Large Exploration for HW/SW partitioning of Multirate and Aperiodic Real-Time Systems

Abdenour Azzedine, Jean-Philippe Diguet, Jean-Luc Pillippe
Université de Bretagne SUD; LESTER lab.; Lorient, France
jean-philippe.diguet@univ-ubs.fr

ABSTRACT

This paper addresses the domain of fine and coarse grain HW / SW codesign for Real-Time System On-Chip. We propose a new method for the real-time scheduling and the HW / SW partitioning of multi-rate or aperiodic tasks. The large design space exploration is based on parallelism/delay trade-off curves.

Keywords

HW / SW Codesign, system design exploration, RT scheduling

1. INTRODUCTION

The use of on-chip heterogenous real-time systems is continuously growing in various areas like mobile communications, automotive, avionics, robots, ... These trends are sustained by the numerous advantages the real-time SOC (system-on-chip)) approach offers :

- the use of an RTOS facilitates the management of aperiodic tasks and real time constraints;
- dedicated HW modules can be implemented on the same chip to speed up critical applications tasks;
- the chip capacity enables to implement complex applications with a large set of architectural alternatives;
- the on-chip communication bandwidth makes HW / SW codesign really attractive;
- the processor core can be tailored reagrding the application in terms of co-processors and dedicated memory hierarchy.

However, the associated design space is conjointly exploding and consequently requires methods and tools to guide the designer towards an (the) interesting solution. First, the increase of application complexity means an increase of the specification granularity. Thus, for each task of an application, a lot of parallelism / cost / time trade-offs should be considered in order to perform an efficient system-level design. Secondly, the system must be able to optimally handle a set of applications which are characterized by both periodic (e.g. audio decoding) and aperiodic tasks which can have hard (e.g. game pad management) or soft (mail loading) real-time constraints. Then, depending on criticality and periodicity aspects, a design tool must address the trade-off between the respect of response times and the area and power requirements.

In this paper we propose a framework to combine the HW / SW partitioning and scheduling problems of real-time systems onto monoprocessor architectures. The specification is a multi-rate periodic and aperiodic task graph. Each task is described with a hierarchical control data flow graph. This graph is estimated for various time constraints in order to produce area/time trade-off curves. The method is based on an accurate static scheduling and a large branch and bound exploration. The rest of the paper is organized as follows. Section 2 places our work within the state of the art. Section 3 details our codesign context. In Section 4 we present the different steps of our exploration, partitioning and scheduling method. Sections 5 and 6 contain respectively results from a real-life application and a conclusion about this work and future developments.

