Chapter 3

EMBEDDED SYSTEM DESIGN USING PUSSEE METHOD

An overview

Nikolaos S. Voros[1]
Colin Snook[2]
Stefan Hallerstede[3]
Thierry Lecomte[4]

[1] INTRACOM S.A., Patra, Greece
[2] University of Southampton, Southampton, United Kingdom
[4] KeesDA S.A., Grenoble, France
[5] ClearSy S.A., Aix en Provence, France

Abstract: The approach presented in this book relies on the unification of system specification environments for developing electronic systems that are formally proven to be correct (correct-by-construction systems). The key concept conveyed is the formal proof of system properties, which is carried out at every phase of the co-design cycle. The main idea is to build fully functional system models that are formally proven to be correct, and based on them to produce automatically the hardware and the software parts of the system. The approach presented relies on the combined use of UML and B language.

Keywords: Co-design, refinement, decomposition, translation.

1. THE UML AND THE B-METHOD

UML is a visual modelling notation for object-oriented systems. Translation to a formal notation that has adequate tool support, such as B [1] enables a model to be formally verified and validated. These verification and validation processes are not avai lable in the UML even if annotated constraints are added in the UML constraint language, OCL [2]. However,

translation from unrestricted UML models is problematic because B language is not object-oriented and contains write access restrictions between components in order to ensure compositionality of proofs. There are several approaches reported for mapping between UML to B language [3]. The key differences in our approach are that we specialize UML in order to ensure that the resulting B is amenable to verification. To overcome write-access restrictions we provide a translation from many classes (a package) to a single B component. We also provide UML mechanisms to support an event style of modelling and decomposition [4].

The use of B language for designing complex systems has already been reported in literature [5,6]. As opposed to existing approaches, the co-design method presented in this book proposes the use of B language as part of a unified hardware/software co-design framework that supports formal proof of system properties throughout the various co-design phases.

For the design of hardware system parts, the B language has been applied to circuit design by various authors [7,8]. In [7] Event B is used to specify and refine a circuit but the approach does not provide a translation into a hardware description language. In the context of the work presented in this book we have extended their work in Event B, while we have developed and used a BHDL translator[1] to generate SystemC and VHDL. The approaches in [8,9] model circuits close to an implementation level. The first one requires that the model uses basic logic gates which are modelled in terms of B machines; the second one allows higher data-types like integers in their models, while it introduces new structuring mechanisms into B which mirror those of VHDL closely. The approach presented differs from them in that it does not use an explicit representation for the system clock.

2. PROVEN ELECTRONIC SYSTEM DESIGN USING FORMAL PROOF OF SYSTEM PROPERTIES

The approach introduced is outlined in Figure 3-1. The properties of a system are formally described (and proven) at every design phase. The key aspects of proposed framework are:

System Modelling using a graphical formal notation. For that purpose a specialized profile of UML, called UML-B profile, has been defined in order to allow designers employ UML for defining system models and their properties during early design stages (upper part of Figure 3-1). The

[1] BHDL is a registered trademark of KeesDA S.A., France.

outcome of this process is a set of system models that can be formally proven to be compliant with the initial system specifications.

Formally proven to be correct refinement where the UML-B models already available can be translated automatically to B language. For proving the validity of a model refinement, the B compliant UML models are automatically translated to B where the B language proving mechanisms are employed in order to prove formally the validity of the refinement.

