



Macrocad Development Inc.

VCM 8015 I2C Synthesizable Model

Description:

The VCM 8015 model from Macrocad is a synthesizable behavior HDL for creating a serial interface controller based on the I²C bus specification. A synchronous parallel interface supports Intel and Motorola interface protocols. Implementations are made easy for both FPGAs and ASICs. Synchronous design and small module size assures worry free synthesis. Hierarchical state machine design is easily modified. Well commented code provides insight into operations. Digital filters to synchronize I²C signals are programmable. Bi-directional signals are contained in the buffer (shell) level, the core contains unidirectional signals only.

Features:

- ◆ Synthesizable RTL HDL code
- ◆ Available in Verilog and VHDL versions
- ◆ Synchronous design
- ◆ Digital signal filtering
- ◆ Well commented code for clarity
- ◆ Test bench is included
- ◆ 100KHz and 400KHz supported
- ◆ Supports High Speed mode, 3.4Mbs
- ◆ 10 bit addressing supported
- ◆ Master and slave
- ◆ 16 Byte buffer for reduced interrupts
- ◆ Designed by hardware engineers for hardware engineers
- ◆ Approx. 7k gates (asic gates)
- ◆ Supports I2C Version 2.1 Spec.

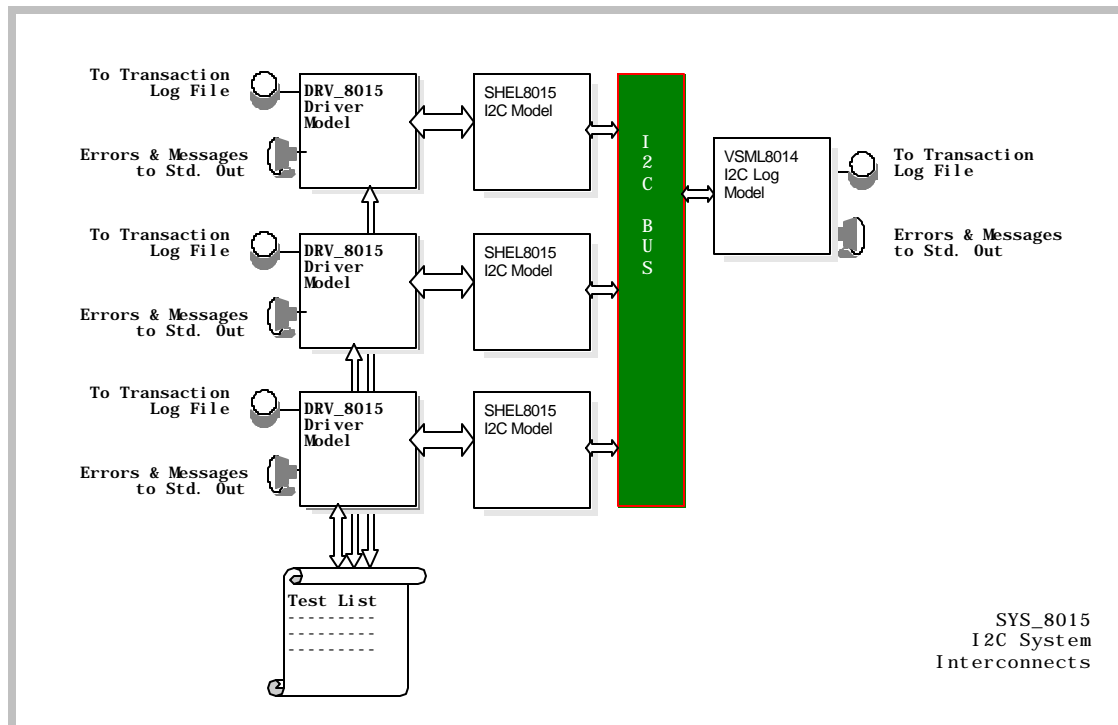


figure 1

Figure 1 shows the development environment for the I2C bus.



