## 1. Overview

The following is an ongoing motion controller project using an S-Curve algorithm, implemented on a PYNQ FPGA board. The goal of the project is to create a universal motion controller compatible with any motion-controlled system or device, tackling the issues found in industrial and open-source motion controllers while also being relatively inexpensive and easy to expand for the creator community.

## 2. What is the S-Curve motion profile

Many of the industrial motion controllers found on the market use a trapezoidal motion profile with a linear acceleration and deceleration phase. However, due to the sudden change in acceleration during the different points shown in Fig. 1 this causes a jerking motion which results in unwanted oscillation in the controlled system. In the S-Curve algorithm, the acceleration and deceleration phases are implemented using a 2nd degree or higher polynomial to reduce this jerking motion and produce smoother motion. The higher the order of the S-Curve, the smoother the resulting motion will become albeit at the cost of additional resources.

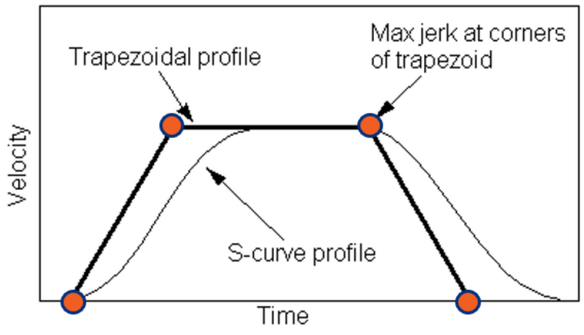

Figure 1: Trapezoidal vs S-Curve (velocity vs time)

The S-Curve algorithm itself has 5 major components as shown in Fig. 2. The 1st and 2nd sections consists of an increasing, positive acceleration phase followed by a decreasing, positive acceleration phase. The 3rd section is the steady state where the acceleration is 0 and the velocity is at its max. The 4th and 5th sections are the inverse of the first two with an increasing, negative acceleration followed by a decreasing negative acceleration until the system reaches its destination. It should be noted that these sections can be further subdivided into additional sections which will help smooth out the motion further.

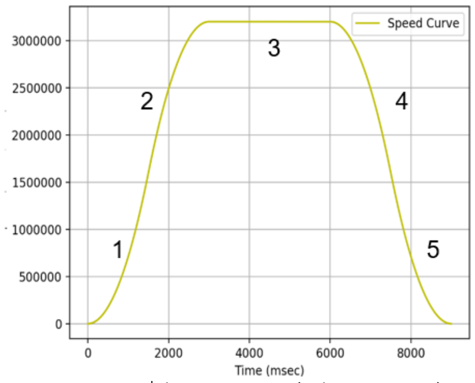

Figure 2: 2nd degree S-Curve (velocity vs time)

## 3. Implementation

The implementation of this controller required three major components, the S-Curve algorithm block (blue), the FIFO block (orange), and the Supervisor block (yellow) which contained the pulse generation to the target device driver. The following is an example of how the S-Curve algorithm could be modeled in code for the algorithm block.

```
long Ta = 3e3;      // acceleration time (milisec)
long Td = 3e3;      // deceleration time (milisec)
long Ts = 3e3;      // Steady state time (millisec)
long Vm= 3200e3;    // Steady state velocity (pulse/milisec) <-Max Velocity
float J=0;//Jerk
float a=0;//acceleration

if(t>=0 && t < (Ta/2)) //1st half of acceleration
       v=J*pow(t,2)/2;
else if(t>(Ta/2) && t < Ta) //2nd half of acceleration
       v=Vm/2+a*(t-(Ta/2))-J*pow(t-Ta/2,2)/2;
else if(t > Ta && t< Ta+Ts) //Steady State
       v=Vm;
else if(t>=Ta+Ts && t<Ta+Ts+Td/2) //1st half of deceleration
       v=Vm-J*pow(t-(Ta+Ts),2)/2;
else if(t>= Ta+Ts+Td/2 && t< Ta+Ts+Td) //2nd half of deceleration
       v=Vm/2-a*(t-(Ta+Ts+Td/2))+J*pow(t-(Ta+Ts+Td/2),2)/2;
```

The FIFO block of this implementation version is used because the frequencies of the S-Curve and Supervisor blocks are different. It acts as a buffer so that the S-Curve will be compatible with the pulse generation in the Supervisor. The Supervisor itself is a combination of several features with the pulse generator or PWM as the most important as it converts the frequency of the results from the S-Curve into a pulse for the driver.

