IMA ADPCM Encoder / Decoder

Core Specifications

Written for publication on:

File name:	*IMA ADPCM EncDec Core Specifications*
Version:	*0.1*
Creation Date:	*6/6/2011*
Update Date:	*18/9/2011*
Author:	*Moti Litochevski*

Table of Contents

Index of Tables

Index of Figures

1. Preface

1.1. Scope

This document describes the IMA ADPCM Encoder & Decoder IP core operation, architecture and interfaces.

1.2. Revision History

Rev	*Date*	*Author*	*Description*
0.1	18/09/11	Moti Litochevski	First Draft

1.3. Abbreviations

ADPCM Adaptive Differential Pulse Code Modulation

IMA Interactive Multimedia Association

2. Introduction

The IMA ADPCM compression algorithm belongs to the Adaptive Differential Pulse Code Modulation type algorithms. The algorithm is based on a simple adaptive quantization of the difference between the input and a predictor. Each 16-bit input sample is converted to a 4-bit coded information which yields a compression ratio of ¼. We will not go through detailed description of the algorithm in this document. Please refer to one of the following links for detailed description of the algorithm:

1. http://serghei.net/docs/programming/algorithms/BOOKS/BOOK10/9711M/9711M.HTM
2. http://wiki.multimedia.cx/index.php?title=IMA_ADPCM

Many other links exist out there, just Google "IMA ADPCM".

The main advantage of the IMA ADPCM compression algorithm are its simplicity. The algorithm is not limited to voice signals and can operate at any input sampling rate thus enabling compression of high quality audio as well.

The implementation in this project does not follow any standard format for the compressed information. The encoder uses a very simple interface to input 16-bit samples and output coded information nibbles (4-bits). A similar interface is used by the decoder to input the coded information nibbles and output 16-bit reconstructed samples.

A bit exact fixed point Scilab implementation of the algorithm is supplied with the core and is used to generate files used during verification. Detailed description of the verification process is given in the Test Bench Description chapter.

The following table summarizes the synthesis results of the encoder core for different FPGA families.

Manufacturer	*Family*	*Device*	*Device Utilization*	*Elements Utilization*	*Fmax*
Xilinx	Spartan 6	xc6slx4-3tqg144	16.00%	96 Slices	>100MHz
Xilinx	Virtex 5	xc5vlx30-3ff324	1.00%	84 Slices	>100MHz
Altera	Cyclone III	ep3c5f256c6		? LEs	>100MHz
Altera	Startix III	ep3sl50f484c2		? Registers ? ALUTs	>100MHz

Table 1: Encoder Core Synthesis Results for Different FPGA Devices

The following table summarizes the synthesis results of the decoder core for different FPGA families.

Manufacturer	*Family*	*Device*	*Device Utilization*	*Elements Utilization*	*Fmax*
Xilinx	Spartan 6	xc6slx4-3tqg144	6.30%	44 Slices	>100MHz
Xilinx	Virtex 5	xc5vlx30-3ff324	1.00%	47 Slices	>100MHz
Altera	Cyclone III	ep3c5f256c6		? LEs	>100MHz
Altera	Startix III	ep3sl50f484c2		? Registers ? ALUTs	>100MHz

Table 2: Decoder Core Synthesis Results for Different FPGA Devices

The above results where obtained using the following software versions:

- Xilinx ISE Webpack 13.2
- Altera Quartus Web Edition 9.0p2

3. Architecture

The project includes two separate modules for the ADPCM encoder and decoder. As described by the ADPCM algorithm the encoder actually includes a decoder but does not use the same decoder module. The following figure depicts the block diagram of the encoder.