Architecture

The SHEL8015 is an interface controller model capable of interfacing the I2C serial bus with an 8 bit parallel programming bus. The top level is an optional bi-directional shell which contains the only bi-directional and tristate signals in the model. The next level is the core level which contains the 5 functional blocks.

Pin List

Figure 2 shows the core8015 signal pins.

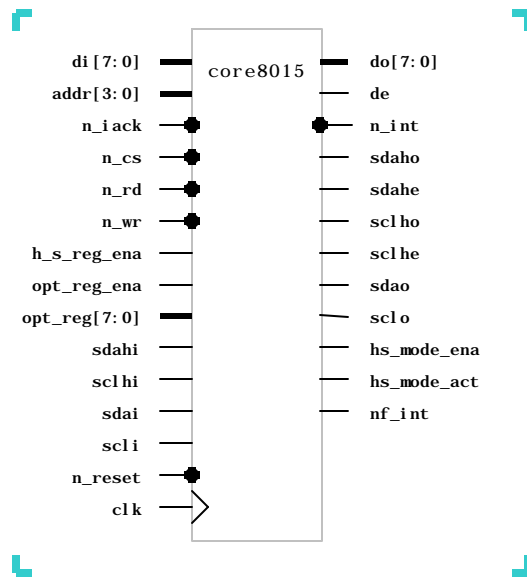


figure 2



Table 1 lists the shel8015 and core8015 signal pins.

SHELL	CORE	TYPE	DESCRIPTION
<i>db7</i>		BI-DIRECT	Programming data bit 7
<i>db6</i>		BI-DIRECT	Programming data bit 6
<i>db5</i>		BI-DIRECT	Programming data bit 5
<i>db4</i>		BI-DIRECT	Programming data bit 4
<i>db3</i>		BI-DIRECT	Programming data bit 3
<i>db2</i>		BI-DIRECT	Programming data bit 2
<i>db1</i>		BI-DIRECT	Programming data bit 1
<i>db0</i>		BI-DIRECT	Programming data bit 0
	<i>do</i> [7:0]	OUT	Programming data bus output
	<i>di</i> [7:0]	IN	Programming data bus input
	<i>de</i>	OUT	Read enable for data bus <not>
	<i>hs_mode_ena</i>	OUT	High/speed mode enabled
	<i>hs_mode_act</i>	OUT	High/speed mode active
	<i>nf_int</i>	OUT	Interrupt high/speed transfer
	<i>h_s_reg_ena</i>	IN	Enables high speed register set, <always 1>
	<i>opt_reg_ena</i>	IN	Enables option register, <always 1>
	<i>opt_reg</i> [7:0]	IN	Option register, <always 0x00>
<i>sdah</i>		BI-DIRECT	High/speed i2c data
	<i>sdaho</i>	OUT	High/speed i2c data output
	<i>sdahe</i>	OUT	High/speed i2c data output enable
	<i>sdahi</i>	IN	High/speed i2c data input
<i>sclh</i>		BI-DIRECT	High/speed i2c clock
	<i>sclho</i>	OUT	High/speed i2c clock output
	<i>sclhe</i>	OUT	High/speed i2c clock output enable
	<i>sclhi</i>	IN	High/speed i2c clock input
<i>sda</i>		BI-DIRECT	Normal/fast i2c data
	<i>sdao</i>	OUT	Normal/fast i2c data output
	<i>sdai</i>	IN	Normal/fast i2c data input
<i>scl</i>		BI-DIRECT	Normal/fast i2c clock
	<i>scl o</i>	OUT	Normal/fast i2c clock output
	<i>scl i</i>	IN	Normal/fast i2c clock input
<i>n_int</i>	<i>n_int</i>	OUT	Interrupt
<i>n_iack</i>	<i>n_iack</i>	IN	Interrupt acknowledge
<i>a0[3:0]</i>	<i>a0[3:0]</i>	IN	Address <register selects>
<i>n_cs</i>	<i>n_cs</i>	IN	Chip select
<i>n_rd</i>	<i>n_rd</i>	IN	Programming register read <active low>
<i>n_wr</i>	<i>n_wr</i>	IN	Programming register write enable <active low>
<i>clk</i>	<i>clk</i>	IN	Master clock
<i>n_reset</i>	<i>n_reset</i>	IN	Reset

table 1



Functional Blocks:

The core logic for the VCM8015 is made up of 6 functional blocks:

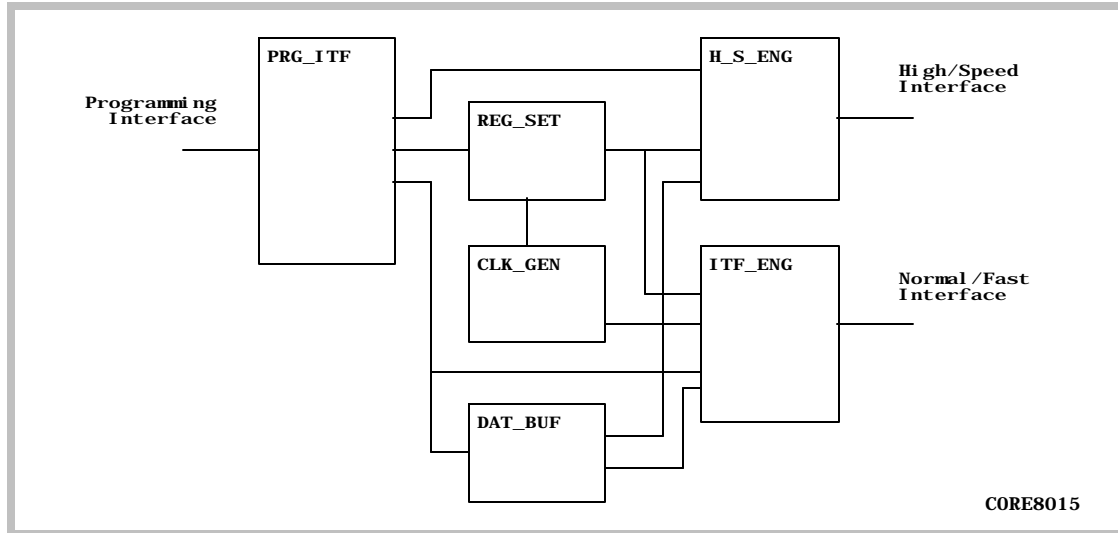


figure 3

PRG_ITF - Parallel Interface

This module interfaces the parallel programming bus to the internal registers and control logic. It includes the register access handshake and normal/fast interrupt logic. Minimum access time is 3 clocks. The active part of the access (n_cs active and either n_rd or n_wr active) is a minimum of 2 clocks. There is a 1 clock minimum recovery between register access cycles.

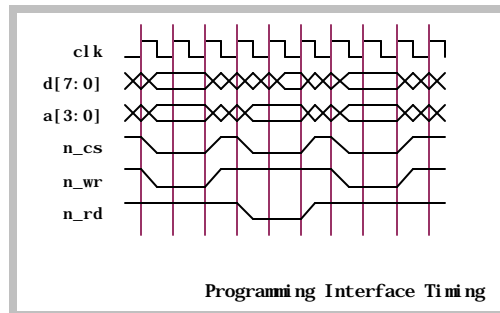


figure 4

REG_SET - Register Set

This module contains 9 of the 16 registers, namely the static registers whose contents can only be modified through the programming interface. These are the S0a, S1c, S2, S3, S8, Sa, Sd, Se, and Sf registers.

CLK_GEN - Clock Generator

This module generates the intermediate clock for the normal/fast internal tracking and timing logic. It is nominally 1.5MHz. It can accept 12, 8, 6, 4.33, or 3MHz input clock frequencies. The clock



select register must be programmed properly to generate the nominal 1.5MHz clock from these input frequencies.

