ACSIJ Advances in Computer Science: an International Journal, Vol. 4, Issue 6, No.18 , November 2015
ISSN : 2322-5157
www.ACSIJ.org

ACSIJ
WWW.ACSIJ.ORG

Design of a Portable Random Access Wireless Network Transmitter

Rashid Hassani[1], Prabhu Gudapusetty[2] and Peter Luksch[3]

[1] Department of computer science, University of Rostock
Rostock, Germany
rashid.hassani@uni-rostock.de

[2] Department of computer science, University of Rostock
Rostock, Germany
prabhu.gudapusetty@uni-rostock.de

[3] Department of computer science, University of Rostock
Rostock, Germany
Peter.luksch@uni-rostock.de

Abstract

There is intensive attention on improving random media access control protocols in wireless environments. This paper proposes low-complexity, portable and flexible development of the wireless transmitter employing Random Access (RA) protocol. To develop this RA solution, the software architecture is roughly divided into three modules. The host software (i.e., micro-PC interface), Universal Serial Bus (USB) and Radio Frequency (RF) module. In our transmitter based scheme, the generated packets move from the higher layer (i.e., MAC layer) to the physical layer and then transmit over the air. At the other end, the packets will be received by the evaluation board, which is acting as a receiver. By using packet sniffer tool, we are able to sniff the radio packets from the micro-PC based RF-fronted wireless transmitter. Our results have been conducted through various test scenarios and emphasize the validity of our development in which, by our new developed RA solution, the generated radio packets are transmitted over the air successfully without any packet loss.

Keywords: *Random access, USB, Wireless network, Wireless transmitter.*

1. Introduction

The design of random media access control (RA MAC) protocol is widely considered as critical issue in wireless environments and currently is considered as an active research area for challenged environments such as the satellite or wireless sensor networks. Since, the number of devices capable of interconnecting is steadily increasing, if the contention among different users in shared medium is not appropriately controlled, this may lead to large number of collision, resulting in wastage of resources such as bandwidth and energy as well as system efficiency [1][2]. This leads to a rise of many interesting research questions on how to manage the shared medium efficiently. To improve the overall quality of service at the user end, there are various random media access control protocols which can be used. For example, for a large set of users, a distributed wireless MAC protocol is preferred, (i.e., a Random Access (RA) protocol). In RA protocols (e.g., ALOHA), the medium is accessed without coordination between the users, leading to possible collisions. RA protocol schemes are suitable for handling initial access, burst traffic and short packets in up-link satellite communication. However, the collision, propagation delay and packet loss are some series of pitfalls of this technique which may vary between different transmitter-receiver pairs [3][10][19]. This leads to a rise of many interesting research questions on how to manage the shared medium efficiently to avoid or resolve collisions and of course packet loss. Multi-User Detection (MUD) is a receiver based scheme where multiple transmitters are transmitting at the receiver end. The user data is separated based on a signature waveform [4][18]. However, in RA, the active number of transmitters at any particular time may not be known at the receiver. It is necessary to first know the active number of users involved in current transmission from the received signal, followed by MUD based on the signature of the waveform. Multi-packet reception reduces the collision because collided packets can be recovered by separating them from overlapping packets by using signal processing techniques [5][6][20].

The contribution of this work is to develop the RA solution which is a portable transmitter with wireless capabilities with proper connection to micro-PC and a radio frontend working in the WiFi frequency bands with ability to support various RA schemes (e.g., ALOHA, slotted ALOHA, etc.). This can be achieved by designing

the software architecture of transmitter through three modules:

i) Micro-PC module. This module generates packets and communicates with the slave side of the USB dongle through host software.

ii) USB module at the RF-frontend. This module is responsible to transfer data packets generated from a higher layer to the RF module in respond to the request of the host software.

iii) RF module. In this module, the data packets are encapsulated with preamble, sync word, length and Cyclic Redundancy Check (CRC) and then modulated and finally transmitted over the air.

The performance of our development has been evaluated through various test scenarios which emphasize the validity of our result.

The rest of the paper is organized as follows: Section 2 provides an overview of our proposed transmitter regarding software and architecture requirements. Section 3 discusses the implementation in details and how the packet is designed and also how the transmission process performs. Experimental results through various tastings are described in section 4. Conclusion and future works are left for the section 5.

2. Transmitter design

Our proposed wireless transmitter composed of micro-PC (i.e., Raspberry Pi) and RF transmitter (i.e., CC2511F32, USB dongle from Texas instruments). This low cost, portable and low power device is capable of handling different RA protocols. Following sections discuss the design requirements of this transmitter.

2.1 Software requirement

We classify the software requirements in two categories: functional and non-functional requirements. Functional requirements represent the functions of the system. In our case, the wireless transmitter generates the packets by reading a file of any format (i.e., txt, bmp, etc.). Generated packets move from MAC layer to physical layer via the USB interface and finally the packets will be transmitted over the air. Non-functional requirements relate to the tools used for development of the software. In our case, the software development (IDE) and debugging have been done using IAR embedded work bench using embedded C programming language through incremental development methodology [11]. As mentioned before, the design of the whole software module is divided into three modules: USB module, radio module and micro-PC interface. The complete software development is based on incremental development. Therefore, each module is developed and tested in a series of versions. New developed version of each module is integrated with the old version. Finally the complete software is developed and validated through its assigned version number.

