Challenges for Multi-function SoC Scheduling and Assistive Living Services

Chi-Sheng Shih
Department of Computer Science and Information Engineering
Graduate Institute of Networking and Multimedia
National Taiwan University, Taipei, Taiwan 106
cshih@csie.ntu.edu.tw

1 Introduction

Tasks on assistive living devices and multi-function SoC (MFSoC) have the requirements to meet certian timing constraint. Examples include the end-to-end throughput constraint on MFSoC and deadline constraint on assistive living devices. However, the workload model on the two application domains are different from conventional real-time workload models. The new workload model, hence, impose new scheduling challenges on real-time computing community. In this short paper, we will discuss the new workload models and the challenging issues on scheduling such tasks.

2 Challenges on task scheduling on MFSoC

Multiple function SoC design are now popular for embedded multimedia devices and consumer electronics because of low energy consumption, and compact chip size. Take video phone as an example. Figure 1 illustrates the task flows on a video phone. In this example, both H.264 decoding flow and MP3 decoding flow require Inverse Discrete Cosine Transform (IDCT) and deQuantization (deQ) (Note that enhanced implementation for the shown functions may be used to optimize the performance but, without loss of generality, we only show the basic implementations.) We can reduce manufacture cost by allowing H.264 decoding and MP3 decoding task flows to share the processing elements for IDCT and deQ. Unlike general microprocessors, task execution on such processing elements (or called IP) cannot be interrupted. Otherwise, the result is lost and the task fails. The system needs to start the task flow from the beginning. Consequently, the task may fail to meet its performance requirement such as jitter requirement. We can avoid such failures by adding memory to queue up the inputs for the upcoming requests, and delay the requests until the requested processing element is available. However, without properly designed buffering and scheduling mechanism, the system performance could be worsen in certain cases.

First of all, adding buffers to the systems may prolong task response times. It is mostly caused by the scheduling anomaly which occurs when non-preemptible processing elements are used in the systems. In particular, it may turn the best-case performance into worst-case performance in the platform-base SoC [5]. The literature in distributed (hard) real-time computing community [4] show that it is extremely difficult to explore such anomaly. Second, adding buffers to the systems also increases the chip size, the manufacture cost, and energy consumption. Figure 2 shows a bluetooth chip designed by Ericsson in which more the memory occupies more than half of the chip area.

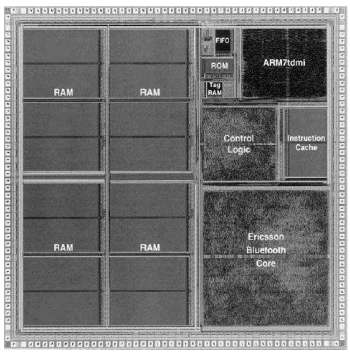

Figure 2. Layout of Ericsson-VLSI Bluetooth Chip [8]

One alternative for adding buffers is using external shared memory to queue up the requests. However, this approach greatly increases the communication overhead due to the resource contention on bus and memory controller.

There are three major challenges for meeting the timing requirements for MFSoC.

- **Worst-case response time analysis**: analyzing the worst case response times for all the tasks allows the designers to foresee the timing behaviors of the tasks during the design time. As discussed above, the tasks on the multi-function SoCs are executed on non-preemptive processing elements and have arbitrary release times. However, there is no poly-nominal algorithm for worst case response time analysis for such workloads.
- **Optimal buffer allocation**: buffer allocation for the processing elements on an SoC has a significant impact on the schedule for bus transactions and task executions. An optimal buffer allocation should minimize the size of

Dagstuhl Seminar Proceedings 08071
Scheduling
http://drops.dagstuhl.de/opus/volltexte/2008/1486

