A Time Predictable Heterogeneous Multicore Processor for Hard Real-time GALS Programs

Zoran Salcic, Muhammad Nadeem, Bjoern Striebing
Department of Electrical and Computer Engineering
University of Auckland, Auckland, New Zealand
z.salcic@auckland.ac.nz, muhammad.nadeem@auckland.ac.nz, bstr566@aucklanduni.ac.nz

***Abstract*—Time-predictability of the execution architecture is a key requirement for successful implementation of hard real-time systems. This paper presents a scalable and fully time-predictable multicore processor, TP-HMP, that comprises of two types of time-predictable cores and a novel NoC interconnect that are used in creation of heterogeneous multiprocessor FPGA SoCs. The NoC interconnect, called TDMA-MIN, is based on a combination of time-division multiple access and multistage interconnect resulting in very good throughput and bounded latency. TP-HMP is used as the execution platform for hard real-time programs written in the GALS language SystemJ. The platform allows application of static timing analysis to SystemJ programs and can be used in design space exploration targeting multiple cores. As such, TP-HMP is the first real-time capable multicore processor supporting execution of SystemJ GALS programs.**

Keywords—component; time-predictability; multi-core; GALS MoC; SystemJ

I. Introduction

Real-time and safety critical systems combine complex control flows, reactivity and concurrency and at the same time may have complex data computations. Multicore SoCs have emerged as most promising candidates for implementation of such systems because of ability to achieve required real-time performance at lower operating frequency with reduced power and energy consumption. They also allow use of different cores targeting different types of computations, e.g. combination of control-dominated and data-dominated computations. For hard real-time systems, correctness of a program not only depends on the correctness of computations, but also on the ability to deliver results on time. Many real-time systems can be conveniently described as collections of concurrent processes (tasks) and can be modelled as a Globally Asynchronous Locally Synchronous (GALS) system, where the asynchronous processes run at mutually unrelated speeds and communicate each with the other. These processes are conveniently specified/designed as a collection of locally synchronous processes, which run at the same speed, using the same logical clock, and can be checked for formal properties and timing constraints.

A promising design tool for GALS systems is the system-level programming language SystemJ [1]. Concurrent processes are described as synchronous reactions that are composed and grouped into asynchronous clock domains (CDs). The logical clock period (tick) can have different execution times and one with the maximal length is referred to as a Worst Case Reaction Time (WCRT). The WCRT represents the maximum time in which SystemJ CD will respond on any event coming from the external environment. Using static analysis tools, Guaranteed Reaction Time (GRT) of clock domains can be calculated [2] when executed on time analysable and relatively simple time-predictable platform.

The real-time properties of multicore processors rely on the time predictability of individual cores and the mechanism used for their communication. Therefore, for a real-time multicore processor, deterministic and statically predictable behaviour of the interconnection network has to be guaranteed. The interconnection network enables creation of Network on Chip (NoC). The inter-core communication in systems with many cores is no longer feasible by using traditional techniques like shared buses due to their low intrinsic scalability, particularly low throughput (bandwidth) and high latency. Traditional buses need additional mechanisms to guarantee time predictability of accesses by multiple cores, e.g. using variants of Time Division Multiple Access (TDMA) scheme, resulting in further reduced bus utilization.

Mesh networks are characterised by regular layout, all nodes are typically placed in a square grid. Each node is connected to a router via a local link. The routers have another four bidirectional links, connecting to neighbours in north, south, east and west directions. An even more regular layout with identical routers is achieved when links across the borders of the network are established, e.g. torus or bi-torus networks. Most developments on mesh based NoCs are aimed at fast best-effort routing, which are unsuitable for real-time applications as certain links might be congested, causing unacceptably long latency [3].

Multistage Interconnection Network (MIN) is a feasible approach to implement connections among cores and memory modules. The bandwidth division is made in the best possible form among different partitions of a MIN. Unfortunately, models for average network delays in buffered and un-buffered

MINs [4][5] do not provide guaranteed bound on worst case latencies.

In this paper we introduce a time predictable NoC interconnect, called *TDMA-MIN*, which combines TDMA and MIN in a novel way with fixed bound on communication cost for any number of used cores or other nodes that are accessed through the same interconnect. TDMA-MIN is used to create a *time-predictable heterogeneous multicore processor*, TP-HMP, which is an ideal execution platform for hard real-time programs written in SystemJ.

The rest of the paper is organised as follows. Section II describes the related work. Section III introduces briefly SystemJ program execution strategies based on separation of control-dominated and data-dominated computations on two types of processor cores. Section IV introduces TP-HMP and Section V time predictable TDMA-MIN NoC interconnects. Section VI presents evaluation of performance of TDMA-MIN and compares it with other interconnects suitable for real-time systems and compares them in terms of throughput, latency, operating frequency and cost. The paper concludes with Section VII.

