Bounding WCET of Applications Using SDRAM with Priority Based Budget Scheduling in MPSoCs

Hardik Shah
ForTISS GmbH
Guerickestr.25, 80805 Munich
Email: shah@fortiss.org

Andreas Raabe
ForTISS GmbH
Guerickestr.25, 80805 Munich
Email: raabe@fortiss.org

Alois Knoll
Institut für Informatik VI
Technical University Munich
85748 Garching

***Abstract*—SDRAM is a popular off-chip memory that provides large data storage, high data rates, and is in general significantly cheaper than SRAM. There is a growing interest in using SDRAMs in safety critical application domains like aerospace, automotive and industrial automation. Some of these applications have *hard real-time* requirements where missing a deadline can have devastating consequence. Before integrating any hardware or software in this type of system it needs to be proven that deadlines will always be met. In practice, this is done by analyzing application's timing behavior and calculating its Worst Case Execution Time (WCET). SDRAMs have variable access latencies depending on the refresh operation and the previous accesses. This paper builds on hardware techniques such as bank interleaving and applying Priority Based Budget Scheduling (PBS) to share the SDRAM among multiple masters. Its main contribution is a technique to bound the WCET of an application accessing a shared SDRAM of a multicore architecture using the worst case access pattern. We implemented and tested an overall memory system on an Altera Cyclone III FPGA and applied the proposed WCET estimation technique. The results show that our technique produces safe and low WCET bounds.**

I. Introduction

Nowadays, many embedded applications are running on multi-core architectures because of their superior performance per energy ratio. In these architectures, resources are shared among multiple masters to reduce cost. An off-chip memory (slave) is one of the most common resources which is shared between multiple cores (masters). For obvious reasons the off-chip memory should be large to accommodate code and data of multiple applications running concurrently on different cores. It should also provide high data rates to be able to service multiple masters in a reasonable time and it should be cost efficient. The SRAM is an ideal off-chip memory considering its specification, but it is very expensive. The SDRAM provides reasonable speed, price and size. The drawback of SDRAM is that it has variable access latency which depends upon the refresh operation and the previous access.

Safety critical embedded systems need to be certified (e.g. according to DO178B in the avionics domain). In such applications, tasks have to satisfy Hard Real Time requirements to adhere to their specification. This property needs to be proven to enable certification. To guarantee that the deadline will be always achieved, timing analysis of the system needs to be performed considering the *worst possible* behavior of the system's hardware and software components. This conservative approach for the WCET analysis yields safe upper bounds on execution times, but sacrifices resources and/or power since it requires over dimensioning of the hardware.

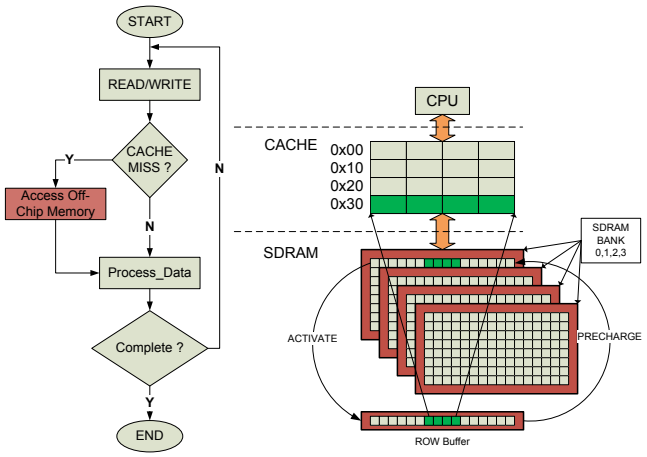
This paper introduces an approach to bound the WCET of an application using its memory trace pattern. The WCET is calculated in isolation considering the worst behavior of the co-executing tasks. Moreover, we also provide the Observed Execution Time (OET) and the Best Case Execution Time (BCET) of the application running on the proposed architecture. Our method was tested by running 6 hardware tasks on an Altera Cyclone III FPGA and the obtained WCET was 1.003 to 3.4 times the OET depending upon master's priority. In Sec. V, we briefly explain a technique to achieve a predictable behavior from a commercial SDRAM controller. In this paper, we are assuming absence of timing anomalies in the processor architecture. Nevertheless, in Sec. IV-A we highlight a solution to calculate the WCET using our method in their presence.

