International Journal for Housing Science and Its Applications
Publish August 10, 2025. Volume 46, Issue 4 Pages 1036-1045

https://doi.org/10.70517/ijhsa46496

Innovative Architecture Design for Embedded System and Digital Media Convergence

Sheng Jin[1,*]
[1] Faculty of Humanities and Arts, Macau University of Science and Technology, Taipa, Macau, 999078, China
Corresponding authors: (e-mail: kiddyyyy@163.com).

Abstract The development and update iteration of technology provide new possibilities for the integration of embedded systems and digital media. In this paper, on the overall design of the convergence hardware, we propose a hardware framework for embedded multimedia player composed of three main parts: hardware system, embedded real-time operating system and application program. And the basic programming model of computing based on service body-execution flow is used to accelerate the data transfer and cooperative work between processors and improve the overall performance of the embedded system. The streaming data transmission scheme of basic queue is introduced to construct the queue-based streaming data transmission framework by real-time parsing and caching multi-channel data to accomplish the multi-channel streaming transmission task. As a result, the overall design of the embedded system and digital media convergence framework model is completed. The design method is applied to construct the hardware dynamic power consumption model of embedded system and digital media, and the temperature fluctuates in a very small interval at about 42°C and the voltage is maintained at about 1.42V in the actual operation of the model. It shows that the proposed embedded system and digital media integration framework modeling method has high feasibility.

Index Terms embedded system, basic programming model, streaming data transmission framework, multimedia player hardware, dynamic power consumption model

I. Introduction

With the development of the times, the network penetration rate and technology level are increasing, and people's demand for information is also increasing. The public's access to information has gradually changed from text and images to video, such as live broadcasting and online classrooms that have emerged in recent years [1], [2]. The increasing development of communication technology, especially the maturity of multimedia transmission technology, has made people's access to audio and video information increasingly diversified [3], [4]. Among them, digital streaming media transmission is a technology that has attracted much attention. Digital streaming media technology, also called streaming media technology, is to transmit continuous image and audio information to a streaming media server after compression and packaging, and then the server transmits the packaged information to the client sequentially or in real time [5].

Streaming media technology, which generally has unique compression, packaging and distribution methods, has gradually become a pillar technology for video transmission, providing high-quality, low-latency multimedia services to the public [6]-[8]. Compared with the traditional way of downloading files, streaming media technology reduces waiting time and saves storage space [9]. At this stage, streaming media systems are generally deployed on PC-based servers and clients, while in contrast, embedded devices have the advantages of high flexibility and reliability [10]-[12]. Therefore, based on the current situation of using many users and high traffic rates in streaming media services, an innovative architecture based on embedded streaming media server is introduced [13], [14]. Relying on this embedded streaming media server architecture, it is possible to realize the transmission of high-definition (HD) and ultra-high-definition (UHD) videos in the form of low-power and low-distortion, so as to effectively satisfy the needs of the vast number of users to watch high-definition (HD) and ultra-high-definition (UHD) videos [15], [16].

In this paper, we first design the overall hardware structure of the embedded multimedia player and describe the core module structure in it. Secondly, based on the embedded system, it explains the execution process of the programming model of service execution flow with the definition of multiple modules and the operation flow. The content of the queue-based streaming data transmission framework and the functional roles of multiple constituent modules are elaborated again, thus forming a framework model for the integration of embedded systems and digital media. The proposed model is trained, the loss function and accuracy performance are analyzed, and the

correlation coefficients of common events in hardware systems are calculated. Finally, the modeling algorithm is set in the form of comparison to construct the hardware dynamic power consumption model. The hardware operation status is monitored to analyze the performance of the proposed convergence framework model for practical applications.

II. A Framework Model for Convergence of Embedded Systems and Digital Media

II. A.Embedded Multimedia Player

II. A. 1) Hardware structure of embedded multimedia player

The embedded multimedia player proposed in this paper is a real-time embedded system, which consists of three parts from the bottom up, i.e., the hardware system, the embedded real-time operating system and the application program.

