Master's Thesis

Encoder and Decoder Design of LTE Physical Downlink Control Signals

Lian Xiangyu

Department of Electrical and Information Technology,
Faculty of Engineering, LTH, Lund University, January 2014

Encoder and Decoder Design of LTE Physical Downlink Control Signals

Lian Xiangyu

01/2014

Supervisor: Michal Stala

Abstract

As mobile communication technologies have evolved rapidly during the last decades, wireless communication standards such as LTE are developed with the need of new services on mobile devices. In this thesis, the design and implementation of the encoder and decoder of LTE physical-layer downlink control signaling, which provides control information for downlink and uplink transmission, are presented. The encoder is designed to encode downlink control information with multiple configurations while decoder is designed to decode the encoded data and provide control information for the terminal. The encoder is implemented in MATLAB and decoder is implemented in MATLAB as well as an FPGA platform from Xilinx. The Xilinx evaluation platform uses a serial port to communicate with a computer. The encoded data can be transmitted through HyperTerminal to the FPGA and decoded results can be displayed on HyperTerminal. The Viterbi decoder is implemented on the FPGA using a Xilinx IP while other functions are implemented in software. The software is run on a Microblaze processor.

Acknowledgements

This master thesis is the most complicated and honorable achievement during my study as master student in System-on-Chip. It cannot be done without my patient supervisor Michal Stala who would like to spend time and go through problems with me. Thanks to Joachim Rodrigues and other people at EIT who provided me equipment or granted me access to equipment. Besides, thanks to my friends who have always been encouraging and cheering me up. Finally, thanks to my parents who are eagerly expecting me to go home provided me the funds to study at Lund University.

Table of contents

1 Introduction

From a radio communication experiment by Guglielmo Marconi in 1890s to a common tool that modern life cannot perform without, mobile communication has shown the world its contribution and development potential in modernization. Especially in the last decades, it has evolved from a high cost technology that serves the few to globally used systems which changed billions of people's lives.

Mobile communication technologies can be divided into generations by evolution milestones. The first generation, referred as 1G, used analog mobile radio systems in 1980s. The usage of the digital mobile systems started 2G, and in 3G mobile systems start to handle broadband data. The 4G started when Long-Term Evolution (LTE) release 10, also referred as LTE-Advanced, was introduced and still evolving. The development of LTE is undertaken by global standards-developing organizations such as the Third Generation Partnership Project (3GPP), which also contributed in the development of 3G system [1].

1.1 Introduction to LTE

LTE is a wireless communication standard for mobile devices and data terminals. The evolution from 3G systems to LTE is driven by the development of mobile devices and the need for more services on mobile devices.

The development of mobile devices is based on the evolution of transistors according to Moore's law which results in processors and memories have better performance with reduced size, cost and power consumption. The need for more services on mobile devices is a result of the rapid spreading usage of the internet. People want Internet Protocol based services approachable from their mobile devices through the mobile broadband. This requires the LTE to have better performance in data rate, delay and capacity. Figure 1 illustrates the capabilities of 3G and 4G communication systems.

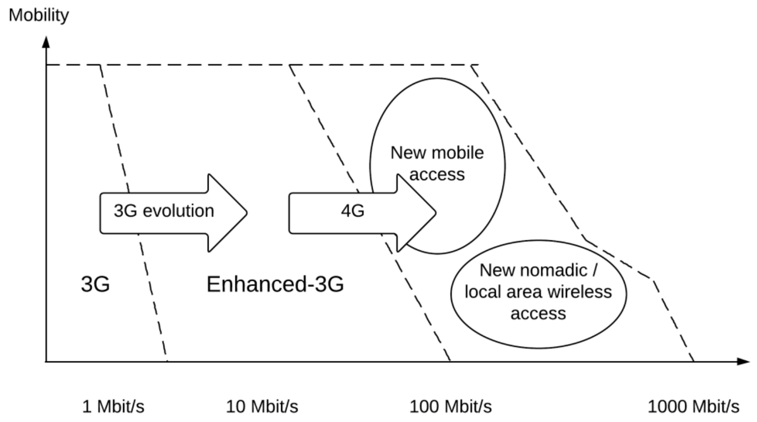


Figure 1 Capabilities of 3G and 4G communication systems [1].

There are four parameters that directly affect the design of LTE, which are advantageous compared to other cellular technologies [1].

- **Data rate**
  Services such as streaming and web browsing require a higher peak data rate. The peak data rate of 4G systems is getting close to 1Gbit/s which is much higher comparing with 384kbit/s for 2G systems and 56Mbit/s for 3G systems.
- **Delay**
  Interactive services which need a real-time operation, such as online gaming, require a lower delay in LTE systems.
- **Capacity**
  As the bandwidth and the number of base stations for mobile broadband are limited, the capacity of LTE systems is also important. The capacity is measured by the total data rate in the system averaged from each serving base station and per hertz available spectrum. If the capacity of a system is low, the Quality-of-Service (QoS) of users may be degraded. So a higher capacity is required in LTE systems.
- **Flexibility**
  As more spectrums are available for mobile broadband, communication systems are required to work efficiently on different frequency bands. The spectrum may be allocated in different sizes and even fragmented, thus requires a higher spectrum flexibility in LTE systems.

The goal of LTE is to provide a smooth transition from 3G systems to 4G systems. With LTE release 10 the goal is integrally achieved.

1.2 Introduction to downlink L1/L2 control signaling

Mobile transmission consists of two parts: transmission from base station to user equipment (downlink) and transmission from user equipment to base station (uplink). The uplink and downlink transmission can be identified by using a Frequency-Division Duplex (FDD) or Time-Division Duplex (TDD). Only FDD is considered in this thesis.

In order to support the transmission of uplink and downlink transport channels, downlink control signaling is required. This is also referred as downlink L1/L2 control signaling where L1 indicates the physical payer (Layer 1) and L2 indicates the Layer 2 MAC. Downlink L1/L2 control signaling consists of information for uplink and downlink transmission, such as downlink scheduling assignments, information for terminal to decode downlink signals and assign uplink transmission, and hybrid-ARQ acknowledgements in response to uplink transmission. Downlink control signaling can also contain information, such as power-control commands for uplink physical channels, if necessary [2].

The downlink signals are transmitted in terms of radio frames. There are 10 subframes within each radio frame and each subframe is mapped by modulated symbols in terms of OFDM symbols in time domain and subcarriers in the frequency domain. The OFDM symbols within a subframe are separated into control region and data region as illustrated in Figure 2.

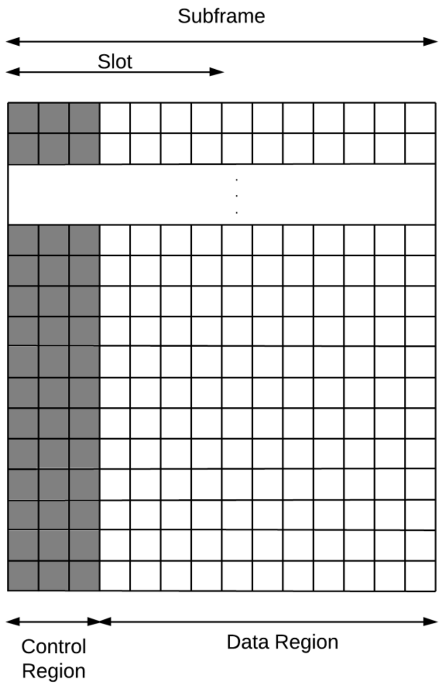


Figure 2 Control region and data region in a subframe [2].

As illustrated in Figure 2, the control region is always transmitted on the first several OFDM symbols to allow terminals to decode downlink scheduling assignments as early as possible. This allows the decoding of data region begins before the end of the subframe is processed. In order to simplify the design, the control region always occupies an integer number of OFDM symbols. The number of OFDM symbols used is decided by the cell bandwidth and the number of users scheduled in the subframe. For narrow cell bandwidths, two to four symbols are used to provide sufficient amount of control signaling. Otherwise, one to three symbols are occupied by control region. If the subframe is scheduled with lots of control information, the control region occupies more OFDM symbols. Otherwise control region is smaller and more OFDM symbols are used for data region. This allows the base station to adjust the amount of radio resource used by control region within each subframe dynamically and efficiently.

The downlink L1/L2 control signaling contains four types of physical-channels [2].

- **PCFICH**
  PCFICH stands for physical control format indicator channel which informs the terminal the number of OFDM symbols used for control region. Only one PCFICH is transmitted on each component carrier.

- **PDCCH**
  PDCCH stands for physical downlink control channel which is used to signal downlink scheduling assignments and uplink scheduling grants. Each PDCCH contains information for a single terminal or a group of terminals depending on the allocation and information carried. Multiple PDCCHs can exist in each component carrier if necessary.
- **PHICH**
  PHICH stands for physical hybrid-ARQ indicator channel which is used to carry hybrid-ARQ acknowledgements in response to uplink transmissions. Multiple PHICHs can exist in each component carrier if necessary.
- **R-PDCCH**
  R-PDCCH stands for relay physical downlink control channel which is used for relaying. This channel is not transmitted in the control region. As relaying is not considered in this thesis, this channel is ignored.

The data region is decoded immediately after the control region. This reduces the delay of downlink data decoding and further reduce the total downlink transmission delay. If a terminal is not scheduled after decoding the downlink L1/L2 control information, a power down operation may start until the next subframe in order to reduce terminal power consumption.

1.3 Introduction to this thesis

The purpose of this thesis work is to build encoder and decoder for LTE based downlink control signaling. The encoder is configured to use transmit diversity on two layers and support Downlink Control Information (DCI) type 1A, which is most commonly used in PDCCH. The decoder is designed to decode the encoded data and return user expected control information. The encoder is implemented in MATLAB and decoder is first implemented in MATLAB and then on Xilinx XUPV5-LX110T evaluation platform. The Xilinx XUPV5-LX110T evaluation platform uses a serial port to communicate with the computer. The encoded data can be transmitted through HyperTerminal to the FPGA and decoded data can be displayed on HyperTerminal. The decoder implemented on FPGA is an embedded system using a Xilinx Viterbi Decoder v7.0 IP as a hardware accelerated core while other functions are implemented in software.

In this thesis, detailed explanations of design parameters and mathematical models used for encoding and decoding the control region are described in Section 2. In Section 3, the implementation and verification of the encoder in MATLAB, implementation of the decoder in MATLAB and on Xilinx XUPV5-LX110T evaluation platform are described. In Section 4, decoding results, design utilities and speed of the decoder is discussed. In Section 5, the results presented in Section 4 are discussed in different respects and a final conclusion is drawn. Appendix A presents the method to set up Xilinx XUPV5-LX110T evaluation platform.

2 Theory

This section presents a detailed explanation of design parameters and mathematical models. Some definitions and commonly used mathematical symbols are listed and explained in Section 2.1. Some general steps of encoding and decoding processes are presented in Section 2.2. Section 2.3 presents detailed steps of encoding process and Section 2.4 presents detailed steps of the decoding process.

2.1 Definitions and Symbols

UE: The terminal is referred as UE which stands for user equipment.

Frame: For FDD LTE, the downlink transmission is processed by transmitting radio frames. The time of transmitting one radio frame (T_{rf}) is $10ms$.

Subframe: A subframe is a part of a downlink radio frame. There are 10 subframes within a radio frame, marked from 0 to 9. The time of transmitting one subframe (T_{sf}) is $1ms$.

Slot: There are 2 slots within a subframe which makes 20 slots within a radio frame. The time of transmitting one slot (T_{slot}) is $0.5ms$.

$\boldsymbol{n_s}$: Slot number within a radio frame, marked from 0 to 19.

Figure 3 illustrates the relation between frame, subframe and slot in FDD.

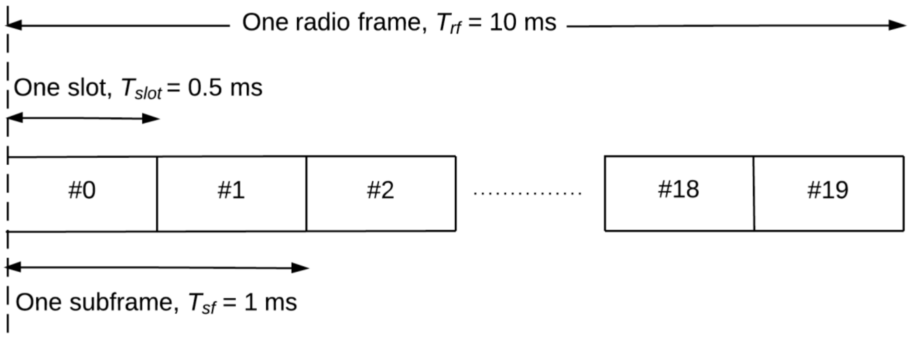

One radio frame, T_{rf} = 10 ms
One slot, T_{slot} = 0.5 ms
#0
#1
#2
#18
#19
One subframe, T_{sf} = 1 ms

Figure 3 Relation between frame, subframe and slot in FDD [3].

OFDM symbol: The transmission is divided into OFDM symbols in time domain. The number of OFDM symbols within in a subframe is 2 times of N_{symb}^{DL} which is introduced later.

Subcarrier: The transmitted signal is divided into subcarriers in the frequency domain.

$\boldsymbol{N_{sc}^{RB}}$: Number of subcarriers within a resource block, depending on the cyclic prefix type and downlink bandwidth as shown in Table 1.

$\boldsymbol{N_{symb}^{DL}}$: Number of OFDM symbols in a downlink slot, depending on the cyclic prefix type and downlink bandwidth as shown in Table 1. Only configuration with $\Delta f = 15kHz$ is considered in this thesis.

Table 1 $\mathrm{N_{sc}^{RB}}$ and $\mathrm{N_{symb}^{DL}}$ depending on cyclic prefix type and downlink bandwidth [4].

Configuration		N_{sc}^{RB}	N_{symb}^{DL}
Normal cyclic prefix	$\Delta f = 15kHz$	12	7
Extended cyclic prefix	$\Delta f = 15kHz$		6
	$\Delta f = 7.5kHz$	24	3

Resource element: A resource element (RE) is represented by one subcarrier on one OFDM symbol.

Resource grid: The transmitted signal in a slot is presented as resource grids which contains all resource elements within the slot.

Resource block: A resource grid can be separated into several resource blocks in the frequency domain.

$\boldsymbol{N_{RB}^{DL}}$: Number of resource blocks within a downlink subframe, depending on the downlink bandwidth.

$\boldsymbol{N_{RB}^{UL}}$: Number of resource blocks within a uplink subframe, depending on the uplink bandwidth.

The relation between resource grids, resource blocks and resource elements is illustrated in Figure 4.

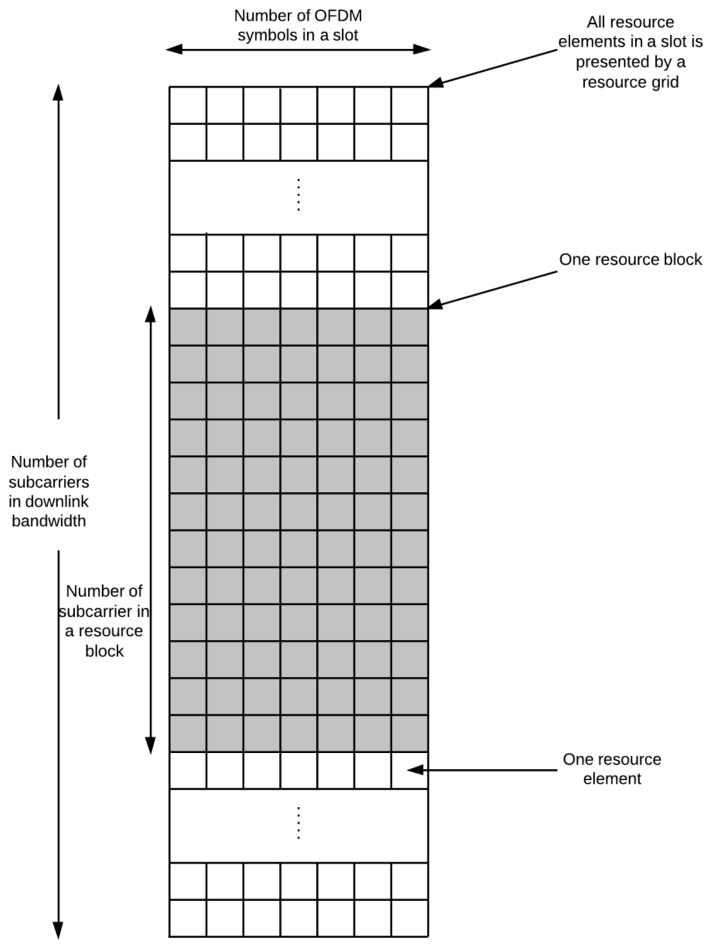

Number of OFDM symbols in a slot
All resource elements in a slot is presented by a resource grid
One resource block
Number of subcarriers in downlink bandwidth
Number of subcarrier in a resource block
One resource element

Figure 4 Relation between resource grids, resource blocks and resource elements [4].

$\boldsymbol{N_{ID}^{cell}}$: Physical layer cell identity. There are 504 unique physical layer cell identities grouped into 168 groups. Each group contains three unique identities. The value of $\mathrm{N_{ID}^{cell}}$ is calculated according to Equation 2.1.1

$$N_{ID}^{cell} = 3N_{ID}^{(1)} + N_{ID}^{(2)} \tag{2.1.1}$$

where $N_{ID}^{(1)}$ is group identity range from 0 to 167 and $N_{ID}^{(2)}$ is cell index within a group range from 0 to 2.

2.2 General Steps

Some commonly used steps for L1/L2 control signal encoding are described in this section including scrambling, modulation, layer mapping, precoding and resource element (RE) mapping. The overview of general steps in encoding process is illustrated in Figure 5.

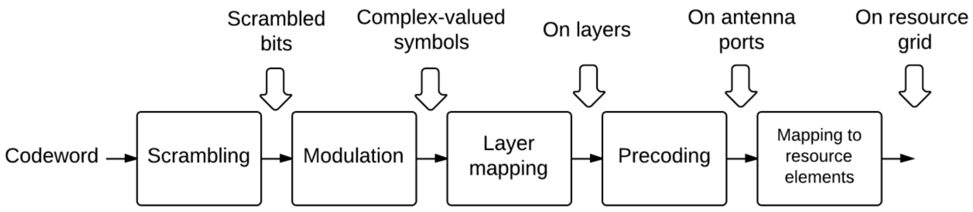


Figure 5 Overview of general steps for L1/L2 control signal encoding [5].

2.2.1 Scrambling

The scrambling process encodes every bit within every codeword depending on the scrambling sequence. The purpose of scrambling is to reject the inter-cell interference. Interference from other cells is descrambled as noise when the received data is descrambled with a known cell specific scrambling sequence at the receiver. The scrambling process is to use input bit sequence to carry out a bit-wise XOR operation with a cell specified pseudo-random sequence generated by length-31 Gold sequence generator. The input bit sequence is the block of bits within the codeword and the output bit sequence is the scrambled sequence with the same length as the input sequence. The scrambling process and methodical models are according to [6]. The overview of scrambling process is illustrated in Figure 6.

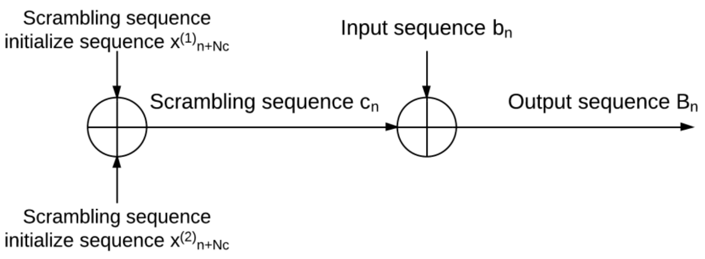

Scrambling sequence initialize sequence $x^{(1)}_{n+Nc}$
Input sequence b_n
Scrambling sequence c_n
Output sequence B_n
Scrambling sequence initialize sequence $x^{(2)}_{n+Nc}$

Figure 6 Overview of scrambling process.

The pseudo-random sequence c_n is a cell- and subframe-specific sequence which is generated according to Equation 2.2.1.

$$c_n = \left(x^{(1)}_{n+N_C} + x^{(2)}_{n+N_C}\right)\text{mod}2 \text{ for } n = 0, 1, \ldots, M_{PN} - 1 \quad (2.2.1)$$

where $N_C = 1600$ and M_{PN} is the length of input bit sequence.

Sequence x_n^1 and x_n^2 are generated according to Equation 2.2.2 and 2.2.3.

$$x_{n+31}^{(1)} = \left(x_{n+3}^{(1)} + x_n^{(1)}\right)\text{mod}2 \text{ for } n = 0, 1, \dots, M_{PN} + N_C - 32 \tag{2.2.2}$$

$$x_{n+31}^{(2)} = \left(x_{n+3}^{(2)} + x_{n+2}^{(2)} + x_{n+1}^{(2)} + x_n^{(2)}\right)\text{mod}2 \text{ for } n = 0, 1, \dots, M_{PN} + N_C - 32 \tag{2.2.3}$$

where x_n^1 and x_n^2 are initialized according to Equation 2.2.4 and 2.2.5.

$$x_0^{(1)} = 1, x_n^{(1)} = 0 \text{ for } n = 1, 2, \dots, 30 \tag{2.2.4}$$

$$\sum_1^{30} x_n^{(2)} \cdot 2^n = C_{init} = \begin{cases} \left(\left\lfloor \frac{n_s}{2} \right\rfloor + 1\right) \cdot \left(2 \cdot N_{ID}^{cell} + 1\right) \cdot 2^9 + N_{ID}^{cell} & \text{for PCFICH and PHICH} \\ \left\lfloor \frac{n_s}{2} \right\rfloor \times 2^9 + N_{ID}^{cell} & \text{for PDCCH} \end{cases} \tag{2.2.5}$$

The scrambled output sequence B_n is generated according to Equation 2.2.6.

$$B_n = (b_n + c_n)\text{mod}2 \text{ for } n = 0, 1, \dots, M_{PN} - 1 \tag{2.2.6}$$

where b_n is the input data sequence.

2.2.2 Modulation

The modulation process generates complex-valued modulation symbols depending on encoded bits. Two types of modulation methods are used in the downlink L1/L2 control signaling: binary phase shift keying (BPSK) and quadrature phase shift keying (QPSK). The modulation process described in this section is according to [7]. For QPSK, every two bits of input bit sequence (B_i and B_{i+1}) are modulated into a complex-valued modulation symbol $d_i = I + jQ$ where *I* and *Q* are generated as in Table 2.

Table 2 QPSK modulation lookup table.

B_i, B_{i+1}	I	Q
00	$1/\sqrt{2}$	$1/\sqrt{2}$
01	$1/\sqrt{2}$	$-1/\sqrt{2}$
10	$-1/\sqrt{2}$	$1/\sqrt{2}$
11	$-1/\sqrt{2}$	$-1/\sqrt{2}$

For BPSK, every bit of input bit sequence B_i is modulated into a complex-valued symbol $d_i = I + jQ$ where *I* and *Q* are generated as in Table 3.

Table 3 BPSK modulation lookup table.

B_i	I	Q
0	$1/\sqrt{2}$	$1/\sqrt{2}$
1	$-1/\sqrt{2}$	$-1/\sqrt{2}$

2.2.3 Layer mapping

The layer mapping process maps the complex-valued modulation symbols onto each layer used for transmission. Two layers are used to carry modulated transmit codewords with transmit diversity in this thesis. Transmit diversity is a communication technique that separate one source signal into two or more independent signals. The seperated signals can be demodulated with identical information to return the source signal. This technique benefits the overcome of fading effects. The use of transmit diversity is implied when using DCI format 1A [8], which is considered in this thesis. The two layers used are marked as Layer 0 and Layer 1. The layer mapping method in this section is according to Section 6.3.3.3 in [5].