II. Related Work

There has been a number of attempts to provide the WCET of an application on a particular multicore architecture due to the growing importance of static execution time analysis. PRET [1] is a cycle accurate repeatable time machine. Tremendous effort has been spent in making a core with repeatable execution time. The off-chip memory access latency is assumed to be constant: 50ns. PREDATOR [2] is an EU funded project that applies a holistic approach investigating the entire spectrum (architecture, compiler, operating system, software development, tooling etc) of the system design to come up with a predictable and an efficient system solution. Both these projects do not analyze SDRAM for WCET analysis.

MERASA [3] is another EU funded project that targets to build a predictable and an efficient multi-core architecture for mixed critical applications. Predictability on the core side is achieved by giving the highest priority to the Hard Real-time Task (HRT) while on the shared bus side access latency is bounded by a Round Robin (RR) arbiter. The SDRAM is accessed through Analyzable Memory Controller (AMC) [4]. The AMC uses Bank Interleaving (BI) to access the SDRAM. Through theoretical analysis, latency parameters are extracted to calculate the WCET of an application. The AMC assumes the maximum of Read/Write and Write/Read switching latencies as a constant worst case latency of every access. Such assumption, while making analysis simple, cannot produce

978-3-9810801-8-6/DATE12/©2012 EDAA



(a) Application Execution (b) Cache Line Mapping on SDRAM

Fig. 1

precise bounds as we will show later. Moreover, the RR policy with one request per master cannot satisfy the need of different bandwidth requirements. If more than one request per master is assigned, the WCET is severely degraded [5].

In [6], authors propose a periodic software task that refreshes the SDRAM in burst fashion and thus removes the asynchronism related to the hardware controlled refresh. This technique will be helpful only for single core architectures. In multi-core architectures, except the core which is running the *refresh task*, other cores will see it as an asynchronous refresh.

Heithecker et al [7] proposes BI with two level arbitration using the RR scheme to bound the maximum latency of requesters. To provide a low latency and bandwidth guarantee to each requester, requesters are assigned high and standard priorities. High priority requesters are scheduled out of order in the RR queue. After each high priority request, one standard priority request is scheduled to avoid starvation. The approach excludes SDRAM latency analysis and inherits drawbacks of RR scheme as mentioned before.

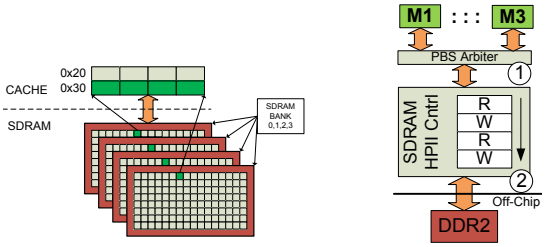
Akesson et al [8] propose a predictable shared memory controller that uses BI to access the SDRAM. High priority is assigned to masters executing latency sensitive applications while masters requiring high bandwidth are assigned higher budget. Thus, the coupling between latency and rate is removed using Credit Controlled Static Priority [9] - CCSP arbiter. CCSP is a class of Priority Based Budget Scheduler (PBS) [10]. We extend this work by providing WCET analysis technique for the PBS arbiter. Moreover, we provide the OET and the BCET to evaluate the precision of our results and the variability [11] of the PBS arbitration scheme.

III. Background

Before we begin with our approach, some fundamentals are explained in this section.

A. Application Execution

Fig. 1a abstracts general behavior of an application execution. At first, application code/data is read or written. If the accessed data is on-chip (cache, registers etc) it is processed immediately. Otherwise, the data is fetched from the off-chip



(a) Cache Line Mapping in BI (b) SDRAM Controller

Fig. 2

memory (SDRAM) and then processed. The off-chip memory is accessed via a cache. The Cache is organized in the form of cache lines as shown in Fig. 1b. When a cache miss/write-back occurs, entire cache line (here 32 bytes) is read from or written to the SDRAM. The SDRAM is accessed only when a cache miss or a cache write-back occurs. It stays idle during on-chip operations. These idle times are denoted in clock cycles as `OnChipProcTime[i]` $\in Z^+$, `i` $\in [0, TotalAccesses)$. `TotalAccesses` is the total number of off-chip memory accesses during the execution in the worst case.

