28.17 Multi Channel Sequencer (MCS)

28.17.1 Overview

The Multi Channel Sequencer (MCS) sub module is a generic data processing module that is connected to the ARU. One of its major applications is to calculate complex output sequences that may depend on the time base values of the TBU and are processed in combination with the ATOM sub module. Other applications can use the MCS sub module to perform extended data processing of input data resulting from the TIM sub module. Moreover, some applications may process data provided by the CPU within the MCS sub module, and the calculated results are sent to the outputs using the ATOM sub modules.

*** 'Generic Design Parameters' on page 348 *** summarizes all available generic design parameters of the MCS hardware structure.

Table 64 Generic Design Parameters

Design Parameter	Description
W	Word width of the data path
T	Number of available MCS channels
RDW	RAM data width of connected RAM
RAW	RAM address width used by the MCS for addressing memory
USR	Use second RAM port (0 - one RAM port available, 1 - two RAM ports available)
BAW	Bus Master Address Width
BDW	Bus Master Data Width
URIP	Use RAM input pipeline registers (0 - no register, 1 - use register)
UROP	Use RAM output pipeline registers (0 - no register, 1 - use register)
UDP	Use Decoder Pipeline register (0 - no register, 1 - use register)
UAP	Use ALU Pipeline register (0 - no register, 1 - use register)
NPS	Total number of pipeline stages (with NPS = 3 + URIP + UROP + UDP + UAP)

All MCS instances in the GTM use the values T=8, W=24, RDW=32, RAW=12, USR=1, BAW = 14, BDW = 32, URIP = 1, UROP = 1, UDP = 1, UAP = 1, and NPS = 7.

28.17.2 Architecture

Figure 106 gives an overview of the MCS architecture assuming that all pipeline registers are implemented.

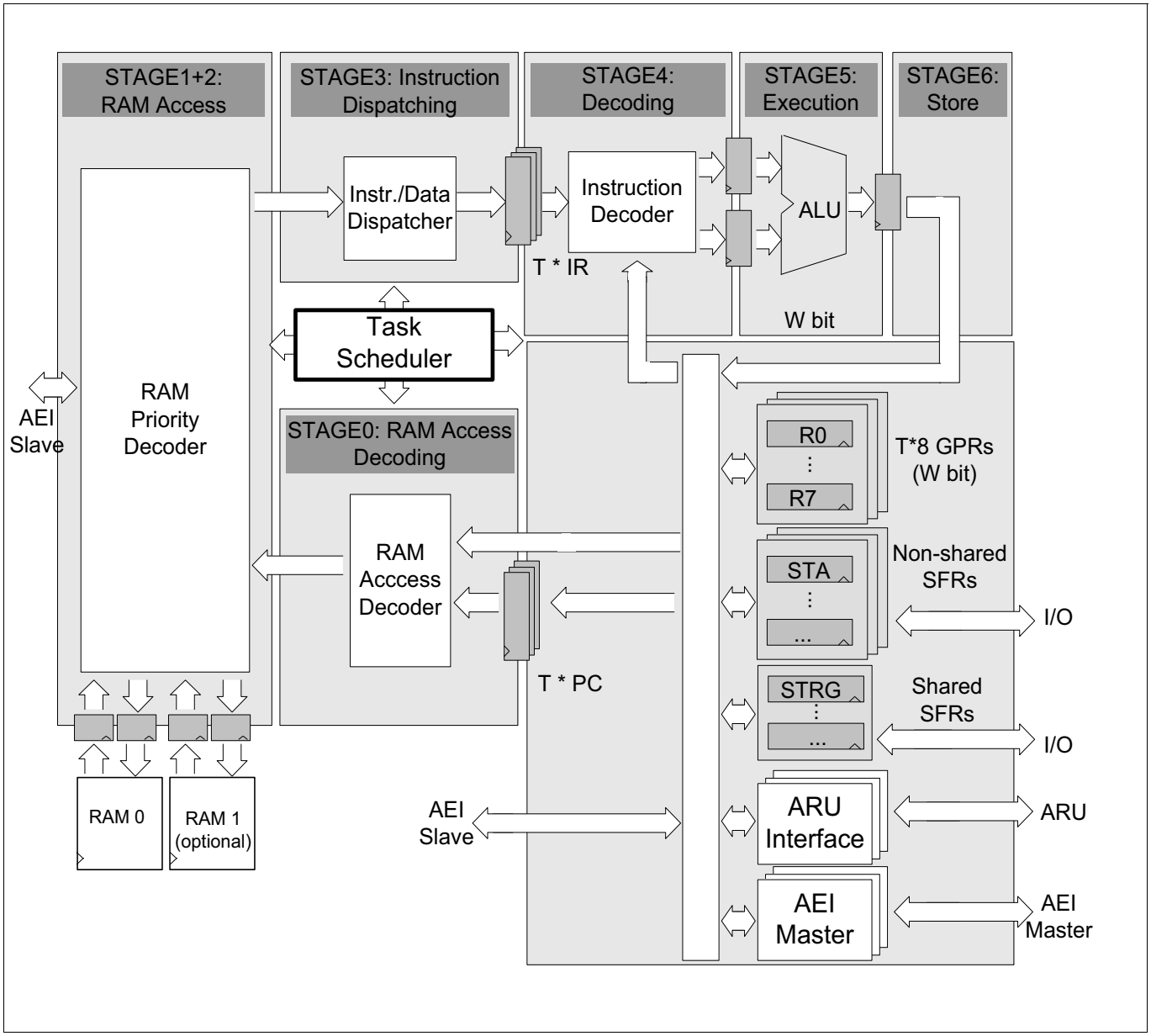


Figure 106 MCS Overview

The data path of the MCS is shared by T so called MCS channels, whereas each MCS channel executes a dedicated micro-program that is stored inside the RAM connected to the MCS module.

The connected RAM may contain arbitrary sized code and data sections that are accessible by all MCS channels and an externally connected master (e.g. a CPU) via AEI Slave interface. More details about the RAM can be found in **Section 28.17.4**

An MCS channel can also be considered as an individual task of a processor that is scheduled to the commonly used data path at a specific point in time. The execution of the different MCS channels on the different pipeline stages is controlled by a central hardware related task scheduler, which enables immediate task switches in parallel to the program execution. Details about the task scheduler and the available scheduling algorithms can be found in **Section 28.17.3**.

Typically, if data has to be exchanged between different MCS channels and/or the CPU, the connected RAM, which is accessible by all MCS channels and the CPU, can be used.

Besides the commonly used data path, each MCS channel has:

- a set of eight General Purpose Registers (GPRs), each W bit wide,
- a set of non-shared Special Function Registers (SFRs) that are only accessible within a dedicated MCS channels.
- a set shared SFRs that are accessible by all MCS channels,
- a channel specific instruction register (IR),
- a channel specific program counter register (PC),
- a dedicated ARU interface for communication with other ARU connected modules,
- and an AEI Bus Master Interface for controlling and configuration of other GTM sub modules.

Generally, the GPRs of an MCS channel x are only accessible by its corresponding MCS channel x. However, the MCS provides a configuration that allows an MCS channel x to access the GPRs of its successor MCS channel x+1. This feature can be used to enlarge the number of registers for a specific MCS channel x and/or to exchange data between neighboring channels.

For safety reasons, the register **MCS[i]_REG_PROT** can be used to define write protections for the neighboring registers of the individual MCS channels.

In order to enable synchronization between different MCS channels and/or the CPU, the MCS provides a common 24 bit wide trigger register that can be accessed as a shared SFR by all MCS channels located in the same module. Writing to **STRG** sets bits and writing to **CTRG** clears bits in the common trigger register. To enable triggering of MCS channels by CPU, the CPU can set bits in the common trigger register by writing to **MCS[i]_STRG** and clear bits by writing to **MCS[i]_CTRG**.

Considering the architecture in the figure above and assuming that all available pipeline stages are implemented (the generic parameters URIP, UROP, UDP, and UAP are set to 1), the main actions of the different pipeline stages are as follows:

- Pipeline stage 0 performs a setup of address, input data, and control signals for the next RAM access of a specific MCS channel.
- The actual RAM access of a specific MCS channel is executed in pipeline stage 1 and 2, assuming an external connection of a synchronous RAM with a latency of one clock cycle.
- Pipeline stage 3 performs pre-decoding and dispatching of instructions and data resulting from the RAM.
- In pipeline stage 4 the instructions are decoded and data from the registers are loaded.
- After that, in pipeline stage 5 the instruction is executed meaning that arithmetic operations are applied.
- Finally, in pipeline stage 6 the calculated results are stored in the registers.

If any of the pipeline registers is not implemented, the adjacent pipeline stages are merged and thus processed within the same clock cycle.

The RAM priority decoder arbitrates RAM accesses that are requested by the CPU via AEI and by the active MCS channel. If both, CPU and an MCS channel request a memory access to the same memory module the MCS channel is prioritized.

Since the internal registers of the MCS can be updated by different sources (MCS write access by various instructions, CPU write access via AEI slave, MCS write access by neighboring channel) a write conflict occurs if more than one source wants to write to the same register. In this case the result of the register is unpredictable. However, the software should setup its application in a way that such conflicts do not occur.

One exception is the common trigger register, which may be written by multiple sources (different MCS channels and CPU) in order to enable triggering of different MCS channels. Typically, the software should setup its application in a manner that different sources should not write the same bits in the trigger register.

28.17.3 Scheduling

The MCS provides a hardware related task scheduler, which globally controls the execution of the tasks in the different pipeline stages. The task scheduler implements four different scheduling modes, that can be selected by the **SCD_MODE** bit field in the **MCS[i]_CTRL_STAT** register. Depending on the selected scheduling mode, the task scheduler is selecting a dedicated MCS channel that will be executed in pipeline stage 0 in the next clock cycle. Additionally, MCS channels that are already present in the pipeline are shifted to its successor pipeline stage, with each clock cycle. This means, that the execution time of an MCS channel in a specific pipeline stage is always one clock cycle.

The MCS task scheduler may also schedule an empty cycle to pipeline stage 0, in order to grant a time slice to the CPU for accessing the connected RAM.

It should be noted, if the task scheduler assigns an MCS channel to pipeline stage 0, but this channel does not access the RAM, the CPU can access the corresponding RAM, even if the scheduler did not reserve an empty clock cycle.

In the following, the available scheduling modes are described.

28.17.3.1 Round Robin Scheduling

The Round Robin Scheduling Mode implements the simplest scheduling algorithm. This algorithm schedules a predefined set of MCS channels in the range [0; **SCD_CH**] in ascending order. After the last channel **SCD_CH** has been assigned to the pipeline, an empty cycle is scheduled in order to enable RAM access for the CPU. The parameter **SCD_CH** can be controlled by the register **MCS[i]_CTRL_STAT**. If the value of **SCD_CH** is greater than T-1, the scheduler assumes a value of T-1 for bit field **SCD_CH**.

Figure 107 shows a timing example of the Round Robin Scheduling with T=8 MCS channels (marked as C_0 to C_7) that are scheduled together with a CPU access to a pipeline with and NPS=7 stages. It is assumed that bit field **SCD_CH** is set to 7.

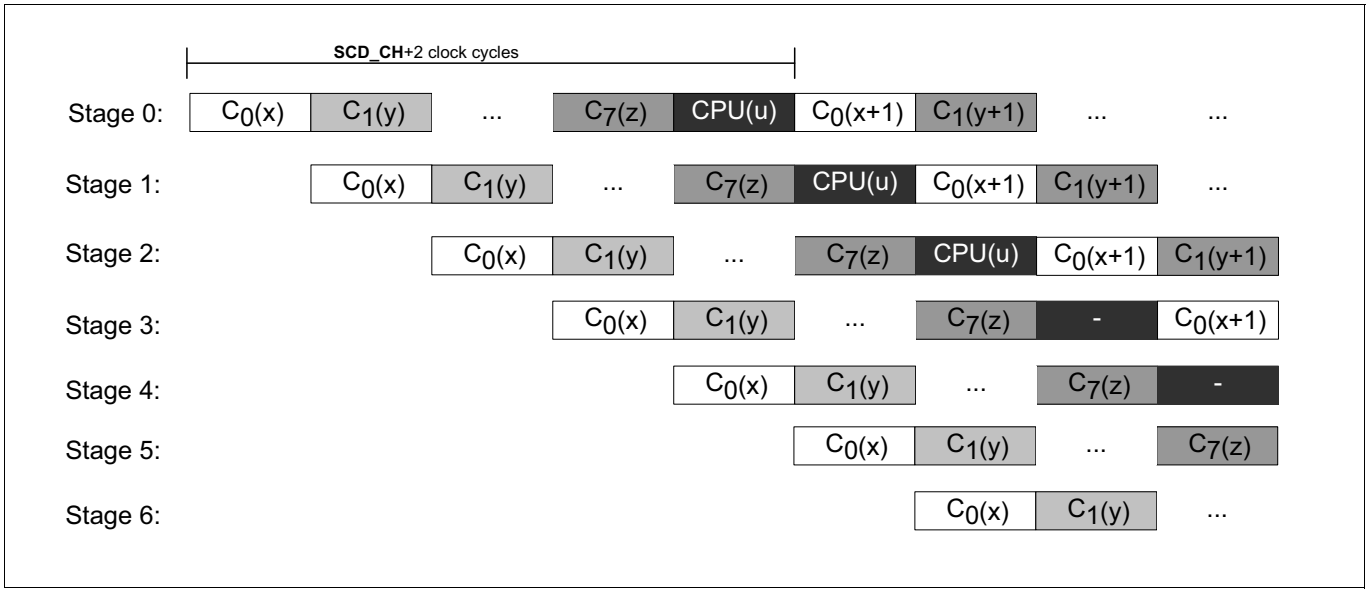


Figure 107 Timing of Round Robin Scheduling

The identifier $C_i(x)$ denotes that MCS channel i is currently executing the instruction or data located in the memory at position x in the corresponding pipeline stage. The figure shows, which MCS channel is activated in specific pipeline stage at a specific point in time.

Moreover, the figure shows that the Round Robin scheduling is always repeated after **SCD_CH**+2 clock cycles, which means that the time duration of an instruction cycle is **SCD_CH**+2 clock cycles. However, if the value

SCD_CH + 2 is less than NPS, the duration of an instruction cycle is limited by the depth of the pipeline to NPS clock cycles. Thus the effective execution time of a single cycle instruction is always MIN(**SCD_CH**+2, NPS) clock cycles, ignoring the latency of the pipeline.

If NPS is greater than T+1, NPS-T-1 additional empty cycles are inserted at the end of a round trip cycle. In this case the round trip time for the scheduler is determined by NPS, and thus the time duration for an instruction cycle is always NPS clock cycles.

The Round Robin scheduling algorithm has the characteristic that it fairly distributes all time slices to all MCS channels and the CPU. This means, that the program execution time of a specific task is independent from the activity of any neighboring task or the CPU RAM access, and thus a correct estimation of the actual program execution time is very easy. However, the round-robin scheduling may waste clock cycles by scheduling MCS channels that are not ready to execute an instruction (e.g. MCS channel is disabled by CPU). The following scheduling modes overcome this issue.

28.17.3.2 Accelerated Scheduling

In order to improve the computational performance, the accelerated scheduling mode provides two key features. Firstly, the scheduler only selects MCS channels that are not suspended and thus can actually execute an instruction. Secondly, the scheduler applies instruction prefetching to minimize empty cycles in the pipeline. An MCS channel is entering suspended state due to one of the reasons:

- An MCS channel is executing a read or write request to an ARU connected sub module (instruction ARD, AWR, ARDI, AWRI, NARD, NARDI).
- An MCS channel is executing a read or write request at its bus master interface (instruction BRD, BWR, BRDI, BWRI).
- An MCS channel waits on a register match event (e.g. instruction WURM), in order to wait on a desired register value (e.g. trigger event from another MCS channel).
- An MCS channel is disabled.

In the case of instruction prefetching, the scheduler will assign an MCS channel C_p to pipeline stage 0, which is already present in another pipeline stage. This means, that the execution of the last instruction of C_p located in the memory MEM(PC/4) is not yet finished completely, whereas PC is the current value of the program counter of MCS channel C_p. Thus, the newly scheduled MCS channel C_p will prefetch a successor instruction MEM(PC/4+PFO) under the assumption that there will be no branch and no memory access in the program between the instructions MEM(PC/4) and MEM(PC/4+PFO). The prefetch offset value PFO is determined by counting the number of already scheduled MCS channels Cp in the pipeline. However, if the assumption fails, the pipeline will be flushed by replacing all MCS channel C_p of the pipeline with an empty cycle, as soon as the instruction decoder detects a branch or a memory access. All other MCS channels unequal to MCS channel C_p within are not affected by the flushing action. The flushing action is always synchronized to the last pipeline stage NPS-1.

Besides the flushing conditions mentioned above, there exist also other conditions that cause a flush of the pipeline for a specific MCS channel. In the following all possible flushing events are summarized:

- An MCS channel is enabled.
- An MCS channel is entering a suspended state.
- An MCS channel is taking a conditional or unconditional branch (instruction JMP, JBS, JBC, CALL, RET, JMPI, JBSI, JBCI, CALLI).
- An MCS channel accessing memory for data transfer (instruction MRD, MWR, MRDI, MWRI, MRDIO, MWRIO, MWRL, MWRIL, PUSH, POP).
- An MCS channel is executing a read or write request at its bus master interface (instruction BRD, BWR, BRDI, BWRI).

- An MCS channel is modifying the trigger register (write access to **CTRG** or **STRG**) and the same channel is reading back this register (read access to **CTRG** or **STRG**) while the delay between both accesses is less than UAP+UDP+1 clock cycles.

In general, each MCS channel can accept instruction prefetching. However, there are some cases in which an upcoming flushing of the pipeline can be easily detected by the MCS hardware due to evaluation of internal states. Therefore, it is defined that an MCS channel accepts instruction prefetching only under the following conditions:

- An MCS channel is currently not in the second cycle of a two-cycle control flow instruction (instruction CALL, RET).
- An MCS channel is currently not in the second cycle of a three-cycle memory access instruction (instruction MWRL, MWRIL).

The accelerated scheduling mode guarantees, that the time duration of an instruction cycle varies between 1 and T+1 cycles. Hence, a single cycle instructions has an effective execution time between 1 to T+1 clock cycles, depending on the number of suspended MCS channels and the actual instruction sequence. The worst case execution time occurs if all channels are active and the CPU also accesses the RAM. The best case occurs e.g. if only one MCS channel is enabled and the executed program sequence has only linear code without branches and memory access.

The algorithm of the accelerated scheduling mode first, evaluates the state of all available MCS channels as well as a CPU request to the RAMs and then it decides if a specific MCS channel or an empty cycle is assigned to pipeline stage 0 in the next clock cycle. It should be noted that the accelerated scheduling mode treats RAM access requests from the CPU in a similar manner as MCS channels, which means that empty cycles for RAM requests are only inserted into the pipeline if there is an active RAM request from the CPU or no other task can be scheduled.

In order to fairly trade all available MCS channels as well as CPU RAM requests and to guarantee a worst case execution time of T+1 clock cycles, an additional task prioritization scheme is applied used that dynamically prioritizes all MCS channels and a CPU memory access depending on the history of the scheduler's decisions. The algorithm of the accelerated scheduler mode is executed every clock cycle and it works in the following manner:

1. Try to find an MCS channel C_r with highest priority that is not suspended and not already scheduled to the pipeline stages 0 to NPS-2. If C_r is found assign C_r to pipeline stage 0 and finish scheduling for current clock cycle.
2. Otherwise, try to find an MCS channel C_p with highest priority that is not suspended and accepts instruction prefetching. If C_p is found assign C_p to pipeline stage 0 and finish scheduling for current clock cycle.
3. Otherwise, try to find an MCS channel C_s with highest priority that is suspended and accepts instruction prefetching. If C_s is found assign C_s to pipeline stage 0 and finish scheduling for current clock cycle.
4. Otherwise, assign an empty cycle to pipeline stage 0 and finish scheduling for current clock cycle.

The underlying task prioritization scheme tracks the history of the scheduled MCS channels in a list consisting of T+1 items. The list is initialized with all MCS channels followed by a reserved time slot for the CPU RAM access. The position of an MCS channel within this list implicitly defines the priority, while the back of this list holds the MCS channel with highest priority. Whenever the scheduling algorithm described above has found an MCS channel C_r or C_p to be scheduled in the next clock cycle, it removes this item from the list and put it to the front of the list. In order to fairly prioritize all MCS channels, the algorithm also removes the item at the back of the list to the second position in the list, after the inserted scheduled front item. Since the list always contains all possible MCS channels and with each clock cycles each nonscheduled item is moved at least one position towards the end of list, it is obvious that each MCS channel will have the highest priority not later than T+1 clock cycles.

Figure 108 shows a timing example of the accelerated scheduling with NPS=7 pipeline stages.

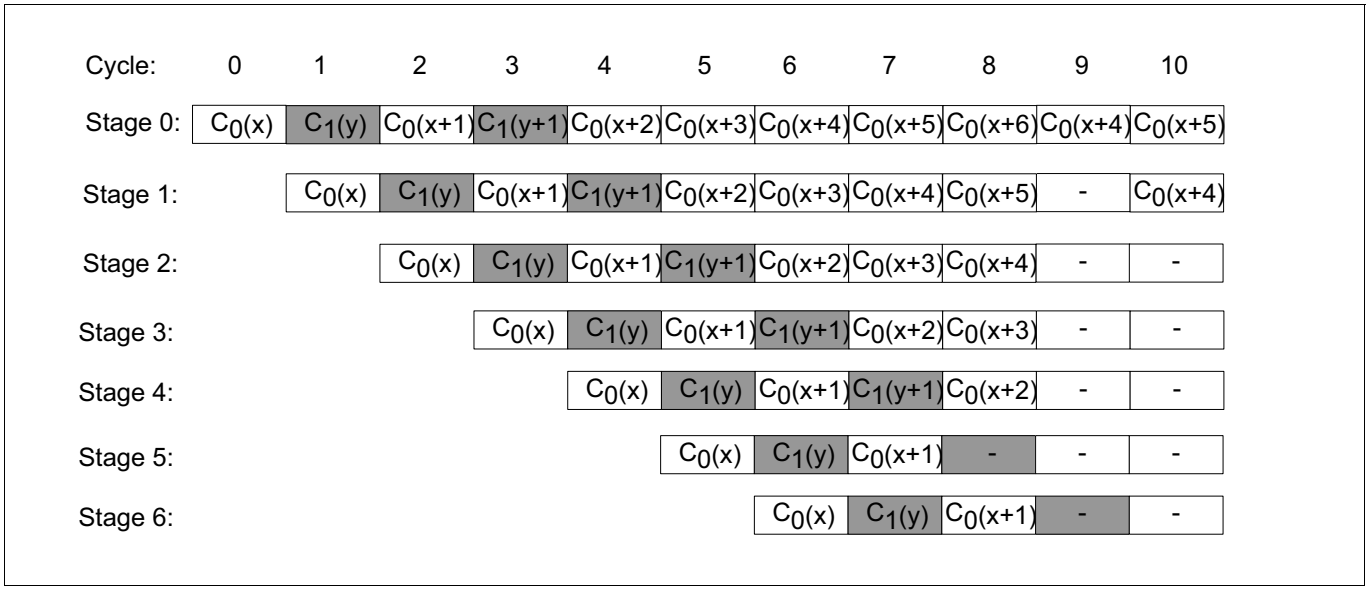


Figure 108 Timing of Accelerated Scheduling

The example assumes that initially MCS channels 0 and 1 are enabled and the program for each MCS channel is located in the RAM as shown in *** 'MCS Code example for Accelerated Scheduling' on page 354 ***.

MCS-Channel 0

Memory Location	Instruction
x+0	ADDL R0, 7
x+1	JBC STA, Z, 4*(x+4)
x+2	MOVL R2, 5

MCS-Channel 1

Memory Location	Instruction
y+0	WURM STRG, R1, 0
y+1	ADDL R2, 5
y+2	ADD R3, R2

Figure 109 MCS Code example for Accelerated Scheduling

Since both channels are ready to run, the scheduler fairly selects the channels in an alternating order, as it can be obtained in stage 0 at the clock cycles before cycle 5. Since MCS channel 1 is entering suspended state (to wait on a trigger bit) at cycle 7 in stage 6 with the instruction of memory location y, the scheduler will only select MCS channel 0 in the following by applying instruction prefetching. Moreover, entering the suspended state in channel 1 also flushes the remaining channels 1 out of the pipeline. But it should be noted, the scheduler applies instruction prefetching during the whole sequence, due to the fact that the number of enabled channels is always less than the available number of pipeline stages NPS.

