A Multicore Approach to Model-Based Analysis and Design of Cyber-Physical Systems

Anil Kanduri[1], Amir-Mohammad Rahmani[1], Pasi Liljeberg[1], Kaiyu Wan[2], Ka Lok Man[2,3], Juha Plosila[1]

[1]Department of Information Technology, University of Turku, Finland
[2]Xi'an Jiaotong-Liverpool University, China
[3]Baltic Institute of Advanced Technology, Lithuania
Email: {spakan, amirah, pakrli, juplos}@utu.fi, {kaiyu.wan,ka.man}@xjtlu.edu.cn

Abstract— **Embedded systems took a leap as combining computational elements with physical systems led to many novel applications, further saw the rise of a new domain - Cyber-Physical Systems (CPS). Growing importance for CPS in industry threw down many challenges in a designer's perspective ranging from computational methods, modeling platforms, programming structures, relevant hardware systems, etc. Ptolemy is the platform which is tailor made for such full scale design of networked and real time systems. In an effort to explore the suitability of Ptolemy II platform for CPS design, we chose an Unmanned Aerial vehicle (UAV) application as a case study. In this paper, we model UAV in Ptolemy II in a modular and hierarchical way such that the system meets the requirements of data flows and dependencies. Key parameters of a typical CPS such as schedulability and predictability were analyzed. In the end, to better the performance of UAV, computational tasks were mapped onto a networks-on-chip based multicore system. Our experimental results show the efficiency of our high level analysis and modeling and the extracted system requirements to enhance the system predictability.**

Keywords- **Cyber-Physical Systems, Unmanned Air Vehicles, Multicore Systems, Networks-on-Chip**

I. INTRODUCTION

Cyber-Physical Systems is a relatively new trend of real time system design that includes physical environment models. Unlike traditional Embedded Systems, CPS design involves modeling of physical and computational processes, giving equal importance to both. Tight coupling of physical system with computational system has been a long standing design challenge for CPS. CPS are characterized to be highly inter-disciplinary[1] in nature involving several fields like mechanics, control systems, computer architecture, electronics and classical physics which are interacting with each other more often than not. This nature brings the demand for expertise from different areas of science and technology. CPS are expected to work over longer periods of time in an infinite loop, in a real time environment. Hence, the system has many key parameters [2] like schedulability, reaction time, performance, speed, quality of service, reliability, predictability, ease of modeling and simulation etc. Keeping these factors in view, designing CPS becomes even hard.

The time to market curve of CPS design is linear, in a sense that each progressive design task is depended on successful completion of previous task. With different processing blocks requiring different times spent on design, it is recommended to work in a concurrent manner. Predicting a particular block's or even the system's parameters would come good in such situations [3]. These predicted values can be taken as a starting point of design in some other concurrent block, which eases design procedure [4]. Since the system complexity, time spent on design and manufacturing costs are relatively higher for CPS, making the system as predictable as possible is necessary to resolve those issues. Also, predicting the output for all possible test cases helps in maintaining safety criticalness of the system. In our work, we tried to address predictability of a system using modular and hierarchical approach to design process.

Apart from above challenges, simulation and implementation of CPS too is apparently complex, mostly attributed to lack of mature tools and platforms in big numbers. Although data flow languages [5] and actor based tools [6] were tried out, they are not yet ready to cater for high scale design. The procedure of designing system architecture, writing appropriate software for it, finding suitable hardware platform until tape-out, and finally verifying it has now become a daunting task with lack of suitable model driven platforms. Our goal is to explore design aspects involved in a typical CPS starting from computational models, platforms until implementation. For this purpose, we choose an Un-manned Air Vehicle Application to be simulated in Ptolemy II platform. In the next sections, UAV application is detailed, followed by simulation in Ptolemy II. Later, an approach to implement this application on multicore systems under two different conditions is proposed.