In the hardware structure frame of the multimedia player, the hardware system uses Sigma Design's digital multimedia processing chip EM8510 as the CPU, which also serves as the hard decompression function module for video files. The audio input and output part of the chip is WM7831. SDRAM, according to the system characteristics, choose two 8M×32bit SDRAM, one of which is used as the special RAM for the CPU to process the video, and the other one is used as the RAM for the embedded operating system and the application software to run, due to the limitation of the hardware resources, the system chooses one piece of 4Mbyte FLASH to store the embedded operating system and the application software, and the other one is used to store the embedded operating system and the application software. Due to the limitation of hardware resources, a 4Mbyte FLASH is used to store the embedded operating system and application software, while other data are stored on HARDDISK.

II. A. 2) Core module structure

The core module consists of EM8510 processor, FLASH and DRAM. In the design, 2 pieces of Samsung's 8M×32-bit K4S643232H are used for DRAM, one of which is used as the dedicated DRAM for EM8510's MPEG operation, and the other is used as the DRAM for the operation of the operating system and the application software, while the 4M×16-bit MBM29LV650 is used for the FLASH. FLASH shares the address bus and data bus with the system DRAM, while the DRAM for MPEG operation uses the dedicated MPEG data bus and address bus. The core module structure is shown in Figure 1.

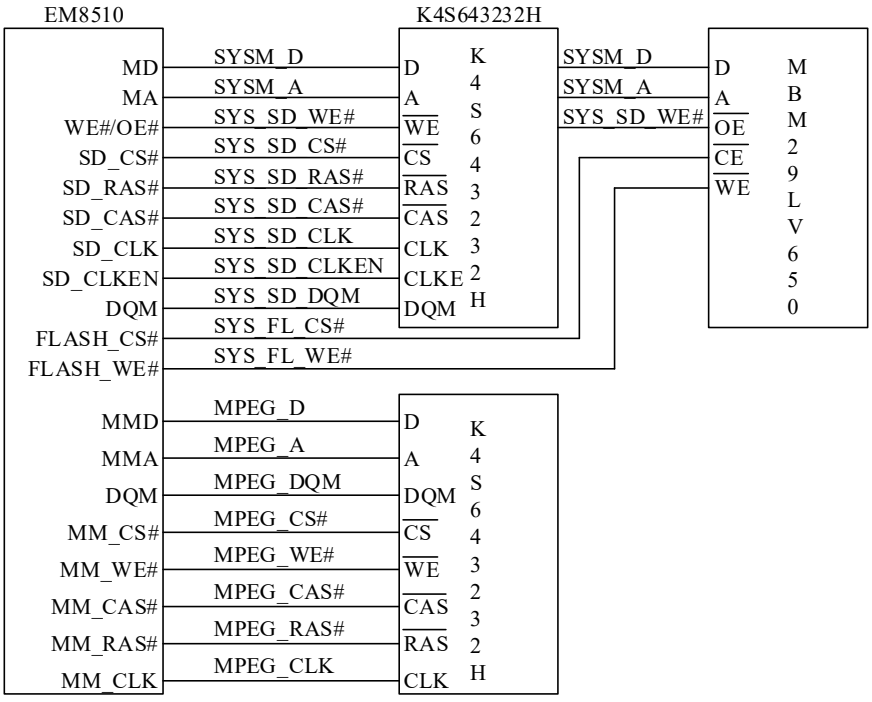

Figure 1: Core module structure

II. B.Basic Programming Model

The application system is abstracted as a data flow processing system and consists of a number of service body-execution flows. The execution process of the application system can be represented as a state migration process of the system, which is maintained by the underlying support environment. The application system is represented as a hexadecimal group as in equation (1):

$$M(S,Q,C,\delta,Q_0,F) \tag{1}$$

where S is the service body, Q is the execution flow, C is the computational resource, δ is the state transfer function, Q_0 is the set of initial states, and F is the termination state. The specific definitions are as follows:

Define service body S: a functional module that can accomplish a specific computational task is called a service body. The code developed by the user for the sub-task algorithm is the service body. The set of system with N service bodies is denoted as equation (2):

$$S=\{S_i \mid i \in N\} \tag{2}$$

A boxed representation of a service body is shown in Figure 2. The underlying support environment of the framework generates executable programs from the code of the service body, which are dynamically mapped to gas pedals and loaded for execution during the runtime of a heterogeneous computing system.

