Dynamics and Motion control

Lecture 6
Implementation of the controller on real-time hardware.

Bengt Eriksson, Jan Wikander
KTH, Machine Design
Mechatronics Lab
e-mail: benke@md.kth.se

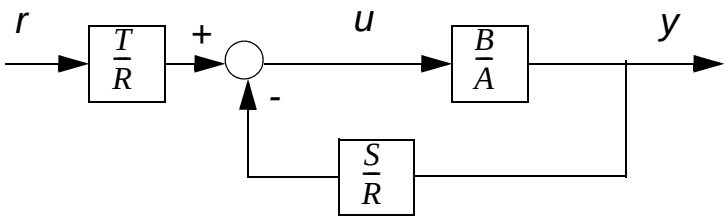
6.1. Lecture outline

- **1. Introduction**
- 2. Timing and computational delay
- 3. Converting to code
- 4. Switching amplifiers

6.1.1 From last lectures

- **A controller is designed either direct in discrete time or a continuous time controller is approximated into a discrete time controller.**
- **The control law is in transfer function form.**

$$u(z) = \frac{T(z)}{R(z)}r - \frac{S(z)}{R(z)}y$$



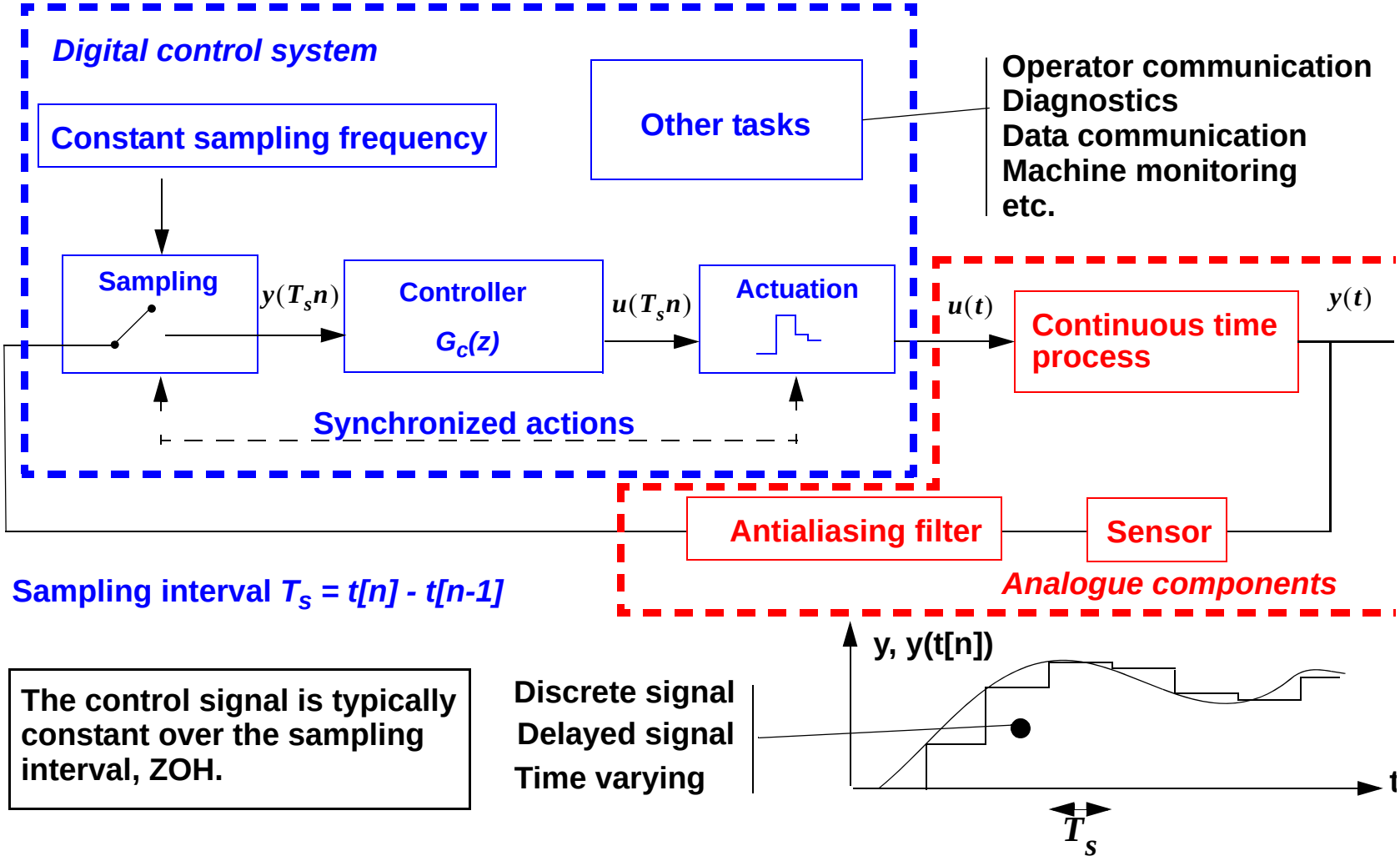
6.1.2 Typical structure of controller code

```
//  Global variabel definitions

/*-------------------------------------*/
/*---- timer interupt -----------------*/
/*-------------------------------------*/

void Task_1ms(void)
{

    //  Local variabel deffinitions

    y = read_sensor();
    uc = get_command_signal();
    u = controller_code(y,uc,x);
    write_u_to_IO(u);
    x = update_state();
}
/*-------------------------------------*/
/*---- main ---------------------------*/
/*-------------------------------------*/
void main(void)
{
    // initialize real-time kernel
    void init_RTK(void)

    // init IO drivers
    void init_IO(void)

    /*-------------------------------*/
    /*---- Background Loop ----------*/
    /*-------------------------------*/
    while( 1 )
    {
    }
}
```

