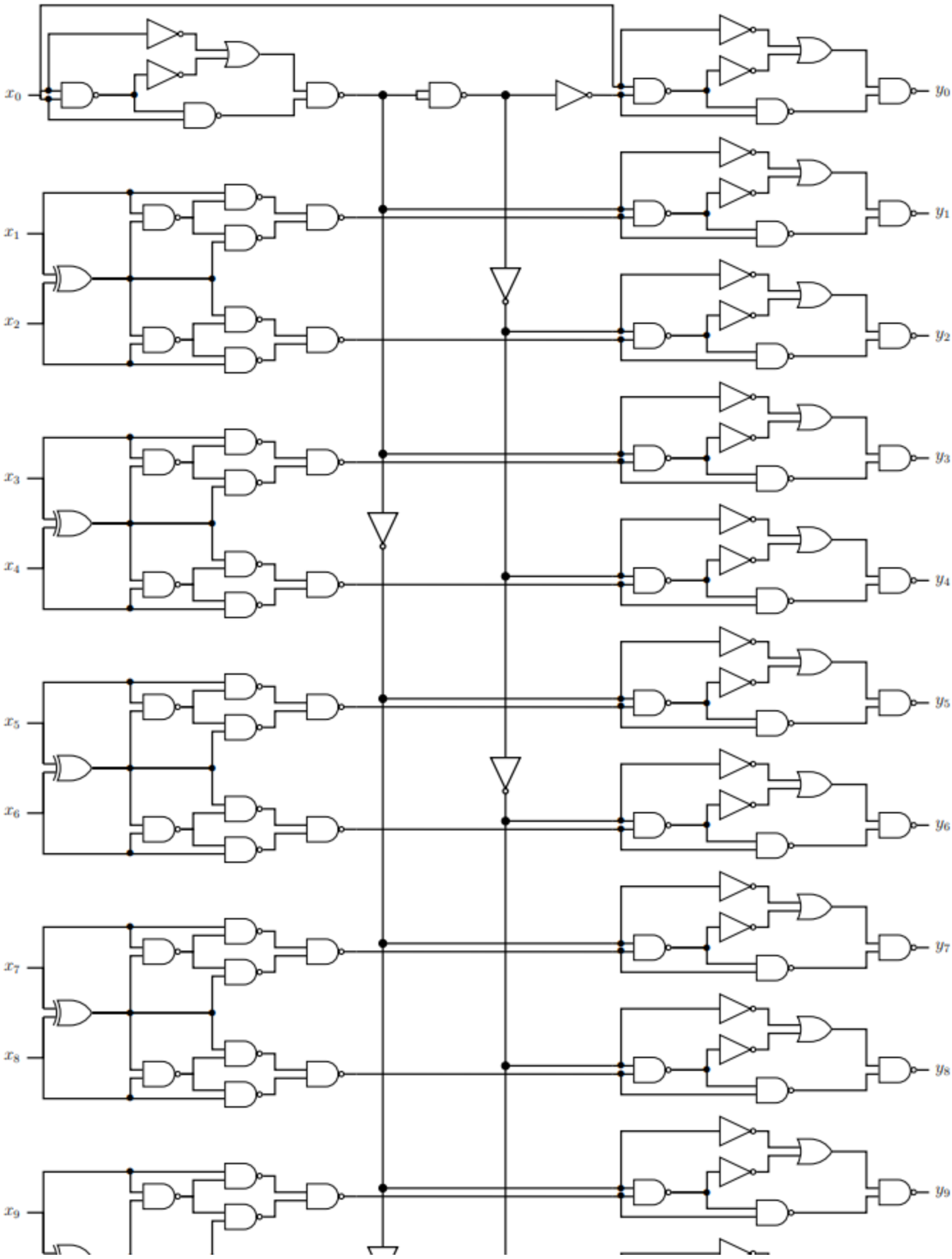
FCSC 2020 - Quarantaine [Hardware 100]

Ce challenge est assez clair : on nous fournit un circuit logique à 40 bits d'entrée pour 40 bits de sortie (d'où le nom), le but est de trouver l'entrée pour laquelle la sortie sera `454088092903` en décimal.

Dans l'énoncé, il est aussi indiqué que `f(19) = 581889079277`, avec f la fonction sur les entiers associée au circuit.



Premiers pas et idées abandonnées

Il y avait dans les challenges d'intro un challenge similaire, mais avec seulement 8 portes. Ayant résolu ce premier challenge à la main, je me voyais mal répliquer la

méthode pour les plus de 400 portes de ce challenge…

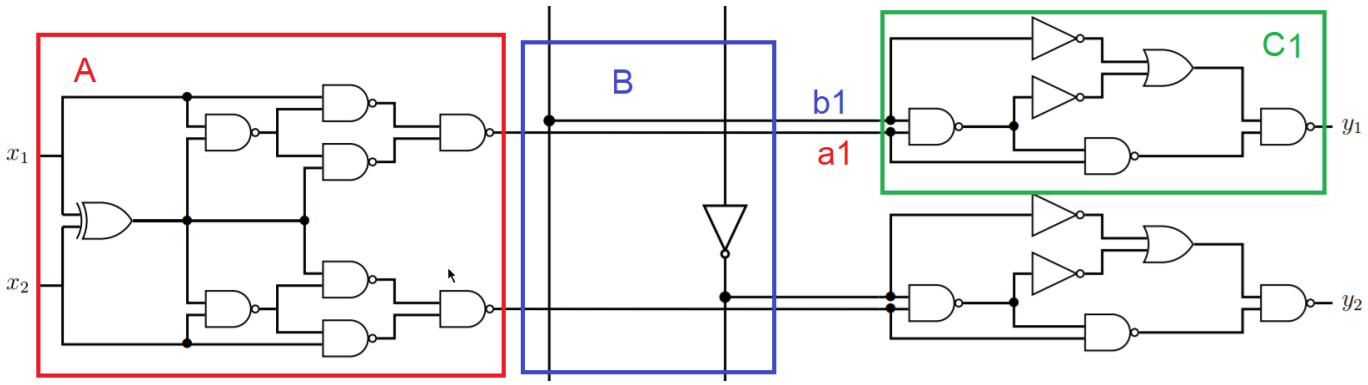
Ma première piste a été de chercher s'il existait une méthode pour reconnaître un circuit dans une image, afin d'utiliser un logiciel de simulation comme Logisim par exemple. Beaucoup de littérature scientifique existe à ce sujet, mais aucune implémentation rapidement utilisable…

J'ai donc rapidement pensé a coder moi-même un script de reconnaissance avec OpenCV et du template matching avant de là aussi rapidement abandonner cette piste : pour un challenge à 100 points, il devait y avoir une alternative plus simple.

Il était donc (enfin) temps de regarder plus près le cricuit !

Un circuit par bloc

Très rapidement, on se rend compte que ce circuit a beaucoup de blocs qui se répètent. Oublions pour le moment les entrées sorties aux extrêmes : les entrées sont toutes regroupées par bloc de 2, et uniquement reliées aux 2 sorties juste en face. Pour l'exemple, penchons nous sur les entrées x1 et x2, et les sorties y1 et y2, mais toutes les entrées sorties sont similaires (sauf les extrêmes x0, x39, y0 et y39).



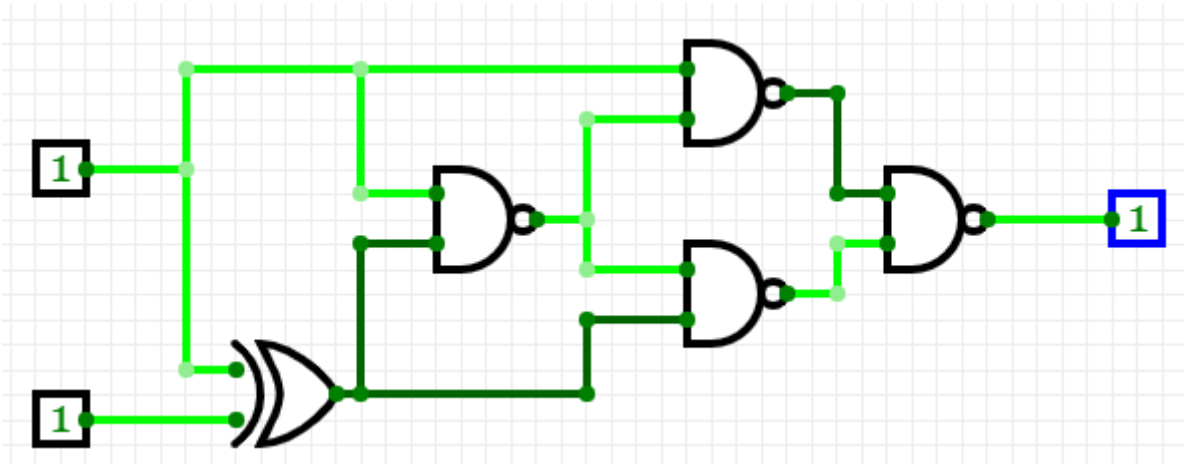
Chaque bloc peut être découper en sous blocs : | Bloc | Entrée | Sortie | | :————:|:————:|:————:| | A | x1 et x2 | 2 sorties, que l'on appelera a1 et a2 | | B | une seule entrée, fonction de x0 ou x39 | 2 sorties, que l'on appelera b1 et b2 | | C1 | a1 et b1 | y1 | | C2 | a2 et b2 | y2 |

Note : le bloc B est légèrement différent à chaque fois, les portes NOT n'étant pas placées au même endroit

Bloc A

La table de vértité de ce premier bloc peut être trouvée à la main. On peut même remarquer que ce bloc étant symétrique, si `a1 = A(x1, x2)` alors `a2 = A(x2,`

`x1)`, avec A la fonction booléenne associée au bloc A. On met tout ça dans circuitverse.org :



Ce qui nous donne le tableau de vérité suivant :

x1	x2	a1
0	0	0
0	1	1
1	0	0
1	1	1

On voit donc très rapidement que… `a1 = x2` ! On a aussi, par symétrie, `a2 = x1`.

Bloc C

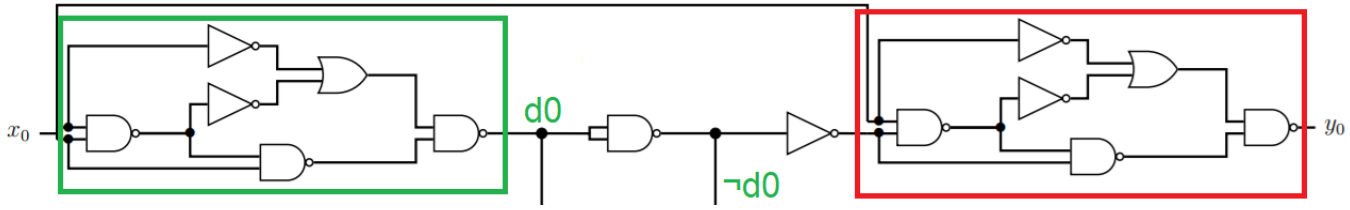
Ce bloc peut également être simplifié en trouvant sa table de vérité sur circuitverse ou tout autre logiciel de simulation :

a1	b1	y1
0	0	0
0	1	1
1	0	1
1	1	0

Là aussi, ce bloc peut être drastiquement simplifié, puisqu'il s'agit d'un simple XOR entre a1 et b1 ! Donc : `y1 = b1` $\oplus$ `a1 = b1` $\oplus$ `x2`

Bloc B

Pour comprendre le bloc B, il faut s'intéresser à ce qu'il se passe aux extrémités du circuit.

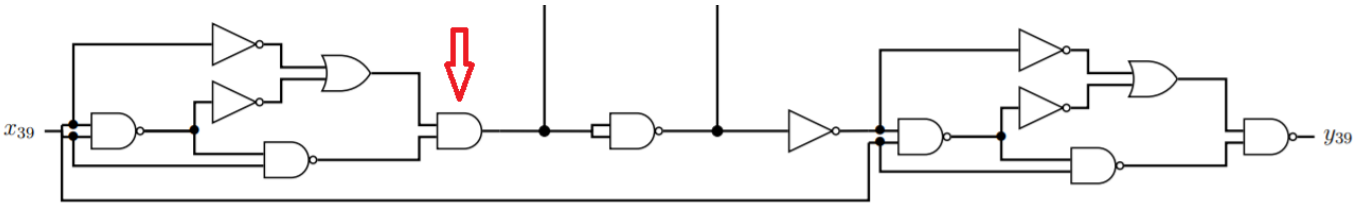


Au premier coup d'oeil, on reconnaît 2 blocs XOR identiques au bloc C. On a donc immédiatement `d0 = x0 ⊕ x0 = 0` ; d0 est donc simplement un `0` quelque soit l'entrée !

Les 2 entrées du bloc B ne dépendent que de d0, et suivant les positions des portes NOT dans le bloc B, `b1` et `b2` seront soit `d0`, soit `¬d0`, c'est-à-dire `0` ou `1` mais seront constants quelque soit l'entrée.

Le sous bloc rouge nous donne également `y0 = d0 ⊕ x0 = 0 ⊕ x0 = x0`.

La même logique est appliquée au bloc de l'autre extrémité, à la différence (mesquine) que la dernière porte du bloc de gauche n'est pas un NAND mais un AND…



Les sorties reliées aux blocs B sont toujours des constantes, mais on a `y39 = d39 ⊕ x39 = 1 ⊕ x39 = ¬x39`

Résultat

Nous avions `y1 = b1 ⊕ x2`, or on sait maintenant que b1 est une constante. Donc quelque soit l'entrée, on aura `y1 = 0 ⊕ x2 = x2` ou `y1 = 1 ⊕ x2 = ¬x2`.

On pourrait retrouver toutes ces constantes b1, b2… à la main simplement en trouvent les positions des portes NOT, mais on a mieux : on peut très facilement les retrouver grâce a l'exemple fourni `f(19) = 581889079277` !

Python à l'aide

Trouver les constantes bi

Pour trouver les constantes, il nous faut :

- Transformer `19` en binaire
- Intervertir 2 par 2 les bits de 1 à 38
- XORer bit par bit avec `581889079277`

En effet, on sait que $y1 = b1 \oplus x2$. Or avec l'exemple, on connait x2 et y1 ; en XORant le tout par x2 : $y1 \oplus x2 = b1 \oplus x2 \oplus x2 = b1 \oplus 0 = b1$

En faisant cela, on retrouvera chaque bit `bi` !

D'où le script Python :

```
def circuitor(n, m):
    # transformation de l'entrée en une liste de 0 & 1 de longue
    bd = list(bin(n)[2:].zfill(40))

    # interversion des bits 1 et 2, 3 et 4... (Bloc A)
    for i in range(1,39,2):
        bd[i],bd[i+1]=bd[i+1],bd[i]

    # on retransforme notre liste en entier
    res = int(''.join(map(str,bd)),2)

    # XOR bit par bit avec nos constantes
    return m ^ res

print(bin(circuitor(19, 581889079277)))
```

On a donc toutes les constantes :
`1000011101111011010010101011011111100000`

Trouver le flag

Maintenant que l'on a les constantes, on réutilise la fonction donnée plus haut, mais en utilisant les constantes :

```
b = circuitor(19, 581889079277)
```

```
print("FCSC{\%d}" % circuitor(1061478808711, b))
```

Et voilà le flag !

Merci à \J et D pour ce challenge !

Written on May 3, 2020

Follow us on Twitter to be informed about new posts ! Follow @h25io