B. SDRAM Fundamentals

SDRAMs have a bank architecture where each bank stores data in a 2D array of rows and columns as shown in Fig. 1b. Cache lines are mapped to the rows of the banks. Each row contains multiple cache lines. Before reading or writing a cache line, the entire row containing that cache line is copied into the row buffer. This operation is called *Row Activation* (ACT). Data is now read or written from the row buffer in a burst fashion with low latency. When the data from another row is requested, the content of the buffer is copied back to the bank. This operation is called Precharge (PCH). Subsequently, the new row is activated. At compile time, it is unknown which row is in the row buffer at a certain point of the execution. Being conservative, an absence of currently accessed row in the buffer is assumed for the worst case latency calculation. This assumption increases the WCET bound of the application. Moreover, SDRAMs have to be refreshed periodically. During a refresh operation the SDRAM cannot be accessed. Since the refresh operation is controlled by a hardware its occurrence time is unknown to the application. This uncertainty about the refresh further degrades WCET bounds.

C. Bank Interleaving

To mitigate the penalty associated with row activation/precharge command bank interleaving (BI) can be applied. BI splits all off-chip accesses into smaller chunks such that their total number equals the number of banks of the SDRAM device. Each of these chunks is executed on unique bank and the bank is precharged at earliest possible time. While data is provided from one bank, another bank can be activated in the background. The resulting cache mapping to the SDRAM is depicted in Fig. 2a.

To facilitate BI, Additive Latency (AL) and Auto Precharge (Auto PCH) commands are included in the standard. When AL is non zero, Read/Write command can follow ACT command without waiting for tRCD time. SDRAM device holds

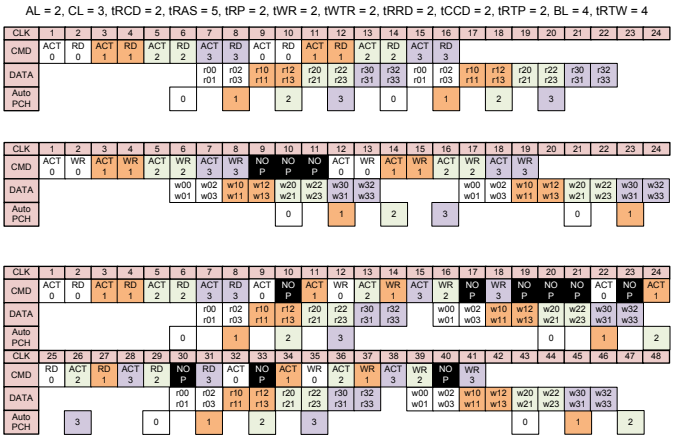


Fig. 3: (a) Continuous Read (b) Continuous Write (c) Alternating Read/Write. BI Timing

Read/Write command for AL clock cycles before applying it internally. Thus, command bus utilization can be maximized. Auto PCH command has to be signaled when applying read-/write command. When Auto PCH is signaled, the device starts prechare operation at earliest possible time automatically. Thus, explicit *Precharge* command is not required and command bus utilization is maximized further.

Fig. 3 explains access timing of the SDRAM for three cases: continuous read, continuous write and alternating read/write. In the figure the numbers associated with Activate (ACT), Auto Precharge (Auto PCH), Read (Rd) and Write (Wr) denote the number of the bank on which these commands are applied. For data bus 'r' and 'w' denote read and write data. The most significant bit of data denotes the number of the bank and the least significant bit denotes the number of data element.

In Fig. 3(a) Bank0 is activated and immediately read command is issued (2^{nd} clock). After the Rd command is issued data is available on the data bus after AL+CL clock cycles (7^{th} clock). The prechage operation begins automatically at AL+BL/2 clock cycles(on 6^{th} clock) after the read command. The Bank0 can be activated again, at the earliest, after tRP (2) clock cycles(on 8^{th} clock) from the precharge. It is clear from the figure that for a seamless data transfer, each bank must be activated after every 8 clock cycles. Example shown in Fig. 3(a) achieves this deadline to get seamless data transfer for a read operation and 100% command and data bus utilization.

For a write transfer (Fig. 3(b)), data is put on the data bus AL+CL-1 clock cycles after the Wr command is issued to the bank (6^{th} clock). The precharge operation starts tWR clock cycles after (10^{th} clock) the last element of data is put on the data bus. The delayed precharge delays next earliest activation of the Bank0 (12^{th} clock) due to the tRP requirement. Thus, voids are created on the command and data bus.

