

Indhold

Indledning	2
Talsystemer	3
Den embeddede programmering verden	4
Håndtering af bits i porte (eller variable generelt)	5
Sætte en eller flere bits i en Port (eller variable generelt).....	7
Resette en eller flere bits i en Port (eller variable generelt)	9
Toggle en eller flere bits i en Port (eller variable generelt).....	12
Afrunding	14
Bilag 1.....	15

Indledning

Denne note er blevet til på baggrund af den snak vi havde i klassen fredag d. 01/11-2024 omkring forskellige talsystemer, bitvise operationer og logiske operationer.

Det viste sig her, at der var lidt uklarhed især med hensyn til forskellen på logiske bitvise operationer og logiske heltal operationer. Håbet er, at noten her vil danne grobund for en større forståelse af disse emner, som er meget essentielle, når man beskæftiger sig med især Embedded Programmering.

Det kan være en rigtig god ide for at få det fulde udbytte af dokumentet her, at man starter med at læse (og forstå 😊👍) Bilag 1, før man læser igennem de andre kapitler.

En rigtig god måde at blive klogere på alt det her med logiske operationer og bit manipulation er at bruge den indbyggede Simulator i Microchip Studio. Lav et C program, der kun indeholder de operationer man ønsker at undersøge og så se i Simulatoren, hvad resultaterne bliver. Dette er meget let gjort og der er megen god læring forbundet hermed. Så snyd endelig ikke jer selv for dette 👍👍👍

Talsystemer

Som vi har snakket om på klassen, bruger man indenfor den embeddede verden rigtig meget det binære (2 tals systemet) og det hexadecimale (16 tals systemet) talsystem.

Grunden til at man tager udgangspunkt i det binære talsystem er, at indenfor den digitale verden findes der "kun" 2 værdier nemlig 0 og 1. Rent elektrisk svarer 0 til 0 volt, mens 1 svarer til 5 volt. Så altså er der spænding (5 volt) på en ledning eller ej (0 volt). Og det binære talsystem har jo kun de 2 cifre 0 og 1 at arbejde med, så det giver sig selv, at vi tager udgangspunkt i det binære talsystem.

Og grunden til at man så efterfølgende har taget det hexadecimale talsystem i brug også indenfor den embeddede verden, er den, at $2^4 = 16$ så man kan dermed koble 4 bits fra det binære talsystem sammen til ét ciffer fra det hexadecimale talsystem. Og dermed sparer man 75% af sit skrivearbejde/tastearbejde, hvis man har med store/lange tal at gøre. Dette kan ses i Tabel 1 herunder.

Binær	Decimal	Hexadecimal
0	0	0
1	1	1
10	2	2
11	3	3
100	4	4
101	5	5
110	6	6
111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F
1 0000	16	10
1 0001	17	11
1 0010	18	12
1 0011	19	13
1 0100	20	14
1111 1111	255	0xFF
1 0000 0000	256	0x100
1111 1111 1111 1111	65535	0xFFFF
1 0000 0000 0000 0000	65536	0x10000

Tabel 1: Oversigt over talsystemer

Den embeddede programmering verden

Den embeddede programmerings verden drejer sig i bud og grund om, at man skal styre nogle bits i nogle porte. Begrebet porte dækker her over, at enhver embedded mikroprocessor har et antal porte, der udgør mikroprocessorens interface til omverdenen. Antallet af porte og bredden af de enkelte porte er bestemt af den pågældende mikroprocessor. For nogle mikroprocessorer vil antal bits i de enkelte porte være på 8bits, mens det for andre gælder at antal bits i de enkelte porte vil være på 16bits, og så fremdeles for 32 bit port systemer og 64 bit port systemer. For at gøre det hele lettere vil jeg her i dokumentet "kun" beskæftige mig med mikroprocessorer, hvor antallet af bits i de enkelte porte er på 8. Men principperne er fuldstændig ens på andre mikroprocessorer, hvor antallet af bits i de enkelte porte er noget andet end 8.

