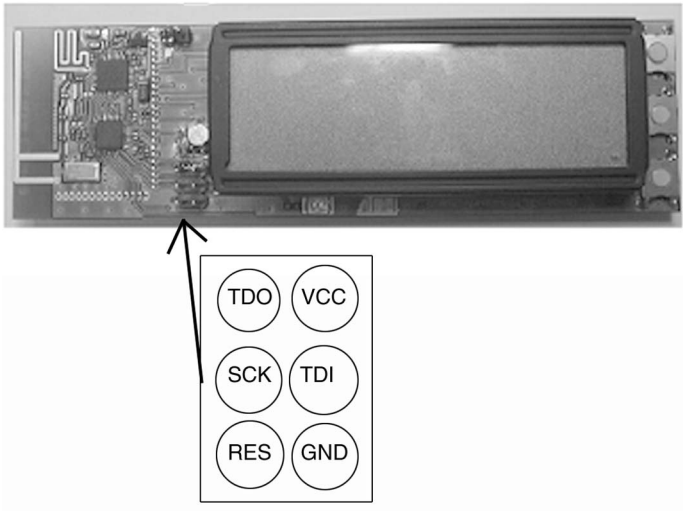
Figure 12-1. Connecting the Supply Voltage



12.1.2 Debugging or Programming the Reference Design Board

The compiler used to develop the advanced RF firmware is the CodeVisionAVR compiler. To program and debug the code the AVRStudio and JTAG MK 2 are used. These two tools are from Atmel.

Figure 12-2. Connecting the JTAG MK 2



AVRStudio can be downloaded from the Atmel web site at no cost. The 2 x 3 connector at the bottom of the reference design board is used to flash and debug the AVR microcontroller. The connection between the reference design board and AVRStudio is realized by the JTAG MK 2 connected to the 2 x 3 connector, using a the squid cable included in the JTAG MK 2 (Figure 12-2 on page 13).

12.1.3 Working With the Reference Design Board

If the reference design board is correctly powered up and an RF link is established, the display should appear as in Figure 12-3. The version of the RF firmware is displayed on the top line. The quality of the RF link is shown on the second line, with the number representing the packet error rate (packets consist of 726 bits). The third line represents the strength of the RF signal while receiving. The last line shows the raw RF data rate and the hopping scheme in use. The three push buttons are used to navigate through the menus of the RF firmware. The top and bottom buttons are used to switch between the different menus, and the middle button is used to change the meanings of the two other buttons. When *Men* (short for *Menu*) appears next to the buttons, the top and bottom button are used to navigate through the menus. Application of the buttons and the functionality of the RF firmware will be discussed in the following sections where the menus are described.

Figure 12-3. Reference Design Board Display

12.1.4 System Start Up

Here are the steps at system startup.

1. The AVR ATmega88 is initialized.
2. The global variables are set to default values and the LCD is initialized.
3. The RF task is initialized.

When this is done, the display appears as shown in Figure 12-4 on page 15. As no RF link is set up yet, the RSSI value (*RSSI*) is low and the link quality (*LinkQual*) is zero. It is very important when setting up an RF link to choose the same channel and hopping parameters for both participants, or no RF link will be established.

Figure 12-4. System After Power On

12.1.5 Link Set Up

To set up an RF link, each reference design board scans the channel to determine if another reference design board is already present, and if one is found, goes on to check if it is possible to connect to the other system. It is only possible to connect two reference design boards together. If two reference design boards are already connected, they do not answer a login request from a third reference design board. This is an important point regarding the collocation of many different reference design boards.

The first reference design board to be switched on will detect that no other possible partner is present to form an RF link. The reference design board will then start sending out packets announcing that it is available to form a link. If another reference design board receives such a packet during the initialization of the RF task, an RF link is set up between the two. The result will be a display as shown in Figure 12-3 on page 14. Pressing the bottom button of the PCB will select the RF-Info menu.

12.1.6 RF-Info

Figure 12-5. RF-Info Menu

The RF-Info menu (Figure 12-5) is identified by the words *RF-Info* on the top line. *LinkQual* represents the quality of the RF link. In Figure 12-5 no RF link is set up yet; therefore the link quality is zero. The RF address of the system is shown following *Random-Adr*. Each system gets its own random address, in order to identify several systems in the same range. The last line shows the value of the hopping width, also random. The hopping width value is only valid if frequency hopping is enabled (NEW_CHANNEL = OLD_CHANNEL + HOPPING_WIDTH).

