Schedulability Analysis of Herschel/Planck Software using UPPAAL

Terma Case Study

Marius Mikučionis, Kim G. Larsen, Jacob I. Rasmussen, Arne Skou, Brian Nielsen, Steen Palm, Jan Pedersen, Poul Hougaard

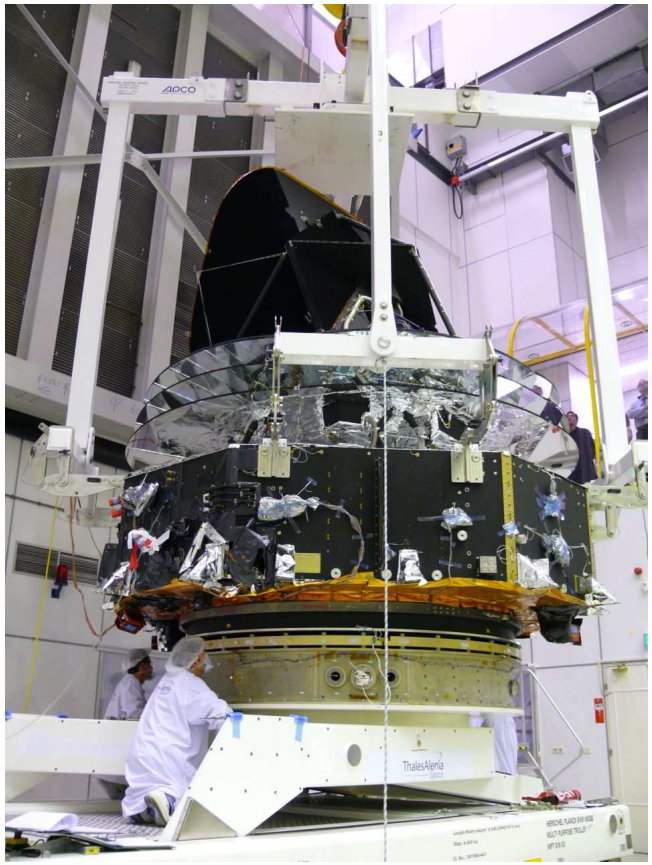
Department of Computer Science
Aalborg University

Terma A/S
2730 Herlev, Denmark

October 20, 2010

Outline

Herschel-Planck Scientific Mission at ESA



Subsystem: software for Attitude and Orbit Control System – ensures that satellite points to the right direction and is in the correct orbit.

