GOP level parallelism implementation for real-time H264/AVC video encoder on multicore DSP TMS320C6472

Nejmeddine Bahri, Thierry Grandpierre, Med Ali Ben Ayed, Nouri Masmoudi, Mohamed Akil

GOP LEVEL PARALLELISM IMPLEMENTATION FOR REAL-TIME H264/AVC VIDEO ENCODER ON MULTICORE DSP TMS320C6472

Nejmeddine Bahri, Thierry Grandpierre,Med Ali Ben Ayed, Nouri Masmoudi, Mohamed Akil

ESIEE Engineering / LIGM Laboratory, University Paris-EST-France
National School of Engineers / LETI Laboratory, University of Sfax-Tunisia
nejmeddine.bahri@esiee.fr thierry.grandpierre@esiee.fr

ABSTRACT

In this paper, we exploit the parallelism offered by six-core Digital Signal Processor (DSP) TMS320C6472 to implement the H264/AVC video encoder in order to meet the real-time constraint for different video resolutions. To enhance the encoding speed, GOP Level Parallelism approach is implemented on 5 slave DSP cores. A master core is reserved to manage data transfers among DSP memory and personal computer in order to perform a real-time video encoding demo taken into account video capture and bit-stream storage. Multithreading algorithm and ping-pong buffers technique are used in order to optimize the communication overhead. Experimental results show that our enhanced implementation allows to overcome the real-time constraint by reaching up to 120 f/s (frame/second) for Common Intermediate Format resolution (CIF 352x288) and 35f/s for Standard Definition (SD 720x480). Our proposed approach can save about 80% of run-time for High Definition resolution (HD 1280x720). The Enhanced GOP Level parallelism approach on five DSP cores achieves a good average speedup factor of 4.88 without inducing any quality degradation or bit-rate increment.

1. INTRODUCTION

Currently, embedded systems are used in several application such as telecommunication, medical, defense and TV video coding etc. New embedded systems technologies such as multicore and multi-processor architectures allow designers to develop more complex applications that require high processing capability. H.264/AVC [1] video encoder is one of those applications. It performs a better video coding efficiency compared to previous standards. However, a high computational complexity is led to this coding efficiency. To meet the real-time constraint, a high-performance computing capability is required. The trend towards high definitions makes it hard to achieve a real-time video encoding on embedded mono-core processors with low CPU frequency. As a result, using multicore and parallel architectures will be imposed to recompense this deficiency and to reduce the run-time of H264/AVC encoder. Several works have been published taken into account the potential parallelism of the H264/AVC encoder. Different partitioning techniques are discussed based on applying functional partitioning algorithms, data partitioning algorithms or both. Multicore, multi-processor and multi-threading encoding systems have been suggested in many papers [2] to [6]. In this paper, a H264/AVC video encoder implementation on a multicore DSP TMS320C6472 applying GOP Level Parallelism (GLP) approach is detailed. A real-time video coding demo is presented taken into account image capture from a digital camera linked to our DSP platform using Ethernet connection, DSP encoding and bit-stream saving. Hiding communication overhead is also presented based on performing a multithreading algorithm and exploiting the standard ping-pong buffers technique.

The rest of this paper is structured as follows: the next section discusses the different partitioning methods and some related works on the parallel implementations of the H264/AVC video encoder. The multicore DSP TMS320C6472 architecture is described in section 3. Section 4 highlights our enhanced implementation of GLP approach on five slave DSP cores and details the experimental results. Finally, section 5 concludes this paper with some perspectives.

2. H264/AVC ENCODER : PARTITIONING METHODS AND RELATED WORKS

The H.264/AVC encoder baseline profile includes several modules such as intra prediction, inter prediction, integer cosine transform, quantification, entropy coding etc. This standard splits a video sequence into a hierarchical structure. The top level of this structure is the sequence that includes one or more groups of pictures (GOP). Each GOP includes one or more frames and always starts with intra frame (I). The other frames are a predicted frames (P). Finally, the frames are divided into one or more slices, subdivided themselves into macroblocks (MB) and blocks. According to functions organization and hierarchical sequence structure in the H264/AVC video encoder, two mainly partitioning approaches are existed:

Task-level parallelization (TLP): it decomposes the encoder into several steps, identify them into a different group of tasks equal to the number of threads available on the system and run these groups of tasks simultaneously as a pipeline. Several works have applied this method as [2]. This approach ensure a low latency encoding however, it is not well efficient for the H264/AVC video encoder because of data dependencies between tasks that require a large amount of data transfers among processors; thus, consumption of the system bandwidth. Also functions in the H.264/AVC encoder have not the same load balance which makes it hard to uniformly map functions among

processors. As a result, the final performance is always limited by the heaviest load processor.

