行政院國家科學委員會補助專題研究計畫成果報告

※※※※※※※※※※※※※※※※※※※※※※※※※
※　　　　　　　　　　　　　　　　　　　　　　　※
※　　　　# ARM 嵌入式系統低耗電匯流排設計　　　　※
※　　　　　　　　　　　　　　　　　　　　　　　※
※※※※※※※※※※※※※※※※※※※※※※※※※

計畫類別：[✓ 個別型計畫]　　□整合型計畫
計畫編號：NSC 90－2213－E－009－135
執行期間：90 年 8 月 1 日至 91 年 7 月 31 日

計畫主持人：單智君 博士

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

執行單位：國立交通大學資訊工程學系

中　華　民　國　91　年　10　月　20　日

行政院國家科學委員會專題研究計畫成果報告

ARM 嵌入式系統低耗電匯流排設計

Design of Low Power Buses in ARM Embedded Systems

計畫編號：NSC 90－2213－E－009－135
執行期限：90 年 8 月 1 日至 91 年 7 月 31 日
主持人：單智君 博士
計畫參與人員：林光彬、林宗蔚、楊敦傑、蔡佳洲

一、中文摘要

近年來，對於減少處理器耗能的需求越來越被重視。研究發現，系統在程式執行中所耗能量，有 50% ~ 80% 是消耗在處理器與外部記憶體的傳輸上，即匯流排上。故想要設計出更省電的系統，必須先從匯流排著手。匯流排上之耗能是與在其上傳送的資料位元變化量成正比。本計畫的研究重點，主要針對指令匯流排耗能的減少，亦即減少處理器要求記憶體傳送的指令間之位元變化。

在這個計畫裡，我們提出了使用基本區塊 (basic-block) 指令字典的方法來減少指令匯流排的耗能。基本區塊是程式在執行時的特性之一；一旦處理器開始執行基本區塊內的指令，除非遇到中斷，否則區塊內所有指令都會被執行。我們的方法是把常用的基本區塊放入指令字典中。與記憶體相比，這個字典有較小的耗能且更接近處理器。而在原始程式中這些被放置到字典內的基本區塊則會被字典索引編碼取代。在執行到這些基本區塊時，只有字典索引編碼會在指令匯流排上傳遞。藉著將原本有多道指令在指令匯流排上傳送變成只有一個編碼被傳送，我們可以減少許多的耗能。此外，當遇到需要重複傳送相同的索引編碼時，我們也藉著應用編碼暫存機制進一步減少更多的耗能。

實驗結果顯示，跟原始系統比較，此法平均可以減少 91.2%的耗能，且抓取指令的時間也會減少大約 83.6%。

關鍵詞：ARM 處理器，嵌入式系統，低耗電，匯流排，位元變化

Abstract

The requirement in reducing the power of a processor has grown dramatically over the past few years. Researches have shown that 50%~80% power are dissipated on the transmission between CPU and Memory. Power consumed on buses is decided by numbers of bit toggles on the buses. In this project, we focus on instructions transmitted on buses to reduce the numbers of bit toggles at instruction buses.

In this project, we propose a basic-block instruction dictionary to reduce power consumption on instruction bus. Basic-block is one of the characteristics of program execution behavior that once a processor step into a basic-block, the whole block will be executed except that interrupt occurs. We place the most frequently used basic-blocks into the instruction dictionary, which has smaller power consumption and is closer to CPU. Then, these basic-blocks at original programs are replaced by codewords indexing to the corresponding basic-blocks in the dictionary. Thus, we can reduce the power consumption when repeatedly executing on these basic-blocks because only the codewords are transferred in the instruction bus. A codeword lookaside buffer is also applied to provide more power reduction when accessing codeword repeatedly.

Experimental results show that on average 91.2% power reduction can be achieved with comparing to the base system. By accessing instructions in the internal dictionary, we can also reduce the execution time by 83.6%.

Keywords：ARM processor, embedded system, low power, bus, bit toggle

二、緣由與目的

For embedded systems, the major power dissipations of embedded systems come from the global communication on external buses. Many embedded systems are designed for multimedia and signal processing applications such as PDA (Personal Digital Assistants) and cellular phones. In such systems that involve multidimensional streams of signals such as images, video or voice sequences, it has been shown that the majority of the area and power cost is not due to the datapath or the controllers, but due to the global communication and memory interaction.

In fact, 50% to 80% of the power cost in application-specific integrated circuits (ASIC) for real-time signal processing is due to memory traffic caused by the ASIC and the off-chip memories. It is because that the off-chip capacitances are three orders of magnitude larger than the typical on-chip capacitances. This implies that design techniques leading to decrease the power dissipation in this part will make a significant impact on the overall power dissipation of the application.