12.1.7 ATR2406-Settings

ATR2406-Settings (Figure 12-6) shows the current settings of the RF transceiver chip. *GaussFilter* displays the setting of the ATR2406-internal Gaussian filter. *External PA* shows the setting of the PA bits. For the meaning of these values, see the ATR2406 datasheet.

Figure 12-6. ATR2406 Settings Menu

12.1.8 RF-Test-TX-nomod

RF-Test-TX-nomod (Figure 12-7) is used for test purposes and sets up the ATR2406 in transmit mode on the selected channel. The carrier is not modulated. This menu exists to check if the carrier frequencies of the corresponding channels are correct. Pressing the middle button switches the menu into the channel selection mode (Figure 12-8 on page 17). In this mode it is possible to change the ATR2406 to the desired channel. Pressing the middle button again switches back to menu selection mode.

Figure 12-7. RF-Test-TX-nomod Menu Selection Menu

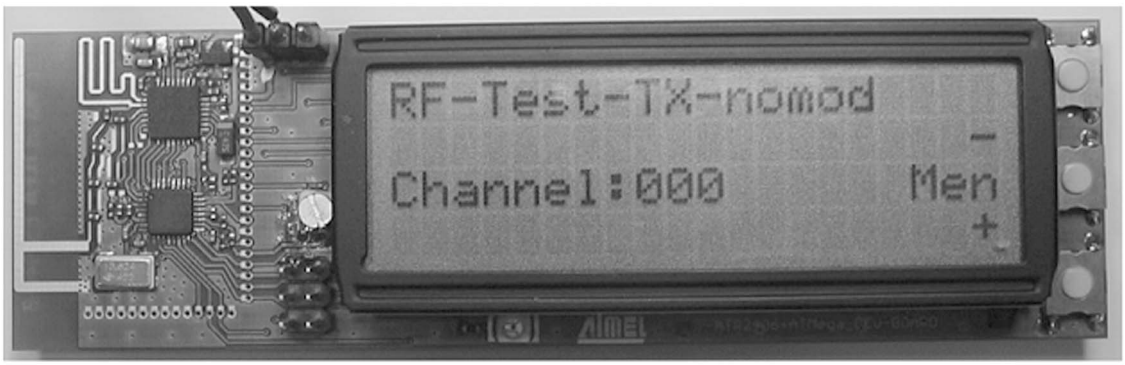
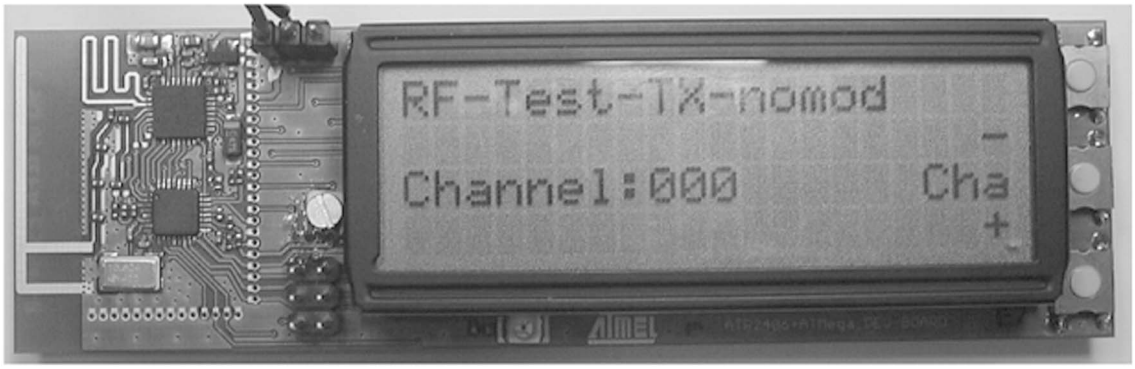


Figure 12-8. RF-Test-TX-nomod Channel Selection Menu



12.1.9 RF-Test-TX-mod

RF-Test-TX-mod (Figure 12-9) is used for test purposes and sets the ATR2406 up in transmit mode on the selected channel. In this menu the carrier is modulated in closed-loop mode. Pressing the middle button switches between several menus to set up this test mode in the desired way. Figure 12-9 shows the menu in the menu selection mode. Pressing the middle button one time switches the menu to the channel selection mode (Figure 12-10 on page 18). In this menu the user can select the desired channel to make RF measurements. Pressing the middle button once more switches the menu to the pattern selection menu (Figure 12-11 on page 18). Here the user can choose the pattern desired to transmit in this test mode. The last menu available in this context is the Gaussian filter selection menu (Figure 12-12 on page 18); in this menu the user can adjust the Gaussian filter to the needed value. All cases of the RF-Test-TX-mod menu set the ATR2406 up in closed-loop mode and can be used to test the correct operation of the transceiver chip.

