A Low-Complexity Improved Successive Cancellation Decoder for Polar Codes

Orion Afisiadis, Alexios Balatsoukas-Stimming, Andreas Burg

Telecommunications Circuits Laboratory
EPFL

November 5th 2014
Outline & Contribution

Contribution
We introduce a successive cancellation-based decoder for polar codes which:

• has improved performance w.r.t. the standard SC decoder.
• has quasi-identical complexity—both computational and memory.
• exhibits an energy-proportional behavior.
Outline & Contribution

Contribution

We introduce a successive cancellation-based decoder for polar codes which:

- Has improved performance w.r.t. the standard SC decoder.
Outline & Contribution

Contribution

We introduce a successive cancellation-based decoder for polar codes which:

- Has improved performance w.r.t. the standard SC decoder.
- Has quasi-identical complexity—both computational and memory.
Outline & Contribution

Contribution

We introduce a successive cancellation-based decoder for polar codes which:

- Has **improved performance** w.r.t. the standard SC decoder.
- Has **quasi-identical complexity** - both computational and memory.
- Exhibits an **energy-proportional behavior**.
Outline & Contribution

Contribution

We introduce a successive cancellation-based decoder for polar codes which:

- Has **improved performance** w.r.t. the standard SC decoder.
- Has **quasi-identical complexity**—both computational and memory.
- Exhibits an **energy-proportional behavior**.

1. Polar Codes
2. Improved Successive Cancellation Decoders
3. Oracle Based Decoder
4. SC Flip Decoder
Polar Codes

Polar codes are a new class of codes, introduced by Arıkan in 2009. They have remarkable properties:

• Very structured encoder and decoder → simple routing and control logic.
• Fine-grained rate adaptation - no need for code reconstruction each time we change the rate.
• Explicit construction - no need to pick from a random ensemble.
• Provably capacity achieving - not only approaching.

Some disadvantages:
• Lower parallelism than LDPC codes → high decoding latency.
• More difficult to achieve good BER for short block lengths.

Much work is currently under way to overcome those sticking points.
Polar Codes

Polar codes are a new class of codes, introduced by Arıkan in 2009. They have remarkable properties:

- Very **structured** encoder and decoder $\rightarrow$ simple routing and control logic.
Polar Codes

Polar codes are a new class of codes, introduced by Arıkan in 2009. They have remarkable properties:

- Very **structured** encoder and decoder $\rightarrow$ simple routing and control logic.
- Fine-grained **rate adaptation** - no need for code reconstruction each time we change the rate.
Polar Codes

Polar codes are a new class of codes, introduced by Arıkan in 2009. They have remarkable properties:

- Very **structured** encoder and decoder → simple routing and control logic.
- Fine-grained **rate adaptation** - no need for code reconstruction each time we change the rate.
- Explicit construction - **no need** to pick from a random ensemble.
Polar Codes

Polar codes are a new class of codes, introduced by Arıkan in 2009. They have remarkable properties:

- Very structured encoder and decoder → simple routing and control logic.
- Fine-grained rate adaptation - no need for code reconstruction each time we change the rate.
- Explicit construction - no need to pick from a random ensemble.
- Provably capacity achieving - not only approaching.
Polar Codes

Polar codes are a new class of codes, introduced by Arıkan in 2009. They have remarkable properties:

- Very **structured** encoder and decoder $\rightarrow$ simple routing and control logic.
- Fine-grained **rate adaptation** - no need for code reconstruction each time we change the rate.
- Explicit construction - **no need** to pick from a random ensemble.
- Provably **capacity achieving** - not only approaching.

Some disadvantages:

- **Lower parallelism** that LDPC codes $\rightarrow$ **high decoding latency**.
Polar Codes

Polar codes are a new class of codes, introduced by Arıkan in 2009. They have remarkable properties:

- Very **structured** encoder and decoder → simple routing and control logic.
- Fine-grained **rate adaptation** - no need for code reconstruction each time we change the rate.
- Explicit construction - **no need** to pick from a random ensemble.
- Provably **capacity achieving** - not only approaching.

Some disadvantages:

- **Lower parallelism** that LDPC codes → **high decoding latency**.
- More **difficult to achieve good BER** for short block lengths.
Polar Codes

Polar codes are a new class of codes, introduced by Arıkan in 2009. They have remarkable properties:

- Very structured encoder and decoder → simple routing and control logic.
- Fine-grained rate adaptation - no need for code reconstruction each time we change the rate.
- Explicit construction - no need to pick from a random ensemble.
- Provably capacity achieving - not only approaching.

Some disadvantages:

- Lower parallelism that LDPC codes → high decoding latency.
- More difficult to achieve good BER for short block lengths.

Much work is currently under way to overcome those sticking points.
The Idea Behind Polarization

Channel polarization is an operation by which one constructs, out of \( N \) independent copies of a given channel \( W \), a second set of \( N \) channels that show a polarization effect:
The Idea Behind Polarization

Channel polarization is an operation by which one constructs, out of $N$ independent copies of a given channel $W$, a second set of $N$ channels that show a polarization effect:

- For large $N$, capacities of the new forged channels tend either close to 1 (good channels) or close to 0 (bad channels).
The Idea Behind Polarization

Channel polarization is an operation by which one constructs, out of \( N \) independent copies of a given channel \( W \), a second set of \( N \) channels that show a polarization effect:

- For large \( N \), capacities of the new forged channels tend either close to 1 (good channels) or close to 0 (bad channels).

