Embedded electronic engineering A: Real-time systems seminar

Multi-core partitioned scheduling

By Shahzaib Waseem

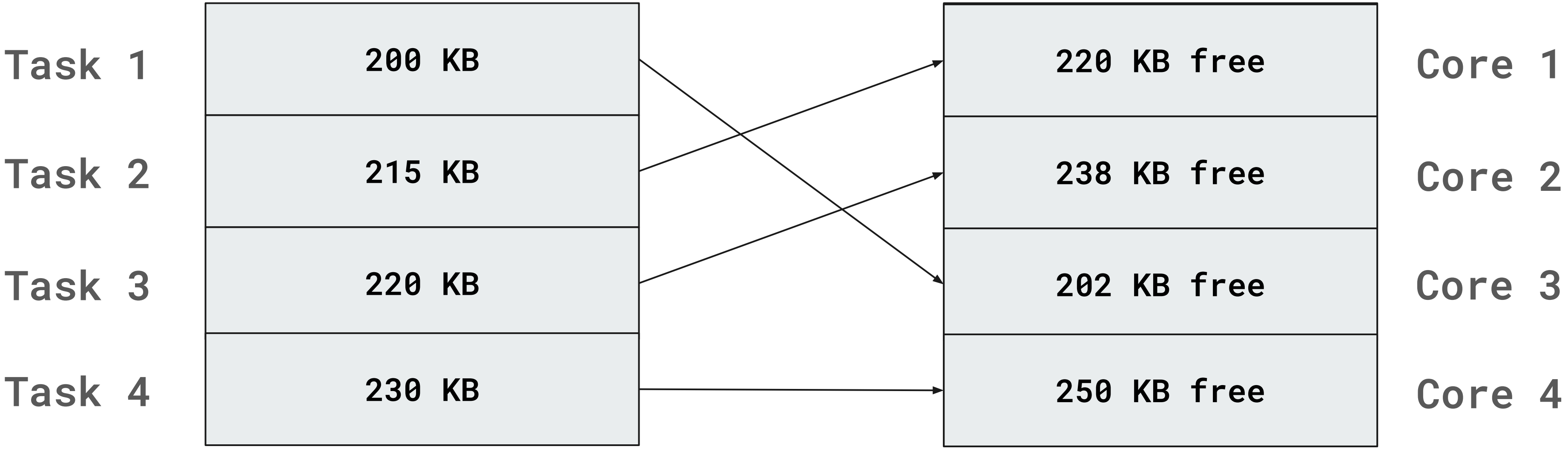
Agenda

- Motivation
- Task allocation
- Partitioned EDF
- Uppaal implementation
- Remarks

Motivation

- The switch from single-core to multi-core systems
  - Better task scheduling
  - Lesser task starvation
- Partitioned scheduling implies:
  - Permanent task assignment
  - Each core has its own ready-queue and scheduling algorithm

