ASTRON
Netherlands Institute for Radio Astronomy

Heater Logic Module Description

	Organisatie / Organization	Datum / Date
Auteur(s) / Author(s): Eric Kooistra	ASTRON	
Controle / Checked: Andre Gunst	ASTRON	
Goedkeuring / Approval: Andre Gunst	ASTRON	
Autorisatie / Authorisation: **Handtekening / Signature** Andre Gunst	ASTRON	

Distribution list:

Group:	Others:
Andre Gunst (AG, ASTRON) Daniel van der Schuur (DS, ASTRON) Rajan Raj Thilak (RT, ASTRON) Arpad Szomoru (JIVE) Jonathan Hargreaves (JH, JIVE) Salvatore Pirruccio (SP, JIVE)	Gijs Schoonderbeek (GS, ASTRON) Sjouke Zwier (SZ, ASTRON)

Document history:

Revision	Date	Author	Modification / Change
0.1	2010-11-09	Eric Kooistra	Creation.
0.2	2010-11-10	Eric Kooistra	Improved the order of the sections.
0.3	2010-11-15	Eric Kooistra	Added util_logic.
0.4	2010-11-30	Eric Kooistra	Updated after review on Nov 25, 2010 with EK, DS, JH, SP. Refer to ASTRON-RP-426 for basic node design features.
0.5	2010-12-01	Eric Kooistra	Moved clock domain section to hardware interface chapter. Used tables to list the files in the appendix.
1.0	2011-02-03	Eric Kooistra	Added known issues section to fit the template.

Table of contents:

Terminology:

DP	Data Path
DSP	Digital Signal Processing
DUT	Device Under Test
FF	Flip Flop (clocked or registered logic)
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
HDL	Hardware Description Language
IO	Input Output
IP	Intellectual Property
LUT	Look Up Table (combinatorial logic)
MAC	Multiply and Accumulate
MAS	Master
MISO	Master In Slave Out
MM	Memory-Mapped
MOSI	Master Out Slave In
Nof	Number of
PHY	Physical layer
PIO	Parallel IO
PRSG	Pseudo Random Sequence Generator
RAM	Random Access Memory
ROM	Read Only Memory
RTL	Register Transfer Level
SLA	Slave
SISO	Source In Sink Out
SOPC	System On a Programmable Chip (Altera)
SOSI	Source Out Sink In
ST	Streaming
UNB	Path to UniBoard Firmware directory
UTIL	Utility
WDI	Watchdog Interrupt

References:

1. www.altera.com, stratix4_handbook.pdf
2. www.altera.com, quartusii_handbook.pdf
3. www.altera.com, mnl_avalon_spec.pdf
4. https://svn.astron.nl/UniBoard_FP7/UniBoard/trunk, the UniBoard FP7 SVN repository, see below.
5. “Common Library Memory and Register Component Descriptions”, ASTRON-RP-415, E. Kooistra
6. “UNB_Common Module Description”, ASTRON-RP-426, Eric Kooistra

Relevant directories in the UniBoard FP7 SVN repository:

$UNB: Firmware/doc/howto/ → How to descriptions
$UNB: Firmware/modules/ → HDL modules
$UNB: Firmware/designs/ → FPGA HDL designs
$UNB: Firmware/software/modules/src/ → C, H modules
$UNB: Firmware/software/apps/ → C main applications

1 Introduction

1.1 Purpose

This document is a user guide and a description of the heater VHDL module.

The purpose of the heater module is to provide an easy way to use almost all of the internal resources on an FPGA; meaning logic, multipliers and RAM. The heater module has no functional purpose. Instead the heater module can be used to verify power consumption of the internal resources of the FPGA and to verify the performance of the FPGA IO like DDR3 access and transceiver links at higher temperatures.

Section 2 describes the software interface of the module and contains sufficient information for on chip microprocessor software development.

Section 3 describes the hardware interface of the module and contains sufficient information for using the module in an SOPC Builder system or for instantiating it manually in a VHDL design.

