Architecture 3

MCU, ASIP, TTA

Contents

Chapter 1

MCU

MCU may refer to:

1.1 Arts & entertainment

- MCU, Japanese hip-hop/rap artist, formerly of Kick the Can Crew
- Marvel Cinematic Universe, a shared fictional universe of films and TV series developed by Marvel Studios
- Medium close-up, camera direction used in British television scripts

1.2 Organizations

- Movement of Unitarian Communists (*Movimento dei Comunisti Unitari*), an Italian communist party
- Modern Churchpeople's Union, an Anglican liberal theological organisation
- Monte Carlo Universal, a computer software project to simulate particle transport using the Monte Carlo method
- Municipal Credit Union, a credit union in New York City

1.3 Science and technology

- Media Control Unit, touchscreen interface on Tesla Model S
- Microcontroller unit, a single computer chip designed for embedded applications
- Memory controller unit, the part of a microprocessor responsible for interfacing it with main memory
- Milk clotting units, a measure of enzyme activity
- Minimum coded unit, the pixel block size of a JPEG computer image
- Mitochondrial Calcium Uniporter, a calcium channel in a human cell's mitochondria
- Modular Concept Unit, the basic avionics packaging compliant with ARINC Specification 600
- Moisture cure polyurethane coatings, corrosion-resistant marine and protective coatings
- Multi-chip unit, a system that contains the processing units of the VAX 9000 minicomputer
- Multipoint control unit, a device used to bridge videoconferencing connections

1.4 Universities

- Marine Corps University, U.S. Marine Corps military graduate school
- Ming Chuan University, Taipei, Taiwan
- Manila Central University, Manila, Philippines
- Marymount California University, Palos Verdes, CA, USA

1.5 Other

- *Major Crimes Unit* in various works of fiction (part of the Gotham City Police Department in the *Batman* comics; agency of the Chicago Police Department in the TV series *Crime Story*; division of the L.A.P.D in the movie Heat)
- Montluçon - Guéret Airport, France, IATA airport code

Chapter 2

Application-specific instruction set processor

An **application-specific instruction set processor** (**ASIP**) is a component used in system-on-a-chip design. The instruction set of an ASIP is tailored to benefit a specific application. This specialization of the core provides a tradeoff between the flexibility of a general purpose CPU and the performance of an ASIC.

Some ASIPs have a configurable instruction set. Usually, these cores are divided into two parts: *static* logic which defines a minimum ISA (instruction-set architecture) and *configurable* logic which can be used to design new instructions. The configurable logic can be programmed either in the field in a similar fashion to an FPGA or during the chip synthesis.

ASIPs can be used as an alternative of hardware accelerators for baseband signal processing[1] or video coding.[2] The traditional hardware accelerators for the baseband or multimedia suffer from inflexibility. It is very difficult to reuse the hardware datapath with handwritten finite-state machines (FSM). The retargetable compilers of ASIPs help the designer to update the program and reuse the datapath. Typically, the ASIP design is more or less dependent on the tool flow because designing a processor from the scratch can be very complicated. There are some commercial tools to design ASIPs, for example, Processor Designer from Synopsys. There is an open source tool as well, TTA-based codesign environment (TCE).

2.1 References

[1] Shahabuddin, Shahriar et al., "Design of a transport triggered vector processor for turbo Decoding", in Springer Journal of Analog Integrated Circuits and Signal Processing, March 2014.

[2] Hautala, Ilkka, et al. "Programmable Low-Power Multicore Coprocessor Architecture for HEVC/H.265 In-Loop Filtering" in IEEE Transactions on Circuits and Systems for Video Technology, November 2014

2.2 Literature

- Dake Liu (2008). *Embedded DSP Processor Design Application Specific Instruction-set Processors*. MA: Elsevier Mogan Kaufmann. ISBN 978-0-12-374123-3.
- Oliver Schliebusch, Heinrich Meyr, Rainer Leupers (2007). *Optimized ASIP Synthesis from Architecture Description Language Models*. Dordrecht: Springer. ISBN 978-1-4020-5685-7.
- Paolo Ienne, Rainer Leupers (eds.) (2006). *Customizable Embedded Processors*. San Mateo, CA: Morgan Kaufmann. ISBN 978-0-12-369526-0.
- Matthias Gries, Kurt Keutzer (eds.) (2005). *Building ASIPs: The Mescal Methodology*. New York: Springer. ISBN 978-0-387-26057-0.

2.3 External links

- TTA-Based Codesign Environment (TCE), an open source (MIT licensed) toolset for design of application specific TTA processors.

