Huffman Encoding

For Lossless Data Compression

PA 23

Jesaya David & Bryan Farras

OUTLINE

01 LATAR BELAKANG

02 IMPLEMENTASI

03 UJI PROGRAM

04 KESIMPULAN

LATAR BELAKANG

Proyek ini berfokus pada implementasi algoritma Huffman Encoding untuk kompresi data lossless menggunakan pendekatan perangkat keras berbasis VHDL. Tujuan utama dari proyek ini adalah untuk menciptakan solusi yang efisien dalam menyimpan dan mengkompres data teks besar dengan memanfaatkan keunggulan FPGA untuk meningkatkan kecepatan dan efisiensi proses.

Untuk meningkatkan efisiensi proses kompresi, algoritma ini diimplementasikan menggunakan VHDL pada FPGA. FPGA dipilih karena kemampuannya untuk memproses data secara paralel dan secara real-time, yang sangat penting untuk menangani string data yang besar.

LATAR BELAKANG

Untuk meningkatkan efisiensi proses kompresi, algoritma ini diimplementasikan menggunakan VHDL pada FPGA. FPGA dipilih karena kemampuannya untuk memproses data secara paralel dan secara real-time, yang sangat penting untuk menangani string data yang besar. Proses implementasi mencakup:

1. Pemodelan Algoritma Huffman Encoding VHDL
2. Simulasi fungsionalitas dan akurasi algoritma sesuai dengan input data
3. Sintesis dan pengujian di platform FPGA guna evaluasi kerja perangkat keras dibandingkan perangkat lunak.

IMPLEMENTASI

HuffmanNode.vhd

- Menyatukan modul-modul utama:
- NodeGenerator
- NodeSorter
- NodeMerger
- HuffmanTranslator

Menghubungkan komponen lewat component instantiation dan penyambungan port (UUT)

Alur kerja dikendalikan clock

IMPLEMENTASI

ReadSort.vhd

- Membaca string dari sebuah file (Input.txt)
- Contoh : “abracadabra”
- Memecah string menjadi karakter-karakter uniknya (a, b, r, c, d)
- Menghitung kemunculan tiap karakter dalam string tersebut (a 5 / b 2 / r 2 / c 1 / d 1)
- Menuliskannya kembali ke dalam sebuah file (Output.txt)

IMPLEMENTASI

NodeGen.vhd

- Membaca Output.txt, yang berisi karakter hasil ReadSort
- Membuat node sebanyak karakter unik yang ada dalam string
- Abracadabra => 5 node (a, b, r, c, d)
- Node awal memiliki nilai karakter dan frekuensi

IMPLEMENTASI

NodeSort.vhd

- Menyortir tiap node menurut frekuensi mereka
- NodeSort membaca hasil dari nodeGen (dihubungkan dalam topmodule lewat instansiasi UUT)
- Sort descending
- Algoritma bubble sort
- abracadabra = (a 5 / b 2 / r 2 / c 1 / d 1)

IMPLEMENTASI

NodeMerger.vhd

- Menerima data dari nodeSort
- Mencari 2 node dengan frekuensi terendah
- Menggabungkannya jadi 1 node dengan data anak dan frekuensi kumulatif
- Mengulangi proses hingga tersisa 1 node root
- Node root memiliki data rekursif ke seluruh node yang ada dalam program
- Data node ditulis ke file HuffmanArray.txt

IMPLEMENTASI

NodeMerger.vhd

- abracadabra = (a 5 / b 2 / r 2 / c 1 / d 1)
- a 5 / b 2 / r 2 / cd 2
- a 5 / rcd 4 / b 2
- rcdb 6 / a 5
- arcdb 11 (root node - final)

IMPLEMENTASI

HuffmanTranslation.vhd

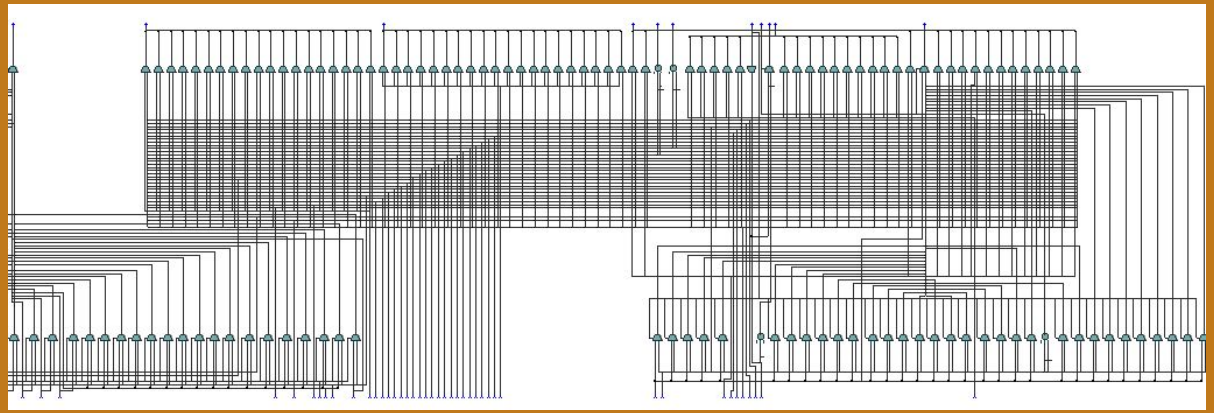
- Membaca data dari HuffmanArray dan menjadikannya array of nodes
- Membangun “tree” dari data pointer anak array
- Menghitung tinggi tree
- Mentraverse tree dan menguji path-path yang ada agar dapat menemukan translasi tiap node leaf / anak
- Menyimpan translasi dalam file
- Mengambil string asli dari Input dan menerjemahkannya / encode dengan data sebelumnya