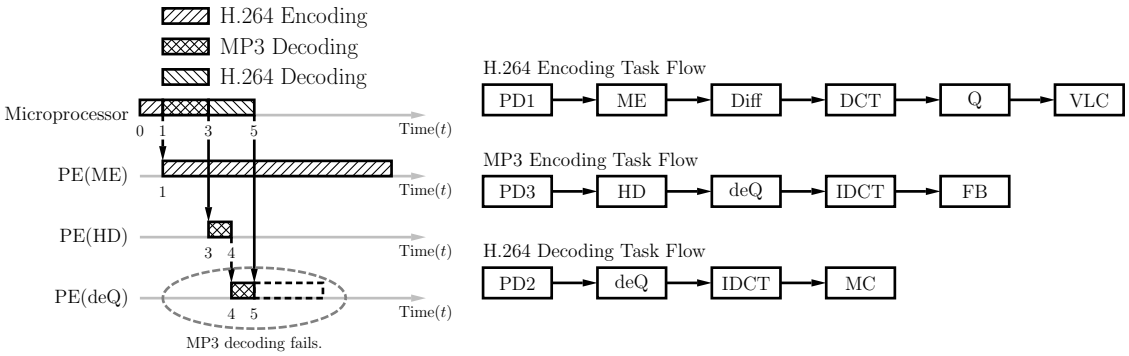

Figure 1. Video Phone Application SoC

buffer so as to reduce the manufacture cost and minimize the impacts on task scheduling.

- **Efficient run-time scheduling**: during the run-time, scheduling decisions are made for every few milliseconds or less. Hence, the mechanism for selecting the tasks to occupy the processing elements should be efficient and scalable in terms of the system workloads.

Below we discuss the causes of the hardware contention between the subtasks and develop a novel approach to avoid such contention. Before moving on, we digress briefly to answer a question that may arise: Why not simply add N-copies buffer for each processing element to queue all new arrivals and avoid hardware contention? Although adding N-copies buffer may avoid or eliminate hardware contention, there are several drawbacks to do so. First of all, adding intermediate buffer increases the chip size and, hence, the manufacturing cost. Secondly, a dispatcher is required for each processing element to store new arrivals and the requesting queue can be updated accordingly. Last but not least, it becomes too complex to simulate or analyze the timing behaviors for the system. Specifically, when non-preemptible resources such as processing elements are shared among tasks, scheduling anomalies may occur [1, 7, 3, 2], as discussed in detail earlier. As a result, adding N-copies buffer in every processing element not only increases the complexity of designing processing elements but also causes unpredictable timing behavior for the system.

According to our observation, a contention free schedule can be found by controlling the starting time of a sequence of hardware component request. The starting time control can be implemented on microprocessors by controlling the departure time of computation task on microprocessors. An example shown in Figure 3 illustrates the rationale of controlling the starting times of task flows.

The example uses the same task flow given in Figure 1. The release times of τ_1, τ_2 and τ_3 are 1, 3 and 5, respectively. As a reminder, subtask $\tau_{3,1}$ will content Π_{deQ} with subtask $\tau_{2,2}$ at time 5 when every task is started at its release time in FIFO manner as shown in Figure 1. The starting time guarder intentionally delays the starting time of τ_3 (for instance, hold τ_3 on the microprocessor) until time 9. At time 9, the guarder triggers subtask $\tau_{3,1}$ to execute function "deQ." By controlling the starting times for task flows, the hardware contention between subtasks $\tau_{2,2}$ and $\tau_{3,1}$ are avoided.

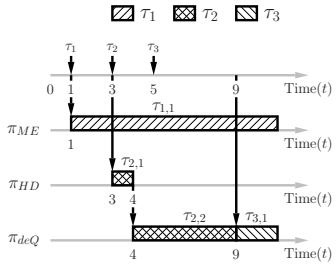

Figure 3. A contention free schedule by controlling staring time.

The major challenges for the approaches are twofold: (1) how to predict the hardware contention at a low run-time overhead when a task flow arrives and (2) how to analyze the worst case response time for each task flow so as to conduct the schedulability test. An intuitive approach for avoiding hardware contention is to analyze all possible combinations of release times, and it is not difficult to show that the solution space exponentially grows.

