NTNU
Department of Engineering Cybernetics

Suggested Solution

Examination paper for TTK4155 Industrial and Embedded Computer Systems Design

Academic contact during examination: Jo Arve Alfredsen

Phone: 90945805

Examination date: Thursday 2017-11-30

Examination time (from-to): 09:00 - 13:00

Permitted examination support material: D

Standard pocket calculator permitted.

Printed and handwritten material not permitted.

Other information:

Answers may be given in English or Norwegian

Read the text carefully. Each question may have several parts.

Answers should be concise.

Exam counts 50% of final grade.

Language: English

Number of pages (front page excluded): 6

Number of pages enclosed: 0

Informasjon om trykking av eksamensoppgave

Originalen er:

1-sidig ☐ **2-sidig** ☐

sort/hvit ☐ **farger** ☐

skal ha flervalgskjema ☐

Checked by:

_____

Date Signature

It should be noted that these are suggested solutions and that there may exist alternative solutions to some of the problems.

Problem 1. (30 %)

a. Figure 1 shows the diagram of the 8 bit Timer/Counter 2 module of the Atmel AVR ATmega162 microcontroller. This timer module may be clocked by an external 32.768 kHz watch crystal to make it operate as a low-power real-time clock (RTC). Determine the value of the prescaler clock select bits CS22:20 that will give a cyclic timer 2 overflow interrupt (TOV2) at exactly 1 Hz.

(5%)

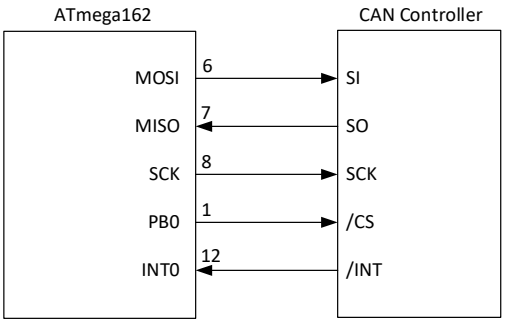
$f_{TOV2} = f_{osc}/(2^8 * \text{prescaler}) = 1$ Hz

$\text{prescaler} = 2^{15}/2^8 = 2^7 = 128$

$CS22:20 = 101_b$

b. The microcontroller in a) should be interfaced to a CAN controller featuring SPI and an interrupt line, constituting an embedded computer intended for a critical control task in a CAN network denoted as CAN node A. See figure 5 for the pinout of ATmega162 and show how you would connect the CAN controller to it.

(5%)



c. Write a program in C/pseudo-code for node A using the TOV2 interrupt that:

1. Keeps track of the number of days, hours, minutes and the total number of seconds the microcontroller has been running since last reset
2. Transmits a CAN message every second containing the total number of seconds since last reset, serving as the "heart beat" signal of node A (assume you have a *CAN_Transmit(CAN_msg_t)* function available)

(10%)

Running time format is days:hours:minutes, e.g. 89:20:56 means 89 days and 20 hours and 56 minutes. Total number of seconds must be stored in an unsigned 32 bit integer (overflows in ~136 years).

```
#include <stdint.h>

#define CAN_ID_TIME = 10

typedef struct {
     unit16_t id;
     unit8_t length;
     unit8_t data[8];
} CAN_msg_t;

uint32_t seconds = 0;
uint8_t minutes = 0;
uint8_t hours = 0;
uint32_t days = 0;

void main {
     // Initialize CAN communication
     Init_CAN();
     // Candidate should somehow show that timer2 is set up in normal mode with the
     // correct prescaler, overflow interrupt enabled and clocked asynchronously
     // by RTC crystal on TOSC1/2
     Init_Timer2(NORMAL,PRESCALER_128,TOV2_ENABLE,ASYNCHRONOUS);
     while (TRUE){
          // Do things here
     }
}

void Send_Time(uint32_t sec) {
     CAN_msg_t msg;

     msg.id = CAN_ID_TIME;
     msg.length = 4;
     msg.data[0] = (sec>>24)&0xFF; // Type casting to uint8_t may be included here
     msg.data[1] = (sec>>16)&0xFF;
     msg.data[2] = (sec>>8)&0xFF;
     msg.data[3] = sec&0xFF;
     CAN_Transmit(msg);
}

ISR(TIMER2_OVF_vect) { // RTC, triggers at 1 Hz
     seconds++;
     if(seconds%60 == 0) {
          minutes++;
          if(minutes >= 60) {
               minutes = 0;
               hours++;
               if(hours >= 24) {
                    hours = 0;
                    days++;
               }
          }
     }
     Send_Time(seconds); // Alternatively set a flag to invoke Send_Time from main loop
}
```

d. A different CAN node B with the same hardware as node A is set to monitor the running state of node A. Node B should trigger an alarm within two seconds if node A stops for some reason. Write a program in C/pseudo-code that implements this mechanism in node B.