For an alternating Rd/Wr transfer shown in Fig. 3(c), the void at clock 10 is caused by *Read to Write Turn Around Time*, tRTP, which defines minimum time from the Rd command to the Wr command. Because of Rd command issued at clock 8 to the Bank3, Wr command cannot be issued before 12^{th} clock cycle. Again, void is created at the clock 17 due to tCCD requirement which states the minimum distance between any two Rd or Wr commands. Three voids created at 19, 20 and 21 are the same as of continues write transfer and has already been explained in the previous paragraph. The void created at clock 23 is because of minimum Wr to Rd command spacing which is defined as CL-1+BL/2+tWTR (internal write to read command delay). Voids created at 30, 33, and 40 have the same reasons as of 17, 10 and 17 respectively. It is clear from the figures that alternating accesses to the SDRAM create the worst latencies than continues accesses. For further detail on the timing parameters readers are referred to [12].

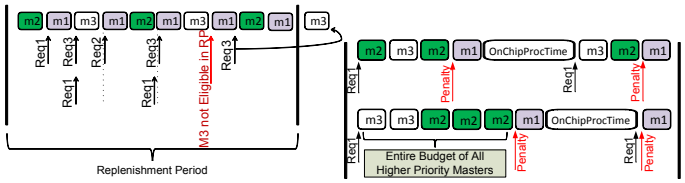
Here we define three parameters Worst Case Write Command Width ($W_cWrCmdWidth$), Worst Case Read Command Width ($W_cRdCmdWidth$) and Worst Case Read Latency (W_cRdLat). As shown in the figure, in the worst case, write operation on command bus starts at clock 9 and completes at clock 18. We define this width as $W_cWrCmdWidth = 10$. The same way, read operation on command bus starts at 19 and completes at 31. We define this width as $W_cRdCmdWidth = 13$. Since in read operation data is read from the SDRAM, the requester has to wait after issuing the Rd command. In the worst case, as shown in figure, after issuing Rd command on clock 31, entire data is received on clock 37. We define this time from clock 31 to 37 as $W_cRdLat = 6$. Thus in the worst case, write operation is 10 clock cycles long, while read operation is 13+6=19 clock cycles long. Clearly, to achieve tight WCET bounds both read and write accesses have to be analyzed separately.

IV. Priority Based Budget Scheduling

In the Priority Based Budget Scheduling (PBS) masters are assigned fixed priorities and fixed budgets at design time [10]. The priority relates to a master's latency requirements and the budget relates to its bandwidth requirements. Access to the shared resource is granted to the requesting master which has highest priority provided that it still has a budget left. When the granted master does a transfer its budget is reduced by one. When a master's budget becomes zero, that master is termed not eligible and cannot be granted the shared resource until the next replenishment period starts. At the beginning of a replenishment period, each master receives its original budget back. In practice, masters with low budget is assigned high priority and masters with high budget is assigned low priority. The replenishment period is kept long enough to accommodate the number of requests, which is equal to the total budget of all masters, in the worst case. Equation (1) is used to calculate the worst case command width W_cCmdWd considering alternating read/write accesses. Equation (2) gives the size of a replenishment period in clock cycles.

$$W_cCmdWd = \left\lceil \frac{W_cRdCmdWd + W_cWrCmdWd}{2} \right\rceil \quad (1)$$

$$R_p = W_cCmdWd \times \sum_{i=1}^{N} Budget[i] \quad (2)$$



(a) PBS: Budget(m3,m2,m1)=(2,3,5); (b) Worst Case Access Latency for M1 Priority(m3,m2,m1)=(1,2,3)

Fig. 4: Priority Based Budget Scheduler

A. Worst Case Interference

Fig. 2b depicts the architecture and Fig. 4a illustrates PBS operation with three masters. Here, master3-m3 is assigned the highest and master1-m1 is assigned the lowest priority. The highest priority master is served with minimum latency as long as it has budget left. When it consumed its entire budget, it is no longer eligible to access until the beginning of the next replenishment period. Since the PBS is non-preemptive, the highest priority master has to wait for only one on-going low priority request even in the worst case. This is illustrated in Fig. 4a: m3 waits until m1 has finished (dotted line).

