LZRW1 Encoder Core

Version 1.0

Table of Contents

1 Introduction

This IP core allows losses data compression based on the Lempel-Ziv-Ross-Williamson-1 algorithm [LZRW1]. Its focus is on high throughput (of uncompressed data) at the expense of an somewhat lower compression ratio. One byte of uncompressed data can be processed at every second clock cycle. Data compression is achieved by encoding repeating sections of data as length/offset pairs instead of transmitting the literal itself. To allow a fast recognition of repeating sections in the stream a hash table is used. This saves the time needed to search an entire buffer (typically several kBytes) and therefore greatly increases coding speed.

The core is fully pipelined to allow high clock speeds. 50MHz can easily be achieved on a Spartan6 FPGA. This results in a maximum compression throughput of almost 25MBytes/sec. It uses a Wishbone compliant slave interfaces to receive uncompressed data and configuration information. A second Wishbone master interface is used by the included DMA unit to directly transfer the compressed data to RAM or another Wishbone compliant slave.

2 Algorithm

The core uses the LZRW1 algorithm published by Ross Williams. For a detailed description please see the original document in [LZRW1]. The basic idea is that both, encoder and decoder, keep a part (4kB in this implementation) of the most recent data which has been en- or decoded. Each byte of the data-stream is either transmitted directly (a so called literal) or as part of 3 to 16 bytes long copy item. The copy item is transmitted as offset/length pair. The offset is a pointer into the stream history. A copy item is transmitted as two bytes (4 bits for length and 12 bits for the offset) and therefor reduces the total amount of data i.e. compression is achieved. However one additional bit is needed for each item to identify it as literal or copy item. To simplify decoding on byte oriented PC systems 8 items are grouped into one frame and all identication bits are grouped into one byte. Note that the original algorithm designed by Ross Williams groups 16 items into one frame. Consequently a frame is somewhere between 9 bytes (1 header + 8 literals) and 17 (1 header + 16 copy items) long. As the shortest copy item is three bytes long only 14 out of the 16 possible codes of the 4 bits long length field are used. The codes 0 and 1 are reserved as control codes. So far only the code with length=0 and offset=0 is used as end-of-data symbol. It marks the end of an encoded data stream.

Note that the core implements a slightly modified version of the LZRW1 algorithm to simplify the implementation in hardware. The difference is that in the core the hash table is update for every new byte of uncompressed data while in the original algorithm this update is skipped for bytes belonging to a repeated section. While this yields different compressed data streams for the same uncompressed data it can still be decompressed correctly.

This is necessary to make the hash table operation independent of the comparator which detects matches between the new and previous data. As the decoder is place later in the pipeline this removes a feedback path and therefore allows much faster clock speeds.

3 Wishbone Interfaces

The core implements two separate Wishbone interface for data and control transfer. One of them is a slave, the other a master. The slave interface allows access to configuration registers and to send data to the core through the input FIFO. The master interface is used by the included DMA unit to write compressed data to RAM or to send it to some other kind of slave device.

3.1 Slave Interface

The slave interface's IO lines are listed in the following table.

Name	Direction	Type
SlCycxSI	in	std_logic;
SlStbxSI	in	std_logic;
SlWexSI	in	std_logic;
SlSelxDI	in	std_logic_vector(3 downto 0);
SlAdrxDI	in	std_logic_vector(4 downto 2);
SlDatxDI	in	std_logic_vector(31 downto 0);
SlDatxDO	out	std_logic_vector(31 downto 0);
SlAckxSO	out	std_logic;
SlErrxSO	out	std_logic;
IntxSO	out	std_logic;

In the table above all but the last signal belong to the Wishbone interface. The last one (IntxSO) is the core interrupt request line. It is high when the condition set by the interrupt configuration register (see below) is fulfilled. Note that the interface supports 32bit access only. Any access where SlSelxDI is not equal to “1111” will be terminated by the error signal (SlErrxSO).

The following table list all registers which can be accessed through the Wishbone slave interface. All registers are 32 bits wide and can be read and written unless noted otherwise.

Name	SlAdrxDI	Num. Adr	Description
DataIn	“000”	0	Data input into input FIFO which will be compressed. (write only)
CfgStat	“001”	4	Configuration and status register
InFifoThr	“010”	8	Input FIFO interrupt thresholds
OutFifoThr	“011”	12	Output FIFO interrupt thresholds
DmaDest	“100”	16	DMA destination address
DmaLen	“101”	20	Max DMA transfer length in bytes
FifoLen	“110”	24	Number of Bytes in Input and Output FIFOs (read only)
Command	“111”	28	Command register (write only)

DataIn Register

Data written to this register will be transfered to the input FIFO and in turn be compressed. The byte written to the core at SlDatxDI(7 downto 0) is written to the FIFO first, the byte on SlDatxDI(31 downto 24) last. This is commonly referred to as little endian encoding.

