ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Technische Informatik
und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Exam Spring 2020 – Sample solution

Embedded Systems

Prof. L. Thiele

First Name, Last Name:	
ETH Student Card Number:	

General information:

- Put your ETH student card on the desk.
- Write down your **First/Last Name** and your **ETH Student Card Number** on this cover page.
- Check that you have received all sheets of this examination paper (pages **1 - 24**).
- To answer the questions, you can use the white space between questions in this examination paper and/or additional sheets.
- Use a black or blue pen for writing your answers; no pencils, no red ink.
- For each additional sheet you use, write down on each sheet (upper right corner) your **First/Last name, ETH Student Card Number**, and the relevant **task number**. Start **each task** on a **new sheet of paper**, not only on a new page.
- **Read** each task **completely** before starting to write the answer.
- **Cross out completely and clearly** all invalid answers.
- At the end of the exam, **submit ALL your answer sheets together with this examination paper (pages 1 - 24)**.

For correction only. Leave blank!

Task	Max. Points	Achieved Points	Initials	Remarks
1	39			
2	40			
3	41			

Total:	120		**Grade:**

Task 1 : Real-Time Scheduling (39 points)

1.1: Short Questions (8 points)

Answer the following questions for the MSP-432 processor:

(a) (3 points) A system uses UART to transmit data, with the following configuration: the baud rate is 115 200 bits/s, 1 start bit, 2 stop bits, 7 data bits and 1 parity bit. How much time is needed to transmit 240 KB of data (1 KB = 1024 bytes)?

Sample solution:
Symbols transmitted with overhead: $\lceil 240\,\text{KB} \times 1\,024 \times 8 \times \frac{11}{7} \rceil = 3\,089\,555\,\text{bit}$
Time to transmit: $3\,089\,555\,\text{bit}/115\,200\,\frac{\text{bit}}{\text{s}} = 26.82\,\text{s}$.

(b) (2 points) How many bytes can be written in a block of memory accessed using byte-addresses `0x3000_0000` through `0x308F_FFFF`?

Sample solution:
`0x308F_FFFF` − `0x3000_0000` + 1 = `0x008F_FFFF` + 1 = `0x0090_0000`
There are 9×2^{20} addresses, which means 9 Megabytes of memory.

(c) (3 points) The following function returns the value that has been read from the appropriate GPIO port (pin 7 being the MSB).

```
uint8_t GPIO_getInputPortValue(uint_fast8_t selectedPort);
```

Pins 0 through 7 of GPIO port PORT1 are equipped with buttons with pull-up resistors (when the button is not pressed, the GPIO is connected to the supply voltage; when the button is pressed, the GPIO is connected to the ground). If only buttons connected to pins 2 and 7 of PORT1 are pressed, what value will variable `uint8_t kk` have after the following line of code is executed?

```
uint8_t kk = GPIO_getInputPortValue(PORT1) & 0x3C;
```

Hint: `&` is the logical AND operator.

Sample solution:
As pull-up resistors are used, a pushed button yields a 0, while a released one yields 1. We therefore have:

```
kk = 0b0111_1011 AND 0b0011_1100 = 0b0011_1000 = 0x38
```

1.2: Cyclic-Executive Scheduling (8 points)

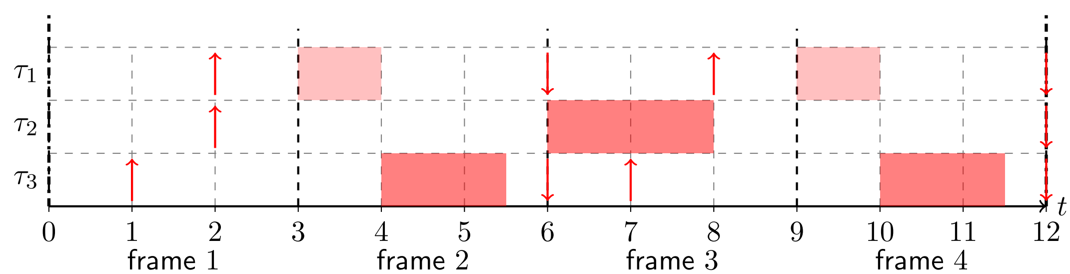
Cyclic-executive scheduling with a period $P = 12$ and a frame length $f = 3$ is used to schedule the task-set given in Table 1. Note that "frame 1" is the first frame of each period.

Table 1: A task set

Task	Period	Deadline	Phase	Execution Time	Frames
τ_1		4		1	2, 4
τ_2	12	10	2	2	
τ_3	6	5	1	1.5	

(a) (4 points) Determine one feasible assignment of tasks τ_2 and τ_3 to frames, construct the schedule for one period P and illustrate it graphically.

Sample solution:
One possible assignment is shown



(b) (4 points) Determine the period of task τ_1 and its minimal possible phase, if the task is executed in frames 2 and 4.

