Servo and Stepper Motor Control

References:

- STMicroelectronics, “L298 Dual Full Bridge Driver,” datasheet, Jan. 2000.
- STMicroelectronics, “L297 Stepper Motor Controllers,” datasheet, Dec. 2001.

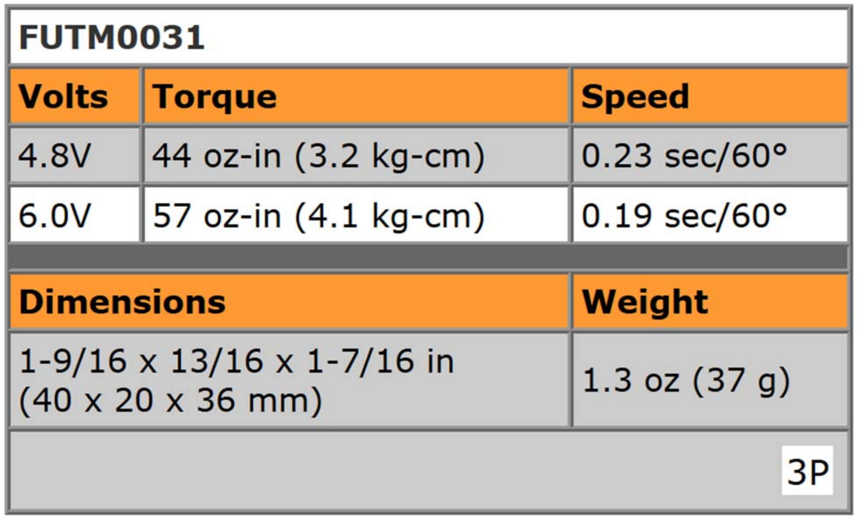
Actuators, Relays, Servos and Stepper Motors

- Microcomputers are often required to control the movement of mechanical devices or to set other physical parameters.
- An ***actuator*** is a device that is used by a microcomputer to control an external physical quantity, such as position.
- A ***relay*** is a mechanical switch that is closed and opened by driving or not driving current through a magnetic coil.
  Normally open (n.o.) relay: close the switch by energizing
  Normally closed (n.c.) relay: open the switch by energizing
- A ***servo*** is an actuator that is used to cause mechanical rotation within some fixed range of angles (< 360 deg.)
- A ***stepper motor*** is an actuator whose rotational position (360 deg.) and speed can be controlled by digital signals.

Servos

- A ***servo*** is an actuator that is used to cause mechanical rotation within some fixed range of angles (< 360 deg.)
- Example application: Servos are used in radio controlled aircraft to control the flight surfaces: *rudder* (yaw control), *elevator* (pitch control), and *ailerons* (roll control).
- A widely used control interface for servos uses an analog signaling method called ***pulse width modulation (PWM)***. Many microcontrollers have PWM interfaces (e.g., the MCF54415 has 8 PWM channels).
- A train of positive pulses is sent at some fixed repetition rate, for example, at 50 Hz. The pulses have a fixed amplitude (e.g., 3 to 5 V) and a width that is controlled to be within some range about a neutral width (e.g., 1.5 ms +/- 0.5 ms). The pulse width determines the angle of a rotor.

A Typical Small Servo: Futaba S3003

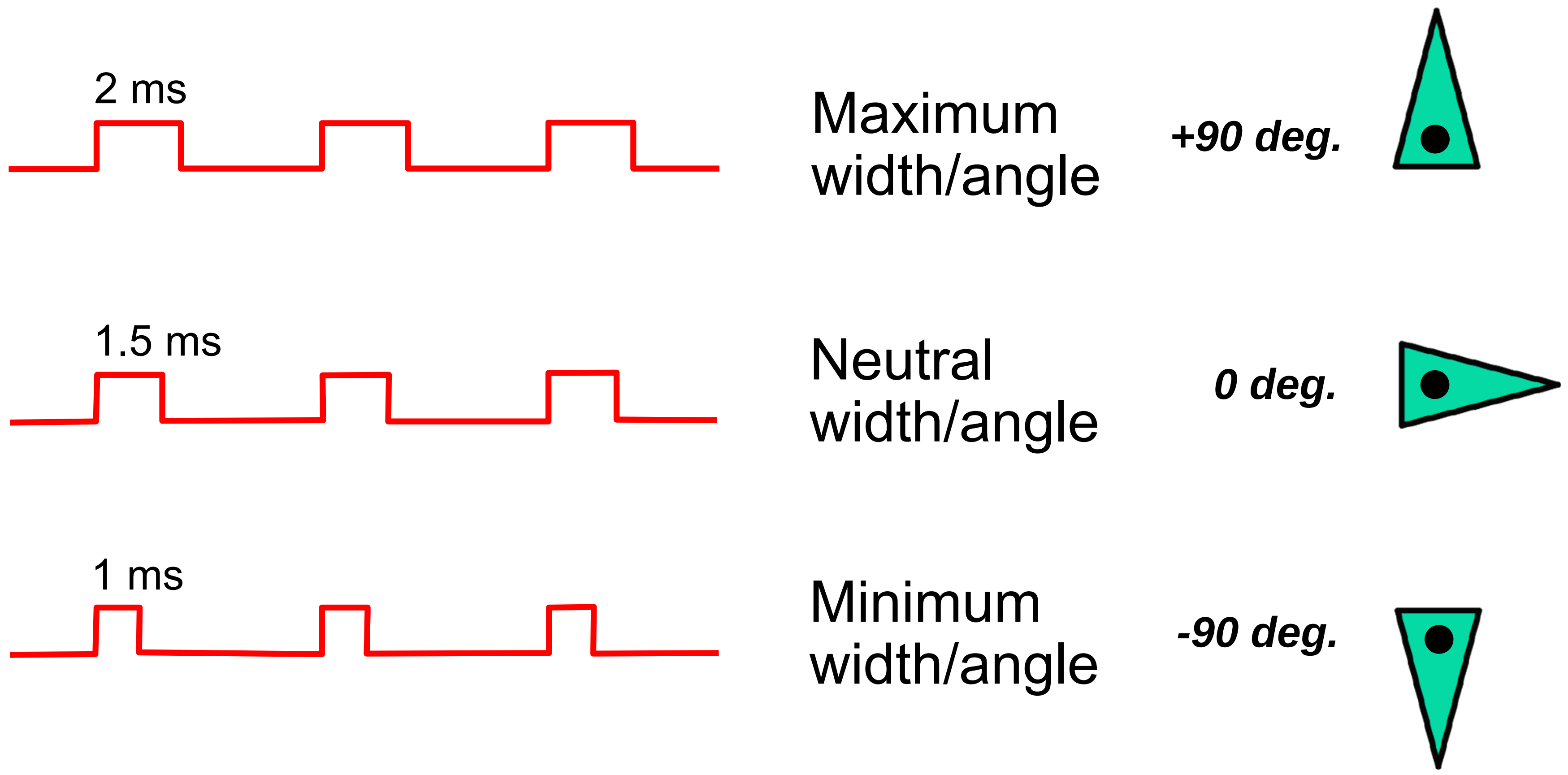


FUTM0031		
Volts	**Torque**	**Speed**
4.8V	44 oz-in (3.2 kg-cm)	0.23 sec/60°
6.0V	57 oz-in (4.1 kg-cm)	0.19 sec/60°
Dimensions		**Weight**
1-9/16 x 13/16 x 1-7/16 in (40 x 20 x 36 mm)		1.3 oz (37 g)
		3P

Figures courtesy of FutabaUSA

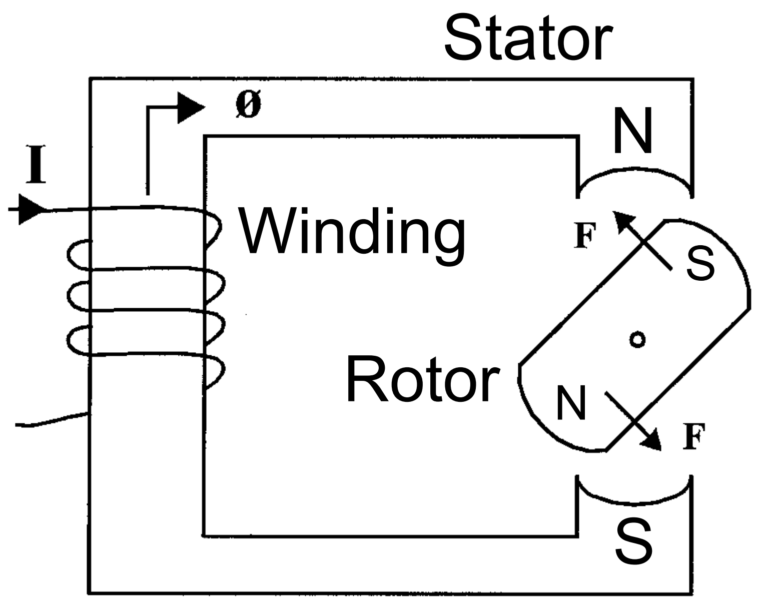
- The voltage given in the table is the supply for the servo motor.
- The given rotational speed is for an unloaded servo. A loaded servo will be slower to rotate in response to PWM width changes.
- The servo motor provides a holding torque that keeps the rotor at the controlled angle, despite any externally applied torque.

PWM Signals for Controlling the Rotor Angle



Note: The duty cycle (i.e., pulse width to pulse period ratio) is not drawn to scale. A 50 Hz pulse rate implies that pulses arrive every 20 ms.

Basic Principles behind the Stepper Motor

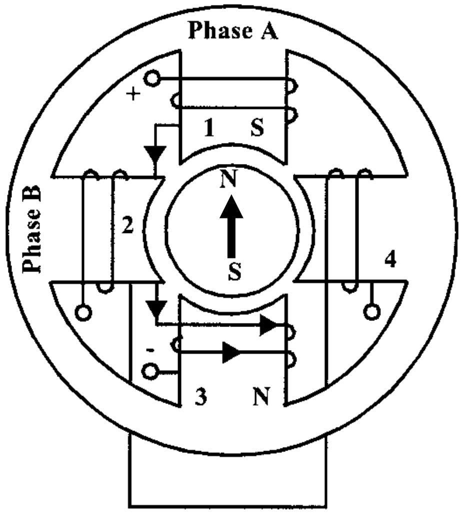


- A rotating ***rotor*** contains a permanent magnet, which has fixed north (N) and south (S) poles.
- A stationary ***stator*** can be energized with current to produce an electro-magnet. The location of the N and S poles of this magnet are determined by the direction of the current driven through the one or more winding(s).

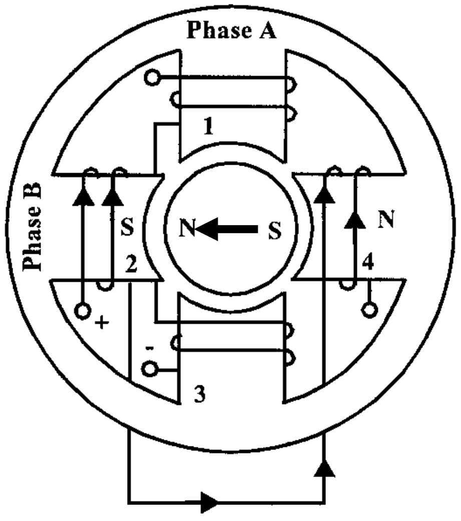
- The rotor experiences a torque that acts to bring the rotor into alignment with the stator (the N and S poles attract one another). The torque disappears when the stator is fully aligned with the rotor.
- Controlled rotation or the rotor can be obtained by energizing *on* and de-energizing *off* multiple stator windings in the correct sequence.

