*E x* *p* *e* *r* *i* *m* *e* *n* *t* *1*

Introduction to Cheddar

Objectives:

In this experiment you will learn how to:

* Use Cheddar Software for simulating scheduling Algorithms.
* Analyse various priority driven Algorithms

**TABLE OF CONTENTS**

* 1. Objective
  2. References
  3. Theoretical Background
  4. Introduction to Cheddar

1.4.1 Starting the simulator

1.4.2 Building a new Project

1.4.3 Running the simulator

* 1. In-Lab Experimental Procedure
     1. Analyze the Scheduling Algorithms like EDF, RMA etc
  2. Result

Appendix A: Pre-Lab Report

Appendix B: In-Lab Report

* 1. Objective

This experiment aims to highlight various features of Cheddar simulator and the procedure for using the tool for analyzing the scheduling algorithms.

* 1. References

1. Liu, Jane W.S., Real Time Systems, Pearson Education, 2000
2. Cheddae Release 3.x user’s guide
   1. Theoretical Background

A priority-driven scheduler is an on-line scheduler. Most commonly supported approach in RTOS. In a priority-driven system, ready jobs are assigned to processors according to their relative priorities.

In an event driven scheduling, the jobs are placed in one or more queues and at each event the ready job with highest priority is executed.

It is also called list scheduling it contains list of priorities assigned to each jobs.

Priorities are assigned to the jobs based on their criticality. It does not pre-compute a

schedule of tasks/jobs: instead assigns priorities to jobs when released, places them on a

run queue in priority order. When pre-emption is allowed, a scheduling decision is made

whenever a job is released or completed. At each scheduling decision time, the scheduler

updates the run queues and executes the job at the head of the queue. Jobs ready for

execution are placed in one or more queues ordered by priorities of the jobs.At any

scheduling decision time, the jobs with the highest priorities are scheduled and executed

on the available processors.

* 1. .Introduction to Cheddar

Cheddar is a free real time scheduling framework. Cheddar is designed for checking task temporal constraints of a real time application/system. It can also helps for quick prototyping of real time schedulers.

The Cheddar project was started in 2002 by Frank Singhoff, Lab-STICC Team, University of Brest. Since 2008, Ellidiss technologies also contributes to the development of Cheddar and provides industrial support.

1.4.1 Starting the simulator

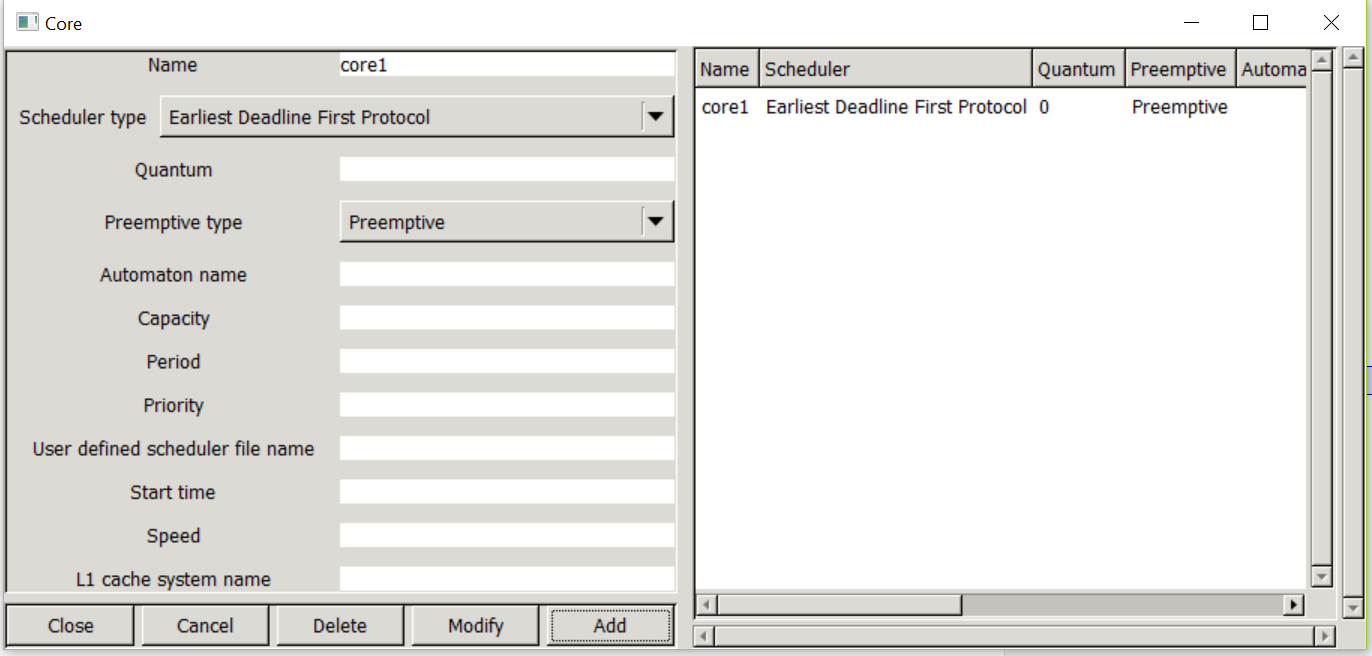
Double click on the cheddar.exe from the ‘Cheddar-3.0-win32-bin’ folder. This will

launch the application.