ITF_ENG - N/F I2C Interface Engine

This module drives the normal/fast I2C serial bus. It contains 4 of the 16 registers, (S1s, S0d, S6 and S7), the status, and data registers. It tracks the bus, and takes care of arbitration and error reporting. It also generates the I2C signals if it is participating in a I2C bus transaction. It can monitor I2C bus activity without participating, can act as a master slave or both. It responds to general calls, and can pace the bus when needed.

H_S_ENG - H/S I2C Interface Engine

This module drives the high/speed I2C serial bus. It monitors the status from the itf_eng normal/fast serial interface to detect the migration to a high/speed transaction. The S8, Sb, and Sc registers are contained in this module, (receive data, status and control respectively). It supports automatic selection of both 7 and 10 bit addresses, supports the start byte, and general call.

DAT_BUF – The data buffer provides 16 bytes of dual port RAM which can reduce interrupt traffic in the system. Up to 16 bytes can be written to the I2C controller in either normal/fast, or high/speed modes.



Register Mapping:

The access to the register utilizes 4 address bits. This access is accomplished by the PRG_ITF module. The registers are split between the upper 8 for the enhanced functions, including 10 bit address support, buffer support and high/speed operations, and the lower 8 for the basic I2C functions.

<i>A[3:0]</i>	<i>N_IACK</i>	<i>OP</i>	<i>MODE</i>	<i>NAME</i>	<i>REGISTER DESCRIPTION</i>
F	1	R	H/S	Sf	1 us TIMER VALUE
E	1	W/R	H/S	Se	OPTIONS AND 10 BIT ADDRESS REGISTER
D	1	W/R	H/S	Sd	I2C SIGNAL FILTER REGISTER
C	1	W/R	H/S	Sc	CONTROL REGISTER
B	1	R	H/S	Sb	STATUS REGISTER
A	1	W/R	H/S	Sa	I2C SIGNAL SPEED REGISTER
9	1	R	H/S	S9	RECEIVE DATA REGISTER
8	1	W/R	H/S	S8	TRANSMIT DATA REGISTER
7	1	R	N/F	S7	TRANSMIT DATA REGISTER
6	1	R	N/F	S6	RECEIVE DATA REGISTER
5	1	R	N/F	S1s	STATUS REGISTER
4	1	W/R	N/F	S0a	OWN ADDRESS REGISTER
3	1	W/R	N/F	S3	INTERRUPT VECTOR
2	1	W/R	N/F	S2	CLOCK CONTROL REGISTER
1	1	W/R	N/F	S1c	CONTROL REGISTER
0	1	R	N/F	S0d	RECEIVE DATA REGISTER
0	1	W	N/F	S0d	TRANSMIT DATA REGISTER
-	0	R	N/F	S3	INTERRUPT VECTOR

table 2



Normal/Fast Register Description:

The normal/fast mode registers are similar to the VCM8010 registers. They are accessed in a simpler way using 4 address pins. These registers control and monitor the normal/fast I2C transactions.

