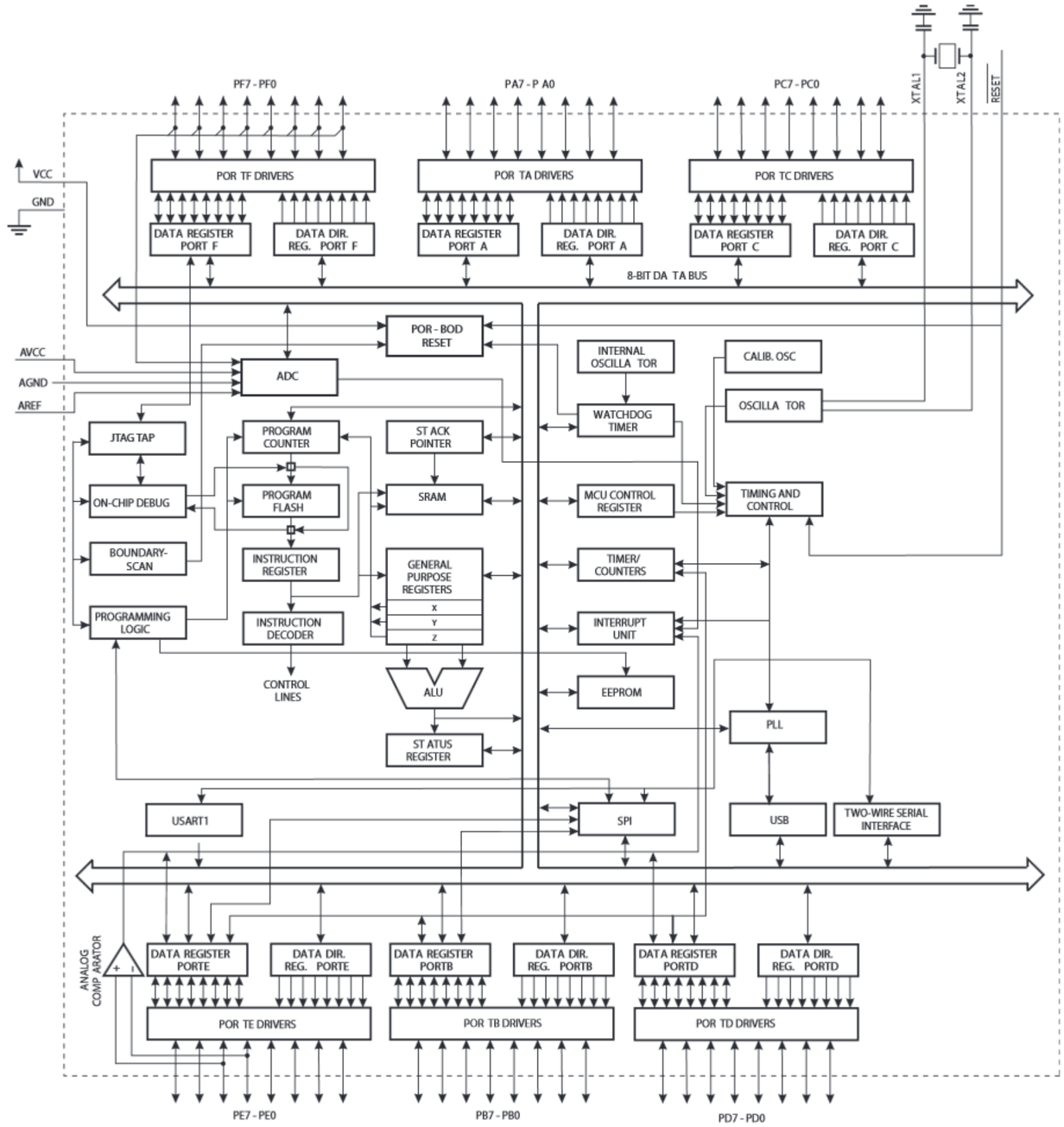
Mikrokontrollerlab

TTK4235

Figure 2-1. Block Diagram



Logistikk

Når:

Uke 14 og 15

Utstyr:

P1000-kort, med Atmel AT90USB1287

JTAGmkII

Div kabler

Utstyr lånes ut kun i labtidene

Oppgaver

Del 1

- Blinkende LED (mikrokontroller-ekvivalent med “Hello world”)
- USART (Skrive til terminal via seriell-kabel)