(10%)

Worst case when Node A dies immediately after CAN message was sent, and the CAN message arrives at Node B immediately after a Timer2 interrupt time update. The alarm must be triggered at latest on the 2.nd Timer2 interrupt after that to satisfy the (within) two second response time.

```
#include <stdint.h>

#define CAN_ID_TIME = 10

typedef struct {
    unit16_t id;
    unit8_t length;
    unit8_t data[8];
} CAN_msg_t;

uint32_t silent_time = 0;

void main {
    // Initialize CAN communication
    Init_CAN();
    // Candidate should somehow show that timer2 is set up in normal mode with the
    // right prescaler and clocked asynchronously by RTC crystal on TOSC1/2
    Init_Timer2(NORMAL,PRESCALER_128,TOV2_ENABLE,ASYNCHRONOUS);
    while (TRUE){
        // Do things here
    }
}

ISR(INT0_vect) { // Triggered by CAN interrupt
    CAN_msg_t msg;

    if(CAN_Receive(&msg)) { // Assume this function exists
        if(msg.id == CAN_ID_TIME) { // Life sign from A: Reset silent time
            silent_time = 0;
        }
        // Optional: Do things with CAN message
    }
}

ISR(TIMER2_OVF_vect) { // Triggers at 1 Hz
    silent_time++;
    if(silent_time >= 2) {  // Acts like a watchdog: check time since last msg from A
        Alarm(silent_time); // Invoke some alarm handler
    }
}
```

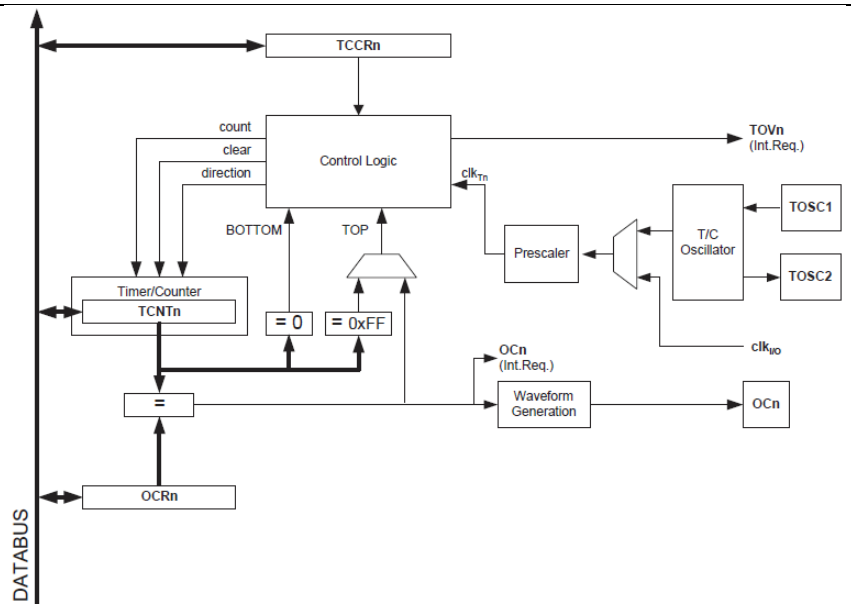
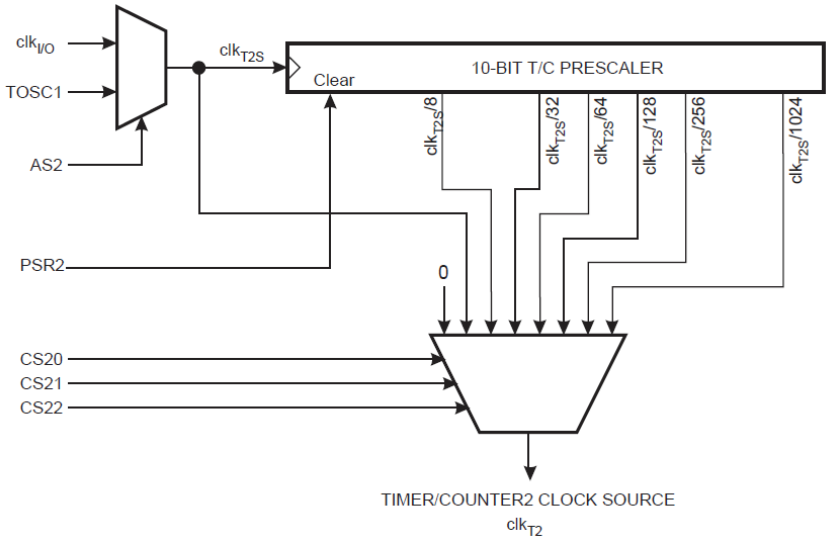


Figure 1. Diagram showing Timer/Counter 2 (n = 2) in the AVR ATmega162.