The actual state of pipeline in cycle 9 and 10 depends on conditional branch instruction of memory location x + 3 (cycle 8 stage 6). If the branch is not taken, the linear code execution of MCS channel 0 is continued. However, if the branch to memory location x+4 is taken, as shown in the Figure, the scheduler will fetch the instruction C0(x+4) in cycle 9 at stage 0 and flush the stages 1 to NPS-1. Note, the flushing of the pipeline only concerns the prefetched instructions of the MCS channel that is currently executed in the last stage. If pipeline stage 1 of cycle 9 would belong to another channel than 0, only the stages greater than 2 would have been flushed.

28.17.3.3 Single Prioritization Scheduling

The Single Prioritization Scheduling mode is an extended variant of the Accelerated Scheduling mode, which additionally applies a task prioritization of a single MCS channel. In this mode, the bit field **SCD_CH** of register **MCS[i]_CTRL_STAT** is used to identify a dedicated MCS channel that is always preferred during scheduling. This

means, that the scheduler will assign preferred MCS channel **SCD_CH** to pipeline stage 0, as long as this channel is not suspended. If the preferred MCS channel is entering its suspended state, the scheduling algorithm switches to the accelerated scheduling as previously described in **Section 28.17.3.2**. Whenever the MCS channel **SCD_CH** is resuming from its suspended state, the scheduler switches back and assign the channel **SCD_CH** to pipeline stage 0 until the next suspension event occurs. If the bit field **SCD_CH** contains the value T or higher, the task scheduler will always prioritize CPU access to the RAM. This means, whenever the task scheduler detects that the CPU wants to access an MCS-RAM, the scheduler will assign an empty cycle into pipeline stage 0. If the CPU does not access the RAM any more, it switches back to the accelerated mode, as described previously in **Section 28.17.3.2**.

In consequence, the Single Prioritization Scheduling mode cannot guarantee a maximum time duration of an instruction cycle for the overall execution of all MCS channels, since it strongly depends on the activity of the prioritized MCS channel **SCD_CH**. However, the Single Prioritization Scheduling mode provides the fastest possible execution for MCS channel **SCD_CH**. Moreover, during the time spawn, in which the prioritized MCS channel **SCD_CH** is suspended, this mode guarantees a duration of 1 to T+1 clock cycles of an instruction cycle for all non-prioritized channels.

28.17.3.4 Multiple Prioritization Scheduling

The Multiple Prioritization Scheduling mode is an extended variant of the Accelerated Scheduling mode, which additionally applies a task prioritization for multiple MCS channels. In this mode, the bit field **SCD_CH** of register **MCS[i]_CTRL_STAT** is used to identify a set of dedicated MCS channels, which are always preferred during scheduling. The identifiers of the prioritized MCS channels are in the range [0; **SCD_CH**] and the non-prioritized channels are in the range [**SCD_CH**+1; T-1]. The individual priority for the set of prioritized MCS channels is applied in descending order, which means that MCS channel 0 has the highest priority, followed MCS channel 1, which has the second highest priority, and so on. The non-prioritized MCS channels do not have any priority. A value of T-1 or higher for the bit field **SCD_CH** means that all T MCS channels are prioritized MCS channels.

With each clock cycle, the Multiple Prioritization Scheduling mode will assign the non-suspended MCS channel with the highest priority from the set of prioritized MCS channels to pipeline stage 0, as long as there are non-suspended prioritized MCS channels available. If all prioritized MCS channels are suspended, the scheduling algorithm switches to the accelerated scheduling as previously described in **Section 28.17.3.2** and it schedules the non-prioritized channels. Whenever a prioritized MCS channel is resuming from its suspended state, the scheduler switches back and applies the described prioritization scheme until the next suspension event of occurs.

In consequence, the Multiple Prioritization Scheduling mode cannot guarantee a maximum time duration of an instruction cycle for the overall execution of all MCS channels, since it strongly depends on the activity of the prioritized MCS channels. However, the Multiple Prioritization Scheduling mode provides the fastest possible execution for prioritized MCS channels. Moreover, during the time spawn, in which all prioritized MCS channels are suspended, this mode guarantees a duration of 1 to T+1 clock cycles of an instruction cycle for all non-prioritized channels.

28.17.4 Memory Organization

The MCS module supports a memory layout of up to $2^{RAW+USR}$ memory locations each RDW bit wide leading to a maximum byte wise address range from 0 to $2^{RAW+USR+2}$-1.

If two RAM ports are used (USR = 1) the entire address space of the MCS is divided into two seamless memory pages. Further, if the GTM provides a memory configuration sub module (MCFG), memory page 0 begins from (byte wise) address 0 and ranges to address MP0-4 and memory page 1 ranges from MP0 to MP1-4, while MP0 and MP1 are configuration parameters provided by MCFG.

The RAM priority decoder of the MCS will always handle a RAM access from an MCS channel with a higher priority compared to a RAM access from AEI.

However, if a set of active MCS channels are only accessing one common RAM port, the MCS will grant any AEI accesses to the other RAM port in parallel to the related RAM accesses of the running MCS channels, which means that AEI may get the full bandwidth to a dedicated RAM.

Basically, the actual access time to the RAMs via AEI depends on the actual scheduling mode and the activity of tasks. In the modes Round Robin Scheduling and Accelerated Scheduling the scheduler guarantees a maximum write access time of T + 4 clock cycles and a maximum read access time of T + 6 clock cycles. In the scheduling modes Single Prioritization Scheduling and Multiple Prioritization Scheduling, the scheduler cannot guarantee a maximum access time for AEI RAM access.

Depending on the silicon vendor configuration, the connected RAM pages are initialized with zeros in the case of an MCS module reset.

If an ECC Error occurs while an MCS channel reads data from a memory module, the corresponding MCS channel is disabled and the ERR bit in register STA is raised.

If the GTM sub module CCM provides several so called address range protectors (ARPs), some code and data sections of the MCS RAM can be write protected. If an MCS channels x writes to such a protected memory region, the MCS channel x is halted, the **ERR** bit in register **STA** is set and the bit field **ERR_SRC_ID** of register **MCS[i]_CTRL_STAT** is updated.

28.17.5 AEI Bus Master Interface

The MCS module provides an AEI bus master interface, which enables to communicate with externally connected modules. The data width of this interface is BDW bit and the address width is BAW bit leading to a maximum byte wise address ranging from 0 to 2^{BAW+2}-1. The bus master interface is shared among all available MCS channels meaning that each MCS channel may initiate a read or write access on the bus but only one channel can be served at a specific point in time.

However, the AEI bus master interface guarantees, that a bus access is always completed within two instruction cycles and bus access of different MCS channels do not modify the latency of each other. The only exceptions are bus accesses to RAM modules (e.g. accessing memory location in a DPLL RAM or FIFO RAM). AEI bus master accesses to RAM modules cannot be completed within a single clock cycle and thus additional wait cycles have to be inserted into the bus protocol leading to the fact that the MCS channel that is accessing the RAM is entering a suspended state. Moreover, if an MCS channel is accessing a RAM module, the latency of a bus access in another MCS channel can also be modified even if the neighboring channel is accessing only a configuration register.

The AEI bus master interface of an MCS module is connected to AEI slave interface GTM-IP in order to control the sub modules of the GTM within MCS. However, it is not possible to access the entire GTM by a single MCS module. The n-th MCS instance can only access the GTM sub modules that are located within the n-th cluster of the GTM. Details about the available clusters of the GTM can be found inside the “GTM Architecture” Overview chapter.

Additionally, the address map for accessing GTM with the AEI bus master interface of an MCS differs from the address map for an externally connected CPU that is using the GTM's AEI slave interface. The address map for accessing GTM with the AEI bus master interface of an MCS can be found in device specific appendix.

Since the sub modules of the GTM can be accessed by the CPU and the AEI bus master of an MCS, the GTM-IP provides an additional arbitration scheme to manage parallel accesses from both master interfaces. If CPU and an MCS want to access a GTM sub module of the same cluster, the arbiter will grant the access to the MCS. However, if the CPU and all the MCS instances want to access GTM sub modules of different clusters, the accesses can be executed in parallel.

The AEI bus master interface can be controlled by the MCS instructions BRD, BRDI, BWR, and BWRI. These instructions are described in **Section 28.17.7**.

28.17.6 ADC Interface

The clusters of the GTM can optionally provide a dedicated interface for the connection of up to 32 external Analog-Digital-Converter (ADC) channels, which can be mapped arbitrarily to physical instances of single- or multi-channel ADCs.

An ADC Interface is directly mapped into the address map of the AEI bus master interface of the current cluster's MCS meaning that the available AEI bus master instructions (BRD and BRDI as described in **Section 28.17.7**) are used to control the connected ADCs.

Since the control of the connected ADCs is silicon vendor specific, the GTM specification does not provide a complete specification for controlling connected ADCs. However, to ensure software compatibility at least for the basic features of an ADC, the functionality described in the following are common to all silicon vendors.

28.17.6.1 Basic ADC Functions

The address map of the AEI bus master interface reserves two unique address items for each ADC channel. The address items can be referred by the labels **ADC_CH[y]_DATA** and **ADC_CH[y]_STA** for the channel y in the range from 0 to 31. The actual address can be found in the appendix.

The MCS can read from address **ADC_CH[y]_DATA** in order to get the conversion result of the ADC that is connected to ADC channel y. The conversion result is represented as a signed 24 bit value and it is stored in the register A (A $\in$ GREG) as referred by the corresponding MCS instruction BRD or BRDI. Additionally, each read access to **ADC_CH[y]_DATA** triggers the ADC that is connected to channel y. Any read access to **ADC_CH[y]_DATA** also provides 8 status bits that are stored in register **MHB**. The bit **MHB**[7] has always the mnemonic **ADC_ACK** and the bit **MHB**[6] has always the mnemonic **ADC_DATA_DATA**. If bit **ADC_ACK** is set the result of the data conversion (register A) and the corresponding status bits (bits **MHB**[6:0]) are validated. If **ADC_NEW_DATA** is set the current conversion result is new and has never been read by a previous bus read access. The meaning of the bits **MHB**[5:0] are vendor specific. Otherwise, if **ADC_ACK** is cleared the read data is invalid and the **MHB**[4:0] indicate the channel identifier (with **MHB**[4:0] <> y) that is currently processed by the ADC. A write access to **ADC_CH[y]_DATA** has no functionality and is always ignored.

The MCS can read from address **ADC_CH[y]_STA** to get additional 31-bit wide vendor specific status information of ADC channel y. The lower 24 bits of the status information is stored in register A (A $\in$ GREG) as referred by the corresponding MCS instruction BRD or BRDI. The upper 7 bit of the status information is stored in register **MHB**[6:0]. The bit **MHB**[7] has always the mnemonic **ADC_ACK**. If bit **ADC_ACK** is set the result of status information in register A and bits **MHB**[6:0] are validated. Otherwise, if **ADC_ACK** is cleared the status information is invalidated. A write access to **ADC_CH[y]_STA** has no functionality and is always ignored.

Any read or write access to a register **ADC_CH[y]_STA** or ADC_CH[y]_DATA updates the AEI status signal that is evaluated in the sub module CCM. The following status information is defined for the AEI status values:

- 00_B: no error occurred
- 01_B: optional information register not implemented (only register **ADC_CH[y]_STA**)
- 10_B: illegal ADC access (e.g. ADC not enabled)
- 11_B: unsupported address (ADC channel y not available)

Note: *Note: If the received status AEI is unequal to "00" ADC_NEW_DATA is always set and ADC_ACK is always cleared.*

28.17.7 Instruction Set

This section describes the entire instruction set of the MCS sub module. First, a brief overview over all available instructions is given and a detailed description of each instruction can be found in **Section 28.17.7.1** and the following sections.

In general, each instruction is RDW bit wide but the duration of each instruction varies between several instruction cycles. As already described in **Section 28.17.3**, the number of required clock cycles for an instruction cycle can be fixed or variable, depending on the selected scheduling mode. In the case of the Round Robin Scheduling, the duration is fixed with T+1 clock cycles, in the case of the Accelerated Scheduling the duration is variable in the range between 1 and T+1 clock cycles, and in all other Scheduling modes the duration is also variable and may even be more than T+1 clock cycles, depending on the application.

Before the available instructions are described, some commonly used terms, abbreviations and expressions are introduced:

OREG: The operation register set OREG = {R0, R1…R7} || {STA, ACB, CTRG, STRG, TBU_TS0, TBU_TS1, TBU_TS2, MHB} include all MCS accessible internal channel specific GPRs {R0, R1…R7} and the sub set {STA, ACB, CTRG, STRG, TBU_TS0, TBU_TS1, TDU_TS2, MHB} of SFRs.

XOREG: The extended operation register set XOREG = OREG || {RS0, RS1…RS7} || {GMI0, GMI1, DSTA, DSTAX} extends the operation registers set OREG by the GPRs of the succeeding MCS channel {RS0, RS1… RS7} and the SFRs {GMI0, GMI1, DSTA, DSTAX}.

WXREG: The extended wait instruction operation register set WXREG = OREG || {GMI0, GMI1, DSTA, DSTAX} extends the operation registers set OREG by the SFRs {GMI0, GMI1, DSTA, DSTAX}.

AREG: The ARU register set AREG = {R0, R1, R2…R7, ZERO} includes the all registers that can be written by incoming ARU transfers (ARD, ARDI, NARD, and NARDI instructions). These registers include all eight general purpose registers. The dummy register ZERO may be used to discard an incoming 24 bit ARU word.

GREG: The general purpose register set GREG = {R0, R1, R2…R7} includes the all channel specific GPRs without GPRs of neighboring channels.

BAREG: The base address register set BAREG = OREG || {RS0, RS1…RS7} extends the register set OREG by the GPRs of neighboring channels.

Note: If the extended operation register set XOREG is disabled (bit **EN_XOREG** of register **MCS[i]_CTRL_STAT** is cleared) the sets **XOREG**, **WXREG**, and **BAREG** only contains the operation register set OREG.

Note: In the following, the register sets **OREG**, **XOREG**, **GREG, WXREG**, **BAREG** and **AREG** are referred by the instructions. Typically, an operation announces W data bits. Whenever, a register of a register set implements less than W bits, it is assumed that these register bits only define the LSBs of an operation. The missing MSBs are always read and written as zeros.

WLIT: The set WLIT = {0,1…2^{W}-1} is a W bit wide literal value used for encoding immediate operands.

ALIT: The set ALIT = {0,1…$2^{RAW+USR}$-1} is a RAW + USR bit wide literal value used for encoding memory addresses.

AOLIT: The set AOLIT = {-$2^{RAW+USR-1}$…-1,0,1…$2^{RAW+USR-1}$-1} is a RAW + USR bit wide literal value used for encoding relative memory address offsets.

ARDLIT: The set ARDLIT = {0,1…2^{9}-1} is a 9 bit literal used for ARU read addresses.

AWRLIT: The set AWRLIT = {0,1…23} is used as ARU write indexes, selecting one of the 24 ARU write address.

BALIT: The set BALIT = {0,1…2^{BAW}-1} is a BAW bit wide literal used for encoding bus master addresses.

SFTLIT: The set SFTLIT = {0,1…W} is used as literal value for shift instructions.

BWSLIT: The set BWSLIT = {1…W} is used as literal value for multiplication instructions.

BITLIT: The set BITLIT = {0,1…15} is a 4 bit literal used for bit indexing.

XBITLIT: The set XBITLIT = {0,1…W-1} is a literal used for bit indexing of register bits.

MSKLIT: The set MSKLIT = {0,1…2^{15}-1} is a 16 bit literal used for bit-masking.

BIT SELECTION: The expression VAR[i] represents the i-th bit of a variable VAR.

BIT RANGE SELECTION: The expression VAR[m:n] represents the bit slice of variable VAR that is ranging from bit n to bit m.

MEMORY ADDRESSING: The expression MEM(X) represents the RDW bit wide value at location x (x ∈ ALIT) of the memory. The expression MEM(x)[m:n] represents the bit slice ranging from bit n to m of the RDW bit wide word at memory location x.

ARU ADDRESSING: In the case of ARU reading, the expression ARU(x) represents the 2*W+5 bit wide ARU word of ARU channel at read address x (x ∈ ARDLIT). In the case of ARU writing, the expression ARU(x) represents a 2*W+5 bit wide ARU word that is written to an ARU channel indexed by the index x (x ∈ AWRLIT). The index x selects a single ARU write channel from the pool of the MCS sub module's allocated ARU write channels. An MCS sub module has 24 dedicated ARU write channels, indexed by values 0 to 23. The expression ARU(x)[m:n] represents the bit slice ranging from bit n to m of the 2*W+5 bit wide ARU word.

BUS MASTER ADDRESSING: In the case of reading/writing from the bus master interface, the expression BUS(x) represents the BDW bit wide data word that is read/written at address x (x ∈ BALIT). The expression BUS(x)[m:n] represents the bit slice ranging from bit n to m of the BDW bit wide data word at the bus.

Figure 110 ff. summarize the entire instruction set of the MCS and **Table 113** ff. shows the encoding of the individual instructions.

Class	Mnemonic	Operation	instruction cycles	Synopsis
Data transfer	MOVL A, C	A ← C	1	Move Literal, A in OREG, C in WLIT
	MOV A, B	A ← B	1	Move, A in XOREG, B in XOREG
	MRD A, C	A ← MEM(C)[W-1:0]; MHB ← MEM(C)[RDW-1:W]	2[1)]	Memory Read, A in OREG, C in ALIT
	MWR A, C	MEM(C)[W-1:0] ← A; MEM(C)[RDW-1:W] ← MHB	2[1)]	Memory Write, A in OREG, C in ALIT
	MRDI A, B [, C]	A ← MEM(B[RAW+USR+1:2]+C)[W-1:0]; MHB ← MEM(B[RAW+USR+1:2]+C)[RDW-1:W]	2[1)]	Memory Read Indirect, A in OREG, B in OREG, C in AOLIT (default C=0)
	MWRI A, B [, C]	MEM(B[RAW+USR+1:2]+C)[W-1:0] ← A; MEM(B[RAW+USR+1:2]+C)[RDW-1:W] ← MHB	2[1)]	Memory Write Indirect, A in OREG, B in OREG, C in AOLIT (default C=0)
	MRDIO A, B	A ← MEM(B[RAW+USR+1:2] + R5[RAW+USR+1:2])[W-1:0]; MHB ← MEM(B[RAW+USR+1:2] + R5[RAW+USR+1:2])[RDW-1:W]	2[1)]	Memory Read Indirect with Offset, A in XOREG, B in BAREG
	MWRIO A, B	MEM(B[RAW+USR+1:2] + R5[RAW+USR+1:2])[W-1:0] ← A; MEM(B[RAW+USR+1:2] + R5[RAW+USR+1:2])[RDW-1:W] ← MHB	2[1)]	Memory Write Indirect with Offset, A in XOREG, B in BAREG
	POP A	A ← MEM(R7[RAW+USR+1:2]]); MHB ← MEM(R7[RAW+USR+1:2])[RDW-1:W] R7 ← R7 - 4	2[1)]	Pop from stack, A in OREG
	PUSH A	R7 ← R7 + 4 MEM(R7[RAW+USR+1:2])[W-1:0] ← A MEM(R7[RAW+USR+1:2])[RDW-1:W] ← MHB	2[1)]	Push to stack, A in OREG
	MWRL A, C	MEM(C)[W-1:0] ← A	3[2)]	Memory Write Literal, A in OREG, C in ALIT
	MWRIL A, B	MEM(B[RAW+USR+1:2])[W-1:0] ← A	3[2)]	Memory Write Indirect Literal, A in OREG, B in OREG
ARU Transfer	ARD A, B,C	A ← ARU(C)[W-1:0] B ← ARU(C)[2*W-1:W] ACB ← ARU(C)[5+2*W:2*W]	>= 1	Blocking ARU Read, A in AREG, B in AREG, C in ARDLIT
	AWR A, B, C	ARU(C)[W-1:0] ← A ARU(C)[2*W:W] ← B ARU(C)[5+2*W:2*W] ← ACB	>= 1	Blocking ARU Write, A in OREG, B in OREG, C in AWRLIT
	ARDI A, B	A ← ARU(R6[8:0])[W-1:0] B ← ARU(R6[8:0])[2*W-1:W] ACB ← ARU(R6[8:0])[5+2*W:2*W]	>= 1	Blocking ARU Read Indirect, A in AREG, B in AREG
	AWRI A, B	ARU(R6[4:0])[W-1:0] ← A ARU(R6[4:0])[2*W-1:W] ← B ARU(R6[4:0])[5+2*W:2*W] ← ACB	>= 1	Blocking ARU Write Indirect, A in OREG, B in OREG
	NARD A, B, C	A ← ARU(C[8:0])[W-1:0] B ← ARU(C[8:0])[2*W-1:W] ACB ← ARU(C[8:0])[5+2*W:2*W]	>= 1[7)]	Non-Blocking ARU Read, A in AREG, B in AREG
	NARDI A, B	A ← ARU(R6[8:0])[W-1:0] B ← ARU(R6[8:0])[2*W-1:W] ACB ← ARU(R6[8:0])[5+2*W:2*W]	>= 1[7)]	Non-Blocking ARU Read Indirect, A in AREG, B in AREG
Bus Master	BRD A, C	A ← BUS(C)[W-1:0] MHB ← BUS(C)[BDW-1:W]	>= 1[8)]	Bus Master Read, A in GREG, C in BALIT
	BWR A, C	BUS(C)[W-1:0] ← A BUS(C)[BDW-1:W] ← MHB	>= 1[8)]	Bus Master Write, A in GREG, B C in BALIT
	BRDI A, B	A ← BUS(B[BAW+1:2])[W-1:0] MHB ← BUS(B[BAW+1:2])[BDW-1:W]	>= 1[8)]	Bus Master Read Indirect, A in GREG, B in GREG
	BWRI A, B	BUS(B[BAW+1:2])[W-1:0] ← A BUS(B[BAW+1:2])[BDW-1:W] ← MHB	>= 1[8)]	Bus Master Write Indirect, A in GREG, B in GREG

Figure 110 Instruction Set Summary (part 1)

