RDBE FPGA HAL Interface

Doc No: X3C 2009_10_13_3
TODO: Add appropriate MIT document number

Document history

Change date	Changed by	Version	Notes
09-07-21 09:12	Mikael Taveniku	PA1	New document
09/08/27	Mikael Taveniku	PA2	Included DBE Commands
			Added RDBE_server implementation description
13/10/9	Mikael Taveniku	PA1	Split document into multiple pieces

Table of Contents

1 OVERVIEW

This document describes the developer access software interface to the FPGA for the RDBE software running on the Roach board. Theses functions are preliminary and will be included in the rdbe_devloper_interface document and the rdbe_fpga_hal document.

2 RDBE FPGA HAL

The FPGA on the RDBE board exports a set of functions that are collected in an library file. These functions are simply here to abstract the intricacies of the hardware implementation of the functionality in the FPGA from the programmer of the external and internal interfaces.

Command	Parameters	Description	Comment

This is a simplistic hardware abstraction layer implemented as a set of “C” library functions.

3 TOP LEVEL CONTROL REGISTERS

```
/**
 * Set the value of fpga control register regNo [0..3]
 * @param regNo - the register to access
 * @param value - the value to set
 * @return - 0 if successful
 */
int set_fpga_control_reg( int regNo, unsigned short value);

/**
 * Read the value of a specific control register, return it in value
 * @param regNo - the register to read [0..3]
 * @param value - return value of the function
 * @return - 0 if success
 */
int get_fpga_control_reg( int regNo, unsigned short *value);

/*
-----------------------------------------------------------------------------------
 * Bit field definitions for control register 0
 * Bit 0 – Arm – synchronizes to the external 1 pps.
 * Bit 1 – adc 3 wire reload, reloads the control data for to the ADC.
 * Bit 2 – ctrl reset, reset to the ADC board.
 * Bit 3 – Enable FIR filter 1, enable the FIR filter module
 * Bit 4 – Enable FFT, enables the FFT module
 * Bit 5 – dcm reset. Resets the DCM use to for managing the ADC clock.
 * Bit 6 – Enable Interpolator, enables the Interpolator module
 * Bit 7 – Enable Mk5BInterface,  enables the Mk5B interface module
 *
-----------------------------------------------------------------------------------
 */

/**
 * Arm the fpga, -- synchronize on next 1pps
 * @return - 0
```

```c
 */
int fpga_arm(void);

/**
 * clear fpga arm flag
 * @return - 0 on success
 */
int fpga_arm_clear(void);

/**
 * reload ADC control registers
 * @return
 */
int fpga_adc_reload(void);


/**
 * Clear ADC reload flag
 * @return - 0 on success
 */
int fpga_adc_reload_clear(void);

/**
 * Reset the ADC block
 * @return - 0 on success
 */
int fpga_adc_reset(void);

/**
 * Clear ADC reset flag
 * @return
 */
int fpga_adc_reset_clear(void);

/**
 * Enable the FIR filter block
 * @return - 0 on success
 */
int fpga_fir_enable(void);

/**
 * Disable FIR filter block
 * @return - 0 on success
 */
int fpga_fir_disable(void);

/**
 * Enable the FFT block
 * @return - 0 on success
 */
int fpga_fft_enable(void);

/**
 * Disable FFT block
 * @return
 */
int fpga_fft_disable(void);
```

```
/**
 * Reset the DCM
 * @return
 */
int fpga_dcm_reset(void);

/**
 * Clear reset flag for DCM
 * @return
 */
int fpga_dcm_reset_clear(void);

/**
 * Enable the Interpolator block
 * @return
 */
int fpga_interpolator_enable(void);

/**
 * Disable interpolator block
 * @return
 */
int fpga_interpolator_disable(void);
/**
 * Enable the MK5 Interface
 * @return - 0 on success
 */
int fpga_mk5_enable(void);

/**
 * Disable MK5 Interface
 * @return - 0 on success
 */
int fpga_mk5_disable(void);

/**
 * Write the value to the status register regNo [0..3] (most of the time this will
fail)
 * @param regNo - the status register to set
 * @param value - the value to write
 * @return - 0 if successful
 */
int set_fpga_status_reg( int regNo, unsigned short value);

/**
 * Read the status register defined by regNo [0..3], and return it in value
 * @param regNo - the register to read
 * @param value - a place to put the returned value in
 * @return - 0 if successful
 */
int get_fpga_status_reg( int regNo, unsigned short *value);
```

