Université de Technologie de Compiègne

MI11

Rapport de TP : Modélisation AADL et ordonnancement

Clément BLANQUET et Rafik CHENNOUF

Juin 2017

Sommaire

Table des figures

Chapitre 1

Exercice 3 : Moniteur médical multi-paramètres

Question 1

On peut déterminer la priorité selon la plus petite période (Rate Monotonic), le plus petit temps de réponse dynamique (Earliest Deadline First) ou encore la plus petite laxité dynamique (Least Laxity First). On a décidé de choisir comme premier critère la période de la tâche (plus elle est petite, plus la tâche est prioritaire => Rate Monotonic) et en second critère, c'est à dire en cas d'égalité, la durée d'exécution de la tâche (plus elle est grande, plus la tâche est prioritaire).

Les paramètres et les priorités des différentes tâches à ordonnancer sont donnés dans le tableau ci-dessous :

Tâches	Période (ms)	Capacité (ms)	Priorité
getPA	10	1	3
getSO2	10	2	2
getECG	40	4	5
dpy	6	1	1
check	12	2	4

La tâche la plus prioritaire est la tâche **dpy** car elle a la période la plus petite (6 ms). Les tâches de priorité 2 et 3 sont respectivement les tâches **getSO2** et **getPA**. Ces deux tâches ayant la même période (10 ms), la tâche la plus prioritaire est celle avec une capacité supérieure, c'est à dire ici la tâche **getSO2**. La tâche de priorité 4 est la tâche **check** avec une période de 12 ms. Enfin, la tâche la moins prioritaire est la tâche **getECG** car elle possède la période la plus élevée.

Question 2

Nous avons implémenté les paramètres précédents dans le fichier AADL *health_monitor.aadl* comme ci-dessous :

```
-- getPA
thread implementation read_sensor.PA
  properties
    Dispatch_Protocol => Periodic ;
    Period => 10 ms;
    Compute_Execution_Time => 1 ms .. 1 ms;
    Deadline => 10 ms;
    Cheddar_Properties::Dispatch_Absolute_Time => 0 ms;
    Cheddar_Properties::Fixed_Priority => 253;
end read_sensor.PA;

-- getECG
thread implementation read_sensor.ECG
  properties
    Dispatch_Protocol => Periodic ;
    Period => 40 ms;
    Compute_Execution_Time => 4 ms .. 4 ms;
    Deadline => 40 ms;
    Cheddar_Properties::Dispatch_Absolute_Time => 0 ms;
    Cheddar_Properties::Fixed_Priority => 251;
end read_sensor.ECG;

-- getSO2
thread implementation read_sensor.SO2
  properties
    Dispatch_Protocol => Periodic ;
    Period => 10 ms;
    Compute_Execution_Time => 2 ms .. 2 ms;
    Deadline => 10 ms;
    Cheddar_Properties::Dispatch_Absolute_Time => 0 ms;
    Cheddar_Properties::Fixed_Priority => 254;
end read_sensor.SO2;

-- check
thread implementation check_bounds.impl
  properties
    Dispatch_Protocol => Periodic ;
    Period => 12 ms;
    Compute_Execution_Time => 2 ms .. 2 ms;
    Deadline => 12 ms;
    Cheddar_Properties::Dispatch_Absolute_Time => 0 ms;
    Cheddar_Properties::Fixed_Priority => 252;
end check_bounds.impl;

-- dpy
thread implementation display.impl
  properties
    Dispatch_Protocol => Periodic ;
```

```
49      Period => 6 ms;
        Compute_Execution_Time => 1 ms .. 1 ms;
51      Deadline => 6 ms;
        Cheddar_Properties::Dispatch_Absolute_Time => 0 ms;
53      Cheddar_Properties::Fixed_Priority => 255;
end display.impl;
```