Section 4 describes an example application and contains sufficient information for synthesizing the module in a design and to run a software main() application.

The subsequent sections describe the VHDL design, implementation and verification of the heater module and its internal details. These sections contain sufficient information for maintaining the module.

1.2 Module overview

Figure 1 shows the interfaces of the heater module. On the application side the heater module is a MM slave with one register that allows dynamically control of switching more or less of the internal resources on or off. The streaming interface only consists of a ST clock and reset input that are used to clock and reset the internal streaming resources. The used internal resources are logic (LUTs and FFs), DSP multipliers (*) and block RAM. The heater module has no streaming data IO, because it implements a dummy function.

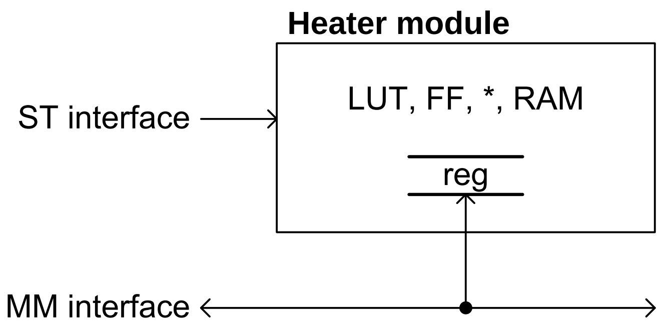


Figure 1: Heater module interfaces

2 Software interface

2.1 Operation

The heater module is a MM slave with only one MM register and no interrupt. The heater module consists of a number of identical elements. The total amount of available elements in an instance of the heater module is fixed and defined by g_nof_mac4 in the HDL. At run time the clock enable for each element can be set on or off via a bit in the MM heater register. When the clock enable is on then the corresponding element runs on the ST clock and dissipates power. When the clock enable is switched off then the corresponding element does not change state so then it dissipates almost no power.

2.2 MM registers

The Nios II architecture is little endian. Words and half words are stored in memory with the more-significant bytes at higher addresses. Bit 31 indicates the MSBit and bit 0 indicates the LSBit of a word. The heater module MM interface has to be accessed per word. Access per byte is not supported. The heater register is little endian. Table 1 lists the registers (only one) that are available in the heater module.

Name	**Address (words)**	**Size (words)**	**Read/ Write**	**Description**
Heater	0	16	RW	Enable heater elements

Table 1: Heater module registers

2.2.1 Heater register

Table 2 defines the bits in the heater register for a maximum of 352 mac4 elements. The maximum number of effective bits is equal to g_nof_mac4. The remaining bits and words in the heater register are empty. Setting a bit to ‘1’ enables the element and setting it to ‘0’ disables the element. Reading a bit yields a varying and undefined result when the element is enabled and a stable result when the element is disabled.

Offset (bytes)	**Bits 31:0**
0	HEATER_ELEMENT_EN[31:0]
4	HEATER_ELEMENT_EN[63:32]
8	HEATER_ELEMENT_EN[95:64]
12	HEATER_ELEMENT_EN[127:96]
16	HEATER_ELEMENT_EN[159:128]
20	HEATER_ELEMENT_EN[191:160]
24	HEATER_ELEMENT_EN[223:192]
28	HEATER_ELEMENT_EN[255:224]
32	HEATER_ELEMENT_EN[287:256]
36	HEATER_ELEMENT_EN[319:288]
40	HEATER_ELEMENT_EN[351:320]
44	-
48	-
52	-
56	-
60	-

Table 2: Heater register (g_nof_mac4 = 352)

Each mac4 element uses 4 18x18 multipliers. The Stratix IV GX230 has 1288 18x18 multipliers, so in total it can fit maximally 322 mac4 elements. For the heater register this implies the maximum that can be used for g_nof_mac4 = 322 and that the register bits [351:322] are then void. The heater register has 352/32 = 11 words. This implies that the register address width is 4 bit and that the words at [15:11] do not exist.