Task allocation: best fit example



Task 1 200 KB

Task 2 215 KB

Task 3 220 KB

Task 4 230 KB

220 KB free Core 1

238 KB free Core 2

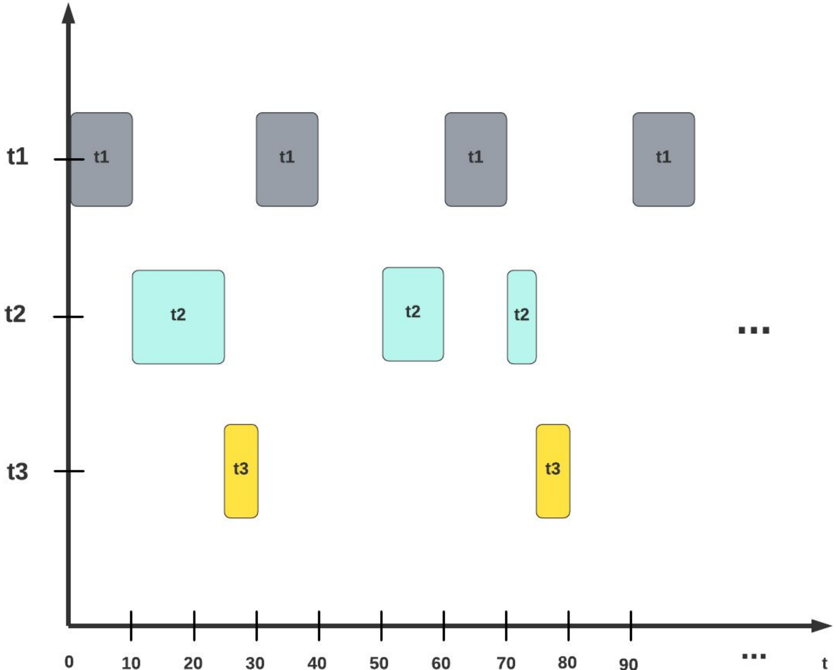
202 KB free Core 3

250 KB free Core 4

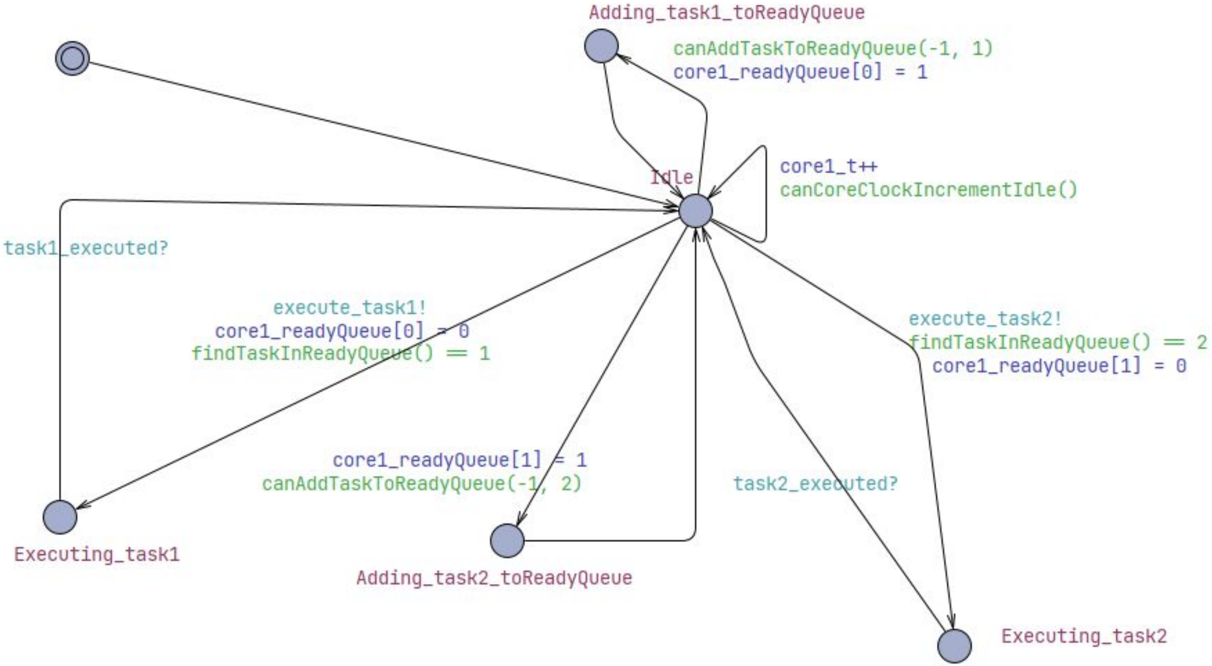
Partitioned EDF

- Each core runs EDF scheduling algorithm
- Earliest deadline first
- Preemptive scheduling
- Higher priority = earlier deadline task