- Ideal for channel coding: send data at rate 1 through the good channels and freeze the bad ones (rate 0).
The Idea Behind Polarization

Channel polarization is an operation by which one constructs, out of $N$ independent copies of a given channel $W$, a second set of $N$ channels that show a polarization effect:

- For large $N$, capacities of the new forged channels tend either close to 1 (good channels) or close to 0 (bad channels).

- Ideal for channel coding: send data at rate 1 through the good channels and freeze the bad ones (rate 0).

- This redundancy creates indeed a channel code.
Successive Cancellation (SC) Decoding

The decision metrics (LLRs) are computed as an FFT-like structure:

\[
L_N^{(i)}(y_1^N, \hat{u}_1^{i-1}) \triangleq \ln \frac{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1} | u_i = 0)}{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1} | u_i = 1)}
\]
Successive Cancellation (SC) Decoding

The decision metrics (LLRs) are computed as an FFT-like structure:

\[ L_N^{(i)}(y_1^N, \hat{u}_1^{i-1}) \triangleq \ln \frac{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1} | u_i = 0)}{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1} | u_i = 1)} \]
Successive Cancellation (SC) Decoding

The decision metrics (LLRs) are computed as an FFT-like structure:

\[ L_N^{(i)}(y_1^N, \hat{u}_1^{i-1}) \triangleq \ln \frac{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1} | u_i = 0)}{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1} | u_i = 1)} \]
Successive Cancellation (SC) Decoding

The decision metrics (LLRs) are computed as an FFT-like structure:

\[
L_N^{(i)}(y_1^N, \hat{u}_1^{i-1}) \equiv \ln \frac{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1}|u_i = 0)}{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1}|u_i = 1)}
\]

Every time the decoder reaches the leftmost column of the graph, a decision for information bit \(i\) is made according to:

- Decide \(\hat{u}_i = 0\) if we are on a frozen position.
- Decide \(\hat{u}_i = 0\) if we are on a non-frozen position and the LLR is \(\geq 0\).
- Decide \(\hat{u}_i = 1\) if we are on a non-frozen position and the LLR is \(< 0\).
Successive Cancellation (SC) Decoding

The decision metrics (LLRs) are computed as an FFT-like structure:

\[
L_N^{(i)}(y_1^N, \hat{u}_1^{i-1}) \triangleq \ln \frac{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1} | u_i = 0)}{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1} | u_i = 1)}
\]

Every time the decoder reaches the leftmost column of the graph, a decision for information bit \( i \) is made according to:

- Decide \( \hat{u}_i = 0 \) if we are on a frozen position.
- Decide \( \hat{u}_i = 0 \) if we are on a non-frozen position and the LLR is \( \geq 0 \).
- Decide \( \hat{u}_i = 1 \) if we are on a non-frozen position and the LLR is \( < 0 \).
**Successive Cancellation (SC) Decoding**

The decision metrics (LLRs) are computed as an **FFT-like** structure:

$$L^{(i)}_N(y^N_1, \hat{u}^{i-1}_1) \triangleq \ln \frac{W^{(i)}_N(y^N_1, \hat{u}^{i-1}_1 | u_i = 0)}{W^{(i)}_N(y^N_1, \hat{u}^{i-1}_1 | u_i = 1)}$$

Every time the decoder reaches the leftmost column of the graph, a decision for information bit $i$ is made according to:

- Decide $\hat{u}_i = 0$ if we are on a frozen position.
- Decide $\hat{u}_i = 0$ if we are on a non-frozen position and the LLR is $\geq 0$.
- Decide $\hat{u}_i = 1$ if we are on a non-frozen position and the LLR is $< 0$. 

---

**Graphical Illustration:**

- **Stage 0:**
  - Node $u_1$ connected to $y^1_1$.
  - Node $u_2$ connected to $y^4_1$, $y^4_2$, and $y^3_1$.
  - Node $u_3$ connected to $y^4_3$ and $y^3_2$.
  - Node $u_4$ connected to $y^4_4$.

- **Stage 1:**
  - Node $y^1_2$ connected to $u_1$.
  - Node $y^2_3$ connected to $u_2$.
  - Node $y^3_2$ connected to $u_3$.

- **Stage 2:**
  - Node $y^1_3$ connected to $u_1$.
  - Node $y^3_3$ connected to $u_3$.
  - Node $y^4_4$ connected to $u_4$.
SC Decoding as a Path Search Procedure on a Full Binary Tree

- Depth-first approach.
- No revision of previous choices.
SC Decoding as a Path Search Procedure on a Full Binary Tree

- Depth-first approach.
- No revision of previous choices.
SC Decoding as a Path Search Procedure on a Full Binary Tree

- Depth-first approach.
- No revision of previous choices.
SC Decoding as a Path Search Procedure on a Full Binary Tree

- Depth-first approach.
- No revision of previous choices.
SC Decoding as a Path Search Procedure on a Full Binary Tree

- Depth-first approach.
- No revision of previous choices.
Outline

1 Polar Codes

2 Improved Successive Cancellation Decoders

3 Oracle Based Decoder

4 SC Flip Decoder
Successive Cancellation List Decoding

- Simple SC decoding examines only one path in the decoding tree.
**Successive Cancellation List Decoding**

- Simple SC decoding examines **only one** path in the decoding tree.

- The ML decoder would examine **all** the possible paths in the binary tree → exponential complexity.
Successive Cancellation List Decoding

- **Simple SC decoding** examines **only one** path in the decoding tree.