The layer mapping process maps the complex-valued symbols to two layers according to Equation 2.2.7 and 2.2.8.

$$x_i^{(0)} = d_{2\cdot i} \text{ for } i = 0, 1, \dots, M_{symb}^{layer} - 1 \tag{2.2.7}$$

$$x_i^{(1)} = d_{2\cdot i+1} \text{ for } i = 0, 1, \dots, M_{symb}^{layer} - 1 \tag{2.2.8}$$

where d_i for $i = 0, \dots, 2 \cdot M_{symb}^{layer} - 1$ denotes the input block of modulated symbols, $x_i^{(0)}$ denotes the result block of symbols mapped on Layer 0 and $x_i^{(1)}$ denotes the result block of symbols mapped on Layer 1 according to Equation 2.2.9 and 2.2.10.

$$M_{symb}^{layer} = M_{symb}^{(0)} \div 2 \tag{2.2.9}$$

$$M_{symb}^{(0)} = \begin{cases} 16 \text{ for PCFICH} \\ 12 \text{ for PHICH} \end{cases} \tag{2.2.10}$$

2.2.4 Precoding

The precoding process encodes the complex-valued modulation symbols mapped by layer mapper for each antenna port. The precoding for transmit diversity is used in combination with layer mapping for transmit diversity. The input of precoding process is the block of symbols mapped on each layer by layer mapper. The number of antenna ports used for transmission is 2 or 4 where antenna port $p \in \{0,1\}$ and $p \in \{0,1,2,3\}$ respectively. In this thesis the number of antenna ports used is 2 which equals to the number of layers used by transmits diversity. The precoding method in this section is according to Section 6.3.4.3 in [5].

For a transmission with two antenna ports, the output block of symbols is generated according to Equation 2.2.11.

$$y_i = \left[y_i^{(0)} \;\; y_i^{(1)}) \right]^T \text{ for } i = 0, 1, \dots, M_{symb}^{ap} - 1 \tag{2.2.11}$$

where $M_{symb}^{ap} = 2 \cdot M_{symb}^{layer}$.

$y_i^{(0)}$ is the block of symbols mapped on antenna port 0 and $y_i^{(1)}$ is the block of symbols mapped on antenna port 1 which are generated according to Equation 2.2.12.

$$\begin{bmatrix} y_{2\cdot i}^{(0)} \\ y_{2\cdot i}^{(1)} \\ y_{2\cdot i+1}^{(0)} \\ y_{2\cdot i+1}^{(1)} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & j & 0 \\ 0 & -1 & 0 & j \\ 0 & 1 & 0 & j \\ 1 & 0 & -j & 0 \end{bmatrix} \begin{bmatrix} \mathrm{Re}(x_i^{(0)}) \\ \mathrm{Re}(x_i^{(1)}) \\ \mathrm{Im}(x_i^{(0)}) \\ \mathrm{Im}(x_i^{(1)}) \end{bmatrix} \tag{2.2.12}$$

2.2.5 RE mapping

The RE mapping process maps complex-valued modulation symbols for each antenna port to corresponding REs on resource grids. All REs are mapped in terms of resource element groups (REGs). Every four REs available for mapping L1/L2 control signals forms a REG. Each REG is mapped with a symbol quadruplet which consists of 4 complex-valued symbols. This mapping method is used to support transmit diversity. For transmission with two antenna ports, the relation between RE and REGs is illustrated in Figure 7.

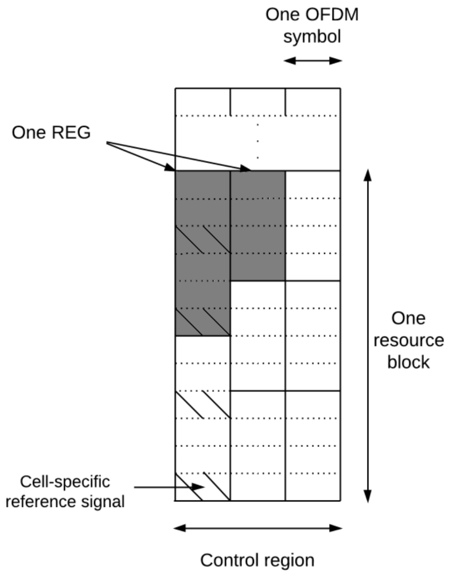


Figure 7 Illustration of REGs within one resource block with two antenna ports [9].

As illustrated in Figure 6, there is a RE reserved for cell-specific reference signal in every 3 REs on the first OFDM symbol. Therefore, there are two REGs within a resource block. The cell-specific reference signals are only mapped on the first and fifth OFDM symbol within a slot. The size of the control region is limited to 4 OFDM symbols. Therefore, for any other OFDM symbols used by control region, there are three REGs within in one resource block.

The quadruplets $z_i^{(p)}$ used for mapping on antenna port p are generated according to Equation 2.2.13.

$$z_i^{(p)} = \langle y_{4\cdot i}^{(p)}, y_{4\cdot i+1}^{(p)}, y_{4\cdot i+2}^{(p)}, y_{4\cdot i+3}^{(p)} \rangle \qquad (2.2.13)$$

where $i = 0,1,2,\ldots,\frac{N_{symbol}}{4}$, N_{symbol} is the number of complex-valued symbols and $p = 0,1$.

The position of each REG is represented by (k, l) where l is the index of OFDM symbol and k is the index of REG in frequency domain increased from 0.

2.2.6 Descrambling

The descrambling process is the reverse of scrambling process which returns the unscrambled bit sequence b_i from scrambled bit sequence B_i. The process is according to Equation 2.2.14.

$$b_i = (B_i + c_i)\text{mod}2 \text{ for } i = 0,\ldots,L-1 \qquad (2.2.14)$$

where L is the length of B_i and c_i is the scrambling sequence generated according to Section 2.2.1.

2.2.7 Demodulation

The BPSK demodulation generates a block of bits β_i for $i = 0,1,..N-1$ where N is the length of input block of symbols d_i. The BPSK demodulation process is according to Table 4.

Table 4 BPSK demodulation lookup table

d_i	β_i
$1+1i$	0
$-1-1i$	1
0	0

The QPSK demodulation generates a block of bits β_i for $i = 0,1,..2\cdot N-1$ where N is the length of input block of symbols d_i. The process has hard decision type and soft decision type. The hard decision process returns the bit sequence with value 0 or 1 and is used for PCIFCH demodulation. The soft decision process returns an integer sequence with value from 0 to 7 and is used for PDCCH demodulation as the Viterbi decoder requires a soft encoded input which described in Section 2.4.3.7. For the soft demodulated integer, value 0 indicates a strongest possibility of value 0 in original bit sequence while value 3 indicates the weakest possibility of value 0 in original bit sequence. Similarly, value 4 indicates a week 1 while 7 indicates a strong 1. The hard decision process is done according to Table 5.

Table 5 Hard decision QPSK demodulation lookup table.

$\mathrm{Re}(d_i)$	$\mathrm{Im}(d_i)$	b_i, b_{i+1}
≥ 0	≥ 0	00
≥ 0	< 0	01
< 0	≥ 0	10
< 0	< 0	11

The soft decision process is done according to Table 6.

Table 6 Soft decision QPSK demodulation lookup table.

$\mathrm{Re}(d_i)$	b_i
$\mathrm{Re}(d_i) \geq 0.9$	0
$0.9 > \mathrm{Re}(d_i) \geq 0.6$	1
$0.6 > \mathrm{Re}(d_i) \geq 0.3$	2
$0.3 > \mathrm{Re}(d_i) \geq 0$	3
$0 > \mathrm{Re}(d_i) \geq -0.3$	4
$-0.3 > \mathrm{Re}(d_i) \geq -0.6$	5
$-0.6 > \mathrm{Re}(d_i) \geq -0.9$	6
$-0.9 > \mathrm{Re}(d_i)$	7

$\mathrm{Im}(d_i)$	b_{i+1}
$\mathrm{Im}(d_i) \geq 0.9$	0
$0.9 > \mathrm{Im}(d_i) \geq 0.6$	1
$0.6 > \mathrm{Im}(d_i) \geq 0.3$	2
$0.3 > \mathrm{Im}(d_i) \geq 0$	3
$0 > \mathrm{Im}(d_i) \geq -0.3$	4
$-0.3 > \mathrm{Im}(d_i) \geq -0.6$	5
$-0.6 > \mathrm{Im}(d_i) \geq -0.9$	6
$-0.9 > \mathrm{Im}(d_i)$	7

2.2.8 Layer Demapping and Deprecoding

The layer demapping and deprecoding returns the block of symbols d_i from block of symbols after layer mapping and precoding $r_i^{(p)}$ where p is the antenna port index. The process is according to Equation 2.2.15 and 2.2.16.

$$d_{2i} = \mathrm{Re}(a_i^0) + \mathrm{Im}(b_i^0) \cdot j \text{ for } i = 0, \dots, \frac{N}{2} - 1 \quad (2.2.15)$$

$$d_{2i+1} = \mathrm{Re}(a_i^1) + \mathrm{Im}(b_i^1) \cdot j \text{ for } i = 0, \dots, \frac{N}{2} - 1 \quad (2.2.16)$$

where N is the length of input complex-valued symbols and $a_i^{(p)}$ and $b_i^{(p)}$ are generated according to Equation 2.2.17 to 2.2.20.

$$a_i^0 = r_{2i}^{(0)} + r_{2i+1}^{(1)} \text{ for } i = 0, \dots, \frac{N}{2} - 1 \quad (2.2.17)$$

$$a_i^1 = r_{2i}^{(1)} - r_{2i+1}^{(0)} \text{ for } i = 0, \dots, \frac{N}{2} - 1 \quad (2.2.18)$$

$$b_i^0 = r_{2i}^{(0)} - r_{2i+1}^{(1)} \text{ for } i = 0, \dots, \frac{N}{2} - 1 \quad (2.2.19)$$

$$b_i^1 = r_{2i}^{(1)} + r_{2i+1}^{(0)} \text{ for } i = 0, \dots, \frac{N}{2} - 1 \quad (2.2.20)$$

2.3 Encoding Algorithms

This section describes the encoding algorithms of each part of the L1/L2 control signaling. Detailed descriptions of PCFICH encoding, PHICH encoding and PDCCH encoding are presented in Section 2.3.1 to 2.3.3.

2.3.1 PCFICH Encoding

The PCFICH carries information about the number of OFDM symbols occupied by the control region, which is referred as control format indicator (CFI). The decoding of other channels relies on the value of CFI. The CFI needs to be decoded firstly and correctly so terminal can decide the size of control region and the starting OFDM symbol of data region [2]. To meet this requirement the PCFICH is always mapped onto the first OFDM symbol in a subframe. The number of OFDM symbols used for control channel is limited to 4 so the value of CFI is from 1 to 4. The PCFICH is transmitted only when the number of OFDM symbols for PDCCH is greater than zero. The range of OFDM symbols used by PDCCH is shown in Table 7.

Table 7 Range of CFI values used by PDCCH in the control region [10].

Subframe	Number of OFDM symbols for PDCCH ($N_{RB}^{DL} > 10$)	Number of OFDM symbols for PDCCH ($N_{RB}^{DL} \leq 10$)
Subframe 1 and 6 for TDD	1,2	2
MBSFN subframes on carrier supporting PDSCH, configured with 1 cell-specific antenna port	1	2
MBSFN subframes on carrier supporting PDSCH, configured with 2 or 4 cell-specific antenna ports	2	2
Subframes on a carrier not supporting PDSCH	0	0
Non-MBSFN subframes (except subframe 6 for TDD) configured with positioning reference signals	1,2,3	2,3
All other cases	1,2,3	2,3,4

MBSFN stands for multicast-broadcast single-frequency network, which is a communication channel used in LTE. So typically, the number of OFDM symbols used for PDCCH is 1 to 3 for N_{RB}^{DL} greater than 10 and 2 to 4 for otherwise. PDSCH stands for physical downlink shared channel which is used to carry transport blocks.

The overview of PCFICH encoding process is illustrated in Figure 8.

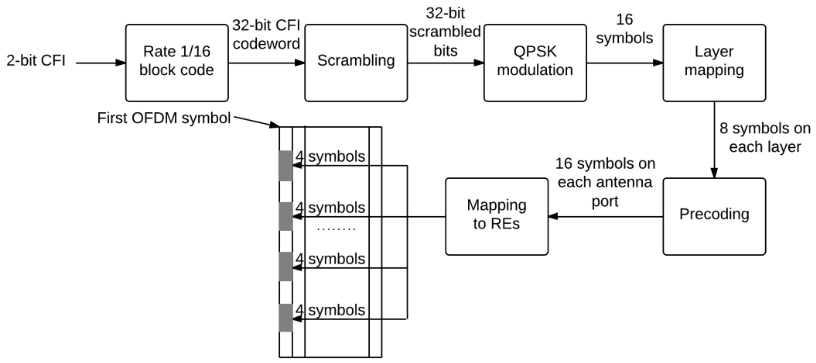


Figure 8 Overview of PCFICH encoding process [2].

2.3.1.1 CFI encoding

CFI is presented in 2 bits as the value is from 1 to 3 or 2 to 4 for narrow bandwidths. The CFI is encoded into a 32-bit codeword by channel coding with block encoding rate of 1/16 which means each input bit generates 16 output bits. The encoded codeword b_i is generated as in Table 8.

Table 8 CFI codeword generation depending on CFI value [11].

CFI	CFI codeword $< b_0, b_1, \dots, b_{31} >$
1	$< 0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1 >$
2	$< 1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0 >$
3	$< 1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1 >$
4 (reserved)	$< 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 >$

2.3.1.2 CFI codeword scrambling

The encoded CFI codeword b_i is then scrambled according to the scrambling method mentioned in Section 2.2.1. The bit sequence after scrambling is B_i where $i = 0,1,\dots,31$.

2.3.1.3 PCFICH modulation

The PCFICH modulation method is QPSK modulation as mentioned in Section 2.2.2. The input bit sequence is B_i and the output block of complex-valued symbols is d_i where $i = 0,1,\dots,15$.

2.3.1.4 PCFICH layer mapping

The block of complex-valued symbols d_i is mapped onto 2 layers for transmit diversity with the method mentioned in Section 2.2.3. The output block of complex-valued symbols after mapping is $x_i^{(p)}$ where p is the layer index and $i = 0,1,\dots,7$.

2.3.1.5 PCFICH precoding

The block of symbols $x_i^{(p)}$ is precoded for 2 antenna ports with the method mentioned in Section 2.2.4. The output block of complex-valued symbols after precoding is $y_i^{(p)}$ where p is the antenna port index and $i = 0,1,\dots,15$.

2.3.1.6 Mapping PCFICH to REs

The number of complex-valued symbols of PCFICH for each antenna port is 16, 4 symbol quadruplets are generated for RE mapping as mentioned in Section 2.2.5. The RE mapping method in this section is based on [7] with a different method of denoting the REG position. As PCFICH is mapped on the first OFDM symbol, $l_i^{PCFICH} = 1$ for REGs $(k_i^{PCFICH}, l_i^{PCFICH})$ used by PCFICH. The value of k_i^{PCFICH} for REGs are calculated according to Equation 2.3.1 to 2.3.4.

$$k_0^{PCFICH} = \gamma \tag{2.3.1}$$

$$k_1^{PCFICH} = \gamma + \lfloor N_{RB}^{DL}/2 \rfloor \cdot N_{sc}^{RB}/12 \quad (2.3.2)$$

$$k_2^{PCFICH} = \gamma + \lfloor 2 \cdot N_{RB}^{DL}/2 \rfloor \cdot N_{sc}^{RB}/12 \quad (2.3.3)$$

$$k_3^{PCFICH} = \gamma + \lfloor 3 \cdot N_{RB}^{DL}/2 \rfloor \cdot N_{sc}^{RB}/12 \quad (2.3.4)$$

where γ is calculated according to Equation 2.3.5.

$$\gamma = (N_{sc}^{RB}/12) \cdot (N_{ID}^{cell} \text{mod}(2 \cdot N_{RB}^{DL}) \quad (2.3.5)$$

The mapping of PCFICH is cell-specific as different N_{ID}^{cell} results in different mapping REGs.

2.3.2 PHICH Encoding

The PHICH channel is used to transmit hybrid-ARQ (HARQ) acknowledgements in response to uplink shared channel (UL-SCH) transmission from terminals. The acknowledgements are represented by 1-bit data where 0 indicates NACK and 1 indicates ACK. If an ACK is incorrectly decoded as NACK by the terminal, this leads to an unnecessary uplink retransmission of a correctly decoded transport block. If a NACK is incorrectly decoded as ACK by the terminal, this leads the loss of a transport block. So the error rate requirement of PHICH is stricter than PCFICH [9]. To meet such requirement, the PHICH is heavily encoded such as using BPSK instead of QPSK and multiple PHICHs are mapped into a PHICH group so each PHICH is separated into several REGs. Typically, the PHICH is mapped onto the first OFDM symbol so the terminal will always decode the PHICH even if the PCFICH decoding failed. The overview of PHICH encoding process is illustrated in Figure 9.

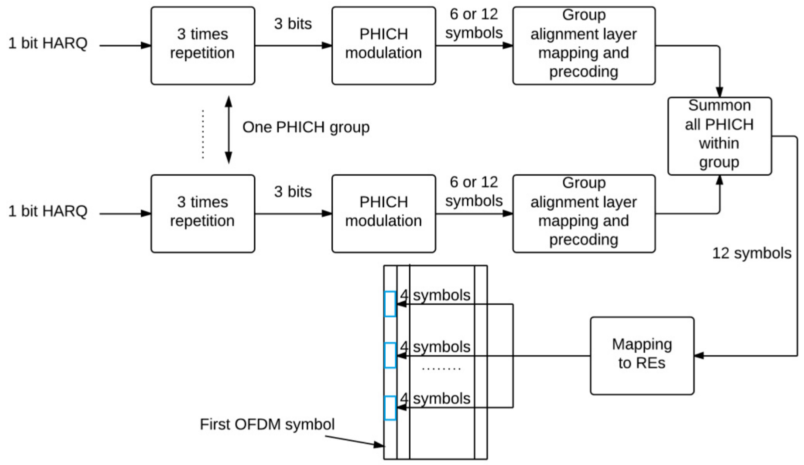


Figure 9 Overview of PHICH encoding process [9].

2.3.2.1 PHICH Groups

As mentioned before, multiple PHICHs are divided into PHICH groups to reduce the error rate. PHICHs within each PHICH group are separated with different orthogonal sequences. This section describes PHICH group and orthogonal index within a group according to [12]. Each PHICH in the control region is identified by $(n_{PHICH}^{group}, n_{PHICH}^{seq})$ where n_{PHICH}^{group} is the PHICH group number and n_{PHICH}^{seq} is the orthogonal sequence index within the group. The range of n_{PHICH}^{group} and n_{PHICH}^{seq} are according to Equation 2.3.6 and 2.3.7.

$$n_{PHICH}^{group} = 0, \dots, N_{PHICH}^{group} - 1 \tag{2.3.6}$$

$$n_{PHICH}^{seq} = \begin{cases} 0, 1, \dots, 7 & \text{for normal cyclic prefix} \\ 0, 1, 2, 3 & \text{for extended cyclic prefix} \end{cases} \tag{2.3.7}$$

where N_{PHICH}^{group} denotes the number of PHICH groups in the control region calculated according to Equation 2.3.8.

$$N_{PHICH}^{group} = \begin{cases} \lceil N_g \cdot (N_{RB}^{DL}/8) \rceil & \text{for normal cyclic prefix} \\ 2 \cdot \lceil N_g \cdot (N_{RB}^{DL}/8) \rceil & \text{for extended cyclic prefix} \end{cases} \tag{2.3.8}$$

where $N_g \in \left\{\frac{1}{6}, \frac{1}{2}, 1, 2\right\}$ and N_g is a scaling factor provided by higher layers to control the number of PHICH groups.

2.3.2.2 HARQ Acknowledgement Encoding

The 1-bit HARQ acknowledgement is encoded by repeated three times to create a 3-bit HARQ bit sequence. The output block of bits b_0, b_1, b_2 are generated according to Table 9.

Table 9 HARQ indicator codeword mapping [13].

HARQ acknowledgement	b_0, b_1, b_2
0	0,0,0
1	1,1,1

2.3.2.3 PHICH modulation

The PHICH modulation process contains BPSK modulation, scrambling and orthogonal sequence multiplication according to Section 6.9.1 in [12] resulting in a block of complex-valued symbols d_i. The HARQ bit sequence is first BPSK modulated as mentioned in Section 2.2.2. The input bit sequence is b_i and the output block of complex-valued symbols is z_i where $i = 0,1,2$.

Block of complex-valued symbols z_i is then modulated according to Equation 2.3.9.

$$d_i = w_{i \bmod N_{SF}^{PHICH}} \cdot (1 - 2 \cdot c_i) \cdot z_{\lfloor i/N_{SF}^{PHICH} \rfloor} \text{ for } i = 0, \dots, M_{symb} - 1 \tag{2.3.9}$$

where c_i is the scrambling sequence generated as mentioned in Section 2.2.1, w_i is the orthogonal sequence generated as in Table 10 to separated PHICHs within the PHICH group, and M_{symb} is calculated according to Equation 2.3.10.

$$M_{symb} = N_{SF}^{PHICH} \cdot M_s \tag{2.3.10}$$

with $M_s = 3$ and N_{SF}^{PHICH} derived from Equation 2.3.11.

$$N_{SF}^{PHICH} = \begin{cases} 4 \text{ for normal cyclic prefix} \\ 2 \text{ for extended cyclic prefix} \end{cases} \tag{2.3.11}$$

Table 10 orthogonal sequence for PHICH [12].

n_{PHICH}^{seq}	$w_{n_{PHICH}^{seq}}$ for normal cyclic prefix	$w_{n_{PHICH}^{seq}}$ for extended cyclic prefix
0	$[+1, +1, +1, +1]$	$[+1, +1]$
1	$[+1, -1, +1, -1]$	$[+1, -1]$
2	$[+1, +1, -1, -1]$	$[+j, +j]$
3	$[+1, -1, -1, +1]$	$[+j, -j]$
4	$[+j, +j, +j, +j]$	-
5	$[+j, -j, +j, -j]$	-
6	$[+j, +j, -j, -j]$	-
7	$[+j, -j, -j, +j]$	-

2.3.2.4 PHICH Resource Group Alignment

The block of symbols after modulation is aligned with resource element group size. This is necessary because the group size is different for normal cyclic prefixed subframe and extended cyclic prefixed subframe. The group alignment method is according to Section 6.9.2 in [12]. For normal cyclic prefix, block of symbols d_i is aligned according to Equation 2.3.12.

$$d_i^{(0)} = d_i \text{ for } i = 0, \dots, M_{symb} - 1 \tag{2.3.12}$$

For extended cyclic prefix, block of symbols d_i is aligned according to Equation 2.3.13.

$$\left[d_{4\cdot i}^{(0)}, d_{4\cdot i+1}^{(0)}, d_{4\cdot i+2}^{(0)}, d_{4\cdot i+3}^{(0)}\right]^T = \begin{cases} [d_{2\cdot i}, d_{2\cdot i+1}, 0, 0]^T \; n_{PHICH}^{group} \bmod 2 = 0 \\ [0, 0, d_{2\cdot i}, d_{2\cdot i+1}]^T \; n_{PHICH}^{group} \bmod 2 = 1 \end{cases} \tag{2.3.13}$$

where $i = 0, \dots, M_{symb}/2 - 1$. The length of output block of symbols $d_i^{(0)}$ is 12 for both normal cyclic prefix and extended cyclic prefix. The number of PHICH groups for extended cyclic prefix is reduced by half after resource group alignment.

2.3.2.5 PHICH Layer Mapping

The block of complex-valued symbols $d_i^{(0)}$ is mapped onto 2 layers for transmit diversity with the method mentioned in Section 2.2.3. The output block of complex-valued symbols after mapping is $x_i^{(p)}$ where p is the layer index and $i = 0,1,\dots,5$.

2.3.2.6 PHICH precoding

The block of symbols $x_i^{(p)}$ is precoded for 2 antenna ports with the method mentioned in Section 2.2.4. The output block of complex-valued symbols after precoding is $y_i^{(p)}$ where p is the antenna port index and $i = 0,1,\dots,11$.

2.3.2.7 Mapping PHICH to REs

The PHICHs are mapped to REs in terms of PHICH groups. The RE mapping method described in this section is according to Section 6.9.3 in [12]. The block of symbols of PHICH group $a_n^{(p)}$ is generated by summing all PHICHs within the group according to Equation 2.3.14.

$$a_n^{(p)} = \sum y_i^{(p)}(n) \text{ for } n = 0,1,\dots,11 \tag{2.3.14}$$

where *i* the index of used PHICH within the group.