Sample solution:
The period of task τ_1 is 6, as it occurs twice in 12 time units. The minimal solution for the initial phase for task τ_1 is 2.

1.3: Rate Monotonic Scheduling (11 points)

A periodic task set is given in Table 5. Assume all phases are zero, and deadlines equal periods.

Table 2: A task set

	τ_{P1}	τ_{P2}	τ_{P3}	τ_{P4}
Period	5	7	11	13
Execution Time	1	2	3	3

(a) (2 points) Test if the given task set is schedulable under rate monotonic (RM) scheduling, using the *sufficient* test (the utilization bound test).

Sample solution:

The sufficient test:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} = \frac{1}{5} + \frac{2}{7} + \frac{3}{11} + \frac{3}{13} = 0.989$$

$$n \times (2^{\frac{1}{n}} - 1) = 4 \times (2^{0.25} - 1) = 4 \times (1.189 - 1) = 0.757$$

$$0.989 > 0.757 \qquad \ldots \text{INCONCLUSIVE}$$

Table 3: The task set, repeated

	τ_{P1}	τ_{P2}	τ_{P3}	τ_{P4}
Period	5	7	11	13
Execution Time	1	2	3	3

(b) (5 points) Test whether the given task set is schedulable under rate monotonic (RM) scheduling, using the *necessary and sufficient* test.

Sample solution:

τ_4:

$R_4^0 = C_4 = 3 \quad I_4^0 = \lceil\frac{3}{5}\rceil 1 + \lceil\frac{3}{7}\rceil 2 + \lceil\frac{3}{11}\rceil 3 = 1 + 2 + 3 = 6 \quad 6 + 3 \neq 3$

$R_4^1 = 6 + 3 = 9 \quad I_4^1 = \lceil\frac{9}{5}\rceil 1 + \lceil\frac{9}{7}\rceil 2 + \lceil\frac{9}{11}\rceil 3 = 2 + 4 + 3 = 9 \quad 9 + 3 \neq 9$

$R_4^2 = 9 + 3 = 12 \quad I_4^2 = \lceil\frac{12}{5}\rceil 1 + \lceil\frac{12}{7}\rceil 2 + \lceil\frac{12}{11}\rceil 3 = 3 + 4 + 6 = 13 \quad 13 + 3 \neq 12$

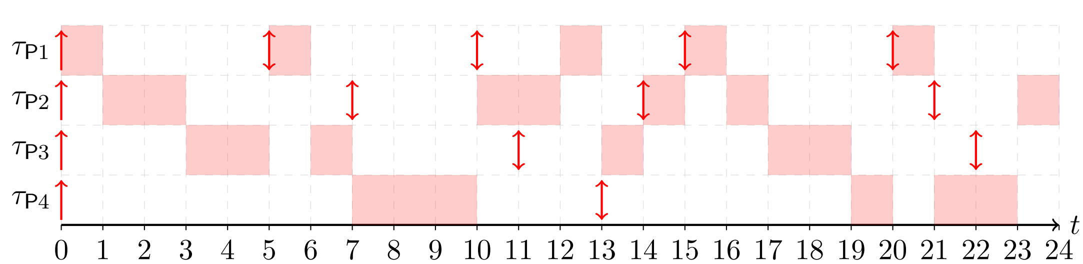
$R_4^3 = 13 + 3 = 16$ ... UNSCHEDULEABLE

Table 4: The task set, repeated

	τ_{P1}	τ_{P2}	τ_{P3}	τ_{P4}
Period	5	7	11	13
Execution Time	1	2	3	3

(c) (4 points) Using *earliest deadline first* (EDF) scheduling, construct a schedule from time 0 to time 24, and illustrate it graphically. Note if any task misses its deadline.

Sample solution:



1.4: Scheduling Mixed Tasks (12 points)

We have a task set with periodic and aperiodic tasks. The periodic tasks are τ_{P1}, τ_{P2} and τ_{P3}; the aperiodic tasks are τ_{A1}, τ_{A2}, and τ_{A3}. Additionally, task τ_{PS} is a polling server meant to service the aperiodic tasks. All tasks are specified in Table 5.

Deadline Monotonic scheduling is used to schedule the periodic tasks and the polling server, while the polling server services aperiodic tasks on a first-come-first-serve basis.

Assume task τ_{A1} arrives at time 1, task τ_{A2} at time 0 and task τ_{A3} at time 14.

