Development time analysis for the design of a ZigBee transmitter in TTool/DIPLODOCUS: a comparison between the Y-chart and Ψ-chart approaches

Andrea ENRICI, Ludovic APVRILLE, Renaud PACALET
{andrea.enrici,ludovic.apvrille,renaud.pacalet}@telecom-paristech.fr

LTCI, CNRS, Telecom ParisTech, Université Paris-Saclay, 06904 Biot Sophia Antipolis, France

Abstract. In this technical report, we present a quantitative analysis of the development time spent when designing data-flow embedded systems at Electronic-System Level (ESL), in the Ψ-chart and in the Y-chart approaches. We compare implementations of these approaches in TTool/DIPLODOCUS, a UML/SysML toolkit for the hardware/software co-design of radio signal processing embedded systems. This analysis quantitatively evaluates the ratio between development time spent in the mapping and Design Space Exploration (DSE) phases of the Ψ-chart over the Y-chart. We numerically evaluate this analysis in the context of the design of the physical layer of a ZigBee transmitter (IEEE 802.15.4 standard).

1 Introduction

Today's embedded systems are more and more realized with architectures where the processing operations, i.e., data and control information, are executed in parallel over a network of interconnected subsystems (e.g., Multi-Processors Systems on Chip, MPSoC, electronic equipments in automotive and avionic systems). The performance, cost and time-to-market of these systems is not only driven by the design of data-processing operations (computations) but also by the design of data-transfer operations. Therefore, it is of utmost importance to account for the design of these data-transfer operations, *communications*, in the early phases of a design process.
Since the late nineties, the Y-chart [1, 2], Fig. 1, is one of the dominant Model Driven Engineering (MDE) approaches that guides the automation of design and Design Space Exploration (DSE) of embedded systems for data-dominated applications. In this approach communications are typically described both in an application model (i.e., the system's functionality, box 1.1 in Fig. 1) and in an architecture model (i.e., the system's resources, box 1.2 in Fig. 1). In the application model, communications are represented in the form of logical dependencies between computations (e.g., channels, events). In the architecture

(platform) model, communications are described in the form of the services provides by hardware and software resources (e.g. DMA engine, bus, CPU and its Operating System) A design is then evaluated (Design Space Exploration, box 3 in Fig. 1) based on a mapping model, box 2 in Fig. 1, that captures a selection of the architecture resources that execute the functionality of the application model.
However, when creating a mapping model it is frequent to incur into a *communication mismatch* between the description of communications contained in the application and in the architecture models. This is due to the mismatch between the primitives and operational semantics used to describe communications in the Model of Computation, MoC, of the application (e.g., point-to-point data channel with blocking read()/write() operations) and those in the Model of Computation[1] of the architecture (e.g., data transfer via DMA and bus transactions).

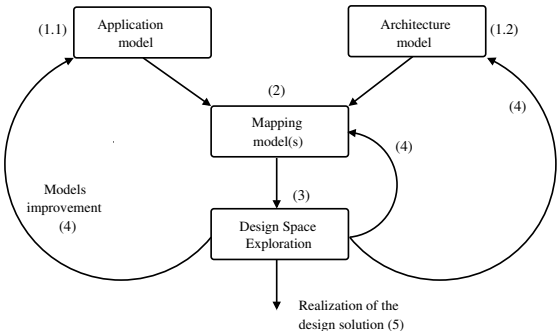

Fig. 1. The Y-chart approach for the design of programmable embedded systems

Several implementations of the Y-chart approach exist that differ in terms of the semantics of the modeling language used to specify a system and or in terms of the purposes and techniques of the DSE phase. Nevertheless, this communication mismatch still remains an open issue. Typically, it is circumvented by designing communications *after* mapping the application onto the architecture, as the mapping of communications depends on the mapping of computations. However, this strategy heavily impacts the modeling phase (i.e., portability) and Design Phase Exploration phase (i.e., rapidity in finding optimal design solutions). Communications that are modeled after the mapping of computations cannot be ported to other target platforms, without additional re-design steps that are time consuming and error-prone (labels 4 in Fig. 1). The DSE of communications occurring after the DSE of computations, local optima are much more likely to be found as a comprehensive view of all design constraints

[1] The Model of Computation of the architecture model is also referred to as Model of Architecture, MoA

is missing.
To solve the communication mismatch, we proposed a novel design approach: the Ψ-chart approach [6], where communication protocols and patterns are modeled independently of the application and architecture models, before mapping.
The structure of this technical report is as follows. Section 2 presents the Ψ-chart approach and details its implementation in TTool/DIPLODOCUS. Section 3 describes the design of the physical layer of the ZigBee transmitter. Section 4 concludes this report with a comparison between design in the Ψ-chart and in the Y-chart and presents the quantitative analysis of the development time.

2 The Ψ-chart design approach

The Ψ-chart, Fig. 2, is an extension of the Y-chart, Fig. 1, where a third input is added to capture communication protocols and patterns independently of the description of communications that is present in the application and architecture models.

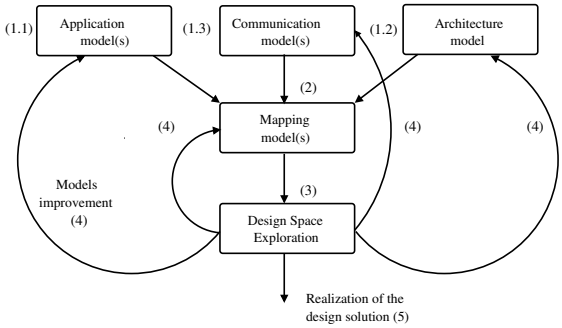

Fig. 2. The Ψ-chart approach for the design of parallel and distributed embedded systems.

