Reusable Embedded Software Platform for Versatile Camera Systems

Wen-Chung Kao, *Member*, IEEE, Chih-Chung Kao, Ching-Kai Lin, Tai-Hua Sun, and Sheng-Yuan Lin, *Member*, IEEE

Abstract — *In this paper we present a reusable embedded software platform that supports multiple functions in digital still cameras (DSCs). The design methodology adopted is to extract the application specific features and device dependant controls from the functional operation modules by well-defined application program interface (API) and device driver interface (DDI), respectively. The proposed system can be reused when some of key hardware components are replaced or parts of operation flow are changed. Seven models of high performance camera as well as some extended applications are developed successfully based on the proposed embedded software platform*[1].

Index Terms —Digital still camera, digital video camera, embedded software platform, real-time system.

I. Introduction

Many integrated signal processors have been proposed for digital still camera (DSC) or digital video camera (DV) applications [1]-[4]. However, very few systematic software platforms are available to support multi-function camera systems. Other than some necessary functions such as capture, playback, display and storage functions, a typical high performance camera system needs to support some attractive features while satisfying miscellaneous timing constraints: (a) continuous shots with instant audio recording, (b) MPEG video/audio compression as well as real-time data streaming, (c) direct print that incorporates picture format and color post-processing, and (d) parsing or preparing configuration files for the captured pictures such that they can be mailed out and be printed automatically from personal computer (PC).

Regarding system performance issues, although many DSCs support fast continuous image capture, most of them do not integrate audio recording at the same time, or do not run auto-exposure (AE), auto-focus (AF) between successive shots. The difficulty is that many tasks are involved, such as AE, AF, pre-capture exposure, raw data readout, key scan, audio recording and image processing. In addition, the system also needs to handle when to start and stop audio recording, immediate capture of next picture, data storage and to avoid accidentally recording the sound of mechanical shutter. Due to the computational power limitation in typical embedded systems, it is difficult to achieve high performance objectives without developing fast algorithms and well-designed system flow control for DSC applications.

In addition to the fancy functions supported in cameras, from the view points of system design engineers the most important features of an ideal embedded software platform include programmability, reusability, calibration and diagnosis capability. Although different camera models possess diverse graphic user interface (GUI) or adopt different key hardware components, many operational functions are common for all cameras, even though they might be operated with different operation sequence. Without a well-organized software platform, codes can not be reused when the product specifications or some hardware components are changed. On the other hand, an ideal software platform must utilize the common functions for both normal operation mode and the engineering mode that can calibrate camera parameters and diagnose the system specifications.

In this paper, an integrated DSC embedded software platform is proposed to address the issues described above. This proposed platform can easily be reused when some of key parts are replaced and image size can be changed at will. The camera system developed on this platform can take still pictures continuously with AE converged. The captured raw image data will be stored in DRAM buffer first, whereas the camera is always ready to take another picture immediately while recording audio. The unconstrained audio recording is started immediately after the picture is taken, and audio data is saved as an embedded bit stream within the same JPEG/EXIF file along with image data. Based on a typical embedded DSC signal processor [1], [5] and other necessary components, the proposed approach has been realized with embedded software only. No extra chip or accelerator is needed to achieve these design requirements.

II. Embedded System Design

A. The Hardware Platform

The hardware architecture of a typical camera system is shown in Fig. 1. The image captured from the CCD/CMOS sensor is stored in SDRAM first and then processed by the signal processor. In addition, audio can be recorded from the microphone and converted into digital signals by the A/D converter. Both image and audio signal are processed by the signal processor, which usually incorporates a microprocessor,

[1] This work was supported by the National Science Council, R.O.C., under Grants NSC93-2218-E-003-003 and NSC94-2218-E-003-002.

Wen-Chung Kao and Chin-Chung Kao are with Institute of Applied Electronic Technology and Department of Industrial Education, National Taiwan Normal University, Taiwan, R.O.C. (email: jungkao@ntnu.edu.tw).

Ching-Kai Lin, Tai-Hua Sun, and Sheng-Yuan Lin is now with Research and Development Department, Foxlink Image Technology Corporation, Taiwan, R.O.C. (email: malcolm.lin@foxlinkimage.com).

Contributed Paper
Manuscript received October 3, 2005

a digital signal processor (DSP) and a live view engine. After processing, the audio/video bit stream data can be stored on the internal flash memory or external flash cards. The color LCD display provides a friendly graphical user interface (GUI) allowing previewing the pictures to be taken. On the other hand, most DSCs support the functions of printing the pictures directly to printers through USB interface in the slave mode, whereas the proposed platform can also print pictures in host mode. This direct print mode enables connection to lower cost simple printers and makes the print operation simpler without having to clumsily set print mode.

