AHRB: A High-Performance Time-Composable AMBA AHB Bus

Javier Jalle[†,*], Jaume Abella[†], Eduardo Quiñones[†], Luca Fossati[⋆], Marco Zulianello[⋆], Francisco J. Cazorla[†,‡]
[†]Barcelona Supercomputing Center, Spain
[*]Universitat Politècnica de Catalunya, Spain
[⋆]European Space Agency, Netherlands
[‡]Spanish National Research Council (IIIA-CSIC), Spain

***Abstract*—Hard real-time systems are moving toward complex systems comprising chips with different IP components connected with standard buses. AMBA is one of the most used bus interfaces and has already been included in processors in the real-time domain. However, AMBA was not designed to provide time composable *Worst Case Execution Time* (WCET) estimates, which are desirable to reduce timing validation and verification costs.**

This paper analyzes and extends the AMBA Advanced High-performance Bus (AHB) specification to enable time-composable WCET estimates by design. Concretely, (1) we analyze in detail the AMBA AHB in the context of hard real-time systems proving that it fails to provide time composability; (2) we define a restricted subset of AMBA AHB features, named *restricted AHB (resAHB)*, that allows deriving time-composable, yet not tight, WCET estimates; and (3) we define an extension of *resAHB*, named *Advanced High-performance Real-time Bus (AHRB)*, that includes the timing constraints in the specification. This allows deriving time-composable and tight WCET estimates. Our results show that AHRB can provide 3.5x tighter estimates than *resAHB* on average for EEMBC benchmarks.

I. Introduction

Computational demands in many Critical Real-Time Embedded System (CRTES) industries such as avionics, space, automotive and railway have experienced an unprecedented growth as a consequence of the need to cope with more sophisticated functionalities at software level. As the number and complexity of functions implemented in software in CRTES increases, achieving *guaranteed high-performance* is of paramount importance in all these markets. This has motivated the use of System-on-Chip architectures with high-performance processor features including cache memories and multicores (MPSoC).

In CRTES, software units, typically referred as tasks, are subject to a deadline and they are often characterized by a computed *Worst-Case Execution Time* (WCET) estimate as a means to provide an upper bound to their maximum execution time. However, the use of high-performance hardware features in CRTES, such as multicore architectures, challenges the computation of tight WCET estimates. The source of this complexity comes from the interferences when accessing hardware resources shared across the different tasks running simultaneously, called *inter-task interferences*.

CRTES increasingly rely on *composability* as a means to enable *incremental qualification*, which allows reducing verification and validation costs [8]. In this paper we focus on *time composability* that exists when the timing properties of a system software component in isolation, i.e. its WCET estimate, do not change when the component is combined with other components when the system is composed. In multicore execution environments, this means that the WCET estimate computed for a task is not affected by other tasks running simultaneously, hence being independent of the potential inter-task interferences when accessing shared resources. This requires bounding the access time of each request of any task to any shared resource, regardless of the load the other tasks in the MPSoC put on that resource.

One of the most important shared resources in current MPSoC for real-time systems is the backbone bus that connects the different cores with the memory/cache subsystem (and possibly other devices or subsystems). The Advanced Microcontroller Bus Architecture (AMBA) [9] is one of the most – if not the most – broadly used bus interfaces. AMBA is used in a wide range of architectures, providing flexibility in the implementation and backward-compatibility with existing AMBA interfaces.

This paper focuses on the Advanced High-performance Bus (AHB), one of the distinct buses defined in the AMBA specification, which aims at high-bandwidth, low-latency, high-frequency and low-complexity. AMBA AHB (or simply AHB) is increasingly being used in multicore processors for real-time industry, e.g. LEON3-based GR712RC [3] and LEON4-based NGMP [2], so providing tight and time-composable bounds to the access latency becomes a desirable property for AHB to deliver. Unfortunately, AHB was not designed with time composability in mind. In order to make AHB-based MPSoC systems to fulfill the desired time composability property, this paper makes three contributions:

Contribution 1. We provide a detailed analysis of the AMBA AHB features and their impact on the timing behavior of connected components (i.e. master and slaves), and by inference, on the applications running on the MPSoC. We identify the AHB features that affect the timing behavior of applications and how AHB-compliant masters and slaves can break time composability.

Contribution 2. We propose to use only a restricted subset of the AHB features, named restricted AHB (*resAHB*), such that if master/slaves adhere to it, the maximum delay that any bus access may suffer due to inter-task interferences can be computed, so fulfilling the time-composability property. The main advantage of *resAHB* is that it is compliant with the AMBA AHB specification, providing the same functionality as the original one. Our results show that this solution is attractive only when the AHB-connected components have a considerably higher latency than the AHB arbitration itself.

Contribution 3. We extend *resAHB* by introducing a new set of *operation modes* currently not specified in the AMBA AHB specification. We call this new AHB specification, *Advanced High-performance Real-time Bus* specification or *AHRB*. AHRB efficiently isolates the timing behavior of the different components connected to the AHRB, allowing to derive tight and trustworthy WCET estimates.

AHRB specification ensures that if all connected compo-

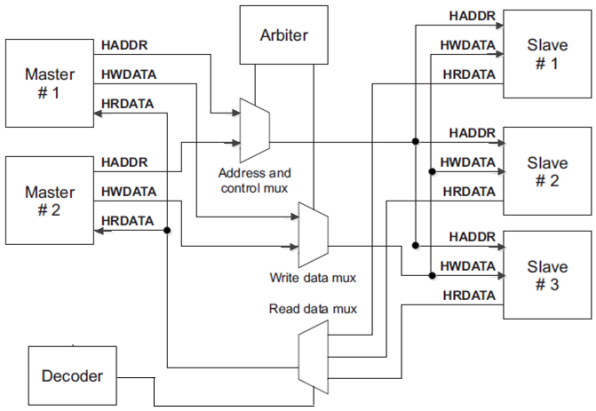


Fig. 1. AMBA AHB main components: Masters, slaves, arbiter and decoder. (Picture from AMBA Specification Rev 2.0)

nents follow it, they will enjoy tight and time-composable bounds for their bus access latency. As a result, although the timing behavior of any IP component[1] may be unknown, its effect on the bus is bounded under *AHRB* without any information or requirement from the component at hardware and software levels because timing requirements are already included in the specification. This is of paramount importance in the context of future multi-IP mixed-criticality MPSoC.

AHRB extends AHB by adding *master and slave modes* that allow specifying the use of the bus that each master and slave does, in such a way that it can be taken into account by other master/slaves at design time. This results in tighter (and bound) access times to the bus, which in turn leads to tighter WCET estimates for the applications running on the MPSoC.

Our results show that, unlike the original AHB, both *resAHB* and *AHRB* enable obtaining time-composable WCET estimates for all tasks running in the MPSoC. Those estimates are independent of the other tasks being run simultaneously. WCET estimates resultant of using *AHRB* are tighter than those for *resAHB*, reducing WCET estimates by 3.5x on average. We also observe that in our setup inter-task interferences cause an average performance degradation on the EEMBC benchmarks ranging from 6% to 35% depending on the workload when using a conventional AHB bus. Such degradation reduces with *resAHB* and *AHRB* to 1%-5%.

The rest of the paper is organized as follows: Section II describes AMBA and introduces time composability. Section III analyzes the effects of AMBA AHB features on timing. Sections IV and V describe *resAHB* and AHRB respectively. Section VI shows the experimental results. Sections VII and VIII present the related work and the main conclusions of this study. Annex I provides examples of how AMBA challenges time composability. Annex II shows an example of how the master and slave modes work. Finally, Annex III covers other AMBA specifications.

II. Background

A. The Advanced Microcontroller Bus Architecture: AMBA

AMBA is a standard bus interface for high-performance embedded microcontrollers, aimed at high-bandwidth, low-latency, high-frequency and low-complexity on-chip communication. Several studies have shown that hierarchical buses

[1]An IP (intellectual property) component is a block of logic or data that is designed to be ported and reused across different products (ASIC or FPGA).