II. Related Work

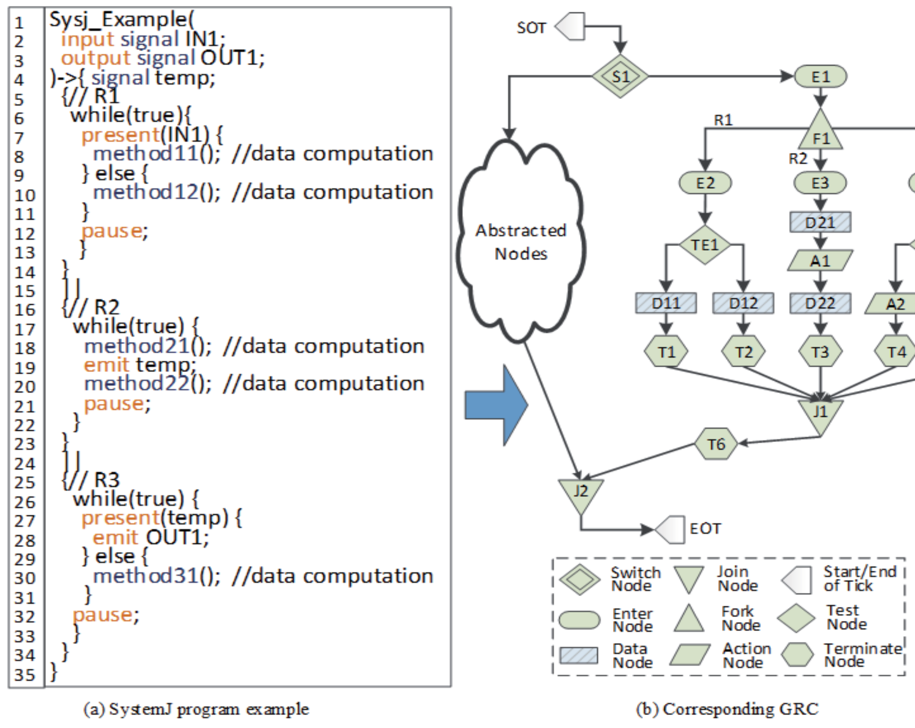
Standard SystemJ program compilation produces Java code that can be executed on any Java Virtual Machine (JVM), but it is not suitable for real-time systems due to unpredictability of JVM execution times. There have been several approaches to develop scalable and predictable execution architecture for SystemJ based on compilation approach that separates control from data computations. The first platform that used this approach for hard real-time systems is the Tandem Processor (TP), TP-JOP [2], which uses a custom-made reactive processor, called ReCOP, as a Control Processor and Java Optimized Processor, JOP, [6] as a Data Processor. Both ReCOP and JOP have time predictable execution. Another processor that integrates ReCOP and JOP into single core, called JOP-Plus [7] has the same execution flow as TP-JOP is more efficient in terms of required implementation resources and is real-time capable. TACO [2] framework calculates the worst case execution/reaction time of synchronous (single clock-domain) SystemJ program when executed on the TP-JOP or JOP-Plus.

More complex embedded systems require multiple cores such as GALS-HMPs [8]. Utilising multiple control and multiple data cores allow the parallelisation of the execution of the program and increase of overall performance. However, GALS-HMP uses ReCOP for control-dominated parts of SystemJ program and Altera Nios II core which requires a JVM and hence is not real-time capable for data-dominated computations. Also its interconnect architecture is not time predictable. GALS-CMP [9], based on JOP-Plus cores, is used for the execution of SystemJ programs but does not allow parallel execution of control dominated and data dominated parts. The GALS-CMP follows the methodology adopted by the time predictable multiprocessor system JOPCMP. Both GALS-CMP and JOPCMP [10] use the TDMA based interconnect with communication time increasing proportionally with the number of cores. TDMA based schemes when statically scheduled are predictable but result in underutilisation of resources.

Traditionally, buses have been used to connect the multiple cores but they have low intrinsic scalability in bandwidth and latency [11]. In indirect networks, computing nodes are connected via a cascaded set of switches. The construction of a MIN needs only $(N/2)\log_2 N$ switches, which is the minimum number possible at all. Here, N is the number of nodes connected. For comparison, a crossbar network requires N^2 switches [11], MINs are fully reachable, but blocking in nature, therefore, constant latency cannot be guaranteed. Non-blocking MINs provide non-blocking path arrangement but no communication can take place during path arrangement. Solution is to use buffers, but it requires draining before re-arrangement introducing non-deterministic delay. Also, the buffered MINs do not scale well. TDMAs schemes when statically scheduled are predictable but result in underutilisation of resources and very low bandwidth. Deferring decision to run time enables higher performance but makes timing analysis more difficult, if not impossible. Æthereal uses TDMA, contention-free scheme and a distributed routing model but suffers from high cost and requires configuration [14].

III. SystemJ and its Execution

Every SystemJ program [1] can be represented in the form of Asynchronous GRaph Code (AGRC) graph, which is used to generate object code for the target execution platform. An example of a typical graph for a small single clock domain program, which is effectively a synchronous program and can be represented by a subset of an AGRC called GRC, with three synchronous reactions R1, R2 and R3, is shown in Fig. 1. GRC describes control flow of the program where individual nodes within reactions define reaction's control flow. All branches of the graph between a fork and its corresponding join node belong to reactions that do not have inter-dependencies and can execute in parallel. The parallel reactions can be executed in any order with or without node interleaving, as long as the order and control flow of execution of nodes within each reaction is preserved. At the same time, fork and join node are the boundaries of a single logical tick of synchronous program and each node in the GRC represents its schedulable unit. All the nodes in a GRC can be categorized into two groups: (a) the data nodes representing data computations which are compiled to Java methods, and (b) the control nodes representing control-driven computations that define the control flow of individual reactions. SOT and EOT nodes conceptually represent the start and end of each logical tick of the clock domain. The switch node S1 decodes the state of the program. Parallel reactions R1, R2 and R3 are forked out by fork node F1, and combined by join node J1. All parallel reactions have to complete their own tick within the tick of a clock domain. Control flow specified by the subgraph that represents a reaction must be completed from the beginning to the end of each reaction. Enter nodes (from E1 to E4) perform state encoding for switch node S1. Each method call in the program is mapped to a unique data node, for



```
1  Sysj_Example(
2   input signal IN1;
3   output signal OUT1;
4  )->{ signal temp;
5   {// R1
6    while(true){
7     present(IN1) {
8       method11(); //data computation
9     } else {
10      method12(); //data computation
11    }
12    pause;
13    }
14  }
15  ||
16  {// R2
17   while(true) {
18    method21(); //data computation
19    emit temp;
20    method22(); //data computation
21    pause;
22   }
23  }
24  ||
25  {// R3
26   while(true) {
27    present(temp) {
28      emit OUT1;
29    } else {
30      method31(); //data computation
31    }
32   pause;
33   }
34  }
35 }
```

