Formal Verification of SoC Based Embedded Design using Context Based Assertions

1CHANDRASEKARAN SUBRAMANIAM, 2PRASANNA VERTIVEL, 3SRINATH BADRI, 3SRIRAM BADRI
1Rajalakshmi Engineering College, 2Easwari Engineering College, 3Sri Venkateswara College of Engineering
Anna University, Chennai
INDIA
chandrasekaran_s@msn.com, prasannavetrivel1990@gmail.com, srinath.badri@live.com, b-sriram@hotmail.com

Abstract: - The objective of the work is to propose a formal verification logic for mixed mode hardware using the internal and external settings of the circuits towards context based assertions. The mixed signal hardware needed for an artificial arm using a programmable system on chip (pSoC) micro controller is verified and developed. The earlier CTL and LTL use the path quantifiers and logical operators to formally verify the behaviour of the system in all possible future behaviour respectively. An extension to the existing layered approach of the verification mechanism, a context layer may be added to fulfil the mode requirement verification of the hardware circuits along with the value driven timely behaviour verification. The MAD (Mixed Analog Digital) circuits that are realized using system on chip (SoC) device are verified using the proposed Hybrid Context Mode (HCM) Logic and verified using the Model Verifier, Alloy. The proposed logic simplifies the model checking and property checking strategies of the individual MAD circuits with the context based assertion technique.

Key-Words: - Hardware Verification, Hybrid circuit, Context based, Assertion technique, Model Verifier, System on chip.

1 Introduction

Embedded systems have computing engine as one of the components that are to be modelled and designed with built-in error avoidance techniques. The performance characteristics of real time embedded systems will be an integrated function of all the components, working environment and the interfaces with which they connect to the real world. The criticality and timeliness of any embedded system behaviour emanates from the basic stem of the modelling objectives considering the electrical environment encompassing the temperature, vibration and humidity factors on which the system performance depends. Model checking is used to find actual execution paths and comes up with a precise figure for end-to-end timing [1]. In the earlier computational model for embedded systems based on Petri nets which is called PRES, an approach based on the co verification of both hardware and software of an embedded system is represented. The Linear temporal logic (LTL) has a linear, non-branching time model with temporal modal operators operating on propositional variables as atomic propositions connected by the logical connectives negation, and, or and implication. Computation tree logic (CTL) is branching time logic, meaning that its model of time is a tree-like structure in which the future is not determined; there are different paths in the future, any one of which might be an actual path that is realized. Property specification using CTL helped in development of formal methods in system design, especially in the area of digital circuit design. In order to have a rigorous specification of analog circuit properties to be formally checked on a discrete state space representation, an analog extension to CTL has been introduced. In order to specify explicit timed behaviour for CTL-A operations, CTL-AT was introduced as an extension, offering to constrain the validity of CTL-formulas to a time interval [4]. Symbolic model checking is used to check the correctness of a system by specifying properties in CTL and verifying if they are satisfied, whereas the formulae in CTL consist of atomic propositions, Boolean connectors and temporal operators. The temporal operators consist of forward time operators which are preceded by a path quantifier. Thus these formulae may be used to represent computational paths over labelled state transition graph. Therefore
the CTL approach causes a very big state explosion problem [2]. In addition to CTL operations, sugar was extended by sequential extended regular expressions (SEREs) to reason about more complex state transitions within Kripke structures. PSL expressions are composed from three layers namely Boolean layer, a temporal layer and a verification layer. The Boolean layer forms the atomic propositions using simple Boolean expressions interfaced with variables of the underlying HDL. For environmental constraints, this may include amplitude of noise, initial conditions of the circuit current and voltages. In the model checking, C code model checkers are used for the verification of ANSI C code whereas the C source code that is found in microcontrollers is not compatible with ANSI C. The model checking in C is still a promising approach to the verification of embedded software [3]. In an earlier model verification methodology automated theorem proving for noise and process variation in analog design was used. This approach was based on modelling the noise based on SDEs, finding a closed form solution and then integrating the device variation [5].

The paper is organized as follows: Section 2 proposes a logic that incorporates the context of operation of an embedded system using hybrid components in system on chip platform. Section 3 explores the model analysis by investigating the effects of temperature and disturbances on the behaviour of the system using Alloy analyzer with its code for bringing the relevant states and transitions in the analysis results. The section 4 proposes an algorithm that identifies the reachable states and permissible transitions in the behaviour of an artificial embedded arm controller with three actuators fitted to perform three degrees of freedom. Section 5 focuses on the assertion technique that can be used to verify the reachability and timeliness properties of the above mentioned prototype through the architecture specific C cross compiler. Section 6 concludes the research work by bringing out the limitations and issues in the design of a SoC based embedded system based on the operational context and its electrical environment.