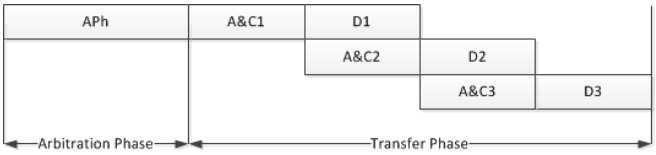


Fig. 2. AHB transactions consist of an arbitration and a transfer phase. The latter is divided into beats with an address and control phase and a data phase.

scale, in terms of performance and energy consumption, to systems with processor counts in the range 32-64 cores [20] [22]. Thus, AMBA is expected to remain in the near future as one of the standard bus interfaces for real-time MPSoCs.

The AMBA specification [4] defines three distinct buses: the *Advanced High-performance Bus* (AHB), the *Advanced System Bus* (ASB) and the *Advanced Peripheral Bus* (APB)[2]. In this paper we focus on the AHB, which is used in several existing architectures such as the Aeroflex Gaisler GR712RC and the NGMP (Next Generation MicroProcessor). AHB has been designed to be a high-performance backbone bus that efficiently connects cores, on-chip memories and IP components[3]. The AHB basic architecture, shown in Figure 1, comprises four different components: a set of *masters*, a set of *slaves*, an *arbiter* and a *decoder*. These components use or control three main buses: the address and control (HADDR in the Figure), read data (HRDATA) and write data (HWDATA) buses.

Figure 2 shows an AHB *transaction* between a master and a slave. A master initiates a read/write transaction to a slave by requesting to the arbiter the access to the address and control bus. During the *arbitration phase*, the arbiter handles the contention across masters by granting access to the bus only to one master at a time according to a predefined arbitration policy (not defined by the AMBA specification), so that concurrent transfers are not allowed. Once a master is granted access to the bus by the arbiter, the *transfer phase* starts. The transfer is split into several *beats* if data cannot be sent through the bus all at once (i.e. *burst transfers*). For instance, the transfer in Figure 2 requires three beats. Every AHB transfer beat consists of two subphases that overlap across beats, as defined in the specification: an *Address-and-Control phase* that lasts for one cycle in the absence of contention, and a *Data phase* that lasts one or more cycles (in case the master/slave cannot provide the data at that moment). The decoder controls the multiplexers to send the address to the appropriate slave, and two data buses are used to send/receive data to/from the slave. When the transaction finishes, the master relinquishes the buses.

B. Time Composability

Several industries such as automotive, avionics and space industries have moved toward an *Integrated Architecture* paradigm: A modular approach in which multiple functions can be assigned to a single hardware unit. Examples of integrated architectures are the Integrated Modular Avionics (IMA) [23] in the avionics domain, the Automotive Open System Architecture (AUTOSAR) [12] [7] in automotive or IMA for spacecraft [25] for the space domain.

[2]AMBA 3 specification also includes *Advanced eXtensible Interface* (AXI), an interconnection protocol independent from the interconnection network topology used. It is later covered in Section VI.

[3]In a multicore based real-time system, in addition to the on-chip interconnection network, we find other networks for the communication of boards. For instance, FlexRay in automotive or AFDX in avionics. This paper focuses on on-chip communication and, in particular, on AMBA.

A key design principle for integrated architectures is the *incremental qualification*, whereby each software component is verified and validated in isolation, including functional and timing analysis, such that no misbehaving function may corrupt other components. At timing level, which is the focus of this paper, this requires time composability such that the timing behavior of a function is not affected by the execution of other functions. Time composability determines that the timing behavior of an individual component does not change in the face of composition when the system is integrated, and so the timing analysis performed in isolation remains valid at system integration. Time composability therefore reduces the cost of system integration and qualification, which is one of the most critical challenges faced by system developers.

III. AMBA AHB and Time Composability

As MPSoCs integrate an increasing number of IP blocks coming from different suppliers and the functionality provided by MPSoCs keeps diversifying, the trend towards MPSoC comprised of multi-party IPs will further exacerbate. Moreover, it is also the case that real-time system IPs may be subject to different safety and security levels. Hence, in order to enable *mixed-criticality functionalities* to be run on the same MPSoC, the impact that one IP component can create on the timing behavior of the others must be limited and this cannot be left to the IP provider, especially to those subject to less-restrictive criticality levels. The safety of the integrated system in terms of timing behavior should be provided by design (construction), in our view, and thus by the bus protocol specification itself. Since time composability is the central element for reducing timing verification and validation costs, it should be provided by design.

Time composability imposes that *the maximum time any request of any task waits to be granted access to the bus is bounded and the bound does not depend on the particular co-running tasks in the MPSoC* [15][16]. This requires that (1) the arbitration phase for every transaction is bounded and this bound does not depend on the behavior of other components; and (2) the duration of the transfer phase of every transaction is also bounded. Note however, that the arbitration-time and transfer-time bounds may depend on hardware implementation details, such as the number of cores contending for the bus. This however does not jeopardize time composability, since those features are known at design time, when WCET estimates are computed for each task.

If every access to hardware shared resources fulfills the time composability property defined above, WCET estimation for a task, either with static timing analysis or measurement-based techniques [24], is independent of the accesses that other co-running tasks may do on the same hardware shared resources. This effectively enables the computation of WCET estimates for each task in isolation, bringing the benefits of reducing time validation and verification costs as explained in the previous section. Unfortunately, at the time AHB was released, time composability was not one of its design goals. As a result, AHB-compliant master/slaves lead to non time-composable behavior that makes difficult analyzing the timing properties of an AHB-based MPSoC. In *Annex I*, we illustrate some non time-composable behavior of AMBA AHB through an example.

In this section we review AHB features and classify them according to their effect on timing. Table I summarizes the features we analyze.

TABLE I. List of AHB features analyzed

ID	Feature	Breaks TC?	Affects WCET bounds?
1	Number of Masters	No	Yes
2	Handover	No	No
3	Back-to-back	No	No
4	Burst operation	Yes	Yes
5	Flow control	Yes	Yes
6	Split transactions	No	Yes
7	Locked transfers	Yes	Yes
8	Bus width	No	Yes
9	Protection control	No	No
10	Error response	No	No
11	Retry response	No	No
12	Idle transfers	No	No
13	Early burst termination	No	No
14	Arbiter	Yes	Yes

1) *Number of masters*: AHB allows up to 16 bus masters contending in one bus. The number of masters does not break time composability because it is a feature known at design time, like the number of cores in a multicore platform. However, it affects the tightness of WCET estimates. In real-time scenarios, the worst-case situation to consider happens when, on the event of a master trying to get access to the bus, all the other masters want to access the bus the same cycle, all of them having higher priority.

2) *Handover*: Changing the ownership of the bus from one master to another is called handover. AHB has a one-cycle master handover, so a master being granted access to the bus in a given cycle, gets the bus in the next cycle, so the minimum arbitration time is one cycle. This effect can be taken into account for each request of a task by adding this extra cycle of delay to the arbitration time. This feature is time-composable and has negligible effect on timing tightness.

3) *Back-to-back execution*: Back-to-back execution occurs when two transfers from different tasks are executed consecutively one after the other, without any idle cycle in between. This is feasible because of the pipelined execution of AHB, where address and data phases of different transfers overlap. This feature has neither an effect on timing nor on time composability.

4) *Burst operation*: Burst operation allows to perform transactions composed of more than one beat. AHB allows to have undefined-length bursts, 4-beat, 8-beat or 16-beat burst transactions. Undefined-length burst can be of any length, although its limit is constrained by the fact that the address cannot cross a 1 KB boundary. The burst length affects WCET estimates since, to compute them, it is needed to assume always the maximum burst length allowed in the system, even though it might be too pessimistic to be useful.

5) *Flow control*: Flow control in AMBA AHB can be performed by both the master and the slave. The slave can extend the data phase of a transfer by inserting *wait states* if, for instance, it needs extra time to process the transaction. Similarly, the master can insert *busy* transfers with the same purpose of extending the time between data transfers.

The original AHB specification states that the number of wait states is limited, but it does not set any specific limit. The absence of a specific limit breaks time composability because it eliminates the possibility of computing how long a transaction can take, so how many other tasks in the system can be delayed because of such transaction. Further note that both, wait states and busy transfers, increase WCET estimates since for every transaction, the maximum number of wait states and busy transfers must be assumed.

