CVSD Final Project: BCH Decoder

Team 003: Chen Yen Tsai, I An Hou

b11202042 b11202014

1 Introduction

This project aims to design and implement a high-performance BCH decoder hardware circuit supporting (63, 51), (255, 239), (1023, 983) BCH codes. Our design supports both hard decision and soft decision decoding modes. In hard decision mode, the decoder processes binary received symbols directly. In soft-decision mode, the decoder utilizes the LLR messages in the channel to perform decoding.

To achieve optimal hardware performance, we have adopted several algorithmic optimization techniques. These include calculating syndromes while inputting, an inverse-free Berlekamp algorithm that eliminates costly division operations in finite field arithmetic, a negative-power-free Chien search that avoids complex inverse power calculations, and an optimized correlation calculation method for soft decision decoding.

On the hardware architecture level, we employed parallelization and pipelining strategies to enhance throughput and reduce critical path delay. For soft decision decoding, we implemented parallelized hardware to process three flipped patterns simultaneously, reducing the Chien search time by 50%. The Chien search module processes eight candidate roots in parallel, achieving an 8x speedup compared to the sequential approach. Additionally, we introduced multi-cycle operations for complex computations such as the Berlekamp algorithm and syndrome calculation to balance timing and area constraints.

For data representation, we adopted polynomial representation in GF(2^m) throughout the design, which is more efficient for multiplication operations compared to power representation. We also implemented several early termination conditions for detected correct codeword and early termination for hard decision in order to skip unnecessary computations.

2 Finite State Machine

Our finite state machine is shown in the figure below.

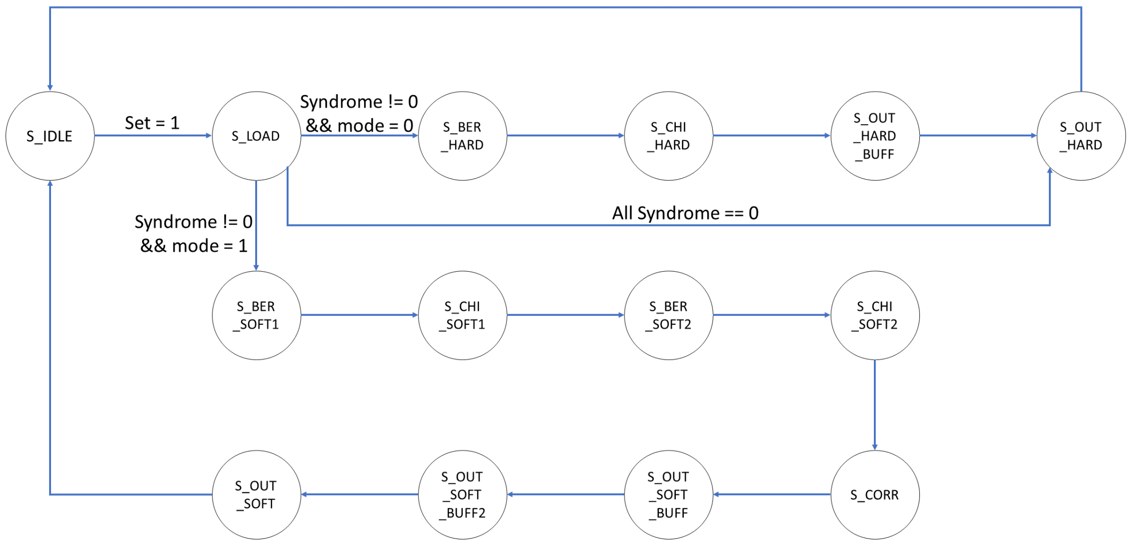


Figure 1: Finite State Machine

The function of each state is as follows:

- S_IDLE: Idle state, waits for the set signal before entering S_LOAD.
- S_LOAD: Load input data and simultaneously calculate syndrome. If mode is soft, simultaneously calculate the position of the minimum LLR. If all calculated syndromes are 0, enter S_OUT_HARD; otherwise, decide to enter S_BER_HARD or S_BER_SOFT based on mode.
- S_BER_HARD: Calculate the error location polynomial corresponding to the syndrome.
- S_CHI_HARD: Calculate the roots of the error location polynomial.
- S_OUT_HARD_BUFF: Provide a buffer cycle to ensure all roots have been stored.
- S_OUT_HARD: Assert finish high, output odata, then return to S_IDLE.
- S_BER_SOFT1: Calculate the error location polynomial corresponding to the unflipped syndrome.
- S_CHI_SOFT1: Calculate the roots of the unflipped error location polynomial and simultaneously generate three other sets of syndromes.
- S_BER_SOFT2: Calculate the error location polynomials corresponding to the other three sets of syndromes.
- S_CHI_SOFT2: Calculate the roots of the error location polynomials for the other three sets.
- S_CORR(1, 2): Calculate the correlation of the four patterns and select the result.
- S_OUT_SOFT(1, 2): Provide a buffer cycle to ensure all roots have been stored.
- S_OUT_SOFT: Assert finish high, output odata, then return to S_IDLE.

3 Algorithm Design

3.1 Finding Two Min LLR

When finding the two minimum LLRs, we use a Swiss-style double elimination tournament. The eight input data at a time are first divided into four pairs for one-on-one comparison as shown in the figure below. The smaller values enter the winner's bracket, while the larger values enter the loser's bracket. When a data has lost twice, it cannot be one of the two minimum LLRs and will be eliminated. After multiple rounds of comparison, we can obtain the two minimum LLR candidates. These two candidates are then compared with the current two minimum values to update the current minimum values. Through this method, we can optimize the hardware implementation.

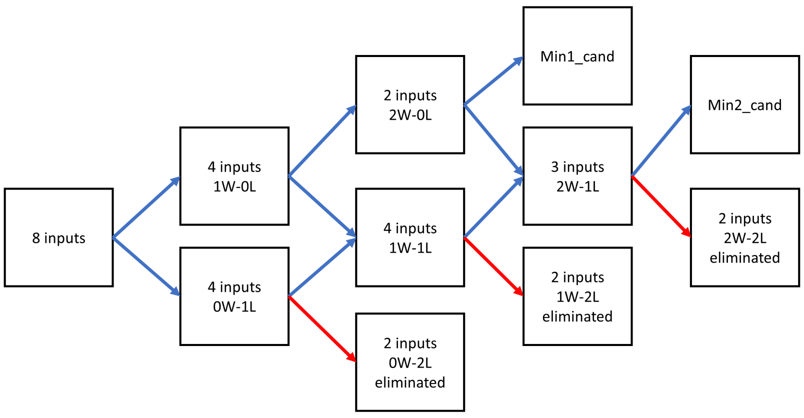


Figure 2: Swiss-style double elimination tournament

3.2 Syndrome Calculation

When calculating syndromes, we use Horner's method, which means the following polynomial sum can be written in another form:

$$a_n x^n + a_{n-1}x^{n-1} + ... + a_2x^2 + a_1x + a_0 = ((a_nx + a_{n-1})x + ... + a_1)x + a_0 \tag{1}$$

We shift the input value with the right amount and xoring it with the previous value in register for syndromes, calculating all syndromes on-the-fly with no additional storage for the codeword required.

3.3 Berlekamp Algorithm

3.3.1 Inverse-Free Berlekamp Algorithm

We implemented an inverse-free Berlekamp algorithm. Instead of updating by $\sigma^{(\mu+1)}(X) = \sigma^{\mu} + d_\mu d_\rho^{-1}\sigma^{\rho}(X)$, we instead updating by $\Delta^{(\mu+1)}(X) = \Delta^{\mu} + d_\mu \Delta^{\rho}(X)$, where $\Delta(X)$ is the "scaled" error location polynomial we obtained. With this method, after each update, we have a relation between the original and the scaled error location polynomial: $C \cdot \sigma(X) = \Delta(X)$, where C is a multiple of all d_ρ. Since the scaling factor is a constant coefficient, it doesn't affect the roots of the original polynomial. Hence, $\Delta(X)$ can be used directly in Chien search and is guaranteed to have the correct roots as $\sigma(X)$.