Når man ønsker at styre bits i en port, skal man ALTID bruge logiske operationer på bit-niveau og ikke anvende logiske operationer på heltal-niveau for at få det ønskede resultat. Se i Bilag 1 for en nærmere beskrivelse af forskellen på logiske operationer på bit-niveau og logiske operationer på heltal-niveau. En af de største grunde til fejl indenfor port manipulation i den embeddede programmerings verden er, at man som programmør kommer til at anvende logiske operationer på heltals-niveau og ikke på bit-niveau. Det har jeg selv set mange gange, når jeg skulle vejlede yngre SW-udviklere indenfor den embeddede SW udviklings verden.

Heldigvis kan man "kun" til at begå denne fejltagelse, når man laver Or og And operationer, da der ikke er syntaks-mæssig understøttelse for at f.eks. at lave en Exclusive Or operation på heltals-niveau.

Håndtering af bits i porte (eller variable generelt)

Lad os se på et konkret eksempel. Vi ønsker **ved programstart** at sætte bits i en port, så alle ”ulige” bits i porten er 1, mens de lige bits i porten er 0. Det vil altså sige, at der er en spænding på 5v på de ulige bits positioner, mens der er en spænding på 0v på de lige bits positioner. Har man f.eks. koblet 8 lysdioder (LED's) til alle bits i den pågældende port, vil lysdioderne koblet til de ulige bits positioner således lyse, mens lysdioder koblet til de lige bits positioner ikke vil lyse.





Den port vi snakker om her kunne f.eks. være Port B i vores AtMega328PB mikro, som vi ved ligger på adresse 0x0025, hvilket kan ses i Figur 1 herunder.

35. Register Summary

Offset	Name	Bit Pos.								
0x23	PINB	7:0	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x24	DDRB	7:0	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
0x25	PORTB	7:0	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x26	PINC	7:0		PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x27	DDRC	7:0		DDRC6	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0
0x28	PORTC	7:0		PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
0x29	PIND	7:0	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x2A	DDRD	7:0	DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0
0x2B	PORTD	7:0	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x2C	PINE	7:0					PINE3	PINE2	PINE1	PINE0
0x2D	DDRE	7:0					DDRE3	DDRE2	DDRE1	DDRE0
0x2E	PORTE	7:0					PORTE3	PORTE2	PORTE1	PORTE0
0x2F	Reserved									
...										
0x34										
0x35	TIFR0	7:0						OCF0B	OCF0A	TOV0
0x36	TIFR1	7:0			ICF1			OCF1B	OCF1A	TOV1
0x37	TIFR2	7:0						OCF2B	OCF2A	TOV2
0x38	TIFR3	7:0			ICF3			OCF3B	OCF3A	TOV3
0x39	TIFR4	7:0			ICF4			OCF4B	OCF4A	TOV4
0x3A	Reserved									
0x3B	PCIFR	7:0					PCIF3	PCIF2	PCIF1	PCIF0
0x3C	EIFR	7:0							INTF1	INTF0
0x3D	EIMSK	7:0							INT1	INT0
0x3E	GPOR0	7:0	GPOR0[7:0]							
0x3F	EEDR	7:0			EEPROM[1:0]	EERIE	EEMPE	EEPE	EERE	
0x40	EEDR	7:0	EEDR[7:0]							
		7:0	EEAR[7:0]							
0x41	EEARL and EEARRH	15:8							EEAR[9:8]	
0x43	GTCCR	7:0	TSM						PSRASY	PSRSYNC

Figur 1: IO Register summary for AtMega328PB mikro.

Da alle porte starter fra bit position 0 til bit position port bredde – 1, altså bit position 0 – bit position 7 for en 8 bit port, kan vi skrive det binære tal, vi skal skrive til vores port som vist i Tabel 2 herunder.

