An Algorithm-Base Fault Tolerance (More Than One Error) Using Concurrent Error Detection for FFT Processors

Chin-Chien Sha and Robert W. Leavene

Department of Electrical and Computer Engineering
University of Missouri at Columbia

Abstract.
An algorithm is proposed to maintain fault tolerance for a highly reliable FFT processor, even after the processor has been reconfigured (by detecting a single fault). It proves that concurrent error detection (CED) scheme using a redundant stage of decimation in frequency FFT (DIF-FFT) butterflies as a decoder can detect all the faults theoretically. This CED scheme and the modification of the standard DIF-FFT processor as a recirculated shuffle exchange one will also alleviate the difficulty of reconfiguration and will provide the ability of some degradation in performance in the presence of more than one fault in the processor.

I. Introduction.

The discrete Fourier transform (DFT) is a computationally intensive operation. Many methods have been developed to reduce the number of multiplication that is usually more critical than the number of additions in this computation. The fast Fourier transform (FFT) is a method to compute the DFT, which is popularized by Cooley and Tukey [6]. Given an N-point data sample, where N is a power of 2, the Cooley and Tukey algorithm reduces the number of complex multiplication from $N^2$ to $N \log_2 N$.

With the advent of VLSI technology, large collections of processing elements that cooperate with each other to achieve high speed computation have become economically feasible. In a high-performance system any functional error may seriously jeopardize the operation of the system. Some level of fault tolerance must be incorporated to ensure the result of durable computations.

Fault tolerance is a continuous operation in case of failure. Technically, the algorithm-based fault tolerance should include three steps: 1. to detect one or more than one fault unit by itself; 2. to locate these faults based on the first step; 3. to reconfigure to achieve continuous operation. The third step is the most difficult technique because the original butterfly structure has a very complex set of interconnection. The reconfigured scheme will impose some additional difficulties to build the rest of the good units. Since the technique used for system detection is the prerequisite for the reconfiguration, it may affect the ability of the system to locate fault units and to reconfigure the system. How to alleviate the complexity of the system reconfiguration is determined by the design of the scheme of the system detection.

To achieve this goal, we provide a new CED scheme that the decoder is modeled as a stage of DIF-FFT butterflies. To make sure that all butterflies are consistent, we set all the butterflies to the DIF-FFT in the FFT processor the same as those in the decoder.

The $N$-point FFT processor in this paper consists of $\log_2 N$ stages and each of them is composed of N/2 butterflies, where $N=2^k$ and $k$ is an integer. A butterfly is made of two complex adders and one complex multiplier. In addition, possible location for faults in the butterfly is labeled 1 through 10 as shown in Fig. 1.

In section II, we propose a new CED scheme for FFT network and prove that this scheme has 100% fault coverage theoretically. In section III, we measure the hardware overhead by the number of multiplier. In section IV, we change the standard butterfly processor to a recirculated shuffle exchange connection to get the concurrent error location and graceful reconfiguration system. Finally, we demonstrate an example of an algorithm-based fault tolerant scheme to prove that the FFT processor can maintain its operation even more than one fault exists.

![Fig. 1. DIF-FFT butterfly](image-url)
II. The concurrent error detection scheme.

An algorithm to construct CED scheme for detecting all the faults in an $N$-point FFT processor is provided as below:

1. Determine each value of the twiddle factors $W^{(i)}_N$ in the decoder (a stage of DIF-FFT butterflies), where $0 \leq i \leq N-1$.

2. Form the sum of the output of the decoder

$$S_{out} = \sum_{i=0}^{N-1} (1+W^{(i)}_N) X(i) \text{ where } X(i) \ (0 \leq i \leq N-1)$$

is the output of the FFT processor.

3. Form the sum of the weighted input

$$S_{in} = \sum_{i=0}^{N-1} b^{i}_N x(i) \text{ in term of } W^{(i)}_N$$

where

$$b^{i}_N = \sum_{k=0}^{N-1} (1+W^{(i)}_N) W^{k}_N \text{ where } 0 \leq i \leq N-1, \text{ and } x(i) \text{ is the input of FFT processor.}$$

4. If $S_{out} \neq S_{in}$, a fault is detected.

The first step in the above algorithm is to determine a method to set values of the twiddle factors in the decoder.

The method to set the $W^{i}_N$, where $0 \leq i \leq N-1$ is defined as follows:

$$a^{d_1d_2...d_k}_N = W^{d_1d_2...d_k}_N$$

where $d_1d_2...d_k$ are binary numbers.

