1 Tool Description

In this section, we discuss briefly the implementation aspects of the system developed. The basic architecture of the BUSpec framework is shown in Fig 1.

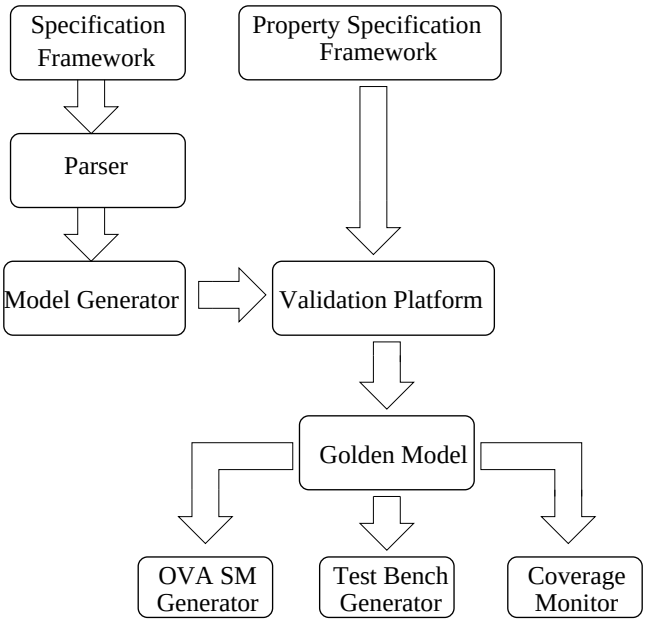
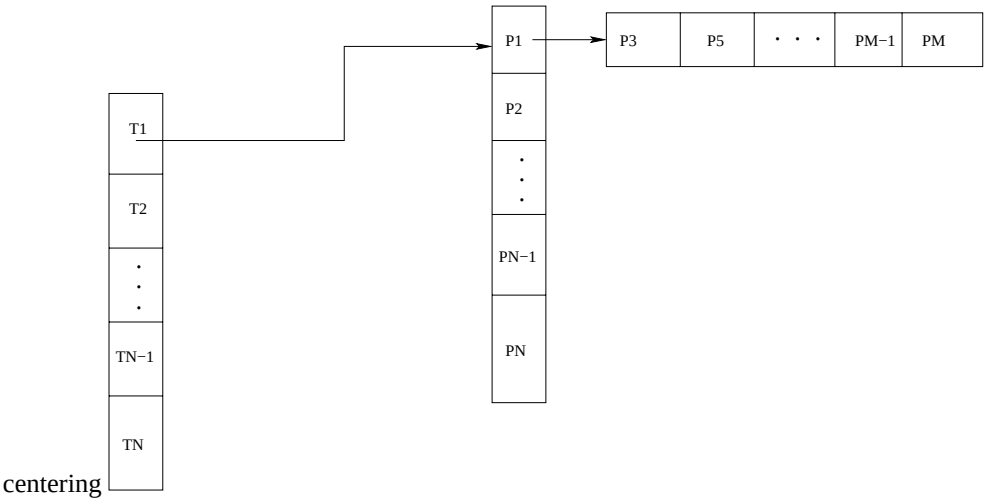


Figure 1: The BUSpec Framework

In the input stage, the designer describes the bus functionality in BUSpec. The description is then parsed by the *parser*. The *parser* produces a linked list of all the transfers supported by the bus. Each node of this list is another linked list containing information about the phases which constitute the transfer. The detailed architecture of this structure is given in Fig 2. Now, according to the requirements specified by



centering

Figure 2: Internal data structure

the user, the *Model Generator* produces system-level state machine or component state machines. Another input to the framework is the set of correctness properties of the system. The designer specifies these properties at different levels of abstraction. The *Validation Framework* takes the set of properties and the models at a particular level of abstraction. It then extracts the *Kripke structure* for the particular level and checks for the correctness at that level against the properties already specified. If the properties are not validated correctly, proper investigation needs to be performed to change either the BUSpec description and/or the correctness properties. If the properties are correctly validated, the *golden models* are generated. Now, depending on the designer/user requirements, OVA state machine and/or Blif description and/or Test benches for the models are generated.

2 The BUSpec Language

Traditionally standard bus protocols such as PCI and AMBA are specified through a well-versed document that primarily describes the types of transfers supported by the protocol, and one or more high level state machines that describe the ways in which these transfers may be sequenced over time. Each transfer is described through a set of timing diagrams. Each timing diagram in turn consists of several phases, such as the bus acquisition phase, the bus usage phase and the bus hand-over phase. A phase may span across several cycles (as in the case of the bus usage phase for burst mode transfers).
By studying several standard bus protocols we have formalized a language for the formal specification of such protocols. This language makes use of the traditional structure of the protocols such as the phases, transfers and high-level state diagrams. Specifically, the proposed BUSpec language enables the designer to write a bus protocol in terms of the following:

1. Specification of the phases that constitute a given transfer type.
2. Specification of each transfer type as a state machine, where each state represents a phase of the transfer.
3. Specification of the system-level transitions that indicates the possible sequences in which the transfers can occur. These transitions are between phases of successive transfers.

2.1 Formal Syntax of BUSpec

In this section the formal syntax of BUSpec is described using Backus-Naur Form (BNF). We then demonstrate the syntax with examples. The conventions used in the syntax are as follows:

- Keywords and punctuations are in **bold** text.
- Syntactic categories are named in non-bold text.
- A vertical bar (|) separates alternatives.
- Square brackets ([ ]) enclose optional items.
- Braces ({ }) enclose items which can be repeated zero or more times.

The formal syntax of BUSpec is given as follows:

System_description ::= **StartFSM** St_mc_desp **EndFSM**
St_mc_desp ::= Trans_desp {Trans_desp} [TrnfTrans]
Trans_desp ::= **StartTransfer** Transfer **EndTransfer**
Transfer ::= Phase [PhaseTrans]
Phase ::= **StartPhase** Phase_desp {Phase_desp} **EndPhase**
Phase_desp ::= Name '{' {Signal_desp} {Predicate_desp} '}'
Signal_desp ::= **signal** '{' signal_val;'}'
signal_val ::= sig_val_desp {, sig_val_desp}
sig_val_desp ::= Name = val
PhaseTrans ::= **StartPhTrans** PhTrns_desp {PhTrns_desp} **EndPhTrans**
PhTrns_desp ::= Name '{' Name Name '}'
TrnfTrans ::= **StartSmTrans** TrnfTrans_desp {TrnfTrans_desp} **EndSmTrans**
TrnfTrans_desp ::= Name '{' Name Name '}'
Predicate_desp ::= **Valid**(Name)|**past**(Name)|**Equal**(Name,Predicate_exp)
Predicate_exp ::= Predicate_desp | Predicate_desp Operator Name
Operator ::= +|-|*
Name ::= [a-zA-Z_]{[a-zA-Z0-9_]}
val ::= 0|1|2|3|4|5|6|7|8|9