Figure 12-9. RF-Test-TX-mod Menu Selection Menu

Figure 12-10. RF-Test-TX-mod Channel Selection Menu

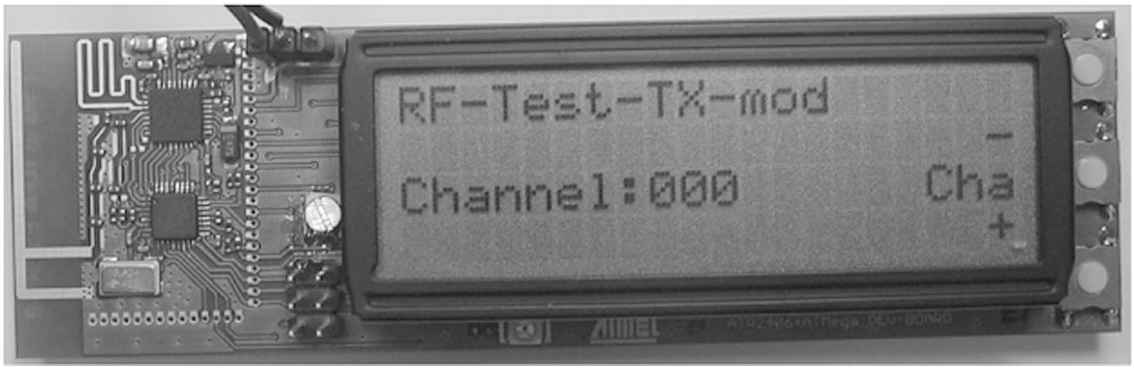


Figure 12-11. RF-Test-TX-mod Pattern Selection Menu

Figure 12-12. RF-Test-TX-mod Gaussian Filter Selection Menu

12.1.10 RF-Test-TX-burst

The RF-Test-TX-burst menu is used for test purposes and sets up the ATR2406 in transmit mode. The ATR2406 sends out data bursts while in this test menu. The modulation in this menu is open loop and is based on the same timings as when in normal operation mode. In this test mode you can select between different menus to put the ATR2406 in the desired state. Figure 12-13 shows the basic menu for this test case. By pressing the middle button the user can select between a hopping and a non-hopping system. This selection is also active when in normal operation mode. A zero indicated that frequency hopping is disabled, and a one indicates that hopping is enabled (see Figure 12-14). Additionally, you can adjust the Gaussian filter setting (Figure 12-15 on page 20) or the channel (Figure 12-16 on page 20). The channel selection affects the system only if frequency hopping is disabled.

Figure 12-13. RF-Test-TX-burst Menu Selection Menu

Figure 12-14. RF-Test-TX-burst Hopping Selection Menu

Figure 12-15. RF-Test-TX-burst Gaussian Filter Selection Menu

Figure 12-16. RF-Test-TX-burst Channel Selection Menu

12.1.11 RF-Test-RX

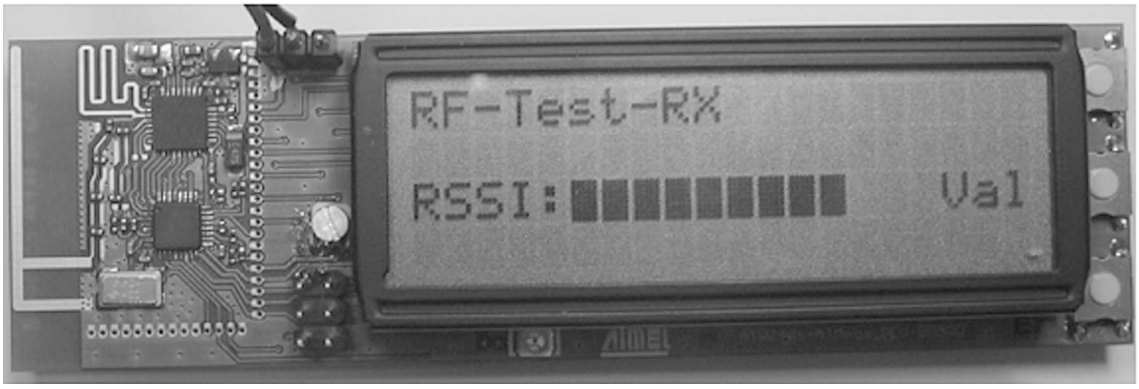
The RF-Test-RX menu (Figure 12-17) is used for test purposes and sets up the ATR2406 in receive mode. The received RF data can be measured on the RX Data pin of the ATR2406. It is also possible to select the different available channels (Figure 12-18 on page 21) and monitor the strength of the received signal (Figure 12-19 and Figure 12-20 on page 21). Compare Figures 12-19 and 12-20, showing the menu with no RF signal present and with RF signal present, respectively.

