Semi-partitioned Cyclic Executives for Mixed Criticality Systems

A. Burns
University of York, UK.
Email: alan.burns@york.ac.uk

S. Baruah
University of North Carolina, US.
Email: baruah@cs.unc.edu

***Abstract*—In a cyclic executive, a series of frames are executed in sequence; once the series is complete the sequence is repeated. Within each frame, units of computation are executed, again in sequence. In implementing cyclic executives upon multi-core platforms, there is advantage in coordinating the execution of the cores so that frames are released at the same time across all cores. For mixed criticality systems, the requirement for separation would additionally require that, at any time, code of the same criticality should be executing on all cores. In this paper we derive algorithms for constructing such multiprocessor cyclic executives for mixed-criticality collections of independent jobs.**

I. Introduction

Recent trends in embedded computing towards the widespread use of multi-core platforms, and the increasing tendency for applications to contain components of different criticality, have thrown up major challenges to the developers of safety-critical real-time systems. In this paper we consider these two challenges in the context of highly safety-critical application domains where cyclic executives remain the scheduling mechanism of choice.

Cyclic executives. A cyclic executive is a simple deterministic scheme that consists, for a single processor, of the continuous executing of a series of *frames* (or *minor cycles* as they are often called). Each frame consists of a sequence of *jobs* that execute in the specified sequence and are required to complete by the end of the frame. The set of frames is called the *major cycle*.

Multicore CPUs. On a multi-core, or multiprocessor, platform each core should have the same frame size and the same major cycle time. The time source from which the run-time support software will execute the jobs contained within each frame, is synchronised so that all cores switch between minor cycles concurrently. Within each frame there are a series of jobs to be executed. If jobs are constrained to execute always within the same minor cycle and always on the same core then the run-time schedule is defined to be *partitioned*. Alternatively, if jobs can migrate from one active frame to another active frame on a different core then the schedule is defined to be *global*. A *semi-partitioned* schedule has a small number of constrained migrations.

Mixed criticality. In mixed-criticality scheduling (MCS) theory, a single job may be characterized by several different WCET parameters denoting different estimates of the true WCET value, these different estimates being made at different levels of assurance. (The workload model used in this paper is formally defined in Section II.) The scheduling objective is then to validate the correct execution of each job at a level of assurance that is consistent with the criticality level assigned to that job: jobs assigned greater criticality must be shown to execute correctly when more conservative WCET estimates are assumed, while less critical jobs need to have their correctness demonstrated only when less conservative WCET estimates are assumed.

Related work. A cyclic executive is a particularly restricted form of static schedule. The issue of mapping mixed criticality code to static schedules has been addressed by Tamas-Selicean and Pop [10], [11]. An alternative approach to implementing the move between criticality levels in a static schedule is by switching between previously computed schedules; one per criticality level - this approach is explored in [2], [9]. However, these schemes are only applicable to single processor systems. The notion of separation used in this paper comes from [7].

In prior work [1], [5], we introduced the concept of implementing cyclic executives for mixed-criticality workloads upon multi-core CPUs. The workshop paper [1] formalized the problem, and proposed some initial approaches towards solving it for systems represented as collections of independent jobs. The scheduling test proposed was based upon a network flow argument and used a polynomial-time reduction. *In this paper we present a much more straightforward yet still optimal scheme.* See Def 1 for a definition of optimality.

II. System Model

The cyclic executive (CE) is defined by two durations, the length of the minor cycle (or frame) T_F and the duration of the major cycle T_M. These values are related by $(T_M = k.T_F)$ where k is a positive integer (usually a power of 2), denoting the number of frames in the repeating major cycle of the CE.

The issue of how to choose T_F and T_M to best support a set of tasks with given periods is beyond the scope of this paper. Rather we follow industrial practice [3] and assume these parameters are fixed by the system definition and that application tasks' periods are constrained to be multiples of T_F (up to the value of T_M).

The mapping of tasks to frames implies that there is a set of jobs allocated to each frame. All jobs within a frame must complete by the end of the frame. However, what it means to complete will depend on the behaviour of the system in terms of its criticality levels – as will be explained shortly.

We assume that the hardware platform consists of m identical (unit speed) processors (or cores). Each job can execute on any core and has identical temporal behaviour on all cores.