2. STATE OF THE ART

Design of complex system on chip requires a specification based on a coarse grain granularity. Such a specification can be data-flow [2, 22], control-data flow (CDFG) [19, 13] or task-graph oriented [23, 18]. Furthermore, embedded systems can be both aperiodic or periodic and even multirate. Moreover, to perform an efficient design space exploration, a large set of parallelism/delay trade-offs must be associated with each node of the specification. For these reasons, we have chosen to specify the application with a hierarchical task graph which is well-adapted to the task-level real-time scheduling. Each task is then described with functions (i.e. CDFG) in order to efficiently explore several architecture solutions. The step detailed in this paper is located after a functional (CDFG) decomposition of the application followed by a task merging step [8]. Three kinds of architectures are usually targeted, the first one is based on monoprocessor architectures with hardware accelerators which can [21, 2] or not run simultaneously [20]. The second one addresses multiprocessor architectures [23, 5, 22] and the last one includes ASIP [7, 12] designs. In this paper, we focus on monoprocessor architectures where the processor can be *parameterized* with fine-grain coprocessors and communicate with coarse-grain accelerators. We first make this choice because their low-cost and low-power characteristics make them very interesting in practice for embedded systems [6]. Secondly, on-chip HW / SW monoprocessor systems can meet hard real-time constraints. Actually, integrated HW accelerators can benefit from the fact that intra and inter task parallelism can be exploited thanks to actual and future chip capacities [24]. Moreover, on-chip communications reduce the penalty due to communication overhead. Note that in this context, we consider that an Hw accelerator can be a DSP used as a slave module (dedicated to a single task). Our goal is to combine a large design space exploration with a real-time scheduling suitable to real-life cases in terms of multirate tasks and low or high priority aperiodic tasks. In the area of multiprocessor architectures single-rate [17] but also multi-rate preemptive [23, 3, 9] scheduling techniques have been proposed. However, in that case dependent tasks (or processes) have the same period and are clustered in independent subgraphs. Moreover, multiprocessor research don't really target a full design space exploration, basically hardware implementations are just considered as processors limited to a single task execution. In [18] Dave and al. propose the *association array* and task clustering to solve the multirate problem in codesign, but these concepts are really interesting only in the context of multiprocessors architectures. The method is extended in [3] to aperiodic tasks for which time slots are reserved within the hyperperiod. This technique is valuable if aperiodic tasks have strict response times but can lead to very costly design if the tasks are rarely launched or if their execution is not critical. In [21] aperiodic tasks are not included but the multirate issue is considered in the context of a monoprocessor HW / SW architecture. The method is based on the fixed priority scheduling theory [14, 16], however the design space exploration is limited to a fine grain solution (processor + coprocessor) or separately to a coarse grain architecture which include a processor and parametrized ASIC.The dependence question is solved while using the Inverse Deadline technique, but like in [3] the question of dependent tasks with distinct periods is not considered (for instance the issue of tasks release time). We have also chosen a static scheduling as it is the only solution to guaranty real time constraints but we've added a low cost scheduling technique for low priority aperiodic tasks. We considere an acyclic task graph mapped on HW / SW mono-processor architecture, we adress the dependencies between task while using task pipelining and priority assignments. Regarding previous work, the originality is a combination of the three following points (i) multirate real-time static scheduling (ii) criticality-based scheduling of aperiodic tasks (iii) large design space exploration combining fine and coarse grain HW / SW codesign.

3. FRAMEWORK

3.1 System specification

Basically, the system is specified with an acyclic task graph (TG). In our approach, a TG is one of the hierarchy level of the specification (cf. fig.1). The system level describes system inputs and outputs. At the task level (TG), a node represents a periodic or aperiodic task and an edge a data (communication channel) or a control (event) dependence between tasks. Note that task deadline constraints can be derived, for periodic tasks, from system input/output constraints as described in [1]. The resource constraints due to data sharing is indicated with a special node (||). Accordingly to real-time theory, the function level is a cluster of sequential or mutually exclusive CDFG which compose a task. Finally, at the instruction level a fonction is described as a hierarchical CDFG. A HCDFG top-down decomposition is performed as follows: a CDFG is built each time a conditionnal instruction is found, when no more conditionnal instruction is present a DFG is created (basic block). Thus, in a CDFG a node represents a conditionnal instruction, a data transfer, (H)CDFG or a DFG. In a DFG, which means the lowest hierarchy level, only remain elementary nodes representing data-transfer or data-processing.

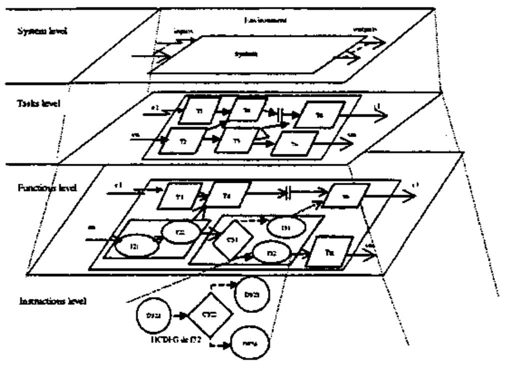


Figure 1: multi-level specification

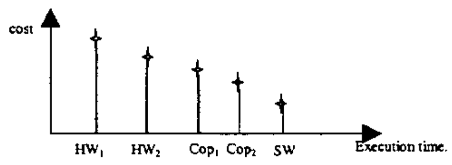


Figure 2: Task dynamic estimations

3.2 Target Architectural Model

In the context of the *Design Trotter* tool, each node (i.e. task) can be linked to a CDFG specification. Such a specification is an entry point for our *dynamic* estimation tool [15] which can provide a set of implementations from a sequential software to a highly parallel hardware module. The estimation result is a parallelism/delay trade-off curve as showed in fig.2.

