Alunos: Chrystian de Sousa Guth 10103131
Lucas Pereira Da Silva 10100754
Renan Oliveira Netto 10103126

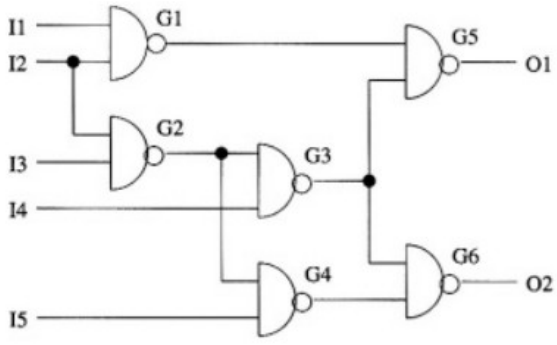
Analise estática de tempo para circuitos combinacionais

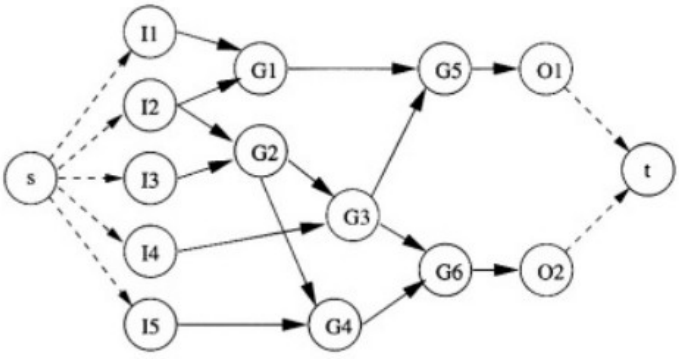
Problema

- Encontrar o caminho crítico num circuito combinacional calculando os tempos de chegada de subida e descida de cada porta lógica, e dado um tempo mínimo de percorrimento desse circuito, encontrar as folgas em cada porta lógica. Assim algumas otimizações podem ser realizadas.

Representação

- Grafo Temporal ***G = (V,A)*** .
- V = {**x** | (**x**,**ad**,**as**) é uma *porta lógica* ou *entrada* ou *saída* do circuito, tem atraso de descida = **ad** e atraso de subida = **as**} .
- A = {(**v**,**w**) é uma conexão da saída de **v** para uma entrada de **w**} .
- São adicionados dois vértices fantasmas, um **fonte** e um **terminal**.
- São adicionadas arestas fantasmas (**s**,**w**) para cada entrada **w** e arestas fantasmas (**v**,**t**) para cada saída **v**.





- Esta construção geralmente resulta num grafo direcionado sem ciclos, porque circuitos combinacionais não tem ciclos.
- Um circuito combinacional com elementos sequenciais deve ser representado como um conjunto de blocos combinacionais entre os *latches* e *flipflops*.

Cálculo de atraso para um bloco lógico combinacional

- Método PERT-CPM.
- Para simplificar, o atraso de n pinos de entrada até um pino de saída em uma porta lógica são idênticos e que todas as portas são inversoras (NOR, NAND, INV), assim, uma transição de subida na entrada, gera uma transição de descida na saída.
- Cada vértice tem dois valores iniciais associados a ele: as (atraso de subida) / ad (atraso de descida).
- Inicialmente assumimos que as entradas primárias tem **as** = **ad** = 0.

Algoritmo CPM

Considerações Iniciais

- Para auxiliar o processo, teremos um arranjo associativo para armazenar o número de vezes que um determinado vértice foi visitado.
- Dispomos também de uma estrutura do tipo fila, a qual tem complexidade *O(1)* para as operações **enfileirar**, **desenfileirar**, **frente**, **vazia**.

Algoritmo CPM

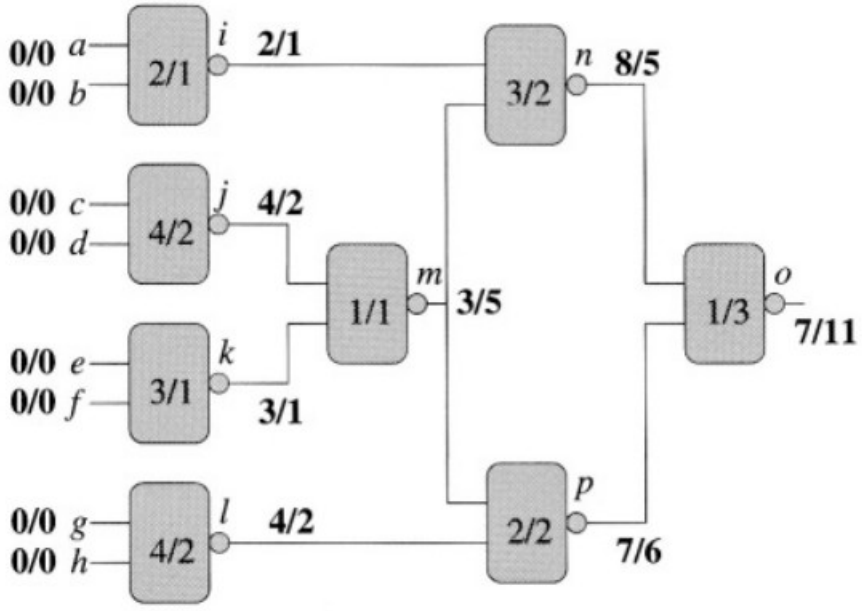
```
fila = 0
para cada vértice i ∈ V
      númeroDeEntradasVisitadas[i] = 0

para cada entrada primária i
      para cada aresta de saída j de i
            destino = j → destino
            númeroDeEntradasVisitadas[destino]++
            se númeroDeEntradasVisitadas[destino] == destino → numeroDeEntradas
                  fila → enfileirar(destino)

enquanto ¬fila → estáVazia
      g = fila → frente
      fila → desenfileirar
      computarAtraso(g);
      para cada aresta de saída j de g
            destino = j → destino
            númeroDeEntradasVisitadas[destino]++
            se númeroDeEntradasVisitadas[destino] == destino → numeroDeEntradas
                  fila → enfileirar(destino)
FIM
```