We consider whole system as an object that can be partitioned in to several smaller objects and layers. Some of these objects are reusable while others need to be modified according to the hardware specifications. In our case, the SoC (System-On-Chip) is CC2511F32 USB dongle which is based on SoC CC2510F32. The most architecture and register settings such as radio module, Direct Memory Access (DMA) and Analog to Digital Converter (ADC) are the same in both SoCs except for the extra USB module.

As shown in the figure 1, the software architecture has been divided in to three distinct layers. The outermost is the application layer which specifies the system behaviors and performs functional and user requirements. The next is the Hardware Abstraction Layer (HAL). In this layer various core functionalities have been defined. The HAL has been divided into two distinct parts: the common and the target specific part. The common part contains common software which can be portable for most of the targets (e.g., a radio module). The target specific part contains specific software to run on a particular hardware platform. It also contains functionality related to the user interface (e.g., Light Emitting Diode (LED) and Liquid Crystal Display (LCD) module or I/O interface, clock, USB module, etc.).

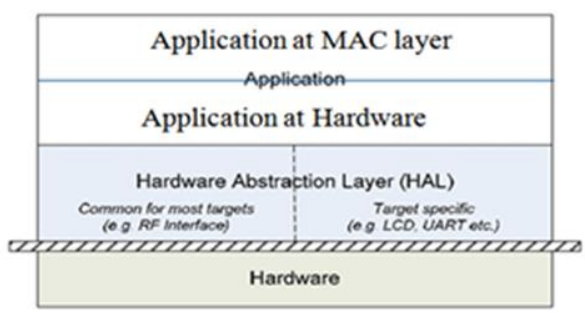

Fig. 1 Software architecture

The innermost layer is hardware dependent layer, which determines an appropriate action and needs to be taken for given set of inputs. These inputs drive the outputs to the desired state. The layer is pre and post-processed by the registers at the hardware end.

2.2 Hardware requirement

The hardware architecture of the wireless transmitter consists of two parts: micro-PC and RF transceiver. The micro-PC and RF-transceiver "CC2511F32" are the products of Raspberry Pi and Texas instruments respectively. The RF-transceiver is a low-cost 2.4 GHz SoC and is mainly used for low power wireless applications [12]. The micro-PC is a credit card sized, low power, affordable and easy to handle. The block diagram of the wireless transmitter is shown in figure 2.

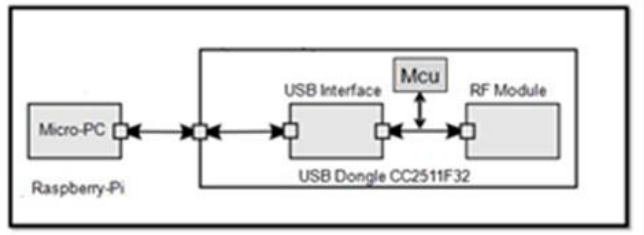

Fig. 2 Hardware architecture

2.3 USB frontend

The main objective of our task is to implement the USB interface between the host and RF module. Therefore, when the host sends a request to the USB controller, it has to respond based on the host request. The micro-PC acts as a host that initiates the communication. The USB dongle acts as slave device to the host. The communication by the USB dongle is done serially due to the supported drivers (e.g., Communication Device Class (CDC), Abstract Control Model (ACM) which emulate like serial device or virtual UART). After implementation of the USB interface, we need to implement the RF module to be able to transmit the packets generated from the micro-PC with the correct modulation and data rate. Therefore the whole task has been divided into two modules: The USB module and the RF module. In the following sections we will discuss the implementation details of USB modules and how it can be interfaced with the RF module and finally transmission of the packets over the air by RF module.

2.4 Radio model CC2511F32

Figure. 3 shows the block diagram of Radio model CC2511F32. By this device, the received RF signal is amplified by a Low Noise Amplifier (LNA) and is converted down in I/Q (in phase and quadrature phase signal) to Intermediate Frequency (IF). I and Q signals are digitized by ADCs. Later on Automatic Gain Control (AGC) and fine channel filtering and demodulating the received signal and packet synchronization are done digitally. In the transmitter side, synthesis of the RF frequency and the frequency synthesizer consist of an on-chip LC Voltage Controlled Oscillator (VCO) and a 90 degree phase shifter which generate I and Q Local Oscillator (LO) signals to down conversion mixers in receive mode. The high frequency crystal oscillator is used to generate reference frequency for frequency synthesizer and also for clocks for ADC and digital part.

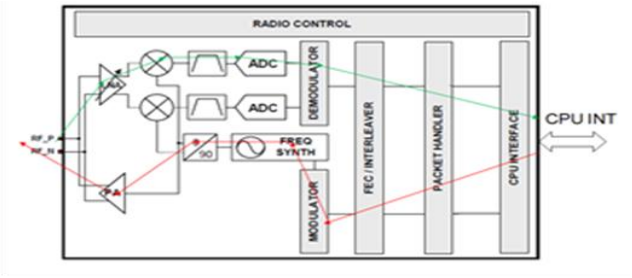

Fig. 3 Radio model CC2511F32 [7]

The digital baseband contains packet handling, channel configuration and data buffering. The SFR interface is used to access the data buffer via CPU. Control and status information are accessed via XDATA memory. In order to configure the radio module, there are a set of command strobes used by the CPU. These are single byte instructions in order to enable transmit and receive mode and also to enable and calibrate frequency synthesizer. These commands are as follows [7].

- STX: If in idle mode, perform calibration and enable transmit mode.
- SRX: If in idle mode, perform calibration and enable receive mode.
- SIDLE: Idle mode when no transmit or receive mode and the frequency synthesizer is OFF.
- SFXTXON: Enable and calibrate frequency synthesizer.
- SCAL: Calibrate the frequency synthesizer and turn off.
- SNOP: No operation.

