Minimizing Multi-Resource Energy for Real-Time Systems with Discrete Operation Modes

Fanxin Kong[†] Yiqun Wang[†] Qingxu Deng[†] Wang Yi[†,‡]
[†] Dept. of Computer Science & Technology, Northeastern University, China
[‡] Dept. of Information Technology, Uppsala University, Sweden
kongfx@ise.neu.edu.cn, dengqx@mail.neu.edu.cn, yi@it.uu.se

***Abstract*—Energy conservation is an important issue in the design of embedded systems. Dynamic Voltage Scaling (DVS) and Dynamic Power Management (DPM) are two widely used techniques for saving energy in such systems. In this paper, we address the problem of minimizing multi-resource energy consumption concerning both CPU and devices. A system is assumed to contain a fixed number of real-time tasks scheduled to run on a DVS-enabled processor, and a fixed number of off-chip devices used by the tasks during their executions. We will study the non-trivial time and energy overhead of device state transitions between active and sleep states. Our goal is to find optimal schedules providing not only the execution order and CPU frequencies of tasks, but also the time points for device state transitions. We adopt the frame-based real-time task model, and develop optimization algorithms based on 0-1 Integer Non-Linear Programming (0-1 INLP) for different system configurations. Simulation results indicate that our approach can significantly outperform existing techniques in terms of energy savings.**

***Keywords*-real-time systems; energy management; dynamic voltage scaling; dynamic power management;**

I. Introduction and Background

Energy efficiency plays an important role in modern embedded systems. Two effective and widely-used energy saving techniques are *Dynamic Voltage Scaling* (DVS) and *Dynamic Power Management* (DPM). DVS is used to reduce CPU power consumption by reducing CPU voltage and frequency, while DPM is used to reduce power consumption of off-chip devices by transitioning a device from the active to the sleep state.

A DVS-enabled processor can adjust its operating voltage and frequency (speed) dynamically to reduce power consumption. DVS is supported by many commercial CPUs, such as Intel XScale and PowerPC 405. Each processor offers a fixed, limited set of valid operational frequencies. There are two types of DVS policies: the CPU frequency can change within a single task for *intra-task* DVS, while the CPU frequency can only be changed at inter-task boundaries for *inter-task* DVS. We adopt the inter-task DVS policy in this paper. In a real-time system, the goal is to minimize energy consumption while still meeting all the deadlines of the real-time tasks. [1] provides a comprehensive survey of research work on the real-time DVS problem.

As the number of off-chip devices in today's embedded systems grows, energy consumption of these devices becomes an important issue in addition to the CPU energy consumption. A typical device has at least two states: a high-power *active* state, and a low-power *sleep* state. DPM is a popular technique for reducing energy consumption of an off-chip device transitioning it to the sleep state when not in use. Since each state transition between active and sleep incurs a certain overhead in terms of time and energy consumption, a device should transition to the sleep state only when the idle duration exceeds a certain threshold, called the *break-even time*. [2], [3] explored DPM via I/O device scheduling for energy efficiency in real-time embedded systems.

Many research works had been focused on the energy saving for either the CPU or the device separately. However, it was shown that lowering the CPU frequency using DVS, even though it reduces CPU energy consumption, may cause a net increase in total system-wide energy consumption [4], [5]. Lower CPU frequency results in decreased processor energy consumption at the expense of increased task execution time. This lengthens the duration that a device must stay in the active state and shortens the idle duration. Furthermore, some devices may not be able to go to sleep if the device idle duration does not exceed its break-even time. The effect is that device energy consumption is increased. Higher CPU frequency generates the opposite effect. Therefore, the energy consumption of both processor and devices should be considered together. In [6]–[8], the concept of *critical speed* (or energy-efficient speed) is defined as the CPU frequency that results in the minimum system-wide energy consumption. In [9], the authors integrated DVS and DPM into the same framework and proposed a practical system-wide energy-efficient scheduling algorithm for preemptive periodic real-time tasks.

The authors, in [10], exactly analyzed the trade-off between the benefit/cost spaces of DVS and DPM techniques and addressed the problem of minimizing system-wide energy consumption with a combination of DVS and DPM while taking into account device state transition overheads for the *frame-based* task model [11]. Two key assumptions in [10] are that all devices used by all tasks must be in active state throughout the execution of the entire task sequence in a frame, i.e., all devices are assumed to be used by all active tasks and devices can go to the sleep state only when all tasks have finished execution, and they assume the processor can adjust the processing frequency continuously and power function is $P(f) = \alpha f^3$. In this paper, we relax these two assumptions. First, we allow each task to use a subset of devices, so that certain devices may go to the sleep state even while certain tasks are still running, as long as the devices are not used by the running tasks. Second, we focus on the realistic case that a processor can only operate at discrete set of valid operational modes (frequencies) and do not make any restriction on the form of power function. The optimization problem has two degrees of freedom: *task execution order* within each frame, and *CPU frequency assignment* to each task assuming inter-task DVS. Since the DVS problem where the target processor operates at discrete frequencies is well known to be NP-hard in general, we present effective algorithms for selecting the task execution order, and formulate the CPU frequency assignment problem as a *0-1 Integer Non-Linear Programming* (0-1 INLP) problem. Performance evaluation shows that our approach can result in significant energy savings compared to related approaches.

The rest of the paper is organized as follows: Section II presents our system model; Section III, Section IV and Section V describe the energy minimization problems; Section VI presents performance evaluation; Section VII gives some discussions and concludes this paper.

II. System Model

Table I summarizes the key notations used in this paper.

We consider a multi-tasking real-time embedded application running on a hardware platform of a single processor and multiple devices. We adopt the *frame-based* task model [11]. The application consists of a set of N periodic tasks, $\Gamma = \{\tau_i, i = 1, \ldots, N\}$. All tasks share the same system-wide frame denoted by T, i.e., the system has a single execution rate, and all tasks are invoked periodically with the same period T at time instants $n * T, n \in Z$. All tasks also share the same relative deadline equal to T. At the n^{th} invocation, all tasks are released at time instant $n * T$, and must complete execution within the time interval $[n * T, (n+1) * T]$. The same task schedule is repeated in all task frames.

A DVS-enabled CPU is capable of operating at discrete set of Q valid frequencies, i.e., Q active states, which are denoted as $\{f_1, f_2, ..., f_Q\}$, $f_1 < f_2 < ... < f_Q$. We normalize the frequency values with respect to f_Q, i.e., the range of f_i is $(0, 1]$. For each frequency level, there is a power consumption associated with it, and thus we have a set of power values: $\{P_{f_1}, P_{f_2}, ..., P_{f_Q}\}$, $P_{f_1} < P_{f_2} < ... < P_{f_Q}$. The Worst-Case Execution Time (WCET) of τ_i at maximum CPU frequency ($f_Q = 1$) is denoted c_i, and it scales linearly

Table I
Key notations used in this paper.