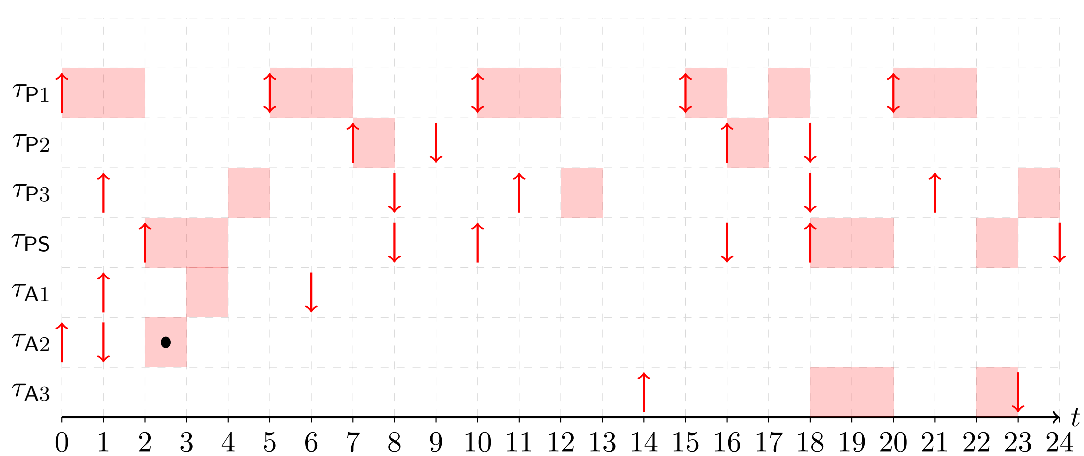
Table 5: Mixed task set

Task	Period	Phase	Deadline	Execution Time
τ_{P1}	5	0	5	2
τ_{P2}	9	7	2	1
τ_{P3}	10	1	7	1
τ_{PS}	8	2	6	3
τ_{A1}			5	1
τ_{A2}			1	1
τ_{A3}			9	3

Illustrate the schedule of all of the tasks graphically, including the polling server, from time 0 to time 24. Note if any task misses its deadline.

Sample solution:

The priorities are $\tau_{\text{P2}} > \tau_{\text{P1}} > \tau_{\text{PS}} > \tau_{\text{P3}}$. There is one deadline violation of the second aperiodic task.



Task 2 : Low Power Design (40 points)

2.1: Dynamic Voltage Scaling (10 points)

Consider a processor whose dynamic power dissipation when running with a clock frequency f in Hz, is given by $P_{\mathrm{dynamic}} = \left(\frac{f}{10^6\mathrm{Hz}}\right)^3 \mathrm{mW}$. It is assumed that the processor has negligible static and leakage power dissipation. Furthermore, its clock frequency can be freely selected from a continuous range of frequencies.

The processor must execute the following task set:

Task	τ_1	τ_2	τ_3
Arrival Time (ms)	0	3	5
Absolute Deadline (ms)	8	6	10
Cycles (x10^3)	8	12	2

Table 6: Task set.

Apply the *offline YDS* algorithm and plot the schedule of tasks and frequencies in Figure 1. Provide the steps of your solution in detail.

Sample solution:

Apply the offline YDS algorithm using intensity function $G[z_1, z_2] = \frac{\sum_{i \in V'} c_i}{z_2 - z_1}$.

Step 1:
All intervals:
$G[0,6] = 12/6 = 2$
$G[0,8] = (8+12)/8 = 2.5$
$G[0,10] = (8+12+2)/10 = 2.2$
$G[3,6] = 12/3 = 4$
$G[3,8] = 12/5 = 2.4$
$G[3,10] = (12+2)/7 = 2$
$G[5,10] = 2/5 = 0.4$
hence run τ_2 @ 4 MHz in [3,6]

Step 2:
New task set:
$\tau_1 : [0,5], C_1 = 8$
$\tau_3 : [3,7], C_3 = 2$
hence new intervals:
$G[0,5] = 8/5 = 1.6$
$G[0,7] = (8+2)/7 = 1.43$
$G[3,7] = 2/4 = 0.5$
hence, run τ_1 @ 1.6 MHz in [0,3] and [6,8]

Step 3:
New task set:
$\tau_3 : [0,2], C_3 = 2$
run τ_3 @ 1 MHz in [8,10]

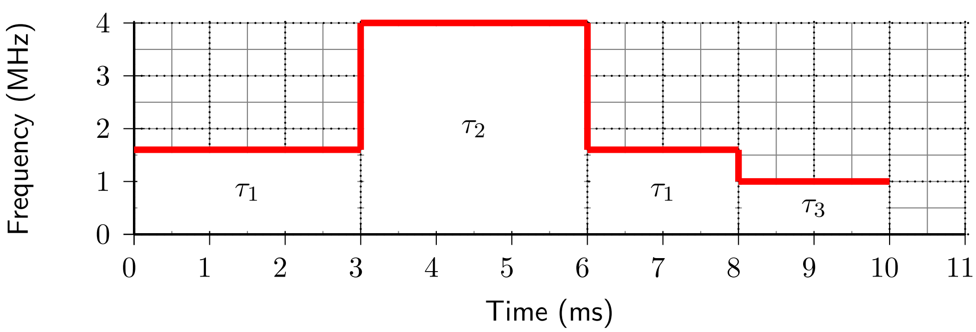


Figure 1: Offline YDS Schedule.

