*ICIC Express Letters*
*Part B: Applications*
Volume **7**, Number **6**, June **2016**

ICIC International ©2016 ISSN 2185-2766
pp. 1323–1328

# SOFT DSP DESIGN METHODOLOGY OF ILBC SPEECH DECODER ON NIOS II EMBEDDED PLATFORM

Wen-You Huang and Chian C. Ho*

Department of Electrical Engineering
National Yunlin University of Science and Technology
No. 123, University Road, Section 3, Douliou, Yunlin 64002, Taiwan
*Corresponding author: futureho@yuntech.edu.tw

Received November 2015; accepted February 2016

Abstract. *Based on the emerging and significant Soft DSP technique, this paper develops Soft DSP of iLBC speech decoder to boost Voice over Internet Protocol (VoIP) embedded system without extra coprocessor cost and power consumption. This paper not only illustrates the HW/SW codesign methodology of Soft DSP of iLBC speech decoder in detail, but also gives a step-by-step implementation tutorial on Nios II embedded platform. The experimental results of Soft DSP of iLBC speech decoder can verify Soft DSP technique without pure HW cost and power consumption achieves faster processing performance than the pure SW technique. Soft DSP technique has been entitled to be the development mainstream of VoIP embedded system field.*
**Keywords:** iLBC speech decoder, Soft DSP, VoIP embedded system

## 1. Introduction.

The advent of "Soft DSP" technique has made it attractive to perform faster signal processing functions in the embedded system domain. Especially, "Soft Speech Codec" is quite beneficial to VoIP embedded system placing important emphasis on real-time speech processing and cost-effective chipset architecture. Therefore, this paper takes "Soft Speech Codec" as a tutorial example to interpret the design and implementation methodology of "Soft DSP" technique.

Among a variety of low bitrate speech codec standards currently adopted by VoIP industries, internet Low Bitrate Codec (iLBC) is likely the favorite choice, because it is a royalty-free and open source narrowband speech codec [1]. More importantly, the speech quality of iLBC is more robust to packet loss or packet delay than patented ITU-T G.729A and G.723.1 under a very noisy environment [2-6]. The experimental result evaluated by Dynastat Inc. proves that Mean Opinion Score (MOS) speech quality of iLBC is better than that of G.729A and G.723.1 under various packet loss conditions [5]. Because of so many advantages, iLBC has already been involved into IETF RFC 3951 standard and integrated into a lot of well-known industrial VoIP softphones and hardphones, such as Gizmo5, webRTC, Ekiga, QuteCom, Google Talk, Yahoo! Messenger, and Polycom IP Phone. Therefore, this paper will focus on the implementation of iLBC codec and further improve the processing performance of iLBC codec by "Soft DSP" technique, not improve the perceptual speech quality of iLBC codec. Specifically, this paper will pay more concentration on exploring iLBC speech decoder's architecture and breaking its computational bottleneck without support of extra coprocessor.

The organization of this paper is as follows. In the next section, a brief overview about the architecture of iLBC speech decoder is given. Then the "Soft DSP" technique based on "Custom Instruction" is adopted and clarified against those pure hardware (HW) or software (SW) techniques in Section 3. Section 4 and Section 5 present design methodology and experimental results implemented on reconfigurable Nios II embedded platform to compare "Soft DSP" with pure HW/SW techniques. Finally, Section 6 draws conclusions and future works.

## 2. Architecture of iLBC Speech Decoder.

Figure 1 shows the architecture and flowchart of iLBC speech decoder. The functionality of each block in iLBC speech decoder is described in order as below. Firstly, the speech parameters are extracted from the payload of the Internet bitstream. Secondly, the corresponding Linear Spectrum Frequency (LSF) vectors are found by simple lookup tables after coefficients of Linear Prediction Coding (LPC) are decoded and interpolated. Thirdly, the 57/58-sample start state needs to be reconstructed through lots of fixed data tables. Fourthly, the memory for code construction is set up by the data of the decoded residual. Fifthly, the residual of all sub-frames is constructed. Sixthly, the residual is enhanced with the post filter. Seventhly, the residual of all sub-frames is synthesized. Finally, the post processing of high-pass filter is carried out, if necessary [6].