τ_i	Task i, i = 1, ..., N
c_i	WCET of τ_i under maximum CPU frequency
f_i	Valid CPU frequency levels, normalized to be a frational number in [0,1], $f_i \in \{f_1, f_2, ..., f_Q\}$
P_{f_i}	The corresponding power consumption when CPU operates at frequency f_i
T	Frame period shared by all tasks
δ_i	Device i, i=1, ..., M
D_i	Set of devices used by task τ_i
Γ_i	Set of tasks using device δ_i
B_i	Break-even time for device δ_i
P_i^a	Power consumption of device δ_i in *active* state
P_i^s	Power consumption of device δ_i in *sleep* state
T_i^{sd},T_i^{wu}	State transition time of δ_i for shutdown (from active to sleep) and wake-up (from sleep to active), respectively
E_i^{sd},E_i^{wu}	State transition energy overhead of δ_i for shutdown and wake-up, respectively
E_i^{tran}	Total state transition energy overheads of δ_i for one shutdown and one wake-up: $E_i^{sd} + E_i^{wu}$
E_d^{cpu}	CPU dynamic energy consumption within a single frame
E_d^{δ}	Total dynamic energy consumption of all devices within a single frame
$E_d^{\delta_i}$	Dynamic energy consumption of device δ_i within a single frame

with CPU frequency, i.e., its WCET at CPU frequency f is c_i/f. We assume inter-task DVS, hence the CPU frequency can only be changed at inter-task boundaries, and stays constant for the entire duration of a given task's execution. In order for all tasks to finish within a single frame, we must have $\sum_i^N c_i/f_i \leq T$, where f_i is the CPU frequency assigned to running task τ_i.

There are a total of M devices used by the application, denoted by the set $D = \{\delta_i, i = 1, \ldots, M\}$. Each device δ_i has two states: *active* and *sleep*. We assume *inter-task device scheduling*, i.e., each device δ_i used by a task τ_j must be in *active* state for the entire duration of τ_j's execution, and can transition to *sleep* state when τ_j completes execution. These assumptions are realistic since device state transitions incur non-trivial costs, and it's difficult to predict when a running task may request any device *during* its execution. The following parameters are associated with each device:

- P_i^a and P_i^s: device power consumption in *active* and *sleep* state, respectively. We assume $P_i^s = 0$ in this paper for simplicity, but our algorithms can be easily adapted to the case when $P_i^s > 0$.
- T_i^{sd} and T_i^{wu}: device state transition times for shutdown (from active to sleep) and wake-up (from sleep to active), respectively.
- E_i^{sd} and E_i^{wu}: device state transition energy overheads for shutdown and wake-up, respectively. We use E_i^{tran}

to denote $E_i^{sd}+E_i^{wu}$, i.e., the total energy consumption of device δ_i's state transition from active to sleep, and back from sleep to active.

Note that the energy and time overhead associated with the CPU frequency switching is on the order of magnitude of microjoule (μJ) and microsecond (μs) respectively. This is negligible compared to the overhead of device state transition which is on the order of millisecond (ms) and millijoule (mJ).

The device *break-even time* B_i denotes the minimum length of idle period that justifies a state transition from *active* to *sleep* state, given as [10], [12]:

$$B_i = \max(\frac{E_i^{tran}}{P_i^a}, T_i^{sd} + T_i^{wu}) \tag{1}$$

The first term denotes the minimum CPU idle interval length during which keeping δ_i in active state consumes at least the same amount of energy as switching it from active to sleep and then back from sleep to active; the second term denotes the total time of two state transitions from active to sleep and then back from sleep to active. If the length of continuous device idle time is less than B_i, then δ_i must remain in active state even when it is idle, since there is not enough idle time to put it to sleep while still achieving a net energy reduction.

Since each frame contains the same task schedule, we focus on the total energy consumption in a single frame. The total system energy E consists two parts: static energy E_s and dynamic energy E_d. The static energy is constant and not affected by DVS or DPM due to fixed frame size, hence it is ignored in our problem formulation. E_d consists of two parts: CPU energy consumption E_d^{cpu} and device energy consumption E_d^{δ}. We have the following expressions:

$$\begin{aligned} E_d^{cpu} &= \sum_{i=1}^{N} \frac{c_i}{f_i} P_{f_i}, \\ E_d^{\delta} &= \sum_{i=1}^{M} (l_i P_i^a + o_i E_i^{tran}), \\ E_d &= E_d^{cpu} + E_d^{\delta} \end{aligned} \tag{2}$$

where l_i denotes the total amount of time that δ_i spends in active state, and o_i denotes the number of times that δ_i goes into sleep state in a single frame. Our optimization objective is to minimize the total dynamic energy consumption E_d by finding the optimal solution of task execution order within each frame and CPU frequency assignment to each task.

In this paper, we consider two possible dependency constraints among tasks:

- Tasks are independent from each other, i.e., their order of execution within each frame is flexible.
- Tasks must execute with a fixed total task execution order imposed by application requirements within each frame.

Assume that each task τ_i uses a set of devices $D_i = \{\delta_{i1}, ..., \delta_{i|D_i|}\}$. Within each set D_i, device indices are sorted in increasing order of their break-even times. We classify device usage pattern by tasks into two categories:

- *zero-full-overlapping*: any given pair of tasks must use the same set of devices, or use two disjoint sets of devices that do not overlap with each other, i.e., $\forall i, j \in [1, \ldots, N]$ $(D_i = D_j) \vee (D_i \cap D_j = \emptyset)$.
- *partial-overlapping*: there may be non-zero overlaps between the set of devices used by any pair of tasks, i.e., $\exists i, j \in [1, \ldots, N]$ $(D_i \neq D_j) \wedge (D_i \cap D_j \neq \emptyset)$.

Table II
SYSTEM CONFIGURATIONS

	$flexible\ order$	$fixed\ order$
$zero-full-overlapping$	$zfov-flex$	$zfov-fix$
$partial-overlapping$	$pov-flex$	$pov-fix$

We consider a total of four different system configurations according to task dependency constraints and device usage, as shown in Table II. We address each of these configurations in the following sections.

III. ZFOV-FLEX: ZERO-FULL-OVERLAPPING & FLEXIBLE ORDER

In this section, we consider the case when all tasks are independent and device usage is *zero-full-overlapping*. We present a two-phase algorithm and formulate a 0-1 Integer Non-Linear Programming (0-1 INLP) problem to determine the optimal solution.

