Application Use Case for Multi-core Schedulability

Technical Note

Fotios Gioulekas, CERTH

Peter Poplavko, Verimag

Alexandros Zerzelidis, CERTH

Panagiotis Katsaros, CERTH

Pedro Palomo, DEIMOS SPACE

December 2017

Abstract

In this document, we describe the procedures and development phases that took place during the porting of the Guidance, Navigation and Control application (GNC app) provided by Elecnor Deimos-Space S.L.U. onto LEON4 multi-core processor which is facilitated by the next-generation multicore platform (NGMP). During this design process, we followed the MoSATT-CMP design methodology and its associated tool-chain. Test-results obtained throughout this process are also shown.

Table of Contents

Index of Figures....4
CHAPTER 1. Description of the Guidance, Navigation and Control Application....5
CHAPTER 2. GNC Design based on MoSATT-CMP design flow....8
2.1 TASTE-IV modelling and TASTE2DOLC model transformation....8
2.1.1. TASTE-IV modelling (GNC-A)....9
2.1.2. TASTE-IV modelling (GNC-B)....12
2.2 DOLC2BIP model transformation....14
2.2.1. GNC-A model....14
2.2.2. GNC-B model....15
2.3 Multithread-BIP execution on LEON4....15
2.3.1. Multithread-BIP Execution for GNC-A....15
2.3.2. Multithread-BIP Execution for GNC-B....16
2.4 Instrumentation and WCET measurements (ALEXANDROS TO PUT STUFF HERE)....18
CHAPTER 3. Conclusions....19

Index of Figures

Figure 1: The Fixed Priority Process Network of the GNC application 6
Figure 2: The MSC of the GNC application 7
Figure 3: The FPPN model of the GNC application in TASTE-IV tool 9
Figure 4: The task graph of the GNC application 12
Figure 5: Real-time execution on LEON4 of the GNC application (GNC-A) 16
Figure 6: Zoomed Gantt-Chart (GNC-A) 16
Figure 7: Real-time execution on LEON4 of the pipelined GNC application (GNC-B) 17
Figure 8: Time window with jobs parallel executions (GNC-B) 17

CHAPTER 1. Description of the Guidance, Navigation and Control Application

The (Guidance, Navigation and Control) GNC module is applicable to space applications for the complete mission profile, from launch to splashdown. The main objective of the GNC real-time software subsystem is to effect the movement of the vehicles and provide the corresponding sensor and controller with the necessary data. The process involves three steps: guidance equipment and software first compute the orbital location required to satisfy mission requirements, navigation then tracks the vehicle's actual location, and flight control then transports the orbital to the required location. Therefore, depending on the specific phase, some components can be inactive, and/or the specific data to be exchanged can have a different format. For example, during the orbital phase, the guidance function will provide inertial reference attitude to the controller, whereas in the re-entry phase reference aerodynamic angles will be sent.

Based on the MoSATT-CMP methodology, the derived FPPN model of the **GNC app** is shown in Figure 1. It comprises four major tasks:

I. The **Guidance Navigation Task** which is responsible to execute the guidance and navigation algorithms. Specifically, it estimates the current translational state of the vehicle based on the measures provided by the sensors and the actuators commands, whenever applicable. Also, it computes the derived air data parameters and aerodynamic. This task keeps the vehicle on track during the flight to reach the desired location for parachute triggering. It calculates the actual location and provides the reference attitude and the calculated air data and aerodynamic parameters to the control task which in its turn assures its objective. It should be noted that if the reference attitude is pre-computed (e.g. coming from reference trajectory for the Orbital phase), it will pass through this block to keep the generality of the operation. This is a periodic process with period equal to 500ms, deadline equal to 500ms and measured WCET equal to 22ms. The Guidance Navigation Task is assigned to functional priority equal to four (4). Its criticality is D.

II. The **Control FM Task** which performs the control and flight management algorithms. This is a periodic process with period equal to 50ms, deadline equal to 50ms and measured WCET equal to 8ms. Its criticality is D. The control FM task is assigned to functional priority equal to two (2).

III. The **Control Output Task** that is in charge of sending the outputs of the GNC to the Dynamics Kinematics and Environment module (DKE). The output data consists of the geodetic altitude, the longitude, the mach and the dynamic pressure values. This is also a periodic process with period equal to 50ms, deadline equal to 50ms and measured WCET equal to 4ms. The Control Output Task is assigned to functional priority equal to three (3). Its criticality is D.

IV. The **Data Input Dispatcher Task**, a periodic process with period equal to 50ms, deadline equal to 50ms and measured WCET equal to 6ms. This task is signalled each time new data is available; it reads the data, decodes the information and dispatches the data to the right destination (i.e. Control FM and Guidance Navigation Tasks). In the GNC model used in this case study, the **MVM** (Mission and Vehicle Management), **IMU** (Inertial Measurement Unit), **GPS** (Global Positioning System) data which is fed to the data input dispatcher has been pre-computed through measurements by their relevant subcomponents and stored in static-memory buffers as C arrays for simplicity reasons. The Data Input Dispatcher Task is assigned to functional priority equal to one (1). Its criticality is D.