REGISTER	BIT	DESCRIPTION
S7		N/F transmit data
	7:0	Serial I2C transmit data
S6		N/F receive data
	7:0	Serial I2C received data
S1s		N/F I2C status register
	7	PIN - state of PIN bit (S1c bit 7) <active low>
	6	RDY - I2C bus is ready <active low>
	5	STS - I2C stop detected in slave mode
	4	BER - I2C bus error detected
	3	LRB - last received bit <AAS = 0>
		OR
		ADO - general call <AAS = 1>
	2	AAS - addressed as slave
	1	LAB - lost arbitration bit
	0	BSY - I2C bus is busy <active low>
S0a		N/F Own address register (used for slave selection)
	7	RESERVED
	6:0	Slave address
S3		N/F Interrupt vector value
	7:0	Interrupt vector
S2		N/F Clock select register
	7:5	RESERVED
	4:2	111 : Divide the main clock by 8 110 : Divide the main clock by 5 101 : Divide the main clock by 4 100 : Divide the main clock by 3 0-- : Divide the main clock by 2
	1:0	11 : Divide the intermediate clock by 1024 10 : Divide the intermediate clock by 128 01 : Divide the intermediate clock by 32 00 : Divide the intermediate clock by 16
S1c		N/F Control register
	7	PIN - Pending interrupt not <active low>
	6	ES0 - Enable N/F I2C bus participation
	5	ES1 - RESERVED
	4	ES2 - RESERVED
	3	ENI - Enable N/F I2C interrupt
	2	STA - Initiate N/F I2C Start
	1	STO - Initiate N/F I2C Stop
	0	ACK - Respond with acknowledge
S0d		N/F Transmit/receive data register
	7:0	Transmit and receive I2C data <S7 on write, S6 on read>

table 3



High/Speed Register Description:

The high/speed registers are accessed using the same 4 address pins. These registers control and monitor the high/speed I2C transactions. The register definition is shown in table 4 below.

REGISTER	BIT	DESCRIPTION
<i>Sf</i>		1 us TIMER VALUE
<i>Se</i>	7:0	Number of main clocks required for 1 us period
		OPTIONS AND 10 BIT ADDRESS REGISTER
<i>Sd</i>	7:6	RESERVED
	5	Buffer empty
	4	High Speed stall at end of transfer enabled
	3	Enable buffer
	2:0	Upper 3 bits of 10 bit address for selection as slave
		I2C OPTION AND SIGNAL FILTER REGISTER
<i>Sc</i>	7	Strict 8584 compliance mode
	6	Enable reset of the N/F busy bit <s1s[0]> if s1c bit 6 is 0
	5	Enable reset of the N/F busy bit <s1s[0]> by writing s1c bit 6 to 0
	4	Enable Normal/Fast bus pacing
	3:2	I2C data signal input filter
	1:0	I2C clock signal input data filter
		CONTROL REGISTER
<i>Sb</i>	7	Reset High/Speed state machines
	6	Enable High/Speed mode
	5	Interrupt High/Speed transfer
	4	Enable ack response for general call
	3	Force I2C bus release from stall
	2	Initiate I2C start or restart process
	1	Initiate I2C stop process
	0	Enable ack response
<i>Sa</i>		STATUS REGISTER
	7	High/Speed mode frame is active
	6	This device is master of this High/Speed frame
	5	This device selected as slave on High/Speed bus
	4	High/Speed enabled
	3	Transaction complete, bus stalled, service interrupt
	2	Normal/Fast interrupt <scl low for 1 us>
	1	Master access in progress
0	State of ack on last I2C transaction	
<i>S9</i>		I2C SIGNAL SPEED REGISTER
	7:4	High/Speed I2C clock signal rate
<i>S8</i>	3:0	Global High/Speed I2C signal rates
		RECEIVE DATA REGISTER
<i>S7</i>	7:0	Data received on the I2C bus
		TRANSMIT DATA REGISTER
<i>S6</i>	7:0	Data to transmit on the high speed I2C bus

table 4



Support Models:

The SYS_8015 model is the top level interconnect model. It contains the 3 SHEL8015 I2C controller models, their DRV_8015 drivers, the test list to be executed, and the VSML8014 I2C bus transaction logger and timing checker. The SHEL8015 models are driven by the DRV_8015 driver models. Stimulus is provided by execution of the test instruction list. Each instruction in the test list specifies the device to be accessed as a part of a sequence of register accesses, similar to software driver routines. The three SHEL8015 models transact data, two as transmitters, and one as a receiver. All of this is controlled via the test instruction list.

