Equivalence Checking between SLM and RTL Using Machine Learning Techniques

Jian Hu*, Tun Li*, Sikun Li*
*College of Computer, National University of Defense Technology, Changsha 410073,China

***Abstract*—The growing complexity of modern digital design makes designers shift toward starting design exploration using high-level languages, and generating register transfer level (RTL) design from system level modeling (SLM) using high-level synthesis or manual transformation. Unfortunately, this translation process is very complex and may introduce bugs into the generated design. In this paper, we propose a novel SLM and RTL sequential equivalence checking method. The proposed method bases on Finite state machines with datapath (FSMD) equivalence checking method. The proposed method recognizes the corresponding path-pairs of FSMDs using machine learning (ML) technique from all the paths. And then it compares the corresponding path-pairs by symbolic simulation. The advantage of our method is that it separates the corresponding path-pairs from all the paths and avoids blind comparisons of path-pairs. Our method can deal with greatly different SLM and RTL designs and dramatically reduce the complexity of the path-based FSMD equivalence checking problem. The promising experimental results show the efficiency and effectiveness of the proposed method.**

Keywords*—*Equivalence Checking, FSMD, Machine Learning, System Level Modeling, Formal Verification

I. Introduction

Increasing in the size and complexity of system-on-a-chip (SoC) has made designers shift to start design from system level using high level languages such as C/C++ and SystemC. RTL designs then can be obtained by behavior synthesis tools or manual transformation. Unfortunately, the translation process is complex and error prone. Therefore, maintaining the functional consistency between SLM and RTL designs is an important issue.

Due to the great temporal and structure difference between SLM and RTL designs, equivalence checking between the two levels becomes a tough task. Some techniques have been proposed for equivalence checking between high-level designs, which can be classified into two types of methods: simulation based and formal technique based.

Bombieri et al. [1] proposed an simulation based equivalence checking method, in which mutation [2] and property (assertion) are used as the quality measures of stimuli. The two designs are simulated using the generated stimuli and the results are compared to check the equivalence. Daniel Groeβe et al. [3] accelerated the process of simulation based equivalence checking method by using multi-thread and variable selection techniques. Bombieri et al. used assertions and checked code to check the equivalence, which avoids duplicated effort for RTL verification [4]. Chen et al.[5] analyzed transaction-level modeling (TLM) designs and generated TLM assertions from TLM fault models. Their method proposed refinement rules to generate corresponding RTL assertions from TLM assertions. But the verification quality of all the assertions based methods depends on the quality of the selected assertions. Hu et al. [6] checked the equivalence of SLM and TLM based on coverage directed simulation. Their technique utilized the similarities between SLM and TLM to greatly reuse the verification efforts. Although the simulation-based methods are easy to use and can quickly spot easy-to-find bugs, it can't guarantee the completeness of equivalence checking.

With regard to formal techniques based method, Shobha Vasudevan et al. [7] computed the sequential compare points and used an SAT solver to verify the points. Their method split the equivalence problem with sequential compare points. Zhu et al. [8] proposed a technique to further decompose the verification problem in both the space and the time dimension and used an SMT solver to check the sequential compare points. Takeshi Matsumoto et al. detected and extracted texture difference of two designs, and checked the equivalence of differences by symbolic simulation [9]. There are some works aimed at checking the equivalence between designs at different stages in high-level synthesis (HLS) [10], [11], [12]. Their methods simulate the design symbolically according to the timing and mapping information given by HLS tools. Although formal methods can verify designs completely, it assumes the timing and mapping information to be available and suffers from state-space explosion problem. These timing and mapping information can not be easily obtained in many situations.

To solve the above problems, Chandan Karfa et al. [13] and Lee et al. [14]presented an equivalence checking method by comparing the path-pairs between cut-points in both FSMDs[15]. This technique can deal with path segments in designs, but the number of path segments is large and the comparison is inefficient without mapping information between the compared FSMDs. Carlos Ivan Castro Marquez et al.[16] presented an equivalence checking method based on equivalence of FSMD. They extracted and compared all the deep sequences of both FSMDs to check the equivalence of the two FSMDs. The deep sequence is a state sequence that starts from the initial state to the final state of FSMD without repeated path segments. Since the method considers the complete sequence instead of single state, the efficiency is improved. But the generation and comparison of deep sequences are still time-consuming.

