CASCADE – CONFIGURABLE AND SCALABLE DSP ENVIRONMENT

Tay-Jyi Lin and *Chein-Wei Jen*

Department of Electronics Engineering,
National Chiao Tung University, Taiwan

ABSTRACT

As the complexity of embedded systems grows rapidly, it is common to accelerate critical tasks with hardware. Designers usually use off-the-shelf components or licensed IP cores to shorten the time to market, but the hardware/software interfacing is tedious, error-prone and usually not portable. Besides, the existing hardware seldom matches the requirements perfectly. CASCADE, the proposed design environment as an alternative, generates coprocessing datapaths from the executing algorithms specified in C/C++ and attaches these datapaths to the embedded processor with an auto-generated software driver. The number of datapaths and their internal parallel functional units are scaled to fit the application. It seamlessly integrates the design tools of the embedded processor to reduce the re-training/design efforts and maintains short product development time as the pure software approaches. A JPEG encoder is built in CASCADE successfully with an auto-generated four-MAC accelerator to achieve 623% performance boost for our video application.

1. INTRODUCTION

Technology improvement and personal fashion statements represented by electronic products significantly shorten the product lifecycle. Thus, time to market has disproportionate impact upon the profits of a product and even determines its success or failure. By the way, the drift from analog to digital signal processing enables the integration of multiple complex functions on a single chip. Platform-based design methodology effectively handles this design complexity to reduce the overall time to market, which enables the designers to spin different products quickly [1]. Fig 1 shows the computing kernel of a heterogeneous platform for most data-intensive applications. It consists of two subsystems with distinct computing paradigms – ***control-flow*** and ***data-flow***. The former controls and coordinates the system tasks and performs some ***reactive tasks*** such as the user interface. It is usually composed of a controller licensed from third-party IP vendors such as ARM, MIPS or low-cost 8051, 6502. Controller-related coprocessors, such as floating-point accelerators or memory management units (MMU), may be plugged in if required. The latter efficiently handles the ***transformational tasks*** with more regular and predictable behaviors, such as the small and well-defined workloads in DSP applications. Small loop-nests with high parallelism usually dominate the execution time in audio, image or video processing, but they are much more deterministic. Data-flow subsystems range from programmable DSP processors with the most flexibility, customized DSP datapaths that can be configured for a specific domain, or some fixed-function ASIC to achieve the maximum performance. An optimal embedded system allocates tasks on a subsystem depending on the characteristics to achieve higher speed, lower power consumption and, most important, the minimum system cost.

Baseband processors in cell-phones are composed of two programmable processors (i.e. RISC & DSP) [2][3], which comply with the aforementioned computing model. The main drawback is their complicated programming model, which supports DSP applications only through limited C libraries. Designers always need to optimize their applications with hand-coded assembly. Because the DSP processor is not customized for this dual-processor configuration, repeated and redundant functions exist in both of the two subsystems. In this paper, we propose an alternative that attaches an existing controller with customized DSP datapaths, which are data-driven and can be easily configured for a wide range of DSP applications through our automatic generator. These coprocessing datapaths are synchronized by simple host instructions. Their performance is scalable as the internal SIMD-like parallel functional units are chosen to fit distinct requirements of various DSP applications. To be brief, based on a controller, our proposed CASCADE design environment synthesizes a tailored computing platform automatically for an application specified in C/C++, while still meeting the short time to market constraints.

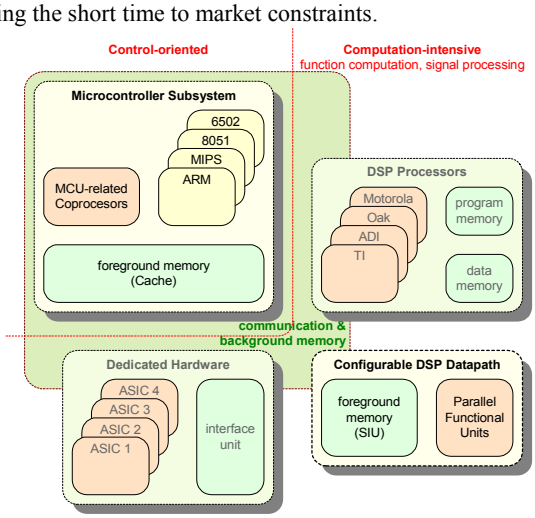

Figure 1. The heterogeneous computing platform

The rest of this paper is organized as follows. Section 2 summarizes some related works and illustrates the computing model of the configurable kernel in our proposed heterogeneous DSP platform. The synchronization mechanism between the two subsystems is also detailed. Then, we introduce our design environment – CASCADE in Section 3, which maps data-intensive applications onto our heterogeneous computing platform. Section 4 describes the configuration of scalable coprocessing

This work was supported by the National Science Council, Taiwan under Grant NSC89-2218-E009-078.

datapaths. Section 5 shows the effectiveness of CASCADE with a standard JPEG encoder. Another modified Motion JPEG implementation is also given to demonstrate our heterogeneous approach features improved silicon area and power consumption than pure software approaches with high-end processors. Finally, Section 6 concludes our work.