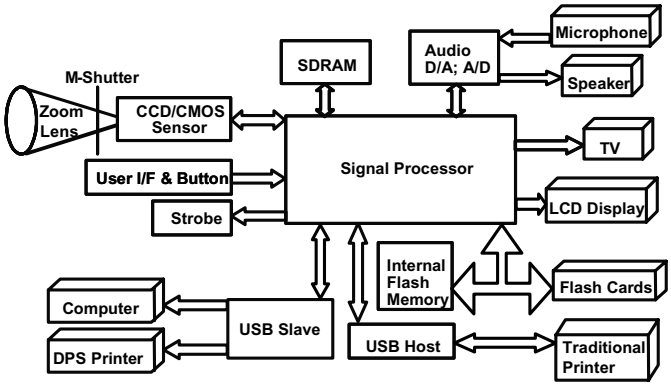

Fig. 1. The hardware architecture of typical camera systems.

B. The Embedded Software Platform

The difficulties of embedded software reuse come from the following facts: (a) The hardware components, such as zoom lens, CCD sensor, color LCD modules, analog front end chip and the type of storage cards, may be replaced with others for different camera models. (b) Even the signal processor itself or real-time operating system (RTOS) may be changed for the next generation product family of higher performance. (c) The flow design of GUI is usually unique for each camera series to differentiate their product definition. (d) The robust camera development system must provide manufacture and diagnosis interfaces such that the design engineers can monitor the device calibration parameters and the system performance indices.

In order to maximize software reuse, the embedded system platform must be abstracted at a level where the basic functional modules can use a device independent interface to the hardware. The application software, which implements various GUI specifications, can use these basic functional modules without considering the implementation details of timing, scheduling and resource allocation on the physical hardware platform.

Fig. 2 shows the proposed embedded software platform for developing versatile camera systems. For improving the programmability and reusability, the whole system is divided into three programming layers and two interfaces: application layer, functional layer, system layer, application programming interface (API) and device driver interface (DDI). To keep the application-specific features apart from the functional operations, the program development of the application layer is based only on the APIs, which are provided by the modules in functional layer. No other lower level functions can be called by the modules in application layer. This prevents the top-level routines from executing lower level operations or accessing hardware resource directly without including some protection mechanisms. Direct function calls may cause the state control to run into abnormal condition that will make the system hang in an unknown state.

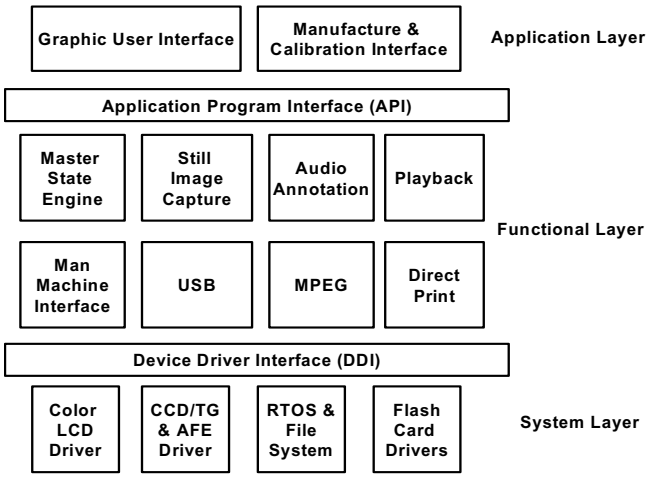

Fig. 2. The proposed embedded software platform.

In the application layer, there are two modules independently running with their own control flows: graphic user interface (GUI) and manufacture/calibration interface (MCI). The state diagram, data flow and dynamic behavior of GUI module are customized by the GUI design specification. The operational flow design of MCI is based on the manufacturing process adopted in production lines. The objectives of MCI are to get high throughput, high product quality and high production reliability. We have designed a common interface between computer and camera for the manufacture/calibration related APIs. Hence the MCI modules can be controlled by computer through USB cable or operated by the camera itself. With these approaches, the production engineers or line operators can monitor the manufacture/calibration progress and collect the production data easily. If the data collection is not needed, the computer on the production line can be saved.

