A Lightweight Messaging Protocol and Node Architecture for MacroNET

Canturk Isci
Department of Electrical Engineering
Princeton University
canturk@princeton.edu

Li-Shiuan Peh
Department of Electrical Engineering
Princeton University
peh@princeton.edu

ABSTRACT

As electronics technology keeps up its pace in producing much more affordable products with vast amount of newer ubiquitous application areas, the use of embedded electronics devices within mundane equipment, such as clothing, plastics bags, etc. has already started happening and the advent of their widespread use in our daily lives is imminent. Provision of such fabricated flexible and deformable devices is one of the current application interests of large area electronics, and although performance limitations are likely to prevent these devices from ever replacing classical CMOS products, these plastic chips will determine the future of 'ambient intelligence' in devices that are lightweight, low cost, flexible and that scale to millions of nodes. Considering the possible operating conditions of these devices, it is imperative to devise a reliable network that can guarantee packet delivery, handle fault tolerance and still preserve message idempotency.

This paper describes the design of an intermixed CPU and router microarchitecture for a large area interactive map, as an application of large area electronics, with very stringent design constraints in terms of transistor count. The paper discusses the appropriate communication methods, suggests instructions and their formats and a very simple idempotent, guaranteed delivery and fault tolerant routing scheme. Then, providing the selected design for the nodes that satisfies the transistor budget, elaborates on the details of the implementation choices and working principles. Moreover, a flit-time based simulator that can emulate a subset of the suggested instructions is described and the results for various topologies is demonstrated and compared in terms of latency, weight and fault tolerance.

1. INTRODUCTION

One of the current application interests in large area electronics is the provision of fabricated flexible and deformable devices, which can be "rolled like a window shade", particularly for large area displays and sensor arrays[1]. There are already examples of commercial applications as in [5] and [6], and a growing R&D effort such as the European formation 'PolyApply' [4] in polymer electronics. Due to the limitations on size (12'' (300mm) wafer as projected until 2007 by [3]), and requirements for flexibility, the fabrication process involves using amorphous Si (a-Si) – in spite of polysilicon – over plastic substrates. Fundamental drawbacks of amorphous Si (a-Si:H) process include[1][2]:

- Low electron mobility (μ_n(a-Si) ~ μ_n(Si)/500) ➔ lower channel current ➔ longer switching
- Unavailability of P channel devices ➔ No CMOS type structures ➔ Static power consumption
- 10-20 V Vdd requirement
- Min technology dimensions (Half Pitch or Channel Length) are ~ 2μm, which is x20 of CMOS (~100nm)
- Area ratio to CMOS is ~ x400
- The current circuit complexity is on the orders of 10s of transistors
- Gate speed for a-Si technology is around 20ns and power per gate is ~10^{-6}W.

In this paper, we concentrate on one of the obvious applications of these MacroNETs, the interactive large area display, which consists of distributed processing units over the pixels, with distributed sensing and actuation circuitry plus some extent of interconnect processing over the nodes. The major design objective is to provide efficient sharing of information across nodes satisfying a very low transistor budget. As the main application is toward simple human interaction, latency is not the prior concern against throughput, fault tolerance and complexity. A concise description of the design target can be stated as a sparse, lightweight topology for a direct network where local computation and routing logic share resources, with some extent of path diversity, with low node degree and therefore with high average hop count, thus higher latency.

We describe the instructions suitable for such an application, possible communication requirements and their implementations. We come up with a final architectural design for the router and local compute node that can satisfy the transistor count constraint. We describe two routing schemes that we devise to route the defined network instructions, while maintaining guaranteed delivery, message idempotency, fault tolerance and network design simplicity. We also verify our design with our MacroNET SIM network simulator, also presenting comparisons between various topologies and fault injection into the network channels.

The rest of the paper is organized as follows: Section 2 describes the motivation and related prior work, section 3 describes the methodology used to accomplish several

phases of this work, section 4 explains the various types of communications in MacroNET, including packet types and messaging protocol. Section 5 discusses the implementation and working principles of the instructions, section 6 describes the arrived router architecture and transistor count computations, section 7 describes the simulation environment and presents evaluation of different topologies. Finally section 8 presents our conclusions.

2. MOTIVATION AND RELATED WORK:

The particular design constraints for the flexible large area electronics produce different design challenges for interconnection networks. Unlike prior work that focuses on performance, this design is more dependent upon the complexity and fault tolerance. Latency is to be traded for throughput in the latter is to be traded for simplicity to some extent. Other lightweight networks, for on chip processors, mostly strive to reduce latency, which also differ significantly from our current application that focuses primarily on transistor budget.