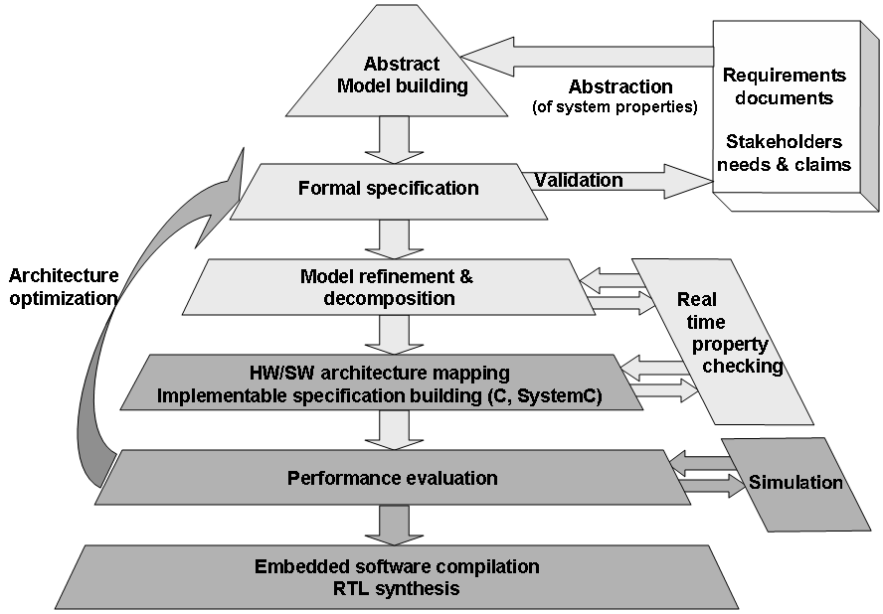

Figure 3-1. Overview of proposed co-design framework (courtesy of KeesDA)

System decomposition into subsystems. Each subsystem can be mapped either to hardware or to software. The initial system is gradually refined and each refinement is proven to be correct. The later is guaranteed through the discharge of all the proof obligations generated (each proof obligation represents a system property that must not violated during refinement). At a specific refinement level, where the system representation is accurate enough, the system is decomposed into subsystems. Each subsystem can be further refined until a fully functional subsystem is reached. The emerging subsystems and the communication between them are described in Event B and they are produced automatically from the decomposition assistant tool developed in the context of the specific approach.

Hardware/software allocation (lower part of Figure 3-1) takes place through direct translations of the emerged subsystems through appropriate translators. The software parts of the system are implemented in C/C^{++},

while the hardware parts of the system are described in VHDL/SystemC. It is important to mention that in both cases, the code produced stems from system models that are formally proven to be correct.

During the last stages of system design, the subsystems produced are simulated together in order to verify their integrated behaviour, while the overall system performance is evaluated. Based on the performance evaluation results, alternative architectures can be explored.

The next sections present the key concepts introduced by the proposed co-design framework.

3. UML-B PROFILE

The UML-B is a profile of UML that defines a subset and specialisation of UML that has a mapping to, and is therefore suitable for translation into B language. The UML-B profile consists of class diagrams with attached statecharts, and an integrated constraint and action language that is based on the B AMN notation. The UML-B profile uses stereotypes to specialise the meaning of UML entities, thus enriching the standard UML notation and increasing its correspondence with B concepts. The UML-B profile also defines tagged values (UML-B clauses) that may be attached to UML entities. UML-B clauses are used to attach details that are not part of the standard UML. Many of these clauses correspond directly with those of B providing a "fallback" mechanism for modelling directly in B when UML entities are not suitable. A few additional clauses, having no direct B equivalent, are provided for adding specific UML-B details to the modelling entities. Several styles of modelling are available within UML-B. Here we use its event systems mode, which corresponds with the Event B modelling paradigm.

UML-B provides a diagrammatic, formal modelling notation. It has a well defined semantics as a direct result of its mapping to B. As with most formal notations there is a strong resistance to the use of B in industrial settings. The popularity of the UML enables UML-B to overcome this resistance. Its familiar diagrammatic notations make specifications accessible to domain experts who may not be familiar with formal notations such as B. UML-B hides B's infrastructure, packages mathematical constraints and action specifications into small sections each being set in the context of its owning UML entity.

4. FORMAL MODEL REFINEMENT

In an event driven approach (like the one presented in this book), refinement performed is only data refinement, as there is no real algorithmic refinement. As described in Figure 3-2, data refinement is based on the introduction of new modelling variables, replacing an existing variable or only complementing the model. The link between the variables of the abstraction and those of the current refinement is located in the invariant and is called *gluing invariant*.