4 SYSTEM REGISTERS IN THE FPGA

```
/**
 * Read the board id, return it in the value
 * @param value - return value
 * @return 0 if successful
 */
int get_board_id(unsigned short *value);

/**
 * set the board id to value (will always fail)
 * @param value - the board id to set
 * @return 0 if successful
 */
int set_board_id(unsigned short value);

/**
 * read the board revision id's
 * @param rev_major - return parameter
 * @param rev_minor - return parameter
 * @param rev_rcs - return parameter
 * @return - 0 if successful
 */
int get_board_revision(unsigned short *rev_major, unsigned short *rev_minor, unsigned
short *rev_rcs);

/**
 * Set the board revision ID -- will always fail
 * @param rev_major
 * @param rev_minor
 * @param rev_rcs
 * @return - 0 if successful
 */
int set_board_revision( unsigned short rev_major, unsigned short rev_minor, unsigned
short rev_rcs);

/**
 * Get the firmware id
 * @param id - return parameter
 * @return - 0 if successful
 */
int get_firmware_id( unsigned short *id);

/**
 * Set the firmware register to id, will always fail
 * @param id - the id to set
 * @return - 0 if successful
 */
int set_firmware_id( unsigned short id);

/**
 * Read the firmware version information
 * @param rev_major - return value
 * @param rev_minor - return value
 * @param rev_rcs - return value
 * @return - 0 if successful
 */
int get_firmware_revision(unsigned short *rev_major, unsigned short *rev_minor,
unsigned short *rev_rcs);

/**
 * Set the firmware version numbers -- will always fail
 * @param rev_major
 * @param rev_minor
 * @param rev_rcs
 * @return - 0 if successful
 */
int set_firmware_revision(unsigned short rev_major, unsigned short rev_minor,
unsigned short rev_rcs);
```