Data-level parallelization (DLP): it exploits the hierarchical data structure of the H264/AVC encoder by simultaneously processing several data levels on multiple processing units. DLP is restricted by data dependencies among the different data units.

In fact, no dependencies are existed among different GOPs because each GOP starts with an intra frame (MB encoding requires only data from its neighboring MBs in the same frame). Hence, several GOPs could be encoded in parallel. This approach is called "GOP Level Parallelism". It has been adopted by several researchers as in [3].This method ensures the best encoding speedup but requires a large memory amount. Thus, it is not adequate for System on Chip platforms (SOC).

Motion estimation in the H264/AVC encoder imposes a partial dependency between the successive P frames of the same GOP. A search window is required in the reference frames (previous frames) to calculate the motion vector of the current MB. Thus, multiple frames can also be encoded in a pipeline structure once the search window in the reference frame has been encoded. Consequently, this method is called "Frame Level Parallelism" [4]. It provides a compromise between encoding latency and implementation efficiency.

Other works apply slice level parallelism such as in [5]. They split the frame into independent slices and simultaneously process them on different units. The major drawback of slice level parallelism is that it induces a bit-rate increment because some data dependencies are not respected.

Finally, in the same frame, several MBs could be encoded at the same time once neighboring MBs of the current MB are encoded in order to respect spatial data dependencies. This scheme is called "MB level Parallelism" [6]. Large amount of data transfers and synchronizations between processors in addition to the non-equal load balance make the MB level parallelism approach not efficient for the parallel H264/AVC video encoder.

Some of the existing implementations have not yet succeeded to meet the real-time constraint for high resolutions. For that, this work will present an academic solution to realize a real-time H264/AVC video encoder demo using a powerful platform and applying an efficient partitioning method.

3. DSP PLATFORM DESCRIPTION

Software flexibility, low cost, low power consumption and time-to-market reduction make the DSP an efficient solution for embedded implementations and high performance applications. According to these merits and motivated by the enhancement of DSP architectures, the multicore DSP TMS320C6472 [7] is chosen to implement the H264/AVC video encoder and meet the real-time video encoder constraint (25~30 f/s). This DSP is characterized by a high computing power, low power consumption and a low price which make it an attractive solution for several implementations that require high performance computing. Six C64x + DSP cores, VLIW (very long instruction word) architecture, SIMD instruction set, a large amount of memory on chip and a 700 MHz of CPU frequency are combined to provide 33600 million instructions per second. Each C64x+ core has two memory levels: L1 cache memory level, which consists of 32 K-Bytes (KB) of L1 program and 32 KB of L1 data, and 608 KB of L2 memory level. In addition to L1 and L2 memories, a 768 KB of SL2 memory is shared between all the six cores. This interesting amount of on-chip memory can reduce the access to external DDR2 memory (256 MB), so reducing the power consuming and enhancing the algorithms processing. Furthermore, performance is ameliorated by an EDMA controller which can handle data transfers instead of the CPU. Our multicore DSP supports several communication links as Gigabit Ethernet for network applications, Serial RapidIO port for DSP to DSP communications and UTOPIA2 for telecommunications.

4. ENHANCED GOP LEVEL PARALLELISM IMPLEMENTATION

In this paper, the GOP Level parallelism approach is selected in order to accelerate the encoding speed. Our choice is based on several reasons: First, this approach ensures a good speedup factor without inducing any rate distortion (Quality degradation or bit-rate increase). Second, no dependencies between GOPs make this approach easy for implementation. No data transfers or synchronizations among processors are required. Finally, no memory constraint is required unlike SOC platforms. Our DSP includes enough memory space that is able to handle all the GOPs frames.

4.1 Video encoder demo

To perform a real-time video encoding demo, DSP frames reception should be also performed in real-time. For that, 332 Mbits/s bandwidth at least is required to transfer 30 frames/s HD 720p resolution on YUV 4:2:0 format ((1280x720x1.5)x8bits x30f/s). Since our DSP evaluation board has not yet any simple frame grabber interface, a personal computer (PC) linked to a Universal Serial Bus (USB) HD webcam is used as first step to send the raw images to the DSP. Our DSP board and the PC support both a Gigabit Ethernet interface that allows us to perform a real-time data transfers among them.

