System Architecture Synthesis of Embedded System for Multimedia Applications

Shiann-Rong Kuang, Kuo-Chin Huang, Ju-Kai Teng, and Jin-Lin Liu

Department of Computer Science Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan, ROC

Abstract*-This paper presents a tool to quickly generate the system architecture of embedded system for complex multimedia applications. Given the system specifications and constraints of a particular multimedia system, the tool not only partitions and maps each computation task of the multimedia application into a hardware/software component, but also schedules and pipelines the execution of these computation tasks. Because of considering the execution time, area, and communication time of each computation task carefully, the proposed tool can quickly construct the feasible system architecture with minimum area for multimedia applications while meeting the real-time requirement.*

Keywords: Embedded System, Hardware/software partitioning, Multimedia Applications.

1. Introduction

Due to the emergence of high-performance digital consumer electronics, from video games to set-top boxes, embedded computers are rapidly growing in the computer market. An important tend in embedded systems is the use of processors cores together with application-specific integrated circuits (ASICs) to meet application's functional and performance requirements. This tend is especially obvious for those embedded multimedia applications with complex functions and computations, e.g. MP3, JPEG, MPEG, etc. To achieve high-performance, low-cost, and flexible implementations for these applications, most of them are implemented by heterogeneous architectures that consist of the embedded software (SW) processor cores and application-specific hardware (HW) circuits. The SW processors reduce the cost and provide the flexibility of the system, and the HW components perform the computation-intensive tasks of the application quickly and enhance the performance of the system.

However, the primary goal of embedded system is usually meeting the performance need at a minimum price, rather than achieving higher performance. Although using many HW components can speed up the running time of multimedia applications, it has the disadvantages of high cost, great depletion of capacity, and big measure of area. Then the drawbacks make it can't apply to embedded system [1]. Therefore, it is necessary to carefully decide which computation task executed by SW component and which computation task implemented by HW component for the sake of keeping the balance between cost and efficiency. Besides, it is also necessary to specify the execution order of these components, so as to speed up the period of system development.

Unfortunately, the process of HW/SW partitioning and scheduling is a complex optimization problem. Consequently, it is important to develop an optimization tool that can automatically partition and map the computing tasks inside the multimedia applications to different HW/SW components in embedded system, so that the hardware cost can be minimized without violating the system constraints. In recent years, many literatures have proposed HW/SW partitioning methods to rapidly establish the system architecture of embedded system [1-6]. However, most of them [2, 3] performed HW/SW partitioning based on a fixed system architecture and communication topology, and thus better system architectures cannot be explored. Some of the previous methods [6] didn't schedule the execution of computation tasks in pipeline such that they cannot construct efficient system architecture for multimedia applications. Other previous methods [4, 5] didn't take the data transmission time between HW and SW into consideration, so that the resulted system architecture didn't really satisfy the system timing constraint. Moreover, these previous methods just fulfilled the system constraints but didn't provide flexibility to meet the special

requirement of users. For example, users may hope to assign a computation task to an available intellectually property (IP) to reduce the design time.

In this paper, we propose a HW/SW partitioning and pipelined scheduling tool to build a system architecture, which consists of SW processor cores, HW components, and a feasible bus system, by the specification of multimedia applications and constraints. The proposed tool minimizes the required area of the system architecture while satisfying time constraint. The number of SW processor cores and HW components in our system architecture is variable and the bus is constructed by the result of partitioning and scheduling. In addition, the proposed tool can interact with users and take the time of data transmission into account such that the resulted system architecture can conform to the real situation.

The remainder of this paper is organized as follows. Section II shows the overview of the proposed system architecture synthesizer and interprets how to establish control flow graph (CFG) by the system specification, and how to implement the HW/SW estimator. Section III explains our technique for HW/SW partitioning and pipelined scheduling. Section IV applies our system architecture synthesizer to some multimedia applications and discusses the experimental results. And finally we give a conclusion.

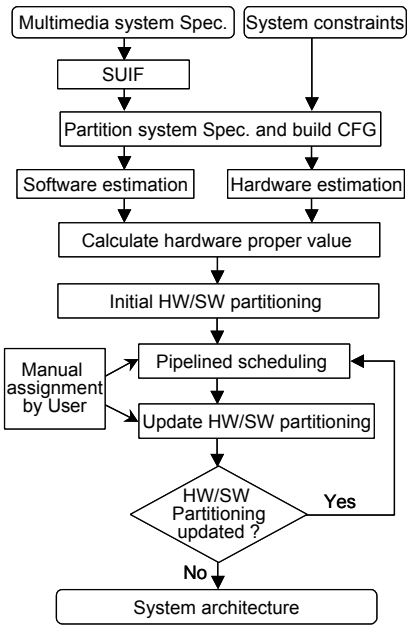


Fig. 1. Flowchart of our system architecture synthesizer

2. Overview of the Proposed Synthesizer

Given a system specification and constraints of a multimedia application, our system architecture synthesizer uses iterative partitioning and pipelined scheduling to obtain a system architecture, such that the timing constraint is satisfied and the total area of the system architecture is minimized.

The flowchart of the proposed system architecture synthesizer is shown in Fig. 1. The SUIF compiler [7], which developed by Stanford University, is used in our synthesizer to parse and check the system specification, exploit the parallelism, and reduce redundant computing tasks of system functions. The details of every procedure of the flowchart are explained in the following sections.

2.1. Control Flow Graph

Our system architecture synthesizer receives a system specification that is an application program written by high-level language, such as C, C++ etc. After the system specification has passed through SUIF, system architecture synthesizer analyzes the data dependencies based on the program language and then transfers the system specifications into the intermediate representation named control flow graph (CFG). An example of CFG is shown in Fig. 2. Nodes in CFG denote the functions of system specification and the numerous computation tasks that operate the function are included. Edges in CFG denote the data dependencies between system functions. Generally, each system function is a subroutine of the system program. Our synthesizer finds subroutines and parameters passed between them to create the corresponding CFG. After finishing the above process, the synthesizer must estimate the cost of HW/SW implementation for every node in CFG, so as to attain the information of HW/SW implementation for functions in multimedia system, e.g. execution time, area, etc. The information will be necessary to HW/SW partitioning.

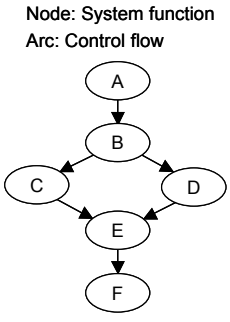


Fig. 2. Control flow graph

2.2. Software estimator

The purpose of software estimation is to estimate the execution time of each node in CFG when it is implemented by SW processor code. Since our synthesizer adopts ARM processor [9] as the default processor, ARM Developer Suite (ADS) v1.2 provided by ARM company is necessary to software estimation. The main task of ADS is to compile each subroutine in the node of CFG first, and then run ARM Symbolic Debugger (armsd) simulator to simulate the subroutine to obtain the execution time on ARM processor in terms of clock cycle number. The armsd simulator can run on different OS such as Windows, Linux, Solaris, etc. To accomplish the software estimation automatically, we establish a script file to quickly profile each subroutine in the node of CFG and collect the generated information.

2.3. Hardware estimator

The purpose of hardware estimation is to estimate the execution time and required area of every node in CFG when it is implemented by hardware components. To achieve the goal, the hardware estimator must schedule the operations in each node of CFG and bind them to available function units by using high-level synthesis. The Synopsis DesignWare library is adopted in our hardware estimator to estimate the execution time and area of needed functional units.

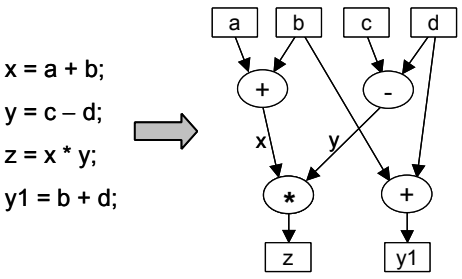


Fig. 3. Data flow graph

To efficiently analyze, schedule, and bind the operations in each node of CFG, the subroutine is transformed into a data flow graph (DFG) of computation tasks. A DFG is shown in Fig. 3. In DFG, nodes denote operations in subroutine and edges denote the data dependencies. Sequentially, we could schedule and bind the DFG to estimate the execution time and required area. In order to take the resource constraint into consideration and obtain the estimation result quickly, we use the list-scheduling algorithm to implement the hardware estimator. List-scheduling algorithm takes some ready nodes form the priority list and schedules them under the resource constraints. It can schedule the ready nodes when hardware resource is enough. Priority list will sort the ready nodes by the priority function. Accordingly, the priority function solves the problem of share resource between operations. When these ready operations conflict with the using resources, the high priority operations will be scheduled, and low priority operations will be delayed to the next or later scheduled steps.

When the software and hardware estimations have finished, the system architecture synthesizer then iteratively performs HW/SW partitioning and pipelined scheduling.

3. HW/SW Partitioning and Pipelined Scheduling

HW/SW partitioning evaluates and maps each system function to an application-specific HW component or a SW processor core. The result of partition is influenced by three factors: execute time of HW/SW, area of HW/SW, and communication time between HW and SW components. In terms of three affected factors, we define and describe the time factor, area factor, and communication factor for each node v in CFG as follows.

Time factor $T(v)$ indicates the suitability when using hardware or software to implement the system function of node v from viewpoint of time. The lager $T(v)$ means the execution time of node v implemented by hardware is shorter than the one implemented by a processor, and node v is more proper implemented by hardware. In contrast, implementing node v by a processor is better if $T(v)$ is smaller.

Area factor $A(v)$ takes the correlation between the hardware area of node v and the total hardware area into consideration when using hardware to implement system functions. Bigger $A(v)$ means that increased hardware area is less when node v implemented by hardware. If hardware area required to implement node v are bigger than system area constraint, the $A(v)$ is set to zero.

Communication factor $C(v)$ means the ratio of data transmission time when implementing node v by using SW processor core or hardware component. Accordingly, the data transmission time of node v implemented by processor is longer than the one implemented by hardware, $C(v)$ is bigger and node v is more proper to be implemented by hardware. Otherwise, hardware implementation needs more data transmission time than software implementation, and node v is more proper to be implemented by software.

Every node *v* in CFG would have a hardware proper value $HP(v) = T(v) \times A(v) \times C(v)$. The bigger the $HP(v)$ is, the more proper the node *v* is implemented by hardware. In order to keep time balance between hardware and software, the nodes in CFG are sequentially and decreasingly assigned to hardware by the hardware proper value, and the execution time of hardware implementation is summed up when hardware area constraint (denoted as A_{const}) doesn't be violated. If the execution time of hardware implementation (denoted as T_H) is longer than the one of software implementation (denoted as T_S), the nodes in CFD are sequentially and increasingly assigned to software by their hardware proper values. The procedure is repeated until all nodes in CFG are assigned to HW or SW and the HW/SW partitioning is finished.

Fig. 4 shows the algorithm for obtaining an initial HW/SW partitioning. Let V be the set of nodes in CFG whose elements has been sorted in descending order by node's hardware proper value, and v_i be the i^{th} element of V. $M[v_j]$ in Fig. 4 records the partitioning result of node v_j. $M[v_j]$ could be HW or SW, and it denotes that v_j is assigned to hardware or software, respectively. $M[v_j]$.area and $M[v_j]$.time denote the required area and execution time when v_j has been assigned to HW or SW. The initial HW/SW partitioning is probably adjusted by the result of the following pipelined scheduling.

```
Algorithm: Initial HW/SW partitioning
Begin
  For (every node in CFG)
    Calculate hardware proper value HP;
  End For
  Sort all nodes in descending order by node's HP;
  N = the number of nodes in CFG;
  k = N – 1;    T_H = T_S = A_H = 0;    i = j =0;
  While ( i < N)
    If (T_H ≤ T_S and (A_H+ M[v_j].area) ≤ A_const)
      M[v_j] = HW;
      T_H = T_H + M[v_j].time;
      A_H = A_H + M[v_j].area;
      i = i + 1;   j = j + 1;
    End If
    If (T_S ≤ T_H or (A_H+M[v_k].area) > A_const)
      M[v_k] = SW;
      T_S = T_S + M[v_j].time;
      i = i + 1;   k = k – 1;
    End If
  End While
  Current_HW_node = j – 1;
End
```

Fig. 4. Algorithm of initial HW/SW partitioning

After each node in CFG is assigned to hardware or software, the CFG then is reconstructed by inserting extra communication nodes and is scheduled in pipeline to determine the execution order of every node. When node is assigned to HW, extra data transmission time between HW and SW probably is required. Therefore, extra communication nodes labeled with communication time are added into CFG to ensure that pipelined scheduling can really satisfy the timing constraint. We perform the pipelined scheduling of reconstructed CFG also based on the list-scheduling algorithm. The main task of the pipelined scheduling includes assigning every node *v* to a pipeline stage and a time slot in pipelined stage, so as to the all ancestors of node *v* could complete its execution before node *v* in former or the same pipeline stage. Besides, if node *v* is implemented by software, it is necessary to make sure that the processor to execute the task shouldn't execute other node during the time slot of node *v*. If node *v* is implemented by hardware, it is necessary to make sure that the other nodes that sharing the hardware shouldn't be executed during time slot of node *v*. The algorithm to perform pipelined scheduling is shown in Fig. 5.

```
Algorithm: Pipelined scheduling
Begin
  ready_list[ ] <= the first node;
  do
    Update ready_list[ ];
    do
      If (node type is SW)
        Find an idle processor;
        If (all processor is busy)
          Pipelined scheduling fail;   Exit;
        Else
          Assign node to processor, suitable time interval
                and pipe stage;
        End If
      Else
        Assign hardware to time interval and pipe stage;
      End If
      Remove node from ready_list[ ];
    While (ready list isn't null)
  While (any node isn't scheduled)
End
```

Fig. 5. Algorithm of pipelined scheduling

When the pipelined scheduling is obtained successfully, we can update the HW/SW partitioning by reassigning the node Current_HW_node in V to SW and performing pipelined scheduling again. If feasible pipelined scheduling can still be obtained, the hardware area can be reduced and we can repeat the procedure by reassigning the node Current_HW_node−1 to SW and then performing pipelined scheduling until no feasible pipelined scheduling can be found. Another method to degrade the system cost is trying to reduce the number of available processors. On the other hand, when the pipelined scheduling cannot be obtained successfully, we can try to reassign the node specified by Current_HW_node+1 to HW and

perform pipelined scheduling again. The procedure can be repeated until a feasible pipelined scheduling is found or A_H is larger than A_{const}.

Finally, system architecture synthesizer will output the HW/SW assignment, pipelined scheduling, and system architecture. If the result generated by automation can't satisfy the requirement of users, users can assign a system function to hardware or software and do the pipelined scheduling manually. The step can be repeated until the most proper system architecture is obtained.

4. Experimental Results

We use JPEG2000 encoding system to inspect the ability of our system architecture synthesizer. The main tasks of JPEG2000 encoding system comprise wavelet transformation and EBCOT. We assume that the encoding system adopted in a digital camera can continuously capture pictures. The system specification of JPEG2000 encoding system is first split and then analyzed by our synthesizer. The corresponding CFG is shown in Fig. 6(a). So far as hardware estimation is concerned, the csa adder and nbw multiplier of 0.35um technology are utilized to estimate in hardware library. The execution time of wavelet transformation and EBCOT implemented by hardware is 0.39 sec and 0.56 sec respectively through the hardware estimator. A 32-bit RISC ARM920T processor is adopted for software estimation. The result of software estimator indicates that the time of executing wavelet transformation is 0.52 sec, and the time of executing EBCOT is 58.6 sec. The hardware proper values of system functions are shown in Fig. 6(b). Finally, our synthesizer uses these hardware proper values to perform partitioning and pipelined scheduling at each node. The comparison of system architectures and times between the JPEG2000 systems implemented without and with pipelined scheduling is shown in Fig. 7.

Except for JPEG2000 encoding system, we adopt the MP3 (MPEG Audio Layer 3) decoding system [8] and WaveVideo system [10] to further test the capability of our synthesizer. Fig. 8 shows the CFG of MP3 decoding system generated by our synthesizer. The CFG contains 9 nodes, and the result of initial HW/SW partitioning is shown in Table 1. Under the time constraint of 12500 ns for per frame, we first generate a feasible pipelined scheduling manually by interacting with the proposed synthesizer. The resulted pipelined scheduling uses two processors and can decode a frame per 11343 ns. Next, another pipelined scheduling is automatically generated by our synthesizer. The pipelined scheduling only uses one processor and can decode a frame per 11893 ns. Both of the pipelined scheduling results meet the time constraint of 12500 ns, but the result generated by our synthesizer automatically has lower cost of system architecture.

For the WaveVideo example, the CFG generated by our synthesizer contains 23 nodes (function blocks). In order to meet the time constraint (30 frames/sec), there are 11 function blocks implemented by hardware and 12 function blocks executed by one processor running at 200 MHz. The proposed synthesizer determines the system architectures of these examples only in few seconds. In the further, we will develop an exhaustive approach to generate the optimal system architecture and make a comparison between the exhaustive approach and the current heuristic approach.

5. Conclusion

This paper has proposed a system architecture synthesizer to rapidly decide the system architectures of embedded system for multimedia applications. The proposed synthesizer minimized the required area of the system architecture while satisfying time constraint. Moreover, our synthesizer can also interact with users and take the time of data transmission into account such that the resulted system architecture can conform to the real situation. As a result, our synthesizer can significantly reduce time and manpower for deciding the system architecture of embedded system for multimedia applications.

Acknowledgment

This work was supported in part by the National Science Council, R.O.C., under Grant NSC-92-2220-E-110-008.

References

[1] F. Balarin et al., *Hardware-Software Co-Design of Embedded Systems: A Polis Approach*. Norwell, MA: Kluwer, 1997.

[2] Asawaree Kalavade and P. A. Subrahmanyam, "Hardware/Software Partitioning for Multifunction Systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 17, No. 9, Sep. 1998.

[3] Karam S. Chatha and Ranga Vemuri, "Hardware-Software Partitioning and Pipelined Scheduling of Transformative Application", *IEEE Trans. on VLSI Systems*, Vol. 10, No. 3, pp. 193-208, June 2002.

[4] F. Balarin, L. Lavagno, P. Murthy, and A. Sangiovanni-Vincentelli, "Scheduling for embedded real-time systems," *IEEE Design Test Comput.*, Jan.-Mar. 1998.

[5] S. Bakshi and D. D. Gajski, "Partitioning and Pipelining for Performance - Constrained Hardware/Software Systems," *IEEE Trans. on VLSI Systems*, Vol. 7, No. 4, pp. 419-432, Dec. 1999.

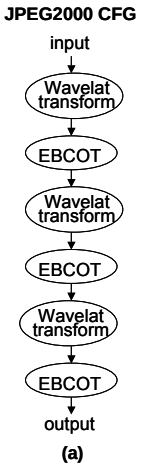
[6] Byoung-Woon Kim and Chong-Min Kyung, "Exploiting Intellectual Properties With Imprecise Design Costs for System-on-Chip Synthesis," *IEEE Trans. on VLSI Systems*, Vol. 10, No. 3, pp.240-252, 2002.

[7] Overview of the SUIF compiler, http://suif.stanford.edu

[8] ISO/IEC 11172-3. *Information Technology - Coding of Moving Pictures and Associated Audio for Digitial Storage Media at up to about 1.5Mbit/s - Part 3: Audio*, 1993.

[9] http://www.arm.com

[10] G. Fankhauser, M. Dasen, N. Weiler, B. Plattner, and B. Stiller, "WaveVideo - An Integrated Approach to Adaptive Wireless Video," *ACM Monet, Special Issue on Adaptive Mobile Networking and Computing*, Vol. 4, No. 4, pp. 255-271, 1999.



	Hardware proper value	Type
Wavelet transform	0	Software
EBCOT	1.67	Hardware

(b)

Fig. 6. (a) JPEG2000 CFG (b) Hardware proper value and component type by partitioning

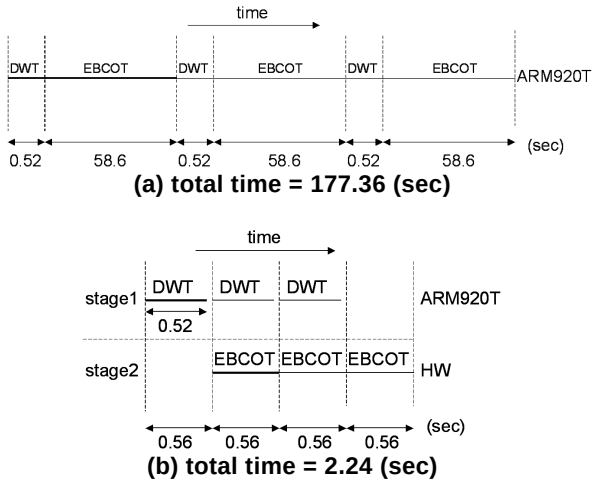


Fig. 7. (a) before, and (b) after partitioning and pipelined scheduling by our synthesizer

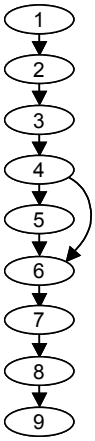


Fig. 8. CFG of MP3 decoding system

Table 1. Initial HW/SW partitioning of MP3 decoding system

Index	Function name of node	Type of component
1	Read_head( )	SW
2	III_get_side_info( )	SW
3	III_get_scale_factors( )	SW
4	InvQ( )	SW
5	Reorder( )	SW
6	III_antialias( )	SW
7	III_hybrid( )	HW
8	III_modify_SubBandSynthesis( )	HW
9	Mp3_end_process( )	SW