Class	Mnemonic	Operation	instruction cycles	Synopsis
Arith. / Logic	ADDL A, C	A ← A + C	1	Add Literal, A in OREG, C in WLIT
	ADD A, B	A ← A + B	1	Add, A in XOREG, B in XOREG
	ADDC A, B	A ← A + B + CY	1	Add with carry, A in XOREG, B in XOREG
	SUBL A, C	A ← A - C	1	Subtract Literal, A in OREG, C in WLIT
	SUB A, B	A ← A - B	1	Subtract, A in XOREG, B in XOREG
	SUBC A, B	A ← A - B - CY	1	Subtract with carry, A in XOREG, B in XOREG
	NEG A, B	A ← − B	1	Negate, A in XOREG, B in XOREG
	ANDL A, C	A ← A AND C	1	AND Litral, A in OREG, C in WLIT
	AND A, B	A ← A AND B	1	AND, A in XOREG, B in XOREG
	ORL A, C	A ← A OR C	1	OR Literal, A in OREG, C in WLIT
	OR A, B	A ← A OR B	1	OR, A in XOREG, B in XOREG
	XORL A, C	A ← A XOR C	1	XOR Literal, A in OREG, C in WLIT
	XOR A, B	A ← A XOR B	1	XOR, A in XOREG, B in XOREG
	SETB A, B	A[B[4:0]] ← 1	1	Set Bit, A in XOREG, B in XOREG
	CLRB A, B	A[B[4:0]] ← 0	1	Clear Bit, A in XOREG, B in XOREG
	XCHB A, B	A[B[4:0]] ↔ CY	1	Exchange Bit with CY, A in XOREG, B in XOREG
	SHR A, C	A ← A >> C	1	Shift Right, A in XOREG, C in SFTLIT
	SHL A, C	A ← A << C	1	Shift Left, A in XOREG, C in SFTLIT
	ASRU A, B	A ← A >> B	1	Shift Right, A in XOREG, B in XOREG
	ASRS A, B	A ← A >> B	1	Shift Right, A in XOREG, B in XOREG
	ASL A, B	A ← A << B	1	Shift Left, A in XOREG, B in XOREG
	MULU A, B[, C]	[[R4,] A] ← A[(C-1):0] * B[(C-1):0]	1	Multiply Unsigned, A in XOREG, B in XOREG, C in BWSLIT (default C=W)
	MULS A, B[, C]	[[R4,] A] ← A[(C-1):0] * B[(C-1):0]	1	Multiply Signed, A in XOREG, B in XOREG , C in BWSLIT (default C=W)
	DIVU A, B[, C]	R4 ← A[(C-1):0] - B[(C-1):0] * ⌊A[(C-1):0] / B[(C-1):0]⌋; A ← ⌊A[(C-1):0] / B[(C-1):0]⌋	C[9)]	Divide Unsigned, A in XOREG, B in XOREG, C in BWSLIT (default C=W)
	DIVS A, B[, C]	R4 ← A[(C-1):0] - B[(C-1):0] * ⌊A[(C-1):0] / B[(C-1):0]⌋; A ← ⌊A[(C-1):0] / B[(C-1):0]⌋	C+4[10)]	Divide Signed, A in XOREG, B in XOREG, C in BWSLIT (default C=W)
	MINU A, B	A ← MIN(A, B)	1	Minimum Unsigned, A in XOREG, B in XOREG
	MINS A, B	A ← MIN(A, B)	1	Minimum Signed, A in XOREG, B in XOREG
	MAXU A, B	A ← MAX(A, B)	1	Maximum Unsigned, A in XOREG, B in XOREG
	MAXS A, B	A ← MAX(A, B)	1	Maximum Signed, A in XOREG, B in XOREG
Test	ATUL A, C	A < C ⇔ CY is set A = C ⇔ Z is set	1	Arithmetic Test Unsigned Literal, A in OREG, C in WLIT
	ATU A, B	A < B ⇔ CY is set A = C ⇔ Z is set	1	Arithmetic Test Unsigned, A in XOREG, B in XOREG
	ATSL A, C	A < C ⇔ CY is set A = C ⇔ Z is set	1	Arithmetic Test Signed Literal, A in OREG, C in WLIT
	ATS A, B	A < B ⇔ CY is set A = C ⇔ Z is set	1	Arithmetic Test Signed, A in XOREG, B in XOREG
	BTL A, C	A AND C	1	Bit Test Literal, A in OREG, C in WLIT
	BT A, B	A AND B	1	Bit Test, A in XOREG, B in XOREG

Figure 111 Instruction Set Summary (part 2)

Class	Mnemonic	Operation	instruction cycles	Synopsis
Control Flow	JMP C	PC ← C << 2	1[3]	Uncondiational Jump, C in ALIT
	JBS A, B, C	PC ← C << 2 if A[B] is set	1[4]	Jump if Bit Set, A in OREG, B in BITLIT, C in ALIT
	JBC A, B, C	PC ← C << 2 if A[B] is clear	1[4]	Jump if Bit Cleared, A in OREG, B in BITLIT, C in ALIT
	CALL C	R7 ← R7 + 4 MEM(R7[RAW+USR+1:2])[RAW+USR+1:0] ← PC+ 4 PC ← C << 2	2[5]	Call Subroutine, C in ALIT
	RET	PC ← MEM(R7[RAW+USR+1:2])[RAW+USR+1:0] R7 ← R7 - 4	2[5]	Return from Subroutine
	JMPI	PC ← R6[RAW+USR+1:2] << 2	1[3]	Uncondiational Jump Indirect
	JBSI A, B	PC ← R6[RAW+USR+1:2] << 2 if A[B] is set	1[4]	Jump if Bit Set Indirect, A in XOREG, B in XBITLIT
	JBCI A, B	PC ← R6[RAW+USR+1:2] << 2 if A[B] is clear	1[4]	Jump if Bit Clear Indirect, A in OREG, B in XBITLIT
	CALLI	R7 ← R7 + 4 MEM(R7[RAW+USR+1:2])[RAW+USR+1:0] ← PC+ 4 PC ← R6[RAW+USR+1:2] << 2	2[5]	Call Subroutine Indirect
Others	WURM A, B, C	wait until A = (B AND ((0xFF << 16) + C))	>= 1[6]	Wait Until Register Match, A in OREG, B in OREG, C in MSKLIT
	WURMX A, B	wait until A = (B AND R6)	>= 1[6]	Wait Until Register Match, A in OREG, B in WXREG
	WURCX A, B	wait until A ≠ (B AND R6)	>= 1[6]	Wait Until Register Change, A in OREG, B in WXREG
	WUCE A, B	wait until cyclic event comparison matches	>= 1[6]	Wait Until Cyclic Event, A in OREG, B in OREG
	NOP		1	No Operation

Footnotes:

1) Not faster than 1+NPS clock cycles due to pipeline flushing.
2) Not faster than 1+2*NPS clock cycles due to pipeline flushing.
3) Not faster than NPS clock cycles due to pipeline flushing.
4) If the jump is executed, it is not faster than NPS clock cycles due to pipeline flushing.
5) Not faster than 2*NPS clock cycles due to pipeline flushing.
6) If the MCS is configured in Single Prioritization or Multiple Prioritization Scheduling Mode the worst case latency for reactivating a prioritized MCS-channel is 2+NPS clock cycles.
7) Always faster than one ARU round trip cycle.
8) Suspends current MCS-channel if addressed slave inserts at least one wait cycle otherwise 2 instruction cycles but not faster than 1+NPS clock cycles due to pipeline flushing.
9) Not faster than C+NPS-1 clock cycles due to pipeline flushing.
10) Not faster than C+3+NPS clock cycles due to pipeline flushing.

Figure 112 Instruction Set Summary (part 3)

Menemonic	Instruction Code
MOVL	0001aaaaccccccccccccccccccccccccc
MOV	1010aaaabbbb0000-a-b----------
MRD	1010aaaa----0001-ccccccccccccc--
MWR	1010aaaa----0010-ccccccccccccc--
MRDI	1010aaaabbbb0011-ccccccccccccc--
MWRI	1010aaaabbbb0100-ccccccccccccc--
POP	1010aaaa----0101----------------
PUSH	1010aaaa----0110----------------
MWRL	1010aaaa----0111-ccccccccccccc--
MWRIL	1010aaaabbbb1000----------------
BRD	1010-aaa----1001ccccccccccccc--
BWR	1010-aaa----1010ccccccccccccc--
BRDI	1010-aaa-bbb1011----------------
BWRI	1010-aaa-bbb1100----------------
MRDIO	1010aaaabbbb1101-a-b----------
MWRIO	1010aaaabbbb1110-a-b----------
XCHB	1010aaaabbbb1111-a-b----------
ARD	1011aaaabbbb0000-------ccccccccc
AWR	1011aaaabbbb0001----------ccccc
NARD	1011aaaabbbb0010-------ccccccccc
NARDI	1011aaaabbbb0011----------------
ARDI	1011aaaabbbb0100----------------
AWRI	1011aaaabbbb0101----------------
SETB	1011aaaabbbb0110-a-b----------
CLRB	1011aaaabbbb0111-a-b----------
ADDL	0010aaaaccccccccccccccccccccccccc
ADD	1100aaaabbbb0000-a-b----------
SUBL	0011aaaaccccccccccccccccccccccccc
SUB	1100aaaabbbb0001-a-b----------
NEG	1100aaaabbbb0010-a-b----------
ANDL	0100aaaaccccccccccccccccccccccccc
AND	1100aaaabbbb0011-a-b----------
ORL	0101aaaaccccccccccccccccccccccccc
OR	1100aaaabbbb0100-a-b----------
XORL	0110aaaaccccccccccccccccccccccccc
XOR	1100aaaabbbb0101-a-b----------
SHR	1100aaaa----0110-a---------ccccc
SHL	1100aaaa----0111-a---------ccccc

Figure 113 Instruction Codes (part 1)

Menemonic	Instruction Code
MULU	1100aaaabbbb1000-a-b-------ccccc
MULS	1100aaaabbbb1001-a-b-------ccccc
DIVU	1100aaaabbbb1010-a-b-------ccccc
DIVS	1100aaaabbbb1011-a-b-------ccccc
MINU	1100aaaabbbb1100-a-b----------
MINS	1100aaaabbbb1101-a-b----------
MAXU	1100aaaabbbb1110-a-b----------
MAXS	1100aaaabbbb1111-a-b----------
ASL	1101aaaabbbb0011-a-b----------
ASRU	1101aaaabbbb0100-a-b----------
ASRS	1101aaaabbbb0101-a-b----------
ADDC	1101aaaabbbb0110-a-b----------
SUBC	1101aaaabbbb0111-a-b----------
ATUL	0111aaaaccccccccccccccccccccccc
ATU	1101aaaabbbb0000-a-b----------
ATSL	1000aaaaccccccccccccccccccccccc
ATS	1101aaaabbbb0001-a-b----------
BTL	1001aaaaccccccccccccccccccccccc
BT	1101aaaabbbb0010-a-b----------
JMP	1110--------0000-ccccccccccccc--
JBS	1110aaaabbbb0001-ccccccccccccc--
JBC	1110aaaabbbb0010-ccccccccccccc--
CALL	1110--------0011-ccccccccccccc--
RET	1110--------0100----------------
JMPI	1110--------0101----------------
JBSI	1110aaaabbbb0110-a-b----------
JBCI	1110aaaabbbb0111-a-b----------
CALLI	1110--------1000----------------
WURM	1111aaaabbbb0000ccccccccccccccc
WURMX	1111aaaabbbb0001---b----------
WURCX	1111aaaabbbb0010---b----------
WUCE	1111aaaabbbb0011----------------
NOP	0000----------------------------

Figure 114 Instruction Codes (part 2)

The individual instructions are decoded by evaluating the bits '0' and '1' at its expected positions, as mentioned in the table above. If the instruction decoder detects an invalid combination of these bits, the corresponding MCS channel is disabled and the ERR bit in the register STA is set. Bit positions marked as '-' are not relevant for the instruction. The bit position 'a', 'b', and 'c' are reserved for binary encoding of the instruction arguments A, B, and C.

Moreover, each instruction can set the **ERR** bit of register **STA** and stop the program execution, if a register write protection of an associated MCS channel is activated by the register **MCS[i]_REG_PROT**. This behavior is not explicitly mentioned in the instruction descriptions below. If an error occurs due to a write access to a protected register, it is ensured that the protected register is not overwritten. However, it is not ensured that other operations (e.g. PC updates) of the bad instruction are executed.

28.17.7.1 MOVL Instruction

Syntax	Operation	Status	Duration
MOVL A, C	A ← C	Z	1 instruction cycle

Transfer literal value C (C ∈ WLIT) to register A (A ∈ OREG).

The zero bit Z of status register STA is set, if the transferred value is zero, otherwise the zero bit is cleared.

The program counter PC is incremented by the value 4.

28.17.7.2 MOV Instruction

Syntax	Operation	Status	Duration
MOV A, B	A ← B	Z	1 instruction cycle

Transfer register B (B ∈ XOREG) to register A (A ∈ XOREG).

The zero bit Z of status register STA is set, if the transferred value is zero, otherwise the zero bit is cleared.

The program counter PC is incremented by the value 4.

28.17.7.3 MRD Instruction

Syntax	Operation	Status	Duration
MRD A, C	A ← MEM(C)[W-1:0]; MHB ← MEM(C)[RDW-1:W]	Z	2 instruction cycles but not faster than 1+NPS clock cycles due to pipeline flushing.

Transfer the lower W bits of memory content at location C (C ∈ ALIT) to register A (A ∈ OREG).

The upper RDW-W bits of the memory content at location C are transferred to the MHB register.

The zero bit Z of status register STA is set, if the lower W bits of the transferred value are zero, otherwise the zero bit is cleared.

If the MHB register is selected as destination register A (A ∈ OREG), the bits 0 to RDW-W-1 of the referred memory location are transferred to MHB.

The program counter PC is incremented by the value 4.

28.17.7.4 MWR Instruction

Syntax	Operation	Status	Duration
MWR A, C	MEM(C)[W-1:0] ← A; MEM(C)[RDW-1:W] ← MHB	-	2 instruction cycles but not faster than 1+NPS clock cycles due to pipeline flushing.

Transfer W bit value of register A (A ∈ OREG) together with the MHB register to the memory at location C (C ∈ ALIT).

The W bit value of register A is stored in the LSBs (bit 0 to W-1) of the memory location.

The MHB register is stored in bits W to RDW-W-1 of the referred memory location.

The program counter PC is incremented by the value 4.

28.17.7.5 MWRL Instruction

Syntax	Operation	Status	Duration
MWRL A, C	MEM(C)[W-1:0] ← A	-	3 instruction cycles but not faster than 1+2*NPS clock cycles due to pipeline flushing.

Transfer W bit value of register A (A ∈ OREG) to memory at location C (C ∈ ALIT).

The W bit value of register A is stored in the LSBs (bit 0 to W-1) of the memory location and the bits W to RDW-W are left unchanged.

The program counter PC is incremented by the value 4.

It should be noted that this operation is not an atomic instruction.

28.17.7.6 MRDI Instruction

Syntax	Operation	Status	Duration
MRDI A, B [, C]	A ← MEM(B[RAW+USR+1:2] + C)[W-1:0] MHB ← MEM(B[RAW+USR+1:2] + C)[RDW-1:W]	Z	2 instruction cycles but not faster than 1+NPS clock cycles due to pipeline flushing.

Transfer the bits 0 to W-1 of a memory location to register A (A ∈ OREG) using indirect addressing.

The upper RDW-W bits of this memory location are transferred to MHB register.

The memory location where to read from depends on register B (B ∈ OREG) and literal C (C ∈ AOLIT) and it is defined as B[RAW+USR+1:2] + C.

If the optional operand C is not available in the assembler syntax, the MCS assembler generates code with a default value of 0 for operand C.

The zero bit Z of status register STA is set, if the transferred bits 0 to W-1 are zero, otherwise the zero bit is cleared.

If the MHB register is selected as destination register A (A ∈ OREG), the bits 0 to RDW-W-1 of the referred memory location are transferred to MHB.

The program counter PC is incremented by the value 4.

28.17.7.7 MRDIO Instruction

Syntax	Operation	Status	Duration
MRDIO A, B	A ← MEM(B[RAW+USR+1:2] + R5[RAW+USR+1:2])[W-1:0] MHB ← MEM(B[RAW+USR+1:2] + R5[RAW+USR+1:2])[RDW-1:W]	Z	2 instruction cycles but not faster than 1+NPS clock cycles due to pipeline flushing.

Transfer the bits 0 to W-1 of a memory location to register A (A ∈ XOREG) using indirect addressing with offset calculation.

The upper RDW-W bits of this memory location are transferred to MHB register.

The memory location where to read from depends on register B (B ∈ BAREG) and register R5 and it is defined as B[RAW+USR+1:2] + R5[RAW+USR+1:2].

The zero bit Z of status register STA is set, if the transferred bits 0 to W-1 are zero, otherwise the zero bit is cleared.

If the MHB register is selected as destination register A, the bits 0 to RDW-W-1 of the referred memory location are transferred to MHB.

The program counter PC is incremented by the value 4.

28.17.7.8 MWRI Instruction

Syntax	Operation	Status	Duration
MWRI A, B [, C]	MEM(B[RAW+USR+1:2] + C)[W-1:0] ← A; MEM(B[RAW+USR+1:2] + C)[RDW-1:W] ← MHB	-	2 instruction cycles but not faster than 1+NPS clock cycles due to pipeline flushing.

Transfer value of register A (A ∈ OREG) to the LSBs 0 to W-1 of a memory location using indirect addressing.

The MHB register is moved to the bits W to RDW-1 at the same memory location.

If the optional operand C is not available in the assembler syntax, the MCS assembler generates code with a default value of 0 for operand C.

The memory location where to write to depends on register B (B ∈ OREG) and literal C (C ∈ AOLIT) and it is defined as B[RAW+USR+1:2] + C.

The program counter PC is incremented by the value 4.

28.17.7.9 MWRIO Instruction

Syntax	Operation	Status	Duration
MWRIO A, B	MEM(B[RAW+USR+1:2] + R5[RAW+USR+1:2])[W-1:0] ← A; MEM(B[RAW+USR+1:2] + R5[RAW+USR+1:2])[RDW-1:W] ← MHB	-	2 instruction cycles but not faster than 1+NPS clock cycles due to pipeline flushing.

Transfer value of register A (A ∈ XOREG) to the LSBs 0 to W-1 of a memory location using indirect addressing with offset calculation.

The MHB register is moved to the bits W to RDW-1 at the same memory location.

The memory location where to write to depends on register B (B ∈ BAREG) and register R5 and it is defined as B[RAW+USR+1:2] + R5[RAW+USR+1:2].

The program counter PC is incremented by the value 4.

28.17.7.10MWRIL Instruction

Syntax	Operation	Status	Duration
MWRIL A, B	MEM(B[RAW+USR+1:0])[W-1:0] ← A;	-	3 instruction cycles but not faster than 1+2*NPS clock cycles due to pipeline flushing.

Transfer W bit value of A (A ∈ OREG) to memory using indirect addressing.

The memory location where to write to is defined by the bits 2 to RAW+1 of register B (B$\in$ OREG).

The W bit value is stored in the LSBs (bit 0 to W-1) of the memory location and the bits W to RDW-1 are left unchanged.

The program counter PC is incremented by the value 4.

It should be noted that this operation is not an atomic instruction.

28.17.7.11 POP Instruction

Syntax	Operation	Status	Duration
POP A	A ← MEM(R7[RAW+USR+1:2])[W-1:0]; MHB ← MEM(R7[RAW+USR+1:2])[RDW-1:W]; R7 ← R7 - 4; SP_CNT ← SP_CNT - 1	Z, EN	2 instruction cycles but not faster than 1+NPS clock cycles due to pipeline flushing.

Transfer the LSBs (bit 0 to W-1) from the top of stack to register A (A$\in$ OREG), followed by decrementing the stack pointer register R7 with the value 4.

The upper bits W to RDW-1 from the top of the stack are transferred to register MHB.

If the MHB register is selected as destination register A (A$\in$ OREG), the bits 0 to RDW-W-1 from the top of the stack are transferred to MHB.

The memory location for the top of the stack is identified by the bits 2 to RAW+1 of the stack pointer register R7.

The zero bit Z of status register STA is set, if the lower W bit of the transferred value is zero, otherwise the zero bit is cleared.

The program counter PC is incremented by the value 4.

The **SP_CNT** bit field inside the **MCS[i]_CH[x]_CTRL** register is decremented.

If an underflow on the **SP_CNT** bit field occurs, the STK_ERR[i]_IRQ is raised.

If an underflow on the **SP_CNT** bit field occurs and the bit **HLT_SP_OFL** of register **MCS[i]_CTRL** is set, the current MCS channel is disabled by clearing the **EN** bit of **STA**.

28.17.7.12 PUSH Instruction

Syntax	Operation	Status	Duration
PUSH A	R7 ← R7 + 4; MEM(R7[RAW+USR+1:2])[W-1:0] ← A; MEM(R7[RAW+USR+1:2])[RDW-1:W] ← MHB SP_CNT ← SP_CNT + 1;	EN	2 instruction cycles but not faster than 1+NPS clock cycles due to pipeline flushing.

Increment the stack pointer register R7 with the value 4, followed by transferring a W bit value of operand A (A$\in$ OREG) together with a MHB register to the new top of the stack. The W bit value of A is stored in the bits 0 to W-1 of the memory location.

The content of the MHB register is stored in the bit W to RDW-1 of the memory location.

The memory location for the top of the stack is referred by the bits 2 to RAW+1 of the stack pointer register.

The program counter PC is incremented by the value 4.

The SP_CNT bit field inside the MCS[i]_CH[x]_CTRL register is incremented.

If an overflow on the SP_CNT bit field occurs, the STK_ERR[i]_IRQ is raised.

If an overflow on the SP_CNT bit field occurs and the bit HLT_SP_OFL of register MCS[i]_CTRL is set, the current MCS channel is disabled by clearing the EN bit of STA.

If an overflow on the SP_CNT bit field occurs and the bit HLT_SP_OFL of register MCS[i]_CTRL is set, the memory write operation for the A and MHB is discarded.

28.17.7.13ARD Instruction

Syntax	Operation	Status	Duration
ARD A, B, C	A ← ARU(C)[W-1:0]; B ← ARU(C)[2*W-1:W]; ACB ← ARU(C)[4+2*W:2*W]	CAT, SAT	Suspends current MCS channel until ARU transfer finished.

Perform a blocking read access to the ARU and transfer both W bit values received at the ARU port to the registers A and B (A $\in$ AREG, B $\in$ AREG), whereas A holds the lower W bit ARU word and B holds the upper W bit ARU word.

If A and B refer to the same register, only the upper W bit ARU word is stored and the lower W bit ARU word is discarded.

If any transferred W bit value from the ARU should not be stored in a register, the dummy register ZERO $\in$ AREG can be selected in A or B to discard the corresponding ARU data. The binary encoding of the address for the dummy register ZERO can be chosen by an arbitrary value within the range 8 to 15.

The received ARU control bits are stored in the register ACB.

The literal C (C $\in$ ARDLIT) define the ARU address where to read from.

At the beginning of the instruction execution the CAT bit in register STA is always cleared. After the execution of the instruction the SAT flag of the register STA is updated in order to show if the transfer was successful (SAT = 1) or if the transfer failed (SAT = 0) due to a cancellation by the CPU.

The program counter PC is incremented by the value 4.

28.17.7.14ARDI Instruction

Syntax	Operation	Status	Duration
ARDI A, B	A ← ARU(R6[8:0])[W-1:0]; B ← ARU(R6[8:0])[2*W-1:W]; ACB ← ARU(R6[8:0])[4+2*W:2*W]	CAT, SAT	Suspends current MCS channel until ARU transfer finished.

Perform a blocking read access to the ARU and transfer both W bit values received at the ARU port to the registers A and B (A $\in$ AREG, B $\in$ AREG), whereas A holds the lower W bit ARU word and B holds the upper W bit ARU word.

If A and B refer to the same register, only the upper W bit ARU word is stored and the lower W bit ARU word is discarded.

If any transferred W bit value from the ARU should not be stored in a register, the dummy register ZERO $\in$ AREG can be selected in A or B to discard the corresponding ARU data. The binary encoding of the address for the dummy register ZERO can be chosen by an arbitrary value within the range 8 to 15.

The received ARU control bits are stored in the register ACB.

The read address is obtained from the bits 0 to 8 of the channels register R6.

At the beginning of the instruction execution the CAT bit in register STA is always cleared. After the execution of the instruction the SAT flag of the register STA is updated in order to show if the transfer was successful (SAT = 1) or if the transfer failed (SAT = 0) due to a cancellation by the CPU.

The program counter PC is incremented by the value 4.

28.17.7.15AWR Instruction

Syntax	Operation	Status	Duration
AWR A, B, C	ARU(C)[W-1:0] ← A; ARU(C)[2*W-1:W] ← B; ARU(C)[4+2*W:2*W] ← ACB;	CAT, SAT	Suspends current MCS channel until ARU transfer finished.

Perform a blocking write access to the ARU and transfer two W bit values to the ARU port using the registers A and B (A$\in$ OREG, B$\in$ OREG), whereas A holds the lower W bit ARU word and B holds the upper W bit ARU word.

The ARU control bits are taken from the register ACB.

The literal C (C$\in$ AWRLIT) defines an index into the pool of ARU write addresses that are used for writing data. This index is mapped to an ARU write address as shown in column "MCS write index" of table "ARU Write Addresses" in device specific appendix.

Each MCS sub module has a pool of several write addresses that can be shared between all MCS channels arbitrarily.

At the beginning of the instruction execution the CAT bit of the register STA is always cleared. After the execution of the instruction the SAT flag of the register STA is updated in order to show if the transfer was successful (SAT = 1) or if the transfer failed (SAT = 0) due to a cancellation by the CPU.

The program counter PC is incremented by the value 4.

28.17.7.16AWRI Instruction

Syntax	Operation	Status	Duration
AWRI A, B	ARU(R6[4:0])[W-1:0] ← A; ARU(R6[4:0])[2*W-1:W] ← B; ARU(R6[4:0])[4+2*W:2*W] ← ACB;	CAT, SAT	Suspends current MCS channel until ARU transfer finished.

Perform a blocking write access to the ARU and transfer two W bit values to the ARU port using the registers A and B (A$\in$ OREG, B$\in$ OREG), whereas A holds the lower W bit ARU word and B holds the upper W bit ARU word.