Block of symbols $a_n^{(p)}$ is then mapped into PHICH mapping units $\varphi_{m'}^{(p)}(n)$. Considering only the FDD and $\varphi_{m'}^{(p)}(n)$ is generated according to Equation 2.3.15.

$$\varphi_{m'}^{(p)}(n) = \begin{cases} \alpha_m^{(p)}(n) & \text{with } m' = m = 0,1,\dots,N_{PHICH}^{group} - 1 \\ \alpha_m^{(p)}(n) + \alpha_{m+1}^{(p)}(n) & \text{with } m' = \frac{m}{2}, m = 0,2,\dots,N_{PHICH}^{group} - 2 \end{cases} \tag{2.3.15}$$

Block of symbols $\varphi_{m'}^{(p)}(n)$ is divided into 3 symbol quadruplets for each PHICH group with the method mentioned in Section 2.2.5. The mapping of generated symbol quadruplets is affected by the number of OFDM symbols used by PHICH, which is referred as PHICH duration. The PHICH duration in different subframe type and cyclic prefix type is specified in Table 11.

Table 11 PHICH duration in different type of subframes [12].

PHICH duration	Non-MBSFN subframes		MBSFN subframes on a carrier supporting PDSCH
	Subframes 1 and 6 for TDD	All other cases	
Normal cyclic prefix	1	1	1
Extended cyclic prefix	2	3	2

The position of REGs used by PHICH (k_i^{PHICH}, l_i^{PHICH}) is calculated according to Equation 2.3.16.

$$l_i^{PHICH}\begin{cases}0 & \text{normal PHICH duration}\\ \left(\left\lfloor\frac{m'}{2}\right\rfloor+i+1\right)\text{mod}2 & \text{for extended PHICH duration in MBSFN subframes}\\ \left(\left\lfloor\frac{m'}{2}\right\rfloor+i+1\right)\text{mod}2 & \text{for extended PHICH duration, in 1,6 subframe of TDD}\\ i & \text{otherwise}\end{cases} \quad (2.3.16)$$

For extended PHICH duration in MBSFN subframes or extended PHICH duration in 1 and 6 subframes ofr TDD, k_i^{PHICH} is calculated according to Equation 2.3.17.

$$k_i^{PHICH}=\begin{cases}\left(\left\lfloor N_{ID}^{cell}\cdot\frac{n_{l_i}}{n_1}\right\rfloor+m'\right)\text{mod}n_{l_i} & i=0\\ \left(\left\lfloor N_{ID}^{cell}\cdot\frac{n_{l_i}}{n_1}\right\rfloor+m'+\left\lfloor n_{l_i}/3\right\rfloor\right)\text{mod}n_{l_i} & i=1\\ \left(\left\lfloor N_{ID}^{cell}\cdot\frac{n_{l_i}}{n_1}\right\rfloor+m'+\left\lfloor 2n_{l_i}/3\right\rfloor\right)\text{mod}n_{l_i} & i=2\end{cases} \quad (2.3.17)$$

where n_{l_i} is the number of REGs that has not been assigned by PCIFCH in OFDM symbol l_i^{PHICH}, and n_1 is the number of REGs that has not been assigned by PCIFCH on the second OFDM symbol.

For any other case k_i^{PHICH} is calculated according to Equation 2.3.18.

$$k_i^{PHICH}=\begin{cases}\left(\left\lfloor N_{ID}^{cell}\cdot\frac{n_{l_i}}{n_0}\right\rfloor+m'\right)\text{mod}n_{l_i} & i=0\\ \left(\left\lfloor N_{ID}^{cell}\cdot\frac{n_{l_i}}{n_0}\right\rfloor+m'+\left\lfloor n_{l_i}/3\right\rfloor\right)\text{mod}n_{l_i} & i=1\\ \left(\left\lfloor N_{ID}^{cell}\cdot\frac{n_{l_i}}{n_0}\right\rfloor+m'+\left\lfloor 2n_{l_i}/3\right\rfloor\right)\text{mod}n_{l_i} & i=2\end{cases} \quad (2.3.18)$$

where n_0 is the number of REGs that has not been assigned by PCIFCH on the first OFDM symbol.

As k_i^{PHICH} calculated for PHICH is ignoring REGs already assigned by PCFICH, k_i^{PHICH} of PHICHs mapped on the first OFDM symbol may not indicate the correct REG index on the resource grid. To calculate the correct position, the value k_i^{PHICH} of every REG used for PHICH on the first OFDM symbol needs to be compared with k_i^{PCFICH}. The value of k_i^{PHICH} is increased by an offset which equals to the number of k_i^{PCFICH}s that the comparing k_i^{PHICH} is larger or equal to. If the increased k_i^{PHICH} equals to any k_i^{PCFICH}, the value is further increased by 1. After the process, k_i^{PHICH} indictes the correct REG index on the resource grid.

2.3.3 PDCCH Encoding

The PDCCH is used for carrying DCI, which informs the terminal about downlink scheduling assignments, uplink scheduling grants and power-control commands [14].

The downlink scheduling assignments mainly include PDSCH resource information, transport format, hybrid-ARQ information and other information that is discussed in detail in Section 2.3.3.1.

Uplink scheduling grants include physical uplink shared channel (PUSCH) resource indication, transport format, hybrid-ARQ information and part of the power control commands for PUSCH.

The power-control commands are used to complete the power control information provided partly in downlink scheduling assignments and uplink scheduling grants.

The DCI is designed in different formats to provide different usage. The DCI format used to carry different DCI information is listed in Table 12.

Table 12 DCI formats and usage [14].

	Usage				
Size	Uplink grant		Downlink assignment		Power control
Small	DCI formats	Specifications	DCI formats	Specifications	DCI formats
	-	-	1C	Special purpose compact assignment	-
	0	Single layer	1A	Contiguous allocations only	3,3A
.	-	-	1B	Codebook-based beam-forming using CRS	-
.	-	-	1D	Multi-user MIMO suing CRS	-
.	4	Spatial multiplexing	-	-	-
	-	-	1	Flexible allocations	-
	-	-	2A	Open-loop spatial multiplexing using CRS	-
	-	-	2B	Dual-layer transmission using CRS	-
	-	-	2C	Multi-layer transmission using CRS	-
Large	-	-	2	Closed-loop spatial multiplexing using CRS	-

In this thesis, DCI format 1A used for downlink scheduling assignment is considered. The overview of PDCCH encoding process is illustrated in Figure 10. Every step of the encoding is described in detail through Section 2.3.3.1to 2.3.3.10.

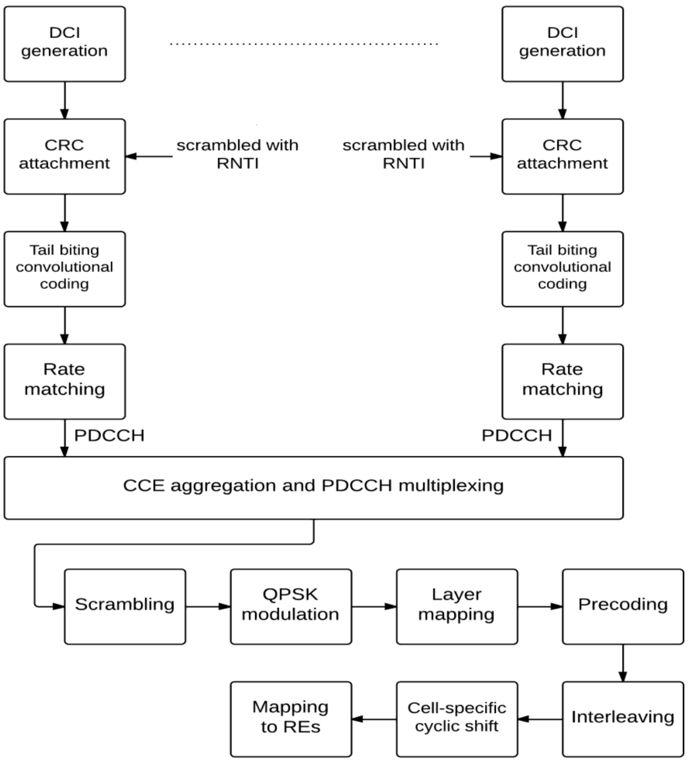


Figure 10 Overview of PDCCH encoding process [15].

2.3.3.1 DCI generation

Information contained in DCI used for downlink scheduling assignment is described according to [14] as follows:

Carrier indicator

The carrier indicator is presented only when cross-carrier scheduling is enabled. Cross-carrier scheduling means a terminal receives or transmits on multiple component carriers [15]. This technique is enabled by using radio resource control (RRC) signaling. The carrier indicator indicates the component carrier that the transmitted DCI is related to.

Resource-block allocation

The resource-block allocation informs the terminal about the resource blocks used to transmit PDSCH in the carrier. There three types of downlink resource block allocation methods, type 0, 1 and 2. The frequency-contiguous resource block allocation is referred as type 2. The structure of different resource-block allocation types is illustrated in Figure 11.

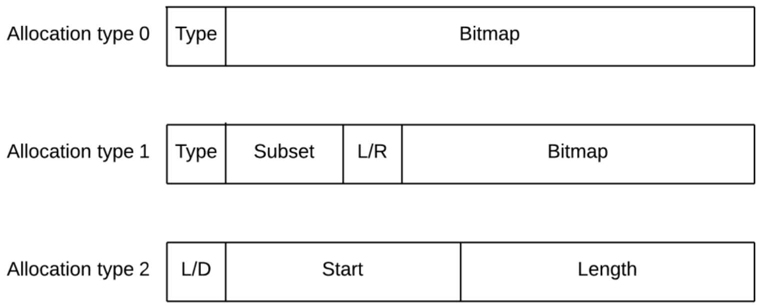


Figure 11 Structure of different allocation types [14].

Allocation type 0 is non-contiguous allocation method. The Type part is used to indicate the type used 0 or 1. The second part is a bitmap with the size same as N_{RB}^{DL}. All allocated resource blocks are indicated by the bitmap. This method creates a large DCI if downlink bandwidth is large but offers high allocation flexibility.

Allocation type 1 is also non-contiguous allocation method. In this case, the downlink resource blocks are divided into several subsets. The Type part is used the same as in type 0. The subsets used for transmitting PDSCH are indicated by the Subset part. The bitmap indicates the resource blocks used in a subset. The L/R part indicates the bitmap start from LSB or MSB in the subset respectively. This method offers a reduced size DCI with lower allocation flexibility comparing with type 0.

Allocation type 2 is contiguous allocation method which means arbitrary allocations are not supported and only frequency-contiguous allocation is allowed. In this case, the virtual resource blocks (VRBs) are used to describe physical resource blocks (PRBs) used for transmission according to Section 6.3.2.1 and 6.3.2.2 in [4].Allocation type 2 is the only allocation type supported in DCI format 1A. Instead of containing a bitmap, allocation type 2 contains the starting position of used resource block and length of the resource block allocation. The L/D part is used to indicate whether the allocation is localized or distributed. Localized allocation means both VRBs and PRBs are allocated contiguously while distributed allocation means VRBs are allocated contiguously and PRBs are allocated with gaps inserted. Allocation type 2 supports both localized and distributed allocation while allocation type 0 and type 1 only supports localized allocation. For localized allocation, the number of VRBs $N_{VRB}^{DL} = N_{RB}^{DL}$ and the index of VRB n_{VRB}^{DL} equals to the index of physical resource block n_{RB}^{DL}. For distributed allocation, N_{VRB}^{DL} is calculated according to Equation 2.3.19.

$$N_{VRB}^{DL} = \begin{cases} 2 \cdot \min\left(N_{gap}, N_{RB}^{DL} - N_{gap}\right) & \text{for } N_{gap} = N_{gap,1} \\ \left\lfloor N_{RB}^{DL} / 2N_{gap} \right\rfloor \cdot N_{gap} & \text{for } N_{gap} = N_{gap,2} \end{cases} \tag{2.3.19}$$

where the value of $N_{gap,1}$ and $N_{gap,2}$ is decided by N_{RB}^{DL} according to Table 13.

Table 13 VRB gap values [4].

N_{RB}^{DL}	$N_{gap,1}$	$N_{gap,2}$
6-10	$\lceil N_{RB}^{DL}/2 \rceil$	-
11	4	-
12-19	8	-
20-26	12	-
27-44	18	-
45-49	27	-
50-63	27	9
64-79	32	16
80-110	48	16

Whether N_{gap} equals to $N_{gap,1}$ or $N_{gap,2}$ is signaled in the DCI.

Hybrid-ARQ process indicator

The hybrid-ARQ process indicator is used to inform the terminal about the hybrid-ARQ process to use for soft combining. This indicator is presented by 3 bits for FDD and 4 bits for TDD.

For the first (or only) transport block

1. Modulation and coding scheme, which informs the terminal about information of modulation scheme, the code rate and the transport block size. 5 bits are used for this part, creating 32 combinations (I_{MCS}).
2. New-data indicator, which informs the terminal of a new transmission. The terminal clears the soft buffer to initial the transmission.
3. Redundancy version, which informs the terminal about the redundancy version of the transmission.

For the second transport block

This part is similar to the previous part. As DCI format 1A does not support the second transport block, this part is not described further in detail.

Multi-antenna information

As different DCI formats may be used with different multi-antenna schemes, the multi-antenna information is encoded in DCI. In DCI format 1A, the transmit diversity is implied, so this part is not described further in detail.

Downlink assignment index

The downlink assignment index informs the terminal about the number of downlink transmission to be generated in response of a single hybrid-ARQ acknowledgement. This part presents in TDD only which is not considered in this thesis.

Transmit-power control

The transmit-power control (TPC) contains the information for power commands of physical uplink control channel (PUCCH). This part is also used as acknowledgement resource indicator (ARI) in some cases.

DCI format 0/1A indicator

This indicator indicates whether the DCI format used is 0 or 1A as they have the same DCI size. Only DCI format 0 and 1A contain this part. For DCI format 3 and 3A, which have the same size as DCI format 0 and 1A, can be identified from the radio-network temporary identifier (RNTI) used.

Padding part

Padding part is added when the size of DCI format 0 and 1A are not the same. After padding DCI format 0 and 1A have the same size to ensure the DCI payload size is the same for uplink and downlink. Padding is also used to ensure DCIs (expect format 0, 1A, 3 and 3A) have different sizes. If the encoded DCI have an ambiguous size then padding is necessary to avoid ambiguous decoding. The ambiguous DCI sizes L_{Am} are according to Equation 2.3.20 [16].

$$L_{Am} = 12, 14, 16, 20, 24, 26, 32, 40, 44, 56 \tag{2.3.20}$$

The DCI format 1A used in this thesis only supports compact downlink scheduling assignment. The format only supports frequency-contiguous resource block allocation and can be used in all transmission modes. For each DCI (including zero-padding bits, if any), the first field is mapped start from the lowest order DCI bit, and the following successive field is mapped to higher order DCI bits. For each information field, the MSB is mapped to the lowest order DCI bit in the field. The mapping procedure of a DCI message with length L and N fields is illustrated in Figure 12.

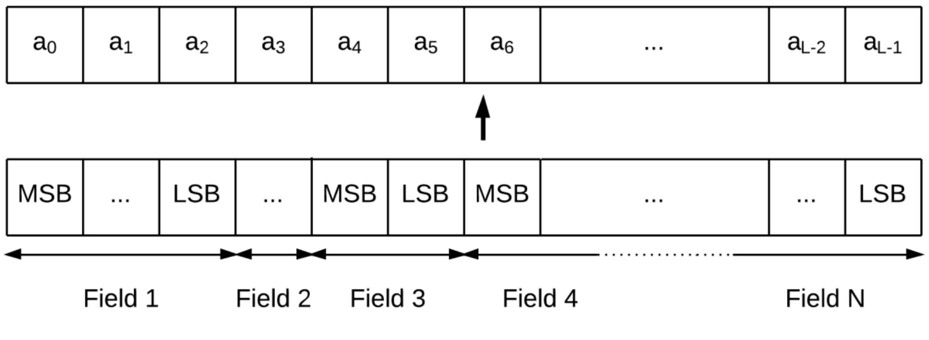


Figure 12 Illustration of DCI message mapping procedure.

The following steps describe the process of generating information fields for DCI format 1A used in this thesis according to [16].

- **Field 1: Carrier indicator: 0 or 3 bits.**
  The carrier indicator value ranges from 0 to 7.
- **Field 2: Format 0/1A differentiation flag: 1 bit.**
  DCI format 1A is identified by value 1 and format 0 by value 0.

If DCI format 1A is used for random access procedure for a UE initiated by a PDCCH order and the DCI is cell RNTI (C-RNTI) scrambled, the remaining fields are described as follows. The random access procedure is used when a UE is trying to connect to the network for the first time.

- **Field 3: Localized/Distributed flag for VRB assignment: 1 bit.**
  Set to 0.
- **Field 4: Resource block assignment: $\left\lceil \boldsymbol{log_2(\frac{N_{RB}^{DL}(N_{RB}^{DL}+1)}{2})} \right\rceil$ bits.**
  All bits are set to 1.
- **Field 5: Preamble index: 6 bits.**
  The preamble index is generated for the random access procedure, range: 0 to 63.
- **Field 6: Physical random access channel (PRACH) Mask Index: 4 bits.**
  The PRACH mask index indicates the PRACH to use, range: 0 to 15.
- **Field 7: Remaining bits.**
  All remaining bits added to make the length equal to the DCI used for compact scheduling assignment of one PDSCH code with format 1A are set to zero.

If DCI format 1A is used for the scheduling of PDSCH codeword in one cell, the remaining fields are described as follows:

- **Field 3: Localized/Distributed flag for VRB assignment: 1 bit.**
  Localized VRB assignment is identified by value 0 and distributed VRB by value 1.
- **Field 4: Resource block assignment: $\left\lceil \boldsymbol{log_2(\frac{N_{RB}^{DL}(N_{RB}^{DL}+1)}{2})} \right\rceil$ bits.**
  For localized VRB, the value is provided by resource allocation value RBA which is calculated according to Equation 2.3.21 [17].

$$RBA = \begin{cases} N_{RB}^{DL} \times (RB_L - 1) + RB_S & \text{for } (RB_L - 1) \leq \lfloor N_{RB}^{DL}/2 \rfloor \\ N_{RB}^{DL} \times (N_{RB}^{DL} - RB_L + 1) + (N_{RB}^{DL} - 1 - RB_S) & \text{for } (RB_L - 1) > \lfloor N_{RB}^{DL}/2 \rfloor \end{cases} \tag{2.3.21}$$

  where RB_L is the length of resource block assignment and RB_S is the starting position of resource block assignment. For distributed VRB, if Ndlrb is smaller than 50 or if the DCI is random access RNTI (RA-RNTI), paging RNTI (P-RNTI) or system information (SI-RNTI) scrambled, the value is provided by resource allocation value r. Otherwise the MSB in the field indicates N_{gap}.

- **Field 5: Modulation and coding scheme: 5 bits.**
  This field presents I_{mcs} in 5 bits.

- **Field 6: HARQ process number: 3 bits.**
  This field indicates the process number of the HARQ range from 0 to 5 for FDD. If the DCI with format 1A is scrambled with RA-RNTI, P-RNTI or SI-RNTI, the HARQ process numbers are reserved.

- **Field 7: New data indicator: 1 bit.**
  For DCI format 1A that is RA-RNIT, P-RNTI or SI-RNTI scrambled: if N_{RB}^{DL} is larger of equal to 50 and VRB assignment is distributed, the new data indicator indicates N_{gap}, where $N_{gap} = N_{gap,1}$ is indicated by value 0 and $N_{gap} = N_{gap,2}$ is indicated by value 1; otherwise the new data indictor is reserved. If the DCI format 1A is not scrambled with RA-RNIT, P-RNTI or SI-RNTI then the new data indicator indicates whether the transmission is new with value 1 indicating a new transmission and value 0 for otherwise.

- **Field 8: Redundancy version: 2 bits.**
  This field indicates the value of redundancy version.

- **Filed 9: TPC command for PUCCH: 2 bits.**
  If the DCI format 1A is RA-RNTI, P-RANTI or SI-RNTI scrambled, the MSB of TPC command is reserved; the LSB of TPC command indicates the column index N_{PRB}^{1A} of transport block size (TBS) table, where value 0 indicates $N_{PRB}^{1A} = 2$ and value 1 indicates $N_{PRB}^{1A} = 3$. Else then the two bits indicates the TPC command.

- **Field 10: Sounding reference symbol (SRS) request: 0 or 1 bit.**

  If this field presents only when DCI format 1A is used for scheduling PDSCH which are mapped onto the UE-specific search space given by the C-RNTI. When presented, value 1 indicates SRS trigger type 1 and value 0 for SRS trigger type 0.

- **Field 11: HARQ-ACK resource offset: 0 or 2 bits**

  This field presents only when the DCI 1A is used in enhanced PDDCH (EPDCCH) transmission. As only PDCCH transmission is considered in this thesis this field is not presented.

The DCI message is generated by multiplexing all fields with the method mentioned before. Padding is used, if necessary, to modify the length of the DCI according to [17] as follows:

- Zeros are added to the MSB of generated DCI until the lengths of DCI format 1A and 0 are equal if the UE is not configured to decode the PDCCH with C-RNTI and size of DCI format 1A is smaller.

- Zeros are added to the MSB of generated DCI until the lengths of DCI format 1A and 0 are equal if the UE is configured to decode the PDCCH with C-RNTI and size of DCI format 1A is smaller than the size of DCI format 0 on the same search space for the same serving cell.

- If the length of DCI has one of the ambiguous DCI sizes one zero bit shall be added to MSB.

As DCI format 0 is not generated in this thesis, only the length L_{DCI0} is calculated according to Equation 2.3.22 [18].

$$L_{DCI0} = \begin{cases} 18 + \left\lceil log_2(\frac{N_{RB}^{UL}(N_{RB}^{UL}+1)}{2}) \right\rceil \text{without carrier indicator} \\ 21 + \left\lceil log_2(\frac{N_{RB}^{UL}(N_{RB}^{UL}+1)}{2}) \right\rceil \quad \text{with carrier indicator} \end{cases} \tag{2.3.22}$$

After the process the DCI bit sequence is generated, represented as a_k for $k = 0, 1, 2, \ldots, A - 1$ where A is the length of DCI. The generated DCI format 1A is illustrated in Figure 13.

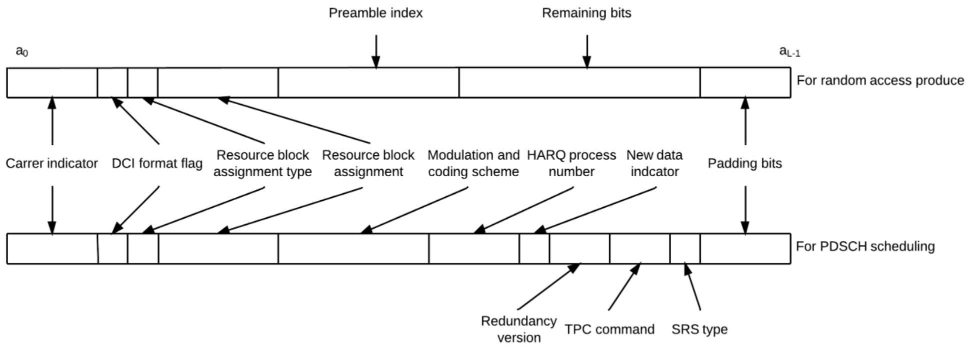


Figure 13 Illustration of generated DCI fields.

2.3.3.2 CRC attachment

The CRC which stands for cyclic redundancy check is a code commonly used for error detection at the terminal. The CRC attachment method described in this section is according to [19]. The payload to attach CRC is denoted by a_k for $k = 0, 1, 2, \ldots, A - 1$ where A is the length of DCI. The CRC length is fixed to 16 and the cyclic polynomial $g_{CRC16}(D)$ is generated according to Equation 2.3.23.

$$g_{CRC16}(D) = [D^{16} + D^{12} + D^5 + 1] \tag{2.3.23}$$

The parity bits calculation process is done as follows:

- Add 16 bits of zeros to the a_k for $k = A, A+1, \dots, A+15$.
- Select 16 bits from a_k, denoted by a_i to a_{i+15} with $i = 0$. Generate a block of bits e_k for $k = 0,1, \dots 15$ which is the same as a_i to a_{i+15}. If e_0 equals to 1 then proceed an XOR operation with the polynomial bits. Otherwise select the next 16 bits from a_k by increasing i by 1 until $i = A - 1$.
- After i has been increased to $A - 1$, the parity bits p_i for $i = 0,1, \dots ,15$ are presented by e_k for $k = 0,1, \dots 15$.

