Journal of Engineering Sciences
Assiut University
Faculty of Engineering
Vol. 42
No. 6
November 2014
PP. 1378 – 1391

DESIGNING AN ARCHITECTURE LEVEL MODEL FOR MULTI-CORE SYSTEMS

Hassan Ali Hassan Ahmed Youness

Staff in Computers and Systems Eng. Depart, Faculty of Engineering, Minia University, Egypt

(Received 13 October 2014; Revised 7 November 2014; Accepted 15 November 2014)

ABSTRACT

The Architecture Level Model (ALM) as a design in space exploration in the early phases of the design process can have a dramatic impact on the area, speed, and power consumption of the resulting systems. A multi-core system is an integrated circuit containing multiple processor cores that implements most of the functionality of a complex electronic system and some other components like FPGA/ASIC on a single chip. In this paper, we present a new approach to synthesize multi-core system architectures from Task Precedence Graphs (TPG) models. The front end engine applies efficient algorithm for scheduling and communication contention resolving to obtain the optimal multi- core system architecture in terms of number of processor cores, number of busses, task-to-processor/channel-to-bus mapping, optimal schedule, and Hardware/Software partition. The back end engine generates a System C simulation model using a well-known commercial tool model generation library to form the architecture level model. The viability and potential of the approach is demonstrated by a case study.

Keywords: ALM, Multi-core, System C, TPG, MPSoC, TLM.

1. Introduction

Multi-core designs represent today the main trend for future architectural designs. It poses number of problems [1], such as extracting parallelism from applications, partitioning application into tasks, allocating and scheduling tasks to multiprocessors and coordinating communication and synchronization between processors. Furthermore the process of automating the generation of realistic multi-core architectural designs from system level specifications remains a challenge for nowadays EDA design tools and the true complexity of the problem is the reason why until to date no tool in this area has become a commercial success.

Several system level modeling and design space exploration methodologies for streaming multiprocessors especially heterogeneous media and signal processing systems have gained wide acceptance in system level community, among those methodologies are SPADE [2], Sesame [3], CASSE [4], and CoWare's approach [5]. These methods share the Y-chart concept that separate between applications and architectures. SPADE and Sesame start with a Kahn Process network (KPN) as design entry and use trace-driven simulation, where execution of architecture model is driven by traces from execution of application model.

CASSE follows a programming model based on Task Transaction Level Interface (TTL) [6] and uses System C based simulation. On the other hand, CoWare uses a set of un-timed reactive System C tasks communicating through a Transaction Level Model interface. In work of [7] they used an extended Task Graph (eTG) as design entry, and generated a scheduled System C performance model that is further synthesized into interacting H/W modules with control unit implemented as FSMs [8, 9, 10].

Researchers have also proposed system level modeling methodologies for integrated HW/SW embedded systems, but aiming at automated HW/SW partitioning, co-design, code generation from formal specification languages, and refinement from abstract system level models to more realistic implementations including communication refinement. The Ptolemy [11], Metropolis [12], and COSY [13] environments are pioneering work in that direction.

In the COSY model higher abstraction level interfaces are proposed, where application level transactions are used for programming a network of functions that specifies what the system is supposed to do. They are refined into system transactions when choosing implementation of functions to software and hardware components.

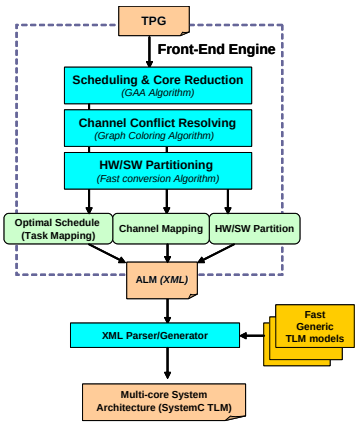


Fig. 1. Proposed System Level Synthesis to ALM Approach.

In this paper, we present a system level synthesis approach to automate the process of architecture exploration and generation of specialized multi-core system architectures from Task Precedence Graphs (TPG). Figure 1 shows the outline of the proposed synthesis to ALM approach.

The design entry is a Task Precedence Graph with communication channels (TPG). The output is an Architecture Level Model (ALM) of a specialized multi-core system architecture described in extended Markup Language (XML) format. The ALM description is then processed by a utility we developed (XML Parser/Generator) that will interpret the structure and operation of an optimal (or semi-optimal) configuration of our specialized multi-core system architecture, instantiate the appropriate System C TLM 2.0 (Transaction Level Model) compliant models from a library of fast generic models and connect the target multi-core system architecture.