In general we assume there are V criticality levels, L_1 to L_V, with L_1 being the highest criticality. Each job j_i is assigned a criticality level, denoted χ_i, and two WCET parameters. One represents its estimated execution time at its own criticality level ($C_i(\chi_i)$) and the other an estimate at the base (i.e., lowest) criticality level ($C_i(L_V)$). It follows that if a job is of the lowest criticality level (i.e., $\chi_i = L_V$) then it only has one WCET parameter. For all other jobs, $C(\chi_i) \geq C(L_V)$. The rationale for having more than one WCET parameter is covered in a number of papers on mixed criticality systems, including the initial work of Vestal [12].

This use of only two C_i values for V criticality levels is a more constrained model than the one proposed by Vestal [12], under which each criticality level may give rise to a distinct WCET estimate. However with say five criticality levels it is unlikely that five distinct estimates of the worst-case execution time of the task would be available, while it can be argued ([6], [4]) that the restriction to just two estimates is sufficient to capture the key properties of a mixed criticality system.

At run-time the system is defined to be executing in one of V *modes*. In mode L_V (the lowest-criticality mode) all deadlines of all jobs must be met. It represents 'normal' behaviour. If every job j_i executes for no more than $C_i(L_V)$ then all deadlines must be guaranteed. If some job j_i executes for more than $C_i(L_V)$ then the mode of the system will degrade towards L_1, with jobs of criticality lower than χ_i no longer guaranteed. This mode change behaviour is explained in more detail later in the paper.

Run-time support

Mixed-criticality scheduling (MCS) theory has primarily concerned itself with the sharing of CPU computing capacity in order to satisfy the computational demand, as characterized by the worst-case execution times (WCET), of pieces of code. However, there are typically many additional resources that are also accessed in a shared manner upon a computing platform, and it is imperative that these resources also be considered. An interesting approach towards such a consideration was advocated by Giannopoulou et al. [7] in the context of multicore platforms: during any given instant in time, all the cores are only allowed to execute code of the same criticality level. This approach has the advantage of ensuring that accesses to all shared resources (memory buses, cache, etc.) during any time-instant are only from code of the same criticality level. We refer to such a scheme of switching between workloads of different criticality levels as *synchronised switching*. We focus our attention in this paper on synchronized switching. That is, we seek to construct cyclic executives in which each minor cycle may be considered partitioned into V criticality levels. Initially the highest criticality jobs are executed, when they have finished the next highest criticality jobs are executed, and so on. This continues until finally the lowest criticality jobs are executed. In a simple system with just two criticality levels, HI and LO, there is a switchover time S defined within each minor frame. Before S each core is executing HI-criticality work, after S each core is executing LO-criticality work. To give resilient fault tolerant behaviour, if the HI-criticality work has not completed by time-instant S on any core then the LO-criticality work is postponded (on every core), thereby giving extra time for the HI-criticality work to execute (up to the end of the minor cycle). In this paper we will explore how to find acceptable (safe and efficient) values for the switching times.

Implementing the criticality switches. Giannopoulou et al. [7] advocated, if supported by the hardware platform, the use of synchronisation barriers. In the case of dual-criticality workloads (the generalization to > 2 criticality levels is straight-forward), each core calls the barrier upon completing its assigned HI-criticality work. When the final core completes and calls the barrier, all the calls are released from the barrier and each core continues with executing LO-criticality work.

The benefit of this barrier-based scheme is that it can take advantage of time gained by jobs executing for less than their estimated WCETs. So at the end of the HI-criticality executions if the signal occurs before the pre-computed barrier S, then all cores can move to LO-criticality executions early. Additionally, there may be situations arising at run-time when a late switch to one criticality level is compensated by time gained from under-execution within jobs of the next criticality level. For example, the switch occurs at some time $> S$, but the LO-criticality jobs end up executing for less than their $C_i(\text{LO})$ WCET values and hence all complete by the end of the frame.

III. Dual Criticality Jobs

In this section, we consider the scheduling of a collection of jobs within a single frame of an m-processor platform, when there are only two criticality levels ($V \equiv 2$). All the jobs are assumed to become available at the start of the frame (without loss of generality, denoted as being at time 0), and they all have a deadline at the end of the frame (denoted D). In keeping with prior work on the scheduling of such dual-criticality systems, we use the notation HI and LO to denote the greater and lesser criticality levels (i.e., $L_1 \equiv \text{HI}$ and $L_V \equiv L_2 \equiv \text{LO}$). The criticality of job j_i is denoted by $\chi_i \in \{\text{LO}, \text{HI}\}$; each LO-criticality job j_i is characterized by a single WCET parameter $C_i(\text{LO})$, while each HI-criticality job is characterized by two WCET parameters $C_i(\text{LO})$ and $C_i(\text{HI})$.