In Fig. 4b, `OnChipProcTime` (Sec. III-A) of m1 is included for considering the worst case latencies. As shown in the upper portion of the figure, m1 requests in the beginning of the replenishment period, but it is interfered by a sequence of accesses from the higher priority masters. For the worst case analysis it has to be assumed that these accesses are of the alternating type and thus cause the longest possible access latency (Sec. III-C). Moreover, when m1 wants to do a *Read* transfer the alternating sequence ends with a *Write* transfer and vice versa. Thus, in the worst case m1 has to consider the read/write switching penalty additionally (red arrow). After considering these latencies, m1 can do a transfer. Before doing the next transfer m1 does an onchip operation. When this is finished, m1 requests another access, which is again interfered by a sequence of higher priority alternating accesses. In PBS, an earlier access request cannot produce longer over all execution time than had it been requested later. Thus, PBS does not introduce timing anomaly [13] and since it is assumed that the architecture is free from timing anomalies, commutativity of accesses in the replenishment period can be considered. The lower portion of the Fig. 4b shows the re-arranged version of accesses shown in the upper portion. For simplicity of worst case analysis, it can be assumed that the first access in the replenishment period of the lowest priority master will be interfered by all higher priority masters with exploitation of their entire budget. For the subsequent accesses in the replenishment period, the lowest priority master is granted immediately since all other masters do not have a budget left. Only the Rd/Wr switching penalty has to be considered as shown in the lower portion of the Fig. 4b. The same way interference for higher priority masters can be calculated. We denote the First Access of Master in a Replenishment Period as FA_m^{rp} and subsequent accesses as Not First Access of Master in Replenishment Period (NFA_m^{rp}). In the worst case, FA_m^{rp} of a master will be interfered by all higher priority masters with exploitation of their entire budgets and one on-going lower priority transfer. For NFA_m^{rp} interference from only one on-going lower priority transfer has to be considered. Clearly, for any transfer the highest priority master has to consider interference from only one on-going lower priority transfer. Equation (3) defines the Worst Case First Access Interference in a Replenishment Period for master M which we denote by $W_cFAI_m^{rp}$. Equation (4) defines the Worst Case Not First Access Interference in a Replenishment Period for master M ($W_cNFAI_m^{rp}$). The master with index N has the highest and index 1 has the lowest priority.

Algorithm 1 Algorithm to Calculate WCET of Single Access

1: //Calculate $W_cFAT_m^{rp}$ when $FA_m^{rp} = write$
2: **if** $(W_cFAI_m^{rp} + 1)$ is even **then**
3: $acc = (W_cWrCmdWd + W_cRdCmdWd) \times (W_cFAI_m^{rp} + 1)/2$
4: **else**
5: $acc = (W_cWrCmdWd + W_cRdCmdWd) \times (W_cFAI_m^{rp})/2$
6: $acc += W_cWrCmdWd$
7: **end if**
8: $W_cFAT_m^{rp} = acc$

9: //Calculate $W_cFAT_m^{rp}$ when $FA_m^{rp} = read$
10: //execute lines 1–8;
11: $W_cFAT_m^{rp} += W_cRdLat$

12: //Calculate $W_cNFAT_m^{rp}$
13: **if** $m \neq 1$ **then**
14: **if** $NFA_m^{rp} = read$ **then**
15: $W_cNFAT_m^{rp} = W_cWrCmdWd + W_cRdCmdWd$
16: $W_cNFAT_m^{rp} += W_cRdLat$
17: **else**
18: $W_cNFAT_m^{rp} = W_cRdCmdWd + W_cWrCmdWd$
19: **end if**
20: **else**
21: **if** $NFA_m^{rp} = read$ **then**
22: $W_cNFAT_m^{rp} = W_cRdCmdWd + W_cRdLat$
23: **else**
24: $W_cNFAT_m^{rp} = W_cWrCmdWd$
25: **end if**
26: **end if**

$$W_cFAI_m^{rp} = \begin{cases} 1, & m = N \\ \sum_{i=m+1}^{N} Budget[i] + 1, & m \in (1, N) \\ \sum_{i=2}^{N} Budget[i], & m = 1 \end{cases} \quad (3)$$

$$W_cNFAI_m^{rp} = \begin{cases} 1, & m \in (1, N] \\ 0, & m = 1 \end{cases} \quad (4)$$

In Algorithm 1, lines 1–11 detail calculation of Worst Case First Access Time in a Replenishment Period for master M, denoted as $W_cFAT_m^{rp}$. Addition of 1 in $W_cFAI_m^{rp}$ at line 2 in the Algorithm considers the first access of master m as the last access in the sequence of alternating reads and writes. On line 16 of the algorithm W_cRdLat is added, since acc only contains the worst case time required to issue the *Read* command to the SDRAM. The same way worst case time for subsequent accesses is computed in lines 12–26.