The front-end engine of our synthesis approach is an efficient algorithm to solve the task allocation/scheduling problem to obtain the optimal schedule on a multiprocessor system and reduce the number of processors in the target system. The algorithm also resolves communication channels conflicts and reduces the overall execution time of the application task

graph by using HW/SW partitioning that depends on fast conversion from SW to HW tasks.

The rest of this paper is organized as follows. Section 2 describes the output & input of our proposed synthesis approach and the algorithm of the front-end engine. Section 3 shows an illustrative example "case-study" to clarify the algorithm operation and back-end generation. Section 4 draws some conclusions.

2. Target system and graph model

2.1. Architecture level model (ALM)

To narrow down the complexity of MPSoCs, we used a specialized multi-core system architecture template that consists of multiple Processor cores for SW (i.e CPU core) and HW tasks (i.e. FPGA).

Each Processor core has its Local Memory (Cache) that is used for communication among the tasks on the same core. The cores communicate with shared memory via multiple buses and a global Real-Time Resource Scheduler (RTRS) module that schedules/controls the tasks operation on these cores as shown in Fig. 2.

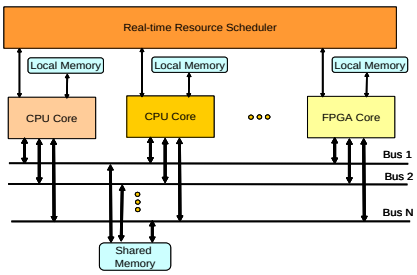
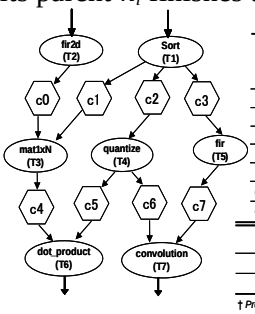


Fig. 2. Specialized Multi-core system architecture.

2.2. Task precedence graph

The design entry is given as a Task Precedence Graph with communication (TPG) as shown in Fig 3. In general, a TPG *G(V,E,W,C)*, is a Directed Acyclic Graph (DAG), such that *V* is the set of nodes, *E* is the set of edges, *W* is the computation costs of the nodes and *C* is the communication costs of the directed edges. A node in the DAG represents a task which in turn is a set of instructions that must be executed sequentially without preemption in the same processor. Edges in the DAG are directed and thus capture the precedence constraints among the tasks. The communication link between tasks denoted by (n_i,n_j) and implies that nj is a child which cannot start until its parent n_i finishes and sends its data to n_j.



TPG Profile Table

Task	Computation Cost Time: (cycles) [Energy: (nJ)]
sort	6762 [571.74]
fir2d	108821 [8455.4]
mat1xN	9715 [719.94]
quantize	925 [86.82]
fir	2086 [163.47]
dot_product	906 [72.25]
convolution	871 [65.84]

Channel	Communication Cost
c0	2000 [38.43]
c1, ..., c7	200 [3.84]

† *Profiled on: M32R Core@50MHz*

Fig. 3. The TPG practical example of JPG picture.

The geometry of the task graph divides it into paths and levels; the path is the way from the start node to the exit node considering all the nodes and the edges in this way. The levels are depths of nodes. In scheduling a task graph *G* onto a target system with a set of processors *P*, each node must be assigned to one processor. Classical assumptions [14] that are made about the target system are:

i) Tasks are non-preemptive.
ii) Local communications are cost free.

Let $t_S(n_i)$ be the execution start time of node n_i, and $w(n_i)$ be the weight of node n_i. The execution finish time of node n_i, $t_f(n_i)$ can be denoted [15] as:

$$t_f(n_i)=t_s(n_i)+w(n_i) \tag{1}$$

where a processor can execute only one task at the same time, two nodes n_i and n_j assigned to the same processor should satisfy the following equation:

$$t_f(n_i)< t_s(n_j) \text{ or } t_f(n_j)< t_s(n_i) \tag{2}$$

For an edge (n_i,n_j), if n_i and n_j are assigned to the same processor, communication cost becomes zero since there is no data transfer between processors. Hence, we define edge finish time $t_f((n_i,n_j))$ as the following equation [16] where $c(n_i,n_j)$ denotes communication cost of edge (n_i,n_j)

$$t_f((n_i,n_j))=\begin{cases} t_f(n_i) & \text{if } n_i, n_j \text{ on same processor} \\ t_f(n_i)+c(n_i,n_j) & \text{if } n_i, n_j \text{ on different processors} \end{cases} \tag{3}$$