Figure 12-17. RF-Test-RX Menu Selection Menu

Figure 12-18. RF-Test-RX Channel Selection Menu

Figure 12-19. RF-Test-RX Value Selection Menu, No Signal Present

Figure 12-20. RF-Test-RX Value Selection Menu, Signal Present



13. Frequency Hopping

The advanced RF firmware includes two hopping modes: no hopping (always the same channel) and a random-based hopping scheme. The advantage of a frequency hopping system is that the designed RF system is much more robust with respect to interference and blocked or disturbed channels. Another advantage is the collocation of many different systems in the same range. To get a working link it is useful to separate the different participants of the whole RF system by channel and time. This makes it necessary to synchronize all RF systems of one link by channel and time. Frequency hopping also increases the system performance, as the RF task only has to be activated during specific time slots, lowering power consumption to a minimum. A disadvantage of such systems is that if the synchronization of time and channel is lost, then the RF link is destroyed and has to be set up again. Table 13-1 gives an overview of the advantages and disadvantages of a frequency hopping spread spectrum (FHSS) system.

Table 13-1. Frequency Hopping Advantages and Disadvantages

Advantages	Disadvantages
Immunity against blocking	More complex to design
Immunity against interference	RF systems have to be initially synchronized
Collocation of many systems	Systems have to continuously resynchronize
Power saving	-
Lowers CPU load	-

The ATR2406 offers 95 different channels in the 2.4 GHz ISM band, allowing a variety of different hopping schemes. The hopping scheme implemented in the advanced RF firmware is random-based; the basis of this hopping algorithm is a true random value, calculated by sampling the receive data pin of the ATR2406. This pin is monitoring noise when no RF signal is present on the actual channel of the ATR2406 in use, and this monitored noise is random. This random number is then used to calculate the next valid channel of the RF link. If the RF link is set up, the master transmits this random number to the slave in order to get the same hopping scheme, otherwise no link could be set up. As the system offers 95 different channels, the random number can have a range from 1 to 95. Therefore, the probability that two different systems have the same random-based hopping scheme is now 1 in 95, or about 1.05%. If the user needs a lower probability than 1.05%, then the RF system could be based, for example, on two or more of these random numbers. For a better understanding, see the following example.

Assumption: Random hopping scheme is based on two random values (start address and hopping width) with a range from 1 to 89 (because 89 is the largest prime number smaller than 95).

Probability of two systems having the same hopping scheme: $(1/89)^2 = 1/7921$

To improve the system even more, the firmware takes a further random number. This random number is placed in the address field of the RF packet. The RF firmware takes only two random numbers, one for calculating the hopping scheme and one is used as a random address for the system. If the user wants to collocate many systems in the same range, then the system should be made more random or adaptive. Of course, we have only been discussing the separation by channel, but the different systems are also separated by time. This means that even if two different systems use the same channel, then a collision of data is very unlikely, as they are also separated by time.

To design a suitable RF link regarding the hopping scheme in use, some calculations have to be done to determine the probability that two systems the same random-based hopping algorithm. It is hard to say how small this number should be, but the smaller it is, the better.

13.1 Example

Problem 1: Each system must get a unique random number as a base for the hopping width

Given a random range of 89 and a desired collocation of 4 independent RF systems

1. Calculation of the good possibilities: $89 \times 88 \times 87 \times 86 = 58599024$
2. Calculation of all the possibilities: $89 \times 89 \times 89 \times 89 = 62742241$
3. Probability of successful coexistence: 93.4%

Coexistence means that the different systems get a different random value. As this can not be guaranteed, the RF firmware has to be designed in such a way that a 100% probability for such a successful coexistence exists. So this is the first problem that has to be solved.

Problem 2: Cyclical blocking of one channel is possible

A second problem is caused by the fact that the available numbers of channels is a prime number. This fact has a very important advantage. The calculation of the random value as base for the hopping width can be done without any restrictions. The prime number of the available number of channels guarantees, that every channel (89) is used by the system regardless of the random number used as hopping width. But a problem that is caused by this fact is that all the possible hopping schemes are cyclical ones with all the same cycle duration of 89 hops. This point is the second problem to solve. The reason for this is the following situation. Imagine, that two independent systems, each with a different hopping width, are present in the range and they are in cycle N on the same channel. Then this packet will be lost, as will every other packet in cycle number $N + (89 \times C)$ (where C is the range from zero to infinity).