Given a collection of such dual-criticality jobs to be scheduled within a frame of duration D upon an m-processor platform, our objective is to determine the switching point S such that only HI-criticality jobs are executed over the interval $[0, S)$. If all HI-criticality jobs complete by time-instant S, then LO-criticality jobs are executed over $[S, D)$; else, the LO-criticality jobs are abandoned and execution of HI-criticality jobs continues over $[S, D)$ as well. It follows that there are three conditions that need to be satisfied:

1) If each HI-criticality job j_i executes for no more than $C_i(\text{LO})$, then all the HI-criticality jobs must fit into the interval $[0, S)$.

2) All the LO-criticality jobs must fit into the interval $[S, D)$
3) If each HI-criticality job j_i executes for no more than $C_i(\text{HI})$, then all the HI-criticality jobs must fit into the interval $[0, D)$.

In Section III-A below, we derive a simple and efficient algorithm for determining S (and the corresponding schedules) such that these conditions are satisfied; in Section III-B, we describe an optimization to this simple method. These algorithms assume minimal run-time support; if additional run-time support is available, then a further optimization is possible – this is described in Section III-C.

A. A simple scheme for constructing CEs

We first define two (potential) candidates for the switching point S:

$S^{\min}$ The earliest instant at which all HI-criticality jobs have completed their LO-criticality work.

$S^{\max}$ The latest instant at which a switch must occur for the LO-criticality work to complete by time D.

It is evident that any candidate S must satisfy the two inequalities $S^{\min} \leq S \leq S^{\max}$.

Let us additionally define two interval durations, which constrain the possible values of $S^{\min}$ and $S^{\max}$.

Δ^{LO} The duration (makespan) of the interval needed for all the LO-criticality jobs to (begin and) complete execution.

Δ^{HI} The duration of the interval needed for all the HI-criticality jobs to execute the extra work they must do in HI-criticality mode — i.e., the amount $(C_i(\text{HI}) - C_i(\text{LO}))$, for each j_i with $\chi_i = \text{HI}$.

To determine these durations, we employ the optimal scheme of McNaughton [8, page 6]. Given a collection of n jobs with execution requirements $c_1, c_2, \ldots, c_n$, McNaughton showed that the minimum makespan of a preemptive schedule for these jobs on m unit-speed processors is given by

$$\max\left(\frac{\sum_{i=1}^{n} c_i}{m}, \max_{i=1}^{n}\{c_i\}\right) \tag{1}$$

The actual schedule is obtained by taking the jobs (in any order) and allocating them to m intervals of the size of the makespan, each representing one of the m processors. As one interval is filled, perhaps with part of a job, the next interval starts with the rest of this job. At most $(m-1)$ jobs are split across intervals in this manner. During run-time a job that was split across two intervals will run at the beginning of the time-interval upon one processor, and towards the end of the time-interval on the other processor.

A direct application of McNaughton's result yields the conclusion that the minimum makespan for a global preemptive schedule for the jobs in LO-criticality mode is given by

$$\Delta^{\text{LO}} \stackrel{\text{def}}{=} \max\left(\frac{\sum_{\chi_i=\text{LO}} C_i(\text{LO})}{m}, \max_{\chi_i=\text{LO}}\left\{C_i(\text{LO})\right\}\right) \tag{2}$$

We therefore set

$$S^{\max} \stackrel{\text{def}}{=} D - \Delta^{\text{LO}} \tag{3}$$

Similarly, a direct application of the makespan result allows the minimum interval for the HI-criticality work (in LO-criticality mode) to be computed:

$$S^{\min} \stackrel{\text{def}}{=} \max\left(\frac{\sum_{\chi_i=\text{HI}} C_i(\text{LO})}{m}, \max_{\chi_i=\text{HI}}\left\{C_i(\text{LO})\right\}\right) \tag{4}$$

Clearly for the whole system to be schedulable, it is necessary that $S^{\min} \leq S^{\max}$ which is equivalent to requiring that

$$\begin{aligned} & S^{\min} \leq D - \Delta^{\text{LO}} \\ \Leftrightarrow \quad & S^{\min} + \Delta^{\text{LO}} \leq D \end{aligned} \tag{5}$$