- Registers TCNT2, TCCR2 and OCR2 are all 8 bits wide.
- In normal mode, Timer/Counter 2 increments TCNT2 by one for each pulse of the clock signal clk_{T2} (0x00→0x01→…→0xFF→0x00→…). The interrupt signal TOV2 (timer overflow) is generated every time TCNT2 overflows from 0xFF to 0x00.
- The signal clk_{T2} is driven by a prescaled version of either the internal clock source $clk_{I/O}$, or an external clock crystal connected between pins TOSC1 and TOSC2.
- The clock signal clk_{T2} is selected by the control bits CS22:20 according to the following diagram:



CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk_T2_S/(No prescaling)
0	1	0	clk_T2_S/8 (From prescaler)
0	1	1	clk_T2_S/32 (From prescaler)
1	0	0	clk_T2_S/64 (From prescaler)
1	0	1	clk_T2_S/128 (From prescaler)
1	1	0	clk_T2_S/256 (From prescaler)
1	1	1	clk_T2_S/1024 (From prescaler)

Problem 2. (30 %)

a. Assume that node A introduced in Problem 1 has an unregulated 6-12 V voltage supply available through an extra wire pair in the CAN cable. To function properly, node A requires both a stable 5 V supply for the microcontroller as well as a 15 V supply for powering an on board ultrasonic transducer.
Explain how you would design a proper power supply for node A and specify the component types you would use.

(5%)

A linear LDO voltage regulator is ideal for the microcontroller 5 V supply because it will work with the low 6 – 5 V = 1 V drop out voltage margin (and provide a stable low noise voltage reference for the comparator circuit). A Switched mode buck regulator could also be considered, especially for power efficiency if the 5 V part of the circuit consumes some current. It is however a bit more expensive, complex and noisy. A standard linear voltage regulator has typically a too large drop out voltage to operate reliably with the specified voltage range, and should not be used.

A switched mode boost regulator could be used to generate the 15 V supply voltage for the ultrasonic transducer.

A dual output flyback regulator could also make a good solution here.

Several solutions are acceptable and the answer will be evaluated based on the candidate's discussion.

b. Node A employs the ultrasonic transducer described in Figure 2 to monitor the fluid level of a tank. Assume the ATmega162's system clock is driven by an external 4 MHz crystal oscillator and use the output compare match unit B of Timer/Counter 1 module described in Figure 3 to time the 10 ms measurement interval of the transducer. The OC1B pin can be set and cleared on match to generate the digital trigger signal.
What value should be written to the OCR1B register?

(5%)

Clocking Counter1 directly from the system clock gives clk_{T1} = 4 MHz, and a compare match value of: OCR1B = 4000000Hz*0.01s = 40000

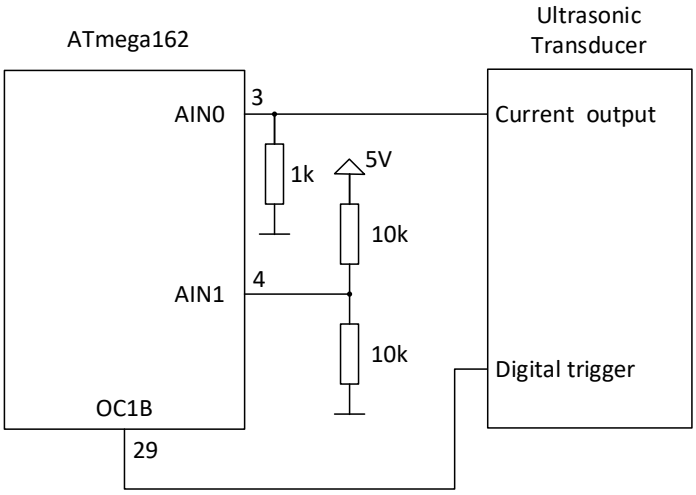
Using prescalers of 8 and 64 will also yield an integer in the OCR1B register, but will only contribute to decrease the time resolution of the measurement.

c. The ATmega162's Analog Comparator module described in Figure 4 can be used to trigger an input capture event in Timer/Counter 1. Explain how you can exploit this mechanism together with a few external resistors to make level measurements with the ultrasonic transducer.

(10%)

The current output from the ultrasonic transducer can be led through a resistor to generate a proper voltage signal to the comparator AIN0 input. A resistor value of 1k will give a voltage v_{AIN0} = 2.5 mA * 1k = 2.5 V. The voltage at the AIN1 input then needs to hold a constant voltage of 2.5 V which easily can be generated by dividing the 5 V voltage supply through two equal resistors of e.g. 10k. The comparator can then be set up using the control bits to generate an input capture event on timer/counter1.