13.2 Solving the problems

13.2.1 Problem 1

To make sure that each system in the range gets a unique hopping width, the RF task gets reset when the RF quality falls below 10%. If the RF quality falls under this border, then there is a high probability that another system in the range has the same hopping width and will always use the same channel. Before getting reset, the RF task stores whether it was slave or master of the system. Then a new random number is fetched through the noise on the RX data pin of the ATR2406 by the master. Then the master and the slave try to reconnect. This ensures that, after a given time, all the systems in the range will have a different and unique random hopping width.

13.2.2 Problem 2

There are two ways of dealing with this kind of problem. The first one is harder to implement and uses fewer system resources. The second one is a bit easier to implement, but needs a lot more system resources than the first proposal. So, regarding required system resources and delivered performance the first described way is the better one.

13.2.2.1 First Proposal

The RF firmware implements a FIFO in each device. This FIFO is used to store the channels where a successful packet reception was impossible. The length of this FIFO can be defined by the user. Every channel contained by that FIFO is left out of the hopping scheme, making this an adaptive hopping scheme, where the most disturbed or blocked channels are deleted. The only thing to do is to keep the FIFOs of the master and the slave synchronized. This issue is performed in the following way and explained in a small example. Suppose a system formed by device A and device B, both forming the RF link with 11 channels in use (Table 13-2). This table shows the hopping scheme based on the random hopping width of one and without the ability of an adaptive frequency hopping.

Table 13-2. Nonadaptive FHSS

Device_A		Device_B	
RX_A	**TX_A**	**RX_B**	**TX_B**
0	####	####	0
####	1	1	####
2	####	####	2
####	3	3	####
4	####	####	4
####	5	5	####
6	####	####	6
####	7	7	####
8	####	####	8
####	9	9	####
10	####	####	10

Table 13-3. Adaptive FHSS

Device_A		Device_B			
RX_A	TX_A	RX_B	TX_B	FIFO_A	FIFO_B
2	####	####	2	0	0
####	3	3	####	1	1
4	####	####	4		
####	5	5	####		
6	####	####	6		
####	7	7	####		
8	####	####	8		
####	9	9	####		
10	####	####	10		

The same system with an adaptive frequency hopping scheme looks like described in Table 13-3. This table shows the hopping scheme and the content of the FIFO immediately after the successful connection of Device_A and Device_B. To keep the hopping of both devices up-to-date, the channels contained per default in the FIFO are left out of the hopping algorithm. In this example the depth of the FIFO is assumed to be 2 bytes. Therefore the hopping uses only 9 different channels and not 11 as shown in Table 13-2. This way of implementing this cyclical blocking avoidance is very simple but effective. Table 13-3 on page 25 shows that the channels contained by the FIFO are not in the hopping scheme, so it is no problem to mask out any other of the channels that are supposed to be blocked by any other device. The important thing in this case is that the number of used channels in the hopping scheme must always be the same no matter if there is a blocked channel or not. As long as no packet fails, the content of the FIFO is the same and the hopping scheme is also not changed. Now it is assumed that channel three is blocked or disturbed. Therefore, the packet transmitted from Device_A and received by Device_B will fail. In the first step only Device_B will know that the exchange of data was not successful and therefore Device_B puts channel three into the FIFO (see Table 13-4).

Table 13-4. FHSS Packet Error

Device_A		Device_B			
RX_A	TX_A	RX_B	TX_B	FIFO_A	FIFO_B
2	####	####	2	0	1
####	3	3	####	1	3
4	####	####	4	Packet Error	
####	5	5	####		
6	####	####	6		
####	7	7	####		
8	####	####	8		
####	9	9	####		
10	####	####	10		

Furthermore, the next packet transmitted from Device_B to Device_A contains information about the possibly disturbed channel three. After the successful packet exchange on channel four both devices have resynchronized their FIFO contents (see Table 13-5). Even if the packets fail on channel four it is no problem to get the FIFOs resynchronized. It is allowed to remove only one channel from the hopping scheme per hopping cycle (FIFO depth divided by two is maximum number of removed channels per hopping cycle). Even if all the following packets of the hopping cycle fail, the FIFOs are resynchronized. Now the hopping scheme looks like Table 13-6 on page 26. Channel zero is now used in the hopping scheme, as it was kicked out of the FIFO, and channels three and one are left out of the hopping scheme. The adaptive system is tolerant to a highly-blocked band as it is not necessary that the FIFOs be updated by the reception of a correct packet. To achieve this robustness the FIFO depth should be twice as high as the maximum number of removed channels per hopping cycle. A hopping cycle is when the device has exchanged data on each possible channel.

