A Locality Preserving One-Sided Binary Tree - Crossbar Switch Wiring Design Algorithm

Devrim Şahin
Department of Computer Science and Engineering
Bilkent University, Ankara, Turkey
devrim.sahin@bilkent.edu.tr

***Abstract*—One-sided crossbar switches allow for a simple implementation of complete K_n graphs. However, designing these circuits is a cumbersome process and can be automated. We present an algorithm that allows designing automatic one-sided binary tree - crossbar switches which do not exceed $\lfloor \frac{n}{2} \rfloor$ columns, and achieves K_n graph without connecting any wires between any three adjacent blocks, thus preserving locality in connections.**

I. Introduction

One-sided switch is a special type of topology used in applications in which input terminals also serve as outputs. Instead of connecting a set of inputs $X = \{x_1, x_2, \cdots, x_n\}$ to a set of outputs $Y = \{y_1, y_2, \cdots, y_r\}$; one-sided switches make connections from a set X onto itself. Thus, every one of the n terminals is connected to every other $(n-1)$, totalling to $\frac{n \cdot (n-1)}{2}$ connections. While switches in general can be represented as bipartite graphs, one-sided switches are equivalent to undirected K_n complete graphs. Figure 1 is an example of a 4-input one-sided switch.

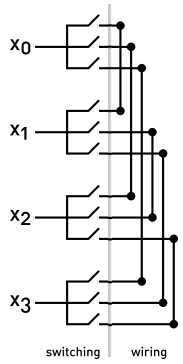

Fig. 1. 4-input binary tree - crossbar switch.

Since there are $n = 4$ inputs, each input x_i should have $n - 1 = 3$ 'copies', that is, separate routes through which x_i can connect to every other x_j. In the implementation above, this is simply achieved by making three copies of each input immediately, then making pairwise connections in the wiring part from every x_i to every x_j $(i \neq j)$.

Given the circuit above, connecting two inputs is trivial. For example, in order to establish a connection between x_0 and x_2, it is required and enough to close two switches (2^{nd} and 7^{th} switches from above, that is, middle x_0 switch and top x_2 switch). However, each of the terminals have a fanout of $n - 1$, which is not a desirable property.

Fanout can be effectively limited to 2 using binary tree switches as described in [1]. For example, if $n = 9$, instead of creating 8 copies immediately, we construct a binary tree of depth 3, where we create 2, 4, and 8 copies step by step, having a fanout of 2 in every step. Figure 2 shows the binary tree adaptation.

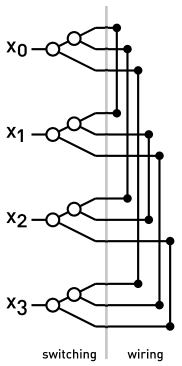

Fig. 2. Binary tree switching. Note that the circles in the switching side depict 1x2 elementary switches.

In the previous examples, the 'rows' (horizontal wires) are ordered as below:

$$(x_0, x_0, x_0), (x_1, x_1, x_1), (x_2, \cdots$$

whereas it is generally more intuitive and efficient to order them as:

$$(x_0, x_1, x_2, x_3), (x_0, x_1, \cdots$$

The advantage of implementing the switching stage in this fashion is that there exists a wiring scheme where the total number of columns in the wiring section does not exceed $\lfloor \frac{n}{2} \rfloor$. A proof is provided in [1]. In this case, binary trees should overlap, but they can simply be implemented on a grid[1] as shown in Figure 3.

The method to obtain a wiring scheme with $\lfloor \frac{n}{2} \rfloor$ columns as described in [1] appeals to cyclic permutation groups. For the example above; the cyclic permutation groups would be as shown below:

978-1-4799-8428-2/15/$31.00 ©2015 IEEE

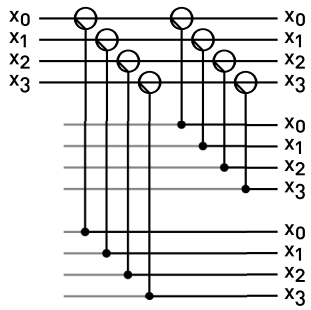

Fig. 3. Grid implementation of binary trees. This implementation also groups x_i into blocks.

$$\begin{aligned} p &= (0123) \\ p^2 &= (02)(13) \\ p^3 &= (0321) \end{aligned}$$

Grouping the numbers obtained by the permutations two by two, one would have the pairs (01), (23), (02), (13), (03), (21). Along with the binary tree implementation depicted in Figure 3, we can implement the wiring step as in Figure 4.