The ARU control bits are taken from the register ACB.

The bits 0 to 4 of the register R6 define an index into the pool of ARU write addresses that are used for writing data. This index is mapped to an ARU write address as shown in column "MCS write index" of table "ARU Write Addresses" in device specific appendix.

Each MCS sub module has a pool of several write addresses that can be shared between all MCS channels arbitrarily.

At the beginning of the instruction execution the CAT bit of the register STA is always cleared. After the execution of the instruction the SAT flag of the register STA is updated in order to show if the transfer was successful (SAT = 1) or if the transfer failed (SAT = 0) due to a cancellation by the CPU.

The program counter PC is incremented by the value 4.

28.17.7.17NARD Instruction

Syntax	Operation	Status	Duration
NARD A, B, C	A ← ARU(C)[W-1:0]; B ← ARU(C)[2*W:W]; ACB ← ARU(C)[4+2*W:2*W]	SAT	Suspends current MCS channel until the ARU is selecting the MCS channel.

Perform a non-blocking read access to the ARU trying to transfer both W bit values received at the ARU port to the registers A and B (A$\in$ AREG, B$\in$ AREG), whereas A holds the lower W bit ARU word, B holds the upper W bit ARU word, and the ACB register holds the received ARU control bits.

The literal C (C$\in$ ARDLIT) defines the ARU address where to read from.

Non-blocking ARU read means that the instruction is suspending the MCS channel until the ARU scheduler is selecting the requesting MCS channel.

If the transfer finished successfully, the bit SAT of the register STA is set and the transferred values are stored in the registers A, B, and ACB.

If the transfer failed due to missing data at the requested source, the bit SAT of the register STA is cleared and registers A, B, and ACB are not changed.

If A and B refer to the same register, only the upper W bit ARU word is stored and the lower W bit ARU word is discard.

If any transferred W bit value from the ARU should not stored in a register, the dummy register ZERO$\in$ AREG can be selected in A or B to discard the corresponding ARU data. The binary encoding of the address for the dummy register ZERO can be chosen by an arbitrary value within the range 8 to 15.

The program counter PC is incremented by the value 4.

28.17.7.18NARDI Instruction

Syntax	Operation	Status	Duration
NARDI A, B	A ← ARU(R6[8:0])[W-1:0]; B ← ARU(R6[8:0])[2*W-1:W]; ACB ← ARU(R6[8:0])[4+2*W:2*W]	SAT	Suspends current MCS channel until the ARU is selecting the MCS channel.

Perform a non-blocking read access to the ARU trying to transfer both W bit values received at the ARU port to the registers A and B (A$\in$ AREG, B$\in$ AREG), whereas A holds the lower W bit ARU word, B holds the upper W bit ARU word, and the ACB register holds the received ARU control bits. The read address is obtained from bits 0 to 8 of the channel register R6.

Non-blocking ARU read access means that the instruction suspends the MCS channel until the ARU scheduler selects the requesting MCS channel. If the transfer finished successfully, the bit SAT of the register STA is set and the transferred values are stored in the registers A, B, and ACB.

If the transfer failed due to missing data at the requested source, the bit SAT of the register STA is cleared and registers A, B, and ACB are not changed.

If A and B refer to the same register, only the upper W bit ARU word is stored and the lower 24 bit ARU word is discard.

If any transferred W bit value from the ARU should not stored in a register, the dummy register ZERO$\in$ AREG can be selected in A or B to discard the corresponding ARU data. The binary encoding of the address for the dummy register ZERO can be chosen by an arbitrary value within the range 8 to 15.

The program counter PC is incremented by the value 4.

28.17.7.19BRD Instruction

Syntax	Operation	Status	Duration
BRD A, C	A ← BUS(C)[W-1:0]; MHB ← BUS(C)[BDW-1:W]	-	Suspends current MCS channel if addressed slave inserts at least one wait cycle (e.g. accessing RAM module) otherwise 2 instruction cycles but not faster than 1+NPS clock cycles due to pipeline flushing.

Initiate a read access at the bus master interface using the address C (C $\in$ BALIT) and transfer the lower W bits of the received data to register A (A $\in$ GREG).

The upper BDW-W bits of the received data are transferred to the MHB register.

If the delay between a BRD instruction and its successor instruction is one or two system clock cycles (e.g. in accelerated scheduling mode) and the successor instruction is reading data resulting from the BRD instruction, a data hazard in the pipeline occurs resulting in a pipeline flush. this means, if very fast program execution is required (e.g. only one task is activated in accelerated scheduling mode) a program sequence like BRD R1,0x288; ADD R3,R1 (9 clock cycles) can be accelerated by reformulating the sequence as BRD R1,0x0288; NOP;NOP;NOP;ADD R3,R1; (4 clock cycles).

Since the MHB register is always transferred via AEI bus master it also figures out another data dependency, which can cause a data hazard resulting in a pipline flush. Therefore a sequence like BRD R1, 0x0288; BWR R3, 0x304 (9 clock cycles) could also be optimized by the sequence BRD R1, 0x0288; NOP; NOP; BWR R3, 0x304(4 clock cycles).

The program counter PC is incremented by the value 4.

28.17.7.20BWR Instruction

Syntax	Operation	Status	Duration
BWR A, C	BUS(C)[W-1:0] ← A; BUS(C)[BDW-1:W] ← MHB	-	Suspends current MCS channel if addressed slave inserts at least one wait cycle (e.g. accessing a RAM module) otherwise 1 cycle.

Initiate a write access at the bus master interface using the address C (C $\in$ BALIT) and transfer the content of register A (A $\in$ GREG) to the bits 0 to W-1 of the bus.

The content of the MHB register is transferred to the bits W to BDW-W-1 of the bus.

The program counter PC is incremented by the value 4.

28.17.7.21BRDI Instruction

Syntax	Operation	Status	Duration
BRDI A, B	A ← BUS(B[BAW+1:2])[W-1:0]; MHB ← BUS(B[BAW+1:2])[BDW-1:W]	-	Suspends current MCS channel if addressed slave inserts at least one wait cycle (e.g. accessing a RAM module) otherwise 1 clock cycles.

Initiate a read access at the bus master interface using indirect addressing and transfer the lower W bits of the received data to register A (A $\in$ GREG).

The upper BDW-W bits of the received data are transferred to the MHB register.

The address for the transfer is identified by the bits 2 to BAW+1 of register B (B$\in$ GREG).

If the delay between a BRDI instruction and its successor instruction is one or two system clock cycles (e.g. in accelerated scheduling mode) and the successor instruction is reading data resulting from the BRDI instruction, a data hazard in the pipeline occurs resulting in a pipeline flush. this means, if very fast program execution is required (e.g. only one task is activated in accelerated scheduling mode) a program sequence like BRD I R1,R6; ADD R3, R1 (9 clock cycles) can be accelerated by reformulating the sequence as BRDI R1, R6; NOP; NOP; NOP; ADD R3, R1; (4 clock cycles).

Since the MHB register is always transferred via AEI bus master it also figures out another data dependency, which can cause a data hazard resulting in a pipline flush. Therefore a sequence like BRDI R1, R2; BWRI R3, R4 (9 clock cycles) could also be optimized by the sequence BRDI R1, R2; NOP; NOP; BWRI R3, R4 (4 clock cycles).

The program counter PC is incremented by the value 4.

28.17.7.22BWRI Instruction

Syntax	Operation	Status	Duration
BWRI A, B	BUS(B[BAW+1:2])[W-1:0] ← A; BUS(B[BAW+1:2])[BDW-1:W] ← MHB	-	Suspends current MCS channel if addressed slave inserts at least one wait cycle (e.g. accessing a RAM module) otherwise 1 instruction cycle.

Initiate a write access at the bus master interface using the indirect addressing and transfer the content of register A (A$\in$ GREG) to the bits 0 to W-1 of the bus.

The content of the MHB register is transferred to the bits W to BDW-W-1 of the bus.

The address for the transfer is identified by the bits 2 to BAW+1 of register B (B$\in$ GREG).

The program counter PC is incremented by the value 4.

28.17.7.23ADDL Instruction

Syntax	Operation	Status	Duration
ADDL A, C	A ← A + C	Z, CY, N, V	1 instruction cycle

Perform addition operation of a register A (A$\in$ OREG) with a W bit literal value C (C$\in$ WLIT) and store the result in register A.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The carry bit CY of status register STA is set, if an unsigned overflow/underflow occurred during addition, otherwise the bit is cleared. An unsigned overflow has occurred when the result of the operation cannot be represented in the interval [0; 2^W-1], assuming that both operands A and C are unsigned values within the interval [0; 2^W-1].

The overflow bit V of status register STA is set, if a signed overflow/underflow occurred during addition, otherwise the bit is cleared. A signed overflow/underflow has occurred when the result of the operation cannot be represented in the interval [-2^{W-1}; 2^{W-1}-1], assuming that both operands A and C are signed values within the interval [-2^{W-1}; 2^{W-1}-1].

The negative bit N of status register STA equals the MSB of the operation result, in order to determine if a calculated signed result is negative (N=1) or positive (N=0), assuming that no overflow/underflow occurred.

The program counter PC is incremented by the value 4.

28.17.7.24ADD Instruction

Syntax	Operation	Status	Duration
ADD A, B	A ← A + B	Z, CY, N, V	1 instruction cycle

Perform addition operation of a register A (A ∈ XOREG) with an operand B (B ∈ XOREG). The result is stored in the register A.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The carry bit CY of status register STA is set, if an unsigned overflow occurred during addition, otherwise the bit is cleared. An unsigned overflow has occurred when the result of the operation cannot be represented in the interval [0; 2^W-1], assuming that both operands A and B are unsigned values within the interval [0; 2^W-1].

The overflow bit V of status register STA is set, if a signed overflow/underflow occurred during addition, otherwise the bit is cleared. A signed overflow/underflow has occurred when the result of the operation cannot be represented in the interval [-2^{W-1}; 2^{W-1}-1], assuming that both operands A and B are signed values within the interval [-2^{W-1}; 2^{W-1}-1].

The negative bit N of status register STA equals the MSB of the operation result, in order to determine if a calculated signed result is negative (N=1) or positive (N=0), assuming that no overflow/underflow occurred.

The program counter PC is incremented by the value 4.

28.17.7.25ADDC Instruction

Syntax	Operation	Status	Duration
ADDC A, B	A ← A + B + CY	Z, CY, N, V	1 instruction cycle

Perform addition operation of a register A (A ∈ XOREG) with an operand B (B ∈ XOREG) and the carry flag CY. The result is stored in the register A.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The carry bit CY of status register STA is set, if an unsigned overflow occurred during addition, otherwise the bit is cleared. An unsigned overflow has occurred when the result of the operation cannot be represented in the interval [0; 2^W-1], assuming that both operands A and B are unsigned values within the interval [0; 2^W-1].

The overflow bit V of status register STA is set, if a signed overflow/underflow occurred during addition, otherwise the bit is cleared. A signed overflow/underflow has occurred when the result of the operation cannot be represented in the interval [-2^{W-1}; 2^{W-1}-1], assuming that both operands A and B are signed values within the interval [$-2^{W.1}$; 2^{W-1}-1].

The negative bit N of status register STA equals the MSB of the operation result, in order to determine if a calculated signed result is negative (N=1) or positive (N=0), assuming that no overflow/underflow occurred.

The program counter PC is incremented by the value 4.

28.17.7.26SUBL Instruction

Syntax	Operation	Status	Duration
SUBL A, C	A ← A - C	Z, CY, N, V	1 instruction cycle

Perform subtraction operation of a register A (A ∈ OREG) with a W bit literal value C (C ∈ WLIT). The result is stored in register A.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The carry bit CY of status register STA is set, if an unsigned underflow occurred during subtraction, otherwise the bit is cleared. An unsigned underflow has occurred when the result of the operation cannot be represented in the interval [0; 2^W-1], assuming that both operands A and C are unsigned values within the interval [0; 2^W-1].

The overflow bit V of status register STA is set, if a signed overflow/underflow occurred during subtraction, otherwise the bit is cleared. A signed overflow/underflow has occurred when the result of the operation cannot be represented in the interval [-2^{W-1}; 2^{W-1}-1], assuming that both operands A and C are signed values within the interval [-2^{W-1}; 2^{W-1}-1].

The negative bit N of status register STA equals the MSB of the operation result, in order to determine if a calculated signed result is negative (N=1) or positive (N=0), assuming that no overflow/underflow occurred.

The program counter PC is incremented by the value 4.

28.17.7.27SUB Instruction

Syntax	Operation	Status	Duration
SUB A, B	A ← A - B	Z, CY, N, V	1 instruction cycle

Perform subtraction operation of a register A (A ∈ XOREG) with an operand B (B ∈ XOREG). The result is stored in register A.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The carry bit CY of status register STA is set, if an unsigned underflow occurred during subtraction, otherwise the bit is cleared. An unsigned underflow has occurred when the result of the operation cannot be represented in the interval [0; 2^W-1], assuming that both operands A and B are unsigned values within the interval [0; 2^W-1].

The overflow bit V of status register STA is set, if a signed overflow/underflow occurred during subtraction, otherwise the bit is cleared. A signed overflow/underflow has occurred when the result of the operation cannot be represented in the interval [-2^W-1; 2^W-1-1], assuming that both operands A and B are signed values within the interval [-2^W-1; 2^W-1-1].

The negative bit N of status register STA equals the MSB of the operation result, in order to determine if a calculated signed result is negative (N=1) or positive (N=0), assuming that no overflow/underflow occurred.

The program counter PC is incremented by the value 4.

28.17.7.28SUBC Instruction

Syntax	Operation	Status	Duration
SUBC A, B	A ← A - B - CY	Z, CY, N, V	1 instruction cycle

Perform subtraction operation of a register A (A ∈ XOREG) with an operand B (B ∈ XOREG) and the carry flag CY. The result is stored in register A.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The carry bit CY of status register STA is set, if an unsigned underflow occurred during subtraction, otherwise the bit is cleared. An unsigned underflow has occurred when the result of the operation cannot be represented in the interval [0; 2W-1], assuming that both operands A and B are unsigned values within the interval [0; 2W-1].

The overflow bit V of status register STA is set, if a signed overflow/underflow occurred during subtraction, otherwise the bit is cleared. A signed overflow/underflow has occurred when the result of the operation cannot be represented in the interval [-2^{W-1}; 2^{W-1}-1], assuming that both operands A and B are signed values within the interval [-2^{W-1}; 2^{W-1}-1].

The negative bit N of status register STA equals the MSB of the operation result, in order to determine if a calculated signed result is negative (N=1) or positive (N=0), assuming that no overflow/underflow occurred.

The program counter PC is incremented by the value 4.

28.17.7.29NEG Instruction

Syntax	Operation	Status	Duration
NEG A, B	A ← - B	Z, N, V	1 instruction cycle

Perform negation operation (2's Complement) with an operand B (B $\in$ XOREG) and store the result in a register A (A $\in$ XOREG).

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The overflow bit V of status register STA is set, if a signed overflow/underflow occurred during subtraction, otherwise the bit is cleared. A signed overflow/underflow has occurred when the result of the operation cannot be represented in the interval $[-2^{W-1}; 2^{W-1}-1]$, assuming that both operands A and B are signed values within the interval $[-2^{W-1}; 2^{W-1}-1]$.

The negative bit N of status register STA equals the MSB of the operation result, in order to determine if a calculated signed result is negative (N=1) or positive (N=0), assuming that no overflow/underflow occurred.

The program counter PC is incremented by the value 4.

28.17.7.30ANDL Instruction

Syntax	Operation	Status	Duration
ANDL A, C	A ← A AND C	Z	1 instruction cycle

Perform bitwise AND conjunction of a register A (A $\in$ OREG) with a W bit literal value C (C $\in$ WLIT) and store the result in register A.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter PC is incremented by the value 4.

28.17.7.31AND Instruction

Syntax	Operation	Status	Duration
AND A, B	A ← A AND B	Z	1 instruction cycle

Perform bitwise AND conjunction of a register A (A $\in$ XOREG) with an operand B (B $\in$ XOREG) and store the result in register A.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter PC is incremented by the value 4.

28.17.7.32ORL Instruction

Syntax	Operation	Status	Duration
ORL A, C	A ← A OR C	Z	1 instruction cycle

Perform bitwise OR conjunction of a register A (A$\in$ OREG) with a W bit literal value C (C$\in$ WLIT) and store the result in register A.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter PC is incremented by the value 4.

28.17.7.33OR Instruction

Syntax	Operation	Status	Duration
OR A, B	A ← A OR B	Z	1 instruction cycle

Perform bitwise OR conjunction of a register A (A$\in$ XOREG) with an operand B (B$\in$ XOREG) and store the result in register A.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter PC is incremented by the value 4.

28.17.7.34XORL Instruction

Syntax	Operation	Status	Duration
XORL A, C	A ← A XOR C	Z	1 instruction cycle

Perform bitwise XOR conjunction of a register A (A$\in$ OREG) with a W bit literal value C (C$\in$ WLIT) and store the result in register A.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter PC is incremented by the value 4.

28.17.7.35XOR Instruction

Syntax	Operation	Status	Duration
XOR A, B	A ← A XOR B	Z	1 instruction cycle

Perform bitwise XOR conjunction of a register A (A$\in$ XOREG) with an operand B (B$\in$ XOREG) and store the result in register A.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter PC is incremented by the value 4.

28.17.7.36SHR Instruction

Syntax	Operation	Status	Duration
SHR A, C	A ← A >> C	Z, CY	1 instruction cycle

Perform right shift operation C (C$\in$ SFTLIT) times of register A (A$\in$ XOREG). The MSBs that are shifted into A are cleared.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The carry bit CY of status register STA is updated to the last LSB that is shifted out of the register. If the shift value C is 0 the carry bit CY is cleared.

The program counter PC is incremented by the value 4.

28.17.7.37SHL Instruction

Syntax	Operation	Status	Duration
SHL A, C	A ← A << C	Z, CY	1 instruction cycle

Perform left shift operation C ($C_{\in}$ SFTLIT) times of register A ($A_{\in}$ XOREG). The LSBs that are shifted into A are cleared.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The carry bit CY of status register STA is updated to the previous MSB that is shifted out of the register. If the register A contains less than W bits or if C is 0, the carry bit CY is always cleared.

The program counter PC is incremented by the value 4.

28.17.7.38ASRU Instruction

Syntax	Operation	Status	Duration
ASRU A, B	A ← A >> B	Z	1 instruction cycle

Perform arithmetic unsigned right shift operation, which means that the unsigned operand of register A ($A_{\in}$ XOREG) is right shifted B times ($B_{\in}$ XOREG). Operand B is also an unsigned type. The MSBs that are shifted into A are cleared.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter PC is incremented by the value 4.

28.17.7.39 ASRS Instruction

Syntax	Operation	Status	Duration
ASRS A, B	A ← A >> B	Z	1 instruction cycle

Perform arithmetic signed right shift operation, which means that the signed operand of register A ($A_{\in}$ XOREG) is right shifted B times ($B_{\in}$ XOREG). Operand B is an unsigned type. The operation also performs a sign extension, which means that value of the MSBs that are shifted into A are determined by the MSB of the original operand A.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter PC is incremented by the value 4.

28.17.7.40 ASL Instruction

Syntax	Operation	Status	Duration
ASL A, B	A ← A << B	Z, CY, V	1 instruction cycle

Perform arithmetic left shift operation for signed and unsigned numbers, which means that the operand of register A ($A_{\in}$ XOREG) is left shifted B times ($B_{\in}$ XOREG). Operand B is always an unsigned type.

The carry bit CY of status register STA is set, if an unsigned overflow occurred during shifting, otherwise the bit is cleared. An unsigned overflow has occurred if the calculated result A*2B cannot be represented in the interval [0; 2^W-1], assuming that both operands A and B are unsigned values within the interval [0; 2^W-1].

The overflow bit V of status register STA is set, if a signed overflow/underflow occurred during shifting, otherwise the bit is cleared. A signed overflow/underflow has occurred when the calculated result A*2B cannot be represented in the interval [-2^{W-1}; 2^{W-1}-1], assuming that signed operand A is within the interval [-2^{W-1}; 2^{W-1}-1] and the unsigned operand B is within the interval [0; 2^{W-1}-1].

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter PC is incremented by the value 4.

28.17.7.41 MULU Instruction

Syntax	Operation	Status	Duration
MULU A, B[, C]	[[R4,] A] ← A[(C-1):0] * B[(C-1):0]	Z	1 instruction cycle

Perform an unsigned multiplication operation of an operand A (A$_\in$ XOREG) with an operand B (B$_\in$ XOREG). The multiplication is only performed with the bits 0 to C-1 (C$_\in$ BWSLIT) of both operands A and B and the bits C to W-1 are ignored.

If C is less than or equal to W/2, the product of the multiplication is stored in register A and register R4 is left unchanged.

If C is greater than W/2, the bits 0 to W-1 are stored in A and the bits W to 2*C-1 are stored in R4. The results stored in the registers are always zero extended to W bits.

If the optional operand C is not specified in the assembler code, the MCS assembler generates code with a default value of W for operand C.

The zero bit Z of status register STA is set, if the calculated product is zero, otherwise the zero bit is cleared.

If the delay between a MULU instruction and its successor instruction is one clock cycle (e.g. in accelerated scheduling mode) and the successor instruction is either a WURM, WURMX, WURCX, WUCE, BRDI, BWR or BWRI instruction that is accessing the multiplication result as argument, a data hazard in the pipeline occurs resulting in a pipeline flush.

This means, if very fast program execution is required (e.g. only one task is activated in accelerated scheduling mode) a program sequence like MULU R1, R2; BWRI R1, R3 (9 clock cycles) can be accelerated by reformulating the sequence as MULU R1, R2; NOP; BWRI R1, R3; (3 clock cycles).

The program counter PC is incremented by the value 4.

28.17.7.42 MULS Instruction

Syntax	Operation	Status	Duration
MULS A, B[, C]	[[R4,] A] ← A[(C-1):0] * B[(C-1):0]	Z, N	1 instruction cycle

Perform a signed multiplication operation of an operand A (A$_\in$ XOREG) with an operand B (B$_\in$ XOREG). The multiplication is only performed with the bits 0 to C-1 (C$_\in$ BWSLIT) of both operands A and B, in which bit C-1 is used as sign bit (-2^(C-1)) and the bits C to W-1 are ignored. If C is less than or equal to W/2, the product of the multiplication is stored in register A and register R4 is left unchanged. If C is greater than W/2, the bits 0 to W-1 are stored in A and the bits W to 2*C-1 are stored in R4. The results stored in the registers are always sign extended to W bits.

If the optional operand C is not specified in the assembler code, the MCS assembler generates code with a default value of W for operand C.

The zero bit Z of status register STA is set, if the calculated product is zero, otherwise the zero bit is cleared.

The negative bit N of status register STA equals the MSB of the operation result, in order to determine if a calculated signed result is negative (N=1) or positive (N=0).

If the delay between a MULS instruction and its successor instruction is one clock cycle (e.g. in accelerated scheduling mode) and the successor instruction is either a WURM,WURMX,WURCX,WUCE,BRDI,BWR or BWRI instruction that is accessing the multiplication result as argument, a data hazard in the pipeline occurs resulting in a pipeline flush.

This means, if very fast program execution is required (e.g. only one task is activated in accelerated scheduling mode) a program sequence like MULS R1, R2; BWRI R1, R3 (9 clock cycles) can be accelerated by reformulating the sequence as MULS R1, R2; NOP; BWRI R1,R 3; (3 clock cycles).

The program counter PC is incremented by the value 4.

28.17.7.43 DIVU Instruction

Syntax	Operation	Status	Duration
DIVU A, B[, C]	R4 ← A[(C-1):0] - B[(C-1):0] * | A[(C-1):0] / B[(C-1):0] |; A ← | A[(C-1):0] / B[(C-1):0] |	CY, Z, ERR	C instruction cycles but not faster than C+NPS-1 clock cycles due to pipeline flushing.

Perform an unsigned division operation of operand A (A $\in$ XOREG \\ {R4, B}) divided by operand B (B $\in$ XOREG \\ {R4, A}). The division is only performed with the bits 0 to C-1 (C $\in$ BWSLIT) of the operands and the remaining bits C to W-1 are ignored. This means that the dynamic range of A and B is defined in the interval [0; 2C-1]. The integral part of the quotient is stored in the register A and the remainder of the division is stored in register R4. The resulting quotient A and remainder R4 are always zero extended to W bits.