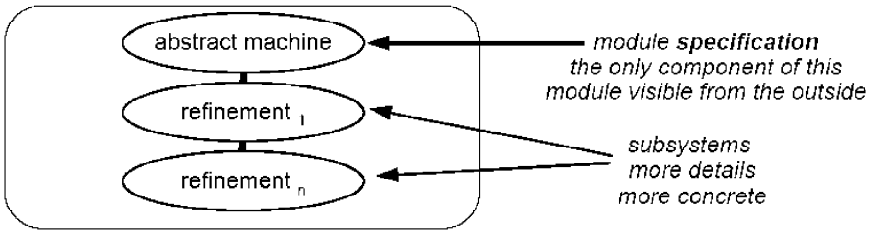

Figure 3-2. Model refinement

Abstraction variables and refinement variables with the same names are considered identical and an implicit gluing invariant is generated ($v_{abstraction} = v_{refinement}$). All new variables must be linked with abstraction variables.

Algorithmic refinement is performed when event based decomposition has lead the designers to identify software/hardware components. Software events for example, can be combined to constitute a single event which represents the specification of one procedure. As such, algorithm specified within this procedure is likely to be refined.

5. BHDL PROFILE

B machines that model hardware, here referred to as BHDL (see Chapter 7), differ in their syntax from B machines that model software. Currently there are BHDL translators to VHDL and SystemC. The VHDL code produced is synthesizable, following the IEEE 1076.6 (draft) standard. The basic data types available in BHDL have been chosen to match with basic SystemC types. The choice of these data types has been made because they are easily portable to other hardware description languages. The substitution language of BHDL is a subset of the substitution language of

B [1]. BHDL machines are expected to be cycle accurate; so while loops have to be encoded using state machines.

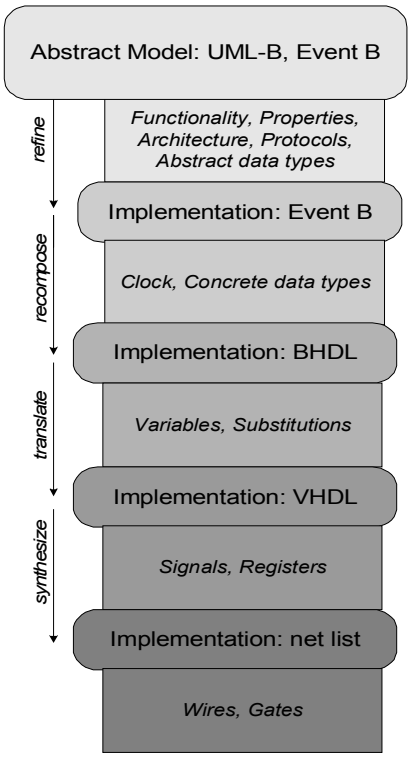

Figure 3-3. Typical BHDL flow

To commence a development on a more abstract level Event B can be used. There is no translation from Event B but a path to BHDL following a simple technique called *event re-composition*. The elements of the BHDL substitution language are interpreted in terms of hardware. Sequential substitution corresponds to causality, simultaneous substitution to concurrency and conditional substitution to multiplexing. Registers are inferred from variables depending on their use. As a rule of thumb, a variable that is read before it is written represents a register. Formally, the translation into hardware description languages is based on the before-after predicate [1], which relates a B substitution to a predicate. Figure 3-3 outlines how the approach is applied to hardware design based on BHDL.

From the abstract model, which is specified in UML-B, we arrive by refinement at an implementable model (still specified in Event B). This model must be cycle-accurate (hw/sw architecture mapping phase in

Figure 3-1). By recomposing events into more complex events we achieve a BHDL model. This event recomposition can be carried out automatically. Once a BHDL model has been produced, this can be translated into VHDL (or SystemC) and subsequently be synthesised. We have successfully synthesised BHDL models using several commercially available VLSI and FPGA synthesis programs. The use of BHDL in this design flow bridges the gap between the often very abstract models produced in Event B and existing synthesis tools. Thus, it delivers within our co-design approach a means to produce correct hardware.