(a) SystemJ program example (b) Corresponding GRC

Fig. 1. SystemJ program and its GRC graph

instance, method11 is mapped to D11, and so on. Methods in the program just illustrate the use of Java code Java code can be included into SystemJ program almost arbitrarily [1]. The present statement in R1 (line 7) is represented by test node TE1, which branches, depending on condition in test node (presence of signal IN1), to the branch containing either data node D11 or D12. Similarly, the present statement in R3 (line 27) is mapped to test node TE2, branching to the action node A2 or data node D31. In R2, action node A1 denotes the emission of signal temp (line 19), which separates data nodes D21 and D22. All the reactions and conditional branches end with terminate nodes (from T1 to T5), before joining at J1. The execution of control and data nodes on different types of cores leads to largely improved performance and code size reduction as shown in [7] and [8].

Fig. 2 illustrates a mapping and scheduling of the SystemJ program from Fig. 1 to control and data cores, i.e. ReCOPs and JOPs, for two configurations with different numbers of cores. The program can be executed with different schedules of AGRC nodes and produce a correct result. However, the time

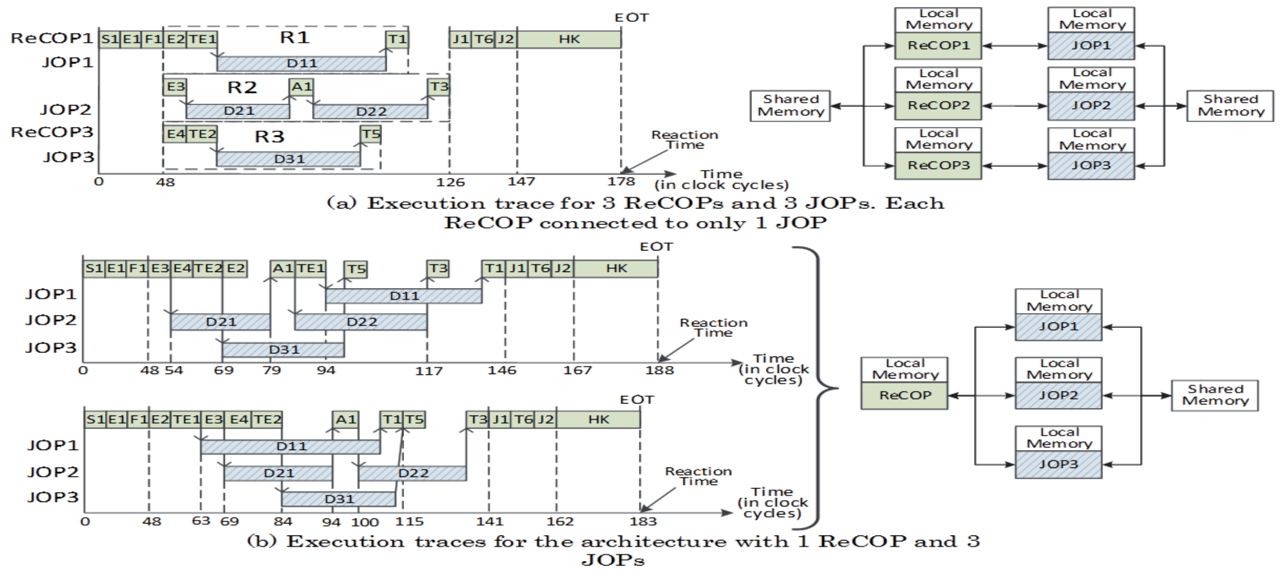


Fig. 2. Execution traces of SystemJ program with different schedules on different architectures

to produce the result in one logical tick of SystemJ program varies in these two cases and reaction (tick) time is different depending on the number of cores used and the type of interconnect mechanism. For some interconnect mechanisms bounds on this time can't be guaranteed. Also, after executing all reactions for one tick, the program has to perform communication with the environment and up-date of statuses of SystemJ signals and channels [1] that happens before starting next logical tick and is labelled as housekeeping (HK) in Fig. 2.

However, communication between control and data cores that enables transfer of control between control and data nodes in AGRC graph in Fig. 2 is assumed to happen instantaneously (see the arrows that represent transfer of control from ReCOP to JOP and vice versa). In terms of physical time, however, a certain number of clock cycles will be needed in order to complete the communication over the interconnection network. This communication cost needs to be included into reaction times when performing timing analysis. To ensure this, a new type of node, called *communication* node, has to be added before and after each data node in AGRC graph (not shown in AGRC graph in Fig. 2) and annotated with its worst case timing characteristics, latency in our case.

IV. TP-HMP

Time predictable execution of GALS programs requires that all components used in program execution, control processors, data processors and interconnection network, are time predictable. A sole measure of the performance for SystemJ hard real time systems is guaranteed worst case reaction time (WCRT), effectively worst case tick time, for each clock domain, and it has to include the worst case communication overhead for each transfer of control from ReCOP to JOP and vice versa.