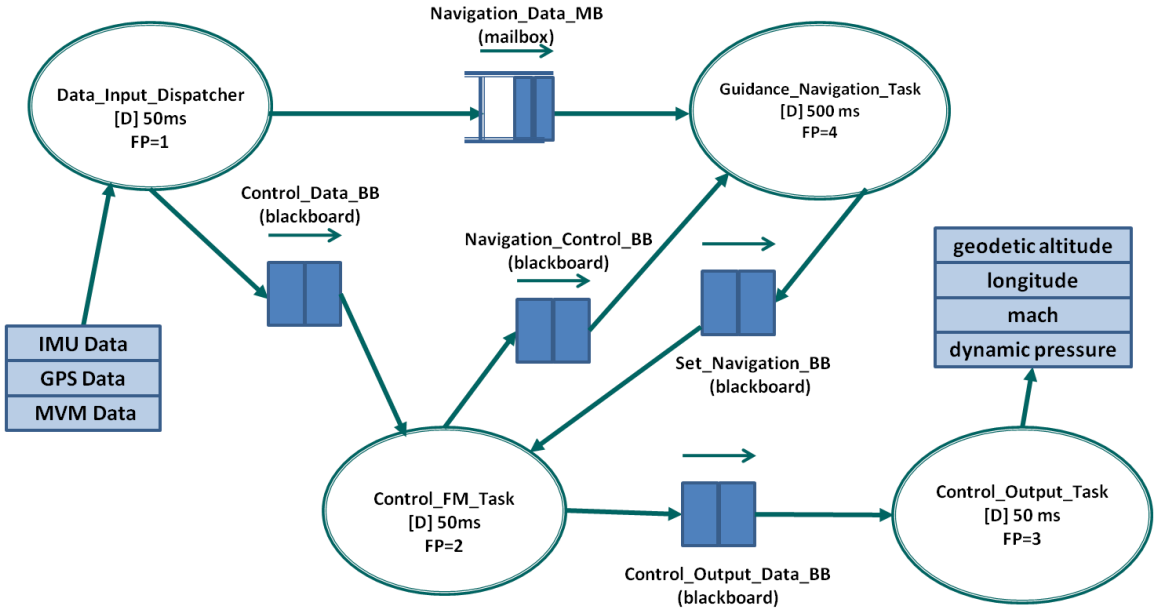

Figure 1: The Fixed Priority Process Network of the GNC application

The GNC app as provided by Elecnor Deimos-Space S.L.U includes RTEMS calls and operates on one of the four cores of the NGMP's LEON4 multi-core processor. The Message sequence chart (MSC) of the GNC app operations are delineated in Figure 2. The main objective in this activity is to port the GNC app so as to utilise more than one core from the available four cores of LEON4 processor. To this end, we followed the MoSATT-CMP design-flow and performed the next design steps toward the GNC app porting procedure:

I. GNC app modelling in TASTE-IV.
II. Removal of RTEMS calls from the C code.
III. Generation of TASTE-IV functional C code primitives.
IV. Production of the GNC app task graph and the equivalent DOLC functional model (xml2xml and c2c model transformation) through the incorporation of TASTE2DOLC tool.
V. Compilation of BIP models to Multithreaded C++ code (c2c++ model transformation) by invoking the DOLC2BIP and RTBIP tools, respectively.
VI. Compilation for Multithread-BIP for LEON4 multi-core with thread-to-core affinity and obtaining real-time performance measurements.
VII. C code instrumentation based on Rapita Verification Suite (RVS), data trace extraction and WCET estimation.

It is noted that while working on steps *i-vi*, we expedited the WCET estimation procedure by following a parallel path. Therefore, we re-organised the C code hierarchy of the GNC app produced in step *ii* listed above, so as to enable successful instrumentation based on Rapita Verification Suite. Specifically, 1) we developed a main function that executes the steps shown in the MSC chart of Figure 2 emulating the GNC app operations, 2) we allocated all source code files to a single folder and created a single make file. The file organisation of the original code was using four hierarchy levels and multiple make-files.

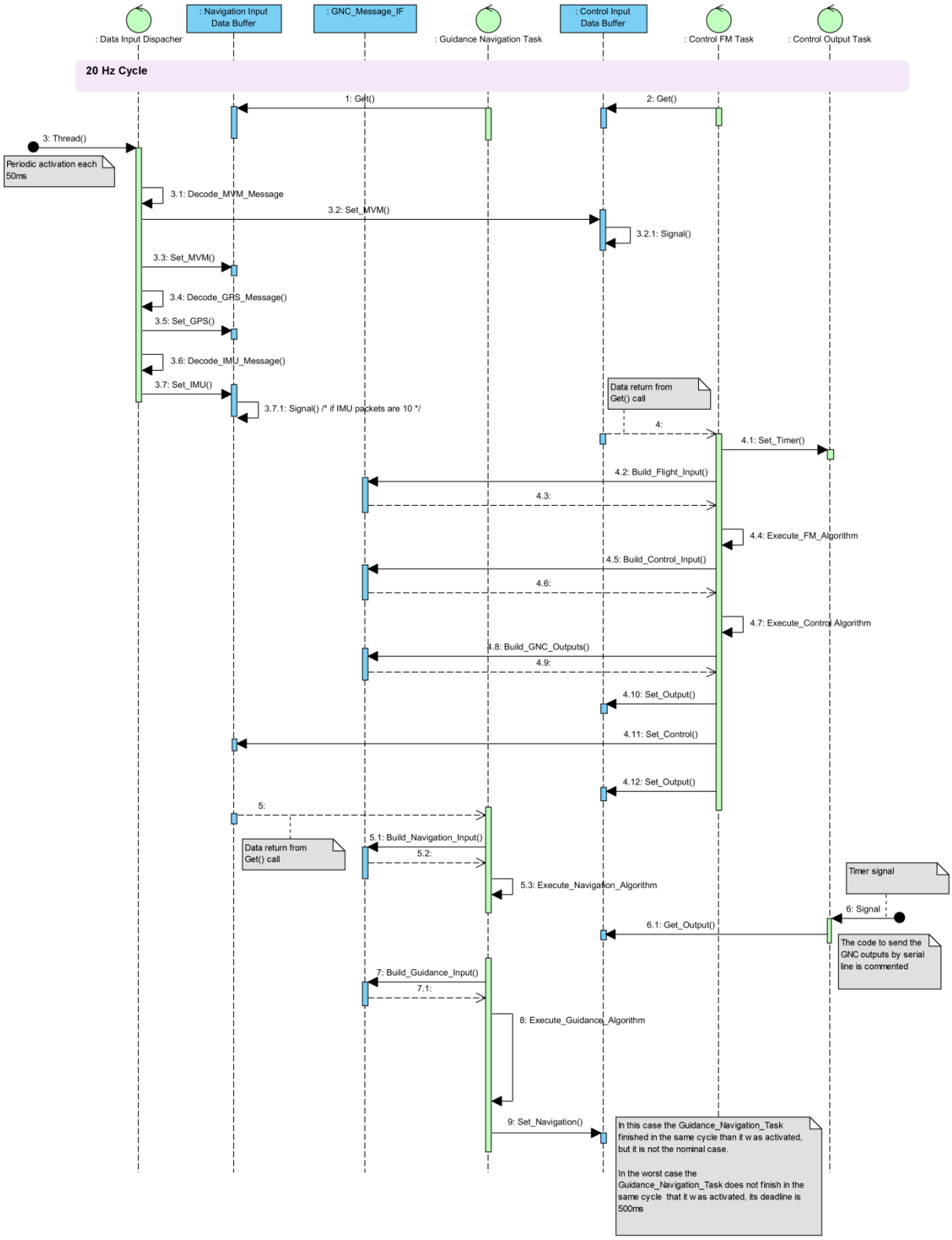

Figure 2: The MSC of the GNC application

CHAPTER 2. GNC Design based on MoSATT-CMP design flow

We proceeded with GNC development targeting to LEON4 execution using more than one compute core. Therefore, two versions of the GNC sub-module were designed.