In the functional layer, eight modules are designed to support versatile camera functions: (a) master state engine (MSE), (b) still image capture, (c) audio annotation, (d) playback, (e) man machine interface, (f) USB, (g) MPEG4 video/audio, and (h) direct print. Each module runs with one or more threads, which contain their own message boxes, state diagrams and data flows. Each particular operation, such as still image captures with audio annotation or MPEG4 video/audio recording, is implemented with the APIs supported by one or several function modules. The communications between each functional module are based on

the message boxes and event flag settings. The hardware resource allocation is controlled by semaphores, and all deadlocks conditions must be eliminated.

In the system layer, the major design objectives are hardware abstraction and software programming environment construction. Since many cameras have similar hardware design with different CCD sensor, color LCD display or types of flash cards, we designed these three key components as the configurable or replaceable software modules. Most of the codes in the upper layers can be reused. In addition, by running the system with RTOS, the programming environment can separate the upper level functional operations from the detailed timing scheduling and file allocation mechanisms.

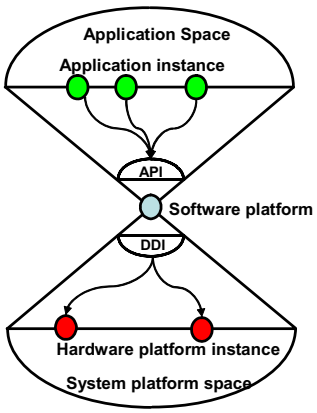

Fig. 3. The concept of the proposed software platform

Fig. 3 summarizes the basic idea of the proposed software platform. Given a new camera system specification, the system designer first maps the specification onto hardware platform by choosing from a suitable family of key components to optimize cost, efficiency, energy consumption and flexibility. Then the codes of the specific hardware drivers to support various hardware components are developed and linked with those pre-defined DDI functions. In the meantime, the application programmers analyze the GUI specification and design their state machines, menu system and artworks. The key of concurrent development is these pre-defined API functions, which abstract the behavior of functional modules and isolate the hardware specific features and their implementation details.

III. Embedded Software Design Methodology

A. Scheduling by RTOS

Although preemptive scheduling is more powerful than non-preemptive one, the analysis of state transition and resource allocation become more difficult than non-preemptive scheduling. The difficulty with preemptive scheduling is due to the fact that the running sequence with preemptive scheduling for a specific operation scenario is unpredictable and a typical camera has several hundreds of operational scenarios. The software debugging and reliability test becomes the bottleneck of the design cycle. From our design experiences, the time period of debugging and verifying thoroughly the software functions as well as reliability test is usually longer than program coding itself. Furthermore, it is also hard to predict and meet the timing constraints if applying preemptive scheduling on the cameras. Although most of RTOSs provide the capability of assigning different task/thread priority, the priority of a thread is not the same in different operational scenarios. It is also very difficult to determine the priority for each thread dynamically to meet the timing constraint for different scenarios.

```
void MasterStateEngineThread()
{
   while (1) {
      Message = WaitMessage () ;
      switch (current_mode) {
         case CaptureMode:
            CaptureModeMsgFunc (Message);
            break ;
         case PlaybackMode:
            PlaybackModeMsgFunc(Message);
            break;
         …
      }
   }
}

void CaptureModeMsgFunc (MSG Message)
{
   switch (CurrentState) {
      case PrecaptureState:
         PrecaptureStateActions (Message);
         break ;
      case ImageCompressionState:
         ImageCompressionStateActions (Message);
       break;
         …
   }
   ChangeState (CaptureMode, CurrentState, Message) ;
}
```

Fig. 4. The program structure for master state engine.

We adopted non-preemptive scheduling while adding some timeout or exception detection mechanisms into the software routines in order to overcome the difficulties of designing, debugging, reliability test and timing constraints problems. The main drawback of non-preemptive scheduling is that the thread may occupy CPU resource if it is trapped on an unknown state or infinite loop. Fortunately, this problem can be easily detected during software development and verification phase. The valid operation scenarios are always defined by the product specification. The design of state machine only need to consider these valid operational scenarios and leaves all un-defined operation scenarios no

actions. The remaining timing constraint problem can be solved by adding some extra codes for particular routines to interrupt the running thread.