Examples of router designs for wormhole and virtual cut through switching, for networks of workstations can be found in [7]. On the other hand, [8] presents the design of a high performance router, optimized for multimedia applications and for addressing QoS issues. [9] describes a generic router architecture that provides flexibility in the choice of routing and switching schemes via a programmable routing controller. More distinguished examples include – but definitely not limited to – the IBM SP2 switch ([10] and [11]), Alpha 21364 architecture [12], Spider chip and Chaos router. Other related work include the Transit communication network of Alewife architecture ([13]), which is geared towards low latency, yet still proposing simplifications in the routing element; and the lightweight idempotent messaging protocol described in [14] that suggests the msg/ack/conf triplet for idempotency and guaranteed delivery for network simplicity.

3. METHODOLOGY

Several decisions that need to be made in the implementation of the MacroNET are somewhat dependent on or effect other decisions in different phases. For instance, transistor budget effects the router and processor architecture as well as choices for topology. Compute node model and topology both in turn effect the traffic pattern, while traffic pattern itself plays a critical role in determination of the topology… Figure 1 shows the strong relations between several design choices and constraints.

In order to break these cyclic dependencies, we start by assuming an initial topology. We choose the initial topology to be a torus topology – 4ary 2 cube – as all other topologies that are later considered are bound to be a subset of torus topology, thus providing us with the flexibility of defining new topologies by injecting proper faults in the

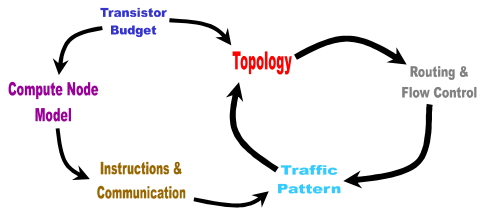


Figure 1, Design Cycle

torus network. Then, we define the nature of communications in the MacroNET under 3 major categories: 1) Communications with the external world: this describes how the instructions are gathered (sensed) from the external world. 2) Network instructions: these are the most important type for our analyses as they define the messages transferred across nodes. 3) Local instructions: these define the instructions that are executed internal to a node. After the conceptual description of communications, we actually propose required packet types and formats to support these instructions as well as to address issues such as guaranteed delivery and idempotency. Afterwards, we conclude the network design phase with a very simple routing model, while still preventing hazards such as livelock. In the next phase, we move to CPU and router modeling and propose a router + processor architecture that is likely to be realized with less than 1000 transistors to satisfy the transistor budget. Finally, we verify our design with the developed MacroNET SIM simulator, while experimenting with several other topologies and fault injection.

4. COMMUNICATIONS

4.1 Input style

There are several levels of communication within the MacroNET. The large area display is assumed to acquire what operation it has to perform via sensors over the pixels and these external signals are converted to interprocessor instruction packets by the nodes. Hypothetically, these external stimuli are similar to mouse clicks on an interactive map or strokes of Mentor Graphics. The interprocessor messages are propagated based on the instruction performed and the routing principles to be described shortly. We also reserve the possibility of intraprocessor instructions, however we do not expatiate on them as they are not the focus of this paper. The set of network instructions we propose are:

- zoom around node(r,c)
- dim / brighten
- Maketop N/S/W/E
- Compute Distance

The input method for all but compute distance is, the user clicks on a cell, the corresponding node's compute unit composes the message and broadcasts to the nearest neighbors. For compute distance, first clicked cell broadcasts a similar message, while the second clicked cell

sets a status flag and waits for the compute distance message to arrive with the source coordinates.

4.2 Packet Types and Communication Protocol

For the packet types, we first determine the flit size as 8 bits, moreover the channel width is also considered to be 8 bits, therefore flits and phits are identical units. Unlike [10], we don't prefer to add additional flow control channels and we don't include routing information in messages. While instructions are decoded along the processor datapath, the routing decision will also be made. Similar to [12] message types, we provide several classes of packets and we accomplish this by tagging the header flits. As discussed in the communication protocols for the Transit network of Alewife architecture ([13]), the ability of sender to retry failed communications avails significant simplifications in routing unit. Moreover, although explicit Acks seem to slow down the network, they are required for guaranteed delivery. Our messaging protocol follows a similar strategy, but during broadcast, a missing Ack does not initiate a retry, but prevents the node from using the channel instead – analogous to an infinite timeout. In the case of directed routing, a timeout mechanism is still used. [14] identifies a fault tolerant messaging protocol should have guaranteed delivery and idempotence properties. Besides, any messaging protocol should in general have either livelock avoidance or livelock recovery property. In our protocol, we follow the message/Ack/Confirm strategy of [14] and we suggest an additional 'Acked' packet to avoid livelock, while broadcasting. The descriptions of the three control packets are as below:

1) Ack: (rcvr → sender) If the receiver received the message and wants the succeeding flits as well. Also handles flow control as the receiver sends Ack when it no longer needs the current flit, thus providing bufferless flow control. This signal is sent once per flit.
2) Acked: (rcvr → sender) If the receiver received the message, but doesn't want the rest of the message. This signal can be sent once per message at most.
3) Confirm: (sender → rcvr) Sender sends this packet after it receives the Ack for the last flit of the message – like an additional tail flit. Receiver can release all resources for the current message after receiving Confirm. This signal is sent once per message.

