newton_control

Generated by Doxygen 1.6.1

Tue Feb 11 16:38:59 2020

Contents

Chapter 1

Newton Control Console Application and Newton Control API

1.1 Newton Control Console Application

The purpose of the Newton Control Console Application is to provide a a program for conrtolling and debugging Newton operation via the SPI slave interface.

Also provided is an API for building C programs to control Newton

1.2 Table of Contents

1.3 Command Line Usage

Usage:

- newton_control.exe load target file_name
- newton_control.exe verify target file_name
- newton_control.exe unload target file_name
- newton_control.exe spi_write address data
- newton_control.exe spi_read address
- newton_control.exe reg_write address data
- newton_control.exe reg_read address

- newton_control.exe reg_dump address word_count
- newton_control.exe reg_fill address data word_count
- newton_control.exe console

Where:

- load : Loads the contents of the specified file into the target memory.
- verify : Compares the contents of the specified file to the contents of the target memory.
- unload : Dumps the content of the target memory to the specified file.

- target is one of the following:
  - useq_seq_ram : Microsequencer Sequence RAM
  - useq_map_ram : Microsequencer MAP RAM
  - useq_wave_ram : Microsequencer Wave RAM
  - datapath_ram : Gain Correction RAM
  - de_ram : Dump Engine RAM
  - lps1_ram : LPS1
  - lps2_ram : LPS2
  - hsp_rom : HSP ROM (FPGA only)
  - hsp_ram : HSP RAM (FPGA only)
  - efuse : eFuse (FPGA only)

- spi_write : Write data to the address. For accessing HSP MBOX registers.
- spi_read : Read data at address. For accessing HSP MBOX registers.
- reg_write : Write data to the Newton register at address
- reg_read : Read the Newton register at address
- reg_dump : Read and display word_count Newton registers starting at address.
- reg_fill : Write word_count word_count Newton registers with data starting at address.
- console : Enter console mode

Options:

--help Shows this help message.

1.4 Command Line Examples

Here are a few examples of the command line interface:

- Loading new HSP ROM and eFuse data:
  - newton_control.exe load hsp_rom ../../firmware/ADI_Firmware_Drop_0.4/ADI_HSP_ROM_0.4.4.vhx

  - newton_control.exe load efuse ../../efuse/hsp_fuse_file.vhx

- Writing the HSP mailbox S2H_MBX_VALID register:
  - spi_write 8 1
- Reading the HSP mailbox S2H_MBX_FIFO_CNT register:
  - spi_read a
- Writing the newton microsequencer gprR0 register:
  - reg_write 60 01234
- Reading the newton microsequencer gprR0 register:
  - reg_read 50

1.5 Console Operation

Console mode supports the same commands and arguments at the command line interface.

```
{.c}
FIXME: need new code example
```

See newton_control.h for detailed documentation.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 revisionID_t Struct Reference

Public Attributes

```
• union {
      struct {
          int major:8
          int minor:8
          int patch:8
          int unused:8
      }
      uint32_t value32
  };
```

The documentation for this struct was generated from the following file:

- newton_control.h

Chapter 5

File Documentation

5.1 newton_control.h File Reference

Newton Control API. `#include <stdio.h>`

```
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <time.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>
#include <fcntl.h>
#include "newton_addr_cdef.h"
#include "newton_typedefs.h"
#include "newton_memMap.h"
#include "fpga_backdoor_addr_cdef.h"
#include "fpga_backdoor_typedefs.h"
#include "hsp_regs_addr_cdef.h"
#include "hsp_regs_typedefs.h"
#include "json.h"
#include "cutils/str_parms.h"
#include <wiringPi.h>
```

Classes

- struct revisionID_t

Defines

- #define **ERROR_CODE_BASE** 0x0ad10000
- #define **BIT_RATE_COUNT** 6

Typedefs

- typedef unsigned long long **u64**
- typedef unsigned int **u32**
- typedef unsigned short **u16**
- typedef unsigned char **u08**
- typedef void(∗ **callback_t**)(void)
- typedef struct str_parms **str_parms**
- typedef enum adi_loadTargets adi_loadTargets_e

  Newton RAM Targets.

- typedef enum adi_hspMemBitWidths **adi_hspMemBitWidths_e**

Enumerations

- enum adi_attribute_e { **no_increment** = 1, **increment** = 2, **addr_data_pair** = 4 }

  HSP mailbox attributes.

