Modern (Embedded) Processor Systems

Prof. Dr. Akash Kumar
Chair for Processor Design
(Ack: my past and current students/PostDocs)
(Some slides adapted from Koren, Krishna, Anand)
Outline

- History of computer systems
- Trends in modern computer systems
- Design flow and considerations
- Modern challenges and solutions(??)
History of Hardware / VLSI

- Vacuum tube
  (Lee De Forest, 1906)

- ENIAC
  (1946, UPenn)

- Transistor
  (1947, Bardeen, Brattain, Shockley)

- Integrated circuit
  (1958, Jack Kilby)
History of Hardware / VLSI

- Intel 4004
  (1971, 1400 transistors)

- Intel Core i7 - Ivy Bridge
  (2012, >1.4 Billion transistors)

- Very Large Scale Integration (VLSI) – originally defined for chips having transistors in the order of 100,000. Other terms such as ULSI came along, but the usage VLSI remains dominant
Moore’s Law

In 1965, Intel’s Gordon Moore predicted that the number of transistors that can be integrated on single chip would double about every two years.
40 Years of microprocessor trend data

Design Productivity Gap

- Increasing number of transistors makes it harder to design the system
  - Late launch of products directly hurts profits
System Design Considerations

- System: sensor -> processor -> actuator
- Considerations
  - Technology
  - Performance
  - Power consumption
  - Volume of production
  - Upgradability / ease of maintenance
  - Reliability
  - Testability
  - Availability of CAD and software tools, IP's, hardware and software libraries
  - Cost, chip area
  - Legal and certification requirements, client specifications
  - .....

© Akash Kumar
Digital Hardware Market Segments

- Processor, GPU
- DRAM, Flash memories
- (Co-)Processor alternatives
  - ASIC (application specific integrated circuit)
  - ASSP (application specific standard product)
  - FPGA (field programmable gate array)
- Convergence as System on Chip (SoC), which may also contain analog, mixed-signal, and radio-frequency functions
Embedded systems architecture

- Trend towards Multi-Processor Systems-on-chip (MPSoC)
- Homogeneous vs heterogeneous systems
- Different memory models
- Different network architectures
  - Network-on-chip
  - Buses
Homogeneous vs heterogeneous
Homogeneous vs heterogeneous

- Heterogeneity is increasing
  - Different levels of parallelism in application
  - uProc – better for control-flow
  - DSP – better for signal processing
  - Dedicated hardware blocks needed for certain parts
  - Improves efficiency and saves power

- Homogeneous systems
  - Better for fault-tolerance
  - Only one compiled version of any application needed
  - Easier to design and replicate
  - Easy to support task migration
Memory usage
Embedded systems – local memory

Local memory is better for more predictability
Network/ bus delay may be unpredictable
Embedded systems – global memory

Global memory may be better for shared data
Communication pattern also determines which architecture is better.
Message passing OR Shared memory

Can also be off-chip!
Embedded systems – network

- Processor 1
- Processor 2
- Processor 3
- Processor 4
- Input/Output
- Memory
- Interconnection network
- Arbiter
Interconnection network-on-chip
Interconnection network – bus

Processor 1

Processor 2

Processor 3

High speed bus

Processor 4

Input/Output

Memory

Arbiter

Arbiter

Arbiter

© Akash Kumar
Point-to-point networks

Processor 1

Processor 2

Processor 3

Processor 4

Input/Output

Memory

Arbiter

© Akash Kumar
System Design – Hw/Sw Codesign

- Take decisions on whether to implement in hardware or software
  - Consider the advantages vs costs
- If hardware, whether to use commercial off the shelf (COTS) components or custom components
Modern Multimedia Embedded Systems

- Guarantee performance
- Minimize power consumption
- Run-time addition of applications
- Large number of use-cases
Predictable Design Flow
Analysis
Design
Management

Real-Time Embedded Systems

CGRA tile
FPGA tile

General Purpose
Stream Processing Unit
SIMD
Power Processor Element
Accelerator

Management

Debug Logic
ANALYSIS: Time Spent in a Restaurant
Estimating the waiting time with multiple clients

Waiter busy + another client

Average waiting time = 3 min

Waiter busy

Average waiting time = 1 min

Waiter available

Average waiting time = 0 min

Average order time = 2 min
ANALYSIS

35 min

Accurate analysis for multiple applications on an embedded system
Multiple food items need to be supported. Combinations change over time.
DESIGN

Automated design technique for multiple combinations of applications
 MANAGEMENT
Resource manager for heterogeneous systems running multiple applications
Design- and Run-time Flow

Design

Analysis

Management

Template??
Design Template

Diagram showing connections between FPGA, Accel A, Accel B, CGRA, and Mem Tile.
Design Template

CA: Communication Assist (DMA like)
Design- and Run-time Flow

Applications??