There are three kinds of implementations. The first one is totally software and its cost is given by the program and data memory sizes. The second one, called *coprocessor solution*, is a software implementation but the processor core is upgraded with coprocessors. This is what is usually called a fine-grain codesign where specific instructions can be speed up by the use of dedicated functionnal units (multiplier, butterfly, MAC ...). There is no communication delay since the additionnal functional units are acceeded through the processing unit registers. Its cost is based in the program and data memory sizes and the coprocessors area. Note, that as coprocessors can be shared between tasks, the cost computation must take into account previous task implementations. Finally a hardware implementation represents a dedicated hardware module with its own control. Contrary to previous solutions, a HW module can run simultaneously with the processor but also implies communication delays. For instance, the different HW solutions correspond to different loop unfolding solutions [15], so to various trade-offs between parallelism exploitation (speed) and local memory size. A hardware module is dedicated to a given task. The different costs are computed as follows :

$$C_{SW} = DataMemSize + PgmMemSize$$
$$C_{Cop} = C_{SW} + \sum_i cost(i) * Max(PrevCop(i), Cop(i)))$$
$$C_{HW} = HWarea + LocalMemSize$$

$PrevCop(i)$ and $Cop(i)$ are respectively the number of coprocesseur of type i required by previous and current tasks. The architecture model is given in fig.3, it has been implemented on a FPGA Virtex 800 with the free soft IP LEON [11] which communicates with HW modules (for instance a DCT accelerator) through the amba bus. Moreover, the Leon's RTOS is RTEMS designed for multiprocessor architectures.

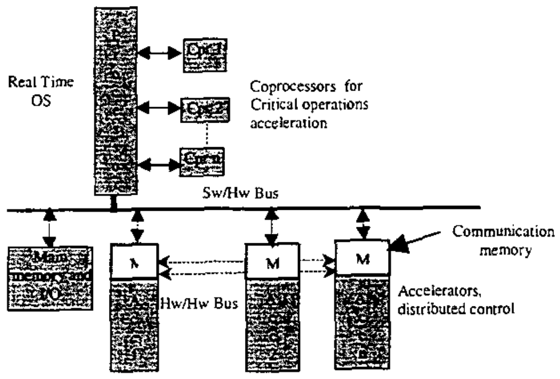

Real Time OS
Coprocessors for Critical operations acceleration
Sw/Hw Bus
Communication memory
M
M
M
Hw/Hw Bus
Accelerators, distributed control
Main memory and I/O

Figure 3: Architectural model

When a periodic task is implemented on a hardware module, the processor RTOS initiliazes the task execution by activating the module timer, then the hardware module runs independently from the processor scheduler. We will see in 4.3 that a real aperiodic task can not be scheduled on a HW component. The processor controls HW / SW communications by reading and writing data in dedicated hardware modules.

4. PARTITIONING / SCHEDULING METHOD

4.1 Basic concepts

Each task k is characterized by the following parameters (see fig.4):

- Task Id T_k
- Absolute Deadline D_k
- Period P_k
- Release time R_k
- Execution vector : $[C_k^1, ...C_k^M]$ where C_k^M is the delay associated with the M^{th} implementation.

Regarding aperiodic tasks, the period means the minimum delay between two successive executions of the task. In the case of hard real-time constraints (RTC), this delay is the lower bound but with soft RTC this delay is an average value.

