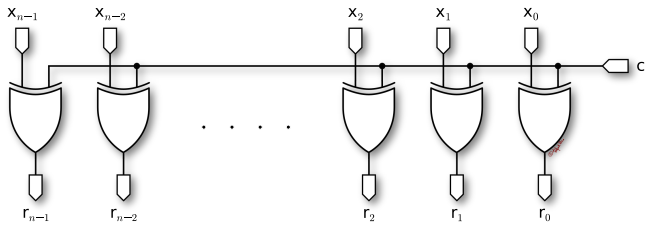
PROBLEMAS: La ruta de datos

1. Diseñe un operador inversor condicional para operandos de 8 bits.

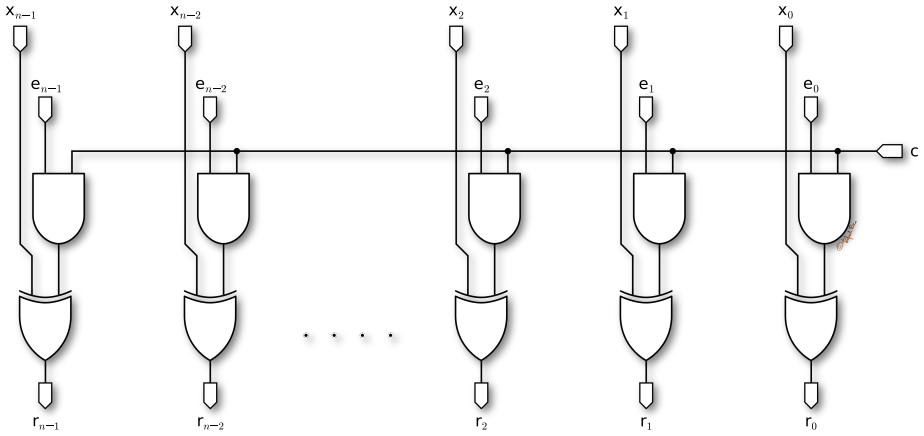
SOLUCIÓN:

Un inversor condicional es una batería de puertas XOR donde una de las entradas es el operando y la otra una señal de control común a todas las puertas. La siguiente figura muestra lo explicado:



Es un operador de bit (no hay propagación de señales entre pesos). Este operador también es el de cambio de signo en complemento a uno.

Puede ser interesante disponer de un inversor condicional por bloque y por bit individualmente. En el diseño de la figura siguiente podemos inhibir la inversión del conjunto con la señal c='0' pero también podemos activar o desactivar la inversión individualmente en cada bit con las señales e_i.

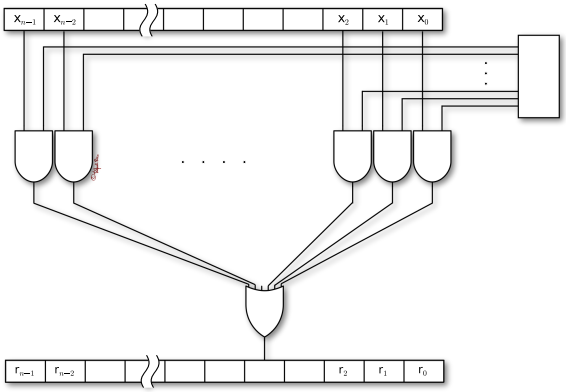


2. ¿Qué posibilidades tenemos a la hora de sintetizar un operador de desplazamiento utilizando álgebra de Boole? Realice un análisis comparativo en términos de coste, rendimiento y viabilidad de la implementación.

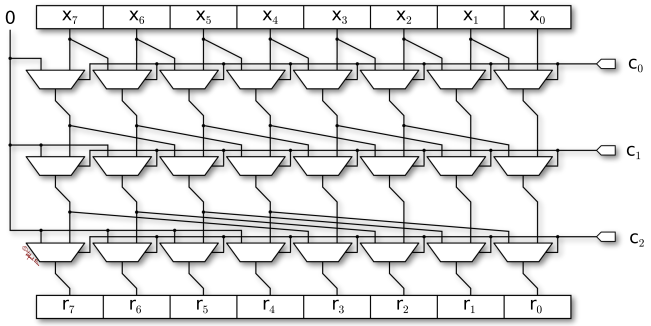
SOLUCIÓN:

Disponemos de operadores de desplazamiento sintetizados a partir del álgebra de Boole y también sintetizados en VLSI (lógica *pullup-pulldown* en CMOS). Respecto a los primeros tenemos los siguientes esquemas.