A. Phase 1

We group all tasks that use the same set of devices into a contiguous sequence in order to maximize the lengths of device idle durations. We partition the total set of N tasks into $m+1$ subsets denoted by $S = \{S_0, S_1, ..., S_m\}, m \leq M$. S_0 denotes the subset of tasks that use no device, and $S_{i|i\neq 0}$ denotes the subset of tasks that use the same set of devices D_i, i.e., $(\bigcup_{i=1}^{m} D_i = \delta) \wedge (D_i \bigcap_{i\neq j|i,j\in[1,\ldots,m]} D_j = \emptyset)$. Each subset S_i is called a *Homogeneous Device Usage* task group, or a *task group* for short. For our purposes, execution order of tasks in the same task group has no effect on system energy consumption. After task grouping, the device idle durations either become larger or stay the same, so any device has higher possibility to transition into sleep state. Therefore, the total energy consumption of CPU and devices must either decrease or remain the same, i.e., task grouping leads to miss no optimal solution. Due to our assumption that device usage is *zero-full-overlapping*, each device is used by at most one task group. Therefore, the active and idle durations of each device coincide with the active and idle durations of the task group that uses it, and there is no fragmentation of idle/active durations of each device within each frame. This means that each device either stays active for the entire frame, or goes from active to sleep and back again *only once* within each frame.

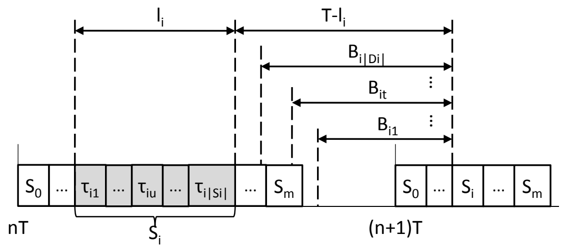


Figure 1. Relationship between break-even time and frame T.

B. Phase 2

In this phase, we formulate a 0-1 INLP problem for minimizing multi-resource energy consumption.

Figure 1 shows the execution order of $m+1$ task groups for two frame periods. S_i contains $|S_i|$ tasks: $\tau_{i1}, \tau_{i2}, ..., \tau_{i|S_i|}$. B_{it} stands for the break-even time of device $\delta_{it}, t \in [1, ..., |D_i|]$ (For ease of presentation, we assume S_0 use one device δ_0 with break-even time and active power equal to zero). For each $S_{i|i\in[0,...,m]}$, we define several binary variables:

- $x_{iuv} = 1$ if task τ_{iu} runs on frequency f_v.
- $y_{it} = 1$ if device δ_{it} can go to sleep, while $y_{it} = 0$ stands for that the device can not sleep.

Where, $f_v \in \{f_1, f_2, ..., f_Q\}$.

If δ_{it} can go to sleep, the sum of its break-even time and its time spent in the active state must not exceed the frame length. Since each device is used by at most one task group, its time spent in the active state within each frame equals to the execution time of the task group that uses it. Hence:

$$\begin{aligned} &\forall t \in 1, ..., |D_i|,\ l_i + y_{it}B_{it} \leq T, \\ &where\ l_i = \sum_{u=1}^{|S_i|}\sum_{v=1}^{Q} x_{iuv}\tfrac{c_{iu}}{f_v} \end{aligned} \tag{3}$$

Recall that within each set $D_i = \{\delta_{i1}, \ldots, \delta_{i|D_i|}\}$, device indices are sorted in increasing order of their break-even times. Therefore, if δ_{it} can go to sleep, devices with smaller or equal break-even times $(\delta_{i1}, \ldots, \delta_{i(t-1)})$ can also go to sleep, since all devices in D_i have identical usage pattern due to our *zero-full-overlapping* device usage assumption. This fact can be expressed with a constraint:

$$\forall t \in 1, ..., |D_i| - 1,\ y_{it} \geq y_{i(t+1)} \tag{4}$$

Total energy consumption $E_d^{D_i}$ of all devices in D_i within a single frame is:

$$\begin{aligned} E_1^{D_i} &= \sum_{t=1}^{|D_i|} y_{it}(l_i P_{it}^a + E_{it}^{tran}) \\ E_2^{D_i} &= \sum_{t=1}^{|D_i|} (1 - y_{it}) T P_{it}^a \\ E_d^{D_i} &= E_1^{D_i} + E_2^{D_i} \end{aligned} \tag{5}$$

where $E_1^{D_i}$ is sum of the energy consumption of those devices that can go to sleep, and $E_2^{D_i}$ is sum of the energy consumption of those devices that must keep active for the entire frame.

The total energy consumption of all devices in the system within a single frame is:

$$E_d^{\delta} = \sum_{i=0}^{m} E_d^{D_i} \tag{6}$$

Each task τ_{iu} can run on only one frequency, which is expressed as:

$$\forall i \in [0, ..., m],\ \forall u \in [1, ..., |S_i|],\ \sum_{v=1}^{Q} x_{iuv} = 1 \tag{7}$$

To meet deadlines, the task set should finish before or at end of the current frame:

$$\sum_{i=0}^{m}\sum_{u=1}^{|S_i|}\sum_{v=1}^{Q} x_{iuv}\tfrac{c_{iu}}{f_v} \leq T \tag{8}$$

The total CPU energy consumption in a single frame is:

$$E_d^{cpu} = \sum_{i=0}^{m}\sum_{u=1}^{|S_i|}\sum_{v=1}^{Q} x_{iuv} P_{f_v} \tfrac{c_{iu}}{f_v} \tag{9}$$

The optimization objective is to minimize E_d, the total dynamic energy consumption of the CPU and all devices within a single frame:

$$\min\ E_d = E_d^{cpu} + E_d^{\delta} \tag{10}$$

Conditions 3 to 9 form a constraint set, and combined with the optimization objective in Condition 10, it forms a 0-1 INLP problem. Furthermore, if substituting each term of x_{iuv} multiplied by y_{it} with one binary variable, we can formulate a 0-1 Integer Linear Programming (0-1 ILP) problem, detail shown in Appendix A.

IV. ZFOV/POV-FIX: ZERO-FULL/PARTIAL-OVERLAPPING & FIXED-ORDER

In this section, we consider the case when all tasks must execute with a fixed total task execution order imposed by application requirements within each frame. Without loss of generality, we assume the execution order is the same as order of task indices, i.e., from τ_1 to τ_N; τ_1 starts to execute at the beginning of each frame. The device usage pattern can be either *zero-full-overlapping* or *partial-overlapping*, and both can be addressed by our algorithm proposed in this section.

For each device, its *active intervals* within a frame is the set of time intervals when the device is in use continuously and hence must stay active. Each active interval is delimited by two tasks' start and end time instants $[start(\tau_j), end(\tau_{j'})]$, where τ_j $(\tau_{j'})$ uses the device at the beginning (end) of the active interval.