The bit sequence after CRC attachment b_k is generated according to Equation 2.3.24.

$$b_k = \begin{cases} a_k & \text{for } k = 0, 1, 2, \dots, A-1 \\ p_{k-A} & \text{for } k = A, A+1, A+2, \dots, A+15 \end{cases} \tag{2.3.24}$$

The bit sequence b_k is then scrambled with RNTI used for the transmission, generating bit sequence c_k according to Equation 2.3.25.

$$c_k = \begin{cases} \begin{cases} b_k & \text{for } k = 0, 1, ,2, \dots, A-1 \\ (b_k + x_{rnti,k-A}) \bmod 2 & \text{for } k = A, A+1, \dots, A+15 \end{cases} & \text{for case 1} \\ \begin{cases} b_k & \text{for } k = 0, 1, ,2, \dots, A-1 \\ (b_k + x_{rnti,k-A} + x_{AS,k-A}) \bmod 2 & \text{for } k = A, A+1, \dots, A+15 \end{cases} & \text{for case 2} \end{cases} \tag{2.3.25}$$

where case 1 stands for UE transmit antenna selection is not configured or applicable and case 2 stands for UE transmit antenna selection is configure and applicable and DCI format is 0. Bit sequence x_{rnti} is the corresponding RNTI with $x_{rnti,0}$ indicating the MSB of the RNTI and bit sequence x_{AS} is the antenna selection mask generated according to Table 14.

Table 14 UE transmit antenna selection mask [19].

UE transmit antenna selection	$< x_{AS,0}, x_{AS,1}, \dots, x_{AS,15} >$
UE port 0	$< 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 >$
UE port 1	$< 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1 >$

2.3.3.3 Channel coding

The channel coding process adds redundant information bits to the sequence to form forward error correction and improves the channel capacity. There are two types of channel coding methods: tail biting convolutional coding and turbo coding. For PDCCH encoding, the tail biting convolution coding is used. The channel coding method described in this section is according to [20]. The tail biting convolutional coding constraint length is set to 7 and coding rate is set to 1/3. The structure of a tail biting convolutional coder is illustrated in Figure 14.

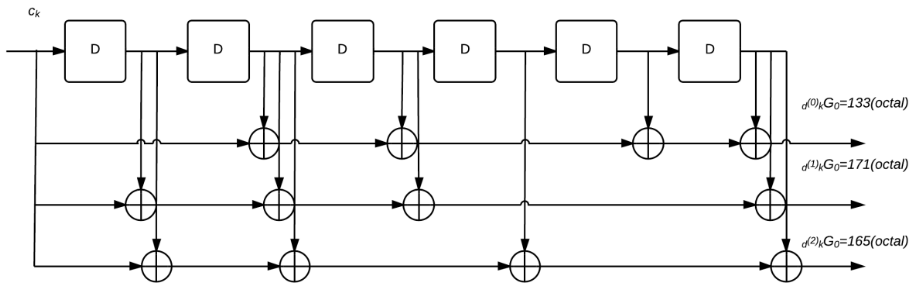


Figure 14 Structure of rate 1/3 tail biting convolutional encoder [20].

As illustrated from Figure 14 that c_k is the input block of bits and $d_k^{(0)}$, $d_k^{(1)}$ and $d_k^{(2)}$ are the first, second and third parity streams of the encoder output. The encoding process can also be described by Equation 2.3.26 to 2.3.28

$$d_k^{(0)} = (c_{k+6} + c_{k+4} + c_{k+3} + c_{k+1} + c_k) \text{mod } 2 \text{ for } k = 0, 1, \dots, K-1 \quad (2.3.26)$$

$$d_k^{(1)} = (c_{k+6} + c_{k+5} + c_{k+4} + c_{k+3} + c_k) \text{mod } 2 \text{ for } k = 0, 1, \dots, K-1 \quad (2.3.27)$$

$$d_k^{(2)} = (c_{k+6} + c_{k+5} + c_{k+4} + c_{k+2} + c_k) \text{mod } 2 \text{ for } k = 0, 1, \dots, K-1 \quad (2.3.28)$$

where K is the length of c_k. There are six shift registers used in the encoder named from left to right as s_0 to s_5. The initial values in the shift registers are set according to Equation 2.3.29.

$$s_i = c_{(K-1-i)} \text{ for } i = 0, 1, \dots, 5 \quad (2.3.29)$$

The input block of bits c_k is shifted through the encoder until the length of output blocks of bits reach K.

2.3.3.4 Rate matching

The purpose of the rate matching process is to produce an output sequence with a desired length. The desired length E is decided by the PDCCH format used for transmission according to Table 15. The rate matching method described in this section is according to [21].

Table 15 Rate matching output length with different PDCCH formats [22].

PDCCH format	E
0	27
1	144
2	288
3	576

The PDCCH format used is decided by PDCCH aggregation level which is explained later in Section 2.3.5. The overview of rate matching process is illustrated in Figure 15.

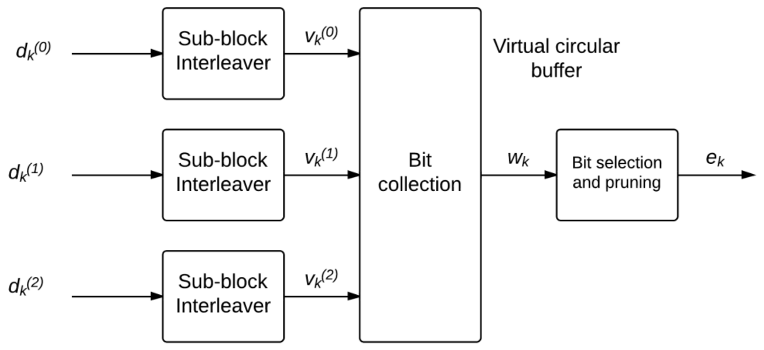


Figure 15 Overview of rate matching [21].

During the rate matching process, the output bit sequence $d_k^{(0)}$, $d_k^{(1)}$ and $d_k^{(2)}$ undergoes a sub-block interleaver respectively. The purpose of interleaving is to improve the performance of forward error correction. The output bit sequences of the interleaver $v_k^{(i)}$ for $i = 0{,}1{,}2$ are derived with following steps.

- Set the number of columns of the interleaver matrix $C_{subblock}^{CC}$ to 32. Numbered from 0 to $C_{subblock}^{CC} - 1$ on the matrix from left to right.
- Calculate the minimum integer number of rows $R_{subblock}^{CC}$ of the interleaver matrix $R_{subblock}^{CC}$ according to Equation 2.3.30.

$$D \leq (R_{subblock}^{CC} \times C_{subblock}^{CC}) \tag{2.3.30}$$

  where D is the length of input sequence. The rows are numbered from 0 to $R_{subblock}^{CC} - 1$ on the matrix from top to bottom.

- If the size of the matrix is larger than D then N_D number of dummy bits are padded. The generated sequence y_k after adding dummy bits is derived according to Equation 2.3.31 and 2.3.32.

$$y_k =< NULL > \quad \text{for } k = 0, 1, \ldots, N_D - 1 \tag{2.3.31}$$

$$y_{N_D+k} = d_k^{(i)} \quad \text{for } k = 0, 1, \ldots, D - 1 \tag{2.3.32}$$

  where N_D is calculated according to Equation 2.3.33.

$$N_D = (R_{subblock}^{CC} \times C_{subblock}^{CC} - D) \tag{2.3.33}$$

Perform the inter-column permutation of the matrix y_P according to Equation 2.3.34.

$$y_{P,(i,j)} = y_{P_{j+(i-1)\cdot C_{subblock}^{CC}+1}} \tag{2.3.34}$$

where $i = 0, 1, \dots, R_{subblock}^{CC} - 1$, $j = 0, 1, \dots, C_{subblock}^{CC} - 1$ and P is the pattern sequence shown in Table 16.

Table 16 Inter-column permutation pattern [21].

$C_{subblock}^{CC}$	$< P_0, P_1, \dots, P_{C_{subblock}^{CC}-1} >$
32	$< 1, 17, 9, 25, 5, 21, 13, 29, 3, 19, 11, 27, 7, 23, 15, 31,$ $0, 16, 8, 24, 4, 20, 12, 28, 2, 18, 10, 26, 6, 22, 14, 30 >$

- Generate the output bit sequence $v_k^{(i)}$ for $i = 0, 1, 2$ by reading out matrix y_P column by column.

After interleaving process, the bit sequence $v_k^{(i)}$ undergoes a circular buffer to do bit collection, selection and pruning. The bit sequence w_k inside the circular buffer is derived according to Equation 2.3.35 to 2.3.37

$$w_k = v_k^{(0)} \quad \text{for } k = 0, \dots, K-1 \tag{2.3.35}$$

$$w_{k+K} = v_k^{(1)} \quad \text{for } k = 0, \dots, K-1 \tag{2.3.36}$$

$$w_{k+2\times K} = v_k^{(2)} \text{ for } k = 0, \dots, K-1 \tag{2.3.37}$$

where *K* is the length of $v_k^{(i)}$.

The output of rate matching process e_k for $k = 0,1, \dots, E-1$ is generated by reading out bits in the circular buffer continuously from beginning to end until the output length E is matched. All <*NULL*> bits are ignored and discard during this process.

2.3.3.5 PDCCH multiplexing

Every PDCCH is transmitted in an aggregation level which indicates the number of consecutive control channel elements (CCEs) occupied by the PDCCH. Each CCE corresponds to 9 REGs. The total number of CCEs available on the PDCCH (N_{CCE}) is derived from Equation 2.3.38 [8].

$$N_{CCE} = \lfloor N_{REG}/9 \rfloor \tag{2.3.38}$$

where N_{REG} is the number of REGs unused by PCFICH or PHICH. As multiple PDCCHs can be transmitted within one subframe and each PDCCH needs to be mapped without overlapping, all PDCCHs must be multiplexed together properly. The PDCCH multiplexing method is according to [23] and Section 6.8.2 in [22].

The basic principle of PDCCH multiplexing is that the index of staring CCE for a PDCCH must be integer times of the aggregation level of the PDCCH. As the UE have no information about the DCI format nor the aggregation level of the PDCCH is using, the PDCCH search space is introduced in addition to the basic principle to simply the decoding process. There are two types of search space: common search space and UE-specific search space. Common search space carries PDCCHs which will be monitored by all UEs while UE-specific search space only carriers PDCCHs scrambled with C-RNTI for specified UEs. There are several groups of CCEs available for PDCCHs within a search space and each possible group of CCEs to allocate a PDCCH is known as a PDCCH candidate. The size of the search space, number of PDCCH candidates are decided by the search space type and PDCCH aggregation level as shown in Table 17.

Table 17 Structure of PDCCH search spaces [23].

PDCCH format	Search space $S_k^{(L)}$			
	Type	Aggregation level L	Number of candidates $M^{(L)}$	Size in CCEs
0	UE-specific	1	6	6
1		2	6	12
2		4	2	8
3		8	2	16
2	Common	4	4	16
3		8	2	16

The CCE positions used to map a PDCCH with aggregation level L is derived according to Equation 2.3.39.

$$\left\{(Y_k + m')\text{mod}\left\lfloor\frac{N_{CCE,k}}{l}\right\rfloor\right\} + i \text{ for } i = 0, \ldots, L-1 \tag{2.3.39}$$

For the UE-specific search space, if the UE is configured with carrier indicator field m' is according to Equation 2.3.40.

$$m' = m + M^{(L)} \times n_{CI} \tag{2.3.40}$$

where n_{CI} is the value of the carrier indicator and $M^{(L)}$ is the number of PDCCH candidates in the search space. For all other cases, m' is according to Equation 2.3.41.

$$m' = m \tag{2.3.41}$$

For the common search space, $Y_k$is set to 0. For the UE-specific search space Y_k is according to Equation 2.3.42.

$$Y_k = (A \times Y_{k-1})\text{mod}D \tag{2.3.42}$$

where $k = \lfloor n_s/2 \rfloor, Y_{-1} = n_{RNTI} \neq 0, A = 39827, D = 65537$ with n_{RNTI} is the RNTI value which is not equal to 0.

The common search space always starts from the first CCE index. PDCCHs mapped into common search space are mapped first. If a group of CCEs in common search space is not used, PDCCH in UE-specific search space can be mapped on those CCEs. This means there is a possible overlap between CCE used by common and UE-specific search space.

After PDCCH multiplexing, a block of bits b_k for $i = 0,1, \dots, L_{PDCCH} - 1$ is generated where L_{PDCCH} denotes the length of bits to occupy all REGs not used by PCFICH or PHICH in the control region. All bits in b_k not used by PDCCH are set to <NIL> bits.

2.3.3.6 PDCCH scrambling

The multiplexed PDCCH bit sequence b_k is scrambled according to the method mentioned in Section 2.2.1. Note that the scrambling sequence is initialized according to Section 6.8.2 in [22]. The bit sequence after scrambling is B_i for $i = 0,1, \dots, L_{PDCCH} - 1$.

2.3.3.7 PDCCH modulation

The PDCCH modulation method is QPSK modulation as mentioned in Section 2.2.2. The input bit sequence is B_i and the output block of complex-valued symbols is d_i where $i = 0,1, \dots, \frac{L_{PDCCH}}{2} - 1$.

2.3.3.8 PDCCH layer mapping

The block of complex-valued symbols d_i is mapped onto 2 layers for transmit diversity with the method mentioned in Section 2.2.3. The index $M_{symb}^{(0)}$ for layer mapping is set to $\frac{L_{PDCCH}}{2}$. The output block of complex-valued symbols after mapping is $x_i^{(p)}$ where p is the layer index and $i = 0,1, \dots, \frac{L_{PDCCH}}{4} - 1$.

2.3.3.9 PDCCH precoding

The block of complex-valued symbols $x_i^{(p)}$ is precoded for 2 antenna ports with the method mentioned in Section 2.2.4. The output block of complex-valued symbols after precoding is $y_i^{(p)}$ where p is the antenna port index and $i = 0,1, \dots, \frac{L_{PDCCH}}{2} - 1$.

2.3.3.10 Mapping PDCCH to REs

The PDCCH RE mapping method described in this section is according to Section 6.8.5 in [8]. The block of complex-valued symbols $y_i^{(p)}$ is divided into symbol quadruplets $z_i^{(P)}$ as mentioned in Section 2.2.5. The generated symbol quadruplets undergo a quadruplet interleaver similar as the sub-interleaver mentioned in Section 2.3.3.4. Notice that *<NULL>* elements shall be added as dummy quadruplet, if necessary, when generating the interleaver matrix and shall be discarded when generating output quadruplets $v_i^{(P)}$.

The block of quadruplets $v_i^{(P)}$ is then cyclically shifted according to Equation 2.3.43.

$$w_i^{(P)} = v_{(i+N_{ID}^{cell})\mathrm{mod}M_{quad}}^{(P)} \text{ for } i = 0, \dots, M_{quad} - 1 \tag{2.3.43}$$

where M_{quad} is the length of $v_i^{(P)}$.

The block of quadruplets $w_i^{(P)}$ is then mapped in terms of REGs. This process maps the quadruplets to every REG $(k_i^{PCFICH}, l_i^{PCFICH})$ unused by PCFICH or PHICH. REGs for PDCCH are used in OFDM symbol order. Starting from $(k_i^{PCFICH} = 0, l_i^{PCFICH} = 0)$, k_i^{PCFICH} is increased by one after l_i^{PCFICH} has been increased to $L - 1$ where L is the number of OFDM symbols used for the control region. This means all REGs with index k_i^{PCFICH} have been mapped. After this process, every REG in the control region is mapped with a symbol quadruplet.

2.4 Decoder algorithms

The decoder algorithms are designed to retrieve the control information encoded during L1/L2 control signaling. Detailed pressures of PCFICH decoding, PHICH decoding and PDCCH decoding are described in Section 2.4.1 to Section 2.4.3.

2.4.1 PCFICH Decoding

The PCFICH is the first channel that UE shall decode as it informs the UE about the size of the control region. The decoding process is described in this section. The overview of PCFICH decoding is illustrated in Figure 16.

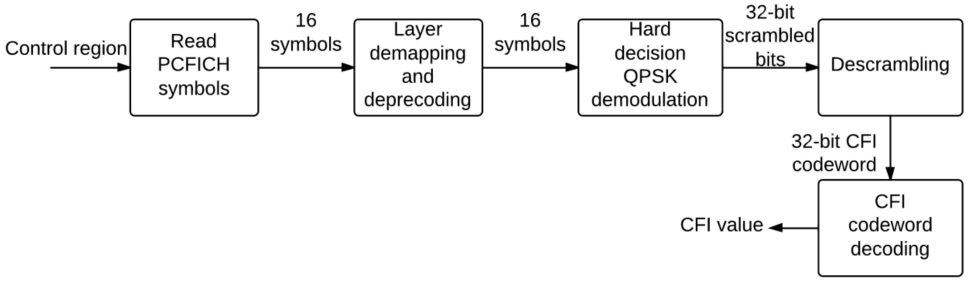


Figure 16 Overview of PCFICH decoding

2.4.1.1 Read PCFICH Symbols

To decode the PCFICH, UE shall first find the REGs used for mapping PCFICH quadruplets as mentioned in Section 2.3.1.6. Blocks of complex-valued symbols on both antenna ports are read according to REG positions. The block of complex-valued symbols is denoted by $r_i^{(p)}$ for $i = 0,1, \dots, 15$ where $p = 0,1$ is the antenna port.

2.4.1.2 PCFICH Layer Demapping and Deprecoding

The block of PCFICH complex-valued symbols $r_i^{(p)}$ is layer demapping and deprecoding with the method mentioned in Section 2.2.8. The output block of complex-valued symbols is denoted by d_i for $i = 0,1, \dots ,15$.

2.4.1.3 PCFICH Demodulation

The PCFICH demodulation method is hard decision QPSK demodulation as mentioned in Section 2.2.7. The input block of complex-valued symbols is d_i and the output block of bits is B_i where $i = 0,1, \dots ,31$.

2.4.1.4 PCFICH Descrambling

The PCFICH descrambling process is according to the method mentioned in Section 2.2.6. The input block of bits is B_i and the output is block of bits is the CFI codeword b_i for $i = 0,1, \dots ,31$.

2.4.1.5 CFI Codeword Decoding

The CFI codeword b_i is decoded by UE after descrambling. The decoding is processed by comparing the codeword bit by bit with standard CFI codewords as in Table 5. The decoded CFI value is chosen from range 1 to 3 which the corresponding CFI codeword has most matches with the decoded CFI codeword. The UE is informed about the PHICH duration from higher layer. If the PHICH duration is extended and is larger than the decode CFI when $N_{RB}^{DL} > 10$, UE shall assume the CFI value equals to the PHICH duration [24]. If $N_{RB}^{DL} \leq 10$, the decoded CFI value 1 to 3 corresponds to 2 to 4 OFDM symbols occupied by control region. Otherwise the decoded CFI value equals to the number of OFDM symbols occupied by control region.

2.4.2 PHICH Decoding

The PHICH is decoded after PCFICH even if PCFICH decoding failed. The UE should know the PHICH group index and the orthogonal sequence index of the individual PHICH it is expecting for before decoding. The PHICH decoding process is described in Section 2.4.2.1 to Section 2.4.6. The overview of PHICH decoding process is illustrated in Figure 17.

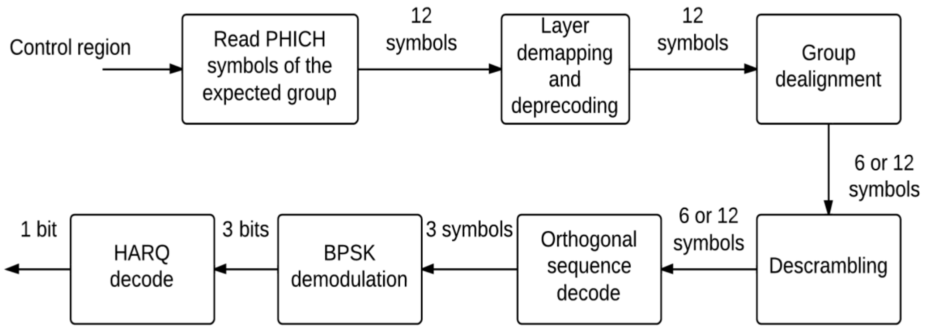


Figure 17 Overview of PHICH decoding

2.4.2.1 Read PHICH Symbols

To decode the PHICH, UE shall first find the REGs used for mapping the expected group of PHICH quadruplets as mentioned in Section 2.3.2.7. The PHICH group index r_{PHICH}^{group} used to read PHICH quadruplets is generated according to Equation 2.4.1.

$$r_{PHICH}^{group} = \begin{cases} n_{PHICH}^{group} & \text{for normal cyclic prefix} \\ \lfloor n_{PHICH}^{group}/2 \rfloor & \text{for extended cyclic prefix} \end{cases} \tag{2.4.1}$$

where n_{PHICH}^{group} is the PHICH group index configured at the UE. The reason the PHICH group index is modified for extended cyclic prefix is that every two PHICH groups are aligned into one PHCIH group at resource group alignment mentioned in Section 2.3.2.4. During the PHICH symbol reading process, blocks of complex-valued symbols on both antenna ports are read according to REG positions. The block of complex-valued symbols is denoted by $r_i^{(p)}$ for $i = 0,1,\dots,11$ where $p = 0,1$ is the antenna port.

2.4.2.2 PHICH Layer Demapping and Deprecoding

The block of PHICH complex-valued symbols $r_i^{(p)}$ undergoes layer demapping and deprecoding with the method mentioned in Section 2.2.8. The output block of complex-valued symbols is denoted by $d_i^{(0)}$ for $i = 0,1,\dots,11$.

2.4.2.3 PHICH Group Dealignment

The block of complex-valued symbols $d_i^{(0)}$ undergoes the reverse group alignment depending on the cyclic prefix type of the subframe.

For normal cyclic prefix, the block of symbols d_i is according to Equation 2.4.2.

$$d_i = d_i^{(0)} \text{ for } i = 0,\dots,M_{symb} - 1 \tag{2.4.2}$$

where M_{symb} is generated same as in Section 2.3.2.3. For extended cyclic prefix, the block of symbols d_i is generated according to Equation 2.4.3 and 2.4.4.

$$d_{2i} = \begin{cases} d_{4i}^{(0)} & \text{if } n_{PHICH}^{group} \bmod 2 = 0 \\ d_{4i+2}^{(0)} & \text{if } n_{PHICH}^{group} \bmod 2 = 1 \end{cases} \quad \text{for } i = 0, \ldots, \left(\frac{M_{symb}}{2}\right) - 1 \qquad (2.4.3)$$

$$d_{2i+1} = \begin{cases} d_{4i+1}^{(0)} & \text{if } n_{PHICH}^{group} \bmod 2 = 0 \\ d_{4i+3}^{(0)} & \text{if } n_{PHICH}^{group} \bmod 2 = 1 \end{cases} \quad \text{for } i = 0, \ldots, \left(\frac{M_{symb}}{2}\right) - 1 \qquad (2.4.4)$$

where n_{PHICH}^{group} is the group index of PHICH the UE is expecting for.

2.4.2.4 PHICH Descrambling

The input of PHICH descrambling is the block of symbols d_i. The scrambling sequence c_i for descrambling is generated as mentioned in Section 2.2.1. Then descramble block of symbols b_i is generated according to Equation 2.4.5.

$$b_i = \frac{d_i}{1 - 2 \cdot c_i} \text{ for } i = 0, \ldots, M_{symb} - 1 \qquad (2.4.5)$$

The output block of symbols b_i denotes the added result of unscrambled PHICH symbols within in the PHICH group.

2.4.2.5 Orthogonal Sequence Decode

The UE needs to decode the expected PHICH within the PHICH group. As the PHICH within in the PHICH group in multiplexed with different orthogonal sequence, the PHICH can be separated from the added result. As the number of PHICHs within in a group is variable from 0 to 8 for normal cyclic prefix and 0 to 4 for extended cyclic prefix, the decoding process is separated into different cases. Block of symbols $a_i^{(r)}$ for $i = 0,1,2$ denotes the block of symbols for PHICH with index r.