The RF transceiver is based on the industry standard CC2500 with following characteristics:

- Operating frequency band of 2480 - 2483.5 MHz.
- Supports packet oriented system, on-chip for preamble detection, sync word detection, address check, variable and fixed packet length mode and automatic CRC check.
- Supports use of DMA. There is minimal intervention from CPU at high data rates.
- Supports programmable channel filter bandwidth.
- Supports three different modulation schemes: 2-Frequency Shift Key (FSK), MSK (Minimum Shift Key) and Gaussian Minimum Shift Key (GMSK).
- Optional automatic whitening and de-whitening of data.

- Programmable Carrier Sense (CS) indicator.
- Programmable preamble quality indicator and improved protection of sync word detection against random noise.
- Supports automatic clear channel assessment.
- Supports Link Quality Indicator (LQI).

The CC2511F32 has a built in state machine that can be used to switch between different operating modes. The radio switches to different states through command strobes or by internal events, such as TX_UNDERFLOW. The state of the radio can be figured out by reading the MARCSTATE status register. The radio has two active modes (i.e., STX and SRX). Writing these command strobes to the RFST register will initiate a TX or RX process. Whenever the radio enters TX mode, at first, a frequency check is done, then checks whether the radio is configured with High speed Crystal Oscillator (HSXOSC) with correct clock frequency and then switches to SIDLE (i.e., idle mode). Whenever the radio switches from SIDLE strobe to STX or SRX or vice versa, frequency synthesis or calibration is done. When transmit mode is activated, the chip will remain in Transmit (TX) state until the packet has been transmitted. After the packet has been transmitted, the radio changes the state. After transmit mode, the radio switches to receive mode through strobe command SRX. Figure. 4 shows the state diagram of the radio module.

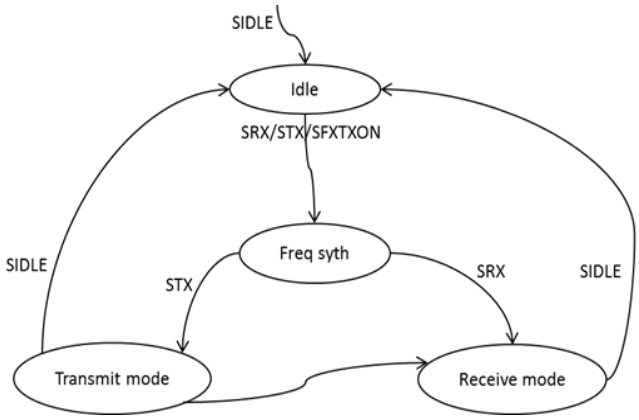

Fig. 4 State diagram of radio module [7]

2.5 Micro-PC

The micro-PC (i.e., Raspberry Pi) needs to be configured to control the USB based RF-frontend in order to initiate packet generation and transmission based on the developed RA MAC protocol for effective sharing of the medium. The task is to design a micro-PC based interface (i.e., how the host software which communicates with the USB based RF-frontend generates packets by reading a file of any size that may be in a txt, bmp, etc. format). After the file is read, it is encapsulated with a header and then transmitted via USB interface to RF module. The following section begins with an introduction to Raspberry Pi as well as developed method to implement micro-PC based interface. Further, we will discuss packet generation along with its testing procedure.

Raspberry Pi

The Raspberry Pi is a credit-card sized single board device developed by the Raspberry Pi foundation [13][17]. It is based on Broadcom 700MHz SoC and is the main central module for controlling the whole transmitter which contains a 32bit ARM1176JZF-S with 700 MHz RISC processor and a Video Core IV GPU. As shown in figure. 5, the Raspberry Pi is composed of a processing unit, memory, power supply, HDMI output, Ethernet port, USB ports and other interfaces. The main reason for choosing it as a micro-PC is its low cost, small in size, low power consumption and its support by Linux based operating system for developing the host software as well as developing various random access protocols at the MAC layer. The micro-PC acts as an intelligent system in which all the events are generated and the firmware at the USB based RF-frontend will respond based on the request of the host. A host is a PC which contains a host-controller and software.

As mentioned before, this project has hardware and software requirements. The hardware requirements have been met by configuring the micro-PC (i.e., Raspberry Pi) as a central controlling device for the wireless transmitter. The Raspberry Pi is compatible with USB 2.0 high speed port so that the USB based RF- frontend can be interfaced with a micro-PC. As software requirements, there is a Linux based operating system in Raspberry Pi. We also used a gcc/g++ compiler in order to compile the developed software under Linux. Regarding the USB based RF-frontend, the driver at the USB based RF-frontend needs to be compatible with Linux (i.e., CDC ACM). In order to develop the host software at the micro-PC to communicate with the USB dongle, we used Libusb v.1.0 [14]. Libusb is an open source library and an API platform to develop software in order to communicate with the USB peripheral. This API supports all kinds of USB transfers such as bulk, interrupt, control and isochronous. This API supports synchronous as well as asynchronous transmission.

Here, the micro-PC acts an intelligence system in which different RA protocols are developed and it is also responsible for generating the packets and establishing communication between the micro-PC and the USB dongle for wireless transmission.