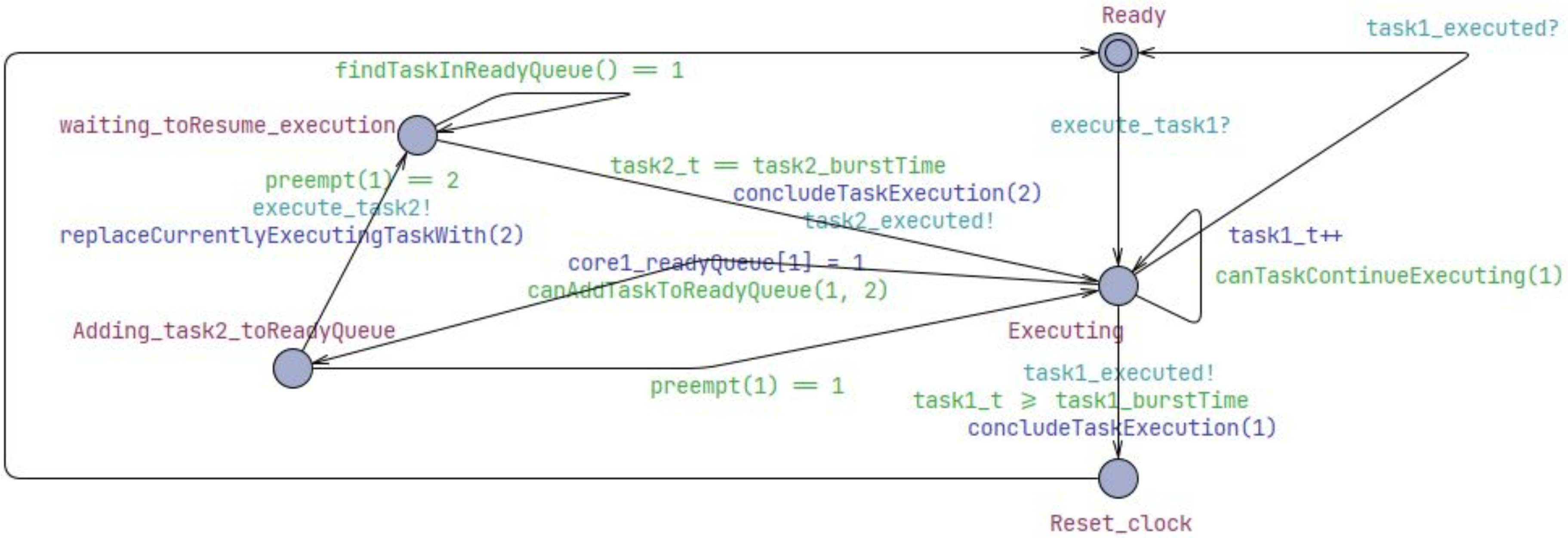
Partitioned EDF



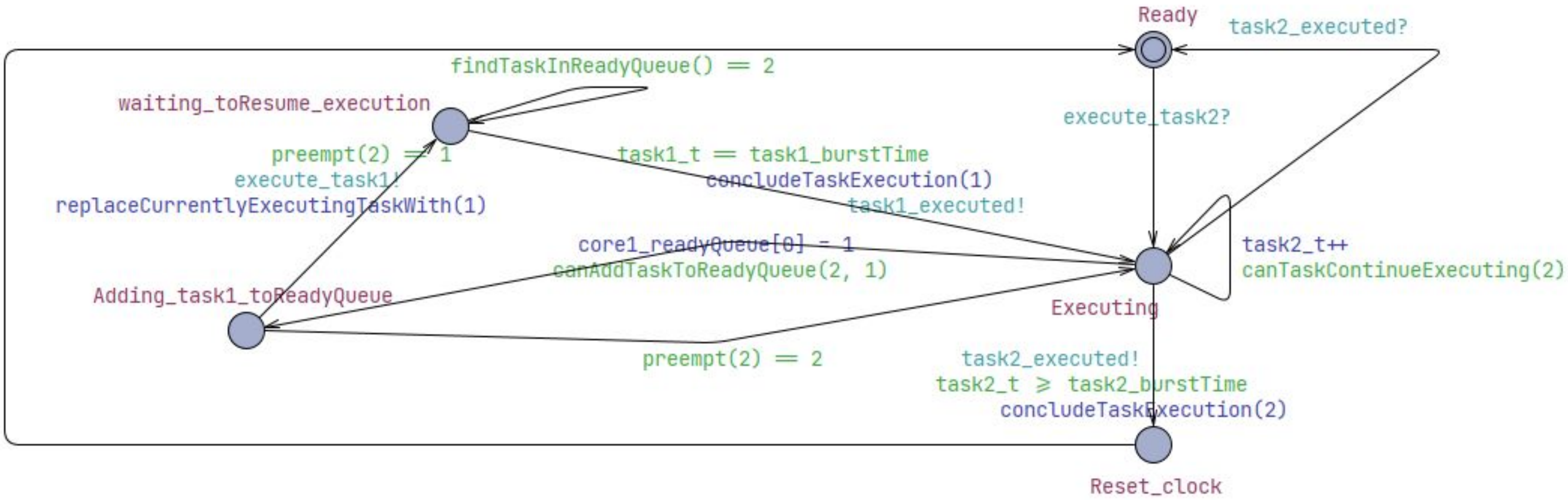
Uppaal implementation: Core



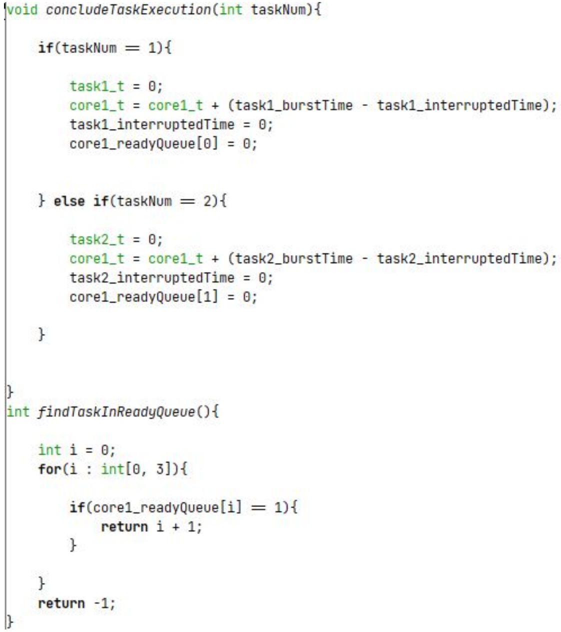
Uppaal implementation: Task 1


Ready
task1_executed?
findTaskInReadyQueue() == 1
waiting_toResume_execution
execute_task1?
task2_t == task2_burstTime
concludeTaskExecution(2)
task2_executed!
preempt(1) == 2
execute_task2!
replaceCurrentlyExecutingTaskWith(2)
task1_t++
core1_readyQueue[1] = 1
canAddTaskToReadyQueue(1, 2)
canTaskContinueExecuting(1)
Adding_task2_toReadyQueue
Executing
task1_executed!
preempt(1) == 1