Del 2

- ADC (Lese input fra joystick)

Del 3

- PWM (styre servo med joystick)

Hele labben godkjennes samlet

Del 1: Bits

Registere I

Vi må kunne lese/skrive data fra/til omverdenen

Mikrokontrolleren har en del innebygd funksjonalitet:

- Timer
- ADC (Analog-til-digital)
- Div. kommunikasjon (USART, SPI, I2C, ..) til andre enheter
- PWM (Pulsbredde-modulasjon)
- GPIO (General-purpose input/output)

Alt dette har konfigurasjonsinstillinger

Registere II

- Vi bruker en 8-bits mikrokontroller:
  - Alle operasjoner skjer med 8 bits om gangen
  - Alle registere er 8 bits
    - I/O-registere
    - Konfigurasjons-registere
      - Hvert bit kan korrespondere til en egen instilling:

USART Control and Status Register n B – UCSRnB

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Hva må man gjøre for å kunne manipulere **kun ett** bit?

Bits og binær

C-syntaks:

`|`	bitvis "eller"	`1001 | 0010 = 1011,  1001 | 1000 = 1001`
`&`	bitvis "og"	`1001 & 1110 = 1000,  1001 & 1101 = 1001`
`<< og >>`	bit-shift	`0010 << 1  = 0100,  1 << 2  = 0100'`
`~`	bitvis "ikke"	`~1001  = 0110`

Foretrekk å bruke heltall med ekspilisitt størrelse:

`uint8_t`, `uint16_t`, `int8_t`, `int16_t`

```
#include <stdint.h>
```

En `int` er 16 bit, en `char` er 8 bit. Hvor stor er en `short`?

Bitvise operasjoner

Sette en variabel til en binær verdi:

```
uint8_t a = 0b00101100; // #include <stdint.h> for int-typer
```

Bitvis og & eller:

```
uint8_t b = a & 0b00001111; // b == 0b00001100
uint8_t c = a | 0b11000000; // c == 0b11101100
```

Bitshift:

```
uint8_t d = (0b11001010 << 3); // d == 0b01010000
```

Manipulere bit nummer `k`:

```
uint8_t e = (1 << k); // Setter bit nr k til 1
UCSR1B = UCSR1B | (1 << TXEN1); // Transmit Enable
UCSR1B |= (1 << TXEN1);
```

