TALLER 1

Carlos Fernando Padilla Mesa

202059962

Universidad del Valle

Facultad de ingeniería de sistemas

Tuluá, Valle del Cauca

19 de septiembre del 2024

Informe sobre la Construcción de un TAD para Circuitos Digitales LCD

Introducción

El objetivo del presente informe es describir la implementación de un Tipo Abstracto de Datos (TAD) para la simulación de circuitos digitales LCD en el lenguaje de programación Racket.

Para la construcción del TAD, se utilizaron cuatro enfoques diferentes, cada uno implementado en un archivo separado. A continuación, se detalla la estructura del código y las diferencias entre las implementaciones.

Implementación

1. Representación mediante Listas (representacion-listas.rkt)

En este archivo, los circuitos y sus chips se representaron usando listas de Racket. Cada compuerta lógica se modela como una lista donde los elementos son los puertos de entrada y salida, junto con su tipo de operación lógica. Este enfoque es simple, pero puede resultar menos eficiente al aumentar el número de chips o compuertas.

Ejecución

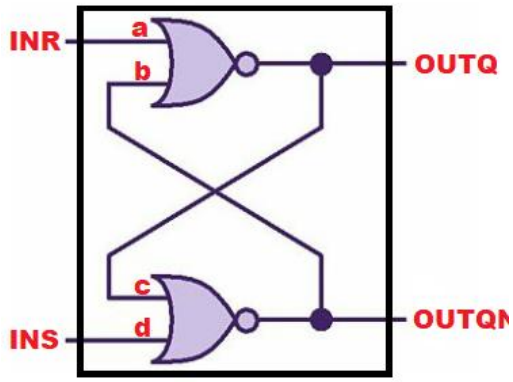

Ilustración 1: Ejemplo visual

```
(define latch
    '(comp-chip
        (INR INS)
        (OUTQ OUTQN)
        (complex-circuit
            (simple-circuit (a b) (c) (prim-chip (chip-nor)))
            (list
                (simple-circuit (c d) (b) (prim-chip (chip-nor)))
            )
            (a b c d)
            (b c)
        )
    )
)
```

Ilustración 2: Ejemplo sintaxis concreta

2. Representación mediante Procedimientos (representacion-procedimientos.rkt)

En esta implementación, las compuertas lógicas y los chips se representaron como procedimientos que reciben las entradas y devuelven los resultados correspondientes en los puertos de salida. Esta abstracción permite una mayor modularidad y flexibilidad, ya que los chips pueden ser construidos y manipulados como funciones de alto orden.