The encodings for the above described packet types are as shown in table 1.

Local instr-n:	[000|xxxxx]
MSG:	[abc|xxxxx][xxxxxxxx][…]
Special Packets:	
ACK:	[111|11111]
ACKED:	[111|10101]
CONFIRM:	[111|01010]

Table 1, Packet Types

The first 3 bits of the header flits are 'tags' and describe the type of the packet. As can be deduced from the table, "000" tag encoding is reserved for local instructions, "111" is for special packets and anything that lies in between is a network instruction type.

5. INSTRUCTIONS

In this section, we first describe what each instruction is expected to do and then describe the packets related to each instruction and explain how they accomplish their tasks across the network, via a few examples. In our design, all instructions whether local or network, pass through the processor datapath. At the decode stage, the instruction tag is decoded first to identify instruction category, and the rest of header flit either being extra node info, subop for local instructions or specific encoding for special instructions will enable the controller to take appropriate actions. Here we only describe the network instructions tagged from 001 to 110, as they are the main concern of our work. We also note down required architectural components to be able to perform these operations, which direct us in the architecture design phase.

Zoom Around (r,c) <001>

First flit defines the instruction and tells the source of instruction. 2nd flit tells the neighbor cell either to keep its current color or to change it to the specified color, similarly 3rd and last flit tells the neighbor cell either to keep its current brightness level or to change it to the specified level. The decision of changing the color or keeping is made by comparing the source coordinates to the node's own coordinates. Therefore, each node should hold its color, brightness and coordinate states. Under this description, the packet format for the zoom instruction is as follows:

```
Zoom Around (1,2):
 [001|01|10|x][color/keep][bright/keep]
```

As an example, zoom(2,1):

(1): (2,1) sends flit 1 to all 4 neighbors ➔
All 4 neighbors identify the instruction and generate their own headers as the same header
(2): All 4 neighbors send back Ack to (2.1) ➔
(2,1) makes ready the second flit, and discards the header
All 4 neighbors send the headers to all neighbor nodes except (2,1) ➔
Other nodes do as in timestep (1), but also produce Acked's for duplicate messages
(3): (2,1) sends 2nd flit: color/keep ➔
All 4 update/keep their colors
(4): All 4 neighbors send back 2nd Ack to (2.1) ➔
(2,1) makes ready the third flit, and discards the 2nd
(5): (2,1) sends 3rd flit: brightness/keep ➔
All 4 update/keep their brightness

(6): All 4 neighbors send back 3rd Ack to (2.1) ➔
(2,1) makes ready the confirm flit, and discards the 3rd
(7): (2,1) sends confirm flit and resets channel states ➔
All 4 receive confirm and reset their channel states

As the example assumes, there are several states that need to be kept throughout the processing of an instruction. Firstly, although we are broadcasting, we send the original sender back Ack rather than bouncing back the message, therefore, we need a mux and a selection scheme in front of each output channel that chooses from the output register or Ack/Acked/confirm. The sender node should keep a status to specify whether the last flit is Acked or still waiting or an 'Acked' is received and should send consequent messages based on the status or hold back. The receiver node should use active vs. Acked status fields to distinguish the sender from future receivers as well as providing idempotency. All these requirements are handled and clarified in the router architecture phase.

Dim/Brighten <010/011>

Both these instructions are single flit messages, with only the tag information. The flits are broadcasted across the network, and the message ceases to exist after Acked signals. The operation is similar to zoom, however no source information is required. The example is not provided for brevity. The encodings are as follows:

- `Dim/Brighten: [010/011|xxxxx]`

Maketop N/S/W/E <100 | 00/01/10/11>

One of the more complicated operations is maketop. Maketop is used to rotate or flip the display by matching the physical top of the display with any of the 4 directions north, south, west or east. The implementation of maketop thus requires another state field for each node 'TOP' to keep the current orientation. The obvious way to implement maketop is, each node forwarding the received maketop instruction as well as producing a second one that includes their color and brightness info. However, further elaboration of this with broadcasting reveals a case of livelock as sketched in figure 2. For a set of updated nodes, shown shaded, they still keep forwarding the maketop flits destined for other nodes such that, the message traffic never ceases.