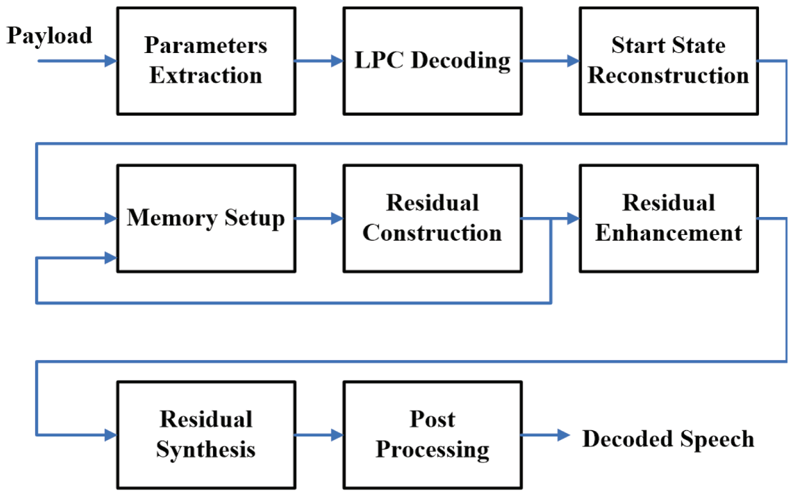


FIGURE 1. Architecture of iLBC speech decoder

Among all blocks of iLBC speech decoder in Figure 1, the computational complexity of the third block, start state reconstruction, is simpler but heavily repeated. That is where our developed "Soft Speech Codec" can work well and improve further, because "Soft DSP" technique is better at solving such a computational bottleneck than those pure HW or SW techniques. More explicit reasons will be clarified in Section 3.

## 3. Soft DSP against Pure HW/SW Techniques.

With the on-going progress of semiconductor technology, it has made "System-on-Chip" (SoC) come true, as well as the reconfigurable computing platform. Reconfigurable computing platform composed of a Reduced Instruction Set Computer (RISC) processor core (e.g., ARM, PowerPC, or Nios II), embedded memory, and programmable logic is a preferable alternative to Application-Specific Integrated Circuit (ASIC) platform and general-purpose processor system, since it can own full advantages of throughput of hardware logic and flexibility of software processors [7-9]. Next, this paper will address the performance-to-cost-ratio difference between the pure HW technique, the pure SW technique, and the HW/SW codesign method like "Soft DSP" technique, and explain why the "Soft DSP" technique is the best way to boost the speech codec on the reconfigurable computing platform like Nios II embedded platform.

Pure HW technique means the computationally intensive segments of code are processed and accelerated by specific hardware fabric, especially by the dedicated coprocessor like Codec chip or the comprehensive coprocessor like DSP. This specific hardware fabric, whether Codec chip or DSP, is working apart from the general-purpose central processor and is able to access the memory, bus, interface and peripherals independently. The pure HW technique with the specific hardware fabric can execute dedicated or comprehensive arithmetic functions much more quickly than any of pure SW and "Soft DSP" techniques, but it consumes more circuit area and power consumption than other techniques. In addition, DSP users must take time to get familiar with another instruction set or specific programming language based on the DSP architecture.

There are a large number of pure SW techniques to raise the processing performance, like algorithm-level reformation, code optimization by profiling tools, preliminary lookup table, programming by assembly or low-level language, and compiler efficiency evolution. They are performed only with the general-purpose central processor, and do not need additional coprocessor hardware fabric and power consumption. So the pure SW techniques are usually the most economical solution to cost-critical issues. However, all pure SW techniques above can only contribute a limited extent of improvement. These SW techniques are usually not good enough.

"Soft DSP" technique is also referred to as "Custom Instruction" or "Very-Long-Instruction-Word-like (VLIW-like) RISC". They are actually similar to each other, because all of them raise the embedded signal processing performance by running some custom instructions instantly and directly on the dedicated VLIW-style logic blocks [10-12]. From the circuit configuration in Figure 2, it is clearly seen that the dedicated logic blocks are compact and adjacent to Arithmetic Logic Unit (ALU) in the processor's data path configuration. Therefore, "Soft DSP" technique not only can finish some complicated arithmetic as soon as the pure HW technique, but also can make circuit cost and power consumption considerations as simple as the pure SW technique. "Soft DSP" technique can own both advantages of effective HW design and efficient SW design without independent memory, bus, interface and peripherals. Therefore, this paper will adopt "Soft DSP" technique to boost VoIP speech codec and illustrate its design methodology.