- enum adi_loadTargets {

  USEQ_SEQ_RAM = 0, **USEQ_MAP_RAM**, **USEQ_WAVE_RAM**, **DATAPATH_RAM**,

  DE_RAM, **LPS1_RAM**, **LPS2_RAM**, **HSP_ROM**,

  HSP_RAM, **EFUSE** }

  Newton RAM Targets.

- enum **adi_hspMemBitWidths** { **HSP_ROM_WIDTH** = 40, **HSP_RAM_WIDTH** = 40, **EFUSE_WIDTH** = 32 }
- enum adi_errorCodes_e {

  ADI_NO_ERROR = 0, ADI_JSON_FILE_NOT_FOUND = 0x0ad10001, ADI_JSON_FILE_OPEN_ERROR = 0x0ad10002, ADI_JSON_PARSE_ERROR = 0x0ad10003,

  ADI_JSON_UNEXPECTED_KEY = 0x0ad10004, ADI_UNEXPECTED_SPI_BYTE_COUNT = 0x0ad10006, ADI_SPI_DRIVER_ERROR = 0x0ad1000a, ADI_SPI_XFER_ERROR = 0x0ad1000b,

  ADI_SPI_BIT_RATE_ERROR = 0x0ad1000c, ADI_BAD_POWER_STATE = 0x0ad10011, ADI_POWER_ON_TIMEOUT = 0x0ad10012, ADI_D32_FILE_NOT_FOUND = 0x0ad100013,

  ADI_D32_FILE_OPEN_ERROR = 0x0ad10014, ADI_MBI_FILE_NOT_FOUND = 0x0ad100015, ADI_MBI_FILE_OPEN_ERROR = 0x0ad10016, ADI_ERROR_CODE_MISSING = 0x0ad10017,

  ADI_UNEXPECTED_ARGS = 0x0ad10018, ADI_BAD_RUNSTALL_STATE = 0x0ad10019, ADI_H2S_VALID_TIMEOUT = 0x0ad10020, ADI_S2H_NOT_VALID_TIMEOUT = 0x0ad10021 }

Error code definitions.

- enum adi_parameter_codes_e {

  REQ_BUF_ADDR = 0xecec0000, **REQ_BUF_SIZE** = 0xecec0001, **REQ_TO_SEND** = 0xecec0002, **ALLOCATE_BUFFER** = 0xecec0003,

  RDY_TO_RECV = 0xecec0004, **START_ADDRESS** = 0xecec0005, **BUFFER_SIZE** = 0xecec0006, **BUFFER_SENT** = 0xecec0007,

  REQ_BUF_ADDR_SIZE = 0xecec0008, **START_ADDRESS_SIZE** = 0xecec0009, **PARAM_VALID** = 0xfafa8000 }

  Enums for specifying the Set / Get Parameter name codes.

- enum adi_data_type_codes_e { **DTYPE_INT32** = 0, **DTYPE_UINT32**, **DTYPE_FLOAT** }

  Enums for specifying the data type codes.

- enum spiBitRates {

  SPI_BIT_RATE_1M = 1000, **SPI_BIT_RATE_2M** = 2000, **SPI_BIT_RATE_4M** = 4000, **SPI_BIT_RATE_8M** = 8000,

  SPI_BIT_RATE_12M = 12000, **SPI_BIT_RATE_16M** = 16000 }

  Enums for specifying valid SPI Clock Rates.

Functions

- int adi_spi_write (int bytes_out, u08 ∗data_out, int bytes_in, u08 ∗data_in)

  Write a 32-bit word to the Newton over the SPI Interface.

- int adi_spi_read_word (u16 address, u16 ∗data)

  Read a 16-bit word from the Newton over the SPI Interface.

- int adi_spi_write_word (u16 address, u16 data)

  Write a 16-bit word to the Newton over the SPI Interface.

- int adi_spi_write_word_multiple (u16 address, int dataLength, u16 ∗dataWritePtr)

  Write multiple 16-bit words to the Newton over the SPI Interface.

- int adi_spi_read_word_multiple (u16 address, int dataLength, u16 ∗dataReadPtr)

  Read multiple 16-bit words from the Newton over the SPI Interface.

- int adi_spi_open (int bitRate)

  Open the SPI Device.

- int adi_spi_close ()

  Close the SPI Device.

- bool adi_is_spi_open ()

  Test if the SPI Device is open.

- int adi_wait_for_hsp_ready ()

  Wait for HSP ready.

- int adi_send_command (u16 command, u16 address, int word_count, adi_attribute_e attribute)

  Send command to HSP.

- int adi_clear_h2s_valid ()

  Clear H2S valid.