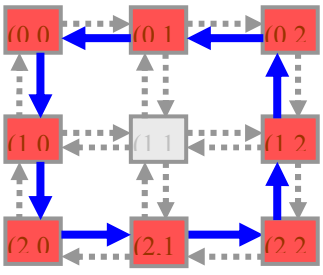


Figure 2, Livelock in Maketop

To produce a livelock free maketop, we need to dissect the maketop instruction into two parts: 1) A uniform maketop instruction, that can be safely broadcast as in previous instructions 2) An exchange instruction specific to each node, which carries the destination and the node info to be exchanged. The node which receives the initial maketop instruction from external world first compares the desired top to its current top and as long as they are different, produces the maketop instruction and sends to all its neighbors, while also preparing its exchange instruction. The neighbors Ack the message and forward to their other neighbors, while also preparing their exchange instruction. After getting at least 1 Ack for the maketop, each node starts sending its exchange message using a scheme similar to dimensional routing. Hence, with directed routing, to provide fault tolerance, we need a timeout timer while waiting for Acks, and a timeout status field for each output channel to identify faulty channels. Moreover, this scheme can still produce livelock in very unlikely cases of sets of broken links. To accommodate for this hazard, we can still set timeout field for an output channel even though the entire message is successfully sent over it, to prevent the message which looped back to be sent through the same cycle, thus breaking the cycle as described in [15]. The formats for maketop and exchange are as shown below. For maketop first 3 bits are the tag, the following 2 bits are the direction to make the physical top. For exchange, first 3 bits are the tag, following 4 bits are the row and column fields of the destination node. 2nd and 3rd flits are the color and brightness level of the source respectively.

- `Maketop S: [100|01|xxx]`
- `Exchange: [110|01|11|x][color][brightness]`

Compute Distance <101>

This is also a single flit instruction. The source node sends the compute distance network instruction along with its own coordinates, and destination node waits for the instruction to arrive with the source coordinates and computes its relative distance to the source. Hence, to perform this operation, a general status flag for each node, that tells the controller that it's waiting for the source coordinates, is required, to distinguish the destination node from any other node. Moreover, for the distance computation here and the destination computation for exchange instruction, we need a simple arithmetic unit that, at minimal, supports addition. This instruction can either be broadcasted as it's identical among the nodes, or can be routed directly to destination. The encoding for the compute distance instruction is shown below. The first 3 bits specify the tag, and the following 4 bits identify the source coordinates of the instruction.

- `Compute Distance: [101|11|00|x]`

6. CPU AND ROUTER

The compute nodes in MacroNET are a composition of a simple processor and auxiliary hardware for routing. The

processing and routing components are highly convoluted. All package headers are instructions that go through the decode stage and walk through the datapath. Decode stage identifies instructions based on their tags. In maketop operation, unidentical packages arrive simultaneously at nodes, therefore we need a simple arbitration scheme, but as the router does not propagate two different messages at a time, this is a simple selection scheme like LRS. As for the exchange instruction, we need a simple routing logic that is incorporated within the node controller. The datapath requires a simple arithmetic unit, and similar to [16], we need status vectors for each channel as well as a global one for the node itself.

Our router architecture is mostly optimized for broadcasting with minimal possible hardware due to stringent transistor budget, and therefore trading ingenuity for simplicity. We use a single cycle, in order datapath, and do no buffering. However, we still need to keep certain node info as mentioned along with the instructions. As already mentioned in related work, other examples in literature ([7], [8], [9], etc) focus on other issues, however a detailed description of the complete design flow for a CPU + router can be found in the Stallion Chip implementation thesis project [17], which includes several design levels from architectural to transistor level and layout. Nevertheless, referring to [18] and [19], we cannot even fit an 8 bit wide 4x4 crossbar into our 1000 transistor budget as a single bit slice 4x4 crossbar, shown in figure 3, uses 640 transistors ([18]) and our desired crossbar itself costs more than 1000 transistors ([18], [19]).

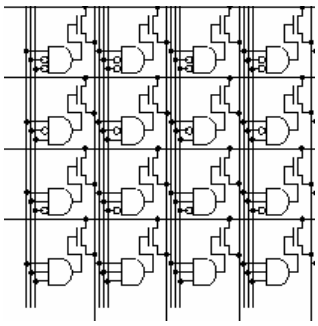


Figure 3, 4x4 crossbar bit slice

Consequently, we have to devise alternative ways of switching, which are less costly. Taking the memory based buffering structure in [16] and the central buffer of the IBM SP2 high performance switch [10] as our start point we implement a memory based switching structure, with static data structure as shown in figure 4. The allocation scheme is LRS and is conducted by the process queue. 4 registers are connected at the downstream end of input channels. The flow control scheme can be considered as credit based flow control with a maximum credit count of 1, i.e. if Ack is received credit is back to 1 and upstream node sends the next flit, otherwise credit drops to 0 and upstream is stalled. Network interface is register mapped, however, as the registers are fixed with the 4 incoming and 1 shared outgoing register, it is actually a 5 register network interface.

