TRINAMIC Motion Control technology white paper, November 17, 2017

Reducing timer interrupt load in 3D printer applications

In the past years, the development of filament-based 3D printers (short FDM) was driven by technical enthusiasts to an impressive technology. Despite dealing with the limitations of 8-bit AVR controllers, the community was able to develop a CNC like path-controlled motor control using stepper motors in nearly all of the designs.

To bridge the chasm between technical enthusiasts and mainstream users, the technology now has to overcome several restrictions of existing designs and become accessible to the consumer market.

As the majority of the printers is desktop-sized and will be located close to or on top of the operator's desk, which in the future will no longer be in a lab, garage, or the maker's cellar, one of the main objectives is to reduce noise. The primary source of noise in the printer is the motors positioning the print bed, the print head, and the extruder itself. Therefore, the best way to reduce acoustic emissions is to reduce resonances and mechanical vibrations by increasing the microstep resolution. As a positive side effect, less mechanical vibrations will also optimize the quality of the print.

The second main objective of current designs is to increase the speed of printing, requiring higher step frequencies. As these step frequencies are still generated by the microcontroller, the maximum step rate nowadays is limited by the highest timer interrupt rate achievable.

This paper provides an overview of different concepts of step pulse generation and a way to achieve both objectives without increasing MCU size and cost, by making use of dedicated motion control peripherals integrated in today's stepper motor controllers.

Background

Typical FDM-type 3D printer firmware on the market for open source solutions uses a principle of operation that's very similar to the principle used by closed source commercial solutions – a central MCU handles communication on a USB or UART port, reads data from a memory card, in some cases controls a display for status information and controls 4 or more stepper motor driver ICs. The typical interface to these drivers is step and direction. The motor driver IC calculates microstep currents and controls and drives the motor current directly.

TRINAMIC
MOTION CONTROL

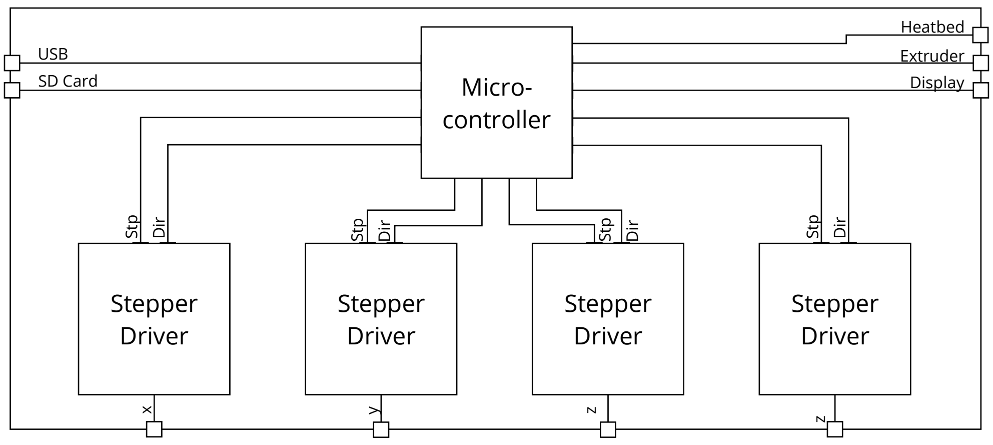
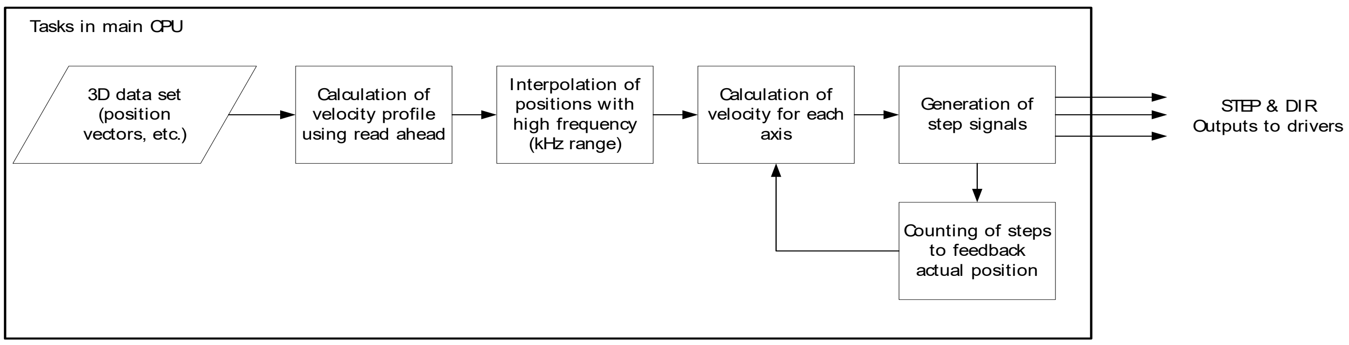