6.2. Lecture outline

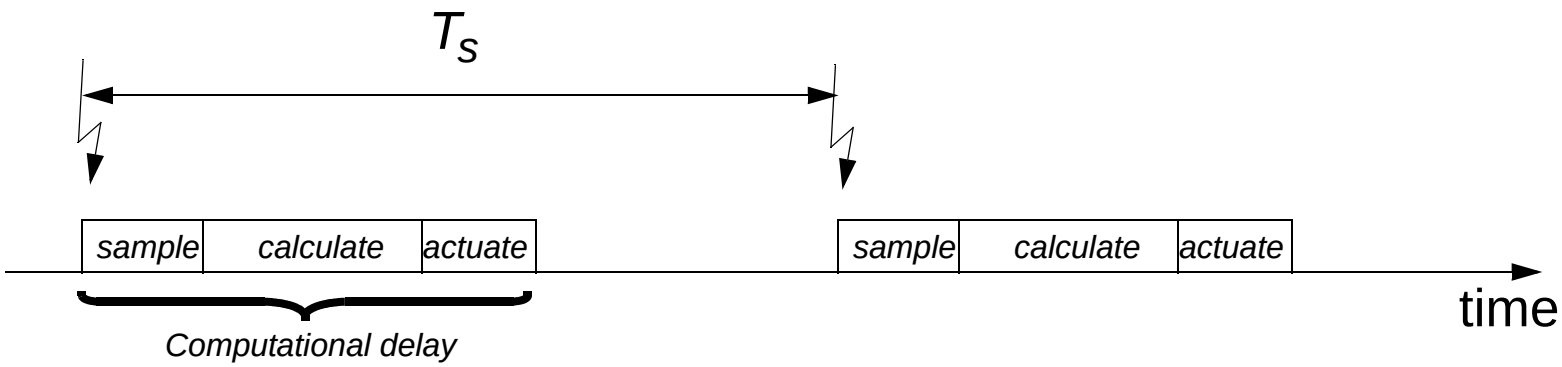
- 1. Introduction
- **2. Timing and computational delay**
- 3. Converting to code
- 4. Switching amplifiers

