Implementation of Accelerating Video Preprocessing based on ZYNQ Platform Resource Management

Mengxue Sheng\(^*\), Wanwan Hou\(^2\) and Juchao Jiang\(^3\)

\(^1\) Beijing Engineering Research Center for IoT Software and Systems, Beijing University of Technology, Beijing, 100124, China
\(^2\) Beijing Engineering Research Center for IoT Software and Systems, Beijing University of Technology, Beijing, 100124, China
\(^3\) Beijing Engineering Research Center for IoT Software and Systems, Beijing University of Technology, Beijing, 100124, China

\(^*\)Corresponding author’s e-mail: 171580864@qq.com

Abstract. Embedded AI rised in recent years, AI algorithm deployment platforms need strong computing power, but embedded systems focus on the balance between performance and cost. The contradiction has become a great challenge for the development of embedded AI. This paper analyzes the "ARM+FPGA" architecture and NEON register resources of the ZYNQ series chips introduced by Xilinx, and proposes a method to reasonably schedule the above resources to accelerate video preprocessing when it is used as an edge processing platform for intelligent monitoring terminals. This method makes the video intelligent analysis process of the monitoring terminal achieve real-time.

1. Introduction

With the rapid development of science and technology, the processing units of AI have become increasingly complex. AI support platforms usually need to have excellent computing power. Cloud deployment is a common solution, but it comes with three major issues: energy consumption, bandwidth limitations, and privacy security. In order to meet these needs edge computing and embedded systems have emerged. Its power consumption is lower, functions are more specific, and its requirements on function, reliability, cost, and size are all stricter. Effectively running various AI algorithms on embedded platforms can make the terminal as intelligent as human. It has a wide range of application scenarios, such as live broadcast, short videos, multi-channel video encoding and decoding in the security and educational industries. These applications place high requirements on video processing capabilities. Major chip design manufacturers have designed artificial AI chips, but embedded systems have always been based on the balance between performance and cost. How to complete the real-time processing of video in the embedded system has become the biggest challenge.

2. System resource analysis

Xilinx launched the ZYNQ series of chips using the "ARM+FPGA" architecture. The ARM part is called Processing System (PS), including a dual-core ARM Cortex-A9 processor with NEON registers. The FPGA part is called Programmable logic (PL). The closely cooperation of PS and PL makes the efficient combination of the software programmability of ARM and the hardware programmability of FPGA, so that the system has the advantages of flexibility, rich resources, powerful computing
capabilities, low power and low cost. The synergy makes ZYNQ process images more efficiently and more orderly.

The ARM Cortex-A9 dual-core processor has a university super-standard pipeline, a NEON media processing engine, a floating-point arithmetic unit, and an optimized first-level cache. It achieves unprecedented peak performance while maintaining low power consumption, extending battery life, accelerating media and signal processing functions, and having industry-leading image processing and computing capabilities.

NEON is a 128-bit SIMD architecture extension of the ARM Cortex-A9 series processors. Compared to the SISD, one instruction of SIMD can complete multiple identical operations at the same time. The earth has improved computing efficiency. NEON combines 64-bit and 128-bit SIMD instruction and provides 128-bit wide vector operations. NEON registers come in many forms, including 16x128bit registers (Q0-Q15), 32x64bit registers (D0-D31), and their combination[1].

FPGA devices are a kind of semi-custom circuits in application-specific integrated circuits. They are programmable logic arrays. They have high speed, high accuracy, low power consumption, low cost, short development cycle, rich resources and high configurability, and combine the advantages of DSP and ASCI, can effectively solve the problem that original device is shortage of gate circuits resources, and can achieve parallel and pipeline structures. The basic structure of FPGA includes programmable input and output units, configurable logic blocks, digital clock management modules, embedded block RAM, wiring resources, embedded dedicated hard cores, and underlying embedded functional units. FPGA uses small look-up tables to implement combinational logic. Each look-up table is connected to the input of a D flip-flop. The flip-flops then drive other logic circuits or I/O. The communication between FPGA and CPU relies on the AXI bus, which has the characteristics of high performance, high bandwidth, and low latency. It is defined by ARM and Xilinx and complies with the AXI4.0 protocol specification.