Previous path-based methods have an extensive trial-and-error comparison process due to the lack of mapping information. For the designs without timing and mapping information, we propose a novel equivalence checking method between SLM and RTL, which uses ML to extract the mapping informa-

This work is supported by National Natural Science Foundation of China under grant No.61272335 and 61133007

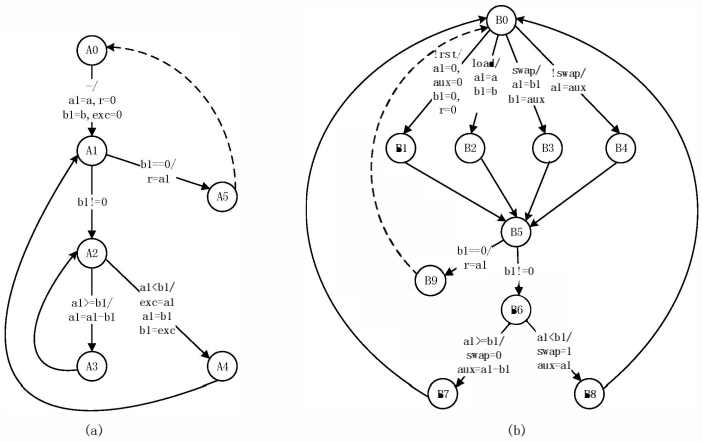

Fig. 1. (a) M_0, the FSMD of the SLM (b) M_1, the FSMD of RTL

tion between these levels to reduce the number of comparisons. In our proposed method, the SLM and RTL designs are modeled using FSMDs. FSMD can strip of the RTL specific constructs and remain the functional aspects, which narrows the gap between SLM and RTL. The equivalence theory used to check the equivalence of the two FSMDs are similar to the methods proposed in [13] and [16] . Our method first generates state sequences from SLM and RTL by simulation. Next, the potential corresponding path-pairs between SLM and RTL designs are recognized using ML technique. Finally, the selected path-pairs are checked to ensure the equivalence of the SLM and RTL designs.

The process of hardware development consists of a series of complex activities. And during each development phase, there is a large amount of data generated and recorded including tests [17]. These tests are used to simulate the SLM and RTL designs, which will generate a mount of data. The ML technique are good at using the generated data to extract information, to gain knowledge that can be used to guide the verification process. ML technique have been applied in many hardware areas, such as regression testing, bug localization, hardware validation, compiler design . It is a novel idea to use ML technique in equivalence checking area, which can determine the corresponding path-pairs and decrease the number of comparisons. Excluding the path-pairs one by one in previous methods is more expensive than simply proving the corresponding path-pairs in our method.

The contributions of this work are the following:

1) By using ML technique, the proposed method can check equivalence of designs without mapping information. ML can find the relevant information from the simulation data.

2) ML technique can automatically generate the corresponding path-pairs. The proposed method can directly compare the corresponding one which will greatly reduce the complexity for path-pairs comparison.

3) Reusing the generated data during the development of hardware design avoids generating the explicit FSMD. The only cost is processing the generated data. The verification efficiency is further improved.

II. Motivating Example

The equivalence problem of FSMDs is undecidable and not even partially decidable [18]. At present, equivalence checking of FSMDs by path segments comparison is commonly used. However, all the existing methods have to enumerate and compare all the path-pairs between the two FSMDs blindly. It can be observed that most of the comparisons will fail for wrongly enumerated path-pairs. If verification engineers know which is the corresponding path-pair to be compared before carrying out the enumeration and comparison, it will reduce most of the verification efforts.

For example, in Fig.1(a) the path A_0A_1 of FSMD M_0 will be compared to B_0B_1, B_0B_2, B_0B_3 and B_0B_4 of FSMD M_1 in [13]. However, if the two designs can be simulated for a while, the true corresponding path-pair (A_0A_1, B_0B_2) can be easily found. Therefore, the number of comparisons will be decreased from 4 to 1. Besides that, the RTL model is greatly different with the SLM model. Except the common variables, the temporal and structure differences are extreme large as shown in Fig. 2(left is the SLM code in C and right is the RTL code in Verilog). It is very hard to automatically find the cutpoints to decompose the verification problem without mapping information.

We take greatest common divisor (GCD)(Fig.1) as an example to compare the efficiency of our method with other path-based methods presented in [16] and[13]. With the method in [16], it will generate 3 SLM deep paths and 76 RTL deep paths, and there will be 99 comparisons while only 3 of them are valid. While with the method in [13], it will generate 3 cutpoints and 7 path segments, and there will be 20 comparisons with only 7 of them are valid. However, using the ML results and the generated simulation data, our method can obtain the mapping information between the two FSMDs. The 4 corresponding path-pairs can be determined and only 4 comparisons are really carried out in the example. The observations and the preliminary results motivate us to propose the novel equivalence checking method using ML technique.

III. FSMD and Equivalence of FSMD

A. Finite State Machines with Data Paths (FSMD)

The FSMD [15] can be used to model the digital design and computer program. In this paper, we extend the definition of traditional FSMD with a addition of a reset state (start state) to encode the design under verification. The FSMD is defined as an ordered tuple $\langle Q, q_0, I, O, V, f, h\rangle$, where

1) Q=$\{q_0, q_1, q_2, ..., q_n\}$ is the finite set of states,

2) q_0 is the reset state,

3) I is the finite set of primary inputs,

4) O is the finite set of primary outputs,

5) V is the finite set of storage variables,

6) $f : Q \times S \rightarrow Q$ is the state transition function, where S is the set of status expression including predicates over $I \cup V$.