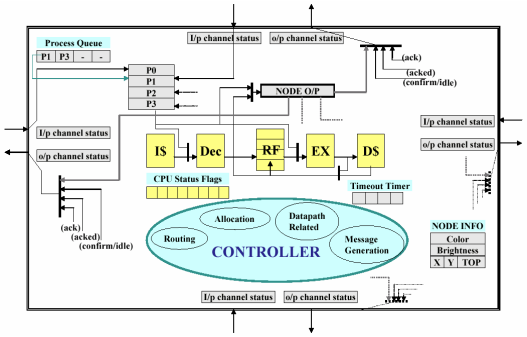


Figure 4, Router Microarchitecture

Each incoming message passes through the decoder and in case the received flit will need to be propagated as is, controller selects the input to the output register directly from the current processed input registers, otherwise if the packet needs to be processed before forwarding, node output comes from the EX output. Moreover, at least one GPR is required for serial internal operation, and controller selects the execution input either from the local Regfile or current register input. Each input and output channel have dedicated status vectors, which help controller to take port specific actions as described in the instructions section, moreover there is a global status vector, to specify node specific states, i.e. as exemplified in compute distance section. Each node also keeps its color, brightness, X,Y coordinates, and current Top as required by the instruction implementations. The output channels have output selection for register output, Ack, Acked and confirm, to be selected by the controller. The status vector fields are described in table 2, and an expanded version of the router architecture is also included in the Appendix.

I/p channel Status Fields [I|R/W|A|Acked]	
I:	Idle <Default>
R/W:	Routing or waiting for its turn
A:	CPU processing packet or sending downstream
Acked:	Another port already rcved/processed same msg

O/p channel Status Fields [Ack|Acked|Timeout|W]	
Ack:	Downstream idle (Acked last flit) <default>
Acked:	Downstream Acked'ed last packet
Timeout:	Downstream's last awaited ACK timed out
W:	Waiting for ACK

Table 2, Status Vector Fields

6.1 Transistor Counts

Regarding the processor architecture in figure 4, we have 13 8 bit registers, 1 4 bit timer, 3 2bit state fields. For the described datapath, I-cache and D-cache may be redundant, while we need at least one internal register for serial computation intermediate results. For the execution unit, we need at least an adder. Additionally, we have some combinational logic for decoder. For the control, we presume not too many states, probably far less than 100, therefore an 8 bit register will suffice with probable margin for reduction. Also the next state encoder and output decoder for the controller will not be very significant considering the small amount of control signal it needs to drive. Our transistor budget is in the order of 1000, and according to [20], around 10% of this budget is consumed for padding. Therefore, our aim is to fit the above architecture within an order of 900 transistors. For the process constraints, as there are no PMOS transistors are available, CMOS like designs are not feasible. Therefore, possible design methods include dynamic logic, only using n channel transistors with inherent delay in successive precharge lines to avoid undesired discharges – as we cannot apply n-p domino logic; or pseudo NMOS logic, with the p channel pull up transistor replaced with an n channel transistor. Hence, both of these design choices suffer severely from static discharge and V_T drop of n transistors, and our EMD people probably have already developed much efficient alternatives to these. Therefore, although having stated there are no PMOS devices, we still provide classical CMOS designs as possible design alternatives, to give an idea of approximate transistor costs.

In our architecture, registers are the most commonly used blocks, and therefore any rigor in transistor reduction should be directed to these. Possible implementations of registers are non overlapping 2 phase flip-flops that are used in [17] design, with only pass gates, which require only 6 transistors per bit, but the 2 phase clock generation scheme itself is complicated. C^2MOS requires 8 transistors per bit, and still needs 2 phase clocking – though not strictly non overlapping. Another alternative is the use of true single phase flip-flop as shown in figure 5. This design has transparency problem when clock is high, but is the most compact and is our implementation choice. With this implementation, the total cost of all registers add up to 660 transistors.

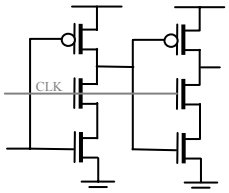


Figure 5, True Single phase FF

For the execution unit, the only required logic is that of an adder / subtractor, and as general subtractor implementation is also an adder with last input vector inverted and an injected hot 1, in 2s complement, we here only describe the adder implementation. Subtraction can be added with an inverter and add/sub control input serving as hot 1. We suggest a serial adder where the intermediate result is stored in the internal register. The most compact adder architecture is the transmission gate adder, which can even further be reduced for pass gate logic, as shown in figure 6. With this implementation we can realize the adder in less than 20 transistors.