If the optional operand C is not specified in the assembler code, the MCS assembler generates code with a default value of W for operand C.

If the bits 0 to C-1 of operand B are zero, the MCS channel is disabled and the ERR bit in the status register STA is set.

The zero bit Z of status register STA is set, if the calculated quotient in A is zero, otherwise the zero bit is cleared.

The carry bit CY of status register STA is set, if the calculated remainder in R4 is not zero, otherwise the zero bit is cleared.

The program counter PC is incremented by the value 4.

28.17.7.44 DIVS Instruction

Syntax	Operation	Status	Duration
DIVS A, B[, C]	R4 ← A[(C-1):0] - B[(C-1):0] * | A[(C-1):0] / B[(C-1):0] |; A ← | A[(C-1):0] / B[(C-1):0] |	CY, Z, N, V, ERR	C + 4 instruction cycles but not faster than C+3+NPS clock cycles due to pipeline flushing.

Perform a signed division operation of operand A (A $\in$ XOREG \\ {R4, B}) divided by operand B (B $\in$ XOREG \\ {R4, A}). The division is only performed with the bits 0 to C-1 (C $\in$ BWSLIT) of both operands, in which bit C-1 is used as sign bit (-2^{C-1}) and the bits C to W-1 are ignored. This means that the dynamic range of A[(C-1):0] and B[(C-1):0] is defined in the interval [-2^{C-1}; $2^{C-1}-1$]. The integral part of the quotient is stored in the register A and the remainder

of the division is stored in register R4. The resulting quotient A and remainder R4 are always sign extended to W bits. The integral part of the quotient is always truncated towards 0. The sign of the remainder is always the same sign as the dividend A[(C-1):0]. The absolute value of the remained is always less than the divisor B[(C-1):0].

If the optional operand C is not specified in the assembler code, the MCS assembler generates code with a default value of W for operand C.

If the bits 0 to C-1 of operand B are zero, the MCS channel is disabled and the ERR bit in the status register STA is set.

The zero bit Z of status register STA is set, if the calculated quotient in A is zero, otherwise the zero bit is cleared.

The carry bit CY of status register STA is set, if the calculated remainder in R4 is not zero, otherwise the zero bit is cleared.

The overflow bit V of status register STA is set, if the calculated quotient in A cannot be represented in the interval $[-2^{W}-1; 2^{W-1}-1]$, otherwise the overflow bit is cleared.

The negative bit N of status register STA equals the MSB of the quotient, in order to determine if a calculated signed result is negative (N=1) or positive (N=0).

The program counter PC is incremented by the value 4.

28.17.7.45 MINU Instruction

Syntax	Operation	Status	Duration
MINU A, B	A ← MIN(A, B)	Z	1 instruction cycle

Determine the minimum of an unsigned operand A (A $\in$ XOREG) and an unsigned operand B (B $\in$ XOREG). If A is less than or equal to B, A is left unchanged. Otherwise, if A is greater than B, the operand B is moved to A.

The zero bit Z of status register STA is set, if the calculated result of A is zero, otherwise the zero bit is cleared.

28.17.7.46 MINS Instruction

Syntax	Operation	Status	Duration
MINS A, B	A ← MIN(A, B)	Z	1 instruction cycle

Determine the minimum of a signed operand A (A $\in$ XOREG) and a signed operand B (B $\in$ XOREG). If A is less than or equal to B, A is left unchanged. Otherwise, if A is greater than B, the operand B is moved to A.

The zero bit Z of status register STA is set, if the calculated result of A is zero, otherwise the zero bit is cleared.

28.17.7.47 MAXU Instruction

Syntax	Operation	Status	Duration
MAXU A, B	A ← MAX(A, B)	Z	1 instruction cycle

Determine the maximum of an unsigned operand A (A $\in$ XOREG) and an unsigned operand B (B $\in$ XOREG). If A is greater than or equal to B, A is left unchanged. Otherwise, if A is less than B, the operand B is moved to A.

The zero bit Z of status register STA is set, if the calculated result of A is zero, otherwise the zero bit is cleared.

28.17.7.48 MAXS Instruction

Syntax	Operation	Status	Duration
MAXS A, B	A ← MAX(A, B)	Z	1 instruction cycle

Determine the maximum of a signed operand A (A ∈ XOREG) and a signed operand B (B ∈ XOREG). If A is greater than or equal to B, A is left unchanged. Otherwise, if A is less than B, the operand B is moved to A.

The zero bit Z of status register STA is set, if the calculated result of A is zero, otherwise the zero bit is cleared.

28.17.7.49ATUL Instruction

Syntax	Operation	Status	Duration
ATUL A, C	A - C	Z, CY	1 instruction cycle

Arithmetic test with an unsigned operand A (A ∈ OREG) and an unsigned W bit literal value C (C ∈ WLIT).

The carry bit CY of status register STA is set if unsigned operand A is less than unsigned literal C.

Otherwise, the carry bit CY of status register STA is cleared if unsigned operand A is greater than or equal to unsigned literal C.

The zero bit Z of status register STA is set, if A equals to C.

Otherwise, the zero bit Z of status register STA is cleared, if A is unequal to C.

The program counter PC is incremented by the value 4.

28.17.7.50ATU Instruction

Syntax	Operation	Status	Duration
ATU A, B	A - B	Z, CY	1 instruction cycle

Arithmetic Test with an unsigned operand A (A ∈ XOREG) and an unsigned operand B (B ∈ XOREG).

The carry bit CY of status register STA is set if unsigned operand A is less than unsigned operand B.

Otherwise, the carry bit CY of status register STA is cleared if unsigned operand A is greater than or equal to unsigned operand B.

The zero bit Z of status register STA is set, if A equals to B.

Otherwise, the zero bit Z of status register STA is cleared, if A is unequal to B.

The program counter PC is incremented by the value 4.

28.17.7.51ATSL Instruction

Syntax	Operation	Status	Duration
ATSL A, C	A - C	Z, CY	1 instruction cycle

Arithmetic Test with a signed operand A (A ∈ OREG) and a signed W bit literal value C (C ∈ WLIT).

The carry bit CY of status register STA is set if signed operand A is less than signed literal C.

Otherwise, the carry bit CY of status register STA is cleared if signed operand A is greater than or equal to signed literal C.

The zero bit Z of status register STA is set, if A equals to C.

Otherwise, the zero bit Z of status register STA is cleared, if A is unequal to C.

The program counter PC is incremented by the value 4.

28.17.7.52ATS Instruction

Arithmetic Test with a signed operand A (A$\in$ XOREG) and a signed operand B (B$\in$ XOREG).

Syntax	Operation	Status	Duration
ATS A, B	A - B	Z, CY	1 instruction cycle

The carry bit CY of status register STA is set if unsigned operand A is less than signed operand B.

Otherwise, the carry bit CY of status register STA is cleared if signed operand A is greater than or equal to signed operand B.

The zero bit Z of status register STA is set, if A equals to B.

Otherwise, the zero bit Z of status register STA is cleared, if A is unequal to B.

The program counter PC is incremented by the value 4.

28.17.7.53BTL Instruction

Syntax	Operation	Status	Duration
BTL A, C	A AND C	Z	1 instruction cycle

Bit test of an operand A (A$\in$ OREG) with a W bit literal bit mask C (C$\in$ WLIT).

The bit test is performed by applying a bitwise logical AND operation with operand A and the bit mask C without storing the result.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter PC is incremented by the value 4.

28.17.7.54BT Instruction

Syntax	Operation	Status	Duration
BT A, B	A AND B	Z	1 instruction cycle

Bit test of an operand A (A$\in$ XOREG) with an operand B (B$\in$ XOREG), whereas usually one of the operands is a register holding a bit mask.

The bit test is performed by applying a bitwise logical AND operation with register A and register B without storing the result.

The zero bit Z of status register STA is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter PC is incremented by the value 4.

28.17.7.55SETB Instruction

Syntax	Operation	Status	Duration
SETB A, B	A[B[4:0]] ←1	Z	1 instruction cycle

Set the B[4:0]-th bit (B ∈ XOREG) of an operand A (A ∈ XOREG) to true. Only the bits 0 to 4 of operand B are used as bit index of operand A and the other bits of B are ignored. If the value B[4:0] is greater than or equal to W, the operation of SETB does not modify operand A but the status flag Z is updated.

The zero bit Z of status register STA is set if the modified value of A is zero, otherwise the bit Z is cleared. The Z bit is set e.g. if the B[4:0]-th bit of A is not writable, its value is zero and all other bits of A are cleared.

The program counter PC is incremented by the value 4.

28.17.7.56CLRB Instruction

Syntax	Operation	Status	Duration
CLRB A, B	A[B[4:0]] ← 0	Z	1 instruction cycle

Clear the B[4:0]-th bit (B ∈ XOREG) of an operand A (A ∈ XOREG). Only the bits 0 to 4 of operand B are used as bit index of operand A and the other bits of B are ignored. If the value B[4:0] is greater than or equal to W, the operation of CLRB does not modify operand A but the status flag Z is updated.

The zero bit Z of status register STA is set if the modified value of A is zero, otherwise the zero bit is cleared.

The program counter PC is incremented by the value 4.

28.17.7.57XCHB Instruction

Syntax	Operation	Status	Duration
XCHB A, B	A[B[4:0]] ← CY	Z, CY	1 instruction cycle

Exchange the B[4:0]-th bit (B ∈ XOREG) of an operand A (A ∈ XOREG) with the CY bit in the status register. Only the bits 0 to 4 of operand B are used as bit index of operand A and the other bits of B are ignored. If the value B[4:0] is greater than or equal to W, the operation of XCHB does not modify operand A but the status flag Z is updated and the bit CY is cleared.

The zero bit Z of status register STA is set if the modified value of A is zero, otherwise the zero bit is cleared.

The program counter PC is incremented by the value 4.

28.17.7.58JMP Instruction

Syntax	Operation	Status	Duration
JMP C	PC ← C << 2	-	1 instruction cycle but not faster than NPS clock cycles due to pipeline flushing.

Execute unconditional jump to the memory location C (C ∈ ALIT).

The program counter PC is loaded with literal C.

28.17.7.59JBS Instruction

Syntax	Operation	Status	Duration
JBS A, B, C	PC ← C << 2 if A[B] is set PC ← PC + 4 if A[B] is clear	-	1 instruction cycle but if the jump is executed, it is not faster than NPS clock cycles due to pipeline flushing.

Execute conditional jump to the memory location C (C $\in$ ALIT).

The program counter PC is loaded with literal C if the bit at position B (B $\in$ BITLIT) of operand A (A $\in$ OREG) is set.

Otherwise, if the bit is cleared the program counter PC is incremented by the value 4.

28.17.7.60JBC Instruction

Syntax	Operation	Status	Duration
JBC A, B, C	PC ← C << 2 if A[B] is clear PC ← PC + 4 if A[B] is set	-	1 instruction cycle but if the jump is executed it is not faster than NPS clock cycles due to pipeline flushing.

Execute conditional jump to the memory location C (C $\in$ ALIT).

The program counter PC is loaded with literal C if the bit at position B (B $\in$ BITLIT) of operand A (A $\in$ OREG) is cleared.

Otherwise, if the bit is set the program counter PC is incremented by the value 4.

28.17.7.61CALL Instruction

Syntax	Operation	Status	Duration
CALL C	R7 ← R7 + 4; MEM(R7[RAW+USR+1:2])[RAW+USR+1:0] ← PC + 4; PC ← C << 2; SP_CNT ← SP_CNT + 1	EN	2 instruction cycles but not faster than 2*NPS clock cycles due to pipeline flushing.

Call subprogram at memory location C (C $\in$ ALIT).

The stack pointer register R7 is incremented by the value 4.

The memory location for the top of the stack is identified by the bits 2 to RAW+1 of the stack pointer register.

After the stack pointer is incremented, the incremented value of the PC is transferred to the top of the stack.

The program counter PC is loaded with literal C.

The **SP_CNT** bit field inside the **MCS[i]_CH[x]_CTRL** register is incremented.

If an overflow on the **SP_CNT** bit field occurs, the *STK_ERR[i]_IRQ* is raised.

If an overflow on the **SP_CNT** bit field occurs and the bit **HLT_SP_OFL** of register **MCS[i]_CTRL_STAT** is set, the channel current MCS channel is disabled by clearing the EN bit of STA.

If an overflow on the SP_CNT bit field occurs and the bit **HLT_SP_OFL** of register **MCS[i]_CTRL_STAT** is set, the memory write operation of the incremented PC is discarding.

28.17.7.62RET Instruction

Syntax	Operation	Status	Duration
RET	PC ← MEM(R7[RAW+USR+1:2])[RAW+USR+1:2] << 2; R7 ← R7 - 4; SP_CNT ← SP_CNT - 1	EN	2 instruction cycles but not faster than 2*NPS clock cycles due to pipeline flushing.

Return from subprogram.

The program counter PC is loaded with current value on the top of the stack.

Finally, the stack pointer register R7 is decremented by the value 4.

The memory location for the top of the stack is identified by the bits 2 to RAW+1 of the stack pointer register.

The SP_CNT bit field inside the **MCS[i]_CH[x]_CTRL** register is decremented.

If an underflow on the SP_CNT bit field occurs, the *STK_ERR[i]_IRQ* is raised.

If an underflow on the SP_CNT bit field occurs and the bit **HLT_SP_OFL** of register **MCS[i]_CTRL** is set, the channel current MCS channel is disabled by clearing the **EN** bit of **STA**.

28.17.7.63JMPI Instruction

Syntax	Operation	Status	Duration
JMPI	PC ← R6[RAW+USR+1:2] << 2	-	1 instruction cycle but not faster than NPS clock cycles due to pipeline flushing.

Execute indirect unconditional jump to the memory location provided in register R6. The destination address is only defined by the bits 2 to RAW+USR+1 of R6 and the other bits are ignored.

The program counter PC is loaded with (R6[RAW+USR+1:2] << 2).

28.17.7.64JBSI Instruction

Syntax	Operation	Status	Duration
JBSI A, B	PC ← R6[RAW+USR+1:2] << 2 if A[B] is set PC ← PC + 4 if A[B] is clear	-	1 instruction cycle but if the jump is executed it is not faster than NPS clock cycles due to pipeline flushing.

Execute indirect conditional jump to the memory location provided in register R6. The destination address is only defined by the bits 2 to RAW+USR+1 of R6 and the other bits are ignored.

The program counter PC is loaded with (R6[RAW+USR+1:2] << 2) only if the bit at position B ($B_{\in}$ XBITLIT) of operand A ($A_{\in}$ XOREG) is set.

Otherwise, if the bit is cleared the program counter PC is incremented by the value 4.

28.17.7.65JBCI Instruction

Syntax	Operation	Status	Duration
JBCI A, B	PC ← R6[RAW+USR+1:2] << 2 if A[B] is set PC ← PC + 4 if A[B] is clear	-	1 instruction cycle but if the jump is executed it is not faster than NPS clock cycles due to pipeline flushing.

Execute indirect conditional jump to the memory location provided in register R6. The destination address is only defined by the bits 2 to RAW+USR+1 of R6 and the other bits are ignored.

The program counter PC is loaded with (R6[RAW+USR+1:2] << 2) only if the bit at position B (B$\in$ XBITLIT) of operand A (A$\in$ XOREG) is cleared.

Otherwise, if the bit is set the program counter PC is incremented by the value 4.

28.17.7.66CALLI Instruction

Syntax	Operation	Status	Duration
CALLI	R7 ← R7 + 4; MEM(R7[RAW+USR+1:2])[RAW+USR+1:0] ← PC + 4; PC ← R6[RAW+USR+1:2] << 2; SP_CNT ← SP_CNT + 1	EN	2 instruction cycles but not faster than 2*NPS clock cycles due to pipeline flushing.

Call subprogram indirectly, where the register R6 is identifying the target memory location. The destination address is only defined by the bits 2 to RAW+USR+1 of R6 and the other bits are ignored.

The stack pointer register R7 is incremented by the value 4.

The memory location for the top of the stack is identified by the bits 2 to RAW+1 of the stack pointer register.

After the stack pointer is incremented, the incremented value of the PC is transferred to the top of the stack.

The program counter PC is loaded with (R6[RAW+USR+1:2] << 2),

The **SP_CNT** bit field inside the **MCS[i]_CH[x]_CTRL** register is incremented.

If an overflow on the **SP_CNT** bit field occurs, the *STK_ERR[i]_IRQ* is raised.

If an overflow on the **SP_CNT** bit field occurs and the bit **HLT_SP_OFL** of register **MCS[i]_CTRL_STAT** is set, the channel current MCS channel is disabled by clearing the **EN** bit of **STA**.

If an overflow on the **SP_CNT** bit field occurs and the bit **HLT_SP_OFL** of register **MCS[i]_CTRL_STAT** is set, the memory write operation of the incremented PC is discarding.

28.17.7.67WURM Instruction

Syntax	Operation	Status	Duration
WURM A, B, C	Wait until register match.	CWT	Suspends current MCS channel. If the MCS is configured in Single Prioritization or Multiple Prioritization Scheduling Mode, the worst case latency for reactivating a prioritized MCS channel is 2+NPS clock cycles. This is the delay between the match event of the corresponding WURM instruction and the beginning of the following MCS instruction.

Suspend current MCS channel until the following register match condition occurs:

A = (B AND MASK),

whereas A ∈ OREG, B ∈ OREG, AND is a bitwise AND operation with bitmask MASK.

The bits 16 to 23 of MASK are set to true and the bits 0 to 15 are copied from the instructions literal C ∈ MSKLIT. If the match condition evaluates to true, the suspended MCS channel is resumed and the program counter PC is incremented by the value 4 meaning that the MCS channel continues its program. However, if the match condition is true at the beginning of the instruction execution, the instruction does not suspend the channel and the program counter PC is incremented by the value 4.

At the beginning of the instruction execution the CWT bit in the register STA is always cleared. After the execution of the instruction the CWT bit is updated in order to show if the instruction finished successfully (CWT = 0) or it was canceled by the CPU (CWT = 1).

This instruction can be used to wait for one or more trigger events generated by other MCS channels or the CPU. In this case register B is the trigger register **STRG**, A is a general purpose register holding the bits with the trigger condition to wait for and C is the bitmask that enables trigger bits of interest. The trigger bits can be set by other MCS channels with a write access (e.g. using a MOVL instruction) to the **STRG** register or the CPU with a write access to the **MCS[i]_STRG** register. The trigger bits are not cleared automatically by hardware after resuming an MCS channel, but they have to be cleared explicitly with a write access to the register **CTRG** by the MCS channel or with a write access to the register **MCS[i]_CTRG** by the CPU. Please note that more than one channel can wait for the same trigger bit to continue.

The instruction can also be used to wait on a specific time/angle event provided by the TBU. In this case register B is the interesting TBU register TBU_TS0 or TBU_TS1, register A is a general purpose register holding the value to wait for and bitmask C should be set to 0xFFFF.

28.17.7.68WURMX Instruction

Syntax	Operation	Status	Duration
WURMX A, B	Wait until extended register match.	CWT	Suspends current MCS channel. If the MCS is configured in Single Prioritization or Multiple Prioritization Scheduling Mode, the worst case latency for reactivating a prioritized MCS channel is 2+NPS clock cycles. This is the delay between the match event of the corresponding WURMX instruction and the beginning of the following MCS instruction.

Suspend current MCS channel until the following register match condition occurs:

A = B AND R6,

whereas A ∈ OREG, B ∈ WXREG, and AND is a bitwise AND operation.

If the match condition evaluates to true, the suspended MCS channel is resumed and the program counter PC is incremented by the value 4 meaning that the MCS channel continues its program. However, if the match condition is true at the beginning of the instruction execution, the instruction does not suspend the channel and the program counter PC is incremented by the value 4.

At the beginning of the instruction execution the CWT bit in the register STA is always cleared. After the execution of the instruction the CWT bit is updated in order to show if the instruction finished successfully (CWT = 0) or it was canceled by the CPU (CWT = 1).

28.17.7.69WURCX Instruction

Syntax	Operation	Status	Duration
WURCX A, B	Wait until extended register change.	CWT	Suspends current MCS channel. If the MCS is configured in Single Prioritization or Multiple Prioritization Scheduling Mode, the worst case latency for reactivating a prioritized MCS channel is 2+NPS clock cycles. This is the delay between the match event of the corresponding WURCX instruction and the beginning of the following MCS instruction.

Suspend current MCS channel until the following register change condition occurs:

$A \neq B$ AND R6,

whereas $A \in$ OREG, $B \in$ WXREG, and AND is a bitwise AND operation.

If the change condition evaluates to true, the suspended MCS channel is resumed and the program counter PC is incremented by the value 4 meaning that the MCS channel continues its program. However, if the change condition is true at the beginning of the instruction execution, the instruction does not suspend the channel and the program counter PC is incremented by the value 4.

At the beginning of the instruction execution the CWT bit in the register STA is always cleared. After the execution of the instruction the CWT bit is updated in order to show if the instruction finished successfully (CWT = 0) or it was canceled by the CPU (CWT = 1).

The WURCX instruction can be used for observation of volatile registers (e.g. register DSTAX) in order to react on status signal changes.

28.17.7.70WUCE Instruction

Syntax	Operation	Status	Duration
WUCE A, B	Wait until cyclic event.	CWT	Suspends current MCS channel. If the MCS is configured in Single Prioritization or Multiple Prioritization Scheduling Mode, the worst case latency for reactivating a prioritized MCS channel is 2+NPS clock cycles. This is the delay between the match event of the corresponding WURE instruction and the beginning of the following MCS instruction.

Suspend current MCS channel until a cyclic event compare matches.

The meaning of a cyclic event is described in section ‘Cyclic Event Compare’. The WUCE instruction can be used to synchronize an MCS program to a cyclic event generated by a TBU channel. If the event is in the future, the MCS channel suspends until the event occurs. If the event is in the past, the WUCE instruction is finished immediately.

The cyclic event compare is used to detect time base overflows and to guarantee, that a compare match event can be set up for the future even when the time base will first overflow and then reach the compare value. Please note, that for a correct behavior of this cyclic event compare, the compare value must not be specified larger/smaller than half of the range of the total time base value (0x7FFFFF).

The actual implementation of the WUCE implementation simply performs the subtraction B - A with each clock cycle and it suspends the MCS channel as long as bit W-1 of the subtraction result is set. If the subtration result is cleared the MCS channel is resumed immediatly.

In order to setup a WUCE instruction correctly, the counting direction of the TBU channel has to be considered. If the TBU channel is counting forward (incrementing), the operand A (A$_{\in}$ OREG) must refer the compare value and operand B (B$_{\in}$ OREG) must refer the desired TBU counter register (e.g. TBU_TS0). On the other hand, if the TBU channel is counting backward (decrementing), the operand A refers the desired TBU counter register and operand B refers the compare value.

If the comparison condition evaluates to true, the suspended MCS channel is resumed and the program counter PC is incremented by the value 4 meaning that the MCS channel continues its program. However, if the condition is true at the beginning of the instruction execution, the instruction does not suspend the channel and the program counter PC is incremented by the value 4.

At the beginning of the instruction execution the CWT bit in the register STA is always cleared. After the execution of the instruction the CWT bit is updated in order to show if the instruction finished successfully (CWT = 0) or it was canceled by the CPU (CWT = 1).

28.17.7.71NOP Instruction

Syntax	Operation	Status	Duration
NOP	-	-	1 instruction cycle

No operation is performed.

The program counter PC is incremented by the value 4.

28.17.8 MCS Internal Register Description

This section describes the MCS internal registers that can be directly addressed with the MCS instruction set. Many of the registers can also be addressed by the CPU but with another Register Label (for details see section 15.11). Some of the internal registers are also shared between neighboring MCS channels.