3. System design

This intelligent monitoring terminals use the ZYNQ7010 development board as edge processing platform, deployed intelligent processing unit to analyze video streams collected by cameras in real time. The intelligent processing unit can only analyze the video frame data in semi-planner422 (uyvy422) format, so the video streams collected directly need to be pre-processed into uyvy422 format. To obtain pixel data must go through the following steps:

![Figure 1. Video stream processing process.](image)

This system uses yuv coding. The y component represents brightness information, that is, the gray value, and the uv components represent color information, describing the image color and saturation. There are three main storage format for yuv encoding: yuv444, yuv422 and yuv420. In the yuv420 format, y, u, and v data are stored separately, and every four y components share a pair of uv components. Data in the yuv422 format are also stored separately, and each two y components share a pair of uv components. The amount of data in uyvy422 format is the same as that of yuv422. The difference is that the uv components are stored crosswise.
The intelligent processing unit performs well with a processing rate of 12 frames per second, but the edge processing platform is limited. To fully improve its performance requires reasonable use of various resources. FPGAs have the characteristics of high performance, low power consumption and reconfigurability, however, if try to completely replace CPU with FPGA, FPGA logic resources will be greatly wasted, and FPGA program development costs will be increased. A more practical approach is that the FPGA and the CPU work together, the locality and repetitiveness belong to the FPGA, and the complex belongs to the CPU\cite{4}.

3.1. Dual-core processor decoding design

The decoding operation of this system is implemented by using the soft decoding method of the Ffmpeg video processing library. Ffmpeg supports multi-thread decoding, and the multi-thread decoder is divided into two types of inter-frame and intra-frame. The basic unit of frames is macroblocks. During decoding, if the reference macroblocks are the same, frames can be decoded at the same time using the intra-level multi-threaded decoder. If the same reference frame is required, the inter-level multi-threaded decoder could processes multi-frame code streams simultaneously. The efficiency of intra-frame multi-thread decoding is not up to expectations, so we use inter-frame multi-thread decoding method. Install the Xilinx SDK software development kit on the host machine, set up a cross-compilation environment, and use the cross-compilation chain arm-xilinx-linux-gnueabi-gcc to cross-compile the Ffmpeg library\cite{2}\cite{3}, generate a binary executable file and port it to the ZYNQ7010 development board. Enable the inter-frame multi-thread decoder of Ffmpeg, and set the number of threads to 2, as follows:

\begin{verbatim}
    pCodeCtx->thread_count = 2;
    pCodeCtx->active_thread_type=FP_THREAD_FRAME;
\end{verbatim}

Ffmpeg defaults to the yuv420 data format, stored in memory as follows:

![yuv420 data format](image)

Figure 2. yuv420 data format.

![yuv422 data format](image)

Figure 3. yuv422 data format.

3.2. NEON format conversion design

To convert yuv420 format to uyvy422 format, yuv422 format can be used as a transition. This process is full of lots of simple and repetitive operations and few control codes appear. SIMD characteristics are more suitable. The number of uv components in yuv422 format has doubled compared to yuv420 format. The simplest conversion method is to copy the uv components directly. The method adopted by this system is: for each u component, take the average of itself and the u component in the same column in the next row as the new component u’, and the last row can be copied directly. The v component operation is the same. As follows:

![yuv420 data storage format](image)

Figure 4. yuv420 data storage format.
There are two implementations of the SIMD instruction: Intrinsics and assembly. Assembly method can maximize performance, but writing assembly language is more difficult and improper use will affect performance. Intrinsics uses C language to operate the NEON register, and provides lots of vector operation functions. Its readability and development speed are better, and the performance after optimization is higher relatively\(^5\). This system uses the Intrinsics method to operate the NEON register. When writing the program, the header file “arm_neon.h” needs to be added. The main code snippet is designed as follows:

```c
for (int uv = 0; uv < UV_height_420; uv++) {
    for (int i = 0; i < UV_width_420; i += 16) {
        uint8x16_t data_uv_src1, data_uv_src2, data_uv_dst;
        data_uv_src1 = vldlq_u8(p_U420 + i + uv*UV_width_420);
        data_uv_src2 = vldlq_u8(p_U420 + i + (uv+1)*UV_width_420);
        data_uv_dst = vqaddq_u8(data_uv_src1, data_uv_src2);
        vstlq_u8(p_U420 + i + uv*UV_width + UV_width_420, data_uv_dst);
    }
    // the same as v...)
}
```

The storage in memory is as follows, U1 ’… U8’, V1 ’… V8’ are newly added elements:

![Figure 5. Schematic diagram of yuv422 data format conversion method.](image)

3.3. FPGA Format Conversion IP Core Design

The process of converting yuv422 to uyvy422 requires the uv component to be cross-inserted into the y component. There are lots of element movement operations. Compared with FPAG, ARM processor’s resources and performance is limited, inter-core communication efficiency is low, can’t meet real-time requirements. Hardware is suitable for this kind of repetitive work, so it can be accelerated by FPAG. Convering by FPAG frees up CPU and enables decoding operations and data conversion to proceed in parallel. The element movement process involves communication between PS and PL. ZYNQ7010 uses VDMA for data transmission. VDMA is an IP core for fast data transmission between PS and PL, which can effectively improve the efficiency of video image transmission. In this paper, three VDMAs were used to realize three-way data transfer of y, u, v components. The method is: As for themselves, y, u, and v components are obtained by row. When converting, they are obtained in order of u, y, v, y. The converted data is stored in a cache through a VDMA. The process is shown in the following figure:

![Figure 6. yuv422 data storage.](image)
The amount of data in the uyvy422 format is the same as that in yuv422 format, but the storage method is different. The formatted element data is stored in memory as:

The VDMA temporary in the figure below is the VDMA that buffers the data. The buffered data is sent to the intelligent analysis unit through the BT1120 interface. VDMAO is the VDMA device that actually transfers the buffered data to the BT1120 interface. The FPGA logic resource design diagram is as follow:

FPGA is designed and developed by HLS, which is a set of development tools integrated with the Vivado development environment provided by Xilinx. It uses C, C++, System C and other high-level languages to describe the circuit structure, which greatly reduces the development process and does not require the use of hardware description languages to carry out the design of complex algorithms and timing. It achieved a full-flow design, and can reach the resource utilization rate similar to the traditional design process. The read and write channels of the VDMA data interface are convenient. The AXI bus data can be written into the DDR controller by simply configuring the interface registers. Controlling VDMA can be implemented by mounting the VDMA device address to the local memory through the memory mounting method in program. The main code is as follows:

```c
for (i=0; i < RES_ROW; i++) {
    for (j = 0; j < RES_COL>>1;j++) {
        yuv422_uin >>axis_u;
        yuv422_yin >> axis_y0;
        axis_uvv1.data.range(7, 0)=axis_u.data;
        axis_uvv1.data.range(15, 8)= axis_y0.data;
        uyvy_out<< axis_uvv1;
        yuv422_yin >>axis_v;
        yuv422_yin >> axis_y1;
        axis_uvv2.data.range(7, 0)=axis_v.data;
        axis_uvv2.data.range(15, 8)= axis_y1.data;
        uyvy_out<< axis_uvv2;
    }
}
```

The system frame design diagram is shown below:
4. Results and analysis

In order to verify that scheduling resources reasonably could improve video processing speed, videos with different resolutions were selected as samples, and their frame rates are all 12 fps/s. Three processing methods were selected for comparison. The ARM serial processing method doesn’t do any acceleration operation, from decoding to uyvy422 format conversion is all done by the CPU in sequence. The ARM parallel processing method performs decoding and uyvy422 format conversion in parallel, using an appropriate buffer to buffer the decoded video frames, starting one thread for decoding and another one for format conversion. If the buffer is full, the decoding thread waits, Empty, the format conversion thread waits. The ARM+FPGA method is provided in this article. It can be known from the results that the processing speed of the ARM+FPGA method is basically equal to the video frame rate. The ARM serial processing method is very slow, has reached almost 10 times of the video frame rate. Although the ARM parallel processing method can accelerate the processing, it is still far from being synchronized with the video playback rate. The speed of format conversion is slower than decoding. Therefore, the time of ARM parallel processing method is equivalent to format conversion speed, reached 4 times of ARM+FPGA method.

| Resolution | ARM serial (ms) | ARM parallel (ms) | ARM+FPGA (ms) |
|------------|----------------|------------------|---------------|
| 960*540    | 862.907        | 431.572          | 103.111       |
| 1920*1080  | 879.280        | 437.125          | 109           |
| 1560*1440  | 855.220        | 479.002          | 115.510       |

5. Concluding remarks

This article makes full use of the different characteristics of the NEON register, dual-core ARM Cortex-A9 processor and FPGA resources of the ZYNQ7010 development board to accelerate the preprocessing process that converting the original video stream to uyvy422 pixel data. The dual-core processor enables the Ffmpeg dual-threaded inter-frame decoder. The NEON register accelerates the conversion process of yuv420 to yuv422 format. The conversion of yuv422 to uyvy422 format is performed using FPGA. Reasonable resource scheduling improves real-time performance of video preprocessing.

References

[1] Wang, Y.B. (2018) Efficient Prewitt Edge Detection Algorithm on NEON. Journal of Chinese Computer Systems, 39:2514-2517.
[2] Zhou, Z.M., Wang, X.B., Xu, Y.F., Wang, Y.Z., Wang, S.S. (2019) Design of Real-time Face Detection System Based On Zedboard. Microcontrollers and Embedded Systems,19:53-56.
[3] Chen, G.C., Wu, G.Y. (2019) Design of real-time video processing system based on ZYNQ chip. Modern Electronics Technique, 42:76-78.
[4] Song, M.H., Wang, B., Wang, Z. (2019) Design of embedded experiment platform based on ARM and FPGA. Industrial Instrumentation and Automation, 49:40-43.

[5] Jian, H. (2014) H.265 Decoder Optimization Based on ARM NEON. Video Engineering, 38:102-105.

[6] Peng, X.L., Yu, L., Yu, X. (2019) HLS-based design and optimization methodology for convolutional neural network. Microelectronics and Computer, 36:63-67.