Simplified View of Stepper Motor Operation

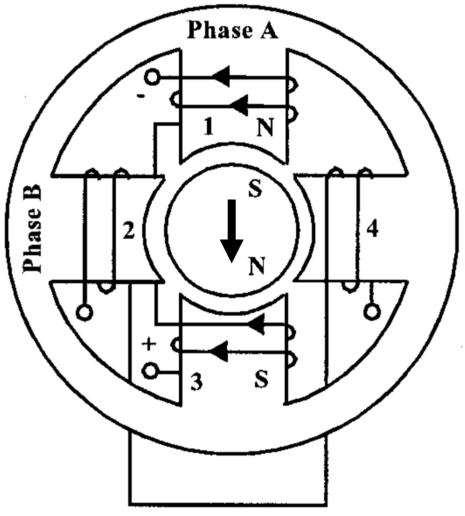
Step #1



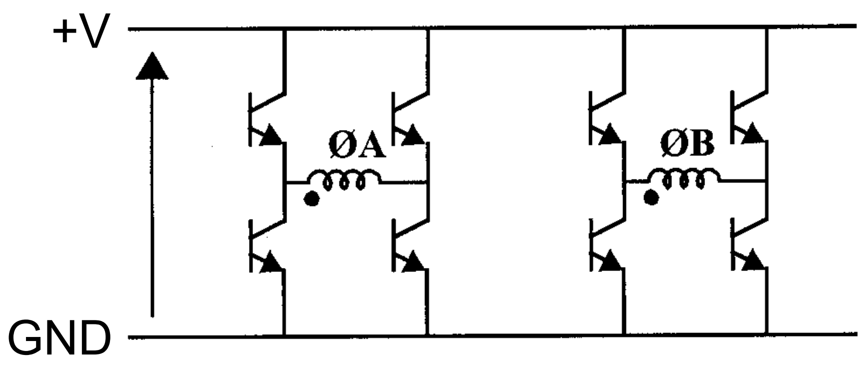
Step #2



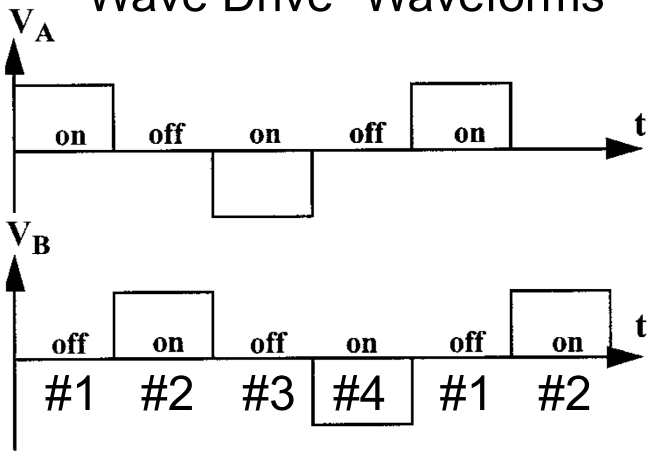
Step #3



Bipolar Full H-Bridge Circuit

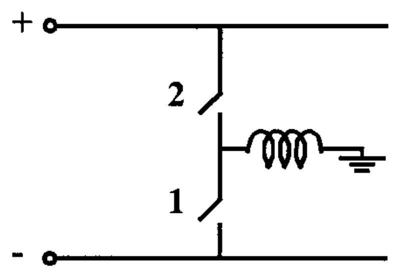


"Wave Drive" Waveforms

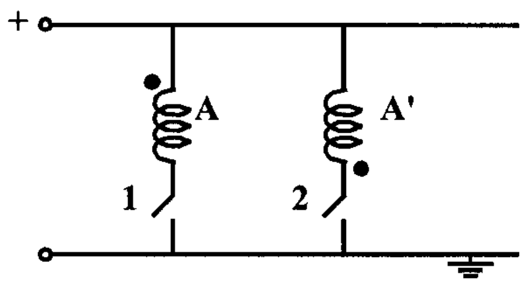


Two Additional Drive Configurations

Bipolar, one winding
(Also called "Half-H Bridge", "Half-Bridge" & 'Totem Pole")



Unipolar, two windings



Pro: Minimum number of windings.

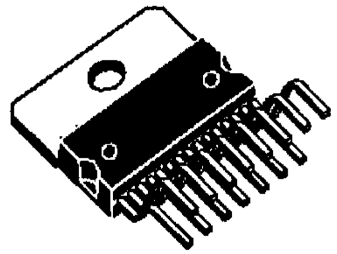
Con: Needs three supply voltages.

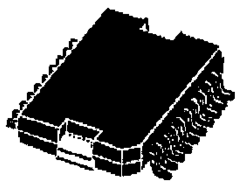
Pro: Simple bridge circuit.

Con: Twice as many windings required on the stator.

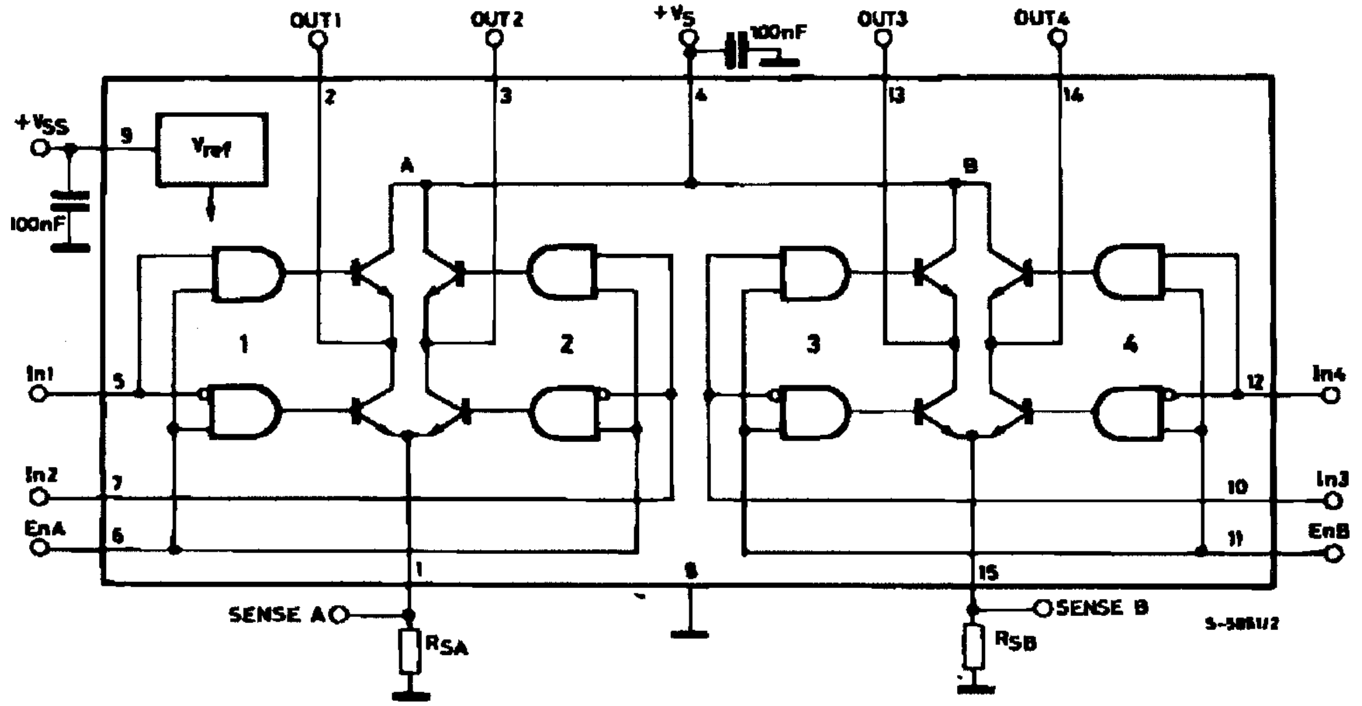
Conclusion: The bipolar full H-bridge circuit is more common.

STMicro L298 Dual H-Bridge Driver IC


Multiwatt15

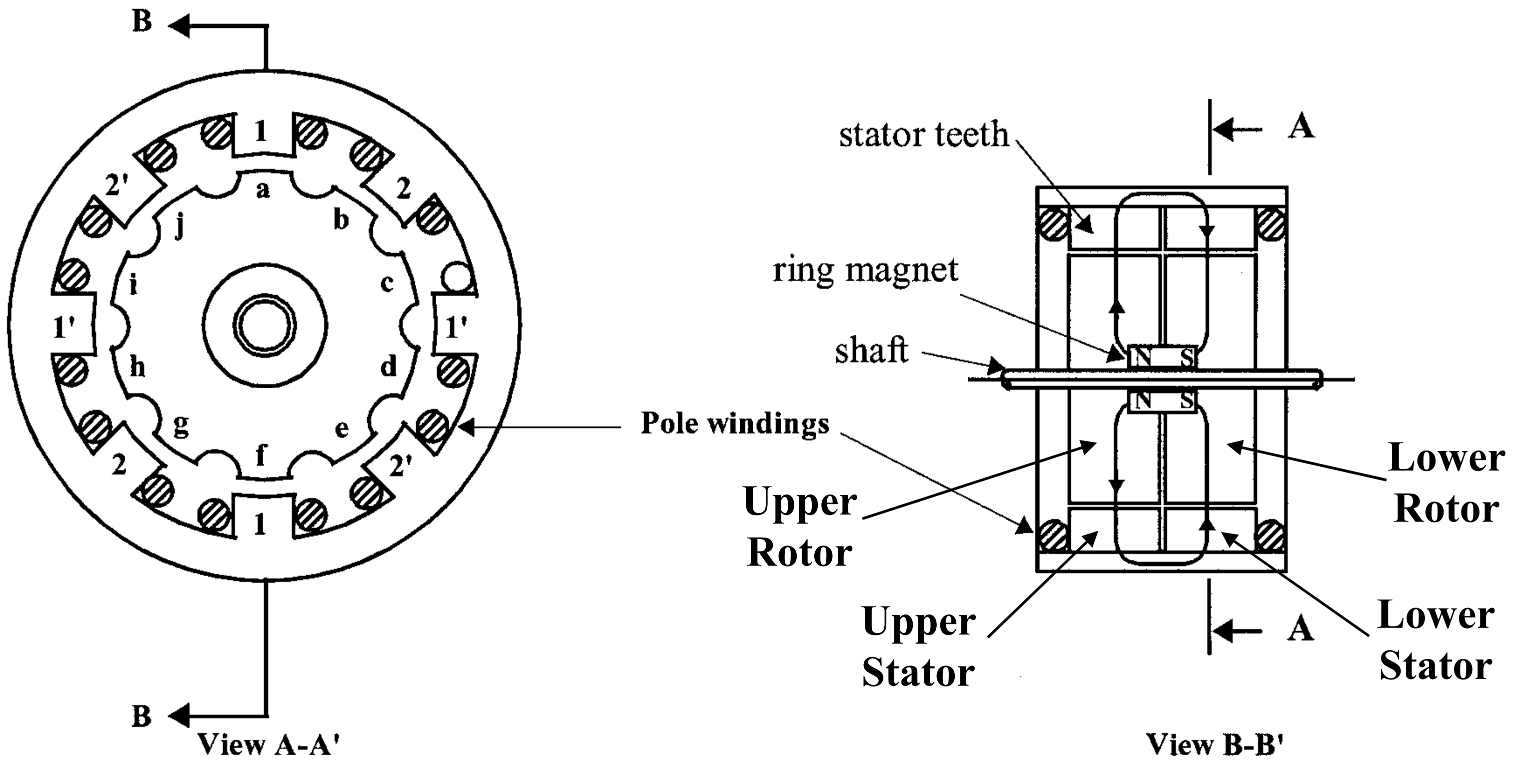

PowerSO20

- TTL-compatible control inputs
- Drives inductive loads at up to 46 V and a total of 4 A DC