Satellite Architecture

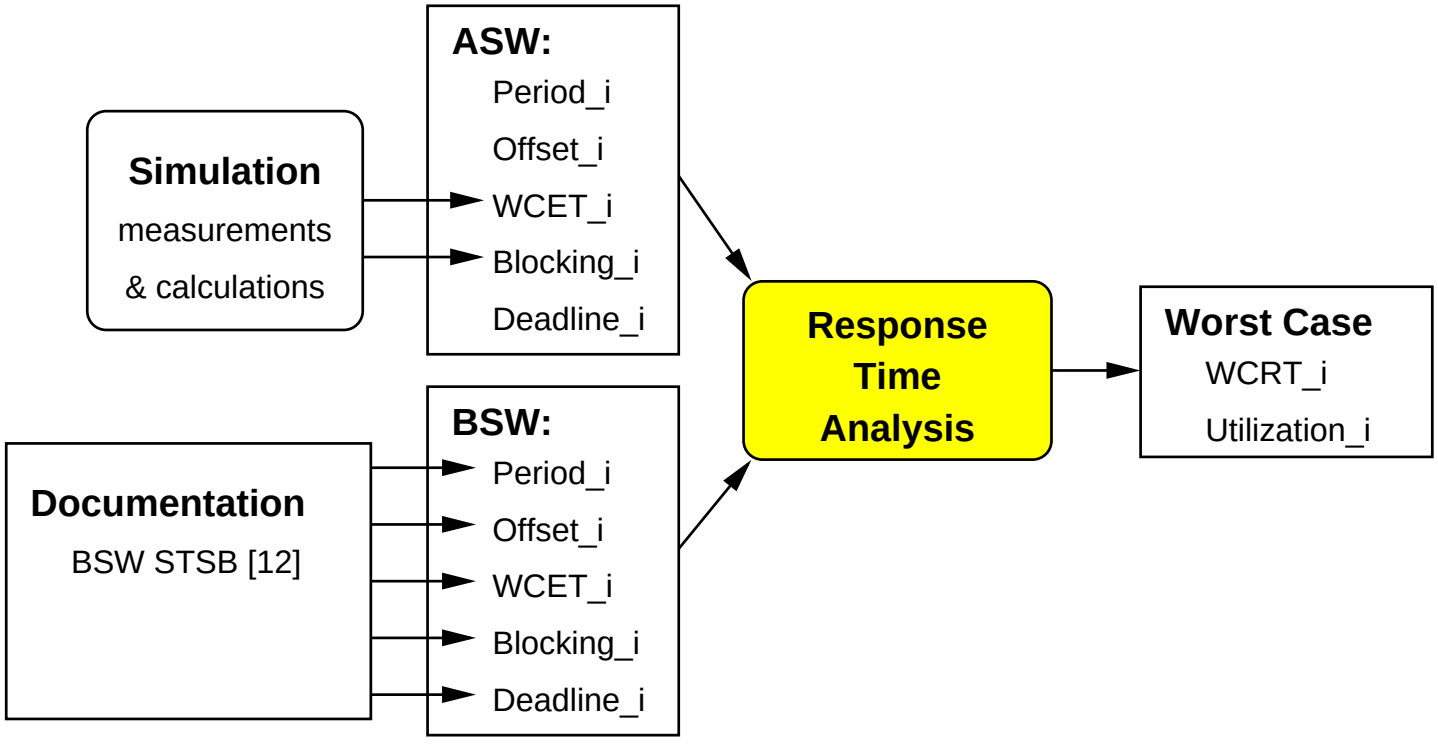
ASW	Application software performs attitude and orbit control, handles tele-commands, fault detection isolation and recovery.
BSW	Basic software is responsible for low level communication and scheduling periodic events.
RTEMS	Real-time operating system, fixed priority preemptive scheduler.
Hardware	Single processor, a few communication buses, sensors and actuators.

Resource Usage by Tasks in Herschel Events

Task \ Resource	Icb_R	Sgm_R	PmReq_R	Other_R
MainCycle		Y		
PrimaryFunctions	Y	Y	Y	
RCSControl				Y
Obt_P	Y			
StsMon_P	Y			
Sgm_P		Y		
Cmd_P	Y			
SecondaryFunc1		Y	Y	Y
SecondaryFunc2	Y			Y

