SYSC 5104 Assignment 1

Glenn Davidson

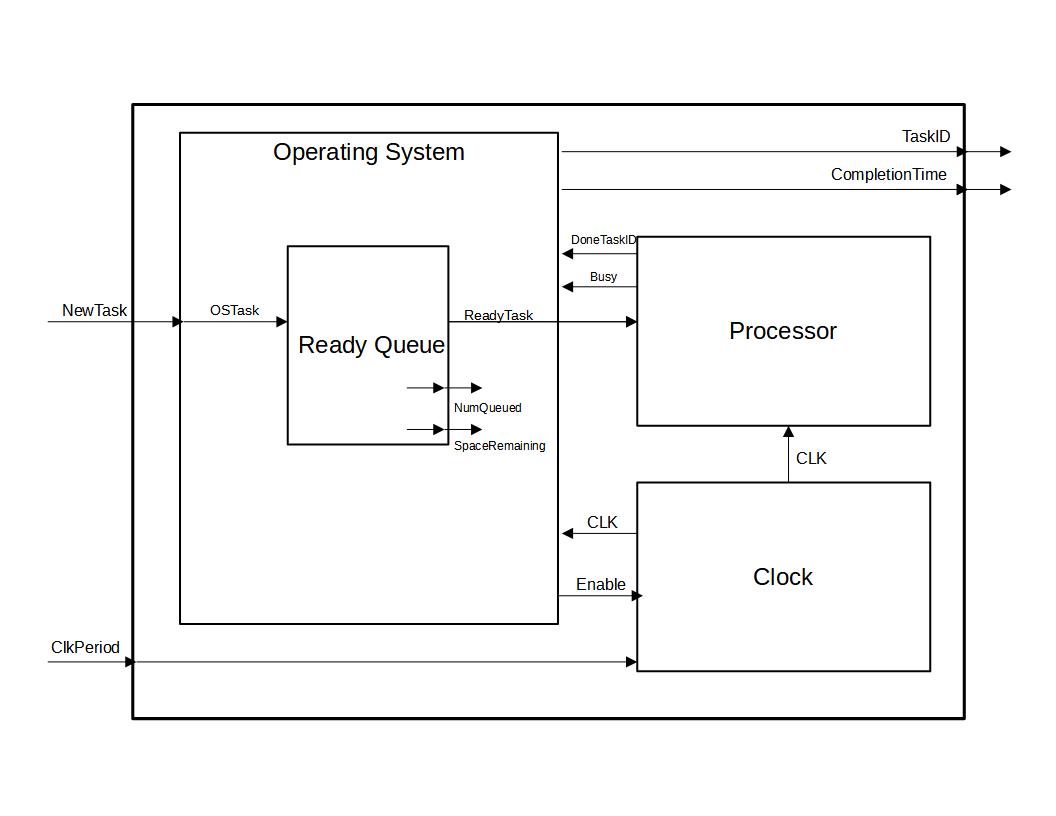
101205195

**1) Introduction**

A DEVS computer system model was attempted but not completed. The specification of three atomic models for the computer system were done, and two of the three specified models were implemented and tested. A DEVS atomic clock model was implemented and shown to be correct. A DEVS Task Queue was implemented and tested, and although it compiles, the implementation of the model is clearly incorrect. This report will outline the progress in this project, from the initial propopsal, the specification of the system, and the testing of the system.

**2) Initial Specification Document**

A DEVS computer system model will be developed to investigate the processing delay in a first come first served single processor system. The model will consist of a simple operating system, a processor, and a system clock. The operating system will log the submission time of new tasks, forward tasks to the processor, accept completion notifications from the processor, and output tasks processing time. The execution of the operating system itself will not be modeled, and will behave as if acting much faster than the processor model completes tasks. Modeling the operating system in this way will hopefully reduce the number of events occuring at the same time at clock events. Also ignored in this model are: task blocking (due to input/output and interrupts), memory access, task pre-emption, and system errors.



**Figure #1: Basic Computer System Diagram**

**Clock:**

The clock model will serve as a the logical clock for the processor. It will accept a clock period input, and an enable signal, and will output a 50% duty cycle square wave with the specified clock period if the clock is enabled. If the clock is disabled, it will output logic zero until it becomes enabled again. The state of the clock can be described as Enabled/Disabled, with state variables ClockPeriod, ClockLevel, EnableInput, and elapsed time. The clock may receive an Enable/Disable signal at any time.