The first one (**GNC-A**) is based on the timing constraints of the original specification, utilising 2 cores. Note that this version gave relatively high offsets (close to the end of their periods) to two tasks out of four. In the second version, which so-called pipeline version (**GNC-B**), we assume that there is headroom to increase the latency constraint (i.e. their deadline) of these two tasks to go higher than their period ($D > T$ is commonly known as "pipelining" and as one of the methods to exploit multi-core parallelism). This version uses 3 compute cores. The two versions of the application can be installed by executing the following command in the folder tree of the repository:

```
source ./tools/sourceme.rc
cd ./app/
./instal.sh
cd gnc-tas2bip
```

Within the gnc-tas2bip folder there are the following folders:

```
gnc-fppn
gnc-taste-initial
pipegnc-fppn
pipegnc-taste-initial
```

The folders **gnc-fppn** and **gnc-taste-initial** contain the DOL-C source code and TASTE-IV model of the first version of the GNC application (**GNC-A**). The folders **pipegnc-fppn** and **pipegnc-taste-initial** include the DOL-C source code and TASTE-IV model of the pipeline version of the application (**GNC-B**).

The TASTE-IV models are used as input to TASTE2DOLC tool to generate the task-graphs of the GNC app and the initial code skeletons in DOL-C. To expedite the work process of porting the original C code of the GNC sub-system and using it in our design methodology, we have built the TASTE-IV model at first and designed a "bridge" API between the C libraries of GNC on one side and the FPPN/DOL-C functional code style on the other side. After generating the task-graph and the skeleton DOL-C code, we ported the C code of the GNC application into the DOLC code and proceeded with the implementation on LEON4 following the proposed design stages.

2.1 TASTE-IV modelling and TASTE2DOLC model transformation

The first step is to design the corresponding TASTE-IV model of the GNC FPPN model. Figure 3 illustrates the equivalent TASTE-IV model of the GNC app and located at the folders **gnc-taste-initial** (**GNC-A**) and **pipegnc-taste-initial** (**GNC-B**). The top-level structure of both versions is the same; the difference is in the attributes.

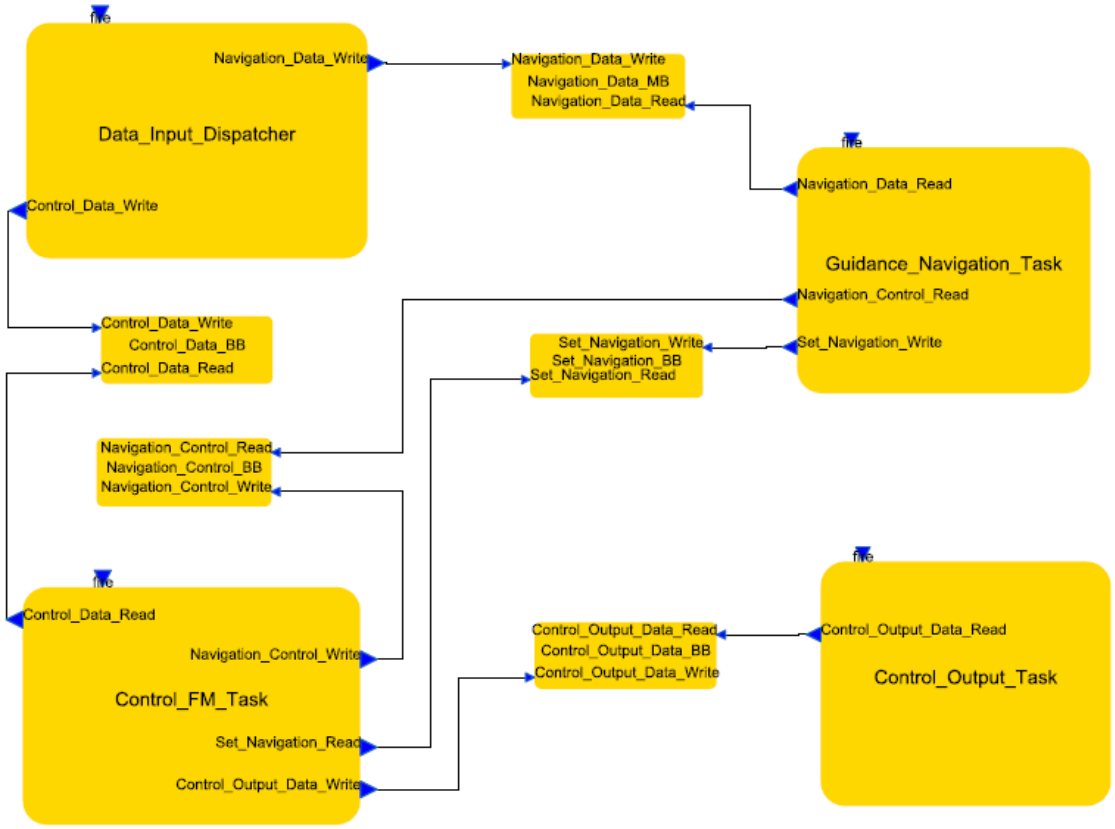

Figure 3: The FPPN model of the GNC application in TASTE-IV tool

2.1.1. TASTE-IV modelling (GNC-A)

The functional priority values for the tasks (based on their execution order in the MSC diagram) were assigned as follows:

I. Data Input Dispatcher: Fpriority=1
II. Control FM Task: Fpriority=2
III. Guidance Navigation Task: Fpriority=4
IV. Control Output Task: Fpriority=3

The length of the Navigation_DATA_MB mailbox is equal to 10 due to the fact that according to the MSC diagram of Figure 2 the Guidance-Navigation task requires 10 IMU packets to be signalled. After the generation of the corresponding C code skeletons and the description of the C function code, the TASTE2DOLC tool was invoked. Since the Control-Output-Task and the Guidance-Navigation-Task exhibit 30ms and 450ms time offsets, respectively, we set the offsets.dat file as follows:

File: offsets.dat
---
control_output_task=30
guidance_navigation_task=450

The TASTE2DOLC tool transformed the TASTE-IV functional model to the equivalent DOLC functional model under the generated fppn folder. The outputs of the TASTE2DOLC generator that are placed into the **fppn** folder under the **gnc-taste-initial** folder comprise: a) generator.out (logging information), b) fppn-dolc.xml (DOLC xml model), c) src folder with the DOLC C code generation, d) fppn_tg.jobsprocs (jobs per task) and e) fppn_tg.jobs (task-graph description). Subsequently, we

proceeded with the RTEMS calls removal from the GNC app C source code and defining FPPN-API bridge between FPPN/DOL-C API of the generated C skeletons and the plain C source code.

d) fppn_tg.jobsprocs

control_fm_task	10
guidance_navigation_task	1
control_output_task	10
data_input_dispatcher	10

Thus, **control_fm_task** generates jobs J1 - J10.

guidance_navigation_task generates job J11.

control_output_task generates jobs J12-J21.

data_input_dispatcher generates jobs J22-J31.

e) fppn_tg.jobs

J1	0	50	2	8	8
J2	50	100	2	8	8
J3	100	150	2	8	8
J4	150	200	2	8	8
J5	200	250	2	8	8
J6	250	300	2	8	8
J7	300	350	2	8	8
J8	350	400	2	8	8
J9	400	450	2	8	8
J10	450	500	2	8	8
J11	450	500	2	22	22
J12	30	80	2	4	4
J13	80	130	2	4	4
J14	130	180	2	4	4
J15	180	230	2	4	4
J16	230	280	2	4	4
J17	280	330	2	4	4
J18	330	380	2	4	4
J19	380	430	2	4	4
J20	430	480	2	4	4
J21	480	500	2	4	4
J22	0	50	2	6	6
J23	50	100	2	6	6
J24	100	150	2	6	6
J25	150	200	2	6	6
J26	200	250	2	6	6
J27	250	300	2	6	6
J28	300	350	2	6	6
J29	350	400	2	6	6
J30	400	450	2	6	6
J31	450	500	2	6	6
	(J1, J2)				
	(J2, J3)				
	(J3, J4)				
	(J4, J5)				
	(J5, J6)				
	(J6, J7)				
	(J7, J8)				
	(J8, J9)				
	(J9, J10)				
	(J10, J11)				
	(J22, J1)				
	(J1, J23)				
	(J23, J2)				
	(J2, J24)				
	(J24, J3)				
	(J3, J25)				
	(J25, J4)				
	(J4, J26)				

	(J26, J5) (J5, J27) (J27, J6) (J6, J28) (J28, J7) (J7, J29) (J29, J8) (J8, J30) (J30, J9) (J9, J31) (J31, J10) (J22, J23) (J23, J24) (J24, J25) (J25, J26) (J26, J27) (J27, J28) (J28, J29) (J29, J30) (J30, J31) (J31, J11) (J1, J12) (J12, J2) (J2, J13) (J13, J3) (J3, J14) (J14, J4) (J4, J15) (J15, J5) (J5, J16) (J16, J6) (J6, J17) (J17, J7) (J7, J18) (J18, J8) (J8, J19) (J19, J9) (J9, J20) (J20, J10) (J10, J21) (J12, J13) (J13, J14) (J14, J15) (J15, J16) (J16, J17) (J17, J18) (J18, J19) (J19, J20) (J20, J21)

The hyperperiod by the TASTE2DOLC generator is H=500ms. The derived task-graph is depicted on Figure 4, where $J_i = p_i\,[k_i]$ is the job name, k_i is some identifier number (different from i), A_i is the arrival time of the job J_i, D_i is the absolute deadline and C_i is WCET. The tasks Guidance Navigation and Control Output exhibit time-offsets of 450 and 30ms, respectively. The next step included the incorporation of the task-graph data to scheduling tool and the porting of the C code of the GNC app to the basic DOLC code. The **gnc-fppn** folder includes the DOLC source code of the GNC app and the relevant scripts for performing the compilations DOLC to BIP and Multithread-BIP. WCET values have been measured based on LEON4 profiling after performing C++ compilation with -O3 compiler option and linking with the Multithread-BIP library.