Chapter 3

Transport triggered architecture

In computer architecture, a **transport triggered architecture** (**TTA**) is a kind of CPU design in which programs directly control the internal transport buses of a processor. Computation happens as a side effect of data transports: writing data into a *triggering port* of a functional unit triggers the functional unit to start a computation. This is similar to what happens in a systolic array. Due to its modular structure, TTA is an ideal processor template for application-specific instruction-set processors (*ASIP*) with customized datapath but without the inflexibility and design cost of fixed function hardware accelerators.

Typically a transport triggered processor has multiple transport buses and multiple functional units connected to the buses, which provides opportunities for instruction level parallelism. The parallelism is statically defined by the programmer. In this respect (and obviously due to the large instruction word width), the TTA architecture resembles the very long instruction word (VLIW) architecture. A TTA instruction word is composed of multiple slots, one slot per bus, and each slot determines the data transport that takes place on the corresponding bus. The fine-grained control allows some optimizations that are not possible in a conventional processor. For example, software can transfer data directly between functional units without using registers.

Transport triggering exposes some microarchitectural details that are normally hidden from programmers. This greatly simplifies the control logic of a processor, because many decisions normally done at run time are fixed at compile time. However, it also means that a binary compiled for one TTA processor will not run on another one without recompilation if there is even a small difference in the architecture between the two. The binary incompatibility problem, in addition to the complexity of implementing a full context switch, makes TTAs more suitable for embedded systems than for general purpose computing.

Of all the one instruction set computer architectures, the TTA architecture is one of the few that has had CPUs based on it built, and the only one that has CPUs based on it sold commercially.

3.1 Benefits in comparison to VLIW Architectures

TTAs can be seen as "exposed datapath" VLIW architectures. While VLIW is programmed using operations, TTA splits the operation execution to multiple *move* operations. The low level programming model enables several benefits in comparison to the standard VLIW. For example, a TTA architecture can provide more parallelism with simpler register files than with VLIW. As the programmer is in control of the timing of the operand and result data transports, the complexity (the number of input and output ports) of the register file (RF) need not be scaled according to the worst case issue/completion scenario of the multiple parallel instructions.

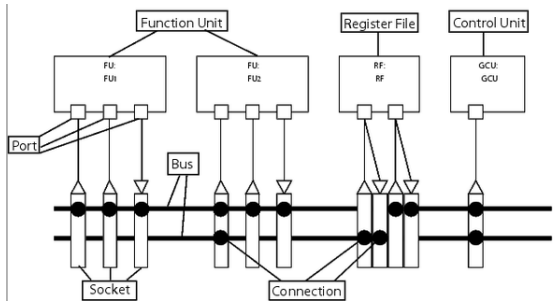
An important unique software optimization enabled by the transport programming is called *software bypassing*. In case of software bypassing, the programmer bypasses the register file write back by moving data directly to the next functional unit's operand ports. When this optimization is applied aggressively, the original move that transports the result to the register file can be eliminated completely, thus reducing both the register file port pressure and freeing a general purpose register for other temporary variables. The reduced RF pressure, in addition simplifying the required complexity of the RF hardware, can lead to significant energy savings, an important benefit especially in mobile embedded systems.[1]

3.2 Structure

TTA processors are built of independent *function units* and register files, which are connected with *transport buses* and *sockets*.

3.2.1 Function unit

Each function unit implements one or more operations, which implement functionality ranging from a simple addition of integers to a complex and arbitrary user-defined application-specific computation. Operands for opera-



Parts of Transport Triggered Architecture

tions are transferred through function unit *ports*.

Each function unit may have an independent pipeline. In case a function unit is fully pipelined, a new operation that takes multiple clock cycles to finish can be started in every clock cycle. On the other hand, a pipeline can be such that it does not always accept new operation start requests while an old one is still executing.

Data memory access and communication to outside of the processor is handled by using special function units. Function units that implement memory accessing operations and connect to a memory module are often called load/store units.

3.2.2 Control unit

Control unit is a special case of function units which controls execution of programs. Control unit has access to the instruction memory in order to fetch the instructions to be executed. In order to allow the executed programs to transfer the execution (jump) to an arbitrary position in the executed program, control unit provides control flow operations. A control unit usually has an instruction pipeline, which consists of stages for fetching, decoding and executing program instructions.

3.2.3 Register files

Register files contain general purpose registers, which are used to store variables in programs. Like function units, also register files have input and output ports. The number of read and write ports, that is, the capability of being able to read and write multiple registers in a same clock cycle, can vary in each register file.

3.2.4 Transport buses and sockets

Interconnect architecture consists of transport buses which are connected to function unit ports by means of *sockets*. Due to expense of connectivity, it is usual to reduce the number of connections between units (function units and register files). A TTA is said to be *fully connected* in case there is a path from each unit output port to every unit's input ports.