**Processor:**

The processor model emulates the computation of tasks given to it, at a speed detemined by the Clock input it accepts. Its inputs are consist of a task to compute and a digital clock signal. The processor will compute one task at a time, and will complete one work unit of its assigned task every clock cycle. A task in this system consists of an integer identifier, and a count of the number of work units required to compete it. The processor may receive a task at any time, but will ignore any input tasks if it is busy. When the processor finishes a task, it will output the completed task’s ID. The states of the processor model include Idle, Busy, and DoneTask.

**Operating System:**

The operating system model accepts new tasks input to it, places them in a queue, and schedules them to run on a processor model if the processor indicates that it is free. The operating system recieves a processor busy signal when a task is already running on the processor, and recieves a task ID from the processor when a task is completed. The operating system logs the submission time of each task when received, and notes the time again when it recieves the indication the specific task has been competed. Upon completion of a task, the Operating System outputs the task’s ID, together with the total time that has passed since the task was submitted to the operating system. The ready queue will be implemented as an atomic model which the Operating System will use. When no tasks are in the queue and the processor is idle, the operating system will disable the clock.

**Ready Queue:**

The ready queue will store tasks given by the operating system, and will output a task when requested by the operating system in a first in, first out order. The queue will indicate to the operating system how many tasks are stored within, as well as how much space is left within the queue.

If the above is not easily achieved, the processor may output completed task IDs to the outside world, instead of the operating system computing the processing delay and outputting this with a task ID. If the above is easily achieved, the processor and operating system may be modified to implement round robin scheduling with a static time quantum, where pre-empted tasks are placed in the rear of the ready queue.

**3) Model Specification and Test Cases**

Specifications were completed for the following components of the ComputerSystem: Clock, Ready Queue, Processor. Their DEVs specification and their proposed tests to verify correctness are given in this section.

**Clock Specification:**

**X:** {Enable} - {bool}

**Y:** {ClkLevel} - {bool}

**S:** {IsEnabled, CurrentLevel} – {bool, bool}

**δext:**

if IsEnabled == true && Enable == false

-> IsEnabled = false; ClkLevel = false

if IsEnabled == false && Enable == true

-> IsEnabled = true; ClkLevel = true

for all other possibilities do nothing

**δint:**

if CurrentLevel = true -> CurrentLevel = false

if CurrentLevel = false -> CurrentLevel = true

**λ:**

ClkLevel = CurrentLevel

t**a(s):**

if IsEnabled == true -> T = ClockPeriod/2

else T = inf

**ClockPeriod = 1s** to begin testing

**Clock Test Case:**

* Begin idle
* Enable and watch run for a set amount of cycles
  + at 00:00:10 input {true} – observe ClkLevel = true
  + at 00:00:10:50 observe ClkLevel = false
  + at 00:00:11 observe ClkLevel = true
  + at 00:00:11:50 o observe ClkLevel = false
  + at 00:00:12 observe ClkLevel = true
* Disable mid cycle
  + at 00:00:12:10 input {false} observe ClkLevel = false
* Enable again and watch run for a set amount of cycles
  + at 00:00:20 input {true} – observe ClkLevel = true
  + at 00:00:20:50 observe ClkLevel = false
  + at 00:00:21 observe ClkLevel = true
  + at 00:00:21:50 observe ClkLevel = false
* Disable at rising clock edge, observe the disable beging executed before the rising edge
  + at 00:00:22:00 observe ClkLevel = false

