Fault-Tolerant Systolic Array Design With Partially-Reconfigurable FPGAs

Nick Iliev
ECE Dept. Multi-Media Lab
Univ. of Illinois
Chicago, Illinois
niliev4@uic.edu

Dan Schonfeld
ECE Dept. Multi-Media Lab
Univ. of Illinois
Chicago, Illinois
dans@uic.edu

Abstract—Partial dynamic reconfiguration, PDR, is an important feature of modern reconfigurable architectures such as the Xilinx Virtex FPGA devices. In this paper we focus on the application of PDR to fault-tolerant systolic arrays in one dimension. Single as well as multiple faults in one or more array processing elements, PEs, are considered. In our approach modular redundancy is not used and the goal is to maintain total processing latency of the array at the original non-faulty value. This avoids stalling the data input due to reconfigurations of one set of PEs to another set to work around the fault. We consider arrays of N PEs with all possible single-fault locations and formulate an integer linear programming, ILP, model and constraints for each case. The optimal solution selects the PE to reconfigure as a temporary replacement of the faulty PE, so that total processing latency is not increased when the processing clock rate is higher than the data arrival rate. We also develop a heuristic method for solving the ILP problem, which overlaps as many reconfiguration and data processing tasks as possible. The runtime of the heuristic method is very reasonable and matches the exact ILP solution in each case. Experimental results for a single-fault N PE array show that our heuristic finds optimal temporary reconfigurations so that input data does not have to stall.

I. INTRODUCTION AND RELATED WORK

One of the important applications of partially reconfigurable FPGAs is in the design and implementation of fault-tolerant systems. Dynamic (run-time) reconfiguration inside the FPGA allows any pre-defined region of the logic fabric to be reconfigured as a new and different logic function. This reconfiguration is typically initiated by a master controller and involves downloading a new programming bitstream from external RAM down into the FPGA’s rows & columns matrix. Most FPGAs have an on-chip dedicated reconfiguration controller, for example Xilinx ICAP, which performs all reconfiguration tasks. The penalty for a single run-time partial-reconfiguration is the time delay required to complete the reconfiguration. This delay must be accounted for by all processing at the system level. Typically this delay is distributed between the master controller software, the FPGA’s static regions, and reconfigurable regions which remain running during a specific region’s reconfiguration. Recent research reported in [1] attempts to minimize the total reconfiguration delay for functionally-required reconfigurations. However reconfigurations required due to single-faults are not considered. In pipelined systolic array systems, the input data stream may have to stall during the reconfiguration process, or a rate-matching FIFO may have to be used to avoid any data stalls. When a processing element of a systolic array develops a fault, the reconfiguration feature of the FPGA which implements the systolic array, allows one to work around the fault so that data processing continues at the same rate, or at a slightly slower rate.

We focus on systolic 1D array systems with many processing elements. A single fault is expected in any processing element (PE), and dynamic partial reconfiguration of an optimally chosen replacement PE is used to work around the fault. The chosen PE is reconfigured as the faulty PE, processes the data intended for the faulty PE, and is then reconfigured back to its original function. The optimality criteria is the total increase in processing latency: a minimal (or zero if possible) increase is desired.

We do not rely on hardware redundancy to accommodate the faulty PE. Several reports have been published on exploiting hardware redundancy, for example [2] uses a duplication and comparison technique. The reports in [3], [4], and [5] use Triple Modular Redundancy in combination with the partially reconfigurable feature. In our approach to mitigate a single-fault a healthy PE is reconfigured so that it becomes a virtual replacement of the faulty PE. This is then followed by an opposite (or dual) reconfiguration. The reconfiguration process overlaps with normal data processing in the remaining PEs. This is similar to the waveform-like reconfiguration reported in [6] but we focus on fault-tolerance. The choice of PE which is to be used as a
reconfiguration to/from M2's functionality. The choice of M4 for reconfiguration is optimal since it did not increase total processing latency. Other choices for reconfiguration, e.g., M1 or M3 are not optimal since they will increase total latency to 5 steps and data will have to stall. If the fault is in M1 and M2 is chosen for reconfiguration, the total latency can increase to 6 steps. If M3 is chosen total latency increases to 5 steps. In general, for the 4 PE pipeline above, the optimal choice for reconfiguration will result in 0 additional steps, while non-optimal choices will increase total latency by one or two steps. This also applies to N-module pipelines under the above assumptions, as shown in the next sections.