As our platform includes 6 DSP cores, the first core "core0" is used as master. It is considered as a TCP server (transmission Control Protocol). It is engaged to establish TCP/IP (Internet Protocol) connection with the camera board side (PC) exploiting Network Developer's Kit library [8]. It is used firstly to receive the GOPs sent by the camera board side and save them into the DDR2 memory which is a shared memory between all the DSP cores. Then, the 5 remaining DSP cores are considered as slaves and they are

used to encode the 5 received GOPs.

Once the encoding is achieved, the core0 will send the bit-stream of all encoded frames to the PC in order to store them in a file or decode them later. For each slave core (1 to 5), a memory section is reserved. It contains the GOP current frames, the reconstructed frame (RECT) and finally the bit-stream buffers where the bit-stream of each frame from the GOP will be saved.

4.2 Optimized GOP level implementation

To enhance the classic GOP level parallelism approach, our optimization consists of hiding communication overhead. The proposed scheme is based on two strategies: the first is using the ping-pong buffers technique on the DSP side in order to overlap GOPs encoding process with reading and writing GOPs processes. The second is exploiting the multi-threading approach on the camera board side. Thus, three threads are created to handle: 1) Reading raw frames and sending them to DSP via Ethernet. 2) Receiving bit-streams from DSP. 3) Saving the received bit-streams in a file. On the DSP side, for each slave core a ping-pong GOP buffers are allocated for the current frames and also a ping-pong bit-stream buffers are allocated for the generated bit-streams for each frame from the GOP. A single buffer is used for the reconstructed frame because no transfers are required for this data. As a result, for each core, one buffer for the reconstructed frame, two GOP size buffers for the current frames and two GOP size buffers also for the bit-streams are allocated into the memory section for each slave core in SDRAM memory. In our work, GOP size is equal to 8. So, 16 buffers (8*2) for current frames, 16 buffers for bit-streams and one buffer for the reconstructed frame are allocated for each DSP core.

In the internal memory of core0, a TCP server program is loaded to establish Ethernet connection between the DSP and the PC. Our H264/AVC program is loaded into each internal memory of the 5 remaining cores. Local variables used during encoding such as predicted MB buffers, transform and quantification matrixes, best predicted modes etc are also allocated into the internal memory of each core to avoid data overlap among different cores.

A C/C++ project is implemented on the camera board side in order to capture raw frames from the USB camera using OpenCv library [9]. This library allows also converting the captured frames from RGB to YCrCb 4:2:0 format used in our H264/AVC encoder. Finally, a TCP socket is created to transfer data between core0 and the camera board via the Gigabit Ethernet link. Our implementation strategy is presented in Figure 1 and detailed as follows:

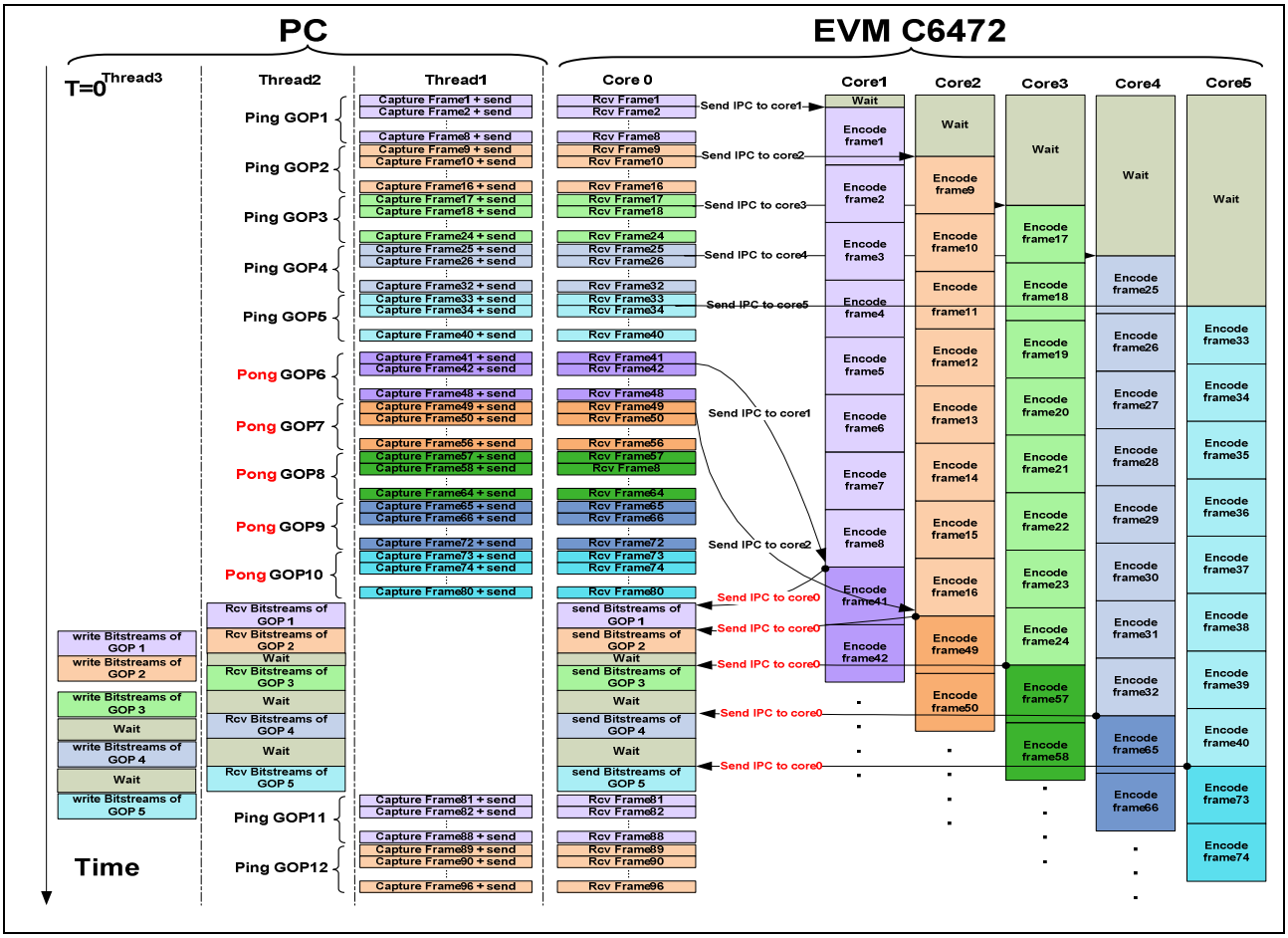

Figure 1 –The chronological steps of the Enhanced GOP Level Parallelism on the multicore DSP TMS320C6472

- The first thread "thread1" captures the first frame from a camera or a file and sends it to core0 which will save it into the ping buffer SRC[0][0] of core1. Core0 sends then an IPC event (inter processor communication interruption) to core1 to indicate that it can start encoding its current frame.
- When receiving an IPC event from core0, core1 starts encoding the first frame of the GOP and at the same time, thread1 continues reading the next frames of the first GOP and sending them to core0 which will save them into the ping buffers of core1 SRC1[0][i] (i=1 to GOP size-1).
- When finishing reading and sending the first GOP, thread1 starts reading the second GOP and sends it to core0 which will save it into the ping buffers of core2 SRC2[0][i]. The same thing as the first GOP, when receiving the first frame of the second GOP, core0 sends an IPC event to core2 to notify it that it can start encoding the first frame of its GOP. This step is repeated until finishing the reception of 5 GOPs. So each core starts encoding as soon as the first frame of its corresponding GOP is received unlike the classic GOP level implementation where encoding is started after receiving all the frames of 5 GOPs.
- During encoding the first 5 GOPs by core1 to core5, thread1 sends the next 5 GOPs to core0 which will save them into the pong buffers SRC[1][i] for each core(i=0 to GOP size-1). Because encoding process takes more time than reading process, communication delays are hidden and they will not be added to the parallel run-time.
- When a DSP core finishes encoding its ping GOP and the bit-stream is saved into the ping buffers Bistream[0][i], it sends an IPC to core0 to notify it that it can send its bit-stream to the PC. After that, this core starts encoding its pong GOP already received and saved into the pong buffers SRC[1][i] of its memory section without any wait. Then, the bit-stream will be saved into the pong buffers Bistream[1][i] to not overwrite data stored into the ping buffers Bistream[0][i] which are being transferred by core0 to the PC.
- Core0 sends the pings Bistream[0][i] to the PC, starting with the bit-streams of core1 and finishing with the bit-streams of core5 in order to be saved in a chronological order. "thread2" receives the ping bit-steams and saves them into the ping buffers Bistream[0][i]. After that, "thread3" writes the bit-streams in a file and at the same time thread1 sends the next 5 GOPs to core0 which will save them into the ping buffers SRC[0][i] of each core.
- The processing is then reprocessed in a reverse order through ping-pong buffers.