For example, if $N=8(v=3)$

$$d_1 \ d_2 \ d_3 \to d_1d_2 \quad d_1d_2 \ d_3 \to d_1d_3$$

$$d_2 = W^0_8 \quad 0 \ 0 \ 0 \to 0 \ 0 \ 0 \quad a^4_8 = W^2_8 \ 1 \ 0 \ 0 \to 0 \ 1 \ 0$$

$$a^1_8 = W^6_8 \quad 0 \ 0 \ 1 \to 1 \ 0 \ 0 \quad a^5_8 = W^6_8 \ 1 \ 0 \ 1 \to 1 \ 1 \ 0$$

$$a^2_8 = W^8_8 \quad 0 \ 1 \ 0 \to 0 \ 0 \ 1 \quad a^6_8 = W^3_8 \ 1 \ 1 \ 0 \to 0 \ 1 \ 1$$

$$a^3_8 = W^8_8 \quad 0 \ 1 \ 1 \to 1 \ 0 \ 1 \quad a^7_8 = W^2_8 \ 1 \ 1 \ 1 \to 1 \ 1 \ 1$$

After the twiddle factors in the decoder are found, the coefficient $b^i_N$ which is constructed according to these factors is produced as follows:

The $N$-point output of the decoder is separated into two parts, the even and the odd sum, the former, denoted SEVEN, consists of the even-numbered output point in the decoder but the latter, denoted SODD, consists of the odd-numbered ones.

Lemma A1: The SEVEN part can only get the zero term $b^0_N = N$ in the encoder.

Proof:

$$SEVEN = \sum_{k=0}^{N-1} S_{(2k)} = [111...11]^k \cdot X(0) \quad N\times X(N-1)$$

Using the matrix representation for the DFT terms $X(k)$ gives

$$SEVEN = \begin{bmatrix} W^0_N & ... & W^{N-1}_N \end{bmatrix} \begin{bmatrix} x(0) \\ \vdots \\ x(N-1) \end{bmatrix}$$

Since $\sum_{k=0}^{N-1} W^k_N = 0$ and $\sum_{k=0}^{N-1} W^0_N = N$, then

$$SEVEN = (N00...000)^k \cdot x(0) = N x(0) \quad b^0_N = N$$

Q.E.D.

Lemma A2: The SODD part can get the general equation $b^i_N = \frac{2-2W^i_N}{1-W^{2i+1}_N}$, for $1 \leq i \leq N-1$, to set all except zero terms in the encoder.

$$SODD = \sum_{k=0}^{N-1} S_{(2k+1)} = [a^0_N a^2_N \ldots a^{N-1}_N]^k \cdot X(0) \quad X(N-1)$$

57
\[
= \begin{bmatrix}
W_N^0 & W_N^{N/2} & \ldots & W_N^{N-1}
\end{bmatrix}
\begin{bmatrix}
W_N^0
\vdots
\vdots
\vdots
W_N^{N-1}
\end{bmatrix}
\begin{bmatrix}
x(0)
\vdots
x(N-1)
\end{bmatrix}
\]

Let \( \overline{W_A} \) refer to the \( I^*N \) matrix of \( W_N^k \) terms and \( \overline{W_B} \) refer to the \( N^*N \) matrix of \( W_N^k \) terms. We multiply the columns of \( \overline{W_B} \) by \( \overline{W_A} \). Hence the \( k^\text{th} \) element of \( \overline{W_B} \) will be multiplied by the \( k^\text{th} \) element of \( \overline{W_A} \). If we interchange the \( k^\text{th} \) and \( i^\text{th} \) elements of \( \overline{W_A} \), we must also interchange the \( k^\text{th} \) and \( i^\text{th} \) rows of \( \overline{W_B} \) to maintain our algebraic integrity.

