An SDRAM controller for real-time systems

Lakis, Edgar; Schoeberl, Martin

Published in:
2013 IEEE 16th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)

Link to article, DOI:
10.1109/ISORC.2013.6913224

Publication date:
2013

Document Version
Peer reviewed version

An SDRAM Controller for Real-Time Systems

Edgar Lakis, Martin Schoeberl

Department of Applied Mathematics and Computer Science
Technical University of Denmark
Email: edgar.lakis@gmail.com, masca@imm.dtu.dk

***Abstract*—For real-time systems we need to statically determine worst-case execution times (WCET) of tasks to proof the schedulability of the system. To enable static WCET analysis, the platform needs to be time-predictable. The platform includes the processor, the caches, the memory system, the operating system, and the application software itself. All those components need to be timing analyzable.**

Current computers use DRAM as a cost effective main memory. However, these DRAM chips have timing requirements that depend on former accesses and also need to be refreshed to retain their content. Standard memory controllers for DRAM memories are optimized to provide maximum bandwidth or throughput at the cost of variable latency for individual memory accesses. In this paper we present an SDRAM controller for real-time systems. The controller is optimized for the worst case and constant latency to provide a base of the memory hierarchy for time-predictable systems.

I. Introduction

The use of modern, conventional architectures in real-time systems (RTS) requires complex analysis and suffers from high resource over-allocation, needed to cover uncertainties stemming from employed speculative, average-case optimizations. The design of time-predictable, RTS optimized architectures that allow easy timing analysis and tight timing guaranties is an active research topic [7], [8], [19].

The goal of this paper is to develop a time-predictable SDRAM controller for the T-CREST platform.[1] The T-CREST project is an ongoing research project supported by the European Union's 7th Framework Programme, aiming to develop a homogeneous time-predictable multi-processor platform. The variable SDRAM access latencies pose some challenges for its effective use in RTS. The many-core T-CREST platform creates a new context for rethinking the previous results and finding new solutions for external memory access.

We implemented a working prototype of a SDR (Single Data Rate) SDRAM controller and integrated it with a processor. We evaluate the controller by comparing it against a series of other SDRAM controllers available for FPGAs. The strict optimization for the worst-case results also in a simple design that consumes less hardware resources than controllers that implement average-case optimizations. Furthermore, the controller can be clocked higher then the available single data rate SDRAM chips and is therefore not a bottleneck of the maximum operation frequency.

[1]Time-predictable Multi-Core Architecture for Embedded Systems (T-CREST), see http://www.t-crest.org/

The paper is organized as follows: The following Section presents related work in the area of SDRAM controllers for real-time systems and approaches to handle the DRAM refresh. Section III gives background information on SDRAM memories. We present the design of the time-predictable SDRAM controller in Section IV and show the evaluation results in Section V. Section VI concludes the paper

II. Related Work

Akesson et al. proposes the predictable controller called *Predator* [1], [2]. To improve data bus utilization, the large memory transfers are pipelined through bank interleaving with automatic precharge after each access (i.e., closed page policy). The controller uses a hybrid approach between the static SDRAM command scheduling, which is good for providing timing guaranties, and the dynamic command scheduling, which provides better average-case memory bandwidth utilization. The elementary operation size is fixed and the sequences of correctly interleaved SDRAM commands to perform a read or write of elementary block are precomputed at design. The patterns and their compositions are analyzed at design time, to derive worst-case response times (WCRT) for each memory operation. The refresh is handled by including it in the WCRT of each memory operation, but accounting for the refresh period for transfers of larger sizes. To allow a better average-case performance, the memory requests are translated dynamically into sequences of static patterns that are executed by the configurable SDRAM command generator. For hard-RT tasks all the responses are delayed to their worst-case latency to provide isolation between different requestors.

Paolieri et al. describe the *Analyzable Memory Controller (AMC)* [14] that was part of MERASA project [21] for predictable multi-core architectures. The bank interleaved command sequences are used as in the Predator. Fine-grained round-robin arbitration is used for hard-RT tasks. They get higher priority than non-HRT tasks. The WCRT is calculated by using the maximum possible time needed for a single transfer multiplied by the number of possible colliding requesters (i.e. one pending non-HRT task and all other hard-RT tasks). Using the single maximum transfer estimate increases the WCRT beyond the value that is possible in the worst case. For example, the worst-case command is usually a write invoked after a read. But it is impossible to have N write-after-read switches in a sequence of N transfers. The same authors propose improvements of the AMC [13]. They account

for maximum possible direction switches and additionally allow preempting the non-HRT transfer at the bank boundary, saving few additional latency cycles. The measurement based worst-case execution time (WCET) estimates for the tasks are used and refresh is handled by synchronizing the start of the task with the refresh operation, to have refresh interference incorporated into WCET.