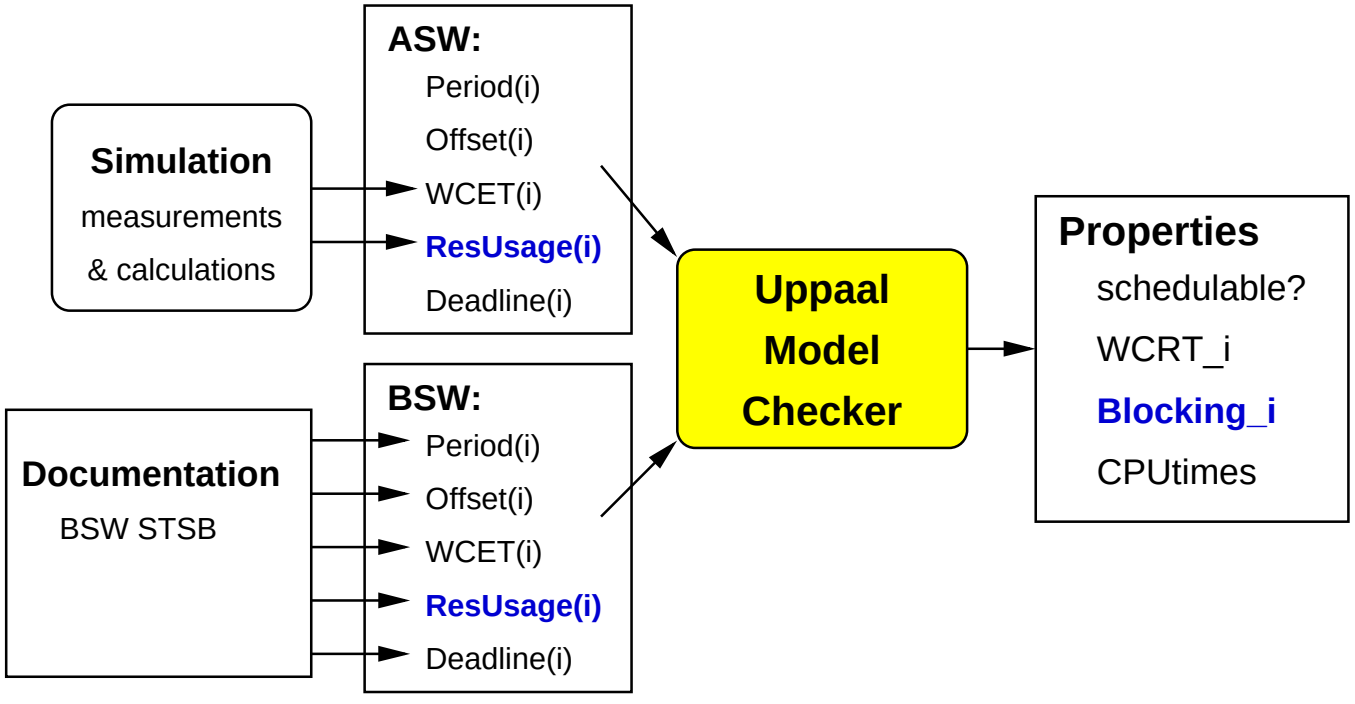
- Tasks are ordered by priority: highest priority first.
- ASW tasks are in **bold**, use priority **ceiling** protocol.
- BSW tasks are in plain, use priority **inheritance** protocol.

Work Flow of Classical Response Time Analysis



- Tasks are schedulable if $WCRT_{task} \leq \text{Deadline}_{task}$
- CPU requirements: $\leq$ 45% (Herschel), $\leq$ 50% (Planck).
- Analysis is split into 0-20ms and 20-250ms windows.
- Result: PrimaryF task may still exceed its deadline.
- Response time analysis is too conservative.

Schedulability Analysis using UPPAAL



- Model the tasks with concrete resource usage patterns, then query the model:

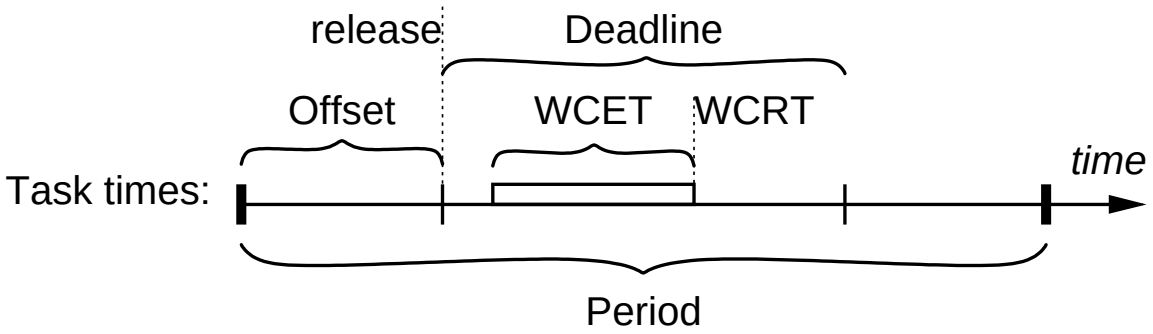
```
E<> error                    (schedulable?)
sup:  WCRT[0], WCRT[1], ...  WCRT[33]
sup:  Blocked[0], Blocked[1], ...  Blocked[33]
sup:  usedTime, idleTime, globalTime
```

Model-Checking Techniques Involved

- Task templates using:
  - Timed automata with clocks to express time constraints.
  - Stop-watches to track task progress.
  - Functions to implement resource sharing protocols.
  - Data structures to specify task flows.
- Symbolic exploration of entire model state space.
- Verification memory reduction via sweep-line method.
- Schedule simulation and visualization with Gantt chart.