Now let the binary value for the \( i^\text{th} \) term in the reordered \( \overline{W_A} \) matrix be given as \( i = (i_1 i_2 \ldots i_N) \). Then the corresponding row in the reordered \( \overline{W_B} \) matrix will be given as \( [W_N^{i_1} W_N^{i_2} \ldots W_N^{i_N}] \), where \( (N-1)(i_2 \ldots i_N) \) represents \( (N-1) \) binary sums of \( (i_2 \ldots i_N) \). Define the reordered \( \overline{W_A} \) matrix as \( \overline{W_{AR}} \), where

\[
\overline{W_{AR}} = \begin{bmatrix}
W_N^{i_1} W_N^{i_2} \ldots W_N^{i_N-2} W_N^{i_N-1} W_N^{i_N-2} W_N^{i_N-1}
\end{bmatrix}
\]

Define the reordered \( \overline{W_B} \) matrix as \( \overline{W_{BR}} \), then

\[
\overline{W_{BR}} = \begin{bmatrix}
W_N^0 W_N^1 \ldots W_N^{N/2 - 2} W_N^{N/2 - 1} \ldots W_N^{N-2} W_N^{N-1}
\end{bmatrix}
\]

The next step is to partition the \( \overline{W_{AR}} \) and \( \overline{W_{BR}} \), matrices as

\[
\overline{A} = \begin{bmatrix}
W_N^{N/2} W_N^{N/2+1} \ldots W_N^{N-1}
\end{bmatrix}
\]

\[
\overline{C} = \begin{bmatrix}
W_N^0 W_N^1 \ldots W_N^{N-1}
\end{bmatrix}
\]

The matrix equation for \( \text{SODD} \) is given as follows:

\[
\text{SODD} = \begin{bmatrix}
\overline{A} & \overline{B}
\end{bmatrix}
\begin{bmatrix}
\overline{C}
\overline{D}
\end{bmatrix}
\begin{bmatrix}
x(0)
\vdots
x(N-1)
\end{bmatrix}
\]

Then, \( \text{Col}_i(j) \) is \( I^*N \) row vector, which is denoted as the product of \( \overline{A} \overline{C} \) and \( \text{Col}_i(j) \) is \( I^*N \) row vector, which is denoted as the product of \( \overline{B} \overline{D} \). We get

\[
\frac{1}{1 - W_N^{2(i+1)^2} - N/2} \frac{1}{1 - W_N^{2i+1}} = 2, 0 \leq i \leq N - 1
\]

\[
\text{Col}_i(j) = \sum_{k=0}^{N/2-1} (W_N^{k+i+k}) \sum_{k=0}^{N/2-1} (W_N^{k+N/2+2(i+1)})
\]

Therefore, the coefficient \( b_N^i = \text{Col}_i(j) + \text{Col}_j(i) \). We get

\[
b_N^i = \frac{2 - W_N^{2i+1}}{1 - W_N^{2i+1}}, 1 \leq i \leq N - 1
\]

Finally, we propose the following CED scheme to detect all errors in the FFT processor.

1. We have to define the levels. The combination of the \( N \)-point FFT processor and the decoder is rearranged and then separated into \( \log_2 N+1 \) levels. Each level is a gain accumulated from an input stage to the final stage(decoder). For example, the first level will accumulate from the first stage to the final stage and the second level will accumulate from the second stage to the final one.

2. We want to prove that any faults in all levels can be detected.

Lemma B: By using the CED, an error is detected at the first level in the \( N \)-point processor if and only if \( b_N^i \neq 0 \).

Proof: Since \( b_N^i \) can be considered as the gain accumulated from the first stage to the decoder, all the errors at the first level can be detected.

If the gain starts from the second stage, the \( N^*N \) matrix \( \overline{W_{BR}} \) which is defined above will be reduced to \( N/2^*N \) \( \begin{bmatrix} \overline{C} & \overline{D} \end{bmatrix} \) matrix, where matrix \( N/2^*N/2 \) \( \overline{C} \) is the left half part of \( \overline{C} \).

This gain vector is denoted as \( \overline{\overline{S}}^i(n) \overline{\overline{S}}^j(n) \), where \( \overline{\overline{S}}^i(n) \) and \( \overline{\overline{S}}^j(n) \) is \( 1^*N/2 \). Then \( \overline{\overline{S}}^i(n) = [\overline{\overline{A}} | \overline{\overline{C}}] \), \( \overline{\overline{S}}^j(n) = [\overline{\overline{B}} | \overline{\overline{C}}] \), where \( 0 \leq n \leq N/2 - 1 \). \( \overline{\overline{A}} \) and \( \overline{\overline{B}} \) are defined as above.
Lemma C: All errors at the second level can be detected if and only if both the following equations are satisfied:
\[ \overline{S}_0(n) = [\overline{A}][\overline{C}_1] \neq 0, \quad \overline{S}_1(n) = [\overline{B}][\overline{C}_1] \neq 0 \]