## 1.4.2 Building a new project:

1. From the file icon in the menu bar, choose ‘new’ XML project. All newly created files have to be saved in the current path with an extension of .sc.
2. **Adding a core:**

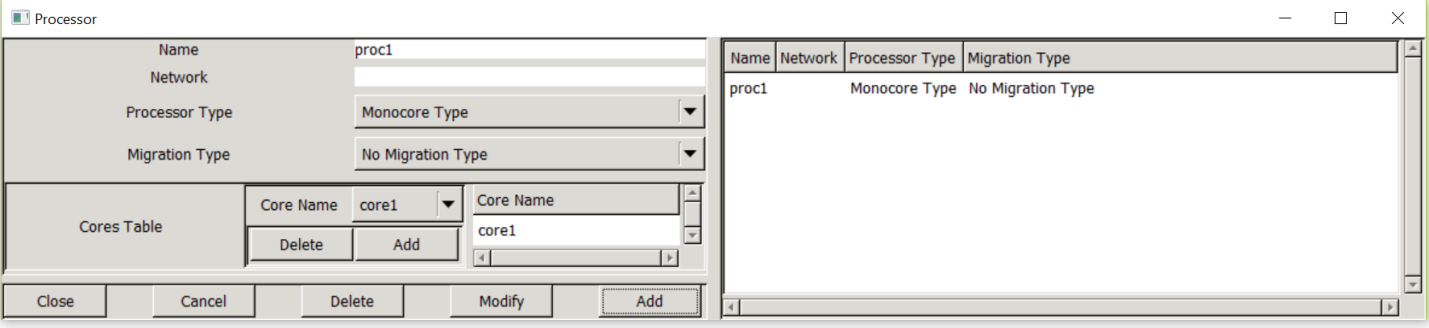
* From the menu bar, Edit -> hardware -> Core.



* Add the name of the core. For example, Core1, as shown in the figure.
* Scheduler type: Add the type of scheduling algorithm
* Preemptive type: Has 2 options:-preemptive and non-preemptive. To be chosen according to the requirement.
* Add the core by selecting the add option. The core name along with the parameters will be reflected on the right hand side.
* Modify the core parameters by selecting the core name and changing the parameter.
* To delete the core by choosing delete.

**Adding the Processor:**

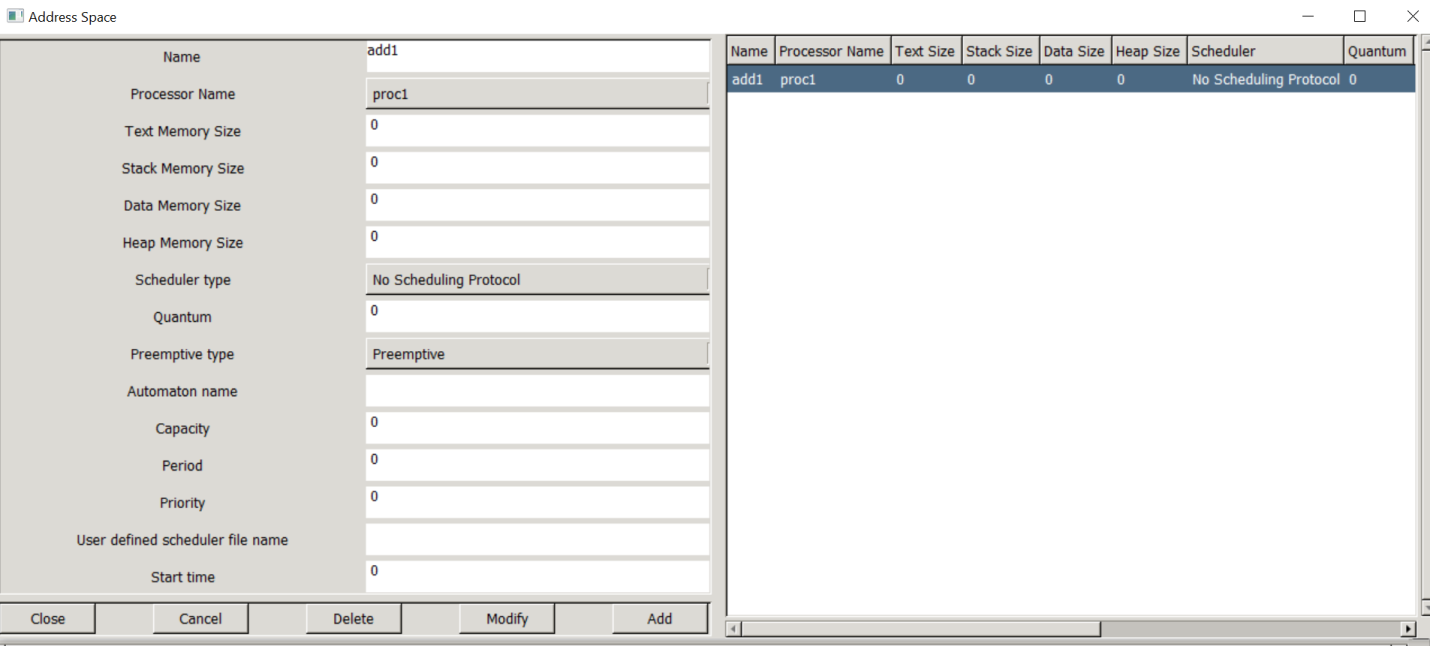
* From the menu bar, Edit -> Hardware -> Processor.