Node n_j can start to execute after all data are ready, i.e. all edges connected to node n_j are finished. Hence data ready time of node n_j, $t_{dr}(n_j)$, is defined as

$$t_{dr}(n_j)=\max_{i,(ni,nj)\in E}\{t_f(n_i,n_j)\} \tag{4}$$

Scheduling length of scheduling *S*, *SL(S),* is the time when all nodes are finished. Hence *SL(S)* is defined as follows:

$$SL(S)=\max_{n\in V}\{t_f(n)\} \tag{5}$$

Two processors p_i and p_j are isomorphic if the ready times of them are equal and the tasks n_i and n_j are equivalent when the following conditions are satisfied [14].

$$\begin{array}{ll} i) & pred(n_i) = pred(n_j), \\ ii) & w(n_i)=w(n_j), \text{and} \\ iii) & succ(n_i) = succ(n_j) \end{array} \tag{6}$$

According to the above assumptions, we can derive the following fact. Scheduling length of optimal task assignment to *P+1* processors is always less than or equal to the one to *P* processors [15],

$$SL(S_{opt}(P+1)) \le SL(S_{opt}(P)), \tag{7}$$

where *SL(S)* represents scheduling length of scheduling *S* and $S_{opt}(P)$ represents optimal scheduling to *P* processors.

2.3. Front-End Engine Algorithm

The main role of the front-end engine algorithm is to obtain optimal design parameters for the multi-core system architecture given a Task Precedence Graph (TPG) model. The objective is to maximize performance in terms of execution speed under some constraints as precedence, area, and communication conflicts. In order to achieve this we divide the front-end engine algorithm into two processes.

2.3.1. Scheduling, cores reduction, and channel conflicts resolving process

We use The *A-star* as a best-first state space search algorithm that maintains two lists *OPEN* and *CLOSED*. It starts by putting the initial state into the *OPEN* list. The states in the list are sorted according to the cost function *f(s).* The cost function *f(s)* is the addition of *g(s)* and *h(s), g(s)* represents the cost from the initial state to the state *s* and *h(s)* is the estimated cost from the state *s* to the goal state. In each step, the state *s* with the lowest *f(s)* value is taken from the list and put onto *CLOSED* list. The algorithm terminates when the state *s* is the goal state.

The task graph is partitioned according to the geometrical shape of paths and levels. The path length (sum of weights of nodes and edges) are calculated and sorted in descending order. From paths and levels we build a matrix called the geometrical matrix (*G_matrix*). Then, we use pruning techniques to reduce the number of search state space, for example, the property of isomorphic processor and node equivalence. Lastly, the algorithm reduces the number of processors to be used for scheduling in case of idle time.

The Geometric A* Algorithm (GAA) [17] is outlined below where *Q* represents the set of used processors (Note. at least one task is assigned to the processor), as shown in Fig. 4. The algorithm schedules the node to only one processor in case of there are many isomorphic processors and at the same time it searches for the used processors that has a ready time less or equal to the ready time of the isomorphic one to reduce the number of processors that can be used in the scheduling process. The algorithm is explained in section 3 as illustrative example.

To derive the time complexity, suppose that the application has *n* nodes with *l* levels, *h* paths and *r* repeated node and this application should be scheduled on *p* processors. The flowchart shows that there is *l* loop of repeated nodes (*r*) and non-repeated nodes (*n-r*) on *p* cores. Then $l[(rp)+((n-r)p)]=lpn$ so, in the worst case *l=E* (i.e. $O(Epn)$) and if all the nodes are fully connected then $E=(n^2-n)/2$, so the time complexity is $O(\frac{p}{2}(n^3-n^2))$.

Channel conflicts are resolved by mapping conflicting channels to different buses. This scheme sacrifices area (bus interconnect) to preserve the optimal schedule length. This can be formulated as follows; Given a Channel Conflict Graph, defined as undirected graph $G_{Conflict}=(V,E)$, where V denotes communication channel and E is conflict relation on V. The algorithm maps $c:V\rightarrow B$ where B is the set of buses such that, $c(u)\neq c(v)\,|\,\forall(u,v)\in G$.

2.3.2. Partitioning process

The partitioning starts after scheduling by searching for the critical path (i.e. the Processor core that has the longest schedule length) and converts SW tasks running on that core to HW tasks provided that SW tasks attributes indicate feasibility to convert to HW. This conversion has an impact on the execution time of each task by a reduction of (1/3 ~1/5) of its original computation time value.

