# Synthesis of AMBA AHB from Formal Specification

Yashdeep Godhal Krishnendu Chatterjee Thomas A. Henzinger

IST Austria (Institute of Science and Technology Austria)

**Abstract**

The standard procedure for hardware design consists of describing circuit in a hardware description language at logic level followed by extensive verification and logic-synthesis. However, this process consumes significant time and needs a lot of effort. An alternative is to use formal specification language as a high-level hardware description language and synthesize hardware from formal specification. In [1] formal specifications for AMBA AHB Arbiter were presented and synthesized. Our contributions are as follows: (1) We present more complete and compact formal specifications for the AMBA AHB Arbiter, and obtain significant (order of magnitude) improvement in synthesis results (both with respect to time and the number of gates of the synthesize circuit); (2) we present formal specification and synthesize to generate compact circuits for the remaining two components of the AMBA AHB protocol, namely, the AMBA AHB Master and AMBA AHB Slave; and (3) from the lessons learnt we present few principles for writing formal specifications for efficient hardware synthesis. Thus with systematically written complete formal specifications we are able to automatically synthesize an important and widely used industrial protocol.

## 1 Introduction

**Hardware design flow.** The first step in traditional standard industrial procedure of hardware design is the decription of a circuit in hardware description language. This step is followed by extensive verification and subsequently by logical synthesis. The outcome of logical syntehsis is gate level implementation of circuit. Among the above steps of design, verification and logical synthesis, the verification step is most time consuming process and requires a lot of effort. An alternative approach is to automatically synthesize the circuit from formal specification.

**Synthesis from formal specification.** Historically, automatic synthesis of digital designs from logical temporal specifications has been considered as one of the most challenging problems in circuit design. The problem was first presented by Church [4] and several methods have been proposed as solutions such as [3] and in [12]. The problem was considered again in [10] in the context of synthesizing reactive modules from a specification given in Linear Temporal Logic (LTL). The method proposed in [10] for a given LTL specification $\varphi$ starts by constructing a Büchi automaton which is converted into a deterministic Rabin automaton. This translation may require a doubly exponential complexity in the size of $\varphi$. The high complexity established in [10] caused synthesis to be deemed hopelessly intractable and discouraged practitioners from attempting to use it for system development. Yet, there exist several interesting cases where, if the specification of the design to be synthesized is restricted to simpler automata or partial fragments of LTL, it has been shown that the synthesis problem can be solved in polynomial time. Major progress has been achieved in [9], which shows that designs can be automatically synthesized from LTL formulas belonging to the class of generalized reactivity of rank 1 (GR(1)), in time $N^3$ where $N$ is the size of the state space of the design. The class GR(1) covers the vast majority of properties that appear in specifications of circuits. The approach of [9] was implemented by [1] in a tool called Anzu [6]. Anzu produces not only a BDD representing a set of possible implementations, but also an actual circuit.

**AMBA AHB Protocol.** In this work we study the automatic synthesis of an important and widely used industrial protocol, namely, *AMBA AHB* protocol. ARM's *Advanced Microcontroller Bus Architecture*

(AMBA) [8] specification defines an on chip communications standard for designing high-performance embedded microcontrollers. AMBA is today the de-facto standard for embedded processors because it is well documented and can be used without royalties. It is widely used in network interconnect chips, RAM and Flash memory controllers, DMA controllers, level2 cache controllers and SoCs including application processors used in portable mobile devices like smartphones, and a few industrial examples of its use are IXP42X Product Line of Intel Network Processors, Infineon gateway controller ADM5120. The most important bus defined within the AMBA specification is *Advanced High-performance Bus AHB*. The AHB acts as the high-performance system backbone bus. AHB supports the efficient connection of processors, on-chip memories, DMA controllers and off-chip external memory interfaces. The AMBA AHB design consists of the following main components: (a) AHB Arbiter; (b) AHB Master and (c) AHB Slave. In this work we synthesize the above three components of the AMBA AHB protocol.

**Our contributions.** The contributions of this work are as follows:

1. In [1] and [2] the synthesis of only AMBA AHB Arbiter was studied. We present more complete and compact formal specifications for the AMBA AHB Arbiter, and obtain significant (order of magnitude) improvement in synthesis results (both with respect to time and the number of gates of the synthesized circuit).

2. We present, for the first time, the formal specifications for the AMBA AHB Master and AMBA AHB Slave (the remaining two components of the protocol). We are able to synthesize very compact circuits from our formal specifications. Thus we are able to completely synthesize an important and widely used industrial protocol by systematically writing the formal specifications.

3. From the lessons that we have learnt in the process of rewriting specifications to obtain efficient synthesis result, we present few principles for writing formal specifications for efficient hardware synthesis.

# 2 Preliminaries

In this section we present preliminaries related to specification language and synthesis.

## 2.1 Property Specification Language

We will use *Property Specification Language (PSL)* to express specifications (a detailed description of PSL can be found in [5]). The specifications presented in this paper are easy to follow for readers familiar with LTL. In particular, always, eventually, and next correspond to $G$, $F$, and $X$, respectively. The until_ operator requires the first operand to hold either forever or up to and including the time that the second operand holds. The construct ($\Phi$ before $\Psi$) is equivalent to ($\neg\Psi$ until_ $\Phi$). We also use one operator that is not in PSL: ($\Phi$ until_[$i$] $\Psi$) means that $\Phi$ holds either forever or up to and including the $i^{th}$ time that $\Psi$ holds.

## 2.2 Synthesis of GR(1) Properties

We briefly review the results presented in [9] on synthesizing GR(1) properties. We are interested in the question of *realizability* of PSL specifications (cf [11]). Assume two sets of Boolean variables $X$ and $Y$. Intuitively $X$ is the set of input variables controlled by the environment and $Y$ is the set of system variables. *Realizability* amounts to checking whether there exists an *open controller* that satisfies the specification. Such a controller can be represented as an automaton which, at any step, reads values of the $X$ variables and outputs values for the $Y$ variables.

Here we concentrate on a subset of PSL for which realizability and synthesis can be solved efficiently. The specifications we consider are of the form $\phi = \phi^e \rightarrow \phi^s$. We require that $\phi^\alpha$ for $\alpha \in \{e, s\}$ can be rewritten as a conjunction of the following parts.

- $\phi_i^\alpha$ - a Boolean formula which characterizes the initial states of the implementation.

- $\phi_t^\alpha$ - a formula of the form $\wedge_i$(always $B_i$) where each $B_i$ is a Boolean combination of variables from $X \cup Y$ and expressions of the form (next $v$) where $v \in X$ if $\alpha = e$, and $v \in X \cup Y$ otherwise.
- $\phi_g^\alpha$ - has the form $\wedge_{i \in I}$ (always eventually $B_i$) where each $B_i$ is a Boolean formula.

In order to allow formulas of other forms (e.g. always$(p \rightarrow (q$ until_ $r))$ where $p$, $q$, and $r$ are Boolean), we augment the set of variables by adding deterministic monitors. Deterministic monitors are variables whose behavior is deterministic according to the choice of the inputs and the outputs. These monitors follow the truth value of the expression nested inside the always operator. We rewrite these types of formulas to the form (always eventually $b$) where $b$ is a Boolean formula using the variables of the monitor. It should be noted that even with these restrictions, all possible (finite state) designs can be expressed as a set of properties.

We reduce the realizability problem of a PSL formula to the decision of the winner in a two-player game played between system and environment. The goal of the system is to satisfy the specification regardless of the actions of the environment. A *game structure* is a multigraph whose nodes are all the truth assignments to $X$ and $Y$. A node $v_1$ is connected by edges to all the nodes $v_2$ such that the truth assignments to $X$ and $Y$ satisfy $\phi_t^e \wedge \phi_t^s$ , where $v_1$ supplies the assignments to the current values and $v_2$ to the next values. We then group all the edges that agree on the assignment of $X$ in $v_2$ to one multi-edge. A play starts by the environment choosing an assignment to $X$ and the system choosing a state in $\phi_i^e \wedge \phi_i^s$ that agrees with this assignment. A play proceeds by the environment choosing a multi-edge and the system choosing one of the nodes connected to this multi-edge. The system wins if this interaction produces an infinite play that satisfies $\phi_g^e \rightarrow \phi_g^s$.

We *solve* the game to decide whether the game is winning for the environment or the system. If the environment wins, then the specification is *unrealizable*. If the system wins, then we *synthesize* a winning strategy. This strategy, a BDD, is a nondeterministic representation of a working implementation. The following theorem summarizes the result of synthesis of PSL specifications.

