RZ 3534 (# 99547) 01/26/04
Electrical Engineering 62 pages

Research Report

The ZRL High-Speed Wireless LAN Testbed: OFDM Physical Layer Architecture and Implementation

Simeon Furrer, Jens Jelitto, Wolfgang Schott, and Beat Weiss

IBM Research GmbH
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

The ZRL High-Speed Wireless LAN Testbed:

OFDM Physical Layer Architecture and Implementation

Version 1.0
01/22/2004

Simeon Furrer, Jens Jelitto, Wolfgang Schott, and Beat Weiss

IBM Zurich Research Laboratory, 8803 Rüschlikon

Abstract

A prototype of a broadband wireless LAN with three mobile stations has been designed and built at the IBM Zurich Research Laboratory. Each station operates in conformance with the IEEE 802.11a WLAN standard and comprises a 5-GHz radio frontend realized with analog vendor components, a digital baseband implemented in a FPGA, and a medium-access control executed on an ARM9 embedded processor. This document describes the architectural design of the OFDM-based physical layer and provides details on the implementation of the analog radio frontend, the digital baseband and its interface to the MAC.

1 Introduction

In the next generation of IP-based mobile communication systems, broadband radio LANs will enable a variety of new wireless services and applications in the office, at home, and on the move. High-speed wireless communication between mobile and control stations will be required to bring the tremendous amount of information provided by the World Wide Web to the mobile user. In addition, the provision of high-speed data services as well as multi-media entertainment applications in hot-spot cells, such as conference centers, airports, hotel lobbies, and similar public places, will continuously gain in importance.

In the United States, broadband radio LANs have been specified in the IEEE 802.11a standardization group [1]. The 802.11a wireless LAN will satisfy the increasing bandwidth demand by providing data rates up to 54 Mbit/s. The radio is operated in frequency bands between 5.1 and 5.8 GHz. The physical layer transmission scheme applies Orthogonal Frequency Division Multiplexing (OFDM) with variable-rate modulation and coding, allowing selectable user data rates between 6 and 54 Mbit/s. In OFDM-based systems [2], the user data stream is split into parallel streams of reduced rate, which are then modulated on separate subcarriers. By appropriately choosing the frequency spacing between subcarriers, the carriers are made orthogonal, allowing some spectral overlap between the subchannels, which leads to a better spectral efficiency than using simple frequency-division multiplexing. OFDM is especially attractive for high-speed wireless LANs because it is robust against multi-path propagation, intersymbol interference, and against narrowband interference. Access to the radio channel by multiple users is controlled by a Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA) scheme as defined in the 802.11 Medium Access Control (MAC) protocol [3]. This Ethernet-like protocol allows an easy integration of the radio LAN into an IP-based backbone network.

We have designed and implemented an 802.11a WLAN testbed at the IBM Zurich Research Laboratory (ZRL). The testbed consists of three standard compliant mobile station prototypes operated without centralized control (ad-hoc mode). Each mobile station comprises a radio frontend, a digital baseband/MAC unit, and an interface to a host computer as illustrated in Figure 1.1.

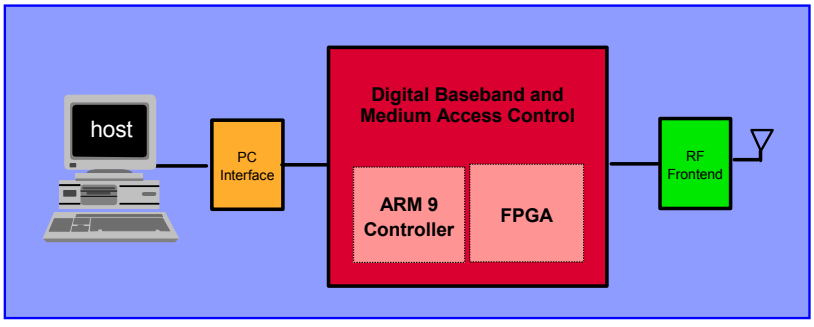


Figure 1.1: 802.11a WLAN mobile station prototype (overview)

This report describes the OFDM-based physical layer (PHY) design and implementation of the mobile station prototype of the high-speed ZRL WLAN testbed. Section 2 presents the overall architecture of the mobile station. We also describe the partitioning of the prototype into several functional units and their mapping onto commercially available hardware and software components. In Section 3, we focus on the physical layer architecture. The interface between the physical layer and the MAC is specified, and basic PHY procedures are discussed. Section 4 provides details on the design and implementation of all signal-processing units required to implement the transmit (TX) and receive (RX) functions of the digital baseband. Finally, Section 6 deals with various implementation aspects of the radio frontend.

2 WLAN Mobile Station Architecture

The ZRL high-speed WLAN prototype consists of three 802.11a-conform mobile stations. Each station has been implemented with a radio frontend, digital baseband/MAC unit, and peripheral personal computers, as indicated in Figure 2.1.

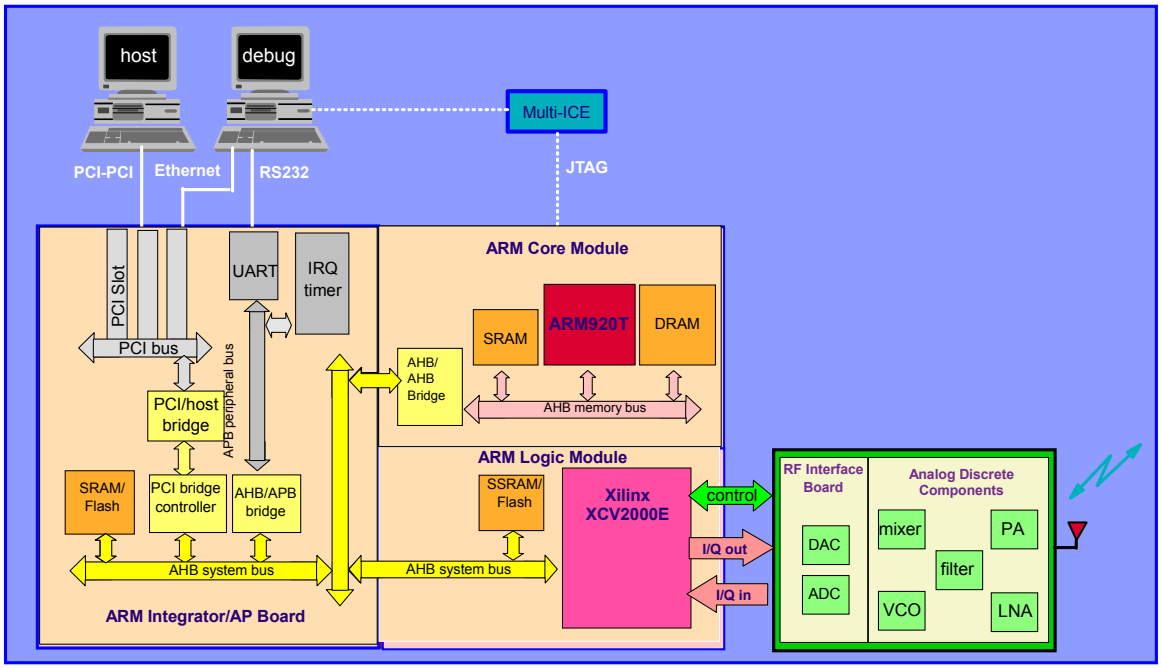


Figure 2.1: 802.11a WLAN mobile station prototype

A 5-GHz radio frontend has been designed with discrete vendor components. It has been implemented on two boards: one comprises all analog components required such as low-noise amplifier (LNA), power amplifier (PA), synthesizer, mixer, and filters, while the other mainly carries the digital-to-analog (D/A) and analog-to-digital (A/D) converters. The frontend is attached to ARM development boards via two digital data I/O interfaces and one control interface. The latter can be used to program the frontend so that the radio signals are emitted in one of the eight pre-defined channels in the lower/middle U-NII frequency band (see Figure 2.2).

The digital baseband/MAC unit has been implemented on three ARM development boards, namely the integrator/AP board, the core module CM920T-ETM, and the logic module LM-XCV2000E [4].

The core module executes the MAC protocol firmware on the real-time operating system VxWorks on an ARM920T embedded controller, which can access, via a memory bus, local memory that stores the 802.11 MAC firmware, and via a bridge the system bus to communicate with components located on the integrator board and logic module.

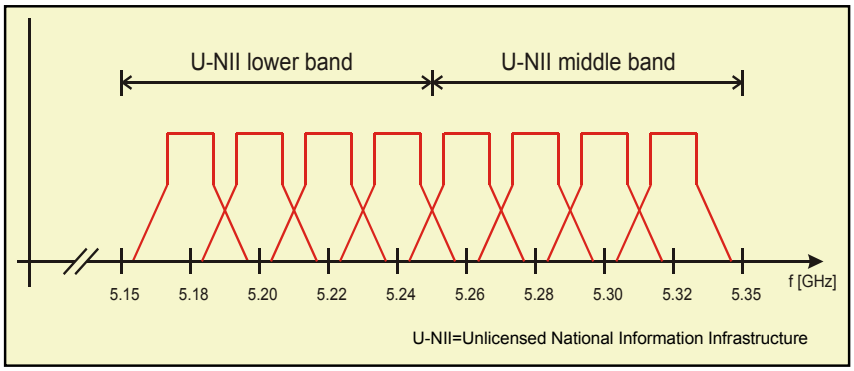


Figure 2.2: Frequency channels of 5 GHz radio frontend

The logic module mainly carries a XILINX XCV2000E FPGA. On this device, digital signal-processing functions are implemented that are required to transmit and receive OFDM frames over the air interface, to communicate asynchronously via the AHB system bus with the MAC, and to control the radio frontend. More details on the physical layer functions implemented in the FPGA are given in Figure 2.3.

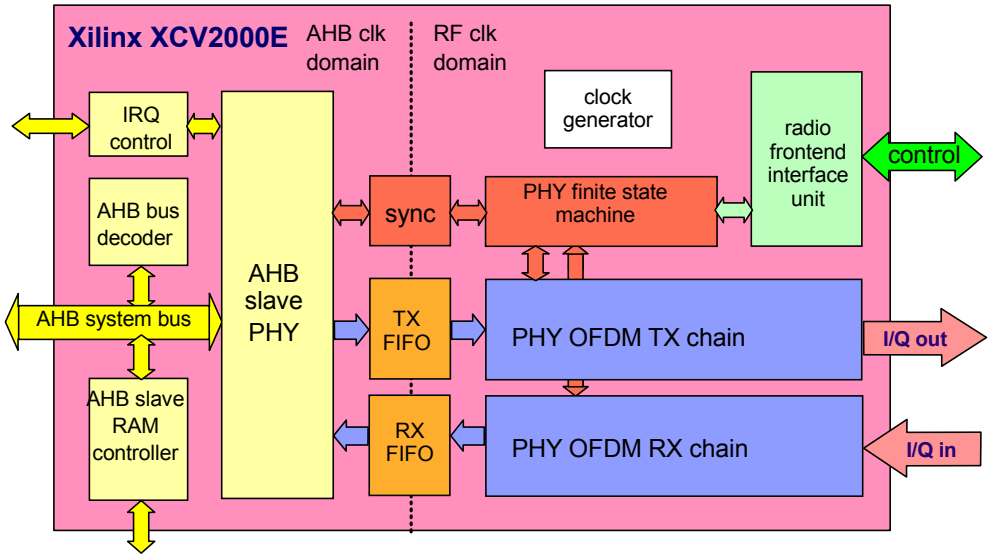


Figure 2.3: Physical layer functions implemented in FPGA

The integrator board mainly provides I/O and control functions to attach peripherals to the AHB system bus. The ARM development boards have been connected via an Ethernet card, a serial link, and a JTAG interface to a computer for downloading the MAC code to the memory, to program the FPGA, and to debug the prototype. Typical wireless applications can be executed on a host computer that can be connected to the integrator board via a PCI-PCI bridge.

3 Physical Layer Architecture

The physical layer has been architected so that it complies with the 802.11a standard and can be efficiently implemented with commercially available hardware components. In this Section, we give an overview on the PHY architecture selected, specify the PHY/MAC interface with abstract service primitives, and describe the behavior of the physical layer by means of the receive and transmit procedures.

3.1 Overview

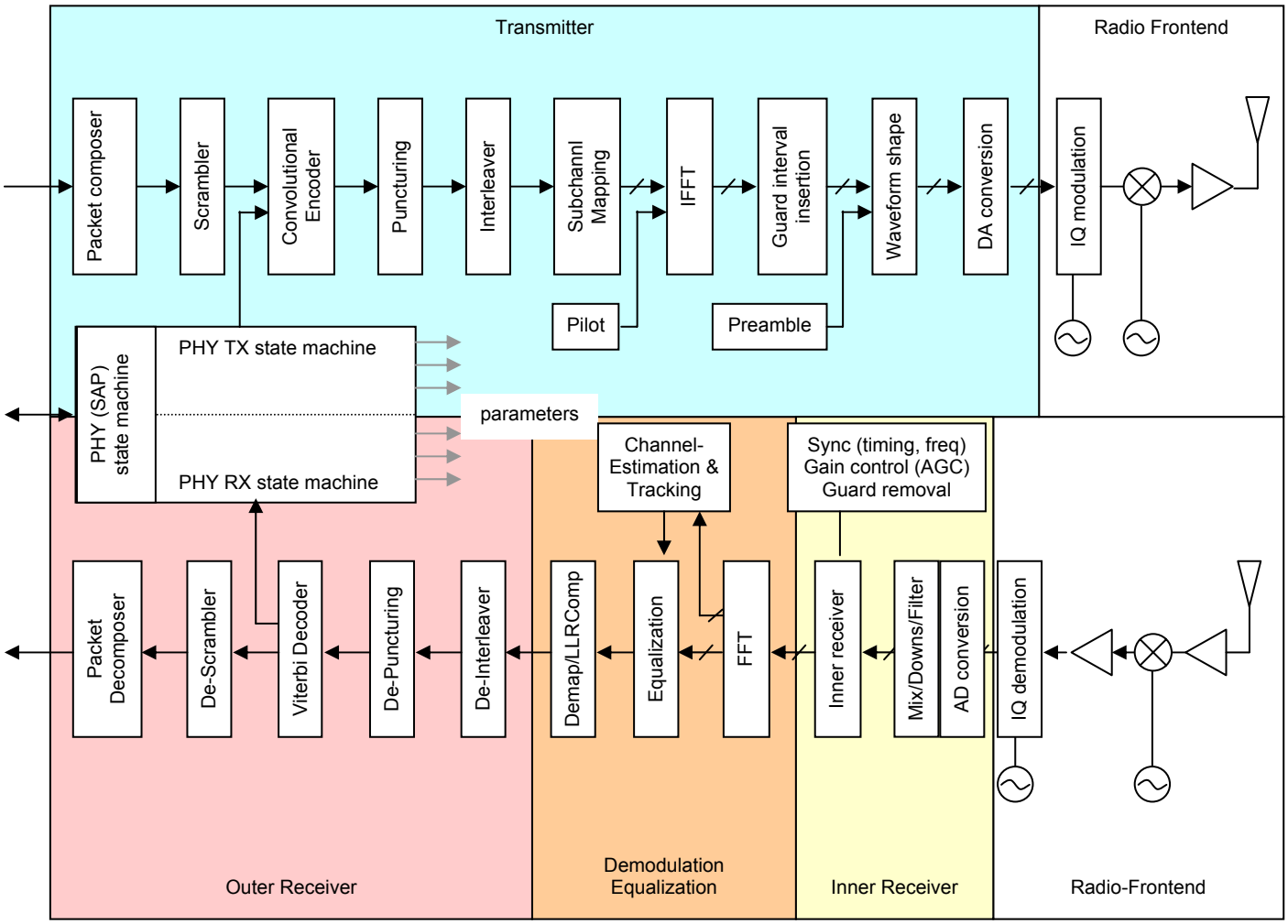


Figure 3.1: Physical layer functional architecture

Figure 3.1 shows an overview of the functional architecture of the physical layer, which consists of the digital baseband and the radio fontend. The digital baseband comprises the TX chain, the RX chain with demodulation/equalization unit and the inner/outer receiver, and a finite state machine (FSM) controlling the operation of the

physical layer. These functions are implemented in the FPGA on the logic module as illustrated in Figure 2.3.

To facilitate the understanding of the subsequent sections, a high-level overview on the functions implemented in the PHY TX and RX chain is given. Some technically important details are skipped on purpose.

When the MAC executed on the ARM controller requests the transmission of a PHY Service Data Unit (PSDU) with a given data rate and length, the payload and physical layer control information are clocked via the AHB system bus into the TX FIFO buffer and control registers (see Figure 2.3). After all control fields have been written, the physical layer FSM activates, configures, and triggers the OFDM TX chain. The data provided by the MAC is clocked from the TX FIFO into the TX chain. The packet composer maps the PSDU into a PHY Protocol Data Unit (PPDU), which also comprises the PHY Convergence Protocol (PLCP) header with the DATARATE and PSDU LENGTH field. The data field of the PPDU is scrambled, encoded, punctured, and interleaved (see Figure 3.1). The data symbols are then mapped on complex symbols according to the modulation scheme chosen before they are modulated on subcarrriers. Some carriers are skipped for subsequently inserting pilot signals. The modulation can be implemented efficiently by performing a 64-point Inverse Fast Fourier Transform (IFFT). The number of samples at the output of the IFFT is further increased to 80 by inserting a guard interval (cyclic extension). The samples are then parallel-to-serial converted and a PLCP preamble is put in front for receiver training. Filtering can then be applied to shape the waveform. Next, the preamble and samples are D/A-converted, I/Q-modulated, up-converted to the 5-GHz frequency band, amplified and finally transmitted over the air. Details on the implementation of the functional units of the TX chain are given in Section 4.2.

When a radio signal is received at the antenna, it is amplified with an LNA, down-converted to an intermediate frequency band, I/Q-demodulated, and A/D-converted. The analog signal is over-sampled by a factor of two. Filtering and down-sampling to the baseband are performed before the digital samples are fed to the inner receiver that exploits known features of the PLCP preamble for detecting signal reception, adjusting the gain of the received signal, compensating frequency offsets, and removing the guard interval. After serial-to-parallel conversion, the 64 subcarriers of the OFDM signal are demodulated by using a Fast Fourier Transform (FFT). Channel estimation and equalization are then performed to remove some adverse effects of the radio channel. To apply soft-decision Viterbi decoding for estimating the transmitted data sequence, the input samples to the Viterbi decoder are pre-processed in the Log-Likelihood Ratio (LLR) computation, de-interleaving, and de-puncturing unit. After de-scrambling the decoded data, the packet decomposer reconstructs the likely transmitted PSDU, stores it into the RX FIFO, and sends an interrupt to the ARM controller (see Figure 2.3). The interrupt triggers the execution of an interrupt service routine that immediately passes the received packet and additional control data via the system bus to the MAC protocol. Details on the implementation of the functional units of the RX chain are given in Section 4.3.

3.2 Physical Layer Service Primitives

The physical layer provides a set of services to the MAC layer, which are abstractly described by means of PHY service primitives in the IEEE 802.11 specification [1, 3]. Most of these services are implemented in the prototype in accordance with the standard and are provided to the MAC at the PHY service access point (SAP). This Section documents the implementation differences compared with the IEEE specification.

PHY_TXSTART.req(E_TX_VECTOR, TIMEPOINT)

The primitive PHY_TXSTART.req is a request by the MAC to the PHY to start a packet transmission. The parameter list E_TX_VECTOR is an extended version of the TX_VECTOR as defined in [1] and includes all information necessary for a packet transmission. The parameters are defined in Table 3.1. A packet transmission is started at the point in time indicated by the parameter TIMEPOINT (see Table 3.2).

Table 3.1: E_TX_VECTOR parameters

Parameter	Type	Explanation
LENGTH	INTEGER	PSDU length (including CRC): 0-4095 (12 bits)
DATARATE	CHAR	Data rate: 6,9,12,18,24,36,48,54 Mbit/s (see encoding rule in [1])
SERVICE	BITSTRING	Service field (16 bits)
TXPWR_LEVEL	INTEGER	Transmit power level: 1-8 (4 bits)
PSDU_HEADER	CHARSTRING	PSDU header
PSDU_PAYLOAD_POINTER	CHARSTRING_PTR	Pointer to PSDU payload
PSDU_PAYLOAD_LENGTH	INTEGER	PSDU payload length

Table 3.2: TIMEPOINT parameter

Parameter	Type	Explanation
TIMEPOINT	UINT32	Point in time (time-stamp): 0-2^31-1 (32 bit) TIMEPOINT=0; immediate execution

Implementation notes: The parameters LENGTH, DATARATE, SERVICE, and TXPWR_LEVEL are written to dedicated control registers in the FPGA, whereas the strings PSDU_HEADER and PSDU_PAYLOAD are consecutively written into the TX_FIFO that has been flushed in advance. The parameter TIMEPOINT has to be written to a dedicated control register in the FPGA after all other parameters have been written.

PHY_TXSTART.conf(TIMEPOINT)

The primitive PHY_TXSTART.conf confirms the reception of a primitive PHY_TXSTART.req. The parameter TIMEPOINT is a time-stamp.

Implementation notes: The primitive is implemented with the interrupt EXP1 and the interrupt service routine (ISR) intExp1SvcRou.

PHY_TXEND.req(TIMEPOINT)

The primitive PHY_TXEND.req can be used by the MAC to prematurely terminate a packet transmission. The parameter TIMEPOINT is optional and is set by default to 0, indicating immediate execution. All pending PHY_TXSTART.req's are disabled if a PHY_TXEND.req is issued.

Implementation notes: The function is not supported in the prototype.

PHY_TXEND.conf(TIMEPOINT)

The primitive PHY_TXEND.conf confirms the reception of a primitive PHY_TXEND.req. The parameter TIMEPOINT is a time-stamp.

Implementation notes: The function is not supported in the prototype.

PHY_TXEND.ind(TIMEPOINT)

The primitive PHY_TXEND.ind indicates the end of a successful PHY packet transmission. The parameter TIMEPOINT is a time-stamp. This primitive is not standardized.

Implementation notes: The primitive is implemented with the interrupt EXP3 and the ISR intExp3SvcRou.

PHY_CCARESET.req(TIMEPOINT)

The primitive PHY_CCARESET.req requests the PHY to reset the clear channel assessment (CCA) finite-state machine. The parameter TIMEPOINT is optional and set by default to 0, indicating immediate execution.

PHY_CCA.ind(STATUS, TIMEPOINT)

The primitive PHY_CCA.ind indicates an activity change on the wireless medium to the MAC. The STATUS parameter is derived from the received signal strength indicator (RSSI). Details are shown in Table 3.3. The parameter TIMEPOINT is a time-stamp.

Table 3.3: STATUS parameter

Parameter	Type	Explanation
STATUS	CHAR	Status = 1: Indicates that the state of the medium changed to BUSY (RSSI >= threshold). Status = 0: Indicates that the state of the medium changed to IDLE (RSSI <= threshold).

Implementation notes: The primitive is implemented with the interrupt EXP0 and the ISR intExp0SvcRou. The ISR reads the STATUS parameter from a dedicated control register implemented on the FPGA.

PHY_RXSTART.ind(E_RXSTART_VECTOR, TIMEPOINT)

The primitive PHY_RXSTART.ind indicates successful reception of the PLCP header to the MAC. The parameter list E_RXSTART_VECTOR is an extended version of the RX_VECTOR as defined in [1] and includes all necessary information for packet reception. The parameters are defined in Table 3.4. The parameter TIME_POINT is a time-stamp.

Table 3.4: E_RXSTART_VECTOR parameters

Parameter	Type	Explanation
LENGTH	INTEGER	PSDU length (including CRC): 0-4095 (12 bits)
RSSI	INTEGER	RSSI value
DATARATE	INTEGER	Data rate: 6,9,12,18,24,36,48,54 Mbit/s (see encoding rule in [1])
SERVICE	BITSTRING	Service field: (16 bits)

Implementation notes: The primitive is implemented with the interrupt EXP1 and the ISR intExp1SvcRou. The ISR reads all parameters from dedicated control registers implemented on the FPGA.