* Add the name of the processor.
* Processor type:
* Mono-core processor- Single core processor system.
* Identical multicores- Multicore system with identical core parameters, that is, same architecture and same speed.
* Uniform multicores- Same Architecture but different speeds.
* Unrelated multicores- Different characteristics.
* Migration Type:
* No migration: The tasks will not be migrated from one processor to another.
* Job level migration: The various instances of the task can be migrated across various processors.
* Time unit migration: All instances of the task will be executed on the same processor.
* Add the core from the ‘cores table’.
* The Processor name along with the other parameters will be reflected on the right hand side.

**Adding the address space:**

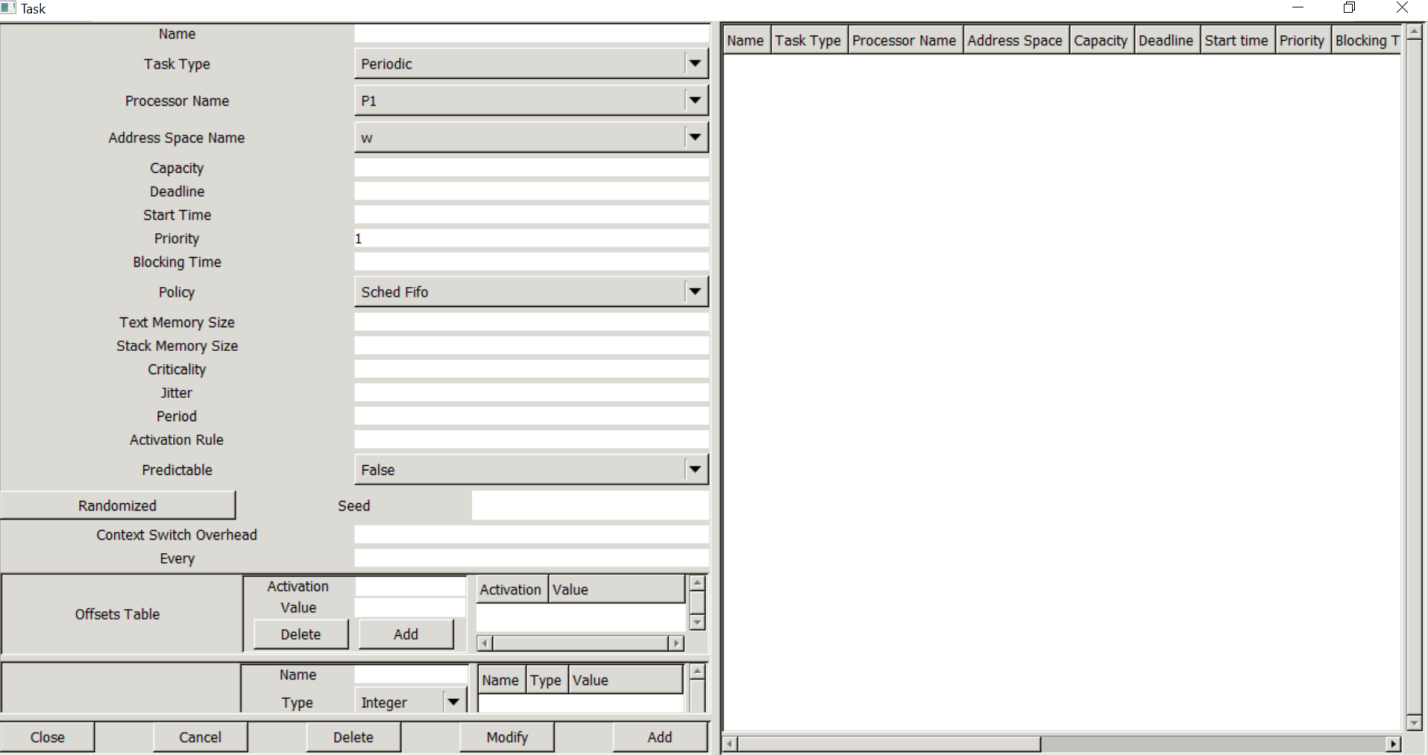
* From the menu bar, Edit -> software -> Address space.



