RECONFIGURABLE COMPUTING FOR HIGHER ORDERS S-CURVE MOTION CONTROL

Brian Tapia

*ECE: Electrical & Computer Engineering*

Pomona, CA, United States of America

[btapia@cpp.edu](mailto:hpluc@cpp.edu)

Bryan J. Banta

*ECE: Electrical & Computer Engineering*

Pomona, CA, United States of America

<bjbanta@cpp.edu>

Hao Luc

*ECE: Electrical & Computer Engineering*

Pomona, CA, United States of America

[hpluc@cpp.edu](mailto:hpluc@cpp.edu)

Spencer D. Moore

*ECE: Electrical & Computer Engineering*

Pomona, CA, United States of America

[sdmoore@cpp.edu](mailto:sdmoore@cpp.edu)

John Alexis S. Villegas

*ECE: Electrical & Computer Engineering*

Pomona, CA, United States of America

<jsvillegas@cpp.edu>

Dr. Mohamed El-Hadedy

*ECE: Electrical & Computer Engineering*

Pomona, CA, United States of America

[mealy@cpp.edu](mailto:mealy@cpp.edu)

*Abstract*—Electric motors are as ubiquitous as knives nowadays. Manufacturing, transportation, and a handful of other sectors in the modern economy rely on this useful tool to increase productivity and get things done. Nonetheless, this tool would not run without a motion controller configured with a certain motion profile. There are a few motion profiles used across the industry. There is the triangular motion profile, which is used for speedy pick and place applications and the trapezoidal motion profile used for dispensing, measuring, and machining. Both of these motion profiles are widely used in the industry for their speed which greatly increases production yields. However, it is important to point out that there are applications which requires great emphasis on positional accuracy as opposed to speed. For this type of application, the S-curve motion profile is used which is known to produce minimal vibrations upon movement.

The purpose of this project is to implement a reconfigurable computing motion controller that can implement two orders of S-curves using a Field Programmable Gate Array (FPGA) in a switching schematic so there is no need for modification of the structure of the code. The structure of the code is further enhanced by implementing a Microblaze softcore processor system. The VHDL system is implemented on a PYNQ Field Programmable Gate Array and will be utilizing a UART connection from the terminal which is interfaced with the board’s data input. The analysis of this project will focus on the demand of resources, actuation performance for various orders of s-curve motion profiles. This report will give an overview into the algorithm and the implementation process of the hardware and system.

Keywords—VHDL, S Curve, Jerk, FPGA, position, kinematic, low-latency, dynamic power, static power, reconfigurable computing

# Introduction

Suppose a motor is being used to drive the gears of a conveyor belt. In its most basic form, a motion controller would be the mechanism responsible for the action of turning on the motor, so that the items on the belt move from one location to another, at which point the motor would be turned off. Without accounting for velocity control, the motor will experience dramatic changes in acceleration as it immediately attempts to reach maximum velocity from a resting position. This becomes a reliable source of vibrational disturbances. This may not always be a point of concern, however for many applications those vibrational disturbances have the potential to compromise the task the motor has been assigned. For this reason, among others, a more refined system is required.

The objective of this project is to design, implement, and analyze a motion controller with an adaptable velocity control system. A velocity control system is used to enhance the performance of a motor, specifically by reducing vibrations associated with rapid changes in acceleration. Furthermore, an adaptable velocity control system is one that can implement various motion control profiles depending on the sensitivity of the task. To accomplish such functionality, a reconfigurable computing platform is desired.

# Motion profile characteristics

A motion profile is a graphical tool that displays the behavioral information of a motor. There is a wide variety of motion profiles that a motion controller can be assigned. The more common motion profiles used by motion controllers are: Triangular, Trapezoidal, and S-Curve.

## Triangular Motion Profile

This motion profile provides the fastest movement out of all the motion profiles enumerated above. It is characterized by a constant acceleration phase and a constant deceleration phase. Fig. 1 shows the velocity, acceleration, and jerk characteristics of the triangular motion profile. It is observed that this motion profile has a triangular shaped velocity curve, hence the name. Notably, this motion profile suffers from infinite jerk at the start, middle and end of the movement which translates to unwanted vibrations not desirable for accurate positional movements.