2.3 Software functions

The heater module software consists of avs_util_heater.c/h and avs_util_heater_regs.h and provides a public function to control the heater peripheral. Table 3 lists the software functions that are available for controlling the heater peripheral.

Function	**Description**
UTIL_HEATER_Control()	Specify the number of heater elements that need to be switched on. Specifying 0 causes all elements to be switched off. Switching on a specific element is not supported, because all elements are equivalent.

Table 3: Heater functions in the avs_util_heater.c/h software module

3 Hardware interface

3.1 Clock domains

Figure 2 shows the two clock domains that are used by the heater module:

- mm_clk = MM clock for the memory-mapped bus with the NIOS II processor
- st_clk = ST clock for the heater module elements

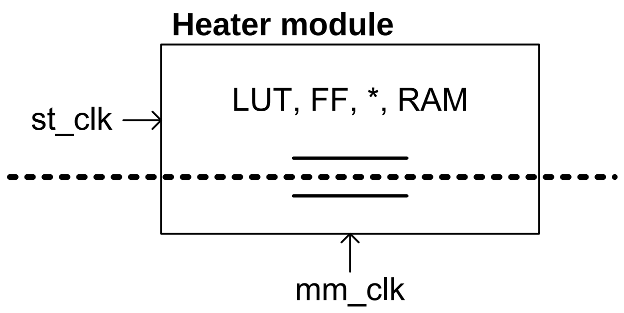


Figure 2: Clock domains of the heater module

3.2 Parameters

Table 4 lists the VHDL generics that can be when the heater module is instantiated in another HDL design or in an SOPC system.

Generic	Type	Description
g_nof_mac4	NATURAL	>= 1, number of multiply 18x18 and accumulate 4 elements in the heater <= c_util_heater_nof_mac4_max = 352. Use this to define the number of heater elements in the heater instance.
g_pipeline	NATURAL	>= 0, number of pipelining register stages. Use this to define the amount of logic (FFs or RAM dependent on the synthesis) that is used per heater element.
g_nof_ram	NATURAL	>= 0, number of 1 kByte RAM blocks. Use this to define the amount of RAM that is used per heater element
g_nof_logic	NATURAL	>= 0, number of logic register stages. Use this to define the amount of logic (LUTs and FFs) that is used per heater element.

Table 4: Heater module parameters

The Stratix4 GX230 PFGA has 1288 multipliers, so it can fit maximum g_nof_mac4 = 322. The maximum values for RAM and logic also depend on the size of the FPGA.

3.3 MM interface

Table 5 defines the clock and reset for the MM interface.

Signal	Type	Description
mm_clk	CLOCK	Clock input for the MM interface side of the heater module
mm_rst	RESET	Reset input for the MM interface side of the heater module

Table 5: Heater MM interface clock and reset

Table 6 defines the interface signals for the heater MM slave register of Table 2.

Signal	Type	Description
sla_out.rddata[31:0]	MISO	Read data word, valid 1 clock cycle after rd
sla_in.address[3:0]	MOSI	Word address range to fit the module registers of Table 1
sla_in.wrdata[31:0]	MOSI	Write data word, must be valid with wr
sla_in.wr	MOSI	Write strobe
sla_in.rd	MOSI	Read strobe

Table 6: Heater MM register interface

In an SOPC system the MM interface signals from Table 5 and Table 6 get connected automatically.

3.4 ST interface

Table 5 defines the clock and reset for the ST interface.

Signal	Type	Description
st_clk	CLOCK	Clock input for the ST interface side of the heater module
st_rst	RESET	Reset input for the ST interface side of the heater module

Table 7: Heater ST interface clock and reset

The ST interface clock and reset are used for running the heater elements. The ST interface of the heater does not source or sink external data.

In an SOPC system the ST interface signals from Table 7 can be connected to an internal clock (e.g. from a PLL in the SOPC system) or via conduit [3] input pins.