7) $h : Q \times S \rightarrow U$ is the update function of outputs and storage variables, where U is a set of outputs or storage variable assignment statements, U=$\{x = e \mid x \in O \cup V$,e is arithmetic expression over $I \cup V\}$

The FSMD is deterministic, thus for any state q, if status expression s_1=s_2, then $f(q, s_1)$=$f(q, s_2)$ and

$h(q, s_1)$=$h(q, s_2)$. For the FSMDs shown in Fig.1 the label on each transition edge is of the form s/h, $s \in S$, for example the label "$swap/a_1$=b_1, $b_1 = aux$" on the transition from B_0 to B_3 of FSMD M_1. In this paper, to compare our method with method in [16], we make a copy of the reset state to be final state, such as state A5 and B9. Therefore, all the incoming edges of the reset state are all altered to point to the final state. And the final state will loop to reset state as depicted in dashed line (Fig.1).

B. Equivalence of FSMD

Let the behavior of SLM be represented by FSMD M_0=$\langle Q_0, q_{00}, I_0, O_0, V_0, f_0, h_0\rangle$ and the behavior of corresponding RTL be represented by FSMD M_1=$\langle Q_1, q_{10}, I_1, O_1, V_1, f_1, h_1\rangle$, respectively. We define the equivalence of FSMD as follows:

Definition 1: A finite path p from q_i to q_j, where q_i, $q_j \in Q$, is a finite transition sequence of states of the form $\langle q_i = q_1 \overrightarrow{c_1} q_2 \overrightarrow{c_2} ... \overrightarrow{c_{n-1}} q_n = q_j\rangle$ such that $\forall l, 1 \leq l \leq n-1, \exists \overrightarrow{c_l} \in S$ such that $f(q_l, \overrightarrow{c_l}) = q_{l+1}$.

Definition 2: The output of a path p is defined as an ordered tuple $\langle e_j\rangle$ of algebraic expressions over $I \cup V$ such that the expression e_j represents the value of the variable v_j after execution of the assignments along the path in terms of the initial data state of the path.

Definition 3: (Path coverage of a FSMD): A finite paths set $P = \{p_0, p_1, p_2, ..., p_k\}$ is said to be a path cover of a FSMD M, if any path of M from initial state to final state can be a concatenation of paths in P.

Definition 4: Let I_{SLM}, I_{RTL}, O_{SLM}, O_{RTL}, V_{SLM}, V_{RTL} represent the set of the primary inputs, primary outputs and storage variables in SLM and RTL, respectively. The common primary inputs, common primary outputs and common storage variables of SLM and RTL are defined as $I_\cap = I_{SLM} \cap I_{RTL}$, $O_\cap = O_{SLM} \cap O_{RTL}$, and $V_\cap = V_{SLM} \cap V_{RTL}$.

From a general point of view, it could be stated that the sets of primary inputs, primary outputs and storage variables of RTL will contain those of SLM, symbolically $I_{SLM} \subseteq I_{RTL}$, $O_{SLM} \subseteq O_{RTL}$, and $V_{SLM} \subseteq V_{RTL}$. For equivalence checking, our method only considers the common variables in $I_\cap$, $O_\cap$, and $V_\cap$ and neglects the variables outside these sets.

Definition 5: A path p_0 of FSMD M_0 and a path p_1 of FSMD M_1 are said to be equivalent, symbolically $p_0 \equiv p_1$, if for any primary input $i \in I_\cap$, the outputs of path p_0, denoted by O_0, and the outputs of path p_1, denoted by O_1, and for each v_{j0} defined by $e_{j0} \in O_0$ and v_{j1} defined by $e_{j1} \in O_1$, $v_{j0} \equiv v_{j1}$ must be hold, where $v_j \in O_\cap \cup V_\cap$ and subscript 0 and 1 represent the variable appeared in M_0 and M_1, respectively.

Definition 6: Two designs are equivalent if their corresponding FSMD M_0 and M_1 are equivalent, symbolically $M_0 \equiv M_1$.

Definition 7: (Equivalence of FSMD): An FSMD M_0 is said to be equivalent with FSMD M_1, if, for any path u_0 of M_0 from initial state to final state, there exists a path u_1 of M_1 from initial state to final state, such that $u_0 \equiv u_1$, and vice versa.

Definition 8: (EVP, CP, UCP): The path composed of the adjacent states from the generated state sequence with the same state value is defined as equal value path (EVP). The path composed of adjacent paths in EVP is defined as connected path (CP). The set of paths composed of unrepeated paths in CP is defined as unrepeated connected path (UCP) set.