B. Latency Analysis

For a single fault, the additional processing latency for a given schedule (fault location and choice of module to reconfigure) can be described as: (number of reconfigurations) - (number of steps with overlapping reconfiguration and data execution tasks), expressed as

\[ \Delta T = \sum_{i=1}^{M_t} \sum_{j=1}^{M_t} (M_{ij} + M_{ji}) - \sum_{k=1}^{C} N_k \]  

with \( M_{ij} = 0 \) for \( i \neq j \), \( i \) and \( j \) indexing all the module types, \( M_1 \) to \( M_t \). Here \( M_1 \rightarrow M_j \) indicates module \( M_i \) reconfigures to \( M_j \). \( M_j \rightarrow M_i \) is the dual operation. A column index is redundant in this expression. For the above 4-module example, \( M_t = 4 \). \( C \) is the maximum number of steps (cycles) for the given schedule (4 for the example above)

- \( N_k = 1 \) when reconfiguration and data execution tasks overlap in step \( k \)
- \( N_k = 0 \) when there’s no overlap in step \( k \)

Following the example, the first term in the expression for \( \Delta T \) has \( M_{4,2} = 1 \), and \( M_{1,2} = 1 \), with all other \( M_{ij} = 0 \). This is because in step 1, M4 reconfigures to M2 (R114), and in step 3, M2 reconfigures back to M4 (R234). The second summation in (1) evaluates to 2 since in step 1 data executes on M1 which overlaps M_{4,2} (R114) and in step 3 data executes on M3 which overlaps M_{2,4} (R234). This results in \( \Delta T = 2 \) -
2 = 0. The choice of M4 for reconfiguration will result in no additional processing latency.

C. Optimal Reconfiguration – Integer Linear Programming (ILP) Formulation

Expression (1) above applies for a given fault location and for a given selection of a replacement module which has an equivalent schedule of data execution and reconfiguration tasks. For an N-module 1D pipeline, for each fault location, there are N-1 choices for a replacement module, or N*N-1 possible schedules. For a given fault location, one seeks the optimal choice of replacement module (optimal schedule) such that (1) is minimized. This can be expressed as:

\[
\min \Delta T_{ij} = C = \sum_{i=1}^{N} \sum_{j=1}^{M} (M_{ij} + M_{ji}) - \sum_{k=1}^{N} N_k \quad (2)
\]

for all i not equal to j, and for all C, where each schedule has its own value for C. For a given fault location, there are N-1 choices of a replacement module, or N-1 schedules, and therefore N-1 evaluations of (2). The minimal evaluation corresponds to the optimal choice of a replacement module.

We formulate N-1 different ILP problems, with the same fault location and different replacement modules in each case. Using binary 0-1 decision variables, one can express the schedules as follows:

- Variable X_{i,j,k}
  - i=module type, X_1 = M_1, X_2 = M_2, X_3 = M_3,…, X_N = M_N
  - j = time (control) step of the schedule
  - k = column (location) of each module for this schedule

A similar ILP formulation has been proposed in [7], but there the focus is on the hardware-software partitioning problem, to be mapped on the reconfigurable fabric or to software tasks on a CPU. In addition, fault-tolerance is not considered in the ILP constraints found in [7]. Decision variable X_{i,j,k} is 1 if data processing task T_i starts execution on M_k at time step j, in column k. It’s zero otherwise. i equals k when M_i does not have a fault. i does not equal k when M_i has a fault.

- Variable R_{i,j,k}
  - R_{i,j,k} = 1 if reconfiguration for module M_i starts at time step j, in column k.
  - R_{i,j,k} = 0 otherwise

- Variable C = total number of time steps for a schedule

Each schedule is expressed in X_{i,j,k} and R_{i,j,k} variables and is constrained by twelve ILP constraints. As part of these constraints, we have developed two fault-specific constraints such as: for every column, there must be a reconfiguration between two different task executions using this column. The second fault-related constant specifies a precedence for the R_{i,j,k} variables: R_{i,j,k} implies R_{j,k} within C time steps at most. We have used an ILP solver package, LPSolve 5.5, to obtain the exact solution for each schedule. Figure 1 above for the 4-module example represents an exact solution, with all constraints satisfied for C=4, X_111=X_224=X_333=X_444=1, and R_114=R_234=1. All other X and R variables are zero.