- The ML decoder would examine **all** the possible paths in the binary tree → exponential complexity.

- **SC list decoding** examines **L paths** simultaneously and at the end decides the most likely one as the estimated codeword.
  - Small values of \( L \) are enough to approach the ML bound.

Moreover if an "oracle" is allowed to pick the path from the final list, performance is comparable to state of the art LDPC codes. Such an oracle can be easily implemented with a CRC.
Successive Cancellation List Decoding

- Simple SC decoding examines only one path in the decoding tree.

- The ML decoder would examine all the possible paths in the binary tree → exponential complexity.

- SC list decoding examines $L$ paths simultaneously and at the end decides the most likely one as the estimated codeword.
  - Small values of $L$ are enough to approach the ML bound.

- Moreover if an “oracle” is allowed to pick the path from the final list, performance is comparable to state of the art LDPC codes.
  - Such an oracle can be easily implemented with a CRC.
SC List as a Path Search Procedure on a Full Binary Tree

$L = 2$
SC List as a Path Search Procedure on a Full Binary Tree

$L = 2$
SC List as a Path Search Procedure on a Full Binary Tree

$L = 2$
SC List as a Path Search Procedure on a Full Binary Tree

$L = 2$

![Diagram of a full binary tree with labels indicating depth levels.](image-url)
SC List as a Path Search Procedure on a Full Binary Tree

$L = 2$

depth 0
depth 1
depth 2
depth 3
SC List as a Path Search Procedure on a Full Binary Tree

$L = 2$
SC List as a Path Search Procedure on a Full Binary Tree

\[ L = 2 \]
SC List as a Path Search Procedure on a Full Binary Tree

\[ L = 2 \]

Breadth-first approach with complexity constraint \( L \).
Complexity of SC and SC List Decoding

Successive Cancellation:

- Computational complexity: $O(N \log N)$
- Memory complexity: $O(N)$
Complexity of SC and SC List Decoding

**Successive Cancellation:**
- Computational complexity: $O(N \log N)$
- Memory complexity: $O(N)$

**SC List:**
- Computational complexity: $O(LN \log N)$
- Memory complexity: $O(LN)$
Complexity of SC and SC List Decoding

**Successive Cancellation:**
- Computational complexity: $\mathcal{O}(N \log N)$
- Memory complexity: $\mathcal{O}(N)$

**SC List:**
- Computational complexity: $\mathcal{O}(LN \log N)$
- Memory complexity: $\mathcal{O}(LN)$

At a given FER, SC List decoding is potentially beneficial one out of $\frac{1}{\text{FER}}$ times on average.
Complexity of SC and SC List Decoding

Successive Cancellation:

- Computational complexity: $O(N \log N)$
- Memory complexity: $O(N)$

SC List:

- Computational complexity: $O(LN \log N)$
- Memory complexity: $O(LN)$

At a given FER, SC List decoding is potentially beneficial one out of $\frac{1}{FER}$ times on average.

Most of the time, additional complexity of SC List decoding is unnecessary.
Outline

1. Polar Codes
2. Improved Successive Cancellation Decoders
3. Oracle Based Decoder
4. SC Flip Decoder
Oracle-based SC Decoder

- **Objective**: improved performance compared to simple SC and low average complexity at the same time.
Oracle-based SC Decoder

- **Objective**: improved performance compared to simple SC and **low average complexity** at the same time.

- **Error propagation** in SC → one wrong decision may result in multiple errors in an erroneous codeword.
Oracle-based SC Decoder

- **Objective**: improved performance compared to simple SC and low average complexity at the same time.

- **Error propagation** in SC → one wrong decision may result in multiple errors in an erroneous codeword.
Oracle-based SC Decoder

- **Objective**: improved performance compared to simple SC and low average complexity at the same time.

- **Error propagation** in SC → one wrong decision may result in multiple errors in an erroneous codeword.

![Error histogram (N = 1024, Eb/N0 = 2dB)](image)
**Oracle-based SC Decoder**

- **Objective**: improved performance compared to simple SC and low average complexity at the same time.

- **Error propagation** in SC → one wrong decision may result in multiple errors in an erroneous codeword.

- In many cases the noise causes a single error.
Oracle-based SC Decoder

- **Objective:** *improved performance* compared to simple SC and *low average complexity* at the same time.

- **Error propagation** in SC $\rightarrow$ one wrong decision may result in multiple errors in an erroneous codeword.

- In many cases the noise causes a single error.

Correct only the first error $\rightarrow$ oracle-based decoder.
Correcting a single error perfectly reveals a significant potential improvement of performance.
Correcting a single error perfectly reveals a significant potential improvement of performance.
Correcting a single error perfectly reveals a **significant potential improvement of performance.**
Outline

1. Polar Codes
2. Improved Successive Cancellation Decoders
3. Oracle Based Decoder
4. SC Flip Decoder
Successive Cancellation Flip Algorithm

- The goal of the SC Flip decoder is to **identify the first error** that occurs during the successive cancellation process **without employing an oracle**.
Successive Cancellation Flip Algorithm

- The goal of the SC Flip decoder is to identify the first error that occurs during the successive cancellation process without employing an oracle.
- We use a CRC that tells us whether the estimated codeword is correct or not.