Question 3

FIGURE 1.1 – Résultat de l'ordonnancement avec Cheddar

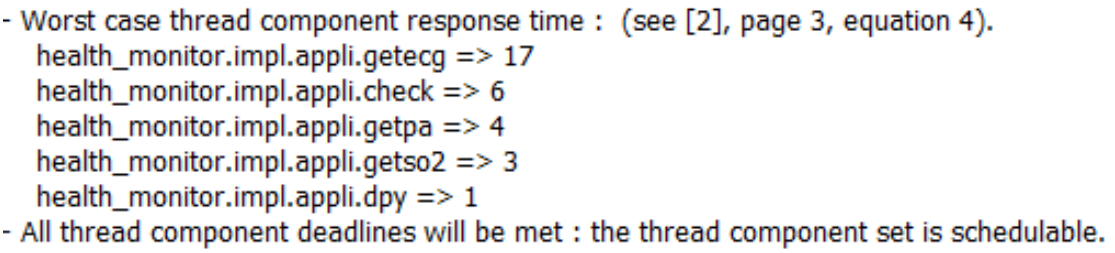

FIGURE 1.2 – Tâches ordonnançables

On voit bien sur Cheddar que toutes nos tâches sont ordonnançables.

Question 4

En ajoutant un second processeur et en modifiant les paramètres des tâches en fonction de ce qui est donné dans le nouveau tableau, on remarque que les tâches ne respectent plus leurs contraintes temporelles. En effet, d'après Cheddar, la tâche **getPA** ne s'exécute plus du tout comme on peut le voir sur la copie d'écran suivante :

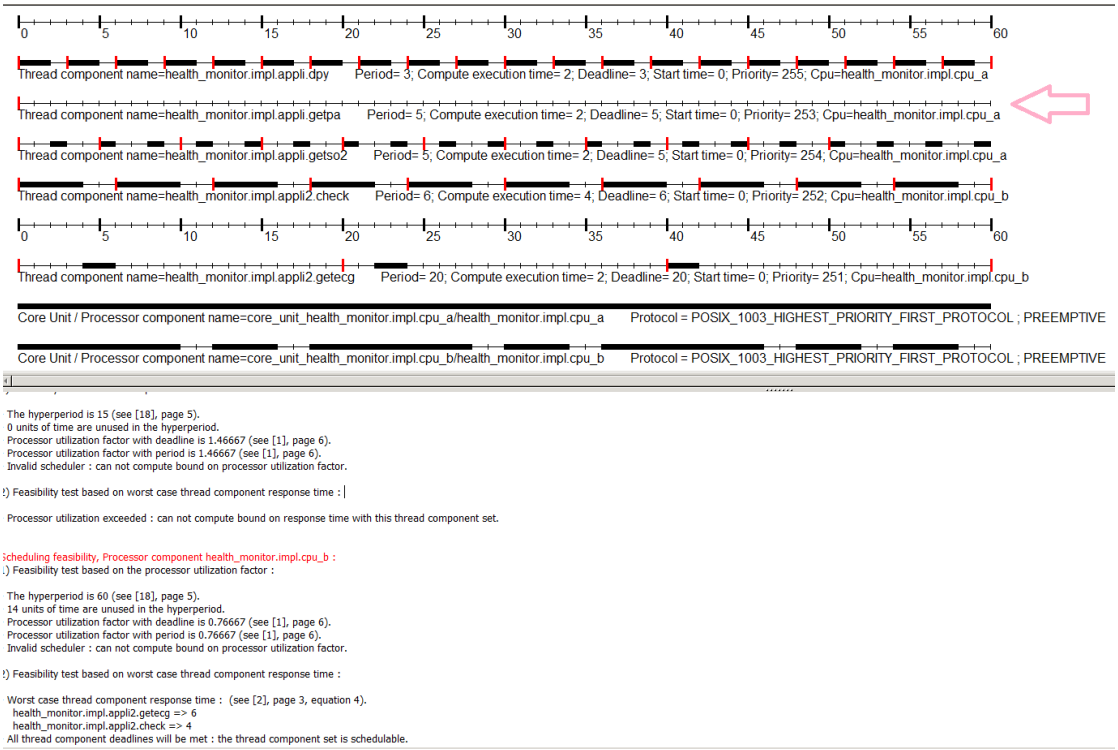

FIGURE 1.3 – Résultat de l'ordonnancement avec Cheddar avec deux processeurs

Ceci est dû au fait que les tâches **getPA** et **getSO2** ont exactement la même période et la même capacité. Comme nous avons décidé arbitrairement que la tâche **getSO2** est prioritaire sur la tâche **getPA**, cette tâche ne pourra jamais s'exécuter puisqu'à chaque fois que la tâche **dpy** aura terminé son exécution, c'est la tâche **getSO2** qui reprendra la main.