28.17.8.1 MCS Internal Register Overview

Table 65 MCS Internal Register Overview

Register Name	Description	see Page
R[y] (y: 0…7)	General Purpose Register y	391
RS[y] (y: 0…7)	Mirror of succeeding channels register R[y]	391
STA	Status Register	392
ACB	ARU Control Bit Register	395
CTRG	Clear Trigger Bits Register	396
STRG	Set Trigger Bits Register	400
TBU_TS0	TBU Timestamp TS0 Register	400
TBU_TS1	TBU Timestamp TS1 Register	401
TBU_TS2	TBU Timestamp TS2 Register	401
MHB	Memory High Byte Register	401
GMI0	GTM Module Interrupt 0 Register	402

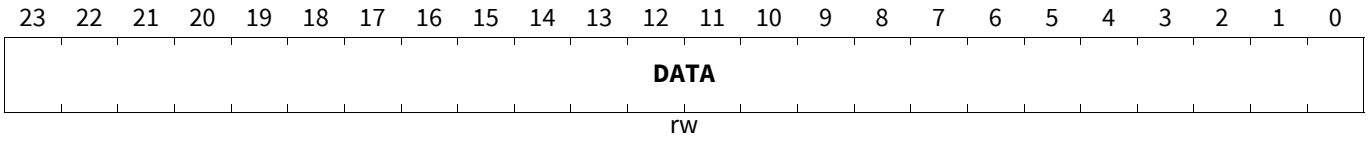
Table 65 MCS Internal Register Overview (cont'd)

Register Name	Description	see Page
GMI1	GTM Module Interrupt 1 Register	403
DSTA	DPLL Status Register	405
DSTAX	DPLL Extended Status Register	406

28.17.9 MCS Internal Register Description

28.17.9.1 Register R[y]

Ry (index)
General Purpose Register y **(0_H+y)** **Reset Value: 000000_H**



Field	Bits	Type	Description
DATA	23:0	rw	**Data field of general purpose register**

*Note: Register **R4** is also used as destination register of upper multiplication result from instructions MULU and MULS.*

*Note: Register **R5** is also used as offset register for the instructions MRDIO and MWRIO.*

*Note: Register **R6** is also used as a mask register for the instruction WURMX and WURCX.*

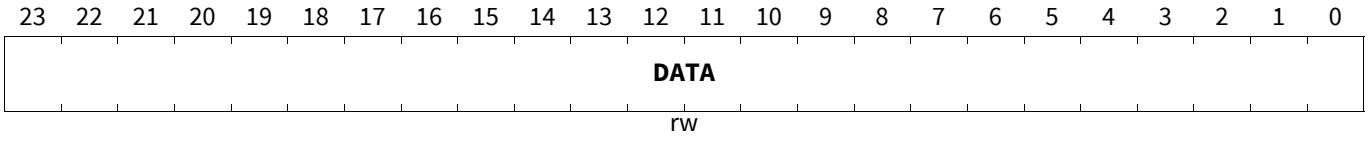
*Note: Register **R6** is also used as address destination register for the instructions JMPI, JBSI, JBCI, and CALLI.*

*Note: Register **R6** used also as index/address register for indirect ARU addressing instructions.*

*Note: Register **R7** is also used as stack pointer register, if stack operations are used in the MCS micro program.*

28.17.9.2 Register RS[y]

RSy (index)
Mirror of succeeding channels register R[y] **(10_H+y)** **Reset Value: 000000_H**



Field	Bits	Type	Description
DATA	23:0	rw	**Data field of general purpose register**

*Note: The register **RS[y]** (with y = 0…7) mirrors the internal general purpose register **R[y]** of the succeeding MCS channel. The successor of MCS channel T-1 is MCS channel 0.*

*Note: The registers **RS[y]** can only be accessed if bit **EN_XOREG** of register **MCS[i]_CTRL_STAT** is set.*

28.17.9.3 Register STA

STA
Status Register **(08_H)** **Reset Value: 000000_H**

23	22	21	20	19	18	17	16
Reserved					SP_CNT		
		r				r	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					SAT	CWT	CAT	N	V	Z	CY	MCA	ERR	IRQ	EN
r					r	r	r	r	r	r	r	rw	rw	rw	rw

Field	Bits	Type	Description
EN	0	rw	**Enable current MCS channel** 0_B Disable current MCS channel. 1_B Enable current MCS channel.
IRQ	1	rw	**Trigger IRQ** 0_B No triggered IRQ signal. 1_B Trigger IRQ signal. Note: An MCS channel triggers an IRQ by writing value 1 to bit IRQ. Writing a value 0 to this bit does not cancel the IRQ, and thus has no effect. Note: This bit mirrors bit 0 of the register **MCS[i]_CH[x]_IRQ_NOTIFY**. Note: The IRQ bit can only be cleared by CPU, by writing a 1 to the corresponding **MCS[i]_CH[x]_IRQ_NOTIFY** register (see **Register MCS[i]_CH[x]_IRQ_NOTIFY**). Note: An MCS channel can read the IRQ bit in order to determine the current state of the IRQ handling. The MCS channel reads a value 1 if an IRQ was released but not cleared by CPU. If an MCS channel reads a value 0 no IRQ was released or it has been cleared by CPU. Note: If NPS > 5 and an MCS program triggers the IRQ (e.g. by MOVL STA, 0x2) the actual interrupt event is delayed by NPS-5 clock cycles, which means that an immediate read of the interrupt notify flag (e.g. by MOV R2, STA) may signalize the state of the IRQ bit before the trigger.

Field	Bits	Type	Description
ERR	2	rw	**Set Error Signal** 0_B No Error occurred. 1_B Error occurred. Note: The **ERR** bit of an MCS channel reflects an Error status that may be caused by one of the following conditions: • MCS channel sets the ERR bit by software (e.g. with instruction ORL STA, 0x4) • ECC RAM Error occurred while accessing the connected RAM (also disables the MCS channel by clearing bit **EN** and the first error occurred updates bit field **ERR_SRC_ID** of register **MCS[i]_CTRL_STAT**) • Decoding an instruction with an invalid opcode (also disables the MCS channel by clearing bit **EN** and the first error occurred updates bit field **ERR_SRC_ID** of register **MCS[i]_CTRL_STAT**) • A memory address range overflow occurred (also disables the MCS channel by clearing bit **EN** and the first error occurred updates bit field **ERR_SRC_ID** of register **MCS[i]_CTRL_STAT**) • Division by zero resulting from a **DIVU**, **DIVS** or **MODU** instruction (also disables the MCS channel by clearing bit **EN** and the first error occurred updates bit field **ERR_SRC_ID** of register **MCS[i]_CTRL_STAT**). • MCS channel wants to write to a GPR that is write protected by register **MCS[i]_REG_PROT** (also disables the MCS channel by clearing bit **EN** and the first error occurred updates bit field **ERR_SRC_ID** of register **MCS[i]_CTRL_STAT**) • MCS channel wants to write to a protected memory range defined by an address range protector (ARP) of the sub module CCM (also disables the MCS channel by clearing bit **EN** and the first error occurred updates bit field **ERR_SRC_ID** of register **MCS[i]_CTRL_STAT**) • MCS channel performs an invalid AEI bus master access while the bit field **HLT_AEIM_ERR** of register **MCS[i]_CTRL_STAT** is set (also disables the MCS channel by clearing bit **EN** and the first error occurred updates bit field **ERR_SRC_ID** of register **MCS[i]_CTRL_STAT**) Note: If the ERR bit is set due to a memory address range overflow any read or write access to the RAM is blocked. Note: If the GTM includes a MON sub module, the ERR signal is always captured by this module. Note: An MCS channel can set the error bit by writing value 1 to bit ERR. Writing a value 0 to this bit does not cancel the error signal, and thus has no effect. In Addition, writing a value 1 to ERR always triggers the ERR interrupt, independently from the current state of the error signal. Note: The ERR bit can only be cleared by CPU, by writing a 1 to the **MCS[i]_ERR** register (see **Register MCS[i]_ERR**). Note: An MCS channel can read the ERR bit in order to determine the current state of the error signal. The MCS channel reads a value 1 if an ERR occurred previously, but not cleared by CPU. If an MCS channel reads a value 0 no error was set or it has been cleared by CPU.

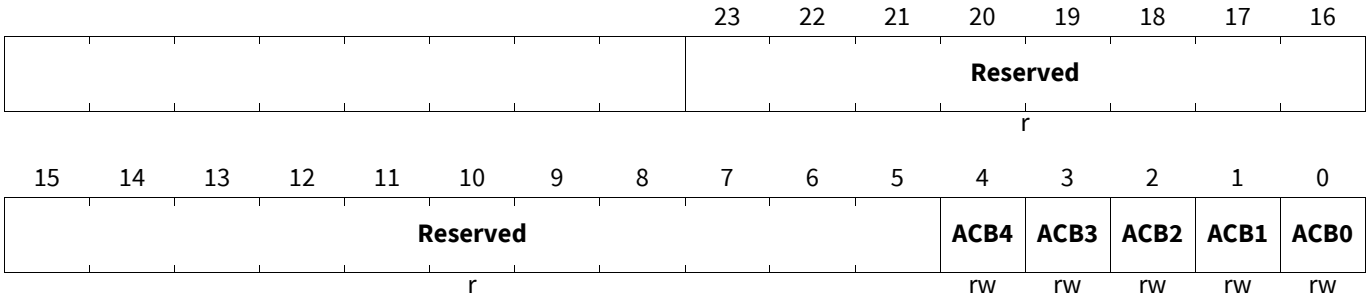
Field	Bits	Type	Description
MCA	3	rw	**MON Activity signaling for MCS channel** 0_B No activity signaled to sub module MON. 1_B Activity singled to sub module MON. *Note: When this bit is set the corresponding channel in the MON sub module register* **MON_ACTIVITY** *is set (see register MON_ACTIVITY_0. This bit is automatically cleared after writing it by the MCS channel program.*
CY	4	r	**Carry bit** The carry bit is updated by several arithmetic and logic instructions. In arithmetic operations, the carry bit indicates an unsigned under/overflow.
Z	5	r	**Zero bit** The zero bit is updated by several arithmetic, logic and data transfer instructions to indicate a result of zero.
V	6	r	**Overflow bit** The overflow bit is updated by arithmetic instructions in order to indicate a signed under/overflow.
N	7	r	**Negative bit** The negative bit is updated by arithmetic instructions in order to indicate a negative result.
CAT	8	r	**Cancel ARU transfer bit** 0_B No cancellation request for ARU transfer. 1_B CPU requests cancellation for ARU transfer. Note: This bit is always cleared at the beginning of the execution of a blocking ARU instruction.
CWT	9	r	**Cancel WURM instruction bit** 0_B Last WURM instruction was not canceled 1_B CPU canceled last WURM instruction of channel. Note: This bit is updated after each WURM instruction and it should be evaluated immediately after the WURM instruction. Otherwise, the CPU could set the bit leading to a bad status information in the MCS program.
SAT	10	r	**Successful ARU transfer bit** 0_B ARU data transfer failed. 1_B ARU data transfer finished successfully. Note: This bit is always updated after the execution of ARU instructions in order to show if the ARU data transfer was successful or not. In the case of non-blocking ARU instructions (NARD, NARDI) a cleared SAT flag signalizes that the data source has no data available and in the case of blocking ARU instructions (ARD, ARDI, AWR, AWRI) a cleared SAT flag signalizes that the data transfer was canceled by the CPU.
Reserved	15:11	r	**Reserved** Read as zero, shall be written as zero.
SP_CNT	18:16	r	**Stack pointer counter value** Note: Actual stack depth of channel. The bit field is incremented on behalf of a CALL or PUSH instruction and decremented on behalf of a RET or POP instruction. The MCS channel STK_ERR_IRQ is raised, when an overflow or underflow is detected on this bit field.

Field	Bits	Type	Description
Reserved	23:19	r	**Reserved** Read as zero, shall be written as zero.

Note: Writing to bits of the register STA with instructions that do implicitly a read-modify-write operation (e.g. "ANDL STA 0xFFFFFE" or "OR STA R0") is dangerous, since writing back the possibly modified content of the read access (which reflects status information) may cause undesirable results. A secure way for writing to bits of the register STA is to use instructions that do not read the content of STA (e.g. "MOVL STA 0x0, MOV STA R1, CLRB STA R0, or SETB STA R1").

28.17.9.4 Register ACB

ACB
ARU Control Bit Register **(09_H)** **Reset Value: 000000_H**



Field	Bits	Type	Description
ACB0	0	rw	**ARU Control bit 0.** Note: This bit is updated by each ARU read access and its value is sent to ARU by each ARU write access on bit 48 of the ARU word.
ACB1	1	rw	**ARU Control bit 1.** Note: This bit is updated by each ARU read access and its value is sent to ARU by each ARU write access on bit 49 of the ARU word.
ACB2	2	rw	**ARU Control bit 2.** Note: This bit is updated by each ARUread access and its value is sent to ARU by each ARU write access on bit 50 of the ARU word.
ACB3	3	rw	**ARU Control bit 3.** Note: This bit is updated by each ARU read access and its value is sent to ARU by each ARU write access on bit 51 of the ARU word.
ACB4	4	rw	**ARU Control bit 4.** Note: This bit is updated by each ARU read access and its value is sent to ARU by each ARU write access on bit 52 of the ARU word.
Reserved	23:5	r	**Read as zero, shall be written as zero.**

28.17.9.5 Register CTRG

CTRG
Clear Trigger Bits Register **($0A_H$)** **Reset Value: 000000_H**

23	22	21	20	19	18	17	16
TRG23	**TRG22**	**TRG21**	**TRG20**	**TRG19**	**TRG18**	**TRG17**	**TRG16**
rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRG15	**TRG14**	**TRG13**	**TRG12**	**TRG11**	**TRG10**	**TRG9**	**TRG8**	**TRG7**	**TRG6**	**TRG5**	**TRG4**	**TRG3**	**TRG2**	**TRG1**	**TRG0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
TRG0	0	rw	**Trigger bit 0** READ access: state of current trigger bit **TRG0 if EN_TIM_FOUT** = 0 state of input signal **TIM[i]_CH0_F_OUT if EN_TIM_FOUT** = 1 WRITE access: 0_B do nothing 1_B clear trigger bit **TRG0**
TRG1	1	rw	**Trigger bit 1** READ access: state of current trigger bit **TRG1** if **EN_TIM_FOUT** = 0 state of input signal **TIM[i]_CH1_F_OUT if EN_TIM_FOUT** = 1 WRITE access: 0_B do nothing 1_B clear trigger bit **TRG1**
TRG2	2	rw	**Trigger bit 2** READ access: state of current trigger bit **TRG2** if **EN_TIM_FOUT** = 0 state of input signal **TIM[i]_CH2_F_OUT if EN_TIM_FOUT** = 1 WRITE access: 0_B do nothing 1_B clear trigger bit **TRG2**
TRG3	3	rw	**Trigger bit 3** READ access: state of current trigger bit **TRG3** if **EN_TIM_FOUT** = 0 state of input signal **TIM[i]_CH3_F_OUT if EN_TIM_FOUT** = 1 WRITE access: 0_B do nothing 1_B clear trigger bit **TRG3**

Field	Bits	Type	Description
TRG4	4	rw	**Trigger bit 4** READ access: state of current trigger bit **TRG4** if **EN_TIM_FOUT** = 0 state of input signal **TIM[i]_CH4_F_OUT if EN_TIM_FOUT** = 1 WRITE access: 0_B do nothing 1_B clear trigger bit **TRG4**
TRG5	5	rw	**Trigger bit 5** READ access: state of current trigger bit **TRG5** if **EN_TIM_FOUT** = 0 state of input signal **TIM[i]_CH5_F_OUT if EN_TIM_FOUT** = 1 WRITE access: 0_B do nothing 1_B clear trigger bit **TRG5**
TRG6	6	rw	**Trigger bit 6** READ access: state of current trigger bit **TRG6** if **EN_TIM_FOUT** = 0 state of input signal **TIM[i]_CH6_F_OUT if EN_TIM_FOUT** = 1 WRITE access: 0_B do nothing 1_B clear trigger bit **TRG6**
TRG7	7	rw	**Trigger bit 7** READ access: state of current trigger bit **TRG7** if **EN_TIM_FOUT** = 0 state of input signal **TIM[i]_CH7_F_OUT if EN_TIM_FOUT** = 1 WRITE access: 0_B do nothing 1_B clear trigger bit **TRG7**
TRG8	8	rw	**Trigger bit 8** READ access: state of current trigger bit **TRG8** if **EN_TIM_FOUT** = 0 state of input signal **TIM[i+1]_CH0_F_OUT if EN_TIM_FOUT** = 1 WRITE access: 0_B do nothing 1_B clear trigger bit **TRG8**
TRG9	9	rw	**Trigger bit 9** READ access: state of current trigger bit **TRG9** if **EN_TIM_FOUT** = 0 state of input signal **TIM[i+1]_CH1_F_OUT if EN_TIM_FOUT** = 1 WRITE access: 0_B do nothing 1_B clear trigger bit **TRG9**

Field	Bits	Type	Description
TRG10	10	rw	**Trigger bit 10** READ access: state of current trigger bit **TRG10** if **EN_TIM_FOUT** = 0 state of input signal **TIM[i+1]_CH2_F_OUT if EN_TIM_FOUT** = 1 WRITE access: 0_B do nothing 1_B clear trigger bit **TRG10**
TRG11	11	rw	**Trigger bit 11** READ access: state of current trigger bit **TRG11** if **EN_TIM_FOUT** = 0 state of input signal **TIM[i+1]_CH3_F_OUT if EN_TIM_FOUT** = 1 WRITE access: 0_B do nothing 1_B clear trigger bit **TRG11**
TRG12	12	rw	**Trigger bit 12** READ access: state of current trigger bit **TRG12** if **EN_TIM_FOUT** = 0 state of input signal **TIM[i+1]_CH4_F_OUT if EN_TIM_FOUT** = 1 WRITE access: 0_B do nothing 1_B clear trigger bit **TRG12**
TRG13	13	rw	**Trigger bit 13** READ access: state of current trigger bit **TRG13** if **EN_TIM_FOUT** = 0 state of input signal **TIM[i+1]_CH5_F_OUT if EN_TIM_FOUT** = 1 WRITE access: 0_B do nothing 1_B clear trigger bit **TRG13**
TRG14	14	rw	**Trigger bit 14** READ access: state of current trigger bit **TRG14** if **EN_TIM_FOUT** = 0 state of input signal **TIM[i+1]_CH6_F_OUT if EN_TIM_FOUT** = 1 WRITE access: 0_B do nothing 1_B clear trigger bit **TRG14**
TRG15	15	rw	**Trigger bit 15** READ access: state of current trigger bit **TRG15** if **EN_TIM_FOUT** = 0 state of input signal **TIM[i+1]_CH7_F_OUT if EN_TIM_FOUT** = 1 WRITE access: 0_B do nothing 1_B clear trigger bit **TRG15**
TRG16	16	rw	**Trigger bit 16** 0_B READ: trigger bit is cleared / WRITE: do nothing 1_B READ: trigger bit is set / WRITE: clear trigger bit

Field	Bits	Type	Description
TRG17	17	rw	**Trigger bit 17** 0_B READ: trigger bit is cleared / WRITE: do nothing 1_B READ: trigger bit is set / WRITE: clear trigger bit
TRG18	18	rw	**Trigger bit 18** 0_B READ: trigger bit is cleared / WRITE: do nothing 1_B READ: trigger bit is set / WRITE: clear trigger bit
TRG19	19	rw	**Trigger bit 19** 0_B READ: trigger bit is cleared / WRITE: do nothing 1_B READ: trigger bit is set / WRITE: clear trigger bit
TRG20	20	rw	**Trigger bit 20** 0_B READ: trigger bit is cleared / WRITE: do nothing 1_B READ: trigger bit is set / WRITE: clear trigger bit
TRG21	21	rw	**Trigger bit 21** 0_B READ: trigger bit is cleared / WRITE: do nothing 1_B READ: trigger bit is set / WRITE: clear trigger bit
TRG22	22	rw	**Trigger bit 22** 0_B READ: trigger bit is cleared / WRITE: do nothing 1_B READ: trigger bit is set / WRITE: clear trigger bit
TRG23	23	rw	**Trigger bit 23** 0_B READ: trigger bit is cleared / WRITE: do nothing 1_B READ: trigger bit is set / WRITE: clear trigger bit

*Note: The trigger bits **TRGx** are accessible by all MCS channels as well as the CPU. Setting a trigger bit can be performed with the **STRG** register, in the case of an MCS channel or the **MCS[i]_STRG** register in the case of the CPU. Clearing a trigger bit can be performed with the **CTRG** register, in the case of an MCS channel or the **MCS[i]_CTRG** register in the case of the CPU. Trigger bits can be used for signalizing specific events to MCS channels or the CPU. An MCS channel suspended with a WURM instruction can be resumed by setting the appropriate trigger bit.*

*Note: Besides setting the trigger bits with register **STRG/MCS[i]_STRG**, the k-th trigger bit **TRGk** (with k < 16) can also be set by the external capture event that is enabled by the k-th bit of register **CCM[i]_EXT_CAP_EN**. If bit k bit is disabled, the k-th trigger bit TRGk can only be set by MCS or CPU.*

*Note: The result of a read access to this register differs in dependency of the bit field **EN_TIM_FOUT** of register **MCS[i]_CTRL_STAT**.*

28.17.9.6 Register STRG

STRG
Set Trigger Bits Register **($0B_H$)** **Reset Value: 000000_H**

23	22	21	20	19	18	17	16
TRG23	**TRG22**	**TRG21**	**TRG20**	**TRG19**	**TRG18**	**TRG17**	**TRG16**
rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRG15	**TRG14**	**TRG13**	**TRG12**	**TRG11**	**TRG10**	**TRG9**	**TRG8**	**TRG7**	**TRG6**	**TRG5**	**TRG4**	**TRG3**	**TRG2**	**TRG1**	**TRG0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
TRGx (x=0-23)	x	rw)	**Trigger bit x** 0_B READ: trigger bit is cleared / WRITE: do nothing 1_B READ: trigger bit is set / WRITE: set trigger bit

*Note: The trigger bits **TRGx** are accessible by all MCS channels as well as the CPU. Setting a trigger bit can be performed with the **STRG** register, in the case of an MCS channel or the **MCS[i]_STRG** register in the case of the CPU. Clearing a trigger bit can be performed with the **CTRG** register, in the case of an MCS channel or the **MCS[i]_CTRG** register in the case of the CPU. Trigger bits can be used for signalizing specific events to MCS channels or the CPU. An MCS channel suspended with a WURM instruction can be resumed by setting the appropriate trigger bit.*

*Note: Besides setting the trigger bits with register **STRG/MCS[i]_STRG**, the k-th trigger bit **TRGx** (with k < 16) can also be set by the external capture event that is enabled by the k-th bit of register **CCM[i]_EXT_CAP_EN**. If bit k bit is disabled, the k-th trigger bit **TRGx** can only be set by MCS or CPU.*

28.17.9.7 Register TBU_TS0

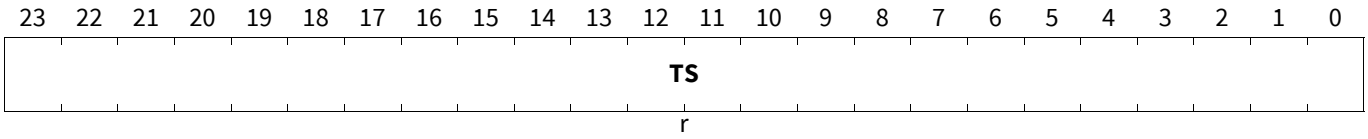
TBU_TS0
TBU Timestamp TS0 Register **($0C_H$)** **Reset Value: 000000_H**

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TS																							
r																							

Field	Bits	Type	Description
TS	23:0	r	**Current TBU time stamp 0.**

28.17.9.8 Register TBU_TS1

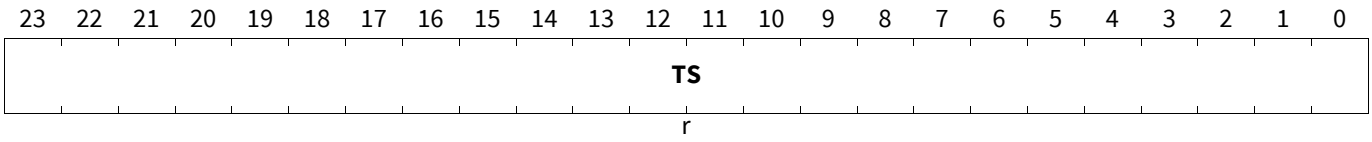
TBU_TS1
TBU Timestamp TS1 Register **($0D_H$)** **Reset Value: 000000_H**



Field	Bits	Type	Description
TS	23:0	r	**Current TBU time stamp 1.**

28.17.9.9 Register TBU_TS2

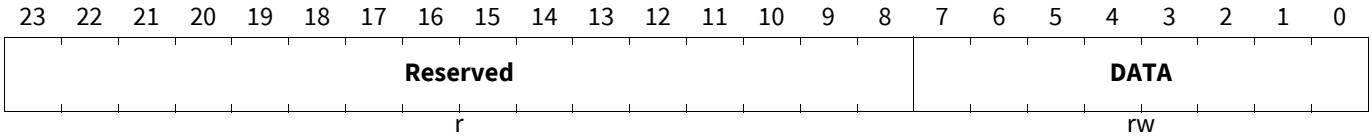
TBU_TS2
TBU Timestamp TS2 Register **($0E_H$)** **Reset Value: 000000_H**



Field	Bits	Type	Description
TS	23:0	r	**Current TBU time stamp 2**

28.17.9.10Register MHB

MHB
Memory High Byte Register **($0F_H$)** **Reset Value: 000000_H**



Field	Bits	Type	Description
DATA	7:0	rw	**High Byte of a memory transfer**
Reserved	23:8	r	**Reserved** Read as zero, shall be written as zero.

