Design Support for
Embedded Processors
and Applications

Prof. Kurt Keutzer
EECS
University of California
Berkeley, CA
keutzer@eecs.berkeley.edu

Embedded system needs meet technology constraints

- Embedded system design needs:
  - Fast-time to market
  - Predictability
  - Reliability
  - Robustness
  - Efficiency
  - Economy
- Application-specific integrated circuits failing to deliver this
  - Design risk
  - Design/tool cost
  - Unmanageable complexity
Demise of ASIC: Total IC Designs

Customer needs meet technology constraints

- Customer needs:
  - Fast-time to market
  - Predictability
  - Reliability
  - Robustness
  - Efficiency
  - Economy

- Looking for "platforms" – devices that will amortize system design costs over multiple generations

- "Based on our analysis, having a software approach is the only way to scale to the next generation," Corgan (Intel PMM said). "If you have to approach each fourfold increase in speed — from OC-48 [2.5 Gbits/second] to OC-192 [10 Gbits/s], say — with a new architecture, it's not cost-effective."
Solution: ASIC => ASSP => ASIP

ASIP: Programmable Platforms

- Develop platforms that allow for amortization of design costs over multiple generations
- Make platforms programmable so that they have maximum flexibility with minimum overhead

Example: Philips Nexperia™

MIPS™
- General Purpose RISC Processor
  - 50 to 300+ MHz
  - 32-bit or 64-bit
- Library of Device Blocks
  - Image coprocessors
  - DSPs
  - UART
  - 1394
  - USB
  - …and more

TriMedia™
- VLIW Media Processor
  - 100 to 300+ MHz
  - 32-bit or 64-bit
- Nexperia System Busses
  - PI bus
  - Memory bus
  - 32-128 bit

Flexible architecture for digital video applications
Configurable/Reconfigurable Processors

Domain-Specialization

Chameleon Systems
Specialized Micro-Architectures
FPGA Processor

Morphics
PMC Sierra
Improvs Systems
Specialized Instruction-Set Architectures
Frontier Design
ARC
Tensilica

Xilinx
Altera
Actel
eASIC
Atmel
Triscend
PMC Sierra
Adaptive Silicon

Cognigine
Cisco
EZchip
Xelerated
IBM
Motola
Lexra
Cognigine

ClearSpeed
Vitesse
Alchemy
Conexant

64 instrs/cycle
16 instrs/cycle
8 instrs/cycle
64 instrs/cycle

Galaxy of Network Processors

Number of PEs

Issue width per PE
**ASIP/Programmable Platform Characteristics**

5 Axes of the Architectural Design Space

- Approaches to Parallel Processing
  - Processing Element (PE) level
  - Instruction-level
  - Bit-level
- Elements of Special Purpose Hardware
- Structure of Memory Architectures
- Types of On-Chip Communication Mechanisms
- Use of Peripherals

Era of a single RISC PE (+ 1 DSP) over!
This is not a "run POSIX on an ARM" problem!
Explosion of application specific solutions
=> ASIPs

---

**Three Key Problem Areas Emerge**

- Development of programmable platforms/ASIP:
  - Characterizing target applications
  - Design space exploration
- Deployment of programmable platforms/ASIP:
  - Development of programming model
  - Provision of software environment
- Mapping applications onto programmable platforms/ASIP:
  - Application modeling
  - Application mapping
Addressing the problem areas

Modern Embedded Systems Compilers
Architectures and Languages

MESCAL research mission:
▲ To bring a disciplined methodology, and a supporting tool set, to the development, deployment, and programming of application-specific programmable platforms aka application specific instruction processors.

www.gigascale.org/mescal

Press coverage Sept 2002:
Programmable Platforms will Rule:
http://www.eetimes.com/story/OEG20020911S0063
High on MESCAL
http://www.eetimes.com/story/OEG20020911S0065

Three Key Problem Areas

◆ Development of programmable platforms:
  ▲ Characterizing target applications
  ▲ Design space exploration
◆ Deployment of programmable platforms:
  ▲ Development of programming model
  ▲ Provision of software environment
◆ Mapping applications onto programmable platforms
  ▲ Application modeling
  ▲ Application mapping
Complementary Issues

Heterogeneous applications
- Programming Environment
- Domain-specific models
  - Domain specific libraries
  - Environmental models
- "Software architecture"/MOC
- Primitive computation and communication mechanisms
- Mapping to implementation