Algorithm:

1. Perform simple SC decoding.
2. Calculate the CRC of the decoded codeword.
3. If the CRC does not detect an error, terminate.
4. If the CRC detects an error, flip the decision in the position that is most likely to have caused the error (lowest LLR).
5. Re-execute simple SC decoding from that position onwards.
6. Calculate the CRC of the newly decoded codeword.
7. If the CRC does not detect an error, terminate.
8. If the CRC detects an error, go to (4) and flip the second most likely error position in the initial codeword.

Maximum of T attempts to find the position of the first error.
**Successive Cancellation Flip Algorithm**

- The goal of the SC Flip decoder is to **identify the first error** that occurs during the successive cancellation process **without employing an oracle**.
- We use a **CRC** that tells us whether the estimated codeword is correct or not.

**Algorithm:**

1. Perform **simple SC decoding**.
Successive Cancellation Flip Algorithm

- The goal of the SC Flip decoder is to **identify the first error** that occurs during the successive cancellation process **without employing an oracle**.
- We use a **CRC** that tells us whether the estimated codeword is correct or not.

**Algorithm:**
1. Perform **simple SC decoding**.
2. Calculate the **CRC** of the decoded codeword.
Successive Cancellation Flip Algorithm

- The goal of the SC Flip decoder is to identify the first error that occurs during the successive cancellation process without employing an oracle.
- We use a CRC that tells us whether the estimated codeword is correct or not.

Algorithm:
1. Perform simple SC decoding.
2. Calculate the CRC of the decoded codeword.
3. If the CRC does not detect an error, terminate.
4. If the CRC detects an error, flip the decision in the position that is most likely to have caused the error (lowest LLR).
5. Re-execute simple SC decoding from that position onwards.
6. Calculate the CRC of the newly decoded codeword.
7. If the CRC does not detect an error, terminate.
8. If the CRC detects an error, go to (4) and flip the second most likely error position in the initial codeword.

Maximum of $T$ attempts to find the position of the first error.
Successive Cancellation Flip Algorithm

- The goal of the SC Flip decoder is to **identify the first error** that occurs during the successive cancellation process **without employing an oracle**.
- We use a **CRC** that tells us whether the estimated codeword is correct or not.

**Algorithm:**

1. Perform **simple SC decoding**.
2. **Calculate the CRC** of the decoded codeword.
3. If the CRC does not detect an error, **terminate**.
4. If the CRC detects an error, **flip** the decision in the position that is **most likely** to have caused the error (**lowest LLR**).
5. Re-execute simple SC decoding from that position onwards.
6. Calculate the CRC of the newly decoded codeword.
7. If the CRC does not detect an error, terminate.
8. If the CRC detects an error, go to (4) and flip the second most likely error position in the initial codeword.

Maximum of $T$ attempts to find the position of the first error.
Successive Cancellation Flip Algorithm

- The goal of the SC Flip decoder is to identify the first error that occurs during the successive cancellation process without employing an oracle.
- We use a CRC that tells us whether the estimated codeword is correct or not.

Algorithm:
1. Perform simple SC decoding.
2. Calculate the CRC of the decoded codeword.
3. If the CRC does not detect an error, terminate.
4. If the CRC detects an error, flip the decision in the position that is most likely to have caused the error (lowest LLR).
5. Re-execute simple SC decoding from that position onwards.
Successive Cancellation Flip Algorithm

- The goal of the SC Flip decoder is to identify the first error that occurs during the successive cancellation process without employing an oracle.
- We use a CRC that tells us whether the estimated codeword is correct or not.

Algorithm:
1. Perform simple SC decoding.
2. Calculate the CRC of the decoded codeword.
3. If the CRC does not detect an error, terminate.
4. If the CRC detects an error, flip the decision in the position that is most likely to have caused the error (lowest LLR).
5. Re-execute simple SC decoding from that position onwards.
6. Calculate the CRC of the newly decoded codeword.
Successive Cancellation Flip Algorithm

- The goal of the SC Flip decoder is to identify the first error that occurs during the successive cancellation process without employing an oracle.
- We use a CRC that tells us whether the estimated codeword is correct or not.

Algorithm:
1. Perform simple SC decoding.
2. Calculate the CRC of the decoded codeword.
3. If the CRC does not detect an error, terminate.
4. If the CRC detects an error, flip the decision in the position that is most likely to have caused the error (lowest LLR).
5. Re-execute simple SC decoding from that position onwards.
6. Calculate the CRC of the newly decoded codeword.
7. If the CRC does not detect an error, terminate.
Successive Cancellation Flip Algorithm

- The goal of the SC Flip decoder is to identify the first error that occurs during the successive cancellation process without employing an oracle.
- We use a CRC that tells us whether the estimated codeword is correct or not.

Algorithm:
1. Perform simple SC decoding.
2. Calculate the CRC of the decoded codeword.
3. If the CRC does not detect an error, terminate.
4. If the CRC detects an error, flip the decision in the position that is most likely to have caused the error (lowest LLR).
5. Re-execute simple SC decoding from that position onwards.
6. Calculate the CRC of the newly decoded codeword.
7. If the CRC does not detect an error, terminate.
8. If the CRC detects an error, go to (4) and flip the second most likely error position in the initial codeword.
Successive Cancellation Flip Algorithm

- The goal of the SC Flip decoder is to identify the first error that occurs during the successive cancellation process without employing an oracle.
- We use a CRC that tells us whether the estimated codeword is correct or not.

