**CORE/SEGMENT SPARTAN COMMANDS –V2\_0, 12th Jul 2012**

The following commands shall be available to the core and segment modules

# CMD TYPE FUNCTION Status

9 LW stores the received stream from xport to sram on the sc card, can be used for bitstream completed

10 SR transmits the contents of sram from a start pointer to an end pointer to the xport completed

11 NW programs the flash from bitstream held in sram not implemented

12 LW writes 6 bytes to register (3 bytes start pointer/3 bytes stop pointer) in cmd 10 not applicable

13 SR reads the 6 registers in 12 above not applicable

14 SR reads status registers (6 bytes) completed

15 SR sram memory check not implemented

16 NW loads the sram from bitstream held in flash not implemented

18 NW serial load operations of the V2PRO bitstream not implemented

19 SR reads the temperatures completed

21 NW parallel load operations of the V2PRO bitstream completed

The following command shall be available to the core only

# CMD TYPE FUNCTION Status

17 NW enable/disable the vertex clk completed

20 NW shut down power completed

30 NW selects access to segment module via core\_xport or seg\_xport completed

40 NW selects internal/external 100MHz ADC clock completed

COMMAND FORMATS

**cmd 9/LW/stores the received stream from xport to sram on the sc card, can be used for bitstream**

LW to segment.

This is a command that can be used to store a LW to the SRAM. One use of this command is to send the vertex bitstream from the user pc to be stored in flash ram. A separate document describes the format that the bitstream must take.

general format as follows:

Segment A0,zz,yy,xx,B0,09, + 6 bytes padding data 00,00,00,00,00,00

Core 20,zz,yy,xx,2C,09, + 6 bytes padding data 00,00,00,00,00,00

* (xxyyzz-8) hex bytes of payload data
* (xxyyzz-8) hex bytes of payload data

The padding bytes are used such that an immediately following SW, SR or cmd 12 LW does not overwrite the payload data.

**cmd 10/SR/ transmits the contents of sram from a start pointer to an end pointer to the xport**

|  |  |
| --- | --- |
| segment | C0,00,00,04,D0,0A,00,00 |
| core | 40,00,00,04,4C,0A,00,00 |

**cmd 11/NW/programs the flash from bitstream held in sram**

|  |  |
| --- | --- |
| segment | 80,00,00,04,90,0B,X,00 |
| core | 00,00,00,04,0C,0B,X,00 |

where X (binary) = “00000000” programs flash 0 ic X (binary) = “00000001” programs flash 1 ic

**cmd 12/LW/writes 6 bytes to register (3 bytes start pointer/3 bytes stop pointer) in cmd 10**

(segment) A0,00,00,08,B0,0C,reg\_ff,reg\_ee,reg\_dd,reg\_cc,reg\_bb,reg\_aa,

(core) 20,00,00,08,2C,0C,reg\_ff,reg\_ee,reg\_dd,reg\_cc,reg\_bb,reg\_aa,

Where:

reg\_ff most significant byte of stop pointer

reg\_ee middle byte of stop pointer

reg\_dd most significant byte of stop pointer

reg\_cc most significant byte of start pointer

reg\_bb middle byte of start pointer

reg\_aa least significant byte of start pointer

\* This command may overwrite the sram. This command reads memory bytes from between the pointers and writes them to address 0 upwards this is likely to overwrite the original sram contents. (On the slow control module SRAM is only available for the vhdl, not for programmer’s memory storage).

**cmd 13/SR/reads the 6 registers in cmd 12**

|  |  |
| --- | --- |
| (segment) | C0,00,00,04,D0,0D,00,00 |
| (core) | 40,00,00,04,4C,0D,00,00 |

return bytes (on success) as follows (see cmd 12 for key)

(segment) C0,00,00,08,D0,0D, reg\_ff,reg\_ee,reg\_dd,reg\_cc,reg\_bb,reg\_aa

(core) 40,00,00,08,4C,0D, reg\_ff,reg\_ee,reg\_dd,reg\_cc,reg\_bb,reg\_aa

**cmd 14/SR/Reads status registers (6 bytes)**

note – this command resets the register that counts the number of watchdog timeouts (the current value is readout by this command)

(segment) C0,00,00,04,D0,0E,00,00

(core) 40,00,00,04,4C,0E,00,00

return bytes (on success) as follows

(segment) C0,00,00,08,D0,0E, reg0,reg1,re2,reg3,reg4,reg5

(core) 40,00,00,08,4C,0E, reg0,reg1,re2,reg3,reg4,reg5

Bytes reg0 thru reg5 indicate the following:

Reg0 –

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Bit no. | Active | Indication |  | |  |  |
|  | HI/Lo |  |  | |  |  |
| 0 | HI | Vertex clk status ( hi = enabled, lo = not enabled, pwr up default = | | | not enabled) | |
| 1 | HI | Core module clk source ( hi = internal , lo = external, pwr up default = | | | | external) |
| 2 | HI | Core Psu status monitor (not available in segment module) | | | | |
| 3 | HI | Segment Psu status monitor | | | | |
| 4 | HI | Module\_present\_bit – (for core module, hi=seg\_present, low = seg not present,  for seg module, hi=core\_present, low = core not present) | | | | |
| 5 | HI | Seg\_xport\_sel\_pin – hi = segment module is accessed by seg\_xport (for use with samwise) | | | | |
| 6 |  | Not assigned | |  |  |  |
| 7 |  | Not assigned | |  |  |  |

reg1 – not used

reg2 – for debug use

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Bit no. | Active | Indication |  | |  |  |
|  | HI/Lo |  |  | |  |  |
| 0 | HI | config\_rdwr\_b\_pin, controls the direction of the virtex programming data bus | | | | |
| 1 | HI | Spartan done pin | | | |  |
| 2 |  | Not assigned | | | |  |
| 3 |  | Not assigned | | | | |
| 4 |  | Not assigned | | | | |
| 5 |  | Not assigned | | | | |
| 6 |  | Not assigned | |  |  |  |
| 7 |  | Not assigned | |  |  |  |

Reg3 – for debug use

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Bit no. | Active | Indication |  | |  |  |
|  | HI/Lo |  |  | |  |  |
| 0 | HI | Seg1 Virtex Done | | | | |
| 1 | HI | Seg2 Virtex Done | | | |  |
| 2 | HI | Seg3 Virtex Done/Core Virtex Done | | | |  |
| 3 | HI | Seg4 Virtex Done/NA in core module | | | | |
| 4 | HI | Seg1 Virtex Echo Done | | | | |
| 5 | HI | Seg2 Virtex Echo Done | | | | |
| 6 | HI | Seg3 Virtex Echo Done/Core Virtex Echo Done | |  |  |  |
| 7 | HI | Seg4 Virtex Echo Done/NA in core module | |  |  |  |

Reg4 – for debug use

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Bit no. | Active | Indication |  | |  |  |
|  | HI/Lo |  |  | |  |  |
| 0 | HI | Seg1 Virtex Busy | | | | |
| 1 | HI | Seg2 Virtex Busy | | | |  |
| 2 | HI | Seg3 Virtex Busy /Core Virtex Busy | | | |  |
| 3 | HI | Seg4 Virtex Busy /NA in core module | | | | |
| 4 | HI | Seg1 Virtex Init B | | | | |
| 5 | HI | Seg2 Virtex Init B | | | | |
| 6 | HI | Seg3 Virtex Init B /Core Virtex Init B | |  |  |  |
| 7 | HI | Seg4 Virtex Init B /NA in core module | |  |  |  |

reg5 - CORE/SEGMENT code identification and version number.

This byte is coded as follows:

Bits 0 thru’6 – this is the current version number of the MCS that the fgpa is loaded with (0-127 values possible) Bit 7 - indicates which module type the code targets (1= CORE module, 0 = SEGMENT module)

**cmd 15/SR/SRAM memory check**

(segment) C0,00,00,04,D0,0F,00,00

(core) 40,00,00,04,4C,0F,00,00

cmd frame returned:

segment C0,00,00,05,D0,0F + 3 bytes sram address (last good sram address written/read - 1F FF FF indicates success)

core 40,00,00,05,4C,0F + 3 bytes sram address (last good sram address written/read - 1F FF FF indicates success)

**cmd 16/NW/loads the sram from bitstream held in flash**

segment 80,00,00,04,90,10,0X,00

core 00,00,00,04,0C,10,0X,00

where X(binary) = “00000000” loads sram from flash 0 ic X(binary) = “00000001” loads sram from flash 1 ic

first flash byte of virtex bitstream at flash location = 0x000008 copied to address 0x000008 of sram last flash byte of virtex bitstream = 0x161B33 copied to address 0x161B33 of sram

**cmd 17/NW/enable/disable the vertex clk**

|  |  |
| --- | --- |
| segment | 80,00,00,04,90,11,X,00 |
| core | 00,00,00,04,0C,11,X,00 |

where X (binary) = “00000000” disables the 100MHz clk on the ADC card X (binary) = “00000001” enables the 100MHz clk on the ADC card

**cmd 18/NW/ serial load operations of the V2PRO bitstream**

segment 80,00,00,04,90,12,aa,bb

core 00,00,00,04,0C,12,aa,bb

where byte aa bits defined as follows: (1 in bit location = enable, any combination of bits allowed).