We now consider the final constraint — the scheduling of HI-criticality jobs executing in HI-criticality mode. It has been shown [1, Example 1] that this is not necessarily ensured by simply computing the makespan (using McNaughton's method, as above) with the $C_i(\text{HI})$ values, and validating that the resulting makespan is $\leq D$. We instead determine the minimal makespan for all the HI-criticality jobs, subject to each such job having received an amount of execution equal to its LO-criticality WCET by time-instant $S^{\min}$. To determine this makespan, we apply McNaughton's scheme to the work that is left to do after time-instant $S^{\min}$ (i.e. $C_i(\text{HI}) - C_i(\text{LO})$ for each job j_i with $\chi_i = \text{HI}$). Letting $C_i(\text{EX})$ denote the "excess" computational requirement of job j_i in HI-criticality mode over LO-criticality mode:

$$C_i(\text{EX}) \stackrel{\text{def}}{=} \big(C_i(\text{HI}) - C_i(\text{LO})\big),$$

we have

$$\Delta^{\text{HI}} \stackrel{\text{def}}{=} \max\left(\frac{\sum_{\chi_i=\text{HI}} C_i(\text{EX})}{m}, \max_{\chi_i=\text{HI}}\left\{C_i(\text{EX})\right\}\right) \tag{6}$$

It is evident that $S^{\min} + \Delta^{\text{HI}} \leq D$ is sufficient for schedulability; earlier (Expression 5) we had shown that $S^{\min} + \Delta^{\text{LO}}$ should also be $\leq D$. Putting these pieces together, we may summarize this method as follows. We compute $S^{\min}, \Delta^{\text{LO}}$, and Δ^{HI} according to Expressions (4), (2), and (6) respectively, and require that

$$S^{\min} + \max\big(\Delta^{\text{LO}}, \Delta^{\text{HI}}\big) \leq D \tag{7}$$

as a sufficient schedulability condition. If this condition is satisfied, $S \leftarrow S^{\min}$ (i.e., we declare $S^{\min}$ to be the switch-point we had set out to compute).

B. An improvement

Let us now suppose that Condition 7 is violated, and $S^{\min} + \max\big(\Delta^{\text{LO}}, \Delta^{\text{HI}}\big) > D$. Since $\big(S^{\min} + \Delta^{\text{LO}} \leq D\big)$ is a necessary condition for schedulability (see Inequality 5), it must be the case that

$$S^{\min} + \Delta^{\text{HI}} > D.$$

Now if $\big(\sum_{\chi_i=\text{HI}} C_i(\text{HI}) \geq mD\big)$, there is nothing to be done. Otherwise, there must be some unused processor capacity in the McNaughton schedule constructed according to Expression 4 for the interval $[0, S)$, and/or in the McNaughton

schedule constructed according to Expression 6 for the interval after time-instant S. Let us consider the situation where the schedule has some unused processor capacity over the interval $[0, S)$ (recall that $S \leftarrow S^{\min}$ in the method of Section III-A). An inspection of Expression (4) reveals that this happens if

$$\frac{\sum_{\chi_i=\text{HI}} C_i(\text{LO})}{m} < \max_{\chi_i=\text{HI}} \{C_i(\text{LO})\}$$

Our idea, intuitively speaking, is that any such unused capacity prior to time-instant S may as well be allocated to some HI-criticality task, for use in the event of the system undergoing a mode-change into HI-criticality mode. (If the system does not undergo such a mode-change, this allocated capacity may end up remaining unused.) Doing so leaves less execution remaining to be completed after the switch instant S in HI-criticality mode, and may thus result in a smaller makespan in HI-criticality modes (i.e., a smaller value for Δ^{HI}).

Such a scheme is particularly effective if the duration of the HI-criticality schedule after S — the one of duration Δ^{HI} — is also dominated by longer jobs, i.e., if in Expression 6

$$\frac{\sum_{\chi_i=\text{HI}} C_i(\text{EX})}{m} < \max_{\chi_i=\text{HI}} \{C_i(\text{EX})\}$$

If this be the case, then the unused capacity prior to time-instant S can be filled so as to minimise the maximum $C_i(\text{EX})$ by bringing forward work to before S — this is accomplished by increasing $C_i(\text{LO})$ for such a job and decreasing its $C_i(\text{EX})$ by the same amount. However, jobs that have $(C_i(\text{LO}) = S)$ cannot have work brought forward in this manner since this would result in S increasing as well.