Algorithm:
1. Perform simple SC decoding.
2. Calculate the CRC of the decoded codeword.
3. If the CRC does not detect an error, terminate.
4. If the CRC detects an error, flip the decision in the position that is most likely to have caused the error (lowest LLR).
5. Re-execute simple SC decoding from that position onwards.
6. Calculate the CRC of the newly decoded codeword.
7. If the CRC does not detect an error, terminate.
8. If the CRC detects an error, go to (4) and flip the second most likely error position in the initial codeword.

Maximum of $T$ attempts to find the position of the first error.
SC Flip Algorithm - Complexity

- \( \text{SC}(y_1^N, A, k) \) performs SC decoding but flips the \( k \)-th decision.

```plaintext
1: function SCFlip(T)

Require: Channel observations \( y_1^N \), non-frozen channels \( A \)

2: \( (\hat{u}_1^N, L(y_1^N, \hat{u}_1^{i-1}|u_i)) \leftarrow \text{SC}(y_1^N, A, 0); \)

3: if \( T > 0 \) and CRC(\( \hat{u}_1^N \)) = failure then

4: \( U \leftarrow i \in A \) of \( T \) smallest \( |L(y_1^N, \hat{u}_1^{i-1}|u_i)| \);

5: for \( j \leftarrow 1 \) to \( T \) do

6: \( k \leftarrow U(j); \)

7: \( \hat{u}_1^N \leftarrow \text{SC}(y_1^N, A, k); \)

8: if CRC(\( \hat{u}_1^N \)) = success then

9: break;

10: end if

11: end for

12: end if

13: return \( \hat{u}_1^N \);
```

Computational complexity:
SC Flip Algorithm - Complexity

- SC($y_1^N, A, k$) performs SC decoding but flips the $k$-th decision.

1: function SCFlip($T$)

Require: Channel observations $y_1^N$, non-frozen channels $A$

2: $(\hat{u}_1^N, L(y_1^N, \hat{u}_1^{i-1} | u_i)) \leftarrow SC(y_1^N, A, 0)$; \hspace{1cm} 2 \rightarrow \mathcal{O}(N \log N)

3: if $T > 0$ and CRC($\hat{u}_1^N$) = failure then

4: $U \leftarrow i \in A$ of $T$ smallest $|L(y_1^N, \hat{u}_1^{i-1} | u_i)|$;

5: for $j \leftarrow 1$ to $T$ do

6: $k \leftarrow U(j)$;

7: $\hat{u}_1^N \leftarrow SC(y_1^N, A, k)$;

8: if CRC($\hat{u}_1^N$) = success then

9: break;

10: end if

11: end for

12: end if

13: return $\hat{u}_1^N$;

Computational complexity:
SC Flip Algorithm - Complexity

- SC$(y_1^N, A, k)$ performs SC decoding but flips the $k$-th decision.

1: function SCFlip($T$)

Require: Channel observations $y_1^N$, non-frozen channels $A$

2: $(\hat{u}_1^N, L(y_1^N, \hat{u}_1^{i-1}|u_i)) \leftarrow$ SC$(y_1^N, A, 0)$; \hspace{1cm} \hspace{1cm} 2 \rightarrow O(N \log N)

3: if $T > 0$ and CRC$(\hat{u}_1^N) =$ failure then \hspace{1cm} 3 \rightarrow O(N)$

4: $U \leftarrow i \in A$ of $T$ smallest $|L(y_1^N, \hat{u}_1^{i-1}|u_i)|$;

5: for $j \leftarrow 1$ to $T$ do

6: \hspace{1cm} $k \leftarrow U(j)$;

7: \hspace{2cm} $\hat{u}_1^N \leftarrow$ SC$(y_1^N, A, k)$;

8: \hspace{2cm} if CRC$(\hat{u}_1^N) =$ success then

9: \hspace{3cm} break;

10: \hspace{2cm} end if

11: end for

12: end if

13: return $\hat{u}_1^N$;

Computational complexity:
SC Flip Algorithm - Complexity

- \( SC(y_1^N, A, k) \) performs SC decoding but flips the \( k \)-th decision.

```plaintext
1: function \( SCFlip(T) \)

Require: Channel observations \( y_1^N \), non-frozen channels \( A \)

2: \( (\hat{u}_1^N, L(y_1^N, \hat{u}_1^{i-1} | u_i)) \leftarrow SC(y_1^N, A, 0); \)

3: if \( T > 0 \) and \( CRC(\hat{u}_1^N) = \text{failure} \) then

4: \( \mathcal{U} \leftarrow i \in A \) of \( T \) smallest \( |L(y_1^N, \hat{u}_1^{i-1} | u_i)| \);

5: for \( j \leftarrow 1 \) to \( T \) do

6: \( k \leftarrow \mathcal{U}(j); \)

7: \( \hat{u}_1^N \leftarrow SC(y_1^N, A, k); \)

8: if \( CRC(\hat{u}_1^N) = \text{success} \) then

9: break;

10: end if

11: end for

12: end if

13: return \( \hat{u}_1^N \);

Computational complexity:
SC Flip Algorithm - Complexity

- \( \text{SC}(y_1^N, \mathcal{A}, k) \) performs SC decoding but flips the \( k \)-th decision.