The modules located in functional or application layers are allocated by one or more dedicated threads. With the scheduling by RTOS, these threads run independently and their initial state are set to the idle state, which is blocked and can only be invoked by any triggering message sent from other modules. In order to help elucidate the concept of the module design, the programming structure of MSE module is discussed as the example shown in Fig. 4, in which partial codes of the MSE module (*MasterStateEngineThread*) are listed. The module includes an infinite loop, which is always waiting for messages from the RTOS with the routine *WaitMessage*. The message received will be dispatched to the other functions that handle different messages accordingly and perform right actions in current operation mode. This coding style is the fundamental mechanism of task scheduling without using preemptions in the software system. For other modules in application and functional layers, similar programming structure is adopted such that the threads do not occupy computation resource unless they receive any messages. Note that the messages or events are triggered by external hardware signal and internal timer interrupts, or they may be sent by the other threads. The hardware interruptions include any key-press or key-release events, ADC/DAC, real-time clock (RTC), CCD timing generator and DSP subsystem.

B. DSP Subsystem Management

Due to the limitation of DSP program memory, the DSP program is in general loaded dynamically according to the camera operation modes it runs on. In the proposed system, the DSP program is partitioned into three different sections: preview/capture, playback, and MPEG recording. Only one of the program sections is loaded into DSP at a time. The interactions between ARM and DSP are through interrupts, where both ARM and DSP have their corresponding interrupt service routines (ISR) to handle the communication.

Since the software system running in ARM microprocessor is message or event driven, the ISR triggered by DSP will send messages to inform the corresponding modules. Besides, the DSP resource must be managed by semaphores and event flags. As shown in Fig. 5, the module may load their corresponding DSP codes first and then its corresponding thread will be blocked on waiting for the DSP responses. The routine *LoadDSP* ensures that DSP resource is available on receiving a semaphore before it loads new DSP codes. It protects the DSP resource from being occupied by two different threads at the same time. In addition, the event flag was set for controlling the execution of the thread. The RTOS function call *WaitDSPEventFlag* in routine *ProcessStillImage* will block the corresponding thread until DSP finishes the job. This thread can be reactivated only after the corresponding event flag is set again in the ISR. The ISR is triggered by DSP subsystem once the assigned jobs are finished.

```
ProcessStillImage()
{
   LoadDSP (Capture);

   // Triggering DSP to run image processing
   StartImageProcess ();

   // Waiting for the completion event flag set by the ISR
   // of DSP interface
   WaitDSPEventFlag(DSPFree);
}

LoadDSP (DSPSectionName)
{
   // Wait for DSP resource freed
   WaitSemaphore (DSP)

   // Get the DSP resource
   SetSemaphore (DSP);

   // DSPFree is the event flag to indicate whether
   // the DSP is free or not
   ClearEventFlag (DSPFree);
}
```

Fig. 5. The algorithm for loading DSP and resource control by semaphore.

C. Dynamic Memory Allocation

The system may violate real-time constraints if the software uses too many system calls to dynamically allocate memory from the system heap. When the thread executes dynamic memory allocation routines, the current execution thread will enter blocked state for waiting the service of RTOS. Then RTOS searches for suitable free space and allocates the required memory. For memory size and execution efficiency considerations, most of RTOSs adopted in cameras are quite small. These systems usually lack strong enough garbage collection mechanism to compact memory fragments. Memory fragment problem and computation overhead for dynamic memory allocation usually result in system unreliability.

In the proposed platform, the dynamic memory allocation is handled by the embedded software itself instead of by RTOS. One reason is the memory arrangement for miscellaneous operation modes is different. Another reason is the dynamic memory management of RTOS can not achieve good performance on memory allocation and free operations for real-time requirements. To solve the problems, we adopted several memory allocation strategies to the problems mentioned above: (a) analyzing the memory usage for each operation mode such that the memory maps for different operation modes are different, (b) statically assigning memory addresses for larger memory buffers and (c) leaving the remaining space for dynamic memory allocation.

As shown in Fig. 6, all camera operations are assigned on four basic modes with different memory maps in the proposed software platform. In these maps, a few larger memory buffers are assigned on the fixed continuous locations and the remaining space is used as the heap memory for dynamic memory allocation. Since typical RTOS cannot dynamically change the heap size and location, the only way to realize the proposed memory allocation strategies is to develop a dedicated memory management routine that can change the heap size and location instead of RTOS. It is particular useful for USB connectivity mode, because several functions are supported when the camera is connected to personal computer or printer through the USB cable. The required memory size as well as their partitioning is diversified. For example, the memory buffer can be used for processing several printing pictures with different aspect ratios or layout formats. It can also be used for processing user command for the settings of the electronic mails corresponding to pictures. With this USB connectivity mode, the memory can be allocated flexibly with higher efficiency.