5 QUANTIZER BLOCK AND REGISTERS

```
 /*
 * Threshold setting registers for each channel are 25 bits wide in the FPGA design
 * When these registers are read out by the PPC the 25 bit values are placed
 * into two 16 bit regs with 0 padding on the lower 7 bits to preserve sign value.
 * Thus, the PPC must read two memory locations and shift contents by 7 to right to
 * form correct value. These are signed fixed point values (15,-16) as represented in
a 32 bit register.
 * Read only for now. But should implement the routines to do the writes also.
 *
-----------------------------------------------------------------------------------
 */

/**
 * Get Quantizer threshold value as 25bit integer sign extended to 32 bits
 * The value is a "real" with the lower 20 bits as fraction
 * @param intFreq - IF [0 or 1]
 * @param channel - Channel [0 .. 15]
 * @param value - the return value
 * @return - 0 on success
 */
int quantizer_get_threshold_plus(int intFreq, int channel, long int *value);

/**
 * Set Quantizer threshold value as 25bit integer sign extended to 32 bits
 * The value is a "real" with the lower 20 bits as fraction
 * @param intFreq - IF [0 or 1]
 * @param channel - Channel [0 .. 15]
 * @param value - the value to set (only low 25 bits are valid
 * @return - 0 on success
 */
int quantizer_set_threshold_plus(int intFreq, int channel, long int value);
```

```
/**
 * Get Quantizer threshold MINUS value as 25bit integer sign extended to 32 bits
 * The value is a "real" with the lower 20 bits as fraction
 * @param intFreq - IF [0 or 1]
 * @param channel - Channel [0 .. 15]
 * @param value - the return value
 * @return - 0 on success
 */
int quantizer_get_threshold_minus(int intFreq, int channel, long int *value);

/**
 * Set Quantizer threshold MINUS value as 25bit integer sign extended to 32 bits
 * The value is a "real" with the lower 20 bits as fraction
 * @param intFreq - IF [0 or 1]
 * @param channel - Channel [0 .. 15]
 * @param value - the value to set (only low 25 bits are valid
 * @return - 0 on success
 */
int quantizer_set_threshold_minus(int intFreq, int channel, long int value);


/*
 * In order to re-quantize the data some state statistics need to be collected for
each channel.
 * These are the number of states counts (high/low) over a given period of time.
 * Only the low bit count is provided since the high count can be calculated by
knowing the number of samples
 * and low bit count. . These are unsigned 16 bit values. Read only.
 */

/**
 * Get the bit state count for IF and Channel
 * signed fractional integer, low 20 bits are fraction
 * @param intFreq - IF [0..1]
 * @param channel - Channel [0..31]
 * @param value - return value
 * @return - 0 on success
 */
int quantizer_get_bit_state_count(int intFreq, int channel, unsigned short *value);

/**
 * Set the bit state count for IF and Channel
 * signed fractional integer, low 20 biits are fraction
 * @param intFreq - IF [0..1]
 * @param channel - Channel [0..31]
 * @param value - return value
 * @return - 0 on success
 */
int quantizer_set_bit_state_count(int intFreq, int channel, unsigned short value);

/*
 * Gain setting registers for each channel are 25 bits wide in the FPGA design
 * For computational purposes, When these same registers are read out by the PPC the
25 bit values are placed
 *  into two 16 bit wide registers with the zero padding on the lower 7 bits to
preserve the sign of the value.
```

```
 *  This implies that the PPC must read two memory locations and shift the contents
by 7 places to the right
 *  to form the correct value. .
 *  These are signed fixed point values (15,-16) as represented in a 32 bit register.
Read only.
 */

/**
 * Get the quantizer gain value
 * This is an signed fractional int with first 20 bits as fraction
 * @param intFreq - IF [0..1]
 * @param channel - Channel [0..31]
 * @param value - return value
 * @return - 0 on success
 */
int quantizer_get_gain(int intFreq, int channel, long int *value);

/**
 * Set the quantizer gain value
 * This is an signed fractional int with first 20 bits as fraction
 * @param intFreq - IF [0..1]
 * @param channel - Channel [0..31]
 * @param value - Gain setting, only first 25 bits are valid
 * @return - 0 on success
 */
int quantizer_set_gain(int intFreq, int channel, long int value);
```