Alternatively, the internal bandgap reference could be used with the transducer signal connected to AIN1 and through a shunt resistor (440R). In case, it should be explained how the comparator should be operated in inverted mode.



d. Write a program in C/pseudo-code for node A that implements the above mechanisms to make a level measurement and transmit it as a CAN message every second.

(10%)

We assume that the transducer is connected to the microcontroller as shown in 2.c). Specification says that one measurement must be made and sent over CAN every second, and the general idea is to let Timer2/TOV2 time the 1 Hz measure&transmit sequence reusing the solution from 1.c). Timer/Counter1 is enabled to time the 10 ms ultrasonic measurement using the compare match B interrupt (OCR1B = 40000), and the input capture event interrupt (triggered by the analog comparator) is enabled to measure the sound pulse time-of-flight. The OC1B-pin is set to start the transducer measurement, and to will clear automatically at the compare match interrupt (depending on the COM1B-bits setting). The interrupts set flags that are picked up in the main-loop to start the ultrasonic level measurement, and calculate and transmit the level as a CAN message. Some of the function calls are not implemented in the code below because they are self-explanatory and/or are hard to detail based on the limited information in the timer/comparator module descriptions given here (which is acceptable for a pseudo-code implementation).

```
#include <stdint.h>

#define CAN_ID_LEVEL 100
#define VC 340
#define H 1700 // Assume height of tank 1700 mm
#define INTERVAL 40000 // Measurement interval time, equals 10 ms

typedef struct {
    unit16_t id;
    unit8_t length;
    unit8_t data[8];
} CAN_msg_t;

uint8_t start_measure_flag = 0;
uint8_t end_measure_flag = 0;
uint16_t time_of_flight = 0;

void main {
    uint16_t level_mm = 0;

    DDRE |= (1<<DDE2); // Set OC1B pin to output for digital trigger to transducer
    // Initialize CAN communication
    Init_CAN();
    // Initialize Timer2 to generate an asynchronously clocked 1 Hz interrupt
    Init_Timer2(NORMAL,PRESCALER_128,TOV2_ENABLE,ASYNCHRONOUS);

    while (TRUE){
        if(start_measure_flag) { // Will trigger every 1 s
            start_measure_flag = 0;
            time_of_flight = 0;
            Enable_Comparator(); // Let comp trigger input capture event (ACIC=1)
            OCR1B = INTERVAL; // Set measurement interval in compare match register
            PORTE |= (1<<PE2); // Set digital trigger signal to transducer
            // Then start timer to generate capture and compare interrupts and
            // clear OC1B pin on compare match (COM1B1:COM1B0 = 10b)
            Start_Timer1(NORMAL,NO_PRESCALER,CAPTURE_ENABLE,COMPB_ENABLE,OC1B_CLR);
        }
        if(end_measure_flag) { // Will trigger when measurement interval completes
            end_measure_flag = 0;
            if(time_of_flight > 0) { // An echo was detected
                level_mm = H - VC*time_of_flight/8000; // Return level in millimeter
                Send_Level(level_mm);
            }
            else {
                // There was no echo, do nothing or send error code “infinity”
            }
        }
    }
}

void Send_Level(uint16_t l) { // Send CAN message containing tank level in mm
    CAN_msg_t msg;

    msg.id = CAN_ID_LEVEL;
    msg.length = 4;
    msg.data[0] = (l>>8)&0xFF;
    msg.data[1] = l&0xFF;
    CAN_Transmit(msg);
}

ISR(TIMER1_COMPB_vect) { // Triggers when counter1 equals INTERVAL, i.e. 10 ms

    Stop_Timer1();  // Measurement interval completed -> stop timer
    end_measure_flag = 1; // Signal that a new level calculation can be made
}

ISR(TIMER1_CAPT_vect) { // Triggers on input capture event generated by comparator
    time_of_flight = ICP1;   // Save the Input Capture value for level calculation
    Disable_Comparator(); // Let only the first comparator event trigger a capture
}

ISR(TIMER2_OVF_vect) { // Triggers at 1 Hz and starts a measure&send sequence
    start_measure_flag = 1;
}
```

