Chisel-based BLE Baseband

Jerry Duan, Yalun Zheng, Mingying Xie, Borivoje Nikolic
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, USA

Abstract

As technologies develop for Bluetooth, an open source CPU is needed for faster and easier testing of the most recent Bluetooth. There is limited research conducted on connecting BLE to an open-source CPU, such as RocketChip. In order to make it easier and more thoroughly to test BLE, we connected a BLE baseband to a RISC-V RocketChip, a Chisel based CPU. The entire design consists of a Packet Assembler (PA), a Packet Disassembler (PDA), Cyclic Redundancy Check module (CRC), Whitening/Dewhitening module, FIFOs, and a RISC-V RocketChip. Experimental C tests demonstrate that two chains (the Packet Assembler Chain and the Packet Disassembler Chain) work well individually, and the loop that is an integration of both also works well.

1. Introduction

The commercialized BLE standard is defined by Bluetooth Special Interest Group (SIG) and the current latest version of Core Specification is Bluetooth v 5.0. Our implementation strictly follows the Spec 5 standard and this section will go into details about the BLE packet structure [1].

Each BLE packet follows a fixed pattern that includes four main parts: Preamble, Access Address, Protocol Data Unit(PDU), and CRC. Fig. 1 gives an overview of the BLE packet structure.

1.1. Preamble

The preamble is a fixed zero-one pattern used to perform frequency synchronization, symbol timing estimation, and Automatic Gain Control (AGC) training for the receiver [2]. For LE 1M packet, the preamble (either to be 01010101 or 10101010) is determined by the last bit of access address.

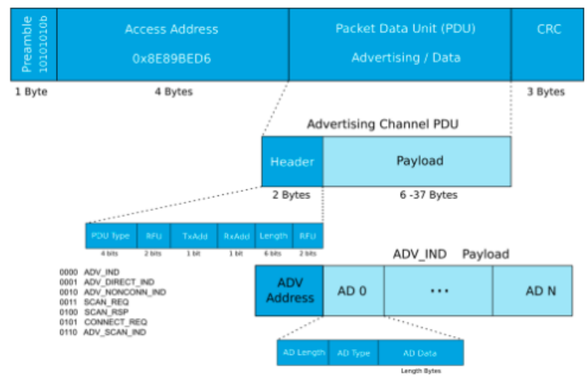

Figure 1. Overview of the BLE Packet Structure

1.2. Access Address

AA defines the physical channel access code. Except "AUX SYNC IND" and any "AUX CHAIN IND" PDU types, the AA for all other advertising channel packets shall be 0x8E89BED6.

1.3. PDU

PDU consists of two parts: the logical link identifiers(PDU header) and logical transport (PDU payload). For the advertising channel PDU, the 16-bit header is as shown in figure Fig. 2.

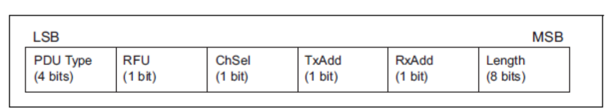

Figure 2. PDU Header

The first four bits indicate the PDU type. Fig. 3 shows other possible types. The payload starts with a 6-byte advertising address. It also contains 1-byte that infers the length and 1-byte that describes the payload section. Then we can have variable length of data (encoded with ASCII code).

PDU Type	PDU Name	Channel	Permitted PHYs		
			LE 1M	LE 2M	LE Coded
0000b	ADV_IND	Primary Advertising	•		
0001b	ADV_DIRECT_IND	Primary Advertising	•		
0010b	ADV_NONCONN_IND	Primary Advertising	•		
0011b	SCAN_REQ	Primary Advertising	•		
	AUX_SCAN_REQ	Secondary Advertising	•	•	•
0100b	SCAN_RSP	Primary Advertising	•		
0101b	CONNECT_IND	Primary Advertising	•		
	AUX_CONNECT_REQ	Secondary Advertising	•	•	•
0110b	ADV_SCAN_IND	Primary Advertising	•		
0111b	ADV_EXT_IND	Primary Advertising	•		•
	AUX_ADV_IND	Secondary Advertising	•	•	•
	AUX_SCAN_RSP	Secondary Advertising	•	•	•
	AUX_SYNC_IND	Secondary Advertising	•	•	•
	AUX_CHAIN_IND	Secondary Advertising	•	•	•
1000b	AUX_CONNECT_RSP	Secondary Advertising	•	•	•
All other values	Reserved for Future Use				

