Microprocessors and Microsystems 44 (2016) 17–21

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Assignment of unexpected tasks in embedded system design process

Adam Górski*, Maciej J. Ogorzałek

Department of Information Technologies, Jagiellonian University, Poland

ARTICLE INFO

Article history:
Received 26 September 2015
Revised 2 December 2015
Accepted 3 January 2016
Available online 8 January 2016

Keywords:
Embedded systems
Unexpected tasks
Co-design
3D integrated circuits
Co-synthesis
Microprocessors and microcontrollers

ABSTRACT

Embedded systems design process focuses on three areas: modeling, validation and implementation. Typically such procedure assumes constant number of tasks in every instance of designing procedure. Thus the designer must predict all possible tasks executed by the system. Serious problems appear when the system has to execute unexpected tasks. In this case the design process must be repeated. We propose a new approach in embedded system design process which covers such situation. In the approach unexpected tasks are assigned to previously allocated resources. Therefore the system can execute more tasks that were predicted by the designer.

1. Introduction

Nowadays embedded systems and software can be found everywhere. For example: modern cars (see. Srovnal et al. [7], Chandra et al. [1]), mobile phones (see Sun et al. [9]), large group of medicine solutions (e.g. Masihpour et al. [6], Shoeb et al. [10], Ölveczky [11]), face recognition (e.g. Acasandrei et al. [2]), coordination of different teams in crisis management (e.g. Mahdjoub et al. [3]), etc.

Most of the papers assume distributed target architecture of embedded systems. In such a representation tasks are executed by Processing Elements (PEs). PEs can be divided on two groups: Programmable Processors (PPs) and Hardware Cores (HCs), which are connected in target architecture using Communication Links (CLs).

Embedded system design process according to De Micheli et al. [23] consists of: modeling, validation and implementation phases.

Modeling is the process which provides a hardware/software model of the system. A part of modeling process is cosynthesis (e.g. Densmore et al. [15], Jozwiak et al. [8]). Cosynthesis automatically generates the architecture of Embedded system using specification given as a list of parallel processes. The process includes allocation of processors, task assignment and task scheduling. Cosynthesis methodologies can be divided on two basic groups: constructive (e.g. Deniziak et al. [12]) and iterative improvement (e.g. Oh et al. [17]). Constructive algorithms build system step by step choosing processing elements for each task separately. Iterative improvements methodologies start from suboptimal solution (usually the fastest) and try to improve system quality by making local changes. Large group of solutions are evolution algorithms like: simulated annealing (e.g. Eles et al. [22]), genetic algorithms (e.g. Guo at al. [14], Dick et al. [21]) and adaptive methodologies (e.g. Górski et al. [4]) which are able to adapt to the environment. Especially good results were obtained using genetic programming (Górski et al. [5]).

Validation (e.g. Ko et al. [13]) verifies if system works properly and that all the requirement were satisfied.

A really serious problem with system design process appears when unexpected tasks must be executed. Then in every existing methodology all design process must be repeated. It is needed because every design methodology assumes constant number of tasks that are executed by the embedded system. Even papers that assume unexpected scenarios of embedded systems (e.g. Mise et al. [24]) concentrate on security and faults detection (e.g. Kovalev et al. [25]). In this work we propose a new approach to system design process. In our approach the designer can assign unexpected task to existing processing elements without repeating the design process in the case where adding a new PE is not possible. That implies a possibility to react in situations when redesigning is impossible or only part of a design process can be repeated. The approach is also suitable when the system is design re-use. We can also decrease designing costs and cost of the system.

The paper is organized as follows: Section 2 describes a representation of embedded system, Section 3 includes problem

* Corresponding author. Tel.: +48508061976.
E-mail addresses: a.gorski@uj.edu.pl (A. Górski), maciej.ogorzalek@uj.edu.pl (M.J. Ogorzałek).

http://dx.doi.org/10.1016/j.micpro.2016.01.001
0141-9331/© 2016 Elsevier B.V. All rights reserved.

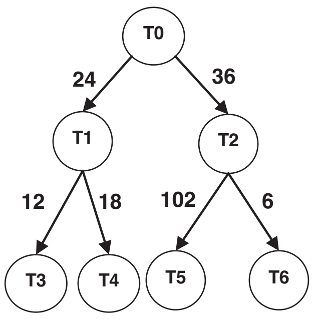

Fig. 1. Example of a task graph for a robot working on Mars. It consists of 7 nodes corresponding to 7 tasks T0-T6. The numbers at the edges correspond to amount of data that has to be sent between the tasks.

Table 1
Example database.