## Trapezoidal Motion Profile

This motion profile provides a relatively smoother and slower movement compared to the triangular motion profile. As seen in Fig. 2, the shape of the velocity curve is a trapezoid with three stages of motion, a constant acceleration phase followed by a constant speed phase which is then ended by a constant deceleration phase. Like the triangular motion profile, it also undergoes infinite jerk at the start, middle and end of the motion which also produce unwanted vibrations during termination of the movement.

A screenshot of a cell phone

Description automatically generated

Fig. 1. Triangular motion profile characteristic curves.

A close up of a logo

Description automatically generated

Fig. 2. Trapezoidal motion profile characteristic curves.

## S-Curve Motion Profile

Compared to the motion profiles mentioned earlier, this profile is the slowest but the most accurate out of the three. Fig. 3 shows the velocity, acceleration, and jerk curves of an S-Curve motion profile. Noticeably, the jerk is finite at all phases of the motion, from start to end. Due to that, vibrations during the movement is kept at a minimal level. Furthermore, increasing the order of the S-Curve results in a trapezoidal shape of the jerk which results in smoother acceleration of movement and in smoother movement overall [1]. Application-wise, this motion profile is ideal for movement that requires accurate positional movement which decreases wait times for end position stability. This, in turn reduces movement time which could increase overall productivity.

A close up of text on a black background

Description automatically generated

Fig. 3. S-Curve motion profile characteristic curves.

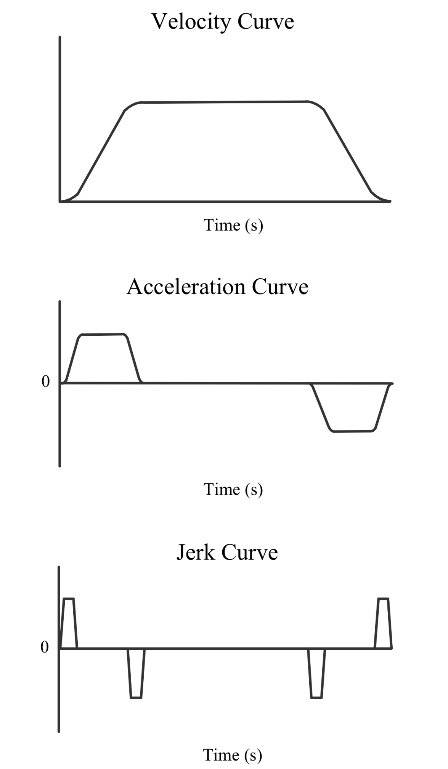


Fig. 4. Higher Order S-Curve motion profile characteristic curves.

# S-Curve Implementation

To implement the S-Curve motion profile, the motion controller needs to be equipped the software technology capable of calculating the velocity of the motor according to the motion profile curve. For this project, the microcontroller is implemented on an FPGA, and the programming language used is VHDL. The S-Curve was implemented by programming into the motion controller a set of mathematical equations. A different equation was needed for each stage of the motion profile. A counter was used to act as the time variable in the equations. The counter used in this project operated at a frequency of one kilohertz. This means that the velocity signal being sent to the motor was updated every thousandth of a second, or one millisecond.

Multiple orders of S-Curve orders can be implemented into one motion controller under this schematic. Each order of S-Curve is segmented into phases and each phase is characterized by a different equation. The number of phases for any given order of S-Curve is found by exponentiating two by the order of the S-curve and then subtracting by one (e.g. a 4th order S-curve will have 24 – 1 = 15 different phases.) In addition to the exponential increase in the amount of equations for increasing orders of the S-Curve, the order of the equations themselves increase.

This becomes problematic when attempting to program a motion controller with many orders of S-Curves. Such problems experienced when implementing multiple orders of S-Curve motion profiles onto the FPGA based motion controller were due to timing constraints as well as physical resource limitations. It was found that meeting certain timing constraints was difficult with higher orders of S-Curves since the equations required to generate the velocity themselves required too much processing time. This led to negative time slack in the system, which compromised the functionality of the motion controller. Additionally, each S-Curve motion profile took up a considerable amount of space in the FPGA. The amount of physical resources required to support each S-Curve depended on the order of each S-Curve, where higher order S-Curves required more resources. The increase in resource demand was due to the increase in resources needed to process the equations to calculate velocity.