3.5 Other interfaces

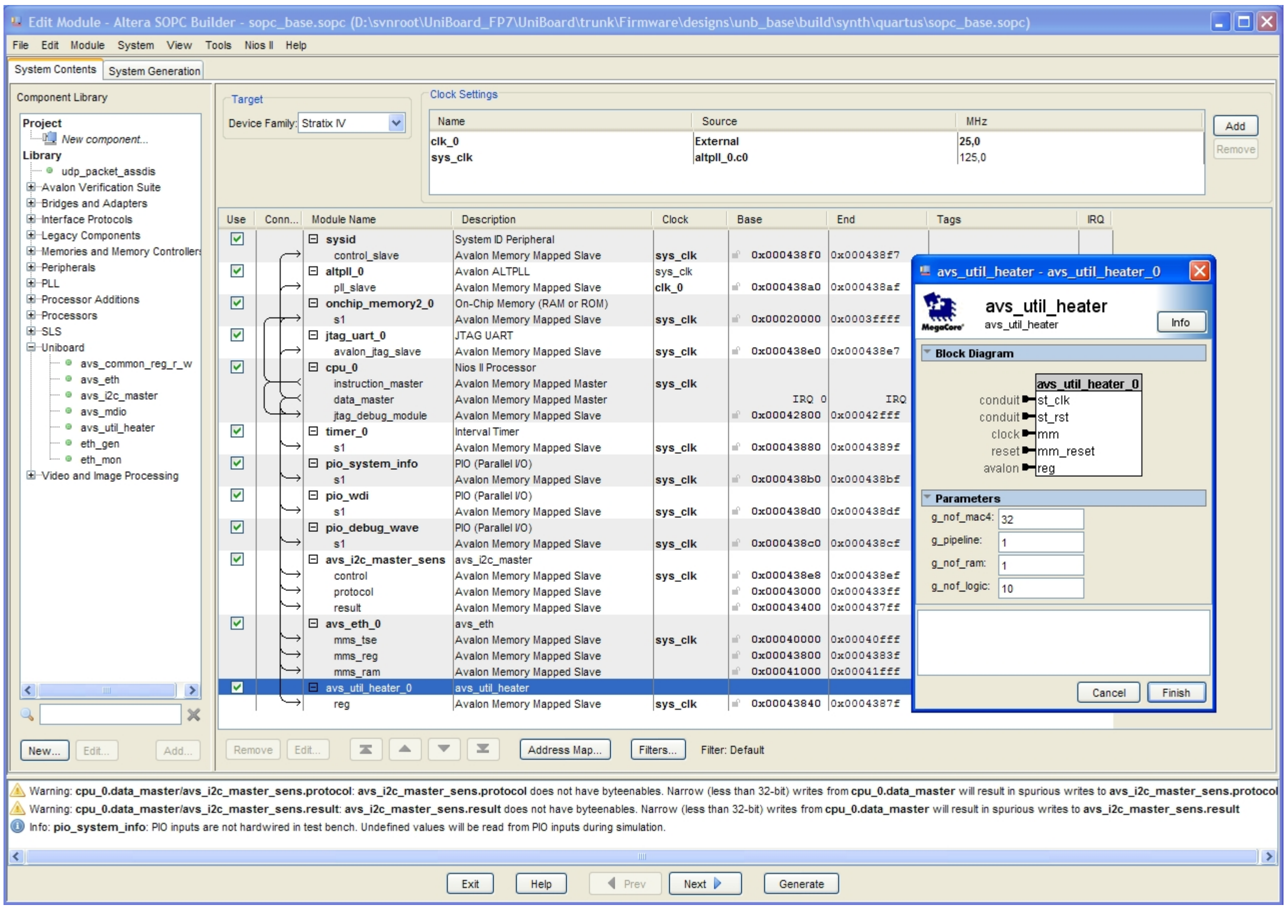
The heater module has no other interfaces than MM or ST.