6) *Split transactions*: Split transactions provide a mechanism for slaves to release the bus when they need some more time to respond. This allows other masters to get access to the bus rather than waiting for the slave to finish. When a slave signals a split transaction, the arbiter masks the request of its corresponding master until the slave indicates that the response is ready, so the master is considered again in the arbitration. Note that *the master has to wait again for arbitration*, and only when the arbiter grants the bus access, the slave can respond to the request. Signaling a split transaction needs a two-cycle response. This effectively adds two cycles to the size of a transfer for WCET estimates.

Therefore, a split transaction only affects the timing behavior of the master that received it, as two arbitration phases occur: the first one when the request is issued, and the second one when the slave is ready to respond. This is not the case for the rest of masters, in which a split transaction is seen as two independent transactions. It is important to remark that, although the master can keep the slave component busy during a split transaction, thus affecting others, this contention is not because of the bus, but because of the slave.

7) *Locked transfers*: Locked transfers allow a master to keep the ownership of the bus until the locked sequence has been completed. Locked transfers ensure that a transfer is done without disturbance, which is necessary, for instance, in the case of read-modify-write requests to an AHB connected cache or memory device. However, locked transfers can be very harmful for time composability, because a master can issue a locked transfer and keep the bus busy indefinitely, thus affecting other masters. Section IV-A provides a discussion of how the functionality provided by locked transfers can be implemented avoiding the issue of affecting the other tasks in a timing unpredictable way.

8) *Bus width*: AMBA AHB allows different bus widths: 8, 16, 32, 64, 128, 256, 512 and 1024 bits-wide buses. It is recommended a minimum of 32 bits. This feature does not affect time composability but it has an effect on timing, because the wider the bus, the fewer the number of transfers needed. Note that the bus width is known at design time, like the number of masters.

9) *Protection control*: AMBA AHB has protection control signals that provide information about a bus access, to be used by any module that implements some level of protection. These signals indicate whether the access is instruction or data access; user or privileged access; bufferable and cacheable. Since no functionality is attached to these signals, because it is something optional and for higher level protocols, it is completely harmless for time composability.

10) *Error response*: A slave can respond to a transfer with an *error*, to indicate that something went wrong. Although errors may break timing behavior of a task, it is a responsibility of the master-slave higher level communication protocols to take care of errors and it does not affect other masters. Hence, it does not affect time composability. Errors require two extra cycles for signaling, as in the case of *split transactions*.

11) *Retry response*: A slave can respond also with a *retry* response, to indicate the master to perform the transfer again. Like *error responses*, retries may also break timing behavior of a task, but this is again responsibility of the master-slave higher level communication protocols and does not affect other masters. Hence, it does not affect time composability. Retries also require two extra cycles for signaling, as in the case of *error responses* and *split transactions*. *Retry responses* can also be used as an alternative to *split transactions* when the slave is unable to provide the response (e.g., due to its high latency). The difference is that with *retry response*, the normal arbitration priority scheme will be maintained. With *split transactions*, the arbiter masks the request of the split master until the slave indicates that the response is ready, which improves performance, since that master will not be granted access unless the response is ready.

12) *Idle transfers*: Idle transfers are used to indicate that no data transfer is required. AMBA uses a default master when all other masters are unable to use the bus. When granted, the default master must only perform *idle transfers*. *Idle transfers* can also be used in case a master cannot continue a burst. It is completely harmless for time-composability.

13) *Early burst termination*: Early burst termination is a mechanism that allows slaves to detect when a burst transfer is incomplete. If during a burst transfer, a slave detects an idle transfer or a non-sequential transfer, it means that the previous transfer finished before it was completed. This may occur if the arbiter changes the ownership of the bus, or if the master cannot finish the burst. This feature is completely harmless for time composability.

14) *Arbiter*: AMBA does not define any restrictions on the arbiter, which means that any arbitration policy can be used. This can completely break time composability as we show in the example in Annex I. The arbitration policy affects the timing behavior of all masters, because it defines how much time a master has to wait to be granted access to the bus, which indirectly depends on the other masters' behavior.

In summary, in order to achieve time composability several features of AMBA AHB must be either limited or disabled. In particular, unrestricted locked transfers must be avoided, only time-composable arbitration policies must be considered, and burst length, wait states and busy cycles must be limited.

IV. Restricted Time Composable AHB: resAHB

As shown in previous section, AMBA AHB specification does not guarantee time composability at bus transaction level. This section defines a restricted usage of a subset of the AHB features, such that if every master and slave follow this restricted AHB specification (*resAHB*), time composability can be *guaranteed by construction* at transaction level. The main goal of *resAHB* is to derive tight upper bounds for every bus transaction, regardless of the actual behavior of other master/slaves components, while keeping AMBA AHB functionality.

In order to achieve time composability, the AHB features that must be considered are burst length, wait states and busy cycles (i.e. flow control), locked transfers and the arbitration policy.

Burst length: AHB allows defining burst sizes of 4-beat, 8-beat, 16-beat and undefined length. Undefined burst sizes break time composability as they prevent bounding the impact of a bus transaction on other transactions belonging to different tasks. Therefore, we enforce maximum burst length to be 16 beats, which will be compatible with existing AHB components. Components needing bus transactions larger than 16 beats must split the transaction in several transactions. It is important to remark that the burst sizes of undefined length can still be set assuming any length smaller than 16 beats. If the burst length of an undefined transaction is above 16 beats, the corresponding master or slave will have to split the transaction.

Flow control: Flow control defines the number of *wait states* and *busy transfers*. As we have seen in the previous section, the original AHB specification states that the number of wait states has to be limited, recommending not to insert more than 16 wait states but it does not specify any limit. In order to provide time-composable AHB transactions, we restrict the number of wait states to 16, which is the maximum recommended by the specification and corresponds to 16 bus cycles. In case a slave needs more than 16 wait states, it can signal either a *split transaction* or a *retry* (in case the resource is busy).

On the master side, busy transfers need also to be limited. Similar to wait states, we enforce masters not to introduce more than 16 busy transfers, which corresponds to 16 bus cycles. If a master needs more than 16 busy transfers, e.g. it cannot continue the current transfer, it will have to signal an *early burst termination* which releases the bus. The transaction can then be performed once the master is ready.

It is worth noting that both, wait states and busy transfers, introduce pessimism in WCET estimates as the worst-case scenario must be considered, i.e. every transfer incurs 16 wait states and busy cycles.

Arbiter. The AMBA AHB specification does not put any restriction on the arbitration; this is not suitable for time composability, so *resAHB* constraints the timing behavior of the selected arbiter.

On the one hand, *resAHB* restricts the arbitration policy to those that allow to derive upper bounds on the time a master needs to wait to be granted to use the bus, e.g. TDMA (*Time Division Multiple Access*) and Round-robin [15]. For example, under round-robin arbitration policy, in the worst case a master waits $N-1$ rounds of arbitration before it gets the bus, where N is the number of masters. Hence, the longest arbitration latency a master suffers due to inter-task interferences is bounded by $Bound_{RR} = (N-1) \times (t_{tran} - 1)$, where t_{tran} is the bus transfer time from which one cycle is subtracted because in consecutive transfers address and data phases overlap. Any other arbitration policy that allows to derive time composable upper bounds on the arbitration time can be considered.

On the other hand, the arbiter must not change bus ownership during a transaction. That is, under *resAHB* a master cannot be preempted once it has been granted access to the bus.

Locked transfers. Under *resAHB*, locked transfers are not allowed to take longer than the maximum transaction latency, that is 50 cycles as shown in Section IV-B. Locked transfers are used to eliminate disturbance when doing an access or a sequence of accesses. If the accesses can be served with a single transfer, locked transfers are not needed, since, as pointed above, bus transactions are not preempted, i.e., other masters cannot use the bus until the current transaction finishes. In the case of a sequence of accesses that require more than one transfer, e.g. read-modify-write accesses, the atomicity provided for individual transfers does not suffice to provide the same functionality as *locked transfers*. In Section IV-A we present a mechanism to provide this functionality without requiring locks.

A. Providing the same functionality as AHB

resAHB restricts the use of locked transfers to avoid masters to lock indefinitely the bus, which would simply kill time composability. However, locked transfers provide a functionality that may be required by masters. For instance, cores (masters) may want to perform an atomic operation on a shared memory to enable the use of higher level locking protocols, like the priority ceiling protocol [18], for which masters use locked transfers. In order to be able to maintain this functionality, we propose two different approaches.