To avoid the problems discussed above, the solution implemented in this project was to limit the number of S-Curve motion profiles programmed into the motion controller. Due to the space shortages, the number of S-Curves that can be programmed into the motion controller is two.This was the chosen solution since it required no compromise in the execution of the S-Curve.

# Hardware design

In order to quantify the effectiveness of the designed S-Curve motion controller, an enclosure with a moving platform was built. The hardware design was inspired from today’s cartesian printers which simplified the overall design process and helped reduce the cost due to the wide availability of parts such as the aluminum extrusion, belt, nuts, screws, angle brackets and precision rods. Furthermore, a lot of the parts in the assembly was designed using Tinkercad and created using a 3D printer which helped drive down cost even more. This also eliminated problems regarding improper fitting.

To move the platform, a stepper motor (4209L-01DE-RO) was selected due to its accuracy in open loop positional movements requiring no sensor for feedback. In addition, a stepper motor driver (A4988) was chosen because of its easy implementation and cost effectiveness. The resolution of the driver was configured at full step with a full step angle of 0.9°. Finally, an MPU6050 accelerometer was mounted on top of the moving platform to record acceleration at a certain time interval during the motion. This is then used to calculate the magnitude of jerk during movement of the platform which will serve as the metric of effectiveness for the designed S-Curve motion controller.

# FPGA and Microblaze design

FPGA is utilized in this project because of its reconfigurable computing. Most of the design code can be done in the Programmable Logic Block in Fig. 5. VHDL is chosen to be the main language that is used in this project. Because of its flexibility, the design ideas and coding phase is implemented, and those blocks can be tested in the simulation environments to verify the expected results from the design phase. If something unexpected happens, those blocks can be modified and put to test again until the expected results are achieved.

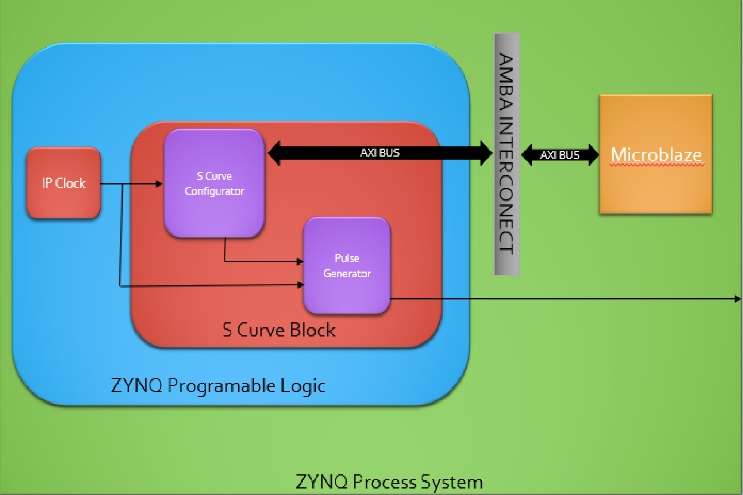


Fig. 5. HDL block diagram.