VSML8014

The I2C bus logger model tracks the I2C bus, both high/speed and normal/fast. All I2C bus transactions can be logged to a file. This feature is optional. I2C timing is checked for normal, (default) or fast timing specifications. To check for fast timing, a select integer needs to be set to 1. This is the `i_speed` integer in the VSML8014 model. This integer is set to its default value at compile time, but can be changed thereafter. High/Speed timing is checked without user intervention. The VSML8014 model tracks all I2C transactions, so it detects when high/speed mode is enabled and enables high speed timing checks. The VSML8014 model instantiates two separate models for these timing checks, one for normal/fast (VSMT8014) and one for high/speed, (VSMH8014).

DRV_8015

The DRV_8015 driver model provides an interface between the test instruction list, and the signal pins of the SHEL8015 model. This model interprets the test instruction which takes the form of a request passed to the driver as an input vector. The driver interprets the input vector and then executes the registers access which has been requested. This is done by executing a sequence of steps which drive the signals of the SHEL8015. Upon completion of the requested register access, the driver responds with an acknowledge output vector. At this point, the next test instruction in the list will be executed. In this way, a series of test instruction sequences can be built up into a more complex test. Access to the devices can be interleaved to make specific I2C bus conditions, like multi-master arbitration simplified.

The formats for the request and acknowledge vectors are shown below.



VECTOR	RANGE	DESCRIPTION	
<i>Req</i>	63:0	DRIVER REQUEST VECTOR	
	63:56	Tag - request / response handshake	
	55:48	Control - type of register access	
	55:49	RESERVED	
	48	1 : write 0 : read	
	47:40	Address	
	47:44	RESERVED	
	43:40	Address - register selects	
	39:32	Data	
	31:24	Mask - mask data bits	
	23:16	Limit limit - upper limit on repeated accesses	
	15:8	Loop interval - time between repeated access	
	7:0	Check	
	7:1	RESERVED	
	0	1 : check for data errors 0 : Ignore received data	
	<i>Rsp</i>	63:0	DRIVER RESPONSE VECTOR
		63:56	Tag - tag request plus 1
55:48		Control <request mirrored>	
47:40		Address <request mirrored>	
39:32		Data on sh18015 data bus signal pins	
31:24		Mask <request mirrored>	
23:16		Limit limit <request mirrored>	
15:8		Loop interval <request mirrored>	
7:0	Check <request mirrored>		

table 5



Programming and Transfer Operations:

The I2C bus protocol should be thoroughly understood prior to programming the 8015 tests. The VCM8015 register set can then be accessed in certain sequences to initiate and respond to I2C activity and transfers. The models used in the I2C system simulation have operating parameters which are compile time constants. There is a file included at the top level, (SYS_8015) model which sets the operating parameters for the simulation. Operator parameters are listed below.

<i>Model</i>	<i>Parameter</i>	<i>Range</i>	<i>Description</i>
<i>sys_8015</i>	<i>p_debug_level</i>	15:0	debug option switches these can turn on or off error reporting, other operational messaging, and stop on error options
<i>sys_8015</i>	<i>p_test_mask</i>	string	test switches each bit will turn on or off an individual test
<i>sys_8015</i>	<i>p_max_sim_time</i>	integer	maximum time trap, used to avoid 'run-away' simulations
<i>sys_8015</i>	<i>p_test_dir</i>	string	test directory
<i>sys_8015</i>	<i>p_clk_period</i>	integer	clock period time for main clock input to shel8015
<i>drv_8015</i>	<i>p_debug_level</i>	15:0	debug option switches these can turn on or off error reporting, other operational messaging, and stop on error options
<i>drv_8015</i>	<i>p_log_enable</i>	integer	Enables the logging of I2C data transactions
<i>drv_8015</i>	<i>p_log_file_name</i>	string	Log file name
<i>drv_8015</i>	<i>p_8010_enable</i>	integer	Enables 8010 (1 bit) addressing option <set to 0>
<i>vsml 8014</i>	<i>p_debug_level</i>	15:0	debug option switches these can turn on or off error reporting, other operational messaging, and stop on error options
<i>vsml 8014</i>	<i>p_speed</i>	integer	1 : 400khz i2c operation 0 : 100khz i2c operation