Proof: \[ \overline{S}_0(n) = \sum_{k=0}^{N/2-1} (W_N^k W_N^{2nk}) = \frac{2}{1 - W_N^{2n+1}} \neq 0 \] \hspace{1cm} (1)

\[ \overline{S}_1(n) = \sum_{k=0}^{N/2-1} (W_N^{nk} W_N^{2nk}) = \frac{2}{1 - W_N^{2n+1}} \neq 0 \] \hspace{1cm} (2)

On the third level, \( \overline{S}_0(n) \) breaks into \( \overline{S}_0^2(n) \) and \( \overline{S}_1^2(n) \); and the \( \overline{S}_1(n) \) breaks into \( \overline{S}_2(n) \) and \( \overline{S}_3(n) \). By using equations (1) and (2) and replacing \( k \) by \( 2s \), we obtain
\[ \overline{S}_0(n) = \sum_{s=0}^{N/2-1} (W_N^{2ns} W_N^{4ns}) = \frac{2}{1 - W_N^{4n+2}} \neq 0 \] \hspace{1cm} (3)

\[ \overline{S}_1(n) = W_N^{N/2} \overline{S}_0^2(n) \neq 0 \] \hspace{1cm} (4)

\[ \overline{S}_2(n) = \sum_{s=0}^{N/2-1} (W_N^{N/2} W_N^{2ns} W_N^{4ns}) = W_N^{N/2} \overline{S}_0^2(n) \neq 0 \] \hspace{1cm} (5)

\[ \overline{S}_3(n) = W_N^{N/4} \overline{S}_0^2(n) = W_N^{3N/4} \overline{S}_0^2(n) \neq 0 \] \hspace{1cm} (6)

Eventually, by the induction from (1) to (6), we can obtain the general equation to measure the gain from the (0+1) levels, where \( 1 \leq l \leq \log_2 N \)
\[ S_p(n) = \frac{2}{1 - W_N^{(2n+1)p}} W_N^{(N/p)p} \quad \text{where} \quad 0 \leq p \leq (l^2 - 1), \quad 0 \leq n \leq N/l^2 \]

Theorem 1. The summation of the odd-numbered output points in the decoder can detect all the errors from the (0+1) level where \( 1 \leq l \leq \log_2 N \) if \( S_p(n) \neq 0 \).

Proof: It can be seen that \( S_p(n) \neq 0 \) since one of the numerators \( W_N^{(N/p)p} \neq 0 \) and denominators \( 1 - W_N^{(2n+1)p} \neq 0 \), where \( 0 \leq p \leq (l^2 - 1), 0 \leq n \leq N/l^2 \)

III. Overhead.

The detection of faults that occur during the computation of the DFT and the subsequent reorganization of the processor will require additional resources. These resources will be in the form of by-pass networks, additional butterflies, extra time, etc. Exactly what types of resources are required will be determined by the fault detection/correction method being used. Since a multiplier is a lot more complex than an adder, we will consider the hardware overhead only in terms of the multipliers.

One measure of these additional resources is to divide the number of extra multipliers required by the number of multipliers required in the FFT networks. Refer to Fig. 2. on the input side, there will be one multiplier required for each input line. This will give \( N \) multipliers. In addition, there is a redundant stage required in the computational algorithm. This will require \( N/2 \) multipliers, so hardware overhead is
\[ \frac{N/2 + N}{N/2 \cdot \log_2 N} = \frac{3}{\log_2 N} \]

The above equation shows only the measure of increased numbers of multipliers. This does not account for the increased complexity of the interconnected networks, nor does it relate the amount of time needed to locate a fault and reconfigure the system, if reconfiguration is possible.

IV. Error location and reconfiguration.

There are two ways to correct an error once it has been detected. One requires substantial hardware resources while the other requires time resources. Some papers had tried to correct this error. However, the technique used by Chou and Males [1] required \( \log_2 N + 5 \) to locate an error and to sacrifice half of the butterflies plus 100 percent time overhead to achieve degraded operation. Jou and Abraham [2] needed 2 computation cycles to find and locate an error. In addition, they have no way to reconfigure their faulty structure. [3] and [4] cannot even provide the solution for the difficulties encountered.

In order to locate the error concurrently and to reconfigure the structure without time overhead, this paper proposes to modify the standard DFT/FFT processor as the recirculated shuffle exchange processor [5].