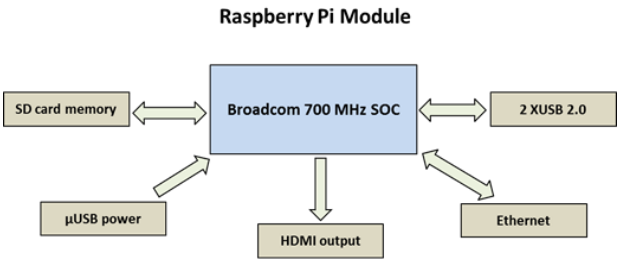

Fig. 5 Raspberry Pi module block diagram

As shown in Figure. 6, we describe how Libusb has been configured. The process begins with *Libusb initialization* (i.e., this indicates the start of session and this function must be called before any other function of Libusb). *Get device* list function will list all the devices connected to the micro-PC and *Get device descriptor* function will read the structures related to device that is located in the flash of a device like VID, PID, etc. The *Open* function opens the device, (i.e., in this case USB dongle) and returns the handle of the device. Further, the *conditional function* looks whether the kernel driver is active or not. If it is *active*, then detach the event, otherwise, the device can *claim* the interface. Once the device claims the interface, it starts communication with the USB dongle by sending control data using a *control* transfer. This function is used to configure the device such as enumeration. For example, in this case, the micro-PC sends control signals such as CTS, RTS, DTR, baud rate, data bits, stop bit and parity bit in order to establish communication between the host and USB dongle. Since the USB dongle has a CDC ACM driver, it emulates a virtual UART. Using bulk transfer mode, the micro-PC is able to transmit the dummy data (e.g., 64 byte) packets to the USB dongle using an OUT endpoint address. The dongle transmits the packets over the air and on the receiver end, the Evaluation Board (EB) which acts as a receiver, along with packet sniffer tool capture the packets from the micro-PC based wireless transmitter. Finally, the interface is released and Libusb session is *closed*.

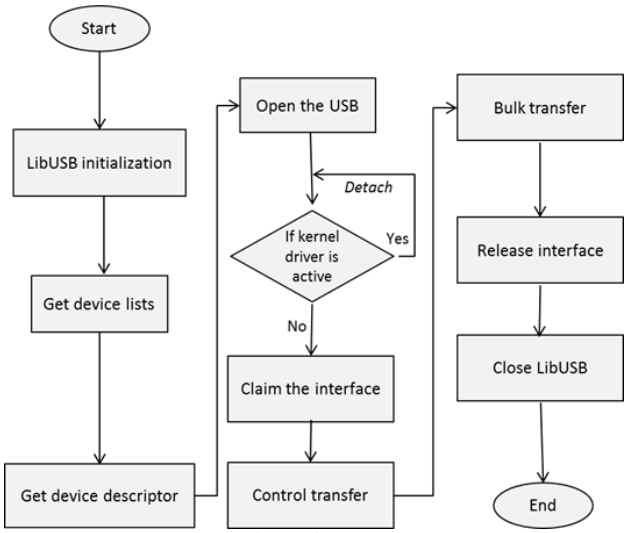

Fig. 6 Host software flow chart using Libusb

3. Implementation

This section explains the design of packet format generated by micro-PC and then the transmission operation in details.

3.1 Packet transmission and reception

The packet has been configured based on the following format shown in Figure. 7.

- Programmable preamble; 8-24 byte
- Programmable synchronization word16 or 32 bits
- Programmable or constant length byte
- Address (optional).
- Payload.
- CRC (optional).

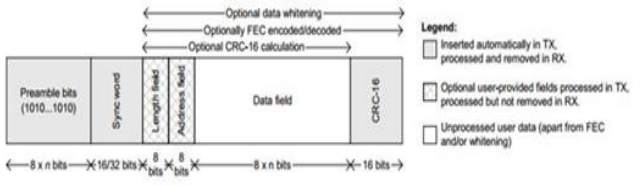

Fig. 7 Packet format [7]

The CC2511F32 has built in hardware support for packet oriented radio protocol. Preamble can be modified from 2-24 bytes and synchronization (Sync word) from 2 to 4 bytes. The length byte is optional and is enabled when the variable packet length mode is selected. Note that in fixed length packet transfer mode, the first byte in the payload will be the destination address if it is enabled, otherwise it will be payload. The maximum packet length is 0-255 bytes and an optional 2 byte CRC which is calculated over the data if it is enabled [7].

The CC2511F32 USB dongle supports three kinds of modulation schemes (i.e., MSK, 2-FSK, Gaussian frequency-shift keying (GFSK)). In this model, we selected MSK because it is similar to GMSK, which is the modulation used for the Automatic Identification System (AIS). MSK is a subclass of Continuous Phase Frequency Shift Key (CPSFK) and the MSK spectrum has no discrete components unlike other modulation. The MSK spectrum is wider than QPSK (Quadrature Phase Shift Key) but the side lobes fall much faster than QPSK. MSK encodes each bit as half sinusoidal and because of this feature, it has a constant envelop, compact spectrum, good error performance which it is mostly used in wireless systems [8][9]. Table 1 shows different modulation and data rates the USB dongle supports.

Table 1: Modulation

Modulation	***Min***	***Max***	***Unit***
MSK	26	500	kBaud
2-FSK	1.2	500	kBaud
GFSK	1.2	250	kBaud

The following procedure explains the packet transmission and reception.