Nyttige Makroer

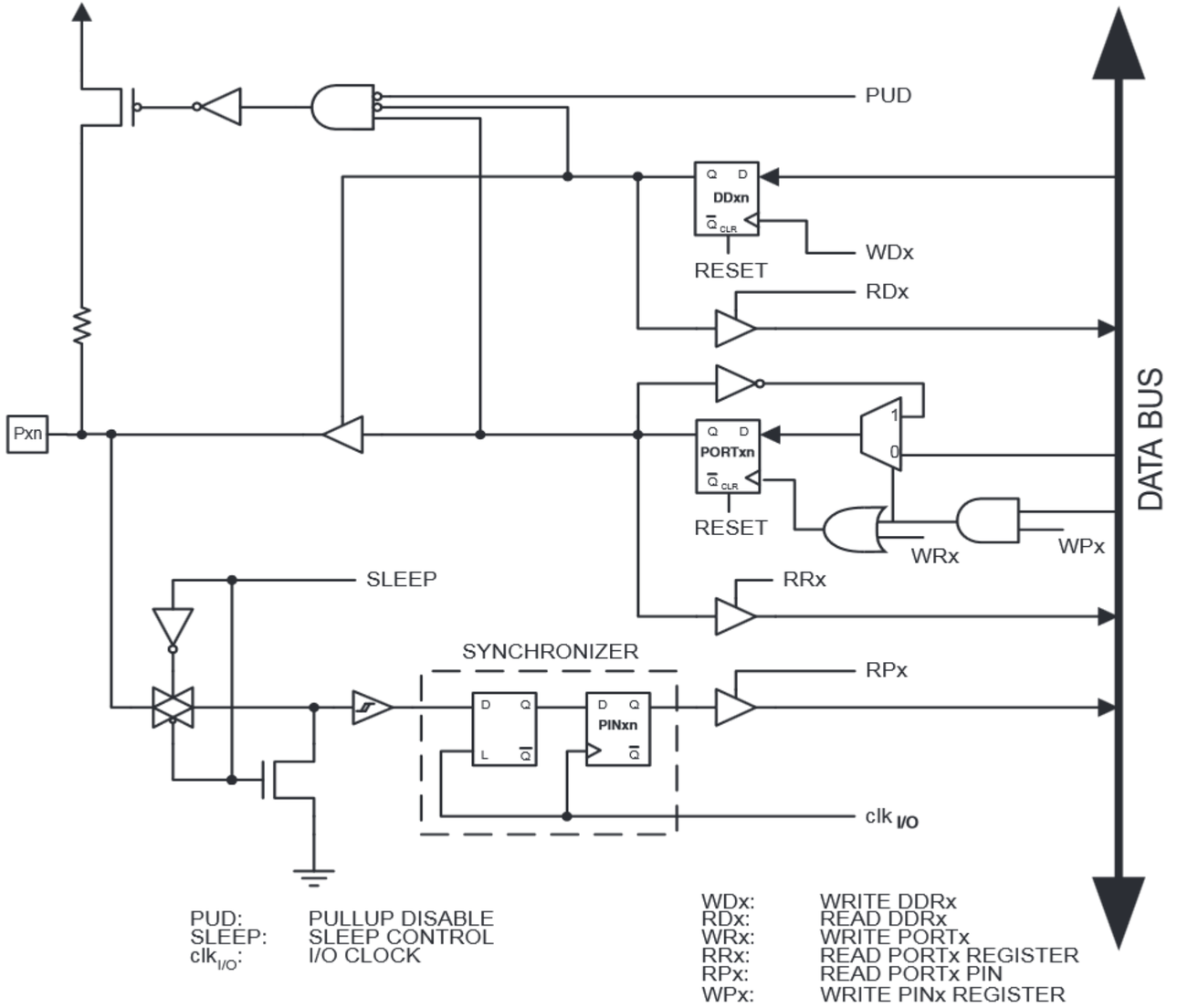
```
#define set_bit(reg, bit) \
  (reg |= (1 << (bit)))

#define clr_bit(reg, bit) \
  (reg &= ~(1 << (bit)))

#define test_bit(reg, bit) \
  (reg & (1 << (bit)))
```

Port vs Pin (side 71 fra databladet)

Figure 10-2. General Digital I/O



Port vs Pin

I/O-pinner kan konfigureres med `DDRx`

(Data Direction Register)

Sett `DDxn` til

- 1 for output
- 0 for input

Eksempel

```
DDRB |= (1<<DDB4);  // Setter pinne B4 til output
DDRB &= ~(1<<DDB5); // Setter pinne B5 til input
```

Port vs Pin

Les verdien på pinnen med `PINx`

Eksempel:

```
uint8_t val = PINB & (1<<PB5);
```

Skriv til pinnen med `PORTx`

Eksempel:

```
if(value){
  PORTB |= (1<<PB4);
} else {
  PORTB &= ~(1<<PB4);
}
```

Del 2: Oppgaver

Hello WorLED

Oppgave:

Les verdien på en bryter, og skriv til korresponderende LED

Mål:

Lære grunnleggende bitmanipulering

Forstå “Data Direction”

Sjekke at hardware fungerer

USART

Oppgave:

- Skriv til skjerm via USART & seriellkabel
- Universal Sync/Async Receiver/Transmitter

Mål:

- Lære å sette opp kontrollregister
- Finne relevant informasjon i datablader
  - Se kapittel 18.9: “USART Register Description”

Asynkron kommunikasjon

- Vi sender informasjon over kun én ledning!
  - Både sender og mottaker må vite hvor "fort" informasjonen sendes
    - Hva skjer hvis de er uenige/går "ut av sync"?
  - Mikrokontrollere har (stort sett) ikke innebygd oscillator
    - Vi må sette opp konfigurasjonsregistere mtp riktig klokkefrekvens
    - Se "USART Baud Rate Register #n": UBRRn
  - En ny transmisjon startes når vi skriver til USART Data Register (UDR)

ADC & joystick

Oppgave:

Les analog verdi på joystick, og konverter til 10-bits digital verdi

Mål:

Forstå forskjellige input-typer til ADC

Free-running vs single conversion, input-multiplekser

Oscilloskop (for debugging)

Interrupts (frivillig)