Communication protocols and patterns can be modeled independently of a system functionality (i.e., application) and resources (i.e., platform) as illustrated by the following explanation. A communication protocol can be defined as a set of rules for the exchange of information between abstract components (e.g., master, slave, controller). As the rules of a communication protocol are specified regardless of the particular characteristics of an implementation of these abstract components (e.g., an ARM CPU, an Intel CPU), a communication protocol can be modeled regardless of the specific resources of a target platform. Secondly, a protocol specifies the rules to transfer data or control information regardless of the algorithm and the processing operations that produce/consume this information (e.g., Fast Fourier Transforms, vector operations). Therefore, a communication protocol can be described independently of an application model.

From the viewpoint of DSE, expressing communications independently of the platform and application models implies that the performance impact of communication protocols and patterns can be evaluated earlier, faster and better with respect to the case where the specification of communications depends on the application-platform models. In this case, because of the communication mismatch, the resulting design flow is forced to rely on the successive refinement of the application-platform models. As a consequence, at each refinement step, only partial architectural trade-offs can be evaluated that do not account for the performance impact of both communications and computations. To retrieve the overall performance characteristics of the system under design, the user is forced to go through the complete set of refinement steps for more specific references). The development time and cost of (re-)iterating the design process when the overall system performance does not meet the desired requirements are higher and result into a slower and more expensive DSE.

The Ψ-chart approach is described by the following enumerated steps, where each number points to the corresponding label in Fig. 2. In spite of the ordering associated by the numbers, steps 1.1-1.3 can be conducted in any order as the corresponding models can be created independently of each other. On the contrary, the numbering of steps 2-5 coincides with the order dependencies of the design flow.

1.1 Create a model (application) of the system's functionality (e.g., a video-compression algorithm). This model must be created regardless the resources that are available for execution purposes (i.e., hardware or software resources) and must express all potential parallelism between operations. This model must express both the processing of information (e.g., computations) and the dependencies (e.g., communications) between these processing operations.

1.2 Create a model (architecture or platform) of the resources (e.g., bus, CPU, memory, middleware, OS) to support the execution of the applications' functionality. This model must express the topology of the available hardware/software resources, the services that these resources offered (e.g., a bus transaction, and operating system call) as well as the costs of these services (e.g., in terms of silicon area, power consumption, computational power).

1.3 Create a model (communication) describing the communication protocols and patterns used/needed by the target platform to transfer information between processing operations. Such an algorithm must be expressed independently of the semantics associated to the communications described in the application model (i.e., dependencies between computations). Similarly, it must also be expressed independently of the specific semantics of the communications expressed by the platform model's resources and the services (e.g., bus, CPU, memory, middleware, Operating System).

2 Couple each application and communication models to the architecture model (mapping). In this step, a mapping model associates an entity requesting a processing service in the application models (e.g., a data-processing oper-

ation, a control task) to a resource providing the corresponding service in the architecture model (e.g., an Application Specific Integrated Circuit, a general-purpose CPU and its Operating System). Similarly, an entity requesting a communication service in the communication models (e.g., a master) is coupled to a resource providing the corresponding service in the architecture model (e.g., a DMA controller).

3 Explore the design space (e.g., via simulation, formal verification) by evaluating the compliance of the mapping model to a set of pre-defined design requirements (e.g., silicon area, power consumption, computational power), box n. 3 in Fig. 2).

4 In case the mapping models do not meet the requirements, the overall design is modified in order to find alternative solutions (label number 4 in Fig. 2). The application and communication models are arranged so as to express the same functional behavior in a different way. In the architecture model, existing resources are re-structured or new resources are introduced. In the mapping model, new associations between the application, the communications and the architecture are explored.

5 The above steps are repeated iteratively until a solution is found that satisfies all design requirements. At this point, the resulting design is passed to the implementation teams that realize it in terms of hardware and software components.

2.1 The Ψ-chart approach in TTool/DIPLODOCUS

TTool/DIPLODOCUS [7, 8] is a UML/SysML framework for the hardware/-software co-design of data-dominated emdedded systems.
In TTool/DIPLODOCUS, an *application* model is a composition of SysML Block Definition and Block Instance diagrams that describe a data-processing algorithm as a set of blocks interconnected by data and control dependencies via ports and channels. The internal behavior of each block is described by a SysML Activity Diagram. An application is described in terms of the two following abstraction principles:

- *Data abstraction:* only the amount of data exchanged between application blocks is modeled. Internal decisions that depend on the value of data are expressed in terms of non-deterministic and static operators (i.e., conditional choice based on the value of a random variable).
- *Functional abstraction:* algorithms are described using abstract cost operators that express the complexity of processing data in terms of the number of operations required to execute them (e.g., number of integer operations).

A *platform* model is denoted using a UML Deployment Diagram that represents a set of interconnected resources, e.g., bus, CPU and its operating system, DMA, memory. These resources are characterized by performance parameters (e.g., the scheduling policy and the number of cores for a CPU) that are used for DSE (e.g., simulation, formal verification) and by implementation characteristics (e.g., addresses of memory areas) that are used for rapid prototyping (i.e.,

control code synthesis).
A *communication* model is also called a Communication Pattern (CP) [6]. A CP is composed of SysML Activity and Sequence Diagrams. An Activity Diagram is used to capture the structure and dependencies between the activities that describe the algorithm of a communication protocols (e.g., program a DMA, execute a bus transaction). Each of these activities is then described either directly by Sequence Diagrams or recursively via other Activity Diagrams. Activities are composed by operators to describe concurrency, sequencing, choice and iteration. The latter two are governed by global control variables. The interface of a Communication Pattern is given by the top-most Activity Diagram, called the *main* Activity Diagram. The Sequence Diagrams of a Communication Pattern describe the way components (e.g., master, slave, controller) interact in order to execute an activity. The lifelines of instances of a Sequence Diagram are of three types:

- A *storage component* is an architecture unit whose main functionality is to store input/output data produced or consumed by a processing operation, e.g., a RAM memory, a buffer.
- A *transfer component* is an architecture unit whose main functionality is to physically move data items between components, e.g., a AMBA bus, a CAN bus, a DMA.
- A *controller component* is an architecture unit whose main functionality is to coordinate a data transfer by configuring a *transfer component*, e.g., a Central Processing Unit, a microcontroller, a Digital Signal Processor.

