The Priority Division Arbiter for low WCET and high Resource Utilization in Multi-core Architectures

Hardik Shah[1], Kai Huang[1,2] and Alois Knoll[1]
[1]Department of Informatics VI,Technical University Munich
85748 Garching, Germany
{shah, huang, knoll}@in.tum.de
[2]School of Mobile Information Engineering, Sun Yat-Sen University
huangk36@mail.sysu.edu.cn

ABSTRACT

Shared memory arbiters play a major role in determining the Worst Case Execution Time (WCET) of applications executing on multi-core architectures. Apart from the produced WCET, shared memory utilization is another important parameter which characterizes the suitability of an arbiter for a particular system. This paper compares the traditional arbiters, the Static priority (aka fixed priority), the Time Division Multiple Access (TDMA) and the Round robin against the Priority division arbiter on the above mentioned merits. The paper extends the Priority division arbiter by providing a new configuration, called `h1`, which is highly attractive for mixed critical systems with a single Hard Real-Time (HRT) application. The paper derives formulas to calculate the worst case latency and the worst case memory utilization. The analysis proves that the Priority division arbiter outperforms the TDMA arbiter in memory utilization and the Round robin arbiter in the produced WCET. Moreover, the Priority division arbiter, under the `h1` mode, outperforms the Static priority arbiter in terms of the produced WCET of the single HRT application. The supporting evidences are achieved by conducting experiments on a quad-core NIOS architecture built on Altera FPGA. The test applications are chosen from the Mälardalen WCET benchmark suit.

1. INTRODUCTION

Today's safety critical applications have become computationally intensive. Multi-core architectures are the most promising candidates to satisfy this need of computation power due to their high performance per watt ratio. Typically, multi-core architectures employ shared resources (*memory*) to reduce package size and thereby the overall cost of the product. The interference on the shared memory prolongs execution of applications unpredictably. This is considered as the biggest challenge in determining the WCET of applications executing on multi-core architectures.

RTNS 2014 , October 8 - 10 2014, Versailles, France
Copyright 2014 ACM 978-1-4503-2727-5/14/10 ...$15.00.
http://dx.doi.org/10.1145/2659787.2659817.

To avoid interference on shared memory, a static time slot can be assigned to each core. This approach is called Time Division Multiple Access (TDMA). Here, only the assigned core can access the shared memory in the corresponding slot. Thus, the interference is avoided altogether, hence, the experienced latency is the worst case or the best case latency. TDMA is well known to produce low WCET bound [12]. However, it leads to reduced shared memory utilization due to the wastage of slot if the assigned master does not use it. Shared memory bandwidth is a scarce resource and its wastage may not be acceptable.

To utilize the wasted slots, Round Robin (RR) arbiter and Priority Division (PD) [16] can be employed. RR and PD have their advantages and drawbacks with respect to the produced WCET and the shared resource utilization. Under all these arbitration schemes (TDMA, RR and PD), finite WCET of applications executing on multiple cores can be achieved. Hence, all these arbiters are multi HRT capable. Under the Static Priority (SP) arbitration, finite WCET of application executing on only the highest priority core can be achieved. Hence, it is termed as single HRT capable.

In this paper, we compare the above mentioned arbiters in terms of the produced WCET and the worst case shared resource utilization. Our analysis and experiments show that the PD arbiter, at slight worst case resource utilization penalty, outperforms the RR and the SP in the produced WCET. The resource utilization penalty is far less than that of the TDMA. Additionally, all the arbiters are similar in the occupied chip area.

The paper compares the effect of different arbiters on the produced WCET and BCET of test applications. Hence, execution time impacting sources, e.g. cache and pipeline effects, branch predictor status etc, are assumed with a constant effect. The motivation behind this assumption is that these effects are also present in single core architectures and existing techniques [21] can be employed for their analysis. The focus of this paper is to study the effect of shared memory interference on the produced BCET and WCET of applications.

The main contributions of the paper are as follows. i) The paper derives conditions for the worst case latency and the worst case resource utilization for the SP, the TDMA, the RR and the PD arbiters. The latency analysis is done in isolation considering no knowledge of the co-existing applications. ii) The paper extends the existing PD arbiter by providing a configuration (`h1`) which is highly attractive to the mixed critical systems with single HRT application.

iii) The experiments are conducted on an FPGA board using a quad-core NIOS processor and test applications from the Mälardalen WCET benchmark suit [6].

The paper is organized as follows. Sec. 2 discusses the existing related work. Sec. 3 builds the background knowledge. Sec. 4 revisits the PD arbiter and extends its capability by providing the `h1` configuration. Sec. 5 describes our experimental set-up and compares the arbiters and Sec. 6 concludes the paper.

2. RELATED WORK

There has been a few arbiter proposals to provide low latency bound as well as high resource utilization. We classify these arbiters in the following categories.

Close to traditional: Static priority (SP), Time Division Multiple Access (TDMA) and Round Robin (RR) are considered traditional arbiters. As described in Sec. 3.5 and Sec. 3.6, TDMA and RR arbiters either provide low worst case latency or high resource utilization, respectively. The SP provides low worst case latency to only the highest priority master. For other masters, the worst case latency is infinite. To balance the worst case latency and the resource utilization, "close to traditional" arbiters are proposed by mixing traditional arbiters.

dTDMA [14] dynamically inserts slots for only active masters in the bus schedule and removes the slot if master is not active. This results in high bus utilization and guaranteed bandwidth to each master. The worst case latencies under the dTDMA is equally high as RR, hence, it results in high WCET bounds. Paolieri `et al` [11] propose interference aware bus arbitration scheme. Here, HRT tasks have higher priority in accessing the shared resource compared to SRT. Contention among HRT is resolved by the RR arbitration. Thus, it mixes SP and RR scheme. However, in an all HRT system, the arbiter behaves as a RR and carries the drawback of high worst case latencies.

Poletti *et al* [13] propose a new arbitration scheme: *slot reservation*. The arbitration scheme is similar to FlexRay [1] where a static slot is assigned to the critical master for guaranteed bandwidth and dynamic slots are arbitrated using the RR scheme by other masters. Thus, it mixes TDMA and RR. Like TDMA, the arbiter results in poor resource utilization if the static slot is unused and incurs high worst case latencies in the dynamic slot, like the RR.