- int adi_wait_for_h2s_valid ()

  WWait for H2S Valid from HSP.

- int adi_wait_for_s2h_not_valid ()

  Wait for S2H not valid from HSP.

- int adi_send_data (int word_count, u16 ∗wr_data)

  Send data to HSP.

- int adi_get_data (int word_count, u16 ∗rd_data)

  Get data from HSP.

- int adi_write_register (u16 addr, u16 wr_data)

  Perform register write through HSP.

- int adi_write_burst (u16 addr, u16 word_count, u16 ∗wr_data)

  Perform write burst through HSP with incrementing addresses.

- int adi_write_burst_no_incr (u16 addr, u16 word_count, u16 ∗wr_data)

  Perform write burst through HSP with non-incrementing addresses.

- int adi_read_register (u16 addr, u16 ∗rd_data)

  Perform register read through HSP.

- int adi_read_burst (u16 addr, u16 word_count, u16 ∗rd_data)

  Perform read burst through HSP with incrementing addresses.

- int adi_read_burst_no_incr (u16 addr, u16 word_count, u16 ∗rd_data)

  Perform read burst through HSP with non-incrementing addresses.

- int adi_load_newton_ram (adi_loadTargets_e loadTarget, char ∗fileName)

  Load the Newton memory image contained in the specified file into the specified HSP memory.

- int adi_load_hsp (adi_loadTargets_e loadTarget, char ∗fileName)

  Load the memory image contained in the specified file into the specified HSP memory.

- int adi_verify_hsp (adi_loadTargets_e verifyTarget, char ∗fileName)

  Verify that the memory image contained in the specified file matches the containt of the specified HSP memory.

- int adi_unload_hsp (adi_loadTargets_e unloadTarget, char ∗fileName)

  Unload the memory image to the specified file from the specified HSP memory.

- int adi_soft_reset ()

  Issue a soft reset to the newton.

- int adi_newton_config (int bitRateOverride)

  Configure the newton control program.

- char ∗ adi_error_msg (int returnCode)

  Return error message string for given error code.

Variables

- enum spiBitRates spiBitRates_e

  Enums for specifying valid SPI Clock Rates.

- struct spi_ioc_transfer **global_tr**

5.1.1 Detailed Description

Newton Control API.

5.1.2 Typedef Documentation

5.1.2.1 typedef enum adi_loadTargets adi_loadTargets_e

Newton RAM Targets. Newton RAM targets.

5.1.3 Enumeration Type Documentation

5.1.3.1 enum adi_attribute_e

HSP mailbox attributes. Newton HSP mailbox attributes.

5.1.3.2 enum adi_data_type_codes_e

Enums for specifying the data type codes. Set / Get Parameter data type codes.

5.1.3.3 enum adi_errorCodes_e

Error code definitions. These error codes are used in function return values.

Enumerator:

ADI_NO_ERROR No error.

ADI_JSON_FILE_NOT_FOUND JSON file not found error.

ADI_JSON_FILE_OPEN_ERROR JSON file open error.

ADI_JSON_PARSE_ERROR JSON parsing error.

ADI_JSON_UNEXPECTED_KEY Unexpected JSON Key.

ADI_UNEXPECTED_SPI_BYTE_COUNT Unexpected SPI count.

ADI_SPI_DRIVER_ERROR SPI Driver Error.

ADI_SPI_XFER_ERROR SPI Transfer Error.

ADI_SPI_BIT_RATE_ERROR Unexpected SPI Bit Rate.

ADI_BAD_POWER_STATE Unexpected power state.

ADI_POWER_ON_TIMEOUT Timeout waiting for power state == ON.

ADI_D32_FILE_NOT_FOUND D32 file not found error.

ADI_D32_FILE_OPEN_ERROR D32 file open error.

ADI_MBI_FILE_NOT_FOUND MBI file not found error.

ADI_MBI_FILE_OPEN_ERROR MBI file open error.

ADI_ERROR_CODE_MISSING Missing error code.

ADI_UNEXPECTED_ARGS Unexpected arguments.

ADI_BAD_RUNSTALL_STATE Unexpected RunStall state.

ADI_H2S_VALID_TIMEOUT H2s valid timeout.

ADI_S2H_NOT_VALID_TIMEOUT S2H not valid timeout.

5.1.3.4 enum adi_loadTargets

Newton RAM Targets. Newton RAM targets.

5.1.3.5 enum adi_parameter_codes_e

Enums for specifying the Set / Get Parameter name codes. Set / Get Parameter name codes.