Interactions between lifelines are described via the exchange of parameterized messages (e.g., Read(), Write()) that represent an abstraction of the protocol signals.
A *mapping* model is created from an instance of the platform model where dedicated UML artifacts are added to map the computations and Communication Patterns. The abstract cost operators are assigned a value according to the performance characteristics (e.g., operating frequency) of the platform's units. TTool/DIPLODOCUS allows a user to map functions that belong to different functional views, namely from different application models.
Design Space Exploration in TTool/DIPLODOCUS evaluates the performance of a mapping solution by simulating the workload of computations and data-transfers [9]. A formal verification engine [9] is also available to verify system properties (e.g., liveness, reachability, scheduling). DSE can be performed both manually via the tool's Graphical User Interface or automatically via a set of scripts that configure the DSE engine to evaluate different mapping alternatives.
The above abstraction principles have been defined as TTool/DIPLODOCUS targets early design and DSE, when not all the details about a system's application (e.g., value and type of data) and platform (e.g., Operating System, size and policy of cache memories for a CPU) are known. The validation of the effectiveness of these abstractions has been described in [10], where TTool/DIPLODOCUS was used for the design of the physical layer of a LTE base station jointly with

Freescale Semiconductors. The resulting design in TTool/DIPLODOCUS lead to performance results that differed by only 10% with respect to the final implementation. To obtain these performance figures, design in TTool/DIPLODOCUS required only a few weeks, whereas manual development of a functionally equivalent system amounted to 6 months.

In the next section, we describe the design of the physical layer of a ZigBee transmitter (IEEE 802.15.4) in the Ψ-chart implementation of TTool/DIPLODOCUS. We also compare a design of the same system in the Y-chart implementation of TTool/DIPLODOCUS and quantitatively evaluate the gain between the two approaches in terms of development time.

3 Case Study

In this section, we deploy our implementation of the Ψ-chart in TTool/DIPLODOCUS to design the physical (PHY) layer of a ZigBee (IEEE 802.15.4 standard) [4] transmitter. The IEEE 802.15.4 standard specifies both the MAC and the PHY layers of the IEEE 802.15.4 protocol. It is a standard for low-rate Wireless Personal Area Networks (WPANs), which are used to convey information over relatively short distances. ZigBee has been deployed for several applications which implement some Wireless Sensor Networks (WSN) for building automation, remote control, health care, smart energy, telecom services.
Among the different schemes that exist for a ZigBee transmitter, derived from the IEEE 802.15.4 standard, we selected the one proposed in [5] displayed in Fig. 3, because of its simplicity in terms of implementation.

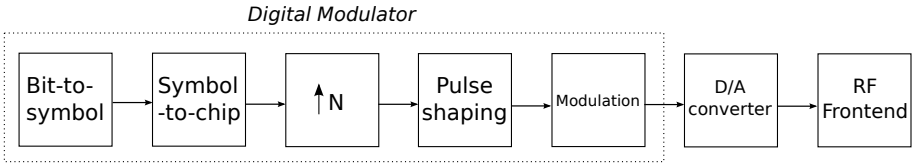

Fig. 3. The functional block diagram of the ZigBee transmitter as proposed in [5]

3.1 The platform

The target hardware architecture for our case study is Embb [3], a generic baseband architecture dedicated to signal processing applications.
Fig. 4a shows the UML Deployment Diagram of the Embb architecture, as modeled in TTool/DIPLODOCUS. Embb is composed of a Digital Signal Processing part (DSP part) and a general purpose control processor (the main CPU). In the DSP part, left-hand side of Fig. 4a, samples coming from the air are processed in

parallel by a distributed set of Digital Signal Processing Units (DSPU1 through DSPUn) interconnected by a crossbar (Crossbar). Fig. 4b illustrates the internal architecture of a DSPU: each unit is equipped with a local micro-controller (μC) that allows to reduce interventions of the main CPU, a Processing Sub-System (PSS) as computational unit and a Direct Memory Access controller (DMA) to transfer data in and out of the DSPU's local memory (the Memory Sub-System, MSS). The latter is mapped on the global address map of the main CPU and is accessible by the DMAs, the μCs and the system interconnect. The system interconnect permits exchanges of control and data items: it is composed of a crossbar (Crossbar), a bridge (Main Bridge) and a main bus (Main Bus). The system interconnect is shared between the DSP part and the main CPU, where the control operations of an application are executed. The main CPU is in charge of configuring and controlling the processing operations performed by the DSPUs and the data transfers. The main CPU disposes of a memory unit (MAINmemory) and a bus interconnect (MAINbus). The latter is linked to the DSP part via the Main Bridge.

To implement the Zigbee transmitter, we configured Embb with four Digital

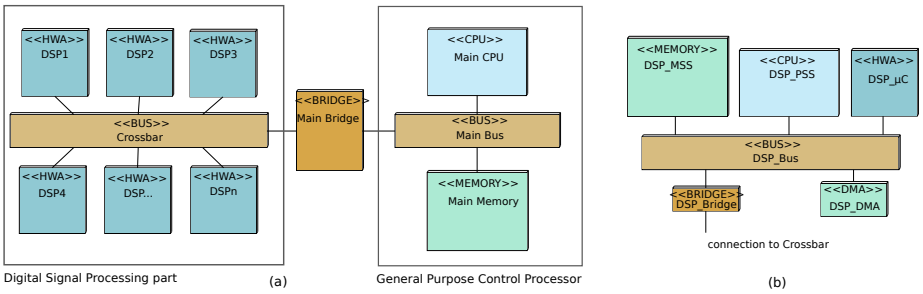

Fig. 4. The UML Deployment Diagrams of Embb. Part (a) displays the global architecture of a Embb instance with its Digital Signal Processing part (left-hand side) and main CPU (right-hand side). Part (b) depicts the internal architecture of each Digital Signal Processing unit

Signal Processing Units:

- Front End Processor (FEP): it implements Discrete Fourier Transform and vector processing operations.
- Interleaver (INTL): it implements permutations (i.e., interleaving and de-interleaving) of sequences of data samples.
- Mapper (MAPPER): it transforms a frame of input symbols into a frame of complex numbers representing the points of a 2D constellation diagram, via Look-Up-Tables.
- Analog to Digital-Digital to Analog Interface (ADAIF): a dispatcher that is capable of both receiving and transmitting up to 4 streams from A/D and D/A converters.