The first approach benefits from the fact that under *resAHB* the arbiter cannot relinquish the access to a master once it is granted access. Hence, if the sequence of accesses requiring an atomic operation (e.g *rmw*) fits within the maximum possible transaction length (i.e. 50 cycles) it will be carried out correctly. This is normally the case, since (1) at software level locking time is reduced as much as possible to improve performance and (2) locked transfers are usually issued to cache memories which have short latency. Note that the first time the master asks for a data with an 'atomic' request (which can be identified using the *HMASTLOCK* signal), the slave has to fetch it, which can take longer than 50 cycles. In that case the slave simply responds with a split or retry response and starts fetching the data. Once the datum is cached, the atomic operation can be carried out in less than 50 cycles.

The second approach consists in moving some functionality to the slave component (e.g., a cache). For instance, in the case of the SPARCv9 architecture, *rmw* operations can be implemented with the *compare-and-swap* (CAS) operation that works as follows:

```
int64 CAS (int64 *word, int64 test_value, int64 new_value )
{
        int64 old_value;

        atomic {
            old_value   = *word;
            if ( *word == test_value )
                *word= new_value;
        }

        return( old_value );
}
```

The core (master) sends a read operation of address $*word$ to the cache. While handling this request, the cache must prevent any other request from accessing $*word$. Once the core receives the answer, i.e. the data in $*word$, it relinquishes the bus. Then the core compares the content of $*word$ with the *test value* provided in the CAS operation. In case of match, the core starts a new transfer on the bus to write the *new value* to $*word$. Otherwise, if there is no match, the core starts a new transfer to write the *old value* to $*word$. The cache will not accept new accesses to $*word$ until the write operation is served. Note that the AHB signal *HMASTLOCK* indicates the slave that the transfer is a locked transfer, allowing the slave to be aware of locked transfers.

In any case, the only restriction is that any locked transaction cannot exceed the maximum transaction length.

B. Deriving bounds to access latency

The *resAHB* specification allows deriving upper bounds on the time an AHB bus transaction takes. The upper bound is defined as $\tau = t_{arb} + t_{tran}$, where t_{arb} is the longest time it takes a master to be granted access to the bus since the cycle in which it requests access to the bus. t_{tran} is the transfer time, the longest time a master is entitled to use the bus once it is granted access by the arbiter.

t_{arb} depends on the arbitration policy, the number of masters and the maximum time each master can use the bus

once it is granted access, i.e., t_{tran}. For instance, for round-robin this time is $t_{arb} = 1 + (N-1) \times (t_{tran} - 1)$, which corresponds to the bound of round-robin plus one extra cycle needed for the initial *handover* [4]. t_{tran} is given by the longest possible transfer that corresponds to a 16-beat burst transfer with 16 wait states and 16 busy transfers. It also includes two extra cycles for signaling an error/retry/split response. This totals 50 bus cycles*, regardless of the specific hardware and software details of master and slave components*. Note that t_{tran} is decreased by 1 cycle since data and address phases across transactions overlap in 1 cycle.

For instance, in a 4-master bus with round-robin policy, the maximum, time-composable arbitration time is $t_{arb} = 1 + (4-1) \times (50-1) = 148$. By computing the WCET estimate for a task assuming this arbitration time, will make its WCET estimate independent of the other tasks it runs with. This is so, because the worst effect that other tasks can cause on the bus is already taken into account by the maximum arbitration time.

V. Advanced High-performance Real-time Bus

AMBA AHB can be made time composable by construction by restricting the use of some AHB features. Though this is an attractive proposition since it keeps compatibility with AMBA, it leads to pessimistic WCET estimates. For instance, we have seen that with 4 masters and round-robin arbitration policy, the arbitration time that a given request has to assume in order to be time composable is as big as 148 cycles.

In this section we extend the AHB specification in the form of extra features that improve AHB time composability properties. Obviously these extended features make the new specification non AMBA-compliant. We call our AHB specification extension AHRB *Advanced High-performance Real-time Bus*.

A. Master and Slave modes

The focus of AHRB is on reducing τ which in turn requires reducing t_{arb}, the arbitration time, in order to produce tighter WCET estimates. t_{arb} directly depends on t_{tran} which is the maximum allowed transfer length, which was 50 cycles under *resAHB*. t_{tran} depends on the duration of (1) the burst length, (2) the wait states and (3) the busy transfers that are allowed to be inserted. Busy transfers and burst length are introduced by the master and depend on the amount of data to transfer (e.g., a cache line) and the master internal behavior; Meanwhile, wait states are introduced by the slave and are used to cover the latencies of the slave components.

If t_{tran} could be determined taking into account the specific timing requirements of the particular masters and slaves using the bus in terms of their burst lengths, wait and busy states, the value of t_{tran} would certainly reduce with respect to the upper bound value used under *resAHB*. For instance, let us assume a 4-core processor with a shared L2 cache, 16-byte cache lines and a 128-bit wide bus, so that each transfer needs 1-beat to send a whole cache line. L2 cache hit latency is 4 cycles. In this scenario, by defining a maximum transfer length of 1-beat plus 4 *wait states* to allow the cache to serve the access, we cover all L2 hit accesses with one transaction with the minimum required length. The maximum transfer time will be $t_{tran} = 1 + 4 + 1 + 1 = 7$ cycles[5], and the maximum arbitration time under round-robin policy $t_{arb} = 1 + (4-1) \times (7-1) = 19$ cycles. These values are less pessimistic than the ones we would obtain with *resAHB*, i.e, $t_{arb} = 148$, which significantly improves worst-case performance. In case of a L2 miss, the L2 cache signals a *split transaction*, spends all the required latency to bring the data and pays another arbitration penalty whenever the response is ready. However, if the access to main memory takes in the order of dozens or even hundreds of cycles, the relative overhead of two arbitration rounds is low. Moreover, the benefit of issuing a *split transaction* for L2 misses is that the bus is free while the miss is being resolved, enabling other masters to use it. Conversely, putting a lower limit on the number of *wait states*, for example only 2, would make that any hit in the L2 signals a split transaction, since the hit latency is at least 4 cycles. This would mean that any L2 hit access would have to perform 2 arbitrations, the regular one and an extra one when the L2 has the data ready to be sent, which would deteriorate performance, because hits are frequent. In Section VI-E we show this behavior with experimental results.

Overall, adapting t_{tran} to the specific needs of the masters and slaves reduces the overhead required to achieve time composability. The challenge lies on defining in the bus specification a mechanism that allows masters and slaves specifying their needs in the use of the bus. To that end, we introduce the concept of *master and slave modes*, which we will call simply *modes*. In the AHRB specification we define modes from 1 to 32, according to the number of busy transfers and burst beats in the case of the master (up to 16 each), and the number of wait states in the case of the slave (up to 16).

The arbiter assigns a master and slave mode when granting the bus to a master that forces the master and the slaves to operate in that mode. Under each master and slave mode of operation, we can easily derive tight bounds for t_{tran} and hence t_{arb}. For instance, a master i with *master mode 4* ($mm(i) = 4$) will insert at most n-beat bursts and m busy transfers where $n + m \leq 4$, and slaves with *slave mode 4* ($sm(i) = 4$) will insert at most 4 wait states, plus the two extra cycles for signaling an error/retry/split response:

$$t_{tran}(i) = mm(i) + sm(i) + 2 = 4 + 4 + 2 = 10 \quad (1)$$

Under round-robin deriving the maximum arbitration time for a master i, $t_{arb}(i)$, is straightforward. $t_{arb}(i)$ is the addition of $t_{tran}(k)$ for every master $k \neq i$, i.e., with round-robin policy:

$$\begin{aligned} t_{arb}(i) &= 1 + \sum_{\substack{k=0 \\ k \neq i}}^{N-1} (t_{tran}(k) - 1) \\ &= 1 + \sum_{\substack{k=0 \\ k \neq i}}^{N-1} (mm(k) + sm(k) + 1) \end{aligned} \quad (2)$$

B. Deriving tight WCET estimates

For each task a WCET estimate is computed under (1) each of the modes of the master and slaves that the task uses and (2) each of the modes the other masters/slaves may have. For a task k running on master i, under each combination of (1) and (2) a different WCET estimate is obtained, $WCET_k^{t_{arb}(i), t_{tran}(i)}$.

[4] The worst-case corresponds to every master accessing the bus at the same cycle which requires an initial cycle for handover. The rest of handovers are not consuming any extra cycle because of the back-to-back execution of requests.

