STORM **CORE** Processor System by Stephan Nolting

General Data Sheet

The **STORM** **CORE** processor system was created by Stephan Nolting.
Contact: stnolting@googlemail.com, zero_gravity@opencores.org

The most recent version of the core and it's documentary can be found at http://www.opencores.com/project,storm_core

Table of content

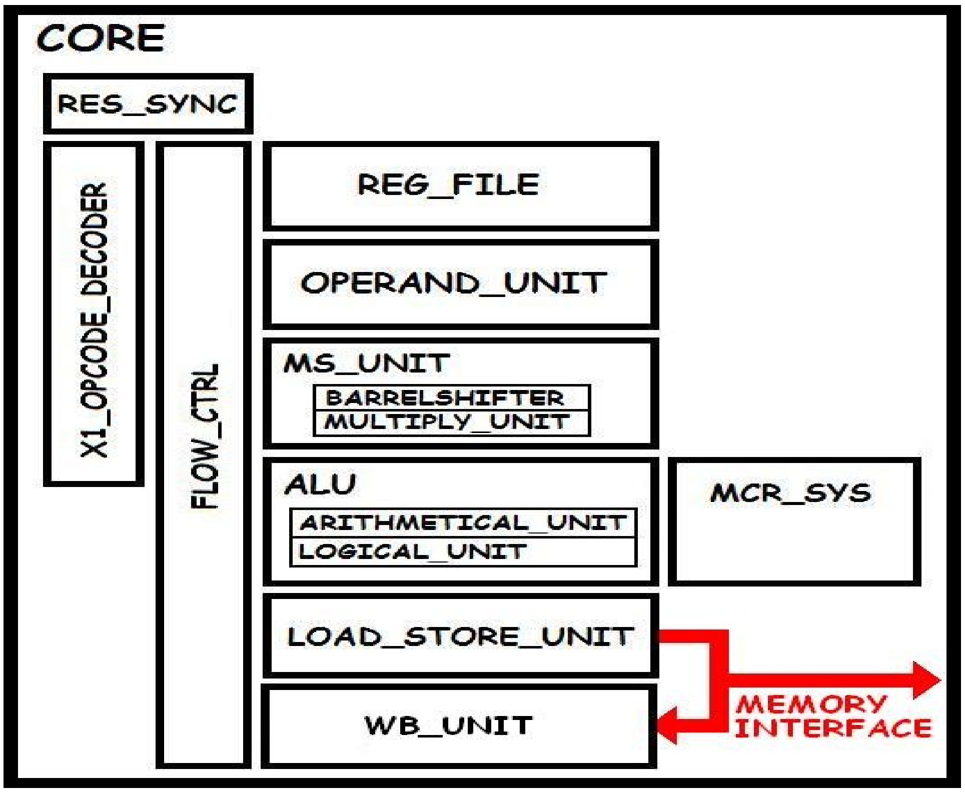
1. Introduction

The STORM CORE 32-bit RISC processor system, which is completely described in VHDL. It uses no dedicated hardware, so it can be synthesized for any FPGA device. The function set, the operation code and the interface timing are fully ARM compatible. For further compatibility see the chapter “Programmer's model”.

The VHDL top entity is “CORE.vhd” and the main package file, where all necessary parameters and modules are included, is “STORM_core.vhd”.

This document only gives a brief overview of the STORM CORE processor right now. But I will keep on working on this document ;)

1.1 VHDL module diagram



1.2 Module description

REG_FILE.vhd
The data register file. It consists of 32 registers, whereof 16 are accessible at one time (depending on the current operating mode).

OPERAND_UNIT.vhd
The operand unit performs the operand fetch for the three operand-slots.
Also the data conflict detector and the forwarding system are placed here.

MS_UNIT.vhd
This unit performs either a multiplication or a barrel shift on one of the ALU operand. Due to the three operand slots this instruction is for free (no additional cycles need to fetch multiplication or shift data).