For the WCET analysis, `OnChipProcTime` plays an important role. It exists between any two consecutive accesses to an SDRAM and it can be obtained from a third party WCET

Algorithm 2 Algorithm to Calculate WCET of Application Running on Master m

1: $budget, acc = 0$
2: **repeat**
3: **if** $budget = 0$ **then**
4: $acc += W_cFAT_m^{rp} + OnChipProcTime[i]$
5: **else**
6: $acc += W_cNFAT_m^{rp} + OnChipProcTime[i]$
7: **end if**
8: $budget++$
9: **if** $budget = cBUDGET$ **then**
10: $WCET += R_p$
11: $acc, budget = 0$
12: **end if**
13: **if** $acc \geq R_p$ **then**
14: $WCET += R_p$
15: $acc = acc - R_p$
16: $budget = 0$
17: **end if**
18: **until** $i \leq TotalAccesses$

19: $WCET += acc$
20: $REFRESH = \lceil WCET/tREFI \rceil$
21: $WCET = (REFRESH + 1) \times tRFC$

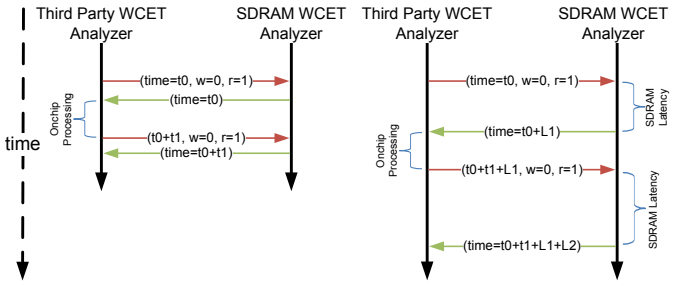


Fig. 5: (a) Memory Trace (b) Adding SDRAM Latency

analyzer. The WCET analyzer analyzes the application binary code and predicts the application execution in the worst case. From the WCET analyzer, memory access trace of each cache miss and the worst case time of its occurrence can be extracted.

In Fig. 4b, because of its long `OnChipProcTime`, m1 can do only 2 transfers in the replenishment period though its budget is 5. As shown in the figure, the $3rd$ transfer of m1 would fall in the next replenishment period. Here, all the higher priority masters would regain their original budget and may interfere with exploitation of their entire budget. Hence, $W_cFAI_1^{rp}$ of m1 has to be considered. Certainly, such `OnChipProcTime` exists in higher priority masters as well. But while analyzing in isolation, worst behavior from others must be assumed. Thus, in the worst case, the lowest priority master will not be able to use its allocated bandwidth entirely. Therefore, the application executing on it may miss the deadline or the QoS will be degraded.

As shown in Fig. 5(a) it is assumed that the memory trace ignoring SDRAM latency is available for all execution paths from the third party WCET analyzer. Afterwards, as described in Algorithm 2, for every access worst case access latency and `OnChipProcTime[]` are accumulated. This is done until the master's budget is exhausted or the accumulated time exceeds the replenishment period. In the first case master has to wait till the beginning of the next Replenishment period and in the second case the current replenishment period has already completed. Thus, time duration of one replenishment period is added to the WCET (lines 10,14). Refresh penalty tRFC is added considering the average number of refreshes. An additional tRFC penalty, at line 21, is to consider an interfering refresh for the very first access of the application.

For BCET calculation, it needs to be assumed that the application under consideration is the only application running on the multi-core architecture. Thus there is no interference from other masters. Moreover, it has to be assumed that the refresh occurs when application is in `OnChipProcTime`. Thus not a single refresh interferes with the application execution.