II. UAV APPLICATION

There many versions of UAV applications being implemented through various sources. . Typical UAV has the traits of any normal flight control system, except for the operation which can be automated from a base station at ground. Minimum requirements for a UAV system to operate are navigation, actuation of motors and object detection. This is achieved through GPS device, servo mechanism and temperature sensors. In our work, we chose the version implemented in [7], the Paparazzi project [8]. Paparazzi is an open source [9], written in C and is tested for air vehicles that weigh up to 25 kilograms. This model works in two modes of operation viz., Fly-by-wire (FBW) and Autopilot (AP), comprising of diverse tasks like sensing, parameter measurement, navigation, linear and angular motion etc. In order to preserve modular and hierarchical nature of application, functionality of system is classified into different tasks, such that these tasks are communicating with each other for data and control flows. Actuation of servos is done in

FBW mode for design ease, although both the modes are capable of generating control signals for servo. Control can be switched to AP mode in case of failure in FBW. Architectural view UAV application as per Paparazzi model is shown in Figure 1.

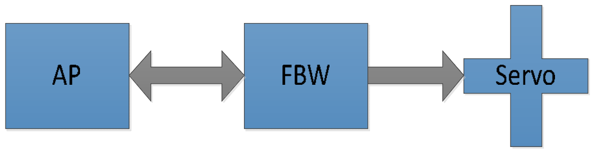


Figure 1. Top level View of UAV Application

Important tasks involved in both FBW and AP modes are elaborated below.

A. Fly- By- Wire Mode

In FBW mode of operation, motion of air vehicle is controlled by radio commands sent from a ground station. Received control signals are used to determine orientation of air vehicle in 3-D space.

Receiver: It receives radio commands from ground station and sends the input stream of data which governs actuation of servos. Servo mechanism involves alternating between power levels (or switching between voltage levels in case of power optimization), hence control data is sent as pulse position modulated signal. This ensures that position control of motor is through presence or absence of pulse, instead of changing power supplies.

Controller: It gets control signals from Receiver and controls the servo directly. This task becomes key when it comes to switching from FBW to AP mode. It also receives control signals from AP mode too, but chooses which mode to operate with.

Checker: Checker validates the data stream sent by Receiver and determines if Controller should operate using FBW signals. In case Checker finds faulty data at the Receiver end, it sends a control variable to Controller so that it can switch to AP mode.

Transfer: This task is used to transfer control from FBW to AP by sending current control variables operating in FBW mode. In case the control is switched to AP mode, it can then start working from the point of failure of FBW.

B. Auto Pilot

Since a CPS like UAV is a real time system with hard deadlines, it is always recommended to have an additional layer of computation ensuring safety. Autopilot mode provides control signals for flight motion and these signals are used by Controller in case of failure in FBW mode. Failures are associated not only with faulty data but also with timed and sequenced arrival of data. Data and control flows among the tasks in both AP and FBW mode are shown in Figure 2. AP mode uses sensors of different domains to track different physical quantities which in turn are used to determine control signals. Tasks that come under AP mode are listed below.

Navigator: Navigator receives co-ordinates of current location from a GPS device. It compares this information against pre-determined flight plan and thus comprehends destination co-ordinates. For a simplified design, speed vector of vehicle too is held by navigator task. Navigator sends these data streams to other tasks in AP mode for necessary computations.

Altitude: Altitude task measures the current height of vehicle from ground level using altimeter. Similar to Navigator, Altitude task compares current height against the height specified in flight plan and determines difference between those two. This error is the 'height to be climbed' by the vehicle which is sent to Climb task.

Climb: Climb task alters the motion of flight from normal cruise to ascend (or descend in case of landing) until it reaches the height specified by Altitude task. This is achieved by generating enough amounts lift and drag forces which help in rising the fuselage to reach the target height. Climb task uses air density determined by pressure and temperature values at current height from their corresponding sensors, and velocity of vehicle to compute lift and drag coefficients. These coefficients represents orientation of vehicle in terms of pitch and yaw, subsequently sent to Stabilization task.