[all of the above are implemented in UPPAAL]

Task Template Parameters



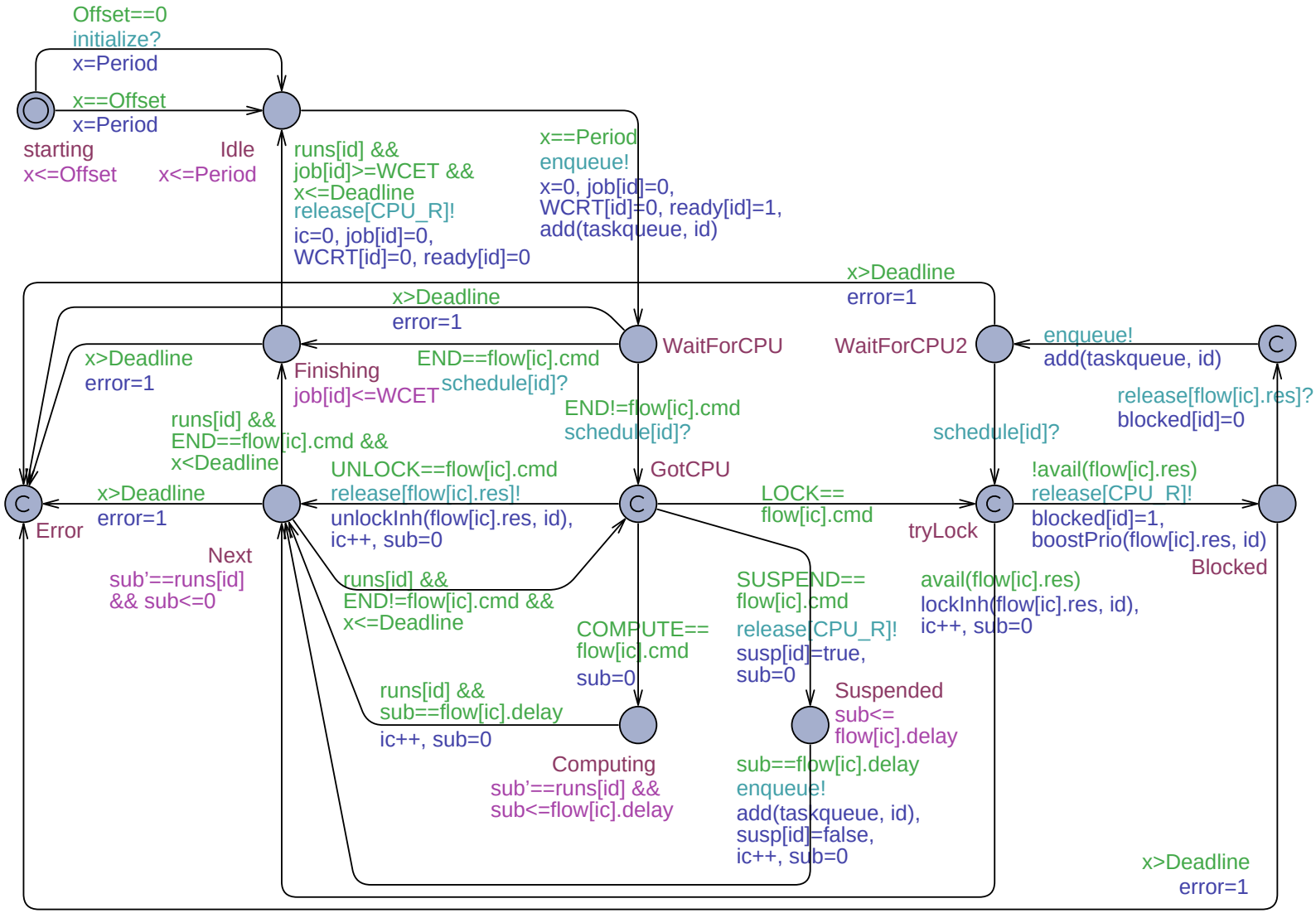
flow_t:	operation_t:	optype_t	resid_t	time_t
	operation_t:	optype_t	resid_t	time_t
	operation_t:	optype_t	resid_t	time_t
	...	...	...	...
	operation_t:	optype_t	resid_t	time_t

optype_t ::= END | COMPUTE | LOCK | UNLOCK | SUSPEND

resid_t ::= Icb_R | Sgm_R | PmReq_R | Other_R

time_t ::= int[0, 10000000]