2 Hybrid Context Mode (HCM) Logic

The embedded system behaviour is to be modelled not only based on its functional states but also based on its context of operation using atomic propositions. The HCM logic allows path quantifiers and temporal operators as in CTL and LTL in addition to mode and context specifiers. For example, there are power saving mode, sleep mode, and normal mode required for the performance enhancement of the system. The system is to be modelled in so as to behave based on the context of electrical environment. The electrical context may be due to not permissible temperature conditions and unwanted noisy situation. The impact of all the specifications of the working environment on the behaviour should be considered in the model checking of any real time embedded system. The model checking based on the context of operation of embedded system includes the following specifications:

1. Mode Specification: Different types of modes, Mode1, Mode2 and Mode3 representing power safe mode, common mode and sleep mode.
   a) The disturbance context can be selected when there is electrical noise due to proximity components affecting the signal to noise ratio.
   b) The temperature can be the surrounding or the environment, or the operating temperature.
3. Fuzzy values: If any operator involved in the computation, a fuzzy value is considered with Low, Very Low, High and Very High values
4. Permissibility Constraints: The states may be reachable but some of the transitions are not permitted as constraints.

The HCM logic specification and its operations are tabulated as shown below in Table 1.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Instance</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode Operator</td>
<td>M1</td>
<td>Mode 1</td>
</tr>
<tr>
<td></td>
<td>M2</td>
<td>Mode 2</td>
</tr>
<tr>
<td></td>
<td>M3</td>
<td>Mode 3</td>
</tr>
<tr>
<td>Context Operator</td>
<td>D</td>
<td>Disturbance</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>Temperature</td>
</tr>
<tr>
<td>Fuzzy Operator</td>
<td>VL</td>
<td>Very Low</td>
</tr>
<tr>
<td></td>
<td>LO</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>HI</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>VH</td>
<td>Very High</td>
</tr>
<tr>
<td>Continuous State Space Variable</td>
<td>Z</td>
<td></td>
</tr>
</tbody>
</table>

| Path Quantifier | | |
|-----------------|------------------|
| A | All Paths |
| E | At least One Path |

| Temporal Operators | | |
|-------------------|------------------|
| F | Eventually |
| G | Generally |
| X | Next |
| U | Until |
2.1 Hybrid Context Mode Logic Representation

The figures below specify ways of representing the proposed logic. The state of a system may change in all its paths, when the temperature exceeds a specified limit. For example, when the surrounding temperature of a motor rotating in any direction suddenly changes from 40 degrees to 80 degrees, the motor seizes to rotate and comes to a standstill position no matter which state the motor is in which is shown in Fig.1. The state of the system may change in at least one of its paths, when there is very high external disturbance even though the temperature remains in its permissible limits. Consider a boiler plant, which consists of many sensors. There is a possibility that an external electrical disturbance due to electromagnetic radiation, causes malfunctioning in one of the sensors present, thereby causing a change in state in one of the paths of the system as shown in Fig.2.

![Fig. 1. A φ₁TV₁φ₂: In all paths, the state changes from φ₁ to φ₂ when temperature is very high.](image1)

![Fig. 2. ETL₀φ₁DV₁φ₂: In at least one path state change from φ₁ to φ₂ when the when the disturbance is very high even though the temperature is low.](image2)

It is also possible for a system to remain in its present state in all of its paths, as long as the temperature and external disturbance remains in its operating limits. In a motor, when there is no external electrical disturbance and the surrounding temperature remains within its operating limits, the motor continues to remain in the state in which it is currently present, represented by Fig.3. When a system can operate in more than one mode, the operating mode of the system needs to be specified. Consider a microcontroller that can operate in two different modes viz. sleep mode and active mode. It is imperative to specify the mode of operation before performing any task using the microcontroller which is shown in Fig.4.