Heterogeneous programmable platforms/ASIP
- Programming model
- Domain-specific presentation
  - Device Specific Libraries
  - Environmental support
- System architecture/micro-architecture/MOC
- Primitive computation and communication mechanisms

Our Approach
- Bottom-up view - create abstractions of existing devices
  - opacity - hide micro-architectural details from programmer
  - visibility - sufficient detail of the architecture to allow the programmer to improve the efficiency of the program
- Top down – experiment with existing modeling/programming environments
  - Learn from their abstractions of the devices
  - Try to maximize performance within these environments
Our Constraint/Angle/Prejudice

- In real-time embedded systems correct logical functionality can never be divorced from system performance.
- In commercial (especially consumer-oriented) embedded systems, system price is an utmost concern.
- Quantitative
  ▲ (Quantitatively) examine trade-offs among:
  ▼ Quality-of-results (e.g. speed, but also power, device cost)
  ▼ Programmer productivity (how long does all this take?)

Application: IPv4 Forwarding Benchmark
**Example Programming target – IXP1200**

- Intel IXP1200
  - Multiple processors
  - Specialized execution units
  - Hardware context swap

- "Intel Corp., ... chose a "fully programmable" architecture with plenty of space for users to add their own software — but one that turned out to be difficult to program."
- [http://www.eetimes.com/story/OEG20020830S0061](http://www.eetimes.com/story/OEG20020830S0061)

- How do we program these architectures? What's the right programming model?

---

**Base-line reference implementations from Intel**

**Assembler**
- Reference application

**uEngine C Features**
- Reference application modified
- Basic C language constructs like loops, condition statements and basic data types (char, int, float)
- IXP library defines additional data types, macros and functions (useful for common networking applications)
- Memory management is user defined. Hence explicit declaration of memory allocation (and no support for pointers).
Commercial NPU programming environment

Teja Technologies

- Teja is founded by Akash Deshpande – Student of Prof. Pravin Varaiya
- Based on his thesis “Control of Hybrid Systems” (1994)

Teja Language Features

- User interacts mostly with the graphical interface (which exports pre-defined application primitives)
- Extending the Teja primitives is done via a FSM-based model (however, this still requires coding in assembly via the graphical interface)
- Memory management for pre-defined primitives is done by Teja. User can alter this process (but is tedious and error prone)

Teja Features
**Our own NPU programming environment: NPClick**

- Based on Click
  - Popular environment for describing/implementing network applications
  - Developed by Eddie Kohler, MIT\(\rightarrow\) ICSI
- NPClick
  - Implemented subset of element library in IXP uC
  - Element communication via function calls
    - maintained semantics (packet push/pull)
  - packet storage fixed:
    - header in SRAM
    - payload in DRAM
- Designer needs to specify:
  - thread boundaries
  - thread/uEngine assignment
  - memory allocation of queues (SRAM, DRAM, Scratch)
- Opportunities for optimization (future work)
  - redundant memory loads/stores based on element/thread mapping
  - schemes for multiplexing hardware resources among multiple element instantiations (e.g. muxing TFIFO among 8 to Device’s)

**Programming Models for IXP1200**

![Throughput graph](image-url)
Productivity Estimates

- "First time" learning curve issues makes it difficult to compare the productivity of these approaches.
- Based on our experience, we estimate the following design times for implementing an IPv4 router:

<table>
<thead>
<tr>
<th></th>
<th>Time to functional correctness</th>
<th>Additional time for performance tuning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASM</td>
<td>8 weeks</td>
<td>8 weeks</td>
</tr>
<tr>
<td>uC</td>
<td>4 weeks</td>
<td>6 weeks</td>
</tr>
<tr>
<td>Teja</td>
<td>2 weeks</td>
<td>3-4 weeks</td>
</tr>
<tr>
<td>NPClick</td>
<td>2 days</td>
<td>2 weeks</td>
</tr>
</tbody>
</table>

- The advantages with Teja and NPClick come from the ability to perform design-space exploration at a higher level.

Conclusions: Programming Embedded Systems

- Neither ASICs or general-purpose processors will fill the needs of most embedded system applications.
- System design teams will increasingly choose ASIPs/programmable platforms.
- Programming these devices is a new challenge:
  - ▲ Parallelism
  - ▼ Process
  - ▼ Operator
  - ▼ Bit/gate level
  - ▲ Special-purpose execution units
- Need to develop matches between application development environments and programming models of ASIPs/programmable platforms.
- Match must consider:
  - ▲ Efficiency
  - ▲ Productivity
  - ▲ Robustness
  - ▲ Reliability