6.2.1 Discrete time control

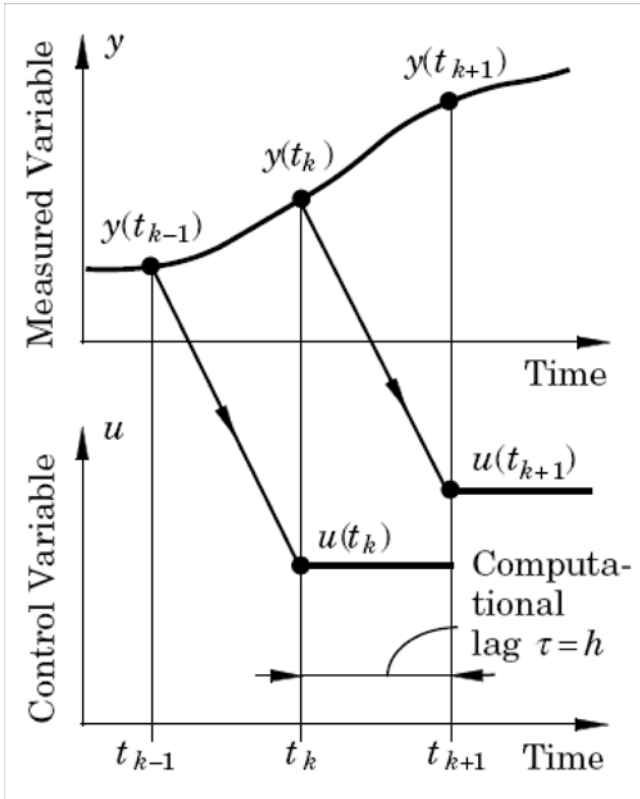


6.2.2 Assumptions/consequences

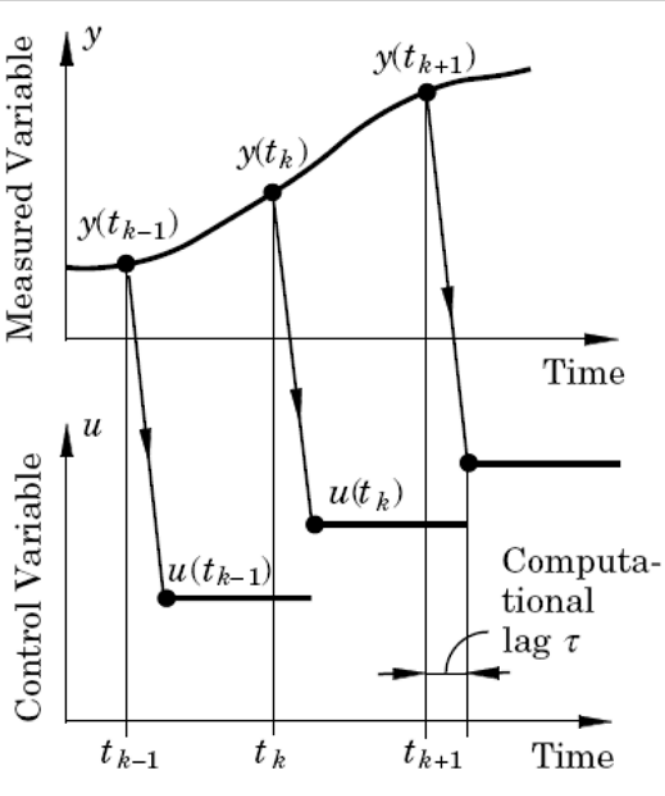
- sampling at constant frequency (constant sampling interval)
- synchronism between sampling and actuation
- zero delay between sampling and actuation (clearly we can not achieve this exactly, execution of the control algorithm takes time)



6.2.3 Synchronizing sampling and actuation



Exactly one sample delay



Shorter but unknown delay.
Important to get it as short as possible

6.2.4 Evaluation of computational delay

- How much the computational delay will effect the performance is very difficult to calculate in advance.
  - The actual delay will depend on the microprocessor, cpu speed, I/O speed, floating or fixed point, etc.
  - The performance decrease will depend on the relation between sampling period and closed loop poles.
- When the controller is implemented is it often easy to measure the delay using a timer.
- The delay can be simulated in Simulink.

