# Trial Work Report

Following diagram outlines the PDSCH (Physical Downlink Shared Channel) Transport Process and shows corresponding 3GPP specification for each process.

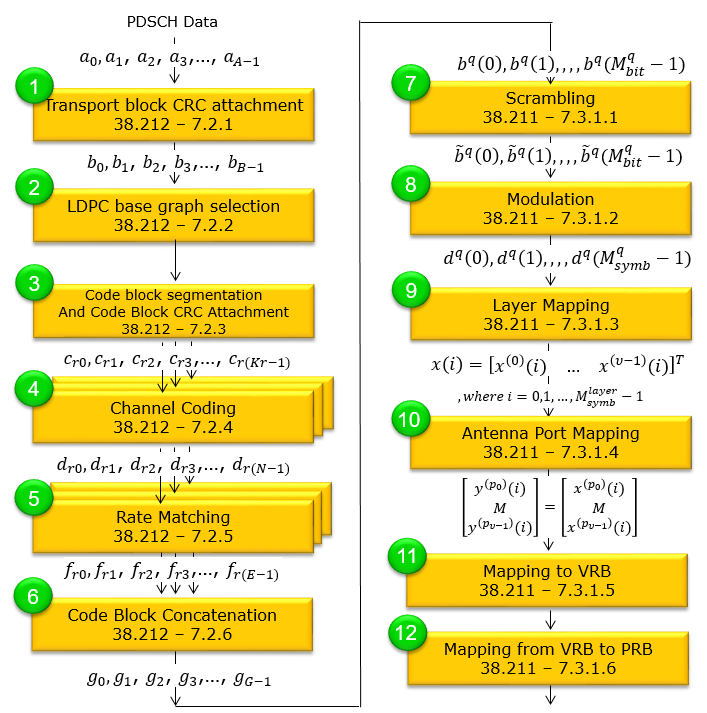


Figure 1: The PDSCH diagram

**Task**: Implement RateMatching function

## Ratematching algorithm

The steps to simulate the RateMatching function in MATLAB are as follows:

* Step 1: Understand the RateMatching algorithm
* Step 2: Implement on MATLAB (Version 2023b)

### Step 1: Understand the RateMatching algorithm

* Define the input parameters, such as:
* inputData: input array representing data to be rate-matched.
* outputData: output array holding rate-matched data.
* inRows: input array representing data to be rate-matched.
* outlen: output array holding rate-matched data.
* inputLength: Length of the input data array.
* outputLength: Length of the output data array after rate matching.
* rv: The redundancy version (rv) of the output is controlled by the rv parameter (0,1,2,3)
* nlayers: The total number of transmission layers associated with the transport block (1...4)
* Qm: Represents the modulation scheme used (e.g., QPSK <=> Qm = 2, 16-QAM <=> Qm = 4).
* Nref: The parameter defined in TS 38.212 Section 5.4.2.1.
* Read nrRateMatchLDPC function on MATLAB
* Determine base graph number from inputLength
* Get code block soft buffer size ()
* Get starting position in circular buffer
* Get rate matching output for all scheduled code blocks and perform code block concatenation according to Section 5.4.2 and 5.5
* Bit selection, Section 5.4.2.1
* Bit interleaving, Section 5.4.2.2

|  |
| --- |
| outputData = nrRateMatchLDPC(inputdata, outlen, rv, modulation, nlayers); |

### Step 2: Implement on MATLAB

Reimplement the ratematching function in MATLAB using the function nrRateMatchLDPC\_modify.

|  |
| --- |
| outputData = nrRateMatchLDPC\_modify(inputdata, outlen, rv, modulation, nlayers); |

## SystemC

### Implement C++

Implement the ratematching function using C++.

Note:

* Avoid using built-in libraries.
* Build my library to support the ratematching function, including:
* *int myFloor(double x)*: Floor function
* *int myCeil(double x)*: Ceil function

### Set up environment

Setting up the environment to simulate the RateMatching function is carried out in three steps as follows:

* Step 1: Download the SystemC library (version 2.3.3).
* Step 2: Install Visual Studio (version 2022).
* Step 3: Install the SystemC library into Visual Studio

**Implement the SystemC library**

The implementation includes three modules, which are:

* **Source**: Source module that generates input data for the rate matching process. This module is responsible for producing a continuous stream of data that meets specified requirements for further processing.
* **RateMatching**: This file implements the rate matching algorithm, which processes input data arrays and generates rate-matched output based on system parameters.
* **Sink**: Sink module that receives and processes output data from the rate matching module. This module is responsible for handling data storage or transmission to the next stage in the system.

### Interface

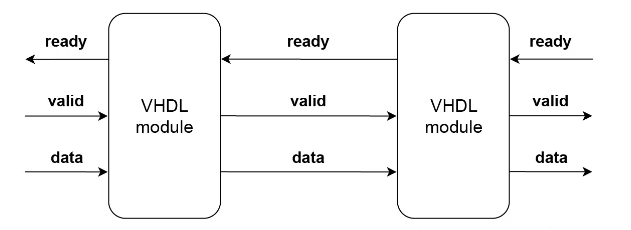


Figure 2: Connecting two modules

Figure illustrates the connection interface between the two modules.

Let *valid\_sig*, *ready\_sig*, *data\_sig* and represent the valid, ready, and data signals of the source module, respectively.

* Source connections
* source.valid\_out(valid\_sig);
* source.data\_out(data\_sig);
* source.ready\_in(ready\_sig);
* Source – RateMatching:
* rate\_matching.valid\_in(valid\_sig);
* rate\_matching.ready\_out(ready\_sig);
* rate\_matching.data\_in(data\_sig);
* RateMatching – Sink
* rate\_matching.valid\_out(valid\_out\_sig);
* rate\_matching.data\_out(data\_out\_sig);
* rate\_matching.ready\_in(ready\_in\_sig);
* Sink connections
* sink.valid\_in(valid\_out\_sig);
* sink.data\_in(data\_out\_sig);
* sink.ready\_out(ready\_in\_sig);

### Handshaking

1. ***Source***:

Wait for FIFO to be ready.

|  |
| --- |
| while (!ready\_in.read()) {  wait(); // Wait until the FIFO is ready to accept data  } |

Send data only when FIFO is ready

|  |
| --- |
| if (ready\_in.read()) |

1. ***RateMatching:***

Ready to receive data if FIFO isn't full

|  |
| --- |
| ready\_out.write(dataFIFO.size() < MAX\_FIFO\_SIZE); |

Push data into FIFO if valid and ready:

|  |
| --- |
| if (valid\_in.read() && ready\_out.read()) |

1. ***Sink***

Indicate that the Sink is ready to receive data

|  |
| --- |
| ready\_out.write(true); |

Check if valid data is received

|  |
| --- |
| if (valid\_in.read()) |