2.2: Dynamic Power Management (14 points)

Consider an embedded system with a processor that can execute in one of three modes, namely *HIGH*, *LOW*, and *SLEEP* mode. The valid transitions between the three execution modes, the power dissipation, and the processor frequency within each mode are summarized in Figure 2. The transition cost in terms of both time and energy are denoted by t_1, t_2 and E_1, E_2, respectively.

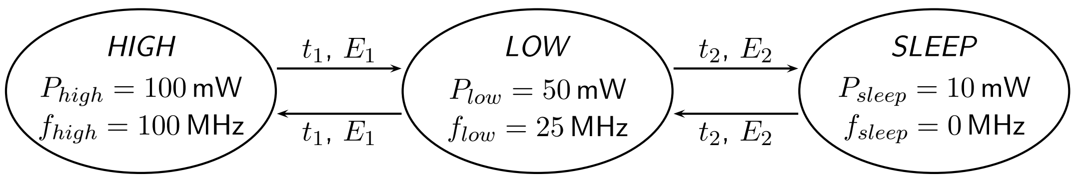


Figure 2: Processor execution modes.

The processor must schedule the following periodic real-time tasks:

Task	τ_1	τ_2	τ_3
Period (ms)	10	10	10
Arrival of First Task (ms)	0	4	7
Relative Deadline (ms)	3	8	10
Cycles ($\times 10^5$)	1	2	2

Table 7: Periodic task set.

(a) (4 points) Assume zero transition energy (i.e. $E_1 = E_2 = 0$) and instantaneous mode transitions (i.e. $t_1 = t_2 = 0$). Plot in Figure 3 a schedule that includes the power consumption over time, as well as tasks that are executing, using a workload-conserving scheduler that minimizes the average power. A workload-conserving scheduler always executes when a ready task is available.

Sample solution:
It can be shown that it is more energy efficient to execute in *HIGH* mode than in *LOW* mode (i.e. $\frac{P_{low}}{f_{low}} > \frac{P_{high}}{f_{high}}$). Since there is no energy overhead in transitioning from *HIGH* to *SLEEP* mode, it is more energy efficient to enter *SLEEP* mode when there are no tasks to execute. The energy efficient workload-conserving schedule is as follows:

□

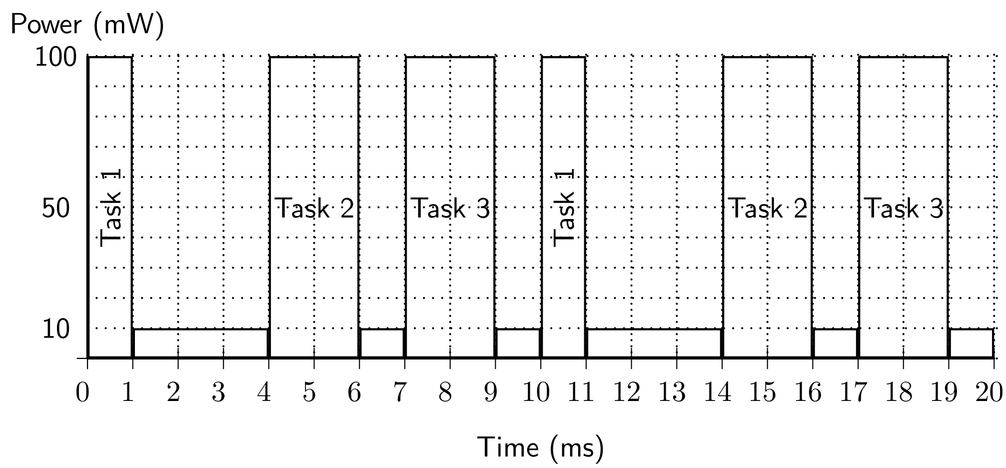


Figure 3: Energy efficient schedule using a workload-conserving scheduler.

For the following tasks assume the time and energy overhead between the execution modes as defined in Figure 2 to be $t_1 = 0.1$ ms, $t_2 = 0.9$ ms, and $E_1 = 10\,\mu$J, $E_2 = 40\,\mu$J, respectively.

(b) (5 points) Calculate the break-even time for the *HIGH* to *SLEEP* mode transitions.

Definition of break-even time: The minimum idle time required to compensate the cost of entering an inactive (sleep) state.

Sample solution:
The energy consumed without entering *SLEEP* mode is:
$E_h = T_w \cdot P_{high}$.

