```
machine Mach_PartProc_Manage
//* ************************************************
//     The Event-B model of ARINC 653 Part 1
//     Created by Yongwang Zhao ( zhaoyongwang@gmail.com)
//     National Key Laboratory of Software Development Environment (NLSDE)
//     School of Computer and Engineering, Beihang University,  Beijing, China
//     ************************************************/
// this refinement defines the behavior of OPERATIONS according to ARINC653

 refines Mach_PartProc_Trans_with_Events   sees Ctx_PartProc_Manage

variables processes processes_of_partition  partition_mode process_state periodtype_of_process

            //process_ids //all created processes which have the ID. error handler does not has ID
             process_wait_type // mainproc_of_partition // the only one main proc of each partition
             locklevel_of_partition
             /* denotes the current lock level of the partition
                preemption_of_partitions */
             startcondition_of_partition
             /* denotes the reason the partition is started*/
             basepriority_of_process // Denotes the capability of the process to manipulate other processes.
```

period_of_process *// Identifies the period of activation for a periodic process. A distinct and unique value should be specified to designate the process as aperiodic*

timecapacity_of_process *// Defines the elapsed time within which the process should complete its execution.*

deadline_of_process *// Specifies the type of deadline relating to the process, and may be "hard" or "soft".*

currentpriority_of_process *// Defines the priority with which the process may access and receive resources. It is set to base priority at initialization time and is dynamic at runtime.*

deadlinetime_of_process *// The deadline time is periodically evaluated by the operating system to determine whether the process is satisfactorily completing its processing within the allotted time.*

releasepoint_of_process

/ the release point of processes */*

delaytime_of_process *// if the proc is delayed started, the delaytime should be saved(used when parttion START --> NORMAL)*

current_partition *// the partition in which a thread is now running. at each time, only one thread is running*

current_process

current_partition_flag *// true:indicate that the current_partition is valid, false: indicate NULL (unavailable)*

current_process_flag *// same as current partition flag*

clock_tick *// system clock ticks*

need_reschedule *// indicate the flag to reschedule after some events, for example suspend a thread*
need_procresch *//after partition scheduling, trigger the process level scheduling*
preempter_of_partition *// the process who execute the lock_preemption (increase the locklevel and disable scheduling), at most one preempter proc in a partition*
timeout_trigger *// all processes waiting for resources with a timeout, will be triggered after the timeout ellapsed.*
errorhandler_of_partition *// each partition has one error handler at most. other error handler can be created only after the previous handler is finished*
process_call_errorhandler *// error handler is created by a process, then the process is preempted by the error handler*

invariants

@inv_process_wait_type process_wait_type $\in$ processes $\nrightarrow$ **PROCESS_WAIT_TYPES** *// partial function, only if the process is in WAITING*