BARREL_SHIFTER.vhd
This unit performs the arm-compatible barrel-shifting of the data in ALU data path B. The shift value can either be an immediate from the opcode, or a register value, which is loaded in the same cycle, no additional data load cycle is needed.

MULLTIPLY_UNIT.vhd
The multiply unit calculates a 32x32 bit operation and outputs the lower 32 bit to the ALU data path B.

ALU.vhd
The ALU holds the primary data operation units. All address-operations are done here(except for the program counter increment). Furthermore it handles the manual read/write access to the different machine status registers.

ARITHMETICAL_UNIT.vhd
The arithmetical unit performs the dedicated arithmetical add & sub operations and also the "arithmetical compares" (CMP, CMN).

LOGICAL_UNIT.vhd
The logical unit performs the dedicated logical operations like "BIC" and "OR" and also the"logical compares" (TST, TEQ).

LOAD_STORE_UNIT.vhd
The load-store unit outputs the correct address (instruction address or data address) to the memory address bus and also sends the write data and the control signals to the external memory interface.

WB_UNIT.vhd
The write-back unit performs the data write back to the register file and also accepts the read data from the memory.

MCR_SYS.vhd
The MCR system holds the machine control registers (program counter, all the saved machine status registers and the current machine status register) as well as the interrupt/context change system.

FLOW_CTRL.vhd
The flow control generates the control signals for each stage and every module. The decoded instruction data is brought to this unit where it triggers all internal operations. Furthermore the cycle arbiter, which solves temporal pipeline conflicts, and the branch arbiter are located here.

X1_OPCODE_DECODER.vhd
This unit decodes the ARM-compatible opcode into processor control signals.

RES_SYNC.vhd
The only purpose of this unit is to ensure that a valid reset holds at least 2 clock cycles (because the reset of all flip-flops are clock triggered).

STORM_core.vhd
This file is the main package file, where all modules and parameter are defined.

2. Signal description

The type of the core inputs and outputs is STD_LOGIC or STD_LOGIC_VECTOR (x downto y). Tie all inputs, that are not listed here, to LOW and leave all outputs, that are not mentioned here, unconnected. This ports are for debugging use or are not implemented yet.

CLK (1 bit, input)
This is the master clock signal for all internal operations. All register trigger the data reception on the rising edge of this signal.

RES (1 bit, input)
When this pin is driven HIGH a rising edge on the CLK input will perform a system reset. When driven LOW again the processor restarts instruction processing from address 0 and in supervisor mode. Attention: The content of the data register file after reset is unknown.

HALT (1 bit, input)
Whenever this pin is tied to HIGH potential, the internal CLK network goes LOW, so the processor can be freezed at any time (for example to wait for data from an external memory).

MODE (5 bit, output)
The current processor operating mode is displayed on this port.

MREQ (1 bit, output)
When this signal goes HIGH, the processor requires memory access during the next clock cycle.

MEM_ADR (32 bit, output)
This port provides the address for any memory access.

MEM_RD_DATA (32 bit, input)
Data from the memory is read from this on the rising or falling edge (configurable in the STORM_core package) of the CLK signal.

MEM_WR_DATA (32 bit, output)
The memory write data is displayed on this port.

MEM_MODE (1 bit, output)
This signal indicates the data transfer quantity for the memory access.
HIGH means word (32 bit) quantity, LOW means byte (8 bit) quantity
MEM_WE (1 bit, output)
This signal enables the data write process for the memory system.

MEM_RW (1 bit, output)
This signal indicates if a write (HIGH) or read (LOW) access is taking place.

MEM_OPC (1 bit, output)
When this signal is HIGH, a instruction fetch takes place, when it es LOW, data is fetched to or from the memory.

MEM_ABORT (1 bit, input)
A HIGH level forces the processor to take the memory abort trap.

IRQ (1 bit, input)
A HIGH level forces the processor to take the IRQ trap.

FIQ (1 bit, input)
A HIGH level forces the processor to take the FIQ trap.