Also, with this modification, the way updating d is slightly different:

$$d_{\mu+1} = \sum_{i=0}^{l_{\mu+1}} \Delta_i^{\mu+1} S_{\mu+2-i}$$

The difference stems from the fact that now Δ_0 is no longer guaranteed to be 1, so the first term is now $\Delta_0^{(\mu+1)} S_{\mu+2}$.

3.3.2 Avoiding Berlekamp in Special Cases

There is a short cut when obtaining coefficients of error location polynomial in hard decision and $t = 2$. In hard decision, the test patterns are restricted to not have $> t$ errors, so the error location polynomial is guaranteed to have degree $\leq t$ ($\sigma_3 = \sigma_4 = 0$). For the case $t = 2$, we can actually explicitly solve the coefficients in one step:

$$\sigma_0 = 1 \tag{2}$$

$$S_1 + \sigma_1 = 0 \Rightarrow \sigma_1 = S_1 \tag{3}$$

$$S_3 + \sigma_1 S_2 + \sigma_2 S_1 + 3\sigma_3 = 0 = S_3 + \sigma_1 S_2 + \sigma_2 S_1 \Rightarrow \sigma_2 = (S_3 + \sigma_1 S_2) S_1^{-1} \tag{4}$$

To keep our algorithm inverse-free, we scale all coefficients by a factor of S_1 (the case where $S_1 = 0$ is handled exclusively):

$$\sigma_0 = S_1 \tag{5}$$
$$\sigma_1 = S_1^2 \tag{6}$$
$$\sigma_2 = S_3 + \sigma_1 S_2 \tag{7}$$

However, this trick is only valid in hard decision. In soft decision, the first pattern (no bits flipped) may have a polynomial with degree > 2, which should be labeled as decode failure. If this trick was applied, we would have missed the high order terms and, if unfortunate, the truncated polynomial also has two roots and this pattern happens to have biggest correlation value, this will leads to a false decoding result. Therefore, this method is only applied in hard decision for $t = 2$.

3.4 Negative-Power-Free Chien Search

To avoid calculating α^{-j} in Chien search, we noticed that in a $GF(2^m)$ field, $\alpha^{-j} = \alpha^{2^m-1-j} = \alpha^{n-j}$ for a (n, k) BCH code. Therefore, after obtaining the coefficients of error location polynomial, we run the following root searching:

Table 1: Chien Search Algorithm

iteration	term 0	term 1	term 2	term 3	term 4	result
1	Δ_0	Δ_1	Δ_2	Δ_3	Δ_4	$\Delta(\alpha^0)$
2	Δ_0	$\Delta_1\alpha$	$\Delta_2\alpha^2$	$\Delta_3\alpha^3$	$\Delta_4\alpha^4$	$\Delta(\alpha^1) = \Delta(\alpha^{62})$
3	Δ_0	$\Delta_1\alpha^2$	$\Delta_2\alpha^4$	$\Delta_3\alpha^6$	$\Delta_4\alpha^6$	$\Delta(\alpha^2) = \Delta(\alpha^{61})$
4	Δ_0	$\Delta_1\alpha^3$	$\Delta_2\alpha^6$	$\Delta_3\alpha^9$	$\Delta_4\alpha^{12}$	$\Delta(\alpha^3) = \Delta(\alpha^{60})$

In each iteration, we shifted the coefficients by 0, 1, 2, 3, 4 respectively to realize multiplication with the powers of α, and we can find the roots by descending order, and no negative powers involved.

3.5 Correlation Calculation

We noticed that the correlation sum can be simplified as follow:

$$\begin{aligned} max\left\{\sum_{i=1}^{n} l_i(1-2\hat{c_i})\right\} &= max\left\{\sum_{i=1}^{n} l_i - 2\sum_{i=1}^{n} l_i\hat{c_i}\right\} && (8)\\ &= min\left\{\sum_{i=1}^{n} l_i\hat{c_i}\right\} && (9)\\ &= min\left\{\sum_{i=1}^{n} l_ic_i - \sum_{i\in\text{flipped}} l_ic_i + \sum_{i\in\text{flipped}} l_i\hat{c_i}\right\} && (10)\\ &= min\left\{\sum_{i\in\text{flipped}} l_i(\hat{c_i}-c_i)\right\} && (11)\end{aligned}$$

Where c_i is the input codeword. Moreover, the term in the final sum has only two cases:

a) $c_i = 1$: This means $\hat{c_i} = 0$ and $l_i < 0$, and the subtraction leads to $l_i(0-1) = -l_i = |l_i|$.

b) $c_i = 1$: This means $\hat{c_i} = 1$ and $l_i > 0$, and the subtraction leads to $l_i(1-0) = l_i = |l_i|$.