Copywrite © 2000 STMicroelectronics

Hybrid Stepper Motor with Two Stator Stacks

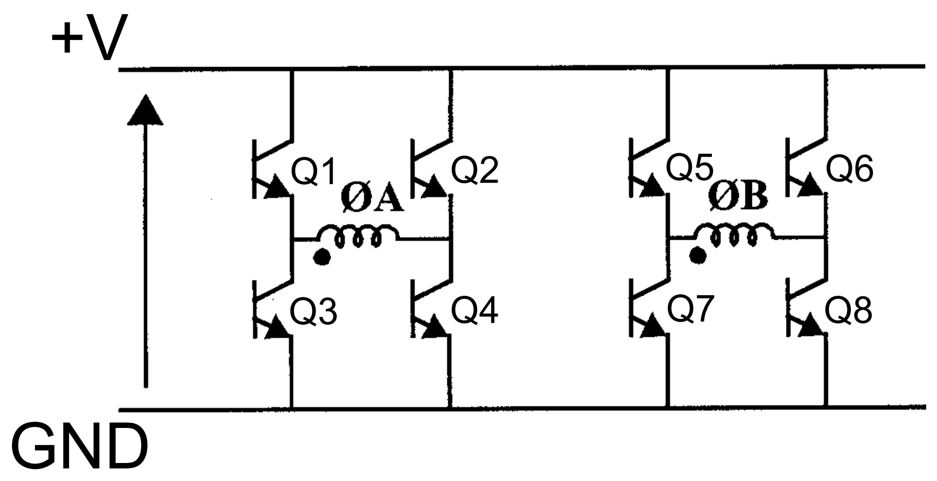


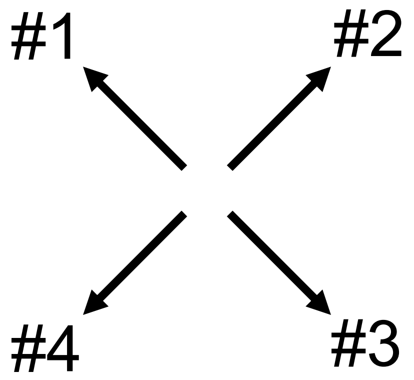
Number of rotor teeth, $n_r = 10$

Number of stator teeth, $n_s = 8$

Step angle = $(360/n_s) - (360/n_r) = 45 - 36 = 9$ degrees

Full-Step Sequence (constant current, bipolar drive)

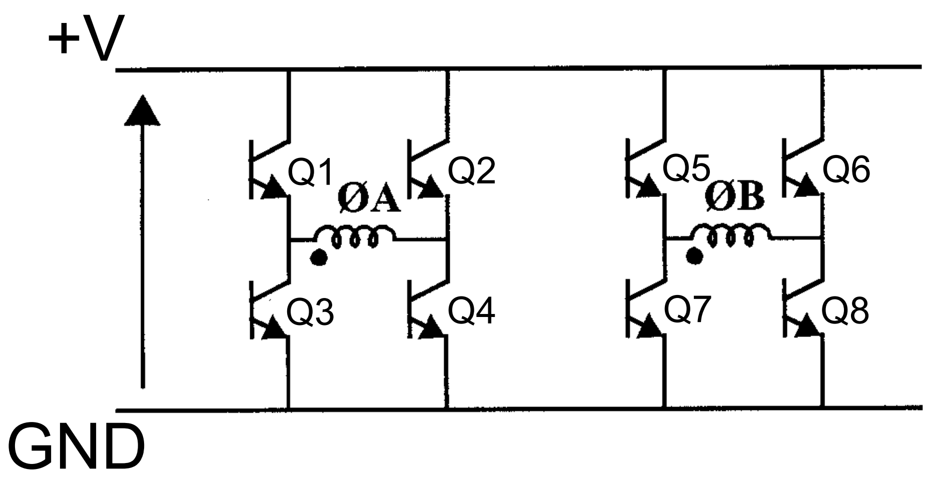


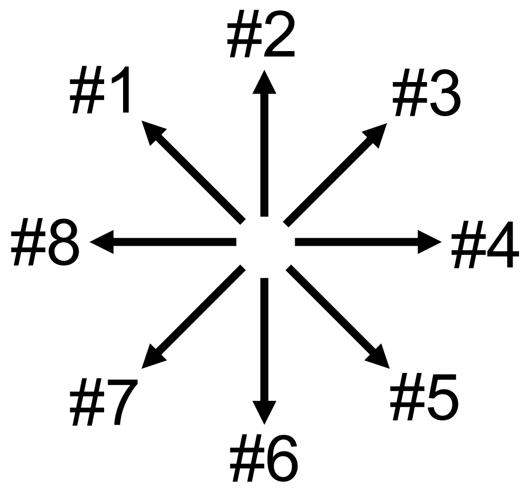


Step	Q1-Q4	Q2-Q3	Q5-Q8	Q6-Q7	Rotor Angle (mod 360 deg.)
#1	on	off	on	off	315 = -45
#2	on	off	off	on	45 = -45 + 90
#3	off	on	off	on	135 = 45 + 90
#4	off	on	on	off	225 = 135 + 90
#1	on	off	on	off	315 = 225 + 90
#2	on	off	off	on	45 = 405 = 315 + 90

Note: The step angle in real stepper motors is never 90 degrees.
A step angle of 1.8 degrees (200 steps per rev.) is fairly common.

Half-Step Sequence (bipolar drive)

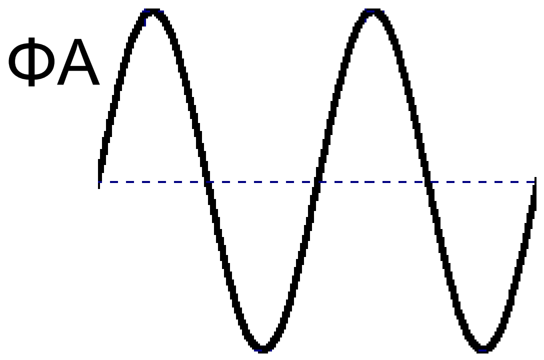




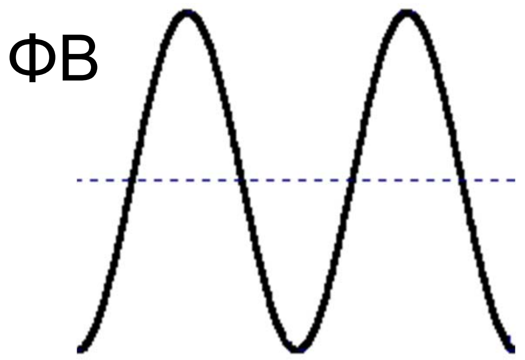
Step	Q1-Q4	Q2-Q3	Q5-Q8	Q6-Q7	Rotor Angle (mod 360 deg.)
#1	on	off	on	off	315 = -45
#2	on	off	off	off	0 = 360 = -45 + 45
#3	on	off	off	on	45 = 0 + 45
#4	off	off	off	on	90 = 45 + 45
#5	off	on	off	on	135 = 90 + 45
#6	off	on	off	off	180 = 135 + 45
#7	off	on	on	off	225 = 180 + 45
#8	off	off	on	off	270 = 225 + 45
#1	on	off	on	off	315 = 270 + 45

Synchronous, Full-Step and Half-Step Drive Waveforms

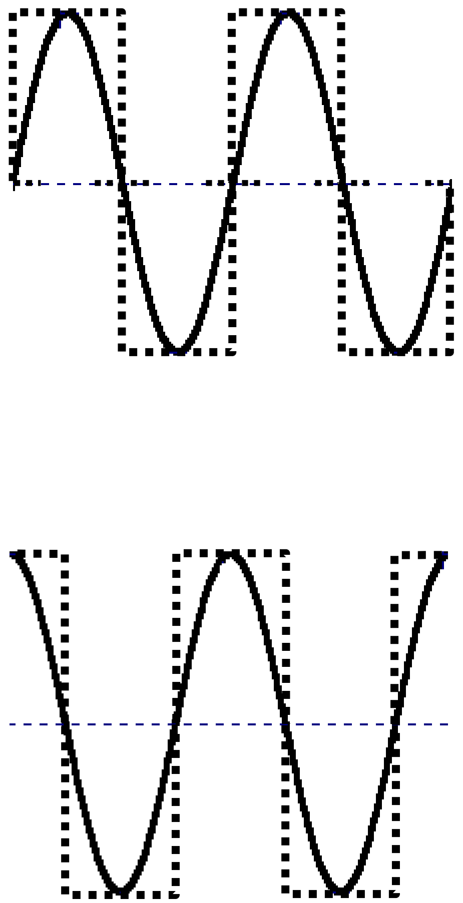
Synchronous Drive
Constant torque & current



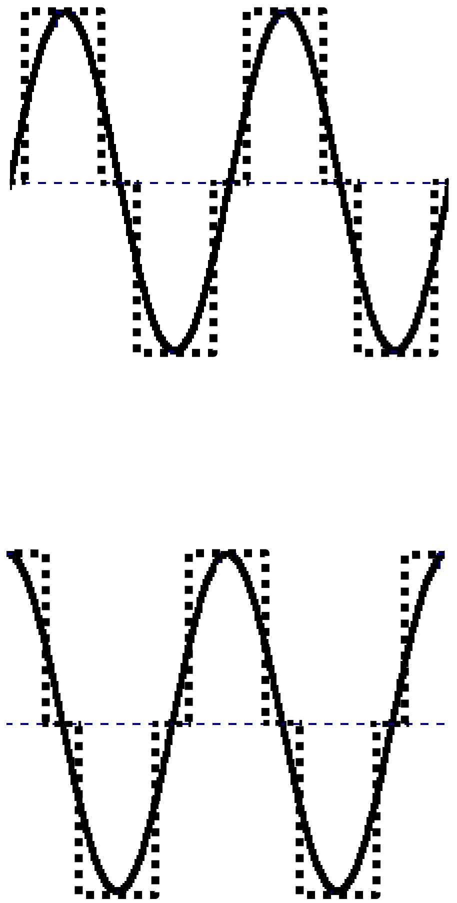
Sinusoids 90 degrees out of phase



Full-Step Drive
Pulsing torque



Half-Step Drive
Pulsing torque



Stator Drive Waveforms

- The simple wave drive waveform is rarely used. The full-step waveform provides stronger holding torque, with the same number of step positions as wave drive.
- *Note*: Both the full-step and half-step waveforms are rough approx-imations to the sinusoidal waveforms used in synchronous motors.
- ***Half-step drive*** provides twice as many stable rotor positions compared to full-step drive, at the cost of an only slightly more complicated drive waveform.
- ***Full-step drive*** provides more constant total winding current. (The current is still not steady in a rotating motor because of the effects of the inductance of the stator windings.)
- In ***microstepping*** waveforms, even finer control of the rotor position is achieved by using a digital-to-analogue (DAC) converter to produce two variable-amplitude drive waveforms that closely approximate sinusoids. Each full step could be split into, say, 256 microsteps.