2.1 Observation on Greedy Approach

One simple way to avoid processing element contention is adding N-copies buffer for every processing element. Intuitively, this approach can queue up the arrival data and increases the utilization of the processing elements. Hence, the total response time should be shorten. However, it is only applicable for independent processes. When the non-preemptible resources are shared among task flows, which is very common in multi-function SoC design, scheduling anomalies occur. Figure 4 and Figure 5 show one example for scheduling anomalies. Two task flows in Figure 4 and Figure 5, τ_1 and τ_2, are used to illustrate the scheduling anomaly problem.

Task flows τ_1 and τ_2 share two processing elements: π_4 and π_7. Vertical arrows in the figures represent the release of subtasks. We assume that the release times of $\tau_{1,1}$ and $\tau_{2,1}$ are time 0 and 2, respectively. When there are N-copies buffer at π_4 and π_7, and STG algorithm is not applied, Figure 4 shows the schedule result. $\tau_{1,3}$ is blocked from time 9 to time 12 and $\tau_{2,6}$ is blocked from time 19 to time 26 for processing element contention. The response times of τ_1 and τ_2 are 26 and

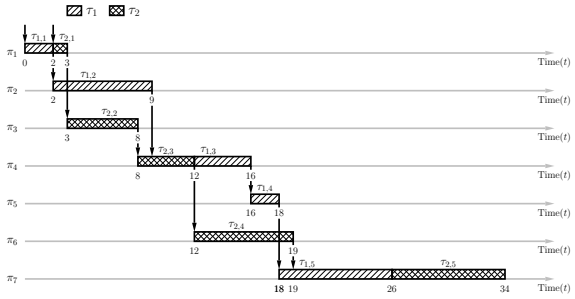

Figure 4. N-copies Buffer Schedule Example

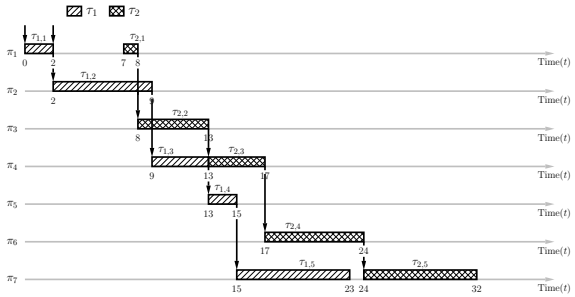

Figure 5. One-copy Buffer Schedule Example

34, respectively. When there is only one copy buffer in every processing element and STG algorithm is applied, Figure 5 shows the schedule results. To avoid processing element contention, the starting time of $\tau_{2,1}$ is delayed for 5 units of time. In other words, subtask $\tau_{2,1}$ on π_1 does not start until time 8. As a result, the response times of task flows τ_1 and τ_2 are 23 and 32, respectively.

As shown in Figure 4, queuing up data on processing element buffer might not help in completing a sequence of subtasks earlier, while no delaying of subtask starting time is provided. This is because the buffering may change the execution order of succussing subtasks and prolong the response time for some of the tasks. Although it is possible to derive the optimal starting times under certain assumptions, the challenges remain for general cases: (1) subtasks might share processing elements and have precedence constraints among them, and (2) a subtask has its flexibility to delay its starting time, and the starting time delay of any subtask could have an impact on that of other subtasks.

We must point out that even though it is very difficult to have an optimal solution to determine a starting time delay for each task flow, we could still have a pessimistic bound on how long a subtask could be delayed by subtasks of other task flows. For example, if a processing element might be shared among m subtasks, each subtask would not be delayed for the execution on the processing element for more than $(m-1)c$ units of time, where c is the execution time on that processing element.

2.2 Challenges for Buffer Placement on MFSoC

In this section, we discuss the challenges for buffer placement of MFSoC. By the observation in Section 2.1, the readers might note that the greedy approach does not always introduce the minimum response time on task flows. However, task flows by STG algorithm will be longer than that of adding N-copies buffer on every processing element in most cases. In this section, we propose an algorithm to minimize the response time by adding "some" processing elements with N-copies buffer and STG supporting. The motivating example is shown in Figure 6 and Figure 7.