When entering the to *SLEEP* mode the total energy consumed during the idle time is:
$E_s = 2 \cdot (E_1 + E_2) + (T_w - 2 \cdot (t_1 + t_2)) \cdot P_{sleep}$

The break-even needs to satisfy:
$T_w \cdot P_{high} \geq 2 \cdot (E_1 + E_2) + (T_w - 2 \cdot (t_1 + t_2)) \cdot P_{sleep}$
$T_w \geq 2 \cdot \frac{E_1 + E_2 - (t_1 + t_2) \cdot P_{sleep}}{P_{high} - P_{sleep}}$
$T_w \geq 0.889$ ms

An additional constraint comes from the time needed to switch from *HIGH* to *SLEEP* mode and back, .i.e. $T_w \geq 2(t_1 + t_2) = 2$ ms.
The final break even time therefore is $T_w = 2.0$ ms.

□

(c) (5 points) Find a schedule (not necessarily workload conserving) that satisfies all task deadlines and minimizes the energy consumption. Plot the schedule in Figure 4 that includes the power consumption over time, as well as tasks that are executing. Assume constant power consumption during the transition between different execution modes.

Sample solution:
The schedule that minimizes the energy consumption removes the overhead associated with executing in *LOW* mode, in favor of executing longer in *SLEEP* mode. This is achieved by delaying the execution of task τ_2 by 2ms and τ_3 by 1ms, thus producing a slack of 5ms between the end of task τ_1 and the beginning of task τ_2. The power consumption between *HIGH* and *LOW* is 100mW, and the power consumption between *LOW* and *SLEEP* is 44 mW. The schedule, including the *HIGH* to *SLEEP* transition times, that minimizes the energy consumption is as follows:

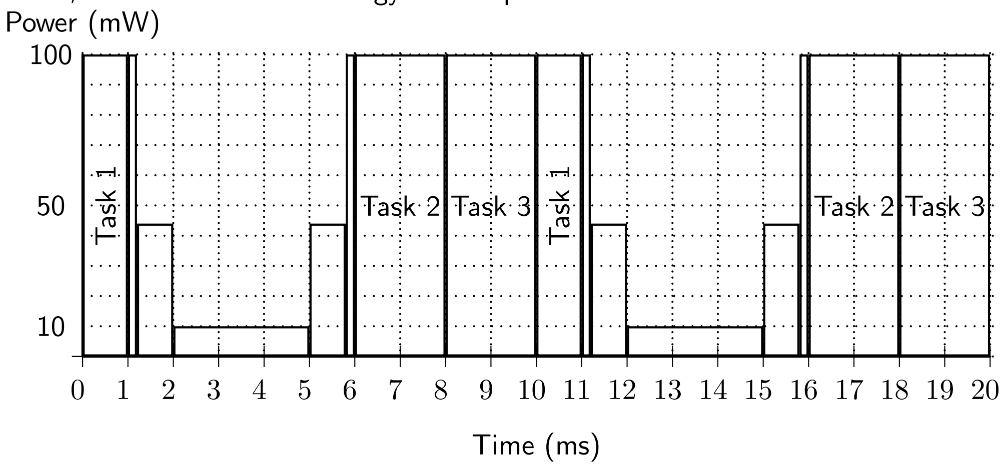


Figure 4: Energy minimizing schedule satisfying all task deadlines.

□

2.3: Solar Cells and Power Point Tracking (5 points)

We are given a solar cell with characteristics

$$I = G - 10^{-4} \times (\frac{U}{0.05})^3, \tag{1}$$

where I is the normalized current, U is the normalized voltage, and G is the relative solar irradiance.

Execute by hand the power point tracking algorithm as presented in the lecture for $G = 0.8$. Determine the voltage $U(k)$ and power $P(k)$ for $k = 0, \ldots, 5$ with $U(0) = 0.5$ and $U(1) = 0.55$, using the stepsize 0.05.

Sample solution:

k	0	1	2	3	4	5	...
$U(k)$	0.50	0.55	0.60	0.65	0.70	0.65	...
$P(k)$	0.350	0.367	0.376	0.377	0.368	0.377	...

Table 8: Values obtained by executing power point tracking algorithm.

2.4: Application Control (11 points)

We consider an application control scenario with the harvested energy in time interval $[t, t+1]$ of $p(t)$, used energy of $u(t)$ and battery capacity B. As the utility function we use $\mu(u) = \sqrt{u}$. The units of p, u, and B are Wh, the unit of t is h.

(a) (3 points) Suppose the energy is harvested by a solar panel, which generates $a(t) = 0.05\,\mathrm{Wh/cm^2}$ per hour in the daylight, which lasts 8 hours every day. During the rest of the day, the solar panel generates $a(t) = 0\,\mathrm{Wh/cm^2}$. For a solar panel with a size of S, the harvested energy function is $p(t) = a(t) \cdot S$. The system is equipped with a solar panel of size $S = 200\,\mathrm{cm^2}$. Is it possible for the system to continuously dissipate a constant power of $3\,\mathrm{W}$ for infinite days? If yes, what is the minimal battery size B in Wh to sustain this operation?