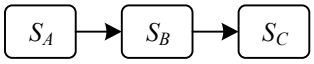

Figure 2: Service body and execution flow

Define execution flow Q: The messages (data structures) passed between service bodies are called execution flows. The set of execution flows in an application system is equation (3):

$$Q=\left\{Q_i^j \mid i,j \in N \wedge Q_i^j \in \square \wedge Q_i^j \le M_i^j\right\} \tag{3}$$

Q_i^j represents the execution flow of $S_i \to S_j$, with i, j taking values in the range of natural numbers [1, N], and N being the total number of service bodies in the system. The value indicates the number of messages on the execution flow that S_i has produced but S_j has not yet consumed. When the leading service body S_i finishes executing and produces a set of outputs, it means that one more message is added to the execution flow, $Q_i^j=1$. When the successor service body S_j feeds that data in, $Q_i^j=0$.

The execution flow is represented by arrows in the figure and by the symbol "×" in the formula. In Fig. 2, the result of the calculation of the service body A is passed to B for further processing, and Q_A^B is denoted as $S_A \times S_B$. The direction of flow of the execution stream can be determined from the symbol ×. The × former term represents the producer of the execution flow, i.e., the execution flow antecedent. The latter term is the consumer, i.e., the execution flow successor. Exchanging the two terms of × will change the direction of flow of the execution flow, and therefore × does not satisfy the exchange law. The completion of the computation of the leading service body will produce an execution flow which has a dual role:

(1) To accomplish the data communication task. The execution flow only represents the communication transfer relationship between service bodies and does not enforce the existence of physical circuits between service bodies.

(2) Driving the successor service body to initiate execution. When the execution of the successor service body is completed, the resulting execution flow starts the successor of the successor, and so on and so forth to drive the completion of the entire computing task. Therefore, it is different from the von Neumann architecture instruction-driven computation, and belongs to the data flow-driven computation.

Defining Buffer Execution Streams: The level flush execution streams are shown in Figure 3. The execution streams contain FIFOs, which can consume one or more data. The maximum legal value of Q_i^j corresponding to a FIFO is defined to be M_i^j, which is equal to this value to represent buffer full.

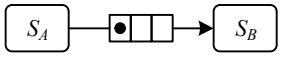

Figure 3: Execution flow in the cascade zone

The subsequently defined resources and transfer functions are maintained by the underlying support environment of the heterogeneous computing framework. Users cannot change these states during development.

Defined resource C : computing component resources in the heterogeneous computing system. Formalized as equation (4):

$$C = \{C_i(R_i) \mid i \in N \wedge C_i \in C_{state} \wedge R_i \in C_{set}\} \tag{4}$$

where the set of resources of the heterogeneous computing system is equation (5):

$$C_{set} = \{GPU, FPGA, CPU, \cdots\} \tag{5}$$

The ellipses can be other types of accelerator resources.

Whether there is a service body mapped to a gas pedal resource is expressed as the state of the resource as in equation (6):

$$C_{state} = \{BUSY, IDLE\} \tag{6}$$

Define the transfer function δ as in equations (7)-(8):

$$\delta\left(C_i = IDLE, \wedge_j (Q_i^j < M_i^j) \wedge \prod_k Q_k^i \neq 0\right) \xrightarrow{\delta} C_i = BUSY \tag{7}$$

$$\delta(C_i = BUSY, C_i = IDLE) \xrightarrow{\delta} (\forall Q_k^i \in Q, Q_k^i --)(\forall Q_i^j \in Q, Q_i^j ++) \tag{8}$$

The meaning of the first transfer function equation (7) is: in the state where the computing resources of the service body S_i are free, when $\wedge_j\left(Q_i^j < M_i^j\right)$ (all the output data of S_i has been entered into the FIFO queue of the execution stream and the queue has free space) and when $\prod_k Q_k^i \neq 0$ (all input ports of S_i are ready), the service body fires, occupies the computational resources and starts execution.

The meaning of the second transfer function Eq. (8) is that when the execution of the service body S_i is completed, i.e., the occupied computational resources change from the BUSY to the IDLE state, the count of each input port decreases and the count of the output ports increases (provided that it is in the range of legal values).

Based on the transfer function, it can be deduced that when a service body satisfies the following three conditions at the same time:

(1) The computing resources of the service body are idle.