28.17.9.11Register GMI0

GMI0
GTM Module Interrupt 0 Register **(18_H)** **Reset Value: 000000_H**

23	22	21	20	19	18	17	16
ATOM_CH7_IRQ	ATOM_CH6_IRQ	ATOM_CH5_IRQ	ATOM_CH4_IRQ	ATOM_CH3_IRQ	ATOM_CH2_IRQ	ATOM_CH1_IRQ	ATOM_CH0_IRQ
rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TOM_CH14_IRQ	TOM_CH12_IRQ	TOM_CH10_IRQ	TOM_CH8_IRQ	TOM_CH6_IRQ	TOM_CH4_IRQ	TOM_CH2_IRQ	TOM_CH0_IRQ	TIM_CH7_IRQ	TIM_CH6_IRQ	TIM_CH5_IRQ	TIM_CH4_IRQ	TIM_CH3_IRQ	TIM_CH2_IRQ	TIM_CH1_IRQ	TIM_CH0_IRQ
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
TIM_CHx_IRQ (x=0-7)	x	rw	**TIM[i]_CHx IRQ** READ access: IRQ signal *TIM[i]_CHx_IRQ.* WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
TOM_CH0_IRQ	8	rw	**TOM[i]_CH0 or TOM[i]_CH1 IRQ** READ access: Logical OR conjunction of IRQ signals *T0M[i]_CH0_IRQ* and *T0M[i]_CH1_IRQ*. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
TOM_CH2_IRQ	9	rw	**TOM[i]_CH2 or TOM[i]_CH3 IRQ** READ access: Logical OR conjunction of IRQ signals *T0M[i]_CH2_IRQ* and *T0M[i]_CH3_IRQ*. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
TOM_CH4_IRQ	10	rw	**TOM[i]_CH4 or TOM[i]_CH5 IRQ** READ access: Logical OR conjunction of IRQ signals *T0M[i]_CH4_IRQ* and *T0M[i]_CH5_IRQ*. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
TOM_CH6_IRQ	11	rw	**TOM[i]_CH6 or TOM[i]_CH7 IRQ** READ access: Logical OR conjunction of IRQ signals *T0M[i]_CH6_IRQ* and *T0M[i]_CH7_IRQ*. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.

Field	Bits	Type	Description
TOM_CH8_IRQ	12	rw	**TOM[i]_CH8 or TOM[i]_CH9 IRQ** READ access: Logical OR conjunction of IRQ signals *T0M[i]_CH8_IRQ* and *T0M[i]_CH9_IRQ*. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
TOM_CH10_IRQ	13	rw	**TOM[i]_CH10 or TOM[i]_CH11 IRQ** READ access: Logical OR conjunction of IRQ signals *T0M[i]_CH10_IRQ* and *T0M[i]_CH11_IRQ*. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
TOM_CH12_IRQ	14	rw	**TOM[i]_CH12 or TOM[i]_CH13 IRQ** READ access: Logical OR conjunction of IRQ signals *T0M[i]_CH12_IRQ* and *T0M[i]_CH13_IRQ*. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
TOM_CH14_IRQ	15	rw	**TOM[i]_CH14 or TOM[i]_CH15 IRQ** READ access: Logical OR conjunction of IRQ signals *T0M[i]_CH14_IRQ* and *T0M[i]_CH15_IRQ*. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
ATOM_CHx_IRQ (x=0-7)	16 + x	rw	**ATOM[i]_CHx IRQ** READ access: IRQ signal ATOM[i]_CHx_IRQ. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.

28.17.9.12Register GMI1

GMI1

GTM Module Interrupt 1 Register **(19_H)** **Reset Value: 000000_H**

23	22	21	20	19	18	17	16
MCS0_CH7_IRQ	**MCS0_CH6_IRQ**	**MCS0_CH5_IRQ**	**MCS0_CH4_IRQ**	**MCS0_CH3_IRQ**	**MCS0_CH2_IRQ**	**MCS0_CH1_IRQ**	**MCS0_CH0_IRQ**
rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TTA_IP1_CH7_IRQ	**TTA_IP1_CH6_IRQ**	**TTA_IP1_CH5_IRQ**	**TTA_IP1_CH4_IRQ**	**TTA_IP1_CH3_IRQ**	**TTA_IP1_CH2_IRQ**	**TTA_IP1_CH1_IRQ**	**TTA_IP1_CH0_IRQ**	**MCS_IP1_CH7_IRQ**	**MCS_IP1_CH6_IRQ**	**MCS_IP1_CH5_IRQ**	**MCS_IP1_CH4_IRQ**	**MCS_IP1_CH3_IRQ**	**MCS_IP1_CH2_IRQ**	**MCS_IP1_CH1_IRQ**	**MCS_IP1_CH0_IRQ**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
MCS_IP1_CHx_IRQ (x=0-7)	x	rw	**MCS[i+1]_CHx IRQ** READ access: IRQ signal *MCS[i+1]_CH0_IRQ* WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
TTA_IP1_CH0_IRQ	8	rw	**Neighboring TIM, TOM, ATOM IRQs** READ access: Logical OR conjunction of IRQ signals *TIM[i+1]_CH0_IRQ*, *TOM[i+1]_CH0_IRQ*, *TOM[i+1]_CH1_IRQ*, and *ATOM[i+1]_CH0_IRQ* WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
TTA_IP1_CH1_IRQ	9	rw	**Neighboring TIM, TOM, ATOM IRQs** READ access: Logical OR conjunction of IRQ signals *TIM[i+1]_CH1_IRQ*, *TOM[i+1]_CH2_IRQ*, *TOM[i+1]_CH3_IRQ*, and *ATOM[i+1]_CH1_IRQ* WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
TTA_IP1_CH2_IRQ	10	rw	**Neighboring TIM, TOM, ATOM IRQs** READ access: Logical OR conjunction of IRQ signals *TIM[i+1]_CH2_IRQ*, *TOM[i+1]_CH4_IRQ*, *TOM[i+1]_CH5_IRQ*, and *ATOM[i+1]_CH2_IRQ* WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
TTA_IP1_CH3_IRQ	11	rw	**Neighboring TIM, TOM, ATOM IRQs** READ access: Logical OR conjunction of IRQ signals *TIM[i+1]_CH3_IRQ*, *TOM[i+1]_CH6_IRQ*, *TOM[i+1]_CH7_IRQ*, and *ATOM[i+1]_CH3_IRQ* WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
TTA_IP1_CH4_IRQ	12	rw	**Neighboring TIM, TOM, ATOM IRQs** READ access: Logical OR conjunction of IRQ signals *TIM[i+1]_CH4_IRQ*, *TOM[i+1]_CH8_IRQ*, *TOM[i+1]_CH9_IRQ*, and *ATOM[i+1]_CH4_IRQ* WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
TTA_IP1_CH5_IRQ	13	rw	**Neighboring TIM, TOM, ATOM IRQs** READ access: Logical OR conjunction of IRQ signals *TIM[i+1]_CH5_IRQ*, *TOM[i+1]_CH10_IRQ*, *TOM[i+1]_CH10_IRQ*, and *ATOM[i+1]_CH5_IRQ* WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.

Field	Bits	Type	Description
TTA_IP1_CH6_IRQ	14	rw	**Neighboring TIM, TOM, ATOM IRQs** READ access: Logical OR conjunction of IRQ signals *TIM[i+1]_CH6_IRQ*, *TOM[i+1]_CH12_IRQ*, *TOM[i+1]_CH13_IRQ*, and *ATOM[i+1]_CH6_IRQ* WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
TTA_IP1_CH7_IRQ	15	rw	**Neighboring TIM, TOM, ATOM IRQs** READ access: Logical OR conjunction of IRQ signals *TIM[i+1]_CH7_IRQ*, *TOM[i+1]_CH14_IRQ*, *TOM[i+1]_CH15_IRQ*, and *ATOM[i+1]_CH7_IRQ* WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
MCS0_CHx_IRQ (x=0-7)	16 + x	rw	**MCS0_CHx IRQs.** READ access: IRQ signal *MCS0_CHx_IRQ* .WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.

28.17.9.13Register DSTA

DSTA
DPLL Status Register **($1A_H$)** **Reset Value: 000000_H**

23	22	21	20	19	18	17	16
Reserved			CDTI	SISI	SASI	TISI	TASI
r			rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STA_S								STA_T							
r								r							

Field	Bits	Type	Description
STA_T	7:0	r	**Status Trigger FSM** Actual status of DPLL Trigger FSM. The description of the FSM states can be found in section “DPLL_STA”.
STA_S	15:8	r	**Status State FSM** Actual status of DPLL State FSM. The description of the FSM states can be found in section “DPLL_STA”.
TASI	16	rw	**Trigger Active Slope Interrupt.** READ access: IRQ signal TASI of DPLL. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.

Field	Bits	Type	Description
TISI	17	rw	**Trigger Inactive Slope Interrupt.** READ access: IRQ signal TISI of DPLL. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
SASI	18	rw	**State Active Slope Interrupt.** READ access: IRQ signal SASI of DPLL. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
SISI	19	rw	**State Inactive Slope Interrupt.** READ access: IRQ signal SISI of DPLL. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
CDTI	20	rw	**Calculation of trigger duration interrupt.** READ access: IRQ signal CDTI of DPLL. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
Reserved	23:21	r	**Reserved** Read as zero, shall be written as zero.

Note: This register is only implemented in MCS instance 0. In other MCS instances, a read access always returns 0 and a write access is always ignored.

28.17.9.14Register DSTAX

DSTAX
DPLL Extended Status Register **($1B_H$)** **Reset Value: 000000_H**

23	22	21	20	19	18	17	16
Reserved			**CDTI**	**SISI**	**SASI**	**TISI**	**TASI**
r			rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												**INC_CNT2_FLAG**	**INC_CNT1_FLAG**	**STA_FLAG_S**	**STA_FLAG_T**
r												rw	rw	rw	rw

Field	Bits	Type	Description
STA_FLAG_T	0	rw	**DPLL status trigger flag** DPLL status trigger flag as described in bit field definition **STA_FLAG_T** of register **DPLL_STA_FLAG** (section ‘Register DPLL_STA_FLAG’).

Field	Bits	Type	Description
STA_FLAG_S	1	rw	**DPLL status state flag** DPLL status state flag as described in bit field definition **STA_FLAG_S** of register **DPLL_STA_FLAG** (section 'Register DPLL_STA_FLAG').
INC_CNT1_FLAG	2	rw	**DPLL INC_CNT1 Flag** DPLL status state flag as described in bit field definition **INC_CNT1_FLAG** of register **DPLL_STA_FLAG** (section 'Register DPLL_STA_FLAG').
INC_CNT2_FLAG	3	rw	**DPLL INC_CNT2 Flag** DPLL status state flag as described in bit field definition **INC_CNT2_FLAG** of register **DPLL_STA_FLAG** (section 'Register DPLL_STA_FLAG').
Reserved	15:4	r	**Reserved** Read as zero, shall be written as zero.
TASI	16	rw	**Trigger Active Slope Interrupt.** READ access: IRQ signal TASI of DPLL. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
TISI	17	rw	**Trigger Inactive Slope Interrupt.** READ access: IRQ signal TISI of DPLL. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
SASI	18	rw	**State Active Slope Interrupt.** READ access: IRQ signal SASI of DPLL. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
SISI	19	rw	**State Inactive Slope Interrupt.** READ access: IRQ signal SISI of DPLL. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
CDTI	20	rw	**Calculation of trigger duration interrupt.** READ access: IRQ signal CDTI of DPLL. WRITE access: 0_B do nothing 1_B issue hw_clear on the connected IRQs.
Reserved	23:21	r	**Reserved** Read as zero, shall be written as zero.

Note: ***Note**: This register is only implemented in MCS instance 0. In other MCS instances, a read access always returns 0 and a write access is always ignored.*

28.17.10 MCS Configuration Register Overview

The MCS Configuration registers of the MCS module are accessible by the AEI bus interface. Some of these registers simply mirror MCS Internal registers to the AEI. Details can be found in the table below and in the individual register descriptions.

Table 66 MCS Configuration Register Overview$_B$

Register Name	Description	see Page
MCS[i]_CH[x]_CTRL	MCSi channel x control register	409
MCS[i]_CH[x]_ACB	MCSi channel x ARU control Bit register	411
MCS[i]_CH[x]_MHB	MCSi channel x memory high byte register	412
MCS[i]_CH[x]_PC	MCSi channel x program counter register	410
MCS[i]_CH[x]_R[y] (y=0..7)	MCSi channel x general purpose register y (y=0..7)	411
MCS[i]_CH[x]_IRQ_NOTIFY	MCSi channel x interrupt notification register	413
MCS[i]_CH[x]_IRQ_EN	MCSi channel x interrupt enable register	414
MCS[i]_CH[x]_IRQ_FORCINT	MCSi channel x force interrupt register	414
MCS[i]_CH[x]_IRQ_MODE	MCSi channel x IRQ mode configuration register	415
MCS[i]_CH[x]_EIRQ_EN	MCSi channel x error interrupt enable register	416
MCS[i]_CTRL_STAT	MCSi control and status register	417
MCS[i]_REG_PROT	MCSi write protection register	419
MCS[i]_CTRG	MCSi clear trigger control register	420
MCS[i]_STRG	MCSi set trigger control register	421
MCS[i]_RESET	MCSi reset register	422
MCS[i]_ERR	MCSi error register	425
MCS[i]_CAT	MCSi cancel ARU transfer instruction	423
MCS[i]_CWT	MCSi cancel WURM instruction	424

28.17.11 MCS Configuration Register Description

28.17.11.1Register MCS[i]_CH[x]_CTRL

MCSi Channel x Control Register

MCSi_CHx_CTRL (i=0-9;x=0-7)
MCSi Channel x Control Register **($0F0020_H$+i*1000_H+x*80_H)** **Application Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0													SP_CNT		
r													rh		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					SAT	CWT	CAT	N	V	Z	CY	0	ERR	IRQ	EN
r					rh	rh	rh	rh	rh	rh	rh	r	rh	rh	rwh

Field	Bits	Type	Description
EN	0	rwh	**Enable MCS-channel** *Note: Enabling or disabling of an MCS-channel is synchronized to the ending of an instruction, and thus it may take several clock cycles, e.g. active memory transfers or pending WURM transfers have to be finished before disabling the MCS-channel. The internal state of a channel can be obtained by reading bit EN.* *Note: To disable an MCS channel reliably, the EN bit should be cleared, followed by setting the CAT and CWT bits in order to cancel any pending WURM or ARU instructions.* *Note: The EN bit is write protected during RAM reset phase.* 0_B Disable current MCS-channel 1_B Enable current MCS-channel
IRQ	1	rh	**Interrupt state** This bit is read only, and it mirrors the internal IRQ state. 0_B No interrupt pending in MCS-channel x 1_B Interrupt is pending in MCS-channel x
ERR	2	rh	**Error state** This bit is read only, and it mirrors the internal error state. 0_B No error signal pending in MCS-channel x 1_B Error signal is pending in MCS-channel x
CY	4	rh	**Carry bit state** This bit is read only and it mirrors the internal carry flag CY.
Z	5	rh	**Zero bit state** This bit is read only and it mirrors the internal zero flag Z.
V	6	rh	**Overflow bit state** This bit is read only and it mirrors the internal carry flag V.
N	7	rh	**Negative bit state** This bit is read only and it mirrors the internal zero flag N.

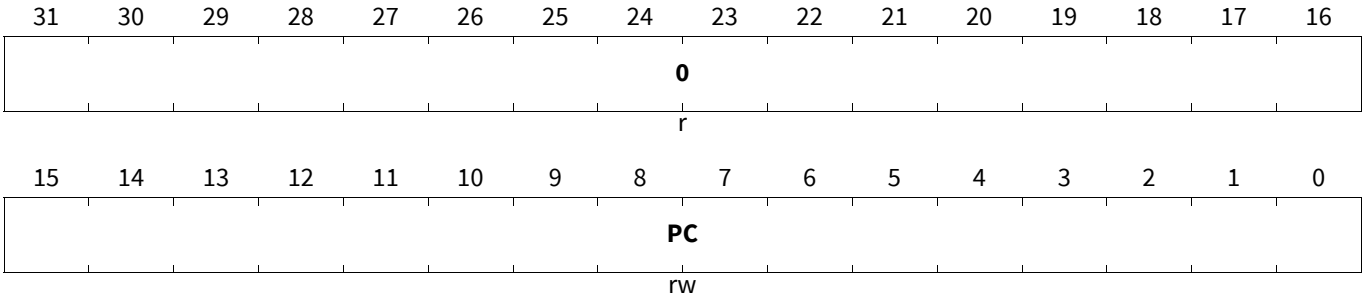
Field	Bits	Type	Description
CAT	8	rh	**Cancel ARU transfer state** This bit is read only and it mirrors the internal cancel ARU transfer status flag CAT.
CWT	9	rh	**Cancel WURM instruction state** This bit is read only and it mirrors the internal cancel WURM instruction status flag CWT.
SAT	10	rh	**Successful ARU transfer bit** This bit is read only, and it mirrors the internal state of the ARU transfer status flag SAT. 0_B ARU data transfer failed 1_B ARU data transfer finished successfully
SP_CNT	18:16	rh	**Stack pointer counter value** Actual stack depth of channel. The bit field is incremented on behalf of a CALL or PUSH instruction and decremented on behalf of a RET or POP instruction. The MCS channel STK_ERR_IRQ is raised, when an overflow or underflow is detected on this bit field.
0	3, 15:11, 31:19	r	**Reserved** Read as zero, shall be written as zero.

28.17.11.2Register MCS[i]_CH[x]_PC

MCSi Channel x Program Counter Register

MCSi_CHx_PC (i=0-9;x=0-7)

MCSi Channel x Program Counter Register($0F0040_H$+i*1000_H+x*80_H) **Reset Value: Table 67**



Field	Bits	Type	Description
PC	15:0	rw	**Current Program Counter** *Note: The program counter is only writable if the corresponding MCS-channel is disabled. The bits 0 and 1 are always written as zeros.* *Note: The actual width of the program counter depends on the MCS configuration. The actual width is RAW+USR+2 bits meaning that only the bits 0 to RAW+USR+1 are available and the other bits (RAW+USR+2 to 31) are reserved.*
0	31:16	r	**Reserved** Read as zero, shall be written as zero.

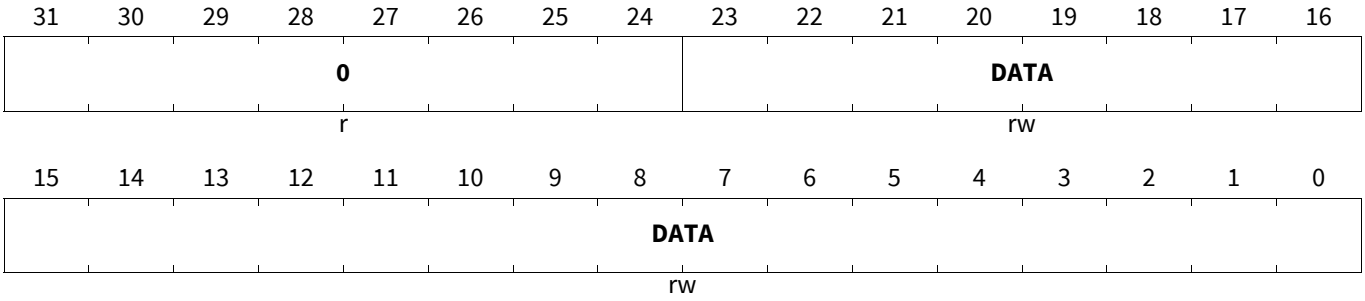
Table 67 Reset Values of MCSi_CHx_PC (i=0-9;x=0-7)

Reset Type	Reset Value	Note
Application Reset	4 * x	

28.17.11.3Register MCS[i]_CH[x]_R[y]

MCSi Channel x General Purpose Register y

MCSi_CHx_Ry (i=0-9;x=0-7;y=0-7)
MCSi Channel x General Purpose Register y($0F0000_H$+i*1000_H+x*80_H+y*4) Application Reset Value: $0000\ 0000_H$

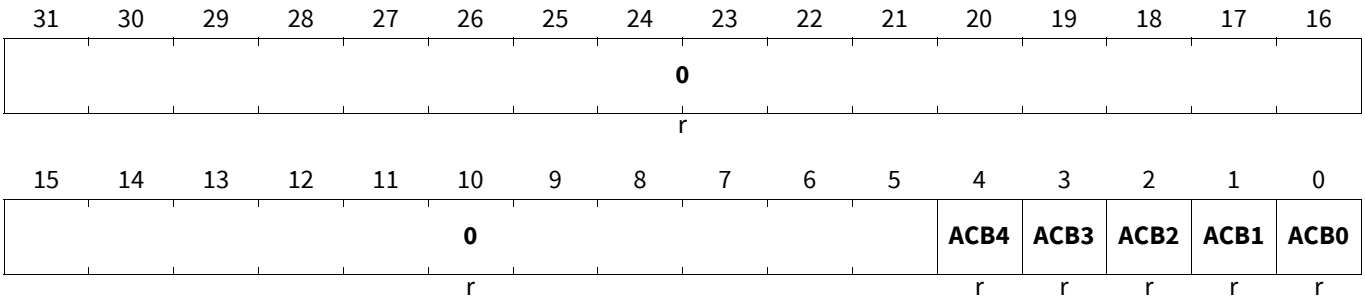


Field	Bits	Type	Description
DATA	23:0	rw	**Data of general purpose register R[y]** *Note: This register is the same as described in internal register section.* *Note: For the register MCS[i]_CH[x]_R6 (see internal register section) an additional write protection during an active ARDI or NARDI instruction is applied*
0	31:24	r	**Reserved** Read as zero, shall be written as zero.

28.17.11.4Register MCS[i]_CH[x]_ACB

MCSi Channel x ARU Control Bit Register

MCSi_CHx_ACB (i=0-9;x=0-7)
MCSi Channel x ARU Control Bit Register($0F0024_H$+i*1000_H+x*80_H) Application Reset Value: $0000\ 0000_H$



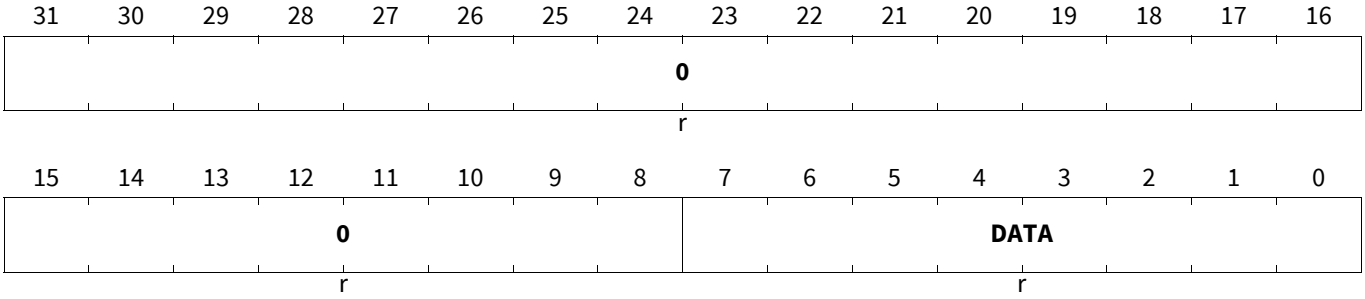
Field	Bits	Type	Description
ACB0	0	r	**ARU Control bit 0** This bit is read only, and it mirrors the internal state.
ACB1	1	r	**ARU Control bit 1** This bit is read only, and it mirrors the internal state.
ACB2	2	r	**ARU Control bit 2** This bit is read only, and it mirrors the internal state.
ACB3	3	r	**ARU Control bit 3** This bit is read only, and it mirrors the internal state.
ACB4	4	r	**ARU Control bit 4** This bit is read only, and it mirrors the internal state.
0	31:5	r	**Reserved** Read as zero, shall be written as zero.