3. Programmer's model

The STORM CORE is intended to be an “ARM7 compatible” processor system, which means, that the programmer's models should be are nearly the same. I designed the internal structure of the STORM CORE by myself, without knowing the internal hardware of an ARM7, so the behavior might differ in some small aspects. Important differences between the original ARM and the STORM are noted in this chapter.

For specific information about the programmer's model of the ARM7, see it's specific data sheet.

3.1 Operating modes

Six operating modes are yet implemented in the STORM CORE. When resetting the processor, it resumes operation always in supervisor mode. To change to a different mode, just write the corresponding MODE code to the lowest 5 bit of the CMSR, when the processor is in a higher privilege mode than “user mode”. The program counter is not affected when using this style of changing the current mode.

The current mode is also displayed on the MODE port of the STORM processor. All mode codes are bit-compatible to the ARM7.

MODE	MODE code
User	“10000”
Undefined	“11011”
Supervisor	“10011”
Abort*	“10111”
IRQ	“10010”
FIQ	“10001”

*) Only data abort (MEM_ABORT pin). There is no prefetch abort implemented yet.

3.2 Registers

The STORM CORE provides six different operating modes, where every mode can access it's own register set (Rx - Ry), including the PC (program counter, always R15), the current machine status register (CMSR) and a saved machine status register (SMSR_<mode>).

MODE	MODE code	Accessible registers
User	“10000”	R0 - R14, CMSR
FIQ	“10001”	R0 - R07, R08_FIQ - R14_FIQ
IRQ	“10010”	R0 - R12, R13_IRQ - R14_IRQ
Supervisor	“10011”	R0 - R12, R13_SVP - R14_SVP
Abort	“10111”	R0 - R12, R13_ABT - R14_ABT
Undefined	“11011”	R0 - R12, R13_UND - R14_UND

When writing to R15 (= PC), a jump to the written value will occur.
When reading from R15, the result is the program counter value (= address) of the corresponding operation (in pipeline stage “EX”), that is reading from R15.

3.3 Exceptions

Some processor modes cannot only be accessed by modifying the least significant 5 bits of the CMSR, but also can be accessed by special events (listed below). In this case, a branch to the corresponding trap address is performed. Furthermore the PC value (= address) of the last executed instruction (+4, → return address) in the previous mode is stored in R14_<new_mode>.

Undefined mode:	Execute the “undefined instruction” instruction.
Supervisor mode:	Execute the “software interrupt” instruction.
Abort mode:	Take the MEM_ABORT pin to high level.
FIQ/IRQ mode:	Take the corresponding pin (IRQ/FIQ) to high level.

The different interrupt vectors of the modes are listed below:

MODE	Interrupt address
User	0x00000000
Undefined	0x00000004
Supervisor	0x00000008
Abort*	0x0000000C
IRQ	0x00000018
FIQ	0x0000001C

*) Only data abort (MEM_ABORT pin). There is no prefetch abort implemented yet.

For further information about exception handling, see the ARM7 data sheet.

3.5 Differences between ARM7 and the STORM CORE

- The original ARM7 has got 3 pipeline stages, the STORM CORE uses 7 internal pipeline stages.
- Moe to come....

4. Core hardware

This chapter is about the internal RTL structure of the STORM CORE processor.

4.1 Data flow

The data flow of the STORM STORM is completely synchronous to the rising edge of the main clock signal (CLK). Even the reset of all register is synchronous to this signal. So an external hold signal (HALT) can be pulled HIGH at any time and for any time.

