An Esterel Virtual Machine (EVM)

Aruchunan Vaseekaran

Why

- Esterel is suited for Deterministic Control Systems
  - Imperative Language
  - Synchronous
  - Concurrency, Preemption
- Not widely available in low cost systems.

Related Work - Pascal

- EVM Concept
  - Compile code for abstract machine.
  - Write Interpreter for each target platform
  - Makes compiler very portable
  - Slower Speed

- Pascal P-Code Machine
  - 1970
  - Machine understood pascal primitive types
  - Single Stack
  - Single Threaded

Related Work - JVM

- Java JVM
  - All java code compiled down to JVM byte codes
  - Stack based – 1 Stack/Thread
  - Supports multiple threads
  - Object creation/de-allocation
  - Monitors for protecting critical code sections

Plan

- Design EVM
- Create Compiler by modifying ECL
- Implement Linux EVM & Test
- Create EVM on Lego Mindstorms
- Compare Approaches
  - Expect EVM approach to be slow
  - Expect EVM code size to be small

EVM Design – Choices

- Either have instructions for Esterel features such as threads, exceptions or C
- Or: make EVM run the C subset which EC compiles down to.
  - Simulates state of resumable instructions using state variables.
- Our Choice: have instructions for Esterel Features:
  - Less code
  - Clearer output, Easier to compile
  - Makes more sense

EVM Design – Using the EVM

- Its initialized with Byte Codes
- Repeat for every cycle:
  - Set external input signals
  - Run the EVM for a cycle
  - Retrieve external output signals

EVM Design – Registers

- Program Counter (PC)
- General Purpose Registers R0-R8
- Stack Pointer (SP) only used for expression evaluation
- State Register (SR)
- Flag Register (FL)
  - Holds result of logical ops

EVM Design – State Information

- For Each Thread:
  - Tread id
  - Address to start of thread
  - Register Save Area
  - Completion Code
    - -1 context switches (coroutine)
    - 1 paused for cycle
    - > 1 terminated with exception

EVM Design – State Information cont..

- For Each Thread
  - Store all active traps:
    - Trap id
    - Address of trap handler

EVM Design – Signal Instructions

- All signals are given unique numbers by the compiler
- Sigtst *signum*
  - Test for present
- Sigemit *signum*
  - Make signal present

EVM Design – Trap Instructions

- Trapdef id, handler_address
  - Defines trap
- Trapdel id
  - Undefines trap
- Exit id
  - Causes an exception:
    - If trap handler exists in current thread
      - Branch to it
    - Else
      - Terminate the current thread and “throw the exception”

EVM Design – Example EVM code for a trap

- trap T in
- s1;
- present S then exit T end;
- s2;
- end

EVM Design – Example EVM code for a Trap cont..

- trapdef #6, trap_end:
    - # S1
    - ..
    - # present S then
    - ..
    - exit 6
    - ...
    - ...
    - # S2
- trap_end:
    - trapdel #6

EVM Design – Thread Instructions

- Threads have unique numbers to EVM
- Threaddef id, start_address
  - Defines a thread to the EVM
- Threaddel id
- Threadrun id
  - Runs one thread for a cycle
- Pause
  - Terminates thread for this cycle only. In next cycle, control will go to instruction after pause.
- Threadone
  - Terminates this thread forever

EVM Design – Thread Instructions cont…

- Threadwait cnt,tid1,tid2,tid3
  - Runs a bunch of threads for a cycles and then reacts to their completion state.
  - If they all terminate normally, threadwait will also terminate and control goes to next instruction after threadwait.
  - If they all finish for this cycle, threadwait will also finish for this cycle.
  - If any one of them hit an exception, threadwait will wait for the remainder to finish for the cycle and then propagate the exception.

EVM Design – Coding a || Statement into Threads

- S1 || S2

- threaddef 5,S1_start:
- threaddef 6,S2_start:
- threadwait 2,5,6
- threaddel 5
- threaddel 6
- jmp par_0_end;
- S1_start: S1
- threaddone
- S2_start: S2
- threaddone
- Par_0_end:

Compiler for EVM - Approach

- modify EC, Stephen Edwards ESUIF based compiler
- ESUIF is based on a set of independent compiler passes
- Passes can be added dropped
- start from Internal Representation (IR) of EC
- IR like C with Esterel semantics
- Added passes to map IR to something close to EVM instructions

Compiler for EVM – Compiling Trap..

- Trap is mapped to try in IR
- trap T in
-         s1;
-         present S then exit T end;
-         s2;
-     end

- Try {
  - S1
  - Present S then exit T end;
  - S2
- }

Compiler for EVM – Compiling Trap..

```
• trapdef #6, trap_end:
•         # S1
•         ..
•         # present S then
•         ..
•         exit 6
•         ...
•         ...
•          # S2
• trap_end:
•         trapdel #6
```

Compiler for EVM – Compiling ||

- S1 || S2

```
•       parallel {
•          thread {
•                   pause
•                   emit A
•          }
•          thread {
•                   pause
•                   pause
•                   emit B
•          }

•       }
```

Compiler for EVM – Compiling || cont..

- threaddef 1,thread_1_start:
  - threaddef 2,thread_2_start:
  - threadwait 2,1,2
  - threaddel 1
  - threaddel 2
  - jmp parallel_1_end:

- thread_1_start:
  - pause
  - sigemit 0
  - threaddone

- thread_2_start:
  - pause
  - pause
  - sigemit 1
  - threaddone
- parallel_1_end:

Status of Compiler and EVM

- Created compiler passes for important mappings
  - try, abort, parallel
  - Hand verified
- Created Design for EVM – not implemented

Conclusions & Future Work

- Achieved
  - – EVM Design
  - – Compiler Design
  - – Implemented core compiler pieces and verified
- Future Work
  - – Finish implementing compiler and EVM (2-3 man months)
  - – Measure performance
  - – Implement on microcontroller for validation
  - – Optimize EVM and compiler
    - Branch optimization

Lego Mindstorms