In order to transfer packet, the data has to be written into RF data register (RFD). In receive mode, the data has to be read from same register. The RFD register has 1 byte FIFO in TX mode if the number of bytes written in RFD register is less than what it has assigned to transmit. Then the radio will enter into TX_UNDERFLOW state. RFIF.IRQ_TXUVF and RFIF.IRQ_DONE flags are set and when a byte is not read from the data register and the next byte is ready to be received, then TX_OVERFLOW flag is set. If no data is written in the RFD register and a STX strobe is issued, after the assertion of the RFTXRXIF flag, the radio will start sending the preamble without going into TX_UNDERFLOW state. A temporary FIFO is created in memory in TX/RX (i.e., TX FIFO and RX FIFO).

In transmit mode, the DMA transfers data from TX FIFO to RFD register and optional length field if variable packet length is enabled. If fixed packet length is enabled, then the first byte of payload is the destination address. The modulator sends the number of preamble and then 2 or 4 bytes of sync word, data content in the payload and 2 bytes CRC which is calculated over the payload. Once receiving the data, it moves from RFD to RX FIFO [7].
In receive mode, the demodulator and packet handler will look for the correct number of bytes of preamble and sync word. When found, the first byte is read. If variable packet length mode is used, then the first byte is the length byte and the packet handler stores the value as a packet length and based on the length of the byte, it checks the received data. If it's programmed as fixed length, then the payload is read and optional CRC check will be done.

3.2 Packet design and transmission process

This section gives a brief overview on how we have designed the packet generated by micro-PC and then the transmission operation in details.

3.2.1 Packet design

The detailed design of the packet format is represented in Figure. 8.

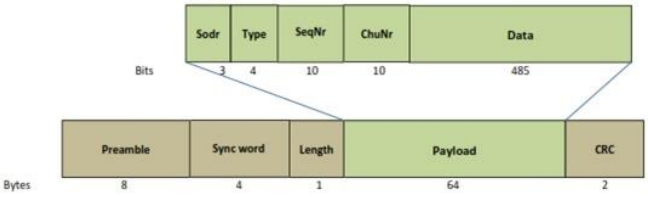

Fig. 8 Designed packet format

The data packets sent by the transmitter are generated by micro-PC. The payload of the data packet can be a portion of an image or text. The header of the packet contains:

- Source address (Sodr); distinguishes different transmitters.
- Type field; describes the data type of the payload.
- Sequence number (SeqNr); represents the portion of the file that is placed in the payload.
- Number of chunks (ChNr); represents the number of packets needed to send entire file.

Once the packet is generated, the MAC packet is moved via the USB interface to the RF-frontend using the control and bulk transfer mode of USB protocol and then transmitted over the air.

3.2.2 Transmission process

The transmission operation has been shown in details in Figure. 9. At the micro-PC end, the packet is generated with above defined format. The header is composed of 3 bits of source address, 4 bits of file type, 10 bits of sequence number and a payload of 64 bytes which is generated by MAC layer. In programming aspect, generally, two structures have been used. One defines the header of the packet and the other encapsulates the header and payload. The union includes both structures which is simply a whole packet. For packet transmission, a buffer must be allocated to load the complete file. The serial port has been configured with a baud rate of 38400 bps and has been provided with 8 data bits, no parity and 1 stop bit. Handshake signals have been enabled before starting communication. The packet header and payload are

autonomously filled once the program loop starts based on the total number of chunks which calculated through file size and payload. For example, to transmit 600 bytes (need 10 chunks, each of size 60 bytes) file, the sequence number is calculated based on the total number of chunks and the number of times the loop runs. Source ID is fixed and assigns a number to file type. The packets must be written to the USB port with a chunk size of 60 bytes along with 4 bytes of header information. Once the host sends the control signals, it is ready to transmit packets. Based on the preceding events, the firmware on the dongle will start responding to the host. The USB controller continuously waits for a packet and once it receives a 64 byte packet or less, it transfers the data to the endpoint of the FIFO. The CPU on the USB dongle reads the FIFO data and moves it to a TX buffer. A packet array has been created from the data held in the TX buffer. The DMA moves the packet array to RFD data register where the STX strobe is initiated. Once the strobe is initiated, the packet is transmitted over the air and the radio goes in RX mode.