3.2 The application

Fig. 5 shows the TTool/DIPLODOCUS model that represents the functionality of our implementation for the ZigBee transmitter. Here, the block labeled Source produces the data to be transmitted in the form of a flow of bits. These data are then converted to symbols by the Symbol2ChipSeq block. In this block, we model the mapping of each incoming 4-bits symbol to one of the 16 sequences of 32 chips as defined by the IEEE standard 802.15.4. The Chip_to_Octet block, then transforms each incoming chip (bit) of a chip sequence into an unsigned 8-bits integer as expressed in equation 1:

$$\{0; 1\} \rightarrow \{0x00; 0x01\} \tag{1}$$

Chip_to_Octet also separates the even-indexed chips that are used to modulate the in-phase (I branch) carrier component from the odd-indexed chips that are used to modulate the quadrature (Q branch) carrier component. The output is then transformed by means of a Component Wise Lookup (CWL block) that maps unsigned 8-bits integers to signed 16 bits integers as expressed by equation 2:

$$\{0x00; 0x01\} \rightarrow \{0xffff; 0x0001\} \tag{2}$$

At this point, given the separation of the I and Q branches, their pulse shaping can be executed independently. The application graph exposes this parallelism by forking the output data of block CWL to two distinct Component Wise Product (CWP) blocks, CWP_I for the I branch and CWP_Q for the Q branch. These blocks multiply the input samples with a half-sine wave to realize the O-QPSK modulation. The quadrature shift between the I and Q branches is implemented by means of an offset between the memory addresses of the output samples. This results into a frame of complex samples (16 bits for the real part and 16 bits for the imaginary part) that is then collected by block Sink and transmitted over the air.
Each block of the model in Fig. 5 is composed of two tasks: one modeling the data-processing and one modeling the related control operations. By convention we name the data-processing tasks with a heading X that stands for eXecution and the control tasks with a heading F that stands for Firing.

3.3 Communications

The communication mismatch Communications are described as: (i) point-to-point data channels between tasks of the application model and as (ii) read/-write operations performed by CPU and DSP units to/from memory units in the platform model. Therefore, communication mismatches arise when data are transferred via paths, in the platform model, that encompass a sequence of more than one pair bus-bridge between a source CPU/DSP and its destination memory. For instance, when data are transferred (i) from `MainMemory` to

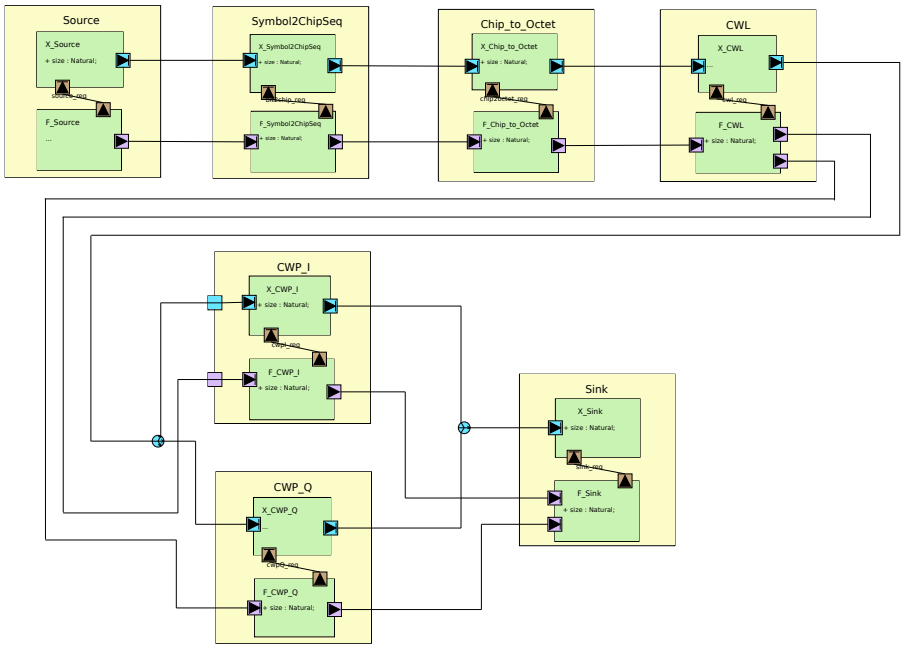

Fig. 5. The TTool/DIPLODOCUS model of the ZigBee transmitter

any of the DSP local memories and vice-versa, and (ii) from a DSP local memory to any other DSP local memory. The only case in which there is a match between the application MoC and the architecture MoA is given by the path `MainCPU-MainBus-MainMemory` or by the path that links any DSP PSS to its local memory.

The platform's communication protocols and mechanisms Data in Embb are transferred in one of the two following ways: (i) via a DMA transfer (to upload data to process in MSS and to download processing results) and (ii) via load/store instructions issued by the main CPU (i.e., General Purpose Control Processor) to read/write data from/to the main memory.

Communication Patterns As described above, the communication protocols and patterns that we need to model are based on DMA transfers with interrupt mechanisms. In these models, described below in the next paragraph, we represented the interrupt signal that is sent by a DMA controller to a CPU controller as the message `TranferTerminated()`.

Modeling a DMA transfer with Communication Patterns The main Activity Diagram of the Communication Pattern for a DMA transfer is illustrated in Fig. 6a. In this diagram we decomposed the communication protocol in three Sequence Diagrams: first the data transfer is configured (ConfigureTransfer in Fig. 6a), then data are transferred (TransferCycle in Fig. 6a) and the data transfer is terminated (TerminateTransfer in Fig. 6a). Data are transferred iteratively, as expressed by the for-loop operator, based on the value assigned to the control variable `counter` in diagram ConfigureTransfer.