PHY_RXEND.ind(E_RXEND_VECTOR, TIMEPOINT)

The primitive PHY_RXEND.ind indicates end of a PSDU reception. The parameter E_RXEND_VECTOR associated with this primitive includes the parameters listed in Table 3.5. The parameter TIMEPOINT is a time-stamp.

Table 3.5: E_RXEND_VECTOR parameters

Parameter	Type	Explanation
RXERROR	CHAR	NoError (=0): Complete PSDU successfully received
		CarrierLost (=1): A change of the RSSI caused the CCA status to return to IDLE before complete PSDU reception
		UnsupportedRate (=2): Rate indicated in SIGNAL field is not supported
		FormatViolation (=3): PLCP header can be received, but parity check of the PLCP header is invalid
CRCERROR	CHAR	NoError (=0): CRC check successful
		Error (=1): CRC check failed
PSDU_HEADER	CHARSTRING	PSDU header
PSDU_PAYLOAD_PTR	CHARSTRING_PTR	Pointer to PSDU payload
PSDU_PAYLOAD_LENGTH	INTEGER	PSDU payload length

Implementation notes: The primitive is implemented with the interrupt EXP2 and the ISR intExp2SvcRou. The routine requires a successful generation of the corresponding PHY_RXSTART.ind message in advance. The RXERROR and CRCERROR parameters are read from dedicated control registers on the FPGA. The PSDU header and payload strings are generated by iteratively reading the RX_FIFO, and subsequently dividing the string into header and payload according to the known lengths of header and payload. The RX_FIFO is flushed by the ISR after its contents has been read.

3.3 Physical Layer Procedures

The behavior of the physical layer is controlled by the PHY top-level FSM depicted in Figure 3.2. After switching the mobile station on or upon receiving a PHY_RESET.req from the MAC, the physical layer is initialized and switched to the receive mode indicated by the state s_PHYrx. A PHY_TXSTART.req issued by the MAC causes a transition of the FSM to the state s_PHYtx. The transmit mode is terminated if the physical layer signal sig_txend (end of transmission) occurs or a PHY_TXEND.req is issued to stop transmission prematurely. In both cases, the FSM returns to the state s_PHYrx to be prepared for receiving new packets over the radio channel.

The PHY top-level FSM can be further specified by the clear channel assessment (CCA), transmit (TX), and receive (RX) state machines. These FSMs define all details of

the PHY CCA, PHY TX, and PHY RX procedures; in the sequel, we will describe them verbally.

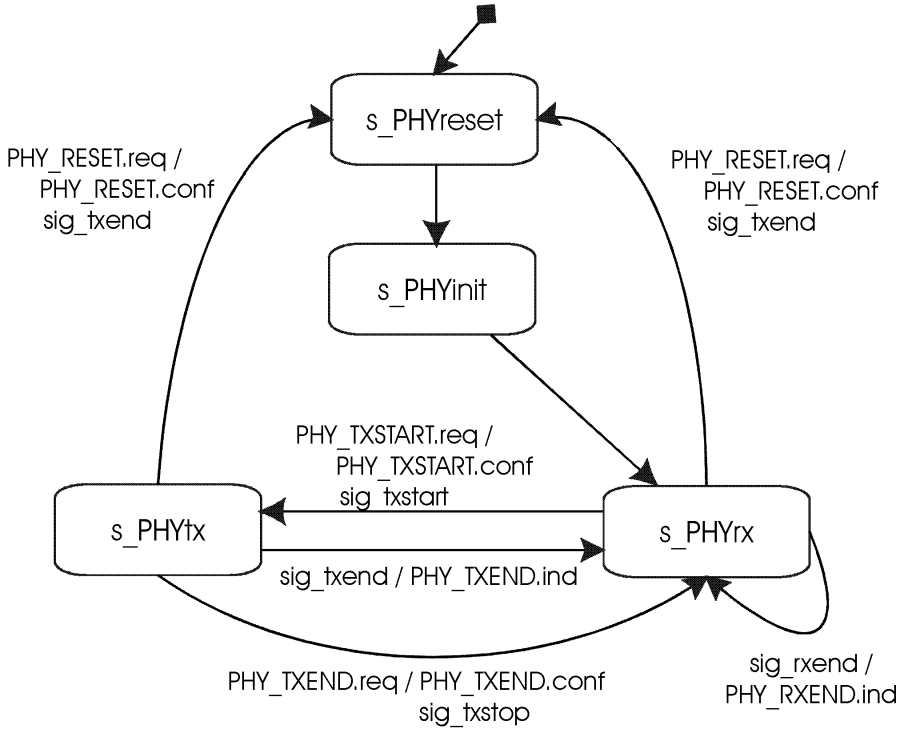


Figure 3.2: Physical layer top-level finite-state machine

The PHY CCA procedure is executed while the receiver is turned on. Based on received signal strength measurements performed by the radio frontend, the procedure is used to determine whether the status of the radio channel is IDLE or BUSY. The CCA FSM comprises two main states s_CCAidle and s_CCAbusy, indicating the last monitored state of the radio channel. If the FSM toggles to a new state, the status of the channel is reported with a PHY_CCA.ind to the MAC. The current channel status can also be obtained by issuing a PHY_CCARESET.req.

The PHY TX procedure is illustrated in Figure 3.3. After the MAC has received a clear-channel indication PHY_CCA.ind(IDLE), it can invoke the transmit procedure at any time by issuing a PHY_TXSTART.req(E_TX_VECTOR, TIMEPOINT). Upon receiving this primitive, the physical layer is switched in the time interval RX/TX from the receive to the transmit mode, and is configured according to the E_TX_VECTOR elements TXPWR_LEVEL and DATARATE. After switching, a PHY_TXSTART.conf message is sent to the MAC, confirming the readiness of the TX chain.

At the time indicated by the parameter TIMEPOINT, the PHY starts transmitting the PLCP preamble. Simultaneously, the MAC is informed with a PHY_CCA.ind message about the change of the channel-medium status to BUSY. The generation of the PLCP

header has to be started just in time to ensure that it can be seamlessly appended to the preamble. The PLCP header contains the fields DATARATE (R), RESERVED (R), LENGTH (L), parity (P), tail (T), and SERVICE (S), with the parity and tail bits being generated in the PHY. All other parameters are elements of the E_TX_VECTOR. The PLCP header is followed by the PHY service data unit (PSDU), comprising a PSDU header and payload. It is extended with six tail bits, which are all set to zero. The extended PSDU as well as the service field are scrambled and encoded in accordance with the requested parameter DATARATE. To fit the number of coded bits into an integer number of OFDM symbols, some PAD bits may be appended to the encoded service data unit (C-PDSU).

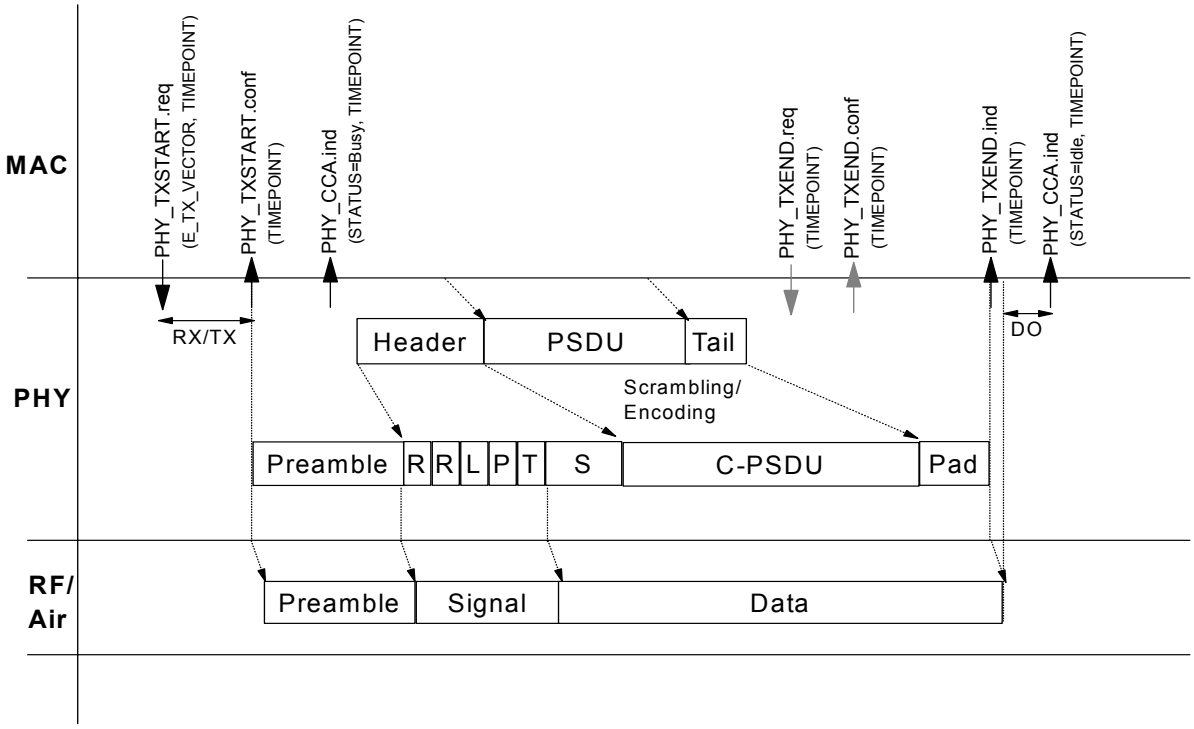


Figure 3.3: Physical layer transmit procedure

The mapping of the PHY information fields to OFDM symbols is shown in the lower part of Figure 3.3 and in more detail in Figure 3.4. The preamble consists of two 8-µs-long frames. In the first frame, a training symbol of duration 0.8 µs is repeatedly transmitted 10 times, while the second frame of the preamble consists of a 1.6-µs-long guard interval (GI2) followed by two 3.2-µs-long training symbols. Both frames are mainly used for signal detection, VGA gain adjustment, and for acquiring an initial, coarse estimate of the frequency offset value at the receiver. After the preamble, the 4-µs-long SIGNAL symbol and DATA symbols are transmitted. The SIGNAL symbol carries the PLCP header without SERVICE field. It is transmitted with the most robust combination of modulation and coding (6 Mbit/s). The 4-µs-long DATA symbols carry the SERVICE field and the C-PSDU with PAD bits. The SIGNAL and each DATA symbol also comprise a short guard interval (GI) of 0.8 µs in which a cyclic extension of

the symbol is transmitted to eliminate intersymbol and intercarrier interference caused by multi-path propagation.

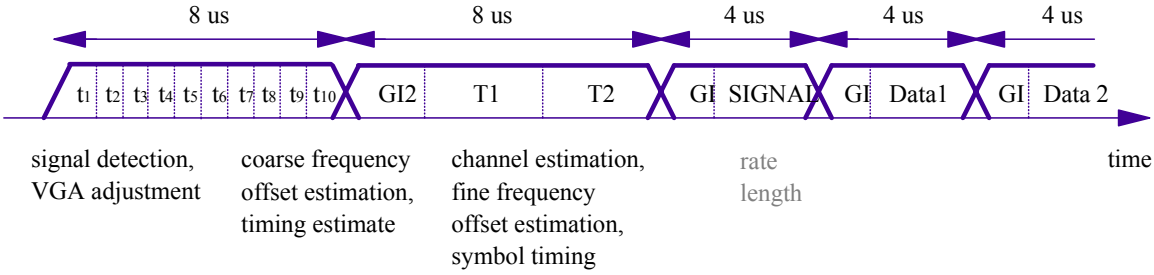


Figure 3.4: Training and data symbols of physical layer

The successful transmission of the packet causes a transition to the state s_PHYrx in the PHY top-level FSM, which is indicated with a PHY_TXEND.ind. After a time interval D0, a PHY_CCA.ind(IDLE) message reports to the MAC that the PHY is ready to accept a new PHY_TXSTART.req. The MAC can always prematurely terminate the TX procedure by sending a PHY_TXEND.req(TIMEPOINT).

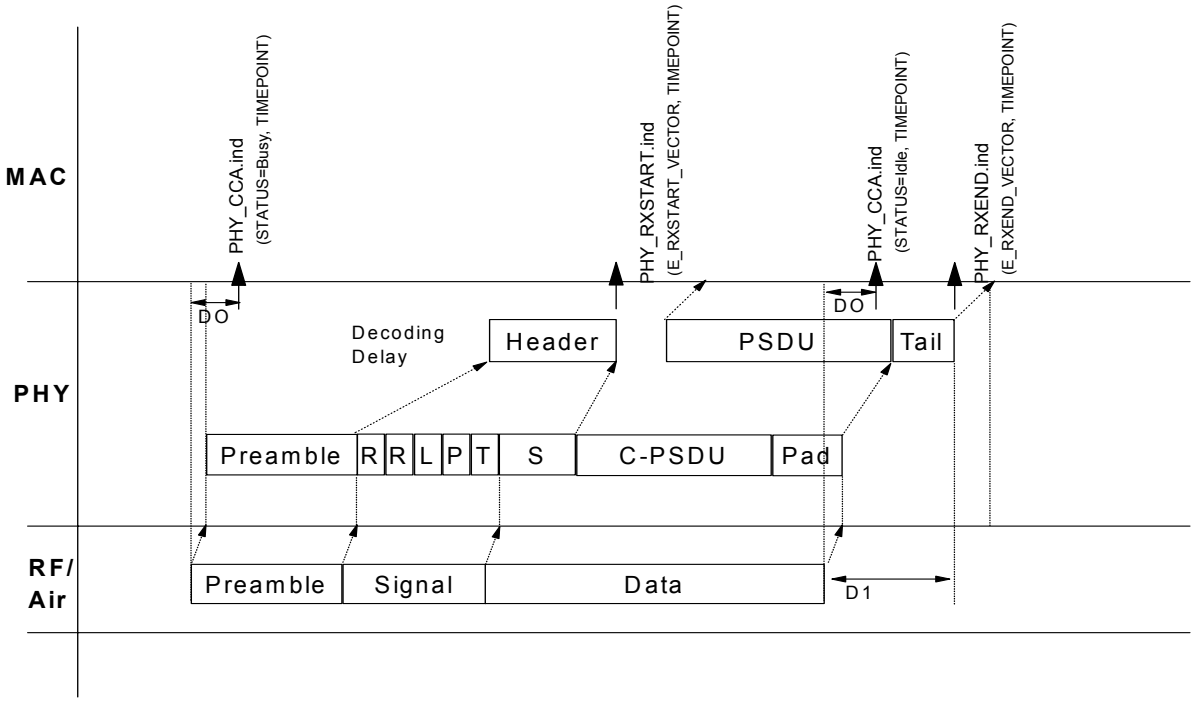


Figure 3.5: Physical layer receive procedure

The PHY receive procedure is shown in Figure 3.5. When the PHY top-level FSM is in the state s_PHYrx, the receive procedure is invoked upon detecting the reception of a portion of the PLCP preamble followed by the PLCP header. In the case of receiving the preamble, the radio frontend first reports a significant RSSI value change to the CCA

FSM. As a consequence, a PHY_CCA.ind(BUSY) message indicates the channel-status change to the MAC. In parallel, the received training symbols of the preamble are compared with their known values and differences are compensated by adjusting the gain of the received signal and correcting frequency offsets accordingly. After setting these control parameters, the physical layer starts searching for end-of-preamble, indicating the start of the PLCP header. The SIGNAL symbol is then decoded to determine the length of the data stream to be received as well as the selected modulation type and coding scheme. Once the SIGNAL symbol is decoded without errors, the proper demodulation and decoding scheme is enabled to receive the OFDM data symbols carrying the encoded PSDU header and payload. The first symbol also carries the SERVICE field. If all PLCP header fields have been successfully received, a PHY_RXSTART.ind (E_RXSTART_VECTOR, TIMEPOINT) is issued to the MAC. The parameters associated with this primitive include the length of the PSDU, the RSSI, the data rate, and the service field. By decoding all OFDM data symbols, the PSDU is reconstructed. After removing the tail bits, the PSDU header and payload are transferred with a PHY_RXEND.ind (E_RXEND_VECTOR, TIMEPOINT) message to the MAC. This primitive is also used to report abnormal termination of the receive procedure to the MAC. After the reception of the last OFDM symbol, the status of the radio channel will usually transit to IDLE. Owing to decoding delay in the physical layer, this change will be indicated to the MAC with a PHY_CCA.ind(IDLE) primitive before the PHY_REXEND.ind is issued.

4 Digital Baseband Implementation

The digital baseband of each 802.11a-conform mobile station of the ZRL high-speed wireless LAN testbed has been implemented on a XILINX FPGA. In each prototype implementation, all digital signal-processing functions required to transmit and receive 802.11a OFDM packets over the radio interface have been implemented. To fit all functions into the FPGA and to satisfy stringent timing requirements, each functional unit has been designed with respect to low gate-count and low latency. Moreover, an efficient inter-block communication mechanism has been applied.

This section will first introduce the communication mechanism applied between neighbor functional units in the PHY transmit and receive chain. Afterwards, details on the design and implementation of the various hardware modules will be given. Finally, latency issues will be discussed.

4.1 Inter-Block Decentralized Communication

To keep the control overhead for the large number of signal-processing units in the TX and RX path small, a decentralized control architecture has been chosen. Every signal-processing unit shall be able to work as autonomously as possible and exchange data bits from/to the preceding/subsequent functional units whenever possible. A simple handshake protocol was designed to coordinate the one-way data communication from a first unit, called the “producer,” to a second unit, the “consumer”.

Whenever the producer unit has a valid data on its data output line (DATA), the request line (REQ) is driven high to indicate a data transfer request. The acknowledge (ACK) line is driven high by the consumer unit as soon as data can be accepted. The data transfer is executed on the rising clock edge whenever REQ and ACK are high. This handshake protocol allows zero latency, synchronous data communication at clock speed. In the following, we highlight two templates from which most of the signal-processing units were derived.

The first template is suitable for units that process data serially (or with low latency) and is depicted in Figure 4.1. In normal operation mode, this unit produces a latency of one clock cycle due to the input register (R_i). If the subsequent unit is unable to accept data (ACK_IN low), the unit FSM changes the multiplexer positions to add a temporary register (R_o) storing the current output data. The FSM will drive ACK_OUT low in the next clock cycle and stop the data processing. When ACK_IN reaches high again, register R_o is emptied and “removed”, and the unit continues processing data at the next rising clock edge. (Examples based on template 1: scrambler, convolutional encoder, etc.)

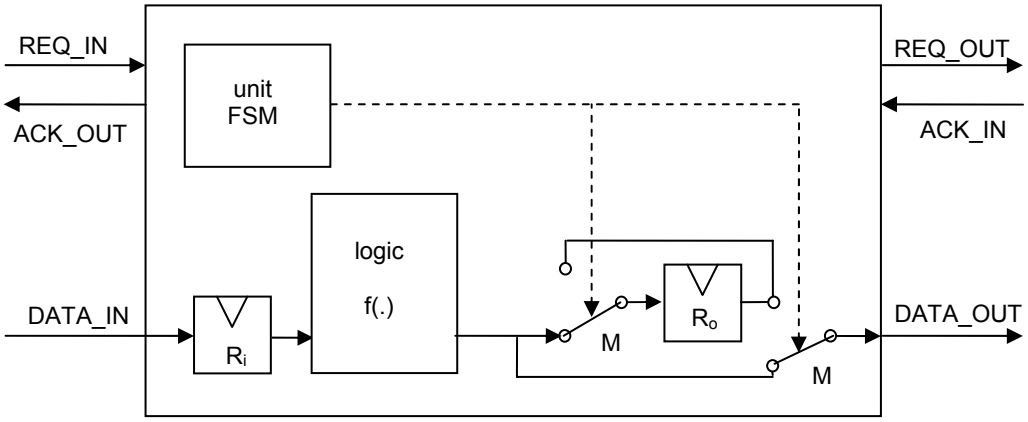


Figure 4.1: Generic signal-processing unit for serial processing ("template 1")

Template 2, shown in Figure 4.2, is appropriate for units working on blocks of data. It implements two data buffers, one connected to the input and the other one to the output. New data can be accepted at the input, processed by $f_{in}(.)$ and stored in the input buffer, while at the same time data from the output buffer can be post-processed by $f_{out}(.)$ and fed to the output. Two state machines, one for the input and one for the output, control the communication to the preceding and subsequent units and coordinate the buffer switching with two multiplexers M1 and M2. The buffers are switched simultaneously if the input buffer is full and the output buffer empty. The unit's latency depends on the time needed to fill/empty the buffers. (Examples based on template 2: Interleaver, FFT, etc.)

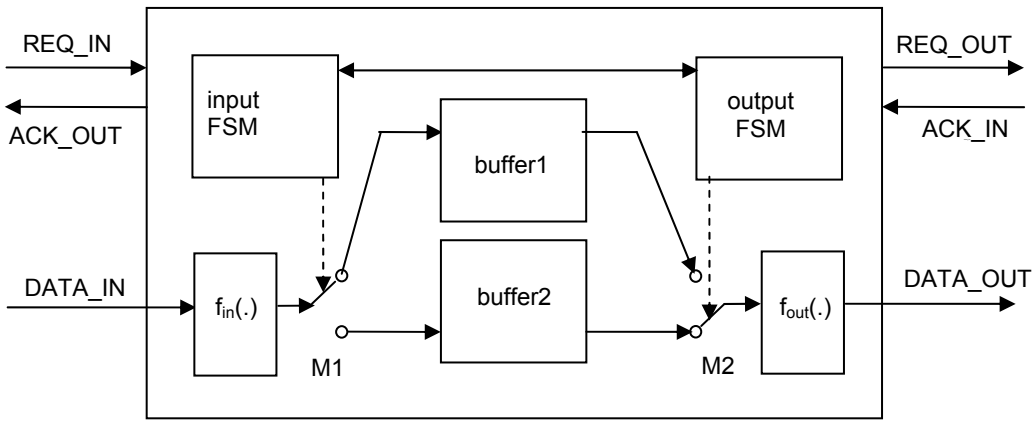


Figure 4.2: Generic signal-processing unit for block processing ("template 2")

4.2 TX Chain Functionality

4.2.1 Overview

The following paragraphs describe the functional units of the transmitter chain, their basic behavior and implementation details. A block diagram of the TX chain is shown in Figure 4.3. Note that always two data bits are clocked into the TX chain in parallel from the TX FIFO and two data bits are clocked from the RX chain into the RX FIFO. All functional units are implemented accordingly to exploit this parallelism, which decreases the required clock rate of the TX/RX chain by a factor of two. The clock data rate was chosen to be 40 MHz.