4 Application

4.1 SOPC design

Thanks to a VHDL wrapper component and a hardware description TCL file, see Appendix 8, the heater module is also available within SOPC Builder [2]. Figure 3 shows the heater module called avs_util_heater in an SOPC system. From the SOPC Builder GUI it is possible to set the heater module parameters that were described in section 3.2.

The example sopc_base SOPC system is instantiated in the unb_base.vhd node design. For more information on instantiating an SOPC system in a UniBoard node design see [6].



Edit Module - Altera SOPC Builder - sopc_base.sopc (D:\svnroot\UniBoard_FP7\UniBoard\trunk\Firmware\designs\unb_base\build\synth\quartus\sopc_base.sopc)

Device Family: Stratix IV

Name	Source	MHz
clk_0	External	25,0
sys_clk	altpll_0.c0	125,0

Use	Conn...	Module Name	Description	Clock	Base	End	Tags	IRQ
✓		sysid	System ID Peripheral					
		control_slave	Avalon Memory Mapped Slave	sys_clk	0x000438f0	0x000438f7		
✓		altpll_0	Avalon ALTPLL	sys_clk				
		pll_slave	Avalon Memory Mapped Slave	clk_0	0x000438a0	0x000438af		
✓		onchip_memory2_0	On-Chip Memory (RAM or ROM)					
		s1	Avalon Memory Mapped Slave	sys_clk	0x00020000	0x0003ffff		
✓		jtag_uart_0	JTAG UART					
		avalon_jtag_slave	Avalon Memory Mapped Slave	sys_clk	0x000438e0	0x000438e7		
✓		cpu_0	Nios II Processor					
		instruction_master	Avalon Memory Mapped Master	sys_clk				
		data_master	Avalon Memory Mapped Master		IRQ 0	IRQ		
		jtag_debug_module	Avalon Memory Mapped Slave		0x00042800	0x00042fff		
✓		timer_0	Interval Timer					
		s1	Avalon Memory Mapped Slave	sys_clk	0x00043880	0x0004389f		
✓		pio_system_info	PIO (Parallel I/O)					
		s1	Avalon Memory Mapped Slave	sys_clk	0x000438b0	0x000438bf		
✓		pio_wdi	PIO (Parallel I/O)					
		s1	Avalon Memory Mapped Slave	sys_clk	0x000438d0	0x000438df		
✓		pio_debug_wave	PIO (Parallel I/O)					
		s1	Avalon Memory Mapped Slave	sys_clk	0x000438c0	0x000438cf		
✓		avs_i2c_master_sens	avs_i2c_master					
		control	Avalon Memory Mapped Slave	sys_clk	0x000438e8	0x000438ef		
		protocol	Avalon Memory Mapped Slave		0x00043000	0x000433ff		
		result	Avalon Memory Mapped Slave		0x00043400	0x000437ff		
✓		avs_eth_0	avs_eth					
		mms_tse	Avalon Memory Mapped Slave	sys_clk	0x00040000	0x00040fff		
		mms_reg	Avalon Memory Mapped Slave		0x00043800	0x0004383f		
		mms_ram	Avalon Memory Mapped Slave		0x00041000	0x00041fff		
✓		avs_util_heater_0	avs_util_heater					
		reg	Avalon Memory Mapped Slave	sys_clk	0x00043840	0x0004387f		

avs_util_heater - avs_util_heater_0

avs_util_heater
avs_util_heater

Parameters

g_nof_mac4:	32
g_pipeline:	1
g_nof_ram:	1
g_nof_logic:	10

```
Warning: cpu_0.data_master/avs_i2c_master_sens.protocol: avs_i2c_master_sens.protocol does not have byteenables. Narrow (less than 32-bit) writes from cpu_0.data_master will result in spurious writes to avs_i2c_master_sens.protocol
Warning: cpu_0.data_master/avs_i2c_master_sens.result: avs_i2c_master_sens.result does not have byteenables. Narrow (less than 32-bit) writes from cpu_0.data_master will result in spurious writes to avs_i2c_master_sens.result
Info: pio_system_info: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
```

Figure 3: SOPC Builder system sopc_base with the heater module

4.2 Synthesis

4.2.1 Design

The example unb_base design with the sopc_base SOPC system including the heater can run on all of the 8 FPGA nodes of the UniBoard. Appendix 8 list the Quartus II project and settings files for synthesizing the unb_base design.

