Low Density Parity Check Decoder

In communication system normally we have message bits and the parity bits added with them in order to send the message to the receiver side. There is a generator matrix which is the implementation of the parity check equations.

G is the generator matrix that generates the code word of the linear block codes.

C=MG

Here, C is the coded message, M is the original message and G is the generator matrix

At the receiver side we have the received message bit and the parity check matrix, from which we decode are message.

In LDPC code there is a property that every code digit is contained in the same no. of equations and each equations contains the same no. of code symbols.

LDPC constructions

LDPC codes are usually represented by the tanner graph. Tanner graph contains two set of vertices

1. n vertices for the code word bits called a bit nodes.
2. M vertices for the check equation called check node.

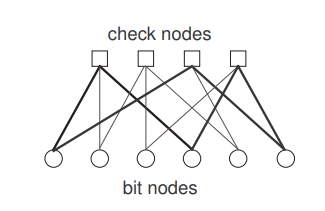
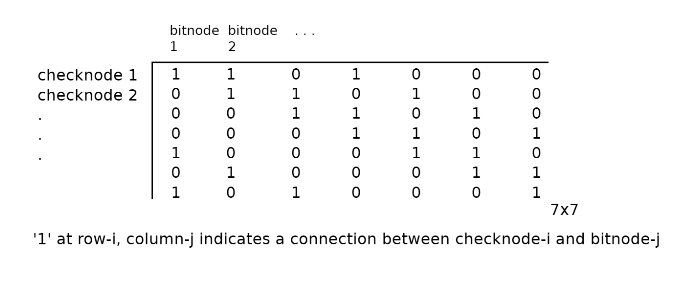


Figure 1: Representation in tanner graph

The following parity matrix is taken in the project



Tanner graph according to the parity matrix is

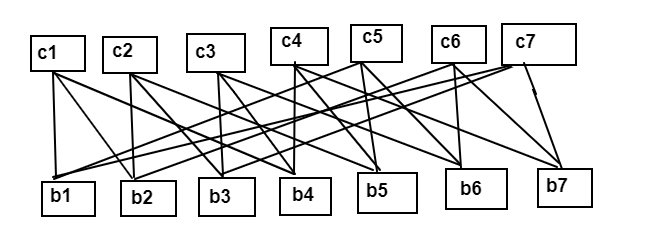


Figure 2: Diagram to show the connection of the bit nodes and check nodes.

ALGORITHM USED

Here we used the Bit Flipping algorithm, in which the received message bits are decoded according to the parity matrix and if the bit obtained is not correct, then the respective bit is flipped and then again the equations are checked. The bit flip decoder directly gives it output whenever a valid code word has obtained which satisfies all the parity check equation.

Here the c1 means the first bit of coded word is taking the input from the b1, b2 ,b4, then bit which it will send back to the b0 is the

C1= b1 ⊕ b2 ⊕ b4

C2= b2 ⊕ b3 ⊕ b5

Similarly, the code words are sending the data back to each bit nodes.

As the b1 getting the coded bit from c1, c6 and c7

Now, the bit assigned to the b1 is the majority of the bits which are send by the c1, c6 and c7.

Like this all the bits have one value assigned to them, now to check whether the decoded messages bits are correct or not all the parity equations should be checked

( b1 ⊕ b2 ⊕ b4) +(b2 ⊕ b3 ⊕ b5)+(b3⊕b4⊕b6)+(b4⊕b5⊕b7)+(b1⊕b5⊕b6)+

(b2⊕b6⊕b7)+(b1⊕b3⊕b7)

If the output of the above equation is zero means the decoded message bits are correct otherwise, it will again send the bit obtained to the check nodes and again do the calculations until the result of the equation is zero.

**DESCRIPTION**

**Verilog Source files:**

To implement this bit flipping algorithm we start with verilog codes on quartus for bitnodes bitnode core,checknodes,checknode core,check decoder,buffers(FIFO\_1 (one bit buffer)and FIFO\_7(7 bit buffer)).These bitnodes includes bitnode core which describes functionality of the bitnodes as per the bit flipping algorithm and six one bit buffers i.e. 3 FIFO\_1 because number of 1’s in H matrix are 3 means one bitnode is receiving data from 3 checknodes, 1 FIFO\_1 for in bit,1 FIFO\_1 for out bit and 1 FIFO\_1 for sending bit from bitnode to checknode. Similarly, checknodes contains checknode core and six FIFO\_1 i.e. 3 FIFO\_1,where one checknode is receiving data from three bitnodes and other 3 FIFO\_1 are used for sending bits to bitnode.Check decoder finds out the majority bits and find out if there exist an error. It receives the 6 bitnodes at its input where each bitnode receives 3 bits from 3 checknodes and gives one output. This output will tell if there is an error(decoded message is different from the actual codeword) and if exist then next iteration will start otherwise the decoded message is same as the received codeword.FIFO\_7 is the 7 bit buffer. It is used here as input buffer giving 7 bit message to the bitnode and output buffer receiving 7 bit message from the bitnode.

**YOSYS:**

It takes verilog source files written in behavioral modelling and convert them into verilog codes written in structural modelling.

We have used **Verilog to VHDL converter** available on softpedia.com. It converts structural verilog code into structural VHDL code so that layout can be generated using this VHDL file.

**Layout:**

We have used electric 9.07 using bottom-up appraoch for creating layout out of the VHDL files.It has created layout for each node with successfull design rule check(DRC) and earlier the plan was to make layout for individual module and then make layout manually for the overall LDPC decoder using these individual module. But the layouts for the individual modules were quit big and creating layout for the whole circuit using those big layout became extremely difficult. Imaging,we are using 7 bitnodes and 7 checknodes and for each node there is an individual layout means 14 layouts only for these bitnodes and checknodes, leaving other nodes aside. This was becoming something which was going out of reach for the mannual layout.Then we decided to combine all the vhdl code files for respective modules in the vhdl code file of the top level circuit and tried to generate the automatic layout from electric 9.07. We have tried to make changes in the placement settings,routing settings and to use metal layers more than 2 but electric remains unresponsive and creating layout with the default settings. But there was a problem in running DRC for this top level circuit layout because it was taking too long to complete the DRC and still not displaying the results/errors.To solve this problem we decided to shift towards Qflow open circuit design tool.

**Schematic:**

As per the bottom-up appraoch, we have designed schematic for the bitnode core,checknode core and check decoder module. Afer that we realized that the making schematic for big modules such as FIFO\_1 and FIFO\_7 would be painfull so instead of doing LVS we started comparing our VHDL netlist with SPI decs written from layout for big modules which was showing correct comparison. Anyhow, we are actaully interested in layouts so that’s why we have chosen this approach of comparing netlist with the SPI files.