6.3. Lecture outline

- 1. Introduction
- 2. Timing and computational delay
- **3. Converting to code**
- 4. Switching amplifiers

6.3.1 Two ways of implementing the controller

- **Implementation with the controller as a transfer function.**

1.) Read the sensor value *y[n]* and perhaps *r[n]*.

2.) calculate the control signal *u[n].*

3.) Write *u[n].*

- **Implementation with the controller as a state space model.**

1.) Read the sensor value *y[n]* and perhaps $u_c[n]$.

2.) calculate the control signal *u[n].*

3.) Write *u[n].*

4.) Update the state of the controller *x[n]*.

6.3.2 Calculating *u* on transfer function form

- $u(z) = \frac{T(z)}{R(z)}r - \frac{S(z)}{R(z)}y$

- $R(z)u = T(z)r - S(z)y$

- **Normally the order, l of $R(z), T(z), S(z)$, is the same**

$$(z^l + r_{l-1}z^{l+1} + \ldots + r_0)u = (t_l z^l + t_{l-1}z^{l-1} + \ldots + t_0)r - (s_l z^l + s_{l-1}z^{l-1} + \ldots + s_0)y$$

multiply both sides with z^{-l}

$$(1 + r_{l-1}z^{-1} + \ldots + r_0 z^{-l})u = (t_l + t_{l-1}z^{-1} + \ldots + t_0 z^{-l})r - (s_l + s_{l-1}z^{-1} + \ldots + s_0 z^{-l})y$$

gives the implementation with, $3l-1$ multiplications and $3l-2$ additions.

$$u[n] = -r_{l-1}u[n-1] - \ldots - r_0 u[n-l] + t_l r[n] + t_{l-1}r[n-1] + \ldots + t_0 r[n-l]$$
$$+ s_l y[n] + s_{l-1}y[n-1] + \ldots + s_0 y[n-l]$$

6.3.3 Converting to s.s. form

- Generally we know that for any:
  - transfer function $G(s) = \frac{B(s)}{A(s)}$ are the poles the roots of the denominator.

    They are not influenced by $B(s)$
  - state space model $\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Bu \end{aligned}$ are the poles the eigenvalues of the *A* matrix. They are not influenced by the *B,C* or *D* matricies.
- Therfor is it possible to write:
  - $u(z) = \frac{T(s)}{R(s)}r - \frac{S(s)}{R(s)}y$, as $\begin{aligned} \dot{x} &= Ax + B_1 r + B_2 y \\ u &= Cx + D_1 r + D_2 y \end{aligned}$. With the same *A* matrix.

    in standard form $\dot{x} = Ax + \begin{bmatrix} B_1 & B_2 \end{bmatrix} \begin{pmatrix} r \\ y \end{pmatrix}, u = Cx + \begin{bmatrix} D_1 & D_2 \end{bmatrix} \begin{pmatrix} r \\ y \end{pmatrix}$

6.3.4 Calculating *u* on state space form

- **From state space form, *y = sensor, u = control, r = reference.***

$$x_c[n+1] = F_c x_c[n] + Gy[n] + G_c r[n]$$
$$u[n] = Cx_c[n] + Dy[n] + D_c r[n]$$

Timer interrupt each Ts second

1.) Read: y[n]
2.) Calculate: u[n] = $Cx_c[n] + Dy[n] + D_c r[n]$;
3.) Write u[n];
4.) Update state: $x_c[n+1] = F_c x_c[n] + Gy[n] + G_c r[n]$;
5.) Save state for next sample

Wait until next timer interrupt

- No matrix operations in C language -> consider using for loops
  The computational delay will depend on the transformation.

6.3.5 Improved version

- Implementation with even less computational delay.