table 6

The following chart shows a typical sequence which will drive the I2C bus to gain ownership, then make the transition to high/speed mode. The test list is comprised of calls to the "io_acc" routine. Eight values are passed to this routine. The first indicates the driver to be accessed. The second indicates a write or a read register access operation. The third value determines which register in the device is to be accessed. The fourth value is the data to be written, or the expected data to be read. The fifth value is the mask value. It has no meaning on writes, but on reads it will indicate the bits to be ignored, (1 = ignore, 0 = check against actual data). The sixth value is the loop limit. The value will show how many read retries should be executed before posting an error. If the data compares, then the "io_acc" routine will complete prior to the "loop_limit" number of reads. The seventh value indicates the interval between read retries. If the eighth value is set to 1, the expected data will be checked against the actual data.

In the test below, first both device 1 and device 2 are programmed to operate on the I2C bus. This includes setting the slave address registers, clock select registers, etc.



Instruction	Description
<code>io_acc(1, p_wr, p_s0a, x"05", x"FF", 1, 3, 1);</code>	set up device 1 as master
<code>io_acc(1, p_wr, p_s2 , x"1c", x"FF", 1, 2, 1);</code>	set own address 05
<code>io_acc(1, p_wr, p_s3 , x"3c", x"FF", 1, 4, 1);</code>	set clock reg 12MHz
<code>io_acc(1, p_wr, p_s1c, x"40", x"FF", 1, 1, 1);</code>	set vector reg 3c
<code>io_acc(1, p_wr, p_sa , x"11", x"FF", 1, 1, 1);</code>	Participate on bus
<code>io_acc(1, p_wr, p_sd , x"00", x"FF", 1, 1, 1);</code>	set high speed clk granularity
	set high speed filters
	set up device 2 as slave
<code>io_acc(2, p_wr, p_s0a, x"13", x"FF", 1, 3, 1);</code>	set own address 13
<code>io_acc(2, p_wr, p_se , x"02", x"FF", 1, 3, 1);</code>	set own 10 bit address 2
<code>io_acc(2, p_wr, p_s2 , x"1c", x"FF", 1, 2, 1);</code>	set clock reg 12MHz
<code>io_acc(2, p_wr, p_s3 , x"3c", x"FF", 1, 4, 1);</code>	set vector reg 3c
<code>io_acc(2, p_wr, p_s1c, x"41", x"FF", 1, 1, 1);</code>	Participate on bus
<code>io_acc(2, p_wr, p_sa , x"11", x"FF", 1, 1, 1);</code>	set high speed clk granularity
<code>io_acc(2, p_wr, p_sd , x"00", x"FF", 1, 1, 1);</code>	set high speed filters
<code>io_acc(2, p_wr, p_s0d, x"09", x"FF", 1, 1, 1);</code>	set high speed clk granularity
<code>io_acc(2, p_wr, p_sc , x"41", x"FF", 1, 1, 1);</code>	Enable h/s participation

table 7a

After set up, an I2C bus idle state must be detected before proceeding. In order to start a normal/fast I2C frame, the address to be transmitted must be written first to the data register, then the start command can be executed. At this point, the test sequence waits for the slave to indicate that the first byte (high speed code) has been completed. This is done by looping (reading) on the status, looking for the PIN bit (7) to be 0.

Instruction	Description
<code>io_acc(1, p_rd, p_s1s, x"01", x"fe", 60, 60, 1);</code>	Wait til bus is idle Wait til idle
<code>io_acc(1, p_wr, p_s0d, x"08", x"FF", 1, 1, 1);</code>	Access slave
<code>io_acc(1, p_wr, p_s1c, x"44", x"FF", 1, 1, 1);</code>	Slave controller select START
<code>io_acc(2, p_rd, p_s1s, x"00", x"7f", 120, 120, 1);</code>	Slave response to master's access
<code>io_acc(2, p_rd, p_s0d, x"08", x"00", 1, 1, 1);</code>	Wait til PIN cleared Slave data

table 7b

The I2C bus should be in high speed mode now. In high speed mode, the command is always written to the Sc register first. Then the data is written to S8 which will be transmitted on the I2C bus.