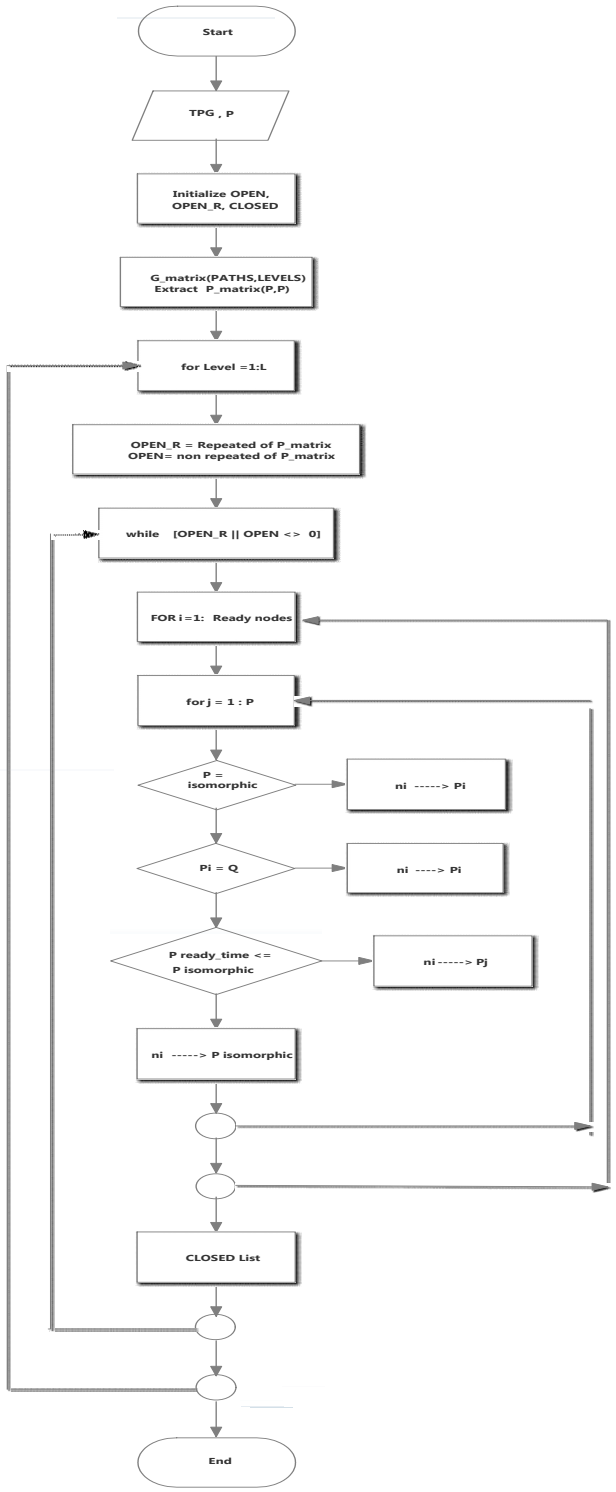


Fig. 4. The Scheduling Algorithm (GAA).

3. Case study and experimental results

We present a case study to illustrate the operation of the front-end engine algorithm using the task graph shown in Fig. 3; initially we assumed the target system contains three processor cores on a single bus*:*

Step (1): The task graph is partitioned into paths and levels, these paths are sorted in descending order by their values. There are 5 paths and 3 levels from start node to the goal node.

Path1 is T2, T3, T6 value (path1) = 121642 cycles
Path2 is T1, T3, T6 value (path2) = 17783 cycles
Path3 is T1, T5, T7 value (path3) = 10119 cycles
Path4 is T1, T4, T6 value (path4) = 8993 cycles
Path5 is T1, T4, T7 value (path1) = 8958 cycles

Step (2): Levels

Level 1 is T1, T2
Level 2 is T3, T4, T5
Level 3 is T6, T7

Step (3): The *G_matrix* is built from paths and levels; it gives a matrix with 5 rows and 3 columns

$$G\_matrix = \begin{bmatrix} T2 & T3 & T6 \\ T1 & T3 & T6 \\ T1 & T5 & T7 \\ T1 & T4 & T6 \\ T1 & T4 & T7 \end{bmatrix}$$

Step (4): According to the target system (*P=3*), a priority matrix called *P_matrix*, has rows and columns equal to the number of processors in the system and the *P_matrix* is extracted from *G_matrix* by taking the first three rows and three columns.