Practical Stepper Motors

- Real stepper motors come in a variety of configurations:
  - different numbers of phase windings
  - unipolar (one dir.) or bipolar (both dirs.) current feeds
  - different numbers of steps (e.g., 200 steps per 360 deg.)
  - different values of generated mechanical torque

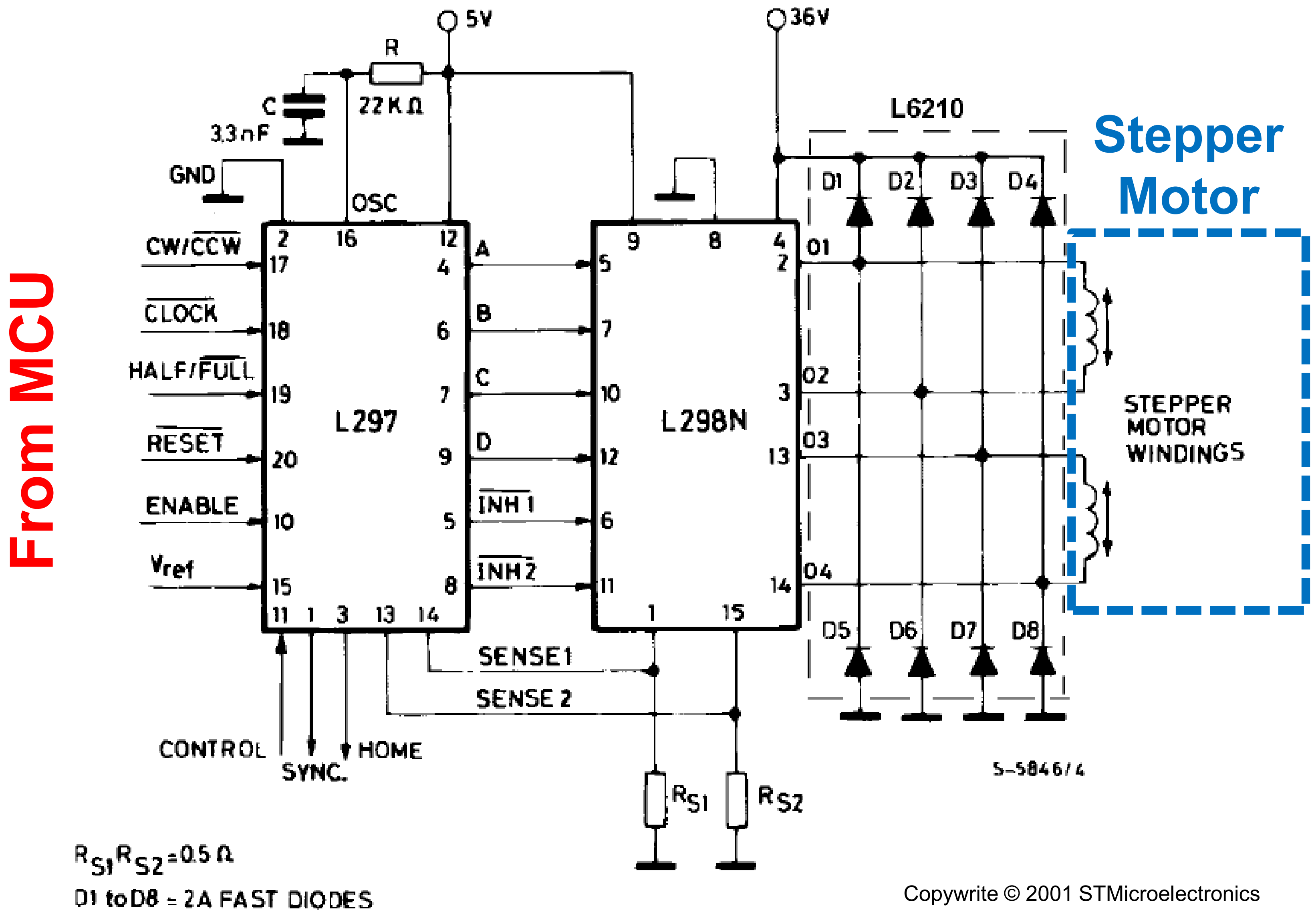
- The bridge circuit controls the switching of current through the windings. The microcontroller must produce transistor control signals with the correct sequence and timing. These control signals may need to be buffered (e.g., to produce sufficiently large base-emitter current).

- The rotational speed must not be changed too suddenly; otherwise, the motor shaft position will slip away from its expected position to another (incorrect) stable position.

STMicro L297 Stepper Motor Controller

- Some microcontroller units (e.g., Freescale MCF5234 and MCF54415) include subsystems that allow complex timing waveforms to be produced.
- The Enhanced Time Processing Unit (eTPU) in the MCF5234 has 16 hardware “channels” that can be associated with software functions that can be used to control stepper motors and many other high-speed timing applications.
- In our laboratory, we will be using the L297 integrated circuit from STMicroelectronics, together with the MCF54415 MCU and an L298 H-bridge driver, to control a stepper motor.

L297 Stepper Motor Controller Connections

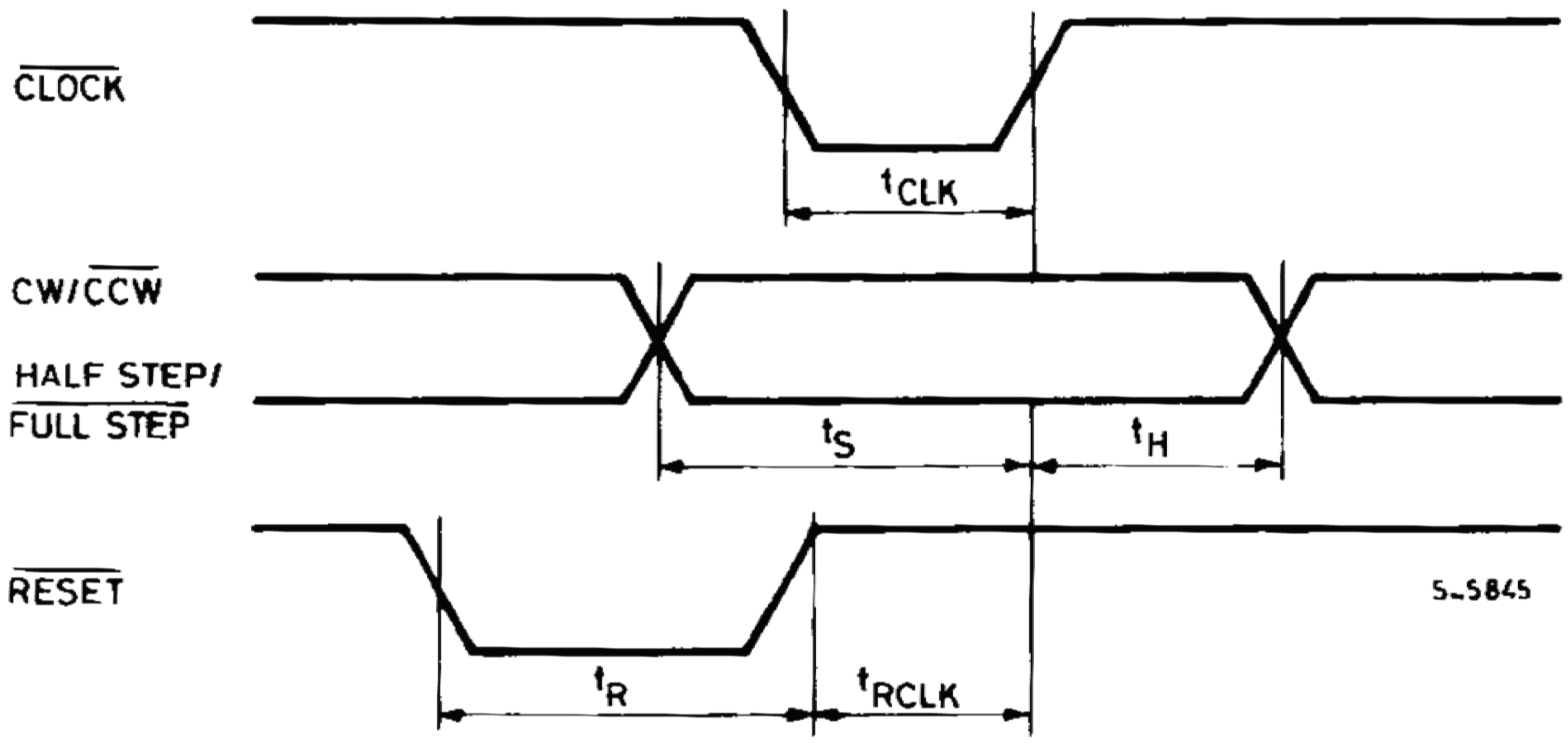


Copywrite © 2001 STMicroelectronics

STMicro L297 Features

- Generates four-phase drive signals that are suitable for controlling both (a) two-phase bipolar stepper motors, and (b) four-phase unipolar stepper motors.
- The full-step, half-step and wave drive waveforms can be generated without detailed intervention from the micro-controller unit (MCU).
- The MCU must provide the step clock (CLOCK*), motor direction (CW/CCW*) and step mode (HALF/FULL*) input signals.
- An active low pulse on CLOCK* advances the motor by one increment on the rising edge of CLOCK*.
- The active low RESET* signal forces the L297 into the output state ABCD = 0101 (state 1).

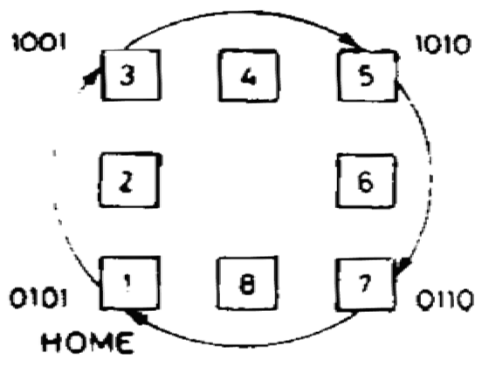
L297 Input Waveforms

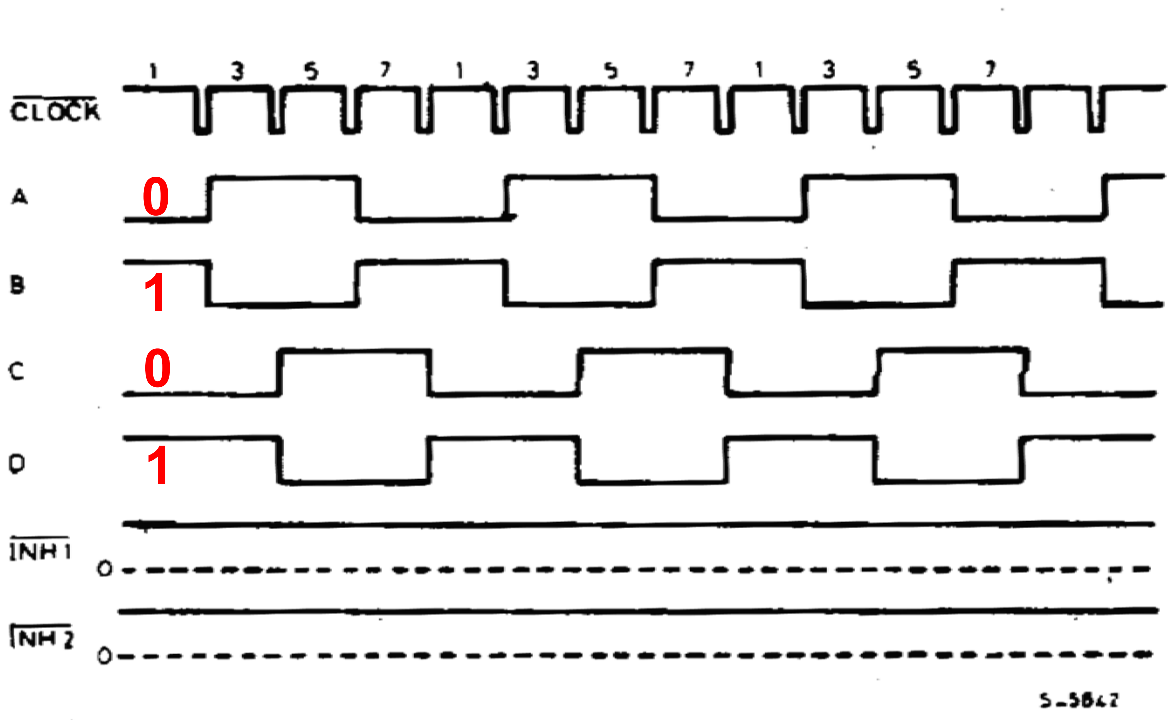


$\overline{\text{CLOCK}}$
t_{CLK}
$\text{CW}/\overline{\text{CCW}}$
$\text{HALF STEP}/\overline{\text{FULL STEP}}$
t_S
t_H
$\overline{\text{RESET}}$
5-5845
t_R
t_{RCLK}

L297 Full-step Output Waveforms

Note: HALF/FULL* is low when the state is 1 (ABCD = 0101) or 3 (1001), 5 (1010) or 7 (0110).