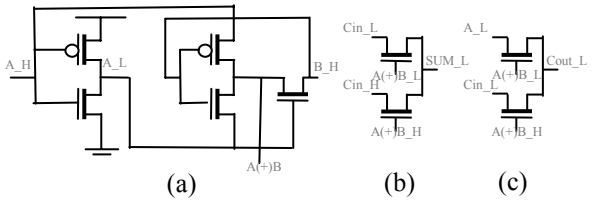


Figure 6, a)TG XOR, b)SUM logic, c)Cout Logic

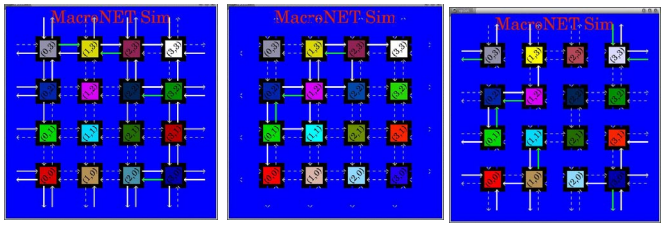
For the controller, the important portion of the design is the state registers, and as stated above, an 8 bit register will be more than necessary. Therefore, for the sequential portion of control logic, less than 50 transistors will be sufficient. The remaining combinational logic, mentioned above will be of low transistor count.

For the rest of the architecture, we can apply various trade-offs such as completely discarding the caches, limiting the regfile to a single register, etc. The budget constraint for the remaining combinational blocks, decoder and NSE and output decoder of controller is around 150 transistors, which is reasonable to achieve with area/transistor based synthesis optimizations.

7. SIMULATION

In order to verify our implementations for instructions and communications, we implemented a flit-time based simulator named MacroNET SIM. The simulator is initially designed for torus topology as a 4ary 2 cube, and fault injection to channels are implemented to verify fault tolerance as well as providing support for topologies as a subset of torus topology. As the node latencies are dependent on process and implementation techniques, we have considered only channel latencies in terms of flit times, which we use to assess network latencies for different topologies. Main principle in the simulator implementation has been trying to keep as loyal as possible to actual decision making strategies, therefore, we implemented similar status fields and similar signaling as discussed in previous sections and verified our described instructions perform correctly within the network. We also implemented a simple gui for demonstration purposes.

For latency analysis, we look at 2 different figures: 1) time for all nodes to update themselves, as this is the actual observation from external world, 2) time for the whole packet traffic to cease, as this validates livelock freedom and enables the network to accept new input from external world. For the 3 topologies considered, as shown in figure 7, the acquired results in flit times are as shown in figure 8. For the acquired times, we consider a single header flit, and we provide the times for the worst case, for the best case and averaged over simulations for each possible node, as for mesh and sparse torus topologies, due to asymmetry, the node from which the instruction starts effects the final time.



(a) (b) (c)

Figure 7, a)TORUS, b)MESH, c)SPARSE TORUS

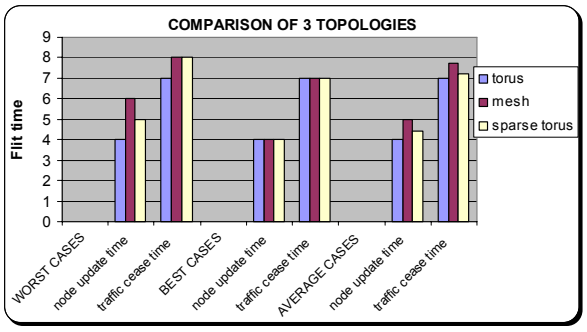


Figure 8, Acquired Latencies for the 3 topologies

As seen in figure 8, torus network performs best for both latencies, however the more important comparison is between the sparser topologies, and sparse torus topology performs significantly better for both worst case and average case analyses. For the best cases, all topologies perform equally well, as several paths send the instructions simultaneously and they all share a common subset of sufficiently quick paths.

Next, we consider fault tolerance. As the above 3 networks already demonstrate, the messaging scheme has significant amount of fault tolerance. Moreover, when we consider additional fault injection to critical path in the above networks, torus is indifferent to fault injection, as there are always multiple equally fast paths to the bottleneck node. For sparse torus, a single fault injection to the critical path causes the worst case latency to increase to 6 and obviously 2 fault injections can break the network, as most nodes have connectivity 2. As for the mesh topology, a single fault doesn't effect the network latency, however similar to sparse torus, 2 faults can break the network.

