01 – Introduction

Oscar Gustafsson

http://www.isy.liu.se/edu/kurs/TSEA26/

Motivation - Why you should read this course!

- Learn processor design
- Learn about efficient hardware design
- Learn some firmware design tricks
- The course is focused on ASIPs (*Application Specific Instruction set Processors*) and signal processing
  - Most of the ideas are applicable in other situations as well
  - For example: Creating highly efficient computing units in FPGAs

Motivation

- After this course you should be able to design a simple application specific processor, or similar device, by yourself
- The course will be challenging, but rewarding

Credits

- Andreas Ehliar wrote: Many of the slides have been adapted from slides initially created by Dake Liu (who had this course before me)
- I heavily relies on Andreas' slides (with modifications though)

Course coverage

- Focus is not on DSP algorithms
- Nor on IC basics
- Focus on **implementation**
  - In the labs, the tutorials, and the exam
  - Hardware on RTL level
  - Software on assembler level

Pre-requirements

- Basic DSP algorithm knowledge
- Basic computer engineering (CPU, assembler, etc)
- Logic design with synchronous logic
- Basic VHDL or Verilog
- Diplomatically speaking, if you don't have the pre-requisites, suffice to say that you will have gained a **lot** of knowledge when passing this course…
  - Students have passed this course before, without having all of the prerequisites, but they probably had to work pretty hard to do so.
  - (Please talk to me in the break if you have any doubts about these requirements.)

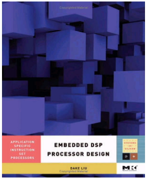
Goal of the course

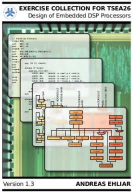
- You should be able to design a simple processor:
  - Design methods
  - The instruction set & architecture
  - The micro architecture
  - Integration and verification
  - Firmware

Scope of the course

- (10%) ASIP Design Methods
- (20%) Explore architecture and instruction set
- (50%) Micro architecture design of the core
- (10%) Integration and verification
- (10%) Firmware and programming tools

Course literature





- Dake Liu, *Embedded DSP Processor Design*
  - Should be available at local book stores
  - Also available as an ebook at `http://www.bibl.liu.se/`
- Andreas Ehliar, *Exercise Collection for TSEA26*
  - Available at `http://www.isy.liu.se/edu/kurs/`

TSEA26 Staff

- Lectures/examiner: Oscar Gustafsson
- Tutorials: Oscar Gustafsson
- Labs: Oscar Gustafsson, Erik Bertilsson
- Course homepage:
  `http://www.isy.liu.se/edu/kurs/TSEA26/`

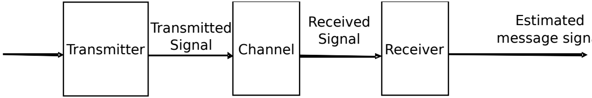
TSEA26 - Course registration

- The computer labs fit exactly 32 students
- A few late arrivals also want to attend the course
- If you for some reason want to drop the course, please let me know!

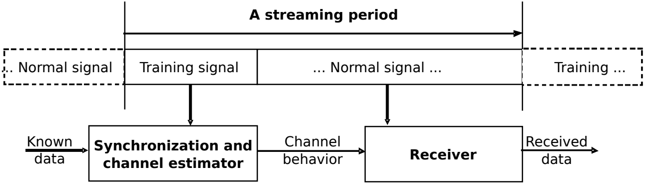
Labs

- Computer based labs
  - Four labs and 10 scheduled lab slots (10h each)
  - Recommended lab slot usage:
    - Slot 1-2: Lab 1
    - Slot 3-5: Lab 2
    - Slot 6-8: Lab 3
    - Slot 9-10: Lab 4
  - No lab sign-up list as there is only one lab group

Application example: Communication



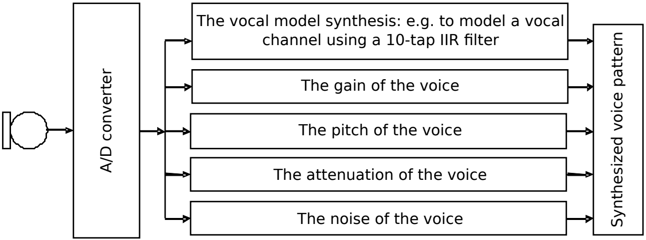
Streaming signal between a transceiver pair