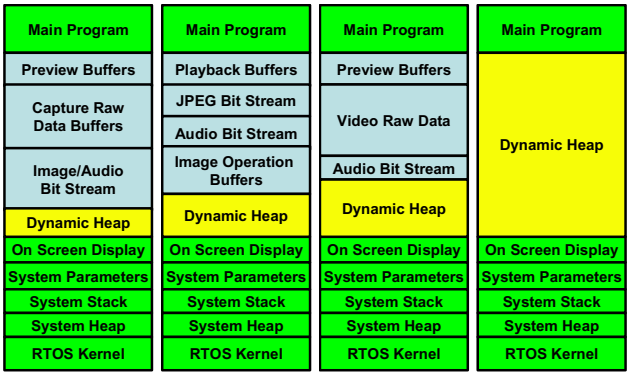

Fig. 6. The memory maps corresponding to the following modes: (a) preview/capture, (b) playback, (c) MPEG, (d) USB connectivity.

D. Application Program Interface and Device Driver Interface

The application program interface (API) provides a unique abstract representation of the function modules with the detailed actions hidden. With an API defined in this way, the application software can easily be reused when new products are developed. It is also possible to change the program modules located in functional layer to accommodate for the new situation, but the modification is usually minimized in the proposed architecture. As shown in Fig. 7, an API does not include the exact execution codes to the functions it need to execute, but only sends messages to the particular module of specified function. The module that receives the message will be invoked and switched into the suitable state to carry out the task. It is worth noting that any application program thread calling the API will enter a blocked state and can only be reactivated after it receives the task finished message.

One problem that might occur in this programming structure is that the thread cannot perform any other tasks before the current task is exactly executed after the thread calls API functions. To solve this problem, another thread named periodical thread, which is automatically invoked by timer interruption every 50 ms, is used for monitoring all camera statuses and can send suitable message to stop the running modules if necessary. This thread only triggers other modules, and does not wait for the response from the others. Therefore, it will not be blocked by any RTOS mechanisms.

```
API_ModuleName_FunctionName()
{
   SendMessage (ModuleName, FunctionName) ;
   WaitMessage (ModuleName, Finished);
}
```

Fig. 7. The program structure of an API.

Similar to API design, the device driver interface (DDI) provides an abstract representation of the hardware device or hardware platform instances. The DDI isolates the device-specific features and unifies the device behaviors such that the program in functional layer can execute their operations without considering the types of devices. For example, the types of storage media may be changed from SD cards to xD cards. The only thing needed to be modified is the low lever driver that directly controls the signal of the storage cards. All the program codes that access storage cards can be reused.

IV. Module Design Guidelines

All modules shown in Fig. 2 share the same hardware sources with their own state machines and data flow. Without carefully analysis and state machine design, a hardware resource may be allocated by two or more threads at the same time, or some resources do not executed efficiently. Hence the embedded software designer must follow good design guidelnes to optimize the system performance. The guidelines include (a) available hardware resource analysis, (b) job scheduling and resource allocation by finite state engine, and (c) background processing and data buffering.

In the following subsections, we use the master engine module that supports continuous shot with real-time audio recording as the example to explain the proposed module design guidelines.

A. Available Hardware Resources Analysis

For each module, the required and available resources must be analyzed first before data flow design and dynamic job scheduling. As shown in Fig. 8, a typical DSC hardware platform includes a CCD sensor, a DRAM buffer and some processing blocks, such as embedded microprocessor, digital signal processor (DSP), DRAM and direct memory access (DMA) controller. The entire system is controlled by the microprocessor, which is supplied with an RTOS to keep schedules among different parts of the system. Other than master engine, the proposed software platform also includes

several threads for key scan, periodic AE/AWB, image processing pipeline, audio processing and DMA control threads. The master state engine must share these hardware resources with other threads.

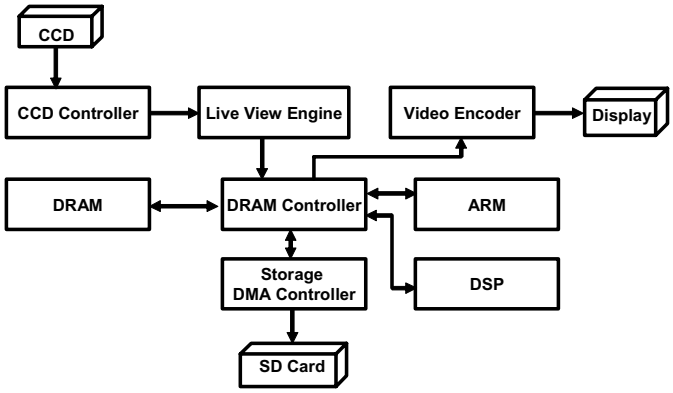

Fig. 8. Typical available hardware resources for processing image/video data.