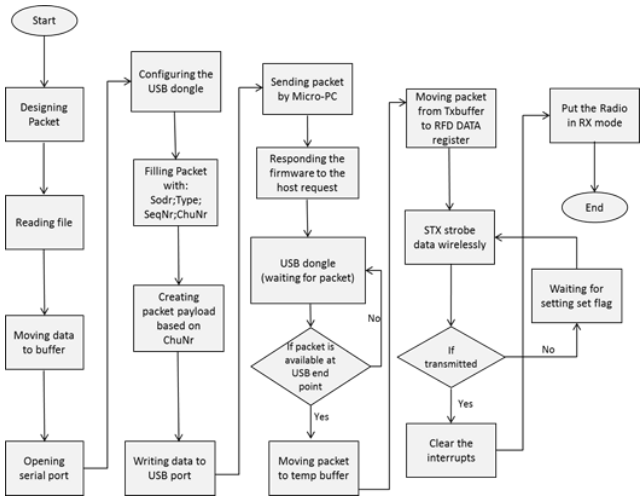

Fig. 9 Transmission process

4. Software testing and validation

This section discusses how the transmitter is tested along with the test bench setup and the software tools. We have developed the host software to communicate with the USB dongle. It is able to transmit data from the micro-PC towards the endpoint of the USB dongle. For example, by sending 64 bytes of dummy data, the dongle acts as a slave device and the firmware is able to respond based on the request of the host. During the test bench setup, the micro-PC based RF-frontend operates as a transmitter along with the packet sniffer tool to capture the radio packets from the micro-PC based RF-fronted. Further the test has been continued with the Smart-RF CC2510CC2511DK development kit from Texas instruments with CC2510F32 SoC which operates as a receiver. The CC2510F32 is a 2.4 GHz SoC which is based on a high performance leading transceiver CC2500. The CC2510F32 and CC2511F32 USB dongle have similar architecture except for clock frequency with 24 - 27 MHz for the CC2510 and 48 MHz for the CC2511. The EB has been configured to hold the characteristics of transmitter. In order to begin sniffing of the transmitter, the Smart-RF EB with CC2510F32 SoC is flashed with firmware from Texas instruments (i.e., sniffer_fw_ccxx10_usart0_alt1.hex) through specifications such as transmitting frequency, data rate and modulation. To sniff the radio packets of the capturing device, the radio configuration tab in the packet sniffer tool needs to have the configuration file of capturing radio which is generated using the Smart-RF studio. The procedures are as follows:

- Flash the transmitter HEX file in the USB dongle using Smart-RF flash programmer.
- Plug the USB dongle into the micro-PC USB port and plug the CC2510F32 SoC EB to the PC.
- Open and run the packet sniffer tool and configure the EB with radio configuration file which is generated from Smart-RF studio.
- Run the host software in the micro-PC. It generates the packets with header and payload by reading the file of any size at higher layer.
- Push the packet to a lower layer and transmit it over the air. On the other end, the receiver receives the packets and then towards sniffing tool.

The radio module has been configured with a base frequency of 2480 MHz, carrier frequency of 2479 MHz with data rate of 500 KBaud, received filter bandwidth of 750 KHz with MSK modulation, zero phase transition time, channel spacing of 199.951172, 0 channel number and with 0 dbm TX power. We have performed three test cases as follows.

4.1 First test case

As a first test case, the radio module has been tested. In the radio module, the packets are generated within the USB dongle with a certain packet format such as 1 byte of packet length, 2 bytes of TX ID and 4 bytes of sequence number and dummy payload. The generated packets have been transmitted by the dongle successfully over the air and on the other end, the EB receives and captures the packets using the packet sniffer tool.

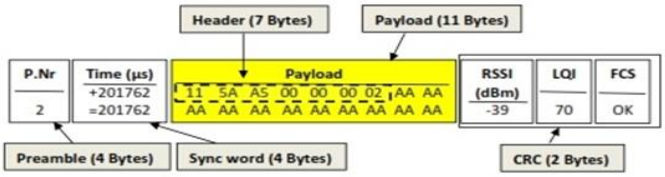

Fig. 10 Sample packet format of radio module

The packet format and sniffer data (captured packets) are shown in figure. 10 and 11 respectively. The First byte of payload represents the length byte and the next two bytes represent TX ID to distinguish among different transmitter at receiver end. The next 4 bytes represents as a sequence number to identify which packets have been received successfully and the reaming data is payload.

P.Nr	Time (µs)	Payload	RSSI (dBm)	LQI	FCS
1	+0 =0	11 5A A5 00 00 00 01 AA AA AA AA AA AA AA AA AA AA	-39	49	OK
2	+201762 =201762	11 5A A5 00 00 00 02 AA AA AA AA AA AA AA AA AA AA	-39	70	OK
3	+201761 =403523	11 5A A5 00 00 00 03 AA AA AA AA AA AA AA AA AA AA	-39	70	OK
4	+201761 =605284	11 5A A5 00 00 00 04 AA AA AA AA AA AA AA AA AA AA	-39	65	OK
5	+201761 =807045	11 5A A5 00 00 00 05 AA AA AA AA AA AA AA AA AA AA	-39	80	OK
6	+201761 =1008806	11 5A A5 00 00 00 06 AA AA AA AA AA AA AA AA AA AA	-39	78	OK
7	+201761 =1210567	11 5A A5 00 00 00 07 AA AA AA AA AA AA AA AA AA AA	-39	100	OK

Fig. 11 Packet sniffer data of radio module

4.2 Second test case

The second test case has been done over the micro-PC based RF transmitter module using Libusb 1.0 API. The setup and the radio configuration remain same as previous test with some more configurations in the host software at micro-PC. Using Libusb API 1.0, the communication has been established between the micro-PC and RF-frontend. Dummy packets of 64 bytes have been generated at the micro-PC. Upon generation of these packets from the higher layer, they are transferred using control and bulk transfer mode via the USB interface to the physical layer. When packets are received at the RF-frontend, they are transmitted over the air. On the other end, packets are received and captured by the EB using packet sniffer tool. The packet format and captured packets are shown in figure. 12 and 13 respectively. Here, the first byte is length byte and remaining is payload.

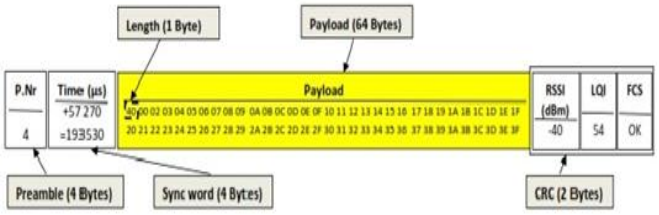

Fig. 12 Sample packet format of micro-PC

P.Nr	Time (µs)	Payload	RSSI (dBm)	LQI	FCS
1	+0 =0	40 00 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F	-40	82	OK
2	+69199 =69199	40 00 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F	-40	44	OK
3	+67061 =136260	40 00 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F	-40	54	OK
4	+57270 =193530	40 00 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F	-40	54	OK
5	+55155 =248685	40 00 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F	-40	68	OK
6	+62088 =310773	40 00 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F	-40	64	OK
7	+54099 =364872	40 00 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F	-40	80	OK

Fig. 13 Packet sniffer data of micro-PC

4.3 Third test case

The final test case has been carried out through micro-PC based RF transmitter using POSIX terminal interface as host software [15]. The test setup remains the same as before, but the dongle is flashed with the firmware of a USB RF-transmitter. This host software has some features such as the ability to generate the packets by reading a file of any size and encapsulates them with the header. The maximum size of the packet that can be transmitted is 64 bytes due to the limitation at the USB RF- frontend (i.e., the USB diver CDC ACM supports maximum packet size of 64 bytes).