Recovering data from a noisy channel

Liu2008, figure 1.13

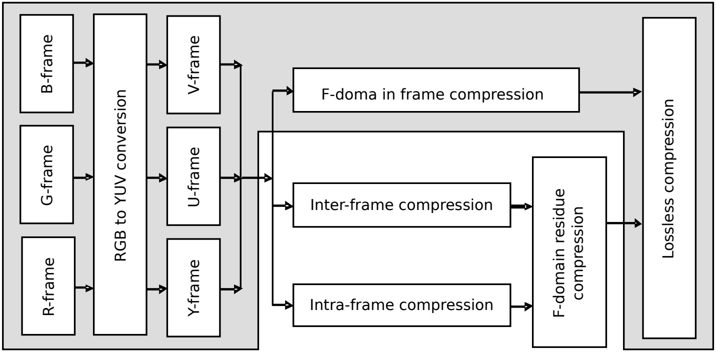
Application example: Voice coding



Can compress voice data from 104 kbit/s to 1.2 kbit/s

Liu2008, figure 1.15

Application example: Video compression



Liu2008, figure 1.16

DSP Alternatives - Normal desktop/laptop

- Suitable for many DSP applications such as video and audio processing
- Specialied instruction set extensions such as SSE (and MMX) allows for efficient parallelization of DSP tasks
- (GPU offloading is also getting more popular)

DSP Alternatives - Normal desktop/laptop

- **Not suitable for applications with hard real-time requirements**
- **Not suitable for applications with low power requirements**
- **Not suitable for medium or high volume embedded applications with low cost requirements**

DSP Alternatives - General purpose DSP processor

- Suitable for most typical DSP tasks
- Suitable for hard real-time tasks
- Probably a good choice for low or medium volume embedded applications
- **Not a good choice for very computationally demanding DSP applications**

Application specific Instruction set Processor (ASIP)

- A processor optimized for a certain kind of application domain
- Instruction set optimized for a certain application
- Accelerators for very demanding parts of the application
- Low power, high speed, low cost
- **The focus of this course**

Application Specific Integrated Circuit (ASIC)

- In theory:
  - Lowest cost (in high volume)
  - Lowest power
  - Highest performance
- In practice:
  - Very high development cost
  - Long development time
  - Only used when absolutely necessary

ASIP Design flow

- DSP architecture
- DSP FW design tools
- DSP algorithm design

Discussion break

- How do you decide on an architecture/instruction set for a processor that is supposed to be very good at running *"Application X"*?

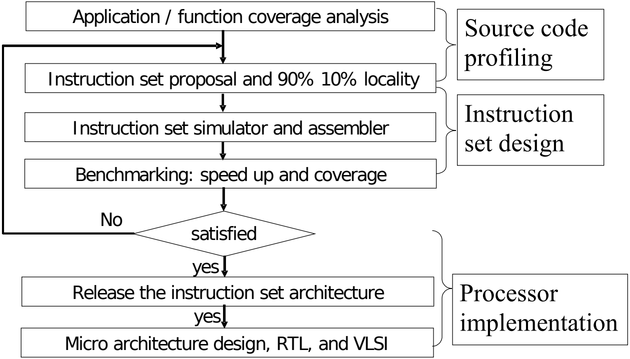
Discussion break

- How do you decide on an architecture/instruction set for a processor that is supposed to be very good at running *“Application X”*?
- **Hint:** In almost all applications there are some functions that are only run once or twice and some functions that are running almost all the time

Some possibilities, discussed during class

- What do we need to know about *Application X*?
- Size of data set, locality of data
- Real-time requirements
- Power requirements
- What kind of operations and how common they are
- Parallelization possibilities
- etc…

Simplified ASIP Design Flow



Liu2008, figure 1.26 (slightly modified)

ASIP Design in the Future

- Ideally:
  - Give a tool the source code of an application (or applications)
  - The tool automatically generates the RTL code of an ASIP optimized for this application (or applications)
  - (There are high-level synthesis tools already though)

ASIP Design in the future

- Reality check
    - Some tools are available to aid in processor design
    - Higher level than VHDL or Verilog
    - Removes the need for some of the “grunt” work
    - Typically limits the design space to reduce the complexity of the problem
    - Still requires a skilled user to make the most of it

To design an efficient ASIP today, we need

- Application specific data path and **data types**
  - **Deep understanding of data types and corner cases**
- Application specific memory access
  - Deep understanding of parallel data features
- Application specific program flow
  - Deep understanding and specification of control complexities

Todays lecture: Finite length arithmetic

- Numerical representations
- Finite length data
- Signal processing under finite data precision
- Corner cases

DEMO

- Two versions of an MP3 decoder
- The number format used for intermediate results in both MP3 decoders are 13 bits wide, yet the files sound very different. Why?