The Sequence Diagram ConfigureTransfer is depicted in Fig. 6b. Here, we model

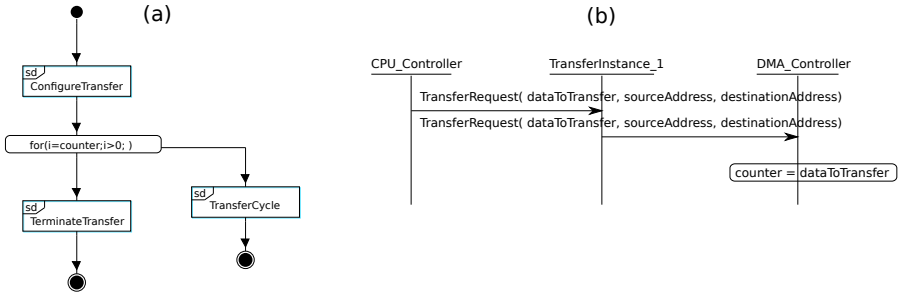

Fig. 6. The main Activity Diagram of a DMA data transfer (a), Sequence Diagram ConfigureTransfer (b).

how a generic CPU unit configures the DMA controller unit. These two units are represented as two instances of type controller, interconnected by an instance of type transfer. The CPU instance sends the source and destination addresses as well as the amount of data to transfer as parameters of the message `TransferRequest()` to the DMA controller, via the transfer instance. The DMA controller, upon reception of the message, assigns variable `dataToTransfer` to `counter`. The value of these variables is not known at modeling phase as CPs are independent of the data dependencies in the application model. A value will be assigned at mapping phase.

In the Sequence Diagram of Fig. 7, TransferCycle, we model one DMA transfer cycle. For this purpose we instantiate the DMA controller of Fig. 6b, a source and destination storage instances interconnected by two transfer instances. In this diagram, the DMA controller reads samples out of the source storage instance via a parameterized `Read()` message. Subsequently, it writes data to the destination storage instance via a `Write()` message and decrements the control variale `counter` that governs the for-loop of Fig. 6. As the values of parameters `size`, `sourceAddress` and `destinationAddress` depend on the architecture units, they will be assigned a value when mapping the instances onto DMA and memory units. The message parameter `size` defines the amount of data that the DMA channel can transfer per each transfer cycle.

In the Sequence Diagram TerminateTransfer of Fig. 8, the DMA controller in-

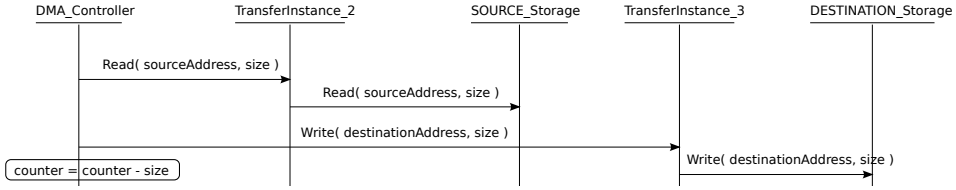

Fig. 7. The Sequence Diagram TransferCycle of Fig. 6a.

forms the CPU instance that the transfer is terminated via an acknowledgment message. Another novel Communication Pattern that we need in this case study

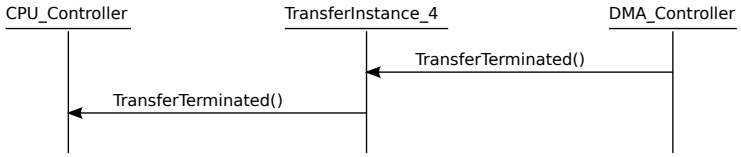

Fig. 8. The Sequence Diagram TerminateTransfer of Fig. 6a.

is the one depicted in Fig. 9a. Here, the main Activity Diagram captures a memory copy transfer. This transfer is used in the context of Embb to model a data transfer from the `MainMemory` to the local memory of any DSP unit, via a store operation issued by the `MainCPU`.
Additionally, we used the CP illustrated in Fig. 9b. This model captures a pair of sequential DMA transfers and can be used to model a copy operation from one source storage to two different destination storages. The main Activity Diagram of this Communication Pattern is composed of two references to Activity Diagrams, that each describe a DMA transfer as the one illustrated in Fig. 9b (DMATransfer1).

3.4 The mapping

We first map the computations of the application model. Such a mapping results in each control task (e.g., F_Symbol2ChipSeq) being executed to the Main CPU unit and the data-processing tasks (e.g., X_Symbol2ChipSeq) being executed to the DSPUs PSS. Secondly, the memories where to store input/output data are chosen. This results into a mapping where the local memory of each DSPU is used to store the input/output data for the computations that have been mapped onto the DSP's Processing SubSystem, e.g., task X_Symbol2ChipSeq is mapped to the Mapper PSS, the input/output data are mapped to the Mapper local Memory SubSystem (MSS).
From our library we instantiate and map 4 Communication Patterns:

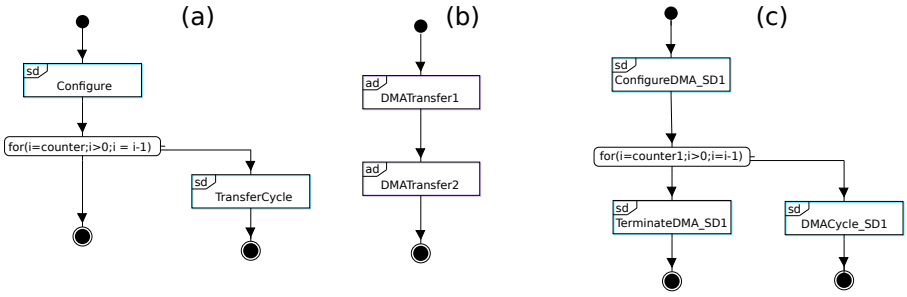

Fig. 9. The main AD for a CP modeling a CPU memory copy (a). The main AD for a CP modeling the sequence of two DMA transfers (b). Part (c) shows the AD referenced by DMATransfer1 in (b)