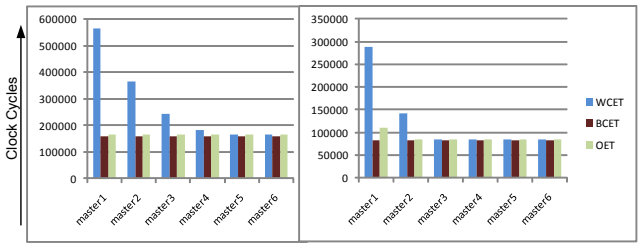
In the presence of timing anomalies, our technique can be implemented as a software API. For each access, a third party WCET analyzer calls the API considering a range of all possible $[B_c, W_c]$ values of the `OnChipProcTime`. The API then returns range of $[B_c, W_c]$ latencies for each possible value of `OnChipProcTime`. Then the WCET analyzer has to pick the combination of `OnChipProcTime` and latency value which produces the longest over all execution time of the program. Thus, the core technique of calculating latencies does not change, only computation of WCET becomes exhaustive which is a typical case in presence of timing anomalies.

V. Test Setup

We connected six hardware traffic generators to an SDRAM controller running on an Altera Cyclone III FPGA, the same way as shown in Fig. 2b. The Altera High Performance Controller II (HP II) does not support BI in the native mode. Therefore BI was implemented using access splitting and user controlled auto precharge [14]. The HP II contains internal FIFOs for commands and data. These FIFOs affect access latencies depending upon their status. In the case of a *write* transfer, the master proceeds with its operation while command and data move from point (1) to point (2) in Fig. 2b and are subsequently written to the SDRAM. For read transfers, the time required for a *read* command to proceed from (1) to (2) must be considered. After arriving at (2), latencies of Fig. 3 have to be considered.

To obtain safe values of worst case latency parameters, we generated continuous alternating traffic to the SDRAM and extracted latency parameters $W_cRdCmdWd$, $W_cWrCmdWd$ and W_cRdLat at (1) in the Fig. 2b using an on-chip logic analyzer (SignalTap II). Since (1) is the only visible point to the user, it is safe to extract all latency parameters from there only. Using Signaltap II, we also found that the Altera SDRAM controller issues a refresh every $tREFI \pm 20$ clock cycles depending upon ongoing transfers. To avoid this variability, we implemented a user controlled refresh circuit [14] that closes all channels at point (1) slightly before tREFI to empty the SDRAM controller FIFO. When the tREFI timer expires, a refresh is applied to the SDRAM immediately. This way a precise issuing of the refresh in tREFI intervals is achieved at very low cost of SDRAM bandwidth. We tested our setup at 125 MHz SDRAM clock frequency.

Assuming the worst case path is known, we generated two types of memory traces for investigating different traffic



(a) Equal Density Traffic. (b) Incremental Density Traffic.

Fig. 6: WCET, BCET and OET in Number of Clock Cycles

scenarios. (1) Equal Density Traffic, where each master generates 2K random alternating accesses to the SDRAM. Average traffic density is identical for all masters. (2) Incremental Density Traffic, where each master generates alternating random accesses such that the traffic density is proportional to its budget. Moreover, the total number of accesses also increases proportionally with traffic density (cf. Table I). In both the tests, master6 has the highest and master1 has the lowest priority. Since in both the cases masters generate alternating accesses, high interference occurs, but the worst case is not guaranteed due to the access randomization. We estimated how much time will be required by each master to complete its transfers in the worst as well as the best case.

	Average `OnChipProcTime[]`		Budget		Total Accesses		Priority P_m
	Eq. Density	Incr. Density	Eq. Density	Incr. Density	Eq. Density	Incr. Density	
master1	2^3	2^0	2^2	2^5	2048	$2^5 \times 100$	6
master2	2^3	2^1	2^2	2^4	2048	$2^4 \times 100$	5
master3	2^3	2^2	2^2	2^3	2048	$2^3 \times 100$	4
master4	2^3	2^3	2^2	2^2	2048	$2^2 \times 100$	3
master5	2^3	2^3	2^2	2^1	2048	$2^1 \times 100$	2
master6	2^3	2^4	2^2	2^0	2048	$2^0 \times 100$	1

TABLE I: Tested Traffic Pattern

VI. Results

For these tests, WCET, BCET and Observed Execution Time (OET) are depicted in Fig. 6. Due to the nature of PBS which prefers a high priority master over a lower priority master, worst case interference experienced by a high priority master is significantly less than that experienced by a low priority master. The estimated WCET is 1.003 times and 3.4 time the OET for the highest and the lowest priority masters respectively. It is clear from the graphs that for high priority masters BCET, WCET and OET are similar. For low priority masters WCET increases drastically due to the conservative assumption of permanently interfering high priority masters. The OET of the lowest priority master in Fig. 6(b) is also slightly more than others though traffic produced by each master is proportional to its budget. Therefore, it can be concluded, that our analysis approach lends itself well to shared SDRAMs using PBS. Nevertheless, the PBS itself does not allow tight bounding of WCET of low priority masters.