- $x_c[n+1] = F_c x_c[n] + Gy[n] + G_c r[n]$
  $u[n] = Cx_c[n] + Dy[n] + D_c r[n]$

Timer interrupt each Ts second

1.) Read: y[n]
2.) Calculate: u[n] = z[n] + Dy[n] + D_cr[n];
3.) Write u[n];
4.) Update state: x_c[n+1] = $F_c x_c$[n] + Gy[n] + G_cr[n];
and precompute: z[n+1] = Cx_c[n+1];
5.) Save state for next sample
Wait until next timer interrupt

6.3.6 Converting from t.f to s.s.

- $u(z) = \frac{T(z)}{R(z)}r - \frac{S(z)}{R(z)}y$ one output and two inputs.

  $$x_c[n+1] = F_c x_c[n] + Gy[n] + G_c r[n]$$
  $$u[n] = Cx_c[n] + Dy[n] + D_c r[n]$$

- The computational delay depends on how it is converted from t.f. to s.s.
  - Canonical, observable, modal or any other form.
- Numeric problems with some conversions, e.g. multiplying very large numbers with very small.
  - Typically modal form has few computations (many zeros in F_c) but it has bad numeric properties.
- The "ss" command in Matlab usually gives a good compromise.

6.3.7 Symbolic convention

- **Symbolically almost as simple** **(small trick when same order in Tf.)**

Example:

$$\frac{S}{R} = \frac{s_1 z + s_0}{z + r_0} = \frac{s_1 z + r_0 s_1 - r_0 s_1}{z + r_0} + \frac{s_0}{z + r_0} = \frac{s_1 (z + r_0)}{z + r_0} + \frac{s_0 - r_0 s_1}{z + r_0} = s_1 + \frac{s_0 - r_0 s_1}{z + r_0}$$

in state space form:
$$u[n] = x[n] + s_1 y[n]$$
$$x[n+1] = -r_0 x[n] + (s_0 - r_0 s_1) y[n]$$

- the "D" matrix is non zero when the order of *R(z)* and *S(z)* is equal.

6.3.8 Example position control

- **The PID controller for the dc motor is**

$$u(z) = \frac{t_2 z^2 + t_1 z + t_0}{z^2 + r_1 z + r_0} r(z) - \frac{s_2 z^2 + s_1 z + s_0}{z^2 + r_1 z + r_0} y(z)$$

- **from earlier example we have**

$$u(z) = \frac{0.65 z^2 - 0.82 z + 0.27}{z^2 - 1.35 z + 0.35} r(z) - \frac{10.9 z^2 - 20 z + 9}{z^2 - 1.35 z + 0.35} y(z)$$

6.3.9 Implementing the example from above.

Write the control law as: $R(z)u(z) = T(z)r(z) - S(z)y(z)$

$$(z^2 - 1.35z + 0.35)u = (0.65z^2 - 0.82z + 0.27)r - (10.9z^2 - 20z + 9)y$$

$$(1 - 1.35z^{-1} + 0.35z^{-2})u = (0.65 - 0.82z^{-1} + 0.27z^{-2})r - (10.9 - 20z^{-1} + 9z^{-2})y$$

1.) sample: `read_from_input(y);`

2.) calculate:
```
u  = 1.35*u1 - 0.35*u2 +0.65*r - 0.82*r1
       + 0.27*r2 -10.9*y + 20*y1 - 9*y2;
```

3:) write: `write_to_output(u);`

4.) shift : `u2 = u1; u1 = u; uc2 = uc1; uc1 = uc; uc2 = uc1;y2 = y1; y1 = y;`

Computational delay 8 multiplications and 7 additions.
3 local variables: u,uc and y;
6 global (non-volatile) variables, u1,u2,uc1,uc2,y1,y2.