Figure 3. Different PDU Types

1.4. CRC

The three-Octet bit CRC is calculated from PDU to perform error detection and error correction [3]. It is checked by the packet disassembler. If the CRC sequence is correct, then the data is processed. Otherwise, the data is rejected. Refer to the spec 5.0, "The CRC shift register shall be preset with 0x555555 for every LE test packet" and the input should be header, advertising address and payloads. Fig. 4 illustrates how the linear feedback shift register (LFSR) is initialized.

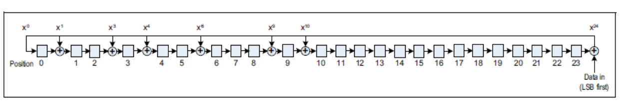

Figure 4. The Initialization of CRC LFSR

2. Background

In this section, we briefly discuss the functionality of each module in a BLE baseband. The BLE baseband includes two major blocks: packet assembler (PA) and disassembler (PDA), which are responsible for TX and RX sides respectively. Two submodules, CRC and (de)whitening, are attached to PA/PDA to follow Bluetooth Specification v5.0.

A packet assembler collects data and forms the packet for transmission [4]. A packet dissembler utilizes CRC module and dewhitening module to check the correctness of the packet [4]. The packet will be processed if the packet is correct; otherwise, the packet will be rejected.

The cyclic redundancy check (CRC) is used for error detection and error correction according to the 24-bit CRC sequence in a BLE packet. The CRC sequence is generated according to PDU when data is transmitted, and is checked by the packet disassembler. A whitening module is used to prevent a long sequence of zeros or ones. It is similar to a CRC despite some subtle difference.

To simplify the testing process, we connected the packet assembler and the packet dissembler separately to RISC-V RocketChip. Then, we connected both the packet assembler and the packet dissembler together,forming a loop, to RISC-V Rocketchip to test the functionality. Section 3 will discuss in details how this process goes.

3. Proposed Algorithm

In order to connect the packet assembler and the packet dissembler to RISC-V RocketChip, we have to write data into both modules, in which case we decided to add the FIFO. We first connected the packet assembler and the packet dissembler separately to the RocketChip by building a packet assembler chain (PA Chain), and a packet dissembler chain (PDA Chain). Each chain contains one module (either a packet assembler or a packet disassembler), one input fifo that writes data into the module (called Writequeue), and one output fifo that reads data from the module (called Readqueue). For the convenience of testing, we then connect two chains. To do so, we combined the output fifo of the PA Chain and the input fifo of the PDA Chain to form one single fifo, called Transitionqueue. This entire structure is called Loop.

3.1. PA Chain

We integrated a complete chain for PacketAssembler testing. PA Chain connects a packet assembler to Rocketchip. By using C code, BLE packet is written into a FIFO bundle by bundle. The PA bundle is transmitting by AXI4StreamNode. We made diplomatic TL node for regmap and used WriteQueue/ReadQueue to access the testing bundle. Then, the FIFO is connected to the packet assembler, and the other side of the packet assembler is connected to another FIFO, which serves as a purpose of checking the result. The diagram of PA chain is illustrated in Fig. 5.