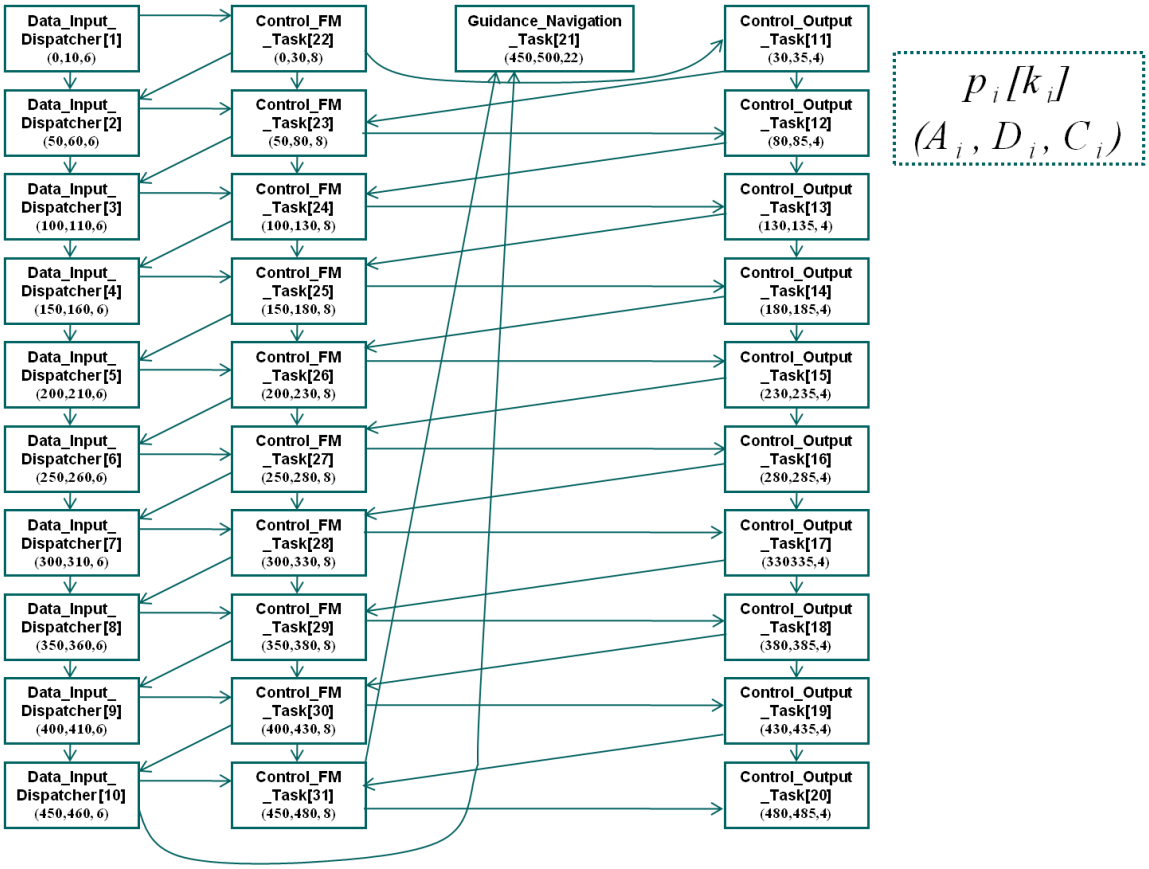

Figure 4: The task graph of the GNC application

2.1.2. TASTE-IV modelling (GNC-B)

The pipelined version of the GNC application, as mentioned above, targets to increase the number of cores to be utilised in parallel. To this end, we modified the execution order of the tasks without violating the data dependency among them. Specifically, we removed the offsets from the Control Output and Guidance Navigation Tasks, but instead we skipped their first execution in the modified version of the corresponding periodic "fire()" function calls. As a result their effective implicit offsets become equal to their respective period (effectively, the offset jumped from 30ms to 50ms for Control Output Task and from 450ms to 500ms for Guidance Navigation Task).

Since now the deferred tasks process the data from the previous period, to preserve the functional behaviour in the new version these tasks should execute earlier, so they get higher (numerically smaller) functional priorities. Further, to ensure extra parallelism (i.e. the goal of this pipelining experiment) we use *buffering* technique: we added one extra place into the mailbox between Data Dispatcher and Guidance Navigation, thus its length increased from 10 to 11 data items.

We assigned the new priority values for the tasks, as follows:

I. Data Input Dispatcher: Fpriority=1

II. Control FM Task: Fpriority=4

III. Guidance Navigation Task: Fpriority=3

IV. Control Output Task: Fpriority=2

Note that if we did not use buffering between Data Dispatcher and Guidance Navigation we would have to inverse the priority order between these two tasks as well, but due to buffering their relative priority order does not matter and they should not have precedence constraints between each other (and hence executable in parallel on two different cores).

However, TASTE2DOLC tool is not aware of the extra buffering place in the mailbox between Data Dispatcher and Guidance Navigation. By default it implies precedence constraints on the mailbox. Therefore, we have modified the generated **fppn-dolc.xml** file manually by removing the '**<precedence_chain>**' element that dictates the priority between Data Input Dispatcher and Guidance Navigation Task. We also modified the generated task-graph description (we deleted the job-precedence arcs (J22, J11) and (J11, J23)).

In this approach we assumed that 20 ms extra latency on producing the output at **control_output** is tolerable for the system. The **pipegnc_fppn** folder contains the DOLC source files and the modified task graph descriptions together with the relevant script files for model transformations.

fppn_tg_MODIFIED.jobs

```
---------------------
0       50      2       8       8
50      100     2       8       8
100     150     2       8       8
150     200     2       8       8
200     250     2       8       8
250     300     2       8       8
300     350     2       8       8
350     400     2       8       8
400     450     2       8       8
450     500     2       8       8
0       500     2       22      22
0       50      2       4       4
50      100     2       4       4
100     150     2       4       4
150     200     2       4       4
200     250     2       4       4
250     300     2       4       4
300     350     2       4       4
350     400     2       4       4
400     450     2       4       4
450     500     2       4       4
0       50      2       6       6
50      100     2       6       6
100     150     2       6       6
150     200     2       6       6
200     250     2       6       6
250     300     2       6       6
300     350     2       6       6
350     400     2       6       6
400     450     2       6       6
450     500     2       6       6
(J11, J1)
(J1, J2)
(J2, J3)
(J3, J4)
(J4, J5)
(J5, J6)
(J6, J7)
(J7, J8)
(J8, J9)
(J9, J10)
(J22, J1)
(J1, J23)
(J23, J2)
(J2, J24)
(J24, J3)
(J3, J25)
(J25, J4)
(J4, J26)
```

(J26, J5)
(J5, J27)
(J27, J6)
(J6, J28)
(J28, J7)
(J7, J29)
(J29, J8)
(J8, J30)
(J30, J9)
(J9, J31)
(J31, J10)
(J23, J24)
(J24, J25)
(J25, J26)
(J26, J27)
(J27, J28)
(J28, J29)
(J29, J30)
(J30, J31)
(J12, J1)
(J1, J13)
(J13, J2)
(J2, J14)
(J14, J3)
(J3, J15)
(J15, J4)
(J4, J16)
(J16, J5)
(J5, J17)
(J17, J6)
(J6, J18)
(J18, J7)
(J7, J19)
(J19, J8)
(J8, J20)
(J20, J9)
(J9, J21)
(J21, J10)
(J12, J13)
(J13, J14)
(J14, J15)
(J15, J16)
(J16, J17)
(J17, J18)
(J18, J19)
(J19, J20)
(J20, J21)
(J22, J23)

2.2 DOLC2BIP model transformation

2.2.1. GNC-A model

The DOLC C code is compiled together with GNC C code (without the RTEMS calls C code) under the invocation of the DOLC2BIP tool and a simulation is executed on BIP framework. Therefore, the DOLC GNC model is transformed to the BIP C++ model. The simulation outputs are consistent with the output produced by the original GNC app single-core RTEMS variant provided by Deimos which we ran on LEON4 as well to obtain reference data output traces. The execution of the script “run” located at the **gnc-fppn** folder performs the dolc2bip transformation and then BIP C++ compilation. It should be noted here that the 30ms and 450ms offsets for the Control Output Task and Guidance Navigation Task, respectively, are inserted by manually modified script during the dolc-to-bip process. To be more specific, we inserted the following lines into the “run” script at dolc to bip mechanism:

```
sed -i 's/cPeriodicSourcecontrol_output_task(0, 50)/cPeriodicSourcecontrol_output_task(30, 50)/g' ${APP_APSW}.bip

sed -i 's/cPeriodicSourceguidance_navigation_task(0, 500)/cPeriodicSourceguidance_navigation_task(450, 500)/g'  ${APP_APSW}.bip
```

This transformation replaces zero offet by the required offset in the `PeriodicSource' BIP components for the respective tasks. These components are part of the `Task Controller' functionality (see TECHNICAL NOTE ON ENSURING SCHEDULABILITY).