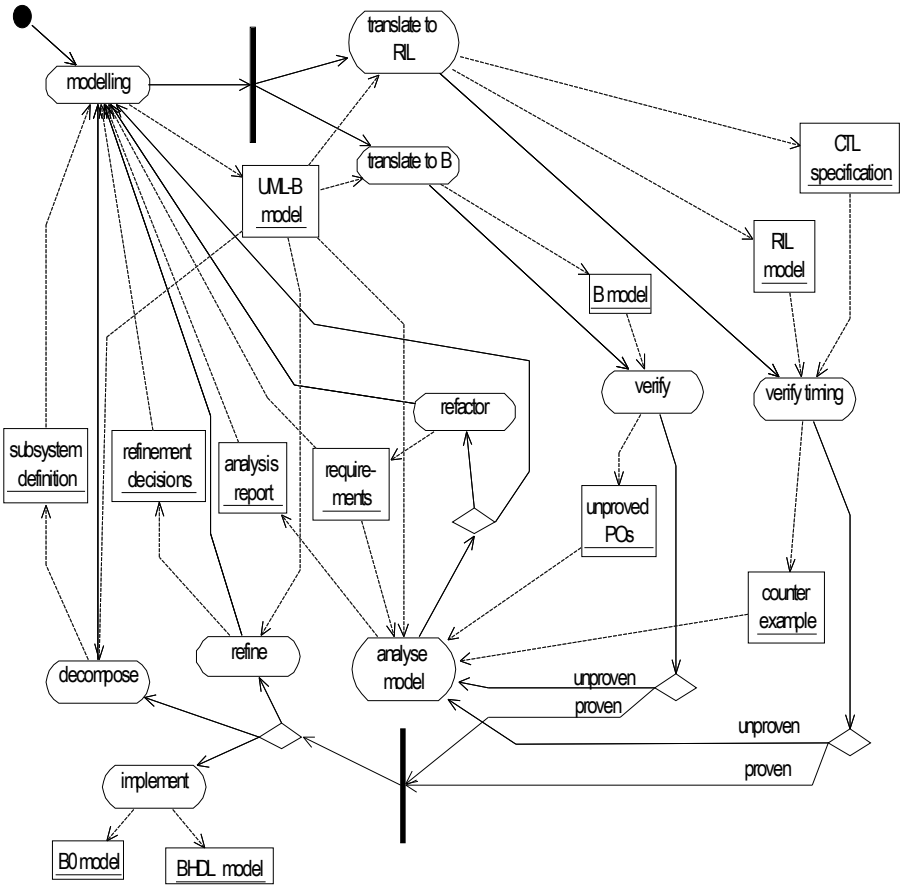

Figure 3-4. Design activities supported by the tool suite

6. THE SUPPORTING TOOLSET

The co-design framework proposed, as outlined in Figure 3-1, is supported by a set of tools that allow system designers to produce systems that are composed of correct-by-construction subsystems. The tools employed in the context of the specific approach cover the all the co-design stages from specification down to implementation.

Figure 3-4 shows this iterative process in more detail, specifically for the notations investigated on the PUSSEE project [10], and illustrates the main information items involved. The semi-formal model is expressed in UML-B (see Chapter 5). The UML-B model is then translated to B and RIL using the U2B (see Chapter 6) and U2R translation tools. RIL and U2R are the subject of Chapter 10.

The proof tools (Atelier B) generate proof obligations and attempt to discharge them automatically. Usually the automatic prover is unable to discharge all the PO's. The PO's should be examined to see whether they indicate an inconsistency in the model. If they appear to be genuine, the interactive prover should be used in an attempt to discharge them. This may still reveal inconsistencies in the model or it may be found that the proof obligations appear to be correct but are just too difficult to prove. If some proof obligations remain unproved they are analysed in the UML-B model to determine the changes needed to achieve proof. The changes are either corrections or re-modelling to make the proofs easier: in some cases it may be found that previous levels of abstraction need to be altered in a 'refactoring' of the model. The experience gained from this analysis can lead to the requirements being changed keeping in mind that abstract modelling serves foremost to detect faults and gaps in the requirements.