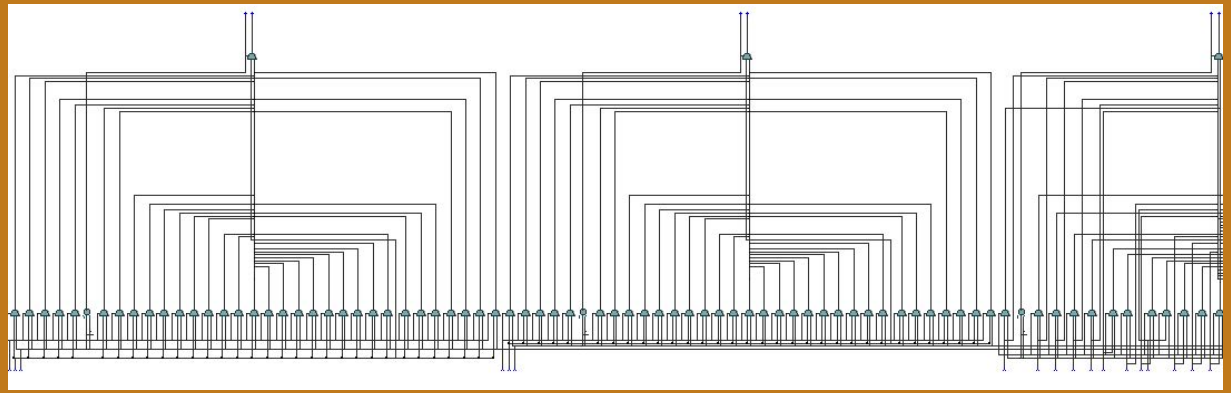
IMPLEMENTASI

HuffmanTranslation.vhd

- Membaca data dari HuffmanArray dan menjadikannya array of nodes
- Membangun “tree” dari data pointer anak array
- Menghitung tinggi tree
- Mentraverse tree dan menguji path-path yang ada agar dapat menemukan translasi tiap node leaf / anak
- Menyimpan translasi dalam file
- Mengambil string asli dari Input dan menerjemahkannya / encode dengan data sebelumnya