Sockets provide means for programming TTA processors by allowing to select which bus-to-port connections of the socket are enabled at any time instant. Thus, data transports taking place in a clock cycle can be programmed by defining the source and destination socket/port connection to be enabled for each bus.

Conditional execution is implemented with the aid of *guards*. Each data transport can be conditionalized by a guard, which is connected to a register (often a 1-bit conditional register) and to a bus. In case the value of the guarded register evaluates to false (zero), the data transport programmed for the bus the guard is connected to is *squashed*, that is, not written to its destination. *Unconditional* data transports are not connected to any guard and are always executed.

3.3 Programming

In more traditional processor architectures, a processor is usually programmed by defining the executed operations and their operands. For example, an addition instruction in a RISC architecture could look like the following.

add r3, r1, r2

This example operation adds the values of general-purpose registers r1 and r2 and stores the result in register r3. Coarsely, the execution of the instruction in the processor probably results in translating the instruction to control signals which control the interconnection network connections and function units. The interconnection network is used to transfer the current values of registers r1 and r2 to the function unit that is capable of executing the add operation, often called ALU as in Arithmetic-Logic Unit. Finally, a control signal selects and triggers the addition operation in ALU, of which result is transferred back to the register r3.

TTA programs do not define the operations, but only the data transports needed to write and read the operand values. Operation itself is triggered by writing data to a *triggering operand* of an operation. Thus, an operation is executed as a side effect of the triggering data transport. Therefore, executing an addition operation in TTA requires three data transport definitions, also called *moves*. A move defines endpoints for a data transport taking place in a transport bus. For instance, a move can state that a data transport from function unit F, port 1, to register file R, register index 2, should take place in bus B1. In case there are multiple buses in the target processor, each bus can be utilized in parallel in the same clock cycle. Thus, it is possible to exploit data transport level parallelism by scheduling several data transports in the same instruction.

An addition operation can be executed in a TTA processor as follows:

r1 -> ALU.operand1 r2 -> ALU.add.trigger ALU.result

-> r3

The second move, a write to the second operand of the function unit called ALU, triggers the addition operation. This makes the result of addition available in the output port 'result' after the execution latency of the 'add'.

The ports associated with the ALU may act as an accumulator, allowing creation of macro instructions that abstract away the underlying TTA:

lda r1 ; “load ALU": move value to ALU operand 1 add r2 ; add: move value to add trigger sta r3 ; “store ALU": move value from ALU result

3.4 Programmer visible operation latency

The leading philosophy of TTAs is to move complexity from hardware to software. Due to this, several additional hazards are introduced to the programmer. One of them is delay slots, the programmer visible operation latency of the function units. Timing is completely a responsibility of programmer. The programmer has to schedule the instructions such that the result is neither read too early nor too late. There is no hardware detection to lock up the processor in case a result is read too early. Consider, for example, an architecture that has an operation *add* with latency of 1, and operation *mul* with latency of 3. When triggering the *add* operation, it is possible to read the result in the next instruction (next clock cycle), but in case of *mul*, one has to wait for two instructions before the result can be read. The result is ready for the 3rd instruction after the triggering instruction.

Reading a result too early results in reading the result of a previously triggered operation, or in case no operation was triggered previously in the function unit, the read value is undefined. On the other hand, result must be read early enough to make sure the next operation result does not overwrite the yet unread result in the output port.

Due to the abundance of programmer-visible processor context which practically includes, in addition to register file contents, also function unit pipeline register contents and/or function unit input and output ports, context saves required for external interrupt support can become complex and expensive to implement in a TTA processor. Therefore, interrupts are usually not supported by TTA processors, but their task is delegated to an external hardware (e.g., an I/O processor) or their need is avoided by using an alternative synchronization/communication mechanism such as polling.

3.5 Implementations

- MAXQ[2][3]

  Currently, the only commercially available microcontroller built upon (though not “featuring”) Transport Triggered Architecture is from Dallas Semiconductor. However, it is an OISC or "one instruction set computer", offering but a **single** though flexible MOVE instruction, which can then function as various virtual instructions by moving values directly to the program counter.

- The “move project” has designed and fabricated several experimental TTA microprocessors.
- The TCE project is a re-implementation of the MOVE tools. The tools are available as open source, and the compiler is built around the LLVM compiler framework.[4][5]
- The architecture of the Amiga Copper has all the basic features of a transport triggered architecture.
- The Able processor developed by New England Digital.
- The WireWorld based computer.
- Dr. Dobb’s published One-Der a 32-bit TTA in Verilog with a matching cross assembler and Forth compiler.[6][7]
- Mali (200/400) vertex processor, uses a 128-bit instruction word single precision floating point scalar TTA.