Stabilization: Stabilization tasks handles 'roll' motion of vehicle using error signal from temperature sensors at both the wings. It also gets pitch and yaw data from Climb task and speed vector from Navigator. Stabilization task sends all these necessary control variables to Controller of FBW mode, thus finalizing AP operation.

Radio: Radio task receives last known working configuration of vehicle motion in FBW mode, so that AP mode can take control from where FBW has stopped working. Radio task receives this from Transfer of FBW mode and sends to Stabilization task.

GPS and TEM: GPS sensor is responsible for geographic co-ordinates of the vehicle and Temperature (TEM) sensor gives the temperature values at the ends both the wings.

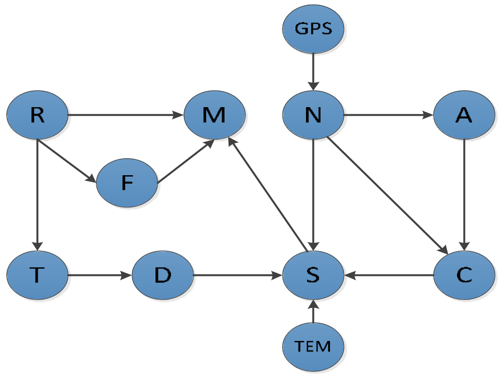


Figure 2. Task graph of UAV application (R – Receiver, M- Controller, N- Navigator, A- Altitude, F – Check Failure, T- Transfer, D: Radio, S- Stabilization, C – Climb Control, E- Temperature sensor, GPS- GPS sensor)

C. Modeling with Ptolemy

Modeling a system is to produce an abstract and behavioral description of corresponding entity. After making specifications of a system, mathematical models of computation are created which are then simulated. One of major challenges in CPS design is choice of modeling platform, as there are not many mature platforms available. Existing platforms [10] [11] [12] demand skills from different spheres like programming hardware and software, mathematical analysis, interpretation etc. Ptolemy II [13] is the platform that we chose to explore more, as it caters CPS applications better in comparison with other platforms. The objective of Ptolemy is to provide a solid modeling and simulation platform for real time, heterogeneous, concurrent systems. Also, it covers many models of computation that form building blocks of ubiquitous system design.

In order to facilitate design of concurrent processes and their scheduling, Ptolemy employs an actor based environment where each process is modelled through an actor and all these actors are communicating in a main model. Triggering an actor is by firing corresponding tokens, hence synchronizing multiple processes becomes lot easier. All the tasks detailed in previous sections are modeled in a modular and hierarchical manner using composite actor structure of Ptolemy II. Individual tasks are described under separate composite actors with required ports, the main model is shown in Figure 3.

Performance of each individual task or a group of tasks (like a block) can be analyzed separately. Adding more details to the processes of a task and/or replacing a task with some other task is easier and sequential flow can be restored too. UAV application is governed by three major parameters of motion viz., pitch, yaw and roll orientations. In FBW mode, these parameters are computed by radio commands from a ground station, whereas in AP mode, they are done by sensory data. Fixed input stream of data is used as radio commands for FBW mode. System's output patterns of motion for given input stream are summarized in Figure 4. For every iteration, angular orientation of vehicle changes in 3-D plane as per the input stream shown in Figure 4. Whenever vehicle changes the angular orientation, it recovers back in next iteration to an inertial position, such that newly attained angular position is the current zero position. In this way, every time there is a change in angular orientation, it is with respect to current angular position, but not with geographic angles. The vehicle follows linear motion for 6 iterations and then yaws at an angle of 56°. Once the position has changed, it should continue in linear motion again, until there is a need to change. So, it assumes that 56° is the new angle zero position in next iteration. Change in angle over progressive iterations completely depend on input data stream. The same principle can be extended for the other two motion parameters as well. Similarly, there is less roll movement for half of the iterations, yet the vehicle rolls at 18° for a while and then comes back to normal. Pitching angle starts to increase gradually and after reaching desired height, the vehicle goes to cruise by slowly decreasing the pitch angle and becoming flat.