Basic Software (BSW) Task Template



```
1  /** Check if the resource is available: */
2  bool avail(resid_t res) { return (owner[res]==0); }
3  void lockCeil(resid_t res, taskid_t task) {/** priority ceiling */
4      owner[res] = task; // mark resource occupied by the task
5      cprio[task] = ceiling[res]; // assume priority of resource
6  }
7  void unlockCeil(resid_t res, taskid_t task){/** priority ceiling */
8      owner[res] = 0; // mark the resource as released
9      cprio[task] = def_prio(task); // return to default priority
10 }
11 void lockInh(resid_t res, taskid_t task) {/** priority inheritance */
12     owner[res] = task; // mark the resource as occupied by the task
13 }
14 void unlockInh(resid_t res, taskid_t task) {/** priority inheritance */
15     owner[res] = 0; // mark the resource as released
16     cprio[task] = def_prio(task); // return to default priority
17 }
18 /** Boost the priority of resource owner based on priority inheritance: */
19 void boostPrio(resid_t res, taskid_t task) {
20     if (cprio[owner[res]] <= def_prio(task)) {
21         cprio[owner[res]] = def_prio(task)+1;
22         sort(taskqueue);
23     }
24 }
```

Primary Functions Timing Description

Each activity is described by CPU time / BSW service time followed by resource usage pattern:

Primary Functions	
- Data processing	20577/2521
	Icb_R(LNS: 2, LCS: 1200, LC: 1600, MaxLC: 800)
- Guidance	3440/0
- Attitude determination	3751/1777
	Sgm_R(LNS: 5, LCS: 121, LC: 1218, MaxLC: 236)
- PerformExtraChecks	42/0
- SCM controller	3479/2096
	PmReq_R(LNS: 4, LCS: 1650, LC: 3300, MaxLC: 3300)
- Command RWL	2752/85

- LNS – total number of times the CPU has been released
- LCS – total time the CPU has been released
- LC – total time the CPU has been locked
- MaxLC – the longest time the CPU has been locked

Primary Functions Flow in UPPAAL

```
1  const ASWFlow_t PF_f = { // Primary Functions:
2      { LOCK,    Icb_R, 0 },            //  0) ------ Data processing
3      { COMPUTE, CPU_R, 1600-1200 }, // 1) computing with Icb_R
4      { SUSPEND, CPU_R, 1200 },    //  2)   suspended with Icb_R
5      { UNLOCK, Icb_R, 0 },            //  3)
6      { COMPUTE, CPU_R, 20577-(1600-1200) }, // 4) computing w/o Icb_R
7      { COMPUTE, CPU_R, 3440 },    //  5) ------ Guidance
8      { LOCK,    Sgm_R, 0 },           //  6) ------ Attitude determination
9      { COMPUTE, CPU_R, 1218-121 }, // 7) computing with Sgm_R
10     { SUSPEND, CPU_R, 121 },     //  8)   suspended with Sgm_R
11     { UNLOCK, Sgm_R, 0 },            //  9)
12     { COMPUTE, CPU_R, 3751-(1218-121) }, //10) computing w/o Sgm_R
13     { COMPUTE, CPU_R, 42 },      //11) ------ Perform extra checks
14     { LOCK,    PmReq_R,0 },          //12) ------ SCM controller
15     { COMPUTE, CPU_R, 3300-1650 }, //13) computing with PmReq_R
16     { SUSPEND, CPU_R, 1650 },    //14)   suspended with PmReq_R
17     { UNLOCK, PmReq_R, 0 },          //15)
18     { COMPUTE, CPU_R, 3479-(3300-1650) },//16) comp. w/o PmReq_R
19     { COMPUTE, CPU_R, 2752 },    //17) ------ Command RWL
20     { END,     CPU_R, 0 }            //18) finished
21 };
```