* Select the name for the address space.
* Processor name: Choose the processor. All the defined processors will appear in the drop down list.
* Scheduler type: An address space is a piece of memory which contains the processor parameters such as tasks and resources. A scheduler type can be defined for the address spaces in case of hierarchical scheduling algorithms, otherwise choose No Scheduling Protocol.
* Preemptive type: opt for non-preemptive scheduling in the address space.
* Add the address space.

**Adding a task:**

* From the menu bar, Edit -> software -> task.



* Add the name of the task.
* Task type:
* Periodic task: The task will occur periodically after the time unit defined in the period field.
* Aperiodic task: Tasks that will occur only once with soft deadline.
* Sporadic task: Tasks that will occur only once with a hard deadline.
* Processor name: Drop down list contains all the processors defined. Choose the appropriate processor for the task.
* Capacity: Worst case Execution time of the task.
* Deadline: Absolute deadline. Refers to the completion time for the task.
* Start time: The time offset for the task, that is, the time at which the task will be ready for execution.

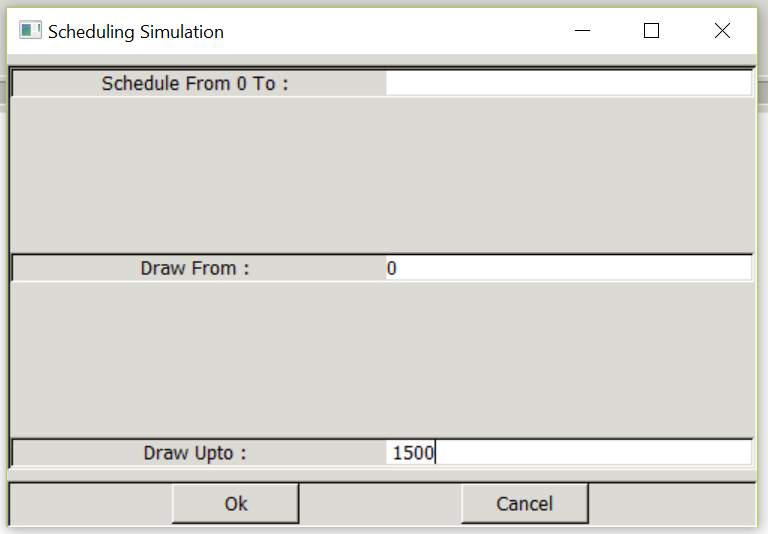
Priority: To define the priority of the task. For user defined priority, use POSIX scheduling (Highest priority first scheduling, Fixed priority). There are a total of 255 priority levels available. By default, 1- lowest priority and 255- highest priority. When 2 or more tasks have the same priority, the scheduler will execute the task depending upon the scheduling policy.

In case of other scheduling algorithms, priority is determined by the scheduler depending upon the task parameters such period, deadline, etc.

* Period: The periodicity of the task, that is, the time after which the task will reoccur. (note: Deadline must be lesser than the periodicity). Do not specify periodicity for aperiodic and sporadic task.

## 1.4.3 Running the Simulation:

* To run the simulation, click on
* The following dialog box will appear, schedule from 0 to hyper-period.



* This will create a .xml file (eg: file1.xml) which will be present in the same path as .sc file.
* To clear the workspace, click on
* To set different colors to the task, From the menu bar choose, Tools->Scheduling->Scheduling options -> Display scheduling->Several colors for time tines.



* To check the schedulability feasibility tests, click on
  1. In-Lab Experimental Procedure