$$P\_matrix = \begin{bmatrix} T2 & T3 & T6 \\ T1 & T3 & T6 \\ T1 & T5 & T7 \end{bmatrix}$$

Step (5): As noticed from the *P_matrix*, there are some nodes that are repeated in the same level according to their relation to the child nodes as T1 here has 3 children. So, these nodes are called the repeated nodes and should be put in the *OPEN_R* list and the other nodes are called non-repeated nodes and should be put in the *OPEN* list

OPEN_R = T1, T3, T6

OPEN = T2, T4, T5, T7

Step (6): The scheduling process will begin as in the algorithm using the *P_matrix* and the *A-star* fitness function *f(s),* Table (1) shows the function values of scheduling for tasks using 3 and 2 cores, respectively.

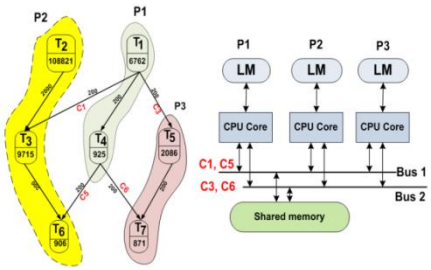
Figure 5 shows the Gantt chart *(schedule)* for each case. By observation, it turned out that scheduling on two cores has the same schedule length as three (i.e. cores are reduced). We need two buses when three cores are used (c1 conflicts with c3 and c5 conflicts with c6

"when drawing the GANTT chart to scale"). Using graph coloring (c1 and c5) are mapped to bus 1 and (c3 and c6) to bus 2.

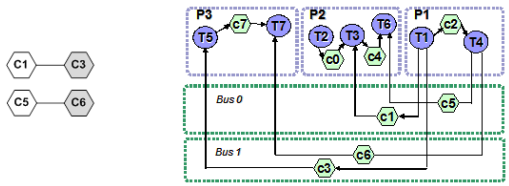
Table 1.
Task Scheduling process on 3 cores.

Level	Task	g(s)+h(s)		
		Core 1	**Core 2**	**Core 3**
1	T1	6762+11021*		
	T2		108821+12821*	
2	T3	120536+1106	118536+1106*	120536+1106
	T4	7687+1106*		7887+1106
	T5	8848+1071		9048+1071*
3	T6	119642+0	119442+0*	119642+0
	T7	11190+0		10990+0*

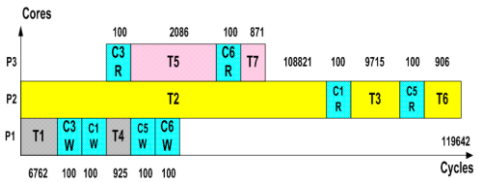
Level	Task	g(s)+h(s)	
		Core 1	**Core 2**
1	T1	6762+11021*	
	T2		108821+12821*
2	T3	120536+1106	118536+1106*
	T4	7687+1106*	
	T5	9773+1071*	
3	T6	119642+0	119442+0*
	T7	10644+0*	119507+0



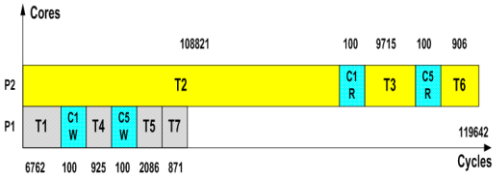
(a)



(b)



(c)



(d)

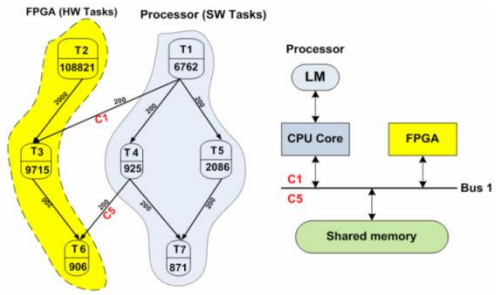
Fig. 5. Scheduling and target system. (a) Task mapping (b) Channel conflict graph and number of buses (c) Gantt chart on 3 cores (d) Gantt chart on 2 cores.

The partitioning process is done after scheduling/ processor cores reduction, and channel conflicts resolving stage as a final step to compact the SL along its critical path. The second choice of two cores on one bus is selected for the partitioning process as outlined next.

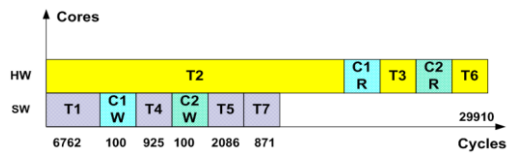
Step 1: Find the longest schedule length on cores, processor P2 has SL = 119642 cycles, including the time for data write (Channel-W) and read time (Channel-R) from and to tasks that are implemented on other core respectively.

