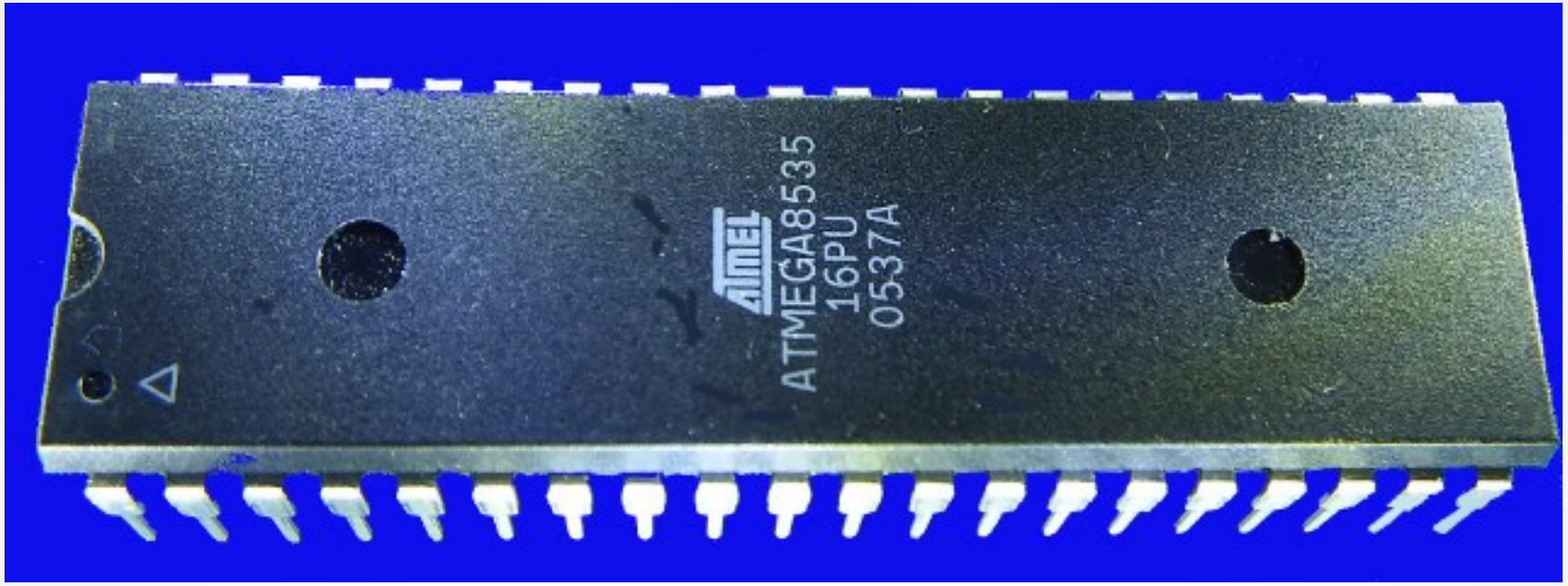


Lecture 7: Interrupts

Hardware, Internals and Programming of AVR Microcontrollers in Assembler

by
Gerhard Schmidt
Kastanienallee 20
D-64289 Darmstadt

Interrupts

- When the timer/counter overflows or reaches compare match A or B it would be a good idea to have a mechanism that interrupts normal program execution, so the controller can react on this event and do the necessary steps.
- After doing these steps the controller shall continue to work as if nothing has happened in between.
- This mechanism is implemented in all AVRs. It requires the following:
  1. The stack has to store the execution address when interrupted.
  2. The controller has to have a place where it jumps to when interrupted.
  3. Further interrupts have to be blocked and their execution has to be delayed, but not simply forgotten.
  4. As the AVRs have only one status register, its content has to be saved and restored before ending the interrupt service routine.