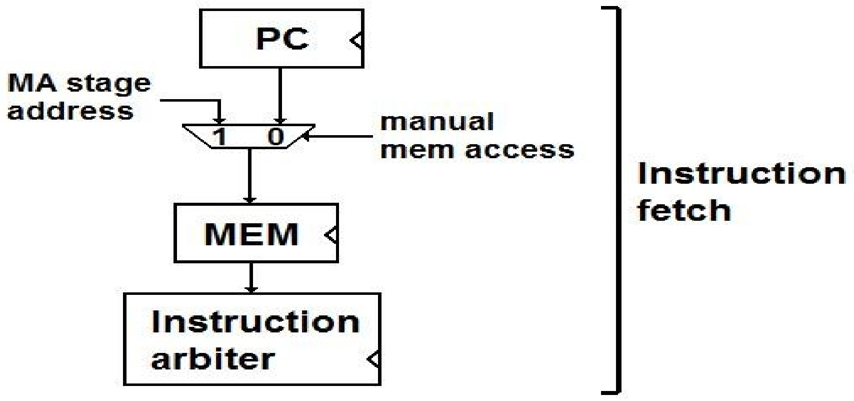
The pipeline consists of 7 internal stages. An external memory system needs at least one additional cycle to provide the data from the memory. The resulting 8 stages of the instruction processing pipeline are:

1. IA: Instruction access (program counter)
2. IF: Instruction fetch (external memory)
3. ID: Instruction decode
4. OF: Operand fetch
5. MS: Multiplication / Shift
6. EX: ALU execution / MCR access
7. MA: Memory access
8. WB: Data write back

When an external memory manager needs more than 1 cycle to output the correct data, it have to pull the HALT signal to HIGH level until it is able to accept the memory request.

The memory request signal (MREQ) itself is always on HIGH level (showing a request) when the pipeline is not stopped or a branch is taking place. If the pipeline is stopped (because of some data dependence), the MREQ signal goes to high in the EX stage, indicating a manual memory access.

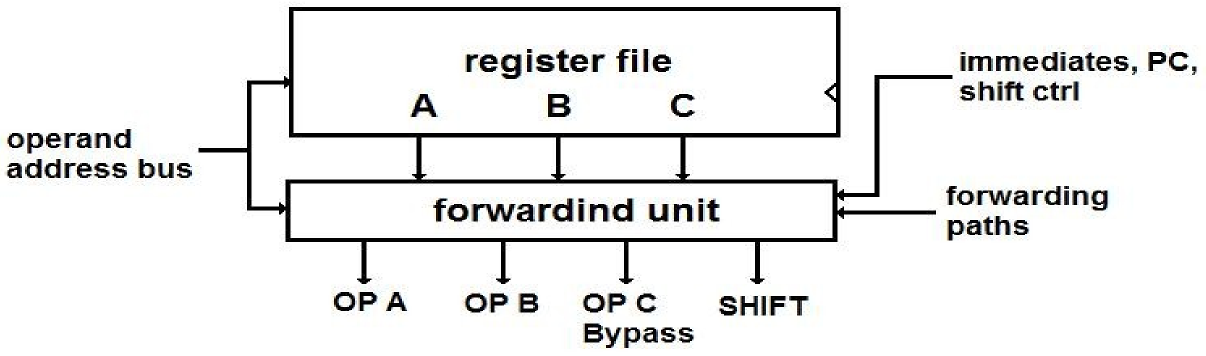
A new instruction (cycle) starts with the new value for the the program counter, which is located in the machine control register file (MCR_SYS.vhd). For normal linear operation, this value is old_program_counter + 4. After a rising_edge this signal is applied to the memory bus, when no manual data memory access is requested in the EX stage.



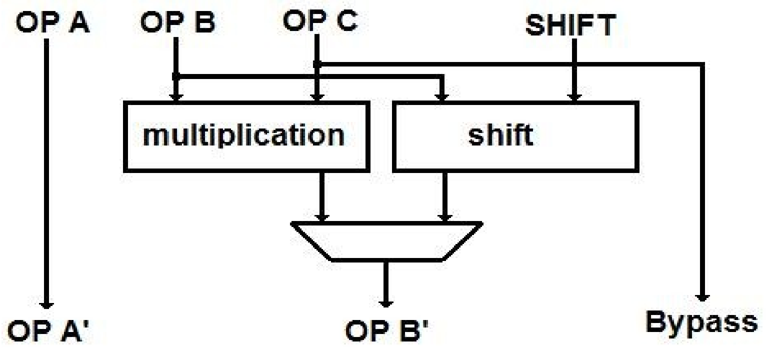
The memory system accepts the instruction request and outputs it after another clock cycle to the instruction arbiter (FLOW_CTRL.vhd).
The arbiter can buffer up to 3 instructions plus the current instruction in a small memory. This allows a continuous instruction flow, even when doing branches or stopping the pipeline for any other reason.