Priority Division (PD) [16] is another "close to traditional" arbiter. It optimizes the TDMA schedule by re-arbitrating the unused slots by static priorities. Hence, the advantage of the TDMA (low WCET) is preserved, however, the drawback (wasted slots) is decreased. If all masters utilize their slots, the PD behaves exactly as the TDMA. This paper extends the work presented in [16] by deriving conditions for the worst case latency and the worst case resource utilization. Moreover, a new configuration, `h1`, suitable to the mixed critical systems with single HRT is presented.

As explained above, all "close to traditional" arbiters, in certain situations, behave exactly as the corresponding traditional arbiters. Apart from the "close to traditional" arbiters, randomized arbiters and budget based arbiters are proposed in literature.

Randomized arbiters: Lottery arbiter [10] and [7] grant the shared resource to a requesting master based on their probabilistic weight. Hence, assumption of the worst case latency for every access can be avoided, instead, a probabilistic latency distribution can be achieved. RT_Lottery [4] employs two level arbitration. Like PD, the first level is arbitrated by TDMA and the second level is arbitrated by the lottery arbitration.

Budget based arbiters: To satisfy bandwidth and latency requirements of different masters, budget based arbiters are proposed. They are widely used in the networking domain. Under these arbitration, the masters are allocated a number of accesses in a unit time. The conflict on the shared resource is resolved using priorities, e.g. in CCSP [2] and PBS [20] arbiters. Under MBBA arbiter [3], number of priority levels are defined, G_i. Each level has N_i cores. Every group is guaranteed to be scheduled after every 2^{G_i} slots which can be considered its budget. Within the slot, the cores are arbitrated using RR. The deficit round robin [19] applies RR under the budgeting.

Arbiter comparisons: Pitter *et al* [12] and Kopetz *et al* [9] compared different traditional arbiters and concluded that the TDMA is the most predictable arbitration scheme. Our work adds to theirs by bringing the SP and the PD to the comparison. Moreover, we derive conditions for the worst case resource utilization and the worst case latency.

The work presented by Kelter *et al* [8] is the closest work to ours. They compared the PD arbiter against the TDMA and the RR. They developed a framework to estimate WCET of applications executing under the arbitration schemes and compared the produced WCET, Average Case Execution Time (ACET) and the resource utilization. They concluded that the PD is a promising alternative for systems running mixed-criticality workloads. Our work adds to their work by deriving conditions for the worst case resource utilization as well as conducting experiments on a real hardware. Their approach is static timing analysis based while our approach is measurements based. Additionally, we extends the PD by providing a single HRT capable configuration which outperforms the SP in the produced WCET.

The randomized arbiters and the budget based arbiters are complex and have large area footprint which hinder their industrial adaptation. As stated earlier, all "close to traditional" arbiters, in certain situations, behave exactly as the corresponding traditional arbiters. Hence, in this paper we compare the PD arbiter directly against the traditional arbiters. The main focus of the paper is to derive conditions for the worst case latency and the worst case resource utilization under different arbiters. We also study the effect of the worst case latency on the produced WCET of applications. Since our approach relies on measurements, it cannot analyze timing anomalies and domino effects [5, 22].

3. BACKGROUND

This section provides the necessary background information to facilitate the discussion presented in the paper.

3.1 Shared resources in multi-cores

COTS based multi-core architectures share on-chip resources in order to reduce number of components on-chip and package size. The reduction in the number of components and package size reduces the over all cost of the product. Main memory is the most common shared resource in multi-core architectures. Fig. 1(b) depicts the basic multi-core architecture.

Here, N number of CPUs are connected to the main memory via a shared bus. Each core is equipped with

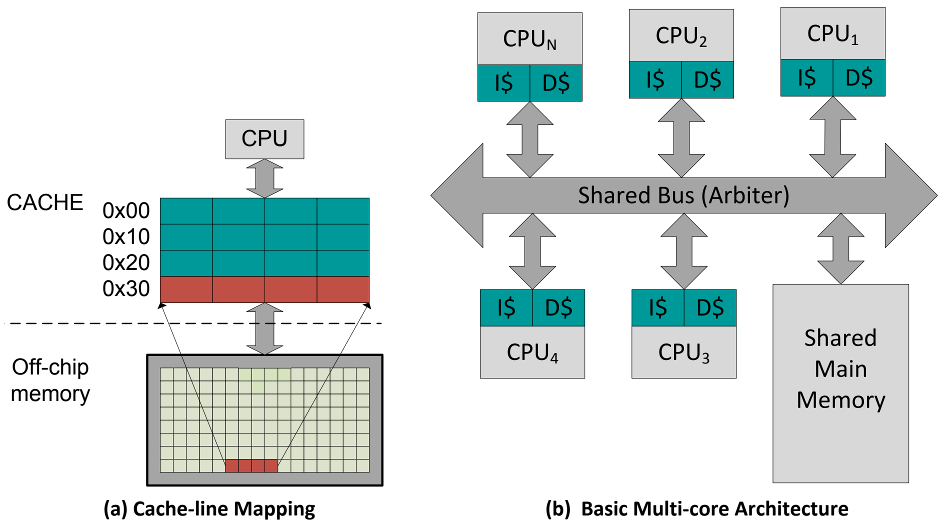

Figure 1: Cache-line Mapping and Basic Multi-core Architecture

instruction and data caches. As long as the required data is available in the caches (cache hit), the main memory is not accessed, unless specified explicitly. However, if the accessed data is not available in the caches (cache miss), the shared main memory is accessed. As depicted in Fig. 1(a), cache is configured in cache-lines. Typically, one of the cache-lines is evicted and the required data is copied from the main memory and inserted in the place of the evicted cache-line. To access the main memory, which is slower than caches, a burst access is issued with burst length sufficient to fill an entire cache-line. The main memory access pattern of any core depends on the application being executed on the respected core, its cache size, cache configuration and the eviction policy employed in the cache.

3.2 Shared resource access latency and utilization

From Fig. 1(b), it is clear that the shared memory serves the masters via a single bus-slave interface. Hence, only one master can access the shared memory at a time. Typically, an arbiter is employed to resolve conflicts on the shared memory. In the event of a collision, one of the cores is granted an access to the shared memory by the arbiter while the others have to wait until they are granted. Thus, the experienced latency of any shared resource access depends on: **i)** Responsiveness of the shared memory itself (generally, memories have slower operating frequencies than the processor cores), **ii)** The employed arbitration policy, and **iii)** The instantaneous activity of co-existing masters in the system at the time when the access request is issued.