Timer interrupts

- To enable the timer's interrupts goes as follows.
- The timers have an interrupt mask port register, here the one for TC0:

Bit	7	6	5	4	3	2	1	0	
0x39 (0x59)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

You can enable the interrupt by setting the appropriate bits high:

```
; Timer overflow interrupt enable
      SBI TIMSK0, TOIE0 ; The overflow interrupt is enabled
      ; Alternative formulation:
      LDI R16, 1<<TOIE0 ; Set the TOIE0 bit
      OUT TIMSK0, R16 ; Write this to interrupt mask TC0
```

- When the timer overflows, it sets its respective flag TOV0 in port register TIFR0:

Bit	7	6	5	4	3	2	1	0	
0x38 (0x58)	–	–	–	–	–	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Interrupt enable and stack

- **The status register has a flag called I that enables reaction to interrupt flags:**

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **To enable this set I to one:**

```
; Enable general interrupt flag
    SEI ; Set I bit in SREG
    ; CLI ; The opposite: clear the I bit and stop interrupt execution
```

- **As the stack is needed to store the return address, stack has to be initiated before interrupts can be allowed:**

```
; Initiate the stack
    ; LDI R16, HIGH(RAMEND) ; In larger devices with more than 256 byte SRAM
    ; OUT SPH, R16 ; Set MSB stackpointer
    LDI R16,LOW(RAMEND) ; In all devices
    OUT SPL, R16 ; Set LSB stackpointer
```

Interrupt vectors

- **The places where all interrupts jump to are the interrupt vectors.**
- **ATtiny24 has 17 different vectors, others have different ones.**
- **Int vectors are located from address 0000 on.**

Table 9-1. Reset and Interrupt Vectors

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset
2	0x0001	INT0	External Interrupt Request 0
3	0x0002	PCINT0	Pin Change Interrupt Request 0
4	0x0003	PCINT1	Pin Change Interrupt Request 1
5	0x0004	WDT	Watchdog Time-out
6	0x0005	TIM1_CAPT	Timer/Counter1 Capture Event
7	0x0006	TIM1_COMPA	Timer/Counter1 Compare Match A
8	0x0007	TIM1_COMPB	Timer/Counter1 Compare Match B
9	0x0008	TIM1_OVF	Timer/Counter1 Overflow
10	0x0009	TIM0_COMPA	Timer/Counter0 Compare Match A
11	0x000A	TIM0_COMPB	Timer/Counter0 Compare Match B
12	0x000B	TIM0_OVF	Timer/Counter0 Overflow
13	0x000C	ANA_COMP	Analog Comparator
14	0x000D	ADC	ADC Conversion Complete
15	0x000E	EE_RDY	EEPROM Ready
16	0x000F	USI_STR	USI START
17	0x0010	USI_OVF	USI Overflow

Filling the int vector jump table

```
; The interrupt vector jump table of an ATtiny24
.CSEG ; Beginning of the code segment
.ORG 0x0000 ; Starts at address 0
      RJMP Main ; On reset jump to the init routines
      RETI ; INT0 interrupt, not used
      RETI ; PCINT0 interrupt, not used
      RETI ; PCINT1 interrupt, not used
      RETI ; WDT interrupt, not used
      RETI ; TC1-CAPT interrupt, not used
      RETI ; TC1-COMP-A interrupt, not used
      RETI ; TC1-COMP-B interrupt, not used
      RETI ; TC1-OVF interrupt, not used
      RETI ; TC0-COMP-A interrupt, not used
      RETI ; TC0-COMP-B interrupt, not used
      RJMP TC0ISR ; TC0-OVF interrupt, used
      RETI ; ANACOMP interrupt, not used
      RETI ; ADC interrupt, not used
      RETI ; EERDY interrupt, not used
      RETI ; USI-STR interrupt, not used
      RETI ; USI-OVF interrupt, not used
```

- To ensure that these vectors are really placed at address 0000, the two directives .CSEG (for code segment) and .ORG (for origin) ensure that no code is placed before the vectors.
- The vector consists of RJMP instructions and RETI instructions.
- The RJMP Main is executed on any hardware reset.
- The TC0 overflow interrupt jumps to a interrupt service routine.
- All other vectors are inactive and return from the interrupt (RETI).