## S Curve Configurator

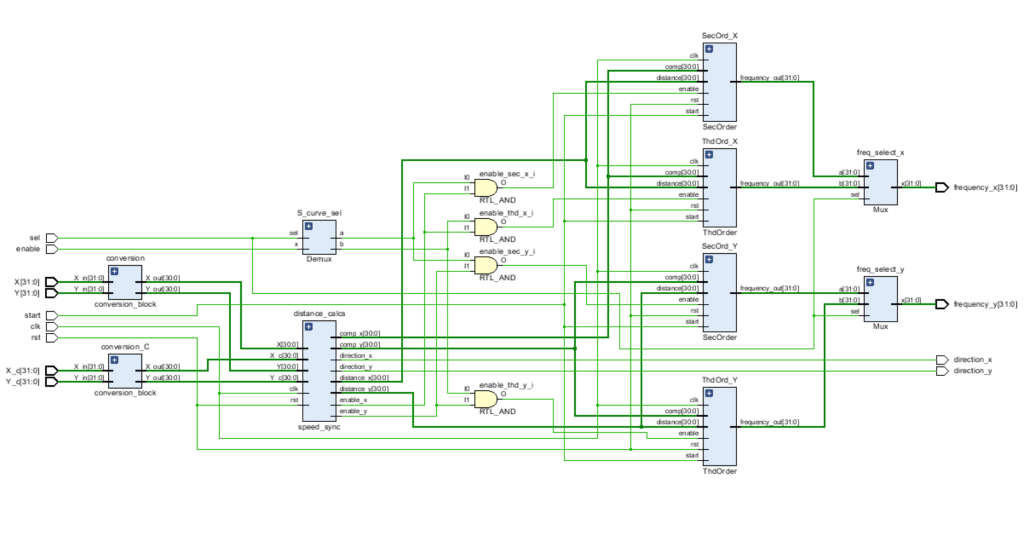


Fig. 6. Schematic for S-Curve Configurator

## Pulse Generator

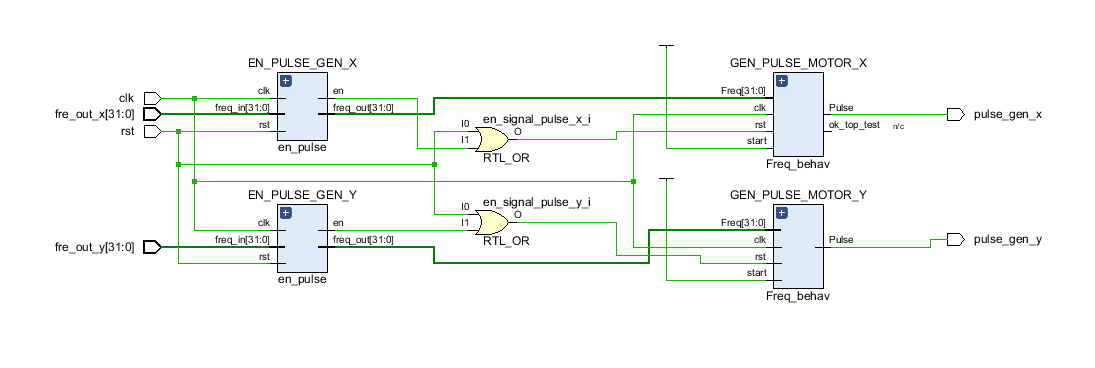


Fig. 7. Schematic for Pulse Generator

In order to run the stepper motor, the FPGA board needs to send out multiple pulses ½ duty cycle with different frequencies. The main function of the pulse generator is to generator pulses with the frequency corresponding to input frequencies from S Curve Configurator (SCC). In first phase of design the pulse generator, the structural model is used to design the block. The design of the pulse generator for the behavioral model also perform to compared to the structural model. Surprisingly, the structural model uses more resources than the behavioral model. Hence, the behavioral model is applied to the design.

The pulse generator makes used the division block [2] instead of division straight from Vivado because it consumes a lot of resources. The percentage difference of the resource can go up to 80% if the division block is used. In addition, the pulse generator can be modified to change to PWM generator easily, if the motor is not stepper motor.

Within the pulse generator, there is an enable block to make sure that the generator will stop sending pulses to the motor because the pulse generator performs incorrectly when SCC sends zero frequency. The design of the enable block is based on D flip-flop, where the frequencies frequency from SCC will pass through the enable block to go to the pulse generator. If the enable block detects zero frequency, there will be a signal send out to disable the pulse generator. There are two pulses which are x and y in Fig.6 to control two motors, which are essential to achieve two-dimensional motion in the cartesian. The main concern is to make sure that speed of the clock feed to the pulse generator need to be at least 10 times faster than the SCC because the division block needs a couple of clock cycles to finish the division after receiving the frequency from the SCC.

The Microblaze design of the project is to implement a soft-core processor on the PYNQ-Z1 or NEXYS4 DDR FPGAs instead of having the overhead computation of using the Jupyter interface. Using Xillinx SDK 2018.3, the design is capable of reading and writing to registers through a custom AXI module that is connected to the S-curve code. This means that a user could send the necessary inputs through the terminal prompt such as direction and distance which in turn would send and calculate the required frequency signals to the stepper motor control wires. The host computer would be able to move the motors by programing the FPGA and running the C code that interfaces with the registers and not have to use the default interface of the PYNQ board.