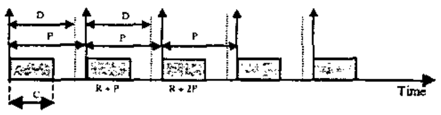


Figure 4: Tasks parameters

4.2 Scheduling Policy

4.2.1 Schedulability analysis

In order to guaranty RTC, we use a fixed priority static and preemptive scheduling. The small complexity of static schedulers is also an important issue in the domain of embedded real-time systems. Since the aim is to minimize resource slack time we don't use the rate monotonic analysis but we compute the task response time with the basic exact analysis [14, 16]. This analysis considers independent periodic tasks with the highest priority first (HPF) policy. The fixed priorities are computed as the inverse of the deadline. The exact analysis principle is detailed in 1, the complete formulation is given in 2. Thus, a task can be scheduled if the response time RT_i of T_i is such as : $\forall T_j \in hp(i), \exists RT_i \leq D_i$ with

$$RT_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{RT_i}{P_j} \right\rceil * C_j \qquad (1)$$

where $hp(i)$ is the set of SW tasks whith a higher priority than task i. As t is present on both sides in 1, it must be computed iteratively until $RT_i(n) = RT_i(n-1)$ or $RT_i(n) > D_i$

4.2.2 Resource sharing constraints

A task T_i can be delayed by lower priority tasks T_i if they share a critical resource (shared memory or I/O). Actually, a critical resource can't be used simultaneously by different tasks and a task must ends its job with this resource even if a higher priority task is waiting for it during its execution. B_i is defined as the longest lock time of the task i due to lower priority task T_j with $j \in lp(i)$ where $lp(i)$ is the set of tasks with a lower priority than task T_i. The complete response time analysis (eq.2) takes into account this locking time.

4.2.3 Communication delays

Communications delays are included in the execution time of software and coprocessor implementations. (They also include an average value of the RTOS overhead). In the HW / HW communications case, the data are transfered through shared memory and the memory accesses are included in estimated execution times. So, only SW / HW or HW / SW communications introduce new delays. These delays are due to the read and write instructions that execute the processor in the local HW accelerator memory. Thus, the exact analysis of task schedulability is finally : $\forall T_j \in hp(i), \exists RT_i \leq D_i$ with

$$RT_i = C_i + B_i + Tcom_i + \sum_{j \in hp(i)} \left\lceil \frac{RT_i}{P_j} \right\rceil * (C_j + Tcom_j) \qquad (2)$$

This is not the scope of this paper to detail precisely how HW/SW communication are correlated with the size of the HW accelerator, but we can indicate that it depends on four parameters given in eq.3 : T_{init} is the initialization delay of the communication, T_{data} is linked with the data format, D_{mem} is the number of data transfered to/from the local memory of an accelerator and N_{it} the number of times the accelerator is fired to execute the task (in a given period).

Thus, we observe a trade-off between the accelerator parallelism (i.e. memory and processing unit sizes) and the number of times the hardware module is used during the task execution.

$$Tcom_i = T_{init} + N_{it} * D_{mem} * T_{data} \tag{3}$$

Remark: we don't take into account the release jitter within the computation as explained in [16] and improved in [10], since the predecessors of a given task are either scheduled on the same processor or executed on a remote hardware module whose the exact execution delay is known and the scheduling date choosen in order to produce data intime as explained in 4.4.2.

4.3 Aperiodic tasks scheduling

Like in [3], we consider the lower bound of the delay between two successive executions as the strict period of an aperiodic task with a hard real time constraints. For instance the tool detailed in [1] produces an intervalle $[P_{min} P_{max}]$ for each aperiodic (sporadic) task within the application. Thus by selecting P_{min} one can guaranty real time constraints while systematically allocating time slots (HW or SW) to this task.

In practice, such a method cannot be applied if the task criticity doesn't justify a costly allocation of hardware resources or processor cycles. This trade-off between hard RTC and resource savings is handled by a task server [4]. This task has the lowest priority. The task server, whose period is fixed by the designer, represents an amount of time slots available for non critical aperiodic tasks which will be triggered in a FIFO order.

4.4 Dependent tasks scheduling

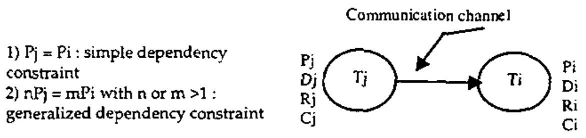
4.4.1 *Priority assignement & Release time shifting*

In real life examples, tasks communicate so are not independent as specified in 4.2. Usually, before applying the exact response time analysis, the dependencies which can be related to precedence constraints are eliminated while modifying absolute deadlines and release times. However, if we consider generalized data dependencies as shown in fig.5 we can avoid the dependencies constraints while considering a pipeline and static execution of dependent tasks. In that case, the dependencies problem resolution can be reduced to a correct choice of release times.