Before we develop the recirculated shuffle exchange connection processor for the DFT/FFT, let us review the time required to compute a standard DFT/FFT algorithm. The DFT on \( N=2^p \) elements can be computed in \( O(\log_2 N) \) time on a shuffle-exchange graph composed of \( \log_2 N \) successive "shuffle and compute" steps. Each such step contains a parallel propagation of the \( N \) data along the shuffle connections, followed by the parallel execution of \( N/2 \) "butterfly" operations. The shuffle connections between each two columns of \( N \) nodes are the same. The output of module 0 is always the input to the next module 0, and output 1 is always the input to 4, and so on. Since all shuffle connections are the same, the non-pipeline DFT processor can be built by using a single stage of \( N \) nodes. The entire computation is performed by recycling data for a total of \( \log_2 N \), (see Figure 3 in [5]). The data is taken from the "Original Data Input" lines, computed and recirculated through the butterflies, and finally outputted on the "Transformed Data Output" lines.

This layout of the recirculated shuffle exchange processor can be modeled as independent stages in the pipeline DFT/FFT processor (see Figure 5 in [5]). Then,
the interconnection for transferring data between two stages is less complicated. Since every stage can compute data independently, it will not propagate an error to the next one. If an error occurs, this structure will make it easier to bypass the faulted stage and to reconstruct the rest of the stages.

To detect an error, each butterfly will calculate in \( \log_2 N + 1 \) time which consists of the calculation of data in \( \log_2 N \) time plus multiplication of the result by the decoder's twiddle factors. To satisfy timing constraints, the redundant stage needs to be inserted into the processor, so that each butterfly can have enough time to calculate. The result of the above calculation will be compared with the encoded value. If they do not match, the current input data will be stored in the input buffer to wait for the next steps. The stage will be set to fault and the fault alarm will be active. If we still require the rest of the stages to have fault-tolerant ability, they need to calculate in \( \log_2 N + 1 \) time. In order to get additional time for the stages to calculate all the twiddle factors, the buffer need to store the input data on the duration of the overhead time. The required overhead time as a function of \( N \) and the number of errors is given by: \( \text{Time} = \text{errors} / (\log_2 N + 1 - \text{errors}) \). The algorithm on this pipeline recirculated shuffle exchange processor is to control switches on the input and output side to perform either a computation or a bypass function (see Figure 3). The switch, with its location is defined as \( \text{sw}(\text{stage}, S_i, S_o) \), where the variable stage varies between 1 and \( \log_2 N \).

![Fig. 3. (a) Two Nodes Performing the Butterfly Operation with Fault Tolerant Switching Capability](image)

![Fig. 4. The Layout of The Pipeline 8 Point DIF-FFT Fault-Tolerant Processor](image)

operation of the switch is defined by the variable \( S_i \) and \( S_o \). They operate as:

- If \( S_i = 1 \) then input data from input line.
- If \( S_i = 0 \) then input data from circulated line.
- If \( S_o = 1 \) then output data to computed line.
- If \( S_o = 0 \) then output data to bypass link.

We can now combine the switching function \( \text{sw}(\text{stage}, S_i, S_o) \) with the operation of a butterfly to become \( \text{swbuf}(\text{stage}, S_i, S_o) \). With this algorithm, there are only two possibilities for the function \( \text{swbuf}(\text{stage}, S_i, S_o) \) on each step. One is \( \text{swbuf}(\text{stage}, 1, 1) \), for instance, \( \text{swbuf}(3, 1, 1) \). This example will output the calculated data on stage 3 and use data in the calculation from the previous stage. The other possibility is \( \text{swbuf}(\text{stage}, 0, 0) \) which will output data to the bypass link and calculate values using the circulated value.

Let us consider a pipeline recirculated shuffle exchange scheme for the 8-point DIF-FFT processor with more than one fault tolerance. We assume the stage 2 and stage 4 have faulty multipliers.

Start from step 1 to step 13, the \( \text{swbuf}() \) of every stage is shown on the Table 1.

In step 6, the result of the comparison produces a mismatch. The recovery strategy is to store \( x(6) \) in the input buffer to wait for the next step, followed by the alarm bell to indicate that stage is faulty.

