Static Analysis by Abstract Interpretation of Functional Properties of Device Drivers in TinyOS

Abdelraouf Ouadjaout, Antoine Miné, Noureddine Lasla, Nadjib Badache

ENS & UPMC, Paris
CERIST & USTHB, Algiers

Workshop on Static Analysis of Concurrent Software
September 11th, 2016
Edinburgh, Scotland

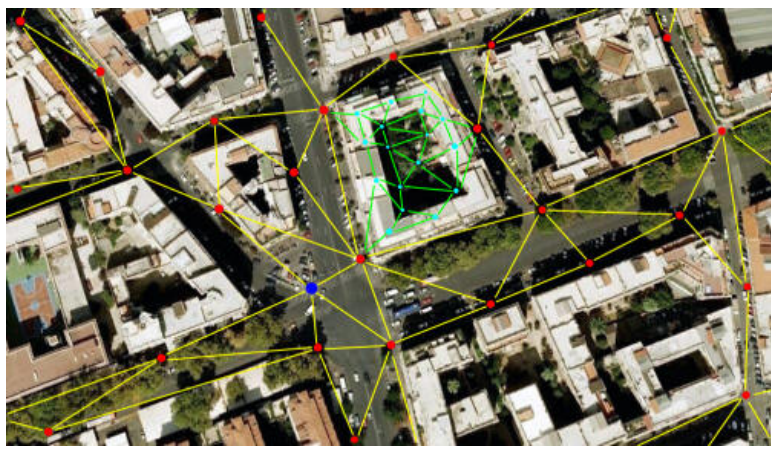
Part I

Context

Context
Application Domain: Wireless Sensor Networks

- We target programs for wireless sensor networks (WSN).
- A distributed system of wirelessly connected embedded nodes for monitoring a physical phenomena.
- Ad hoc communications and collaborative routing.
- Many applications: irrigation, weather/pollution monitoring, fire detection, *etc.*






We aim at verifying the correctness of device drivers in TinyOS programs.

Motivation

Drivers difficult to develop/debug, error-prone and critical.

Summary of Our Approach

- We focus on functional properties specifying programming rules to access hardware correctly.
- We employ Abstract Interpretation to automatically verify that all possible executions obey such specifications.

For more details:
Static Analysis by Abstract Interpretation of Functional Properties of Device Drivers in TinyOS

In *Journal of Systems and Software*, 2016.

Part II

Expressing Specifications

Specifications
Example: ATmega128 Timer/Counter0

ATmega128

Specifications
Example: ATmega128 Timer/Counter0

ATmega128

- When writing to one of the registers TCNT0, OCR0, or TCCR0, the value is transferred to a temporary register, and latched after two positive edges on TOSC1. The user should not write a new value before the contents of the Temporary Register have been transferred to its destination. Each of the three mentioned registers have their individual temporary register, which means that e.g., writing to TCNT0 does not disturb an OCR0 write in progress. To detect that a transfer to the destination register has taken place, the Asynchronous Status Register – ASSR has been implemented.
- When entering Power-save or Extended Standby mode after having written to TCNT0, OCR0, or TCCR0, the user must wait until the written register has been updated if Timer/Counter0 is used to wake up the device. Otherwise, the MCU will enter sleep mode before the changes are effective. This is particularly important if the Output Compare0 interrupt is used to wake up the device, since the output compare function is disabled during writing to OCR0 or TCNT0. If the write cycle is not finished, and the MCU enters sleep mode before the OCR0UB bit returns to zero, the device will never receive a compare match interrupt, and the MCU will not wake up.
- If Timer/Counter0 is used to wake the device up from Power-save or Extended Standby mode, precautions must be taken if the user wants to re-enter one of these modes: The interrupt logic needs one TOSC1 cycle to be reset. If the time between wake-up and re-entering sleep mode is less than one TOSC1 cycle, the interrupt will not occur, and the device will fail to wake up. If the user is in doubt whether the time before re-entering Power-save or Extended Standby mode is sufficient, the following algorithm can be used to ensure that one TOSC1 cycle has elapsed:
  1. Write a value to TCCR0, TCNT0, or OCR0.
  2. Wait until the corresponding Update Busy flag in ASSR returns to zero.
  3. Enter Power-save or Extended Standby mode.
- When the asynchronous operation is selected, the 32.768kHz Oscillator for Timer/Counter0 is always running, except in Power-down and Standby modes. After a Power-up Reset or wake-up from Power-down or Standby mode, the user should be aware of the fact that this Oscillator might take as long as one second to stabilize. The user is advised to wait for at least one second before using Timer/Counter0 after power-up or wake-up from Power-down or Standby mode. The contents of all Timer/Counter0 Registers must be considered lost after a wake-up from Power-down or Standby mode due to unstable clock signal upon start-up, no matter whether the Oscillator is in use or a clock signal is applied to the TOSC1 pin.
- Description of wake up from Power-save or Extended Standby mode when the timer is clocked asynchronously: When the interrupt condition is met, the wake up process is started on the following cycle of the timer clock, that is, the timer is always advanced by at least one before the processor can read the counter value. After wake-up, the MCU is halted for four cycles, it executes the interrupt routine, and resumes execution from the instruction following SLEEP.
- Reading of the TCNT0 Register shortly after wake-up from Power-save may give an incorrect result. Since TCNT0 is clocked on the asynchronous TOSC clock, reading TCNT0 must be done through a register synchronized to the internal I/O clock domain. Synchronization takes place for every rising TOSC1 edge. When waking up from Power-save mode, and the I/O clock ($clk_{I/O}$) again becomes active, TCNT0 will read as the previous value (before entering sleep) until the next rising TOSC1 edge. The phase of the TOSC clock after waking up from Power-save mode is essentially unpredictable, as it depends on the wake-up time. The recommended procedure for reading TCNT0 is thus as follows:
  1. Write any value to either of the registers OCR0 or TCCR0.
  2. Wait for the corresponding Update Busy Flag to be cleared.
  3. Read TCNT0.

Datasheet, pp. 107

Specifications
Example: ATmega128 Timer/Counter0

ATmega128

- When writing to one of the registers TCNT0, OCR0, or TCCR0, the value is transferred to a temporary register, and latched after two positive edges on TOSC1. The user should not write a new value before the contents of the Temporary Register have been transferred to its destination. Each of the three mentioned registers have their individual temporary register, which means that e.g., writing to TCNT0 does not disturb an OCR0 write in progress. To detect that a transfer to the destination register has taken place, the Asynchronous Status Register – ASSR has been implemented.
- When entering Power-save or Extended Standby mode after having written to TCNT0, OCR0, or TCCR0, the user must wait until the written register has been updated if Timer/Counter0 is used to wake up the device. Otherwise, the MCU will enter sleep mode before the changes are effective. This is particularly important if the Output Compare0 interrupt is used to wake up the device, since the output compare function is disabled during writing to OCR0 or TCNT0. If the write cycle is not finished, and the MCU enters sleep mode before the OCR0UB bit returns to zero, the device will never receive a compare match interrupt, and the MCU will not wake up.
- If Timer/Counter0 is used to wake the device up from Power-save or Extended Standby mode, precautions must be taken if the user wants to re-enter one of these modes: The interrupt logic needs one TOSC1 cycle to be reset. If the time between wake-up and re-entering sleep mode is less than one TOSC1 cycle, the interrupt will not occur, and the device will fail to wake up. If the user is in doubt whether the time before re-entering Power-save or Extended Standby mode is sufficient, the following algorithm can be used to ensure that one TOSC1 cycle has elapsed:
  1. Write a value to TCCR0, TCNT0, or OCR0.
  2. Wait until the corresponding Update Busy flag in ASSR returns to zero.
  3. Enter Power-save or Extended Standby mode.
- When the asynchronous operation is selected, the 32.768kHz Oscillator for Timer/Counter0 is always running, except in Power-down and Standby modes. After a Power-up Reset or wake-up from Power-down or Standby mode, the user should be aware of the fact that this Oscillator might take as long as one second to stabilize. The user is advised to wait for at least one second before using Timer/Counter0 after power-up or wake-up from Power-down or Standby mode. The contents of all Timer/Counter0 Registers must be considered lost after a wake-up from Power-down or Standby mode due to unstable clock signal upon start-up, no matter whether the Oscillator is in use or a clock signal is applied to the TOSC1 pin.
- Description of wake up from Power-save or Extended Standby mode when the timer is clocked asynchronously: When the interrupt condition is met, the wake up process is started on the following cycle of the timer clock, that is, the timer is always advanced by at least one before the processor can read the counter value. After wake-up, the MCU is halted for four cycles, it executes the interrupt routine, and resumes execution from the instruction following SLEEP.
- Reading of the TCNT0 Register shortly after wake-up from Power-save may give an incorrect result. Since TCNT0 is clocked on the asynchronous TOSC clock, reading TCNT0 must be done through a register synchronized to the internal I/O clock domain. Synchronization takes place for every rising TOSC1 edge. When waking up from Power-save mode, and the I/O clock ($clk_{I/O}$) again becomes active, TCNT0 will read as the previous value (before entering sleep) until the next rising TOSC1 edge. The phase of the TOSC clock after waking up from Power-save mode is essentially unpredictable, as it depends on the wake-up time. The recommended procedure for reading TCNT0 is thus as follows:
  1. Write any value to either of the registers OCR0 or TCCR0.
  2. Wait for the corresponding Update Busy Flag to be cleared.
  3. Read TCNT0.

Datasheet, pp. 107

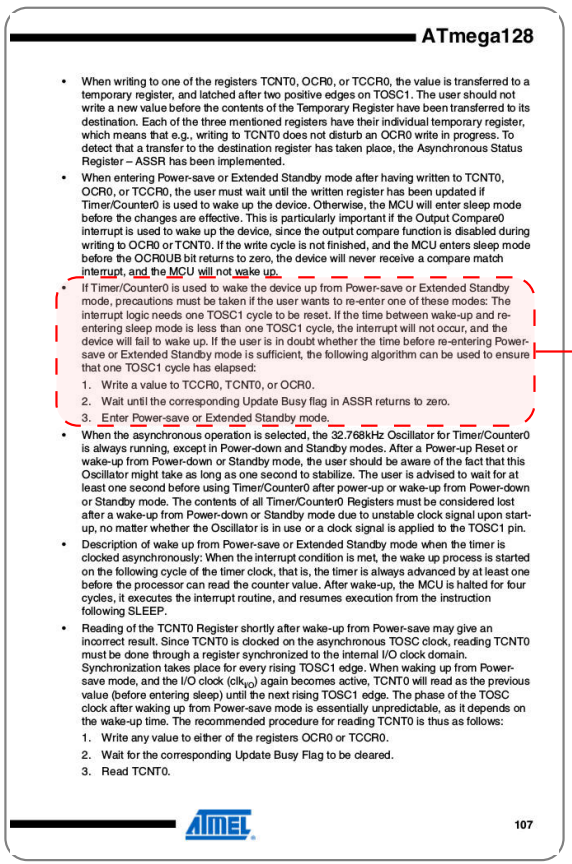
Rule

If Timer/Counter0 is used to wake the device up [...], precautions must be taken [...]