B. Job Scheduling and Resource Allocation by Finite State Machine

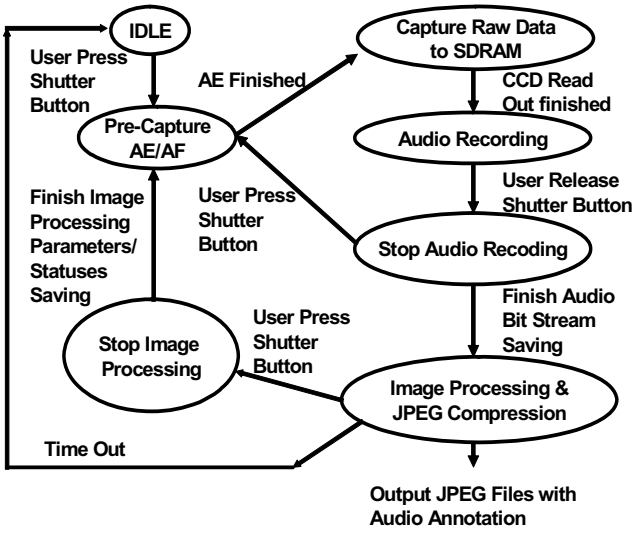

Fig. 9. The state diagram of master state engine.

A complete state machine must have their states definition, input events, state transition as well as output actions. The state diagram of the master engine is shown in Fig. 9. Once the key scan thread detects that shutter button is pressed, it sends one message to activate master engine. The master engine module will switch to pre-capture state after receiving this message. Note that preview AE/AWB, pre-capture AE and image pipeline are executed in the same DSP processor. Only one of them can be executed in DSP at any time. On the other hand, the behavior of periodic AE for preview is different from that of pre-capture AE so that pre-capture AE cannot use the result calculated from periodic AE thread. To overcome the conflicts in DSP resource, as shown in Fig. 10 and Fig. 11, the master engine sends one message to stop periodic AE/AWB thread, interrupts DSP and then switches DSP to execute pre-capture AE metering. After reading CCD raw data, the master engine records audio first before image processing starts. The image processing and JPEG compression are performed after audio recording is finished. It should be noted that each of the running processes can be interrupted if the user presses the shutter button again. The current status for these processes will be saved. The remaining unprocessed data are queued into the buffers until DSP resource is freed.

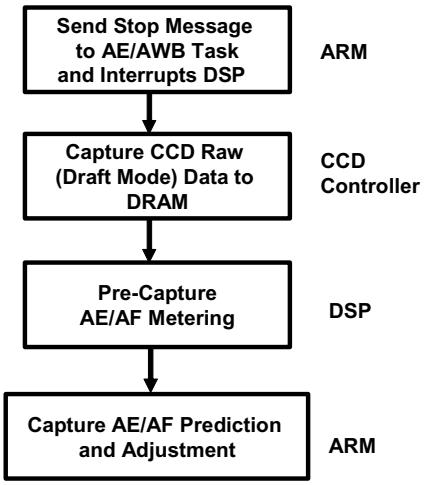

Fig. 10. The flow chart in pre-capture state.

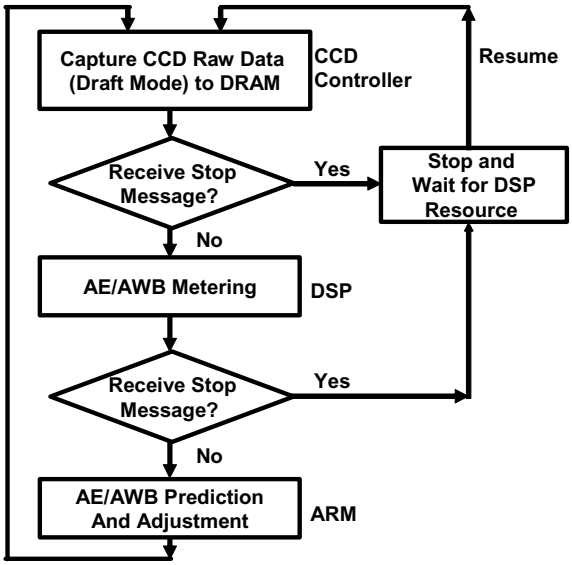

Fig. 11. The flow chart of periodic AE/AWB thread.