- *CP01*: a memory copy CP that transfers the output data of task X_Source. It is composed of: 1 controller instance (CPU_Controller), 2 storage instances (Src_Storage, Dst_Storage) and 2 transfer instances.
- *CP02*: a DMA CP that transfers the output data of X_Symbol2ChipSeq. It is composed of: 2 controller instances (CPU_Controller, DMA_Controller), 2 storage instances (Src_Storage, Dst_Storage) and 4 transfer instances.
- *CP03*: a DMA CP that transfers the output data of X_Chip2Octet. It is composed of: 2 controller instances (CPU_Controller, DMA_Controller), 2 storage instances (Src_Storage, Dst_Storage) and 4 transfer instances.
- *CP04*: the sequence of two DMA CPs that transfer the output data of X_CWP_I and X_CWP_Q. It is composed of: 4 controller instances (2 CPU_-Controllers, 2 DMA_Controllers), 4 storage instances (2 Src_Storage, 2 Dst_Storage) and 8 transfer instances.

Table 1 and Table 2 show the mapping of the above Communication Ptterns onto the platform units of Fig. 4.

Table 1. The mapping of the CPs' controller and storage instances

Identifier	Instance	Architecture unit
CP01	Src_Storage, Dst_Storage	Main Memory $\text{MAPPER}_{\text{MSS}}$
CP02	DMA_Controller, Src_Storage, Dst_Storage	$\text{MAPPER}_{\text{DMA}}$ $\text{MAPPER}_{\text{MSS}}$ INTL_{MSS}
CP03	DMA_Controller, Src_Storage, Dst_Storage	INTL_{DMA} INTL_{MSS} FEP_{MSS}
CP04	DMA_Controllers, Src_Storages, Dst_Storages	FEP_{DMA} FEP_{MSS} $\text{ADAIF}_{\text{MSS}}$

Table 2. The mapping of the CPs' transfer instances

Identifier	Transfer instance 1	Transfer instance 2	Transfer instance 3	Transfer instance 4
CP01	Main Bus Main Bridge Crossbar MAPPER Bridge MAPPER Bus	Main Bus Main Bridge Crossbar MAPPER Bridge MAPPER Bus		
CP02	Main Bus Main Bridge Crossbar MAPPER Bridge MAPPER Bus	MAPPER Bus	INTL Bus INTL Bridge Crossbar MAPPER Bridge MAPPER Bus	MAPPER Bus MAPPER Bridge Crossbar Main Bridge Main Bus
CP03	Main Bus Main Bridge Crossbar INTL Bridge INTL Bus	INTL Bus	FEP Bus FEP Bridge Crossbar INTL Bridge INTL Bus	INTL Bus INTL Bridge Crossbar Main Bridge Main Bus
CP04	Main Bus Main Bridge Crossbar FEP Bridge FEP Bus	FEP Bus	FEP Bus FEP Bridge Crossbar ADAIF Bridge ADAIF Bus	FEP Bus FEP Bridge Crossbar Main Bridge Main Bus

4 A comparison between Design Space Exploration in the Y-chart and in the *Ψ*-chart

In this subsection we compare the design of the ZigBee transmitter described so far with a design of the same system conducted in the version of TTool/DIPLODOCUS, previous to our contributions, that was based on the Y-chart approach. In order to circumvent the communication mismatch in TTool/DIPLODOCUS, the Y-chart approach results in the methodology depicted in Fig. 10, as opposed to the ideal design flow of Fig. 1. Once an application (step 1) and architecture (step 2) models are available, the processing tasks are mapped (step 3) onto the processing units of the platform. In this phase, also the data channels of the application model are mapped onto the memory units to provide information about where the data are read from and/or written to. In Fig. 10 we called this model *Partial mapping* model. Next, a second model for the system functionality is created, step 4, from the initial application model (step 1) and from the Computations mapping model (step 3). We called this *Hybrid application model* to distinguish it from the initial application model that we called *Pure application* model as it does not contain platform-dependent information. Indeed, the Hybrid application model is an instance of the Pure application model, where additional tasks are added between the processing operations, in order to capture

the communication protocols and patterns of the platform. These communication tasks are added when one or more communication mismatches at step 3 prevent the creation of a unique mapping model. For the case study described in this chapter, Fig. 11 shows the hybrid application model where dedicated tasks (i.e., DMAmapper, DMA_INTL, DMA_FEP) have been added to model the DMA transfers. In Fig. 11, the memory copy transfer between tasks X_Source and X_Bits2Symbol does not need to be modeled via a dedicated task as it does not correspond to a communication mismatch between the application MoC and the architecture MoA. The data transfer to the sink operation (DMA_FEP) is modeled as a single task, instead of two separate transfer operations, in order not to force the scheduling of operations CWP_I and CWP_Q.
From the Hybrid application model the tasks dedicated to capture communications and their data channels are mapped, step 5 in Fig. 10. This results into a second mapping model, called *Complete mapping* model that is then used to perform Design Space Exploration (step 6 in Fig. 10).

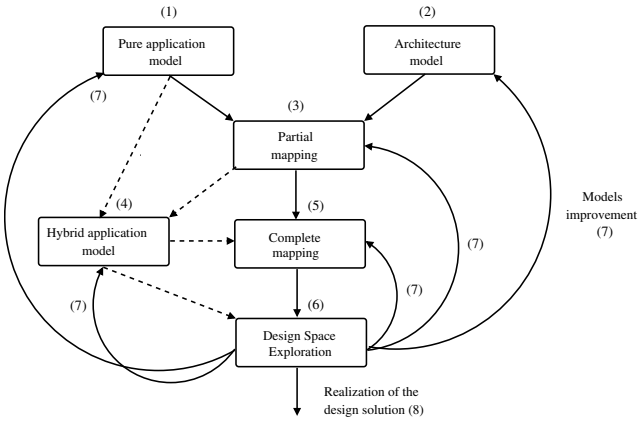

Fig. 10. Design in the version of TTool/DIPLODOCUS based on the Y-chart. Communications are designed in the Hybrid application model, (4), after mapping the computations in the Pure application model (1)