In case of normal cyclic prefix, a block of symbols S_i^n for $n = 0,1,2,3$ is generated according to Equation 2.4.6 to 2.4.9.

$$S_i^0 = b_{4i} + b_{4i+1} + b_{4i+2} + b_{4i+3} \qquad (2.4.6)$$

$$S_i^1 = \begin{cases} b_{4i+1} + b_{4i+3} - 2 \cdot a_i^0 - 2 \cdot a_i^4 \cdot j & \text{for abs}\left(\text{Re}(S_i^0)\right) + \text{abs}\left(\text{Im}(S_i^0)\right) \geq 6 \\ b_{4i+1} + b_{4i+3} & \text{for abs}\left(\text{Re}(S_i^0)\right) + \text{abs}\left(\text{Im}(S_i^0)\right) < 6 \end{cases} \qquad (2.4.7)$$

$$S_i^2 = \begin{cases} b_{4i+2} + b_{4i+3} - 2 \cdot a_i^0 - 2 \cdot a_i^4 \cdot j & \text{for abs}\left(\text{Re}(S_i^0)\right) + \text{abs}\left(\text{Im}(S_i^0)\right) \geq 6 \\ b_{4i+2} + b_{4i+3} & \text{for abs}\left(\text{Re}(S_i^0)\right) + \text{abs}\left(\text{Im}(S_i^0)\right) < 6 \end{cases} \qquad (2.4.8)$$

$$S_i^3 = \begin{cases} b_{4i+1} + b_{4i+2} - 2 \cdot a_i^0 - 2 \cdot a_i^4 \cdot j & \text{for abs}\left(\text{Re}(S_i^0)\right) + \text{abs}\left(\text{Im}(S_i^0)\right) \geq 6 \\ b_{4i+1} + b_{4i+2} & \text{for abs}\left(\text{Re}(S_i^0)\right) + \text{abs}\left(\text{Im}(S_i^0)\right) < 6 \end{cases} \qquad (2.4.9)$$

where $i = 0,1,2$ and b_i is the block of symbols after descrambling. If $\text{abs}\left(\text{Re}(S_i^0)\right) + \text{abs}\left(\text{Im}(S_i^0)\right) \geq 5$ then at least one of the first or fifth PHICH within the group is used. Otherwise, neither is used. If the at least one of the first or fifth PHICH within the group is used a_i^0 and a_i^4 are derived according to Table 18.

Table 18 Algorithm for calculating a_i^0 and a_i^4.

$-4 < \text{abs}\left(\text{Re}(S_i^0)\right) - \text{abs}\left(\text{Im}(S_i^0)\right) < 4$			$-4 \geq \text{abs}\left(\text{Re}(S_i^0)\right) - \text{abs}\left(\text{Im}(S_i^0)\right)$ or $\text{abs}\left(\text{Re}(S_i^0)\right) - \text{abs}\left(\text{Im}(S_i^0)\right) \geq 4$		
Cases	a_i^0	a_i^4	Cases	a_i^0	a_i^4
$\text{Re}(S_i^0) < 0$ & $\text{Im}(S_i^0) < 0$	$-1-1i$	0	$\text{Re}(S_i^0) \geq 0$ & $\text{abs}\left(\text{Re}(S_i^0)\right) > \text{abs}\left(\text{Im}(S_i^0)\right)$	$1+1i$	$-1-1i$
$\text{Re}(S_i^0) < 0$ & $\text{Im}(S_i^0) \geq 0$	0	$1+1i$	$\text{Re}(S_i^0) < 0$ & $\text{abs}\left(\text{Re}(S_i^0)\right) > \text{abs}\left(\text{Im}(S_i^0)\right)$	$-1-1i$	$1+1i$
$\text{Re}(S_i^0) \geq 0$ & $\text{Im}(S_i^0) < 0$	0	$-1-1i$	$\text{Re}(S_i^0) < 0$ & $\text{abs}\left(\text{Re}(S_i^0)\right) < \text{abs}\left(\text{Im}(S_i^0)\right)$	$-1-1i$	$-1-1i$
$\text{Re}(S_i^0) \geq 0$ & $\text{Im}(S_i^0) \geq 0$	$1+1i$	0	$\text{Re}(S_i^0) \geq 0$ & $\text{abs}\left(\text{Re}(S_i^0)\right) < \text{abs}\left(\text{Im}(S_i^0)\right)$	$1+1i$	$1+1i$

PHICH symbols with value 0 imply that the PHICH is not included in the group. PHICH symbols with other PHICH index are derived according to Table 19.

Table 19 Algorithm for calculating PHICH symbols except a_i^0 and a_i^4.

$-2 < \text{abs}(\text{Re}(S_i^n)) - \text{abs}(\text{Im}(S_i^n)) < 2$ for $n = 1,2,3$			$-2 \geq \text{abs}(\text{Re}(S_i^n)) - \text{abs}(\text{Im}(S_i^n))$ or $\text{abs}(\text{Re}(S_i^n)) - \text{abs}(\text{Im}(S_i^n)) \geq 2$ for $n = 1,2,3$		
Cases	$a_i^{1,2,3}$	$a_i^{5,6,7}$	Cases	$a_i^{1,2,3}$	$a_i^{5,6,7}$
$\text{Re}(S_i^n) < 0$ & $\text{Im}(S_i^n) < 0$	$1+1i$	0	$\text{Re}(S_i^n) \geq 0$ & $\text{abs}(\text{Re}(S_i^n)) > \text{abs}(\text{Im}(S_i^n))$	$-1-1i$	$1+1i$
$\text{Re}(S_i^n) < 0$ & $\text{Im}(S_i^n) \geq 0$	0	$-1-1i$	$\text{Re}(S_i^n) < 0$ & $\text{abs}(\text{Re}(S_i^n)) > \text{abs}(\text{Im}(S_i^n))$	$1+1i$	$-1-1i$
$\text{Re}(S_i^n) \geq 0$ & $\text{Im}(S_i^n) < 0$	0	$1+1i$	$\text{Re}(S_i^0) < 0$ & $\text{abs}(\text{Re}(S_i^n)) < \text{abs}(\text{Im}(S_i^n))$	$1+1i$	$1+1i$
$\text{Re}(S_i^n) \geq 0$ & $\text{Im}(S_i^n) \geq 0$	$-1-1i$	0	$\text{Re}(S_i^0) \geq 0$ & $\text{abs}(\text{Re}(S_i^n)) < \text{abs}(\text{Im}(S_i^n))$	$-1-1i$	$-1-1i$

In case of extended cyclic prefix, a block of symbols S_i^n for $n = 0{,}1$ is generated according to Equation 2.4.10 and 2.4.11.

$$S_i^0 = b_{4i} + b_{4i+1} \tag{2.4.10}$$

$$S_i^1 = \begin{cases} b_{4i} - 2 \cdot a_i^0 - 2 \cdot a_i^2 \cdot j & \text{for abs}\left(\text{Re}(S_i^0)\right) + \text{abs}\left(\text{Im}(S_i^0)\right) \geq 3 \\ b_{4i} & \text{for abs}\left(\text{Re}(S_i^0)\right) + \text{abs}\left(\text{Im}(S_i^0)\right) < 3 \end{cases} \tag{2.4.11}$$

where $i = 0{,}1{,}2$ and b_i is the block of symbols after descrambling. If $\text{abs}\left(\text{Re}(S_i^0)\right) + \text{abs}\left(\text{Im}(S_i^0)\right) \geq 3$ then at least one of the first or third PHICH within the group is used. Otherwise, neither is used. If the at least one of the first or third PHICH within the group is used a_i^0 and a_i^2 are derived according to Table 20.

Table 20 Algorithm for calculating a_i^0 and a_i^2.

$-2 < \text{abs}\left(\text{Re}(S_i^0)\right) - \text{abs}\left(\text{Im}(S_i^0)\right) < 2$			$-2 \geq \text{abs}\left(\text{Re}(S_i^0)\right) - \text{abs}\left(\text{Im}(S_i^0)\right)$ or $\text{abs}\left(\text{Re}(S_i^0)\right) - \text{abs}\left(\text{Im}(S_i^0)\right) \geq 2$		
Cases	a_i^0	a_i^2	Cases	a_i^0	a_i^2
$\text{Re}(S_i^0) < 0$ & $\text{Im}(S_i^0) < 0$	$-1 - 1i$	0	$\text{Re}(S_i^0) \geq 0$ & $\text{abs}\left(\text{Re}(S_i^0)\right) > \text{abs}\left(\text{Im}(S_i^0)\right)$	$1 + 1i$	$-1 - 1i$
$\text{Re}(S_i^0) < 0$ & $\text{Im}(S_i^0) \geq 0$	0	$1 + 1i$	$\text{Re}(S_i^0) < 0$ & $\text{abs}\left(\text{Re}(S_i^0)\right) > \text{abs}\left(\text{Im}(S_i^0)\right)$	$-1 - 1i$	$1 + 1i$
$\text{Re}(S_i^0) \geq 0$ & $\text{Im}(S_i^0) < 0$	0	$-1 - 1i$	$\text{Re}(S_i^0) < 0$ & $\text{abs}\left(\text{Re}(S_i^0)\right) < \text{abs}\left(\text{Im}(S_i^0)\right)$	$-1 - 1i$	$-1 - 1i$
$\text{Re}(S_i^0) \geq 0$ & $\text{Im}(S_i^0) \geq 0$	$1 + 1i$	0	$\text{Re}(S_i^0) \geq 0$ & $\text{abs}\left(\text{Re}(S_i^0)\right) < \text{abs}\left(\text{Im}(S_i^0)\right)$	$1 + 1i$	$1 + 1i$

The second and fourth PHICH in the group a_i^1 and a_i^3 are derived according to Table 21.

Table 21 Algorithm for calculating a_i^1 and a_i^3.

$-1 < \mathrm{abs}\left(\mathrm{Re}(S_i^1)\right) - \mathrm{abs}\left(\mathrm{Im}(S_i^1)\right) < 1$			$-1 \geq \mathrm{abs}\left(\mathrm{Re}(S_i^1)\right) - \mathrm{abs}\left(\mathrm{Im}(S_i^1)\right)$ or $\mathrm{abs}\left(\mathrm{Re}(S_i^1)\right) - \mathrm{abs}\left(\mathrm{Im}(S_i^1)\right) \geq 1$		
Cases	a_i^0	a_i^2	Cases	a_i^0	a_i^2
$\mathrm{Re}(S_i^1) < 0$ & $\mathrm{Im}(S_i^1) < 0$	$-1-1i$	0	$\mathrm{Re}(S_i^1) \geq 0$ & $\mathrm{abs}\left(\mathrm{Re}(S_i^1)\right) > \mathrm{abs}\left(\mathrm{Im}(S_i^1)\right)$	$1+1i$	$-1-1i$
$\mathrm{Re}(S_i^1) < 0$ & $\mathrm{Im}(S_i^1) \geq 0$	0	$1+1i$	$\mathrm{Re}(S_i^1) < 0$ & $\mathrm{abs}\left(\mathrm{Re}(S_i^1)\right) > \mathrm{abs}\left(\mathrm{Im}(S_i^1)\right)$	$-1-1i$	$1+1i$
$\mathrm{Re}(S_i^1) \geq 0$ & $\mathrm{Im}(S_i^1) < 0$	0	$-1-1i$	$\mathrm{Re}(S_i^1) < 0$ & $\mathrm{abs}\left(\mathrm{Re}(S_i^1)\right) < \mathrm{abs}\left(\mathrm{Im}(S_i^1)\right)$	$-1-1i$	$-1-1i$
$\mathrm{Re}(S_i^1) \geq 0$ & $\mathrm{Im}(S_i^1) \geq 0$	$1+1i$	0	$\mathrm{Re}(S_i^1) \geq 0$ & $\mathrm{abs}\left(\mathrm{Re}(S_i^1)\right) < \mathrm{abs}\left(\mathrm{Im}(S_i^1)\right)$	$1+1i$	$1+1i$

The UE only need to calculate the PHICH with orthogonal index expected instead of all PHICHs within the group.

2.4.2.6 PHICH Demodulation

The PHICH demodulation method is BPSK demodulation according to the method mentioned in Section 2.2.7. Only the expected PHICH is demodulated. The input block of complex-valued symbols is d_i and the output block of bits is β_i where $i = 0,1,2$.

2.4.2.7 HARQ Acknowledgement Decode

The HARQ acknowledgement decode is done by summing β_0 , β_1 and β_2. If the accumulated result equals to 3, the HARQ is considered ACK. Otherwise the HARQ is considered NACK. The reason of preferring NACK instead of ACK is mentioned at the beginning of Section 2.3.2. A NACK-to-ACK error results in a loss of transport block which is more critical than an ACK-to-NACK error resulting in a retransmission of successfully decoded block.

2.4.3 PDCCH Decoding

The PDCCH carries the largest payload in the control region. As the UE has no information about the DCI format, DCI data length or the aggregation level that the base station is using, the decoding of PDCCH is referred as blind decoding. This means the UE needs to try all possibilities until the expected data is successfully decoded. The overview of PDCCH decoding is illustrated in Figure 18 and the detailed steps of the decoding process are described in this section.

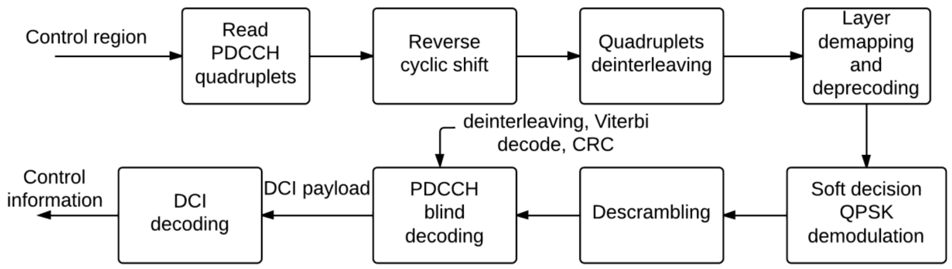


Figure 18 Overview of PDCCH decoding

2.4.3.1 Read PDCCH Symbols

To decode the PHICH, UE shall first find the REGs used for mapping PDCCH quadruplets as mentioned in Section 2.3.3.10. Blocks of complex-valued quadruplets on both antenna ports are read according to REG positions. The block of complex-valued quadruplets is denoted by $w_i^{(p)}$ for $i = 0,1,\dots,N$ where $p = 0,1$ is the antenna port and N is the total number of REGs used by PDCCHs.

2.4.3.2 Reverse Cyclic Shift

Complex-valued quadruplets $w_i^{(p)}$ undergoes the reverse cyclic shift to generate symbol quadruplets $v_i^{(p)}$ according to Euqation 2.4.12.

$$v^{(p)}_{(i+N_{ID}^{cell})\mathrm{mod}M_{quad}} = w_i^{(p)} \text{ for } i = 0,\dots,M_{quad}-1 \tag{2.4.12}$$

where M_{quad} is the length of $w_i^{(P)}$.

2.4.3.3 Deinterleaving

The block of quadruplets $v_i^{(p)}$ generated by reverse cyclic shift undergoes a deinterleaving block to generate block of quadruplets $z_i^{(p)}$. The deinterleaving process is described in the following steps:

- Calculate deinterleaving parameters $R_{subblock}^{CC}, C_{subblock}^{CC}, N_D$ with the same method as mentioned in Section 2.3.3.4 using quadruplets $v_i^{(p)}$ as input instead of a block of bits.
- Add N_D number of $< NULL >$ elements to a block of dummy bits which has the same length as $v_i^{(p)}$. All elements of the dummy block are set to 1.
- Perform the interleaving of the dummy block. Find out the positions of all $< NULL >$ elements after interleaving and record as $NULL_i$ for $i = 0,..,N_D - 1$.

- Add $< NULL >$ elements to $v_i^{(p)}$ at position $NULL_i$ and generate deinterleaving matrix $y_{P,(i,j)}$ where $i = 0, 1, \dots, R_{subblock}^{CC} - 1$ and $j = 0, 1, \dots, C_{subblock}^{CC} - 1$
- Generate deinterleaving block $y_i \; for \; i = 0,1,.., R_{subblock}^{CC} \cdot C_{subblock}^{CC} - 1$ according to Equation 2.4.13.

$$y_{P_{j+(i-1)\cdot C_{subblock}^{CC}+1}} = y_{P,(i,j)} \tag{2.4.13}$$

  where $i = 0, 1, \dots, R_{subblock}^{CC} - 1$ and $j = 0, 1, \dots, C_{subblock}^{CC} - 1$.

- Generate output block $z_i^{(p)}$ by removing all $< NULL >$ elements according to Equation 2.4.14.

$$z_i^{(p)} = y_{N_D+k} \quad \text{for } k = 0, 1, \dots, D - 1 \tag{2.4.14}$$

  where D is the length of $v_i^{(p)}$.

- After deinterleaving, the block of quadruplets is separated into a larger block of complex-valued symbols $r_i^{(p)}$ for $i = 0,1,\dots,4 \cdot D - 1$ according to Equation 2.4.15.

$$\langle r_{4\cdot i}^{(p)}, r_{4\cdot i+1}^{(p)}, r_{4\cdot i+2}^{(p)}, r_{4\cdot i+3}^{(p)} \rangle = z_i^{(p)} \text{ for } i = 0,1,\dots,D - 1 \tag{2.4.15}$$

2.4.3.4 PDCCH Layer Demapping and Deprecoding

The block of PDCCH complex-valued symbols $r_i^{(p)}$ undergoes layer demapping and deprecoding with the method mentioned in Section 2.2.8. The output block of complex-valued symbols is denoted by $d_i^{(0)}$ for $i = 0,1,\dots,N - 1$ where N is the length of $r_i^{(p)}$.

2.4.3.5 PDCCH Demodulation

The PDCCH demodulation method is soft decision QPSK demodulation as mentioned in Section 2.2.7. The input block of complex-valued symbols is d_i and the output block of integers is B_i for $i = 0,1,\dots,2 \cdot N - 1$ where N is the length of d_i.

2.4.3.6 PDCCH Descrambling

- The block of integers B_i is descrambled to generate a block of integers $b_i$according to Equation 2.4.16.

$$b_i = \begin{cases} B_i & \text{for } c_i = 0 \\ 7 - B_i & \text{for } c_i = 1 \end{cases} \text{ for } i = 0,1,\dots,N - 1 \tag{2.4.16}$$

where N is the length of B_i and c_i is scrambling sequence generated according to Section 2.2.1.

2.4.3.7 Viterbi Decoder

The Viterbi decoder is used to decode a bit stream that is encoded by convolutional encoding as mentioned in Section 2.3.3.3. The input stream contains integers range for 0 to 7 indicating the reliability of the received symbols with relation shown in Table 22.

Table 22 Relation between input integer and reliability of received symbol [24].

Integer	Received symbol
0	Strongest 0
1	Relatively strong 0
2	Relatively weak 0
3	Weakest 0
4	Weakest 1
5	Relatively weak 1
6	Relatively strong 1
7	Strongest 1

There are three parts in the Viterbi decoder: branch metric unit (BMU), path metric unit (PMU) and traceback unit (TBU). The BMU calculates the branch metrics which contains the distance between every possible symbol before convolution encoding and the input symbol. The PMU summarizes the branch metric to get 2^{K-1} paths and choose the optimal one where K is the Viterbi constrains length. The TBU stores the optimal path and output the results in bit reverse order [25]. In downlink control signal decoding, the Viterbi decoder is configured with $K = 7$, output rate $R = 3$ and input trellis $p = 133, 165, 171$.

2.4.3.8 PDCCH CRC Calculation

The PDCCH CRC calculation checks whether the received block of bits is the expected DCI. The process is done in following steps.

- The input block of bits c_k with length N is first descrambled with RNTI to generate a block of bits b_k according to Equation 2.4.17.

$$b_k = \begin{cases} \begin{cases} c_k & \text{for } k = 0,1,,2,\dots,A-1 \\ (c_k + x_{rnti,k-A})\text{mod}2 & \text{for } k = A, A+1,\dots,A+15 \end{cases} & \text{for case 1} \\ \begin{cases} c_k & \text{for } k = 0,1,,2,\dots,A-1 \\ (c_k + x_{rnti,k-A} + x_{AS,k-A})\text{mod}2 & \text{for } k = A, A+1,\dots,A+15 \end{cases} & \text{for case 2} \end{cases} \tag{2.4.17}$$

  where $\mathrm{A} = N - 16$. Case 1 indicates UE transmit antenna selection is not configured or applicable and case 2 indicates UE transmit antenna selection is configure and applicable. x_{rnti} is the corresponding RNTI with $x_{rnti,0}$ indicates the MSB of the RNTI and x_{AS} is the antenna selection mask generated according to Table 15.

- Generate the cyclic polynomial bits $g_{CRC16}(D)$ according to Section 2.3.3.2.

- Select 16 bits from b_k, denoted by b_i to b_{i+15}, starting from $i = 0$. Generate a block of bits e_k for $k = 0,1, \dots 15$ which is the same as a_i to a_{i+15}. If e_0 equals to 1 then proceed an XOR operation with the polynomial bits, otherwise select the next 16 bits from b_k by increasing i by 1 until $i = A - 1$.

- After i has been increased to $A - 1$, the parity bits p_i for $i = 0,1, \dots ,15$ are presented by e_k for $k = 0,1, \dots 15$.

- If the parity bits p_i are all zeros then the input block bits passes the CRC and block of bits b_k for $k = 0,1, \dots , A - 1$ denotes the DCI expected by UE. Otherwise the input block of bits does not pass the CRC.

2.4.3.9 PDCCH Blind Decoding

In this process the UE tries to decode the PDCCH on every possible PDCCH candidate until the payload passes the CRC is found. The UE always decode all candidates in the common search space first and then decode candidates in the UE-specific search space if the UE is configured with C-RNTI. The UE is configured with different modes to monitor different DCI formats with C-RNTI as described in Table 23.

Table 23 Downlink DCI formats monitored by UE in different search space for C-RNTI [26].

UE Mode	Monitored DCI formats in search space			Description
	Common	UE-Specific	UE-Specific	
1	1A	1A	1	Single antenna transmission
2	1A	1A	1	Transmit diversity
3	1A	1A	2A	Open-loop spatial multiplexing
4	1A	1A	2	Closed-loop spatial multiplexing
5	1A	1A	1D	Multi-user MMO
6	1A	1A	1B	Single layer codebook-based precoding
7	1A	1A	1	Single-layer transmission using DM-RS
8	1A	1A	2B	Dual-layer transmission using DM-RS
9	1A	1A	2C	Multi-layer transmission using DM-RS

In this thesis the UE is configured with mode 2 and the decoding process for a PDCCH candidate is performed in following steps.

1. Choose the type of search space the UE is decoding. The UE always decodes the common search space first and decode the UE-specific search space only when it is configured with C-RNTI.

2. Assume the DCI format used in the PDCCH and calculate the DCI length. As the UE has no information about the DCI format the PDCCH is using, every possible DCI format needs to be assumed for decoding. The UE calculates the length of the assumed DCI L_D

depending on the DCI format assumed, N_{RB}^{DL} and carrier indictor field. In this thesis, only DCI format 1A is considered and according to Section 2.3.3.1 the length of DCI format 1A is calculated according to Equation 2.4.18.

$$L_D = \begin{cases} \max\left(18 + \left\lceil log_2\left(\frac{N_{RB}^{UL}(N_{RB}^{UL}+1)}{2}\right)\right\rceil, 15 + \left\lceil log_2\left(\frac{N_{RB}^{DL}(N_{RB}^{DL}+1)}{2}\right)\right\rceil\right) \text{for case 1} \\ \max\left(21 + \left\lceil log_2\left(\frac{N_{RB}^{UL}(N_{RB}^{UL}+1)}{2}\right)\right\rceil, 18 + \left\lceil log_2\left(\frac{N_{RB}^{DL}(N_{RB}^{DL}+1)}{2}\right)\right\rceil\right) \text{for case 2} \\ \max\left(18 + \left\lceil log_2\left(\frac{N_{RB}^{UL}(N_{RB}^{UL}+1)}{2}\right)\right\rceil, 16 + \left\lceil log_2\left(\frac{N_{RB}^{DL}(N_{RB}^{DL}+1)}{2}\right)\right\rceil\right) \text{for case 3} \\ \max\left(21 + \left\lceil log_2\left(\frac{N_{RB}^{UL}(N_{RB}^{UL}+1)}{2}\right)\right\rceil, 19 + \left\lceil log_2\left(\frac{N_{RB}^{DL}(N_{RB}^{DL}+1)}{2}\right)\right\rceil\right) \text{for case 4} \end{cases} \qquad (2.4.18)$$

where case 1 is the common search space without carrier indicator, case 2 is the common search space with carrier indicator, case 3 is the UE-specific search space without carrier indicator and case 4 is UE-specific search space with carrier indicator.

3. Make assumption of the aggregation level L in the assumed search space A and calculate the number of PDCCH candidates $M^{(L)}$ within the search space according to Table 18.