1. Write a value to `TCCR0`, `TCNT0`, or `OCR0`.
2. Wait until the corresponding Update Busy flag in `ASSR` returns to zero.
3. Enter Power-save or Extended Standby mode.

Specifications
Example: ATmega128 Timer/Counter0

ATmega128



ATmega128

- When writing to one of the registers TCNT0, OCR0, or TCCR0, the value is transferred to a temporary register, and latched after two positive edges on TOSC1. The user should not write a new value before the contents of the Temporary Register have been transferred to its destination. Each of the three mentioned registers have their individual temporary register, which means that e.g., writing to TCNT0 does not disturb an OCR0 write in progress. To detect that a transfer to the destination register has taken place, the Asynchronous Status Register – ASSR has been implemented.
- When entering Power-save or Extended Standby mode after having written to TCNT0, OCR0, or TCCR0, the user must wait until the written register has been updated if Timer/Counter0 is used to wake up the device. Otherwise, the MCU will enter sleep mode before the changes are effective. This is particularly important if the Output Compare0 interrupt is used to wake up the device, since the output compare function is disabled during writing to OCR0 or TCNT0. If the write cycle is not finished, and the MCU enters sleep mode before the OCR0UB bit returns to zero, the device will never receive a compare match interrupt, and the MCU will not wake up.
- If Timer/Counter0 is used to wake the device up from Power-save or Extended Standby mode, precautions must be taken if the user wants to re-enter one of these modes: The interrupt logic needs one TOSC1 cycle to be reset. If the time between wake-up and re-entering sleep mode is less than one TOSC1 cycle, the interrupt will not occur, and the device will fail to wake up. If the user is in doubt whether the time before re-entering Power-save or Extended Standby mode is sufficient, the following algorithm can be used to ensure that one TOSC1 cycle has elapsed:
  1. Write a value to TCCR0, TCNT0, or OCR0.
  2. Wait until the corresponding Update Busy flag in ASSR returns to zero.
  3. Enter Power-save or Extended Standby mode.
- When the asynchronous operation is selected, the 32.768kHz Oscillator for Timer/Counter0 is always running, except in Power-down and Standby modes. After a Power-up Reset or wake-up from Power-down or Standby mode, the user should be aware of the fact that this Oscillator might take as long as one second to stabilize. The user is advised to wait for at least one second before using Timer/Counter0 after power-up or wake-up from Power-down or Standby mode. The contents of all Timer/Counter0 Registers must be considered lost after a wake-up from Power-down or Standby mode due to unstable clock signal upon start-up, no matter whether the Oscillator is in use or a clock signal is applied to the TOSC1 pin.
- Description of wake up from Power-save or Extended Standby mode when the timer is clocked asynchronously: When the interrupt condition is met, the wake up process is started on the following cycle of the timer clock, that is, the timer is always advanced by at least one before the processor can read the counter value. After wake-up, the MCU is halted for four cycles, it executes the interrupt routine, and resumes execution from the instruction following SLEEP.
- Reading of the TCNT0 Register shortly after wake-up from Power-save may give an incorrect result. Since TCNT0 is clocked on the asynchronous TOSC clock, reading TCNT0 must be done through a register synchronized to the internal I/O clock domain. Synchronization takes place for every rising TOSC1 edge. When waking up from Power-save mode, and the I/O clock ($clk_{I/O}$) again becomes active, TCNT0 will read as the previous value (before entering sleep) until the next rising TOSC1 edge. The phase of the TOSC clock after waking up from Power-save mode is essentially unpredictable, as it depends on the wake-up time. The recommended procedure for reading TCNT0 is thus as follows:
  1. Write any value to either of the registers OCR0 or TCCR0.
  2. Wait for the corresponding Update Busy Flag to be cleared.
  3. Read TCNT0.

107

Datasheet, pp. 107

Rule

If Timer/Counter0 is used to wake the device up [...], precautions must be taken [...]

1. Write a value to `TCCR0`, `TCNT0`, or `OCR0`.
2. Wait until the corresponding Update Busy flag in `ASSR` returns to zero.
3. Enter Power-save or Extended Standby mode.

Formalism

To formalize this rule, we use an automaton tailored to describe patterns of hardware interactions

Specifications
Abstract Device Properties

An *abstract device property* is a special register automaton describing patterns of hardware interactions:

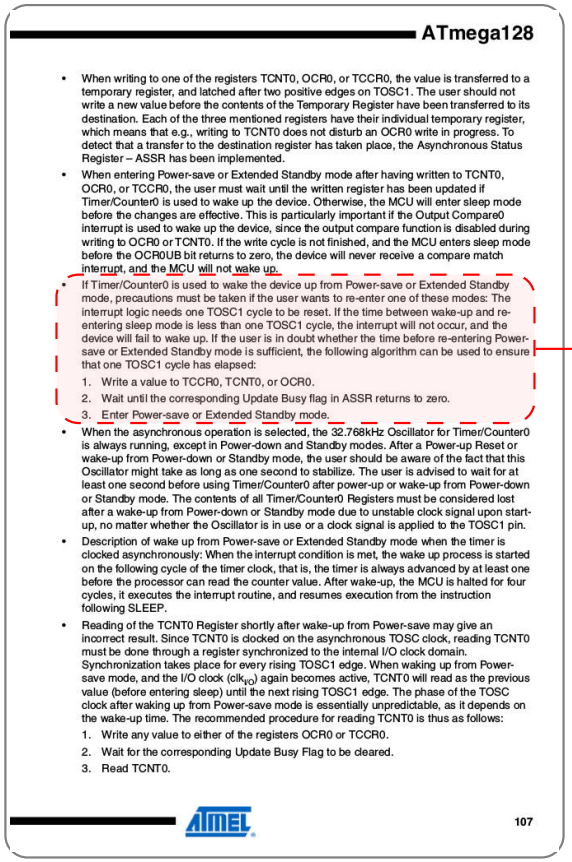
$$\mathcal{A} = (\mathcal{S}, s_0, s_{\texttt{BUG}}, \mathcal{R}, \xi, \mathcal{T})$$

where:

$\mathcal{S}$	set of states
s_0	initial state
$s_{\texttt{BUG}}$	bug state
$\mathcal{R}$	set of hardware registers
ξ	$= \{\texttt{X}^{\diamond} \mid \texttt{X} \in \mathcal{R}, \diamond \in \{r, w\}\} \cup \{\texttt{int}_i \mid i \in \mathbb{I}\} \cup \{\alpha, \texttt{sleep}\}$
$\mathcal{T}$	$\subseteq \mathcal{S} \times \xi \times \mathcal{S} \times Stmt_C \times Stmt_C$

Specifications
ADP Example

ATmega128



ATmega128

- When writing to one of the registers TCNT0, OCR0, or TCCR0, the value is transferred to a temporary register, and latched after two positive edges on TOSC1. The user should not write a new value before the contents of the Temporary Register have been transferred to its destination. Each of the three mentioned registers have their individual temporary register, which means that e.g., writing to TCNT0 does not disturb an OCR0 write in progress. To detect that a transfer to the destination register has taken place, the Asynchronous Status Register – ASSR has been implemented.
- When entering Power-save or Extended Standby mode after having written to TCNT0, OCR0, or TCCR0, the user must wait until the written register has been updated if Timer/Counter0 is used to wake up the device. Otherwise, the MCU will enter sleep mode before the changes are effective. This is particularly important if the Output Compare0 interrupt is used to wake up the device, since the output compare function is disabled during writing to OCR0 or TCNT0. If the write cycle is not finished, and the MCU enters sleep mode before the OCR0UB bit returns to zero, the device will never receive a compare match interrupt, and the MCU will not wake up.
- If Timer/Counter0 is used to wake the device up from Power-save or Extended Standby mode, precautions must be taken if the user wants to re-enter one of these modes: The interrupt logic needs one TOSC1 cycle to be reset. If the time between wake-up and re-entering sleep mode is less than one TOSC1 cycle, the interrupt will not occur, and the device will fail to wake up. If the user is in doubt whether the time before re-entering Power-save or Extended Standby mode is sufficient, the following algorithm can be used to ensure that one TOSC1 cycle has elapsed:
  1. Write a value to TCCR0, TCNT0, or OCR0.
  2. Wait until the corresponding Update Busy flag in ASSR returns to zero.
  3. Enter Power-save or Extended Standby mode.
- When the asynchronous operation is selected, the 32.768kHz Oscillator for Timer/Counter0 is always running, except in Power-down and Standby modes. After a Power-up Reset or wake-up from Power-down or Standby mode, the user should be aware of the fact that this Oscillator might take as long as one second to stabilize. The user is advised to wait for at least one second before using Timer/Counter0 after power-up or wake-up from Power-down or Standby mode. The contents of all Timer/Counter0 Registers must be considered lost after a wake-up from Power-down or Standby mode due to unstable clock signal upon start-up, no matter whether the Oscillator is in use or a clock signal is applied to the TOSC1 pin.
- Description of wake up from Power-save or Extended Standby mode when the timer is clocked asynchronously: When the interrupt condition is met, the wake up process is started on the following cycle of the timer clock, that is, the timer is always advanced by at least one before the processor can read the counter value. After wake-up, the MCU is halted for four cycles, it executes the interrupt routine, and resumes execution from the instruction following SLEEP.
- Reading of the TCNT0 Register shortly after wake-up from Power-save may give an incorrect result. Since TCNT0 is clocked on the asynchronous TOSC clock, reading TCNT0 must be done through a register synchronized to the internal I/O clock domain. Synchronization takes place for every rising TOSC1 edge. When waking up from Power-save mode, and the I/O clock ($clk_{I/O}$) again becomes active, TCNT0 will read as the previous value (before entering sleep) until the next rising TOSC1 edge. The phase of the TOSC clock after waking up from Power-save mode is essentially unpredictable, as it depends on the wake-up time. The recommended procedure for reading TCNT0 is thus as follows:
  1. Write any value to either of the registers OCR0 or TCCR0.
  2. Wait for the corresponding Update Busy Flag to be cleared.
  3. Read TCNT0.

107

Datasheet, pp. 107

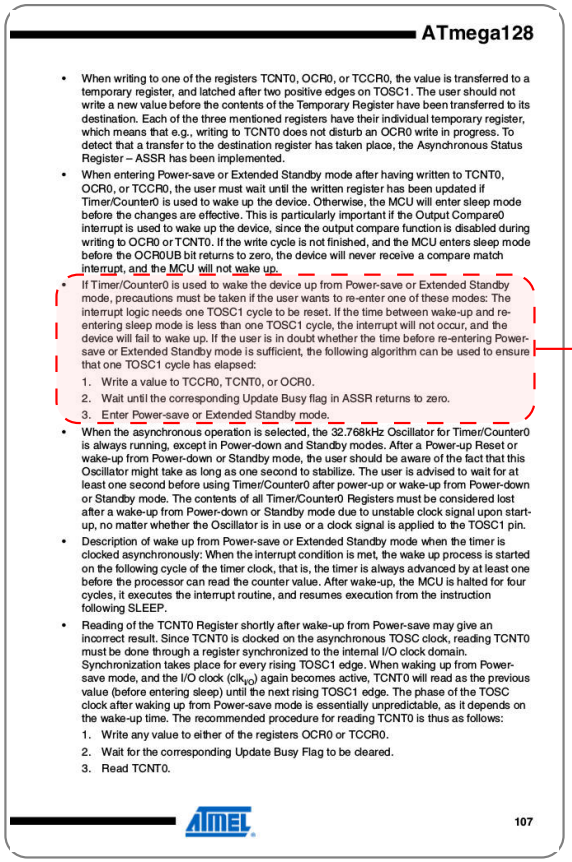
Rule

If Timer/Counter0 is used to wake the device up [...], precautions must be taken [...]