```
1: function SCFlip(T)

Require: Channel observations \( y_1^N \), non-frozen channels \( \mathcal{A} \)

2: \( (\hat{u}_1^N, L(y_1^N, \hat{u}_1^{i-1} | u_i)) \leftarrow \text{SC}(y_1^N, \mathcal{A}, 0); \)

3: if \( T > 0 \) and CRC(\( \hat{u}_1^N \)) = failure then

4: \( \mathcal{U} \leftarrow i \in \mathcal{A} \) of \( T \) smallest \(|L(y_1^N, \hat{u}_1^{i-1} | u_i)|\);

5: for \( j \leftarrow 1 \) to \( T \) do

6: \( k \leftarrow \mathcal{U}(j); \)

7: \( \hat{u}_1^N \leftarrow \text{SC}(y_1^N, \mathcal{A}, k); \)

8: if CRC(\( \hat{u}_1^N \)) = success then

9: break;

10: end if

11: end for

12: end if

13: return \( \hat{u}_1^N \);
```

**Computational complexity:**
SC Flip Algorithm - Complexity

- SC\((y_1^N, A, k)\) performs SC decoding but flips the \(k\)-th decision.

1: function SCFlip\((T)\)

Require: Channel observations \(y_1^N\), non-frozen channels \(A\)

2: \((\hat{u}_1^N, L(y_1^N, \hat{u}_1^{i-1}|u_i)) \leftarrow \text{SC}(y_1^N, A, 0)\);  

3: if \(T > 0\) and CRC(\(\hat{u}_1^N\)) = failure then

4: \(\mathcal{U} \leftarrow i \in A\) of \(T\) smallest \(|L(y_1^N, \hat{u}_1^{i-1}|u_i)|\);  

5: for \(j \leftarrow 1\) to \(T\) do

6: \(k \leftarrow \mathcal{U}(j)\);  

7: \(\hat{u}_1^N \leftarrow \text{SC}(y_1^N, A, k)\);  

8: if CRC(\(\hat{u}_1^N\)) = success then

9: break;  

10: end if

11: end for

12: end if

13: return \(\hat{u}_1^N\);

Computational complexity:
**SC Flip Algorithm - Complexity**

- SC($y_1^N, \mathcal{A}, k$) performs SC decoding but flips the $k$-th decision.

```python
1: function SCFlip(T)

Require: Channel observations $y_1^N$, non-frozen channels $\mathcal{A}$

2: $(\hat{u}_1^N, L(y_1^N, \hat{u}_1^{i-1}|u_i)) \leftarrow$ SC($y_1^N, \mathcal{A}, 0$);

3: if $T > 0$ and CRC($\hat{u}_1^N$) = failure then

4: $\mathcal{U} \leftarrow i \in \mathcal{A}$ of $T$ smallest $|L(y_1^N, \hat{u}_1^{i-1}|u_i)|$;

5: for $j \leftarrow 1$ to $T$ do

6: $k \leftarrow \mathcal{U}(j)$;

7: $\hat{u}_1^N \leftarrow$ SC($y_1^N, \mathcal{A}, k$);

8: if CRC($\hat{u}_1^N$) = success then

9: break;

10: end if

11: end for

12: end if

13: return $\hat{u}_1^N$;
```

### Computational complexity:

- 2 $\rightarrow \mathcal{O}(N \log N)$
- 3 $\rightarrow \mathcal{O}(N)$
- 4 $\rightarrow \mathcal{O}(N \log N)$
- 5 $\rightarrow \mathcal{O}(TN \log N)$
- 7 $\rightarrow \mathcal{O}(N \log N)$
- 8 $\rightarrow \mathcal{O}(N)$
SC Flip Algorithm - Complexity

- SC($y_1^N, A, k$) performs SC decoding but flips the $k$-th decision.

```
1: function SCFlip(T)
Require: Channel observations $y_1^N$, non-frozen channels $A$
2: \[ (\hat{u}_1^N, L(y_1^N, \hat{u}_1^{i-1}|u_i)) \leftarrow SC(y_1^N, A, 0); \]
3: if $T > 0$ and CRC($\hat{u}_1^N$) = failure then
4: \[ U \leftarrow i \in A \text{ of } T \text{ smallest } |L(y_1^N, \hat{u}_1^{i-1}|u_i)|; \]
5: for $j \leftarrow 1$ to $T$ do
6: \[ k \leftarrow U(j); \]
7: \[ \hat{u}_1^N \leftarrow SC(y_1^N, A, k); \]
8: if CRC($\hat{u}_1^N$) = success then
9: \[ \text{break}; \]
10: end if
11: end for
12: end if
13: return $\hat{u}_1^N$;
```

Computational complexity: $\mathcal{O}(TN \log N)$
SC Flip Algorithm - Complexity

- SC($y_1^N, A, k$) performs SC decoding but flips the $k$-th decision.

1: function SCFlip($T$)
2: Require: Channel observations $y_1^N$, non-frozen channels $A$
3: $\left(\hat{u}_1^N, L(y_1^N, \hat{u}_1^{i-1}|u_i)\right) \leftarrow \text{SC}(y_1^N, A, 0)$;
4: if $T > 0$ and CRC($\hat{u}_1^N$) = failure then
5: $U \leftarrow i \in A$ of $T$ smallest $|L(y_1^N, \hat{u}_1^{i-1}|u_i)|$;
6: for $j \leftarrow 1$ to $T$ do
7: $k \leftarrow U(j)$;
8: $\hat{u}_1^N \leftarrow \text{SC}(y_1^N, A, k)$;
9: if CRC($\hat{u}_1^N$) = success then
10: break;
11: end if
12: end for
13: end if
14: return $\hat{u}_1^N$;

Computational complexity: $O(N \log N (1 + T \cdot \text{FER}))$
SC Flip Algorithm - Complexity

- SC($y_1^N, A, k$) performs SC decoding but flips the $k$-th decision.