It is evident that this scheme is effective since:

- Any work brought forward will not change S,
- The first term in Expression (6) is not increased by bringing work forward, and
- The second term in Expression (6) is reduced by always choosing the largest value and decreasing it.

We note that if more than one job has the same $C_i(\text{EX})$ value then an arbitrary choice is made (and has no impact on optimality).

And what if there is no unused processor capacity in the schedule over $[0, S)$? In that case, the switch-point S may be increased to any value $\leq S^{\max}$ (where $S^{\max}$ is as defined by Expression (3)). An obvious choice for S is $S \leftarrow S^{\max}$; an algorithm for achieving the smallest value of S (i.e., the earliest possible switch-time) is as follows. Setting the switch point S to be $S^{\min} + 1$ will generate m free slots. So $C_i(\text{LO})$ values of HI-criticality jobs can be increased by this amount (and the corresponding $C(\text{EX})$ values decreased). If this will reduce the size of Δ^{HI} by more than one then an overall decrease in $S + \Delta^{\text{HI}}$ will have been achieved . This cycle is repeated (i.e. adding 1 to S) until either no further gain is made or S takes the value of $S^{\max}$. At each step of the cycle no $C(\text{LO})$ value should increase beyond the current value of S.

	χ_i	$C_i(\text{LO})$	$C_i(\text{HI})$	$C_i(\text{HI}) - C_i(\text{LO})$
j_1	LO	3	-	-
j_2	LO	2	-	-
j_3	LO	2	-	-
j_4	HI	2	7	5
j_5	HI	3	7	4
j_6	HI	3	3	0
j_7	HI	4	4	0

TABLE I
AN EXAMPLE DUAL-CRITICALITY JOB INSTANCE

Example 1: To illustrate the above scheme consider the scheduling of the mixed-criticality instance of Table I upon 3 unit-speed processors with a frame length of 8 ($D = 8$).

We can immediately use the equations above to compute: $\Delta^{\text{LO}} = 3$ (and hence $S^{\max} = 5$) and $S^{\min} = 4$. So the first step to schedulability is satisfied (i.e. $S^{\min} \leq S^{\max}$). We note that if we ignore mixed criticality issues then the minimum makepsan for the HI-criticality jobs (ignoring LO-criticality work) is 7. So a completely separated scheme would require a frame size of 10 $(7 + 3)$.

If we initially focus on $S^{\min}$ then we note that there are no free slots, so equation(6) gives a makespan in HI-criticality mode (Δ^{HI}) of 5. So the use of this value for S (i.e. 4) gives a required frame size of 9 (4+5); since the frame-size is 8, the instance would be deemed unschedulable with $S \leftarrow 4$.

However, if we set $S \leftarrow (S^{\min}+1)$ which equals $S^{\max} = 5$ then the total work available on three processors by time 5 is 15. The work required using $C(\text{LO})$ values for HI-criticality work is 12. Hence 3 units of work can be added to these $C(\text{LO})$ values. If we make $C_4(\text{LO}) = 4$ and $C_5(\text{LO}) = 4$ then maximum $C_i(\text{EX})$ becomes equal to 3. Hence $\Delta^{\text{HI}} = 3$ and $S^{\max} + \Delta^{\text{HI}} = 8$. Therefore the job set fits into the frame size of 8, with a switch time of 5. ■

C. More flexible implementations

The cyclic executives constructed as discussed above are implementable as lookup tables. Three lookup tables are constructed as dictated by the McNaughton procedure: one for the interval $[0, S)$, another for HI-criticality jobs over the interval $[S, D)$, and a third for LO-criticality jobs over the interval $[S, D)$. The first lookup table is always executed, while one of the other two is selected depending upon whether all HI-criticality jobs have completed or not by time-instant S.

Lookup tables are a very restrictive form of run-time dispatching. If a certain amount of additional *flexibility* is permitted, then more efficient use of platform resources may be possible, We illustrate with an example.

Suppose that $C_1(\text{LO})$ in the example instance of Table I were equal to 4 (rather than 3 as listed in Table I). It may be verified that Δ^{LO} for this instance is then equal to 4; the switch-point must therefore be $\leq (8 - 4)$ or 4. But we saw in Example 1 that this is not possible, since setting $S \leftarrow 4$ results in a makespan of $(4 + 5 =)$ 9 in HI-criticality mode.