Consider the following task set. Scheduling algorithm: EDF

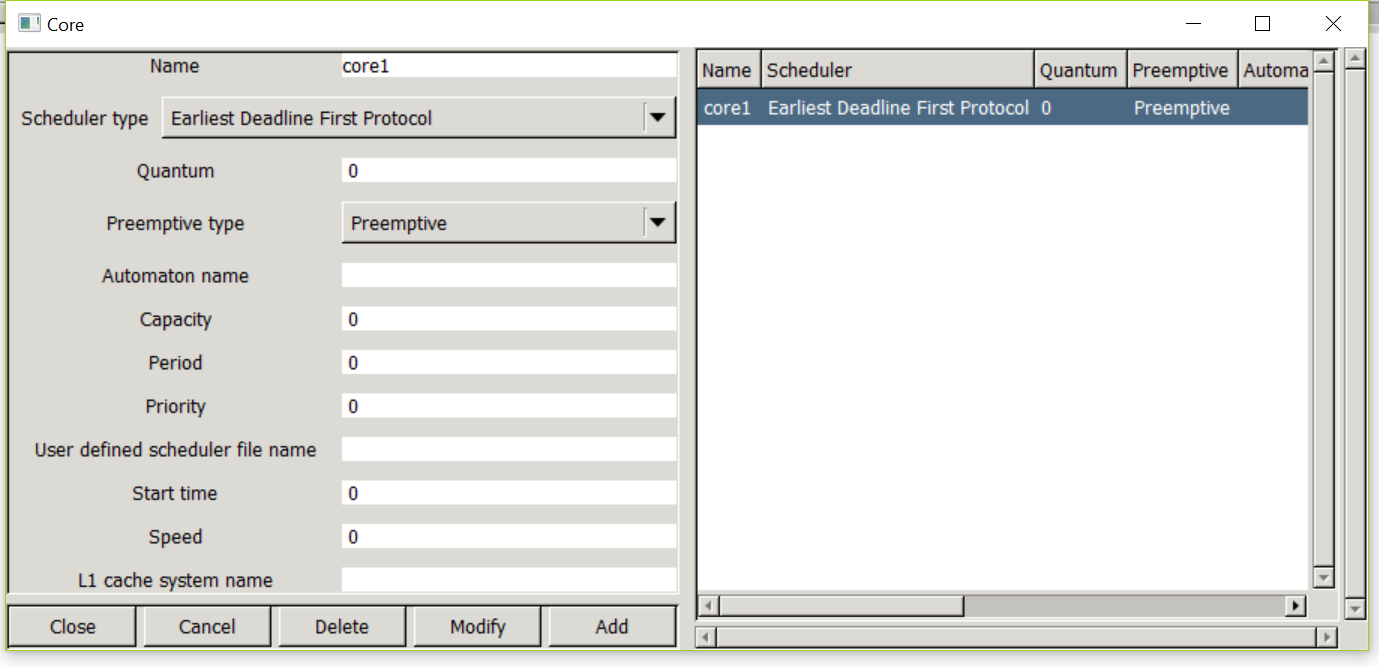
T1 = (1,4)

T2 = (2,5)

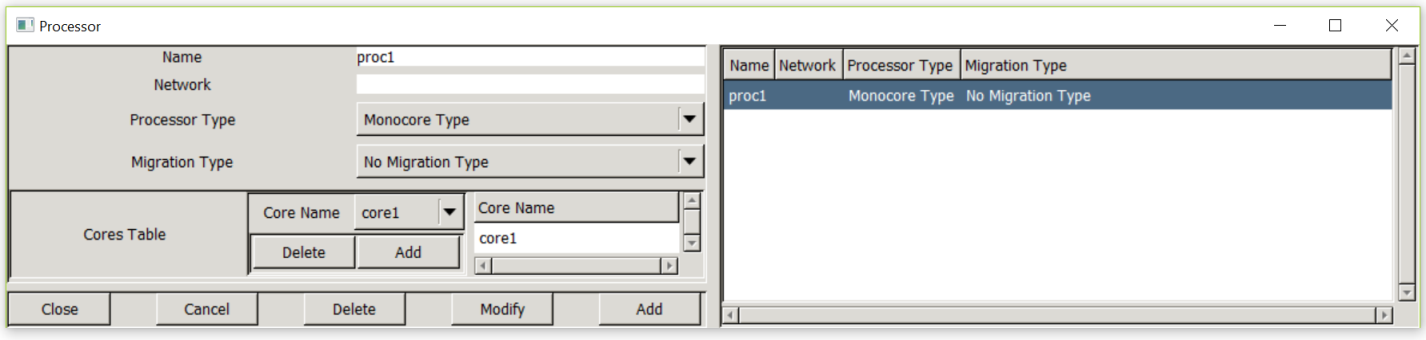
T3 = (1,20)

T4 = (2,20)