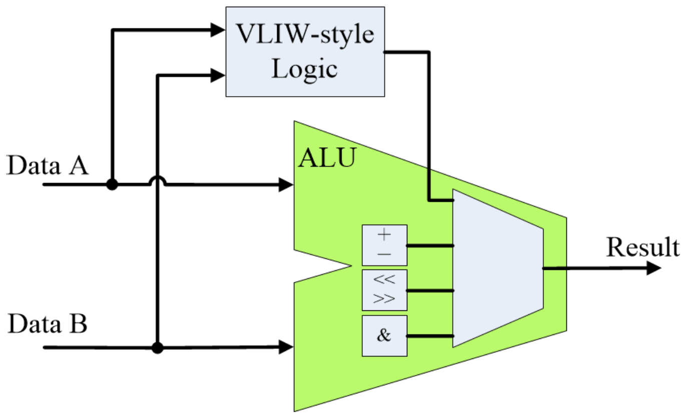


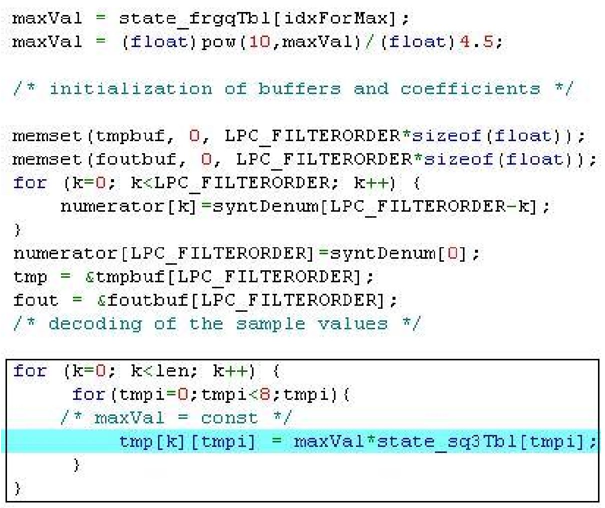
FIGURE 2. Circuit configuration for Soft DSP technique

## 4. **Design Methodology.**

As commented in Section 3, "Soft DSP" technique is an HW/SW codesign method utilizing throughput of hardware logic and flexibility of software processor. So reconfigurable Nios II embedded platform featuring programmable processor core and tailored instruction set is one of highly qualified benchmarks to implement a lab example about "Soft DSP" technique and learn the interesting HW/SW codesign methodology. This paper will make good use of reconfigurable Nios II embedded platform and Nios II HW/SW seamless toolchain, like "SOPC Builder", "Quartus II", and "Nios II IDE", to design the experimental implementation and illustrate the design methodology of "Soft Speech Codec" as below.

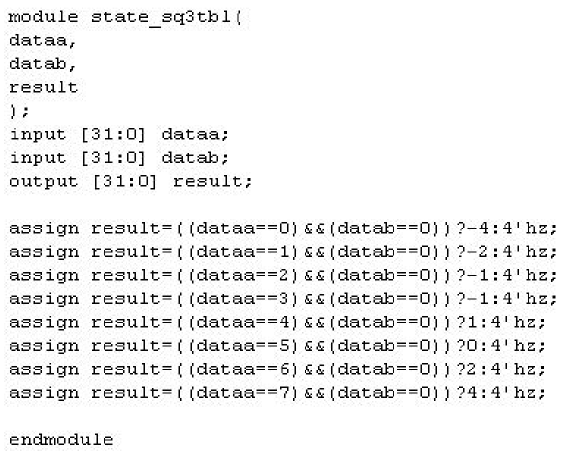
Besides, as mentioned in Section 2, the code of start state reconstruction block of iLBC speech decoder is where the "Soft DSP" technique is good at and where this paper is interested in because it is rightly a simpler but heavily repeated computation. The experimental implementation in this paper will concentrate on the software code of start state reconstruction block of iLBC speech decoder, and will create a dedicated logic and a custom instruction to replace and improve on that.