@inv_proc_waittype2 $\forall p\cdot(p\in$processes $\land$ (process_state(p)=**PS_Waiting** $\lor$ process_state(p)=**PS_WaitandSuspend**)$\Rightarrow p\in$dom(process_wait_type))
@inv_locklevel locklevel_of_partition $\in$ **PARTITIONS** $\rightarrow \mathbb{N}$
@inv_start_condition startcondition_of_partition $\in$ **PARTITIONS** $\nrightarrow$ **PARTITION_STARTCONDITIONS**

```
  @inv_start_imply_locklevel ∀p·(p∈PARTITIONS∧(partition_mode(p)=PM_COLD_START ∨
partition_mode(p)=PM_WARM_START) ⇒locklevel_of_partition(p)>0)
  @inv_locklevel0_imply_normal ∀p·(p∈PARTITIONS ∧ locklevel_of_partition(p)=0 ⇒
partition_mode(p)=PM_NORMAL)


  @inv_basepriority_processes basepriority_of_process ∈ processes → MIN_PRIORITY_VALUE..
MAX_PRIORITY_VALUE
  @inv_period_processes period_of_process ∈ processes → ℤ // infinite(-1) means aperiodic
  @inv_timecapacity_processes timecapacity_of_process ∈ processes → ℤ // infinite(-1) means no deadline
time
  @inv_deadline_processes deadline_of_process ∈ processes → DEADLINE_TYPE
  @inv_currentpriority_processes currentpriority_of_process ∈ processes → MIN_PRIORITY_VALUE..
MAX_PRIORITY_VALUE
  @inv_deadlinetime_processes deadlinetime_of_process ∈ processes ⇸ ℕ
  @inv_releasepoint_of_process releasepoint_of_process ∈ processes ⇸ ℕ // @inv_nextreleasepoint_of_process
nextreleasepoint_of_process ∈ processes ⇸ ℕ
  @inv_delaytime_of_process delaytime_of_process ∈ processes ⇸ ℕ
  @inv_delaytime2 ∀p·(p∈processes ∧ partition_mode(processes_of_partition(p))=PM_NORMAL ∧
process_state(p)=PS_Waiting ∧ process_wait_type(p)=PROC_WAIT_DELAY ⇒ p∈dom(delaytime_of_process) )
  @inv_periodtype1 ∀p·(p∈processes ⇒(periodtype_of_process(p)=APERIOD_PROC⇔
period_of_process(p)=INFINITE_TIME_VALUE))
  @inv_periodtype2 ∀p·(p∈processes ⇒(periodtype_of_process(p)=PERIOD_PROC⇔ period_of_process(p)≠
INFINITE_TIME_VALUE))

  @inv_curpart current_partition ∈ PARTITIONS
  @inv_curpart_flag current_partition_flag ∈ BOOL
  @inv_curproc_flag current_process_flag ∈ BOOL
  @inv_curproc (current_process_flag = TRUE ⇒  current_process ∈ processes)
  @inv_curprocimplycurpart current_process_flag = TRUE ⇒ current_partition_flag = TRUE
  @inv_cur_proc_part (current_process_flag = TRUE ∧ current_partition_flag = TRUE ⇒
processes_of_partition(current_process) = current_partition)
  @inv_partstate_curr (current_partition_flag = TRUE ⇒ partition_mode(current_partition) ≠ PM_IDLE)
  @inv_procstate_curr (current_process_flag = TRUE ⇒ process_state(current_process) = PS_Running ∧
partition_mode(current_partition)=PM_NORMAL)

  @inv_clocktick clock_tick ∈ ℕ
  @inv_need_reschedule need_reschedule ∈ BOOL // @inv_preemption preemption_of_partitions ∈
PARTITIONS → BOOL
  @inv_need_procresch need_procresch∈BOOL
  @inv_preempter_of_partition preempter_of_partition ∈PARTITIONS ⤔ processes // partial injection.
  @inv_locklevel_imply_preempter ∀p·(p∈PARTITIONS ∧ partition_mode(p)=PM_NORMAL ∧
locklevel_of_partition(p)  > 0 ⇒   p∈dom(preempter_of_partition))
   @inv_locklevel_imply_preempter2 ∀p·(p∈PARTITIONS ∧ partition_mode(p)=PM_NORMAL ∧ p∈
dom(preempter_of_partition) ⇒ locklevel_of_partition(p)  > 0 )

   /* @inv_preemption_locklevel ∀p·(p∈PARTITIONS ⇒ ((locklevel_of_partition(p)  > 0 ⇔
preemption_of_partitions(p)  = FALSE)
        ∧ (locklevel_of_partition(p)  = 0 ⇔ preemption_of_partitions(p)  = TRUE))) */
   @inv_tmout_trig_type timeout_trigger∈processes ⇸ (PROCESS_STATES × ℕ1) // a process waiting for some
resource with a timeout and will be transited to another state
   @inv_tmout_trig_state ∀p·(p∈dom(timeout_trigger) ⇒ (process_state(p)  = PS_Waiting ∨ process_state(p)  =
PS_WaitandSuspend ∨ process_state(p)  = PS_Suspend)) // @inv_tmout_trig_nextstate
dom(ran(timeout_trigger))  = {PS_Ready} // in the kernel, in fact, the next state when time out is always READY
   @inv_errhandler_partition errorhandler_of_partition  ∈ PARTITIONS ⤔ processes // partial injection. a
partition has one handler at most, when error happens
   @inv_errhandler_has_maxpriority ∀p·(p∈ran(errorhandler_of_partition) ⇒   (p∈dom(currentpriority_of_process)
⇒ currentpriority_of_process(p)  = MAX_PRIORITY_VALUE))
   @inv_errhandler_inpartition ∀part,p·(part↦p  ∈  errorhandler_of_partition ⇒   processes_of_partition(p)  = part)
// @inv_atmostoneerrhandler_inpartition ∀p·(p∈processes ⇒ card(errorhandler_of_partition∼[{p}])  ≤ 1) // an
error handler only belongs to a partition
   @inv_process_call_errorhandler process_call_errorhandler  ∈ processes ⤔ processes // partial injection
   @inv_errhandlerandcaller_insamepart ∀p1,p2·(p1 ↦ p2 ∈ process_call_errorhandler ⇒
processes_of_partition(p1)=  processes_of_partition(p2))  // error handler and its creator process is in same
partition
   @inv_from_errhandler_to_caller  dom(process_call_errorhandler)  = ran(errorhandler_of_partition)  ∧
ran(process_call_errorhandler) ⊆ processes ∖ran(errorhandler_of_partition) //each error handler, this the only one
caller

events
   event INITIALISATION extends INITIALISATION
      then
         @act100 process_wait_type  ≔ ∅
         @act10 locklevel_of_partition  ≔ PARTITIONS  × {1}
         @act12 startcondition_of_partition  ≔  ∅ // @act121 schedulable_of_partition  ≔   PARTITIONS  × {FALSE}
         @act13 basepriority_of_process  ≔ ∅
         @act14 period_of_process  ≔  ∅
         @act15 timecapacity_of_process  ≔  ∅
         @act16 deadline_of_process  ≔  ∅
         @act17 currentpriority_of_process  ≔  ∅
         @act18 deadlinetime_of_process  ≔  ∅
         @act19 releasepoint_of_process  ≔ ∅ // @act20 nextreleasepoint_of_process  ≔ ∅
         @act200 delaytime_of_process  ≔ ∅
         @act21 current_partition_flag  ≔  FALSE
```

```
    @act22 current_process_flag ≔ FALSE
    @act23 current_partition :∈ PARTITIONS
    @act24 current_process :∈ PROCESSES
    @act25 clock_tick ≔ 1
    @act26 need_reschedule ≔  FALSE
    @act28 need_procresch ≔ FALSE
    @act27 preempter_of_partition ≔ ∅
    /* @act17 remain_timecapacity_of_process ≔  ∅
       @act18 wakeuptime_of_process ≔  ∅
       @act_asgn_preemption preemption_of_partitions ≔ PARTITIONS × {TRUE} */
    @act_asgn_tmouttrig timeout_trigger ≔ ∅
    @act_asgn_errhdlofpart errorhandler_of_partition ≔ ∅
    @act_process_call_errorhandler process_call_errorhandler ≔ ∅
end

event ticktock // timer interrupt event, triggered by the timer in hardware. one tick in each ONE_TICK_TIME
  then
    @act01 clock_tick ≔ clock_tick + 1
    @act02 need_reschedule ≔ TRUE
end
```

```
event partition_schedule extends partition_schedule
  any found // current time is in one partition window?

  where
    @grd10 need_reschedule = TRUE
    @grd11 found ∈ BOOL
    //the next two line are commented by the reason that ARINC653 does not implement the scheduling
    @grd12 ∃x,y,b,n·(((x↦y)↦b) ∈ partitionTimeWindows ∧ timeWindowsofPartition((x↦y)↦b) = part ∧
          (x + n∗majorFrame) < clock_tick∗ONE_TICK_TIME ∧ clock_tick∗ONE_TICK_TIME < (x +y + n∗
majorFrame)) ⇒found=TRUE
    @grd13 ¬(∃x,y,b,n·(((x↦y)↦b) ∈ partitionTimeWindows ∧ timeWindowsofPartition((x↦y)↦b) = part ∧
          (x + n∗majorFrame) < clock_tick∗ONE_TICK_TIME ∧ clock_tick∗ONE_TICK_TIME < (x +y + n∗
majorFrame))) ⇒found=FALSE
  then
    @act11 current_partition_flag ≔ found
    @act12 current_partition ≔ part // if flag is FALSE, the assign is arbitrary
    @act13 current_process_flag ≔ FALSE
    @act14 need_procresch :∣ ((partition_mode(part) = PM_NORMAL) ⇒ need_procresch' = TRUE) ∧
((partition_mode(part) = PM_COLD_START ∨ partition_mode(part) = PM_WARM_START) ⇒ need_procresch' =
FALSE )