Figure 2 shows an example when device δ_i has two active intervals within a frame, which partition a frame T

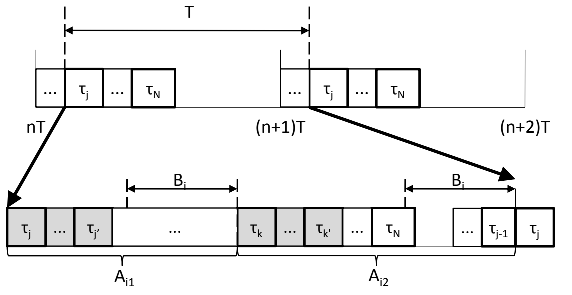


Figure 2. Device δ_i has two active intervals (shaded area in the lower figure) within a frame: $[start(\tau_j), end(\tau_{j'})]$, $[start(\tau_k), end(\tau_{k'})]$.

into two *segments*: A_{i1} and A_{i2}. We use the example in Figure 2 to explain our two-phase approach. Phase 1: find all the *segments* partitioned by the active intervals. Phase 2: formulate a 0-1 INLP problem to determine the frequencies which lead to the minimum energy consumption.

A. Phase 1

A device $\delta_i \in \{\delta_1, ..., \delta_M\}$ may have more than one active intervals (segments) within a frame. We use A_i and $|A_i|$ to denote the set and the number of all segments for device δ_i, respectively, and use $A = \{A_1, \ldots, A_M\}$ to denote the whole set of the segments for all the M devices. $A_{iw|w\in[1,\ldots,|A_i|]}$ stands for w^{th} segment. As shown in Figure 2, two segments are denoted as $A_{i1} = [start(\tau_j), start(\tau_k)], A_{i2} = [start(\tau_k), next\_start(\tau_j)]$, each of which starts from the start of an active interval and ends at the start of the next active interval. (We use $next\_start(\tau_j)$ to denote start time of the instance of τ_j in the next frame.) The usage relationship of M devices and N tasks form a $M \times N$ matrix, which can be traversed to obtain all the active intervals and the whole segment set A. Since the segments of a device δ_i may overlap with the segments of other devices, we should consider each device separately and calculate the energy consumption of δ_i by summing up the energy for each segment A_{iw}.

B. Phase 2

From Phase 1, we have obtained A for all tasks in increasing order of task indices, i.e, the beginning task in A_{i1} is the first task use δ_i within a frame. For each $A_{iw|w\in[1,\ldots,|A_i|]}$, we define several binary variables:

- $y_{iw} = 1$ if δ_i can transition to sleep while $y_{iw} = 0$ for keeping active, in A_{iw}.

Note that here y_{iw} has different meaning with that in Section III. One denotes w^{th} device in D_i while the other represent the w^{th} segment in A_i. For each task τ_u, define Q binary variables:

- $x_{uv} = 1$ if τ_u runs at frequency f_v.

Since the processor may become idle only after the execution of the last task in a frame, we treat the *last* segment ($A_{i|A_i|}$) as a separate case from all other *non-last* segments ($A_{iw|w\in[1,\ldots,|A_i|-1]}$). For example, in Figure 2, the set of segments A_i for device δ_i in a frame has size 2: the first one, $[start(\tau_j), start(\tau_k)]$, is a *non-last* segment; the second one $[start(\tau_k), next\_start(\tau_j)]$, is the *last* segment in a frame. (If there is only one segment, we treat it as last segment.) Next, we present the constraints for $A_{iw|w\in[1,\ldots,|A_i|-1]}$, followed by those for $A_{i|A_i|}$.

1) Constraint set for the non-last segments (A_{i1}): If the device idle duration $[end(\tau_{j'}), start(\tau_k)]$ in Figure 2 exceeds break-even time B_i, δ_i can go to sleep at the end of the active interval ($end(\tau_{j'})$). Since the scheduling discipline is work-conserving, the length of this device idle duration is equal to sum of the execution times of the set of tasks from $\tau_{j'+1}$ to τ_{k-1}. This condition can be expressed as:

$$y_{i1} B_i \le \sum_{u=j'+1}^{k-1} \sum_{v=1}^{Q} x_{uv} \tfrac{c_u}{f_v} \tag{11}$$

The total energy consumption of device δ_i in the time interval A_{i1} can be obtained as follows:

$$\begin{aligned} E_1^{\delta_i} &= y_{i1}(\sum_{u=j}^{j'} \sum_{v=1}^{Q} x_{uv} \tfrac{c_u}{f_v} P_i^a + E_i^{tran}) \\ E_2^{\delta_i} &= (1 - y_{i1}) \sum_{u=j}^{k-1} \sum_{v=1}^{Q} x_{uv} \tfrac{c_u}{f_v} P_i^a \\ E_{d1}^{\delta_i} &= E_1^{\delta_i} + E_2^{\delta_i} \end{aligned} \tag{12}$$

where $E_1^{\delta_i}$ is the device energy consumption when δ_1 can go to sleep at $end(\tau_{j'})$, and $E_2^{\delta_i}$ is the device energy consumption when δ_i keeps active for all the length of A_{i1}.

2) Constraint set for the last segment (A_{i2}): Unlike the non-last segments, CPU may become idle after τ_N execution in the last segments. Therefore, the device idle duration $[end(\tau_{k'}), next\_start(\tau_j)]$ can not be expressed by sum of the execution times of the set of tasks from τ_k to the next instance of τ_{j-1}. We use the following constraint to represent the relationship between the idle duration and break-even time B_i:

$$y_{i2} B_i \le T - \sum_{u=j}^{k'} \sum_{v=1}^{Q} x_{uv} \tfrac{c_u}{f_v} \tag{13}$$

The total energy consumption of device δ_i in the time interval A_{i2} can be obtained as follows:

$$\begin{aligned} E_1^{\delta_i} &= y_{i2}(\sum_{u=k}^{k'} \sum_{v=1}^{Q} x_{uv} \tfrac{c_u}{f_v} P_i^a + E_i^{tran}) \\ E_2^{\delta_i} &= (1 - y_{i2})(T - \sum_{u=j}^{k-1} \sum_{v=1}^{Q} x_{uv} \tfrac{c_u}{f_v}) P_i^a \\ E_{d2}^{\delta_i} &= E_1^{\delta_i} + E_2^{\delta_i} \end{aligned} \tag{14}$$

where $E_1^{\delta_i}$ is the device energy when δ_1 can go to sleep at $end(\tau_{k'})$, and $E_2^{\delta_i}$ is the device energy when δ_i keeps active for the device idle duration $[end(\tau_{k'}), next\_start(\tau_j)]$. And, the energy consumption of δ_i is:

$$E_d^{\delta_i} = \sum_{w=1}^{|A_i|} E_{dw}^{\delta_i} = E_{d1}^{\delta_i} + E_{d2}^{\delta_i} \tag{15}$$

For clarity of presentation, we have described the constraint conditions for the example in Figure 2 with two active intervals (segments) in a frame for device δ_i. These conditions can be easily generalized for the case with arbitrary number of active intervals for device δ_i.