**Theorem 1** *[9] Given sets of variables $X$ and $Y$ and a PSL formula $\phi$ of the form presented above with $m$ and $n$ conjuncts, we can determine using a symbolic algorithm whether $\phi$ is realizable in time proportional to $O(mn2^{d+|X|+|Y|})$, where $d$ is the number of variables added by the monitors for $\phi$.*

## 2.3 Generating circuits from BDDs

We briefly review the results presented in [2] on generating circuits from BDDs. The strategy is a BDD over the variables $X$, $Y$, $X^{'}$ and $Y^{'}$, where $X$ are input variables, $Y$ are output variables and the primed versions represent next state variables. The corresponding circuit contains $|X| + |Y|$ flipflops to store the values of the inputs and outputs in the last clock tick. In every step, the circuit reads the next input values $X^{'}$ and determines the next output values using combinational logic with inputs $I = X \cup Y \cup X^{'}$ and outputs $O = Y^{'}$. The strategy does not prescribe a unique combinational output for every combinational input. In most cases, multiple outputs are possible, in states that are not reachable (assuming that the system adheres to the strategy), no outputs may be allowed.

We write $o \in O$ for a combinational output and $i \in I$ for a combinational input. The strategy is denoted by $S$ and $O \setminus o$ is the set of combinational outputs excluding output $o$. For every combinational output $o$ we construct a function $f$ in terms of $I$ that is compatible with the given strategy BDD. The algorithm proceeds through the combinational outputs $o$ one by one: First, we build $S^{'}$ to get a BDD that restricts only $o$ in terms of $I$. Then we build the *positive and negative cofactors* $(p, n)$ of $S^{'}$ with respect to $o$, that is, we find the sets of inputs for which $o$ can be 1 (0, respectively). For the inputs that occur in the positive and in the negative cofactor, both values are allowed. The combinational inputs that are neither in the positive nor in the negative cofactor are outside of the winning region and thus represent situations that cannot occur (as long as the environment satisfies the assumptions). Thus, $f$ has to be 1 in $p \wedge \neg n$ and 0 in $\neg p \wedge n$, which give us the set of care states. We minimize the positive cofactors with the care set to obtain the function $f$. Finally, we substitute variable $o$ in $S$ by $f$, and proceed with the next variable. The substitution is necessary since a combinational outputs may be related.

The resulting circuit is constructed by writing the BDDs for the functions using CUDDs DumpBlif command [13]. We then optimize the result using ABC [14] and map it to a library of standard cells. We also use ABC to estimate the number of gates needed.

# 3 AMBA AHB Protocol

In this section we describe the details of the main components of the AMBA AHB protocol. ARM's *Advanced Microcontroller Bus Architecture* (AMBA) [8] specification defines an on chip communications standard for designing high-performance embedded microcontrollers. The most important bus defined within the AMBA specification is *Advanced High-performance Bus*. The AHB acts as the high-performance system backbone bus. AHB supports the efficient connection of processors, on-chip memories, DMA controllers and off-chip external memory interfaces. The AMBA AHB design contains the following components:

**AHB master**: A bus master is able to initiate read and write operations by providing an address and control information. Only one bus master is allowed to actively use the bus at any one time.

**AHB slave**: A bus slave responds to a read or write operation within a given address-space range. The bus slave signals back to the active master the success, failure or waiting of the data transfer.

**AHB arbiter**: The bus arbiter ensures that only one bus master at a time is allowed to initiate data transfers. Even though the arbitration protocol is fixed, any arbitration algorithm, such as highest priority or fair access can be implemented depending on the application requirements.

**AHB decoder**: The AHB decoder is used to decode the address of each transfer and provide a select signal for the slave that is involved in the transfer.

Consider an AHB system with arbiter, masters and slaves. Every slave shall have some address range. AHB decoder receives address as input, checks in which range that address lies and provides select signal for slave that corresponds to this address. In essence, it works as a de-multiplexer. For a system with single slave, the select signal shall always be high, if valid address is put on bus. Hence we consider the synthesis of the main components of AHB design i.e. AHB Master, AHB Slave and AHB Arbiter.

## 3.1 AHB Arbiter

The role of the arbiter in an AMBA system is to control which master has access to the bus. Every bus master has a REQUEST/GRANT interface to the arbiter and the arbiter uses a prioritization scheme to decide which bus master is currently the highest priority master requesting the bus. Each master also generates an $\mathrm{HLOCK_x}$ signal which is used to indicate that the master requires exclusive access to the bus. The arbitration protocol is not specified and can be defined for each application.

## 3.2 AHB Master

Function of AHB master is to do read and write operations. Before initiating any transfer, it sends a request to arbiter for accessing bus. Once arbiter grants the bus, master initiates read/write operation by providing address and control information. Master 0 is the *default master* and is selected whenever there are no requests for the bus.

## 3.3 AHB Slave

An AHB bus slave responds to transfers initiated by bus masters within the system. The slave uses a select signal $\mathrm{HSEL_x}$ from the decoder to determine when it should respond to a bus transfer. All other signals required for the transfer, such as the address and control information, will be generated by the bus master.

The AHB is a pipelined bus. This means that different masters can be in different stages of communication. At one instant, multiple masters can request the bus, while another master transfers address information, and a yet another master transfers data. A bus *access* can be a *single* transfer or a *burst*, which consists of a specified or unspecified number of transfers. Access to the bus is controlled by the arbiter. All

devices that are connected to the bus are Moore machines, that is, the reaction of a device to an action at time $t$ can only be seen by the other devices at time $t + 1$.

# 4 AMBA AHB Arbiter Synthesis

In this section we present our results related to synthesis of AHB arbiter. We first present the arbiter signals, then present the formal specifications and our result for synthesis.

## 4.1 AHB Arbiter Signals

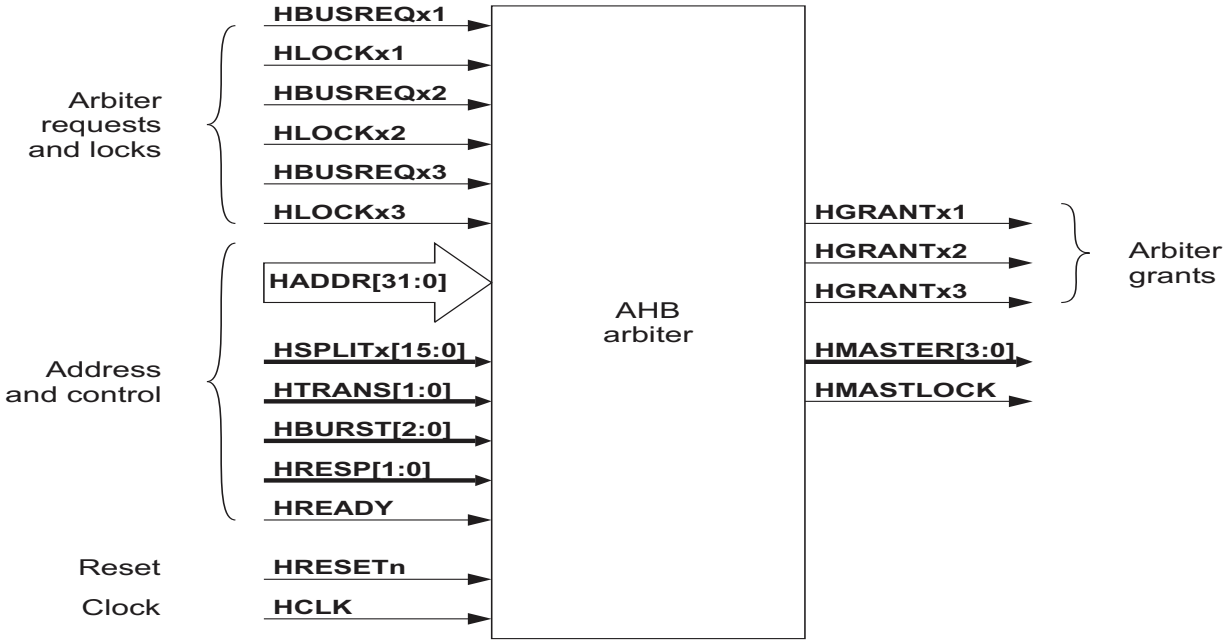


Figure 1: AHB Arbiter [8]

Figure 1 shows AHB arbiter signals. The description of these signals are as follows (the notation S[n:0] denotes an (n+1)-bit signal):

- $\text{HBUSREQ}_\text{i}$ - A signal from bus master $i$ to the bus arbiter which indicates that the bus master requires access to the bus.
- $\text{HLOCK}_\text{i}$ - Indicates that the master requires locked access to the bus. No other master should be granted the bus until this signal is lowered.
- HREADY- This signal is driven by the bus slave. It indicates that a transfer has finished on the bus. This signal may be lowered to extend a transfer.
- $\text{HGRANT}_\text{i}$ - This signal indicates that if HREADY is high, then HMASTER= i will hold in the next tick.
- HMASTLOCK - Indicates that the current master is performing a locked sequence of transfers.
- HMASTER[3:0] - These signals from the arbiter indicate which bus master is currently performing a transfer.