We also use `sed' to modify the initial arrival time assumed for these tasks in the `PrecedenceMutex' components. These components ensure correct precedence order according to FPPN MoC semantics and make part of the `MoC Controller' functionality (see TECHNICAL NOTE ON ENSURING SCHEDULABILITY).

This modification of the default design script demonstrates the possibility for the designer to have access to the model transformation procedure of BIP and control the associated model generation. Furthermore, the designer can also address late changes of the model behaviour without requiring to return back to the first design step i.e. TASTE-IV model and without hacking the runtime environment at the low level.

2.2.2. GNC-B model

Similarly, the execution of the script "run" located at the **pipegnc-fppn** folder performs the dolc2bip transformation and the BIP C++ compilation. The file named **fppn-dolc_SWModel.bip.x** is the BIP executable model. Also in this case the BIP simulation exhibited the same functional behaviour as the original single-core GNC application.

2.3 Multithread-BIP execution on LEON4

2.3.1. Multithread-BIP Execution for GNC-A

The last step in the MoSATT-CMP design flow is to launch Multithread-BIP tool and compile the generated BIP model under the BIP RTE framework and target it for execution on LEON4 processor. The execution of the script "build-leon4", which builds the Multithread-BIP executable based on the compilation with the sparc-rtems-gcc (C part of the GNC app) and sparc-rtems-g++ (BIP code parts of the GNC app) compilers. The executable was loaded under grmon and executed on LEON4.

Figure 5 delineates the Gantt Chart of the execution traces within a time frame equal to the hyperperiod of the 500 ms. The PDF Gantt charts are obtained as described in Tast2BIP User Manual.

Core-0 is used as the `control core' and runs BIP RTE engine and all the controller components (as specified in textual file "**threadmap.txt**"). Core-1 and core-2 are `compute cores' and run the processes (as specified in the same file).

The GNC-A application built under the MoSATT-CMP methodology utilises 2 compute cores of the LEON4 processor of the NGMP. Process P1 corresponds to the Data Input Dispatcher, P2 to the Control FM Task, P3 to the Control Output Task and P4 to the Guidance Navigation Task. P20 is the BIP Real Time Engine that runs on core-0. Processes P1, P2 and P3 utilise core -1 while P4 executes its operations on core-2.

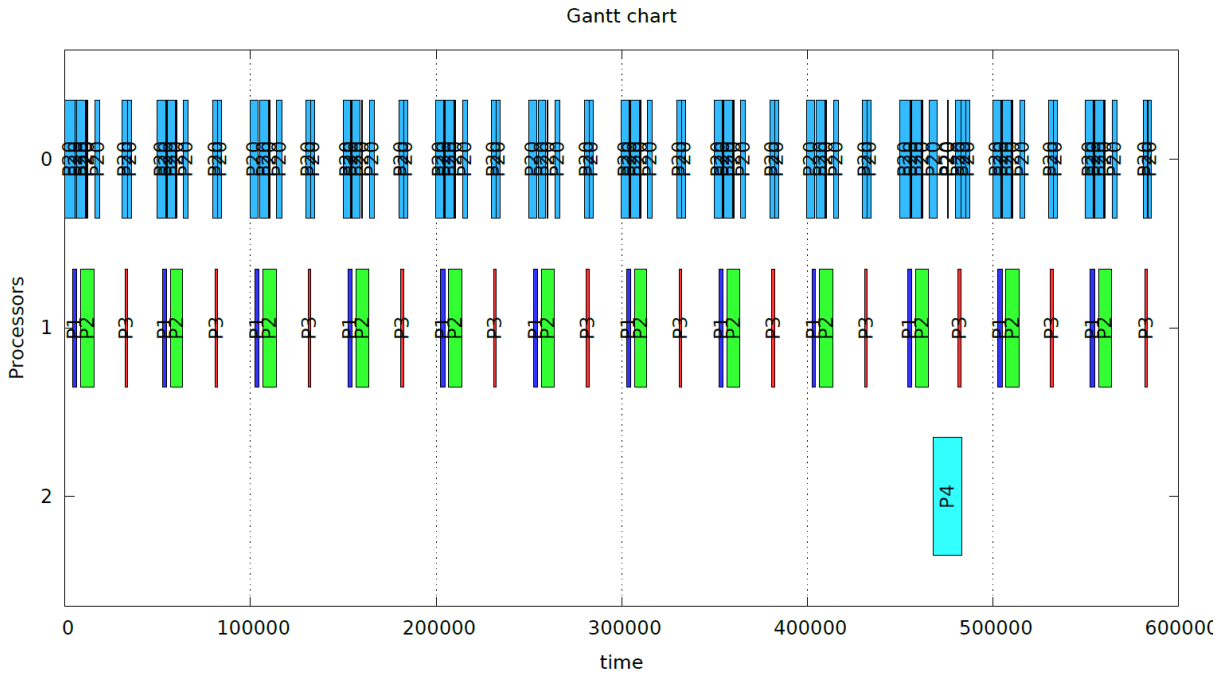

Figure 5: Real-time execution on LEON4 of the GNC application (GNC-A)

Figure 6 zooms to the time frame where the parallel execution of P4 with P3 is shown with more details. Right after the execution of 10 consecutive jobs of P1 and P2 the first job on P4 is executed. The 10th job of P3 is operating in parallel with the 1st job of P4. Moreover, minor time shifts to the jobs execution time are noticed and this is due to the P20 overhead.