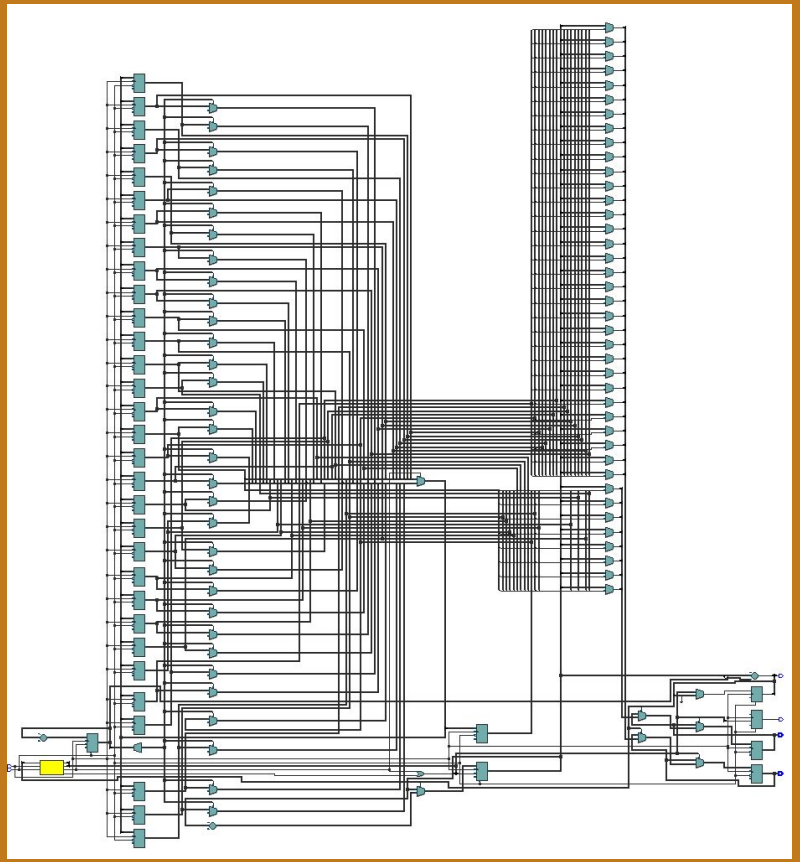
Schematics - ReadSort





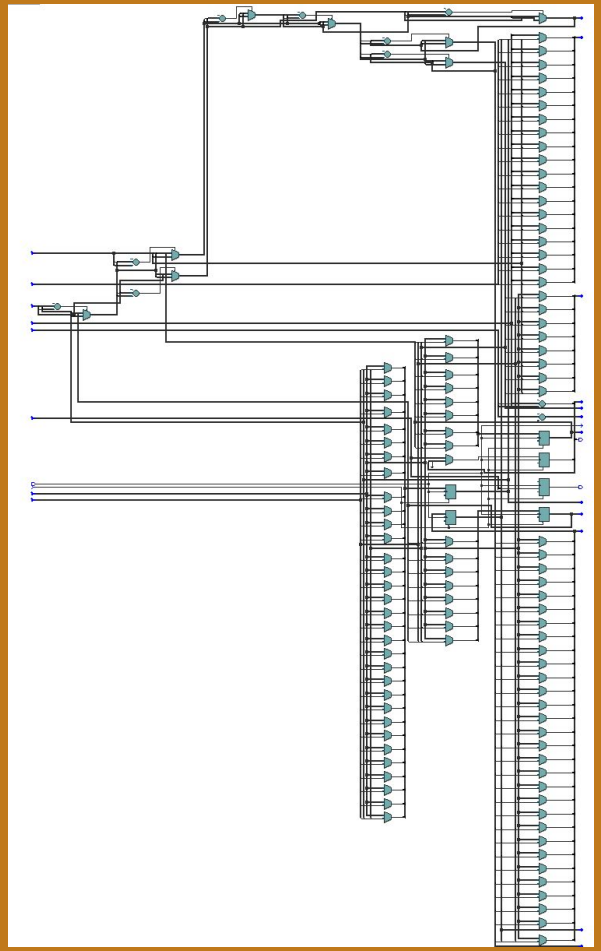
(Note : Line-line dynamic disubstitusi dengan nilai konstan agar dapat disintesis)

Schematics - NodeGenerator



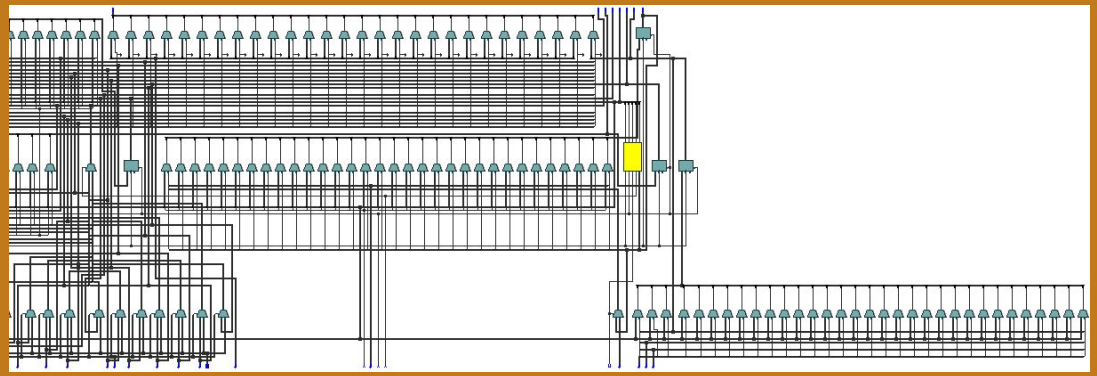
(Note : Line-line dynamic disubstitusi dengan nilai konstan ag
dapat disintesis)

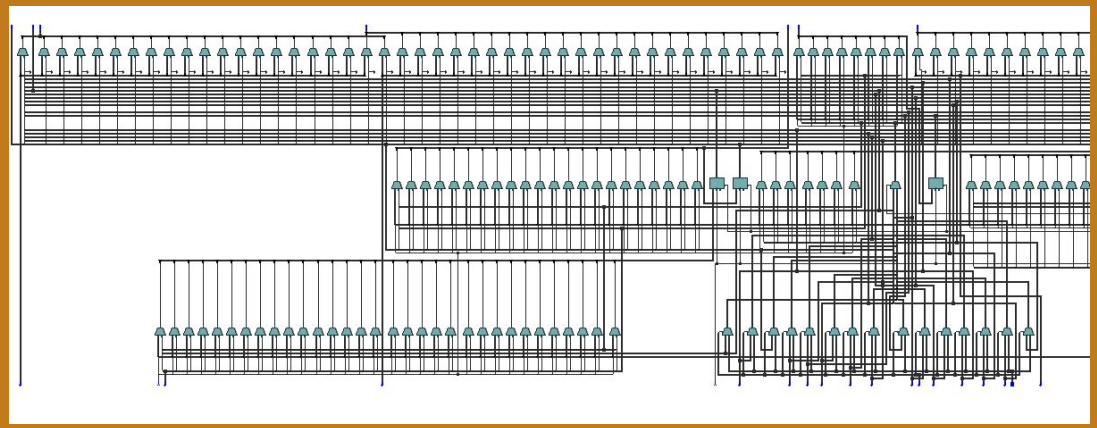
Schematics - NodeSorter



(Note : Line-line dynamic disubstitusi dengan nilai konstan agar dapat disintesis)

