A Period Assignment Algorithm for Real-Time System Design*

Minsoo Ryu and Seongsoo Hong

School of Electrical Engineering and ERC-ACI,
Seoul National University, Seoul 151-742, Korea.
{msryu,sshong}@redwood.snu.ac.kr
http://redwood.snu.ac.kr

Abstract. Digital controllers found in many industrial real-time systems consist of a number of interacting periodic tasks. To sustain the required control quality, these tasks possess the maximum activation periods as performance constraints. An essential step in developing a real-time system is thus to assign each of these tasks a constant period such that the maximum activation requirements are met while the system utilization is minimized [3].
Given a task graph design allowing producer/consumer relationships among tasks [4], resource demands of tasks, and range constraints on periods, the period assignment problem falls into a class of nonlinear optimization problems. This paper proposes a polynomial time approximation algorithm which produces a solution whose utilization does not exceed twice the optimal utilization. Our experimental analysis shows that the proposed algorithm finds solutions which are very close to the optimal ones in most cases of practical interest.

1 Introduction

Real-time systems often consist of a number of interacting tasks. Most of these tasks execute *periodically* at fixed rates reading data from producer tasks and writing data to consumer tasks. A typical example is a feedback control system. A control task periodically samples input data, computes control laws, and generates output for actuators, and all these activities should be done periodically.

In order to sustain the required quality of control, each real-time task possesses as its timing constraint the maximum activation period that is derived from a given performance specification. An essential step in developing a real-time system is thus to assign each task a constant period such that the maximum activation requirements are met while the system utilization is minimized. Where intermediate tasks are shared by several others that run with different rate constraints, the problem falls into a class of nonlinear optimization problems.

* The work reported in this paper was supported in part by Engineering Research Center for Advanced Control and Instrumentation (ERC-ACI) under Grant 96K3-0707-02-06-3 and by KOSEF under Grants 97-0102-05-01-3 and 981-0924-127-2.

W.R. Cleaveland (Ed.): TACAS/ETAPS'99, LNCS 1579, pp. 34–43, 1999.

In [3] Gerber *et al.* addressed this problem as part of the real-time system design problem and formulated it into a nonlinear optimization problem in their design methodology. They proposed a set of heuristics which can be used to reduce the search space of the problem. Unfortunately, these heuristics possess only limited utility, since they may still leave a inhibitively large search space even for a problem with a modest size. To the best of our knowledge, no polynomial time approximation algorithm with a known performance bound has been proposed to address this period assignment problem.

A similar problem was addressed by Seto *et al.* [9] for the design of real-time control systems. Given that the control performance requirement and the schedulability constraint determine the ranges of sampling periods, their proposed algorithm derives schedulable sampling periods that optimize the performance index of the control system. However, the application model in [9] is rather simple: in [9], all tasks in a system run independently, while the task graph in [3] allows producer/consumer relationships and data sharing between tasks.

In this paper, we propose a polynomial time period assignment algorithm for the real-time design problem formulated in [3]. Our algorithm makes use of two effective heuristics, namely (1) optimal GCD assignment; and (2) harmonic period assignment among tail tasks. In the first heuristic, a producer task is assigned as its period the greatest common divisor of the periods of its consumers. In the second, tasks with no consumers (tail tasks) are assigned harmonic periods. We formally prove via the worst case analysis that the proposed algorithm with these heuristics yields a solution whose utilization does not exceed twice the optimal utilization. In reality, our experimental analysis shows that the algorithm produces solutions which are very close to the optimal ones in most cases of practical interest.

This paper is organized as follows. In section 2, we give a formal specification of the period assignment problem by defining a task graph design of a real-time system, along with the constraints and the objective function of the problem. In section 3, we propose two heuristics and present the approximation algorithm. In section 4, we make a formal analysis of the proposed algorithm and prove its worst case performance bound. In section 5, we experimentally evaluate the algorithm by comparing it with an optimal algorithm using exhaustive search. Section 6 concludes the paper.

