Documentation for Interleavers

Introduction

Interleavers play a crucial role in turbo codes by dispersing burst errors across multiple code blocks. This process makes the errors appear more random, which enhances the performance of error correction algorithms. These algorithms are generally more effective when handling random errors as opposed to clustered ones. Pseudorandom interleavers allow for the use of deinterleavers during the decoding phase, ensuring that the original data order is restored for accurate correction.

In this document, we discuss three basic types of pseudorandom interleavers:

1. Modular Interleaver

The Modular Interleaver leverages the mathematical property that when m and n are coprimes, the multiples of m (i.e., $m, 2m, 3m, 4m, \ldots, n \times m$) produce distinct remainders when divided by n. These remainders create a permutation of values ranging from 0 to $n - 1$.

Inputs: Two integers m and n

Purpose: Creates a simple permutation that spreads out input values across a range, ensuring diversity in their distribution.

Listing 1: Modular Interleaver Code

```c
#include <stdio.h>
#include <stdlib.h>

// Function to allocate and generate the interleaver permutation
int *generatePermutation(int m, int n) {
    // Allocate memory for the permutation array
    int *permut = (int *)malloc(n * sizeof(int));

    // Check if memory allocation is successful
    if (permut == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);  // Exit if memory allocation fails
    }

    // Fill the permutation array with modular interleaver values
    for (int i = 0; i < n; ++i) {
        permut[i] = (m * (i + 1)) % n;
    }
    return permut;
}

// Function to print the permutation array
void printPermutation(int *permut, int n) {
    for (int i = 0; i < n; ++i) {
        printf("%d ", permut[i]);
    }
    printf("\n");
}

int main() {
    int m, n;

    // Take user input for m and n
    printf("Enter values for m and n: ");
    scanf("%d %d", &m, &n);

    // Generate the modular interleaver permutation
    int *p = generatePermutation(m, n);

    // Print the permutation result
    printPermutation(p, n);

    // Free the allocated memory
    free(p);

    return 0;
}
```

2. Block Interleaver

A Block Interleaver organizes data into a 2D matrix, filling the matrix row by row, and then reads it out column by column to form the interleaved array.

Logic:

- Fill a matrix row-wise with sequential data.
- Read the matrix column-wise to form the interleaved array.

Inputs: Number of rows r and columns c

Purpose: This method is particularly useful for managing burst errors by spreading data across multiple blocks, providing a structured form of interleaving.

Listing 2: Block Interleaver Code

```
#include <stdio.h>
#include <stdlib.h>

// Function to print the interleaved array
void Print(int *p, int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", p[i]);  // Print each element
    }
    printf("\n");
}

// Block interleaver function: it arranges elements in a 2D matrix and reads them column-wise
```

```c
int *Block_Interleaver(int r, int c) {
    // Allocate memory for the interleaved array
    int *p = (int *)malloc(r * c * sizeof(int));
    if (p == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);  // Exit if memory allocation fails
    }

    // Create a 2D array (row-major order) to fill with the numbers
    int **arr = (int **)malloc(r * sizeof(int *));
    for (int i = 0; i < r; i++) {
        arr[i] = (int *)malloc(c * sizeof(int));
    }

    // Fill the 2D array with sequential values
    int k = 1;
    for (int i = 0; i < r; ++i) {
        for (int j = 0; j < c; ++j) {
            arr[i][j] = k;
            k++;
        }
    }

    // Now interleave the values column by column into the 1D array ’p’
    int idx = 0;
    for (int i = 0; i < c; ++i) {  // Iterate over columns
        for (int j = 0; j < r; ++j) {  // Iterate over rows
            p[idx] = arr[j][i];  // Column-major order filling
            idx++;
        }
    }

    // Free the dynamically allocated 2D array
    for (int i = 0; i < r; i++) {
        free(arr[i]);
    }
    free(arr);

    return p;
}

int main() {
    int r, c;

    // Prompt the user for the number of rows and columns of the matrix
    printf("Enter the number of rows and columns for the interleaver matrix (r, c): ");
    scanf("%d %d", &r, &c);

    // Call the Block_Interleaver function to get the interleaved data
    int *p = Block_Interleaver(r, c);

    // Print the interleaved data
    int size = r * c;
    printf("Interleaved Data: ");
    Print(p, size);

    // Free the memory allocated for the interleaved array
    free(p);

    return 0;
}
```

3. Random Interleaver

A Random Interleaver rearranges the input data by shuffling the elements using a random number generator. The shuffling helps to mitigate the effects of burst errors, which tend to affect consecutive bits in data.

How It Works: The program uses a user-provided seed to initialize the random number generator. This seed ensures that the shuffling process is repeatable. The data is then reordered according to a randomly generated sequence of indices.

Purpose: By randomizing the order of data, this interleaver reduces the likelihood of burst errors affecting consecutive bits, making error correction more effective.

Listing 3: Random Interleaver Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Function to generate an interleaved sequence
int *interleaver(int *data, int size, int key) {
    // Allocate memory for the interleaved data and indices
    int *p = (int *)malloc(size * sizeof(int));
    int *indices = (int *)malloc(size * sizeof(int));

    // Use the provided key as the seed for the random number generator
    srand(key); // Initialize the random number generator with the provided key (seed)

    // Initialize indices with original positions (0, 1, 2, ..., size-1)
    for (int i = 0; i < size; i++) {
        indices[i] = i;
    }

    // Shuffle the indices randomly based on the key
    for (int i = 0; i < size; i++) {
        int j = rand() % size;  // Pick a random index j
        int temp = indices[i];
        indices[i] = indices[j];
        indices[j] = temp;
    }

    // Apply the permutation to data (data is shuffled according to indices)
    for (int i = 0; i < size; i++) {
        p[i] = data[indices[i]]; // Place the data elements into the new shuffled order
    }

    // Free the memory allocated for indices
    free(indices);

    // Return the interleaved data
    return p;
}

// Function to print an array
void printArray(int *arr, int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int size, key;

    // Ask user for the size of the data (number of elements)
    printf("Enter the number of elements (size): ");
    scanf("%d", &size);
```

```c

    // Ask user for the key (seed) for the random number generator
    printf("Enter the key (seed) for random number generation: ");
    scanf("%d", &key);

    // Automatically generate the data array as 1, 2, 3, 4, ..., size
    int *data = (int *)malloc(size * sizeof(int));
    for (int i = 0; i < size; i++) {
        data[i] = i + 1;  // Initialize data as 1, 2, 3, ..., size
    }

    // Generate interleaved data
    int *interleavedData = interleaver(data, size, key);
    printf("Interleaved data: ");
    printArray(interleavedData, size);

    // Free the dynamically allocated memory
    free(data);
    free(interleavedData);

    return 0;
}
```