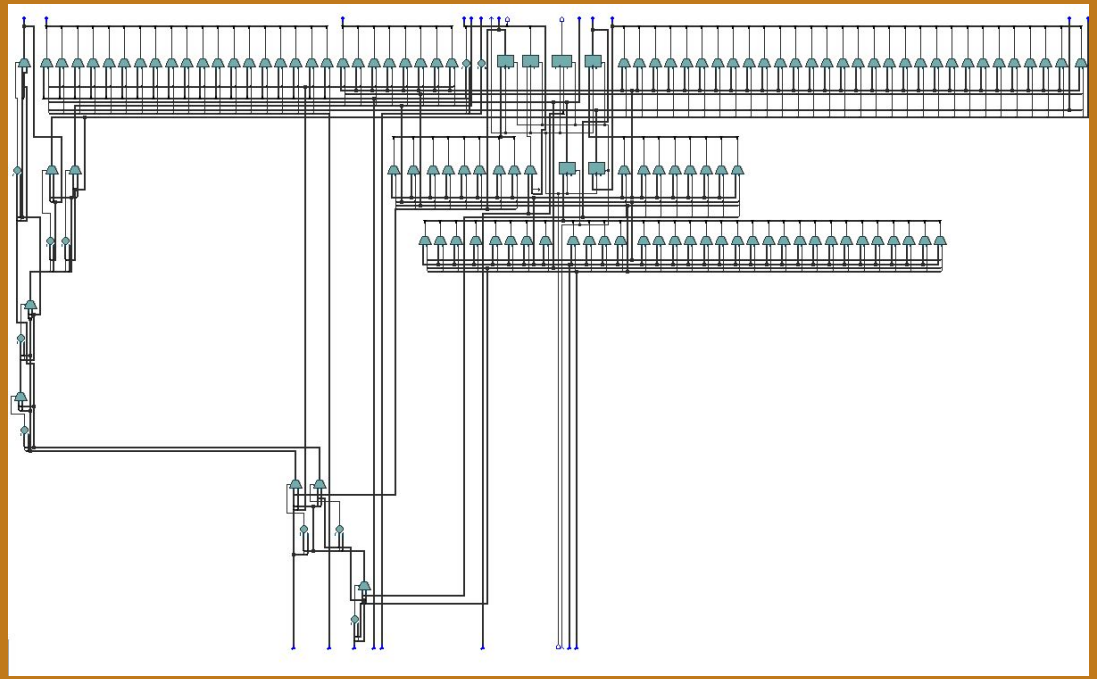
Schematics - NodeMerger





(Note : Line-line dynamic disubstitusi dengan nilai konstan agar dapat disintesis)

Schematics - HuffmanTranslator



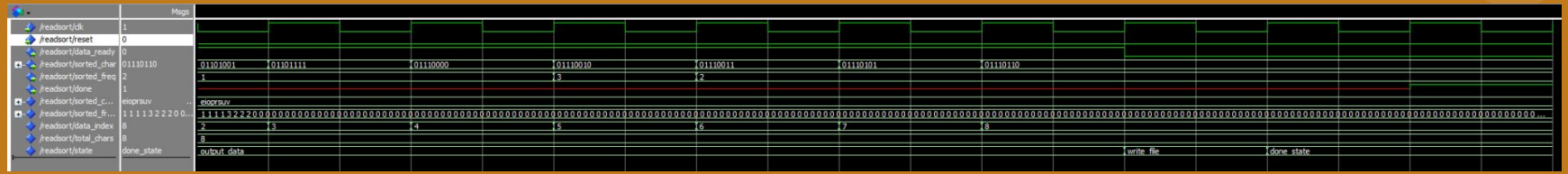
(Note : Line-line dynamic disubstitusi dengan nilai konstan agar dapat disintesis)

Pengujian Program

supersurvivor

Isi file Input = “supersurvivor”