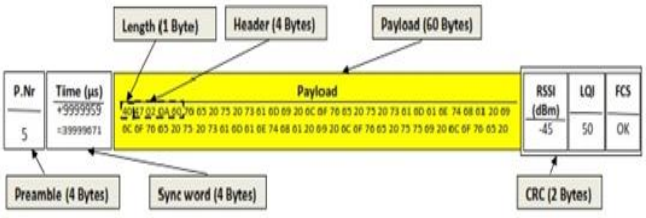

Fig. 14 Sample packet format of micro-PC using POSIX terminal

To better assess the impact of packet size, in this test case, we have examined small and large packet sizes with same radio configuration as mentioned above. At the first step, the host software reads small size file of 311 bytes. It starts by reading a complete file and allocating a buffer according to the size of the file. The reason is that the host software cannot send a complete file due to limitations at the USB dongle (i.e., maximum size is 64 byte for CDC ACM). Therefore, the file is divided into 64 bytes chunks with 4 bytes of header and 60 bytes of payload. Note that, the total number of chunks is calculated using the size of the file and maximum size of the payload that can be transmitted via the USB RF-frontend. For example, to

transmit 311 bytes, the maximum payload that can be transmitted is 60 bytes and we require 6 packets to complete the whole transfer. Figure. 14 shows the packet format and figure. 15 shows the complete transfer of 311 bytes.

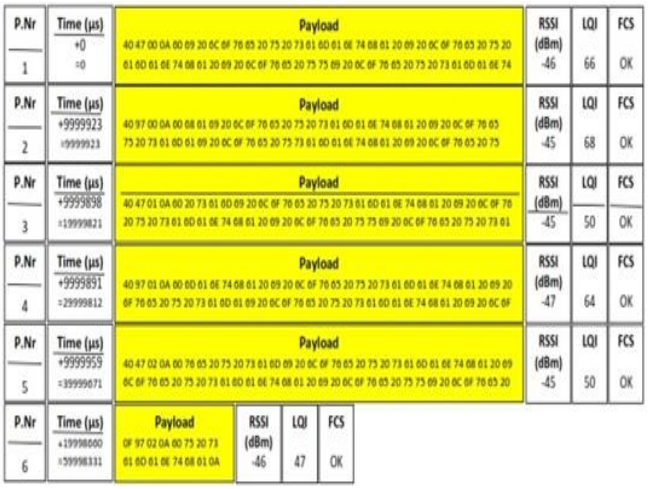

Fig. 15 Packet sniffer data of micro-PC using POSIX terminal for 311 bytes

The second step of the test has been done by reading a file with 416 bytes size and transmitting it via USB interface to RF-frontend and then transmitting it wirelessly as shown in figure. 16.

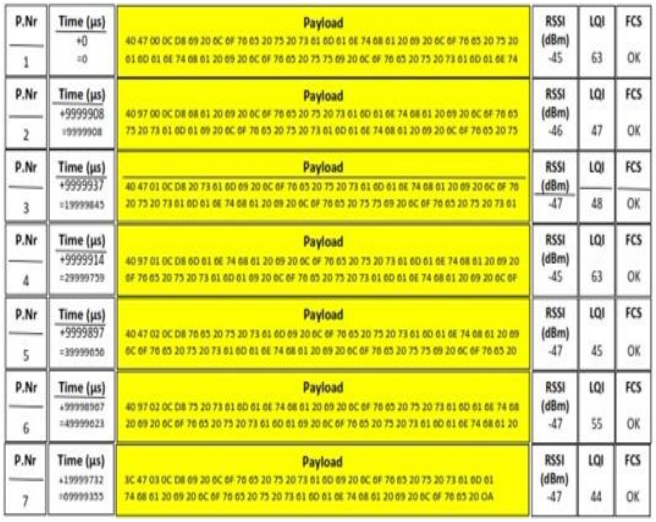

Fig. 16 Packet sniffer data of micro-PC using POSIX terminal for 416 bytes

5. Conclusions

In this work, we have designed and developed a portable random access wireless transmitter composed by micro-PC and a radio frontend working in the WiFi frequency bands to support different random access protocol. To achieve this RA solution, we needed a deep understanding of proposed hardware architecture to implement different modules such as USB, radio and micro-PC. At the beginning, the packets at the micro-PC side have been successfully generated. Then an appropriate USB link between the micro-PC and the RF-frontend at the USB dongle has been implemented. By setup configuration, the RF frontend is able to transmit and receive wireless data using the evaluation board. The complete transmitter with the micro-PC based RF-frontend along with the CC2510F32 EB which emulates a receiver, have been finally tested with a packet sniffer tool. As a result of this project, the generated radio packets at the micro-PC side have been transmitted over the air successfully without any packet loss.