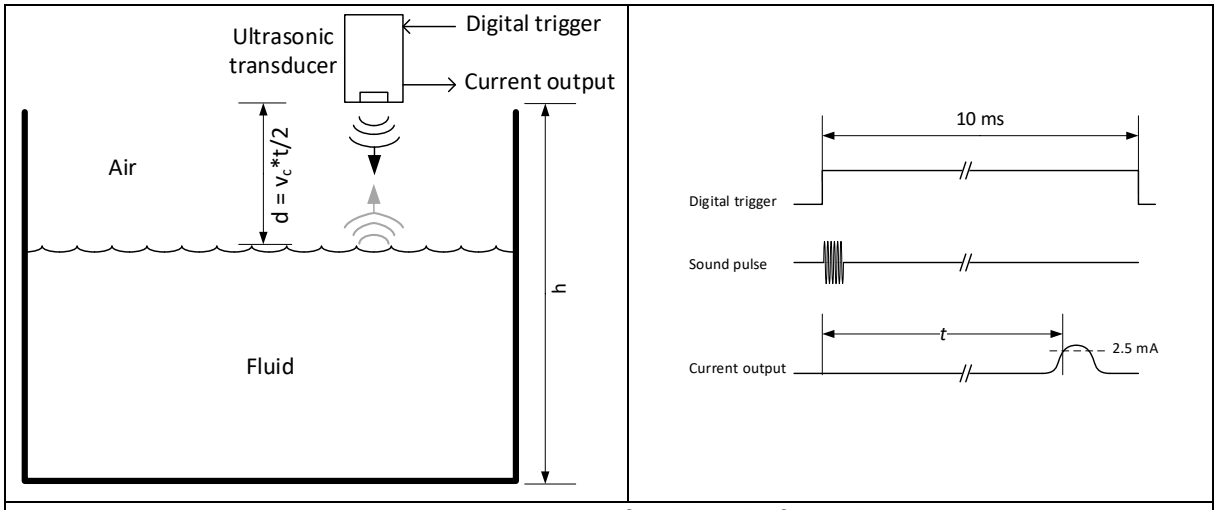


Figure 2. Ultrasonic transducer measuring the fluid level of a tank.

When triggered by a digital signal, the transducer emits a short pulse of sound and then immediately starts to sense sound energy reflected back from objects along the path of the sound wave. The time it takes for the sound to travel from the transducer to the object and back, represents the distance to the object (time-of-flight measurement). The level of the tank can then be calculated as $l = h - d = h - v_c*t/2$, where t is the time-of-flight of the ultrasonic signal and v_c = 340 m/s is the speed of sound.

The transducer outputs a current signal proportional to the rms-amplitude (root-mean-square) of the reflected signal during the 10 millisecond time interval immediately after emission of the sound pulse. Calibration tests have shown that a current output signal greater than 2.5 mA indicates reflection off the fluid surface at all relevant distances.