Reineke et al. describe the SDRAM controller for the ARM based precision timed architecture (PTARM) processor [15]. The processor implements a four thread-interleaved pipeline, and a separate bank is assigned for each thread, thus conflicts are avoided. Because this privatization removes the concept of a shared memory, PTARM threads use the on-chip memory for shared data. The refresh is performed manually. Because of their tight integration with the interleaved pipeline, they exploit some properties of the architecture while issuing refresh. The refresh is deferred to the end of the read operation where it does not incur additional cost, because the pipeline cannot use two consecutive read slots. For larger memory transfers (DMA transfers to/from the scratchpad) they account for maximal possible interference from refresh.

Atanassov and Puschner [5] described a problem with refresh incorporation into WCET of the task without considering each transfer. They showed, that even though the WCET augmented with the possible refresh interference is safe, the actual WCET path of the program might be different. This does not seem to be a problem in practice, if well behaved architecture without timing anomalies is used.

Bhat and Mueller propose grouping the refresh operations together and executing them in a separate task [6]. This eliminates refresh interference uncertainty, because the refresh interference can be handled by the schedulability analysis. The refresh operation does not have to be pessimistically incorporated into the WCRT of each memory operation. Unfortunately the authors did not mention that there are strong limitations on burst refreshes in later generations of SDRAM memories.

III. SDRAM Background

This section provides the background on Dynamic Random Access Memory (DRAM). In this work, the focus is on Synchronous DRAM (SDRAM), which has been the most prevalent volatile off-chip memory for more than a decade. It is called synchronous to distinguish it from the asynchronous DRAM interfaces that were dominant at the time the standard was created. The JEDEC Solid State Technology Association prepares the standard and several generations of the standards have evolved over the years.

A. Structure and Operation

The DRAM is called dynamic because the value of each bit is represented as a charge of a small capacitor, which discharges due to leakage over time. The capacitors must be refreshed periodically to preserve the valid value. In addition to the capacitor, the bit cell contains a pass transistor, which is enabled when the value is read or written. The bits are not accessible individually; instead, they are organized in arrays of rows×columns. A row must be prepared before a bit of relevant column can be read. This requires two steps: (1) *precharge* – the bit-lines (columns) are charged to midpoint voltage between logical 0 and 1; (2) *activate* – the capacitors of the single row are connected to the bit-lines. During the activation the charge of the capacitor creates a small voltage swing on the bit-line, which is recovered to full voltage by the sense amplifier. Both steps contribute significantly to the latency of the DRAM access, because the bit-line runs over the thousands of rows and has huge capacitance. However,, once the row has been activated its columns can be read/written with lower latency.

To increase the throughput, a memory device contains several independent arrays called *banks*. The data can be transfered from one bank while other banks are precharging or activating. Similarly, few devices can be connected to the same SDRAM interface and enabled by chip-select. This address dimension is called *rank*.

B. Signaling

The controller accesses the memory device through parallel buses, i.e., each signal bit is sent independently at the same clock cycle. The data bus and the control/address bus are independent; this allows sending commands to a different bank during the longer data transfer cycles.

There are no handshaking or acknowledge signals in the interface; instead, the controller obeys the implicit timing parameters of the memory chip to assure proper operation. The timing parameters are presented in Section III-D.

C. Commands

The SDRAM has a synchronous interface and is operated by a predefined set of commands. The four main commands are *precharge*, *activate*, *read* and *write*. The other commands are: *auto refresh* which is discussed in separate subsection; *mode register set* configures the memory; and *burst stop*, which is redundant, and has been removed starting from DDR2 generation.

Precharge (bank or ALL) charges the bit-lines of the specified or all banks to the reference voltage, to enable data recovery by the sense amplifiers during subsequent row activation. The precharge requires t_{RP} time before the *activate* command can be issued.

Activate (bank, row) activates the row of the specified bank. The data of the activated row will be available for *read*/*write* after t_{RCD} time, but restoration of the values into the bit capacitors usually requires more time, specified as t_{RAS}. The bank can not be precharged before that.

Read (bank, column, optional auto-precharge) requests to read a number of words from the active row. The data must be sampled from the data bus after t_{CAC} cycles. The length of the transfer is configured by a *mode register set* command. The device will issue the *precharge* command automatically at earliest allowed time if auto-precharge is enabled for this *read* command. In addition to t_{RCD} mentioned in *activate* command, there might be a constraint on minimal separation

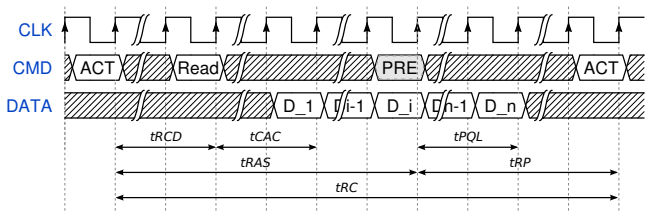

Fig. 1. SDRAM (SDR) read related timing parameters.

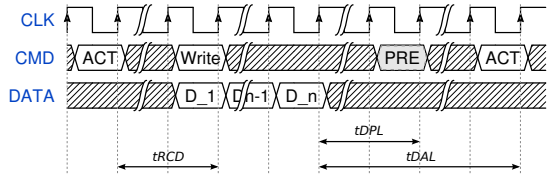

Fig. 2. SDRAM (SDR) write related timing parameters (the t_{RAS}, t_{RP} and t_{RC} are not shown).

between consecutive *read* and *write* commands specified as t_{CCD}.