Other aspects of the application are synchronous data flow and schedulability. Time taken to compute control signals vary between the two modes of operation, hence the Motor Control task waits for both incoming data streams to arrive. With respect to data rates, critical path of N→ A→C→S→M would take the maximum time among all the other paths, so is made as optimal as possible. Inherently, the model works on a message passing like environment where tokens are transferred through firing actors. So, each actor waits for required tokens in order to fire, ensuring synchronous data flow.

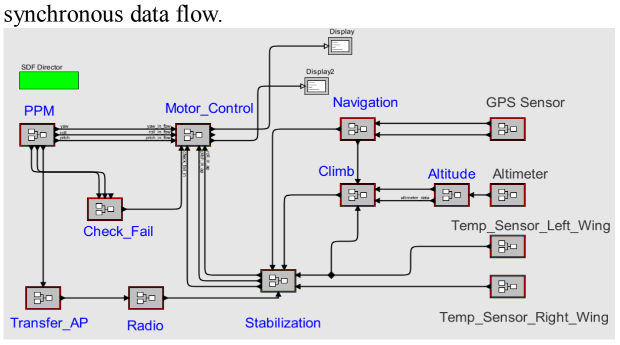
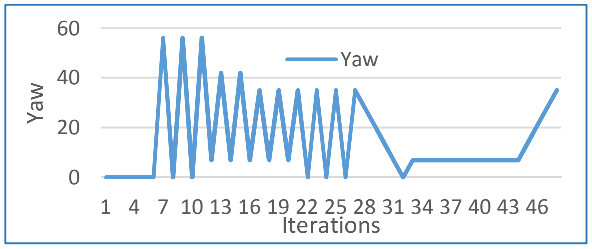
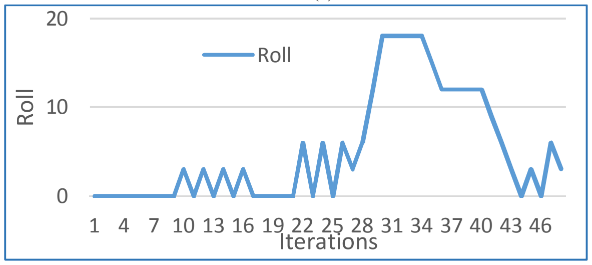


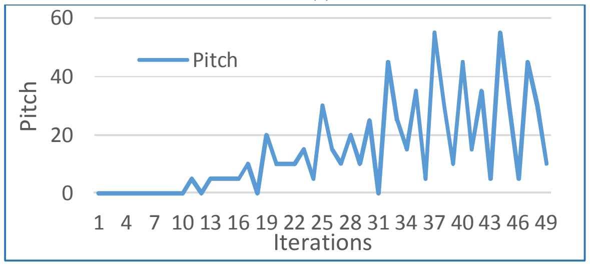
Figure 3. UAV graphical model in Ptolemy II



(a)



(b)



(c)

Figure 4. Yaw, Roll and Pitch orientations

III. APPROACH FOR MULTICORE DOMAIN

As mentioned in the previous sections, UAV application has numerous tasks involving intensive computations and complex communication among these tasks. Being a real time system with hard deadlines, UAV application cannot afford any penalties on performance. To handle such modular applications, implementation through Networks-on-Chip (NoC) [14] paradigm is the most sought out approach. The goal is to map many tasks onto multicore chip so that each task has enough resources to perform better [15]. Although there would be additional overhead caused by communication among the multiple cores, it is still beneficial to employ it.