The new multi-core architecture called time predictable heterogeneous multicore processor, TP-HMP, for execution of SystemJ with the new NoC interconnect is shown in the Fig. 3 and can be made up from an arbitrary combination of ReCOP and JOP cores. A processor with *k* ReCOPs and *l* JOPs, where N=*k*+*l* is the total number of cores, is called TP-HMP[k:l], e. g. TP-HMP[2:6] is a system consisting of 2 ReCOPs and 6

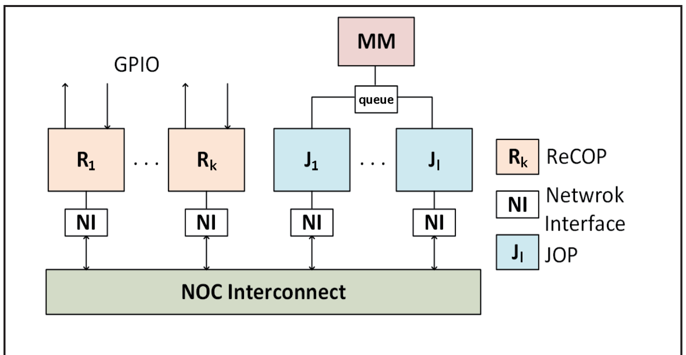


Fig. 3. TP-HMP architecture

JOPs. JOPs are used for executing the Java methods representing data nodes stored in shared main memory MM) in the same fashion as in JOPCMP [10]. The interconnection network is implemented as time predictable TDMA-MIN NoC interconnect and it is described in Section 5. In order to make comparisons with competing interconnects we have also implemented classical TDMA interconnect, mesh NoC and MIN interconnect in FPGA and the results of the comparisons are presented in Section VI.

V. TDMA-MIN NoC Interconnect

Since the performance of TP-HMP system relies on the predictability of the inter-processor communication, the deterministic behavior of the NoC interconnect has to be guaranteed. The conventional packet routing that takes place in direct, i.e. static, networks has an undesirable non-determinism in terms of network latency due to multi-hop routing together with packet prioritization. Addressing the poor network utilization of such networks, multi-stage interconnect networks (MINs) provide an option for bounded latency and high throughput. In order to make MIN time predictable, the use of

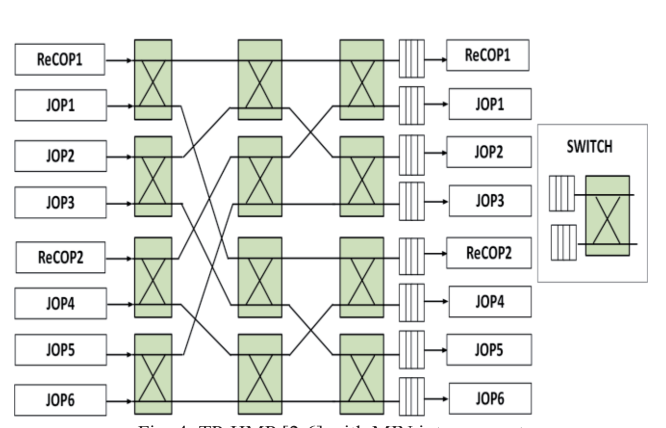


Fig. 4. TP-HMP [2:6] with MIN interconnect

queue at every node is necessary. MIN switches are small, but queue depth grows in size with every new stage. On the other hand, the TDMA network introduces excessive latencies as each processor is allowed to send and receive data only during the assigned slot. Therefore, it has to wait for certain number of slots, depending on the TDMA policy, before it can communicate.

A. MIN – Multistage Interconnect

MIN, provides connections from N input ports to N output ports through a set of permutations in s transmission stages, where $s = log_2$ N, and N has to be equal to a power of 2 or be rounded up to the closest. Each stage features N_s parallel switches, $N_s = N/2$.

Fig. 4 shows an 8-core TP-HMP[2:6] configuration that uses two ReCOPs and six JOPs connected through a buffered MIN. Only FIFOs in the network interface are shown in the figure. FIFOs on switch ports are omitted for clarity. In every stage, the MIN is divided into two symmetric sub-networks. Routing therefore follows the divide and conquer strategy. At each stage, a single bit of the target address is sufficient to control the switch function. Switches can be in one of two modes: straight and cross. Each switch can theoretically forward two packets at the same time, provided they need to be forwarded to different outputs.

However, if two packets, addressed to the same sub-network, arrive at a switch simultaneously, a conflict arises: both packets request the same output link. A toggling priority is implemented on the input ports. Thus, every switch alternates between both inputs in case it gets heavily congested. In order to avoid packet loss, the input ports of each switch needs to be buffered. For this reason, FIFOs are placed at every switch input port. A worst case scenario occurs, when all cores/nodes send a packet to a single receiver at the same time. As the number of parallel links leading to the correct output port decreases by half at every stage, packet queues become longer. For a packet queue of *l* elements in the first stage, the necessary FIFO-depth d(s) in each stage s is therefore equal to:

$$d(s) = ls$$

This increase in FIFO-depth outweighs the advantages of otherwise small routers. Typically, l will be kept small to minimise FIFO expansion.

B. *TDMA-MIN – Multistage Interconnect*

A TDMA-MIN interconnect we propose and implement is inspired by [12] that was used to resolve crosstalk issues in optical networks. Their schedule was used to avoid simultaneous transmission of two packets at one switch with the resulting small network size, high throughput, and flexible options to configure sub-networks [11], Our TDMA-MIN NoC interconnect combines the buffered MIN with TDMA, resulting in simple implementation while providing shorter and predictable transmission times. By adopting TDMA interfaces to control packet insertion, as used in [13], collisions along the transmission paths are avoided. This results in a very small and fast switch in each stage of MIN. Fig. 5 shows the new NoC interconnect architecture where only the input interfaces use FIFO buffers. The fixed schedule and the regular network structure enable switching based on a global counter rather than packet headers. Only a single bit of this target port address is needed to route packets through every stage of the network. Since switching is done based on a global value, all switches react at the same time. Packets can be sent through the whole network in a single clock cycle.

