Inter-page Network Resource Sharing Model for SCORE

Norman Zhou and Xuejue (Cathy) Huang
{normzhou, xuejue}@eecs.berkeley.edu

CS252 Class Project
Fall, 2000
Prof. John Kobiatowicz

Abstract – This paper details an attempt to quantify the benefits and tradeoffs involved in an inter-page network resource sharing models for SCORE (Stream Computations Organized for Reconfigurable Execution) architecture. Modifications were introduced to the basic network structure to allow resource sharing to better utilize the network resources. Simulation results show that resource sharing reduces the amount of area needed by the programmable network while maintaining comparable performance or from another point of view increase performance/area ratio. For example, by enabling wire sharing, JPEG application obtained comparable performance using a 1/3 sized network. By allowing the freedom of unroutable streams to share contended resources the runtime routing problem is also alleviated.

I. INTRODUCTION

One distinct feature in any reconfigurable architecture is the programmable interconnect network. The flexibility offered by having a programmable interconnect network gives a reconfigurable architecture many advantages over other devices such as ASIC and microprocessors. However the area overhead of having that degree of flexibility in the network is also large. In many programmable devices the amount of die area consumed by the programmable interconnect easily exceeds 50% in some cases even as high as or more than 90% in the case of HSRA.

The richness of the interconnect guarantees that the programmable device can accommodate a range of different demands on the amount of interconnect resources. However it is probably the case that in most case the amount of interconnect that is available is not very well utilized by the designs that are mapped on to the programmable device

Figure 1. SCORE model

SCORE

SCORE (Stream Computations Organized for Reconfigurable Execution) is a stream based compute model which virtualizes reconfigurable computing resources (compute, storage and communication) by dividing a computation up into fixed-size “pages” and time multiplexing the virtual pages on available physical hardware. Compute Pages (CP) are linked together in a data-flow manner with streams. A runtime OS manager allocates and schedules pages at runtime for both computations and memory. Physically a compute page is a fixed-size block of reconfigure logic which is the basic unit of virtualization and scheduling. There is also contiguous block of memory called configurable memory block (CMB), which is the basic unit for data page management. A microprocessor for run-time support and record user code. (Fig. 1) Stream link is a logical connection between the output of one
page (CP, memory segment, processor, or I/O) and the input of another page. Stream implementation will be physically bounded, but the execution model provides a logically unbounded stream abstraction. [Caspi 00]

A virtual stream is mapped on to the physical network when both source and sink are loaded on the physical hardware, or if one end is not resident, the stream data can be linked (or sourced) from a stream buffer segment active in some CMB on the component.

II. INTER-PAGE NETWORK MODEL

The Inter-page network for SCORE is a hierarchical fat-tree. It can be viewed as a binary tree of depth \( \log_2(n) \) where \( n \) is the number of pages and memory blocks. The leafs of the tree are the FPGA pages and memory blocks. The network physically implements the streams connecting the leafs of the tree. Streams within the network are allowed to have variable width. The connections with in the network are synchronous and pipelined so that different routes are timing independent.

The SCORE interconnect is a full 2-ary hierarchical array with shortcut connections like a Fat-Pyramid [Tsu 99]. What differs from a traditional FPGA design are the balance and implementation of retiming resources and the presence of pipeline registers in the interconnect. Hierarchical network was chosen for it’s ability to scale according to Rent’s rule, which can be expressed as:

\[
\text{Number of IO} = C \times N^P \\
0 < P < 1.0
\]

Rent’s rule is an empirical formula which states that there is a geometric relationship between the area of a given circuit and total amount of wires entering and exiting that area. In the above formula \( C \) is the base case IO of a page, \( N \) is the number of pages, and \( N \) is a growth factor that is characteristic of a particular design.

In our particular model we choose a \( P \) to be 0.5 where the number of wires doubles every time you go up 2 level of hierarchy. (Fig. 2, 3) The parameterized growth is 1-1-2-2-4-4.

At each of the switch point in the tree there is an switch box which allows the incoming streams on one set of wires to crossover to another set of wires. In our network model we limited the switch boxes to be linearly populated which restricts the set of wires which one set of incoming wires can connect to (Fig. 3). The reason for choosing a linear switch popula-

Figure 2. Inter-page network model for SCORE. The interconnect level is marked in circle.

Figure 3. Hierarchical fat-tree. The number of wires in each level is marked, following Rent’s Rule.