Having obtained the energy consumption of each device, the total energy of all M devices in a frame is:

$$E_d^{\delta} = \sum_{i=1}^{M} E_d^{\delta_i} \tag{16}$$

Each task τ_u can run on only one frequency, which is expressed as:

$$\forall u \in [1, ..., N], \ \sum_{v=1}^{Q} x_{uv} = 1 \tag{17}$$

To meet deadlines, the total execution time can not exceed the frame:

$$\sum_{u=1}^{N} \sum_{v=1}^{Q} x_{uv} \frac{c_u}{f_v} \leq T \tag{18}$$

The total CPU energy consumption in a single frame is:

$$E_d^{cpu} = \sum_{u=1}^{N} \sum_{v=1}^{Q} x_{uv} P_{f_v} \frac{c_u}{f_v} \tag{19}$$

The optimization objective is to minimize E_d, total dynamic energy consumption of the CPU and all devices within a single frame:

$$\min \ E_d = E_d^{cpu} + E_d^{\delta} \tag{20}$$

V. POV-FLEX: PARTIAL-OVERLAPPING & FLEXIBLE ORDER

In this section, we address the case when device usage pattern is *partial-overlapping* and task execution order is flexible within each frame. We present two algorithms with different tradeoffs between running speed and optimality.

A. Conservative Approximation

We group any two tasks that share at least one common device into a single task group that uses the union of devices that are used by the two original tasks, i.e., for any pair of tasks τ_i and τ_j, if $D_i \cap D_j \neq \emptyset$, then group τ_i and τ_j into a single task group that uses $D_i \cup D_j$. Then the techniques presented in Section III can be applied. This is a conservative approximation of the actual device usage pattern, hence the solution obtained is sub-optimal, but it is very efficient since the solver only needs to be invoked once.

B. Efficient Heuristic

To improve the efficacy, we proposed an efficient heuristic algorithm base on lexicographical order to determine a task execution order, and then the techniques presented in Section IV can be adopted.

Algorithm 1 Determine the task running order

```
Input: ($\Gamma, \Gamma_i, D, D_i$);
Output: a task running order;
1: for $i = 1$ to $M$ do
2:     if $\sum_{\tau_j \in \Gamma_i} c_j + B_i \geq T$ then
3:         $D = D - \delta_i$;
4:     end if
5: end for
6: if $D = \emptyset$ then
7:     return any task running order is OK;
8: end if
9: sort $D$ according to the decreasing order of $P_i^a$, and in
   increasing order of $B_i$ ($E_i^{tran}$) if some $P_i^a$ equals;
10: sort $\Gamma$ in the reverse lexicographical order;
11: reverse the task order in each odd group, which becomes
    the lexicographical order in each odd group;
12: return the task running order;
```

We would like to let the devices with higher P_i^a and lower B_i (E_i^{tran}) go to sleep. There are two reasons: (i) this can save more energy due to devices with high P_i^a can sleep; (ii) these devices can go to sleep less costly due to small B_i (E_i^{tran}). A group in Algorithm 1 is headed with the same device. Lines from (1) to (8) check a trivial case if all devices can not switch to sleep even when all tasks run at maximum frequency. Line (9) sorts the device set accordingly. Line (10) make tasks using common devices execute continuously in each group and line (11) may make the common devices in two adjacent groups go to sleep too. Suppose three devices has $P_1^a > P_2^a > P_3^a$ and $\delta_1, \delta_2, \delta_3$ can be seen as alphabets c, b, a respectively. The device usage of four tasks is $D_1 = \{\delta_1\} = c, D_2 = \{\delta_2, \delta_3\} = ba, D_3 = \{\delta_1, \delta_2, \delta_3\} = cba, D4 = \{\delta_2\} = b$. The first group is $\{\tau_1, \tau_3\}$ headed by δ_1, and the second group is $\{\tau_2, \tau_4\}$ headed by δ_2. After execution line (10), the task order is the reverse lexicographical order $\tau_3, \tau_1, \tau_2, \tau_4$. Then, the order changes to $\tau_1, \tau_3, \tau_2, \tau_4$ by line (11), where δ_3 is also grouped together. We can see that more devices are used continuously, thus more devices may go to sleep.

VI. PERFORMANCE EVALUATION

Our simulation is based on processor and device parameters taken from [13] and [12] respectively. Table III lists parameters of some devices[1]. The Intel XScale processor

[1]Since we assume the device power consumption in the sleep state ($P_i^s = 0$) is zero, the device parameters from [12] are adjusted by subtracting its power consumption in the sleep state from that in the active state.

Table III
DEVICE PARAMETERS

$Device$	$P^a_{(mW)}$	$T^{sd(wu)}_{(ms)}$	$E^{sd(wu)}_{(mJ)}$
$Realtek\,Ethernet\,Chip$	105	10	0.4
$MaxStream\,wireless\,module$	745	40	3.8
$IBM\,Microdrive$	1200	12	4.8
$SST\,Flash\,SST39LF020$	124	1	0.049
$SimpleTech\,Flash\,Card$	205	2	0.16
$Fujitsu\,2300AT\,Hard\,disk$	1300	20	10

Table IV
ACTUAL CPU OPERATION FREQUENCIES AND APPROXIMATE POWER FUNCTION

$Normalized\,frequency$	0.15	0.4	0.6	0.8	1
$Power(mW)$	80	170	400	900	1600
$Approximate\ power\ function$: $40 + 1560f^3$					

is used, with the discrete operation mode and approximate power function shown in Table IV after normalizing the CPU frequency so that $0 < f \leq 1$.

In general, there are two approaches in designing the energy saving schemes for the discrete operation modes. The first one is to patch the scheduling obtained under the continuous frequency model and reassign each task frequency to the closest higher frequency, by which we consider two algorithms as follows:

- DVS-ONLY, which is treated as the baseline approach. All tasks shared the same CPU frequency, which is lowered aggressively to minimize CPU energy consumption while preserving schedulability, i.e., $f = \frac{\sum_{i=1}^{N} c_i}{T}$, under continuous frequency model. Then, the frequency is reassigned to the closest higher valid frequency. All devices keep active throughout the entire frame period.
- TL-CS: the *Task-Level Critical Speed*, defined as [7] under continuous frequency model, is the frequency obtained by setting the derivative of the following equation to 0 and equal to $(\frac{P^a(D_i)}{2\alpha})^{\frac{1}{3}}$.

$$\alpha c_i f_i^2 + P^a(D_i)\frac{c_i}{f_i}$$

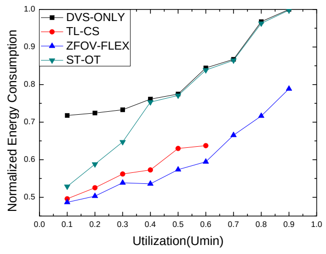
where f_i is the CPU frequency for task τ_i; $P^a(D_i)$ is the total power consumption of D_i, devices used by τ_i. Then, reassign f_i to the closest higher valid frequency. A device will *always* go to sleep, if it is feasible to do so for the reassigned frequency and break-even time.