C. Background Processing and Data Buffering

The data buffering is the key of background data processing in the camera. In order to support unlimited audio recording length, we adopt two ping-pong buffers for both audio and image processing, as shown in Fig. 12. DSP handles the audio raw data and writes the processed audio bit stream to one of the ping-pong buffers. Once the current buffer is full, DSP interrupts the microprocessor to enable the DMA thread for streaming the data into the storage. DSP continues to process incoming audio data, and DMA thread enables DMA

controller to write the audio bit stream concurrently to the storage card. Similar to audio processing, DSP starts to process the raw image with ping-pong buffers when no extra actions are enabled. Note that some of the unprocessed data may be queued in the raw image buffer as shown in Fig. 13. A dedicated background thread is used for managing the remaining data processing. This is because image processing usually takes longer time, and could be interrupted by the event that user presses the shutter button again.

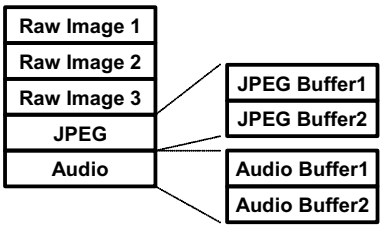

Fig. 12. Memory map for still image capture with real-time audio recording.

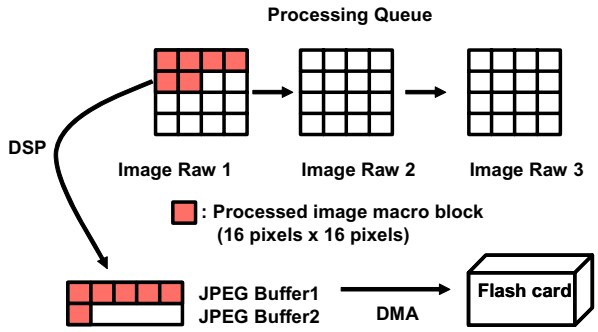

Fig. 13. Still image background processing.

V. Experimental Results and Discussion

The proposed embedded software platform has been implemented and used for developing seven models of commercialized digital still/video cameras. Over than 200,000 lines of C codes are implemented in this platform without counting the sources codes of RTOS and file system. In addition to the necessary functions, the cameras developed with this platform also support direct print via USB host or slave mode. The users can also set some parameters in the camera for sending email to the assigned people in the camera. The built-in extended markup language (XML) parser as well as the proposed dynamic heap technology supports such a function.

The design experiences show that the development cycle for the new product is reduced abruptly compared to our previous software system. In the initial design phase of developing this embedded software, it took over one year with twenty engineers to delivery the first product. But the next several products of the same family, which adopt different GUI, sensor, CLCD, and some other hardware components, were delivered within six months. Lots of the codes were reused for the modules in the functional layer and most of software engineer efforts are put on developing the program in application layer for modifying GUI operation flow. For another camera family that shares the same GUI as one of products but use different hardware components, the development cycle is furthered reduced to less than two month.

Regarding the performance of the module design example called master engine, the cameras designed with this platform can take successive pictures with instant audio recording. The shot-to-shot delay time is less than 0.5 seconds. Unlike traditional continuous shots, the pre-capture process ensures AE convergence even if the lighting conditions are changed abruptly for successive pictures. Based on the method of calculation of AE in current implementation, the final capture AE error is less than 1/8 EV (exposure value) step, which is about 9% errors in linear domain and is easy to be recovered by post image processing, as the dynamic range of the CCD output is usually higher than the JPEG format allows.

Except for camera products, we have also extended application targets to other related systems based on the proposed embedded software platform [6]. The successful cases include homecare system [7], automatic camera calibration system for manufacture line [8], and real-time surveillance with face tracking capability [9]. The platform was also used to study hyper resolution image and extremely wide dynamic range image generation based on multiple image captures. It was also possible to add simple video game functions onto the platform in playback mode. New features can be added easily by following up the proposed design methodologies. The remaining programs can be kept the same or only adjust few parameters and codes.

Acknowledgment

The authors would like to thank the discussion with Dr. Hsien-Che Lee and fully supports from Nelson Hu at Foxlink Image Technology Corporation, Taiwan, R.O.C. The authors would also like to thanks Tai-Yi Chiang, Vic Lin, I-Ru Tsay, Hong-Hsin Wu, Hsien-Wu Huang, Yi-Wen Wang, Bound Tsai, and other embedded software team members in Foxlink Image Technology Corporation for kindly helping to build the system.