Task flows τ_1 and τ_2 share three processing elements: π_4, π_5 and π_6. Vertical arrows in the figures represent the release times of subtasks. We assume that the release times of $\tau_{1,1}$ and $\tau_{2,1}$ are time 0 and 2, respectively. When π_4 has N-copies buffer and the STG algorithm is applied, Figure 6 shows the schedule result. To avoid processing element contention, the starting time of $\tau_{1,3}$ is delayed for 1 units of time in π_4. The response times of τ_1 and τ_2 are 27 and 22, respectively. When there is only one copy buffer in every processing element and STG algorithm is applied, Figure 7 shows the schedule results. To avoid processing element contention, subtask $\tau_{2,1}$ on π_1 does not start until time 8. As a result, the response times of task flows τ_1 and τ_2 are 23 and 28, respectively. The reader might note that the response times of each task in Figure 6 will be the same as adding N-copies buffer on every processing element in the same task flow set. However, this greedy approach will need more buffer size than Figure 6.

We can prove the NP-completeness of buffer placement by showing that Knapsack problem[6] is as hard as the problem. One can consider the makespan of the task flow set as the profit, processing element as an object and the size of N-copies buffer of a processing element as the cost of an object, in the Knapsack problem. In this section, we shall propose an algorithm for the multi-flow buffer minimization problem. A pseudo-polynomial time dynamic programming-based algorithm is proposed to solve the problem.

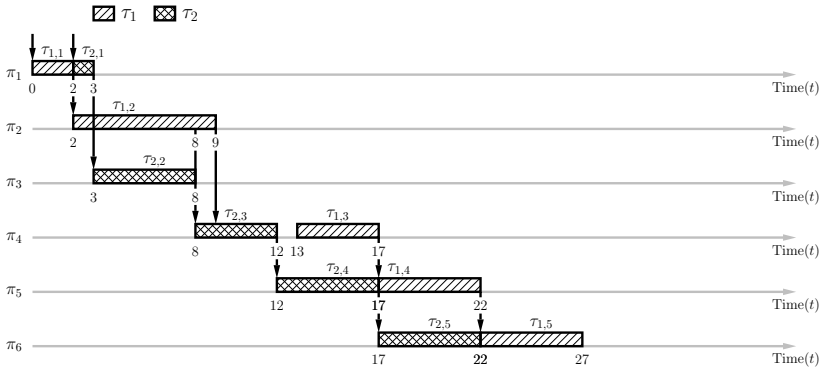

Figure 6. π_4 with N-copies Buffer Schedule Example

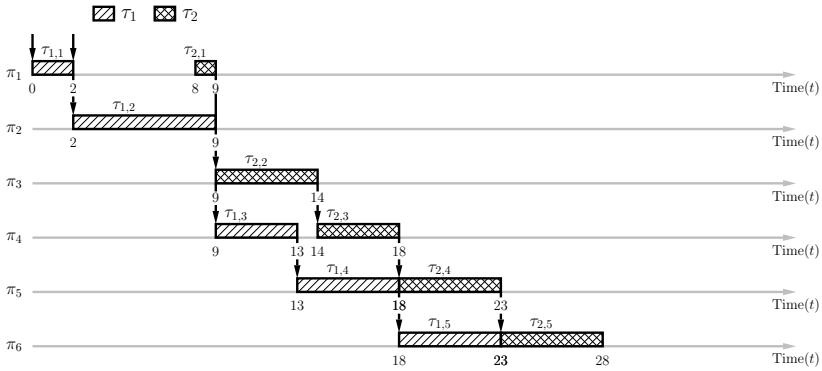

Figure 7. One-copy Buffer Schedule Example

3 Challenges for Integrating Assistive Living Devices

The use of medications is ubiquitous in terms of location and time. According to the report issued by Committee on Identifying and Preventing Medication Errors in 2007 [**?**], more than 75% of U.S. adults take one medication (including prescription or over the counter drugs, vitamin/mineral, or herbal supplement.) Among them, more than thirty percent take five different medications per week. Unfortunately, errors in the medication-use process can occur at any point of the process and in any care-setting. For instance, the report [**?**] estimates that on average, at least one medication error per day per patient, with considerable variation in error rates across facilities. Such medication errors harm at least 1.5 million people every year in United States, according to the same report. The extra costs for treating drug-related injuries occurring in hospitals alone conservatively amount to 3.5 billion US dollars a year, not including lost wages and productivity.