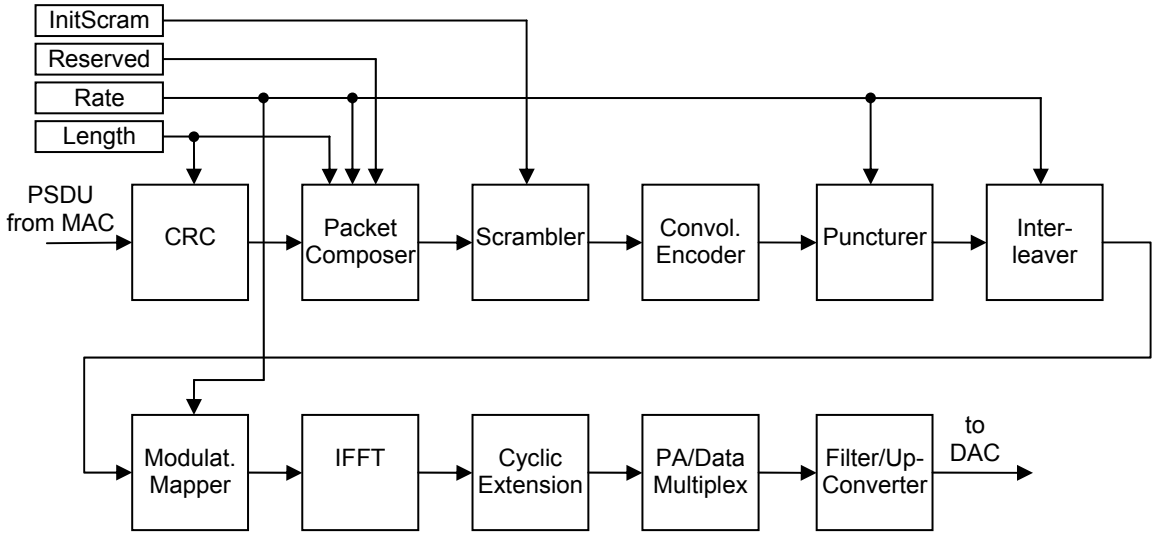


Figure 4.3: TX chain block diagram

4.2.2 CRC Generation

Each MAC frame consists of three basic components [3], as depicted in Figure 4.4:

- MAC header, containing frame control, duration, address and sequence control information;
- frame body of variable length, containing frame-type-specific information, and
- Frame Check Sequence (FCS), containing an IEEE 32-bit cyclic redundancy code (CRC).

Octets:2	2	6	6	6	2	6	0-2312	4
Frame Control	Duration/ ID	Address 1	Address 2	Address 3	Sequence Control	Address 4	Frame Body	FCS

Figure 4.4: MAC frame format

The MAC frame is composed in the MAC layer. However, because of the numerical complexity of the CRC generation, this FCS is generated in the physical layer hardware. Therefore, the MAC frame transmitted to the TX chain contains four octets filled with

‘0’s in the FCS field. These ‘0’s are replaced by the actual FCS after it has been generated in the CRC generation unit. The 32-bit FCS field is calculated over all MAC header fields and the frame body field. The generator polynomial is

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1.$$

The CRC update computation is done in parallel for every incoming byte. The incoming data (2 bits per clock cycle) are stored in a byte memory until the next byte is available for the CRC update. Note that the input bytes have to be reflected (MSB⇔LSB) before the CRC update. When the entire frame, with exception of the last 32 bits (‘0’s), has been passed through the CRC update unit, the resulting CRC is used to overwrite the last 32 bits of the MAC frame to generate the correct FCS. Before, the 32-bit CRC bytes have to be reflected back. The output order of the FCS from left to right is given by

CRC Bits [24 ... 31 16 ... 23 8 ... 15 0 ...7]

↓

MAC frame [0 … end-32 | FCS].

4.2.3 Packet Composition

The packet composer unit forms the PPDU frame without the PLCP preamble symbols as depicted in Figure 4.5. The PLCP preamble generation is described in Subsection 4.2.11.

RATE 4 bits	RESERVED 1 bit	LENGTH 12 bits	Parity 1 bit	Tail 6 bits	SERVICE 16 bits	PSDU= MAC frame	Tail 6 bits	Pad bits
SIGNAL					DATA			

Figure 4.5: PPDU frame format without preamble

All required information for the fields RATE, RESERVED, LENGTH, SERVICE and PSDU is delivered by the MAC layer. The transmit order for all fields is LSB first.

Table 4.1: Coding of RATE field

Rate (Mbit/s)	Bits R1-R4
6	1101
9	1111
12	0101
18	0111
24	1001
36	1011
48	0001
54	0011

The packet composer state machine composes the PPDU in the following steps:

- The data rates have to be coded according to Table 4.1.
- One RESERVED bit is appended.
- The PSDU length is encoded as the number of bytes in the LENGTH field.
- A parity bit is appended at bit position 17, which is generated as even parity bit (number of 1's including the parity bit is even),

$$\text{parity} = \text{bit}\,0 \oplus \text{bit}\,1 \oplus .... \oplus \text{bit}\,16.$$

- Six SIGNAL tail bits are appended at bit positions 18-23, which are set to '0'.
- A two-byte SERVICE field is inserted at the beginning of the DATA field. It consists of seven '0's, which are used for the scrambler initialization. Additionally, nine bits are reserved for future use and are set to '0' in the current implementation.
- Next, the PSDU (MAC frame) as shown in Figure 4.4 is appended.
- At the end of the PSDU, six '0's are appended, which are required to return the convolutional encoder to the "zero state". These tail bits have to be set back to '0's after scrambling. Therefore, the packet composer generates an output signal that indicates the tail bits, which are than set back to '0' within the scrambler unit.
- Finally, in order to guarantee a data length, which is a multiple of the number of coded bits per OFDM symbol, an appropriate number of pad bits have to be added. The computation of this number is described in [1, Section 17.3.5.3].

Two input and output data are read and written in parallel in every clock cycle.

4.2.4 Scrambling

The scrambler unit scrambles the entire DATA field, including the SERVICE, PSDU, TAIL and PAD bits. This function will prevent long runs of identical bits in the data stream. Furthermore, in the case of necessary retransmissions, the selection of changed scrambler initialization values results in different data patterns, which will increase the probability of a successful frame retransmission.

The generator polynomial is given by

$$S(x) = x^7 + x^4 + 1.$$

The resulting standard scrambler implementation is shown in Figure 4.6.

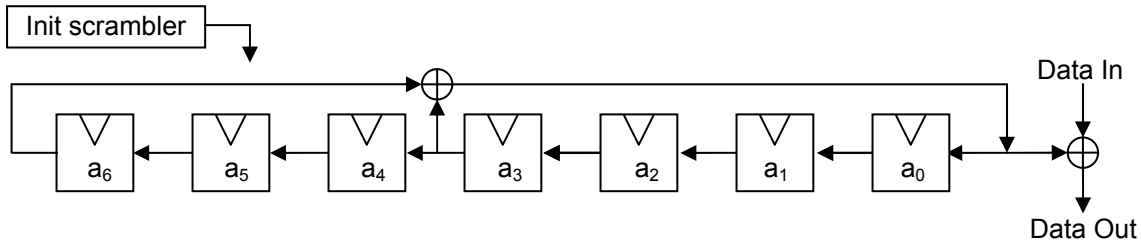


Figure 4.6: Data scrambler, serial implementation

All register values a_0 to a_6 are shifted to the left by one once per clock cycle, and the register a_0 is updated with the value $a_0 = a_3 \oplus a_6$. We can rewrite the update operation for two consecutive cycles as

$$a_1(\text{cycle2}) = a_3 \oplus a_6(\text{cycle1})$$
$$a_0(\text{cycle2}) = a_2 \oplus a_5(\text{cycle1})$$

To exploit the parallelism of the input data bits (2 input bits per clock cycle), a parallel implementation of the scrambler is introduced. In the parallel implementation, the two functions described above are executed in one clock cycle, and the register values are then shifted by two. Therefore, the shift register of length 7 can be subdivided into an "even" and an "odd" shift register of length 4 and 3, respectively. A block diagram of the parallel implementation is shown in Figure 4.7.

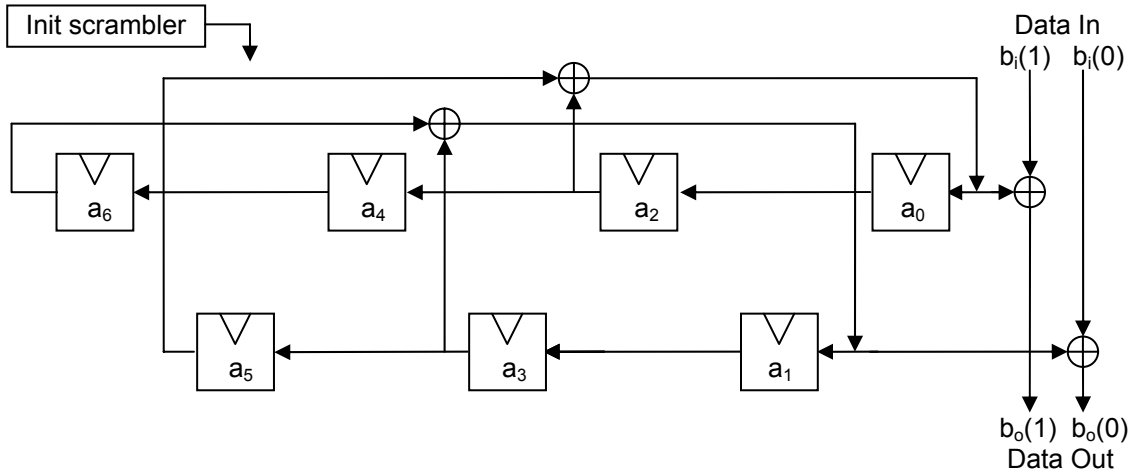


Figure 4.7: Data scrambler, parallel implementation

The scrambling operation of consecutive data bits $b_i(0)$, $b_i(1)$, which are fed into the scrambler unit in parallel, is done by the operation

$$b_o(0) = a_3 \oplus a_6 \oplus b_i(0)$$
$$b_o(1) = a_2 \oplus a_5 \oplus b_i(1)$$

The additional signal p_scram_en from the packet composer is used to indicate the tail bits. These tail bits are set back to '0', if p_scram_en = '0'.

4.2.5 Convolutional Encoding

The entire DATA field has to be encoded using a convolutional code of rate R = ½. Additionally, several puncturing schemes are applied to generate different code rates (R = ½, ⅔, ¾). These puncturing schemes enable variable data rates, which are traded with the error protection capabilities.

The convolutional encoder uses the generator polynomials $g_0 = 133_8$ and $g_1 = 171_8$ resulting in an encoder structure as depicted in Figure 4.8. The next input data i_0 into the encoder is assumed to reside in the input register r_0.

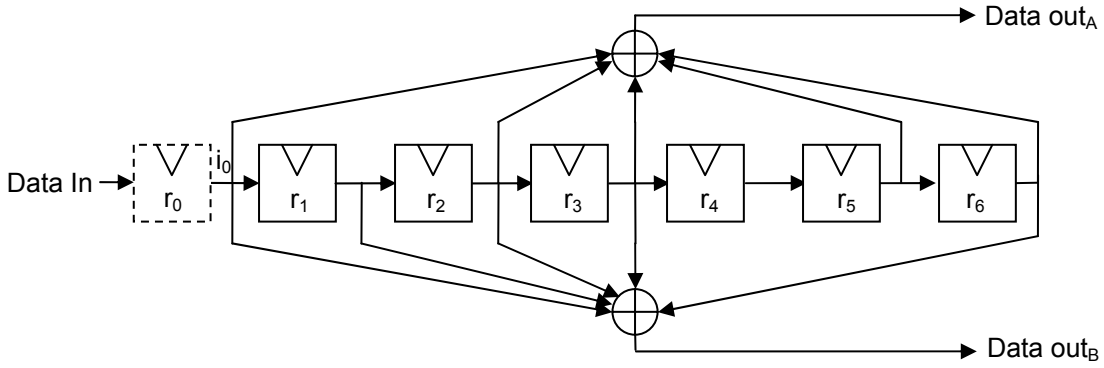


Figure 4.8: Convolutional encoder with constraint length 7, serial implementation

To guarantee processing parallelism in the encoding unit, several modifications are necessary. First, the shift register chain is subdivided into an even and an odd part with the following mappings:

$$\begin{aligned}
&\text{input}: && i_0 = r_0 \mapsto e_0, \\
&\text{even}: && r_2 \mapsto e_1, r_4 \mapsto e_2, r_6 \mapsto e_3, \\
&\text{odd}: && r_1 \mapsto o_1, r_3 \mapsto o_2, r_5 \mapsto o_3 .
\end{aligned}$$

The equations for the output data can then be written as

$$\begin{aligned}
\text{out}_A(i_0) &= r_0 \oplus r_2 \oplus r_3 \oplus r_5 \oplus r_6 = e_0 \oplus e_1 \oplus e_3 \oplus o_2 \oplus o_3, \\
\text{out}_B(i_0) &= r_0 \oplus r_1 \oplus r_2 \oplus r_3 \oplus r_6 = e_0 \oplus e_1 \oplus e_3 \oplus o_1 \oplus o_2 .
\end{aligned}$$

The input data i_1 following i_0 appears at the input of the shift register with a delay of one cycle. Therefore, input data i_1 is assumed to reside in an additional input register r_{-1} and can be mapped as

$$\text{input}: \; i_1 = r_{-1} \mapsto o_0 .$$

In the parallel implementation, the two consecutive bits i_0 and i_1 are concurrently shifted into the encoder in one clock cycle, which implies a shift of all register values by two. This in turn means that the "even" input data i_0 resides in the even register chain, whereas the "odd" data i_1 resides in the odd register chain. The output equations for input i_1 will take the form

$$\begin{aligned}
\text{out}_A(i_1) &= r_{-1} \oplus r_1 \oplus r_2 \oplus r_4 \oplus r_5 = o_0 \oplus o_1 \oplus o_3 \oplus e_1 \oplus e_2, \\
\text{out}_B(i_1) &= r_{-1} \oplus r_0 \oplus r_1 \oplus r_2 \oplus r_5 = o_0 \oplus o_1 \oplus o_3 \oplus e_0 \oplus e_1 .
\end{aligned}$$

The resulting parallel implementation of the convolutional encoder is shown in Figure 4.9.

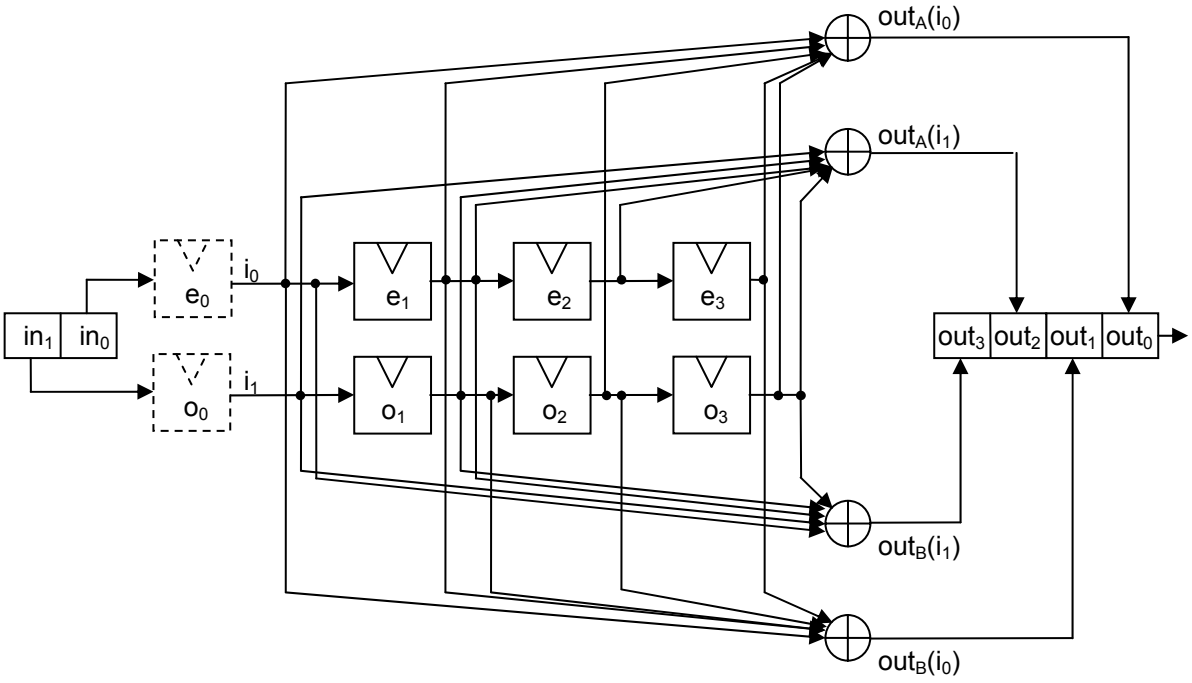


Figure 4.9: Convolutional encoder with constraint length 7, parallel implementation

4.2.6 Puncturing

The puncturing unit takes the encoded data stream from the convolutional encoder and applies different puncturing schemes depending on the target data rate. Higher data rates are achieved by puncturing out certain encoded bits. The puncturing schemes are described in detail in [1, Figure 115]. The four output data bits from the encoder as depicted in Figure 4.9 are fed into the puncturing unit concurrently. Depending on the puncturing scheme, up to four successive 4-bit-inputs are written into successive memory cells. The memory size is chosen such that the number of output bits after puncturing is a multiple of four, because always four bits have to be fed to the output in parallel. The memory requirements are shown in Table 4.2.

Table 4.2: Input/output relationship and memory requirements in the puncturing unit

Rate	Input bits	Punctured bits	Memory size	Output bits	Delay cycles
½	1 × 4 = 4	1 × 0 = 0	1 × 4 = 4	1 × 4 = 4	1
⅔	4 × 4 = 16	4 × 1 = 4	4 × 3 = 12	3 × 4 = 12	4
¾	3 × 4 = 12	2 × 1 + 1 × 2 = 4	2 × 3 + 1 × 2 = 8	2 × 4 = 8	3

Implementation details are depicted in Figure 4.10. An input process writes 4 input data bits per clock cycle into memory bank 0, until this memory is filled according to Table 4.2. The subsequent input data are written into a second memory bank 1. At the same time, an output process writes the data from memory bank 0 to the output. The delay introduced by the data buffering is also shown in Table 4.2.

Note: The first OFDM symbol (= SIGNAL field) is always transmitted with 6 Mbit/s, which corresponds to BPSK with coding rate ½. Therefore, the first 48 encoded bits are

always treated correspondingly; the RATE information fed into the puncturing unit is not used for the SIGNAL field bits.

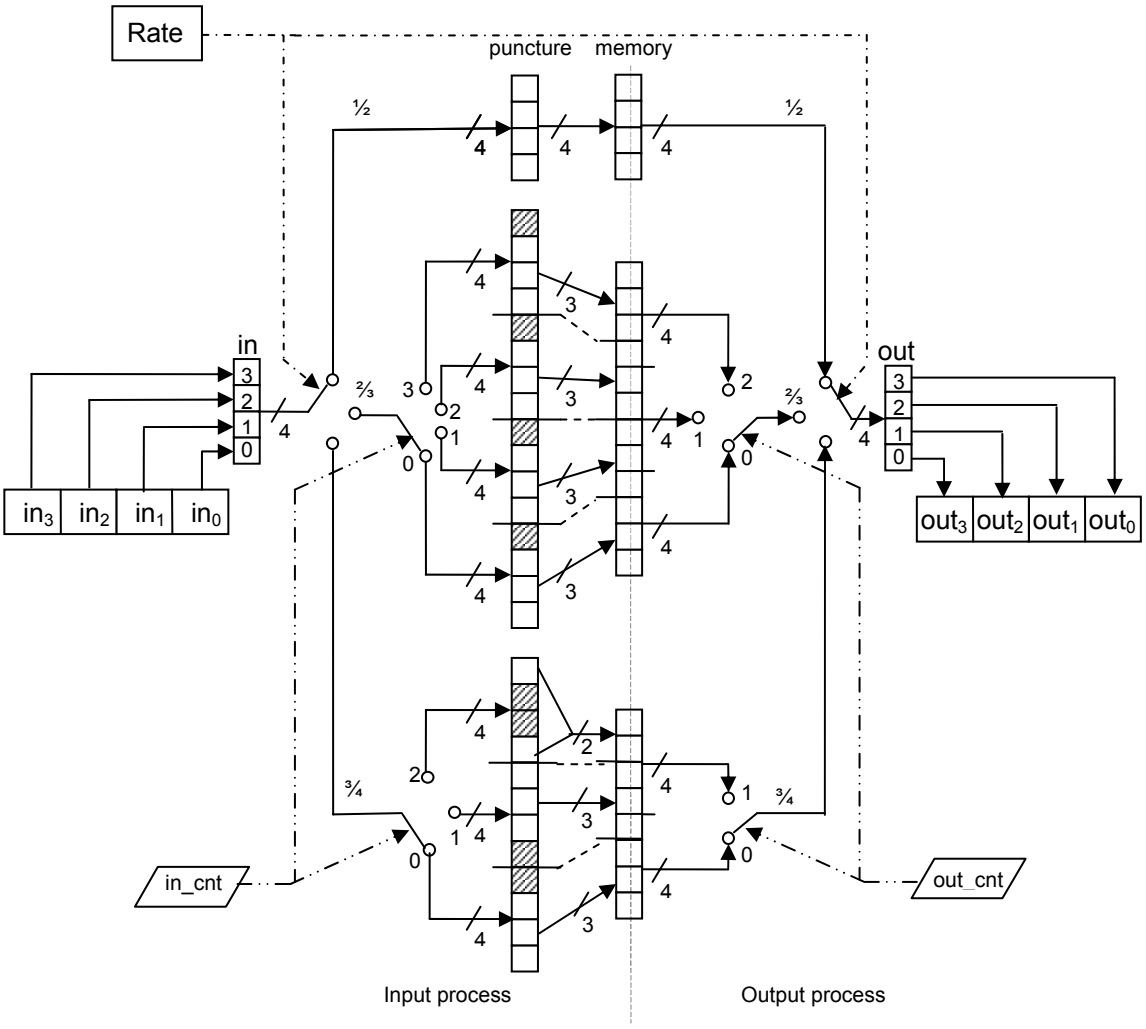


Figure 4.10: Puncturing unit, parallel implementation

4.2.7 Interleaving

The block interleaving unit implements a two-step permutation [1, Section 17.3.5.6] of the coded data bits within one OFDM symbol. The first permutation reads in the data bits row by row and writes them out column by column, and therefore prevents burst errors in the packet transmission by mapping adjacent coded bits onto nonadjacent subcarriers. The second permutation ensures that adjacent coded bits are mapped alternately onto less and more significant bits of the constellation, thereby long runs of low-reliability bits are avoided.

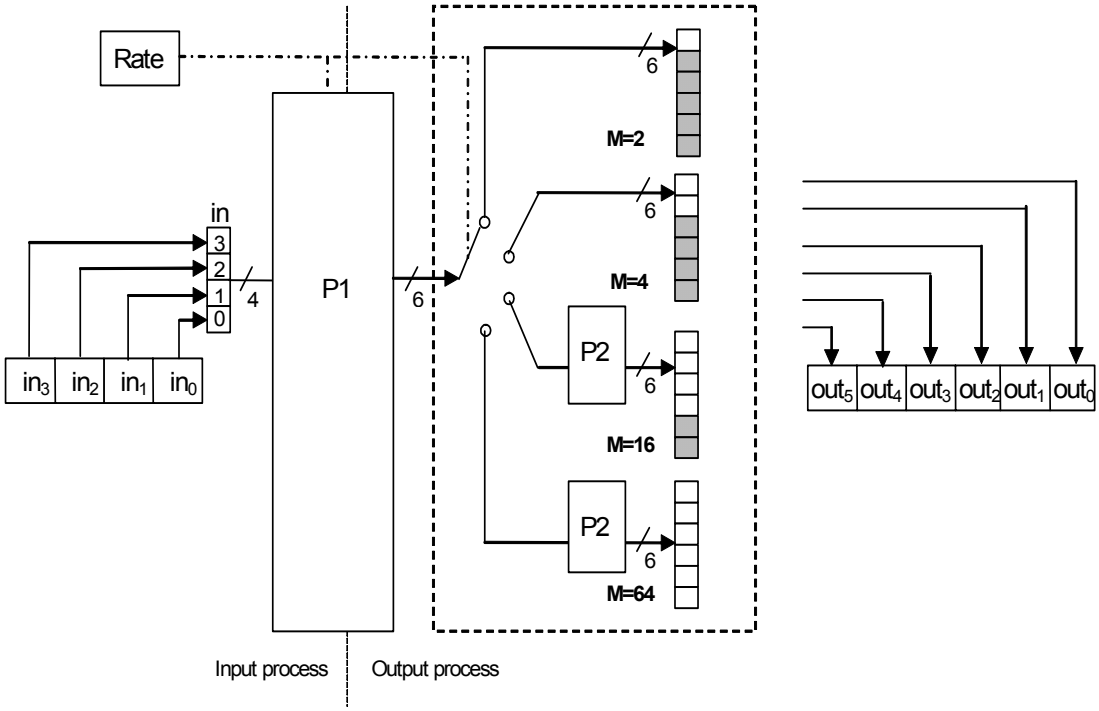


Figure 4.11: Interleaving unit, parallel implementation

A block diagram of the interleaving unit implementation is shown in Figure 4.11. Four input bits are processed per clock cycle and stored in the first permutation unit's (P1) memory, which consists of 16×24 cells of one bit. Memory address generators for input and output accomplish the first permutation task. Targeted for a resource-efficient FPGA implementation, the permutation memory is physically organized as 24 columns of 16×1 SRAMs. This structure allows simultaneous read/write operations in multiple columns, but only one cell per column can be accessed at a time. A memory map keeping the address generation for input and output addressing simple and still permitting an output of 1 bit (BPSK), 2 bits (QPSK), 4 bits (16QAM) or 6 bits (64QAM) in parallel is depicted in Table 4.3. The number in the cells refers to the n-th bit of the input sequence.