Design

Design Space Exploration

Analysis

Analysis Results

Reliability/Throughput

Applications A, B, C

Template??

FPGA

Accel A

Accel B

CGRA

Accel B

Mem Tile

42

© Akash Kumar
Design- and Run-time Flow

- Applications are known?
- Can multiple applications run simultaneously?
- Application models are available?
- Application domain(s) is known?
- Use representative applications…
Analysis – SDF Graph

- First proposed in 1987 by Edward Lee
- SDF Graphs used extensively
  - SDFG: Synchronous Data Flow Graphs
  - DSP applications
  - Multimedia applications
- Similar to task graphs with dependencies
Synchronous Dataflow Graphs

Actors
- Execution time per processor
- Memory requirement per processor

Channels
- Buffer constraints
- Token size
- Bandwidth requirements

Graph
- Throughput constraint
Analysis – SDF Graph

- Analyze deadlocks
- Check for consistency
- Compute throughput
- Model mapping of tasks on processors
- Model scheduling – depends on the algorithm
- Model communication bandwidth
- Model buffers – local memory and network interface
- Evaluate throughput-buffer trade-offs
Throughput-buffer trade-offs

A,1 2 α 3 B,2 1 β 2 C,2

© Akash Kumar
Predictable Design Flow

Use-case 1

Use-case 2

Use-case 3

Applications Specifications & Constraints

Mapping & Analysis Results

1. Mapping applications to the architecture
   - Model all aspects, leading to a predictable system
   - Verify if mapping is deadlock-free
   - Calculate buffer-distributions
   - Compute static order schedules for hard-RT apps
   - Integrated into SDF³ (Synchronous Data Flow For Free) tool flow

2. Architecture Specifications & Constraints

3. Applications

4. System Design and Synthesis

Synchronous Data Flow (SDF) graphs

- DSP & Multimedia applications
- Allow performance analysis

static int local_variable_A;
void actor_A (TypeB *toB, TypeC *toC)
{
    // calculate something
    // and write the output tokens
    toB[0] = calculate_valueB1();
    toB[1] = calculate_valueB2();
    *toC = calculate_valueC(local_variable_A);
}

© Akash Kumar
Predictable Design Flow

Multi-Application Multi-Processor

- Synthesis
  - Instantiate processing components
  - Instantiate interconnect components
  - Route connections, generate VHDL code

Software
- Generate wrapper code for each actor
- Reserve memory for communication
- Program connections, if needed

Hardware Specification

System Design and Synthesis

Tile 1
- Memory
- Periph. ↔ PE
- NI

Tile 2
- Memory
- PE
- NI

Tile 3
- Inst. ↔ PE
- Data. ↔ CA
- NI

Tile 4
- IP
- NI

Interconnect

Multi-Application Multi-Processor Synthesis

Hardware
- Instantiate processing components
- Instantiate interconnect components
- Route connections, generate VHDL code

Software
- Generate wrapper code for each actor
- Reserve memory for communication
- Program connections, if needed
Predictable Design Flow

Applications Specifications & Constraints

Use-case 1

Use-case 2

Use-case 3

Architecture Specifications & Constraints

System Design and Synthesis

Mapping & Performance Analysis

Mapping

Design Space Exploration

Analysis Results

Reliability/Energy

Throughput

A

B

C

Applications

Mapping

Hardware Specification

Arbiter

Arbiter

Arbiter

Arbiter

RM

Arbiter

Arbiter

Arbiter

Arbiter

Xilinx Tool-chain

1

2

3
Predictable Design Flow

- Design synthesized using TCL scripts
  - Script ensures compatibility with different Xilinx software versions
  - Carry out design space exploration

- Tool-flow (MAMPS) targeted towards Xilinx FPGAs
  - Virtex 6 – Xilinx ML605 board
  - Supports run-time reconfiguration

- Tool available online for use

Currently used by 20 research groups worldwide

Generated a design with 100 Microblazes!!