The following signals are multiplexed using HMASTER as the control signal. For example, although every master has an address bus, only the address provided by the currently active master is visible on HADDR.

- HADDR[31:0] - These signals indicate the address where read or write transaction will take place.

- HBURST[1:0] - One of SINGLE (a single transfer), INCR (unspecified length burst) or INCR4 (burst of four transfers). Though the standard allows for burst of eight and sixteen transfers too but we have not taken them into account. That would lengthen the specification.
- HTRANS[1:0] - Indicates the type of the current transfer, which can be NONSEQ, SEQ or IDLE. The standard allows for BUSY transfers also. HTRANS = BUSY indicates that master wants to introduce some delay during ongoing transfer. This is an optional feature. For simplicity we have left this feature out.

Furthermore, as an optional feature of the AHB, a slave is allowed to split a burst access and request that it be continued later (signals HRESP and HSPLIT shown in Figure 1 serve that purpose). We have left this feature out for simplicity.

Both optional features i.e. SPLIT and BUSY transfers are also not considered in [1] while writing specifications for AHB Arbiter. Though they can be handled by this approach but that would lengthen the specification.

## 4.2 Formal Specifications

The first formal specification for AMBA AHB arbiter was given in [1]. We have systematically re-written the specifications to make it more complete. The two important changes are as follows: (a) the HTRANS[1:0] signal, which plays an important role in AHB transfers, was not used in earlier specifications, whereas with the use of HTRANS signal, we make the formal specifications more complete; and (b) the other important change from the specifications of [1] is related to de-assertion of HLOCK signal: according to ARM [7], the AHB Master should deassert the HLOCK signal when the address phase of the last transfer in the locked sequence has started.

Along with the signals described above, we use two auxilary signals DECIDE and BUSREQ, that were introduced in [2]. The signal DECIDE indicates the time slot in which the arbiter decides who the next master will be and whether its access will be locked. The decision is based on $\text{HBUSREQ}_\text{i}$ and $\text{HLOCK}_\text{i}$. The signal BUSREQ points to the $\text{HBUSREQ}_\text{i}$ signal of the master that currently owns the bus. Two auxilary variables START and LOCKED, that were introduced in [1], are not used in our specification. It is because with the inclusion of HTRANS signal and change of nature in de-assertion of HLOCK signal, START and LOCKED have become redundant. We introduce a new auxilary variable GRANTED which is driven by the arbiter. The signal GRANTED is used for deciding start of new access. When both GRANTED and HREADY signals are high simultaneously, new access shall start in next cycle. Thus a decision can be executed at the earliest two time steps after the $\text{HBUSREQ}_\text{i}$ and $\text{HLOCK}_\text{i}$ signals are read.

We follow the convention used in [1]: guarantees are properties that the arbiter must fulfill, and assumptions are properties that the arbiters environment must fulfill. Our specification for the arbiter consists of *9 assumtions* and *12 guarantees* whereas the specification from paper [1] had *4 assumptions* and *11 guarantees*. Figure 2 shows timing diagram for AHB arbiter signals. Table 1 contains formal specification of arbiter in PSL. The bold faced **A** and **G** signify new/re-written property whereas non-bold faced indicate existing property from [2]. The assumptions(A) and guarantees(G) for the arbiter are described below.

**Assumptions** The assumptions are as follows.

A1 During a locked unspecified length burst, leaving $\text{HBUSREQ}_\text{i}$ high locks the bus. This is forbidden by the standard.
A2 Leaving HREADY low locks the bus, the standard forbids it.
A3 Signals $\text{HLOCK}_\text{i}$ and $\text{HBUSREQ}_\text{i}$ are asserted by AHB Master at the same time.
**A4** When HREADY signal is low, all control signals should hold their values.
**A5** If no transfer is taking place, HTRANS signal can not become SEQ in the next cycle.
**A6** In burst sequence (i.e. HBURST = INCR4), if HREADY is high, NONSEQ transfer shall always be followed by SEQ transfer.
**A7** First transfer of any AHB sequence is NONSEQ in nature.
**A8** When none of the AHB Masters is making a request for bus, no transfer will take place.
A9 All input signals are low initially.

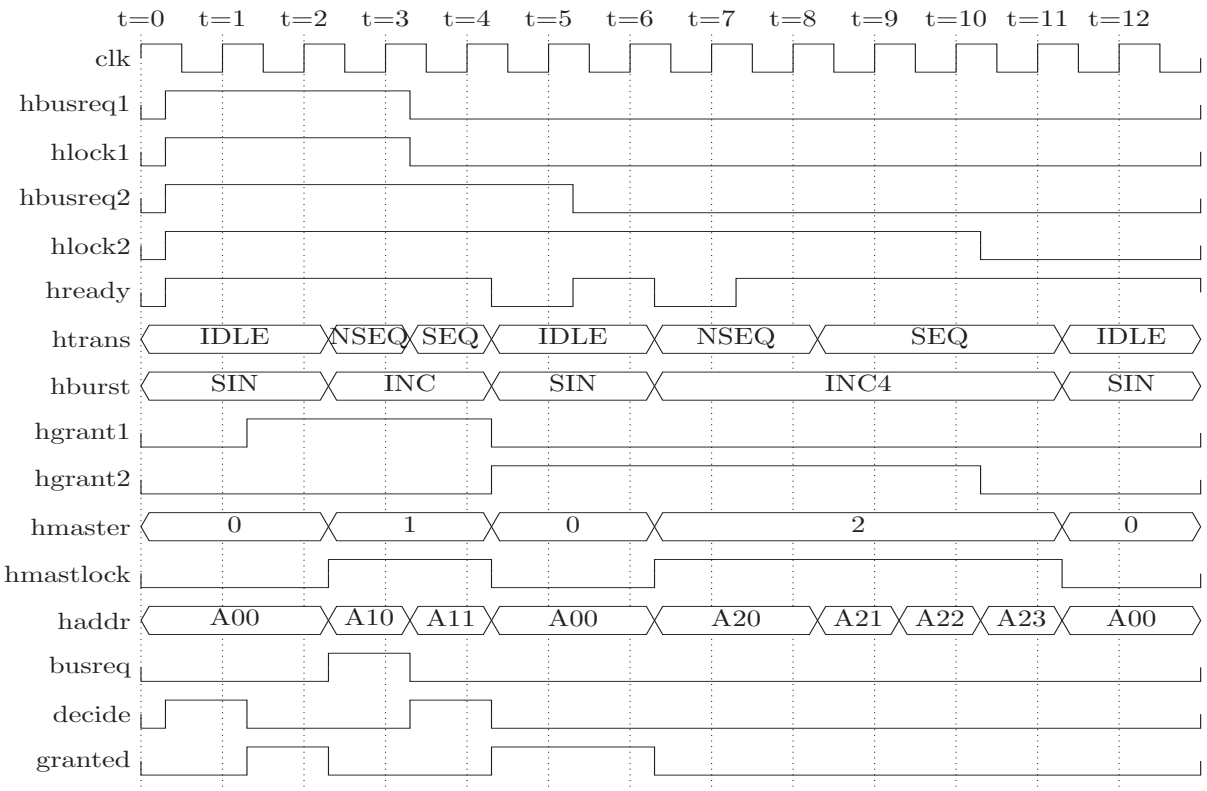


Figure 2: Signals for the AHB Arbiter and timing diagram

Table 1: PSL Specifications for AHB Arbiter.