So both cases have the same result, and the final equation becomes

$$max\left\{\sum_{i=1}^{n} l_i(1-2\hat{c_i})\right\} = min\left\{\sum_{i\in\text{flipped}} |l_i|\right\}$$

This means that instead of summing all the terms of the correlation value, we can simply just add the absolute value of llrs for all flipped bits in each pattern, and pick the one with the smalles sum.

3.6 Early Stop and Identifying Invalid Situation

Whether in hard decision or soft decision mode, if all calculated syndromes are 0 after completing S_LOAD, this indicates that the codeword is correct. The subsequent steps will be skipped, and the process will directly enter S_OUT and output 1023. Furthermore, in hard decision, when 2 or 4 roots are found, we'll directly enter S_OUT as well.

In soft decision mode, since there may be situations where the number of errors exceeds t, the following conditions will be determined as decode failed: when determining the polynomial using the Berlekamp Algorithm, if the polynomial degree is greater than t, or after Chien Search, if the number of roots found is less than the degree of the equation.

4 Hardware Implementation

4.1 System Architecture

Simplified architecture is shown in the figure below, notice that hardware in `BER2` and `CHI2` are three copies of `BER1` and `CHI1`, and the parallelism will be explained later.

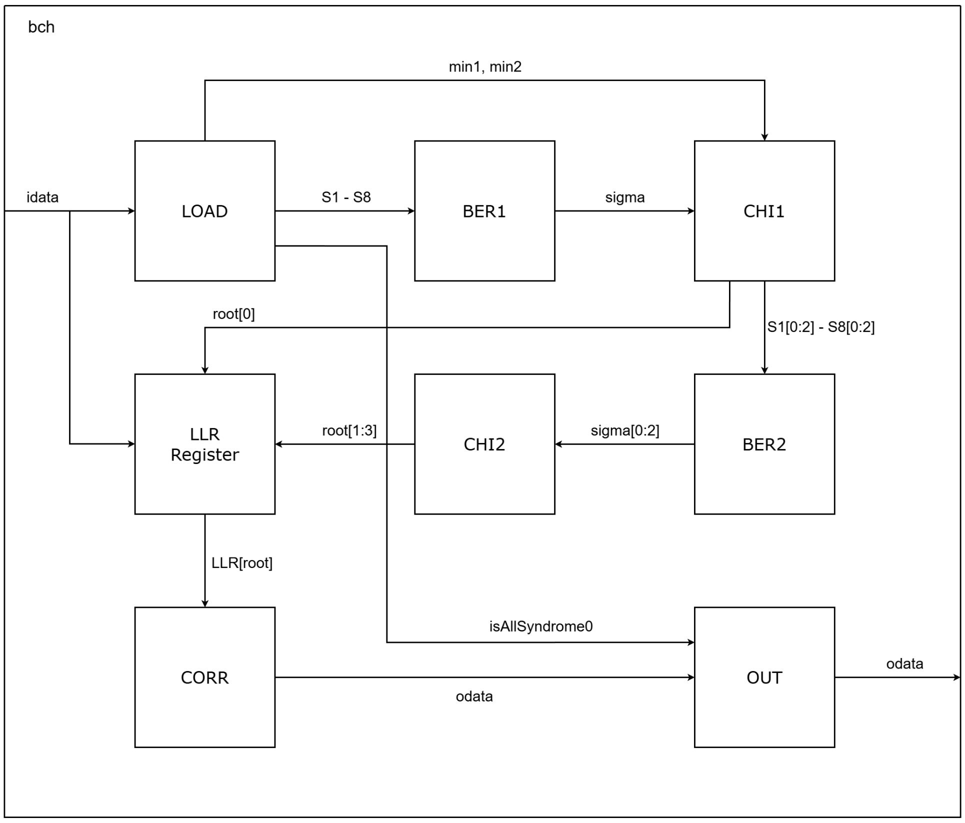


Figure 3: Hardware Architecture of BCH Decoder

4.2 Data Storage

In this project, we stored all datas in polynomial represenation of $GF(2^m)$. We chose this form for calculation efficiency, and the comparison is shown below:

Table 2: Comparison of Representations

	power representation	polynomial representation
area required	10 bits register (0 to 1023)	10 bits register (0 to 9)
addition	vector concatenation	xor
multiplication	addition	shifting

The two representations require the same storage area, but when regarding multiplication, shifting is clearly more area and time efficient than addition. The biggest flaw for power representation is that once addition is required, we must reduce the temporary polynomial to power, which is either time consuming for calculation or area consuming for LUT. Therefore, we decided to use polynomial representation.

4.3 Parallelism

4.3.1 Soft Decision (Big Parallelism)

We implemented a big parallelized hardware to run the three flipped patterns of soft decision at the same time. With this parallelized hardware, we cut the needed time for Chien searches from $4T$ to $2T$, which is a 2x improvement.

4.3.2 Chein Search

In order to speed up Chien search, we process eight possible roots in parallel (α^{j+7} to α^{j}). Though resulting in some hardware overhead, we speed up the whole process with only $\frac{1}{8}$ of cycles needed compared to the vanilla approach,

4.4 Pipeline

To decrease the length of critical path, there are two places where we implemented pipelined dataflow.

4.4.1 Chien search

After experiments, we identified a critical path starting from coefficients of $\Delta(X)$ and the ending at root list. The datapath is:

a) Shift coefficients

b) Xoring them

c) Compare it with 0

d) Store root and its llr value into root list and llr list

We put buffers between b and c to cut the path into two pieces, and this indeed led to smaller cycle time, from 5.5 s to around 4.7 s.

4.4.2 Finding Min LLR

When finding the two minimum LLRs using the Swiss-style system, the critical path consists of a total of 7 comparisons. Therefore, we used a 4-stage pipeline, as shown in the figure below, which reduces the critical path to 2 comparisons, and this indeed led to smaller cycle time, from 15 s to around 4.7 s.

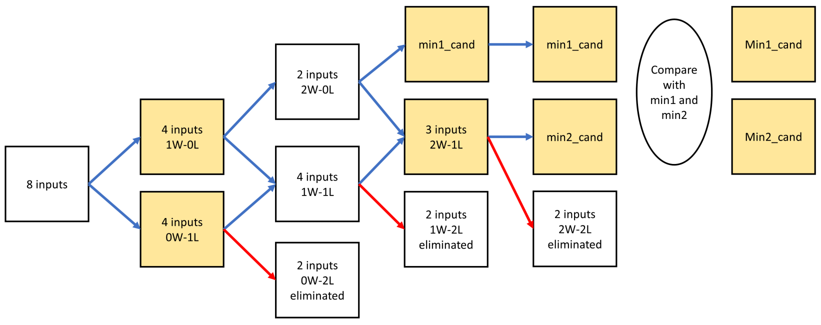


Figure 4: Pipelined swiss-style double elimination tournament

4.5 Multi-Cycle Operation

4.5.1 Berlekamp Algorithm

We also found that running a complete iteration of Berlekamp algorithm in one cycle creates a long critical path. However, Berlekamp algorithm cannot be pipelined because the next iteration fully relies on the data updated in the current iteration. To deal with this problem, we divided an iteration into two cycles. The first cycle updates $\Delta(X)$ and l, and the second one updates d. This modification reduce the cycle time from 7.5 s to around 5.5 s.

