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

This Slide Courtesy of E-TrolZ, Inc., www.e-trolz.com.



A Simple if – else branch executing. The return value is on the stack.

F Machine Compiler/Interpreter					
File Edit	Project Function Wir	ndow			
Functi	ion PC	Instruction	Operand	Loop	
	0	initialized		1: int Loop (int x) {	
Loop	1	push	n	2: int n = 5;	
Loop	2	pushn	5	$\begin{array}{llllllllllllllllllllllllllllllllllll$	
Loop	3	=		5: $s = s + x$:	
Loop	4	push	<u>s</u>	6: }	
Loop	5	pushn	0	7: return s;	
Loop	- 6	=		8: }	
Loop		push		9:	
Loop	8	pusnn	<u> </u>	11:	
Loop	9	=	-	12:	
Loop	10	pusn		13:	
Loop	12	pusn	S	14:	
Loop	13	- push	^	15:	
Loop	14			10:	
Loop	15	- nush	-	18:	
Loop	16	++		19:	
Loop	17	nush		20:	
Loop	18	push	n	21:	
Loop	19	<		22:	
Loop	20	imp if true	-11	23:	
Loop	21	push	s	25:	
Loop	22	return		26:	
				27:	
Data Coo	le Functions Project	Types		128.	
			(
Syr	mbol Va	lue	Туре 🔤 📤	Loop2 open for editing	
×	3	int		Loop open for editing	
s	0	int		Loop.fun is loading	
S	0	int		Loop loaded successfully	
			-		
pa La su l			•	1 JL	
Loop					

A "for" loop looks just like C.

F Machine Compiler/Interpreter							
File Edit Project Function Window							
FunctionPCInstructionOperand0initializedFact21pushnFact22pushn0Fact23==-Fact23==-Fact24imp if false2Fact25pushn1Fact26return-Fact27pushnFact28pushnFact29pushn1Fact210Fact212return-Fact212return-	<pre>Fact2 1: int Fact2(int n) { 2: if(n == 0) { 3: return 1; 4: } else { 5: return n * Fact2(n-1); 6: } 7: } 8: 9: 10: 11: 12: 13: 14: 15: 16: 17: </pre>						
Symbol Value Type	Fact2.fun is loading Fact2 loaded successfully Fact2 open for editing						
n 5 int	races open for curcing						
Fact2							

A Recursive Factorial Function.

_ 8 × F Machine Compiler/Interpreter File Edit Project Function Window 1 >4 E- Punctions Shape 🗄 🧰 Timers 1: Types Shape = Cir | Tri | Rec { └─● Timer 2: Lists 3: float Area; 🗄 🦳 🔄 Input/Output 4: Types Cir(float r) = C1 | C2{ 5: 🔶 Anin 6: Area = 3.14 + r + r;AnOut 7: 🔹 Dialn SubType C1 { 8: DigOut 9: 🖆 🧰 User Area > 10.0;10: - Case 11: } BalAdd 12: 13: SubType C2 { BalAdd2 14: Fact Area <= 10.0;15: 🔶 Eib 16: } cond1 17: 🗢 doit 18: } 🔹 array 19: Type Tri(float b, float h) { 20: foo 21: Area = 0.5 * b * h;• 🗢 🔿 🗢 22: SW } 23: Loop 24: Type Rec(float l,float w) { 🔹 Square 25: Case2 26: Area = $1 \star w$; Shape 27: SubType Sqr(float 1) { 28: Square2 29: Fact2 w = l;30: case2 31: } - 🗰 🛛 i it 32: } 33: 34: } 35: 36: 37: 38: 39: Data Code Functions Project • مه ا ► 4 Þ Shape open for editing * Sumbol • Shape - 😌 🛃 😹 🏀 👹 🕼 🕲 🥃 🥘 🖳 🖳 😔 🖉 👹 👘 👘 👘 🚺 💶 JPad P... 🔂 text 🔁 2 java 🗸 🖪 Micros... 🛛 🔞 6:09 PM 🏄 Start 📗

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.