We have also formulated the worst case latencies and number of links as a function of network size (nxn) as shown in tables 3 and 4, for a more analytical approach and verified our results with an 8x8 network as well. As can be deduced from table 3, the latency of sparse torus exceeds mesh for n > 6, and table 4 reveals that for n>3, number of links for mesh exceeds that of sparse torus.

Worst Case Latencies	
Torus	n
Mesh	2n - 2
Sparse Torus	(5/2)n - 5

Table 3, Network Latencies

Number of channels	
Torus	$4n^2$
Mesh	4(n-1).n
Sparse Torus	4[(n-1).n/2 + n]

Table 4, Number of links

Consequently, our simulation and later formulation reveals, sparse torus topology incurs additional delay resulting from additional channel absence compared to mesh and torus. However, the average behavior of sparse torus is yet closer to that of mesh even with large number of nodes, and can be a worthy trade-off for the MacroNET considering additional savings from node connectivity. However, fault tolerance of sparse torus is not robust as mesh or torus, and can be a caveat in real applications.

8. CONCLUSION

In this paper we have explored a fresh area of interconnection networks, namely MacroNETs, where major design constraint is the transistor budget. We have concentrated on a large area display as an obvious application to MacroNETs and described a suitable set of instructions for the application. Assuming an interactive input method, we showed a generally broadcasting mechanism with minor routing tweaks for certain instructions works well for such a system. We also provided different types of packets, to provide both livelock free and idempotent messaging under broadcast. We applied a tagging mechanism to identify instruction classes and enabling each instruction to be decoded in the same datapath decoder. We have shown, small number of flits suffice for a large set of instructions and all decisions can be made with only the header flit. Although our broadcast method simplifies nodes and routing, the amount of packet traffic is tripled, which might be an obstacle in extending this work to a simultaneous instructions framework. We

have presented an architectural design for the lightweight node, and noted the required node states and network status fields to support the current set of instructions, unlike most contemporary designs, our routing node cannot make use of a crossbar for switching due to transistor budget, and we suggest a memory based buffering and switching strategy, with explicit Acks for each flit, thus slowing down the traffic significantly. Our architecture represents an extremely primitive approach to router design, with flitwise flow control, fixed register based network interface, static data structure and a single shared output register. Several well-known implementation methods are chosen for the transistor level implementation, and a final transistor count estimate is provided for unaccounted for units. The conclusion from this experience had been mildly frustrating as the projected transistor budget is hardly sufficient for a reliable design. We also presented a simulation environment, MacroNET SIM, to evaluate three topologies and network fault tolerance. For low node count, sparse torus topology is seen to be superior over mesh, however an analytical approach revealed as number of nodes increases, although the channel count of sparse torus maintains to be much lower, the latency becomes inferior to mesh, increasing linearly with n.

We note that our discussion of MacroNET topologies is preliminary at best, as the space of possible topologies is enormous, a more comprehensive analysis was beyond the scope of this paper. However, we believe, using the flexible fault insertion scheme and scalability of MacroNET SIM, a more exhaustive analysis of topologies can be performed over a longer period of time, as an extension to the presented work.

9. REFERENCES

[1] Project Description document, Prof Peh

[2] Amorphous Si TFTs on Plastically-Deformed Spherical Domes, Hsu P.I., Gleskova H., et al, ICAMS19, 2000

[3] ITRS 2001 – Front End Processes

[4] Europe quietly forms polymer electronis project, EETimes, February 2003

[5] Electronic Ink, www.eink.com

[6] Power Paper, www.powerpaper.com

[7] A comparison of Router Architectures for Virtual cut-through wormhole switching in a NOW environment, Duato J., Robles A., et al, IEEE IPPS 13th & SPDP 10th, 12-16 April 1999

[8] MMR: A High performance multimedia router – Architecture and Design Trade-offs, Duato J., Yalamanchili S., et al, HPCA 5, 1999

[9] A router architecture for flexible routing, Daniel S., Shin K., et al, IEEE transactions on PADS, vol. 10, no. 1, January 1999

[10] The SP2 high Performance switch, Stunkel et al, IBM Systems Journal, vol. 34, no. 2, 1995

[11] Assessing the performance of the new IBM SP2 communication system, Miguel J. et al, IEEE parallel and distributed technology, pp. 12-22, 1996

[12] The Alpha 21362 Network Architecture, Mukherjee S., et al, IEEE Micro, 2002

[13] Technologies for Low latency interconnection switches, Knight T., et al, Symposium on Parallel algorithms and architectures, March 1991

[14] A lightweight idempotent messaging protocol for faulty networks, Brown J., et al, SPAA'02, August 2002