4.5.2 Syndrome Calculation

We process 8 input data per cycle. In the first cycle, we compute the initial syndrome using 7 data bits. In subsequent cycles, we process 8 new data bits and combine them with the previous syndrome result (S_r) using Horner's method, effectively computing:

$$S_w = (S_r \times \alpha^8) \oplus \text{new inputs.}$$

This multicycle approach allows us to iteratively accumulate the syndrome over multiple clock cycles until all input data has been processed, and the syndrome calculation is completed simultaneously when the input ends. This same technique is also used to calculate syndromes for the three fipped pattern when Chien search for the first pattern is performing, so the parallelized Berlekamp can be conducted right after first Chien search is done.

4.6 Clock Gating

The flip-flops for llr values consume most of the energy in the whole process. We use manual clock gating to seperate the 1024 registers into three sections: 0 - 63, 64 - 255, 256 - 1023, and the gating method is:

a) Hard decision: No llr registers are needed, so they are all gated.

b) Soft decision with $n = 63$: Only regs 0 - 63 are activated.

c) Soft decision with $n = 255$: Regs 0 - 255 are activated.

d) Soft decision with $n = 1023$: All registers are activated.

5 APR Result

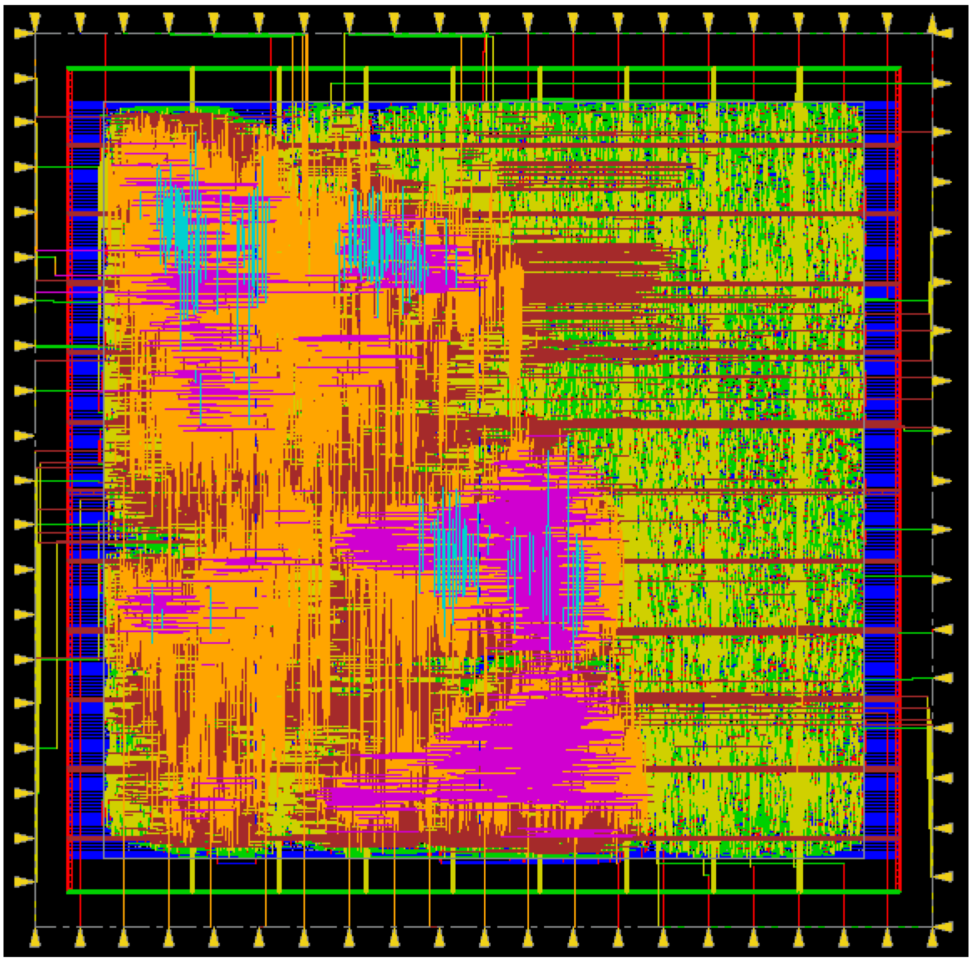


Figure 5: APR Result

6 Performance Evaluation

6.1 Time

Below is the comparison of processing time (input time is subtracted from each case) for different mode and code, including error and correct codewords.

Table 3: Processing Time Comparison

n	63	255	1023
time (hard)	158 ns	278 ns	588 ns
time (soft)	313 ns	553 ns	1588 ns
time (correct codeword)	103 ns	103 ns	103 ns

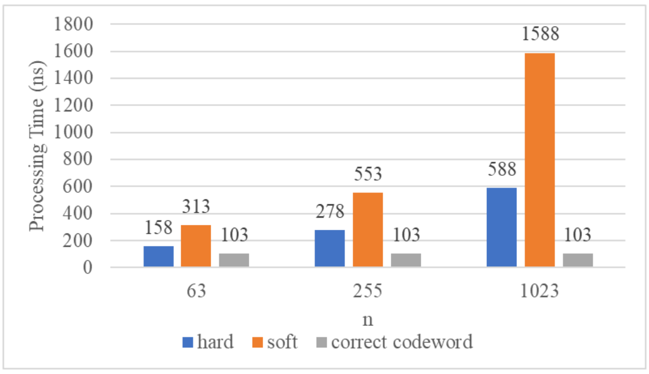
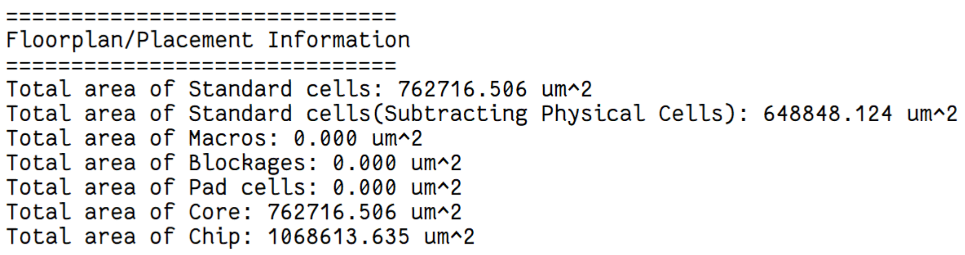


Figure 6: Processing Time Comparison

As shown above, the processing time for soft decision is only about 2x the time of hard decision, resulting from the big parallelized hardware. Additionally, with the early stop, the processing time of correct codeword can be held fix at a low constant value.

6.2 Area

The core area after APR is 762716.506 μm^2, with an utilization of 0.95.



```
===============================
Floorplan/Placement Information
===============================
Total area of Standard cells: 762716.506 um^2
Total area of Standard cells(Subtracting Physical Cells): 648848.124 um^2
Total area of Macros: 0.000 um^2
Total area of Blockages: 0.000 um^2
Total area of Pad cells: 0.000 um^2
Total area of Core: 762716.506 um^2
Total area of Chip: 1068613.635 um^2
```

Figure 7: Area Report

6.3 Power

Below is the comparison of power for different mode and code.

Table 4: Power Comparison

n	63	255	1023
hard	0.0215 W	0.0207 W	0.0217 W
soft	0.023 W	0.0303 W	0.0594 W

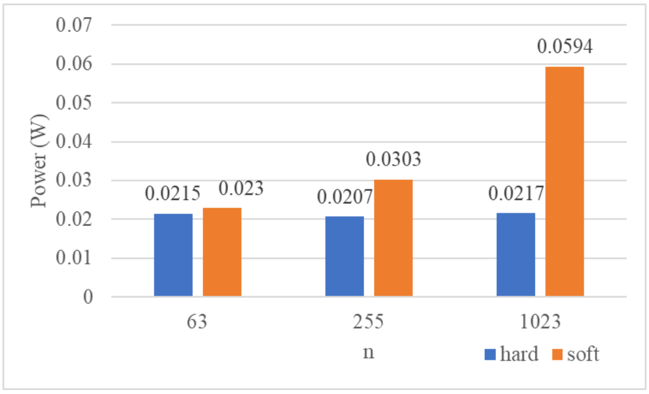


Figure 8: Power Comparison

As shown above, proper clock gating, we maintain the power of hard decision to a low fixed value and is around $\frac{1}{3}$ the power needed if no gating (soft and $n = 1023$ case, where all FFs for llr values are used).