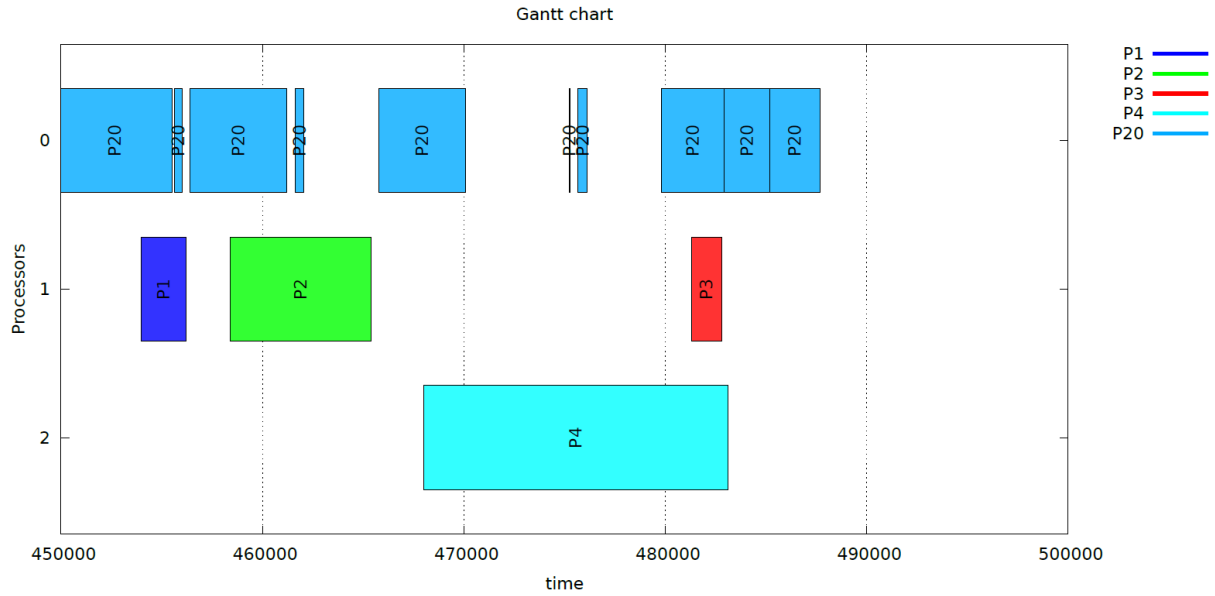

Figure 6: Zoomed Gantt-Chart (GNC-A)

2.3.2. Multithread-BIP Execution for GNC-B

Similarly to the previous procedures we executed the "build-leon4" located at the **pipegnc-fppn** folder and executed the Multithread-BIP pipelined version of the GNC app on LEON4. Figure 7 shows the Gantt Chart of the execution traces within a time frame equal to the hyperperiod of the 500ms while Figure 8 focuses on the time-window that jobs are executed in parallel on 4 cores.

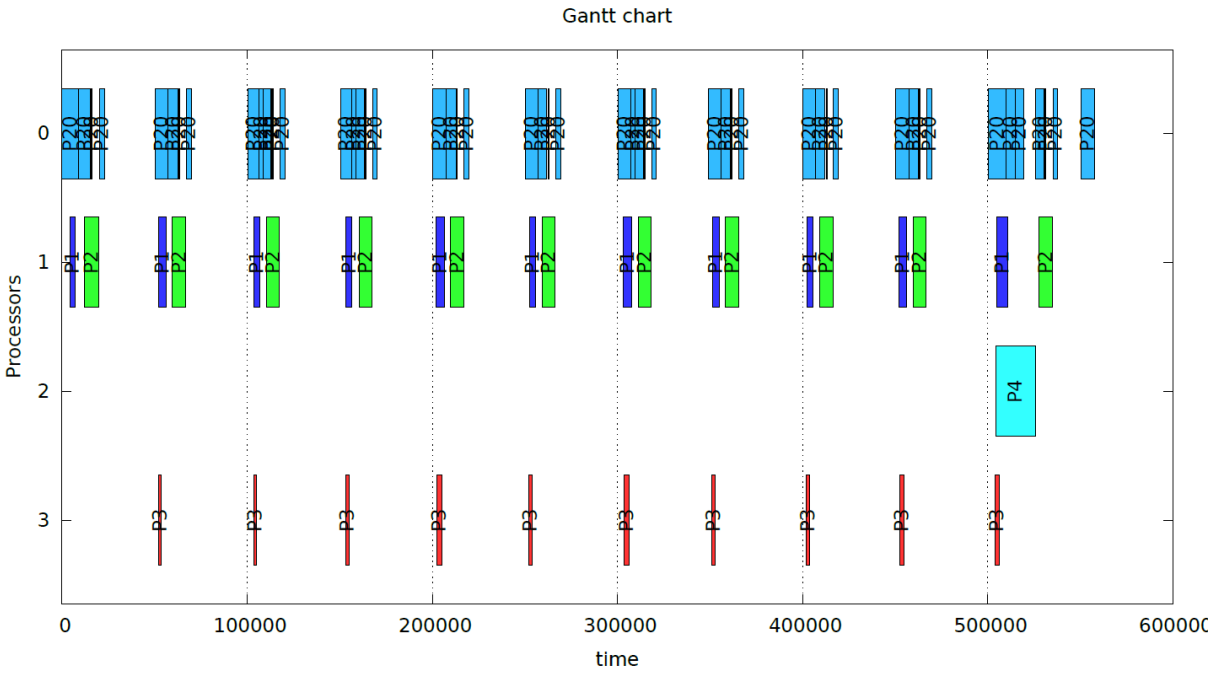

Figure 7: Real-time execution on LEON4 of the pipelined GNC application (GNC-B)

