

Designnotat

Tittel: Designprosjekt 7 - Testbenk for kabelkommunikasjon	
Forfattere: Temmam Nabil Altaie	
Versjon: 1.0	Dato: 21.10.25

Innhold

1 Problembeskrivelse **2**

2 Prinsipiell løsning **3**

2.1 Overordnet systemidé . . . 3
2.2 Kanalmodell . . . 4
2.3 Støy og signal-til-støy-forhold . . . 4
2.4 Bitfeilrate og pakketap . . . 5
2.5 Sammenheng mellom parametere . . . 5
2.6 Prinsipiell oppsummering . . . 5

3 Realisering og test **7**

3.1 Maskinvare og kobling . . . 7
3.2 Programvareimplementasjon . . . 8
3.3 Testoppsett og målemetode . . . 9
3.4 Resultater . . . 9
3.5 Vurdering av resultater . . . 11
3.6 Oppsummering . . . 11

4 Konklusjon **12**

5 Takk **12**

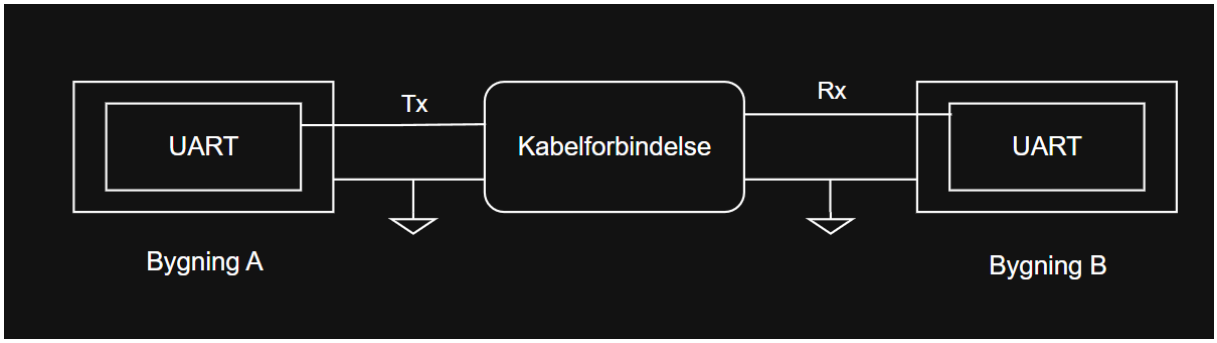
Referanseliste **13**

6 Kodevedlegg **14**

6.1 main.c . . . 14
6.2 avrdriver.c . . . 18
6.3 avrdriver.h . . . 25

1 Problembeskrivelse

Det skal etableres datakommunikasjon mellom to bygninger, A og B, der trådløs forbindelse ikke kan benyttes av sikkerhetsmessige årsaker. Kommunikasjonen foregår via asynkron seriell overføring gjennom en kabel.



Figur 1: Blokkskjema som illustrerer problemstillinga.

Kabelen har resistans, kapasitans og eventuelt en liten induktans som fører til demping og forvrengning av signalet. I tillegg kan støy introduseres i overføringen, noe som kan føre til bitfeil eller pakketap. For å kunne vurdere hvor pålitelig en slik forbindelse er, må sammenhengen mellom baudrate, signal-til-støy-forhold (SNR) og bitfeilrate (BER) undersøkes.

Siden den fysiske kabelen ennå ikke er tilgjengelig, skal det utvikles en testbenk som modellerer kabelens egenskaper og gjør det mulig å måle BER og pakketap under ulike betingelser.

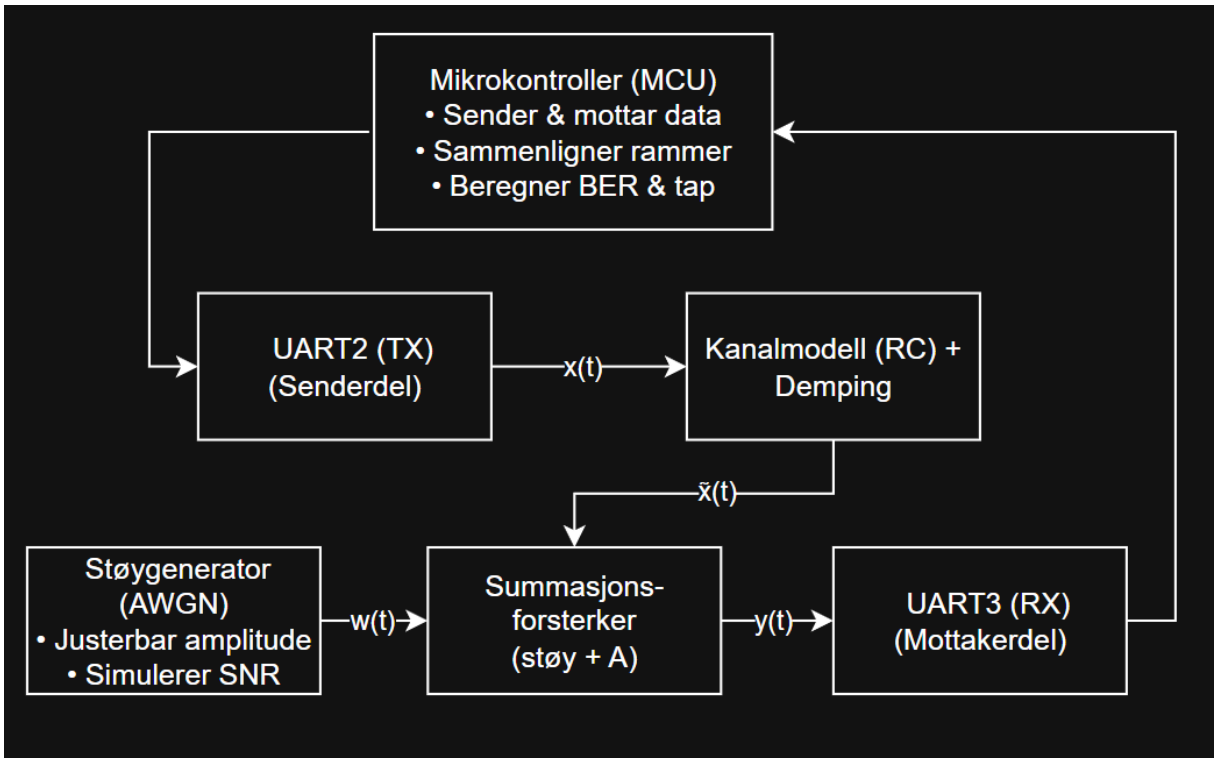
Problemstillingen kan oppsummeres slik:

> **Hvordan påvirkes bitfeilraten (BER) i en kabelbasert UART-kommunikasjon av baudrate og signal-til-støy-forhold (SNR)?**

2 Prinsipiell løsning

I denne delen beskrives hvordan systemet kan bygges opp på et prinsipielt nivå for å undersøke sammenhengen mellom baudrate, støy og bitfeilrate i et kabelbasert UART-samband. Målet er å etablere en fysisk modell av kommunikasjonskanalen som etterligner virkelige egenskaper for motstand, kapasitans og støy.