    @act15 need_reschedule :∣ ((partition_mode(part) = PM_NORMAL) ⇒ need_reschedule' = FALSE) ∧
```

```
((partition_mode(part) = PM_COLD_START ∨ partition_mode(part) = PM_WARM_START) ⇒ need_reschedule' =
TRUE )

  end

  event process_schedule // if there is not error handler and preempter in this partition
  extends process_schedule
    where
      @grd10 need_procresch = TRUE
      @grd11 current_partition_flag = TRUE ∧ current_partition = part
      @grd12 (current_partition∉dom(errorhandler_of_partition) ∨
process_state(errorhandler_of_partition(current_partition))=PS_Dormant) ∧
locklevel_of_partition(current_partition) = 0   //current_partition∉dom(preempter_of_partition)
      @grd13 ∀p·(p∈processes_of_partition∼[{part}] ⇒ currentpriority_of_process(p) ≤
currentpriority_of_process(proc))
    then
      @act22 current_process ≔ proc
      @act24 current_process_flag ≔ TRUE
      @act25 need_reschedule ≔ FALSE
      @act26 need_procresch ≔ FALSE
  end
  event run_errorhandler_preempter // if there is the error handler, it is executed, otherwise the preempter is
executed
  extends process_schedule
  when
      @grd30 need_procresch = TRUE
      @grd31 current_partition_flag = TRUE ∧ current_partition = part
      @grd32 (current_partition∈dom(errorhandler_of_partition) ∧
process_state(errorhandler_of_partition(current_partition))≠PS_Dormant) ∨
locklevel_of_partition(current_partition) > 0 //current_partition∈dom(preempter_of_partition)
      @grd33 current_partition∈dom(errorhandler_of_partition) ⇒ proc =
errorhandler_of_partition(current_partition)
      @grd34 current_partition∉dom(errorhandler_of_partition) ∧ locklevel_of_partition(current_partition) > 0 ⇒
proc = preempter_of_partition(current_partition)
    then
      @act22 current_process ≔ proc
      @act24 current_process_flag ≔ TRUE
      //@act26 process_state(proc) ≔ PS_Running
      @act25 need_reschedule ≔ FALSE
      @act26 need_procresch ≔ FALSE
  end
```

```
event get_partition_status
   where
      @grd01 current_partition_flag = TRUE
end

event set_partition_mode_to_idle // shutdown the partition
extends set_partition_mode_to_idle
when
  @grd40 current_partition_flag = TRUE ∧ current_partition=part
   then
      @act401 process_wait_type ≔ procs ⩤ process_wait_type
      @act402 locklevel_of_partition(part) ≔ 1
      /* @act403 preemption_of_partitions(part) ≔ TRUE
           @act404 startcondition_of_partition(part) ≔    NORMAL_START
           @act404 schedulable_of_partition(part) ≔ FALSE */
      @act405 basepriority_of_process ≔   procs ⩤ basepriority_of_process
      @act406 period_of_process ≔     procs ⩤ period_of_process
      @act407 timecapacity_of_process ≔     procs ⩤ timecapacity_of_process
      @act408 deadline_of_process ≔     procs ⩤ deadline_of_process
      @act409 currentpriority_of_process ≔  procs ⩤ currentpriority_of_process
      @act410 deadlinetime_of_process ≔  procs ⩤ deadlinetime_of_process
      @act411 releasepoint_of_process ≔  procs ⩤ releasepoint_of_process // @act412
nextreleasepoint_of_process ≔  procs ⩤ nextreleasepoint_of_process
      @act413 delaytime_of_process ≔ procs ⩤ delaytime_of_process
      @act414 timeout_trigger ≔  procs ⩤ timeout_trigger
      @act415 errorhandler_of_partition ≔  {part} ⩤ errorhandler_of_partition
      @act416 process_call_errorhandler ≔  procs ⩤ process_call_errorhandler
      @act417 current_partition_flag ≔ FALSE
      @act418 current_process_flag ≔ FALSE
      @act419 preempter_of_partition ≔  {part} ⩤ preempter_of_partition
  end

  event set_partition_mode_to_normal refines set_partition_mode_to_normal
    any part procs procs2 staperprocs dstaperprocs suspaperprocs stperprocs dstperprocs rlt nrlt1 nrlt2 newm
dl1 dl2 dl3 dl4
    where
      @grd01 part ∈ PARTITIONS
      @grd02 partition_mode(part) = PM_COLD_START ∨  partition_mode(part) = PM_WARM_START
      @grd40 current_partition_flag = TRUE ∧ current_partition=part
      @grd08 card(processes_of_partition∼[{part}]) > 0
```

```
@grd09 procs =processes_of_partition∼[{part}]  ∩ process_state∼[{PS_Waiting}] // transit to normal, some WAITING procs (aperiod, not suspended) will be transit to READY
@grd10 procs2 = processes_of_partition∼[{part}]  ∩ process_state∼[{PS_WaitandSuspend}] // transit to normal, the WAITandSuspend procs will be transit to suspend
@grd401 staperprocs = procs ∩ period_of_process∼[{INFINITE_TIME_VALUE}]  ∩ process_wait_type∼[{PROC_WAIT_PARTITIONNORMAL}]
@grd402 dstaperprocs = procs ∩ period_of_process∼[{INFINITE_TIME_VALUE}]  ∩ process_wait_type∼[{PROC_WAIT_DELAY}]
@grd403 suspaperprocs = procs2
@grd404 stperprocs = (procs ∖ period_of_process∼[{INFINITE_TIME_VALUE}])  ∩ process_wait_type∼[{PROC_WAIT_PARTITIONNORMAL}]
@grd405 dstperprocs =  (procs ∖ period_of_process∼[{INFINITE_TIME_VALUE}])  ∩ process_wait_type∼[{PROC_WAIT_DELAY}]
@grd406 rlt ∈ dstaperprocs → ℕ
@grd407 ∀p·(p∈dstaperprocs ⇒ rlt(p) = clock_tick∗ONE_TICK_TIME + delaytime_of_process(p))
@grd408 nrlt1 ∈ stperprocs → ℕ
@grd409 ∀p,x,y,b·(p∈stperprocs ∧ ((x↦y)↦b)= firstperiodicprocstart_timeWindow_of_Partition(part)⇒ nrlt1(p) = ((clock_tick∗ONE_TICK_TIME)÷majorFrame+1)∗majorFrame + x)
@grd410 nrlt2 ∈ dstperprocs → ℕ
@grd411 ∀p,x,y,b·(p∈dstperprocs ∧ ((x↦y)↦b)= firstperiodicprocstart_timeWindow_of_Partition(part) ⇒ nrlt2(p) = ((clock_tick∗ONE_TICK_TIME)÷majorFrame+1)∗majorFrame + x +delaytime_of_process(p) )
```

```
        @grd412 newm = PM_NORMAL
        @grd413 dl1∈staperprocs ∪suspaperprocs → ℕ
        @grd414 ∀p·(p∈staperprocs ∪suspaperprocs ⇒ dl1(p)=clock_tick∗ONE_TICK_TIME +
timecapacity_of_process(p))
        @grd415 dl2 ∈dstaperprocs → ℕ
        @grd416 ∀p·(p∈dstaperprocs ⇒ dl2(p)=clock_tick∗ONE_TICK_TIME +delaytime_of_process(p)+
timecapacity_of_process(p))
        @grd417 dl3∈stperprocs → ℕ
        @grd418 ∀p·(p∈stperprocs ⇒ dl3(p)=clock_tick∗ONE_TICK_TIME +timecapacity_of_process(p))
        @grd419 dl4∈dstperprocs → ℕ
        @grd420 ∀p·(p∈dstperprocs ⇒ dl4(p)=clock_tick∗ONE_TICK_TIME +delaytime_of_process(p) +
timecapacity_of_process(p))
     with
        @procsstate procsstate = (staperprocs ×{PS_Ready}) ∪ ((dstaperprocs ∪ stperprocs ∪ dstperprocs)×
{PS_Waiting})
     then
        @act400 partition_mode(part) ≔ newm
        @act401 process_state ≔ (process_state (staperprocs ×{PS_Ready})) (suspaperprocs ×
{PS_Suspend})
        @act402 releasepoint_of_process ≔ releasepoint_of_process rlt nrlt1 nrlt2 // @act403
nextreleasepoint_of_process ≔ nextreleasepoint_of_process nrlt1 nrlt2
```

```
      @act403 deadlinetime_of_process ≔  deadlinetime_of_process      dl1      dl2      dl3      dl4
      @act404 locklevel_of_partition(part) ≔ 0 // @act405 schedulable_of_partition(part) ≔ TRUE
  end

  event set_partition_mode_to_coldstart extends set_partition_mode_to_coldstart
    when
      @grd40 current_partition_flag = TRUE ∧ current_partition=part
    then
      @act401 process_wait_type ≔ procs ⩤ process_wait_type
      @act402 locklevel_of_partition(part) ≔ 1
      /* @act403 preemption_of_partitions(part) ≔ TRUE
         @act404 startcondition_of_partition(part) ≔    NORMAL_START
         @act404 schedulable_of_partition(part) ≔ FALSE */
      @act405 basepriority_of_process ≔   procs ⩤ basepriority_of_process
      @act406 period_of_process ≔    procs ⩤ period_of_process
      @act407 timecapacity_of_process ≔    procs ⩤ timecapacity_of_process
      @act408 deadline_of_process ≔    procs ⩤ deadline_of_process
      @act409 currentpriority_of_process ≔    procs ⩤ currentpriority_of_process
      @act410 deadlinetime_of_process ≔    procs ⩤ deadlinetime_of_process