Task	PP1 C=200		PP2 C=250		HC1	
	t	c	t	c	t	c
T0	100	12	103	10	30	100
T1	87	23	60	34	5	93
T2	45	18	50	20	3	60
T3	90	9	86	8	10	95
T4	35	5	40	8	2	40
T5	22	1	30	3	2	55
T6	190	25	150	30	25	150
CL1, b=6	c=3		c=2		c=10	

statement, Section 4 presents a methodology, Section 5 the example and in chapter 6 conclusions and future work are given.

2. Representation of embedded system

Some of the most popular representations of behavior of an embedded system are: task graph representation (this representation was used in this paper) and conditional task graph (e.g. Xie et al. [26]). Another important types of representation, amount others, are: c-code (e.g. Venkataramani et al. [16]) and MATLAB representation (e.g. Banerjee et al. [19]).

Task graph G = {V, E} is consisted of nodes (V) and edges (E). In the graph each node v_i Є V represents a task. Each edge e_{ij} Є E represents amount of data that has to be transferred between two connected tasks v_i and v_j. Eles et al. [20] proposed the concept of an extended task graph. Klaus et al. [18] applied this concept for analysis of an autonomous robot. We decided to adapt part of this example to describe a robot working on the planet Mars. Fig. 1 shows an example of task graph for that robot.

The system described by above graph executes 7 tasks: T0, T1, T2, T3, T4, T5 and T6. Tasks T1, T2 and T3,T4, T5, T6 are parallel. An example database for the system is presented in Table 1.

The values in the table were generated randomly. t is the time of the execution of a task, and c is the cost of execution the task or cost of connecting task to CL. C is a cost of a PP. The cost of HC is included in the cost of execution tasks. In the example there are three kinds of PE – two programmable processors and one hardware core. All PEs can be connected using one CL (CL1). All tasks can be executed by every resource, and CL1 can be connected to every PE.

t_{ij} is the transmission time between tasks v_i and v_j.

$$t_{ij} = \frac{e_{ij}}{b_{CL_{ij}}} \tag{1}$$

where b_{CLij} is bandwidth between the tasks. If tasks v_i and v_j are implemented on the same PE, the transmission time will be zero.

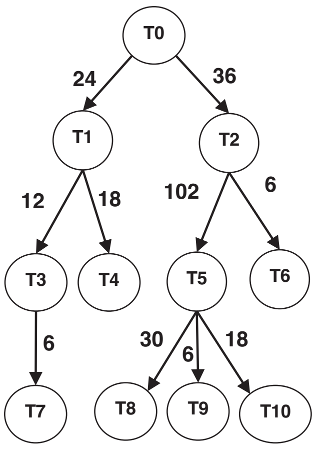

Fig. 2. Modified task graph. It contains four additional tasks T7-T10 (compare with Fig. 1).

3. New problem statement

3.1. Overview

Embedded system must execute the designated tasks. All of existing design methodologies assume constant number of tasks. Therefore in Hardware/Software co-synthesis process the database includes time and cost of execution of all predicted tasks. We decided avoid this assumption -unexpected tasks can appear during the design process. Such a problem can be easily resolved by allocation of new components or by new tasks assignment. Therefore much more interesting is situation when all the system is designed and realized. Than it is not possible to extend it by adding new resources. We will concentrate on this problem.

We assume that the embedded system is realized and than during its operation unexpected tasks appear. The tasks can also appear after (or during) re-design, partly re-design or design re-use of the system. The system was not originally designed to execute such tasks. Allocation of those tasks can for example extend possibilities and decrease costs of smartphones. It can be also important in space industry for robots working on other planets (moons) in the solar system. In such case there is no possible to change the architecture after unexpected tasks appear. Sending a new robot is also too expensive.

The new tasks are inserted in the task graph as separate nodes. Depending on the situation unexpected tasks can appear between the original tasks or after them. If unexpected tasks appear between original tasks the structure of the task graph will have to be modified. Unexpected tasks can be split to a number of subtasks (which can be executed on separate resources). In such a situation all of the subtasks must be present in the task graph as separate nodes. In some rare situations, especially when a task consists of a few subtasks, some tasks (subtasks) will have to start theirs execution at the same time.

3.2. New representation

In this work we also assume that embedded system under consideration is described by a task graphs. However the graph is modified during the design process. Fig. 2 presents an example of a modified task graph from Fig. 1.

The Fig. 2 indicates that there are four unexpected tasks: T7, T8, T9 and T10. Task T5 is a predecessor of tasks: T8, T9 and T10. Task T3 is a predecessor of task T7. In this case all additional tasks are parallel and do not modify the order of previous tasks and structure of graph presented on Fig. 1. We must underline that in