One cycle later, the correct instruction word is applied to the instruction decoder (X1_OPCODE_DECODER.vhd).

Another cycle later the decoded control signals arrive the first operation stage, the operand fetch stage. The operand fetch unit (OPERAND_UNIT.vhd) loads register values from the register file, constructs immediate values, loads the current or past program counter if required and loads the values for the barrelshifter (value, direction).

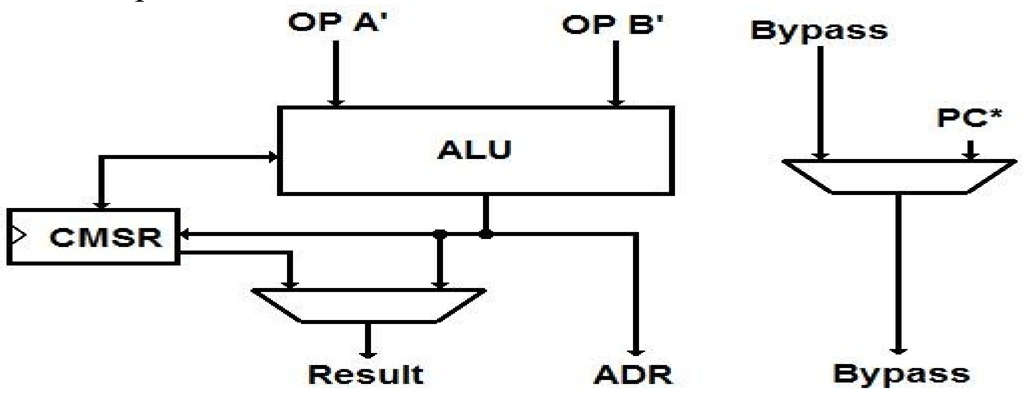


In parallel, the other control signals from the decoder are brought to the flow control, which serves as a large shiftregister for the control signals.
In the next stage a multiplication or a shift is performed (MS stage).

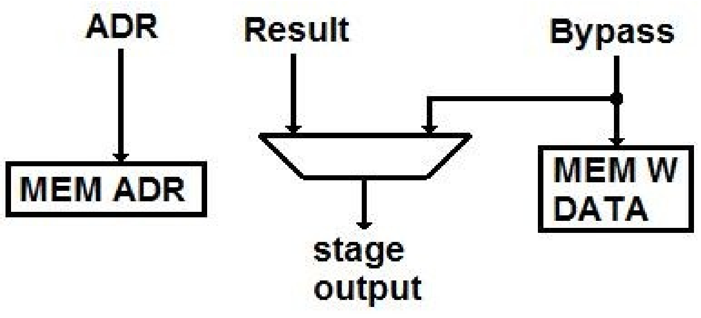


The next stage is the main execution stage (EX). The arithmetical and logical operations take place in this module. Also the current machine status register is located here. Flags can be read for proper arithmetical/logical operations and also the condition check is done in this stage (but in an other module: FLOW_CTRL). So all instructions, even with a not fulfilled condition code, are valid until this stage, if they were not marked as invalid by the instruction arbiter or the branch control.

The ALU can also perform write access to the MCR file.
When a rising edge appears, the CMSR is updated with the new flag values (arithmetical/logical flags, mode code, …) and also the PC is loaded with its new value, which is “old value” + 4 for normal operations. The data result from the arithmetical/logical unit can either be the true arithmetical/logical calculation result, or a machine status register (CMSR).
The bypass path is used for memory-write data and also for starting interrupts. The PC value of the instruction, which was in EX stage when the interrupt request came, will set on the bypass bus and saved in the link register to resume normal operation, when the interrupt handler has finished.