(2) Data from all output ports have been transferred to the succeeding execution streams.

(3) The execution streams of all input ports are ready.

At this point the service body starts execution, a process known as the priming mechanism. The state of the entire system can be represented using the state sets C and Q. The state transition process of the service body commanded to operate by the priming mechanism is shown in Fig. 4. To make the description easier, the state of the service body in the figure is represented as equation (9):

$$S_{state} = \{READY, EXEC, BLOCK\} \tag{9}$$

Can be derived from C and Q. The service body generates an execution flow after the computation is completed and at the same time the service body enters a blocking state. Although the computational resources of the service body are free at this time, it cannot start any new task execution if the subsequent FIFO buffer is full. This is because if the execution continues, the resulting new execution stream will overwrite the FIFO, resulting in erroneous results. A service body in the blocking state enters the ready state when the FIFOs of each of its successor execution streams are free.

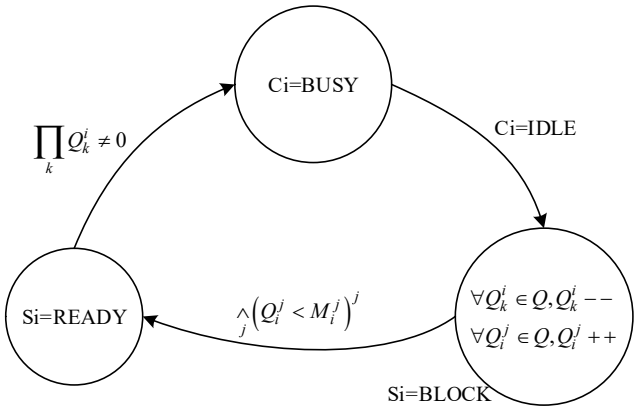

Figure 4: Description of Service Body state transition

A service body in the ready state is input to the execution stream when an execution stream exists in its antecedent to start the next round of computation. F is the termination state, $\forall Q_i^j \mid Q_i^j = 0$. Set Q_0 to be the set of execution streams in the initial state when the system is loaded. From the general state of embedded systems, the system computation is triggered to execute by data generated by external sensors and continues to be processed without interruption. Both the initial state Q_0 and the termination state F are defined by execution streams, and the two may overlap in a longer time dimension. That is, the termination of one phase may in turn be the initial of the next phase.

II. C.Streaming Data Transfer Framework

In the hardware high-performance streaming data transmission architecture of this paper, the hardware is required to transmit data to the buffer opened by the driver, but the driver has a limited number of consecutive buffers for receiving data, so the software running on the processor system is required to establish a dynamic mapping relationship between the buffer and multiple types of data according to the real-time processing requirements, and the hardware selects the corresponding channel from the multi-channel streaming data in real time according to the software commands to transmit the corresponding channel to the specified buffer. The hardware selects the corresponding channel data from the multi-channel stream data according to the software command and transfers it to the specified buffer. Meanwhile, in order to reduce the burden of the transmission interface and the impact of hardware transmission delay, the data should be pre-selected inside the FPGA and transmitted after the buffer data reaches the command size requirement. Based on the above analysis, this chapter proposes a queue-based streaming data transmission scheme, constructs a hardware and software command interaction mechanism based on queue management, and accomplishes the task of multi-channel streaming data transmission through real-time parsing and caching of multi-channel data. Figure 5 shows the streaming data transmission framework, in order to visualize the connection between the processor system PS and the streaming data transmission, the transparent transmission module is removed to virtually connect the processor system and the FPGA together, and the functions of each module will be elaborated in turn below.

In this paper, the command interaction between hardware and software is realized by using submit queue SQ, completion queue CQ, doorbell register and command queue management module. The command queue management module manages doorbell information and queue commands to realize the information transfer with PS. The following two points exist in the command queue management module: the software can release a maximum of one-time commands of the queue depth minus one in the submission queue, and the hardware takes out the commands in batch according to the actual processing tempo, so as to ensure the continuity of the commands. Due to the different bandwidth of each channel stream data, real-time parsing and caching module for each command processing speed is not equal, in order to enable the software can quickly respond to read out the buffer data for the next round of commands to make preparations for the command, allowing the completion of each command in disordered order to return to the completion queue.