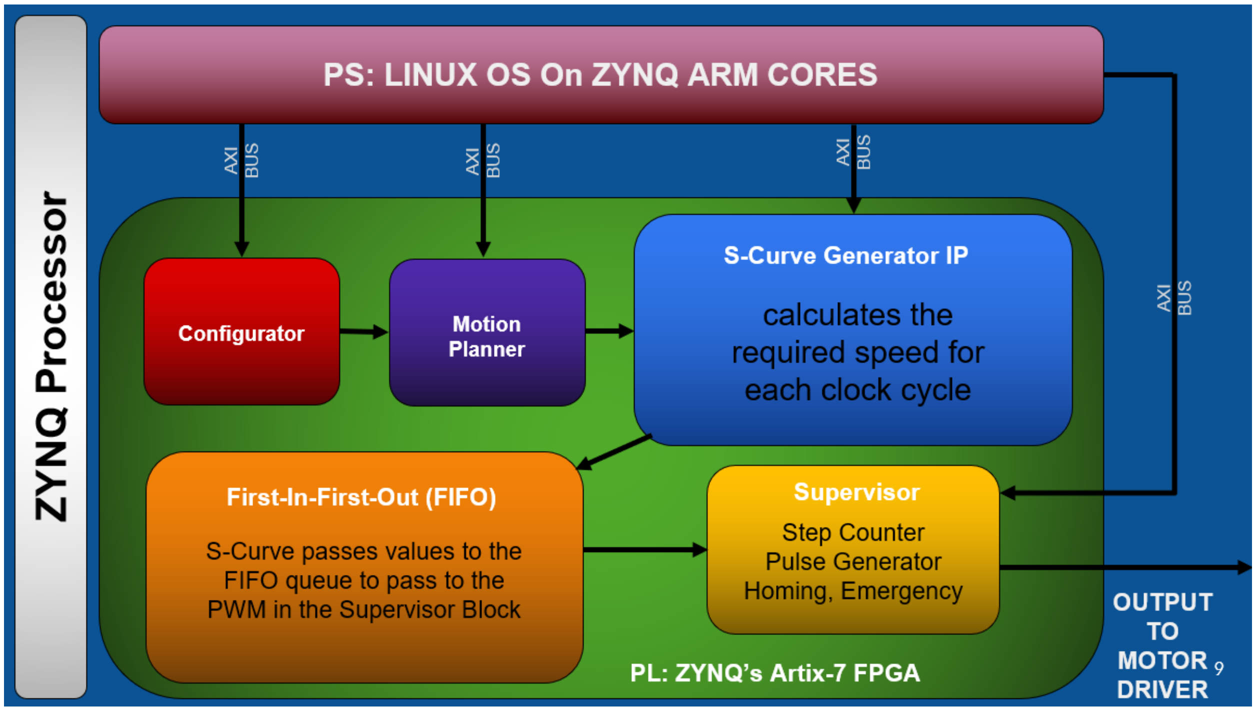
ZYNQ Processor
PS: LINUX OS On ZYNQ ARM CORES
AXI BUS
AXI BUS
AXI BUS
AXI BUS
Configurator
Motion Planner
S-Curve Generator IP
calculates the required speed for each clock cycle
First-In-First-Out (FIFO)
S-Curve passes values to the FIFO queue to pass to the PWM in the Supervisor Block
Supervisor
Step Counter
Pulse Generator
Homing, Emergency
PL: ZYNQ's Artix-7 FPGA
OUTPUT TO MOTOR DRIVER

Figure 3: Implementation Block Diagram

## 4. Current project state and future plans

The current state of the project is a working 2nd degree S-Curve motion controller with plans to expand to higher orders as well as multiple axis of motion. For the power and resource usage, the motion controller only uses about 318 mW and utilizes less than 30% of the PYNQ board's resources leaving plenty of room for expansion.

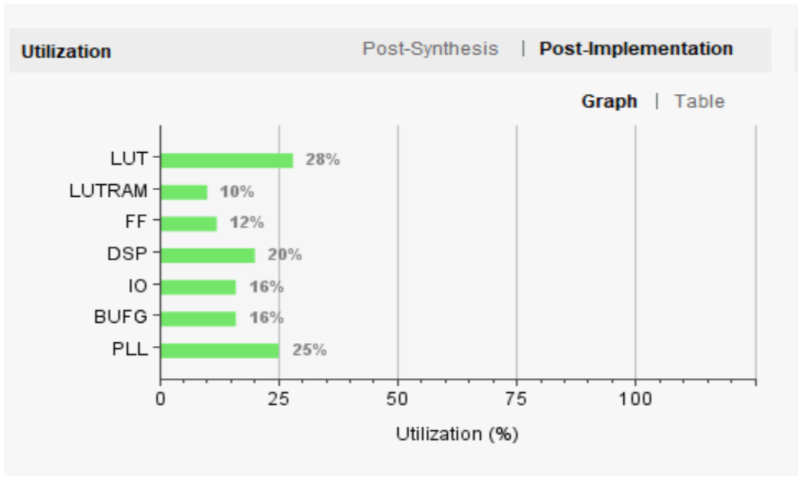

Figure 4: Resource Utilization