The next stage performs the memory access (MA). The correct address (instruction or data address) gets send to the memory interface, also all memory control signals are generated in this stage, except for the MREQ signal, which is computed in the EX stage. Also a logic for producing a byte or a word output data packet is implemented here.



The final stage is the data write back stage (WB). Read-data from an external memory is read into this stage, where it also gets aligned, depending on the read data quantity and the "word" address offset (see chapter "Memory Interface").
Data from this stage is directly written to the destination register in the register file.

4.2 Pipeline conflicts

When executing linear programs (no branches) without any dependencies between instructions in the pipeline and no manual memory accesses, there are no pipeline conflicts. For all other cases we need a logic, that solves this pipeline conflicts. There are two different types of conflicts: Just to differentiate between them, I will call them "local pipeline conflicts" and "temporal pipeline conflicts".

4.2.1 Local pipeline conflicts

Local pipeline conflicts just means, that the needed results for further processing have not yet reached the register file and are somewhere else in the pipeline.

For example imagine a program like this:

	ADD R1, R2, 1	(**R1** = R2 + 1)
	ADD R5, R4, 2	(R5 = R4 + 4)
	ADD R3, R1, 1	(R3 = **R1** + 1)

The third "ADD" need the result of the first "ADD". But when the third ADD is in the operand fetch stage, the first just have reached the EX stage. Since the first instruction needs no further processing, the result is already correct, we just need to wait until it is written back to register file. But then we would have to wait for the whole instruction to finish the pipeline.
But since the data dependence is local, the correct result is just somewhere else in the pipeline. We can now just use the forwarding unit to forward the data from the ALU result output to the operand fetch unit. There the value of R3 gets not read from the register file, but from the forwarding system instead.

The forwarding system can forward data from the EX stage, the MA stage and the WB stage. It is based in the OPERAND_UNIT.vhd file.

4.2.2 Temporal pipeline conflicts

Temporal pipeline conflicts occur, when the processor is trying to forward a result, that has not been calculated yet. So the conflict cannot be solved by forwarding data from some other pipeline stage, since the correct data does not exist.

Program example: **ADD R1, R2, 1** (**R1** = R2 + 1)
SUB R3, R1, 1 (R3 = **R1** - 1)

When the second instruction is in the operand fetch stage, the first is in the MS stage, so no subtraction has taken place yet.
The processor can detect this conflict and stalls the instruction fetch for one cycle. That means, the ADD instruction can resume processing in the pipeline, while the SUB instruction stays in the OF stage until all needed date is available. The empty "slots" between this instruction (OF: SUB, MS: NOP, EX: ADD) are filled with "NOPs". This "no-operation" instructions do not perform any data manipulation.

Temporal data dependencies can occur in the OF, the MS and the EX stage, when trying to get not yet calculated data or when trying to access the memory when the instruction pointer is currently accessing the memory.
The unit, which solves this conflicts is the "Temporal Data Dependence Detector" in the OPERAND_UNIT.vhd file and the "Instruction Arbiter" in the FLOW_CTRL.vhd file.

4.2.3 Branches

There are three causes for a non linear change of the program counter:

- unconditional/conditional branches
- interrupts
- manual writing to R15

All these operations result in a branch to a new address (new PC value). The PC gets updated with non-linear data (= when the new PC value is not "old_value + 4") between the EX and the MA stage.

Imagine a jump to subroutine:

B subroutine (branch)
ADD R3, R0, R1 (obsolete)
EOR R5, R0, R1 (obsolete)
SUB R2, R0, R1 (obsolete)

When the branch instruction ("B") is in EX stage, the ADD is in MS stage, the SUB is in ID stage and the EOR is in OF stage. All this instructions have to be invalidated by the branch arbiter ("branch cycle arbiter" in the FLOW_CTRL.vhd file). Until the processing can resume at the new position, the new address hast to be clock into the PC, send to the memory and the new opcode needs to be stored in the instruction arbiter, so the instruction processing - starting in the ID stage – needs to be disabled for the next 2 cycles.