[5] 1 cycle to send the address, 4 cycles to read the data from L2, 1 cycle to transfer the data, and 1 cycle in case the slave asks for retry or signals an error.

TABLE II. $t_{tran}(0)$-$t_{tran}(1)$-$t_{tran}(2)$-$t_{tran}(3)/t_{arb}(0)$ UNDER DIFFERENT MASTER AND SLAVE MODES IN A 4-MASTER CONFIGURATION. $t_{arb}(0)$ IS FOR MASTER 0 AND $t_{tran}(i)$ FOR MASTER i.

	Master modes			
slave mode	1-1-1-1	1-1-1-4	1-1-4-4	1-4-4-4
2	5-5-5-5/13	5-5-5-8/16	5-5-8-8/19	5-8-8-8/22
4	7-7-7-7/19	7-7-7-10/22	7-7-10-10/25	7-10-10-10/28

For instance, Table II shows t_{tran} and t_{arb} for an example with 4 masters and a slave where different modes are assumed for each master and slave. In particular we use 2 modes for the masters, 1 and 4, and modes 2 and 4 for the slave. t_{tran} and t_{arb} are computed with Equations 1 and 2 respectively. Obviously, not all combinations of all potential master and slave modes have to be considered, but only those combinations with the lowest values that still upper bound the most common requests.

At integration time, the scheduler selects the mode that each task will have for each master and slave. Task k does not violate its deadline for any master i such that:

$$WCET_k^{t_{arb}(i),t_{tran}(i)} \leq d_k \quad (3)$$

where d_k is the deadline of task k. Tasks are not allowed to change the mode of any master or slave. Only the Operating System scheduler is allowed to change those modes, for which we envisage a privileged instruction such as writing to a special purpose register. This prevents, tasks with different criticality levels affecting each other timing behavior beyond the setup set by the scheduler. An example illustrating how the scheduler interacts with the master/slave modes is provided in Annex II.

C. Providing the same functionality as AHB

As shown in Section IV-A, only the functionality provided by locked transactions is affected when using the restricted version of AHB. In *AHRB*, we keep the same principle to deal with locked transactions, however, the timing restrictions now depend on the master and slave modes. This means that any locked transaction has to fit these restrictions, i.e., a master contribution to a transfer must never exceed its operation mode limit (the same applies for slaves). The best solution to provide the same functionality as locked transactions, is to implement the functionality of *atomic operations* in the slave, as shown in Section IV-A. This is so because in order to provide tighter WCET estimates, *AHRB* may have more severe timing restrictions that make more difficult to perform locked transactions within the allowed maximum transaction time.

D. AHRB architecture

In addition to the AHB architecture shown in Figure 1, *AHRB* needs some extra hardware support. Every master and slave requires knowing the mode it is allowed to operate. The arbiter keeps the information about the mode of every master and the slave mode associated to each master. This information can be changed by software (in supervisor mode), in order to allow the scheduler to change master and slaves modes so it can use different WCET estimates for any given task. This can be done through special purpose registers, which is a common mechanism in current processor architectures.

We introduce two new signals that define the master mode, *HMMODE*, and the slave mode, *HSMODE*. These signals are generated in the arbiter and reach the master or slave component respectively. The extended bus architecture is shown in Figure 3.

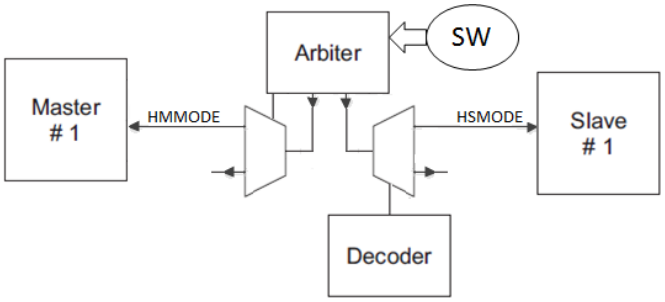


Fig. 3. AHRB extended architecture with HMMODE and HSMODE signals.

VI. EVALUATION

In this section we quantitatively compare the time composable AHB proposals, *resAHB* and *AHRB* against the original AMBA AHB in terms of time composability, WCET estimates and average performance.

A. Experimental setup

Chip Setup. We use SoCLib [21] simulator to model a multicore architecture with 4 cores connected through a bus to the L2 cache and a I/O controller.

Each core is a pipelined processor core comprising fetch, decode, execute and commit stages. Each core has its own private instruction (IL1) and data (DL1) caches, which is common in current high-performance and real-time embedded processor designs [1] [10]. 16KB 4-way 16-byte-line IL1 and DL1 caches have been considered. The shared second level (L2) cache is 256KB with 8 banks, 8 ways and 16-byte lines. IL1 and DL1 hit and miss latencies are 1 and 2 cycles respectively. L2 hit and miss latencies are 4 and 6 cycles respectively. DL1 is write-through and L2 write-back. All caches use LRU replacement policy. The bus connecting the cores to the L2 and the I/O device is 128-bit wide, which means that a cache line can be transferred in a single-beat transfer and no burst transfers are needed. The I/O controller has 20 cycle latency.

The L2 cache deploys a cache partitioning technique, way partitioning, that deals with inter-task interferences [15] and has been implemented in real chips like the ARM Cortex A9 [5]. For the memory controller we use the low-overhead solution proposed in [16], which upper bounds the effect of inter-task interferences on the requests of a core to the memory controller. Overall, the only source of inter-task interferences that jeopardizes time-composability, potentially affecting WCET estimation, is the AMBA AHB bus.

Benchmarks. We use the EEMBC Autobench suite [17] as reference programs, which behave as some real-world automotive critical applications. In particular we use: *a2time*, *aifftr*, *aifirf*, *aiifft*, *basefp*, *cacheb*, *canrdr*, *idctrn*, *iirflt*, *matrix*, *pntrch*, *puwmod*, *rspeed*, *tblook* and *ttsprk*.

We also develop a set of synthetic kernels that carry out a fixed number of accesses to the bus, either to the L2 or to the I/O device. We vary the percentage of accesses of the benchmark to the L2 and to the I/O device, such that, if a kernel makes $X\%$ of access to the I/O, the remaining $100\% - X\%$ of the accesses go to the L2.

B. Achieving Time Composability

As MPSoCs integrate an increasing number of IP blocks coming from different suppliers and those IPs may be subject to different safety and security levels, enabling *mixed-criticality functionalities* to be run on the same MPSoC requires limiting the impact that one IP component can create

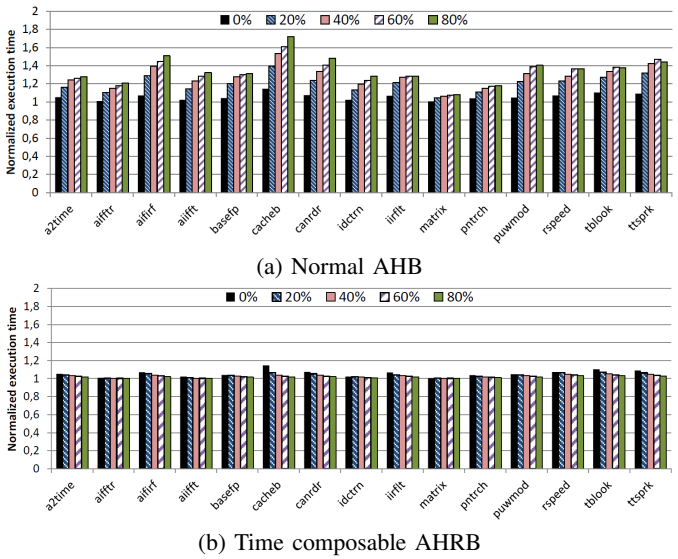
on the timing behavior of the others. Leaving this to the IP provider, especially to those subject to less-restrictive criticality levels, may jeopardize the whole system time composability property. AHRB is precisely designed to provide trustworthiness in terms of timing behavior by design (construction), so that any misbehaving IP component does not affect the time composability of the other components. Hence, we evaluate our proposal in a setup in which time composability can be broken, as it is shown to happen with AHB.