      @act411 releasepoint_of_process ≔    procs ⩤ releasepoint_of_process // @act412
nextreleasepoint_of_process ≔    procs ⩤ nextreleasepoint_of_process
```

```
        @act413 delaytime_of_process ≔ procs ⩤ delaytime_of_process
        @act414 timeout_trigger ≔  procs ⩤ timeout_trigger
        @act415 errorhandler_of_partition ≔  {part} ⩤ errorhandler_of_partition
        @act416 process_call_errorhandler ≔  procs ⩤ process_call_errorhandler // @act417 current_partition_flag
≔ FALSE
        @act418 current_process_flag ≔ FALSE
        @act419 preempter_of_partition ≔  {part} ⩤ preempter_of_partition
  end

  event set_partition_mode_to_warmstart extends set_partition_mode_to_warmstart
    when
       @grd40 current_partition_flag = TRUE ∧ current_partition=part
    then
        @act401 process_wait_type ≔ procs ⩤ process_wait_type
        @act402 locklevel_of_partition(part) ≔ 1
        /* @act403 preemption_of_partitions(part) ≔ TRUE
           @act404 startcondition_of_partition(part) ≔   NORMAL_START
           @act404 schedulable_of_partition(part) ≔ FALSE */
        @act405 basepriority_of_process ≔  procs ⩤ basepriority_of_process
        @act406 period_of_process ≔   procs ⩤ period_of_process
        @act407 timecapacity_of_process ≔   procs ⩤ timecapacity_of_process
```

```
      @act408 deadline_of_process ≔  procs ⩤ deadline_of_process
      @act409 currentpriority_of_process ≔  procs ⩤ currentpriority_of_process
      @act410 deadlinetime_of_process ≔  procs ⩤ deadlinetime_of_process
      @act411 releasepoint_of_process ≔  procs ⩤ releasepoint_of_process // @act412
nextreleasepoint_of_process ≔  procs ⩤ nextreleasepoint_of_process
      @act413 delaytime_of_process ≔ procs ⩤ delaytime_of_process
      @act414 timeout_trigger ≔  procs ⩤ timeout_trigger
      @act415 errorhandler_of_partition ≔  {part} ⩤ errorhandler_of_partition
      @act416 process_call_errorhandler ≔  procs ⩤ process_call_errorhandler // @act417 current_partition_flag
≔ FALSE
      @act418 current_process_flag ≔ FALSE
      @act419 preempter_of_partition ≔  {part} ⩤ preempter_of_partition
  end

  event get_process_id
    any proc
    where
      @grd01 current_partition_flag = TRUE
      @grd02 proc ∈ processes
      @grd03 processes_of_partition(proc) = current_partition
  end
```

```
event get_process_status
  any proc
  where
    @grd01 current_partition_flag = TRUE
    @grd02 proc ∈ processes
    @grd03 processes_of_partition(proc) = current_partition
end

event create_process extends create_process
  any basepriority period timecapacity dl
  where
    @grd201 current_partition_flag = TRUE
    @grd200 part = current_partition
    @grd20 basepriority ∈ MIN_PRIORITY_VALUE .. MAX_PRIORITY_VALUE
    @grd21 period ∈ ℤ
    @grd22 timecapacity ∈ ℤ
    @grd23 period≠ INFINITE_TIME_VALUE ⇒ (∃n·(n∈ℕ ∧period =n∗Period_of_Partition(part)))
    @grd24 period≠ INFINITE_TIME_VALUE ⇒ (timecapacity ≤ period)
    @grd25 dl ∈DEADLINE_TYPE
    @ptype1 (ptype=APERIOD_PROC⇔ period=INFINITE_TIME_VALUE)
    @ptype2 (ptype=PERIOD_PROC⇔ period≠INFINITE_TIME_VALUE)
  then
    @act21 basepriority_of_process(proc)  ≔  basepriority
    @act22 period_of_process(proc)  ≔   period
    @act23 timecapacity_of_process(proc)  ≔   timecapacity
    @act34 deadline_of_process(proc)  ≔   dl
    @act35 currentpriority_of_process(proc)  ≔   basepriority

    //@act36 process_ids  ≔ process_ids  ∪  {proc}
end

event set_priority
  any p pri
  where
    @grd10 current_partition_flag = TRUE
    @grd11 p ∈processes
    @grd12 p ∈  processes_of_partition∼[{current_partition}]
    @grd14 pri ∈  MIN_PRIORITY_VALUE  ..  MAX_PRIORITY_VALUE
    @grd15 process_state(p)  ≠  PS_Dormant
    //@grd16 p∉ran(errorhandler_of_partition)
  then
```

```
        @act10 currentpriority_of_process(p) ≔ pri
        //@act11 need_reschedule :∣ (locklevel_of_partition(current_partition) =0 ∧(process_state(p)=PS_Ready ∨
process_state(p)=PS_Running)⇒ need_reschedule' = TRUE) ∧ (locklevel_of_partition(current_partition) ≠0 ⇒
need_reschedule' = need_reschedule)
        @act11 need_reschedule :∣ (locklevel_of_partition(current_partition) =0 ⇒ need_reschedule' = TRUE) ∧
(locklevel_of_partition(current_partition) ≠0 ⇒ need_reschedule' = need_reschedule)
  end

  event suspend_self
  //extends suspend_self
 //    any timeout timeouttrig waittype
 refines suspend_self
 any part proc newstate timeout timeouttrig waittype
    where
        @grd01 part ∈ PARTITIONS
     @grd02 proc ∈ processes
     @grd03 newstate ∈ PROCESS_STATES
     @grd06 processes_of_partition(proc) = part
     @grd31 partition_mode(part) = PM_NORMAL
     @grd32 process_state(proc) = PS_Running
     @grd33 newstate = PS_Suspend
     @grd34 periodtype_of_process(proc)  = APERIOD_PROC

        @grd401 timeout∈ℤ ∧ timeout≠0
        @grd402 current_process_flag = TRUE ∧ current_partition_flag = TRUE
        @grd200 part = current_partition
        @grd403 proc = current_process
        @grd404 part∈dom(errorhandler_of_partition) ⇒ proc ≠ errorhandler_of_partition(part)
        @grd405 locklevel_of_partition(part) = 0
        @grd406 period_of_process(proc)  ≠ INFINITE_TIME_VALUE
        @grd407 timeouttrig ∈ processes ⇸ (PROCESS_STATES × ℕ1)
        @grd408 timeout ≠ INFINITE_TIME_VALUE ∧ timeout≠0⇒ timeouttrig = {proc ↦ (PS_Ready ↦ (timeout
+clock_tick ∗ ONE_TICK_TIME))}
        @grd409 timeout = INFINITE_TIME_VALUE ⇒  timeouttrig = ∅
        @grd410 waittype∈processes⇸PROCESS_WAIT_TYPES
        @grd411 timeout>0 ⇒ waittype={proc ↦ PROC_WAIT_TIMEOUT}
        @grd412 (timeout =  INFINITE_TIME_VALUE ∨ timeout = 0) ⇒ waittype = ∅
    then
        @act11 process_state(proc) ≔ newstate
        @act40 current_process_flag :∣(timeout=0⇒current_process_flag' = TRUE) ∧
(timeout>0⇒current_process_flag' = FALSE)
        @act41 timeout_trigger ≔ timeout_trigger     timeouttrig
```

```
        @act42 need_reschedule :|(timeout=0⇒need_reschedule' = FALSE) ∧ (timeout>0⇒need_reschedule' =
TRUE)
        @act43 process_wait_type ≔ process_wait_type     waittype
  end

  event suspend
  //extends suspend
  refines suspend
  any part proc newstate
    where
        @grd01 part ∈ PARTITIONS
     @grd02 proc ∈ processes
     @grd03 newstate ∈ PROCESS_STATES
     @grd06 processes_of_partition(proc) = part
     @grd30 partition_mode(part) = PM_NORMAL ∨ partition_mode(part) = PM_COLD_START ∨
partition_mode(part) = PM_WARM_START
     @grd31 partition_mode(part) = PM_NORMAL ⇒  (process_state(proc) = PS_Ready ∧ newstate =
PS_Suspend)∨  (process_state(proc) = PS_Waiting ∧ newstate = PS_WaitandSuspend)
     @grd32 (partition_mode(part) = PM_COLD_START ∨  partition_mode(part) = PM_WARM_START)⇒
(process_state(proc) = PS_Waiting ∧ newstate = PS_WaitandSuspend)
```

```
        @grd40 current_process_flag = TRUE  ∧  current_partition_flag = TRUE
        @grd200 part = current_partition
        @grd41 current_process_flag = TRUE ⇒ proc ≠ current_process
        @grd42 locklevel_of_partition(part) = 0 ∨ proc ∉ ran(process_call_errorhandler) // preemption is enabled
or the PROCESS_ID is not the process which the error handler has pre-empted
        @grd43 period_of_process(proc) = INFINITE_TIME_VALUE // periodic process could not be suspend
        @grd45 process_state(proc) ≠ PS_Dormant
        @grd46 process_state(proc) ≠PS_Suspend ∧  process_state(proc) ≠PS_WaitandSuspend

  then
    @act11 process_state(proc) ≔ newstate
  end

  event resume
  //extends resume
  refines resume
    any part proc newstate reschedule
    where
        @grd01 part ∈ PARTITIONS
    @grd02 proc ∈ processes
    @grd03 newstate ∈ PROCESS_STATES
```