task1_t ≥ task1_burstTime
concludeTaskExecution(1)
Reset_clock

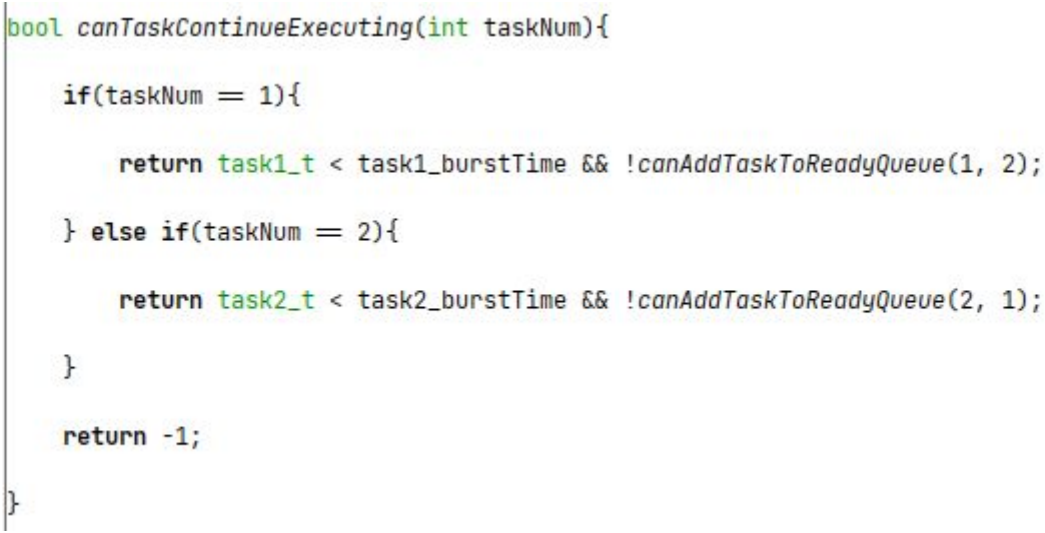
Uppaal implementation: Task 2


Ready
task2_executed?
findTaskInReadyQueue() == 2
waiting_toResume_execution
execute_task2?
preempt(2) == 1
execute_task1!
replaceCurrentlyExecutingTaskWith(1)
task1_t == task1_burstTime
concludeTaskExecution(1)
task1_executed!
core1_readyQueue[0] = 1
canAddTaskToReadyQueue(2, 1)
task2_t++
canTaskContinueExecuting(2)
Adding_task1_toReadyQueue
Executing
preempt(2) == 2
task2_executed!
task2_t ≥ task2_burstTime
concludeTaskExecution(2)
Reset_clock