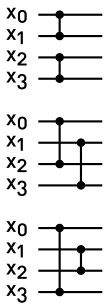

Fig. 4. Wiring step of the one-sided binary tree - crossbar switch.

For even n, obtaining the pairs using cyclic groups is straightforward. However, for odd n, selection of pairs might become tedious. For example, for $n = 5$, cyclic groups are as shown below:

$$\begin{aligned} p &= (01234) &&\rightarrow (01)(23)(4) \\ p^2 &= (02413) &&\rightarrow (02)(41)(3) \\ p^3 &= (03142) &&\rightarrow (03)(1)(42) \\ p^4 &= (04321) &&\rightarrow (04)(3)(21) \end{aligned}$$

Notice that pairs (43) and (13) were not generated, but rather should be constructed by combining the remaining numbers.

In this paper, we present another approach to the problem, which intuitively uses adjacency matrices for representation, and is algorithmically more straightforward. Our method also preserves locality of connections by restricting the interconnections between blocks.

II. Method

For simplicity and clarity, we will separately handle the two cases for even n and odd n. The reader will easily notice by the end that these two are the reflections of the same approach, only separately defined for mathematical completeness. First, we introduce a notion of equivalence between pairs of terminals.

Let $<i, j>$ denote the pair between terminals x_i and x_j. Two pairs $<a, b>$ and $<c, d>$ are equivalent if and only if $(a = c) \wedge (b = d)$, or $(a = d) \wedge (b = c)$. For example, $<0, 3>$ is equivalent to $<0, 3>$ and $<3, 0>$. Using this, we can make the following two statements.

Proposition 1. For any a, b and $x \neq y$, $0 < a, b, < n, 0 < x, y \leq \frac{n}{2}$, the pairs $<a, (a + x) \bmod n>$ and $<b, (b + y) \bmod n>$ cannot be equivalent.

Proof. By the definition of equivalence of pairs, one of the following two cases must be hold.

Case 1: $a = b$ and $(a + x) \bmod n = (b + y) \bmod n$. Then $(a + x) \bmod n = (a + y) \bmod n$, which is possible if and only if $x = y$ [1], which is a contradiction.

Case 2: $a = (b + y) \bmod n$ and $b = (a + x) \bmod n$. Then

$$\begin{aligned} &a - (b + y) = k_1 n, \\ &b - (a + x) = k_2 n \\ \implies &-x - y = (k_1 + k_2)n = -cn \end{aligned}$$

where k_1, k_2, c are integers. But since $0 < x \neq y \leq \frac{n}{2}$, $0 < x + y < n$; therefore $0 < c < 1$, a contradiction. □

Proposition 2. Pairs $<a, (a + x) \bmod n>$ and $<b, (b + x) \bmod n>$ cannot be equivalent for any $a \neq b$ and $0 < x < \frac{n}{2}$.

Proof. By the definition of equivalence of pairs, since $a \neq b$, it immediately follows that these two pairs are equivalent if and only if $a = (b + x)$ **mod** n and $b = (a + x) \bmod n$. Then

$$\begin{aligned} &a - (b + x) = k_1 n, \\ &b - (a + x) = k_2 n \\ &\implies -2x = (k_1 + k_2)n = -cn \end{aligned}$$

where k_1, k_2, c are integers. Since $0 < x < \frac{n}{2}$, $0 < cn < n$, therefore $0 < c < 1$, again, a contradiction. □

In the sequel, we will be using these propositions as we describe our methodology. Also we will make use of adjacency matrices to represent pairs of inputs visually[1]. In our diagrams, '1's will be represented by dark cells.

A. Odd n Case

Given that n is odd, all pairs can be defined in the following fashion:

[1]An adjacency matrix is a binary $n \times n$ matrix in which '1's represent pairings; a 1 in row i and column j represents a pairing between inputs i and j.

$<i, (i+k) \bmod n>, \quad 0 \le i < n, \quad 1 \le k < \frac{n}{2}.$

For example, for $n = 5$, the pairs are:

k=1: $<0,1>$ $<1,2>$ $<2,3>$ $<3,4>$ $<4,0>$,
k=2: $<0,2>$ $<1,3>$ $<2,4>$ $<3,0>$ $<4,1>$.

For a range of $0 \le x < n$, $(x+i) \bmod n$ is a diagonal line on the adjacency matrix, as shown on Figure 5.