2 Problem Formulation

A real-time system in our problem is represented by a directed acyclic graph $G(V, E)$ where V is a set of tasks $\{p_1, p_2, \ldots p_n\}$ and E is a set of directed edges between two tasks. Each task has a bounded execution time e_i which can be estimated in various ways [7]. An edge $p_i \rightarrow p_j$ denotes a *producer/consumer* relationship in that p_i produces data that p_j consumes [4].

Let T_i be the period of p_i. Each producer/consumer pair $\langle p_i, p_j \rangle$ is constrained to have *harmonic* periods such that T_j is an integer multiple of T_i, which is denoted "$T_i | T_j$". Figure 1 shows an example task graph with their

assigned harmonic periods. If these tasks ran with arbitrary periods, task executions would get out of phase resulting in large latencies in communication. The harmonicity constraint ensures that the two tasks stay "in-phase," thus reducing communication latencies [3]. It also guarantees predictable data flow from a common producer to multiple consumers. For example, in Figure 1, it is guaranteed that p_4 receives every data item p_3 generates while p_5 gets every third item of what p_3 produces. In the remainder of the paper, we use the following additional notations.

notation	description
$\mathcal{P}_{tail}$	A set of tasks having no outgoing edges.
$\mathcal{P}^i_{tail}$	A set of tail tasks p_j which are reachable from p_i.
$\mathcal{P}^i_{succ}$	A set of tasks having an incoming edge from p_i.

Every output task $p_i \in \mathcal{P}_{tail}$ possesses a range constraint on T_i such that $1 \leq T_i \leq T_i^u$ where T_i^u is the maximum period constraint which is derived from the performance requirement. Due to harmonicity constraints, each non-tail task p_j is also subject to a period constraint such that $1 \leq T_i \leq T_i^u$ where T_i^u is the smallest maximum period of all consumer tasks. Therefore, T_i^u of non-tail task p_i is determined such that $T_i^u = \min\{T_j^u \mid p_j \in \mathcal{P}^i_{succ}\} = \min\{T_j^u \mid p_j \in \mathcal{P}^i_{tail}\}$.

The objective function of this period assignment problem is to maximize the chance of the system being schedulable. As in [3], we adopt, as the objective function, minimization of utilization $U = \sum_{i=1}^{n} \frac{e_i}{T_i}$. There have been a number of vastly different measures of schedulability depending on real-time scheduling algorithms such as nonpreemptive, calendar-based scheduling [11], and preemptive, static and dynamic priority scheduling [6]. In this paper, we assume a preemptive priority scheduling strategy. Note that in preemptive priority scheduling such as RMS and EDF, utilization is proven to be a sufficient measure of the schedulability of real-time systems [6,5].

Finally, the problem at hand is stated as follows.

> "*Given a task graph and the range constraints on task periods, assign each task a harmonic period such that the range constraints are met and the system utilization U is minimized where $U = \sum_{i=1}^{n} \frac{e_i}{T_i}$.*"

3 Period Assignment Algorithm

Due to the harmonicity constraints and the objective function, the problem is nonlinear optimization problem. To find solutions in a reasonable amount of time, we propose an approximation algorithm which consists of two heuristic steps to minimize the utilization.

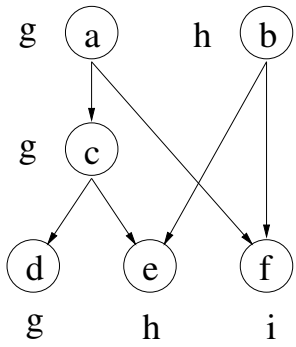

Fig. 1. Task graph.

3.1 Optimal GCD assignment

Our first heuristic is the optimal GCD (greatest common divisor) assignment. It is a backward period assignment method in that a non-tail task p_i gets period T_i such that $T_i = GCD\{T_j \mid p_j \in \mathcal{P}^i_{succ}\}$. Such period assignment starts from those non-tail tasks that are the immediate predecessors of tail tasks, and is iteratively applied to their predecessors until all non-tail tasks are assigned their periods. As a result of the optimal GCD assignment, a producer task always gets the largest possible period which is harmonic to the periods of its consumer tasks.

Theorem 1 proves that the optimal GCD assignment is a necessary step to obtain the optimal period assignment of a whole system.