4.3 Cache coherence

Algorithms processing on a multicore platform with a shared memory can lead to a cache coherence problem. This is happen when a data allocated in a shared memory is processed by several cores with a private cache memory. In general purpose multi-processor, programmers don't have such problem because it is automatically handled by a complex hardware. In our case, developers have to manage the cache memory. In order to avoid the cache coherence problem, the Chip Support Library [10] from TI provides two API functions:

- CACHE_wbL2() to return back the cached data from the cache level to their locations in the shared memory.
- CACHE_invL2() to invalidate the cache lines and force the processor to read data from the correct location in the shared memory.

In our implementation, after reading the captured frames from the PC, core0 have to write back the cached lines to their locations in the DDR2 external memory in order to be used later by the remaining cores for encoding. From the other side, core1 to core5 have to invalidate the current data addresses in the cache memory before starting the encoding. This allows encoding the updated data written by core0 and not the old data existed in their cache memories. Furthermore, after finishing encoding, core1 to core5 should write-back the bit-streams from the cache memory to their location in the external memory. Similarly, core0 should invalidate the bit-streams from its cache memory in order to send the updated values to the PC.

4.4 Experimental results

Experiments are performed on several video sequences with different characteristics and resolutions: CIF, SD and HD on 5 DSP cores running each at 700MHz. The GOP size was 8 and number of frames was 300 for CIF resolution and 1200 frames for SD and HD resolutions. For performance evaluation, encoding speed is computed for mono-core and multicore implementation using GOP level parallelism approach on 5 slave DSP cores.

In our tests, data transfer time which consists of frames capturing, GOP structure transferring to DSP, receiving them by core0, and loading them to DSP memory is included in our calculation and added to the encoding time in order to evaluate our enhancement techniques for hiding communication overhead. Table I, II and III show respectively the encoding speeds and speedups for CIF, SD and HD resolutions for the H264/AVC video encoder on a unique core and on a 5 DSP cores.

Experimental results show that the mono-core implementation does not meet the real-time constraint (30f/s) even for low resolution. Applying our enhanced multicore implementation allows us to overcome the real-time encoding constraint by reaching up to 120 f/s and 35 f/s in average for CIF and SD resolutions respectively. For

HD resolution, real-time is unfortunately not achieved but encoding speed is efficiently improved. Run-time's gain might reach up to 80%. Regarding rate distortion, our multicore implementation does not induce any visual quality distortion or bit-rate increment comparing to mono-core implementation since data dependencies are respected.

TABLE I
ENCODING SPEED FOR CIF (352X288) RESOLUTION

CIF sequence	Encoding speed on a single core (f/s)	Encoding speed on 5 cores (f/s)	Speedup
Foreman	24.90	121.74	4.89
Akiyo	25.56	123.32	4.82
News	26.03	127.76	4.91
Container	25.68	125.37	4.88
Tb420	23.37	114.12	4.88
Mobile	22.42	109.16	4.87
average	**24.66**	**120.24**	**4.88**

TABLE II
ENCODING SPEED FOR SD (720X480) RESOLUTION

SD sequence	Encoding speed on a single core (f/s)	Encoding speed on 5 cores (f/s)	Speedup
Planets	7.24	35.23	4.87
Nature power	7.19	35.28	4.91
Turtle	7.29	35.51	4.87
Vague	7.13	34.65	4.86
Nature	7.36	36.29	4.93
Bird	7.93	38.34	4.83
average	**7.36**	**35.88**	**4.88**