2. RELATED WORKS & OUR PROPOSED MODEL

Hardware accelerators with handcrafted interfaces, which have traditionally been used to boost system performance, to reduce power consumption, or to cut manufacturing cost, are error-prone and seldom reusable. New methodologies are emerging from the CAD domain in the field of hardware/software (HW/SW) co-design. Cosyma [4] uses extended C with concurrency and timing constraints for system specification. HW and SW are partitioned automatically based on simulation annealing with estimated schedule times to maximize speedup. The HW/SW interfacing is done manually. Vulcan [5] takes HardwareC with timing and resource constraints as its input. In contrast to Cosyma, it minimizes hardware cost by moving operations to software from an initial pure-hardware implementation while satisfying the imposed timing constraints. Recent researches, mainly in the embedded processor community, propose configurable and/or extensible architectures that easily adapt for specific application requirements. Tensilica Xtensa [6] allows the designers to manually define specific instructions with its TIE language and generates the synthesizable processor core with a rich set of customized software tools. HP PICO [7] partitions an application written in C between custom non-programmable hardware and the compiled software code executing on an application-specific VLIW processor. The hardware interfaces to the global memory with a specific controller.

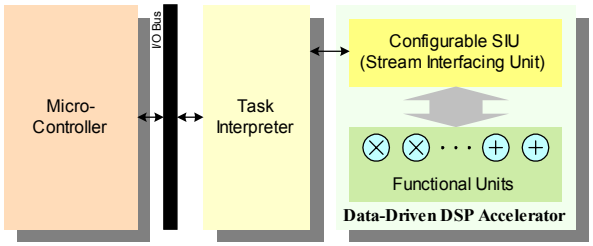

Figure 2. The proposed DSP computing platform

Fig. 2 shows our proposed computing platform for data-intensive applications. The controller is responsible for system coordination and all interfaces. The data-driven datapaths serve as slave coprocessing accelerators attached to the controller, which are much more efficient for coarse-grain tasks with regular, predictable behaviors and high parallelism. The task interpreter translates the original control-flow semantics of the dispatched C/C++ task into dataflow mechanism and drives the attached coprocessing datapaths. The stream interface unit (SIU) [8] is an application-specific foreground memory with configurable routing that interacts with the task interpreter. Fig. 3 shows the baseline SIU model, which consists chiefly of multiple register queues for temporary storage and MUX-based interconnection networks. Various extension models exist in the literature, such as [9][10]. The SIU handles data format conversion if required and supports extremely high data bandwidth to the parallel functional units by massive data reuse explored in the executing DSP algorithms. The coprocessing datapaths boost the performance through (i) highly parallel computation with SIMD-like functional units, (ii) elimination of control overheads with configurable data routing, and (iii) reduction of loads and stores with foreground SIU buffering and embedded address generation.

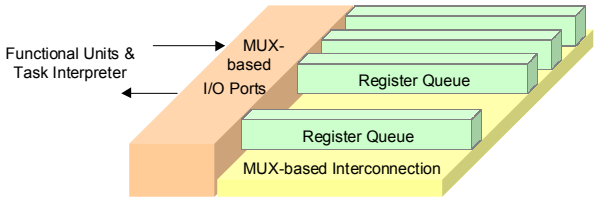

Figure 3. Multi-queue Stream Interface Unit (SIU)