![Fig. 3. ATL₀DL₀φ₁: In all paths, when the temperature is low and the disturbance is low, generally the state remains in φ₁.](image3)

![Fig. 4. M₁ATL₀F φ₁: In Mode 2, if the temperature is low, eventually in all paths, φ₁ is present.](image4)

3 Model Analysis of HCM Logic

The main focus of the work is to propose a context based assertion technique using hybrid context mode logic for the verification of the behaviour of the analog and mixed circuit IPs used in SoC based embedded system. An artificial limb controller is modelled and designed to illustrate the application of the proposed logic and its formal verification is carried out. Alloy language is a modelling language that follows first order logic to describe the system behaviour. The ALLOY analyzer developed by MIT’s software design group is used to check the user specified properties described in the ALLOY language. Alloy supports two kinds of analysis: simulation, in which the consistency of an invariant or operation is demonstrated by generating a state or transition, and checking, in which a consequence of the specification is tested by attempting to generate a counterexample. In context based checking using HCM logic, the model is verified for the various modes and contexts of operation like temperature changes and electrical noisy environment. The minimum and maximum change in temperature may be specified during the requirements stage well before the modelling starts. Similarly the working environment or the operational workspace of the embedded system should be pointed out in the early
stages of modelling to design the reaction of the software under those conditions. For example, the work considered two context parameters, say temperature change and the electrical disturbance adjacent to the device.

3.1 Behavior Model Based on Temperature Context
The system changes its states according to the variations in the temperature. At operating temperature, the state remains in the expected state (optimal_state) and the state changes from optimal state to accepted state (accepted_state) when temperature deviation in the accepted level (Min_deviate_temp). If the temperature deviation from the working temperature is large (Max_deviate_temp), the device resets (reset_state) itself. Once the device reaches the reset state, if the temperature falls back to the specified value (reset_temp), the device goes back to the initial state again as shown in Fig. 5.

3.2 Behavior Based on Electrical Disturbance
The device reacts to the disturbances around it, and changes its state appropriately as shown in Fig. 6. When the disturbance is low (low_disturbance), the device remains in normal state. If the disturbance increases (high_disturbance), the device may get affected, and behaviour of system changes from initial state (optimal_state) to disturbed state (disturbed_state). When the device is in the disturbed state a distorted output is produced. The system goes back to the initial state, if the disturbance is decreased. The other forms of electrical disturbances in the case of multiple devices are not considered in the present analysis.