Figure 4. network with different base channel capacity. Total number of pages is 16 here.
tion is to keep the switch box area from dominating the area cost of the interconnect network. As a result of not fully populating the switch boxes the number of wires at each branch of the tree is no longer the only limiting factor on whether a given graph of streams is routable.

**Base Channel Capacity (BCC)**

Given that the network growth parameter P and the switch population is fixed for this project, the variable that we will use to scale the size of the network will be C, or base channel capacity. C is directly proportional to the number of IO per page in an specific hardware implementation. As shown in (Fig. 4) the more IO assigned to a page, the larger the network becomes. However a larger C also means that the specific implementation can handle applications with more stream connections in-between the abstract compute graph nodes.

**Area Model**

All the areas are modeled based on 0.18 um technology ($\lambda = 0.1$ um). The page size is modeled based on 512 4-LUT with 10% control overhead. Typical area for each 4-LUT in industrial standard FPGA is about $600\lambda^2$. [DeHon 96]

The interconnect is pipelined at level $\geq 4$ (Fig. 2). The level 1-4 are routed using pass-transistors. The size of the registered switch point is estimated as $62.5\lambda^2$ and the pass-transistor switch point as $12.5\lambda^2$. [Tsu 99] (Fig. 5). Global interconnect are used in constructing the network, with a width and spacing = 0.6 um (wire pitch = 12 $\lambda$). As a common design practice, a shield wire is inserted every 4 bus lines to reduce signal-line coupling. The interconnect area is estimated considering the dominate component, either the switch point, or the total width and spacing of the wires themselves. The final area estimation equation for the interconnect network and pages are listed in Table 2. The number of memory blocks is equal to the number of pages. Fig. 6 shows a typical network work area change with base channel capacity, with 32 CPs and 32 CMBs. The total net

**Network Utilization from Profiling**

The way that the post simulation analysis was done was that each stream’s timestamp log was analyzed to see how many times that stream would prevent it’s
source page from producing a token if the stream was throttled to a fraction of it’s maximum bandwidth. ie. At maximum a stream can absorb one token per cycle but if it is throttled by n then it will only absorb 1 token every n cycles. And each time there is a token that the slowed down stream cannot absorb then it counts as a stall. So the higher the stall count the worse a stream performs under throttled condition. So if we try to figure out how well a stream would have performed if we assigned it half of the wires and therefore half the bandwidth the stall count would give us a good indication of how much the performance would suffer.

Table 1. Bandwidth utilization of

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>&lt; ½ speed</th>
<th>&lt; ¼ speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wavelet</td>
<td>62</td>
<td>61</td>
<td>49</td>
</tr>
<tr>
<td>Jpeg</td>
<td>60</td>
<td>58</td>
<td>57</td>
</tr>
<tr>
<td>CCD color conv.</td>
<td>57</td>
<td>19</td>
<td>0</td>
</tr>
</tbody>
</table>

As shown from (Table 1) many of the streams are not using the bandwidth assigned to them in the. In the Extreme case of Jpeg all but 2 of the streams can be throttled to ¼ and therefore take ¼ the resource and still retain comparable performance. A simulation of wavelet with 49 streams throttled to ¼ bandwidth and 12 streams throttled to ½ bandwidth manually ran at the same speed as the full-speed version.

Although post simulation profiling can accurately model the amount of bandwidth needed by the stream and seems to agree with experimental data. It cannot accurately reflect the effect of added latency on the performance.

III. RESOURCE SHARING NETWORK MODELS

From profiling we have found that many of the full speed connections assigned to streams are not being utilized. This leads to the conclusion that resources could be shared in the network. This would lead to less resources need in the network and therefore less area.

Which resource to share is decided at runtime by the operating system. This allows a smaller network to be used with an application that would normally require more network resources. In the case of no sharing the scheduler would have to schedule fewer pages to be physically resident to alleviate network resource conflicts, or in the worst case the design would not run at all.

The ability to share the contending resources allows a given network to handle application that might otherwise take a network with 2 to 4 times as many resources to handle. The fact that many streams don’t need full bandwidth means that a smaller network could easily handle a large demand by sharing resources and in many cases with minimal performance hit.