The Raven model checker (described in Chapter 10) checks the RIL model against a CTL specification. The CTL specification has to be derived from the modelling process, e.g. from the UML-B model. Verification may fail either due to a violation of the given CTL specification or due to "state explosion". In case of failure, the model and its specification have to be analysed and corrected. Discovery of a violated CTL specification will in many cases be accompanied by a counter-example to support analysis. Failure due to the "state-explosion" problem may be assumed, after a reasonable amount of time has passed.

Once the model has been successfully verified, both functionally and temporally, it can be developed further. Eventually, it can be implemented. Implementation models are written in B0 [1] or BHDL respectively [11]. Using the corresponding translators these can be translated into software or hardware descriptions. In the PUSSEE project the hardware description has been newly developed.

A system may be decomposed into subsystems when it becomes too large and complicated to verify or comprehend. The decomposition tool assists in carrying out decompositions. Allocation of variables and events to subsystems need to be specified, and the corresponding subsystems are generated automatically. The definition of the subsystems can also serve for documentation and further analysis, e.g. number of shared variables, number of shared events, etc.

6.1 U2B translator

The U2B translator converts UML-B models into B components. Translation from UML-B into B is necessary to gain access to other tools in the toolset.

In many respects B components resemble an encapsulation and modularization mechanism suitable for representing a class. A component encapsulates variables that may only be modified by the operations of the component. UML-B models can be constructed to utilize this natural correspondence between UML classes and B components. Component operations may call the operations of other components when a navigable association exists between the corresponding classes. However, to ensure compositionality of proof, B imposes restrictions on the way variables can be modified by other components (even via local operations). Using this first option imposes corresponding restrictions on the relationships between classes. Only hierarchical class association structures can be modelled. A second option translates a complete UML package (i.e. many classes and their relationships) into a single B component. To use this option a stereotype is given to the package depending on which kind of B component is required. The instance, attributes, associations and operations of the classes in the package are represented in B in the same way but are collated into a single component. This option allows unconstrained (non-hierarchical) class relationship structures to be modelled but no operation calling is possible because all operations are within the same B component (a further restriction for proof reasons). However, if we view the operation bodies as declarative specifications of behaviour rather than encompassing design issues such as how that behaviour is allocated to operations throughout the system, this is not a severe restriction on the UML modelling.

Since B language is not object oriented, class instances must be modelled explicitly. Attributes and associations are translated into variables whose type is a function from the class instances to the attribute type or associated class. For example a variable instance class A with attribute x of type X would result in the following B component:

```
MACHINE         A_CLASS
SETS            A_SET
VARIABLES       A, x
INVARIANT       A : POW(A_SET) & x : A --> (X)
INITIALISATIONA := {} ||  x := {}
...
```

Operation behaviour may be represented textually in a notation based on B's AMN (abstract machine notation) or as a state chart attached to the class or as a simultaneous combination of both.
Further details of the UML-B modelling language, which is a profile of the UML, are given in Chapter 5. A full description of how UML-B models are translated into B by the U2B translator is given in Chapter 6.

6.2 Atelier B

The use of B language for describing and proving system properties between successive model refinement is crucial for the specific approach. The total number of proofs required, even for small systems, is usually big and difficult to handle without the support of an appropriate tool.

For the automation of proving process Atelier B[2] tool is employed. It is mainly composed of static analyzers that include (a) a *type checker*, which is responsible for the syntactic and the semantic verification of a B component, (b) a *B0 checker* which performs verification specific to the B0 language[3] [1] and (c) the *project verification analyzer*, which performs global verifications on the among system components in order to control the overall system architecture.

Additionally, Atelier B includes proof tools that allow formal proof of the successive B model refinements, where the proof obligations required can be proven either automatically or interactively. More specifically, the proof tools available from Atelier B include:

- The automatic generator of proof obligations from the components in B.
- The rule base manager (the rule base includes more than 2200 rules).
- The automatic prover, which discharges automatically most of the proof obligations. It can be set to a power level to compromise between its speed and proof rate.
- The interactive prover, which is used when the automatic prover has failed. The designer is able to guide the prover with commands and additions of new rules.