In a case as depicted in fig.5, we assume that $n.P_j = m.P_i$ means that the task i needs the data produced by n previous periods of task j, these data are then used during m periods of the task i (if the asumption is not true, the scheduling policy is not optimal but correct since this is the worst case). Actually, T_i and T_j can be considered as independent if T_i processes n data sets previously computed by the task T_j while T_j computes the n next data sets. Thus, the dependency constraint can be avoided by correctly shifting release times.
If $n = m = 1$ this is a single rate dependency.
If $n = m \neq 1$ this is a single rate dependency with latency constraints, i.e. the task T_i starts with the same period after n executions of the task T_j. So a new release time is computed.
So finally the priority assignment is simply given by : $Priority(T_k) \# \frac{1}{D_k}$

4.4.2 *Release time computation*

The schedulability analysis is performed while considering independent tasks, then when an implementation solution is found, the



Communication channel
1) Pj = Pi : simple dependency constraint
2) nPj = mPi with n or m >1 : generalized dependency constraint
Pj
Dj
Rj
Cj
Tj
Ti
Pi
Di
Ri
Ci

Figure 5: Precedence generalized constraints

new release times R_i^* are computed in order to respect the assumption, namely the precedence constraints. The computations are performed in the precedence order within the task graph. Four cases must be distinguished regarding the tasks $T_j \rightarrow T_i$ from fig. 5

1. HW $\rightarrow$ SW or HW $\rightarrow$ HW.
$$R_i^* = \max\left\{R_i; R_j^* + (n-1)P_j + C_j\right\}$$
T_j is implemented on a hardware module so its response time is equal to C_j (no interruption).

2. SW $\rightarrow$ HW.
$$R_i^* = \max\left\{R_i; R_j^* + (n-1)P_j + TR_j + Tcom_j\right\}$$
TR_j is the response time of T_j and $Tcom_j$ is the communication delay to be added to TR_j for the transfer of data from the processor to the hardware module memory

3. SW $\rightarrow$ SW and priority $(T_j) >$ priority (T_i)
$$R_i^* = \max\left\{R_i; R_j^* + (n-1)P_j\right\}$$

4. SW $\rightarrow$ SW and priority $(T_j) \leq$ priority (T_i)
$$R_i^* = \max\left\{R_i; R_j^* + (n-1)P_j + TR_j\right\}$$

TR_j is computed with the method of exact analysis (eq.2) but the set of conflicting tasks is no more $hp(j)$ but $hpp(j)$ which includes two kinds of software tasks : i) the predecessors of T_j and ii) higher priority tasks which are not successors of task T_j.

Remark : if T_i has several predecessors, R_i^* is computed for all its predecessors and the maximal result is selected.

In fig.6 an example of dependency resolution is given. The exact analysis shows that the four tasks are can be scheduled. Then we compute the associated release times while considering $T_{com} = 0$ for simplicity :

$R_1^* = R_2^* = 0, R_3^* = \max\{R_3; \max\{R_1^* + P_1 + TR_1; R_2^* + 4.P_2 + TR_2\}\} = 520$; $R_4^* = \max\{R_4; R_3^* + 2.P_3 + C_3\} = 960$

4.5 Principle of the partitioning algorithm

We currently use an exact *Branch & Bound* algorithm which find out the lowest cost solution regarding the silicon area (so the FPGA or Chip size).

The B&B tree is organized as follows : on a given branch, tasks are ordered according to their priority (highest first). And for a given level, implementations are ordered according to their costs (i.e. left edge solution is fully software).