Question 5

Étant donné que la tâche **getPA** ne peut pas s'exécuter et que le processeur b est deux fois plus rapide que le processeur a, une solution simple serait de déplacer la tâche **getPA** vers le processeur b. Ceci permet d'alléger la charge de travail du processeur a et de permettre aux tâches **getPA** et **getSO2** d'être complètement indépendantes. Il faut aussi diviser la capacité initiale de la tâche **getPA** par 2 ce qui donne 1.

Tâches	Processeur	Capacité	Période
check	b	4	6
getECG	b	2	20
getPA	b	1	5
dpy	a	2	3
getSO2	a	2	5

Une deuxième solution aurait été de former un jeu de tâches harmoniques comme dans l'exercice 1. On cherche dans notre tableau des tâches qui possèdent des périodes multiples entre elles et on les regroupe sur le même processeur. Dans notre cas, on aurait pu regrouper les tâches **check** et **dpy** de périodes 6 et 3 sur le processeur b et les tâches **getPA**, **getSO2** et **getECG** de périodes 5, 5 et 20 sur le processeur a.

Question 6

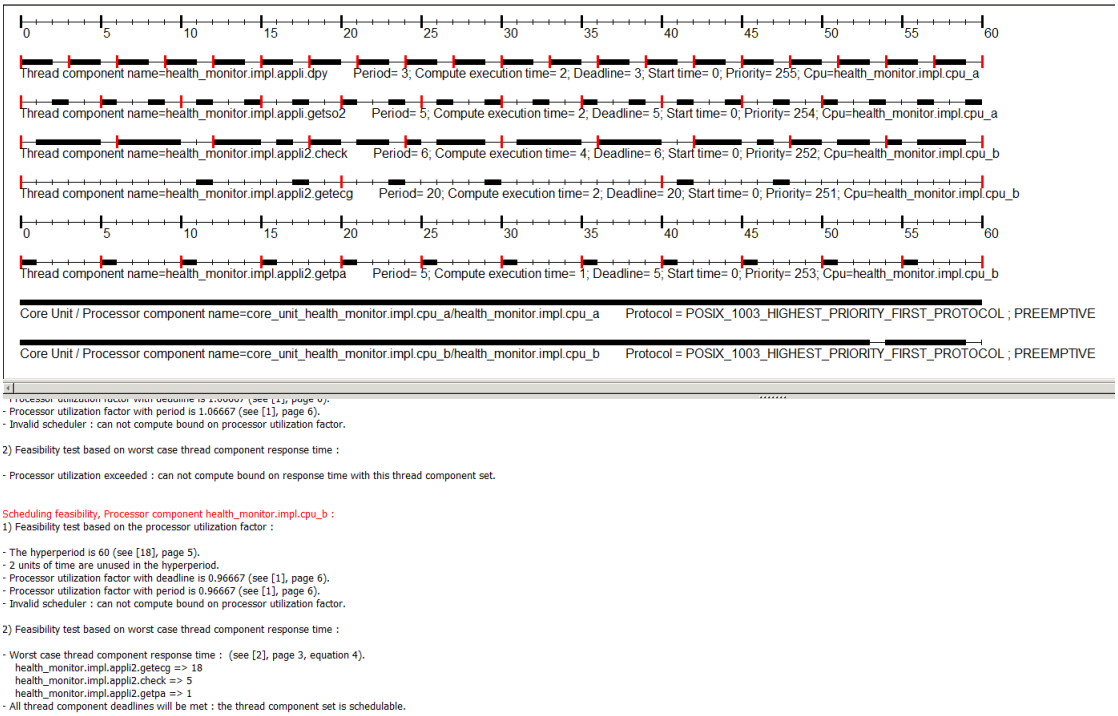

FIGURE 1.4 – Résultat de l'ordonnancement avec Cheddar avec deux processeurs sachant que la tâche **getPA** s'exécute sur b