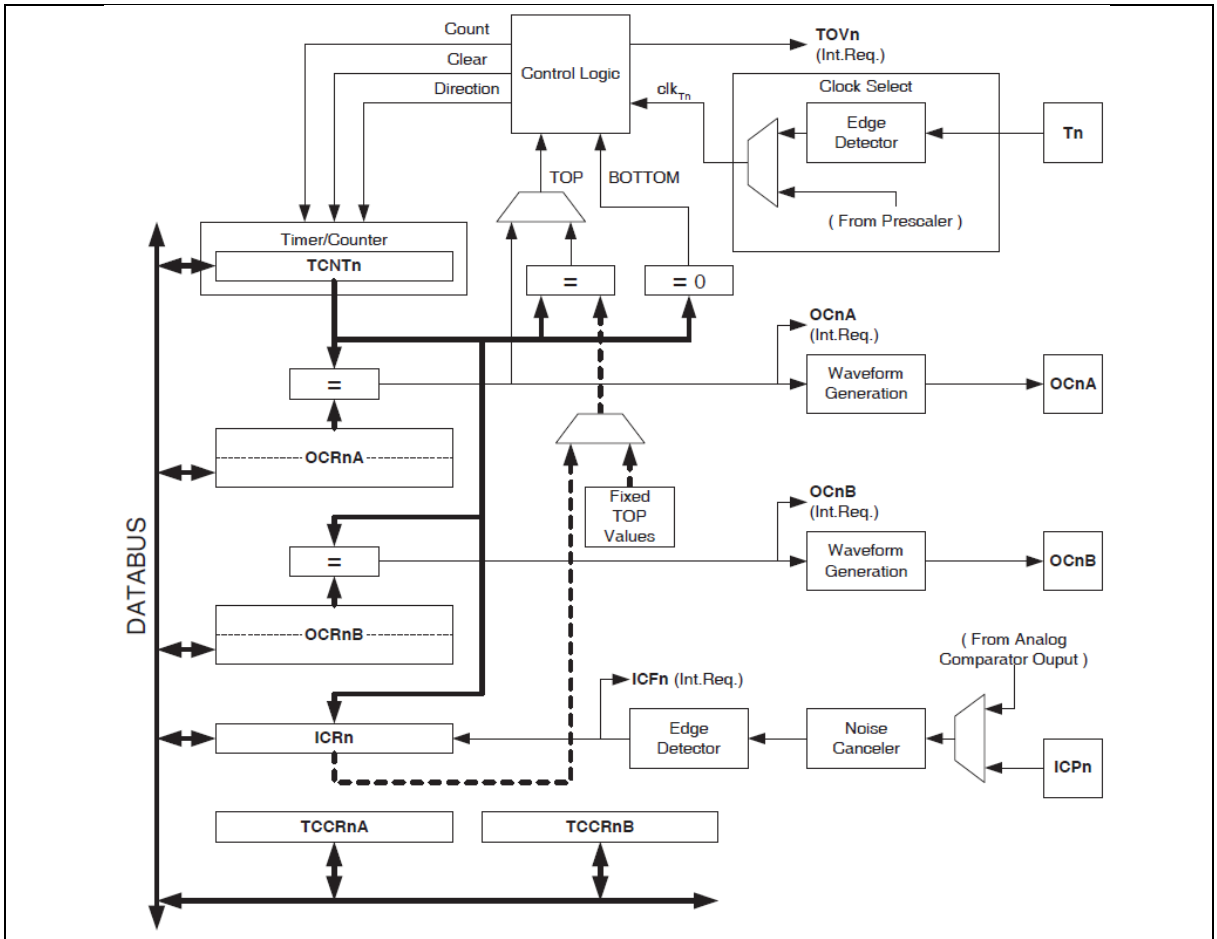


Figure 3. Diagram showing Timer/Counter 1 and 3 in the AVR ATmega162.

- In Timer/Counter n (n = 1 or 3), the TCNTn, OCRnA, OCRnB and ICRn registers are all 16 bit.
- The Timer/Counter can be driven by internal or external clock sources, either directly from the system clock, via a prescaled (divided) version of the system clock, or via a clock signal on pin Tn. The selectable prescaler values are 8, 64, 256 or 1024.
- In *Normal* mode, TCNTn increments by one for each clock pulse (0x0000→0x001→…→0xFFFF→0x0000→…). The interrupt signal TOVn (timer overflow) is generated every time the TCNTn overruns from 0xFFFF to 0x0000.
- The ICRn (input capture register) is used in normal mode to capture/timestamp external events in terms of rising and falling edges on the ICPn pin or a trigger signal from the microcontroller's internal Analog Comparator unit. When a rising or falling edge is detected on the ICPn pin or comparator output, the current value of the timer register TCNTn gets loaded immediately into ICRn and the input capture interrupt signal ICFn is generated. The ICESn bit of the timer control register TCCRnB selects which edge of the ICPn pin signal triggers the capture (0 = falling edge, 1 = rising edge).
- The OCRnA and OCRnB are two output compare registers that are used to generate the interrupt signals OCnA and OCnB as well as setting or resetting the OCnA and OCnB pins when the timer TCNTn matches their content.
- When Timer n is set to *Clear Timer on Compare (CTC) mode*, TCNTn will be cleared to zero when a match with the content of OCRnA is detected and the interrupt signal OCnA will be generated and the OCnA pin will be toggled.
- When Timer n is set to *Fast PWM mode*, TCNTn counts from BOTTOM (0x0000) to TOP and then restarts from BOTTOM again. TOP can be set to fixed values 0x00FF, 0x01FF, 0x03FF or the content of OCRnA or ICRn. The OCnx pin is set on a compare match between TCNTn and OCRnx, and cleared at TOP. A TOVn interrupt is generated when TCNTn rolls over from TOP to BOTTOM.

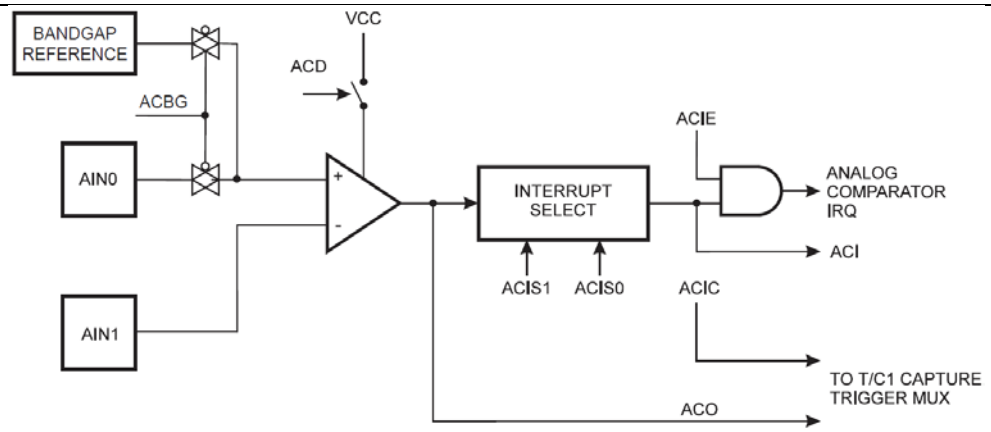


Figure 4. Diagram of the internal Analog Comparator module of the AVR ATmega162.