5.1.3.6 enum spiBitRates

Enums for specifying valid SPI Clock Rates. Valid SPI Clock Rates.

5.1.4 Function Documentation

5.1.4.1 int adi_clear_h2s_valid ()

Clear H2S valid. This task clears H2S Valid.

Returns:

status value. Indicates success or failure of the function.

5.1.4.2 char∗ adi_error_msg (int *returnCode*)

Return error message string for given error code.

Parameters:

errorCode error code value.

Returns:

error message string.

5.1.4.3 int adi_get_data (int *word_count*, u16 ∗ *rd_data*)

Get data from HSP. This task gets read data from HSP.

Parameters:

word_count Word Count.

rd_data Read Data Array.

Returns:

status value. Indicates success or failure of the function.

5.1.4.4 bool adi_is_spi_open ()

Test if the SPI Device is open.

Returns:

status value. Indicates success or failure of the function.

5.1.4.5 int adi_load_hsp (adi_loadTargets_e *loadTarget*, char ∗ *fileName*)

Load the memory image contained in the specified file into the specified HSP memory. This function performs the following:

- Loads the memory image contained in the specified file into specified HSP memory. This function is only valid for the newton FPGA.

Parameters:

loadTarget HSP memory to load.

fileName name of the file containing the HSP ROM to be loaded.

Returns:

status value. Indicates success or failure of the function.

5.1.4.6 int adi_load_newton_ram (adi_loadTargets_e *loadTarget*, char ∗ *fileName*)

Load the Newton memory image contained in the specified file into the specified HSP memory. This function performs the following:

- Loads the memory image contained in the specified file into specified Newton memory.

Parameters:

loadTarget Newton memory to load.

fileName name of the file containing the Newton RAM to be loaded.

Returns:

status value. Indicates success or failure of the function.

5.1.4.7 int adi_newton_config (int *bitRateOverride*)

Configure the newton control program.

Parameters:

bitRateOverride use this SPI bit rate instead of the default.

Returns:

status value. Indicates success or failure of the function.

5.1.4.8 int adi_read_burst (u16 *addr*, u16 *word_count*, u16 ∗ *rd_data*)

Perform read burst through HSP with incrementing addresses. This task performs read burst through HSP with incrementing addresses.

Parameters:

addr Read address.

word_count Word Count.

rd_data Read Data Array.

Returns:

status value. Indicates success or failure of the function.

5.1.4.9 int adi_read_burst_no_incr (u16 *addr*, u16 *word_count*, u16 ∗ *rd_data*)

Perform read burst through HSP with non-incrementing addresses. This task performs read burst through HSP with non-incrementing addresses.

Parameters:

addr Read address.

word_count Word Count.

rd_data Read Data Array.

Returns:

status value. Indicates success or failure of the function.

5.1.4.10 int adi_read_register (u16 *addr*, u16 ∗ *rd_data*)

Perform register read through HSP. This task performs register read through HSP.

Parameters:

addr Read address.

rd_data Read Data.

Returns:

status value. Indicates success or failure of the function.

5.1.4.11 int adi_send_command (u16 *command*, u16 *address*, int *word_count*, adi_attribute_e *attribute*)

Send command to HSP. This task sends the command to the HSP.

Parameters:

command Command Code.

address Address.

word_count Word Count.

attribute Attributes.

Returns:

status value. Indicates success or failure of the function.

5.1.4.12 int adi_send_data (int *word_count*, u16 ∗ *wr_data*)

Send data to HSP. This task sends the data to the HSP.

Parameters:

wr_data Write Data Array.

word_count Word Count.

Returns:

status value. Indicates success or failure of the function.

5.1.4.13 int adi_soft_reset ()

Issue a soft reset to the newton. Issue a soft reset.

Parameters:

channel_id the channel ID of the to be written.

Returns:

status value. Indicates success or failure of the function.
status value. Indicates success or failure of the function.

5.1.4.14 int adi_spi_close ()

Close the SPI Device.

Returns:

status value. Indicates success or failure of the function.

5.1.4.15 int adi_spi_open (int *bitRate*)

Open the SPI Device.

Parameters:

bitRate the bit rate that the SPI interface should operate at.

Returns:

status value. Indicates success or failure of the function.

5.1.4.16 int adi_spi_read_word (u16 *address*, u16 ∗ *data*)

Read a 16-bit word from the Newton over the SPI Interface.

Parameters:

address the address of the 32-bit word.

data32 a pointer to the data read from the Newton.

Returns:

status value. Indicates success or failure of the function.