@grd06 processes_of_partition(*proc*) = *part*

@grd31 partition_mode(*part*) = **PM_NORMAL** ∨ partition_mode(*part*) = **PM_COLD_START** ∨ partition_mode(*part*) = **PM_WARM_START**

@grd40 current_partition_flag = TRUE

@grd200 *part* = current_partition

@grd41 current_process_flag = TRUE ⇒ *proc* ≠ current_process

@grd42 process_state(*proc*) ≠ **PS_Dormant**

@grd43 period_of_process(*proc*) = **INFINITE_TIME_VALUE**

@grd44 process_state(*proc*) =**PS_Suspend** ∨ process_state(*proc*) = **PS_WaitandSuspend**

@grd45 *reschedule* ∈ BOOL

@grd46 (process_state(*proc*) = **PS_Suspend** ⇒ *reschedule* = TRUE) ∧ (process_state(*proc*) = **PS_WaitandSuspend** ⇒ *reschedule* = FALSE)

//@grd47 process_state(proc) =PS_Suspend ∨ (process_state(proc) =PS_WaitandSuspend ∧ process_wait_type(proc)≠PROC_WAIT_TIMEOUT ∧ process_wait_type(proc)≠PROC_WAIT_OBJ) ⇒ newstate = PS_Ready

//@grd48 process_state(proc) =PS_WaitandSuspend ∧ process_wait_type(proc)=PROC_WAIT_TIMEOUT ∧ process_wait_type(proc)=PROC_WAIT_OBJ ⇒ newstate = PS_Waiting

//these two lines are from RESUME operation of ARINC653, the next two lines are correct