4.2.2 Module

There is also a Quartus II project for synthesizing the avs_util_heater.vhd without pin assignments. This is useful to quickly verify the effect of the generics on the synthesis result.

4.3 Software main

The sopc_base SOPC Builder system of Figure 3 also contains a Nios II microprocessor. Therefore the software that runs on the Nios II determines how the unb_base example design behaves. The unb_base_heater/main() example from Appendix 8 can run on the design. Other software can also run on the design, it is not necessary to synthesize the logic again when the software has changed. The compiled software image is loaded into the on chip memory component that is connected to the Nios II in Figure 3.

4.4 Known issues

There are no known issues with the heater module.

5 Design

5.1 Architecture

Figure 4 shows the block diagram of the heater module. Each MAC4 heater element contains 4 multipliers that get their input from a set of PRSG sources. The output of the heater element is narrowed down to a single bit signal that can be read via the MM register. Using the output ensures that the heater element will not get optimized away by synthesis.

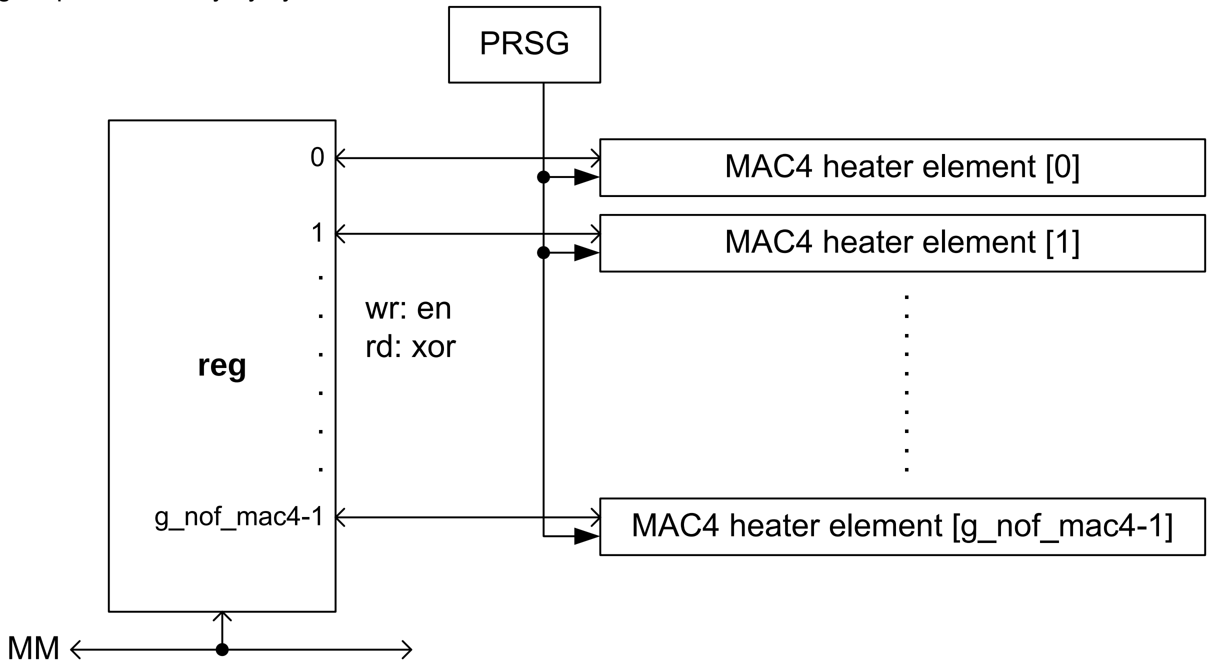


Figure 4: Top level block diagram of the heater module

6 Implementation

6.1 Control

The control register in Figure 4 uses the common_reg_r_w.vhd component from the $UNB common library [4].