1. Write a value to `TCCR0`, `TCNT0`, or `OCR0`.
2. Wait until the corresponding Update Busy flag in `ASSR` returns to zero.
3. Enter Power-save or Extended Standby mode.

Specifications
ADP Example

ATmega128



ATmega128

- When writing to one of the registers TCNT0, OCR0, or TCCR0, the value is transferred to a temporary register, and latched after two positive edges on TOSC1. The user should not write a new value before the contents of the Temporary Register have been transferred to its destination. Each of the three mentioned registers have their individual temporary register, which means that e.g., writing to TCNT0 does not disturb an OCR0 write in progress. To detect that a transfer to the destination register has taken place, the Asynchronous Status Register – ASSR has been implemented.
- When entering Power-save or Extended Standby mode after having written to TCNT0, OCR0, or TCCR0, the user must wait until the written register has been updated if Timer/Counter0 is used to wake up the device. Otherwise, the MCU will enter sleep mode before the changes are effective. This is particularly important if the Output Compare0 interrupt is used to wake up the device, since the output compare function is disabled during writing to OCR0 or TCNT0. If the write cycle is not finished, and the MCU enters sleep mode before the OCR0UB bit returns to zero, the device will never receive a compare match interrupt, and the MCU will not wake up.
- If Timer/Counter0 is used to wake the device up from Power-save or Extended Standby mode, precautions must be taken if the user wants to re-enter one of these modes: The interrupt logic needs one TOSC1 cycle to be reset. If the time between wake-up and re-entering sleep mode is less than one TOSC1 cycle, the interrupt will not occur, and the device will fail to wake up. If the user is in doubt whether the time before re-entering Power-save or Extended Standby mode is sufficient, the following algorithm can be used to ensure that one TOSC1 cycle has elapsed:
  1. Write a value to TCCR0, TCNT0, or OCR0.
  2. Wait until the corresponding Update Busy flag in ASSR returns to zero.
  3. Enter Power-save or Extended Standby mode.
- When the asynchronous operation is selected, the 32.768kHz Oscillator for Timer/Counter0 is always running, except in Power-down and Standby modes. After a Power-up Reset or wake-up from Power-down or Standby mode, the user should be aware of the fact that this Oscillator might take as long as one second to stabilize. The user is advised to wait for at least one second before using Timer/Counter0 after power-up or wake-up from Power-down or Standby mode. The contents of all Timer/Counter0 Registers must be considered lost after a wake-up from Power-down or Standby mode due to unstable clock signal upon start-up, no matter whether the Oscillator is in use or a clock signal is applied to the TOSC1 pin.
- Description of wake up from Power-save or Extended Standby mode when the timer is clocked asynchronously: When the interrupt condition is met, the wake up process is started on the following cycle of the timer clock, that is, the timer is always advanced by at least one before the processor can read the counter value. After wake-up, the MCU is halted for four cycles, it executes the interrupt routine, and resumes execution from the instruction following SLEEP.
- Reading of the TCNT0 Register shortly after wake-up from Power-save may give an incorrect result. Since TCNT0 is clocked on the asynchronous TOSC clock, reading TCNT0 must be done through a register synchronized to the internal I/O clock domain. Synchronization takes place for every rising TOSC1 edge. When waking up from Power-save mode, and the I/O clock ($clk_{I/O}$) again becomes active, TCNT0 will read as the previous value (before entering sleep) until the next rising TOSC1 edge. The phase of the TOSC clock after waking up from Power-save mode is essentially unpredictable, as it depends on the wake-up time. The recommended procedure for reading TCNT0 is thus as follows:
  1. Write any value to either of the registers OCR0 or TCCR0.
  2. Wait for the corresponding Update Busy Flag to be cleared.
  3. Read TCNT0.

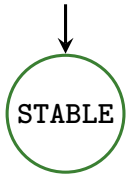
107

Datasheet, pp. 107

Rule

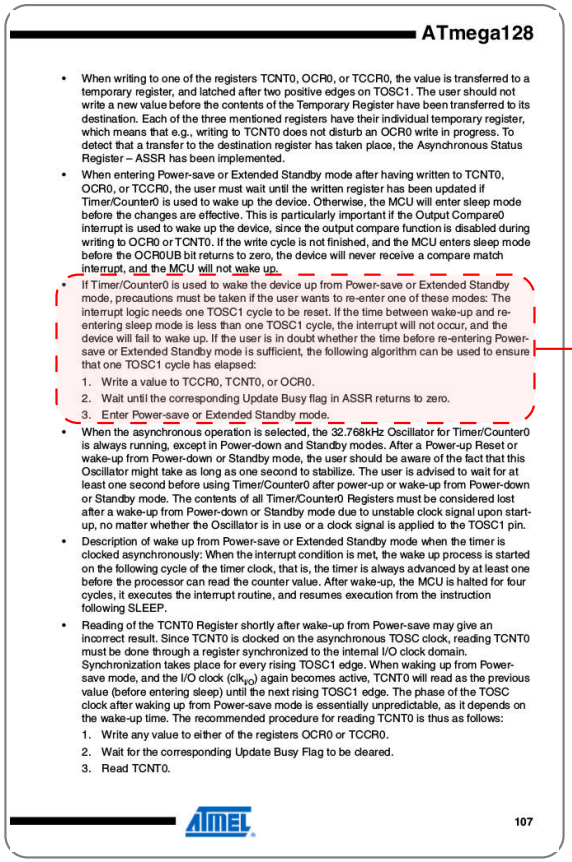
If Timer/Counter0 is used to wake the device up [...], precautions must be taken [...]

1. Write a value to `TCCR0`, `TCNT0`, or `OCR0`.
2. Wait until the corresponding Update Busy flag in `ASSR` returns to zero.
3. Enter Power-save or Extended Standby mode.



Specifications
ADP Example

ATmega128



ATmega128

- When writing to one of the registers TCNT0, OCR0, or TCCR0, the value is transferred to a temporary register, and latched after two positive edges on TOSC1. The user should not write a new value before the contents of the Temporary Register have been transferred to its destination. Each of the three mentioned registers have their individual temporary register, which means that e.g., writing to TCNT0 does not disturb an OCR0 write in progress. To detect that a transfer to the destination register has taken place, the Asynchronous Status Register – ASSR has been implemented.
- When entering Power-save or Extended Standby mode after having written to TCNT0, OCR0, or TCCR0, the user must wait until the written register has been updated if Timer/Counter0 is used to wake up the device. Otherwise, the MCU will enter sleep mode before the changes are effective. This is particularly important if the Output Compare0 interrupt is used to wake up the device, since the output compare function is disabled during writing to OCR0 or TCNT0. If the write cycle is not finished, and the MCU enters sleep mode before the OCR0UB bit returns to zero, the device will never receive a compare match interrupt, and the MCU will not wake up.
- If Timer/Counter0 is used to wake the device up from Power-save or Extended Standby mode, precautions must be taken if the user wants to re-enter one of these modes: The interrupt logic needs one TOSC1 cycle to be reset. If the time between wake-up and re-entering sleep mode is less than one TOSC1 cycle, the interrupt will not occur, and the device will fail to wake up. If the user is in doubt whether the time before re-entering Power-save or Extended Standby mode is sufficient, the following algorithm can be used to ensure that one TOSC1 cycle has elapsed:
  1. Write a value to TCCR0, TCNT0, or OCR0.
  2. Wait until the corresponding Update Busy flag in ASSR returns to zero.
  3. Enter Power-save or Extended Standby mode.
- When the asynchronous operation is selected, the 32.768kHz Oscillator for Timer/Counter0 is always running, except in Power-down and Standby modes. After a Power-up Reset or wake-up from Power-down or Standby mode, the user should be aware of the fact that this Oscillator might take as long as one second to stabilize. The user is advised to wait for at least one second before using Timer/Counter0 after power-up or wake-up from Power-down or Standby mode. The contents of all Timer/Counter0 Registers must be considered lost after a wake-up from Power-down or Standby mode due to unstable clock signal upon start-up, no matter whether the Oscillator is in use or a clock signal is applied to the TOSC1 pin.
- Description of wake up from Power-save or Extended Standby mode when the timer is clocked asynchronously: When the interrupt condition is met, the wake up process is started on the following cycle of the timer clock, that is, the timer is always advanced by at least one before the processor can read the counter value. After wake-up, the MCU is halted for four cycles, it executes the interrupt routine, and resumes execution from the instruction following SLEEP.
- Reading of the TCNT0 Register shortly after wake-up from Power-save may give an incorrect result. Since TCNT0 is clocked on the asynchronous TOSC clock, reading TCNT0 must be done through a register synchronized to the internal I/O clock domain. Synchronization takes place for every rising TOSC1 edge. When waking up from Power-save mode, and the I/O clock ($clk_{I/O}$) again becomes active, TCNT0 will read as the previous value (before entering sleep) until the next rising TOSC1 edge. The phase of the TOSC clock after waking up from Power-save mode is essentially unpredictable, as it depends on the wake-up time. The recommended procedure for reading TCNT0 is thus as follows:
  1. Write any value to either of the registers OCR0 or TCCR0.
  2. Wait for the corresponding Update Busy Flag to be cleared.
  3. Read TCNT0.