As described above, to cope with the communication mismatches the user is forced to decouple the mapping step in two separate phases, one dedicated to the mapping of computations (step 3 in Fig. 10) and the second one dedicated to the mapping of communications (step 5 in Fig. 10). As a consequence of this decoupling, two additional models, i.e., the Complete mapping and the Hybrid application, are added to the design flow. On one hand, in terms of Design Space Exploration, this decoupling limits the degree of freedom that a designer has when it comes to evaluate the impact of computations and communications. In fact, in the Partial mapping model (step 3 in Fig. 10) only the charge of the

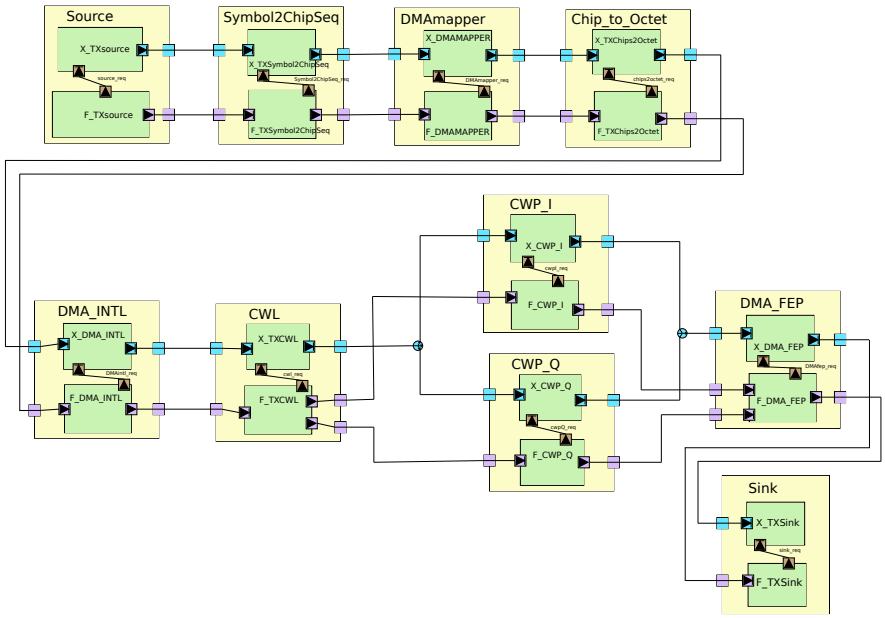

Fig. 11. The TTool/DIPLODOCUS Hybrid application model of the ZigBee transmitter

processing units can be evaluated. On the contrary, the exploration of design alternatives in the Complete mapping model, step 5, is limited by the fact that communications have been modeled at step 4 in Fig. 10 depending on the mapping of computations. Therefore to evaluate alternate communication schemes, a user has to first change the mapping of the processing operations and then adapt the hybrid application model at step 4 in Fig. 10. This results into a tedious and error-prone cycle of iterations between DSE and modeling at a level (steps 3-5) where there is no separation of concerns between the functionality of the system and the platform.

On the other hand, in terms of design productivity, in the design based on the Y-chart, 5 models (steps 1-5 in Fig. 10) are eligible for modifications after the evaluation of the DSE phase. On the contrary, when the design is conducted in the frame of the Ψ-chart approach, only 3 models (application, architecture and mapping) are eligible for modifications. In fact, Communications Patterns are instantiated from a library and therefore are not eligible for modifications as they are created at mapping time. Moreover, evaluating the impact of both computations and communication protocols only requires to change the mapping of processing functions and Communication Patterns.

4.1 Quantitative analysis

Equation 12 provides a quantitative evaluation of the development time that is saved, at mapping phase, in the TTool/DIPLODOCUS implementation of the Ψ-chart (Fig. 2), with respect to the implementation of the Y-chart approach (Fig. 10). The gain between the mapping times of the two approaches is expressed as a function of ${t_{\text{block}}}^{\text{map}}$ the time required to map a block, ${t_{\text{block}}}^{\text{modeling}}$ the time required to model a block, n the number of computation blocks and m the number of communication blocks in a design. Equation 12 is based on the following symplifying assumptions: (i) the time required to map a Communication Pattern is equal to the time required to map communications in the Hybrid application model (step 4, Fig. 10); (ii) the time required to map/model a block in the Hybrid application model is constant.

This evaluation is based on the following mathematical analysis. Equation 3 expresses the time spent in the mapping phase of the Y-chart implementation of TTool/DIPLODOCUS in Fig. 10.

$$ {t_{\text{Ychart}}}^{\text{map}} = {t_{\text{comp}}}^{\text{map}} + {t_{\text{Hybrid}}}^{\text{modeling}} + {t_{\text{Hybrid}}}^{\text{map}} \tag{3} $$

where:

- ${\text{t}_{\text{comp}}}^{\text{map}}$ is the time spent to map computations, step 3 in Fig. 10.
- ${\text{t}_{\text{Hybrid}}}^{\text{modeling}}$ is the time spent to create the Hybrid application model, step 4 in Fig. 10.
- ${\text{t}_{\text{Hybrid}}}^{\text{map}}$ is the time spent to map the blocks representing communication protocols and patterns in the Hybrid application model, step 5 in Fig. 10.

Equation 4 expresses the time spent in the mapping phase of the Ψ-chart (step 2 in Fig. 2) implementation of TTool/DIPLODOCUS. We underline here that there is no difference between the generic approach of Fig. 2 and its implementation, as opposed to the ideal scenario of the Y-chart in Fig. 1 and its implementation in Fig. 10.

$$ {t_{\Psi\text{chart}}}^{\text{map}} = {t_{\text{comp}}}^{\text{map}} + {t_{\text{CPs}}}^{\text{map}} \tag{4} $$

where:

- ${\text{t}_{\text{comp}}}^{\text{map}}$ is the time spent to map computations.
- ${\text{t}_{\text{CPs}}}^{\text{map}}$ is the time spent to map Communication Patterns.