Fig. 1 Typical system diagram of a 3D printer with software-based ramping

In a 3D printing application, at least the 3 axes must be path-controlled: X, Y, and extruder. This means the motors must be synchronized. Not only for the target positions but for every coordinate in a motion segment. All coordinated axes must be calculated and controlled closely at the same time and step pulses for all of those have to be generated synchronously – so the maximum interrupt frequency limits the highest achievable pulse rate and indirectly the highest achievable printer speed.



Tasks in main CPU
3D data set (position vectors, etc.)
Calculation of velocity profile using read ahead
Interpolation of positions with high frequency (kHz range)
Calculation of velocity for each axis
Generation of step signals
Counting of steps to feedback actual position
STEP & DIR Outputs to drivers

Fig. 2 Typical partitioning of motor control functions in 3D printers

Challenges with software-based step pulse generation

Generating step pulses using a fixed interrupt frequency falls apart in two cases:

1. $f_{step} < f_{interrupt}$ but close to $f_{interrupt}$
2. $f_{step} > f_{interrupt}$

Case 1: $f_{step} < f_{interrupt}$

Generating a step pulse frequency from a fixed frequency timer that does not divide the desired frequency without remainder causes a beat as shown in Fig. 3. Thus, the actual step frequency generated by this solution changes all the time. As the step frequency translates into the commanded motor velocity, this results in continuous accelerations and decelerations, requiring additional torque.

Velocity error

Depending on the ratio of f_{step} to $f_{interrupt}$, the absolute timing error (T_{error}) is between a minimum of 0 – in case the interrupt period ($T_{interrupt}$) divides the commanded step period ($T_{interrupt}$) without remainder – and a maximum of just below $T_{interrupt}$.

With a step width of x_{step}, the resulting velocity is:

$$v_{res} = \frac{x_{step}}{T_{error} + T_{step}}$$

And the commanded velocity:

$$v_{target} = \frac{x_{step}}{T_{step}}$$

The relative velocity error results in:

$$v_{error_{rel}} = \frac{v_{res}}{v_{target}} = \frac{\frac{x_{step}}{T_{error} + \Delta T_{step}}}{\frac{x_{step}}{T_{step}}} = \frac{1}{\frac{T_{error}}{T_{step}} + 1}$$

With a maximum error of:

$$T_{error} \rightarrow \frac{1}{\mathrm{f_{interrupt}}}$$

The absolute velocity error will tend to:

$$v_{error} = v_{target} \cdot v_{error_{rel}} = v_{target} \cdot \frac{1}{2}$$

Assuming an achievable interrupt frequency of e.g. $1kHz$, the maximum timing error will be in the range of $1ms$.