D. Heuristic Reconfiguration

The above ILP formulation allows us to search the solutions space for choosing the optimal replacement module, the one which minimizes C, or the equivalent \Delta T. However, it requires significant amount of computation even for a small problem such as the 4 module array above. We define the following heuristic instead: for the case above, where all reconfigurations take the same amount of time, and reconfiguration time equals data processing time:

**Heuristic for equal reconfiguration and execution times:**

For any fault location, choose the replacement module one location upstream from the fault, or, stated differently, choose the one-over location. This will result in zero additional processing latency, \Delta T, due to the choice of the replacement module. When choosing the one-over upstream module, wrapping around the column on the edge is allowed.

The figure for the 4 modules example above illustrates a corner case, where the one-over upstream selection wraps around the first column. The fault is in M2, and the one-over upstream choice for reconfiguration is M4, after wrapping around the M1 column (column on the edge). This optimal choice for reconfiguration still ensures \Delta T = 0 and C=4. The input data will not have to be stalled.

III. SIMULATION RESULTS

We simulated three different pipeline designs, with 4, 8, and 16 modules. In each case data arrives for the task in the M1 column at a rate four times slower than the processing clock. The number of processing steps is 4 for the 4 module case, see Fig. 1, 8 for the 8 module case, and 16 for the 16 module case. A single-fault in a module renders the module unusable. For each design, all possible single-fault locations are considered, and all possible choices for replacement module and related partial reconfigurations are evaluated. The heuristic method is used in each case to find the optimal (minimal total latency increase) reconfiguration schedule. The corresponding ILP problem is also solved to confirm the solution generated by our heuristic. Table I shows parameters of the ILP formulation of the 1D systolic arrays for the 4, 8, and 16 PE case. Table II shows all possible partial reconfigurations for the single fault case, all possible fault locations, and the heuristic’s choice for an optimal replacement module, resulting in \Delta T = 0, no change in total processing latency.
<table>
<thead>
<tr>
<th>Design Number of modules</th>
<th>ILP variables</th>
<th>ILP constraints</th>
<th>Average total ILP solver time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>49</td>
<td>24</td>
<td>0.02325</td>
</tr>
<tr>
<td>8</td>
<td>385</td>
<td>72</td>
<td>0.03125</td>
</tr>
<tr>
<td>16</td>
<td>579</td>
<td>104</td>
<td>0.04275</td>
</tr>
</tbody>
</table>

Table I summarizes the simulation results using the above heuristic. Module Mi-2 is the one-over upstream module for a fault located in Mi. The i=1 and i=2 cases represent wrap-around conditions.

IV. PARTIAL-RECONFIGURATION FOR MULTIPLE-FAULT RECOVERY

The method will always reconfigure for the next step if the faulty module will be required in the next step, while processing data in the current step. For example, for the case of faults in M2 and M3, four reconfigurations can be overlapped with four data processing tasks, with a net increase of 0 for total processing latency, $\Delta T = 0$. Step 1 overlaps X111 and R114, step 2 overlaps R221 and X224, step 3 overlaps X331 and R334, and step 4 overlaps R441 and X444.

V. CONCLUSION

In this paper we presented a design methodology for fault-tolerant one dimensional systolic arrays using partial dynamic reconfiguration. Modular redundancy is not used and the faulty processing element is virtually replaced by an optimal choice of a replacement element. Partial dynamic reconfiguration of the replacement element is overlapped with data processing on other elements so that the overall total processing latency is unchanged or increased minimally. For the case of a processing clock rate higher than the data arrival rate the optimal choice of a replacement element results in an unchanged total processing latency. Future work includes the application of the heuristic for selection of optimal replacement element to multiple fault recovery in systolic FIR filters and systolic 1D DSP processing arrays. A Verilog implementation of a 32 PE systolic FIR array for a Xilinx Virtex-5 FPGA, with multiple PE faults will be considered as a practical case study.

REFERENCES