@grd47 process_state(*proc*) =**PS_Suspend** ⇒ *newstate* = **PS_Ready**

```
        @grd48 process_state(proc) =PS_WaitandSuspend ⇒ newstate = PS_Waiting

      then
        @act11 process_state(proc) ≔ newstate
        @act41 timeout_trigger :∣  (newstate = PS_Ready ⇒ timeout_trigger'=  {proc}⩤ timeout_trigger) ∧
(newstate ≠ PS_Ready ⇒ timeout_trigger'=timeout_trigger)
        @act42 need_reschedule :∣ (locklevel_of_partition(current_partition) =0 ∧  reschedule = TRUE ⇒
need_reschedule' = TRUE)
              ∧ (locklevel_of_partition(current_partition) > 0 ∨  reschedule = FALSE ⇒ need_reschedule' =
need_reschedule)
    end

    event stop_self refines stop_self
      any part proc newstate newlocklevel newproc resch
      where
        @grd01 part ∈ PARTITIONS
      @grd02 proc ∈ processes
      @grd03 newstate ∈ PROCESS_STATES
      @grd06 processes_of_partition(proc) = part
      @grd30 partition_mode(part) = PM_NORMAL
```

```
        @grd40 current_process_flag = TRUE ∧ current_partition_flag = TRUE
        @grd42 proc = current_process
        @grd43 (part∉dom(errorhandler_of_partition) ∨ proc ≠ errorhandler_of_partition(part)) ⇒ newlocklevel =
{part ↦ 0}
       @grd44 (part∈dom(errorhandler_of_partition) ∧ proc = errorhandler_of_partition(part)) ⇒ newlocklevel =
∅
        @grd45 part∈dom(errorhandler_of_partition) ∧ proc = errorhandler_of_partition(part) ∧
locklevel_of_partition(current_partition) > 0
                ∧ process_state(process_call_errorhandler(proc))≠PS_Dormant ⇒   (newproc =
process_call_errorhandler(proc) ∧ resch = FALSE) // If (current process is the error handler process and
preemption is disabled and previous process is not stopped) then return to previous process
        @grd46 ¬(part∈dom(errorhandler_of_partition) ∧ proc = errorhandler_of_partition(part) ∧
locklevel_of_partition(current_partition) > 0
                ∧ process_state(process_call_errorhandler(proc))≠PS_Dormant) ⇒   (newproc = proc ∧ resch =
TRUE) // If (current process is the error handler process and preemption is disabled and previous process is not
stopped) then return to previous process
         @grd47 newstate = PS_Dormant
    then
        @act11 process_state(proc) ≔ newstate
        @act41 current_process_flag :∣ (resch = FALSE ⇒ current_process_flag'=TRUE) ∧ (resch = TRUE ⇒
current_process_flag'=FALSE)
        @act42 locklevel_of_partition ≔ locklevel_of_partition   newlocklevel
        @act43 current_process ≔ newproc
        @act44 need_reschedule :∣ (resch = TRUE ⇒  need_reschedule' = TRUE) ∧ (resch = FALSE ⇒
need_reschedule' = need_reschedule)
  end

  event stop refines stop
    any part proc newstate newlocklevel
    where
        @grd01 part ∈ PARTITIONS
      @grd02 proc ∈ processes
      @grd03 newstate ∈ PROCESS_STATES
      @grd06 processes_of_partition(proc) = part
      @grd31 partition_mode(part) = PM_NORMAL ∨ partition_mode(part) = PM_COLD_START ∨
partition_mode(part) = PM_WARM_START

        @grd41 current_partition_flag = TRUE
        @grd42 current_process_flag = TRUE ⇒ proc ≠ current_process
        @grd200 part = current_partition
        @grd43 process_state(proc) ≠ PS_Dormant
        @grd45 (current_process_flag = TRUE ∧ part∈dom(errorhandler_of_partition) ∧ current_process =
errorhandler_of_partition(part)
```

```
                    ∧  proc = process_call_errorhandler(current_process))⇒  newlocklevel = {part ↦ 0}
        @grd46 ¬(current_process_flag = TRUE ∧ part∈dom(errorhandler_of_partition) ∧ current_process =
errorhandler_of_partition(part)
                    ∧  proc = process_call_errorhandler(current_process))⇒  newlocklevel = ∅
        @grd47 newstate = PS_Dormant
      then
        @act11 process_state(proc) ≔ newstate
        @act41 locklevel_of_partition ≔  locklevel_of_partition     newlocklevel
        @act42 timeout_trigger ≔  {proc}⩤ timeout_trigger
        //need to add statement of remove proc from deadlinetime, releasepoint, delaytime,errorhandler,
process_call_errorhandler
    end

    event start_aperiodprocess_instart // start an aperiodic process in COLD_START or WARM_START mode
    //extends start
    refines start
    any part proc newstate
      where
        @grd01 part ∈ PARTITIONS
      @grd02 proc ∈ processes
      @grd03 newstate ∈ PROCESS_STATES
```

```
  @grd06 processes_of_partition(proc) = part

    @grd41 current_partition_flag = TRUE
    @grd40 part = current_partition
    @grd43 partition_mode(part) = PM_COLD_START ∨  partition_mode(part) = PM_WARM_START
    @grd44 process_state(proc) = PS_Dormant
    @grd45 newstate = PS_Waiting
    @grd46 period_of_process(proc) = INFINITE_TIME_VALUE
  then
    @act11 process_state(proc) ≔ newstate
    @act41 currentpriority_of_process(proc) ≔ basepriority_of_process(proc)
    @act42 process_wait_type(proc) ≔ PROC_WAIT_PARTITIONNORMAL
end

event start_aperiodprocess_innormal // start an aperiodic process in NORMAL mode
//extends start
refines start
any part proc newstate
  where
    @grd01 part ∈ PARTITIONS
  @grd02 proc ∈ processes
  @grd03 newstate ∈ PROCESS_STATES
  @grd06 processes_of_partition(proc) = part

    @grd41 current_process_flag = TRUE ∧ current_partition_flag = TRUE
    @grd40 part = current_partition
    @grd43 partition_mode(part) = PM_NORMAL
    @grd44 process_state(proc) = PS_Dormant
    @grd45 newstate = PS_Ready
    @grd47 period_of_process(proc) = INFINITE_TIME_VALUE
  then
    @act11 process_state(proc) ≔ newstate
    @act03 currentpriority_of_process(proc) ≔ basepriority_of_process(proc)
    @act04 deadlinetime_of_process(proc) ≔ clock_tick∗ ONE_TICK_TIME + timecapacity_of_process(proc)
    @act05 need_reschedule :| (locklevel_of_partition(part) =0 ⇒ need_reschedule'=TRUE)
              ∧ (locklevel_of_partition(part) > 0 ⇒ need_reschedule'=need_reschedule)
end

event start_periodprocess_instart // start a periodic process in COLD_START or WARM_START mode
//extends start
refines start
any part proc newstate
```

```
  where
    @grd01 part ∈ PARTITIONS
   @grd02 proc ∈ processes
   @grd03 newstate ∈ PROCESS_STATES
   @grd06 processes_of_partition(proc) = part

    @grd41 current_partition_flag = TRUE
    @grd40 part = current_partition
    @grd42 partition_mode(part) = PM_COLD_START ∨ partition_mode(part) = PM_WARM_START
    @grd43 process_state(proc) = PS_Dormant
    @grd44 newstate = PS_Waiting
    @grd45 period_of_process(proc) ≠ INFINITE_TIME_VALUE
  then
    @act11 process_state(proc) ≔ newstate
    @act03 currentpriority_of_process(proc) ≔ basepriority_of_process(proc)
    @act42 process_wait_type(proc) ≔ PROC_WAIT_PARTITIONNORMAL
end

event start_periodprocess_innormal // start a periodic process in NORMAL mode
//extends start
refines start
```

```
  any part proc newstate fstrl
    where
      @grd01 part ∈ PARTITIONS
    @grd02 proc ∈ processes
    @grd03 newstate ∈ PROCESS_STATES
    @grd06 processes_of_partition(proc) = part

      @grd41 current_process_flag = TRUE ∧ current_partition_flag = TRUE
      @grd40 part = current_partition
      @grd43 partition_mode(part) = PM_NORMAL
      @grd44 process_state(proc) = PS_Dormant