Xilinx Toolchain
Predictable Design Flow

Applications Specifications & Constraints

Use-case 1

A
\[ \begin{align*}
a_0 & \rightarrow a_2 \\
a_2 & \rightarrow a_3
\end{align*} \]

Use-case 2

B
\[ \begin{align*}
b_0 & \rightarrow b_1 \\
b_1 & \rightarrow b_2
\end{align*} \]

Use-case 3

C
\[ \begin{align*}
c_0 & \rightarrow c_1 \\
c_1 & \rightarrow c_2
\end{align*} \]

Architecture Specifications & Constraints

System Design and Synthesis

Mapping & Performance Analysis

Design Space Exploration

Analysis Results

Throughput

Analysis Results

Reliability/Energy

Applications

A
B
C

Mapping

Hardware Specification

Xilinx Tool-chain
One iteration decodes a single MCU (minimal coded unit)

Each MCU consists of up to 10 blocks of frequency values

WCET determined through measurement and scenario detection techniques
## Designer Effort

<table>
<thead>
<tr>
<th>Step</th>
<th>Time spent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallelizing the MJPEG code</td>
<td>&lt; 3 days</td>
</tr>
<tr>
<td>Creating the SDF graph</td>
<td>5 minutes</td>
</tr>
<tr>
<td>Gathering required actor metrics</td>
<td>1 day</td>
</tr>
<tr>
<td>Creating application model</td>
<td>1 hour</td>
</tr>
<tr>
<td>Generating architecture model</td>
<td>1 second</td>
</tr>
<tr>
<td>Mapping the design (SDF3)</td>
<td>1 minute</td>
</tr>
<tr>
<td>Generating Xilinx project (MAMPS)</td>
<td>16 seconds</td>
</tr>
<tr>
<td>Synthesis of the system</td>
<td>17 minutes</td>
</tr>
<tr>
<td><strong>Total time spent</strong></td>
<td>~ 4 days</td>
</tr>
</tbody>
</table>
Design- and Run-time Flow

Design → Analysis → Management

(Re-)Configuration??
(Re-)Configuration??

- Determine which resource to use when
- Change the device types?
- Change the device functionality?
- Change the communication?
- Change the mapping
- Change the schedule
Reconfigurable Heterogeneous MPSoC

- Customizable at run-time depending upon the application requirements
- The tasks taking a long time in software can be accelerated by configuring the programmable tiles appropriately
- The reconfigurable tiles can be configured to achieve fault-tolerance as well
- Size and cost reduction by time-multiplexing the reconfigurable hardware
Partially Reconfigurable MPSoC
Loading Processor Executable Code at Run-time
Migrating Tasks
Modern Challenges
Issues and Modern Trends

- The communication bottleneck
  - 3D Chips
  - Optical interconnects

- Leakage current limiting size reduction
  - Multi-gate or gate-all-around transistors (Intel 22nm uses 3D/tri-gate transistors)
  - Channel strain engineering, silicon-on-insulator-based technologies, and high-k/metal gate materials

- One may not fit all
  - Hardware/Software Co-design
  - Fault-tolerant / reconfigurable computing

- Power issues
  - Multi-core and heterogeneous architectures
## Technology Scaling

### Dennard scaling principles [1]

<table>
<thead>
<tr>
<th>Device Parameters</th>
<th>Scaling Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device dimension</td>
<td>1/k</td>
</tr>
<tr>
<td>Doping concentration</td>
<td>1/k</td>
</tr>
<tr>
<td>Voltage</td>
<td>1/k</td>
</tr>
<tr>
<td>Current</td>
<td>1/k</td>
</tr>
<tr>
<td>Capacitance</td>
<td>1/k</td>
</tr>
<tr>
<td>Delay time per circuit</td>
<td>1/k</td>
</tr>
<tr>
<td>Power dissipation</td>
<td>1/k²</td>
</tr>
<tr>
<td>Area</td>
<td>1/k²</td>
</tr>
<tr>
<td>Power density</td>
<td>1</td>
</tr>
</tbody>
</table>