Bit position	7	6	5	4	3	2	1	0
Bit position vægtning - Decimal	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Bit position vægtning – Decimal værdi	128	64	32	16	8	4	2	1
Ønsket bit værdi i Port	1	0	1	0	1	0	1	0
Tilkoblet Lysdiode status								

Tabel 2: Værdi der skal skrives til 8 bit port.

Det er let at se, at hvis vil opnå resultatet vist i Tabel 2, skal vi skrive det binære tal: **0b10101010** til porten. Det er også ret let at finde det hexadecimale tal, der skal skrives til porten. **Hvis vi bruger Tabel 1 og vores viden om, at 4 binære bits giver ét hexadecimalt ciffer**, kommer vi hurtigt frem til, at det er det hexadecimale tal: **0xAA**, der skal skrives til porten.

Vil vi finde det decimale tal, der skal skrives til porten, er det langt mere besværligt. Så er vi nødt til at finde Bit position vægtningen for alle bit positioner, og lægge alle de bit positioner vægtninger, hvor der står 1 i den ønskede bit værdi sammen. Lidt knudret skrevet så lad os vise det i praksis:

Decimal værdi der skal skrives til port = $1 \cdot 128 + 0 \cdot 64 + 1 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1$

⇕

Decimal værdi der skal skrives til port = $128 + 32 + 8 + 2$

⇕

Decimal værdi der skal skrives til port = 170

Man skal være end almindelig godt kørende i forhold til de forskellige talsystemer for at kunne se, at det decimale tal 170 giver det ønskede bitmønster på porten. **Så derfor bruger man yderst sjældent decimale tal i den embeddede verden, i hvert fald når det drejer sig om port manipulation.**

Sætte en eller flere bits i en Port (eller variable generelt)

Lad os nu antage, **at vi efter den Initiele opsætning** også ønsker at sætte bit 0 i Port B. Hvordan gør vi nu dette, så vi kun får sat denne ene bit og ikke ændrer på status på nogle af de andre 7 bits i Port B. **Dette gør vi ved at lave en bitvis Or (|) og altså ikke en heltals Or (|)** mellem den nuværende værdi i Port B og den binære værdi 00000001 (i dette binære tal er der kun bit 0 positionen, der er sat til 1, da det kun er denne bit position, vi ønsker at sætte til 1 og dermed få 5v på den pågældende bit position i porten).










Koden i vores C program vi skal have udført, er som vist herunder, hvor der er vist forskellige syntakser for at opnå det samme resultat:

- PORTB = PORTB | 0b00000001;
- PORTB |= 0b00000001;
- PORTB = PORTB | 0x01;
- PORTB |= 0x01;
- PORTB = PORTB | (1 << 0);
- PORTB |= (1 << 0);

I alle tilfældene vist herover vil bit værdien på port B efterfølgende blive 0b10101011.

Personligt fortrækker jeg en af de 2 sidste syntakser, da de for mig er lettere at læse. Disse syntakser kan måske godt være lidt svære at forstå. Og hvad står der egentlig? Jo der står: Tag værdien på PORTB og lav en bitvis Or af denne med 1 venstre skiftet 0 bit positioner (operatoren << betyder altså venstre skift).

Måske det hele er lettere at forstå og overskue, hvis vi ser på det i tabelform. Dette er vist i Tabel 3 herunder:













Bit position	7	6	5	4	3	2	1	0
Bit position vægtning - Decimal	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Bit position vægtning – Decimal værdi	128	64	32	16	8	4	2	1
Bit værdi i Port før operation	1	0	1	0	1	0	1	0
Tilkoblet Lysdiode status før operation								
Bit mønster der skal laves en bitvis Or () med	0	0	0	0	0	0	0	1
Bit værdi i Port efter operation	1	0	1	0	1	0	1	1
Tilkoblet Lysdiode status efter operation								

Tabel 3: Bit manipulation => sæt 1 bit på Port

Det viste eksempel herover i Tabel 3 med at sætte én bit i en port efterfølgende kan let udvides til at sætte flere bits i en port efterfølgende. Lad os antage, at vi efter at have sat bit 0 i porten, ønsker at sætte bit 4 og bit 6 positionerne i denne. Dette kan vi gøre ved at bruge en af kodelinjerne vist herunder.