```plaintext
1: function SCFlip(T)
Require: Channel observations $y_1^N$, non-frozen channels $A$
2: $(\hat{u}_1^N, L(y_1^N, \hat{u}_1^{i-1}|u_i)) \leftarrow SC(y_1^N, A, 0); \quad 2 \rightarrow O(N)$
3: if $T > 0$ and CRC($\hat{u}_1^N$) = failure then
4: $\mathcal{U} \leftarrow i \in A$ of $T$ smallest $|L(y_1^N, \hat{u}_1^{i-1}|u_i)|$;
5: for $j \leftarrow 1$ to $T$ do
6: $k \leftarrow \mathcal{U}(j)$;
7: $\hat{u}_1^N \leftarrow SC(y_1^N, A, k)$;
8: if CRC($\hat{u}_1^N$) = success then
9: break;
10: end if
11: end for
12: end if
13: return $\hat{u}_1^N$;
```

Computational complexity: $O(N \log N(1 + T \cdot FER))$

Memory complexity:
SC Flip Algorithm - Complexity

- SC($y_1^N, A, k$) performs SC decoding but flips the $k$-th decision.

1: function SCFlip($T$)

Require: Channel observations $y_1^N$, non-frozen channels $A$

2: \( (\hat{u}_1^N, L(y_1^N, \hat{u}_1^{i-1}|u_i)) \leftarrow SC(y_1^N, A, 0); \)
\( 2 \rightarrow \mathcal{O}(N) \)

3: if $T > 0$ and CRC($\hat{u}_1^N$) = failure then

4: \( U \leftarrow i \in A$ of $T$ smallest $|L(y_1^N, \hat{u}_1^{i-1}|u_i)|; \)
\( 4 \rightarrow \mathcal{O}(N) \)

5: for $j \leftarrow 1$ to $T$ do

6: \( k \leftarrow U(j); \)

7: \( \hat{u}_1^N \leftarrow SC(y_1^N, A, k); \)

8: if CRC($\hat{u}_1^N$) = success then

9: \hspace{1em} break;

10: end if

11: end for

12: end if

13: return $\hat{u}_1^N$;

Computational complexity: \( \mathcal{O}(N \log N(1 + T \cdot FER)) \)

Memory complexity:
**SC Flip Algorithm - Complexity**

- SC\( (y_1^N, A, k) \) performs SC decoding but flips the \( k \)-th decision.

1: function \( \text{SCFlip}(T) \)

Require: Channel observations \( y_1^N \), non-frozen channels \( A \)

2:\( \left( \hat{u}_1^N, L(y_1^N, \hat{u}_1^{i-1}|u_i) \right) \leftarrow \text{SC}(y_1^N, A, 0) \);  \( \rightarrow \mathcal{O}(N) \)

3: if \( T > 0 \) and CRC(\( \hat{u}_1^N \)) = failure then

4: \( U \leftarrow i \in A \) of \( T \) smallest \( |L(y_1^N, \hat{u}_1^{i-1}|u_i)| \); \( \rightarrow \mathcal{O}(N) \)

5: for \( j \leftarrow 1 \) to \( T \) do

6: \( k \leftarrow U(j) \);

7: \( \hat{u}_1^N \leftarrow \text{SC}(y_1^N, A, k) \); \( \rightarrow \) no additional

8: if CRC(\( \hat{u}_1^N \)) = success then

9: break;

10: end if

11: end for

12: end if

13: return \( \hat{u}_1^N \);

**Computational complexity:** \( \mathcal{O}(N \log N(1 + T \cdot \text{FER})) \)

**Memory complexity:**
SC Flip Algorithm - Complexity

- SC($y_1^N, A, k$) performs SC decoding but flips the $k$-th decision.

1: function SCFlip($T$)

Require: Channel observations $y_1^N$, non-frozen channels $A$

2: ($\hat{u}_1^N, L(y_1^N, \hat{u}_1^{i-1} | u_i)$) ← SC($y_1^N, A, 0$); $2 \rightarrow O(N)$

3: if $T > 0$ and CRC($\hat{u}_1^N$) = failure then

4: $U \leftarrow i \in A$ of $T$ smallest $|L(y_1^N, \hat{u}_1^{i-1} | u_i)|$; $4 \rightarrow O(N)$

5: for $j \leftarrow 1$ to $T$ do

6: $k \leftarrow U(j)$;

7: $\hat{u}_1^N \leftarrow$ SC($y_1^N, A, k$);

8: if CRC($\hat{u}_1^N$) = success then

9: break;

10: end if

11: end for

12: end if

13: return $\hat{u}_1^N$;

Computational complexity: $O(N \log N (1 + T \cdot FER))$

Memory complexity: $O(N)$
Computational Complexity of SC Flip Decoder

SC Flip exhibits an energy proportional behavior.
Computational Complexity of SC Flip Decoder

SC Flip exhibits an **energy proportional** behavior.

- When the problem is relatively easy it follows the simplest, easiest and most energy efficient way.
Computational Complexity of SC Flip Decoder

SC Flip exhibits an energy proportional behavior.

- When the problem is relatively easy it follows the simplest, easiest and most energy efficient way.
- When the problem gets harder it uses its ability to try up to $T$ times for each erroneous codeword.
Computational Complexity of SC Flip Decoder

SC Flip exhibits an **energy proportional** behavior.

- When the problem is relatively easy it follows the simplest, easiest and most energy efficient way.
- When the problem gets harder it uses its ability to try up to $T$ times for each erroneous codeword.
- Thus the **average complexity** depends on the SNR.
Computational Complexity of SC Flip Decoder

SC Flip exhibits an \textbf{energy proportional} behavior.

- When the problem is relatively easy it follows the simplest, easiest and most energy efficient way.
- When the problem gets harder it uses its ability to try up to $T$ times for each erroneous codeword.
- Thus the \textbf{average complexity} depends on the SNR.

![Graph showing the computational complexity of SC Flip decoder]
Computational Complexity of SC Flip Decoder

SC Flip exhibits an **energy proportional** behavior.

- When the problem is relatively easy it follows the simplest, easiest and most energy efficient way.
