Lab 1:
Reactive Jamming

Physical Layer Security in Wireless Systems

Secure Mobile Networking Lab – SEEMOO
Matthias Schulz (mschulz@seemoo.tu-darmstadt.de)

November 11, 2014

TECHNISCHE
UNIVERSITÄT
DARMSTADT

SEEMOO
SECURE MOBILE NETWORKING

Whenever you touch the Wireless Open-Access Research Platform (WARP) or the antenna attached to the WARP, you have to protect yourself against electrostatic discharge (ESD) by wearing a wrist band that is connected to ground.

Contents

1 Introduction

In today's lab exercise, you will create a reactive WiFi jammer on the Wireless Open-Access Research Platform (WARP), which is a Software-Defined Radio (SDR) with a reference implementation of 802.11g. The jammer should receive WiFi transmissions and analyze their Medium Access Control (MAC) addressing to decide if a frame should be jammed or not. The physical layer of the frame is processed in real-time on a Field Programmable Gate Array (FPGA). You have to program a MicroBlaze processor in C which can process received frames while they are still in reception. If you decide to jam a frame, you switch your radio frontend from receiving to transmitting and directly start transmitting wireless frames to disturb the reception of the frames you want to jam. At the end, you have to analyse the impact of your jammer on a wireless link between a computer and an access point. In the following, we first give you an introduction into the 802.11 reference design for the WARP and then we continue with the lab tasks.

2 Background on the 802.11 reference design

In Figure 1, we illustrate components of the WARP and the implementation of the 802.11 reference design in the FPGA. From left to right, you see the following components. First, the antenna that is used for transmission and reception. With

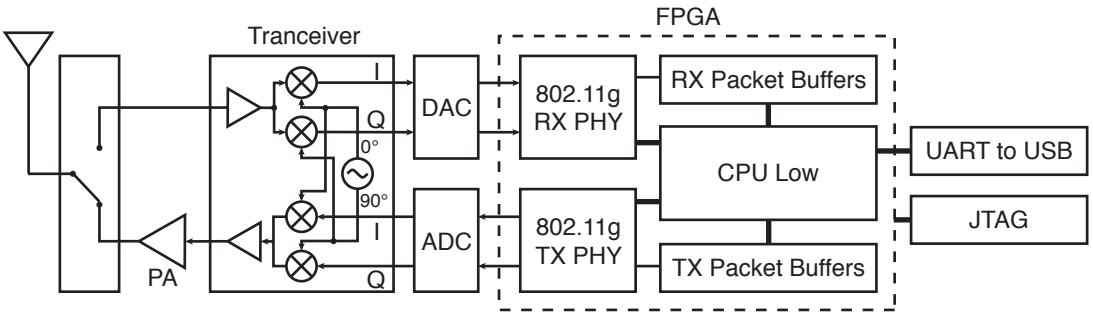

Figure 1: Block diagram of the 802.11 reference design on the WARP

Whenever you touch the Wireless Open-Access Research Platform (WARP) or the antenna attached to the WARP, you have to protect yourself against electrostatic discharge (ESD) by wearing a wrist band that is connected to ground.

the WARP you can either transmit or receive at a time. Hence, the antenna is followed by an antenna switch to either connect the antenna to the transmit or the receive path. In the transmit path, you find a Power Amplifier (PA) with a fixed gain. It is fed by the output of the transceiver chip, that includes a quadrature modulator to upconvert a complex baseband signal to Radio Frequency (RF). The receive path looks similar to the transmit path and is used to downconvert from RF to baseband. Between the transceiver chip and the FPGA, we have Digital-to-Analog Converters (DACs) and Analog-to-Digital Converters (ADCs) to convert between continuous analog and discrete signals.

The contents of the FPGA are twofold. On the one hand, there are the implementations of the 802.11 physical layer. On the other hand, there are two MicroBlaze processors to handle the MAC layer, as well as, the control of the physical layer implementation. As we require only the *CPU low* in this lab, the *CPU high* is omitted in Figure 1.

Whenever the WARP is not transmitting, it is listening for incoming transmissions. As soon as the *RX PHY* core detects a frame, it starts to process the transmitted symbols and to extract the transmitted data bits, which it directly writes into *RX Packet Buffers*. These buffers are in a shared memory, that is directly accessible by the *RX PHY*, as well as, the *CPU low*. Accessing the memory block by the *CPU low* is simple, as the block is directly mapped into the address space of the processor. The same is true for control registers of the *RX PHY*. To check if a new frame is currently being received or to select a target packet buffer, the *CPU low* directly reads from or writes to registers that are mapped into the processors address space. It is important to note, that the *CPU low* can already start processing incoming frames, while they are still being received by the *RX PHY*. This allows us to build a reactive jammer just by changing the program running on the *CPU low*.