The enable bits written to the control register are put across the mm_clk domain to the st_clk domain shown in Figure 2 by passing each bit via a common_async.vhd synchronization component. Similar the corresponding XOR bits that can be read via the control register are put across from the st_clk domain to the mm_clk domain via a common_async.vhd synchronization component. It is not necessary to use common_reg_cross_domain.vhd, see [5], because the register bits are used independently (i.e. not interpreted as numbers > 1).

6.2 Heater element

Figure 5 shows the block diagram of a heater element as used in Figure 4. The MAC4 uses 4 18x18 multipliers, the pipeline uses flip flops (or RAM blocks), the FIFO uses RAM blocks and the logic uses LUTs and flip flops, so together with the number of heater elements this allows the heater module to use an arbitrary mix of internal FPGA resources. The configuration is done via generics as defined in Table 4.

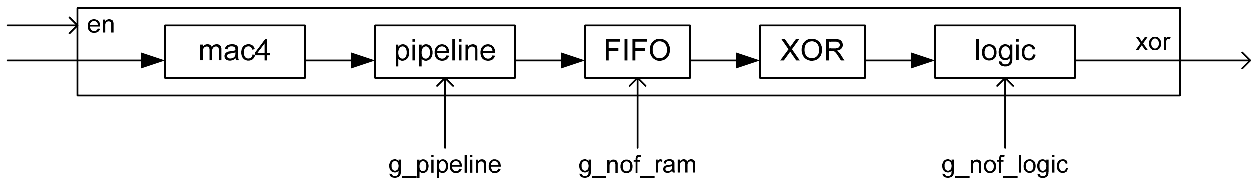


Figure 5: Block diagram of the heater element

The MAC4 block uses the RTL implementation of common_mult_add4.vhd. The generics for the common_mult_add4 are set such that the MAC4 maps efficiently on the Stratix IV DSP blocks [1]. The pipeline block uses the common_pipeline.vhd. For larger g_pipeline settings the synthesis tool may implement this logic in RAM blocks. The FIFO block uses the common_fifo_sc.vhd, Typically g_nof_ram should be a power of 2, possibly because of the internal (Gray-encoded) addressing in the FIFO. The logic block uses util_logic.vhd. The difference between common_pipeline and util_logic is that in util_logic each register stage input is the XOR of its own feed back output and the output from the preceding register stage. This is necessary to avoid that the synthesis tool may implement the logic in RAM blocks. The final XOR is used to reduce the multi-bit vector signal into a single bit element output signal and is available as a function in the common(pkg).vhd. The common components come from the common library [4].

7 Verification

7.1 Simulation

7.1.1 Test bench for the heater module

The tb_util_heater is a VHDL test bench to verify the util_heater module in Modelsim. The test bench uses the procedures listed in Table 8 to more easily access the MM register in the heater module.

Item	Description
proc_mm_access()	Used by proc_util_heater_wr() and proc_util_heater_rd() to do the low level control for a write or a read access.
proc_util_heater_wr()	Write a word to an address in the heater register.
proc_util_heater_rd()	Read a word from an address in the heater register.

Table 8: Heater register access procedures

The p_mm_stimuli process in the VHDL test bench of Figure 6 uses the procedures from Table 8 to access the MM register in the heater module. The test bench is not self checking. The proper behaviour of the internal heater elements needs to be observed in the Modelsim Wave window.