[2] Atelier B is a trademark of ClearSy S.A.
[3] B0 is a subset of B language, which is used only in implementations to ensure that those can be directly translated in C/C++/Ada.

- The predicate prover, which demonstrates rules added by the user.
- The graphical visualization of the proof tree of a proof obligation.

Finally, Atelier B supports translation of B implementations to C, C^{++} and Ada for the software parts of the system under development.

6.3 Decomposition assistant

The decomposition assistant[4] automates the decomposition process [12,13] outlined in Figure 3-5. Decomposition is precisely the process by which a certain model can be split into various component models in a systematic fashion. As a result, the complexity of the proving process is reduced while the emerging subsystems can be implemented using different technologies. The communication among the components is automatically generated by the decomposition assistant. Communication consistency is based on the exchange of events among the subsystems and can be formally refined until the final communication scheme is reached.

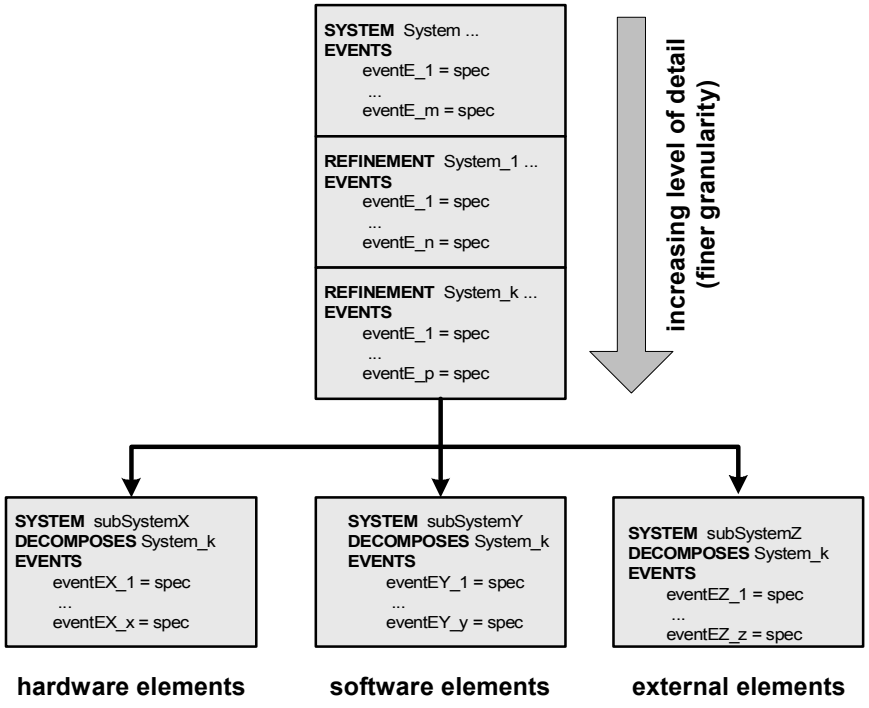

Figure 3-5. System decomposition and hardware/software allocation

[4] The decomposition assistant has been developed by ClearSy S.A. in the context of IST PUSSEE Project [10] .

As described in Figure 3-6, during the decomposition process each variable is allocated to one and only one subsystem and references in other subsystems are copies that need to be updated. The lower part of the figure describes the modules generated by the decomposition assistant: SS1 and SS2. ISS1S2, which contains the communication primitives between the two subsystems, can be further formally refined in order to reach a fully functional protocol implementation. System decomposition [14] is based on a decomposition profile defined by the designer. Decomposition assistant uses the decomposition profile as input and automatically produces the description of each subsystem along with the variables required for describing the selected communication scheme. Reuse of formally proven protocol descriptions through a protocol library is also applicable.