- PORTB = PORTB | 0b01010000;
- PORTB |= 0b01010000;
- PORTB = PORTB | 0x50;
- PORTB |= 0x50;
- PORTB = PORTB | ((1 << 6) | (1 << 4));
- PORTB |= (1 << 6) | (1 << 4);

Og vist på tabelform ser det ud som i Tabel 4 herunder:







Bit position	7	6	5	4	3	2	1	0
Bit position vægtning - Decimal	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Bit position vægtning – Decimal værdi	128	64	32	16	8	4	2	1
Bit værdi i Port før operation	1	0	1	0	1	0	1	1
Tilkoblet Lysdiode status før operation								
Bit mønster der skal laves en bitvis Or () med	0	1	0	1	0	0	0	0
Bit værdi i Port efter operation	1	1	1	1	1	0	1	1
Tilkoblet Lysdiode status efter operation								

Tabel 4 : Bit manipulation => sæt 2 bits på port.

Lad os bare lige for "sjov skyld" komme til at lave den før omtalte fejl med at komme til at bruge en Or operation på heltals-niveau (|) og ikke på bit-niveau (|). Vi tager udgangspunkt i Tabel 3, og vi ønsker igen at sætte bit 6 og bit 4 i porten, men vi kommer til at anvende den forkerte kode syntaks. Så vi kommer til at skrive én af kodelinjerne vist herunder:

- `PORTB = PORTB | 0b01010000;`
- `PORTB |= 0b01010000;` (giver compiler fejl 🙄)
- `PORTB = PORTB | 0x50;`
- `PORTB |= 0x50;` (giver compiler fejl 🙄)
- `PORTB = PORTB | ((1 << 6) | (1 << 4));`
- `PORTB |= (1 << 6) | (1 << 4);` (giver compiler fejl 🙄)

Set på tabelform vil resultatet blive som vist i Tabel 5 herunder:

Bit position	7	6	5	4	3	2	1	0
Bit position vægtning - Decimal	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Bit position vægtning – Decimal værdi	128	64	32	16	8	4	2	1
Bit værdi i Port før operation	1	0	1	0	1	0	1	1
Tilkoblet Lysdiode status før operation								
Bit mønster der laves en heltals Or () med	0	1	0	1	0	0	0	0
Bit værdi i Port efter operation	0	0	0	0	0	0	0	1
Tilkoblet Lysdiode status efter operation								

Tabel 5: Bit manipulation => sæt 2 bits på port med fejlresultat.

Vi kan ud fra Tabel 5 konkludere, at vi altid fejlagtig (kun) vil få sat bit 0 i vores port, uanset hvilken/hvilke bit/bits vi ønsker at sætte i porten, når vi fejlagtig bruger en heltals-niveau Or i stedet for en bit-niveau Or !!!

Resette en eller flere bits i en Port (eller variable generelt)

















Nu har vi set på, hvordan man sætter en bit/flere bits i en port. Så er det næste spørgsmål. Hvordan resetter man en eller flere bits i en port. Lad os som udgangspunkt antage, at vi ønsker at resettet bit 0 i Port B.

Dette gør vi ved at lave en bitvis And (&) og altså ikke en heltals And (&&) mellem den nuværende værdi i Port B og den negerede binære værdi af 00000001.