Lemma 1: If tests can cover all the paths segments between states of FSMD M_0, after simulation the generated state sequences grouped to EVP set P = $\{p_0, p_1, p_2, ..., p_t\}$ and CP set Q=$\{p_0p_1, p_1p_2, p_2p_3, ..., p_{t-1}p_t\}$, then $P \cup Q$ is the path cover of M_0.

Theorem 1: Let $P_0 = \{p_{00}, p_{01}, p_{02}, ..., p_{0k}\}$ be the path cover of FSMD M_0. If there exists a finite path set $P_1 = \{p_{10}, p_{11}, p_{12}, ...p_{1k}\}$ of FSMD M_1 and $p_{0i} \equiv p_{1i}$, where $0 \leq i \leq k$, and vice versa, then M_0 is equivalent with M_1.

IV. Equivalence Checking Algorithm

Theorem 1 suggests a verification method for checking equivalence of two FSMDs which consists of the following steps:

- Constructing training and testing set: Reuse the generated tests during the development of hardware and simulate both models to prepare original state sequences of both SLM and RTL.
- Generation of mapping information: After the construction of training set and testing set, ML is used to generate the mapping information between the states in SLM and RTL.
- Constructing corresponding path-pairs: The EVP set, CP set and UCP set are constructed according to the classification of ML and the corresponding paths in UCP set between SLM and RTL are selected as the corresponding path-pairs between the two FSMDs.
- Checking the corresponding path-pairs in SLM and RTL using symbolic simulation and an SMT solver.

The details of the algorithm are described in the following subsections.

A. Constructing Training Set and Testing Set

The training and testing set are the state sequences output from SLM and RTL designs using simulation in our method. The FSMD state sequences of SLM and RTL are generated by simulation, which avoids the complex process for generating FSMD explicitly.

1) Reusing the generated tests: As mentioned above, the development of hardware generates a large amount of data that can be reused in verification process. The used tests should cover all the path segments of the FSMD, which guarantees the completeness of verification.Previously existing equivalence checking methods are inefficient use of collected data. Our method reuses the generated tests to simulate the SLM and RTL models.

2) Outputing state sequences: In our method, the output statements are automatically inserted into the source code of SLM and RTL by our tool to obtain the state sequences. When the simulation finished, the state name and its value will be output, such as "A0:a1=1,b1=2,r=3", where A is the state name followed by the state variable with its value. Fig. 2 illustrates the working process. Here the output statements are inserted by our tool, such as "//output A0".

SLM code snippet

```
int gcd(int a, int b)
{
    //Output A0
    int a,b,aux,r;
    a1=a;
    b1=b;
    r=0;
    aux=0;
    //Output A1
    while (b1 != 0)
    {
     //Output A2
     while (a1 >= b1)
     {
      //Output A3
      a1 = a1 – b1;
     }
     //Output A4
     aux = a1;
     a1 = b1;
     b1 = aux;
    }
    r=a1;
    //Output A5
    return r;
}
```

RTL code snippet

```
always @(posedge clk or negedge rst)
  //Output B0
  if (!rst)
  //Output B1
   begin
    a1=0;
    b1=0;
    r=0;
    aux=0;
   end
  else if (load)
     //Output B2
     begin
      a1=in1;
      b1=in2;
     end
  else if (swap)
     //Output B3
     begin
      a1=b1;
      b1=aux;
     end
  else
     //Output B4
     a1=aux;
endcase
```

```
always @(a1 or b1)
  //Output B5
  if (b1!=0)
     begin
     //Output B6
     out=0;
     if (a1>=b1)
        begin
         //Output B7
         swap=0;
         aux=a1-b1;
         end
     else
        begin
        //Output B8
        swap=1;
        aux=a1;
        end
     end
  else
       //Output B9
       r=a1;
endcase
```

Fig. 2. GCD snippet for state generation

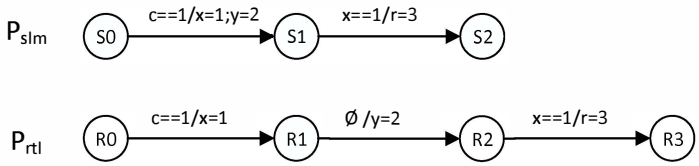

Fig. 3. Construction of CP

B. Generation of Mapping Information Using ML

Designs without mapping information are hard to be verified. And the abundance of generated states and paths makes verification low efficient. But ML is a promising way to learn from these volumes of generated data and provide the mapping information between greatly different designs, in order to guide the verification process. ML is applied in many areas where user correctly transforms the problem into a form recognized by an existing ML algorithm or tool. ML is a scientific discipline that can be subdivided into supervised and unsupervised learning [19]. Supervised learning is the machine learning task of inferring a learning model from labeled training data, which can be used for mapping new data set. In our method, supervised learning is used to map the states in RTL to the corresponding classes of states in SLM, which can direct our verification process. The process consists of the following steps.