TABLE III
ENCODING SPEED FOR HD (1280X720) RESOLUTION

HD sequence	Encoding speed on a single core (f/s)	Encoding speed on 5 cores (f/s)	Speedup
Planets	2.79	13.72	4.92
Nature power	2.74	13.27	4.84
Turtle	2.78	13.63	4.90
Vague	2.79	13.52	4.85
Nature	2.81	13.79	4.91
Bird	3.03	14.76	4.87
average	**2.82**	**13.78**	**4.88**

Applying multicore implementation on 5 DSP cores allows getting an interesting encoding speedup by a factor of 4.88 in average for the different resolutions. This speedup factor is very close from the theoretical value which is 5. This tiny decrease in speedup factor is first due to inter-communications needed among the master (core0) and the slaves (core1 to core5) such as write-backs and cached data invalidations and second to the impossibility of simultaneous access to the external memory by all DSP cores to read and write data. Although the data transfer time is taken into account, this time does not affect the encoding speed which affirms that our proposed data transfer scheduling technique efficiently hides the communication overhead. In fact, data transfer times will be only noticed for the first GOPs as shown in figure 1 but after that, these transfers are mostly overlapped with the encoding process. So, when testing an importing number of frames, the first transfer time will be insignificant and does not induce a performance penalty especially that encoding process is more important than reading or writing processes.

5. CONCLUSION

In this paper, an optimized H264/AVC encoder implementation on a multicore DSP was presented. GLP approach was performed to accelerate the encoding speed. Exploiting a multi-threading algorithm combined with using a ping-pong buffers technique allows enhancing our implementation and efficiently hides communication overhead. Experimental results on 5 DSP cores showed that real-time was achieved by reaching up to 120 f/s and 35 f/s as encoding speeds respectively for CIF and SD resolutions. Our parallel implementation saved about 80% of run-time for HD resolution and ensured a good speedup factor of 4.88 without resulting any video quality degradation or bit-rate increase. As perspectives, we will try to achieve a real-time encoding for HD resolution by implementing our approach on the latest generation of TI's DSP (TMS320C6678) which includes 8 DSP cores each running at 1.25 GHz, giving a large possibility to meet the real-time constraint. Also, we will move to the implementation of the new video standard HEVC-H265 exploiting our knowledge on multicore DSP implementation. Finally, it is important to notice that our demo architecture based on a camera board connected to a multicore DSP board can be used in a lot of image and video processing applications.

REFERENCES

[1] Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, "Draft ITU-T Recommendation and Final Draft international Standard of Joint Video Specification (ITU-T Rec. H.264 ISO/IEC 14496-10 AVC)", JVT-G050, 2003.

[2] Zhibin Xiao, Stephen Le and Bevan Baas," A Fine-grained Parallel Implementation of a H.264/AVC Encoder on a 167-processor Computational Platform," ACSSC 2011 – Pacific Grove, CA, 2011.

[3] S.Sankaraiah, H.S.Lam, C.Eswaran and Junaidi Abdullah, "GOP Level Parallelism on H.264 Video Encoder for Multicore Architecture," International Conference on Circuits, System and Simulation IPCSIT vol.7 IACSIT Press, Singapore 2011.

[4] Zhuo Zhao; Ping Liang, "A Highly Efficient Parallel Algorithm for H.264 Video Encoder," Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on , vol.5, no., pp.V,V, 14-19 May 2006.

[5] António Rodrigues, Nuno Roma, and Leonel Sousa," p264: Open Platform for Designing Parallel H.264/AVC Video Encoders on Multi-Core Systems," NOSSDAV '10 Proceedings of the 20th international workshop on Network and operating systems support for digital audio and video Pages 81-86, Amsterdam, 2010.

[6] Shenggang Chen; Shuming Chen; Huitao Gu; Hu Chen; Yaming Yin; Xiaowen Chen; Shuwei Sun; Sheng Liu; Yaohua Wang, "Mapping of H.264/AVC Encoder on a Hierarchical Chip Multicore DSP Platform," High Performance Computing and Communications 12th IEEE International Conference , pp.465,470, 1-3 Sept. 2010

[7] TMS320C6472 manual datasheet, online available: http://www.ti.com/lit/ds/sprs612g/sprs612g.pdf

[8] TI NDK User's Guide, online available: http://www.ti.com/lit/ug/spru523h/spru523h.pdf

[9] OpenCv library: http://opencv.org/

[10] CSL API reference Guide, online available: http://software-dl.ti.com/sdoemb/sdoemb_public_sw/csl/CSL_C6472/latest/index_FDS.html