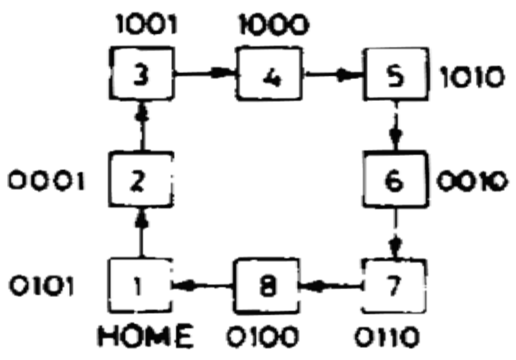


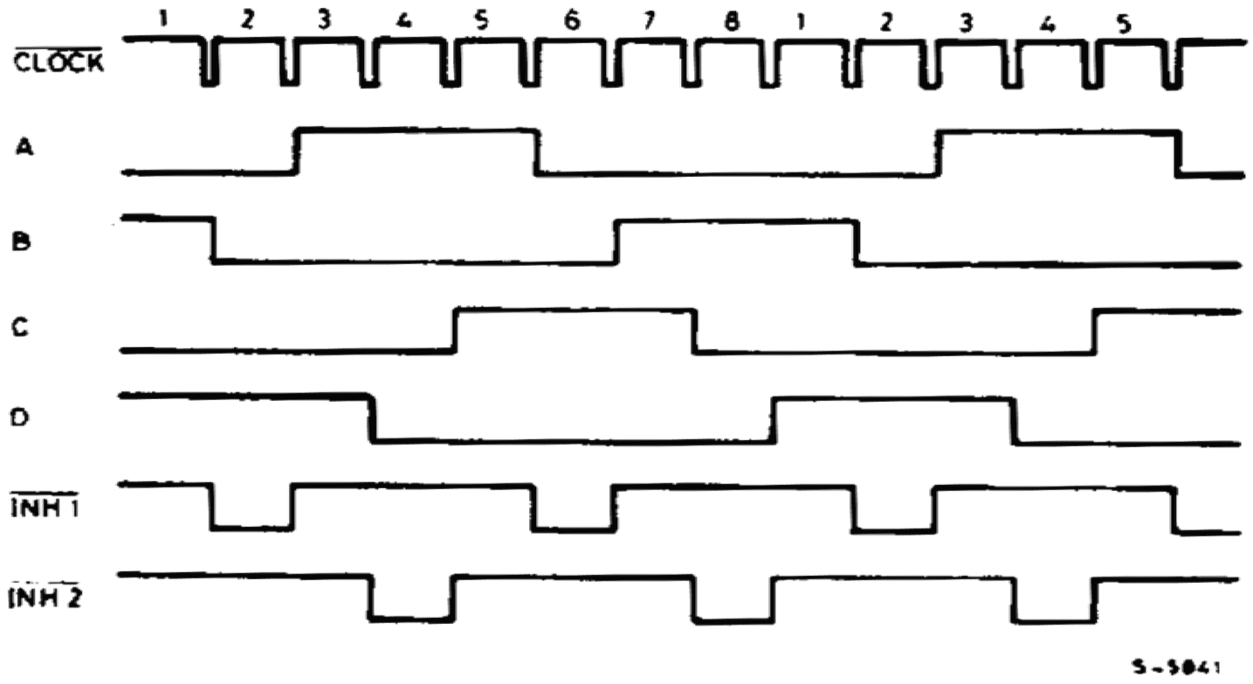


Copywrite © 2001 STMicroelectronics

L297 Half-step Output Waveforms

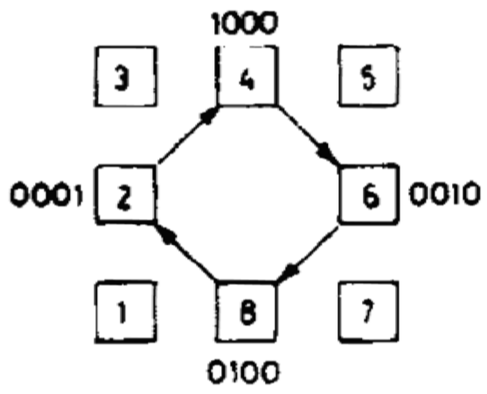
Note: HALF/FULL* is held high. The state sequence is 1 (ABCD = 0101), 2 (0001), 3 (1001), 4 (1000), 5 (1010), 6 (0010), 7 (0110) and 8 (0100).

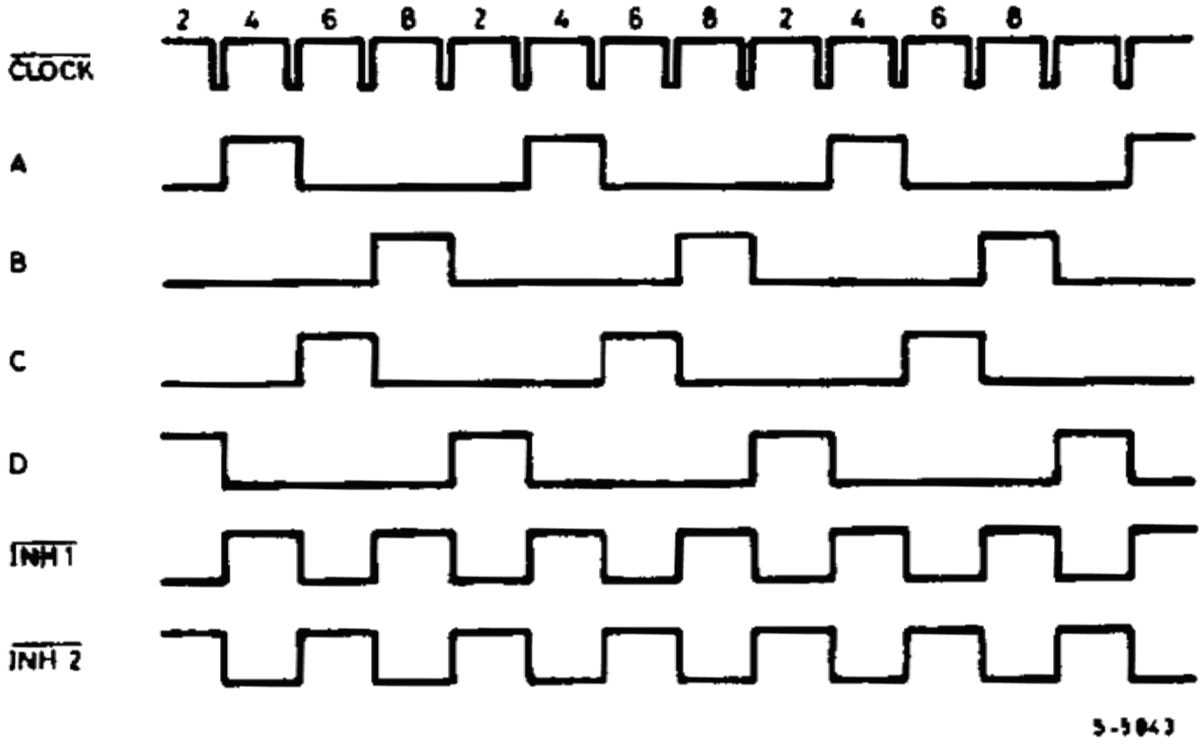




L297 Wave Drive Output Waveforms

Note: HALF/FULL* is low when the state is 2 (ABCD = 0001) or 4 (1000), 6 (0010) or 8 (0100).