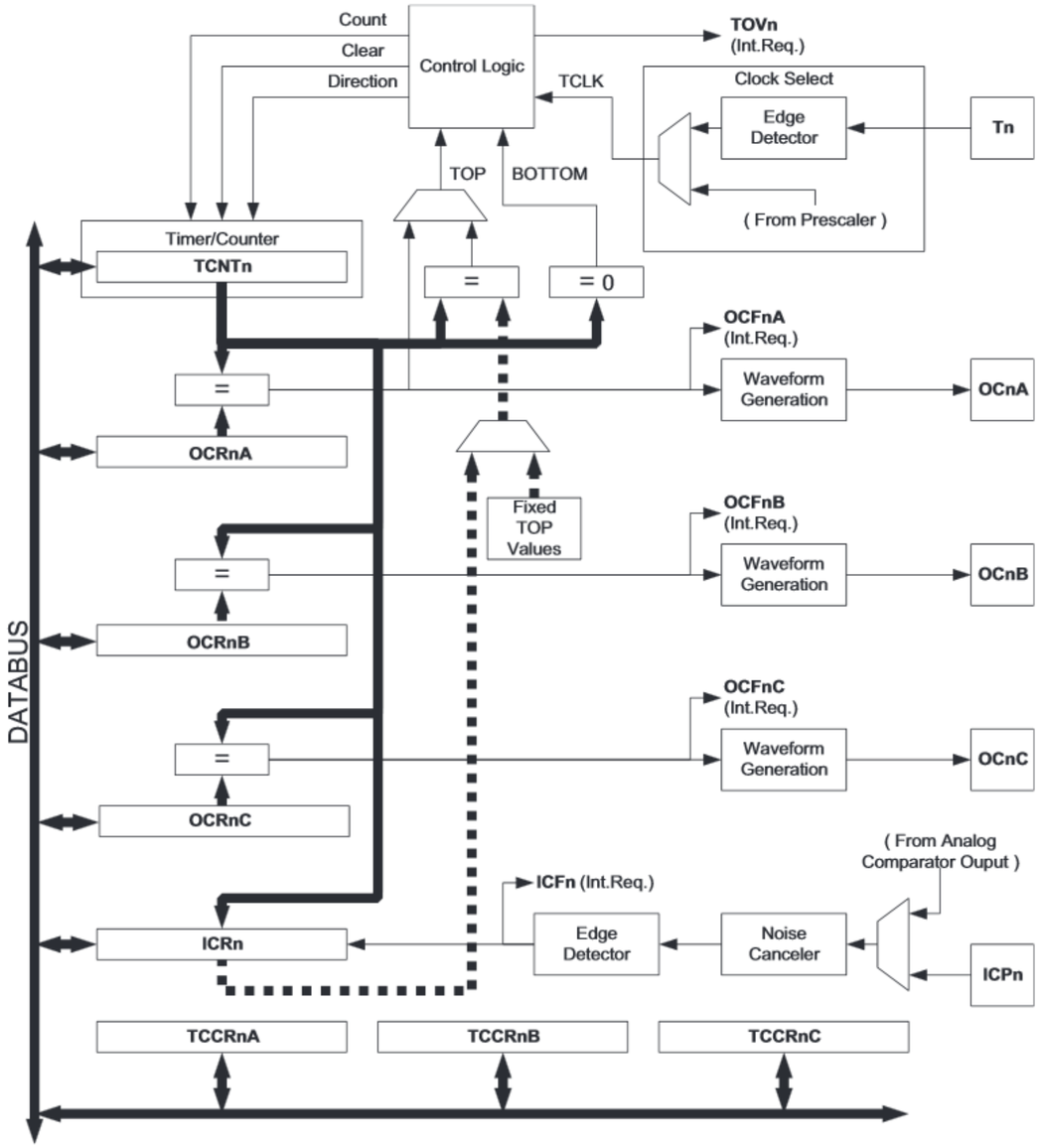
Tips:

Midpunkt på joysticken er ikke nødvendigvis 2.5V.

Lagre midpunkt til joysticken ved å lese ADC’en under initialisering

PWM (side 118 fra databladet)

Figure 14-1. 16-bit Timer/Counter Block Diagram



PWM & servomotor

Oppgave:

Generer et pulsbredde-modulert signal for å styre en servomotor

Mål:

Forstå klokkegenerering og skalering

Mer avansert bruk av oscilloskop

Tips:

Bruk oscilloskop for å måle PWM-signalet før servomotoren kobles til

Prøv å ødelegge færre servoer enn jeg har gjort...