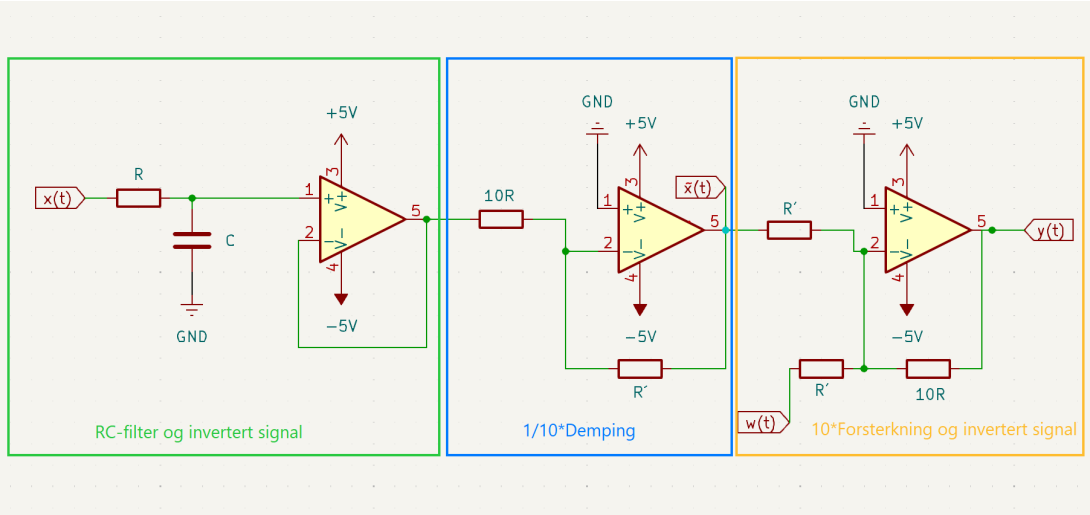
2.1 Overordnet systemidé



Figur 2: Prinsipielt blokkskjema av testbenken med mikrokontroller, kanalmodell og støygenerator.

Som vist i figur 2 er systemet bygget opp rundt en mikrokontroller som sender og mottar data via to uavhengige USART-enheter (UART2 og UART3). Mellom sender og mottaker er det koblet inn en *kanalmodell* som representerer kabelens elektriske egenskaper. I tillegg til signalforvrengning kan det tilføres justerbar støy fra en signalgenerator.

Systemet registrerer antall feil i mottatt datastrøm ved å sammenligne utsendte og mottatte pakker. Ut fra dette beregnes *bitfeilrate (BER)* og sannsynlighet for pakketap.



Figur 3: Prinsipielt kretsdesign av den analoge delen.

Figur 3 viser kretsdesignet til de analoge komponentene. Kretsen inneholder et RC-Ledd, en inverterende operasjonsforsterker (opamp), dempende opamp og forsterkende opamp.

2.2 Kanalmodell

(Formlene som blir presentert, tar utgangspunkt i dokument[1].)

Kabelen modelleres som et lineært, tidsinvariant system (LTI) med demping og lavpasskarakteristikk. Av praktiske hensyn benyttes en førsteordens RC-modell der tidskonstanten τ bestemmer hvor raskt signalet endrer seg:

$$\tau = RC \tag{1}$$

Overføringsfunksjonen for et slikt system er gitt ved

$$H(s) = \frac{1}{1 + sRC} \tag{2}$$

Modellen gir økende dempning med frekvens, noe som gjør at høye baudrater fører til større forvrengning av de raske overgangene i et digitalt signal. Når baudraten B øker slik at signalets hovedfrekvenskomponenter nærmer seg systemets grensefrekvens

$$f_c = \frac{1}{2\pi RC}, \tag{3}$$

vil signalet miste sin firkantform og dermed bli vanskeligere å tolke på mottakersiden.

Et forenklet uttrykk for forholdet mellom utsendt og mottatt spenning er

$$\frac{V_{\text{ut}}}{V_{\text{inn}}} = \frac{1}{\sqrt{1 + (2\pi f RC)^2}} \tag{4}$$

2.3 Støy og signal-til-støy-forhold

For å simulere realistiske forhold tilføres kanalen hvit gaussisk støy $w(t)$ med middelverdi 0 og effektivverdi V_w. Signalnivået på mottakssiden betegnes V_x. Signal-til-støy-forholdet (SNR)

defineres da som

$$SNR = 20 \log_{10}\left(\frac{V_x}{V_w}\right) \text{ [dB]} \tag{5}$$

Ved å justere amplituden på støyen kan man undersøke hvordan bitfeilraten endres for ulike verdier av SNR. Når støyen øker, vil flere bits tolkes feil, og BER vil stige.

2.4 Bitfeilrate og pakketap

Bitfeilraten estimeres ved å sende et stort antall bits N_s og telle hvor mange N_f som mottas feil:

$$\hat{p}_e = \frac{N_f}{N_s} \tag{6}$$

Presisjonen til estimatet avhenger av antall sendte bits, og standardavviket for estimert BER kan uttrykkes som

$$SD = \frac{1}{\sqrt{N_s}} \sqrt{p_e(1 - p_e)} \tag{7}$$

I tillegg estimeres sannsynligheten for pakketap P_{tap}, som defineres som andelen sendte pakker som ikke blir korrekt mottatt innen en tidsgrense (timeout). Dette oppstår gjerne ved lav SNR eller høy baudrate.

2.5 Sammenheng mellom parametere

For et UART-samband gjelder generelt at høyere baudrate reduserer tiden per bit og dermed øker kravet til båndbredde i kanalen. Økt baudrate vil derfor gi større sannsynlighet for forvrengning og bitfeil, spesielt når B nærmer seg kanalens grensefrekvens f_c. Kvaliteten på kommunikasjonen kan derfor oppsummeres som en funksjon av både baudrate og støy:

$$BER = f(B,\, SNR) \tag{8}$$

Ved høye SNR og lave baudrater forventes $BER \approx 0$, mens lav SNR og høy baudrate gir $BER \to 0.5$, som tilsvarer helt tilfeldig mottak.

2.6 Prinsipiell oppsummering

Den prinsipielle løsningen består dermed av tre hovedelementer:

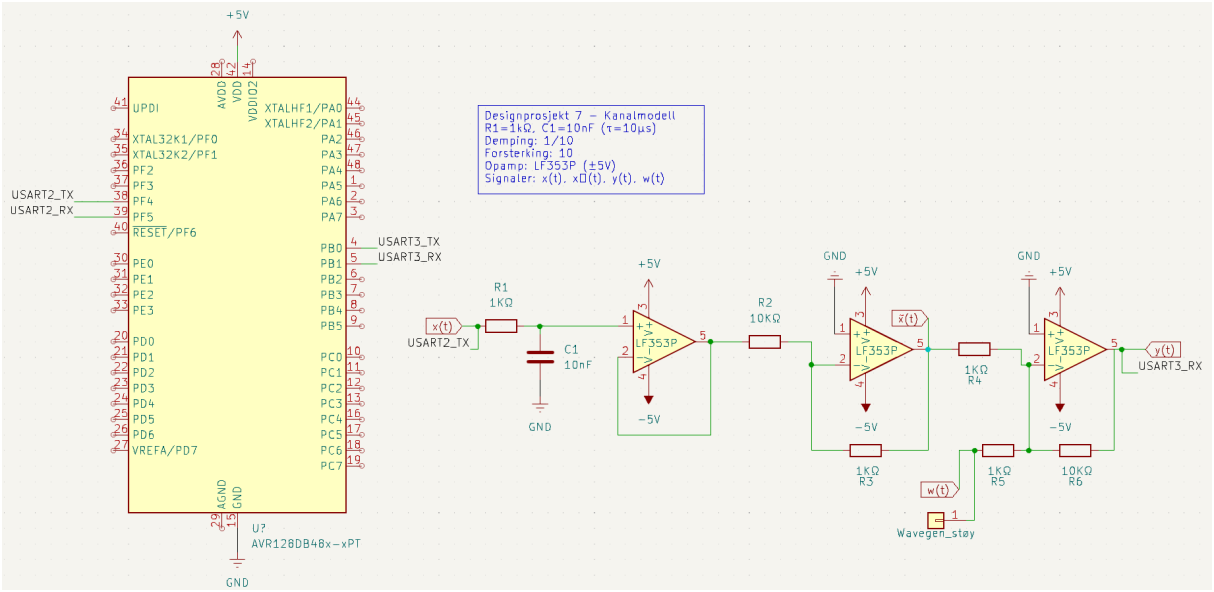
- **Kanalmodell:** RC-ledd som representerer kabelens forvrengning.
- **Støytilførsel:** Hvit gaussisk støy fra signalgenerator for å variere SNR.
- **Testbenk:** Mikrokontroller som sender og mottar data for beregning av BER og pakketap.

Denne strukturen gjør det mulig å undersøke hvor robust en UART-forbindelse er under realistiske forhold, og hvordan baudrate og støy påvirker kommunikasjonens kvalitet.

3 Realisering og test

I denne delen beskrives hvordan testbenken ble implementert og testet i praksis. Målet var å undersøke hvordan baudrate og signal-til-støy-forhold (SNR) påvirker bitfeilrate (BER) og pakketap i et UART-basert kabelsamband.

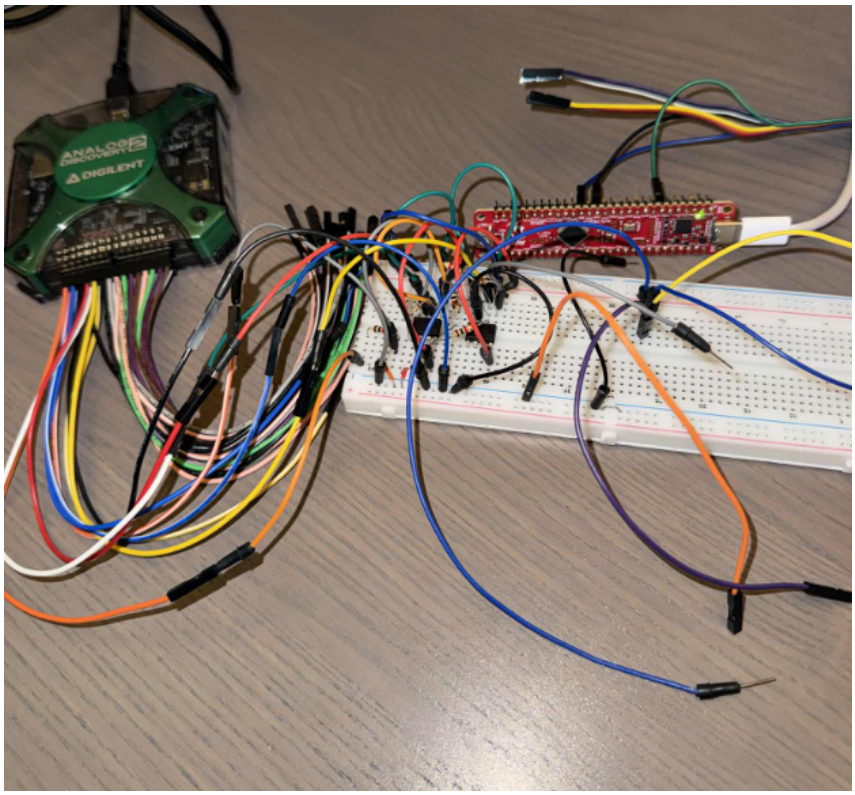
3.1 Maskinvare og kobling



Figur 4: Endelig koblingsskjema for kanalmodell og mikrokontrollergrensesnitt.

Det endelige koblingsskjemaet for den fysiske kanalmodellen er vist i figur 4. Kretsen består av et RC-ledd som modellerer kabelens lavpasskarakteristikk, etterfulgt av en bufferforsterker for å unngå lastpåvirkning. Deretter følger en dempende opamp ($A^{-1} = 0.1$) og en forsterkende opamp (A=10), slik at den totale forsterkningen etter kanal og støytilførsel blir nøytral. Summasjonsforsterkeren mottar signalet fra kanalen samt hvit gaussisk støy $w(t)$ fra en signalgenerator (Wavegen i Analog Discovery).

Mikrokontrolleren (AVR128DB48) ble koblet slik at `USART2` sender data (TX) gjennom kanalmodellen, mens `USART3` mottar signalet (RX). Begge UART-kanalene var konfigurert med samme baudrate. Et felles jordreferansepunkt ble brukt for mikrokontroller, støygenerator og analog krets for å sikre stabil signaloverføring.



Figur 5: Realisert krets koblet på koblingsbrett.

Figur 5 viser den realiserte kretsen. RC-leddet og de tre operasjonsforsterkerne ble bygget på koblingsbrett. Forsterkningen ble testet ved å sende et kjent firkantsignal gjennom kanalen og observere utgangen på oscilloskopet. Demping og forsinkelse samsvarte med forventet lavpass-respons.

3.2 Programvareimplementasjon

Programvaren ble utviklet i C for AVR128DB48 og er basert på koden vist i kapittel 2. Systemet genererer tilfeldige datarammer på `USART2`, sender dem gjennom kanalmodellen, og leser resultatet via `USART3`. Forskjellen mellom sendte og mottatte data brukes til å beregne antall bitfeil og pakketap.

- **Datagenerering:** Funksjonen `generate_random_frame()` lager en tilfeldig bitstrøm.
- **Sending:** `send_frame_usart2()` sender rammen over kanalen.
- **Mottak:** `receive_frame_usart3()` leser data med `timeout` for å kunne registrere pakketap.
- **Analyse:** `count_bit_errors()` teller bitforskjeller mellom sendt og mottatt data.
- **Logging:** `printf()` skriver ut resultatene til terminal via USB.

