Bitcoin Double HASH256

User Manual

Author: Yu Peng

Version: 1.0

Change History

Version	Data	Author	Description
1.0	**2013-11-10**	**Yu Peng**	**Initial version**

Table of Contents

1. Overview

This design is a Bitcoin double HASH256 module. It is optimized specially for Bitcoin hashing work, then consume less resource and achieve higher hash rate compare with general HASH256 implementation.

2. Source Code Organization

All the HDL source code is under folder of "trunk\rtl\vhdl".

The top level module is "btc_dsha.vhd", which is in the folder of " trunk\rtl\vhdl\sha256core". This folder also contains all the code related with HASH algorithm.

The folder " trunk\rtl\vhdl\misc" contains some nessecery general purpose modules which are used by the design.

The folder " trunk\rtl\vhdl\TestBench" contains test bench of the design.

3. User Interface

3.1 Interface Signal Descriptions

Name	Direction	Description
iRst_async	In	Asynchronous reset, high effective
iClkReg	In	Clock of register interface
iClkProcess	In	Clock of hash processing and internal control state machine
iValid_p	In	One clock pulse valid for writing control registers, sync with "iClkReg"
ivAddr[3:0]	In	Address of writing register operation, sync with "iClkReg"
ivData[31:0]	In	Data of writing register operation, sync with "iClkReg"
oReachEnd_p	Out	One clock pulse to indicate all nonce has been searched and without any result satisfy the difficulty target, sync with "iClkProcess"
oFoundNonce_p	Out	One clock pulse to indicate one nonce has been found which satisfy the difficulty target, sync with "iClkProcess". At the same time the nonce is sent out through "ovNonce" port
ovNonce[31:0]	Out	When " oFoundNonce" is high. It output the the nonce which satisfy the difficulty target, sync with "iClkProcess"
ovDigest	Out	This port is for simulation purpose, should leave open in actual design

3.2 Theory of Operation

The design use a set of 32 bits register to input initial data and control the internal searching state machine. The detail definition of the registers is specified in section 3.3.

First, you should load the initial data into register 0x1 to 0xC. Then writing 0x00000001 into 0xD to start the searching state machine. The "Start Nonce" and "End Nonce" registers specified the searching range. If a valid nonce is found, the output signal "oFoundNonce" goes to high for one clock cycle, and in the same clock cycle the nonce is sent out through "ovNonce". If all the nonce are calculated, and none of the them satisfied the difficulty, the "oReachEnd" goes to high for one clock cycle to indicate the end of searching.

3.3 Control Register Description

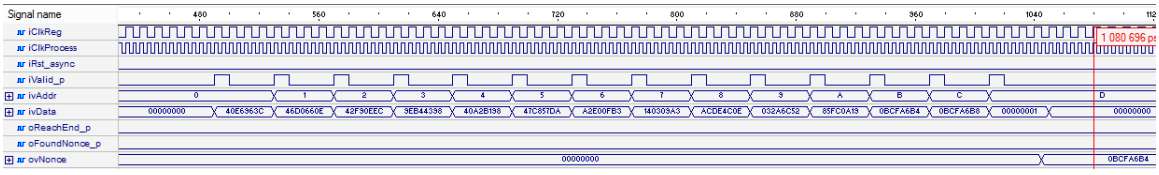
Address	Description
0x0	Doword 0 of Midstate
0x1	Doword 1 of Midstate
0x2	Doword 2 of Midstate
0x3	Doword 3 of Midstate
0x4	Doword 4 of Midstate
0x5	Doword 5 of Midstate
0x6	Doword 6 of Midstate
0x7	Doword 7 of Midstate
0x8	Doword 7 of Merkle Tree Root
0x9	Time Stamp
0xA	Target Bits
0xB	Start Nonce of Searching
0xC	End Nonce of Searching
0xD	Command Register Writing 0x0000001 into this register can start or restart the internal searching state machine.

3.3 Configuration Parameter

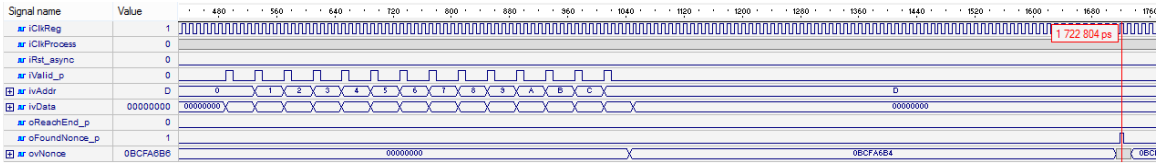
Name	Default Value	Description
gBASE_DELAY	1	This parameter can be set to 1 or 3. It cannot be set to other value. When setting to 1, the design consume less resource but with a lower hash rate. while setting to 3, the design consume more resource with higher hash rate.

4. Simulation Result

4.1 Writing control registers



4.2 Found Nonce



4.3 Reaching End