References

[1] K. Illgner, H-G Gruber, P. Gelabert, J. Liang,, Y. Yoo, W. Rabadi, R. Talluri, "Programmable DSP platform for digital still cameras", in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, Mar. 1999, pp. 2235-2238.

[2] S. Okada, Y. Matsuda, T. Yamada, and A. Kobayashi, "System on a chip for digital still camera", *IEEE Trans. Consumer Electronics*, vol. 45, no. 3, pp. 584-590, Aug. 1999.

[3] S. Kawamura, "Capturing images with digital still cameras", *IEEE Micro*, pp. 14-19, Nov.-Dec. 1998.

[4] N. Nakano, R. Nishimura, H. Sai, A. Nishizawa and H. Komatsu, "Digital still camera system for megapixel CCD", *IEEE Trans. Consumer Electronics*, vol. 44, no. 3, pp. 581-586, Aug. 1998.

[5] TMS320DM310 Digital Media DSP Technical Reference Manual, Version 2.0, *Texas Instruments*, 2003.

[6] W. C. Kao, S. H. Chen, T. H. Sun, T. Y. Chiang, and S. Y. Lin, "An integrated software architecture for real-time video and audio recording systems," *IEEE Trans. Consumer Electronics*, pp. 879-884, vol. 51, no. 3, Apr, 2005.

[7] W. C. Kao, W. H. Chen, C. K. Yu, and S. Y. Lin, "A real-time system for portable homecare applications," *Proc. IEEE International Symposium Consumer Electronics (ISCE)*, pp. 369-374, Jun, 2005, Macau, 2005.

[8] W. C. Kao, C. M. Hong, and S. Y. Lin, "An automatic calibration system for digital still cameras," *Proc. IEEE International Symposium Consumer Electronics (ISCE)*, pp. 301-306, Macau, Jun, 2005.

[9] W. C. Kao, C. C. Kao, S. H. Chen, Y. W. Hung, and C. H. Huang, "Real-Time Human Face Tracking for Portable Surveillance Systems," accepted by *2005 CACS Automatic Control Conference,* Taiwan, Nov. 2005.

Wen-Chung Kao (M'05) received the B.S. degree in electrical engineering from National Central University, Taiwan, R.O.C. in 1990, and the M.S. and Ph.D. degrees in electrical engineering from National Taiwan University, Taiwan, in 1992 and 1996, respectively. From 1996 to 2000, he was a Department Manager at SoC Technology Center (STC), ERSO, ITRI, Taiwan. From 2000 to 2004, he was an Assistant Vice President at NuCam Corporation, Taiwan, where he was responsible for leading embedded software team to develop digital still/video cameras. In 2004, he became an Assistant Professor at the Department of Industrial Education and Institute of Applied Electronic Technology, National Taiwan Normal University, Taipei, Taiwan. His current research interests include system-on-a-chip (SoC) with embedded software design, digital camera system, homecare system, pattern recognition, and color imaging science.

Chih-Chung Kao is pursuing the M.S. degree in Department of Industrial Education, National Taiwan Normal University, Taipei, Taiwan. His research interests are in the areas of surveillance system and real-time embedded system design.

Ching-Kai Lin is manager of R&D Department at Foxlink Image Technology Corporation, Taipei, Taiwan. He received the B.S. degree in Electronics Engineering from National Taiwan Ocean University, Taiwan, R.O.C. in 1990. He was with Lite-On Technology Corporation during 1995~2000. He joined NuCam/FIT in 2000 and is now in charge of camera module development for mobile phones. His interests are in low level driver and embedded system integration.

Tai-Hua Sun is a Firmware Manager at NuCam Corporation, Taipei, Taiwan. He was with DMP, Taiwan, from 1999 to 2000, working on fingerprint recognition. He joined Foxlink Image Technology, which was named Nucam at the time, in 2000 and now is in charge of digital still camera image pipeline and video system integration. His research interests are in image, video processing, and DSP architecture.

Sheng-Yuan Lin (M'96) is Vice President of R&D Department at Foxlink Image Technology Corporation, Taipei, Taiwan. He received the B.S. and M.S. degrees in electrical engineering from National Taiwan University, Taiwan, R.O.C., in 1983 and 1987, respectively, and the Ph.D. degree in electrical engineering from University of Pennsylvania, Philadelphia, Pennsylvania, in 1994. He was with OES, ITRI, Taiwan, from 1995 to 1997, working on digital camera system characterization. He joined Foxlink Image Technology, which was named Nucam at the time, in 1997 and now is in charge of digital still camera system integration. His research interest is in optics, electronics and image processing integration. He is also a member of SPIE.