Koden i vores C program vi skal have udført, er som vist herunder, hvor der er vist forskellige syntakser for at opnå det samme resultat:

- `PORTB = PORTB & ~(0b00000001);`
- `PORTB &= ~(0b00000001);`
- `PORTB = PORTB & ~(0x01);`
- `PORTB &= ~(0x01);`
- `PORTB = PORTB & ~(1 << 0);`
- `PORTB &= ~(1 << 0);`

Og vist på tabelform ser det ud som i Tabel 6 herunder, idet vi antager at udgangspunktet har været som vist i Tabel 4:

Bit position	7	6	5	4	3	2	1	0
Bit position vægtning - Decimal	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Bit position vægtning – Decimal værdi	128	64	32	16	8	4	2	1
Bit værdi i Port før operation	1	1	1	1	1	0	1	1
Tilkoblet Lysdiode status før operation								
Bit mønster der skal laves en bitvis And (&) med (efter negering)	1	1	1	1	1	1	1	0
Bit værdi i Port efter operation	1	1	1	1	1	0	1	0
Tilkoblet Lysdiode status efter operation								

Tabel 6: Bit manipulation => reset 1 bit på port.

Grunden til at vi bruger syntaksen med at negere bit mønsteret er, at dette er lettere rent skivemæssigt, da vi som regel kun ønsker at resettet en eller max 2 bits ad gangen i en port. Vi kunne også have valgt **IKKE** at bruge negeringsoperatoren. I givet fald skulle vores kode skrevet ovenover Tabel 6 være som vist herunder:

















- `PORTB = PORTB & (0b11111110);`
- `PORTB &= (0b11111110);`
- `PORTB = PORTB & (0xFE);`
- `PORTB &= (0xFE);`
- `PORTB = PORTB & ((1 << 7) | (1 << 6) | (1 << 5) | (1 << 4) | (1 << 3) | (1 << 2) | (1 << 1));`
- `PORTB &= (1 << 7) | (1 << 6) | (1 << 5) | (1 << 4) | (1 << 3) | (1 << 2) | (1 << 1);`

Hvis man foretrækker denne syntaks, skal man nok ikke kombinere det med venstre skift operator syntaksen, da man så får vildt meget skrivearbejde 🙄

Det viste eksempel herover i Tabel 6 med at resette én bit i en port efterfølgende kan let udvides til at resette flere bits i en port efterfølgende. Lad os antage, at med udgangspunkt i Tabel 6, ønsker at resette bit 4 og bit 6 positionerne i denne. Dette kan vi gøre ved at bruge en af kodelinjerne vist herunder.

- `PORTB = PORTB & ~(0b01010000);`
- `PORTB &= ~(0b01010000);`
- `PORTB = PORTB & ~(0x50);`
- `PORTB &= ~(0x50);`
- `PORTB = PORTB & ~(1 << 6 | 1 << 4);`
- `PORTB &= ~(1 << 6 | 1 << 4);`

Og vist på tabelform ser det ud som i Tabel 7 herunder:

Bit position	7	6	5	4	3	2	1	0
Bit position vægtning - Decimal	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Bit position vægtning – Decimal værdi	128	64	32	16	8	4	2	1
Bit værdi i Port før operation	1	1	1	1	1	0	1	0
Tilkoblet Lysdiode status før operation								
Bit mønster der skal laves en bitvis And (&) med (efter negering)	1	0	1	0	1	1	1	1
Bit værdi i Port efter operation	1	0	1	0	1	0	1	0
Tilkoblet Lysdiode status efter operation								

Tabel 7: Bit manipulation => reset 2 bits på port.