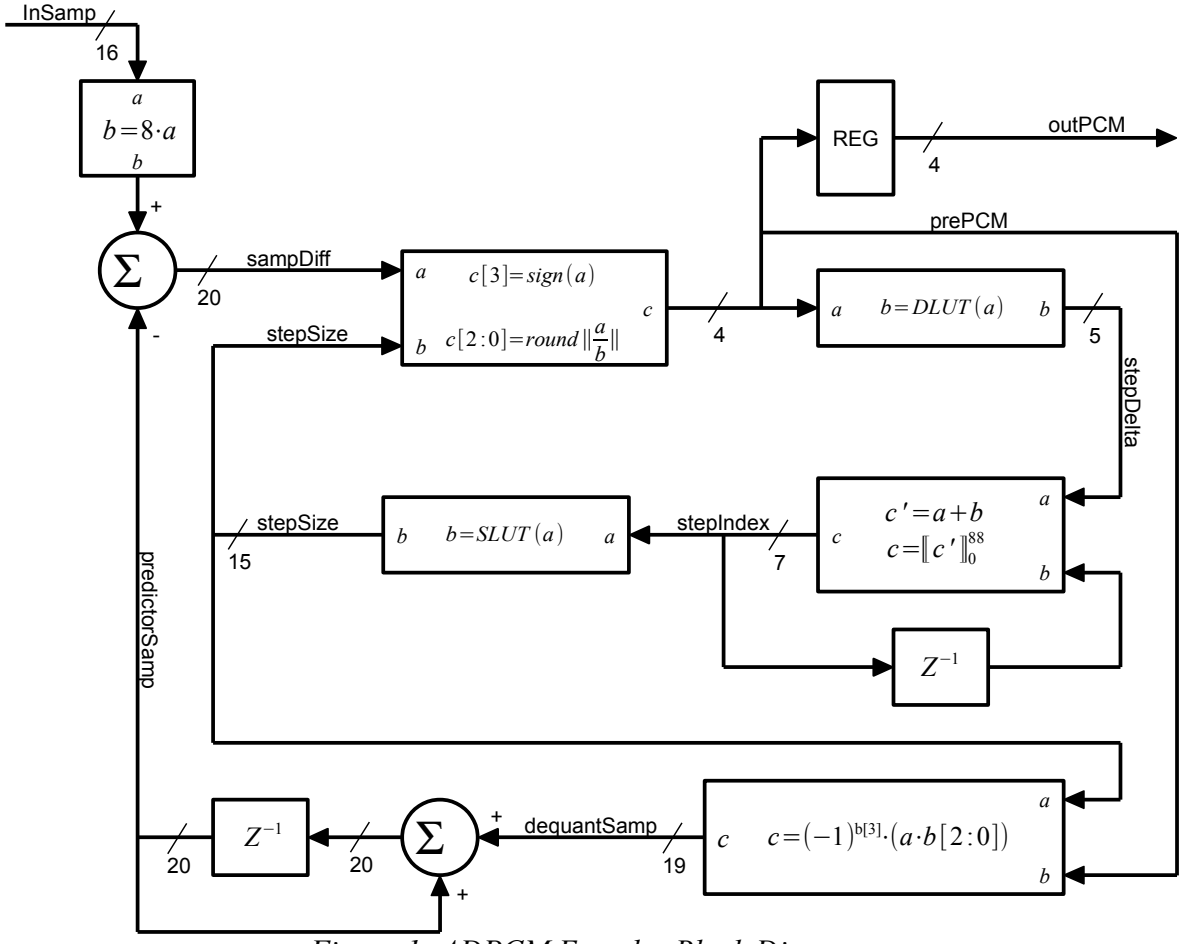

Figure 1: ADPCM Encoder Block Diagram

The encoder implements the adaptive quantizer, predictor and quantize the difference between the input sample and the predicted sample value. The predicted sample is calculated with full precision so the input sample is first scaled to the internal state scale. The step delta and size tables are taken from the IMA ADPCM. The encoder module operation is controlled using a state machine which processes a new input sample every 6 clock cycles.

The following figure depicts the block diagram of the ADPCM decoder.