First step in implementation of the application on a NoC platform is to map the tasks on to cores accordingly. Mapping should be as optimal as possible avoiding long or multiple hops among key tasks, focusing on critical sections and data rates. There are many algorithms and heuristics [16] [17] that can be employed during task mapping, subject to parameters like congestion control [18], thermal awareness [19], architectural topology [20] etc. Keeping the task graph from Figure 2 in view, UAV has complex data flows along with dependencies that induce further latencies. In addition to them, there are sensors and actuators which are assumed to be off the chip, hence there has to be a separate path for communication with those peripherals. Considering that these sensory tasks are equivalent to other computational tasks in the system, mapping proposed in Figure 5 could hold good. In this case, data paths are shared among A→C, E→S and a node is left empty with a 4x3 mesh.

In contrast to above mentioned mapping, sensor components can be present away from the silicon region, purely as peripherals. UAV requires data from a GPS device, altimeter, pressure and temperature sensors and in the end, it has to control a servo motor. With five different streams of data arriving and leaving, there is added congestion to the network. Another fact to consider here is the multiple port versus single media of communication between peripherals and on-chip network. Since different tasks require data from different sensors, corresponding sensors can be placed closer to those tasks, allowing a dedicated port for each sensor. In practicality, this is rather fancy approach. Widely existing scalable embedded processors have been using a common media for communication [21], where bandwidth holds the key for better performance. Mapping of tasks assuming a common media is shown in Figure 6. In this case, total number of tasks is reduced to 9, hence needing a simple 3x3 mesh structure. Shared paths include N→S, D→A, S→D, R→T and F→M. Both the mappings are based on task graph from Figure 2 as well as on data flow rates between different tasks.

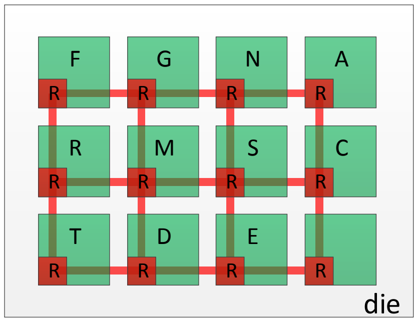


Figure 5. Mapping without peripherals

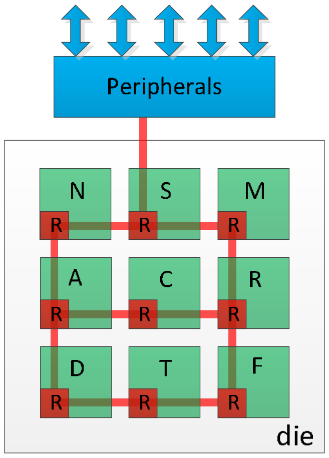


Figure 6. Mapping with peripherals

IV. CONCLUSION AND FUTURE WORK

We tried to explore modeling platforms for Cyber-physical System design in our work. In doing so, an unmanned air vehicle has been used as case study. The application is split up into different tasks such that these tasks describe both physical and computational processes of UAV. The system is modeled and simulated using Ptolemy II frame work. With intensive computations involved, logical solution for implementation of this application is to use NoC platform. Mapping of tasks onto 4x3 and 3x3 mesh structures, considering the peripherals were proposed. While framing the mathematical background for some physical processes involved in the application, we felt that adding more details to those processes would bring credibility for the design. We followed minimalistic approach during the designing, assuming right flow of data at right time. In a real test bed with the involvement of physical environment which can hazardous at times, sensors might not work as assumed in an automated design. Once we get to implement the system on a typical NoC platform, there is a serious urge to verify the design under different test conditions to ensure safety and reliability.

V. ACKNOWLEDGEMENT

This work has been partially supported by Academy of Finland and EU COST Action IC1202: Timing Analysis on Code-Level (TACLe).