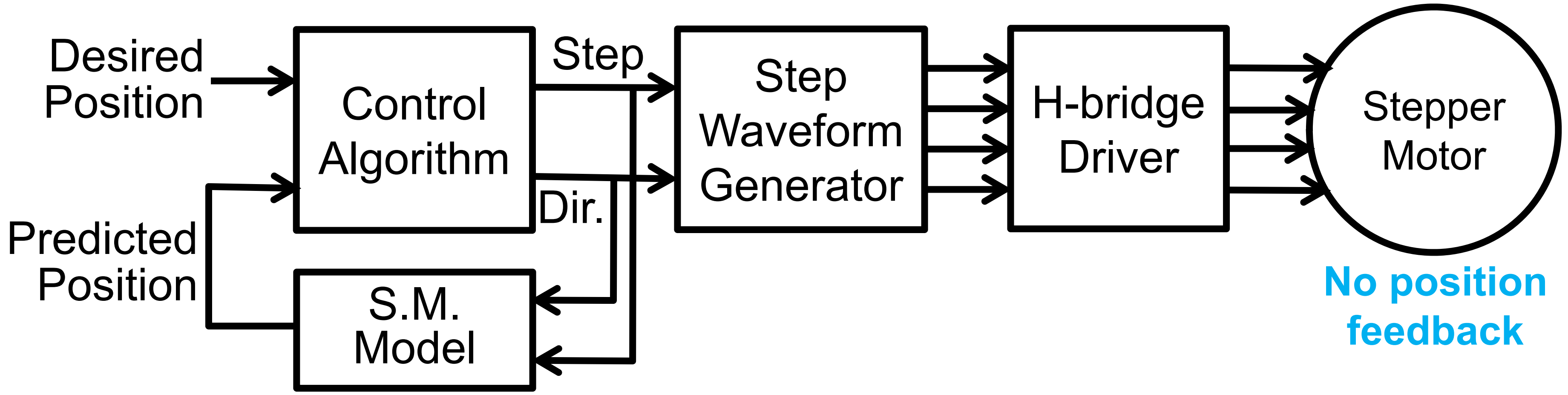




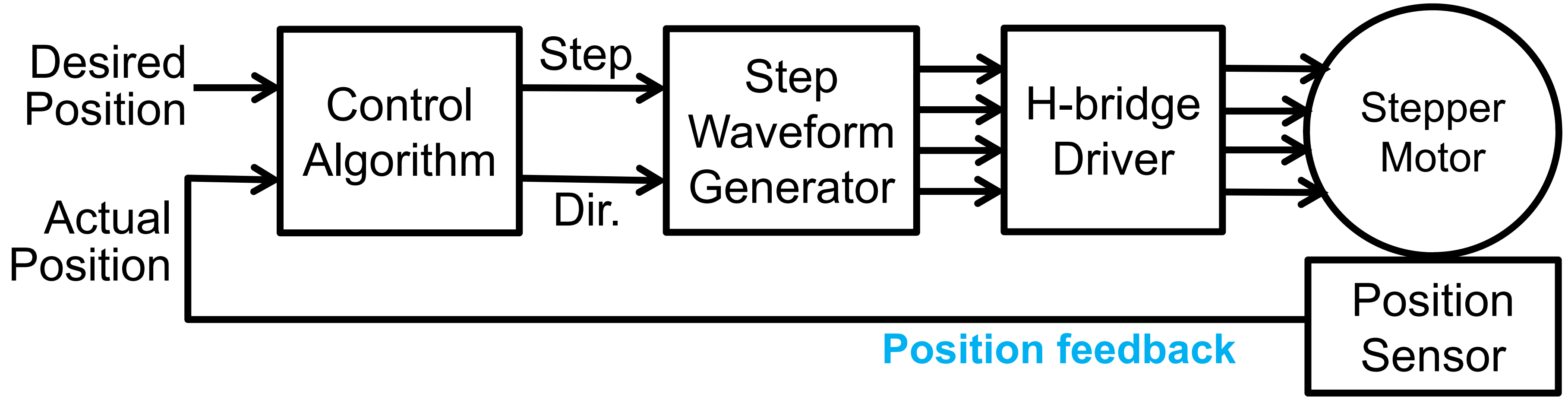
Copywrite © 2001 STMicroelectronics

Open-loop vs. Closed-loop Control (1)

Open-loop Stepper Motor Control Configuration:



Closed-loop Stepper Motor Control Configuration:



Open-loop vs. Closed-loop Control (2)

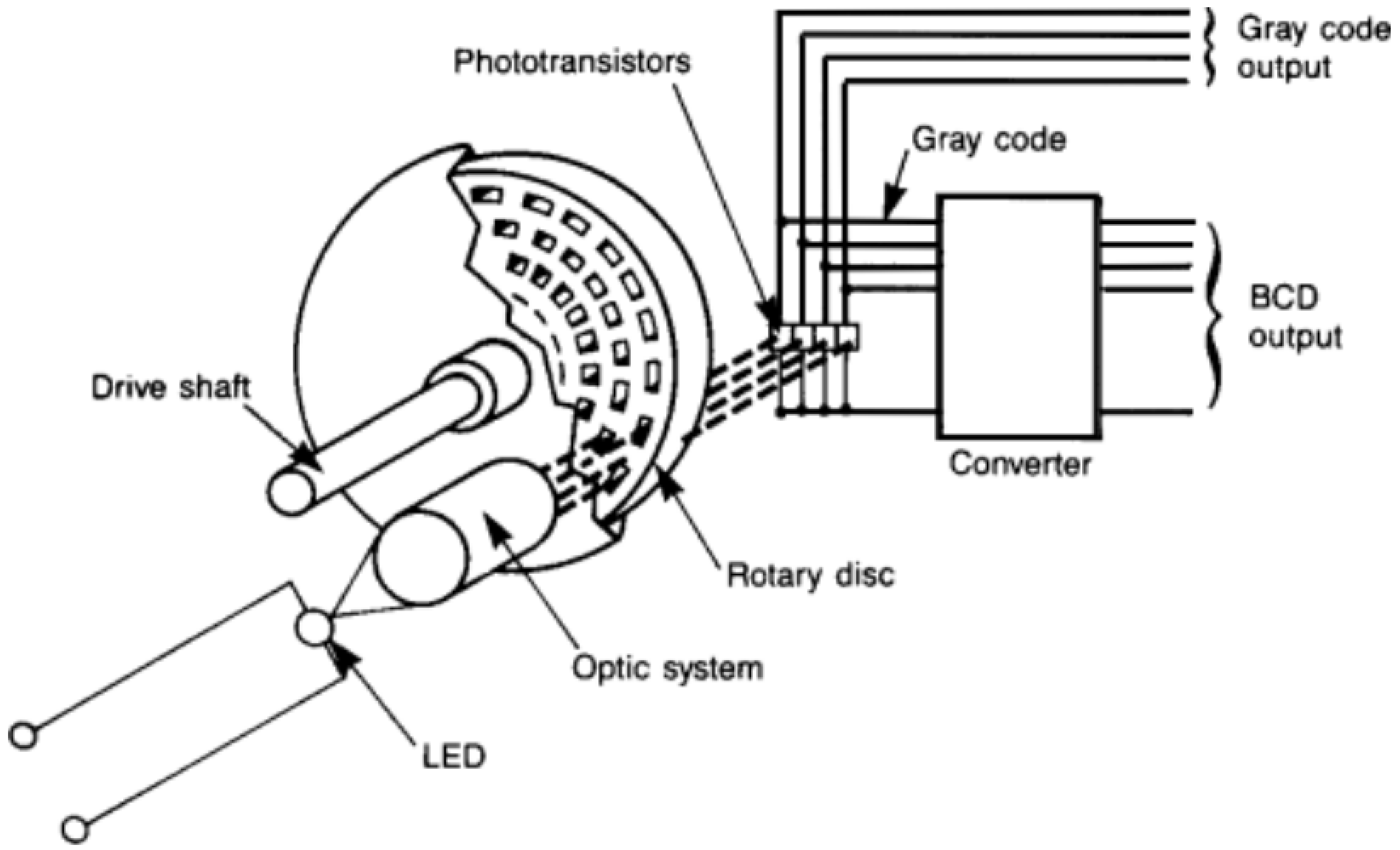
Open-loop Stepper Motor Control Configuration:

- A method is required to put the rotor into the initial position.
- If the rotor fails to keep up with step commands (e.g., the step commands were too fast and the inertia of the rotor and load caused the rotor to slip behind), the actual rotor position might differ from the desired position.
- The control algorithm is blind and cannot make corrections to the motor position to account for slippage.

Closed-loop Stepper Motor Control Configuration:

- A position sensor (e.g., a shaft encoder) at the motor determines the actual rotor position and sends this information as feedback to the control algorithm.
- The control algorithm first computes the difference between the desired position and the *actual position* before it determines what step commands to produce.

Shaft Encoder



From www.plcdev.com

Gray (or Reflected) Code (1)

- The Gray code (invented by Frank Gray in 1947 at Bell Laboratories) is a sequence of binary codewords where adjacent codewords differ in one bit.
- The Gray code is useful in position sensors because, even if the sensor of one bit is slightly slow or fast, the detected codeword is always accurate to within one position.
- The codewords in a Gray code are constructed using an iterative reflection algorithm, starting with 0 and 1:

0, 1

00, 01, 11, 10

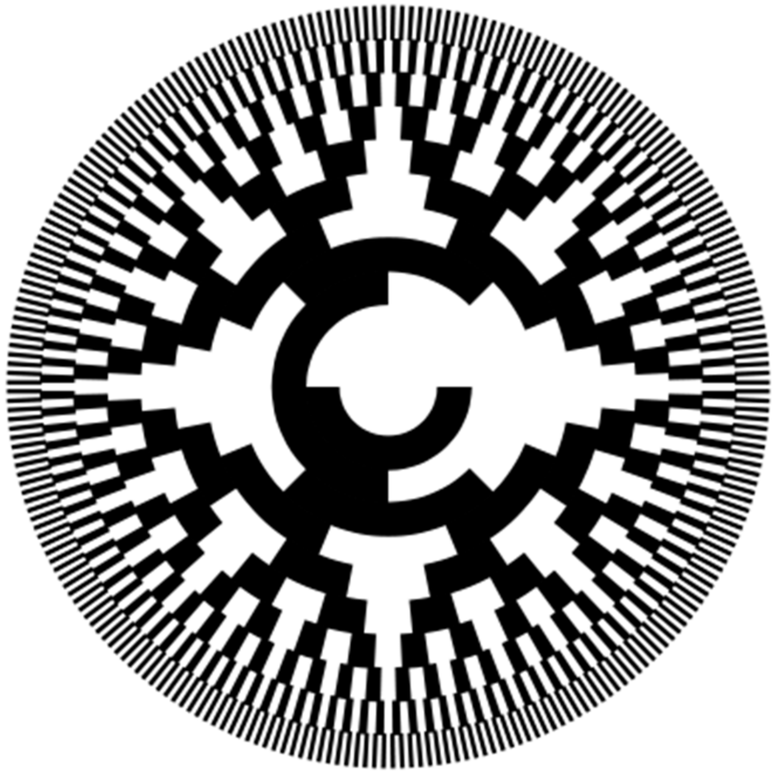
000, 001, 011, 010, 110, 111, 101, 100

etc.

Gray (or Reflected) Code (2)

- A Gray code can be used as a pattern that is painted on a disc (or painted on a shaft directly).
- The Gray code words can then be read using a linear array of light sensors (e.g., phototransistors).

```
   #0: 0000000000
   #1: 0000000001
   #2: 0000000011
           etc.
#1021: 1000000011
#1022: 1000000001
#1023: 1000000000
```


A Gray code disc
with 1024 positions

Stepper Motor Control Algorithms

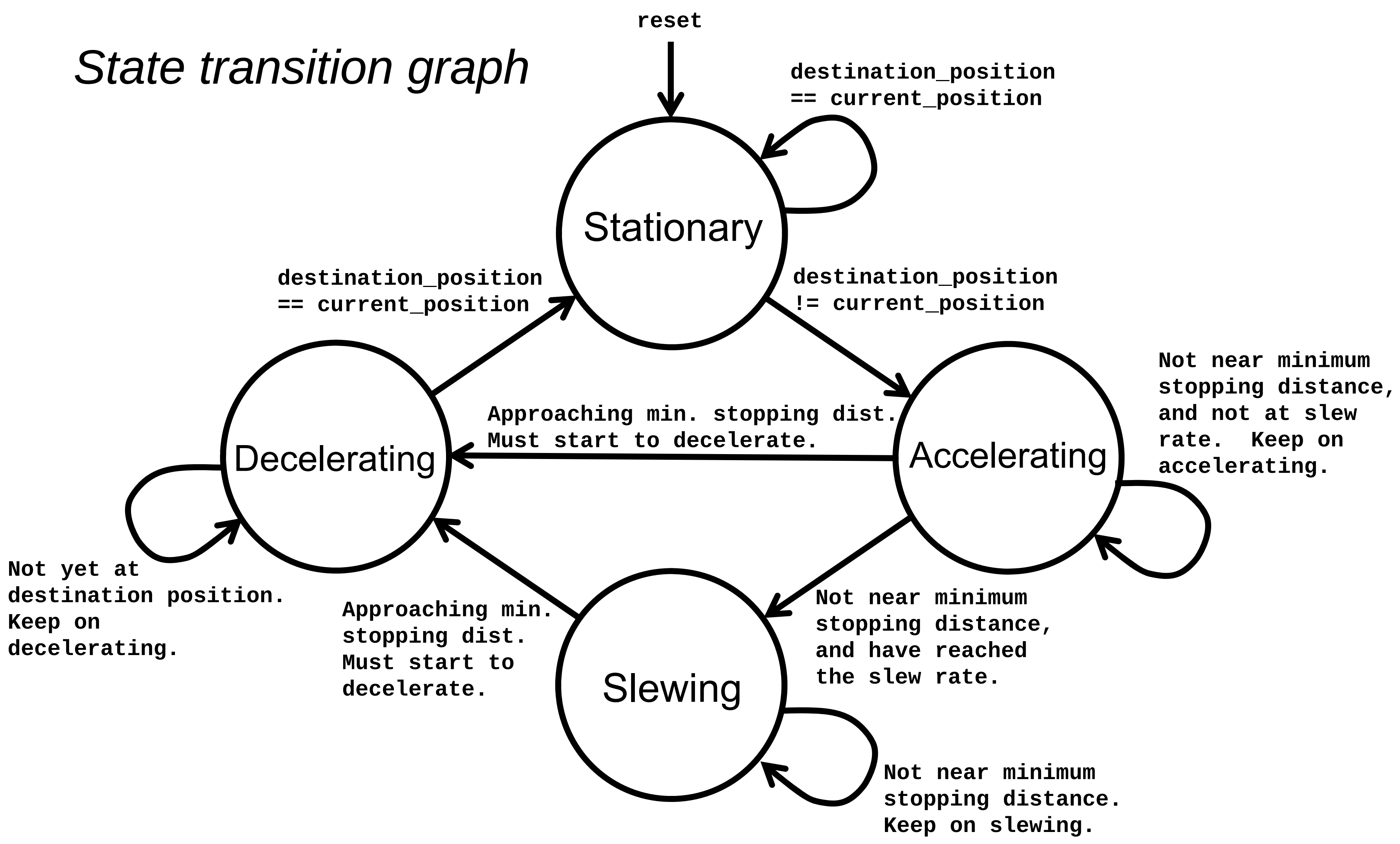
- Modern microcontroller units (MCUs) can implement a wide variety of possible algorithms in software for controlling stepper motors (SMs) using either the open-loop or closed loop configurations.
- A SM control algorithm must take into account practical constraints:
  - The rotor and attached load have *angular inertia* and the motor has a *maximum torque*. The control algorithm should not attempt to exceed the resulting maximum possible angular acceleration.
  - The motor will have a maximum allowed rotational speed (*slew rate*). The controller should not attempt to cause the motor to exceed that maximum speed. This means there is a *maximum allowed step rate*.