5.1 MISCELANEOUS QUANTIZER REGISTERS.

```
/**
 * Get the total number of samples collected for state count statistics
 * @param value - number of samples collected
 * @return - 0 on success
 */
int quantizer_get_number_samples(unsigned long *value);

/**
 * Write the bit mask 32 bits to enable(1) or disable(0) each quantizer channel
 * @param channels - [0..31 bits]
 * @return - 0 on success
 */
int quantizer_set_enable_table(unsigned long channels);

/**
 * Get the current value of which channels are enabled
 * @param channels - bit-field with 0..31 bits
 * @return - 0 on success
 */
int quantizer_get_enable_table(unsigned long *channels);
```

```c
/**
 * Disable a specific quantizer channel
 * @param channel - [0..31]
 * @return - 0 on success
 */
int quantizer_disable_channel(int channel);

/**
 * Enable a specific quantizer channel
 * @param channel - [0..31]
 * @return - 0 on success
 */
int quantizer_enable_channel(int channel);

/**
 * Get the quantizer status register value
 * (1)indicates that the corresponding channel quantization levels have been
calculated.
 * @param status - the returned status value
 * @return - 0 on success
 */
int quantizer_get_status(unsigned long *status);

/**
 * Write value to quantizer status register -- will always fail
 * (1)indicates that the corresponding channel quantization levels have been
calculated.
 * @param status - the value to write
 * @return - 0 on success
 */
int quantizer_set_status(unsigned long status);

/**
 * Write the value to quantizer control register
 * @param value - the value to set [0 disable, 1 enable]
 * @return - 0 on success
 */
int quantizer_set_control(unsigned short value);

/**
 * Get the quantizer control register value
 * @param value - return parameter [0 disabled, 1 enabled]
 * @return - 0 on success
 */
int quantizer_get_control(unsigned short *value);

/**
 * Get the quantizer failure register values
 * Quantizer Failure Resister indicates that the corresponding channel quantization
levels have
 * been calculated and did not meet the set criteria.
 * @param value - the value bit-field 0..31
 * @return - 0 on success
 */
int quantizer_get_failure(unsigned long *value);

/**
```

```c
 * Write a failure pattern to the quantizer failure register
 * Quantizer Failure Resister indicates that the corresponding channel quantization
levels have
 * been calculated and did not meet the set criteria.
 * @param value - bit-field 0..31
 * @return - 0 on success
 */
int quantizer_set_failure(unsigned long value);

/**
 * Comparator Enable Table Register is a 32 bit register. When a bit is set the
corresponding
 * channels comparators are enabled.
 * @param value - the value bit-field 0..31
 * @return - 0 on success
 */
int quantizer_get_comparator_enable(unsigned long *value);

/**
 * Comparator Enable Table Register is a 32 bit register. When a bit is set the
corresponding
 * channels comparators are enabled.
 * @param value - bit-field 0..31
 * @return - 0 on success
 */
int quantizer_set_comparator_enable(unsigned long value);

/**
 * Get the quantizer state machine status
 * @param value -
 * @return - 0 on success
 */
int quantizer_get_state_machine_status(unsigned short *value);
```

6 MARK V B INTERFACE

```c
/**
 * Get the sync word from MKV
 * @param value - return value
 * @return 0 on success
 */
int mark5_get_sync_word(unsigned long *value);

/**
 * Set the MKV sync word
 * @param value - value to write
 * @return - 0 on success
 */
int mark5_set_sync_word(unsigned long value);
```

```
/**
 * Get the number of years from 2000
 * @param value - return value
 * @return - 0 on success
 */
int mark5_get_years(unsigned short *value);

/**
 * Set the number of years from 2000
 * @param value - return value
 * @return - 0 on success
 */
int mark5_set_years(unsigned short value);

/**
 * Set the T-Bit (only bit 0 is valid)
 * @param value - [0..1]
 * @return - 0 on success
 */
int mark5_set_t_bit(unsigned short value);

/**
 * Get the T-Bit (only bit 0 is valid)
 * @param value - [0..1]
 * @return - 0 on success
 */
int mark5_get_t_bit(unsigned short *value);

/**
 * Set the user specified register value (12 bit)
 * @param value - the value to set
 * @return - 0 on success
 */
int mark5_set_user_reg(unsigned short value);

/**
 * Get the current user register value
 * @param value - return parameter (12 bit)
 * @return - 0 on success
 */
int mark5_get_user_reg(unsigned short *value);

/**
 * Set the 32 bits of the unassigned register value
 * @param value - the value bits 0..31
 * @return - 0 on success
 */
int mark5_set_unassigned_reg(unsigned long value);


/**
 * Get the 32 bits of the unassigned register value
 * @param value - the returned value bits 0..31
 * @return - 0 on success
 */
int mark5_get_unassigned_reg(unsigned long *value);
```

```
/**
 * Get mark5 state machine status
 * @param value -
 * @return - 0 on success
 */
int mark5_get_state_machine_status(unsigned short *value);
```