Another benefit of sharing resources is that the amount of time spent on routing is reduced. As stated earlier the fact that the switch boxes are linearly populated makes finding the optimal routing harder. The current router operates over many iterations ripping out streams with contended resources trying to figure out a possible route. Because of the ability to resolve resource contention by sharing the router is not as hard pressed to find a route. This means that we can operate with a one-pass router and just share the contended resources. This also enables a trade-off between routing time and number of resources shared. (this trade off is not explored in this investigation)

Parallel to Serial conversion
One way of sharing resources is to simply give streams fewer wires. For example in the full speed case a stream of width $w$ would be assigned $w$ wires. If we would like to share resources with $n$ other streams then we could just assign those streams $w/n$ wires each.

This way all of the streams would only be able to send one token every $n$ cycles, but if those streams only need $1/n$ of full bandwidth anyways and the added latency doesn’t matter too much then the performance would be the same while less network resources are used.

To implement this form of resource sharing the amount of extra hardware introduced is minimal. The addition of some MUXes at the output of a page and a DEMUXes at the input of a page are all that’s needed.

**Dynamic Need-Based Sharing**

Another way of sharing resources is to dynamically allocate the shared resource to the streams based on need. In this case if $n$ streams are sharing one resource, then only one will get it, once it has the resource it can send one token every cycle just like a full speed connection, however if the it fails to send a token during one cycle then another stream will get to use the shared resource. Also if the tokens of a stream arrives at a resource that is already allocated the another stream it will have to wait there until the resource is free and then re allocated to it before it can proceed.

The implementation of this form of sharing is much more complicated than the parallel to serial case. The decision of which stream gets assigned to which wire when must be made in hardware. This is because changes to the configurations of a switch box can happen on a cycle-by-cycle bases. In this method before start to send a series of continues (back to back) tokens a page must first send a programming token which will program each of the network switch to the proper configuration to direct the tokens to the right place.

To make this method possible each stream in a page must have to be programmed with the code to correctly set switch box configurations. Also each switch point must have the right circuit to decode the “header” packet from a burst of data and be able to change its routing configuration appropriately. Also the number of pipeline registers must be doubled because of the ability of a token to be stalled halfway across the network.

Dynamic wire sharing works best when the data from various streams sharing the resources come in short busts in between long idles. With this type of behavior each stream essentially sees a full speed network because they each get the network at different times. However if each stream sharing the network stays at a constant rate (for example one token every other cycle) then the parallel to serial sharing with no header overhead and smaller area cost would win.

**Over Clocking**

Another way of reducing network area overhead is simply overclocking the interconnect, so that only a portion of the wires is needed to transfer the data between pipeline stages in one compute-page cycle. This method requires very little overhead except MUXes and DEMUXes at page IOs and a few more pipeline stages. But it requires least two on-chip clock frequencies, larger buffer to drive the interconnects, and consume more power. The result is a network with higher bandwidth per area ratio and almost comparable latency (latency increase is only the setup and hold time of the added pipeline stages). Since the characteristics of an 2X overclocked network with half as many wires is basically the same as a regular network it does not need to be simulated to asses performance. It is also an orthogonal change that can be applied in conjunction to the other model.

**IV. HARDWARE MODEL**

Fig. 7 shows the proposed hardware implementation for the static wire-sharing model. Extra local interconnect and MUXes (or tri-state buffers) are added to the original page in order to cut down the number of bits transferred per cycle. The MUXes are used for converting the parallel data into serial. The control of the muxes (or tri-state buffers) could be implemented in simple shifters.
The extra area taken by this static sharing method is quite small, with only the page size increases slightly.

**Dynamic Wire Sharing Hardware**

Fig. 8 shows the simplified version of the hardware model for dynamic wire sharing. Each stream in a page must have an address register to record the destination address and logic to insert the address information as the first token to a series of data. Also the network pipeline stages are always *tripled*, because when two streams are congested, one has to be saved into an *extra* register while the other one pass through the register. Also because the congestion information (backpressure) has to travel one cycle to inform the previous stages not to send data any more, during which time a new token will arrive, which requires a *third* register to save the data. Also a decoder and router is need to decode the address information and configure the network accordingly in runtime.

The potential area penalty of the dynamic wire-sharing method is significant, with pipeline stages been tripled and extra decoder and router at each switch point.