Write (bank, column, optional auto-precharge) requests to write a number of words into the active row. The controller of an SDR SDRAM starts the data burst together with the command. A *mode register set* command controls the burst length. The length is either one word or the value of the read length. The notes regarding the t_{CCD}, t_{RCD} and auto-precharge of *read* apply to *write* as well. Additional constraints (t_{WTR} and t_{WR}) are required to allow the last data word to be stored properly.

D. Timing Parameters

The timing parameters of the SDRAM device need to be obeyed by the controller. The parameters can be divided into two groups. The first group contains the signal integrity requirements that are present when interfacing any synchronous component. Those are setup/hold requirements for the inputs with respect to clock/strobe signals. The clock-to-output timing are provided for all the outputs and the timing of the tristate buffers needed for bidirectional signals. The other group contains protocol level requirements which describe the separation between commands/data on the SDRAM bus. For example, the sharing of the I/O and control hardware by the banks requires delays between the commands even if they are directed to different banks. Also, the single bidirectional data bus requires extra delay when the direction of the transfer is changed. The extra separation might also be required when getting data from the different ranks [11].

The Figure 1 and Figure 2 show the timing requirements for read and write transactions. The *precharge* command is issued explicitly to show its place in the transaction. The action could also be scheduled for automatic execution by enabling the auto-precharge during *read* and *write* commands.

E. Performing Refresh

There is some flexibility in performing refresh. The methods of invoking a refresh are listed first, followed by the possible ways of organizing them in time.

Activating a row has a side effect of refreshing its capacitors because the sense amplifiers restore the charge. Activation is performed for a row of a single bank, so in case of *refresh by activate* all the banks have to be refreshed separately. Additionally, the controller/software needs to keep track of the row counters. *Auto refresh*[2] is triggered by issuing a dedicated refresh command. The command does not specify the address of the row; instead a counter inside the memory chip points to the row. The same row is updated in all the banks in parallel, and the counter is incremented to point to the row for the next refresh operation. *Self refresh* is an autonomous refresh mode, which can be performed by the SDRAM chip during longer inactivity periods.

There are two strategies regarding the refresh invocation of different rows: distributed and burst refresh. The *distributed refresh* is commonly used. The refresh operations are spread out evenly in time. Alternatively, the *burst refresh* can be used. The longer period without refresh is followed by several refresh operations invoked one after another.

The first generation SDRAM (Single Data Rate) devices usually allow performing the refresh command bursts of arbitrary length. Thus refresh actions for all the rows can be grouped together and invoked at a convenient time to avoid refresh interference with normal operation. The later, Double Data Rate generations are less flexible. The maximum of 8 auto refresh operations can be postponed or pulled (issued in advance), but not both. In another words, the maximum of $9 \times t_{REFI}$ interval between surrounding auto refresh commands is allowed [12].

F. Refresh Timing

The SDR SDRAM describes refresh requirement in terms of retention time t_{REF} for each cell (which is the same as each row). The specifications of subsequent generations use t_{REFI} parameter, which is longest period between two consecutive refreshes operations. For simple memories—which perform a refresh of single row during one operation—the relation between the two parameters is $t_{REFI} = t_{REF}/N_{rows}$. But, bigger devices of later generations refresh multiple rows during the single auto refresh operation; hence the t_{REFI} parameter is given in the specifications. While the time needed by the auto refresh operation is given in t_{RFC} parameter. It is usually slightly greater than row cycle time (t_{RC}) for very small devices, but can be several times greater for the larger devices.

At a higher temperatures the leakage current is larger and the retention time is smaller. The specifications usually requires doubling the refresh rate if the temperature is higher than $85°C$. This slightly reduces the memory throughput, but does not complicate the controller design.

[2]Also known as CBR (CAS Before RAS) from the times of non-synchronous DRAM interface

TABLE I
RELEVANT SDRAM(SDR) TIMING PARAMETERS

		7 ns clk (cycles)	8 ns clk (cycles)	10 ns clk (cycles)	Min (ns)	Max (ns)
	Clock Frequency (MHz)	143	125	100		
t_{CAC}	CAS Latency	3	3	2		
t_{RRD}	Row to Row Delay	2	2	2	14	
t_{RCD}	Row to Column Delay	3	3	2	20	
t_{RAS}	Row Access Strobe	7	6	5	45	120K
t_{RC}	Row Cycle	10	9	7	67.5	
t_{RP}	Row Precharge	3	3	2	20	
t_{CCD}	Column Command Delay Time	1	1	1		
t_{DPL}	Input Data to Precharge	2	2	2	14	
t_{DAL}	Input Data to Activate	5	5(4)	4	35	
t_{RBD}	Burst Stop to High Impedance	t_{CAC}	t_{CAC}	t_{CAC}		
t_{WBD}	Burst Stop to Input in Invalid	0	0	0		
t_{PQL}	Last Output to Auto-Precharge Start	1-t_{CAC}	1-t_{CAC}	1-t_{CAC}		
t_{QMD}	DQM to Output	2	2	2		
t_{DMD}	DQM to Input	0	0	0		
t_{MRD}	Mode Register Program Time	3(2)	2	2	15	
t_{REF}	Refresh Cycle (8192 rows)					64M