Table 13-5. FHSS FIFO Update

Device_A		Device_B			
RX_A	**TX_A**	**RX_B**	**TX_B**	**FIFO_A**	**FIFO_B**
2	####	####	2	1	1
####	3	3	####	3	3
4	####	####	4	Packet Error	
####	5	5	####	FIFO_A update	
6	####	####	6		
####	7	7	####		
8	####	####	8		
####	9	9	####		
10	####	####	10		

Table 13-6. FHSS Scheme Update

Device_A		Device_B			
RX_A	**TX_A**	**RX_B**	**TX_B**	**FIFO_A**	**FIFO_B**
0	####	####	0	1	1
####	2	2	####	3	3
4	####	####	4		
####	5	5	####		
6	####	####	6		
####	7	7	####		
8	####	####	8		
####	9	9	####		
10	####	####	10		
####	0	0	####		

13.2.2.2 Second Proposal:

The second proposal is a nonadaptive frequency hopping with a high random factor. The disadvantage of this hopping scheme is that many more system resources are needed than in the first proposal. The basis of this hopping scheme is an array of 93 random values (range 0-95). These random values represent the channel that has to be used. This array needs 93 bytes of memory. To spare the valuable SRAM resources of the AVR ATmega88 this array could be placed into the EEPROM. A 16-bit linear feedback shift register (LFSR) has to be implemented in the RF firmware. A Perl script is available to simulate the results achieved with the algorithm used for the LFSR. The LFSR is initialized with a 16-bit random value. The algorithm of this LFSR is chosen in such a way that a cycle width of 93 is achieved. As 16 bits have 65536 possible states there exist 700 different sets of these 93 pseudorandom sequences. At system start-up, the channel selection array is filled with 93 random values and the LFSR is initialized with a random value. While hopping through the band, the LFSR is shifted one time if a hop is needed. The result of the LFSR is divided modulo 93 and the result is used as index for the channel selection array. The random value of the channel selection array is now used as a channel to exchange the RF data.

13.2.2.3 Implementation Example

The example implemented in the reference design board RF firmware is based on the first proposal, described at the beginning of this chapter. To deal with this task of an adaptive FHSS there are several additional bytes of SRAM necessary (Table 13-7 gives an overview of the needed SRAM variables and arrays).

Table 13-7. Variables Needed for the Adaptive FHSS

Name	Size	Type	Function
uc_8 ch_blocked[8]	8 bytes	FIFO	Basic evaluation of the channels in use
uc_8 ch_blocked_ptr	1 byte	Pointer	Pointer for the FIFO access
uc_8 hopp_fifo_tmp[4]	4 bytes	FIFO	Temporary FIFO for the hopping scheme in use
uc_8 hopp_fifo_tmp_ptr	1 byte	Pointer	Pointer for the FIFO access
uc_8 hopp_fifo_use[4]	4 bytes	FIFO	FIFO holding the forbidden channels of the hopping scheme
uc_8 hopp_fifo_use_ptr	1 byte	Pointer	Pointer for the FIFO access
uc_8 hopp_count	1 byte	Counter	Cycle counter to control the FIFOs

The array *ch_blocked[8]* is used as a FIFO to store the disturbed or blocked channels. This task only has to be done by the master of the RF link. After each incorrect or failed RF packet, the master puts the value of the channel into the FIFO. If a certain channel resides more than two times in this FIFO, the channel is determined to be very noisy or blocked. Therefore this channel is copied to the *hopp_fifo_tmp[4]* FIFO. This FIFO is able to hold four values of blocked or disturbed channels. These actions only have to be performed by the master and not by the slave, because the contents of *hopp_fifo_tmp[4]* is transmitted from the master to the slave from time to time. Additionally, both master and slave contain the variable *hopp_count*. This variable is synchronized between both systems and is used to generate a trigger for the adaptation of the hopping scheme.

If the variable *hopp_count* reaches a certain value, the contents of the *hopp_fifo_tmp* is copied to *hopp_fifo_use*. It is very important that this is done in both systems (master and slave) at the same time, because otherwise the RF link could be destroyed. Now the hopping scheme is adapted to the new situation and all the blocked or disturbed channels are not used anymore in the FHSS. The depth of the FIFO can be adjusted to the special needs of the application. To be even more robust, the band is divided into some subchannels. Each subchannel consists of several ATR2406 channels. Table 13-8 on page 28 gives an overview of the channels and subchannels. The advantage of this is that if a channel in the subchannel is blocked, all the other channels included in the subchannel are also assumed to be disturbed.