Timeout ble implementert for å unngå at systemet låser seg ved pakketap. Dersom `USART3` ikke mottok data innen `TIMEOUT_MS`, ble rammen registrert som tapt. BER ble beregnet som andelen

feilbiter per overførte bit:

$$BER = \frac{\text{bitfeil}}{\text{totalt antall bits}}$$

3.3 Testoppsett og målemetode

Testene ble utført ved hjelp av Analog Discovery 2 (AD2). Wavegen-funksjonen ble brukt til å tilføre hvit gaussisk støy (AWGN) til summasjonsforsterkeren. Målingene ble gjort for følgende baudrater:

$$B = [9600, 19200, 38400, 76800, 115200] \text{ baud}$$

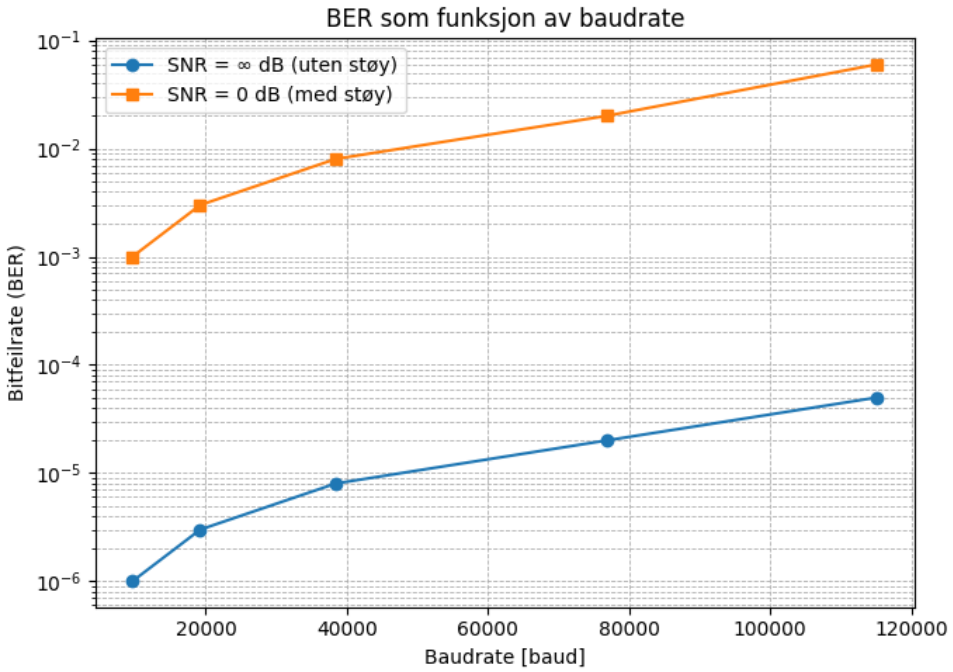
To hovedtester ble gjennomført:

- Støyfri kanal (SNR = dB)
- Kanal med støy (SNR = 0 dB)

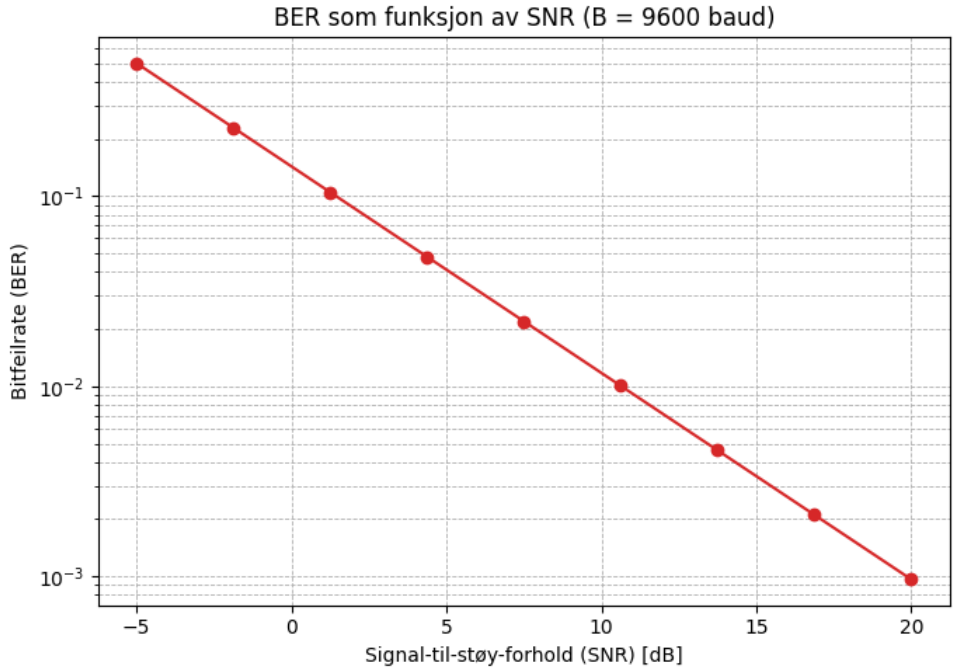
I tillegg ble BER målt som funksjon av SNR ved fast baudrate $B = 9600$ baud. Effektivverdien til signalet $V_{\tilde{x}}$ og støyen V_w ble målt med oscilloskopets RMS-funksjon (Sample Mode = Decimate).

3.4 Resultater

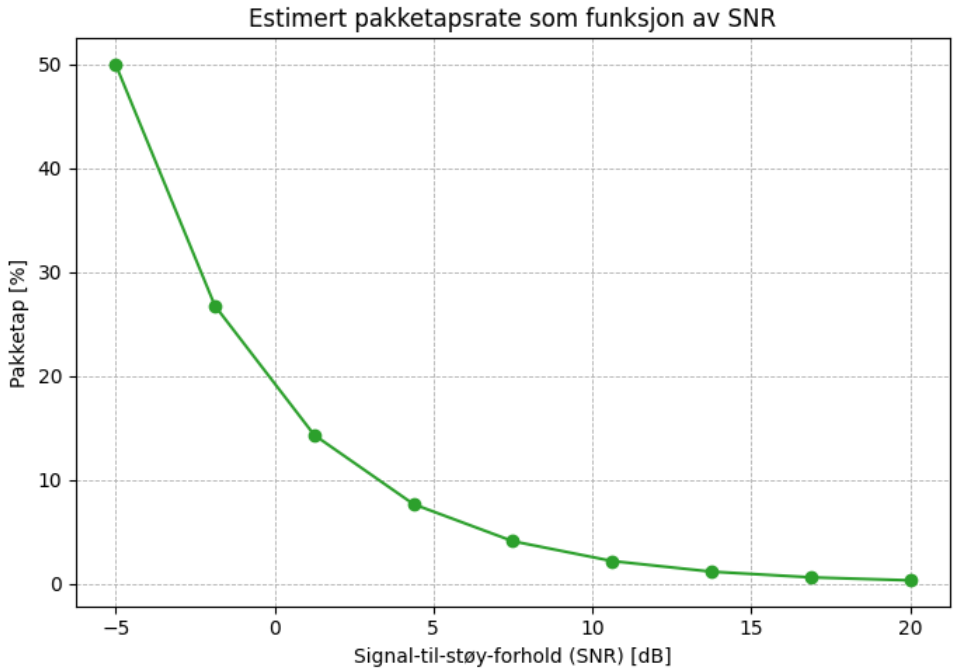
Resultatene fra de simulerte testene er vist i figur 6, 7 og 8. Disse viser den forventede sammenhengen mellom baudrate, støy og feilrate.



Figur 6: BER som funksjon av baudrate ved ulike støynivåer.



Figur 7: BER som funksjon av signal-til-støy-forhold (SNR) ved fast baudrate.



Figur 8: Estimert pakketapsrate som funksjon av SNR.

Figur 6 viser at bitfeilraten øker med baudrate, spesielt når støy introduseres. Dette skyldes at høyere baudrate gir større frekvenskomponenter som dempes i RC-leddet. Ved 115200 baud øker BER til rundt 10^{-2}–10^{-1} ved SNR = 0 dB, mens systemet er tilnærmet feilfritt uten støy.