| | |
|---|---|
| A1 | $\forall i$ : always ((HMASTLOCK $\wedge$ HBURST = INCR) $\rightarrow$ ( next eventually $\neg$BUSREQ)) |
| A2 | always eventually HREADY |
| A3 | $\forall i$ : always (($\neg$HBUSREQ$_i$ $\wedge$ $\neg$HLOCK$_i$ $\wedge$ (next HLOCK$_i$)) $\rightarrow$ (next HBUSREQ$_i$)) |
| A4 | always ($\neg$HREADY $\rightarrow$ (HTRANS = j $\leftrightarrow$ next HTRANS = j))<br>always ($\neg$HREADY $\rightarrow$ (HBURST = j $\leftrightarrow$ next HBURST = j)) |
| A5 | always ((HTRANS = IDLE) $\rightarrow$ (next (HTRANS $\neq$ SEQ))) |
| A6 | always (((HTRANS = NONSEQ) $\wedge$ (HBURST = INCR4) $\wedge$ HREADY) $\rightarrow$ (next (HTRANS = SEQ))) |
| A7 | always ((GRANTED $\wedge$ HREADY) $\rightarrow$ (next (HTRANS = NONSEQ))) |
| A8 | always (($\wedge_{i=0}^{n-1}\neg$HBUSREQ$_i$) $\rightarrow$ (HTRANS = IDLE)) |
| A9 | $\forall i$ : ($\neg$HBUSREQ$_i$ $\wedge\neg$HLOCK$_i$ $\wedge\neg$HREADY $\wedge$ (HTRANS = IDLE) $\wedge$ (HBURST = SINGLE)) |
| G1 | $\forall i$ : always ((HMASTER = i ) $\rightarrow$ (BUSREQ $\leftrightarrow$ HBUSREQ$_i$)) |
| G2 | $\forall i$ : always ((HMASTLOCK $\wedge$ (HBURST = INCR) $\wedge$ HREADY $\wedge$ (HTRANS = NONSEQ)) $\rightarrow$<br>next ((HTRANS = SEQ) until_ $\neg$BUSREQ)) |
| G3 | $\forall i$ : always ((HMASTLOCK $\wedge$ (HBURST = INCR4) $\wedge$ HREADY $\wedge$ (HTRANS = NONSEQ)) $\rightarrow$<br>next ((HTRANS = SEQ) until_[3] HREADY)) |
| G4 | always ((DECIDE $\wedge(\vee_{i=0}^{n-1}$ HBUSREQ$_i$)) $\rightarrow$ (next GRANTED)) |
| G5 | always ((GRANTED $\wedge\neg$HREADY) $\rightarrow$ (next GRANTED))<br>always ((GRANTED $\wedge$ HREADY) $\rightarrow$ (next $\neg$GRANTED)) |
| G6 | $\forall i$ : always (HREADY $\rightarrow$ (HGRANT$_i$ $\leftrightarrow$ next (HMASTER = i))) |
| G7 | always ((HREADY $\wedge(\vee_{i=0}^{n-1}$ (HLOCK$_i$ $\wedge$ HGRANT$_i$)) $\rightarrow$ next (HMASTLOCK))) |
| G8 | $\forall i$ : always (($\neg$HREADY$\vee$ $\neg$GRANTED) $\rightarrow$ (HMASTER= i$\leftrightarrow$ next HMASTER= i))<br>$\forall i$ : always (($\neg$HREADY$\vee$ $\neg$GRANTED) $\rightarrow$ (HMASTLOCK$\leftrightarrow$ next HMASTLOCK)) |
| G9 | $\forall i$ : always ($\neg$DECIDE $\rightarrow$ (HGRANT$_i$ $\leftrightarrow$ next HGRANT$_i$)) |
| G10 | $\forall i \neq 0$ : always ($\neg$HGRANT$_i$ $\rightarrow$ (HBUSREQ$_i$ before HGRANT$_i$))<br>always (DECIDE $\wedge\forall i$ : $\neg$HBUSREQ$_i$ $\rightarrow$ next HGRANT0) |
| G11 | $\forall i$ : always (HBUSREQ$_i$ $\rightarrow$ eventually ($\neg$HBUSREQ$_i$ $\vee$ (HMASTER = i))) |
| G12 | DECIDE $\wedge$ HGRANT0 $\wedge$ (HMASTER = 0) $\wedge$ $\neg$GRANTED $\wedge\neg$HMASTLOCK $\wedge\forall i \neq 0$ : $\neg$HGRANT$_i$ |

**Guarantees** The guarantees are as follows.

G1 Variable BUSREQ points to HBUSREQ$_i$ of the master that is currently granted access to the bus.

**G2** When a locked unspecified length burst starts, a new access does not start until the currentmaster (i) releases the bus by lowering HBUSREQi.

**G3** When a length-four locked burst starts, no other accesses start until the end of the burst. We can only transfer data when HREADY is high, so the current burst ends at the fourth occurrence of HREADY.

**G4** Whenever, there is at least one bus request present and signal DECIDE is high, GRANTED gets asserted in the next cycle.

**G5** If HREADY is low, then GRANTED signal holds its value. Whereas, if HREADY and GRANTED signals are simultaneously high, then GRANTED gets deasserted in next cycle.

G6 The HMASTER signal follows the grants: When HREADY is high, HMASTER is set to the master that is currently granted. It implies that no two grants may be high simultaneously and the arbiter cannot change HMASTER without giving a grant.

**G7** Whenever signal HREADY, HLOCK$_i$ and HGRANT$_i$ are simultaneously high, HMASTLOCK gets asserted in the following cycle.

**G8** When any of GRANTED or HREADY signals is low, the HMASTER and HMASTLOCK signals do not change.

G9 Whenever DECIDE is low, $HGRANT_i$ signal do not change.
G10 We do not grant the bus without a request, except to Master 0. If there are no requests, the bus is granted to Master 0.
G11 We have a fair bus i.e. every master that has made a request shall be serviced eventually.
G12 The signals DECIDE and HGRANT0 are high at first clock tick and all others are low.

Assumptions A1, A2, A3, A9 and Guarantees G1, G2, G3, G6, G8, G9, G10, G11, G12 mentioned above are taken directly from [1]. Remaining guarantees in [1] were related to auxilary signals which have become redundant in our case with inclusion of HTRANS signal. Out of the above, G2, G3 and G8 have been re-written with the original meaning kept intact. Thus all assumptions and guarantees from [1] are taken care in our specification, and along with it we have more assumptions and guarantees.

### 4.3 Synthesis Results

| Num of Masters | Synthesis time (sec) from Fig 8 in [2] | Synthesis time (sec) for specifications [2] in our experiments | Minimum synthesis time (sec) of the last two columns | Synthesis time(sec) for our specifications |
|---|---|---|---|---|
| 2 | 2 | 2 | 2 | 1 |
| 3 | 20 | 22 | 20 | 5 |
| 4 | 100 | 103 | 100 | 9 |
| 5 | 200 | 203 | 200 | 53 |
| 6 | 800 | 677 | 677 | 86 |
| 7 | 2400 | 2696 | 2400 | 206 |
| 8 | 12000 | 7931 | 7931 | 146 |
| 9 | 2000 | 2533 | 2000 | 550 |
| 10 | 19000 | 18789 | 18789 | 630 |
| 11 | | | | 577 |
| 12 | | | | 992 |
| 13 | | | | 1610 |
| 14 | | | | 2100 |
| 15 | | | | 3486 |
| 16 | | | | 3630 |

Table 2: Synthesis time comparison

Anzu [6] is used to synthesize the circuit from specifications. Table 2 shows comparison of time taken by Anzu tool to synthesize AHB arbiter for different specifications. First column shows number of masters for which arbiter was synthesized. Second column shows data taken from Figure 8 of [2] and third column shows time taken in synthesizing specification from [2] on our machine (2GB RAM). In fourth column, we have taken the minimum of these two columns to have the best possible estimate of synthesis time for arbiter specifications in [2]. Fifth column shows the time in seconds for the arbiter synthesized using our formal specifications.

The results (Table 2) show that using the earlier specifications from [2], the synthesis procedure fails for more than 10 masters. With our improved specifications we can synthesize arbiter serving upto 16 masters nearly in an hour. The AHB standard allows for maximum 16 masters, and arbiter synthesized using our specifications can serve upto 16 masters. Thus we are able to syntesize arbiter serving the maximum number of masters as required by the protocol. Moreover, our improved specifications show significant (order of magnitude) improvement over the earlier specification: for example, for arbiter serving 10 masters the synthesis of earlier specifications takes nearly 5 hours, whereas our specification is synthesized in less than 11 minutes.

In Table 3, NA corresponds to not available and NM refers to not mappable by ABC.

| Num of Masters | Gate count from Fig 9 in [2] | Gate count for specifications [2] in our experiments | Minimum gate count of the last two columns | Gate count for our specifications |
|---|---|---|---|---|
| 2 | 1000 | 982 | 982 | 182 |
| 3 | 3500 | 2626 | 2626 | 409 |
| 4 | 8500 | 6801 | 6801 | 776 |
| 5 | 11000 | 9033 | 9033 | 920 |
| 6 | 18000 | 12448 | 12448 | 1443 |
| 7 | 15000 | 19777 | 15000 | 2015 |
| 8 | 36000 | NM | 36000 | 2431 |
| 9 | NA | 50012 | 50012 | 3047 |
| 10 | 50000 | 45912 | 45912 | 2825 |
| 11 | | | | 2994 |
| 12 | | | | 5178 |
| 13 | | | | 3712 |
| 14 | | | | 4112 |
| 15 | | | | 4199 |
| 16 | | | | 6056 |

Table 3: Gate count comparison

Anzu [6] tool generates a file in .blif format. This file is mapped by using ABC [14] to standard library. ABC tool is also useful for counting number of gates required to realize the circuit. Table 3 shows comparison of number of gates mapped by ABC for realizing different specifications for arbiter. First column shows the number of masters for which the arbiter is synthesized. Second column shows data taken from Figure 8 in [2] and third column shows number of gates mapped by ABC tool on our machine (2GB RAM) for existing specification in [2]. In fourth column, we have taken the minimum of second and third columns to have a best estimate of number of gates for existing specifications. In fifth column, gate count for our circuit synthesized from our specification. Table 3 shows that arbiter synthesized using specifications from [2] serving 10 masters has nearly forty-six thousand gates, whereas, the AHB arbiter synthesized with our specifications serving 10 masters has only three thousand gates, and even arbiter serving 16 masters needs only six thousand gates. Thus our specifications not only improve the time taken for synthesizing, but also improve the gate count of synthesized circuit by an order of magnitude.