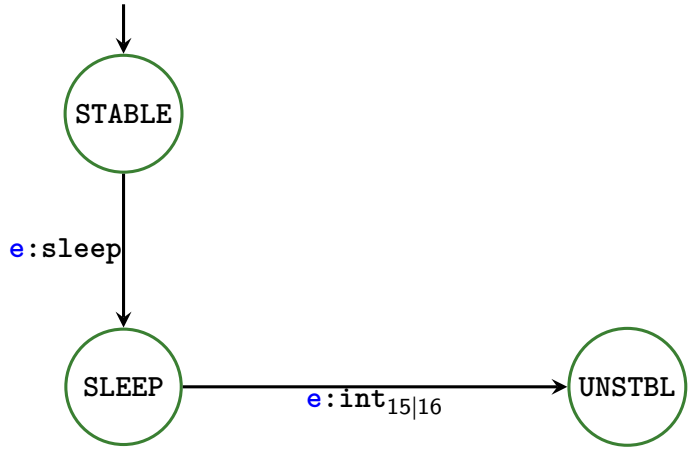
107

Datasheet, pp. 107

Rule

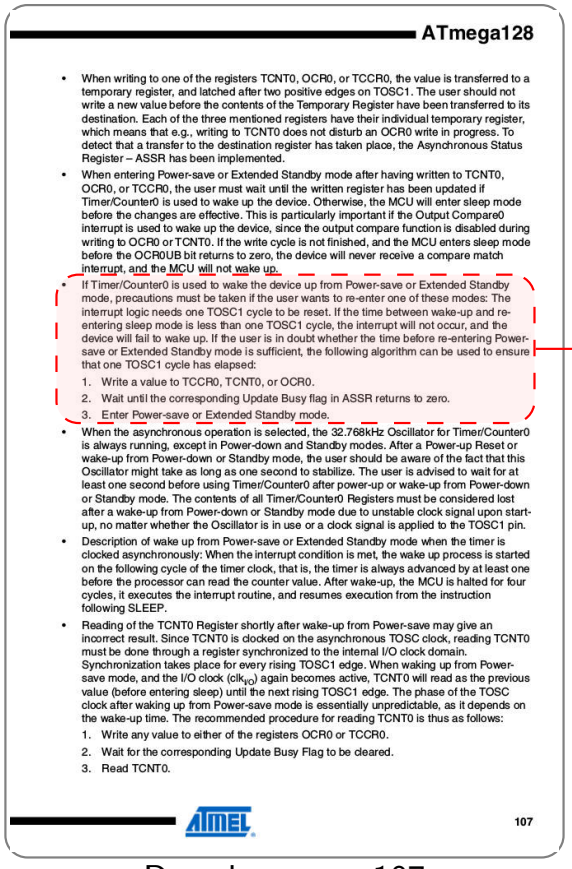
If Timer/Counter0 is used to wake the device up [...], precautions must be taken [...]

1. Write a value to `TCCR0`, `TCNT0`, or `OCR0`.
2. Wait until the corresponding Update Busy flag in `ASSR` returns to zero.
3. Enter Power-save or Extended Standby mode.



Specifications
ADP Example

ATmega128



ATmega128

- When writing to one of the registers TCNT0, OCR0, or TCCR0, the value is transferred to a temporary register, and latched after two positive edges on TOSC1. The user should not write a new value before the contents of the Temporary Register have been transferred to its destination. Each of the three mentioned registers have their individual temporary register, which means that e.g., writing to TCNT0 does not disturb an OCR0 write in progress. To detect that a transfer to the destination register has taken place, the Asynchronous Status Register – ASSR has been implemented.
- When entering Power-save or Extended Standby mode after having written to TCNT0, OCR0, or TCCR0, the user must wait until the written register has been updated if Timer/Counter0 is used to wake up the device. Otherwise, the MCU will enter sleep mode before the changes are effective. This is particularly important if the Output Compare0 interrupt is used to wake up the device, since the output compare function is disabled during writing to OCR0 or TCNT0. If the write cycle is not finished, and the MCU enters sleep mode before the OCR0UB bit returns to zero, the device will never receive a compare match interrupt, and the MCU will not wake up.
- If Timer/Counter0 is used to wake the device up from Power-save or Extended Standby mode, precautions must be taken if the user wants to re-enter one of these modes: The interrupt logic needs one TOSC1 cycle to be reset. If the time between wake-up and re-entering sleep mode is less than one TOSC1 cycle, the interrupt will not occur, and the device will fail to wake up. If the user is in doubt whether the time before re-entering Power-save or Extended Standby mode is sufficient, the following algorithm can be used to ensure that one TOSC1 cycle has elapsed:
  1. Write a value to TCCR0, TCNT0, or OCR0.
  2. Wait until the corresponding Update Busy flag in ASSR returns to zero.
  3. Enter Power-save or Extended Standby mode.
- When the asynchronous operation is selected, the 32.768kHz Oscillator for Timer/Counter0 is always running, except in Power-down and Standby modes. After a Power-up Reset or wake-up from Power-down or Standby mode, the user should be aware of the fact that this Oscillator might take as long as one second to stabilize. The user is advised to wait for at least one second before using Timer/Counter0 after power-up or wake-up from Power-down or Standby mode. The contents of all Timer/Counter0 Registers must be considered lost after a wake-up from Power-down or Standby mode due to unstable clock signal upon start-up, no matter whether the Oscillator is in use or a clock signal is applied to the TOSC1 pin.
- Description of wake up from Power-save or Extended Standby mode when the timer is clocked asynchronously: When the interrupt condition is met, the wake up process is started on the following cycle of the timer clock, that is, the timer is always advanced by at least one before the processor can read the counter value. After wake-up, the MCU is halted for four cycles, it executes the interrupt routine, and resumes execution from the instruction following SLEEP.
- Reading of the TCNT0 Register shortly after wake-up from Power-save may give an incorrect result. Since TCNT0 is clocked on the asynchronous TOSC clock, reading TCNT0 must be done through a register synchronized to the internal I/O clock domain. Synchronization takes place for every rising TOSC1 edge. When waking up from Power-save mode, and the I/O clock ($clk_{I/O}$) again becomes active, TCNT0 will read as the previous value (before entering sleep) until the next rising TOSC1 edge. The phase of the TOSC clock after waking up from Power-save mode is essentially unpredictable, as it depends on the wake-up time. The recommended procedure for reading TCNT0 is thus as follows:
  1. Write any value to either of the registers OCR0 or TCCR0.
  2. Wait for the corresponding Update Busy Flag to be cleared.
  3. Read TCNT0.

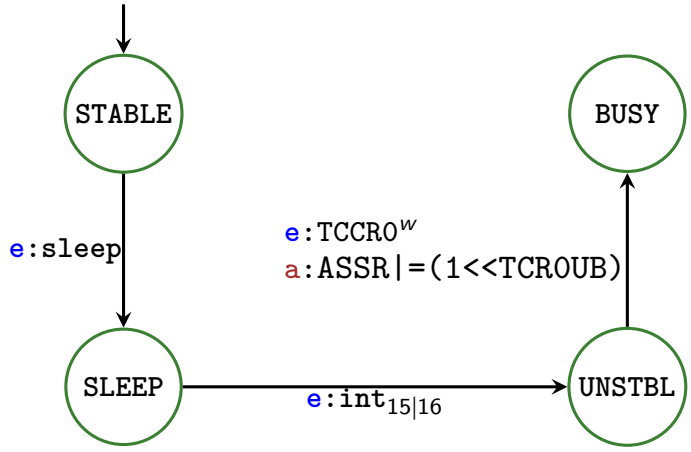
107

Datasheet, pp. 107

Rule

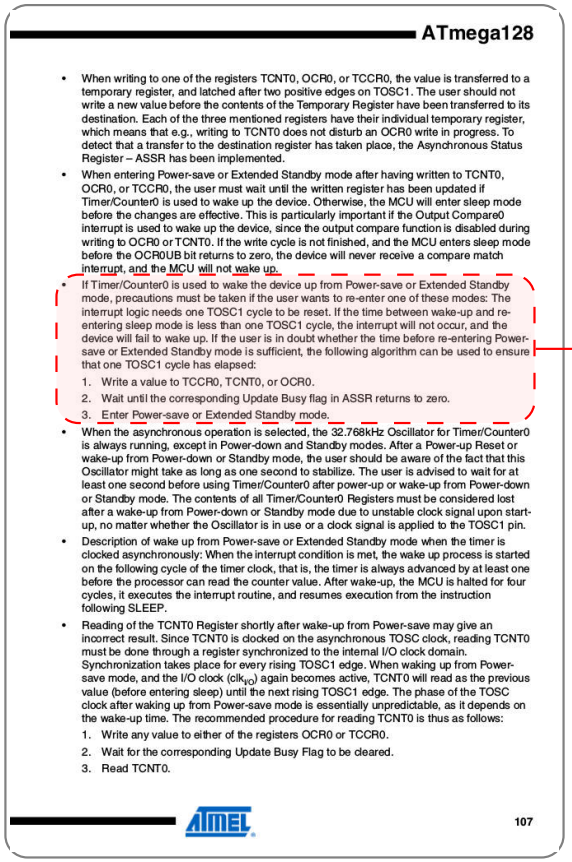
If Timer/Counter0 is used to wake the device up [...], precautions must be taken [...]

1. Write a value to `TCCR0`, `TCNT0`, or `OCR0`.
2. Wait until the corresponding Update Busy flag in `ASSR` returns to zero.
3. Enter Power-save or Extended Standby mode.



Specifications
ADP Example

ATmega128



ATmega128

- When writing to one of the registers TCNT0, OCR0, or TCCR0, the value is transferred to a temporary register, and latched after two positive edges on TOSC1. The user should not write a new value before the contents of the Temporary Register have been transferred to its destination. Each of the three mentioned registers have their individual temporary register, which means that e.g., writing to TCNT0 does not disturb an OCR0 write in progress. To detect that a transfer to the destination register has taken place, the Asynchronous Status Register – ASSR has been implemented.
- When entering Power-save or Extended Standby mode after having written to TCNT0, OCR0, or TCCR0, the user must wait until the written register has been updated if Timer/Counter0 is used to wake up the device. Otherwise, the MCU will enter sleep mode before the changes are effective. This is particularly important if the Output Compare0 interrupt is used to wake up the device, since the output compare function is disabled during writing to OCR0 or TCNT0. If the write cycle is not finished, and the MCU enters sleep mode before the OCR0UB bit returns to zero, the device will never receive a compare match interrupt, and the MCU will not wake up.
- If Timer/Counter0 is used to wake the device up from Power-save or Extended Standby mode, precautions must be taken if the user wants to re-enter one of these modes: The interrupt logic needs one TOSC1 cycle to be reset. If the time between wake-up and re-entering sleep mode is less than one TOSC1 cycle, the interrupt will not occur, and the device will fail to wake up. If the user is in doubt whether the time before re-entering Power-save or Extended Standby mode is sufficient, the following algorithm can be used to ensure that one TOSC1 cycle has elapsed:
  1. Write a value to TCCR0, TCNT0, or OCR0.
  2. Wait until the corresponding Update Busy flag in ASSR returns to zero.
  3. Enter Power-save or Extended Standby mode.
- When the asynchronous operation is selected, the 32.768kHz Oscillator for Timer/Counter0 is always running, except in Power-down and Standby modes. After a Power-up Reset or wake-up from Power-down or Standby mode, the user should be aware of the fact that this Oscillator might take as long as one second to stabilize. The user is advised to wait for at least one second before using Timer/Counter0 after power-up or wake-up from Power-down or Standby mode. The contents of all Timer/Counter0 Registers must be considered lost after a wake-up from Power-down or Standby mode due to unstable clock signal upon start-up, no matter whether the Oscillator is in use or a clock signal is applied to the TOSC1 pin.
- Description of wake up from Power-save or Extended Standby mode when the timer is clocked asynchronously: When the interrupt condition is met, the wake up process is started on the following cycle of the timer clock, that is, the timer is always advanced by at least one before the processor can read the counter value. After wake-up, the MCU is halted for four cycles, it executes the interrupt routine, and resumes execution from the instruction following SLEEP.
- Reading of the TCNT0 Register shortly after wake-up from Power-save may give an incorrect result. Since TCNT0 is clocked on the asynchronous TOSC clock, reading TCNT0 must be done through a register synchronized to the internal I/O clock domain. Synchronization takes place for every rising TOSC1 edge. When waking up from Power-save mode, and the I/O clock ($clk_{I/O}$) again becomes active, TCNT0 will read as the previous value (before entering sleep) until the next rising TOSC1 edge. The phase of the TOSC clock after waking up from Power-save mode is essentially unpredictable, as it depends on the wake-up time. The recommended procedure for reading TCNT0 is thus as follows:
  1. Write any value to either of the registers OCR0 or TCCR0.
  2. Wait for the corresponding Update Busy Flag to be cleared.
  3. Read TCNT0.