After step 8, this system only has two stages available. The operation to be calculated four times will produce 100 percent overhead time.
Procedure DIF-FFT (fault tolerance)

N: the number of point DFT
stage: integer[1... n = log₂N]
L: the number of the loops
sbuf((stage, Si, So)): it is defaulted to sbuf((stage, 0, 0)) which will bypass the input data and calculate circulated values. If sbuf((stage, Si, So)) is set to sbuf((stage, 1, 1)) after one step it will automatically reset to sbuf((stage, 0, 0)).
macthstage(i): compare the result of stage i with that of the encoder.

begin
  faultstage = 0; match = true; n = log₂N + 1
for j = 1 to L+1
begin
  for i = 1 to n
  begin
    if (i == faultstage) then store (input data)
    if (i < faultstage)
      sbuf((i, 1, 1))
    matchstage(i)
    if (match == fail)
      store (input signal)
      alarm fault bell
      faultstage = 1
  end
end
endrepeat
endrepeat.

V. Conclusion.

This paper proposes a complete fault tolerant method for FFT networks. The proposed scheme can not only detect more than one error, but also locate the fault concurrently and restructure it easily. Even though the whole system is restructured after detecting an error, it still provides the ability to remain operational possibility with some degradation in performance in the presence of more than one fault in the processor.

Reference


--- Table 1 ---

<table>
<thead>
<tr>
<th>STEP</th>
<th>INPUT</th>
<th>STAGE 1 sbuf()</th>
<th>STAGE 2 sbuf()</th>
<th>STAGE 3 sbuf()</th>
<th>STAGE 4 sbuf()</th>
<th>OUTPUT</th>
<th>COMPARE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x₁</td>
<td>1,1,1</td>
<td>2,0,0</td>
<td>3,0,0</td>
<td>4,0,0</td>
<td>unknown</td>
<td>unknown</td>
</tr>
<tr>
<td>2</td>
<td>x₂</td>
<td>1,0,0</td>
<td>2,1,1</td>
<td>3,0,0</td>
<td>4,0,0</td>
<td>unknown</td>
<td>unknown</td>
</tr>
<tr>
<td>3</td>
<td>x₃</td>
<td>1,0,0</td>
<td>2,0,0</td>
<td>3,1,1</td>
<td>4,0,0</td>
<td>unknown</td>
<td>unknown</td>
</tr>
<tr>
<td>4</td>
<td>x₄</td>
<td>1,0,0</td>
<td>2,0,0</td>
<td>3,0,0</td>
<td>4,1,1</td>
<td>unknown</td>
<td>unknown</td>
</tr>
<tr>
<td>5</td>
<td>x₅</td>
<td>1,1,1</td>
<td>2,0,0</td>
<td>3,0,0</td>
<td>4,0,0</td>
<td>X₁</td>
<td>match</td>
</tr>
<tr>
<td>6</td>
<td>x₆</td>
<td>1,0,0</td>
<td>2,1,1</td>
<td>3,0,0</td>
<td>4,0,0</td>
<td>X₂</td>
<td>mismatch</td>
</tr>
<tr>
<td>7</td>
<td>x₇</td>
<td>1,0,0</td>
<td>2,0,0</td>
<td>3,1,1</td>
<td>4,0,0</td>
<td>X₃</td>
<td>match</td>
</tr>
<tr>
<td>8</td>
<td>x₈</td>
<td>1,0,0</td>
<td>2,0,0</td>
<td>3,0,0</td>
<td>4,1,1</td>
<td>X₄</td>
<td>mismatch</td>
</tr>
<tr>
<td>9</td>
<td>x₉</td>
<td>1,1,1</td>
<td>2,0,0</td>
<td>3,0,0</td>
<td>4,0,0</td>
<td>X₅</td>
<td>match</td>
</tr>
<tr>
<td>10</td>
<td>x₈</td>
<td>1,0,0</td>
<td>2,0,0</td>
<td>3,0,0</td>
<td>4,0,0</td>
<td>X₆</td>
<td>match</td>
</tr>
<tr>
<td>11</td>
<td>x₈</td>
<td>1,0,0</td>
<td>2,0,0</td>
<td>3,1,1</td>
<td>4,0,0</td>
<td>X₇</td>
<td>match</td>
</tr>
<tr>
<td>12</td>
<td>x₉</td>
<td>1,0,0</td>
<td>2,0,0</td>
<td>3,0,0</td>
<td>4,0,0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>x₉</td>
<td>1,1,1</td>
<td>2,0,0</td>
<td>3,0,0</td>
<td>4,0,0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>