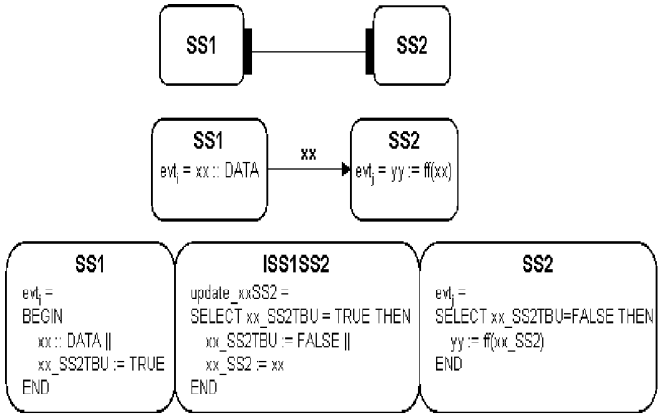

Figure 3-6. Communication between subsystems, as it is generated by the decomposition assistant tool

6.4 BHDL translator

The BHDL subset [11] of the B language is similar to traditional (procedural) B. Structurally the main differences are the presence of input and output for a module, and the restriction of the operation clause to one substitution. Variables of BHDL machines are split into the categories INPUTS, OUTPUTS and VARIABLES. The first two, the ports of the design, are externally visible with the obvious meaning. The other variables are local to the design. The OPERATION clause of a BHDL machine contains a single substitution describing the behaviour of the design. Type and constant declarations may not be made in this kind of BHDL machines.

They must be made in dedicated BHDL machines. This facilitates porting to different target languages.

BHDL data types are restricted to BOOL, INTn and UINTn. In addition, enumeration types can be used and arrays. These types are contained in a basic BHDL machine, called *BHDL.mch* that must be imported in BHDL machines. In SystemC the types correspond to bool, sc_int<n>, sc_uint<n> and enumeration types and arrays. This choice of data types facilitates portability to other hardware description languages.

Arrays are represented as total functions in BHDL and may not be synthesizable after translation. If the design resulting from translation is to be synthesizable, it may be necessary to modify the design first by refinement. For simulation, however, this is not necessary. In practice, we found that most produced VHDL descriptions were readily synthesisable.

The substitution language of BHDL is a subset of the substitution language of B. BHDL machines are cycle accurate models of hardware which can be represented by a design on register transfer level in hardware description languages like VHDL or SystemC. In fact, register transfer level is not strictly enforced because some BHDL constructs and expressions correspond to the behavioural level. This is convenient for simulation.

For performance analysis, a higher abstraction level (and earlier translation in the design process) would be useful. This work is under way for the transaction level of SystemC.

The substitution language of BHDL comprises assignment “x := E”, simultaneous substitution “S || T”, sequential substitution “S; T”, and conditional substitution “IF B THEN S ELSE T END”. Arrays can only be assigned as a whole, i.e. assignments of the form “x(k):=E” are not possible. The reason is that tracking intermediary signals created in the translation would be complex and error prone, whereas the price of the incurred restriction in practice is very low. In this article we use lambda expressions to represent array values. So an array assignment has the form: “$x := k.(k \in K|E) \cup k.(k \in L|F)$”, where “$dom(x)=K \cup L$”. The right-hand side may contain more than two lambda expressions.

Registers are inferred from variables declared in the VARIABLES clause of a BHDL machine depending on their use. Two sets read and write are calculated for a BHDL machine. Inputs declared in the machine must be contained in read, outputs in write. Variables that are contained in read and write are translated into registers. Input and output variables are translated into corresponding ports, and all remaining variables into wires.

Formally the translation into hardware description languages tr(S) of S is based on the before-after predicate $prd_x(S)$ of a substitution S. This is described in Chapter 7.