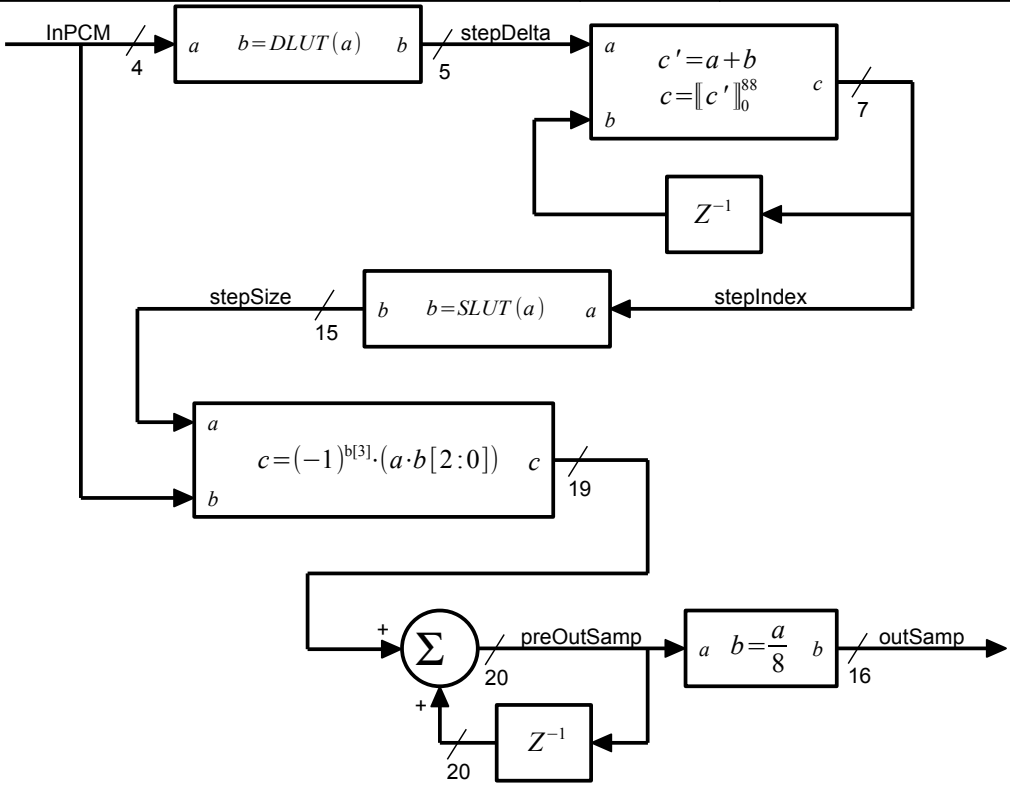

Figure 2: ADPCM Decoder Block Diagram

As depicted in the above figure the encoder and decoder modules have two internal state registers: the step index and internal sample value. When the compressed bit steam at the decoder input may have errors or gaps the decoder internal state must be synchronized to minimize the affect on the reconstructed audio signal. The decoder processes an output sample every other clock cycle, 2 cycles per sample.

The encoder and decoder modules have optional internal state outputs and inputs which may be used to format any protocol to transmit or store the internal state periodically. The modules supplied in this project do not implement any specific standard protocol, this is left to the user.

4. Core Interfaces

The following table summarizes the ADPCM encoder core interface ports.

Name	*Direction*	*Width*	*Description*
clock	input	1	Global core clock signal.
reset	input	1	Global core asynchronous reset, active high.
inSamp	input	16	Input sample to encode.
inValid	input	1	Input valid flag. This input is checked only when inReady flag is active.
inReady	output	1	Input ready output flag. This flag indicates that the module is ready to encode a new sample. This output may be used as an input FIFO read control with the valid input set as fifo **not** empty flag.
outPCM	output	4	Output encoded sample PCM value. The MSB is the sign bit and the other 3 bits are the quantized value.
outValid	output	1	Output valid flag.
outPredictSamp	output	16	Internal predictor sample value. This value should be sampled with the output valid flag and should be loaded to the decoder after the respective PCM value is presented to the decoder.
outStepIndex	output	7	Internal quantization step index value. This value should follow the same timing described for the internal predictor sample value above.

Table 3: ADPCM Encoder Interfaces Description

The following table summarizes the ADPCM decoder core interface ports.

