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

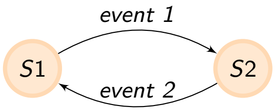
{dietrich,hoffmann,lohmann}@cs.fau.de

Friedrich-Alexander University
Erlangen-Nuremberg

7. July 2015

supported by DFG

Event-Triggered vs. State-Machine



Event-Triggered Systems

- Threads and interrupts
- Explicit synchronization
- Prioritized scheduling

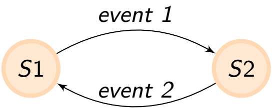
\+ Easy to implement in software
\+ Familiar OS interface
− Implementation is hard to verify

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



Event-Triggered Systems

- Threads and interrupts
- Explicit synchronization
- Prioritized scheduling

+ Easy to implement in software
+ Familiar OS interface
− Implementation is hard to verify

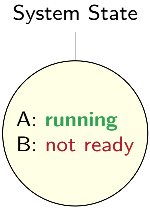
State-Machine Systems

- States, events, and actions
- Implicit synchronized
- Implicit prioritization

+ Easy to implement in hardware
− Unintuitive for complex problems
+ Verification (almost) trivial

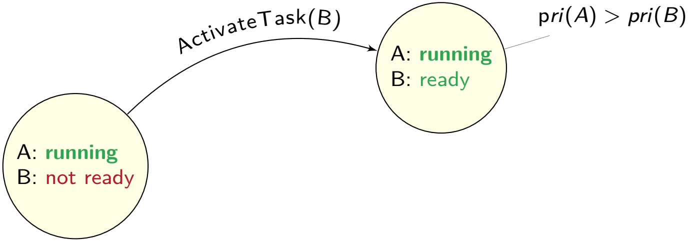
Markup Language $\xrightarrow[\text{Translation}]{\text{Automatic}}$ **Implementation**

The Operating System as a State Machine



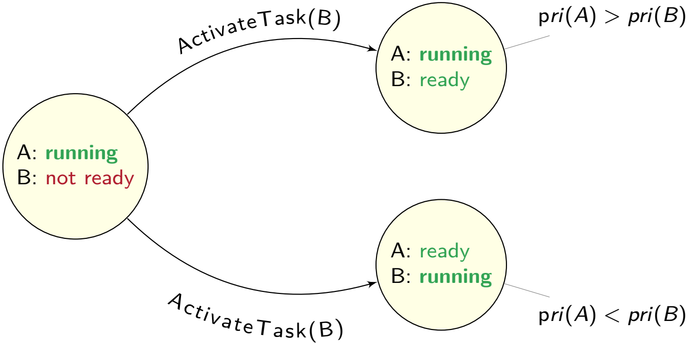
- The operating-system memory is the state of state machine
  - System calls manipulate the OS state
  - System semantic describes the OS' reaction (e.g., scheduling policy)

The Operating System as a State Machine



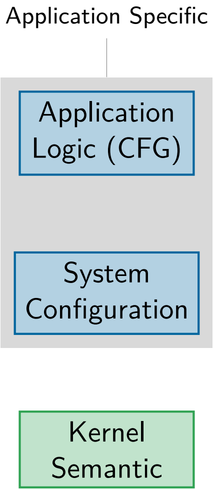
- The operating-system memory is the state of state machine
  - System calls manipulate the OS state
  - System semantic describes the OS' reaction (e.g., scheduling policy)

The Operating System as a State Machine

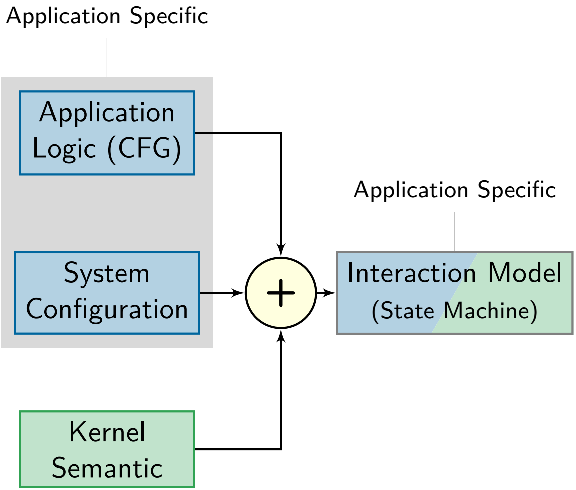


- The operating-system memory is the state of state machine
  - System calls manipulate the OS state
  - System semantic describes the OS' reaction (e.g., scheduling policy)