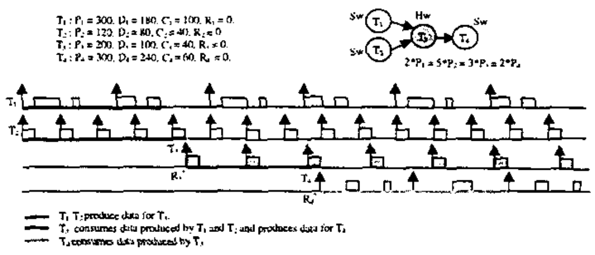

$T_1 : P_1 = 300, D_1 = 180, C_1 = 100, R_1 = 0.$
$T_2 : P_2 = 120, D_2 = 80, C_2 = 40, R_2 = 0$
$T_3 : P_3 = 200, D_3 = 100, C_3 = 40, R_3 = 0.$
$T_4 : P_4 = 300, D_4 = 240, C_4 = 60, R_4 = 0.$
Sw
Hw
Sw
Sw
$2*P_1 = 5*P_2 = 3*P_3 = 2*P_4$
T_1
T_2
T_3
R_3^*
T_4
R_4^*
T_1 T_2 produce data for T_3
T_3 consumes data produced by T_1 and T_2 and produces data for T_4
T_4 consumes data produced by T_3

Figure 6: Dependent graph scheduling example

The tree is travelled from left to right. So, if a schedulable solution is found for a given task (according to (2), the right side remaining solutions can be eliminated and the algorithm goes down to the next level (i.e. next task). The main difficulty of the method appears during the tree exploration when an HW solution is selected after a SW solution. In that case the algorithm must go up to the previous level to add the SW/HW communication time and recheck the schedulability of the SW solution.

4.6 Global Methodology

Finally the global methodology can be divided in five main steps :

1. *Multi-level system specification*;
2. *System-level estimation* : estimation tools are used to produce sw / sw+cop / hw trade-off curves;
3. *Priority assignment & aperiodic task management* : the priorities are computed and set of independent tasks is then considered, hard RTC aperiodic tasks are transformed into periodic tasks and weak RTC aperiodic tasks allocated to the server task;
4. *RT Static Scheduling / partitioning* : the B&B algorithm produces a low cost architecture. Regarding the processor slack time, the task server is defined or step 4 restarted with a task server period constraint.
5. *Task release time computation* : regarding multi-rate and precedence constraints, the correct task release times are provided to the RTOS.

5. RESULTS

In order to get derived constraints we reuse the *Dashboard controller* example given in [1] and described in fig.7 (T_i for $i \in [1..10]$). These tasks are aperiodic but have hard real-time constraints known through intervals $[P_{min}P_{max}]$. So, we consider these tasks periodic with a period equal to P_{min}. In order to extend the application scope, we imagine new tasks to be handled by the system. First, we add two periodic tasks $T_{11,12}$ which address the speed control. Then, some aperiodic tasks with soft RTC like air conditioning control (T_{13}) or board light control T_{14} are introduced as candidates to cluster within a the task server. Since we don't have tasks codes neither HCDFG descriptions, we can only propose coarse sw / sw+cop / hw estimates which are given in fig.8. However, the goal here is mainly the illustration of our approach combined with constraint derivation methods.

First, we only address critical tasks, the task server is considered afterwards regarding the processor slack time. The designer can define a minimum slack time allocated to tasks server, If this delay is no sufficient then the server task, with a minimum period, is included in the initial task vector. So, our tool input data are the task vector : $VT = [T_1,..T_{12}]$ and the period / deadline vector : $VP = [228, 456, 456, 19, 138168, ..., 1000000](10^{-5})s$, the implementation arrays (fig.8), the communication descriptors (T_{init}, T_{dma}), the task descriptors (D_i, R_i) and the task graph.