1) Feature extraction and Label determination: The features for learning are extracted based on the generated state sequence from SLM and RTL models. The number of input, output and storage variables in RTL is larger than that in SLM, hence, only the common primary inputs $I_{\cap}$, primary outputs $O_{\cap}$ and storage variables $V_{\cap}$ are considered. All the variables not in these set will be neglected by our solution. In our equivalence checking framework, state variables $O_{\cap}$ and $V_{\cap}$ are taken as the classification feature for ML, such as $A_i = \langle a, b, o\rangle$,where A_i is a state name, a,$b \in V_{\cap}$ and $o \in O_{\cap}$. These features are represented as vectors for the input of ML tool. Each element in the training set should be assigned a label after feature extraction. The states with the same feature value are automatically assigned the same label. At the end of this step, a tuple set $S_s = \{\langle A_0 : l_0\rangle, \langle A_1 : l_1\rangle, \langle A_2 : l_2\rangle, ..., \langle A_t : l_k\rangle\}$ can be obtained, where A_i is the state name and l_i is the label,$1 \leq l_k \leq t$.

2) Machine learning processing: Many ML tools and libraries have enabled us to access the generated data to extract the inner mapping information. It is performed on training set element x_i of an object i, with $i = 1, 2...N$, together with their classification(label) y_i. Each variable x_i represents a vector of n features $xi = \{x_{i0}, x_{i1}...x_{in}\}$ and each label y_i indicates its class. In our case, the generated states during simulation make this problem an ideal target for applying ML technique to extract the mapping information to guide the verification.After ML learns the generated SLM states, the classifier induces a prediction model, which will predict the classes of the RTL states. In our algorithm, libsvm is used as our ML tool to generate the mapping information.

C. Constructing the corresponding path-pairs

The result of ML is the states with their corresponding labels. What really needed in our equivalence checking method is the path between states not a single state. Hence, the training set and prediction result are used to construct the corresponding path-pairs between SLM and RTL. The adjacent states with the same class label are grouped to construct the EVP set and the labels of the states become the labels of the paths, such as SLM EVP set E_0=$\{p_{00}, p_{01}, p_{02}, ..., p_{0m}\}$, RTL EVP set E_1=$\{p_{10}, p_{11}, p_{12}, ..., p_{1n}\}$, where p_{0i} = $\{A_c$,$A_{c+1}, ..., A_d\}$, and p_{1j}=$\{B_q, B_{q+1}, ..., B_r\}$, where A_c, A_d and B_q, B_r are the state, $0 \leq i \leq m$, $0 \leq j \leq n$. The adjacent paths in EVP (E_0) of SLM are connected to obtain CP set C_0 = $\{p_{00}p_{01}, p_{01}p_{02}, ..., p_{0m-1}p_{0m}\}$ and the paths of corresponding classes in RTL are connected to obtain CP set C_1 = $\{p_{10}p_{11}...p_{1i}, p_{1i+1}p_{1i+2}...p_{1i+j}, ..., p_{1n-k}...p_{1n}\}$. Because RTL is more specific than SLM, there exist some states with the value that do not appear in SLM. The corresponding paths in CP set of RTL are not the adjacent two paths in EVP but several adjacent paths in EVP.

As shown in Figure 3, the EVPs of SLM and RTL are $E_0 = \{S_0, S_1, S_2\}$ and $E_1 = \{R_0, R_1, R_2, R_3\}$. All the paths in EVP of RTL between the corresponding classes are connected. Since the corresponding classes are (S_0, R_0),(S_1, R_2),(S_2, R_3), in our method, we just connect the paths between S_0 and S_1 in E_0 and between R_0 and R_2 in E_1 . Therefore, the CPs of SLM and RTL are $C_0 = \{S_0S_1, S_1S_2\}$ and $C_1 = \{R_0R_1R_2, R_2R_3\}$. Although the state R_1 has not a corresponding state in SLM, it can also be included in one CP and has a corresponding path. Since there are repeated paths in CP and the repeated paths need not be verified twice, hence, the repeated paths are deleted to obtain UCP set, and the results will be $U_0 = \{p_{00}p_{01}, ..., p_{0j-1}p_{0j}\}$, and U_1=$\{p_{10}p_{11}...p_{1k}, ..., p_{1t-h}...p_{1t}\}$, where $0 \leq j \leq m$,$0 \leq k, t \leq n$. The corresponding paths in the UCPs of SLM and RTL are the corresponding path-pairs between SLM and RTL. The state value in paths of EVP are the same, which means no assignment statements operate on the state variables. Therefore, the paths in EVPs of SLM and RTL need not be verified.