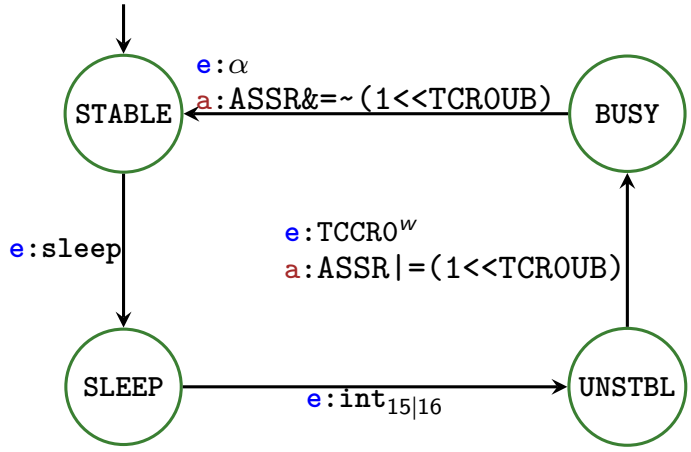
107

Datasheet, pp. 107

Rule

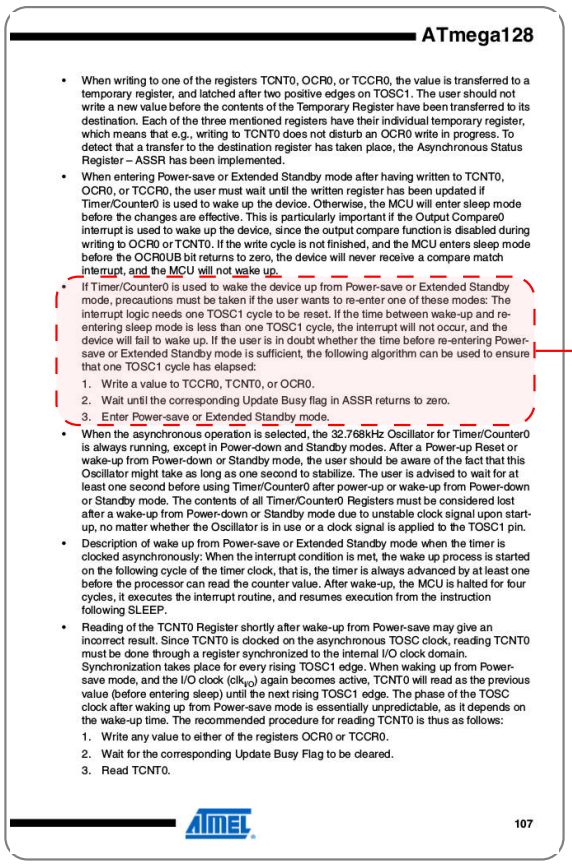
If Timer/Counter0 is used to wake the device up [...], precautions must be taken [...]

1. Write a value to `TCCR0`, `TCNT0`, or `OCR0`.
2. Wait until the corresponding Update Busy flag in `ASSR` returns to zero.
3. Enter Power-save or Extended Standby mode.



Specifications
ADP Example

ATmega128



ATmega128

- When writing to one of the registers TCNT0, OCR0, or TCCR0, the value is transferred to a temporary register, and latched after two positive edges on TOSC1. The user should not write a new value before the contents of the Temporary Register have been transferred to its destination. Each of the three mentioned registers have their individual temporary register, which means that e.g., writing to TCNT0 does not disturb an OCR0 write in progress. To detect that a transfer to the destination register has taken place, the Asynchronous Status Register – ASSR has been implemented.
- When entering Power-save or Extended Standby mode after having written to TCNT0, OCR0, or TCCR0, the user must wait until the written register has been updated if Timer/Counter0 is used to wake up the device. Otherwise, the MCU will enter sleep mode before the changes are effective. This is particularly important if the Output Compare0 interrupt is used to wake up the device, since the output compare function is disabled during writing to OCR0 or TCNT0. If the write cycle is not finished, and the MCU enters sleep mode before the OCR0UB bit returns to zero, the device will never receive a compare match interrupt, and the MCU will not wake up.
- If Timer/Counter0 is used to wake the device up from Power-save or Extended Standby mode, precautions must be taken if the user wants to re-enter one of these modes: The interrupt logic needs one TOSC1 cycle to be reset. If the time between wake-up and re-entering sleep mode is less than one TOSC1 cycle, the interrupt will not occur, and the device will fail to wake up. If the user is in doubt whether the time before re-entering Power-save or Extended Standby mode is sufficient, the following algorithm can be used to ensure that one TOSC1 cycle has elapsed:
  1. Write a value to TCCR0, TCNT0, or OCR0.
  2. Wait until the corresponding Update Busy flag in ASSR returns to zero.
  3. Enter Power-save or Extended Standby mode.
- When the asynchronous operation is selected, the 32.768kHz Oscillator for Timer/Counter0 is always running, except in Power-down and Standby modes. After a Power-up Reset or wake-up from Power-down or Standby mode, the user should be aware of the fact that this Oscillator might take as long as one second to stabilize. The user is advised to wait for at least one second before using Timer/Counter0 after power-up or wake-up from Power-down or Standby mode. The contents of all Timer/Counter0 Registers must be considered lost after a wake-up from Power-down or Standby mode due to unstable clock signal upon start-up, no matter whether the Oscillator is in use or a clock signal is applied to the TOSC1 pin.
- Description of wake up from Power-save or Extended Standby mode when the timer is clocked asynchronously: When the interrupt condition is met, the wake up process is started on the following cycle of the timer clock, that is, the timer is always advanced by at least one before the processor can read the counter value. After wake-up, the MCU is halted for four cycles, it executes the interrupt routine, and resumes execution from the instruction following SLEEP.
- Reading of the TCNT0 Register shortly after wake-up from Power-save may give an incorrect result. Since TCNT0 is clocked on the asynchronous TOSC clock, reading TCNT0 must be done through a register synchronized to the internal I/O clock domain. Synchronization takes place for every rising TOSC1 edge. When waking up from Power-save mode, and the I/O clock ($clk_{I/O}$) again becomes active, TCNT0 will read as the previous value (before entering sleep) until the next rising TOSC1 edge. The phase of the TOSC clock after waking up from Power-save mode is essentially unpredictable, as it depends on the wake-up time. The recommended procedure for reading TCNT0 is thus as follows:
  1. Write any value to either of the registers OCR0 or TCCR0.
  2. Wait for the corresponding Update Busy Flag to be cleared.
  3. Read TCNT0.

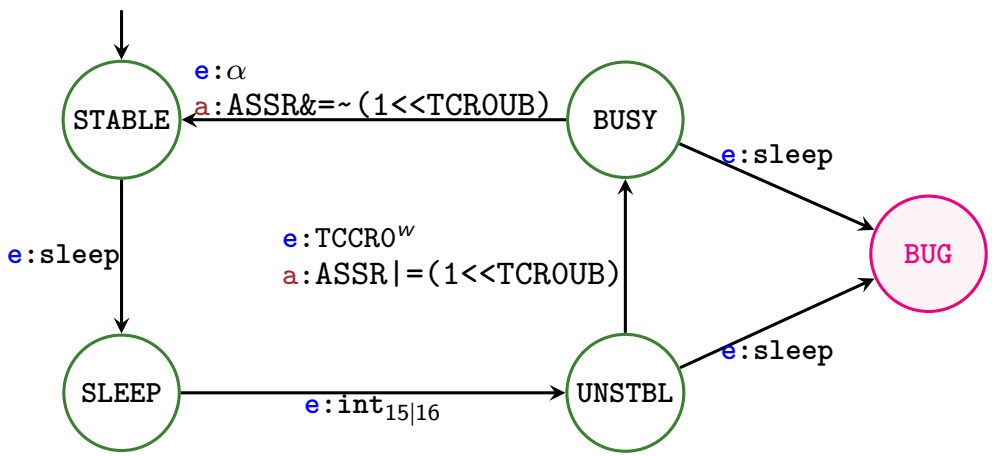
107

Datasheet, pp. 107

Rule

If Timer/Counter0 is used to wake the device up [...], precautions must be taken [...]

1. Write a value to `TCCR0`, `TCNT0`, or `OCR0`.
2. Wait until the corresponding Update Busy flag in `ASSR` returns to zero.
3. Enter Power-save or Extended Standby mode.



Part III

Abstractions

- TinyOS is an open source OS developped by Berkely.
- Mixture of preemptive and cooperative execution models.

1. Hardware interrupts can preempt execution at any time (if not masked).
2. Tasks are functions that are posted for being executed when system is idle.

- TinyOS is an open source OS developped by Berkely.
- Mixture of preemptive and cooperative execution models.

1. Hardware interrupts can preempt execution at any time (if not masked).
2. Tasks are functions that are posted for being executed when system is idle.

Concrete Environment

$$\mathcal{E} = \underbrace{\mathcal{M}}_{\text{Memory}} \times \overbrace{\mathcal{S}}^{\text{HW state}} \times \underbrace{\mathcal{Q}}_{\text{Tasks queue}} \times \overbrace{\mathcal{I}}^{\text{Interrupts}}$$

Hardware State Partitioning
Definition

- Hardware state is the primary information in the analysis.
- We should keep precise information about it.
- Memory content should be in relation with the hardware state because program infer the state by accessing hardware registers and/or some program variables.

Hardware State Partitioning
Definition

- Hardware state is the primary information in the analysis.
- We should keep precise information about it.
- Memory content should be in relation with the hardware state because program infer the state by accessing hardware registers and/or some program variables.