Medication-use system includes the prescription entry sub-system in hospitals and clinics, prescription refill subsystems in pharmacies, and medication dispensing subsystems for medication users. Figure 8 illustrates the major sub-systems in medication-use systems and the interactions between them. Fortunately, many of the medication-use injuries are preventable. Among the preventable injuries, one forth occurs in hospitals, one half occurs in long-term care centers and the remainder occurs in outpatient clinics. How information about a drug is communicated to providers and consumers can directly affect the frequency of medication errors. Gandhi et al.[**?**] suggest that computerized prescribing will be important in the outpatient setting, although it may not yield significant safety benefits without added decision support. Equally important are likely to be approaches that improve communication between patients and providers.

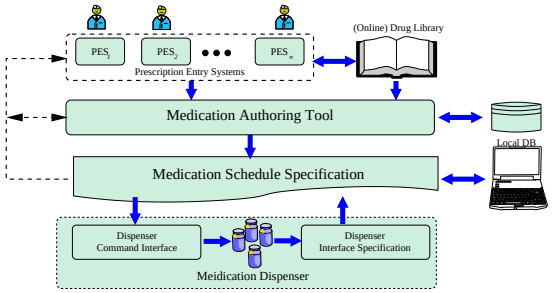

Figure 8. Medication Process

Integrating medication sub-systems and automating the medication process enhance the medication compliance and prevent the medication errors, caused by communication mistakes between care-takers and medical institutes. There are many attempts to integrate medication-use sub-systems to help the patient in hospital or outpatient settings to comply with medication directions. Many of them are designed and implemented with typical medication-use process as shown in Figure 8 and most of the systems are implemented as a tightly-coupled close system.

A tightly-coupled close system has its own merits. For

instance, it may have better performance, compared to loosely coupled systems. However, it also has several drawbacks.

- **Interoperability**: interoperability is the ability of two or more systems or components to exchange information and to use the information that has been exchanged. A close system may use proprietary message exchange protocol, communication protocol, or message format. As a result, other systems have less chance to exchange information with it. As discussed earlier, medication-use system is a dynamic system. The participant and how the participant interact with each other change over time. When a new participant joins or an existing feature is revised, it may lead to major revision to other sub-systems in the system.

- **Reliability**: a close system assumes that the participants of the system are known and static. Hence, how the participants interact with each other is defined in the system and is difficult to revise. Hence, when some of the participants malfunction, the system may not be able to function well and, hence, have less reliability.

References

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In IEEE Real Time System Symposium, pages 4–13, 1998.

[2] Y.-S. Chen, L.-P. Chang, T.-W. Kuo, and A. K. Mok. Real-time task scheduling anomaly:observations and prevention. In ACM Symposium on Applied Computing, Mar 2005.

[3] R. Graham. Bounds on the performance of scheduling algorithm. Computer and Job Shop Scheduling Theory, E.G.Coffman, pages 165–227, 1976.

[4] John A. Stankovic, Marco Spuri, Marco Di Natale, and Giorgio C. Buttazzo. Implications of classical scheduling results for real-time systems. Computer, 28(6):16 – 25, June 1995.

[5] K. Richter, M. Jersak, and R. Ernst. A formal approach to mpsoc performance verification. Computer, 36(4):60 – 67, April 2003.

[6] M. R. Garey and D. S. Johnson. Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman, 1979.

[7] A. K. Mok. Tracking real-time systems requirements. In Workshop on Modelling Software System Structures in a fastly moving scenario, June 2000.

[8] S.B. Furber. ARM System-on-Chip Architecture. Addison Wesley Longman, 2000.