Table 13-8. FHSS Table

Channel	Subchannel	Symbol	Channel	Subchannel	Symbol	Channel	Subchannel	Symbol
0	0	0	30	7	7	60	15	?
1	0	0	31	7	7	61	15	?
2	0	0	32	8	8	62	15	?
3	0	0	33	8	8	63	15	?
4	1	1	34	8	8	64	16	@
5	1	1	35	8	8	65	16	@
6	1	1	36	9	9	66	16	@
7	1	1	37	9	9	67	16	@
8	2	2	38	9	9	68	17	A
9	2	2	39	9	9	69	17	A
10	2	2	40	10	:	70	17	A
11	2	2	41	10	:	71	17	A
12	3	3	42	10	:	72	18	B
13	3	3	43	10	:	73	18	B
14	3	3	44	11	;	74	18	B
15	3	3	45	11	;	75	18	B
16	4	4	46	11	;	76	19	C
17	4	4	47	11	;	77	19	C
18	4	4	48	12	<	78	19	C
19	4	4	49	12	<	79	19	C
20	5	5	50	12	<	80	20	D
21	5	5	51	12	<	81	20	D
22	5	5	52	13	=	82	20	D
23	5	5	53	13	=	83	20	D
24	6	6	54	13	=	84	21	E
25	6	6	55	13	=	85	21	E
26	6	6	56	14	>	86	21	E
27	6	6	57	14	>	87	21	E
28	7	7	58	14	>	88	22	F
29	7	7	59	14	>	-	-	-

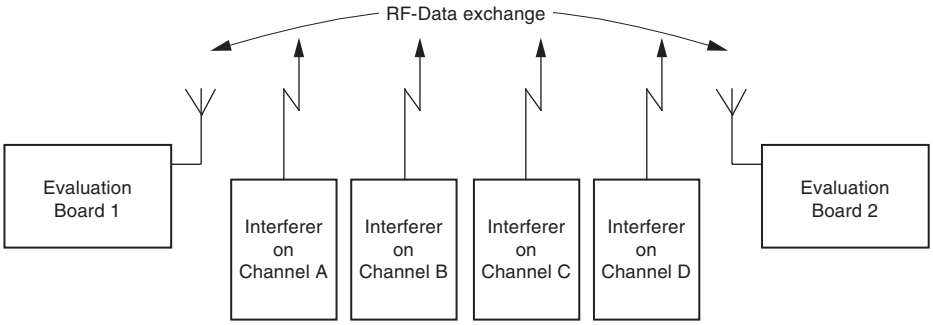
13.3 FHSS Info

The FHSS-Info menu displays the status of the adaptive FHSS. The contents of the FIFO that determines which subchannels have to be left out of the hopping scheme is displayed in the line *Blocked SC:* (Figure 13-1 on page 29). The symbol "/" indicates that no disturbed subchannels have been encountered yet. During normal operation, the menu will display the blocked subchannels (SC). If subchannels 7, 15, 18 and 3 are blocked, the LCD will display *Blocked SC: 7?B3*, indicating that the subchannels 7, 15, 18 and 3 (indicating the channels 28, 29, 30, 31, 60, 61, 62, 63, 72, 73, 74, 75, 12, 13, 14, 15) are left out of the hopping scheme. If the FIFO gets an overrun, the oldest value is kicked out of the FIFO, because this subchannel might not be blocked anymore. In Figure 13-1 on page 29 the LCD shows that subchannels 2, 1, 6 and 3 are blocked.

Figure 13-1. Adaptive FHSS

13.4 Adaptive FHSS Performance

Figure 13-2. Adaptive FHSS Test



This section describes the achieved performance when using the adaptive FHSS instead of normal FHSS. Assume a situation such as the one shown in Figure 13-2. There is an existing RF link between Reference_Design_Board_1 and Reference_Design_Board_2. Both are configured to use the adaptive frequency hopping implemented through Atmel's RF firmware. If there is no disturbance present, the link quality is 100% and indicates a perfect RF link. In the first step the interferer on channel A (representing any possible channel in the 2.4 GHz ISM band) is powered up and will transmit continuously on channel A. This will block channel A for the RF link of the two reference design boards.

At first the RF quality will be lowered by the disturbance, but after a short period of time (about half a second) both reference design boards will know about the blocked channel and kick the channel out of their hopping schemes. Now the other three interferers are powered up. This will result in another three blocked channels. The RF performance is temporarily lowered as the two reference design boards first recognize the blocked channels, but after a short time, the link quality will be again 100%, leaving out all the blocked channels.