4.3 Stage control bus

The stage control bus ("CTRL" in the VHDL modules) contains all signals, that determine the function of a module. Every top unit of each pipeline stage is connected to these bus, which comes from the "stage control shift register" (→ FLOW_CTRL.vhd). The signal locations inside the bus can be found in the "STORM_core.vhd" package file.

NOTE: The processor is still in the "planning" phase, so some of the signal definitions can slightly change in further STORM versions.

0: CTRL_EN:
When set to HIGH, this signal shows that the stage is processing a valid instruction. The enable signal allows the write back to a register (data register, memory interface register, CMSR,...). It is generated by the instruction arbiter and AND-ed later with the enable signal of the branch cycle arbiter and the enable signal of the condition validation system.

1: CTRL_CONST:
This signal is HIGH when operand b is not a register value, but an immediate value. It is only used in the operand fetch unit.

2: CTRL_BRANCH:
When set to HIGH, this signal indicates that the current instruction is a branch instruction (= modification of PC). In the EX stage this signal is AND-ed with the CTRL_EN signal, to show if a branch has been taken.

3: CTRL_LINK:
When this signal is set, the operand unit fetches the corresponding PC into OP_C, so it can be saved later on as link address.

4: CTRL_SHITR:
The positions data is being shifted comes from a register, when this signal is HIGH. It comes from an immediate, when the signal is LOW.

5: CTRL_WB_EN:
A data write back to the register file is only performed when this signal and the CTRL_EN signal are both HIGH. So this signal can be use to determine if the result of an instruction shall be written back to the register file or not (→ compares).

6 – 9: CTRL_RD:
This bits hold the destination register address, in which the operation result shall be written back.

10: CTRL_SWI:
This bit triggers the software interrupt when set to HIGH and the CTRL_EN is set too.

11: CTRL_UND:
This bit triggers the undefined instruction interrupt when set to HIGH and the CTRL_EN is set too.

12 – 15: CTRL_COND:
This bits hold the condition code of the corresponding operation.

16: CTRL_MS:
When set to 1, the MS stage outputs the multiplication result. When set to 0, the MS stage outputs the shift result.

17: CTRL_AF:
When this bit is set and the corresponding instruction is valid (CTRL_EN = 1), the new calculated flags form the ALU will be stored in the CMSR.

18 – 21: CTRL_ALU_FS:
This bits determine the ALU function set.

22: CTRL_MEM_ACC:
When set to one, this bit indicates that a manual memory access shall be performed.

23: CTRL_MEM_M:
This bit determines the data quantity for manual memory access (1 = byte quantity, 0 = word quantity).

24: CTRL_MEM_RW:
Read/ write signal for manual memory access (0 = read, 1 = write).

25: CTRL_MREG_ACC:
When set to one, this bit indicates that a manual CMSR/SMSR access shall be performed.

26: CTRL_MREG_M:
MCR access regulation, a 0 will access the CMSR, a 1 will access the SMSR.

27: CTRL_MREG_RW:
Read/ write signal for manual CMSR/SMSR access (0 = read, 1 = write).