Figure 3(a) shows some original code segments of start state reconstruction block of iLBC speech decoder. In the highlighted section of software code of Figure 3(a), the variable *maxVal* is a constant value acquired from the prior instructions, and the functionality of *state_sq3Tbl[]* is simply to access a $1 \times 8$ fixed data table from the external memory. And, the loop number *len* means the sample size of every speech frame. Here, the loop number *len* is required to equal 240 (30 ms) or 160 (20 ms) for iLBC speech codec. For this so-called simpler but heavily repeated bottleneck, the experimental implementation will reform it into "Soft Speech Codec" by "Soft DSP" technique, specifically by the enhanced custom instruction of *state_sq3Tbl[]*. The detailed design methodology is illustrated as follows.



```
maxVal = state_frgqTbl[idxForMax];
maxVal = (float)pow(10,maxVal)/(float)4.5;

/* initialization of buffers and coefficients */

memset(tmpbuf, 0, LPC_FILTERORDER*sizeof(float));
memset(foutbuf, 0, LPC_FILTERORDER*sizeof(float));
for (k=0; k<LPC_FILTERORDER; k++) {
    numerator[k]=syntDenum[LPC_FILTERORDER-k];
}
numerator[LPC_FILTERORDER]=syntDenum[0];
tmp = &tmpbuf[LPC_FILTERORDER];
fout = &foutbuf[LPC_FILTERORDER];
/* decoding of the sample values */

for (k=0; k<len; k++) {
     for(tmpi=0;tmpi<8;tmpi){
    /* maxVal = const */
         tmp[k][tmpi] = maxVal*state_sq3Tbl[tmpi];
     }
}
```

(a)



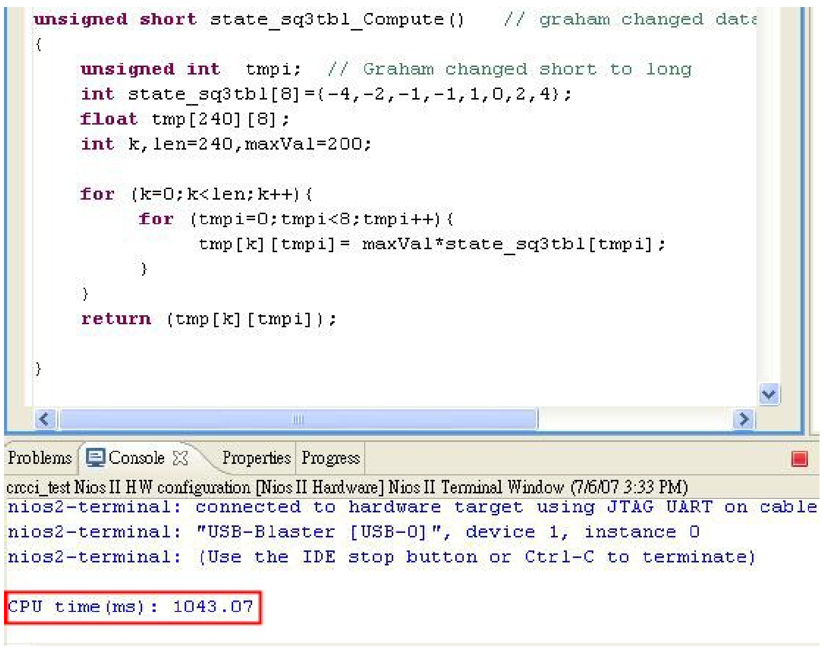
```
module state_sq3tbl(
dataa,
datab,
result
);
input [31:0] dataa;
input [31:0] datab;
output [31:0] result;

assign result=((dataa==0)&&(datab==0))?-4:4'hz;
assign result=((dataa==1)&&(datab==0))?-2:4'hz;
assign result=((dataa==2)&&(datab==0))?-1:4'hz;
assign result=((dataa==3)&&(datab==0))?-1:4'hz;
assign result=((dataa==4)&&(datab==0))?1:4'hz;
assign result=((dataa==5)&&(datab==0))?0:4'hz;
assign result=((dataa==6)&&(datab==0))?2:4'hz;
assign result=((dataa==7)&&(datab==0))?4:4'hz;

endmodule
```

(b)

FIGURE 3. (a) Original code segment of start state reconstruction block; (b) Verilog code of custom instruction of *state_sq3Tbl[]*