The command table management module obtains commands from the command queue management module and issues commands to the real-time parsing and caching module. To ensure that the real-time parsing and caching module caches streaming data for multiple software buffers at the same time, a command table structure is created to control the maximum number of simultaneous caching channels, which is equal to the number of software buffers. The valid stream data channel numbers in the command table represent the channels that need

to be cached at the same time, and the command management module manages the command table according to the pending commands and the idle status of the command table.

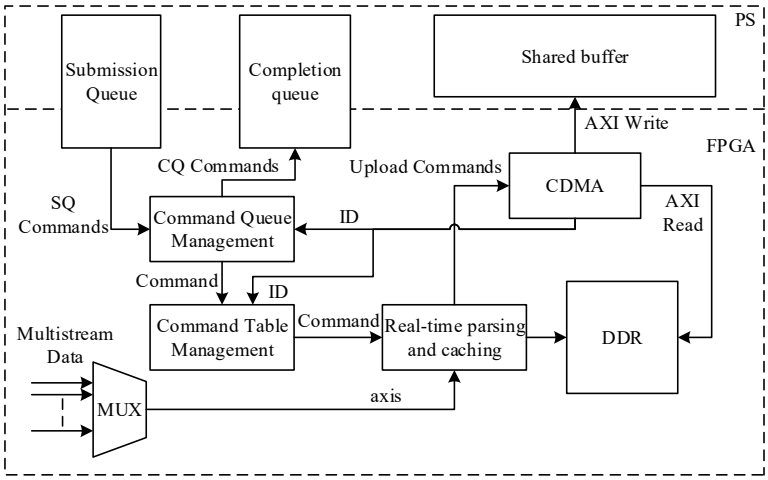

Figure 5: Stream data transmission framework

The real-time parsing and caching module undertakes the task of real-time selection of multi-channel streaming data and controls the data writing to the DDR. Since the data source for data transmission is multi-channel variable-length streaming data, in order to make full use of the PS's task scheduling capability and the FPGA's real-time data processing capability, a double data rate synchronous dynamic random memory DDR is used for large-capacity caching on the FPGA side. In this paper, DDR4SDRAM is used, which has a faster speed compared to the previous three generations and provides DDR controller IP core, encapsulates the device interface logic, and provides only native or AXI interface to the user for reading and writing data directly.The internal space of the FPGADDR is divided equally according to the streaming data channels, and the mapping relationship between the streaming data channel number, the DDR, and PS buffers is shown in Fig. 6. See Figure 6, the division of the DDR internal buffer corresponds to the stream data channel number, and there is no definite mapping relationship between the DDR buffer and the PS buffer. The real-time parsing and caching module writes the data of the matching channel into the corresponding buffer according to the valid commands in the command table, and discards the unmatched data until the caching is completed and then notifies the CDMA module to transfer the corresponding channel data in the DDR buffer to the PS buffer.

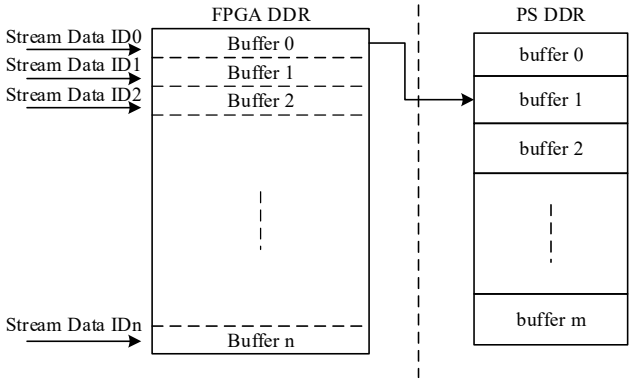

Figure 6: Stream data channel mapping relationship

III. Analysis of the application of the Convergence Framework Model

III. A. Training performance of the model

After the training task is created, jump to the task running page. The loss function and accuracy curves are displayed in Fig. 7. It can be seen that the model accuracy gradually stabilizes at about 70.00% as the number of iterations increases, and the loss value tends to stabilize at 3.25 at about 100 iterations, which combines strong convergence performance with superior overall runtime performance.