In this project, we design some power reduction techniques to reduce the runtime power dissipated on the system buses. We focus on the instruction bus and the related address bus to exploit the repetitions of instructions for reducing power dissipations on the buses. To prevent the repeated instructions from transmitting on the buses, a pre-selected basic-block-based dictionary is applied. This basic-block dictionary is working as internal memory nearby the processor core. It includes codeword decoder and basic-block storages which works somewhat like an instruction cache but is demanded for lower power.

三、結果與討論

3.1 Architecture Model

Our baseline architecture model is shown in Figure 1. In this baseline system, processor sends address request and receives instructions from main memory directly. We find that repeatedly executed instructions will continuously drive the same bus transactions and consume power.

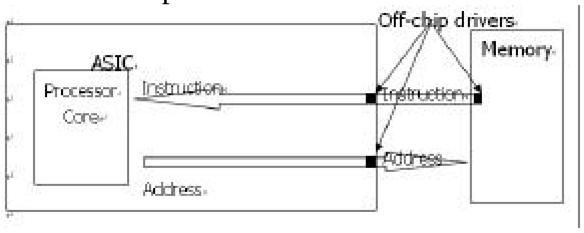

Figure 1. Architecture model of baseline system

Figure 2 is our design model. In this model, we add a hardware mechanism, called basic-block dictionary, between the processor and the memory. Our design is focused on the instruction bus and the related address bus to exploit the repetitions of instructions for reducing power dissipations on buses. Transactions on these buses are influenced by program execution behaviors. Instructions are transmitted on buses repeatedly and are grouped by basic blocks. To prevent repeatedly instructions and address requests from transmitting on buses, we design a basic block dictionary. The basic block dictionary works as an intermediate between processor and main memory. It receives address requests from processor and returns instructions from main memory or its own instruction table.

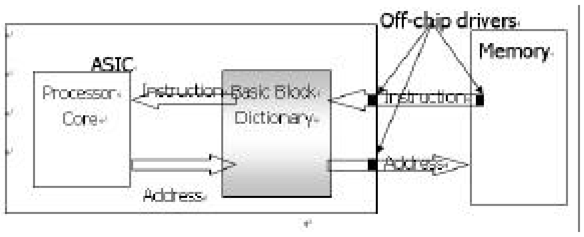

Figure 2. Architecture model of our design

3.2 Basic Block Dictionary

In our design, the basic-block dictionary is used to store the most frequently used basic-blocks and works like a read-only memory (ROM). The procedure for this power reduction method is divided into two phases: ***basic-block dictionary building phase*** and ***dictionary accessing phase***. The phase of basic-block dictionary building is responsible for choosing the frequently used basic-blocks into dictionary to lower the numbers of bit toggles on buses. This phase is processed at software offline. The dictionary accessing phase includes decoding control logic to control the operations of fetching the codewords of dictionary entry and convert codewords into original instructions.

3.2.1 Basic-block Dictionary Building Algorithm

Our building algorithm is applied to analyze program-execution behavior by calculating the number of bit toggles and execution count of each basic block. The key idea of the algorithm is to select the most frequently executed basic blocks to be added into the dictionary. The algorithm is divided into three parts as follows:

1. Choosing basic-blocks for the dictionary,
2. Replacing basic blocks with codewords for a program, and
3. Modifying branch targets in the coded program.

After all dictionary entries are chosen, we need to encode the original program by replacing each basic block that already in dictionary with its mapping index, called codeword.

One obvious side effect of the basic-block dictionary scheme is that it alters the locations of instructions in the program. This presents a special problem for branch instructions since branch targets are changed as a result of program compression. To avoid this problem, the targets of branch instructions are patched to the new locations in the coded program.

3.2.2 Dictionary accessing Phase

The hardware mechanism of dictionary accessing phase consists of two main modules: basic-block dictionary and decoding hardware. An additional mask can be place at the control circuits in memory side to gain more benefit in transmitting codewords.

The block diagram of the proposed, basic-block dictionary is shown in Figure 3. The blocks inside the dotted line is our designed circuits, the decoding-control logic, that contain the basic-block dictionary and the decoding-control logic. The decoding-control logic is further divided into instruction fetcher, codeword detector, boundary detector and address synchronized unit. This hardware mechanism may be combined with the processor core into a single chip.

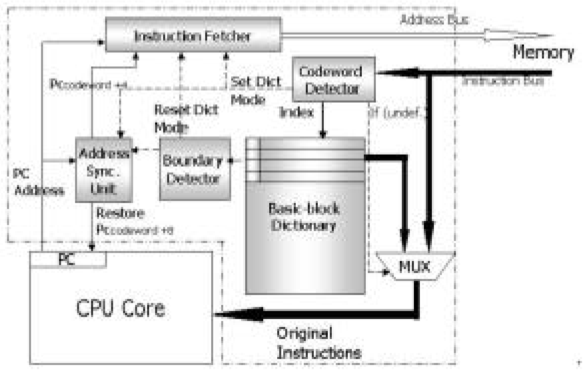

Figure 3. System architecture with basic-block dictionary

The basic-block dictionary stores the basic blocks selected by the basic-block dictionary building algorithm. It is like a read-only memory and each elemental line has 32-bit instruction and one boundary bit.