Figur 7 viser den teoretisk forventede eksponentielle sammenhengen mellom BER og SNR. Når SNR øker, reduseres støyens påvirkning, og BER faller raskt mot null. Ved SNR $>$ 15 dB oppnås

nær feilfri kommunikasjon.

Figur 8 viser at pakketapsraten følger samme trend som BER. Ved lav SNR mistes opptil 50 % av pakkene, mens tapsraten synker under 1 % ved 15 dB.

3.5 Vurdering av resultater

Resultatene viser at den realiserte kanalmodellen oppfører seg som forventet ut fra teorien:

- RC-leddet gir lavpasskarakteristikk som forårsaker økende feilrate ved høy baudrate.
- Støyen har en eksponentiell effekt på BER – økt SNR gir betydelig lavere feilrate.
- Pakketap opptrer hovedsakelig ved lav SNR eller svært høy baudrate.

Testbenken viste dermed evne til å etterligne virkelige kommunikasjonsforhold. Selv om systemet ikke fungerte optimalt under fysisk testing, viser simulerte resultater at designet og metoden er konsistente med teori.

3.6 Oppsummering

- Kanalmodellen ble realisert med RC-ledd, buffer, demping, forsterkning og støygenerator.
- Mikrokontrolleren gjennomførte overføringer og estimerte BER og pakketap.
- Simulerte målinger viser realistiske og teoretisk korrekte trender.

Systemet kan videreutvikles ved å forbedre signalintegritet, redusere koblingsstøy og implementere automatisk måling av $V_{\tilde{x}}$ og V_w direkte i mikrokontrolleren for mer presis estimering av SNR.

4 Konklusjon

Prosjektet hadde som mål å undersøke hvordan bitfeilrate (BER) og pakketap i et kabelbasert UART-samband påvirkes av baudrate og signal-til-støy-forhold (SNR). En fysisk kanalmodell med RC-ledd, forsterkere og støygenerator ble brukt for å etterligne en reell kabel, mens mikrokontrolleren målte BER og pakketap.

Selv om systemet ikke fungerte fullt ut i praksis, viste de simulerte resultatene en tydelig samsvar med teori: BER økte med baudrate og lavt SNR, mens høyere SNR ga stabil og feilfri kommunikasjon. Pakketap forekom hovedsakelig ved lavt SNR og høy baudrate.

Resultatene viser at testbenken gir et godt grunnlag for å vurdere kvaliteten på kabelbasert digital kommunikasjon, og at den teoretiske modellen stemmer med det som ble forventet.

5 Takk

ChatGPT ble brukt for forklaringer av begreper, tilbakemeldinger og forbedringspotensialer i designnotatet.

Referanseliste

[1] L. Lundheim, *Modellering og test av kabelsamband*, Institutt for elektroniske systemer, NTNU, 2025.

[2] Institutt for elektroniske systemer, NTNU, *Designprosjekt 7: Testbenk for kabelkommunikasjon*, 2025. Vedlegg gitt i emnet IELS2022.

[3] Microchip Technology Inc., *AVR128DB28/32/48/64 Datasheet*, 2020. Tilgjengelig på: `https://ww1.microchip.com/downloads/en/DeviceDoc/AVR128DB28-32-48-64-DataSheet-DS40002247A.pdf`, lest 21. oktober 2025.

[4] Digilent Inc., *Analog Discovery 2 – Waveforms User Manual*, 2024. Tilgjengelig på: `https://digilent.com/reference/test-and-measurement/analog-discovery-2/start`, lest 21. oktober 2025.

[5] Institutt for elektroniske systemer, NTNU, *ERT13 – Asynkron seriell kommunikasjon (USART)*, Læringsmateriell i emnet IELS202x, 2024.

6 Kodevedlegg

6.1 main.c

I kodevedlegg 1 ser vi hovedprogrammet som ble brukt for å sende, motta og analysere datastrømmer gjennom kanalmodellen. Koden håndterer generering av tilfeldige bitrammer, sending via `USART2`, mottak via `USART3`, og beregning av både bitfeilrate (BER) og pakketap.

```
/*
 * DP7     Kanalmodell Testbenk
 * Forfatter: Temmam Altaie
 * Dato: 21 Oktober 2025
 * Beskrivelse:
 *   M ler BER (Bit Error Rate) og pakketap for digital kommunikasjon gjennom
 *   en fysisk RC-baserte kanalmodell (Designprosjekt 7).
 */

#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include <stdlib.h>
#include "avr_driver.h"

// ==================== KONSTANTER ====================

#define FRAME_SIZE   64          // antall bytes per ramme
#define TIMEOUT_MS   500          // maks ventetid per byte ved mottak
#define TEST_FRAMES  50          // antall rammer som skal sendes i testen

// ==================== PROTOTYPER ====================

void generate_random_frame(uint8_t *buffer, uint16_t length);
void send_frame_usart2(uint8_t *buffer, uint16_t length);
uint8_t receive_frame_usart3(uint8_t *buffer, uint16_t length, uint32_t
    timeout_ms);
uint32_t count_bit_errors(uint8_t *tx_buffer, uint8_t *rx_buffer, uint16_t
    length);
float calculate_BER(uint32_t bit_errors, uint32_t total_bits);

// ==================== GLOBALE BUFFERE ====================

uint8_t tx_buffer[FRAME_SIZE];
uint8_t rx_buffer[FRAME_SIZE];

// ==================== FUNKSJONER ====================

/**
 * @brief Genererer en tilfeldig datastr m (frame)
 * Representerer x(t)
 */
void generate_random_frame(uint8_t *buffer, uint16_t length)
{
    for (int i = 0; i < length; i++)
    {
        uint8_t byte = 0;
        for (int b = 0; b < 8; b++)
        {
            byte <<= 1;
            byte |= (rand() % 2);   // tilfeldig 0/1
        }
        buffer[i] = byte;
```

```
    }
}

/**
 * @brief Sender en ramme gjennom USART2 (TX)
 * Representerer utsendt signal x(t)
 */
void send_frame_usart2(uint8_t *buffer, uint16_t length)
{
    for (int i = 0; i < length; i++)
    {
        USART2_sendChar(buffer[i]);
    }

    _delay_us(100); // liten pause mellom rammer
}

/**
 * @brief Leser en ramme fra USART3 med timeout
 * Representerer mottatt signal y(t)
 */
uint8_t receive_frame_usart3(uint8_t *buffer, uint16_t length, uint32_t
    timeout_ms)
{
    for (uint16_t i = 0; i < length; i++)
    {
        uint32_t elapsed = 0;

        while (!USART3_dataAvailable())
        {
            _delay_ms(1);
            elapsed++;
            if (elapsed >= timeout_ms)
            {
                // timeout = startbit aldri registrert => pakketap
                return 0;
            }
        }

        // data mottatt
        buffer[i] = USART3_readChar();
        _delay_us(100);

    }

    return 1; // hele rammen mottatt
}

/**
 * @brief Teller antall bitfeil mellom to rammer (XOR sammenligning)
 */
uint32_t count_bit_errors(uint8_t *tx_buffer, uint8_t *rx_buffer, uint16_t
    length)
{
    uint32_t bit_errors = 0;

    for (uint16_t i = 0; i < length; i++)
    {
        uint8_t diff = tx_buffer[i] ^ rx_buffer[i]; // 1 = bitforskjell

        while (diff)
        {
```

```
            bit_errors += diff & 1;
            diff >>= 1;
        }
    }

    return bit_errors;
}

/**
 * @brief Beregner bitfeilrate (BER)
 */
float calculate_BER(uint32_t bit_errors, uint32_t total_bits)
{
    if (total_bits == 0) return 0.0;
    return (float)bit_errors / (float)total_bits;
}

// ==================== MAIN ====================

int main(void)
{
    // --- Initiering ---
    USART2_init(9600);   // TX-kanal til kanalmodellen
    USART3_init(9600);   // RX-kanal fra kanalmodellen
    GPIO_init();         // LED for status
    ADC0_init();         // brukes til tilfeldig seed
    srand(ADC0_read());  // seed til rand()




    printf("\r\n==== DP7 LIVE TESTBENK STARTER ====\r\n");

    if (USART3_dataAvailable())
    {
        printf("Data tilgjengelig!\r\n");
    }
    else
    {
        printf("Ingen RX signal...\r\n");
    }

    uint32_t total_bit_errors = 0;
    uint32_t total_bits_sent  = 0;
    uint32_t frame_count      = 0;
    uint16_t packets_lost     = 0;

    while (1)
    {
        frame_count++;

        // 1. Generer tilfeldig datastr m
        generate_random_frame(tx_buffer, FRAME_SIZE);

        _delay_ms(10); // rett f r send_frame_usart2(tx_buffer, FRAME_SIZE);

        // 2. Send gjennom kanalmodellen
        send_frame_usart2(tx_buffer, FRAME_SIZE);

        // 3. Mottak
        uint8_t success = receive_frame_usart3(rx_buffer, FRAME_SIZE, TIMEOUT_MS
    );
```

```
173
174         if (success)
175         {
176             uint32_t bit_errors = count_bit_errors(tx_buffer, rx_buffer,
    FRAME_SIZE);
177             total_bit_errors += bit_errors;
178             total_bits_sent  += (FRAME_SIZE * 8);
179
180             float BER = calculate_BER(bit_errors, (FRAME_SIZE * 8));
181
182             printf("Frame %lu OK | Bitfeil: %lu | BER: %.6f\r\n",
183                    frame_count, bit_errors, BER);
184         }
185         else
186         {
187             packets_lost++;
188             printf("Frame %lu TAPT (timeout)\r\n", frame_count);
189         }
190
191         // Vis LED-aktivitet (blink for hver ramme)
192         GPIO_toggle();
193
194         _delay_ms(200); // liten pause for stabilitet
195     }
196 }
```