The second permutation (P2), used for modes 16QAM and 64QAM only, is conducted by a state machine and corresponding multiplexers before the data is communicated to the subsequent functional unit, i.e. to the modulation mapping.

Table 4.3: Interleaver memory map

M = 2 BPSK

0						1						2						3					
16						17						18						19					
32						33						34						35					
4						5						6						7					
20						21						22						23					
36						37						38						39					
8						9						10						11					
24						25						26						27					
40						41						42						43					
12						13						14						15					
28						29						30						31					
44						45						46						47					

M = 4 QPSK

0	16					1	17					2	18					3	19				
32	48					33	49					34	50					35	51				
64	80					65	81					66	82					67	83				
4	20					5	21					6	22					7	23				
36	52					37	53					38	54					39	55				
68	84					69	85					70	86					71	87				
8	24					9	25					10	26					11	27				
40	56					41	57					42	58					43	59				
72	88					73	89					74	90					75	91				
12	28					13	29					14	30					15	31				
44	60					45	61					46	62					47	63				
76	92					77	93					78	94					79	95				

M = 16 16QAM

0	16	32	48			1	17	33	49			2	18	34	50			3	19	35	51		
64	80	96	112			65	81	97	113			66	82	98	114			67	83	99	115		
128	144	160	176			129	145	161	177			130	146	162	178			131	147	163	179		
4	20	36	52			5	21	37	53			6	22	38	54			7	23	39	55		
68	84	100	116			69	85	101	117			70	86	102	118			71	87	103	119		
132	148	164	180			133	149	165	181			134	150	166	182			135	151	167	183		
8	24	40	56			9	25	41	57			10	26	42	58			11	27	43	59		
72	88	104	120			73	89	105	121			74	90	106	122			75	91	107	123		
136	152	168	184			137	153	169	185			138	154	170	186			139	155	171	187		
12	28	44	60			13	29	45	61			14	30	46	62			15	31	47	63		
76	92	108	124			77	93	109	125			78	94	110	126			79	95	111	127		
140	156	172	188			141	157	173	189			142	158	174	190			143	159	175	191		

M = 64 64QAM

0	16	32	48	64	80	1	17	33	49	65	81	2	18	34	50	66	82	3	19	35	51	67	83
96	112	128	144	160	176	97	113	129	145	161	177	98	114	130	146	162	178	99	115	131	147	163	179
192	208	224	240	256	272	193	209	225	241	257	273	194	210	226	242	258	274	195	211	227	243	259	275
4	20	36	52	68	84	5	21	37	53	69	85	6	22	38	54	70	86	7	23	39	55	71	87
100	116	132	148	164	180	101	117	133	149	165	181	102	118	134	150	166	182	103	119	135	151	167	183
196	212	228	244	260	276	197	213	229	245	261	277	198	214	230	246	262	278	199	215	231	247	263	279
8	24	40	56	72	88	9	25	41	57	73	89	10	26	42	58	74	90	11	27	43	59	75	91
104	120	136	152	168	184	105	121	137	153	169	185	106	122	138	154	170	186	107	123	139	155	171	187
200	216	232	248	264	280	201	217	233	249	265	281	202	218	234	250	266	282	203	219	235	251	267	283
12	28	44	60	76	92	13	29	45	61	77	93	14	30	46	62	78	94	15	31	47	63	79	95
108	124	140	156	172	188	109	125	141	157	173	189	110	126	142	158	174	190	111	127	143	159	175	191
204	220	236	252	268	284	205	221	237	253	269	285	206	222	238	254	270	286	207	223	239	255	271	287

4.2.8 Modulation Mapping

The modulation mapping unit integrates the following functions:

- Generation of modulation symbols using 1, 2, 4 or 6 encoded input bits, depending on the modulation alphabet (BPSK, QPSK, 16QAM, 64QAM).
- Generation of pilot symbols with BPSK modulation using a pseudo binary sequence.
- Introduction of zero subcarriers into OFDM symbol.
- Generation of physical subcarrier mapping addresses for IFFT input buffer addressing (note: this address generation is now also performed within the IFFT wrapper function to save address lines in the case of FPGA split).

Table 4.4: Encoding table for modulation mapping of input bits b0…b5 into I/Q outputs

BPSK			QPSK			16QAM					64QAM								
b0	0	1	**b0**	0	1	**b0|b1**	00	01	11	10	**b0|b1|b2**	000	001	011	010	110	111	101	100
I-out	-1	+1	**I-out**	-1	+1	**I-out**	-3	-1	+1	+3	**I-out**	-7	-5	-3	-1	+1	+3	+5	+7
			b1	0	1	**b2|b3**	00	01	11	10	**b3|b4|b5**	000	001	011	010	110	111	101	100
Q-out	0	0	**Q-out**	-1	+1	**Q-out**	-3	-1	+1	+3	**Q-out**	-7	-5	-3	-1	+1	+3	+5	+7

The interleaver unit uses a 6-bit interface to provide the interleaved data bits to the modulation mapping unit. However, only 1, 2, 4, or 6 bits are valid and used for the mapping depending on the modulation scheme chosen. The modulation mapping is done according to Table 4.4 [1, Tables 82-85]. Additionally, data scaling is performed using a modulation-dependent normalization factor KMOD and an integer multiplier of 30,000 to guarantee the highest possible precision in the subsequent IFFT unit. The KMOD and scaling factors as well as the resulting I/Q-output values are summarized in Table 4.5. The maximum output amplitude level is 32403, fitting to the 16-bit precision of the IFFT unit.

Table 4.5: Scaling factors and amplitude levels of the modulation mapping unit

	KMOD	scale = KMOD • 30000	scale • I/Q-out
BPSK	1	30000	± 30000
QPSK	$1/\sqrt{2}$	21213	± 21213
16QAM	$1/\sqrt{10}$	9487	± 9487, ± 28461
64QAM	$1/\sqrt{42}$	4629	±4629, ±13887, ±23145, ±32403

To form one OFDM-symbol, 48 modulated data symbols have to be generated. Next, 4 pilot symbols are appended. The pilot symbols, which correspond to the set of logical subcarriers [–21; –7; +7; +21], are defined by the mapping

$$P_{-21;-7;+7;+21} = \pm 1 \bullet 30000 \bullet [1;\ 1;\ 1;\ -1].$$

The sign bit is determined once for every OFDM symbol by using the output of a PN-code generator with polynomial $S(x) = x^7 + x^4 + 1$. This generator is initialized with "all ones''; 1's are replaced with –1 and 0's with 1. The generator is the same as in the scrambler unit. However, a serial implementation is used for the pilot generation. Finally, 12 zero subcarriers filled with 0's are appended, as the outer subcarriers (–32…–27, 27…31) and the 0's (DC) subcarrier are not used for data transmission.

4.2.9 FFT/IFFT

The FFT/IFFT unit is shared between the TX and RX chains. A wrapper unit guarantees the correct FFT operation mode by controlling a "forward-inverse flag". The flag is set to '0' for IFFT and to '1' for FFT operation.

The FFT/IFFT unit implementation uses the XILINX 64-point complex FFT/IFFT core [5]. A single-memory-space configuration was chosen, which uses a common memory as input and working buffer. This basic configuration is shown in Figure 4.12.

The FFT core is used in scale mode, which performs a scaling by 2 on the first processing pass. Therefore, the result is scaled up by 2 to compensate for this factor.

The processing time of the FFT core including the result-write operation takes 209 cycles (= 5.2 μs @ 40 MHz). The maximum allowable processing delay for real-time operation per FFT unit is 4 μs. Therefore, two FFT units are instantiated in parallel to guarantee real-time operation. This concept is depicted in Figure 4.13.

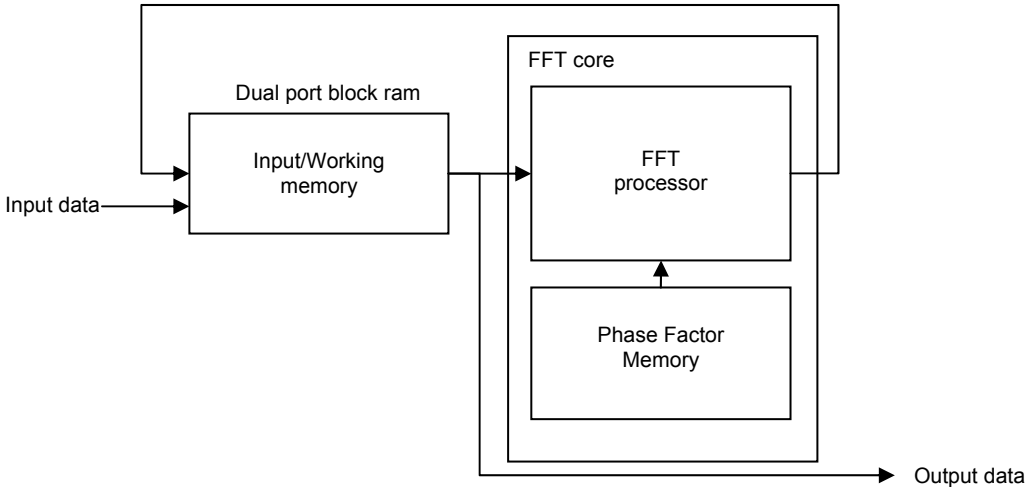


Figure 4.12: Single-memory-space FFT configuration

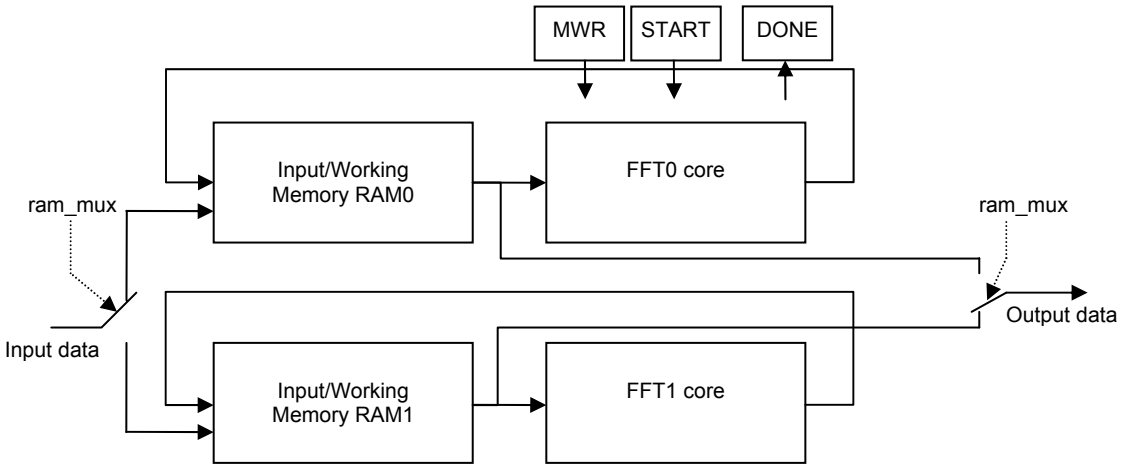


Figure 4.13: Double-FFT core configuration

A three-stage sequence of operations is used to perform the transform [5]:

- Data-load phase,
- compute phase, and
- data-output phase.

Data-load phase: During the data-load phase, 64 input data are alternately written into the input memories RAM0/1. This operation is initiated by asserting an input data write strobe signal MWR to the FFT core. When 64 new input data have been loaded into RAM0/1, the compute phase has to be initiated by asserting a START signal to the FFT0/1 core. Both operations are controlled by the FFT/IFFT input state machine, which is shown in Figure 4.14. If RAM0/1 is available and FFT0/1 is idle, a transition from the Wait1/0 to Load0/1 state is performed and the associated write strobe MWR is assigned. Then, 64 input data are written into RAM0/1 in state Load0/1. Note: For the IFFT, the input addresses generated by the FFT core are not used. Instead, the logical-to-physical address mapping, which is required after the modulation-mapping stage, is performed during the data-load phase. The address generation for the input memories RAM0/1 is done by a mapping-address lookup from a ROM table. The ROM table contains the following address order, which corresponds to the output order of the modulation-mapping unit:
[38,39,40,41,42,44,45,46,47,48,49,50,51,52,53,54,55,56,58,59,60,61,62,63,1,2,3,4,5,6,8,
9,10,11,12,13,14,15,16,17,18,19,20,22,23,24,25,26,43,57,7,21,0,27,28,29,30,31,32,33,34,
35,36,37].

Compute phase: During the transition from Load0/1 to Wait0/1 or Load1/0, the FFT/IFFT input state machine asserts the START signal to the FFT0/1 core to initiate the compute phase. After this transition, the state machine waits for the next FFT1/0 core (+RAM1/0) to become available for input data.

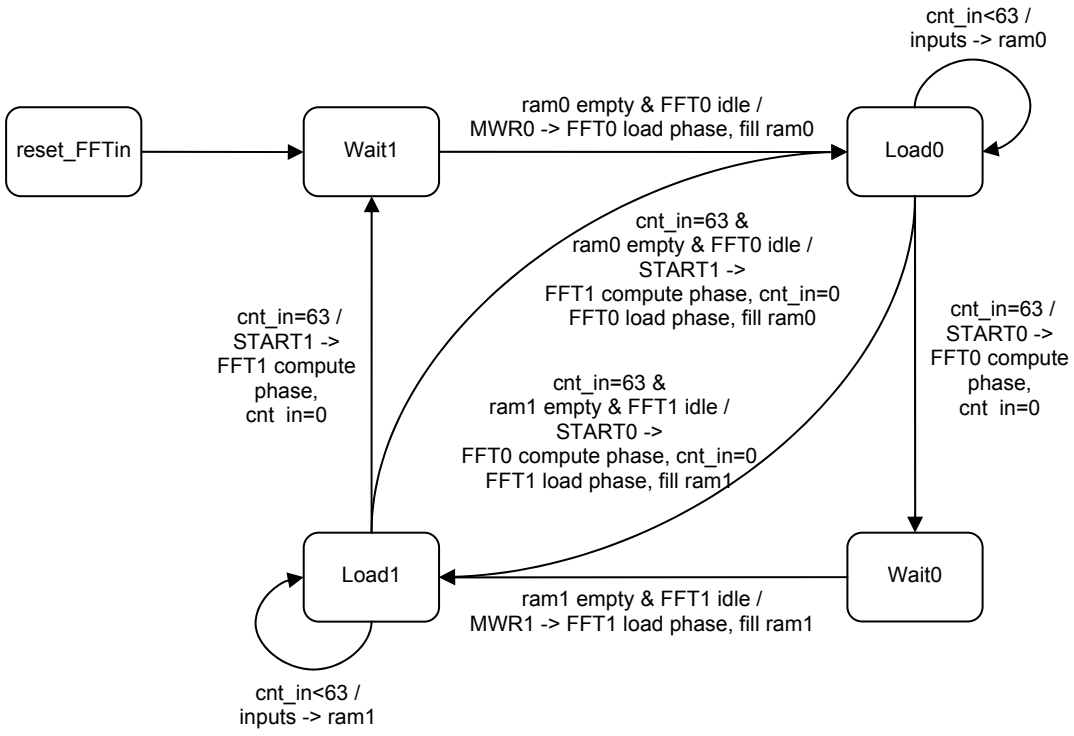


Figure 4.14: FFT/IFFT input FSM, controls data-load phase and compute-phase start

The FFT core generates a DONE signal at the end of the compute phase. This signal controls the FFT/IFFT output state machine, which is shown in Figure 4.15. When the FFT0/1 DONE signal is asserted, the output FSM switches from the Wait0/1 to the Output0/1 state. In this state, the next 64 output data are written into a memory of the next functional unit. If the next unit is not ready to take data, the output process is stalled. Also, if the subsequent FFT computation is finished before the current output data are completely written, the subsequent FFT process is stalled (chip enable = '0'). The RAM0/1 resources are released, when all 64 output data have been written into the next unit. This in turn enables the FFT/IFFT input FSM to initiate the next load process. Note: The output data are provided in bit-reversed order. Therefore, the data have to be rearranged into their natural order during the write operation into the subsequent blocks. In the receive chain, this unit (channel estimator) uses an internal address generator to perform this reordering. This allows the removal of these address lines from the interface, which is done to facilitate the split of the design onto two FPGA boards.

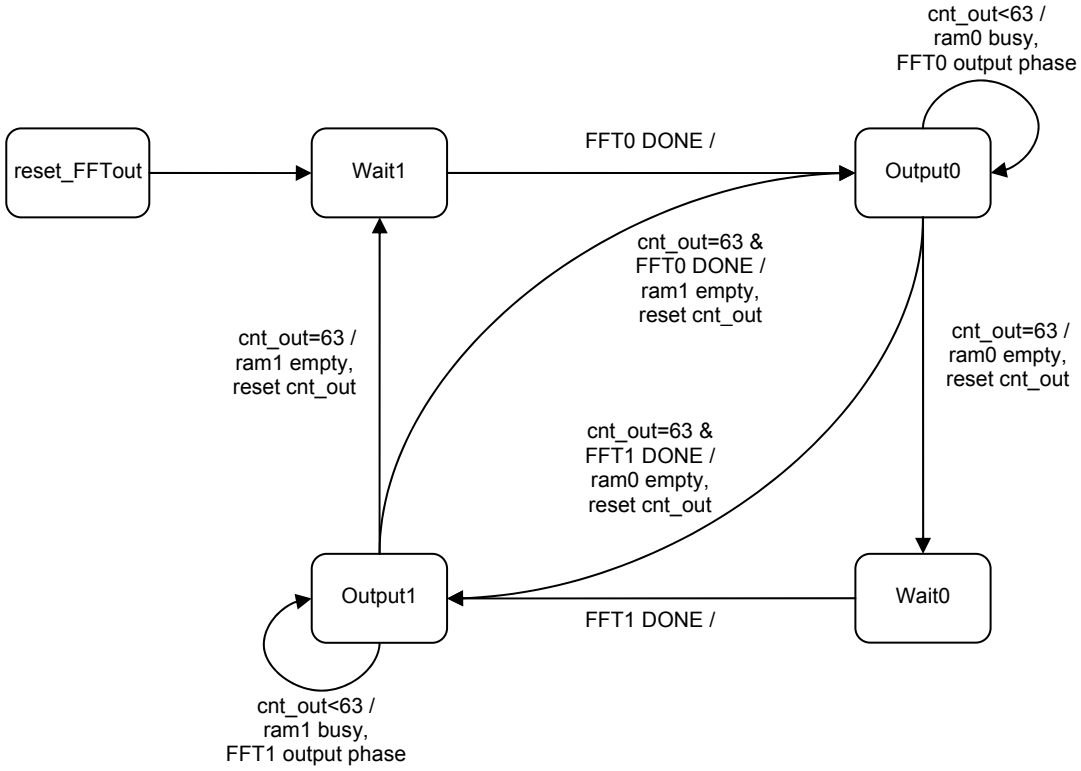


Figure 4.15: FFT/IFFT output state machine, controls data-output phase and release of RAM resources

4.2.10 Cyclic Extension

The cyclic extension unit stores the IFFT output data into a memory and prepends the physical subcarriers 48…63 as cyclic extension. This extension forms the guard interval. The resulting symbol of length 80 forms one OFDM symbol.

The cyclic extension unit applies a dual-ported RAM. The input data are written into the address locations defined by the IFFT output address generator. Subcarrier data 48…64 are written concurrently into their corresponding address locations. Therefore, the 80 memory locations are filled within 64 cycles. The resulting OFDM symbol is written to the frame-formatting unit. To guarantee continuous transmission, a swinging-buffer concept is applied. While input data are written into RAM0, the OFDM symbol in RAM1 is output, and vice versa.

4.2.11 Frame Formatting

The frame-formatting unit is responsible for the concatenation of the PREAMBLE and DATA symbols. After resetting the TX chain, this unit immediately starts to output the short preamble. This is done by outputting 10 repetitions of the precomputed and stored short preamble symbol. Next, 2½ repetitions of the precomputed long preamble symbol are appended. The output of both preambles takes 16 μs. Within this 16-μs timeframe, the TX chain has to finish the generation of the first data symbol. The data symbols are consecutively appended to the preambles to form the transmit frame (PPDU). The frame-formatting unit determines the timing behavior of the entire transmit chain. The output of this unit needs to be continuous with a rate of 20 MHz. The input and therefore the TX chain might be stalled during preamble operation and because the TX chain works on a 40-MHz clock.

4.3 RX Chain Functionality

4.3.1 Overview

The following paragraphs describe the functional units of the RX chain, their basic behavior and implementation details. A block diagram of the RX chain is shown in Figure 4.16. Note that in every clock cycle the RX chain writes 2 decoded data bits in parallel into the RX FIFO.

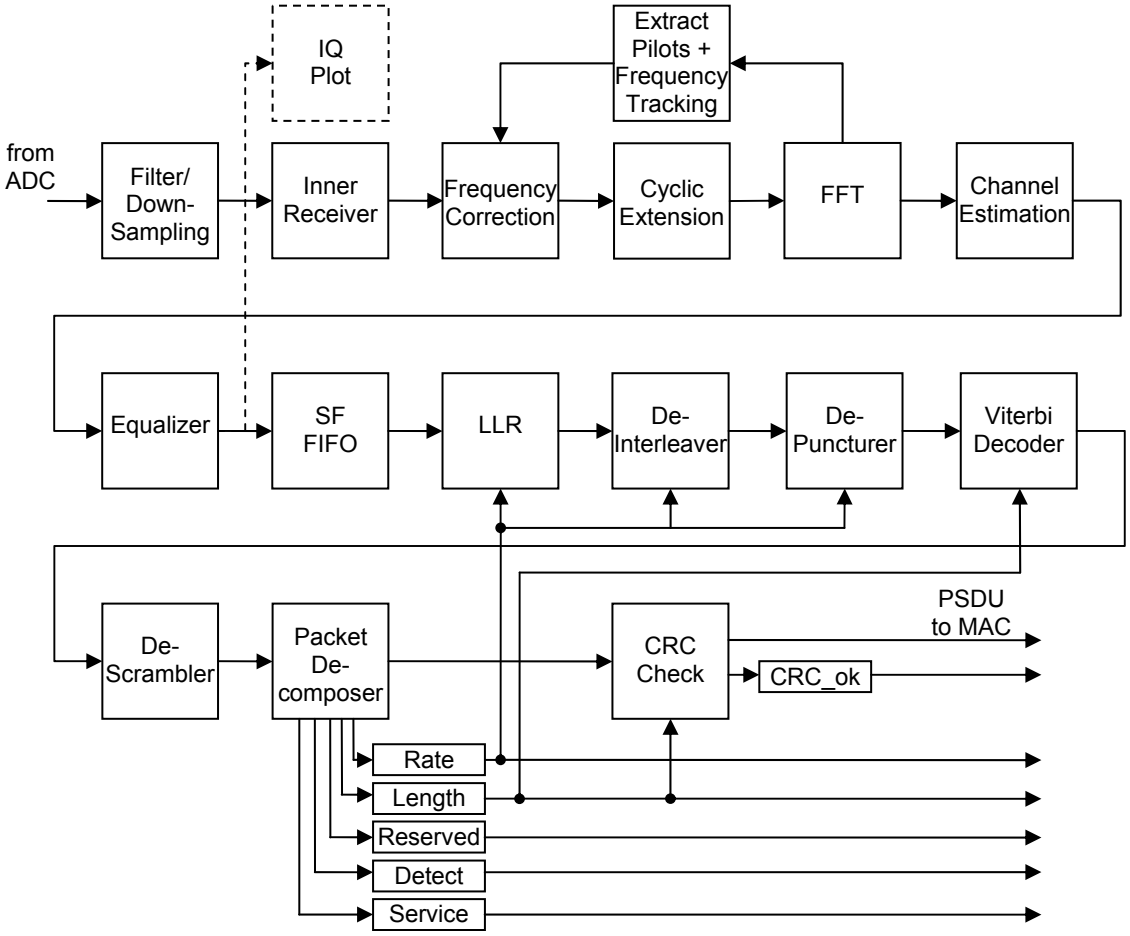


Figure 4.16: RX chain block diagram

4.3.2 Inner Receiver

The inner receiver unit is the first computational unit in the receive chain working on data received over the transmission channel after filtering and down-sampling to 20 MHz. The main tasks of the inner receiver are

- signal detection and timing synchronization using correlation and matched filtering methods,

- frequency offset estimation and initiation of analog and digital frequency offset compensation, and
- power estimation and closed-loop automatic gain control by adjusting a variable gain amplifier in the analog frontend.