Graphical comparison for arbiters serving different number of masters is shown in Figure 3 and Figure 4. Figure 3 shows comparison for synthesis time whereas Figure 4 depicts comparison for gate count.

# 5 AMBA AHB Master

In this section we present the synthesis results for AHB Master: we first present the signals, then the specification, and then the synthesis results.

## 5.1 AHB Master Signals

We first introduce the signals for AHB Master that have not been introduced.

- HWRITE - This signal from bus master indicates nature of transfer. When HWRITE is low, it indicates read transfer. If high, it indicates write transfer.
- HADDR[31:0] - These signals from the master provide information about location where write or read transfer shall take place.
- HWDATA[31:0] - These signals from the master provide information about data to be written in case of write transaction.

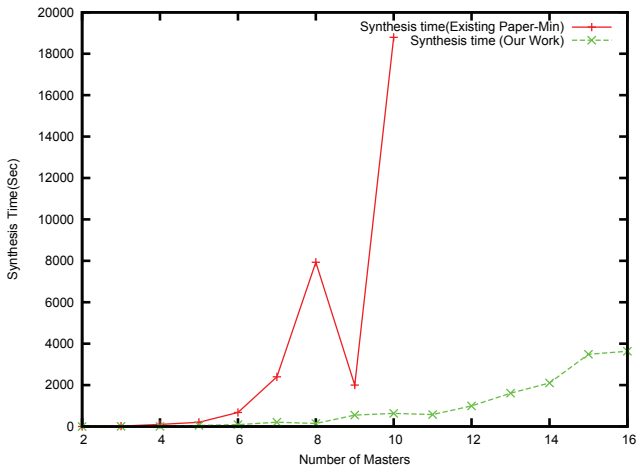


Figure 3: Synthesis Time Comparison.

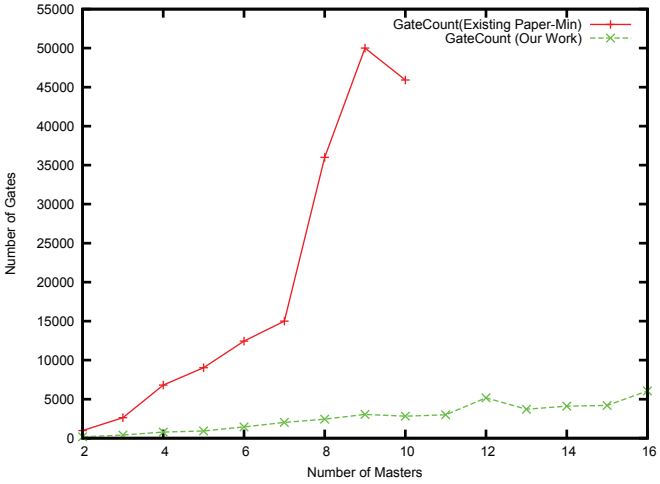


Figure 4: Gate Comparison.

- HRDATA[31:0] - These signals from bus slave to bus master provide information about data read in case of read transaction.
- HSIZE[2:0] - This signal from bus master to bus slave provides information about bus width. It can be one of byte(8-bit), half-word(16-bit), word(32-bit) and up to 1024 bits. We have fixed it to word i.e. data bus shall be 32-bit wide.
- HRESP[1:0] - This signal from bus slave to bus master provides transfer response. It can be one of OKAY, ERROR, SPLIT and RETRY. SPLIT and RETRY are optional feature allowed in standard. For simplicity, we have fixed it to OKAY otherwise it would lengthen the specification.

The AMBA AHB specification also allows protection controls but for simplicity, we have left that feature out. Few auxilary signals are also used. They are as follows:

- REQ_VLD - This signal is input to bus master. It is used by bus master for deciding HBUSREQ. HBUSREQ signal is asserted whenever REQ_VLD is asserted.
- WR - This signal is input to bus master. It indicates that write transaction shall take place. HWRITE shall be HIGH if WR is high.
- RD - This signal is input to bus master. If high, it indicates that read transaction shall take place and hence HWRITE shall be set LOW.
- LEN1 - This signal is input to bus master. It indicates that single transfer shall take place.
- LEN4 - This signal is input to bus master. It informs that the transfer should be a burst sequence of four transfers.
- LENX - This signal is input to bus master. It informs that the transfer should be a burst sequence of unspecified length.
- IN_ADDR[31:0] - These signals are input to the master providing information about address. These signals are used to decide HADDR.
- IN_DATA[31:0] - These signals are input to the master providing information about write data. These signals are used to decide HWDATA.
- LAST - This signal is input to bus master. It indicates the last transfer in a sequence of transfers.
- OUT_DATA[31:0] - These signals from the master provide information about read data.
- REQ_ADDR - This signal from the master is request for address. If this signal is high, in the next clock cycle, master shall receive IN_ADDR.
- REQ_WR_DATA - This signal from the master is request for data. If this signal is high, in the next clock cycle, master shall receive IN_DATA.

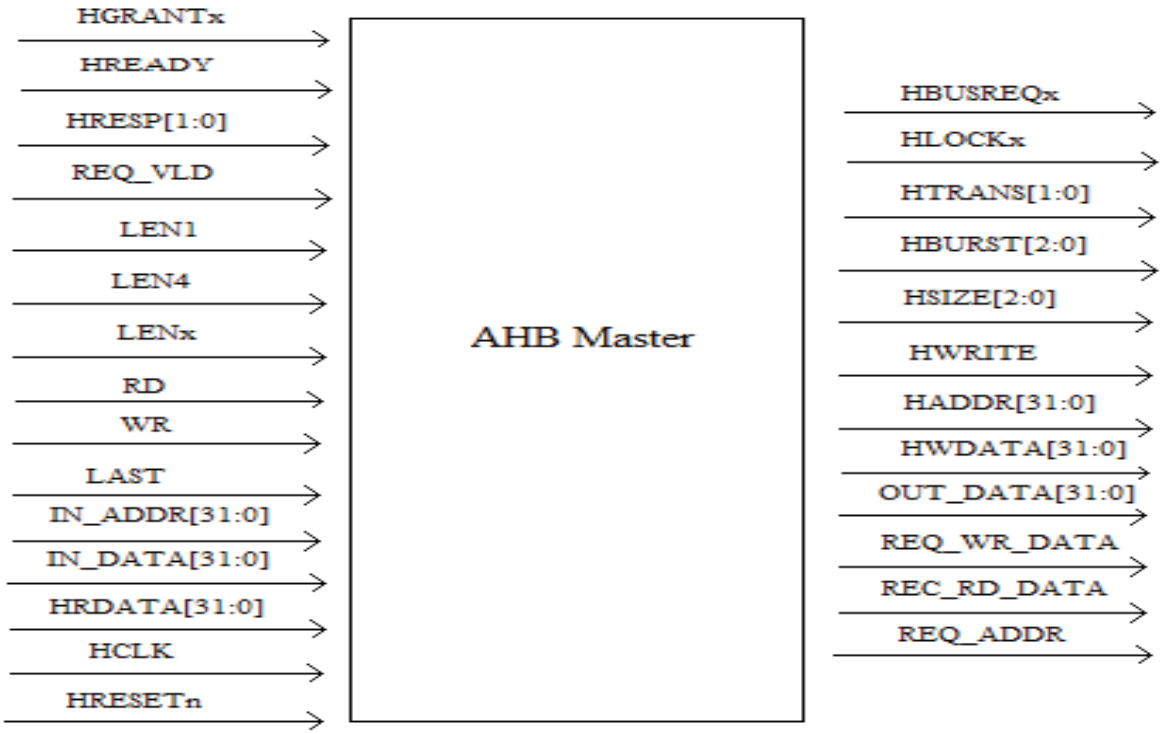


Figure 5: AHB Master

- REC_RD_DATA - This signal from the master provides acts as valid signal for read data. If it is high, HRDATA shall be copied to OUT_DATA.