From the above factors, the instantaneous activity of co-existing masters is extremely difficult to predict which makes the analysis of the experienced latency extremely difficult. However, the upper and lower bounds on the access latency can be analyzed. They are known as the worst case latency (W_L) and the best case latency (B_L), respectively.

As explained earlier, the main memory is accessed when the required data is unavailable in the cache. Hence, the execution on the waiting cores suspends temporarily until they are granted an access to the shared main memory and subsequently, they fill their caches with the required data from the main memory[1]. Thus, the B_L and W_L factors play an important role in determining the execution time bounds for an application executing on a particular hardware.

As the W_L is used to estimate the WCET (Sec. 3.3), the B_L is used to estimate the Best Case Execution Time (BCET). While deriving the theoretical value of B_L, all favorable scenario is assumed, e.g. no interference or idle shared resource. If the arbiter does not utilize the shared resource efficiently, it wastes clock cycles before scheduling an access in all favorable conditions. Hence, the B_L and the corresponding BCET is higher in the non-work conserving arbiters compared to the work conserving[2] arbiters. The BCET of application is inversely proportional to the number of clock cycles wasted in scheduling an access from the application under all favorable conditions. Thus, low BCET is an indicator of high shared resource utilization, U.

Equation (1) defines the resource utilization considering the master, m, on which the test-application is executing as the only backlogged master ($b_m = 1$). All co-existing masters are considered not backlogged ($b_{m' \neq m} = 0$) to assume all favorable conditions.

$$(b_m = 1) \wedge (b_{m' \neq m} = 0) \ \mid \ U = \frac{\mu_{busy} \times 100}{\mu_{busy} + \mu_{idle}} \% \qquad (1)$$

In the above equation, the right side of "|" derives a value provided that the condition on the left side of "|" holds. Here, μ_{busy} and μ_{idle} are the number of clock cycles for which the shared resource is busy and idle, respectively, when an access from m is backlogged. Considering a constant shared resource access pattern and a constant memory responsiveness, B_L, W_L and U depend only on the employed arbitration policy. The remaining paper studies the effect of the arbitration policy on B_L, W_L and U.

Asymmetric multiprocessing: In this paper, we assume asymmetric multiprocessing, i.e. independent appli-

[1]An out of order core can execute few more instructions while waiting, if, these instructions are not dependent on the required data.

[2]A work conserving arbiter schedules an access immediately as long as the shared resource is idle.

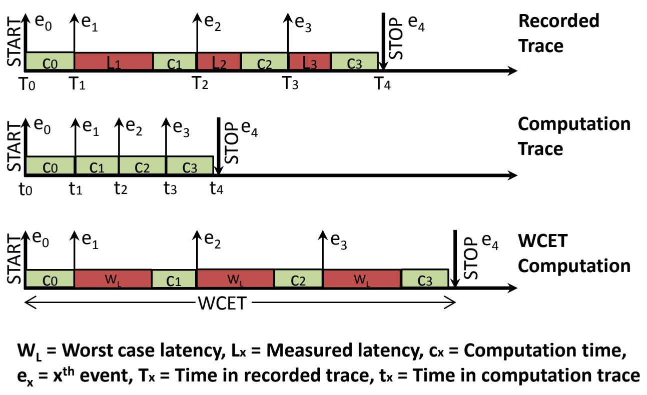

Figure 2: WCET Computation using the Computation Trace

cations are deployed on different cores and the applications do not communicate with each other. The assumption lets us exclusively focus on execution time variation due to the shared resource interference. Our analysis is valid for symmetric multiprocessing as well, however, an additional dependency analysis must be performed. We leave the symmetric multiprocessing for future work.

Before we study the arbitration policies in detail, the following subsection explains the computation trace [18].

3.3 Computation trace

The computation trace is used to estimate the worst case execution time (WCET) of a particular application path considering the worst case interference. It is an execution trace of an application path where cache misses are denoted by timeless events and are separated by computation time $(c_0, c_1, ...)$. The computation time is the time during which the processor executes only from the caches and on-chip registers. The shared main memory is not accessed (cache miss does not occur). The motivation behind the timeless events is the following. In the shared memory architecture, the shared main memory is accessed when a cache miss occurs. The contention on the shared memory delays service to this memory access. Typically, the collision of cache misses on the shared memory is extremely difficult to predict. Moreover, the delay in service also delays the subsequent cache misses (memory accesses) of the application-under-test by the same amount. This causes difficulties in estimating the worst case interference and its impact on the WCET. To avoid these difficulties, at first, we remove all the latencies related to the memory accesses. This means, there is no interference at all and the shared memory takes zero cycles to respond. Later, theoretically calculated worst possible latency, W_L, is added for each cache miss.

Computation trace, depicted in the Fig. 2, can be easily obtained by simulation or using the technique presented in [15]. At first, occurrence time $(T_0, T_1, ...)$ of each cache miss event $(e_0, e_1, ...)$, and its experienced latency $(L_0, L_1, ...)$ are recorded in a trace by executing the application on cycle accurate simulation model. Later, these latencies are removed and each event is shifted towards left in time. The resulting trace is the *computation trace*. Now, to compute the WCET considering the worst case interference, each cache miss event in the computation trace is annotated by the highest possible latency (W_L) considering the worst case interference. Hence, all the subsequent accesses are shifted to the right. Now, the computed WCET contains the effect of the worst possible interference the application may experience. Similarly, by inserting B_L for each cache miss event, BCET of application can be achieved.

The artificial insertion of W_L for each cache miss for conservative WCET analysis is intuitive for in-order processor cores. Today's out-of-order processor cores can keep on executing instructions after a cache miss occurs until the execution is blocked by a dependent instruction. In the following we prove that insertion of worst case latency (execution blocking time), W_L, due to a cache miss is a conservative assumption also for an out-of-order processor.