Unfortunately, as of right now there are timing constraints that are not being met do to limitations in the performance of our current design. There is success in running and testing the motion profile on both our PYNQ and NEXYS4 DDR FPGA boards with modules containing only the S-curve algorithm. The Microblaze interface also correctly

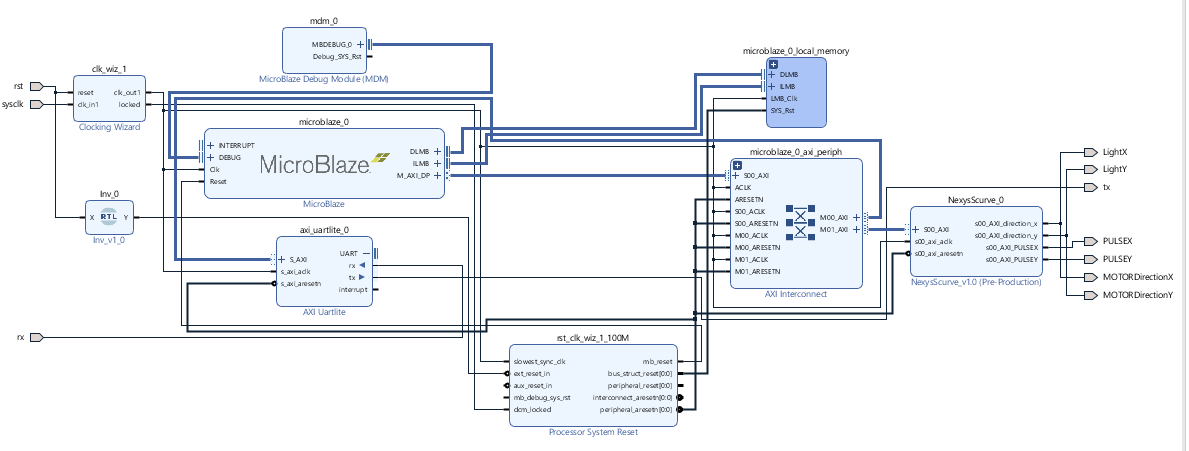


Fig. 7. Microblaze Block Diagram

# Technical difficulties

Troubleshooting the stepper motor driver was time consuming. The first issue was the Voltage supplied to the driver was twice the amount of what was needed. Finding the limiting current of the driver was also an issue. A driver bought from different brands would need different reference voltages (VREF). This equation would be found on the data sheet from the driver. Having an elevated structure with the use of aluminum and mounting the motor directly to the frame produced unwanted vibrations at lower frequencies. These vibrations will be sensed on the platform where the object was placed and gave some inaccuracies when looking at the accelerometer data.

# Motion controller performance

Obtaining the jerk performance metric requires measuring accelerometer values along two dimensions, X and Y-axis. After the measurement of these acceleration values, the overall acceleration was then obtained using Pythagorean theorem, and the magnitude for jerk is then calculated. Fig. 7 shows the acceleration and jerk for the S-Curve motion control, and Fig. 8 shows the acceleration for the Higher order S-Curve motion control. As observed from the figures, the acceleration curves for both configurations of S-Curve are similar, resembling a positive plateau at the start of the motion and an inverted plateau at the end. Furthermore, the jerk graph for both configurations also shows similar pattern having a finite amount of jerk across the motion. Performance-wise, there seems to be no difference between the two configurations.