4. Choose a PDCCH candidate m' where $m' = 0, \ldots, M^{(L)} - 1$ and calculate position of m' in CCEs according to Section 2.3.3.5. Collect all integers within the candidate to generate a block of integer a_i for $i = 0,1, \ldots, 72 \cdot L - 1$.

5. Calculate the number of redundancy sets N in the PDCCH according to Equation 2.4.19.

   $$N = \left\lfloor L \cdot \frac{72}{(L_D+16)\cdot 3} \right\rfloor \qquad (2.4.19)$$

   Summarize all redundancy sets and take average by N to generate a block of integers w_i for $i = 0,1, \ldots, (L_D + 16) \cdot 3 - 1$.

6. The block of integers w_i undergoes reverse bit collection to generate a block of integers $v_k^{(i)}$ for $i = 0, 1, 2$ according to Equation 2.4.20 to 2.4.22.

   $$v_k^{(0)} = w_k \quad \text{for } k = 0, \ldots, L_D + 15 \qquad (2.4.20)$$

   $$v_k^{(1)} = w_{k+K} \quad \text{for } k = 0, \ldots, L_D + 15 \qquad (2.4.21)$$

   $$v_k^{(2)} = w_{k+2\times K} \text{ for } k = 0, \ldots, L_D + 15 \qquad (2.4.22)$$

7. The block of integers $v_k^{(i)}$ undergoes deinterleaving with the method mentioned in Section 2.4.3.3 using block of integers as input data instead of block of quadruplets. The generated output is block of integers $d_k^{(i)}$ for $k = 0, \ldots, L_D + 15$ where $i = 0, 1, 2$.

8. The block of integers $d_k^{(i)}$ undergoes Viterbi Decoder to generate a block of bits c_k for $k = 0, \dots, L_D + 15$ with the decoder configured according to Section 2.4.3.7.

9. The block of bits c_k undergoes CRC according to Section 2.4.3.8 to determine whether c_k contains the DCI expected by the UE.

10. If the block of bits c_k passes the CRC then the DCI is denoted by a_k for $k = 0,1, \dots, L_D - 1$ then returns to Step 2 and assume another DCI format which is not considered in this thesis. Otherwise returns to Step 4 and choose another PDCCH candidate.

11. If all PDCCH candidates within in search space A has been decoded and failed to pass the CRC, returns to Step 3 and make another assumption of L.

12. If the common search space has been decoded and the UE is configured with C-RNTI then returns to Step 1 and choose UE-specific search space.

If the DCI is decoded successfully, the block of bits a_k for $k = 0,1, \dots, L_D - 1$ is generated in Step 10. Then the UE decodes all encoded control information with the DCI.

2.4.3.10 DCI Format 1A Decoding

The DCI is decoded to retrieve all encoded control information from DCI payload a_k. The decoding process is done in following steps.

- If the PDCCH is configured with carrier indicator, the first three bits of a_k indicates the component carrier in bit reverse order. If the carrier indicator presents the offset L_{CI} is set to 3, otherwise L_{CI} is set to 0.

- The DCI is in format 1A if $a_{L_O} = 1$ and in format 0 if $a_{L_O} = 0$.

- Calculate the number of bits used for resource block assignments L_B according to Equation 2.4.23.

$$L_B = \left\lceil \log_2\left(\frac{N_{RB}^{DL}(N_{RB}^{DL}+1)}{2}\right) \right\rceil \tag{2.4.23}$$

- If a_{L_O+1} equals to 0, block of bits $a_{L_{CI}+2}$ to $a_{L_{CI}+L_B+1}$ all equals to 1, block of bits $a_{L_{CI}+L_B+12}$ to a_{L_D-1} all equals to 0 and the UE is configured to decode with C-RNTI then the decoding process, goes to Case 1. Otherwise goes to Case 2

Case 1 indicates that the DCI is used for random access procedure. In this case the control information is decoded as:

- The value of preamble index PI is calculated according to Equation 2.4.24.

$$PI = \sum_{i=0}^{5} a_{L_{CI}+L_B+7-i} \cdot 2^i \tag{2.4.24}$$

- The value of PRACH mask index RAM is calculated according to Equation 2.4.25.

$$RAM = \sum_{i=0}^{3} a_{L_{CI}+L_B+11-i} \cdot 2^i \tag{2.4.25}$$

Case 2 indicates that the DCI is used for compact scheduling for PDSCH. In this case the control information is decoded as:

- If $a_{L_{CI}+1}$ is 0 then the VRB assignment is localized and the resource block assignment index RBA is calculated according to Equation 2.4.26.

  $$RBA = \sum_{i=0}^{L_B-1} a_{L_{CI}+L_B+1-i} \cdot 2^i \tag{2.4.26}$$

  If $a_{L_{CI}+1}$ is 0 then the VRB assignment is distributed. If $N_{RB}^{DL} < 50$ or the UE is configured with P-RNTI, SI-RNTI or RA-RNTI then RBS is derived same as localized. Otherwise the gap value N_{gap} and RBS are calculated according to Equation 2.4.27 and 2.4.28.

  $$N_{gap} = \begin{cases} N_{gap,1} \text{ for } a_{L_{CI}+2} = 0 \\ N_{gap,2} \text{ for } a_{L_{CI}+2} = 1 \end{cases} \tag{2.4.27}$$

  $$RBA = \sum_{i=0}^{L_B-2} a_{L_{CI}+L_B+1-i} \cdot 2^i \tag{2.4.28}$$

  The starting position $RB_S$and length RB_L of resource block assignment are calculated according to Equation 2.4.29 and 2.4.30.

  $$RB_L = \begin{cases} \lfloor RBA/N_{RB}^{DL} \rfloor + 1 & \text{for } \lfloor RBA/N_{RB}^{DL} \rfloor \le \lfloor N_{RB}^{DL}/2 \rfloor \\ N_{RB}^{DL} - \lfloor RBA/N_{RB}^{DL} \rfloor + 1 & \text{for } \lfloor RBA/N_{RB}^{DL} \rfloor > \lfloor N_{RB}^{DL}/2 \rfloor \end{cases} \tag{2.4.29}$$

  $$RB_S = \begin{cases} RBA \bmod N_{RB}^{DL} & \text{for } \lfloor RBA/N_{RB}^{DL} \rfloor \le \lfloor N_{RB}^{DL}/2 \rfloor \\ N_{RB}^{DL} - (RBA \bmod N_{RB}^{DL}) - 1 & \text{for } \lfloor RBA/N_{RB}^{DL} \rfloor > \lfloor N_{RB}^{DL}/2 \rfloor \end{cases} \tag{2.4.30}$$

- The modulation and coding scheme I_{macs} is calculated according to Equation 2.4.31.

$$I_{macs} = \sum_{i=0}^{4} a_{L_{CI}+L_B+6-i} \cdot 2^i \tag{2.4.31}$$

- The Hybrid-ARQ process number $HARQ_{pn}$ is calculated according to Equation 2.4.32.

$$HARQ_{pn} = \sum_{i=0}^{2} a_{L_{CI}+L_B+9-i} \cdot 2^i \tag{2.4.32}$$

- If UE is configured with P-RNTI, SI-RNTI or RA-RNTI, the VRB assignment is distributed and $N_{RB}^{DL} \ge 50$ then the gap value N_{gap} is calculated according to Equation 2.4.33.

$$N_{gap} = \begin{cases} N_{gap,1} \text{ for } a_{L_{CI}+L_B+10} = 0 \\ N_{gap,2} \text{ for } a_{L_{CI}+L_B+10} = 1 \end{cases} \tag{2.4.33}$$

Otherwise a new transmission is indicated by $a_{L_{CI}+L_B+10}$ equals to 1 and no new transmission by $a_{L_{CI}+L_B+10}$ equals to 0.

- The redundancy version RV is calculated according to Equation 2.4.34.

$$RV = \sum_{i=0}^{1} a_{L_{CI}+L_B+12-i} \cdot 2^i \tag{2.4.34}$$

- If UE is configured with P-RNTI, SI-RNTI or RA-RNTI then the column index N_{PRB}^{1A} is calculated according to Equation 2.4.35.

$$N_{PRB}^{1A} = \begin{cases} 2 \text{ for } a_{L_{CI}+L_B+14} = 0 \\ 3 \text{ for } a_{L_{CI}+L_B+14} = 1 \end{cases} \tag{2.4.35}$$

Otherwise the transmit power command TPC is calculated according to Equation 2.4.36.

$$TPC = \sum_{i=0}^{1} a_{L_{CI}+L_B+14-i} \cdot 2^i \tag{2.4.36}$$

- If the UE is configured with C-RNTI then the SRS trigger type is type 0 for $a_{L_O+L_B+15} = 0$ and type 1 for $a_{L_O+L_B+15} = 1$.

After the DCI decoding, the decoding process of the control region is finished.

3 Implementation

This section describes the process of encoder implementation in MATLAB, encoder verification, decoder implementation in MATLAB and decoder implementation on Xilinx XUPV5-LX110T evaluation platform.

3.1 Encoder Implementation on MATLAB

The encoder implementation process is described in this section with similar structures in Section 2.3. Implementation of general steps, PCFICH encoding, PHICH encoding, PDCCH encoding, mapping to resource grid and noise attachment are described in Section 3.1.1 to 3.1.5. The top level hierarchy of encoder implementation is illustrated in Figure 19.

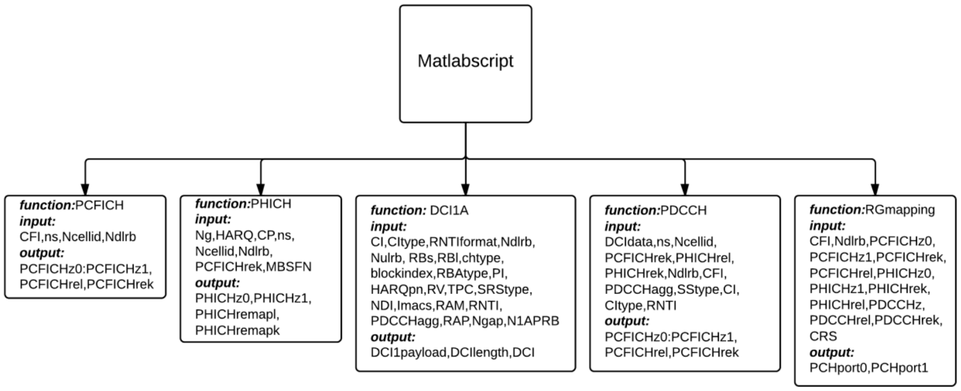


Figure 19 Top level hierarchy of encoder implementation.

3.1.1 Implementation of General Steps

General steps mentioned in Section 2.2 are implemented with several functions.

The scrambling sequence `scramblingseq` is generated by function

```
[scramblingseq]=scrambling(ns,Ncellid,flag,L)
```

where `ns` is the slot index n_s, `Ncellid` is the cell-specific identification N_{ID}^{cell}, `L` is the expected length of scrambling sequence, `flag` is a scrambling type selection with 0 indicates PCFICH or PHICH scrambling and 1 indicates PDCCH scrambling.

The QPSK modulated symbols `QPSK` are generated by function

```
[QPSK]=QPSKmodulation(dataseq,N)
```

where `dataseq` is the input bit sequence, `N` is the length of the input sequence.

The BPSK modulated symbols `BPSK` are generated by function

```
[BPSK]=BPSKmodulation(dataseq,N)
```

where `dataseq` is the input bit sequence, `N` is the length of the input sequence.

The layer mapping and precoding process of transmit diversity of two antenna ports is integrated into one function

```
[port0,port1]=layermapprecode2(dataseq,N)
```

where `port0` and `port1` denotes the block of symbols of antenna port 0 and antenna port 1, `dataseq` is the input block of symbols, `N` is the length of block of symbols.

The quadruplets for RE mapping are generated by function

```
[port0,port1]=quadruplet(dataseq0,dataseq1,N)
```

where `port0` and `port1` denotes the block of quadruplets of antenna port 0 and antenna port 1, `dataseq0` and `dataseq1` denotes the input block of symbols, `N` is the length of generated block of quadruplets.

3.1.2 PCFICH Encoder Implementation

The PCFICH encoding process is implemented by function

```
[PCFICHz0,PCFICHz1,PCFICHrel,PCFICHrek]
=PCFICH(CFI,ns,Ncellid,Ndlrb)
```

where `PCFICHz0` and `PCFICHz1` denotes the block of quadruplets mapped on antenna port 0 and antenna port 1, `PCFICHrel` and `PCFICHrek` denotes the position of REGs $(PCFICHrek, PCFICHrel)$ occupied by PCFICH. The input parameters defined in Table 24.

Table 24 Input parameter definitions of function `PCFICH`

Input parameter	Type	Definitions
`CFI`	Double	Number of OFDM symbols occupied by the control region
`ns`	Double	n_s
`Ncellid`	Double	N_{ID}^{cell}
`Ndlrb`	Double	N_{RB}^{DL}

The hierarchy of function `PCFICH` is illustrated in Figure 20.

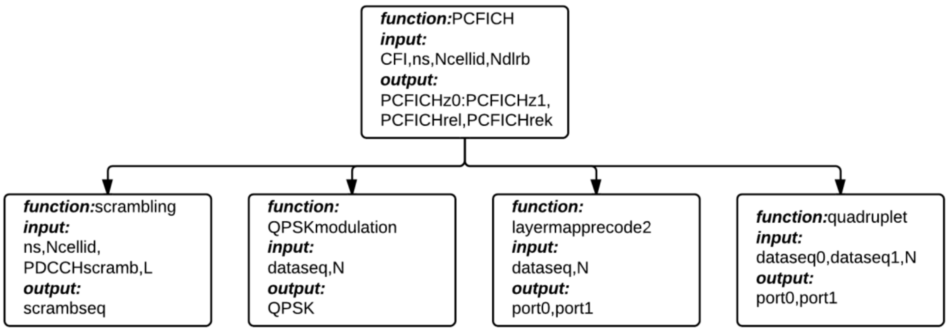


Figure 20 Hierarchy of function `PCFICH`.

3.1.3 PHICH Encoder Implementation

The PHICH encoding process is implemented by function

```
[PHICHz0,PHICHz1,PHICHremapl,PHICHremapk]
=PHICH(Ng,HARQ,CP,ns,Ncellid,Ndlrb,PCFICHrek,MBSFN)
```

where `PHICHz0` and `PHICHz1` denotes the block of quadruplets mapped on antenna port 0 and antenna port 1, `PHICHremapl` and `PHICHremapk` denotes the position of REGs $(PHICHrek, PHICHrel)$ occupied by PHICH. The input parameters are defined in Table 25.

Table 25 Input parameter definitions of function `PHICH`

Input parameter	Type	Definitions
`Ng`	Double	N_g
`HARQ`	N_{PHICH}^{group}-by-8 double matrix	Hybrid-ARQ acknowledgements with 1 indicating ACK, 0 indicating NACK and unused PHICH set to `NAN` which is "not a number" in MATLAB.
`CP`	Double	Cyclic prefix type with 0 indicating normal cyclic prefix and 1 indicating extended cyclic prefix
`ns`	Double	n_s
`Ncellid`	Double	N_{ID}^{cell}
`Ndlrb`	Double	N_{RB}^{DL}
`PCFICHrek`	1-by-4 double matrix	k indexes of REGs occupied by PCFICH
`MBSFN`	Double	Subframe type indication flag with 1 indicating MBSFN subframe and 0 indicating otherwise.

The hierarchy of function `PHICH` is illustrated in Figure 21.

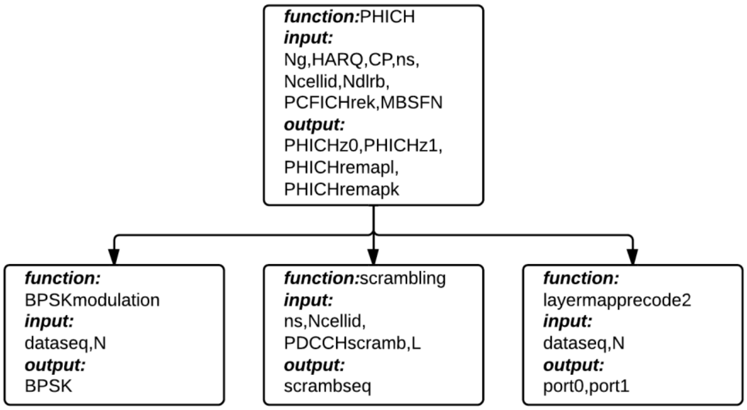


Figure 21 Hierarchy of function PHICH.

3.1.4 DCI Payload Generator Implementation

The DCI payload generator is described in several steps as DCI generation, CRC attachment, channel coding and rate matching. The hierarchy of DCI generation function is illustrated in Figure 22.

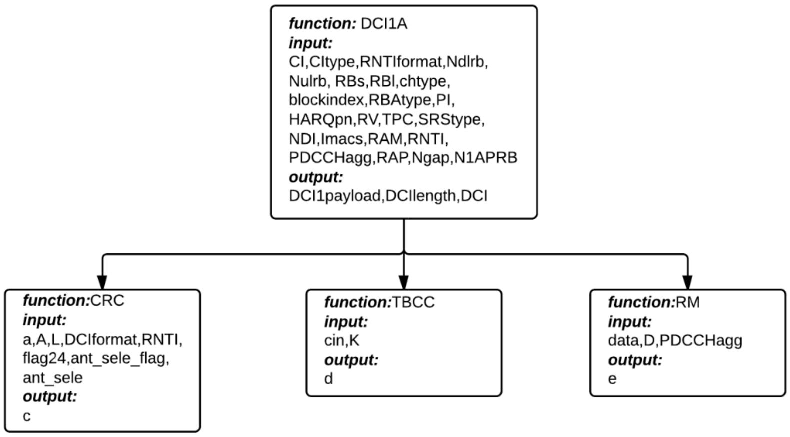


Figure 22 Hierarchy of DCI payload generator.

The DCI payload generation is implemented by function

```
[DCI1payload,DCIlength,DCI]
=DCI1A(CI,CItype,RNTIformat,Ndlrb,Nulrb,RBs,RBl,chtype,blockin
dex,RBAtype,PI,HARQpn,RV,TPC,SRStype,NDI,Imacs,RAM,RNTI,
PDCCHagg,RAP,Ngap,N1APRB)
```

where `DCI1Apayload` denotes the PDCCH payload after rate matching, `DCIlength` indicates the length of generated DCI and `DCI` denotes generated DCI in block of bits. The input parameters are defined in Table 26.

Table 26 Input parameter definitions of function `DCI1A`

Input parameter	Type	Definitions
`CI`	Double	Value of carrier indicator
`CItype`	Double	Cross carrier scheduling indication with 1 and 0 indicating the control region is specified with carrier indicator or not
`RNTIformat`	Char	RNTI format indication such as ‘C-RNTI’ for C-RNTI scrambling
`Ndlrb`	Double	N_{RB}^{DL}
`Nulrb`	Double	N_{RB}^{UL}
`RBs`	Double	Downlink assignment allocations starting position
`RBl`	Double	Length of downlink assignment allocations
`chtype`	Char	Channel indication with ‘PDCCH’ indicates PDCCH transmission and ‘EPDCCH’ indicates E-PDCCH
`blockindex`	Char	Transmit block indicator with ‘first’ indicating first transmit block and ‘sec’ indicating the second transmit block
`HARQpn`	Double	Hybrid-ARQ process number
`RV`	Double	Value of redundancy version
`TPC`	Double	Value of transmit power commands
`SRStype`	Double	SRS trigger type indication with 0 indicates SRS trigger type 0 and 1 indicates SRS trigger type 1
`NDI`	Double	Value of new data indicator
`Imacs`	Double	Value of modulation and coding scheme
`RNTI`	Char	RNTI value in hexadecimal
`PDCCHagg`	Double	PDCCH aggregation level
`RAP`	Double	Rand access pressure indication flag with 1 indicates random access pressure initialized and 0 indicates otherwise.
`PI`	Double	Preamble index value used for random access pressure.
`RAM`	Double	Value of PRACH mask index
`Ngap`	Double	Gap type indication for distributed allocation with 1 indicates $N_{gap,1}$ and 2 indicates $N_{gap,2}$
`N1APRB`	Double	Value of column index N_{PRB}^{1A}

3.1.4.1 CRC Attachment

The CRC attachment process is implemented by function

`c=CRC(a,A,L,DCIformat,RNTI,flag24,ant_sele_flag,ant_sele)`

where `c` is block of bits after CRC attachment. Input parameters are defined in Table 27.

Table 27 Input parameter definitions of function `CRC`

Input parameter	Type	Definitions
`a`	1-by-A double matrix	Input block of bits
`A`	Double	Length of input
`L`	Double	Length of CRC
`DCIformat`	Double	DCI format indication with 0 indicates DCI format 0 and 1 indicates otherwise
`RNTI`	Char	RNTI value in hexadecimal
`flag24`	Double	24-bit CRC selection flag with 0 indicates type A and 1 indicates type B.
`ant_sele_flag`	Double	UE transmit antenna selection flag with 0 indicates antenna is not selected and 1 indicates antenna is selected
`ant_sele`	Double	UE transmit antenna selection with 0 indicates UE port 0 and 1 indicates UE port 1

3.1.4.2 Channel Coding

The tail biting convolutional coding is implemented by function

`out=TBCC(cin,K)`

where `out` is a 3-by-K double matrix denotes the encoded result. `cin` is the input block of bits and `K` is the length of input.

3.1.4.3 Rate Matching

The rate matching process is implemented by function

`e=RM(data,D,PDCCHagg)`

where `e` is the block of bits denotes the encoded result. `data` is a 3-by-D double matrix denotes the input blocks of bits, `D` is the length of input and `PDCCHagg` is the PDCCH aggregation level.

3.1.5 PDCCH Encoder Implementation

PDCCH encoder implementation includes implementation of PDCCH multiplexing, scrambling, QPSK modulation, layer mapping, precoding and mapping to REs. The hierarchy of PDCCH encoder is illustrated in Figure 23.

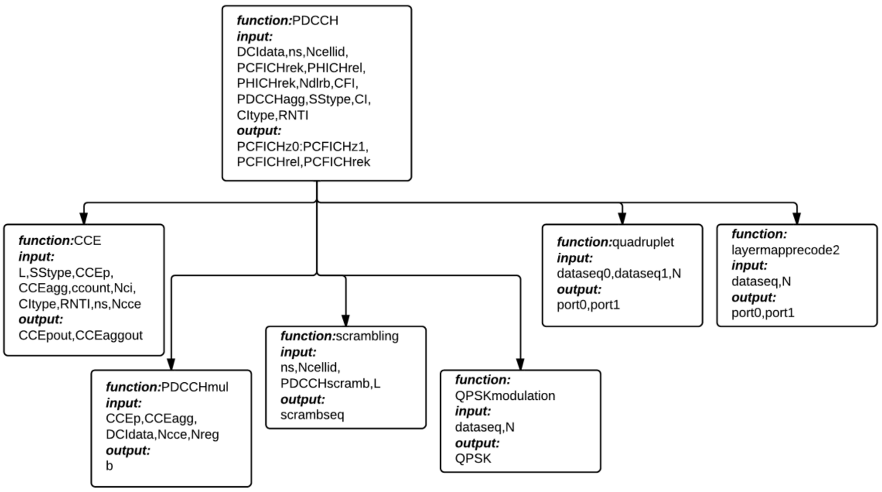


Figure 23 Hierarchy of PDCCH encoder.

The process is implemented by function

```
[PDCCHs,PDCCHrel,PDCCHrek]
=PDCCH(DCIdata,ns,Ncellid,PCFICHrek,PHICHrel,PHICHrek,Ndlrb,
CFI,PDCCHagg,SStype,CI,CItype,RNTI)
```

where `PDCCHs` is a 2-by-N matrix denoting two blocks of PDCCH quadruplets for 2 antenna ports with length N equals to the total number of REGs not occupied by PCFICH or PHICH, `PDCCHrel` and `PDCCHrek` are double vectors indicating the REGs used for mapping PDCCH. The input parameters are defined in Table 28.