Assume that for an event e, occurring at time t in a computation trace, the execution does not suspend immediately due to out-of-order execution. Instead, it moves forward for δ amount of time before being blocked by a dependent instruction. In our analysis, we assume that the execution suspends at t and restarts at $t+W_L$. The analyzed worst case execution blocking time is $t + W_L - t = W_L$. However, in the out-of-order case, execution moves forward for δ time after experiencing a cache miss at t. Hence, the *effective* blocking time $= t + W_L - (t + \delta) = W_L - \delta < W_L, \forall \delta > 0$. This proves that insertion of W_L as the worst case latency for each cache miss is a conservative assumption for both, in order and out-of-order execution. The $\delta \geq W_L$ indicates that the execution does not suspend until the cache miss is served. In other words, the application has sufficient *independent* instructions to execute in pipe-line, hence, the *effective* blocking time is 0. Here, again, insertion of W_L as the worst case latency is a conservative assumption.

Note that the latencies in the recorded trace depend on shared memory interference at the time of measurement. However, the computation trace remains unchanged[3] when the same path is executed multiple times, provided that each time we start the application from the same cache state and use the same data as an input. The WCET computed using this method is WCET of that particular *path*. To find the critical path (worst case path), the analysis must be done for each path through application execution.

3.4 Static Priority Arbiter

Static Priority or *Fixed Priority* arbiter is one of the simplest arbiters. Here, each master is assigned a fixed priority. In the event of contention, the master with the highest priority is granted the access. The SP is generally configured in a *non-preemptive* mode for shared memory architectures. Hence, the highest priority master cannot interrupt an on-going lower priority burst and it is blocked until the lower priority burst is finished. To prevent starvation of the highest priority master, the burst length[4] is fixed. Let us denote the burst length by SS[5]. This guarantees that a new scheduling decision is made, at most, after every SS clock cycles.

Latency bound: It is clear that only the highest priority

[3]Here, we assume absence of jitter in occurrence of cache misses due to operating system, FPU, pipeline etc.

[4]The maximum number of clock cycles for which a master can occupy the shared resource after requesting once.

[5]SS stands for slot size which is explained in the next subsection

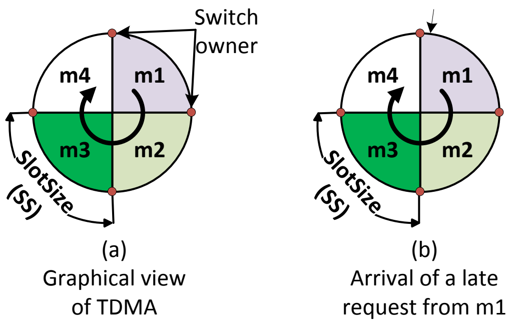
Switch owner
m4
m1
m3
m2
SlotSize (SS)
(a) Graphical view of TDMA
(b) Arrival of a late request from m1

Figure 3: The TDMA Arbiter

master has guaranteed worst case latency bound[6]. In the worst case, the highest priority master must assume that there is an ongoing lower priority burst and it has to wait until the lower priority master finishes its burst. Thus, in the worst case, the highest priority master needs in total twice the SS clock cycles to finish its access. Similarly, in the best case, the shared resource is idle and the highest priority master is scheduled immediately. The following equations, express these bounds.

$$W_L^{sp} = 2 \times SS \tag{2}$$

$$B_L^{sp} = SS \tag{3}$$

To estimate the WCET under the SP for the highest priority master, $W_L = W_L^{sp}$ must be appended in the computation trace. The SP arbitration is used if there is *only one* Hard Real-time (HRT) application among all applications mapped to the multi-core. This HRT application gets the first preference in the event of contention. Certainly, the HRT application can starve other applications. However, typically, the HRT applications are the most carefully designed and are expected to behave properly with high confidence. Moreover, their failure to properly behave may lead to catastrophe, hence, it may be unimportant if the co-existing Soft Real-time (SRT) applications achieve deadline or not if the HRT fails.

Resource utilization: Under SP the shared resource is always busy as long as there is at least one pending request. Hence, $\mu_{idle} = 0$ in equation (1). This leads to $U^{sp} = 100\%$ which indicates the highest shared resource utilization.

3.5 Time Division Multiple Access (TDMA) arbiter

Fig. 3(a) depicts the TDMA arbitration graphically. Here, the shared resource contenders are assigned fixed number of slots in a virtual ring. The figure shows four cores (m1, m2, m3 and m4) in the ring. Each slot has a unique owner. At the beginning of each slot (denoted by *switch owner* points in the figure), owner is switched and new owner is granted the shared resource. The *SlotSize* - *SS* is big enough to accommodate a burst issued to fill a single cache line. Since each contender has a unique window in which it can access the shared resource exclusively, the interference is removed from the system. Now, application executing on one core cannot impact execution time of application executing on another core. However, such predictable execution time comes at the cost of shared resource utilization.

[6]For other priority masters the latency bound is positive infinite since the highest priority master may never let the shared memory idle.

Latency bound: Under TDMA arbitration collisions between accesses are avoided by the exclusive slot allotment. Here, one core cannot impact execution time of another core. The experienced latency depends on the time when a request is issued. Considering N masters in a system, the following equations give the experienced latency L_i for the i^{th} access.

$$\overline{t_i} = N \times SS - \{c_i \bmod (N \times SS)\} \tag{4}$$

$$L_i^{tdma} = \begin{cases} \overline{t_i}, & \overline{t_i} \geq SS \\ \overline{t_i} + (N \times SS), & \overline{t_i} < SS \end{cases} \tag{5}$$

The Equation (4) computes amount of time left in the wheel when the access arrives. Remember that c_i is the time gap between two accesses (cache misses). If $\overline{t_i}$ is less than SS, the access cannot finish before the next slot starts. Hence, it has to wait until the wheel turns around and its slot arrives. However, if there is sufficient time left in the wheel, then the master is guaranteed to be finished at the end of the remaining time (last slot).

The L_i values are bounded by $SS \leq L_i^{tdma} \leq (N+1) \times SS$, however, for the WCET analysis, the theoretical bound is not required. Since one core cannot impact execution time of other core, the existing single-core techniques can be used for WCET measurements. Here, start to end time of each path through application execution is measured and the experienced latency can be considered for the WCET. From all measured execution times, the highest is considered as the WCET and the corresponding path as the critical path.