In our experimental setup, the I/O controller has 20-cycles latency and, under AHB, it benefits from unrestricted timing by keeping the bus busy for those 20 cycles when needed. In the case of the time composable *AHRB*, we configure the slaves in *slave mode 4* (L2 hit latency is 4) and masters in *master mode 1* (only 1 beat is required to transfer a cache line), so that all L2 hit requests are served with only one bus transaction. The I/O component has to relinquish the bus on each request because the bus is configured in slave mode 4, which means that a component can only insert 4 wait state cycles, which is less than the latency of the I/O component (20 cycles). Hence, the I/O component issues a split transaction on every access, and so every request goes through two arbitrations.

In order to understand the potential contention that a transaction can suffer in the bus, we run one EEMBC benchmark in one core while the other three cores run 3 copies of the synthetic kernel that continuously accesses memory and the I/O controller, so that they affect significantly the EEMBC benchmark execution time. Different fractions of memory and I/O accesses are considered for this synthetic kernel. Figure 4 shows the normalized execution time of each EEMBC benchmark when running against 3 copies of the synthetic kernel with respect to its execution time in isolation (running alone in the MPSoC). Results are shown for both the regular AHB and the time-composable *AHRB* scenarios. Note that the execution times in isolation are *the same* under both setups since cache hits are handled in one transaction while misses require two transactions using split transactions. The I/O device is not accessed by EEMBC benchmarks which only access the L2 cache.

Bars show the results as we increase the percentage of I/O requests sent by the synthetic kernel. Figure 4(a) shows the results for the standard AMBA AHB. As shown, bus contention significantly affects the execution time of EEMBC benchmarks (6% to 35% in average). This is so because every I/O access keeps the bus busy for 20 cycles and as the percentage of I/O accesses grows, this saturates the bus, thus delaying L2 cache accesses for EEMBC benchmarks due to bus contention. Note that for *cacheb*, which makes the most intensive use of the bus across all EEMBC benchmarks, the impact of the synthetic kernels is high (up to 72%) when the percentage of I/O accesses grows. Eventually, a slave component could be in place with unspecified or arbitrarily long latency. In such case no WCET estimate could be provided for any program, or such estimate would be so high that it would be of no use.

Figure 4(b) shows the results for the time-composable *AHRB*. We observe that bus contention becomes negligible (1% to 5% in average), since I/O accesses become split transactions, thus letting L2 cache accesses of the EEMBC benchmarks to proceed. As the fraction of I/O accesses of the 3 copies of the synthetic kernel increases, their *bus access frequency decreases*. This is so, because for this kernel, the accesses to L2 frequently hit so they have shorter latency than I/O accesses. As a result, with low L2 access rate, and hence with high I/O access rate, the kernel access less frequently



(a) Normal AHB

(b) Time composable AHRB

Fig. 4. Execution times of EEMBC benchmarks against 3 synthetic kernels that continuously access memory and the I/O controller for AHB and AHRB. The percentage indicates the amount of accesses to the I/O controller.

to the bus. Hence, in the presence of I/O intensive kernels, the EEMBC benchmark is granted access to the bus quickly and competes with fewer requests in memory in case of a L2 miss, thus executing faster. This effect is particularly noticeable for *cacheb* and *canrdr* that have slightly higher execution time when the kernels execute low percentage of I/O operations.

Our results show that the potential effect that tasks can suffer due to inter-task conflicts can be reduced using *AHRB*. *AHRB* also provides time-composable upper bounds by construction, without any requirements on the components at hardware or software level (apart from being compliant with the *AHRB* specification), which significantly simplifies WCET estimation. This allows using components that otherwise would degrade significantly WCET estimates or simply would not allow to obtain such estimates. The main limitation of the standard AMBA AHB is that WCET estimates depend on the actual behavior of components in place. If such timing behavior changes, for example due to a firmware update that changes the I/O component latency to 25 cycles, or simply other applications make a different use of the component triggering higher latencies, the timing analysis of all tasks running on the system is invalidated, even if those tasks do not ever use such component. By putting the timing restrictions on the bus interface instead of on the components, we avoid this issue because the timing upper bounds are set by the bus interface itself and not by the IP component.

C. Average performance

With time composable *AHRB*, I/O transactions are split into two transactions. This, on the one hand, increases the latency of each transaction as it has to pay two arbitration rounds. On the other hand, however, each arbitration round is much shorter since the bus cannot be locked for long time. Our results show that the benefits of shorter arbitration rounds largely offset the extra latency due to the fact that I/O transactions have to pay two arbitration rounds.

Figure 4 show that EEMBC benchmarks have higher average performance, i.e. shorter execution time, under *AHRB* than under AHB. To complement the average performance study, we measured the performance of the synthetic kernel under all percentages of I/O and L2 operations (i.e. 0%,

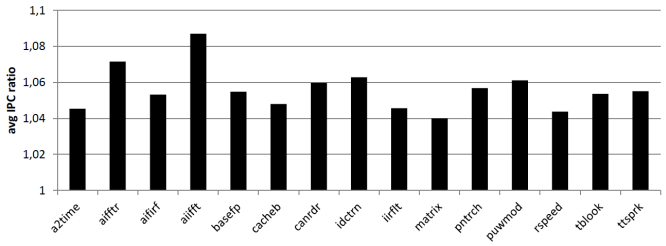


Fig. 5. Average IPC improvement of *AHRB* over AHB, for the 3 copies of the synthetic kernel when running in a 4-workload setup with each EEMBC.

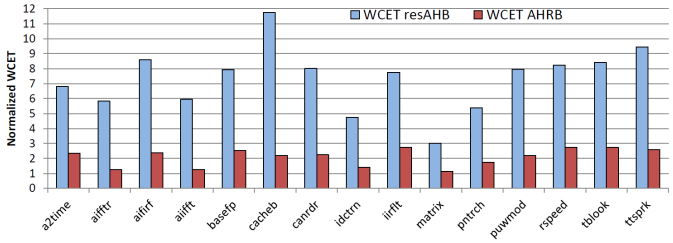


Fig. 6. WCET estimates comparison for *resAHB* and *AHRB*.

20%, 40%, 60% and 80%). Figure 5 shows the average throughput improvement of *AHRB* over AMBA AHB, i.e., the average of IPC_{AHRB}/IPC_{AHB} (IPC stands for instructions per cycle) for all 3 kernels, when they run with each EEMBC under all I/O-L2 percentages. Results confirm the benefits of short arbitration periods over several arbitration phases, hence making AHRB to provide higher average performance than AMBA AHB. These improvements range from 4% to 9%.

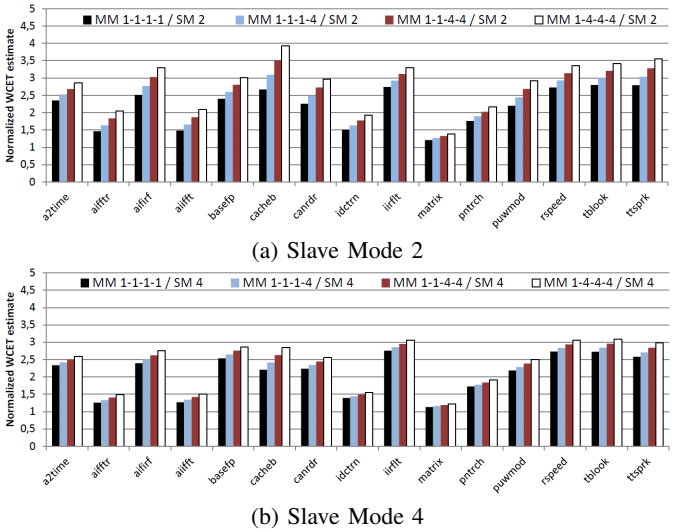
D. WCET estimates

Regular AMBA AHB does not allow deriving time-composable WCET estimates. Instead, they can only be derived with *resAHB* or *AHRB*. For both, *resAHB* and *AHRB*, we have to assume that every access to the bus experiences the maximum arbitration latency [15] and also every memory access experiences the longest *request inter-task delay* [14] (i.e., the maximum possible delay due to requests from the other cores), since they are the only sources of *inter-task interferences* in our platform. For the bus, we take 148 bus cycles as the maximum arbitration latency for *resAHB* as explained in Section IV-B and 19 for *AHRB*, applying Equation 2 with master modes being 1 and slave modes 4 (also shown in Table II).