On voit que cette solution permet à toutes les tâches de s'exécuter. Ci-dessous le code AADL complet :

```
1  ------------------------
   -- Composants matériels
3  ------------------------

5  -- Bus

7  -- Les capteurs sont sur bus CAN
   bus CAN
9  end CAN;

11 -- L''afficheur est sur bus LVDS
   bus LVDS
13 end LVDS;

15
```

```
-- Capteurs

-- Capteur saturation O2
device SO2_sensor
  features
    SO2 : out data port;
    iobus : requires bus access CAN;
end SO2_sensor;

device implementation SO2_sensor.impl
end SO2_sensor.impl;

-- Capteur électrocardiogramme
device ECG_sensor
  features
    ECG : out data port;
    iobus : requires bus access CAN;
end ECG_sensor;

device implementation ECG_sensor.impl
end ECG_sensor.impl;

-- Capteur Pression artérielle
device PA_sensor
  features
    PA : out data port;
    iobus : requires bus access CAN;
end PA_sensor;

device implementation PA_sensor.impl
end PA_sensor.impl;

-- Autres périphériques
device LCD_display
  features
    fb : in data port;          -- framebuffer
    lvds : requires bus access LVDS;  -- lcd bus
end LCD_display;

device Alarm
  features
    trigger : in event data port;
    iobus : requires bus access CAN;
end Alarm;

-- Processeur
processor armadeus
  features
    iobus : requires bus access CAN;
```

```
65      lvds : requires bus access LVDS;
    end armadeus;
67
    processor implementation armadeus.impl
69    properties
        Scheduling_Protocol => POSIX_1003_HIGHEST_PRIORITY_FIRST_PROTOCOL
        ;
71      Cheddar_Properties::Preemptive_Scheduler => True ;
    end armadeus.impl;
73
    processor armadeus2
75    features
        iobus : requires bus access CAN;
77      lvds : requires bus access LVDS;
    end armadeus2;
79
    processor implementation armadeus2.impl -- NOUVEAU CPU
81    properties
        Scheduling_Protocol => POSIX_1003_HIGHEST_PRIORITY_FIRST_PROTOCOL
        ;
83      Cheddar_Properties::Preemptive_Scheduler => True ;
    end armadeus2.impl;
85
    ------------------------
87  -- Composants logiciels
    ------------------------
89  thread read_sensor
      features
91      raw : in data port;
        cooked : out data port;
93  end read_sensor;

95  thread implementation read_sensor.PA
      properties
97      Dispatch_Protocol => Periodic ;
        Period => 5 ms;
99      Compute_Execution_Time => 1 ms .. 1 ms;
        Deadline => 5 ms;
101     Cheddar_Properties::Dispatch_Absolute_Time => 0 ms;
        Cheddar_Properties::Fixed_Priority => 253;
103 end read_sensor.PA;

105 thread implementation read_sensor.ECG
      properties
107     Dispatch_Protocol => Periodic ;
        Period => 20 ms;
109     Compute_Execution_Time => 2 ms .. 2 ms;
        Deadline => 20 ms;
111     Cheddar_Properties::Dispatch_Absolute_Time => 0 ms;
```

```
    Cheddar_Properties::Fixed_Priority => 251;
end read_sensor.ECG;

thread implementation read_sensor.SO2
  properties
    Dispatch_Protocol => Periodic ;
    Period => 5 ms;
    Compute_Execution_Time => 2 ms .. 2 ms;
    Deadline => 5 ms;
    Cheddar_Properties::Dispatch_Absolute_Time => 0 ms;
    Cheddar_Properties::Fixed_Priority => 254;
end read_sensor.SO2;

thread check_bounds
  features
    PA : in data port;
    ECG : in data port;
    SO2 : in data port;
    alarm : out event data port;
end check_bounds;

thread implementation check_bounds.impl
  properties
    Dispatch_Protocol => Periodic ;
    Period => 6 ms;