Resource utilization: Due to the static slot assignment, the TDMA results in a poor utilization. Consider the scenario in Fig. 3(b). Here, the master $m1$ issues a request just one clock cycle after its scheduling opportunity is passed. Hence, its ongoing slot is wasted and it has to wait until the wheel turns around (bus cycle) and its slot arrives. In the definition of the resource utilization, equation (1), we assumed that none of the co-existing masters are backlogged. Thus, all the slots in the ongoing bus cycle are wasted and the slot of $m1$ in the next bus cycle is utilized. This results in tremendous loss of scarce shared memory bandwidth.

If an application does all accesses just one clock cycle after its slot has started, the worst resource utilization occurs. Here, the utilization depends on the access pattern and is given by the following equation.

$$\begin{gathered} \forall i, c_i = (N-1) \times SS + n \times SS + 1, n \in \mathbb{N}^0 \quad | \\ U_w^{tdma} = \frac{100 \times SS}{(N+1) \times SS} = \frac{100}{N+1}\% \end{gathered} \tag{6}$$

Similarly, if an application does all accesses just when its slot is about to start, no slot is wasted which results in the highest utilization.

$$\forall i, c_i = (N-1) \times SS + n \times SS, n \in \mathbb{N}^0 \ | \ U_b^{tdma} = 100\% \tag{7}$$

To maximize the available resource utilization, the round robin arbitration is employed which is described in the following subsection.

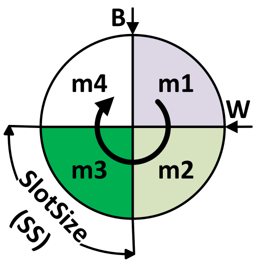

Figure 4: Graphical View of the Round Robin Arbiter

3.6 Round robin arbiter

The Fig. 4 depicts the Round Robin (RR) arbitration graphically. Like TDMA, under the RR scheme, the shared resource contenders are assigned fixed number of slots in a virtual ring depending on their bandwidth requirements. The arbiter continuously searches for a backlogged master. As soon as a backlogged master is encountered, it is granted the memory for a predefined maximum number of clock cycles (`SlotSize - SS`). After the granted master finishes its burst access, the search process resumes from the next slot in the ring. Note, that there is no fixed *start time* of a slot. As soon as a backlogged master is encountered, its slot is started immediately. Thus, the memory is always occupied as long as there is at least one backlogged master (hence it is also known as "*greedy* TDMA") which maximizes the resource utilization. This increased resource utilization comes at the following cost.

Latency bound: Let us assume that the test application is executing on `m1`. For this architecture, an access request from `m1` experiences the worst case completion latency ($W_L = 4 \times SS$) if it is issued when the arbiter pointer is at **W** in Fig. 4 *AND* all other masters utilize their slots. Similarly, an access request experiences the best case completion latency ($B_L = 1 \times SS$) if it is issued when the arbiter pointer is at **B** in the figure. If the exact location of the arbiter pointer *AND* activity of other masters are unknown, being conservative, the worst case latency, W_L, must be considered for each cache miss. Hence, the WCET of applications executing on the architecture with round robin arbiter is significantly higher compared to that produced under the TDMA.

$$W_L^{rr} = N \times SS \tag{8}$$

$$B_L^{rr} = 1 \times SS \tag{9}$$

Resource utilization: RR is a work conserving arbiter. Hence, no clock cycle is wasted as long as there is at least one backlogged master. Thus, similar to SP, RR has the highest resource utilization. $U^{rr} = 100\%$.

4. PRIORITY DIVISION (PD) ARBITER

This section describes the priority division (PD) arbiter. The arbiter has a mix of TDMA and SP arbitration.

4.1 Basic operation

Fig. 5 depicts the graphical view of the PD arbiter. Like TDMA, PD divides the bus schedule into a number of slots. Instead of declaring one master as the owner of a particular

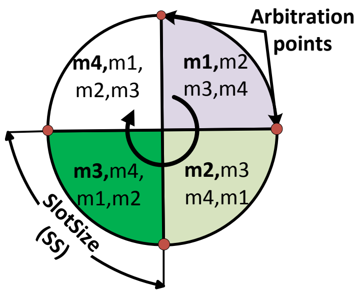

Figure 5: Graphical View of the Priority Division Arbiter

time slot, the PD assigns each master a priority in the according slot. If the highest priority master of a particular slot is not backlogged at the beginning of its slot (denoted by arbitration points in the figure), then a backlogged master is searched in descending priority order. Except in the `h1` configuration (Sec. 4.2), to guarantee *starvation free* characteristic to all masters, each master must have highest priority in at least one time slot. In the figure, each master has one slot in which it has the highest priority (shown by bold letters). The numbers within a slot denote priorities of masters in descending order.

The PD does not enforce any rule for *secondary* priorities. This opens a new dimension of design space exploration where the combined effect of number of slots and priorities inside slots can be investigated. However, given that no knowledge about the requirements of the applications running on the masters can be obtained, the choice of priorities is straight forward. To evenly distribute the shared memory bandwidth not used by the highest priority master, rotating priorities by one from slot to slot is the simplest solution (Fig. 5). If it is foreseen that one core is going to execute memory intensive applications, then this core should be assigned second highest priority in all slots except the slot in which it already has the highest priority, already. This gives the core the first preference to acquire the unused slots.

Latency bound: In PD, if every core utilizes its slot, the operation becomes exactly like the TDMA. Since each core utilizes its slot, the application-under-test does not benefit by utilizing any idle slot and that results in the worst case for PD. Hence, using equations (4) and (5), experienced latency under the PD arbitration can be obtained,

$$W_{Li}^{pd} = L_i^{tdma} \tag{10}$$

If all the co-existing cores are idle, then the master-under-investigation can be scheduled at the starting point of each slot. This is the best case latency under the PD arbitration.

$$B_{Li}^{pd} = (c_i \bmod SS) + SS \tag{11}$$

To measure the WCET under the PD, synthetic stress patterns are executed on all co-existing cores. Thus, these cores never give away their slot converting PD into the TDMA logically. Hence, like TDMA, end-to end execution time can be used for the WCET estimation of the application.

Resource utilization: Under PD, if the highest priority master of a particular slot does not want to use its slot, a lower priority master can use it. However, this happens only at the arbitration points. Once an arbitration point is passed and none of the masters issues a request then the slot is wasted. Hence, under PD the worst resource utilization occurs if a master sends a request just one clock cycle after *any* slot has begun. This master is guaranteed to be scheduled in the next slot since the co-existing masters are assumed inactive.

$$\forall i, c_i = n \times SS + 1, n \in \mathbb{N}^0 \;\mid\; U_w^{pd} = \frac{100 \times SS}{2 \times SS} = 50\% \quad (12)$$

From equation (6) and (12), it is clear that the PD utilizes the shared resource more efficiently compared to the TDMA. $U_w^{pd} >> U_w^{tdma}$.