28.17.11.5Register MCS[i]_CH[x]_MHB

MCSi Channel x Memory High Byte Register

MCSi_CHx_MHB (i=0-9;x=0-7)

MCSi Channel x Memory High Byte Register($0F003C_H$+i*1000_H+x*80_H) Application Reset Value: 0000 0000_H

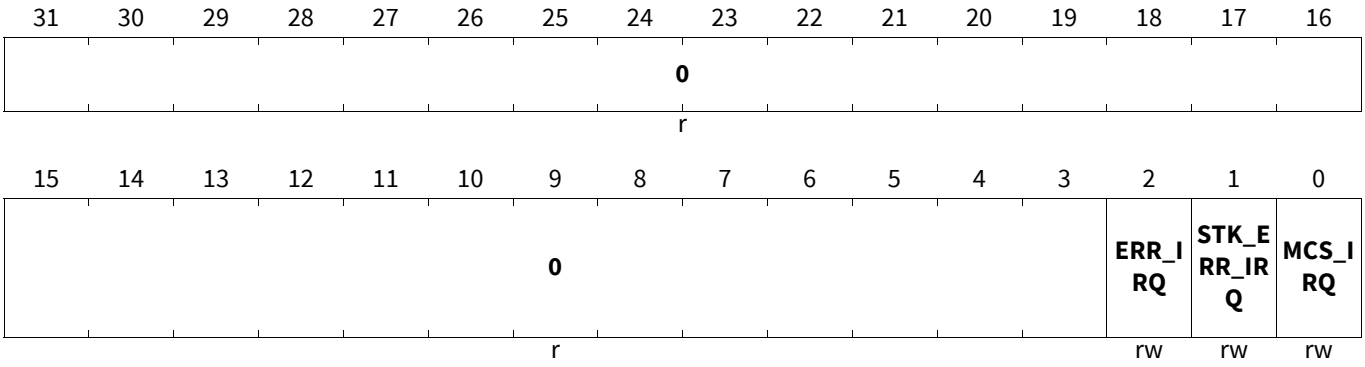


Field	Bits	Type	Description
DATA	7:0	r	**Data of memory high bit register MHB**
0	31:8	r	**Reserved** Read as zero, shall be written as zero.

28.17.11.6Register MCS[i]_CH[x]_IRQ_NOTIFY

MCSi Channel x Interrupt Notification Register

MCSi_CHx_IRQ_NOTIFY (i=0-9;x=0-7)
MCSi Channel x Interrupt Notification Register($0F0044_H$+i*1000_H+x*80_H) Application Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
MCS_IRQ	0	rw	**Interrupt request by MCS-channel x** *Note: This bit will be cleared on a CPU write access with a value of '1'. A read access leaves the bit unchanged.* *Note: By writing a '1' to this register, the IRQ flag in the MCS channel status register STA is cleared.* 0_B No IRQ released 1_B IRQ released by MCS-channel
STK_ERR_IRQ	1	rw	**Stack counter overflow/underflow of channel x** *Note: This bit will be cleared on a CPU write access with a value of '1'. A read access leaves the bit unchanged.* 0_B No IRQ released 1_B A stack counter overflow or underflow occurred
ERR_IRQ	2	rw	**MCS channel x ERR interrupt** *Note: If the ERR bit of register STA is triggered, the ERR_IRQ will also be set.* *Note: This bit will be cleared on a CPU write access with a value of '1'. A read access leaves the bit unchanged.* 0_B No IRQ released 1_B MCS-channel ERR IRQ released
0	31:3	r	**Reserved** Read as zero, shall be written as zero.

28.17.11.7Register MCS[i]_CH[x]_IRQ_EN

MCSi Channel x Interrupt Enable Register

MCSi_CHx_IRQ_EN (i=0-9;x=0-7)

MCSi Channel x Interrupt Enable Register($0F0048_H$+i*1000_H+x*80_H) **Application Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0													ERR_IRQ_EN	STK_ERR_IRQ_EN	MCS_IRQ_EN
r													rw	rw	rw

Field	Bits	Type	Description
MCS_IRQ_EN	0	rw	**MCS channel x MCS_IRQ interrupt enable** 0_B Disable interrupt 1_B Enable interrupt
STK_ERR_IRQ_EN	1	rw	**MCS channel x STK_ERR_IRQ interrupt enable** 0_B Disable interrupt 1_B Enable interrupt
ERR_IRQ_EN	2	rw	**MCS channel x ERR_IRQ interrupt enable** 0_B Disable interrupt 1_B Enable interrupt
0	31:3	r	**Reserved** Read as zero, shall be written as zero.

28.17.11.8Register MCS[i]_CH[x]_IRQ_FORCINT

MCSi Channel x Force Interrupt Register

MCSi_CHx_IRQ_FORCINT (i=0-9;x=0-7)

MCSi Channel x Force Interrupt Register($0F004C_H$+i*1000_H+x*80_H) **Application Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0													TRG_ERR_IRQ	TRG_STK_ERR_IRQ	TRG_MCS_IRQ
r													rw	rw	rw

Field	Bits	Type	Description
TRG_MCS_IRQ	0	rw	**Trigger IRQ bit in MCS_CH_[x]_IRQ_NOTIFY register by software** *Note: This bit is cleared automatically after write.* *Note: This bit is write protected by bit RF_PROT of register GTM_CTRL.* 0_B No interrupt triggering 1_B Assert corresponding bit in MCS[i]_CH[x]_IRQ_NOTIFY register
TRG_STK_ERR_IRQ	1	rw	**Trigger IRQ bit in MCS_CH_[x]_IRQ_NOTIFY register by software** *Note: This bit is cleared automatically after write.* *Note: This bit is write protected by bit RF_PROT of register GTM_CTRL.* 0_B No interrupt triggering 1_B Assert corresponding bit in MCS[i]_CH[x]_IRQ_NOTIFY register
TRG_ERR_IRQ	2	rw	**Trigger IRQ bit in MCS_CH_[x]_IRQ_NOTIFY register by software** *Note: This bit is cleared automatically after write.* *Note: This bit is write protected by bit RF_PROT of register GTM_CTRL.* 0_B No interrupt triggering 1_B Assert corresponding bit in MCS[i]_CH[x]_IRQ_NOTIFY register
0	31:3	r	**Reserved** Read as zero, shall be written as zero.

28.17.11.9Register MCS[i]_CH[x]_IRQ_MODE

MCSi Channel x Interrupt Mode Configuration Register

MCSi_CHx_IRQ_MODE (i=0-9;x=0-7)
MCSi Channel x Interrupt Mode Configuration Register($0F0050_H$+i*1000_H+x*80_H) Application Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0														IRQ_MODE	
r														rw	

Field	Bits	Type	Description
IRQ_MODE	1:0	rw	**IRQ mode selection** The interrupt modes are described in **Section 28.4.5**. 00_B Level mode 01_B Pulse mode 10_B Pulse-Notify mode 11_B Single-Pulse mode
0	31:2	r	**Reserved** Read as zero, shall be written as zero.

28.17.11.10Register MCS[i]_CH[x]_EIRQ_EN

MCSi Channel x Error Interrupt Enable Register

MCSi_CHx_EIRQ_EN (i=0-9;x=0-7)
MCSi Channel x Error Interrupt Enable Register($0F0054_H$+i*1000_H+x*80_H) Application Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0													ERR_EIRQ_EN	STK_ERR_EIRQ_EN	MCS_EIRQ_EN0
r													rw	rw	rw

Field	Bits	Type	Description
MCS_EIRQ_EN0	0	rw	**MCS channel x MCS_EIRQ error interrupt enable** 0_B Disable error interrupt 1_B Enable error interrupt
STK_ERR_EIRQ_EN	1	rw	**MCS channel x STK_ERR_IRQ error interrupt enable** 0_B Disable error interrupt 1_B Enable error interrupt
ERR_EIRQ_EN	2	rw	**MCS channel x ERR_EIRQ error interrupt enable** 0_B Disable error interrupt 1_B Enable error interrupt
0	31:3	r	**Reserved** Read as zero, shall be written as zero.

28.17.11.11Register MCS[i]_CTRL_STAT

MCSi Control and Status Register

MCSi_CTRL_STAT (i=0-9)
MCSi Control and Status Register **($0F0064_H$+i*1000_H)** **Application Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0					HLT_AEIM_ERR	EN_XOREG	EN_TIM_FOUT	0	ERR_SRC_ID			0		HLT_SP_OFL	RAM_RST
r					rw	rw	rw	r	r			r		rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				SCD_CH				0						SCD_MODE	
r				rw				r						rw	

Field	Bits	Type	Description
SCD_MODE	1:0	rw	**Select MCS scheduling mode** 00_B Accelerated Scheduling 01_B Round Robin Scheduling 10_B Single Priority Scheduling 11_B Multiple Priority Scheduling
SCD_CH	11:8	rw	**Channel selection for scheduling algorithm** MCS-channel identifier used by several scheduling modes. *Note: The actual width of the bit field SCD_CH is calculated as ceiling[log_2(T+1)].* Unused MSBs are reserved and read as zero.
RAM_RST	16	rw	**RAM reset bit** *Note: The RAM reset initializes the memory content with zeros. RAM access and enabling of MCS channels is disabled during active RAM reset.* *Note: This bit is only writable if the bit RF_PROT in register GTM_CTRL is cleared, and all MCS-channels are disabled.* *Note: The actual reset value of this bit depends on the silicon vendor configuration. The reset value is 1, if the RAM reset is performed together with the sub-module reset, otherwise, the reset value is 0. If the reset value is 1, the reset value is changed to 0 by hardware when the RAM reset has finished.* 0_B READ: no RAM reset is active / WRITE: do nothing 1_B READ: MCS currently resets RAM content / WRITE: trigger RAM reset

Field	Bits	Type	Description
HLT_SP_OFL	17	rw	**Halt on stack pointer overflow** 0_B No halt on MCS-channel stack pointer counter over/underflow 1_B MCS-channel is disabled if a stack pointer counter over/underflow occurs
ERR_SRC_ID	22:20	r	**Error source identifier** *Note: This register is updated once, if an error was detected by the MCS. The register is set to its initial value 000_B after each write access to an existing ERR bit in register MCS[i]_ERR. If multiple errors occur, ERR_SRC_ID is holding the first type of error which has occurred.* 000_B No HW generated Error occurred 001_B Detected ECC error 010_B Detected memory overflow 011_B Detected invalid opcode 100_B Divide by zero 101_B Invalid register write access to GPR from write protected channel 110_B Invalid memory write access to protected memory region 111_B Invalid AEI bus master access
EN_TIM_FOUT	24	rw	**Enable routing of TIM[i]_CH[x]_F_OUT signal** 0_B Read access to register CTRG/MCS[i]_CTRG provides state of the internal trigger registers 1_B Read access to register CTRG/MCS[i]_CTRG provides state of the external signal TIM[i]_CH[x]_F_OUT
EN_XOREG	25	rw	**Enable extended register set** *Note: If the extended operation register sets are disabled, the MCS instructions can only use the subset OREG of the register set as arguments in the instructions. In this case, the upper address bits in the instructions are always read as zeros, which leads to unexpected results of the MCS program if arguments A or B refer to a register that is not part of OREG.* 0_B Extended operation register sets XOREG, BAREG, and WXREG are disabled 1_B Extended operation register sets XOREG, BAREG, and WXREG are enabled

Field	Bits	Type	Description
HLT_AEIM_ERR	26	rw	**Halt on AEI bus master error** *Note: If the register HLT_AEIM_ERR is set, and an MCS channel x is executing an invalid bus master access, the MCS channel x is halted, the ERR bit of its register STA is set, and the bit field ERR_SRC_ID of this register is updated.* *Note: If the bus master is accessing a slave that does not insert wait cycles (e.g. register access), it takes two additional clock cycles until the MCS channel is halted. Within that time span, the MCS channel can continue with its program execution, depending on the selected scheduling mode.* *Note: The registers AEIM_XPT_STA and AEIM_XPT_ADDR of the GTM sub-module CCM are always updated on the first invalid AEI bus master access, independently of the state of HLT_AEIM_ERR* 0_B Ignore invalid AEI bus master access 1_B Halt MCS-channel on invalid AEI bus master access
0	7:2, 15:12, 19:18, 23, 31:27	r	**Reserved** Read as zero, shall be written as zero.

28.17.11.12Register MCS[i]_REG_PROT

MCSi Write Protection Register

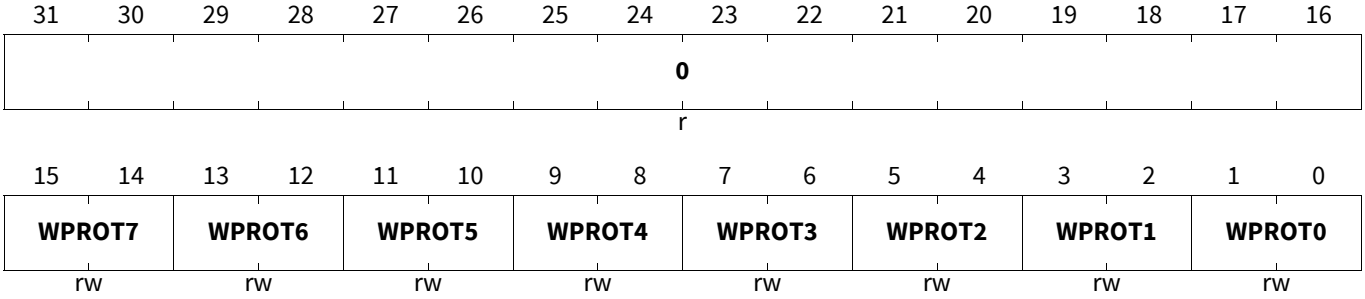
*Note: Only the first T bit fields of this register (bit 0 to 2*T-1) are functionally implemented. The other bits (bit 2*T to 31) are reserved bit fields.*

Note: The predecessor channel of MCS channel 0 is MCS channel T-1.

Note: If an MCS channel x is writing to a general purpose register that is write protected by register MCS[i]_REG_PROT the ERR bit of the register STA is set, the MCS channel x is halted and the ERR_SRC_ID bit field of register MCS[i]_CTRL_STAT is updated.

Note: This register is only writable if the bit RF_PROT in register GTM_CTRL is cleared

MCSi_REG_PROT (i=0-9)
MCSi Write Protection Register **($0F0060_H$+i*1000_H)** **Application Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
WPROTx (x=0-7)	2*x+1:2*x	rw	**Register Write Protection of MCS-channel x** 00_B No register write protection activated 01_B Predecessor MCS channel cannot write to its RS[y] registers 10_B Current MCS channel cannot write to its R[y] registers 11_B Reserved
0	31:16	r	**Reserved** Read as zero, shall be written as zero.

28.17.11.13Register MCS[i]_CTRG

MCSi Clear Trigger Control Register

Note: The trigger bits TRGx are accessible by all MCS channels as well as the CPU. Setting a trigger bit can be performed with the STRG register, in the case of an MCS-channel or the MCS[i]_STRG register in the case of the CPU. Clearing a trigger bit can be performed with the CTRG register, in the case of an MCS-channel or the MCS[i]_CTRG register in the case of the CPU. Trigger bits can be used for signalizing specific events to MCS-channels or the CPU. An MCS-channel suspended with a WURM instruction can be resumed by setting the appropriate trigger bit.

Note: Besides setting the trigger bits with register STRG/MCS[i]_STRG, the k-th trigger bit TRGx (with k < 16) can also be set by the external capture event that is enabled by the k-th bit of register GTM_EXT_CAP_EN_[i]. If bit k bit is disabled, the k-th trigger bit TRGk can only be set by MCS or CPU.

Note: In the scheduling modes Accelerated Scheduling and Round Robin Scheduling, a write access to MCS[i]_CTRG may take up to T + 1 clock cycles, since the write access is scheduled to the next CPU time slot determined by the MCS scheduler. In the modes Single Prioritization Scheduling and Multiple Prioritization Scheduling, no upper limit access time for a write access to MCS[i]_CTRG can be guaranteed. The High Prioritized tasks have to be disabled in order to guarantee fast write access to MCS[i]_CTRG.

Note: Note: The result of a read access to this register differs in dependency of the bit field EN_TIM_FOUT of register MCS[i]_CTRL_STAT.

MCSi_CTRG (i=0-9;x=0-7)

MCSi Clear Trigger Control Register (0F0028$_H$+i*1000$_H$+x*80$_H$) Application Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								TRG23	TRG22	TRG21	TRG20	TRG19	TRG18	TRG17	TRG16
r								rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRG15	TRG14	TRG13	TRG12	TRG11	TRG10	TRG9	TRG8	TRG7	TRG6	TRG5	TRG4	TRG3	TRG2	TRG1	TRG0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
TRGk (k=0-7)	k	rw	**Trigger bit k** READ access: State of current trigger bit**TRGk if EN_TIM_FOUT**= 0 State of input signal**TIM[i]_CH**[**k**]**_F_OUT if EN_TIM_FOUT**= 1 WRITE access: 0_B Do nothing 1_B Clear trigger bit TRGx
TRGk (k=8-15)	k	rw	**Trigger bit k** READ access: State of current trigger bit **TRGk if EN_TIM_FOUT** = 0 State of input signal **TIM[i+1]_CH**[k-8]**_F_OUT if EN_TIM_FOUT** = 1 WRITE access: 0_B Do nothing 1_B Clear trigger bit TRGk
TRGk (k=16-23)	k	rw	**Trigger bit k** 0_B READ: trigger bit is cleared / WRITE: do nothing 1_B READ: trigger bit is set / WRITE: clear trigger bit
0	31:24	r	**Reserved** Read as zero, shall be written as zero.

28.17.11.14Register MCS[i]_STRG

MCSi Set Trigger Control Register

Note: The trigger bits TRGx are accessible by all MCS channels as well as the CPU. Setting a trigger bit can be performed with the STRG register, in the case of an MCS-channel or the MCS[i]_STRG register in the case of the CPU. Clearing a trigger bit can be performed with the CTRG register, in the case of an MCS channel or the MCS[i]_CTRG register in the case of the CPU. Trigger bits can be used for signalizing specific events to MCS-channels or the CPU. An MCS-channel suspended with a WURM instruction can be resumed by setting the appropriate trigger bit.

Note: Besides setting the trigger bits with register STRG/MCS[i]_STRG, the k-th trigger bit TRGk (with k < 16) can also be set by the external capture event that is enabled by the k-th bit of register CCM[I]_EXT_CAP_EN. If bit k bit is disabled, the k-th trigger bit TRGk can only be set by MCS or CPU.

Note: In the scheduling modes Accelerated Scheduling and Round Robin Scheduling, a write access to MCS[i]_STRG may take up to T + 1 clock cycles, since the write access is scheduled to the next CPU time slot determined by the MCS scheduler. In the modes Single Prioritization Scheduling and Multiple Prioritization Scheduling, no upper limit access time for a write access to MCS[i]_STRG can be guaranteed. The High Prioritized tasks have to be disabled in order to guarantee fast write access to MCS[i]_STRG.

MCSi_STRG (i=0-9;x=0-7)
MCSi Set Trigger Control Register **(0F002C$_H$+i*1000$_H$+x*80$_H$)** **Application Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								TRG23	TRG22	TRG21	TRG20	TRG19	TRG18	TRG17	TRG16
r								rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRG15	TRG14	TRG13	TRG12	TRG11	TRG10	TRG9	TRG8	TRG7	TRG6	TRG5	TRG4	TRG3	TRG2	TRG1	TRG0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
TRGk (k=0-23)	k	rw	**Trigger bit k** 0_B READ: trigger bit is cleared / WRITE: do nothing 1_B READ: trigger bit is set / WRITE: set trigger bit
0	31:24	r	**Reserved** Read as zero, shall be written as zero.

28.17.11.15Register MCS[i]_RESET

MCSi Reset Register

Note: Only the first T bits of this register (bit 0 to T-1) are functionally implemented. The other bits (bit T to 31) are reserved bits.

Note: The RSTx (x = 0 ...T-1) bits is cleared automatically after write access of CPU. All channel related registers of channel x are set to their reset values and channel operation is stopped immediately.

Note: Channel related registers of channel x are all registers MCS[i]_CH[x]_*, all MCS internal registers accessible by the corresponding channel, with exception of the common trigger register (accessed by MCS[i]_CTRG/MCS[i]_STRG) and the commonly used general purpose registers MCS[i]_CH[x]_R4 and MCS[i]_CH[x]_R5.

MCSi_RESET (i=0-9)

MCSi Reset Register **(0F0068$_H$+i*1000$_H$)** **Application Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								RST7	RST6	RST5	RST4	RST3	RST2	RST1	RST0
r								rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
RSTx (x=0-7)	x	rw	**Software reset of channel x** 0_B No action 1_B Reset channel
0	31:8	r	**Reserved** Read as zero, shall be written as zero.

28.17.11.16Register MCS[i]_CAT

MCSi Cancel ARU Transfer Instruction Register

Note: Only the first T bits of this register (bit 0 to T-1) are functionally implemented. The other bits (bit T to 31) are reserved bits.

MCSi_CAT (i=0-9)

MCSi Cancel ARU Transfer Instruction Register(0F006C$_H$+i*1000$_H$) **Application Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								CAT7	CAT6	CAT5	CAT4	CAT3	CAT2	CAT1	CAT0
r								rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
CATx (x=0-7)	x	rw	**Cancel ARU transfer for channel x** *Note: The CATx (x = 0..T-1) bit inside the STA register of the corresponding MCS channel is set, and any pending blocking ARU read or write request is canceled. The MCS channel resumes with the instruction after the blocking ARU transfer instruction.* *Note: The CATx (x = 0..T-1) bit is cleared by the corresponding MCS channel when the channel is entering a blocking ARU read or write instruction.* 0_B Do nothing 1_B Cancel any pending ARU read or write transfer
0	31:8	r	**Reserved** Read as zero, shall be written as zero

28.17.11.17Register MCS[i]_CWT

MCSi Cancel WURM Instruction Register

Note: Only the first T bits of this register (bit 0 to T-1) are functionally implemented. The other bits (bit T to 31) are reserved bits.

MCSi_CWT (i=0-9)
MCSi Cancel WURM Instruction Register (0F0070$_H$+i*1000$_H$) Application Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								CWT7	CWT6	CWT5	CWT4	CWT3	CWT2	CWT1	CWT0
r								rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
CWTx (x=0-7)	x	rw	**Cancel WURM instruction for channel x** *Note: The CWTx (x = 0..T-1) bit inside the STA register of the corresponding MCS channel is set, and any pending WURM instruction is canceled. The MC -channel resumes with the instruction after the WURM instruction.* *Note: The CWTx (x = 0..T-1) bit is cleared by the corresponding MCS channel when the channel reaches a WURM instruction.* 0_B Do nothing 1_B Cancel any pending WURM instruction
0	31:8	r	**Reserved** Read as zero, shall be written as zero.

28.17.11.18Register MCS[i]_ERR

MCSi error register

Note: Only the first T bits of this register (bit 0 to T-1) are functionally implemented. The other bits (bit T to 31) are reserved bits.

MCSi_ERR (i=0-9)
MCSi error register **(0F007C$_H$+i*1000$_H$)** **Application Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								ERR7	ERR6	ERR5	ERR4	ERR3	ERR2	ERR1	ERR0
r								rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
ERRx (x=0-7)	x	rw	**Error State of MCS-channel x** *Note: The CPU can read the ERRx (x = 0..T-1) bits in order to determine the current error state of the corresponding MCS-channel x.* *Note: The error state is also evaluated by the sub-module MON, if this module is available.* *Note: Writing the value 1 to this bit resets the corresponding error state, and resets the channel internal ERR bit in the STA and channel CTRL registers. Moreover, each write access to this bit also sets the ERR_SRC_ID bit field of register MCS[i]_CTRL_STAT to its reset value.* 0_B No error signal 1_B Error signal is pending
0	31:8	r	**Reserved** Read as zero, shall be written as zero.