Theorem 1. *If all tail tasks are given the optimal periods, the GCD assignment always finds the optimal period assignment for the whole tasks.*
Proof. We prove the theorem by contradiction. Suppose that the optimal solution has an intermediate task p_i which has a non-GCD period T_i. Let $T_i^{GCD} = GCD\{T_j \mid p_j \in \mathcal{P}^i_{succ}\}$. Due to the harmonicity constraint, T_i is a common divisor of the periods of tasks in $\mathcal{P}^i_{succ}$, and thus $T_i | T_i^{GCD}$. Hence, T_i can be replaced by T_i^{GCD} without affecting the periods of other tasks, and this yields a lower utilization. This contradicts the assumption. Therefore, the optimal solution is obtained by the GCD assignment method. □

Theorem 1 helps significantly reduce the problem size, since it allows us to focus only on tail tasks for the optimal period assignment.

3.2 Harmonic Period Assignment for Tail Tasks

Our second heuristic is the harmonic period assignment for tail tasks. In order to minimize the utilization, it is desirable to assign tail tasks the largest possible harmonic periods so that large GCD values are assigned to non-tail tasks. To do so, the algorithm introduces additional harmonicity constraints among the tail tasks. It then assigns tail tasks harmonic periods to satisfy this extra constraints.

This heuristic step works as follows. First, tail tasks are sorted in nondecreasing order of T_i^u. Second, a harmonicity constraint is established on each of the two adjacent tasks in the list. We can represent this additional harmonicity constraint by slightly modifying a given task graph. Figure 2 shows a modified task graph of Figure 1 after it is extended with extra harmonicity constraints.

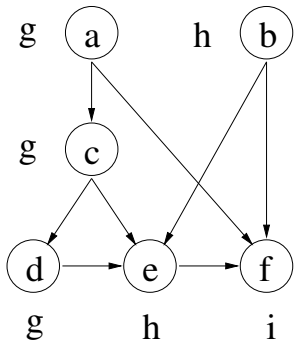

Fig. 2. Modified task graph with harmonicity constraints between tail tasks.

Due to the harmonicity constraints between tail tasks, the original problem is reduced into the problem of assigning a period to only the tail task with the smallest T^u.

The entire algorithm is presented below. In Step (1), it sorts tail tasks in the nondecreasing order of their maximum period constraints. In Step (2), the algorithm chooses a period value for the tail task with the smallest maximum period constraint. In order to bound the worst case performance of the algorithm, it is required to choose T_1 between $\lceil T_1^u/2 \rceil$ and T_1^u. This requirement will be proved shortly in the subsequent section. In Step (3), the algorithm assigns the largest harmonic periods to other tail tasks according to the extra harmonic constraints imposed on them. Finally, in Step (4), it performs GCD period assignment for non-tail tasks.

Period Assignment Algorithm

{

(1) Let $\langle p_1, p_2, \ldots, p_m \rangle$ be a sorted list of $\mathcal{P}_{tail}$ such that $T_1^u \leq T_2^u \leq \ldots \leq T_m^u$.

(2) Choose any T_1 such that $\lceil T_1^u/2 \rceil \leq T_1 \leq T_1^u$.

(3) $T_i = \lfloor \frac{T_i^u}{T_{i-1}} \rfloor \cdot T_{i-1}$ for $2 \leq i \leq m$.

(4) Perform GCD assignment for non-tail tasks.

}

4 Algorithm Analysis

In this section we make the worst case analysis of the proposed algorithm and summarize it in Theorem 2. It states that the algorithm, even in the worst case, yields a solution whose utilization does not exceed twice the optimal utilization.

Theorem 2. *Let U_{alg} be utilization computed by the algorithm, and U_{opt} the optimal utilization. U_{alg} is always less than $2U_{opt}$.*

Proof. Let $\langle p_1, p_2, \ldots, p_m \rangle$ be a sorted list of $\mathcal{P}_{tail}$. Suppose that $(T_1, \ldots, T_m)$ is a period assignment for tail tasks. Due to Theorem 1, the period of non-tail task in V can be represented with some combination of $\{T_1, \ldots, T_m\}$. Thus, utilization U can be generally written as follows.

$$U = \frac{\hat{e}_1}{T_1} + \frac{\hat{e}_2}{T_2} + \ldots + \frac{\hat{e}_m}{T_m} + \frac{\hat{e}_{m+1}}{GCD(T_1, T_2)} + \frac{\hat{e}_{m+2}}{GCD(T_2, T_3)} + \ldots + \frac{\hat{e}_{2^m-1}}{GCD(T_1, T_2, \ldots, T_m)} \tag{1}$$

where $\hat{e}_i = \sum_{p_j \in Q_i} e_j$ and Q_i is the set of tasks p_j which has a path to every tail task appearing in the i^{th} denominator, but to no other. Clearly, if Q_i is empty, $\hat{e}_i$ is zero.