At first, the dedicated logic for running the custom instruction of *state_sq3Tbl[]* must be designed through "Verilog" Hardware Description Language (HDL) as shown in Figure 3(b). The objective of the dedicated logic *state_sq3tbl.v* in Figure 3(b) is just to access a $1 \times 8$ fixed data table rapidly in hardware way. After the connectivity diagram and pin assignment of Nios II processor is ensured to be valid and appropriate through "SOPC builder" tool, the dedicated logic in Figure 3(b) can be integrated into Nios II processor core through "SOPC builder" tool. The imported combinational logic file *state_sq3tbl.v* is rightly the aforementioned dedicated logic for running the custom instruction of *state_sq3Tbl[]*. Next, it is still necessary to resynthesize Nios II processor core through "Quartus II" tool and download the updated .sof image file to the embedded platform through "Quartus II" tool. Afterward, this reformed Nios II hardware platform has the capability to run the enhanced custom instruction effectively and efficiently.

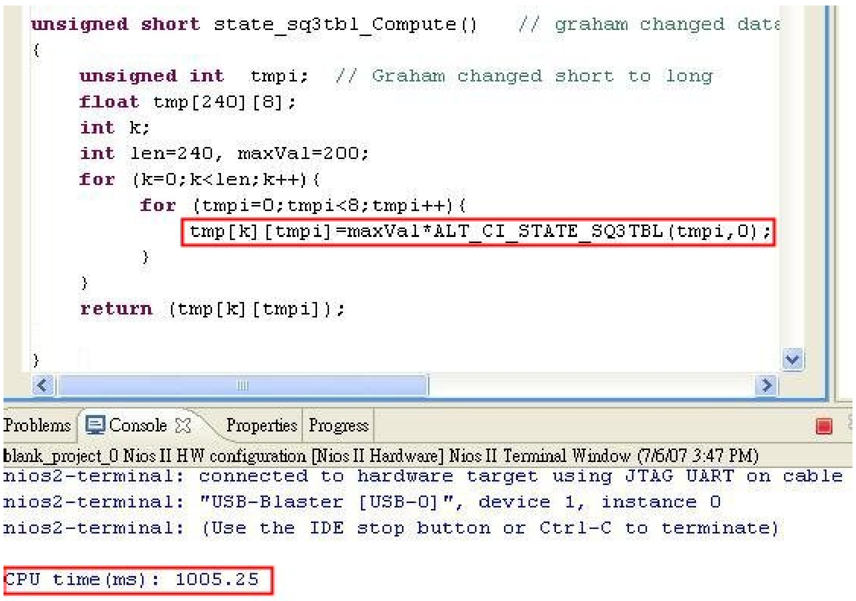
On the other end, for the software code segment as shown in Figure 4(a), the experimental implementation has to replace the original instruction *state_sq3Tbl[]* with the enhanced custom instruction *ALT_CI_STATE_SQ3TBL()* through "Nios II IDE" tool as highlighted in Figure 4(b). In the highlighted section of software code of Figure 4(b), the variable *maxVal* is still a constant value acquired from the prior instructions, and *ALT_CI_STATE_SQ3TBL()* is created and engaged in driving the dedicated logic



```
unsigned short state_sq3tbl_Compute()   // graham changed data
{
     unsigned int  tmpi;  // Graham changed short to long
     int state_sq3tbl[8]={-4,-2,-1,-1,1,0,2,4};
     float tmp[240][8];
     int k,len=240,maxVal=200;

     for (k=0;k<len;k++){
          for (tmpi=0;tmpi<8;tmpi++){
               tmp[k][tmpi]= maxVal*state_sq3tbl[tmpi];
          }
     }
     return (tmp[k][tmpi]);

}
```

```
crcci_test Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (7/6/07 3:33 PM)
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

CPU time(ms): 1043.07
```

(a)



```
unsigned short state_sq3tbl_Compute()   // graham changed data
{
     unsigned int  tmpi;  // Graham changed short to long
     float tmp[240][8];
     int k;
     int len=240, maxVal=200;
     for (k=0;k<len;k++){
          for (tmpi=0;tmpi<8;tmpi++){
               tmp[k][tmpi]=maxVal*ALT_CI_STATE_SQ3TBL(tmpi,0);
          }
     }
     return (tmp[k][tmpi]);

}
```

```
blank_project_0 Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (7/6/07 3:47 PM)
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

CPU time(ms): 1005.25
```

(b)

Figure 4. SW code and execution time of (a) original and (b) soft iLBC speech decoder

*state_sq3tbl.v* to access a $1 \times 8$ fixed data table rapidly. Next, "Nios II IDE" tool applies GNU C Compiler (GCC) to rebuilding the reformed software code of iLBC speech decoder into .elf executable file and downloads the executable file to the embedded platform. In short, the HW/SW codesign methodology of "Soft Speech Codec" is expansion of the hardware core design followed by reformation of the software code design.