Hardware State Partitioning $\mathcal{D}^\sharp_S$

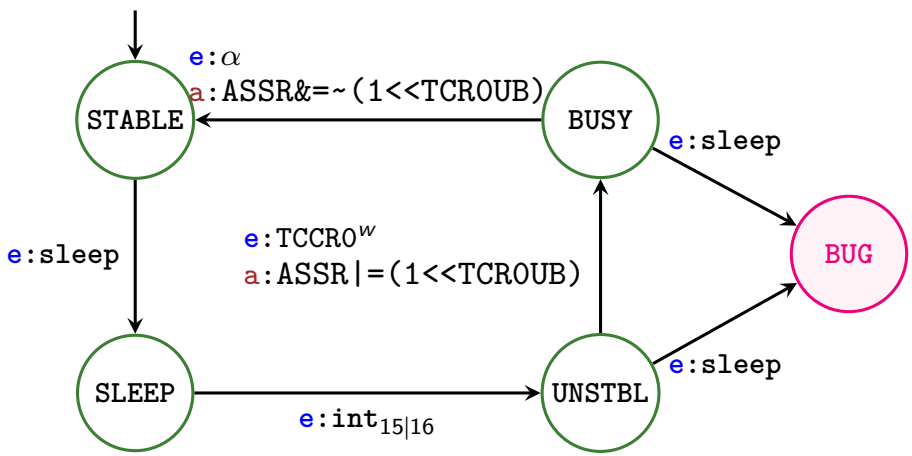
1. First, we partition the environments w.r.t. automaton states $\mathcal{S}$.
2. Then, we use a numerical abstract domain $\langle \mathcal{D}^\sharp_{\mathcal{M}}, \sqsubseteq_{\mathcal{M}} \rangle$ to abstract the values of registers and variables.

$$\langle \wp(\mathcal{E}), \subseteq \rangle \leftrightarrows \langle \mathcal{S} \to \mathcal{D}^\sharp_{\mathcal{M}}, \dot{\sqsubseteq}_{\mathcal{M}} \rangle$$

Hardware State Partitioning
Example

```
1  void main() {
2    //Config timer
3    ...
4    //Wait for interrupt
5    while(1) {
6      asm volatile("sleep"::);
7      ...
8    }
9  }
```

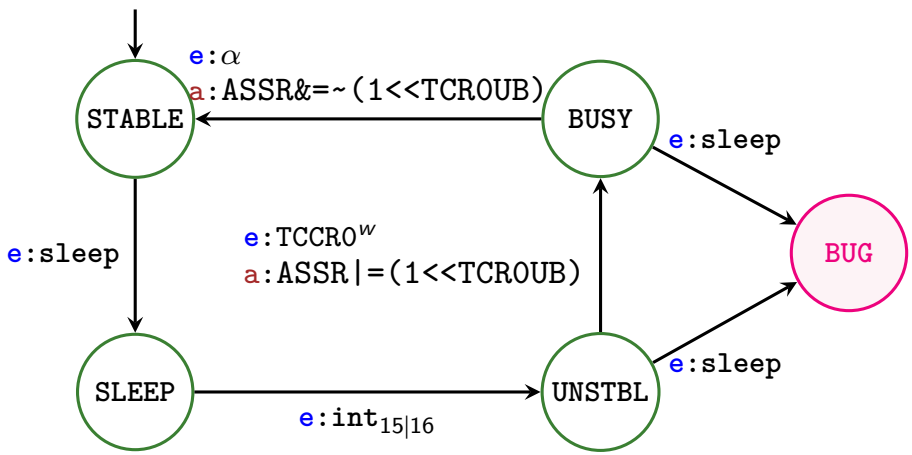
```
10 ISR(TIMER0_OVF_vect) {
11   ...
12   //Stabilize the timer
13   TCCR0 = TCCR0;
14   while
15    (ASSR & 1 << TCR0UB);
16    //Continue work
17    ...
18 }
```



Hardware State Partitioning
Example

```
1  void main() {
2    //Config timer
3    ...
4    //Wait for interrupt
5    while(1) {
6      asm volatile("sleep"::);
7      ...
8    }
9  }
```

```
10 ISR(TIMER0_OVF_vect) {
11   ...
12   //Stabilize the timer
13   TCCR0 = TCCR0;
14   while
15    (ASSR & 1 << TCR0UB);
16    //Continue work
17    ...
18 }
```



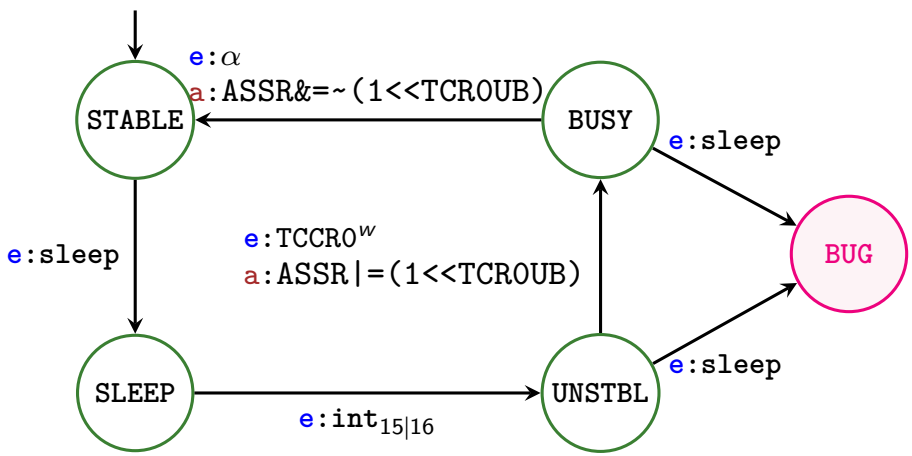
X_6 X_{11} X_{14} X_{17}

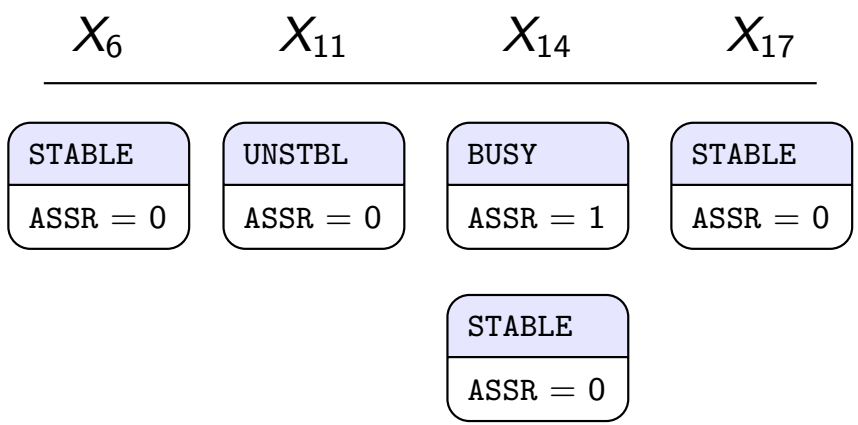
Hardware State Partitioning

Example

```
1  void main() {
2    //Config timer
3    ...
4    //Wait for interrupt
5    while(1) {
6      asm volatile("sleep"::);
7      ...
8    }
9  }
```

```
10 ISR(TIMER0_OVF_vect) {
11   ...
12   //Stabilize the timer
13   TCCR0 = TCCR0;
14   while
15    (ASSR & 1 << TCR0UB);
16    //Continue work
17    ...
18 }
```





Tasks Queue Partitioning
Motivation

Example: SPI serial transfer with a task polling (instead of an active polling).

Tasks Queue Partitioning
Motivation

Example: SPI serial transfer with a task polling (instead of an active polling).

```
task void tx() {
  if (i < m_len) {
    SPDR = m_data[i];
    post check();
    return;
  }
  post end();
}
```

```
task void check() {
  if !(SPSR&(1<<SPIF))
    post check();
  else {
    m_answer[i] = SPDR;
    i++;
    post tx();
  }
}
```

```
task void end()
  SPCR
   &=~(1<<SPE);
  ...
}
```

Tasks Queue Partitioning
Motivation

Example: SPI serial transfer with a task polling (instead of an active polling).

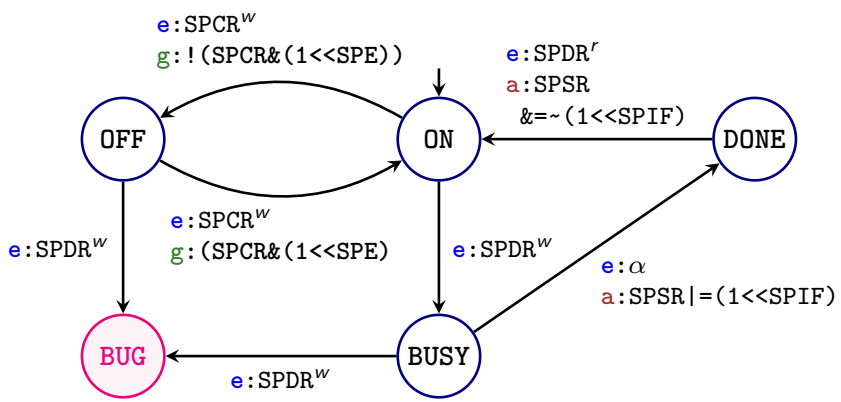
```
task void tx() {
  if (i < m_len) {
    SPDR = m_data[i];
    post check();
    return;
  }
  post end();
}
```

```
task void check() {
  if !(SPSR&(1<<SPIF))
    post check();
  else {
    m_answer[i] = SPDR;
    i++;
    post tx();
  }
}
```

```
task void end()
  SPCR
   &=~(1<<SPE);
  ...
}
```

Rule

Data register SPDR can not be used when the bus is not free.



Tasks Queue Partitioning
Definition

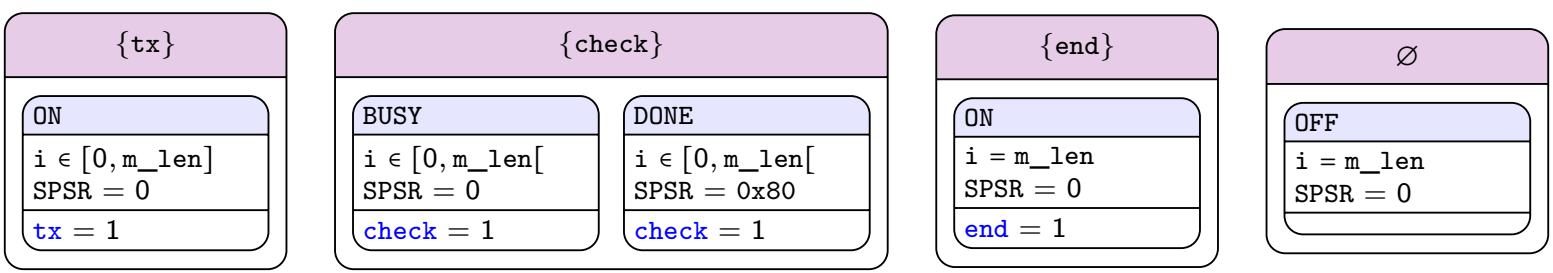
Tasks control flow problem

We can not verify this ADP with $\mathcal{D}^{\sharp}_{\mathcal{S}}$ since we consider every possible order of tasks.