    Compute_Execution_Time => 4 ms .. 4 ms;
    Deadline => 6 ms;
    Cheddar_Properties::Dispatch_Absolute_Time => 0 ms;
    Cheddar_Properties::Fixed_Priority => 252;
end check_bounds.impl;

thread display
  features
    PA : in data port;
    ECG : in data port;
    SO2 : in data port;
    fb : out data port;
end display;

thread implementation display.impl
  properties
    Dispatch_Protocol => Periodic ;
    Period => 3 ms;
    Compute_Execution_Time => 2 ms .. 2 ms;
    Deadline => 3 ms;
    Cheddar_Properties::Dispatch_Absolute_Time => 0 ms;
    Cheddar_Properties::Fixed_Priority => 255;
end display.impl;
```

```
process health_monitoring
  features
    PA : in data port;
    ECG : in data port;
    SO2 : in data port;
    fb : out data port;
    alarm : out event data port;
end health_monitoring;


process implementation health_monitoring.impl
  subcomponents
    getSO2 : thread read_sensor.SO2;
    dpy : thread display.impl;
end health_monitoring.impl;


process health_monitoring2
  features
    PA : in data port;
    --ECG : in data port;
    SO2 : in data port;
    fb : out data port;
    alarm : out event data port;
end health_monitoring2;


process implementation health_monitoring2.impl -- NOUVELLE
    APPLICATION
  subcomponents
    getECG : thread read_sensor.ECG;
    check : thread check_bounds.impl;
    getPA : thread read_sensor.PA;
end health_monitoring2.impl;

--------------------
-- système complet
--------------------
system health_monitor
end health_monitor;

system implementation health_monitor.impl
  subcomponents
    -- Les bus
    can : bus CAN;
    lvds : bus LVDS;

    -- Les processeurs
    cpu_a : processor armadeus.impl;
```

```
209     cpu_b : processor armadeus2.impl; -- NOUVEAU CPU

211     -- Les capteurs
        SO2_sense : device SO2_sensor;
213     PA_sense : device PA_sensor;
        ECG_sense : device ECG_sensor;
215
        -- Les périphériques de sortie
217     lcd : device LCD_display;
        alarm : device Alarm;
219
        -- L''application
221     appli : process health_monitoring.impl;
        appli2 : process health_monitoring2.impl;    -- NOUVELLE
       APPLICATION
223


225   connections
        -- connexions de bus
227     bus access can -> SO2_sense.iobus;
        bus access can -> PA_sense.iobus;
229     bus access can -> ECG_sense.iobus;
        bus access can -> alarm.iobus;
231     bus access can -> cpu_a.iobus;

233     bus access lvds -> lcd.lvds;
        bus access lvds -> cpu_a.lvds;
235
        -- connexions de ports
237     so2_conn : data port SO2_sense.SO2 -> appli.SO2;
        pa_conn : data port PA_sense.PA -> appli.PA;
239     ecg_conn : data port ECG_sense.ECG -> appli.ECG;
        fb_conn : data port appli.fb -> lcd.fb;
241     alarm_conn : event data port appli.alarm -> alarm.trigger;

243   properties
        -- Allocation logiciel - plateforme d''exécution
245     Actual_Processor_Binding => reference cpu_a applies to appli;
        Actual_Processor_Binding => reference cpu_b applies to appli2;
       -- NOUVEAU CPU
247
        Actual_Connection_Binding => reference can applies to so2_conn;
249     Actual_Connection_Binding => reference can applies to pa_conn;
        Actual_Connection_Binding => reference can applies to ecg_conn;
251     Actual_Connection_Binding => reference can applies to alarm_conn;
        Actual_Connection_Binding => reference lvds applies to fb_conn;
253 end health_monitor.impl;
```