The main functional blocks of the inner receiver unit are shown in Figure 4.17. These blocks work concurrently on the input data. They are controlled by the inner receiver FSM, which is discussed in detail in [6, 7]. Some control information has to be interchanged between the inner receiver FSM and the computational unit. Most importantly, the signal PERIOD is used to distinguish between the short and long preamble phases. This signal is set to '0' until the short preamble (with periodicity of 16 samples) has been detected. Once the end of the short preamble is found, the signal PERIOD is set to '1' to indicate the reception of the long preamble (with periodicity of 64 samples) to the computational units.

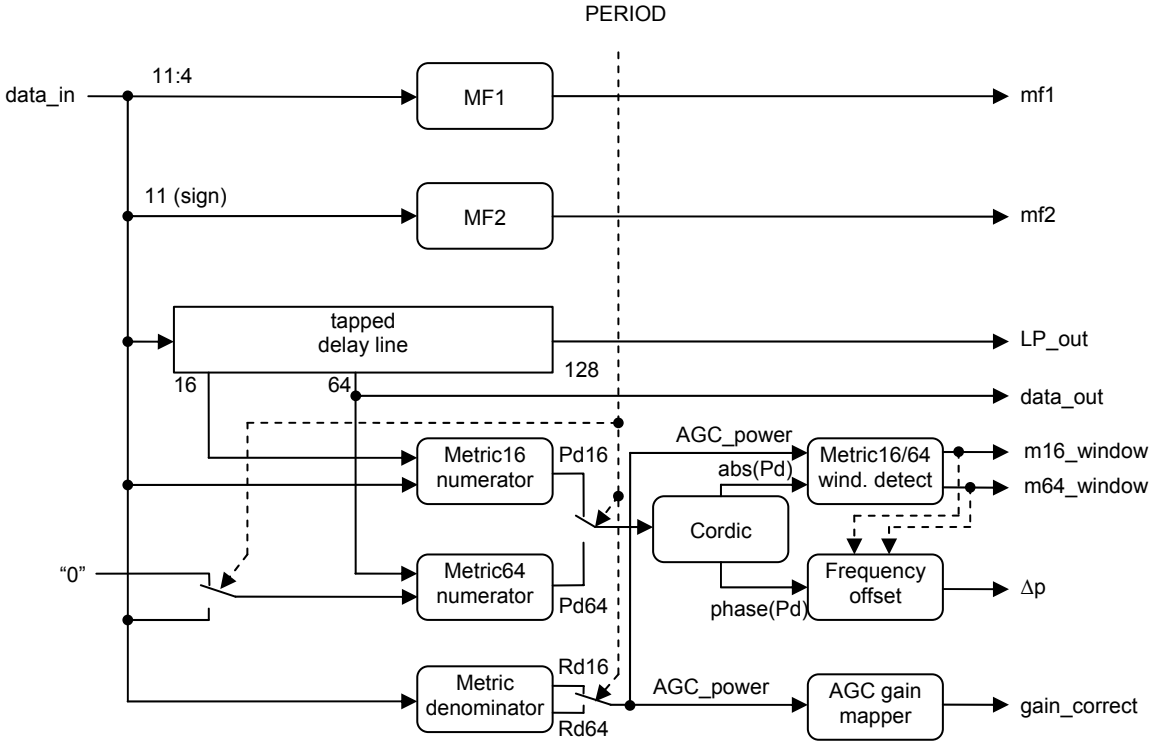


Figure 4.17: Inner receiver computational unit

In the following paragraphs, the blocks of the computational unit and their behavior are described briefly. More comprehensive descriptions can be found in [6, 7].

Matched Filter1 (MF1): This block takes the most significant 8 bits from the input data and does a matched filter operation on these data. The reference sequence is the nominal response of an ideal channel to the transmission of a short preamble symbol ($coeff_{0...15}$). This filter operation is implemented as a transposed FIR filter structure performing the operation

$$y_n = \left| \sum_{k=0}^{15} \text{data\_in}_{n-k} \bullet \text{coeff}^*_{15-k} \right|^2 .$$

This transposed structure is well suited for hardware implementation, as the parallel addition of 16 intermediate filter results can be omitted ([6], see Figure 6.3). However, this block still is computational very expensive because in theory it requires 16 complex multipliers (= 64 real multipliers) and 2 adders/subtractors in parallel (the implementation requires fewer multipliers owing to '0'-coefficients etc.). Finally, the squared magnitude of the matched filter response is output, which requires 2 additional multipliers and one adder. The squared magnitude is used to prevent an additional square-root operation.

Matched Filter2 (MF2): To minimize the complexity of this matched filter with length 64, only the sign bits of the input data and of the reference sequence are compared, and the 64 results are summed. The complex sign-bit multipliers can be replaced by a look-up table with 4 inputs (signs of real and imaginary parts of data and coefficients) and 2 outputs (real and imaginary parts of the sign-bit multiplications). Finally, the squared magnitude of the matched filter response is output, which requires 2 multipliers and one adder. Again, the squared magnitude is used to avoid a square-root operation.

Tapped delay line: The input data have to be stored in a tapped delay line to perform cross-correlation operations and to save some history of the input data until the correct timing synchronization has been determined. This tapped delay line is organized as a chain of distributed memory blocks with block length of 16. This assures a very efficient implementation [6] compared with register chains. The tapped delay line storing the input data comprises three taps, which correspond to delays of 16, 64 and 128 data samples. The first two taps are used for the correlation operations with periodicity 16 and 64, respectively. The third tap is required to store the long preamble, which is twice as long as a normal OFDM symbol. Finally, after timing synchronization, the first and second halves of the long preamble are output in parallel via "LP_out" and "data_out" as indicated in Figure 4.17. The two parts of the long preamble are further processed and averaged in the frequency-correction unit.

Metric16/64 numerator: To find periodic structures in the received signal, the numerator of the periodicity metric [7] is computed as cross-correlation between the input data and a delayed (by 16/64 samples) version of this signal,

$$\text{Pd}16/64_n = \sum_{k=0}^{15/63} \text{data\_in}_{n-k} \bullet \text{data\_in}^*_{n-16/64-k} .$$

An iterative implementation is possible using the update formula

$$\text{Pd}16/64_n = \text{Pd}16/64_{n-1} + \text{data\_in}_n \bullet \text{data\_in}^*_{n-16/64} - \text{data\_in}_{n-16/64} \bullet \text{data\_in}^*_{n-2*(16/64)} .$$

The implementation of this operation is depicted in Figure 4.18. Here, the correlator only performs the complex multiplication $\text{data\_in}_n \bullet \text{data\_in}^*_{n-16/64}$.

In the iterative implementation, care has to be taken to prevent a bias in the register. This is achieved by correctly resetting it before correlation calculation. Note that for periodicity 64 the numerator is divided by 4 to compensate for the 4-fold summation length.

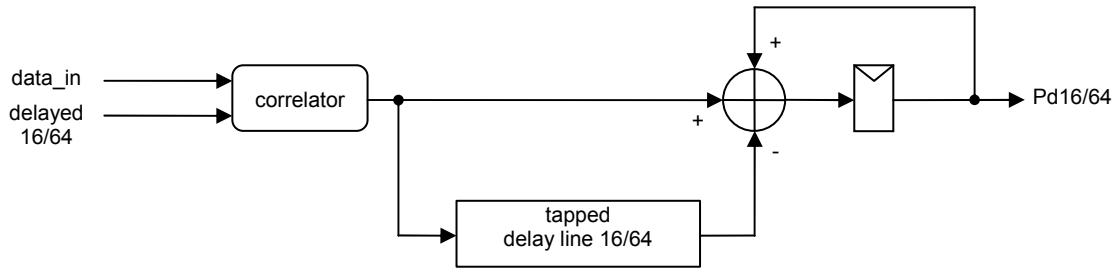


Figure 4.18: Iterative implementation of cross-correlation computation

Cordic: The outputs of the Metric16/64 numerator blocks are fed into a cordic processor to estimate the magnitude and phase of the complex metric values. Note that the magnitude is scaled by a factor 1.6468 as a result of the cordic implementation. This scaling factor needs to be compensated for in the calculation of the threshold values, which are required for periodicity detection. The cordic is implemented as on-line processor [6] with 12 processing stages. After 6 processing stages, one pipeline stage is introduced to meet the target processing speed of 20 MHz (sample input speed).

Metric denominator: The denominator computation is conceptually identical to the numerator computation as shown in Figure 4.18, but here an auto-correlation with elements $\text{data\_in}_n \bullet \text{data\_in}_n^*$ is computed. Both denominators (16/64) are determined in parallel using a tapped delay line with taps at correlation delays of 16 and 64.

Metric16/64 window detect: The periodicities of the short and long preambles result in correlation values, which remain on a certain plateau level as long as the input signals maintain the periodicity. This behavior is used to detect the short- and long-preamble reception by tracing the plateau phases [7]. A running sum is used to find the plateaus,

$$S_n^{16/64} = \begin{cases} \min\left(S_{n-1}^{16/64}+1, S_{max}^{16/64}\right) & \text{if } Pd16/64 > \text{met\_thr} \bullet 1.6468 \bullet Rd16/64 \\ \max\left(S_{n-1}^{16/64}-1, 0\right) & \text{if } Pd16/64 \le \text{met\_thr} \bullet 1.6468 \bullet Rd16/64 \end{cases}.$$

The detection of the short/long preamble is indicated to the inner receiver FSM if $S_n^{16/64} = S_{max}^{16/64}$.

Frequency offset: The frequency-offset estimation is performed by averaging the estimated phase values “phase_Pd” over 16 consecutive phase values, as shown in Figure 4.19. Additionally, the averaged phases are divided by the periodicity 16/64 to estimate the phase offset Δp. The related frequency offset is

$$\Delta f = \frac{\Delta p}{2\pi T} = \frac{\Delta p \bullet 20\,\mathrm{MHz}}{2\pi}.$$

In addition to the phase offset Δp the unnormalized Δp_x64 is output, which serves as a more precise estimation for the frequency-correction unit.

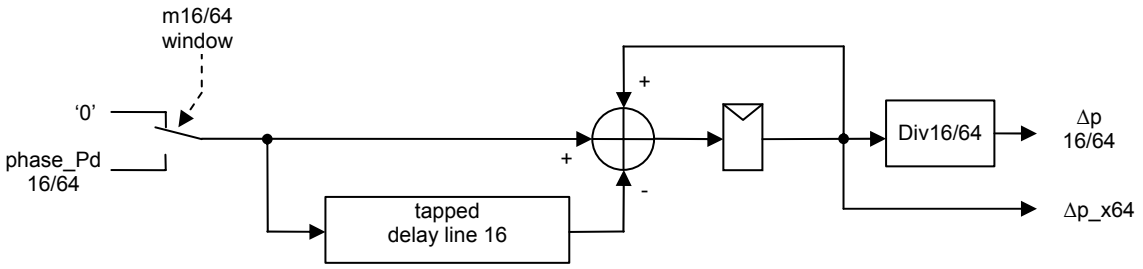


Figure 4.19: Frequency-offset estimation unit

Coarse (analog) frequency update: The frequency-offset estimate resulting from the short preamble is used to program a direct digital synthesizer (DDS), which in turn adjusts the mixer frequency for the IF down-conversion. The output frequency of the DDS is determined by the formula

$$f_{out} = \frac{\Delta\mathrm{phase} \bullet CLK_{in}}{2^{32}}$$

with Δphase as 32-bit tuning word to be fed into the DDS, CLK_{in} = 120 MHz as DDS reference clock, and f_{out} as DDS output frequency in MHz. Hence, the DDS tuning word is generated by

$$\Delta\mathrm{phase} = \frac{f_{out}}{120\,\mathrm{MHz}} \bullet 2^{32} = \frac{40\mathrm{MHz} + \Delta f}{120\mathrm{MHz}} \bullet 2^{32} = \frac{1}{3} \bullet 2^{32} + \frac{\Delta p \bullet 20\,\mathrm{MHz}}{2\pi \bullet 120\mathrm{MHz}} \bullet 2^{32}.$$

The first term $1/3 \times 2^{32}$ = 1,431,655,765 = 01010101010101010101010101010101b of Δphase is used as standard update value for the DDS (e.g. after reset). The second term, which is the frequency-correction term, is updated after successful frequency estimation within the short preamble period. The value π is presented by the integer value 32767 at the output of the frequency offset estimator, which results in a frequency-scaling factor of

$$\mathrm{scale}_f = \left\lfloor \frac{32767}{\pi} \right\rfloor = 10430.$$

Hence, the second term of Δphase can be simplified to

$$\Delta\mathrm{phase\_offset} = \frac{\Delta p \bullet 20}{2\pi \bullet 10430 \bullet 120} \bullet 2^{32} = \Delta p \frac{1}{3} 2^{15} = \Delta p \bullet 10923.$$

AGC gain mapper: The basic functionality of the gain control loop is described in [6, 7]. The AGC block takes the estimated signal energy computed in the metric denominator unit (see Figure 4.17) as input and calculates a power error value. This error value is used

to update the gain of the variable gain amplifier (VGA) and to adjust the received power accordingly. As the VGA characteristic is exponential, the iterative gain update can be described by

$$g_{n+1}^{dB} = g_n^{dB} + \Delta g_n^{dB}.$$

The AGC gain mapper, which outputs g_{n+1}^{dB} after every iteration, needs to find Δg_n^{dB} as a function of the estimated signal energy E_n,

$$\Delta g_n^{dB} = f_{NL}\left(E_{err,n}^{dB}\right)$$

$$E_{err,n}^{dB} = 10\log\left(\frac{E_n}{E_{tar}}\right)[\text{dB}].$$

The logarithm is not computed explicitly. Instead, the input energy E_n is compared with certain thresholds to find a quantized measure for the deviation from the target energy E_{tar}. The thresholds are precomputed according to

$$E_{thr} = E_{tar} \cdot 10^{\frac{thr(E_{err}^{dB})}{10}}.$$

The predefined thresholds and the associated nonlinear mapping into gain-correction values are shown in Table 4.6. If the energy error is less than ±0.25 dB relative to the target value, no gain correction is performed ($\Delta g_n^{dB} = 0$). In the case of a positive energy error between +0.25 and +1 dB, the gain correction term will be –0.015625 dB to slightly decrease the VGA gain. Higher energy-error values result in a nonlinear (over-proportional) increase of the gain-correction terms. Note: The target energy term in the implementation is $E_{tar} = 2789.5$.

Table 4.6: Predefined threshold energies and nonlinear mapping into Δg_n^{dB}

$thr(E_{err}^{dB})$	±0.25dB	±1dB	± 2.5dB	±4.5dB	±6.5dB
Δg_n^{dB}	∓0.015625dB	∓0.03125dB	∓0.1dB	∓0.25dB	∓0.4dB

4.3.3 Digital Frequency-Offset Correction

Fine frequency-offset correction

The inner receiver unit computes the fine phase-offset estimate Δp from the long preamble. The frequency-offset correction unit de-rotates the complex input data by the estimated phase

$$\text{data\_out} = \text{data\_in} \cdot \exp(-j2\pi(\Delta f + \Delta f_{resid})nT) = \text{data\_in} \cdot \exp(-j(\Delta p + \Delta p_{resid})n)$$

with n as time index and Δp_{resid} as residual phase offset. This rotation operation is done by a cordic algorithm. To prevent overflow and to avoid the compensation of the cordic

scaling factor (1.6468), the input data are divided by 2. The necessary scaling is then done implicitly in the equalizer. The cordic is fed with the time-variant phase $(\text{delta\_p} \cdot n) \bmod \pi$.

To remove the frequency offset from the entire Long Preamble (LP) (from which the offset is estimated), the first LP symbol is stored in the inner receiver and then fed into the frequency-correction unit in parallel with the second LP. After parallel frequency compensation, both LP parts are averaged and passed to the FFT unit, as shown in Figure 4.20. Note that the cyclic extension of the LP has already been removed in the inner receiver to prevent additional latency.

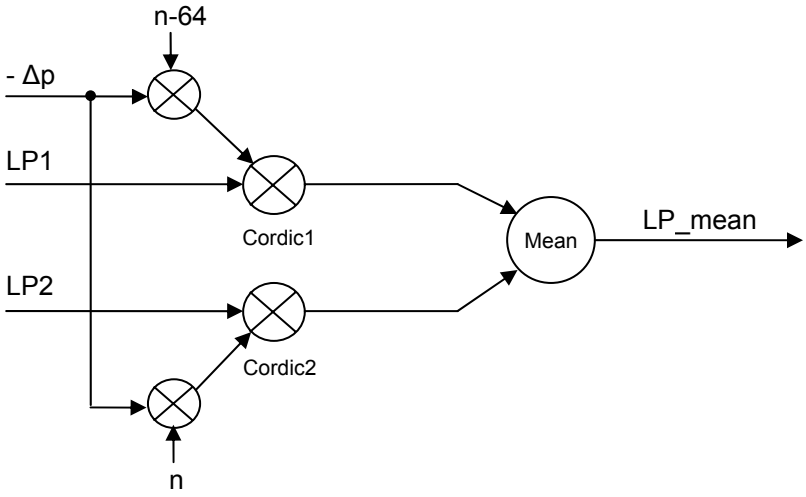


Figure 4.20: Frequency correction and average for long preamble

Frequency-tracking loop

To compensate the residual frequency error after the fine frequency correction, an additional frequency-tracking loop has been implemented. This loop is driven by the pilot subcarriers. The four pilots $P_{-21;\,-7;\,+7;\,+21}$ are separated at the output of the FFT. The long preamble pilots $P_{LP:-21;\,-7;\,+7;\,+21} = [1;\,-1;\,1;\,1]$ are used for pilot-channel estimation as described in Section 4.3.5 and saved as reference values. The pilots within the data symbols are then used to estimate the residual frequency offset and phase noise:

- First, the polarity introduced to the pilot subcarriers (see Section 4.3.5) has to be removed using a LFSR.
- Next, the pilot at logical address 21 is inverted to fit the PA pilot reference.
- The pilot-correlation terms $P_{data} P_{LP}^{*} = r_{P_{data}} r_{P_{LP}} \cdot \exp(\varphi_{P_{data}} - \varphi_{P_{LP}})$ are determined for all 4 pilot subcarriers.
- The phase angle $\Delta\varphi_{resid}$ is estimated from the mean of the correlation terms, which can be viewed as a weighted sum of the phase angles per pilot subcarrier. A cordic rotation is used for the angle computation.
- The frequency correction is done in the time domain within the digital frequency correction unit in order to minimize inter-carrier interference. For small frequency

deviations (compared with the subcarrier spacing), the transformation into the time-domain correction phase is $\Delta p_{resid} \approx \gamma \cdot 2/64$, with γ as loop gain (set to ¼ in the current implementation).

4.3.4 Cyclic Extension Removal

The cyclic extension removal unit takes the input data frame from the inner receiver at a sample rate of 20 MHz. It than periodically removes 16 samples forming the guard interval and forwards the 64 samples belonging to one OFDM symbol into the FFT input buffer (see Section 4.2.9). The input FSM toggles between two states "cyclic extension removal" and "read data from input". Note: The first OFDM symbol received by this unit is the average of the two halves of the long preamble. For this preamble symbol, the cyclic extension will already be removed in the inner receiver unit for complexity reasons. Therefore, the input FSM is initialized to start in the "read data from input" state.

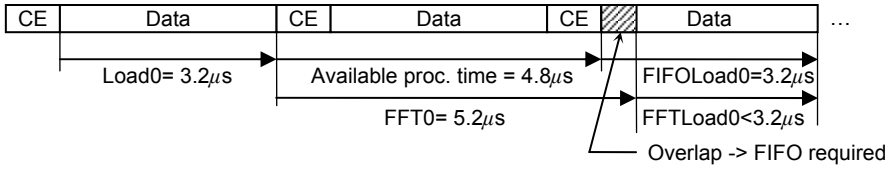


Figure 4.21: Timing issues in cyclic extension unit

To guarantee real-time operation, the input data have to be written into a FIFO. The timing requirements are shown in Figure 4.21. The FFT computation following the cyclic extension removal stage takes 5.2 μs (210 cycles @ 40 MHz). The next input samples for this FFT unit arrive after 4.8 μs (2 × 16 cycles cyclic extension (CE) + 1 × 64 cycles data @ 20 MHz). Therefore, the next input data have to be buffered. As the RX chain runs at 40 MHz, the latency introduced can be removed again later by using a FIFO. As soon as the FFT is completed, data are clocked into the FFT input buffer at 40 MHz from the FIFO. In parallel, input data are written from the inner receiver into the FIFO at 20 MHz. Hence, the overlap introduced can be compensated for during the data-load phase.

The output FSM starts writing data to the output ports if the FIFO is not empty. The output data are written into the FFT memory in natural order, using the internal address generator of the FFT core (see Section 4.2.9).

4.3.5 Channel Estimation

The channel-estimation unit, which follows the FFT processor, performs the following tasks:

- Sort bit-reversed input data from the FFT output into natural order using an internal address generator (addr_in),
- perform physical-to-logical mapping of the frequency domain data using a second internal address generator (addr_out),

- estimate and store channel coefficients for every subcarrier (channel estimator), and
- feed frequency domain data and associated channel coefficients c_i in parallel into equalizer.

A block diagram of the channel estimator is shown in Figure 4.22.

