Functional Programming in Embedded System Design

Al Strelzoff
CTO, E-TrolZ
Lawrence, Mass.
al.strelzoff@e-trolz.com
Hard Real Time Embedded Systems MUST:

• Be highly Dependable. Zero failures.
• Do what they’re supposed to do.
• Not ever crash.
• Integrate complex hardware and software.
• Ensure behavior by some means before they are run.
What Designers Want

- No run time exceptions:
  - No null pointers.
  - No out of range arrays.
  - No class casts.
  - No arithmetic exceptions - most difficult.
- Well specified execution semantics.
- No distinction between hardware and software.
- Implicit parallelism.
- Compatibility with existing languages if possible.
Look to the Basics of Computer Science

• Functional Programming. Examples: Haskell and pH.
• Build execution semantics into the language.
  – Cycle based execution from sampling theory and synchronous computing.
  – Set the cycle rate based on input data streams.
  – Leads to synthesizable Verilog subset and synchronous dataflow graphs.
• Must be a net-list language so that we can map a program to one or more processors or to hardware.
Take Out the Garbage

- All instantiation at initialization.
- After instantiation
  - Software: the program runs in a cyclical loop at a fixed rate (or set of rates) on one or more processors.
  - Hardware: the design is mapped to RTL(Verilog).
- The design is statically analyzable before being run.
- Analyzable side effects.
Why Not Just Haskell?

• Layout syntax is not industrial strength.
• No clear treatment of memory and state.
• Input/Output?
• Object oriented design?
  – Inheritance.
  – Types and Type Classes.
Introduce a Special Memory Function

• A “register” sources and/or sinks data each cycle.
• Single assignment rules:
  – Write once in a cycle. New value updated at the end of cycle.
  – Multiple reads during a cycle (old value only).
• Synchronous, unblocked reads and writes.
• Registers are instantiated only at initialization.
• Input and Output are memory mapped to Registers.
Execution Cycle

- Fetch data from each input and from registers and place it at the input to all functions.
- Execute all functions.
- Not lazy evaluation.
- Store the produced values away in the appropriate registers.
- Conventional “drivers” must then fill and empty these registers outside the functional program.
What if the Input Data rates vary widely?

- No interrupts.
- Multiple “Tasks” each cycle at a different rate.
- Inter-task communication by rate adapting filters.
- System model is locally synchronous and globally asynchronous.
System Modeling with Finite State Machines

• Each Task can be represented as a Finite State Machine.
• The State is contained in the registers of each task.
• State changes only at an execution cycle boundary.
• Leads to a design model of concurrent FSM’s.
Software Implementation on a Stack

• The compiler converts C-like expressions to postfix.
• Execute the postfix directly on a simple stack machine.
• Is this Forth? Almost!
  – No user written postfix.
  – Strongly typed (like Haskell).
• Stack machines are small, efficient and a target for soft cores in an FPGA.
Hardware Implementation

• The front end of the compiler is the same!
• Only difference is that for hardware, the “interpreter” unwinds the postfix code into structural Verilog.
• Can we do without synthesis?
• For parallel processing, a complex mapping problem remains.
Types and Type Classes

• Type Classes:
  – Members can be sub TypeClasses or Types
  – Abstract, not instantiated.
  – Lists methods that must be supplied by its Types.

• Types:
  – Are Instantiated.
  – Supply constructor and methods required by its Type Class.

• Sub-types differentiate with relations on property values of types.
Polymorphism and Inheritance

- Simple tree like single inheritance.
- Parametric polymorphism.
- No class casts.
- Simplified form of Object Oriented Design.
Partitioning an Embedded Architecture

Separate Soft from Hard Real Time Measurement and Control

A Simple if – else branch executing. The return value is on the stack.
A “for” loop looks just like C.
A Recursive Factorial Function.
A Shape Type Class. Each Type provides its own Area implementation. Sub-Types provide relations on the properties of their parent Types.
Summary

- Start with Functional programming (Haskell).
- Add a “memory” function; a non-blocking register.
- Instantiation followed by cyclical execution determined by the sampling rate.
- A type system whose instantiated objects have “state”.
- Simplified object and inheritance model.
- C like syntax. But not C or Java compatible.
- Execute on simple stack machine(s) or translate into hardware.