Algoritmo computarAtraso(Vertice : v)

```
maior = 0
para cada aresta de entrada j de v
      origem = j → origem
```

```
      se origem → tcs > maior
            maior = origem → as
v → tcd = maior + v → ad
maior = 0
para cada aresta de entrada j de v
origem = j → origem
      se origem → tcd > maior
            maior = origem → ad
v → tcs = maior + v → as
FIM
```

- Com a execução do algoritmo do Caminho Crítico, cada vértice do grafo terá associado a ele um valor de **tempo de chegada de descida** e **tempo de chegada de subida**.



- Aplicando o algoritmo Calcula Folgas (passando um tempo requerido como argumento) serão calculadas as folgas em cada porta lógica, assim possibilitando a substituição de alguma a fim de reduzir o atraso do circuito (ou aumentar, caso tenha rodado em menos tempo do que o requerido).

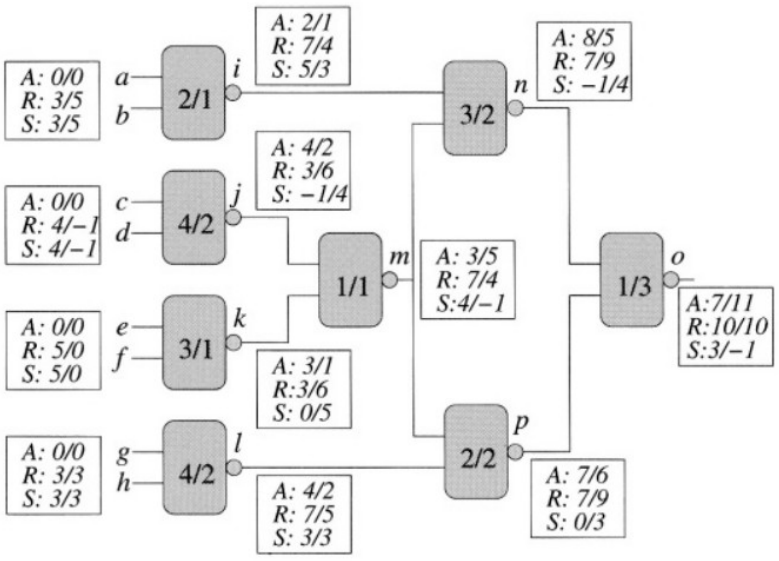
Algoritmo Calcula Folgas(TempoRequerido : r)

```
fila = 0
terminal → rs = rd = r
fila → enfileirar(terminal)

enquanto ¬fila → estáVazia
      g = fila → frente
      fila → desenfileirar
      computarFolgas(g);
      para cada aresta de entrada j de g
            origem = j → origem
            fila → enfileirar(origem)
FIM
```

Algoritmo computarFolgas(Vertice : v)

```
menor = ∞
para cada aresta de saída j de v
      destino = j → destino
      se destino → rd - destino → ad < menor
            menor =  destino → rd - destino → ad
v → rs = menor
menor = ∞
para cada aresta de saída j de v
      destino = j → destino
      se destino → rs - destino → as < menor
            menor =  rs - destino → as
v → rd = menor
FIM
```

- No fim da execução do algoritmo teremos todas as folgas. As portas que tiverem folgas negativas precisam ser trocadas por portas mais rápidas, pois o tempo requerido foi violado. As portas que tem folgas positivas, podem ser trocadas por portas mais lentas (caso a troca não viole o tempo requerido), assim a potência dissipada seria otimizada, já que geralmente quanto maior o atraso de uma porta, menor a potência necessária.


A: 0/0
R: 3/5
S: 3/5
a
b
2/1
i
A: 2/1
R: 7/4
S: 5/3
A: 0/0
R: 4/−1
S: 4/−1
c
d
4/2
j
A: 4/2
R: 3/6
S: −1/4
A: 0/0
R: 5/0
S: 5/0
e
f
3/1
k
A: 3/1
R:3/6
S: 0/5
A: 0/0
R: 3/3
S: 3/3
g
h
4/2
l
A: 4/2
R: 7/5
S: 3/3
1/1
m
A: 3/5
R: 7/4
S:4/−1
3/2
n
A: 8/5
R: 7/9
S: −1/4
2/2
p
A: 7/6
R: 7/9
S: 0/3
1/3
o
A:7/11
R:10/10
S:3/−1