WCRT and Blocking Times

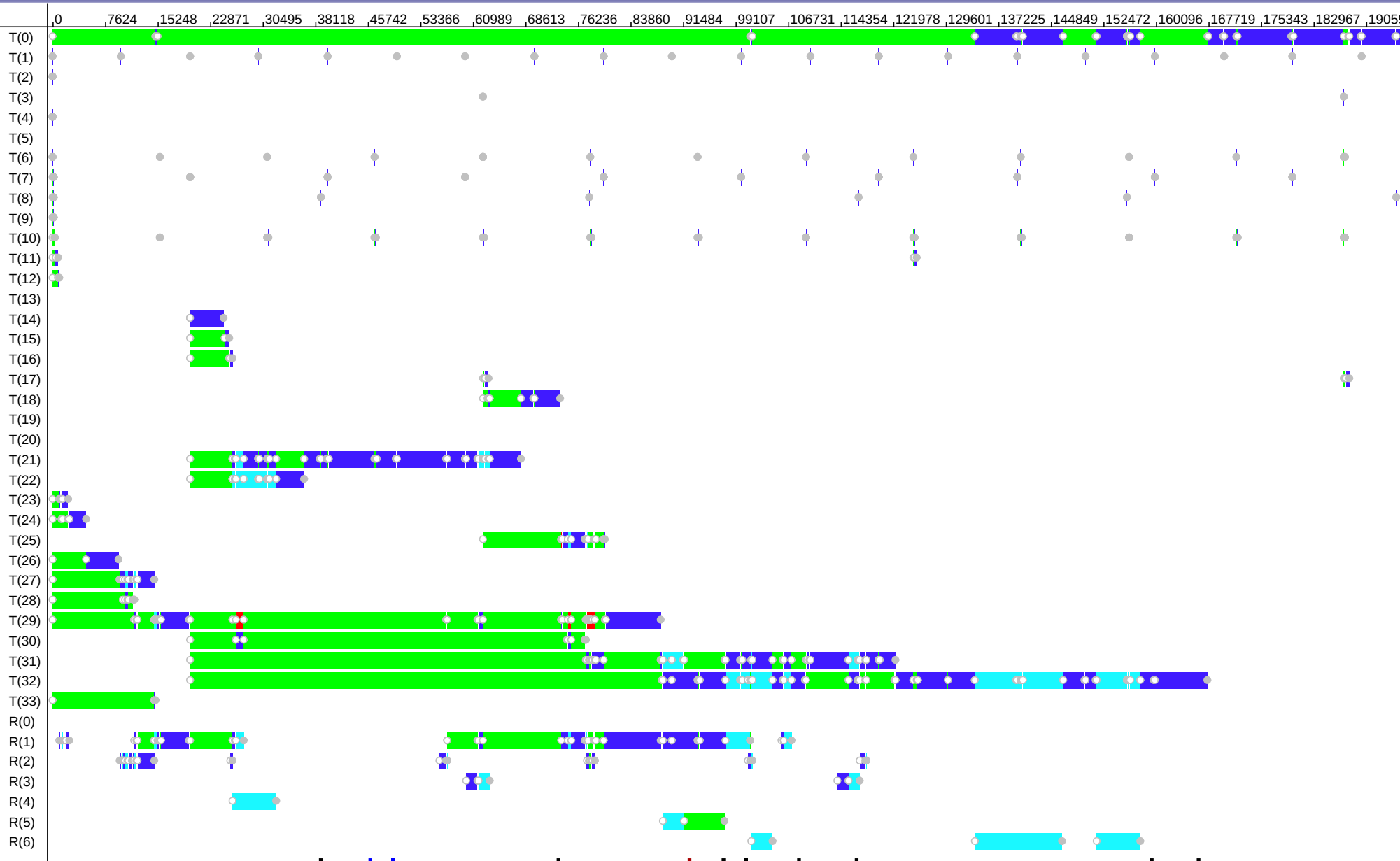
ID	Task	Specification			Blocking times			WCRT	
		Period	WCET	Deadline	Terma	UPPAAL	Diff	Terma	UPPAAL
1	RTEMS_RTC	10.000	0.013	1.000	0.035	0	0.035	0.050	0.013
2	AswSync_SyncPulseIsr	250.000	0.070	1.000	0.035	0	0.035	0.120	0.083
3	Hk_SamplerIsr	125.000	0.070	1.000	0.035	0	0.035	0.120	0.070
4	SwCyc_CycStartIsr	250.000	0.200	1.000	0.035	0	0.035	0.320	0.103
5	SwCyc_CycEndIsr	250.000	0.100	1.000	0.035	0	0.035	0.220	0.113
6	Rt1553_Isr	15.625	0.070	1.000	0.035	0	0.035	0.290	0.173
7	Bc1553_Isr	20.000	0.070	1.000	0.035	0	0.035	0.360	0.243
8	Spw_Isr	39.000	0.070	2.000	0.035	0	0.035	0.430	0.313
9	Obdh_Isr	250.000	0.070	2.000	0.035	0	0.035	0.500	0.383
10	RtSdb_P_1	15.625	0.150	15.625	3.650	0	3.650	4.330	0.533
11	RtSdb_P_2	125.000	0.400	15.625	3.650	0	3.650	4.870	0.933
12	RtSdb_P_3	250.000	0.170	15.625	3.650	0	3.650	5.110	1.103
14	**FdirEvents**	250.000	5.000	230.220	0.720	0	0.720	7.180	5.153
15	**NominalEvents_1**	250.000	0.720	230.220	0.720	0	0.720	7.900	5.873
16	**MainCycle**	250.000	0.400	230.220	0.720	0	0.720	8.370	6.273
17	HkSampler_P_2	125.000	0.500	62.500	3.650	0	3.650	11.960	5.380
18	HkSampler_P_1	250.000	6.000	62.500	3.650	0	3.650	18.460	11.615
19	Acb_P	250.000	6.000	50.000	3.650	0	3.650	24.680	6.473
20	IoCyc_P	250.000	3.000	50.000	3.650	0	3.650	27.820	9.473
21	**PrimaryF**	250.000	34.050	**59.600**	5.770	0.966	4.804	**65.470**	**54.115**
22	**RCSControlF**	250.000	4.070	239.600	12.120	0	12.120	76.040	53.994
23	Obt_P	1000.000	1.100	100.000	9.630	0	9.630	74.720	2.503
24	Hk_P	250.000	2.750	250.000	1.035	0	1.035	6.800	4.953
25	StsMon_P	250.000	3.300	125.000	16.070	0.822	15.248	85.050	17.863
26	TmGen_P	250.000	4.860	250.000	4.260	0	4.260	77.650	9.813
27	Sgm_P	250.000	4.020	250.000	1.040	0	1.040	18.680	14.796
28	TcRouter_P	250.000	0.500	250.000	1.035	0	1.035	19.310	11.896
29	Cmd_P	250.000	14.000	250.000	26.110	1.262	24.848	114.920	94.346
30	**NominalEvents_2**	250.000	1.780	230.220	12.480	0	12.480	102.760	65.177
31	**SecondaryF_1**	250.000	20.960	189.600	27.650	0	27.650	141.550	110.666
32	**SecondaryF_2**	250.000	39.690	230.220	48.450	0	48.450	204.050	154.556
33	Bkgnd_P	250.000	0.200	250.000	0.000	0	0.000	154.090	15.046