The interrupt service routine

- The place where the TC0-OVF interrupt jumps to has to be written to a location outside the vectors, e. g. like this:

```
; The interrupt service routine of the TC0-OVF
Tc0Isr:
        SBI PINB, PINB2 ; Toggle the PB2 pin
        RETI ; Return from interrupt
```

- The SBI on the input port register PINB toggles the pin PB2.
- The RETI instruction does two things:
  - It sets the I flag in the SREG to one again. I was cleared when the interrupt began to block further interrupts.
  - It returns to the program address where the interrupt occurred, which is taken from the stack.
- It is recommendable to fill all unused interrupt vectors with RETI to have all interrupts finalized in a controlled manner.

Priority rules

- **If two or more interrupt conditions occur at exactly the same time:**

1. **The interrupt scheme is prioritized. The higher the respective interrupt is listed in the jump vector list, the earlier is it executed. The highest priority always have the external interrupts.**
2. **If an interrupt vector is executed, its interrupt flag is automatically cleared.**
3. **If an interrupt is not yet executed because a higher-ranked interrupt is executed first, its interrupt flag still remains set.**
4. **Immediately after the higher-ranked interrupt is terminated with RETI, the pending lower-ranked interrupt is executed.**
5. **To avoid interrupt overruns (a set interrupt flag is set again) interrupt-service-routines shall always be short enough to avoid such overruns of lower-ranked interrupts.**

The status register during interrupts

- The status register is only once implemented. If the interrupt service routine uses instructions that alter SREG flags, their initial state has to be restored.
- The SREG can be stored in a register (here R15).

```
.def rSreg = R15 ; Place to store SREG during interrupts
; Interrupt service routine TC0 overflow
Tc0Isr:
        IN rSreg, SREG ; Store the status register‘s content
        DEC R16 ; Decrease R16 (instruction changes flags)
        BRNE Tc0IsrRet
        SBI PINB,PINB2 ; Toggle PB2
Tc0IsrRet:
        OUT SREG, rSreg ; Restore previous content of SREG
        RETI ; Return from interrupt
```

- As interrupts occur asynchroneous to the normal program flow, it has to be ensured that SREG flags are not changed uncontrolled by any interrupt service routine, otherwise unpredicable conditions can result.

Advantages of interrupts

- **The advantages of using interrupts for programming controllers are:**
  - **Interrupts are very fast, they react within 7 clock cycles (a few micro-seconds) on events (e.g. on external interrupts).**
  - **Lots of different tasks can be programmed to run simultaneously and without delaying other time-critical tasks.**
  - **Interrupts do not waste time and program code, even not for complex timing tasks, the CPU can do other tasks in that time.**
  - **Interrupts save operating current: the controller can go to sleep in idle mode, and only wakes up if needed (search for SLEEP in the handbook).**
- **Interrupt Service Routines have to be short (never use timing loops inside such a routine). If you need longer actions, use flags.**

Timer in PWM modes

- **Beside normal-mode and CTC-mode all timer in AVRs have also PWM modes.**
- **In PWM modes the timer counts up (in fast PWM mode) or up and then down (in phase-correct PWM mode).**
- **In PWM modes the OC pins work differently:**
  1. **At the beginning of the PWM cycle the OC pin is either set (normal PWM) or cleared (reverse PWM).**
  2. **When the compare match occurs, the OC pin is either cleared (in normal PWM) or set (in reverse PWM).**
- **That produces a signal with a controlled pulse-width on the pin.**
- **With compare match = 0 a signal of one pulse duration is produced, with compare match = 255 (in an 8-bit timer), the signal is fully on.**

Timer in PWM modes - TOP=255

- **Beside normal-mode and CTC-mode all timer in AVRs have also PWM modes.**
- **In PWM modes the timer counts up (in fast PWM mode) or up and then down (in phase-correct PWM mode).**
- **In PWM modes the OC pins work differently:**
  1. **At the beginning of the PWM cycle the OC pin is either set (normal PWM) or cleared (reverse PWM).**
  2. **When the compare match occurs, the OC pin is either cleared (in normal PWM) or set (in reverse PWM).**
- **That produces a signal with a controlled pulse-width on the pin.**
- **With compare match = 0 a signal of one pulse duration is produced, with compare match = 255 (in an 8-bit timer), the signal is fully on.**