For the analysis of the algorithm, the lower bound U_{low} of U is derived from Eq.(1) by replacing its denominators with maximum possible periods, as follows.

$$U_{low} = \frac{\hat{e}_1}{T_1^u} + \frac{\hat{e}_2}{T_2^u} + \ldots + \frac{\hat{e}_m}{T_m^u} + \frac{\hat{e}_{m+1}}{T_1^u} + \frac{\hat{e}_{m+2}}{T_2^u} + \ldots + \frac{\hat{e}_{2^m-1}}{T_1^u} \tag{2}$$

Since the proposed algorithm imposes the harmonicity constraints on $\mathcal{P}_{tail}$, $GCD(T_i, \ldots, T_j) = T_i$ for $1 \leq i < j \leq m$. Thus, utilization U_{alg} computed by the algorithm can be derived from Eq.(1), as follows.

$$U_{alg} = \frac{\hat{e}_1}{T_1} + \frac{\hat{e}_2}{T_2} + \ldots + \frac{\hat{e}_m}{T_m} + \frac{\hat{e}_{m+1}}{T_1} + \frac{\hat{e}_{m+2}}{T_2} + \ldots + \frac{\hat{e}_{2^m-1}}{T_1} \tag{3}$$

Now Eq.(2) and Eq.(3) are compared term by term. Let u_i^{alg} and u_i^{low} be the i^{th} terms in U_{alg} and U_{low}, respectively.

(a) *For $i = 1$:* $u_i^{alg} \leq 2u_i^{low}$, since T_1 is assigned a period no less than $T_1^u/2$ in the proposed algorithm.

(b) *For* $2 \leq i \leq 2^m - 1$ *and* $u_i^{low} \neq 0$:

$$\frac{u_i^{alg}}{u_i^{low}} = \frac{T_j^u}{\lfloor T_j^u / T_{j-1} \rfloor \cdot T_{j-1}}$$

for some $1 < j \leq m$. Since $\frac{x}{\lfloor x \rfloor} < 2$ for $x \geq 1$, $u_i^{alg} < 2u_i^{low}$.

The above comparison leads to $U_{alg} < 2U_{low}$. Let U_{opt} be the optimal utilization, then it is obvious that $U_{low} \leq U_{opt}$. Combining these, we have

$$U_{alg} < 2U_{opt}.$$

This proves the theorem. □

For a task graph with n sorted tasks (tail tasks are sorted in nondecreasing order of T_j^u and non-tail tasks are topologically sorted), $O(n)$ period assignment steps are required for the proposed algorithm. Note that the GCD assignment step in the algorithm is simplified through mathematical manipulation shown in Eq.(3). For a GCD assignment, $GCD(T_i, \ldots, T_j) = T_i$ can be used.

5 Empirical Performance

From the previous discussion, we know that the performance ratio of the proposed algorithm to the optimal one is less than 2.0. However, this result is derived via an analysis made for the worst case. In this section, we perform an empirical study to show that the performance of the algorithm is very close to the optimal one in most cases of practical interest.

To do so, we have implemented both the proposed algorithm and an optimal, exhaustive search algorithm. Then, we have generated artificial workloads using five types of representative task graphs that are commonly encountered in algorithmic structures of control applications. The task graph structures considered here are in-tree, out-tree, fork-join, Laplace equation solver [10], and FFT (fast Fourier transform) [2] types. Figure 5 pictorially shows these task graph types. For each of them, we generated three distinct task graphs by varying the number of tasks and their maximum period constraints. For the in-tree, fork-join, and Laplace equation solver task graphs that possess only one tail task, we have added two or more tail tasks to the task graphs to prevent the algorithms from generating trivial period assignments. Due to the inherent time complexity of the exhaustive search algorithm, the experiments were carried out with small-sized problems possessing 10 to 20 tasks.