Technology Scaling

- Digression from Dennard’s scaling beyond 65nm
  - Non-ideal voltage scaling: limit on threshold voltage scaling
  - Non-ideal gate oxide scaling
  - Sub-threshold leakage power

- Power dissipation increases with technology scaling
  - Heat localization (hot spots)
  - Higher temperature $\Rightarrow$ device wear-out
Technology Scaling and Power Density

CMOS Voltage Scaling

Power Density vs. Gate Length

Source: P. Packan (Intel), 2007 IEDM Short Course

Source: B. Meyerson (IBM), Semico Conf., January 2004

© Akash Kumar
Technology Scaling and Power Density

Transistor Scaling

Manufacturing defects (e.g. Imperfect Lithographic patterning)

Increased Variability (e.g. Random Dopant Fluctuation)

Increasing Power Density

Increase T

Increased Fault Rate

© Akash Kumar
What cause Faults?

Manufacturing Defects

Aging (a.k.a., Circuit Wearout)
What causes Faults?

- Internal Electronic Noise
- Electromagnetic Interference
What cause Faults?

1962: Mariner
1998: Mars Climate Orbiter

Bugs
Malicious attack

© Akash Kumar
Fault Classification

Fault Rate

Permanent Faults
- Manufacturing defects, wear-outs
- Non-recoverable
- Use of redundant hardware

Intermittent Faults
- Wear-outs, PVT variations
- Few cycles to few seconds or more
- Suspending system operation

Transient Faults
- Alpha and neutron particle strike
- Single event upsets
- Task re-execution and information redundancy

© Akash Kumar
Failures during Lifetime

- Three phases of system lifetime
  - Infant mortality (imperfect test, weak components)
  - Normal lifetime (transient/intermittent faults)
  - Wear-out period (circuit aging)
The Impact of Technology Scaling

- More leakage
- More process variability
- Smaller critical charges
  - Trends show soft-error rates incr. exp., 8% per tech generation
- Weaker transistors and wires
Effect on Embedded systems

- Decreased Lifetime:
  - Mission failures
  - Reduced safety in critical systems
    - Power plants, transportation, medical etc.
  - Reduced product lifetime
Effect on Embedded systems

- **Soft errors:**
  - Direct effect on reliability
    - Functional reliability
    - Timing reliability
  - Indirect effect
    - Mitigation methods lead to faster aging

Fault Tolerance Timing Overheads
Fault-Aware System Design

- Faults are inevitable....learn to live with faults !!!
- How to address them??
  - Fault prevention
  - Fault tolerance
  - Fault removal
  - Fault forecasting
Single-layer Fault tolerance

- The usual “phenomenon-based” approach
- Provide a “perfect” hardware to upper layers
Levels of Fault Tolerance

Software redundancy

Virtualization
Task migration
Redundant multithreading
Fault-tolerant scheduling

Core-level redundancy
TMR/ DWC
Dynamic verification & correction
Block-level redundancy
ECC for memory
Circuit hardening
### Application areas and requirements

- **Variation**

<table>
<thead>
<tr>
<th>Application Area</th>
<th>Priority of reliability requirements</th>
<th>Other relevant metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Functional Reliability</td>
<td>Timing Reliability</td>
</tr>
<tr>
<td>Banking</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Multimedia</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Portable multimedia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Health monitoring</td>
<td>High</td>
<td>Medium ~ High</td>
</tr>
<tr>
<td>Satellites / Space Missions</td>
<td>Medium</td>
<td>Medium ~ High</td>
</tr>
</tbody>
</table>

Not all applications require the same level of reliability.
Cross-layer Approach

Need to do a cost-benefit analysis!!

Application Design
Performance metrics, Acceptable miss-rate, Error Tolerance, Profiled data, Acceptance test time …

System Software Design
Masking factor, Execution overhead, Error detection and/or correction time, other overheads …

Hardware
Masking factor, Power/Energy overheads, Fault detection/correction overhead…

Resilience Mechanism
© Akash Kumar
Case-Study – Nanosatellites

- Light-weight: Wet mass of 1-10kg
- Small satellites: Notion of cube-sats, 1U=10x10x10
- Increasingly being used as they are cheaper to design and launch
  - 2004-2013: 75 launches in total
  - 2014 Q1: 94 launches
- Typically low earth orbit
- Satellite swarms are also used
Case-Study – Nanosatellites