Eksekusi ReadSort



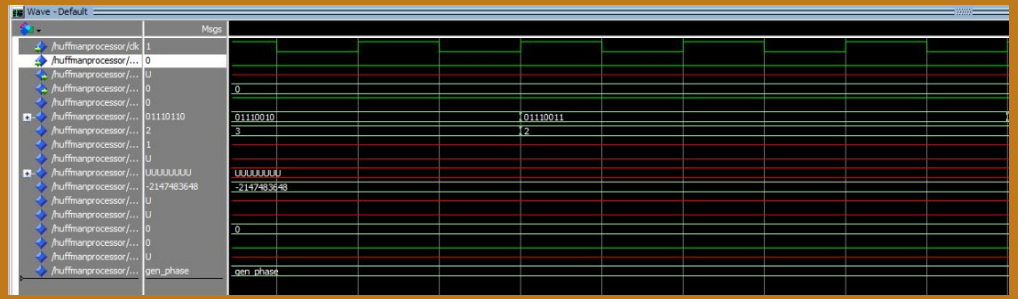
Hasil file Output

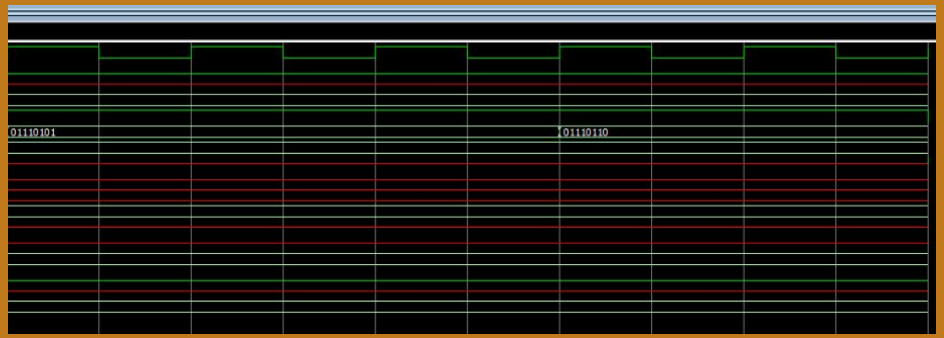
e 1
i 1
o 1
p 1
r 3
s 2
u 2
v 2

Pengujian Program

Output dipindahkan ke nodeGen
Eksekusi nodeGen (Cek duplikat -> Buat node -> Masuk array)

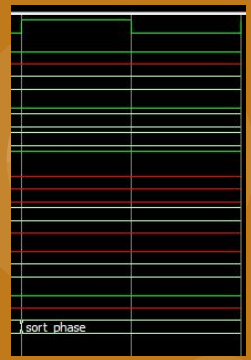
```
run
# ** Note: Character read: u, Frequency read: 2
#    Time: 2 ns  Iteration: 0  Instance: /huffmanprocessor/node_gen
# ** Note: Receiving new node - Char: s Freq: 2
#    Time: 2 ns  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Character already exists at index 5
#    Time: 2 ns  Iteration: 0  Instance: /huffmanprocessor/node_sort
run
# ** Note: Outputting node: Char = u, Freq = 2
#    Time: 2100 ps  Iteration: 0  Instance: /huffmanprocessor/node_gen
# ** Note: Receiving new node - Char: s Freq: 2
#    Time: 2100 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Character already exists at index 5
#    Time: 2100 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
run
# ** Note: Receiving new node - Char: u Freq: 2
#    Time: 2200 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Adding new node at index 6
#    Time: 2200 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
run
# ** Note: Character read: v, Frequency read: 2
#    Time: 2300 ps  Iteration: 0  Instance: /huffmanprocessor/node_gen
# ** Note: Receiving new node - Char: u Freq: 2
#    Time: 2300 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Character already exists at index 6
#    Time: 2300 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
run
# ** Note: Outputting node: Char = v, Freq = 2
#    Time: 2400 ps  Iteration: 0  Instance: /huffmanprocessor/node_gen
# ** Note: Receiving new node - Char: u Freq: 2
#    Time: 2400 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Character already exists at index 6
#    Time: 2400 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
run
# ** Note: Receiving new node - Char: v Freq: 2
#    Time: 2500 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Adding new node at index 7
#    Time: 2500 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
VSIM 281> run
# ** Note: Node generation completed. Total nodes: 8
```





Pengujian Program

Output disortir lewat nodeSort (Bubble).




```
# ** Note: Sort complete. Final array:
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Node[0]: Char=r Freq=3
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Node[1]: Char=s Freq=2
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Node[2]: Char=u Freq=2
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Node[3]: Char=v Freq=2
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Node[4]: Char=e Freq=1
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Node[5]: Char=i Freq=1
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Node[6]: Char=o Freq=1
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Node[7]: Char=p Freq=1
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
```

```
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Swapping nodes 6 and 7
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Swapping nodes 2 and 3
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Swapping nodes 3 and 4
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Swapping nodes 4 and 5
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Swapping nodes 5 and 6
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Swapping nodes 1 and 2
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Swapping nodes 2 and 3
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Swapping nodes 3 and 4
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Swapping nodes 4 and 5
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Swapping nodes 0 and 1
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Swapping nodes 1 and 2
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Swapping nodes 2 and 3
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Swapping nodes 3 and 4
#    Time: 2800 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
```