Open-loop Stepper Motor Control using the eTPU

- The *Enhanced Time Processing Unit* (eTPU) block in the MCF5234 MCU allows a list of increasingly fast step rates to be defined in advance, ranging from a slowest step rate up to a maximum step rate (the slew rate).
- When the desired position is changed by the CPU, the eTPU determines the direction of movement, and then starts producing step signals at the slowest step rate.
- The step rate is then increased gradually up to the next fastest step rate, and then the next step rate, right up to possibly the fastest step rate.
- When the destination position is approached, the step rates are ramped down in the opposite order until the rotor reaches the desired position at the slowest step rate.
- In the MCF54415, the same controller behaviour can be achieved using software.

Open-loop Stepper Motor Control Algorithm (1)

State transition graph



reset
destination_position == current_position
Stationary
destination_position != current_position
Accelerating
Not near minimum stopping distance, and not at slew rate. Keep on accelerating.
Approaching min. stopping dist. Must start to decelerate.
Decelerating
destination_position == current_position
Not yet at destination position. Keep on decelerating.
Not near minimum stopping distance, and have reached the slew rate.
Slewing
Not near minimum stopping distance. Keep on slewing.
Approaching min. stopping dist. Must start to decelerate.

Open-loop Stepper Motor Control Algorithm (2)

```
present_state = ST_STATIONARY;
present_position = STARTING_POSITION;
step_period = STATIONARY_POLLING_PERIOD;

while (1) {
    switch ( present_state ) {
        case ST_STATIONARY:   do_stationary();
                              break;
        case ST_ACCELERATING: do_accelerating();
                              break;
        case ST_SLEWING:      do_slewing();
                              break;
        case ST_DECELERATING: do_decelerating();
                              break;
    }  // end switch

    OS_delay( step_period );
}  // end while
```

Open-loop Stepper Motor Control Algorithm (3)

```
void do_stationary()
{
    if ( motor_enabled )
        if ( destination_position == current_position )
            present_state = ST_STATIONARY;
            step_period = STATIONARY_POLLING_PERIOD;
        else
            if ( destination_position > current_position )
                delta_position = +1;
                output_forward_signal_to_motor_driver;
            else
                delta_position = -1;
                output_reverse_signal_to_motor_driver;
            endif;
            present_state = ST_ACCELERATING;
            step_period = MAXIMUM_STEP_PERIOD; // start slowly
        endif;
    else
        present_state = ST_STATIONARY;
        step_period = STATIONARY_POLLING_PERIOD;
    endif;
}
```

Open-loop Stepper Motor Control Algorithm (4)

```
void do_accelerating()
{
    output_step_signal_to_motor_driver;

    current_position = current_position + delta_position;
    distance_remaining = abs(destination_position –
                                                current_position);
    look_up min_stopping_distance for step_period;
    if (min_stopping_distance > distance_remaining - MARGIN)
        look_up period_increase for step_period;
        step_period = step_period + period_increase;
        present_state = ST_DECELERATING;

    elsif (step_period <= SLEW_PERIOD + MIN_PERIOD_DECREASE)
        step_period = SLEW_PERIOD;
        present_state = ST_SLEWING;

    else
        look_up period_decrease for step_period;
        step_period = step_period – period_decrease;
        present_state = ST_ACCELERATING;
    endif;
}
```

Open-loop Stepper Motor Control Algorithm (5)

```
void do_slewing()
{
    output_step_signal_to_motor_driver;

    current_position = current_position + delta_position;
    distance_remaining = abs(destination_position –
                                    current_position);
    look_up min_stopping_distance from step_period;

    if (min_stopping_distance > distance_remaining - MARGIN)
        look_up period_increase for SLEWING_PERIOD;
        step_period = SLEWING_PERIOD + period_increase;
        present_state = ST_DECELERATING;

    else
        step_period = SLEWING_PERIOD;
        present_state = ST_SLEWING;
    endif;
}
```

Open-loop Stepper Motor Control Algorithm (6)

```
void do_decelerating()
{
    output_step_signal_to_motor_driver;

    current_position = current_position + delta_position;
    distance_remaining = abs(destination_position –
                                 current_position);
    look_up min_stopping_distance for step_period;

    if (current_position == destination_position)
        step_period = STATIONARY_POLLING_PERIOD;
        present_state = ST_STATIONARY;

    else
        look_up period_increase for step_period;
        step_period = step_period + period_increase;
        present_state = ST_DECELERATING;
    endif;
}
```

Open-loop Stepper Motor Control Algorithm (7)

Possible improvements to this SM control algorithm:

- Provide the ability to change the direction of motion by allowing the entry, at any time, of a destination position that differs from the current position in the opposite direction from the current direction of motion.

- Provide an emergency stop sequence.

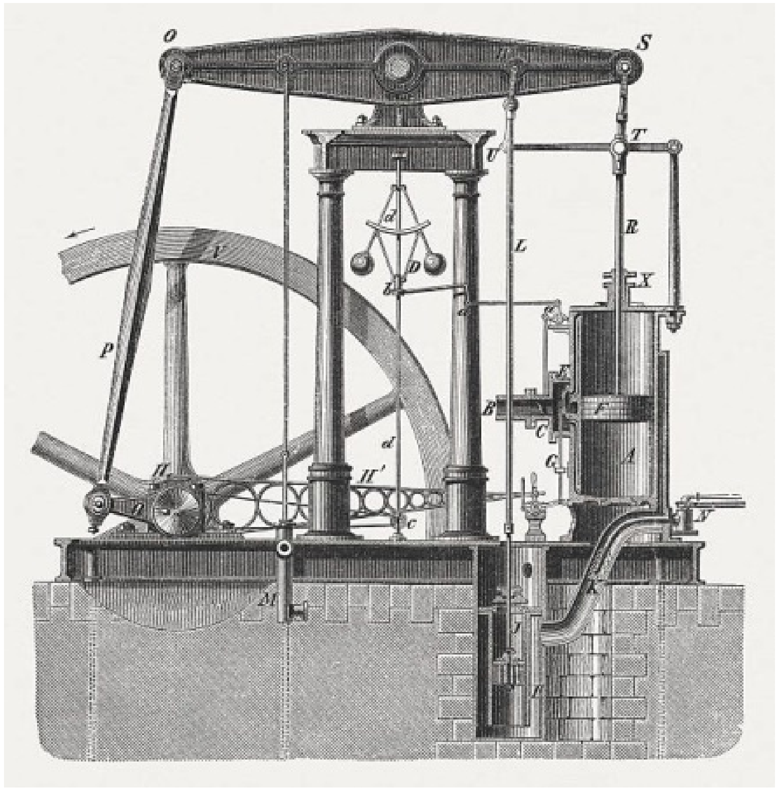
- Allow the slew rate to be changed.

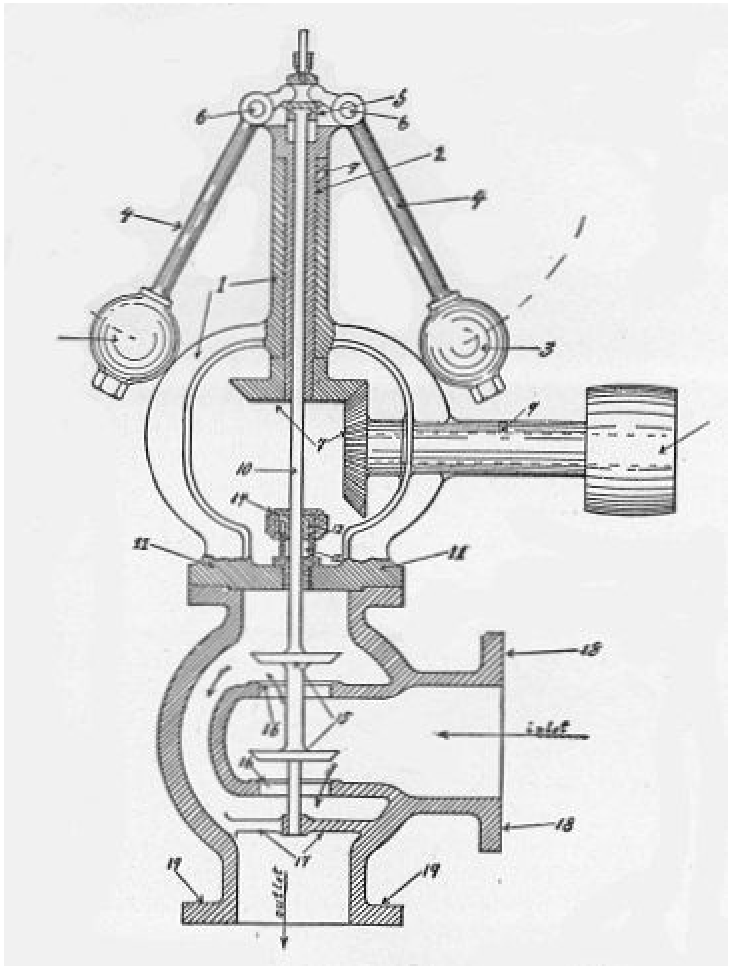
- Provide position feedback and implement a closed-loop control algorithm.