Sample solution:
Yes, it is possible. In 24h we harvest $8 \cdot 200 \cdot 0.05 = 80\,\mathrm{Wh}$ and we consume $24 \cdot 3\,\mathrm{Wh} = 72\,\mathrm{Wh}$.

During all daylight hours, the harvested energy is larger than the consumed energy. Therefore, the battery only needs to store energy for the night ($24 - 8 = 16\,\mathrm{h}$). Thus, the minimal battery size is $B = 16 \cdot 3\,\mathrm{Wh} = 48\,\mathrm{Wh}$

(b) (8 points) Suppose now that $B = 33\,\mathrm{Wh}$. The harvested energy function $p(t)$ is given in Figure 5. Determine an optimal energy usage function $u^*(t)$ that maximizes the utility and never leads to failure state. Draw $u^*(t)$ in Fig. 5.

Hint: All values of $u^*(t)$ are integers.

Sample solution:
We use the theorem explained in the lecture. The optimal energy use $u^*(t)$ is determined by satisfying the conditions in the theorem.

- When choosing u(t) for the whole day, then this appears to be the maximal minimum use energy: At b(17), the battery is full and at b(8) it is empty. Any further increase in u(t) = 3 will inevitably lead to an unfeasible solution.
- During the time interval with surplus energy ($t \in [9, 16]$)) we can increase the use function to u(t) = 4 without violating any feasibility constraint and still achieving a full battery at time 17. Red dots in Fig. 5 represent $u^*(t)$.

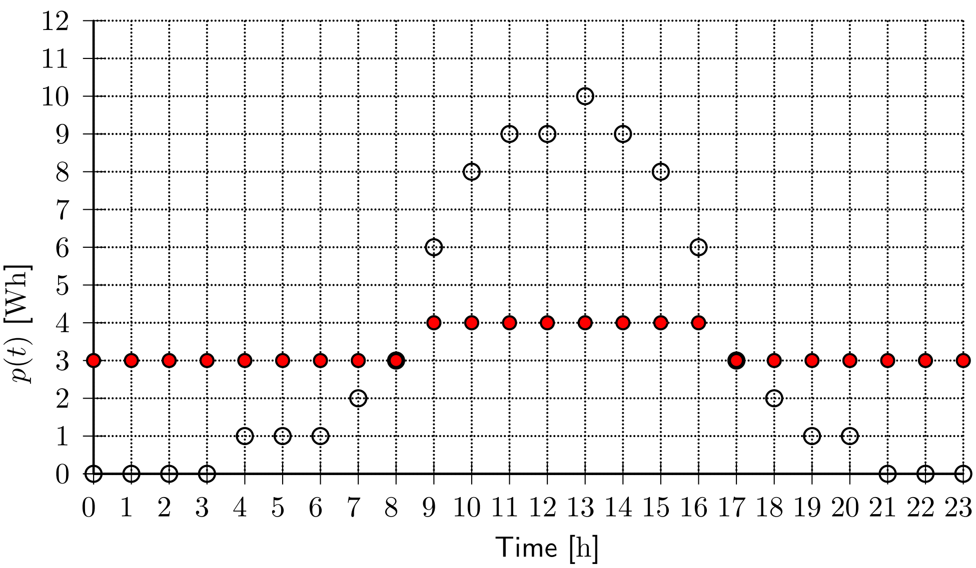


Figure 5: Application Control, Task (b)

Task 3 : Models and Architecture Synthesis (41 points)

3.1: Architecture Synthesis Fundamentals (9 points)

Mark the following statements as *true* or *false* and provide a one sentence explanation.

- (1 point) The "Stack Policy" as introduced in the lecture could be used in combination with EDF task scheduling

  Sample solution:
  ☒ True ☐ False
  Explanation: The "Stack Policy" can be used with dynamic priorities

  ☐

- (1 point) A dependence graph as defined in the lecture can represent parallelism in a program and can represent branches in control flow.

  Sample solution:
  ☐ True ☒ False
  Explanation: It does not represent branches in control flow.

  ☐

- (1 point) The throughput of an implementation of an iterative algorithm can be increased by decreasing the iteration interval.

  Sample solution:
  ☒ True ☐ False
  Explanation: Throughput is the inverse of the iteration interval

  ☐

- (1 point) A marked graph is designed to be implemented in software only.

  Sample solution:
  ☐ True ☒ False
  Explanation: A marked graph can be implemented in software and hardware.

  ☐

- (1 point) In a marked graph, a node with 2 input edges requires at least 2 tokens on any of the input edges to be activated.

  Sample solution:

  ☐ True ☒ False

  Explanation: It requires at least one token on each edge.