Unlike the mesh network presented in [13], a schedule enabling N-to-N communication can easily be found for the proposed architecture. For a network consisting of eight nodes (3 stages) only eight clock cycles are required to execute the full schedule once. A straightforward implementation of the schedule can be found when all switches of a stage perform parallel or cross switching at the same time. In that case, the state of all switches can be captured with a single 3-bit counter. The same counter is also fed into the network interfaces to determine when packets should be inserted into the network. All switches are used in two different modes only: parallel or cross switching. In parallel mode input-port-A (INA) gets forwarded to output-port-A (OUTA): INA => OUTA and likewise INB => OUTB. Similarly the cross switching mode can be described as INA => OUTB and INB => OUTA. Initially, all switches are in parallel mode.

Connections between input ports and destination ports for each TDMA slot are shown in Table 1. When looking at the connected input and output port numbers in binary format, it becomes clear that the matching output ports are simply mirrored input port numbers in slot 0. It is also clear that no two ports will transmit the packet to same destination port resulting in a collision free transmission without the need of the buffer in the intermediate stages of the network. The global counter bits indicate the switch modes of individual switches: 0→parallel, 1→cross, a function for the global schedule is described as in equation:

$$n_{rx} = \mathrm{Mirror}(n_{tx}) \text{ XOR } T_i,\ i=0,1,\ldots,N-1$$

where T_i (i=0, 1,…, N-1) is the current TDMA slot binary value, n_{rx} is the receiving port, and n_{tx} the port of the transmitting node. Mirror(n_{tx}) denotes n_{tx} mirrored in binary domain, so that the most significant bit becomes the least significant. All packets coming from the transmitting nodes are stored in a FIFO buffer. The packet comprises of two parts: 1) target address and 2) payload. In case of ReCOP-to-JOP communication the target address is the port number of the JOP to which the data call is being made, while payload contains the information about the called data node such as the address of the method to be executed to perform required computations. In case of JOP-to-ReCOP communication, the payload contains result of computation performed by the data nodes method.

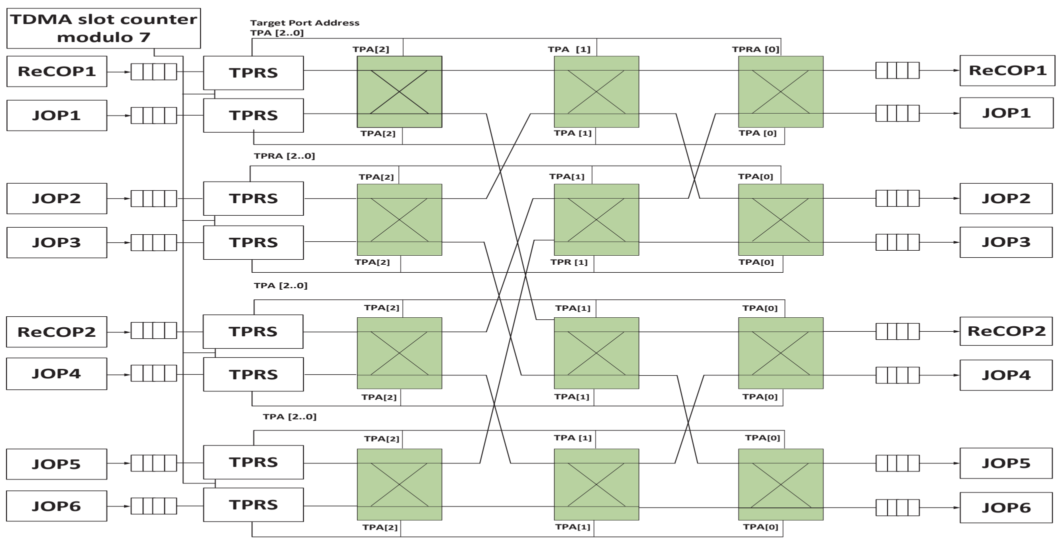


Fig. 5. TP-HMP[2:6] TP-HMP with TDMA-MIN interconnect

Table I Connections in TDMA Slots/states

input port	**mirror**	Destination port in TDMA slots							
		slot 0	**slot 1**	**slot 2**	**slot 3**	**slot 4**	**slot 5**	**slot 6**	**slot 7**
000	000	000	001	010	011	100	101	110	111
001	100	100	101	110	111	000	001	010	011
010	010	010	011	000	001	110	111	100	101
011	110	110	111	100	101	010	011	000	001
100	001	001	000	011	010	101	100	111	110
101	101	101	100	111	100	001	000	011	010
110	011	011	010	001	000	111	110	101	100
111	111	111	110	101	100	011	010	001	000