3.6 See also

- Application-specific instruction-set processor (ASIP)
- Very long instruction word (VLIW)
- Explicitly parallel instruction computing (EPIC)
- Dataflow architecture

3.7 References

[1] V. Guzma, P. Jääskeläinen, P. Kellomäki, and J. Takala, “Impact of Software Bypassing on Instruction Level Parallelism and Register File Traffic”

[2] “MAXQ Family User’s Guide”. Section “1.1 Instruction Set” says “A register-based, transport-triggered architecture allows all instructions to be coded as simple transfer operations. All instructions reduce to either writing an immediate value to a destination register or memory location or moving data between registers and/or memory locations.”

[3] Introduction to the MAXQ Architecture – Includes transfer map diagram

[4] TTA Codesign Environment, an open source (MIT licensed) toolset for design of application specific TTA processors.

[5] Article about TTAs, explaining how the TTA-based Codesign Environment project uses LLVM

[6] Dr. Dobb's article with 32-bit FPGA CPU in Verilog

[7] Web site with more details on the Dr. Dobb's CPU

3.8 External links

- MOVE project: Automatic Synthesis of Application Specific Processors
  - Advantages of transport-triggered architectures
- Microprocessor Architectures from VLIW to TTA
- BYTE overview article

3.9 Text and image sources, contributors, and licenses

3.9.1 Text

- **MCU** *Source:* https://en.wikipedia.org/wiki/MCU?oldid=706674302 *Contributors:* DopefishJustin, Shizhao, Rfc1394, Pengo, Soman, CanisRufus, CyberSkull, RJFJR, Zntrip, Lapunkd, Yuriybrisk, Zbxgscqf, Nihiltres, Chobot, YurikBot, Durval, Zyxw, ERcheck, Bluebot, TimBentley, Fishhead64, Gobonobo, Woodshed, Thijs!bot, Jasonbrouwer, STBot, WarthogDemon, Mikael Häggström, Mentally Challenged University, Marlene Sinclair, AlleborgoBot, Dsmcu, Rilak, FaithLehaneTheVampireSlayer, Addbot, Shimito, Walruse, Depictionimage, FrescoBot, Harry Audus, Zollerriia, Onamreh, ClueBot NG, Iamozy, Mmontgomry, Twittelator and Anonymous: 34
- **Application-specific instruction set processor** *Source:* https://en.wikipedia.org/wiki/Application-specific_instruction_set_processor?oldid=673432181 *Contributors:* Bamakhrama, Tweenk, ZeroOne, Intgr, RussBot, Toffile, Rwwww, SmackBot, PekkaJ, FollowTheMedia, Underpants, Kozuch, EnOreg, Addbot, Dawynn, SpBot, Steven G Cox, AnomieBOT, I dream of horses, AndyHe829, Drdirkd, DrMKJainMLSU, Foia req, Shahriar.cwc and Anonymous: 5
- **Transport triggered architecture** *Source:* https://en.wikipedia.org/wiki/Transport_triggered_architecture?oldid=710636714 *Contributors:* Damian Yerrick, Shd~enwiki, DavidCary, Abdull, ZeroOne, Pearle, Alansohn, Kri, Rmky87, Cedar101, Rwwww, Chris Chittleborough, SmackBot, PekkaJ, VladoG~enwiki, JonHarder, Cybercobra, 16@r, UncleDouggie, Raysonho, Yettie0711, WhatamIdoing, Teknomunk, Hugh16, EnOreg, MarkMLl, Wikicat, SilvonenBot, MystBot, PerttiK, Addbot, Yobot, MinimanDragon32, Txt.file, John of Reading, ZéroBot, BG19bot, Comp.arch and Anonymous: 19

3.9.2 Images

- **File:Disambig_gray.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/5/5f/Disambig_gray.svg *License:* Cc-by-sa-3.0 *Contributors:* ? *Original artist:* ?
- **File:Microelectronics_stub.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/c/c3/Microelectronics_stub.svg *License:* LGPL *Contributors:* Integrated circuit icon.svg: *Original artist:* Integrated_circuit_icon.svg: Everaldo Coelho and YellowIcon
- **File:Text_document_with_red_question_mark.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/a/a4/Text_document_with_red_question_mark.svg *License:* Public domain *Contributors:* Created by bdesham with Inkscape; based upon Text-x-generic.svg from the Tango project. *Original artist:* Benjamin D. Esham (bdesham)
- **File:Transport_Triggered_Architecture.png** *Source:* https://upload.wikimedia.org/wikipedia/en/9/96/Transport_Triggered_Architecture.png *License:* Cc-by-sa-3.0 *Contributors:* ? *Original artist:* ?

3.9.3 Content license

- Creative Commons Attribution-Share Alike 3.0