- FPGA use increasing in nanosats – lower price, faster development
- Nanosats affected by high energy particles in space leading to glitches
- Most common error in FPGAs—Single Event Upset (SEU) — a transient error that might flip configuration bits
CFAED Paths
Path G: Resilience

- New technologies will have higher failure rates

![Diagram of resilience pathways]

- Application Code
- Dynamic Skeletons
- Permanent errors
- Transient errors

Materials-Inspired Paths (Paths A – E)

© Akash Kumar
Overview – Resilience at TU Dresden

Constraints: Error rates, energy, deadline
Objectives: Performance requirements
Dependencies: Software requirements

Application

Distributed Middleware
Databases
Run-time Libraries
Compiler
Networking
Operating System

Adaptive run-time manager

Configure

Fault rate sensor
Intel/Arm/AMD existing CPUs
TomaHawk experimental CPU
Architecture
Circuits
CMOS devices
Fault-injection framework
Post-CMOS devices

Entire Software Stack

Hardware

Configure

Fault rate

© Akash Kumar
Approximate Computing
The Computational Efficiency Gap

IBM Watson playing Jeopardy, 2011
Humans Approximate

Task: Division

is \( \frac{923}{21} > 1.75? \)

is \( \frac{923}{21} > 45? \)

21) 923 (43

\[
\begin{array}{c|c}
84 & 923 \\
83 & 21 \\
63 & \hline
\end{array}
\]

Effort expended increases with required accuracy

Application context dictates required accuracy of results

\(~1\) Petaflop/W

Accuracy
But Computers DO NOT

\[ \frac{923}{21} > 45 \]

NO

\[ \frac{923}{21} > 1.75 \]

YES

- Overkill (for many applications)
- Leads to inefficiency
- Can computers be more efficient by producing “just good enough” results?

© Akash Kumar
Its an Approximate World … At the Top

- No golden answer (multiple answers are equally acceptable)
  - Web search, recommendation systems

- Even the best algorithm cannot produce correct results all the time
  - Most recognition / machine learning problems

- Too expensive to produce fully correct or optimal results
  - Heuristic and probabilistic algorithms, relaxed consistency models, …
Its an Approximate World ... At the Top

- No golden answer
- Perfect/correct answers not always possible
- Too expensive to produce perfect/correct answers

- Miller-Rabin primality test
- Eventual consistency

© Akash Kumar
Approximate Computing Throughout the Stack

- No golden answer
- Perfect/correct answers not always possible
- Too expensive to produce perfect/correct answers

Programming Languages, Compilers, Runtimes

Architecture

Logic

Circuits

© Akash Kumar
Approximation in System Design

- Arising from the application level
  - Inherent lack of notion or ability for a single ‘correct’ answer
  - ‘Noisy’ or redundant real-world data
  - Perceptual limitations

- Arising from the transistor level
  - Increasing fault-rates
  - Increased effort/resource to achieve fault-tolerance

© Akash Kumar
Approximation in System Design

Application Approximation
Program Analysis for variable approximation

Approximate computing systems/architectures
Approximate computing processors
Reconfigurable approximate modules
Approximate circuit design

© Akash Kumar
Conclusions

- Transistor scaling leading to increased faults
- Designing systems to tolerate faults inevitable
- Need to handle faults at all levels of critical systems

- Applications often lack notion of a ‘correct’ result
- Immense need/potential to trade-off performance and energy consumed
Ongoing Research Activities

Reliability/Energy Optimization
- Reconfigurable approximate computing at run-time
- Optimize energy and reliability
- Minimize thermal cycling and peak temperature
- Task remapping and scheduling for dealing with faults

Processing Architecture Design
- Determine and design appropriate system architecture
- Design predictable components – network and communication assist
- Partially reconfigurable tile-based heterogeneous multiprocessor systems
- Task-migration module in hardware for predictable delay

Low-Power and Fault-Tolerant FPGA Designs
- Improving fault-tolerance of FPGA through LUT content manipulation
- Novel error-correction mechanisms for FPGAs
- Leakage-aware resource management techniques
- Electronic Design Automation – Place and Route for FPGAs
Chair for Processor Design
Questions and Answers

Email: akash.kumar@tu-dresden.de