C. TDMA-MIN Packet Insertion Mechanism

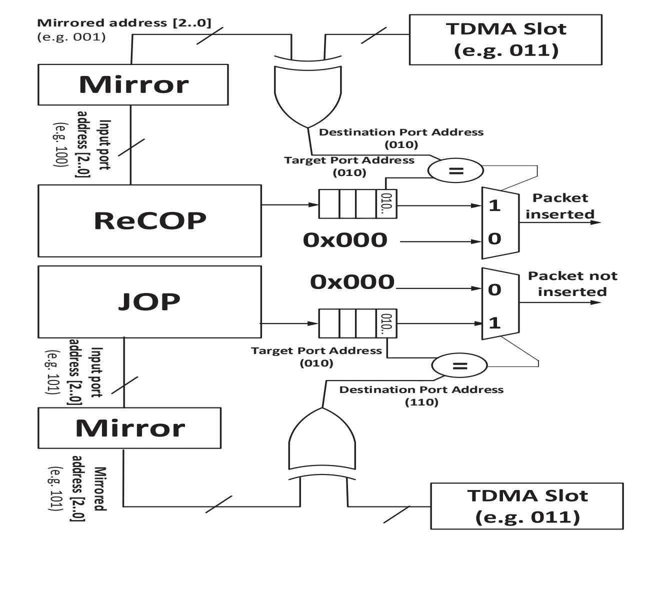


Fig.6. Packet insertion mechanism

The packet insertion circuit is shown in Figure 6. The address of the destination port is calculated during each slot. The port number of the processor making datacall is mirrored and XORed with the counter value (TDMA Slot #) to find the address of the destination port. The datacall (packet) is read from the FIFO and target address provided in the datacall is compared against the calculated destination port address. If the target address matches the destination port address for the current TDMA slot, the packet will be transmitted through the network. Routing through the multiplexer of every stage is based only on the current TDMA slot. Connections that specify the packet route can therefore be made prior to packet transmission. Time required to forward a packet through each stage is reduced to the time required to pass through one 2-input multiplexer. Similar to the buffered MIN architecture a register can be added between selected or even all pipeline stages. Since collisions are avoided through the TDMA schedule, no FIFO-buffers with increasing depth are needed and the design remains scalable.

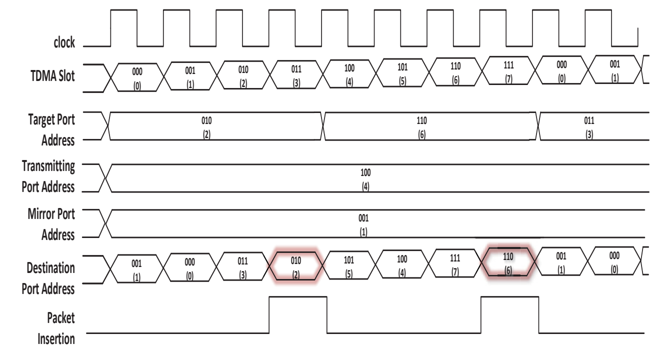


Fig. 7. Timing relationships in the TDMA-MIN interconnect

Figure 7 explains the packet insertion mechanism in more detail. The slot number is provided by modulo 7 counter which increments on every rising edge of the clock. Suppose that the processor with port address number 4 makes three datacalls to three different processors with port addresses 2, 6 and 3, respectively. The incoming datacalls are stored in the FIFO and each datacall (packet) contains the target port address. The address of destination port is calculated in each slot. For example, in slot 0, calculated port address is 1 while target port address is 2. Since a mismatch occurs, no packet is inserted in the network, indicated by a value 0x000. It should be noted that all ports can transmit the packet in each TDMA slot, but not to the same destination port, provided a match occurs, thus providing a very high throughput through the switch.

Writing a packet into the input buffer of the network interface is identical to the process in other NoCs that will be used for comparisons. Equally, buffer is considered to be the final hop through the network. It is accounted for $\tau_{Transmission}$. $\tau_{Interface}$ remains unchanged at 1 clock cycle. Depending on the desired operating frequency, interconnect can be pipelined The transmission time in clock cycles is equal to the number of pipeline stages p in the network:

$$\tau_{Transmission} = p \text{ [clock cycles]}$$

Moreover, TDMA slots are not allocated per input port, as it is the case for the TDMA bus interface. A sequence of matching output ports is cycled through instead. The length of this sequence is determined by the number of used network I/O-ports Np, where Np is determined with the following equation:

$$N_p = 2^{\log_2 N}$$

One of the Np slots in a TDMA round is used to transmit the packet. This results in a maximum delay for τ_{TDMA} clock cycles.

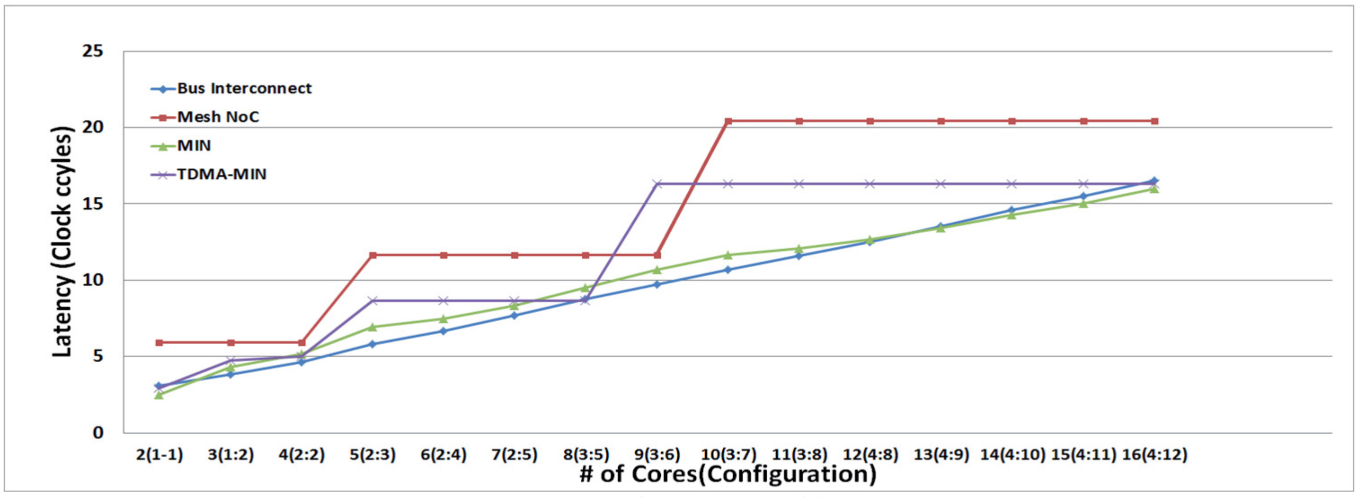


Fig.8 Comparison of worst case network latencies

$$\tau_{TDMA}=[2^{\log_2 N} -1] \text{ [clock cycles]}$$

The upper latency bound in a TDMA-MIN is given by:

$$\tau_{TDMA\text{-}MIN\text{-}UB}=[2^{\log_2 N} + p] \text{ [clock cycles]}$$

VI. Results and Discussion

The proposed TDMA-MIN is implemented on Altera Cyclone III FPGA and is compared against the other approaches (TDMA-based bus, mesh, MIN) in terms of latency, required resources and speed in FPGA implementation. The different configurations of the all compared NoC interconnects and the whole TP-HMP were also prototyped.

A. Latency and throughput bounds

Fig. 8 shows the comparison of worst case network latencies of compared NoC architectures for selected TP-HMP configurations. In the group of TDMA based networks, the bus based interconnect benefits from minimal TDMA cycle lengths. The mesh network and TDMA-MIN suffer from extended TDMA schedules when the number of cores does not match optimal configurations for the network. Deriving worst case latencies on the buffered MIN shows that lower bounds can be guaranteed for large scale networks.

The communication time between ReCOP and JOP, or vice versa, in TP-HMP also includes delays introduced in I/O-ports of the cores themselves is given by the following equation:

$$\tau_{Com.Node} = \tau_{ReCOPport} + \tau_{JOPport} + \tau_{Latency}$$

where $\tau_{ReCOPport}$ and $\tau_{JOPport}$, are the delays for read/write access to the ports by ReCOP and JOP, respectively.

No extra delay is inferred on JOP side. Therefore the overall communication latency is increased by one clock cycle as compared to the NoC latency. The throughput in all compared networks, except TDMA bus network is equal to the number of connected nodes (N). The throughput in the TDMA bus network is only 1 per slot regardless of number of nodes. Thus, the maximum throughput increases linearly with the number of nodes connected to the network for all networks and remains limited to a single packet per clock cycle in TDMA bus network. Also, in TDMA based bus each node can transmit only one packet during the one TDMA slot, while in case of TDMA-MIN each node can transmit during each cycle provided port match occurs for that particular slot.

B. Resource Usage

As TP-HMP is a multiprocessor system based on soft-cores and prototyped on FPGA, the number of required LEs is of great significance when assessing the suitability of different architectures. Fig. 9 shows the resource utilization for different target platforms in terms of the logic elements used when implemented on an Altera Cyclone III FPGA.. For