Let us therefore choose $S \leftarrow 5$ as mandated by the arguments in Example 1, and consider the CE schedule specified in Figure 1 over the interval $[0, 5)$. Notice that this schedule is

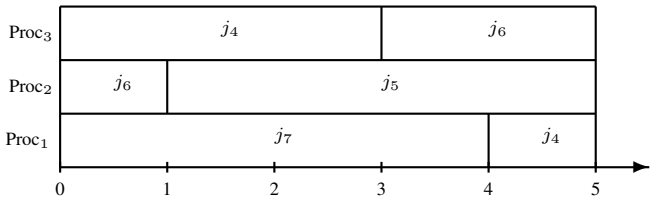

Fig. 1. Dynamic switching.

compliant with the requirements of Example 1: j_4 and j_5 both execute for 4 units over $[0, 5)$ while j_6 and j_7 each execute for their $C_i(\text{LO})$ values of 3 and 4 respectively. Hence in HI-criticality mode all HI-criticality jobs would complete by the end of the frame, at time-instant 8.

Now observe that in any LO-criticality behaviour,

- j_4 would complete $C_4(\text{LO}) = 2$ units of execution by time-instant 2; hence, the execution of j_6 on processor 3 could be moved forward[1] to the interval $[2, 4)$.
- As a consequence, j_6 would complete its $C_6(\text{LO}) = 3$ units of execution by time-instant 4.
- j_5 would complete its $C_5(\text{LO}) = 3$ units of execution over $[1, 4)$, also completing by time-instant 4, and
- j_7 would execute to completion over $[0, 4)$.

Thus, all the HI-criticality jobs processors will have completed their LO-criticality execution by time-instant 4, and the platform becomes available for the LO-criticality jobs to execute at time-instant 4 and complete by time-instant 8.

This example illustrates that the added run-time flexibility of adjusting the pre-computed schedule may permit enhanced schedulability — instances not schedulable without this flexibility can be scheduled correctly. We are currently working on better understanding what kinds of run-time flexibility are reasonable to permit within the context of cyclic executives; we leave as future work the design of algorithms that would construct schedules such as the one shown in Figure 1.

IV. Jobs with More than Two Criticality Levels

We now assume there are $V > 2$ criticality levels, L_1 (the highest) to L_V (the lowest). Recall from Section II that each job j_i, of criticality χ_i, has just has two WCET estimates, one for the base criticality level L_V, $C_i(\text{normal}) = C_i(L_V)$, abbreviated to $C_i(\text{NL})$, and one for its own criticality level, $C_i(\text{self}) = C_i(\chi_i)$, abbreviated to $C_i(\text{SF})$. We define $C_i(\text{EX}) \stackrel{\text{def}}{=} C_i(\text{SF}) - C_i(\text{NL})$. We overload the symbols L_i to also denote the set of jobs of that criticality. We seek to compute $(V-1)$ switch points S^1 to S^{V-1} that are constrained as follows (for notational convenience we let S^0 and S^V denote the start and end of the frame respectively (i.e $S^0 \equiv 0$ and $S^V \equiv D$)). So for each criticality level L_i, and frame f we require that

- If each job $j_i \in L_i$ executes for no more than $C_i(\text{NL})$, then all the jobs in the set L_i must fit into the interval $(S^{i-1}, S^i]$
- If each job $j_i \in L_i$ executes for no more than $C_i(\text{SF})$, then all the jobs in the set L_i must fit into the interval $(S^{i-1}, S^V]$.

To compute the switching times we extend the process defined in Section III above to > 2 criticality levels. It is possible to start at the lowest or highest criticality level; experimentation shows that it is better to start at the highest. So first we compute minimum makespan for criticality level L_1:

$$S_1^{\min} \stackrel{\text{def}}{=} \max\left(\frac{\sum_{j_i \in L_1} C_i(\text{NL})}{m}, \max_{j_i \in L_1}\left\{C(\text{NL})\right\}\right) \tag{8}$$

Next we compute Δ^1 and check that $S_1^{\min} + \Delta^1$ is no greater than S^V $(= D)$:

$$\Delta^1 \stackrel{\text{def}}{=} \max\left(\frac{\sum_{j_i \in L_1} C_i(\text{EX})}{m}, \max_{j_i \in L_1}\left\{C_i(\text{EX})\right\}\right) \tag{9}$$

If $S_1^{\min} + \Delta^1 > S^V$ then work must be brought forward so that $S_1^{\min}$ is increased but Δ^1 is decreased by a greater amount. This is achieved by adding to $C(\text{NL})$ so as to minimise the maximum $C(\text{EX})$ (for jobs of criticality L_1). If such alterations cannot deliver $S_1^{\min} + \Delta^1 \leq S^V$ then the job set is unschedulable. Alternatively, S^1 is fixed to be the minimum value computed.