Lad os bare lige for "sjov skyld" komme til at lave den før omtalte fejl med at komme til at bruge en And operation på heltals-niveau (|) og ikke på bit-niveau (&). Vi tager udgangspunkt i Tabel 6, og vi ønsker igen at resette bit 6 og bit 4 i porten, men vi kommer til at anvende den forkerte kode syntaks. Så vi kommer til at skrive én af kodelinjerne vist herunder:

- `PORTB = PORTB && ~(0b01010000);`
- `PORTB &&= ~(0b01010000);` (giver compiler fejl 👍)
- `PORTB = PORTB && ~(0x50);`
- `PORTB &&= ~(0x50);` (giver compiler fejl 👍)
- `PORTB = PORTB && ~(1 << 6 | 1 << 4);`
- `PORTB &= ~(1 << 6 | 1 << 4);` (giver compiler fejl 👍)

Set på tabelform vil resultatet blive som vist i Tabel 8 herunder:

Bit position	7	6	5	4	3	2	1	0
Bit position vægtning - Decimal	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Bit position vægtning – Decimal værdi	128	64	32	16	8	4	2	1
Bit værdi i Port før operation	1	0	1	0	1	0	1	0
Tilkoblet Lysdiode status før operation								
Bit mønster der laves en heltals And (&&) med (efter negering)	0	1	0	1	0	0	0	0
Bit værdi i Port efter operation	0	0	0	0	0	0	0	1
Tilkoblet Lysdiode status efter operation								

Tabel 8: Bit manipulation => sæt 2 bits på port med fejlresultat.

Vi kan ud fra Tabel 8 konkludere, at vi altid fejlagtig (kun) vil have sat bit 0 i vores port, uanset hvilken/hvilke bit/bits vi ønsker at resette i porten, når vi fejlagtig bruger en heltals-niveau And i stedet for en bit-niveau!!!

Toggle en eller flere bits i en Port (eller variable generelt)

Når man toggler med en bit "vender" man bittens værdi. Det vil sige, at hvis bittens værdi før var 1, bliver den efter togglig lig med 0. Og modsat hvis bitten før var 0, bliver den efter togglig lig med 1. Denne egenskab har man mange gange brug indenfor den embeddede programmerings verden. Vi har set eksempler på dette i vores kode med en blinkende lysdiode. Det der får vores lysdiode/lysdiodes til at blinke, er netop, at vi toggler med en eller flere bits i den port, lysdiodesne er koblet op på.

Og hvordan får man en bit i en port (eller en variabel) til at toggle. Det gør man ved at bruge den logiske Exclusive operator (^). I Tabel 11 i Bilag 1 kan vi se sandhedstabellen for Exclusive Or operatoren. Ved at se på denne kan vi forhåbentlig overbevise os selv om, at:

- 1) Hvis en bit position har værdien 0 og vi laver en Exclusive Or med en variabel, der har værdien 1 på denne bit position, så bliver output værdien på denne bit position 1.
- 2) Hvis en bit position har værdien 1 og vi laver en Exclusive Or med en variabel, der har værdien 1 på denne bit position, så bliver output værdien på denne bit position 0.










Med de 2 punkter ovenover in mente kan vi se, at ex Exclusive Or operator er ideel til en toggle funktionalitet.

Lad os igen prøve at se på et eksempel på dette. Vi bruger igen Port B og antager, at vores udgangspunkt er som i Tabel 7. Og vi vil nu toggle med bit 0 i Port B. Rent kodemæssigt kan vi igen gøre det ved brug af forskellige syntakser som vist herefter:

- `PORTB = PORTB ^ 0b00000001;`
- `PORTB ^= 0b00000001;`
- `PORTB = PORTB ^ 0x01;`
- `PORTB ^= 0x01;`
- `PORTB = PORTB ^ (1 << 0);`
- `PORTB ^= 1 << 0;`

Da udgangspunktet for værdien på bit 0 positionen var 0, vil vi efter udførelse af den valgte kodelinje herover få værdien 1 på bit position 0.

Sat op på tabelform kan vi se resultatet herunder i Tabel 9.