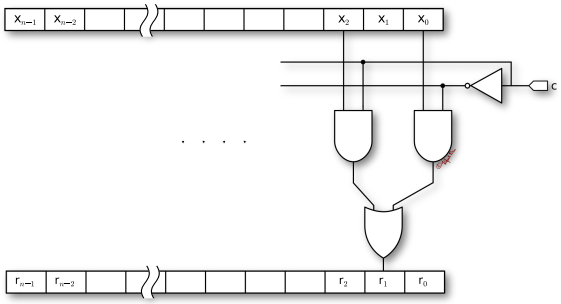
Desplazador de n bits cuya complejidad en retardo es $\mathcal{O}(cte)$ y en coste $\mathcal{O}(n^2)$. Imposible de implementar tal cual si n es grande debido a problemas de *fan-in* en la puerta OR.



Desplazador de 8 bits a la derecha constituido por multiplexores 2 a 1. Tiene una complejidad en retardo de $\mathcal{O}(log_2 n)$ y en coste $\mathcal{O}(nlog_2 n)$.



Desplazador de 1 bit a izquierda o derecha. Tiene una complejidad en retardo de $\mathcal{O}(cte)$ y en coste $\mathcal{O}(n)$. Si queremos hacer desplazamientos de más de 1 bit hay que usarlo iterativamente.

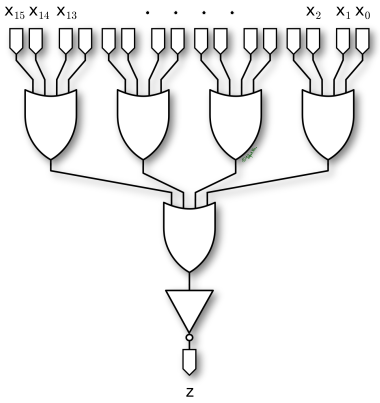


En cuanto a los operadores sintetizados en VLSI, tenemos el *barrel shifter*.

3. Un detector de cero es un operador que toma como entrada una ristra de n bits y devuelve un '1' si todos los bits son '0' o devuelve un '0' en caso contrario. ¿Qué posibilidades tenemos para sintetizarlo usando álgebra de Boole? Presente varias opciones y comente sus ventajas e inconvenientes.

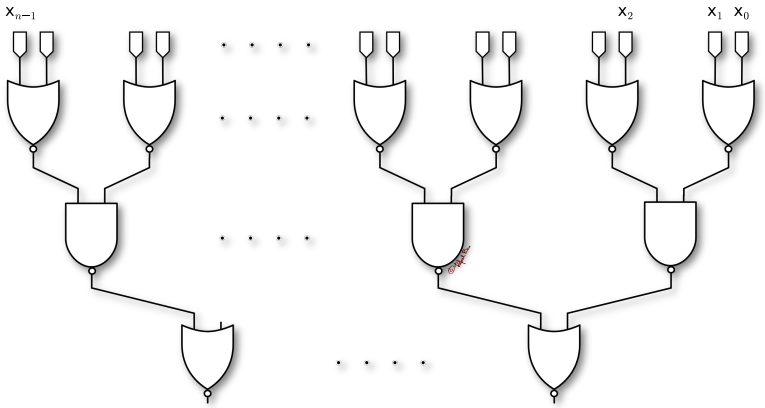
SOLUCIÓN:

Utilizando sínteis mediante álgebra de Boole tenemos dos clases de detectores. Los que operan en cascada analizando en paralelo todos los bits del operando y los que propagan el análisis bit a bit desde el LSB al MSB en serie. Respecto a los primeros tenemos:



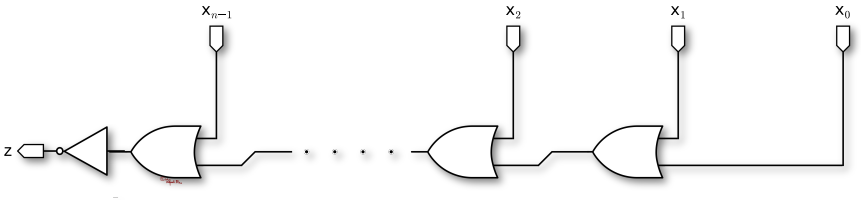
Como vemos, es importante disponer de puertas con un alto *fan-in* (f) ya que influye en el coste y en el retardo. Este circuito tiene complejidad en retardo de $\mathcal{O}(log_f n)$ y en coste $\mathcal{O}(\frac{n}{f-1})$.