Table 28 Input parameter definitions of function `PDCCH`

Input parameter	Type	Definitions
`DCIdata`	1-by-M cell with 1-by-L double matrix	Cell of M DCI payloads with length L.
`ns`	Double	n_s
`Ncellid`	Double	N_{ID}^{cell}
`PCFICHrek`	1-by-4 double matrix	k indexes of REGs occupied by PCFICH.
`PHICHrel`	1-by-3 × N_{PHICH}^{group} double matrix	l indexes of REGs occupied by PHICH.
`PHICHrek`	1-by-3 × N_{PHICH}^{group} double matrix	k indexes of REGs occupied by PHICH.
`Ndlrb`	Double	N_{RB}^{DL}
`CFI`	Double	Number of OFDM symbols available for the control region.
`PDCCHagg`	1-by-N double matrix	Aggregation levels of N PDCCHs.
`SStype`	1-by-N double matrix	Search space types of N PDCCHs.
`CI`	Double	Value of carrier indicator.
`CItype`	Double	Cross carrier scheduling indication with 1 and 0 indicating the control region is specified with carrier indicator or not
`RNTI`	Double	RNTI value in decimal

3.1.5.1 PDCCH Multiplexing

The PDCCH multiplexing is implemented with two parts, first calculated the CCE position for all PDCCHs then multiplexes all PDCCH together. The CCE position calculation is implemented by function

```
[CCEpout,CCEaggout]=CCE(L,SStype,CCEp,CCEagg,ccount,
                CI,CItype,RNTI,ns,Ncce)
```

where `CCEpout` is the CCE starting position of a PDCCH and `CCEaggout` is the PDCCH aggregation level. The input parameters are defined in Table 29.

Table 29 Input parameter definitions of function `CCE`

Input parameter	Type	Definitions
`L`	Double	PDCCH aggregation level
`SStype`	Char	Search space type indication with 'com' indicates common search space and 'spec' indicates UE-specific search space
`CCEp`	1-by-N double matrix	CCE positions occupied by N other PDCCHs. If no PDCCH is mapped then set to 0.
`CCEagg`	1-by-N double matrix	PDCCH aggregation level of N other mapped PDCCHs. If no PDCCH is mapped then set to 0.
`ccount`	Double	Number of mapped PDCCHs
`CI`	Double	Value of carrier indicator
`CItype`	Double	Cross carrier scheduling indication with 1 and 0 indicating the control region is specified with carrier indicator or not
`RNTI`	Double	RNTI value in decimal
`ns`	Double	n_s
`Ncce`	Double	Total number of CCEs available for PDDCH

All PDCCHs are multiplexed together to form a payload by function

```
b=FDCCHmul(CCEp,CCEagg,DCIdata,Ncce,Nreg)
```

where `b` is the payload after multiplexing. The input parameters are defined in Table 30.

Table 30 Input parameter definitions of function `FDCCHmul`

Input parameter	Type	Definitions
`CCEp`	1-by-N double matrix	CCE starting position of N mapped PDCCHs
`CCEagg`	1-by-N double matrix	Aggregation level of N mapped PDCCHs
`DCIdata`	1-by-M cell with 1-by-L double matrix	Cell of M DCI payloads with length L.
`Ncce`	Double	Total number of CCEs
`Nreg`	Double	Total number of REGs available for PDCCH.

3.1.6 Mapping to Resource Grid

The quadruplets of PCFICH, PHICHs and PDCCHs are mapped to resource grid by function

```
[PCHport0,PCHport1]
=RGmapping(CFI,Ndlrb,PCFICHz0,PCFICHz1,PCFICHrek,PCFICHrel,
PHICHz0,PHICHz1,PHICHrek,PHICHrel,PDCCHz,PDCCHrel,PDCCHrek,CRS)
```

where `PCHport0` and `PCHport1` are the block of symbols for the control region on antenna port 0 and antenna port 1with column index denotes the OFDM symbol index and row index denotes the subcarrier index. The input parameters are defined in Table 31.

Table 31 Input parameter definitions of function `REmapping`

Input parameter	Type	Definitions
`CFI`	Double	Number of OFDM symbols available for the control region.
`Ndlrb`	Double	N_{RB}^{DL}
`PCFICHz0`	1-by-4 cell	Block of quadruplets of PCFICH on antenna port 0
`PCFICHz1`	1-by-4 cell	Block of quadruplets of PCFICH on antenna port 1
`PCFICHrek`	1-by-4 double matrix	k indexes of REGs occupied by PCFICH.
`PCFICHrel`	1-by-4 double matrix	k indexes of REGs occupied by PCFICH.
`PHICHz0`	1-by-3 $\times N_{PHICH}^{group}$ cell	Block of quadruplets of PHICH on antenna port 0
`PHICHz1`	1-by-3 $\times N_{PHICH}^{group}$ cell	Block of quadruplets of PHICH on antenna port 1
`PHICHrek`	1-by-3 $\times N_{PHICH}^{group}$ double matrix	k indexes of REGs occupied by PHICH.
`PHICHrel`	1-by-3 $\times N_{PHICH}^{group}$ double matrix	l indexes of REGs occupied by PHICH.
`PDCCHz`	2-by-N cell	Block of quadruplets of PDCCH on antenna port 0 and 1 with N denoting the number of REGs not used by PCFICH or PHICH in the control region
`PDCCHrel`	1-by-N double matrix	l index of REGs occupied by PDCCH with N denoting the number of REGs not used by PCFICH or PHICH in the control region
`PDCCHrek`	1-by-N double matrix	k index of REGs occupied by PDCCH with N denoting the number of REGs not used by PCFICH or PHICH in the control region
`CRS`	2-by-M double matrix	Block of symbols with length M denoting cell-specific reference signals on antenna port 0 and 1

3.1.7 Noise Attachment and Save to Files

The generated resource grid is added by white Gaussian noise by function

```
PCHportn=awgn(PCHport,noise)
```

where `PCHportn` is the resource grid for an antenna port with noise. `PCHport` is the original resource grid for an antenna port and `noise` is the level of Gaussian noise in decimal.

The resource grid for both antenna ports are saved to a file by function

```
dlmwrite('PCHportn.txt',PCHport0n)
dlmwrite('PCHportn.txt',PCHport1n,'-append')
```

where `PCHport0n` is the resource grid for antenna port 0, `PCHport1n` is the resource grid for antenna port 1 and `PCHportn.txt` is the file name.

3.2 Encoder Verification

The encoded results are verified with functions available from the LTE system toolbox in MATLAB 2013b. The LTE system toolbox provides encoding and decoding functions with LTE standard which are used to verify the encoder implementation in this thesis.

3.2.1 Verification of General Steps

The scrambling sequence for PCFICH and PHICH are verified by comparing with function

```
PCFICHseq=ltePCFICHPRBS(enb,n)
PHICHseq=ltePHICHPRBS(enb,n)
```

where `PCFICHseq` and `PHICHseq` are the scrambling sequence for PCFICH and PHICH. `enb` is the cell-wide settings including N_{ID}^{cell} and subframe number and `n` is the length of scrambling sequence.

The modulation functions are verified by comparing with function

```
out=lteSymbolModulate(in,mod)
```

where `out` is modulated symbols in column vector, `in` is the input bits in column vector, `mod` is modulation scheme which is set to `'QPSK'`.

The layer mapping and precoding function is verified by comparing with function

```
LMout=lteLayerMap(in,nu,txscheme)
Precodeout=lteDLPrecode(LMout,p,txscheme)
```

where `LMout` is a N-by-2 matrix denotes the block of symbols with length N for 2 layers and `Precodeout` is an M-by-2 matrix denotes the block of symbols with length M for 2 antenna ports. `in` is the input block of symbols in column vector. `nu` and `p` is the number of layers and the number of antenna ports which both set to 2. `txscheme` is the transmission scheme which is set to `'TxDiversity'`.

3.2.2 PCFICH Encoder Verification

The generated quadruplets for PCFICH are verified by comparing with function

```
sym=ltePCIFCH(enb,cw)
```

where `sym` is a matrix of complex modulation symbols. `enb` is the cell-wide settings structure including N_{ID}^{cell}, number of antenna ports, and subframe number. `cw` is the encoded CFI generated by

```
cw=lteCFI(enb)
```

The position of REGs occupied by PCFICH is verified by comparing with function

```
ind=ltePCFICHIndices(enb)
```

where `ind` is the a matrix of resource element indices.

3.2.3 PHICH Encoder Verification

The generated quadruplets are verified by comparing with function

```
[sym,info]=ltePHICH(enb,hiset)
```

where `sym` is a matrix of complex modulation symbols. `enb` is the cell-wide settings and `hiset` is a matrix with three columns. Each row of `hiset` defines a PHICH in terms of `[nGroup, nSeq, hi]` where `nGroup` is PHICH group index, `nSeq` is the orthogonal index and `hi` is 0 or 1 indicating ACK or NACK. Info presents some control information of output symbols.

The position REGs occupied by PHICH is verified by comparing with function

```
ind=ltePHICHIndices(enb)
```

where `ind` is the a matrix of resource element indices.

3.2.4 PDCCH Encoder Verification

The PDCCH encoder is verified in several steps as verification of DCI generation, verification of CRC attachment, verification of channel coding, verification of rate matching, verification of PDCCH multiplexing and verification of PDCCH signaling.

3.2.4.1 Verification of DCI Generation

The generated DCI message is verified by comparing with function

```
[dcistr,dcibits]=lteDCI(enb,istr)
```

where `dcistr` is generated DCI described by a string and `dcibits` is generated DCI presented in block of bits. `enb` is the cell-wide setting, `istr` is the input structure. The PDCCH payload `DCI1Apayload` is verified by comparing with function

```
Cw=lteDCIEncode(ue,dcibits)
```

where `cw` is output column vector, `ue` is parameter structure including PDCCH format and RNTI value, `dcibits` in the generated DCI bits in a column vector.

3.2.4.2 Verification of CRC Attachment

The result of CRC attachment is verified by comparing with function

```
blksrc=lteCRCEncode(blk,poly)
```

where `blksrc` is the block of bits after CRC attachment, `blk` is the input block of bits and `poly` is CRC polynomial selection.

3.2.4.3 Verification of Channel Coding

The result channel coding is verified by comparing with function

```
output=lteConvolutionalEncode(input)
```

where `output` is a column vector with length equal to 3 times of the input length and `input` is the input block of bits represented in a column vector.

3.2.4.4 Verification of Rate Matching

The result of rate matching is verified by comparing with function

```
out=lteRateMatchConvolutional(in,outlen)
```

where `out` is the output block of bits in column vector, `in` is the input block of bits in column vector and `outlen` is expected output length derived from PDCCH format.

3.2.4.5 Verification of PDCCH Multiplexing

The calculated CCE positions for PDCCHs are verified by function

```
Ind=ltePDCCHSpace(enb,ue)
```

where `Ind` (0, 2, 4, 6)-by2 matrix denotes the starting position and ending position of all PDCCH candidate within a search space. `enb` is cell-wide setting and `ue` is UE-specific cell-wide settings.

3.2.4.6 Verification of PDCCH Signaling

The PDCCH signaling result can be verified by comparing with function

```
[sym,info]=ltePDCCH(enb,cw)
```

where sym is a N-by-2 matrix denoting two blocks of PDCCH quadruplets for 2 antenna ports with length N equals to the total number of REs not occupied by PCFICH, PHICH or Cell-specific reference signals and info is some basic information. Enb is cell-wide settings and cw is a specified PDCCH payload.

3.3 Decoder Implementation on MATLAB

The decoder implementation process is described in this section with similar structures in Section 2.4. Implementation PCFICH decoding, PHICH decoding and PDCCH decoding are described in Section 3.3.1 to Section 3.3.3. The decoded results are verified by comparing with the encoded control information.

3.3.1 PCFICH Decoder Implementation

The PCIFCH decoding process is implemented by function

```
[CFIdecode,PCFICHdek]=PCFICHdecoder(Ncellid,Ndlrb,PCHport0,
                PCHport1,ns,CP,MBSFN)
```

where CFIdecode is the decoded number of OFDM symbols occupied by control region and PCFICHdek is the decoded k indexes of REGs occupied by PCFICH. The input parameters are defined in Table 32.

Table 32 Input parameter definitions of function PCFICHdecoder

Input parameter	Type	Definitions
Ncellid	Double	N_{ID}^{cell}
Ndlrb	Double	N_{RB}^{DL}
PCHport0	N-by-M double matrix	The block of symbols for control region on antenna port 0 with N OFDM symbols and M subcarriers.
PCHport1	N-by-M double matrix	The block of symbols for control region on antenna port 1 with N OFDM symbols and M subcarriers.
ns	Double	n_s
CP	Double	Cyclic prefix type with 0 indicating normal cyclic prefix and 1 indicating extended cyclic prefix
MBSFN	Double	Subframe type indication flag with 1 indicating MBSFN subframe and 0 indicating otherwise.

3.3.2 PHICH Decoder Implementation

The PHICH decoding process is implemented by function

```
[ACK,PHICHdel,PHICHdek]=PHICHdecoder(Ng,PCHport0,PCHport1,CP,
        ns,Ncellid,Ndlrb,PCFICHrek,MBSFN,PHICHgin,PHICHo)
```

where `ACK` is the Hybrid-ARQ acknowledgement expected by the UE with 0 indicates NACK and 1 indicates ACK, `PHICHdel` and `PHICHdek` are the decoded k and l indexes of REGs occupied by PHICH. The input parameters are defined in Table 33.

Table 33 Input parameter definitions of function `PHICHdecoder`

Input parameter	Type	Definitions
`Ng`	Double	Scaling factor from higher layer
`PCHport0`	N-by-M double matrix	The block of symbols for control region on antenna port 0 with N OFDM symbols and M subcarriers.
`PCHport1`	N-by-M double matrix	The block of symbols for control region on antenna port 1 with N OFDM symbols and M subcarriers.
`CP`	Double	Cyclic prefix type with 0 indicating normal cyclic prefix and 1 indicating extended cyclic prefix
`ns`	Double	n_s
`Ncellid`	Double	N_{ID}^{cell}
`Ndlrb`	Double	N_{RB}^{DL}
`PCFICHrek`	1-by-4 double matrix	k indexes of REGs occupied by PCFICH
`MBSFN`	Double	Subframe type indication flag with 1 indicating MBSFN subframe and 0 indicating otherwise.
`PHICHgin`	Double	Expected PHICH group index
`PHICHo`	Double	Expected PHICH orthogonal index

3.3.3 PDCCH Decoder Implementation

The PDCCH decoding process is implemented by function

```
[DeCI,DeDCItype,DePI,DeRAM,DeRBAtype,DeRBl,DeRBs,DeNgap,
DeImacs,DeHARQpn,DeND,DeRV,DeN1aPRB,DeTPC,DeSRStype]=
PDCCHdecoder(UE_CItype,UE_CI,UE_RNTItype,Ndlrb,Nulrb,DeCFI,PHI
CHdel,PHICHdek,PCFICHdek,PCHport0,PCHport1,Ncellid,ns,UE_RNTI)
```

The output parameters are defined in Table 34 and input parameters are defined in Table 35. The default values of outputs are `NaN` for data type double and `'none'` for data type char.

Table 34 Output parameter definitions of function `PDCCHdecoder`

Output parameter	Type	Definitions
`DeCI`	Double	Decoded number of OFDM symbols
`DeDCItype`	Char	Decoded DCI format
`DePI`	Double	Decoded value of preamble index used for random access pressure.
`DeRAM`	Double	Decoded value of PRACH mask index
`DeRBAtype`	Char	Decoded downlink assignment allocation type
`DeRBl`	Double	Decoded starting position of downlink assignment allocations
`DeRBs`	Double	Decoded length of downlink assignment allocations
`DeNgap`	Double	Decoded gap indication for distributed allocation with 1 indicates $N_{gap,1}$ and 2 indicates $N_{gap,2}$
`DeImacs`	Double	Decoded value of modulation and coding scheme
`DeHARQpn`	Double	Decoded value of hybrid-ARQ process number
`DeND`	Double	Decoded value of new data indicator
`DeRV`	Double	Decoded value of redundancy version
`DeN1aPRB`	Double	Decoded value of column index N_{PRB}^{1A}
`DeTPC`	Double	Decoded value of transmit power commands
`DeSRStype`	Char	Decoded SRS trigger type

Table 35 Input parameter definitions of function `PDCCHdecoder`

Input parameter	Type	Definitions
`UE_CItype`	Char	UE carrier indicator configuration indication
`UE_CI`	Double	UE configured carrier indicator value
`UE_RNTItype`	Char	UE configured RNTI type
`Ndlrb`	Double	N_{RB}^{DL}
`Nulrb`	Double	N_{RB}^{UL}
`DeCFI`	Double	Decoded number of OFDM symbols occupied by control region
`PHICHdel`	1-by-3 × N_{PHICH}^{group} double matrix	l indexes of REGs occupied by PHICH
`PHICHdek`	1-by-3 × N_{PHICH}^{group} double matrix	k indexes of REGs occupied by PHICH
`PCFICHdek`	1-by-4 double matrix	k indexes of REGs occupied by PCFICH
`PCHport0`	N-by-M double matrix	The block of symbols for control region on antenna port 0 with N OFDM symbols and M subcarriers.
`PCHport1`	N-by-M double matrix	The block of symbols for control region on antenna port 1 with N OFDM symbols and M subcarriers.
`Ncellid`	Double	N_{ID}^{cell}
`ns`	Double	n_s
`UE_RNTI`	Char	UE RNTI value in hexadecimal

The deinterleaving process used during PDCCH decoding is implemented by function

```
[out]=deinterleaving(va)
```

where `out` is the output block of quadruplets or bits and `va` is the input block of quadruplets or bits.

The CRC calculation process used during PDCCH decoding is implemented by function

```
[reminder,out]=deCRC(input,RNTI,UE_ant_flag,UE_ant_port)
```

where `reminder` is the output parity bits after CRC and `out` is the block of bits before CRC attachment. The input parameters are defined in Table 36

Table 36 Input parameter definitions of function `deCRC`

Input parameter	Type	Definitions
`input`	1-by-N double matrix	Input block of bits with length N.
`RNTI`	Char	UE configured RNTI value in hexadecimal
`UE_ant_flag`	Double	UE transmit antenna selection indicator with 0 indicates antenna is not selected and 1 indicates antenna is selected
`UE_ant_port`	Double	UE transmit antenna port indicator with 0 indicates UE port 0 and 1 indicates UE port 1

3.4 Decoder Implementation on FPGA

The decoder is implemented on Xilinx XUPV5-LX110T evaluation platform by using an embedded system design. The software requirements of the implementation are Xilinx Platform Studio and Xilinx Core Generator. The overall structure of the design is illustrated in Figure 24.

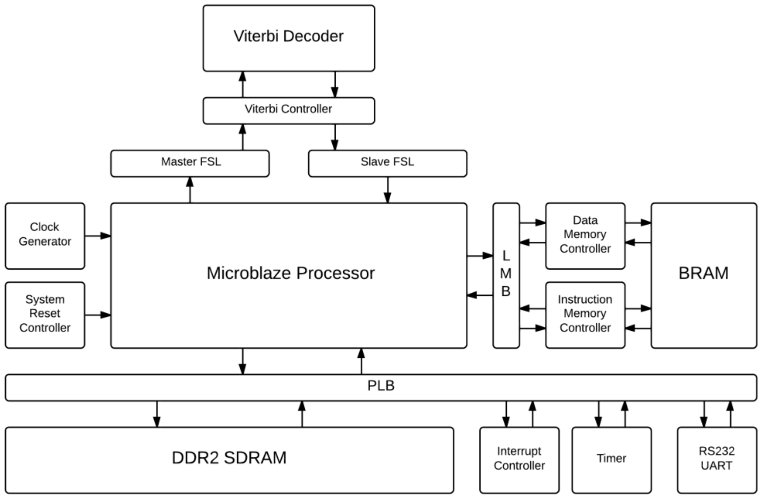


Figure 24 Overall structure of decoder implementation on FPGA.

The design is first implemented on a Nexys 3 board as an AXI based system and encountered the problem of lack of hardware resource. The final design is implemented on Xilinx board as a PLB based system with a 125MHz system clock. All components in the decoder use a clock with the same speed as the system clock. As illustrated from Figure 19, the input symbols are transmitted from a PC through the UART and stored in DSRAM. The Microblaze processor runs the decoding algorithms according to instructions and parameters stored in the BRAM. The Viterbi decoder is implemented as hardware accelerated core and communicate with the

CPU through two Fast Simplex Links (FSLs). The detailed implementation of each part of the design is described in this Section 3.4.1 to Section 3.4.4.

3.4.1 UART Implementation

The UART communication is based on Xilinx IP core: xps_uartlite. The UART controller is interrupt based with an interrupt handler verifies the input character and convert the data format to float complex. The xps_uartlite uses following hardware configuration: baud rate equal to 19200 with 8 bits in a serial frame and no parity. The software setup for the UART controller is achieved with following steps.

- Initialize the UART controller with UART device ID.
- Initialize the interrupt controller with interrupt unit device ID.
- Connect the interrupt controller with the interrupt handler.
- Connect the UART controller with the interrupt handler.
- Start and enable interrupt controller.
- Initialize and enable the interrupt exception handler.
- Set the UART receive handler when interrupted.
- Enable the UART controller in interrupt mode.
- Enable Microblaze processor interrupt globally.

The receive handler of UART controller is set to verify every received character and combine several characters into a complex-valued float number indicating one modulated symbol. The combined complex-valued float number is saved in the SDRAM after successfully generated. This process adds complexity to the UART interrupt handler and the transmission time is increased because the transmission is interrupted during most of the time. In the file saved by MATLAB, the number of characters used for a symbol is depending on the value. Thus the handler counts the number of complex-valued floats generated instead of the number of bytes the UART has received. After the expected number of symbols is received, the UART transmission is aborted. All outputs of the decoder are also transmitted through UART by Xilinx simplified version of

function `printf` which is `xil_printf`.

3.4.2 Timer Implementation

A timer is used to verify the decoding speed of the design. The timer is based on Xilinx IP core: xps_timer. All hardware configurations are set as default. The timer is initialized or reset by configuring the control status registers in software. To start and stop the timer, use function `XTmrCtr_Enable(int device_base_address, int device_ID)` and `XTmrCtr_Disable(int device_base_address, int device_ID)` where `device_base_address` and `device_ID` are depending on hardware configuration. The counted number of clock cycles is read from the timer counter register.

3.4.3 Viterbi Decoder Implementation

The Viterbi decoder is implemented as hardware accelerated core based on Xilinx IP core: Viterbi Decoder v7.0. This is the only part of the decoder algorithm implemented in hardware and software combined instead of software only. The reason to use this complex approach is that the software only approach takes roughly three seconds per decoding set. This does not match the basic timing requirements of the decoding process. The Viterbi Decoder v7.0 is generated by Xilinx Core generation with following constrains:

- Viterbi type set to standard Viterbi without reduced latency.
- Constraint length set to 7 and traceback length set to 42.
- Viterbi architecture set to parallel.
- Delete output ports used for the best states.
- Input data is soft coded with the format set to offset binary.
- Output rate set to 3 with three convolution channels set to 133, 171 and 165.
- Delete output ports used for BER symbol count.
- Add input control pins CE and SCLR for clock enable and reset

The reason of using parallel structure instead of serial structure is that the serial mode is approximately 10 times slower than the parallel mode and the serial mode control signals are useless as they cannot control the decoder as Xilinx described in the datasheet. In parallel mode, the decoder reads one input data every one clock cycle. As FSL takes two clock cycles to transfer one word and it takes more clock cycles to form one FSL word by software, the CE pin is added to control the Viterbi Decoder v7.0. As the Viterbi decoder might be used more than one time during the decoding process, the SCLR pin is added to reset the decoder before every input data set. The output bits from Viterbi Decoder v7.0 starts at precisely 192 working clock cycles after the first input is read. As there is no output enable pin, the controller needs to count the number of working clock cycles of the decoder to find the output bits. The decoder produces one output bit every one clock cycle. The reduced latency option is not used to stabilize the latency.

The FSLs are configured with the system clock and connected to Microblaze FSL streaming ports. The master FSL uses only 16 buffers as Viterbi decoder input buffer because the master transmission from Microblaze to Viterbi decoder controlled by software is much slower than the Viterbi controller processing speed. The slave FSL uses 60 buffers as Viterbi decoder output buffer because the slave transmission from Viterbi decoder to Microblaze is much faster than Microblaze reading speed. The expected size of a DCI format 1A with CRC attachment is smaller than 60 thus the output buffer is sufficient in size.

The structure of the Viterbi decoder is illustrated in Figure 25.