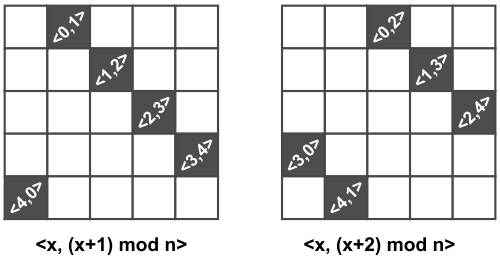

Fig. 5. Adjacency representation of pairs of inputs.

It follows from Proposition 1 that for different values of k, we always have non-overlapping sets of pairs. In addition, Proposition 2 shows that for a given k, all pairs with the same k are different from each other. Note that, for a given k; the number of $<i, (i+k) \bmod n>$ for $0 \le i < n$ pairs is exactly n, and each i appears twice in these pairs: once on the left-hand side, once on the right-hand side. For a given k the number of 'copies' of x_i required is exactly 2; thus, blocks of $x_0, x_1 \cdots x_{n-1}$ can be grouped two by two. Since cases for different k values do not have interconnections, they can be handled separately in parallel.

Another example is shown for $n = 7$ as in Figure 6. In this case k is between 1 and 3 because $n = 7$ and $1 \le k \le \frac{1}{2}(n-1) = (7-1)/2 = 3$. Since the adjacency matrix is symmetric ($<i,j> \equiv <j,i>$), all pairs can be depicted in the upper triangle as below.

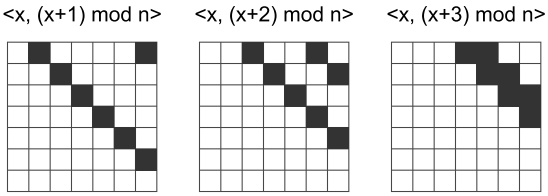

Fig. 6. $n = 7$ case. All pairs are shown in the upper triangle.

B. Even n Case

The equation introduced for the odd case is valid for the even case, although it is not sufficient to cover all possible pairs:

$<i, (i+k) \bmod n>, \quad 0 \le i < n, \quad 1 \le k < \frac{n}{2}.$

For $n = 6$, the exact number of possible pairs is $\frac{1}{2}n(n-1) = \frac{1}{2}6 \cdot 5 = 15$, whereas the definition above generates $n \cdot (\frac{n}{2} - 1) = 6 \cdot 2 = 12$ pairs. The three missing pairs belong to the case where $k = \frac{n}{2} = 3$.

However, including $k = \frac{n}{2}$ to the boundary of the equation above would violate the conditions defined for k in Proposition 2. To demonstrate, let $n = 6$ and $k = 3$. The pairs generated are as follows:

$$<0,3> \quad <1,4> \quad <2,5> \quad <3,0> \quad <4,1> \quad <5,2>$$

In this case, all pairs appear twice; once in order, then in reverse, such that $<a + \frac{n}{2}, ((a + \frac{n}{2}) + \frac{n}{2}) \textbf{ mod } n> = <a + \frac{n}{2}, a> \equiv <a, a + \frac{n}{2}>$, when $k = \frac{n}{2}$. The problem is solved if $0 \le a < \frac{n}{2}$ instead of $0 \le a < n$, in which case the modulo operator disappears, and the pair definition becomes simplified as $<i, i + \frac{n}{2}>$.

Then the two subsets that cover the even n case are as follows:

$$<i, (i+k) \textbf{ mod } n>, \quad 0 \le i < n, \quad 1 \le k < \frac{n}{2},$$
$$<i, i + \frac{n}{2}>, \quad 0 \le i < \frac{n}{2}.$$

Since k are different, Proposition 1 shows that these two subsets never overlap. Set 1 has $n(\frac{n}{2} - 1)$ elements, and set 2 has $\frac{n}{2}$ elements; adding up to $(n+1)(\frac{n}{2}) - n = \frac{1}{2}n(n+1) - \frac{1}{2}2n = \frac{1}{2}n(n-1)$ unique elements; covering every necessary pair.

Figure 7 shows how pairs are generated for $n = 8$. The rightmost figure shows the second set ($k = \frac{8}{2} = 4$ case).

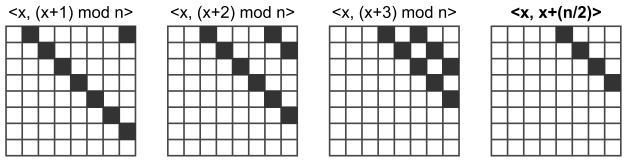

Fig. 7. Pair generation for even n. The rightmost figure depicts the $k = n/2$ set.

Note that the special definition provided for the $k = \frac{n}{2}$ case, prevents the last 4 cells to be spanned twice.