Figure 5 shows signals for AHB Master and Figure 6 shows timing diagram for those signals.

## 5.2 Formal Specifications

In the formal specification of AMBA AHB Master, we have 10 assumptions and 15 guarantees.

**Assumptions** The assumptions are as follows.

**A1** Length of transfer will be specified with REQ_VLD signal i.e. whenever REQ_VLD is high, one of LEN1, LEN4 and LENX signal shall be high.

**A2** Nature of transfer will be specified with REQ_VLD signal i.e. whenever REQ_VLD signal is high, one of RD and WR signal shall be high.

**A3** If REQ_VLDsignal is low, RD, WR, LEN1, LEN4 and LENX shall hold their values.

**A4** There can not be conflict between signals indicating nature of transfer thus RD and WR signal can not be high simultaneously.

**A5** There can not be conflict between signals indicating length of transfer thus LEN1, LEN4 and LENX signals can not be high simultaneously.

**A6** Input HRESP signal shall be OKAY throughout.

**A7** The bus is fair one, hence every HBUSREQ shall eventually be answered.

**A8** During a locked unspecified length burst, leaving HBUSREQ high locks the bus. This is forbidden by the standard.

**A9** Eventually HREADY will be high.

**A10** We are not considering it as default bus master for the sake of generality. Hence eventually REQ_VLD and HGRANT signals will be low.

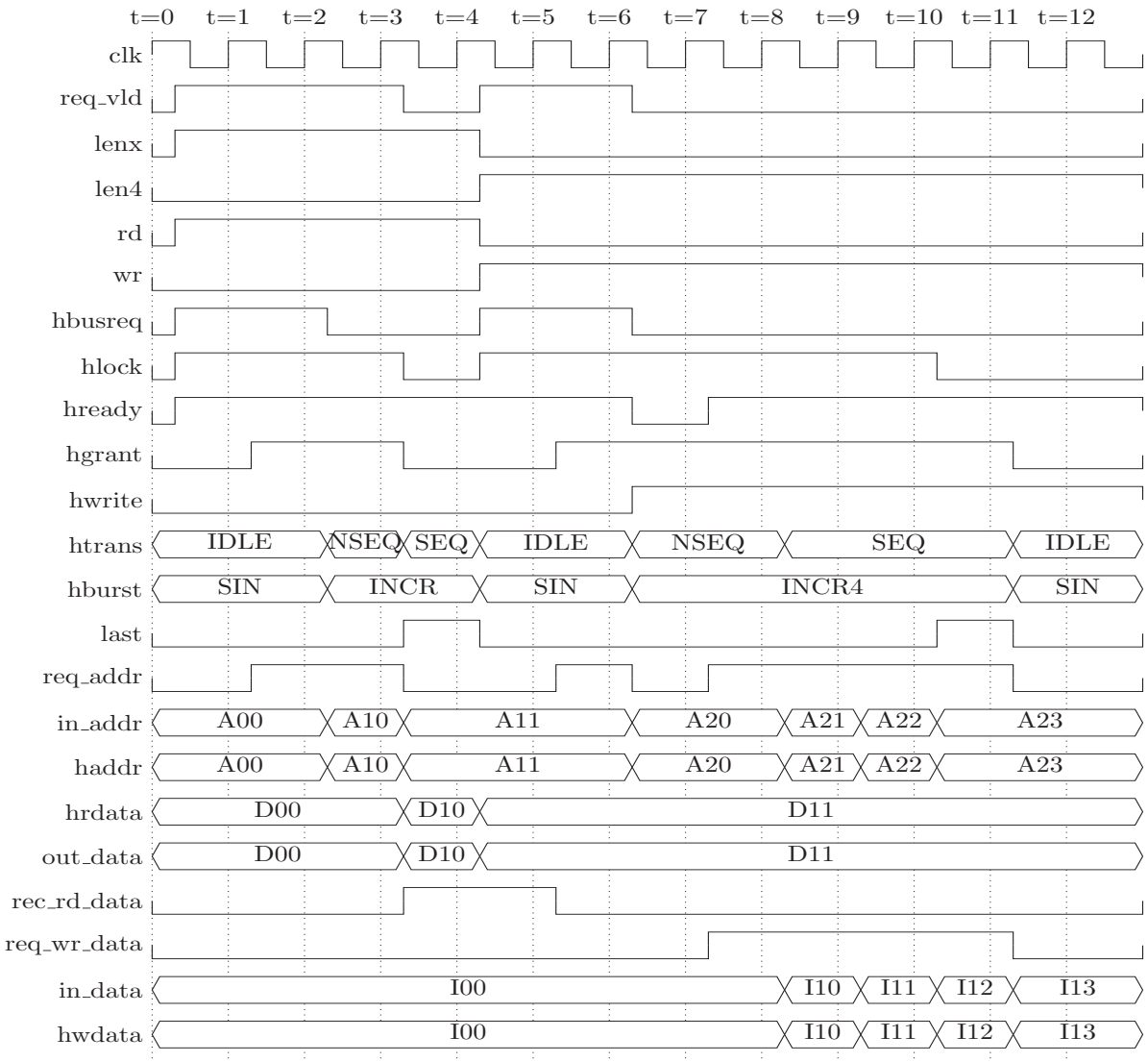


Figure 6: Signals for the AHB Master

We are assuming that data bus is 32-bit wide, hence HSIZE will be fixed to WORD. To make this bus master more general, another assumption is that this bus master requests for only locked transfers.

**Guarantees** The guarantees are as follows.

**G1** Data bus is 32-bit wide. Thus HSIZE shall be fixed to WORD throughout.
**G2** HBUSREQ signal gets asserted and de-asserted with REQ_VLD.
**G3** Bus master requests only for locked transfer.
**G4** If the ongoing transfer is last transfer of an ahb sequence, HLOCK shall be lowered.
**G5** Length four burst (HBURST = INCR4) shall end at fourth occurence of HREADY.
**G6** HBURST shall be set according to length of the transfer indicated by LEN1, LEN4 and LENX.
**G7** First transfer of an AHB sequence is always NONSEQ in nature. All following transfers in sequence shall be SEQ in nature.

**G8** Nature of transfer shall be set according to WR and RD signals.
**G9** If HREADY is low, all control signals shall hold their values.
**G10** When HREADY and HGRANT are simultaneously high, REQ_ADDR signal shall be high. It ensures that in next cycle, master can put address on address bus.
**G11** When both REQ_ADDR and WR signals are high, REQ_WR_DATA signal shall also be high. It ensures that data shall be put on data bus one cycle after address is put on address bus.
**G12** When a read transfer is taking place and HREADY is high, REC_RD_DATA signal shall also be high.
**G13** When REQ_ADDR is high, in the next cycle, incoming IN_ADDR shall be copied to address bus.
**G14** When REQ_WR_DATA is high, in the next cycle, incoming IN_DATA shall be copied to data bus.
**G15** When read transaction is in progress and HREADY is high, OUT_DATA shall copy the value of HRDATA.

### 5.3 Synthesis Results

The synthesis time for the AHB Master is 8.3 seconds. The generated circuit is mapped using ABC tool. It has 157 gates with area 210 square units. It is a very small circuit even with respect to manual implementations. Thus we are not only able to synthesize the AHB Master from its formal specifications, but the synthesized circuit is also very compact.

## 6 AMBA AHB Slave

In this section we present the synthesis results for AHB Slave.