- In a first step, priorities Py_i are computed : $Py_4 = 1, Py_1 = 2, Py_2 = 3, Py_3 = 4, Py_5 = 5, Py_6 = Py_7 = 6, Py_8 = 7, Py_9 = Py_{10} = 8, Py_{11} = 9, Py_{12} = 10$
- Second step, B&B search : $best_{cost} = 1668$ [(T_1,Sw), (T_2,Cop), (T_3,Hw), (T_4,Sw), (T_5,Sw), (T_6,Cop), (T_7,Cop), (T_8,Sw), (T_9,Sw), (T_{10},Sw), (T_{11},Sw), (T_{12},Sw)]
- Third step, release time computation : $R2^* = R3^* = R11^* = R12^* = 288, R4^* = 348, R5^* = R6^* = R7^* = R8^* = R9^* = R10^* = 137940$
- In this example, the tasks T_{13}, T_{14} can be assigned to the tasks server (with a priority equals to 11) because the processor slack time equals 0.1%. Both tasks will be correctly scheduled if they don't require more than 0.1% of the processor time, in the opposite case their execution will be delayed.

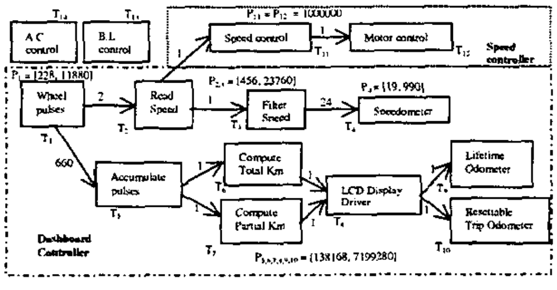


Figure 7: Car controller task graph

T_1	SW_1
C	60
Cost	120

T_2	SW_2	Cop_{21}
C	200	120
Cost	160	100

T_3	SW_3	Cop_{31}	HW_3
C	150	120	60
Cost	130	100	300

T_4	SW_4
C	6
Cost	90

T_5	SW_5	HW_5
C	220	150
Cost	400	900

T_6	SW_6	Cop_{61}
C	60	40
Cost	80	50

T_7	SW_7	Cop_{71}
C	50	35
Cost	70	45

T_8	SW_8	HW_8
C	360	210
Cost	190	430

T_9	SW_9
C	70
Cost	85

T_{10}	Sw_{10}
C	50
Cost	96

T_{11}	Sw_{11}	Hw_{11}
C	400	120
Cost	160	220

T_{12}	Sw_{12}	Hw_{12}
C	165	60
Cost	135	198

	Cop_1	Cop_2	Cop_3
Cost	80	96	112
cop_{21}	1	0	1
cop_{31}	0	1	1
cop_{61}	1	0	0
cop_{71}	1	0	0

Coprocessor Implementation costs :
The "Cop_{ij}" cost only includes the Sw cost as "Cop_{ij}" solution can share coprocessors, the final cost is obtained by adding the cost of allocated coprocessors.

Figure 8: SW/Cop/HW solutions

6. CONCLUSION

In this paper we have presented the Scheduling and Partitioning task for Real-Time Embedded Systems of our design tool called *Design Trotter*. The method is based on a static fixed priority scheduling and we propose a solution to schedule multirate and aperiodic tasks. The technique takes advantage from the task pipelining while the assumption of independent tasks is guaranteed by the shifting of release times. The partitioning algorithm is linked to dynamic estimation task which provides trade-off curves in order to perform a design space exploration according to the huge VLSI potential of future system on chip. Our future work will address the refinements of communication and RTOS overhead aspects. Currently a heuristic approach of the B&B algorithm is under development in order to adress large task graphs.

7. REFERENCES

[1] A.Dasdan, D.Ramanathan, and R.K.Gupta. Rate derivation and its application to reactive, real-time embedded systems. In 35th ACM/IEEE *Design Automation Conf.*, San Francisco, USA, 1998.

[2] A.Kalavade and E.Lee. Global criticality/local phase driven algorithm for the constrained hw/sw partitioning problem. In *Int. Work. on H/S Codesign*, Sept. 1994.

[3] B.P.Dave, G.Lakshminarayana, and N.K.Jha. COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems. IEEE *Trans. on Software Engineering*, 7(1), Mar. 1999.

[4] B.Sprunt, L.Sha, and J.P.Lehoczky. Aperiodic task scheduling for hard real-time systems. *Journal of real-time systems*, Sept. 1989.