C. Wiring

The actual wiring of the generated pairs is described in the following section.

For even n, wiring of the second set is trivial, because the first half of inputs connect to the second half. Since there are $\frac{n}{2}$ wires, the number of columns is exactly $\lfloor \frac{n}{2} \rfloor = \frac{n}{2}$. Only

one block is necessary and sufficient for the wiring of this set. An example is shown in Figure 8 for $n = 8$.

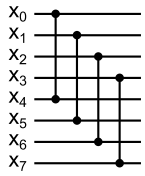

Fig. 8. Wiring of the block handling the $k = n/2$ case for $n = 8$.

Having handled the special case for even n, the remaining pairs are identically defined for odd and even n:

$$<i, (i+k) \textbf{ mod } n>, \quad 0 \leq i < n, \quad 1 \leq k < \tfrac{n}{2}.$$

In order to get rid of the modulo operator and represent all pairs $<a, b>$ in the upper triangle such that $a < b$, the definition above should be separated into two subsets as shown below for each k ($1 \leq k < \frac{n}{2}$):

$$< i, (i+k) > \qquad \text{for } 0 \leq i < n-k,$$

$$< (i+k-n), i > \qquad \text{for } n-k \leq i < n.$$

The second subset can also be written as follows:

$$< i, i+n-k > \qquad \text{for } 0 \leq i < k.$$

$k < \frac{n}{2} \implies k < n-k$; therefore the range for i in the second set is a strict subset of the range for i in the first set. Since the first set uses the first copies of the terminals, the second set should strictly be placed on to the second block (recall that exactly two blocks are allocated for each value of k). That is, the first k values for i in the first set are forced to belong to the first block. Since these pairs are defined as $<i, (i+k)>$; the $x_k \cdots x_{2k-1}$ terminals in the first block are used as the destination ports; therefore these connections are made in the second block, and so forth.

The resulting distribution of connections can be formalized in the following manner:

- The second set always belongs to the second block.
- For a pair in the first set $<i, (i+k)>$, if $\lfloor \frac{i}{k} \rfloor$ **mod** $2 = 0$, put the pair in the first block, otherwise put it in the second block.

Recall that these formulations are defined for a fixed k, and should be repeated for every k in range. Figure 9 is the resulting circuit for $n = 8$. The condition for $k = 3$ is handled separately and uses only one block. The other blocks are clustered in two, and have no interconnections.

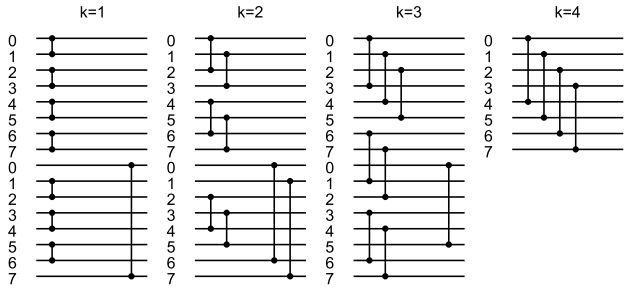

Fig. 9. Complete wiring scheme for n=8. Each cluster of inputs correspond to a separate value of k, and these clusters have no connection in between.

For demonstration purposes, pairs generated by the second set are shown on the right-hand side.

A verbal guideline for the approach can be provided as follows:

- Given a k, insert the first k pairs ($<0, k>$,$<1, k+1>$,$\cdots$,$<k-1, 2k>$) into the first block.
- Insert the next k blocks to the second; then the following k into the first...
- Continue this process until the first set is exhausted.
- Insert every item in the second set into the second block.
- Repeat steps above for every k.
- If n is an even number, consider the $k = n/2$ case separately as described above.

As long as the first-second block selection process is respected, regardless of the order in which the pairs are inserted, no collisions can occur. Finally, it is not difficult to prove that the number of columns does not exceed $\lfloor \frac{n}{2} \rfloor$ in this scheme.

III. Conclusion

We presented an algorithm to connect all pairs of n terminals in an n-terminal one-sided, binary-tree switch in $\lfloor n/2 \rfloor$ columns of wiring. We have implemented and tested this algorithm in Java. The implementation provides an .svg vector output as well as an ASCII console representation of the wiring scheme. The code for the algorithm is available at https://github.com/kubuzetto/crossbarWiring/.

[1] A. Yavuz Oruç. One-sided binary-tree-based crossbar switch fabrics. Invention Disclosure. University of Maryland, College Park. PS-2014-57. Apr. 9, 2014.