The accelerators are I/O-mapped to the address space of the host controller with 4 memory-mapped registers in the task interpreter, which is coordinated by software. Software codes for the dispatched task are replaced with an auto-generated driver, which has an identical interface to remnant software modules (i.e. the software driver together with the data-driven accelerators maintains the same semantics as the original dispatched task). Fig 4 depicts the template of the simple software driver, where the dataflow constructs guarantee correct execution under the optimization by the compiler, the assembler statically or even dynamically with the RTOS. The driver prepares and feeds data, while handshaking with the task interpreter through the 4 memory-mapped registers. Because data movement is completely under the host control, no explicit data coherence mechanism is required. By the way, additional FIFO queues are allocated in the task interpreter to regulate the execution of the data-driven accelerators for heavy-loaded I/O buses or distinct operating frequencies between host and accelerators. The type or the number of parallel functional units is chosen to match the specification.

```
void dispatch_fun(fix *Din, int size, fix *Dout)
{ int   index1=0, index2=0;
  fix   din, dout;
  bool  valid_in, valid_out;

  Codes for Exception Handling

  while(index2 <= size)
  { valid_in = (index1 < size);
    din = Din[index1++];
        IO_Operation (din, valid_in, &dout, &valid_out);
    Dout[index2] = dout;
    index2 = index2 + valid_out;
} }
```

Figure 4. The software driver template

Optimal workload distribution at task level between the two subsystems with appropriate computing paradigm significantly reduces the synchronization overheads. The proposed DSP computing platform provides a more optimal embedded system solution with higher speed, and/or lower power consumption, and the most important, reduced total cost.

3. CASCADE DESIGN ENVIRONMENT

CASCADE is the design environment that targets data-intensive embedded systems on our DSP computing platform. It seamlessly integrates the original design environment of any available micro-controller and maps various DSP algorithms onto customized coprocessing datapaths. Designers that are already familiar with the micro-controller design flow do not need much retraining effort. CASCADE provides design validation of the auto-generated DSP datapaths with the software driver by HW/SW co-simulation and a preliminary formal verifier.

3.1. Design Flow

The CASCADE design flow is shown in Fig. 5. The software design flow is tightly coupled with the micro-controller design environment and shown on the left-hand side. The algorithm development/simulation and code debugging of a new product are completed in software. Designers first compile and assemble the source codes following the original micro-controller design flow. The performance estimation in the trial compilation is forwarded to CASCADE. Designers that are indeed experts of the target application domain supervise the task dispatch with special compiler directives. Code segments with high parallelism are selected as candidates to be accelerated on the coprocessing datapaths at function/task level. With the assisted profiling information, inexperienced designers can also explore an optimal architecture with minimal iterations.

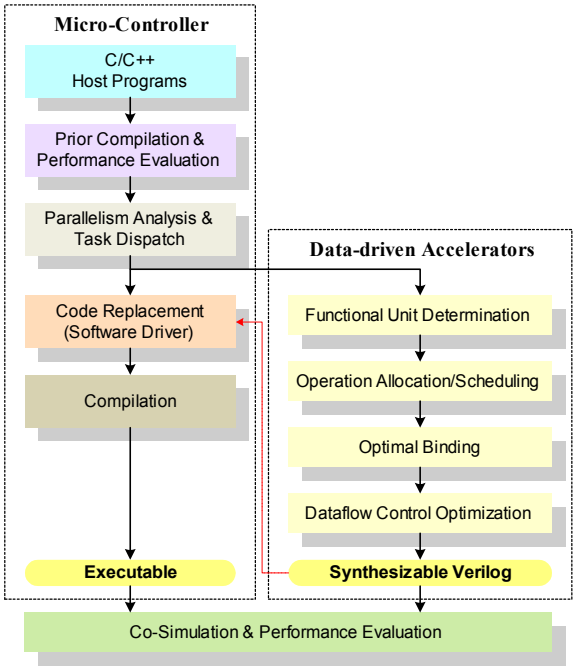

Figure 5. The CASCADE design flow

The dispatched candidates usually contain code segments for rarely occurring exceptional cases, which effectively improve the software robustness. Predication techniques in VLIW compilers are used here to extract a hyperblock [11] by joining basic blocks along frequently executed control paths with highly parallelism. A synchronous dataflow graph (SDFG) is derived from the hyperblock for the hardware flow on the right-hand side, which generates optimal coprocessing datapaths in synthesizable RTL Verilog. The auto-generated software driver prepares and generates proper I/O sequence to the coprocessing datapaths with the same procedural interface as the original task. The driver also takes care of the remnant codes for exceptions and those non-supported operations. For synchronous interfacing, CASCADE collects deterministic performance parameters from the HW synthesis result, such as the computation time and latency. The software driver is ANSI-C compatible and thus the modified codes (i.e. the dispatch task replaced by the driver) can be still compiled and simulated again easily in the original micro-controller environment.

