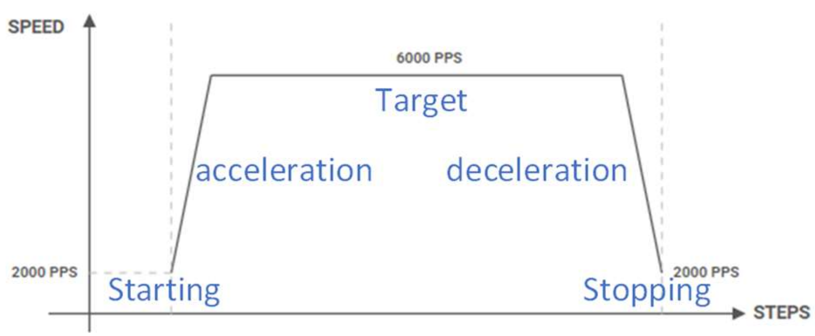
Stepper Motion Control

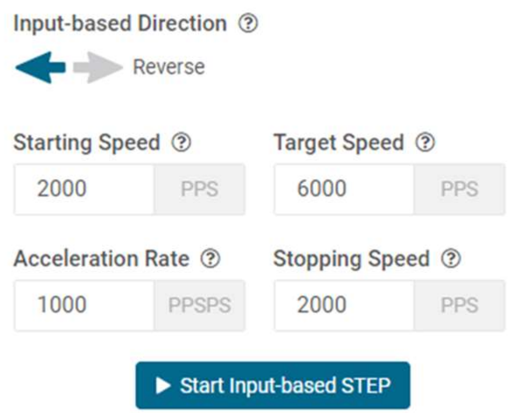
Brushed, Stepper, Solenoid Drivers

Murugavel Raju

Easy Motion Control and Ramp Profile

- Running a stepper from 0 to full speed with a mass (inertia) attached to it will result in lost steps as well as mechanical damage. A trapezoidal velocity ramp profile is commonly used when starting and stopping a stepper to avoid such issues.
- This profile comprises of a linear acceleration segment, a steady state constant velocity segment and a linear deceleration segment. This profile is used in the EVM GUI app.



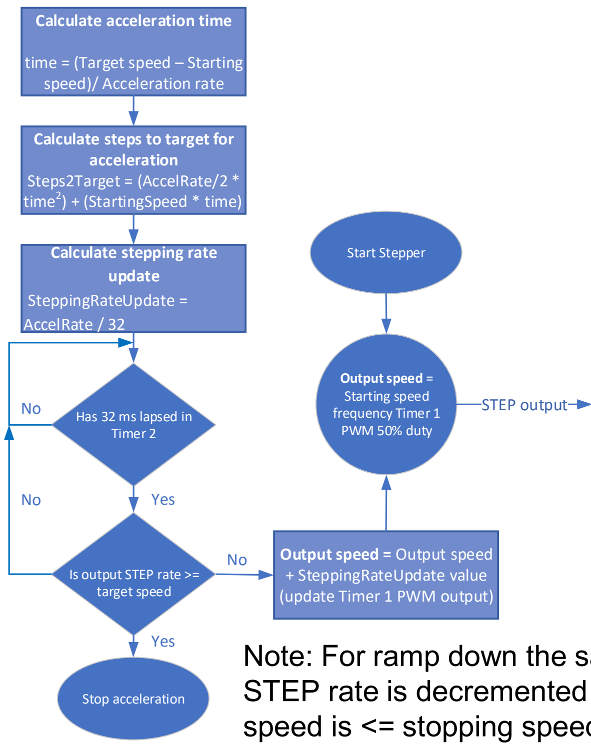


Easy Motion Control and Ramp Profile Implementation

- The user determines the starting and stopping speed in PPS (pulses per second) for their application using calculations based on system inertia and the stepper output torque.
- The acceleration and deceleration (ramp up and down) rate in PPSPS (PPS per second) are also determined by system inertia, desired time and the stepper output torque.
- The target velocity or speed in PPS is determined by the time required for the stepper to reach its target position.
- Ramp profile implementation with an external microcontroller (The EVM uses MSP430).
  - Uses two Timers, one of them with a PWM output.
  - The Timer with the PWM output generates the STEP output frequency with 50 % duty cycle.
  - The second timer is configured for generating a steady timing of 32 ms.
    - Every 32 ms the STEP pulses rate is increased until it reaches the target speed for ramp up.
    - Every 32 ms the STEP pulses rate is decreased until it reaches the stopping speed for ramp down.
  - The EVM GUI uses the acceleration rate value for deceleration rate.
  - The complete MSP430 CCS project and C-source code files for the DRV8462EVM are available for download from ti.com, hyperlink.

Easy Motion Control and Ramp Profile Pseudo Code

Ramp up profile generation



Note: For ramp down the same flow is used. Output STEP rate is decremented every 32 ms until output speed is <= stopping speed.

Pseudo code, double click to open

Source: “stepper.c” of the DRV84x2_DRV82x2_DRV89x2_EVM CCS project

```
// update steps to speed and steps to stop calculations
// only calculate when the motor is idle and commanded to move
time = (float) (DesiredTargetSpeed - StartingSpeed) / AccelRate;
time_squared = time * time;
// calculate steps to target
Steps2Target = ((AccelRate >> 1) * (time_squared))
        + (StartingSpeed * time);
// calculate Steps2Stop if:
//  StepsToStop in GUI is >1 and
//  StoppingSpeed is not equal Starting Speed
//
if ((StepsToStop == 2 && use_one_edge == 1) || StepsToStop == 1)
{
    if (StartingSpeed != StoppingSpeed)
    {   // recalculate time if necessary
        time = (float) (DesiredTargetSpeed - StoppingSpeed) / AccelRate;
        time_squared = time * time;
    }
    // calculate steps required to stop
    Steps2Stop = ((AccelRate >> 1) * (time_squared))
            + (StartingSpeed * time);
}
else    // defined number of steps to stop
{
    Steps2Stop = StepsToStop;
}
// calculate stepping rate update (integer value divide by 32)
if (AccelRate > 31 || AccelRate == 0) // if >=32 or 0
{
    SteppingRateUpdate = AccelRate >> 5; // divide by 32 by right
                                         // shifting 5 positions
}
else
{
    SteppingRateUpdate = 1; // set it to 1 to get correct value
}

if (StartingSpeed >= DesiredTargetSpeed)
// Special case -- override prior settings
// allow motor to attempt to run only at starting speed
{
    SteppingRateUpdate = 0;
    Steps2Target = 0;
    Steps2Stop = 0;
}
// transition from ACCEL to DECEL when count equals(StepsToMove -
   Steps2Stop) <=0
if ((Steps2Stop > (StepsToMove >> 1))
        && (StartingSpeed == StoppingSpeed))
    Steps2Decel = StepsToMove >> 1; // decel at midpoint
else
    Steps2Decel = StepsToMove - Steps2Stop;
```

Thank you.