D. Comparison of the Corresponding Path-pairs

The classified path-pairs will be verified using word-level symbolic simulation and an SMT solver. First, the static single assignment form (SSA) [20] of corresponding path-pairs in SLM and RTL will be generated. Second, the SSA code of each corresponding path-pairs are conjuncted. Third, for the common state variables of the SSA expression $r_{slm} \in V_{\cap} \cup O_{\cap}$, $r_{rtl} \in V_{\cap} \cup O_{\cap}$, the formula $(r_{slm} \neq r_{rtl})$ is conjuncted to the SSA expression. Fourth, for all the corresponding inputs of the SSA expression $q_{slm} \in I_{\cap}$ and $q_{rtl} \in I_{\cap}$, the formula $\bigwedge(q_{slm} = q_{rtl})$ is conjuncted to the SSA expression. Fifth, the obtained SSA expression is converted to SMT format. Finally, the generated SMT expression is fed into an SMT solver, such as Z3 [22], to prove the equivalence.

V. Example

In this section, the working of our method for equivalence checking between SLM and its corresponding RTL is illustrated with the GCD example. The FSMDs of SLM and RTL are depicted in Fig.1 (a) and (b), respectively. Common variable $V_{\cap} = a, b$ and $O_{\cap} = r$ are selected as the state variables (feature).

1) First, the output statements of state are automatically inserted in the SLM and RTL code by our tool. And the generated tests during the development of hardware design are reused to simulate SLM and RTL designs. After simulation the state sequences with its state variable values of SLM and RTL are output as shown below(A_i and B_i denote the states of FSMDs followed by their state value):

Training Set(SLM states):$A_0(0,0,0), A_1(4,3,0), A_2(4,3,0)$
$, A_3(1,3,0), A_2(1,3,0), A_4(3,1,0), A_1(3,1,0), A_2(3,1,0)$
$, A_3(2,1,0), A_2(2,1,0), A_3(1,1,0), A_2(1,1,0), A_3(0,1,0)$
$, A_2(0,1,0), A_4(1,0,0), A_1(1,0,0), A_5(1,0,1)$

Testing Set(RTL states):$B_0(0,0,0), B_2(4,3,0), B_5(4,3,0)$
$, B_6(4,3,0), B_7(4,3,0), B_0(4,3,0), B_4(1,3,0), B_5(1,3,0)$
$, B_6(1,3,0), B_8(1,3,0), B_0(1,3,0), B_3(3,1,0), B_5(3,1,0)$
$, B_6(3,1,0), B_7(3,1,0), B_0(3,1,0), B_4(2,1,0), B_5(2,1,0)$
$, B_6(2,1,0), B_7(2,1,0), B_0(2,1,0), B_4(1,1,0), B_5(1,1,0)$
$, B_6(1,1,0), B_7(1,1,0), B_0(1,1,0), B_4(0,1,0), B_5(0,1,0)$
$, B_6(0,1,0), B_8(0,1,0), B_0(0,1,0), B_3(1,0,0), B_5(1,0,0)$
$, B_9(1,0,1)$

2) Seconde, the states of SLM are assigned labels according to their state values and the states of RTL are assigned labels 0 for initialization. ML(libsvm) tool learns the states of SLM to generate a predicting model, in order to classify the states in RTL, to the corresponding classes of SLM. Table I shows the classification result. The 1^{st} column is the labels of the classified states. The 2^{nd} and 3^{rd} columns represent the states of SLM and RTL belonging to the corresponding classified classes. The last column is the state variables value of the corresponding states in the class. Taking the third row of table I as an illustration, the states A_1,A_2 of SLM and B_2,B_5,B_6,B_7,B_0 of RTL belonging to class 1 are the corresponding states.

3) Third, the EVP and UCP set are constructed using the labeled training set and the generated predicted result. The adjacent states in the same class of training set and predicted result are extracted to constitute the EVP set of SLM and RTL. As shown in table I, the states in the same class construct the path in EVP set.

TABLE I. Classification of State Sequences Using ML

Classification	SLM	RTL	(a,b,r)
0	$A_{\bullet}$	$B_{\bullet}$	(0,0,0)
1	A_1,A_2	B_2,B_5,B_6,B_7,$B_{\bullet}$	(4,3,0)
2	A_3,A_2	B_4,B_5,B_6,B_8,$B_{\bullet}$	(1,3,0)
3	A_4,A_1,A_2	B_3,B_5,B_6,B_7,$B_{\bullet}$	(3,1,0)
4	A_3,A_2	B_4,B_5,B_6,B_7,$B_{\bullet}$	(2,1,0)
5	A_3,A_2	B_4,B_5,B_6,B_7,$B_{\bullet}$	(1,1,0)
6	A_3,A_2	B_4,B_5,B_6,B_8,$B_{\bullet}$	(0,1,0)
7	A_4,A_1	B_3,B_5	(1,0,0)
8	A_5	$B_{\bullet}$	(1,0,1)