Pengujian Program

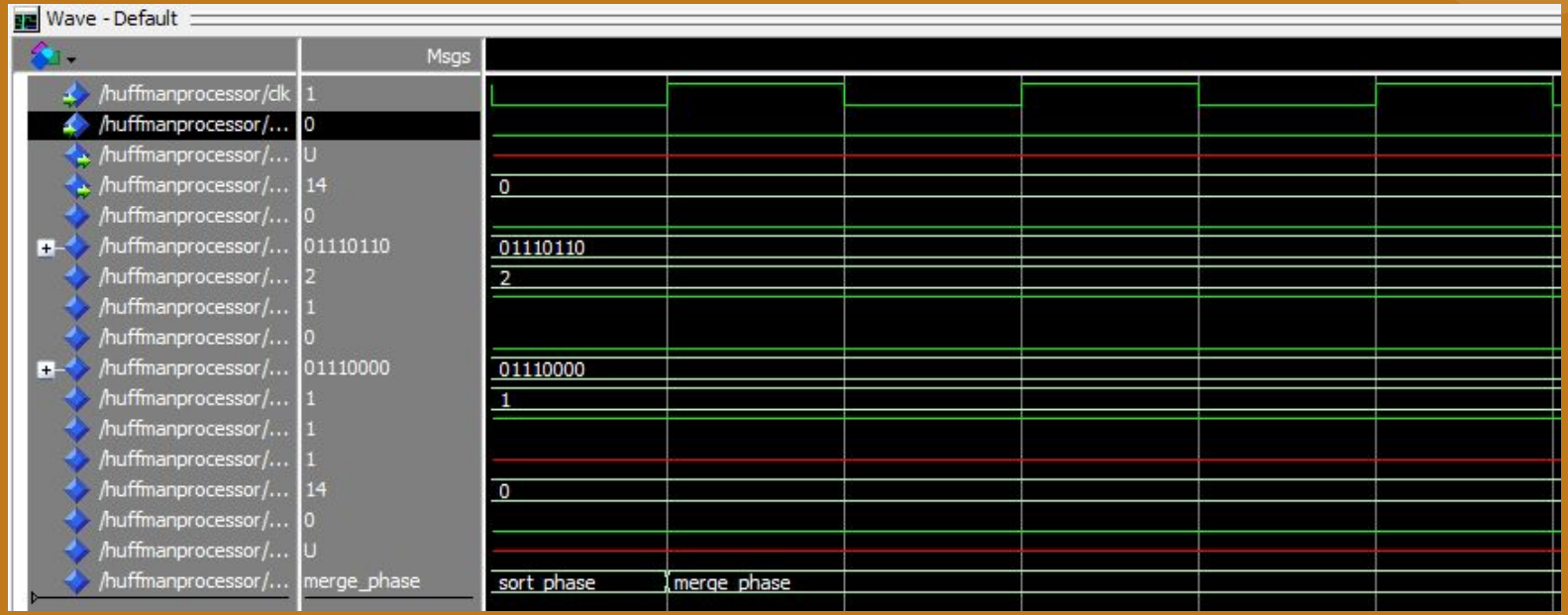
Array dari node tadi dipindahkan ke modul nodeMerger. Setelah receive, node awal diproses (leaf), lalu mulai siklus merge -> sort -> merge

```
# ** Note: Node generation completed. Total nodes: 8
#    Time: 4 ns  Iteration: 0  Instance: /huffmanprocessor/node_gen
# ** Note: Sorting process complete
#    Time: 4 ns  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: ----------------------------------------
#    Time: 4 ns  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Initial Sort:
#    Time: 4 ns  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Array:
#    Time: 4 ns  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 0: Node 4 (Char = e, Freq = 1, Leaf = true)
#    Time: 4 ns  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 1: Node 5 (Char = i, Freq = 1, Leaf = true)
#    Time: 4 ns  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 2: Node 6 (Char = o, Freq = 1, Leaf = true)
#    Time: 4 ns  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 3: Node 7 (Char = p, Freq = 1, Leaf = true)
#    Time: 4 ns  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 4: Node 1 (Char = s, Freq = 2, Leaf = true)
#    Time: 4 ns  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 5: Node 2 (Char = u, Freq = 2, Leaf = true)
#    Time: 4 ns  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 6: Node 3 (Char = v, Freq = 2, Leaf = true)
#    Time: 4 ns  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 7: Node 0 (Char = r, Freq = 3, Leaf = true)
#    Time: 4 ns  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: ----------------------------------------
#    Time: 4 ns  Iteration: 0  Instance: /huffmanprocessor/node_merge
```

```
# ** Note: Node generation completed. Total nodes: 8
#    Time: 4500 ps  Iteration: 0  Instance: /huffmanprocessor/node_gen
# ** Note: Sorting process complete
#    Time: 4500 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Merging: Node 9 (Freq=2) and Node 0 (Freq=3) -> New Node 12 (Total Freq=5, Left=9, Right=0)
#    Time: 4500 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: After merge and sort:
#    Time: 4500 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 0: Node 10 (Freq = 4)
#    Time: 4500 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 1: Node 11 (Freq = 4)
#    Time: 4500 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 2: Node 12 (Freq = 5)
#    Time: 4500 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
run
# ** Note: Node generation completed. Total nodes: 8
#    Time: 4600 ps  Iteration: 0  Instance: /huffmanprocessor/node_gen
# ** Note: Sorting process complete
#    Time: 4600 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Merging: Node 10 (Freq=4) and Node 11 (Freq=4) -> New Node 13 (Total Freq=8, Left=10, Right=11)
#    Time: 4600 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: After merge and sort:
#    Time: 4600 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 0: Node 12 (Freq = 5)
#    Time: 4600 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 1: Node 13 (Freq = 8)
#    Time: 4600 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
run
# ** Note: Node generation completed. Total nodes: 8
#    Time: 4700 ps  Iteration: 0  Instance: /huffmanprocessor/node_gen
# ** Note: Sorting process complete
#    Time: 4700 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Merging: Node 12 (Freq=5) and Node 13 (Freq=8) -> New Node 14 (Total Freq=13, Left=12, Right=13)
#    Time: 4700 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: After merge and sort:
#    Time: 4700 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 0: Node 14 (Freq = 13)
#    Time: 4700 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
```