3.2. Design Validation

Assuming code debugging is finished in the micro-controller design environment, CASCADE only needs to guarantee no error is introduced in our proposed flow (i.e. the target system is functionally equivalent to the all-software specification). Direct FPGA emulation with the host controller is used here, because instruction-set simulators from most vendors provide no or very poor interface to programming languages. CASCADE then performs equivalence checking (EC) on the auto-generated datapaths to verify their functional equivalence to the original C/C++ sources. We have also constructed a preliminary formal equivalence checker [12]. It provides complete EC as opposed to simulation or emulation, which checks the equivalence only to some extent that the test suite exercises the design.

4. GENERATION OF COPROCESSING DATAPATH

A tool set has been constructed to generate the synthesizable coprocessing datapaths that have proper computing power to meet the application requirements. These tools communicate in plain-text files and have been integrated with user-friendly GUI.

4.1. Functional Unit Determination

Designers choose the types of functional units in our interactive tool to construct the coprocessing datapaths, such as multipliers with adders, adders with shifters or some other combinations. CASCADE performs required transformations on the SDFG derived from the hyperblock (e.g. shift-add decompositions [13] for multiplication operations). Bit-width analysis [14] can be used here to reduce the wordlength of the synthesized datapaths. This first CASCADE prototype supports linear operations only and uses identical wordlength to the host for simplicity.

4.2. Operation Scheduling and Allocation

Depending on the profiling information and the specification from the trail compilation, CASCADE estimates the required speedup factor for the dispatched tasks with the Amdahl's law [15]. It computes the maximum allowable computation cycles with an acceptable latency depending on this factor. Then, it performs time-constrained scheduling (force-directed scheduling [16]) and allocation based on these parameters while trying to minimize the number of functional units in the coprocessing datapaths.

4.3. Optimal Binding

CASCADE first calculates the number of required queuing cycles for each variable, including new arriving input samples or computed data items from the parallel functional units. For each edge in the scheduled SDFG, this number equals to

$$D_F\left(U \xrightarrow{e} V\right) = N \cdot w(e) - P_U + v - u$$

i.e. the number of delay elements on the edge, $w(e)$, multiplied by the operation period, N, for one iteration, minus the number of internal delays (models pipelining) of the computation unit, P_U, and then adjusted with the scheduled indices v and u within the N cycles [17]. The default value of P_U is 2, which represents the I/O registers of the functional units to allow a full clock-cycle delay in SIU routing. Optional retiming [18] for minimal buffering improves the binding, which is formulated as an ILP problem:

Minimize

$$\sum q(U)$$

Subject to

(a) Feasibility constraints

$$r(U) - r(V) \le \left\lfloor \frac{D_F\left(U \xrightarrow{e} V\right)}{N} \right\rfloor$$

(b) Maximum queue for each variable

$$D_F\left(U \xrightarrow{e} V\right) \le q(U)$$

, where

$$\begin{aligned} & D_F\left(U \xrightarrow{e} V\right) \\ &= N \cdot w_r(e) - P_U + v - u \\ &= N \cdot [w(e) + r(V) - r(U)] - P_U + v - u. \end{aligned}$$

Here, $q(U)$ stands for the number of maximum queuing cycles for the variable U by constraints (b) among the multiple fanouts from U. Feasibility constraints (a) force each D_F to be non-negative after retiming to guarantee system causality. This ILP problem is solved using Lindo package [19] in CASCADE. It saves 25% to 35% SIU registers in average in our experiments.

4.4. Dataflow Control Optimization (DCO)

SIU controls and buffers the dataflow among the functional units and I/O ports to task interpreter. Various SIU architectures exist and CASCADE targets and optimizes the SIU in this stage. For example, lifetime analysis is first performed to determine the number of required registers in each queue of the baseline SIU in Fig. 3. The variables are then allocated to the register queues while minimizing the routing complexity [8]. Finally, the synthesizable Verilog description is generated.

5. EXAMPLE

We have ported a standard JPEG [20] encoder with CASCADE on an ARM7TDMI-hosted DSP computing platform for video-rate (thirty 320×240 frames per second). The host runs at 50MHz with DCT dispatched to a 4-MAC coprocessing datapath. Table 1 summarizes the performance improvement by the auto-generated DCT accelerator. The required computation of the run-length calculation and modified Huffman coding is highly data dependent, so the average cycle count is used.

Table 1 Performance comparison*

	ARM alone	+Accelerator
8×8 DCT	5,595 cycles	246 cycles
	111.90 µs	4.92 µs
320×240 Frame	152.928 ms	24.552 ms

* An ideal memory subsystem is assumed for simplicity (i.e. the system does not have memory stalls), and performance improvement is pessimistically estimated.