The second approach is to design the optimal energy saving scheme considering the discrete frequency model from the beginning.

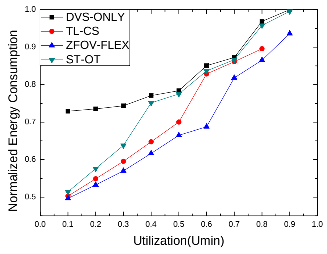
- ST-OT: The approach in [10] assumes that *Sequence of Tasks as One Task* and all tasks within one frame use all devices. Assuming continuous frequency processor model, the approach determines the global minimum energy consumption through trying all local extreme frequency points of $M + 1$ intervals obtained by dividing $[\sum_{i=1}^{N} c_i, T]$ with the non-decreasing order of break-even times. Due to the discrete frequency model in this paper, we adapt this approach by trying all possible frequencies and returning the minimum energy consumption value. So, it doesn't have to try $M + 1$ but Q frequency values.
- ZFOV-FLEX, ZFOV/POV-FIX and POV-FLEX: Our algorithms presented in Sections III, IV, V respectively.

We compare the performance of the above three approaches for different system configurations.

We use ILOG CPLEX (not ILOG Parallel CPLEX) solver to solve the 0-1 INLP problems. The experiments are performed on a Windows PC with an Intel Core2 2.83GHZ 32-bit processor and 2GB main memory. In all the figures in this section, U_{min} denotes the minimum CPU utilization when CPU frequency is set to be the maximum normalized value of 1. The energy consumptions in each figure are normalized to that of the DVS-ONLY for each setting, respectively.



(a) 1-2 devices



(b) 3 devices

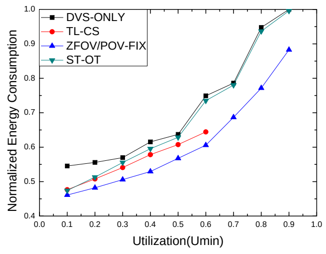
Figure 3. Comparison of ZFOV-FLEX, ST-OT, TL-CS, and DVS-ONLY. T is $120ms$; each task uses 1-2 and 3 devices.

A. ZFOV-FLEX: zero-full-overlapping & flexible order

In this experiment, we use the following procedure to set the *zero-full-overlapping* device usage pattern for ZFOV-FLEX. First, a *partition* of all devices in Table III is generated randomly denoted by $\{D_1, ..., D_m\}$, where $m \leq 6$. Then, one device set is chosen randomly from $\{\emptyset, D_1, ..., D_m\}$ to be used by τ_i. For fairness of comparison, the same device usage pattern is used and tasks used the same device set are also grouped for TL-CS. Figures 3 shows the results. As U_{min} increases, the taskset becomes non-schedulable with TL-CS, hence the TL-CS curve is shorter than others in some figures.

B. ZFOV/POV-FIX: zero-full/partial-overlapping & fixed order

It's randomly chosen several devices from Table III for each task in this experiment. Figures 4 shows the performance comparison of ZFOV-/POV-FIX, ST-OT, TL-CS, and DVS-ONLY. From the results of the above two experiments, we can see that the proposed approaches outperform the other approaches across the whole range of U_{min}.


(a) 1-2 devices

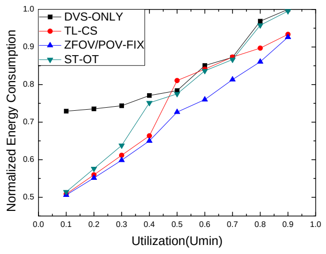
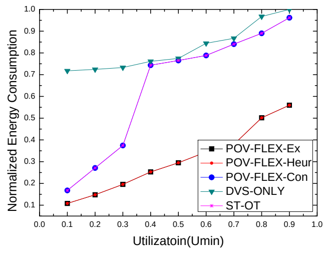

(b) 3 devices

Figure 4. Comparison of ZFOV-/OV-FIX, ST-OT, TL-CS, and DVS-ONLY. T is $120ms$; each task uses 1-2 and 3 devices.


(a) 1-2 devices

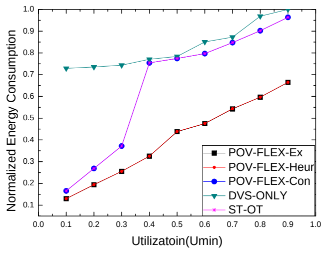
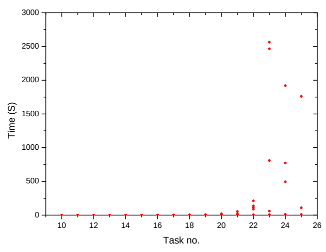

(b) 3 devices

Figure 6. Comparison of POV-FLEX-Ex, POV-FLEX-Con, POV-FLEX-Heur, ST-OT, and DVS-ONLY. T is $120ms$; each task uses 1-2 or 3 devices.


(a) ZFOV-FLEX

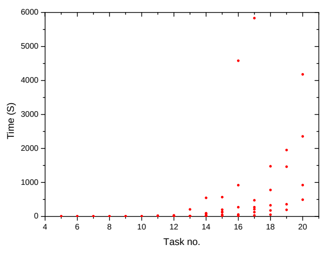

(b) ZFOV/POV-FIX

Figure 5. Running time of the solver for the experiments in Figure 3(b) and in Figure 4(b).

ZFOV-FLEX and ZFOV/POV-FIX save more energy compared to ST-OT up to 29.1% and 17.6% respectively. Due to the assumption that all devices are active when the processor is non-idle, the performance of ST-OT moves towards DVS-ONLY when the utilization grows.

ZFOV-FLEX and ZFOV/POV-FIX save more energy compared to TL-CS up to 17.0% and 10.4% respectively, when up to 3 devices are used. This saving increases as the number of devices grows, since more and more devices can go to sleep state by our approaches. The maximum energy savings occurs in the utilization range of $[0.4, 0.7]$. The reason is that most devices can go to sleep in both our algorithms and TL-CS when utilization is less than 0.4 while most devices have to keep active all frame long when utilization is greater than 0.7, and the numbers of devices can sleep in our approaches and TL-CS have maximum differences between the two extreme conditions.