Similar to the TDMA, the best resource utilization occurs if a request arrives at the beginning of a slot.

$$\forall i, c_i = n \times SS, n \in \mathbb{N}^0 \;\mid\; U_b^{pd} = 100\% \quad (13)$$

4.2 h1 Configuration

The h1 configuration stands for only one HRT supported. Under this configuration, the PD supports only one HRT application and the master executing this application has the highest priority in all slots. Hence, the highest priority master is guaranteed to be scheduled at the beginning of each slot, if it requests. This results in *always* the best case latency for the critical master.

$$L_i^{h1} = (c_i \bmod SS) + SS \quad (14)$$

From equation (2) and (14), $L_i^{h1} \leq W_L^{sp}$. Both, SP and h1 configuration supports only one HRT. Similarly, if the system consists only two HRT, h2 mode can be configured, and so on. Due to the static slots, the h1 configuration is expected to produce lower WCET than the SP. Indeed, the PD under h1 configuration results in the low resource utilization compared to the SP due to the static slotting.

5. COMPARISON OF THE ARBITERS

In this section, we compare the PD arbiter against other arbiters. In the first test, the PD arbiter is compared against TDMA and RR. The comparison criteria are the following: i) Increase or decrease in the WCET due to arbitration, ii) Increase or decrease in BCET due to arbitration, and iii) Required chip area. In the second test, we compare PD in h1 configuration against the SP. Here, both the arbiters are *single* HRT *capable* arbiters. Before we begin with our comparison, the following subsection details the test setup.

5.1 Test setup

We implemented the architecture as depicted in the Fig. 1(b) on Altera Cyclone III FPGA. The multi-core is built using four NIOS II cores. Each core has a 512 Bytes of instruction and data caches. The cache-line size is 32 Bytes. We tested applications from the Mälardalen WCET benchmark suit. Here, we tested applications with *single path*[7] through application execution. We implemented all the arbiters using VHDL. An on-chip SRAM serves as a shared main memory. We avoided use of wait states during burst transfers and arbiter lock signals as suggested in [17] to produce a valid WCET bound.

[7]Multi-path applications are targeted in our another publication [15].

In our setup, core1 ($m1$) executes test applications from the Mälardalen WCET benchmark suit and core2, core3 and core4 execute interfering applications. This is *not* a restriction in our approach. The test applications can be executed on any core, however, the analysis method remains the same. The only restriction is when analyzing SP and h1, understandably, the test applications must be executed on the highest priority core.

We placed the arbiters one by one between the cores and the shared memory and conducted measurements for the utilization, U. The μ_{busy} and μ_{idle} were measured by probing the request and wait signals of $m1$.

5.2 Starvation free arbiters

For the TDMA arbiter, the execution time of test application does not deviate depending on the co-existing cores due to the statically defined exclusive slot allotment. Hence, the Observed Execution Time (OET) is equal to, both, WCET and BCET. For the RR arbiter, the WCET was analyzed as explained in Sec. 3.3. Under PD, the worst case for the test application occurs when it cannot utilize any slot of the co-existing cores. Therefore, to create the worst case, we executed shared memory stressing synthetic applications[8] on co-existing cores.

The Table 1 presents U values in % and WCET values in clock cycles under the *starvation free* arbiters for the test applications. The RR arbiter produces the highest WCET among the three due to the assumption of always interfering co-existing masters. The PD arbiter provides the best trade-off between shared resource utilization and the worst case latency. It utilizes the shared memory more than twice as efficiently as the TDMA arbiter. This expected behavior is due to the fact that an unused slot is arbitrated instead of wasted. As explained in Sec. 3.2, the measurement of utilization is carried out only when an access from the test-application is pending. During the computation time, the measurement is paused. Hence, it is interesting to analyze how much did the application benefit due to this increase in the utilization. This is depicted in Fig. 6 by decrease in BCET compared to the TDMA bar.

The Fig. 6 depicts three factors related to performance for each application. The first bar shows the % decrease of BCET if the PD arbiter is used instead of the TDMA. The PD arbiter can utilize unused slots of co-existing master which results in high resource utilization compared to TDMA and subsequently, reduction in the BCET of application.

The second bar shows the % decrease in the WCET if the PD arbiter is used instead of the RR. Due to the absence of fixed slot staring points in the RR, for conservative analysis, the worst case latency – W_L must be considered for each shared memory access. This increases the WCET of applications significantly. Instead, PD has statically fixed slot starting points (arbitration points) and the highest priority master of any slot is guaranteed access at these points. Thus, like TDMA, the worst case latency is not required to consider for every shared resource access.

And the third bar shows the % increase in BCET if the PD arbiter is used instead of the RR. This is a drawback of the PD arbiter. The PD arbiter does not utilize unused slots

[8]The synthetic code has a data structure twice the size of the data cache. Each instruction accesses the data structure with stride of a cache-line size to make sure that each time a new cache-line is accessed.

Benchmark	TDMA		Round Robin		Priority Division	
	U in %	OET	U in %	WCET	U in %	WCET
compress	30.37	26591	100	30506	71.27	26591
cover	31.56	15805	100	18024	71.97	15805
crc	30.03	106013	100	109163	67.49	106045
duff	34.49	4920	100	5281	76.28	4920
edn	33.62	494584	100	553972	72.04	494584
expint	32.65	16472	100	16708	74.42	16472
fac	30.19	1176	100	1240	70.07	1176
fdct	45.65	21918	100	31837	70.28	21918
fibcall	35.16	1150	100	1228	74.42	1150
fir	32.26	2005692	100	2225970	66.65	2005660
jane	30.35	1016	100	1108	76.19	1016
jfdcint	36.02	31486	100	37035	71.86	31390
matmul	31.01	1633112	100	1764383	67.03	1633048
minver	32.53	161624	100	191270	70.92	161624
ludcmp	34.29	371320	100	456766	69.56	371352
prime	32.58	180831	100	200651	78.17	180990
quart	33.97	223896	100	270686	71.46	223800
recursion	33.33	6813	100	6898	72.72	6813
ud	33.17	40247	100	48212	72.03	40247