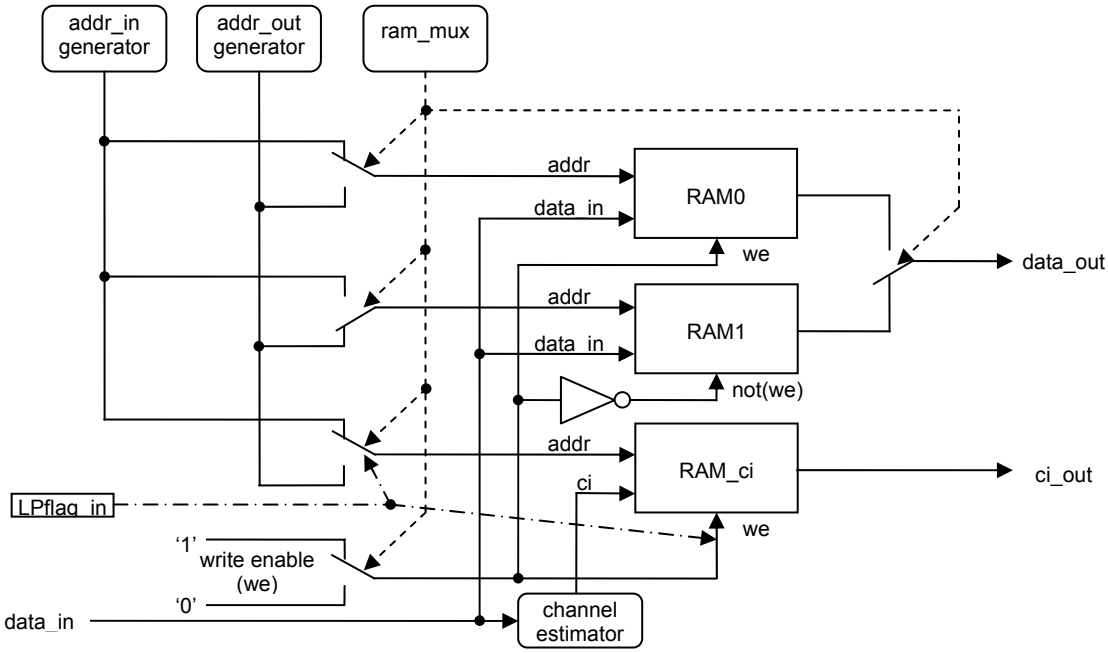


Figure 4.22: Channel-estimation unit

The removal of the bit-reversed order of the FFT output data is done by the "addr_in" generator during the data-input process. The address generator is implemented in the estimation unit to remove the address lines from the interface. The input address generator periodically generates 64 addresses [0 16 32 48 4 20 36 52 8 24 40 56 12 28 44 60 1 17 33 49 ... 15 31 47 63].

A diagram of the input FSM is depicted in Figure 4.23. At the beginning of the RX operation, the first 64 input data (long preamble) are sampled into RAM0. The input FSM then waits for the subsequent input data blocks to be written alternatingly into RAM1/0. Once a buffer is filled, the signal ram_req1/0 = '1' indicates the availability of the data for the output process. The switch ram_mux is driven by the input FSM and controls the address multiplexers and input/output behavior as indicated in Figure 4.22.

The channel-estimation task is performed in parallel with the input operation of the long preamble. A least-squares estimator [8] is implemented to calculate the complex channel coefficients

$$c_i = \frac{\text{lp\_data}_i}{\text{lp\_reference\_data}_i}.$$

Because the long-preamble reference data can only take on the values –1 or +1, the channel estimation is done by the following routine, which can be very easily implemented in hardware:

```
// channel estimator
for all elements of long PA
  if lp_reference_data_i == -1
          then c_i = -1 * lp_data_i
          else c_i =      lp_data_i
  end if
end for
```

The estimated channel coefficients are written into the memory RAM_ci during the long-preamble phase.

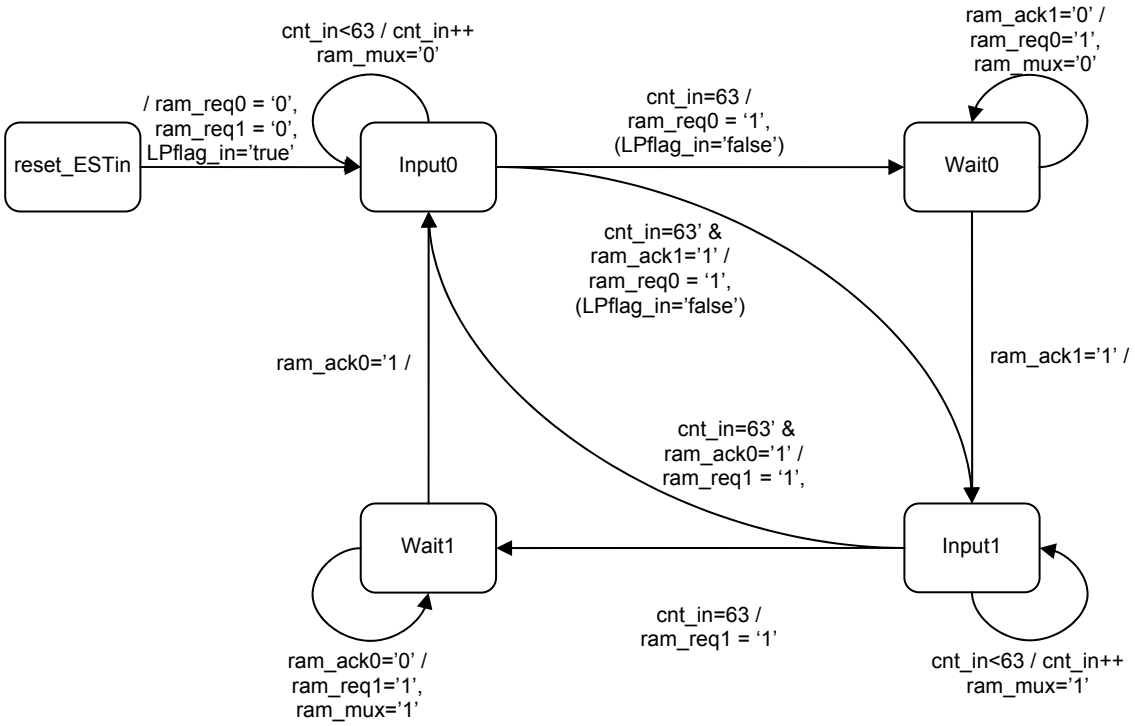


Figure 4.23: Channel-estimation input FSM, controls data-input phase and allocation of RAM resources

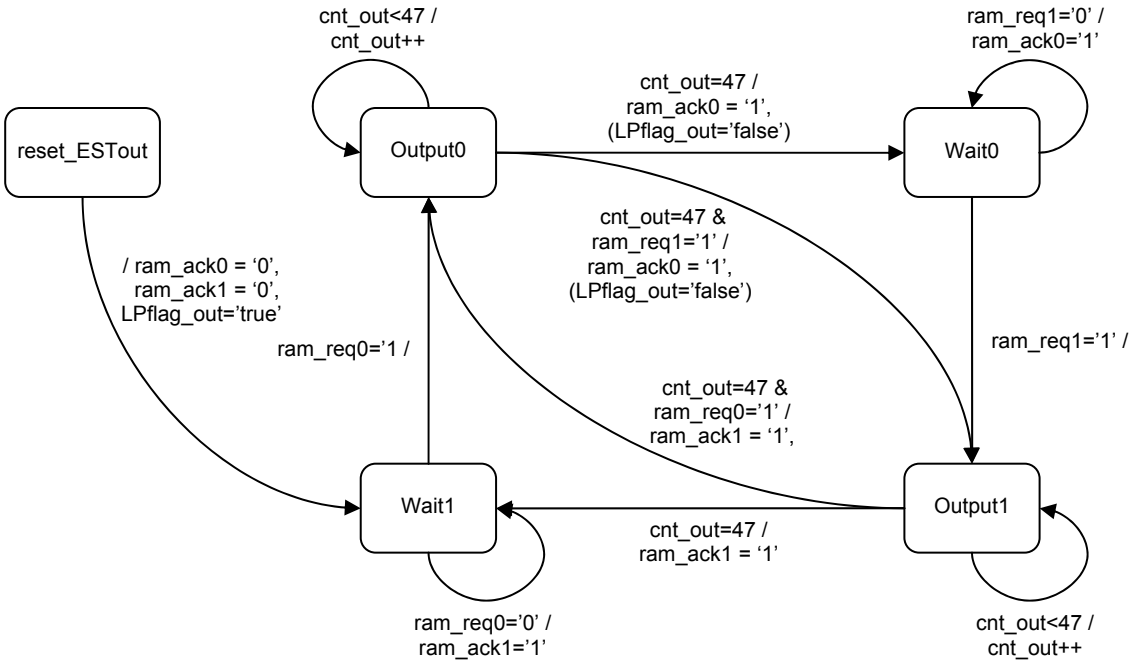


Figure 4.24: Channel-estimation output FSM, controls data-output phase and release of RAM resources

The output FSM controls the data-output phase as shown in Figure 4.24. If the input FSM indicates the availability of an input buffer RAM0/1 by assigning a logical '1' to ram_req0/1, the output FSM starts to write the 48 data subcarrier samples and the associated channel coefficients to the output. During the output process, the output address generator is responsible for the physical-to-logical address remapping. Following the output operation, the signal ram_ack0/1 is set to '1' to indicate that the associated memory is available for the next input data.

data in	$lp\_data_{(0\ldots63)}$	$payload_{(0\ldots63)}$	$payload_{(64\ldots127)}$	…	$payload_{(end-63\ \ldots end)}$	
data out		$c_{(0\ldots47)}$	$d\_out_{(0\ldots47)}$	$d\_out_{(48\ldots95)}$	…	$d\_out_{(end-47\ \ldots end)}$
ci out		$c_{(0\ldots47)}$	$c_{(0\ldots47)}$	$c_{(0\ldots47)}$	…	$c_{(0\ldots47)}$

Figure 4.25: Input-output timing and data contents

The input-output timing behavior and the data contents are shown in Figure 4.25. Note that the first output-data block contains the channel coefficients. The equalizer unit will then be used to calculate the required quantities $|c_i|^2$ from the first channel-estimator output-data block by using the complex multiplier implemented there (first symbol: $c_i \bullet c_i^*$, payload symbols: $data_i \bullet c_i^*$).

4.3.6 Equalization

A complex single-tap equalizer is used to compensate for the channel distortion. Every subcarrier signal is equalized (in amplitude and phase) by the associated channel coefficient c_i. The equalized modulation- symbol estimates are usually generated by

$$\text{data\_sym}_i = \frac{\text{data}_i}{c_i} = \frac{\text{data}_i \bullet c_i^*}{|\, c_i \,|^2}.$$

However, the division by $|c_i|^2$ is not performed to reduce the RX chain complexity. Instead, this operation has been moved into the log-likelihood computation unit (see Section 4.3.8), resulting in a simplified equalizer comprising only one complex multiplication per incoming data sample,

$$\text{data\_eq}_i = \text{data}_i \bullet c_i^* = \text{data\_sym}_i \bullet |c_i|^2.$$

As was shown in Figure 4.25, the first 48 input data from the channel estimator contain the channel coefficients. The complex multiplier is used in this mode to calculate the weighting factors $|c_i|^2$. These coefficients are stored in a memory and are subsequently output periodically in parallel with the associated equalized data values.

To reduce the complexity of the multipliers, the input data are first divided by 8 (reduction from 16 to 13 bit). Next, the input data and channel coefficients are limited to the range ±255, resulting in a data width of 9 bits. This limitation is possible without performance degradation, as the FFT operation results in a downscaling factor of 64 (amplitude downscaling by 8). The limitation still enables the compensation of rather large channel fluctuations in the range of $|c_i|_{max} = 5|c_i|_{perfect,}$ where $|c_i|_{perfect}$ stands for the theoretical channel coefficients with ideal power control and frequency flat channel characteristics.

The scaled and limited data and channel coefficients are multiplied using four 9×9 bit multipliers because the complex multiplication $\text{data\_eq}_i = \text{data}_i \bullet c_i^*$ can be decomposed as

$$\begin{aligned} \text{real}(\text{data\_eq}_i) &= \text{real}(\text{data}_i) \bullet \text{real}(c_i) + \text{imag}(\text{data}_i) \bullet \text{imag}(c_i) \\ \text{imag}(\text{data\_eq}_i) &= \text{imag}(\text{data}_i) \bullet \text{real}(c_i) - \text{real}(\text{data}_i) \bullet \text{imag}(c_i) \end{aligned}.$$

The results are scaled down by 4 to be presentable with 16-bit precision (15 bits for $|c_i|^2$, no sign bit).

4.3.7 SIGNAL FIELD FIFO for Processing Delay

The units following the equalizer stage need to know the data rate and length parameters of the data packet currently received. This information is encoded in the SIGNAL FIELD (see Figure 4.5). All further receiver processing starting with the log-likelihood computation needs to wait for the decoded SIGNAL-field information.

Therefore, a FIFO is introduced to buffer the equalized data and squared channel coefficients $|c_i|^2$.

The FIFO forwards the first 48 samples containing the SIGNAL field into the remaining RX chain. Afterwards, the output is disabled until the SIGNAL-field decoding has been completed. The latency is on the order of 150 cycles, where the main latency of 137 cycles is introduced by the VITERBI decoder (VD).

After deferring the data output for the predefined latency period, the data output process to the log-likelihood unit is re-enabled. As the FIFO and the processing units are able to work with 40 MHz, the latency can be reduced during the payload-decoding process.

4.3.8 Log-likelihood Ratio (LLR) Computation

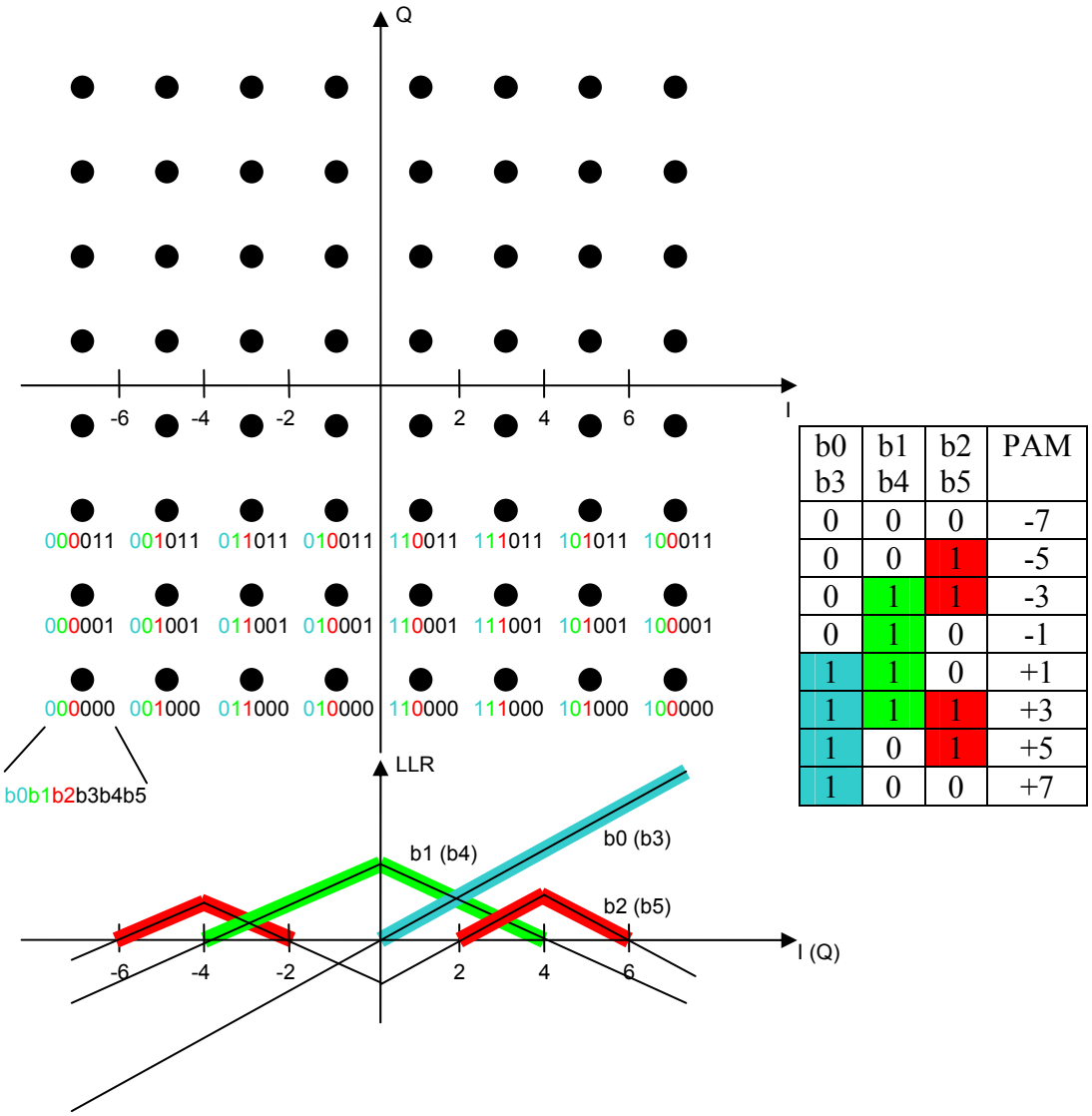
To enable soft-decision Viterbi decoding instead of a hard-decision de-mapping, a suitable LLR computation unit is required. In [9], a LLR computation has been presented, which allows the usage of a simple soft-input VD that need to be aware of several bit types with different importance.

The equalized samples at the input of the LLR computation unit correspond to weighted estimates of the transmitted modulation symbols,

$$\text{data\_eq}_i = \text{data\_sym}_i \bullet |c_i|^2.$$

Each complex input symbol corresponds to 1, 2, 4, or 6 transmitted bits, depending on the modulation mapping (see Table 4.4). The real and imaginary parts of the received symbols can be treated independently as 2-, 4-, or 8-PAM signals. Hence, each PAM symbol corresponds to 1, 2, or 3 bits. Note that for BPSK the imaginary part is not used. The modulation mapping (Gray encoding) between the bits and the PAM symbols and their corresponding amplitudes is visualized for 64QAM in Figure 4.26. We observe that the bits b0/b3 are always '1' for positive and '0' for negative symbol amplitude values. Hence, these bits determine the sign of the symbols and have the highest significance. The bits b1/b4 = '1' correspond to the PAM symbols with small magnitude (±1, ±3) and b1/b4 = '0' to symbols with large magnitude (±5, ±7). Hence, the bits b1/b4 are also called "first-magnitude bits" with medium significance. Finally, the bits b2/b5 distinguish the small-magnitude values and the large-magnitude values and represent the least significant bits.

The knowledge about the bit significance can be exploited in the LLR computation. As indicated by the blue line in Figure 4.26, the LLR corresponding to the sign bits b0/b3 can have larger likelihood ratio values than the LLRs for the less significant bits. This supports the fact that the correct sign of a received symbol can be detected with a higher reliability than the associated magnitude. The "first-magnitude" bits b1/b4 can have higher likelihood ratio values than the "second-magnitude" bits b2/b5, as indicated by the green and red lines.



b0 b3	b1 b4	b2 b5	PAM
0	0	0	-7
0	0	1	-5
0	1	1	-3
0	1	0	-1
1	1	0	+1
1	1	1	+3
1	0	1	+5
1	0	0	+7

Figure 4.26: Modulation mapping for Gray-encoded 64QAM and LLR plot showing the significance of the bit positions

In [9], the following LLR have been derived:

$$\Lambda^{(1,0)}(x) = |c_i|^2 \bullet x$$

$$\Lambda^{(2,0)}(x) = |c_i|^2 \bullet \begin{cases} 2(x+1) & , \quad \text{if } x < -2 \\ x & , \quad \text{if } -2 \le x \le 2 \\ 2(x-1) & , \quad \text{if } x > 2 \end{cases}$$

$$\Lambda^{(2,1)}(x) = |c_i|^2 \bullet (2 - |x|)$$

$$\Lambda^{(3,0)}(x) = |c_i|^2 \bullet \begin{cases} 4(x+3) & , \text{ if } x < -6 \\ 3(x+2) & , \text{ if } -6 \le x < -4 \\ 2(x+1) & , \text{ if } -4 \le x < -2 \\ x & , \text{ if } -2 \le x \le 2 \\ 2(x-1) & , \text{ if } 2 < x \le 4 \\ 3(x-2) & , \text{ if } 4 < x \le 6 \\ 4(x-3) & , \text{ if } 6 < x \end{cases}$$

$$\Lambda^{(3,1)}(x) = |c_i|^2 \bullet \begin{cases} 2(3-|x|) & , \text{ if } 0 \le |x| \le 2 \\ 4-|x| & , \text{ if } 2 < |x| \le 6 \\ 2(5-|x|) & , \text{ if } 6 < |x| \end{cases}$$

$$\Lambda^{(3,2)}(x) = |c_i|^2 \bullet \begin{cases} |x|-2 & , \text{ if } 0 \le |x| \le 4 \\ 6-|x| & , \text{ if } 4 < |x| \end{cases}$$

Note that $x \equiv \text{data\_sym}_i$ and $|c_i|^2 \equiv 1/\sigma^2_{w,i}$. Furthermore, the notation $\Lambda^{(q,d)}$ describes a LLR, where q is the number of bits associated to a symbol (q = 1: 2-PAM, q = 2: 4-PAM, q = 3: 8-PAM) and d is the bit number (d = 0: b0-"sign bit", d = 1: b1, d = 2: b2). It can be observed that for large values of x the slopes of the LLRs increase. However, the number of soft bits for the representation of the LLRs needs to be limited to 5. This in turn limits the dynamic range of the LLRs. Moreover, the LLR computation presented above requires many comparisons and is numerically rather complex.

To simplify the LLR computation, the following simplifications have been made:

- The piecewise linear LLR computation has been further linearized to prevent heavy usage of comparators.
- The multiplication with $|c_i|^2$ has been omitted for the data symbols because the normalization with this factors is skipped in the equalizer stage. Instead, the thresholds have to be adapted. This can be done without multipliers, as will be shown later.
- The quantization is done very efficiently without a bank of comparators using the number representation.

The LLR computation formulas have been simplified to

$$\Lambda^{(1,0)}(y) = y$$

$$\Lambda^{(2,0)}(y) = y$$

$$\Lambda^{(2,1)}(y) = 2|c_i|^2 - |y|$$

$$\Lambda^{(3,0)}(y) = y$$

$$\Lambda^{(3,1)}(y) = 4|c_i|^2 - |y|$$

$$\Lambda^{(3,2)}(y) = \begin{cases} |y| - 2|c_i|^2 & , \quad \text{if } 0 \le |y| \le 4|c_i|^2 \\ 6|c_i|^2 - |y| & , \quad \text{if } 4|c_i|^2 < |y| \end{cases}$$

with $y \equiv \text{data\_eq}_i = x \bullet |c_i|^2$. The threshold comparisons have been reduced to a minimum. Furthermore, the input data y appears only directly or as absolute value: all multiplications have been removed. The values $2|c_i|^2, 4|c_i|^2, 6|c_i|^2$ can be calculated by simple shift operations and one addition $6|c_i|^2 = 2|c_i|^2 + 4|c_i|^2$. Note that the scaling with the factor KMOD as defined in Table 4.5 is compensated for by moving this factor into the LLR computation as $\tilde{|c_i|}^2 = \text{KMOD(rate)} \bullet |c_i|^2$, resulting in one real multiplication.

Table 4.7: Soft input mapping for Viterbi decoder and associated quantization levels for LLRs saturated to [–1024, 1023]

Viterbi input	meaning	Quantization values (2's compl.)	quantization range
11111	strongest 1	01111xxxxxx	960…1023
11110	second strongest 1	01110xxxxxx	896…959
...	…	…	…
10000	weakest 1	00000xxxxxx	0…63
01111	weakest 0	11111xxxxxx	–64…–1
...	…	…	…
00001	second strongest 0	10001xxxxxx	–960…–897
00000	strongest 0	10000xxxxxx	–1024…–961

The resulting LLR values Λ need to be adapted to the input-mapping scheme of the Viterbi decoder unit. This task requires a mapping of 16-bit LLRs onto 5-bit Viterbi soft inputs, which are described in Table 4.7. First, the LLRs are saturated to [–1024, +1023] or, equivalently, to [10000000000, 01111111111] in 2's complement representation. This range has to be quantized to $2^5 = 32$ levels, resulting in $2048/32 = 64 = 2^6$ values per quantization level. Hence, the lowest 6 bits are not relevant for the quantization process. Therefore, the lowest quantization level corresponds to 10000xxxxxx (–1024 to –961) and the highest quantization level to 01111xxxxxx (960 to 1023). The quantization levels are also shown in Table 4.7. A comparison of the first and third columns reveals a very simple mapping scheme: To create the soft inputs it is sufficient to invert the sign bit of the saturated LLR value and then take the resulting 5 most significant bits as Viterbi input bits. This mapping procedure prevents heavy usage of comparators, which would be required for other quantization mappings. Depending on the modulation mapping, 1 (BPSK), 2 (QPSK), 4 (16QAM), or 6 (64QAM) soft bits are output in parallel for each complex input symbol.