Verification Resources and System CPU Utilization

cycle limit	Uppaal resources			Herschel CPU utilization				
	CPU, s	Mem, KB	States, #	Idle, μs	Used, μs	Global, μs	Sum, μs	Used, %
1	465.2	60288	173456	91225	160015	250000	251240	0.640060
2	470.1	59536	174234	182380	318790	500000	501170	0.637580
3	461.0	58656	175228	273535	477705	750000	751240	0.636940
4	474.5	58792	176266	363590	636480	1000000	1000070	0.636480
6	474.6	58796	178432	545900	955270	1500000	1501170	0.636847
8	912.3	58856	352365	727110	1272960	2000000	2000070	0.636480
13	507.7	58796	186091	1181855	2069385	3250000	3251240	0.636734
16	1759.0	58728	704551	1454220	2545850	4000000	4000070	0.636463
26	541.9	58112	200364	2363640	4137530	6500000	6501170	0.636543
32	3484.0	75520	1408943	2908370	5091700	8000000	8000070	0.636463
39	583.5	74568	214657	3545425	6205745	9750000	9751170	0.636487
64	7030.0	91776	2817704	5816740	10183330	16000000	16000070	0.636458
78	652.2	74768	257582	7089680	12411420	19500000	19501100	0.636483
128	14149.4	141448	5635227	11633480	20366590	32000000	32000070	0.636456
156	789.4	91204	343402	14178260	24821740	39000000	39000000	**0.636455**
256	23219.4	224440	11270279	23266890	40733180	64000000	64000070	0.636456
312	1824.6	124892	686788	28356520	49643480	78000000	78000000	0.636455
512	49202.2	390428	22540388	46533780	81466290	128000000	128000070	0.636455
624	3734.7	207728	1373560	56713040	99286960	156000000	156000000	0.636455