The decoding-control logic is responsible for sending instruction to processor from memory or dictionary. It first fetches instructions from memory and then determines if the fetched instruction is a codeword. If the fetched instruction is a codeword, the original instructions will be gathered from dictionary. It is also designed to check if the dictionary entry is finished and to synchronize the address between processor and memory.

Based on the design of the basic-block dictionary, the power consumption can be further reduced by attaching a *Codeword Lookaside Buffer* (CLB). When a program is executed, the processor may often execute some sequences of instructions repeatedly. These sequences of instructions are known as loops. A loop contains either one or more basic blocks. The mapping relation is stored when a codeword is fetched and decoded with the CLB, we may eliminate some bus transaction for the codewords whose addresses have already been recorded in the buffer.

The CLB is not available when the processor is in dictionary mode. It becomes available when the processor is out from the dictionary mode and the next program-counter address is sent to the decoding-control logic. The decoding-control logic will query this buffer to find if this address is already in CLB. If yes, the PC address points to a codeword in the external memory and we may use *Dictionary Index* field in buffer to index the dictionary. Otherwise, we still need to fetch this instruction from memory.

3.3 Simulation Results and Analysis

We adapt the MediaBench consisting of well-known multimedia and communication applications for our benchmark. The input data we used for this benchmark suite are the recommended input data included in the MediaBench.

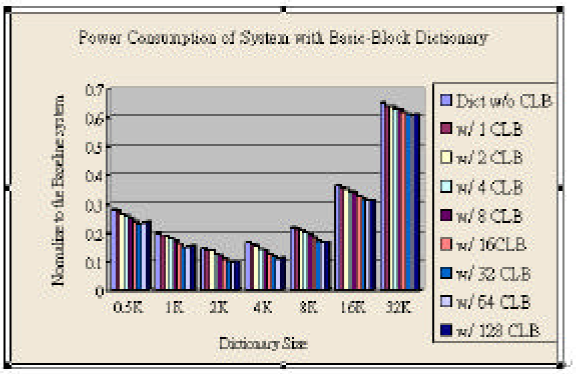

Figure 4. Power concumption

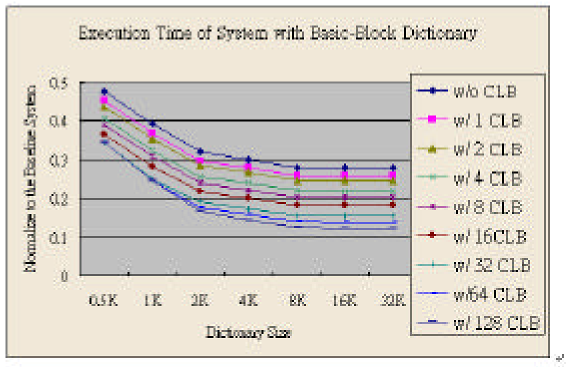

Figure 5. Execution time

The power consumption and execution time of a system with different size of basic-block dictionary and CLB are shown in Figure 4 and 5, respectively. A 2K byte dictionary with 64-entry codeword lookaside buffer achieves best power reduction as much as 91.2%. Moreover, it may achieve 83.6% of execution time saving for the 2-Kbyte dictionary with 64-entry CLB.

四、計畫成果自評

In this project, we have examined a basic-block dictionary to reduce power consumption on instruction bus and related address bus. A codeword lookaside buffer is also proposed to work with the dictionary to reduce more power consumption. The key idea of our method is to apply a dictionary which stores frequently executed basic blocks to make use of the repetitions of basic blocks at program execution time for reducing bit toggles on instruction bus.

五、參考文獻

[1] *Wen-Tsong Shiue and Chaitali Chakrabarti*, "**Memory design and exploration for low power embedded systems**," Signal Processing Systems, 1999. SiPS 99. 1999 IEEE Workshop on, Page(s): 281 –290, 1999

[2] *Benini, L.; Macii, A.; Macii, E.; Poncino, M.*, "**Selective instruction compression for memory energy reduction in embedded systems",** Low Power Electronics and Design, 1999. Proceedings, Page(s): 206 -211. 1999 International Symposium on , 1999.

[3] *Lea Hwang Lee; Moyer, B.; Arends, J.*, "**Instruction fetch energy reduction using loop caches for embedded applications with small tight loops,"** Low Power Electronics and Design, 1999. Proceedings, Page(s): 267 –269. 1999 International Symposium on , 1999

[4] *C. Lee, M. Potkonjak, and W. H. M.-Smith*, "**MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems**", 30th Annual ACM/IEEE International Symposium on Microarchitecture, 1997.

[5] *Wilton, S.J.E.; Jouppi, N.P.* "**CACTI: an enhanced cache access and cycle time model,"** Solid-State Circuits, IEEE Journal of , Volume: 31 Issue: 5 , Page(s): 677 –688, May 1996