TABLE II
TIMING REQUIREMENTS FOR VALID SIGNALING IN NS

Symbol	Parameter	Min	Max
t_{AC2}	Access Time from CLK (CL=2)		6.5
t_{AC3}	Access Time from CLK (CL=3)		5.4
t_{OH2}	Output Data Hold Time (CL=2)	2.7	
t_{OH3}	Output Data Hold Time (CL=3)	3	
t_{HZ}	CLK to High Impedance Time	2.7	5.4
t_{LZ}	CLK to Low Impedance Time	0	0
t_{SU}	Input Setup Time	2	
t_H	Input Hold Time	1	

IV. A TIME-PREDICTABLE MEMORY CONTROLLER

This section provides information on the design, implementation, integration, and testing of the SDR (Single Data Rate) SDRAM controller. Section V presents the evaluation of the controller.

A. Analysis

The DE2-70 FPGA board, used for the project, contains two IS42S16160B-7TLI SDRAM chips [10]. They are organized as 16-bit words in 4 banks of 8192 rows by 512 columns. The chips are speed grade 7, so according to the specification they can be clocked up to 143 MHz (with CAS latency (CL) of 3) or 100 MHz (with CL=2). Table I lists the SDRAM parameters for the chip, while Table II list the parameters relevant for signal integrity.

1) Separation Between Transactions: The minimum number of cycles needed between two operations are summarized in Table III. They depend on the type of the operation as well as the banks the transactions use.

For random addresses the worst-case is when consecutive accesses happen to different rows of the same bank (top part of the table). Because we are interested in optimizing the worst-case performance, we use the closed page policy, as it assures the smaller worst case latency. The read transaction requires: row activation, CAS latency, burst transfer cycles, and precharge. In this case the precharge can be overlapped with last few data transfers cycles; the negative t_{PQL} term in the equation accounts for this. The burst transfer cycles are $BL-1$, as t_{CAC} already contains the first cycle of the data transfer (see Figure 1). The $max()$ is used to satisfy the t_{RAS} for smaller BL, and the whole sum must always be at least t_{RC}. The write transaction requires: precharge, activate, burst transfer, and write recovery cycles. $BL-1$ is used, because the transfer starts at the cycle of the write command (see Figure 2). Again, the whole sum must be at least t_{RC}.

The length of the burst and two timing parameters affect the separation of operations on different banks: the separation between activates (t_{RRD}) and the read or write commands (t_{CCD}). For SDR the smallest meaningful burst length for interleaving is 2, because each operation requires at least two command bus cycles (activate and read or write). Because t_{CCD} and t_{RRD} are usually not larger than 2 clock cycles, the separation becomes BL. This means that two consecutive memory operations of the same kind (read or write), can result in uninterrupted transfer on the data bus.

What remains to be considered, are the separation between operations of different directions. The read after write is constrained by t_{RTW}, but for SDR memories this is one clock cycle, which means that commands can be issued in consecutive cycles. Because the write data transfer starts at the same cycle as the write command, the minimum separation between the commands is BL as in the previous case. However, this creates t_{CAC} idle cycles on the data bus. The read command is issued in the next cycle after the last data is written, but the read data comes a few cycles later. The write after read needs an extra $t_{CAC}+1$ cycles between commands. The t_{CAC} is needed to let the read data to finish, and 1 extra idle cycle that is needed to allow the tristate buffers of the SDRAM to become high impedance before the controller starts driving the data onto the bus.

The command separation limits the maximum bandwidth available for random accesses. Table IV shows lengths of

TABLE III

MINIMAL SEPARATION BETWEEN SDRAM(SDR) TRANSACTIONS IN CLOCK CYCLES. THE SUBSCRIPT$_{n-1}$ DENOTES PREVIOUS OPERATION, AND B_{n-1} THE BANK IT USED. *BL* DENOTES BURST LENGTH, AND THE TIMING PARAMETERS ARE ROUNDED TO FULL CYCLES.

	$Read_{n-1}(b_{n-1})$	$Write_{n-1}(b_{n-1})$
$Read_n(b_n=b_{n-1})$	$max(t_{RC},$	$max(t_{RC},$
$Write_n(b_n=b_{n-1})$	$t_{RCD} + t_{CAC} + (BL-1) + t_{PQL} + t_{RP})$	$t_{RCD} + (BL-1) + t_{DPL} + t_{RP})$
$Read_n(b_n!=b_{n-1})$	$max(t_{RRD}, t_{CCD}, BL)$	$max(t_{RRD}, t_{CCD}, BL)$
$Write_n(b_n!=b_{n-1})$	$max(t_{RRD}, t_{CCD}, BL + t_{CAC} + 1)$	$max(t_{RRD}, t_{CCD}, BL)$

TABLE IV

THE MAXIMUM GUARANTEED PERCENTAGE OF DATA TRANSFER CYCLES FOR DIFFERENT BURST LENGTHS. THE TIMING PARAMETERS ARE FOR 100 MHZ OPERATION OF THE SDR MEMORY ON DE2-70 BOARD.