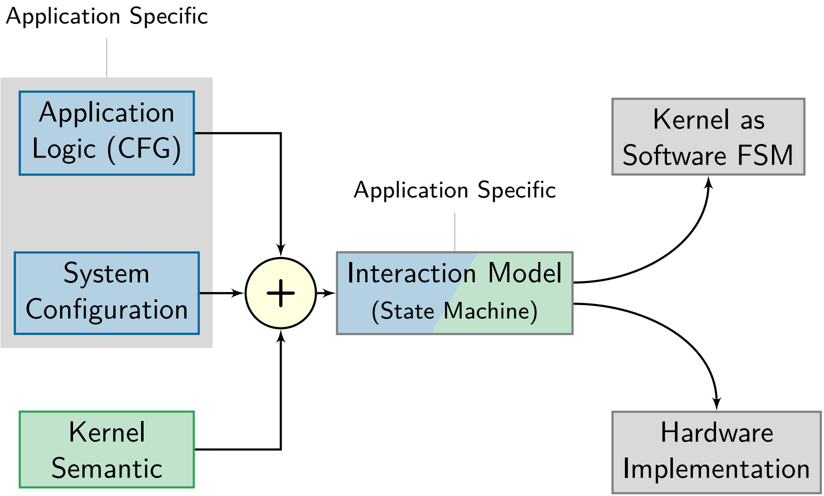
Our Idea in a Nutshell



Our Idea in a Nutshell



Our Idea in a Nutshell



Outline

- **Question 1:**
  How to calculate the interaction model?

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

Outline

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

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

Example Application

→ CFG | Computation | System Call

ISR (priority: ∞)

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

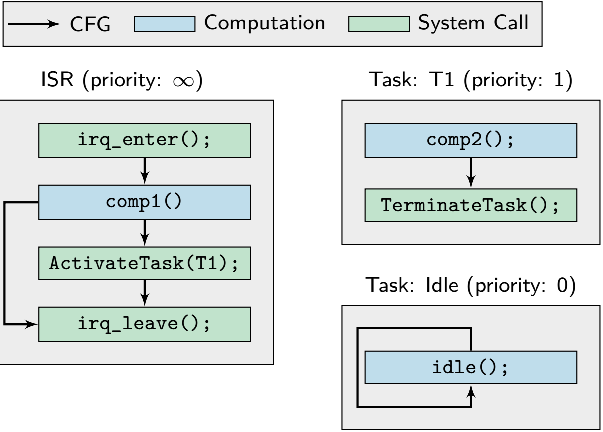
Task: T1 (priority: 1)

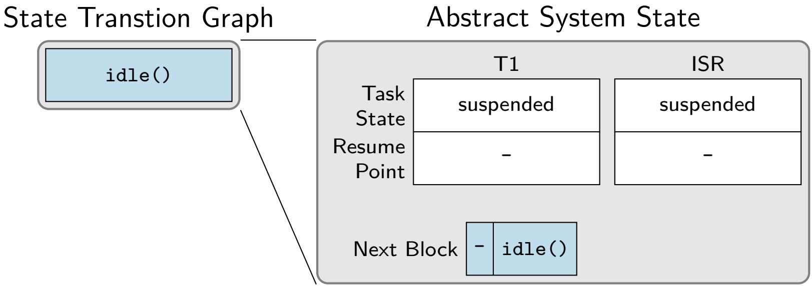
```
comp2();
TerminateTask();
```

Task: Idle (priority: 0)

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

Example Application



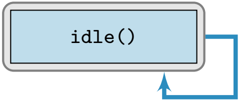



```
ISR() {
 if (comp1()) {
   ActivateTask(T1);
 }
}
Task(T1) {
  comp2();
  TerminateTask();
}
```

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

System-State Enumeration (LCTES'15[1])

State Transtion Graph



Abstract System State

	T1	ISR
Task State	suspended	suspended
Resume Point	–	–

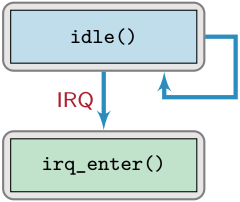
Next Block | – | idle() |

```
ISR() {
 if (comp1()) {
   ActivateTask(T1);
 }
}
Task(T1) {
  comp2();
  TerminateTask();
}
```

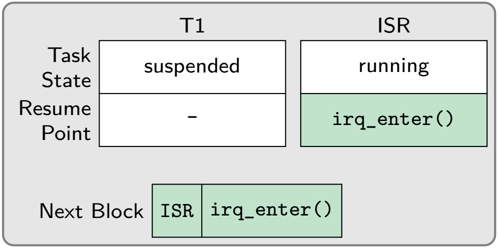
[1] Cross-Kernel Control-Flow–Graph Analysis for Event-Driven Real-Time Systems; Dietrich, Hoffmann, Lohmann

System-State Enumeration (LCTES'15[1])

State Transtion Graph



Abstract System State

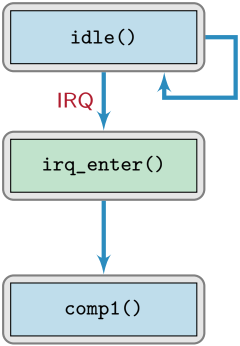



```
ISR() {
 if (comp1()) {
   ActivateTask(T1);
 }
}
Task(T1) {
  comp2();
  TerminateTask();
}
```

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

System-State Enumeration (LCTES'15[1])

State Transtion Graph



Abstract System State

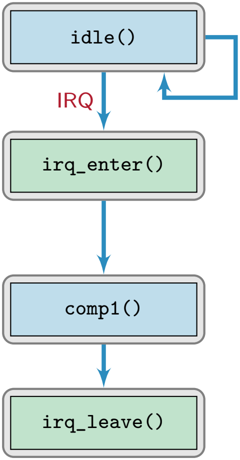
	T1	ISR
Task State	suspended	running
Resume Point	–	comp1()

Next Block: ISR | comp1()

```
ISR() {
 if (comp1()) {
   ActivateTask(T1);
 }
}
Task(T1) {
  comp2();
  TerminateTask();
}
```

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

State Transtion Graph



Abstract System State

	T1	ISR
Task State	suspended	running
Resume Point	–	irq_leave()

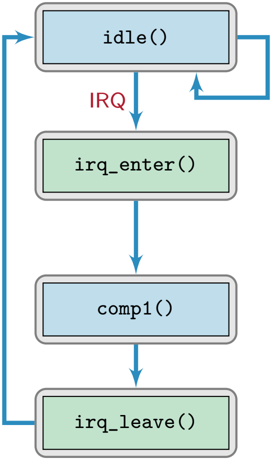
Next Block: ISR | irq_leave()

```
ISR() {
 if (comp1()) {
   ActivateTask(T1);
 }
}
Task(T1) {
  comp2();
  TerminateTask();
}
```

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

System-State Enumeration

State Transtion Graph



Abstract System State

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

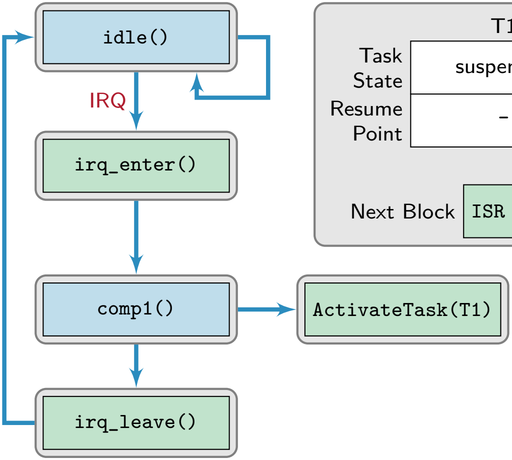
Next Block: | ISR | irq_leave() |

```
ISR() {
 if (comp1()) {
   ActivateTask(T1);
 }
}
Task(T1) {
  comp2();
  TerminateTask();
}
```

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

System-State Enumeration (LCTES'15[1])

State Transtion Graph



Abstract System State

	T1	ISR
Task State	suspended	running
Resume Point	-	`ActivateTask()`

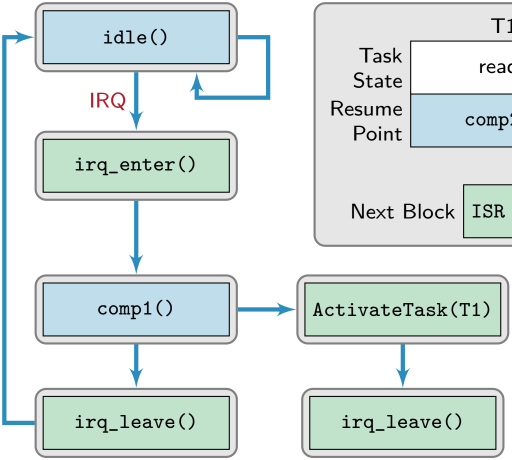
Next Block	`ISR`	`ActivateTask(T1)`

```
ISR() {
 if (comp1()) {
   ActivateTask(T1);
 }
}
Task(T1) {
  comp2();
  TerminateTask();
}
```

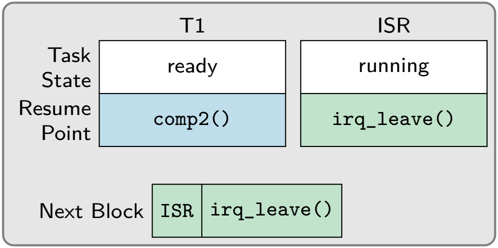
[1] Cross-Kernel Control-Flow–Graph Analysis for Event-Driven Real-Time Systems; Dietrich, Hoffmann, Lohmann

System-State Enumeration (LCTES'15[1])

State Transtion Graph



Abstract System State

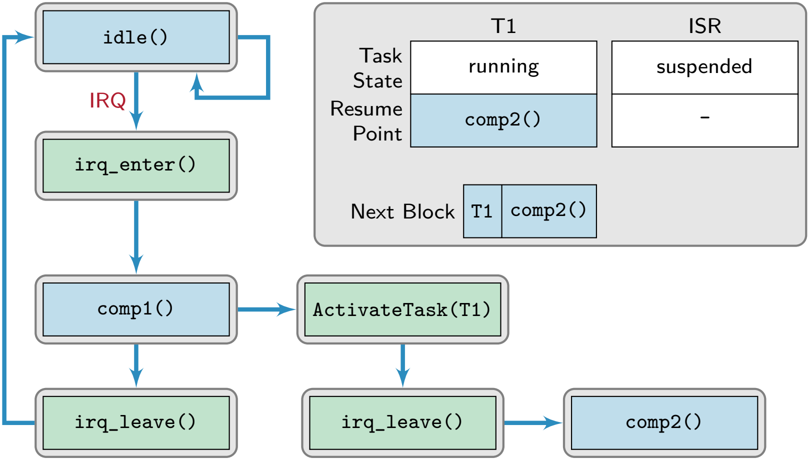



```
ISR() {
 if (comp1()) {
   ActivateTask(T1);
 }
}
Task(T1) {
  comp2();
  TerminateTask();
}
```

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

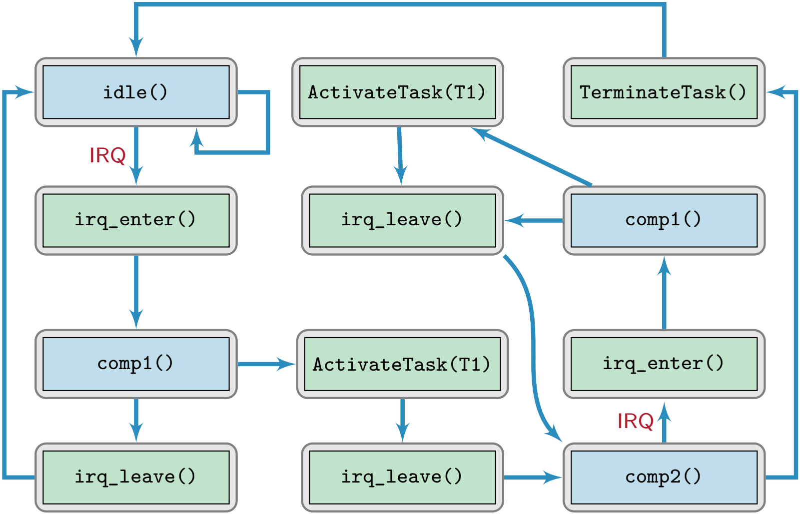
State Transtion Graph

Abstract System State



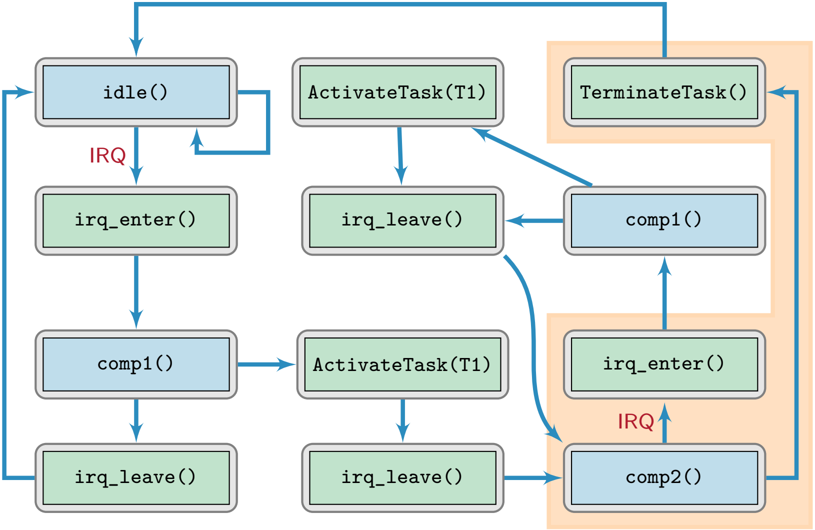
[1] Cross-Kernel Control-Flow–Graph Analysis for Event-Driven Real-Time Systems; Dietrich, Hoffmann, Lohmann

System-State Enumeration (LCTES'15[1])



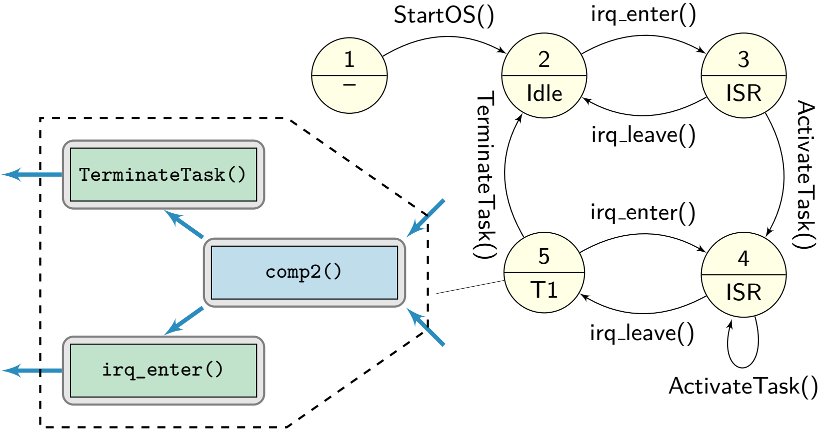
[1] Cross-Kernel Control-Flow–Graph Analysis for Event-Driven Real-Time Systems; Dietrich, Hoffmann, Lohmann

System-State Enumeration (LCTES'15[1])



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

Finite State Machine



- 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

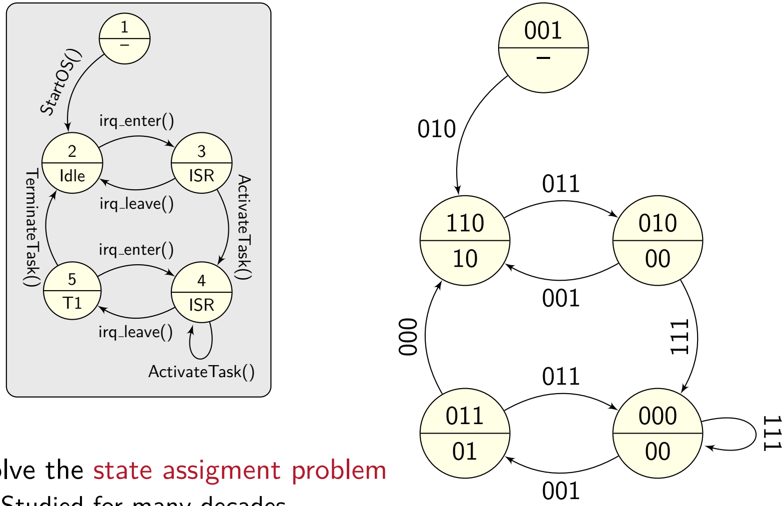
- **Question 1:**
  How to calculate the interaction model?
  ↳ State-Transition Graph $\rightarrow$ Finite State Machine ✓

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

Outline

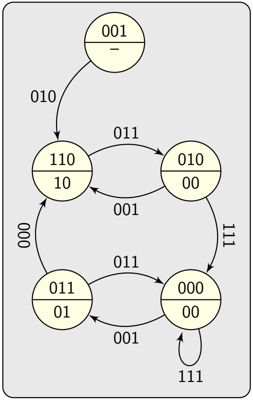
- **Question 1:**
  How to calculate the interaction model?
  ↳ State-Transition Graph $\rightarrow$ Finite State Machine ✓
- **Question 2:**
  How to implement the state machine?
  ↳ Finite State Machine $\rightarrow$ Minimized Truth Table

State Assignment Problem



- Solve the state assigment problem
  - Studied for many decades
  - Many heuristic methods used in practical applications
  - We used the NOVA tool (Villa et al., 1988)

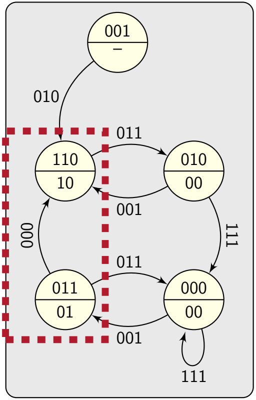
A Minimized Truth Table



System-Call Site Number	Old State	New State	Next Task
001	010	110	10
011	110	010	00
011	011	000	00
001	000	011	01
010	001	110	10
111	000	000	00
111	010	000	00
000	011	110	10

- 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

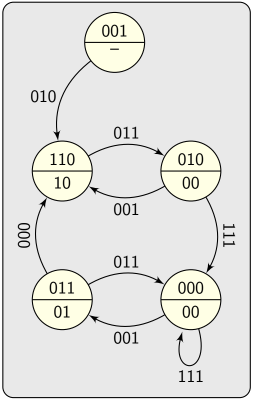


System-Call Site Number	Old State	New State	Next Task
001	010	110	10
011	110	010	00
011	011	000	00
001	000	011	01
010	001	110	10
111	000	000	00
111	010	000	00
000	011	110	10

TerminateTask

- 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



System-Call Site Number	Old State	New State	Next Task
-0-	-1-	110	10
0--	--0	010	00
-0-	-0-	001	01
---	-01	110	10

- 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

Connecting the Application

System Calls invoke State Machine

TerminateTask()

```
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)

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 ✓

Preliminary Results

- 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

Step		
State-Transition Graph	[S(T)]	$1.56 \cdot 10^6$ ($2.1 \cdot 10^6$)
Minimized FSM	[S(T)]	2,938 (8,822)
Minimized Truth-Table	[Rows]	5,144
Software ROM Memory	[Bytes]	35,798

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 implemenation
  - Software-based implemenation 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?