Figure 6 shows the normalized WCET estimates of each EEMBC benchmark with respect to its WCET when running in isolation for both *resAHB* and *AHRB*. It can be clearly seen that *AHRB* leads to much tighter WCET estimates than *resAHB*, in average 3.5x tighter. This is so because the arbitration latency bound is high for *resAHB*, whereas such bound is much tighter for *AHRB* since we fit the bound to the master and slave characteristics with the *master and slave modes*. Although *resAHB* allows deriving WCET estimates, those estimates are so high (above 4x the ones in isolation) that it would be better to use just one core in the platform to schedule all tasks. Instead, *AHRB* provides WCET estimates largely below 4x, thus providing higher guaranteed performance when exploiting the 4 cores in the 4-core platform than when using just one core.

E. AHRB: Master & Slave modes

In order to show the effect of the master and slave modes on WCET, we compute WCET estimates for EEMBC under *AHRB* with different modes. In particular, we show results for



(a) Slave Mode 2

(b) Slave Mode 4

Fig. 7. WCET estimates for *AHRB* with different master modes (1 and 4) and slave modes (2 and 4).

slave modes 2 and 4. With slave mode 4 (equal to the L2 hit latency) we cover each hit request with one transaction. Instead for the slave mode 2, we need to issue split transactions and pay two arbitrations for every L2 hit. Hence, in slave mode 2, L2 accesses suffer lower bus contention due to the shorter transactions in the other cores, but suffer higher arbitration delay in case of a hit due to the split transaction. Therefore, depending on the L2 access frequency and L2 hit rate, the net effect in WCET estimates will vary.

The maximum arbitration time also depends on the other masters' mode. The lower the other masters' mode is, the smaller the arbitration time is. In this evaluation we assume that the master that runs the EEMBC is in master mode 1 and it runs with three other masters, either in mode 1 or mode 4, which correspond to masters that need only 1 transfer per access (e.g., cache line size matches bus width) and masters that need 4 transfers per access (e.g., cache line size is 4 times larger than bus width so a 4-beat burst is needed)[6]. In this case it is clear that the lower the addition of the transfer sizes of the other masters, the tighter the WCET estimate we will be able to derive. The possible combinations and the associated maximum arbitration time computed with Equation 2 are shown in Table II.

Figure 7 shows the results of two slave modes (2 and 4) for the EEMBC benchmarks, varying the master modes, shown as MM x-x-x-x in Figure 7. The first master, which is used by the EEMBC benchmark, is always in mode 1. We observe that WCET estimates depend on the rest of the masters modes. The higher the other master modes, the larger the WCET estimate is. We also observe that there are benchmarks that are barely affected by the modes, like *matrix*, because of the reduced use of the bus, and others are, however, significantly affected like *cacheb*. Finally, our results show that under *sm* 2, WCET estimates increase by 6% to 19% on average for the different master setups, because L2 hits, which are very frequent for many benchmarks, need 2 transactions to be completed, because the slave (cache) needs to perform split transactions. For instance, we observe that *cacheb* is severely affected when lowering the slave mode to 2 because it makes an intensive use of the L2 cache. For *matrix* the effect is

[6]Note that a high master mode might be useful to enable efficient (large) DMA transfers performed by a DMA master.

negligible because it barely accesses L2 cache.

F. AHRB principles in AMBA3 (AXI)

AXI is a specification in AMBA3 which only defines interfaces between (1) the master and the slave, (2) the master and the interconnect and (3) the slave and the interconnect, allowing the chip designer to use potentially any interconnect. On the one hand, in order to determine whether time-composable access delays for the interconnect can be achieved, a specific interconnect has to be defined and analyzed. For instance, crossbars are time-composable by design. However, the fact that AXI does not define the timing aspects in the master-slave-interconnect communication provides full freedom to define an AXI-compliant time-composable interconnect (*tcAXI*).

In our view, this offers an excellent opportunity to real-time industry to define a *tcAXI* enjoying a well-defined and standardized interface such as AXI, while adding specific restrictions to make it time-composable. The principles that rule the definition of *tcAXI* should be in line with those we define in this paper.

Besides AHB and AXI, the AMBA specification defines other interfaces that we qualitatively analyze in terms of time composability in Annex III.

VII. Related Work

Most of the previous work on bus architectures for real-time multicores like [19] [15] [11] focuses on the bus arbitration policies. Such work assumes a generic simple bus that has a bounded (or fixed) transaction latency, which is not always the case in real implementations, as shown for the AHB bus.

There exist some real-time buses, like FlexRay used in automotive, AFDX used in avionics and the Time-Triggered Architecture [13]. Those buses are designed to connect different processing units and peripherals. Transactions on those buses are visible to the software. This enables scheduling the requests of the different running tasks, so that interferences in the use of the bus are prevented, hence removing the need for arbitrating them at hardware level. AMBA, however, is used at much lower granularity, for instance to communicate cores and the L2 cache in a multicore. At this granularity, the requests (e.g., L2 cache accesses) of the different tasks cannot be scheduled to prevent interferences. The responsibility to handle interferences is left to the hardware, in this case to the arbiter of the AMBA bus.

To our knowledge, existing studies on AMBA buses focus on RTL models and efficient implementations of the different AMBA interfaces. As an example, [6] analyzes several arbitration algorithms for AHB in terms of latency and power dissipation. However, no work considers adapting the AMBA specification for real-time time-composable systems, as it is the focus of our study.

VIII. Conclusions and Future Work

MPSoCs have become a must in critical real-time embedded systems (CRTES) since they deliver high performance needed for increasingly computational intensive applications. Integrating different IP components in those processors requires a standard specification for the communication bus, and AMBA AHB has been proven to be a suitable interface in embedded systems. Unfortunately, the need for incremental qualification poses requirements regarding time composability in the AMBA AHB.

In this paper we thoroughly review AHB features identifying those that make AHB fail to provide time composability and those that, although time-composable, lead to non-tight WCET estimates. Then, we propose a time-composable AHB (*resAHB*) specification, which enables computing application's WCET, though with large overestimations. Finally, we introduce a new bus specification based on AHB, *AHRB*, which achieves both, time composability and tight WCET estimates. Our experiments show that WCET estimates can be derived on top of *resAHB* and *AHRB*, and *AHRB* improves WCET estimates by 3.5x w.r.t. *resAHB* with negligible average performance degradation over single-core execution.

Acknowledgements

The research leading to these results has received funding from the European Space Agency under NPI Contract 40001102880 and the European Community's Seventh Framework Programme under grant agreement no. 287519 (parMERASA). This work has also been supported by Spanish Ministry of Science and Innovation grant TIN2012-34557.

References

[1] *NGMP Preliminary Datasheet Version 2.1, May 2013.*

[2] Aeroflex Gaisler. *Quad Core LEON4 SPARC V8 Processor - LEON4-NGMP-DRAFT - Data Sheet and Users Manual*, 2011.

[3] Areroflex Gaisler. *Leon3 Processor*. http://www.gaisler.com/.

[4] ARM Ltd. AMBA specification (rev. 2), 1999.

[5] ARM Ltd. The ARM Cortex-A9 processors (white paper), 2009.

[6] M. Conti, M. Caldari, G. Vece, S. Orcioni, and C. Turchetti. Performance analysis of different arbitration algorithms of the AMBA AHB bus. In *Design Automation Conference, 2004. Proceedings. 41st*, pages 618–621, 2004.

[7] M. Di Natale and A. Sangiovanni-Vincentelli. Moving from federated to integrated architectures in automotive: The role of standards, methods and tools. *Proceedings of the IEEE*, 98(4):603–620, April.

[8] J. Elmqvist, S. Nadjm-Tehrani, K. Forsberg, and S. Nordenbro. Demonstration of a formal method for incremental qualification of IMA systems. In *Digital Avionics Systems Conference*, 2008.

[9] D. Flynn. AMBA: enabling reusable on-chip designs. *Micro, IEEE*, 17(4):20–27, 1997.

[10] Infineon. AURIX Safety joins Performance.

[11] J. Jalle, J. Abella, E. Quinones, L. Fossati, M. Zulianello, and F. J. Cazorla. Deconstructing bus access control policies for real-time multicores. In *International Symposium on Industrial Embedded Systems*, pages 31–38, 2013.

[12] F. Kirschke-Biller. AUTOSAR - a global standard. *4th AUTOSAR Open Conference*, June 2012.