This process is repeated for each criticality level, L_i using:

$$S_i^{\min} \stackrel{\text{def}}{=} \max\left(\frac{\sum_{j_i \in L_i} C_i(\text{NL})}{m}, \max_{j_i \in L_i}\left\{C_i(\text{NL})\right\}\right) \tag{10}$$

and

$$\Delta^i \stackrel{\text{def}}{=} \max\left(\frac{\sum_{j_i \in L_i} C_i(\text{EX})}{m}, \max_{j_i \in L_i}\left\{C_i(\text{EX})\right\}\right) \tag{11}$$

with the conditions

$$S^{i-1} + S_i^{\min} + \Delta^i \leq S^V \tag{12}$$

and for all jobs of criticality L_i

$$C_i(\text{NL}) \leq S_i^{\min}. \tag{13}$$

At all stages, modification to $C_i(\text{NL})$ (and hence $C_i(\text{EX})$) are made to ensure these two conditions are met. Note that some movement of computation time may be possible without increasing a $S_i^{\min}$ value. Each step fixed S^i.

A. An Example

We illustrate the above scheme upon an example with two cores, four criticality levels and three jobs per criticality level. Table II lists the parameters for the jobs. The frame length is 20 units. We note that independent makespans for the four criticality levels would require a frame length of 48 (20+15.5+8.5+4).

First $S_1^{\min}$ and Δ^1 are computed; they are seen to equal 3 and 18 respectively. Together this is too large (as the frame size is 20). So S^1 is set to 4 (i.e. $C_1(\text{NL}) = 4$ with $C_1(\text{EX}) = 16$).

[1]Note that this moving forward of j_6's execution is not permitted in a pure lookup table dispatcher; this is the additional implementation flexibility that is sought in this section.

	χ_i	$C_i(\text{NL})$	$C(\text{SF})$	$C_i(\text{EX})$
j_1	L_1	2	20	18
j_2	L_1	1	8	7
j_3	L_1	3	9	6
j_4	L_2	6	13	7
j_5	L_2	1	3	2
j_6	L_2	4	15	11
j_7	L_3	5	6	1
j_8	L_3	3	8	5
j_9	L_3	1	3	2
j_{10}	L_4	3	3	0
j_{11}	L_4	4	4	0
j_{12}	L_4	1	1	0

TABLE II
AN EXAMPLE MIXED-CRITICALITY JOB SET.

	χ_i	$C_i(\text{NL})$	$C(\text{SF})$	$C_i(\text{EX})$
j_1	L_1	4	20	16
j_2	L_1	1	8	7
j_3	L_1	3	9	6
j_4	L_2	6	13	7
j_5	L_2	1	3	2
j_6	L_2	7	15	8
j_7	L_3	5	6	1
j_8	L_3	4	8	4
j_9	L_3	1	3	2
j_{10}	L_4	3	3	0
j_{11}	L_4	4	4	0
j_{12}	L_4	1	1	0

TABLE III
THE EXAMPLE JOB SET OF TABLE II TRANSFORMED.

This now delivers $S_1^{\min} = 4$ and $\Delta^1 = 16$ which is sufficient for criticality level L_1.

Next level L_2 is checked. Note the frame size for this criticality level is, in effect, 16 (i.e. 20 - S^1). So, $S_2^{\min} = 6$ and $\Delta^2 = 11$; again this is too long so $S_2^{\min}$ is set to 7, with the result that $C_6(\text{NL})$ is made equal to 7 and $C_6(\text{EX})$ is 8. As a result Δ^2 is now equal to 8.5, and the sum of the two intervals is 15.5 which is sufficient. This fixes S^2 to be 11 (4+7).

Continuing with L_3. Frame size is now 9. Value of $S_3^{\min}$ is 5, and Δ^3 is 5 also. As $10 > 9$ there is again a need to reduce Δ^3. Here this can be done without increasing $S_3^{\min}$ (as this interval was not 'full'). Let $C_8(\text{NL}) = 4$ and hence $C_8(\text{EX}) = 4$. Now $\Delta^3 = 4$ and $S_3^{\min} + \Delta^3 = 9$ (5+4). Again this is sufficient and S^3 is set equal to 16.