### 6.1 AHB Slave Signals

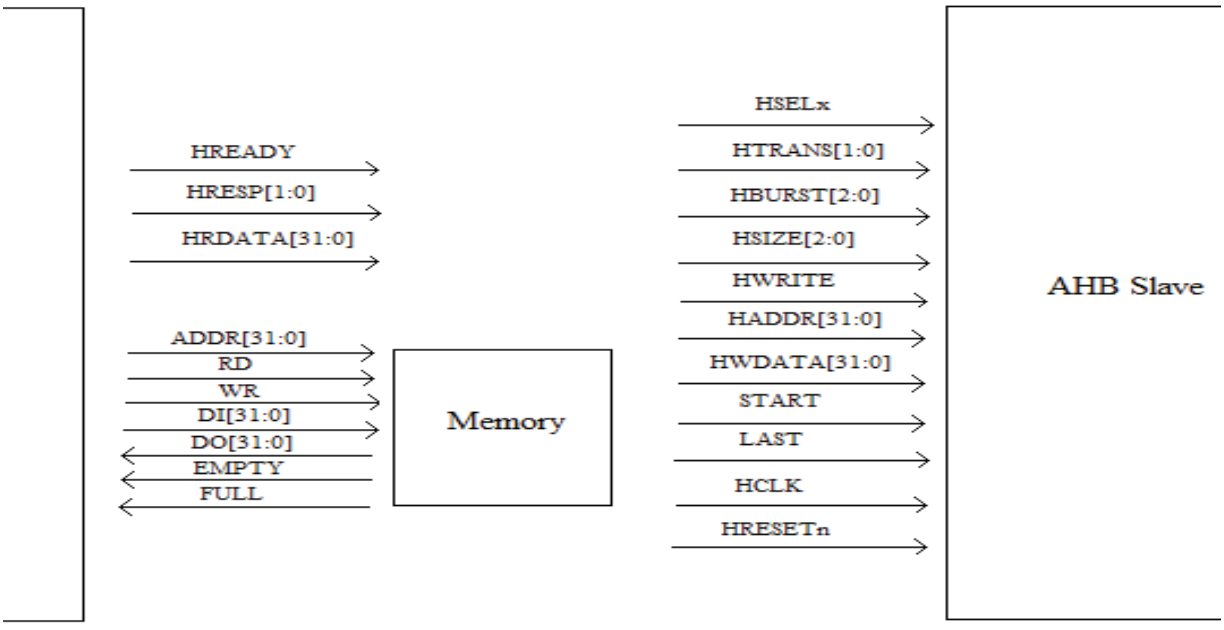


Figure 7: AHBSlave

The signals that are useful for AHB slave are already described in previous sections. We have introduced an interface between slave and a memory so that read and write transactions can be implemented. We are

Table 4: PSL Specifications for AHB Master

| | |
|---|---|
| A1 | always (REQ_VLD → (LENX ∨ LEN1 ∨ LEN4)) |
| A2 | always (REQ_VLD → (WR ∨ RD)) |
| A3 | always ((next ¬REQ_VLD) → (¬LEN1 ↔ (next ¬LEN1)))<br>always ((next ¬REQ_VLD) → (¬LENX ↔ (next ¬LENX)))<br>always ((next ¬REQ_VLD) → (¬LEN4 ↔ (next ¬LEN4)))<br>always ((next ¬REQ_VLD) → (¬WR ↔ (next ¬WR)))<br>always ((next ¬REQ_VLD) → (¬RD ↔ (next ¬RD))) |
| A4 | always (WR → ¬ RD)<br>always (RD → ¬ WR) |
| A5 | always (LENX → (¬LEN1 ∨ ¬LEN4))<br>always (LEN1 → (¬LENX ∨ ¬LEN4))<br>always (LEN4 → (¬LENX ∨ ¬LEN1)) |
| A6 | always (HRESP = OKAY) |
| A7 | always (REQ_VLD → eventually HGRANT) |
| A8 | always ((HLOCK ∧ (HBURST = INCR)) → next eventually ¬REQ_VLD) |
| A9 | always (eventually HREADY) |
| A10 | always (eventually (¬REQ_VLD ∧ ¬HGRANT)) |
| G1 | always (HSIZE = WORD) |
| G2 | always (REQ_VLD → HBUSREQ) |
| G3 | always ((¬HBUSREQ ∧ next HBUSREQ ∧ ¬HLOCK) → next HLOCK) |
| G4 | always (LAST → ¬HLOCK) |
| G5 | always ((HLOCK ∧ (HBURST = INCR4) ∧ HREADY ∧ (HTRANS = NONSEQ)) →<br>next ((HTRANS = SEQ) until_[3] HREADY)) |
| G6 | always (HBUSREQ ∧ HGRANT ∧ (HTRANS = IDLE) ∧ HREADY ∧ LEN1 → next (HBURST = SINGLE))<br>always (HBUSREQ ∧ HGRANT ∧ (HTRANS = IDLE) ∧ HREADY ∧ LENX → next (HBURST = INCR))<br>always (HBUSREQ ∧ HGRANT ∧ (HTRANS = IDLE) ∧ HREADY ∧ LEN4 → next (HBURST = INCR4)) |
| G7 | always (HBUSREQ ∧ HGRANT ∧ (HTRANS = IDLE) ∧ HREADY → next (HTRANS = NONSEQ))<br>always (¬LAST ∧ (HTRANS = NONSEQ) ∧ HREADY → next (HTRANS = SEQ))<br>always ((HTRANS = IDLE) → (HBURST = SINGLE)) |
| G8 | always (HGRANT ∧ (HTRANS = NONSEQ) ∧ HREADY ∧ WR → HWRITE)<br>always (HGRANT ∧ (HTRANS = NONSEQ) ∧ HREADY ∧ RD → ¬HWRITE) |
| G9 | always (¬HREADY → ((HTRANS = j) ↔ next (HTRANS = j)))<br>always (¬HREADY → ((HBURST = j) ↔ next (HBURST = j))) |
| G10 | always ((HREADY ∧ HGRANT) → REQ_ADDR) |
| G11 | always ((REQ_ADDR ∧ HWRITE) → REQ_WR_DATA) |
| G12 | always ((HREADY ∧ ((HTRANS = NONSEQ) ∨ (HTRANS = SEQ)) ∧¬HWRITE) → REC_RD_DATA) |
| G13 | $\forall i$ : always (REQ_ADDR → ((next ($\text{IN\_ADDR}_i$ = j)) ↔ (next ($\text{HADDR}_i$ = j)))) |
| G14 | $\forall i$ : always (REQ_WR_DATA → ((next ($\text{IN\_DATA}_i$ = j)) ↔ (next ($\text{HWDATA}_i$ = j)))) |
| G15 | $\forall i$ : always (HREADY ∧¬HWRITE ∧ ((HTRANS = SEQ)<br>∨ (HTRANS = NONSEQ)) → ((next ($\text{HRDATA}_i$ = j)) ↔ (next ($\text{OUT\_DATA}_i$ = j)))) |

considering memory with two status signals EMPTY and FULL.

Two auxilary signals have also been added named START and LAST. START signal indicates start of an AHB transfer or sequence whereas LAST signal is used to indicate last transfer of an AHB sequence.

The signals used in this interface are shown in Figure 7. Figure 8 shows the timing diagram from AHB slave signals. The description of signals used in interface between slave and memory is given below:

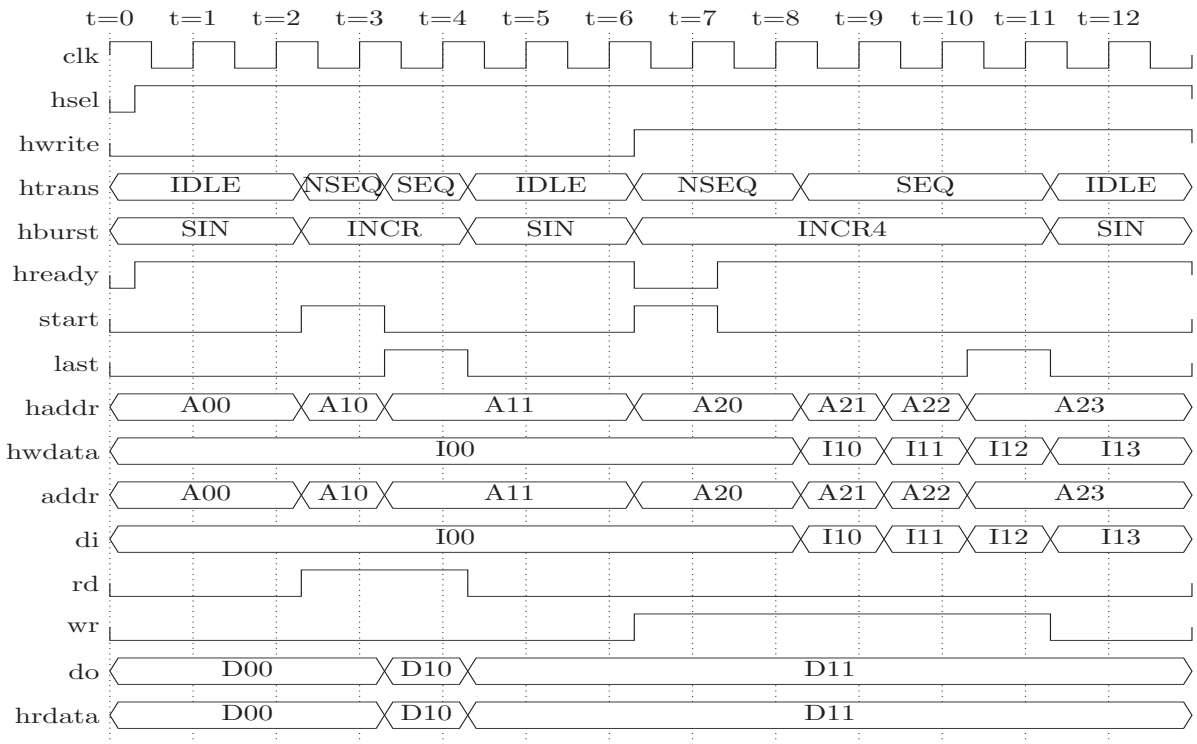


Figure 8: Signals for the AHB Slave

- FULL - This signal is input to bus slave indicating memory is full. No more data can be written into it without first being read.
- EMPTY - This signal is input to bus slave indicating memory is empty. No more data can be read from it without first being written.
- ADDR[31:0] - These signals are output from slave providing address information.
- DI[31:0] - These signals are output from slave and input to memory providing information about data that should be written into memory.
- DO[31:0] - These signals are output from memory and input to slave providing information about data that has been read from memory.
- RD - This signal is input to memory from slave. It indicates that the read operation is being executed.
- WR - This signal is input to memory from slave. It indicates that the write operation is being executed.