Uppaal implementation: code

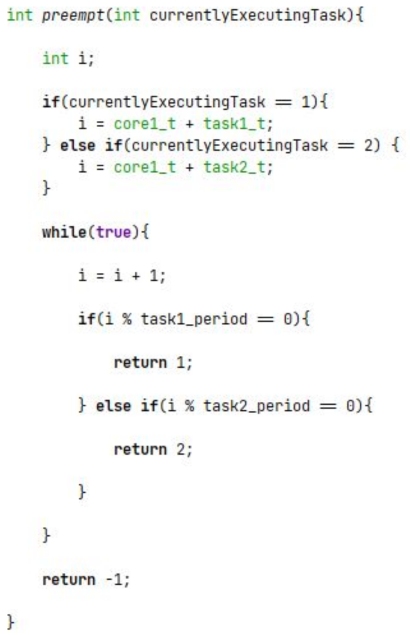


```
void concludeTaskExecution(int taskNum){

    if(taskNum == 1){

        task1_t = 0;
        core1_t = core1_t + (task1_burstTime - task1_interruptedTime);
        task1_interruptedTime = 0;
        core1_readyQueue[0] = 0;


    } else if(taskNum == 2){

        task2_t = 0;
        core1_t = core1_t + (task2_burstTime - task2_interruptedTime);
        task2_interruptedTime = 0;
        core1_readyQueue[1] = 0;

    }


}
int findTaskInReadyQueue(){

    int i = 0;
    for(i : int[0, 3]){

        if(core1_readyQueue[i] == 1){
            return i + 1;
        }

    }
    return -1;
}
```

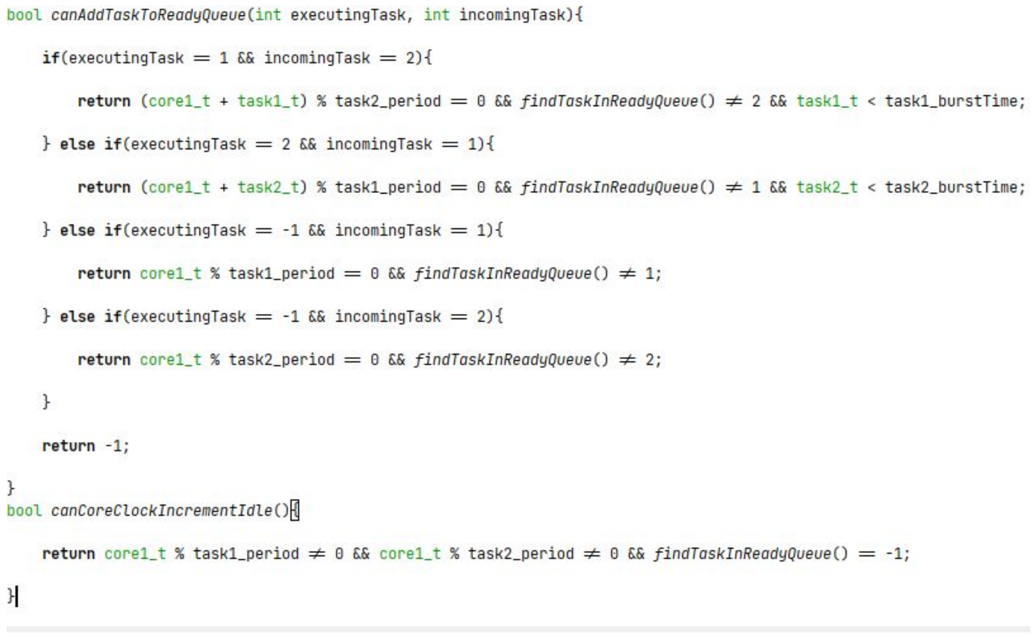


```
bool canTaskContinueExecuting(int taskNum){

    if(taskNum == 1){

        return task1_t < task1_burstTime && !canAddTaskToReadyQueue(1, 2);

    } else if(taskNum == 2){

        return task2_t < task2_burstTime && !canAddTaskToReadyQueue(2, 1);

    }

    return -1;

}
```