The final step is to check that the lowest criticality jobs will fit into the interval left for them. The interval is of length 4, and the makespan (Δ^4) for this set is 4. So they are accommodated and the job set can be declared schedulable.

To further illustrate the process of modifying the job set to obtain a schedulable one, Table III gives the parameters of the job set obtained after modification. It is easy to observe that the new job set is obtained from the initial one by just adding to the $C(\text{NL})$ estimates. And also it is clear that the new job set is schedulable with switch points 4, 11 and 16.

Optimality. The scheme outlined at the beginning of this section via equations/conditions (10) to (13), and illustrated with the example above, is optimal in the following sense.

Definition 1: An allocation scheme (of jobs to frames) is *optimal* if it leads to the smallest possible switching points and a schedulable system.

This notion of optimal is intuitive as for each criticality level the earliest switching point maximises the time available for the lower criticality levels. The scheme produces the optimal value for each switching point, S_i, as:

- If $S_i = S_i^{\min}$ satisfies condition (12) then this is the minimum makespan by definition [8, page 6].
- If condition (12) is not satisfied the scheme increases $C(\text{NL})$ values by the minimum amount commensurate with decreasing the maximum $C(\text{SF})$ so that the condition is met. This leads to a minimum increase in S_i.
- If no S_i can be found (i.e. it continues to increase until $\Delta^i = 0$ without satisfying condition (12) then the system is unschedulable.

V. CONCLUSIONS

Single processor safety-critical systems are often constrained so that they can be implemented as a series of frames in a repeating cyclic executive. In this paper we have extended this approach to incorporate multi-core platforms and mixed criticality applications. We allow a minimum number of jobs to be split across the frames, and propose a practical means of constructing the necessary cyclic schedule. Future work will extend our approach to multi-cycle systems.

Acknowledgements. This research is partially supported by NSF grants CNS 1115284, CNS 1218693, CNS 1409175, and CPS 1446631, AFOSR grant FA9550-14-1-0161, ARO grant W911NF-14-1-0499, and a grant from General Motors Corp. It is also supported by ESPRC grant MCC (EP/K011626/1). No new primary data were created during this study.

REFERENCES

[1] S. Baruah and A. Burns. Achieving temporal isolation in multiprocessor mixed-criticality systems. In *Proc. of the 2nd Workshop on Mixed Criticality Systems (WMC)*, 2014.

[2] S. Baruah and G. Fohler. Certification-cognizant time-triggered scheduling of mixed-criticality systems. In *Proc. of the Real-Time Systems Symposium (RTSS)*, 2011.

[3] I. Bate and A. Burns. An integrated approach to scheduling in safety-critical embedded control systems. *Real-Time Systems*, 25(1):5–37, 2003.

[4] A. Burns. An augmented model for mixed criticality. In Sanjoy K. Baruah, Liliana Cucu-Grosjean, Robert I. Davis, and Claire Maiza, editors, *Mixed Criticality on Multicore/Manycore Platforms (Dagstuhl Seminar 15121)*, volume 5(3). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2015.

[5] A. Burns, T. Fleming, and S. Baruah. Cyclic executives, multi-core platforms and mixed criticality applications. In *Proc. ECRTS*, 2015.

[6] D. de Niz, K. Lakshmanan, and R. Rajkumar. On the scheduling of mixed-criticality real-time task sets. In *Proc. of the Real-Time Systems Symposium*, pages 291–300, 2009.

[7] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. Scheduling of mixed-criticality applications on resource-sharing multicore systems. In *International Conference on Embedded Software (EMSOFT)*, pages 17:1–17:15, 2013.

[8] R McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6:1–12, 1959.

[9] D. Socci, P. Poplavko, S. Bensalem, and M. Bozga. Time-triggered mixed critical scheduler. In *Proc. of the Workshop on Mixed Criticality Systems (WMC)*, pages 67–72, 2013.

[10] D. Tamas-Selicean and P. Pop. Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures. In *Proc. of the Real-Time Systems Symposium (RTSS)*, 2011.

[11] D. Tamas-Selicean and P. Pop. Task mapping and partition allocation for mixed-criticality real-time systems. In *Dependable Computing (PRDC), 17th Pacific Rim International Symposium on*, pages 282–283, 2011.

[12] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proc. of the Real-Time Systems Symposium*, pages 239–243, 2007.