Numerical Representation

- Fixed-point: Integer
- Fixed-point: Fractional
- Floating-point basic
  - Floating-point: IEEE 754
  - Floating-point: DSP specific floating-point
- (Block floating-point/Dynamic scaling)

General requirements

- Low silicon cost requirements often lead to custom data types in DSP applications
- Typical nomenclature: word (16 bits), double word (32 bits)
- Other possibilities: custom word length for computation, memories, bus widths as required by the application

Two's-complement representation

- From hardware point of view: addition/subtraction is identical to unsigned addition/subtraction
- In this course: almost always **fixed-point** two's complement representation

Fixed-point numerical representation

- Why fixed-point **(Important)**
  - Easy to implement data path hardware
  - Low hardware cost (low chip area)
  - Short physical critical path
  - Low power

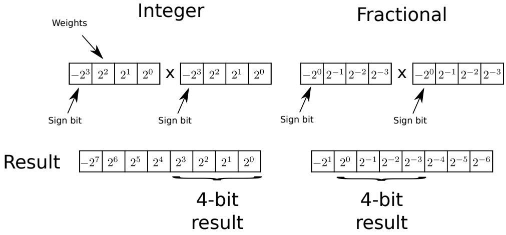
Fixed-point numerical representation

- What is fixed-point numerical representation?
  - Integer or fractional representation, dominates DSP field
  - Integer: Between -2^{n-1} and $+2^{n-1} - 1$
  - Fractional: Between -1 and $+1 - 2^{-n+1}$
  - (Where n is the number of bits)

Fractional numerical representation

- Fractional representation is straightforward:
  - $-a_0 + a_{-1} \times 2^{-1} + a_{-2} \times 2^{-2} + ...a_{-n} \times 2^{-n}$
  - a_0 is the sign bit, the range is $-1 \leq x < 1$
  - (You do remember that the secret with two's complement numbers is that the sign-bit has a negative weight rather than a positive weight?)
- Example:
  - $01010000_2 = 0.625$ (decimal)
  - $= -0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-3} = 0.625$

Fractional vs integer multiplication



- Integer: Overflow is a real concern
- Example: $0111 \times 0111 = 00110001$
  - Integer: $00110001_2 = 49_{10}$
  - Integer (truncated): $0001_2 = 1$ (Fatal error)
  - Fractional: $00.110001 = 0.765625$

Fixed point

- A more general case – the radix point can be located between any bits:
- $Q(n, m)$ notation is often used to signify this:
- n: the number of bits to the left of the radix point
- m: the number of bits to the right of the radix point
- Beware of confusion:
  - In the textbook: n includes the sign bit
  - In many other sources: n do not include the sign bit
- To be on the safe side – mention the width separately (for example, a 16-bit number in $Q(2, 14)$ format.

Useful definitions

- Precision
  - The distance between the smallest values that can be represented using a certain number format
  - Example: Using 16-bit fractional numbers, the precision of the data is $0.000000000000001_2 \approx 0.000305_{10}$

Useful definitions

- Dynamic range (of a digital signal)
  - The ratio between the largest range a certain number format can represent and the precision.
  - For example, the dynamic range of a 16-bit fixed-point number format is $65535/1$
  - Commonly measured in dB. Example: $20 \times \log_{10} 65535 \approx 96dB$. Each extra bit adds about $6dB$

Useful definitions

- Quantization error (of digital signals)
  - The numerical error introduced when a longer numeric format is converted to a shorter one
  - E.g: $s.ffffffffff \rightarrow s.ffff$
  - (Quantization errors that occur during A/D and D/A conversions are not discussed much in this course.)

Typical DSP Kernel

- Read data from memory
- Calculate using as many bits as required
- Store data in memory using as many bits as required (typically fewer)

Drawbacks of Fixed-Point

- Sometimes it is not possible to separate dynamic range and precision
- Higher firmware design costs (for example, when using a Matlab model as a reference)

Floating-Point Repetition

- Numbers are represented by a mantissa (m), an exponent (e), and a sign bit (s)
- value = $-1^s \times m \times 2^e$

IEEE-754 Standard for Floating-Point Numbers

- IEEE-754 Single precision floating-point number (32 bits)

31	30 23	22 0
s	Exponent	Mantissa

- s: Sign bit, 0 is positive, 1 is negative
- Exponent: 8-bit field in excess-127 format. If 255: A special value such as infinity or *NaN* (Not a Number)
- Mantissa: 23-bit field containing the normalized fraction (using an *implicit one*)

Why Floating-Point?

- Large dynamic range using a small number of bits
- Usually easier for the programmer $\rightarrow$ faster *time to market* (TTM)

Why not Floating Point?

- When precision is more important than dynamic range
- Complicated data path, longer critical path in the data path, higher silicon cost for arithmetic units
- Harder to reason about the stability of calculations
    - Example: $x + y + z \neq z + y + x$

IEEE-754 Standard for Floating-Point Numbers

- 23-bit mantissa (with implicit one)
- 8-bit exponent
- 1 sign bit
- Other features:
  - Rounding (Round to $+\infty$, $-\infty$, 0, and round to nearest even)
  - Subnormal numbers (sometimes called denormalized numbers)

Discussion break

- *If you are designing an application specific processor where you need floating-point numbers, can you reduce the hardware cost by using a custom FP format?*
- (Note: Not really a discussion break, rather left as something to think of until Lecture 2)

Oscar Gustafsson

www.liu.se

LINKÖPING UNIVERSITY