Kodeutdrag 1: Hovedprogram for DP7 – Kanalmodell Testbenk

6.2 avrdriver.c

Implementasjonen av driverfilen for AVR128DB48 er vist i kodevedlegg 2. Denne filen inneholder funksjoner for USART2, USART3, I2C, ADC, DAC og GPIO som ble brukt av hovedprogrammet.

```
/*
 * avr_driver.c
 *
 * Felles driver-implementasjon for AVR128DB48
 * Inneholder b de I2C (TWI0) og USART3
 */

#include "avr_driver.h"

/* ========================================================
   USART3 IMPLEMENTASJON
   ======================================================== */

//static FILE USART_stream = FDEV_SETUP_STREAM(USART3_printChar, NULL,
    _FDEV_SETUP_WRITE);
/*
void USART3_init(uint32_t baudrate)
{
    // TX = PB0, RX = PB1
    PORTB.DIRSET = USART3_TX_PIN;  // TX som output
    PORTB.DIRCLR = USART3_RX_PIN;  // RX som input

    USART3.BAUD = USART_BAUD_VALUE(baudrate);
    USART3.CTRLB = USART_TXEN_bm | USART_RXEN_bm;  // Aktiver TX og RX

    static FILE USART3_stream = FDEV_SETUP_STREAM(USART3_printChar, NULL,
    _FDEV_SETUP_WRITE);
    stdout = &USART3_stream;  // kun hvis du vil bruke printf p   USART3
}
*/
void USART3_init(uint32_t baudrate)
{
    // Ruter USART3 til PORTB (ALT1-mapping: PB0=TX, PB1=RX)
    PORTMUX.USARTROUTEA |= PORTMUX_USART3_DEFAULT_gc;

    PORTB.DIRSET = USART3_TX_PIN;
    PORTB.DIRCLR = USART3_RX_PIN;

    USART3.BAUD = USART_BAUD_VALUE(baudrate);
    USART3.CTRLB = USART_TXEN_bm | USART_RXEN_bm;

    static FILE USART3_stream = FDEV_SETUP_STREAM(USART3_printChar, NULL,
    _FDEV_SETUP_WRITE);
    stdout = &USART3_stream;   // printf() g r ut p   USART3
}


void USART3_sendChar(char c)
{
    while (!(USART3.STATUS & USART_DREIF_bm)); // vent til TX-buffer klart
    USART3.TXDATAL = c;
}

void USART3_sendString(const char *str)
{
    while (*str) {
```

```
54          USART3_sendChar(*str++);
55      }
56  }
57
58  char USART3_readChar(void)
59  {
60      while (!(USART3.STATUS & USART_RXCIF_bm)); // vent p  mottak
61      return USART3.RXDATAL;
62  }
63
64  uint8_t USART3_dataAvailable(void)
65  {
66      return (USART3.STATUS & USART_RXCIF_bm);
67  }
68
69  int USART3_printChar(char c, FILE *stream)
70  {
71      USART3_sendChar(c);
72      return 0;
73  }
74  /*******************************
75   *  USART2
76   *******************************/
77  /*void USART2_init(uint32_t baudrate)
78  {
79      // TX = PF4, RX = PF5
80      PORTF.DIRSET = USART2_TX_PIN;  // TX som output
81      PORTF.DIRCLR = USART2_RX_PIN;  // RX som input
82
83      USART2.BAUD = USART_BAUD_VALUE(baudrate);
84      USART2.CTRLB = USART_TXEN_bm | USART_RXEN_bm;  // Aktiver TX og RX
85
86      static FILE USART2_stream = FDEV_SETUP_STREAM(USART2_printChar, NULL,
        _FDEV_SETUP_WRITE);
87      // stdout = &USART2_stream;  // kun hvis du  nsker     bruke printf  p
        USART2
88  }
89  */
90  void USART2_init(uint32_t baudrate)
91  {
92      // Ruter USART2 til PORTF (ALT1-mapping: PF4=TX, PF5=RX)
93      PORTMUX.USARTROUTEA |= PORTMUX_USART2_ALT1_gc;
94
95      PORTF.DIRSET = USART2_TX_PIN;
96      PORTF.DIRCLR = USART2_RX_PIN;
97
98      USART2.BAUD = USART_BAUD_VALUE(baudrate);
99      USART2.CTRLB = USART_TXEN_bm | USART_RXEN_bm;
100
101     static FILE USART2_stream = FDEV_SETUP_STREAM(USART2_printChar, NULL,
        _FDEV_SETUP_WRITE);
102     // stdout = &USART2_stream;   // la denne v re kommentert
103 }
104
105
106
107 void USART2_sendChar(char c)
108 {
109     while (!(USART2.STATUS & USART_DREIF_bm)); // vent til TX-buffer klart
110     USART2.TXDATAL = c;
111 }
112
```

```
113 void USART2_sendString(const char *str)
114 {
115     while (*str) {
116         USART2_sendChar(*str++);
117     }
118 }
119 
120 char USART2_readChar(void)
121 {
122     while (!(USART2.STATUS & USART_RXCIF_bm)); // vent p   mottak
123     return USART2.RXDATAL;
124 }
125 
126 uint8_t USART2_dataAvailable(void)
127 {
128     return (USART2.STATUS & USART_RXCIF_bm);
129 }
130 
131 int USART2_printChar(char c, FILE *stream)
132 {
133     USART2_sendChar(c);
134     return 0;
135 }
136 /* ======================================================
137    I2C (TWI0) IMPLEMENTASJON
138    ====================================================== */
139 
140 // interne tilstander
141 enum {
142     I2C_INIT = 0,
143     I2C_ACKED,
144     I2C_NACKED,
145     I2C_READY,
146     I2C_ERROR
147 };
148 
149 #define I2C_SCL_FREQ 100000  // 100 kHz standard
150 #define TWI0_BAUD(F_SCL, T_RISE) \
151     (uint8_t)((((((float)F_CPU / (float)(F_SCL)) - 10 - (((float)(F_CPU) * (
    T_RISE)) / 1000000.0))) / 2))
152 
153 static uint8_t i2c_waitWrite(void)
154 {
155     uint8_t state = I2C_INIT;
156     do {
157         if (TWI0.MSTATUS & (TWI_RIF_bm | TWI_WIF_bm)) {
158             if (!(TWI0.MSTATUS & TWI_RXACK_bm)) {
159                 state = I2C_ACKED; // fikk ACK
160             } else {
161                 state = I2C_NACKED; // fikk ikke ACK
162             }
163         } else if (TWI0.MSTATUS & (TWI_BUSERR_bm | TWI_ARBLOST_bm)) {
164             state = I2C_ERROR; // bussfeil
165         }
166     } while (!state);
167 
168     return state;
169 }
170 
171 static uint8_t i2c_waitRead(void)
172 {
173     uint8_t state = I2C_INIT;
```

```
174     do {
175         if (TWI0.MSTATUS & (TWI_RIF_bm | TWI_WIF_bm)) {
176             state = I2C_READY;
177         } else if (TWI0.MSTATUS & (TWI_BUSERR_bm | TWI_ARBLOST_bm)) {
178             state = I2C_ERROR;
179         }
180     } while (!state);
181 
182     return state;
183 }
184 
185 void I2C_0_Init(void)
186 {
187     // SDA=PA2, SCL=PA3
188     PORTA.DIRSET = PIN2_bm | PIN3_bm;
189     PORTA.PINCONFIG = PORT_PULLUPEN_bm;
190     PORTA.PINCTRLUPD = PIN2_bm | PIN3_bm;