3.3 Alloy Code for Temperature Context Based Model
The following Alloy code is written to specify the various state transitions that occur in a system depending upon the changes in temperature. Alloy can also be written to specify the system’s reaction to the surrounding disturbances.

```alloy
module temperature_behaviour
open util/ordering[State] as ord
//different temperature variations as objects.
abstract sig Object{
max_deviate_temp, min_deviate_temp, reset_temp:
set Object}
//State changes are defined as subsets of Object.
one sig accepted_state, reset_state, optimal_state extends Object {} //Defines the fact, how state changes during temperature variations.
fact state_changes
{max_deviate_temp= accepted_state-> reset_state+ optimal_state-> reset_state}
fact state_changes1
{min_deviate_temp= optimal_state-> accepted_state}
fact state_changes2
{reset_temp= reset_state-> optimal_state}
//Specifies the various modes, the device operates
sig State {mode_1: set Object, mode_2 : set Object}
//Defines the initial state.
fact initialState {let s0 = ord/first | s0.mode_1 = Object}
// Solves the problem to produce various instances present.
pred state_change [state_1,state_1', state_2, state_2']:
set Object}{
(state_1' = state_1-state_1.max_deviate_temp) ||
(state_2' = state_2-state_2.min_deviate_temp )}
//Indicates the various possible state transitions.
fact stateTransition {all s: State, s': ord/next[s]
{optimal_state in
s.mode_1 || optimal_state in s.mode_2 =>

Fig. 5. Temperature Context Based Model.

Fig. 6. Electrical Disturbance Context Based Checking.
state_change [s.mode_1, s'.mode_1, s.mode_2, s'.mode_2] else
state_change [s.mode_2, s'.mode_2, s.mode_1, s'.mode_1]}

//Predicate is run for two instances.
pred solve_changes
{ord/last.mode_2= Object || last.mode_1=Object}
rub solve_changes for 2

3.4 Alloy Results
The results for the Alloy code that was written for the change in temperature, change in disturbance and the complete model are compared in Table.2. The number of variables, primary variables, clauses and the time taken to arrive at the consistency of the predicate is also mentioned.

Table 2. Predicate Consistency Check.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperature Change</td>
<td>76</td>
<td>16</td>
<td>113</td>
<td>31ms.</td>
</tr>
<tr>
<td>Disturbance Change</td>
<td>67</td>
<td>11</td>
<td>97</td>
<td>16ms.</td>
</tr>
<tr>
<td>Complete Model</td>
<td>102</td>
<td>46</td>
<td>117</td>
<td>15ms.</td>
</tr>
</tbody>
</table>


4 Reachable State Permissible Transition (RSPT) Algorithm
The RSPT algorithm shows the reachable states and permissible transitions that are allowed in the system. The reachable states are those states that the system can reach while non-reachable states are those which the system can never access. Transitions can occur between the various reachable states present in the system. The transitions that are allowed from the present state to the next state are called permissible transitions and those which are not are non-permissible transitions.

DESTINATION_STATES is a set of all possible states, INITIAL_STATES is a set of all possible reachable states. An example of an artificial arm, which consists of three motors for fingers, elbow and shoulder are taken. The finger motor performs seize and release operation, the elbow motor performs stretch and fold, and the shoulder motor performs up, down and back operations. The various reachable states and permissible transitions of the proposed artificial arm controller are shown in Fig.7. The complexity of the algorithm is in the order of \(2^n\) where \(n\) represents the total number of actuators and at the same time it checks the consistency of the entire model with 102 variables. If more number of states is permitted, the algorithm finds a little time consuming to reach the consistent state. The algorithm also predicts whether the model reaches the wait state or error state if the transition is not getting completed within a specified time delay as shown below.

| Destination_states = \{s_1, s_2, s_3, \ldots, s_n\} |
| Initial_states \subseteq Destination_states |
| Initial_states = \{r_1, r_2, r_3, \ldots, r_n\} |
| Permissible_transitions = \{t_1, t_2, t_3, \ldots, t_n\} |
| Current_State = Initial_states |
| While Current_State! = NULL{ |
| REMOVE Some s From Current_State if (s \in Destination_states) |
| Return (Destination Reached) else |
| Current_State = Current_State U\{s' | s' t s | t \in T\} |
| Return (Not Reachable State) |

Fig.7. RSPT Algorithm.

5 SoC Based Embedded System Assertion using C
This pSoC model is used to read the system temperature by using a temperature sensor through the temp_sensor pin and the disturbance through the disturbance_sensor pin. The value from the temp_sensor is given to two comparators and is checked if it lies within the acceptable limits. Similarly the sensed disturbance is compared with the acceptable disturbance level. If either temperature or disturbance exceeds the acceptable limit, the system is reset. The three PWMs control the three actuators of the artificial arm controller (finger motor, elbow motor and shoulder motor) as
shown in Fig.8. The first comparator (Comp_1) is used to compare the operating temperature of the system with the maximum level of temperature that can be tolerated by the system (Max_accepted_temp). The second comparator (Comp_2) checks if the sensed temperature falls below the minimum acceptable temperature (Min_accepted_temp). A third comparator (Comp_3) compares the external electrical disturbances with the maximum acceptable level of disturbance (Accept_dist_level). The outputs of Comp_1 and Comp_2 are given to a XOR gate. The output of the XOR gate and the result of COMP_3 is given to an OR gate. The working of the SoC based embedded system depends upon various parameters and is necessary to assert them. The system consists of four clocks, one for the three comparators and one each for each PWM. It is essential for all the clocks to operate in synchronism to ensure proper functioning of the system. Input given to a comparator should lie within its recommended limits (V_{dd} to V_{ss}). When the input exceeds the maximum limit, it may damage the comparator device. On the other hand the comparator refuses to sense the input given if the input falls below the threshold value (V_{th}).

6 Conclusion
Formal Verification of an embedded system using software defined hardware intellectual properties configured into a System On Chip mixed signal micro controller is carried out using the HCM Logic. The model is analyzed for its reachable states and timeliness behavior using Alloy analyzer and the results are tabulated. The reachable states and permissible transitions in an artificial embedded arm controller are identified using RSPT algorithm. The mixed signal components and its circuits are verified using the hybrid context mode logic by asserting the behavior of the system using embedded C language code. The limitation of the work is the size of embedded application to be built using the limited configurable blocks in the target micro controller device. The HCM logic may have serious limitation while verifying the behavior of a distributed embedded system like a web server. The future work targets the design of a suitable formal verification technique when number of such devices is collaborating to complete the submitted tasks.
References: