The Far End of Static Tailoring
How to Implement a Real-Time Operating System as an Application-Specific State Machine

Christian Dietrich, Martin Hoffmann, Daniel Lohmann
{dietrich,hoffmann,lohmann}@cs.fau.de

Friedrich-Alexander University
Erlangen-Nuremberg

Event-Triggered vs. State-Machine Systems

**Event-Triggered Systems**
- Threads and interrupts
- Explicit synchronization
- Prioritized scheduling
- Easy to implement in software
- Familiar OS interface
- Implementation is hard to verify

**State-Machine Systems**
- States, events, and actions
- Implicit synchronized
- Implicit prioritization
- Easy to implement in hardware
- Unintuitive for complex problems
- Verification (almost) trivial
Event-Triggered vs. State-Machine Systems

**Event-Triggered Systems**
- Threads and interrupts
- Explicit synchronization
- Prioritized scheduling
- Easy to implement in software
- Familiar OS interface
- Implementation is hard to verify

**State-Machine Systems**
- States, events, and actions
- Implicit synchronized
- Implicit prioritization
- Easy to implement in hardware
- Unintuitive for complex problems
- Verification (almost) trivial

**Markup Language** → **Automatic Translation** → **Implementation**
The operating-system memory is the state of state machine

- System calls manipulate the OS state
- System semantic describes the OS’ reaction (e.g., scheduling policy)
The operating-system memory is the state of state machine
- System calls manipulate the OS state
- System semantic describes the OS’ reaction (e.g., scheduling policy)
The operating-system memory is the state of state machine

- System calls manipulate the OS state
- System semantic describes the OS’ reaction (e.g., scheduling policy)
Our Idea in a Nutshell

Application Specific

Application Logic (CFG)

System Configuration

Kernel Semantic
Our Idea in a Nutshell

Application Specific

Application Logic (CFG)

System Configuration

Kernel Semantic

Interaction Model (State Machine)
Our Idea in a Nutshell

Application Specific

Application Logic (CFG)

System Configuration

Kernel Semantic

+ 

Interaction Model (State Machine)

Kernel as Software FSM

Hardware Implementation
Question 1: How to calculate the interaction model?

Question 2: How to implement the state machine?
Question 1: How to calculate the interaction model?

State-Transition Graph → Finite State Machine

Question 2: How to implement the state machine?
Example Application

ISR (priority: $\infty$)

```
if (comp1()) {
    ActivateTask(T1);
}
```

Task: T1 (priority: 1)

```
comp2();
TerminateTask();
```

Task: Idle (priority: 0)

```
while(1) {
    idle();
}
```
Example Application

ISR (priority: ∞)

- irq_enter();
- comp1();
- ActivateTask(T1);
- irq_leave();

Task: T1 (priority: 1)

- comp2();
- TerminateTask();

Task: Idle (priority: 0)

- idle();
System-State Enumeration

(State Transition Graph)

idle()

(Abstract System State)

ISR

T1

suspended

ISR

suspended

Resume Point

Task State

Next Block

ISR()

if (comp1()) {
    ActivateTask(T1);
}

Task(T1) {
    comp2();
    TerminateTask();
}
System-State Enumeration

State Transition Graph

Abstract System State

ISR() {
    if (comp1()) {
        ActivateTask(T1);
    }
}
Task(T1) {
    comp2();
    TerminateTask();
}
System-State Enumeration

State Transition Graph

- idle()
- irq_enter()
- IRQ

Abstract System State

<table>
<thead>
<tr>
<th>Task State</th>
<th>ISR</th>
</tr>
</thead>
<tbody>
<tr>
<td>suspended</td>
<td>running</td>
</tr>
<tr>
<td>-</td>
<td>irq_enter()</td>
</tr>
</tbody>
</table>

ISR() {
  if (comp1()) {
    ActivateTask(T1);
  }
}

Task(T1) {
  comp2();
  TerminateTask();
}

ISR

1Cross-Kernel Control-Flow–Graph Analysis for Event-Driven Real-Time Systems; Dietrich, Hoffmann, Lohmann
System-State Enumeration

State Transition Graph

- `idle()`
- `irq_enter()`
- `comp1()`

Abstract System State

- **ISR**
  - Running
  - `comp1()`

- **Task** State
  - Suspected
  - `-`

Next Block

- `ISR`
  - `comp1()`

ISR() {
  if (comp1()) {
    ActivateTask(T1);
  }
}

Task(T1) {
  comp2();
  TerminateTask();
}
System-State Enumeration

State Transition Graph

- idle()
- irq_enter()
- comp1()
- irq_leave()

Abstract System State

- Task T1
  - State: suspended
  - Resume Point: -
- ISR
  - State: running
  - Resume Point: irq_leave()

ISR() {
  if (comp1()) {
    ActivateTask(T1);
  }
}
Task(T1) {
  comp2();
  TerminateTask();
}
System-State Enumeration

State Transition Graph

Abstract System State

ISR() {
    if (comp1()) {
        ActivateTask(T1);
    }
}
Task(T1) {
    comp2();
    TerminateTask();
}
System-State Enumeration

State Transition Graph

idle()

irq_enter()

comp1()

irq_leave()

Abstract System State

ISR

ActivateTask(T1)

Task State

T1

suspended

ISR Resume Point

-

Next Block

ISR

ActivateTask(T1)

ISR()

if (comp1()) {
    ActivateTask(T1);
}

Task(T1) {
    comp2();
    TerminateTask();
}
System-State Enumeration

State Transition Graph

- **idle()**
  - IRQ
  - **irq_enter()**
  - **comp1()**
    - **irq_leave()**

Abstract System State

- **ISR**
  - running
    - **irq_leave()**

- **Task**
  - ready
    - **comp2()**
  - **ISR**
    - **irq_leave()**

**ISR()**

```c
ISR() {
    if (comp1()) {
        ActivateTask(T1);
    }
}
```

**Task(T1)**

```c
Task(T1) {
    comp2();
    TerminateTask();
}
```
**System-State Enumeration**

(State Transition Graph)

**Abstract System State**

- **ISR**
  - suspended
- **Task State**
  - running
  - suspended

**Resume Point**

- **irq_enter()**
- **comp1()**
- **ActivateTask(T1)**
- **irq_leave()**

**Next Block**

- **T1**
  - **comp2()**
- **T1**
  - **comp2()**

System-State Enumeration

(LCTES’15)
System-State Enumeration

(LCTES’15)

Abstract System State

Cross-Kernel Control-Flow-Graph Analysis for Event-Driven Real-Time Systems; Dietrich, Hoffmann, Lohmann

FAU The Far End of Static Tailoring
Extract state machine from state-transition graph
- Groups of states visible in system calls
- Group–group transition become FSM transitions
- System calls as triggers; scheduled task as output
Outline

- **Question 1:**
  How to calculate the interaction model?
  - State-Transition Graph $\rightarrow$ Finite State Machine

- **Question 2:**
  How to implement the state machine?
Outline

- **Question 1:** How to calculate the interaction model?
  - State-Transition Graph $\rightarrow$ Finite State Machine

- **Question 2:** How to implement the state machine?
  - Finite State Machine $\rightarrow$ Minimized Truth Table
Solve the state assignment problem

- Studied for many decades
- Many heuristic methods used in practical applications
- We used the NOVA tool (Villa et al., 1988)
A Minimized Truth Table

- Construct a truth table from the FSM
  - Each transition becomes a line in the table
  - Prepare FSM for hardware implementation
  - Minimize the table with ESPRESSO

System-Call Site Number
Next Task
Old State
New State
A Minimized Truth Table

- **Construct a truth table from the FSM**
  - Each transition becomes a line in the table
  - Prepare FSM for hardware implementation
  - Minimize the table with ESPRESSO
A Minimized Truth Table

Construct a truth table from the FSM
- Each transition becomes a line in the table
- Prepare FSM for hardware implementation
- Minimize the table with ESPRESSO

System-Call Site Number
Next Task

Old State

New State
Connecting the Application

System Calls invoke State Machine

```c
TASK(T1) {
    comp2();
    enter_kernel();
    OS_state, task = fsm_step(0b000, OS_state);
    switch_to(task);
    leave_kernel();
}
```

Implementation in Software
- Iterate over all lines; compare patterns; calculate result
- Scheduling, signaling, and coordinate needs 2 bytes of volatile memory

Implementation in Hardware (in progress)
- Application specific processor with custom instructions; e.g. kernel 0x0
- Hardware switches between register sets, when dispatching (one cycle)
Question 1: How to calculate the interaction model?
- State-Transition Graph → Finite State Machine

Question 2: How to implement the state machine?
- Finite State Machine → Minimized Truth Table
Evaluation Scenario: Quadrotor Flight Control

- 11 tasks, 3 alarms, 1 ISR
- 53 system-call sites
## Preliminary Results

### Evaluation Scenario: Quadrotor Flight Control
- 11 tasks, 3 alarms, 1 ISR
- 53 system-call sites

### Size of different intermediate steps

<table>
<thead>
<tr>
<th>Step</th>
<th>[Steps] (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>State-Transition Graph</td>
<td>$S(T)$</td>
</tr>
<tr>
<td>Minimized FSM</td>
<td>2,938</td>
</tr>
<tr>
<td>Minimized Truth-Table</td>
<td>5,144</td>
</tr>
<tr>
<td>Software ROM Memory</td>
<td>35,798</td>
</tr>
</tbody>
</table>
Conclusion

- Extraction of a specialized, application specific interaction model
  - Construct state-transition graph by system-state enumeration
  - Use state-transition graph to construct a kernel FSM

- Replace kernel by FSM implementation
  - Software-based implementation uses 2 bytes of OS volatile memory
  - Possible hardware implementation as a processor extension

Questions to the audience
- Where do you see additional applications for this approach?
- What additional knowledge has a developer that is not codified yet?