To generate the workloads, we varied task execution times and maximum period constraints for each task graph type. For the maximum period constraints T_j^u for tail tasks, we used the normal probability distributions $N(\mu, \sigma^2)$ with mean μ and standard deviation σ. For each task graph type, three distinct test cases were generated using $N(600, 300^2)$, $N(500, 250^2)$ and $N(400, 200^2)$. Task execution times e_i were generated using $N(10, 5^2)$ in all test cases.

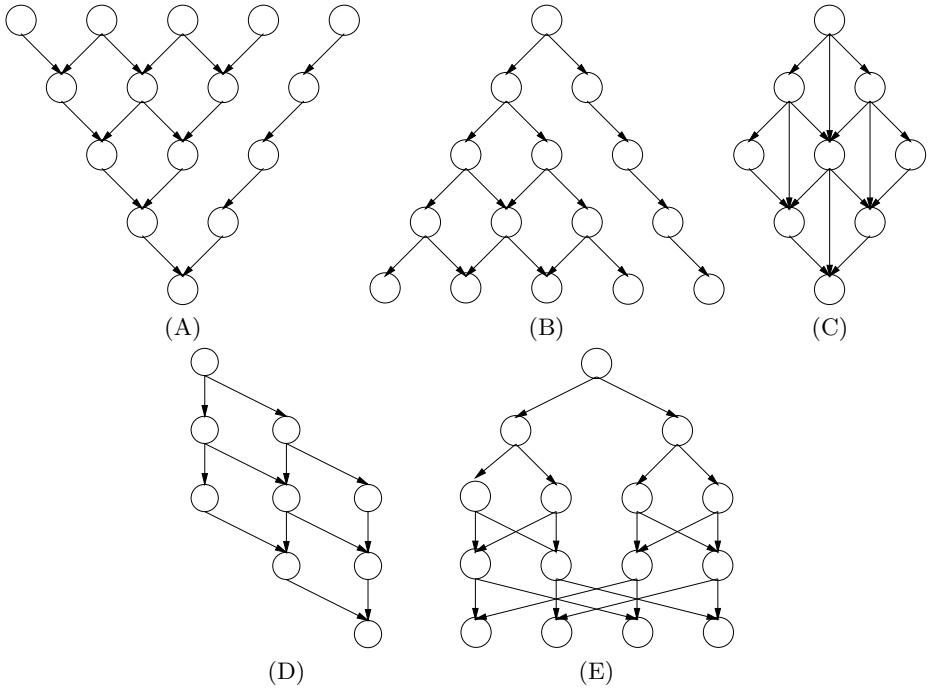

Fig. 3. Task graph types: (A) in-tree, (B) out-tree, (C) fork-join, (D) Laplace equation solver, and (E) FFT.

Figure 4 summarizes the results of our experiments. As is clear from Figure 4, the proposed algorithm yields solutions which are very close to the optimal ones in all cases. On the average, the performance ratio $\frac{U_{alg}}{U_{opt}}$ is 1.0330 in our experiments.

6 Conclusion

We have presented a period assignment algorithm which requires $O(n)$ period assignment steps and is capable of finding solutions close to the optimal ones. We have formally proved that the proposed algorithm has a performance ratio $\frac{U_{alg}}{U_{opt}} \leq 2$ and experimentally showed that it yields almost optimal solutions in practice. During the experiments, the performance ratio was $\frac{U_{alg}}{U_{opt}} = 1.0330$ on the average.

The proposed algorithm can be used as an essential component of the real-time system design methodology formulated in [3]. Since the methodology is currently applicable to a real-time system design built on top of a distributed platform, we are currently extending our algorithm for this purpose.