## 5. Experimental Results.

In order to analyze and compare the experimental results, this paper writes a software code behind the code of iLBC speech decoder to count the processing time of Nios II processor with/without "Soft DSP" technique. The software code and execution time of the code segment of interest in "original iLBC speech decoder" and "soft iLBC speech decoder" are shown in Figures 4(a) and 4(b), respectively. The experimental results verify that the processing time of Nios II processor with support of only one custom instruction is about 38 ms shorter than original Nios II processor for every speech frame's decoding. So the boosting effect of "Soft Speech Codec" without extra coprocessor cost and power consumption is dramatic and significant.

6. **Conclusions.** According to the experimental results of iLBC speech decoder in the experimental implementation, the processing time of Nios II processor with "Soft DSP" technique can be decreased by about 38 ms for each 240-sample speech frame's decoding. In other words, the "Soft iLBC speech decoder" implementation in the lab example can decrease the processing time of $38n$ ms totally when decoding a speech bitstream consisting of $n$ speech frames. Furthermore, it is worth noting that only one custom instruction is implemented in this case of the experimental implementation, but Nios II processor is permitted to extend and couple at most 256 unique custom instructions [12]. In the future research direction, the more bottlenecks that are simpler but heavily repeated are found out to reform, the better "Soft DSP" technique can improve the speech codec processing performance of VoIP embedded platform without extra coprocessor cost and power consumption. Besides, against the pure HW technique, "Soft DSP" technique features: 1) smaller circuit area, 2) lower power consumption, 3) better design flexibility, and 4) richer feature scalability.

## REFERENCES

[1] Wikimedia Foundation, Internet Low Bitrate Codec (iLBC), *Wikipedia*, https://en.wikipedia.org/wiki/Internet_Low_Bitrate_Codec, 2015.

[2] M. Menth, A. Binzenhofer and S. Muhleck, Source models for speech traffic revisited, *IEEE Trans. Networking*, vol.17, pp.1042-1051, 2009.

[3] K. Seto and T. Ogunfunmi, Scalable speech coding for IP networks: Beyond iLBC, *IEEE Trans. Audio, Speech, Language Processing*, vol.21, pp.2337-2345, 2013.

[4] F. Mousavipour and M. J. Khoseavipour, VoIP quality enhancement with wideband extension method in broadband networks, *IEEE Latin America Transactions*, vol.10, pp.1190-1194, 2012.

[5] Global IP Sound, iLBC – Designed for the future, *White Paper*, 2004.

[6] S. Andersen, A. Duric, H. Astrom, R. Hagen, W. Kleijn and J. Linden, Internet Low Bit Rate Codec (iLBC), *IETF RFC 3951*, 2004.

[7] V. Dumitriu and L. Kirischian, SoPC self-integration mechanism for seamless architecture adaptation to stream workload variations, *IEEE Trans. Very Large Scale Integration Systems*, 2015.

[8] F. A. Escobar, X. Chang and C. Valderrama, Suitability analysis of FPGAs for heterogeneous platforms in HPC, *IEEE Trans. Parallel and Distributed Systems*, 2015.

[9] J. Noguera and R. M. Badia, HW/SW codesign techniques for dynamically reconfigurable architectures, *IEEE Trans. Very Large Scale Integration Systems*, vol.10, pp.399-415, 2002.

[10] P. Biswas and N. D. Dutt, Code size reduction in heterogeneous-connectivity-based DSPs using instruction set extensions, *IEEE Trans. Computers*, vol.54, pp.1216-1226, 2005.

[11] A. Lodi, M. Toma, F. Campi, A. Cappelli, R. Canegallo and R. Guerrieri, A VLIW processor with reconfigurable instruction set for embedded applications, *IEEE Journal of Solid-State Circuits*, vol.38, pp.1876-1886, 2003.

[12] Altera, *Nios II Custom Instruction User Guide*, 2011.