6.3.10 In state space form

$$x_c[n+1] = \begin{bmatrix} 0 & -0.7 \\ 0.5 & 1.34 \end{bmatrix} x_c[n] + \begin{bmatrix} 2.6 \\ -1.28 \end{bmatrix} y[n] + \begin{bmatrix} 0.024 \\ 0.013 \end{bmatrix} r[n]$$

$$u[n] = \begin{bmatrix} 0 & 4 \end{bmatrix} x_c[n] + 10.9y[n] + 0.65r[n]$$

1.) sample `read_from_input(y);`

2.) calculate u: `u = z + 10.9*y + 0.65*r;`

3.) write: `write_to_output(u);`

4.) update state:
```
xc1_next = -0.7*xc2 + 2.6*y + 0.024* r;
xc2_next = 0.5*xc1 + 1.34*xc2 - 1.28*y +0.013*r;
z = 4*xc2_next;
```

5.) shift: `xc1 = xc1_next;  xc2 = xc2_next;`

Computational delay 2 multiplications and 2 additions. (+8 mult and 5 add.)
5 local variables: u,y,uc,xc1_next and xc2_next;
3 global (non-volatile) variables, xc1, xc2 and z.

6.3.11 Summary

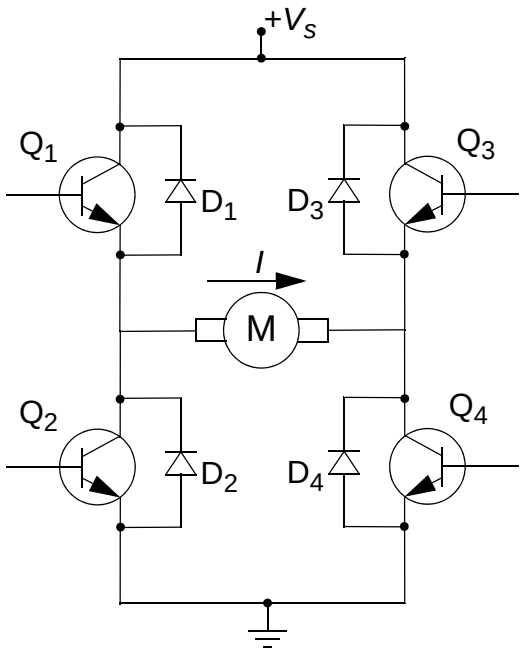
- **If it is necesary with a short computational delay implement the controller in state space form where you first calculate the output and thereafter update the states for next sample.**

6.4. Lecture outline

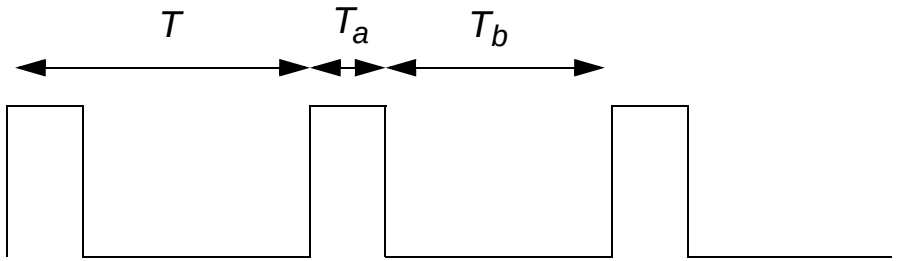
- 1. Introduction
- 2. Timing and computational delay
- 3. Converting to code
- **4. Switching amplifiers**