[13] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.

[14] M. Paolieri, E. Quiñones, and F. J. Cazorla. Timing effects of DDR memory systems in hard real-time multicore architectures: Issues and solutions. *ACM Trans. Embed. Comput. Syst.*, 12(1s), 2013.

[15] M. Paolieri, E. Quinones, F. J. Cazorla, G. Bernat, and M. Valero. Hardware support for WCET analysis of hard real-time multicore systems. In *International Symposium on Computer Architecture*, pages 57–68, 2009.

[16] M. Paolieri, E. Quinones, F. J. Cazorla, and M. Valero. An analyzable memory controller for hard real-time CMPs. Embedded System Letters (ESL), 2009.

[17] J. Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.

[18] R. Rajkumar, L. Sha, and J. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Real-Time Systems Symposium*, 1988.

[19] J. Rosen, A. Andrei, P. Eles, and Z. Peng. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In *Real-Time Systems Symposium*, 2007.

[20] E. Salminen, T. Kangas, V. Lahtinen, J. Riihimäki, K. Kuusilinna, and T. D. Hämäläinen. Benchmarking mesh and hierarchical bus networks in system-on-chip context. *Journal of Systems Architecture*, 2007.

[21] SoCLib. -, 2003-2012. http://www.soclib.fr/trac/dev.

[22] A. N. Udipi, N. Muralimanohar, and R. Balasubramonian. Towards scalable, energy-efficient, bus-based on-chip networks. In *International Symposium On High Performance Computer Architecture*, 2010.

[23] C. Watkins and R. Walter. Transitioning from federated avionics architectures to integrated modular avionics. In *Digital Avionics Systems Conference*, 2007.

[24] Wilhelm R. et al. The worst-case execution-time problem overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7:1–53, May 2008.

[25] J. Windsor, M.-H. Deredempt, and R. De-Ferluc. Integrated modular avionics for spacecraft - user requirements, architecture and role definition. In *Digital Avionics Systems Conference (DASC)*, 2011.

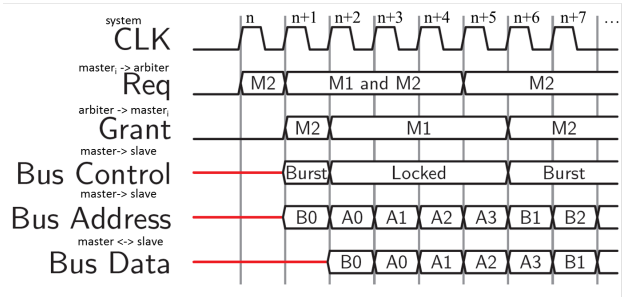


Fig. 8. Example of non time-composable behavior.

ANNEX I:EXAMPLE OF NON TIME-COMPOSABLE BEHAVIOR UNDER AMBA

Figure 8 shows an example that illustrates three AHB-compliant non-time composable behaviors of a master/slave communication. Concretely, it shows the timing diagram of the AHB bus of two masters, $M1$ and $M2$, each sending transactions A and B respectively, both with a burst transfer size of 4 beats. The transfer part of the transaction from master $M1$ occurs between cycles $n+2$ and $n+6$. In those transfers, phases overlap across beats. For instance, address phase of transfer $A1$ overlaps with the data phase of transfer $A0$ given that they use separate buses. Non time-composable behaviors are as follows:

1) $M1$ requests the bus in cycle $n+1$ and gets it in cycle $n+2$, thus having an arbitration time of 1 cycle. Master $M1$ issues a locked transaction at cycle $n+2$. As defined in the AMBA specification the arbiter cannot relinquish the bus during a locked transaction, so that this transaction from $M1$ can affect the timing of other masters in an unpredictable manner potentially preventing them from using the bus indefinitely (4 cycles in the example). Hence, the existence of locked transactions is a feature that breaks time composability.

2) Masters can issue unspecified-length burst transfers, which does not allow bounding the latency of a transaction. For instance, in Figure 8, $M1$ issues a burst length of 4, which means that for WCET estimation we have to assume that every possible transaction from $M1$ is at least 4 beats long. However, if a new master is connected with a burst-length of 8, it invalidates the previous analysis.

3) Finally and more revealing, AHB specification does not put any requirements on the arbiter behavior, and only the arbiter interface is specified. For instance, the arbiter can take away the bus grant from a master, as occurs in cycle $n+2$ when $M2$ is sending a burst transfer and the arbiter changes the bus ownership to $M1$. This can happen due to the fact that any arbitration policy to select the next master to grant access to the bus can be used. Under some arbitration policies, there may not be an upper bound on the time one master can delay the others to access the bus, thus, breaking time composability. For instance, if $M1$ has higher priority than $M2$ and both tasks attempt to access the bus simultaneously, $M2$ is stalled until $M1$ finishes. However, if before the first request from $M1$ finishes, another request from $M1$ becomes ready, $M2$ will also wait for the second $M1$ request to finish as well. Thus, the bus contention that $M2$ suffers depends on $M1$. Even though this effect can have an upper bound and be computed knowing $M1$, it breaks time composability, because if we change $M1$ behavior (e.g., the task running on top of such master) it invalidates the timing analysis for $M2$.

TABLE III. WCET (IN MILLISECONDS) FOR THREE TASKS UNDER SEVERAL MM/SM

Tasks	mm/sm modes							
	111-2	114-2	141-2	411-2	144-2	414-2	441-2	444-2
t_0	1.2	1.4	1.4	1.4	1.6	1.6	1.6	1.8
t_1	1.3	1.4	1.4	1.4	1.5	1.5	1.5	1.7
t_2	1.7	1.3	1.9	1.9	1.5	1.5	2.3	2.1

ANNEX II: TASK SCHEDULER AND THE MASTER/SLAVE MODES, AN EXAMPLE

The following synthetic example details how the scheduler interacts with the arbiter and the master/slave modes. For simplicity we focus on the case in which each master can be used by only one task. In our example we assume three tasks (t_0, t_1 and t_2) scheduled in three different cores (masters) and using one or more slaves. Each master can operate under mode 1 or 4, and all slaves operate always under mode 2. The deadline and period for all tasks is 2ms. Table III shows the WCET estimate in milliseconds for each task under each master/slave mode combination. For instance, *114-2* corresponds to the case where t_0 and t_1 operate in master mode 1, t_2 in master mode 4 and the slave in mode 2.

In the example, t_0 and t_1 send short transfers, so master mode 1 provides them good performance. Conversely, t_2 achieves lower WCET estimates under master mode 4 since its transfers are longer and master mode 1 produces split transactions. Therefore, the lowest WCET estimates for t_0 and t_1 occur under 111-2 modes, and for t_2 under 114-2 modes.

Different approaches can be followed to choose the proper modes for each task. However, in all cases 441-2 and 444-2 are not eligible because t_2 would violate its deadline (2ms). Out of the other combinations of modes, 114-2 would be the one minimizing the WCET estimates for all tasks, $1.4+1.4+1.3 = 4.1$, hence reducing the CPU capacity used by those tasks, which can be left to other tasks. 111-2 modes minimize the CPU capacity required by t_0 and t_1.

Once a particular combination of modes is selected (e.g., 114-2), the arbiter is in charge of properly configuring masters and slaves on each arbitration every time it grants access to a master. For instance, if the master where t_0 runs is granted access, the arbiter sets it to master mode 1 and the slaves as slave mode 2. Similarly, when the arbiter grants access to t_2 master, it enforces such master to operate in mode 4.

In the general case, unlike in our example above where each task is bound to a master, masters and slaves can be used by several tasks. However, the same principles drawn in this example rule the allocation of modes to task master and slaves.

ANNEX III: OTHER AMBA SPECIFICATIONS

Besides AHB, the AMBA specification defines other interfaces that we qualitatively analyze next in terms of time composability.

AHBLite/APB. APB and AHBLite are intended for a single master, which prevents inter-task interferences and hence the need of time-composable access delays.

ASB. ASB was designed to be the main system bus, like AHB. However, AMBA recommends AHB for all new designs because AHB provides improved features. Anyway, the same approach explained in this paper can be carried out in ASB.

ACE. ACE adds system-level coherence support to AXI, which does not have any impact on timing.

ATB. ATB is a trace bus for on-chip debug, which is free of real-time constraints.