## 6.2 Formal Specifications

In the formal specification of AMBA AHB Slave, we have 7 assumptions and 9 guarantees. They are as follows.

**Assumptions** The assumptions are as follows.

**A1** When the slave is not selected by the decoder, all control signals shall be low.

Table 5: PSL Specifications for AHB Slave

| | |
|---|---|
| A1 | always (¬HSEL → ((HTRANS = IDLE) ∧ (HBURST = SINGLE) ∧ ¬HWRITE ∧ ¬START ∧ ¬LAST)) |
| A2 | always ((HTRANS = IDLE) → ((HBURST = SINGLE) ∧ ¬HWRITE ∧ ¬START ∧ ¬LAST)) |
| A3 | always (START → (HTRANS = NONSEQ)) |
| A4 | always (¬LAST ∧ (HTRANS = NONSEQ) ∧ HREADY → next (HTRANS = SEQ)) |
| A5 | always ((HLOCK ∧ (HBURST = INCR4) ∧ HREADY ∧ (HTRANS = NONSEQ)) → next((HTRANS = SEQ) until_[3] HREADY)) |
| A6 | always ((LAST ∧ next ¬START) → next (HTRANS = IDLE)) |
| A7 | always (¬HREADY →((HTRANS = j) ↔ next (HTRANS = j)))<br>always (¬HREADY →((HBURST = j) ↔ next (HBURST = j)))<br>always (¬HREADY →((HADDR = j) ↔ next (HADDR = j)))<br>always (¬HREADY →((HWDATA = j) ↔ next (HWDATA = j)))<br>always (¬HREADY →((DO = j) ↔ next (DO = j))) |
| G1 | always (¬HSEL → HREADY) |
| G2 | always (¬HSEL → (HRESP = OKAY)) |
| G3 | always ((HTRANS = IDLE) → (HRESP = OKAY)) |
| G4 | always ((WR ∧ HSEL) → ¬RD)<br>always ((RD ∧ HSEL) → ¬WR) |
| G5 | always ((HSEL ∧ FULL ∧ WR) → (HRESP = ERROR))<br>always ((HSEL ∧ EMPTY ∧ RD) → (HRESP = ERROR)) |
| G6 | always ((HSEL ∧ ((HTRANS = NONSEQ) ∨ (HTRANS = SEQ)) ∧ HWRITE) → WR)<br>always ((HSEL ∧ ((HTRANS = NONSEQ) ∨ (HTRANS = SEQ)) ∧¬HWRITE) → RD) |
| G7 | always ((HSEL ∧ ((HTRANS = NONSEQ) ∨ (HTRANS = SEQ)) → ((HADDR = j) ↔ (ADDR = j))) |
| G8 | always ((HSEL ∧ ((HTRANS = NONSEQ) ∨ (HTRANS = SEQ)) ∧ HWRITE) → ((HWDATA = j) ↔ (DI = j))) |
| G9 | always ((HSEL ∧ ((HTRANS = NONSEQ) ∨ (HTRANS = SEQ)) ∧¬HWRITE) → ((DO = j) ↔ (HRDATA = j))) |

**A2** When HTRANS is IDLE, all control signals shall be low.
**A3** First transfer of any sequence is NONSEQ in nature.
**A4** Non-first transfer of an AHB sequence will always be SEQ in nature.
**A5** Burst sequence of length four shall end at fourth occurence of HREADY.
**A6** If this is last transaction of a sequence and next cycle is not start of another sequence, HTRANS shall be IDLE in next cycle.
**A7** If HREADY is low, all control signals, address and data buses shall hold their values.

**Guarantees** The guarantees are as follows.

**G1** When the slave is not selected by the decoder, HREADY signal shall be high.
**G2** When the slave is not selected by the decoder, HRESP shall be OKAY.
**G3** When no AHB transaction is taking place, HRESP shall be OKAY.
**G4** RD and WR signal can not be high simultaneously.
**G5** If memory is full and write tranfer is attempted, slave shall send ERROR response. Similarly, if memory is empty and read transfer is attempted, slave shall send ERROR response.
**G6** When slave is involved in a transfer, HWRITE is used to decide values of WR and RD.
**G7** When slave is involved in any transfer, signal HADDR is used to decide ADDR.
**G8** When slave is involved in write transfer, signal HWDATA is used to decide DI.
**G9** When slave is involved in read transfer, signal DO is used to decide HRDATA.

### 6.3 Synthesis Results

The synthesis time for the AHB Slave is 21.5 seconds. The circuit generated, when mapped using ABC, has 214 gates with area 429 unit squared. It is a very small circuit even with respect to manual implementations. Thus we are not only able to synthesize the AHB Slave from its formal specifications, but the synthesized circuit is also very compact.

## 7 Lessons Learned

In the process of systematically re-writing the formal specifications for efficient synthesis, we learnt a few lessons about writing formal specifications for synthesis. We present these lessons with examples below.

- In the process of writing specifications, we should first simplify the design (if possible), write realizable specification for that can be synthesized efficiently for the simple design, and finally add necessary complexities to have the complete specification. For example, while writing AHB Master specifications, we first fixed all data and address signals width to one bit, synthesized the simpler design successfully and efficiently. This was followed by increasing data and address signal widths to 32-bit and adding necessary changes to AHB Master specifications to make it complete and synthesizable.
- While writing specifications, proceeding with the execution order of events is helpful. For example, while writing AHB Arbiter specifications, we proceeded with writing properties related to requesting access, granting access followed by properties related to AHB transfers.
- The use of auxilary signals is helpful in scenarios that cannot be emulated using only input output signals. For example, in AHB Slave specifications, we have introduced auxilary signals to emulate slave-memory interactions.
- The eventual specifications were the most time-consuming and difficult ones for synthesis and they need special attention.

In general, most data intensive applications are not reacive designs of degree one, and the above approach may not be ideal for those applications, but we believe that the above synthesis approach should work well for many control specific applications.

**Acknowledgment.** We thank Barbara Jobstmann for explaining the changes made in the specifications from [1] to [2].

## References

[1] Roderick Bloem, Stefan Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Martin Weiglhofer. Interactive presentation: Automatic hardware synthesis from specifications: a case study. In *DATE*, pages 1188–1193, 2007.

[2] Roderick Bloem, Stefan Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Martin Weiglhofer. Specify, compile, run: Hardware from psl. *Electr. Notes Theor. Comput. Sci.*, 190(4):3–16, 2007.

[3] J. Richard Buchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

[4] Alonzo Church. Logic, arithmetic, and automata. In *Proc. Internat. Congr. Math (Stockholm)*, pages 23–35, 1963.

[5] Cindy Eisner and Dana Fisman. *A Practical Introduction to PSL (Series on Integrated Circuits and Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[6] Barbara Jobstmann, Stefan Galler, Martin Weiglhofer, and Roderick Bloem. Anzu: A tool for property synthesis. In *CAV*, pages 258–262, 2007.

[7] ARM Ltd. Arm information center. `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.faqs/588.html`.

[8] ARM Ltd. Amba specification (rev. 2), 1999. `http://www.arm.com/products/solutions/AMBA_Spec.html`.

[9] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In *VMCAI*, pages 364–380, 2006.

[10] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL '89: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 179–190, New York, NY, USA, 1989. ACM.

[11] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989.

[12] Michael Oser Rabin. *Automata on Infinite Objects and Church's Problem.* American Mathematical Society, Boston, MA, USA, 1972.

[13] F. Somenzi. Cudd: Cu decision diagram package. University of Colorado at Boulder, `ftp://vlsi.colorado.edu/pub/`.

[14] Berkeley Logic Synthesis and Verification Group. Abc: A system for sequential synthesis and verification, release 61225. `http://www.eecs.berkeley.edu/~alanmi/abc/`.