VII. Conclusion and Future Work

In this paper, we introduce a method to calculate the WCET of tasks when an SDRAM is used as a shared memory. The WCET is calculated using worst case memory access traces produced by a third party WCET analyzer for each execution path. No knowledge of the behavior of tasks running on other masters is assumed. We analyzed and tested the worst case behavior of a Priority Based Budget Scheduled (PBS) SDRAM on an Altera Cyclone III FPGA. From the memory traces of an individual master, each access is analyzed for its worst possible interference considering the priority of the master, its assigned share in one replenishment period and the total share of higher priority masters. Since PBS favors a high priority master over a low priority master, the WCET produced for the highest priority master is much less than that of the lowest priority master. We conclude that our method can readily be applied to the multi-core architectures with shared SDRAM. In our future work, we will apply this approach in real life multi-core applications and compare performance of various real time capable shared SDRAM arbitration schemes.

VIII. Acknowledgment

This work was supported by the European funded project RECOMP (Reduced Certification Costs Using Trusted Multi-core Platforms) from ARTEMIS JOINT UNDERTAKING (JU) 2009 Program, grant agreement no.: 100202.

References

[1] B. Lickly, I. Liu, S. Kim, H. D. Patel, S. A. Edwards, and E. A. Lee, "Predictable programming on a precision timed architecture," in *Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems*, ser. CASES '08. New York, NY, USA: ACM, 2008, pp. 137–146.

[2] "Predator project." [Online]. Available: http://www.predator-project.eu/

[3] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quinones, M. Gerdes, M. Paolieri, J. Wolf, H. Casse, S. Uhrig, I. Guliashvili, M. Houston, F. Kluge, S. Metzlaff, and J. Mische, "Merasa: Multicore execution of hard real-time applications supporting analyzability," *IEEE Micro*, vol. 30, pp. 66–75, September 2010.

[4] M. Paolieri, E. Quinones, F. J. Cazorla, and M. Valero, "An analyzable memory controller for hard real-time cmps," *IEEE Embedded Systems Letters*, vol. 1, pp. 86–90, 2009.

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

[6] B. Bhat and F. Mueller, "Making dram refresh predictable," in *Real-Time Systems (ECRTS), 2010 22nd Euromicro Conference on*, july 2010.

[7] S. Heithecker and R. Ernst, "Traffic shaping for an fpga based sdram controller with complex qos requirements," in *Proceedings of the DAC*. New York, USA: ACM, 2005.

[8] B. Akesson, K. Goossens, and M. Ringhofer, "Predator: a predictable sdram memory controller," in *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, ser. CODES+ISSS '07. New York, NY, USA: ACM, 2007.

[9] B. Akesson, L. Steffens, and K. Goossens, "Efficient service allocation in hardware using credit-controlled static-priority arbitration," in *Proceedings of the 2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, ser. RTCSA '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 59–68.

[10] M. Steine, M. Bekooij, and M. Wiggers, "A priority-based budget scheduler with conservative dataflow model," in *Digital System Design, Architectures, Methods and Tools, 2009. DSD '09. 12th Euromicro Conference on*, aug. 2009, pp. 37 –44.

[11] R. Kirner and P. Puschner, "Time-predictable computing," in *Proceedings of the 8th IFIP WG 10.2 international conference on Software technologies for embedded and ubiquitous systems*, ser. SEUS'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 23–34.

[12] "Ddr2 sdram specification. jesd79-2f." [Online]. Available: http://www.jedec.org/

[13] G. Gebhard, "Timing Anomalies Reloaded," in *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*, B. Lisper, Ed., vol. 15, Dagstuhl, Germany, 2010, pp. 1–10.

[14] "Ddr and ddr2 sdram controller with altmemphy ip user guide, external memory interface handbook." [Online]. Available: http://www.altera.com/