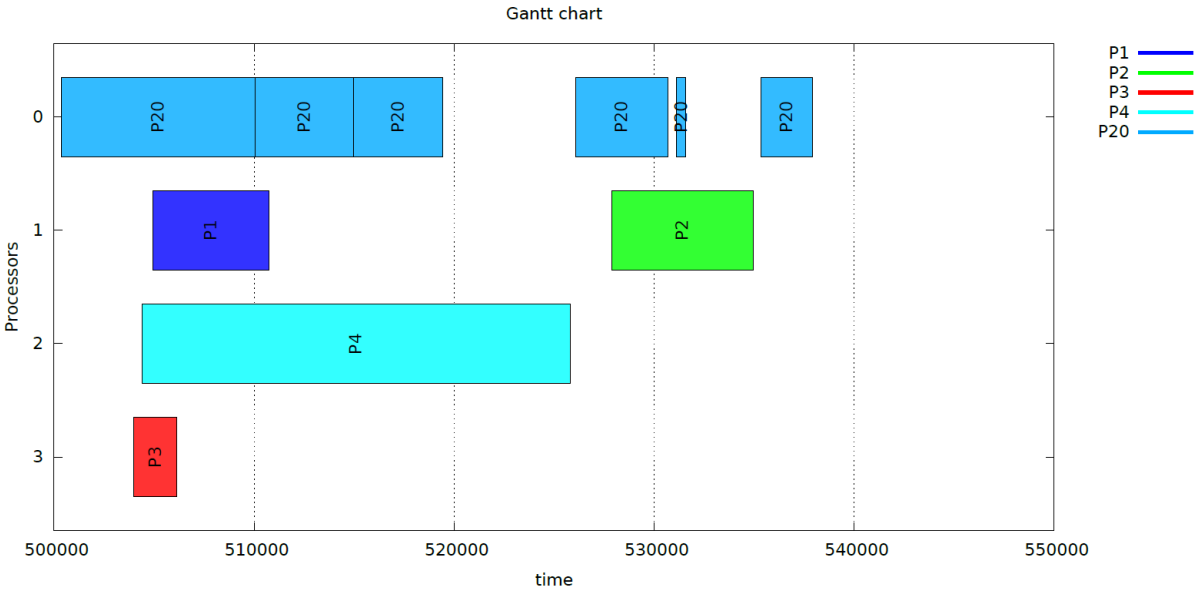

Figure 8: Time window with jobs parallel executions (GNC-B)

As mentioned previously P4 and P3 processes skip their first job execution. This is noticeable in Figure 7 where the first job of P4 (Guidance Navigation Task) is executed after the time-frame of the 500ms. In the same way, the first job of P3 is executed after 50ms. We also see that the 11th job of P1 is executed in parallel with the 1st job of P4, which means that P4 and P1 access to the buffered mailbox concurrently. But this should happen correctly, as P4 reads the first 10 valid IMU frames stored there while P1 writes the 11th frame. In the pipelined version, we, in fact, employ all 4 cores of LEON4. P3 is executed on core-3 and in parallel to P1 and P4 which are executed on core-1 and core 2, respectively.

Although, the pipelined GNC application utilises more cores than the first version, we notice that, unfortunately, the aspired parallelism gain, which should increase the potential throughput speedup of the application or minimize its required timing-partition size, was not achieved. This is visible if we compare Figure 8 and Figure 6. The timing window where the task run after the beginning of the period in the two cases is roughly the same: roughly 38 ms, leaving the slack of only 12 ms.

This is hypothetically due to the strong interference between P1 to P4. The job in P4 (Figure 8) requires more time to be completed than the corresponding job of P4 shown in Figure 6. However, there is room for improvement by improving the implementation of the mailbox and letting P4 ten data items in one call to the `read' interface instead of issuing 10 calls as it is done now.

2.4 Instrumentation and WCET measurements (ALEXANDROS TO PUT STUFF HERE)

We proceeded with the instrumentation after the GNC C code was "cleaned" from the RTEMS calls. As mentioned previously, we redesigned the GNC C code hierarchy so as to include all files under a single directory and form one makefile substituting the original multiple Makefiles, towards the WCET measurements of the GNC app based on the instrumentation. Furthermore, we defined a main function that models the GNC app operation. Table 1 shows the main file. The new C code hierarchy was built both under gcc and bcc compiler and downloaded on one of the 4 cores of LEON4. The file WCET_main_flat_bcc.tar.gz located at \trunk\tools\work_forge\GNC_app\ includes the single directory file hierarchy of the GNC app compiled under bcc compiler.

File: wcetgnc.c

```
----------------------------------------
#include <stdio.h>
#include "data_input_dispatcher_API.h"
#include "control_fm_task_FPPN_API.h"
#include "control_output_task_FPPN_API.h"
#include "data_input_dispatcher_API.h"
#include "guidance_navigation_task_FPPN_API.h"
#include "Compilation_Flags.h"

int main(void) {

  // Call init functions for buffers
 GNC_Message_IF_FPPN__Init();
 Navigation_Input_Data_Buffer_FPPN__Init();
 Control_Input_Data_Buffer_FPPN__Init();

 Data_Input_Dispatcher_Task_FPPN__Init();
 void* status_DIP = Data_Input_Dispatcher_FPPN__ThreadInit();

 Control_FM_Task_FPPN__Init();
 void* status_Control_FM = Control_FM_Task_FPPN__ThreadInit();

 Control_Output_Task_FPPN__Init();
 void* status_Control_Output = Control_Output_Task_FPPN__ThreadInit();

 Guidance_Navigation_Task_FPPN__Init();
 void* status_GN = Guidance_Navigation_Task_FPPN__ThreadInit();


 int i,j;
 j = 0;
 while (1) { // repeat hyperperiod forever = models 500 ms cycle
   for (i=0; i<9; i++)
              {
              Data_Input_Dispatcher_FPPN__Fire(status_DIP);
              Control_FM_Task_FPPN__Fire(status_Control_FM);
              Control_Output_Task_FPPN__Fire(status_Control_Output);
              }//end for
                        Data_Input_Dispatcher_FPPN__Fire(status_DIP);
              Control_FM_Task_FPPN__Fire(status_Control_FM);
```

```
                        Guidance_Navigation_Task_FPPN__Fire(status_GN);
            Control_Output_Task_FPPN__Fire(status_Control_Output);
#ifdef LOGGING_DEBUGDATA
                printf("\n##################################\n");
                        printf("[TRACE while loop] =  %d\n",j);
            printf("##################################\n");
#endif

                        j++;
                        if(j==400)

                        {

                                return 0;
                        }

  }//end while

  return 0;

}
```

Table 1: GNC main function

Instead of instrumenting the RTE BIP code of the GNC app as generated by the MoSATT-CMP design-flow process, we have performed this step in parallel with the porting of the GNC app to the TATE-IV, DOLC and BIP design phases due to the lack of time.

CHAPTER 3. Conclusions

This document describes the development and porting of the GNC application to the LEON4 multi-core processor of the NGMP platform. We have successfully ported the code on the LEON4 multi-core utilising more than 1 core in parallel. Two versions of the GNC application were designed to show potential possibilities to imrove the parallelism (and hence either the throughput or the timing partition size) at the cost of paying extra latency. The obtained results were reported and demonstrated the capabilities of the current MoSaTT-CMP methodology and its associated design tools.