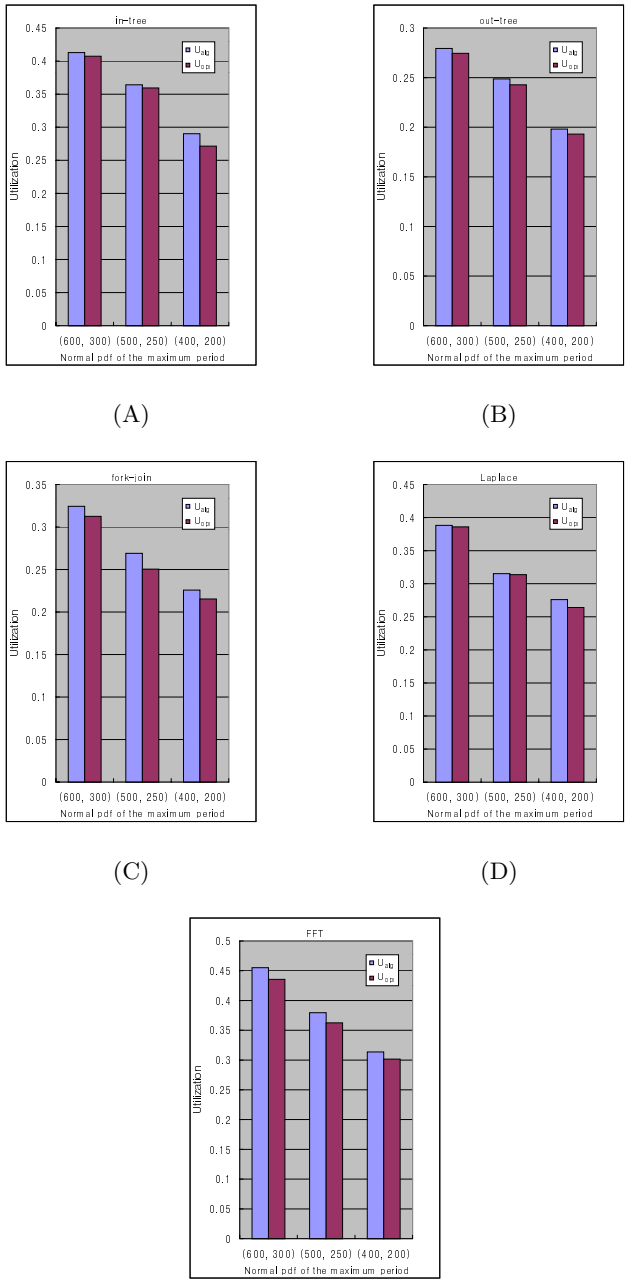

Fig. 4. Performance comparisons: (A) in-tree, (B) out-tree, (C) fork-join, (D) Laplace equation solver, and (E) FFT.

References

1. T. F. Abdelzaher, E. M. Atkins, and K. G. Shin. QoS negotiation in real-time systems and its application to automated flight control. In *Proceedings of IEEE Real-Time Technology and Applications*, pages 228–238, December 1997.
2. V. A. F. Almeida, I. M. Vasconcelos, J. N. C. Arabe, and D. A. Menasce. Using random task graphs to investigate the potential benefits of heterogeneity in parallel systems. In *Proceedings of Supercomputing*, pages 683–691, 1992.
3. R. Gerber, S. Hong, and M. Saksena. Guaranteeing real-time requirements with resource-based calibration of periodic processes. *IEEE Transactions on Software Engineering*, 21(7):579–592, July 1995.
4. K. Jeffay. The real-time producer/consumer paradigm: A paradigm for the construction of efficient, predictable real-time systems. In *ACM/SIGAPP Symposium on Applied Computing*, pages 796–804. ACM Press, February 1993.
5. J. Lehoczky, L. Sha and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 166–171, December 1989.
6. C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.
7. C. Park and A. Shaw. Experimenting with a program timing tool based on source-level timing schema. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 72–81, December 1990.
8. R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for QoS management. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 298–307, December 1997.
9. D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin. On task schedulability in real-time control systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 13–21, December 1996.
10. M. Y. Wu and D. D. Gajski. Hypertool: A programming aid for message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(3):330–343, July 1990.
11. J. Xu and D. Parnas. Scheduling processes with release times, deadlines, precedence and exclusion relations. *IEEE Transactions on Software Engineering*, 16(3):360–369, March 1990.

10.1007/b107031130003