Uppaal implementation: code

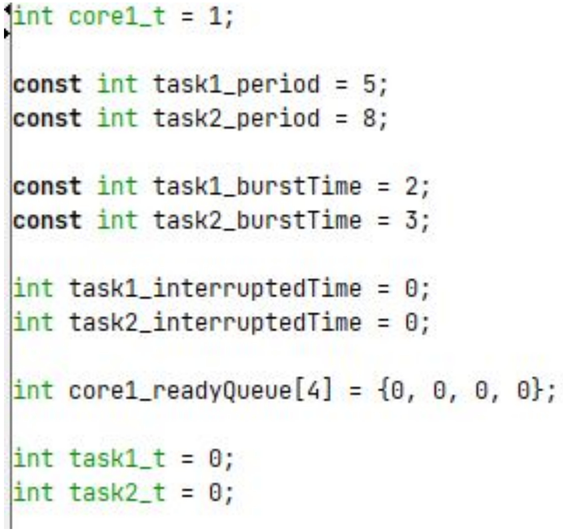


```
int preempt(int currentlyExecutingTask){

    int i;

    if(currentlyExecutingTask == 1){
        i = core1_t + task1_t;
    } else if(currentlyExecutingTask == 2) {
        i = core1_t + task2_t;
    }

    while(true){

        i = i + 1;

        if(i % task1_period == 0){

            return 1;

        } else if(i % task2_period == 0){

            return 2;

        }

    }

    return -1;

}
```

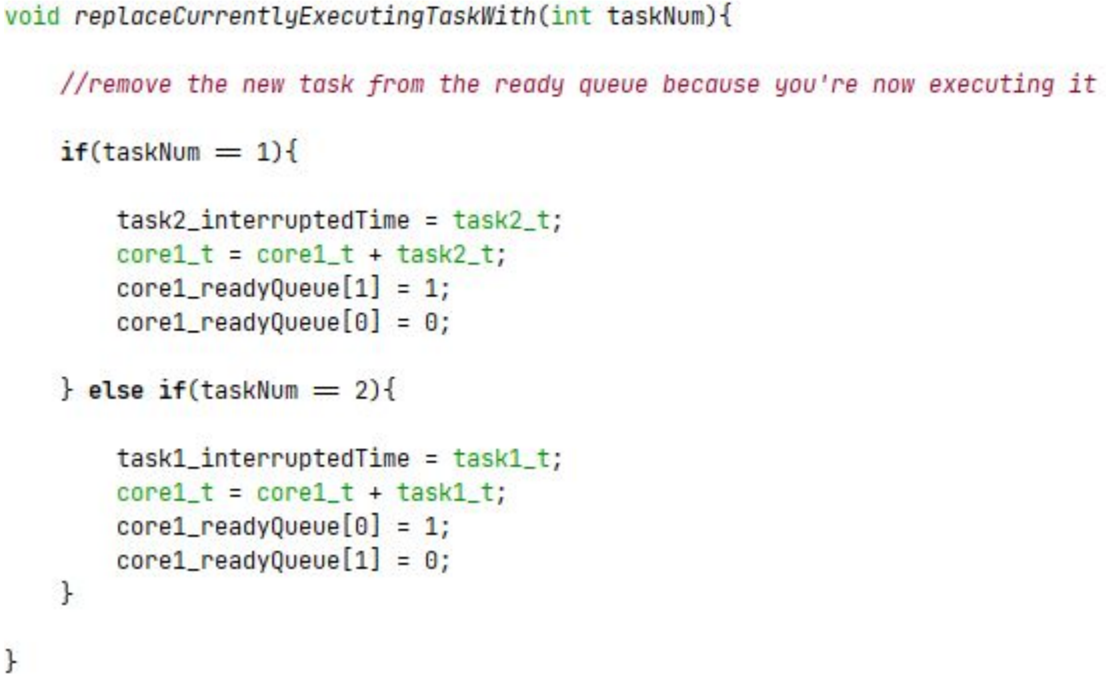


```
bool canAddTaskToReadyQueue(int executingTask, int incomingTask){

    if(executingTask == 1 && incomingTask == 2){

        return (core1_t + task1_t) % task2_period == 0 && findTaskInReadyQueue() != 2 && task1_t < task1_burstTime;

    } else if(executingTask == 2 && incomingTask == 1){

        return (core1_t + task2_t) % task1_period == 0 && findTaskInReadyQueue() != 1 && task2_t < task2_burstTime;

    } else if(executingTask == -1 && incomingTask == 1){

        return core1_t % task1_period == 0 && findTaskInReadyQueue() != 1;

    } else if(executingTask == -1 && incomingTask == 2){

        return core1_t % task2_period == 0 && findTaskInReadyQueue() != 2;

    }

    return -1;

}
bool canCoreClockIncrementIdle(){

    return core1_t % task1_period != 0 && core1_t % task2_period != 0 && findTaskInReadyQueue() == -1;

}
```