VI. REFERENCES

[1] L. Ordinez, O. Alimenti, E. Rinland, M. Gómez and J. Marchetti, “Modeling and Specifying Requirements for Cyber-Physical Systems,” in *Proceedings of IEEE Latin America Transactions,* 2013, Vol. 11, No. 1, pp. 625-632.
[2] E. Lee, “Cyber Physical Systems: Design Challenges,” in *Proceedings of 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, 2008, pp. 363 – 369.
[3] J. Sifakis, “Rigorous design of cyber-physical systems,” in *Proceedings of International Conference on Embedded Computer Systems (SAMOS)*, 2012, pp. 319.
[4] Kwei-Jay Lin; Panahi. M, “A real-time service-oriented framework to support sustainable cyber-physical systems,” in *Proceedings of 8th IEEE International Conference on Industrial Informatics (INDIN)*, 2010, pp. 15 – 21.
[5] Vicaire, P.A, Hoque. E, Zhiheng Xie, Stankovic. J. A, “Bundle: A Group-Based Programming Abstraction for Cyber-Physical Systems,” in *Proceedings of IEEE Transactions on Industrial Informatics*, Vol: 8, Issue: 2, pp. 379 – 392.
[6] Chandhoke. S, Hayles. T, Kodosky. J, Guoqiang Wang, “A model-based methodology of programming cyber-physical systems,” in *Proceedings of 7th International Conference on Wireless Communications and Mobile Computing (IWCMC)*, 2011, pp. 1654 – 1659.
[7] F. Nemer, H. Cassé, P. Sainrat, J. P. Bahsoun, “PapaBench: A Free Real-Time Benchmark,” in *Proceedings of Workshop on Worst-Case Execution Time Analysis (WCET 06)*, 2006, pp. 1-6.
[8] Paparazzi project home page. Available at http://paparazzi.enac.fr/wiki/Main_Page
[9] Paparazzi open source for unix installation. Available at https://github.com/paparazzi/paparazzi
[10] Overview of Intel Cofluent Studio for time behavioral modeling. Available at http://www.intel.in/content/www/xl/es/cofluent/cofluent-studio-overview-benefits.html
[11] LabView for parallel real time systems. Available at http://zone.ni.com/devzone/cda/pub/p/id/1675
[12] Quantum Leaps graphical modeling tool. Available at http://www.state-machine.com/qm/index.php
[13] Overview of Ptolemy project. Available at http://ptolemy.eecs.berkeley.edu/publications/papers/03/overview/overview03.pdf
[14] Benini. L, De Micheli. G, “Networks on chips: A new SoC paradigm,” in *Proceedings of IEEE Computer*, 2002, Vol.35, Issue 1, pp. 70-78.
[15] Jantsch. A, “NoCs: A new contract between hardware and software,” in *Proceedings of Euromicro Symposium on Digital System Design*, 2003, pp. 10-16.
[16] Tang Lei, Kumar. S, “A two-step genetic algorithm for mapping task graphs to a network on chip architecture,” in *Proceedings of Euromicro Symposium on Digital System Design*, 2003, pp. 180-187.
[17] Carvalho. E, Calazans. N, Moraes. F, “Heuristics for Dynamic Task Mapping in NoC-based Heterogeneous MPSoCs,” in *Proceedings of 18th IEEE/IFIP International Workshop on Rapid System Prototyping*, 2007, pp. 34-40.
[18] Addo-Quaye. C, “Thermal-aware mapping and placement for 3-D NoC designs,” in *Proceedings of IEEE International SOC Conference*, 2005, pp. 25-28.
[19] Carvalho. E, “Congestion-aware task mapping in heterogeneous MPSoCs,” in *Proceedings of International Symposium on System-on-Chip*, 2008, pp. 1-4.
[20] Wooyoung Jang, “A3MAP: Architecture-Aware Analytic Mapping for Networks-on-Chip,” in *Proceedings of 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2010, pp. 523-528.
[21] Tilera Corporation, TILE-Gx8072 Processor Specification brief. Available at http://www.tilera.com/sites/default/files/productbriefs/TILE-Gx8072_PB041-02.pdf

ISOCC 2013