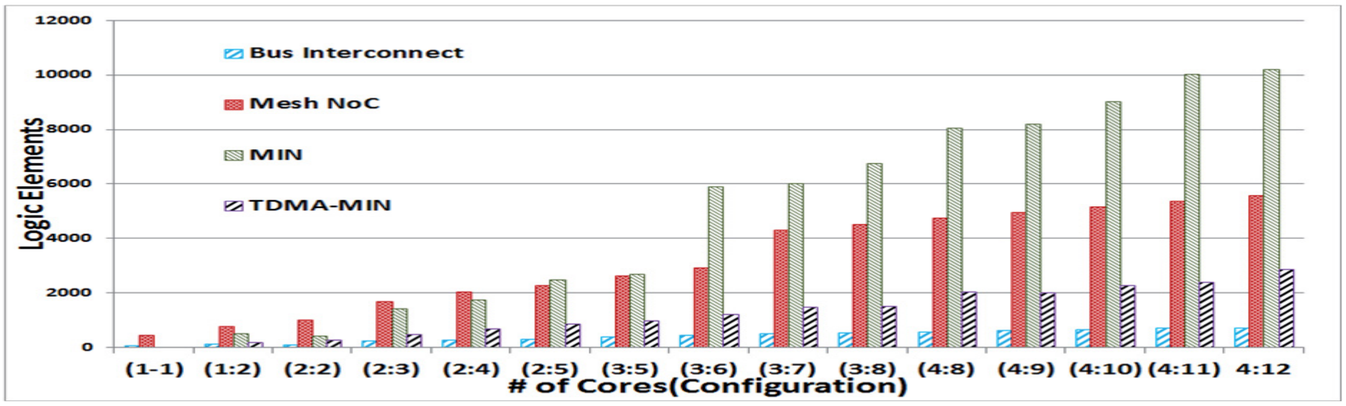


Fig. 9 Comparison of implementation complexity on Altera Cyclone III FPGA

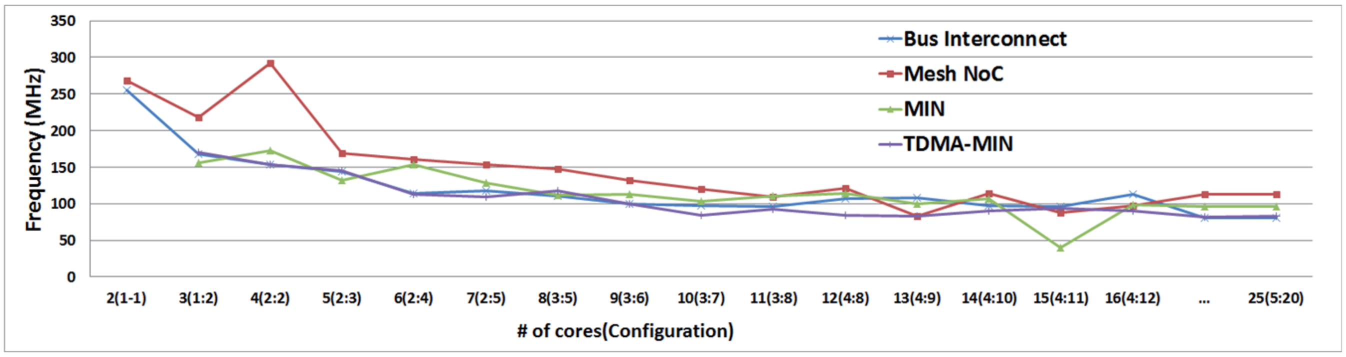


Fig. 10. Comparison of maximum operating frequency on Altera Cyclone III FPGA

networks with more than eight nodes, the required FIFOs in the buffered MIN lead to excessive LE usage, since they are implemented in logic cells for shorter access times. This architecture clearly does not scale to larger networks. Similarly, the mesh network requires a considerable amount of LEs for the five-port routers. A 3 x 3 mesh network requires as many LEs as one JOP core does. The bus and TDMA-MIN architectures therefore have an advantage over MIN and mesh networks.

C. *Operating Frequencies*

The maximum operating frequency f_{max} for different interconnects driven by the same clock are compared in Fig. 10. Network latencies can be shortened when higher clock speeds are used to drive the NoC. However, additional logic is required to synchronise the different physical clocks which introduces additional network delay. Systems which utilise a single clock to drive all components require f_{max} greater than the JOP operating frequency of 60MHz on the Altera Cyclone III FPGA.

VII. Conclusion

This paper introduces a time predictable multicore processor, TP-HMP, capable of executing hard real-time programs developed in SystemJ GALS language. TP-HMP comprises of two types of time predictable cores and TDMA-MIN NoC interconnect that is scalable and provides not only time guaranteed communication between cores but also the throughput of messages exchanged between cores. At the same time TDMA-MIN can be implemented for any number of connected ports with much lower number of logic elements than equivalent mesh network and MIN. TP-HMP, by having all its components time analysis friendly, allows static timing analysis of SystemJ programs and finding tight WCRT. The resulting TP-HMP architecture can be deployed with arbitrary combinations of ReCOPs and JOPs. A configuration, meeting application specific requirements can be found using design space exploration. Our next goal is integration of the TP-HMP into the automated design flow for hard real-time SystemJ programs that will allow the use of design space exploration to determine optimal configuration of TP-HMP for specific application.

References

[1] Malik, A., Salcic, Z., Roop, P.S., Girault, A. SystemJ: A GALS language for system level design. Comput. Lang. Syst. Struct. 36, 317-344 (2010)

[2] Li, Z., Malik, A. AND Salcic, Z.: TACO: A scalable framework for timing analysis and code optimization of synchronous programs. In Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on IEEE, 1-8, (2014)

[3] J. Sparso.: Design of Networks-on-Chip for Real-time Multi-processor Systems on-Chip. In 12th International Conference on Application of Concurrency to System Design, pp. 1–5, IEEE, Hamburg (2012).

[4] Kruskal, C.P. and Snir, M. The performance of multistage interconnection networks for multiprocessors. Computers, IEEE Transactions on 100, 1091-1098 (1983)

[5] Mun, Y., Youn, Y. Performance analysis of finite buffered multistage interconnection networks. Computers, IEEE Transactions on 43, 153-162 (1994)

[6] Schoeberl, M. A Java processor architecture for embedded real-time systems. J. Syst. Archit. 54, 265-286 (2008)

[7] Nadeem, M., Biglari-Abhari, M., Salcic, Z.: JOP-Plus - A Processor for Efficient Execution of Java Programs Extended with GALS Concurrency. In Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific, pp. 17-22. IEEE, 2012.

[8] Salcic, Z., Malik, A. GALS-HMP: A heterogeneous multiprocessor for embedded applications. Transactions on Embedded Computing Systems, 12(SUPPL1). doi:10.1145/2435227.2435254 (2013)

[9] Nadeem, M., Park, H., Li, Z., Biglari-Abhari, M., Salcic, Z.: GALS-CMP: chip-multiprocessor for GALS embedded systems. In Proceedings of the 26th international conference on Architecture of Computing Systems, Prague, Czech Republic 2013 Springer-Verlag, 2450040, 147-158, 2013

[10] Pitter, C., Schoeberl, M.: A real-time Java chip-multiprocessor. ACM Trans. Embed. Comput. Syst. 10, 1-34 (2010)

[11] Qiao, C., Melhem, R. 1994. Reconfiguration with time division multiplexed MIN's for multiprocessor communications. Parallel and Distributed Systems, IEEE Transactions on 5, 337-352 (1994)

[12] Qiao, C., Melhem, R.G., Chiarulli, D.M., Levitan, S.P.: A time domain approach for avoiding crosstalk in optical blocking multistage interconnection networks. Lightwave Technology, Journal of 12, 1854-1862 (1994)

[13] Schoeberl, M., Brandner, F., Sparso, J., Kasapaki, E.: A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on IEEE, 152-160 (2012)

[14] Goossens, K.; Dielissen, J.; Radulescu, A., "AEthereal network on chip: concepts, architectures, and implementations," in Design & Test of Computers, IEEE , vol.22, no.5, pp.414-421, Sept.-Oct. 2005