Step 2: The SW tasks on the longest SL will be converted to HW tasks to reduce the overall execution time. We assume the execution time of a HW task on FPGA equals one third to one fifth of the execution time of a SW task on CPU core. In this case study, all tasks have feasible attribute for conversion to H/W tasks.

Step 3: The target architecture is reduced to one bus, one processor core with HW tasks (i.e. FPGA) and one processor core with SW tasks. Task T1 send data to task T3 via channel (C1) and task T4 send data to T6 via channel (C5), the channels are mapped to one bus (no conflicts) and the SL is reduced to 29910 cycles, see Fig. 6.



(a)



(b)

Fig. 6. Scheduling and target system for 2 cores. (a) Task mapping

(b) Gantt chart after HW/SW partitioning

As for backend generation, the optimal *(or any semi-optimal configuration)* can be

selected for SystemC generation; both the structural and scheduling info is written to an XML file. The information is parsed and processed by a utility we developed to integrate the front-end engine to Mentor Graphics® Vista Architect™ Model Builder tool [18, 19]. The utility would instantiate the appropriate System C TLM 2.0 compliant models from a library of fast generic models (e.g. Processor core with SW/HW Thread, Bus "AHB/AXI", MEM/Cache, etc.) and connect the target multi-core system architecture based on the design parameters and configuration in the XML file. As for the interconnect (See Fig. 7) and dynamics (see Fig. 8) of the final generated multi-core system, we use the built-in generic memory model for modeling the shared and the local memories, the bus model which is currently an AHB bus for modeling the system interconnect, and the CPU model for modeling the Real-Time Resource Scheduler (RTRS) [20]. We build a number of processor core models for the main processing element with a SW or HW Task thread. For the interface protocols, we used an AHB master/slave TLM protocol for the bus interface between RTRS, Processor cores, as well as shared memory and the system AHB bus. We also used a Tightly Coupled Memory (TCM) protocol for the interface between the processor cores and their associated local memories. The tasks running on each core is statically bounded according to the scheduling algorithm results. Each task is modeled as an SC_THREAD that is suspended for the specified execution time based on the computation cost of the task (SW or HW). Access to channels is modeled as bus read/write transactions from specific memory addresses.

The memory address space that can be accessed through each bus is specified in a parameter file to correctly map the channels to their dedicated buses. The time spent in a channel read/write transaction (i.e. communication cost) is specified using the parameterized memory read/write associated delay timing policy. The SW thread associated with the RTRS is modified to initiate tasks at different cores to run according to the scheduling algorithm results. An SC_METHOD "control callback" in each processor core is sensitive to the control interfacing port connected to the RTRS. This method notifies the "Go" event associated with each of the statically allocated tasks which are implemented as SC_THREADS. The Scheduler thread in the RTRS is responsible for generating correct control information at appropriate time to realize the pre-determined scheduling.

Upon simulating the generated System C code equivalent to the three processor configuration in Fig. 9, the following statistical charts are directly obtained using the Vista Architect™ analyzing utility. Fig 8. shows three cores (red = P1, green = P2, and blue = P3) and tasks id's on Y-axis versus time in cycles on X-axis which is equivalent the Gantt chart of Fig. 5. (c) when drawn to scale. Other analysis widgets are also available for calculating throughput and latency on busses for example Fig. 10 shows the read/write transactions on buses 1 and 2 for the given configuration.

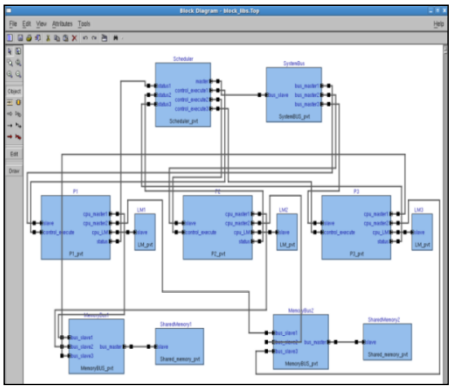


Fig. 7. Interconnect of the Generated multi-core system (3 cores).

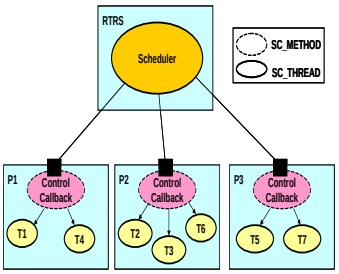


Fig. 8. Dynamics of the Generated multi-core system.