  □

- (2 points) Given two operations, v_1 and v_2, with execution time $w(v_1) = 2$ and $w(v_2) = 2$, respectively. "$\tau(v_2) - \tau(v_1) \leq 4$" models the following constraint : "v_2 must start not later than 2 time units after the end of v_1".

  Sample solution:

  ☒ True ☐ False

  Explanation: $\tau(v_1) + w(v_1) + 2 \geq \tau(v_2)$.

  □

- (1 point) Energy consumption can be improved by using pipelining instead of sequential processing of a given task set.

  Sample solution:

  ☒ True ☐ False

  □

- (1 point) List scheduling is an optimal algorithm for task scheduling with resource constraints.

  Sample solution:

  ☐ True ☒ False

  Explanation: It is not an optimal algorithm.

  □

3.2: LIST Scheduling (13 points)

Given the sequence graph in Figure 6.

(a) (8 points) Suppose that **one** multiplier and **one** adder are available as resources. Addition take one time unit for execution with the adder (r_1). Multiplication take two time units for execution with the multiplier (r_2). The first operation starts at $t = 0$ and the top node ('nop') is executed within zero time unit. The priority is assigned for each operation as maximal distance to the bottom node ('nop'). In case of equal priorities choose the node with the lowest index number. Fill out Table 9 using LIST scheduling algorithm. For a timestep t, the value $U_{t,k}$ denotes the set of operations that are ready to be scheduled on resource r_k (to be more specific, the set of operations that can be mapped on resource r_k and whose predecessors are all completed). $S_{t,k}$ denotes the set of operations that start at time t on resource r_k, while $T_{t,k}$ is the set of operations in execution at time t on resource r_k.

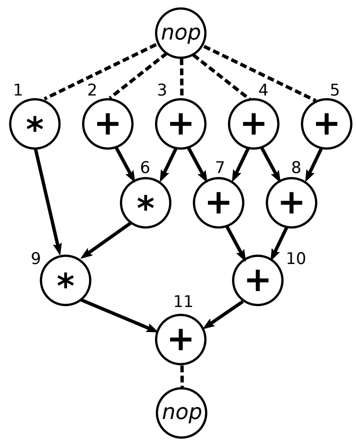


Figure 6: A sequence graph

t	k	$U_{t,k}$	$T_{t,k}$	$S_{t,k}$
0	r_1	$\nu_2, \nu_3, \nu_4, \nu_5$	—	ν_2
	r_2	ν_1	—	ν_1
1	r_1	ν_3, ν_4, ν_5	—	ν_3
	r_2	—	ν_1	—
2	r_1	ν_4, ν_5	—	ν_4
	r_2	ν_6	—	ν_6
3	r_1	ν_5, ν_7	—	ν_5
	r_2	—	ν_6	—
4	r_1	ν_7, ν_8	—	ν_7
	r_2	ν_9	—	ν_9
5	r_1	ν_8	—	ν_8
	r_2	—	ν_9	—
6	r_1	ν_{10}	—	ν_{10}
	r_2	—	—	—
7	r_1	ν_{11}	—	ν_{11}
	r_2	—	—	—
8	r_1	—	—	—
	r_2	—	—	—
9	r_1	—	—	—
	r_2	—	—	—
10	r_1	—	—	—
	r_2	—	—	—
11	r_1	—	—	—
	r_2	—	—	—
12	r_1	—	—	—
	r_2	—	—	—
13	r_1	—	—	—
	r_2	—	—	—

Table 9: Table for subtask (a)

(b) (1 point) What is the resulting latency?

Sample solution:
$L = 8$.

□

(c) (2 points) Suppose it is allowed to add either one adder or one multiplier, which resource should be added to shorten the latency? What is the corresponding latency?

Sample solution:
Adder because the critical paths (e.g. $5 \to 8 \to 10 \to 11$) are delayed by adders, not multipliers. The corresponding is $L = 7$.

□

(d) (2 points) In case of unlimited hardware resource, what is the minimized latency? Explain why.

Sample solution:
$L = 6$. With infinite adders and multipliers, the critical paths $(2 \rightarrow 6 \rightarrow 9 \rightarrow 11)$ are now delayed by multipliers.

□

□

3.3: Iterative Algorithms (19 points)

Consider the marked graph G_M in Figure 7. The nodes labeled with **+**, **x2**, **+4** represent addition, multiplication by 2, and addition with 4, respectively. f_1, f_3 and f_4 need **1** time unit each, f_2 needs **2** time units. The input u is a sequence of numbers, with $u(k)$ representing the k-th number.