In some cases, such as the utilization range $[0.45, 0.6]$ in Figure 4(b), TL-CS performs even worse than DVS-ONLY. The reason is that TL-CS derives the task-level critical speed which leads to the minimum energy consumption for each task, but not for the whole task set. When all devices must keep active for entire frame in both the two algorithms, DVS-ONLY has lower CPU energy consumption and thus performs better than TL-CS.

Figure 5 shows the running time of CLPEX solver for the experiments in Figure 3 and Figure 4, with task numbers ranging from 10 to 25 and from 5 to 20, respectively. We can see that ZFOV-FLEX's has lower running time and better scalability than ZFOV/POV-FIX, which is mainly due to the number of variables in ZFOV-FLEX is less than that in ZFOV/POV-FIX.

C. POV-FLEX: partial-overlapping & flexible order

Each task randomly chooses several devices from Table III. In Figure 6, POV-FLEX-Ex denotes an exhaustive search approach, where we try all possible task execution orders exhaustively, and determine the solution of CPU frequency assignments with minimum energy consumption for each task execution order using the techniques presented in Section IV. POV-FLEX-Con and POV-FLEX-Heur denote the conservative approximation approach and efficient heuristic as presented in Section V. We can see that POV-FLEX-Heur has very good performance and its results are nearly the same as the optimal results of POV-FLEX-Ex when each task uses either 1-2 or 3 devices. The reason is that POV-FLEX-Heur almost guarantees each of the devices with high P_i^a can be used as continuously as possible and has higher possibility to switch into sleep state. When device sets overlap heavily and all tasks may be formed to only one group, POV-FLEX-Cons performance is compromised and degenerates to ST-OT. In this case, the performance of POV-FLEX-Con and ST-OT moves towards to DVS-ONLY as the system utilization becomes higher because most devices are used for a long time and may keep active within the whole frame.The running time of POV-FLEX-Heur is longer than that of POV-FLEX-Con, which is also shown in Figure 5.

VII. Discussions and Conclusions

A. Discussions

An assumption in Section III to V is that all tasks must execute without any idle time in-between within a frame, i.e., the work-conserving schedule discipline is adopted. The formulations can be easily adapted to the non-work-conserving case by setting some variables of the idle times, which may lower the energy consumption further.

For Section III, since each device is used by at most one task group, the device either stays active for the entire frame, or swithces from active to sleep state and back again only once within each frame. Therefore, the *0-1 INLP* derives

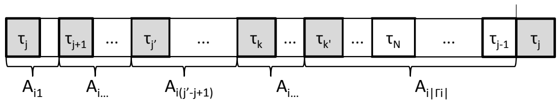


Figure 7. Segments after insert idle time after each task.

minimum energy consumption under either work-conserving or non-work-conserving schedule discipline.

For Section IV, we set variable t_j of an idle time after each task τ_j. In phase 1, the segments in Figure 2 becomes $|\Gamma_i|$ segments shown in Figure 7, where $|\Gamma_i|$ is the number of tasks using device δ_i. Each segment begins from one task using the device and ends at the beginning of the next task using the device, so there is only one task using the device in each segment, i.e., each active interval contains only one task. We do not have to differentiate the last and non-last segments. We only need to insert the idle time t_i into all the formula and change "$\leq$" to "$=$" in Condition (18) in phase 2, which formulates a mixed integer non-linear programming (*MINLP*) problem, detail shown in Appendix B.

B. Conclusions

In this paper, we addressed the multi-resource energy minimization problem for frame-based real-time tasks running on a DVS-capable processor of discrete set of operational modes with multiple off-chip DPM-capable devices. The optimization problem has two degrees of freedom: *task execution order* within each frame, and *CPU frequency assignment* to each task assuming inter-task DVS. We present effective algorithms for selecting the task execution order, and formulate the CPU frequency assignment problem as a *0-1 Integer Non-Linear Programming* (0-1 INLP) problem under different system configurations. Simulation results show that our approach can result in significant energy savings compared to related approaches.

Acknowledgement: We would like to thank the anonymous reviewers for their constructive comments. This work was partially supported by the NSF of China under Grant No. 60973017 and the Fundamental Research Funds for the Central Universities under Grant No. N090104001.

References

[1] J. Chen and C. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms," in *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2007, pp. 28–38.

[2] V. Swaminathan and K. Chakrabarty, "Energy-conscious, deterministic I/O device scheduling in hard real-time systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 7, pp. 847–858, 2003.

[3] ——, "Pruning-based, energy-optimal, deterministic I/O device scheduling for hard real-time systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 4, no. 1, p. 167, 2005.

[4] X. Fan, C. Ellis, and A. Lebeck, "The Synergy between Power-aware Memory Systems and Processor Voltage Scaling," in *Workshop on Power-Aware Computing Systems*, 2003.

[5] D. Snowdon, S. Ruocco, and G. Heiser, "Power management and dynamic voltage scaling: Myths and facts," in *2nd International Workshop on Power-Aware Real-Time Computing (PARC)*, 2005.

[6] R. Jejurikar and R. Gupta, "Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems," in *International Symposium on Low Power Electronics and Design*, 2004, pp. 09–11.

[7] J. Zhuo and C. Chakrabarti, "System-level energy-efficient dynamic task scheduling," in *42nd Design Automation Conference*, 2005, pp. 628–631.

[8] H. Aydin, V. Devadas, and D. Zhu, "System-level energy management for periodic real-time tasks," in *27th IEEE International Real-Time Systems Symposium*, 2006, pp. 313–322.

[9] H. Cheng and S. Goddard, "Integrated device scheduling and processor voltage scaling for system-wide energy conservation," in *2nd International Workshop on Power-Aware Real-Time Computing (PARC)*, 2005.

[10] V. Devadas and H. Aydin, "On the interplay of dynamic voltage scaling and dynamic power management in real-time embedded applications," in *8th ACM international conference on Embedded software*, 2008.

[11] R. Xu, D. Mossé, and R. Melhem, "Minimizing expected energy in real-time embedded systems," in *5th ACM international conference on Embedded software*, 2005, pp. 251–254.

[12] H. Cheng and S. Goddard, "Online energy-aware I/O device scheduling for hard real-time systems," in *Design, automation and test in Europe*, vol. 6, no. 10, 2006, pp. 1055–1060.

[13] R. Xu, D. Mosse, and R. Melhem, "Minimizing Expected Energy Consumption in Real-Time Systems through Dynamic Voltage Scaling," *ACM Transactions on Computer Systems-TOCS*, vol. 25, no. 4, 2007.

Appendix

A. 0-1 Integer Linear Programming (0-1 ILP) for ZFOV-FLEX

In phase 2 of Section III, we can formulate a 0-1 ILP problem for minimizing multi-resource energy consumption.