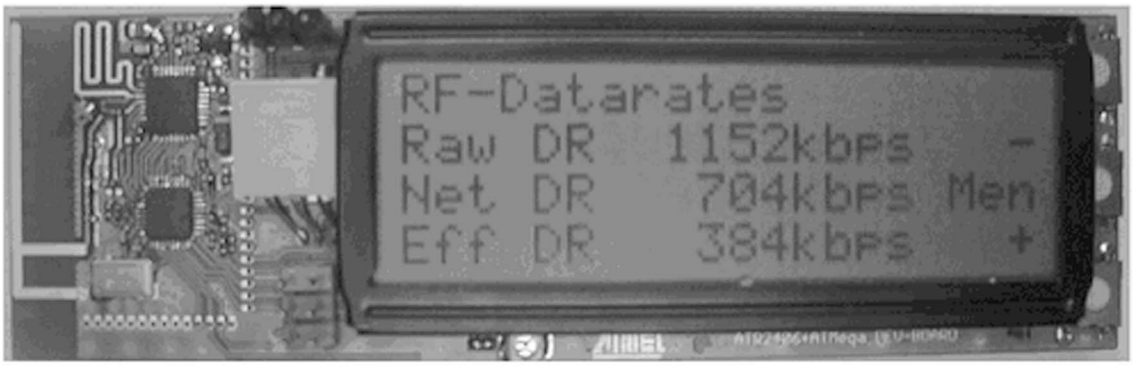
Test results with a test case as shown in Figure 13-2:

- With adaptive frequency hopping: RF quality = 100%
- Simple frequency hopping: RF quality = ~80%

13.5 RF Data Rates

The RF Data Rate menu (Figure 13-3) informs the user about the achieved data rates of the implemented RF link. The raw data rate (*Raw DR*) is 1.152 Mbps; this is the bit rate via the air when the ATR2406 is active. As the transceiver is not active all the time (loop filter settling, etc.) this data rate is lowered to 704 Kbps, and called the net data rate (*Net DR*). The net data rate accounts for the complete protocol overhead and the start, stop and MPC bits of the USART interface (remember the connection between the ATR2406 and the AVR is realized via a USART interface). The absolute effective payload throughput is 384 Kbps (*Eff DR*). This number can be increased if the protocol is changed or if, for example, the USART is used in normal (not MPC) mode. If the USART is used in normal mode, the effective payload throughput could be increased to 422 Kbps.

Figure 13-3. RF Data Rates



13.6 Configuring the RF Firmware

One goal of the RF firmware is to be adjustable to the specific needs of the user. The potential adjustments are outlined following.

- The adaptation of the necessary RF data rate is shown. In this description it is assumed that the user takes the basic protocol coming with the RF firmware. When talking about the data rate, we refer to the net data rate, or the payload throughput. To adjust the necessary data throughput the user has to take a look into "RF Task Timing" on page 8 and of course a close look into the RF firmware. The file *defines.h* is used to specify the necessary net data rate. In this file there are many definitions for a working RF link, where the user can specify the necessary net RF data rate. This is done via defines for the preprocessor of the C-Compiler. The hopping rate of the RF link is also changed by this definitions, as the RF firmware hops for each packet through the 2.4 GHz band. This means using the maximum net data rate, the RF firmware performs 1000 hops per second. Using, for example, only a quarter (384 Kbps/4) the RF firmware performs 250 hops per second. This high number of hops per second allows the design of a very robust RF link. Also, the protocol can be changed by the user to adopt it to the specific needs. Everything regarding the protocol can be changed except the start of frame (SOF) and the end of frame (EOF). These two things will give the user a large number of different configurations of the RF link.
- The reference design board firmware includes many features, such as adaptive frequency hopping and built-in self test. These features can be removed from the firmware in order to save CPU power and Flash memory. If self test is not needed, simply remove the files test.c and test.h from the project, and delete the function test_mode_pcb in the file *appl.c* (line 93). This will reduce the amount of Flash needed for the RF firmware by more than one kilobyte. A different way of removing the self-testing software, is to remove the #define SELF_TEST from the file *defines.h*.
- If no adaptive FHSS is needed, simply remove the define ADAPTIVE_FHSS from the file *defines.h*. This will also decrease the required CPU power and the Flash usage. (Switching off both self-test mode and adaptive FHSS saves about two kilobytes of Flash memory.)

Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

Regional Headquarters

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature

Printed on recycled paper.

4858B–ISM–07/05