Table 1: Execution time in Clock Cycles: TDMA vs RR vs PD

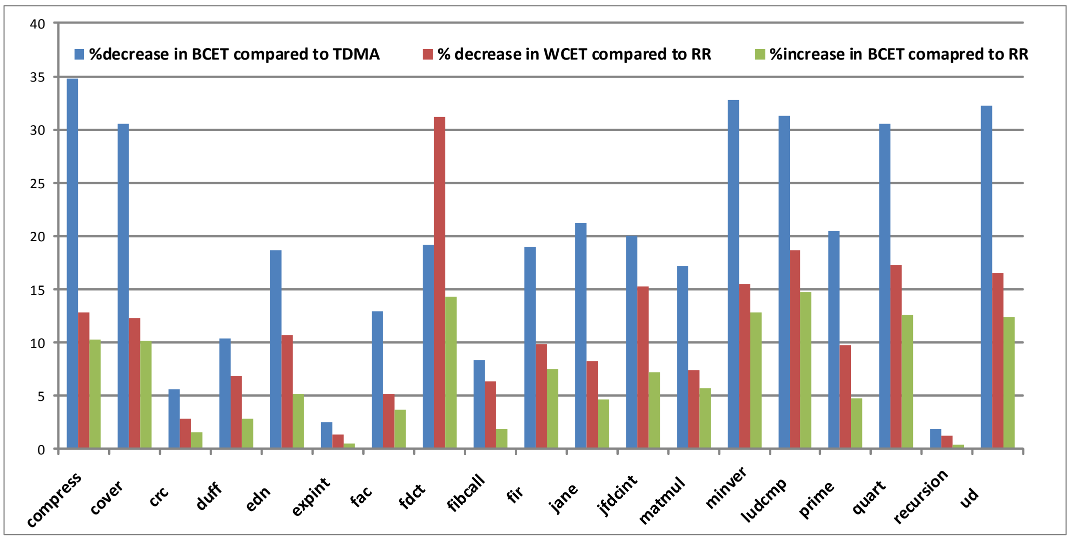

Figure 6: Improvement over TDMA and Round robin arbiters

of co-existing cores as efficiently as the RR arbiter does. It is interesting to note that the third bar is the smallest for all applications which means the advantages of the PD compared to the TDMA and the RR outweigh its drawback.

5.3 Single HRT capable arbiters

In this test, we compare the PD in `h1` configuration against the SP arbiter. Note that these arbitration schemes guarantee upper bound on access latency to only the highest priority master. For all other masters, the worst case access latency is infinite.

Under the SP arbitration, the best case latency occurs when none of the co-existing masters are active. Hence, an access from the master-under-investigation (highest priority) is scheduled immediately. The worst case latency occurs when the master-under-investigation request an access just one clock cycle after a lower priority master is scheduled. Here, although the master-under-investigation has the highest priority, it must wait until the lower priority master completes its burst.

For PD in `h1` configuration, only one HRT core is supported and it has the highest priority in all slots. Only at the beginning of each slot, scheduling decision is made. Thus, irrespective of activity of co-existing masters, if the master-under-investigation (HRT core) is back-logged at the beginning of a slot, then it is scheduled immediately. This removes jitter in execution time due to the interference completely and the OET = WCET = BCET.

The Table 2 shows execution times under PD in `h1` configuration and the SP. Again, these results are valid only

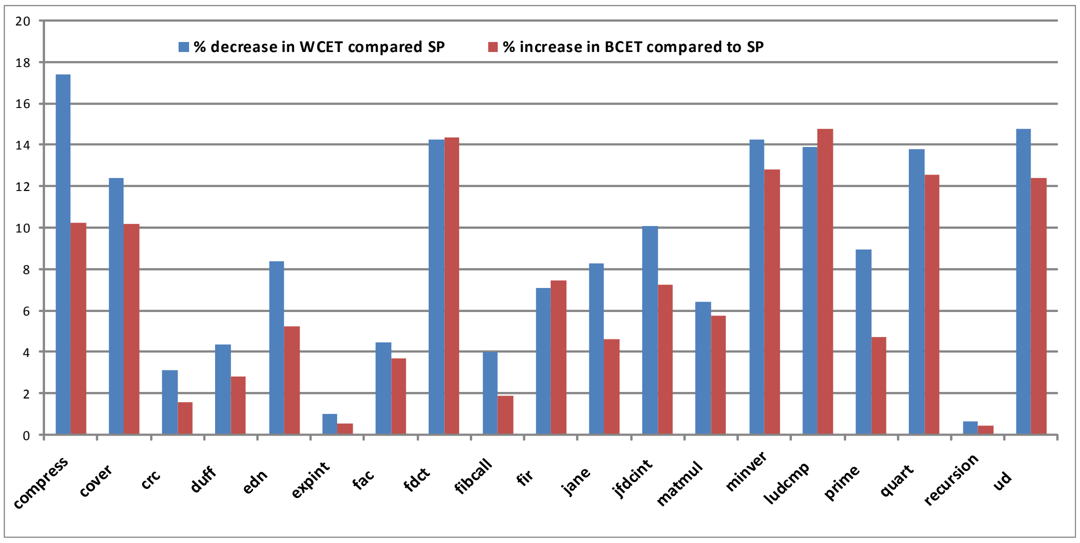

Figure 7: Improvement over Sp arbiter

Benchmark	Static Priority WCET	PD^{h1} WCET
compress	20970	17327
cover	12537	10981
crc	103231	100021
duff	4608	4408
edn	439203	402368
expint	16210	16048
fac	1072	1024
fdct	20649	17710
fibcall	1098	1054
fir	1748622	1624372
jane	872	800
jfdcint	28008	25190
matmul	1444996	1352416
minver	126731	108704
ludcmp	296187	255024
prime	157944	143790
quart	180263	155416
recursion	6730	6685
ud	31993	27271

Table 2: Execution time in Clock Cycles: Sp vs Pd in `h1` mode.

for the highest priority master. The utilization values for the Sp is 100 %. For the Pd, the utilization values are the same as in the Table 1. The `h1` mode does not increase the utilization, it only gives the first preference to the critical application in all slots. While measuring the utilization, we turn-off other cores which implicitly give the first preference to the test-application. Hence, the utilization values are the same as in the Table 1.