Bit position	7	6	5	4	3	2	1	0
Bit position vægtning - Decimal	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Bit position vægtning – Decimal værdi	128	64	32	16	8	4	2	1
Bit værdi i Port før operation	1	0	1	0	1	0	1	0
Tilkoblet Lysdiode status før operation								
Bit mønster der skal laves en bitvis Exclusive Or med	0	0	0	0	0	0	0	1
Bit værdi i Port efter operation	1	0	1	0	1	0	1	1
Tilkoblet Lysdiode status efter operation								











Tabel 9: Bit manipulation => toggle 1 bit på port.

havde udgangspunktet for værdien på bit 0 positionen været 1, vil vi efter udførelse af den valgte kodelinje over Tabel 9 få værdien 0 på bit position 0.

Som tilfælde er med den bitvise Or operation og den bitvise And operation kan vi umiddelbart udvide til at lade det her beskrive virke på flere bits. Lad os antage at vores udgangspunkt er som vist i Tabel 9, og vi nu ønsker at toggle bit 0 og bit 2 i Port B. Dette kan vi gøre med den valgte af kodelinjerne herunder:

- `PORTB = PORTB ^ 0b00000101;`
- `PORTB ^= 0b00000101;`
- `PORTB = PORTB ^ 0x05;`
- `PORTB ^= 0x05;`
- `PORTB = PORTB ^ ((1 << 2) | (1 << 0));`
- `PORTB ^= (1 << 2) | (1 << 0);`

Sat op på tabelform kan vi se resultatet herunder i Tabel 10.

Bit position	7	6	5	4	3	2	1	0
Bit position vægtning - Decimal	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Bit position vægtning – Decimal værdi	128	64	32	16	8	4	2	1
Bit værdi i Port før operation	1	0	1	0	1	0	1	1
Tilkoblet Lysdiode status før operation								
Bit mønster der skal laves en bitvis Exclusive Or med	0	0	0	0	0	1	0	1
Bit værdi i Port efter operation	1	0	1	0	1	1	1	0
Tilkoblet Lysdiode status efter operation								

Tabel 10: Bit manipulation => toggle 2 bits på port.

Afrunding

Det meste af det der er skrevet i dokumentet her, har omhandlet logiske operatorer på bitvis-niveau i forbindelse med port manipulation. Men alt det i dokumentet skrevne kan i lige så høj grad bruges på "almindelige" variable, som ikke nødvendig er knyttet til en port i en mikroprocessor.

Til slut derfor lige en lille sjov opgave som I kan prøve at se, om I kan finde svaret på. Den logiske operator Xor bruges i alle afskygninger af SW-udvikling (ikke kun embedded SW-udvikling) til at foretage en helt specifik operation. En operation der i den grad får ens øjne op for, at de folk der har bestemt den ting, som denne helt specifikke operation virker på, virkelig har tænkt sig rigtig godt om. Kan en eller flere af jer finde ud af, hvilken operation der tænkes på her. Som en lille hjælp kan jeg nævne, at I skal tænke både stort og småt 😊👍

Bilag 1

Sandhedstabellen for de mest almindelige anvendte logiske operationer på bit niveau indenfor den embeddede programmerings verden kan ses i Tabel 11 herunder. Der findes også andre logiske operationer indenfor elektronikverdenen, men disse bruger man ikke så meget indenfor den embeddede programmerings verden. For fuldstændighedens skyld er disse dog også medtaget i Tabel 12 herunder.

Bit 1	Bit 2	Operation	Operation	Operation
		Logisk Or ()	Logisk And (&)	Logisk Exclusive Or (^)
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Tabel 11 : Oversigt over de mest almindelige anvendte logiske operationer på bit niveau indenfor elektronik/den embeddede programmerings verden.

Bit 1	Bit 2	Operation	Operation
		Logisk Nor ~()	Logisk Nand ~(&)
0	0	1	0
0	1	0	0
1	0	0	0
1	1	0	1

Tabel 12 : Oversigt over de "udvidede" logiske operationer på bit niveau indenfor elektronik/den embeddede programmerings verden.