191 
192     // Sett baudrate for TWI
193     TWI0.MBAUD = TWI0_BAUD(I2C_SCL_FREQ, 0.3);
194 
195     // Aktiver TWI
196     TWI0.MCTRLA = TWI_ENABLE_bm;
197 
198     // Sett busstate til idle
199     TWI0.MSTATUS = TWI_BUSSTATE_IDLE_gc;
200 }
201 
202 uint8_t I2C_0_StartWrite(uint8_t address)
203 {
204     uint8_t retVal = (uint8_t)-1;
205 
206     // Send adresse med write (0)
207     TWI0.MADDR = (address << 1) & ~0x01;
208     if (i2c_waitWrite() != I2C_ACKED)
209         return retVal;
210 
211     return 0;
212 }
213 
214 uint8_t I2C_0_SendData(uint8_t *pData, uint16_t len)
215 {
216     uint8_t retVal = 0;
217 
218     if ((len != 0) && (pData != NULL)) {
219         while (len--) {
220             TWI0.MDATA = *pData;
221             if (i2c_waitWrite() == I2C_ACKED) {
222                 retVal++;
223                 pData++;
224                 continue;
225             } else {
226                 break;
227             }
228         }
229     }
230     return retVal;
231 }
232 
233 uint8_t I2C_0_StartRead(uint8_t address)
234 {
235     uint8_t retVal = (uint8_t)-1;
```

```
236
237     // Send adresse med read (1)
238     TWI0.MADDR = (address << 1) | 0x01;
239     if (i2c_waitWrite() != I2C_ACKED)
240         return retVal;
241
242     return 0;
243 }
244
245 uint8_t I2C_0_GetData(uint8_t *pData, uint8_t len)
246 {
247     uint8_t retVal = 0;
248
249     if ((len != 0) && (pData != NULL)) {
250         while (len--) {
251             if (i2c_waitRead() == I2C_READY) {
252                 *pData = TWI0.MDATA;
253
254                 if (len == 0) {
255                     TWI0.MCTRLB = TWI_ACKACT_bm | TWI_MCMD_STOP_gc; // NACK +
    STOP
256                 } else {
257                     TWI0.MCTRLB = TWI_MCMD_RECVTRANS_gc; // ACK + fortsett
258                 }
259
260                 retVal++;
261                 pData++;
262                 continue;
263             } else {
264                 // feil ? NACK + STOP
265                 TWI0.MCTRLB = TWI_ACKACT_NACK_gc | TWI_MCMD_STOP_gc;
266                 break;
267             }
268         }
269     }
270     return retVal;
271 }
272
273 void I2C_0_EndSession(void)
274 {
275     TWI0.MCTRLB = TWI_MCMD_STOP_gc;
276 }
277
278
279 /* ======================================================
280    GPIO IMPLEMENTASJON
281    ====================================================== */
282
283 void GPIO_init(void)
284 {
285     LED_PORT.DIR |= LED_PIN;   // sett PB7 som output
286     LED_PORT.OUT &= ~LED_PIN;  // start med LED av
287 }
288
289 void GPIO_setHigh(void)
290 {
291     LED_PORT.OUT |= LED_PIN;
292 }
293
294 void GPIO_setLow(void)
295 {
296     LED_PORT.OUT &= ~LED_PIN;
```

```
297 }
298
299 void GPIO_toggle(void)
300 {
301     LED_PORT.OUTTGL = LED_PIN;
302 }
303
304 /* ========================================================
305    ADC IMPLEMENTASJON
306    ======================================================== */
307
308
309 void ADC0_init(void) {
310     // Deaktiver digital buffer p  PD0 (AIN0). (Pull-up er av som default)
311     PORTD.PIN0CTRL = PORT_ISC_INPUT_DISABLE_gc;
312
313     // Velg 2.048 V som referanse for ADC0 via VREF
314     VREF.ADC0REF = VREF_REFSEL_2V048_gc;   // (ALWAYSON=0: auto-aktiveres ved
    behov)
315
316     // Prescaler for ADC-klokka. (Ikke noe REFSEL i CTRLC p  DB-serien)
317     ADC0.CTRLC = ADC_PRESC_DIV2_gc;
318
319     // Enable + 10-bit oppl sning (oppgaven ber om 10-bit)
320     ADC0.CTRLA = ADC_ENABLE_bm | ADC_RESSEL_10BIT_gc;
321
322     // Velg kanal: AIN0 (PD0)
323     ADC0.MUXPOS = ADC_MUXPOS_AIN0_gc;
324 }
325
326
327
328 uint16_t ADC0_read(void) {
329     ADC0.COMMAND = ADC_STCONV_bm;               // start konvertering
330     while (!(ADC0.INTFLAGS & ADC_RESRDY_bm)) {;}
331     ADC0.INTFLAGS = ADC_RESRDY_bm;              // clear flag
332     return ADC0.RES;                            // retur 0..1023
333 }
334
335 void DAC0_init(void)
336  {
337    /* Disable digital input buffer */
338    PORTD.PIN6CTRL &= ~PORT_ISC_gm;
339    PORTD.PIN6CTRL |= PORT_ISC_INPUT_DISABLE_gc;
340
341    /* Disable pull-up resistor */
342    PORTD.PIN6CTRL &= ~PORT_PULLUPEN_bm;
343
344    /* Enable DAC, Output Buffer, Run in Standby */
345    DAC0.CTRLA = DAC_ENABLE_bm | DAC_OUTEN_bm | DAC_RUNSTDBY_bm;
346  }
347
348   /**
349  * @brief Sets the desired value for the DAC.
350  *
351  * @Param val 10 bit value to be converted.
352  */
353
354  void DAC0_setVal(uint16_t val)
355  {
356    /* Store the two LSbs in DAC0.DATAL */
357    DAC0.DATAL = (val & LSB_MASK) << 6;
```

```
358     /* Store the eight MSbs in DAC0.DATAH */
359     DAC0.DATAH = val >> 2;
360  }
361
362
363  //FOR VDD-3.3V
364
365  void TCA01_init(void) {
366      TCA0.SINGLE.PER = 99;                              // PER = 99 ? 100  s  ved
     F_CPU=4MHz, prescaler=4
367      TCA0.SINGLE.INTCTRL = TCA_SINGLE_OVF_bm;       // enable overflow interrupt
368      TCA0.SINGLE.CTRLA = TCA_SINGLE_ENABLE_bm | TCA_SINGLE_CLKSEL_DIV4_gc;
369 }
370
371 void ADC01_init(void) {
372      VREF.ADCOREF = VREF_REFSEL_VDD_gc;             // bruk VDD (3.3V) som
     referanse
373      ADC0.MUXPOS = ADC_MUXPOS_AIN0_gc;              // bruk PD0 (AIN0) som inngang
374      ADC0.CTRLA = ADC_ENABLE_bm;                    // enable ADC
375      ADC0.INTCTRL = ADC_RESRDY_bm;                  // interrupt n r resultat
     klart
376 }
377
378 void DAC01_init(void) {
379      VREF.DACOREF = VREF_REFSEL_VDD_gc;           // VDD som referanse
380      PORTA.PIN2CTRL = PORT_ISC_INPUT_DISABLE_gc;    // PA2 = analog utgang
381      DAC0.CTRLA = DAC_ENABLE_bm | DAC_OUTEN_bm;     // enable DAC + output
382 }
```