The Analog Comparator compares the input voltages on the positive pin AIN0 and negative pin AIN1. When the voltage on the positive pin AIN0 is higher than the voltage on the negative pin AIN1, the Analog Comparator Output, ACO, is set. In addition, the comparator can trigger a separate interrupt (ANA_COMP), exclusive to the Analog Comparator. The user can select interrupt triggering on comparator output rise, fall or toggle by setting the control bits ACIS1 and ACIS0 to 11, 10 or 00 respectively. The comparator's output can also be set to trigger the Timer/Counter1 Input Capture function by setting the ACIC control bit. When the ACBG control bit is set, a fixed bandgap reference voltage of 1.10V replaces the positive input AIN0 to the Analog Comparator.

Problem 3. (40 %)

Node A from Problem 1 and 2 should be developed further into a fully functional industrial sensor device featuring more memory and a display. Sensors for temperature and humidity should also be included as sound velocity depends on these parameters. The device's detailed additional specification is given as follows:

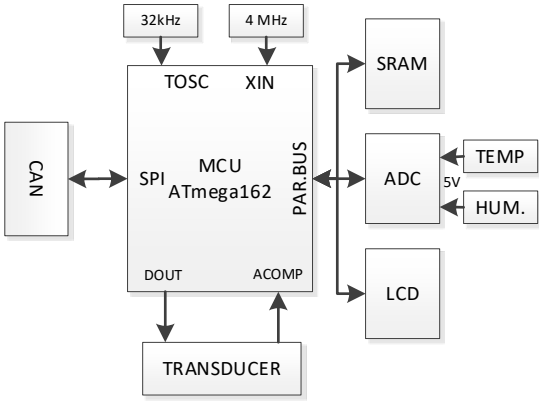
- The system should be built up around a microcontroller of type AVR ATmega162 as shown in Figure 5.
- ATmega162 features a relatively small amount of internal SRAM and requires inclusion of an extra external 32 kByte SRAM.
- The temperature and humidity sensors measure temperature in the range -15 – 50 °C and relative humidity in the range 0 – 100 %, respectively, and outputs analog signals in the range 0 – 5 V.
- To read sensor signals you have a 2-channel AD converter with 16-bit resolution and a parallel bus interface available. The AD converter accepts voltage signals in the range 0 – 5 V. The AD converter has an internal address space of 6 bytes. The AD conversion is started when an 8-bit control word is written to the control register at address 0 in the AD converter. Address 1 is reserved for status information (status register). When the conversion has finished, the result (16-bit) will be stored at the addresses 2n+2 and 2n+3, where $n \in [0, 1]$ is the channel number. The interrupt pin of the AD converter is then pulled low. Note: the AD converter features a parallel bus interface. In other words, it must be interfaced as a pure memory mapped I/O unit using the external address and data bus of the microcontroller.
- The display has 2 lines x 20 characters and features a parallel bus interface. The display operates as a pure memory mapped I/O and can be accessed by writing/reading to/from a linear 40 byte memory inside the display where the addresses corresponds to the character

positions on the display (first character on the first line has address 0, first character on the second line has address 20). Moreover, the display features ten 8-bit status- and control registers, making the display require a total of 50 bytes of the address space.

a. Describe the system in terms of a *high-level* block diagram (a detailed circuit schematic not requested here). Read the specification above in detail, focus on identifying and drawing the individual function blocks/modules that together make up the system, and then indicate the interface between them using simple arrows and labels (no detailed bus/signal connections requested here). If deemed necessary, make your own reasonable assumptions that will make the system function properly.

(10%)

Here Node A from Problem 1 and 2 should be extended to include SRAM, LCD, ADC and sensors. All components and their interfaces must show.



b. Assume that the base address of the display should put at 0x4000. Explain how you would organize the address space of the computer in the simplest possible way, and derive the associated decoding logic (remember that the 1280 lowest addresses (0x0000 - 0x04FF) of the ATmega162 are reserved).

(15%)