Pengujian Program

Wave Diagram



Pengujian Program

Contoh -> supersurvivor

r 3	r 3	r 3	opei 4	opei 4	rs 5	rs 5	**opeiuvrs**
s 2	s 2	s 2	r 3	r 3	opei 4	opeiuv 8	**13 (root)**
u 2	u 2	u 2	s 2	s 2	uv 4		
v 2	v 2	v 2	u 2	uv 4			
e 1	op 2	op 2	v 2				
i 1	e 1	ei 2					
o 1	i 1						
p 1							

Pengujian Program

Saat merging, tiap node yang dimerge diberikan data mengenai anak mereka (node kanan dan kiri). Ini dilakukan agar program dapat mengetahui arah traverse untuk mencari translasi nanti.

Data index awal disimpan dalam array yang berbeda, sehingga tidak tergeser sorting

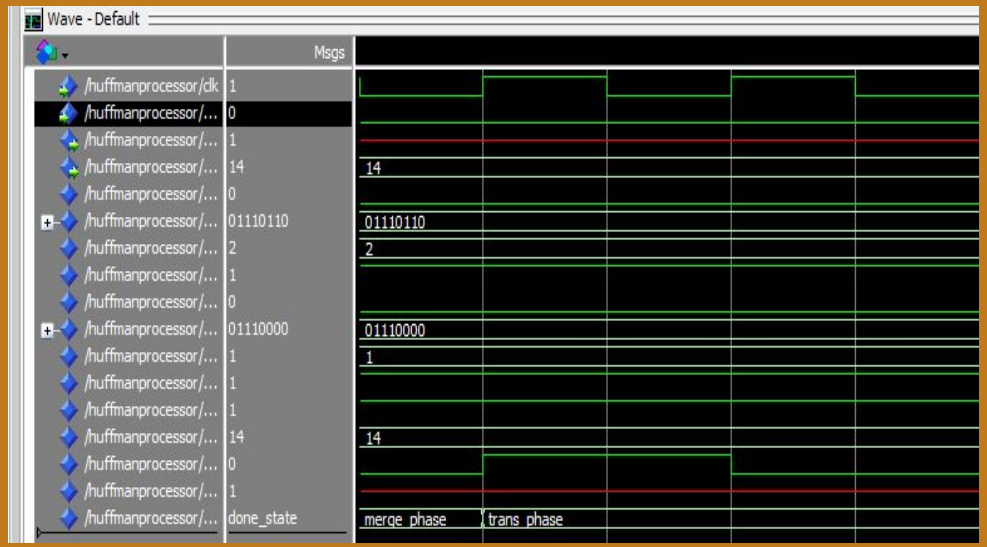
Status leaf/bukan juga disimpan, dan node yang sudah dimerge tak bisa merge lagi. Data node ini dituliskan ke file “HuffmanArray”

```
0,r,3,-1,-1,true
1,s,2,-1,-1,true
2,u,2,-1,-1,true
3,v,2,-1,-1,true
4,e,1,-1,-1,true
5,i,1,-1,-1,true
6,o,1,-1,-1,true
7,p,1,-1,-1,true
8,-,2,4,5,false
9,-,2,6,7,false
10,-,4,1,2,false
11,-,4,3,8,false
12,-,5,9,0,false
13,-,8,10,11,false
14,-,13,12,13,false
```

```
# ** Note: Node generation completed. Total nodes: 8
#    Time: 4500 ps  Iteration: 0  Instance: /huffmanprocessor/node_gen
# ** Note: Sorting process complete
#    Time: 4500 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Merging: Node 9 (Freq=2) and Node 0 (Freq=3) -> New Node 12 (Total Freq=5, Left=9, Right=0)
#    Time: 4500 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: After merge and sort:
#    Time: 4500 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 0: Node 10 (Freq = 4)
#    Time: 4500 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 1: Node 11 (Freq = 4)
#    Time: 4500 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 2: Node 12 (Freq = 5)
#    Time: 4500 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
run
# ** Note: Node generation completed. Total nodes: 8
#    Time: 4600 ps  Iteration: 0  Instance: /huffmanprocessor/node_gen
# ** Note: Sorting process complete
#    Time: 4600 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Merging: Node 10 (Freq=4) and Node 11 (Freq=4) -> New Node 13 (Total Freq=8, Left=10, Right=11)
#    Time: 4600 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: After merge and sort:
#    Time: 4600 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 0: Node 12 (Freq = 5)
#    Time: 4600 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 1: Node 13 (Freq = 8)
#    Time: 4600 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
run
# ** Note: Node generation completed. Total nodes: 8
#    Time: 4700 ps  Iteration: 0  Instance: /huffmanprocessor/node_gen
# ** Note: Sorting process complete
#    Time: 4700 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Merging: Node 12 (Freq=5) and Node 13 (Freq=8) -> New Node 14 (Total Freq=13, Left=12, Right=13)
#    Time: 4700 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: After merge and sort:
#    Time: 4700 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
# ** Note: Index 0: Node 14 (Freq = 13)
#    Time: 4700 ps  Iteration: 0  Instance: /huffmanprocessor/node_merge
```