Timer in PWM modes - TOP=255

- **The 8-bit TC0 modes are controlled with the WGM bits:**

Table 11-8. Waveform Generation Mode Bit Description

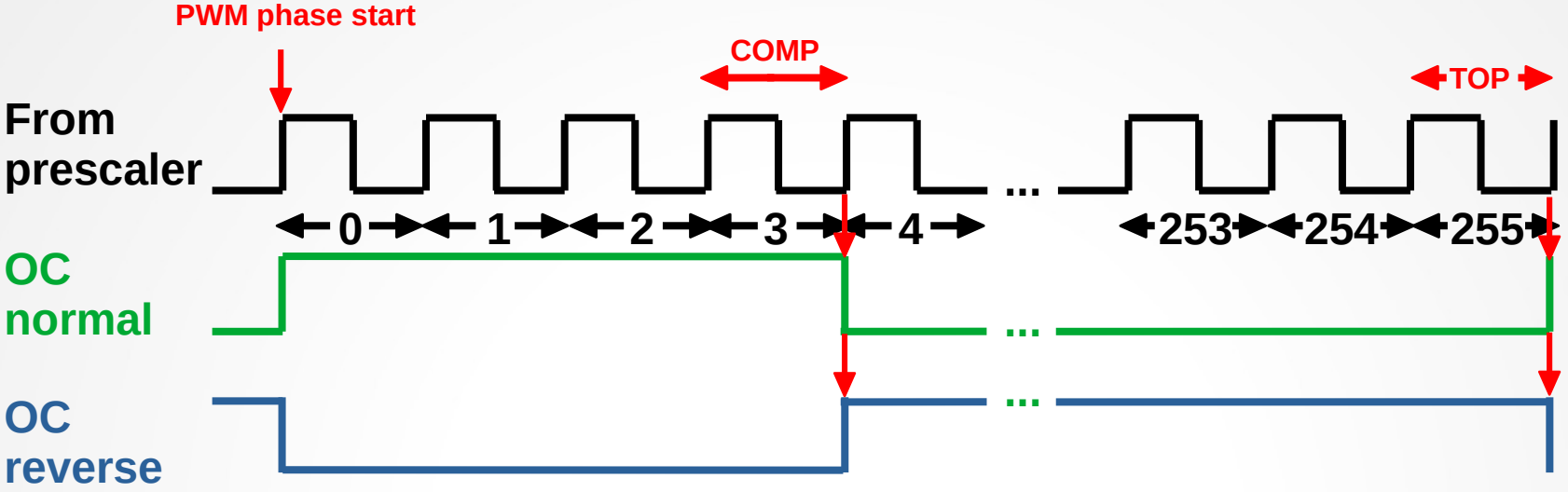
Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Note: 1. MAX = 0xFF
BOTTOM = 0x00

- **There are two Fast PWM modes: one with 0xFF and one with OCR0A as TOP values.**

Timer in PWM modes - TOP=255

- The fast PWM mode with TOP=255 works as follows:



- At TCNT0=0 (Phase start) the OC pin is set (OC normal) or cleared (OC reverse).
- On the next counting pulse after compare match has occurred, the OC pin is cleared (OC normal) or set (OC reverse).
- That changes only when the TOP value (255) is reached: on the next counting pulse following TOP match the PWM restarts at 0 and the OC pins are set or cleared.

Timer in PWM modes - TOP=255

- In PWM modes changed compare values are not immediately applied.
- Writing compare values writes those to an interim storage instead of the compare match port register.
- Only after a new PWM cycle starts at 0 the stored new compare match value is actually written to the compare port register.
- That mechanism ensures that in each PWM cycle exactly one compare match will occur, even if the new compare match value is below the current count: it is only applied at the next PWM cycle start.
- The column „Update OCRx“ in the mode table shows that.
- In fast PWM with TOP=255 all three interrupts (overflow, compare match A and compare match B) can be used.