Figure 1 shows the execution order of $m+1$ task groups for two frame periods. For each $S_{i|i\in[1,...,m]}$, we define several binary variables:

- $X_{ituv} = 1$ iff device δ_{it} can go into sleep state and task τ_{iu} run on frequency f_v.
- $Y_{ituv} = 1$ iff task τ_{iu} run on frequency f_v but device δ_{it} cannot sleep.

Where, $f_v \in \{f_1, f_2, ..., f_Q\}$.

If δ_{it} can go to sleep, then sum of its break-even time and its time spent in the active state must not exceed the frame length. Hence:

$$\begin{aligned} & l_i + B_{it}\sum_{v=1}^{Q} X_{it1v} \le T, \\ & where\ l_i = \sum_{u=1}^{|S_i|}\sum_{v=1}^{Q} X_{ituv}\tfrac{c_{iu}}{f_v} \end{aligned} \tag{21}$$

As shown in Figure 1, in one frame beginning from τ_{i1}, since the operational state of device δ_{it} must be the same and each task can be assigned only one frequency, we have the following constraints:

$$\begin{aligned} \forall u, u' \in [1, ..., |S_i|],\ & (\sum_{v=1}^{Q} X_{ituv} = \sum_{v=1}^{Q} X_{itu'v}) \\ & \wedge(\sum_{v=1}^{Q} Y_{ituv} = \sum_{v=1}^{Q} Y_{itu'v}) \end{aligned} \tag{22}$$

Note that we use index $it1v$ in some equations or constraints due to the equivalence in Condition (22). And, δ_{it} either can go into sleep state or keep active all frame long:

$$\sum_{v=1}^{Q} (X_{it1v} + Y_{it1v}) = 1 \tag{23}$$

Besides Condition (22), we use the following equations to guarantee that each task runs at only one frequency:

$$\begin{aligned} & \forall t, t' \in [1, |D_i|],\ \forall u \in [1, |S_i|],\ \forall v \in [1, Q], \\ & X_{ituv} + Y_{ituv} = X_{it'uv} + Y_{it'uv} \end{aligned} \tag{24}$$

Note that is why we use index $i1uv$ in some equations or constraints.

Recall that within each set $D_i = \{\delta_{i1}, \ldots, \delta_{i|D_i|}\}$, device indices are sorted in increasing order of their break-even times. Therefore, if δ_{it} can go to sleep, devices with smaller or equal break-even times $(\delta_{i1}, \ldots, \delta_{i(t-1)})$ can also go to sleep, since all devices in D_i have identical usage pattern due to our *zero-full-overlapping* device usage assumption. This fact can be expressed with a constraint:

$$\forall t \in [1, |D_i| - 1],\ \sum_{v=1}^{Q} X_{it1v} \ge \sum_{v=1}^{Q} X_{i(t+1)1v} \tag{25}$$

Total energy consumption $E_d^{D_i}$ of all devices in D_i within a single frame is:

$$\begin{aligned} E_1^{D_i} &= \sum_{t=1}^{|D_i|} (P_{it}^a \sum_{u=1}^{|S_i|}\sum_{v=1}^{Q} X_{ituv}\tfrac{c_{iu}}{f_v} + E_{it}^{tran}\sum_{v=1}^{Q} X_{it1v}) \\ E_2^{D_i} &= \sum_{t=1}^{|D_i|} TP_{it}^a \sum_{v=1}^{Q} Y_{it1v} \\ E_d^{D_i} &= E_1^{D_i} + E_2^{D_i} \end{aligned} \tag{26}$$

where $E_1^{D_i}$ is sum of the energy consumption of those devices that can go to sleep, and $E_2^{D_i}$ is sum of the energy consumption of those devices that must keep active for the entire frame.

Total energy consumption of all devices in the system within a single frame is:

$$E_d^{\delta} = \sum_{i=1}^{m} E_d^{D_i} \tag{27}$$

To meet deadlines, each task group should finish before or at end of the current frame:

$$\sum_{i=0}^{m}\sum_{u=1}^{|S_i|}\sum_{v=1}^{Q} (X_{i1uv} + Y_{i1uv})\tfrac{c_{iu}}{f_v} \le T \tag{28}$$

Total CPU energy consumption within a single frame is:

$$\sum_{i=0}^{m}\sum_{u=1}^{|S_i|}\sum_{v=1}^{Q} (X_{i1uv} + Y_{i1uv})P_{f_v}\tfrac{c_{iu}}{f_v} \tag{29}$$

The optimization objective is to minimize E_d, total dynamic energy consumption of the CPU and all devices within a single frame:

$$\min\ E_d = E_d^{cpu} + E_d^{\delta} \tag{30}$$

Conditions 21 to 29 form a linear constraint set, and combined with the optimization objective in Condition 21, it forms a 0-1 ILP problem. In general, many problems with thousands of variables and constraints can be solved efficiently with modern ILP tools, such as CPLEX. The complexity of ILP solving is determined by the number of variables. In the worst case, every task use all M devices and has to be associated with M variable X sets and M variable Y sets and multiplied by Q CPU operation modes, the number of binary variables is $2MNQ$. However, in practice, since it seldom occurs that all devices in one system used by each task, the number of variables in sharply lower than $2MNQ$.

B. MINLP

There is only one task τ_w using device δ_i in each segment A_{iw}, and we formulate a MINLP as follows:

$$\begin{aligned} \min\quad & E_d = E_d^{cpu} + E_d^{\delta} \\ sub.to\quad & y_{iw}B_i \le t_{iw(\sim\delta_i)}, \\ & E_{dw}^{\delta_i} = \sum_{v=1}^{Q} x_{wv}\tfrac{c_w}{f_v}P_i^a + y_{iw}E_i^{tran} \\ & \qquad +(1 - y_{iw})t_{iw(\sim\delta_i)}P_i^a, \\ & E_d^{\delta_i} = \sum_{w=1}^{|A_i|} E_{dw}^{\delta_i},\ E_d^{\delta} = \sum_{i=1}^{M} E_d^{\delta_i}, \\ & \forall u \in [1, ..., N],\ \sum_{v=1}^{Q} x_{uv} = 1, \\ & E_d^{cpu} = \sum_{u=1}^{N}\sum_{v=1}^{Q} x_{uv}P_{f_v}\tfrac{c_u}{f_v}, \\ & \sum_{u=1}^{N} (\sum_{v=1}^{Q} x_{uv}\tfrac{c_u}{f_v} + t_u) = T \end{aligned} \tag{31}$$

Where $t_{iw(\sim\delta_i)}$ denotes the time duration when δ_i is not in use in A_{iw}, which is equal to the sum of the processor idle time and the execution time of tasks not using δ_i in A_{iw}. t_u is the processor idle time after τ_u.