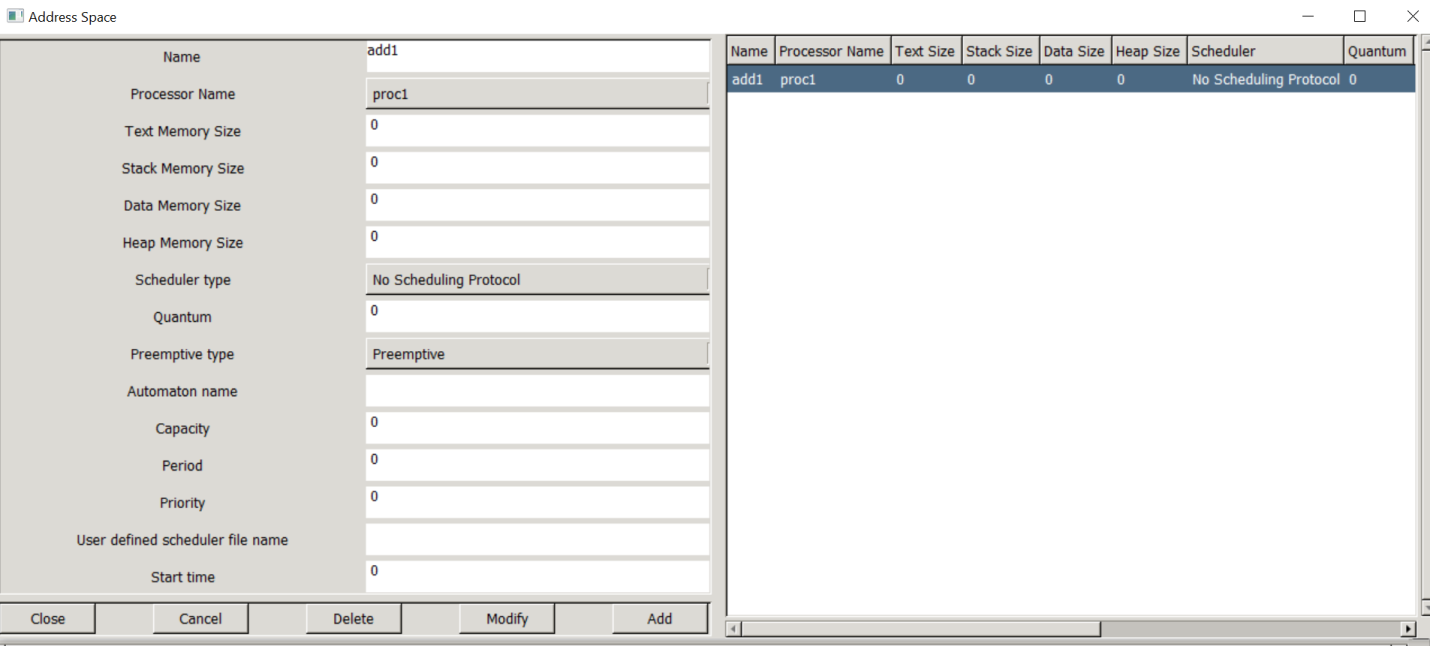
1. Adding the core:



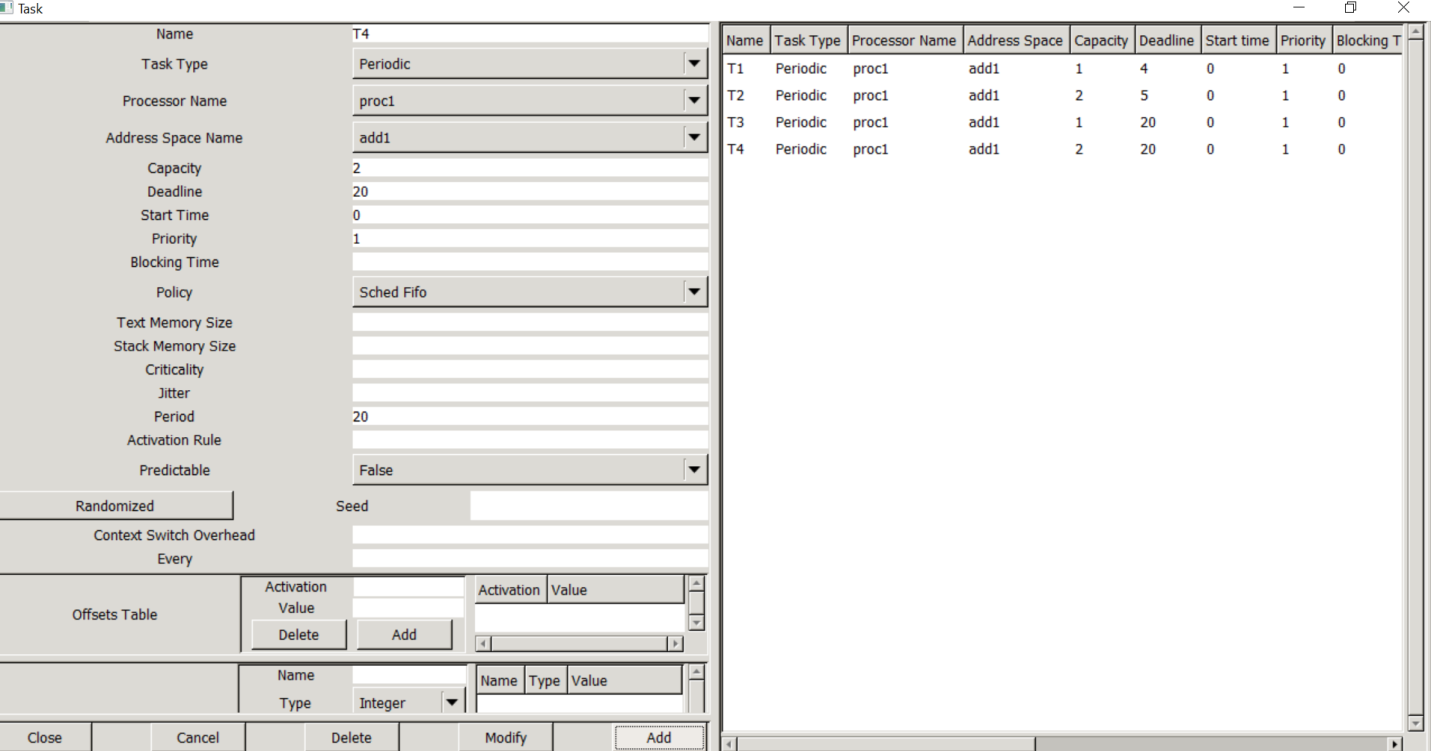
1. Adding the processor



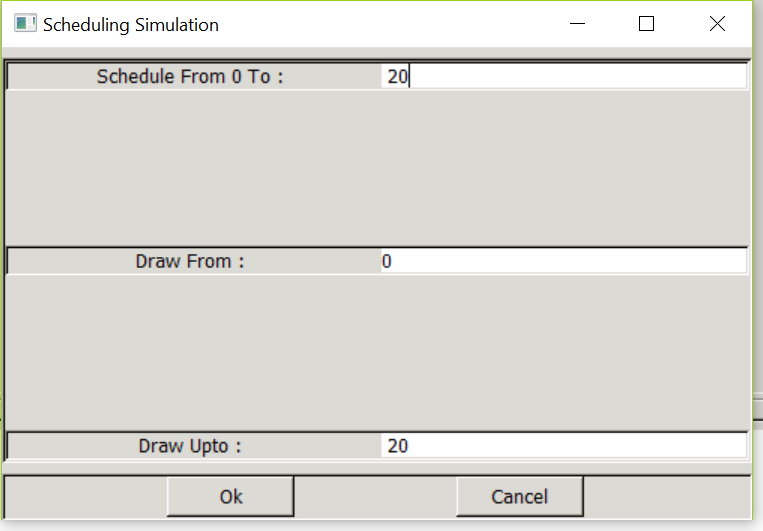
1. Adding the address space:



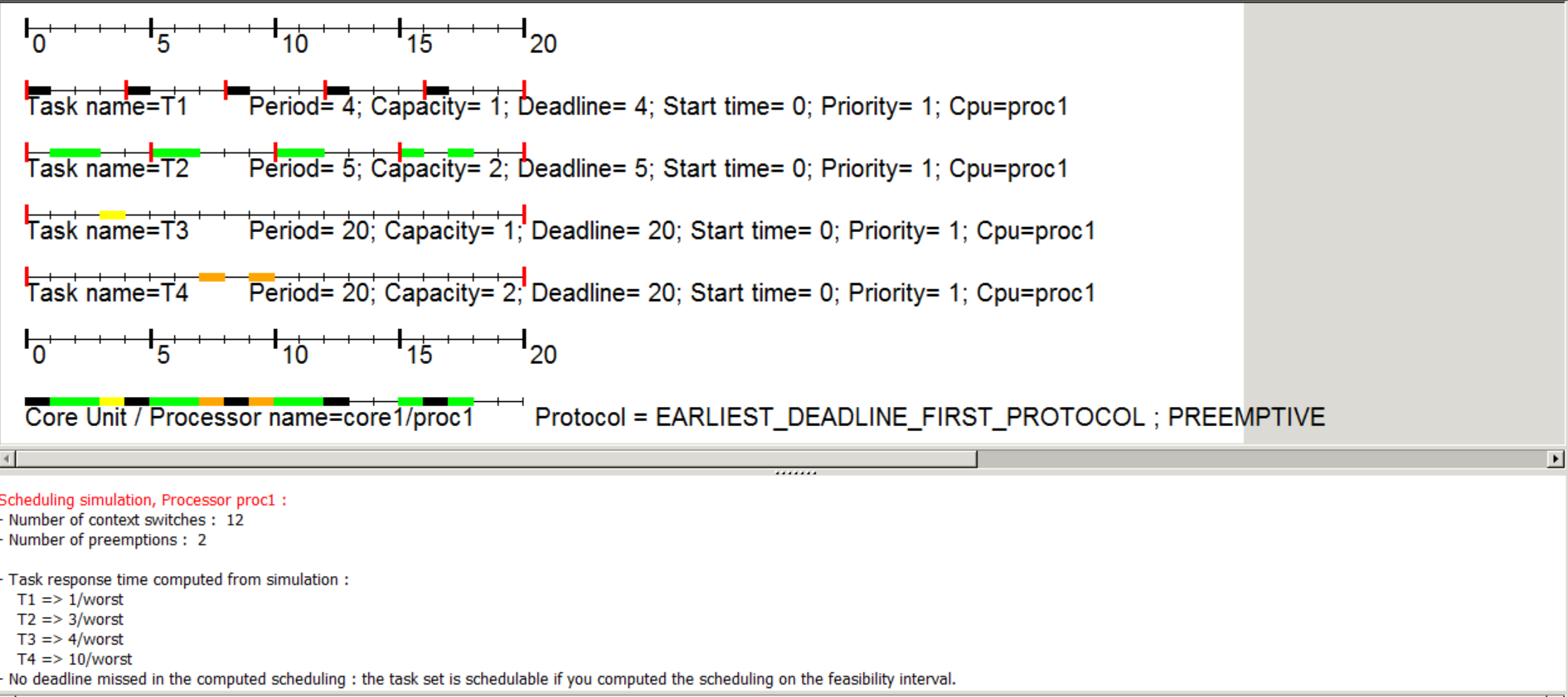
1. Adding the tasks:



1. Run the simulation.



* 1. Result



* According to EDF, the tasks are given the priorities as follows: T1 > T2 > T3 = T4. Though T3 and T4 have the same priority, T3 will be scheduled first as it follows a scheduled FIFO policy.
* At time unit 7, T4 (deadline = 20) will start execution, but at t = 8, third instance of T1 will arrive which has a deadline of 12, hence it preempts T4. T4 will continue its execution at t = 9.
* Similarly, at t = 16, T1 will preempt T2 as it has lower deadline (hence higher priority) in comparison to T2.

Appendix A

***Experiment 1***

**Introduction to Cheddar**

**Pre-Lab Report**

**Question 1:**

What are the pros and cons of clock driven scheduling

**Question 2:**

What is slack time of a job

**Question 3:**

What are the various dynamic priority algorithms.

**Appendix B**

***Experiment 1***

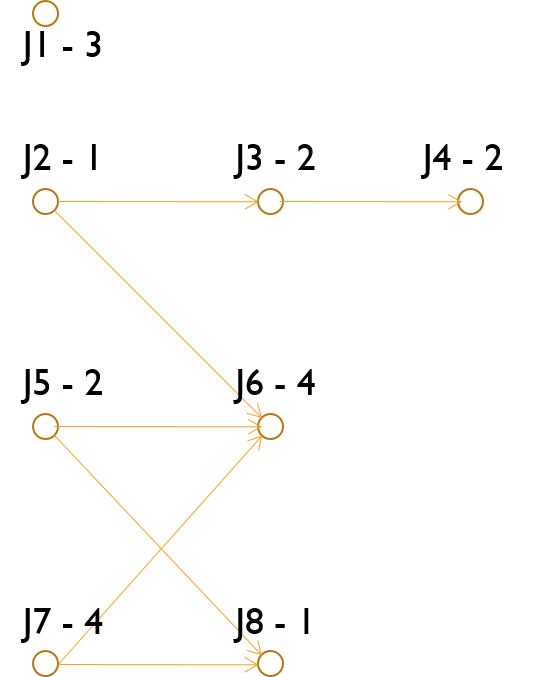
***Introduction to cheddar***

**In-Lab Report**

For the following Task sets and scheduling algorithms – submit the screen shoot of the schedule generated by Cheddar – with proper interpretations and conclusions regarding the schedule generated. If the task set is not schedulable – suggest an alternate method of scheduling

### Q1

For the task graph show below show that a priority driven schedule with non-preemption will produce a better schedule than priority driven schedule with pre-emption



All tasks are aperiodic with a deadline of 12. All tasks except for J5 release at 0. J5 releases at 4. The priority of jobs is such that J1 has a higher priority when compared to J2 has a higher priority when compared to J3 and so on. The scheduling is done on a processor which has dual identical cores with jobs migratable at any point in time.

### Q2

Show that the following tasks table is schedulable using EDF on a mono-core processor, if pre-emption is allowed but is not schedulable if pre-emption is not allowed. All tasks are aperiodic.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **J1** | **J2** | **J3** |
| r | 0 | 2 | 4 |
| e | 3 | 6 | 4 |
| d | 10 | 14 | 12 |

### Q3

Show that the following tasks table is schedulable using LLF but not using EDF on a dual-core processor. All tasks are aperiodic.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **J1** | **J2** | **J3** |
| r | 0 | 0 | 0 |
| e | 1 | 1 | 5 |
| d | 1 | 2 | 5 |

### Q4

Show that the following tasks table causes indeterminism due to the execution time – when scheduled using EDF. All tasks are periodic

|  |  |  |  |
| --- | --- | --- | --- |
|  | R | d | e |
| J1 | 0 | 10 | 5 |
| J2 | 0 | 10 | 2-6 |
| J3 | 4 | 15 | 8 |
| J4 | 0 | 20 | 10 |

### Q5

Show that the following tasks are not schedulable using either EDf/LLF on a tri-core processor. All tasks are periodic and preemptable. Migration is not possible between a Job.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **A** | **B** | **C** | **D** | **E** |
| φ | 0 | 0 | 0 | 0 | 0 |
| e | 1 | 1 | 1 | 6 | 6 |
| p | 2 | 2 | 2 | 8 | 8 |