The standard JPEG encoder was modified for a surveillance system that has a still background most of time. It completely skips an 8×8 block coding if the block is similar to that of the received frame in the same position. The criterion for similarity is the DC difference (thus DCT is not required for similarity testing) with an adjustable threshold. To reduce power consumption, the synthesized DCT accelerator is shut down and stays powered off with simple circuitry, while the host continuously performs the similarity testing and skips blocks. To meet real-time constraints, pure software implementation requires a high-performance micro-controller at a high price, but infrequently achieves the peak performance. Power management is therefore crucial but very complicated through the "SLEEP" mode switching available in most modern processors. This is because the similarity testing is still needed for each incoming block (36,000 blocks per second isochronously in the modified JPEG encoder). The proposed heterogeneous DSP computing platform with attached datapaths provides a much more cost- and energy-effective solution.

6. CONCLUSION

We have presented CASCADE design environment in this paper. It can easily configure the proposed DSP computing platform, which attaches the auto-generated accelerators to an existing micro-controller, and scale its performance. The coprocessing datapaths are driven by host instructions in the software interface, which is also auto-generated with a memory map table, to simplify the synchronization problem. CASCADE seamlessly integrates the original micro-controller design environment to reduce the time-to-market as short as pure software approaches. It can effectively lengthen the life span of an existing controller IP licensed from the third-party (i.e. maximize the usage) for more complex applications.

7. REFERENCES

[1] H. Chang, et al, *Surviving the SOC Revolution – a Guide to Platform-Based Design*, Kluwer Academic Publishers, 1999

[2] A. Gatherer, et al, "DSP-based Architectures for Mobile Communications: Past, Present and Future," *IEEE Communications*, Jan 2000

[3] *AD6522 Digital Baseband Processor Design Specification*, Analog Devices/TTP Comm., May 2000

[4] T. Benner, et al, "Scalable Performance Scheduling for Hardware-Software Co-synthesis," *European Design Automation Conference (EDAC)*, 1995

[5] R. K. Gupta and G. D. Micheli, "Hardware-Software Co-synthesis for Digital Systems," *IEEE Design & Test of Computers*, Sep 1993

[6] R. E. Gonzalez, "Xtensa: A Configurable and Extensible Processor," *IEEE Micro*, Mar-Apr 2000

[7] B. R. Rau and M. S. Schlansker, "Embedded Computer Architecture and Automation," *IEEE Computers*, Apr 2001

[8] T. J. Lin and C. W. Jen, "Data Stream Generation for Concurrent Computation in VLSI Signal Processors," *International Conference on Signal Processing (ICSP)*, 2000

[9] K. Srivatsan, C. Chakrabarti and L. Lucke, "Low Power Data Format Converter Design Using Semi-Static Register Allocation," *International Conference on Computer Design*, 1995

[10] M. Majumdar and K. K Parhi, "Design of Data Format Converters Using Two-Dimensional Register Allocation," *IEEE Transactions on Circuits and Systems II*, April 1998

[11] S. A. Mahlke, et al, "Effective Compiler Support for Predicated Execution Using the Hyperblock," *International Symposium on Microarchitecture (MICRO 25)*, 1992

[12] T. J. Lin and C. W. Jen, "Formal Equivalence Checking of Folded Architectures," *WSES/IEEE World Multiconference on Circuits, Systems, Communications & Computers (CSCC)*, July 2001

[13] H. T. Nguyen and A. Chatterjee, "Number-Splitting with Shift-and-Add Decomposition for Power and Hardware Optimization in Linear DSP Synthesis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, August 2000

[14] M. Stephenson, J. Babb and S. Amarasinghe, "Bitwidth Analysis with Application to Silicon Compilation," *International Conference on Programming Language Design and Implementation (PLDI)*, 2000

[15] J. L. Hennessy and D. A. Patterson, *Computer Architecture – A Quantitative Approach*, 2nd Edition, 1996

[16] D. D. Gajski, et al, *High Level Synthesis – Introduction to Chip and System Design*, Kluwer Academic Publisher, 1992

[17] K. K. Parhi, *VLSI Digital Signal Processing Systems – Design and Implementation*, John Wiley & Sons, 1999

[18] C. E. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry," *Algorithmica*, June 1991

[19] Lindo Package, available at http://www.lindo.com

[20] W. B. Pennebaker and J. L. Mitchell, *JPEG – Still Image Data Compression standard*, Van Nostrand Reinhold, 1993