Ejecución

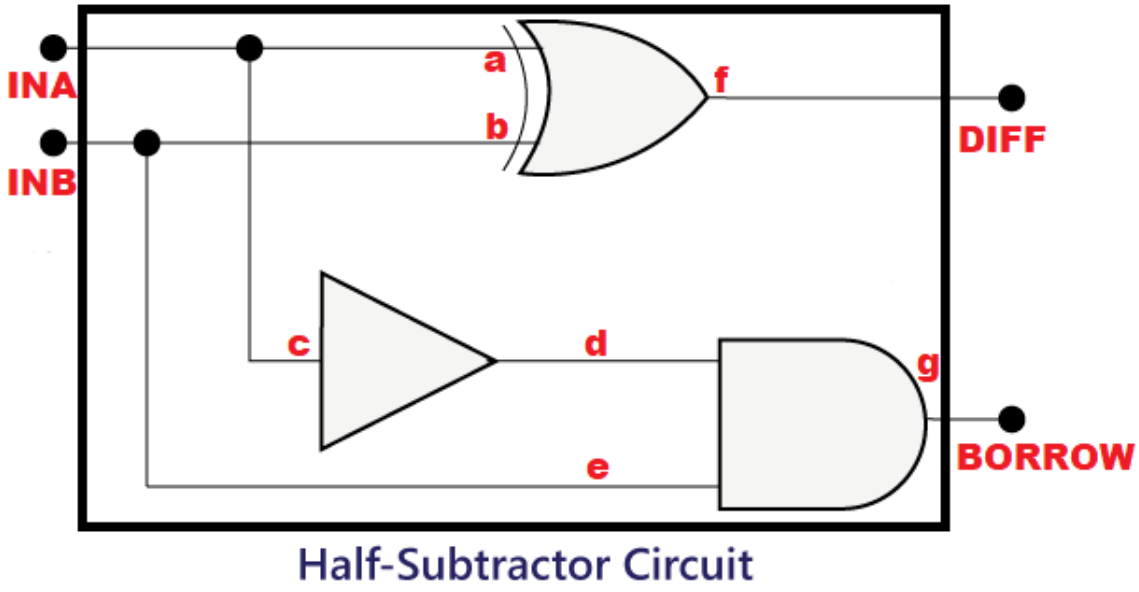

Ilustración 3: Ejemplo visual

```
(define half-substractor
   '(comp-chip
        (INA INB)
        (DIFF BORROW)
        (complex-circuit
            (simple-circuit (a b) (f) (prim-chip (chip-xor)))
            (list
                (simple-circuit (c) (d) (prim-chip (chip-not)))
                (simple-circuit (d e) (g) (prim-chip (chip-and)))
            )
            (a b c e)
            (f g)
        )
    )
)
```

Ilustración 4: Ejemplo sintaxis concreta

3. Representación mediante Datatypes (representacion-datatype.rkt)

En este archivo, se usaron tipos de datos personalizados para definir la estructura de los chips y las compuertas lógicas. Esta implementación permite un mejor control sobre el tipo de datos, evitando errores de tipo en la simulación de los circuitos. Además, facilita la validación de la información que pasa por los chips.

Ejecución

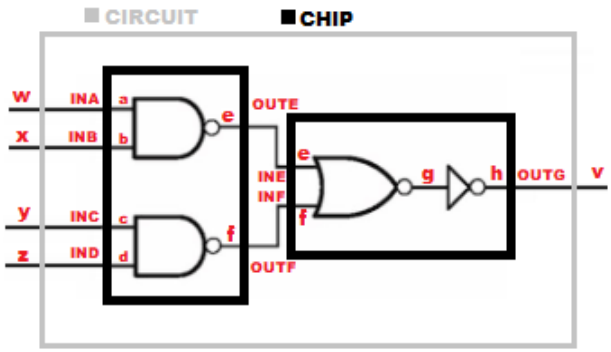

Ilustración 5: Ejemplo visual

```
(define circuito1
    '(complex-circuit
        (simple-circuit
            (w x y z)
            (e f)
            (comp-chip
                (INA INB INC IND)
                (OUTE OUTF)
                (complex-circuit
                    (simple-circuit (a b) (e) (prim-chip (chip-nand)))
                    (list
                        (simple-circuit (c d) (f) (prim-chip (chip-nand)))
                    )
                    (a b c d)
                    (e f)
                )
            )
        )
        (list
            (simple-circuit
                (e f)
                (g)
                (comp-chip
                    (INE INF)
                    (OUTG)
                    (complex-circuit
                        (simple-circuit (e f) (g) (prim-chip (chip-nor)))
                        (list
                            (simple-circuit (g) (h) (prim-chip (chip-not)))
                        )
                        (e f)
                        (h)
                    )
                )
            )
        )
        (w x y z)
        (v)
    )
)
```

Ilustración 6: Ejemplo sintaxis abstracta

4. Parser y Unparser (parser-unparser.rkt)

El archivo parser-unparser.rkt, el “parser” se encargó de convertir la sintaxis concreta a sintaxis abstracta, esto permite que algún tipo de compilador lea fácilmente el código que nosotros le brindamos, y luego nos pueda arrojar un resultado utilizando el ”unparser” implementado para darnos la sintaxis concreta.

Ejecución

A continuación, se verán una serie de imágenes donde se ejecuta el parser y el unparser a dos tipos de ejemplos, un circuito y un chip.

```
"parser-unparser.rkt"> circuito1
'(complex-circuit
  (simple-circuit
   (w x y z)
   (e f)
   (comp-chip
    (INA INB INC IND)
    (OUTE OUTF)
    (complex-circuit
     (simple-circuit (a b) (e) (prim-chip (chip-nand)))
     (list (simple-circuit (c d) (f) (prim-chip (chip-nand))))
     (a b c d)
     (e f))))
  (list
   (simple-circuit
    (e f)
    (g)
    (comp-chip
     (INE INF)
     (OUTG)
     (complex-circuit
      (simple-circuit (e f) (g) (prim-chip (chip-nor)))
      (list (simple-circuit (g) (h) (prim-chip (chip-not))))
      (e f)
      (h)))))
  (w x y z)
  (v))
```