To load a serial bitstream a logic HI is placed in the corresponding bit position below). Any or all of the adc cards may be selected for programming. Programming of all selected cards will be carried out simultaneously

|  |  |  |  |
| --- | --- | --- | --- |
| Bit no. Action Core Module | | Action Segment Module | |
| 0 | | Load bitstream from serial prom core adc card | | Load bitstream from serial prom segment adc card 1 | |
| 1 | | Load bitstream from serial prom segment adc card 5 | | Load bitstream from serial prom segment adc card 2 | |
|  | |  | |  | |
| 2 | | Load bitstream from serial prom segment adc card 6 | | Load bitstream from serial prom segment adc card 3 | |
|  | |  | |  | |
| 3 | |  | | Load bitstream from serial prom segment adc card 4 | |
|  | |  | |  | |
| 4 | |  | |  | |
|  | |  | |  | |
| 5 | |  | |  | |
|  | |  | |  | |
| 6 | |  | |  | |
|  | |  | |  | |
| 7 | |  | |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **cmd 19/SR/ reads the temperatures** | | |  |  |
| (segment) | C0,00,00,04,D0,13,00,00 | |  |  |
| (core) | 40,00,00,04,4C,13,00,00 | |  |  |
| the returned message = header + 20 bytes payload data has the following format: | | |  |  |
| header |  |  |  |  |
| segment: | C0,00,00,16,D0,13 |  |  |  |
| core: | 40,00,00,16,4C,13 |  |  |  |
| payload |  |  |  |  |

20 bytes of payload data as follows: header + payload byte 0 (PL0) + (PL1) …. (PL19)

assignment of payload bytes:

core:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| byte order | | Reading sensor: | byte order | | Reading sensor: |
| lsb | msb |  | lsb | msb |  |
| PL1 | PL0 | Seg1 virtex | PL11 | PL10 | Core analog |
| PL3 | PL2 | Seg1 analog | PL13 | PL12 | Psu 0 |
| PL5 | PL4 | Seg2 virtex | PL15 | PL14 | Psu 1 |
| PL7 | PL6 | Seg2 analog | PL17 | PL16 | Psu 2 |
| PL9 | PL8 | Core virtex | PL19 | PL18 | Not assigned |
|  |  |  |  |  |  |

segment:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| byte order | | Reading sensor: | byte order | | Reading sensor: |
| lsb | msb |  | lsb | msb |  |
| PL1 | PL0 | Seg1 virtex | PL11 | PL10 | Seg3 analog |
| PL3 | PL2 | Seg1 analog | PL13 | PL12 | Seg4 virtex |
| PL5 | PL4 | Seg2 virtex | PL15 | PL14 | Seg4 analog |
| PL7 | PL6 | Seg2 analog | PL17 | PL16 | Psu 1 |
| PL9 | PL8 | Seg3 virtex | PL19 | PL18 | Psu 2 |
|  |  |  |  |  |  |

The bit order within the two bytes used for each reading is as follows:

|  |  |  |
| --- | --- | --- |
| d0,d1,d2 | - ignore |  |
| d3 | := lsb data thru to d14 msb (12 bit resolution, 1 bit | = 0.0625 degC) |
| d15 | := sign bit (1 = below zero celsius) |  |

Note: reading the temperature sensors on a core or segment module will clear the temperature status bits (cmd 14)

**cmd 20/NW/ shut down power**

|  |  |
| --- | --- |
| segment | 80,00,00,04,90,14,X,00 |
| core | 00,00,00,04,0C,14,X,00 |

where X (binary) = d7 thru d0

d0 - currently unassigned

d1 - currently unassigned

d2 - currently unassigned

d3 - external control shut down power NOW!!

Note - written values of d0 thru d2 are read back through the status registers bits 4,5,6 of reg3 of cmd 14. (legacy)

**cmd 21/NW/ parallel load operations of the V2PRO bitstream**

segment 80,00,00,04,90,15,aa,00

core 00,00,00,04,0C,15,aa,00

where aa is the Virtex that is going to be programmed as per the table below (parallel programming is possible):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Bit no. | Active | Indication |  | |  |  |
|  | HI/Lo |  |  | |  |  |
| 0 | HI | Seg1 Virtex | | | | |
| 1 | HI | Seg2 Virtex | | | |  |
| 2 | HI | Seg3 Virtex /Core Virtex | | | |  |
| 3 | HI | Seg4 Virtex /NA in core module | | | | |
| 4 |  | not assigned | | | | |
| 5 |  | not assigned | | | | |
| 6 |  | not assigned | |  |  |  |
| 7 |  | not assigned | |  |  |  |

**cmd 30/NW/ selects access to segment module via core\_xport or seg\_xport**

core 00,00,00,04,0C,1E,X,00

x = “00000000” selects core\_xport for accessing the segment module,

x = “00000001” selects seg\_xport for accessing the segment module.

**cmd 40/NW/ selects internal/external 100MHz ADC clock**

core 00,00,00,04,0C,28,X,00

x = “00000000” enables external clk,

x = “00000001” enables internal clk