6.4.1 Switching amplifiers

H bridge



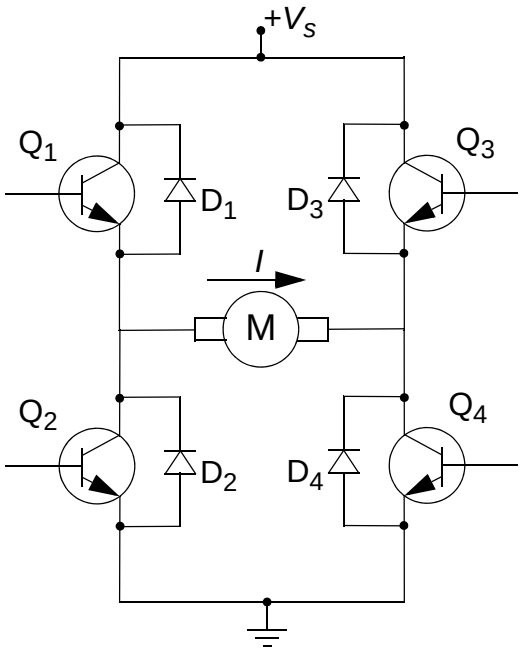
The PWM signal

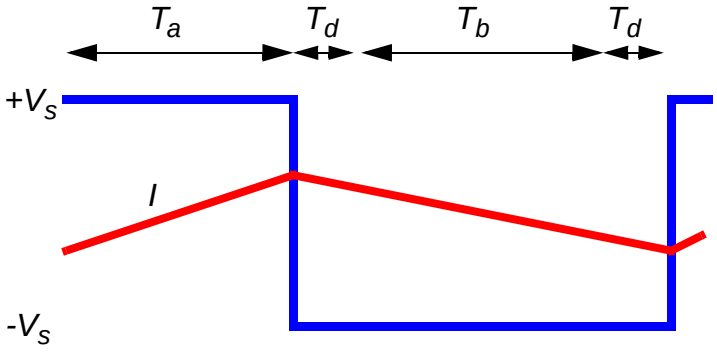


Frequency = $1/T$

Duty cycle = T_a/T

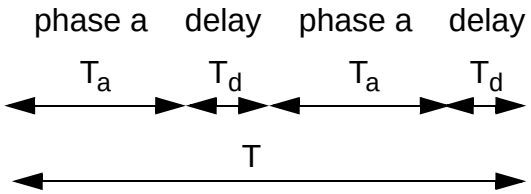
6.4.2 Bipolar drive (fast decay mode)





Voltage and current over the motor for one period

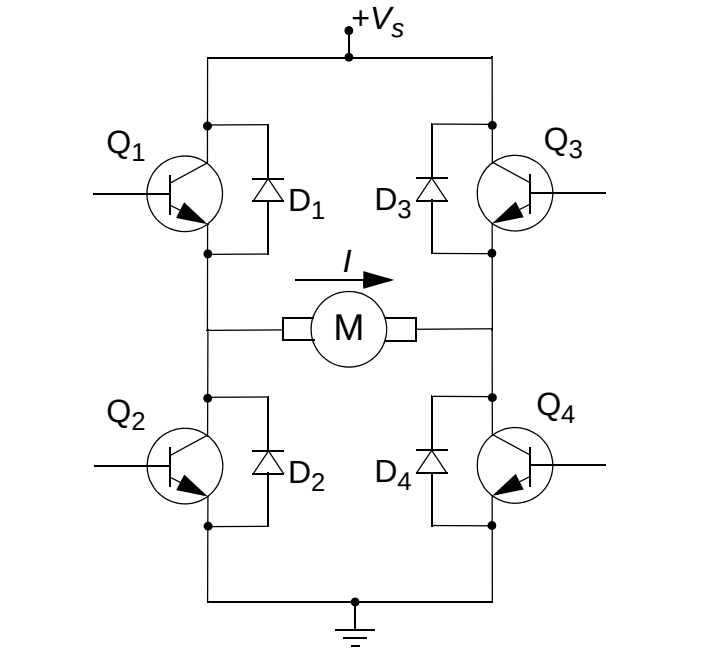
The phases of the switching period

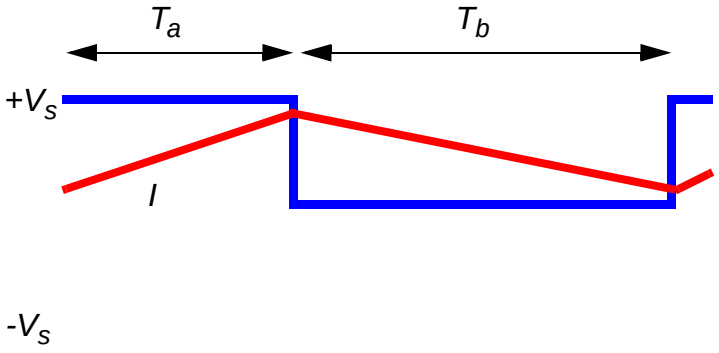


phase a:	Q_1, Q_4	On	$V_m = V_s$
	Q_2, Q_3	Off	
phase b:	Q_2, Q_3	On	$V_m = -V_s$
	Q_1, Q_4	Off	
delay phase:	Q_1, Q_2, Q_3,Q_4	Off	$V_m = -V_s\, sgn(I)$

ex. for I>0 will the motor current flow through D_2 and D_3 during the delay phase