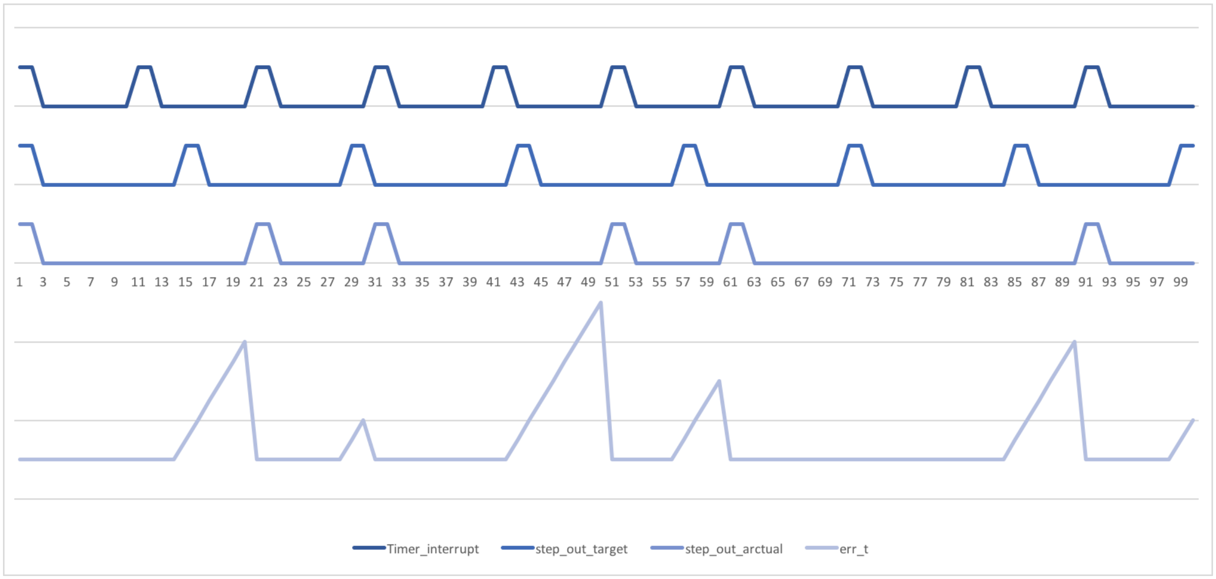


Fig. 3 Timing diagram with timer to pulse ratio of 10/14

Case 2: $f_{step} > f_{interrupt}$

As with an interrupt frequency of $1 \ldots 3kHz$, the maximum step pulse rate is quite limited and does not allow for high speed and medium to high microstep resolutions. To overcome this limitation the software solutions use a concept that is called “double step”. If the intended step frequency is higher than the interrupt frequency, multiple step pulses are generated in one interrupt service routine, resulting in a short sequence of steps with a fixed frequency at the beginning of this routine with a following break until the next interrupt service routine starts.

Separating calculation and execution

As an alternative to software-based solutions, pulse and ramp generation can also be generated by dedicated peripherals such as external ramp generators.

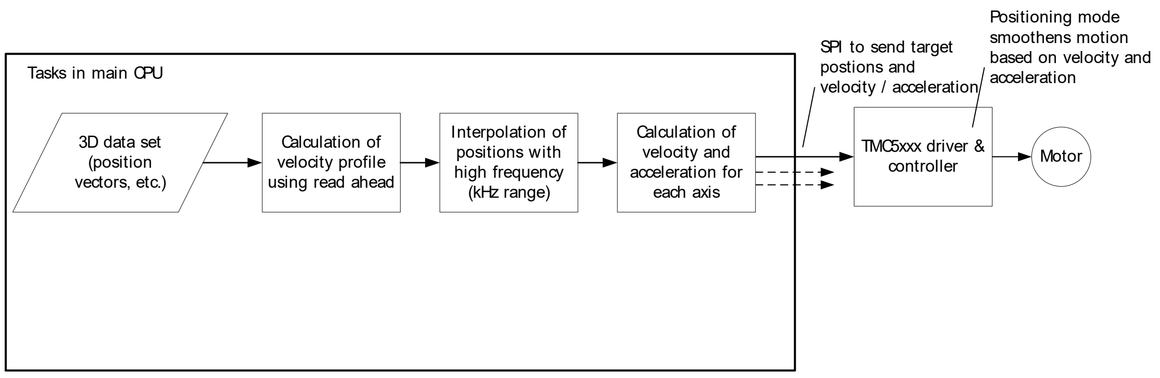

Tasks in main CPU
3D data set (position vectors, etc.)
Calculation of velocity profile using read ahead
Interpolation of positions with high frequency (kHz range)
Calculation of velocity and acceleration for each axis
SPI to send target postions and velocity / acceleration
Positioning mode smoothens motion based on velocity and acceleration
TMC5xxx driver & controller
Motor

Fig. 4 Partitioning in a cDriver-based 3D printer

Firmware design considerations and implementation using the TMC5130 cDriver

To offload the pulse generation to a component with a dedicated ramp controller, the microcontroller has to precalculate the parameters of the ramps of all coordinated axes before the segment can be executed. All whilst the peripheral component takes care of the real-time task of executing the actual ramps and generating the high-frequency step pulses.

In this case, the microcontroller has to take care of the following precalculations:

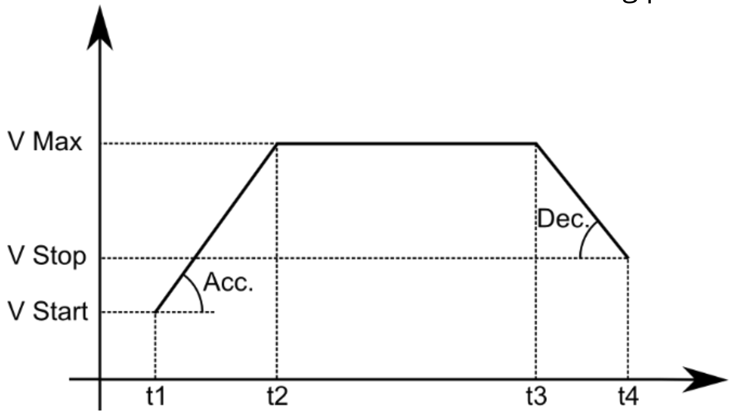


Fig. 5 Typical ramping profile utilized in 3D printers

Acceleration phase:

$$\mathrm{t}_2 - \mathrm{t}_1 = \frac{V_{Max} - V_{Start}}{\mathrm{Acceleration}}$$