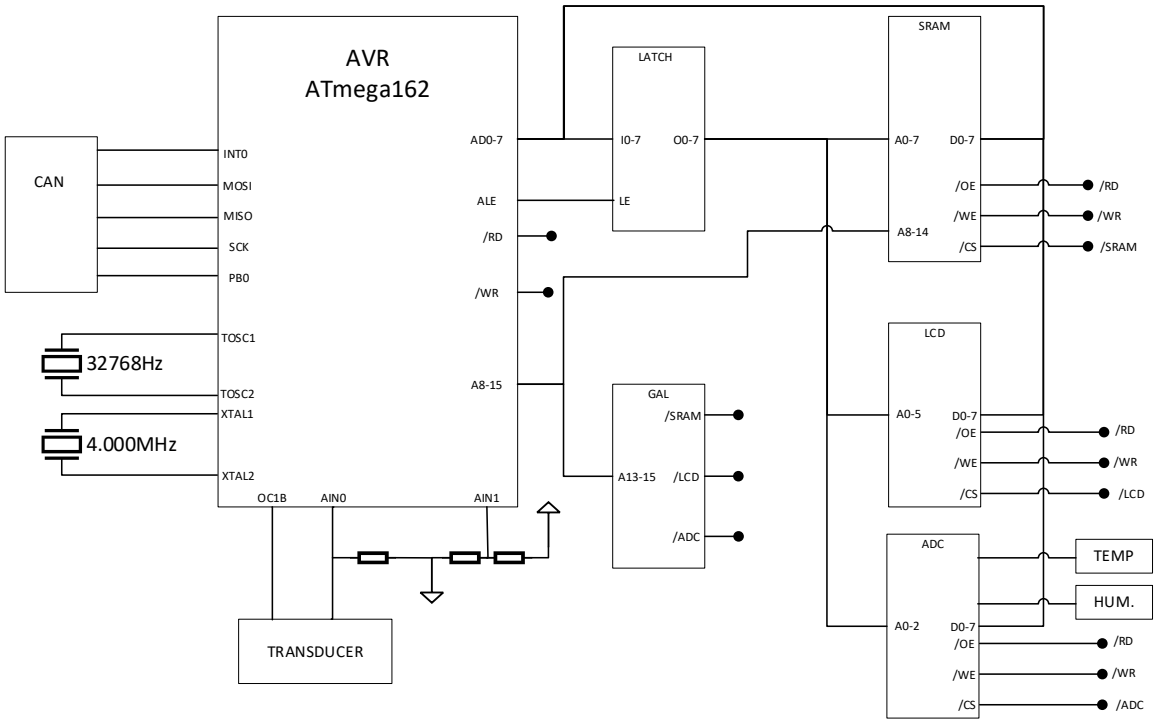
According to the specification, four devices must be made room for in the 64 kB (16 bit) address space of the MCU: SRAM, ADC, LCD and AVR (reserved addresses). The SRAM, taking half of the memory space, can be located to the upper half of the address space. Furthermore, according to the partial decoding approach, the lower half of the address space can be divided into four equally sized separate address ranges to accommodate for the remaining three devices. Decoding: Let A_{15} MSB of the address bus select SRAM with base address 0x8000, and the two next bits $A_{14}A_{13}$ select one of the four lower ranges. 0x0000 is reserved for the AVR, making 0x2000, 0x4000 and 0x6000 possible base addresses for the LCD and ADC. LCD is specified to base address 0x4000, making either 0x2000 or 0x6000 available base address for ADC (LCD could alternatively be decoded by using only $A_{15}A_{14}$ giving it 16 kB in the range 0x4000-0x7FFF).

Reserved AVR 8 kB (0x0000 – 0x1FFF)
ADC 8 kB (0x2000 – 0x3FFF)
LCD 8 kB (0x4000 – 0x5FFF)
Free 8 kB (0x6000 – 0x7FFF)
SRAM 32 kB (0x8000 – 0xFFFF)

Active low logic: $CS_{RAM} = A_{15}$, $CS_{ADC} = /A_{15}/A_{14}A_{13}$, $CS_{LCD} = /A_{15}A_{14}/A_{13}$

c. Draw and explain a circuit schematic that shows how the components of the system are connected together (the details can be limited to central signal lines, but the width of all buses and which ports/bit signals are connected to must be shown). Specify the circuits and signals you find necessary to add.

(15%)



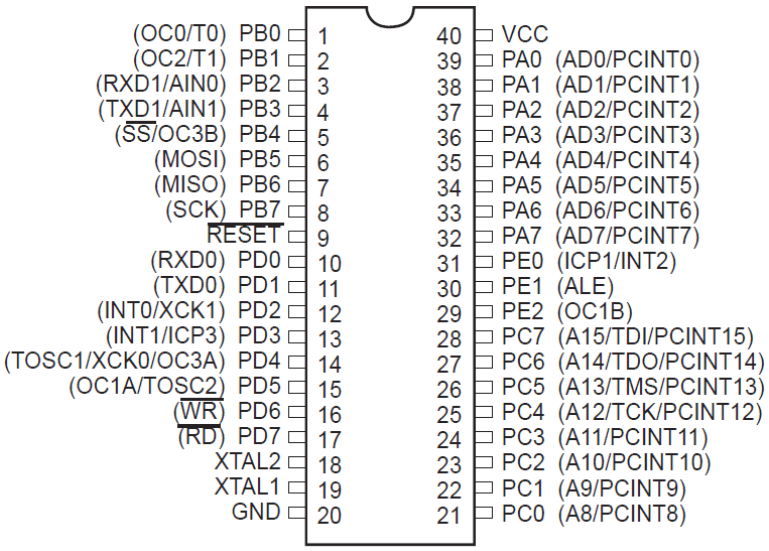


Figure 5. Atmel AVR ATmega162.