BL	$Cycles_{Read}$	$Cycles_{Write}$	Bw_{Read}	Bw_{Write}
1	7	7	14.28%	14.28%
2	7	7	28.57%	28.57%
4	8	9	50%	44.44%
8	12	13	66.66%	61.53%
2*2	8	8	50%	50%
2*4	8+3	8+3	72.72%	72.72%
pair of 2*4	8+3+8	8+3+8	84.21%	84.21%

read and write bursts in cycles and the resulting bandwidth in percent of the theoretical maximum bandwidth. The top part of the table lists the numbers for the simple transactions calculated according the Table III formulas. The bottom of the table contains 3 examples of transactions interleaved over two banks. The efficiency is increased, because precharge and activation of one bank is overlapped with transfers by the other bank. The 8 word transfers interleaved over 2 banks (2*4 bursts) allow to fully utilize the data bus, but 3 cycles gap is needed if the transfer direction is changed. The last row in the table accounts for the fact that at most one change in direction is needed for two transfers. The estimates in the table do not account for cycles wasted for refresh and will be lower. The efficiency will also be lower if higher frequency operation is used.

2) Performing Refresh: The SDRAM refresh operation interferes with regular read/write operations. Even though the refresh is required to be invoked relatively infrequently, its asynchronous nature can lead to memory access latency overestimation. There are three options to handle refresh. (1) Refresh operations can be grouped together and analyzed at schedulability analysis level [6]. (2) They can be invoked individually at a known time, for example in a dedicated TDM slot of a CMP system. (3) The refresh is a higher priority periodic operation invoked by the controller and increases the WCRT of each memory accesses.

The first option is only effective for the SDR generation of SDRAMs. Newer generations limits the grouping to only 8 refresh operations. The second option is constrained for larger density devices, where refresh requires much more time than the regular operations. The third option results in a conservative WCET if the refresh interference is pessimistically incorporated into each request. However, tighter WCET bounds are possible by calculating the refresh interference for the whole execution of the task and not for each memory request [5]. The architecture needs to be free from timing anomalies for the approach to be valid.

B. Design

To simplify the first design we have decided to the build controller using simple, non-interleaved transactions. The simple transaction controller issues the same sequence of commands for different configurations. This way it can be made configurable, where the target frequency and burst length are specified as generic parameters and all the wait cycles are calculated automatically by the synthesis software.

The controller can later be extended to use bank interleaving. This would potentially require different state machines for different configurations. However, the command patterns can be very regular resulting in a simple design.

After reset, the controller performs the initialization sequence. The memory access requests are translated to predefined sequence of actions. The single Mealy type state machine is used to control the sequencing. The Mealy type allows saving one clock cycle of latency on the requester's interface. This also works well with the SDRAM interface signals that have to be registered in IO cells to improve the timing.

We decided to use a non-proprietary interface standard to allow the reuse of the controller. The Open Core Protocol (OCP) interface is chosen. The out of band signals defined in the OCP protocol are used for manual triggering of refresh (if the option of automatic refresh is disabled).

C. Implementation

The implementation has following features:

- It is a simple RTL state-machine in one entity. The result is comprehensible and easily maintainable code, which is synthesized into a reasonable implementation by a standard vendor tool chain (see Section V-D)
- The code is meant to be portable. Therefore, we avoided vendor specific components or design patterns. The only vendor specific component is the PLL/DCM, which is necessary for higher clock frequencies, but it is instantiated externally to the controller. Also the use of IO-Block registers has to be specified in a vendor specific way. For Altera the signal attributes are used, but a .qsf file can be used as well. For Xilinx, this can be specified on a per signal basis in the constraint file.
- All the possible parameters are configurable through generics: (1) the signal widths for the requester interface (address/data) and the SDRAM interface (chip-selects/banks/address); (2) the address mapping from a

linear address of the requester to rank, bank, row and column; (3) the burst size supported by the SDRAM specification; (4) the frequency, access latency, refresh period, and other timing parameters from the memory chip specification. The timing parameters are specified using the VHDL time constants and are translated into required number of clock cycles according the user specified clock period automatically.
- Registers in IO-Blocks allow better setup times and clock to output delay.
- Wait states and counters insure the correct timing between SDRAM bus commands. The binary counters are sufficient for the needed wait ranges.

A 3 ns skew is introduced between the SDRAM clock and the clock used for the controllers state-machine, to adjust the clock edge with the data for the same setup/hold slack for both read and write operations as described in [3].

D. Integration

This section describes the integration of the controller with two processors for RTS.