Solution

- We distinguish between the case when a task is posted or not.
- We count the number of occurances of posted tasks.

$$\langle \wp(\mathcal{E}), \subseteq \rangle \leftrightarrows \langle \wp(\mathbb{T}) \to (\mathcal{D}^{\sharp}_{\mathcal{S}} \times \mathcal{N}^{\sharp}_{\mathbb{T}\to\mathbb{N}}), \sqsubseteq_{\mathcal{Q}} \rangle$$



Abstraction of Preemption
Overview

Modular Preemption Analysis

- Avoid re-analyzing interrupts each time they are enabled.

```
task void t() {
  SPCR |= (1<<SPE);
  ...
  x = y * i;
}
```

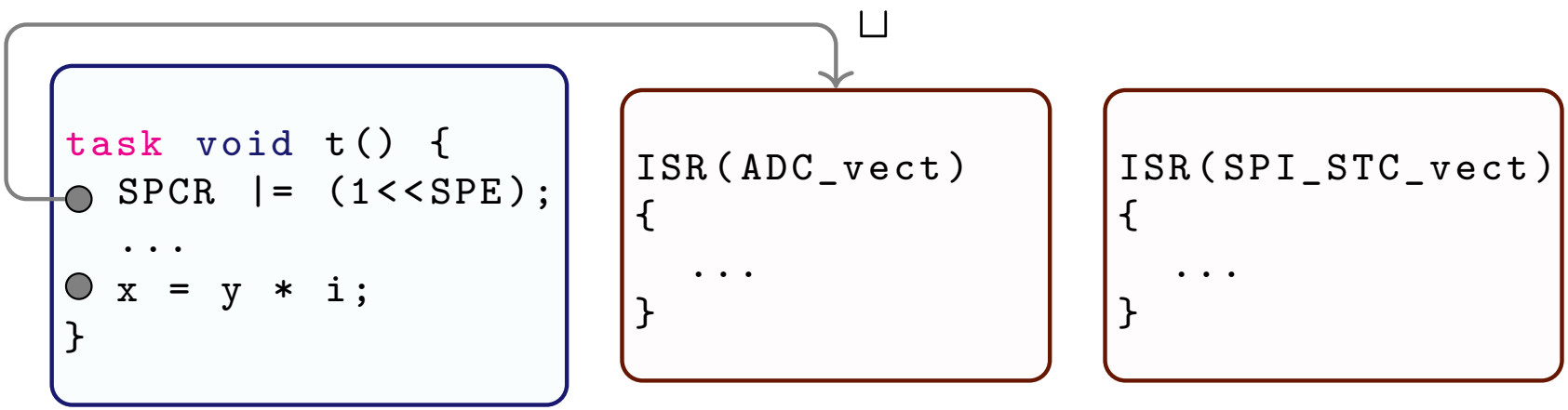
```
ISR(ADC_vect)
{
  ...
}
```

```
ISR(SPI_STC_vect)
{
  ...
}
```

Abstraction of Preemption
Overview

Modular Preemption Analysis

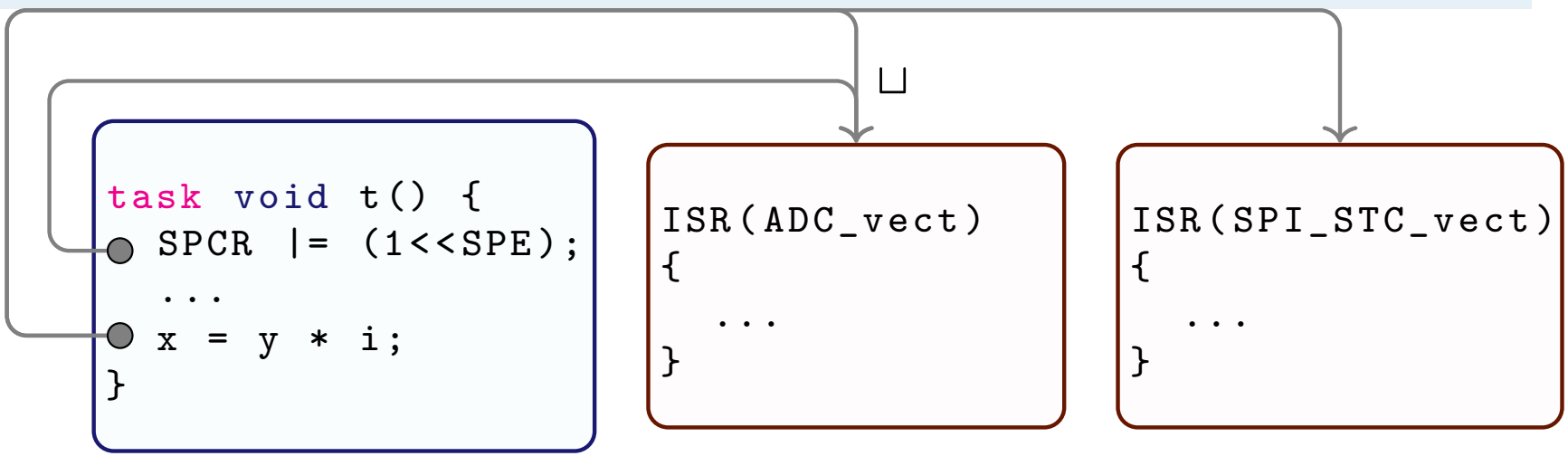
- Avoid re-analyzing interrupts each time they are enabled.
- Collect the preemption environments at possible fire sites and analyze interrupts independently.



Abstraction of Preemption
Overview

Modular Preemption Analysis

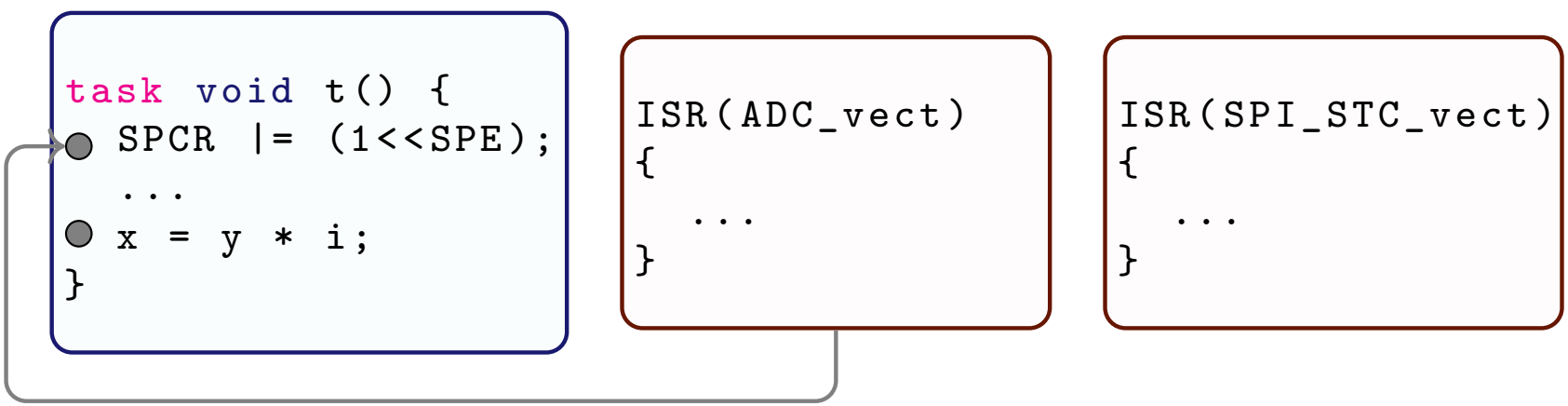
- Avoid re-analyzing interrupts each time they are enabled.
- Collect the preemption environments at possible fire sites and analyze interrupts independently.



Abstraction of Preemption
Overview

Modular Preemption Analysis

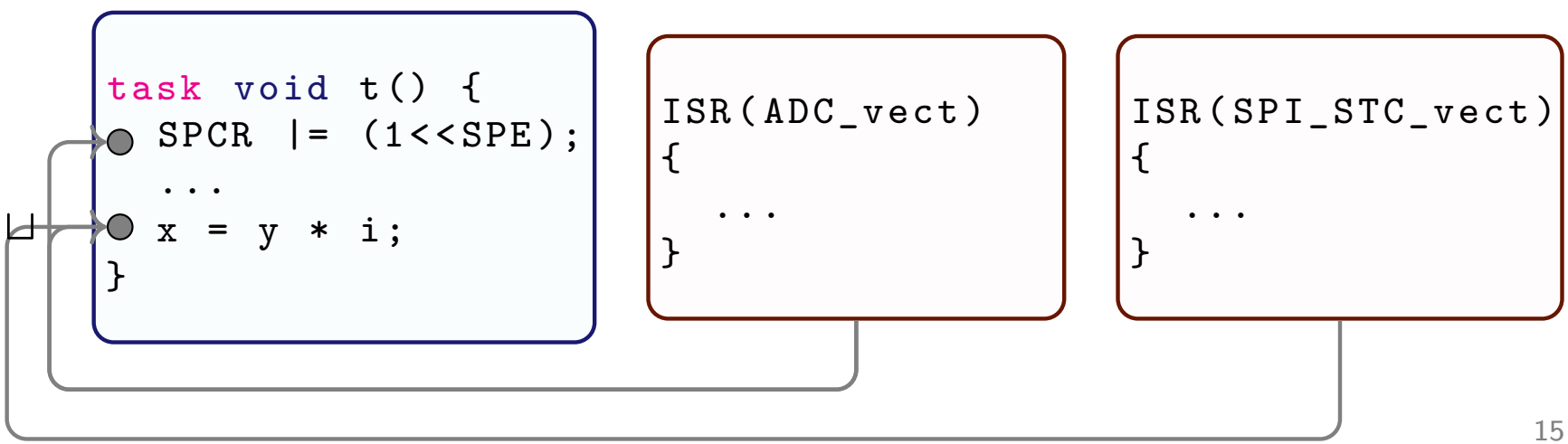
- Avoid re-analyzing interrupts each time they are enabled.
- Collect the preemption environments at possible fire sites and analyze interrupts independently.
- The result environments are injected at fire sites to approximate the effect of interrupts.



Abstraction of Preemption
Overview

Modular Preemption Analysis

- Avoid re-analyzing interrupts each time they are enabled.
- Collect the preemption environments at possible fire sites and analyze interrupts independently.
- The result environments are injected at fire sites to approximate the effect of interrupts.



Abstraction of Preemption
Definition

Abstraction of Interrupts Masks

$$\mathcal{D}^\sharp_{\mathcal{K}} \triangleq \underbrace{\{\bot_{\mathbb{01}}, \mathbb{0}, \mathbb{1}, \top_{\mathbb{01}}\}}_{\mathcal{K}^\sharp_{\mathcal{G}}} \times \underbrace{(\wp(\mathbb{I}) \to \mathcal{D}^\sharp_{\mathcal{Q}})}_{\mathcal{K}^\sharp_{\mathcal{P}}}$$

- $\mathcal{K}^\sharp_{\mathcal{G}}$ is an abstraction of the global interrupt flag. It is modified by statements like `asm volatile("sei":::"memory")`.
- $\mathcal{K}^\sharp_{\mathcal{P}}$ is a partitioning of environments w.r.t. active interrupts. It is affected by modification of specific hardware registers.