4.3.9 De-Interleaving

The de-interleving unit inverses the operation carried out in the transmitter chain. Its structure is very similar to that of the interleaver described in Section 4.2.7. The major differences are the following:

- Instead of bits, LLR values, each consisting of 4 bits, are processed.
- Input: 1 to 6 LLR values are processed in parallel at the input. Permutation 2 is reversed on the fly at the input and the LLRs are stored in memory (see interleaver for memory map).
- Output: 4 LLR values are produced in parallel at the output.

4.3.10 De-Puncturing

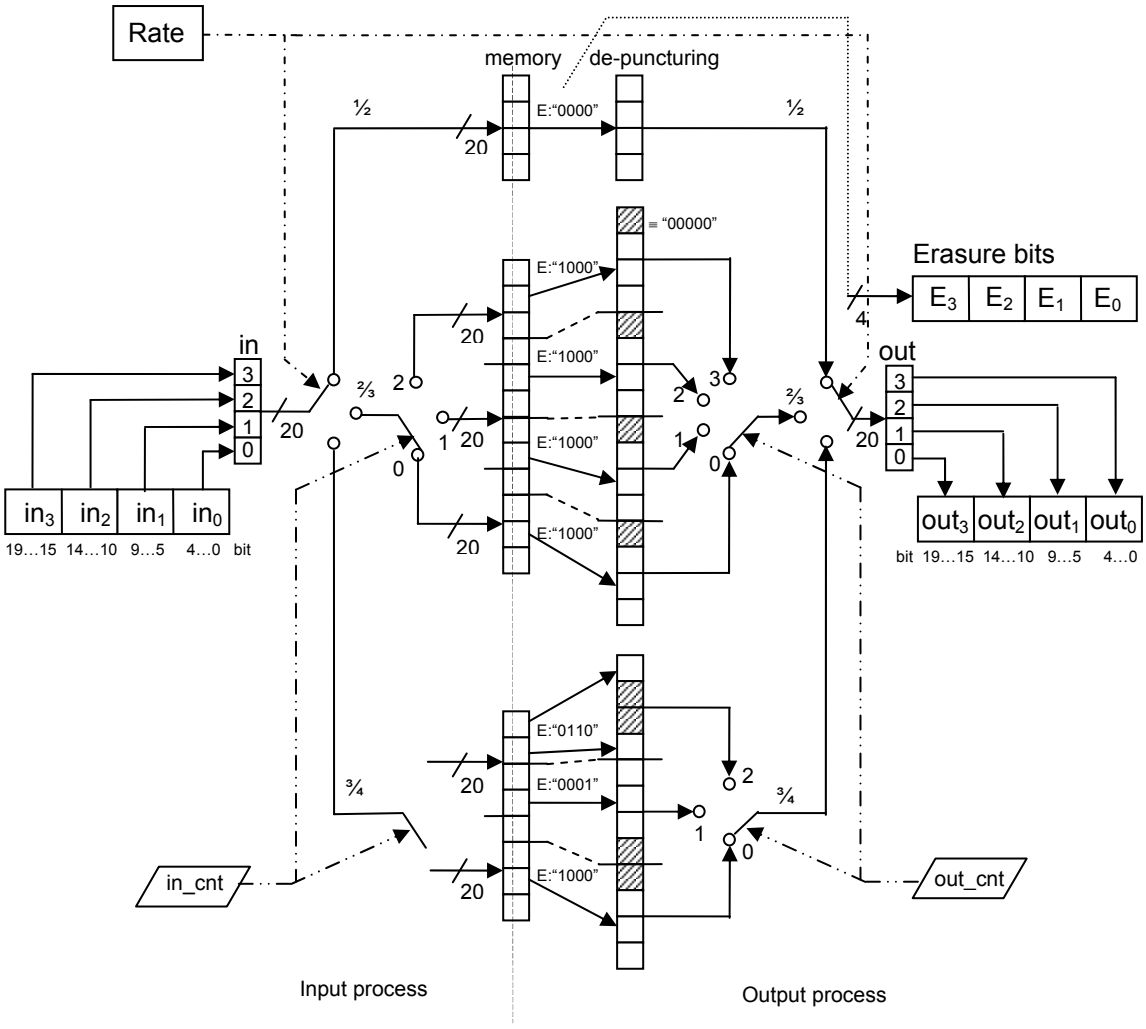


Figure 4.27: De-puncturing unit, parallel implementation, output erasure bit E:"xxxx" in parallel with associated soft bits

A certain number of encoded bits have been removed from the data stream during the puncturing process in the transmitter (see Section 4.2.6). The de-puncturing unit has to re-introduce these erased bits into the data stream before Viterbi decoding to regenerate the coding rate ½. The implementation of the de-puncturing process is shown in Figure 4.27. However, the erasure bits are unknown to the receiver. Therefore, a neutral soft bit has to be inserted. As no neutral soft bit is defined in the mapping table (Table 4.7), the inserted bits are labeled by additional erasure bits, which are output in parallel with the associated soft bits. The generation of the erasure bits is depicted in Figure 4.27 as E:"xxxx", where E:"1000" corresponds to the insertion of a soft bit at output out_3. The inserted soft bits are marked as hatched rectangles and filled with the value "00000".

Four soft bits are fed into the Viterbi decoder concurrently with the four associated erasure bits. Note: The first OFDM symbol (= SIGNAL field) is always transmitted with 6 Mbps, which corresponds to BPSK with coding rate ½. Therefore, the first 48 encoded bits are always treated correspondingly; the RATE information fed into the de-puncturing unit is not used for the SIGNAL field bits.

4.3.11 Viterbi Decoding

The Viterbi decoder unit is implemented as a core [10] with a fixed set of parameters:

- Code with constraint length 7, rate ½ and generators $(133)_8$ and $(171)_8$;
- soft input bits;
- decision depth equal to 64;
- radix-4 architecture with 64 ACS processors;
- de-puncturing unit interface present;
- BER estimation unit not present.

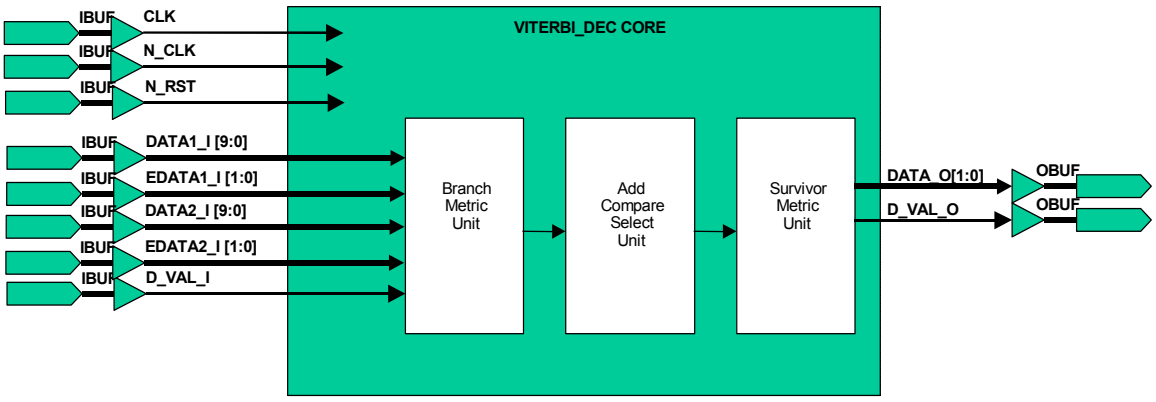


Figure 4.28: Viterbi decoder core block diagram

A general block diagram of the core is shown in Figure 4.28. The four input soft bits are provided as two pairs at the inputs DATA1_I and DATA2_I. Equivalently, the four input erasure bits are provided as two pairs at the inputs EDATA1_I and EDATA2_I.

Note that the generator polynomials in the TILAB core are $(171\ 133)_8$, whereas 802.11a uses $(133\ 171)_8$ so that the de-punctured data and the erasure bits have to be swapped at the VITERBI decoder input to fit to the associated generator polynomial. The output consists of 2 decoded bits in parallel at port DATA_O.

The Viterbi core is controlled by an FSM as depicted in Figure 4.29. After reset, the first 48 data values (24 encoded soft bits) are clocked into the Viterbi core. Next, a number of dummy bits have to be clocked into the core, before the decoded data are available at the decoder output. This is done by repeating the last data values and by labeling these data as erasure bits. The latency consists of a decoding delay Ddecod = 128 clock cycles and an additional processing delay Dproc = 9 clock cycles. After this latency, the SIGNAL field has been completely decoded. Hence, the transmission parameters data rate (Prate) and length (Plength) are available for the RX chain. The Viterbi core has to be reset, before the SERVICE and DATA fields can be decoded. This is done by assigning VITnreset = '0' to the port N_RST. Next, the SERVICE field and the DATA field of length Plength*8bit are clocked into the core. These values have been stored in the SIGNAL field FIFO until the data rate and packet-length decoding process has been completed (see Section 4.3.7). Finally, to clock out the decoded bits, 128 dummy values are clocked into the core by labeling these values as erasure bits.

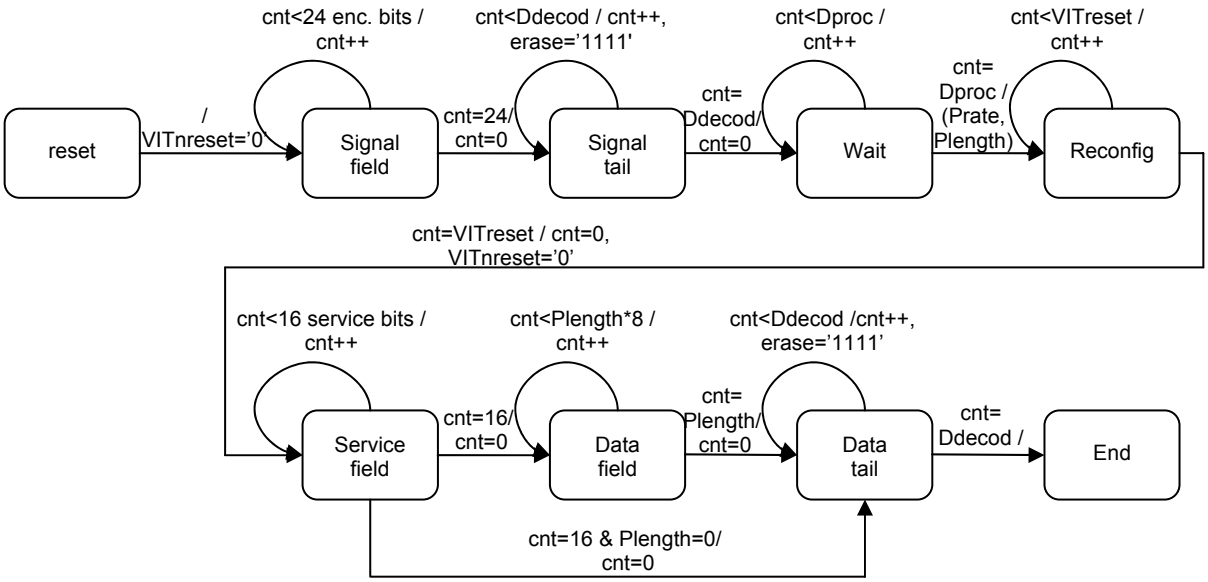


Figure 4.29: Viterbi decoder control FSM

4.3.12 Descrambling

The entire DATA field including SERVICE, PSDU, TAIL and PAD bits, has been scrambled in the transmitter (see Section 4.2.4); the SIGNAL field remains unscrambled. In the RX chain, the same scrambler as described in Figure 4.7 for the TX chain can be used.

Random values are used to initialize the scrambler registers. However, the initialization values are not known in the RX chain. Therefore, these values have to be determined on the fly in the descrambler unit. This initialization can be done as follows:

- Register value a_0 as depicted in Figure 4.6 is updated in every clock cycle with $a_0 = a_3 \oplus a_6$.
- The first 7 bits of the SERVICE field are 0's. Hence, for the first seven clock cycles the scrambled data output can be written as $\text{data\_out} = a_3 \oplus a_6 \oplus 0 = a_0$. The first seven scrambled bits are identical to the register values for any initialization.
- In the receiver, a correct initialization of the descrambler requires that the registers contain the same bit sequence as described above at the end of the SERVICE field. Hence, the initialization of the descrambler can easily be done by clocking the seven bits containing the (scrambled) SERVICE field into the descrambler registers as shown in Figure 4.30.
- The descrambled output can be determined as $\text{data\_out} = y \oplus \text{data\_in}$. The output for the SERVICE field is determined by $\text{data\_in} \oplus \text{data\_in} = 0$.
- The parallel implementation can be derived directly from Figure 4.7 by introducing the SERVICE field switch, resulting in Figure 4.31. Note that in the parallel implementation one transition state has to be introduced. While the last (7's) SERVICE field bit $b_i(0)$ is clocked into register a_1, the switch for the first RESERVED field bit $b_i(1)$ already has to be switched into the normal descrambling mode.

Note also that the first 24 bits of the SIGNAL field are directly forwarded to the output by skipping the descrambler.

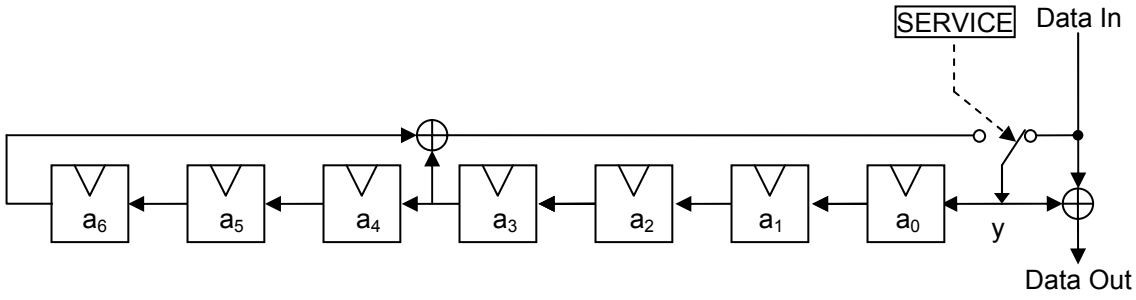


Figure 4.30: Data descrambler, serial implementation

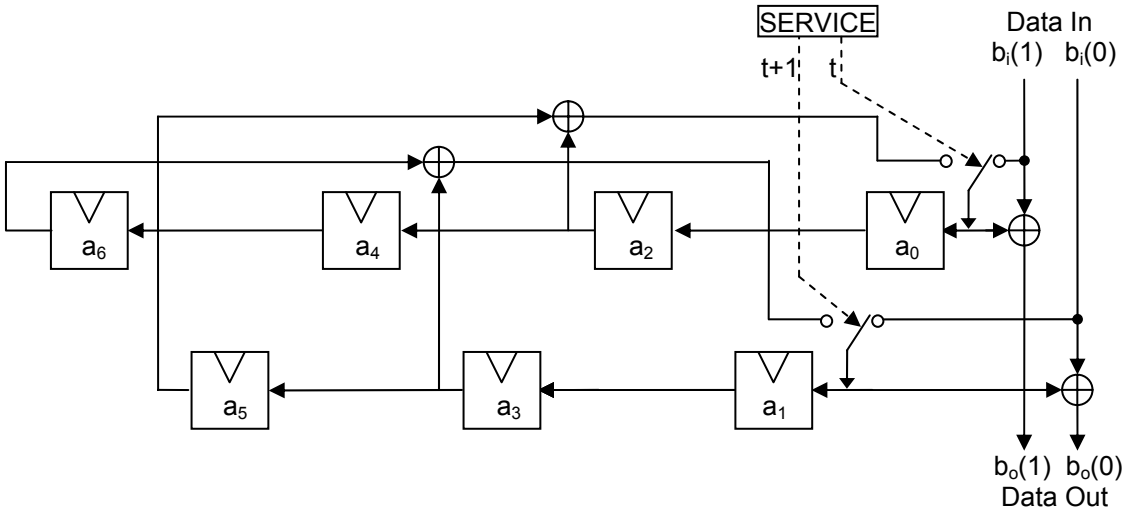


Figure 4.31: Data descrambler, parallel implementation

4.3.13 Packet Decomposition

After descrambling, the packet-decomposition unit separates the received PPDU frame according to its structure, which was shown in Figure 4.5. The FSM of this unit is depicted in Figure 4.32. In the first part, the fields RATE, RESERVED, LENGTH are stored in registers. These values are fed back into the RX chain using the ports Prate, Preserved, and Plength. A parity check is executed over these fields, and the result is output to port Pparity. After the determination of the parity bit, an additional signal DETECT0 is raised to inform the PHY-RX state machine.

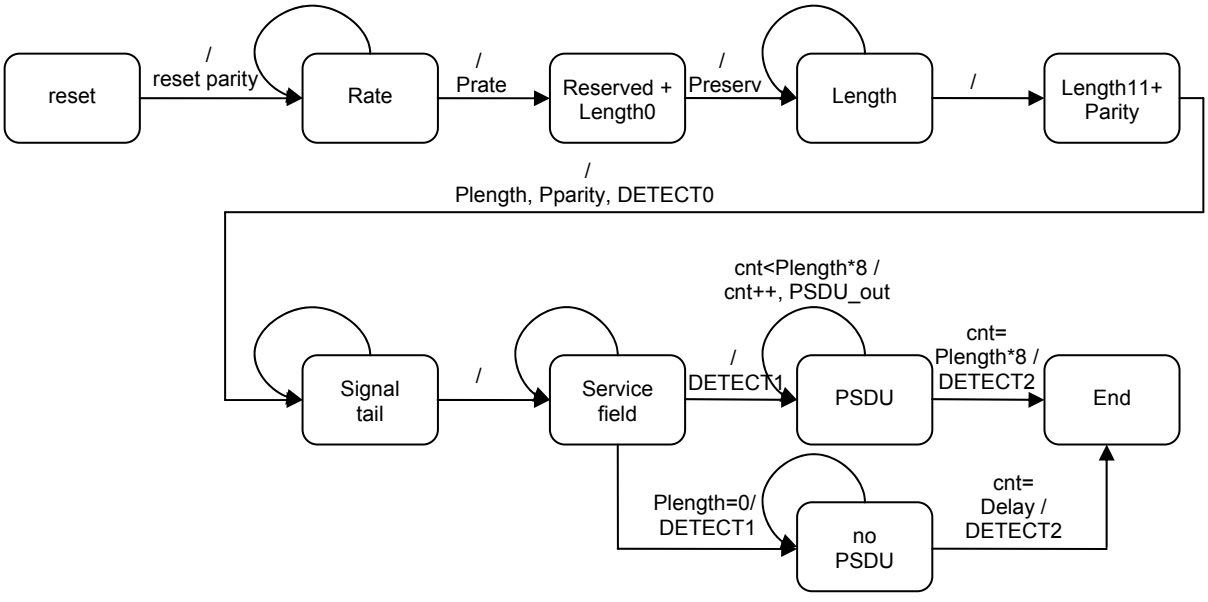


Figure 4.32: Packet-decomposer FSM

Next, after the removal of the SIGNAL tail bits, the SERVICE field is saved into registers. The end of the SERVICE field or, equivalently, the start of the PSDU is indicated by the signal DETECT1. Finally, the PSDU is output. The end of the PSDU is indicated by the signal DETECT2. The signals DETECT0…DETECT2 are forwarded to the MAC layer to indicate the actual status of physical data reception.

4.3.14 CRC Check

The packet decomposer outputs the entire PSDU or MAC frame. Before the frame is passed to the MAC layer, the CRC check is done in the PHY hardware to save computational power in the MAC layer. The CRC value is generated exactly in the same way as in the transmitter (see Section 4.2.2). The input bits are stored in a memory until the next byte is formed. Next, the CRC update is performed (the input bytes have to be reflected MSB⇔LSB). After the reception of the entire PSDU including the FCS field, the resulting CRC value has to be compared bitwise with the so-called magic number 0xc704dd7b, which describes the expected remainder polynomial,

$$C(x) = \underset{\substack{bin\\hex}}{x^{31}} + \underset{\substack{1100\\c}}{x^{30}} + x^{26} + \underset{\substack{0111\\7}}{x^{25}} + x^{24} + \underset{\substack{0000\\0}}{} + \underset{\substack{0100\\4}}{x^{18}} + x^{15} + \underset{\substack{1101\\d}}{x^{14}} + x^{12} + x^{11} + \underset{\substack{1101\\d}}{x^{10}} + x^{8} + x^{6} + \underset{\substack{0111\\7}}{x^{5}} + x^{4} + x^{3} + \underset{\substack{1011\\b}}{x+1}.$$

If the bitwise comparison result is zero, the correctness of the CRC check is indicated to the MAC by a signal P_CRC_OK = ‘1’.

Note that the CRC check unit performs a word alignment at its output. If the PSDU length is not a multiple of 32 bits, the unit appends the necessary number of 0’s at the end of the frame. This operation is required because the subsequent FIFO (interface to MAC) works on a word basis rather than byte-wise.

4.4 PHY Latency Considerations

In the following subsection, some fundamental latency considerations for the physical layer implementation are discussed, which are essential to meet the allowable PHY characteristics as defined in the 802.11a standard [1, Table 93]. The critical times that have to be met by the PHY layer are the SIFS (short inter-frame spacing) time and the CCA (clear channel assessment) time. Here, we only concentrate on the SIFS time. The SIFS is the time from the end of the last symbol of the preceding frame to the beginning of the first symbol of the preamble of the subsequent frame as seen at the air interface. This is shown in Figure 4.33, where a data reception has to be acknowledged after SIFS. The SIFS includes the following delays:

SIFS = RxRFDelay + RxPLCPDelay + MACProcessingDelay + RxTxTurnaroundTime.

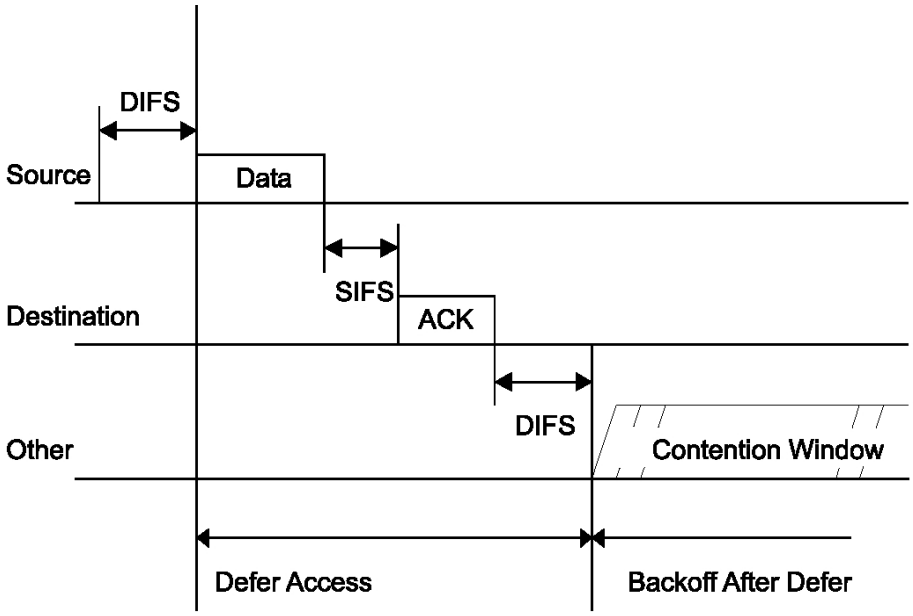


Figure 4.33: DCF timing details

In the following, the delay (RxPLCPDelay) introduced by the receive chain units is described in detail.