1) Integration with the Patmos Processor: At the time of this writing the Patmos processor [20] from the T-CREST platform was still under development. The support for caches was not yet finished and the processors pipeline used simplified accesses to local memories without stalling. The integration was performed through a simple, I/O controlled DMA (Direct Memory Access) like device. The device can be asked to perform external memory transfers with its local buffer. The single cycle access is provided to the buffer to prevent stalling the processor pipeline. Instead, the processor polls the device status to find out if the memory transfer has been completed.

2) Integration with the JOP Processor: JOP is a time-predictable processor for real-time Java applications [17]. The JOP processor accesses the memory and I/O devices through the SimpCon [16] interface. The interface is optimized for the processor's pipeline. The processor drives the output signals for one clock cycle and waits until the slave completes the transaction. The interface allows the slave to perform the early completion acknowledgements, by providing the master a hint on how many wait cycles are needed before the data is ready. A 2-bit `rdy_cnt` signal is used for this purpose, where the maximum value of 3 has a special meaning of unknown number of cycles. The slave is required to keep the data value and the acknowledgement after the transaction is complete.

A small VHDL entity was created to adapt the controller to the SimpCon interface. Non-optimized adaptation is used, without the early acknowledgement and burst length of 1. When the transaction is issued the adapter sets the SimpCon `rdy_cnt` signal to 3 (busy with unknown completion time). The command, address, and write data signals are also registered to keep them stable as required by the controller. When the controller acknowledges the data, the `rdy_cnt` register is reset to 0 (ready, i.e. zero wait cycles left); and the data value is registered.

The inefficiency of not using the SimpCon early acknowledgement can easily be solved, as the controller keeps track of when the data will be available. This information can be used for `rdy_cnt`. Enabling burst transfers needs extensions to SimpCon interface and possible modification of the controller. The pipelined transactions supported by SimpCon could be used. Additional signals are needed to denote when the transaction corresponding to the same burst are finished. The controller could then use this information to start the bank precharge. In addition the controller would need to be modified to support pipelined transactions.

E. Testing

We used two testing methods: a VHDL testbench for controller simulation and test programs for checking the controller operation in the FPGA system.

1) VHDL Testbench: The testbench (TB) for RTL level simulation of the controller tests its behavior in isolation. The TB was also useful for locating the source of flaws, because the complete observability of the controller's state is available during the simulation.

The TB performs writes and reads with different addressees and checks that the read data matches the one written to that location. The mismatched entries are reported. There are also flags controlling the verbosity level of the reporting of the transactions on both the controller and processor interfaces. The reporting allows running the TB in batch mode, for simple check of correctness without the need to examine the signal waveforms.

The TB does not try to test all the configurations supported by the controller, because there are many. Instead, one configuration is tested, which is selected by specifying the configuration constants. The refresh period was configured to a small value to check the refresh logic interference with the regular transactions. The controller's conformance with the SDRAM timing constraint is verified with the SDRAM simulation model.

2) Patmos Test Programs: The synthesized version of the controller was tested on the FPGA configured with a system composed of a Patmos processor, the memory controller accessible through an I/O controlled DMA-like device (see Section IV-D1), and serial port for communication with the PC over an RS-323 cable. The test programs were written in C. At the time of the writing only C programs with limited features could be executed successfully by the available infrastructure. Therefore, two simple test programs were written.

The first program uses only a tiny part of the memory. The memory mapping of the I/O device is checked. Next some strings are written to the memory, read back for comparison, and are output to the serial port for examination. The second program tests the whole addressable memory range, but does not give hints on what went wrong. Distinct values are written into each memory location. Before the read-back is performed, the program waits for any input from the serial terminal. This is used to check if the SDRAM is refreshed to preserve the

correct values. The program reports the error when it occurs as well as the number of all errors discovered at the end.

3) JOP Hello World Test: The non-exhaustive test was used to check the JOP integration (Section IV-D2). The JOP system with the SDRAM as the only external memory was configured on FPGA. Than the "Hello World" program was transmitted to the FPGA over the RS-323 cable. The bootloader received the program and wrote it into the external memory and executed it from there.

V. Controller Evaluation

In this section we present the synthesis results of the controller. We compare our controller with three other SDR SDRAM controllers.

A. Altera SDR SDRAM Reference Design

The Altera reference design for an SDRAM controller is described in [4]. The controller is coded structurally without an explicit state machine. Instead the state is distributed into a number of internal control signals and counters, which control the multiplexers driving SDRAM signals. The controller is pipelined, introducing 4 cycles of additional data latency. The pipeline does not allow multiple outstanding commands to different banks, i.e., the new command is accepted after the previous one is completed (except for precharge used to interrupt the ongoing full-page read or write burst).

The controller provides a low-level interface to the SDRAM. The user must initialize the SDRAM and must keep track of data latency cycles (i.e., the controller does not acknowledge the valid data). The read/write requests to the controller are translated into pairs of activate and read/write (with auto-precharge) SDRAM commands. The requests corresponding to precharge, refresh, and mode register set commands are also available and are used to control the SDRAM initialization sequence. Two more requests configure the runtime parameters of the controller.