7 OPTIMAL INPUT LEVEL DETECTOR REGISTERS

```
/**
 * Get input level detector control value
 * @param value - the control register value
 * @return - 0 on success
 */
int input_level_detector_get_control(unsigned short *value);


/**
 * Set the input level control register to value
 * @param value - the value to write
 * @return - 0 on success
 */
int input_level_detector_set_control(unsigned short value);


/**
 * Get the input level detector status
 * @param status - returned value
 * @return - 0 on success
 */
int input_level_detector_get_status(unsigned short *status);

/**
 * Set the input level detector status
 * @param status - status to set
 * @return - 0 on success
 */
int input_level_detector_set_status(unsigned short status);


/**
 * Get the state count
 * @param cnt - returned value
 * @return - 0 on success
 */
int input_level_detector_get_low_state_count(unsigned short *cnt);

/**
 * Set the state count
 * @param cnt - value to set
 * @return - 0 on success
 */
int input_level_detector_set_low_state_count(unsigned short cnt);

/**
 * Get the state count
 * @param cnt - returned value
 * @return - 0 on success
 */
int input_level_detector_get_mid_state_count(unsigned short *cnt);

/**
 * Set the state count
 * @param cnt - value to set
 * @return - 0 on success
 */
int input_level_detector_set_mid_state_count(unsigned short cnt);

/**
 * Get the state count
 * @param cnt - returned value
 * @return - 0 on success
 */
int input_level_detector_get_high_state_count(unsigned short *cnt);

/**
 * Set the state count
 * @param cnt - value to set
 * @return - 0 on success
 */
int input_level_detector_set_high_state_count(unsigned short cnt);

/**
 * Get the Input Level Detector number of samples
 * @param cnt - returned value
 * @return - 0 on success
 */
int input_level_detector_get_num_samples(unsigned short *cnt);

/**
 * Set the Input Level Detector number of samples
 * @param cnt - number of samples
 * @return - 0 on success
 */
int input_level_detector_set_num_samples(unsigned short cnt);

/**
 * Get the Input Level Detector state machine status
 * @param status - returned value
 * @return - 0 on success
 */
int input_level_detector_state_machine_status(unsigned short *status);
```