Timer in PWM modes - TOP=OCRA

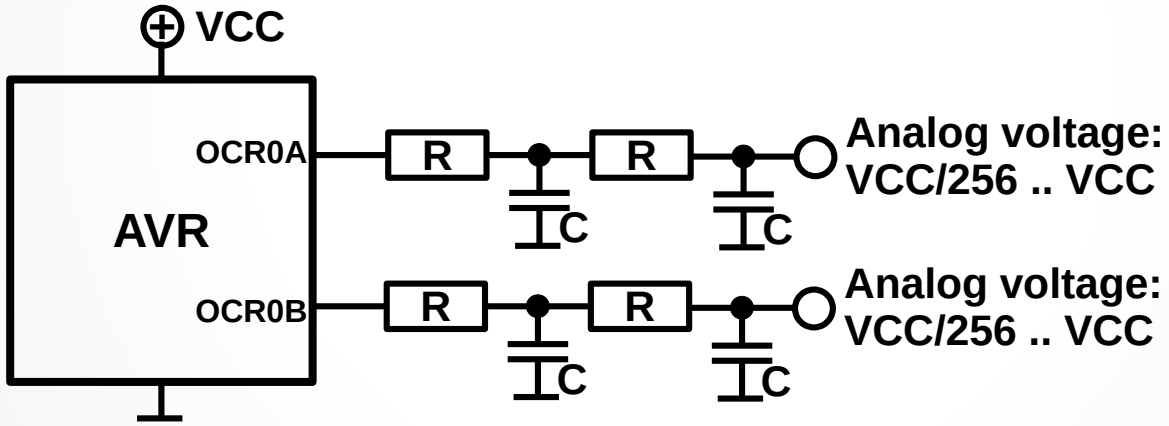
- In fast PWM mode with TOP in OCRA the TOP value can be set to any value to achieve a shorter PWM period (e. g. 128, 64 or 10).
- This increases the PWM frequency, but limits the compare match range, too.
- Changes to OCRA also come only into effect on PWM cycle start.
- As OCRA is used, only OCRB can be used as PWM channel.
- If OCRA is below 255, the overflow interrupt condition will never be reached, use the compare match A and/or B interrupt instead.

Practical applications of fast PWM

- **Fast PWM can be used for the following:**

  a) for LED dimming: the LED is only on in the active phase of the PWM, so the time over which the LED lights can be reduced from full down to 1 / 256 = 0.39% of the time.

  b) for conversion of a digital value to an analog voltage (use an RC filter to reduce the ripple of the PWM voltage):



  c) for motor power regulation (switched power): use a power transistor to achieve higher power consumption, avoid humming by a higher-than-audible PWM frequency.

Conclusions timers and PWM

- **With a controller producing signals with a timer in the different modes and with interrupts you can:**

  a) **generate exact signals and control those with interrupts,**

  b) **avoid lengthy counting sequences if you need exact timing,**

  c) **with a few lines of fast, lean and effective source code resolve any timing and signal task,**

  d) **replace a lot of CMOS or analog electronics by one single controller.**

Questions and tasks in Lecture 7

Task 7-1: Write a program that blinks a LED with a crystal clock at 4 MHz in an exact one-second rhythm using TC0 in CTC mode and its COMP-A interrupt to down-count a register. Don‘t forget to restart the counter register when exhausted!

Bonus question: Without counter register restart: which blink frequency would result instead of 1 Hz?

Questions and tasks in Lecture 7 - Continued

Task 7-2: Write a program with the 16-bit counter TC1 as sound generator with 1,000 Hz in CTC mode and use the COMP-A-interrupt to down-count a 16-bit double register R25:R24 to stop the sound after exactly two seconds. (Hints: 16-bit double registers can be down-counted in 16 bit mode with SBIW R24, 1 (SuBtract Immediate Word) and sound output can be stopped by clearing the pin on compare match when keeping TC1 still running).

Bonus question: What would be the COMPA values and the down-count value in R25:R24 for the whole a-to-g gambit frequencies?

Questions and tasks in Lecture 7 - Continued

Task 7-3: Write a program that provokes two interrupts at exactly the same time with the two timers TC0 and TC1 in the ATtiny24 and with the two compare A interrupts. Simulate this with avr_sim and see if the prioritization of interrupts function as desired within the simulator.