Name	*Direction*	*Width*	*Description*
clock	input	1	Global core clock signal.
reset	input	1	Global core asynchronous reset, active high.
inPCM	input	4	Input sample to decode.
inValid	input	1	Input valid flag. This input is checked only when inReady flag is active.
inReady	output	1	Input ready output flag. This flag indicates that the module is ready to encode a new sample. This output may be used as an input FIFO read control with the valid input set as fifo **not** empty flag.
inPredictSamp	input	16	Internal predictor sample register value input. This input should be loaded into the decoder when it is not busy processing an output sample (inReady output is high).
inStepIndex	input	7	Internal step index register value input. This input should be loaded into the decoder when it is not busy processing an output sample (inReady output is high).
inStateLoad	input	1	Internal state registers load control signal. This input should be set high to load the internal state register inputs to the internal decoder state while the inReady output is high.
outSamp	output	16	Decoded output sample value.
outValid	output	1	Decoded output sample valid flag.

Table 4: ADPCM Decoder Interfaces Description

Note:
The port direction in the table above is as defined in the core top module.

5. Test Bench Description

The core distribution includes a full bit exact Scilab simulation model. The Scilab functions are used with the Verilog test bench to verify the core functionality. The Scilab functions are located in the 'scilab\' directory which also includes a script that reads an input WAV file, runs the ADPCM encoder and decoder functions, and optionally saves Verilog input and output test vectors.

The following table lists the relevant files in the 'scilab\' directory:

Filename	*Description*
ima_adpcm_enc.sci	The ADPCM encoder bit exact simulation function. The function accepts a single input vector of 16-bits samples and returns a vector with the same length of encoded 4-bits samples.
ima_adpcm_dec.sci	The ADPCM decoder bit exact simulation function. The function accepts a single input vector of 4-bits encoded samples and returns a vector with the same length of decoded 16-bits samples.
test_ima_adpcm.sce	This is the simulation test script used to test the algorithm and generate the Verilog test vectors. A description of the simulation options is given below.
1234.wav	This is a sample WAV file used by the Scilab simulation to test the algorithm and generate the Verilog test vectors.
test_in.bin	Verilog simulation input samples file containing the samples read from the Scilab input WAV file. By default the “1234.wav” file is used.
test_enc.bin	Verilog simulation encoder output test vector. This file is used by the test bench to verify the encoder output.
test_dec.bin	Verilog simulation decoder output test vector. This file is used by the test bench to verify the decoder output.

Table 5: Scilab Simulation Directory File List

The Verilog test bench uses three input files:

1. Encoder input samples file (“test_in.bin”) which contains the input 16-bits samples read from the source WAV file. Each 16-bits sample is stored as two bytes in the binary file.
2. Encoder output samples file (“test_enc.bin”) which contains the encoder 4-bits output samples. Each 4-bits encoded sample is stored as a single byte in the binary file. The values read from this file are compared to the encoder output values.
3. Decoder output samples file (“test_dec.bin”) which contains the decoder 16-bits output samples. Each 16-bits sample is stored as two bytes in the binary file. The values read from this file are compared to the decoder output values.

The Scilab simulation script is configured by setting some parameters in the initialization of the script. The following figure contains the script configuration parameters and the comments

describing the different options.

```
// set script parameters
// set to non zero to enable user selected WAV file
select_file_gui = 1;
// set default file name when GUI is disabled or canceled
default_filename = "1234.wav";
// maximum length of audio samples to limit the runtime & verilog
// vector files set to 0 to disable any size limiting
maximum_samp_len = 0;
// set to non zero value to enable verilog simulation input & output
// vectors creation
verilog_vec_enable = 0;
```

Figure 3: Scilab Simulation Script Parameters

The Verilog test bench is located in the 'verilog\bench\ directory. Compilation batch files are included in the 'verilog\sim\icarus' directory used to simulate the core using Icarus Verilog. The test bench reads the binary files used during simulation from the 'scilab' directory directly.

The directory also includes batch file to run the simulation, 'run.bat', and another to call gtkwave to view the simulation VCD output file, 'gtk.bat'.