B. Xilinx SDRAM Reference Design

The design is described in [22]. The design is almost 14 years old, and one can see that the synthesis tools had different capabilities and a different hardware description style had to be used. The controller is coded structurally at a very low-level. The finite state machine (FSM) uses manually specified one-hot encoding for the state register. The Xilinx SRL16 primitives (lookup table used as shift register) are manually instantiated. Each counter is described in a separate entity and the design consists of 9 entities in total. The design provides the same functionality as the Altera design, albeit through a slightly different interface. The controller is run-time configurable and the user must initialize both the controller and the SDRAM. The requests are handled as single burst transaction with auto-precharge. The user must count the cycles to know when the valid data should be sampled after read. But the write data is accepted right away and is delayed internally.

TABLE V
Synthesis results for the evaluated SDR controllers. The columns show: clock frequency in MHz, the overall number of Logic-Cells/Slices, the number of Look-Up-Tables and the number of Flip-Flops

Design	F_{max} (MHz)	LC	LUT	FF
Altera	392.77	309	107	284
JOP	207.30	592	457	355
$JOP_{Optimized}$	249.38	308	174	238
Our	221.39	194	126	129
$Our_{SimpCon}$	222.42	272	119	211
$Our_{Optimized}$	349.41	200	127	131
Xilinx Spartan 3	F_{max}	Slices	LUT	FF
$Xilinx_{S3}$	116.20	229	165	293
Our_{S3}	117.79	114	147	130

The controller uses a double frequency clock for the SDRAM interface, and communicates the data with the user on both edges of the slower clock. The controller seams to be designed for use in an external chip, because a single 32-bit inout signal is used for address, datain, dataout and part of the command encoding. The controller introduces 8 SDRAM cycles (4 system cycles) of additional data latency during read.

C. JOP SDRAM Controller

The JOP controller [9] was made to enable the access to the SDRAM from JOP [18] on the Altera DE2 board. The timing parameters are hardcoded. Differently to the two previously presented designs, the controller performs SDRAM initialization automatically. The controller provides a 32-bit SimpCon [16] interface, but uses a 16-bit wide SDRAM chip. Therefore, some extra buffers and logic is used for this purpose. Two FSM are used, one handles the SDRAM command sequence, while the other interacts with the SimpCon and assembles/splits the two half-words. The SDRAM address multiplexing is performed in a separate process.

D. SDRAM Controllers Synthesis Results

The synthesis results for three designs mentioned in previous sections and our design are presented in Table V. The top part shows the numbers for the Altera Cyclone II target (EP2C70F896C6). The bottom part list numbers for a Xilinx Spartan 3 (xc3s5000-5fg900), because Xilinx reference design is not portable. The table has two entries for the JOP design. The numbers for the original design (JOP) were suspiciously large, and we were able to fix the problem by constraining the ranges of the counters ($JOP_{Optimized}$). The original design used full range integer types, resulting in the inferring of 32-bit counters. For the purpose of the JOP comparison we have also included the numbers ($Our_{SimpCon}$) of our design adapted for the use in JOP system (Section IV-D2). Finally the line $Our_{Optimized}$ represents the design after splitting the increment and comparison logic of the refresh counter into separate cycles (by registering the done flag). The initialization counter was also separated and made free running. The optimization is not necessary for our application, because the unoptimized device can run above the required speed. Nevertheless, the

high frequency of Altera design, stimulated us to see how big the speed gain of the optimization will be.

Comparing the hardware resources, we can see that our design is the smallest design. The comparison shows that the performance of our simple behavioral controller description is reasonable, and also that some speed gains are possible by simple optimization of critical path (see $\text{Our}_{\text{Optimized}}$ vs. Our).

The Altera design is clearly optimized for speed, so the high frequency is not surprising. The design is pipelined and the control logic is distributed and always depends on just a few bits. The critical path is on some wide multiplexer used to initialize a delay counter according the configuration register.

The higher FF count of Altera design results from pipelining, especially of the 32-bit wide data signal. All the input/outputs are also additionally latched, whereas our design expects the requester to hold the address and data stable. The Xilinx design uses many FF to register the input and additionally to store it in a few places, because the address/configuration/data all use a single bus.

Even though the maximum clock frequency of the Xilinx and our design are almost the same, the Xilinx controller supports higher SDRAM frequencies, because the SDRAM interface is operated with double frequency. The same trick could probably be applied to speed up our design if needed. Thought the frequency might decrease because of a possible introduction of critical path at the shorter period paths.

VI. Conclusion

Real-time systems need time-predictable components to support static WCET analysis. In this paper we presented an SDRAM controller that provides constant and well-known access time to the SDRAM memory. The controller is placed into open source and has been tested with two real-time platforms: the Java processor JOP and the time-predictable CMP platform T-CREST.

As future work we consider to explore the possibilities of performing memory access analysis at schedulability level. The precision and feasibility of this approach will depend on the modeling of the task's memory demand.