8 10 GIGABIT ETHERNET FUNCTIONS

```c
/**
 * Set the local MAC address for the 10GE port
 * @param port - Ethernet port [0..3]
 * @param mac - array of 6 bytes LE format [00:01:02:03:...] = 00:02:02 ..
 * @return - 0 on success
 */
int fpga_10ge_set_local_mac(unsigned port, unsigned char *mac);

/**
 * Get the local MAC address for the 10GE port
 * @param port - Ethernet port [0..3]
 * @param mac - return value - array of 6 bytes LE format [00:01:02:03:...] =
00:02:02 ..
 * @return - 0 on success
 */
int fpga_10ge_get_local_mac(unsigned port, unsigned char *mac);

/**
 * Set the local IP address for the 10GE port
 * @param port - Ethernet port [0..3]
 * @param ip - 1 byte LE format [192, 168, 0,1] = 1
 * @return - 0 on success
 */
int fgpa_10ge_set_local_gw(unsigned port, unsigned char ip);

/**
 * Get the local Gateway address for the 10GE port
 * @param port - Ethernet port [0..3]
 * @param ip - return value - 1 byte with end of ip 192.168.0.1 returns "1"
 * @return - 0 on success
 */
int fpga_10ge_get_local_gw(unsigned port, unsigned char *ip);

/**
 * Set the local Gateway address for the 10GE port
 * @param port - Ethernet port [0..3]
 * @param ip - 1 bytes 192.168.0.1 then ip should be [192,168,0,1]
 * @return - 0 on success
 */
int fpga_10ge_set_local_ip(unsigned port, unsigned char *ip);

/**
 * Get the local IP address for the 10GE port
 * @param port - Ethernet port [0..3]
 * @param ip - return value - array of 4 bytes LE format [192, 168, 0,1] =
192.168.0.1
 * @return - 0 on success
 */
int fpga_10ge_get_local_ip(unsigned port, unsigned char *ip);
```

```c
/**
 * Set the buffer size value for the RX/TX buffer
 * @param port - Ethernet port
 * @param sz - size of buffer (valid data in the buffer)
 * @return - 0 on success
 */
int fpga_10ge_set_buffer_sizes(unsigned port, unsigned short sz);

/**
 * Read the value of valid data in receive buffer
 * @param port - Ethernet port
 * @param sz - valid number of bytes
 * @return - 0 on success
 */
int fpga_10ge_get_buffer_sizes(unsigned port, unsigned short *sz);

/**
 * TODO:
 * @param port
 * @param ports
 * @return
 */
int fpga_10ge_set_valid_ports(unsigned port, unsigned short ports);

/**
 * TODO:
 * @param port
 * @param ports
 * @return
 */
int fpga_10ge_get_valid_ports(unsigned port, unsigned short *ports);

/**
 * Read the status register of the XILINX XAUI
 * @param port - the port to read
 * @param status - see XILINX documentation
 * @return - 0 n success
 */
int fpga_10ge_get_xaui_status(unsigned port, unsigned short *status);

/**
 * TODO:
 * @param port
 * @param config
 * @return
 */
int fpga_10ge_set_phy_config(unsigned port, unsigned short config);

/**
 * TODO:
 * @param port
 * @param config
 * @return
 */
int fpga_10ge_get_phy_config(unsigned port, unsigned short *config);

/**
```

```c
 * Write bytes to the TX buffer on port
 * @param port - the port to write to
 * @param buf - array of bytes to write (2k always)
 * @return - 0 on success
 */
int fpga_10ge_set_tx_buffer(unsigned port, unsigned char *buf);

/**
 * Read data from TX buffer on port
 * @param port - the port to read from
 * @param buf - return array of bytes from buffer (user must allocate 2k)
 * @return - 0 on success
 */
int fpga_10ge_get_tx_buffer(unsigned port, unsigned char *buf);

/**
 * Write bytes to the RX buffer on port
 * @param port - the port to write to (2k always)
 * @param buf - array of bytes to write
 * @return - 0 on success
 */
int fpga_10ge_set_rx_buffer(unsigned port, unsigned char *buf);

/**
 * Write bytes to the TX buffer on port
 * @param port - the port to write to
 * @param buf - return - array of bytes to write 2k user must allocate
 * @return - 0 on success
 */
int fpga_10ge_get_rx_buffer(unsigned port, unsigned char *buf);

/**
 * Write a full APR cahce table to FPGA port
 * @param port - the port [0..3]
 * @param arp_table - 128 entries of 6 bytes each in LE format
 * @return - 0 on success
 */
int fpga_10ge_set_arp_cache(unsigned port, unsigned char *arp_table);

/**
 * Write a full APR cahce table to FPGA port
 * User must allocate return buffer
 * @param port - the port [0..3]
 * @param arp_table - (return value) 128 entries of 6 bytes each in LE format
 * @return - 0 on success
 */

int fpga_10ge_get_arp_cache(unsigned port, unsigned char *arp_table);

/**
 * Get the arp table entry for port, and ip with ending number pos
 * User must allocate return value
 * @param port - Ethernet port [0..3]
 * @param pos - Ending IP address [0..127]
 * @param mac - return value - 6 bytes LE array of byte
 * @return - 0 on success
 */
```

```
int fpga_10ge_get_arp_value(unsigned port, unsigned char pos, unsigned char *mac);

/**
 * Set the arp table entry for port, and ip with ending number pos
 * @param port - Ethernet port [0..3]
 * @param pos - Ending IP address [0..127]
 * @param mac - 6 bytes LE array of byte
 * @return - 0 on success
 */
int fpga_10ge_set_arp_value(unsigned port, unsigned char pos, unsigned char *mac);
```