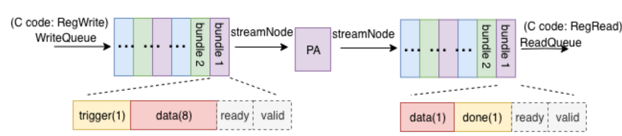

Figure 5. PA Chain Structure

3.2. PDA Chain

PDA Chain connects a packet disassembler back to Rocketchip. By using C code, the output of PA chain is sent bundle by bundle to a FIFO. Then, the FIFO is connected to the packet disassembler. The other side of the packet disassembler is connected to another FIFO, which serves as a purpose of checking the result. The process is quite similar to PA chain. The diagram of PDA chain is illustrated in Fig. 6.

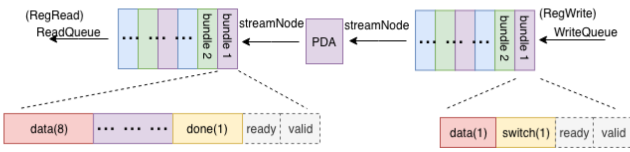

Figure 6. PDA Chain Structure

3.3. Loop

In order to fully verified the functionality of PA Chain and PDA Chain, we connected the output of PA Chain with the input of PDA Chain. The input of the PA Chain and the output of the PDA Chain should be the same. For future work, the loop can be broken, and be connected to other modules (like analog and RF circuits). The diagram of loop is illustrated in Fig. 7.

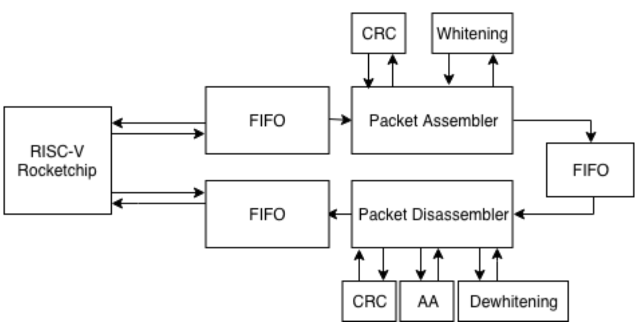

Figure 7. Loopback Chain Structure

4. Experimental Results

In this section, we will elaborate how to perform top level C test for PA Chain, PDA Chain, and Loop. Some of the test results will be given as well. Here is the procedure to perform top level test. For the detailed procedure, please refer to the Github repository.

1. Build the project in verisim folder and modify Makefile to point at the corresponding module.

2. Modify Makefile in tests folder to point at the corresponding module.

3. Apply the Testharness and prepare executable file for C tester by typing "make debug" in verisim folder.

4. A waveform can also be generated.

4.1. PA Chain Results

The top level test result of the packet assembler chain is displayed in Fig. 8. PA packs the contents of BLE packet byte by byte, forming the input bundle and yields the output. The zero-one sequence below the "pack data: 00000043" is the data field of the output. Here the latter part is not shown. For the full sequence, please refer to the repo.

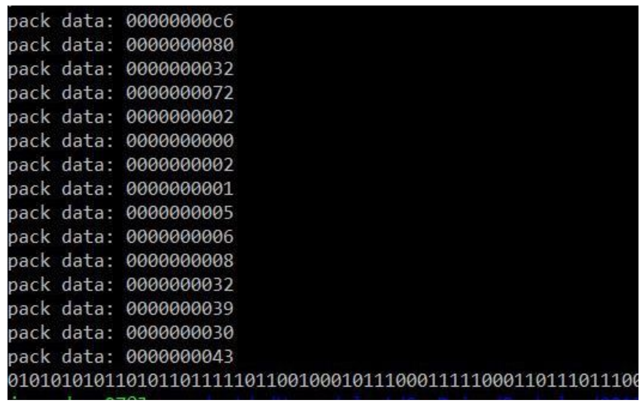

Figure 8. PA Chain Result

4.2. PDA Chain Results

Packet DisAssembler unpacks the data and checks the CRC. The top level test result is shown below. The right figure shows the test case when the CRC does not match (CRC invalid) and the packet would be rejected. Here also the unpack data is not fully shown.