Acknowledgment

This work was partially funded under the European Union's 7th Framework Programme under grant agreement no. 288008: Time-predictable Multi-Core Architecture for Embedded Systems (T-CREST).

Source Access

We provide the VHDL code of the controller in open-source within the T-CREST repository at: https://github.com/t-crest/sdram

References

[1] Benny Akesson. *Predictable and Composable System-on-Chip Memory Controllers*. PhD thesis, Eindhoven University of Technology, February 2010.

[2] Benny Akesson, Kees Goossens, and Markus Ringhofer. Predator: a predictable SDRAM memory controller. In Soonhoi Ha, Kiyoung Choi, Nikil D. Dutt, and Jürgen Teich, editors, *Proceedings of the 5th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2007, Salzburg, Austria, September 30 - October 3, 2007*, pages 251–256. ACM, 2007.

[3] Altera. *SDRAM Controller Core, Quartus II Handbook Version 9.1 Volume 5: Embedded Peripherals*, v9.1 edition, November 2009.

[4] Altera Corporation. *SDR SDRAM Controller White Paper*, ver. 1.1 edition, August 2002.

[5] Pavel Atanassov and Peter Puschner. Impact of dram refresh on the execution time of real-time tasks. In *Proc. IEEE International Workshop on Application of Reliable Computing and Communication*, pages 29–34, Dec. 2001.

[6] Balasubramanya Bhat and Frank Mueller. Making DRAM refresh predictable. *Real-Time Systems*, 47(5):430–453, 2011.

[7] Christoph Cullmann, Christian Ferdinand, Gernot Gebhard, Daniel Grund, Claire Maiza, Jan Reineke, Benoît Triquet, and Reinhard Wilhelm. Predictability considerations in the design of multi-core embedded systems. In *Proceedings of Embedded Real Time Software and Systems*, May 2010.

[8] Stephen A. Edwards and Edward A. Lee. The case for the precision timed (PRET) machine. In *DAC '07: Proceedings of the 44th annual conference on Design automation*, pages 264–265, New York, NY, USA, 2007. ACM.

[9] Julian Grahsl. JOP DRAM support for Altera DE2 board (source code), 2012.

[10] Integrated Silicon Solutions, Inc. *IS42S16160B – 16Meg×16 256-MBIT Synchronous DRAM*, March 2007.

[11] Bruce L. Jacob, Spencer W. Ng, and David T. Wang. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2008.

[12] JEDEC Solid State Technology Association. *Double Data Rate DDR SDRAM Standard, (JESD79F)*, February 2008.

[13] M. PAOLIERI, E.Q.U.I. NONES, and F.J. CAZORLA. Timing effects of ddr memory systems in hard real-time multicore architectures: Issues and solutions.

[14] Marco Paolieri, Eduardo Quiñones, Francisco J. Cazorla, and Mateo Valero. An analyzable memory controller for hard real-time CMPs. *Embedded Systems Letters*, 1(4):86–90, 2009.

[15] Jan Reineke, Isaac Liu, Hiren D. Patel, Sungjun Kim, and Edward A. Lee. PRET DRAM controller: bank privatization for predictability and temporal isolation. In Robert P. Dick and Jan Madsen, editors, *Proceedings of the 9th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2011, part of ESWeek '11 Seventh Embedded Systems Week, Taipei, Taiwan, 9-14 October, 2011*, pages 99–108. ACM, 2011.

[16] Martin Schoeberl. SimpCon - a simple and efficient SoC interconnect. In *Proceedings of the 15th Austrian Workhop on Microelectronics, Austrochip 2007*, Graz, Austria, October 2007.

[17] Martin Schoeberl. A Java processor architecture for embedded real-time systems. *Journal of Systems Architecture*, 54/1–2:265–286, 2008.

[18] Martin Schoeberl. *JOP Reference Handbook: Building Embedded Systems with a Java Processor*. Number ISBN 978-1438239699. CreateSpace, August 2009. Available at http://www.jopdesign.com/doc/handbook.pdf.

[19] Martin Schoeberl. Time-predictable computer architecture. *EURASIP Journal on Embedded Systems*, vol. 2009, Article ID 758480:17 pages, 2009.

[20] Martin Schoeberl, Pascal Schleuniger, Wolfgang Puffitsch, Florian Brandner, Christian W. Probst, Sven Karlsson, and Tommy Thorn. Towards a time-predictable dual-issue microprocessor: The Patmos approach. In *First Workshop on Bringing Theory to Practice: Predictability and Performance in Embedded Systems (PPES 2011)*, pages 11–20, Grenoble, France, March 2011.

[21] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quiñones, M. Gerdes, M. Paolieri, and J. Wolf. Merasa: Multi-core execution of hard real-time applications supporting analysability. *Micro, IEEE*, 30(5):66–75, 2010.

[22] Xilinx. *Synthesizable High Performance SDRAM Controller, Application Note XAPP134*, v3.1 edition, February 2000.