Terma RTA	worst cycle CPU utilization	62.4%
UPPAAL	worst cycle CPU utilization	64.0060%
UPPAAL	average utilization	63.6455%

Gantt Chart of the First Cycle



green=ready, blue=running, red=blocked, cyan=suspended

Conclusions

- Schedulability analysis using UPPAAL:
  - Reusable and customizable task templates.
  - *Blocking* times and WCRTs can be derived from the model.
  - WCRTs of all tasks are more optimistic than in RTA.
  - There are very few blocking times and they are short.
  - PrimaryF meets deadline (59.6ms) with WCRT=54.1ms (65.5ms in RTA).
  - Herschel event mode is **schedulable**.
- UPPAAL verification for schedulability:
  - can be **scaled** using sweep-line method,
  - takes up to **2min** to verify schedulability of 32 task system,
  - takes up to **8min** to find all WCRTs and CPU utilization.
- In addition, it is possible to:
  - simulate the system model and examine details,
  - render a Gantt chart, validate and inspect visually.

Future Work

Is the model fair with respect to the actual system?

- Is the modeled resource usage time-line realistic?
- Challenge: sporadic tasks imply lots of non-determinism.
- Operation primitives can be expanded.
- Context switch time can/should be modeled explicitly in the Scheduler.

Margin analysis

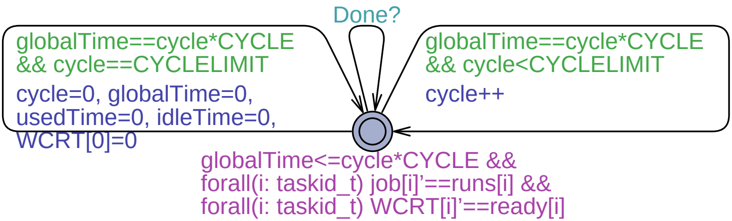
The End

Thank You!

Gantt Chart Declaration

```
1  gantt {
2    T(i:taskid_t):
3      (ready[i] && !runs[i]) -> 1,// green: ready
4      (ready[i] && runs[i]) -> 2, // blue:   running
5      (blocked[i]) -> 0,          // red:    blocked
6      susp[i] -> 9;               // cyan:   suspended
7    R(i:resid_t):
8      (owner[i]>0 && runs[owner[i]]) -> 2,  // blue: locked and actively used
9      (owner[i]>0 && !runs[owner[i]] && !susp[owner[i]]) -> 1, // green: locked
           but preempted
10     (owner[i]>0 && susp[owner[i]]) -> 9;  // cyan: locked and suspended
11 }
```

Sweep-Line Method via Progress Measure




```
const int CYCLE = 250*1000;
const int CYCLELIMIT = 3;
int cycle = 1;
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
system Scheduler, Bkgnd_P < secondF_2 < secondF_1 < NominalEvents_2 <
    Cmd_P < TcRouter_P < Sgm_P < TmGen_P < StsMon_P < Hk_P <
    Obt_P < rCSControlF < primaryF < IoCyc_P < Acb_P < HkSampler_P_1 <
    HkSampler_P_2 < mainCycle < NominalEvents_1 < FdirEvents <
    RtSdb_P_3 < RtSdb_P_2 < RtSdb_P_1 < Obdh_Isr < Spw_Isr <
    Bc1553_Isr < Rt1553_Isr < SwCyc_CycEndIsr < SwCyc_CycStartIsr <
    Hk_SamplerIsr < AswSync_SyncPulseIsr < RTEMS_RTC, IdleTask, Global;
progress { cycle; }
```