**Clock Test Case File Input:**

00:00:10 1

00:00:12:10 0

00:00:20 1

00:00:22 0

**Clock Test Case Log For Above File:**

00:00:00:000

State for model input\_reader is next time: 00:00:00:000

State for model clock1 is clk\_high: 0 & clk\_enabled: 0

00:00:00:000

State for model input\_reader is next time: 00:00:10:000

State for model clock1 is clk\_high: 0 & clk\_enabled: 0

00:00:10:000

State for model input\_reader is next time: 00:00:02:010

State for model clock1 is clk\_high: 1 & clk\_enabled: 1

00:00:10:500

State for model input\_reader is next time: 00:00:02:010

State for model clock1 is clk\_high: 0 & clk\_enabled: 1

00:00:11:000

State for model input\_reader is next time: 00:00:02:010

State for model clock1 is clk\_high: 1 & clk\_enabled: 1

00:00:11:500

State for model input\_reader is next time: 00:00:02:010

State for model clock1 is clk\_high: 0 & clk\_enabled: 1

00:00:12:000

State for model input\_reader is next time: 00:00:02:010

State for model clock1 is clk\_high: 1 & clk\_enabled: 1

00:00:12:010

State for model input\_reader is next time: 00:00:07:990

State for model clock1 is clk\_high: 0 & clk\_enabled: 0

00:00:20:000

State for model input\_reader is next time: 00:00:02:000

State for model clock1 is clk\_high: 1 & clk\_enabled: 1

00:00:20:500

State for model input\_reader is next time: 00:00:02:000

State for model clock1 is clk\_high: 0 & clk\_enabled: 1

00:00:21:000

State for model input\_reader is next time: 00:00:02:000

State for model clock1 is clk\_high: 1 & clk\_enabled: 1

00:00:21:500

State for model input\_reader is next time: 00:00:02:000

State for model clock1 is clk\_high: 0 & clk\_enabled: 1

00:00:22:000

State for model input\_reader is next time: inf

State for model clock1 is clk\_high: 0 & clk\_enabled: 0

**Queue Specification:**

**X:** {NewTask, GiveTask} = {TaskMessage\_t, bool}

**Y:** {OutTask, QueueSize} = {TaskMessage\_t, int}

**S:** {QueueState, TaskQueue} = {enum, queue<TaskMessage\_t>}

QueueState enum = {“waiting”, “pushtask”,“poptask”}

**δext:**

{if QueueState == “waiting” && X:NewTask && NumWaiting < MaxSize

-> QueueState = “pushtask”; QueueStructure.pushback(NewTask)}

{if QueueState == “waiting” && X:GiveTask == true && NumWaiting > 0

-> QueueState = “poptask”}

{else ignore}

**δint:**

**//** case update the queue size only

if(QueueState == “pushtask”)

-> QueueState = “waiting”

**//** case output a task from the queue

{if QueueState == “poptask”

-> queue.pop(); QueueState = “waiting”}

**λ:**

**//** output new queue size only

{ if QueueState == “pushtask” -> Queuesize = queue.size()

**//** case output the next task in the queue and the new queue size

{if QueueState == “poptask” -> OutTask = Next; Queuesize = queue.size() -1}

t**a(s):**

**//** if a new task arrives and the queue is not full, put it in the queue immediatley, update size

{if QueueState == “pushtask” -> T = 0}

**//** if a GiveTask signal is recieved, output a task and size from the queue right away

{if QueueState == “poptask” -> T = 0}

// wait for new tasks forever

{if QueueState == “waiting” -> T = inf

**Queue Test Case:**

* Begin idle
* Input a series of new tasks to queue till full (Max Queue Size 4)
  + at 00:00:10 - input {NewTask1} – output QueueSize = 1
  + at 00:00:11 - input {NewTask2} – output QueueSize = 2
  + at 00:00:12 - input {NewTask3} – output QueueSize = 3
  + at 00:00:14 - input {NewTask4} – output QueueSize = 4
* Input a task to a full queue and observe it being ignored
  + at 00:00:15 - input {NewTask5} – output QueueSize = 4
* Push out the tasks one by one, with some arriving intermittently
  + at 00:00:16 - input {true} – output OutTask = NewTask1, QueueSize = 3
  + at 00:00:17 - input {true} – output OutTask = NewTask2, QueueSize = 2
  + at 00:00:18 - input {true} – output OutTask = NewTask3, QueueSize = 1
  + at 00:00:19 - input {NewTask6} – output QueueSize = 2
  + at 00:00:20 - input {true} – output OutTask = NewTask4, QueueSize = 1
  + at 00:00:21 - input {true} – output OutTask = NewTask6, QueueSize = 0
* Attempt to push a task out of an empty queue, observe no response

at 00:00:22 - input {true}

**Processor Specification (Internal Clock):**

**X:** {NewTask (TaskMessgae)}

**Y:** {Busy (bool), DoneTaskID (int)}

**S:** {TaskID (int), WorkUnitsRemain (int), ProcState}

ProcState = {“idle”, “begin”, “processing”}

**del\_ext:**

// case handle new task

{ if ProcState == “idle” -> TaskID = NewTask.ID; WorkUnitsRemain = NewTask.WorkUnits;