Identifiers clock and reset must not be used in BHDL machines. Correspondingly named signals for use with registers are produced by the translation. BHDL machine *DELAY* below shows a B implementation of a buffer that delays its input by 8 clock cycles.

```
MACHINE
    DELAY
SEES
    BHDL
INPUTS
    din
OUTPUTS
    dout
VARIABLES
    neg, buffer
INVARIANT
    neg ∈ UINT4 ∧ buffer ∈ 1..8 → UINT4
INITIALISATION
    buffer := λx.( x ∈ 1..8 | 0 )
OPERATION
    BEGIN
        neg := 15 - buffer(1);
        dout := neg
    END ||
    buffer := λx.( x ∈ 1..7 | buffer( x - 1 )) ∪ λx.( x ∈ {8} | din )
END
```

Variable buffer is initialised to an array of zeros. The translation generates a set of registers with reset for variable buffer. The state transition specified by the substitution is translated into a combinatorial circuit. Inputs and outputs are referred to by their original names. The design resulting from the translation is synthesisable. Thus, so is the original BHDL machine *DELAY* from which the VHDL code was produced by automatic translation. The BHDL machine could be further refined, e.g. by implementing the subtraction on bit level. In general, we prefer to allow as many high level constructs as possible, to allow earlier translation and analysis in the development process. For this reason we are also working on a translator to SystemC transaction level.

7. CONCLUSION

We have presented an overview of the PUSSEE method and the associated software tools developed in the EU IST project PUSSEE [10]. The method offers functional and temporal verification in B and RIL, respectively, at high levels of abstraction, but also allows deriving systems from it that are correct by construction. This achieved by means of the B method using formal refinement. Initial system models can be created in UML and verified in B. By formal refinement these are step by step made more concrete towards implementations in the subsets B0 and BHDL of the B language. Models in B0 can be translated into C and BHDL models into SystemC, for instance. Thus the PUSSEE method can be applied to developments that require co-design and (formal) verification.

REFERENCES

1. J-R. Abrial, *The B Book: Assigning programs to meanings*, Cambridge University Press, 1996.
2. J. Warmer, A. Kleppe, *The Object Constraint Language: Precise Modeling with UML*, Addison-Wesley, 1999.
3. P.Facon, R. Lelau, H.P. Nguyen, *Combining UML with the B formal method for the Specification of database applications*, Research Report, CEDRIC Laboratory, Paris, 1999.
4. ClearSy, *Event B Reference Manual. Version 1.0*, Available at: http://www.atelierb.societe.com/ressources/evt2b/eventb_reference_manual.pdf, 2001.
5. J. Draper et al, *Evaluating the B method on an avionics example*, Proceedings of Data Systems in Aerospace (DASIA) Conference, 1996.
6. C. Snook, L. Tsiopoulos, M. Walden, *A Case Study in Requirement Analysis of Control Systems using UML and B*, Proceedings of International Workshop on Refinement of Critical Systems, Methods, Tools and Developments, 2003.
7. J-R. Abrial, *Event Driven Electronic Circuit Construction*, Available at: http://www.atelierb.societe.com/ressources/articles/cir.pdf, 2001.
8. J-L. Boulanger et al, *Formalization of Digital Circuits Using the B Method*, Proceedings of 8th International Conference on Computer Aided Design, Manufacture and Operation in the Railway and Other Advanced Mass Transit Systems, 2002.
9. W. Ifill et al, *The Use of B to Specify, Design and Verify Hardware*, High Integrity Software, Kluwer Academic Publishers, 43-62, 2001.
10. PUSSEE Project. Available at: http://www.keesda.com/pussee, 2004.
11. KeesDA, *BHDL User Guide. Preliminary Version*, Available at http://www.keesda.com.
12. J-R. Abrial, *Event Model Decomposition*, Available at: http://www.atelierb.societe.com/ resources/articles/dcmp3.pdf, 2001.
13. T. Lecomte, *D4.4.1: Methodological Guidelines: Interface based synthesis/ refinement in B*, IST-2000-30103 PUSSEE, Project Report, 2003.
14. T. Lecomte, J. R. Abrial, F. Badeau, C. Czernecki, D. Sabatier, C. Snook, *Abstract modeling: System level modelling and refinement in B*, Technical Report, Project IST-2000-30103 PUSSEE, 2003.