6.4.3 Unipolar drive (slow decay mode)





Voltage and current over the motor for one period in positive direction

In positive direction

phase a - on: Q_1, Q_4 On; Q_2, Q_3 Off → $V_m = V_s$

phase b - off: Q_1 On; $Q_2, Q_3\ Q_4$ Off → $V_m = 0$ *(freewheeling)*

In negative direction

phase a - on: Q_2, Q_3 On; Q_1, Q_4 Off → $V_m = -V_s$

phase b - off: Q_3 On; $Q_1, Q_2\ Q_4$ Off → $V_m = 0$ *(freewheeling)*

6.4.4 Selecting the switching frequency

- **The switching frequency (PWM signal frequency) must be selected corectly.´**
  - The lowest PWM frequency is decided by the electric time constant $\tau = L/R$ of the motor. You can not allow the current to increase and decrease to much for T_a and T_b respectively. A rule of thumb is that the PWM period T should be at least 10 times shorter then the electric time constant. $10T < \tau$.
  - The highest possible PWM frequency is decided by the delay time of the H-bridges transistors. If the delay time is a to long compared to T then the drive will become non-linear.
    Unfortuently does most H-bridges use delays also in unipolar drive mode.

6.4.5 Data from Infinion TLE 620

Output Delay Times (device not in stand-by for $t > 1$ ms)

High-side ON	$t_{d\,ON\,H}$	–	4	10	µs	V_S = 13.2 V, Resistive load of 12 Ω
High-side OFF	$t_{d\,OFF\,H}$	–	0.6	1	µs	
Low-side ON	$t_{d\,ON\,L}$	–	2	3.5	µs	
Low-side OFF	$t_{d\,OFF\,L}$	–	2.5	4	µs	

Output Switching Times (device not in stand-by for $t > 1$ ms)

High-side switch rise time	$t_{RISE\,H}$	–	1.8	3.5	µs	V_S = 13.2 V, Resistive load of 12 Ω
High-side switch fall time	$t_{FALL\,H}$	–	0.2	0.8	µs	
Low-side switch rise time	$t_{RISE\,L}$	2	6.5	11	µs	
Low-side switch fall time	$t_{FALL\,L}$	2	4.3	6.5	µs	

Note: For switching time definitions, see ***Figure 6****.*

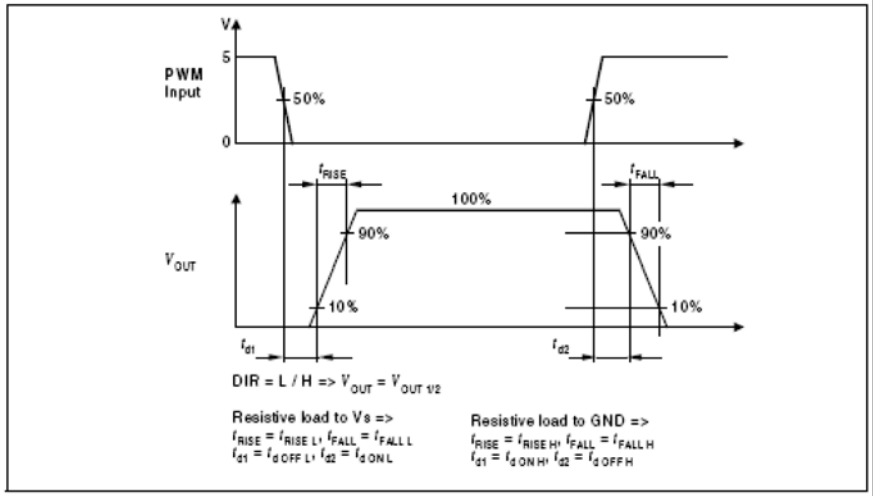


Figure 6 Output Delay and Switching Time Definitions