Early Feedback Control Systems

- Embedded systems are widely used today to implement ***feedback control systems***, which are systems that respond to feedback signals and produce control signals that cause a larger engineering system to produce the desired behaviour.
- For example, the Scottish inventor James Watt (1736-1819) made many improvements to Thomas Newcomen's original steam engine. These improvements greatly improved the efficiency and power of the steam engine, which was key to starting the Industrial Revolution in 18th century Great Britain.
- One of these improvements was the ***centrifugal governor*** (1788), in which a rotating pendulum assembly with two iron weights was used to measure the rotational speed of the engine (a ***measured process variable***) to provide feedback to a valve (a ***control signal***) that regulated the flow of steam and thus regulated the engine speed.
- The speed of the engine could thus be automatically kept near to the desired speed (the ***set point*)** for the steam engine.

James Watt’s Centrifugal Governor





From N. Hawkins, *New Catechism of the Steam Engine*, Theo Audel, New York, 1904.

Feedback Control Theory

- Feedback control theory is the branch of applied mathematics that deals with modeling the behaviour of systems with feedback and the design of control algorithms that cause a controller to produce control signals that induce the system to have a desired behaviour.
- Typical desired system behaviours:
  - Stable and efficient operation at a selected set point (e.g., speed, input rate, output rate) despite the presence of perturbations.
  - Fast and smooth transition from one set point to a new set point.
- In 1868 James Clerk Maxwell wrote a classic paper ("On Governors", *Proc. Royal Society of London*, vol. 16, pp. 270-283) that provided a mathematical framework for the centrifugal governor and other control mechanisms that had been developed to control machinery in the first 100 years of the Industrial Revolution.
- In the early 20th century, the problem of steering large boats was influential in the development of modern PID control theory.

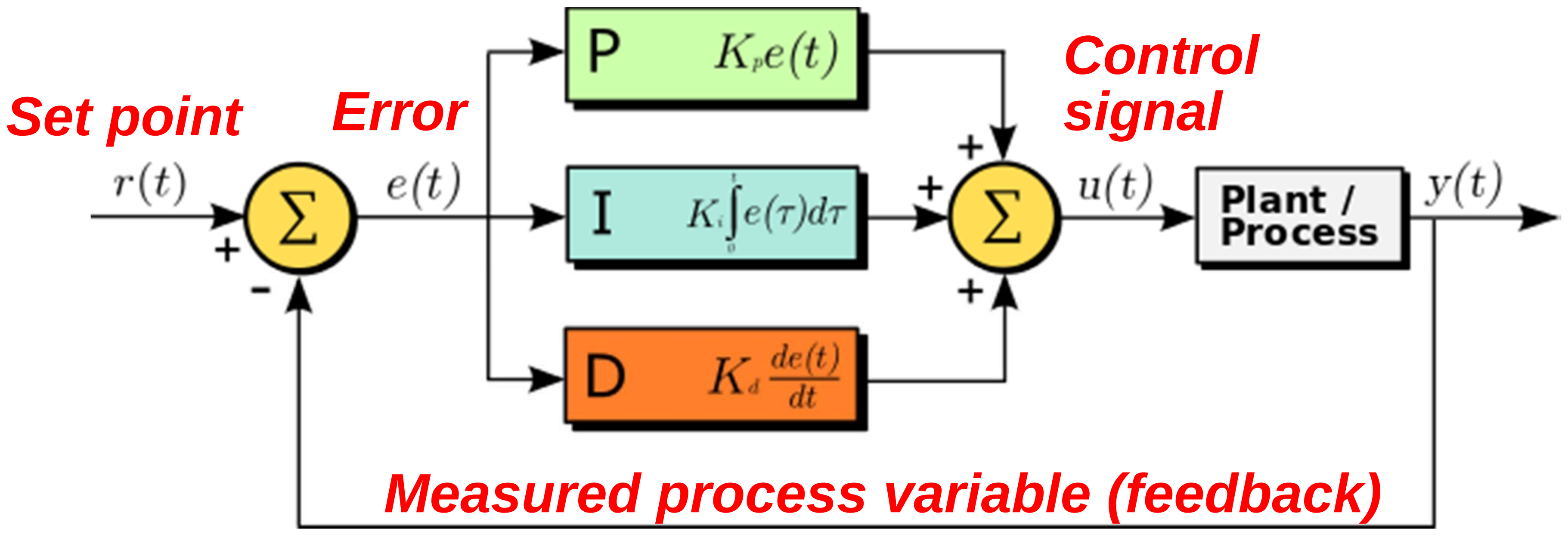
A Control Example: Steering a Large Boat

Consider the problem of using the rudder to change the direction of motion of a boat from one direction (a straight line or an arc) to a new direction (a straight line or an arc):

- **Proportional term:** When the boat is moving in the new direction (the new steady state), the rudder's position is in direct proportion to the new direction (that it, either go straight, turning left, or turning right). A sharper turn requires a greater deviation in the rudder's position.
- **Integral term:** However, the boat is sluggish (slow to respond to a new rudder position) because of its heavy weight (that is, its rotational inertia). As the boat slowly starts to turn, an error builds up between the desired new boat direction and the actual boat direction. The rudder should be oversteered to overcome this error in the boat's direction.
- **Derivative term:** If the boat starts to turn too fast, then a future error can be predicted to build up between the actual position of the boat and the desired position. The rudder should be understeered a bit to slow down the boat's rotation so that the predicted error is avoided.

Proportional-Integral-Derivative (PID) Controllers

- PID Controllers are widely using in industrial controller applications when there is a closed-loop configuration.



www.wikipedia.org

Proportional term: overcomes steady-state ***present error***

Integral term: overcomes accumulated ***past errors***

Derivative term: avoids predicted ***future errors***

The PID Controller Equation

$$u(t) = \mathrm{MV}(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(\tau)\, d\tau + K_\mathrm{d} \frac{de(t)}{dt}$$

$u(t)$	is the ***control signal***, or manipulated variable, that is to be computed. For a stepper motor, $u(t)$ is the step rate and the rotational direction (i.e., the sign of the step rate).
$e(t)$	is the ***error signal***, which is computed as the difference of the desired ***set point*** $r(t)$ and the ***measured process variable*** $y(t)$. For a stepper motor, $r(t)$ is an angular position that is increasing (or decreasing) at a constant rate if the motor is to be rotating at a constant speed.
K_p	is the ***proportional gain***, which is a first tuning parameter.
K_i	is the ***integral gain***, which is a second tuning parameter.
K_d	is the ***differential gain***, which is a third tuning parameter.

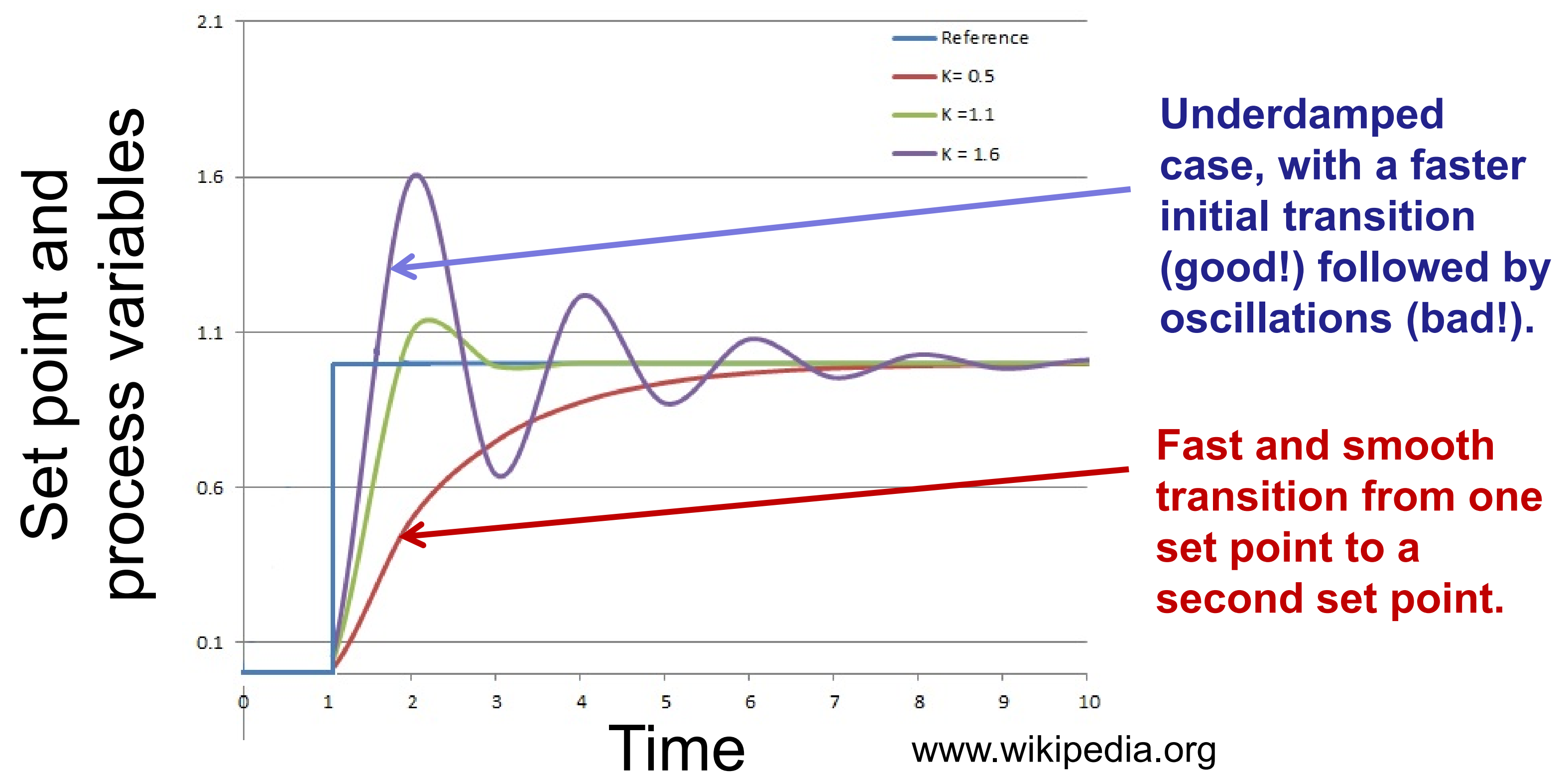
PID Controller Tuning (1)

$$u(t) = \mathrm{MV}(t) = K_{\mathrm{p}} e(t) + K_{\mathrm{i}} \int_0^t e(\tau)\, d\tau + K_{\mathrm{d}} \frac{de(t)}{dt}$$

- PID controller theory is a complex topic that is the subject of many books, papers and entire courses in control theory.
- For a shorter summary consult: K. H. Ang, G.C.Y. Chong and Y. Li, “PID control system analysis, design, and technology,” *IEEE Transactions on Control Systems Technology*, July 2005, vol. 13, no. 4, pp. 559-576.
- The three parameters K_p, K_i and K_d must be tuned to produce acceptable controller performance. We want to avoid large errors and oscillations while still having quick response to new set points. A common strategy is to adjust K_p first, then K_i, and then finally K_d in that order.

PID Controller Tuning (2)

- The figure below shows the change in the measured process variable $y(t)$ when the reference set point r(t) is suddenly changed, for three different values of K_p and with K_i and K_d held at constant values.



www.wikipedia.org

Using a PID Controller for a Stepper Motor

- Any stepper motor control algorithm must take into account practical and safety limits, such as maximum allowed acceleration and deceleration, and maximum allowed speed.
- The basic PID control algorithm must be modified to incorporate these constraints when controlling SMs.
- Closed loop SM controllers will often use a simpler algorithm than a PID controller.
- More complex control scenarios (e.g., control of high-speed robot arms, jet aircraft) will benefit from the superior performance of tuned PID controllers.