Når man skal lære alt dette her, som jeg beskriver i dokumentet her, kan man godt gå hen og blive en anelse forvirret. **Denne forvirring opstår, fordi der i f.eks. C sproget både kan anvendes logiske operationer på bit niveau for de logiske Or og And operationer samt logiske operationer på "hele tals niveau" for de logiske Or og And operationer.** Eksempler på dette er vist herunder i Tabel 13:

Tal1	Tal2	Tal3 Logisk Or på bit-niveau (Tal1 Tal2)	Tal3 Logisk And på bit-niveau (Tal1 & Tal2)	Tal3 Logisk Or på heltals- niveau (Tal1 Tal2)	Tal3 Logisk And på heltals-niveau (Tal1 && Tal2)	Tal Exclusive Or på bit-niveau (Tal1 ^Tal2)
0xAA (1010 1010)	0x55 (0101 0101)	0xFF (1111 1111)	0x00 (0000 0000)	1	1	0xFF (1111 1111)
0xEA (1110 1010)	0x54 (0101 0100)	0xFE (1111 1110)	0x40 (0100 0000)	1	1	0xFE (1111 1110)
0xEA (1110 1010)	0x01 (0000 0001)	0xEB (1110 1011)	0x00 (0000 0000)	1	1	0xEB (1110 1011)
0xEA (1110 1010)	0x00 (0000 0000)	0xEA (1110 1010)	0x00 (0000 0000)	1	0	0xEA (1110 1010)
0x00 (0000 0000)	0x00 (0000 0000)	0x00 (0000 0000)	0x00 (0000 0000)	0	0	0x00 (0000 0000)

Tabel 13: Eksempler på logiske operationer på både bit-niveau og på heltals-niveau i C sproget.

Ud fra Tabel 13 kan vi opstille et par generelle regler.

- 1) **Exclusive Or operation kan KUN udføres på bit-niveau i C og alle andre sprog.**
- 2) For at en OR operation skal give resultatet 0 skal begge tal/operander være 0. Er dette tilfældet, vil både OR operationen på bit-niveau og på heltals-niveau give 0.
- 3) Hvis bare et af tallene/operanderne har en bit, der ikke er 0, vil begge OR operationer give et resultat, der ikke er 0. Or operationen gældende på heltals-niveau vil altid give resultatet 1.
- 4) Hvis den logiske AND operation på heltals-niveau giver 0, så vil den logiske And operation på bit-niveau også give 0. Det modsatte er IKKE tilfældet. En logisk And operation på bit-niveau kan godt give resultatet 0, uden at resultatet på heltals-niveau giver resultatet 0.
- 5) **Alle logiske operationer der virker på heltals-niveau, kan kun give resultatet 0 eller 1 for Or og And operationer.**

Alle de logiske operationer der hidtil er vist her i Bilag 1, virker alle på to operander. **Udover de logiske operationer, der virker på to operander, er der også en logisk operation, der altid virker på én operand og altid virker på bit-niveau, når vi arbejder med integer variable. Dette er not operatoren (negeringsoperatoren) ~, der virker på én operand og altid opererer på bit-niveau. Not operatoren vender alle bits i et tal. Eksempler på dette kan ses i Tabel 14 herunder.**

Tal	~(Tal)
0xAA (1010 1010)	0x55 (0101 0101)
0x55 (0101 0101)	0xAA (1010 1010)
0x00 (0000 0000)	0xFF (1111 1111)
0xFF (1111 1111)	0x00 (0000 0000)

Tabel 14: Eksempler på Not (Negerings) operatoren.

En rigtig god måde at blive klogere på alt det her med logiske operationer og bit manipulation er at bruge den indbyggede Simulator i Microchip Studio. Lav et C program, der kun indeholder de operationer man ønsker at undersøge og så se i Simulatoren, hvad resultaterne bliver. Dette er meget let gjort og der er megen god læring forbundet hermed. Så snyd endelig ikke jer selv for dette 👍👍👍