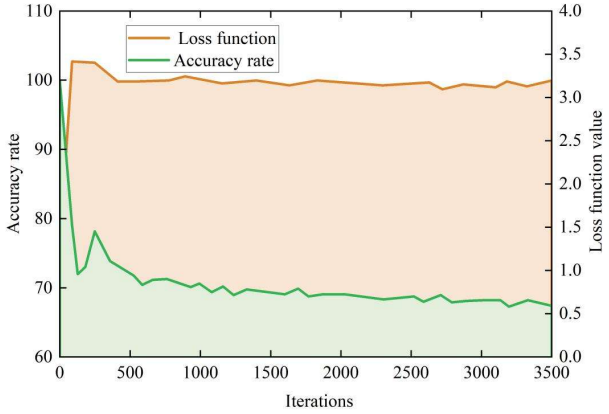

Figure 7: Loss function and accuracy rate

III. B. Construction of dynamic power consumption model for hardware

Based on the overall performance of the convergence framework model and the characteristics of embedded systems and digital media, this paper selects eight operational performance events to construct a dynamic power consumption model for the converged hardware of embedded systems and digital media.

III. B. 1) Screening of operational performance events

In this subsection, Pearson correlation coefficient is used to calculate the correlation between a total of 11 runtime performance events and power consumption, and the 8 events with the highest correlation are selected. The performance events are (I1) block split, (I2) :block, (I3) plug, (I4) emulation-faults, (I5) page-faults, (I6) context _switches, (I7) I2 cache_refill, (I8) L2_cache -, (I9)LI_cache_refill, (I10)L1_ccche-, (I11)cyccles, (I12)branch pred -, and (I13)instructions.The results of the correlation coefficients of the performance events are shown in Fig. 8, in which (I1)block split, (I3)plug, (I7) I2 cache_refill total three events have correlation coefficients lower than 0.500, so they are eliminated.

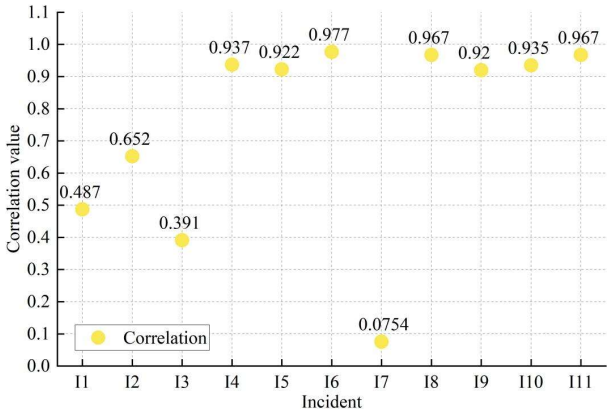

Figure 8: Performance event correlation coefficient

III. B. 2) Modeling Dynamic Power Consumption

In this paper, four different approaches, (M1) Decision Tree, (M2) SVM, (M3) Linear Regression, and (M4) Neural Network, are used to fit the dynamic power consumption model, and the results of the average mean square error of the power consumption model are shown in Fig. 9. Analyzing the time consumed by these four methods for modeling as well as the average error, it can be seen that the average error and the average computation time basically do not change with the number of executions, and the results are relatively stable for methods other than (M4) neural network.

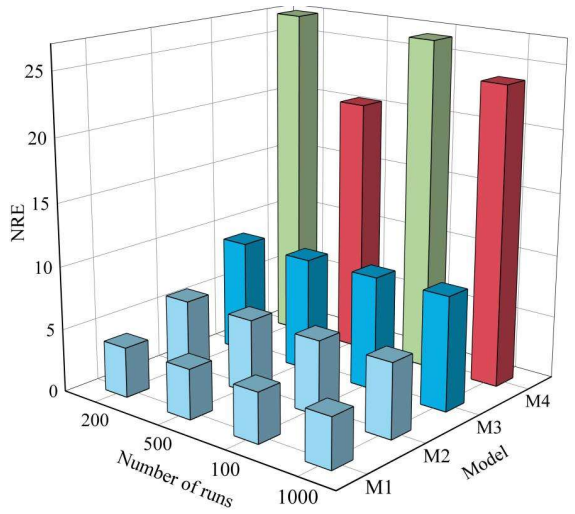

Figure 9: The average mean square error result of the power consumption model

The results of the average computation time for solving the dynamic power consumption model are shown in Figure 10.