Reading the register retuns x”00000000”.

CfgStat Register

This is the cores central configuration and status register. Unused bits will be read as zero and must be written as zero. Writing the register will clear the status flags in bits 24..29. However the FIFO full and empty flags might be set again if the condition is still met.

Bit #	Name	R/W	Description
0..2			*unused*
3	DmaBusy	R	'1' indicates that the DMA unit is busy (ie DmaLen is not zero)
4..7			*unused*
8	IncDestAdr	R/W	If set, DMA unit will write to successively increasing addresses.
9..15			*unused*
16	IEInFifoEmpty	R/W	Enable input FIFO empty interrupt
17	IEInFifoFull	R/W	Enable input FIFO full interrupt
18	IEOutFifoEmpty	R/W	Enable output FIFO emtpy interrupt
19	IEOutFifoFull	R/W	Enable output FIFO full interrupt
20	IEDmaErr	R/W	Enable DMA error interrupt
21	IECoreDone	R/W	Enable core done interrupt
22..23			*unused*
24	InFifoEmtpy	R	Input FIFO empty interrupt flag
25	InFifoFull	R	Input FIFO full interrupt flag
26	OutFifoEmpty	R	Output FIFO empty interrupt flag
27	OutFifoFull	R	Output FIFO full interrupt flag
28	DmaErr	R	DMA Error Flag (set when DMA transfer is terminated by Err signal)
29	CoreDone	R	Set when all data has been compressed (after FLUSH command was sent)
30..31			*unused*

InFifoThr Register

This register configures the full (bits 31..16) and empty (bits 15..0) interrupt thresholds for the input FIFO. When the number of bytes in the input FIFO is below the empty threshold or is greater or equal the full threshold the InFifoEmpty or InFifoFull flags are set. Writing this register clears both flags.

OutFifoThr Register

This register configures the full (bits 31..16) and empty (bits 15..0) interrupt thresholds for the output FIFO. When the number of bytes in the output FIFO is below the empty threshold or is greater or equal the full threshold the OutFifoEmpty or OutFifoFull flags are set. Writing this register clears both flags.

DmaDest Register

This register sets the destination address at which the DMA unit will write the compressed data. Note that this register can only be written when the DmaLen register zero (ie the DmaBusy bit in the CfgStat register is zero). If the DmaLen register is not zero any write to destination register will be ignored. However the register can be read at any time.

Note that the value of this register increases with every DMA transfer if and only if the IncDestAdr flag in the CfgStat register is set. The register is incremented when the data transfer has been acknowledged by a slave device connected to the Wishbone master interface.

DmaLen Register

This register counts the number of bytes which can be transfered by the DMA unit at max. It is normally set to max buffer length when the DMA controller is configured. Whenever a byte is transfered from the output FIFO to the Wishbone controller this register is decremented by one. Once the register reaches zero no new Wishbone transfers will be initiated. The DMA unit can be stopped at any time by writing this register to zero.

FifoLen Register

This read only register allows to read the number of bytes current stored in the ouput (bits 31..16) and input (bits 15..0) FIFOs.

Command Register

Writing this register triggers special commands listed in the following table. All values not defined in the table are reserved for further use and must not be written.

Name	Value	Description
RESET	x”00000001”	Reset the compressor core, the DMA unit and the FIFOs. Any data stored in the core or the FIFO is discarded. If a DMA Wishbone transfer is currently in progress it is aborted immediately. Note that the configuration registers will not be modified by this command. Right after writing the RESET command to the command register new data can be written to the data input register.

FLUSH	x”00000002”	This command sets an internal flag which blocks any further data from be written to the input FIFO. Subsequent writes to the data input register will be ignored. When the input FIFO in turn is empty the core will be flushed causing all internal buffers to be emptied. Once the last byte has been encoded the an end-of-data symbol will be created by the core. Finally the CoreDone flag in the CfgStat register is set to indicated that the compression process has been finished and all data has been transfered to the output FIFO. This will generate an interrupt request if the IECoreDone flag is set in the CfgStat register. Once this command has been executed no new data can be compressed until the RESET command is used to re-initiate the core.