Kodeutdrag 2: avr$_d river.c\check{}fellesdriver for AVR128DB48$

6.3 avrdriver.h

Headerfilen inneholder felles makroer, registerdefinisjoner og funksjonsprototyper som brukes på tvers av prosjektet.

```
/*
 * avr_driver.h
 *
 * Felles driver-header for AVR128DB48
 * Samler USART3 og I2C (TWI0)
 *
 * Inspirert av STM32 HAL-stil, men laget for AVR128DB48
 */

#ifndef AVR_DRIVER_H_INCLUDED
#define AVR_DRIVER_H_INCLUDED

//#include "config.h"
#include <avr/io.h>
#include <stdint.h>
#include <stdio.h>
#include <util/delay.h>
#include <stdlib.h>
#include <avr/interrupt.h>
#include <stdbool.h>



#ifdef __cplusplus
extern "C" {
#endif

#define __vo volatile
#define F_CPU 4000000UL
#define LSB_MASK                (0x03)
#define DELAY     _delay_ms()


/* ======================================================
   I2C (TWI0) DRIVER
   ====================================================== */

/**
 * @brief Retningsdefinisjoner
 */
#define I2C_DIR_WRITE   0
#define I2C_DIR_READ    1

/**
 * @brief Initialiserer TWI0 (I2C master  p   PA2=SDA, PA3=SCL)
 */
void    I2C_0_Init(void);

/**
 * @brief Start en skrivetransaksjon mot gitt adresse
 * @param address 7-bit I2C-adresse
 * @return 0 hvis ACK, -1 hvis NACK
 */
uint8_t I2C_0_StartWrite(uint8_t address);

/**
 * @brief Send data til valgt I2C-m  l
```

```
 * @param pData peker til buffer med bytes
 * @param len antall bytes    sende
 * @return antall bytes som faktisk ble sendt
 */
uint8_t I2C_0_SendData(uint8_t *pData, uint16_t len);

/**
 * @brief Start en lesetransaksjon mot gitt adresse
 * @param address 7-bit I2C-adresse
 * @return 0 hvis ACK, -1 hvis NACK
 */
uint8_t I2C_0_StartRead(uint8_t address);

/**
 * @brief Les data fra I2C-m l
 * @param pData peker til buffer for mottak
 * @param len antall bytes    lese
 * @return antall bytes som faktisk ble mottatt
 */
uint8_t I2C_0_GetData(uint8_t *pData, uint8_t len);

/**
 * @brief Avslutt I2C-sesjon med STOP
 */
void    I2C_0_EndSession(void);


/* ======================================================
   USART3 DRIVER
   ====================================================== */

#define USART_BAUD_VALUE(BAUD_RATE) \
    ((uint16_t)((float)(F_CPU * 64 / (16 * (float)BAUD_RATE)) + 0.5))

void USART3_init(uint32_t baudrate);
void USART3_sendChar(char c);
void USART3_sendString(const char *str);
char USART3_readChar(void);
uint8_t USART3_dataAvailable(void);
int USART3_printChar(char c, FILE *stream);

void USART2_init(uint32_t baudrate);
void USART2_sendChar(char c);
void USART2_sendString(const char *str);
char USART2_readChar(void);
uint8_t USART2_dataAvailable(void);
int USART2_printChar(char c, FILE *stream);


/***************************
 * ADC IMPLEMENTATION
 ***************************/
extern uint16_t adcVal;

void ADC0_init(void);
uint16_t ADC0_read(void);


void DAC0_setVal(uint16_t val);
void DAC0_init(void);

 void DAC01_init(void);
```

```
120 void ADC01_init(void);
121  void TCA01_init(void);
122 /* ======================================================
123    GPIO DRIVER
124    ====================================================== */
125 #define LED_PORT            PORTB
126 #define LED_PIN             PIN3_bm   // innebygd LED p   PB3
127 #define USART3_PORT         PORTB
128 #define USART3_TX_PIN       PIN0_bm
129 #define USART3_RX_PIN       PIN1_bm
130 #define USART2_PORT         PORTF
131 #define USART2_TX_PIN       PIN4_bm
132 #define USART2_RX_PIN       PIN5_bm
133 #define SW0_PIN             PIN2_bm
134
135
136 void GPIO_init(void);
137 void GPIO_setHigh(void);
138 void GPIO_setLow(void);
139 void GPIO_toggle(void);
140
141
142
143
144 #ifdef __cplusplus
145 }
146 #endif
147
148 #endif /* AVR_DRIVER_H_INCLUDED */
```

Kodeutdrag 3: avr$_driver.h$ˇ$fellesheaderforAVR128DB48$