The absolute gain is defined as the difference between the mapping time spent in the Y-chart and in the Ψ-chart, as displayed by equations 5 and 6.

$$ \Delta t^{\text{map}} = {t_{\Psi\text{chart}}}^{\text{map}} - {t_{\text{Ychart}}}^{\text{map}} \tag{5} $$

$$ \Delta t^{\text{map}} = {t_{\text{CPs}}}^{\text{map}} + {t_{\text{comp}}}^{\text{map}} - {t_{\text{Hybrid}}}^{\text{modeling}} - {t_{\text{Hybrid}}}^{\text{map}} - {t_{\text{comp}}}^{\text{map}} \tag{6} $$

To simplify equation 6, we assume that the time spent to map Communication Patterns is equal to the time spent to map the blocks that are added to model communication protocols and patterns in the Hybrid application model. This results into equations 7 and 8.

$$\Delta t^{\text{map}} = -{t_{\text{Hybrid}}}^{\text{modeling}} \tag{7}$$

$${t_{\Psi\text{chart}}}^{\text{map}} - {t_{\text{Ychart}}}^{\text{map}} = -{t_{\text{Hybrid}}}^{\text{modeling}} \tag{8}$$

From equation 8, the relative gain in terms of mapping time is computed as the ration between the single mapping times spent in the two different approaches, equation 9.

$$\frac{{t_{\Psi\text{chart}}}^{\text{map}}}{{t_{\text{Ychart}}}^{\text{map}}} = 1 - \frac{{t_{\text{Hybrid}}}^{\text{modeling}}}{{t_{\text{Ychart}}}^{\text{map}}} \tag{9}$$

From equation 3, we can express ${t_{\text{Hybrid}}}^{\text{modeling}}$ and ${t_{\text{Ychart}}}^{\text{map}}$ in terms of the number of computation blocks n, of the number of communication blocks m, of the time taken to model a block ${t_{\text{block}}}^{\text{modeling}}$ and of the time taken to map a block ${t_{\text{block}}}^{\text{map}}$, as expressed in equations 10 and 11.

$${t_{\text{Hybrid}}}^{\text{modeling}} = n{t_{\text{block}}}^{\text{modeling}} \tag{10}$$

$$\begin{aligned} {t_{\text{Ychart}}}^{\text{map}} &= {t_{\text{comp}}}^{\text{map}} + {t_{\text{Hybrid}}}^{\text{modeling}} + {t_{\text{Hybrid}}}^{\text{map}} \\ &= n{t_{\text{block}}}^{\text{map}} + (n+m){t_{\text{block}}}^{\text{modeling}} + m{t_{\text{block}}}^{\text{map}} \\ &= {t_{\text{block}}}^{\text{map}}(n+m) + {t_{\text{block}}}^{\text{modeling}}(n+m) \\ &= ({t_{\text{block}}}^{\text{modeling}} + {t_{\text{block}}}^{\text{map}})(n+m) \end{aligned} \tag{11}$$

$$\frac{{t_{\Psi\text{chart}}}^{\text{map}}}{{t_{\text{Ychart}}}^{\text{map}}} = 1 - \frac{{t_{\text{Hybrid}}}^{\text{modeling}}}{{t_{\text{Ychart}}}^{\text{map}}} = 1 - \frac{n{t_{\text{block}}}^{\text{modeling}}}{({t_{\text{block}}}^{\text{modeling}} + {t_{\text{block}}}^{\text{map}})(n+m)} \tag{12}$$

We measured the time taken to map and model a block in TTool/DIPLODOCUS and assigned a value to ${t_{\text{block}}}^{\text{map}}$ of 6 seconds and to ${t_{\text{block}}}^{\text{modeling}}$ of 60 seconds. Substituting these values and the total number of computation and communication blocks for the ZigBee transmitter of Fig. 11 ($n = 10$, $m = 3$) to equation 12 results into a gain, for the Ψ-chart approach, of 30%.

References

1. Kienhuis, B., Deprettere, E.F., Vissers, K., van der Wolf, P.: An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures. In: International Conference on Application-Specific Systems, Architectures and Processors (ASAP), pp. 338-349 (1997),
2. Kienhuis, B., Deprettere, E.F., van der Wolf, P., Vissers, K.: A Methodology to Design Programmable Embedded Systems - The Y-chart Approach. In: Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation - SAMOS, pp. 18-37 (2002)

3. Muhammad, N.-u.-I., Rasheed, R., Pacalet, R., Knopp, R., Khalfallah, K.: Flexible Baseband Architectures for Future Wireless Systems. In: EUROMICRO Digital System Design, pp. 39-46. (2008)
4. IEEE 802.15.4: IEEE 802.15 Wireless Personal Area Networks (WPAN) Task Group 4 (TG4). Available at: [http://www.ieee802.org/15/pub/TG4.html] (2003)
5. Koteng, R.M.: Evaluation of SDR-implementation of IEEE 802.15.4 Physical Layer. Master thesis, Norwegian University of Science and Technology (NTNU) (2006)
6. Enrici, A., Apvrille, L., Pacalet, R.: A UML Model-Driven Approach to Efficiently Allocate Complex Communication Schemes. In: MODELS, pp. 370-385, (2014)
7. Apvrille, L., Muhammad, W., Ameur-Boulifa, R., Coudert, S., Pacalet, R.: A UML-based Environment for System Design Space Exploration. In: Electronics, Circuits and Systems (ICECS), pp. 1272-1275 (2006)
8. Apvrille, L.: TTool for DIPLODOCUS: An Environment for Design Space Exploration. In: NOTERE, pp. 28:1-28:4 (2008)
9. Knorreck, D.: UML-Based Design Space Exploration, Fast Simulation and Static Analysis. PhD dissertation, Telecom ParisTech (2011)
10. Jaber, C.: High-Level SoC modeling and performance estimation applied to a multi-core implementation of a LTE enodeb physical layer. PhD disseration, Telecom ParisTech (2011)