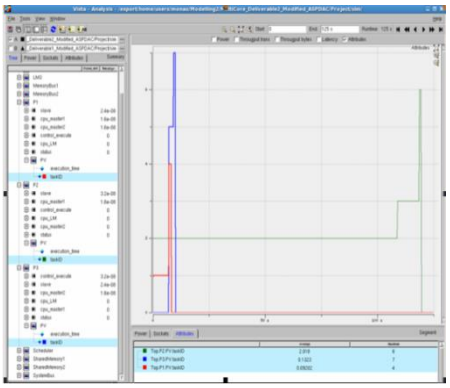


Fig. 9. Processors Tasks GANTT chart (3 cores).

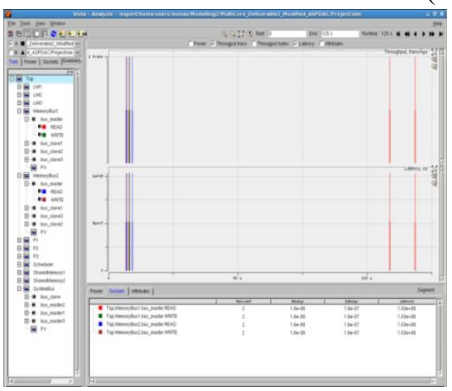


Fig. 10. Read/Write transactions on Bus 1 and Bus 2

4. Conclusion

In this paper we presented a system level synthesis approach for multi-core system architectures. The front-end uses an efficient algorithm for optimal architecture selection. The algorithm solves task scheduling problem and obtain the optimal schedule on a specialized multi-core system architecture with a capability of (i) reducing the number of cores in the target system, (ii) resolving communication contention over buses and (iii) minimizing execution time by searching for the cores that have the longest schedule length and converts their tasks to be executed in hardware instead of software if feasible. The output of the algorithm which is typically an optimal ALM structure/schedule is then fed to a utility (back-end generator) that uses Vista Architect Model builder™ to architecturally build the system with appropriate TLM components and dynamics for System C simulation.

5. Acknowledgment

I would like to express my greatest gratitude to our friends who have helped & supported me throughout my project in Mentor Graphics Egypt and my greatest thanks to Mentor Graphics Company that helped me to use the tools over than one year throughout this project.

REFERENCES

[1] G. Martin. “Overview of the MPSoC Design Challenge,” in Proc. Of DAC’06, pp. 274-279, CA, U.S.A., July 2006.

[2] V. Zivkovic, E. Deprettere, P. Van der Wolf and E. De Kock. “Design space exploration of streaming multiprocessor architectures,” in Proc. of SIPS’02, pp. 228-234, USA, 2002.

[3] A. Pimentel, L. Hertzberger, P. Lieverse, P. Van der Wolf and E. Deprettere. “Exploring Embedded-Systems Architectures with Artemis,” in IEEE Computer, pp. 57-63, Vol. 34, No. 11, Nov. 2001.

[4] V. Reyes, T. Bautista, G. Marrero, P. Carballo, W. Kruijtzer. “CASSE: A System-Level Modeling and Design-Space Exploration Tool for Multiprocessor Systems-on-Chip,” in Proc. of DSD'04, pp. 476-483, France, Aug. 2004.

[5] T. Kempf, M. Doerper, R. Leupers, G. Ascheid, H. Meyr, T. Kogel and B. Vanthournout. “A Modular Simulation Framework for Spatial and Temporal Task Mapping onto Multi-Processor SoC Platforms,” in Proc. of DATE’05, pp. 876-881, Germany, Mar. 2005.

[6] P. Van der Wolf, E. De Kock, T. Henriksson, W. Kruijtzer and G. Essink. “Design and Programming of Embedded Multiprocessors: An Interface Centric Approach,” in Proc. of CODES+ISSS’04, pp. 206-217, Sweden, Sep. 2004.

[7] S. Klaus, S. Huss and T. Trautmann. “Automatic Generation of Scheduled SystemC Models of Embedded Systems From Extended Task Graphs,” in System Specification & Design Languages - Best of FDL'02, pp. 207-217, Kluwer, 2003.

[8] Digalwar, M. ; Gahukar, P. ; Mohan, S., “Design and development of a real time scheduling algorithm for mixed task set on multi-core processors”, 7th International Conference on Contemporary Computing (IC3), DOI: 10.1109/IC3.2014.6897184, pp. 265 – 269, 2014.