ProcState = “begin”}

// case ignore new task because processing

{ if ProcState == “begin” || “processing” -> no change }

**del\_int:**

// case beginning a task

{ if ProcState == “begin”

-> ProcState = “processing” }

// case finished a task – OUTPUT FIRST

{ if ProcStatre == “processing”

-> ProcState = “idle”; TaskID = 0; WorkUnitsRemain = 0; }

**lamda:**

// case beginning a new task

{if ProcState == “begin” -> Busy = true;}

// case finished a task

{Busy = false; DoneTaskID = TaskID}

t**a(s):**

// case start new task – output busy and advance to processing

{ if ProcState == “begin” -> T = 0 }

// case advance to end task

{ if ProcState == “processing” -> T = n\*WorkUnitsRemain }

// case idle

{ if ProcState == “idle” -> T = inf}

**NewTask Message Structure: {**int ID int WorkUnits**}**

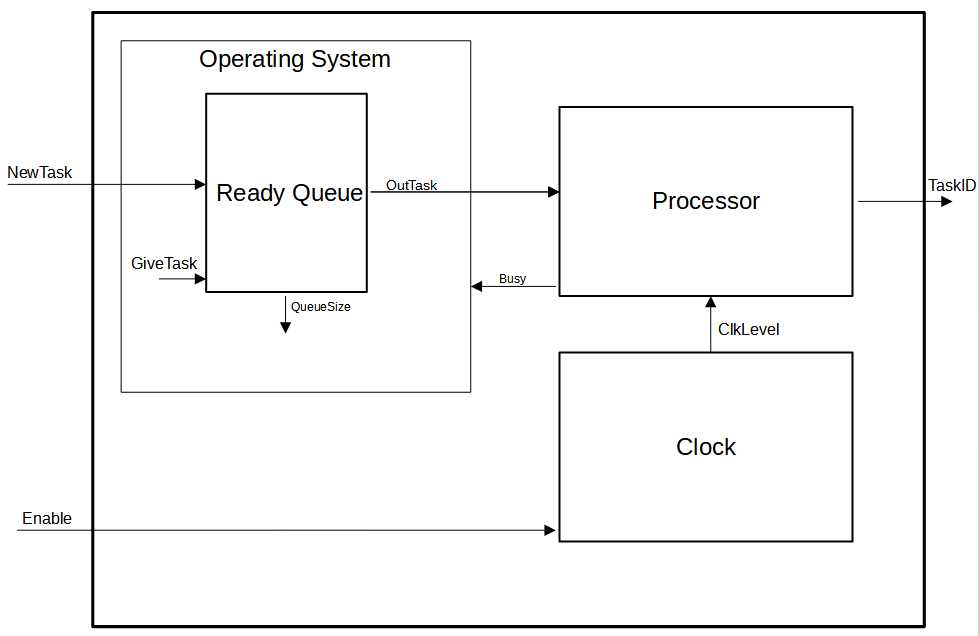
**WorkUnit (Implicit Clock Period) :** 1s

**Processor Test Case:**

* Begin idle
* Input new task and let it run to completion: (id = 1000, WorkUnits = 5)
  + at 00:00:10 - input {1000 5}
  + at 00:00:15 – observe completion
* Input new task after being idle for some time, ignore incoming task while processing
  + at 00:00:20 - input {1001 10}
  + at 00:00:21 - input {1002 2}
  + at 00:00:30 – observe completion
* Input new task after being idle for some time, input consecutive task at the time of completion and observe the new tasking being ignored as the external transition function should be computed before the external transition function
  + at 00:00:40 - input {1003 5}
  + at 00:00:45 - input {1004 10} – observe the new task being ignored
* Simulation end

**4) Results**

The initial system as specified has not yet been completed. A slightly simplified version of the system became the new target upon learning how to use Cadmium. The modified system can be seen in the figure below.



This model has also not been completed as of yet. It does appear that the queue structure is close to being correct, but there may be an issue with the input readers that is casuing the model to not behave propely. The to do list remaining for this project is:

* Fix the ready queue implementation
* Specify and implement a processor that uses an external clock signal
* Implement an operating system model that uses the ready queue structure and processor busy signal to implement scheduling
* Integreate the three models and test.