To jam a transmission, we have to prepare a new frame in the *TX Packet Buffers*, that hold frames that are supposed to be transmitted. When a new frame is ready, we can use the *CPU low* to abort the reception in the *RX PHY*, flip the antenna switch to the transmit path and instruct the *TX PHY* to read the bits from our transmit buffer and to modulate them into a complex baseband signal that is directly upconverted and transmitted. Just in time to disturb the reception of the frame we intend to jam.

In Figure 1, there are only two components left to explain. The *UART to USB* block is used as a serial console to the *CPU low*. Whenever, we write to the standard output in our C program (e.g., by using the xil_printf function) the output can be observed in a terminal window on a computer connected to the *UART to USB* port. The *JTAG* port is used to transfer FPGA images to the FPGA. Those images contain the implementation of the signal processing blocks, as well as, the MicroBlaze processors and the code that is running on the latter. During the lab, you will connect to the *JTAG* port with a *USB to JTAG* adapter to directly load FPGA images from the Eclipse-based development environment.

3 Task 1: Starting the development environment

You will perform your lab experiments, using a Virtual Machine (VM) that includes all necessary development tools. To this VM, you have to hand through the JTAG adapter, as well as, the connection to the serial console. We created a snapshot of the VM, when all tools were already started. If you load this snapshot you might not need to perform the following steps and you might directly continue working at Task 2.

Use the following login data to sign in:

- Username: physec
- Password: physec

Then start a terminal and enter:

- /opt/Xilinx/14.4/SDK/settings64.sh

Then you can start the Eclipse-based Xilinx Software Development Kit (SDK) by entering:

- sudo /opt/Xilinx/14.4/SDK/SDK/bin/lin64/xsdk

Use putty as a terminal client:

- sudo putty

. . . and connect to the terminal console by opening the predefined *WARP UART* session.

Whenever you touch the Wireless Open-Access Research Platform (WARP) or the antenna attached to the WARP, you have to protect yourself against electrostatic discharge (ESD) by wearing a wrist band that is connected to ground.

4 Task 2: Running the jamming code on CPU low

To run the jamming implementation, you have to program the FPGA by executing the *Program FPGA* command in the *Xilinx Tools* menu in the *Xilinx SDK*. The default *bitstream* and *BMM file* settings are fine. The software configuration for *CPU high* should be *bootloop* and the one for *CPU low* should be set to the *wlan_mac_low_jamming.elf* file. After programming the FPGA, the WARP reboots and should now receive WiFi frames. You can take a look at the serial console and already see some information.

5 Task 3: Printing debug messages and reacting to user interaction

Now, we take a closer look on how the *CPU low* code works. In the main function, we see some function calls to the xil_printf function to output some information on the serial console. Then some initialization functions are called, which are followed by a while loop. During program execution, the *CPU low* always cycles through the while loop and executes each of the functions in the loop. The wlan_mac_low_poll_frame_rx function, for example, always checks if the *RX PHY* received a new frame and then calls the frame_receive function to handle the reception. The push_button_checker function on the other hand is still empty and should contain the implementation according to the following instructions:

- As a first test, insert a call to xil_printf [1] and simply print a dot (“.”).
- Run the program by programming the FPGA.
- You should see that many dots get printed to the serial console.
- Now, we first want to read if the *up* pushbutton was pressed and only then print a dot. Therefore, we need to read the register that contains the status of the pushbuttons (userio_read_inputs(USERIO_BASEADDR)) and check if the correct bit (W3_USERIO_PB_U) for the *up* pushbutton is set.
- Check in the serial console if only one dot appears for each button press. If multiple dots appear, change your code, so that only one dot appears for each button press.

As soon as, you finish this task, we will start the transmission of WiFi frames on each button press.

6 Task 4: Transmitting frames on button press

In this task you should create a valid WiFi frame that can be received by a normal WiFi device running in monitor mode on the same WiFi channel as the WARP. The channel WIFI_CHANNEL should be set to 14, which is reserved for research applications in Germany. All MAC addresses in that frame should be set to de:ad:be:ef:0x:0x, where x should be your group number. The WiFi frame should consist of the following MAC header:

- u8 frame control 1 set to MAC_FRAME_CTRL1_SUBTYPE_DATA
- u8 frame control 2 set to MAC_FRAME_CTRL2_FLAG_FROM_DS
- u16 duration id set to 0
- Three MAC addresses set to your group’s MAC address
- u16 sequence control set to 0

In the WARP, WiFi frames have to be put into a *TX Packet Buffer* and then the *TX PHY* core has to be instructed to use this packet buffer to transmit the frame. Luckily, the code already contains a frame_transmit function, that handles the control of the *TX PHY*. Whenever you call the frame_transmit function, the frame in the packet buffer passed during the call will directly be transmitted. The structure of each packet buffer is as follows:

- tx_frame_info struct that is used to save information about the transmission. This part is not used in our jammer.