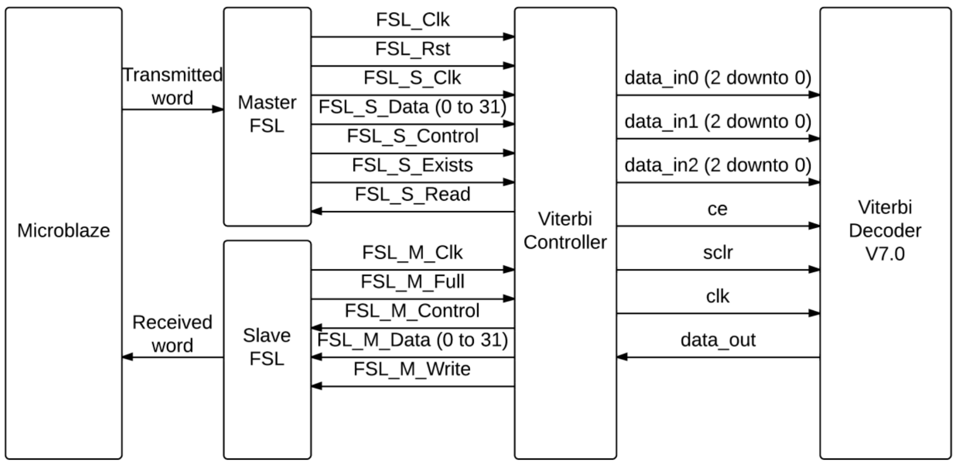


Figure 25 Structure of the Viterbi decoder

The Viterbi control is designed as a state machine that uses the same clock as the FSL. The state machine is illustrated in Figure 26.

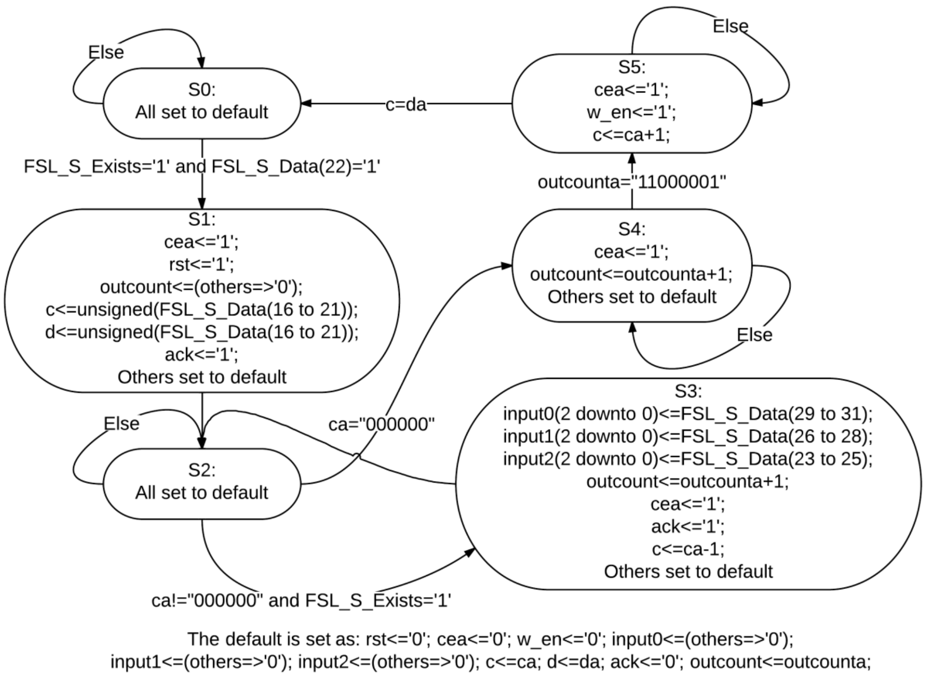

Else
S0:
All set to default
c=da
Else
S5:
cea<='1';
w_en<='1';
c<=ca+1;
FSL_S_Exists='1' and FSL_S_Data(22)='1'
outcounta="11000001"
S1:
cea<='1';
rst<='1';
outcount<=(others=>'0');
c<=unsigned(FSL_S_Data(16 to 21));
d<=unsigned(FSL_S_Data(16 to 21));
ack<='1';
Others set to default
S4:
cea<='1';
outcount<=outcounta+1;
Others set to default
Else
S3:
input0(2 downto 0)<=FSL_S_Data(29 to 31);
input1(2 downto 0)<=FSL_S_Data(26 to 28);
input2(2 downto 0)<=FSL_S_Data(23 to 25);
outcount<=outcounta+1;
cea<='1';
ack<='1';
c<=ca-1;
Others set to default
Else
ca="000000"
S2:
All set to default
ca!="000000" and FSL_S_Exists='1'
The default is set as: rst<='0'; cea<='0'; w_en<='0'; input0<=(others=>'0');
input1<=(others=>'0'); input2<=(others=>'0'); c<=ca; d<=da; ack<='0'; outcount<=outcounta;

Figure 26 Illustration of Viterbi controller state machine

As illustrated in Figure 26, the state machine starts from S0. If there is a new word in the FSL input buffer and the 23th bit of the input word is high then the state machine goes to S1. Otherwise the state machine stays at S0. The 23th bit of the input word is set high by software to indicate a new set of input data. At S1, the Viterbi Decoder v7.0 is reset by setting clock enable and reset signal high. All counters used by Viterbi controller are also reset and an input read acknowledgement is sent to slave FSL. After S1 the state machine goes to S2. If the expected input data set is not finished and there is a new word in the buffer, then the state machine goes to S3. If the expected data set is finished then the state machine goes to S4. If the expected data set is not finished and there is no new input word, the state machine stays at S2. At S3, the input bits of the Viterbi decoder are read from the input word, the read acknowledgement is set high and clock enable is set high. The input read counter is decreased by 1 and the decoder working clock cycle counter is increased by 1. After S3, the state machine goes back to S2. At S4, all input data have been read by the decoder and the decoding process continues by set clock enable pin high and the clock cycle counter is increased by 1 every clock cycle. Note that during the S4 the decoder is reading dummy bits as input. After the working clock cycle count reaches the expected value which is 193, the state machine goes to S5. At S5 the output enable is set high and output counter is increased by 1 every clock cycle. When the output length matches the size of input data set, the state machine goes to S0. The output bits from the Viterbi decoder after S5 are ignored by controller as they are decoded results of dummy bits read at S4.

At software side, the Viterbi decoding process is called by function

```
void viterbidecoder(int*b1,int*b2,int*b3,int inputlength)
```

where `b1`, `b2`, `b3` are the input block of integers and `inputlength` indicates the length of the input. The function communicates with FSLs with function `putfsl` and `getfsl` length with the first transmitted word indicates the input length and a new data set.

3.4.4 Decoder Algorithms Implementation

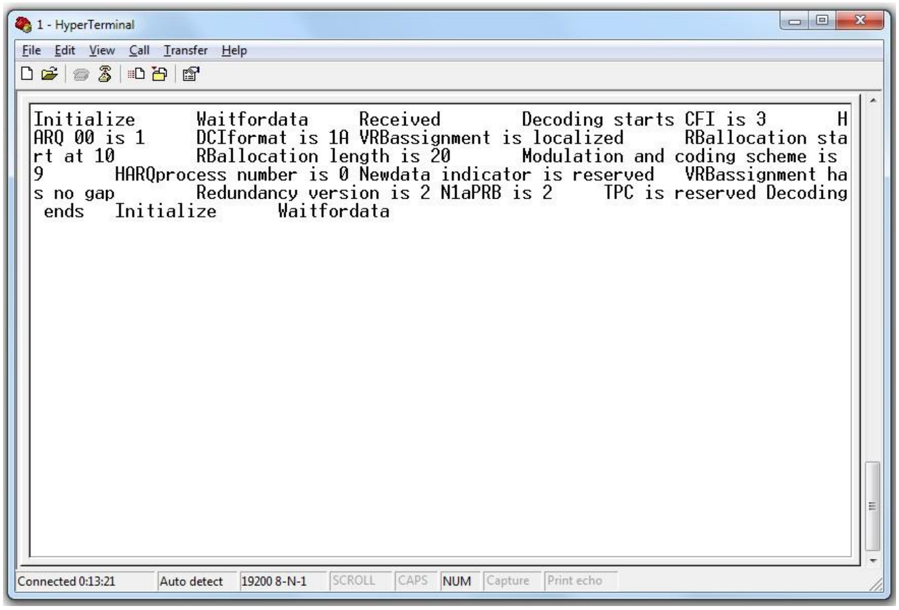
PCFICH decoding, PHICH decoding and PDCCH decoding are implemented by function

```
int PCFICHdecoder(void)

int PHICHdecoder(void)

int PDCCHdecoder(void)
```

All input parameters are defined in the header file and output parameters defined as global variables. Most variables used by these functions are dynamically allocated in the heap. The heap memory is allocated in the SDRAM with approximately 2MB in size. In order to save heap memory and reduce computation time, the position indexes of symbols are used instead of symbols in deinterleaving and reverse cyclic shift. All other algorithms are implemented similar to MATLAB implementation with C programming language. The algorithms are optimized to use less hardware resources.

4 Result and Discussion

This section presents the results and discussion of the implementation. In the MATLAB implementation, the encoded control information is decoded successfully by comparing encoder input parameters and decoder output variables. The implementation on Xilinx XUPV5-LX110T evaluation platform is verified by reading outputs on HyperTerminal. An example of FPGA decoded results is illustrated in Figure 27.



```
Initialize      Waitfordata     Received        Decoding starts CFI is 3       H
ARQ 00 is 1     DCIformat is 1A VRBassignment is localized      RBallocation sta
rt at 10        RBallocation length is 20       Modulation and coding scheme is
9       HARQprocess number is 0 Newdata indicator is reserved   VRBassignment ha
s no gap        Redundancy version is 2 N1aPRB is 2     TPC is reserved Decoding
 ends   Initialize      Waitfordata
```

Figure 27 Printed results on HyperTerminal.

As indictable from Figure 27 that `Initialize` indicates the initialization process of the timer and UART controller. `Waitfordata` indicates the UART is waiting for or receiving input characters from the PC. `Received` indicates expected number of symbols are received and stored in the SDRAM ready for decoding. `Decoding start` indicates the decoding process starts and all decoded control information are printed following. `Decoding over` indicates the decoding process ends and all decoded control information on printed on HyperTerminal. After that the decoder is restarted and waiting for new transmission as indicated by printing `Waitfordata` again.

The implementation synthesis summary is shown in Table 36 and design utilities are shown in Table 37.

Table 36 Synthesis summary of decoder implementation on XUPV5-LX110T evaluation platform.

Unit	Flip Flops Used	LUTs Used	BRAMS Used
System total	8366	8204	40
Clock generator	4	3	-
UART	147	127	-
Master FSL	55	114	1
Slave FSL	50	102	1
Viterbi decoder	24	57	-
Microblaze	1942	1624	-
Interrupt controller	132	92	-
System reset unit	69	53	-
Debug module	126	123	-
Timer	358	291	-
SDRAM	540	295	5
Instruction memory controller	3	6	-
Data memory controller	3	6	-
Local memory	-	-	32
PLB	156	409	-

Table 37 Design utilities of decoder implementation on XUPV5-LX110T evaluation platform.

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	7003	69120	10%
Number of Slice LUTs	7696 (7460 logic)	69120	11% (10% logic)
Number of route-thrus	263	-	-
Number of occupied Slices	4044	17280	23%
Number of LUT Flip Flop pairs used	10710	-	-
Number of BlockRAM	44	148	29%

Besides the correct decoding results, the noise isolation and performance are also considered in this section.

4.1 Noise isolation

The encoded symbols are multiplex with white Gaussian noise as mentioned in Section 3.1.6. The decoding noise isolation is measured by running the encoder and decoder 50 times. The decoding success rate percentage is calculated by multiplying the successful decoding times by two. The decoding success rate with different noise level is shown in Table 38.

Table 38 Decoding success rate with different noise level

	Decoding success rate (%)			
Signal to noise ratio(dB)	PDCCH (no redundancy sets)	PHICH		PCFICH
		NACK-ACK	ACK-NACK	
15	100	100	100	100
10	94	100	100	100
5	76	100	100	100
0	50	100	98	100
-5	6	100	96	96
-10	0	98	90	76
-15	0	92	80	44

As indictable from Table 36, the PHICH decoder has the best decoding success rate under noise. The NACK-ACK error rate is lower than ACK-NACK error rate as expected. This satisfies the error selection requirement mentioned in Section 2.3.2. The PCFICH decoder also has good noise isolation. The PDCCH decoder cannot handle much noise comparing with PCFICH decoder and PHICH decoder. The PDCCH decoding process is barely successful when the signal to noise ratio is higher than 0dB. This can be improved by adding more redundancy sets during rate matching process which means higher PDCCH aggregation level. Over all the noise isolation of the decoder is satisfying. Methods to improve the noise isolation are as follows:

- The overall decoding success rate is mainly limited by the decoder algorithm design which can be improved to have higher accuracy. For example, using soft decision demodulation for PCFICH decoding and PHICH decoding. This may also add complexity to the PHICH orthogonal decoding process and increase processing time.

- Use higher PDCCH aggregation level which results in more redundancy data sets. Decode each set to find the expected PDCCH when perform PDCCH blind decoding instead of take the average of all data sets. This may reduce the complexity of the PDCCH decoding algorithm but may increase the processing time greatly.

4.2 Performance

The performance of the decoder is measured by the timer. The timer measures the number of clock cycles to decode a certain channel. The performance and effective parameters of each channel is described as follows:

PCFICH decoder

The processing speed of PCFICH decoder is slightly affected by the number of downlink resource blocks. Generally the number of clock cycles for PCFICH decoding is approximately 3.45×10^6. With 125MHz system clock, the PCIFCH decoding takes approximately 27.61ms.

PHCIH decoder

The processing speed of PHICH decoder is mainly affected by the orthogonal index of expected PHICH. If the orthogonal index is 0 or 4, the decoding process takes approximately 3.47×10^6 clock cycles. Otherwise the decoding process takes approximately 3.48×10^6 clock cycles. With 125MHz system clock, the PCIFCH decoding takes approximately 27.74ms.

PDCCH decoder

The processing speed of PDCCH decoder is greatly affected by the PDCCH blind decoding process and the number of REGs occupied by the PDCCH. If the UE finds the PDCCH successfully on the first PDCCH candidate it assumed, in case the number of REGs for PDCCH is roughly 7000, the PDCCH decoding process takes approximately 4.73×10^7 clock cycles which is 378.64ms. In case the number of REGs for PDDCH is roughly 3000, the decoding process takes approximately 2.55×10^7 clock cycles which is 204.15ms. If the UE does not find the PDCCH on the first PDDCH candidate it assumed, every extra blind decoding access takes roughly 5.26×10^5 clock cycles which is 42.08ms. Note that in one blind decoding access, the number of clock cycles consumed by Viterbi decoder is only 3.04×10^4 which is approximately 0.24ms in time. This means that the hardware accelerated core works very efficiently.

In real time implementation, the decoding time of the control region is at most 1ms which is the duration of one subframe as mentioned in Section 2.1. In this design, the total decoding process may take up to 460ms with the largest control region and worst PDCCH blind decoding case. This is mainly limited by the system clock speed and software implemented decoder algorithms. The most time consuming process is the descrambling and deinterleaving of PDCCH which can be improved greatly with hardware implementation. The methods to improve the decoder performance are as follows:

- Increase the system clock speed. As all components in the decoder runs at system clock speed, increasing system clock speed can improve the performance linearly. The maximum clock speed of the Microblaze processor on Xilinx XUPV5-LX110T evaluation platform is 125MHz, thus increasing the clock speed means using a faster FPGA board.
- Implement more decoder algorithms in hardware as hardware accelerated core instead of implementing in software. As mentioned before, the Viterbi decoder consumes a lot of time in software and takes only approximately 6% time in one blind PDCCH decoding access. If the decoding process such as descrambling, deinterleaving and CRC calculation is implemented in hardware, the total decoding time is reduced greatly.
- Use multiple processors instead of a single processor. For example, if two processors are used, the second processor can starts PHICH decoding when the first processor performs PCFICH decoding. Both processors can perform PDCCH decoding by memory sharing.

- Enable blast mode in SDRAM. As the heap memory is allocated in the SDRAM, and most variables are dynamically allocated on the heap, enable SDRAM blast mode may reduce time to read a large vector on the heap.
- Enhance the Microblaze processor. The processor can be enhanced by adding integer divider and optimize performance. This results in more BRAMs used for generating the Microblaze.
- Set system preferences to performance optimization instead of default. This step optimizes the speed of every component in the design but also resulting in larger area.
- Optimize software by improving coding style. This approach requires more C coding experience.

5 Conclusion

The purpose of this thesis is to design and implement the encoder and decoder of LTE physical downlink control signals. The thesis aim is achieved after design and implementation. As mentioned in Section 4, the final design functions correctly. During the encoder and decoder algorithm design, a lot of knowledge of LTE downlink signaling is gained. The encoder and decoder implementation on MATLAB sharpens my MATLAB programming skills. The decoder implementation on FPGA sharpens my C programming skills and increases my experience with designing embedded systems. After all, this thesis work is a critical part of my study as a master student in system-on-chip and may contribute to my future career. There are some limitations of this thesis work as following:

- The performance of the decoder is far behind real time implementation.
- The decoder implementation on FPGA uses a serial port and HyperTerminal which is neither fast nor convenient to use.
- The decoder algorithms are not designed in the most efficient way.
- The decoder implementation is not programed into the ROM of the FPGA. Thus the FPGA needs to be programmed by Xilinx Platform Studio to preform decoding.

According to the limitations mentioned above and discussion of results in Section 4, some further work can be done as follows:

- Reduce decoding error rate with methods mentioned in Section 4.1.
- Increase decoding speed with methods mentioned in Section 4.2.
- Use MATLAB instead of HyperTerminal to transfer data to the FPGA and simply the implementation.
- Use high speed communication methods such as an Ethernet port instead of serial port to reduce data transfer time.
- Program the FPGA by using configuration stored in ROM instead of using a PC with Xilinx Platform Studio to make the implementation easier to use.

References

[1]. Erik Dahlman, Stefan Parkvall and Johan Skold, "LTE/LTE-Advanced for Mobile Broadband", 2011, pp. 1-7.

[2]. Erik Dahlman, Stefan Parkvall and Johan Skold, "LTE/LTE-Advanced for Mobile Broadband", 2011, pp. 174-175.

[3]. 3GPP TS 36.211 Physical Channels and Modulation, version 11.2.0, Release 11, 2013-04, Section 4.1, p. 10.

[4]. 3GPP TS 36.211 Physical Channels and Modulation, version 11.2.0, Release 11, 2013-04, Section 6.2, pp. 52-55.

[5]. 3GPP TS 36.211 Physical Channels and Modulation, version 11.2.0, Release 11, 2013-04, Section 6.3, p. 57.

[6]. 3GPP TS 36.211 Physical Channels and Modulation, version 11.2.0, Release 11, 2013-04, Section 7.2, pp. 104-105.

[7]. 3GPP TS 36.211 Physical Channels and Modulation, version 11.2.0, Release 11, 2013-04, Section 7.1.1-7.1.2, p. 102.

[8]. Erik Dahlman, Stefan Parkvall and Johan Skold, "LTE/LTE-Advanced for Mobile Broadband", 2011, p. 163.

[9]. Erik Dahlman, Stefan Parkvall and Johan Skold, "LTE/LTE-Advanced for Mobile Broadband", 2011, pp. 175-177.

[10]. 3GPP TS 36.211 Physical Channels and Modulation, version 11.2.0, Release 11, 2013-04, Section 6.7, pp. 67-68.

[11]. 3GPP TS 36.212 Multiplexing and Channel Coding, version 11.2.0, Release 11, 2013-04, Section 5.3.4, p. 78.

[12]. 3GPP TS 36.211 Physical Channels and Modulation, version 11.2.0, Release 11, 2013-04, Section 6.9, pp. 73-77.

[13]. 3GPP TS 36.212 Multiplexing and Channel Coding, version 11.2.0, Release 11, 2013-04, Section 5.3.5, p. 79.

[14]. Erik Dahlman, Stefan Parkvall and Johan Skold, "LTE/LTE-Advanced for Mobile Broadband", 2011, pp. 180-185.

[15]. Erik Dahlman, Stefan Parkvall and Johan Skold, "LTE/LTE-Advanced for Mobile Broadband", 2011, pp. 193-197.

[16]. 3GPP TS 36.212 Multiplexing and Channel Coding, version 11.2.0, Release 11, 2013-04, Section 5.3.3.1.3, pp. 60-62.

[17]. 3GPP TS 36.213 Physical Procedures, version 11.2.0, Release 11, 2013-04, Section 7.1.6.3, pp. 36-37.

[18]. 3GPP TS 36.212 Multiplexing and Channel Coding, version 11.2.0, Release 11, 2013-04, Section 5.3.3.1.1, pp. 58-59.

[19]. 3GPP TS 36.212 Multiplexing and Channel Coding, version 11.2.0, Release 11, 2013-04, Section 5.1.1 and Section 5.3.3.2, pp. 8-9 and pp. 77-78.

[20]. 3GPP TS 36.212 Multiplexing and Channel Coding, version 11.2.0, Release 11, 2013-04, Section 5.1.3, pp. 11-12.

[21]. 3GPP TS 36.212 Multiplexing and Channel Coding, version 11.2.0, Release 11, 2013-04, Section 5.1.4, pp. 15-20.

[22]. 3GPP TS 36.211 Physical Channels and Modulation, version 11.2.0, Release 11, 2013-04, Section 6.8, pp. 68-70.

[23]. 3GPP TS 36.213 Physical Procedures, version 11.2.0, Release 11, 2013-04, Section 9.1.1, pp. 116-118.

[23]. 3GPP TS 36.213 Physical Procedures, version 11.2.0, Release 11, 2013-04, Section 9.1.3, p. 120.

[24]. Xilinx DS247 LogiCORE IP Viterbi Decoder v7.0 Data Sheet, 2011-03, p. 2.

[25]. MIT 6.02 Viterbi Decoding of Convolutional Codes, 2010-06, Section 9.2, pp. 2-5.

[26]. Erik Dahlman, Stefan Parkvall and Johan Skold, “LTE/LTE-Advanced for Mobile Broadband”, 2011, p.201.

Appendix

Hardware generation of the embedded decoder implementation on Xilinx XUPV5-LX110T evaluation platform

The hardware of the embedded system is generated by Xilinx base system builder (BSB) with Xilinx embedded IP cores. The generation process is described as follows:

- Download BSB files for XUPV5-LX110T evaluation platform named "EDK-XUPV5-LX110T-Pack" available at: http://www.xilinx.com/univ/xupv5-lx110t-bsb.htm.
- Open BSB in Xilinx Platform Studio (XPS). Make a valid project name and choose interconnect type to PLB. Set the project peripheral repository search path to the "lib" folder under "EDK-XUPV5-LX110T-Pack". Click "OK".
- Choose a new design and click "Next".
- Choose the XUPV5-LX110T evaluation platform in the "Board Name" selection. It is only available when the project peripheral repository search path is set correctly. Click "Next".
- Choose single processor design and click "Next".
- Leave everything as default and click "Next".
- On the peripherals selection page, keep "DDR2_SDRAM", "RS232_Uart_1", "dlmb_cntlr" and "ilmb_cntlr" as build in IP cores and remove all others. Click on the "RS232_Uart_1", set the baud rate to 19200 and enable interrupt. Click "Next".
- Use no cache and click "Next". After reading the generation report click "Finish".
- Under the "Bus Interfaces" tab, right click on the "microblaze_0" processor and configure it. Click "Advanced" and under "Buses" tab, add one stream link to the processor. Click "OK".
- Find "fsl_v20" IP under "Bus and Bridge" in the "IP Catalog" tab and add two of them to the system. Connect the both FSL to port "MFSL0" and "SFSL0" by expand the

"microblaze_0" IP under "Bus Interfaces" tab. Connect both FSL with system clock by connecting port "FSL_Clk" to the clock generator output under "Ports" tab.

- Copy the "virtex5softparallel_v1_00_a" folder to "edk_user_repository" under Xilinx folder.
- Choose "Create or Import Peripheral" under "Hardware". Choose "Import existing peripheral" and import the "virtex5softparallel" hardware core to the design. This is the core of hardware accelerated Viterbi Decoder.
- Connect the "virtex5softparallel" with both FSL under "Bus Interface" tab.
- Set the local memory size to 128kB under "Addresses" tab.
- Click "Generate Bitstream" to generate the hardware design.

http://www.eit.lth.se