Abstraction of Preemption
Definition

Abstraction of Interrupts Masks

$$\mathcal{D}^\sharp_{\mathcal{K}} \triangleq \underbrace{\{\bot_{\mathbb{01}}, \mathbb{0}, \mathbb{1}, \top_{\mathbb{01}}\}}_{\mathcal{K}^\sharp_{\mathcal{G}}} \times \underbrace{(\wp(\mathbb{I}) \to \mathcal{D}^\sharp_{\mathcal{Q}})}_{\mathcal{K}^\sharp_{\mathcal{P}}}$$

- $\mathcal{K}^\sharp_{\mathcal{G}}$ is an abstraction of the global interrupt flag. It is modified by statements like `asm volatile("sei":::"memory")`.
- $\mathcal{K}^\sharp_{\mathcal{P}}$ is a partitioning of environments w.r.t. active interrupts. It is affected by modification of specific hardware registers.

Abstraction of Interruption Environments

$$\mathcal{D}^\sharp_{\mathcal{I}} \triangleq \mathcal{D}^\sharp_{\mathcal{K}} \times \underbrace{(\mathbb{I} \to \mathcal{D}^\sharp_{\mathcal{K}})}_{\text{Preemption}} \times \underbrace{(\mathbb{I} \to \mathcal{D}^\sharp_{\mathcal{K}})}_{\text{Return}}$$

Experiments
Implementation

- Prototype implementation called SADA (*Static Analysis with Device Abstraction*).
- Done in OCaml ($\sim$ 4000 lines of code).
- Using CIL and Apron.
- Seven ADPs were analyzed related to three devices:
  1. Wireless transceiver CC2420.
  2. Analog switch ADG715.
  3. ATmega128 timer.
- Two driver implementations for each device: TinyOS 1.x and 2.x.
  - Very different design and algorithms.
  - Same property can be used.

Experiments
Results

Time (s) × Memory (MB) × {✔, ✘, ⊗}

Driver	LoC	# ISR	$\lvert\mathbb{T}\rvert$	ADP	$\lvert\mathcal{S}\rvert$	$\mathcal{D}^{\sharp}_{\mathcal{I}}(\mathcal{D}^{\sharp}_{\mathcal{S}})$			$\mathcal{D}^{\sharp}_{\mathcal{I}}(\mathcal{D}^{\sharp}_{\mathcal{Q}})$		
CC2420 1.x	2666	1	1	$\mathcal{A}_{\texttt{SPI-SS}}$	4	12	12	✔	850	42	✔
				$\mathcal{A}_{\texttt{SPI-TX}}$	10	21	13	✔	1600	74	✔
CC2420 2.x	10133	2	10	$\mathcal{A}_{\texttt{SPI-SS}}$	4		∞		34	18	✔
				$\mathcal{A}_{\texttt{SPI-TX}}$	10		∞		39	18	✔
ADG715 1.x	2038	1	1	$\mathcal{A}_{\texttt{PULL-UP}}$	4	1	11	✘	1	11	✘
				$\mathcal{A}_{\texttt{TWI-TX}}$	6	4	11	✔	7	12	✔
ADG715 2.x	4412	1	6	$\mathcal{A}_{\texttt{PULL-UP}}$	4	23	14	✔	8	13	✔
				$\mathcal{A}_{\texttt{TWI-TX}}$	6	40	16	✘	6	14	✘
Timer 1.x	1627	1	3	$\mathcal{A}_{\texttt{STBL}}$	7	6	11	⊗	39	16	⊗
				$\mathcal{A}_{\texttt{OCRO}}$	4	3	10	⊗	29	15	⊗
				$\mathcal{A}_{\texttt{TCCRO}}$	4	3	10	✔	37	16	✔
Timer 2.x	2384	2	2	$\mathcal{A}_{\texttt{STBL}}$	4	7	12	✔	26	13	✔
				$\mathcal{A}_{\texttt{OCRO}}$	4	10	12	✔	38	13	✔
				$\mathcal{A}_{\texttt{TCCRO}}$	4	10	11	⊗	23	13	⊗

Experiments
Results

Time (s) × Memory (MB) × {✔, ✘, ⊗}

Driver	LoC	# ISR	$\lvert\mathbb{T}\rvert$	ADP	$\lvert\mathcal{S}\rvert$	$\mathcal{D}^{\sharp}_{\mathcal{I}}(\mathcal{D}^{\sharp}_{\mathcal{S}})$			$\mathcal{D}^{\sharp}_{\mathcal{I}}(\mathcal{D}^{\sharp}_{\mathcal{Q}})$		
CC2420 1.x	2666	1	1	$\mathcal{A}_{\texttt{SPI-SS}}$	4	12	12	✔	850	42	✔
				$\mathcal{A}_{\texttt{SPI-TX}}$	10	21	13	✔	1600	74	✔
CC2420 2.x	10133	2	10	$\mathcal{A}_{\texttt{SPI-SS}}$	4		∞		34	18	✔
				$\mathcal{A}_{\texttt{SPI-TX}}$	10		∞		39	18	✔
ADG715 1.x	2038	1	1	$\mathcal{A}_{\texttt{PULL-UP}}$	4	1	11	✘	1	11	✘
				$\mathcal{A}_{\texttt{TWI-TX}}$	6	4	11	✔	7	12	✔
ADG715 2.x	4412	1	6	$\mathcal{A}_{\texttt{PULL-UP}}$	4	23	14	✔	8	13	✔
				$\mathcal{A}_{\texttt{TWI-TX}}$	6	40	16	✘	6	14	✘
Timer 1.x	1627	1	3	$\mathcal{A}_{\texttt{STBL}}$	7	6	11	⊗	39	16	⊗
				$\mathcal{A}_{\texttt{OCRO}}$	4	3	10	⊗	29	15	⊗
				$\mathcal{A}_{\texttt{TCCRO}}$	4	3	10	✔	37	16	✔
Timer 2.x	2384	2	2	$\mathcal{A}_{\texttt{STBL}}$	4	7	12	✔	26	13	✔
				$\mathcal{A}_{\texttt{OCRO}}$	4	10	12	✔	38	13	✔
				$\mathcal{A}_{\texttt{TCCRO}}$	4	10	11	⊗	23	13	⊗

Experiments
Results

Time (s) × Memory (MB) × {✔, ✘, ⊗}

Driver	LoC	# ISR	$\lvert\mathbb{T}\rvert$	ADP	$\lvert\mathcal{S}\rvert$	$\mathcal{D}^{\sharp}_{\mathcal{I}}(\mathcal{D}^{\sharp}_{\mathcal{S}})$			$\mathcal{D}^{\sharp}_{\mathcal{I}}(\mathcal{D}^{\sharp}_{\mathcal{Q}})$		
CC2420 1.x	2666	1	1	$\mathcal{A}_{\texttt{SPI-SS}}$	4	12	12	✔	850	42	✔
				$\mathcal{A}_{\texttt{SPI-TX}}$	10	21	13	✔	1600	74	✔
CC2420 2.x	10133	2	10	$\mathcal{A}_{\texttt{SPI-SS}}$	4		∞		34	18	✔
				$\mathcal{A}_{\texttt{SPI-TX}}$	10		∞		39	18	✔
ADG715 1.x	2038	1	1	$\mathcal{A}_{\texttt{PULL-UP}}$	4	1	11	✘	1	11	✘
				$\mathcal{A}_{\texttt{TWI-TX}}$	6	4	11	✔	7	12	✔
ADG715 2.x	4412	1	6	$\mathcal{A}_{\texttt{PULL-UP}}$	4	23	14	✔	8	13	✔
				$\mathcal{A}_{\texttt{TWI-TX}}$	6	40	16	✘	6	14	✘
Timer 1.x	1627	1	3	$\mathcal{A}_{\texttt{STBL}}$	7	6	11	⊗	39	16	⊗
				$\mathcal{A}_{\texttt{OCRO}}$	4	3	10	⊗	29	15	⊗
				$\mathcal{A}_{\texttt{TCCRO}}$	4	3	10	✔	37	16	✔
Timer 2.x	2384	2	2	$\mathcal{A}_{\texttt{STBL}}$	4	7	12	✔	26	13	✔
				$\mathcal{A}_{\texttt{OCRO}}$	4	10	12	✔	38	13	✔
				$\mathcal{A}_{\texttt{TCCRO}}$	4	10	11	⊗	23	13	⊗

Experiments
Results

Time (s) × Memory (MB) × {✔, ✘, ⊗}

Driver	LoC	# ISR	$\lvert\mathbb{T}\rvert$	ADP	$\lvert\mathcal{S}\rvert$	$\mathcal{D}^{\sharp}_{\mathcal{I}}(\mathcal{D}^{\sharp}_{\mathcal{S}})$			$\mathcal{D}^{\sharp}_{\mathcal{I}}(\mathcal{D}^{\sharp}_{\mathcal{Q}})$		
CC2420 1.x	2666	1	1	$\mathcal{A}_{\texttt{SPI-SS}}$	4	12	12	✔	850	42	✔
				$\mathcal{A}_{\texttt{SPI-TX}}$	10	21	13	✔	1600	74	✔
CC2420 2.x	10133	2	10	$\mathcal{A}_{\texttt{SPI-SS}}$	4		∞		34	18	✔
				$\mathcal{A}_{\texttt{SPI-TX}}$	10		∞		39	18	✔
ADG715 1.x	2038	1	1	$\mathcal{A}_{\texttt{PULL-UP}}$	4	1	11	✘	1	11	✘
				$\mathcal{A}_{\texttt{TWI-TX}}$	6	4	11	✔	7	12	✔
ADG715 2.x	4412	1	6	$\mathcal{A}_{\texttt{PULL-UP}}$	4	23	14	✔	8	13	✔
				$\mathcal{A}_{\texttt{TWI-TX}}$	6	40	16	✘	6	14	✘
Timer 1.x	1627	1	3	$\mathcal{A}_{\texttt{STBL}}$	7	6	11	⊗	39	16	⊗
				$\mathcal{A}_{\texttt{OCRO}}$	4	3	10	⊗	29	15	⊗
				$\mathcal{A}_{\texttt{TCCRO}}$	4	3	10	✔	37	16	✔
Timer 2.x	2384	2	2	$\mathcal{A}_{\texttt{STBL}}$	4	7	12	✔	26	13	✔
				$\mathcal{A}_{\texttt{OCRO}}$	4	10	12	✔	38	13	✔
				$\mathcal{A}_{\texttt{TCCRO}}$	4	10	11	⊗	23	13	⊗

Related Work

Approach	Tool	Properties	Preemption
Dynamic analysis	SafeTinyOS	Runtime	Enumeration
	TemporalMonitors	Runtime/LTL	
Model checking	TOS2CProver	Runtime	Sequenlization
	i-CBMC	Runtime	Partial order encoding
Deductive methods	[Duan & Regeher]	Hardware	Not supported
Abstract interpretation	[Monniaux]	Hardware	Sequenlization
	MCSquare	Runtime	
	[Wu et al.]	Runtime	

Conclusion

Summary

- An abstract interpreter for verifying hardware interactions in TinyOS device drivers was presented.
- It is based on different levels of partitioning for handling hardware state, tasks queue and interrupt masks.
- Preemption is formalized using an iterative and modular fixpoint analysis.

Future Work

- Develop abstract domains for timing specifications of devices.
- Consider other cooperative models (e.g. Protothreads of Contiki).