As for the future work, the transmission blocks at the developed micro-PC end, such as encoding, filtering, etc., has to be implemented and tested using various random access protocols.

Acknowledgments

Part of this work has been carried out at the Institute of Communications and Navigation of the German Aerospace Center (DLR) in Oberpfaffenhofen, Germany. The Authors would like to thank Dr. Andrea Munari and Mr. Federico Clazzer who supervised this work at DLR Company for the resources and technical support [16].

References

[1] P. Pal, "Medium access control (mac) techniques", IIT, Kharagpur, available at "http://nptel.ac.in/courses/106105080/pdf/M5L2.pdf", accessed on 13/11/2015.

[2] R. Rom, M. Sidi, "Multiple access protocols: performance and analysis", available at "http://books.google.de/books?id=yAVTAAAAMAAJ", accessed on 13/11/2015.

[3] N. Abramson, "The throughput of packet broadcasting channels", IEEE Transactions on communication, vol. 25, no. 1, pp. 117–128, 1977.

[4] B. Chen, L. Tong, "Traffic-aided multiuser detection for random- access cdma networks", IEEE Transactions on Signal Processing, vol. 49, no. 7, pp. 1570–1580, 2001.

[5] H. Lee, "Random access schemes for multichannel communication and multi-packet reception in wireless networks", Ph.D. dissertation, University of Adelaide, 2011.

[6] Rashid Hassani, Ganesh Chavan, Peter Luksch, "Optimization of Communication in MPI-Based Clusters", In proceedings of the international conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC 2014), pp. 143-149, 2014.

[7] "Low-Power SoC (System-on-Chip) with MCU", Texas Instruments, available at

"http://www.ti.com/lit/ds/swrs055g/swrs055g.pdf", accessed on 13/11/2015.
[8] S. Pasupathy, "Minimum shift keying: A spectral efficient modulation", IEEE Communications magazine, available at "http://www.q.hscott.net/reads/QMSK.pdf", accessed on 13/11/2015.
[9] Rashid Hassani, and Peter Luksch, "Optimizing Bandwidth by Employing MPLS AToM with QoS Support", In proceedings of the IEEE NAS 2012, pp. 104-108, 2012.
[10] Lei Zheng, Lin Cai, "AFDA: Asynchronous Flipped Diversity ALOHA for Emerging Wireless Networks with Long and Heterogeneous Delay", IEEE Transactions on Emerging Topics in Computing, pp. 1-11, 2014
[11] "IAR embedded workbench", Debugging using the IAR C-spy debugger, available at "http://supp.iar.com/FilesPublic/UPDINFO/005832/arm/doc/infocenter/ tutor_debugging.ENU.html", accessed on 13/11/2015.
[12] "Silicon Labs", USB overview, available at "http://www.silabs.com/Support% 20Documents/Software/USB_Overview.pdf", accessed on 13/11/2015.
[13] V. Sathish Kumar, G. Senthilkumar, K. Gopalakrishna, "Embedded image capturing system using raspberry pi system", International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), vol. 3, no. 2, pp. 213-215, 2014.
[14] "libusb", available at "http://libusb.sourceforge.net/api-1.0/", accessed on 13/11/2015.
[15] Michael R. Sweet, "Serial Programming Guide for POSIX Operating Systems", available at "http://www.netzmafia.de/skripten/ hardware/Seriell/SerialPort_Programming_c.pdf", accessed on 13/11/2015.
[16] "University of Rostock", available at "http://wwwvhr.informatik.uni-rostock.de/projekt_bachelor_und_master_arbeiten/vhr_diplomarbeiten/", accessed on 13/11/2015.
[17] Vujovic. V, Bosnia Herzegovina, Maksimovic. M, " Raspberry Pi as a Wireless Sensor node: Performances and constraints", Information and Communication Technology, pp. 1013-1018, 2014
[18] Cedomir Stefanovic, Petar Popovski, "ALOHA Random Access that Operates as a Rateless Code", IEEE Transactions on Communications, vol. 61, no.11, pp. 4653-4662, 2013.
[19] Minh Hanh Ngo, Vikram Krishnamurthy, Lang Tong, "Optimal Channel-Aware ALOHA Protocol for Random Access in WLANs With Multipacket Reception and Decentralized Channel State Information", IEEE Transactions on signal processing, vol. 56, no.6, pp. 2575-2588, 2008.
[20] Rashid Hassani, Shiv. R. P. N Amgoth, Peter Luksch, "Efficient Consolidation of Virtual Machines for HPC Applications in Cloud", International Journal of Intelligent Information Processing, vol. 5, no. 4, pp. 19-26, 2015.

Rashid Hassani received his BE degree in Information Technology from VTU, India. He graduated with M.Sc. degree in Computer System Engineering emphasized in Embedded and Cooperating Systems from Halmstad University, Sweden. He is currently a research assistant and academic teacher in the department of computer science at University of Rostock, Germany. His research interests include parallel and high performance computing, Cloud computing and networking protocols.

Prabhu Gudapusetty received his M.Sc. degree in Computational Engineering at university of Rostock.

Peter Luksch received his Ph.D. degree in Parallel Discrete Event Simulation on Distributed Memory Multiprocessors from Technische Universität München, Germany. He finished his Postdoctoral Lecture Qualification (Habilitation) in Increased Productivity in Computational Prototyping with the Help of Parallel and Distributed Computing. Currently, Prof. Dr. rer. nat. habil Peter Luksch is head of the department of Distributed High Performance Computing at University of Rostock, Germany.