Pengujian Program

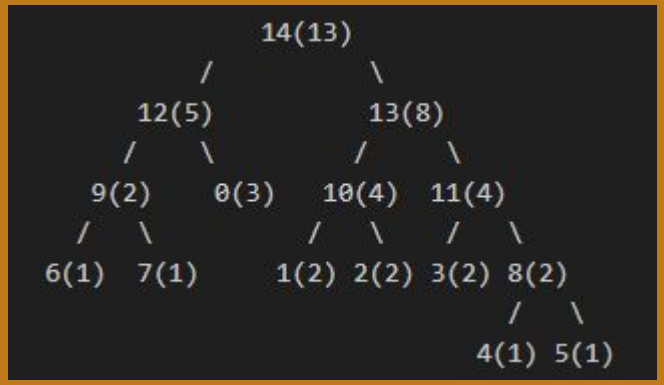
Program kemudian melakukan proses translasi. Data node dari HuffmanArray diambil dan diinisasi jadi array. Program kemudian membuat sebuah tree dari data ini




```
# ** Note: Node generation completed. Total nodes: 8
#    Time: 5300 ps  Iteration: 0  Instance: /huffmanprocessor/node_gen
# ** Note: Sorting process complete
#    Time: 5300 ps  Iteration: 0  Instance: /huffmanprocessor/node_sort
# ** Note: Found translation for o at depth 3
#    Time: 5300 ps  Iteration: 0  Instance: /huffmanprocessor/translator
# ** Note: Found translation for p at depth 3
#    Time: 5300 ps  Iteration: 0  Instance: /huffmanprocessor/translator
# ** Note: Found translation for r at depth 2
#    Time: 5300 ps  Iteration: 0  Instance: /huffmanprocessor/translator
# ** Note: Found translation for s at depth 3
#    Time: 5300 ps  Iteration: 0  Instance: /huffmanprocessor/translator
# ** Note: Found translation for u at depth 3
#    Time: 5300 ps  Iteration: 0  Instance: /huffmanprocessor/translator
# ** Note: Found translation for v at depth 3
#    Time: 5300 ps  Iteration: 0  Instance: /huffmanprocessor/translator
# ** Note: Found translation for e at depth 4
#    Time: 5300 ps  Iteration: 0  Instance: /huffmanprocessor/translator
# ** Note: Found translation for i at depth 4
#    Time: 5300 ps  Iteration: 0  Instance: /huffmanprocessor/translator
```

Pengujian Program

Program kemudian melakukan proses translasi. Data node dari HuffmanArray diambil dan diinisasi jadi array. Program kemudian membuat sebuah tree dari data ini



Algoritma kemudian menghitung tinggi tree, dan mencoba seluruh path yang tersedia untuk mencari tiap karakter / node leaf. Gerak ke kanan + 1 dan kiri + 0.

Pengujian Program

Setelah mendapatkan seluruh data translasi, program menyimpan mappingnya ke file output. Program juga menggunakan data tadi dengan mengambil string awal dari Input.txt dan menerjemahkannya dengan data yang didapatkan (encoding)

```
o: 000
p: 001
r: 01
s: 100
u: 101
v: 110
e: 1110
i: 1111
Original text translation: supersurvivor
Binary translation: 10010100111100110010101110111111000001
```

Dari sini, kita dapat mengencode “supersurvivor” tanpa loss menggunakan map yang telah dibangun via tree dan node huffman.

KESIMPULAN

Algoritma Huffman berhasil diterapkan untuk mengKompresi string secara efisien dengan menghasilkan representasi biner yang lebih hemat. Proses dimulai dengan menghitung frekuensi setiap karakter unik dalam string, membangun pohon Huffman dari node-node berdasarkan frekuensi tersebut, dan memberikan kode biner ke setiap karakter berdasarkan posisi di pohon. String asli kemudian diterjemahkan menggunakan kode Huffman ini, menghasilkan representasi biner yang lebih ringkas.

Program juga memastikan hasil translasi biner dan kode karakter ditulis ke file output untuk dokumentasi. Metode ini membuktikan efisiensinya dalam mengurangi ukuran data tanpa kehilangan informasi, yang sangat berguna untuk kompresi tanpa hilang data.

Terima Kasih!