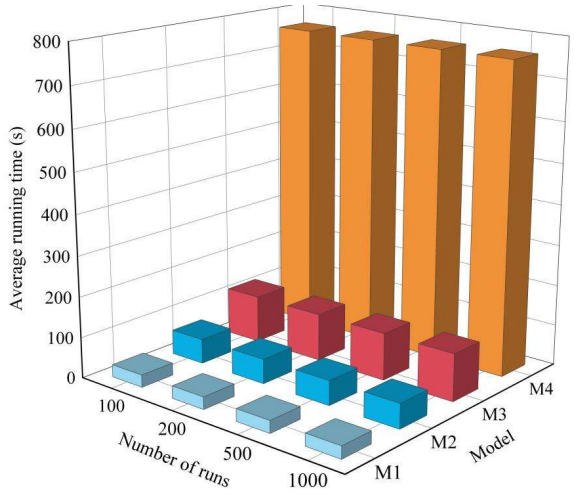

Figure 10: The average computing time for solving the power consumption model

Combining the mean error and mean running time, modeling with the (M1) decision tree was the most effective, with a mean error of less than 5 and a mean running time of only 30.51 s. The (M4) fully-connected neural network was the least effective, with the (M4) fully-connected neural network having the largest mean error (>19.00) as well as the highest mean running time (>700 s). This may be due to the fact that neural networks can be affected by random assignments of model parameters at the initial time, in addition to the fact that performance events are triggered orders of magnitude differently from each other.

III. C. Real-world operational performance of dynamic power consumption modeling

III. C. 1) DMA efficiency test

Hardware DMA commands also have a delay between the total length of each DMA can be improved to enhance the efficiency of the different lengths of the DMA command test measured in Table 1. it can be seen that, in a single DMA data length increases, the CPU side of the waiting time for the command interaction accounted for a smaller proportion of the overall efficiency of the corresponding increase in if a single transmission of the length of the data is too short, then it will spend more time waiting for the command, the transmission efficiency will decline. If the single transmission data length is too short, it will spend more time waiting for commands, and the transmission efficiency will decrease. Therefore, when the control frame is short, the data of multiple frames will be grouped into

packets and then sent to the PCIE link in order to ensure the DMA efficiency. When the length of the transmitted data is 10000KB, the waiting time is 3731 milliseconds and the transmission efficiency reaches 77%.

Table 1: DMA efficiency under different lengths

Data length (KB)	Time consumption for ten thousand transmissions (milliseconds)	Transfer efficiency (%)
100	448	73
200	807	75
500	1540	76
1000	3724	77
2000	7390	77
10000	3731	77

III. C. 2) Hardware operating status

The results of hardware status monitoring from 11:23:00 to 11:24:00 are shown in Fig. 11. Overall, the hardware status is relatively normal and stable, with the temperature fluctuating in a very small range around 42°C, and the voltage remaining at around 1.42V. In addition, we can also see some err signals of the system with or without error reporting, as well as the number of bursts of the state machine's para_num signal and comp_num signal on behalf of the number of detections have been received and sent, has been incremental means that the system is working normally.

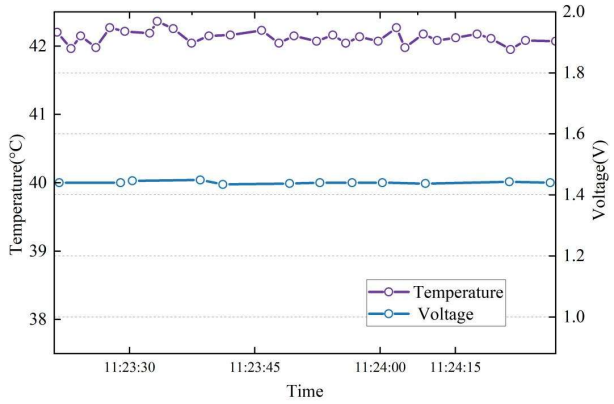

Figure 11: Hardware status monitoring results

IV. Conclusion

In this paper, within the framework of the designed embedded multimedia player hardware, the programming model of service execution flow is used to adapt and optimize the hardware processor's multiple data transmission acceleration needs and the efficient collaboration between the processing, and the streaming data transmission framework is established to realize the real-time parsing and caching of multi-channel streaming data, and the convergence framework model of embedded system and digital media is constructed in an innovative way.