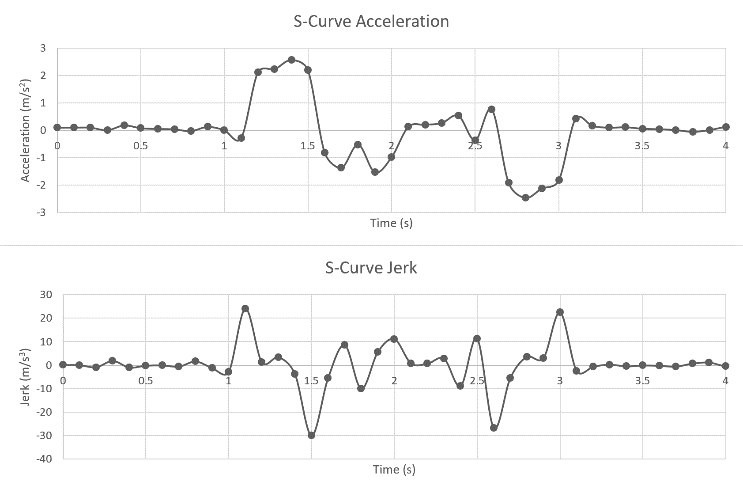


Fig. 8. S-Curve acceleration and Jerk.

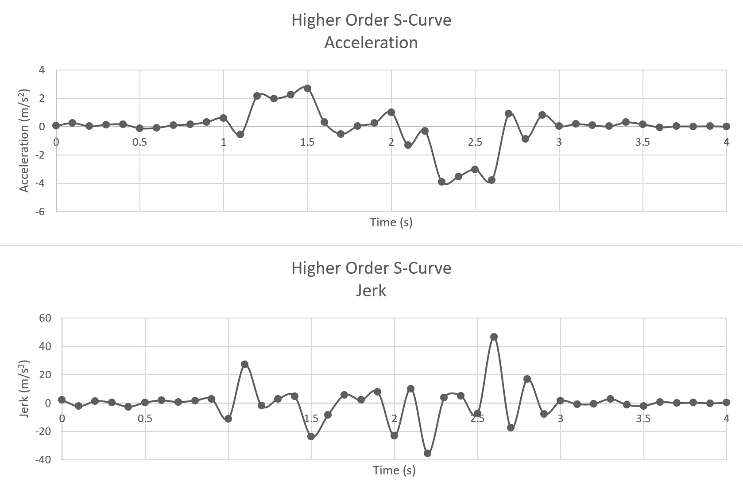


Fig. 9. Higher order S-Curve acceleration and Jerk.

# Future Improvements

Currently there are issues with timing and hardware space with implementation of the S-curve module and MicroBlaze. They both work independently in the current setup but merging them together causes many issues with timing. Improvements in the future would be dedicated hardware optimization of the S-curve algorithm so that timing constraints can be readily made. That way the MicroBlaze interface can be fully utilized in the system and for future versions that can handle more and higher orders of the S-curve. The current hardware set up also has the potential to be implemented as a three-dimensional platform so future projects can expand on this set up without having to create an entire new base like our project.

##### References

1. K. Nguyen, T. Ng, and I. Chen, “On Algorithms for Planning S-Curve Motion Profiles,” in International Journal of Advanced Robotic Systems, vol. 5, pp. 99-101, 2008.
2. “32-Bit Unsigned Divider in Verilog.” *FPGA Projects, Verilog Projects, VHDL Projects - FPGA4student.Com*, [www.fpga4student.com/2016/12/32-bit-unsigned-divider-in-verilog.html](http://www.fpga4student.com/2016/12/32-bit-unsigned-divider-in-verilog.html).
3. Wimberly,, Daniel, and Sean Coss. “ARTY MICROBLAZE SOFT PROCESSING SYSTEM IMPLEMENTATION TUTORIAL1Arty MicroBlaze Soft Processing SystemImplementation Tutorial.” [*Http://Www.ee.nmt.edu*](Http://Www.ee.nmt.edu), www.ee.nmt.edu/~erives/554\_17/MicroBlaze\_Arty.pdf.
4. Vivado Design Suite User Guide: Using Constraints, Xilinx, 2018. Accessed on: Feb. 30, 2020. [Online]. Available: <https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug903-vivado-using-constraints.pdf>
5. M. Bastani, A. DomoyoDaniel, K. Chang, M. Abdal, and D. Avalos, “2nd Degree S-Curve Motion Controller with Definable Kinematics,” Senior Project Report, California Polytechnic State University, Pomona, Ca, 2019.
6. H. Li, "A Jerk-constrained Asymmetric Motion Profile for High-speed Motion Stages to Reduce Residual Vibration". University of New South Wales, Sydney, Australia, 2012.