5.1.4.17 int adi_spi_read_word_multiple (u16 *address*, int *dataLength*, u16 ∗ *dataReadPtr*)

Read multiple 16-bit words from the Newton over the SPI Interface.

Parameters:

address the address of the data to be read.

dataLength the number of words to be read.

dataPtr a pointer to the data read from the Newton.

Returns:

status value. Indicates success or failure of the function.

5.1.4.18 int adi_spi_write (int *bytes_out*, u08 ∗ *data_out*, int *bytes_in*, u08 ∗ *data_in*)

Write a 32-bit word to the Newton over the SPI Interface.

Parameters:

bytes_out the number of bytes to be sent over the SPI interface to the Newton.

data_out the data to be sent over the SPI interface to the Newton.

bytes_in the number of bytes to be recevied from the Newton over the SPI interface.

data_in the data to be recevied from the Newton over the SPI interface.

Returns:

status value. Indicates success or failure of the function.

5.1.4.19 int adi_spi_write_word (u16 *address*, u16 *data*)

Write a 16-bit word to the Newton over the SPI Interface.

Parameters:

bytes_out the number of bytes to be sent over the SPI interface to the Newton.

data_out the data to be sent over the SPI interface to the Newton.

bytes_in the number of bytes to be recevied from the Newton over the SPI interface.

data_in the data to be recevied from the Newton over the SPI interface.

Returns:

status value. Indicates success or failure of the function.

5.1.4.20 int adi_spi_write_word_multiple (u16 *address*, int *dataLength*, u16 ∗ *dataWritePtr*)

Write multiple 16-bit words to the Newton over the SPI Interface.

Parameters:

address the address of the data to be written.

dataLength the number of words to be written.

dataPtr a pointer to the data written to the Newton.

Returns:

status value. Indicates success or failure of the function.

5.1.4.21 int adi_unload_hsp (adi_loadTargets_e *unloadTarget*, char ∗ *fileName*)

Unload the memory image to the specified file from the specified HSP memory. This function performs the following:

- Unloads the memory to the specified file from specified HSP memory. This function is only valid for the newton FPGA.

Parameters:

unloadTarget HSP memory to unload.

fileName name of the file containing the HSP ROM to be loaded.

Returns:

status value. Indicates success or failure of the function.

5.1.4.22 int adi_verify_hsp (adi_loadTargets_e *verifyTarget*, char ∗ *fileName*)

Verify that the memory image contained in the specified file matches the containt of the specified HSP memory. This function performs the following:

- Verifies that the memory image contained in the specified file matches the contents of the specified HSP memory. This function is only valid for the newton FPGA.

Parameters:

verifyTarget HSP memory to verify.

fileName name of the file containing the HSP ROM to be verified.

Returns:

status value. Indicates success or failure of the function.

5.1.4.23 int adi_wait_for_h2s_valid ()

WWait for H2S Valid from HSP. This task polls the h2s valid bit until it is set.

Returns:

status value. Indicates success or failure of the function.

5.1.4.24 int adi_wait_for_hsp_ready ()

Wait for HSP ready. This task waits for the HSP to be ready.

Returns:

status value. Indicates success or failure of the function.

5.1.4.25 int adi_wait_for_s2h_not_valid ()

Wait for S2H not valid from HSP. This task polls the S2h valid bit until it is cleared.

Returns:

status value. Indicates success or failure of the function.

5.1.4.26 int adi_write_burst (u16 *addr*, u16 *word_count*, u16 $*$ *wr_data*)

Perform write burst through HSP with incrementing addresses. This task performs write burst through HSP with incrementing addresses.

Parameters:

addr Write address.

word_count Word Count.

wr_data Write Data Array.

Returns:

status value. Indicates success or failure of the function.

5.1.4.27 int adi_write_burst_no_incr (u16 *addr*, u16 *word_count*, u16 $*$ *wr_data*)

Perform write burst through HSP with non-incrementing addresses. This task performs write burst through HSP with non-incrementing addresses.

Parameters:

addr Write address.

word_count Word Count.

wr_data Read Data Array.

Returns:

status value. Indicates success or failure of the function.

5.1.4.28 int adi_write_register (u16 *addr*, u16 *wr_data*)

Perform register write through HSP. This task performs register write through HSP.

Parameters:

addr Write address.

wr_data Write Data.

Returns:

status value. Indicates success or failure of the function.

5.1.5 Variable Documentation

5.1.5.1 enum spiBitRates spiBitRates_e

Enums for specifying valid SPI Clock Rates. Valid SPI Clock Rates.

Index