[15] Interconnection Networks, William James Dally & Brian Patrick Towles

[16] Hardware microarchitecture presentation, Canturk

[17] VLSI implementation of a Run-time configurable computing Integrated circuit – The Stallion Chip, He Y., Thesis, Virginia Polytechnic Institute

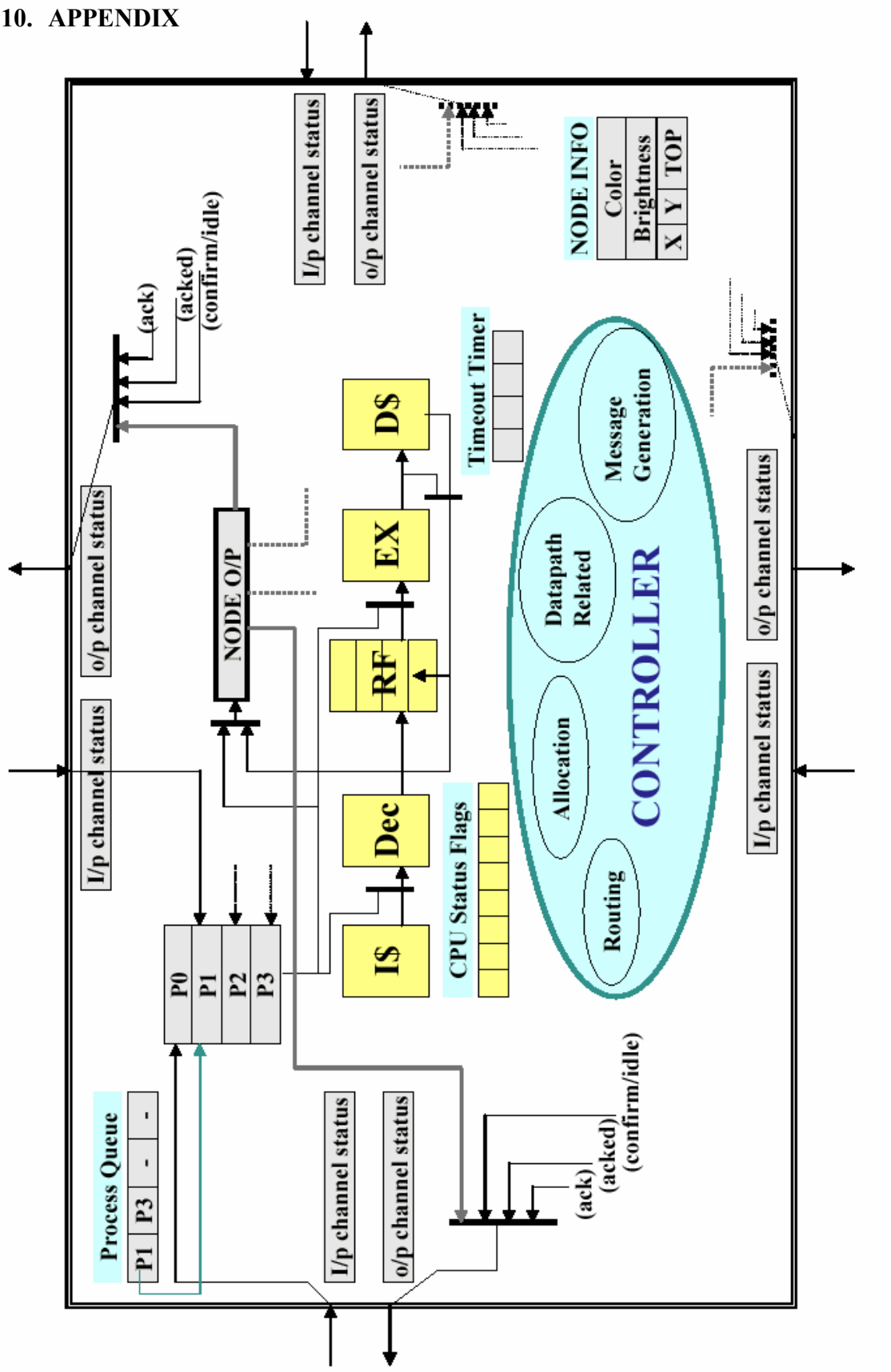
[18] Fast Subword Permutation Instructions Using Omega and Flip Network Stages, Yang X., et al, ICCD 2000

[19] Power, Area and Speed comparison of Omega and full crossbar networks, Jacobson N., et al, 2001

[20] RAW microprocessor, Taylor M., et al, IEEE MICRO, March 2002

[21] Principles of CMOS VLSI Design, Eshragian K., 1996

10. APPENDIX



Router Microarchitecture