The designed convergence framework model has strong convergence, and the accuracy is stable at about 70.00%, and the loss value is stable at 3.25. For the embedded multimedia player hardware designed by this convergence framework model, the decision tree algorithm is chosen as the modeling method to construct the dynamic power consumption model. Among them, when the length of transmitted data is 10000KB, the transmission efficiency reaches 77%. And the overall hardware operation state is relatively stable, in the monitoring within 2 minutes, the temperature fluctuates in a very small interval around 42°C, and the voltage is maintained at about 1.42V.

Acknowledgement

This work was supported by the Cultural Heritage Science and Technology Protection Project of Zhejiang Province (Project Number: 2020015).

References

[1] Bateman, J. A. (2021). What are digital media?. Discourse, Context & Media, 41, 100502.

[2] Giuffrida, G., Mazzeo Rinaldi, F., & Russo, A. (2021, October). Analyzing communication broadcasting in the digital space. In International Conference on Machine Learning, Optimization, and Data Science (pp. 518-530). Cham: Springer International Publishing.
[3] Colbjørnsen, T. (2021). The streaming network: Conceptualizing distribution economy, technology, and power in streaming media services. Convergence, 27(5), 1264-1287.
[4] Hsu, T. H., & Tung, Y. M. (2020). A social-aware P2P video transmission strategy for multimedia IoT devices. IEEE Access, 8, 95574-95584.
[5] Ozgun, A., & Treske, A. (2025). Streaming Media Platforms and Film/Video Arts Distribution. Markets, Globalization & Development Review, 9(4).
[6] Barakabitze, A. A., Barman, N., Ahmad, A., Zadtootaghaj, S., Sun, L., Martini, M. G., & Atzori, L. (2019). QoE management of multimedia streaming services in future networks: A tutorial and survey. IEEE Communications Surveys & Tutorials, 22(1), 526-565.
[7] Bentaleb, A., Taani, B., Begen, A. C., Timmerer, C., & Zimmermann, R. (2018). A survey on bitrate adaptation schemes for streaming media over HTTP. IEEE Communications Surveys & Tutorials, 21(1), 562-585.
[8] Yaqoob, A., Bi, T., & Muntean, G. M. (2020). A survey on adaptive 360 video streaming: Solutions, challenges and opportunities. IEEE Communications Surveys & Tutorials, 22(4), 2801-2838.
[9] Kua, J., Armitage, G., & Branch, P. (2017). A survey of rate adaptation techniques for dynamic adaptive streaming over HTTP. IEEE Communications Surveys & Tutorials, 19(3), 1842-1866.
[10] Koulamas, C., & Lazarescu, M. T. (2018). Real-time embedded systems: Present and future. Electronics, 7(9), 205.
[11] Wang, X. (2020). An overview of efficient deep learning on embedded systems. Handbook Of Pattern Recognition And Computer Vision, 107-117.
[12] Pan, T., & Zhu, Y. (2018). Designing Embedded Systems with Arduino. Designing Embedded Systems with Arduino.
[13] Duraipandian, I., & Manzke, R. (2018). Synchronized real time audio streaming over ethernet in embedded systems. In 2018 International Symposium on Ambient Intelligence and Embedded Systems, AMIES (pp. 1-4).
[14] Jadhav, A. S., & Sudarshan, D. (2016). Real time embedded video streaming using Raspberry Pi. International Journal of Innovative Research in Science, Engineering and Technology, 5(11), 19315-19320.
[15] Álvarez Ariza, J. (2019). Dscblocks: An open-source platform for learning embedded systems based on algorithm visualizations and digital signal controllers. Electronics, 8(2), 228.
[16] Lulic, D., Jovanovic, N., & Radonjic, M. (2017, September). Flexible multimedia player architecture for embedded systems. In 2017 IEEE 7th International Conference on Consumer Electronics-Berlin (ICCE-Berlin) (pp. 181-183). IEEE.