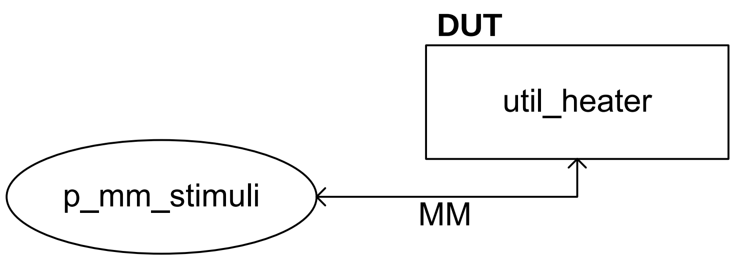


Figure 6: Architecture of tb_util_heater to verify the heater module

7.1.2 Test bench for the unb_base SOPC design with the heater module

By running the unb_base_heater/main.c program from Appendix 8 on the Nios II it is possible to run a design with the heater module as Nios II peripheral in simulation. This main() program calls the heater control function from Table 3 to switch the heater elements on or off at regular intervals. The VHDL test bench tb_unb_base can be used to verify the unb_base design in simulation, see [6] for more information.

7.2 Target hardware

By enabling more or less heater elements in a design that runs on the 8 FPGAs of the UniBoard it is possible to verify the power consumption for different loads. By changing the ST clock frequency (≤ the maximum clock frequency constraint that was set for synthesis, see Appendix 8) it is possible to verify the power consumption in relation to the clock frequency.

The power consumption can be derived from the voltage and current readings of the 48 V power supply or by reading the measurements from the I^2C hot swap controller sensor via UniBoard back node 3.

8 Appendix: List of files

Not all files for the heater module and the unb_base example design in the SVN repository [4] are listed in this section, only the top level files, packages, test benches and the project files. For more general information on how to use the files see [6].

8.1 Firmware VHDL

8.1.1 Heater module

The heater module hardware files are kept at: $UNB/Firmware/modules/util

File	Description
util_heater.vhd	Heater module implementation of Figure 4 and Figure 5
tb_util_heater.vhd	Test bench of Figure 6 to verify the heater module in the Modelsim Wave window
avs_util_heater.vhd	Heater module wrapper for standard Avalon Interface IO [3].
avs_util_heater_hw.tcl	Hardware description file to make the module available in SOPC builder

Table 9: Source files for the heater module

8.1.2 Heater example design

The unb_base design serves as example design for the heater module and is kept at:

$UNB/Firmware/designs/unb_base/

File	Description
sopc_base.sopc	SOPC Builder system of Figure 3
unb_base.vhd	Example SOPC design with the heater module
tb_unb_base.vhd	VHDL test bench for unb_base.vhd.

Table 10: Source files for the unb_base example design that uses the heater module

8.2 Software C, H

8.2.1 Module

The heater module software files are kept at: $UNB/Firmware/software/modules/src

File	Description
avs_util_heater_regs.h	Constants and macros to MM access the heater module.
avs_util_heater.h	Public functions to control the heater module.
avs_util_heater.c	Implements for avs_util_heater.h

Table 11: Module software C, H files

8.2.2 Main

The heater module software main files are kept at: $UNB/Firmware/software/apps

File	Description
unb_base_heater/main.c	Enable or disable the heater elements.

Table 12: Application software main C files

8.3 Simulation

The simulation project files are located in:

$UNB/Firmware/modules/util/build/sim/modelsim/ → for heater module util_lib library.
$UNB/Firmware/designs/unb_base/build/synth/quartus/sopc_base_sim/ → for the unb_base design.

File	Description
util.mpf	Modelsim project file that builds the util_lib module library and provides the simulation configurations for the heater module test benches of Table 9.
unb_base.mpf	Modelsim project file for the unb_base design. Builds the unb_base work library and provides the simulation configurations for the test bench.

Table 13: Simulation project files

8.4 Synthesis

The synthesis files are kept at:

$UNB/Firmware/modules/util/build/synth/quartus → for heater module.
$UNB/Firmware/designs/unb_base/build/synth/quartus/ → for the unb_base design.

File	Description
avs_util_heater.qpf, qsf	Quartus II Project and Settings File for avs_util_heater.
unb_base.qpf, qsf	Quartus II Project and Settings File for unb_base

Table 14: Synthesis project and settings files