También podemos implementarlo con puertas universales alternando filas de NAND y NOR:



La complejidad es la misma que en el caso anterior pero sabemos que las puertas universales son más rápidas ya que se implementan con menos bloques de conmutación.

Finalmente, el detector de cero basado en propagación es más barato pero más lento:



Su complejidad en retardo de $\mathcal{O}(n)$ y en coste $\mathcal{O}(n)$.

4. ¿Cómo se cambia el signo de un operando representado en exceso a 2^{n-1}? Demuestre formalmente su respuesta.

SOLUCIÓN:

Sea $\mathsf{r}_{EX2^{n-1},n}$ la representación del negativo de $\mathsf{x}_{EX2^{n-1},n}$. Debe cumplir entonces:

$$\mathsf{r}_{EX2^{n-1},n} = \mathit{EX2}^{n-1}\left(-\mathit{VAL\text{-}EX2}^{n-1}(\mathsf{x}_{EX2^{n-1},n}), n\right) = \mathit{BIN}\left(2^{n-1} - \mathit{VAL\text{-}EX2}^{n-1}(\mathsf{x}_{EX2^{n-1},n}), n\right)$$

$$\mathit{VAL\text{-}EX2}^{n-1}(\mathsf{x}_{EX2^{n-1},n}) = \sum_{i=0}^{n-2} x_i \cdot 2^i - \overline{x}_{n-1} \cdot 2^{n-1}$$

de donde:

$$\mathsf{r}_{EX2^{n-1},n} = BIN\left(2^{n-1} - \sum_{i=0}^{n-2} x_i \cdot 2^i - \overline{x}_{n-1} \cdot 2^{n-1}, n\right) = BIN\left(1 + \sum_{i=0}^{n-2} 2^i - \sum_{i=0}^{n-2} x_i \cdot 2^i - \overline{x}_{n-1} \cdot 2^{n-1}, n\right)$$

es decir:

$$\mathsf{r}_{EX2^{n-1},n} = BIN\left(1 + \sum_{i=0}^{n-2} (1 - x_i) \cdot 2^i - \overline{x}_{n-1} \cdot 2^{n-1}, n\right) = BIN\left(1 + \sum_{i=0}^{n-1} \overline{x}_i \cdot 2^i\right)$$

5. Sintetice un operador de cambio de signo para operandos representados en exceso a 2^{n-1}. Evalúe la complejidad *hardware* de su diseño.

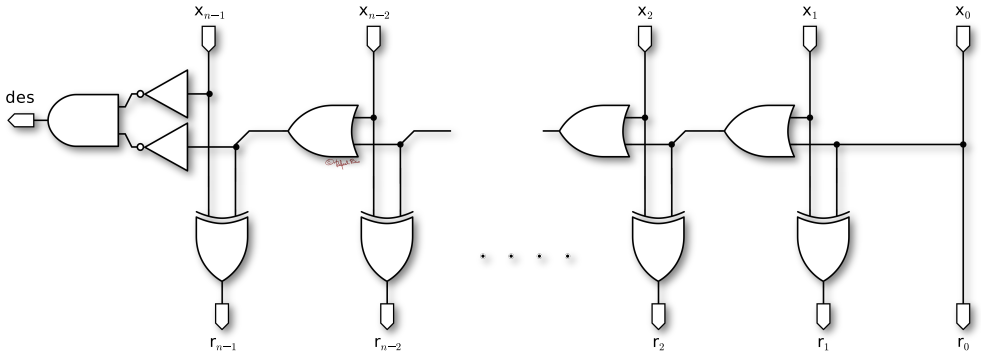
SOLUCIÓN:

Como hemos visto en el problema anterior, el cambio de signo en exceso a 2^{n-1} es equivalente al cambio de signo en complemento a dos. Por tanto, el operador será idéntico salvo en la detección del desbordamiento:

La señal de desbordamiento (des) se activa cuando la entrada es el código "00 $\cdots$ 00", es decir, cuando la cadena de detección alcanza el bit $n-1$ a '0' (señal cero) y x_{n-1}='0'.

$$\mathsf{des} = \overline{\mathsf{cero}} \cdot \overline{\mathsf{x}}_{n-1}$$

En este caso, el código saliente es igual al entrante ("00 $\cdots$ 00").



6. El operador $CMPL2(\mathsf{x}_{2,n})$ es, por definición, $BIN(2^n - x,\ n)$ donde $\mathsf{x}_{2,n}$ es la representación del valor $x \in \mathbb{N}$ sobre n bits. Demuestre que $CMPL2(\mathsf{x}_{2,n}) = \overline{\mathsf{x}}_{2,n} + 1$.

SOLUCIÓN:

Sabemos que $2^n = 1 + \sum_{i=0}^{n-1} 2^i$ y que $x = \sum_{i=0}^{n-1} x_i \cdot 2^i$ de donde:

$$CMPL2(\mathsf{x}_{2,n}) = BIN(2^n - x,\ n) = BIN\left(1 + \sum_{i=0}^{n-1} 2^i - \sum_{i=0}^{n-1} x_i \cdot 2^i,\ n\right) = BIN\left(1 + \sum_{i=0}^{n-1} (1 - x_i) \cdot 2^i,\ n\right)$$

Como x_i vale 0 o 1, tenemos que $1 - x_i = \overline{x}_i$ de donde:

$$CMPL2(\mathsf{x}_{2,n}) = BIN\left(1 + \sum_{i=0}^{n-1} \overline{x}_i \cdot 2^i,\ n\right) = \overline{\mathsf{x}}_{2,n} + 1$$

7. El operador $CMPL1(\mathsf{x}_{2,n})$ es, por definición, $BIN(2^n - 1 - x,\ n)$ donde $\mathsf{x}_{2,n}$ es la representación del valor $x \in \mathbb{N}$ sobre n bits. Demuestre que $CMPL1(\mathsf{x}_{2,n}) = \overline{\mathsf{x}}_{2,n}$.

SOLUCIÓN:

Sabemos que $2^n - 1 = \sum_{i=0}^{n-1} 2^i$ y que $x = \sum_{i=0}^{n-1} x_i \cdot 2^i$ de donde:

$$CMPL1(\mathsf{x}_{2,n}) = BIN(2^n - 1 - x,\, n) = BIN\left(\sum_{i=0}^{n-1} 2^i - \sum_{i=0}^{n-1} x_i \cdot 2^i,\, n\right) = BIN\left(\sum_{i=0}^{n-1} (1 - x_i) \cdot 2^i,\, n\right)$$

Como x_i vale 0 o 1, tenemos que $1 - x_i = \overline{x}_i$ de donde:

$$CMPL1(\mathsf{x}_{2,n}) = BIN\left(\sum_{i=0}^{n-1} \overline{x}_i \cdot 2^i,\, n\right) = \overline{\mathsf{x}}_{2,n}$$

8. Dada una representación $\mathsf{x}_{2,n}$ de $x \in \mathbb{N}$ sobre n bits, demuestre que hacer el complemento a dos del complemento a uno de $\mathsf{x}_{2,n}$ es tanto como sumar 1 a dicha representación. ¿Qué sucede si cambiamos el orden? Demuestre también este caso.

SOLUCIÓN:

Veamos:

$$CMPL2\left(CMPL1(\mathsf{x}_{2,n})\right) = BIN\left(2^n - (2^n - 1 - x), n\right) = BIN\left(x + 1, n\right)$$

En definitiva hemos sintetizado un incrementador unidad.

Y cambiando el orden tengo:

$$CMPL1\left(CMPL2(\mathsf{x}_{2,n})\right) = BIN\left(2^n - 1 - (2^n - x), n\right) = BIN\left(x - 1, n\right)$$

En definitiva hemos sintetizado un decrementador unidad.