Ilustración 7: Circuito1

```
"parser-unparser.rkt"> (parser circuito1)
(complex-circuit
 (simple-circuit
  '(w x y z)
  '(e f)
  (comp-chip
   '(INA INB INC IND)
   '(OUTE OUTF)
   (complex-circuit
    (simple-circuit '(a b) '(e) (prim-chip (chip-nand)))
    (list (simple-circuit '(c d) '(f) (prim-chip (chip-nand))))
    '(a b c d)
    '(e f))))
 (list
  (simple-circuit
   '(e f)
   '(g)
   (comp-chip
    '(INE INF)
    '(OUTG)
    (complex-circuit
     (simple-circuit '(e f) '(g) (prim-chip (chip-nor)))
     (list (simple-circuit '(g) '(h) (prim-chip (chip-not))))
     '(e f)
     '(h)))))
 '(w x y z)
 '(v))
```

Ilustración 8: Parser a circuito1

```
"parser-unparser.rkt"> (unparser (parser circuito1))
'(complex-circuit
  (simple-circuit
   (w x y z)
   (e f)
   (comp-chip
    (INA INB INC IND)
    (OUTE OUTF)
    (complex-circuit
     (simple-circuit (a b) (e) (prim-chip (chip-nand)))
     ((simple-circuit (c d) (f) (prim-chip (chip-nand))))
     (a b c d)
     (e f))))
  ((simple-circuit
    (e f)
    (g)
    (comp-chip
     (INE INF)
     (OUTG)
     (complex-circuit
      (simple-circuit (e f) (g) (prim-chip (chip-nor)))
      ((simple-circuit (g) (h) (prim-chip (chip-not))))
      (e f)
      (h)))))
  (w x y z)
  (v))
```

Ilustración 9: Unparser a circuito1

Ejemplo a un chip:

```
"parser-unparser.rkt"> chip1
'(comp-chip
  (INA INB INC)
  (OUTA)
  (complex-circuit
   (simple-circuit (a b) (d) (prim-chip (chip-and)))
   (list
    (simple-circuit (a c) (e) (prim-chip (chip-and)))
    (simple-circuit (b) (f) (prim-chip (chip-not)))
    (simple-circuit (c) (g) (prim-chip (chip-not)))
    (simple-circuit (e) (h) (prim-chip (chip-not)))
    (simple-circuit (f g) (i) (prim-chip (chip-not)))
    (simple-circuit (d h i) (p) (prim-chip (chip-not))))
   (a b c)
   (p)))
```

Ilustración 10: Chip1

```
"parser-unparser.rkt"> (parser chip1)
(comp-chip
 '(INA INB INC)
 '(OUTA)
 (complex-circuit
  (simple-circuit '(a b) '(d) (prim-chip (chip-and)))
  (list
   (simple-circuit '(a c) '(e) (prim-chip (chip-and)))
   (simple-circuit '(b) '(f) (prim-chip (chip-not)))
   (simple-circuit '(c) '(g) (prim-chip (chip-not)))
   (simple-circuit '(e) '(h) (prim-chip (chip-not)))
   (simple-circuit '(f g) '(i) (prim-chip (chip-not)))
   (simple-circuit '(d h i) '(p) (prim-chip (chip-not))))
  '(a b c)
  '(p)))
```

Ilustración 11: Parser a chip1

```
"parser-unparser.rkt"> (unparser-chip (parser chip1))
'(comp-chip
  (INA INB INC)
  (OUTA)
  (complex-circuit
   (simple-circuit (a b) (d) (prim-chip (chip-and)))
   ((simple-circuit (a c) (e) (prim-chip (chip-and)))
    (simple-circuit (b) (f) (prim-chip (chip-not)))
    (simple-circuit (c) (g) (prim-chip (chip-not)))
    (simple-circuit (e) (h) (prim-chip (chip-not)))
    (simple-circuit (f g) (i) (prim-chip (chip-not)))
    (simple-circuit (d h i) (p) (prim-chip (chip-not))))
   (a b c)
   (p)))
```

Ilustración 12: Unparser a chip1

Conclusión

La construcción del TAD para circuitos digitales en Racket a través de múltiples enfoques permitió explorar distintas formas de modelar y simular la lógica digital. Cada implementación ofrece ventajas y desventajas en cuanto a simplicidad, flexibilidad y control sobre los datos. Este ejercicio permitió una mayor comprensión de la abstracción de circuitos digitales y su representación en lenguajes de programación.