Plateau:

$$\mathrm{t}_3 - \mathrm{t}_2 = \frac{\mathrm{PlateauDistance}}{\mathrm{V}_{Max}}$$

Deceleration phase:

$$\mathrm{t}_4 - \mathrm{t}_3 = \frac{V_{Stop} - V_{Max}}{Deceleration} \overset{acc=dec}{\Longleftrightarrow} \frac{V_{Max} - V_{Stop}}{Acceleration}$$

The only time critical operation left in the microcontroller is to start the new segment just before the last segment finishes.

At t_1, the processor has to write the acceleration in A_{max} and D_{max} registers, the initial, maximum and final speeds in V_{Start}, V_{Max} and V_{Stop}, and finally the final position into X_{target}. The timer must be set here to the sum of the three movement phases.

Timing considerations

Timing for the next ramp segment can be generated in two different ways.

Starting the ramp segment with a timer interrupt

As the expected ramp times have been calculated in preparation to the movement, ($t_1 + t_2 + t_3 + t_4$), this information can easily be used to preload a timer interrupt to trigger the transmission of the next segment. Since this interrupt only occurs once per segment, the overall timer interrupt load is reduced by the segment length.

Making use of the position interrupt output of the TMC5130 ramp controller

By loading the `X_COMPARE` register of one of the ramp controllers with a value of a few microsteps lower than the target position of the segment, the controller will issue a position-reached interrupt at `SWP_DIAG1`. This will leave enough time to safely update all motion controllers with the next segment so the next motion segment will start during the deceleration ramps.

When the motion controller receives the new target position in time before the last segment is finished, it will automatically take care of the transition from one segment to the following. This allows for a timing-uncertainty of a few microsteps without influencing print quality.

Concern: time lags caused by serial commanding several axes via SPI

A potential time lag between the updated data of the different motion controllers will be in the microseconds range. Even for high step-frequencies like 256 microsteps per full step, this results in less than one microstep latency. It could also be pre-compensated for, as the time sequence of the SPI transfers is well-known.

Programming a register within the TMC5xxx family driver requires 40 SPI bits to be sent. Assuming a $4MHz$ SPI rate, which is well below the limit of the IC, this will require about $10\mu s$, which is well below the median error achieved by step direction calculations and does not accumulate.

This way, several ICs and parameters can quickly be set and the delay between the axes to the different controllers is negligible. It even can be compensated for by software. As an option, to achieve a real zero delay, all drivers can be put into a single SPI chain. However, this will require an n-times 40-bit SPI datagram to be sent for each access, even when accessing only one single driver.

Implementation using TMC5130 cDriver

See Application Note 046 - Adapting step/dir CNC firmware for ramp generators.

Conclusion

Making use of dedicated ramp controllers can reduce the MCU load, improve print quality and make your printer faster and more silent with no additional components required and at no additional or even lower cost than systems with comparable performance.

Using dedicated ramp controllers solves technical restrictions such as vibrations and speed limitations, and reduces software design risks and development time. Technical products usually enjoy several firmware updates and development steps during their lifecycle, so the same reduction in design and test efforts applies to all future releases as well, reducing the total cost of ownership significantly.

About TRINAMIC Motion Control

Instead of semiconductor makers that just want to sell more chips, TRINAMIC Motion Control (TMC) began with engineers who wanted to design better drives. Its founders knew that designing motion control technology in hardware, not software, was essential for motors that work efficiently, reliably, smoothly, and quietly.

Headquartered in Hamburg, Germany with subsidiaries in Tallinn, Estonia, and Chicago, IL, USA, the fabless semiconductor company's own motion control architecture divides power electronics into three building blocks: a microcontroller for digital processing, a driver that translates digital signals into electrical power, and a gate driver plus power switches. This simplifies development and makes it easier to design components best suited for each block. In 2015, TMC became the first supplier worldwide with an EtherCAT-compliant slave-controller IC for integrating complex, real-time, latency-free I/O peripherals.

TMC now has over 20 years of application experience in embedded motion control. It develops and sells ICs, modules, and integrated mechatronics to market-leading manufacturers worldwide. Examples include Agfa, BQ, Dremel, ConnectM, GE, Honeywell, KUKA, Markforged, Össur, Xerox, and Zeiss. The company's products have been integrated into a wide range of applications in digital manufacturing, biotechnology, lab automation, materials handling, and factory automation.