      @grd45 newstate = PS_Waiting
      @grd46 fstrl ∈ ℕ1
      @grd47 period_of_process(proc) ≠ INFINITE_TIME_VALUE
      @grd48 ∃x,y,b·( ((x↦y)↦b)= firstperiodicprocstart_timeWindow_of_Partition(part)⇒ fstrl= ((clock_tick∗
ONE_TICK_TIME)÷majorFrame+1)∗majorFrame + x)
    then
      @act11 process_state(proc) ≔ newstate
      @act03 currentpriority_of_process(proc) ≔ basepriority_of_process(proc)
      @act05 releasepoint_of_process(proc) ≔ fstrl
      @act04 deadlinetime_of_process(proc) ≔  fstrl + timecapacity_of_process(proc)
    @act42 process_wait_type(proc) ≔ PROC_WAIT_PERIOD
end

event delaystart_aperiodprocess_instart
//extends delayed_start
refines delayed_start
  any part proc newstate delaytime
  where
    @grd01 part ∈ PARTITIONS
   @grd02 proc ∈ processes
   @grd03 newstate ∈ PROCESS_STATES
   @grd06 processes_of_partition(proc) = part

    @grd400 delaytime ∈ ℕ ∧ delaytime≠INFINITE_TIME_VALUE
    @grd41 current_partition_flag = TRUE
    @grd40 part = current_partition
    @grd43 partition_mode(part) = PM_COLD_START ∨  partition_mode(part) = PM_WARM_START
    @grd44 process_state(proc) = PS_Dormant
    @grd45 newstate = PS_Waiting
    @grd46 period_of_process(proc) = INFINITE_TIME_VALUE
  then
```

```
        @act11 process_state(proc) ≔ newstate
        @act41 currentpriority_of_process(proc) ≔ basepriority_of_process(proc)
        @act42 process_wait_type(proc)≔PROC_WAIT_DELAY
        @act43 delaytime_of_process(proc) ≔   delaytime
    end

    event delaystart_aperiodprocess_innormal //if delaytime=0, then immediately transit to READY, this is
  modelled in start_aperiod_process_whennormal
    //extends delayed_start
    //   any delaytime
    refines delayed_start
    any part proc newstate delaytime
      where
        @grd01 part ∈ PARTITIONS
      @grd02 proc ∈ processes
      @grd03 newstate ∈ PROCESS_STATES
      @grd06 processes_of_partition(proc) = part

        @grd40 delaytime > 0 ∧ delaytime≠INFINITE_TIME_VALUE
        @grd41 current_process_flag = TRUE ∧ current_partition_flag = TRUE
        @grd42 part = current_partition
        @grd43 partition_mode(part) = PM_NORMAL
        @grd44 process_state(proc) = PS_Dormant
        @grd45 newstate = PS_Waiting
        @grd47 period_of_process(proc) = INFINITE_TIME_VALUE
      then
        @act11 process_state(proc) ≔ newstate
        @act41 currentpriority_of_process(proc) ≔ basepriority_of_process(proc)
        @act42 deadlinetime_of_process(proc) ≔  clock_tick∗ ONE_TICK_TIME + timecapacity_of_process(proc) +
delaytime
        @act43 timeout_trigger ≔  timeout_trigger     {proc ↦ (PS_Ready↦ (delaytime +clock_tick ∗
ONE_TICK_TIME))}
        @act44 need_reschedule :∣ (locklevel_of_partition(part) =0 ⇒ need_reschedule'=TRUE)
               ∧ (locklevel_of_partition(part) > 0 ⇒ need_reschedule'=need_reschedule)
        @act45 process_wait_type(proc)≔PROC_WAIT_DELAY
        @act46 delaytime_of_process(proc) ≔    delaytime
    end

    event delaystart_periodprocess_instart
    //extends delayed_start
    refines delayed_start
      any part proc newstate delaytime
```

```
  where
    @grd01 part ∈ PARTITIONS
   @grd02 proc ∈ processes
   @grd03 newstate ∈ PROCESS_STATES
   @grd06 processes_of_partition(proc) = part

    @grd400 delaytime ∈ ℕ ∧ delaytime≠INFINITE_TIME_VALUE  ∧ delaytime < period_of_process(proc)
    @grd41 current_partition_flag = TRUE
    @grd40 part = current_partition
    @grd42 partition_mode(part) = PM_COLD_START ∨  partition_mode(part) = PM_WARM_START
    @grd43 process_state(proc) = PS_Dormant
    @grd44 newstate = PS_Waiting
    @grd45 period_of_process(proc) ≠ INFINITE_TIME_VALUE
  then
    @act11 process_state(proc) ≔ newstate
    @act41 currentpriority_of_process(proc) ≔ basepriority_of_process(proc)
    @act42 process_wait_type(proc)≔PROC_WAIT_DELAY
    @act43 delaytime_of_process(proc) ≔    delaytime
end

event delaystart_periodprocess_innormal
```

```
//extends delayed_start
refines delayed_start
  any part proc newstate delaytime fstrl
  where
    @grd01 part ∈ PARTITIONS
   @grd02 proc ∈ processes
   @grd03 newstate ∈ PROCESS_STATES
   @grd06 processes_of_partition(proc) = part

    @grd41 delaytime ∈ ℕ ∧ delaytime≠INFINITE_TIME_VALUE ∧ delaytime < period_of_process(proc)
    @grd42 current_process_flag = TRUE ∧ current_partition_flag = TRUE
    @grd40 part = current_partition
    @grd43 partition_mode(part) = PM_NORMAL
    @grd44 process_state(proc) = PS_Dormant
    @grd45 newstate = PS_Waiting
    @grd46 fstrl ∈ ℕ1
    @grd47 period_of_process(proc) ≠ INFINITE_TIME_VALUE
    @grd48 ∃x,y,b·( ((x↦y)↦b)= firstperiodicprocstart_timeWindow_of_Partition(part)⇒ fstrl= ((clock_tick∗
ONE_TICK_TIME)÷majorFrame+1)∗majorFrame + x)
  then
    @act11 process_state(proc) ≔ newstate
    @act41 currentpriority_of_process(proc) ≔ basepriority_of_process(proc)
    @act42 releasepoint_of_process(proc) ≔ fstrl + delaytime
    @act43 deadlinetime_of_process(proc) ≔ fstrl + delaytime + timecapacity_of_process(proc)
    @act45 process_wait_type(proc)≔PROC_WAIT_DELAY
    @act46 delaytime_of_process(proc) ≔ delaytime
end

event lock_preemption
  any part
  where
    @grd0 current_process_flag = TRUE ∧ current_partition_flag = TRUE
    @grd01 part ∈ PARTITIONS ∧ part = current_partition
    @grd02 part∈dom(errorhandler_of_partition) ⇒ current_process ≠ errorhandler_of_partition(part)
    @grd03 partition_mode(part) = PM_NORMAL
    @grd04 locklevel_of_partition(part) < MAX_LOCK_LEVEL
  then
    @act01 locklevel_of_partition(part) ≔ locklevel_of_partition(part) + 1
    @act02 preempter_of_partition(part) ≔ current_process
end

event unlock_preemption
```

```
    any part resched preempter
    where
      @grd0 current_process_flag = TRUE ∧ current_partition_flag = TRUE
      @grd01 part ∈ PARTITIONS ∧ part = current_partition
      @grd02 part∈dom(errorhandler_of_partition) ⇒ current_process ≠ errorhandler_of_partition(part)
      @grd03 partition_mode(part) = PM_NORMAL
      @grd04 locklevel_of_partition(part) > 0
      @grd05 locklevel_of_partition(part) = 1 ⇒ resched = TRUE
      @grd06 locklevel_of_partition(part) > 1 ⇒ resched = FALSE
      @grd07 locklevel_of_partition(part) = 1 ⇒ preempter = {(part ↦ current_process)}
      @grd08 locklevel_of_partition(part) > 1 ⇒ preempter = ∅
    then
      @act01 locklevel_of_partition(part) ≔ locklevel_of_partition(part) − 1
      @act02 need_reschedule :| (resched = TRUE ⇒ need_reschedule'=TRUE)
                 ∧ (resched = FALSE ⇒ need_reschedule'=need_reschedule)
      @act03 preempter_of_partition ≔ preempter_of_partition ∖ preempter
  end

  event get_my_id
    where
      @grd0 current_process_flag = TRUE ∧ current_partition_flag = TRUE
      @grd01 current_partition∈dom(errorhandler_of_partition) ⇒ current_process ≠
errorhandler_of_partition(current_partition)
  end

  event timed_wait extends timed_wait
    any delaytime tmout_trig wt
    where
      @grd40 delaytime ≠ INFINITE_TIME_VALUE
      @grd41 current_process_flag = TRUE ∧ current_partition_flag = TRUE
      @grd42 part = current_partition
      @grd43 proc = current_process