Uppaal implementation: code



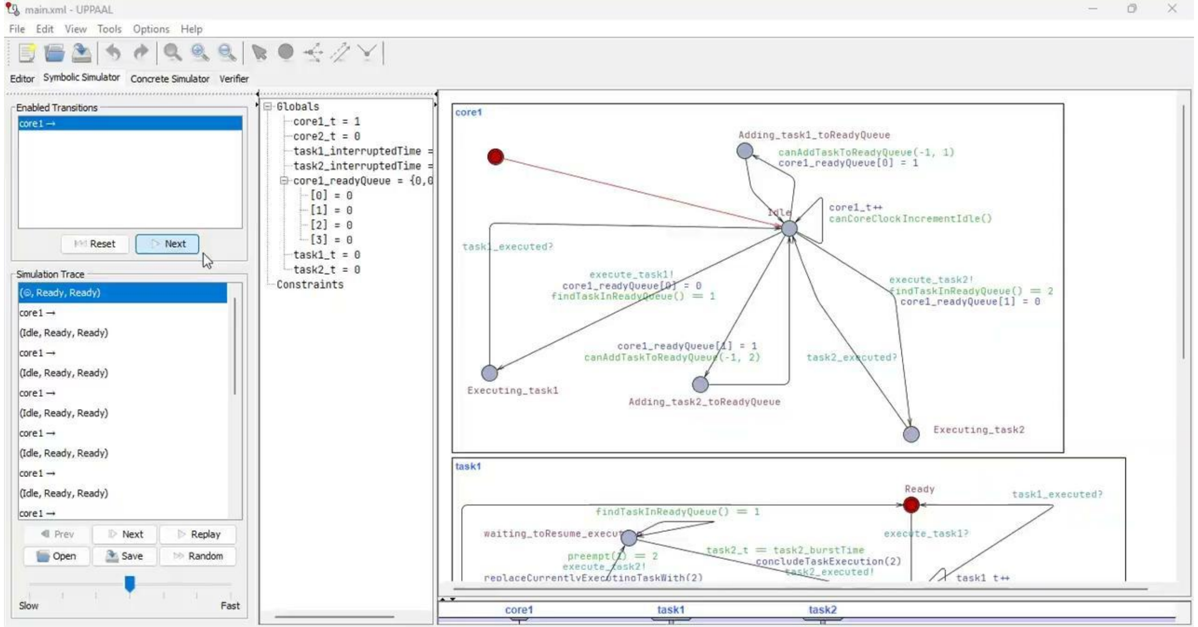
```
int core1_t = 1;

const int task1_period = 5;
const int task2_period = 8;

const int task1_burstTime = 2;
const int task2_burstTime = 3;

int task1_interruptedTime = 0;
int task2_interruptedTime = 0;

int core1_readyQueue[4] = {0, 0, 0, 0};

int task1_t = 0;
int task2_t = 0;
```



```
void replaceCurrentlyExecutingTaskWith(int taskNum){

    //remove the new task from the ready queue because you're now executing it

    if(taskNum == 1){

        task2_interruptedTime = task2_t;
        core1_t = core1_t + task2_t;
        core1_readyQueue[1] = 1;
        core1_readyQueue[0] = 0;

    } else if(taskNum == 2){

        task1_interruptedTime = task1_t;
        core1_t = core1_t + task1_t;
        core1_readyQueue[0] = 1;
        core1_readyQueue[1] = 0;
    }

}
```

Uppaal implementation: simulation

	Task 1	Task 2
Period	5	8
Burst Time	2	3



```
Globals
  core1_t = 1
  core2_t = 0
  task1_interruptedTime =
  task2_interruptedTime =
  core1_readyQueue = {0,0
    [0] = 0
    [1] = 0
    [2] = 0
    [3] = 0
  task1_t = 0
  task2_t = 0
Constraints
```

```
(⊚, Ready, Ready)
core1 →
(Idle, Ready, Ready)
core1 →
(Idle, Ready, Ready)
core1 →
(Idle, Ready, Ready)
core1 →
(Idle, Ready, Ready)
core1 →
(Idle, Ready, Ready)
core1 →
```

Remarks

- Potential improvements:
    - Using clocks and invariants
    - Handling more than 2 tasks