Ruhr University of Bochum

Chair for Embedded Systems for Information Technology

Henry Bathke

www.esit.rub.de

- Build a Hardware encoder and decoder suitable for storage devices
- Create a system for testing the encoder and decoder

- Multiple parity check equations
- Each bit is contained in multiple equations
- With these equations we can correct errors

- Multiple parity check equations
- Each bit is contained in multiple equations
- With these equations we can correct errors
- We construct the check matrix with the check equations

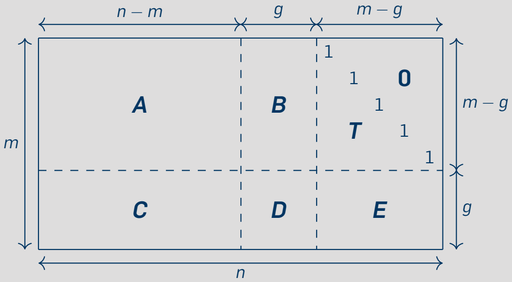
$$x_1 \oplus x_3 \oplus x_4 \oplus x_7 = 0$$

$$\boldsymbol{H} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

- High complexity of LDPC codes
- Reduce complexity by adding structure to the PCM
- Split PCM into submatrices
- Only allow shifted version of a submatrix

$$\boldsymbol{B} = \begin{bmatrix} -1 & 0 & 2 & -1 \\ 0 & 1 & -1 & -1 \\ -1 & 1 & 0 & 2 \end{bmatrix}$$

- Is usually done with generator matrix
- The generator matrix is dense due to the inversion
- With long codes the dense matrix multiplication is large
- Use transforms on the PCM to convert it into a more desireable form



- Reach minimum gap g by doing only row and column permutations
- Only need an inverted matrix of size $g \times g$

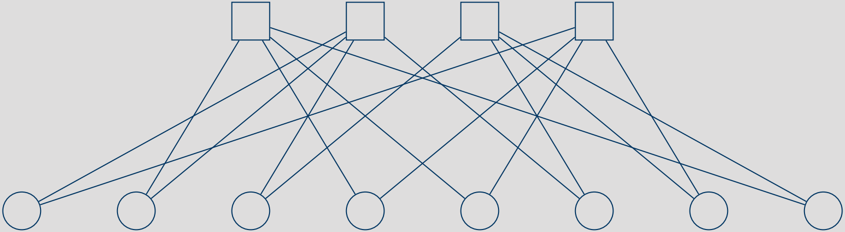
- Only large sparse matrix multiplication and back substitution
- One small dense matrix multiplication

Operation	Type
$\boldsymbol{A}s^T$	sparse multiplication
$\boldsymbol{T}^{-1}\boldsymbol{A}s^T$	sparse back substitution
$-\boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{A}s^T$	sparse multiplication
$\boldsymbol{C}s^T$	sparse multiplication
$\left(-\boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{A}s^T\right) + \left(\boldsymbol{C}s^T\right)$	vector addition
$\phi^{-1}\left(-\boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{A}s^T + \boldsymbol{C}s^T\right)$	dense $g \times g$ multiplication

- Implemented as combinatorial logic
- Connects to the other modules with an axi stream bus
- Repacking is needed as bit counts dont evenly divide
- Encoder is generated from the QC PCM with Python scripts

For wifi ldpc with rate 0.5 it looks like this

- I implemented a message passing decoder
- Messages are passed along the edges on the tanner graphicspath
- Computations are done on the nodes



$$r_{mn} = \left(\prod_{n' \in M(m) \backslash n} \operatorname{sign}(q_{n'm}) \right) \min_{n' \in M(m) \backslash n} (|q_{n'm}|)$$

$$q_{nm} = y_n + \sum_{m' \in N(n) \backslash m} r_{m'n}$$

$$L_n = y_n + \sum_{m \in N(n)} r_{m'n}$$

- We have to implement these equations
- Can exploit similarities

draw my glorious two pass tree or the block diagram

- LDPC codes are usable with flash memory
- A latency vs. error correction capability trade off enables more control
- Overclocking directly improves user experience

Questions?