- When the problem gets harder it uses its ability to try up to $T$ times for each erroneous codeword.
- Thus the **average complexity** depends on the SNR.

SC Flip complexity is close to simple SC decoder complexity for many useful SNRs.
Performance of SC Flip Decoder

Close-to-oracle performance with $T = 32$ for $N = 1024$. 
Performance close to SC List ($L = 2$), with lower complexity.
Conclusion

- We studied the impact of error propagation on SC decoding of polar codes.
Conclusion

- We studied the impact of error propagation on SC decoding of polar codes.

- Based on that insight, we introduced an SC-based decoder with
  - Improved FER performance
  - Low computational and memory complexity
  - Energy-proportional behavior
Conclusion

- We studied the impact of error propagation on SC decoding of polar codes.

- Based on that insight, we introduced an SC-based decoder with
  - Improved FER performance
  - Low computational and memory complexity
  - Energy-proportional behavior

- Performance is limited by inability to correct multiple errors
  - Ongoing work!
Thank you!

Questions?
As $N$ increases it is more probable that only one error exists. **But:** more difficult to find its position.
As $N$ increases it is more probable that only one error exists. **But:** more difficult to find its position.
SC Flip - SC List Performance Comparison

Performance close to SC List \((L = 2)\), with lower complexity.
Performance close to SC List ($L = 2$), with lower complexity.
Oracle-based SC FlipMore

- SC Flip has a bound given by the oracle-based decoder.
Oracle-based SC Flip More

- SC Flip has a bound given by the oracle-based decoder.

- To overcome it we need more error corrections $S > 1$. 
Oracle-based SC FlipMore

- SC Flip has a bound given by the oracle-based decoder.

- To overcome it we need more error corrections $S > 1$.

- This idea does not have a performance bound.
Oracle-based SC FlipMore

- SC Flip has a bound given by the oracle-based decoder.

- To overcome it we need more error corrections $S > 1$.

- This idea does not have a performance bound.

- Region of $S$: No real need for $S > 4$. 
Oracle-based SC FlipMore

- SC Flip has a bound given by the oracle-based decoder.

- To overcome it we need more error corrections $S > 1$.

- This idea does not have a performance bound.

- Region of $S$: No real need for $S > 4$.

- Oracle-based implementation of this idea.
Successive Cancellation FlipMore

Figure: FER of SC and oracle SC decoders with $S=1,2,3$ ($N = 1024$ and $R = 0.5$)
Future Work

- Implementation of SC FlipMore
Future Work

- Implementation of SC FlipMore
  - A-priori information of channel quality known from code construction
Future Work

- Implementation of SC FlipMore
  - A-priori information of channel quality known from code construction
  - Channel parameters match to LLRs
Future Work

- Implementation of SC FlipMore
  - A-priori information of channel quality known from code construction
  - Channel parameters match to LLRs
  - First heuristic approach: adding them to the LLRs
    \[ |LLR(A)| + 2 \cdot \text{channelParam}(A) \]
Future Work

- Implementation of SC FlipMore
  - A-priori information of channel quality known from code construction
  - Channel parameters match to LLRs
  - First heuristic approach: adding them to the LLRs
    \[ |LLR(A)| + 2 \cdot \text{channelParam}(A) \]
  - Define a more effective metric
Future Work

- Implementation of SC FlipMore
  - A-priori information of channel quality known from code construction
  - Channel parameters match to LLRs
  - First heuristic approach: adding them to the LLRs
    \[ |LLR(A)| + 2 \cdot \text{channelParam}(A) \]
  - Define a more effective metric

- Divide and check
Future Work

- Implementation of SC FlipMore
  - A-priori information of channel quality known from code construction
  - Channel parameters match to LLRs
  - First heuristic approach: adding them to the LLRs
    \[ |LLR(A)| + 2 \cdot \text{channelParam}(A) \]
  - Define a more effective metric

- Divide and check
  - LLR approach
Future Work

- Implementation of SC FlipMore
  - A-priori information of channel quality known from code construction
  - Channel parameters match to LLRs
  - First heuristic approach: adding them to the LLRs
    \[ |LLR(A)| + 2 \cdot \text{channelParam}(A) \]
  - Define a more effective metric

- Divide and check
  - LLR approach
  - Small CRCs approach
Future Work

- Implementation of SC FlipMore
  - A-priori information of channel quality known from code construction
  - Channel parameters match to LLRs
  - First heuristic approach: adding them to the LLRs
    \[ |LLR(A)| + 2 \cdot \text{channelParam}(A) \]
  - Define a more effective metric

- Divide and check
  - LLR approach
  - Small CRCs approach

- Hardware implementation of SC Flip decoder