Note that the high speed registers are now being used to control the shel8015 I2C controller models. The same basic procedure is used, the command and data are written, then the completion of the transmission is detected by reading the status register. Here a 10 bit address is used.



Instruction	Description
	HIGH SPEED MODE
	Access slave
<code>i o_acc(1, p_wr, p_sc , x"44", x"FF", 1, 1, 1);</code>	START
<code>i o_acc(1, p_wr, p_s8 , x"f4", x"FF", 1, 1, 1);</code>	Slave select 10 bit addr
	Slave reponse to master's access
<code>i o_acc(1, p_rd, p_sb , x"00", x"7f", 3, 3, 1);</code>	Wait til PIN cleared
<code>i o_acc(2, p_rd, p_s9 , x"f4", x"00", 1, 1, 1);</code>	Slave data

table 7c

The second portion of the address (lower 7 bits) is now transmitted, and a selection occurs in the slave (device 2). This will put device 2 in a state of being selected as a slave.

Instruction	Description
	Master initiates I2C transfer
<code>i o_acc(1, p_wr, p_s8 , x"26", x"FF", 1, 1, 1);</code>	Slave select 7 bit addr
	Slave reponse to master's access
<code>i o_acc(1, p_rd, p_sb , x"00", x"7f", 3, 3, 1);</code>	Wait til PIN cleared
<code>i o_acc(2, p_rd, p_s9 , x"26", x"00", 1, 1, 1);</code>	Slave data

table 7d

Device 1 now transmits the hex value 0xb0 to device 2. Once the transmission is complete, (detected by reading the status register), device 1 begins to transmit another byte. Since the I2C bus is still stalled, (by the slave, device 2), this will not result in a loss of data. The next byte (0xb0) will not be transmitted until the first byte is read from device 2, thereby removing the stall condition on the bus allowing the second byte to be transmitted.

Instruction	Description
	HIGH SPEED SELECTED NOW
	Master initiates I2C transfer
<code>i o_acc(1, p_wr, p_s8 , x"b0", x"FF", 1, 1, 1);</code>	Write location
	Slave reponse to master's access
<code>i o_acc(1, p_rd, p_sb , x"00", x"7f", 3, 3, 1);</code>	Wait til PIN cleared
	Master initiates I2C transfer
<code>i o_acc(1, p_wr, p_s8 , x"0b", x"FF", 1, 1, 1);</code>	Write location - release bus
<code>i o_acc(2, p_rd, p_s9 , x"b0", x"00", 1, 1, 1);</code>	Slave data - release bus

table 7e

Once the second byte is transmitted, device 1 is instructed to execute a stop on the high speed I2C bus. This also results in a stop being executed on the normal fast I2C bus. Now both busses should be in the idle state.



Instruction	Description
<code>io_acc(1, p_rd, p_sb , x"00", x"7f", 3, 3, 1);</code>	Slave reponse to master's access
<code>io_acc(2, p_rd, p_s9 , x"0b", x"00", 1, 1, 1);</code>	Wait til PIN cleared Slave data
<code>io_acc(1, p_wr, p_sc , x"42", x"FF", 1, 1, 1);</code>	MASTER STOPS STOP
	NORMAL SPEED MODE
<code>io_acc(1, p_rd, p_sb , x"00", x"7f", 10, 10, 1);</code>	Slave reponse to master's access
<code>io_acc(1, p_rd, p_s1s, x"81", x"7e", 10, 10, 1);</code>	Look for idle Look for idle

table 7f