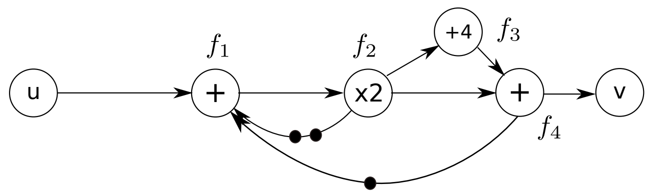


Figure 7: Marked graph G_M

(a) (7 points) Determine the output value $v(k)$ corresponding to the graph in Figure 7 as a function of the input values $u(\cdot)$ and previous output values $v(\cdot)$. Assume $k > 2$.

Sample solution:

$$\left.\begin{aligned} f_1(k) &= u(k) + f_2(k-2) + f_4(k-1) \\ f_2(k) &= 2 \cdot f_1(k) \\ f_3(k) &= f_2(k) + 4 = 2 \cdot f_1(k) + 4 \\ f_4(k) &= f_2(k) + f_3(k) = 2 \cdot f_2(k) + 4 = 4 \cdot f_1(k) + 4 \\ v(k) &= f_4(k) \end{aligned}\right\}$$

$\Rightarrow \quad f_1(k) = \frac{f_4(k)-4}{4}$
$\Rightarrow \quad f_2(k) = \frac{f_4(k)-4}{2}$
$\Rightarrow \quad \frac{f_4(k)-4}{4} = u(k) + \frac{f_4(k-2)-4}{2} + f_4(k-1)$
$\Rightarrow \quad v(k) = 4 \cdot u(k) + 2 \cdot v(k-2) + 4 \cdot v(k-1) - 4$

□

(b) (4 points) Suppose that we implement the algorithm using functional pipelining. Express all constraints which stem from the data dependencies in G_S, considering also the data dependencies among successive iterations.

Hint: Determine constraints of the form $\tau(f_j) - \tau(f_i) \geq w(f_i) - d_{ij} \cdot P$, $\forall\, (f_i, f_j) \in E_S$, where P is the iteration interval of the pipelined implementation, $w(f_i)$ denotes the execution time of f_i, and d_{ij} represents the index displacement associated with edge (f_i, f_j).

Sample solution:
Data dependency constraints:

$$\begin{aligned}
\tau(f_2) - \tau(f_1) \geq &\quad 1 && (1)\\
\tau(f_3) - \tau(f_2) \geq &\quad 2 && (2)\\
\tau(f_4) - \tau(f_2) \geq &\quad 2 && (3)\\
\tau(f_4) - \tau(f_3) \geq &\quad 1 && (4)\\
\tau(f_1) - \tau(f_2) \geq &\quad 2 - 2P && (5)\\
\tau(f_1) - \tau(f_4) \geq &\quad 1 - 1P && (6)
\end{aligned}$$

□

(c) (3 points) Assuming unlimited resources, determine the smallest feasible iteration interval P_{min}. Justify your solution.

Sample solution:
Based on the previous system of inequalities:

(1)+(3)+(6)	$\implies 0 \geq 1 + 2 + 1 - P$	$\implies P \geq 4$
(1)+(2)+(4)+(6)	$\implies 0 \geq 1 + 2 + 1 - P$	$\implies P \geq 5$
(1)+(5)	$\implies 0 \geq 1 + 2 - 2P$	$\implies P \geq \frac{3}{2}$

Hence, $P_{min} = 5$.

□

(d) (3 points) Now assume that, there are only **one** multiplier to compute f2 and **one** adder to compute f1,f3 or f4 available. First, depict a pipelined scheduling in Figure 8 under this resource constraint with a predefined minimal iteration interval $P = 5$. Then, indicate the latency L of the schedule. Draw **three** consecutive iterations, and mark different iterations clearly, e.g. with different colors, different textures, or different numbers.

$L = 5$

□

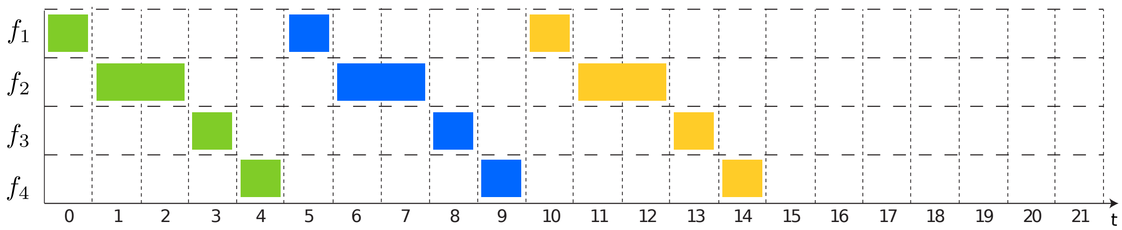


Figure 8: Pipelined Scheduling with resource constraints

(e) (2 points) Can the latency of a schedule given the marked graph be decreased by using an unlimited number of adders and multipliers? Explain why?

No, because the outputs of the adders depend on each other.

□