[1] Note that you should always use the lightweight xil_printf function and not the regular printf function to print out debug messages.

Whenever you touch the Wireless Open-Access Research Platform (WARP) or the antenna attached to the WARP, you have to protect yourself against electrostatic discharge (ESD) by wearing a wrist band that is connected to ground.

- PHY_TX_PKT_BUF_PHY_HDR_SIZE byte long physical layer header containing control information for the *TX PHY* core, such as: transmit power, transmit rate, frame length, etc. Before sending a frame with the frame_transmit function, you should set the transmit power in dBm as a value between TX_POWER_MIN_DBM and TX_POWER_MAX_DBM. Thereto, you can use the set_tx_power function. You also have to define, which antenna interface to use on the WARP. As we only have one antenna attached to port A, you should call set_tx_ant_mode and set TX_ANTMODE_SISO_ANTA.

- After the physical layer header comes the MAC header that is described above.

The starting address of the tx_frame_info struct in the *TX Packet Buffers* can be found with the TX_PKT_BUF_TO_ADDR macro. Your task is to create and send a valid WiFi frame whenever you press the down button next to the seven segment displays on the WARP. When calling the frame_transmit function, set the transmit rate to WLAN_PHY_RATE_BPSK12 and the low_tx_details to NULL.

7 Task 5: Analyzing received frames

Now, we take a look at the frame_receive function that gets called whenever a new frame is received. Currently, it waits until the end of the reception (call to wlan_mac_dcf_hw_rx_finish) and then it overwrites the contents in the packet buffer with zeros and tells the *RX PHY* core to use this packet buffer again for the next reception.

Each of the *RX Frame Buffers* contains the following:

- rx_frame_info struct that is used to save inforamtion about the transmission. This part is not used in our jammer.

- PHY_TX_PKT_BUF_PHY_HDR_SIZE byte long physical layer header that contains additional information similar to the transmit physical layer header.

- mac_header_80211 struct that contains the MAC header of the received packet.

Your task is to extract the MAC addresses of the received packets and print them to the serial console.

8 Task 6: Filtering for MAC addresses

Now, filter for MAC addresses that you intend to jam. There will be frames with MAC addresses in the form of DE:AD:BE:EF:xx:xx, where x is either 0, 1, 2, 3 or 4. The frames, where x equals 0 should never be jammed. The different groups should only jam those frames where x equals their group numbers. You can use wlan_addr_eq to compare MAC addresses.

9 Task 7: Jamming

Whenever you receive a frame, that contains the MAC address that you should jam, according to your group number, directly start a transmission. Keep in mind, that you have to start jamming before the packet reception is finished. That means before the call to the blocking wlan_mac_dcf_hw_rx_finish function. However, you still need to wait until a sufficient number of samples is received, so that you can analyze the MAC addresses. Thereto, you can use the wlan_mac_get_last_byte_index function that tells you how many bytes are currently received. Hint: The thirteenth byte is the beginning of the first MAC address.

10 Task 8: Performance evaluation

Using the available equipment (such as, a netbook running Wireshark), evaluate the performance of your jammer, by analyzing the effect on the packet throughput when the jammer is turned on or off.

Whenever you touch the Wireless Open-Access Research Platform (WARP) or the antenna attached to the WARP, you have to protect yourself against electrostatic discharge (ESD) by wearing a wrist band that is connected to ground.

11 Lab report

Use the LaTeX template for lab reports and write exactly two pages about our lab experiment. You can hand in your reports in a team or on your own. However, both need to contribute the same amount of material. Plagiarism will not be tolerated! The following points should be included in the report:

- An introduction, describing the goal of this lab experiment and the differentiation to other WiFi jamming schemes, for example, those using deauthentication frames.
- Describe how the frame handling works on the WARP. Why there are multiple receive and transmit buffers and how they are organized.
- Describe, how you evaluated the performance of your jammer.
- Discuss, how your implementation could be improved, for example, in terms of energy consumption and detectability.

Acronyms

SDR	Software-Defined Radio
DAC	Digital-to-Analog Converter
ADC	Analog-to-Digital Converter
WARP	Wireless Open-Access Research Platform
MAC	Medium Access Control
FPGA	Field Programmable Gate Array
PA	Power Amplifier
RF	Radio Frequency
LLC	Logical Link Control
DSAP	Destination Service Access Point
SSAP	Source Service Access Point
ESD	electrostatic discharge
VM	Virtual Machine
SDK	Software Development Kit