There is only a few clock cycles' worth of latency introduced by the RX-filter/decimator. In the inner receiver, the incoming data have to be stored in a tapped delay line until the timing synchronization is done. The introduced latency can be removed by using a FIFO concept, where the output data towards the FFT are written at twice the input rate (40 MHz). The FFT compute time is T_{1_FFT} = 3.65 µs, the output buffering takes another T_{2_FFT} = 1.6 µs (64 cycles @ 40 MHz). These times are currently fixed because a predefined Xilinx FFT core form is used. To be able to calculate an FFT every 4 µs, two FFTs are instantiated and used alternatingly (see Section 4.2.9) in the current implementation. The resulting FFT latency is T_{FFT} = 5.25 µs. The de-interleaver has to read 48 data @ 40 MHz from the input before being able to output data, which introduces a latency of T_{1_IL} = 1.2 µs. The data output will take T_{2_IL} = 0.3 ... 1.8 µs (BPSK: 48 data / 4 parallel streams @ 40 MHz ...; 64QAM: 288 data / 4 parallel streams @ 40 MHz). All units between the de-interleaver and the Viterbi decoder process 4 data streams in parallel to decrease the latency and to feed the radix-4 Viterbi core. The resulting de-interleaver latency is $T_{IL} \leq 3$µs. The de-puncturing unit expands data by a maximum factor of 3/2 for rate 3/4. Hence, the maximum latency introduced by the de-puncturing unit is T_{PUNCT} = 0.9 µs. The Viterbi decoder uses a radix-4 architecture with a decision delay of 64 cycles. The core introduces a delay of 2 × 64 + 8 cycles, resulting in T_{VIT} = 3.4 µs. The CRC check is done on the fly and does not introduce additional latency. The overall worst case delay of the baseband implementation is currently

$$T_{FFT} + T_{IL} + T_{PUNCT} + T_{VIT} = 5.25\ \mu s + 3\ \mu s + 0.9\ \mu s + 3.4\ \mu s = 12.6\ \mu s.$$

Hence, the implementation meets the SIFS timing requirements, leaving about 3 µs for the transmit preparation of the ACK frame.

However, the latency can still be reduced considerably by implementing the following changes:

- The use of a more appropriate custom FFT:
  - one FFT with twice the number of butterflies reduces T_{1_FFT} by 50% to 1.825 µs,
  - use of a Decimation-in-Time FFT to eliminate T_{2_FFT} completely because the input data reordering can be done on the fly.
- The shortening of the Viterbi core decoding delay to 32 (instead of 64) is possible with marginal performance loss and reduces the latency to $T_{VIT32} = 1.7$ µs.

Introducing these changes would result in a PHY latency of approximately 7.5 ... 8 µs.

Another approach to reduce the latency of the receive chain is to increase the FPGA clock speed. However, the current clock speed of 40 MHz was chosen under the following constraints:

- The RF frontend delivers a 40-MHz clock, which is used for the AD/DA-converters and as symbol clock frequency reference.

Higher clock frequencies will cause problems for the place-and-route algorithm and would require more algorithm pipelining.

5 Radio Frontend Implementation Issues

The radio frontend of the mobile station prototype of the ZRL high-speed wireless LAN has been developed in a joint project with the Laboratory for Electromagnetic Fields and Microwave Electronics of the Swiss Federal Institute of Technology (ETH) Zurich [11]. The radio frontend consists of the RF interface board, which is attached via a control interface and two data interfaces to the digital baseband implemented in the FPGA, and a set of commercially available analog components such as low-noise amplifier, power amplifier, synthesizer, filter, etc. The frontend is compatible with the 802.11a specification.

In this Section, we describe the radio frontend architecture, specify the interface between the digital baseband and the frontend, and discuss some filter design issues.

5.1 Radio Frontend Architecture

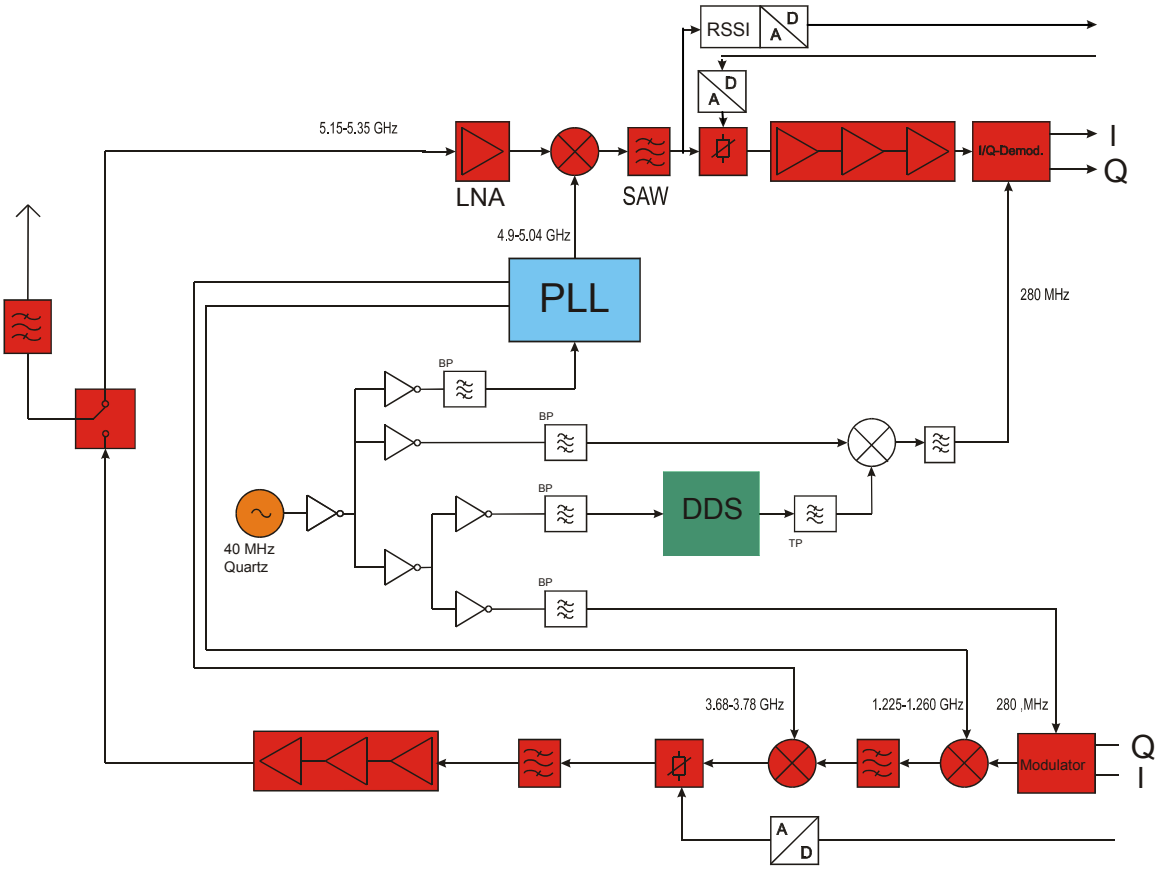


Figure 5.1: 802.11a radio frontend architecture

The 802.11a-compliant analog radio frontend design applies a heterodyne radio architecture (see Figure 5.1). It can be operated in one out of the eight 20-MHz-wide

channels in the lower and middle U-NII frequency band between 5.15 and 5.35 GHz. Switching between the channels can be performed within a fraction of 1 μs.

By default, the radio frontend is operated in the receive mode. The analog receive path has been designed with the aim to operate with a minimum input signal sensitivity of –90 dBm, to provide a wide dynamic range of 60 dB, and to keep the phase noise and inter-modulation distortion at a low value. As shown in Figure 5.1, any incoming signal at the antenna is first filtered and passed via the RX-TX antenna switch to the low-noise amplifier MGA-86576 [12] that amplifies the signal by 20 dB. Then the amplified RF signal is down-converted with a mixer to the intermediate frequency (IF) band at 280 MHz. The carrier signal necessary depends on the channel selected and is provided by the PLL frequency synthesizer. Next the IF signal is filtered with a SAW filter with a bandwidth of 17 MHz. Its output signal is monitored by a power detector (logarithmic amplifier AD8310 [13]) to derive the received signal strength indicator (RSSI) value; this value is converted with a 6-bit flash A/D converter CA3306 [15] and passed via the RF interface board to the baseband-processing unit. The filtered IF signal is then scaled in amplitude with several amplifiers and attenuators. Two adjustable FET attenuators adjust the amplitude of the signal in the range from –17 dB to 46 dB; the gain value is set according to a control voltage that is determined by the baseband-processing unit and generated with a 10-bit D/A converter [13]. The filtered and scaled IF signal is fed to the I/Q demodulator that down-converts the IF signal to the in-phase and quadrature component of the complex baseband signal. This is accomplished by splitting the IF signal into two signals with identical phase characteristic. Two identical mixers are then used to down-convert the components to the baseband by using a synthesized 280-MHz clock signal and a 90º-phase-shifted version of it, respectively. The I and Q components of the signal are then fed to the RF interface board for A/D conversion and further processing.

At the request of the physical-layer FSM, the radio frontend can be switched into transmit mode. The transmit path has been designed with the goal of transmitting a radio signal with a power of 200 mW in one of the 20-MHz-wide 5-GHz frequency bands and keep the phase-noise, inter-modulation distortion, and spurious signal content at a low level. When the frontend is transmitting, the I and Q components of the complex baseband signal are up-converted with the I/Q modulator to the IF band at 280 MHz, and then in two further up-conversion steps to the target frequency band between 5.15 and 5.35 GHz. The pre-defined nominal signal power level is adjusted with digitally controlled attenuator circuits and the power amplifier MAAM26100-P1 [14]. To convert the digital signals “AGC control” and “transmit-power control” to analog signals, low-cost D/A converters [13] with a latch circuit in front were selected. The radio signal generated is then fed via the RX-TX antenna switch and filter to the antenna and transmitted over the air.

To satisfy the 802.11a physical layer specification, the frequencies of all carrier signals are derived from a single 40-MHz quartz oscillator. The Analog Devices ADF4112 RF PLL Frequency Synthesizer and the AD9850 Direct Digital Synthesizer (DDS) generate all carrier signals that are required to convert the baseband signal via the

IF band of 280 MHz to the target band and vice versa [13]. The PLL and the DDS are controlled by the baseband signal-processing unit via the radio frontend interface.

5.2 Radio Frontend Interface

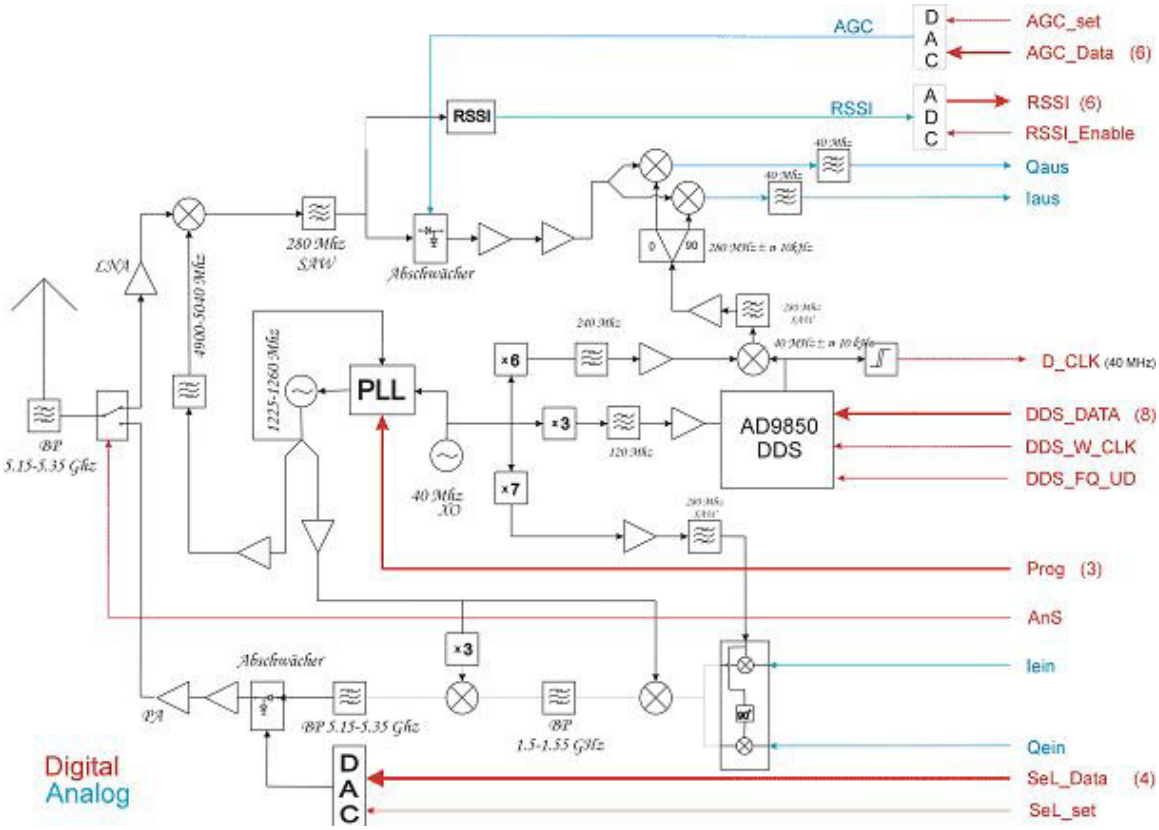


Figure 5.2: 802.11a Radio frontend with interface signals

Figure 5.2 shows the block diagram of the analog radio frontend with all interface signals. The interface comprises 38 digital and 4 analog signals, which are used to control the radio frontend and exchange data between the digital baseband and the analog radio frontend components. The analog and digital radio-interface signals are defined in Tables 5.1 and 5.2.

Table 5.1: Digital radio-frontend interface signals

Signal Name	**Number of Pins**	**Remarks**
AGC_set	1 input	Automatic Gain Control value set
AGC_data[9..0]	10 inputs	AGC value
RSSI_enable	1 input	RSSI DAC clock
RSSI[5..0]	6 outputs	Received Signal Strength Indicator
D_CLK	1 input	RSSI DAC clock
DDS_fq_ud	1 input	DDS frequency update
DDS_w_clk	1 input	DDS write clock
DDS_data[7..0]	8 inputs	DDS register data
PROG[2..0]	3 inputs	PLL programming
AnS	1 input	Antennea Switch
SeL_set	1 input	Transmit Power set
SeL_data[3..0]	4 inputs	Transmit Power value
Total signals	**38**	**Digital in- and outputs**

Table 5.2: Analog radio-frontend interface signals

Signal Name	**Range / DC Offset**	**Remarks**
Iaus	2[Vpp] / 2[V]	Receiver I signal, impedance =50 Ohm
Qaus	2[Vpp] / 2[V]	Receiver Q signal, impedance = 50 Ohm
Iein	1.2[Vpp] / 1.5[V]	Transmitter I signal, impedance = 50 Ohm
Qein	1.2[Vpp] / 1.5[V]	Transmitter Q signal, impedance = 50 Ohm

Figure 5.3 shows the block diagram of the radio interface that connects the analog 5-GHz radio frontend (RFE) to the digital baseband signal-processing unit implemented in the FPGA. The interface consists of several A/D and D/A converters, analog and digital low-pass filters, up- and down-sampling devices, and some analog and digital signal-conditioning circuits. The converters and all required analog components have been implemented on the RF interface board, whereas all digital processing functions have been integrated into the XILINX FPGA.

The A/D and D/A conversions are performed with the Dual D/A Converter AD9763 and Dual A/D Converter AD9238 [13] at a rate of 40 MHz. The sampling rate is thus twice the Nyquist rate and, therefore, two-times oversampling is applied. The in-phase and quadrature components of the digital transmit signal are represented with 10 bits, whereas the components of the received signal are quantized with 12 bits. Up-sampling of the digital baseband signal from a 20-MHz sampling rate to 40 MHz and down-sampling of the oversampled receive signal to the Nyquist rate is implemented in the FPGA as an additional signal-processing unit in the TX and RX chain, respectively.

The low-pass filters are required to shape the signal spectrum, perform signal smoothing, and avoid aliasing effects. Identical filter functions are implemented in the corresponding in-phase and quadrature channels of the transmit and receive paths;

especially in OFDM-based systems, any imbalance in the two channels would lead to a severe performance degradation [16]. In addition, we decided to use the same filters in the transmit and receive path to advantageously re-use hardware resources.

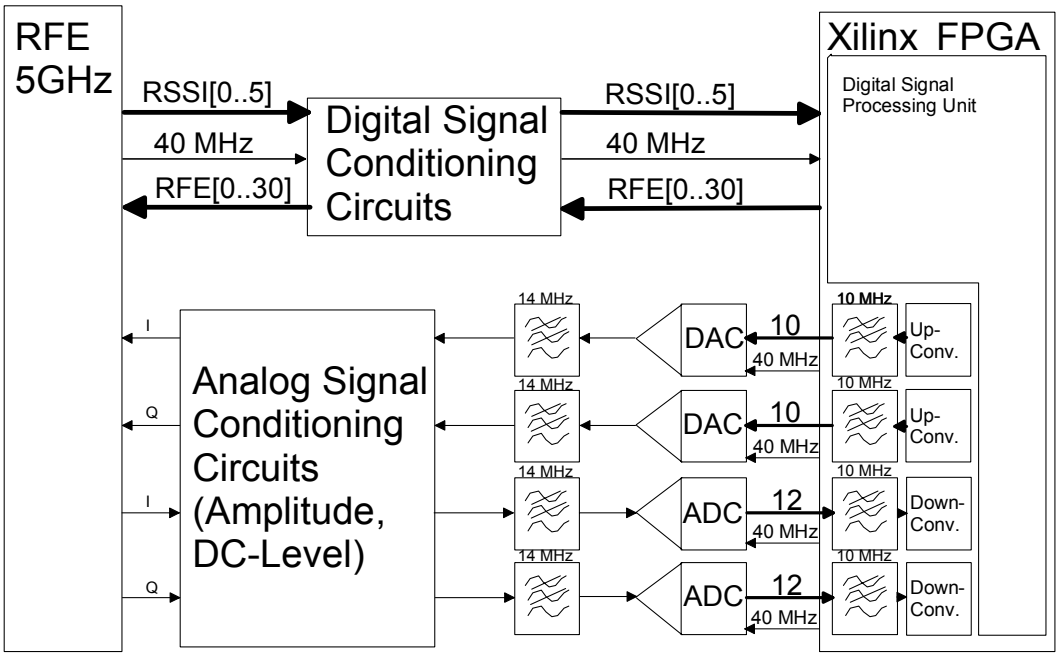


Figure 5.3: Radio interface block diagram

Because oversampling is applied, each filter function can be implemented with partly analog and partly digital means. This allows the implementation of the filter function by means of a commercially available analog filter with relaxed filter requirements and a digital filter that mainly determines the overall characteristic. As the main part of the filter is implemented digitally, exact copies of the digital filter can be provided in the corresponding in-phase and quadrature channels, thus avoiding the introduction of a severe I/Q imbalance.

The analog filter is a seventh-order Chebyshev filter with a ripple of less than 0.1 dB and a –3 dB cutoff frequency at 14 MHz. The group delay deviation is less than 15 ns at frequencies below 9 MHz. The digital filter is a programmable 19-tap half-band FIR filter with a distributed arithmetic filter architecture. Further characteristics of the filter are two-times oversampling, a latency of 8 clock cycles, and the use of a Hanning window.

To eliminate DC offsets on the received analog signals, a first-order 50-kHz high-pass filter is inserted between the radio frontend and the A/D converters. Moreover, the input/output standard of all digital signals is LVTTL (3.3V) compliant. To satisfy the requirements of LVTTL levels for all digital signals, some conditioning circuits are necessary, as indicated in Figure 5.3

6 Summary

To evaluate the potential of the broadband WLAN technology for next-generation mobile communication systems, a WLAN testbed with three mobile stations has been designed and built at the IBM Zurich Research Laboratory. The physical layer of the WLAN has been implemented in accordance with the IEEE 802.11a specification. Therefore, OFDM has been applied as basic transmission technology, and the radio frontend has been developed for operation in the 5-GHz frequency band.

In this document, we have disclosed the architectural design of a mobile station that consists of a radio frontend, digital baseband/MAC unit, and additional peripheral test and debug equipment. We have briefly described the mapping of these units to hardware components that are available on a set of ARM software/hardware development boards. Emphasis, however, has been given to the description of the OFDM-based physical layer architecture. We have specified the PHY/MAC interface with service primitives and described the basic behavior of the physical layer by means of the basic receive and transmit procedure.

The major portion of the report has been devoted to the implementation of the digital baseband. Hardware implementations of all digital signal-processing algorithms required to transmit and receive 802.11a OFDM packets over the air interface have been given. Moreover, we have discussed the communication mechanism applied between the functional units. The design of all functional units has been tailored so that all units fit into a commercially available FPGA and the stringent latency requirements of the 802.11a standard can be satisfied.

Finally, a 5-GHz radio frontend implementation has been presented that has been jointly developed with engineers from the ETH Zurich.

7 References

[1] "IEEE P802.11a/D7.0 - 1999, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification: High Speed Physical Layer in the 5 GHz Band," IEEE, New York, July 1999.

[2] R. van Nee and R. Prasad, "OFDM for Wireless Multimedia Communications," Artech House, Boston and London, February 2000.

[3] "IEEE Standard 802.11 - 1997, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification," IEEE, New York, November 1997.

[4] Data Sheets and User Guides for ARM Integrator/AP, Integrator/CM920T, and Integrator/LM-XCV600E+, http://www.arm.com/

[5] Xilinx Product Specification: High-Performance 64-Point Complex FFT/IFFT V1.0.5, http://www.xilinx.com/

[6] P. Coronel, "Implementation of Digital Signal Processing Algorithms for an OFDM-Based Broadband Wireless Local Area Network," IBM Research Report, RZ 3428, January 2003.

[7] P. Coronel, S. Furrer, J. Jelitto, D. Maiwald, W. Schott, and B. Weiss, "Acquisition and Adjustment of Gain, Receiver Clock Frequency, and Symbol Timing in an OFDM Receiver," CH8-2002-0053, submitted to EPO, December 2002.

[8] S. Furrer and D. Dahlhaus, "Mean Bit-Error Rates for OFDM Transmission with Robust Channel Estimation and Space Diversity Reception," in Proc. of the 17th Int'l Zurich Seminar on Broadband Communications 2002 (IEEE, Piscataway, NJ, 2002), pp. 47-1 - 47-6, February 2002.

[9] F. Neser, "OFDM: Coding Aspects," IBM Internal Report, March 2000.

[10] Telecom Lab Italia: VIP library data sheet, Viterbi Decoder Macro, http://www.telecomitalia.com/

[11] M. Stadler, "The 802.11a Radio Frontend Project", Preliminary Technical Report and Presentation, ETH Zurich, Switzerland, August 2003.

[12] Data sheet for MGA-86576 SMT Amplifier, Agilent, http://www.agilent/

[13] Data sheets for ADF4112 RF PLL Frequency Synthesizer, AD9850 125 MHz complete DDS Synthesizer, AD1161 10-bit Monolithic D/A-Converter, AD8310 Fast Voltage-Out DC-440 MHz 95 dB Logarithmic Amplifier, 10-bit 125 MSPS Dual D/A Converter AD9763, 12-bit 65 MSPS Dual A/D Converter AD9238, Analog Devices, http://www.analog.com/

[14] Data sheet for MAAM26100-P1 Power Amplifier, M/A-COM, http://www.macom.com

[15] Data sheet for 6-bit 15 MSPS, Flash A/D Converter, Intersil, http://www.intersil.com/

[16] S. Furrer, J. Jelitto, W. Schott, and B. Weiss, "Modulation and Demodulation of OFDM Signals," CH8-2003-0081, submitted to EPO, January 2004.