Fig. 7 depicts comparison of these two arbiters quantitatively. The first bar shows the % decrease in the Wcet if Pd is employed in `h1` configuration instead of the Sp. The second bar shows the drawback of Pd compared to Sp. It shows the increase in the Bcet which indicates inefficient resource utilization of the Pd arbiter. Similar to the previous test, the advantage of the Pd in `h1` configuration outweighs its drawback.

5.4 Area overheads

Arbiter	Sp	Tdma	Rr	Pd
Number of Logic Elements	281	277	288	285

Table 3: Area overheads of arbiters

The Table 3 lists the required number of Logic Elements for the arbiters when synthesized for a Cyclone III Fpga and 125 MHz clock frequency. The arbiters are synthesized for four ports – at most, four cores can be connected to them.

Due to the functional similarities of arbiters, they consume similar area on chip. The Rr arbiter is slightly more complex since it needs to maintain dynamic slots. Tdma and Pd have fixed slot starting times. Apart from fixed slot starting times, the Pd has to maintain a priority queue which makes it slightly larger than the Tdma. The Sp must guarantee that a scheduling decision is made after each burst access. Hence, the arbiter has to count number of transfers and implement a dynamic arbitration mechanism. This makes it slightly larger than the Tdma.

6. CONCLUSION

This paper has compared the traditional arbiters, the Time Division Multiple Access (Tdma), the Round Robin (Rr) and the Static priority (Sp) against the Priority division (Pd) arbiter. The merits for the comparison are the produced Wcet of applications executing on a multi-core architecture and the worst case shared resource utilization. The paper has also extended the Pd arbiter by providing a single Hard Real-time (Hrt) capable configuration, called `h1`. The theoretical analysis of the worst case latency and the worst case resource utilization is provided. The experiments are conducted on a quad core NIOS processor built on an Altera Fpga. The test applications are chosen from the

Mälardalen WCET benchmark suit. The experiments show that the PD arbiter, in its native mode, produces up to 31% less WCET compared to the RR arbiter and utilizes the shared resource more than twice as efficiently as the TDMA arbiter. In the single HRT mode, the PD produces up to 15% less WCET compared to the SP for the highest priority application. Additionally, the PD consumes similar chip area as other arbiters.

The analysis and the experiments presented in this paper conclude that if a very small utilization penalty is tolerated (much smaller compared to the TDMA), the PD arbiter is an ideal arbiter for the multiple HRT systems as well as the single HRT mixed critical system. The analysis for the symmetric multiprocessing is left for the future work.

7. ACKNOWLEDGMENTS

This work was funded by German BMBF projects ECU (13N11936) and Car2X (13N11933).

8. REFERENCES

[1] Flexray communications system.

[2] B. Akesson, L. Steffens, E. Strooisma, and K. Goossens. Real-time scheduling using credit controlled static priority arbitration. In *RTCSA '08.*

[3] R. Bourgade, C. Rochange, M. De Michiel, and P. Sainrat. Mbba: a multi-bandwidth bus arbiter for hard real-time. In *Embedded and Multimedia Computing (EMC), 2010 5th International Conference on*, pages 1–7. IEEE, 2010.

[4] C.-H. Chen, G.-W. Lee, J.-D. Huang, and J.-Y. Jou. A real-time and bandwidth guaranteed arbitration algorithm for soc bus communication. In *Design Automation, 2006. Asia and South Pacific Conference on*, pages 6–pp. IEEE, 2006.

[5] G. Gebhard. Timing Anomalies Reloaded. In *WCET 2010*, Dagstuhl, Germany.

[6] J. Gustafsson *et al.* The Mälardalen WCET benchmarks – past, present and future.

[7] J. Jalle, L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. Bus designs for time-probabilistic multicore processors. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, pages 50:1–50:6, 3001 Leuven, Belgium, Belgium, 2014. European Design and Automation Association.

[8] T. Kelter, T. Harde, P. Marwedel, and H. Falk. Evaluation of resource arbitration methods for multi-core real-time systems. In *OASIcs-OpenAccess Series in Informatics*, volume 30. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

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

[10] K. Lahiri, A. Raghunathan, and G. Lakshminarayana. The lotterybus on-chip communication architecture. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 14(6):596 –608, june 2006.

[11] M. Paolieri, E. Quiñones, F. J. Cazorla, G. Bernat, and M. Valero. Hardware support for wcet analysis of hard real-time multicore systems. *SIGARCH Comput. Archit. News*, 37:57–68, June 2009.

[12] C. Pitter and M. Schoeberl. A real-time java chip-multiprocessor. *ACM Transactions on Embedded Computing Systems (TECS)*, 10(1):9, 2010.

[13] F. Poletti, D. Bertozzi, L. Benini, and A. Bogliolo. Performance analysis of arbitration policies for soc communication architectures. *Design Automation for Embedded Systems*, 8(2-3):189–210, 2003.

[14] T. Richardson *et al.* A hybrid soc interconnect with dynamic tdma-based transaction-less buses and on-chip networks. In *VLSI Design, 2006. Held jointly with 5th International Conference on Embedded Systems and Design., 19th International Conference on*, page 8 pp., 3-7 2006.

[15] H. Shah, A. Coombes, A. Raabe, K. Huang, and A. Knoll. Measurement based wcet analysis for multi-core architectures. In *RTNS 2014, Versailles, France.*

[16] H. Shah, A. Raabe, and A. Knoll. Priority division: A high-speed shared-memory bus arbitration with bounded latency. In *Proc. DATE*, 2011.

[17] H. Shah, A. Raabe, and A. Knoll. Challenges of wcet analysis in cots multi-core due to different levels of abstraction. *Workshop on High-performance and Real-time Embedded Systems (HiRES 2013)*, 2013.

[18] H. Shah *et al.* Timing Anomalies in Multi-core Architectures due to the Interference on the Shared Resources. In *ASP-DAC*, 2014.

[19] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '95, New York, NY, USA, 1995. ACM.

[20] M. Steine *et al.* A priority-based budget scheduler with conservative dataflow model. In *Proc. DSD*, 2009.

[21] R. Wilhelm *et al.* The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3), 2008.

[22] R. Wilhelm *et al.* Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. 28(7), 2009.