V. SIMULATOR IMPLEMENTATION

The score simulator is a work in progress, consisting of three pieces of code, the array simulator, the array scheduler, and the user application. The array simulator is the piece of code that simulates the physical array on a cycle by cycle bases. A network model with latency information and bandwidth had to be added. So that each stream can accurately model the effect of sharing resources and network latency. A separate network router written by Randy Huang also integrated with the original SCORE simulator to simulate runtime allocation of resources. (Fig. 9)

*Static Wire-sharing*

Fig. 9 shows the flow of the static wire-sharing implementation based on router feedback. Once the pages are mapped to the physical array a netlist of streams is handed to the router. Then for every network resource with $n$ contending streams, each is throttled to $1/n$ of it’s original bandwidth by assigning less wires. Then for sanity check the new netlist is passed to the router to make sure it will route. In really life the sanity check is not needed and the router will only need to do it’s job once with out the need to tell the scheduler to depopulated the pages to alleviate
Dynamic Wire-sharing

For dynamic sharing, during routing if there is a contended resource then the contending stream would all have to share it. The rule for sharing is whichever has it keeps it until it can’t use the resource anymore then another stream else gets it. This type of allocating power also has the same effect that the router is not forced to try very hard to find a route.

VI. Simulation Results

The baseline we are comparing at is the performance where the page has big enough base channel capacity and no sharing is necessary. This is the best performance that is achievable for a certain application.

Wavelet, JPEG and camera color conversion.

Parallel to Serial Conversion

As predicted from profiling, wavelet and jpeg both with large number of low bandwidth streams were able to share resource to a large degree with out losing performance. (Fig. 10, 11) The very last point on the right side of the graph represents the runtime of the application when there is no resource sharing. Both graphs stay flat for base channel capacity much smaller than that of the minimum requirement for no resource sharing.

CCD color conversion application’s performance degraded faster than the other application. This is also understandable because profiling data shows that the number of full speed streams in the CCD app color is much larger than the number for the other classes. Therefore when assigning streams to less bandwidth it is much more likely that one of the streams that was assigned less bandwidth actually needed more. (Fig. 12)

Dynamic wire-sharing

For dynamic sharing of resources the data that we collected show that the network performs the same for wavelet and jpeg even if the base channel capacity is reduced to a very low level. This data was a little troubling because we were not able get enough information to show that the dynamic simulation model data is correctly obtained. However it is expected that the dynamic method will give better performance at the same base channel capacity. (Fig. 10)
All performance measurements were done in the fully spatial case where all pages are physically present in the array. This was done because when the array is smaller than the number of virtual pages then the scheduler chooses which ones to run. Unfortunately the choice of which page to run has much more impact on the runtime than the network sharing therefore we elected to only show results of fully special computation where the runtime is not affected by scheduler decisions.

**VII. CONCLUSION**

Both post simulation profiling and runtime simulation show that most streams in the present set of applications for SCORE do not fully utilize a full speed connection. The static wire-sharing method greatly reduces interonnnet network area without sacrificing performance.

This proves that resource sharing is a good idea for SCORE, and that it could possibly prove to be a good idea for other designs where the programmable network take up a large portion of the chip area.

Sharing resources also has the benefit of allowing apps with larger network resource requirements to be able to run, and helping the router by not forcing it to come up with a fully routable solution.

The latency increase associated with resource sharing will harm performance of applications with tight feedback loops however the current set of
bentch marcks cantain only feed forward application
with little or no beedback.

**VIII. Future Work**

Currently the streams that the router cannot route are the ones that become throttled. However the router does not give priority to streams with large bandwidth requirements. This could lead to the throttling of a stream that would slowdown the whole computation. On the other hand if the router had a cost associated with routing different nets then we can always get a better list of nets to throttle.

Another interesting study is to see how often (on what time-scale) bandwidth change in different wires. This could show us on what time scale we should be reallocating bandwidth to each stream. If the streams tend to have short bursts on a scale much smaller than reconfigure ration time then hardware might be better. On the other hand if the streams tend to maintain the same bandwidth usage then a software-based allocation will make much more since.

**Acknowledgements**

Thanks to Andre DeHon, Randy Huang, Joseph Yeh and Yury Markovskiy for valuable discussions and helps.

**References**

[Caspi 00] Eylon Caspi, Michael Chu, Randy Huang, Joseph Yeh, Yury Markovskiy, John Wawrzynek, and André DeHon, “Stream Computations Organized for Reconfigurable Execution (SCORE),” appearing in *Conference on Field Programmable Logic and Applications*, August 2000