[9] Brinkschulte, U., “Introducing Virtual Accelerators to Decrease the Communication Overhead of an Artificial Hormone System for Task Allocation”, IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), DOI: 10.1109/ISORC.2014.23, pp. 117-124, 2014.

[10] Yi Wang ; Zili Shao ; Chan, H.C.B. ; Duo Liu ; Yong Guan, “Memory-Aware Task Scheduling with Communication Overhead Minimization for Streaming Applications on Bus-Based Multiprocessor System-on-Chips”, IEEE Transactions on Parallel and Distributed Systems, Volume: 25 , Issue: 7 DOI: 10.1109/TPDS.2013.172, pp: 1797 – 1807, 2014.

[11] J. Buck, S. Ha, E. Lee and D. Messerschmitt. "Ptolemy: a framework for simulating and prototyping heterogeneous systems," International Journal of Computer Simulation, on Simulation Software Development, Jan. 1990.

[12] F. Balarin, H. Hseih, L. Passerone and A. Sangiovanni-Vincentelli. "Metropolis: An integrated electronic system design environment," in IEEE Computer, pp. 45-52, Vol. 36, No. 4, April 2003.

[13] J. Brunel, W. Kruijtzer, H. Kenter, F. Pétrot, L. Pasquier, E. De Kock and W. Smits. "COSY Communication IP's," in Proc. of DAC'00, pp. 406-409, California, U.S.A., June 2000.

[14] Y. K. Kwok and I. Ahmad. "Dynamic critical path scheduling: An elective technique for allocating task graphs to multiprocessors", IEEE Transactions on Parallel and Distributed Systems 7(5):506–521, 1996.

[15] O. Sinnen, "Task Scheduling for Parallel Systems", Wiley Publisher, Hoboken, New Jersey, USA, 2007.

[16] S. Banerjee and N. Dutt, "Efficient Search Exploration for HW-SW Partitioning", Proc. of Int'l Conf., CODES+ISSS'04, pp. 122-127, Sept., 2004.

[17] H. Youness, K. Sakanushi, Y. Takeuchi, A. Salem, A. Wahdan and M. Imai, "Optimal Scheme for Search State Space and Scheduling on Multiprocessor Systems", IEICE Transaction On Fundamentals of Electronics, Communications and Computer Sciences, Vol. E92-A, No.4, pp.1088-1015, Apr. 2009.

[18] Mentor Graphics® Vista Architect™ Tool [Online 2014]: Available: http://www.mentor.com/products/esl/design_verification/vista_architect.

[19] Mentor Graphics® A Complete TLM 2.0-based Solution [Online 2014]: Available: http://www.mentor.com/esl.

[20] Rudat, A. ; Battat, J. ; Cameron, B., "The modeling and evaluation of interplanetary manned missions using system architecting techniques" IEEE on Aerospace Conference, DOI: 10.1109/AERO.2013.6496944, pp. 1-17, 2013.

تصميم نموذج لعمارة الحاسب لانظمة الحاسبات متعددة المعالجات

الملخص العربى:

يعتبر النموذج المعمارى للحاسب (**ALM**) هو التصميم الامثل فى استكشاف الحلول في مراحل مبكرة من عملية التصميم، حيث يكون له تأثير كبير على استهلاك مساحة صغيرة من الشريحة الالكترونية وزيادة سرعة الحاسب وكذلك الحد من استهلاك الطاقة. نظام متعددة النواة هو دائرة متكاملة تحتوي على النوى المتعددة في المعالجات التي تطبق معظم وظائف النظام إلكتروني المعقد وبعض المكونات الأخرى مثل **ASIC/ FPGA**على شريحة واحدة. في هذه الورقة، نقدم نهجا جديدا لتجميع أبنية نظام تعدد المعالجات من نماذج المهام ذات الأسبقية فى الرسوم البيانية **(TPG)**. المحرك الامامى يقوم بتطبيق خوارزم فعال لجدولة المهام على المعالجات وحل مشكلة الاتصالات بينها وكذلك تقليل عدد المسارات وعدد المعالجات على الشريحة وتقسيم النظام بين التصميم البرمجى والمادى بشكل امثل. اما المحرك الخلفى فيستخدم لتوليد نموذج محاكاة **System C** باستخدام اداة مكتبية لتكوين النموذج المعمارى للحاسب والحصول على افضل التصميمات على الشريحة الكترونية وهو **ALM** . وجدوى وامكانيات هذة الطريقة تتجلى فى نتائج دراسة الحالة المستخدمة.