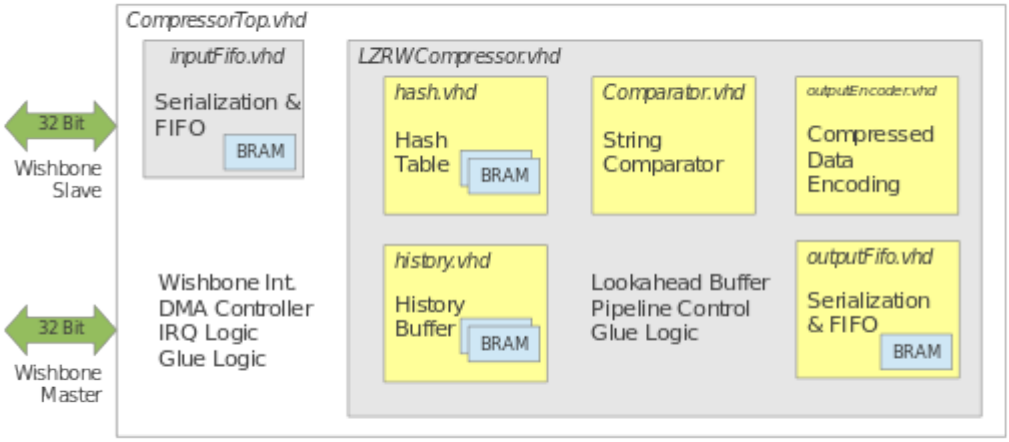
3.2 Master Interface

The master interface is used by the DMA unit to write the stream of compressed data to configured destination address. The signals follow the Wishbone specification but use a slightly different naming convention. Note that if cycle is terminated by the error signal (MaErrxSI) the DMA unit will reset the DmaLen register to zero and the DmaErr flag in the CfgStat register is set. The DmaErr flag is cleared when the CfgStat register is written.

Name	Direction	Type
MaCycxSO	out	std_logic;
MaStbxSO	out	std_logic;
MaWexSO	out	std_logic;
MaSelxDO	out	std_logic_vector(3 downto 0);
MaAdrxDO	out	std_logic_vector(31 downto 0);
MaDatxDO	out	std_logic_vector(31 downto 0);
MaDatxDI	in	std_logic_vector(31 downto 0);
MaAckxSI	in	std_logic;
MaErrxSI	in	std_logic

4 Implementation

The entire core and its test benches are implemented in VHDL. All names (for entities, signals,...) follow the standard published in [DZ-Naming]. The block diagram can be seen in Drawing 1.



Drawing 1: Block Diagram

4.1 File Overview

The following sections give a brief introduction to what is done in the individual files. For details please refer directly to the fully commented source code.

CompressorTop.vhd

This is the top level file. It implements the two Wishbone interfaces, one slave for configuration and data input, and one master for the DMA unit. Further it implements all configuration and status logic and the DMA unit.

inputFifo.vhd

Using a 2kB Xilinx Block RAM (BRAM) this file implements a 4Byte-In-1Byte-Out FIFO. It receives data from the top level file and automatically transfers it to the core when it is ready to accept new data.

LZRWCompressor.vhd

The main pipeline executing the actual data compression is implement in this file. It uses several sub blocks to perform the specific tasks and keeps track of pipeline control. Further it implements the look ahead buffer used by the algorithm and some glue logic necessary to connect the blocks.

hash.vhd

This file implements the hash table which serves as central index for the recognition of repeated

strings. The incoming, three bytes wide, key is hashed using the hash function given in the original algorithm. Two 2kB Xilinx Block RAMs are used to store the table.

histroy.vhd

The history buffer is simple 4kBytes long, byte oriented, FIFO buffer. It uses two 2kB Xilinx Block RAMs.

Comparator.vhd

Once a candidate has been loaded from the history buffer this unit compares it to the current lookahead buffer and determines how many bytes match. This length is presented on an output for further processing.

outputEncoder.vhd

This file encodes the literals and copy items into frames of eight items each. Once a frame is finished it is prepended with an header and emitted in serialized form.

outputFifo.vhd

The output FIFO buffer receives compressed data byte by byte from outputEncoder and stores it a Xilinx Block RAM. It is retrieve from the FIFO by the DMA controller.

Bibliography

LZRW1: Ross N. Williams, An Extremely Fast ZIV-LempelData Compression Algorithm, 1991
DZ-Naming: , VHDL Naming Conventions, , http://dz.ee.ethz.ch/en/information/hdl-help/vhdl-naming-conventions.html