[5] D.Kirovski and M.Potkonjak. System-level synthesis of low-power hard real-time systems. In 34th ACM/IEEE *Design Automation Conf.*, San Francisco, USA, 1997.

[6] F.Balarin and A. Sangiovanni-Vincentelli. Schedule validation for embedded reactive real-time systems. In 34th ACM/IEEE *Design Automation Conf.*, Anaheim, USA, 1997.

[7] F.Charot and V.Messé. A flexible code generation framework for the design of application specific programmable processors. In 7th *Int. Work. on H/S Codesign*, Roma, Italy, May 1999.

[8] H.Gomaa. *Software Design Methods for Concurrent and Real-Time Systems*. Addison-Wesley, 1993.

[9] H.Oh and S.Sha. A hw/sw co-synthesis technique based on heterogeneous multiprocessor scheduling. In ACM *Int. Symp. CODES*, Roma, May 1999.

[10] J.C.Palencia and M. Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In 20th *Real-time System Symp.*, Scottsdale, USA, Dec. 1999.

[11] J.Gaisler. Leon sparc processor core. http://www.gaisler.com/leon.html.

[12] K.Küçükakar. An ASIP design methodology for embedded systems. In 7th *Int. Work. on H/S Codesign*, Roma, Italy, May 1999.

[13] M.Auguin, L.Capella, F.Cuesta, and E.Gresset. Synthesis of signal processing systems from binary controlled data-flow specifcations. In *Int. Conf. on Signal Proc. Appli. & Techno.* (ICSPAT), Dallas, USA, Oct. 2000.

[14] M.Joseph and P.Pandya. Finding response time in a real-time system. IEEE *Design and Test of Computers*, 29(5):390–395, 1986.

[15] Y. Moullec, J-Ph.Diguet, and J-L.Philippe. Fast and adaptive dataflow and data-transfer scheduling for large design space exploration. In 12th ACM *Great Lake Symposium on VLSI*, New York, USA, Apr. 2002.

[16] N.Audsley, A.Burns, M.Richardson, K.Tindell, and A.J.Welling. Applying new scheduling theory to static priority preemptive scheduling. *Software Egineering Journal*, pages 284–292, sep 1993.

[17] P.Bjørn-Jørgensen and J.Madsen. Critical path driven cosynthesis for heterogeneous target architectures. In 5th *Int. Workshop on H/S Codesign, Codes/CASHE'97*, Braunschweig, Germany, Mar. 1998.

[18] P.Dave and N.K.Jha. CASPER: Concurrent hardware-software co-synthesis of hard real-time aperiodic specification of embedded system architectures. In *Design, Automation & Test in Europe Conf.*, Paris, France, Feb. 1998.

[19] P.Eles, K.Kuchcinski, Z.Peng, A.Doboli, and P.Pop. Scheduling of conditionnal process graphs for the synthesis of embedded systems. In *Design, Automation & Test in Europe Conf.*, Paris, France, Feb. 1998.

[20] R.Gupta and G. Michelli. Hardware-software cosynthesis for digital systems. IEEE *Design and Test of Computers*, Sept. 1993.

[21] R.Kamden, A.Fonkua, and A.Zenatti. Hardware/software partitioning of multirate system using static scheduling theory. In IEEE *Int. Conf. on Computer Design*, Texas, Sept. 1999.

[22] T.Grandpierre, C.Lavarenne, and Y.Sorel. Optimized rapid prototyping for real time embedded heterogenous multiprocessors. In ACM *Int. Symp. CODES*, Roma, May 1999.

[23] T.Y.Yen and W.Wolf. Sensitivity-driven cosynthesis of distributed embedded systems. In 8th IEEE/ACM *Int. Symp. on System Synthesis*, Cannes, France, Sept. 1995.

[24] W.R.Davis, N.Zhang, K.Camera, F.Chen, D.Markovic, N.Chan, B.Nikolic, and R.W.Brodersen. A design environment for high throughput, low power,dedicated signal processing systems. In IEEE *Custom Integrated Circuits Conference, CICC'2001*, San Diego, CA, May 2001.