      @grd44 current_partition∈dom(errorhandler_of_partition) ⇒ current_process ≠
errorhandler_of_partition(current_partition)
      @grd45 locklevel_of_partition(current_partition) = 0
      @grd36 tmout_trig ∈ processes ⇸ (PROCESS_STATES × ℕ1)
      @grd37 (delaytime = 0 ⇒ (newstate = PS_Ready ∧ tmout_trig = ∅ ∧ wt=∅))
              ∧ (delaytime >0 ⇒ (newstate = PS_Waiting ∧ wt={proc↦PROC_WAIT_TIMEOUT} ∧ tmout_trig =
{current_process↦(PS_Ready↦ (delaytime +clock_tick ∗ ONE_TICK_TIME))}))
    then
      @act05 timeout_trigger ≔ timeout_trigger      tmout_trig
      @act04 process_wait_type ≔ process_wait_type      wt
```

```
        @act06 need_reschedule ≔ TRUE
        @act07 current_process_flag ≔ FALSE
    end

    event period_wait extends period_wait
      where
        @grd40 current_process_flag = TRUE ∧ current_partition_flag = TRUE
        @grd41 part = current_partition
        @grd42 proc = current_process
        @grd43 current_partition∈dom(errorhandler_of_partition) ⇒ current_process ≠
errorhandler_of_partition(current_partition)
        @grd44 locklevel_of_partition(current_partition) = 0
        @grd45 period_of_process(proc) ≠ INFINITE_TIME_VALUE // requesting process is not aperiodic
      then
        @act41 releasepoint_of_process(proc) ≔  releasepoint_of_process(proc) + period_of_process(proc)
        /* Next release point := process period plus previous release point;
           @act42 nextreleasepoint_of_process(proc) ≔  nextreleasepoint_of_process(proc) +
period_of_process(proc) */
        @act43 deadlinetime_of_process(proc) ≔   releasepoint_of_process(proc) + timecapacity_of_process(proc)
        @act44 need_reschedule ≔ TRUE
        @act45 current_process_flag ≔ FALSE
```

```
   end

   event get_time
   when
     @grd01 current_process_flag = TRUE ∧ current_partition_flag = TRUE
     @grd02 partition_mode(current_partition) = PM_NORMAL
   end

   event replenish
      any budget_time ddtm
      where
         @grd01 budget_time ∈ ℤ
         @grd02 current_process_flag = TRUE ∧ current_partition_flag = TRUE
         @grd03 partition_mode(current_partition) = PM_NORMAL
         @grd04 current_partition∈dom(errorhandler_of_partition) ⇒ current_process ≠
errorhandler_of_partition(current_partition)
         @grd05 period_of_process(current_process) ≠ INFINITE_TIME_VALUE
                    ∧ clock_tick ∗ ONE_TICK_TIME + budget_time ≤
releasepoint_of_process(current_process)+timecapacity_of_process(current_process)
         /* requesting process is not aperiodic
            requesting process is aperiodic or new deadline will not exceed next release point */
         @grd06 ddtm∈ℤ
         @grd07 budget_time > 0 ⇒ ddtm = clock_tick ∗ ONE_TICK_TIME + budget_time
         @grd08 (budget_time = INFINITE_TIME_VALUE  ∨
timecapacity_of_process(current_process)=INFINITE_TIME_VALUE) ⇒ ddtm = INFINITE_TIME_VALUE
    then
         @act01 deadlinetime_of_process(current_process) ≔ ddtm
   end

   event aperiodicprocess_finished extends process_finished
    where
         @grd40 current_partition_flag = TRUE ∧ current_process_flag = TRUE
         @grd41 part = current_partition
         @grd42 proc = current_process
         @grd44 newstate = PS_Dormant
         @grd45 period_of_process(proc) = INFINITE_TIME_VALUE
    then
         @act41 need_reschedule ≔ TRUE
         @act42 current_process_flag ≔ FALSE
   end

   event periodicprocess_finished extends process_finished
    where
         @grd40 current_partition_flag = TRUE  ∧  current_process_flag  = TRUE
         @grd41 part = current_partition
         @grd42 proc = current_process
         @grd44 newstate = PS_Waiting
         @grd45 period_of_process(proc)  ≠  INFINITE_TIME_VALUE
    then
         @act41 need_reschedule  ≔  TRUE
         //@act42 releasepoint_of_process(proc)  ≔   releasepoint_of_process(proc) + timecapacity_of_process(proc)
         @act43 process_wait_type(proc)  ≔  PROC_WAIT_PERIOD
         @act44 current_process_flag  ≔  FALSE
   end

   event time_out extends time_out
    any time// time is the absolute time ,not the "timeout"

    where

         @grd41 proc  ∈  dom(timeout_trigger)
         @grd42 newstate ↦  time = timeout_trigger(proc)
         //@grd43 process_state(proc)  = PS_Waiting
```

```
      @grd44 time ≥ (clock_tick − 1)∗ONE_TICK_TIME ∧ time ≤ clock_tick∗ONE_TICK_TIME // when the end
time is ellapsed one tick
      //@grd45 state = newstate
    then
      @act41 timeout_trigger ≔ timeout_trigger ∖{proc↦(newstate↦time)}
      @act42 process_wait_type ≔  {proc} ⩤ process_wait_type
  end

  event req_busy_resource extends req_busy_resource
  any wt timeout tmout_trig
  where
    @grd40 current_partition_flag = TRUE ∧ current_process_flag = TRUE
    @grd41 part = current_partition
    @grd42 proc = current_process
    @grd43 wt∈PROCESS_WAIT_TYPES ∧ (wt= PROC_WAIT_OBJ ∨ wt=PROC_WAIT_TIMEOUT)
    //@grd06 tmout >0 ∨ tmout = INFINITE_TIME_VALUE
    //this line is correct, the next line is from ARINC653
    @grd44 timeout ≠0
    @grd45 tmout_trig ∈ processes ⇸ (PROCESS_STATES × ℕ1)
    @grd46 (timeout = INFINITE_TIME_VALUE ⇒  tmout_trig = ∅)
           ∧ (timeout ≠INFINITE_TIME_VALUE ⇒ tmout_trig = {proc↦(PS_Ready↦ (timeout +clock_tick ∗
ONE_TICK_TIME))})
    @grd47 timeout≠INFINITE_TIME_VALUE ⇒ wt = PROC_WAIT_TIMEOUT
    @grd48 timeout =   INFINITE_TIME_VALUE ⇒ wt = PROC_WAIT_OBJ

  then
    @act41 need_reschedule ≔ TRUE
    @act42 current_process_flag ≔ FALSE
    @act43 process_wait_type(proc) ≔ wt
    @act05 timeout_trigger ≔ timeout_trigger        tmout_trig
  end

  event resource_become_available extends resource_become_available
  any resch
  where
    @grd40 process_wait_type(proc)=  PROC_WAIT_OBJ
    @grd41 resch∈BOOL
  then
    @act41 process_wait_type ≔   {proc}⩤process_wait_type
    @act42 timeout_trigger ≔ {proc}⩤timeout_trigger
    @act43 need_reschedule ≔ resch
  end
```

```
event resource_become_available2  extends resource_become_available2
any resch
where
  @grd40 ∀proc·(proc∈procs ⇒ process_wait_type(proc)=  PROC_WAIT_OBJ)
  @grd41 resch∈BOOL
then
  @act41 process_wait_type  ≔   procs⩤process_wait_type
  @act42 timeout_trigger  ≔  procs⩤timeout_trigger
  @act43 need_reschedule  ≔  resch
end

event periodicproc_reach_releasepoint    //has already finished execution of this period.
extends periodicproc_reach_releasepoint
where
     @grd11 period_of_process(proc)  ≠  INFINITE_TIME_VALUE
     @grd12 clock_tick∗ONE_TICK_TIME  ≥  releasepoint_of_process(proc)
     @grd13 process_state(proc)  =  PS_Waiting
     @grd14 process_wait_type(proc)  =  PROC_WAIT_PERIOD
then
  @act41 releasepoint_of_process(proc)  ≔   releasepoint_of_process(proc)  + period_of_process(proc)
```

```
    @act42 deadlinetime_of_process(proc) ≔ releasepoint_of_process(proc) + timecapacity_of_process(proc)
  end

  event coldstart_partition_fromidle extends coldstart_partition_fromidle
  then
    @act401 locklevel_of_partition(part) ≔ 1
  end

  event warmstart_partition_fromidle extends warmstart_partition_fromidle
  then
    @act401 locklevel_of_partition(part) ≔ 1
  end
end
```