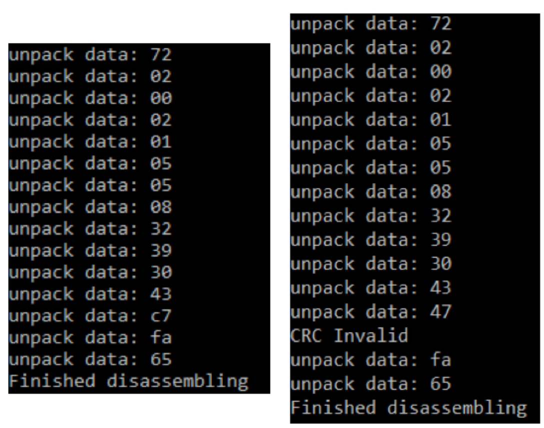

Figure 9. PDA Chain Result

4.3. Loop Results

The top level test result is shown below. This test takes string "UCBerkeley" as the payload data. You could observe that the last ten "pack data" is 0x55, 0x43, 0x42, 0x65, 0x72, 0x6b, 0x65, 0x6c, 0x65, 0x79, which are the ASCII representation of "U", "C", "B", "e", "r", "k", "e", "l", "e", "y" respectively. The expected successful results should be that the unpack data at the end of the loop chain exactly matches the pack data at the beginning of the loop chain.

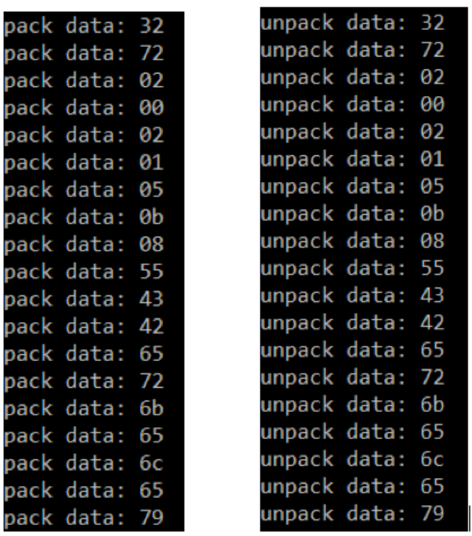

Figure 10. Loopback Chain Result

5. Conclusion

In this paper, we proposed a fast and easy way of testing BLE baseband. Instead of using Direct Memory Access, We connected the BLE baseband to RISC-V RocketChip. We built Packet Assembler Chian and Packet Disassembler Chain to test if both modules can work with RISC-V RocketChip independently. After verifying that both chains can work, we constructed a loop that connects both chains together. The main idea of this new architectures is to use the FIFO. In order to improve performance as well as efficiency, we limited the width of FIFO, so that our architecture is area-efficient. Experimental results have verified the functionality of our algorithm.

Future improvement includes connecting the packet assembler chain and the packet disassembler chain to RF design, which requires a specific frequency that is or is not the same as the frequency of the CPU. More add-ons may include additional functional blocks, such as FEC, as specified by Bluetooth 5 spec.

5.1. Acknowledgement

Here is our appreciation to Prof. Borivoje Nikolic, Prof. Kristofer Pister and the GSI Paul Rigge for guiding us in this project. Their valuable suggestions and feedback help us move forward. Also the work from last semester's group inspired us greatly and here is their tapeout (https://github.com/tapeout/ble-baseband). Lastly, we would like to thank David Burnett and Rachel Zoll for helping us get on board and explain the former BLE stucture and tests.

References

[1] M. Hughes, "What is Bluetooth 5? Learn about the Bit Paths Behind the New BLE Standard", *All About Circuit*, July. 2017.

[2] Bluetooth, "Bluetooth Core Specification V5.0", vol.6, Dec. 2016.

[3] Z. Gao, "CRC and Whitening", *UCB EE290C Spring 2018*, May. 2018.

[4] R. Renn, "Packet Assembler and Disassembler Final Report", *UCB EE290C Spring 2018*, May. 2018.