EVP(SLM)=$\{A_0,\ A_1A_2,\ A_3A_2,\ A_3A_2,\ ,...,\ A_4A_1,\ A_5\}$
EVP(RTL)=$\{B_0,\ B_2B_5B_6B_7B_0,,...,\ B_3B_5,\ B_9\}$

The CP sets are obtained by connecting the adjacent paths in the EVP set of SLM and RTL. Since there may exist some repeated paths in CP set that need not be verified twice, the repeated paths are deleted to obtain UCP set.
UCP(SLM)=$\{A_0A_1A_2,\ A_1A_2A_3A_2,...,A_4A_1A_5\}$
UCP(RTL)=$\{B_0B_2B_5B_6B_7B_0,\ B_2B_5B_6B_7B_0B_4B_5B_6B_8$
$B_0,...,B_3B_5B_9\}$

4) Finally, the corresponding path-pairs in UCP sets of SLM and RTL are proved using symbolic simulation and an SMT solver, such as $A_0A_1A_2 \equiv B_0B_2B_5B_6B_7B_0$. it can be proved that all the corresponding path-pairs are equivalent. According to theorem 1, the FSMD of SLM is equivalent to the FSMD of RTL.

VI. Experimental results

The equivalence checking framework has been implemented based on the previously described algorithm. We implement our symbolic simulator based on the codes of Pycparser[21]. We use Z3[22] as our SMT solver and libsvm[23] as our ML tool for its high accuracy classification. The experimental results are obtained on a 2.3 GHz Intel i5 workstation with 4G RAM. We carried out our experiments on different kinds of benchmarks. Some of the benchmarks are from spark publicly available example suite, such as IDCT, MINSORT and SPARK_CONTINUE . Some are from high level synthesis benchmarks[24], such as GCD and LRU. The rest are from SystemC-2.3.0 source code like Pipe and FFT. The benchmarks not only include control intensive design, such as GCD, MINSORT and LRU, but also include data intensive design, such as IDCT and FFT. MINSORT and LRU have been translated to two versions. A real reduced instruction set computer cpu (RSIC_CPU) module is also verified to prove the practical applicability and scalability of our method. System level descriptions of these benchmarks are implemented in C and the corresponding RTL models are in Verilog.

Table II presents the results of our experiments. The 2^{nd} column is the number of code lines of SLM design. The 3^{rd} and the 4^{th} columns are the number of paths in EVP and UCP respectively. The 5^{th}, 6^{th} and 7^{th} columns represent the simulation time, ML processing time and path verification time respectively. The 8^{th} column is the total validation time of our algorithm. To show the efficiency of our proposed method, we compared the total verification time with the state of the art method in [16]. The time it costs is given in the 9^{th} column. The last column shows the validation result of each case. "Yes" means the designs are equivalent,while "No" means not equivalent. Comparing with the method in [16],

TABLE II. EXPERIMENTAL RESULTS

Benchmarks	C #line	EVP	UCP	Our method(ms)				Method in [16](ms)	Equivalent
				Simulation	Machine Learning	SMT Verification	Total		
GCD	20	7	4	15	46	77	138	3036	Yes
SPARK_CONTINUE	21	9	4	12	38	124	174	273	Yes
MINSORT	33	10	4	32	48	154	234	286	No
MINSORT2	33	10	4	32	48	185	265	347	Yes
IDCT	56	10	9	42	42	280	364	136	Yes
PIPE	34	16	5	26	46	217	289	228	Yes
LRU	41	19	4	138	108	169	415	683	Yes
LRU2	41	19	4	138	108	169	415	683	Yes
FFT	122	6	2	58	46	92	196	3255	Yes
RISC_CPU	256	25	10	608	92	337	1037	10881	Yes

our method is more efficient except the designs IDCT and PIPE. Since the two designs are very simple with only a few branches, loops and paths, comparing all the path-pairs is more efficient than finding and comparing the corresponding ones. The rest designs are more complex with many branches, loops and paths. It costs much time to compare all the path-pairs between SLM and RTL models without mapping information in [16]. Since our method utilizes the ML technique to recognize corresponding path-pairs, we can check the complex designs more efficiently. To prove the practical applicability and scalability of our method, we apply our method on a real RSIC_CPU module that supports 11 different instructions. The experimental result shows that our method can check the complex designs efficiently. And a fault (change $a < min$ to $a \leq min$) is manually inserted in the MINSORT design of RTL , which makes it not equivalent with its SLM design. Our method can detect the fault with less time cost.

ML technique as a statistical solution may classify the states of RTL into wrong classes in our algorithm, which makes the results of equivalence checking wrong. But the possibility of wrong classification is low. Because we only need the corresponding states of RTL classified correctly in our algorithm and ML has learned the values of the corresponding RTL states in the learning phase. It's less likely to classify the corresponding RTL states that has been learned by ML into wrong classes. Hence, our approach can guarantee the correctness of equivalence checking in most situations without mapping information and these situations can hardly be handled by other formal methods.

VII. CONCLUSION

This paper proposes a sequential equivalence checking method between SLM and RTL using ML technique. There are two factors which make the proposed method be more efficient. First, ML directs verification process to check the corresponding path-pairs in SLM and RTL. Second, the tests are reused to simulate the designs to generate state sequences without generating FSMD explicitly. The overall vision of our method is to analyze the existing paths to provide feedback that can guide the verification process in order to improve verification efficiency. The experimental results confirm the efficiency and effectiveness of our method.

REFERENCES

[1] Nicola Bombieri, Franco Fummi, and Graziano Pravadelli. RTL-TLM equivalence checking based on simulation. In EWDTS'08, p.214-217, 2008.

[2] N. Bombieri, F. Fummi, and G. Pravadelli. A mutation model for the systemc tlm 2.0 communication interfaces. In DATE'08, p.396-401, 2008.

[3] Daniel Groβe and Markus Groβ and Ulrich Khne and Rolf Drechsler Drechsler. Simulation-based equivalence checking between systemc models at different levels of abstraction. In GLSVLSI'13, p.223-228, 2011.

[4] N.Bombieri, F.Fummi, and G.Pravadelli. Incremental abv for functional validation of tl-to-rtl design refinement. In DATE'07, p.882-887, 2007.

[5] Mingsong Chen and Prabhat Mishra. Assertion-based functional consistency checking between tlm and rtl models. In VLSI Design'12, p.320-325, 2012.

[6] Jian Hu, Tun Li, and Sikun Li. Equivalence checking between slm and tlm using coverage directed simulation. In CADCG'13, p.101-106, 2013.

[7] S. Vasudevan, V. Viswanath, J. A. Abraham, and J. Tu. Sequential equivalence checking between system level and rtl descriptions. IEEE DAES, vol.12, no.4, p.377-396, 2006.

[8] Dan Zhu, Tun Li, Yang Guo, and Si kun Li. 2d decomposition sequential equivalence checking of system level and rtl descriptions. In ISQED'08, p.637-642, 2008.

[9] Takeshi Matsumoto, Tasuku Nishihara, Yoshihisa Kojima, and Masahiro Fujita. Equivalence checking of high-level designs based on symbolic simulation. In ICCCAS'09, p.1129-1133, 2009.

[10] Tun Li, Yang Guo, Wanwei Liu, and Chiyuan Ma.Efficient translation validation of high-level synthesis. In ISQED'13, p.516-523, 2013.

[11] Kecheng Hao, Fei Xie, Sandip Ray, and Jin Yang. Optimizing equivalence checking for behavioral synthesis. In DATE'10, p.1500-1505, 2010.

[12] Zhenkun Yang, Kecheng Hao, Ray S., and Fei Xie. Handling design and implementation optimizations in equivalence checking for behavioral synthesis. In DAC'13, p.1-6, 2013.

[13] ChandanKarfa, DipankarSarkar, ChittaranjanMandal, and Pramod Kumar. An equivalence-checking method for scheduling verification in high-level synthesis. IEEE TCAD, vol.27, no.3, p.556-569, 2008.

[14] C.Lee, C.Shih, J.Jou J.Huang. Equivalence checking of scheduling with speculative code transformations in high-level synthesis. In APS-DAC, p.497-502, 2011.

[15] D. Gajski, N. Dutt, A. C.-H. Wu, and S. Y.-L. Lin. High-level synthesis: Introduction to chip and system design. In MA: Kluwer, 1992.

[16] Carlos Ivan Castro Marquez, Marius Strum, and Wang Jiang Chau. Formal equivalence checking between high-level and rtl hardware designs. In LATW'13, p.1-6, 2013.

[17] Farkash and Monica C. Using data mining to increase controllability and observability in functional verification. In UT Electronic Theses and Dissertations, 2014.

[18] Z. Manna. Mathematical theory of computation. In New York: McGrawHill, 1974.

[19] C.M. Bishop. Pattern Recognition and Machine Learning. Springer, Berlin, 2008.

[20] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. ACM TOPLAS, vol.13, no.4, p.451-490, 1991.

[21] http://pypi.python.org/pypi/pycparser

[22] http://z3.codeplex.com/

[23] http://www.csie.ntu.edu.tw/~cjlin/

[24] http://computing.ece.vt.edu/~mhsiao/hlsyn.html