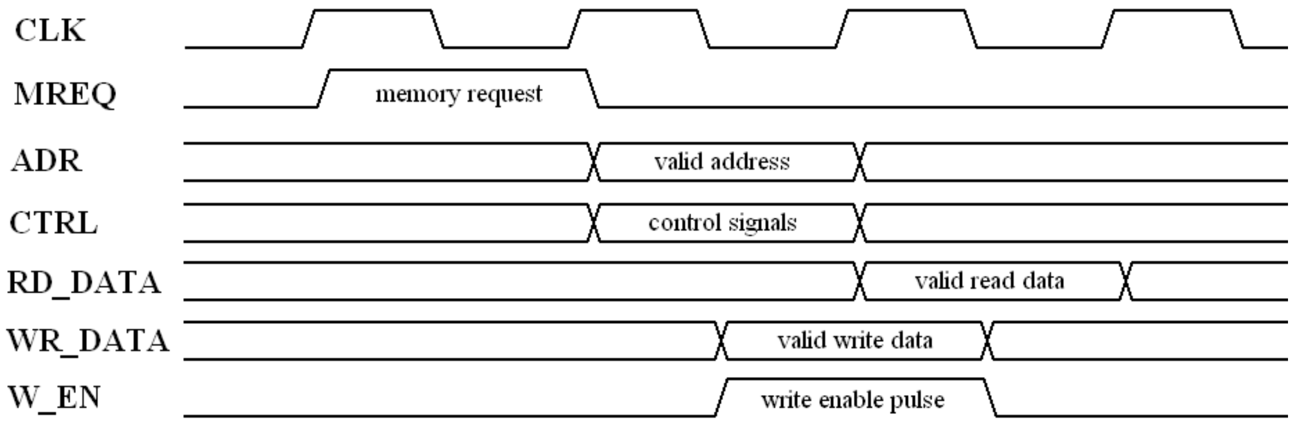
28: CTRL_MREG_FA:
CMSR access mode control: A 0 gives full access to the CMSR, a 1 will only update the flag bits with manual data.

29: CTRL_MC:
When set to one, this signal indicates that the corresponding instruction will perform a mode change.

17 – 21: CTRL_MODE:
This bits are reused at the end of the pipeline to store the corresponding mode for a correct data write back.

5. Memory interface

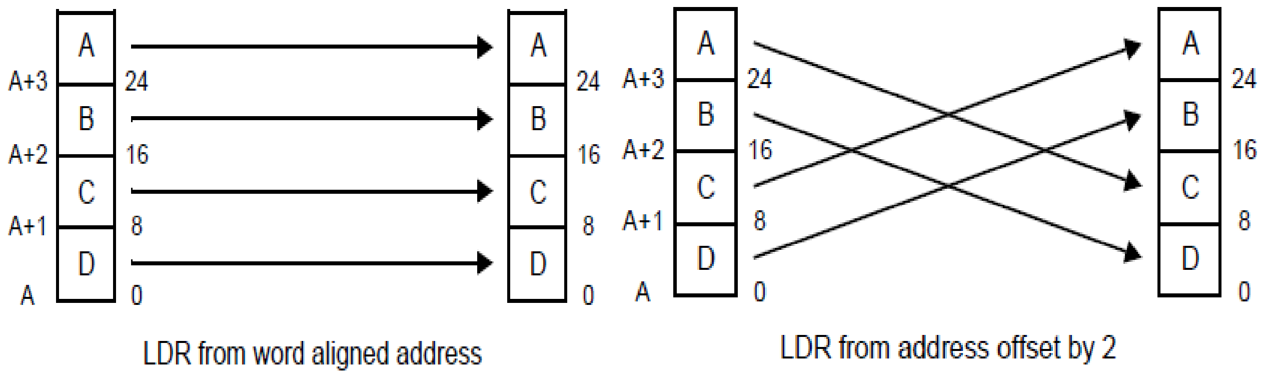
The STORM processor uses the same bus system for fetching instructions and for reading or writing data to the memory. All buses have a data width of 32 bit, so a single instruction can be loaded in one cycle when the memory is organized as Nx32bit memory. Otherwise the memory manager needs to take the HALT signal to HIGH level, until all parts of a complete opcode are loaded.
The "CTRL bus" contains the MEM_RW, MEM_OPC and the MEM_MODE signal.



(single memory read/ write access)

If a memory controller cannot accept the memory request of the processor, it needs to set the HALT signal to HIGH level. This freezes the whole processor (in any state).

Instructions need to be stored at word aligned addresses, because the program counter increases by 4. Data can be stored and loaded with any offset (0 to 3 byte).


A+3
A+2
A+1
A
A
B
C
D
24
16
8
0
LDR from word aligned address
LDR from address offset by 2